TEXAS INSTRUMENTS

# Digital Signal Processing Applications with the TMS320 Family

Application Book Volume 2

## Theory, Algorithms, and Implementations

Volume 2

Digital Signal Processing Applications with the TMS320 Family

1990

1990 Digital Signal Processor Products

# Digital Signal Processing Applications with the TMS320 Family

## Volume 2

*Edited by*
*Panos Papamichalis, Ph.D.*
*Digital Signal Processing*
*Semiconductor Group*
*Texas Instruments*

TEXAS INSTRUMENTS

## IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to or to discontinue any semiconductor product or service identified in this publication without notice. TI advises its customers to obtain the latest version of the relevant information to verify, before placing orders, that the information being relied upon is current.

TI warrants performance of its semiconductor products to current specifications in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Unless mandated by government requirements, specific testing of all parameters of each device is not necessarily performed.

TI assumes no liability for TI applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

## TRADEMARKS

*ADI* and *AutoCAD* are trademarks of Autodesk, Inc.
*Apollo* and *Domain* are trademarks of Apollo Computer, Inc.
*ATVista* is a trademark of Truevision, Inc.
*CodeView, MS-Windows, MS*, and *MS-DOS* are trademarks of Microsoft Corp.
*DEC, Digital DX, VAX, VMS,* and *Ultrix* are trademarks of Digital Equipment Corp.
*DGIS* is a trademark of Graphic Software Systems, Inc.
*EPIC*, *XDS, TIGA,* and *TIGA-340* are trademarks of Texas Instruments, Inc.
*GEM* is a trademark of Digital Research, Inc.
*GSS*CGI* is a trademark of Graphic Software Systems, Inc.
*HPGL* is a registered trademark of Hewlett-Packard Co.
*Macintosh* and *MPW* are trademarks of Apple Computer Corp.
*NEC* is a trademark of NEC Corp.
*PC-DOS, PGA,* and *Micro Channel* are trademarks of IBM Corp.
*PEPPER* is a registered trademark of Number Nine Computer Corp.
*PM* is a trademark of Microsoft Corp.
*PostScript* is a trademark of Adobe Systems, Inc.
*RTF* is a trademark of Microsoft Corp.
*Sony* is a trademark of Sony Corp.
*Sun 3*, *Sun Workstation*, *SunView, SunWindows,* and *SPARC* are trademarks of Sun Microsystems, Inc.
*UNIX* is a registered trademark of AT&T Bell Laboratories.

# CONTENTS

## PART VI. TOOLS

# Foreword

Much has happened in the TMS320 Family since Volume 1 of *Digital Signal Processing Applications with the TMS320 Family* was published, and Volumes 2 and 3 are a timely update to the family history.

The DSP microcomputers keep changing the perspective of the systems designers by offering more computational power and better interfacing capabilities. The steps of change are coming more quickly, and the potential impact is greater and greater. Because things change so rapidly in this area, there is a pressing need for ways to quickly learn how to utilize the new technology. These new volumes respond to that need.

As with Volume 1, the purpose of these books is to teach us about the issues and techniques that are important in implementing digital signal processing systems using microprocessors in the TMS320 Family. Volume 2 highlights the TMS320C25; and Volume 3, the TMS320C30 chip. A large part of the books is devoted to such matters as characteristics of the TMS320C25 and TMS320C30 chips, useful program code for implementing special DSP functions, and details on interfacing the new chips to external devices. The remainder of the books illustrates how these chips can be used in communications, control, and computer graphics applications.

What these two volumes make clear is how remarkably fast the field of DSP microcomputing is evolving. IC technologists and designers are simply packing more and more of the right kind of computing power into affordable microprocessor chips. The high-speed floating-point computing power and huge address spaces of chips like the TMS320C30 open the door to a whole new class of applications that were difficult or impractical with earlier generations of fixed-point DSP chips. The signal processing theorists and system designers are clearly being challenged to match the creativity of the chip designers.

The present books differ from Volume 1 in the inclusion of a small section on tools. This is a hopeful sign, because it is progress in this area that is likely to have the greatest impact on speeding the widespread application of DSP microprocessors. While useful design tools are beginning to emerge, much more can be done to help system designers manage the complexity of sophisticated DSP systems, which often involve a unique combination of theory, numerical and symbolic processing algorithms, real-time programming, and multiprocessing. No doubt future volumes of *Digital Signal Processing Applications with the TMS320 Family* will have more to say about this important topic. Until then, Volumes 2 and 3 have much useful information to help system designers keep up with the TMS320 Family.

Ronald W. Schafer
Atlanta, Georgia
November 14, 1989

# Preface

With the advancement of DSP devices, the application of Digital Signal Processing has become more widespread. Areas that were considered outside the domain of DSP devices because of cost, processing power, or peripheral capabilities (such as graphics, control, and consumer products) have seen applications using digital signal processors. On the other hand, the diverse needs of the designer have been addressed in the architectures and the performance of the newer devices.

Volume 2 of *Digital Signal Processing Applications with the TMS320 Family* contains applications on the first and second generations of the TMS320 Family (fixed-point devices). It is a continuation of Volume 1 in the sense that it addresses the same needs of the designer. The designer still has the task of selecting the DSP device with the appropriate cost, performance, and support, developing the DSP algorithm that will solve his problem, and implementing the algorithm on the processor. This volume tries to help the designer by bringing him up to date in the applications of newer processors or in different applications of earlier processors.

The objectives remain the same as in Volume 1. First, the application reports can be used as examples of device use. They can also serve as tutorials in programming the devices. Of course, the same purpose is served on a more elementary basis by the software and hardware applications sections of the corresponding user's guides. Second, since the source code of each application is provided with the report, the designer can take it intact (or extract a portion of it) and place it in his application.

It is assumed that the reader has exposure to the TMS320 devices or, at least, has the necessary manuals (such as the appropriate TMS320 user's guides) that will help him understand the explanations in the reports. The reports themselves include as references the necessary background material. Additionally, the Introduction gives a brief overview of the available devices at the time of the writing, and points to sources of more information.

The reports are grouped by application area. The term *report* is used here in a broad sense, since some articles from technical publications are also included. The authors of the reports are either the digital signal processing engineering staff of the Texas Instruments Semiconductor Group (including both field and factory personnel, and summer students) or third parties.

The source code associated with the reports is also available in electronic form, and the reader can download it from the TI DSP Electronic Bulletin Board (telephone (713) 274–2323). If more information is needed, the DSP Hotline can be called at (713) 274–2320.

The editor wishes to thank all the authors and the reviewers for their contribution to this volume of application reports.

Panos E. Papamichalis, Ph.D.
Senior Member of Technical Staff

# Part I. Introduction

1. The TMS320 Family and Book Overview
2. The TMS320 Family of Digital Signal Processors
   (Kun-Shan Lin, Gene A. Frantz, and Ray Simar, Jr., reprinted from *PROCEEDINGS OF THE IEEE*, Vol. 75, No. 9, September 1987)
3. The Texas Instruments TMS320C25 Digital Signal Microcomputer
   (Gene A. Frantz, Kun-Shan Lin, Jay B. Reimer, and Jon Bradley, reprinted from *IEEE Micro Magazine*, Vol. 6, No. 6, December 1986)

# TMS320 Family and Book Overview

Digital signal processors have found applications in areas where they were not even considered a few years earlier. The two major reasons for such proliferation are an increase in processor performance and a reduction in cost. Volume 2 of *Digital Signal Processing Applications with the TMS320 Family* presents a set of application reports on the first- and second-generation TMS320 devices.

## Organization of the Book

The application reports in this book are grouped by subject area:

- Introduction
- DSP Interface Techniques
- Data Communications
- Telecommunications
- Control
- Tools
- Bibliography

The **Introduction** contains this overview and two review articles. The first article gives a general description of the TMS320 family and is reprinted from a special issue of the *IEEE Proceedings*, while the second article discusses the TMS320C25 device and is reprinted from the *IEEE Micro Magazine*. The overview points out how the TMS320 family has grown since the two articles were published and also introduces newer devices.

The section on **DSP Interface Techniques** contains articles on interfacing first- and second-generation devices with external hardware, such as memories, A/D and D/A converters, or microcontroller devices like the TMS370 series. Other articles cover the implementation of a UART on the TMS320C25 and the power dissipation of the TMS320C25.

The three articles in the **Data Communications** section deal with different aspects of modem implementations. A V.22 design is presented in the first article, a 300-bps FSK modem in the second, and an Automatic Gain Control (AGC) in the third. In all cases, first-generation devices are considered.

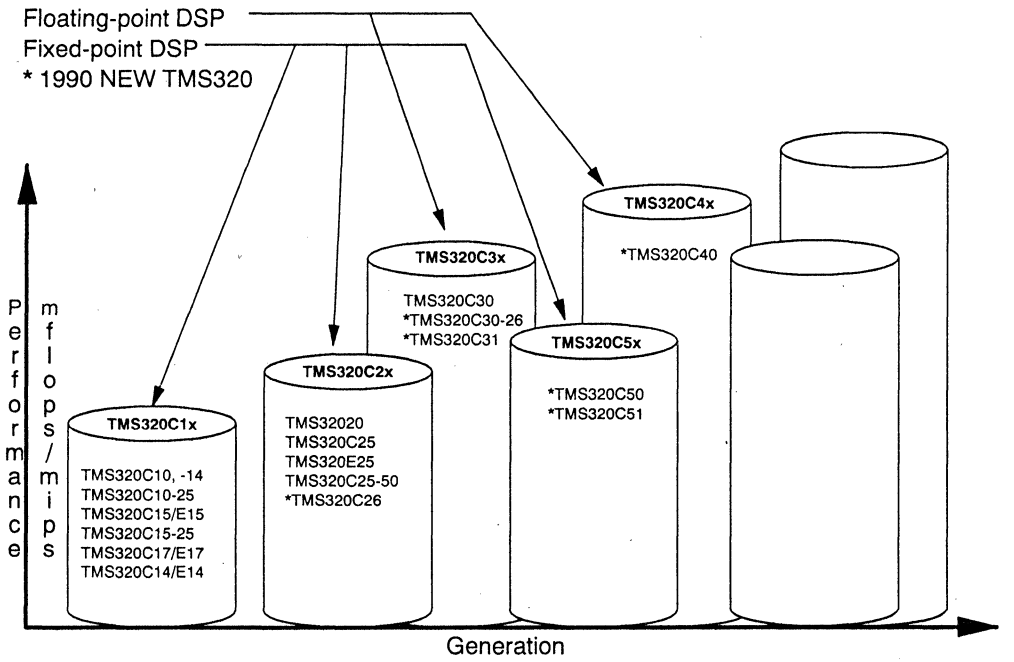
The following three sections contain one article each. In the **Telecommunications** section, a generalized tone decoding and DTMF detection method is presented. The **Control** section article gives insight into the relatively new application of digital signal processors in digital control. In the **Tools** section, the article describes ways to debug the algorithms with the aid of spreadsheets and other packages.

The **Bibliography** section contains a list of articles mentioning DSP implementations using TMS320 devices. The different titles are listed chronologically and are grouped by subject. The list is not exhaustive, but it gives enough pointers for pursuing practical implementations in representative application areas.

## The TMS320 Family of Processors

The TMS320 Family of digital signal processors started with the TMS32010 in 1982, but it has been expanded to encompass five generations (at the time of this writing) with devices in each generation. Figure 1 shows this progression through the generations. The TMS320 devices can be grouped in two broad categories: fixed-point and floating-point devices. As implied by Figure 1, the first, second, and fifth generations are the fixed-point devices, while the third and the fourth generations (the last one under development) support floating-point arithmetic.

**Figure 1. TMS320 Family Roadmap**



The following article, "The TMS320 Family of Digital Signal Processors," by Lin, et. al., is reprinted from the proceedings of the IEEE and gives an overview of the TMS320 family. Since additional devices have been developed from the time the article was written, this section highlights these newer devices. Table 1 shows a comprehensive list of the currently available TMS320 devices and their salient characteristics.

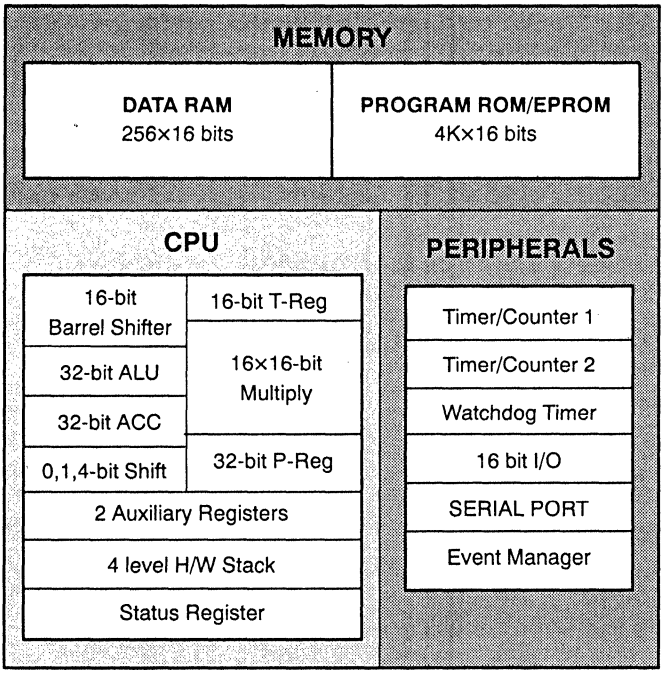
## Table 1. TMS320 Family Overview

| Gen | Device | Data Type | Cycle Time (ns) | Memory | | | | I/O | | | On-Chip Timers | Package |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | RAM | On-Chip ROM | EPROM | Off-Chip | Parallel | Serial | DMA | | |
| 1st | TMS320C10 ¶ | Integer | 200 | 144 | 1.5K | | 4K | 8x16 | | | | DIP/PLCC |
| | TMS320C10-25 | Integer | 160 | 144 | 1.5K | | 4K | 8x16 | | | | DIP/PLCC |
| | TMS320C10-14 | Integer | 280 | 144 | 1.5K | | 4K | 8x16 | | | | DIP/PLCC |
| | TMS320E14 | Integer | 160 | 256 | | 4K | 4K | 7x16 | 1 | | 4 | CERQUAD |
| | TMS320C15 ¶ | Integer | 200 | 256 | 4K | | 4K | 8x16 | | | | DIP/PLCC |
| | TMS320C15-25 ¶ | Integer | 160 | 256 | 4K | | 4K | 8x16 | | | | DIP/PLCC |
| | TMS320E15 ¶ | Integer | 200 | 256 | | 4K | 4K | 8x16 | | | | DIP/CERQUAD |
| | TMS320E15-25 | Integer | 160 | 256 | | 4K | 4K | 8x16 | | | | DIP/CERQUAD |
| | TMS320C17 | Integer | 200 | 256 | 4K | | 4K | 6x16 | 2 | | 1 | DIP/PLCC |
| | TMS320E17 | Integer | 200 | 256 | | 4K | 4K | 6x16 | 2 | | 1 | DIP/CERQUAD |
| 2nd | TMS32020 ¶ | Integer | 200 | 544 | | | 128K | 16x16 | 1 | † | 1 | PGA |
| | TMS320C25 ¶ | Integer | 100 | 544 | 4K | | 128K | 16x16 | 1 | † | 1 | PGA/PLCC |
| | TMS320C25-50 ¶ | Integer | 80 | 544 | 4K | | 128K | 16x16 | 1 | † | 1 | PGA/PLCC |
| | TMS320E25 ¶ | Integer | 100 | 544 | | 4K | 128K | 16x16 | 1 | † | 1 | CERQUAD |
| | TMS320C26 | Integer | 100 | 1.5K | 256 | | 128K | | 1 | † | 1 | PLCC |
| 3rd | TMS320C30 ¶ | Float Pt | 60 | 2K | 4K | | 16M | 16Mx32 | 2 | ‡ | 2 | PGA |
| 5th | TMS320C50 ¶ | Integer | 50 | 8.5K | 2K | | 128K | 16x16 | 1 | † | 1 | CLCC |

† External DMA
‡ External/Internal DMA
¶ For information on military versions of these devices, contact your local TI sales office.

The additions to the first generation are the TMS320C14 and the TMS320E14; the latter is identical with the former, except that the latter's on-chip program memory is EPROM. The TMS320C14/E14 devices have features that make them suitable for control applications. Figure 2 shows the components of these devices. The memory and the CPU are identical to those of the TMS320C15/E15, while the peripherals reflect the orientation of the devices toward control.

**Figure 2. TMS320C14/E14 Key Features**




Some of the key features of the TMS320C14/E14 are:

- 160-ns instruction cycle time
- Object-code-compatible with the TMS320C15
- Four 16-bit timers
  - Two general-purpose timers
  - One watchdog timer
  - One baud-rate generator
- 16 individual bit-selectable I/O pins
- Serial port/USART with codec-compatible mode
- Event manager with 6-channel PWM D/A
- CMOS technology, 68-pin CERQUAD

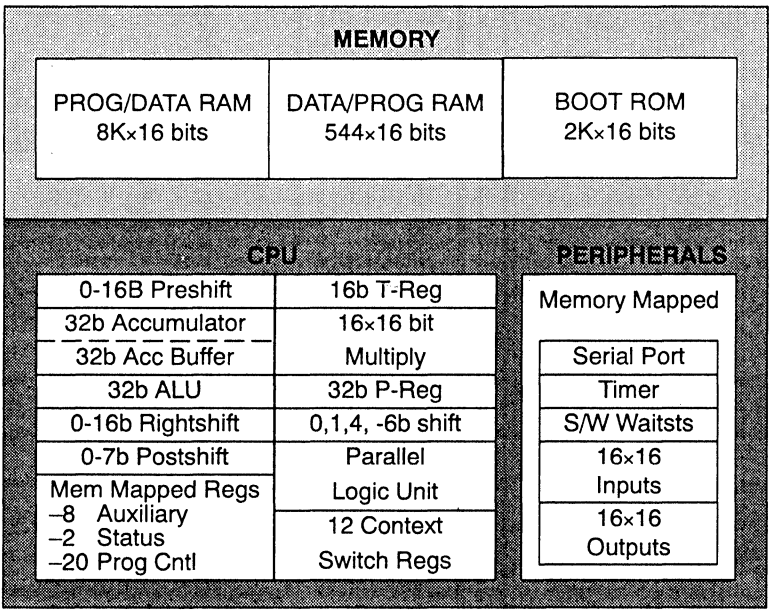
The additions to the second generation are the TMS320E25, the TMS320C25-50, and the TMS320C26. The TMS320E25 is identical to the TMS320C25, except that the 4K-word on-chip

program memory is EPROM. Since increased speed is very important for the real-time implementation of certain applications, the TMS320C25-50 was designed as a faster version of the TMS320C25 and has a clock frequency of 50 MHz instead of 40 MHz.

The TMS320C26 is a modification of the TMS320C25 in which the program ROM has been exchanged for RAM. The memory space of the TMS320C26 has 1.5K words of on-chip RAM and 256 words of on-chip ROM, making it ideal for applications requiring larger RAM but minimal external memory.

A new generation of higher-performance fixed-point processors has been introduced in the TMS320 Family: the TMS320C5X devices. This generation shares many features with the first and the second generations, but it also encompasses significant new features. Figure 3 shows the basic components of the first device in the fifth generation, the TMS320C50.

**Figure 3. TMS320C50 Key Features**




Some of the important features of the TMS320C50 are listed below:

- Source code is upward compatible with the TMS320C1x/C2x devices
- 50/35-ns instruction cycle time
- 8K words of on-chip program/data RAM
- 2K words boot ROM
- 544 words of data/program RAM
- 128K words addressable total memory
- Enhanced general-purpose and DSP-specific instructions
- Static CMOS, 84-pin CERQUAD
- JTAG serial scan path

The software and hardware development tools available for the TMS320 family make the development of applications easy. Such tools include assemblers, linkers, simulators, and C compilers for software and evaluation modules, software development boards, and extended development systems for hardware. These tools are mentioned in the following paper by Lin, et. al. The interested reader can find much more information in additional literature that is published by Texas Instruments and mentioned in the next section. In particular, the *TMS320 Family Development Support Reference Guide* is an excellent source.

One important addition to the list of tools is the SPOX operating system, developed by Spectron Microsystems. SPOX permits you to write an application in a high-level language (C) and run it on actual DSP hardware. The operating-system of SPOX hides the details of the interface from you and lets you concentrate on your algorithm while running it at supercomputer speeds on the TMS320C30.

## References

Texas Instruments publishes an extensive bibliography to help designers use the TMS320 devices effectively. Besides user's guides for corresponding generations, there are manuals for the software and the hardware tools. The Development Support Reference Guide is particularly useful because it provides information not only on development tools offered by TI, but also on those produced by third parties. Here is a partial list of the literature available (the literature number is in parentheses):

- *TMS320 Family Development Support Reference Guide* (SPRU011A)
- *TMS320C1x User's Guide* (SPRU013A)
- *TMS320C2x User's Guide* (SPRU014)
- *TMS320C3x User's Guide* (SPRU031)
- *TMS320C1x/TMS320C2x Assembly Language Tools User's Guide* (SPRU018)
- *TMS320C30 Assembly Language Tools User's Guide* (SPRU035)
- *TMS320C25 C Compiler Reference Guide* (SPRU024)
- *TMS320C30 C Compiler Reference Guide* (SPRU034)
- *Digital Signal Processing Applications with the TMS320 Family, Volume 1* (SPRA012)
- *Digital Signal Processing Applications with the TMS320 Family, Volume 3* (SPRA017)

You can request this literature by calling the Customer Response Center at 1–800–232–3200, or the DSP Hotline at 1–713–274–2320.

### *Contents of Other Volumes of the Application Book*

**Volume 1**

Part I. Digital Signal Processing and the TMS320 Family

- Introduction
- The TMS320 Family

Part II. Fundamental Digital Signal Processing Operations

- Digital Signal Processing Routines

- Implementation of FIR/IIR Filters with the TMS32010/TMS32020
- Implementation of Fast Fourier Transform Algorithms with the TMS32020
- Companding Routines for the TMS32010/TMS32020
- Floating-Point Arithmetic with the TMS32010
- Floating-Point Arithmetic with the TMS32020
- Precision Digital Sine-Wave Generation with the TMS32010
- Matrix Multiplication with the TMS32010 and TMS32020

- DSP Interface Techniques
  - Interfacing to Asynchronous Inputs with the TMS32010
  - Interfacing External Memory to the TMS32010
  - Hardware Interfacing to the TMS32020
  - TMS32020 and MC68000 Interface

Part III. Digital Signal Processing Applications

- Telecommunications
  - Telecommunications Interfacing to the TMS32010
  - Digital Voice Echo Canceller with a TMS32020
  - Implementation of the Data Encryption Standard Using the TMS32010
  - 32K-bit/s ADPCM with the TMS32010
  - A Real-Time Speech Subband Coder Using the TMS32010
  - Add DTMF Generation and Decoding to DSP-μP Designs
- Computers and Peripherals
- Speech Coding/Recognition
  - A single-Processor LPC Vocoder
  - The Design of an Adaptive Predictive Coder Using a Single-Chip
  - Digital Signal Processor
  - Firmware-Programmable C Aids Speech Recognition
- Image/Graphics
  - A Graphics Implementation Using the TMS32020 and TMS34061
- Digital Control
  - Control System Compensation and Implementation with the TMS32010

**Volume 3**

Part I. Introduction

- Book Overview
- The TMS320 Family of DSP
- The TMS320C30 Floating-Point DSP

Part II. Digital Signal Processing Routines

- Implementation of FFT, DCT, and other Transforms on the TMS320C30
- Doublelength Floating-Point Arithmetic on the TMS320C30
- An 8 x 8 Discrete Cosine Transform Implementation on the TMS320C25 and the TMS320C30
- Implementation of Adaptive Filters with the TMS320C25 and TMS320C30
- A Collection of Functions for the TMS320C30

Part III. DSP Interface Techniques

- Hardware Interfacing to the TMS320C30
- TMS320C30 – IEEE Floating-Point Format Converter

Part IV. Telecommunications

- Implementation of a CELP Speech Coder for the TMS320C30 Using SPOX

Part V. Computers

- A Digital Signal Processor Based 3-D Graphics System

Part VI. Tools

- TMS320C30 Applications Board Functional Description

# The TMS320 Family of Digital Signal Processors

**Kun-Shan Lin**
**Gene A. Frantz**
**Ray Simar, Jr.**

**Digital Signal Processor Products — Semiconductor Group**
**Texas Instruments**

**Reprinted from**
*PROCEEDINGS OF THE IEEE*
**Vol. 75, No. 9, September 1987**

# The TMS320 Family of Digital Signal Processors

KUN-SHAN LIN, MEMBER, IEEE, GENE A. FRANTZ, SENIOR MEMBER, IEEE, AND RAY SIMAR, JR.

*This paper begins with a discussion of the characteristics of digital signal processing, which are the driving force behind the design of digital signal processors. The remainder of the paper describes the three generations of the TMS320 family of digital signal processors available from Texas Instruments. The evolution in architectural design of these processors and key features of each generation of processors are discussed. More detailed information is provided for the TMS320C25 and TMS320C30, the newest members in the family. The benefits and cost-performance tradeoffs of these processors become obvious when applied to digital signal processing applications, such as telecommunications, data communications, graphics/image processing, etc.*

## Digital Signal Processing Characteristics

Digital signal processing (DSP) encompasses a broad spectrum of applications. Some application examples include digital filtering, speech vocoding, image processing, fast Fourier transforms, and digital audio [1]–[10]. These applications and those considered digital signal processing have several characteristics in common:

- mathematically intensive algorithms,
- real-time operation,
- sampled data implementation,
- system flexibility.

To illustrate these characteristics in this section, we will use the digital filter as an example. Specifically, we will use the Finite Impulse Response (FIR) filter which in the time domain takes the general form of

$$y(n) = \sum_{i=1}^{N} a(i) * x(n - i) \qquad (1)$$

where $y(n)$ is the output sample at time $n$, $a(i)$ is the $i$th coefficient or weighting factor, and $x(n - i)$ is the $(n - i)$th input sample.

With this example in mind, we can discuss the various characteristics of digital signal processing: mathematically intensive algorithms, real-time processing, sampled data implementation, and system flexibility. First, let us look at the concept of mathematically intensive algorithms.

Manuscript received October 6, 1986; revised March 27, 1987.
The authors are with the Semiconductor Group, Texas Instruments Inc., Houston, TX 77521-1445, USA.
IEEE Log Number 8716214.

### *Mathematically Intensive Algorithms*

From (1), we can see that to generate every $y(n)$, we have to compute $N$ multiplications and additions or sums of products. This computation makes it mathematically intensive, especially when $N$ is large.

At this point it is worthwhile to give the FIR filter some physical significance. An FIR filter is a common technique used to eliminate the erratic nature of stock market prices. When the day-to-day closing prices are plotted, it is sometimes difficult to obtain the desired information, such as the trend of the stock, because of the large variations. A simple way of smoothing the data is to calculate the average closing values of the previous five days. For the new average value each day, the oldest value is dropped and the newest value added. Each daily average value (average $(n)$) would be the sum of the weighted value of the latest five days, where the weighting factors ($a(i)$'s) are 1/5. In equation form, the average is determined by

$$\text{average } (n) = \frac{1}{5} * d(n - 1) + \frac{1}{5} * d(n - 2) + \frac{1}{5} * d(n - 3) + \frac{1}{5} * d(n - 4) + \frac{1}{5} * d(n - 5) \qquad (2)$$

where $d(n - i)$ is the daily stock closing price for the $(n - i)$th day. Equation (2) assumes the same form as (1). This is also the general form of the convolution of two sequences of numbers, $a(i)$ and $x(i)$ [5], [6]. Both FIR filtering and convolution are fundamental to digital signal processing.

### *Real-Time Processing*

In addition to being mathematically intensive, DSP algorithms must be performed in real time. Real time can be defined as a process that is accomplished by the DSP without creating a delay noticeable to the user. In the stock market example, as long as the new average value can be computed prior to the next day when it is needed, it is considered to be completed in real time. In digital signal processing applications, processes happen faster than on a daily basis. In the FIR filter example in (1), the sum of products must

be computed usually within hundreds of microseconds before the next sample comes into the system. A second example is in a speech recognition system where a noticeable delay between a word being spoken and being recognized would be unacceptable and not considered real-time. Another example is in image processing, where it is considered real-time if the processor finishes the processing within the frame update period. If the pixel information cannot be updated within the frame update period, problems such as flicker, smearing, or missing information will occur.

*Sampled Data Implementation*

The application must be capable of being handled as a sampled data system in order to be processed by digital processors, such as digital signal processors. The stock market is an example of a sampled data system. That is, a specific value (closing value) is assigned to each sample period or day. Other periods may be chosen such as hourly prices or weekly prices. In an FIR filter as shown in (1), the output $y(n)$ is calculated to be the weighted sum of the previous $N$ inputs. In other words, the input signal is sampled at periodic intervals (1 over the sample rate), multiplied by weighting factor $a(i)$, and then added together to give the output result of $y(n)$. Examples of sample rates for some typical sampled data applications [2], [4] are shown in Table 1.

**Table 1** Sample Rates versus Applications

| Application | Nominal Sample Rate |
|---|---|
| Control | 1 kHz |
| Telecommunications | 8 kHz |
| Speech processing | 8–10 kHz |
| Audio processing | 40–48 kHz |
| Video frame rate | 30 Hz |
| Video pixel rate | 14 MHz |

In a typical DSP application, the processor must be able to effectively handle sampled data in large quantity and also perform arithmetic computations in real time.

*System Flexibility*

The design of the digital signal processing system must be flexible enough to allow improvements in the state of the art. We may find out after several weeks of using the average stock price as a means of measuring a particular stock's value that a different method of obtaining the daily information is more suited to our needs, e.g., using different daily weightings, a different number of periods over which to average, or a different procedure for calculating the result. Enough flexibility in the system must be available to allow for these variations. In many of the DSP applications, techniques are still in the developmental phase, and therefore the algorithms tend to change over time. As an example, speech recognition is presently an inexact technique requiring continual algorithmic modification. From this example we can see the need for system flexibility so that the DSP algorithm can be updated. A programmable DSP system can provide this flexibility to the user.

## Historical DSP Solutions

Over the past several decades, digital signal processing machines have taken on several evolutions in order to incorporate these characteristics. Large mainframe computers were initially used to process signals in the digital domain. Typically, because of state-of-the-art limitations, this was done in nonreal time. As the state of the art advanced, array processors were added to the processing task. Because of their flexibility and speed, array processors have become the accepted solution for the research laboratory, and have been extended to end-applications in many instances. However, integrated circuit technology has matured, thus allowing for the design of faster microprocessors and microcomputers. As a result, many digital signal processing applications have migrated from the array processor to microprocessor subsystems (i.e., bit-slice machines) to single-chip integrated circuit solutions. This migration has brought the cost of the DSP solution down to a point that allows pervasive use of the technology. The increased performance of these highly integrated circuits has also expanded DSP applications from traditional telecommunications to graphics/image processing, then to consumer audio processing.

A recent development in DSP technology is the single-chip digital signal processor, such as the TMS320 family of processors. These processors give the designer a DSP solution with its performance attainable only by the array processors a few years ago. Fig. 1 shows the TMS320 family in graphical form with the $y$-axis indicating the hypothetical performance and the $x$-axis being the evolution of the semiconductor processing technology. The first member of the family, the TMS32010, was disclosed to the market in 1982 [11], [12]. It gave the system designer the first microcomputer capable of performing five million DSP operations per second (5 MIPS), including the add and multiply functions [13] required in (1). Today there are a dozen spinoffs from the TMS32010 in the first generation of the TMS320 family. Some of these devices are the TMS320C10, TMS320C15, and TMS320C17 [14]. The second generation of devices include the TMS32020 [15] and TMS320C25 [16]. The TMS320C25 can perform 10 MIPS [16]. In addition, expanded memory space, combined single-cycle multiply/accumulate operation, multiprocessing capabilities, and expanded I/O functions have given the TMS320C25 a 2 to 4 times performance improvement over its predecessors. The third generation of the TMS320 family of processors, the TMS320C30 [26], [27], has a computational rate of 33 million DSP floating-point operations per second (33 MFLOPS). Its performance (speed, throughput, and precision) has far exceeded the digital signal processors available today and has reached the level of a supercomputer.

It we look closely at the TMS320 family as shown in Fig. 1, we can see that devices in the same generation, such as the TMS320C10, TMS320C15, and TMS320C17, are assembly object-code compatible. Devices across generations, such as the TMS320C10 and TMS320C25, are assembly source-code compatible. Software investment on DSP algorithms therefore can be maintained during the system upgrade. Another point is that since the introduction of the TMS32010, semiconductor processing technology has emerged from 3-$\mu$m NMOS to 2-$\mu$m CMOS to 1-$\mu$m CMOS.

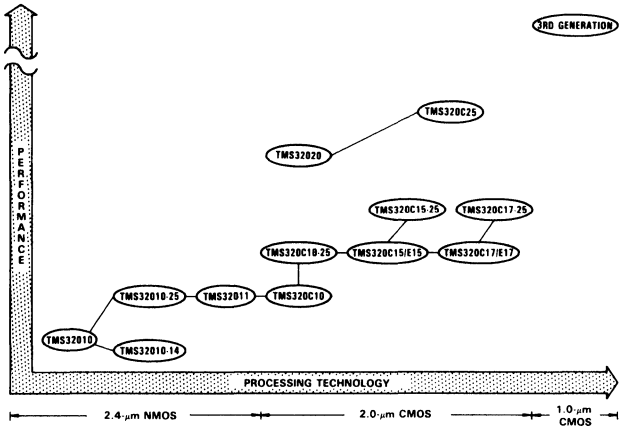



**Fig. 1.** The TMS320 family of digital signal processors.

The TMS320 generations of processors have also taken the same evolution in processing technology. Low power consumption, high performance, and high-density circuit integration are some of the direct benefits of this semiconductor processing evolution.

From Fig. 1, it can be observed that various DSP building blocks, such as the CPU, RAM, ROM, I/O configurations, and processor speeds, have been designed as individual modules and can be rearranged or combined with other standard cells to meet the needs of specific applications. Each of the three generations (and future generations) will evolve in the same manner. As applications become more sophisticated, semicustom solutions based on the core CPU will become the solution of choice. An example of this approach is the TMS320C17/E17, which consists of the TMS320C10 core CPU, expanded 4K-word program ROM (TMS320C17) or EPROM (TMS320E17), enlarged data RAM of 256 words, dual serial ports, companding hardware, and a coprocessor interface. Furthermore, as integrated circuit layout rules move into smaller geometry (now at 2 μm, rapidly going to 1 μm), not only will the TMS320 devices become smaller in size, but also multiple CPUs will be incorporated on the same device along with application-specific I/O to achieve low-cost integrated system solutions.

### Basic TMS320 Architecture

As noted previously, the underlying assumption regarding a digital signal processor is fast arithmetic operations and high throughput to handle mathematically intensive algorithms in real time. In the TMS320 family [11]–[17], [26], [27], this is accomplished by using the following basic concepts:

- Harvard architecture,
- extensive pipelining,
- dedicated hardware multiplier,
- special DSP instructions,
- fast instruction cycle.

These concepts were designed into the TMS320 digital signal processors to handle the vast amount of data characteristic of DSP operations, and to allow most DSP operations to be executed in a single-cycle instruction. Furthermore, the TMS320 processors are programmable devices, providing the flexibility and ease of use of general-purpose microprocessors. The following paragraphs discuss how each of the above concepts is used in the TMS320 family of devices to make them useful in digital signal processing applications.

#### *Harvard Architecture*

The TMS320 utilizes a modified Harvard architecture for speed and flexibility. In a strict Harvard architecture [18], [19], the program and data memories lie in two separate spaces, permitting a full overlap of instruction fetch and execution. The TMS320 family's modification of the Harvard architecture further allows transfer between program and data spaces, thereby increasing the flexibility of the device. This architectural modification eliminates the need for a separate coefficient ROM and also maximizes the processing power by maintaining two separate bus structures (program and data) for full-speed execution.

#### *Extensive Pipelining*

In conjunction with the Harvard architecture, pipelining is used extensively to reduce the instruction cycle time to its absolute minimum, and to increase the throughput of the processor. The pipeline can be anywhere from two to four levels deep, depending on which processor in the family is used. The TMS320 family architecture uses a two-level pipeline for its first generation, a three-level pipeline for its second generation, and a four-level pipeline for its third generation of processors. This means that the device is processing from two to four instructions in parallel, and each instruction is at a different stage in its execution. Fig. 2 shows an example of a three-level pipeline operation.

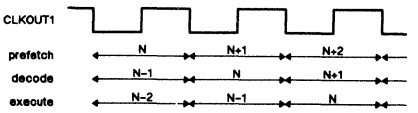



**Fig. 2.** Three-level pipeline operation.

In pipeline operation, the prefetch, decode, and execute operations can be handled independently, thus allowing the execution of instructions to overlap. During any instruction cycle, three different instructions are active, each at a different stage of completion. For example, as the *N*th instruction is being prefetched, the previous ($N - 1$)th instruction is being decoded, and the previous ($N - 2$)th instruction is being executed. In general, the pipeline is transparent to the user.

*Dedicated Hardware Multiplier*

As we saw in the general form of an FIR filter, multiplication is an important part of digital signal processing. For each filter tap (denoted by *i*), a multiplication and an addition must take place. The faster a multiplication can be performed, the higher the performance of the digital signal processor. In general-purpose microprocessors, the multiplication instruction is constructed by a series of additions, therefore taking many instruction cycles. In comparison, the characteristic of every DSP device is a dedicated multiplier. In the TMS320 family, multiplication is a single-cycle instruction as a result of the dedicated hardware multiplier. If we look at the arithmetic for each tap of the FIR filter to be performed by the TMS32010, we see that each tap of the filter requires a multiplication (MPY) instruction.

```
LT     ;LOAD MULTIPLICAND INTO T REGISTER
DMOV   ;MOVE DATA IN MEMORY TO DO DELAY
MPY    ;MULTIPLY
APAC   ;ADD MULTIPLICATION RESULT TO ACC
```

The other three instructions are used to load the multiplier circuit with the multiplicand (LT), move the data through the filter tap (DMOV), and add the result of the multiplication (stored in the product register) to the accumulator (APAC). Specifically, the multiply instruction (MPY) loads the multiplier into the dedicated multiplier and performs the multiplication, placing the result in a product register. Therefore, if a 256-tap FIR filter is used, these four instructions are repeated 256 times. At each sample period, 256 multiplications must be performed. In a typical general-purpose microprocessor, this requires each tap to be 30 to 40 instruction cycles long, whereas in the TMS320C10, it is only four instruction cycles. We will see in the next section how special DSP instructions reduce the time required for each FIR tap even further.

*Special DSP Instructions*

Another characteristic of DSP devices is the use of special instructions. We were introduced to one of them in the previous example, the DMOV (data move) instruction. In digital signal processing, the delay operator ($z^{-1}$) is very important. Recalling the stock market example, during each new sample period (i.e., each new day), the oldest piece of data (the closing price five days ago) was dropped and a new one (today's closing price) was added. Or, each piece of the old data is delayed or moved one sample period to make room for the incoming most current sample. This delay is the function of the DMOV instruction. Another special instruction in the TMS32010 is the LTD instruction. It executes the LT, DMOV, and APAC instructions in a single cycle. The LTD and MPY instruction then reduce the number of instruction cycles per FIR filter tap from four to two. In the second-generation TMS320, such as the TMS320C25, two more special instructions have been included (the RPT and MACD instructions) to reduce the number of cycles per tap to one, as shown in the following:

```
RPTK  255  ;REPEAT THE NEXT INSTRUCTION 256 TIMES
           (N + 1)
MACD       ;LT, DMOV, MPY, AND APAC
```

*Fast Instruction Cycle*

The real-time processing capability is further enhanced by the raw speed of the processor in executing instructions. The characteristics which we have discussed, combined with optimization of the integrated circuit design for speed, give the DSP devices instruction cycle times less than 200 ns. The specific instruction cycle times for the TMS320 family are given in Table 2. These fast cycle times have made the TMS320 family of processors highly suited for many real-time DSP applications. Table 1 showed the sample rates for some typical DSP applications. This table can be combined with the cycle times indicated in Table 2 to show how many instruction cycles per sample can be achieved by the various generations of the TMS320 for real-time applications (see Fig. 3).

**Table 2** TMS320 Cycle Times

| Device | Cycle Time (ns) |
|---|---|
| TMS320C10* | 160–200 |
| TMS32020 | 160–200 |
| TMS320C25 | 100–125 |
| TMS320C30 | 60–75 |

*The same cycle time applies to all of the first-generation processors.

As we can see from Fig. 3, many instruction cycles are available to process the signal or to generate commands for real-time control applications. Therefore, for simple control applications, the general-purpose microprocessors or controllers would be adequate. However, for more mathematically intensive control applications, such as robotics and adaptive control, digital signal processors are much better suited [24]. The number of available instruction cycles is reduced as we increase the sample rate from 8 kHz for typical telecommunication applications to 40–48 kHz for audio processing. Since most of these real-time applications require only a few hundreds of instructions per sample (such as ADPCM [4], and echo cancelation [4]), this is within the reach of the TMS320. For higher sample rate applications, such as video/image processing, digital signal processors available today are not capable of handling the processing of the real-time video data. Therefore, for these

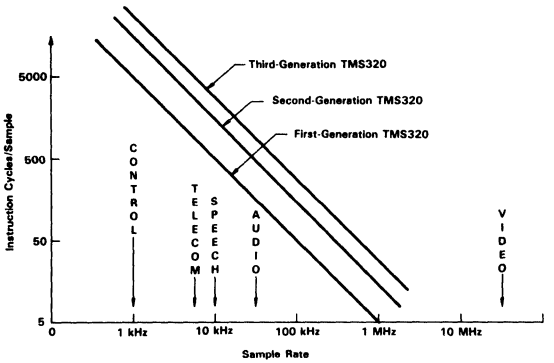



**Fig. 3.** Number of instruction cycles/sample versus sample rate for the TMS320 family.

types of applications, multiple digital signal processors and frame buffers are usually required. From Fig. 3, it can also be seen that for slower speed applications, such as control, the first-generation TMS320 provides better cost-performance tradeoffs than the other processors. For high sample rate applications, such as video/image processing, the second and third generations of the TMS320 with their multiprocessing capabilities and high throughput are better suited.

Now that we have discussed the basic characteristics of digital signal processors, we can concentrate on specific details of each of the three generations of the TMS320 family devices.

### The First Generation of the TMS320 Family

The first generation of the TMS320 family includes the TMS32010 [13], and TMS32011 [17], which are processed in 2.4-$\mu$m NMOS technology, and the TMS320C10 [13], TMS320C15/E15 [14], and TMS320C17/E17 [14], processed in 1.8-$\mu$m CMOS technology. Some of the key features of these devices are [14] as follows:

- Instruction cycle timing:
  - -160 ns
  - -200 ns
  - -280 ns.
- On-chip data RAM:
  - -144 words
  - -256 words (TMS320C15/E15, TMS320C17/E17).
- On-chip program ROM:
  - -1.5K words
  - -4K words (TMS320C15, TMS320C17).
- 4K words of on-chip program EPROM (TMS320E15, TMS320E17).
- External memory expansion up to 4K words at full speed.
- 16 × 16-bit parallel multiplier with 32-bit result.
- Barrel shifter for shifting data memory words into the ALU.
- Parallel shifter.
- 4 × 12-bit stack that allows context switching.
- Two auxiliary registers for indirect addressing.
- Dual-channel serial port (TMS32011, TMS320C17, TMS320E17).
- On-chip companding hardware (TMS32011, TMS320C17, TMS320E17).
- Coprocessor interface (TMS320C17, TMS320E17).
- Device packaging
  - -40-pin DIP
  - -44-pin PLCC.

#### *TMS320C10*

The first generation of the TMS320 processors is based on the architecture of the TMS32010 and its CMOS replica, the TMS320C10. The TMS32010 was introduced in 1982 and was the first microcomputer capable of performing 5 MIPS. Since the TMS32010 has been covered extensively in the literature [4], [11]–[14], we will only provide a cursory review here. A functional block diagram of the TMS320C10 is shown in Fig. 4.

As shown in Fig. 4, the TMS320C10 utilizes the modified Harvard architecture in which program memory and data memory lie in two separate spaces. Program memory can reside both on-chip (1.5K words) or off-chip (4K words). Data memory is the 144 × 16-bit on-chip data RAM. There are four basic arithmetic elements: the ALU, the accumulator, the multiplier, and the shifters. All arithmetic operations are performed using two's-complement arithmetic.

*ALU:* The ALU is a general-purpose arithmetic logic unit that operates with a 32-bit data word. The unit can add, subtract, and perform logical operations.

*Accumulator:* The accumulator stores the output from the ALU and is also often an input to the ALU. It operates with a 32-bit word length. The accumulator is divided into a high-order word (bits 31 through 16) and a low-order word (bits 15 through 0). Instructions are provided for storing the high- and low-order accumulator words in data memory (SACH for store accumulator high and SACL for store accumulator low).

*Multiplier:* The 16 × 16-bit parallel multiplier consists of three units: the T register, the P register, and the multipler array. The T register is a 16-bit register that stores the multiplicand, while the P register is a 32-bit register that stores the product. In order to use the multiplier, the multiplicand

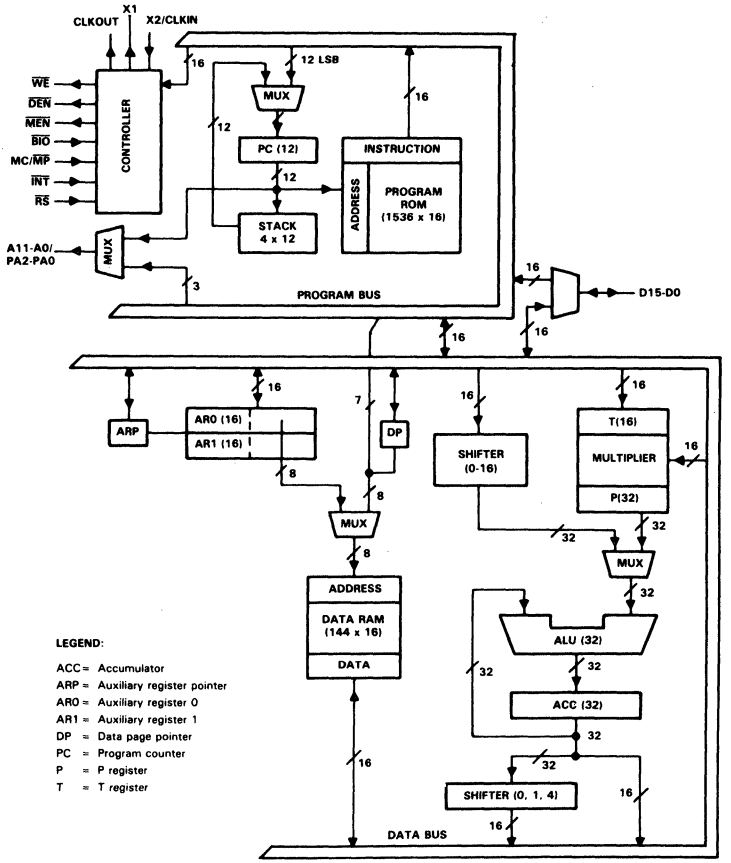



**Fig. 4.** TMS320C10 functional block diagram.

must first be loaded into the T register from the data RAM by using one of the following instructions: LT, LTA, or LTD. Then the MPY (multiply) or the MPYK (multiply immediate) instruction is executed. The multiply and accumulate operations can be accomplished in two instruction cycles with the LTA/LTD and MPY/MPYK instructions.

*Shifters:* Two shifters are available for manipulating data: a barrel shifter and a parallel shifter. The barrel shifter performs a left-shift of 0 to 16 bits on all data memory words that are to be loaded into, subtracted from, or added to the accumulator. The parallel shifter, activated by the SACH instruction, can execute a shift of 0, 1, or 4 bits to take care of the sign bits in two's-complement arithmetic calculations.

Based on the architecture of the TMS32010/C10, several spinoffs have been generated offering different processor speeds, expanded memory, and various I/O integration. Currently, the newest members in this generation are the TMS320C15/E15 and the TMS320C17/E17 [14].

*TMS320C15/E15*

The TMS320C15 and TMS320E15 are fully object-code and pin-for-pin compatible with the TMS32010 and offer expanded on-chip RAM of 256 words and on-chip program ROM (TMS320C15) or EPROM (TMS320E15) of 4K words. The TMS320C15 is available in either a 200-ns version or a 160-ns version (TMS320C15-25).

*TMS320C17/E17*

The TMS320C17/E17 is a dedicated microcomputer with 4K words of on-chip program ROM (TMS320C17) or EPROM (TMS320E17), a dual-channel serial port for full-duplex serial communication, on-chip companding hardware (u-law/A-law), a serial port timer for stand-alone serial communication, and a coprocessor interface for zero glue interface between the processor and any 4/8/16-bit microprocessor. The TMS320C17/E17 is also object-code compatible with the TMS32010 and can use the same development tools. The

**Table 3** TMS320 First-Generation Processors

| TMS320 Devices | Instruction Cycle Time (ns) | Process | On-Chip Prog ROM (words) | On-Chip Prog EPROM (words) | On-Chip Data RAM (words) | Off-Chip Prog (words) | Ref |
|---|---|---|---|---|---|---|---|
| TMS32010 | 200 | NMOS | 1.5K | | 144 | 4K | [13] |
| TMS32010-25 | 160 | NMOS | 1.5K | | 144 | 4K | [13] |
| TMS32010-14 | 280 | NMOS | 1.5K | | 144 | 4K | [13] |
| TMS32011 | 200 | NMOS | 1.5K | | 144 | | [17] |
| TMS320C10 | 200 | CMOS | 1.5K | | 144 | 4K | [13] |
| TMS320C10-25 | 160 | CMOS | 1.5K | | 144 | 4K | [13] |
| TMS320C15 | 200 | CMOS | 4.0K | | 256 | 4K | [13] |
| TMS320C15-25 | 160 | CMOS | 4.0K | | 256 | 4K | [14] |
| TMS320E15 | 200 | CMOS | | 4.0K | 256 | 4K | [14] |
| TMS320C17 | 200 | CMOS | 4.0K | | 256 | | [14] |
| TMS320C17-25 | 160 | CMOS | 4.0K | | 256 | | [14] |
| TMS320E17 | 200 | CMOS | | 4.0K | 256 | | [14] |

device is based on the TMS320C10 core CPU with added peripheral memory and I/O modules added on-chip. The TMS320C17/E17 can be regarded as a semicustom DSP solution suited for high-volume telecommunication and consumer applications.

Table 3 provides a feature comparison of all members of the first-generation TMS320 processors. References to more detailed information on these processors are also provided.

### The Second Generation of the TMS320 Family

The second-generation TMS320 digital signal processors includes two members, the TMS32020 [15] and the TMS320C25 [16]. The architecture of these devices has been evolved from the TMS32010, the first member of the TMS320 family. Key features of the second-generation TMS320 are as follows:

- Instruction cycle timing:
  - -100 ns (TMS320C25)
  - -200 ns (TMS32020).
- 4K words of on-chip masked ROM (TMS320C25).
- 544 words of on-chip data RAM.
- 128K words of total program data memory space.
- Eight auxiliary registers with a dedicated arithmetic unit.
- Eight-level hardware stack.
- Fully static double-buffered serial port.
- Wait states for communication to slower off-chip memories.
- Serial port for multiprocessing or interfacing to codecs.
- Concurrent DMA using an extended hold operation (TMS320C25).
- Bit-reversed addressing modes for fast Fourier transforms (TMS320C25).
- Extended-precision arithmetic and adaptive filtering support (TMS320C25).
- Full-speed operation of MAC/MACD instructions from external memory (TMS320C25).
- Accumulator carry bit and related instructions (TMS320C25).
- 1.8-$\mu$m CMOS technology (TMS320C25):
  - -68-pin grid array (PGA) package.
  - -68-pin lead chip carrier (PLCC) package.
- 2.4-$\mu$m NMOS technology (TMS32020):
  - -68-pin PGA package.

#### *TMS320C25 Architecture*

The TMS320C25 is the latest member in the second generation of TMS320 digital signal processors. It is a pin-compatible CMOS version of the TMS32020 microprocessor, but with an instruction cycle time twice as fast and the inclusion of additional hardware and software features. The instruction set is a superset of both the TMS32010 and TMS32020, maintaining source-code compatibility. In addition, it is completely object-code compatible with the TMS32020 so that TMS32020 programs run unmodified on the TMS320C25.

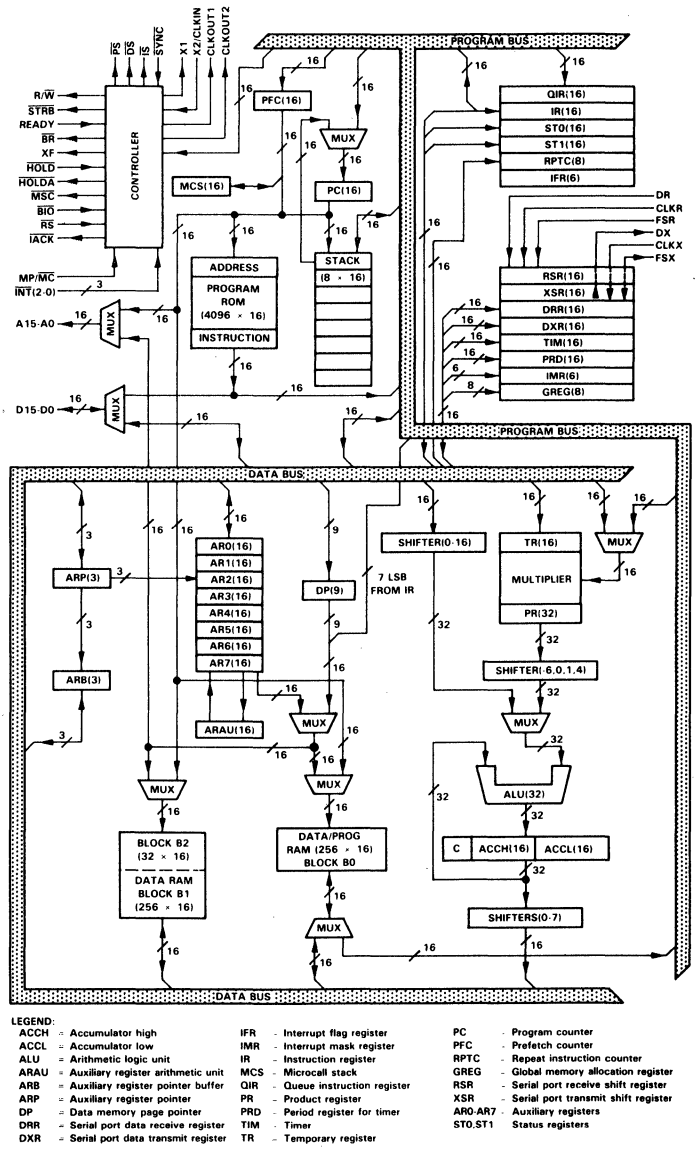
The 100-ns instruction cycle time provides a significant throughput advantage for many existing applications. Since most instructions are capable of executing in a single cycle, the processor is capable of executing ten million instructions per second (10 MIPS). Increased throughput on the TMS320C25 for many DSP applications is attained by means of single-cycle multiply/accumulate instructions with a data move option (MAC/MACD), eight auxiliary registers with a dedicated arithmetic unit, instruction set support for adaptive filtering and extended-precision arithmetic, bit-reversal addressing, and faster I/O necessary for data-intensive signal processing.

Instructions are included to provide data transfers between the two memory spaces. Externally, the program and data memory spaces are multiplexed over the same bus so as to maximize the address range for both spaces while minimizing the pin count of the device. Internally, the TMS320C25 architecture maximizes processing power by maintaining two separate bus structures, program and data, for full-speed execution.

Program execution in the device takes the form of a three-level instruction fetch-decode-execute pipeline (see Fig. 2). The pipeline is essentially invisible to the user, except in some cases where it must be broken (such as for branch instructions). In this case, the instruction timing takes into account the fact that the pipeline must be emptied and refilled. Two large on-chip data RAM blocks (a total of 544 words), one of which is configurable either as program or data memory, provide increased flexibility in system design. An off-chip 64K-word directly addressable data memory address space is included to facilitate implementations of DSP algorithms. The large on-chip 4K-word masked ROM can be used for cost-reduced systems, thus providing for a true single-chip DSP solution. The remainder of the 64K-word program memory space is located externally. Large

programs can execute at full speed from this memory space. Programs may also be downloaded from slow external memory to on-chip RAM for full-speed operation. The VLSI implementation of the TMS320C25 incorporates all of these features as well as many others such as a hardware timer, serial port, and block data transfer capabilities.

A functional block diagram of the TMS320C25, shown in Fig. 5, outlines the principal blocks and data paths within




LEGEND:

| | | | | | |
|---|---|---|---|---|---|
| ACCH | Accumulator high | IFR | Interrupt flag register | PC | Program counter |
| ACCL | Accumulator low | IMR | Interrupt mask register | PFC | Prefetch counter |
| ALU | Arithmetic logic unit | IR | Instruction register | RPTC | Repeat instruction counter |
| ARAU | Auxiliary register arithmetic unit | MCS | Microcall stack | GREG | Global memory allocation register |
| ARB | Auxiliary register pointer buffer | QIR | Queue instruction register | RSR | Serial port receive shift register |
| ARP | Auxiliary register pointer | PR | Product register | XSR | Serial port transmit shift register |
| DP | Data memory page pointer | PRD | Period register for timer | AR0-AR7 | Auxiliary registers |
| DRR | Serial port data receive register | TIM | Timer | ST0,ST1 | Status registers |
| DXR | Serial port data transmit register | TR | Temporary register | | |

**Fig. 5.** TMS320C25 functional block diagram.

the processor. The diagram also shows all of the TMS320C25 interface pins.

In the following architectural discussions on the memory, central arithmetic logic unit, hardware multiplier, control operations, serial port, and I/O interface, please refer to the block diagram shown in Fig. 5.

*Memory Allocation:* The TMS320C25 provides a total of 4K 16-bit words of on-chip program ROM and 544 16-bit words of on-chip data RAM. The RAM is divided into three separate Blocks (B0, B1, and B2). Of the 544 words, 256 words (block B0) are configurable as either data or program memory by CNFD (configure data memory) or CNFP (configure program memory) instructions provided for that purpose; 288 words (blocks B1 and B2) are always data memory. A data memory size of 544 words allows the TMS320C25 to handle a data array of 512 words while still leaving 32 locations for intermediate storage. The TMS320C25 provides 64K words of off-chip directly addressable data memory space as well as a 64K-word off-chip program memory space.

A register file containing eight Auxiliary Registers (AR0–AR7), which are used for indirect addressing of data memory and for temporary storage, increase the flexibility and efficiency of the device. These registers may be either directly addressed by an instruction or indirectly addressed by a 3-bit Auxiliary Register Pointer (ARP). The auxiliary registers and the ARP may be loaded from either data memory or by an immediate operand defined in the instruction. The contents of these registers may also be stored into data memory. The auxiliary register file is connected to the Auxiliary Register Arithmetic Unit (ARAU). Using the ARAU accessing tables of information does not require the CALU for address manipulation, thus freeing it for other operations.

*Central Arithmetic Logic Unit (CALU):* The CALU contains a 16-bit scaling shifter, a 16 × 16-bit parallel multiplier, a 32-bit Arithmetic Logic Unit (ALU), and a 32-bit accumulator. The scaling shifter has a 16-bit input connected to the data bus and a 32-bit output connected to the ALU. This shifter produces a left-shift of 0 to 16 bits on the input data, as programmed in the instruction. Additional shifters at the outputs of both the accumulator and the multiplier are suitable for numerical scaling, bit extraction, extended-precision arithmetic, and overflow prevention.

The following steps occur in the implementation of a typical ALU instruction:

1) Data are fetched from the RAM on the data bus.
2) Data are passed through the scaling shifter and the ALU where the arithmetic is performed.
3) The result is moved into the accumulator.

The 32-bit accumulator is split into two 16-bit segments for storage in data memory: ACCH (accumulator high) and ACCL (accumulator low). The accumulator has a carry bit to facilitate multiple-precision arithmetic for both addition and subtract instructions.

*Hardware Multiplier:* The TMS320C25 utilizes a 16 × 16-bit hardware multiplier, which is capable of computing a 32-bit product during every machine cycle. Two registers are associated with the multiplier:

- a 16-bit Temporary Register (TR) that holds one of the operands for the multiplier, and
- a 32-bit Product Register (PR) that holds the product.

The output of the product register can be left-shifted 1 or 4 bits. This is useful for implementing fractional arithmetic or justifying fractional products. The output of the PR can also be right-shifted 6 bits to enable the execution of up to 128 consecutive multiple/accumulates without overflow. An unsigned multiply (MPYU) instruction facilitates extended-precision multiplication.

*I/O Interface:* The TMS320C25 I/O space consists of 16 input and 16 output ports. These ports provide the full 16-bit parallel I/O interface via the data bus on the device. A single input (IN) or output (OUT) operation typically takes two cycles; however, when used with the repeat counter, the operation becomes single-cycle. I/O devices are mapped into the I/O address space using the processor's external address and data buses in the same manner as memory-mapped devices. Interfacing to memory and I/O devices of varying speeds is accomplished by using the READY line.

A Direct Memory Access (DMA) to external program/data memory is also supported. Another processor can take complete control of the TMS320C25's external memory by asserting HOLD low, causing the TMS320C25 to place its address, data, and control lines in the high-impedance state. Signaling between the external processor and the TMS320C25 can be performed using interrupts. Two modes of DMA are available on the device. In the first, execution is suspended during assertion of HOLD. In the second "concurrent DMA" mode, the TMS320C25 continues to execute its program while operating from internal RAM or ROM, thus greatly increasing throughput in data-intensive applications.

### *TMS320C25 Software*

The majority of the TMS320C25 instructions (97 out of 133) are executed in a single instruction cycle. Of the 36 instructions that require additional cycles of execution, 21 involve branches, calls, and returns that result in a reload of the program counter and a break in the execution pipeline. Another seven of the instructions are two-word, long-immediate instructions. The remaining eight instructions support I/O, transfers of data between memory spaces, or provide for additional parallel operation in the processor. Furthermore, these eight instructions (IN, OUT, BLKD, BLKP, TBLR, TBLW, MAC, and MACD) become single-cycle when used in conjunction with the repeat counter. The functional performance of the instructions exploits the parallelism of the processor, allowing complex and/or numerically intensive computations to be implemented in relatively few instructions.

*Addressing Modes:* Since most of the instructions are coded in a single 16-bit word, most instructions can be executed in a single cycle. Three memory addressing modes are available with the instruction set: direct, indirect, and immediate addressing. Both direct and indirect addressing are used to access data memory. Immediate addressing uses the contents of the memory addressed by the program counter.

When using direct addressing, 7 bits of the instruction word are concatenated with the 9 bits of the data memory page pointer (DP) to form the 16-bit data memory address. With a 128-word page length, the DP register points to one of 512 possible data memory pages to obtain a 64K total data memory space. Indirect addressing is provided by the aux-

iliary registers (AR0–AR7). The seven types of indirect addressing are shown in Table 4. Bit-reversed indexed addressing modes allow efficient I/O to be performed for the resequencing of data points in a radix-2 FFT program.

**Table 4** Addressing Modes of the TMS320C25

| Addressing Mode | Operation |
|---|---|
| OP A | direct addressing |
| OP * (,NARP) | indirect; no change to AR. |
| OP *+(,NARP) | indirect; current AR is incremented. |
| OP *−(,NARP) | indirect; current AR is decremented. |
| OP *0+(,NARP) | indirect; AR0 is added to current AR. |
| OP *0−(,NARP) | indirect; AR0 is subtracted from current AR. |
| OP *BR0+(,NARP) | indirect; AR0 is added to current AR (with reverse carry propagation). |
| OP *BR0−(,NARP) | indirect; AR0 is subtracted from current AR (with reverse carry propagation). |

*Note:* The optional NARP field specifies a new value of the ARP.

*TMS320C25 System Configurations*

The flexibility of the TMS320C25 allows systems configurations to satisfy a wide range of application requirements [16]. The TMS320C25 can be used in the following configurations:

- a stand-alone system (a single processor using 4K words of on-chip ROM and 544 words of on-chip RAM),
- parallel multiprocessing systems with shared global data memory, or
- host/peripheral coprocessing using interface control signals.

A minimal processing system is shown in Fig. 6 using external data RAM and PROM/EPROM. Parallel multiprocessing and host/peripheral coprocessing systems can be designed by taking advantage of the TMS320C25's direct memory access and global memory configuration capabilities.

In some digital processing tasks, the algorithm being implemented can be divided into sections with a distinct processor dedicated to each section. In this case, the first and second processors may share global data memory, as well as the second and third, the third and fourth, etc. Arbitration logic may be required to determine which section of the algorithm is executing and which processor has access to the global memory. With multiple processors dedicated to distinct sections of the algorithm, throughput can be increased via pipelined execution. The TMS320C25 is capable of allocating up to 32K words of data memory as global memory for multiprocessing applications.

THE THIRD GENERATION OF THE TMS320 FAMILY

The TMS320C30 [26]–[27] is Texas Instruments third-generation member of the TMS320 family of compatible digital signal processors. With a computational rate of 33 MFLOPS (million floating-point operations per second), the TMS320C30 far exceeds the performance of any programmable DSP available today. Total system performance has been maximized through internal parallelism, more than twenty-four thousand bytes of on-chip memory, single-cycle floating-point operations, and concurrent I/O. The total system cost is minimized with on-chip memory and on-chip peripherals such as timers and serial ports. Finally, the user's system design time is dramatically reduced with the availability of the floating-point operations, general-purpose instructions and features, and quality development tools.

The TMS320C30 provides the user with a level of performance that, at one time, was the exclusive domain of supercomputers. The strong architectural emphasis of providing a low-cost system solution to demanding arithmetic algorithms has resulted in the architecture shown in Fig. 7.

The key features of the TMS320C30 [26], [27] are as follows:

- 60-ns single-cycle execution time, 1-$\mu$m CMOS.
- Two 1K × 32-bit single-cycle dual-access RAM blocks.
- One 4K × 32-bit single-cycle dual-access ROM block.
- 64 × 32-bit instruction cache.
- 32-bit instruction and data words, 24-bit addresses.
- 32/40-bit floating-point and integer multiplier.
- 32/40-bit floating-point, integer, and logical ALU.
- 32-bit barrel shifter.
- Eight extended-precision registers.
- Two address-generators with eight auxiliary registers.
- On-chip Direct Memory Access (DMA) controller for concurrent I/O and CPU operation.
- Peripheral bus and modules for easy customization.
- High-level language support.
- Interlocked instructions for multiprocessing support.
- Zero overhead loops and single-cycle branches.

The architecture of the TMS320C30 is targeted at 60-ns and faster cycle times. To achieve such high-performance

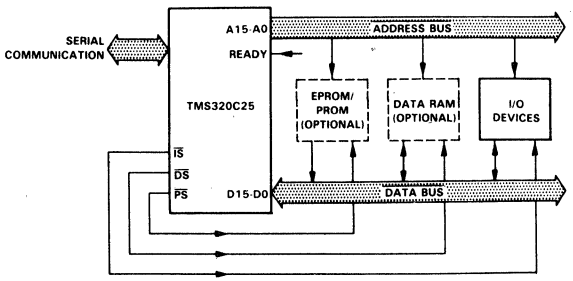



**Fig. 6.** Minimal processing system with external data RAM and PROM/EPROM.

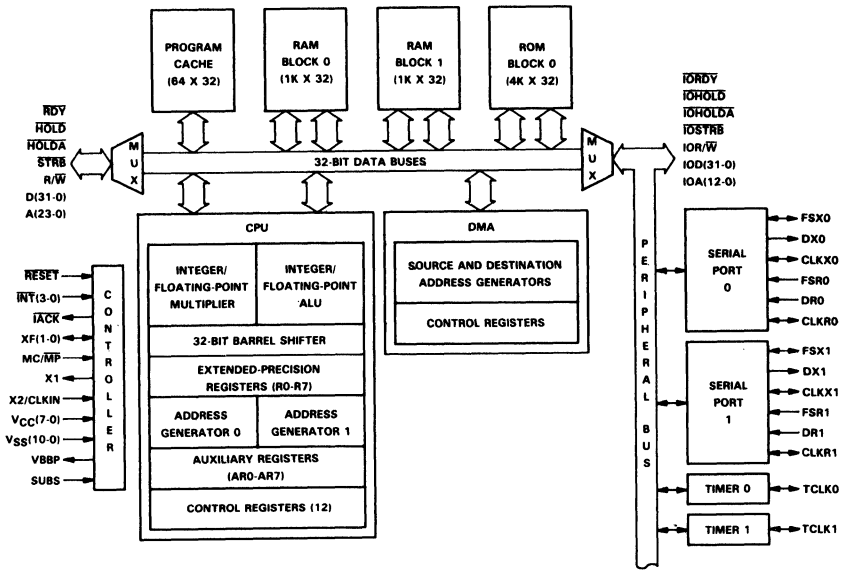



**Fig. 7.** TMS320C30 functional block diagram.

goals while still providing low-cost system solutions, the TMS320C30 is designed using Texas Instruments state-of-the-art 1-μm CMOS process. The TMS320C30 's high system performance is achieved through a high degree of parallelism, the accuracy and precision of its floating-point units, its on-chip DMA controller that supports concurrent I/O, and its general-purpose features. At the heart of the architecture is the Central Processing Unit (CPU).

*The CPU*

The CPU consists of the following elements: floating-point/integer multiplier; ALU for performing floating-point, integer, and logical operations; auxiliary register arithmetic units; supporting register file, and associated buses. The multiplier of the CPU performs floating-point and integer multiplication. When performing floating-point multiplication, the inputs are 32-bit floating-point numbers, and the result is a 40-bit floating-point number. When performing integer multiplication, the input data is 24 bits and yields a 32-bit result. The ALU performs 32-bit integer, 32-bit logical, and 40-bit floating-point operations. Results of the multiplier and the ALU are always maintained in 32-bit integer or 40-bit floating-point formats. The TMS320C30 has the ability to perform, in a single cycle, parallel multiplies and adds (subtracts) on integer or floating-point data. It is this ability to perform floating-point multiplies and adds (subtracts) in a single cycle which give the TMS320C30 its peak computational rate of 33 MFLOPS.

Floating-point operations provide the user with a convenient and virtually trouble-free means of performing computations while maintaining accuracy and precision. The TMS320C30 implementation of floating-point arithmetic allows for floating-point operations at integer speeds. The floating-point capability allows the user to ignore, to a large extent, problems with overflow, operand alignment, and other burdensome tasks common to integer operations.

The register file contains 28 registers, which may be operated upon by the multiplier and ALU. The first eight of these registers (R0–R7) are the extended-precision registers, which support operations on 40-bit floating-point numbers and 32-bit integers.

The next eight registers (AR0–AR7) are the auxiliary registers, whose primary function is related to the generation of addresses. However, they also may be used as general-purpose 32-bit registers. Two auxiliary register arithmetic units (ARAU0 and ARAU1) can generate two addresses in a single cycle. The ARAUs operate in parallel with the multiplier and ALU. They support addressing with displacements, index registers (IR0 and IR1), and circular and bit-reversed addressing.

The remaining registers support a variety of system functions: addressing, stack management, processor status, block repeat, and interrupts.

*Data Organization*

Two integer formats are supported on the TMS320C30: a 16-bit format used for immediate integer operands and a 32-bit single-precision integer format.

Two unsigned-integer formats are available: a 16-bit format for immediate unsigned-integer operands and a 32-bit single-precision unsigned-integer format.

The three floating-point formats are assumed to be normalized, thus providing an extra bit of precision. The first

is a 16-bit short floating-point format for immediate floating-point operands, which consists of a 4-bit exponent, 1 sign bit, and an 11-bit fraction. The second is a single-precision format consisting of an 8-bit exponent, 1 sign bit, and a 23-bit fraction. The third is an extended-precision format consisting of an 8-bit exponent, 1 sign bit, and a 31-bit fraction.

The total memory space of the TMS320C30 is 16M (million) × 32 bits. A machine word is 32 bits, and all addressing is performed by word. Program, data, and I/O space are contained within the 16M-word address space.

RAM blocks 0 and 1 are each 1K × 32 bits. The ROM block is 4K × 32 bits. Each RAM block and ROM block is capable of supporting two data accesses in a single cycle. For example, the user may, in a single cycle, access a program word and a data word from the ROM block.

The separate program data, and DMA buses allow for parallel program fetches, data reads and writes, and DMA operations. Management of memory resources and busing is handled by the memory controller. For example, a typical mode of operation could involve a program fetch from the on-chip program cache, two data fetches from RAM block 0, and the DMA moving data from off-chip memory to RAM block 1. All of this can be done in parallel with no impact on the performance of the CPU.

A 64 × 32-bit instruction cache allows for maximum system performance with minimal system cost. The instruction cache stores often repeated sections of code. The code may then be fetched from the cache, thus greatly reducing the number of off-chip accesses necessary. This allows for code to be stored off-chip in slower, lower cost memories. Also, the external buses are freed, thus allowing for their use by the DMA or other devices in the system.

*DMA*

The TMS320C30 processes an on-chip Direct Memory Access (DMA) controller. The DMA controller is able to perform reads from and writes to any location in the memory map without interfering with the operation of the CPU. As a consequence, it is possible to interface the TMS320C30 to slow external memories and peripherals (A/Ds, serial ports, etc.) without affecting the computational throughput of the CPU. The result is improved system performance and decreased system cost.

The DMA controller contains its own address generators, source and destination registers, and transfer counter. Dedicated DMA address and data buses allow for operation with no conflicts between the CPU and DMA controller.

The DMA controller responds to interrupts in a similar way to the CPU. This ability allows the DMA to transfer data based upon the interrupts received. Thus I/O transfers that would normally be performed by the CPU may instead be performed by the DMA. Again, the CPU may continue processing data while the DMA receives or transmits data.

*Peripherals*

All peripheral modules are manipulated through memory-mapped registers located on a dedicated peripheral bus. This peripheral bus allows for the straightforward addition, removal, and creation of peripheral modules. The initial TMS320C30 peripheral library will include timers and serial ports. The peripheral library concept allows Texas Instruments to create new modules to serve a wide variety of applications. For example, the configuration of the TMS320C30 in Fig. 7 includes two timers and two serial ports.

*Timers:* The two timer modules are general-purpose timer/event counters, with two signaling modes and internal or external clocking.

Available to each timer is an I/O pin that can be used as an input clock to the timer or as an output signal driven by the timer. The pin may also be configured as a general-purpose I/O pin.

*Serial Ports:* The two serial ports are modular and totally independent. Each serial port can be configured to transfer 8, 16, 24, or 32 bits of data per frame. The clock for each serial port can originate either internally or externally. An internally generated divide-down clock is provided. The pins of the serial ports are configurable as general-purpose I/O pins. A special handshake mode allows TMS320C30s to communicate over their serial ports with guaranteed synchronization. The serial ports may also be configured to operate as timers.

*External Interfaces*

The TMS320C30 provides two external interfaces: the parallel interface and the I/O interface. The parallel interface consists of a 32-bit data bus, a 24-bit address bus, and a set of control signals. The I/O interface consists of a 32-bit data bus, a 13-bit address bus, and a set of control signals. Both ports support an external ready signal for wait-state generation and the use of software-controlled wait states.

The TMS320C30 supports four external interrupts, a number of internal interrupts, and a nonmaskable external reset signal. Two dedicated, general-purpose, external I/O flags, XF0 and XF1, may be configured as input or output pins under software control. These pins are also used by the interlocked instructions to support multiprocessor communication.

*Pipelining In the TMS320C30*

The operation of the TMS320C30 is controlled by five major functional units. The five major units and their function are as follows:

- *Fetch Unit* (F) which controls the program counter updates and fetches of the instruction words from memory.
- *Decode Unit* (D) which decodes the instruction word and controls address generation.
- *Read Unit* (R) which controls the operand reads from memory.
- *Execute Unit* (E) which reads operands from the register file, performs the necessary operation, and writes results back to the register file and memory.
- *DMA Channel* (DMA) which reads and writes memory concurrently with CPU operation.

Each instruction is operated upon by four of these stages; namely, fetch, decode, read, and execute. To provide for maximum processor throughput these units can perform in parallel with each unit operating on a different instruction. The overlapping of the fetch, decode, read, and execute operations of different instructions is called pipelining. The DMA controller runs concurrently with these units. The pipelining of these operations is key to the high per-

formance of the TMS320C30. The ability of the DMA to move data within the processor's memory space results in an even greater utilization of the CPU with fewer interruptions of the pipeline which inevitably yields greater performance.

The pipeline control of the TMS320C30 allows for extremely high-speed execution rate by allowing an effective rate of one execution per cycle. It also manages pipeline conflicts in a way that makes them transparent to the user.

While the pipelining of the different phases of an instruction is key to the performance of the TMS320C30, the designers felt it essential to avoid pipelining the operation of the multiplier or ALU. By ruling out this additional level of pipelining it was possible to greatly improve the processor's useability.

*Instructions*

The TMS320C30 instruction set is exceptionally well suited to digital signal processing and other numerically intensive applications. The TMS320C30 also possesses a full complement of general-purpose instructions. The instruction set is organized into the following groups:

- load and store instructions;
- two-operand arithmetic instructions;
- two-operand logical instructions;
- three-operand arithmetic instructions;
- three-operand logic instructions;
- parallel operation instructions;
- arithmetic/logical instruction with store instructions;
- program control instructions;
- interlocked operations instructions.

The load and store instructions perform the movement of a single word to and from the registers and memory. Included is the ability to load a register conditionally. This operation is particularly useful for locating the maximum and minimum of a set of data.

The two-operand arithmetic and logical instructions consist of a complete set of arithmetic instructions. They have two operands; src and dst for source and destination, respectively. The src operand may come from memory, a register, or be part of the instruction word. The dst operand is always a register. This portion of the instruction set includes floating-point integer and logical operations, support of multiprecision arithmetic, and 32-bit arithmetic and logical shifts.

The three-operand arithmetic and logical instructions are a subset of the two-operand arithmetic and logical instructions. They have three operands: two src operands and a dst operand. The src operands may come from memory or a register. The dst operand is always a register. These instructions allow for the reading of two operands from memory and/or the CPU register file in a single cycle.

The parallel operation instructions allow for a high degree of parallelism. They support very flexible, parallel floating-point and integer multiplies and adds. They also include the ability to load two registers in parallel.

The arithmetic/logical and store instructions support a high degree of parallelism, thus complementing the parallel operation instructions. They allow for the performance of an arithmetic or logical instruction between a register and an operand read from memory, in parallel with the storing of a register to memory. They also provide for extremely rapid operations on blocks of memory.

The program control instructions consist of all those operations that affect the program flow. This section of the instruction set includes a set of flexible and powerful constructs that allow for software control of the program flow. These fall into two main types: repeat modes and branching.

For many algorithms, there is an inner kernel of code where most of the execution time is spent. The repeat modes of the TMS320C30 allow for the implementation of zero overhead looping. Using the repeat modes allows these time-critical sections of code to be executed in the shortest possible time. The instructions supporting the repeat modes are RPTB (repeat a block of code) and RPTS (repeat a single instruction). Through the use of the dedicated stack-pointer, block repeats (RPTBs) may be nested.

The branching capabilities of the TMS320C30 include two main subsets: standard and delayed branches. Standard branches, as in any pipelined machine that comprehends them, empty the pipeline to guarantee correct management of the program counter. This results in a branch requiring, in the case of the TMS320C30, four cycles to execute. Included in this subset are calls and returns. A standard branch (BR) is illustrated below.

```
        BR    THREE   ; standard branch.
        MPYF          ; not executed.
        ADDF          ; not executed.
        SUBF          ; not executed.
        AND           ; not executed.
         .
         .
         .
THREE   MPYF          ; fetched 3 cycles after BR
                        is fetched.
         .
         .
         .
```

Delayed branches do not empty the pipe, but rather, guarantee that the next three instructions will be fetched before the program counter is modified by the branch. The result is a branch that only requires a single cycle. Every delayed branch has a standard branch counterpart. A delayed branch (BRD) is illustrated below.

```
        BRD   THREE   ; delayed branch.
        MPYF          ; executed.
        ADDF          ; executed.
        SUBF          ; executed.
        AND           ; not executed.
         .
         .
         .
THREE   MPYF          ; fetched after SUBF fetched.
         .
         .
         .
```

The combination of the repeat modes, standard branches, and delayed branches provides the user with a set of programming constructs which are well suited to a wide range of performance requirements.

The program control instructions also include conditional calls and returns. The decrement and branch conditionally instruction allows for efficient loop control by combining the comparison of a loop counter to zero with

the check of condition flags, i.e., floating-point overflow. The condition codes available include unsigned and signed comparisons, comparisons to zero, and comparisons based upon the status of individual condition flags. These conditions may be used with any of the conditional instructions.

The interlocked operations instructions support multiprocessor communication. Through the use of external signals, these instructions allow for powerful synchronization mechanisms, such as semaphores, to be implemented. The interlocked operations use the two external flag pins, XF0 and XF1. XF0 signals an interlocked-operation request and XF1 acts as an acknowledge signal for the requested interlocked operation. The interlocked operations include interlocked loads and stores. When an interlocked operation is performed the external request and acknowledge signals can be used to arbitrate between multiple processors sharing memory, semaphores, or counters.

### Development and Support Tools

Digital signal processors are essentially application-specific microprocessors (or microcomputers). Like any other microprocessor, no matter how impressive the performance of the processor or the ease of interfacing, without good development tools and technical support, it is very difficult to design it into the system. In developing an application, problems are encountered and questions are asked. Oftentimes the tools and vendor support provided to the designer are the difference between the success and failure of the project.

The TMS320 family has a wide range of development tools available [25]. These tools range from very inexpensive evaluation modules for application evaluation and benchmarking purposes, assembler/linkers, and software simulators, to full-capability hardware emulators. A brief summary of these support tools is provided in the succeeding subsections.

#### *Software Tools*

Assembler/linkers and software simulators are available on PC and VAX for users to develop and debug TMS320 DSP algorithms. Their features are described as follows:

*Assembler/Linker:* The Macro Assembler translates assembly language source code into executable object code. The Linker permits a program to be designed and implemented in separate modules that will later be linked together to form the complete program.

*Simulator:* The Simulator simulates operations of the device in software to allow program verification and debug. The simulator uses the object code produced by the Macro Assembler/Linker.

*C Complier:* The C Compiler is a full implementation of the standard Kernighan and Ritchie C as defined in *The C Programming Language* [28]. The compiler supports the insertion of assembly language code into the C source code. The user may also write functions in assembly language, and then call these functions from the C source. Similarly, C functions may be called from assembly language. Variables defined in the C source may be accessed in assembly language modules and *vice versa*. The result is a complier that allows the user to tailor the amount of high-level programming versus the amount of assembly language according to his application. The C compiler is supported on the TMS320C25 and the TMS320C30.

#### *Hardware Tools*

Evaluation modules and emulation tools are available for in-circuit emulation and hardware program debugging for developing and testing DSP algorithms in a real product environment.

*Evaluation Module (EVM):* The EVM is a stand-alone single-board module that contains all of the tools necessary to evaluate the device as well as provide basic in-circuit emulation. The EVM contains a debug monitor, editor, assembler, reverse assembler, and software communications to a host computer or a line printer.

*SoftWare Development System (SWDS):* The SoftWare Development System is a PC plug-in card with similar functionality of the EVM.

*Emulator (XDS):* The eXtended Development System provides full-speed in-circuit emulation with real-time hardware breakpoint/trace and program execution capability from target memory. By setting breakpoints based on internal conditions or external events, execution of the program can be suspended and the XDS placed into the debug mode. In the debug mode, all registers and memory locations can be inspected and modified. Full-trace capabilities at full speed and a reverse assembler that translates machine code back into assembly instructions are included. The XDS system is designed to interface with either a terminal or a host computer. In addition to the above design tools, other development support is available [25]:

### Applications

The TMS320 is designed for real-time DSP and other computation-intensive applications [4]. In these applications, the TMS320 provides an excellent means for executing signal processing algorithms such as fast Fourier transforms (FFTs), digital filters, frequency synthesis, correlation, and convolution. The TMS320 also provides for more general-purpose functions via bit-manipulation instructions, block data move capabilities, large program and data memory address spaces, and flexible memory mapping.

To introduce applications performed by the TMS320, digital filters will be used as examples. The remaining portion of this section will briefly cover applications, and conclude by showing some benchmarks.

#### *Digital Filtering*

As discussed several times in this paper, the FIR filter is simply the sum of products in a sampled data system. This was shown in (1). A simple implementation of the FIR filter uses the MACD instruction (multiply/accumulate and data move) for each filter tap, with the RPT/RPTK instruction repeating the MACD for each filter tap. As we saw earlier, a 256-tap FIR filter can be implemented by using the following two instructions:

```
RPTK   255
MACD   *-,COEFFP
```

In this example, the coefficients may be stored anywhere in program memory (reconfigurable on-chip RAM, on-chip ROM, or external memories). When the coefficients are

For this application, a large on-chip RAM of 544 words and on-chip ROM of 4K words on the TMS320C25 provides for a 256-tap adaptive filter (32-ms echo cancellation) to be executed in a single chip without external data or program memory.

*High-Speed Modems:* The TMS320 can perform numerous functions such a modulation/demodulation, adaptive equalization, and echo cancellation [21], [22]. For lower speed modems, such as Bell 212A and V.22 bis modems, the TMS320C17 provides the most cost-effective single-chip solution to these applications. For higher speed modems, such as the V.32, requiring more processing power and multiprocessing capabilities, the TMS320C25 and TMS320C30 are the designer's choice.

*Voice Coding:* Voice-coding techniques [3], [4], such as full-duplex 32-kbit/s ADPCM (CCITT G.721), CVSD, 16-kbit/s subband coders, and LPC, are frequently used in voice transmission and storage. Arithmetic speed, normalization, and the bit-manipulation capability of the TMS320 provide for implementation of these functions, usually in a single chip. For example, the TMS320C17 can be used as a single-chip ADPCM [4], subband [4], or LPC [4] coder. An application of voice coding is an ADPCM transcoder implemented in half-duplex on a single TMS320C17 or full-duplex on a TMS320C25 for telecommunication multiplexing applications. Another example is a secure-voice communication system, requiring voice coding, as well as data encryption and transmission over a public-switched network via a modem; the TMS320C25 offers an ideal solution.

### *Graphics/Image Processing Applications*

In graphics and image processing applications [4], the ability to interface with a host processor is important. Both the TMS320C30 and the TMS320C25 multiprocessor interface enable them to be used in a variety of host/coprocessor configurations [4]. Graphics and image processing applications can use the large directly addressable external data space and global memory capability to allow graphical images in memory to be shared with a host processor, thus minimizing unnecessary data transfers. The indexed indirect addressing modes allow matrices to be processed row-by-row when performing matrix multiplication for three-dimensional image rotations, translations, and scaling.

The TMS320C30 has a number of features that support graphics and image processing extremely well. The floating-point capabilities allow for extremely precise computation of perspective transformations. They also support more sophisticated algorithms such as shading and hidden line removal, operations which are computationally intensive.

The large address space allows for straightforward addressing of large images or displays. The flexible addressing registers, coupled with the integer multiply, support powerful addressing of multiple-dimensional arrays. Vector-oriented instructions allow the user to efficiently manipulate large blocks of memory. Finally, the on-chip DMA controller allows the user to easily overlap the processing of data with its I/O.

### *High-Speed Control*

High-speed control applications [4], [24] use the TMS320C17 and TMS320C25 general-purpose features for bit-test and logical operations, timing synchronization, and high data-transfer rate (ten million 16-bit words per second). Both devices can be used in closed-loop systems for control signal conditioning, filtering, high-speed computing, and multichannel multiplexing capabilities. The following demonstrates two typical control applications:

*Disk Control:* Digital filtering in a closed-loop actuation mechanism positions the read/write heads over the disk surface. Supplemented with many general-purpose features, the TMS320 can replace costly bit-slice/custom/analog solutions to perform such tasks as compensation, filtering, fine/coarse tuning, and other signal conditioning algorithms.

*Robotics:* Digital signal processing and bit-manipulation power, coupled with host interface, allow the TMS320C25 to be useful in robotics control [24]. The TMS320C25 can replace both the digital controllers and analog signal processing hardware for communication to a central host processor and for the performance of numerically intensive control functions.

### *Instrumentation*

Instrumentation, such as spectrum analyzers and various high-speed/high-precision instruments, often requires a large data memory space and the high performance of a digital signal processor. The TMS320C25 and TMS320C30 are capable of performing very long-length FFTs and generating precision functions with minimal external hardware.

### *Numeric Processing*

Numeric and array processing applications benefit from TMS320 performance. High throughput resulting from features, such as a fast cycle time and an on-chip hardware multiplier, combined with multiprocessing capabilities and data memory expansion, provide for a low-cost, easy-to-use replacement for a typical bit-slice solution. The TMS320C30's floating-point precision, high throughput, and interface flexibility are excellent for this application.

### *TMS320 Benchmarks*

To complete the discussion on the applications that the TMS320 can perform, we will provide some benchmarks. The TMS320 has demonstrated impressive benchmarks in performing some of the common DSP routines and system applications. Table 5 shows typical TMS320 benchmarks [4].

**Table 5** TMS320 Family Benchmarks

| DSP Routines/Applications | First Generation | Second Generation | Third Generation |
|---|---|---|---|
| FIR filter tap | 400 ns | 100 ns | 60 ns |
| 256-tap FIR sample rate | 9.25 kHz | 37 kHz | >60 kHz |
| LMS adaptive FIR filter tap | 700 ns | 400 ns | 180 ns |
| 256-tap adaptive FIR filter sample rate | 5.4 kHz | 9.5 kHz | >20 kHz |
| Bi-quad filter element (five multiplies) | 2 $\mu$s | 1 $\mu$s | 360 ns |
| Echo canceler (single chip) | 8 ms | 32 ms | >64 ms |

## Summary

This paper has discussed characteristics of digital signal processing and how these characteristics have influenced the architectural design of the Texas Instruments TMS320 family of digital signal processors. Three generations of the

TMS320 family were covered, and their support tools necessary to develop end-applications were briefly reviewed. The paper concluded with an overview of digital signal processing applications using these devices.

## References


[1] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975.

[2] A. V. Oppenheim, Ed., *Applications of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1978.

[3] L. R. Rabiner and R. W. Schafer, *Digital Processing of Speech Signals*. Englewood Cliffs, NJ: Prentice-Hall, 1978.

[4] K. Lin, Ed., *Digital Signal Processing Applications with the TMS320 Family*. Englewood Cliffs, NJ: Prentice-Hall, 1987

[5] A. V. Oppenhiem and R. W. Schafer, *Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975.

[6] C. Burrus and T. Parks, *DFT/FFT and Convolution Algorithms*. New York, NY: Wiley, 1985.

[7] T. Parks and C. Burrus, *Digital Filter Design*. New York, NY: Wiley, 1987.

[8] J. Treichler, C. Johnson, and M. Larimore, *A Practical Guide to Adaptive Filter Design*. New York, NY: Wiley, 1987.

[9] P. Papamichalis, *Practical Approaches to Speech Coding*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[10] R. Morris, *Digital Signal Processing Software*. Ottawa, Ont., Canada: DSPS Inc., 1983.

[11] K. McDonough, E. Caudel, S. Magar, and A. Leigh, "Microcomputer with 32-bit arithmetic does high-precision number crunching," *Electronics*, pp. 105–110, Feb. 24, 1982.

[12] S. Magar, E. Caudel, and A. Leigh, "A Microcomputer with digital signal processing capability," in *1982 Int. Solid State Conf. Dig. Tech. Pap.*, pp. 32–33, 284, 285.

[13] *First Generation TMS320 User's Guide*. Houston, TX: Texas Instruments Inc., 1987.

[14] *TMS320 First-Generation Digital Signal Processors Data Sheet*. Houston, TX: Texas Instruments Inc., 1987.

[15] *TMS32020 User's Guide. Houston, TX: Texas Instruments Inc., 1985.*

[16] *TMS320C25 User's Guide*. Houston, TX: Texas Instruments Inc., 1986.

[17] *TMS32011 User's Guide*. Houston, TX: Texas Instruments Inc., 1985.

[18] H. Cragon, "The elements of single-chip microcomputer architecture," *Comput. Mag.*, vol. 13, no. 10, pp. 27–41, Oct. 1980.

[19] S. Rosen, "Electronic computers: A historical survey," *Comput. Surv.*, vol. 1, no. 1, Mar. 1969.

[20] M. Honig and D. Messerschmitt, *Adaptive Filters*. Dordrecht, The Netherlands: Kluwer, 1984.

[21] R. Lucky *et al.*, *Principles of Data Communication*. New York, NY: McGraw-Hill, 1965.

[22] P. Van Gerwen *et al.*, "Microprocessor implementation of high speed data modems," *IEEE Trans. Commun.*, vol. COM-25, pp. 238–249, 1977.

[23] M. Bellanger, "New applications of digital signal processing in communications," *IEEE ASSP Mag.*, pp. 6–11, July 1986.

[24] Y. Wang, M. Andrews, S. Butner, and G. Beni, "Robot-controller system," in *Proc. Symp. on Incremental Motion Control Systems and Devices*, pp. 17–26, June 1986.

[25] *TMS320 Family Development Support Reference Guide*. Houston, TX: Texas Instruments Inc., 1986.

[26] R. Simar, T. Leigh, P. Koeppen, J. Leach, J. Potts, and D. Blalock, "A 40 MFLOPS digital signal processor: The first supercomputer on a chip," in *Proc. IEEE Int'l Conf. on Acoustics, Speech, and Signal Processing*, Apr. 1987.

[27] *TMS320C30 User's Guide*. Houston, TX: Texas Instruments Inc., 1987.

[28] B. Kernighan and D. Ritchie, *The C Programming Language*. Englewood Cliffs, NJ: Prentice-Hall, 1978.

# The Texas Instruments TMS320C25 Digital Signal Microcomputer

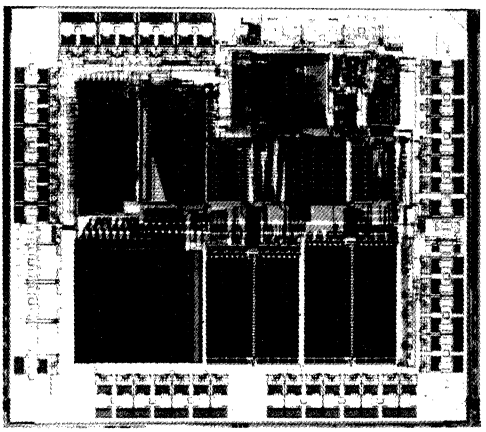
Gene A. Frantz
Kun-Shan Lin
Jay B. Reimer
Jon Bradley

Digital Signal Processor Products — Semiconductor Group
Texas Instruments

Reprinted from
*IEEE MICRO MAGAZINE*
Vol. 6, No. 6, December 1986

# The Texas Instruments TMS320C25 Digital Signal Microcomputer

Gene A. Frantz, Kun-Shan Lin,
Jay B. Reimer, and Jon Bradley
Texas Instruments Incorporated



*Capable of 10 million operations per second, the newest member of the TMS320 family can serve as an inexpensive alternative to bit-slice processors or custom ICs in digital signal processing applications.*

Digital signal processing encompasses a variety of applications, including digital filtering, speech vocoding, image processing, fast Fourier transforms, and digital audio.[1-5] All DSP applications have several characteristics in common. First, they employ algorithms that are mathematically intensive. An example is the finite-duration impulse response, or FIR, filter, which in the time domain takes the form

$$y(n) = \sum_{i=1}^{N} a(i) \cdot x(n-i), \quad (1)$$

where $y(n)$ is the output sample at time $n$, $a(i)$ is the $i$th coefficient or weighting factor, and $x(n-i)$ is the $(n-i)$th input sample. From this equation, we can see that the FIR filter contains an abundance of multiplications and additions (that is, sums of products). This equation is the general form of an FIR filter[6] as well as the convolution of two sequences of numbers $a(i)$ and $x(i)$.[7] Both operations are fundamental to digital signal processing.

Second, DSP algorithms must be performed in real time; i.e., they must not produce a delay noticeable to the user. In a speech recognition system, for example, the algorithms must not produce a noticeable delay between a word being spoken and that word being recognized. In an image processing system, processing needs to be completed within a frame update period.

Third, all DSP applications involve the sampling of a signal. Referring to Equation 1, we can see that the output $y(n)$ is calculated to be the weighted sum of the previous $N$ inputs. In other words, the input signal is sampled at periodic intervals, and the samples are multiplied by a weighting factor $a(i)$ and then added together to give the output result $y(n)$. In a typical DSP application, the processor must be able to perform arithmetic computations and effectively handle sampled data in large quantities.

Last, DSP systems must be flexible enough to incorporate improvements in the state of the art. Many DSP techniques are still developing, and therefore their algorithms tend to change. Speech recognition, for example, is presently an inexact technique still undergoing algorithmic modification. This implies that DSP systems need to be programmable so that they can easily accommodate revised algorithms.

Over the past several decades, digital signal processing machines have taken several forms in response to application need and available technology. Array processors have long been the accepted solution for the research laboratory and have been extended to end applications in some instances. However, as integrated circuit technology has matured, digital signal processing has migrated from the array processor to the bit-slice processor to the single-chip processor. This has brought the cost of DSP solutions down to a point that allows pervasive use of the technology.

The members of the TMS320 family of devices are examples of the single-chip digital signal processor. The first member of the family, the TMS32010, was introduced to the market in 1983.[8,9] It can perform five million DSP

operations per second, including the add and multiply functions [10] required in Equation 1. The newest member of the family, the TMS320C25, can perform 10 million DSP operations per second, [11] and it combines the multiply/accumulate functions into one single-cycle operation.

## Basic TMS320 architecture

The fundamental attribute of a digital signal processor is fast arithmetic operations. The members of the TMS320 family, [10-12] like many other digital signal processors, achieve fast arithmetic operations by employing

- a Harvard architecture,
- a dedicated hardware multiplier,
- special DSP instructions, and
- extensive pipelining.

Use of these concepts allows a digital signal processor to handle a vast amount of data and execute most DSP operations in a one-cycle instruction.

The TMS320 family utilizes a modified Harvard architecture for speed and flexibility. In a strict Harvard architecture, [13,14] the program memory and data memory lie in two separate spaces, permitting a full overlap of the instruction fetch and execution. The TMS320 family's modification of the Harvard architecture allows transfers between the program space and data space, thereby increasing the flexibility of the devices in the family. This architectural modification eliminates the need for a separate coefficient ROM and also maximizes processing power by maintaining two separate bus structures (program and data) for full-speed execution.

The TMS320 family's dedicated hardware multiplier employs a 16 × 16-bit organization, which yields a 32-bit result and allows multiplication to take place in a single cycle. The special DSP instructions include DMOV (data move) and RPT (repeat), which speed up DSP operations. The extensive pipelining ensures maximum throughput for real-time applications.

## The TMS320C25 architecture

The TMS320C25 digital signal processor is a microcomputer with a 32-bit internal Harvard architecture and a 16-bit external interface. It is a pin-compatible CMOS version of the TMS32020 microprocessor but has an instruction execution rate twice as fast and includes additional hardware and software features. The TMS320C25's instruction set is a superset of that of the TMS32010 and that of the TMS32020, and it maintains source-code compatibility with them. In addition, it is completely object-code-compatible with the TMS32020 so that TMS32020 programs can run unmodified on the TMS320C25. Some of the major features of the TMS320C25 are

- a 32-bit ALU and accumulator,
- an instruction cycle time of 100 ns,
- a single-cycle multiply/accumulate,
- use of low-power CMOS technology with a power-down mode,
- 4K 16-bit words of on-chip masked ROM,
- 544 words of on-chip data RAM,
- 128K words of data/program memory space,
- eight auxiliary registers with a dedicated arithmetic unit,
- an eight-level hardware stack,
- a fully static double-buffered serial port,
- concurrent DMA that uses an extended hold operation,
- bit-reversed addressing modes for fast Fourier transforms,
- extended-precision arithmetic and adaptive filtering support,
- full-speed operation of data move instructions from external memory,
- an accumulator carry bit and related instructions, and
- fabrication in 1.8-$\mu$m CMOS and packaging in a 68-pin PLCC.

The 100-ns instruction cycle time provides a significant throughput advantage for many applications. Since most of the TMS320C25's instructions can execute in a single cycle, it can execute 10 million instructions per second. Most of the other features listed above also contribute to the TMS320C25's high throughput.

The TMS320C25 includes instructions to perform the data transfers between program space and memory space discussed earlier. Externally, the program and data memory spaces are multiplexed over the same bus so as to maximize the address range for both spaces and minimize the pin count of the device. Internally, the TMS320C25 architecture maximizes processing power by maintaining two separate bus structures, program and data, for full-speed execution.

Program execution in the device takes the form of a three-level instruction fetch-decode-execute pipeline. This pipeline is invisible to the user except in cases in which it must be broken, such as for branch instructions. In this case, the instruction timing takes into account the fact that the pipeline must be emptied and refilled.

Two large, on-chip data RAM blocks (a total of 544 words), one of which is configurable either as program or data memory, are provided. An off-chip, 64K-word, directly addressable data memory address space is included to facilitate implementations of DSP algorithms with large data memory requirements. Four-K words of on-chip program ROM and 64K words of off-chip program address space are available. Large programs can execute at full speed from this memory space. Programs can also be

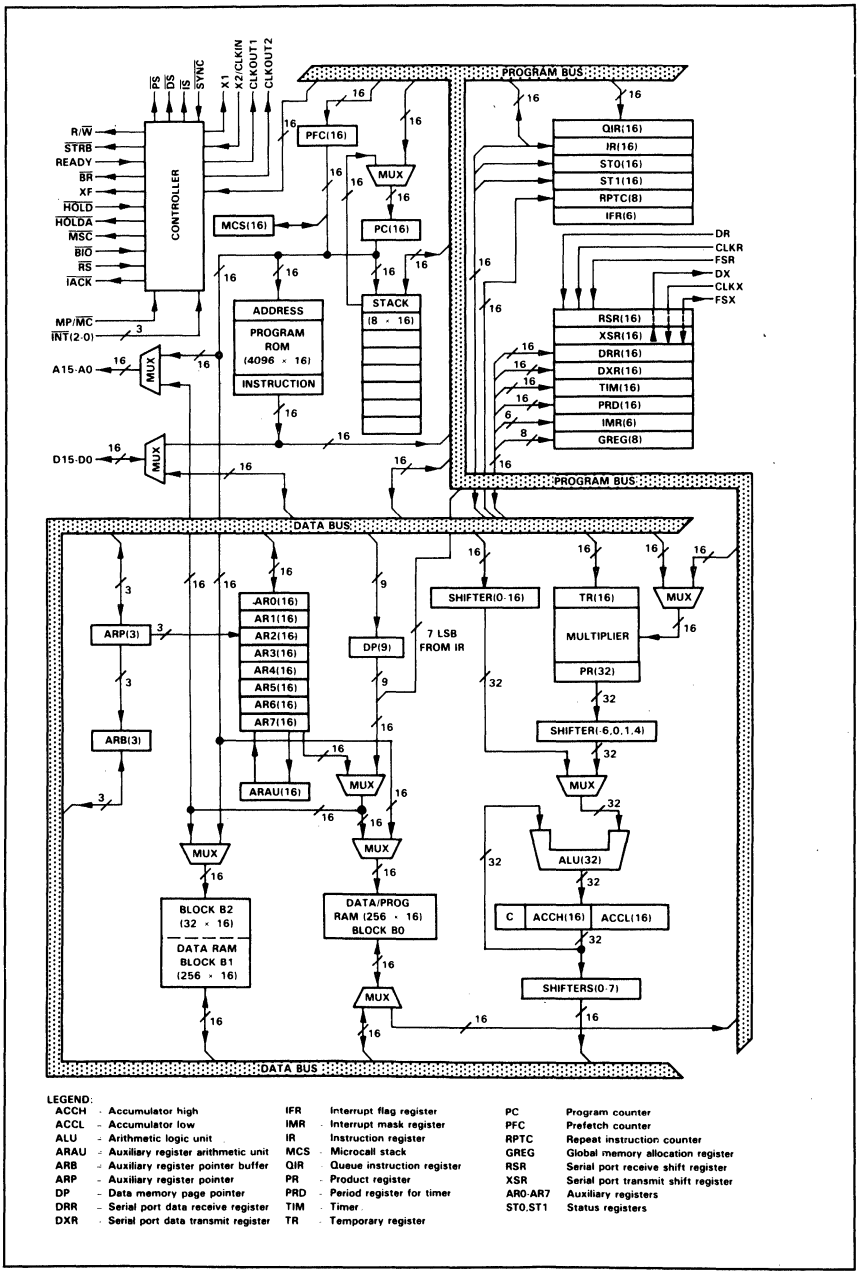



**Figure 1.**
**TMS320C25**
**block diagram.**

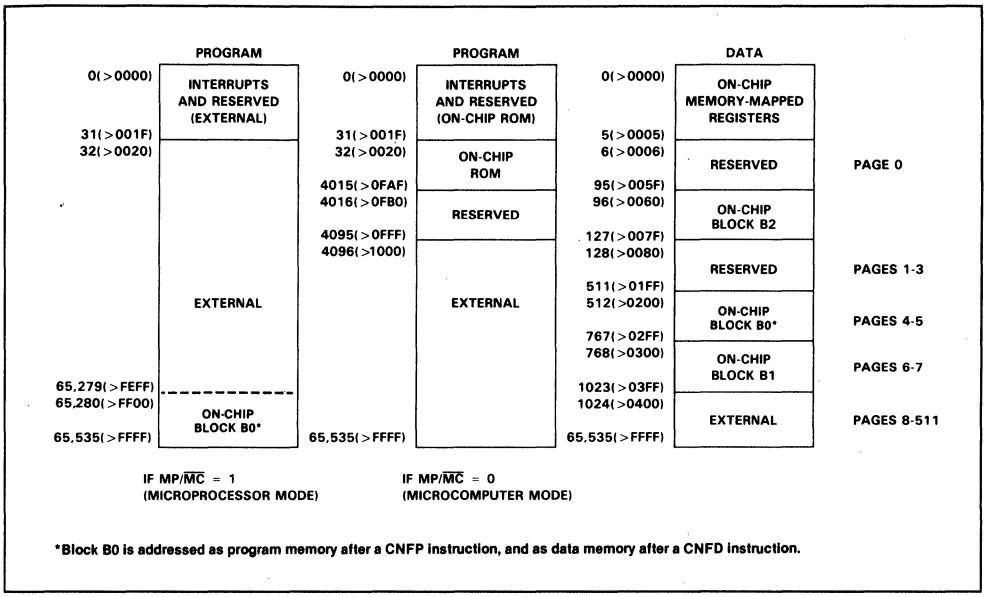



Figure 2. TMS320C25 memory maps.

downloaded from slow external memory to on-chip RAM for full-speed operation.

The TMS320C25 also incorporates a hardware timer and a block data transfer capability.

The diagram of the TMS320C25 in Figure 1 shows the principal blocks and data paths within the processor. It also shows all of the TMS320C25's interface pins.

The TMS320C25's architecture is built around the program and data buses. The program bus carries the instruction code and immediate operands from program memory. The data bus interconnects elements such as the central arithmetic logic unit (CALU) and the auxiliary register file to the data RAM. Together, the program and data buses can carry data from on-chip data RAM and internal or external program memory to the multiplier in a single cycle for multiply/accumulate operations.

A high degree of parallelism exists in the device—for example, while data are being operated on by the CALU, arithmetic operations can be implemented in the auxiliary register arithmetic unit (ARAU). Such parallelism results in a powerful set of arithmetic, logical, and bit-manipulation operations that can be performed in a single machine cycle.

**Memory allocation.** As mentioned above, the TMS320C25 provides 4K 16-bit words of on-chip program ROM and 544 16-bit words of on-chip data RAM. The RAM is divided into three blocks, B0, B1, and B2. Of the 544 words, 256 words (block B0) are configurable as either data memory or program memory; 288 words (blocks B1 and B2) are always data memory. A data memory size of 544 words allows the TMS320C25 to handle a data array of 512 words but still leaves 32 locations for intermediate storage.

The TMS320C25 maintains separate address spaces for program memory, data memory, and I/O. In addition to blocks B0, B1, and B2, the on-chip data memory map (see Figure 2) includes memory-mapped registers. Six peripheral registers, the serial-port registers (DRR and DXR), timer register (TIM), period register (PRD), interrupt mask register (IMR), and global memory allocation register (GREG), have been mapped into the data memory space so they can be easily modified.

The TMS320C25 has a register file containing eight auxiliary registers that can be used for indirect addressing of data memory or for temporary storage. These registers, AR0-AR7, can be either directly addressed by an instruction or indirectly addressed by a three-bit auxiliary register pointer (ARP). The auxiliary registers and the ARP can be loaded either from data memory or by an immediate operand defined in the instruction. The contents of the registers can also be stored in data memory.

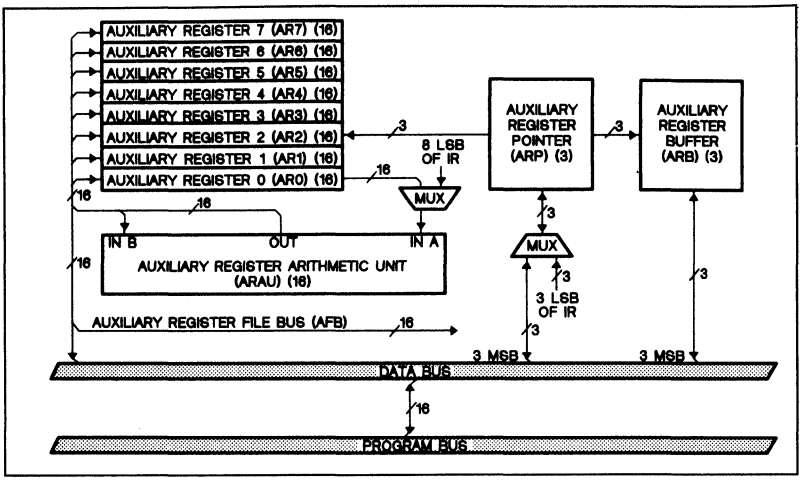



Figure 3. Auxiliary register file.

The auxiliary register file is connected to the auxiliary register arithmetic unit as shown in Figure 3. The ARAU can autoindex the current auxiliary register while the data memory location is being addressed. The current auxiliary register can also be indexed either by $+1/-1$ or by the contents of AR0. As a result, the accessing of tables of information does not require the CALU for address manipulation, thereby freeing it for other operations.

Although the ARAU was designed to support address manipulation in parallel with other operations, it can also serve as an additional general-purpose arithmetic unit since the auxiliary register file can communicate directly with data memory. The ARAU implements 16-bit unsigned arithmetic, whereas the CALU implements 32-bit two's-complement arithmetic. The ARAU also provides branches dependent on the comparison of AR0 to the auxiliary register pointed to by the ARP.

**Central arithmetic logic unit.** The CALU contains a 16-bit scaling shifter, a 16 × 16-bit parallel multiplier, a 32-bit ALU, and a 32-bit accumulator. The scaling shifter has a 16-bit input connected to the data bus and a 32-bit output connected to the ALU. This shifter produces a left shift of 0 to 16 bits on the input data, as programmed in the instruction. The least significant bits of the output are filled with zeroes, and the most significant bits are either filled with zeroes or sign-extended, depending upon the state of the sign-extension mode bit of status register ST1. Additional shifters at the outputs of both the accumulator and the multiplier are suitable for numerical scaling, bit extraction, extended-precision arithmetic, and overflow prevention. Due to the pipelining in the TMS320C25, shifting is accomplished as part of an instruction and thus does not require additional cycles for execution.

The 32-bit ALU and accumulator perform a wide range of arithmetic and logical instructions. An overflow saturation mode permits the accumulator to be loaded with the most positive or negative number (the choice depending on the direction of overflow), and it allows an overflow flag to be set whenever an overflow occurs. One of the two inputs to the ALU is always provided from the accumulator, and the other may be transferred from the product register (PR) of the multiplier or from the scaling shifter loaded from data memory.

The implementation of a typical ALU instruction requires these steps:

- data are fetched from the Rcarrn the data bus;
- data are passed through the scaling shifter and through the ALU, where the arithmetic is performed; and
- the result is moved into the accumulator.

The 32-bit accumulator is split into two 16-bit segments for storage in data memory: ACCH (accumulator high) and ACCL (accumulator low). Shifters at the output of the accumulator provide a shift of 0 to 7 places to the left. This shift is performed while the data are being transferred to the data bus for storage. The contents of the accumulator remain unchanged. The accumulator also has an in-place one-bit shift to the left or right (SFL or SFR instruction) and a rotate through carry (ROL or ROR instruction) for shifting its contents.

A carry bit is provided to the accumulator, allowing more efficient extended-precision computation. ADDC (add with carry) and SUBB (subtract with borrow) are two instructions using the carry bit. Branch instructions that use the carry bit are also provided.

**Hardware multiplier.** The TMS320C25 uses a 16 × 16-bit hardware multiplier that can compute a 32-bit product during every machine cycle. Two registers are associated with the multiplier: a 16-bit temporary register (TR) that holds one of the operands for the multiplier, and a 32-bit product register (PR) that holds the product.

The output of the product register can be left-shifted one or four bits. This is useful for implementing fractional arithmetic or justifying fractional products. The output of the PR can also be right-shifted six bits to enable the execu-

tion of up to 128 consecutive multiply/accumulates without overflow.

The multiplier performs both signed and unsigned operations. Two signed instructions, MAC (multiply/accumulate) and MACD (multiply/accumulate and data move), can process both operands simultaneously, thereby fully utilizing the computational bandwidth of the multiplier. For MAC and MACD, the two operands are transferred to the multiplier at each cycle via the program and data buses. This enables MAC and MACD to be performed in a single cycle when they are used with repeat (RPT or RPTK) instructions. The program bus can supply data from internal or external memory (RAM or ROM) and still maintain single-cycle operation. An unsigned multiply (MPYU) instruction facilitates extended-precision multiplication. It multiplies the unsigned contents of the TR by the unsigned contents of the addressed data memory location, and places the result in the PR.

**Control operations.** Control operations are provided on the TMS320C25 by an on-chip timer, a repeat counter, three external maskable user interrupts, and internal interrupts generated by serial-port operations or by the timer.

A memory-mapped 16-bit timer (TIM) register (a down counter) is continuously clocked by CLKOUT1. A timer interrupt (TINT) is generated whenever the timer decrements to zero. The timer is reloaded with the value contained in the period (PRD) register within the first cycle after it reaches zero so that interrupts may be programmed to occur at regular intervals of (PRD + 1) • CLKOUT1 cycles. This feature is useful for control operations and for synchronous sampling of or writing to peripherals.

The repeat counter (RPTC) is loaded with either a data memory value (in the case of the RPT instruction) or an immediate value (in the case of the RPTK instruction). The repeat feature enables a single instruction to be executed up to 256 times. It can be used with instructions such as multiply/accumulates, block moves, I/O transfers, and table read/writes. Those instructions that are normally multicycle are pipelined when the repeat feature is used and effectively become single-cycle instructions. For example, the table read (TBLR) instruction ordinarily takes three or more cycles, but when it is repeated, it becomes a single-cycle instruction.

The three external maskable user interrupts, $\overline{\text{INT2}}$ to $\overline{\text{INT0}}$, enable external devices to interrupt the processor. Internal interrupts are generated by either the serial port, the timer, or the software interrupt instruction. Interrupts are prioritized, with reset having the highest priority and the serial-port transmit interrupt the lowest.

**Serial port.** An on-chip serial port provides direct communication with serial devices such as codecs and serial A/D and D/A converters. The serial port's interface requires a minimum of external hardware. The port has two memory-mapped registers—a data transmit register and a data receive register—which can be operated in either an eight-bit byte mode or a 16-bit word mode. The transmit framing sync pulse can be generated internally or externally. The serial port's maximum speed is 5 MHz.

The primary enhancements of the TMS320C25's serial port are

- double buffering for both receive and transmit operations,
- the elimination of a minimum CLKR/CLKX frequency (fmin = 0 Hz), and
- the provision of a frame sync mode (FSM) bit, which allows continuous operation with no frame sync pulses.

The FSM is useful for communicating on pulse-code-modulated telephone system highways. As a result the TMS320C25 can communicate directly on PCM highways such as AT&T T-1 and CCITT G.711/712 by counting the transmitted and received bytes in software and performing the instructions needed to set (SFSM) and reset (RFSM) the FSM bit.

**I/O interface.** The TMS320C25's I/O space consists of 16 input and 16 output ports. These ports provide a full 16-bit parallel I/O interface via the processor's data bus. A single input (IN) or output (OUT) operation typically takes two cycles; however, when executed in the repeat mode, such an operation becomes single-cycle. The TMS320C25 supports a range of system interfacing requirements. As previously mentioned, three separate address spaces—program, data, and I/O—provide interfacing to memory and I/O, thereby maximizing system throughput. The TMS320C25 simplifies I/O design by treating I/O the same way it treats memory. It maps I/O devices into the I/O address space using its external address and data buses in the same way as it uses them for mapping memory devices into memory address space.

The local memory interface consists of a 16-bit parallel data bus (D15-D0), a 16-bit address bus (A15-A0), three pins for data memory, program memory, and I/O space select ($\overline{\text{DS}}$, $\overline{\text{PS}}$, and $\overline{\text{IS}}$, respectively), and various system control signals. The R/$\overline{\text{W}}$ signal controls the direction of a data transfer, and $\overline{\text{STRB}}$ provides a timing signal to control the transfer. When using on-chip program RAM, ROM, or high-speed external program memory, the TMS320C25 runs at full speed without wait states. By using the READY signal, it can generate wait states so it can communicate with slower off-chip memories.

The TMS320C25 supports direct memory access to external program and data memory. Another processor can take complete control of the TMS320C25's external memory by asserting $\overline{\text{HOLD}}$ low, causing the TMS320C25 to place its address, data, and control lines in the high-impedance state. Two modes are available on the device. In the first mode, execution is suspended during assertion of $\overline{\text{HOLD}}$. In the second mode—the "concurrent DMA mode"—the TMS320C25 continues to execute its program while operating from internal RAM or ROM, thereby greatly increasing throughput in data-intensive applications. Signaling between the external processor and the TMS320C25 can be performed through interrupts.

## Table 1.
## TMS320C25 instructions.

| ACCUMULATOR MEMORY REFERENCE INSTRUCTIONS | | | |
|---|---|---|---|
| MNEMONIC | DESCRIPTION | NO. WORDS | OPERATION |
| ABS | Absolute value of accumulator | 1 | \|(ACC)\| → ACC |
| ADD | Add to accumulator with shift | 1 | (ACC) + \|(dma) × $2^{shift}$\| → ACC |
| ADDC‡ | Add to accumulator with carry | 1 | (ACC) + (dma) + (C) → ACC |
| ADDH | Add to high accumulator | 1 | (ACC) + \|(dma) × $2^{16}$\| → ACC |
| ADDK‡ | Add to accumulator short immediate | 1 | (ACC) + 8-bit constant → ACC |
| ADDS | Add to low accumulator with sign extension suppressed | 1 | (ACC) + (dma) → ACC |
| ADDT† | Add to accumulator with shift specified by T register | 1 | (ACC) + \|(dma) × $2^{(Treg)}$\| → ACC |
| ADLK† | Add to accumulator long immediate with shift | 2 | (ACC) + \|16-bit constant × $2^{shift}$\| → ACC |
| AND | AND with accumulator | 1 | (ACC(15-0)).AND.(dma) → ACC(15-0), 0 → ACC(31-16) |
| ANDK† | AND immediate with accumulator with shift | 2 | (ACC)30-0)).AND.\|16-bit constant × $2^{shift}$\| → ACC(30-0), 0 → ACC(30-0) |
| CMPL† | Complement accumulator | 1 | $(\overline{ACC})$ → ACC |
| LAC | Load accumulator with shift | 1 | (dma) × $2^{shift}$ → ACC |
| LACK | Load accumulator immediate short | 1 | 8-bit constant → ACC |
| LACT† | Load accumulator with shift specified by T register | 1 | (dma) × $2^{(Treg)}$ → ACC |
| LALK† | Load accumulator long immediate with shift | 2 | (16-bit constant) × $2^{16}$ → ACC |
| NEG† | Negate accumulator | 1 | −(ACC) → ACC |
| NORM† | Normalize contents of accumulator | 1 | |
| OR | OR with accumulator | 1 | (ACC(15-0)).OR.(dma) → ACC(15-0) |
| ORK† | OR immediate with accumulator with shift | 2 | (ACC(30-0)).OR.\|16-bit constant × $2^{shift}$\| → ACC(30-0) |
| ROL‡ | Rotate accumulator left | 1 | (ACC(30-0)) → ACC(31-1), (C) → ACC(0), (ACC(31)) → C |
| ROR‡ | Rotate accumulator right | 1 | (ACC(31-1)) → ACC(30-0), (C) → ACC(31), (ACC(0)) → C |
| SACH | Store high accumulator with shift | 1 | \|(ACC) × $2^{shift}$\| → dma |
| SACL | Store low accumulator with shift | 1 | \|(ACCL) × $2^{shift}$\| → dma |
| SBLK† | Subtract from accumulator long immediate with shift | 2 | (ACC) − \|16-bit constant × $2^{shift}$\| → ACC |
| SFL† | Shift accumulator left | 1 | (ACC(30-0)) → ACC(31-1), 0 → ACC(0) |
| SFR† | Shift accumulator right | 1 | (ACC(31-1)) → ACC(30-0), (ACC(31)) → ACC(31) |
| SUB | Subtract from accumulator with shift | 1 | (ACC) − \|(dma) × $2^{shift}$\| → ACC |
| SUBB‡ | Subtract from accumulator with borrow | 1 | (ACC) − (dma) − $(\overline{C})$ → ACC |
| SUBC | Conditional subtract | 1 | |
| SUBH | Subtract from high accumulator | 1 | (ACC) − \|(dma) × $2^{16}$\| → ACC |
| SUBK‡ | Subtract from accumulator short immediate | 1 | (ACC) − 8-bit constant → ACC |
| SUBS | Subtract from low accumulator with sign extension suppressed | 1 | (ACC) − (dma) → ACC |
| SUBT† | Subtract from accumulator with shift specified by T register | 1 | (ACC) − \|(dma) × $2^{(Treg)}$\| → ACC |
| XOR | Exclusive-OR with accumulator | 1 | (ACC(15-0)).XOR.(dma) → ACC(15-0) |
| XORK† | Exclusive-OR immediate with accumulator with shift | 2 | (ACC(30-0)).XOR.\|16-bit constant × $2^{shift}$\| → ACC(30-0) |
| ZAC | Zero accumulator | 1 | 0 → ACC |
| ZALH | Zero low accumulator and load high accumulator | 1 | (dma) × $2^{16}$ → ACC |
| ZALR‡ | Zero low accumulator and load high accumulator with rounding | 1 | (dma) × $2^{16}$ + >8000→ ACC |
| ZALS | Zero accumulator and load low accumulator with sign extension suppressed | 1 | (dma) → ACCL, 0 → ACCH |

†These instructions are not included in the TMS32010 instruction set.
‡These instructions are not included in the TMS32020 instruction set.

| AUXILIARY REGISTERS AND DATA PAGE POINTER INSTRUCTIONS | | | |
|---|---|---|---|
| MNEMONIC | DESCRIPTION | NO. WORDS | OPERATION |
| ADRK‡ | Add to auxiliary register short immediate | 1 | (ARn) + 8-bit constant → ARn |
| CMPR† | Compare auxiliary register with auxiliary register AR0 | 1 | If ARn \| CM \| AR0, then 1 → TC; else 0 → TC |
| LAR | Load auxiliary register | 1 | (dma) → (ARn) |
| LARK | Load auxiliary register short immediate | 1 | 8-bit constant → ARn |
| LARP | Load auxiliary register pointer | 1 | 3-bit constant → ARP, (ARP) → ARB |
| LDP | Load data memory page pointer | 1 | (dma) → DP |
| LDPK | Load data memory page pointer immediate | 1 | 9-bit constant → DP |
| LRLK† | Load auxiliary register long immediate | 2 | 16-bit constant → ARn |
| MAR | Modify auxiliary register | 1 | |
| SAR | Store auxiliary register | 1 | (ARn) → dma |
| SBRK‡ | Subtract from auxiliary register short immediate | 1 | (ARn) − 8-bit constant → ARn |

| T REGISTER, P REGISTER, AND MULTIPLY INSTRUCTIONS | | | |
|---|---|---|---|
| MNEMONIC | DESCRIPTION | NO. WORDS | OPERATION |
| APAC | Add P register to accumulator | 1 | (ACC) + (shift Preg) → ACC |
| LPH† | Load high P register | 1 | (dma) → Preg (31-16) |
| LT | Load T register | 1 | (dma) → Treg |
| LTA | Load T register and accumulate previous product | 1 | (dma) → Treg, (ACC) + (shifted Preg) → ACC |
| LTD | Load T register, accumulate previous product, and move data | 1 | (dma) → Treg, (dma) → dma + 1, (ACC) + (shifted Preg) → ACC |
| LTP† | Load T register and store P register in accumulator | 1 | (dma) → Treg, (shifted Preg) → ACC |
| LTS† | Load T register and subtract previous product | 1 | (dma) → Treg, (ACC) − (shifted Preg) → ACC |
| MAC† | Multiply and accumulate | 2 | (ACC) + (shifted Preg) → ACC, (pma) × (dma) → Preg |
| MACD† | Multiply and accumulate with data move | 2 | (ACC) + (shifted Preg) → ACC, (pma) × (dma) → Preg, (dma) → dma + 1 |
| MPY | Multiply (with T register, store product in P register) | 1 | (Treg) × (dma) → Preg |
| MPYA‡ | Multiply and accumulate previous product | 1 | (ACC) + (shifted Preg) → ACC, (Treg) × (dma) → Preg |
| MPYK | Multiply immediate | 1 | (Treg) × 13-bit constant → Preg |
| MPYS‡ | Multiply and subtract previous product | 1 | (ACC) − (shifted Preg) → ACC, (Treg) × (dma) → Preg |
| MPYU‡ | Multiply unsigned | 1 | Usgn (Treg) × Usgn (dma) → Preg |
| PAC | Load accumulator with P register | 1 | (shifted Preg) → ACC |
| SPAC | Subtract P register from accumulator | 1 | (ACC) − (shifted Preg) → ACC |
| SPH‡ | Store high P register | 1 | (shifted Preg (31-16)) → dma |
| SPL‡ | Store low P register | 1 | (shifted Preg (15-0)) → dma |
| SPM† | Set P register output shift mode | 1 | 2-bit constant → PM |
| SQRA† | Square and accumulate | 1 | (ACC) + (shifted Preg) → ACC, (dma) × (dma) → Preg |
| SQRS† | Square and subtract previous product | 1 | (ACC) − (shifted Preg) → ACC, (dma) × (dma) → Preg |

| SYMBOL | MEANING |
|---|---|
| ACC | Accumulator |
| ARB | Auxiliary register pointer buffer |
| ARn | Auxiliary Register n (AR0 through AR7 are predefined assembler symbols equal to 0 through 7 respectively) |
| ARP | Auxiliary register pointer |
| BIO | Branch control input |
| C | Carry bit |
| CM | 2-bit field specifying compare mode |
| CNF | On-chip RAM configuration control bit |
| dma | Data memory address |
| DP | Data page pointer |
| FO | Format status bit |
| FSM | Frame synchronization mode bit |
| HM | Hold mode bit |
| INTM | Interrupt mode flag bit |
| >nn | Indicates nn is a hexadecimal number (All others are assumed to be decimal values) |
| OV | Overflow flag bit |
| OVM | Overflow mode bit |
| P | Product register |

| SYMBOL | MEANING |
|---|---|
| PA | Port address (PA0 through PA15 are predefined assembler symbols equal to 0 through 15 respectively) |
| PC | Program counter |
| PM | 2-bit field specifying P register output shift code |
| pma | Program memory address |
| Preg | Product register |
| RPTC | Repeat counter |
| STn | Status Register n (ST0 or ST1) |
| SXM | Sign extension mode bit |
| T | Temporary register |
| TC | Test control bit |
| TOS | Top of stack |
| Treg | Temporary register |
| TXM | Transmit mode bit |
| Usgn | Unsigned value |
| XF | XF pin status bit |
| → | Is assigned to |
| \| \| | An absolute value |
| [ ] | Optional items |
| ( ) | Contents of |

## Table 1 cont'd

| BRANCH/CALL INSTRUCTIONS | | | |
|---|---|---|---|
| MNEMONIC | DESCRIPTION | NO. WORDS | OPERATION |
| B | Branch unconditionally | 2 | pma → PC |
| BACC† | Branch to address specified by accumulator | 1 | (ACC(15-0)) → PC |
| BANZ | Branch on auxiliary register not zero | 2 | If (AR(ARP)) ≠ 0, then pma → PC; else (PC) + 2 → PC |
| BBNZ† | Branch if TC bit ≠ 0 | 2 | If (TC) = 1, then pma → PC; else (PC) + 2 → PC |
| BBZ† | Branch if TC bit = 0 | 2 | If (TC) = 0, then pma → PC; else (PC) + 2 → PC |
| BC‡ | Branch on carry | 2 | If (C) = 1, then pma → PC; else (PC) + 2 → PC |
| BGEZ | Branch if accumulator ≥ 0 | 2 | If (ACC) ≥ 0, then pma → PC; else (PC) + 2 → PC |
| BGZ | Branch if accumulator > 0 | 2 | If (ACC) > 0, then pma → PC; else (PC) + 2 → PC |
| BIOZ | Branch on I/O status = 0 | 2 | If ($\overline{BIO}$) = 0, then pma → PC; else (PC) + 2 → PC |
| BLEZ | Branch if accumulator ≤ 0 | 2 | If (ACC) ≤ 0, then pma → PC; else (PC) + 2 → PC |
| BLZ | Branch if accumulator < 0 | 2 | If (ACC) < 0, then pma → PC; else (PC) + 2 → PC |
| BNC‡ | Branch on no carry | 2 | If (C) = 0, then pma → PC; else (PC) + 2 → PC |
| BNV† | Branch if no overflow | 2 | If (OV) ≠ 0, then pma → PC; else (PC) + 2 → PC |
| BNZ | Branch if accumulator ≠ 0 | 2 | If (ACC) ≠ 0, then pma → PC; else (PC) + 2 → PC |
| BV | Branch on overflow | 2 | If (OV) = 0, then pma → PC; else (PC) + 2 → PC |
| BZ | Branch if accumulator = 0 | 2 | If (ACC) = 0, then pma → PC; else (PC) + 2 → PC |
| CALA | Call subroutine indirect | 1 | (ACC(15-0)) → PC, (PC) + 1 → TOS |
| CALL | Call subroutine | 2 | (PC) + 2 → TOS, pma → PC |
| RET | Return from subroutine | 1 | (TOS) → PC |
| I/O AND DATA MEMORY OPERATIONS | | | |
| MNEMONIC | DESCRIPTION | NO. WORDS | OPERATION |
| BLKD† | Block move from data memory to data memory | 2 | (dma1, addressed by PC) → dma2 |
| BLKP† | Block move from program memory to data memory | 2 | (pma, addressed by PC) → dma |
| DMOV | Data move in data memory | 1 | (dma) → dma + 1 |
| FORT† | Format serial port registers | 1 | 1-bit constant → FO |
| IN | Input data from port | 1 | (data bus, addressed by PA) → dma |
| OUT | Output data to port | 1 | (dma) → data bus, addressed by PA |
| RFSM‡ | Reset serial port frame synchronization mode | 1 | 0 → FSM |
| RTXM† | Reset serial port transmit mode | 1 | 0 → TXM |
| RXF† | Reset external flag | 1 | 0 → XF |
| SFSM‡ | Set serial port frame synchronization mode | 1 | 1 → FSM |
| STXM† | Set serial port transmit mode | 1 | 1 → TXM |
| SXF† | Set external flag | 1 | 1 → XF |
| TBLR | Table read | 1 | (pma, addressed by ACC (15-0)) → dma |
| TBLW | Table write | 1 | (dma) → pma, addressed by ACC (15-0) |
| CONTROL INSTRUCTIONS | | | |
| MNEMONIC | DESCRIPTION | NO. WORDS | OPERATIONS |
| BIT† | Test bit | 1 | (dma bit at (15-bit code)) → TC |
| BITT† | Test bit specified by T register | 1 | (dma bit at (15-Treg)) → TC |
| CNFD† | Configure block as data memory | 1 | 0 → CNF |
| CNFP† | Configure block as program memory | 1 | 1 → CNF |
| DINT | Disable interrupt | 1 | 1 → INTM |
| EINT | Enable interrupt | 1 | 0 → INTM |
| IDLE† | Idle until interrupt | 1 | (PC) + 1 → PC, powerdown |
| LST | Load status register ST0 | 1 | (dma) → ST0 |
| LST1† | Load status register ST1 | 1 | (dma) → ST1 |

†These instructions are not included in the TMS32010 instruction set.
‡These instructions are not included in the TMS32020 instruction set.

| CONTROL INSTRUCTIONS cont'd | | | |
|---|---|---|---|
| NOP | No operation | 1 | (PC) + 1 → PC |
| POP | Pop top of stack to low accumulator | 1 | (TOS) → ACC |
| POPD† | Pop top of stack to data memory | 1 | (TOS) → dma |
| PSHD† | Push data memory value onto stack | 1 | (dma) → TOS |
| PUSH | Push low accumulator onto stack | 1 | (ACCL) → TOS |
| RC‡ | Reset carry bit | 1 | 0 → C |
| RHM‡ | Reset hold mode | 1 | 0 → HM |
| ROVM | Reset overflow mode | 1 | 0 → OVM |
| RPT† | Repeat instruction as specified by data memory value | 1 | (dma) → RPTC |
| RPTK† | Repeat instruction as specified by immediate value | 1 | 8-bit constant → RPTC |
| RSXM† | Reset sign-extension mode | 1 | 0 → SXM |
| RTC† | Reset test/control flag | 1 | 0 → TC |
| SC‡ | Set carry bit | 1 | 1 → C |
| SHM‡ | Set hold mode | 1 | 1 → HM |
| SOVM | Set overflow mode | 1 | 1 → OVM |
| SST | Store status register ST0 | 1 | ST0 → dma |
| SST1† | Store status register ST1 | 1 | ST1 → dma |
| SSXM† | Set sign-extension mode | 1 | 1 → SXM |
| STC‡ | Set test/control flag | 1 | 1 → TC |
| TRAP† | Software interrupt | 1 | (PC) + 1 → TOS, 30 → PC |

The TMS320C25's conditions and modes are stored in two status registers, ST0 and ST1. Instructions are provided to allow these registers to be stored in or loaded from data memory. This capability allows the current status of the device to be saved during interrupts and subroutine calls.

## TMS320C25 software

Earlier, we characterized digital signal processing as the real-time processing of mathematically intensive algorithms. This characterization equates to a requirement for high-speed, multiply/accumulate capability in a processor. The performance of a signal processor is therefore measured in terms appropriate to this requirement—that is, it is measured in terms of the speed of execution of individual instructions, the power of the instruction set, and the I/O capabilities. The speed is given as the basic instruction cycle time and the number of cycles required to complete any instruction.

As we noted earlier, pipelining of instruction fetching, decoding, and execution provides an instruction cycle time of only 100 ns. The overwhelming majority of the TMS320C25's instructions (97 out of 133) are executed in a single instruction cycle. Of the 36 instructions requiring additional cycles for execution, 21 involve branches, calls, and returns that result in a reload of the program counter and a break in the execution pipeline. Another seven of the instructions are two-word, long immediate instructions. The remaining eight—IN, OUT, BLKD, BLKP, TBLR, TBLW, MAC, and MACD—support I/O and transfers of data between memory spaces, or provide for additional parallel operation in the processor. Furthermore, these eight instructions become single-cycle when used in conjunction with the repeat counter. The instruction set of the TMS320C25 exploits the parallelism of the processor, allowing complex or numerically intensive computations to be implemented in relatively few instructions. Table 1 lists the TMS320C25's instructions.

**Addressing modes.** Most TMS320C25 instructions are coded in a single 16-bit word—the reason most can be executed in a single cycle. The 16-bit word comprises an eight-bit opcode and an eight-bit address. Three memory addressing modes are available: direct, indirect, and immediate (Table 2). Both direct and indirect addressing are used to access data memory. Immediate addressing uses the contents of the memory addressed by the program counter. Figure 4 illustrates operand addressing in the direct, indirect, and immediate modes.

In direct addressing, seven bits of the instruction word are concatenated with the nine-bit data memory page pointer (DP) to form the 16-bit data memory address. The DP register points to one of 512 possible data memory pages, each 128 word in length, to obtain a 64K total data memory space. The seven-bit address in the instruction

**Table 2. Addressing modes.**

| ADDRESSING MODE | OPERATION |
|---|---|
| OP A | Direct addressing. |
| OP *(NARP) | Indirect; no change to AR. |
| OP *+(NARP) | Indirect; current AR is incremented. |
| OP *-(NARP) | Indirect; current AR is decremented. |
| OP *0+(NARP) | Indirect; AR0 is added to current AR. |
| OP *0-(NARP) | Indirect; AR0 is subtracted from current AR. |
| OP *BR0+(NARP) | Indirect; AR0 is added to current AR (with reverse carry propagation). |
| OP *BR0-(NARP) | Indirect; AR0 is subtracted from current AR (with reverse carry propagation). |

NOTE: The optional NARP field specifies a new value of the ARP.

points to the specific location within the data memory page.

Indirect addressing is provided by the eight auxiliary registers AR0-AR7. These registers can be used to indirectly address data memory, as loop counters, or for temporary data storage. Indirect auxiliary register addressing (Figure 5) allows placement of the data memory address of an instruction operand into one of the eight auxiliary registers. These registers are pointed to by a three-bit auxiliary register pointer (ARP) that is loaded with a value from 0 through 7 designating AR0 through AR7, respectively. The auxiliary registers and the ARP may be loaded either from data memory or by an immediate operand defined in the instruction. Furthermore, the contents of the auxiliary registers may be stored in data memory.

There are seven types of indirect addressing (see Table 2 again):

- indexing with increment,
- indexing with decrement,
- indexing by adding the contents of AR0,
- indexing by subtracting the contents of AR0,
- indexing by adding the contents of AR0 with the carry propagation reversed (for bit-reversing an FFT),
- indexing by subtracting the contents of AR0 with the carry propagation reversed (also for bit-reversing an FFT), and
- no indexing.

All indexing operations are performed on the current auxiliary register in the same cycle as the original instruction, with loading of a new ARP value available as an option. The operations performed in the ARAU can even be performed during branch instruction execution, allowing efficient control with conditional looping.

Bit-reversed indexed addressing modes allow efficient I/O to be performed for the resequencing of data points in a radix-2 FFT program. The direction of carry propagation in the ARAU is reversed when this mode is selected, and AR0 is added to or subtracted from the current auxiliary register.

In immediate addressing, the instruction word contains the value of the immediate operand. Both single-word (8-bit and 13-bit constant) short immediate instructions and two-word (16-bit constant) long immediate instructions are included in the instruction set. In the case of long immediate instructions, the word following the instruction opcode is used as the immediate operand. MPYK is an example of an immediate instruction; it multiplies the contents of the T register by a signed 13-bit constant. Seventeen immediate operand instructions are included in the instruction set (see Table 1 again).

**Instruction set parallelism—an example.** The MACD (multiply/accumulate and data move) instruction serves as an informative example of the parallelism designed into the TMS320C25 instruction set as well as into the TMS320C25 architecture. As shown in Equation 1, the requirement for parallelism exists in common DSP operations such as convolution and filtering.[6,7]

Parallelism in the execution of instructions enables a complete multiply/accumulate/data move operation to be completed in a single 100-ns instruction cycle. The execution of the MACD involves the following steps:

1) The contents of the 32-bit P register are shifted (scaled) by an output shifter.

2) The 32-bit ALU accumulates the shifted result of the 32-bit P register with the current contents of the 32-bit accumulator.

3) The 16-bit contents of a data memory location (usually addressed indirectly via one of the auxiliary registers) are loaded into the T register.

4) The 16-bit contents of a program memory location (addressed via the prefetch counter PFC) are introduced to the multiplier and a 16 × 16-bit multiply is executed, resulting in a new 32-bit product. The product is placed in the P register to be accumulated during the next cycle.

5) The 16-bit contents of the data memory location are copied to the next higher data memory address.

6) The carry and overflow status bits are set, as appropriate, in the status registers.

7) The 16-bit contents of the auxiliary register pointed to by the ARP are modified (typically decremented) in preparation for the use of the data memory address on the next cycle.

8) The 16-bit contents of the PFC are incremented in preparation for the use of the program memory address on the next cycle.

9) The repeat counter is decremented.

As can be seen from the above, one of the data values is taken from data memory while the other is taken from program memory. A single-cycle execution and data move is accomplished when the data memory being addressed is the on-chip data memory. The program memory location can be either on or off chip and, if on chip, can come from either ROM or the reconfigurable memory block B0.

Parallel operation of certain subsets of TMS320C25 functions is also available. These subsets include loading the T register in combination with addition (LTA), subtraction (LTS), or a move of the P register's contents to the accumulator (LTP). The accumulation can be supplemented by the data move function (LTD). Another combination (MPYA/MPYS) provides the accumulation of the previous

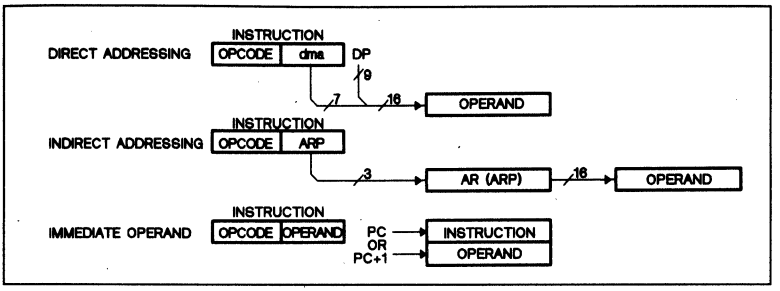



Figure 4. Methods of addressing the instruction operand.

product along with the execution of the multiplier to generate a new product. This combination is particularly useful in adaptive filtering techniques such as those embodied in the least-mean-square (LMS) algorithm.[4,15] The implementation of an adaptive filter by means of these instructions will be described in detail in the section on applications.

**Block moves.** The TMS320C25 provides six instructions for data and program block moves and transfers of data via the I/O ports. When these instructions are pipelined by means of the repeat instruction, significantly higher throughput is achieved—the pipelining results in a transfer rate of 160 million bits per second.

The BLKD instruction moves a block within data memory, and the BLKP instruction moves a block from program memory to data memory. Block transfers between program and data memory spaces can also be implemented with the TBLR and TBLW (table read and table write) instructions. The advantages of TBLR and TBLW are that they allow the source address as well as the destination address to be determined during programming and that they permit the data to be transferred from data memory to program memory. The IN and OUT instructions permit data to be transferred between the I/O and data memory spaces. While the source address is determined by the prefetch counter, which is incremented on every cycle, the destination address is determined by an auxiliary register whose contents can be modified in any of the previously specified ways. This permits sequential and contiguous data placement (*+, *−), sequential but noncontiguous data placement (*0+, *0−), or scrambled data placement (*BR0+, *BR0−). The value of these address modifications during block data transfers becomes particularly apparent in the use of indexing with reverse-carry propagation to set up the data block in an FFT. The result is not only a savings in execution time but a savings in program memory space as well.

**Floating-point support.** The TMS320C25 supports floating-point operations for applications requiring a large dynamic range. The NORM (normalization) instruction normalizes fixed-point numbers contained in the accumulator or by performing left shifts. The LACT (load accumulator with shift specified by the T register) instruction denormalizes a floating-point number by arithmetically left-shifting the mantissa through the input scaling shifter. The shift count, in this case, is the value of the exponent specified by the four low-order bits of the T register. ADDT and SUBT instructions (add to/subtract from accumulator with shift specified by the T register) have been provided to allow additional arithmetic operations.

## TMS320C25 hardware

The most important task for a hardware designer is interfacing the DSP device to the rest of the system as inexpensively as possible. Here, we will discuss the TMS320C25's interfacing capabilities.

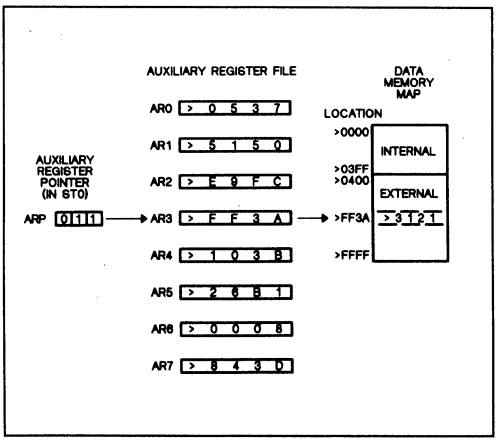



**Figure 5. Example of indirect auxiliary register addressing.**

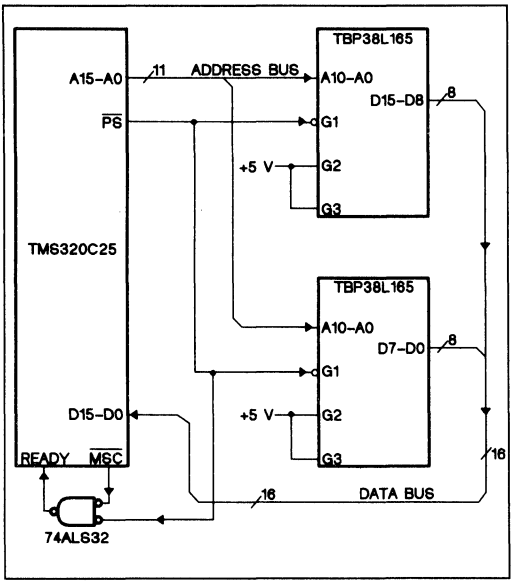



Figure 6. Minimal configuration for external program memory.

**System configurations.** The flexibility of the TMS320C25 allows systems configurations that satisfy a broad range of application requirements. The TMS320C25 can be configured as

- a stand-alone system (that is, as a single processor using 4K words of on-chip ROM and 544 words of on-chip RAM),
- part of a parallel multiprocessing system (two or more TMS320C25s) with shared global data memory, or
- a coprocessor for a host processor.

The stand-alone system interface consists of a 16-bit parallel data bus, a 16-bit address bus, three pins for memory space select, and various system control signals. In Figure 6, an external data RAM and a PROM/EPROM have been added to the basic stand-alone system. The READY signal is used for wait-state generation for communicating with slower off-chip memories. All the memories and I/O devices are directly controlled by the TMS320C25, thus minimizing external hardware requirements.

Parallel multiprocessing and host/coprocessor systems take advantage of the TMS320C25's direct memory access and global memory configuration capabilities.

**Direct memory access.** The TMS320C25 supports direct memory access to its external program/data memory and I/O space through its $\overline{\text{HOLD}}$ and $\overline{\text{HOLDA}}$ signals. Direct memory access can be used for multiprocessing: Execution on one or more processors can be temporarily halted to allow another processor to read from or write to the halted processor's local off-chip memory. Here the multiprocessing is typically performed in a master/slave configuration. The master can initialize the slave by downloading a program into its program memory space or provide the slave with the data needed to complete a task.

In a direct memory access scheme, the master may be a general-purpose CPU, a TMS320C25, or perhaps even an A/D converter. A master TMS320C25 takes complete control of the slave's external memory by asserting $\overline{\text{HOLD}}$ low through its external flag (XF). This causes the slave to place its address, data, and control lines in a high-impedance state. By asserting $\overline{\text{RS}}$ in conjunction with $\overline{\text{HOLD}}$, the master processor can load the slave's local program memory with the necessary initialization code on reset or power-up. The two processors can be synchronized through use of the $\overline{\text{SYNC}}$ pin to make the transfer over the memory bus faster and more efficient.

After control of the slave's buses is given to the master processor, the slave alerts the master by asserting $\overline{\text{HOLDA}}$. This signal can be tied to the master's $\overline{\text{BIO}}$ pin. The slave's XF pin can be used to indicate to the master when the slave has finished performing its task and needs to be reprogrammed or given additional data to continue processing. In a multiple-slave configuration, the priority of each slave's task can be determined by tying the slave's XF signals to the appropriate $\overline{\text{INT}}$ pin on the master.

A PC environment provides an example of a direct memory access scheme in which the system bus is used for data transfer. In this configuration, either the master CPU or a disk controller may place data on the system bus for downloading into the local memory of the TMS320C25. Here the TMS320C25 acts like a peripheral processor with multifunction capability. In a speech application, for example, the master can load the TMS320C25's program memory with algorithms to perform tasks such as speech analysis, synthesis, or recognition, and its data memory with the required speech templates. In a graphics application, the TMS320C25 can serve as a dedicated graphics engine. programs can be stored in ROM or downloaded via the system bus into program RAM. Again, data can come from PC disk storage or be provided directly by the master CPU. In this configuration, decode and arbitration logic is used to control the direct memory access. When the address on the system bus resides in the local memory of the peripheral TMS320C25, this logic asserts the $\overline{\text{HOLD}}$ signal while sending the master a not-ready indication to allow wait states. After the TMS320C25 acknowledges the direct memory access by asserting $\overline{\text{HOLDA}}$, READY is asserted and the information is transferred.

**Global memory.** In some digital signal processing tasks, the algorithm being implemented can be divided into sections and a processor dedicated to each. In this case, the

first and second processors can share global data memory, as can the second and third, the third and fourth, and so on. Arbitration logic may be required to determine which section of the algorithm will execute and which processor will have access to the global memory. The dedication of each processor to a distinct section of the algorithm makes pipelined execution—and thus higher throughput—possible.

External memory can be divided into global and local sections. Special registers and pins on the TMS320C25 allow multiple processors to share up to 32K words of global data memory. This facilitates efficient "shared data" multiprocessing, in which data are transferred between two or more processors. Unlike a direct memory access scheme, reading or writing global memory does not require one of the processors to be halted.

## TMS320C25 development tools and support

A digital signal processor is essentially an application-specific microprocessor or microcomputer. Like any microprocessor, it needs good development tools and technical support—no matter how impressive its performance or how easy its interfacing to other devices, it cannot be easily designed into systems without such tools and support. In developing an application, a designer encounters problems and needs to ask questions. Often the tools and vendor support given him are the difference between the success and failure of his project.

*No matter how impressive its performance or how easy its interfacing to other devices, a digital signal processor cannot be designed into systems without good development tools and vendor support.*

The TMS320C25 is supported by many development tools.[16] These tools range from inexpensive modules for application evaluation and benchmarking to an assembler/linker and software simulator to a full-capability hardware emulator.

**Software tools.** An assembler/linker and software simulator that enable users to develop and debug TMS320 DSP algorithms are available for the TI PC, IBM PC, and VAX.

*Assembler/linker.* The macro assembler translates assembly language source code into executable object code. It allows the programmer to work with mnemonics rather than hexadecimal machine instructions and to reference memory locations with symbolic addresses. It supports macro calls and definitions along with conditional assembly. The linker permits a program to be designed and implemented in separate modules that are later linked to form the complete program. The linker resolves external definitions and references for relocatable code, creating an object file that can be executed by the simulator, emulator, or the TMS320C25 processor. The macro assembler/linker is currently available for the VAX/VMS, TI PC/MS-DOS, and IBM PC/PC-DOS operating systems.

*Simulator.* The simulator is a software program that simulates TMS320 operations to allow program verification. Its debug mode enables the user to monitor the state of the simulated TMS320 while his program is executing. The simulator uses the object code produced by the macro assembler/linker. During program execution, the internal registers and memory of the simulated TMS320 are modified as each instruction is interpreted by the host computer. Once program execution is suspended, the internal registers and the program and data memories can be inspected and modified. The simulator is currently available for the VAX/VMS, TI PC/MS-DOS, and IBM PC/PC-DOS operating systems.

**Hardware tools.** Tools are provided for in-circuit emulation and hardware program debugging such as breakpointing and tracing so that DSP algorithms can be developed and tested in a real-product environment.

*Evaluation module.* The evaluation module, or EVM, is a stand-alone board that contains all the hardware tools needed to evaluate the TMS320C25 and that provides in-circuit emulation of it. The EVM's firmware package contains a debug monitor, an editor, an assembler, a reverse assembler, and software communication to two EIA ports. These ports allow the EVM to be connected to a terminal and to either a host computer or a line printer. The EVM accepts either source or object code downloaded from the host computer. Its resident assembler converts incoming source text into executable code in just one pass by automatically resolving labels after the first assembly pass is completed. When a session is finished, code is saved via the host computer interface.

*Software development system.* The SWDS is a plug-in card for the TI PC and IBM PC that provides the same functionality as the EVM.

*Emulator.* The XDS (Extended Development System) is an emulator providing full-speed in-circuit emulation with real-time hardware breakpointing and tracing and program execution capability from target memory. The XDS allows integration of hardware and software modules in the debug mode. By setting breakpoints based on internal conditions

or external events, the XDS user can suspend execution of the program and give control to the debug mode. In the debug mode, he can inspect and modify all registers and memory locations. Single-step execution is available. Full-trace capabilities at full speed and a reverse assembler that translates machine code back into assembly instructions also increase debugging productivity. The XDS system is designed to interface with either a terminal or a host computer. Object code generated by the assembler/linker can be downloaded to the XDS and then controlled through a terminal.

*Analog interface board.* The AIB is an analog-to-digital (A/D) and digital-to-analog (D/A) conversion board that can be used in conjunction with the EVM or XDS. It can also be used in an educational environment to help familiarize the user with real-world digital signal processing techniques. The AIB includes A/D and D/A converters with 12-bit resolution as well as antialiasing and smoothing filters that have a cut-off frequency programmable from 4.7 kHz to 20 kHz.

In addition to the above design tools, development support includes

- the Digital Filter Design Package, which runs on both TI and IBM PCs and which allows the user to design digital filters (low-pass, high-pass, band-pass, and band-stop types) using a menu-driven approach,
- TI Regional Technology Centers staffed with qualified engineers who provide technical support and design services,
- access to third parties with DSP expertise in various application areas,
- a series of DSP books covering DSP theory, algorithms, and applications and TMS320 implementations,[4,5,7]
- documentation such as user's guides,[10-12] data sheets, a development support reference guide,[16] and comprehensive application reports,[4] and
- a technical support hotline and a bulletin board service.

## TMS320C25 applications

The TMS320C25 is designed for real-time DSP and other computation-intensive tasks in telecommunications, graphics, image processing, high-speed control, speech processing, instrumentation, and numeric processing. In these applications, the TMS320C25 provides an excellent means for executing signal processing algorithms such as fast Fourier transforms (FFTs), digital filters, frequency synthesizers, correlators, and convolution routines. It can also execute general-purpose functions since it includes bit-manipulation instructions, block data move capabilities, large program and data memory address spaces, and flexible memory mapping.

Since digital filters are used in so many DSP applications, let us examine them as a prelude to our discussion of TMS320C25 applications.

**Digital filtering.** Filters are often implemented in digital signal processing systems. Such filters fall into two categories: finite impulse response (FIR) filters and infinite impulse response (IIR) filters.[4,6] For both types of filter, the coefficients of the filter (weighting factors) may be fixed or adapted during the course of the signal processing. The TMS320C25 reduces the execution time of all filters by virtue of its 100-ns instruction cycle time and optimized instructions for filter operations.

As we stated earlier, the FIR filter is simply the sum of products in a sampled data system (see Equation 1 again). A simple implementation of the FIR filter uses the MACD instruction (multiply/accumulate and data move) for each filter tap and the RPT/RPTK instruction to repeat the MACD for each tap. Thus, a 256-tap FIR filter can be implemented as

```
RPTK  255
MACD  *-,COEFFP
```

Here, the coefficients can be stored anywhere in program memory (in the reconfigurable on-chip RAM, in the on-chip ROM, or in external memories). When the coefficients are stored in on-chip ROM or externally, the entire on-chip data RAM can be used to store the sample sequence. This allows filters of up to 512 taps to be implemented. Execution of the filter will be at full speed, or 100 ns per tap, as long as the memory (either on-chip RAM or high-speed external RAM) supports full-speed execution.

Up to this point, we have assumed that the filter coefficients are fixed from sample to sample. If the coefficients are adapted or updated with time, as they are in adaptive filters for echo cancellation,[4,15] the DSP algorithm requires a greater computational capacity from the processor. To adapt or update the coefficients, usually with each sample, the TMS320C25 uses three instructions—multiply and add/substract previous product to/from accumulator (MPYA/MPYS), zero-out low-order accumulator bits and load high-order accumulator bits with data (ZALR), and store high-order bits of accumulator to data memory (SACH). The method it uses to adapt the coefficients is the least-mean-square, or LMS, algorithm, which can be expressed as

$$b_k(i+1) = b_k(i) + 2B\,[e(i) \cdot x(i-k)], \qquad (2)$$

where $b_k(i+1)$ is the weighting coefficient for the next sample period, $b_k(i)$ is the weighting coefficient for the present sample period, B is the gain factor or adaptation step size, $e(i)$ is the error function, and $x(i-k)$ is the input of the filter.

In an adaptive filter, the coefficients $b_k(i)$ must be updated to minimize the error function $e(i)$, which is the difference between the output of the filter and a reference signal. Quantization errors arising during coefficient updating can strongly affect the performance of the filter, but these errors can be minimized if the updated values are obtained by rounding rather than truncating. For each coefficient in the filter at a given point in time, the factor

$2*B*e(i)$ is a constant. This factor can be computed once and stored in the T register for each of the updates. This reduces the computational requirement to one multiply/accumulate plus rounding. Without the new instructions, the adaptation of each coefficient would take five instructions corresponding to five clock cycles, as the following instruction sequence shows:

```
LRLK  AR2,COEFFD ; LOAD ADDRESS OF COEFFICIENTS.
LRLK  AR3,LASTAP ; LOAD ADDRESS OF DATA SAMPLES.
LARP  AR2
LT    ERRF       ; errf = 2*B*e(i)
 .
 .
 .
ZALH  *,AR3      ; ACC = bk(i)*2**16
ADD   ONE,15     ; ACC = bk(i)*2**16 + 2**15
MPY   *-,AR2
APAC             ; ACC = bk(i)*2**16
                   + errf*x(i-k) + 2**15

SACH  *+         ; SAVE bk(i+1).
 .
 .
 .
```

When the MPYA and ZALR instructions are used, the adaptation reduces to three instructions corresponding to three clock cycles, as shown below:

```
LRLK  AR2,COEFFD ; LOAD ADDRESS OF COEFFICIENTS.
LRLK  AR3,LASTAP ; LOAD ADDRESS OF DATA SAMPLES.
LARP  AR2
LT    ERRF       ; errf = 2*B*e(i)
 .
 .
 .
ZALR  *,AR3      ; ACC = bk(i)*2**16 + 2**15
MPYA  *-,AR2     ; ACC = bk(i)*2**16
                   + errf*x(i-k) + 2**15
                 ; PREG = errf*x(i-k+1)
SACH  *+         ; SAVE bk(i+1).
 .
 .
 .
```

Note that the processing order has been slightly changed to incorporate the use of the MPYA instruction. This is due to the fact that the accumulation performed by the MPYA is the accumulation of the previous product.

We have now seen the basic code for a FIR filter tap and a coefficient update. Figure 7 shows a routine to filter a signal and update the coefficients for a 256-tap adaptive FIR filter. Note that for each tap one instruction cycle is needed to perform the FIR filter (i.e., to execute a MACD), three instruction cycles are needed to update the filter coefficients, and 33 instruction cycles are needed for overhead. Therefore, the total number of execution cycles needed for the routine is $33 + 4n$, where $n$ is the filter length. Also, note that data memory and program memory requirements are $5 + 2n$ and $30 + 3n$ words, respectively. For adaptive filters, the filter length is restricted by both execution time and memory. There is obviously more processing to be completed per sample due to the adaptation, and the adaptation

```
        TITL  'ADAPTIVE FILTER'
        DEF   ADPFIR
        DEF   X,Y
*
* THIS 256-TAP ADAPTIVE FIR FILTER USES ON-CHIP MEMORY BLOCK
* B0 FOR COEFFICIENTS AND BLOCK B1 FOR DATA SAMPLES. THE
* NEWEST INPUT SHOULD BE IN MEMORY LOCATION X WHEN CALLED.
* THE OUTPUT WILL BE IN MEMORY LOCATION Y WHEN RETURNED.
* ASSUME THAT THE DATA PAGE IS 0 WHEN THE ROUTINE IS CALLED.
*
COEFFP EQU   >FF00       ; B0 PROGRAM MEMORY ADDRESS
COEFFD EQU   >0200       ; B0 DATA MEMORY ADDRESS
*
ONE    EQU   >7A         ; CONSTANT ONE          (DP=0)
BETA   EQU   >7B         ; ADAPTATION CONSTANT (DP=0)
ERR    EQU   >7C         ; SIGNAL ERROR          (DP=0)
ERRF   EQU   >7D         ; ERROR FUNCTION      (DP=0)
Y      EQU   >7E         ; FILTER OUTPUT       (DP=0)
X      EQU   >7F         ; NEWEST DATA SAMPLE  (DP=0)
FRSTAP EQU   >0300       ; NEXT NEWEST DATA SAMPLE
LASTAP EQU   >03FF       ; OLDEST DATA SAMPLE
*
* FINITE IMPULSE  RESPONSE (FIR) FILTER.
*
ADPFIR CNFP              ; CONFIGURE B0 AS PROGRAM:
       MPYK  0           ; Clear the P register.
       LAC   ONE,14      ; Load output rounding bit.
       LARP  AR3
       LRLK  AR3,LASTAP  ; Point to the oldest sample.
FIR    RPTK  255
       MACD  COEFFP,*-   ; 256-tap FIR filter.
       CNFD              ; CONFIGURE B0 AS DATA:
       APAC
       SACH  Y,1         ; Store the filter output.
       NEG
       ADD   X,15        ; Add the newest input.
       SACH  ERR,1       ; err(i) = x(i) - y(i)
*
* LMS ADAPTATION OF FILTER COEFFICIENTS.
*
       LT    ERR
       MPY   BETA
       PAC               ; errf(i) = beta * err(i)
       ADD   ONE,14      ; ROUND THE RESULT.
       SACH  ERRF,1
*
       MAR   *+
       LAC   X           ; INCLUDE NEWEST SAMPLE.
       SACL  *
*
       LRLK  AR2,COEFFD  ; POINT TO THE COEFFICIENTS.
       LRLK  AR3,LASTAP  ; POINT TO THE DATA SAMPLES.
       LT    ERRF
       MPY   *-,AR2      ; P = 2*beta*err(i)*x(i-255)
*
ADAPT  ZALR  *,AR3       ; LOAD ACCH WITH b255(i) & ROUND.
       MPYA  *-,AR2      ; b255(i+1) = b255(i) + P
*                        ; P = 2*beta*err(i)*x(i-254)
       SACH  *+          ; STORE b255(i+1).
*
       ZALR  *,AR3       ; LOAD ACCH WITH b254(i) & ROUND.
       MPYA  *-,AR2      ; b254(i+1) = b254(i) + P
*                        ; P = 2*beta*err(i)*x(i-253)
       SACH  *+          ; STORE b254(i+1).
*
       ZALR  *,AR3       ; LOAD ACCH WITH b253(i) & ROUND.
       MPYA  *-,AR2      ; b253(i+1) = b253(i) + P
*                        ; P = 2*beta*err(i)*x(i-252)
       SACH  *+          ; STORE b253(i+1).
        .
        .
        .
*
       ZALR  *,AR3       ; LOAD ACCH WITH b1(i) & ROUND.
       MPYA  *-,AR2      ; b1(i+1) = b1(i) + P
*                        ; P = 2*beta*err(i)*x(i-0)
       SACH  *+          ; STORE b1(i+1).
*
       ZALR  *,AR3       ; LOAD ACCH WITH b0(i) & ROUND.
       APAC  *-,AR2      ; b0(i+1) = b0(i) + P
       SACH  *+          ; STORE b0(i+1).
*
       RET               ; RETURN TO CALLING ROUTINE.
```

**Figure 7. 256-tap adaptive FIR filter routine.**

itself dictates that the coefficients be stored in the reconfigurable block of on-chip RAM. Thus, an adaptive filter with no external data memory is limited to 256 taps.

**Telecommunications applications.** Digital signal processing will be more extensively used in telecommunications as it evolves toward all-digital networks.[17] Below, we discuss several typical uses of the TMS320C25 in telecommunications applications.

*Echo cancellation.* In echo cancellation, an adaptive FIR filter performs the modeling routine and signal modifications needed to adaptively cancel the echo caused by impedance mismatches in telephone transmission lines. The TMS320C25's large on-chip RAM of 544 words and on-chip ROM of 4K words allow it to execute a 256-tap adaptive filter (32-ms echo cancellation) without external data or program memory.

*High-speed modems.* For high-speed modems, the TMS320C25 can perform functions such as modulation and demodulation, adaptive equalization, and echo cancellation.[18,19]

*Voice coding.* Voice-coding techniques such as full-duplex, 32,000-bit-per-second adaptive differential pulse-code modulation (CCITT G.721), CVSD, 16,000-bit-per-second subband coding, and linear predictive coding are frequently used in voice transmission and storage. The speed of the TMS320C25 in performing arithmetic and its normalization and bit-manipulation capabilities enable it to implement these functions, usually within itself (i.e., with no external devices).

**Graphics and image processing applications.** In these applications, a signal processor's ability to interface with a host processor is important. The TMS320C25 multiprocessor interface enables it to be used in a variety of host/coprocessor configurations. Graphics and image processing applications can use the TMS320C25's large directly addressable external data space and global memory capability to allow graphical images in memory to be shared with a host processor, thus minimizing data transfers. The TMS320C25's indexed indirect addressing modes allow matrices to be processed row by row when matrix multiplication is performed for 3-D image rotation, translation, and scaling.

**High-speed control applications.** These applications use the TMS320C25's general-purpose features for bit-test and logical operations, timing synchronization, and fast data transfers (10 million 16-bit words per second). They use the TMS320C25 in closed-loop systems for control signal conditioning, filtering, high-speed computing, and multichannel multiplexing. The following examples demonstrate typical control applications.

*Disk control.* In disk drives, a closed-loop actuation mechanism positions the read/write heads over the disk surface. Accurate positioning requires various signal conditioning tasks to be performed. The TMS320C25 can replace costly bit-slice, custom, and analog solutions in performing such tasks as compensation, filtering, and fine/coarse tuning.

*Robotics.* The TMS320C25's digital signal processing and bit-manipulation power, coupled with its host interface, allow it to be useful in robotics control. The TMS320C25 can replace both the digital controllers and the analog signal processing hardware a robot needs to communicate to a central host processor, and it can perform the numerically intensive control functions typical of robotic applications.

**Instrumentation.** Instruments such as spectrum analyzers often require a large data memory space and a processor capable of performing long-length FFTs and generating high-precision functions with minimal external hardware. The TMS320C25 fulfills these requirements.

**Numeric processing applications.** Numeric and array processing applications benefit from the TMS320C25's performance. The device's high throughput and its multiprocessing and data memory expansion capabilities make it a low-cost, easy-to-use replacement for a typical bit-slice array processor.

**Benchmarks.** The TMS320C25 has demonstrated impressive performance of benchmarks representing common DSP routines and applications. Table 3 shows this performance.

The TMS320C25 digital signal processor is the newest member of the TMS320 family. It is a pin-compatible, CMOS version of the TMS32020 but offers several enhancements of that device—a 100-ns instruction cycle time, 4K words of on-chip masked ROM, eight auxiliary registers, an eight-level hardware stack, and a double-buffered serial port. It also enhances the TMS32020 instruction set to support adaptive filtering, extended-precision arithmetic, bit-reversed addressing, and faster I/O.

The TMS320C25's multiprocessor capability, large memory spaces, and general-purpose features allow it to be used in a variety of systems, including ones currently employing costly bit-slice processors or custom ICs. ■

Table 3.
TMS320C25 benchmarks.

| DSP ROUTINES/APPLICATIONS | PERFORMANCE |
|---|---|
| FIR filter tap | 100 ns per tap |
| 256-tap FIR filter sample rate | 37 kHz |
| LMS adaptive FIR filter tap | 400 ns per tap |
| 256-tap adaptive FIR filter sample rate | 9.5 kHz |
| Biquad filter element | 1 μs |
| Echo canceller | 32 ms per single chip (with internal memory) |
| 32,000-bit/s CCITT ADPCM | 1 channel full-duplex, single-chip (with internal memory) |
| 16,000-bit/s subband coding | 2 channels full-duplex, single-chip (with 0.5K external data memory) |
| 2400-bit/s LPC-10 coding | 2 channels full-duplex, single-chip (with 2K external data memory) |

## References


1. L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, N.J., 1975.
2. A. V. Oppenheim, ed., *Applications of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, N.J., 1978.
3. L. R. Rabiner and R. W. Schafer, *Digital Processing of Speech Signals*, Prentice-Hall, Englewood Cliffs, N.J., 1978.
4. *Digital Signal Processing Applications with the TMS320 Family*, Texas Instruments Inc., 1986.
5. R. Morris, *Digital Signal Processing Software*, DSPS Inc., Ottawa, Ont., 1983.
6. A.V. Oppenheim and R.W. Schafer, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, N.J., 1975.
7. C. Burrus and T. Parks, *DFT/FFT and Convolution Algorithms*, John Wiley & Sons, New York, 1985.
8. K. McDonough, E. Caudel, S. Magar, and A. Leigh, "Microcomputer with 32-bit Arithmetic Does High-Precision Number Crunching," *Electronics*, Feb. 24, 1982, pp. 105-110.
9. S. Magar, E. Caudel, and A. Leigh, "A Microcomputer with Digital Signal Processing Capability," *Digest of Tech. Papers—1982 IEEE Int'l Solid-State Circuits Conf.*, pp. 32-33 and 284-285.
10. *TMS32010 User's Guide*, Texas Instruments Inc., 1983.
11. *TMS320C25 User's Guide*, Texas Instruments Inc., 1986.
12. *TMS32020 User's Guide*, Texas Instruments Inc., 1985.
13. H. G. Cragon, "The Elements of Single-Chip Microcomputer Architecture," *Computer*, Vol. 13, No. 10, Oct. 1980, pp. 27-41.
14. S. Rosen, "Electronic Computers: A Historical Survey," *Computing Surveys*, Vol. 1, No. 1, Mar. 1969.
15. M. Honig and D. Messerschmitt, *Adaptive Filters*, Kluwer Academic Publishers, Hingham, Mass., 1984.
16. *TMS320 Family Development Support Reference Guide*, Texas Instruments Inc., 1986.
17. M. Bellanger, "New Applications of Digital Signal Processing in Communications," *IEEE ASSP Magazine*, July 1986, pp. 6-11.
18. R. Lucky et al., *Principles of Data Communication*, McGraw-Hill, New York, 1965.
19. P. Van Gerwen et al., "Microprocessor Implementation of High Speed Data Modems," *IEEE Trans. Communications*, Vol. COM-25, 1977, pp. 238-249.

**Gene A. Frantz** has been Texas Instruments' applications manager for digital signal processing products since 1984. He is also a senior member of the Technical Staff at TI. He joined TI in 1974 as a system design engineer and worked on calculators in TI's Consumer Products Division. In 1976 he was assigned to the Li'l Professor design team. He was next assigned to the Speak & Spell project, where he served as program manager. Since then, he has been involved with every speech-related consumer product developed at TI.

Frantz received a BSEE from the University of Central Florida in 1971, an MSEE from Southern Methodist University, and an MBA from Texas Tech University.

**Kun-Shan Lin** has been involved in digital signal processing applications in the TI Semiconductor Group since 1984. He is a senior member of the TI Technical Staff. He joined Texas Instruments in 1979 and was assigned to the Consumer Products Division, where he developed speech techniques for learning aids. Prior to joining TI, he was an assistant professor of electrical engineering at Tennessee State University and an adjunct assistant professor of EE at the University of New Mexico. Lin received his PhD from the University of New Mexico in 1976.

**Jay Reimer**, a member of the TI Technical Staff, handles DSP applications engineering for the TMS320 family of products. He joined TI in 1979 to work with speech products in the company's Consumer Products Division. In 1984, he transferred to the Semiconductor Group to work with digital signal processors. His responsibilities include software development for the TMS320 family and applications assistance for customers using the processors. Reimer received a BS in physics from Fort Hays State University, Kansas, in 1975 and an MS in physics from the University of Kansas in 1977.

**Jon Bradley** is an applications engineer for the TMS320 family. He joined Texas Instruments in 1976 and has been an applications engineer for most of TI's microprocessor and peripheral products, starting with the TMS9900 family. His responsibilities have included microprocessor system design, digital and analog circuit design, integrated circuit design, test engineering, and programming. Bradley received a BSEE from Worcester Polytechnic Institute, Massachusetts, in 1976.

# Part II. Digital Signal Processing Interface Techniques

4. Hardware Interfacing to the TMS320C2x
   (George Troullinos and Jon Bradley)
5. Interfacing the TMS320 Family to the TLC32040 Family
   (Linear Products — Texas Instruments)
6. $I_{CC}$ Requirements of a TMS320C25
   (Dave Zalac)
7. An Implementation of a Software UART Using the TMS320C25
   (Dave Zalac)
8. TMS320C17 and TMS370C010 Serial Interface
   (Peter Robinson)

# Hardware Interfacing to the TMS320C2x

George Troullinos
Jon Bradley

Digital Signal Processor Products — Semiconductor Group
Texas Instruments

## Introduction

Each member of the TMS320 Second-Generation Digital Signal Processors family has the power and flexibility to satisfy a wide range of system requirements. The second-generation TMS320 line includes the TMS32020, TMS320C25, TMS320C25-50, TMS320E25, and TMS320C26. Please refer to the Second-Generation TMS320 User's Guide[1] for details on device-to-device variation.

All TMS320 second-generation DSPs are pin-compatible and thus have the same set of external interface signals. For convenience, the following notation will be used throughout this report: Second-generation TMS320 devices refer to all members of this family, TMS320C2x refers to all members of the second-generation family except the TMS32020 (i.e., TMS320C25, TMS320C25-50, TMS320E25, and TMS320C26). In other TI literature, TMS320C2x normally refers to the entire second-generation family. This report will focus on TMS320C2x hardware interfacing.

All second-generation TMS320 devices can address 64K 16-bit words in data space, 64K words in program space, and 16 16-bit wide I/O ports. The 128K-word address space for program and data memory can be utilized in applications that require large amounts of memory by interfacing external memories using the control signals of second-generation TMS320 devices. In other applications, the internal program and data resources of second-generation TMS320 devices can be used to implement single-chip solutions. Peripheral devices can be interfaced to second-generation TMS320 devices to perform analog signal acquisition at different levels of signal quality.

This report suggests hardware design techniques for interfacing memories and peripherals to the TMS320C2x. Differences between the TMS320C2x and the TMS32020 are pointed out when appropriate. The first section presents the design interfaces of PROMs, EPROMs, and static RAMs (SRAM) to the TMS320C2x. Timing requirements of the processor and external memories are considered. The second section discusses the interface of a combo-codec (PCM coder-decoder), an analog-to-digital converter, and a digital-to-analog converter to the TMS320C2x. All interfaces in this report have been built and tested to verify their operation.

## Ready Generation Techniques

This section describes techniques for generating the READY input signal for the TMS320C2x. READY can be used to extend external bus cycles by an integer number of machine cycles. The READY input thereby provides a means of interfacing the TMS320C2x to external devices that cannot be accessed at full speed, such as memory devices having access times longer than those required by the TMS320C2x.

The access time ($t_a$) of a given device determines the number of dormant cycles (wait-states) required for each access of that device. In general, N wait-states are required for a particular access if

$$[\, t_{c(C)} * (N-1) + t_{a(A)} \,] < t_a < [\, t_{c(C)} * N + t_{a(A)} \,] \,, \; N > 0$$

where $t_{c(C)}$ is the period of CLKOUT1/2 (the reciprocal of the machine rate) and $t_{a(A)}$ is the access time from address specified in the appropriate second-generation TMS320 device electrical specification, Table 1 gives appropriate values of N for several ranges of $t_a$ for a TMS320C25 operating

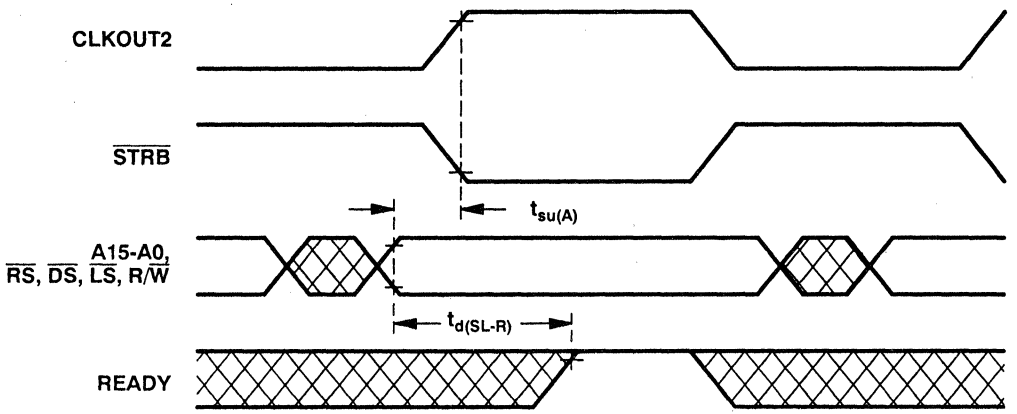
with a 100 ns instruction cycle time and a TMS320C25-50 operating with a 80 ns instruction cycle time.

**Table 1. Number of Wait-States Required for a Memory or Peripheral Access**

| TMS320C25 | | TMS320C25-50 | |
|---|---|---|---|
| Access Time | Number of Wait States Required | Access Time | Number of Wait States Required |
| $t_a$ < 40 ns | 0 | ta < 29 ns | 0 |
| 40 ns < $t_a$ < 140 ns | 1 | 29 ns < $t_a$ < 109 ns | 1 |
| 140 ns < $t_a$ < 240 ns | 2 | 109 ns < $t_a$ < 189 ns | 2 |
| 240 ns < $t_a$ < 340 ns | 3 | 189 ns < $t_a$ < 269 ns | 3 |
| 340 ns < $t_a$ < 440 ns | 4 | 269 ns < $t_a$ < 349 ns | 4 |

The timing requirements for generation of the READY signal are specified in the TMS320C25 electrical specifications by $t_{su(A)}$ and $t_{d(SL-R)}$ or $t_{d(C2H-R)}$.

**Figure 1. Ready Timing Requirement**




READY (see Figure 1) must be valid no later than $t_{su(A)} + t_{d(SL-R)}$ after the address bus and interface control signals (except $\overline{STRB}$) become valid. This evaluates to

$$t_{su(A)} + t_{d(SL-R)} = (Q–11) + (Q–20) = 9 \text{ ns}$$

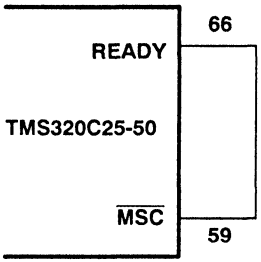
for a TMS320C25-50 operating with an input clock frequency of 50.0 MHz, and

$$t_{su(A)} + t_{d(SL-R)} = (Q–12) + (Q–20) = 18 \text{ ns}$$

for a TMS320C25 operating with an input clock frequency of 40.0 MHz. Note that for bus cycles with wait-states, CLKOUT2 serves as the timing reference, whereas for no-wait cycles either $\overline{STRB}$ or CLKOUT2 can be used as the timing reference. Any skew between these two signals may be disregarded as $t_{d(SL-R)}$ and $t_{d(C2H-R)}$ are guaranteed independently.

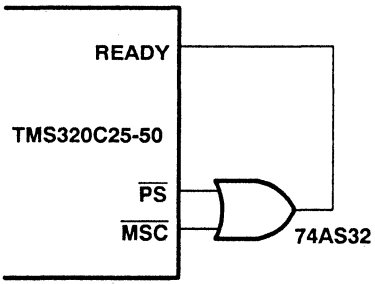
If all external bus cycles are to occur with no wait-states, READY can simply be tied high with a pull-up resistor. Extending all external bus cycles with one wait-state can easily be accomplished by connecting the $\overline{\text{MSC}}$ output to READY as shown in Figure 2.

**Figure 2. Connection for One Wait-State External Accesses**




Similarly, $\overline{\text{MSC}}$ and the $\overline{\text{PS}}$, $\overline{\text{DS}}$, and $\overline{\text{IS}}$ signals can be used to generate wait-state mixes such as that resulting from the circuit in Figure 3. With this circuit, all program space accesses are one wait-state accesses while all data space and I/O accesses occur at full speed.

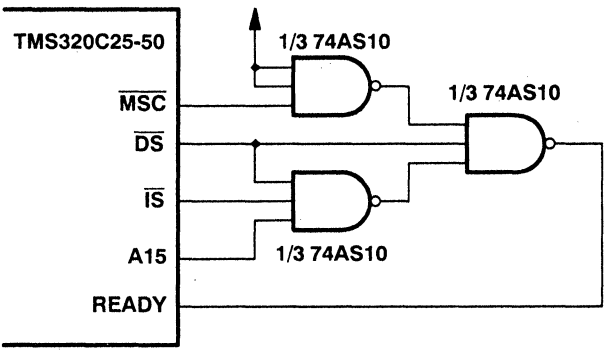
**Figure 3. Ready Generation for One Wait-State Program Space Accesses**



Applications having sufficiently simple address partitioning can make use of one or more levels of standard logic gates to generate READY. The circuit shown in Figure 4 has the following wait-state map:

| External Space | Address Range | Number of Wait-States |
|---|---|---|
| Program | 0000h–7FFFh | 1 |
| Program | 8000h–FFFFh | 0 |
| Data | 0000h–FFFFh | 0 |
| I/O | 0000h–000Fh | 1 |

**Figure 4. Ready Generator with Simple Address Partitioning**




Note that this circuit just meets the READY specification of the TMS320C25-50 with READY guaranteed valid no later than 9 ns from address valid.

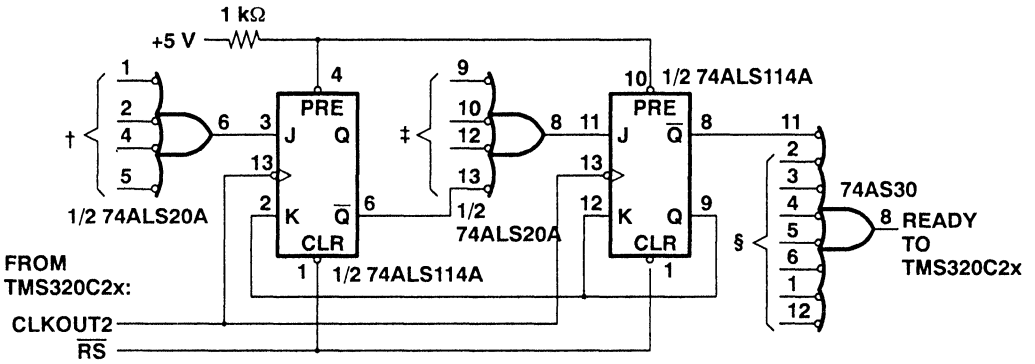
TMS320C25-50 applications requiring more extensive address decoding will in most cases require the use of a high-speed programmable logic device to generate READY sufficiently fast. Two such devices are listed in Table 2.

**Table 2. High-Speed Programmable Logic Devices**

| Manufacturer | Part Number | $t_{pd}$ (ns) |
|---|---|---|
| TI | TIBPAL16L8-7 | 7.5 |
| AMD | PAL16L8-7 | 7.5 |

The wait-state generator shown in Figure 5 can be used to generate the READY signal for a TMS320C25 interfaced to external devices requiring up to 2 wait-states. A timing diagram for this circuit is shown in Figure 6.

## Figure 5. Two Wait-State Generator Design

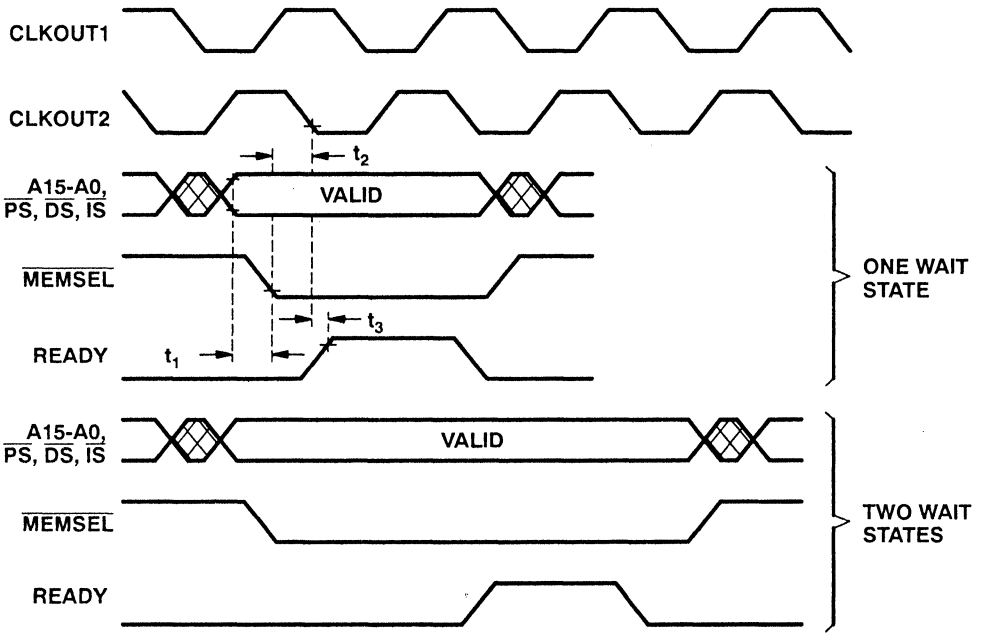



† Connections to other devices in the system that require two wait states. (Inputs not used by other devices should be pulled up.)

‡ Connections to other devices in the system that require one wait state. (Inputs not used by other devices should be pulled up.)

§ Connections to other devices in the system that require zero wait states. (Inputs not used by other devices should be pulled up.)

## Figure 6. Timing Diagram for Two Wait-State Generator Design




With this arrangement, READY is driven by a multiple-input NAND gate. This can be a standard gate such as a 74AS30 or can be part of the logic implemented by a high-speed programmable logic device. The output of this gate is low unless at least one of the inputs is low. The propagation delay of READY decode logic selecting zero wait-state devices in addition to the NAND delay

must be short enough to satisfy the READY specification discussed above. For zero wait-state accesses, the flip-flop J inputs are low, the $\overline{Q}$ outputs are high and neither flip-flop switches state.

Now consider the circuit operation when a one or two wait-state device is selected. The $\overline{Q}$ output of each JK flip-flop is high at the start of the access, which can be considered to begin with the falling edge of CLKOUT2. All the inputs to the NAND gate generating READY are high and thus READY is low during the first cycle and the TMS320C25 inserts one wait-state. If a one wait-state device is decoded, the J input of the first flip-flop goes high. The Q output goes low on the next falling edge of CLKOUT2 and READY goes high.

If a two wait-state device is decoded, the J input of the second flip-flop goes high. Two cycles are required for this signal to propagate to the READY line. For each cycle, one wait-state is inserted.

Referring to Figure 6, the following two inequalities must be satisfied in order for the setup time specification of the flip-flops to be met:

1) $t_{(decode)} + t_{(NAND)} + t_{su(74ALS114A)} < t_{su(A)} + 2Q$
2) $t_{p(74ALS114A)} + t_{(NAND)} + t_{su(74ALS114A)} < 4Q$

where $t_{(decode)}$ is the propagation delay of the decode logic for the selected device, $t_{(NAND)}$ is the delay associated with the NAND gate at the flip-flop input, $t_{su(74ALS114A)}$ and $t_{p(74ALS114A)}$ are the data setup time and prop delay of the 74ALS114A, respectively, and $Q = 1/4t_{c(C)}$. In Figure 6,

$t_1 = t_{(decode)}$
$t_2 = t_{(NAND)} + t_{su(74ALS114A)}$ and
$t_3 = t_{p(74ALS114A)} + t_{(NAND)}$.

A third inequality must be satisfied for the READY specification to be met:

3) $t_{p(74ALS114A)} + t_{(NAND)} < t_{d(C2H-R)} + 2Q$

For a TMS320C25-50 operating at 50 MHz, inequality (1) evaluates to

1) $t_{(decode)} + 5\text{ ns} + 22\text{ ns} < 9\text{ ns} + 40\text{ ns}$

or

$t_{(decode)} < 22\text{ ns}$

This inequality specifies the maximum decode time in order for the setup time specification of the pertinent flip-flop to be met.

The remaining two inequalities are satisfied:

2) $19\text{ns} + 5\text{ns} + 22\text{ns} < 80\text{ns}$
3) $19\text{ns} + 5\text{ns} < 0\text{ns} + 40\text{ns}$

All three of these inequalities should be considered if different flip-flops and/or gates are used to implement the wait-state generator.

Note that special considerations should be made with respect to READY timing if the TI Extended Development Support (XDS) in-circuit emulator is used. Please refer to *TMS320 Second-Generation User's Guide*[1] and/or *Extended Development Support Products User's Guide* (literature number SPYF001) for further details on READY timing requirements.

# Interfacing Memories to the TMS320C25

This section describes interfaces of external memory devices to the 40 MHz speed version of the TMS320C25. Interfaces to PROMS, EPROMs, and SRAMS are included. A separate section is included in this document to describe memory interfaces to the TMS320C25-50.

The TMS320C2x offers 544 words of RAM and 4K words of masked ROM. For prototyping and/or system expansion, however, external memories may be required. The speed, cost, and power limitations imposed by a particular application determine the selection of a specific memory device. If speed and maximum throughput are desired, the TMS320C2x can run with no wait-states. In this case, memory accesses are performed in a single machine cycle. Alternatively, slower memories can be accessed by introducing an appropriate number of wait-states or by slowing down the system clock. The latter approach is more appropriate when interfacing to memories with access times slightly longer than those required by the TMS320C2x at full speed.

When wait-states are required, the number of wait-states depends on the memory access time (see Table 1 on page 2). With no wait-states, the READY input to the TMS320C2x can be pulled high. If one or more wait-states are required, the READY input must be driven low during the cycles in which the TMS320C2x enters a wait-state.

The TMS320C2x implements two separate and distinct memory spaces: program space (64K words) and data space (64K words). Distinction between the two spaces is made through the use of the PS (program space) and DS (data space) pins. A third space, the I/O space, is also available for interfacing with peripherals. This space is selected by the IS (I/O space) pin, and is discussed in the Interfacing Peripherals section of this report.

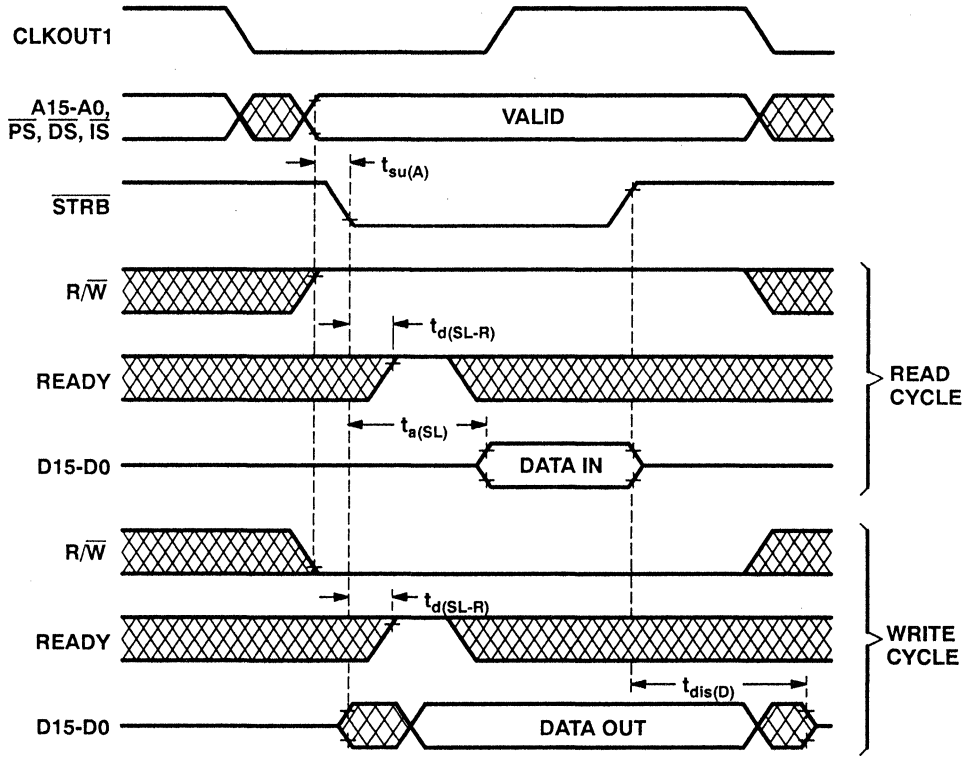
The following brief discussion describes the TMS320C2x read and write cycles. A more complete discussion is contained in the *Second-Generation TMS320 User's Guide.*[1] Throughout this report, Q is used to indicate the duration of a quarter-phase of the output clock (CLKOUT1 or CLKOUT2). Memory interfaces discussed in this report assume that the TMS320C2x is running at 40 MHz; i.e., Q = 25 ns. The memory read and write timings are shown in Figure 7. In a read cycle, the following sequence occurs:

1) Near the beginning of the machine cycle (CLKOUT1 goes low), the address bus and one of the memory select signals ($\overline{PS}$, $\overline{DS}$, or $\overline{IS}$) becomes valid. R/W goes high to indicate a read cycle.
2) $\overline{STRB}$ goes low in not less than $t_{su(A)} = (Q - 12)$ ns after the address bus becomes valid.
3) Early in the second half of the cycle, the READY input is sampled. READY must be stable (low or high) at the TMS320C2x no later than $t_{d(SL-R)} = (Q - 20)$ ns after $\overline{STRB}$ goes low.
4) With no wait-states (READY is high), data must be available no later than $t_{a(SL)} = (2Q - 23)$ ns after $\overline{STRB}$ goes low.

The sequence of events that occurs during an external write cycle is the same as the above, with the following differences:

1) R/$\overline{W}$ goes low to indicate a write cycle.
2) The data bus begins to be driven approximately concurrently with $\overline{STRB}$ going low.
3) The data bus enters a high-impedance state no later than $t_{dis(D)} = (Q + 15)$ ns after $\overline{STRB}$ goes high.

**Figure 7. Read and Write Timings**




## Interfacing with a PROM

A convenient means of implementing program memory in a TMS320C2x system is provided through the use of PROMs. Two separate approaches for interfacing PROMs to the TMS320C2x are considered. The first approach does not require address decoding since the system contains only a small amount of one type of memory. The second approach illustrates an interface that utilizes address decoding to distinguish between two or more memory types with different access times.

### *Direct PROM Interface*

An example of a no wait-state memory system is the direct PROM interface design shown in Figure 8. In this design, the TMS320C2x is interfaced with the Texas Instruments TBP38L165-35, a low-power, 2K × 8-bit PROM. The interface timing for the design of Figure 8 is shown in Figure 9.

**Figure 8. Direct Interface of the TBP38L165-35 to the TMS320C2x**

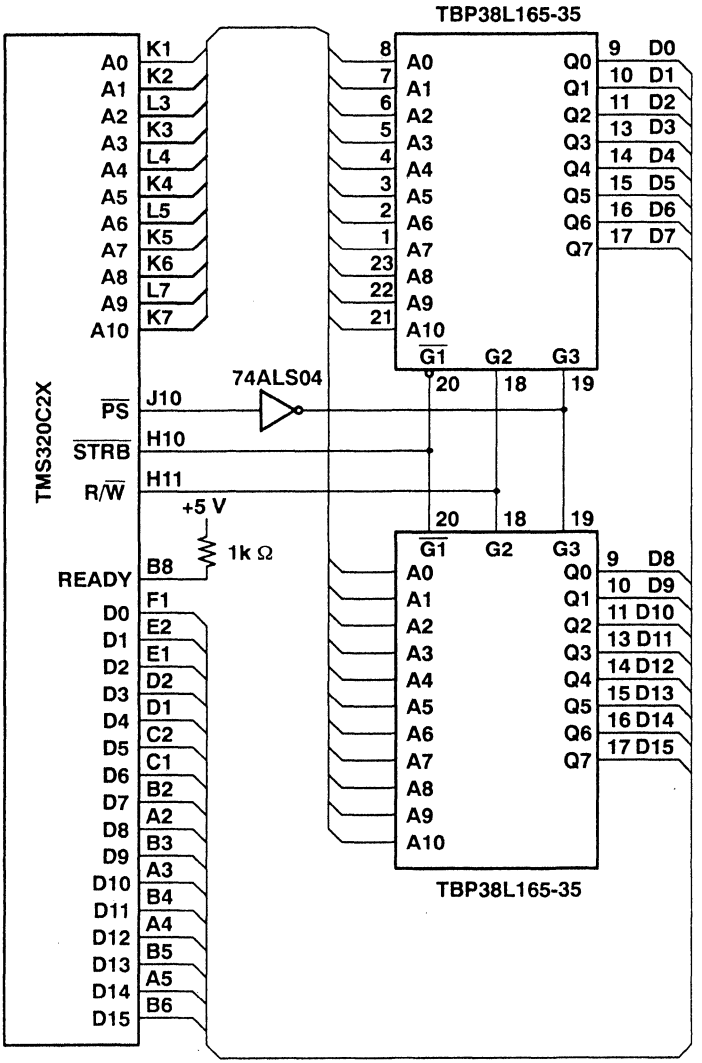
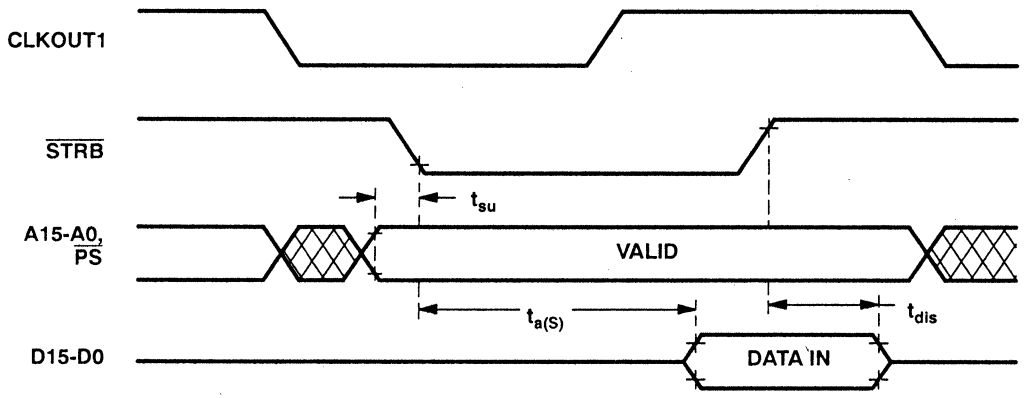


**Figure 9. Interface Timing of the TBP38L165-35 to the TMS320C2x**




As discussed earlier, the TMS320C2x expects data to be valid no later than (2Q – 23) ns after $\overline{\text{STRB}}$ goes low; this is 27 ns for a TMS320C2x operating at 40 MHz. The access times of the TBP38L165-35 are 35 ns maximum from address ($t_{a(A)}$), and 20 ns maximum from chip enable ($t_{a(S)}$). On the TMS320C2x, address becomes valid a minimum of $t_{su}$ = (Q – 12) ns = 13 ns before $\overline{\text{STRB}}$ goes low (see Figure 1). The memory is not enabled, however until $\overline{\text{STRB}}$ goes low. Therefore, the data appears on the data bus within 27 ns after $\overline{\text{STRB}}$ goes low, as required by the TMS320C2x.

Bus conflict may occur when a TMS320C2x write cycle is followed by a memory read cycle. In this case, the TMS320C2x data lines must enter a high-impedance state before the memory starts driving the data bus. In a write cycle, the TMS320C2x enters a high-impedance state no later than 15 ns after the beginning of the next cycle. Since the design of Figure 8 utilizes $\overline{\text{STRB}}$ to enable the TBP38L165s, these memories cannot drive the data bus before $\overline{\text{STRB}}$ goes low, i.e., Q ns after the beginning of the cycle. Therefore, bus conflict is avoided since 25 ns > 15 ns.

Note that the TMS320C2x R/$\overline{\text{W}}$ line is connected to the $G_2$ enable line on both TBP38L165s. Therefore, the PROMs are disabled whenever R/$\overline{\text{W}}$ goes low, even if $\overline{\text{STRB}}$ is active. This prevents the bus conflict that occurs if the PROMs are written to when using the TBLW instruction, which transfers data from the data memory space to the program memory space.[1] Such transfers, however, were intended to be made only when RAMs are used in the program space.

The most critical timing parameters of the TBP38L165-35 direct interface to the TMS320C2x are summarized in Table 3.

**Table 3. Timing Parameters of the TBP38L165-35 Direct Interface to the TMS320C2x**

| Description | Symbol Used in Figure 9 | Value |
|---|---|---|
| Address setup time | $t_{su}$ | 13 ns (min) |
| TBP38L165-35 access time from chip enable | $t_{a(S)}$ | 20 ns (max) |
| TBP38L165-35 disable time | $t_{dis}$ | 15 ns (max) |

### *PROM Interface with Address Decoding*

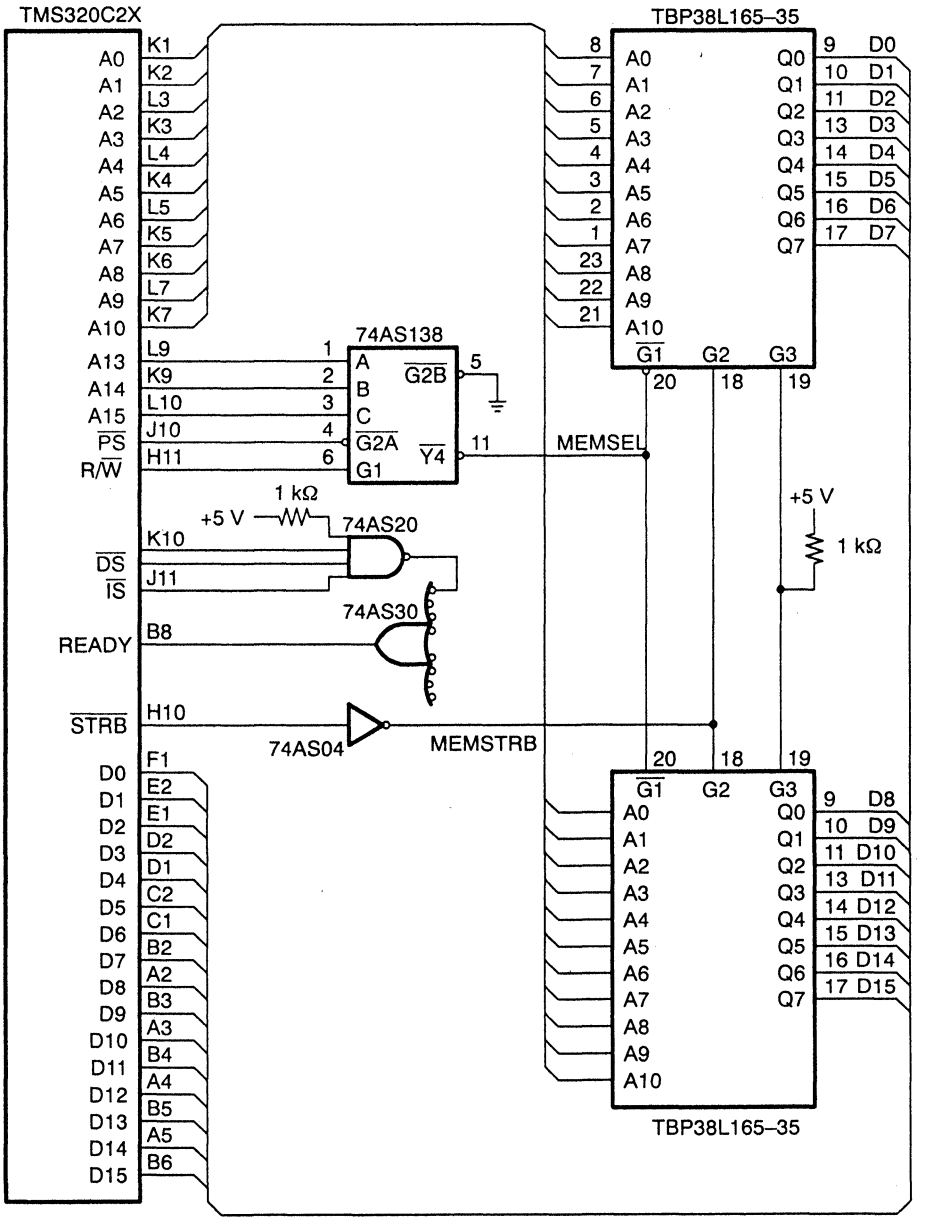
The second design example considers the interface of PROMs to the TMS320C2x using address decoding. A major issue when designing an interface with address decoding is that the TMS320C2x requires the READY signal to be stable no later than (Q – 20) ns after $\overline{\text{STRB}}$ goes low. Since the setup time for the address is (Q – 12) ns, the TMS320C2x requires (worst case) a stable READY at least (2Q – 32) ns after the address has been stabilized. This is 18 ns at 40 MHz. Proper address decoding may require two levels of gating. A third level of gating is required when more than one type of memories or peripherals with different numbers of wait-states is used. Using 'AS interface logic (the fastest currently available), these three levels of gating have a total propagation delay of 15 ns (worst case). Using a 74AS138 three-to-eight-line decoder to implement the first two levels of gating does will not result in any significant improvement in the propagation delay. (The 74AS138 has a maximum propagation delay of 9.5 ns for a high-to-low transition.)

An approach that can be used to meet the READY timing requirements is shown in Figure 10. This design utilizes one address decoding scheme to generate READY, and a second address decoding scheme to enable the different memory banks.

In this design, the memories with no wait-states are mapped at the upper half (upper 32K) of the program space. The lower half is used for memories with one or more wait-states. This decoding is implemented with the 74AS20 four-input NAND gate. The output of this gate is low when the following are true:

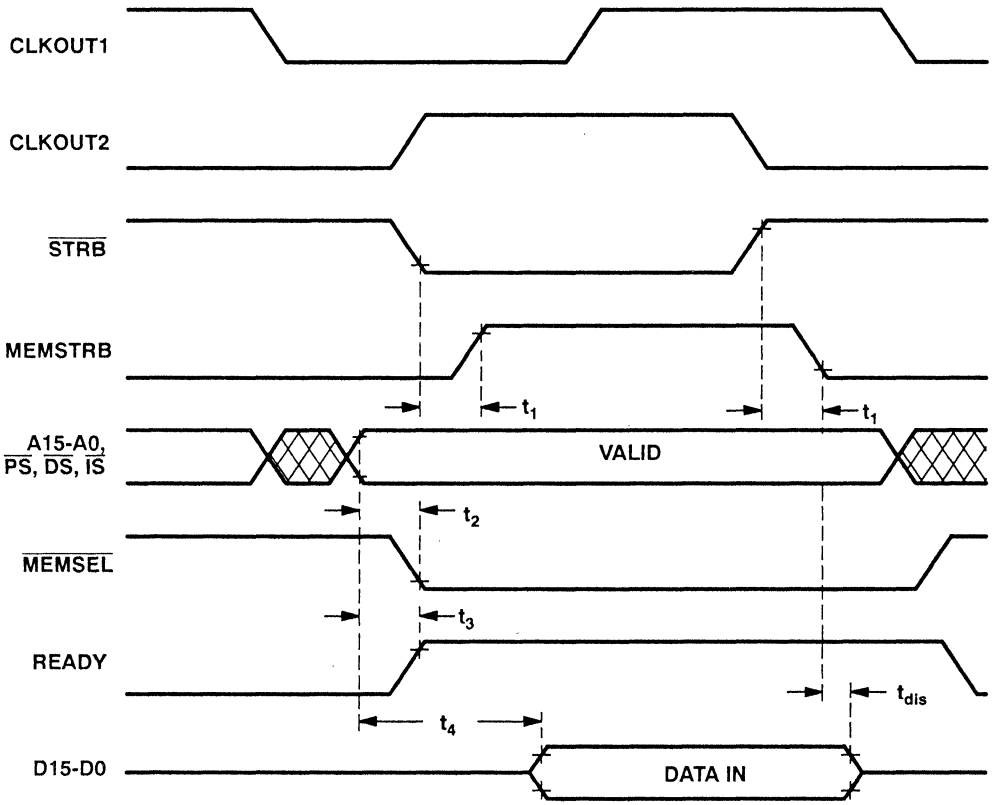
1) Address line A15 is high; i.e., the upper 32K words are selected.
2) DS and IS are high; i.e., an external program memory cycle is in progress.

**Figure 10. Interface of the TBP38L165-35 to the TMS320C2x**



The timing of READY is shown in Figure 11. READY goes high 10 ns (worst case) after the address has become valid.

**Figure 11. Interface Timing of the TBP38L165-35 to the TMS320C2x (with Address Decoding)**




Address decoding is implemented by the 74AS138. This decoding separates the program space into eight segments of 8K words each. The first four of these segments (lower 32K of address space) are enabled by the Y0, Y1, Y2, and Y3 outputs of the 74AS138. These segments are used for memories with one or more wait-states. The other four segments select memories with no wait-states (the TBP38L165s are mapped in segment #5 starting at address 8000h). Note that in Figure 10, R/$\overline{W}$ is used to enable the 74AS138. This prevents a bus conflict from occurring if an attempt is made to write to the PROMs.

In Figure 10, $\overline{\text{MEMSEL}}$ goes low no later than 10 ns (time $t_2$ in Figure 11) after address is valid. The PROMs are not enabled, however, until MEMSTRB goes high, i.e., a minimum of 5 ns after $\overline{\text{STRB}}$ goes low (time $t_1$ in Figure 11). Valid data appears on the data bus within 25 ns later. This meets the 27 ns or (2Q – 23) ns access time required from $\overline{\text{STRB}}$ low by the TMS320C2x. Note that in the design of Figure 10, $\overline{\text{STRB}}$ is used to enable the PROMs so that no bus conflict occurs

if the memory read cycle is followed by a write cycle. As seen in Figure 11, the memory enters a high-impedance state within ($t_1 + t_{dis}$) = 20 ns after $\overline{STRB}$ goes high. Therefore, if a memory read cycle is followed by a write cycle, no bus conflict occurs since the TMS320C2x starts driving the data bus no earlier than Q ns after the beginning of the write cycle.

The most critical timing parameters of the TBP38L165-35 interface with address decoding to the TMS320C2x are summarized in Table 4.

**Table 4. Timing Parameters of the TBP38L165-35 Interface with Address Decoding to the TMS320C2x**

| Description | Symbol Used in Figure 11 | Value |
|---|---|---|
| Propagation delay through the 74AS04 | $t_1$ | 5 ns (max) |
| Propagation delay through the 74AS138 | $t_2$ | 10 ns (max) |
| Address valid to READY | $t_3$ | 10 ns (max) |
| TBP38L165-35 disable time | $t_{dis}$ | 15 ns (max) |

In summary, when interfacing to PROM memories with the TMS320C2x, two different approaches can be taken depending on whether or not any of the memories in the system require wait-states. When no wait-states are required for any of the memories, READY can be tied high, and the interface to the PROMs becomes a direct connection. When some of the system memories require wait-states, address decoding must be performed, and a valid READY signal that meets the TMS320C2x timing requirements must be provided. An efficient method of accomplishing this is to use one section of circuitry to generate the address decode, and a second, independent section to generate the READY signal.

## EPROM Interfacing

EPROMs may be used to debug TMS320C2x algorithms. Three different EPROM interfaces to the TMS320C2x are presented in this subsection. First, the direct interface of an EPROM that requires no wait-states is discussed. This is followed by descriptions of EPROM interfaces that require one and two wait-states.

### *Direct EPROM Interface with No Wait-States*

A Texas Instruments TMS27C292-35 EPROM can interface directly to the TMS320C2x with no wait-states, as shown in Figure 12. The TMS27C292-35 is a CMOS EPROM with access times of 35 ns from valid address and 25 ns from chip select. The timing of the interface is shown in Figure 13.

**Figure 12. Direct Interface of the TMS27C292-35 to the TMS320C2x**

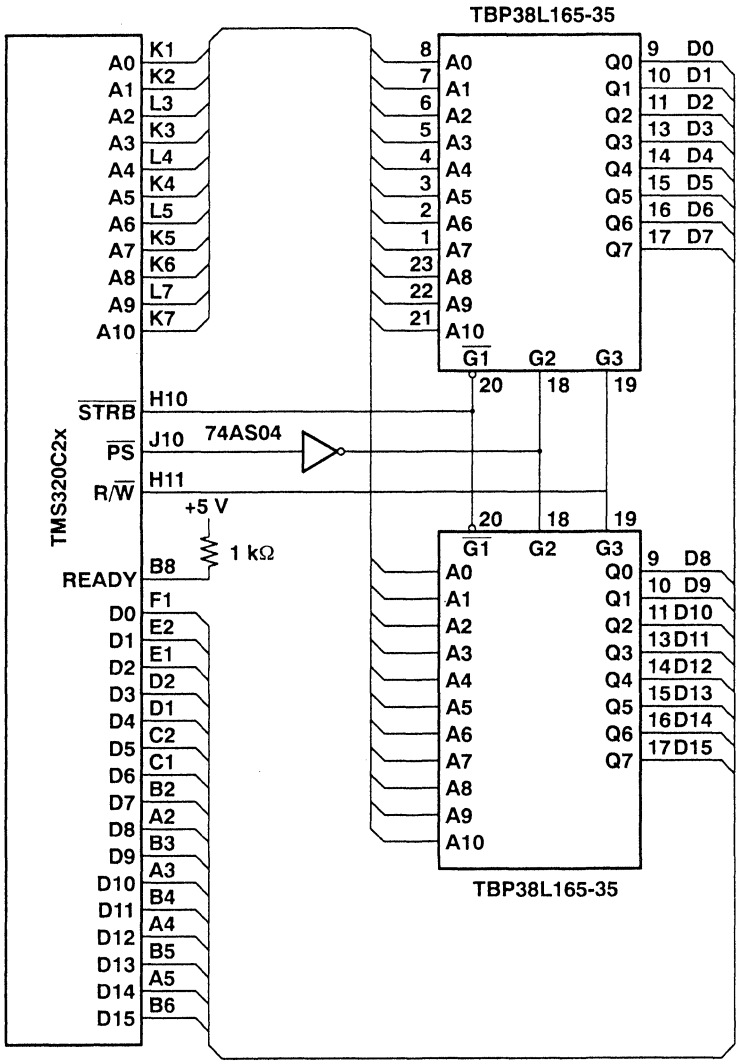
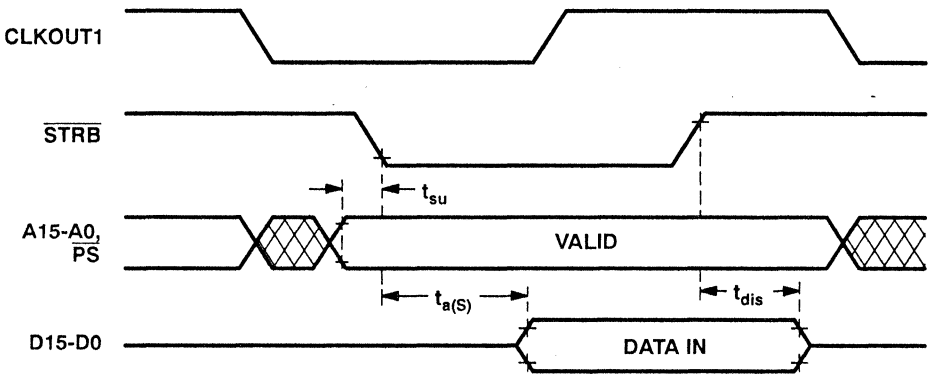


**Figure 13. Interface Timing of the TMS27C292-35 to the TMS320C2x**




As shown in Figure 13, the EPROMs are not enabled until $\overline{\text{STRB}}$ goes low. Since the address has been valid for at least $t_{su}$ = 13 ns before $\overline{\text{STRB}}$ goes low, valid data appear on the data bus $t_{a(S)}$ = 25 ns (max) later. The EPROMs are disabled with $\overline{\text{STRB}}$ going high, and their output buffers enter a high-impedance state, $t_{dis}$ = 25 ns (max) later. Therefore, no bus conflict occurs even if the memory read cycle is followed by a write cycle.

The most critical timing parameters of the TMS27C292-35 direct interface to the TMS320C2x are summarized in Table 5.

**Table 5. Timing Parameters of the TMS27C292-35 Direct Interface to the TMS320C2x**

| Description | Symbol Used in Figure 11 | Value |
|---|---|---|
| Address setup time | $t_{su}$ | 13 ns (min) |
| TMS27C292-35 access time from chip enable | $t_{a(S)}$ | 25 ns (max) |
| TMS27C292-35 disable time | $t_{dis}$ | 25 ns (max) |

### *EPROM Interface with One Wait-State*

The hardware interface of the Wafer Scale WS57C64F-12 (8K × 8-bit EPROMs) to the TMS320C2x is shown in Figure 14. The WS57C64F-12s are mapped at address 2000h. The interface timing diagram is provided in Figure 15.

**Figure 14. Interface of the WS57C64F-12 to the TMS320C2x**

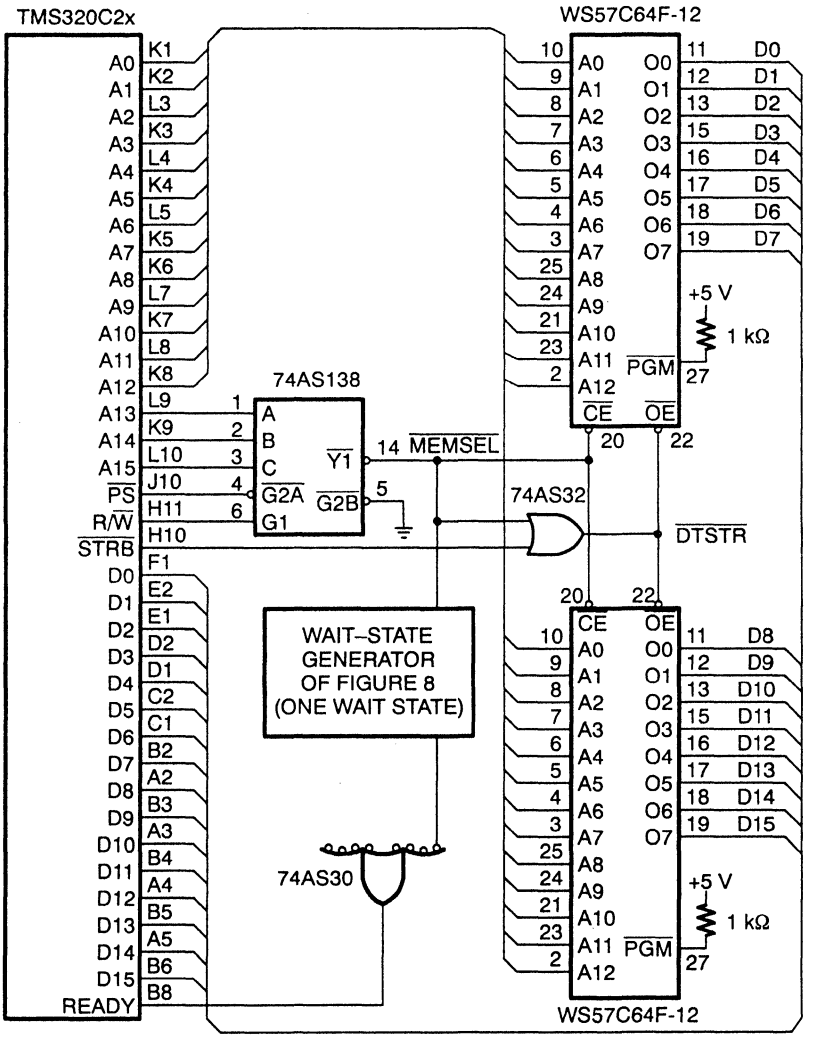
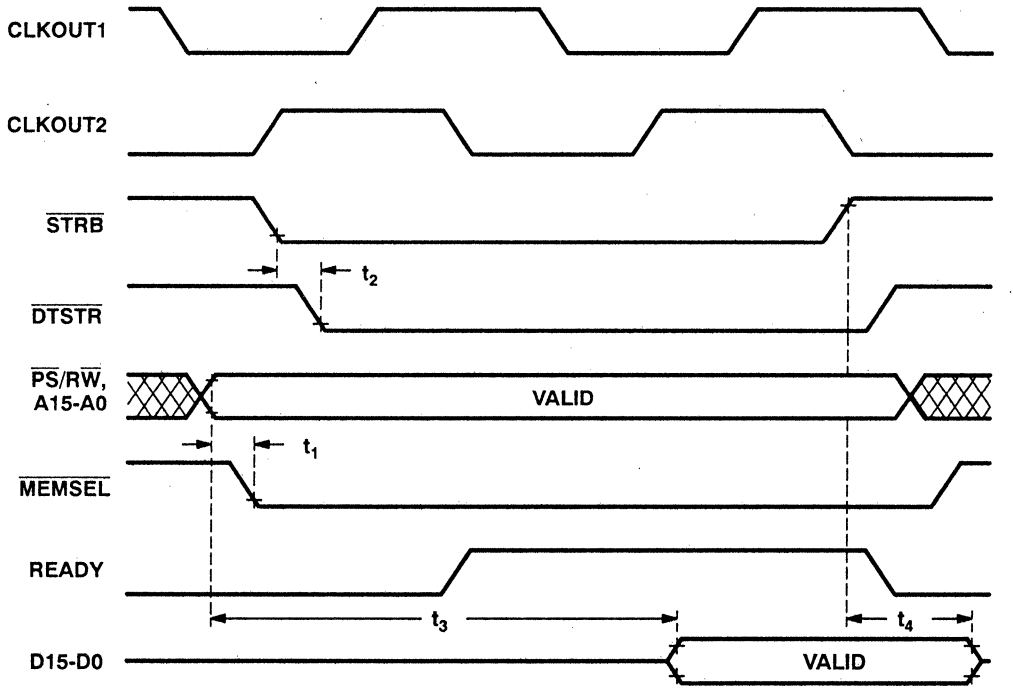


**Figure 15. Interface Timing of the WS57C64F-12 to the TMS320C2x**




The WS57C64-12 access times from valid address, chip select, and output enable are $t_{a(A)}$ = 120 ns (max), $t_{a(CE)}$ = 120 ns (max), and $t_{a(OE)}$ = 35 ns (max), respectively. As shown in Figure 14, the 74AS138 is used for address decoding. $\overline{PS}$ and R/$\overline{W}$ are used to drive the $\overline{G2A}$ and G1 enable inputs of the 74AS138, respectively. The latter prevents any bus conflict resulting from an accidental write (using the TBLW instruction) to the program space. $\overline{MEMSEL}$ going low $t_1$ = 10 ns (max) after address valid (see Figure 15) is used for two purposes:

1) to drive the wait-state generator, as discussed earlier; and
2) to generate a strobe signal, $\overline{DTSTR}$, that activates the output buffers of the WS57C64-12s.

Time $t_3$ in Figure 15, is the time from valid address to valid data on the data bus, i.e., $t_3 = t_1 + t_{a(CE)}$ = 130 ns (max). Since 40 ns < $t_3$ < 140 ns, one wait-state is required. The wait-state generator of Figure 14 may be used to implement this wait-state. Also, note that the WS57CF64-12 is the slowest member of the WS57C64F EPROM series, and still meets the specifications for one wait-state.

With $\overline{STRB}$ going high, the read has been completed. $\overline{DTSTR}$ is then used to turn off the memory output buffers. The output disable time of the WS57C64F-12 is $t_{dis}$ = 35 ns (max). Time $t_4$ in Figure 15 is used to indicate the time from $\overline{STRB}$ high to output entering a high-impedance state. With a propagation delay of $t_p$ = 5.8 ns (max) through the 74AS32, $t_4 = t_p + t_{dis}$ = 40.8 ns (max). Since this time is less than 50 ns (the earliest the TMS320C2x can start driving the data bus when the next instruction is a write), there is no bus conflict.

Table 6 summarizes the most critical timing parameters of the WS57C64F-12 interface to the TMS320C2x.

**Table 6. Timing Parameters of the WS57C64F-12 Interface to the TMS320C2x**

| Description | Symbol Used in Figure 11 | Value |
|---|---|---|
| Address valid to $\overline{\text{MEMSEL}}$ low | $t_1$ | 10.5 ns (max) |
| $\overline{\text{STRB}}$ to $\overline{\text{DTSTR}}$ low | $t_2$ | 5.8 ns (max) |
| TMS320C2x address valid to WS57C64F-12 data valid | $t_3$ | 130.0 ns (max) |
| $\overline{\text{STRB}}$ high to WS57C64F-12 output disable | $t_4$ | 40.8 ns (max) |

*EPROM Interface with Two Wait-States*

The interface of the TMS27C64-20 to the TMS320C2x is shown in Figure 16. The TMS27C64-20 is a CMOS 8K × 8-bit EPROM with an access time of 200 ns. The timing diagram is shown in Figure 17.

**Figure 16. Interface of the TMS27C64-20 to the TMS320C2x**

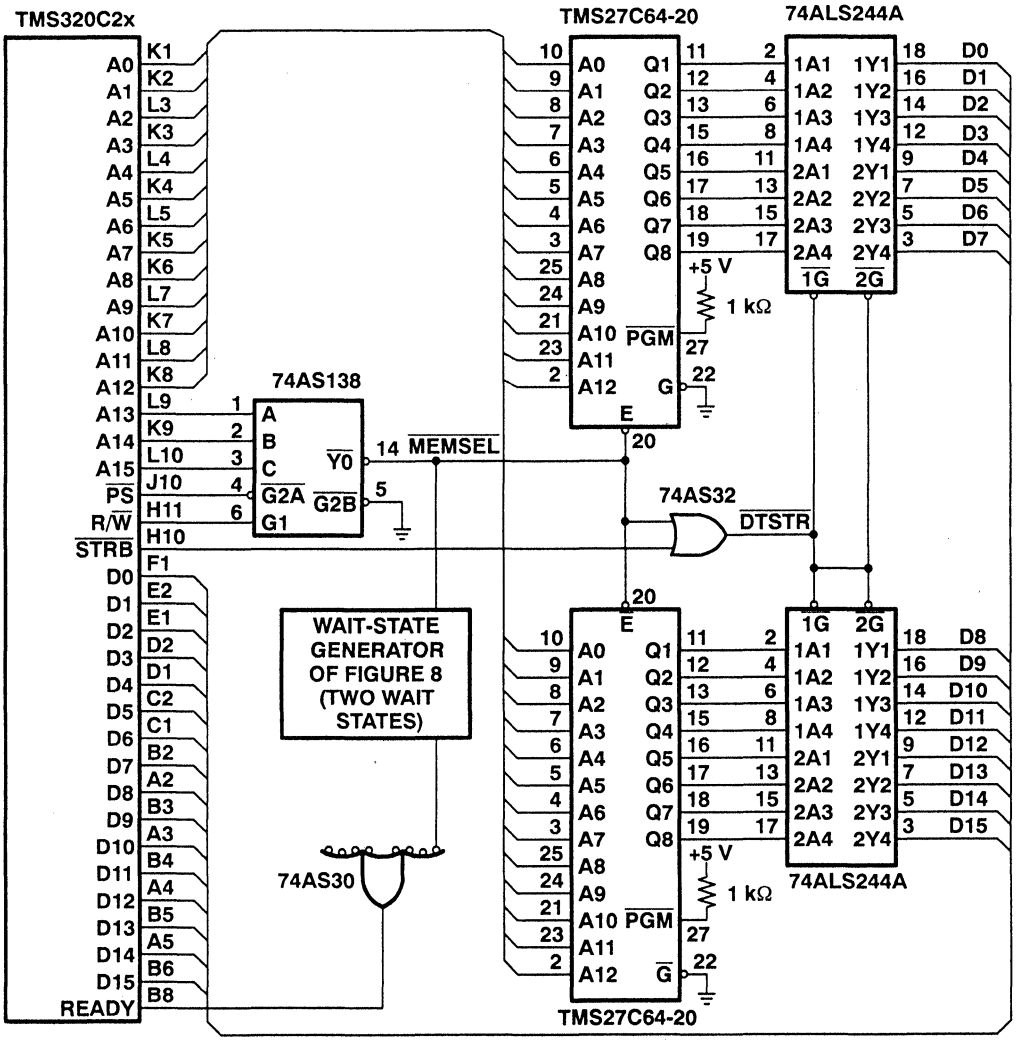
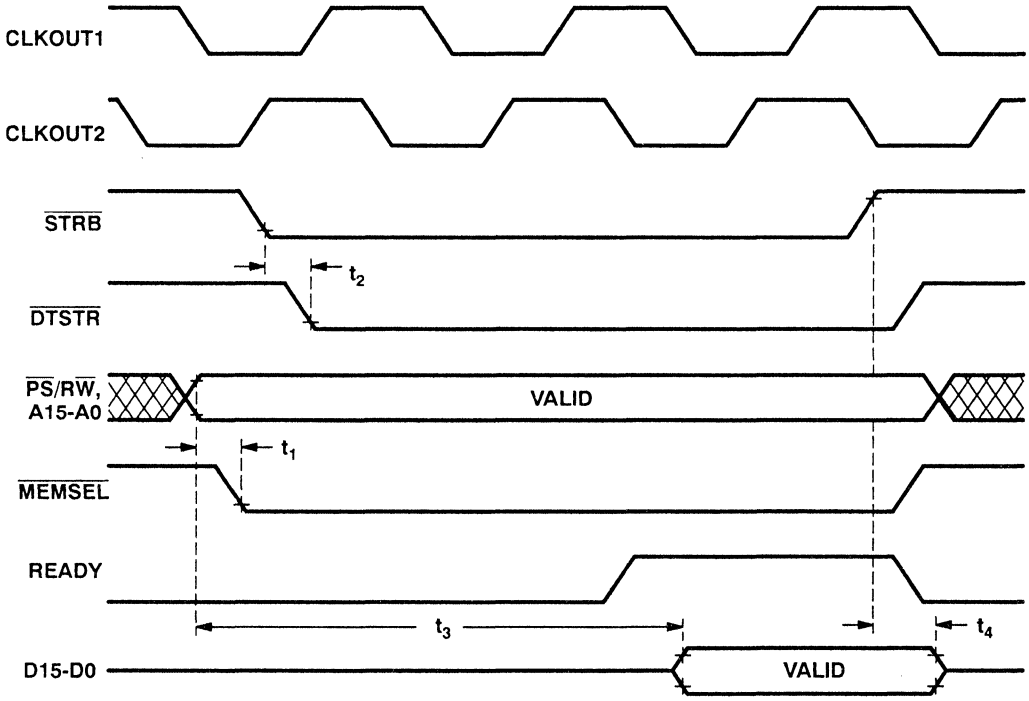


**Figure 17. Interface Timing of the TMS27C64-20 to the TMS320C2x**




With a 200-ns access time, two wait-states are needed. These can be implemented using the wait-state generator of Figure 14(a). Address decoding is similar to that used for the WS57C64F-12, and the TMS27C64 is mapped at address 0000h. The memory cycle starts with address valid. $\overline{\text{MEMSEL}}$ becomes low $t_1$ = 10 ns (max) later, due to propagation delay through the 74AS138. With $\overline{\text{MEMSEL}}$ active, valid data appear on the TMS27C64 data lines, $t_a$ = 200 ns (max) later. As shown in Figure 16, the 74ALS244A octal buffers are used to buffer the memories from the TMS320C2x. These buffers are enabled with $\overline{\text{DTSTR}}$, which is a logical-OR signal of both $\overline{\text{MEMSEL}}$ and $\overline{\text{STRB}}$. The maximum propagation delay through these buffers is $t_p$ = 10 ns. Therefore, valid data appear on the TMS320C2x data bus no later than $t_3 = t_1 + t_a + t_p$ = 220 ns from valid address. This is the overall access time, and 140 ns < $t_3$ < 240 ns, i.e., two wait-states are sufficient.

With $\overline{\text{STRB}}$ going high, the TMS320C2x has completed the memory read. $\overline{\text{DTSTR}}$ follows $\overline{\text{STRB}}$, and $t_2$ = 5.8 ns (maximum propagation delay through the 74AS32) after STRB goes high; $\overline{\text{DTSTR}}$ also goes high. This forces the 74ALS244As to enter a high-impedance state 13 ns (max) later. Therefore, no later than $t_4$ = (13 + 5.8) ns = 18.8 ns after $\overline{\text{STRB}}$ goes high, the outputs of the 74ALS244As are in a high-impedance state (see Figure 12). Buffers were used because the disable time of the TMS27C64-20 is 60 ns, which will generate a conflict on the data bus.

Table 7 summarizes the most critical timing parameters of the TMS27C64-20 interface to the TMS320C2x.

**Table 7. Timing Parameters of the TMS27C64-20 Interface to the TMS320C2x**

| Description | Symbol Used in Figure 11 | Value |
|---|---|---|
| Address Valid to $\overline{\text{MEMSEL}}$ low | $t_1$ | 10.5 ns (max) |
| $\overline{\text{STRB}}$ low to $\overline{\text{DTSR}}$ low | $t_2$ | 5.8 ns (max) |
| TMS320C2x address valid to TMS27C64-20 data valid | $t_3$ | 220.0 ns (max) |
| $\overline{\text{STRB}}$ high to TMS27C64-20 output disable | $t_4$ | 18.8 ns (max) |

In summary, EPROMs can be a valuable tool during the prototyping stages of a design, and may even be desirable for production. When EPROMs that are fast enough are used with the TMS320C2x, a direct interface similar to that used for PROMs may be used. When slower, less costly EPROMs are used, a simple flip-flop circuit can be used to generate one or more wait-states. With slower EPROMs, however, data output turnoff can be slow, and must be taken into consideration in the design. The same advantages are offered by the TMS320E25, which has an on-chip 4K-word EPROM in place of the 4K-word on-chip ROM of the TMS320C25.

## Interfacing SRAMS

The TMS320C2x can utilize SRAM as either program or data memory. When used as program memory, object code can be downloaded into the RAM and executed. SRAM can also be used as data memory to extend the TMS320C2x's 544 words of internal RAM. In the first case, the SRAM is mapped into the TMS320C2x program space, while the second case maps the SRAM into the data space.

The SRAM chosen for this interface is the Cypress Semiconductor CY7C169-25 4K × 4-bit SRAM. This SRAM has a 25-ns access time from address ($t_{a(A)}$) and a 15-ns access time from chip enable ($t_{a(CE)}$). Note that these access times are fast enough that a wait-state generator is not required for this interface. If, however, RAMs that require wait-states are used in the system,the wait-state generator described in the Interfacing EPROMs subsection can be used.

RAMs with a 4K × 4-bit organization are used in this application to minimize the package count for the desired number of words of memory being implemented. In this case, only four packages are required. In contrast, if 16K × 1-bit memories had been used, 16 packages would have been required, and much of the memory might have gone unused. In general, the choice of memory organization for a particular system should be based on the amount of memory required and the organization of the memories currently available in the industry.

The hardware interface to this RAM is shown in Figure 18, and a timing diagram of the interface is presented in Figure 19.

## Figure 18. Interface of the CY7C169-25 to the TMS320C2x

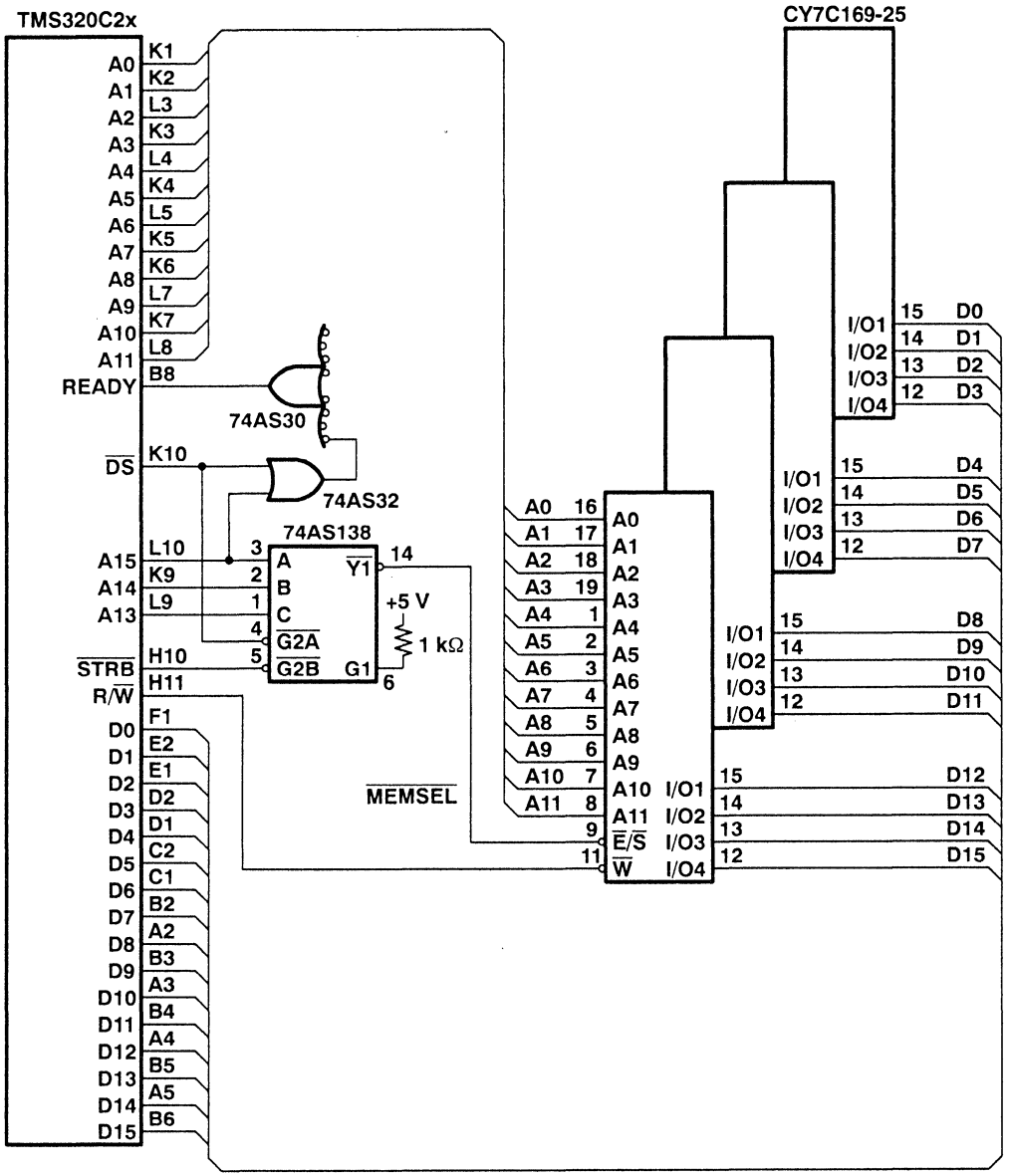
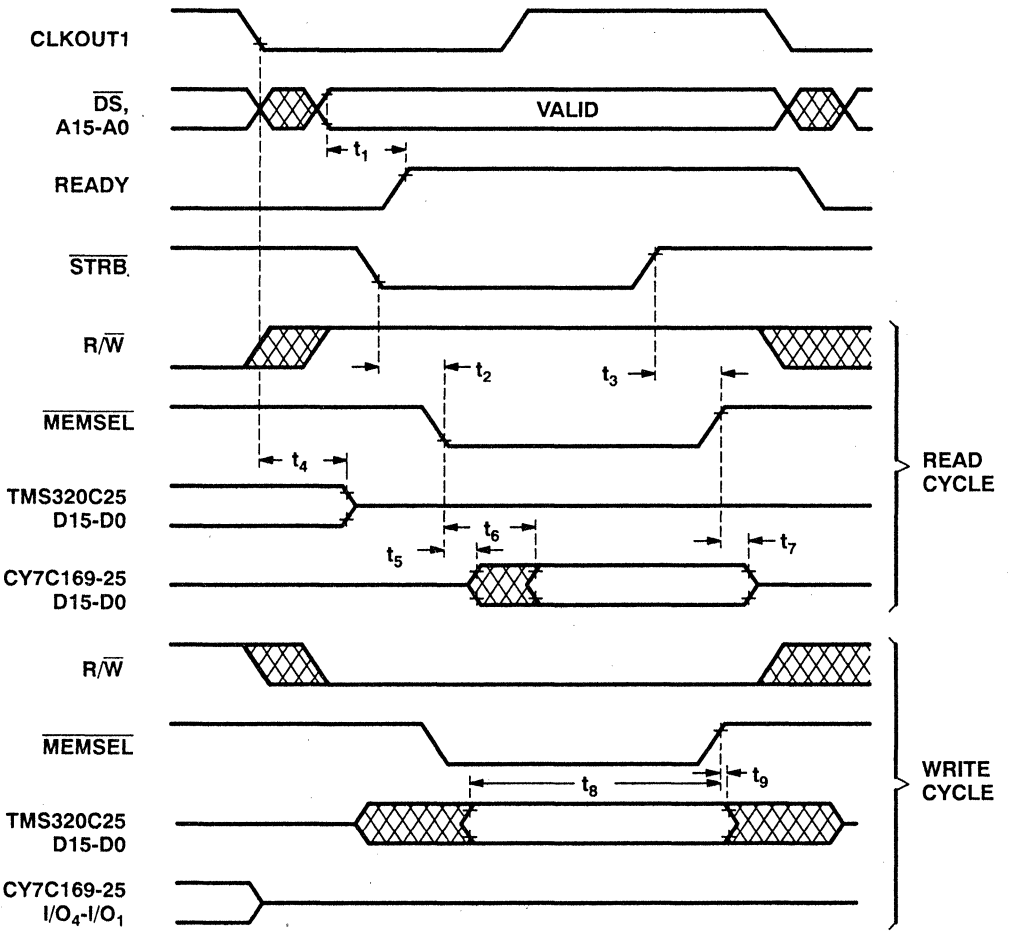


**Figure 19. Interface Timing of the CY7C169-25 to the TMS320C2x**




The design of Figure 18 utilizes a similar approach to the one described in the Interfacing PROMs and Interfacing EPROMs subsections; i.e., one address decoding scheme is used to generate READY, and a second address decoding scheme is used to enable the SRAM. In this design, RAMs with no wait-states are mapped at the lower half (lower 32K words) of the TMS320C2x data space. The upper half is used for memories with one or more wait-states. This decoding is implemented with the 74AS32 two-input OR gate. The output of this gate is low (active) when $\overline{DS}$ is low (i.e., access to external data space requested), and A15 is low (i.e., lower 32K words selected). Time $t_1$ in Figure 19 indicates the time from valid address to READY going high. The maximum value for $t_1$ is

$$t_1 = t_{p(74AS32)} + t_{p(74AS30)} = (5.8 + 5)\ ns = 10.8\ ns$$

where $t_{p(X)}$ denotes the maximum propagation delay through device X.

As shown in Figure 18, address decoding that enables the RAM is implemented with the 74AS138. This decoding separates the data space into eight segments with 8K words per segment. The first four segments are enabled by the $\overline{Y0}$, $\overline{Y1}$, $\overline{Y2}$, and Y3 outputs of the 74AS138. These segments are used for memories with no wait-states. Note, in Figure 18, that the CY7C169s are enabled by $\overline{Y1}$; i.e., the memories are mapped at address 2000h. The rest of those segments, enabled by the other outputs of the 74AS138 decoder, are used for memories with one or more wait-states.

### *Memory Read Cycle*

Figure 19 shows the timing for memory read and write cycles. In a read cycle, R/W goes high concurrently with valid address, indicating that a read rather than a write cycle has been initiated. With $\overline{STRB}$ used to enable the 74AS138, $\overline{MEMSEL}$ goes low no later than $t_2$ = 8.5 ns after $\overline{STRB}$ goes low. This is the maximum propagation delay of the 74AS138 before outputting a high-to-low transition from the $\overline{G}$ enable pin. The CY7C169s begin driving the data bus no earlier than $t_5$ = 5 ns after $\overline{MEMSEL}$ goes low. By then, all of the devices having access to the data bus must have entered a high-impedance state. Figure 19 shows the TMS320C2x data lines entering a high-impedance state no later than $t_4$ = 15 ns after the beginning of the read cycle. This is the case when the present read cycle is preceded by a write cycle.

The RAMs provide valid data no later than $t_6$ = 15 ns after $\overline{MEMSEL}$ goes low. Therefore, the worst-case access time from $\overline{STRB}$ going low is $t_2 + t_6$ = 23.5 ns. This meets the 27-ns access time required by the TMS320C2x operating at 40 MHz.

The TMS320C2x read cycle is concluded with $\overline{STRB}$ going high. $\overline{MEMSEL}$ follows $\overline{STRB}$ and goes high within $t_3$ = 7.5 ns. This time is the maximum propagation delay through the 74AS138 for a low-to-high transition. The CY7C169 data lines enter a high-impedance state no later than $t_7$ = 15 ns after $\overline{MEMSEL}$ goes high. Therefore, no bus conflict occurs if the present read cycle is followed by a write cycle.

### *Memory Write Cycle*

As shown in Figure 19, the memory write cycle is similar to the read cycle with the exception that R/$\overline{W}$ is low. The TMS320C2x begins driving the data bus as soon as $\overline{STRB}$ goes low, while $\overline{MEMSEL}$ follows $\overline{STRB}$ within $t_2$ = 8.5 ns. Since R/$\overline{W}$ is low when $\overline{MEMSEL}$ goes low, the CY7C169s do not drive the data bus.

Data is clocked into the CY7C169s on the rising edge of $\overline{MEMSEL}$. Time $t_8$ in Figure 19 is the time that data is valid before $\overline{MEMSEL}$ goes high. This time is no less than the TMS320C2x minimum data setup time before $\overline{STRB}$ goes high ($t_8$ = (2Q – 20) ns = 30 ns when operating at 40 MHz) plus the 2-ns minimum propagation delay through the 74AS138. Therefore, $t_8$ is equal to or greater than 32 ns. Note that this time meets the 10-ns minimum data setup time required by the CY7C169.

Table 8 summarizes the most critical timing parameters that must be considered when interfacing the CY7C169s with the TMS320C2x.

**Table 8. Timing Parameters of the CY7C169-25 Interface to the TMS320C2x**

| Description | Symbol Used in Figure 11 | Value |
|---|---|---|
| Address valid to READY valid | $t_1$ | 10.8 ns (max) |
| STRB low to MEMSEL low | $t_2$ | 8.5 ns (max) |
| STRB high to MEMSEL high | $t_3$ | 7.5 ns (max) |
| CLKOUT1 low to TMS320C2x data bus entering the high-impedance state | $t_4$ | 15.0 ns (max) |
| MEMSEL low to CY7C169-25 driving | $t_5$ | 5.0 ns (min) |
| MEMSEL low to CY7C169-25 data valid | $t_6$ | 15.0 ns (max) |
| MEMSEL high to CY7C169-25 entering the high-impedance state | $t_7$ | 15.0 ns (max) |
| Data setup time for a write | $t_8$ | 32.0 ns (min) |
| Data hold time | $t_9$ | 7.5 ns (min) |

In summary, interfacing external RAM to the TMS320C2x is quite useful for expanding the internal data memory or implementing additional RAM program memory. In cases where RAMs of different execution times are used, separate schemes for address decoding and READY generation can be used to meet READY timing requirements in a similar manner to that used for the PROM interface as described in this report. RAMs with similar access times may then be grouped together in one segment of memory.

## Interfacing Memories to the TMS320C25-50

TMS320C25-50 memory interfaces are similar or identical in form to those of the 40-MHZ version of the TMS320C25. In many cases, the interfacing techniques given in the preceding section can be used, with higher-speed versions of the memory devices substituted.

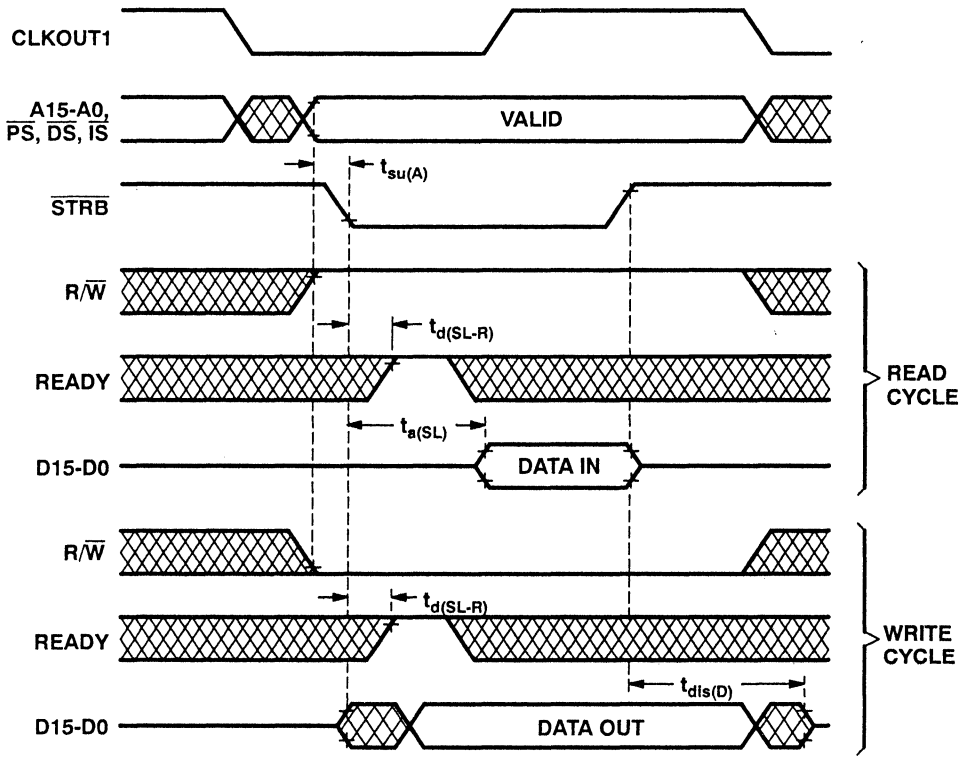
This section describes the memory interface timing requirements of the TMS320C25-50. Determining appropriate memory device speeds requires an understanding of TMS320C25-50 external bus cycles and the timing specification of the device.

The following excerpt from the TMS320C25-50 Electrical Specification and Figure 20 show the information necessary to determine the minimum memory device speed for a given application.

| | Min | Max | Units |
|---|---|---|---|
| $t_{a(A)}$ | | 3Q–31 | ns |
| $t_{su(A)}$ | Q–11 | | ns |
| $t_{su(D)R}$ | 17 | | ns |

Figure 20 shows a TMS320C25-50 memory read and write cycle. Either of two timing requirements must be satisfied to guarantee a successful read operation. These two requirements are specified by $t_{a(A)}$ and $t_{su(D)R}$. Note that it is not necessary to satisfy both requirements, as each parameter is guaranteed independently.

**Figure 20. TMS320C25-50 Memory Read and Write Cycle**



A timing requirement of special interest is the memory access time measured from the falling edge of $\overline{STRB}$. The specification of this requirement is jointly implied by the device $t_{a(A)}$ and $t_{su(D)R}$ specifications as shown in the following.

$t_{a(A)}$ is defined as follows:

$$t_{a(A)} = t_{su(A)}\text{min} + t_{w(SL)} + t_{r(C)} - t_{su(D)R}\text{min}$$

For convenience, define $t_{w(S)}$ as follows:

$$t_{w(S)} = t_{w(SL)} + t_{r(C)}$$

Then $t_{a(A)}$ is given by

$$t_{a(A)} = t_{su(A)}\text{min} + t_{w(S)} - t_{su(D)R}\text{min}$$

The $t_{a(A)}$ specification guarantees that

$$t_{a(A)} > t_{a(A)\text{max}}$$

or

$$t_{su(A)\text{min}} + t_{w(S)} - t_{su(D)R\text{min}} > t_{a(A)\text{max}}$$

The above inequality is potentially confusing in that it guarantees a minimum on a parameter with a *max* subscript. As with any parameter specified as a *maximum*, the measured $t_{a(A)}$ value of a given device must be *greater* than the specified maximum in order for the device to pass the $t_{a(A)}$ test performed on the device. In this way, all values of $t_{a(A)}$ *less* than $t_{a(A)\text{max}}$ are guaranteed to meet the device $t_{a(A)}$ requirement.

$t_{a(A)\text{max}}$ is specified as

$t_{a(A)\text{max}} = 3Q–35$ ns (40 MHz TMS320C25)
$t_{a(A)\text{max}} = 3Q–31$ ns (TMS320C25-50)

Thus, the following inequalities are guaranteed:

$Q–12 + t_{w(S)} – 23 > 3Q–35$ (40 MHz TMS320C25)
$Q–11 + t_{w(S)} – 17 > 3Q–31$ (TMS320C25-50)

which evaluate to

$t_{w(S)} > 2Q$ (40 MHz TMS320C25)
$t_{w(S)} > 2Q–3$ (TMS320C25-50)

The ta(A) specification thus implies a minimum value for $t_{w(S)}$.

On a memory read cycle, data must be valid no later than $t_{su(D)R\text{min}}$ prior to $\overline{STRB}$ going high. The maximum access time from $\overline{STRB}$ low (define this as $t_{a(SL)}$) is thus

$$t_{a(SL)\text{max}} = t_{w(S)\text{min}} - t_{su(D)R\text{min}}$$
$$= 2Q - 23 \qquad \text{(40 MHz TMS320C25)}$$

or

$$= (2Q–3) - 17 = 2Q–20 \qquad \text{(TMS320C25-50)}$$

The specification of $t_{a(SL)}$ typically determines the maximum access time from chip select and/or output enable for a memory device, as discussed in the following sections. Note that the

specification of the minimum value of $t_{w(SL)}$ ($\overline{STRB}$ – low pulse width) is in no way involved in assessing access time from address or from $\overline{STRB}$ going low.

### *Full-Speed Interfaces*

The TMS320C25-50 can be interfaced to fast SRAM with no wait-states. Two key memory device specifications for such an interface are access time from address valid and access time from chip select and/or output enable. The key TMS320C25-50 timing requirements are specified by $t_{a(SL)}$ and $t_{a(A)}$.

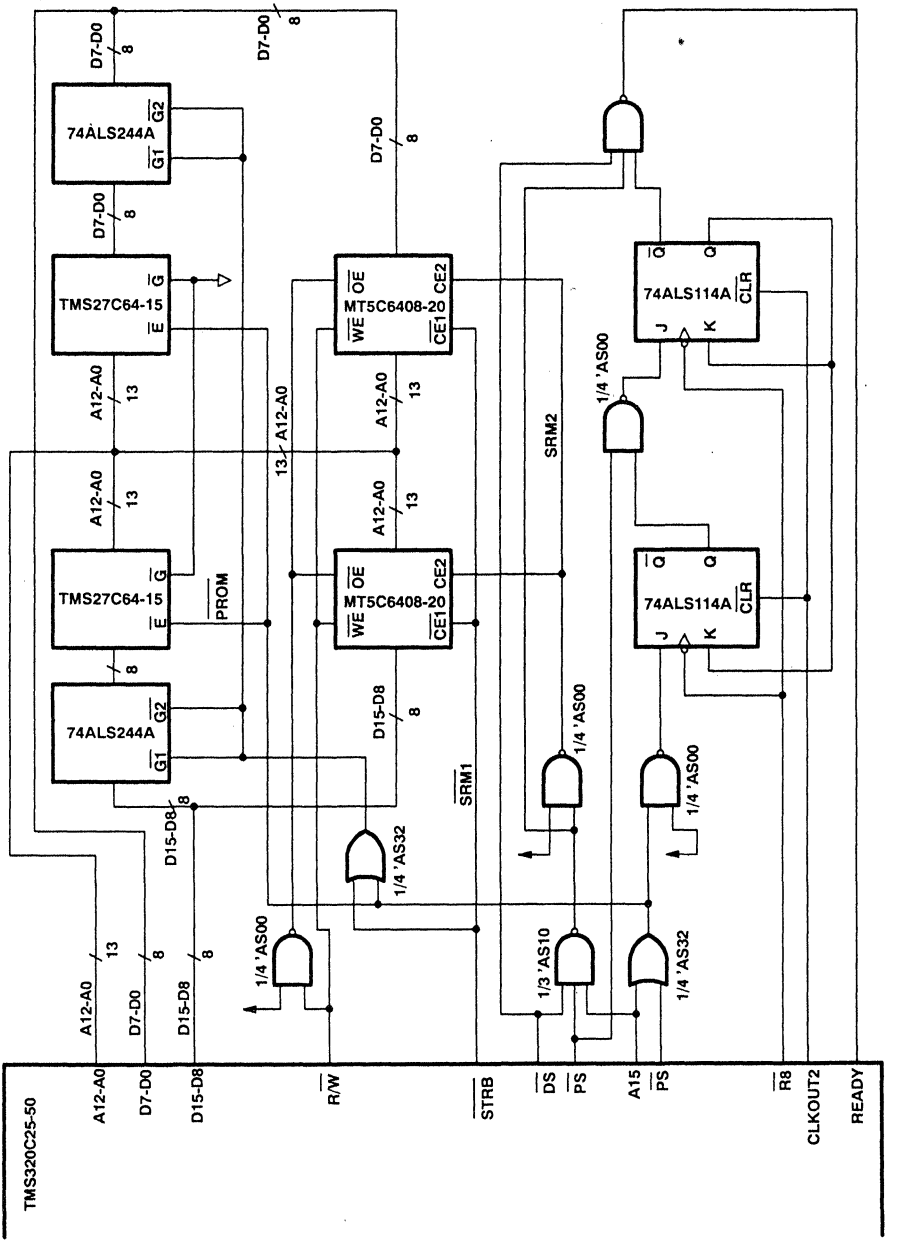
If $\overline{STRB}$ is an input to logic that generates the chip select and/or output enable signal for a memory device, data must be guaranteed valid no later than $t_{a(SL)} - t_d$ from $\overline{STRB}$ falling, where $t_d$ is the delay imposed by the logic used to generate the chip select or output enable signal.

Typically, devices with both chip select and output enable signals can more easily accommodate the $t_{a(SL)}$ requirement, as $\overline{STRB}$ can directly serve as the output enable signal (active low), resulting in the condition $t_d = 0$. Logic internal to the memory device enables the device's input or output buffers (depending on the state of R/$\overline{W}$) only if the chip is selected via its chip select input.

Interfaces to memory devices having a chip select input but no output enable input will include chip select logic having $\overline{STRB}$ as one of its inputs. In these cases $t_d$ is nonzero and thus the requirement on access time from chip select is tightened.

Figure 21 shows a TMS320C25-50 interfaced to 8K-words of full-speed SRAM and 8K-words of two wait-state EPROM. The operation of this circuit is discussed in the following section.

**Figure 21. TMS320C25-50 Interfaced to Full-Speed SRAM and Two Wait-State EPROM**



*Full-Speed SRAM in Program Space*

The cost and/or availability of non-volatile memory devices able to support TMS320C25-50 full-speed program execution may be prohibitive for some applications. (One such device is the Cypress Semiconductor 2K × 8 EPROM, part number CY7C291A-25.) The program code for Figure 21 can be stored in EPROM and self-booted into the SRAM devices at powerup for subsequent full-speed execution.

Table 9 shows the wait-state map for this circuit. Note that the READY generation logic for this arrangement is simple enough that inexpensive gates can be used for its implementation. Refer to the Ready Generation Techniques section earlier in this report for details of operation of the READY generation logic.

**Table 9. Wait-State Map for Circuit of Figure 21.**

| External Space | Address Range | Number of Wait-States |
|---|---|---|
| Program | 0000h–7FFFh | 2 |
| Program | 8000h–FFFFh | 0 |
| Data | 0000h–FFFFh | 0 |
| I/O | 0000h–000Fh | 1 |

The TI TMS27C64 EPROM devices reside in the two wait-state portion of program space at locations 0000h–1FFFh; the Micron MT5C6408-20 SRAM devices reside in the zero-wait portion of program space at locations 8000h–9FFFh.

*Timing Analysis*

Figure 22 shows the interface timing for accesses of the TMS27C64 EPROMs. Key timings are listed in Table 10. The output disable time of the TMS27C64 is too long to guarantee that no bus conflict will occur if an external write cycle follows a TMS27C64 read cycle; this is solved by buffering the data lines with TMS74ALS244A octal buffer ICs.

**Figure 22. Interface Timing for Accesses of TMS27C64-15 to the TMS320C25-50**

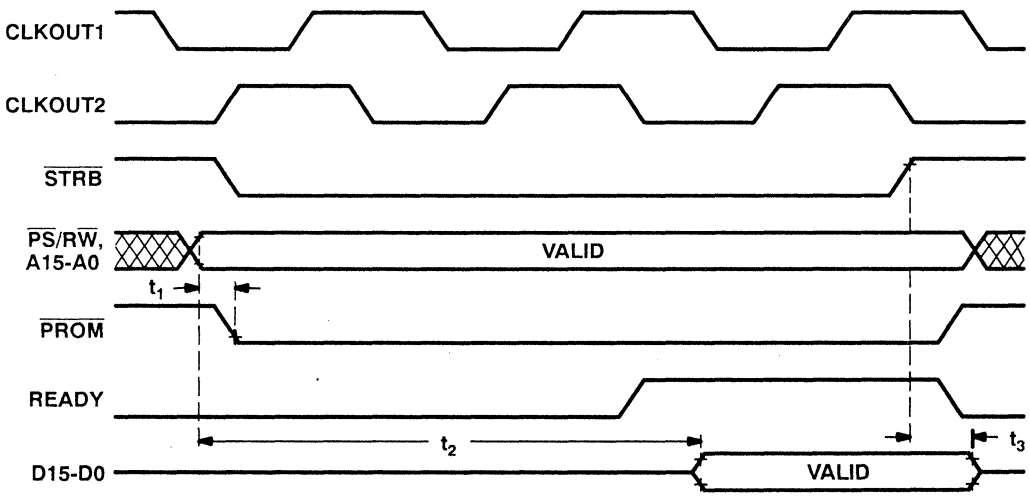


**Table 10. TMS27C64 Interface Timing Parameters**

| Parameter Name | Designation in Figure 22 | Time Duration |
|---|---|---|
| Address valid to $\overline{\text{PROM}}$ valid | $t_1$ | 5.8 ns (max) |
| $\overline{\text{PROM}}$ valid to TMS27C64 data valid | | 150 ns (max) |
| Address valid to TMS320C25 data valid | $t_2$ | 165.8 ns (max) |
| $\overline{\text{STRB}}$ high to TMS74ALS244A outputs high-Z | $t_3$ | 18.8 ns (max) |

As shown in Figure 11, data is valid on the TMS27C64 data lines 5.8 ns + 150 ns (max) after address becomes valid. The delay through the TMS74ALS244A buffers is 10 ns (max). Data is valid on the TMS320C25-50 data bus $t_1 + t_2 + 10 = 165.8$ ns (max) after address valid. Thus the inequality $t_1 + t_2 + 10$ (max) $< t_{a(A)} + N_{tc(C)}$ is satisfied; 165.8 ns < 29 ns + 2 * 80 ns. Note that $t_{c(C)}$ is assumed to equal 80 ns. The buffer outputs are set in the high-impedance state $t_3 = 5.8$ ns + 13 ns = 18.8 ns (max) after $\overline{\text{STRB}}$ goes high.

Figure 23 shows the interface timing for accesses of the MT5C6408 SRAMs. Key interface timing parameters are given in Table 11.

**Figure 23. Interface Timing for Accesses of the MT5C6408-20 to the TMS320C25-50**

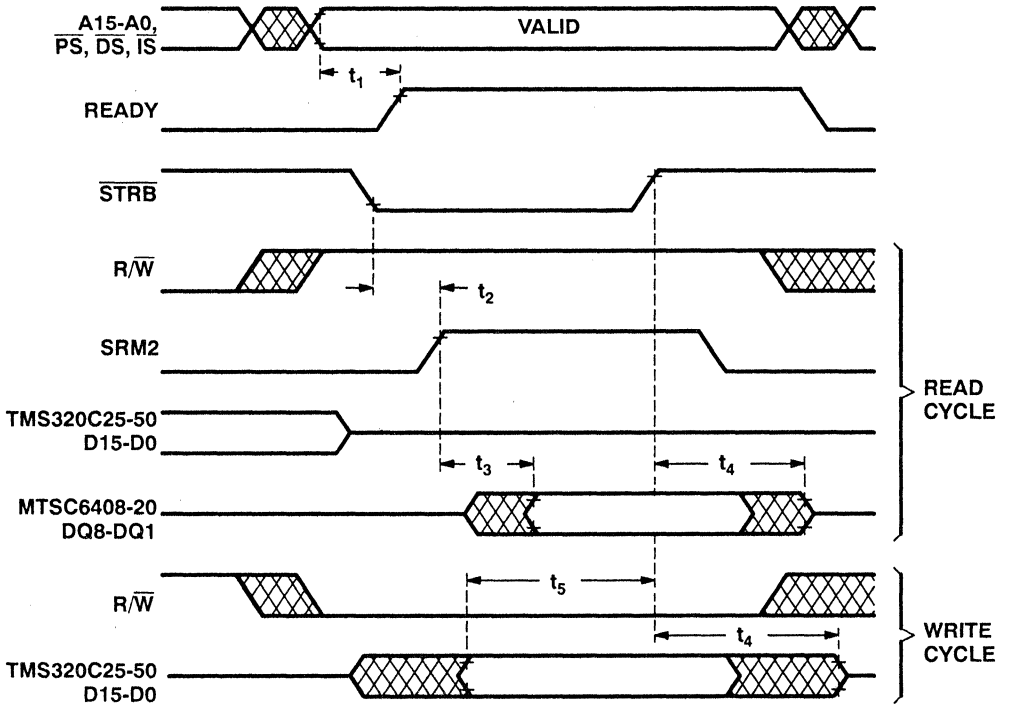


**Table 11. MT5C6408-20 Interface Timing Parameters**

| Read Cycle | | |
|---|---|---|
| **Parameter Name** | **Designation in 23** | **Time Duration** |
| Address valid to READY valid | $t_1$ | 9 ns (max) |
| Address valid to SRM2 valid | $t_2$ | 9 ns (max) |
| Address valid to SRM1 valid | | 9 ns (max) |
| SRM1/SRM2 valid to data valid | $t_3$ | 20 ns (max) |
| STRB high to data bus high-Z | $t_4$ | 15 ns (max) |
| **Write Cycle** | | |
| **Parameter Name** | **Designation in Figure 23** | **Time Duration** |
| Address valid to READY valid | $t_1$ | 9 ns (max) |
| Address valid to SRM2 valid | $t_2$ | 9 ns (max) |
| Address valid to SRM1 valid | | 9 ns (max) |
| Data valid before STRB high | $t_5$ | 23 ns (max) |
| STRB high to data bus high-Z | $t_4$ | 15 ns (max) |

The SRAMs are enabled if CE1 is low and CE2 is high. CE2 is high when IS, DS, and A15 are high. (Making use of the fact that the 3 external spaces are mutually exclusive and exhaustive, 1 gate delay is saved by using IS and DS rather than PS. This is crucial for satisfying the READY timing requirement.) CE1 is driven directly by STRB.

The function of the OE input of the MT5C6408s is the inverse of that of the WE input.

### *Read Cycle*

As shown in Table 11, both chip enable inputs are valid no later than 9 ns from address valid. Data is valid no later than 20 ns after CE1 and CE2 are valid, thus satisfying the condition $t_{a(SL)} \leq t_{a(SL)max}$. The outputs are tristated no later than 15 ns from STRB high.

### *Write Cycle*

As shown in Table 11, both chip enable inputs are valid no later than 9 ns from address valid. Data is valid 23 ns (min) prior to STRB going high, satisfying the MT5C6408 data setup time requirement of 12 ns (min). The outputs are tristated no later than 35 ns from STRB high.

The complete electrical specifications and additional information pertaining to the TMS320C25-50 may be found in the *Second-Generation TMS320 User's Guide.*[1]

## System Control Circuitry

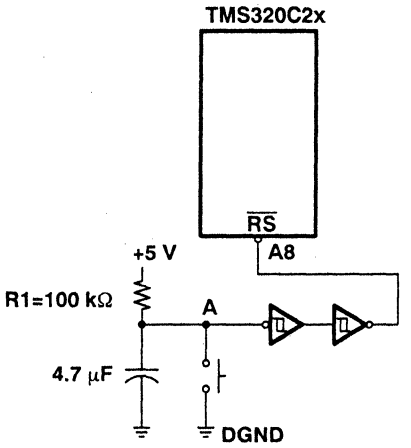
A system control circuitry performs important functions in system initialization and operation. A powerup reset circuit design and a crystal oscillator circuit design are presented in this section.

### Reset Circuit

The reset circuit shown in Figure 24 performs a power-up restart operation; i.e., the TMS320C2x is reset when power is applied. Note that the switch circuit must contain debounce

circuitry. Driving the RS signal low initializes the processor. Reset affects several registers and status bits. For a detailed description of the effect of reset on the processor status, refer to the *Second-Generation TMS320 User's Guide*.[1]

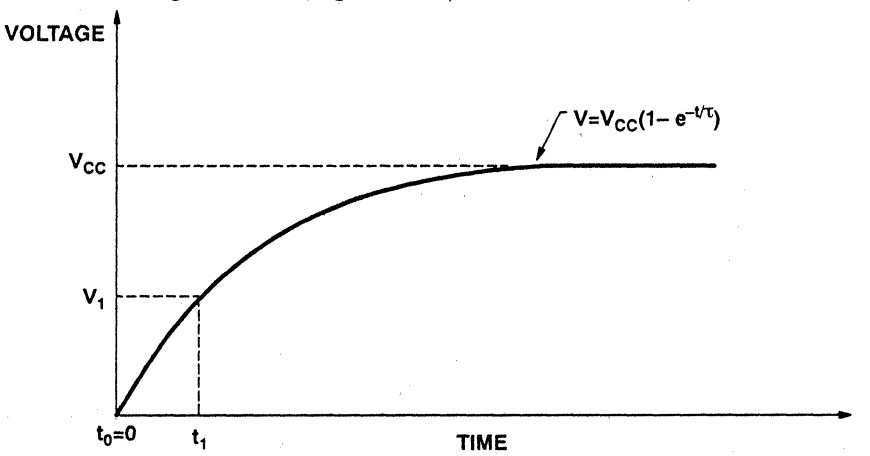
**Figure 24. Powerup Reset Circuit**




For proper system initialization, the reset signal must be applied for at least three CLKOUT cycles; i.e., 300 ns for a TMS320C2x operating at 40 MHz. Upon powerup, however, it can take up to hundreds of milliseconds before the system oscillator reaches a stable operating state. Therefore, the powerup reset circuit should generate a low pulse on the reset line until the oscillator is stable (between 100 and 200 ms). Once a proper reset pulse has been applied, processor operation begins at program memory location 0 which normally contains a branch (B) statement to direct program execution to the system initialization routine.

The voltage on node A is controlled by the $R_1C_1$ network (see Figure 24). After a reset, the voltage rises exponentially to the time constant $R_1C_1$, as shown in Figure 25.

**Figure 25. Voltage on the TMS320C2x Reset Pin**



The duration of the low pulse on the reset pin is approximately $t_1$, which is the time it takes for the capacitor $C_1$ to fully charge; i.e., 1.5 V. This is approximately the voltage at which the reset input switches from a logic level 0 to a logic level 1. The capacitor's voltage is given by

$$V = V_{CC}\left[1 - e^{-\frac{t}{\tau}}\right]; \tag{1}$$

where $\tau = R_1C_1$ is the reset circuit time constant.

Solving (1) for t gives:

$$t = -R_1R_1 \ln\left[1 - \frac{V}{V_{CC}}\right]. \tag{2}$$

Setting the following:

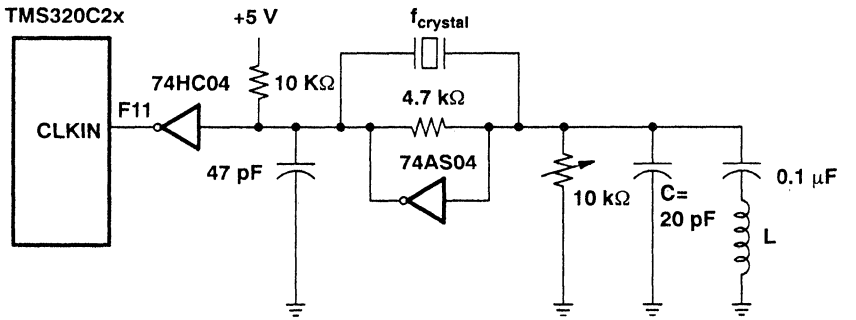
$R1 = 1\ M\Omega$
$C1 = 0.47\ \mu F$
$V = V_1 = 1.5\ V$
$V_{CC} = 5\ V$

gives $t = t_1 = 167$ ms. The Schmitt triggers shown in Figure 25 appropiately reshape the signal on node A.Therefore, the reset circuit of Figure 24 can generate a low pulse of an appropriate duration (167 ms) to ensure the stabilization of the system oscillator when most systems are powered.

## Crystal Oscillator Circuit

The crystal oscillator circuit shown in Figure 26 is suitable for providing the input clock signal to any TMS320C2x device except the TMS32020. Since crystals with fundamental oscillation frequencies of 30 MHz and above are not readily available, a parallel-resonant third-overtone oscillator is used. If a packed clock oscillator is used, oscillator design is of no concern.

**Figure 26. Crystal Oscillator Circuit**




The 74AS04 inverter in Figure 26 provides the 180-degree phase shift that a parallel oscillator requires. The 4.7-kΩ resistor provides the negative feedback that keeps the oscillator in a stable

state; i.e., the poles of the system are constrained in a narrow region about the j axis of the s-plane (analog domain). The 10-kΩ potentiometer is used to bias the 74AS04 in the linear region. This potentiometer is adjusted as follows: Before the crystal is placed on the system board, adjust the potentiometer so that the voltage at the input of the inverter is in the transition region between a logic level 0 and a logic level 1 (i.e., approximately 1.5 V). Then install the crystal.

In a third-overtone oscillator, the crystal fundamental frequency must be attenuated so that oscillation is at the third harmonic. This is achieved with an LC circuit that filters out the fundamental, thus allowing oscillation at the third harmonic. The impedance of the LC network must be inductive at the crystal fundamental frequency and capacitive at the third harmonic. The impedance of the LC circuit is given by:

$$Z(\omega) = \frac{\frac{L}{C}}{j\left[L - \frac{1}{\omega C}\right]} \tag{3}$$

Therefore, the LC circuit has a pole at:

$$\omega_p = \frac{1}{\sqrt{LC}} \tag{4}$$

At frequencies significantly lower than $\omega_p$, the $1/(\omega C)$ term in (3) becomes the dominating term while $\omega L$ can be neglected. This gives:

$$z(\omega) = j\omega L, \quad \text{for} \quad \omega << \omega_p \tag{5}$$

In (5), the LC circuit appears inductive at frequencies lower than ωp. On the other hand, at frequencies much higher than ωp, the ωL term is the dominant term in (3), and $1/(\omega C)$ can be neglected. This gives:

$$z(\omega) = \frac{1}{j\omega C} \qquad \text{for} \; \omega >> \omega_p \tag{6}$$

The LC circuit in (6) appears increasingly capacitive as frequency increases above $\omega_p$. This is shown in Figure 27, which is a plot of the magnitude of the impedance of the LC circuit of Figure 26 versus frequency.

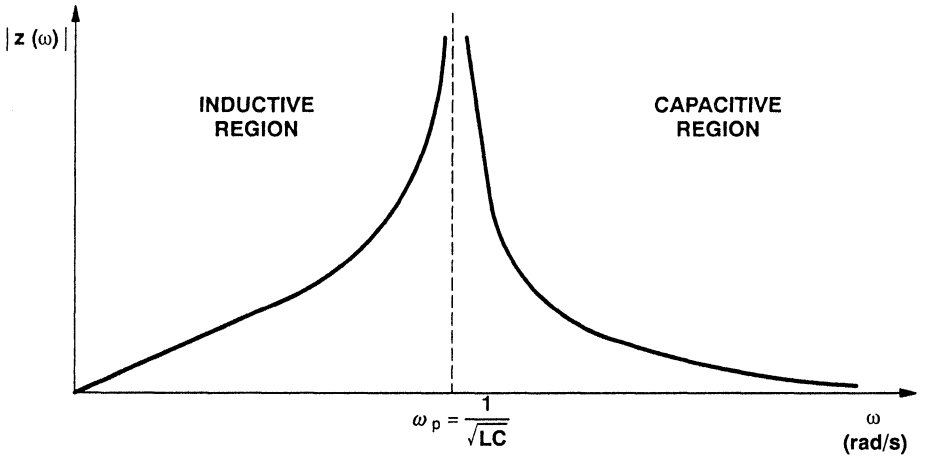
Based on the discussion above, the design of the LC circuit proceeds as follows: Choose the pole frequency $\omega_p$ approximately halfway between the crystal fundamental and the third harmonic. The circuit now appears inductive at the fundamental frequency and capacitive at the third harmonic.

In the oscillator of Figure 26, choose $\omega_p$ = 166.5 rads/s for the 40.96 MHz design or $\omega_p$ = 223.6 for the 51.2 MHz design. These angular frequencies lie approximately halfway between the respective fundamentals and third harmonics. Choose C = 20 pF. The appropriate value of L may then be computed using (4). Values of L for three different TMS320C2x devices operating at different frequencies are tabulated in Table 12.

**Table 12. Values of $f_{crystal}$ and L for TMS320C2x Devices**

| | $f_{crystal}$ (MHz) | L (μH) |
|---|---|---|
| TMS320C25 | 40.96 | 1.8 |
| TMS320C25-50 | 51.20 | 1.0 |
| TMS320E25 | 40.96 | 1.8 |

**Figure 27. Magnitude of the Impedance of the Oscillator LC Network**




The 0.1 μF capacitor in series with the 1.8 μH inductor is a coupling capacitor, requiring no DC path to ground. The 74AS04 inverter is included to shorten the rise and fall times of the waveform generated by the oscillator.

Consider the case where the TTL inverter goes low. In this case, the current flowing through the 10-kΩ resistor is less than 5 V/10-kΩ = 0.5 mA. This is an acceptable current level since the 74AS04 inverter can sink up to 20 mA.

The output of the oscillator drives the CLKIN input of the TMS320C2x, thus providing the four phases required for each machine cycle. With a 40.96 MHz input clock frequency, the TMS320C2x machine cycle is 97.6 ns.

In summary, the system control circuitry performs functions that, while often overlooked, are critical for proper system initialization and operation. The powerup reset circuit assures that a reset of the part occurs only after the oscillator is running and stabilized. The oscillator circuit described allows the use of third-overtone crystals that are more readily available at frequencies above 20 MHz.

## Interfacing Peripherals

Most DSP systems implement some amount of I/O using peripherals in addition to any memory included in the system. Quite commonly this includes analog input and output, which can

be performed through the parallel and serial I/O ports on the TMS320C2x. In this section, hardware interfaces of the TMS320C2x to a codec, an analog-to-digital converter (A/D), and a digital-to-analog converter (D/A) are described. Interfacing TMS320 devices to the Texas Instruments TLC32040 Analog Interface Chip is described in the applications report *Interfacing the TMS320 Family to the TLC32040 Family* found in this book.

## Combo-Codec Interface

In speech, telecommunications, and many other applications that require low-cost analog-to-digital and digital-to-analog converters, a combo-codec may be used. Combo-codecs are single-chip pulse-code-modulated encoders and decoders (PCM codecs). They are designed to perform the encoding (A/D conversion) and decoding (D/A conversion), as well as the antialiasing and smoothing filtering functions. Since combo-codecs perform these functions in a single 300-mil DIP package at low cost, they are extremely economical for providing system data conversion functions. The design presented here uses a Texas Instruments TCM29C16 codec, interfaced using the serial port of the TMS320C2x.

### *TMS320C2x Serial Port*

The TMS320C2x serial port provides direct synchronous communication with serial devices. The interface signals are compatible with codecs and other serial components so that minimum external hardware is required. Externally, the serial port interface is implemented using the following pins on the TMS320C2x:

- DX (transmitted serial data)
- CLKX (transmit clock)
- FSX (transmit framing synchronization signal)
- DR (received serial data)
- CLKR (receive clock)
- FSR (receive framing synchronization signal)

Data on DX and DR are clocked by CLKX and CLKR, respectively. These clocks are only required during serial transfers. Note that this is different from the TMS32020 serial port in which the clocks must be present at all times if the serial port is being used. Also, the TMS320C2x serial port is double-buffered while that of the TMS32020 is not.

Serial port transfers are initiated by framing pulses on the FSX and FSR pins for transmit and receive operations respectively. For transmit operations, the FSX pin can be configured as an input or output. This option is selected by the transmit mode (TXM) bit of status register ST1.[1] In this design, FSX is assumed to be configured as an input; therefore, transmit operations are initiated by a framing pulse on the FSX pin. Upon completion of receive and transmit operations, an RINT (serial port receive interrupt) and an XINT (serial port transmit interrupt) are generated, respectively.

The format (FO) bit of status register ST1 is used to select the format (8-bit byte or 16-bit word) of the data to be received or transmitted. For interfacing the TMS320C2x to a codec, the format bit should be set to one, formatting the data in 8-bit bytes.[1]

After the information from the codec is received by the TMS320C2x, the μ- or A-law companded data must be converted back to a linear representation for use in the TMS320C2x. Software companding routines appropriate for use on the TMS320C2x are provided in the book, *Digital Signal Processing with the TMS320 Family Volume 1.*[2]

The software required to initialize the TMS320C2x-codec interface is shown next. The initialization routine should include the following:

```
INIT        DINT            ; Disable interrupts
            FORT   1        ; Set 8-bit data format
            LACK   10h
            LDPK   0
            SACL   DMA4     ; Enable RINT (through IMR)
            *
            *
            *
            EINT            ; Enable interrupts
```

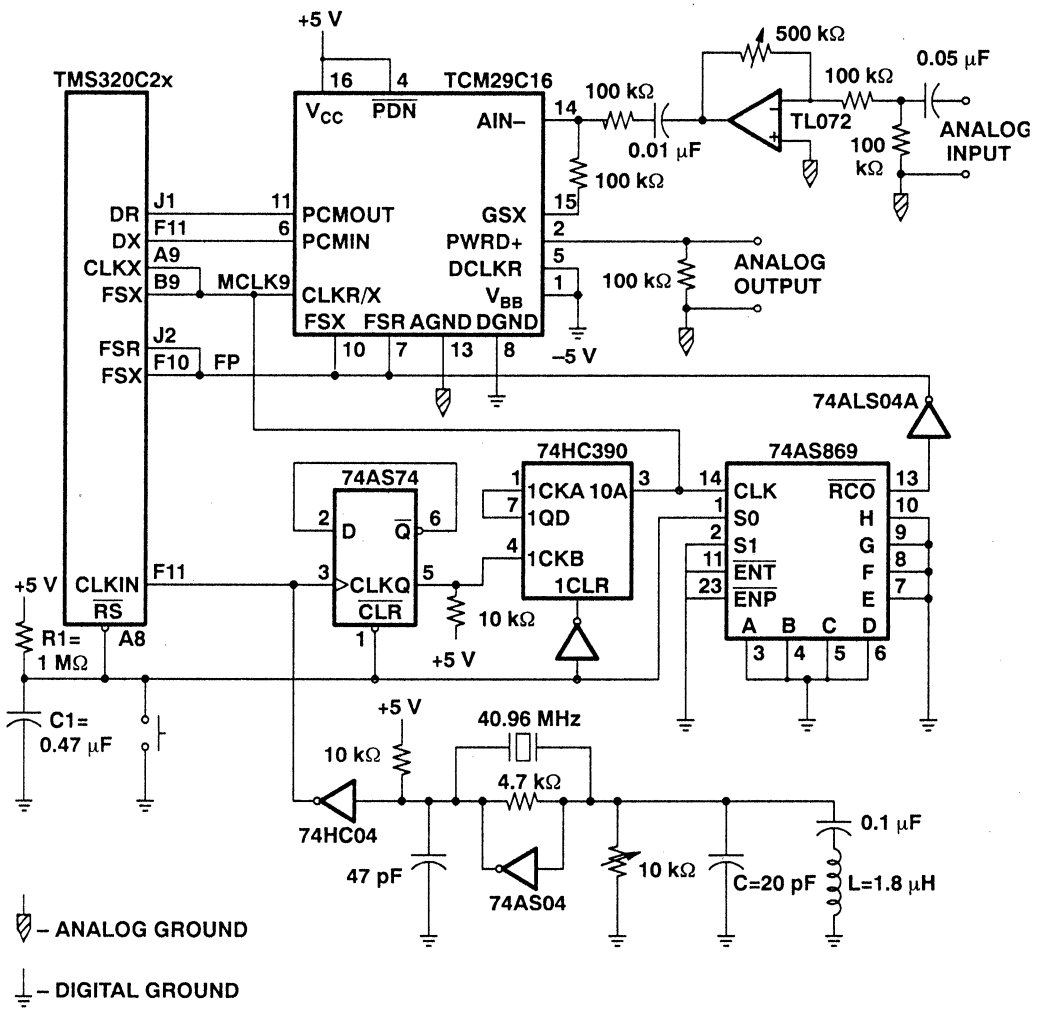
Note that since reset initializes the TXM (transmit mode) and FSM (frame synchronization mode) bits to the values required by this interface, it was not necessary to explicitly initialize these values in the routine shown above. However, in digital communications with peripherals/devices/ports ($T_1$ trunks) that do not require a framing pulse for every byte/word transmitted, the FSM bit must be set to 0 using the RFSM instruction.[1]

The interrupt mask register (IMR) located at data memory location 4h of the TMS320C2x data memory is used to enable the serial port receive interrupts (RINT). To access that memory location, the data page pointer must be set to zero. Also, the data page pointer must be initialized after reset since its contents are random at powerup. A value of 10h in the IMR enables only the RINT; all other interrupt sources are disabled.

Interrupts are disabled upon reset. Before exiting the initialization routine, interrupts are reenabled with the EINT instruction.

The hardware interface between the TMS320C2x and the TCM29C16 combo-codec is shown in Figure 28.

**Figure 28. Interface of the TMS320C2x to the TCM29C16 Codec**




*Clock Divider Circuit*

A combo-codec configured in the fixed-data-rate mode requires the following external clock signals:

A 2.048-MHz clock to be used as the masterclock, and

8-kHz framing pulses required to initialize the data transfers.

Both of these signals can be derived from the 40.96 MHz system clock with appropriate divider circuitry. This is the primary justification for selecting 40.96 MHz as the system clock fre-

quency. The clock divider circuit consists of a 74AS74 D-type flip-flop, a 74HC390 decade counter, and a 74AS869 8-bit up/down counter. The hardware connections between these devices are shown in Figure 28.

To generate the 2.048-MHz master clock for the combo-codec, a division by 20 of the 40.96-MHz system clock is required. The 74HC390 contains on-chip two divide-by-2 and two divide-by-5 counters. Since the 74HC390 cannot be clocked with frequencies above approximately 27 MHz, a 74AS74 configured as a T-type flip-flop is used. This implements a divide-by-2 of the 40.96-MHz clock, thus making the output of the 74AS74 slow enough (20.48 MHz) to properly clock the 74HC390. The 10-kΩ pullup resistor shown in Figure 28 is used to ensure the compatibility between the logic levels of the TTL (74AS74) and HCMOS (74HC390) devices.

The 74HC390 is first used to implement a divide-by-5, which appears at the output pin $1Q_D$ (pin #7) of the 74HC390 (see Figure 28). This in turn drives the divide-by-2 counter, at the output of which (pin $1Q_A$) the 2.048 MHz clock appears. Note that the divide-by-5 precedes the divide-by-2 because the codec requires a clock with a minimum duty cycle of 40 percent, while the output of the divide-by-5 has a duty cycle of only 20 percent. By following the divide-by-5 counter with the divide-by-2, the duty cycle at the output of the 74HC390 is 50 percent.

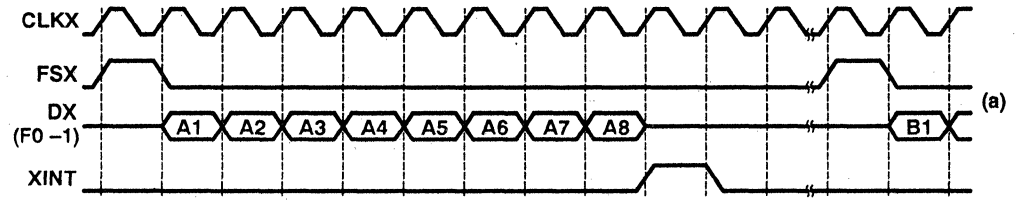
The 74AS869 is configured to count down (S0 = 1 and S1 = 0 in Figure 28); therefore, the counting sequence is 255, 254, ..., 1, 0, 255, ..., and so on. The ripple carry output generates a low-level pulse while the count is zero. The duration of this pulse is one input clock cycle, i.e., 488 ns. The frequency of the ripple carry output is 2.048 MHz/256 = 8 kHz. By inverting this signal, positive pulses at 8 kHz are generated. These pulses are used by the TMS320C2x and codec as framing pulses to initiate data transfers.

### *TMS320C2x-Codec Interface*

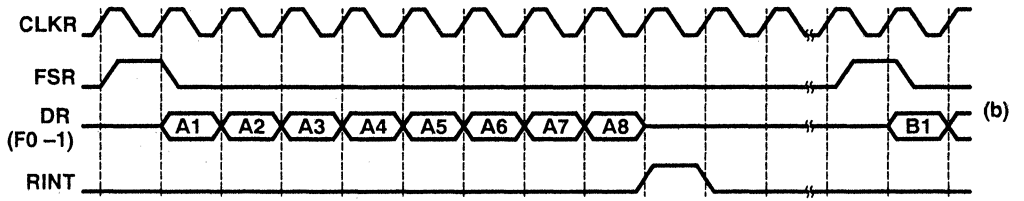
The TMS320C2x interfaces directly to the codec, as shown in Figure 28, with no additional logic required. The PCM μ-law data generated by the codec at the PCMOUT pin is read by the TMS320C2x from the data receive (DR) pin, which is internally connected to the receive serial register (RSR).[1] The data transmitted from the data transmit (DX) pin of the TMS320C2x is received by the PCMIN input of the codec. During the digital-to-analog conversion, this data is converted from μ-law PCM to linear. The resulting analog waveform is lowpass-filtered by the codec's internal smoothing filter. Therefore, no additional filtering is required at the codec output (PWRO+).

The timing diagram of the TMS320C2x-codec interface is shown in Figure 29.

**Figure 29. Interface Timing of the TMS320C2x to the TCM29C16 Codec**




(a) DATA TRANSMITTED BY THE TMS320C25




(a) DATA RECEIVED BY THE TMS320C25

As indicated in Figure 29, both the transmit and receive operations are initiated by a framing pulse on the FSX and FSR pins of the TMS320C2x and the codec. The receive and transmit interrupts shown in Figure 29 occur only if they are enabled. Note that Figure 29 corresponds to the burst-mode serial port operation of the TMS320C2x.[1] Continuous-mode operation using framing pulses or without framing pulses is also available.

*Analog Input*

The level of the analog input signal is controlled using the TL072 opamp connected in the inverting configuration (see Figure 28). Using the 500-kΩ potentiometer, the gain of this circuit can be varied from 0 to 5. The output of the 0.01-μF coupling capacitor drives the TCM29C16's internal opamp. This opamp is connected in the inverting configuration with unity gain (feedback and input impedances having the same value of 100 kΩ).

In summary, codecs, combo-codecs in particular, are most effective in serving DSP system data-conversion requirements. These inexpensive devices interface directly to the TMS320C2x, occupy minimal board space, and perform both filtering and data conversion functions. Codecs interface to the TMS320C2x by means of the serial port and provide a companded, PCM-coded digital representation of analog input samples. This PCM code is easily translated into a linear form by the TMS320C2x for use in processing. Interface to the codec on the serial port is initialized by a simple software routine in the TMS320C2x.

## Interfacing an Analog-to-Digital (A/D) Converter

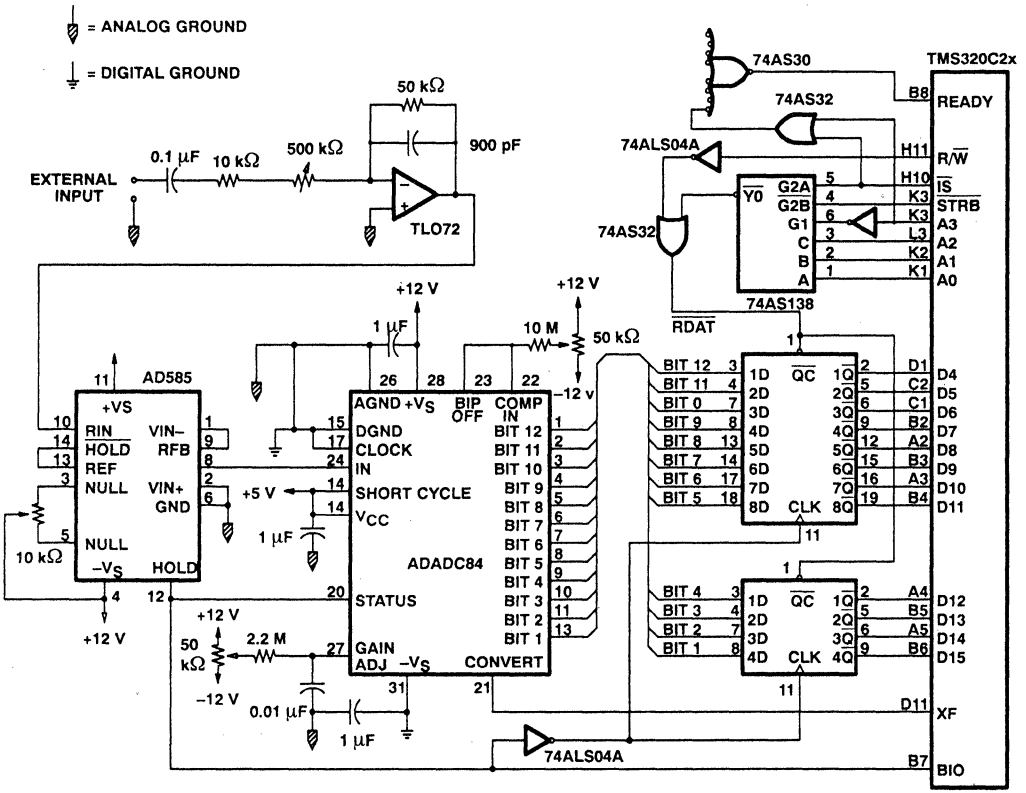
Many digital signal processing applications require a higher level of signal quality than that offered by the eight companded bits of a combo-codec. For these applications, linear analog-to-dig-

ital converters with 10, 12, or 14 bits are commonly used. The improved signal quality obtained with these converters, however, is accompanied by increased system complexity and higher cost.

The hardware interface of a 12-bit linear analog-to-digital (A/D) converter to the TMS320C2x is discussed in this subsection. In this design, the A/D is mapped into the input/output (I/O) space of the TMS320C2x. The distinction between the I/O space and the program and data spaces is made by using the $\overline{IS}$ pin. This pin goes active (low) when the I/O space is accessed. The TMS320C2x space contains 16 ports that can be read from or written to. These ports are accessed with the IN and OUT instructions.[1]

The hardware design of this interface is shown in Figure 30. This design utilizes an antialiasing (lowpass) filter, the Analog Devices' AD585 sample-and-hold and ADADC84 analog-to-digital converter, two 74AS534 octal D-type flip-flops, plus additional logic to generate the READY signal.

**Figure 30. Interface of the ADADC84 to the TMS320C2x**




The design of Figure 30 consists of two sections: the analog-to-digital conversion and the interface to the TMS320C2x. Each of these sections is considered separately.

## Analog-to-Digital Conversion

The analog-to-digital conversion section of this interface performs the function of sampling and coding the input waveform. This circuit consists of the antialiasing filter, the sample-and-hold, and the analog-to-digital converter.

To avoid distortion during an analog-to-digital conversion, the sampling theorem states that the analog signal must contain no frequency components greater than half the sampling frequency. If this condition is not met, distortion occurs in the form of aliasing; i.e., high-frequency components are superimposed on the low frequencies of the signal spectrum. To avoid this phenomenon, an antialiasing (lowpass) filter is used.

In the design of Figure 30, the antialiasing filter is implemented using a TL072 opamp connected in the inverting configuration. The gain of the opamp is determined by the values of two fixed resistors (10 kΩ and 50 kΩ) and a 500-kΩ potentiometer. The resistance of the potentiometer inversely varies the gain of the opamp. The minimum gain of 0.098 (50 kΩ/510 kΩ) is reached when the potentiometer is 500 kΩ. The maximum gain of 5 (50 kΩ/10kΩ) is achieved when the potentiometer is decreased to zero resistance.

To satisfy the sampling theorem, the cutoff frequency of the antialiasing filter must be less than half the sampling rate. In the design of Figure 30, the 900 pF capacitor in the feedback path introduces a pole at the frequency f defined by:

$$f = \frac{1}{2\pi RC} = \frac{1}{2\pi(50k\Omega)(0.9)nF)} = 3.5kHz$$

After 3.5 kHz, the frequency response of the filter drops by 6 dB per decade. This rejection, however, may not be adequate for some applications. In such cases, a lowpass filter of higher order is required. Such a filter is presented in the next subsection.

The output of the antialiasing filter is connected to the input of the AD585 sample-and-hold, which is configured for a gain of –1. The operation of this device is controlled by the HOLD input. When HOLD is low, the output of the sample-and-hold ($V_{OUT}$) follows the input (lowpass version of the external input). When HOLD is high, the output stays constant. The time from HOLD high to output stable is referred to as the aperture time, specified as 35 ns for the AD585.

A/D conversions are implemented by the ADADC84, a 12-bit linear A/D converter in which data is represented in complementary two's-complement form. A conversion begins when the CONVERT input goes high. The XF (external flag) output of the TMS320C2x is used to drive the CONVERT input. Since the XF pin is software controlled, the TMS320C2x internal timer may be used to generate programmable sampling rates. This is discussed in more detail later.

When CONVERT goes high, the ADADC84 begins the conversion and STATUS goes high. This puts the AD585 in the hold mode. The A/D conversion lasts for 10 μs, with the MSB decision made approximately 820 ns after STATUS goes high. Note that the aperture time of the AD585 is only 35 ns, and as a result, the input to the A/D converter is stable well before the time the MSB decision is made. The LSB decision is made at least 40 ns before STATUS goes low. When STATUS goes low, the AD585 enters the sample mode with a gain of –1; i.e., the output follows the inverted

input waveform. As shown in Figure 30, the $\overline{BIO}$ pin of the TMS320C2x is connected to STATUS. By polling $\overline{BIO}$, the TMS320C2x can detect when an A/D conversion is completed.

The falling edge of STATUS generates a rising edge at the clock inputs of the 74AS534s. This rising edge clocks the ADADC84 data into the 74AS534s. Since the LSB decision is made 40 ns before STATUS goes low, the 3 ns setup time for the 74AS534s is met. Since the 74AS534s are inverting-type flip-flops, the ADADC84 outputs are complemented to give data in two's-complement form. This data, however, does not appear on the TMS320C2x data bus until the output buffers of the 74AS534s are enabled.

### *Interface to the TMS320C2x*

The interface logic in Figure 30 is used to perform the following functions:

- Generate READY, and
- Enable the output buffers of the 74AS534s so that the TMS320C2x can read the data from the A/D conversion

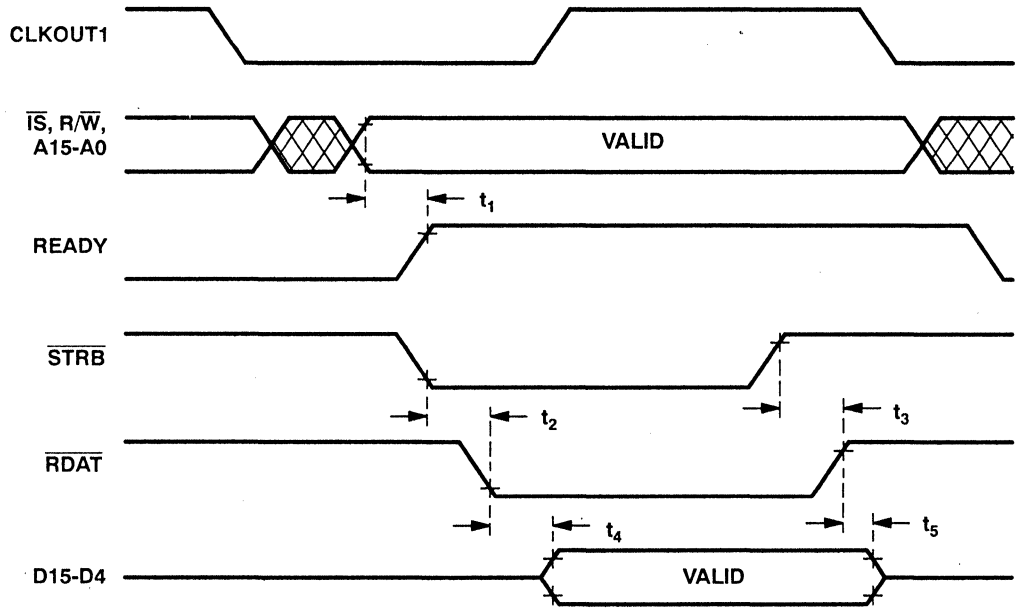
To meet the TMS320C2x READY timing requirements, two separate address decoding schemes are used to implement these two functions. One decoding scheme is used for READY, and a second is used to enable the I/O-mapped devices.

The address decoding for READY is implmented with the 74AS32 positive-OR gate. The output of the 74AS32 goes low when both $\overline{IS}$ and A3 go low; i.e., access to ports 0 through 7 is requested. This scheme generates READY for devices that do not require wait-states. I/O devices that generate one or more wait-states can utilize ports 8 through 15.

To enable the I/O devices, a 74AS138 is used. Outputs $\overline{Y0}$ through $\overline{Y7}$ of the 74AS138 can be used to enable the devices 0 through 7, respectively. In Figure 30, $\overline{Y0}$ is used to enable a read from the A/D converter. Note that $\overline{Y0}$ is ORed with the inverted R/$\overline{W}$. This prevents the bus conflict that occurs if the TMS320C2x writes to port 0.

The timing diagram of a TMS320C2x read from port 0 is shown in Figure 31.

**Figure 31. Interface Timing of the ADADC84 to the TMS320C2x**




Time $t_1$ in Figure 31 indicates the time from valid address to READY high. This is less than 10.8 ns, the maximum propagation delay through the READY generation logic. Therefore, the 18-ns READY timing requirement (at 40 MHz) is met.

RDAT in Figure 31 is used to enable the output buffers of the 74AS534s. $\overline{\text{RDAT}}$ goes active (low) no later than $t_2 = t_{p(74AS138)} + t_{p(74AS32)} = 14.3$ ns after $\overline{\text{STRB}}$ goes low ($\overline{\text{STRB}}$ is used to enable the 74AS138). With a low level on the output control ($\overline{\text{OC}}$) of the 74AS534s, valid data appears on the TMS320C2x data bus within $t_4 = 10$ ns. The worst-case access time is $t_2 + t_4 = 24.3$ ns from $\overline{\text{STRB}}$ going low, which is less than the 27 ns required by the TMS320C2x.

When $\overline{\text{STRB}}$ goes high, $\overline{\text{RDAT}}$ follows within $t_3 = 13.3$ ns. With a high logic level on the output control ($\overline{\text{OC}}$), the output buffers of the 74AS534s enter a high-impedance state within $t_5 = 6$ ns. Since $t_3 + t_5 = 19.3$ ns after $\overline{\text{STRB}}$ goes high, the 74AS534s have entered a high-impedance state, and no bus conflict will occur if a write cycle follows the present read cycle.

Table 13 summarizes the most critical timing parameters of the ADADC84 interface to the TMS320C2x.

**Table 13. Timing Parameters of the ADADC84 Interface to the TMS320C2x**

| Description | Symbol Used in Figure 3 | Value |
|---|---|---|
| Address valid to READY valid | $t_1$ | 10.8 ns (max) |
| $\overline{\text{STRB}}$ low to $\overline{\text{RDAT}}$ low | $t_2$ | 14.3 ns (max) |
| $\overline{\text{STRB}}$ high to $\overline{\text{RDAT}}$ high | $t_3$ | 13.3 ns (max) |
| Progagation delay through the 74AS534 ($\overline{\text{QC}}$ to $\overline{\text{Q}}$) | $t_4$ | 10.0 ns (max) |
| 74AS534 disable time | $t_5$ | 6.0 ns (max) |

*Controlling A/D Conversions with the TMS320C2x Timer*

The TMS320C2x timer can generate periodic interrupts that may be used to set the A/D sampling frequency. The TMS320C2x timer logic consists of a 16-bit timer register and a 16-bit period register. At every CLKOUT1 cycle, the timer register is decremented by one. When the count reaches zero, a timer interrupt (TINT) is generated. In the next cycle, the contents of the period (PRD) register are loaded into the timer register. Therefore, a timer interrupt is generated every PRD + 1 cycle of CLKOUT1, and the frequency of these interrupts is CLKOUT1/(PRD + 1).

As an example, consider a TMS320C2x operating at 40 MHz. The design of Figure 30 is utilized to interface the A/D converter to the TMS320C2x. A sampling rate of 10 kHz is desired.

To generate timer interrupts at the 10 KHz sampling rate, the value of the period register is calculated as follows: Since

$$f_s = \frac{\text{CLKOUT1}}{\text{PRD} + 1}$$

the period register is

$$\text{PRD} = \frac{\text{CLKOUT}}{f_s} - 1$$

With CLKOUT1 = 10 MHz and $f_S$ = 10 kHz, the value of the period register is PRD = 999. By loading the period register (data memory location 3) with 999, timer interrupts (if enabled) occur at a 10 kHz frequency. This can be implemented with the following TMS320C2x source code:

```
LDPK   0        ; Point to Data Page #0
LALK   999      ; ACC 999
SACL   DMA3     ; Period Register ACC
LACL   8        ; Enable TINT
OR     DMA4     ; through
SACL   DMA4     ; the IMR
```

To start the A/D conversion, the interrupt service routine must generate a positive pulse on the XF output. This can be implemented with the following code:

```
ISE    SXF      ; Set external flag (XF)
       RXF      ; Clear external flag (XF)
       EINT     ; Enable interrupts
       RET
```

Note that upon entering the interrupt service routine, the interrupts are disabled. Interrupts are reenabled by the EINT instruction just before exiting the interrupt service routine. Also, the conversion pulse that this routine generates is 100 ns long, easily meeting the 50-ns minimum conversion pulse width required by the ADADC84.

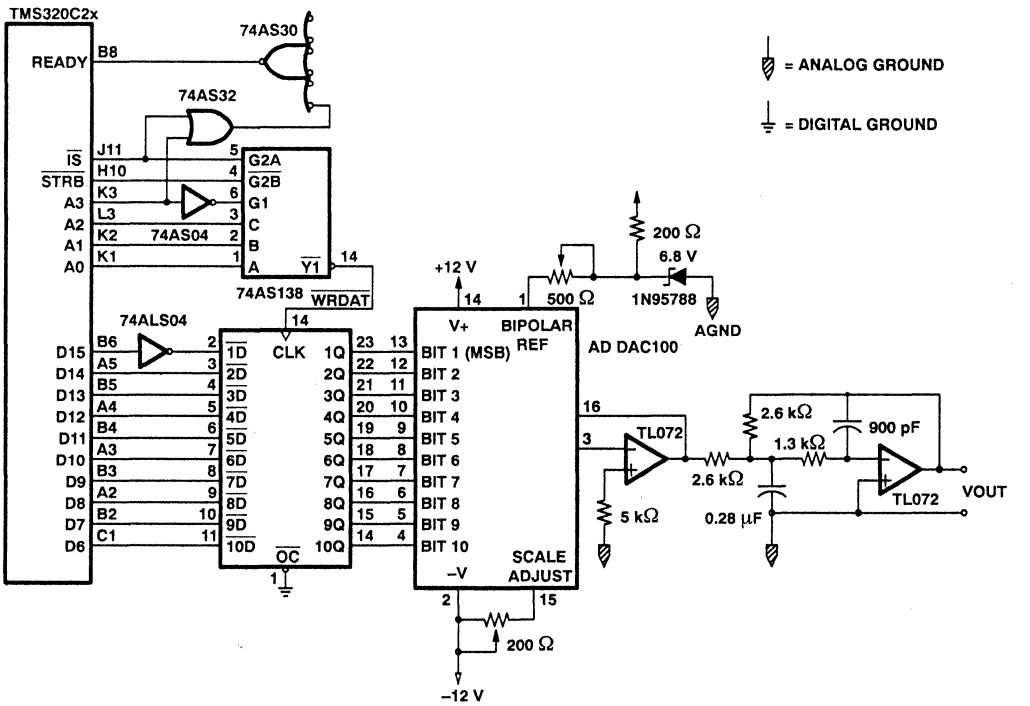
To summarize, 10-bit to more than 14-bit linear A/D converters are often used to perform data conversions in DSP systems that require more resolution than is provided by codecs. The circuit shown in Figure 30 describes the interface of an A/D conversion subsystem to the

TMS320C2x. This subsystem contains antialiasing filters, a sample-and-hold circuit, and a 12-bit A/D converter. Communication with the TMS320C2x is provided via the I/O space. The A/D converter is isolated from the processor's data bus by high-impedance buffers when data transfers are not being performed. The TMS320C2x's internal timer is used to establish the A/D sample rates, thus reducing system logic requirements.

## Interfacing a Digital-to-Analog (D/A) Converter

This subsection discusses the hardware interface of a 10-bit digital-to-analog converter to the TMS320C2x. The design, shown in Figure 32, utilizes the Analog Device's ADDAC100 digital-to-analog converter, a 74AS822 10-bit flip-flop, a smoothing filter, plus additional logic to generate READY.

**Figure 32. Interface of the ADDAC100 to the TMS320C2x**




This design consists of three sections: the interface to the TMS320C2x, the D/A converter, and the smoothing filter. Each of these sections is considered separately.

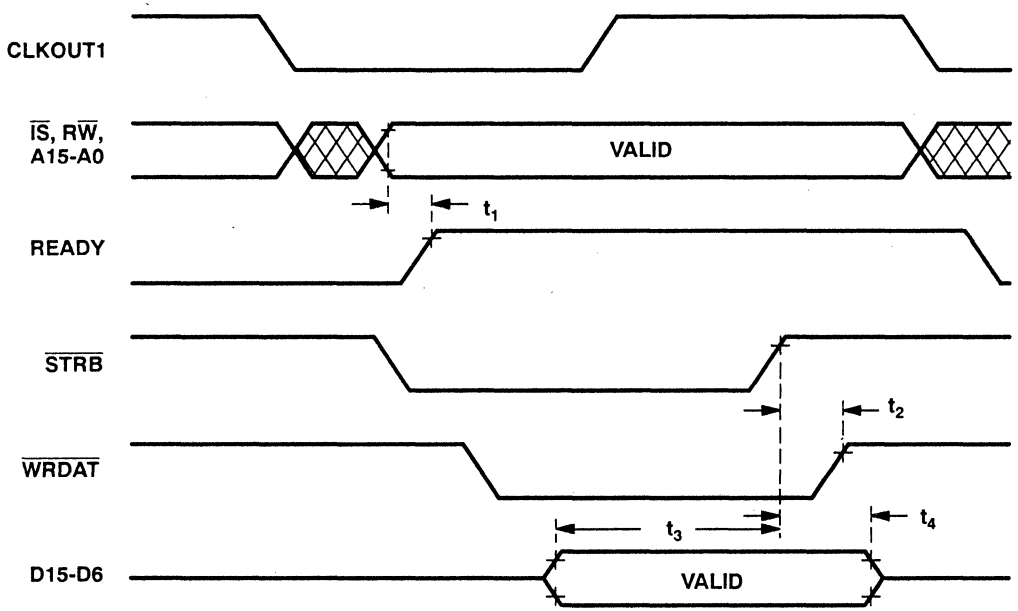
### *Interface to the TMS320C2x*

The 74AS822 is used to latch the data from the TMS320C2x. Since the output control ($\overline{OC}$) of the 74AS822 is always active (grounded), the latched data is available at the inputs of the D/A converter immediately following a write from the TMS320C2x. In bipolar mode, the ADDAC100 accepts data in complementary offset binary form. By inverting the MSB of the two's-complement

data from the TMS320C2x, the data input to the 74AS822 is converted to offset binary form. This data is inverted by the 74AS822 so that the input to the ADDAC100 becomes complementary offset binary form.

The circuit shown in Figure 32 utilizes the same address decoding technique used for the analog-to-digital converter interface. This technique maps devices that require no wait-states into ports 0 through 7. Ports 8 through 15 are used for devices that require one or more wait-states. In this design, the D/A converter is mapped into port 1 of the TMS320C2x I/O space. The timing diagram for a write to the D/A is shown in Figure 33.

**Figure 33. Interface Timing of the ADDAC100 to the TMS320C2x**




When port 1 is addressed, $\overline{\text{WRDAT}}$ goes low. No later than $t_2$ = 7.5 ns after $\overline{\text{STRB}}$ goes high, $\overline{\text{WRDAT}}$ follows. This rising edge of $\overline{\text{WRDAT}}$ clocks the data into the 74AS822. The minimum setup time for the data before $\overline{\text{WRDAT}}$ goes high is $t_3$ min + $t_2$ min (see Figure 33). Time $t_3$ min is the minimum setup time for the TMS320C2x data before $\overline{\text{STRB}}$ goes high (30 ns), minus the maximum propagation delay through the 74ALS04 (11 ns). Time $t_2$ min is the minimum propagation delay through the 74AS138 (2 ns). Therefore, the minimum setup time for the data before $\overline{\text{WRDAT}}$ goes high is 21 ns, which is greater than the 6-ns minimum setup time required by the 74AS822.

Table 14 summarizes the most critical timing parameters of the ADDAC100 interface to the TMS320C2x.

**Table 14. Timing Parameters of the ADDAC100 Interface to the TMS320C2x**

| Description | Symbol Used in Figure 3 | Value |
|---|---|---|
| Address valid to READY valid | $t_1$ | 10.8 ns (max) |
| STRB high to WRDAT high | $t_2$ | 7.5 ns (max) |
| Data setup time before STRB high | $t_3$ | 19.0 ns (min) |
| Data setup time before WRDAT high | $t_3 + t_2$ | 21.0 ns (min) |
| Data hold time from STRB high | $t_4$ | 15.0 ns (min) |
| Data hold time from WRDAT high | $t_4 - t_2$ | 7.5 ns (min) |

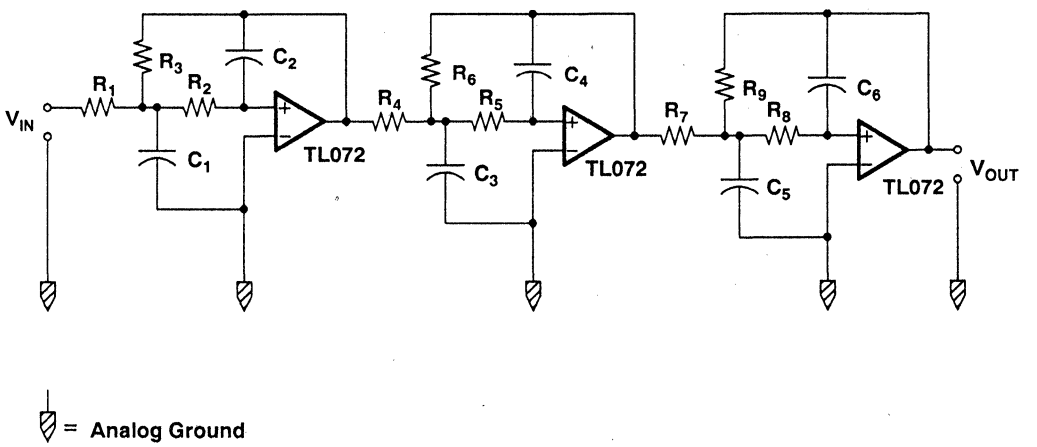
### *D/A Converter*

The ADDAC100 10-bit digital-to-analog converter converts a digital input to an output current. The standard current-to-voltage conversion is implemented using the TL072 opamp. This is the opamp closest to the ADDAC100 in Figure 32. The offset and gain ajustments are implemented with the 500-Ω and 200-Ω potentiometers, respectively.

### *Smoothing Filter*

The output of the ADDAC100 contains high-frequency components to be removed by the smoothing filter. In the design of Figure 32, this filter is implemented with the TL072 opamp configured to implement a second-order lowpass filter with a cutoff frequency around 1.7 KHz. For some applications, however, a rejection of 12 dB per decade is not adequate. A design that implements a sixth-order lowpass filter is shown in Figure 34. This design is a cascade of three opamps, each implementing a second-order section.

**Figure 34. Sixth-Order Lowpass Filter Used for Antialiasing and Smoothing Filter Operations**



The design of Figure 34 is used to implement the antialiasing and smoothing filtering operations in the TMS32010 Analog Interface Board. The cutoff frequency of this filter depends on the values of the passive components. The values of these components for several cutoff frequencies are shown in Table 15.[3]

**Table 15. Lowpass Filter Component Values for Various Frequencies**

| f | 1.7 kHz | 4.7 kHz | 7.7 kHz | 10 kHz | 12 kHz | 16 kHz | 20 kHz |
|---|---|---|---|---|---|---|---|
| R1 | 2.588 | 2.588 | 2.588 | 2.588 | 2.588 | 2.588 | 2.588 |
| C1 | 0.280 | 0.101 | 0.0617 | 0.0475 | 0.0396 | 0.0297 | 0.0238 |
| R2 | 1.294 | 1.294 | 1.294 | 1.294 | 1.294 | 1.294 | 1.294 |
| R3 | 2.588 | 2.588 | 2.588 | 2.588 | 2.588 | 2.588 | 2.588 |
| C2 | 0.00936 | 0.00339 | 0.00207 | 0.00160 | 0.00133 | 0.000995 | 0.000796 |
| R4 | 7.071 | 7.071 | 7.071 | 7.071 | 7.071 | 7.071 | 7.071 |
| C3 | 0.0375 | 0.0136 | 0.00827 | 0.00637 | 0.00531 | 0.00398 | 0.00318 |
| R5 | 3.536 | 3.536 | 3.536 | 3.536 | 3.536 | 3.536 | 2.536 |
| R6 | 7.071 | 7.071 | 7.071 | 7.071 | 7.071 | 7.071 | 7.071 |
| C4 | 0.00936 | 0.00339 | 0.00207 | 0.00160 | 0.00133 | 0.000995 | 0.000796 |
| R7 | 9.659 | 9.659 | 9.659 | 9.659 | 9.659 | 9.659 | 9.659 |
| C5 | 0.0201 | 0.00726 | 0.00443 | 0.00341 | 0.00284 | 0.00213 | 0.00171 |
| R8 | 4.830 | 4.830 | 4.830 | 4.830 | 4.830 | 4.830 | 4.830 |
| R9 | 9.659 | 9.659 | 9.659 | 9.659 | 9.659 | 9.659 | 9.659 |
| C6 | 0.00936 | 0.00339 | 0.00207 | 0.00160 | 0.00133 | 0.000995 | 0.000796 |

**Note:** The unit for resistance is kΩ
The unit for capacitance is μF
The above values are not industry-standard values

In summary, the 10-bit linear D/A converter provides analog output for the TMS320C2x. The D/A converter is interfaced to the processor through the I/O space and is driven by latches that store the digital data for the current sample until the next sample period. A smoothing filter provides final analog signal reconstruction by eliminating extraneous high-frequency components in the output waveform.

### *Summary*

The interface of memories and peripherals to the TMS320C2x has been described in this application report. Both direct interfaces and interfaces that utilize address decoding have been considered, with special attention given to READY timing requirements. The design techniques used in these interfaces can be extended to encompass interface of other devices to the TMS320C2x.

## References


1) *Second-Generation TMS320 User's Guide* (literature number SPRU014A), Texas Instruments (1989).
2) *Digital Signal Processing Applications with the TMS320 Family, Volume 1* (literature number SPRA012A), Texas Instruments (1986).
3) *TMS32010 Analog Interface Board User's Guide* (literature number SPRU006), Texas Instruments (1983).
4) The *TTL Data Book Volume 2* (literature number SDLD001), Texas Instruments (1985).
5) The *TTL Data Book Volume 3* (literature number SDAD001A), Texas Instruments (1984).
6) *MOS Data Book*, Micron Technology, Inc. (1990).
7) *CMOS/BiCMOS Data Book*, Cypress Semiconductor (1989).

# Interfacing the TMS320 Family to the TLC32040 Family

Linear Products — Semiconductor Group
Texas Instruments

# 1 Introduction

The TLC32040 and TLC32041 analog interface circuits are designed to provide a high level of system integration and performance. The analog interface circuits combine high resolution A/D and D/A converters, programmable filters, digital control and timing circuits as well as programmable input amplifiers and multiplexers. Emphasis is placed on making the interface to digital signal processors (the TMS320 family) and most microprocessors as simple as possible. This user's guide describes the software and circuits necessary to interface to numerous members of the TMS320 family. It presents three circuits for interfacing the TLC32040 Analog Interface Circuit to the TMS320 family of digital signal processors. Details of the hardware and software necessary for these interfaces are provided.

To facilitate the discussion of the software, the following definitions and naming conventions are used:

1. >nnnn – a number represented in hexadecimal.
2. Interrupt service routine – a subroutine called in direct response to a processor interrupt.
3. Interrupt subroutine – any routine called by the interrupt service routine.
4. Application program (application routine) – the user's application dependent software (e.g. digital filtering routines, signal generation routines, etc.)

# 2 TLC32040 Interface to the TMS32010/E15

## 2.1 Hardware

Because the TLC32040 (Analog Interface Circuit) is a serial-I/O device, the interface to the TMS32010, which has no serial port, requires a small amount of glue-logic. The circuit shown in Figure 2-1 accomplishes the serial-to-parallel conversion for the AIC operating in synchronous mode.

### 2.1.1 Parts List

The interface circuit for the TMS32010 uses the following standard logic circuits:

1. One SN74LS138 3-to-8-line address decoder
2. One SN74LS02 Quad NOR-Gate
3. One SN74LS00 Quad NAND-Gate
4. One SN74LS04 Hex Inverter
5. One SN74LS74 Dual D-Flip-Flop
6. Two SN74LS299 8-bit Shift Registers

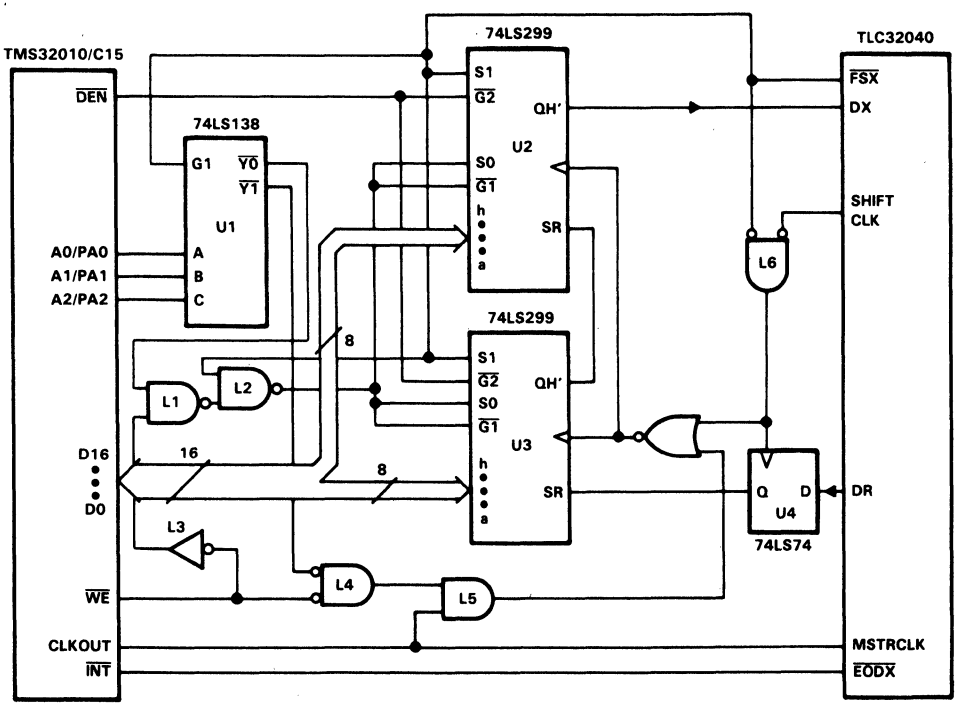



**Figure 2-1. AIC Interface to TMS32010/E15**

### 2.1.2 Hardware Description

The 74LS138 is used to decode the addresses of the ports to which the TLC32040 and the interface logic have been mapped. If no other ports are needed in the development system, this device may be eliminated and the address lines of the TMS32010 used directly in place of $\overline{Y1}$ and $\overline{Y0}$ (see Figure 2-1).

Since the interface circuits are only addressed when the TMS32010 executes an IN or an OUT instruction, gates L1, L2, L3, L4, and L5 are required to enable reading and writing to the shift registers only on these instructions. The TBLW instruction is prohibited because it has the same timing as the OUT instruction. Flip-flop U4 ensures that the setup and hold times of 74LS299 shift registers are met.

Although not shown in the circuit diagram, it is recommended that the $\overline{CLR}$ pins of the 74LS299 shift registers as well as the $\overline{RESET}$ pin of the AIC be tied to the power-up reset circuit shown in the AIC data sheet. This ensures that the registers are clear when the AIC begins to transfer data and decrease the possibility that the AIC will shift in bad data which could cause the AIC to shut down or behave in an unexpected manner.

## 2.2 Software

The flowcharts for the communication program along with the TMS32010 program listing are presented in Appendix A. If this software is to be used, an application program that moves data into and out of the transmit and receive registers must be supplied.

### 2.2.1 Initializing the TMS32010/E15

As shown in the flowcharts in Appendix A, the program begins with an initialization routine which clears both the transmit/receive-end flag and the secondary communication flag, and stores the addresses of the interrupt subroutines. The program uses the MPYK..PAC instruction sequence to load data memory locations with the 12-bit address of the subroutines. This sequence is only necessary if the subroutines are to reside in program memory locations larger than >00FF. Otherwise, the instructions LACK and SACL may be used to initialize the subroutine-address storage locations.

### 2.2.2 Communicating with the TLC32040

After the storage registers and status register have been initialized, the interrupt is enabled and control is passed to the user's application routine (i.e. the system-dependent software that processes received data and prepares data for transmission). The program ignores the first interrupt that occurs after interrupts are enabled (page A-6, line 206, IGINT routine), allowing the AIC to stabilize after a reset. The application routine should not write to the shift registers while data is moving into (and out of) them. In addition, it should ensure that no primary data is written to the shift registers between a primary and secondary data-communication pair. The first objective can be accomplished by writing to the 74LS299 shift registers as quickly as possible after the receive interrupt. The number of instruction cycles between the data transfers can be calculated from the conversion frequency. By counting instruction cycles in the application program, it is possible to determine whether the data transfer will conflict with the OUT instruction to the shift register. The second objective can be accomplished by monitoring SNDFLG in the application program. If SNDFLG is true (>00FF), secondary communication has not been completed.

When the processor receives an interrupt, the program counter is pushed onto the hardware stack and then the program counter is set to >0002, the location of the interrupt service routine, INTSVC (page A-3, line 46). The interrupt service routine then saves the contents of the accumulator and the status register and calls the interrupt subroutine to which XVECT points. If secondary communication is to follow the upcoming primary communication, XVECT, is set by the application program to refer to SINT1, otherwise, XVECT defaults to NINT (i.e the normal interrupt routine).

Because the interrupt subroutine makes one subroutine call and uses two levels of the hardware stack, the application program can only use two levels of nesting (i.e., if stack extension is not used). This means that any subroutine called by the application program can only call subroutines containing no instructions that use the hardware stack (e.g. TBLW) and that make no other subroutine calls. In addition, if the application program and communication program are being implemented on an XDS series emulator, the emulator consumes one level of the hardware stack and allows the application program only one level of nesting (i.e., one level of subroutine calls).

As shown in the flowcharts in Appendix A, the normal interrupt routine reads the A/D data from the shift registers and then sets the receive/transmit end-flag (RXEFLG). The application program must write the outgoing D/A data word to the shift registers at a time convenient to the application routine. It should have the restriction that the data be written before the next data transfer.

### 2.2.3 TLC32040 Secondary Communication

If it is necessary to write to the control register of the AIC or configure any of the AIC internal counters, the application program must initiate a primary/secondary communication pair. This can be accomplished by placing a data word in which bits 0 and 1 are both high into DXMT, placing the secondary control word (see program listing page A-3) in D2ND, and placing the address of the secondary communication subroutine, SINT1, in XVECT. When the next interrupt occurs, the interrupt subroutine will call routine SINT1. SINT1 reads the A/D information from the shift registers and writes the secondary communication word to the shift registers.

# 3 TLC32040 Interface to the TMS32020

## 3.1 Hardware Description

Because the TLC32040 is designed specifically to interface with the serial port of the TMS32020/C25, the interface requires no external hardware. Except for CLKR and CLKX, there is a one-to-one correspondence between the serial port control and data pins of TMS32020 and TLC32040. CLKR and CLKX are tied together since both the transmit and the receive operations are synchronized with SHIFT CLK of the TLC32040. The interface circuit, along with the communication program (page B-5), allow the AIC to communicate with the TMS32020/C25 in both synchronous and asynchronous modes. See Figures 3-1, 3-2, and 3-4.

## 3.2 Software

The program listed in Appendix B allows the AIC to communicate with the TMS32020 in synchronous or asynchronous mode. Although originally written for the TMS32020, it will work just as well for the TMS320C25.

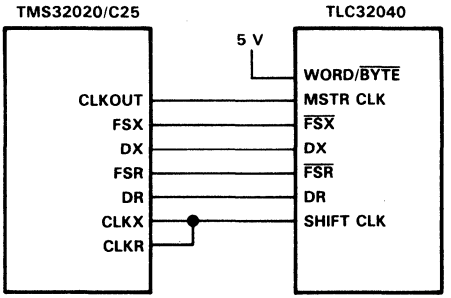
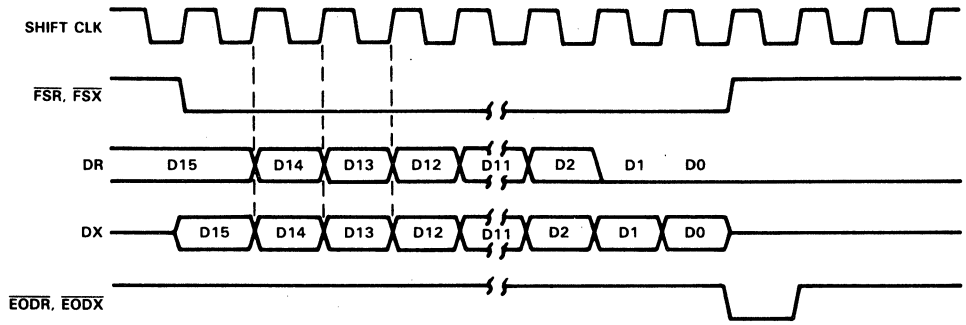



**Figure 3-1. AIC Interface to TMS32020/C25**




The sequence of operation is:
1. The FSX or FSR pin is brought low.
2. One 16-bit word is transmitted or one 16-bit byte is received.
3. The FSX or FSR pin is brought high.
4. The EODX or EODR pin emits a low-going pulse as shown.

**Figure 3-2. Operating Sequence for AIC-TMC32020/C25 Interface**

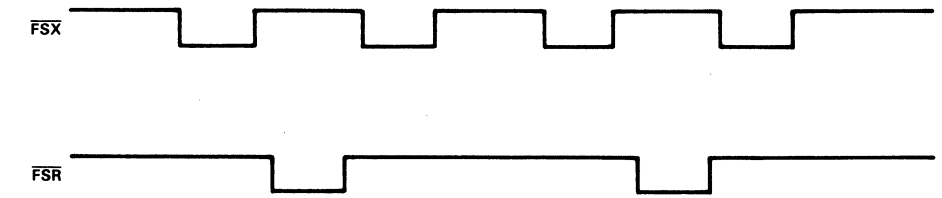



**Figure 3-3. Asynchronous Communication AIC-TMS32020/C25 Interface**

### 3.2.1 Initializing the TMS32020/C25

This program starts by calling the initialization routine. The working storage registers for the communication program and the transmit and receive registers of the DSP are cleared, and the status registers and interrupt mask register of the TMS32020/C25 are set (see program flow charts in Appendix B). The addresses of the transmit and receive interrupt subroutines are placed in their storage locations, and the addresses of the routines which ignore the first transmit and receive interrupts are placed in the transmit and receive subroutine pointers (XVECT and RVECT). The TMS32020/C25 serial port is configured to allow transmission of 16-bit data words (FO, the serial port format bit of the TMS32020/C25 must be set to zero) with an externally generated frame synchronization (FSX and FXR are inputs, TXM bit is set to 0).

### 3.2.2 Communicating with the TLC32040

After the TMS32020/C25 has been initialized, interrupts are enabled and the program calls subroutine IGR. The processor is instructed to wait for the first transmit and receive interrupts (XINT and RINT) and ignore them. After the TMS32020 has received both a receive and a transmit interrupt, the IGR routine will transfer control back to the main program and IGR will not be called again.

If the transmit interrupt is enabled, the processor branches to location 28 in program memory at the end of a serial transmission. This is the location of the transmit interrupt service routine. The program context is saved by storing the status registers and the contents of the accumulator. Then the interrupt service routine calls the interrupt subroutine whose address is stored in the transmit interrupt pointer (XVECT).

A similar procedure occurs on completion of a serial receive. If the receive interrupt is enabled, the processor branches to location 26 in program memory. As with the transmit interrupt service routine (XINT, page B-8, line 223), the receive interrupt service routine (page B-8,line 191) saves context and then calls the interrupt subroutine whose address is stored in the receive interrupt pointer (RVECT). It is important that during the execution of either the receive or transmit interrupt service routines, all interrupts are disabled and must be re-enabled when the interrupt service routine ends.

The main program is the application program. Procedures such as digital filtering, tone-generation and detection,and secondary communication judgment can be placed in the application program. In the program listing shown in Appendix B, a subroutine (C2ND) is provided which will prepare for secondary communication. If secondary communication is required, the user must first write the data with the secondary code to the DXMT register. This data word should have the two least significant bits set high (e.g. >0003). The first 14 bits transmitted will go to the D/A converter and the last two bits indicate to the AIC that secondary communication will follow. After writing to the DXMT register, the secondary communication word should be written to the D2ND register.

This data may be used to program the AIC internal counters or to reconfigure the AIC (e.g. to change from synchronous to asynchronous mode or to bypass the bandpass filter). After both data words are stored in their respective registers, the application program can then call the subroutine C2ND which will prepare the TMS32020 to transmit the secondary communication word immediately after primary communication.

### 3.2.3 Secondary Communications - Special Considerations

This communication program disables the receive interrupt (RINT) when secondary communication is requested. Because of the critical timing between the primary and secondary communication words and because RINT carries a higher priority than the transmit interrupt, the receive interrupt cannot be allowed to interrupt the processor before the secondary data word can be written to the data-transmit register. If this situation were to occur, the AIC would not receive the correct secondary control word and the AIC could be shut down.

In many applications, the AIC internal registers need only be set at the beginning of operation, (i.e, just after initialization). Thereafter, the DSP only communicates with the AIC using primary communication. In cases such as these, the communication program can be greatly simplified.

# 4 Interfacing the TLC32040 to the TMS320C17

## 4.1 Hardware Description

As shown in Figure 4-1, the TMS320C17 interfaces directly with the TLC32040. However, because the TMS320C17 responds more slowly to interrupts than the TMS32010/E15 or the TMS32020/C25, additional circuit connections are necessary to ensure that the TMS320C17 can respond to the interrupt, accomplish the context-switching that is required when an interrupt is serviced, and proceed with the interrupt vector. This must all be accomplished within the strict timing requirements imposed by the TLC32040. To meet these requirements, $\overline{FSX}$ of the TLC32040 is connected to the $\overline{EXINT}$ pin of the TMS320C17. This allows the TMS320C17 to recognize the transmit interrupt before the transmission is complete. This allows the interrupt service routine to complete its context-switching while the data is being transferred. The interrupt service routine branches to the interrupt subroutines only after the FSX flag bit has been set. This signals the end of data transmission.

The other hardware modification involves connecting the $\overline{EODX}$ pin of the TLC32040 to the $\overline{BIO}$ pin of the TMS320C17. Because the TMS320C17 serial port accepts data in 8-bit bytes (see Figure 4-2) and the TLC32040 controls the byte sequence (i.e. which byte is transmitted first, the high-order byte or the low-order byte) it is important that the TMS320C17 be able to distinguish between the two transmitted bytes. The $\overline{EODX}$ signal is asserted only once during each transmission pair, making it useful for marking the end of a transmission pair and synchronizing the TMS320C17 with the AIC byte sequence. After synchronism has been established, the $\overline{BIO}$ line is no longer needed by the interface program and may be used elsewhere.

Because the TMS320C17 serial port operates only in byte mode, 16-bit transmit data should be separated into two 8-bit bytes and stored in separate registers before a transmit interrupt is acknowledged. Alternatively, the data can be prepared inside the interrupt service routine before the interrupt subroutine is called. From the time that the interrupt is recognized to the end of the data transmission is equivalent to 28 TMS320C17 instruction cycles.

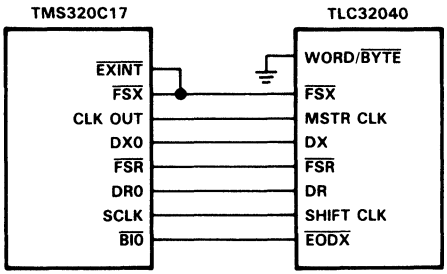
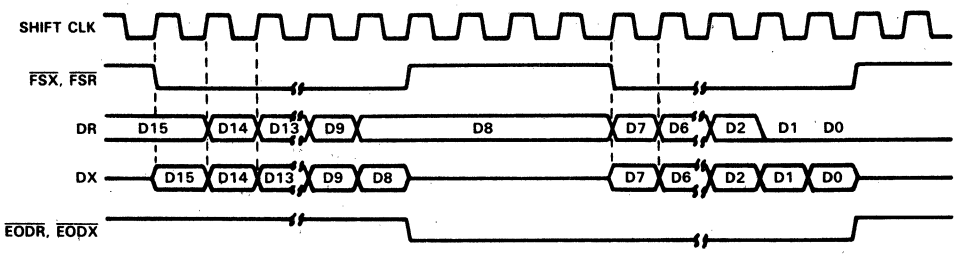



**Figure 4-1. AIC Interface to TMS320C17**




The sequence of operation is:

1. The $\overline{FSX}$ or $\overline{FSR}$ pin is brought low.
2. One 8-bit word is transmitted or one 8-bit byte is received.
3. The $\overline{EODX}$ or $\overline{EODR}$ pins are brought low.
4. The $\overline{FSX}$ or $\overline{FSR}$ emit a positive frame-sync pulse that is four shift clock cycles wide.
5. One 8-bit byte is transmitted and one 8-bit byte is received.
6. The $\overline{EODX}$ and $\overline{EODR}$ pins are brought high.
7. The $\overline{FSX}$ and $\overline{FSR}$ pins are brought high .

**Figure 4-2. Operating Sequence for AIC-TMS320C17**

## 4.2 Software

The software listed in Appendix C only allows the AIC to communicate with the TMS320C17 in synchronous mode. This communication program is supplied with an application routine, DLB (Appendix C, program listing line 236), which returns the most recently received data word back to the AIC (digital loopback).

### 4.2.1 Initializing the TMS320C17

The program begins with an initialization routine (INIT, page C-5, line 122). Interrupts are disabled and all the working storage registers used by the communication program are cleared. Both transmit registers are cleared, the constants used by the program are initialized and the addresses of the subroutines called by the program are placed in data memory. This enables the interrupt service routine to call subroutines located in program-memory addresses higher than 255. After the initialization is complete, the program monitors the $\overline{BIO}$ line of the TMS320C17 and waits for the end of the first interrupt pair (the AIC is in byte mode). Afterwards, interrupts are enabled and control is passed to the main program.

### 4.2.2 AIC Communications and Interrupt Management

Because the AIC $\overline{FSX}$ pin is tied to the $\overline{EXINT}$ line of the TMS320C17 and the delay through the interrupt multiplexer, the interrupt service routine is called four instruction cycles after the falling edge of $\overline{FSX}$. The interrupt service routine (INTSVC, Appendix C, program listing, line 91) completes its context switching and then monitors the lower control register, polling the $\overline{FSX}$ flag bit that indicates the end of the 8-bit serial data transfer. If the $\overline{FSX}$ flag bit is set, the transfer is complete. After this bit is set, control is transferred to the interrupt subroutine whose address is stored in VECT. The serial communication must be complete before data is read from the data receive register.

When no secondary communication is to follow, the interrupt subroutines, NINT1 and NINT2, are called. If data has been stored in DXMT2 (the low-order eight bits of the transmit data word), which does not indicate that secondary communication is to follow, the interrupt service routine calls NINT1 when the first 8-bit serial transfer is complete. NINT1 immediately writes the second byte of transmit data, (i.e., the contents of DXMT2) to transmit data register 0 (TR0). It then moves the first byte of the received data (i.e. the high-order byte of the A/D conversion result) into DRCV1. NINT1 then stores in VECT the address of NINT2. NINT2 is called at the end of the next 8-bit data transfer and resets the $\overline{FSX}$ interrupt flag bit by writing a logic high to it. The next interrupt (a falling edge on $\overline{EXINT}$) occurs before the interrupt service routine returns control to the main program. This is an acceptable situation since the TMS320C17, on leaving the interrupt service routine, recognizes that an interrupt has occurred and immediately responds by servicing the interrupt.

The interrupt subroutine NINT2 is similar in operation to NINT1. It stores the low-order byte of receive data (bits 7 through 0 of the A/D conversion result) and stores the address of the next interrupt subroutine in VECT. NINT2 does not write to the transmit data register, TR0. This task has been left to the application program. After the transmit data has been prepared by the main program and the data has been stored in DXMT1 and DXMT2, the main program stores the first byte of the transmit data in transmit data register 0 (TR0).

### 4.2.3 Secondary Communications

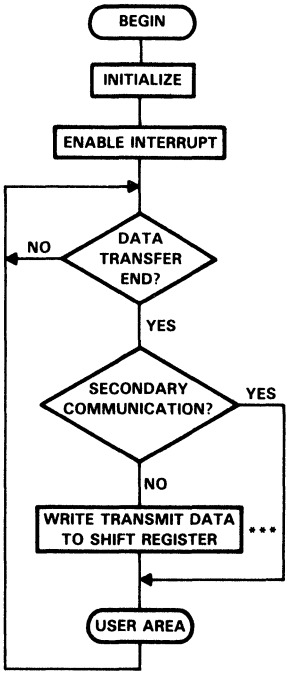
The interrupt subroutines SINT1 through SINT4 are called when secondary communication is required. For secondary communication, DXMT1 and DXMT2 will hold the primary communication word. DXMT3 and DXMT4 will hold the secondary communication word. VECT, the subroutine pointer should then be initialized to the address of SINT1. As with the normal (primary communication only) interrupt subroutines (i.e., NINT1 and NINT2), the secondary communication routines will change VECT to point to the succeeding routine (e.g., SINT1 will point to SINT2, SINT2 will point to SINT3, etc.).

# 5 Summary

The TLC32040 is an excellent choice for many digital signal processing applications such as speech recognition/storage systems and industrial process control. The different serial modes of the AIC (synchronous, asynchronous, 8- and 16-bit) allow it to interface easily with all of the serial port members of the TMS320 family as well as other processors.

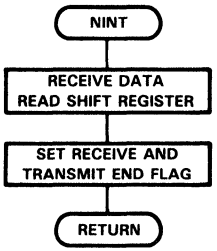
# A TLC32040 and TMS32010 Flowcharts and Communication Program

## A.1 Flowcharts

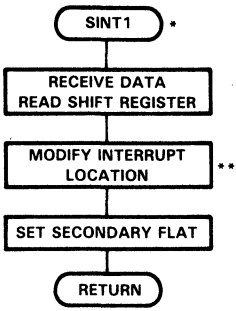


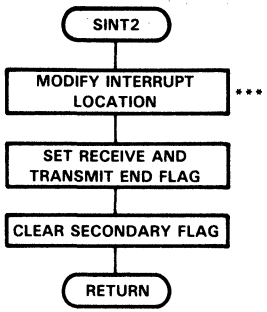

***Modified to call NINT.

a. MAIN




b. PRIMARY INTERRUPT ROUTINE




*Set, if need secondary.
**Modify to call SINT2.
***Modify to call NINT.
****Must execute before transfer beginning.

c. SECONDARY DATA COMMUNICATIONS 1

d. SECONDARY DATA COMMUNICATION 2

## A.2 Communication Program List

```
0001            ***************************************************************
0002            *   When using this program, the circuit in the TLC32040    *
0003            *  data sheet or its  equivalent circuit must be used.      *
0004            *  TMS32010 port 0 and port 1 are reserved for data          *
0005            *  receiving and data  transmitting. TBLW command is        *
0006            *  prohibited because it has the same timing as the OUT     *
0007            *  command. TLC32040 is used only in synchronous mode.      *
0008            ***************************************************************
0009            *
0010      0002  RXEFLG  EQU   >02       receive & xmit end flag.
0011      0003  SNDFLG  EQU   >03       secondary communication flag.
0012      0004  DRCV    EQU   >04       receive data storage.
0013      0005  DXMT    EQU   >05       xmit    data storage.
0014      0006  D2ND    EQU   >06       secondary data storage.
0015      0007  XVECT   EQU   >07       interrupt address storage.
0016      0008  ACHSTK  EQU   >08       ACCH stack.
0017      0009  ACLSTK  EQU   >09       ACCL stack.
0018      000A  SSTSTK  EQU   >0A       Status stack.
0019      000C  ANINT   EQU   >0C       interrupt address 1
0020      000D  ASINT1  EQU   >0D       interrupt address 2
0021      000E  ASINT2  EQU   >0E       interrupt address 3
0022      000F  TMP0    EQU   >0F       temporary register.
0023            *
0024      00FF  SET     EQU   >FF
0025      0001  ONE     EQU   >01
0026            *    =================
0027            *      Reset vector.
0028            *    =================
0029 0000             AORG  >0000     program start address.
0030 0000 F900        B     EPIL      jump to Initialization.
     0001 000D
```

```
0031            ***********************************************************
0032            *     ==================                                  *
0033            *      Interrupt vector.                                  *
0034            *     ==================                                  *
0035            *       For secondary communication,modify the contents   *
0036            *     of XVECT to the address of secondary communication  *
0037            *     and store secondary data in D2ND.                   *
0038            *   ex.                                                   *
0039            *       LAC   ASINT1,0 modify XVECT.                      *
0040            *       SACL  XVECT,0                                     *
0041            *           |                                             *
0042            *       LAC   D2ND,0   store secondary data.              *
0043            ***********************************************************
0044 0002              AORG  >0002    interrupt vector.
0045 0002
0046 0002 7C0A  INTSVC SST   SSTSTK   push status register.
0047 0003 6E01         LDPK  ONE      set data pointer one.
0048 0004 5808         SACH  ACHSTK   push ACCH.
0049 0005 5009         SACL  ACLSTK   push ACCL.
0050 0006 2007         LAC   XVECT,0  load interrupt address.
0051 0007 7F8C         CALA           branch to interrupt routine.
0052 0008 6508         ZALH  ACHSTK   pop ACCH
0053 0009 7A09         OR    ACLSTK   pop ACCL.
0054 000A 7B0A         LST   SSTSTK   pop stack register.
0055 000B 7F82         EINT           enable interrupt.
0056 000C 7F8D         RET            return from interrupt routine.
0057            ***********************************************************
0058            *     ============================                        *
0059            *      Initialization after reset.                        *
0060            *     ============================                        *
0061            *                                                         *
0062            *   Data RAM locations 82H(130) through 8FH(143),         *
0063            *   12 words of Page 1,are reserved for this program.     *
0064            *   The user must set the status register by adding       *
0065            *   the SST command at the end of the initial routine     *
0066            ***********************************************************
0067            *
0068            *
0069            *
0070 000D              AORG  $        initial program.
0071 000D
0072 000D 6E01  EPIL   LDPK  ONE      set Data page pointer one.
0073 000E
0074 000E 7E01         LACK  ONE      save normal communication address
0075 000F 500F         SACL  TMP0     to its storage.
0076 0010 6A0F         LT    TMP0
0077 0011 802C         MPYK  NINT
0078 0012 7F8E         PAC
0079 0013 500C         SACL  ANINT
0080 0014
0081 0014 8030         MPYK  SINT1    save secondary communication address1
0082 0015 7F8E         PAC            to its storage.
0083 0016 500D         SACL  ASINT1
```

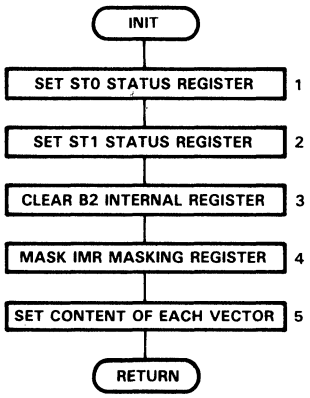
```
0084 0017
0085 0017 8037          MPYK  SINT2     save secondary communication address2
0086 0018 7F8E          PAC             to its storage.
0087 0019 500E          SACL  ASINT2
0088 001A
0089 001A 803E          MPYK  IGINT     ignore interrupt once after master
0090 001B 7F8E          PAC             reset.
0091 001C 5007          SACL  XVECT
0092 001D
0093 001D 7F89          ZAC             clear flags.
0094 001E 5002          SACL  RXEFLG,0
0095 001F
0096 001F 5003          SACL  SNDFLG,0
0097 0020
0098 0020
0099 0020
0100 0020 7F82          EINT            enable interrupt.
0101                *
0102                ************************************************************
0103                *           ====================                           *
0104                *             Main program.                                *
0105                *            User can modify.                              *
0106                *           ====================                           *
0107                *                                                          *
0108                *  This program allows the user 2 levels of nesting, since *
0109                * two levels are used as stack for the interrupt.          *
0110                * When the RXEFLG flag is false, no data transfer has      *
0111                * occurred; if true then data transfer has finished.       *
0112                * User routines such as digital filter, secondary-data-    *
0113                * communication judgement etc. must be placed in this      *
0114                * location. Depending on the sampling rate (conversion     *
0115                * rate), these user routines must write the xmit data to   *
0116                * the shift registers within approximately 500 instruction *
0117                * cycles. If the user requires secondary communication, it *
0118                * will be necessary to delay the OUT instruction until the *
0119                * secondary data transfer has finished.                    *
0120                ************************************************************
0121 0021
0122 0021 2002  MAIN    LAC   RXEFLG,0 wait for interrupt.
0123 0022 FF00          BZ    MAIN
     0023 0021
0124 0024
0125 0024 2003          LAC   SNDFLG,0 skip OUT instruction during secondary
0126 0025 FE00          BNZ   MAIN1    communication.
     0026 0028
0127 0027
0128 0027 4905          OUT   DXMT,PA1 write xmit data to shift register.
0129 0028
0130 0028 7F89  MAIN1   ZAC             clear flags.
0131 0029 5002          SACL  RXEFLG
0132 002A
0133 002A F900          B     MAIN      loop.
     002B 0021
```

```
0134                *
0135                ************************************************************
0136                *    ============================                          *
0137                *      normal interrupt routine.                           *
0138                *    ============================                          *
0139                *                          destroy ACC,DP.                 *
0140                *                                                          *
0141                *   Write the contents of DXMT to the LS299's, receive     *
0142                *   DAC data in DRCV, and set RXEFLG flag.                 *
0143                ************************************************************
0144 002C
0145 002C 4004     NINT     IN     DRCV,PA0 Receive data from shift register.
0146 002D
0147 002D 7EFF              LACK   SET      set receive and xmit ended flag.
0148 002E 5002              SACL   RXEFLG
0149 002F
0150 002F 7F8D              RET            return.
0151                *
0152                ************************************************************
0153                *     ================================================     *
0154                *       secondary communication interrupt routine 1.       *
0155                *     ================================================     *
0156                *                             destroy ACC,DP               *
0157                *                                                          *
0158                *   Write the contents of D2ND to the 'LS299s, receive     *
0159                *  data in DRCV, and modify XVECT for secondary communi    *
0160                *  -cation  interrupt.                                     *
0161                ************************************************************
0162 0030
0163 0030 4004     SINT1    IN     DRCV,PA0 receive data from shift register.
0164 0031
0165 0031 4906              OUT    D2ND,PA1 write secondary data to shift
0166                *                                                 register.
0167 0032 200E              LAC    ASINT2,0 modify interrupt location.
0168 0033 5007              SACL   XVECT    secondary communication 2
0169 0034
0170 0034 7EFF              LACK   SET      set secondary communication flag.
0171 0035 5003              SACL   SNDFLG,0
0172 0036
0173 0036 7F8D              RET            return.
0174 0037
0175                ************************************************************
0176                *     ================================================     *
0177                *       secondary communication interrupt routine 2.       *
0178                *     ================================================     *
0179                *                             destroy ACC,DP               *
0180                *                                                          *
0181                *   Modify XVECT for normal communication, and set RXEFLG  *
0182                *  flag.                                                   *
0183                ************************************************************
0184 0037
0185 0037 200C     SINT2    LAC    ANINT    modify interrupt location
0186 0038 5007              SACL   XVECT    normal communication.
```

```
0187 0039
0188 0039 7EFF          LACK  SET       set receive and xmit ended flag.
0189 003A 5002          SACL  RXEFLG
0190 003B
0191 003B 7F89          ZAC             Clear secondary communication flag.
0192 003C 5003          SACL  SNDFLG,0
0193 003D
0194 003D 7F8D          RET             return.
0195 003E
0196              ************************************************************
0197              *     ====================================                 *
0198              *      ignoring first interrupt after reset.               *
0199              *     ====================================                 *
0200              *                              destroy ACC,DP.             *
0201              *   Ignore first interrupt after reset. TLC32040 receives  *
0202              * zero as DAC data but no ADC data in DRCV.                *
0203              *                                                          *
0204              ************************************************************
0205 003E
0206 003E 200C  IGINT   LAC   ANINT     modify interrupt location
0207 003F 5007          SACL  XVECT     normal communication.
0208 0040
0209 0040 7F8D          RET             return.
0210 0041
0211                    END
NO ERRORS, NO WARNINGS
```

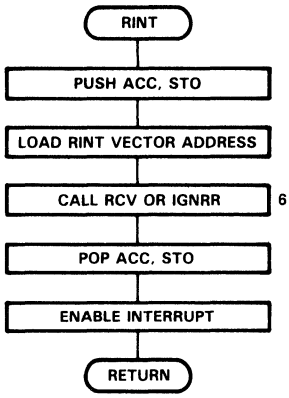
# B TLC32040 and TMS32020 Flowcharts and Communication Program

## B.1 Flowcharts

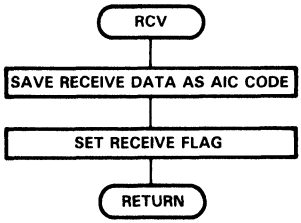



1 – Alterable AR pointer and OVM.
2 – Alterable CNF, SXM and XF.
3 – Must clear at least 108 through 127, 19 of internal RAM.
4 – If IMR is changed by user program, INST must be changed.
5 – Their contents will be changed by their routine locations.
6 – IGNRR is executed only once after reset.

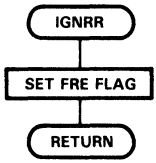
a. INITIALIZATION



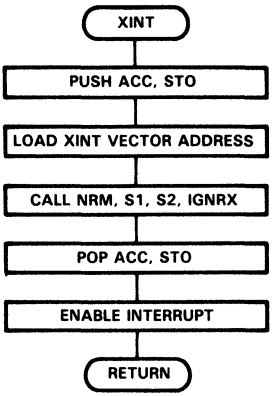

b. RECEIVE INTERRUPT SERVICE ROUTINE




c. RECEIVE SUBROUTINE

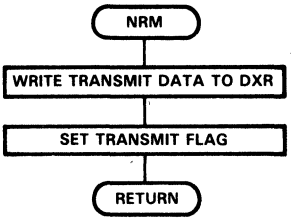



d. IGNORE INTERRUPT

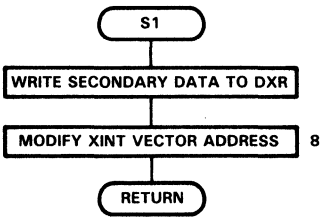



7 – IGNRX is executed only once after reset.

**e. TRANSMIT INTERRUPT SERVICE ROUTINE**

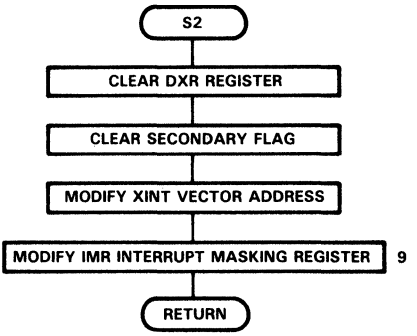



**f. PRIMARY TRANSMISSION ROUTINE**

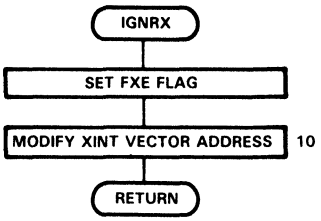



8 – Modify to S2 address.
9 – Modify to NRM address.

**g. PRIMARY-SECONDARY COMMUNICATIONS 1**

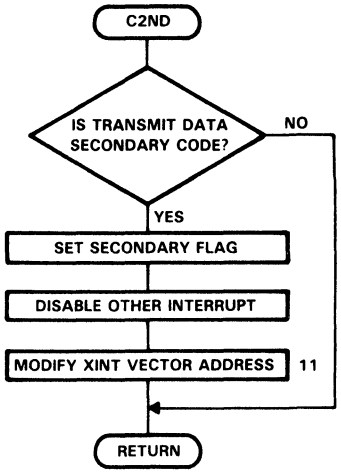



**h. PRIMARY-SECONDARY COMMUNICATIONS 2**

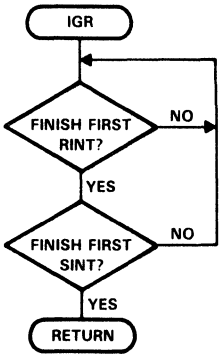



10 - Modify to NRM address.
11 - Modify to S1 address.

i. IGNORE TRANSMIT INTERRUPT




j. SECONDARY COMMUNICATION JUDGMENT




k. IGNORE FIRST INTERRUPTS

## B.2 Communication Program List

```
0001            ***************************************************************
0002            *   ========================================                  *
0003            *    TLC32040 & TMS32020 communication program.               *
0004            *   ========================================                  *
0005            *               by H.Okubo & W.Rowand                         *
0006            *                  version 1.0      7/15/87.                  *
0007            *                                                             *
0008            *  This is a TMS32020 - TLC32040 communication program        *
0009            * that can can be used in many systems. To use this           *
0010            * program, the TMS32020 and the TLC32040 (AIC) must be        *
0011            * connected as shown in Volume 3 of Linear and Interface      *
0012            * Applications. The program reserves TMS32020 internal        *
0013            * data memory 108 through 127 (B2) as flags and data          *
0014            * storage. When secondary communication is needed, every      *
0015            * maskable interrupt except XINT interrupt is disabled        *
0016            * until that communication finishes. This means that XINT     *
0017            * will be valid only during one DAC conversion time.          *
0018            * If you have any questions, please let us know.              *
0019            ***************************************************************
0020            *
0021            *
0022            * ==========================
0023            *   Memory mapped register.
0024            * ==========================
0025            *
0026      0000  DRR     EQU  0        * data receive register address.
0027      0001  DXR     EQU  1        * data xmit    register address.
0028      0004  IMR     EQU  4        * interrupt mask register address.
0029            *
0030            *  ========================================
0031            *    Reserved onchip RAM as flags and storages.
0032            *      (block B2 108 through 127.)
0033            *  ========================================
0034            *
0035      006C  FXE     EQU  108      * ignore first XINT flag.
0036      006D  FRE     EQU  109      * ignore first RINT flag.
0037      006F  TMP0    EQU  111      * temporary register.
0038      0070  ACCHST  EQU  112      * stack for ACCH.
0039      0071  ACCLST  EQU  113      * stack for ACCL.
0040      0072  SSTST   EQU  114      * stack for ST0 register.
0041      0073  INTST   EQU  115      * stack for IMR register.
0042      0074  RVECT   EQU  116      * vector for RINT.
0043      0075  XVECT   EQU  117      * vector for XINT.
0044      0076  VRCV    EQU  118      * RINT vector storage.
0045      0077  VNRM    EQU  119      * XINT vector storage.
0046      0078  VS1     EQU  120      * secondary vector storagel.
0047      0079  VS2     EQU  121      * secondary vector storage2.
0048      007A  DRCV    EQU  122      * receive data storage.
0049      007B  DXMT    EQU  123      * xmit data storage.
0050      007C  D2ND    EQU  124      * secondary data storage.
0051      007D  FRCV    EQU  125      * receive flag.
0052      007E  FXMT    EQU  126      * xmit flag.
0053      007F  F2ND    EQU  127      * secondary communication flag.
0054            *
```

```
0055                ************************************************************
0056                *   Processor starts at this address after reset.          *
0057                *                                                          *
0058 0000                   AORG 0       *  program start address.            *
0059 0000 FF80              B  STRT      *  jump to Initialization routine.   *
     0001 0020
0060                ************************************************************
0061                *
0062                ************************************************************
0063                *   Receive interrupt location.                            *
0064                *                                                          *
0065 001A                   AORG 26      * Rint vector.                       *
0066 001A FF80              B  RINT      * jump to receive interrupt routine. *
     001B 004A
0067                ************************************************************
0068                *
0069                ************************************************************
0070                *   Transmit interrupt location.                           *
0071                *                                                          *
0072 001C                   AORG 28      * Xint vector.                       *
0073 001C FF80              B  XINT      * jump to xmit interrupt routine.    *
     001D 005A
0074                ************************************************************
0075                *
0076 0020                   AORG 32      * start initial program.
0077                *
0078                ************************************************************
0079                *  User must initialize DSP with the routine INIT.  The    *
0080                * user may modify this routine to suit his system require- *
0081                * ments as he likes.                                       *
0082                ************************************************************
0083 0020
0084 0020 FE80      STRT    CALL INIT    *
     0021 0025
0085 0022 CE00              EINT         * enable interrupt.
0086 0023 FE80              CALL IGR
     0024 008B
```

```
0087            *
0088            ***************************************************************
0089            *                ==============                               *
0090            *                  User area                                  *
0091            *                ==============                               *
0092            *                                                             *
0093            *  This program allows the user 2 levels of nesting, since    *
0094            * 2 levels are used as stack for the interrupt. When the      *
0095            * FXMT flag is false no data transmit has occurred. When      *
0096            * the FRCV flag is false no data has been received. As        *
0097            * those flags are not reset by any routine in this program    *
0098            * the user must reset the flag to read or write new data      *
0099            * and note that >00FF means true, >0000 means false.          *
0100            * User routines such as digital filtering, secondary-data-    *
0101            * communication judgement etc. must be placed in this         *
0102            * location. Depending on the sampling rate (conversion        *
0103            * rate), these user routines must write the xmit data to      *
0104            * DXMT registers within approximately 500 instruction         *
0105            * cycles. If the user requires secondary communication,       *
0106            * first write data with secondary code to DXMT, then write    *
0107            * secondary data to D2ND and call C2ND routine to set F2ND    *
0108            * and modify XVECT for secondary communication. Note that     *
0109            * every maskable interrupt except XINT is disabled during     *
0110            * this conversion cycle including secondary communication.    *
0111            ***************************************************************
0112            *
0113            ***************************************************************
0114            *      ======================                                 *
0115            *       Initializing routine.                                 *
0116            *      ======================                                 *
0117            *  This routine initializes the status registers, flags,      *
0118            * vector storage contents and internal data locations         *
0119            * 96 through 107. Note that the User can modify these         *
0120            * registers (i.e. ST0 ST1 IMR), as long as the contents do    *
0121            * not conflict with the operation the AIC.                    *
0122            ***************************************************************
0123 0025 C800  INIT    LDPK 0          * set status0 register.
0124 0026 D001          LALK >0E00,0    * 0000 1110 0000 0000B
     0027 0E00
0125 0028 606F          SACL TMP0,0     * ARP=0  AR pointer 0
0126 0029 506F          LST  TMP0       * OV =0 (Overflow reg.clear)
0127            *                       * OVM=1 (Overflow mode set to 1)
0128            *                       * ?  =1  Not affected.
0129            *                       * INTM=1 Not affected
0130            *                       * DP 000000000 page 0
0131            *
0132            *                       * set status1 register.
0133            *                       *
0134 002A D001          LALK  >03F0     * 0000 0011 1111 0000B
     002B 03F0
0135 002C 606F          SACL  TMP0,0    * APB=0
0136 002D 516F          LST1  TMP0      * CNF=0 (Set B0 data memory)
0137            *                       * TC =0
```

```
0138                *                       * SXM=1 (enable sign extend mode.)
0139                *                       * D9-D5=111111 not affected.
0140                *                       * XF=1 (XF pin status.)
0141                *                       * F0=0 (16bit data transfer mode.)
0142                *                       * TXM=0 (FSX input)
0143                *
0144                *                       *
0145                *
0146 002E CA00             ZAC               * clear  registers
0147 002F 6001             SACL  DXR,0       *
0148 0030 6000             SACL  DRR,0       *
0149 0031 C060             LARK  AR0,96      * clear Block B2.
0150 0032 CB1F             RPTK  31          *
0151 0033 60A0             SACL  *+,0        *
0152                *
0153                *   Interrupt masking
0154                *
0155 0034 CA30             LACK  >30         * 0000 0000 0011 0000B
0156 0035 6004             SACL  IMR,0       * XINT________|| ||||
0157 0036 6073             SACL  INTST,0     * RINT________| ||||
0158                *                       * TINT__________||||
0159                *                       * INT2_________|||
0160                *                       * INT1________||
0161                *                       * INT0_______|
0162                *
0163 0037 D001             LALK  NRM,0       * normal xint routine address.
     0038 0065
0164 0039 6077             SACL  VNRM,0      *
0165                *
0166 003A D001             LALK  S1,0        * secondary xint routine address 1.
     003B 006A
0167 003C 6078             SACL  VS1,0       *
0168                *
0169 003D D001             LALK  S2,0        * secondary xint routine address 2.
     003E 006F
0170 003F 6079             SACL  VS2,0       *
0171                *
0172 0040 D001             LALK  RCV,0       * rint routine address.
     0041 0055
0173 0042 6076             SACL  VRCV,0
0174                *
0175 0043 D001             LALK  IGNRR,0     * set ignore first rint address.
     0044 0092
0176 0045 6074             SACL  RVECT,0
0177                *
0178 0046 D001             LALK  IGNRX,0     * set ignore first xint address.
     0047 0097
0179 0048 6075             SACL  XVECT,0
0180 0049 CE26             RET               * return.
```

```
0181                *
0182                ****************************************************************
0183                *      ================================                        *
0184                *          Receive interrupt routine.                          *
0185                *      ================================                        *
0186                *  This routine stores receive data in its storage             *
0187                * DRCV (112 page0) and sets receive flag FRCV (125 page0).     *
0188                * As 2 levels nesting are used, this routine allows the        *
0189                * user 2 levels nesting, without stack extension.              *
0190                ****************************************************************
0191 004A 7872     RINT    SST SSTST       * push ST0 register.
0192 004B C800              LDPK 0          * data pointer page 0.
0193 004C 6071              SACL ACCLST,0   * push ACCL.
0194 004D 6870              SACH ACCHST,0   * push ACCH.
0195 004E 2074              LAC  RVECT,0    * load ACC vector address.
0196 004F CE24              CALA
0197 0050 4171              ZALS ACCLST     * pop ACC
0198 0051 4870              ADDH ACCHST
0199 0052 5072              LST SSTST       * pop ST register.
0200 0053 CE00              EINT            * enable interrupts.
0201 0054 CE26              RET             * return.
0202                *
0203 0055 2000     RCV      LAC DRR,0       * load data from DRR.
0204 0056 607A              SACL DRCV,0     * save it to its storage.
0205 0057 CAFF              LACK >FF        * set receive flag.
0206 0058 607D              SACL FRCV       *
0207 0059 CE26              RET             * return.
0208                *
0209                ****************************************************************
0210                *      ================================                        *
0211                *          Xmit interrupt routine.                             *
0212                *      ================================                        *
0213                *  This routine writes xmit data (the contents of DXMT         *
0214                * (123 page0)) to DXR register according to communication      *
0215                * condition, i.e. normal communication or secondary            *
0216                * communication. For normal communication call normal com-     *
0217                * munication routine (NRM). For secondary, call secondary      *
0218                * communication routines (S1 and S2). Because these            *
0219                * routines use 2 levels of nesting, the user is allowed 2      *
0220                * levels of nesting if stack extension is not used.            *
0221                *                                                              *
0222                ****************************************************************
0223 005A 7872     XINT    SST SSTST       * push ST register.
0224 005B C800              LDPK 0          * data pointer page 0.
0225 005C 6071              SACL ACCLST,0   * push ACCL.
0226 005D 6870              SACH ACCHST,0   * push ACCH.
0227 005E 2075              LAC XVECT,0     * load vector address.
0228 005F CE24              CALA            * call xmit routine.
0229 0060 4171              ZALS ACCLST     * pop ACC
0230 0061 4870              ADDH ACCHST
0231 0062 5072              LST SSTST       * pop ST register.
0232 0063 CE00              EINT            * enable interrupt.
0233 0064 CE26              RET             * return.
```

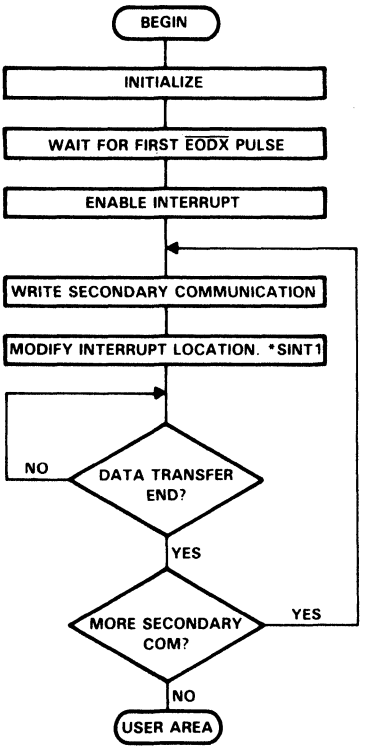
```
0234                    *
0235                    ************************************************************
0236                    *      ===================================                 *
0237                    *        Normal data writing routine.                      *
0238                    *      ===================================                 *
0239                    *  This routine is called when normal communication occurs.*
0240                    * this routine writes xmit data to DXR, and sets xmit flag *
0241                    * (126 page0).                                              *
0242                    ************************************************************
0243 0065 207B  NRM      LAC DXMT,0       * write DXR data.
0244 0066 6001           SACL DXR,0
0245 0067 CAFF           LACK >FF         * set flag.
0246 0068 607E           SACL FXMT
0247 0069 CE26           RET              * return.
0248                    *
0249                    ************************************************************
0250                    *      ====================================                *
0251                    *        Secondary data writing routine 1.                 *
0252                    *      ====================================                *
0253                    *  This routine is called when secondary communication     *
0254                    * occurs. This routine writes secondary data to DXR, and   *
0255                    * modifies the content of XVECT(117 page0) for continuing  *
0256                    * the secondary communication.                             *
0257                    * ************************************************************
0258 006A 207C  S1       LAC D2ND,0       * write DXR 2nd data.
0259 006B 6001           SACL DXR,0
0260 006C 2079           LAC VS2,0        * modify for next XINT.
0261 006D 6075           SACL XVECT,0
0262 006E CE26           RET              * return.
0263                    *
0264                    ************************************************************
0265                    *      ====================================                *
0266                    *        Secondary data writing routine 2.                 *
0267                    *      ====================================                *
0268                    *                                                          *
0269                    *  This routine is called when secondary communication     *
0270                    *  occurs. This routine writes dummy data to DXR to clear  *
0271                    *  the secondary code for the protection of double writing *
0272                    *  the secondary code and reset secondary flag(127 page0), *
0273                    *  modify the content of XVECT(117 page0) for normal XINT. *
0274                    ************************************************************
0275 006F CA00  S2       ZAC              * clear data for protection.
0276 0070 6001           SACL DXR,0       * of double secondary communication.
0277 0071 607F           SACL F2ND        * clear secondary flag.
0278 0072 CAFF           LACK >FF         * set xmit end flag.
0279 0073 607E           SACL FXMT,0
0280 0074 2077           LAC  VNRM,0      * set normal communication vector.
0281 0075 6075           SACL XVECT,0
0282 0076 2073           LAC  INTST,0     * enable all interrupts.
0283 0077 6004           SACL IMR,0
0284 0078 CE26           RET              * return.
```

```
0285                    *
0286                    ************************************************************
0287                    *     ======================                               *
0288                    *      Check  secondary code.    destory DP pointer.       *
0289                    *     ======================           ACC.               *
0290                    *                                                         *
0291                    *  This routine checks whether the data in DXMT (123 page0)*
0292                    * has secondary code or not. If secondary code exists,     *
0293                    * then disable maskable interrupts except XINT, modify the *
0294                    * contents of XVECT(117 page0) for secondary communication,*
0295                    * and set secondary flag. Note that we recommend calling   *
0296                    * this routine to send control words to AIC.               *
0297                    ************************************************************
0298 0079 C800  C2ND    LDPK 0          * data page pointer 0.
0299 007A CA03          LACK 03
0300 007B 606F          SACL TMP0
0301 007C 207B          LAC  DXMT,0     * is this data secondary code ?
0302 007D 4E6F          AND TMP0
0303 007E 106F          SUB TMP0,0      *
0304 007F F680          BZ   C2ND1      * if yes, then next.
     0080 0082
0305 0081 CE26          RET             * else return.
0306                    *
0307 0082 CAFF  C2ND1   LACK >FF        * set secondary flag.
0308 0083 607F          SACL F2ND,0
0309 0084 CA20          LACK >20        * enable only XINT.
0310 0085 6004          SACL IMR,0
0311 0086 2078          LAC VS1,0       *modify vector address for secondary
0312 0087 6075          SACL XVECT,0    * communication.
0313 0088 207B          LAC  DXMT,0     * write primary data to DXR.
0314 0089 6001          SACL DXR,0
0315 008A CE26          RET             * return.
0316                    *
0317                    ************************************************************
0318                    *                                                          *
0319                    *   ======================                                 *
0320                    *    Check first interrupt                                 *
0321                    *   ======================                                 *
0322                    *                                                          *
0323                    *   This routine check whether both first interrupts have  *
0324                    *  occurred. If this routine is called after reset, this   *
0325                    *  routine waits for both interrupts then returns.         *
0326                    ************************************************************
0327 008B 206D  IGR     LAC  FRE,0    * check first interrupt after
0328 008C F680          BZ   IGR      * master reset.
     008D 008B
0329 008E 206C          LAC  FXE,0
0330 008F F680          BZ   IGR
     0090 008B
0331 0091 CE26          RET
```

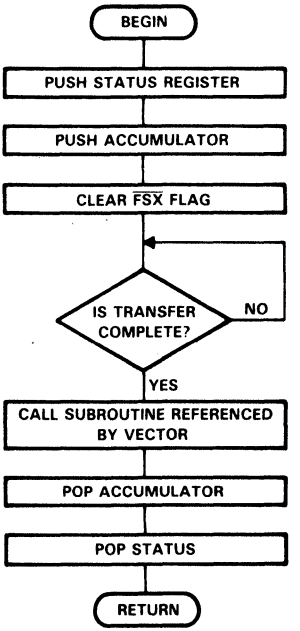
```
0332            *
0333            ****************************************************************
0334            *      ==============================                          *
0335            *        Ignore interrupt routine.                             *
0336            *      ==============================                          *
0337            *  These routines are for the purpose of ignoring the first*
0338            * RINT and XINT after the DSP reset. The routines only set *
0339            * flags and modify each vector address to normal interrupt *
0340            * address but do not read or write to serial ports.        *
0341            * Note that first data that the first data that the AIC    *
0342            * will receive after the DSP reset is 0000H.               *
0343            ****************************************************************
0344 0092 CAFF  IGNRR   LACK >FF
0345 0093 606D          SACL FRE,0
0346 0094 2076          LAC  VRCV,0     * set normal receive address.
0347 0095 6074          SACL RVECT,0    *
0348 0096 CE26          RET             * return.
0349            *
0350 0097 CAFF  IGNRX   LACK >FF
0351 0098 606C          SACL FXE,0
0352 0099 2077          LAC  VNRM,0     * set normal xmit address.
0353 009A 6075          SACL XVECT,0    *
0354 009B CE26          RET             * return.
0355            *
0356                    END
NO ERRORS, NO WARNINGS
```

# C TLC32040 and TMS320C17 Flowcharts and Communication Program

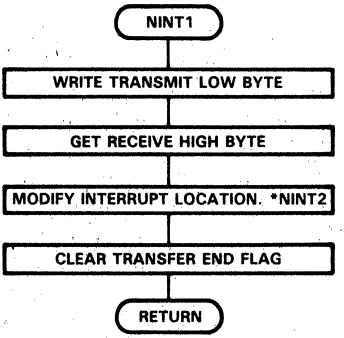
## C.1 Flowcharts



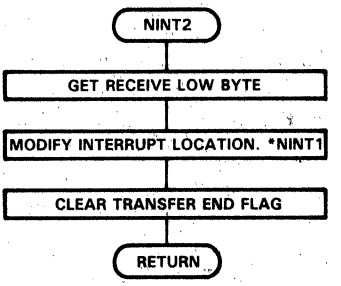

a. MAIN



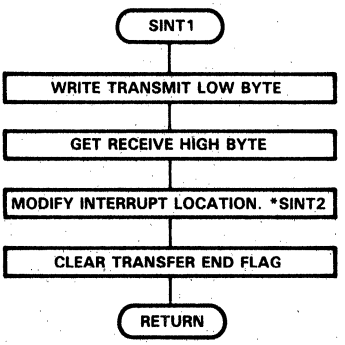

b. INTERRUPT SERVICE ROUTINE



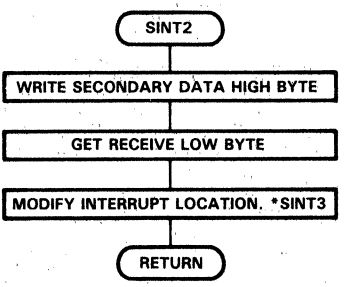

c. PRIMARY COMMUNICATION 1



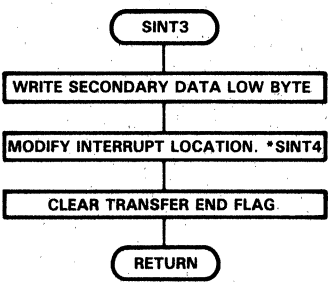

d. PRIMARY COMMUNICATION 2



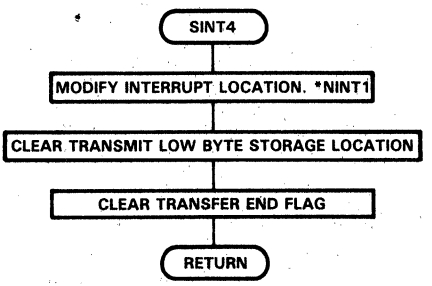

e. PRIMARY-SECONDARY COMMUNICATION 1



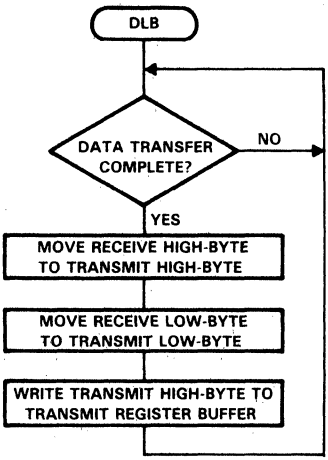

f. PRIMARY-SECONDARY COMMUNICATION 2




g. PRIMARY-SECONDARY COMMUNICATION 3




h. PRIMARY-SECONDARY COMMUNICATION 4




i. DIGITAL LOOPBACK

## C.2 Communication Program List

```
0001            ***************************************************************
0002            *                                                             *
0003            *  =========================================================  *
0004            *      TLC32040 to TMS320C17 Communication Program            *
0005            *                      version 1.1                            *
0006            *                                                             *
0007            *              by Hironori Okubo and Woody Rowand             *
0008            *                    Texas Instruments                        *
0009            *                                                             *
0010            *  =========================================================  *
0011            *                                                             *
0012            *  This program uses the circuit published in the vol. 3      *
0013            *  of Linear and Interface Circuit Applications with the      *
0014            *  following modifications:                                   *
0015            *    1. BIO- of the TMS320C17 must be connected to EODX-      *
0016            *       of the TLC32040.                                      *
0017            *    2. INT- of the TMS320C17 must be connected to FSX-       *
0018            *       of the TLC32040.                                      *
0019            *                                                             *
0020            *  In this configuration, the program will allow the          *
0021            *  TLC32040 to communicate with the TLC320C17 with the        *
0022            *  with the restriction that all interrupts except INT-       *
0023            *  are prohibited and only synchronous communication          *
0024            *  can occur. The program allows the user 2 levels of         *
0025            *  nesting in the main program; the remaining 2 levels        *
0026            *  being reserved for the interrupt vector and sub-           *
0027            *  routines.                                                  *
0028            *                                                             *
0029            *  If desired, this program may be used with the              *
0030            *  TMS32011 Digital Signal Processor with the following       *
0031            *  change. Since the TMS32011 has only sixteen words of       *
0032            *  data RAM on data page 1, all of the registers used by      *
0033            *  this program should be moved to data page 0, except        *
0034            *  for SSTSTK (the temporary storage location for the         *
0035            *  status register) which must remain on page 1 (since        *
0036            *  the SST instruction always addresses page 1).              *
0037            *                                                             *
0038            ***************************************************************
0039    0000    SSTSTK EQU   >00        stack for status (SST) register.
0040    0001    ACHSTK EQU   >01        stack for accumulator high (ACCH).
0041    0002    ACLSTK EQU   >02        stack for accumulator low (ACCL).
0042    0003    RXEFLG EQU   >03        xmit/receive in progress.
0043    0004    DRCV1  EQU   >04        storage for high byte receive data.
0044    0005    DRCV2  EQU   >05        storage for low byte receive data.
0045    0006    DXMT1  EQU   >06        storage for high byte xmit data.
0046    0007    DXMT2  EQU   >07        storage for low byte xmit data.
0047    0008    DXMT3  EQU   >08        storage for high byte secndry data.
0048    0009    DXMT4  EQU   >09        storage for low byte secndry data.
0049    000A    VECT   EQU   >0A        storage for interrupt vector addr.
0050    000B    ANINT1 EQU   >0B        storage for normal xmit/rcv vect 1.
0051    000C    ANINT2 EQU   >0C        storage for normal xmit/rcv vect 2.
0052    000D    ASINT1 EQU   >0D        storage for secndry xmit/rcv vect 1.
```

```
0053      000E  ASINT2 EQU   >0E        storage for secndry xmit/rcv vect 2.
0054      000F  ASINT3 EQU   >0F        storage for secndry xmit/rcv vect 3.
0055      0010  ASINT4 EQU   >10        storage for secndry xmit/rcv vect 4.
0056      0011  CNTREG EQU   >11        storage for control register.
0057      0012  MXINT  EQU   >12        storage for xmit interrupt mask.
0058      0013  CLRX   EQU   >13        storage for xmit interrupt clear.
0059      0014  TEMP   EQU   >14        temporary register.
0060      00FF  FLAG   EQU   >FF        flag set.
0061 0000
0062            *   ======================================
0063            *      Branch to Initialization Routine.
0064            *   ======================================
0065 0000             AORG  >0000
0066 0000 F900        B     INIT        branch to initialization routine.
     0001 0013
0067 0002
0068            ************************************************************
0069            *  ================================                        *
0070            *      Interrupt Service Routine.                          *
0071            *  ================================                        *
0072            *                                                          *
0073            *  To initiate secondary communications, change the        *
0074            *  contents of VECT to the address of the secondary        *
0075            *  communications subroutine and store the secondary       *
0076            *  communication information in DXMT3 and DXMT4.           *
0077            *                                                          *
0078            *  e.g.                                                    *
0079            *       LAC   ASINT1     modify VECT.                      *
0080            *       SACL  VECT                                         *
0081            *          |                                               *
0082            *       LAC   H1         store high-byte of secondary      *
0083            *       SACL  DXMT3      information in DXMT3.             *
0084            *       LAC   H2         store low-byte in DXMT4.          *
0085            *       SACL  DXMT4                                        *
0086            *                                                          *
0087            ************************************************************
0088 0002
0089 0002             AORG  >0002       interrupt vector.
0090 0002
0091 0002 6E01  INTSVC LDPK  1
0092 0003 7C00        SST   SSTSTK      push status register.
0093 0004 5801        SACH  ACHSTK      push accumulator high.
0094 0005 5002        SACL  ACLSTK      push accumulator low.
0095 0006 4813        OUT   CLRX,PA0    make sure FSX-flag is clear.
0096 0007
0097 0007 4011  WAIT1  IN    CNTREG,PA0 read control register.
0098 0008 2011        LAC   CNTREG,PA0 load accumulator with control reg.
0099 0009 7912        AND   MXINT       mask-off xmit interrupt flag.
0100 000A FF00        BZ    WAIT1       loop until xmit interrupt flag is
     000B 0007
```

```
0101            *                       recognized.
0102 000C 200A          LAC   VECT      load acc with interrupt vector.
0103 000D 7F8C          CALA            call appropriate xmit/rcv routines.
0104 000E 6501          ZALH  ACHSTK    pop accumulator high.
0105 000F 7A02          OR    ACLSTK    pop accumulator low.
0106 0010 7B00          LST   SSTSTK    pop status register.
0107 0011 7F82          EINT            enable interrupts.
0108 0012 7F8D          RET             return to main program.
0109 0013
0110            ********************************************************
0111            *  ==================================                  *
0112            *      Initialization after Reset.                     *
0113            *  ==================================                  *
0114            *                                                      *
0115            *  Data RAM locations >80 through >92 are reserved by  *
0116            *  this program. The user must set the status register *
0117            *  at the end of this program with the SST command or  *
0118            *  a combination of SOVM, LDPK etc.                    *
0119            *                                                      *
0120            ********************************************************
0121 0013
0122 0013 7F81  INIT  DINT              disable interrupts.
0123 0014 6E01          LDPK  1         set Data page pointer one.
0124 0015
0125 0015 7F89          ZAC             clear registers.
0126 0016 6880          LARP  0
0127 0017 7083          LARK  0,RXEFLG+>80
0128 0018 50A8          SACL  *+
0129 0019 50A8          SACL  *+
0130 001A 50A8          SACL  *+
0131 001B 50A8          SACL  *+
0132 001C 50A8          SACL  *+
0133 001D 50A8          SACL  *+
0134 001E 50A8          SACL  *+
0135 001F 5088          SACL  *
0136 0020
0137 0020 4906          OUT   DXMT1,PA1 clear transmit registers.
0138 0021 4906          OUT   DXMT1,PA1
0139 0022
0140 0022 7E04          LACK  ?00000100
0141 0023 5012          SACL  MXINT     initialize xmit-int mask.
0142 0024
0143 0024 7E01          LACK  1         prepare for serial port initial-
0144 0025 5014          SACL  TEMP      ization and initialization of regis-
0145 0026 6A14          LT    TEMP      ters containing 16-bit constants.
0146 0027
0147 0027 8094          MPYK  CLX1      initialize interrupt flag clear.
0148 0028 7F8E          PAC
0149 0029 6713          TBLR  CLRX
0150 002A 4813          OUT   CLRX,PA0  configure serial port.
0151 002B
0152 002B 806E          MPYK  NINT1     save normal communication address
0153 002C 7F8E          PAC             to its storage.
```

```
0154 002D 500B          SACL  ANINT1
0155 002E 500A          SACL  VECT        preset interrupt address.
0156 002F
0157 002F 8074          MPYK  NINT2       save normal communication address 2
0158 0030 7F8E          PAC               to its storage.
0159 0031 500C          SACL  ANINT2
0160 0032
0161 0032 807B          MPYK  SINT1       save secondary communication
0162 0033 7F8E          PAC               address 1 to its storage.
0163 0034 500D          SACL  ASINT1
0164 0035
0165 0035 8081          MPYK  SINT2       save secondary communication
0166 0036 7F8E          PAC               address 2 to its storage.
0167 0037 500E          SACL  ASINT2
0168 0038
0169 0038 8087          MPYK  SINT3       save secondary communication
0170 0039 7F8E          PAC               address 3 to its storage.
0171 003A 500F          SACL  ASINT3
0172 003B
0173 003B 808C          MPYK  SINT4       save secondary communication
0174 003C 7F8E          PAC               address 4 to its storage.
0175 003D 5010          SACL  ASINT4
0176 003E
0177 003E F600  IGNOR1  BIOZ  IGNOR2      ignore first FSX pair after reset.
     003F 0042
0178 0040 F900          B     IGNOR1
     0041 003E
0179 0042 F600  IGNOR2  BIOZ  IGNOR2
     0043 0042
0180 0044
0181 0044 7F82          EINT              enable interrupt.
0182 0045
0183              ****************************************************************
0184              *  ================================                            *
0185              *      Main Program (user area)                                *
0186              *  ================================                            *
0187              *                                                              *
0188              *  This program allows the user 2 levels nesting, since        *
0189              *  one level is used as stack for the interrupt and the        *
0190              *  interrupt service routine makes one subroutine call.        *
0191              *  User routines such as digital filtering and secondary-      *
0192              *  communication judgement. Depending on the sampling rate     *
0193              *  the user's routines must write the data to the transmit     *
0194              *  registers within approximately 500 instruction cycles.      *
0195              *                                                              *
0196              *  In the example below, the first two transmissions send      *
0197              *  secondary information to the AIC. The first configures      *
0198              *  the TB and RB registers. The second configures the          *
0199              *  control register.                                           *
0200              *                                                              *
0201              ****************************************************************
0202 0045
```

```
0203 0045 7F89  MAIN   ZAC              prepare first control word.
0204 0046 5006         SACL   DXMT1
0205 0047 7E03         LACK   >03
0206 0048 5007         SACL   DXMT2      should be xxxx xx11.
0207 0049 7E24         LACK   >24
0208 004A 5008         SACL   DXMT3
0209 004B 7E92         LACK   >92
0210 004C 5009         SACL   DXMT4
0211 004D 200D         LAC    ASINT1     set VECT for secondary
0212 004E 500A         SACL   VECT       communications.
0213 004F 4906         OUT    DXMT1,PA1  store first transmit byte in
0214             *                        transmit buffer.
0215 0050 7F89         ZAC
0216 0051 5003         SACL   RXEFLG     clear xmit/rcv end flag.
0217 0052
0218 0052 2003  MAIN1  LAC    RXEFLG
0219 0053 FF00         BZ     MAIN1      wait for data transfer to complete.
     0054 0052
0220 0055
0221 0055 7F89         ZAC              prepare second control word.
0222 0056 5006         SACL   DXMT1
0223 0057 7E03         LACK   >03
0224 0058 5007         SACL   DXMT2
0225 0059 7E00         LACK   >00
0226 005A 5008         SACL   DXMT3
0227 005B 7E67         LACK   >67
0228 005C 5009         SACL   DXMT4
0229 005D 200D         LAC    ASINT1
0230 005E 500A         SACL   VECT
0231 005F 4906         OUT    DXMT1,PA1
0232 0060
0233 0060 7F89         ZAC
0234 0061 5003         SACL   RXEFLG     clear xmit/rcv end flag.
0235 0062
0236             ****************************************************************
0237             *  ================================                            *
0238             *       Digital LoopBack Program                               *
0239             *  ================================                            *
0240             *                                                              *
0241             *  This program serves as an example of what can be done       *
0242             *  in the user area.                                           *
0243             *                                                              *
0244             ****************************************************************
0245 0062
0246 0062 2003  DLB    LAC    RXEFLG     wait for data transfer to complete.
0247 0063 FF00         BZ     DLB
     0064 0062
0248 0065 2004         LAC    DRCV1      move receive data to transit
0249 0066 5006         SACL   DXMT1      registers.
0250 0067 2005         LAC    DRCV2
0251 0068 5007         SACL   DXMT2
0252 0069 4906         OUT    DXMT1,PA1  write first transmit byte to
0253             *                        transmit buffer.
```

```
0254 006A 7F89          ZAC
0255 006B 5003          SACL  RXEFLG      clear rcv/xmit-end flag.
0256 006C F900          B     DLB
     006D 0062
0257 006E
0258                ****************************************************************
0259                *  ================================                            *
0260                *       Normal Interrupt Routines.                             *
0261                *  ================================                            *
0262                *                                                              *
0263                *  These routines destroy the contents of the accumulator      *
0264                *  and the data page pointer, making it necessary to save      *
0265                *  these before the routines begin.                            *
0266                *                                                              *
0267                *  Write the contents of DXMT2 to the transmit buffer and      *
0268                *  read the receive buffer into DRCV1.                         *
0269                *                                                              *
0270                ****************************************************************
0271 006E
0272 006E 4907  NINT1   OUT   DXMT2,PA1   write xmit-low to xmit register.
0273 006F 4104          IN    DRCV1,PA1   read rcv-data-high from rcv reg.
0274 0070 200C          LAC   ANINT2      prepare next interrupt vector.
0275 0071 500A          SACL  VECT
0276 0072 4813          OUT   CLRX,PA0    clear xmit interrupt flag.
0277 0073 7F8D          RET
0278 0074
0279 0074 4105  NINT2   IN    DRCV2,PA1   read receive-data-low from rcv reg.
0280 0075 200B          LAC   ANINT1      prepare next interrupt vector.
0281 0076 500A          SACL  VECT
0282 0077 4813          OUT   CLRX,PA0    clear xmit interrupt flag.
0283 0078 7EFF          LACK  FLAG
0284 0079 5003          SACL  RXEFLG      set xmit/rcv end flag.
0285 007A 7F8D          RET
0286 007B
0287                ****************************************************************
0288                *  ======================================                      *
0289                *       Secondary Interrupt Routines                           *
0290                *  ======================================                      *
0291                *  These routines destroy the contents of the accumulator      *
0292                *  and the data page pointer.                                  *
0293                *                                                              *
0294                *  The following routines write the low byte of primary        *
0295                *  communications and the high and low byte of secondary       *
0296                *  communication. They also read the A/D information from      *
0297                *  the receive registers.                                      *
0298                ****************************************************************
0299 007B
0300 007B 4907  SINT1   OUT   DXMT2,PA1   write xmit-data-low to xmit reg.
0301 007C 4104          IN    DRCV1,PA1   read receive-data-high from rcv reg.
0302 007D 200E          LAC   ASINT2      prepare next interrupt vector.
0303 007E 500A          SACL  VECT
0304 007F 4813          OUT   CLRX,PA0    clear xmit interrupt flag.
0305 0080 7F8D          RET
```

```
0306 0081
0307 0081 4908  SINT2  OUT   DXMT3,PA1  write secondary-data-high to xmit.
0308 0082 4105         IN    DRCV2,PA1  read receive-data-low from rcv.
0309 0083 200F         LAC   ASINT3     prepare next interrupt vector.
0310 0084 500A         SACL  VECT
0311 0085 4813         OUT   CLRX,PA0   clear xmit interrupt flag.
0312 0086 7F8D         RET
0313 0087
0314 0087 4909  SINT3  OUT   DXMT4,PA1  write secondary-data-low to xmit.
0315 0088 2010         LAC   ASINT4     prepare next interrupt vector.
0316 0089 500A         SACL  VECT
0317 008A 4813         OUT   CLRX,PA0   clear xmit interrupt flag.
0318 008B 7F8D         RET
0319 008C
0320 008C 200B  SINT4  LAC   ANINT1     prepare next interrupt vector.
0321 008D 500A         SACL  VECT
0322 008E 4813         OUT   CLRX,PA0   clear xmit interrupt flag.
0323 008F 7F89         ZAC
0324 0090 5007         SACL  DXMT2      clear DXMT2 immediately to eliminate
0325 0091 7EFF         LACK  FLAG       unnexpected secondary communications.
0326 0092 5003         SACL  RXEFLG     set xmit/rcv end flag.
0327 0093 7F8D         RET
0328            ************************************************************
0329            *                                                          *
0330            *    CONTROL REGISTER INFORMATION                          *
0331            *                                                          *
0332            *      SERIAL-PORT CONFIG.     INT. MASK    INT. FLAG      *
0333            *    | 1  0  0 0  1  1  1  0| 0  0  0  1| 0  1  0  0|      *
0334            *     15 14 13 12 11 10  9  8  7  6  5 4  3  2  1  0       *
0335            *                      |                   |  |  |  |_INT  *
0336            *                      |_XF status         |  |  |____FSR  *
0337            *                                         |  |_______FSX  *
0338            *                                         |__________FR   *
0339            *                                                          *
0340            *                                    (write 1's to clear)  *
0341            ************************************************************
0342 0094 8E14  CLX1   DATA  >8E14
0343                   END
NO ERRORS, NO WARNINGS
```

# $I_{CC}$ Requirements of a TMS320C25

**Dave Zalac**

**Digital Signal Processor Products — Semiconductor Group**
**Texas Instruments**

## Introduction

Minimization of total power dissipation of an electronic system is often an important design objective. If tight contraints on supply current are imposed on a design (such as in battery-powered systems), considerations relating to supply current are especially critical. Optimization of such designs is facilitated by an understanding of the tradeoffs involved in the behavior of the supply current requirement of each component of the system.

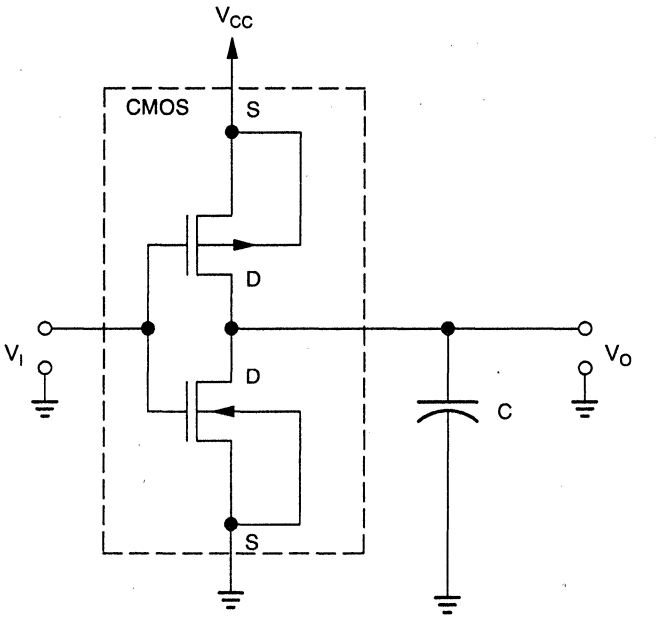
The supply current ($I_{CC}$) requirement of the TMS320C25 digital signal processor varies significantly under different sets of user-imposed conditions. The purpose of this report is to present a characterization of that requirement with respect to operating frequency, supply voltage, output loading, and temperature. Given an understanding of the variations of TMS320C25 $I_{CC}$, the system designer can make appropriate design tradeoffs.

In this report, a description of supply current as time-averaged capacitor-charging will be developed by considering the supply current requirement of a CMOS inverter. Characterization data describing the behavior of the $I_{CC}$ requirement of the TMS320C25 in normal and low-current modes will be presented. The effects on $I_{CC}$ of output loading and temperature variation are discussed. Finally, some low frequency considerations are made.

## Supply Current Requirement of a CMOS Inverter

Some insight into the behavior of supply requirement under varying conditions can be gained through consideration of the basic CMOS converter shown in Figure 1. The capacitor shown in the figure represents the total load capacitance presented by the capacitances of gates connected to the output node, capacitances associated with the inverter structure itself, and interconnect capacitance.

**Figure 1. Basic CMOS Inverter**




If the input voltage is fixed at a logic high or logic low level, one of the two inverter transistors will be non-conducting (off) while the other has a highly conductive channel (on). Under this condition, the supply current is equal to the negligibly small P-N junction leakage current through the off device.

If the input makes a transition from a logic high to a logic low level (or vice-versa), there will be a short interval of time during which both transistors conduct as the inverter is switching. The supply current during this interval is much larger than that under DC-input conditions.

Thus, appreciable current is drawn from the supply only when the inverter is switching. This is in contrast to NMOS logic inverters, in which both the load and driver transistors are always conducting. The absence of a current path under DC-input conditions is thus responsible for the strong dependence of power consumption on operating frequency in CMOS logic circuitry.

Let us assume a transition of the input signal is possible every T seconds. The average supply current can be computed by taking into account the supply currents associated with each of three possible events of the output signal (no transition, high-to-low transition, low-to-high transition).

As already stated, the supply current is negligibly small under static input conditions. Thus we will take the average current to be zero for an interval T wide during which the inverter does not switch.

If the input voltage makes a high-to-low transition, the N-channel transistor will turn off and the capacitor C will be charged through the conducting P-channel device to the output high level of $V_{OH}$ volts. The total charge Q delivered to C is given by

$$Q = C \times (V_{OH} - V_{OL}) \quad (1)$$

The output levels for a typical CMOS inverter approach $V_{OH} = V_{CC}$ and $V_{OL} = 0$ V. Thus $C \times V_{CC}$ coulombs are transferred to C each time the output makes a low-to-high transition. The average charging current during the interval is given by

$$I_C = \frac{Q}{T} = C \times V_{CC} \times f \quad (2)$$

where f = 1/T.

When the output makes a high-to-low transition, C discharges through the N-channel device. The energy stored on the capacitance C is dissipated primarily in the N-type channel. The current sourced by the supply for high-to-low transitions of the output is zero as the P-channel device is off.

Given this description of supply current, low-to-high transitions of the output are the only events during which current is sourced by the power supply. The average supply current is thus given by:

$$I_{AVE} = k \times C \times V_{CC} \times f \quad (3)$$

where k is equal to the normalized number of transitions that are from a low to a high output level. Thus the average supply current is linearly related to output capacitance, supply voltage, and operating frequency. The average power delivered by the supply is the average product of supply voltage and current and is given by:

$$P_{AVE} = (V \times I)_{AVE} = V_{CC} \times I_{AVE} = k \times C \times V_{CC}^2 \times f \quad (4)$$

Similar variations with operating frequency, supply voltage, and node capacitances can be expected of the behavior of the supply current of a complex CMOS integrated circuit. Each time the machine is clocked, charge is transferred to some nodes from either the power supply or from previously charged nodes. Some of the charge on nodes previously at a logic high is lost due to leakage. Additional supply current may be required to replenish the charge on these nodes.

The total charge requirement for a given machine cycle depends, as in the case of the inverter, on the product of $V_{CC}$ and the total capacitance charged during that machine cycle. The total capacitance of the IC is directly related to the area of the die. Thus we expect the IC's supply current requirement to be proportional to supply voltage, operating frequency, and die size.

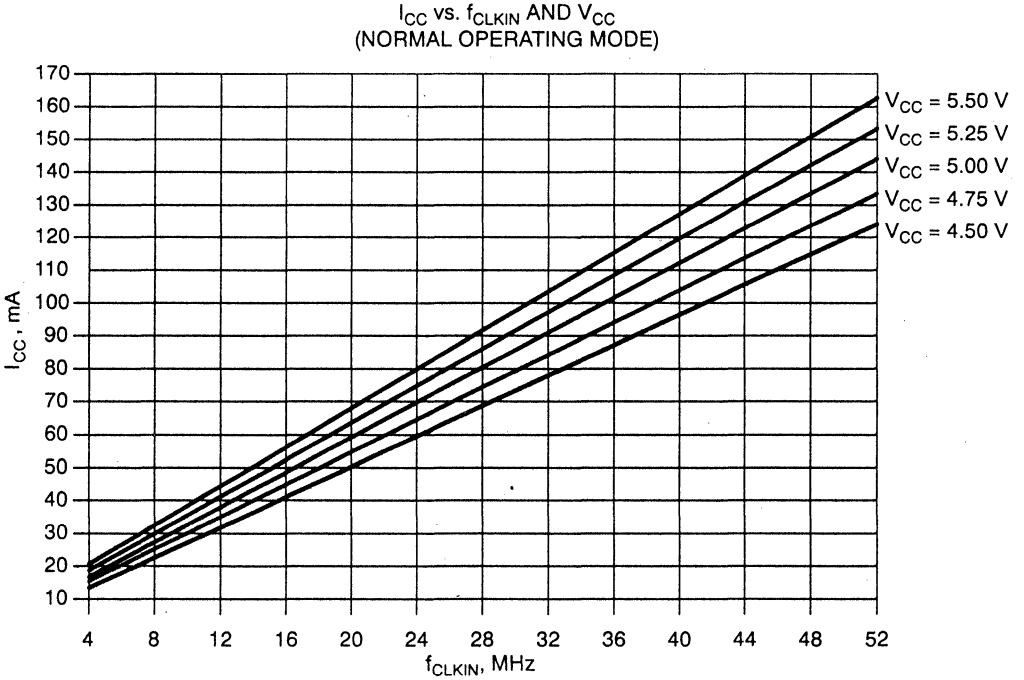
Recall that both $I_{AVE}$ and $P_{AVE}$ for the CMOS inverter are proportional to k. The implication this has for a complex CMOS integrated circuit is that of a relationship between power dissipation and the binary representation of the code being executed by the device and data driven on the external bus. Execution of different pieces of code can result in different supply current requirements under otherwise equal conditions.

Given this information, let us now look specifically at the TMS320C25 with respect to supply current requirement. It is important for the reader to understand that the data presented in the following sections are used only to characterize the way in which $I_{CC}$ varies as externally imposed

conditions are varied. The data should not be taken to supersede the TMS320C25 electrical specification. Furthermore, as a result of process variations and enhancements, the relationship between $I_{CC}$ and external conditions can itself vary. For example, the slopes of the lines in the graphs shown in Figure 2 may increase or decrease somewhat with process parameter variations. In all cases, however, the supply current specification is met by every TMS320C25 device.

Shown in Figure 2 are plots of supply current vs. frequency for five values of supply voltage for the TMS320C25.

**Figure 2. TMS320C25 Supply Current Versus Frequency Plots**




For a fixed value of supply voltage $V_{CC0}$, $I_{CC}$ increases linearly with the frequency of the CLKIN signal with a slope m given by

$$m = 0.37 \times V_{CC0} - 0.71 \text{ milliamperes per megahertz} \quad (5)$$

Note that m = 1.1 mA/MHz at $V_{CC0}$ = 5.0 V. For a fixed operating frequency $f_0$, $I_{CC}$ increases linearly with supply voltage with a slope m given by

$$m = 0.37 \times f_0 + 0.14 \text{ milliamperes per megahertz} \quad (6)$$

Note that m = 15 mA/V at $f_0$ = 40 MHz. Full loading of the device outputs was imposed in the measurement of the values given. This is explained in detail in the following section. The same data is given in tabular form in Table 1.

**Table 1. $I_{CC}$ vs. $f_{CLKIN}$ (MHz) and $V_{CC}$ (V) in Normal Operating Mode**

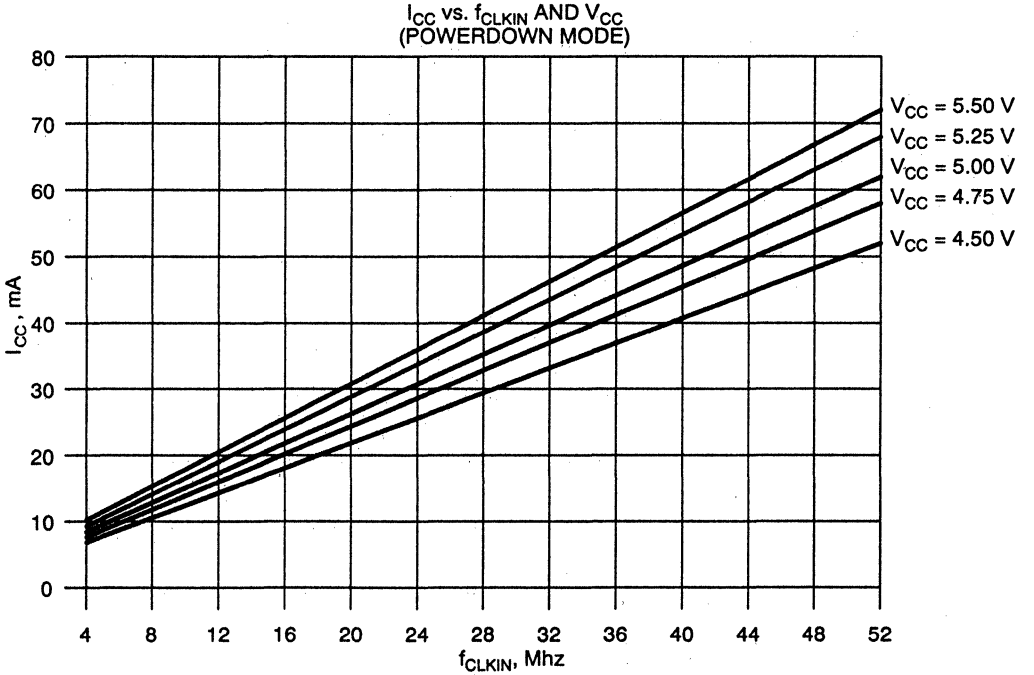
| $f/V_{CC}$ | 4.50 | 4.75 | 5.00 | 5.25 | 5.50 |
|---|---|---|---|---|---|
| 4 | 13 | 15 | 17 | 19 | 21 |
| 8 | 22 | 25 | 28 | 30 | 33 |
| 12 | 32 | 35 | 38 | 41 | 45 |
| 16 | 41 | 45 | 49 | 52 | 56 |
| 20 | 50 | 54 | 59 | 64 | 68 |
| 24 | 59 | 64 | 69 | 75 | 80 |
| 28 | 68 | 74 | 80 | 86 | 92 |
| 32 | 77 | 84 | 90 | 97 | 103 |
| 36 | 87 | 94 | 101 | 108 | 115 |
| 40 | 96 | 103 | 111 | 119 | 127 |
| 44 | 105 | 113 | 122 | 130 | 138 |
| 48 | 114 | 123 | 132 | 141 | 150 |
| 52 | 123 | 133 | 143 | 152 | 162 |

## Variation of $I_{CC}$ with Output Loading

The TMS320C25 has two modes in which the device's supply current requirement is significantly reduced. These modes are referred to as the POWERDOWN and HOLD modes. When in HOLD mode, the address, data, and control lines of the TMS320C25 are put into a high-impedance state. The HOLD mode is invoked by lowering the $\overline{\text{HOLD}}$ input on the device. If the HOLD mode is invoked with the HM status bit set to zero and program execution is from internal memory, execution will continue until an attempt to access external resources is made. Concurrent DMA is possible in this mode. If the HOLD mode is invoked with HM set to one, program execution ceases until the HOLD mode is exited by raising $\overline{\text{HOLD}}$. POWERDOWN mode is identical to HOLD mode with HM = 1 if it is entered by driving $\overline{\text{HOLD}}$ low. However, POWERDOWN mode may also be invoked in software by executing an IDLE instruction. In this case, only the data lines are placed in the high-impedance state. Please refer to the *Second-Generation TMS320 User's Guide* for further details on these modes of operation.

Shown in Figure 3 are plots of supply current vs. CLKIN frequency for five values of supply voltage with the TMS320C25 in POWERDOWN mode.

**Figure 3. TMS320C25 POWERDOWN Mode Plots of Supply Current Versus CLKIN Frequency**




Note that relative to normal operating conditions, the supply current is reduced by approximately 50%. Table 2 shows the same information in tabular form.

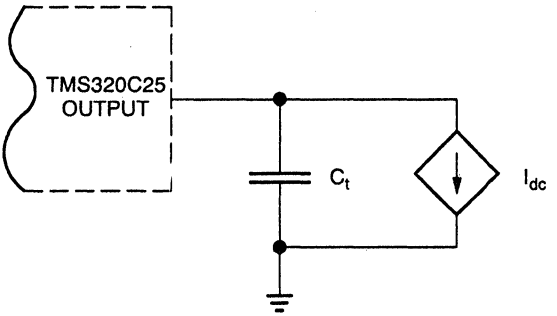
**Table 2. $I_{CC}$ vs. $f_{CLKIN}$ (MHz) and $V_{CC}$ (V) in POWERDOWN Mode**

| f/$V_{CC}$ | 4.50 | 4.75 | 5.00 | 5.25 | 5.50 |
|---|---|---|---|---|---|
| 4 | 8 | 8 | 8 | 9 | 9 |
| 8 | 11 | 12 | 13 | 14 | 15 |
| 12 | 15 | 16 | 18 | 19 | 20 |
| 16 | 19 | 21 | 22 | 24 | 25 |
| 20 | 23 | 25 | 27 | 28 | 30 |
| 24 | 27 | 29 | 31 | 33 | 36 |
| 28 | 30 | 33 | 36 | 38 | 41 |
| 32 | 34 | 37 | 40 | 43 | 46 |
| 36 | 38 | 41 | 45 | 48 | 51 |
| 40 | 42 | 45 | 49 | 53 | 57 |
| 44 | 45 | 50 | 54 | 58 | 62 |
| 48 | 49 | 54 | 58 | 63 | 67 |
| 52 | 53 | 58 | 63 | 67 | 72 |

The above shows that a significant percentage of $I_{CC}$ is spent on driving the device outputs. These include the address, data, and control lines. A typical load is shown in Figure 4. The capaci-

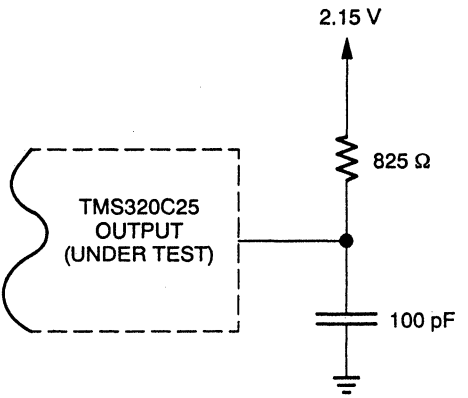
tance Ct is made up of capacitances associated with the output buffer itself, capacitance of the output conductor to ground, and input capacitances of other connected devices. The DC current load (such as that presented by TTL inputs) is represented by the current sink.

**Figure 4. Device Output Typical Load**




An equivalent load connected to each device output during the TMS320C25 $I_{CC}$ measurement is shown in Figure 5. Note that when the output is high, the device sources $(V_{OH} - 2.15)/825 = 303\ \mu A$ at $V_{OH} = 2.4$ V. The device sinks $(2.15 - V_{OL})/825 = 2.12$ mA at $V_{OL} = 0.4$ V when the output is low. When the output is switching, the output buffer drives 100 pF of capacitance in addition to the resistor current.

**Figure 5. Device Output Equivalent Load**




The user can estimate the TMS320C25 supply current (mA) for a particular set of conditions using the following relationship:

$$I_{CC}\ (\textit{normal operating mode}) = (I_{PWRDWN} + 30) + \frac{C}{C_0} \times [I_{NORM} - (I_{PWRDWN} + 30) - 11] + I_{DC} \quad (7)$$

where $I_{PWRDWN}$ is the supply current in POWERDOWN mode taken from Figure 3, $I_{norm}$ is the supply current in normal operation with full output loading taken from Figure 2, C is the average load capacitance imposed on a device output by the user (in pF), and $C_0 = 100$ pF, as shown in Figure 5.

The above expression can be derived as follows. The total supply current is given by

$$I_{CC} = I_{CORE} + I_{AC} + I_{DC} \tag{8}$$

where $I_{CORE}$ is the supply current with the outputs in the high impedance state and with active internal program execution, $I_{AC}$ is the capacitive load current, and $I_{DC}$ is the DC load current.

$I_{CORE}$ depends on supply voltage and operating frequency, but does not depend on output loading. $I_{CORE}$ is given by

$$I_{CORE} = I_{PWRDWN} + 30\ mA \tag{9}$$

$I_{PWRDWN}$ can be taken directly from Figure 3. The supply current with the outputs in the high-impedance state when the device is executing code internally is approximately 30 mA greater than when the device is in POWERDOWN mode at $f_{CLKIN} = 40$ MHz. This value (30 mA) is frequency-dependent, but for simplicity is given as a constant.

$I_{AC}$ for a given capacitive loading is a scaled version of the maximum AC load current sourced under the condition $C = C_0 = 100$ pF. $I_{NORM}$ (taken directly from Figure 2) is related to the maximum AC load current as follows:

$$I_{NORM} = I_{CORE} + I_{AC(max)} + I_{DC(max)} \tag{10}$$

Thus $I_{AC}$ for a given capacitive load is given by

$$\begin{aligned} I_{AC} &= \frac{C}{C_0} \times I_{AC(max)} \\ &= \frac{C}{C_0} \times [I_{NORM} - I_{CORE} - I_{DC(max)}] \end{aligned} \tag{11}$$

Since Figures 2 and 3 provide the values for $I_{NORM}$ and $I_{PWRDWN}$, $I_{DC(max)}$ is the only quantity still needed to evaluate $I_{AC}$. $I_{DC(max)}$ is given by

$$I_{DC(max)} = N \times I_{OH(max)} \tag{12}$$

where N is the number of device outputs driving $I_{OH}$ milliamperes of current and $I_{OH(max)} = 300$ μA, as given in the device electrical specification. For the TMS320C25, an appropriate value of N is 36. Thus $I_{DC(max)}$ is approximately 11 mA.

Finally, $I_{DC}$ is the total DC load current under the user's loading conditions. Plugging this and the results of expressions (9) through (12) into (8) yields the relationship given in (7).

As in the case of a simple inverter, the current requirement of a given output depends on the average number of low-to-high transitions per second, the value of $C_t$, and the magnitude of the output voltage swing.

## Variation of $I_{CC}$ with Ambient Temperature

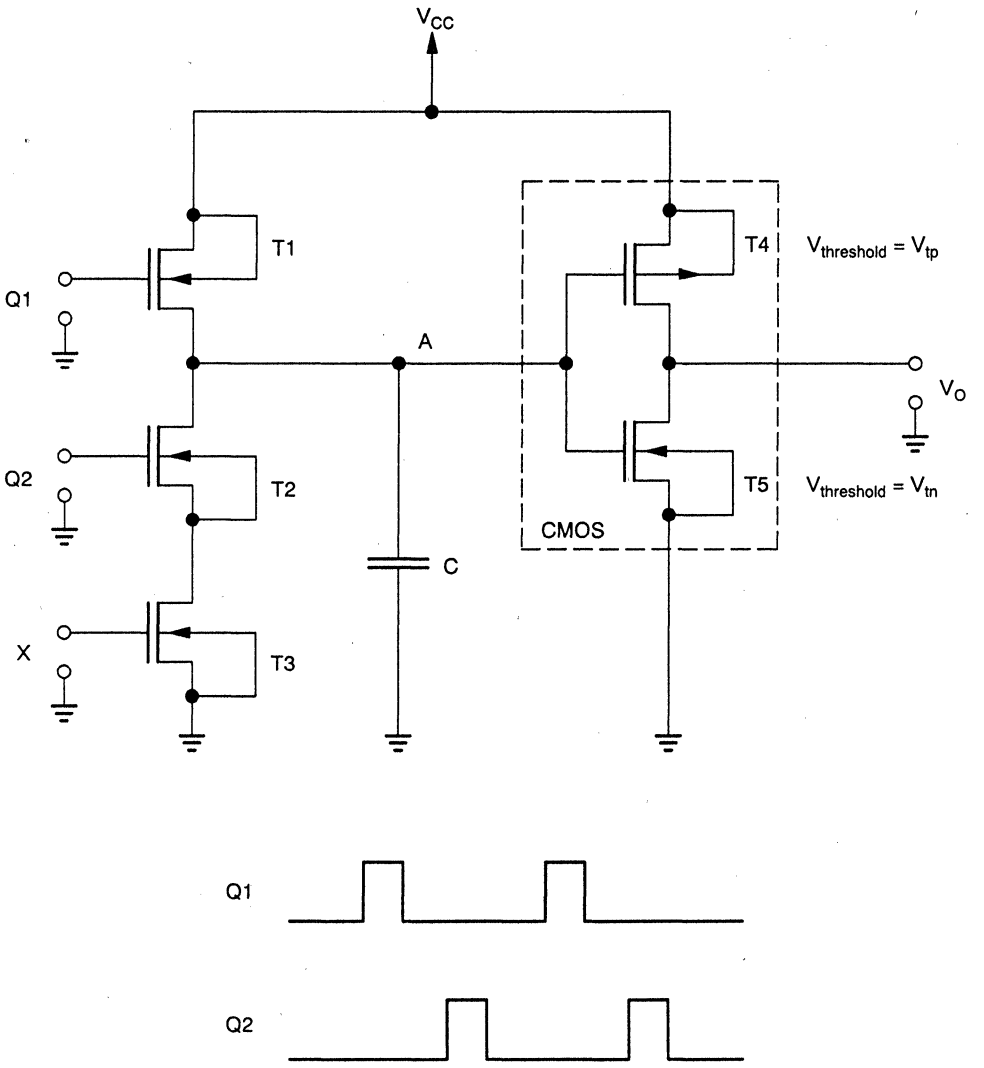
The behavior of supply current with temperature is complex in that there are several temperature-dependent quantities involved that affect $I_{CC}$. (See References [2] and [4] for a detailed discussion.) Variations in MOS transistor threshold voltages and other MOS device parameters, increases in leakage currents, and other variations typically result in a slight decrease in supply current with increasing temperature. However, the supply current of the TMS320C25 exhibits no significant variation with temperature over the 0°C – 70°C range specified in the recommended operating conditions for the device. The TMS320C25 $I_{CC}$ vs. temperature characteristic exhibits a slight downward taper outside this range. The value of $I_{CC}$ at either military temperature range (–55°C to 125°C) endpoint is approximately 10 mA less than that at commercial temperature (0°C to 70°C).

## Low-Frequency Considerations

There are some mechanisms in dynamic logic circuitry that are of issue if the device is operated at a very low clock frequency (less than 100 kHz) that give rise to dramatic increases in supply current. Since the TMS320C25 has dynamic logic circuitry, let us briefly examine a simple dynamic circuit to understand one of these mechanisms.

Shown in Figure 6 is a CMOS inverter, identical to that in Figure 1, being driven by a second inverter. When the clock signal Q1 goes high, node A is pre-charged to a logic 1 level through transistor T1. Note that no current path exists through T2 and T3 during this interval, as Q2 is low and therefore T2 is off. When Q2 goes high, node A is conditionally discharged through T2 and T3. Suppose the input X is low. In this case node A will not go low when Q2 goes high because T3 will be off. During the interval bounded by the falling edge of Q1 and the next rising edge of Q1, the input to the CMOS inverter is held high only by the charge on C; i.e., high impedance is seen looking into the output node A. Under this high-impedance condition, node A is said to be floating. In contrast, recall that if the input to the inverter in Figure 1 is high or low, a current path exists between the output node and a supply node. The output node is thus always driven, as is always the case in conventional static logic. Node A is referred to as a dynamic logic node. The distinguishing feature of dynamic logic is the storage of logic information on high-impedance nodes. While Q1 is low, some of the charge on C is lost due to leakage. If the low interval of Q1 is long enough for the potential at A to drop by one transistor threshold voltage, the P-channel device driven from node A will begin to conduct. If $V_{tn} \leq V_a \leq V_{CC} - |V_{tp}|$, both T4 and T5 will conduct, and the current drawn from the supply will be much larger than the quiescent current required when $V_a$ (voltage on node A) is fixed high or low.

**Figure 6. CMOS Inverter Driven by a Second Inverter**




Again, this high-current condition is not of concern in static logic circuits, and is only of concern in dynamic logic circuits at very low clock frequency. The minimum input clock frequency of the TMS320C25 is specified at 6.7 MHz. This lower limit is orders of magnitude higher than the frequency at which adverse mechanisms in dynamic logic come into play; its choice was driven by other practical considerations such as test time minimization.

Thus the TMS320C25 must be always be clocked at a sufficiently high rate when under power.

## Summary

It has been shown that the TMS320C25 supply current increases approximately linearly with operating frequency and supply voltage. Loading of the device outputs also has a significant effect on the magnitude of $I_{CC}$. The TMS320C25 supply current does not vary significantly with temperature over the 0°C – 70°C commercial temperature range. The device must be clocked at a sufficiently high rate when biased.

The $I_{CC}$ specification for the TMS320C25 is given in Table 3. Also given are several "typical" values. (These values assume 50% output loading, rather than the 100% output loading imposed in the measurement of the data presented in Figure 2 and Table 2.) Note that careful consideration of the behavior of $I_{CC}$ can result in a supply current requirement significantly less than the specified maximum.

**Table 3. $I_{CC}$ Specification and Typical Values**

| $V_{CC}$ | Ta | f(CLKIN) | $I_{CC}$ † | MAX/TYP | NORMAL/POWERDOWN |
|---|---|---|---|---|---|
| 5.25 | 0 | 40.96 | 185 | MAX | NORMAL |
| 5.25 | 0 | 40.96 | 100 | MAX | POWERDOWN |
| 4.75 | † | 40 | 89 | TYP | NORMAL |
| 5.00 | † | 40 | 95 | TYP | NORMAL |
| 5.25 | † | 40 | 101 | TYP | NORMAL |
| 4.75 | † | 20 | 55 | TYP | NORMAL |
| 5.00 | † | 20 | 58 | TYP | NORMAL |
| 5.25 | † | 20 | 61 | TYP | NORMAL |

† $I_{CC}$ is approximately constant over 0°C – 70°C range.

## References


1) Hodges, David A. and Jackson, Horace G., *Analysis and Design of Digital Integrated Circuits*, McGraw-Hill, 1983.
2) Mavor, J., Jack, M.A., and Denyer, P.B., *Introduction to MOS LSI Design*, Addison-Wesley, 1983.
3) Pierret, Robert F., *Field Effect Devices*, (Modular Series on Solid State Devices, Volume 4), Addison Wesley, 1983.
4) Streetman, Ben G., *Solid State Electronic Devices*, Prentice-Hall, 1980.
5) Texas Instruments, *Second-Generation TMS320 User's Guide* (literature number SPRU014), Texas Instruments, 1989.

# An Implementation of a Software UART Using the TMS320C25

**Dave Zalac**

**Digital Signal Processor Products — Semiconductor Group**
**Texas Instruments**

# Introduction

Interfacing to asynchronous devices is a common problem in transmitting to and receiving data from a processing engine such as the TMS320C25 digital signal processor. This report describes a software implementation of a Universal Asynchronous Receiver & Transmitter (UART) that provides the ability to communicate with asynchronous serial devices in a system with a minimum of external hardware.

Asynchronous communications are characterized by the absence of a timing reference such as a clock or framing signal. Various tradeoffs arise from this distinction from synchronous communications in terms of hardware and software requirements and data throughput capacity. Synchronous communications require a timing reference, but otherwise have minimal hardware and software requirements. Asynchronous communications require a mechanism for deriving a timing reference from the received signal. Additionally, various error-checking functions are typically implemented. These requirements impose hardware and/or software overhead that is not imposed in the synchronous case. Moreover, synchronous interfaces can typically support much higher data throughput rates than asynchronous interfaces.

Implementing a UART in software imposes CPU overhead whose acceptability is application-dependent. In applications where the overall data throughput rate is sufficiently low, or in cases in which a UART is to be used only for booting system memory at powerup, use of a software UART may be justifiable. A hardware solution (i.e., a UART IC) may be more appropriate in high data rate applications and in applications requiring low I/O overhead. A detailed analysis of overhead imposed by the TMS320C25 software UART is given later in this report.

A high-speed synchronous serial interface is provided by the on-chip serial port of the TMS320C25. A full description and specification of the serial port may be found in the *Second-Generation TMS320 User's Guide*. [4]

# Overview

The functions provided by a UART are simply the transmission and reception of serial data and the checking and signalling of various error conditions. These functions are described in detail in the following sections.

## Data Format

Shown in Figure 1 is the layout of a word in a format assumed by the UART. Bit 0 is a space (logic low) and is referred to as the start bit. Bits 1 through N are the N data bits of the word with the LSB occupying bit position 1. Typically, N has a value of 5, 6, 7, or 8. The maximum value of N is given by Nmax = 14−M, where M is the

number of stop bits. Bit N+1 is referred to as the parity bit and has a value such that the total number of ones in the word (bits 1 through N+1) is odd if odd parity is selected and even if even parity is selected. Bits N+2 through N+M+1 are referred to as stop bits, and each has a value of one. The total word length WORD_LEN is thus given by WORD_LEN = N+M+2.

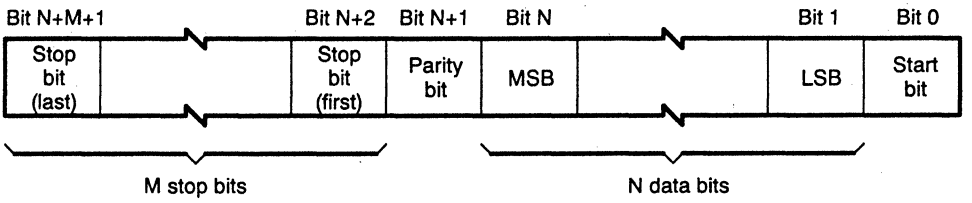



**Figure 1. UART Word Format**

## Data Reception

Reception of a data word starts with detection of the start bit. One way of performing start bit detection is to sample the input data signal at a rate that is large compared to the bit rate, then testing each sample for a space (logic low). An optional check can be performed to verify that the first logic low detected represents a valid start bit and not just noise. This check is performed by testing that the input signal is low one-half-bit duration after the start bit has been detected.

Once the start bit has been detected, the UART simply recovers the data from the input signal and keeps track of the data parity. The parity is checked against the received parity bit after all the data bits have been received. Finally, the integrity of the word framing is checked by testing that the input signal is high when the first stop bit is expected.

## Data Transmission

Transmission is considerably simpler than reception in that timing information does not have to be recovered from an asynchronous signal. Furthermore, no error checking is performed by the UART transmitter. Transmitting a data word is preceded by appropriately formatting the data to be transmitted; i.e., adding start, stop and parity bits. Formatting is done in TMS320C25 software. The output signal is generated from the data and appears on the UART's output signal line.

# Implementation

The UART implementation described in this report makes use of two TMS320C25 general-purpose I/O pins (XF and BIO/) and the timer interrupt. The input signal is received on the $\overline{\text{BIO}}$ pin via the TMS320C25 BIOZ instruction. The output signal appears on the general-purpose flag pin XF. The state of XF is controlled in software via the SXF and RXF instructions. The TMS320C25 serial port is not used. As shown if Figure 2,

the transmitter and receiver are "serviced" each time a timer interrupt is generated. The timer interrupt rate is an integer multiple (K) of the bit rate.

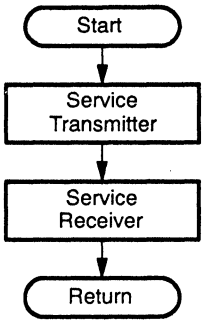



**Figure 2. Timer Interrupt Service Routine**

Several pieces of code comprise the UART software:

1. Timer interrupt service routine
2. UART_INIT initialization routine
3. XMT routine
4. RCV routine
5. PUT_DATA
6. GET_DATA
7. XCOMPOSE

The UART transmitter and receiver are located in the timer interrupt service routine. (No context save/restore is included in this interrupt service routine. Refer to the "Precautions" section for details.) UART_INIT initializes the UART with the values appearing in the assembly-time constants section of the source listing. XMT and RCV are user-written routines that interface the UART to the user's program. XCOMPOSE, PUT_DATA and GET_DATA are auxiliary routines available to the user for executing UART interface housekeeping tasks. Each of these seven routines is described in detail later in this report.

Figure 3 shows the UART structure and the transmit and receive data paths. The transmit and receive buffers TDATA and RDATA are 16-bit wide TMS320C25 on-chip data memory locations. The transmit and receive software shift registers TSHF and RSHF are also located in data memory. The UART status word USTAT maintains UART status and error information. USTAT will be discussed in detail later in this report. The value of USTAT is written to I/O port UARTPORT each time USTAT is updated, thus allowing the capability of externally monitoring the UART status. The input and output pins ($\overline{BIO}$ and XF) may be interfaced to RS-232-compatible transmit and receive lines. Finally, the locations in data memory for transmit and receive data are pointed at by two TMS320C25 auxiliary registers AR(OPT__PTR) and AR(INP__PTR), respectively.

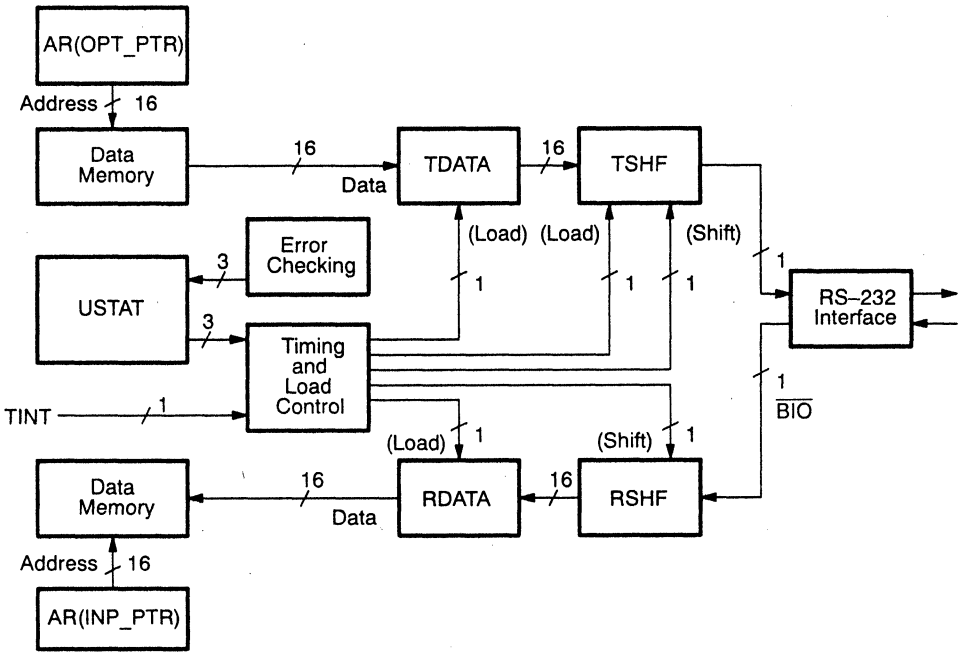



**Figure 3. UART Architecture**

All UART variables are mapped into TMS320C25 on-chip RAM block B2. The code size is 332 words and can be executed from on-chip ROM, EPROM (TMS320E25), or off-chip program memory. The maximum bit rate supported is 19.2 kilobits per second (full duplex).

## Transmitter

Figure 4 shows a flowchart of the transmitter routine. When the transmitter completes transmission of the current word, a new word (TDATA) is loaded into the transmitter software shift register if the TDA (Transmit Data Available) flag is set.

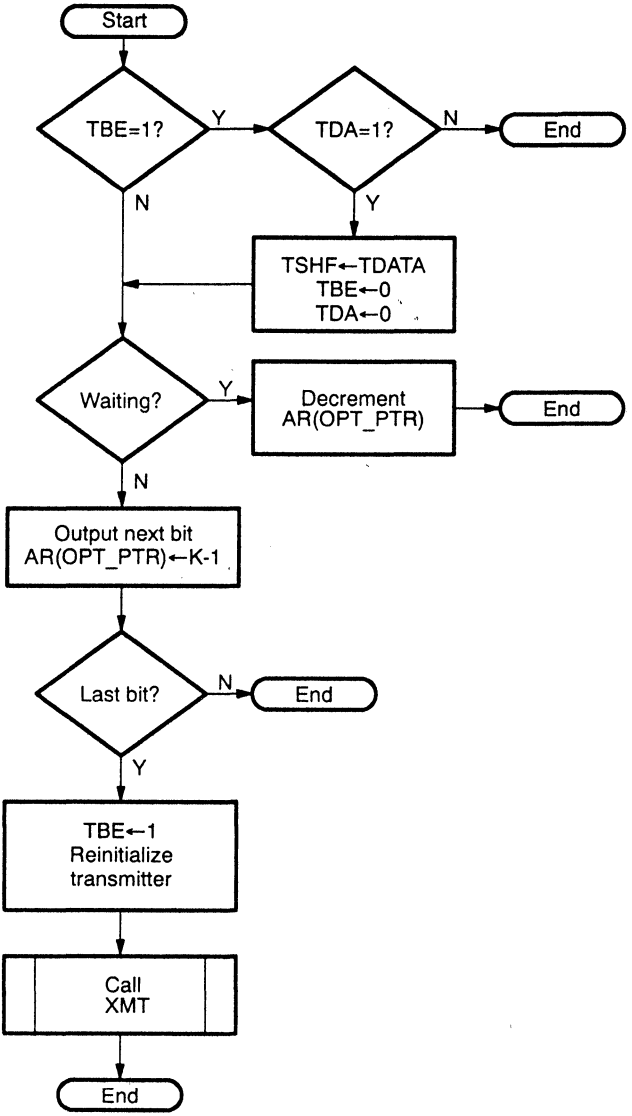



**Figure 4. Transmitter Routine**

TDA is one of 6 flags residing in the UART status word (see Figure 5). The word to be transmitted is shifted out on the XF pin at the user-specified bit rate.

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| Not used | Not used | TDA | RDE | RDA | ROR | FRM | RPE |

**Figure 5. UART Status Word**

Feeding data to the transmitter consists of three steps: appropriately reformatting the data to be transmitted (i.e., adding start, parity, and stop bits), loading the data into the UART variable TDATA, and indicating to the UART that valid transmit data is present in TDATA by setting the TDA flag. The first step can be accomplished by pointing AR(OPT__PTR) at the data to be transmitted and calling auxiliary routine XCOMPOSE. XCOMPOSE does an in-place reformatting of the data per the values in the UART parameter variables (see source listing). Routine PUT__DATA may be called to load TDATA and set TDA.

Each time the transmit shift register empties, a call to XMT is made. Management of AR(OPT__PTR) and calls to XCOMPOSE and PUT__DATA may be made from XMT. Alternatively, these functions can be executed from the user's program. This is the preferred approach, because all code in XMT adds to the maximum path length through the timer interrupt routine and thus decreases the maximum bit rate. However, calls to PUT__DATA from the user's program should be made only if TDA = 0. If this condition is not satisfied, the current word to be transmitted will be overwritten. (The condition TDA = 0 is guaranteed if PUT__DATA calls are made from XMT and needn't be checked.)

Initiation of transmission of the first word in a string of words (string = one or more words) must be made from the user's program by calling XCOMPOSE and PUT__DATA, as no XMT calls can be made until the transmitter is started.

The status of the transmitter can be ascertained by reading bit 5 of the UART status word, as shown in Figure 5:

USTAT

BIT 5 Transmit Data Available TDA = 1 indicates to the UART that valid transmit data is present in TDATA.

## Receiver

Shown in Figure 6 is a flowchart of the receiver routine. The state of the receiver is indicated by the value of RSTAT and bits 0 - 4 of the UART status word as shown below.

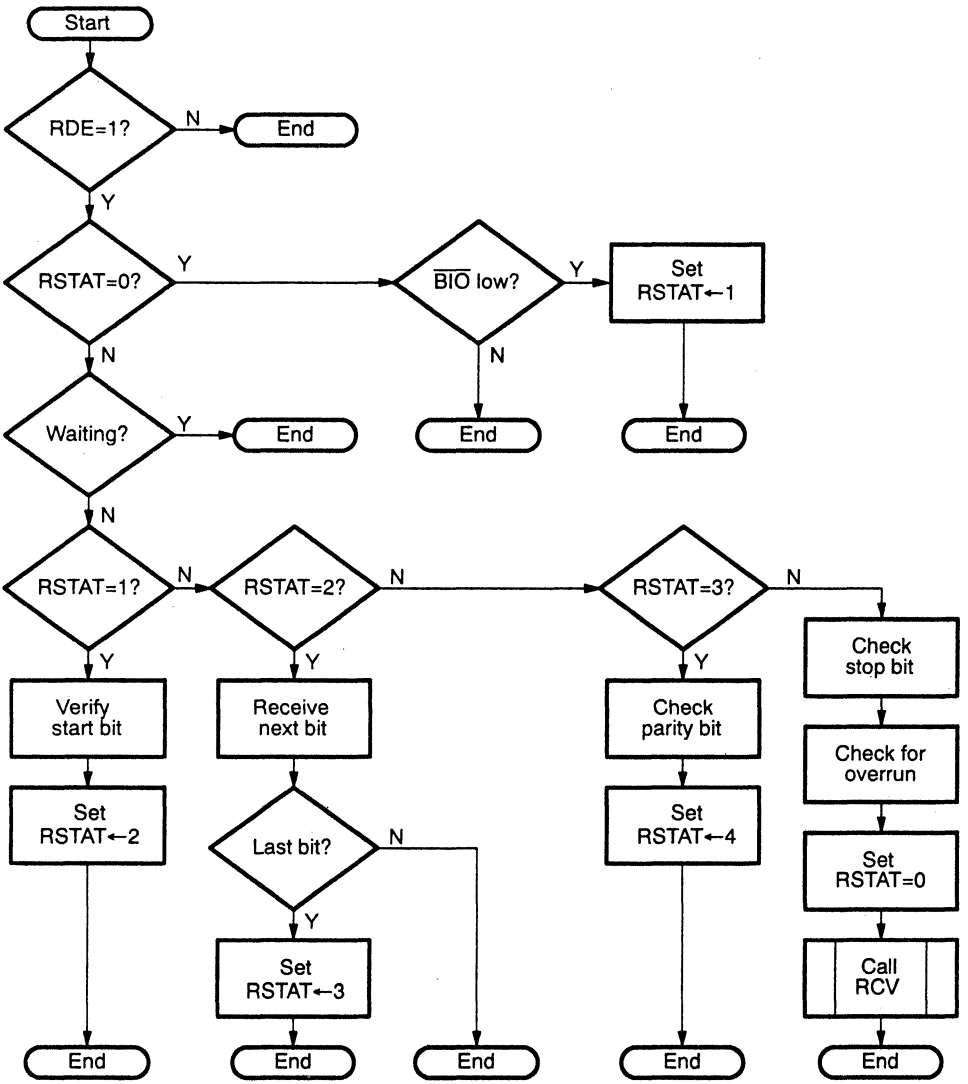



**Figure 6. Receiver Routine**

| RSTAT | Receiver Status |
|---|---|
| 0 | Waiting for start bit |
| 1 | Waiting for start bit center |
| 2 | Waiting for data bit |
| 3 | Waiting for parity bit |
| 4 | Waiting for stop bit |

USTAT

| | | |
|---|---|---|
| BIT 0 | Receive Parity Error | If receive parity checking is active (RPACTIVE = 1), RPE is set if a parity error is detected. |
| BIT 1 | Framing Error | This bit is set if a logic low is sensed on $\overline{BIO}$ when the first stop bit is expected. |
| BIT 2 | Receiver Overrun | This bit is set if RDA is not cleared before reception of the next word is completed. |
| BIT 3 | Receive Data Available | This bit is set when reception of a word is completed and indicates the presence of valid data in RDATA. |
| BIT 4 | Receive Data Enable | The receiver routine is bypassed if this bit is a zero. |

Reception is initiated by setting RDE. Each time a complete word is received, a call to RCV is made. RCV can call GET_DATA to copy the new data to the location pointed at by AR(INP_PTR) and to clear RDA. If RDA is indeed cleared by every call to RCV, some overhead can be eliminated by deleting the setting, clearing, and checking of RDA because the overrun detect function is superfluous in this case.

The received data in RDATA is right-justified with the LSB in the zero'th bit position and with start, stop, and parity bits stripped.

## Overhead and Optimization

The overhead imposed by the UART is primarily determined by the length of the timer interrupt service routine and the timer interrupt rate (K * bit rate). An expression for overhead is given below:

$$\text{overhead } (\%) = K \times (\text{bit rate}) \times (T + R) \times T_{c(C)} \times 100\%$$

K = (timer interrupt rate)/(bit rate). $T_{c(C)}$ is the period of CLKOUT1 and CLKOUT2. In the timer interrupt service routine shown in the source listing, T and R are given by

$$T = (8 \times \text{WORD\_LEN} \times K + 14 \times \text{WORD\_LEN} + 15)/(\text{WORD\_LEN} \times K)$$

$$R = (17 \times \text{WORD\_LEN} \times K + 36 \times \text{WORD\_LEN} + 169)/(\text{WORD\_LEN} \times K)$$

The values of T and R at WORD_LEN = 10 and K = 16 are T = 9 cycles per timer interrupt and R = 20 cycles per timer interrupt. (These are the correct values when the timer interrupt service routine is in zero-waitstate external program memory. T and R have smaller values if the service routine is in internal program memory.)

T and R represent the average path lengths (in processor cycles) through the transmitter and receiver routines, respectively, in continuous full-duplex operation. Values of overhead for several bit rates and values of K are tabulated in Table 1. Overhead associated with XMT, RCV, GET_DATA, PUT_DATA, and XCOMPOSE code is not included. Note that continuous full-duplex operation constitutes worst-case scenario, a scenario unlikely in applications using asynchronous I/O.

**Table 1. UART Overhead (%) vs. Bit Rate (r in bps) and K for WORD_LEN=10†**

| K/r | 300 | 1200 | 2400 | 4800 | 9600 | 19,200 |
|---|---|---|---|---|---|---|
| 6 | 1 | 3 | 5 | 10 | 21 | 42 |
| 8 | 1 | 3 | 6 | 13 | 26 | 51 |
| 10 | 1 | 4 | 8 | 15 | 31 | – |
| 12 | 1 | 4 | 9 | 18 | 35 | – |
| 14 | 1 | 5 | 10 | 20 | 40 | – |
| 16 | 1 | 6 | 11 | 22 | 45 | – |

†Overhead imposed by the UART is relatively insensitive to WORD_LEN.

There are several ways the user can modify the UART code to reduce the values of T and/or R. Some of these involve eliminating the setting, clearing, and checking of flags in the USTAT register that are not necessary in a fixed configuration. Others involve streamlining of the interface between the UART software and the user's program.

## UART Configuration

The initial values of the UART parameters (e.g., bit rate and parity type) appear in the assembly-time constants section of the source listing given in the appendix of this report. Note that an initial-value constant exists for each UART parameter and has the same name as its corresponding parameter, but with an "I" prefix. If the UART is to be run in a fixed configuration, the user needs only to modify the initial value parameters and re-assemble and link the program. However, if the configuration is to be modified "on the fly", the following measures need to be taken:

| To Respecify | Modify |
|---|---|
| RPACTIVE | RPACTIVE (0 or 1) |
| K | KM1, K2M1, and TINTPER (KM1 = K-1; K2M1 = K/2-1) |
| N | N and WORDLEN, CALL PARINIT |
| WORD_LEN | WORD_LEN |
| ODD | ODD, CALL PARINIT (initialize parity templates) |
| TINTPER | TINTPER |
| # of stop bits | WORD_LEN (# of stop bits = WORD_LEN-N-2) |

## Precautions

No context save/restore is provided in the timer interrupt service routine as the user will want to write and optimize this part of the routine for his own application. The timer interrupt routine affects the following registers and memory locations:

| CPU Registers | Memory Locations |
|---|---|
| Accumulator | 60h-77h (RAM block B2) |
| T register | dma<AR(INP_PTR)> |
| P register | TIM register |
| Auxiliary registers 1-7 | PRD register |
| Status regs ST0 & ST1 | |

The timer interrupt routine uses two levels of stack plus as many levels as are required to accommodate subroutine calls from XMT and RCV.

If the PRD register contains a value less than 64 (19.2 kbps @ K = 8 or 9.6 kbps @ K = 16), the sampling of some receive bits may be significantly delayed from bit interval centers and some transmit signal edges may be delayed.

The actual transmit and receiver-sampling bit rate r is given by

$$r = 1/[P \times K \times T_{c(C)}],$$

where P is the sum of the contents of the PRD register and one. If no integer value of P exists for a specified r, K, and $T_{c(C)}$, the receiver should typically be allowed to run at the rate closest to but greater than the ideal bit rate.

If the receiver bit rate is exactly equal to the transmit bit rate of the external transmitting equipment, the sampling of incoming bits will occur at times close to the centers of the corresponding bit intervals. Some error is introduced by the latency between the falling edge of the start bit and the time at which the start bit is detected. The maximum value of that error ($e_1$) is equal to one period of the timer interrupt.

Additional error is introduced if the receiver bit rate differs from the bit rate of the incoming data stream. Let the bit duration dictated by the timer interrupt rate be denoted by $T_1$ and let the bit duration of the incoming data be denoted by $T_2$. The error introduced by the inequality of $T_1$ and $T_2$ ($e_2$) for the n'th bit is given by

$$e_{2(n)} = (T_1 - T_2) \times (n - 1/2) \quad (1)$$

The start bit corresponds to n = 1. The cumulative error for one word is equal to $e_2$ evaluated at n=WORD_LEN.

Still another source of error is the latency associated with multicycle instructions. Should a timer interrupt occur during execution of a multicycle instruction or repeat loop, an error $e_3$ will delay the sampling of $\overline{BIO}$ by a minimum of zero and a maximum of I−1 cycles, where I is the length (in cycles) of the longest instruction or repeat loop.

The total difference between the sampling time and a corresponding bit interval center is the sum of $e_1$, $e_2$, and $e_3$. In general, the absolute value of the sum of $e_1$, $e_2$, and $e_3$ must be less than one-half the duration of one bit in the incoming data stream in order that all sampling instants fall in corresponding bit intervals; i.e.,

$$|e_1 + e_2 + e_3| < T_2/2 \quad (2)$$

The above constraint is appropriate for a receive signal having negligible rise and fall times and equal space and mark durations. If either of these conditions is not satisfied, the constraint expression should be modified accordingly.

## Worst-Case Error Analysis

Following are descriptions of the two worst-case scenarios in terms of the three error components. The results of this analysis are then plugged in the constraint expression given in (2) to yield a description of the error constraint in terms of rate difference, K and I.

If the incoming data rate is higher than the receiver bit rate, $e_1$, $e_2$, and $e_3$ are all greater than or equal to zero. The worst-case value of $e_1$ is its maximum value given by

$$e_{1(max)} = t_{c(C)} \times [< \text{PRD reg} > + 1]$$
$$= T_1/K$$

The $e_2$ contribution is the cumulative error resulting from the inequality of $T_1$ and $T_2$ and is given by

$$e_{2(max)} = (T_1 - T_2) \times (\text{WORD\_LEN} - 1/2)$$

The worst-case value of $e_3$ is given by

$$e_{3(max)} = (I_{max} - 1) \times t_{c(C)}$$

If the incoming data rate is lower than the receiver bit rate, $e_1$, $e_2$, and $e_3$ are all less than or equal to zero. The worst-case value of $e_1$ is its minimum value given by

$$e_{1(min)} = 0$$

The $e_2$ contribution is the cumulative error resulting from the inequality of $T_1$ and $T_2$ and is given by

$$e_{2(min)} = (T_1 - T_2) \times (WORD\_LEN - 1/2)$$

The worst-case value of $e_3$ is given by

$$e_{3(min)} = 0$$

The error constraint (2) is thus satisfied if the following pair of inequalities is satisfied:

$$e_{1(min)} + e_{2(min)} + e_{3(min)} > -T_2/2 \qquad (3)$$

$$e_{1(max)} + e_{2(max)} + e_{3(max)} < T_2/2$$

where expressions for the extreme values of each error component are given above.

The inequalities in (3) specify the overall constraint on maximum rate difference, minimum value of K and maximum value of I. For example, suppose

$T_2$ = 0.100 ms
$T_1$ = 0.103 ms
I = 20
$t_{c(C)}$ = 100 ns
WORD__LEN = 10

Since $T_1$ is sufficiently close to $T_2$, the first inequality in (3) is satisfied:

$$0 + [(103 \times 10-6) - (100 \times 10^{-6})] \times (10-0.5) + 0 > (-100 \times 10^{-6})/2$$

Evaluation of the second inequality in (3) yields

$$(103 \times 10^{-6})/K + [(103 \times 10^{-6}) - (100 \times 10^{-6})] \times (10 - 0.5) + (20 - 1)$$

$$\times (100 \times 10^{-9}) < (100 \times 10^{-6})/2$$

or

$$K \geq 6$$

Thus (2) translates into a specification for the minimum value of K for a given $T_1$, $T_2$, I, $t_{c(C)}$, and WORD__LEN.

In summary, considerations must be made with respect to the data rate of the external transmitting equipment, the data rate resulting from the timer interrupt rate, and the latencies associated with start bit detection and multicycle instructions. The two inequalities in (2) must be satisfied for all bits for proper UART operation.

## Loopback Test

In the source code given, the XMT and RCV routines are structured to implement a loopback test at 9600 bps, 7 data bits, 1 stop bit and odd parity. The circuit shown in Figure 7 can be used to interface to RS-232-compatible transmit and receive lines. No other RS-232 signals are supported.

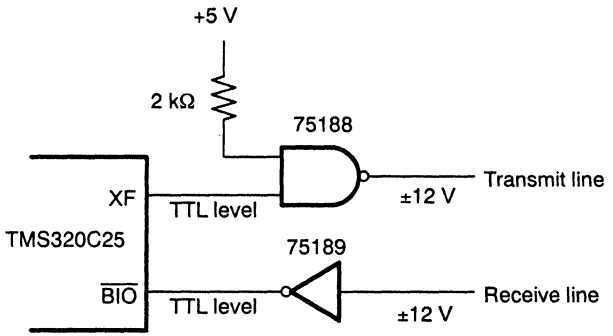



**Figure 7. RS-232 Interface**

## References


[1] *TMS7000 Family Data Manual*, literature number SPND001C, Texas Instruments, 1989.

[2] McNamara, John, *Technical Aspects of Data Communications*, Digital Equipment Corporation, 1982.

[3] *Data Communications Standards*, McGraw-Hill, 1982.

[4] *Second-Generation TMS320 User's Guide*, literature number SPRU014A, Texas Instruments, 1989.

```
*****************************************************************
*
*   SOFTWARE UART USING THE TMS320C25
*
*   WRITTEN BY: DAVE ZALAC
*               TEXAS INSTRUMENTS, INC.
*               1/31/89
*
*   PROGRAM MEMORY REQUIREMENT: 332   WORDS
*   DATA MEMORY REQUIREMENT:     23   WORDS
*   MAXIMUM BIT RATE:           19.2 KBPS
*
*****************************************************************
*
*   ASSEMBLY-TIME CONSTANTS SECTION
*
*****************************************************************
*
*   STATUS BIT MASKS
*
*****************************************************************
*
RPE_MSK   .set    001h          ; RECEIVE PARITY ERROR
*
RPE_SET   .set    001h
RPE_CLR   .set    0FEh
*
FRMERR_MSK  .set  002h          ; RECEIVE FRAMING ERROR
FRMERR_SET  .set  002h
FRMERR_CLR  .set  0FDh
*
ROR_MSK   .set    004h          ; RECEIVE DATA OVERRUN
ROR_SET   .set    004h
ROR_CLR   .set    0FBh
*
RDA_MSK   .set    008h          ; RECEIVE DATA AVAILABLE
RDA_SET   .set    008h
RDA_CLR   .set    0F7h
*
RDE_MSK   .set    010h          ; RECEIVER ENABLE
RDE_SET   .set    010h
RDE_CLR   .set    0EFh
*
TDA_MSK   .set    020h          ; TRANSMIT DATA AVAILABLE
TDA_SET   .set    020h
TDA_CLR   .set    0DFh
*
*****************************************************************
*
*   USTAT REGISTER
*
*   BIT #          5      4      3      2      1      0
*   FLAG           TDA    RDE    RDA    ROR    FRM    RPE
*   INITIAL VALUE  0      0      0      0      0      0
*
*****************************************************************
*
UARTPORT  .set    0             ; PORT ADDRESS OF UART STATUS REGISTER
*
*****************************************************************
*
*   INITIAL VALUES OF UART PARAMETERS
*
*****************************************************************
*
IRPACTIVE .set    1             ; RPACTIVE = 1 INDICATES RCV PARITY
*                               ;   CHECKING ON
IKM1      .set    15            ; TINT RATE = K*BIT RATE; KM1 = K-1
IK2M1     .set    7             ; K2M1 = K/2-1
INM1      .set    6             ; NUMBER OF DATA BITS/WORD
IWORD_LEN .set    10            ; TOTAL # OF BITS/WORD
IODD      .set    1             ; ODD = 1 SELECTS ODD PARITY
*                               ; ODD = 0 SELECTS EVEN PARITY
ITINTPER  .set    259           ; TINTPER = TIMER INTERRUPT PERIOD IN
*                               ; CLKOUT1 CYCLES. TINTPER SHOULD BE SET TO
*                               ; (1/(K*R*TC(C)))-1, ROUNDED ROUNDED DOWN
*                               ; TO THE NEAREST INTEGER. R IS THE BIT RATE
*                               ; IN BPS.
*
*****************************************************************
*
*   REGISTER ASSIGNMENTS
*
*****************************************************************
*
TWG       .set    7             ; DIVIDES TINT RATE BY K (XMT)
RWG       .set    6             ; DIVIDES TINT RATE BY K (RCV)
XBITS_REG .set    5             ; USED BY TRANSMITTER TO COUNT DATA BITS
RBITS     .set    4             ; USED BY RECEIVER TO COUNT DATA BITS
XBREG     .set    3             ; USED BY XCOMPOSE TO COUNT DATA BITS
INP_PTR   .set    2             ; POINTS AT LOCATION TO PUT RECEIVED DATA
OPT_PTR   .set    1             ; POINTS AT DATA TO BE TRANSMITTED
*
*****************************************************************
*
*   END ASSEMBLY-TIME CONSTANTS SECTION
*
*****************************************************************
*
*
*   DATA MEMORY SPACE RESERVATION
*
*****************************************************************
*
*   VARIABLES
*
*****************************************************************
*
```

```
        .bss    RSTAT,1         ; RECEIVER STATUS
        .bss    USTAT,1         ; UART STATUS REGISTER
        .bss    SHF,1           ; SHIFT FACTOR
        .bss    INP,1           ; RECEIVED BIT
        .bss    DWORD,1         ; RECEIVER SHIFT REGISTER
        .bss    RPAR,1          ; LOCALLY-GENERATED PARITY
        .bss    RPARI,1         ; INITIAL VALUE OF RPAR
        .bss    XSTOP,1         ; STOP BIT STRING
        .bss    XPARI,1         ; INITIAL VALUE OF XSTOP
        .bss    XPAR,1          ; TRANSMIT APRITY
        .bss    RDATA,1         ; RECEIVE BUFFER
        .bss    XPARTOG,1       ; TRANSMIT PARITY TOGGLE MASK
        .bss    TDATA,1         ; TRANSMIT DATA
        .bss    TSHF,1          ; TRANSMIT SHIFT REGISTER
        .bss    TEMP,1          ; SCRATCH VARIABLE
        .bss    RPACTIVE,1      ; RECEIVE PARITY ACTIVE (0/1)
        .bss    KM1,1           ; K-1
        .bss    K2M1,1          ; K/2-1
        .bss    NM1,1           ; N-1
        .bss    WORD_LEN,1      ; OVERALL WORD LENGTH
        .bss    ODD,1           ; SELECTS ODD/EVEN PARITY (1/0)
        .bss    TINTPER,1       ; TIMER INTERRUPT PERIOD
        .bss    TBE,1           ; TRANSMIT BUFFER EMPTY
*
*****************************************************************
*
*   PROGRAM SECTION
*
*****************************************************************
*
        .sect   "vectors"
*
RS      B       MAIN
*
        .space  ((RS+24-$)*16) ; POSITION TINT VECTOR
*
TIMINT  B       XMIT           ; BRANCH TO XMT/RCV ROUTINE
*
        .text
*
MAIN    LDPK    0              ; INITIALIZE
        ZAC
        SACL    5              ; NO GLOBSL MEMORY
        SOVM                   ; SET OVERFLOW MODE
        SSXM                   ; SET SIGN-EXT. MODE
        SPM     0              ; P-REG SHIFT = 0 BITS
        CNFD                   ; CONFIGURE BLOCK B0 AS DATA MEMORY
        CALL    UART_INIT      ; INITIALIZE UART
*
        LACK    10h
        SACL    USTAT          ; ENABLE RECEIVER
*
*****************************************************************
*
*   THE USER'S PROGRAM SHOULD APPEAR HERE.
*
SELF    B       SELF
*
*   END PROGRAM SECTION
*
*****************************************************************
*
*   UART INITIALIZATION ROUTINE
*
*   THE FOLLOWING CODE INITIALIZES THE UART PER THE VALUES IN THE ASSEMBLY-
*   TIME CONSTANTS SECTION ABOVE. ROUTINE PARINIT IS A SUBSET OF UART_INIT
*   AND MAY BE CALLED INDEPENDENTLY.
*
*****************************************************************
*
UART_INIT  LDPK    0
*
        LALK    IRPACTIVE      ; INITIALIZE UART PARAMETER VARIABLES
        SACL    RPACTIVE
        LALK    IKM1
        SACL    KM1
        LALK    IK2M1
        SACL    K2M1
        LALK    INM1
        SACL    NM1
        LALK    IWORD_LEN
        SACL    WORD_LEN
        LACK    IODD
        SACL    ODD
        LALK    ITINTPER
        SACL    TINTPER
        LACK    1
*       SACL    TBE
        ZALS    TINTPER        ; INITIALIZE TIMER PERIOD
        SACL    2
        SACL    3
        LALK    0FFC8h         ; ENABLE TINT ONLY
        SACL    4
        ZAC
        SACL    RSTAT          ; RSTAT: = 0
        SACL    USTAT          ; USTAT: = 0
        OUT     USTAT,UARTPORT
        LAR     TWG, KM1       ; XMT WAIT: = 1/R
        LAR     XBITS_REG, WORD_LEN  ; INITIALIZE XBITS_REG
        ZAC
        SACL    DWORD          ; DWORD: = 0
        LRLK    INP_PTR, 0200h ; INITIALIZE DATA POINTERS
        LRLK    OPT_PTR, 0200h
*
*****************************************************************
*
*   CONFIGURE PARITY-RELATED CONSTANTS
*
*****************************************************************
```

```
*
PARINIT  ZALS    ODD
         SACL    RPAR
         SACL    XPARI
         LACK    1
         SACL    XPARTOG        ; INITIALIZE TOGGLE MASK TO BIT 0
         LALK    0FFFFh
         SACL    XSTOP          ; INITIALIZE STOP MASK TO BIT 0
         LARP    0
         LAR     0,NM1
         MAR     *+
SHIFX    LAC     XPARI,1        ; SHIFT PARITY BIT LOCATION BY N + 1 BITS
         SACL    XPARI
         LAC     XPARTOG,1      ; SHIFT TOGGLE MASK BY N + 1 BITS
         SACL    XPARTOG
         LAC     XSTOP,1        ; SHIFT STOP BIT STRING BY N + 1 BITS
         SACL    XSTOP
         BANZ    SHIFX
*
         LAC     XSTOP,1        ; TOTAL # OF SHIFTS FOR XSTOP = N + 2
         SACL    XSTOP
*
         EINT
         RET                    ; END UART INITIALIZATION
*
*************************************************************
*
*   END UART_INIT
*
*************************************************************
*
*
*   TRANSMITTER
*
*************************************************************
*
XMIT     ZALS    TBE
         BZ      NOT_EMPTY      ; IF NOT ZERO THEN EMPTY
*
EMPTY    ZALS    USTAT
         ANDK    TDA_MSK
         BZ      RCV            ; IF TDA = 0, THEN SKIP TO RECEIVER
*
         ZALS    TDATA          ; TSHF = TDATA
         SACL    TSHF
*
         ZALS    USTAT          ; CLEAR TDA AND TBE
         ANDK    TDA_CLR
         SACL    USTAT
         OUT     USTAT,UARTPORT
         ZAC
         SACL    TBE
*
NOT_EMPTY LARP   TWG            ; IF WAITING, SKIP TO RECEIVER
         BANZ    RCV
*
         ZALS    TSHF           ; OUTPUT NEXT BIT
         ROR
         BC      XONE
         NOP                    ; TIME COMPENSATION
         RXF
         B       XZERO
XONE     SXF
XZERO    SACL    TSHF
         LAR     TWG, KM1       ; WAIT 1 BIT INTERVAL BEFORE OUTPUTTING
*                               ;   NEXT BIT
*
         LARP    XBITS_REG      ; IF LAST BIT, SET TBE
         BANZ    RCV
*
         LACK    1              ; SET TBE
         SACL    TBE
*
         LAR     XBITS_REG, WORD_LEN
*
         CALL    XMT            ; SIGNAL END-OF-WORD
*
*************************************************************
*
*   END TRANSMITTER
*
*************************************************************
*
*
*   RECEIVER
*
*************************************************************
*
RCV      ZALS    USTAT          ; SKIP RECEIVE ROUTINE IF RDE = 0
         ANDK    RDE_MSK
         BNZ     CONT
         B       RETURN
*
CONT     ZALS    RSTAT          ; RSTAT = 0 IMPLIES WAITING FOR START BIT
         BNZ     NOTSTART
*
*************************************************************
*
*   RSTAT = 0
*
*************************************************************
*
         BIOZ    STARTBIT       ; LOOK FOR START BIT ON BIO/
         B       RETURN
*
STARTBIT LACK    1              ; UPDATE RSTAT
         SACL    RSTAT
         LAR     RWG, K2M1      ; WAIT 1/2 BAUD INTERVAL AFTER START BIT
```

```
        B  *        RETURN          ;  DETECTION
*
*****************************************************************
*
*    RSTAT <> 0
*
*****************************************************************
*
NOTSTART  LARP      RWG             ; IF WAITING, RETURN
          BANZ      RETURN
*
          ZALS      RSTAT           ; RSTAT = 1 INDICATES START BIT
          XORK      1               ;   VERIFICATION PENDING
          BNZ       NOTVER
*
*****************************************************************
*
*   RSTAT = 1
*
*****************************************************************
*
          BIOZ      VALID_START     ; IF INVALID START BIT, START OVER
          ZAC
          SACL      RSTAT
          B         RETURN
*
VALID_START  LACK   1               ; INITIALIZE BIT POSITION INDICATOR
          SACL      SHF
*
          LACK      2               ; UPDATE RSTAT
          SACL      RSTAT
*
          LAR       RBITS, NM1      ; N BITS/CHARACTER
          LAR       RWG, KM1        ; WAIT 1 BAUD INTERVAL BEFORE SAMPLING DATA
*
          B         RETURN
*
NOTVER    ZALS      RSTAT
          XORK      2
          BNZ       NOTDATA
*
*****************************************************************
*
*   RSTAT = 2
*
*****************************************************************
*
          ZAC                       ; PRESET DATA TO ZERO
          SACL      INP
          BIOZ      DAT_ZERO
*
          LACK      1               ; SET DATA TO 1 IF MARK
          SACL      INP
*
DAT_ZERO  LT        SHF             ; SHIFT INPUT BIT TO APPROPRIATE POSITION
          MPY       INP             ;   AND APPEND TO INPUT STRING
          PAC
          OR        DWORD
          SACL      DWORD
*
          ZALS      INP
          BZ        NOCHANGE        ; NO PARITY CHANGE IF SPACE
*
          ZALS      RPAR            ; TOGGLE PARITY-GEN BIT IF MARK
          XORK      1
          SACL      RPAR
*
NOCHANGE  LAC       SHF,1           ; UPDATE SHIFT FACTOR
          SACL      SHF
*
          LAR       RWG, KM1        ; WAIT 1 BAUD INTERVAL BEFORE SAMPLING NEXT
*                                   ;   BIT
*
          LARP      RBITS           ; END-OF-WORD DETECT
          BANZ      SKP3
*
          LACK      3               ; UPDATE RSTAT WHEN FULL WORD IS RECEIVED
          SACL      RSTAT
*
SKP3      B         RETURN
*
NOTDATA   ZALS      RSTAT           ; RSTAT = 3 IMPLIES WAITING FOR PARITY BIT
          XORK      3
          BNZ       NOTPAR
*
*****************************************************************
*
*   RSTAT = 3
*
*****************************************************************
*
          ZAC                       ; PRESET PARITY TO ZERO
          SACL      INP
*
          BIOZ      PZERO           ; SAMPLE PARITY BIT
          LACK      1
          SACL      INP
*
PZERO     ZALS      INP             ; CHECK AGAINST LOCALLY-GENREATED PARITY
          XOR       RPAR
*
          AND       RPACTIVE        ; IGNORE IF RECEIVE PARITY-CHECKING IS
          BZ        RCVPAR_OK       ;   DE-ACTIVATED
*
          ZALS      USTAT           ; REPORT PARITY ERROR
          ORK       RPE_SET
          SACL      USTAT
          OUT       USTAT,UARTPORT
```

```
*
RCVPAR_OK LAR     RWG,KM1       ; WAIT 1 BAUD INTERVAL
*
        LACK    4
        SACL    RSTAT
*
        B       RETURN
*
*****************************************************************
*
*   RSTAT = 4
*
*****************************************************************
*
NOTPAR  BIOZ    STOPO         ; INVALID STOP BIT
*
        B       STOP_OK
*
STOPO   ZALS    USTAT         ; REPORT FRAMING ERROR
        ORK     FRMERR_SET
        SACL    USTAT
        OUT     USTAT,UARTPORT
*
STOP_OK ZALS    USTAT         ; OVERRUN DETECT
        ANDK    RDA_MSK
        BZ      NO_OVERRUN
*
OVERRUN ZALS    USTAT         ; REPORT OVERRUN
        ORK     ROR_SET
        SACL    USTAT
        OUT     USTAT,UARTPORT
*
NO_OVERRUN ZALS DWORD         ; COPY DATA TO RDATA
        SACL    RDATA
*
        ZALS    USTAT         ; SET RDA
        ORK     RDA_SET
        SACL    USTAT
        OUT     USTAT,UARTPORT
*
        ZAC
        SACL    RSTAT
        SACL    DWORD
*
        ZALS    RPARI
        SACL    RPAR
*
        CALL    RECV
*
RETURN  EINT
        RET
*
*****************************************************************
*
*   END RECEIVER
*
*****************************************************************
*
*   TRANSMITTER DATA FEED ROUTINE
*
*   THE FOLLOWING XMT/RCV ROUTINES IMPLEMENT A LOOPBACK TEST.
*
*****************************************************************
*
XMT     RET
*
*****************************************************************
*
*   END XMT
*
*****************************************************************
*
*   RECEIVER DATA RECOVERY ROUTINE
*
*   THIS SET OF XMT/RCV ROUTINES IMPLEMENTS A LOOPBACK TEST.
*
*****************************************************************
*
RECV    CALL    GET_DATA      ; GET RECEIVED DATA
        CALL    XCOMPOSE      ; REFORMAT
        CALL    PUT_DATA      ; TRANSMIT SAME DATA
        RET
*
*****************************************************************
*
*   END RCV
*
*****************************************************************
*
*   TRANSMITTER INTERFACE ROUTINE
*
*   PUT_DATA COPIES <AR(OPT_PTR)> TO TDATA AND SET TDA
*
*****************************************************************
*
PUT_DATA LARP   OPT_PTR       ; COPY DATA TO TDATA
        ZALS    *
        SACL    TDATA
*
        ZALS    USTAT         ; SET AND REPORT TDA
        ORK     TDA_SET
        SACL    USTAT
        OUT     USTAT,UARTPORT
*
        RET
*
*****************************************************************
*
```

```
*    END PUT_DATA
*
*************************************************************************
*
*    RECEIVER INTERFACE ROUTINE
*
*    GET_DATA COPIES RDATA TO LOCATION POINTED AT BY AR(INP_PTR) AND CLEARS
*    RDA, ROR, FRM, AND RPE
*
*************************************************************************
*
GET_DATA  LARP     INP_PTR          ; COPY DATA
          ZALS     RDATA
          SACL     *
*
          ZALS     USTAT
          ANDK     RDA_CLR          ; CLEAR RDA
          ANDK     0FFF8h           ; CLEAR ERROR FLAGS
          SACL     USTAT
          OUT      USTAT,UARTPORT
*
          RET
*
*************************************************************************
*
*    END GET_DATA
*
*************************************************************************
*
*    TRANSMIT DATA COMPOSE ROUTINE
*
*    XCOMPOSE ADDS START, STOP, AND PARITY BITS TO DATA POINTED AT BY
*    AR(OPT_PTR)
*
*************************************************************************
*
XCOMPOSE  ZALS     XPARI            ; COMPOSE TRANSMIT WORD
          SACL     XPAR             ; INPUT IS <AR(OPT_PTR)>
*                                   ; OUTPUT IS <AR(OPT_PTR)>
*
          LARP     OPT_PTR
          ZALS     *
          SACL     TEMP
*
          LAR      XBREG, NM1       ; NUMBER OF TRANSMIT DATA BITS
          LARP     XBREG
*
C_PAR     ZALS     TEMP
          ROR                       ; DETERMINE PARITY BIT VALUE
          SACL     TEMP
          BNC      XNOCHANGE        ; NO PARITY CHANGE IF SPACE
*
          ZALS     XPAR             ; TOGGLE PARITY-GEN BIT IF MARK
          XOR      XPARTOG
          SACL     XPAR
*
XNOCHANGE BANZ     C_PAR
*
          LARP     OPT_PTR
          LAC      *,1              ; ADD START BIT
          OR       XPAR             ; ADD PARITY BIT
          OR       XSTOP            ; ADD STOP BITS
          SACL     *
*
          RET
*
*************************************************************************
*
*    END XCOMPOSE
*
*************************************************************************
*
```

# TMS320C17 and TMS320C010 Serial Interface

**Peter Robinson**

**Regional Technology Center — Waltham, Massachusetts**
**Texas Instruments**

# Introduction

Knowledge-based digital signal processing (DSP) systems, in which the processor learns its environment (adapts) and saves what it has learned in memory, have become increasingly desirable to designers. These systems can utilize any one of the many TMS320 DSPs. The TMS320 family is well-suited for high-speed realtime processing in knowledge-based control systems, where adaptation and retention of a learned environment is important.

These intelligent DSP systems need memory backup in order to save acquired knowledge in the event of a power failure. This report shows how the problems encountered with power failure can be remedied.

DSP systems also need a variety of real-world interactions. Host microcontrollers can facilitate these interactions by interfacing to real-world elements such as reading keyboards, UART interface, general-purpose system I/O, and power failure detection.

Data retention can be provided by battery backed-up RAM or by electrically-erasable programmable read-only memory (EEPROM). Since the present line of TMS320 does not have integrated on-chip EEPROM, its internal data RAM must be stored off-chip to guard against power failure. A typical host processor interface with a DSP uses a memory-mapped approach in which the DSP and microcontroller are defined as peripherals to one another and must be physically close, due to the extensive interconnections required.

This application report shows how to connect

- One serial port of the TMS320C17/E17 to a serial analog interface IC (in our example, a CODEC), and
- The other serial port of the TMS320C17/E17 to the serial peripheral interface module of the TMS370C010 (a member of TI's advanced EEPROM-based microcontroller family).

This system approach addresses the need for both battery back-up and host processor interface by presenting

1) A seven-wire serial communications interface between a TMS320C17/E17 and a TMS370C010 (the TMS370C010 data EEPROM provides the nonvolatile data storage for the TMS320C17/E17)
2) The required arbitration logic for the interface
3) A communications protocol
4) Associated TMS320/TMS370 assembly code to implement the transfer protocol

## System Configuration

Figure 1 shows a block diagram of the TMS320C17/E17 communicating with the TMS370C010 via a seven-wire serial communications interface. This configuration has the following features:

- The ability to transmit commands and/or system data to a remote processor or to receive them from the processor.
- The ability to interface to a CODEC or other serial A/Ds.

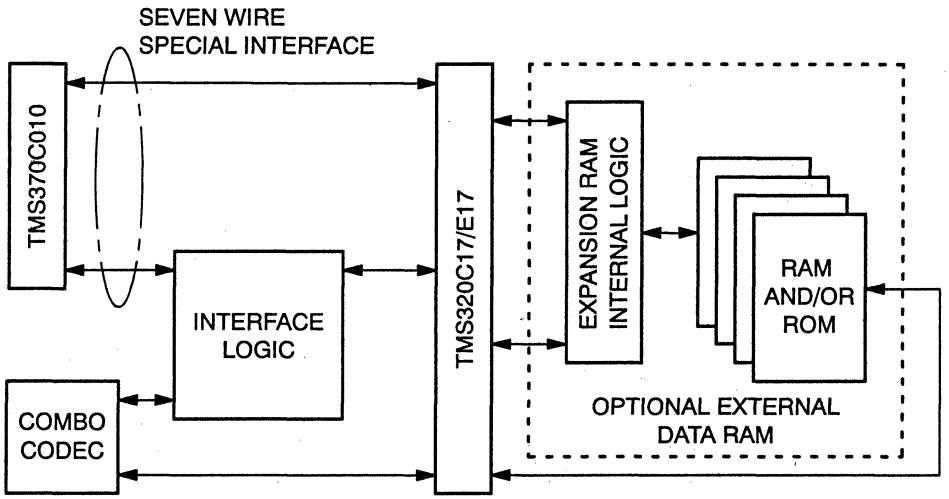



**Figure 1. Block Diagram of TMS320C17/E17/TMS370C010 System**

The TMS370C010 can, via the serial interface, provide the TMS320C17/E17 with nonvolatile storage of up to 256 bytes of memory. This can be 128 16-bit words of the TMS320C17/E17 memory and/or processor status.

You can use such an interface to implement the following DSP/microcontroller systems:

1) Voice pattern recognition/security access

2) Speech synthesis/recognition

3) Intelligent motion control

4) Intelligent vibration monitoring

5) Intelligent adaptive filter

6) Adaptive MODEM/FAX

## TMS320C17/E17 Overview

The TMS320C17/E17 is a low-cost stand-alone single-chip digital signal processor that combines the flexibility of a high-speed controller with the numerical capability of an array processor. It offers an inexpensive alternative to custom VLSI and multichip bit-slice processors. The TMS320C17/E17's powerful instruction set, high-speed number-crunching capabilities, and innovative architecture have made this high-performance, cost-effective processor the ideal solution to many telecommunication, computer, commercial, and industrial control applications. The TMS320C17/E17 employs a Harvard architecture (separate program and data busses) and highly pipelined data structure to obtain a 5-MIPS throughput (200 ns per instruction). The TMS320C17/E17's instruction set comprises data control, program control, and logical and digital signal processing instructions, making this DSP's code development process typical of most 16-bit microcontrollers.

The TMS320C17 and TMS320E17, shown in Figure 2, are dedicated microcomputers with 256 words of on-chip RAM and 4K words of on-chip ROM (TMS320C17) or EPROM (TMS320E17). The TMS320C17/E17 features a dual-channel serial interface, on-chip companding hardware ($\mu$-law/A-law), the TMS320C1X core processor, and a 16-bit latched I/O or coprocessor interface. The TMS320C17/E17 is also available in a 160-ns version capable of a 6.25-MIPS throughput.

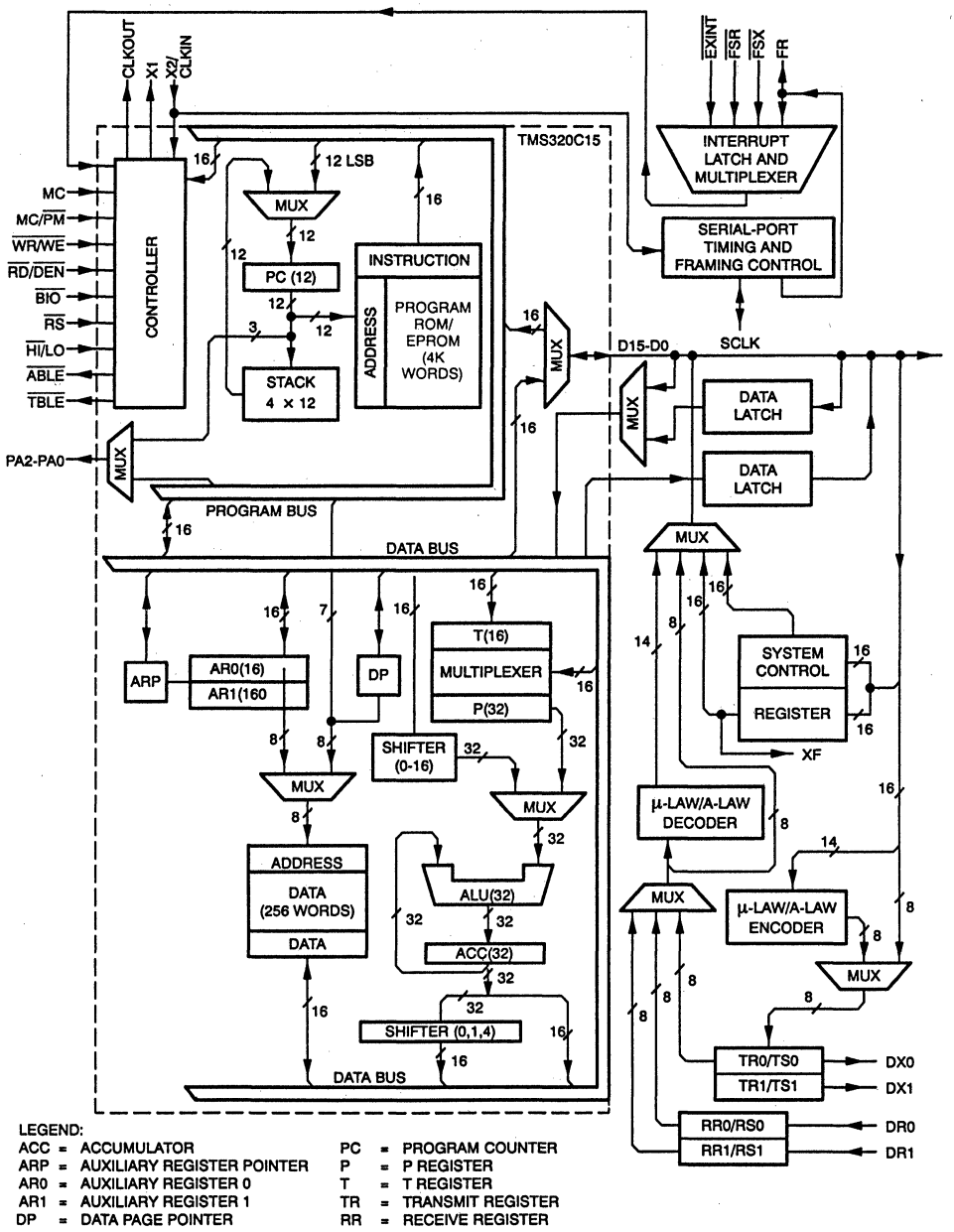
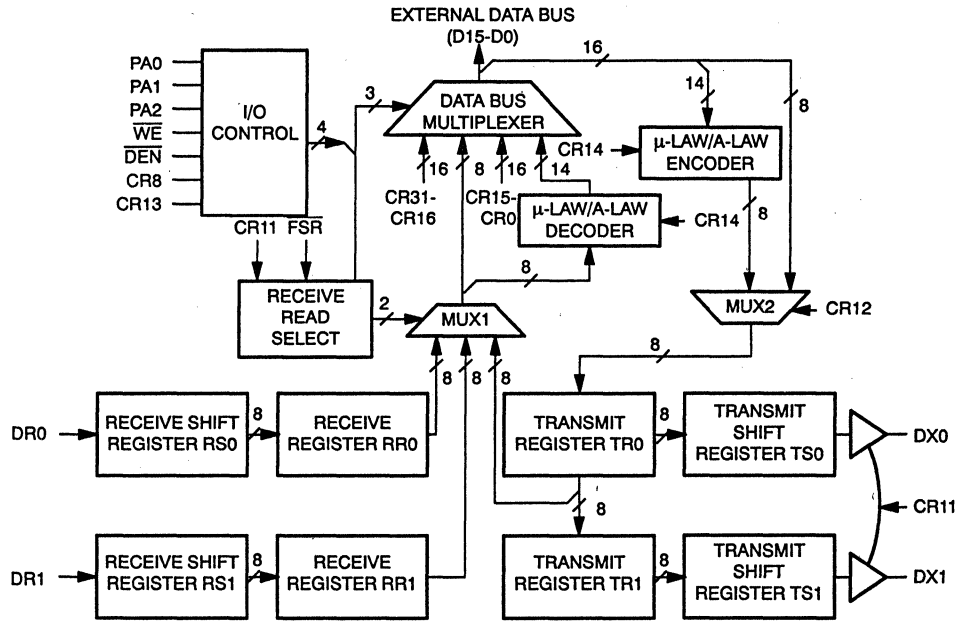



**Figure 2. TMS320C17/E17 Block Diagram**

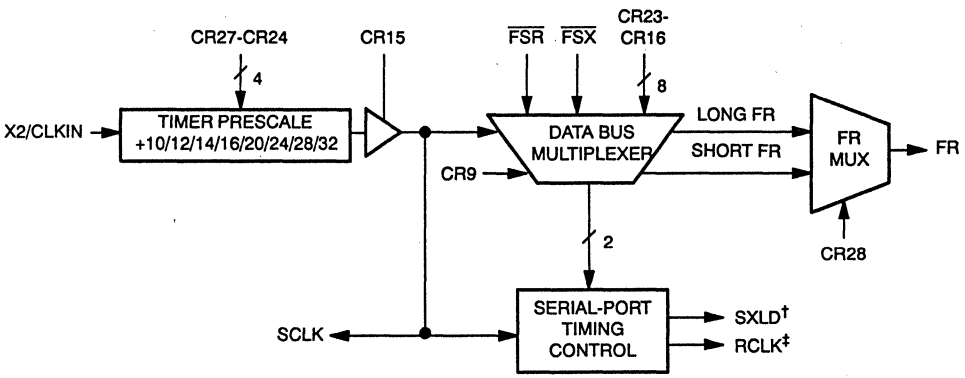
## TMS320C17/E17 Serial Port

Two of the memory-mapped I/O ports on the TMS320C17/E17 are dedicated to the serial ports and associated companding hardware (the $\mu$-law/A-law, decoder/encoder). I/O port 0 is dedicated to control register 0, which controls the serial port, interrupts, and companding hardware. I/O port 1 accesses control register 1, as well as the two 8-bit serial port channels and the companding hardware. The six remaining I/O ports are available for external 16-bit parallel interface.

The two serial ports support a frame-type serial transfer scheme that serves as a zero-glue logic interface to industry standard CODECs. Figure 3 shows the timing associated with the two serial ports. You can use the framing pulse of this interface to set the transfer time of the eight data bits. The transfer of data is accomplished by means of three signals: the transfer clock (SCLK), the transmit or receive pins (DX0/DX1 and DR0/DR1), and the transmit or receive frame pins (FSR and FSX). A fourth signal, frame request (FRF), is provided for arbitration.




Serial Port and Companding Hardware




†SXLD = Load transmit shift registers (TS0, TS1) from transmit registers (TR0, TR1)
‡RCLK = Load receive registers (RR0, RR1) from receive shift registers (RS0, RS1)

Serial Port Timing and Framing Control

**Figure 3. TMS320C17/E17 Serial Interface Logic**

The TMS320C17/E17 serial ports are provided with six interface lines:

| | |
|---|---|
| DX0/DX1 | Transmit shift register output pins |
| DR0/DR1 | Receive shift register input pins |
| SCLK | Serial port timing control pin (bi-directional) |
| FR | Frame request output pin |
| FSR | External frame receive |
| FSX | External frame transmit |

These six lines provide a full-duplex serial communications interface and direct interface to combo-CODECs (I.E.TMS29C1X series CODECs), PCM systems, serial A/D converters (such as the TLC3204X, TLC54X, and TLC154X), and most microcontrollers with a programmable serial interface.

## Interfacing a TMS320C17/E17 to a CODEC

The system configuration utilizes the serial ports of the TMS320C17/E17 for two distinct purposes:

- Interfacing to CODECs (which provide the analog interface for the system)
- Interfacing the DSP with the host microcontroller.

To clarify the two interface techniques, the simple CODEC interface is described first; then the serial communications interface of the TMS370C010 is discussed in detail.

As noted earlier, the TMS320C17/E17 is equipped with a hardware serial I/O module designed specifically to interface with two CODECs. The interface is straightforward and requires only that

- The transmit and receive lines of the TMS320C17/E17 and the CODEC be connected and that
- The frame request line (FR) be connected to the CODEC frame transmit and receive lines (FSX/FSR).

The timing clock is provided by the TMS320C17/E17's SCLK pin, which is connected to the CODEC's CLK/X pin.

The interface, shown in Figure 4, is described on page 6-7 in the *First-Generation TMS320 User's Guide* (literature number SPRU013B).

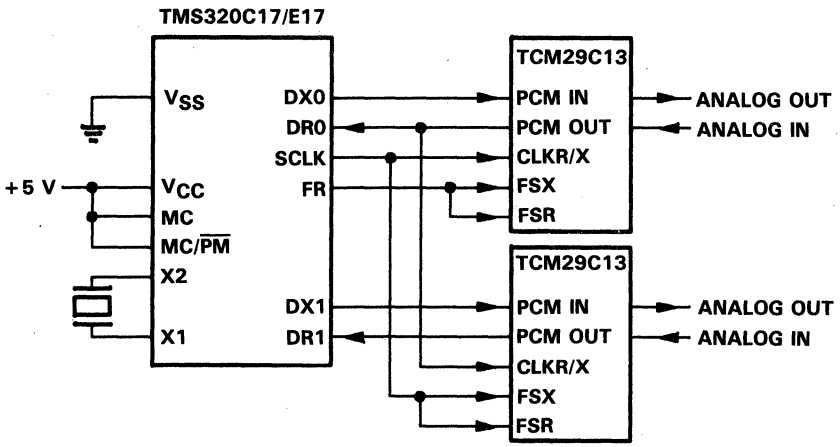



**Figure 4. Simple TMS320C17/E17/CODEC Interface**

## Interfacing the TMS320C17/E17 to the TMS370C010 SPI Module

To implement a bi-directional communications interface with the lines used by the protocol presented in this report, the connections shown in Figure 5 are required.

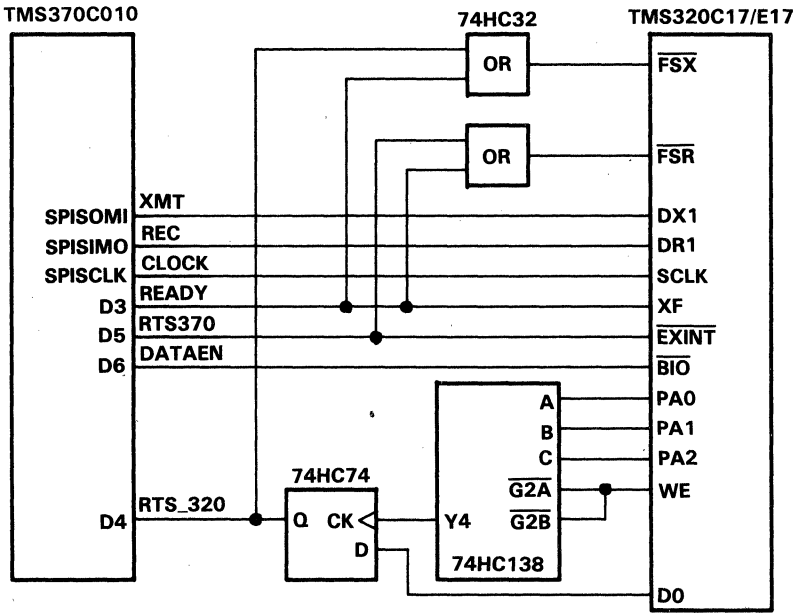



**Figure 5. Stand-Alone TMS320C17/E17/TMS370 Serial Interface**

This part of the design (showing only the serial peripheral interface, no CODEC interface) uses the transmit and receive pins of both the TMS320C17/E17 and TMS370C010 connected directly. To handle the frame timing (the TMS320C17/17 is slave to the TMS370C010), the following handshaking scheme is used.

1) When the TMS320C17/E17 wants to transmit to the TMS370C010, it writes to port 4, asserting a request to send signal RTS_320.

2) When the TMS370C010 is ready to receive the packet (packet is used to describe the 8-bit transfer), it asserts READY low, which, when ORed with the RTS_320 line, asserts FSX low and starts the transfer.

The TMS320's receive sequence is similar to the TMS370's as noted below.

1) The TMS370C010 asserts RTS_370 (TMS370C010 request to send line).

2) When the TMS320C17/E17 is ready to receive, it asserts the READY line with its XF output, which, via the OR gate, generates the FSR frame signal.

Assembly code examples in the appendices present a system flow showing exactly how this interface operates. It is recommended that Section 3.9 of the *First-Generation TMS320 User's Guide* (literature number SPRU013B) be reviewed to fully comprehend the workings of the TMS320C17/E17's serial ports.

## TMS370C010 Overview

The TMS370 family consists of several VLSI, 8-bit, CMOS microcontrollers with on-chip EEPROM storage and peripheral-support functions. These microcontrollers give superior performance in complex realtime control applications in demanding environments. Since TMS370 devices are available in mask-programmable read-only memory (ROM) or programmable EPROM and EEPROM, you have a significant range of processor options from which to choose.

This report explains how to interface the TMS370C010, shown in Figure 6, to the TMS320C17/E17 serial port by using the serial peripheral interface (SPI) module. It should be noted that other TMS370 devices can also be used because many of the TMS370 family members contain the SPI module.

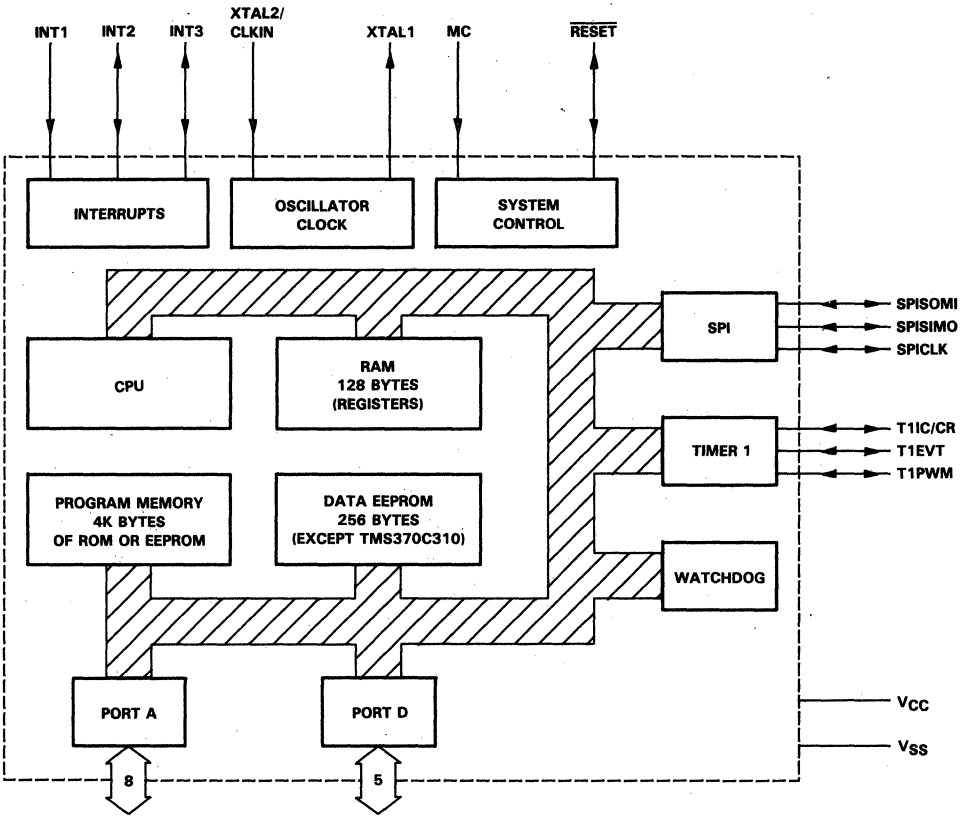



**Figure 6. TMS370C010 Block Diagram**

## TMS370C010 SPI Module

The SPI module, shown in Figure 7, is a high-speed user-configurable synchronous serial I/O unit that allows a serial bit stream of programmed length (8 bits in this case) to be shifted into and out of the device at a programmed bit-transfer rate. The SPI is normally used for communications between the microcontroller and an external peripheral (the TMS320C17/E17 in this case).

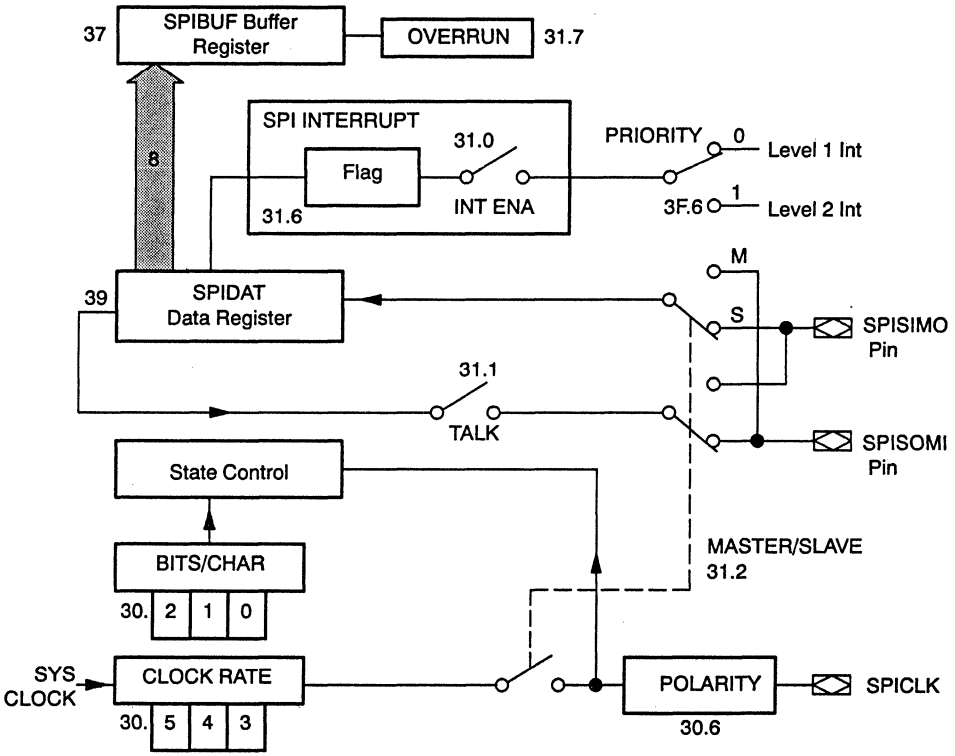



**Figure 7. TMS370C010 SPI Module**

The SPI module has a single 8-bit register used for both transmitting and receiving serial data. In this system, the serial data is clocked into the SPISOMI pin and clocked out of the SPISIMO pin. The SPICLK is used for counting and timing the data. Because the TMS370C010 is the **master** processor, the SPICLK is used by the TMS320C17/E17 for timing transmission and reception of all data transfers.

For more information on the TMS370C010's SPI module, refer to Section 10 of the *TMS370 Family Data Manual* (literature number SPNS014). There is also an application report for serial communication entitled *Using the TMS370 SPI and SCI Modules* (literature number SPNA006).

## TMS320C17/E17/TMS370C010 Transmission/Reception Protocol

The TMS320C17/E17 and the TMS370C010 are connected by the seven wires shown in Table 1.

**Table 1. Serial Interface Connections**

| TMS370C010 | TMS320C17/E17 | Name | Function |
|---|---|---|---|
| D3 | XF | READY | READY line |
| D4 | (D0@PA4) | RTS_320 | Request to send (TMS320C17/E17) |
| D5 | $\overline{\text{EXINT}}$ | RTS_370 | Request to send (TMS370C010) |
| D6 | $\overline{\text{BIO}}$ | DATA EN | Signals 320 – clk 9 sent |
| SPISCLK | SCLK | CLOCK | Data clock |
| SPISIMO | DR1 | REC | Data receive line |
| SPISOMI | DX1 | XMT | Data transmit line |

Sections 5.1 and 5.2 describe how the TMS320C17/E17 and TMS370C010 communicate over the serial interface. A timing diagram is illustrated for both transmission and reception of data, and signal sequencing steps (data flow and handshaking) are outlined for each case. In each example, the TMS370C010 is assumed to be the master of the system.

### TMS370C010 Transmits Data to the TMS320C17/E17

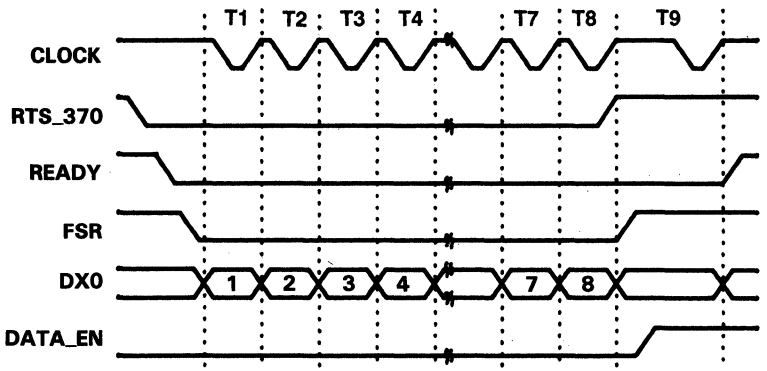



**Figure 8. Timing Diagram for TMS370C010 to TMS320C17/E17 Transmission**

The signal sequencing in Figure 8 is defined in the following steps:

1) The TMS370C010 interrupts the TMS320C17/E17 by asserting RTS 370 low.

2) When acknowledged, the TMS320C17/E17 ensures that the serial port is set up appropriately for receiving data, sets the SCLK for input clock, and kills any transmissions pending or in progress (in agreement with the system's characteristics of the TMS370C010 being master). The TMS320C17 then signals the TMS370C010 that it is ready for reception by bringing its XF pin (READY) low, which, in turn, sets the TMS320C17 FSR low (via an external OR gate). This notifies the onboard serial logic that data is soon to follow.

3) The TMS370C010 then transmits the 8 data bits. When the last bit is sent, the TMS370C010 sets the RTS 370 line high.

4) When the RTS 370 line goes high, the external OR gate asserts FSR high, causing a TMS320C17/E17 interrupt. The FSR interrupt indicates that the 8 data bits have been clocked into the receive shift register RS0. This puts the TMS320C17/E17 into a polling routine, waiting for the BIO (DATA EN) to go high.

5) The TMS370C010 sends a ninth clock pulse to transfer the 8-bit value in the RS0 register into the receive register, RR0.

   NOTE: This ninth clock pulse is required by the TMS320C17/E17 internal logic to transfer the contents of the RS0 to RR0. If the clock were free running, this ninth clock pulse would simply be the next clock pulse.

6) The BIO (DATA EN) is brought high by the TMS370C010 to signify that the ninth clock pulse has been sent and that the data is ready to be read by the TMS320C17/E17.

7) The TMS320C17/E17 reads the data, stores it, and sets an internal software flag, indicating a new data word has been received. The TMS320C17/E17 then ends the transmission by setting the XF (READY) line high. It also clears the interrupt and enables the EXINT interrupt for the next byte.

## TMS320C17/E17 Transmits Data to the TMS370C010

Figure 9 shows a timing diagram for transmission of data from the TMS320C17/E17 to the TMS370C010.

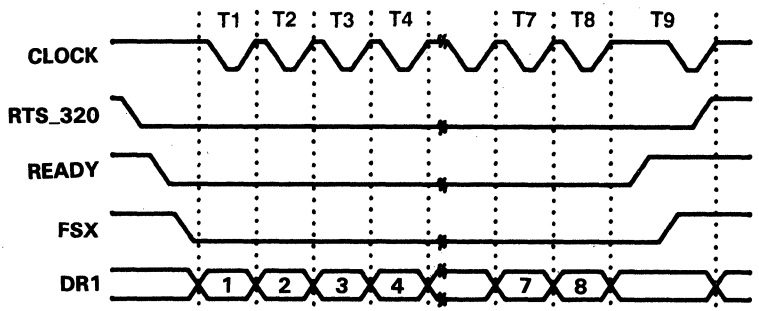



**Figure 9. Timing Diagram for TMS320C17/E17 to TMS370C010 Transmission**

Signal Sequencing in Figure 9 is defined in the following steps:

1) The TMS320C17/E17 puts the 8-bit value to be transmitted into the serial transmission register TR1.

2) By sending a zero data value OUT to I/O port 4, the TMS320C17/E17 signals to the TMS370C010 that its serial port is configured and that data is ready to be transmitted. The TMS370C010 sees this as a one-to-zero transition of the RTS__320 line (TMS320C17/E17 request to send).

3) The TMS370C010 sets READY low via the external OR gate, causing FSX to go low. The FSX transition causes the transfer of the 8-bit value in the TR0 register to the TS0 register, starting the transmission.

4) When the eighth data bit is received from the TMS320C17/E17, the TMS370C010 sets READY high, causing FSX to go high, which interrupts the TMS320C17/E17.

5) When the FSX interrupt is received from the TMS370C010, the TMS320C17/E17 transmission is assumed complete. The TMS320C17/E17 sends a one data value OUT to I/O port 4 (asserting data line D0 high), clearing the RTS__320 line. This restores the port to the initial state and makes it ready for the next RTS__370 (TMS370C010 request to send).

6) The last operation performed is the transmission of a ninth clock pulse by the TMS370C010. The ninth clock pulse is sent to reset the TMS320C17/E17 TR1 register to set up the TMS320C17 logic for the next transmission.

## Interfacing the TMS320C17/E17 to the TMS370 and CODEC Simultaneously

To allow the TMS320C17/E17 to communicate with a TMC29C13, the SCLK pin must be switched, under software control, between the SPICLK on the TMS370C010 and the CLK pin on the TMC29C13. The additional external logic needed to do this is shown in Figure 10. The logic consists of two AND gates (1/2 - SN74HC00) and three buffers (2/3 - SN74HC125).

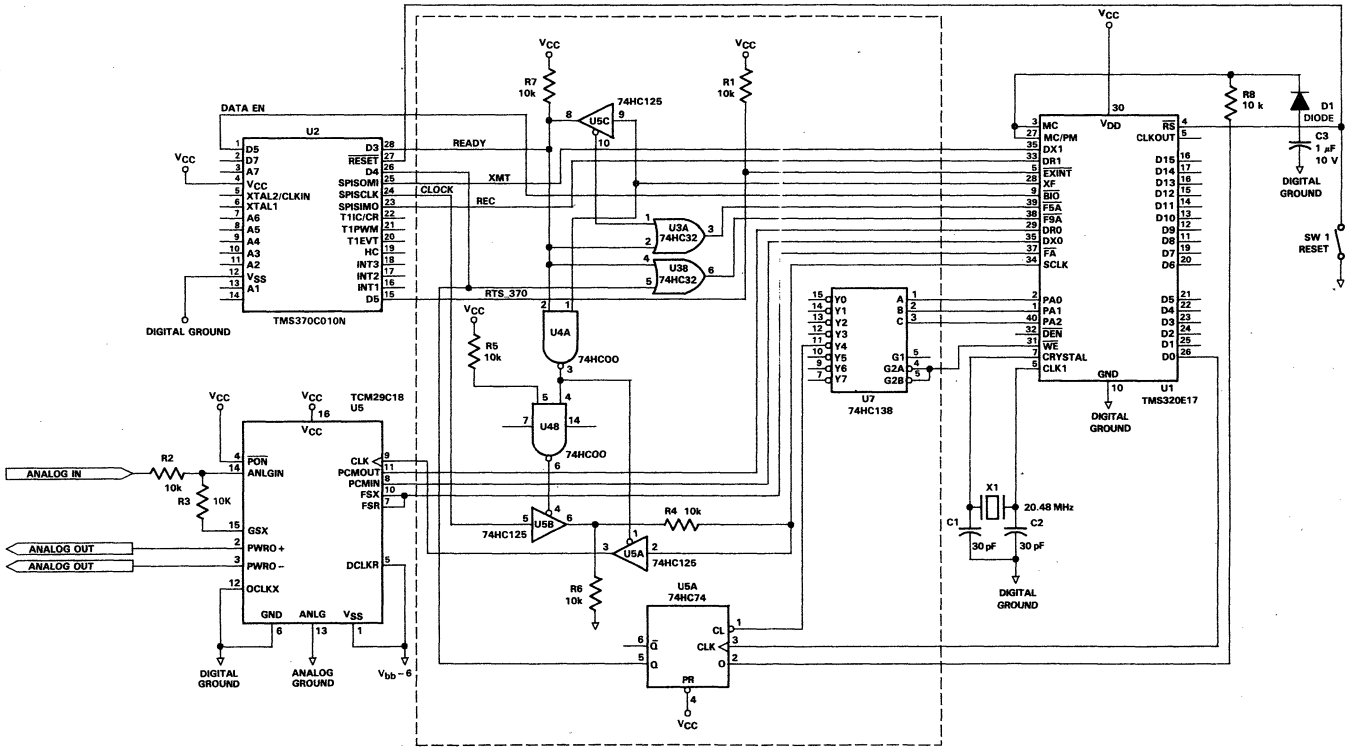



**Figure 10. Full System Schematic**

This additional logic adds to the system cost, board space, and power requirements. To reduce this burden, a TIBPAL16L8 can be used to absorb the AND, OR, buffer, and address decoder, reducing the system to the circuit shown in Figure 11.

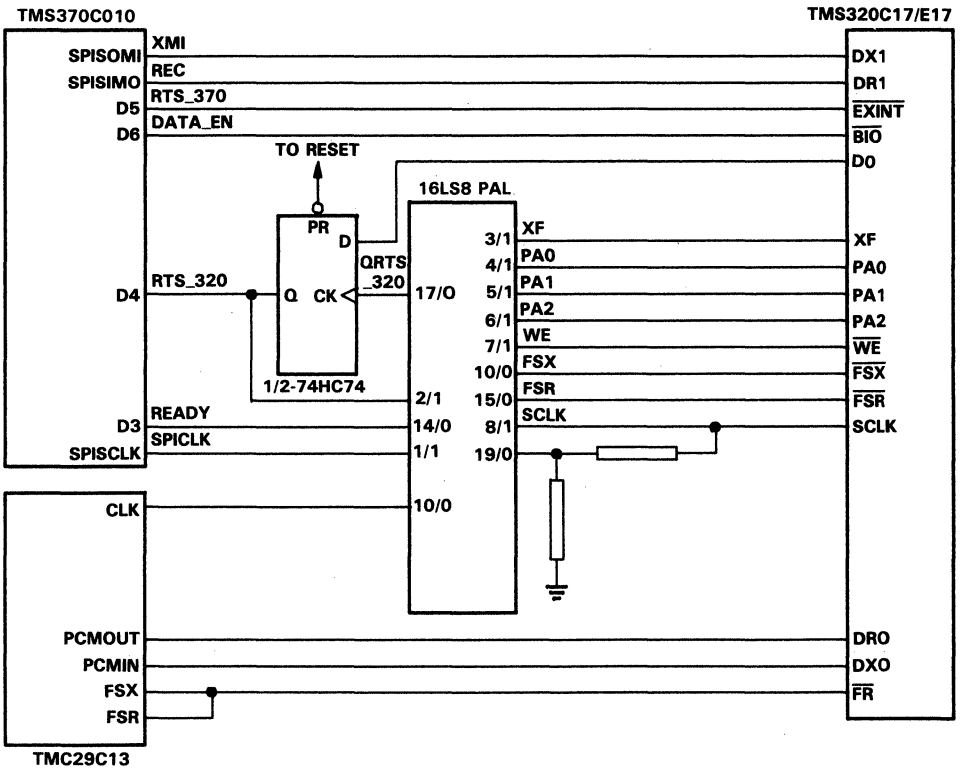



**Figure 11. Serial Interface with CODEC**

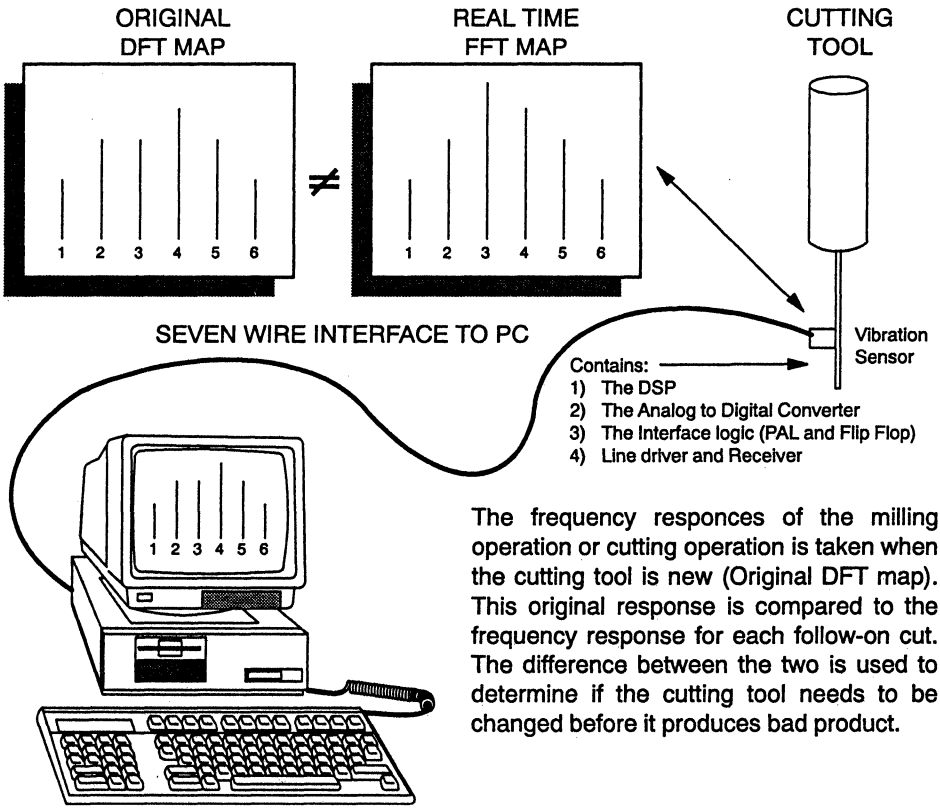
This brings the system chip count down to the following five ICs:

1. TMS370C010 - System microcontroller
2. TMS320C17/E17 - System digital signal processor (DSP)
3. TMC29C13 - CODEC (analog in and out)
4. 1/2 of an SN74HC74 - D flip-flop
5. TIBPAL16L8-15 - 15 ns PAL

## Design Example

An attractive feature of the approach shown in Figure 11 is the ability to isolate the DSP and CODEC from the microcontroller. The host controller can thus be placed 2 to 5 yards from the DSP/CODEC and connected via seven wires (READY, RTS__320, RTS__370, DATA EN, CLOCK, REC, and XMT). Using line drivers and receivers, this distance can be increased substantially.

An example of a system benefitting from the TMS370/TMS320 interface is a vibration monitoring device used to monitor an in-service automated numerical control milling machine (refer to Figure 12). In such a system, the DSP performs a 64-point real DFT, comparing its results to a DFT mask taken when the cutting tool in the milling machine was new. Such data can be used to predict when the cutting tool will go out of specification. A similar system could apply to almost any machine containing bearings or producing vibration relating back to the machines performance; i.e., copiers, automobiles, steam or jet engines, etc.




The frequency responces of the milling operation or cutting operation is taken when the cutting tool is new (Original DFT map). This original response is compared to the frequency response for each follow-on cut. The difference between the two is used to determine if the cutting tool needs to be changed before it produces bad product.

Within the PC is an interface board that has the TMS370C010 on it. The TMS370 holds the original map in EEPROM and compares it to each follow-on DFT map.

**Figure 12. Design Example**

## Conclusion

You can see that the TMS320 and TMS370, when paired together, provide a low-cost, high-performance DSP system ideally suited for adaptive DSP tasks requiring battery back-up. The TMS320C17/E17, with its serial interface logic, connects with zero-glue logic to combo-CODECs and, with the addition of only a PAL and one flip-flop, can communicate with the TMS370's SPI interface. The TMS370C010 is shown to be a powerful 8-bit microcontroller with onboard EEPROM. In the event of a power failure, the data EEPROM, in conjunction with the SPI serial peripheral interface, can be used as a means of preserving the TMS320C17/E17's and TMS370C010's data RAM and processor status. In addition, the TMS370C010 has the power and flexibility to read a keyboard, interface to a display, and/or communicate with a serial communication device (SCI/UART interface). Note that the Texas Instruments integrated circuits presented in this application report are offered in two surface-mount packages, thus giving a small-end system form factor.

## Source Code Examples

Source code examples are presented for both the TMS320 and TMS370 transmit and receive routines. The TMS370C010 code in Appendix A presents SPI initialization source modules, plus transmit and receive 8-bit values for the TMS320C17/E17. As noted earlier, the TMS370C010 is assumed to be the master while the TMS320C17/E17 is the slave. In the TMS320C17/E17 source code examples (Appendix B), the header presents a full narrative description, which closely follows the narrative presented within this report. Both the TMS320 and TMS370 source code examples are written in modular fashion so you can choose what you want to include to meet your unique system needs.

Appendix C gives the reduced equations and chip diagram for a PAL example.

```
*****************************************************************
*
*   APPENDIX A - TMS320C17/E17 SOURCE CODE EXAMPLE
*
*****************************************************************
*            COPYRIGHT (C) 1988, TEXAS INSTRUMENTS INCORPORATED,
*                        ALL RIGHTS RESERVED
*****************************************************************
*
*   SIO370
*
*   THIS SOFTWARE DEMONSTRATES THE USE OF THE TMS320C17 SERIAL INTERFACE TO
*   COMMUNICATE WITH A TMS370C010 CHIP; THE TMS370C010 IS ASSUMED TO BE THE
*   MASTER OF THE SYSTEM.
*
*   THIS DEMONSTRATION IS ACCOMPLISHED BY SIMPLY ECHOING TO THE TMS370C010
*   WHATEVER IS RECEIVED. THE SCENARIO IS AS FOLLOWS:
*
*   1. THE TMS370C010 INTERRUPTS THE TMS320C17, INDICATING A TMS370C010
*      REQUEST TO SEND.
*
*   2. THE TMS320C17 ENSURES THAT THE SERIAL PORT IS APPROPRIATELY SET UP FOR
*      THE TRANSMISSION, WITH EXTERNAL SCLK AND FRAMING. THE TMS320C17 THEN
*      SIGNIFIES IT IS READY FOR THE RECEPTION BY BRINGING THE XF PIN LOW. IT
*      ALSO KILLS ANY TRANSMISSIONS PENDING OR IN PROGRESS TO THE TMS370C010,
*      IN AGREEMENT WITH THE SYSTEM'S CHARACTERISTIC OF THE TMS370C010 BEING
*      MASTER.
*
*   3. THE TMS320C17 RECEIVES AN FSR INTERRUPT INDICATING THAT THE DATA HAS
*      BEEN CLOCKED INTO THE RECEIVE SHIFT REGISTER. A NINTH SCLK IS THEN
*      SENT FROM THE TMS370C010 TO TRANSFER THE DATA FROM THE SHIFT REGISTER
*      TO THE RECEIVE REGISTER, AND THE BIO PIN IS BROUGHT HIGH TO SIGNIFY
*      THAT THE DATA IS READY. THE TMS320C17 READS THE DATA, STORES IT, AND
*      SETS A FLAG INDICATING THAT A NEW DATA WORD HAS BEEN RECEIVED. THE
*      TMS320C17 THEN CLEARS THE INTERRUPT AND ENABLES THE NEXT TMS370C010
*      RTS INTERRUPT.
*
*   4. THE DUMMY APPLICATION POLLS THE DATA FLAG AND RESPONDS TO ITS BEING
*      SET. IT THEN ENSURES THAT THE SERIAL PORT IS SET UP FOR TRANSMISSION
*      AND SENDS THE DATA OUT TO THE SERIAL TRANSMISSION REGISTER. IT SIGNALS
*      THAT THE CONFIGURATION AND DATA ARE READY BY PERFORMING AN OUT
*      INSTRUCTION (A ZERO) TO I/O PORT 4, WHICH IS SEEN BY THE TMS370C010 AS
*      A TMS320C17 REQUEST TO SEND. FINALLY, IT ENABLES BOTH THE FSX AND
*      TMS370C010 RTS INTERRUPTS.
*
*   5. WHEN THE FSX (OR TMS370C010 RTS) INTERRUPTS ARE RECEIVED, THE TRANS-
*      MISSION IS ASSUMED COMPLETE. THE TMS320C17 SENDS A 1 OUT TO I/O PORT
*      4, CLEARING THE TMS320C17 REQUEST TO SEND, AND RESTORES THE PORT TO
*      THE INITIAL STATE, READY FOR THE NEXT TMS370C010 RTS.
*
*****************************************************************
        .page
        .title  'TMS370C17/TMS370XX COMMUNICATION SOFTWARE'
*
        .text
*
        B       SIO370          ; BRANCH TO PROCESSOR RESET
*
*****************************************************************
*
*   INTERRUPT HANDLER
*
*   THE INTERRUPT ENABLE BITS ARE AND'ED WITH THE INTERRUPT FLAG BITS TO
*   DETERMINE WHICH INTERRUPT TO SERVICE.
*
*****************************************************************
*
        .sect   code

        SST     SAVEST          ; SAVE STATUS
        LDPK    PAGE1           ; ENFORCE PAGE 1
        SACL    SAVEAL          ; SAVE ACCUMULATOR
        SACH    SAVEAH
        SAR     AR0,SAVAR0      ; SAVE AUXILIARY REGISTERS
        SAR     AR1,SAVAR1
*
        IN      SCRACH,CNFRG0   ; READ IN INTERRUPT FLAGS
*
        LAC     SCRACH,12       ; GET INTERRUPTS ENABLED
        OR      SCRACH          ; OR IN INTERRUPT FLAGS
        SACH    SCRACH          ; SAVE ENABLED BITS
        AND     SCRACH          ; MASK FLAGS WITH ENABLE BITS
        SACL    SCRACH          ; AND SAVE RESULT
*
        LACK    4               ; CHECK FOR RESPONSE SENT
        AND     SCRACH
        BNZ     RSPSNT
*
        LACK    2               ; CHECK FOR DATA RECEIVED
        AND     SCRACH
        BNZ     DATREC
*
        LACK    1               ; CHECK FOR 370 RTS
        AND     SCRACH
        BNZ     RTSREC
        .page
*****************************************************************
*
*   NO VALID INTERRUPT, RE-INITIALIZE INTERRUPTS
*
*****************************************************************
*
        LACK    08Eh            ; SCLK IN, NO CODEC CODING,
        SACL    SCRACH          ; XF HIGH, EXTERNAL FRAMING, SERIAL
*                               ;   PORT 1
```

```
        LACK    01Fh            ; ENABLE TMS370C010 RTS INTERRUPT,
        ADD     SCRACH,8        ; CLEAR ALL INTERRUPTS
        SACL    SCRACH
        B       INTRET          ; DEFAULT TO RETURN
*
*****************************************************************
*
*   INTERRUPT 1 - TMS370C010 REQUEST TO SEND RECEIVED
*
*   THE TMS370C010 HAS ISSUED A REQUEST TO SEND. ENSURE THAT THE SERIAL PORT
*   IS CONFIGURED CORRECTLY, KILL ANY PENDING TRANSMISSIONS OR TMS320C17
*   REQUESTS TO SEND, AND NOTIFY THE TMS370C010 OF READINESS BY SETTING THE
*   XF PIN LOW.
*
*****************************************************************
*
RTSREC  IN      SCRACH,CNFRGO   ; 370 RTS RECEIVED
        LACK    040h            ; CHECK IF FSX ENABLED,
        AND     SCRACH          ; EXPECTING TRANSMISSION
        BZ      NOFSX           ; BRANCH IF NOT
*
        OUT     ONE1,RTS320     ; CLEAR TMS320C17 RTS, KILL
*                               ; TRANSMISSION
*
NOFSX   LACK    08Ah            ; SCLK IN, NO CODEC CODING,
        SACL    SCRACH          ; XF LOW, EXTERNAL FRAMING
        LACK    025h            ; ENABLE FSR INTERRUPT,
        ADD     SCRACH,8        ; CLEAR FSX AND TMS370C010 RTS
*                               ; INTERRUPTS
        SACL    SCRACH          ; STORE NEW CONFIGURATIONS
*
        B       INTRET          ; BRANCH TO RETURN
        .page
*****************************************************************
*
*   INTERRUPT 2 - DATA RECEIVED
*
*   THE DATA HAS BEEN SHIFTED INTO THE RECEIVE SHIFT REGISTER FROM THE
*   TMS370C010, AND THE FSR LINE HAS GONE HIGH, BUT THE NINTH SCLK PULSE MAY
*   NOT HAVE ARRIVED TO TRANSFER THE DATA TO THE SECOND RECEIVE REGISTER.
*   WAIT FOR BIO TO GO HIGH, SIGNIFYING NINTH CLOCK HAS BEEN SENT AND DATA IS
*   READY.
*
*****************************************************************
*
DATREC  BIOZ    DATREC          ; WAIT FOR BIO TO GO HIGH AFTER NINTH
*                               ; CLOCK
*
        LDPK    PAGE0           ; GET DATA INTO PAGE 0
        IN      DATIN,SERIAL    ; READ DATA TWICE TO GET FROM SECOND
        IN      DATIN,SERIAL    ; RECEIVE REGISTER
*
        LACK    1               ; SET DATA FLAG
        SACL    DATFLG
        LDPK    PAGE1           ; NEW CONFIGURATION IN PAGE 1
        LACK    08Eh            ; SCLK IN, NO CODEC CODING,
        SACL    SCRACH          ; XF HIGH, EXTERNAL FRAMING
        LACK    012h            ; ENABLE TMS370C010 RTS INTERRUPT,
        ADD     SCRACH,8        ; CLEAR FSR INTERRUPT
        SACL    SCRACH          ; STORE NEW CONFIGURATION
*
        B       INTRET          ; BRANCH TO RETURN
*
*****************************************************************
*
*   INTERRUPT 4 - TRANSMISSION OF RESPONSE COMPLETE
*
*   THE DATA HAS BEEN SHIFTED OUT OF THE TRANSMIT SHIFT REGISTER TO THE
*   TMS370C010, AND THE TMS320C17 RETURNS TO WAITING FOR THE NEXT INPUT.
*
*****************************************************************
*
RSPSNT  OUT     ONE1,RTS320     ; CLEAR 320 RTS
*
        LACK    08Eh            ; SCLK IN, NO CODEC CODING,
        SACL    SCRACH          ; XF HIGH, EXTERNAL FRAMING
        LACK    014h            ; ENABLE TMS370C010 RTS INTERRUPT,
        ADD     SCRACH,8        ; CLEAR FSX INTERRUPT
        SACL    SCRACH          ; STORE NEW CONFIGURATION
        .page
*****************************************************************
*
*   FINISHED WITH THE INTERRUPT - RETURN
*
*****************************************************************
*
INTRET  LDPK    PAGE1           ; FORCE PAGE 1
        LAR     AR0,SAVAR0      ; RESTORE AUXILIARY REGISTERS
        LAR     AR1,SAVAR1
*
        ZALH    SAVEAH          ; RESTORE ACCUMULATOR
        ADDS    SAVEAL
        OUT     SCRACH,CNFRGO   ; OUTPUT NEW CONFIGURATION
        LST     SAVEST          ; RESTORE STATUS
        EINT                    ; INTERRUPTS
        RET                     ; RETURN
*
*****************************************************************
*
*   RESET PROCESSOR AND FALL THROUGH TO APPLICATION
*
*****************************************************************
*
SI0370  ZAC                     ; ACCUMULATOR
        LARP    AR0             ; SELECT AR0
        LARK    AR0,0FFh        ; INITIALIZE ADDRESS/COUNT
```

```
*
CLRMEM  SACL    *,0,AR0         ; LOOP TO CLEAR RAM
        BANZ    CLRMEM
*
        LDPK    PAGE1
        LACK    1               ; INITIALIZE ONE1 IN PAGE1
        SACL    ONE1
*
        LDPK    PAGE0
        SACL    ONE             ; AND ONE IN PAGE0
*
        OUT     ONE,RTS320      ; CLEAR TMS320C17 REQUEST TO SEND
*
        LACK    08Fh            ; SCLK IN, NO CODEC CODING,
        SACL    TEMP            ; XF HIGH, EXTERNAL FRAMING,
*                               ; UPPER CONTROL REGISTER
        LACK    0Fh             ; DISABLE ALL INTERRUPTS,
        ADD     TEMP,8          ; CLEAR ALL INTERRUPTS
        SACL    TEMP
        OUT     TEMP,CNFRG0
*
        LACK    0Ch              ; INITIALIZE PORT 1 TO 0CFEh
        SACL    TEMP
        LACK    0FEh
        ADD     TEMP,8
        SACL    TEMP
        OUT     TEMP,CNFRG1
*
        LACK    08Eh            ; SCLK IN, NO CODEC CODING,
        SACL    TEMP            ; XF HIGH, EXTERNAL FRAMING,
*                               ; SERIAL PORT 1
        LACK    01Fh            ; ENABLE 370 RTS INTERRUPT,
        ADD     TEMP,8          ; CLEAR ALL INTERRUPTS
        SACL    TEMP
        OUT     TEMP,CNFRG0
*
        EINT                    ; INTERRUPTS
*
*****************************************************************
*
*   APPLIC - DUMMY APPLICATION DRIVER
*
*   THIS ROUTINE POLLS THE DATA FLAG (DATFLG) TO DETERMINE THAT THE NEW DATA
*   HAS BEEN RECEIVED AND NEEDS TO BE RETURNED TO THE TMS370C010.
*
*****************************************************************
*
APPLIC  LAC     DATFLG          ; LOOP UNTIL DATFLG SET
        BZ      APPLIC
*
        ZAC                     ; DATFLG
        SACL    DATFLG
*
        LACK    08Eh            ; SCLK IN, NO CODEC CODING,
        SACL    TEMP            ; XF HIGH, EXTERNAL FRAMING
        LACK    054h            ; ENABLE FSX AND TMS370C010 RTS
        ADD     TEMP,8          ; INTERRUPTS, CLEAR FSX INTERRUPT
        SACL    TEMP
        OUT     TEMP,CNFRG0
*
        OUT     DATIN,SERIAL    ; SEND DATA OUT TWICE TO GET
        OUT     DATIN,SERIAL    ; TO SECOND TRANSMIT REGISTER
*
        OUT     ZERO,RTS320     ; ASSERT ACTIVE LOW TMS320C17 RTS
*
        B       APPLIC          ; LOOP BACK TO APPLIC
        .page
*****************************************************************
*   VARIABLE ASSIGNMENTS
*****************************************************************
* START  .BSS     SECTION AT DATA MEMORY ADDRESS ZERO ON PAGE ZERO
*
top     .BSS     ONE,1           ; CONSTANT 1*16
        .BSS     ZERO,1          ; CONSTANT 0*16
        .BSS     TEMP,1          ; SCRATCH LOCATION*16
        .BSS     DATIN,1         ; DATA IN FROM SERIAL PORT*16
bottom  .BSS     DATFLG,1        ; FLAG SET FOR DATA RECEIVED*16
length  .EQU     bottom-top
        .BSS     SPACE,80h-(length+1);.bss POINTER TO TOP OF PAGE ONE
*
* RESUME .BSS     SECTION AT DATA MEMORY ADDRESS 080h TOP OF PAGE ONE
*
        .BSS     ONE1,1          ; INTERRUPT CONSTANT 1*16
        .BSS     SAVEAL,1        ; INTERRUPT LOW ACCUMULATOR*16
        .BSS     SAVEAH,1        ; INTERRUPT HIGH ACCUMULATOR*16
        .BSS     SAVAR0,1        ; INTERRUPT AR0*16
        .BSS     SAVAR1,1        ; INTERRUPT AR1*16
        .BSS     SAVEST,1        ; INTERRUPT STATUS*16
        .BSS     SCRACH,1        ; INTERRUPT SCRATCH LOCATION*16
*
*****************************************************************
*
*   I/O PORTS
*
*****************************************************************
*
CNFRG0  .SET     0               ; CONFIGURATION REGISTER 0
CNFRG1  .SET     1               ; CONFIGURATION REGISTER 1
SERIAL  .SET     1               ; SERIAL I/O PORT
RTS320  .SET     4               ; TMS320C17 REQUEST TO SEND PORT
*
```

```
*************************************************************
*
*  DATA PAGE DEFINITIONS
*
*************************************************************
*
PAGE0   .SET    0               ; LOWER DATA MEMORY PAGE
PAGE1   .SET    1               ; UPPER DATA MEMORY PAGE
*
        .end
```

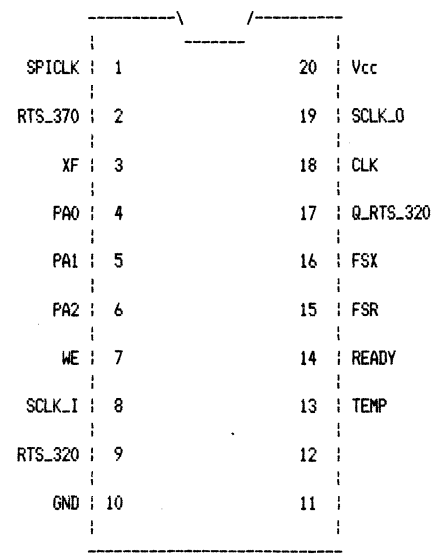
```
*****************************************************************
*
*   APPENDIX B - TMS370C010 SOURCE CODE EXAMPLE
*
*****************************************************************
*
*   THE FOLLOWING SOURCE CODE IS BROKEN INTO FOUR ROUTINES:
*
*   SPI MODULE INITIALIZATION ROUTINE (CALL AS SPINIT)
*   DIGITAL I/O PORT INITIALIZATION ROUTINE (CALL AS DINIT)
*   SEND DATA TO THE TMS320C17 ROUTINE (CALL AS TXD)
*   RECEIVE DATA FROM THE TMS320C17 ROUTINE (CALL AS RXD)
*
*****************************************************************
*
        .list
*
BIT0    .equ    01H             ; EQUATES FOR BIT TEST AND SET WITH OR
BIT1    .equ    02H
BIT2    .equ    04H
BIT3    .equ    08H
BIT4    .equ    10H
BIT5    .equ    20H
BIT6    .equ    40H
BIT7    .equ    80H
*
NBIT0   .equ    0FEH            ; EQUATES TO CLEAR A BIT WITH AND
NBIT1   .equ    0FDH
NBIT2   .equ    0FBH
NBIT3   .equ    0F7H
NBIT4   .equ    0EFH
NBIT5   .equ    0DFH
NBIT6   .equ    0BFH
NBIT7   .equ    07FH
*
*****************************************************************
*
*   VARIABLE ASSIGNMENTS IN BSS SECTION
*
*****************************************************************
*
        .bss    BUFFER,4        ; MEMORY STORAGE FOR PASSED VALUE
        .bss    POINTER1,1      ; DATA POINTER ONE
        .bss    POINTER2,1      ; DATA POINTER TWO
*
*****************************************************************
*
*   PERIPHERAL ASSIGNMENTS
*
*****************************************************************
*
DPORT1  .equ    P02C            ; ALTERNATE EXPANSION MODE SELECT FOR
*                               ; CONTROL BUS
DPORT2  .equ    P02D            ; EXPANSION MODE SELECT FOR CONTROL
*                               ; BUS
DDATA   .equ    P02E            ; I/O PORT OR EXPANSION CONTROL BUS
DDIR    .equ    P02F            ; I/O DPORT DIRECTION
INT1    .equ    P017            ; EXTERNAL INTERRUPT A CONTROL
*                               ; REGISTER
SPICCR  .equ    P030            ; SPI CONFIGURATION CONTROL REGISTER
SPICTL  .equ    P031            ; SPI OPERATION CONTROL REGISTER
SPIBUF  .equ    P037            ; RECEIVE DATA BUFFER REGISTER
SPIDAT  .equ    P039            ; SERIAL DATA REGISTER
SPIPC1  .equ    P03D            ; SPI PIN CONTROL 1
SPIPC2  .equ    P03E            ; SPI PIN CONTROL 2
SPIPRI  .equ    P03F            ; SPI PRIORITY CONTROL
*
*****************************************************************
*
*   RAM EQUATES
*
*****************************************************************
*
DATA    .equ    R4              ; DATA REGISTER TO SEND DATA TO THE
*                               ; TMS320
*
        .text
*
*****************************************************************
*
*   SPI INITIALIZATION
*
*****************************************************************
*
SPINIT
        MOV     #80h,SPICCR     ; RESET SPI
        OR      #00111111b,SPICCR ; SET BIT RATE TO CLKIN/1048, SET FOR
*                               ; 8 DATA BITS, SET SPI CLOCK POLARITY
*                               ; FOR INACTIVE LOW, AND CLEAR S/W
*                               ; RESET
        MOV     #00000100b,SPICTL ; SET AS MASTER, NO TRANSMISSION FOR
*                               ; NOW, AND DISABLE SPI INTERRUPTS
        MOV     #00000011b,SPIPC1 ; SET SPI CLOCK PIN AS OUTPUT CLOCK
        MOV     #00110010b,SPIPC2 ; SET DATA LINES AS SPI SOMI AND SIMO
        MOV     #01000000b,SPIPRI ; CLEAR SPI STEST BIT, SET SPI
*                               ; INTERRUPTS TO LEVEL 2, DISABLE
*                               ; EMULATOR SUSPEND BIT
        AND     #NBIT7,SPICCR   ; CLEAR S/W RESET BIT TO LOCK IN SPI
*                               ; CONFIGURATION
*
*****************************************************************
*
*   DIGITAL PORT INITIALIZATION
*
*****************************************************************
*
DINIT
        MOV     #00000000b,DPORT1 ; SET PORTB SELECT1 AND SELECT2 BITS
```

```
        MOV     #00000000b,DPORT2   ; TO DIGITAK I/O FUNCTION
        MOV     #01100100b,DDIR     ; SET PORTD AS FOLLOWS;
*                                   ;D3 = READY (OUTPUT)
*                                   ;D4 = RTS 320 (INPUT)
*                                   ;D5 = RTS 370 (OUTPUT)
*                                   ;D6 = DATA EN (OUTPUT)
*                                   ;D7 & D8 = UNDEFINED (INPUTS)
*
*****************************************************************
*
*   SEND DATA TO THE TMS320C17 ROUTINE
*
*****************************************************************
*
TXD     MOV     #00,A               ; ZERO A REGISTER
        MOV     A,BUFFER            ; CLEAR THE DATA BUFFER FOR EACH
        MOV     A,BUFFER-1          ; TRANSMIT TMS370 DATA TO THE
        MOV     A,BUFFER-2          ; TMS320C17
        MOV     A,BUFFER-3
        MOV     #32,A               ; 32 TO A REGISTER
        MOV     A,POINTER1          ; INITIALIZE POINTER 1 AND POINTER 2
        MOV     A,POINTER2          ; TO THE TOP OF THE DATA POINTER
        BTJO    BIT2,DDATA,TXD1     ; CHECK TO SEE IF A TMS320C17
*                                   ; TRANSMISSION IS IN PROGRESS, IF SO
*                                   ; GO AND RECEIVE DATA FROM THE
        JMP     TXD6                ; TMS320C17
TXD1    AND     #NBIT0,DDATA        ; PULL THE TMS370C010 RTS LINE LOW TO
*                                   ; START A TMS370C010 TRANSMIT CYCLE
TXD2    BTJO    #BIT1,DDATA,TXD2    ; WAIT FOR READY LINE TO GO LOW
        AND     #NBIT3,DDATA        ; SET THE DATA ENABLE LINE LOW
        MOV     DATA,SPIDAT         ; SEND DATA TO TMS320
TXD3    BTJZ    #BIT6,SPICTL,TXD3   ; WAIT FOR TRANSMISSION TO COMPLETE
        MOV     SPIBUF,A            ; READ THE SPIBUF REGISTER TO CLEAR
*                                   ; SPI INTERRUPT FLAG
        OR      #BIT0,DDATA         ; SET 370 RTS LINE HIGH TO END THIS
*                                   ; TRANSMISSION
        MOV     #00h,SPIDAT         ; ALLOW EXTRA CLOCKS TO LATCH DATA
*                                   ; INTO THE TMS320
TXD4    BTJZ    #BIT6,SPICTL,TXD4   ; WAIT FOR EXTRA CLOCKS TRANSMISSION
*                                   ; TO END
        OR      #BIT3,DDATA         ; SET THE DATA ENABLE LINE HIGH TO
*                                   ; SIGNAL THAT EXTRA CLOCKS HAVE BEEN
*                                   ; SENT
TXD5    BTJZ    #BIT1,DDATA,TXD5    ; WAIT FOR READY LINE TO GO HIGH
        MOV     SPIBUF,A            ; READ THE SPIBUF REGISTER TO CLEAR
*                                   ; SPI INTERRUPT FLAG
TXD6    AND     #NBIT7,INT1         ; CLEAR INT1 FLAG
        RTI                         ; RETURN FROM INTERRUPT SERVICE
*
*****************************************************************
*
*   RECEIVE DATA FROM THE TMS320C17 ROUTINE
*
*****************************************************************
*
RXD     OR      #BIT1,DDATA         ; PRESET B1 LINE HIGH FOR TMS320
*                                   ; TRANSMISSION
        OR      #BIT1,DDIR          ; CHANGE B1 TO AN OUTPUT READY LINE
*                                   ; FROM THE TMS370C010
        AND     #NBIT1,DDATA        ; SET THE READY LINE LOW TO ENABLE THE
*                                   ; TRANSMISSION CLOCK (SPICLK)
        MOV     #000h,SPIDAT        ; LOAD WITH DUMMY DATA INITIATE SPICLK
*                                   ; TO RECEIVE DATA FROM THE TMS320C17
RXD1    BTJZ    #BIT6,SPICTL,RXD1   ; WAIT FOR TRANSMISSION TO COMPLETE
        OR      #BIT1,DDATA         ; SET THE READY LINE HIGH TO INDICATE
*                                   ; THAT THE DATA HAS BEEN RECEIVED
        MOV     SPIBUF,A            ; READ THE TRANSMITTED DATA AND CLEAR
*                                   ; THE SPI INTERRUPT FLAG
RXD2    BTJZ    #BIT2,DDATA,RXD2    ; WAIT FOR THE RTS TMS320C17 LINE TO
*                                   ; GO HIGH TO INDICATE THE END OF THIS
*                                   ; TRANSMISSION CYCLE
        MOV     #000h,SPIDAT        ; LOAD WITH DUMMY DATA AGAIN TO SEND
*                                   ; MORE CLOCKS TO CLEAR THE TMS320C17
*                                   ; TRANSMIT BUFFER
RXD3    BTJZ    #BIT6,SPICTL,RXD3   ; WAIT FOR EXTRA CLOCKS TO BE SENT
        MOV     SPIBUF,B            ; DUMMY READ TO CLEAR SPI INTERRUPT
*                                   ; FLAG
RXD4    AND     #NBIT3,DDIR         ; CHANGE D3 READY LINE TO AN INPUT
        RTS                         ; RETURN FROM SUBROUTINE
*
        .end
```

# Appendix C. PAL Reduced Equations and Chip Diagram

```
Reduced Equations For Device U4

        SCLK_O = !(!SPICLK);

        enable SCLK_O = (!TEMP);

        CLK = !(!SCLK_I);

        enable CLK = (TEMP);

        Q_RTS_320 = !(PA0 & PA1 & !PA2 & !WE);

        FSX = !(!READY & !RTS_320);

        FSR = !(!RTS_370 & !XF);

        READY = !(!XF);

        enable READY = (!RTS_370);

        TEMP = !(!READY # !XF);
```

Chip diagram for Device U4



# Part III. Data Communications

# Theory an Implementation of a Splitband Modem Using the TMS32010

George Troullinos
Peter Ehlig
Raj Chirayil
Jon Bradley
Domingo Garcia

Digital Signal Processor Products — Semiconductor Group
Texas Instruments

# Introduction

With the predominant usage of computers and especially PCs, data communications are of increasing importance. Communication between the various computer systems and terminals is frequently accomplished by means of the Public Switched Telephone Network (PSTN). The essential element for this data communication is the modem, which interfaces computer systems and terminals with the telephone network.

In the past, modems have been traditionally implemented in the analog domain using discrete components. Recently, modem manufacturers have realized the flexibility and high performance offered by digital approaches. With the drastic reduction in the cost of digital signal processors, the power of Digital Signal Processing (DSP) becomes available for the implementation of medium-speed and high-speed modems.

This application report discusses the digital implementation of a modem using the TMS32010 Digital Signal Processor. Attention is focused on splitband modems, a class of modems that splits the bandwidth of the communications channel (telephone network) so that full-duplex operation can occur. The splitband technique is mainly used for implementing modems with data rates up to 2400 bps (bits per second). This report describes the theory and implementation of the Bell 212A/V.22 Recommendation, a 1200-bps splitband modem. Note that in the remainder of this report, the designations Bell 212A and V.22 are used synonymously to refer to the modem implemented. This report is not intended to provide a commercial product, but to introduce the implementation considerations and merits of digital signal processing-based approaches. Some of the protocol requirements for the Bell 212A/V.22 Recommendation are not implemented: the answer mode, the 300-bps Frequency Shift Keying (FSK) modem, and the notch filter required to reject the guard tone from the received signal.

Modems are sophisticated devices consisting of many functional blocks that must be correctly implemented. The interface of the functional blocks must also be appropriately adjusted for the overall structure to function properly. The different functional blocks can be implemented in many ways. For example, the receiver input bandpass filters can be recursive or nonrecursive, and different algorithms can be used for the carrier recovery and clock recovery. In addition to the possibility of implementing different algorithms, new algorithms may need to be added to the already existing structure, such as an adaptive equalizer or a second loop within the carrier recovery for the suppression of carrier-phase jitter. These considerations indicate the advantage of the microprocessor-based over the analog-based approach. Using the microprocessor approach, the designer can test different algorithms by simply modifying the software. Additional functional blocks can be included by simply adding new code. Therefore, high-performance modems can be implemented in a very short period of time.

The computational burden for digital modem implementation is very heavy. This implies the need of special features for the microprocessor to be used. The TMS32010, with its 200-ns cycle time, on-chip multiplier, and specialized instruction set is uniquely architected for digital signal processing. Because of this, the TMS32010 can implement the modem functional blocks using only a portion of its available processing power. Another major advantage of the microprocessor approach is the possibility of implementing variable-rate modems using the same hardware. Specifically, the same hardware used for the implementation of the Bell 212A/V.22 Recommendation can be used to implement 2400-bps splitband modems (CCITT V.22 bis Recommendation) by merely changing the software. Besides implementing various modems using the same hardware, additional functions can be included, such as speech store-and-forward and the Data Encryption Standard (DES)[1] for secure data communications.

This report is organized as follows: The first section, Modem Functional Blocks, is a description of the functional blocks required for implementation of the Bell 212A/V.22 Recommendation. The second section, Modem Hardware Description, is a brief discussion of the hardware used for the modem implementation. The functions implemented within the TMS32010 are described in detail in the third section, while the fourth section contains an overview of the functions implemented in the modem controller (the Texas Instruments TMS7742 microcomputer). The performance of the TMS32010-based modem is presented in the fifth section. Finally, the last section suggests alternative hardware configurations that can result in reduced system cost.

The prerequisites for understanding and gaining maximum benefit from this report are the level of a Bachelor's degree in Electrical Engineering and a basic understanding of Digital Signal Processing and Data Communications. Background material can be found in *Digital Signal Processing* (Chapters 1 through 7) by A.V. Oppenheim and R.W. Schafer; "Implementation of FIR/IIR Filters with the TMS32010/TMS32020," an application report in the book, *Digital Signal Processing Applications with the TMS320 Family*, offered by Texas Instruments; and in *Understanding Communications Systems* and *Understanding Data Communications*, books published by Texas Instruments.

## Modem Functional Blocks

A modem (MOdulator-DEModulator) is a device that modulates the baseband information at the transmitter, and demodulates the received signal to retrieve the baseband information at the receiver. The Bell 212A is a full-duplex modem with the receiver and transmitter sharing the available bandwidth of the communications channel. This type of modem is said to operate in either the originate or answer mode (see Figure 1). In the originate mode, it initiates the communication process, transmits with a carrier frequency of 1200 Hz, and receives at the frequency of 2400 Hz. At the other end of the communications channel is a modem that operates in the answer mode, i.e., receives at 1200 Hz and transmits at 2400 Hz.

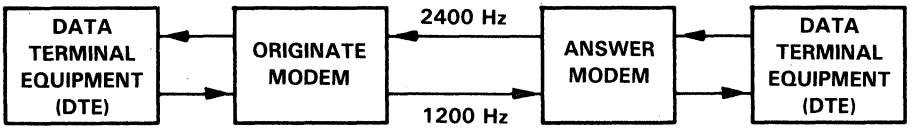



**Figure 1. Originate/Answer Configuration**

Table 1 shows the different functional blocks that comprize the modem transmitter and receiver.

**Table 1. Modem Functional Blocks**

| Modem Transmitter | Implemented |
|---|---|
| Guard Tone Generator | No |
| Scrambler | Yes |
| Encoder | Yes |
| Digital Lowpass Filters | Yes |
| Originate Mode Modulator | Yes |
| Answer Mode Modulator | No |
| **Modem Receiver** | **Implemented** |
| Notch Filter | No |
| Originate Mode Bandpass Filters | Yes |
| Answer Mode Bandpass Filters | No |
| Automatic Gain Control | Yes |
| Demodulator | Yes |
| Decision Block | Yes |
| Decoder | Yes |
| Descrambler | Yes |
| Clock Recovery | Yes |
| Carrier Recovery | Yes |

In the following two subsections, the operation of the modem transmitter and receiver are described. The transmitter accepts data (bits) from the Data Terminal Equipment (DTE). The DTE may be a dumb terminal, a PC, or a mainframe computer. The modem transmitter then performs the necessary processing to place this data in the proper form for transmission through the Public Switched Telephone Network. This processing basically consists of the modulation of the baseband information (logical ones and zeros (bits) sent by the DTE) into the passband of the communications channel for transmission. The receiver collects the information from the telephone network and transforms it back to its original form, i.e., the bits sent by the DTE.

## Modem Transmitter

The Bell 212A/V.22 is a 1200-bps modem that uses the Differential Phase Shift Keying (DPSK) modulation technique to transmit data through the communications channel. In the first part of this subsection, the equation that describes the operation of Differential Phase Shift Keying modulation systems is derived from an intuitive approach. A rigorous derivation is given in Appendix A. The rest of this subsection discusses the functional blocks required to correctly implement this equation.

In Differential Phase Shift Keying, the information is encoded as the phase change of the transmitter carrier. With $\phi(n)$ denoting the phase that contains the information to be transmitted, the transmitted signal s(n) is represented mathematically by

$$s(n) = A \cos(\omega n + \phi(n)) \tag{1}$$

where $\omega$ is the carrier frequency. The parameter A determines the amplitude of the transmitted signal. Use of the trigonometric identity

$$\cos(X+Y) = \cos(X)\cos(Y) - \sin(X)\sin(Y)$$

gives

$$s(n) = A \{\cos(\omega n)\cos[\phi(n)] - \sin(\omega n)\sin[\phi(n)]\} \tag{2}$$

The substitution of

$$I(n) = A \cos[\phi(n)]$$

$$Q(n) = -A \sin[\phi(n)]$$

into (2) results in (3) used to describe DPSK modulation systems.

$$s(n) = I(n)\cos(\omega n) + Q(n)\sin(\omega n) \tag{3}$$

From (3) it can be seen that the transmission of the baseband sequence {I(n),Q(n)} is accomplished by using two separate modulation carriers, a sine wave and a cosine wave. These waves are orthogonal; i.e., the information in the direction of the one wave (cosine) is independent of the information in the direction of the other wave (sine), and therefore this information is recoverable. Each value of the {I(n),Q(n)} sequence corresponds to one signaling element (symbol) transmitted. The number of signaling elements transmitted per second is commonly referred to as the baud rate, which for the Bell 212A/V.22 is set by the protocol to 600.

Some widely used terminology becomes apparent from (3). The baseband sequence that modulates the cosine wave is called the In-phase sequence. The baseband sequence that modulates the sine wave is called the Quadrature-phase sequence since the sine-wave carrier is 90 degrees (one Quadrant) out-of-phase from the cosine-wave carrier. The part of the transmitter/receiver that processes the In-phase sequence is commonly referred to as the I-channel, while the part of the transmitter/receiver that processes the Quadrature-phase sequence is referred to as the Q-channel.

The derivation of (3) indicates that the incoming sequence $d_S(n)$ is encoded into the sequence $\{I(n),Q(n)\}$, and the latter is transmitted. The mapping rule used is unique for each system; i.e., the mapping rule used for the Bell 212A/V.22 is different from the mapping rules used for other modems (V.22 bis, V.27, V.29, etc.). For example, for the Bell 212A/V.22, the sequence $\{I(n),Q(n)\}$ contains phase information, while for the V.22 bis, it contains phase and amplitude information. The set of possible values of the sequence $\{I(n),Q(n)\}$ determines the signal constellation, which is given in a two-dimensional representation.[2] The signal constellation, commonly referred to as the constellation diagram, is a geometric picture that emphasizes the fact that the two channels are 90 degrees (Quadrature) out-of-phase.

The Bell 212A/V.22, with a 600-baud rate, accomplishes the transmission of 1200 bps by encoding two incoming bits (dibit) in a single baud. Since there are four possible values for every dibit, the constellation diagram for the Bell 212A/V.22 contains four points. Each constellation point, i.e., each value of the $\{I(n),Q(n)\}$ sequence, corresponds to a total phase value to be transmitted. The calculation of the total phase from the incoming dibits will be discussed later. Figure 2 shows the constellation diagram for the Bell 212A/V.22. The four constellation points, notated A, B, C, and D, lie on a circle. Since there is no amplitude information transmitted, the radius of this circle is normalized to unity. The total phase information represented by each constellation point is enclosed in parentheses.

The encoding of the incoming sequence $d_S(n)$ into the values of the sequence $\{I(n),Q(n)\}$ is implemented by the encoder. The encoder is a one-input, two-output functional block, whose function is to map every two incoming bits (dibit) of the incoming sequence $d_S(n)$ to a total phase. The total phase is then represented by the values of the sequence $\{I(n),Q(n)\}$, and the latter is transmitted. The mapping rule used to encode the total phase into the values of the coder outputs $\{I(n),Q(n)\}$ is shown in Table 2. Each $\{I,Q\}$ entry in this table corresponds to one point in the constellation diagram of Figure 2. This is indicated in the third column of Table 2.

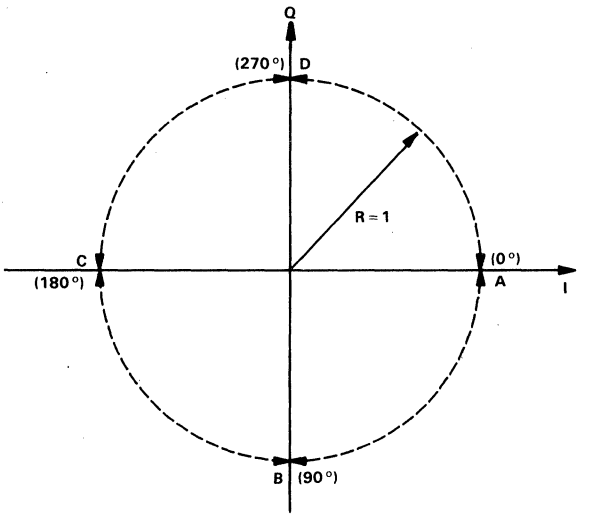



**Figure 2. Signal Constellation for the Bell 212A/V.22**

**Table 2. Total-Phase-to-Coder-Output Mapping Rule**

| Total Phase | Encoder Output { I , Q } | Point in Constellation Diagram of Figure 2 |
|---|---|---|
| 0 degrees | { 1 , 0 } | A |
| 90 degrees | { 0 , −1 } | B |
| 180 degress | { −1 , 0 } | C |
| 270 degrees | { 0 , 1 } | D |

The calculation of the total phase from the incoming dibits is accomplished in two steps. First, each incoming dibit is mapped to a unique phase change. Second, this phase change is added to the previous total phase to obtain the new total phase. The mapping rule used to uniquely map each dibit to a phase change is shown in Table 3.

**Table 3. Dibit-to-Phase Change Correspondence**

| Dibit | Phase Change |
|---|---|
| 00 | 90 degrees |
| 01 | 0 degrees |
| 10 | 180 degrees |
| 11 | 270 degrees |

To illustrate the two-step procedure used to calculate the total phase, consider the following example. The previous total phase is 90 degrees, and the incoming dibit is 10. From Table 3, the phase change corresponding to a 10 dibit is 180 degrees. Therefore, the new total phase is

new total phase = modulo 360 (previous total phase + phase change)

= (90 degrees) + (180 degrees) = 270 degrees

Using Table 2, for this value of total phase (270 degrees), the encoder output is {I,Q} = {0,1}. For the next incoming dibit, the above procedure is repeated with a 270-degree previous total phase.

At the receiver, the total phase is determined from the received {I,Q} value. This total phase is subtracted from the previous total phase (the one transmitted during the previous baud), and the difference is the phase change. Since the phase-change-to-dibit mapping is unique, using the calculated value of the phase change results in the transmitted dibit being uniquely recovered at the receiver.

This differential approach (i.e., the calculation of the phase change instead of an absolute phase) is used because if the dibits were to correspond to an absolute phase, then a common-phase reference for both the receiver and the transmitter would be required. This in turn implies the need of a training sequence between the transmitter and the receiver so that a common-phase reference can be established. This training sequence, however, is not provided for the Bell 212A/V.22.

An overall block diagram for the modem transmitter is shown in Figure 3.[3] The basic structural blocks are the scrambler, encoder, digital lowpass filter, and digital modulator.

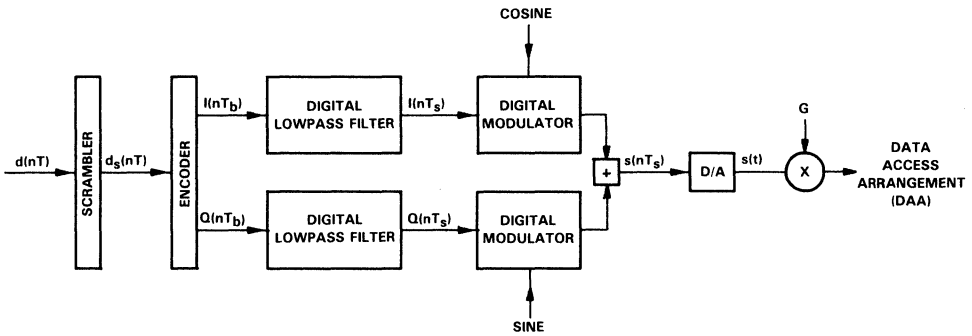



**Figure 3. Modem Transmitter Block Diagram**

### *Scrambler*

The scrambler scrambles the bits sent by the DTE. To understand the need for a scrambler, consider the situation where the DTE sends a series of 01 dibits. From Table 3, each 1 dibit corresponds to a 0-degree phase change. Therefore, the total phase transmitted is the same. From the geometrical point of view, this results in transmitting the same constellation point (same total phase). At the receiver end, however, phase changes are required for correct clock recovery (see the clock recovery discussion in the Modem Receiver subsection). Therefore, the transmission of a series of 01 dibits generates problems for the receiving modem, such as losing carrier lock. To avoid this, the scrambler is introduced to minimize the probability that such 'ill-conditioned' dibits occur. With d(nT) input to the scrambler, the output $d_S(nT)$ is given by

$$d_S(nT) = d(nT) \text{ XOR } d_S((n-14)T) \text{ XOR } d_S((n-17)T) \quad (4)$$

where XOR indicates the exclusive-OR operation and T is the data period, i.e., the time between two successive bits sent by the Data Terminal Equipment. The signal flowgraph of the modem transmitter scrambler is shown in Figure 4 in which $z^{-n}$ is used to indicated an n-sample delay.

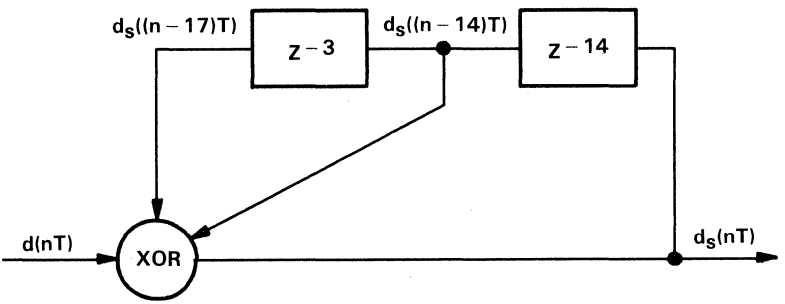



**Figure 4. Signal Flowgraph of Transmitter Scrambler**

### *Encoder*

The function of the encoder, i.e., the mapping of the incoming sequence $d_S(n)$ to the values of the sequence {I(n),Q(n)}, was discussed earlier. However, there is one more related issue associated with the encoder, i.e., the change of the sampling frequency at the encoder output. Every two bits that the modem transmitter accepts from the DTE correspond to a unique phase to be transmitted. Therefore, at the encoder output, the sampling period changes from T (sampling period of incoming data) to $T_b$, i.e., from 1/1200 s to 1/600 s. The subscript b in $T_b$ represents baud since the encoder output (I and Q channels) changes at the baud rate. The above discussion implies that $T_b = 2T$; i.e., the I and Q channels are updated after every pair of bits received from the DTE.

### *Digital Modulators and Lowpass Filters*

Since the telephone network behaves as a bandpass filter with the passband starting around 300 Hz and ending around 3200 Hz, the baseband encoder outputs, $I(nT_b)$ and $Q(nT_b)$, cannot be directly transmitted through the communications medium. They first must be modulated up in frequency. The modulation is not attempted directly on the encoder outputs for two reasons. First, as discussed at the end of this subsection, the sampling frequency must increase from $1/T_b$ to $1/T_s$, with $1/T_s$ being at least 6.4 kHz. This increase in the sampling frequency is accomplished by interpolation. Second, if the modulation is attempted directly on the encoder outputs, the instantaneous changes of the $I(nT_b)$ and $Q(nT_b)$ generate higher-order harmonics. Some of these harmonics fall in the frequency region reserved for the receiver. To eliminate the harmonics and to also increase the sampling frequency by interpolation, the encoder outputs must be digitally lowpass-filtered. The characteristics and the implementation of the digital lowpass filters are discussed in detail in the Transmit Filters subsection of "Functions Implemented in the TMS32010."

At the output of the lowpass filters, the I-channel modulates a cosine wave and the Q-channel a sine wave. The modulating frequency is 1200 Hz for an originate modem and 2400 Hz for a answer modem. Finally, the two channels are summed before they are transformed into the analog signal transmitted through the telephone network. The output of the digital transmitter (before the D/A converter) is given by equation (3), repeated below for convenience.

$$s(nT_s) = I(nT_s)\cos(\omega nT_s) + Q(nT_s)\sin(\omega nT_s)$$

The sampling period $T_s$ is $T_s = 1/f_s$ where $f_s$ is the sampling frequency. This frequency must be at least twice the highest frequency component of the transmitted information (Nyquist rate) to satisfy the sampling theorem. Since the telephone network cuts off at approximately 3.2 kHz, the sampling frequency must be at least 6.4 kHz. Practical considerations (integer number of samples per baud, etc.) impose the necessity of higher sampling rates. For the present application, the sampling frequency used was 9.6 kHz. Since the baud frequency is 600 Hz, 16 (9600/600) samples correspond to each baud interval.

## Modem Receiver

This subsection discusses the issues associated with the functional blocks required to implement a Bell 212A/V.22 modem receiver. The receiver structure is more sophisticated than that of the transmitter. For a low bit-error-rate performance (percentage of error bits received), an Automatic Gain Control (AGC) subsystem, adaptive equalization of the overall transmitting system, and a noise-independent carrier recovery and clock recovery are required. Since the issues associated with the carrier recovery and the clock recovery are critical in a modem design and difficult to understand, a good portion of this subsection is devoted to their discussion.

The adaptive equalizer is an adaptive filter that compensates for intersymbol interference and Doppler spread effects introduced during transmission over the telephone lines.[4] The magnitude of these effects depends on the bit rate and the quality of the telephone line. The effects are more severe at high bit rates (2400 bps and above) and over a worst-case telephone line, which is commonly represented by the 3002 line simulator.[5] The Bell 212A/V.22 protocol does not require the presence of an adaptive equalizer; therefore, this implementation does not include one. However, for increased performance on a 3002 line where even at medium speeds, such as 1200 bps, intersymbol interference and Doppler spread effects become severe, an adaptive equalizer is recommended. An important point here is that the addition of an adaptive equalizer in the current TMS32010 implementation of the Bell 212A/V.22 modem does not require any hardware changes. Increased performance results from an increase in the algorithmic sophistication.

An overall block diagram of the modem receiver is shown in Figure 5. The basic structural blocks of the modem receiver are the input bandpass filters, the automatic gain control (AGC), the demodulator, the decision block, the decoder, the descrambler, the carrier recovery, and the clock recovery.

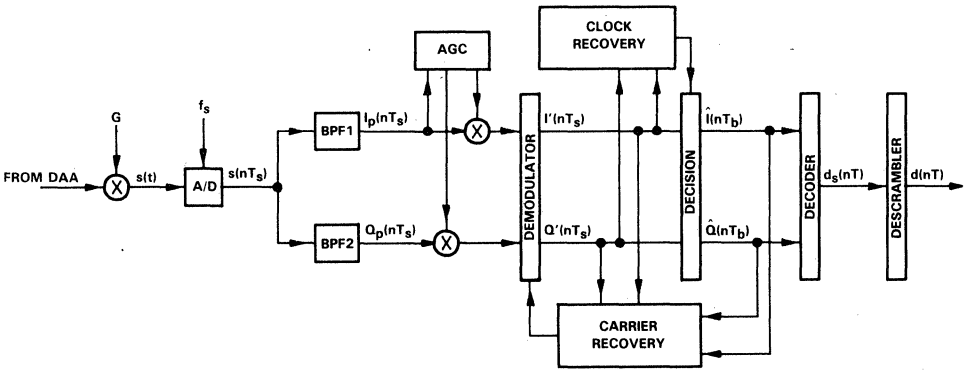



**Figure 5. Modem Receiver Block Diagram**

### *Input Bandpass Filters*

The incoming analog signal s(t) is digitized at the sampling frequency $f_S$ to obtain its digital counterpart $s(nT_S)$. This signal is then bandpass-filtered for three reasons:

1. Rejection of out-of-band noise, including the rejection of the transmit signal spectrum due to the near-end echo path,
2. Introduction of 90-degree relative phase shift required for the I and Q channel separation (see Appendix A), and
3. Fixed equalization for line distortion.

The second reason mentioned above implies the need of receiver bandpass filters that achieve a 90-degree relative phase shift. It is theoretically justified in Appendix B that if the two bandpass filters, denoted by BPF 1 and BPF 2 in Figure 5, achieve an exact 90-degree relative phase shift, there are no harmonics at the output of the receiver demodulator. If this condition is not met, harmonics appear at twice the carrier frequency. These harmonics were observed in the modem implementation when a set of bandpass filters not meeting the above condition was used. Elimination of the harmonics due to an inexact 90-degree relative phase shift involves the use of lowpass filters at the output of the demodulator. However, the group delay and the possible phase distortion introduced by the lowpass filters affect the carrier recovery and decision algorithms. Compensation for these side-effects of the lowpass filters results in a more complicated modem receiver design.

In the analog domain, where component drift is due to aging and/or temperature, it is virtually impossible to design bandpass filters or Hilbert transformers that achieve an exact 90-degree relative phase shift. Hilbert transformers, a special class of filters, are discussed in Appendices A and B. In the digital domain, however, the design of bandpass filters or Hilbert transformers that achieve an exact 90-degree relative phase shift is relatively easy. Digital filter design packages, such as the Digital Filter Design Package (DFDP) offered by the Atlanta Signal Processors Incorporated (ASPI), can be used to design modem receiver input filters on the TMS32010 that meet the exact 90-degree relative phase shift requirement. The characteristics and implementation of the modem receiver input bandpass filters are discussed in detail in the Receive Filters subsection of "Functions Implemented in the TMS32010."

### *Automatic Gain Control (AGC)*

Because of the attenuation introduced by the telephone lines, the peak-to-peak voltage of the incoming analog signal s(t) ranges between 2 mV and 700 mV. However, signal levels in the receiver must be independent of the attenuation introduced by the communications channel. This is of paramount importance because the carrier recovery and clock recovery algorithms use error signals and thresholds dependent on the I and Q channel values. Therefore, the Automatic Gain Control subsystem is required to adjust the envelope of the I and Q channels so that they are of the same magnitude. The AGC algorithm used and its implementation is discussed in the Automatic Gain Control Implementation subsection of "Functions Implemented in the TMS32010."

### *Demodulator*

The demodulator translates the passband information back to the baseband. With $I_p(nT_S)$ and $Q_p(nT_S)$ inputs to the demodulator (see Figure 5), the outputs $I'(nT_S)$ and $Q'(nT_S)$ are given by (see derivation in Appendix A)

$$I'(nT_S) = I_p(nT_S)\cos(\omega' nT_S) + Q_p(nT_S)\sin(\omega' nT_S) \quad (5)$$

$$Q'(nT_S) = I_p(nT_S)\sin(\omega' nT_S) - Q_p(nT_S)\cos(\omega' nT_S) \quad (6)$$

where $\omega'$ is the local carrier frequency. Figure 6 shows the demodulator structure that implements (5) and (6).

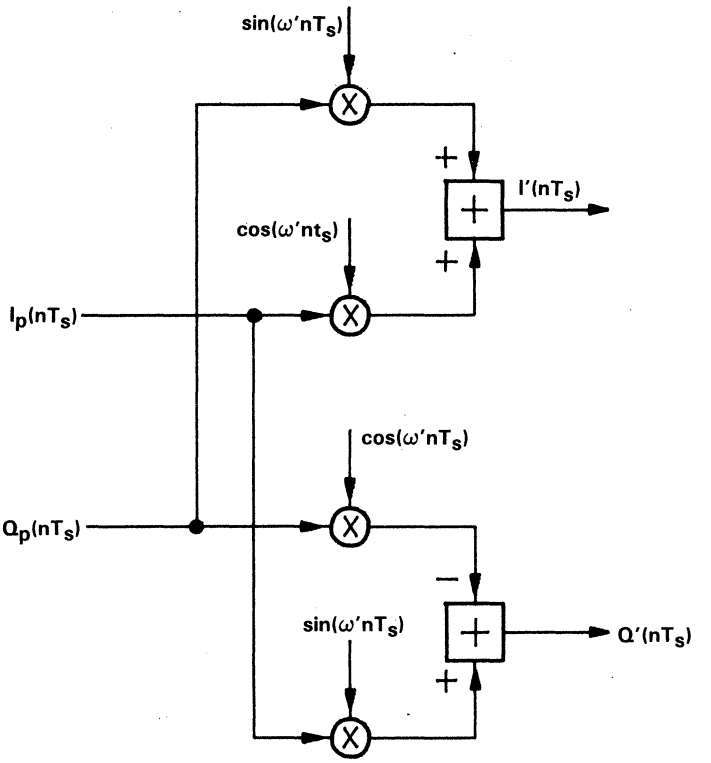



**Figure 6. Demodulator Structure**

Even with an ideal receiver, the $I'(nT_S)$ and $Q'(nT_S)$ channels shown in Figure 6 are 'noisy' replicas of the baseband I and Q channels at the output of the transmitter digital lowpass filters. The 'noise' has been injected by the telephone network as group delay, frequency jitter, and Gaussian noise.[4]

### *Decision Block and Descrambler*

The decision block in Figure 5 calculates the total phase from the values of the baseband I and Q channels. By subtracting it from the previous total phase (the phase transmitted during the previous baud interval), the phase change is computed. Each phase change (total of four) has a corresponding unique dibit (see Modem Transmitter subsection). This dibit is fed into the descrambler (see Figure 5) to recover the originally transmitted dibit. The output of the descrambler is described by

$$d(nT) = d_S(nT) \text{ XOR } d_S((n-14)T) \text{ XOR } d_S((n-17)T) \quad (7)$$

where T is the data period (1/1200 s for the Bell 212A). The signal flowgraph of the receiver descrambler is shown in Figure 7.

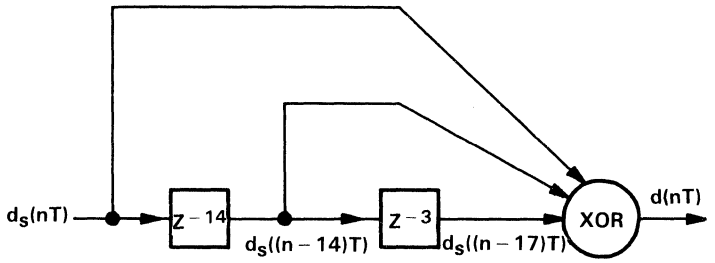



**Figure 7. Signal Flowgraph of Receiver Descrambler**

*Carrier Recovery*

A very important task of the modem receiver is the generation of a carrier that has the same frequency and phase with the incoming carrier. This receiver-generated carrier, called the local carrier, is used by the demodulator of Figure 6 to demodulate the incoming signal and therefore retrieve the baseband information. The process of generating this carrier is called carrier recovery. The standard approach to this is to use a phase-locked loop.[6] Figure 8 shows the basic blocks of a phase-locked loop: the phase detector (PD), loop filter and Voltage Controlled Oscillator (VCO).

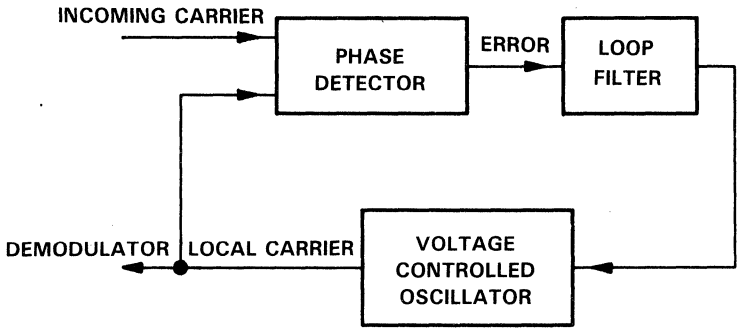



**Figure 8. Carrier Recovery Phase-Locked Loop**

For a microprocessor implementation, the blocks in Figure 6 are implemented digitally. The digital implementation is discussed in the Carrier Recovery Implementation subsection of "Functions Implemented in the TMS32010." Only the issues associated with the carrier recovery phase-locked loop are considered here.

The phase detector (PD) generates an error signal that is used to synchronize the local carrier to the incoming carrier. This error signal must contain the information about the phase and frequency difference between the local and the incoming carriers. To implement the correct carrier recovery algorithm, it is critical to know the exact dependence of the phase detector output on the frequency and phase difference between the two carriers (discussed later in this subsection). The phase detector output is of the form[7,8]

$$E(nT_b) = \hat{Q}(nT_b)\, I'(nT_b) - \hat{I}(nT_b)\, Q'(nT_b) \tag{8}$$

where $\hat{I}$ and $\hat{Q}$ are the I and Q channel decisions and $T_b$ is the baud period (1/600 s). If the decisions are correct, then

$$\hat{Q}(nT_b) = Q(nT_b) \tag{9}$$

$$\hat{I}(nT_b) = I(nT_b) \tag{10}$$

i.e., the outputs of the transmitter coder (see Figure 3) have been successfully recovered. The probability that these decisions are correct is maximum in the middle of each baud because the incoming signal energy is maximum here. Based on the error signal $E(nT_b)$, the local carrier is corrected once every baud, i.e., at a 600-Hz frequency. Geometrically, the error $E(nT_b)$ is a measure of the geometrical distance between the point used to make the decision and the optimum one. The optimum decision points are the constellation points. It is shown later that when the local carrier has the same phase and frequency with the incoming carrier, the error $E(nT_b) = 0$. In this case, the point used to make the decision coincides with a constellation point. The optimality of the receiver constellation points is discussed next.

Optimality in the receiver, in terms of low probability of error, is determined only by the geometrical distance between the constellation points.[9] The four constellation points of Figure 2, notated as A, B, C, and D, are optimum. The following intuitive argument helps to illustrate this. The four points lie on a circle of normalized unity radius. In the configuration of Figure 2, point A is equidistant from points B and D. This means that the probability of error p when deciding between points A or B, i.e., deciding point A when point B is correct and vice versa, is equal to the probability of error when deciding between points A or D. If point A moves counterclockwise, it moves away from point B but closer to point D. Since at the new location, point A is farther from point B, the probability of error $p_1$ when deciding between points A or B decreases, i.e., $p_1 < p$. However, at this new location, point A is closer to point D, and therefore, the probability of error $p_2$ when deciding between points A or D increases, i.e., $p_2 > p$. Using the analytical tools discussed in [9], it can be shown that $p_1 + p_2 > 2p$. Since the overall probability of error increases ($p_1 + p_2 > 2p$) if point A moves away from the location indicated in Figure 2, the resulting structure is no longer optimum. This is not true, however, if all four constellation points are equally rotated by an arbitrary amount in the clockwise or counterclockwise direction. Therefore, an infinite set of constellation points that preserve optimality in the receiver exist. The final choice depends on implementation considerations.

For the modem implementation described in this report, two considerations lead to a 45-degree rotation (see Figure 9) of the transmitter constellation diagram of Figure 2.

1. For the constellation points of Figure 9, the decision boundaries are the I and Q axes. That is, the decision region for point A is the first quadrant, the decision region for point D the second quadrant, and so on. Therefore, a decision can be made based only on the sign of the demodulated I ( $I'(nT_S)$ ) and Q ( $Q'(nT_S)$ ) channels.
2. For this set of constellation points, the products $\hat{Q}(nT_b)\ I'(nT_b)$ and $\hat{I}(nT_b)\ Q'(nT_b)$, required to calculate the phase error $E(nT_b)$ (see equation (8)), obtain on the average maximum values. Therefore, an optimum utilization of the dynamic range is achieved, and the error function calculated by (8) is the least-noise sensitive.

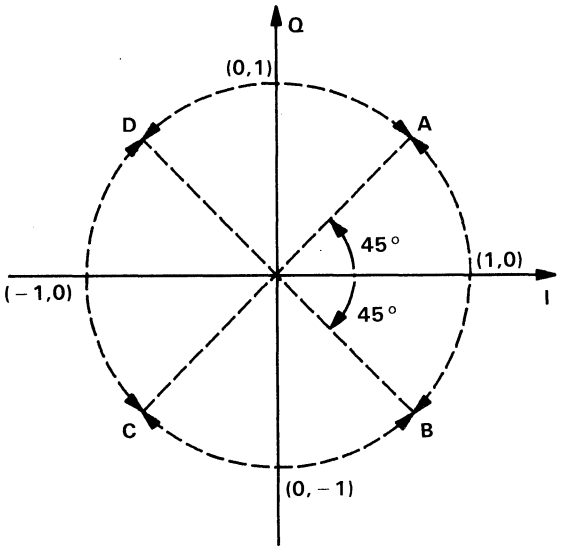



**Figure 9. Modem Receiver Decision Points**

The error $E(nT_b)$, the output of the phase detector, as given by (8) shows no apparent dependence on the phase or frequency difference between the local and incoming carriers. The discussion that follows shows the dependence of $E(nT_b)$ on the phase difference between the two carriers. This discussion is then extended to include the case of frequency as well as phase difference.

The inputs $I_p(nT_S)$ and $Q_p(nT_S)$ of the receiver demodulator (see Figure 6) are given by (see Appendix A)

$$I_p(nT_S) = I(nT_S)\cos(\omega nT_S + \theta_r) + Q(nT_S)\sin(\omega nT_S + \theta_r) \quad (11)$$

$$Q_p(nT_S) = I(nT_S)\sin(\omega nT_S + \theta_r) - Q(nT_S)\cos(\omega nT_S + \theta_r) \quad (12)$$

where $\omega$ and $\theta_r$ are the incoming (received) carrier frequency and phase, respectively. The outputs $I'(nT_S)$ and $Q'(nT_S)$ of the demodulator are described by equations (5) and (6), respectively. Introducing an arbitrary phase $\theta_l$ in the local carrier, (5) and (6) can be rewritten as

$$I'(nT_S) = I_p(nT_S)\cos(\omega' nT_S + \theta_l) + Q_p(nT_S)\sin(\omega' nT_S + \theta_l) \quad (13)$$

$$Q'(nT_S) = I_p(nT_S)\sin(\omega' nT_S + \theta_l) - Q_p(nT_S)\cos(\omega' nT_S + \theta_l) \quad (14)$$

where $\omega'$ is the local carrier frequency.

Assuming no frequency difference ($\omega' = \omega$), substitution of (11) and (12) into (13) and (14) gives

$$I'(nT_b) = I(nT_b)\cos(\theta_e) + Q(nT_b)\sin(\theta_e) \quad (15)$$

$$Q'(nT_b) = Q(nT_b)\cos(\theta_e) - I(nT_b)\sin(\theta_e) \quad (16)$$

where $\theta_e = \theta_r - \theta_l$ is the phase difference between the two carriers. Note that if $\theta_r = \theta_l$, then $\theta_e = 0$. From (15) and (16),

$$I'(nT_b) = I(nT_b)$$

$$Q'(nT_b) = Q(nT_b)$$

i.e., the output of the receiver demodulator at the middle of the baud is the same as the output of the transmitter coder (baseband information).

Assuming the decisions are correct, equations (9) and (10) hold. Substitution of (9), (10), (15), and (16) into the error signal defined by (8) gives

$$E(nT_b) = \{I^2(nT_b) + Q^2(nT_b)\}\sin(\theta_e) \quad (17)$$

The quantity $I^2(nT_b) + Q^2(nT_b)$ is a positive quantity (sum of squares). With

$$I^2(nT_b) + Q^2(nT_b) = K,$$

(17) can be rewritten as

$$E(nT_b) = K\sin(\theta_e) \qquad \text{where } K > 0 \quad (18)$$

Equation (18) is the same as (8) under the assumption of correct decisions ((9) and (10)). However, the phase information is more apparent in (18) than in (8), and leads to the following algorithm for the carrier recovery: If the phase of the received carrier is greater than the phase of the local carrier ($\theta_r > \theta_l$), the phase error $\theta_e$ is positive ($\theta_e > 0$). From (18), this implies that the output of the phase detector is also positive ($E(nT_b) > 0$). Therefore, if $E(nT_b) > 0$, the phase of the local carrier must be advanced, resulting in a smaller phase error. On the other hand, if the phase of the received carrier is less than the phase of the local carrier ($\theta_r < \theta_l$), the phase error is negative ($\theta_e < 0$). From (18), this implies that the output of the phase detector is also negative ($E(nT_b) < 0$). Therefore, if $E(nT_b) < 0$, the phase of the local carrier must be delayed.

In the case of frequency as well as phase difference, a similar development leads to

$$E(nT_b) = K \sin(\omega_e\, nT_b + \theta_e) \qquad \text{where } K > 0 \tag{19}$$

where $\omega_e = \omega - \omega'$ is the frequency difference between the incoming and local carriers. Since this frequency is very small (on the order of a few Hz) and the phase error correction is applied every baud (600 Hz), the term $\omega_e\, nT_b$ can be considered to be constant and the term $\omega_e\, nT_b + \theta_e$ in (19) an overall phase error. Therefore, using the algorithm discussed earlier, the frequency difference is compensated for as phase difference. Also note that in (19), $E(nT_b) = 0$ when $\omega_e = 0$ and $\theta_e = 0$; i.e., the local carrier is completely synchronized with the incoming carrier. Therefore, the error signal $E(nT_b)$ generated by the phase detector contains the information about the frequency and phase difference between the incoming and local carriers.

The error signal $E(nT_b)$ generated by the phase detector is processed by the loop filter as shown in Figure 8. Only the DC and low-frequency components of this signal must drive the Voltage Controlled Oscillator (VCO).[6] Therefore, the loop filter is basically a lowpass filter, whose most important characteristic is its bandwidth.

A large bandwidth of the loop filter implies that high-frequency components pass through the filter. Since the high-frequency information is applied to the VCO, the local carrier quickly locks-on to the incoming carrier. However, noise also passes through the filter, and the Bit Error Rate (BER) of the receiver increases. A narrow bandwidth decreases the BER but the lock-on time increases. An intelligent solution consists of starting with a wide bandwidth and, after the receiver is locked-on to the incoming carrier, narrow it down. This approach is used in this implementation and is described further in the Carrier Recovery Implementation subsection of "Functions Implemented in the TMS32010."

### *Clock Recovery*

The purpose of the Clock Recovery block in Figure 5 is to detect the middle of each baud. Once this is known, the decision block can make decisions with minimum probability of error because the energy of the incoming signal is maximum at the middle of the baud. The following paragraphs discuss a robust clock recovery approach.

As the demodulation point moves from one constellation point to another, at least one of the two channels is expected to cross zero (see Figure 9). This zero crossing indicates the beginning of a new baud interval. Therefore, one approach is to look at the zero crossings of the $I'(nT_s)$ and/or $Q'(nT_s)$ channels. However, there are two problems with that approach:

1. From (15) and (16), it can be seen that the presence of a phase difference $\theta_e$ between the two carriers can cause severe distortion of the zero crossings. To illustate this point, consider the first of the two equations, repeated here for convenience.

   $$I'(nT_b) = I(nT_b) \cos(\theta_e) + Q(nT_b) \sin(\theta_e)$$

The correct zero crossing information lies in $I(nT_b)$. Multiplication by $\cos(\theta_e)$ scales the $I(nT_b)$ curve, but does not change the location of the zero crossings. This is accomplished by the second additive term $Q(nT_b)\sin(\theta_e)$, which moves the scaled curve up or down depending on the term's sign.

2. The quantization noise in a digital implementation may result in undesirable nonlinearities and mislocation of the zero crossings. This is because finding the zero crossings involves monitoring the change of the sign of a particular variable (I channel and/or Q channel). A zero crossing occurs when this variable changes from a small positive value to a small negative value, and vice versa. Since the quantization noise can seriously affect small quantities (numbers), mislocation of the zero crossings may result.

The first of the above problems indicates that a clock recovery approach is required that is independent of the phase or frequency difference between the two carriers. This becomes clearer by considering the operation of the modem. The first task that the receiver must perform is to adjust the baud clock. During this adjustment, the two carriers are most likely to have a phase and/or frequency difference. Then, once the baud clock is adjusted, the carrier recovery algorithm places the local carrier in phase and in frequency with the incoming carrier.

Consider the energy of the incoming signal

$$\text{Energy} = I'^2(nT_s) + Q'^2(nT_s) \tag{20}$$

Substitution of (15) and (16) into (20), and the use of the identity

$$\sin^2(\theta_e) + \cos^2(\theta_e) = 1$$

gives

$$\text{Energy} = I^2(nT_s) + Q^2(nT_s) \tag{21}$$

This is the energy sent out by the transmitting modem. Equation (21) shows that the energy is independent of any phase and/or frequency difference between the two carriers. Geometrically, the energy is the square of the length of the vector that has its beginning at the intersection of the I and Q axis of Figure 9 and its tip at the demodulation point plotted on the constellation diagram. The path traced by the tip of the energy vector for a series of four consecutive baud intervals, each corresponding to a 90-degree phase change, is shown in Figure 10.

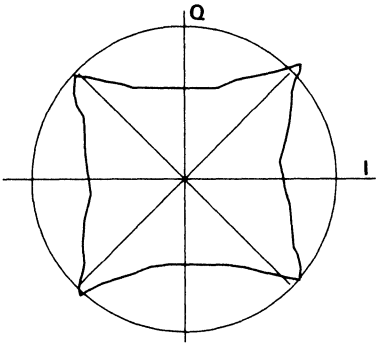



**Figure 10. Trace of Demodulation Point Plotted on the Constellation Diagram**

The plot shown in Figure 10 was obtained using a simulator. It can be seen that the signal energy ($I^2 + Q^2$) achieves its maximum value at the middle of each baud. Before and after the middle of the baud, the length of this vector is less than maximum. If there is a transition from one quadrant to another, this vector goes through a minimum, thus indicating the beginning of a new baud interval. Only if the same constellation point is transmitted because of a zero-degree phase change does such a transition not occur. It is easy to explain now why a series of zero-degree phase changes can create problems for the receiver. Zero-degree phase changes imply that the transmitter keeps sending the same constellation point. Therefore, the energy vector at the receiver does not go through a minimum for a series of baud intervals; i.e., during these intervals the receiver cannot adjust the baud clock and therefore may lose lock. This situation is avoided with the inclusion of the scrambler in the transmitter structure.

The frequency of the energy minima is discussed next. From Figure 9, it can be seen that there are four possible transitions for each constellation point. For example, consider constellation point A. The four possible transitions are: from point A to B, from A to C, from A to D, and from point A to A (i.e., receiving a zero-degree phase change). Three out of the four possible transitions result in a quadrant change (transitions from point A to points B, C, or D). For these transitions, the energy vector goes through a minimum. The fourth transition (from point A to itself), does not result in a quadrant change, but due to the presence of the scrambler, the probability of its occurrence is less than 0.25 (one out of four). Therefore, the average frequency of these minima is greater than 450 Hz for a 600-Hz baud frequency.

For the baud clock adjustment, the advantages of the energy-based approach over the zero crossings-based approach are:

1. The energy-based approach is independent of the phase and frequency difference between the two carriers, and therefore it gives the correct information about the incoming baud boundaries.

2. The average frequency of the energy minima is greater than 450 Hz while the average frequency of the zero crossings of the I or Q channels is between 300 and 400 Hz. The explanation follows. In the four possible transitions for each constellation point, two of them result in a zero crossing for a particular channel. Considering, for example, the transitions of constellation point A of Figure 9, the transitions from point A to points C or D result in a zero crossing for channel I. The transitions from point A to points B or C result in a zero crossing for channel Q. This implies that for a baud frequency of 600 Hz, the frequency of the zero crossings of a particular channel (I or Q) is on the average 300 Hz (two out of four). Because of the scrambler, the probability of retransmitting the same constellation point (zero-degree phase change) is minimized. This implies that on the average the frequency of the zero crossings of a particular channel increases. In the limit (no zero-degree phase changes), the average frequency of the zero crossings approaches 400 Hz (two out of three). Therefore, the average frequency of the zero crossings of a particular channel is between 300 and 400 Hz. To obtain more information from the zero crossings (greater average zero crossings frequency), the zero crossings of both the I and Q channels must be considered. However, this approach involves monitoring two quantities (I channel and Q channel) compared to monitoring only one (energy) if the energy-based approach is used.
3. Using the energy-based clock recovery technique described in the Baud Clock Alignment Implementation subsection of "Functions Implemented in the TMS32010," the quantization noise effects are less severe compared to those of a zero crossing-based approach.

## Modem Hardware Description

A brief description of the hardware used for the implementation of the Bell 212A/V.22 modem is covered in this section. Most of the signal processing required for the implementation of the modem functional blocks described in the previous section is performed digitally by the TMS32010 digital signal processor (see "Functions Implemented in the TMS32010"). The DTE interface and the protocol are handled by the TMS7742[10,11], an 8-bit EPROM microcomputer with an on-chip UART. Therefore, the hardware required for the system is minimal and consists primarily of the TMS32010 and TMS7742 processors, their memory, the A/D and D/A converters, and the associated antialiasing and smoothing filters.

To aid in the development and prototyping of this project, off-the-shelf development tools were used to build the modem hardware. The TMS32010 and the TMS7742 were emulated using Extended Development Systems (XDS) (part # TMDS3262211 for the TMS32010, and part # TMDS7062230 for the TMS7742). For the A/D and D/A conversions, the TMS32010 Analog Interface Board (AIB) ( part # RTC/EVM320C/06) was used. The Cermetek CH1810, Data Access Arrangement (DAA) approved by the Federal Communications Commission (FCC), is used for the telephone-line interface. A block diagram of the modem system hardware is shown in Figure 11.

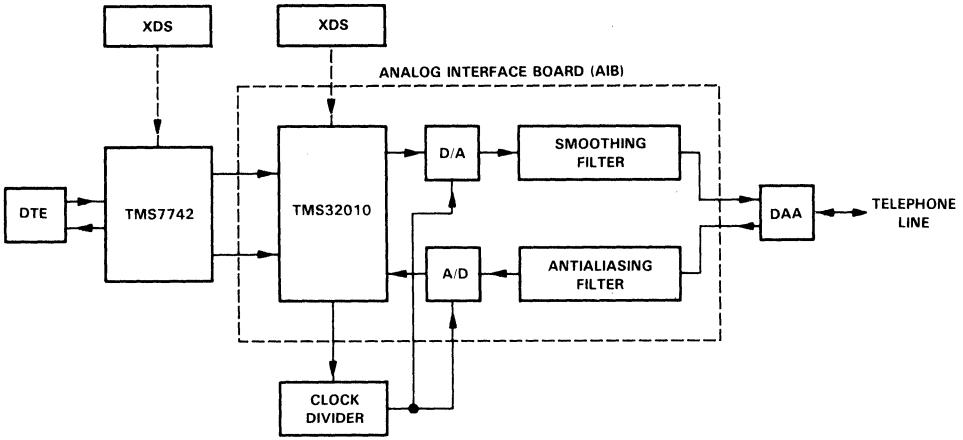



**Figure 11. Modem Hardware Block Diagram**

## Analog Interface

Compensative gain circuits have been placed between the DAA and the analog-to-digital converter. The gain circuit on the receiver side (see Figure 5) was added to match the peak amplitude of the signal from the phone line ($-9$ dbm or 0.77 V peak-to-peak) with the maximum range of the analog-to-digital converter (10 V peak-to-peak). This allows as much as possible of the A/D's dynamic range to be used without causing saturation. The gain circuit on the transmitter side (see Figure 3) is designed to attenuate the output of the digital-to-analog converter (10 V peak-to-peak) to a level consistent with the phone system signal strength limits ($-12$ dbm or 0.55 V peak-to-peak).

## Filters

The analog antialiasing and smoothing filters used by the A/D and D/A converters are sixth-order lowpass filters existing on the AIB, implemented using cascaded second-order opamp filter sections. These filters are designed with cutoff frequencies around 4.7 kHz in order to satisfy the Nyquist criterion requirements of the system.

## Data Converters

Analog Devices' AD565A and ADC80, monolithic A/D and D/A converters on the AIB, are configured for a $\pm 10$ V full-scale range and are interfaced to I/O port 2 of the TMS32010. The sampling rate for the conversions is determined by a set of presettable counters configured as frequency dividers. These counters are driven by the TMS32010's CLKOUT signal and produce a periodic sampling clock that initiates A/D and D/A conversions. The sampling frequency used is 9.6 kHz.

## TMS32010/TMS7742 Interface

The TMS32010 interfaces to the TMS7742 in parallel as shown in Figure 12.

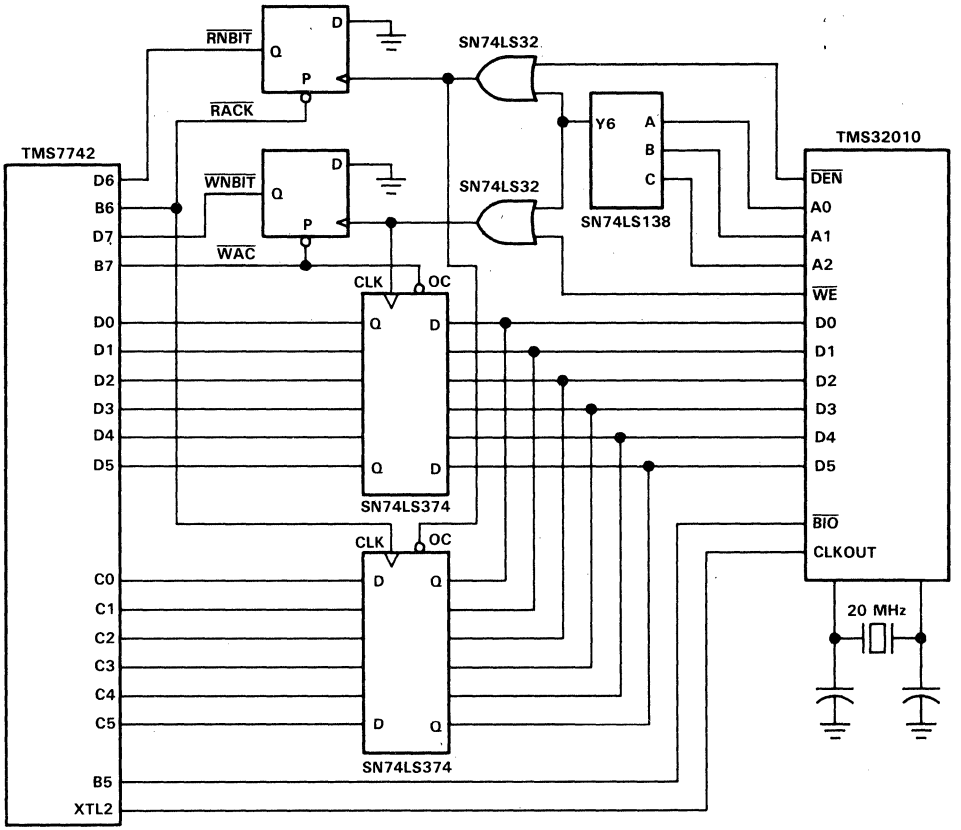



**Figure 12. TMS7742 and TMS32010 Interface**

The TMS7742 is mapped as an I/O device at Port 6 of the TMS32010. When the TMS32010 writes to Port 6, the $\overline{\text{WNBIT}}$ line goes active (D7). The TMS7742 polls this line and when active, reads data and status bits from the buffer into Port D. It also resets the $\overline{\text{WNBIT}}$ by sending a low pulse to the write acknowledge ($\overline{\text{WACK}}$) line. Similarly, when the TMS32010 reads from Port 6, the $\overline{\text{RNBIT}}$ line goes active. The TMS7742 immediately writes the new data to Port C and resets the read acknowledge ($\overline{\text{RACK}}$) line.

Six bits are used to interface the TMS32010 to the TMS7742. Two of these are used to pass the dibit data, two are used to send commands or status, and the other two are reserved to pass additional data if implementing the V.22 bis. The V.22 bis is a 2400-bps splitband modem that uses quad (four) bits instead of dibits. Table 4 lists the commands. The symbol X is used to indicate a don't care condition.

**Table 4. Modem Controller Commands**

| Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 | Command Description |
|---|---|---|---|---|---|---|
| 0 | 0 | X | X | X | X | Idle |
| 0 | 1 | X | X | X | X | Run local digital loopback |
| 1 | 0 | X | X | D1 | D0 | Run modem |
| 1 | 1 | D3 | D2 | D1 | D0 | Configure TMS32010 according to D3-D0 |

In the idle mode, the TMS32010 continues to monitor the commands from the TMS7742. In the local digital loopback mode, the TMS32010 reads the scrambled dibits from the TMS7742 and sends them back to the TMS7742. In the run mode, the TMS32010 reads the scrambled dibits from the TMS7742 and does the required encoding and modulation for the transmission through the telephone network. It also decodes the demodulated data and sends it to the TMS7742 for descrambling. Bits D0 and D1 are used to carry the dibit information. Bits D2 and D3 can be used when implementing the V.22 bis. In the configuration mode, the TMS32010 configures the transmit and receive filters for the originate or answer mode, depending on the data on D3-D0 (see TMS7742 source code provided in Appendix E).

## Functions Implemented in the TMS32010

The functions discussed in this section include all of the functional blocks described in the Modem Transmitter and Modem Receiver subsections with the exception of the transmitter scrambler and the receiver descrambler, which are implemented in the TMS7742. Table 5 shows the modem functions that are implemented on each device.

**Table 5. Modem Functions Implemented in the TMS32010 and TMS7742**

| Modem Transmitter | Implemented |
|---|---|
| Guard Tone Generator | No |
| Scrambler | TMS7742 |
| Encoder | TMS32010 |
| Digital Lowpass Filters | TMS32010 |
| Originate Mode Modulator | TMS32010 |
| Answer Mode Modulator | No |
| DTE Interface | TMS7742 |
| **Modem Receiver** | **Implemented** |
| Notch Filter | No |
| Originate Mode Bandpass Filters | TMS32010 |
| Answer Mode Bandpass Filters | No |
| Automatic Gain Control | TMS32010 |
| Demodulator | TMS32010 |
| Decision Block | TMS32010 |
| Decoder | TMS32010 |
| Descrambler | TMS7742 |
| Clock Recovery | TMS32010 |
| Carrier Recovery | TMS32010 |
| DTE Interface | TMS7742 |

Each variable used in this section is referred to by its name in the TMS32010 program enclosed in parentheses (see Appendix D).

## Transmit Filters

The transmit lowpass filters are implemented using 48-tap FIR structures, whose frequency responses exhibit a raised-cosine shape. A raised-cosine response is a filter response whose pass and stopbands are flat and whose rolloff characteristic is defined as a constant times a (1 + cos) term. The (1 + cos) term results in the rolloff shape being a portion of a cosine wave raised above the X-axis by one, hence the term 'raised-cosine response'. The raised-cosine response is used since it has been shown that it minimizes the intersymbol interference.[12]

The response shape of the transmit filters is actually defined by the square root of a raised-cosine response since the raised-cosine characteristic is split equally between the transmitter and receiver; i.e., both the transmitter and receiver filters are designed to exhibit the square root of the raised-cosine response. This results in the combined, end-to-end response of the path from transmitter to receiver being the full raised-cosine response.

The frequency-response characteristic of the transmit lowpass filters, as shown in Figure 13, rolls off smoothly to approximately −40 dB at 600 Hz.

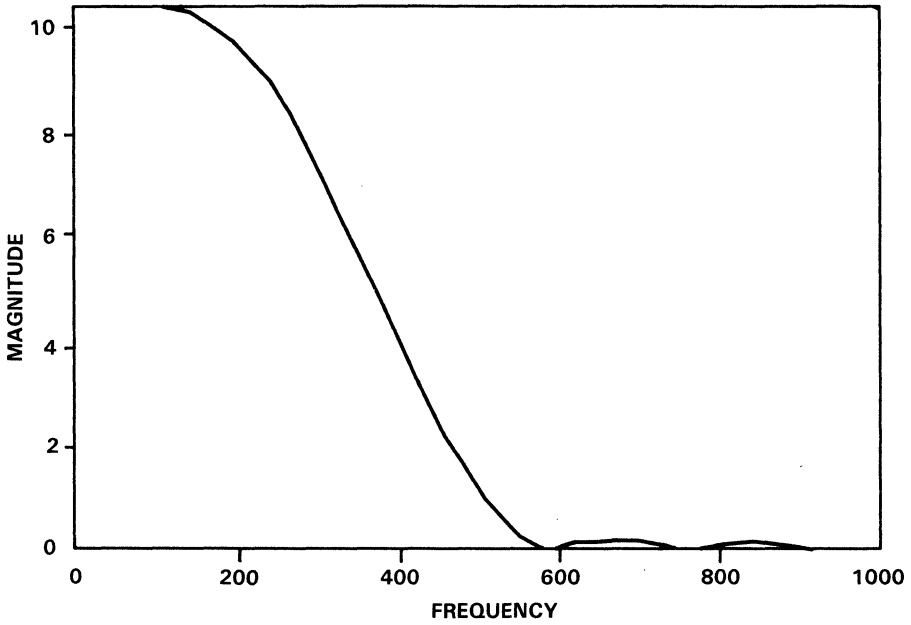



**Figure 13. Frequency Response Characteristics of Transmit Lowpass Filters**

The FIR structure is well suited to implementation of these filters, because FIR filters are stable, simple in structure, and can be designed to exhibit linear phase. These filters are easily implemented on the TMS32010 since the processor provides special instructions and architectural features that facilitate this type of algorithm. A signal flowgraph of the FIR filter structure is shown in Figure 14.

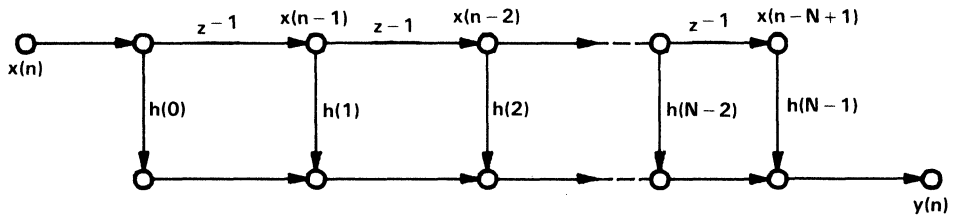



**Figure 14. Signal Flowgraph of the FIR Filter Structure**

As the flowgraph illustrates, this type of filter uses no feedback, which accounts for its stable behavior. FIR filters implement a transfer function of the form

$$H(z) = B_0 + B_1 z^{-1} + B_2 z^{-2} + B_3 z^{-3} + \ldots + B_n z^{-n} \quad (22)$$

The parameters in (22) that determine the characteristics of the specific filter implemented are the B coefficients $B_0$-$B_n$. In the case of the modem filters, the primary task in designing the filter is the determination of these coefficients so that the filter has the desired response shape, in this case, the raised-cosine response shape. For a detailed description of FIR and IIR filter design for the TMS32010, refer to "Implementation of FIR/IIR Filters with the TMS32010/TMS32020," an application report[13], and to *Digital Filter Design*, a book by T.W. Parks and C.S. Burrus.[14]

The raised-cosine response shape is defined by

$$H(f) = \begin{cases} \dfrac{1}{2\ B_t} & |f| < f_1 \\ \dfrac{1}{4\ B_t} \left\{ 1 + \cos \left[ \dfrac{\pi\ (\ |f| - f_1\ )}{2\ B_t - 2\ f_1} \right] \right\} & f_1 < |f| < 2\ B_t - f_1 \\ 0 & |f| > 2\ B_t - f_1 \end{cases} \qquad (23)$$

where $f_1 = (1 - p)\ B_t$.

For this design, $B_t = 300$ Hz and $p = 0.75$. Note that (23) describes the ideal zero-phase version of the raised-cosine response.

The actual frequency response of the transmit filters, shown in Figure 13, is the square root of the raised-cosine response described by (16).

To calculate the B coefficients required to implement this response in an FIR filter, the square root of (23) is first calculated. The Inverse Fourier Transform of the response is then used to generate the time-domain representation of the filter transfer function (the impulse response of the filter). In an FIR filter, the impulse response of the filter corresponds directly to the filter coefficients. Therefore, obtaining the coefficients requires merely shifting the impulse response in time to obtain a linear-phase version of the filter, and then sampling the impulse response at the system sampling rate.

After the filter coefficients are obtained, implementation of the filter digitally in the TMS32010 is accomplished by directly translating the signal flowgraph of Figure 14 into assembly language code.

As shown in Figure 14, the output of the filter is defined to be the sum of each of the delayed versions of the input, multiplied by the appropriate coefficient. In the TMS32010, the delayed versions of the previous input samples are stored in a table with the oldest sample stored at the highest address and the newest sample stored at the lowest address.

In the TMS32010 implementation, the transmit filters are arranged in a somewhat different manner from that which is commonly used for digital filters. In many digital filters, the input sample rate is the same as the output sample rate. In the transmit filters, however, the input sample rate is reduced because the rate of change of the information

entering the filter is known to be slower than the filter sample rate. Filters of this type are known as interpolating filters, and the ratio of the output sample rate to the input sample rate is referred to as the interpolation factor. In the modem transmit filters, the input sample rate is 600 Hz (the baud rate), and the output sample rate is 9.6 kHz, resulting in an interpolation factor of 16. As a result, the input of the filter is updated only after every 16 output samples, and is zero otherwise. Thus, the effective input $a(nT_S)$ to the transmit filters can be described for the I channel as

$$a(nT_S) = \begin{cases} I\left(\frac{nT_b}{L}\right) & \text{for } n = 0, \pm L, \pm 2L, \text{ etc.} \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

and for the Q channel as

$$a(nT_S) = \begin{cases} Q\left(\frac{nT_b}{L}\right) & \text{for } n = 0, \pm L, \pm 2L, \text{ etc.} \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

This technique reduces the number of multiplications required to compute the filter output from N to N/L where N is the length of the filter and L is the interpolation factor.

In both the transmit and receive filters, sampling of the nonzero portion of the filter impulse response at the system sample rate results in only 37 taps required for proper implementation of the filters. However, since the transmit filters are interpolating filters with an interpolation factor of 16, 16 taps are processed for each sample of the input. As a result, the number of taps in the filter must be an integer multiple of 16. In this case, 48 actual taps are used.

With a 48-tap filter and an interpolation factor of 16, only three multiplies are required to calculate the output of the filter. Because of this, these filters are coded on the TMS32010 somewhat differently than FIR filters that are not interpolated. In most filters, the data is shifted each time a sample is processed. With interpolation, the data is shifted only when a new input is processed, i.e., every 16 samples. During the remaining samples (when a new input is not being received), instead of shifting the data, a pointer (XPTR) is shifted through the table of coefficients so that effectively the coefficients are shifted. Thus, the complete filter output can be calculated with the following short section of code:

```
ZAC             * CLEAR ACCUMULATOR
LT      XIBUF2  * LOAD OLDEST SAMPLE
MPY     CX2     * MPY BY COEFF 2
LTD     XIBUF1  * LOAD NEXT SAMPLE
MPY     CX1     * MPY BY COEFF 1
LTD     XIBUF0  * LOAD NEWEST SAMPLE
MPY     CX0     * MPY BY COEFF 0
APAC            * MAKE FINAL SUM
SACH    XIOUT,1 * STORE OUTPUT
```

This code calculates the output of the I channel filter when a new input sample is being processed. The code that implements the filter output calculation when a new sample is not being input is similar to this code except that LTA instructions are used in place of LTD instructions.

During samples in which new inputs are being received, the inputs and the coefficients are shifted. This results in savings in data RAM space since only three data values must be stored.

## Receive Filters

The receiver bandpass filters are implemented using 37-tap FIR structures, which also exhibit a raised-cosine frequency response characteristic. These filters are virtually identical in structure to the transmit lowpass filters, with the exceptions that the cutoff frequencies are different and the receive bandpass filters do not interpolate since the input and ouput sample rates are the same. Like the transmit lowpass filters, the actual response implemented in these filters is the square root of the raised-cosine response since this response is split equally between the transmit and receive sections. The receive filters are centered around the carrier frequency $f_c$ (1200 Hz for originate and 2400 Hz for answer), and roll off smoothly to approximately $-40$ dB at $f_c \pm 600$ Hz. The frequency response characteristic of these filters is shown in Figure 15.

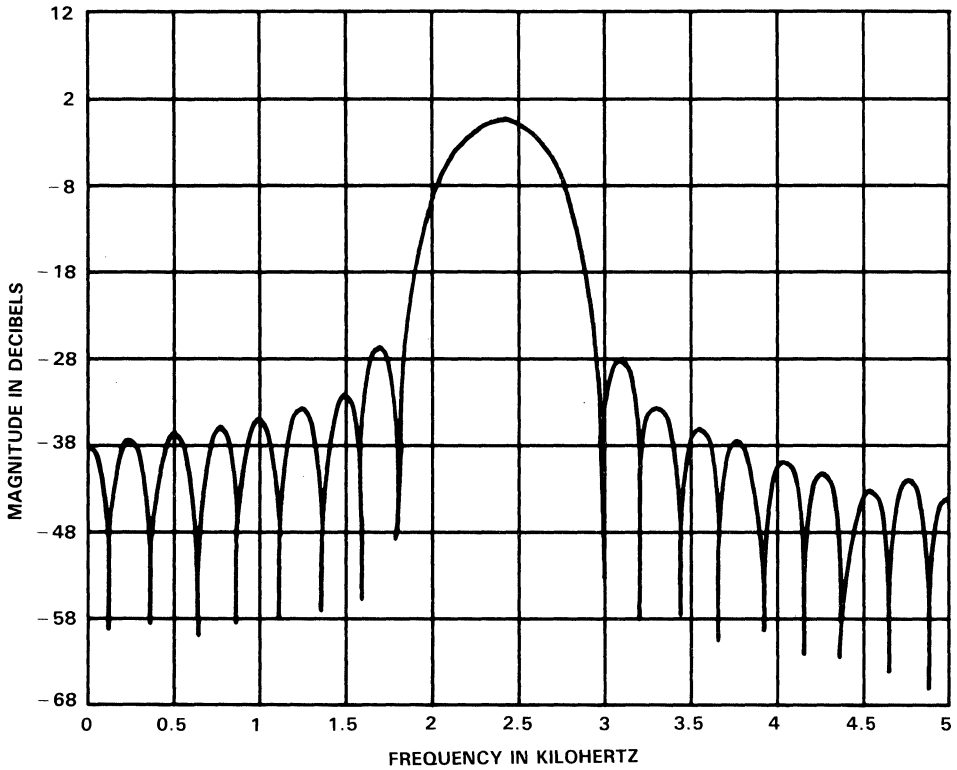



**Figure 15. Frequency Response of Receiver Bandpass Filters**

Except for the difference in filter order (the number of taps in the filter), the signal flowgraph and transfer function equation for the receive filters are identical to those of the transmit lowpass filters shown in Figure 13 and described by equation (22), respectively.

Besides being similar in structure to the transmit lowpass filters, the receive filters are actually designed directly from the transmit filters by simply shifting the filters' center frequencies. This is possible because the bandwidth of the transmit filters is the same as that required for the receive filters, and the transmit filters exhibit the raised-cosine response also required for the receive filters.

In order to generate the proper coefficients to implement the receive filters, the coefficients of the transmit filter are multiplied by a sine wave to obtain the I channel coefficients and by a cosine wave to obtain the Q channel coefficients. Specifically, if $h(nT_S)$ are the transmit filter coefficients, the receive filter coefficients $h_1(nT_S)$ and $h_2(nT_S)$ are obtained by

$$h_1(nT_S) = 2\ h(nT_S)\ \cos(n\omega T_S) \quad \text{(I channel)}$$
$$h_2(nT_S) = 2\ h(nT_S)\ \sin(n\omega T_S) \quad \text{(Q channel)} \tag{26}$$

where $T_S$ is the sampling period. Note that the factor of two must be included for the original frequency spectrum to be translated to the new frequency with the same magnitude.

The result of multiplying the transmit filter coefficients by sine and cosine is to effectively modulate their frequency response characteristics by a carrier at the frequency of the sine and cosine waves. This translates the frequency spectrum of the resultant filter up in frequency to a point centered around the frequency of the modulating signal, which is precisely what is required for the receive bandpass filters. Accordingly, bandpass filters for the originate mode are multiplied by sine and cosine functions at 1200 Hz and those for the answer mode are multiplied by sine and cosine functions at 2400 Hz, thus yielding the exact filters required.

In addition to shifting the frequency spectrum of the filters to the appropriate center frequencies, the fact that the I channel filter is multiplied by a sine function and the Q by a cosine function results in another important characteristic of these filters; the outputs of these filters are exactly 90 degrees out of phase with respect to each other. This provides a convenient method for implementing the phase shift required for proper demodulation of the I and Q channels. Also, since the filters are symmetric FIR structures, their phase response is linear, and the difference in phase shift between the two filters is precisely 90 degrees. This is beneficial because deviations from a precise 90-degree phase shift can cause serious distortion in other parts of the modem receiver.

In a direct implementation of this type of filter on the TMS32010, the filter output is calculated by repeatedly using the following two-instruction sequence:

```
LTD     * LOAD T, ACCUMULATE, DATA SHIFT
MPY     * MULTIPLY BY NEW COEFFICIENT
```

This sequence performs the following four operations:

1. Loads the T register with the input value,
2. Multiplies the input value by the appropriate coefficient,
3. Adds the product to the accumulator, and
4. Shifts the input data one place in the table, making room for the next input sample.

For FIR filters, a sequence of pairs of the LTD and MPY instructions is all that is required to implement the complete filter.

In the TMS32010, the receive filters are implemented in a somewhat more conventional manner than the transmit filters. The receive filters do not interpolate; however, due to careful choice of sample points on the impulse response, every second

coefficient in each filter is zero, reducing by a factor of two the number of LTD/MPY instruction pairs that must be executed to calculate the filter output.

Another feature of the FIR structure, which simplifies the implementation of these filters, is that since there is no feedback, the delay path ($x(n-1)$, $x(n-2)$,..., in Figure 14) for the two filters contains the same values for input samples in data RAM. Because of this, the same delay path can be used for both the I and the Q channel filters. This reduces by a factor of two the RAM required for data storage. As a result, the code that implements the I channel filter (processed first) uses LTA instructions instead of LTDs, performing no shift of the input data table within memory. A one-position shift of the data table is then performed when the Q channel filter output is calculated.

Even though every other coefficient is zero, each sample in the delay table must still be shifted by one memory location during each pass through the filter. Since the Q channel filter performs this shifting but only operates on every second data point, an additional DMOV instruction is coded between each LTD/MPY instruction pair in order to shift the even-numbered data table entries.

The assembly code that implements the Q channel bandpass filter is shown below.

```
*
*     FIRST (Nth) TAP SETS UP FOR REST OF FILTER
*
        ZAC                     * CLEAR ACCUMULATOR
        LT      RBUF35          * LOAD T REGISTER
        DMOV    RBUF35          * SHIFT OLD VALUE
        MPYK    QCF35           * MPY BY COEFFICIENT
        DMOV    RBUF34          * PERFORM EXTRA SHIFT
*
*     SECOND TAP
*
        LTD     RBUF33          * LOAD T, ACCUMULATE, DATA SHIFT
        MPYK    QCF33           * MULTIPLY BY COEFFICIENT
        DMOV    RBUF32          * PERFORM EXTRA SHIFT
          .
          .
          .
*
*     LAST TAP
*
        LTD     RBUF1           * LOAD T, ACCUMULATE, DATA SHIFT
        MPYK    QCF1            * MULTIPLY BY COEFFICIENT
        DMOV    RBUF0           * PERFORM EXTRA SHIFT
*
        APAC                    * ADD LAST SUM
        SACH    QSUM,4          * STORE FILTER OUTPUT
```

## Automatic Gain Control Implementation

To better control the signal strength of the receiver, a software Automatic Gain Control (AGC) algorithm was added. The need of an AGC stems from the use of thresholds in both the carrier recovery and clock recovery algorithms. For increased performance, these thresholds (discussed in the following two subsections) must remain valid (unchanged) for different levels of the incoming signal. This is achieved with the use of the software AGC.

The arrangement of the AGC with respect to the other functional blocks of the modem receiver was shown in Figure 5. The AGC monitors the I channel of the receiver and calculates a gain correction factor. Both the I and Q channels are then multiplied by this gain correction factor so that the signal maxima remain within a certain range. This range is narrow compared to the range of the incoming signal maxima. The peak-to-peak voltage of the incoming signal is between 2 mV and 700 mV. In 16-bit hexadecimal Q15 format,[15] this range is from >5C to >5999. However, with the use of the software AGC, the signal maxima are in the range 780 mV (>6400) to 820 mV (>6900).

The gain correction factor is calculated once every three bauds by a two-step process. First, the three maximum values of the signal (BSMAX), each one corresponding to one baud (16 samples), are monitored and added to each other. A counter (AGCNT) is used to keep the count of the signal maxima. The previous running average is then added to this sum, and the result is divided by four to obtain the new running average (AGCRA). The division by four is accomplished by shifting the final sum, contained in the accumulator, two locations to the right before storing it in the memory as the new running average (AGCRA). The section of code that implements this step is listed below.

```
*
*     DETECT MAX SIGNAL STRENGTH OF I CHANNEL PER BAUD
*              (THIS CODE IS EXECUTED EVERY CYCLE)
*
AGCAL     EQU    $
          LAC    ISUM        * AGC VALUE CALCULATED USING ISUM
          ABS                * GET MAGNITUDE OF SIGNAL
          SUB    BSMAX       * COMPARE TO PREVIOUS MAX VALUE
          BLZ    OVRMAX      * IF LESS THAN, THEN SKIP UPDATE
          ADD    BSMAX       * RESTORE VALUE AND
          SACL   BSMAX       * STORE AS NEW MAX
*
*      MULTIPLY I  AND Q  CHANNELS BY AGC FACTOR
*
OVRMAX
```

```
*
*       UPDATE THE RUNNING AVERAGE ONCE EVERY THREE BAUDS
*            (THIS CODE IS EXECUTED ONCE EVERY BAUD)
*
AGCUPT  ZALH    AGCRA       * ADD THE NEW BSMAX VALUE
        ADD     BSMAX,14    * TO THE RUNNING AVERAGE
        SACH    AGCRA       * AND SAVE IT
        LAC     AGCNT       * DECREMENT RUNNING AVERAGE
        SUB     ONE         * SAVE IT AND
        SACL    AGCNT       * CHECK FOR ZERO
        SACH    BSMAX       * ZERO OUT RUNNING SIGNAL MAX
        BZ      OVROUT      * IF ZERO, THEN UPDATE AGC
        RET                 * ELSE RETURN TO CALLING SEQUENCE
OVROUT  LACK    3           * RESET RUNNING AVERAGE COUNT
        SACL    AGCNT       * TO THREE
        LAC     AGCRA       * MOVE AGCRA
        SACL    AGCLEV      * TO THE CALCULATION LEVEL
        LAC     AGCRA,14    * DIVIDE RUNNING AVERAGE SUM
        SACH    AGCRA       * BY 4 TO GET NEW RUNNING AVERAGE
```

At the second step, the gain correction factor (AGC) is calculated, based on the running average. A brute force approach is to divide the maximum-allowed signal level by the running average and obtain the gain correction factor as the result of this division. The maximum value of the product of the signal times the gain correction factor should then remain close to the maximum-allowed signal level. However, since divisions are costly in processing time, the second step is implemented by using the running average as an index (AGCLEV) to a 32-word lookup table. The offset to this table (AGCOFF) is added to the index (AGCLEV) to calculate the table entry on which the gain correction factor (AGC) is located. The TBLR instruction is then used to transfer the gain correction factor from program memory to data memory. To lessen the code space required to handle the AGC lookup table, the code uses only the six most significant bits of the running average. This requires a 64-word lookup table. However, since the most significant bit of the six bits is always one, only 32 entries of the table are needed. The gain correction factor, obtained by the table lookup, is shifted so that the product of the gain correction factor times the incoming signal is in Q14 format (designer's choice). The shift factor is provided by the BASIC program used to generate the AGC lookup table (see Appendix C). The TMS32010 code that implements the calculation of the gain correction factor is shown below.

```
        LAC     AGCLEV      * GET AVERAGE MAX SIGNAL LEVEL
        SUB     ONE,14      * COMPARE TO 16384
        BLZ     ASHF1       * IF LESS THAN SHIFT TABLE LOOKUP
        LAC     AGCLEV,7    * GET LOOKUP VALUE
        SACH    TEMP        * MOVE LOOKUP VALUE TO
```

```
        LAC     TEMP        * THE LOW HALF OF THE ACC
        ADD     AGCOFF      * ADD IN TABLE OFFSET
        TBLR    AGC         * AND GET AGC VALUE
        LAC     AGC,15      * DIVIDE THE AGC VALUE
        SACH    AGC         * BY 2 TO FORCE TO Q14 MODE
        RET                 * RETURN TO CALLING SEQUENCE
ASHF1   ADD     ONE,13      * COMPARE TO 8192
        BLZ     ASHF2       * IF LESS THAN SHIFT TABLE LOOKUP
        LAC     AGCLEV,8    * GET LOOKUP VALUE
        SACH    TEMP        * MOVE LOOKUP VALUE TO
        LAC     TEMP        * THE LOW HALF OF THE ACC
        ADD     AGCOFF      * ADD IN TABLE OFFSET
        TBLR    AGC         * AND GET AGC VALUE
        RET                 * RETURN TO CALLING SEQUENCE
ASHF2   ADD     ONE,12      * COMPARE TO 4096
        BLZ     ASHF3       * IF LESS THAN SHIFT TABLE LOOKUP
        LAC     AGCLEV,9    * GET LOOKUP VALUE
        SACH    TEMP        * MOVE LOOKUP VALUE TO
        LAC     TEMP        * THE LOW HALF OF THE ACC
        ADD     AGCOFF      * ADD IN TABLE OFFSET
        TBLR    AGC         * AND GET AGC VALUE
        LAC     AGC,1       * AGC VALUE*2 TO ADJUST
        SACL    AGC         * FOR LOWER SIGNAL STRENGTH
        RET                 * RETURN TO CALLING SEQUENCE
          .
          .
          .
ASHF6   ADD     ONE,5       * COMPARE TO 32
        BLZ     NOEDT       * LOST MINIMUM ENERGY LEVEL
        LAC     AGCLEV,13   * GET LOOKUP VALUE
        SACH    TEMP        * MOVE LOOKUP VALUE TO
        LAC     TEMP        * THE LOW HALF OF THE ACC
        ADD     AGCOFF      * ADD IN TABLE OFFSET
        TBLR    AGC         * AND GET AGC VALUE
        LAC     AGC,5       * AGC VALUE*32 TO ADJUST
        SACL    AGC         * FOR LOWER SIGNAL STRENGTH
        RET                 * RETURN TO CALLING SEQUENCE
```

The AGC table was generated by the BASIC program listed in Appendix C. This program is written to execute on any MS-DOS operating system. The program prompts the user for the table size and gain range factor, and then generates and stores the AGC table. The table is stored in a format that allows insertion directly into the user's code.

## Carrier Recovery Implementation

The carrier recovery is implemented with a phase-locked loop, as explained in the Modem Receiver subsection. In Figure 8, the functional blocks that must be digitally implemented are the phase detector, loop filter, and Voltage Controlled Oscillator (VCO).

### *Phase Detector*

In the middle of each baud, the phase detector block calculates an equation equivalent to (8), repeated below for convenience,

$$E(nT_b) = \hat{Q}(nT_b)\, I'(nT_b) - \hat{I}(nT_b)\, Q'(nT_b)$$

where I′ (RECI) and Q′ (RECQ) are the baseband (demodulated) I and Q channels, and $\hat{I}$ and $\hat{Q}$ are the I and Q channel decisions. The derivation of the equivalent equation to (8) is discussed next.

In Figure 9, the I channel decision for constellation point A is the length of the projection of the vector $\vec{OA}$ on the I axis. Similarly, the Q channel decision for constellation point A is the length of the projection of the vector $\vec{OA}$ on the Q axis. Since the four constellation points A, B, C, and D are located on the 45 and −45 degree lines, the lengths of these projections are the same. With this common length denoted by L, the I channel decisions can be expressed as

$$\hat{I}(nT_b) = \begin{cases} +L \text{ for points A and B} \\ -L \text{ for points C and D} \end{cases} \tag{27}$$

The value of L depends on the radius of the circle on which the four constellation points are located. Equation (27) can equivalently be expressed as

$$\hat{I}(nT_b) = \text{sgn}(I'nT_b))\, L \tag{28}$$

where sgn is the sign function defined as

$$\text{sgn}(I'(nT_b)) = \begin{cases} +1 & \text{if } I'(nT_b) > 0 \text{ (points A and B)} \\ -1 & \text{if } I'(nT_b) < 0 \text{ (points C and D)} \end{cases} \tag{29}$$

Similarly,

$$\hat{Q}(nT_b) = \text{sgn}(Q'(nT_b))\, L \tag{30}$$

Substitution of (28) and (30) into (8) gives

$$E(nT_b) = L\, \{\text{sgn}(Q'(nT_b))\, I'(nT_b) - \text{sgn}(I'(nT_b))\, Q'(nT_b)\} \tag{31}$$

Equations (31) and (8) are identical. However, (31) is the final step towards the equation implemented in the TMS32010. Since L in (31) is a positive constant, an equivalent error function that contains the phase and frequency information is

$$E'(nT_b) = \text{sgn}(Q'(nT_b))\, I'(nT_b) - \text{sgn}(I'(nT_b))\, Q'(nT_b) \tag{32}$$

Equation (32) is the one implemented in the TMS32010 as part of the carrier recovery algorithm. In this equation, sgn(I′) (SIGNI) and sgn(Q′) (SIGNQ) are the I and Q channel decisions, respectively. The TMS32010 code used to implement (32) is shown below.

```
*
* COMPUTE CARRIER ERROR SIGNAL
*
* e(t) = RECI*SIGNQ - RECQ*SIGNI
*
COMERR LT      RECI      * T=RECI
       MPY     SIGNQ     * P=RECI*SIGNQ
       LTP     RECQ      * T=RECQ, ACC=RECI*SGNQ
       MPY     SIGNI     * P=RECQ*SIGNI
       SPAC               * ACC=RECI*SIGNQ-RECQ*SIGNI
       SACH    ERROR,1   * STORE IN ERROR
```

***Loop Filter***

The error signal $E'(nT_b)$ (ERROR), generated by the phase detector (equation (32)), is filtered by the carrier recovery loop filter (see Figure 8). The filter was implemented as a first-order Infinite Impulse Response structure. In other words, the loop filter is just an integrator with transfer function

$$H_1(z) = \frac{B_1}{1 - A_1 z^{-1}} \tag{33}$$

where $A_1$ (PLL1) and $B_1$ (PLL2) are the filter coefficients.

A higher-order filter was not used, because high-order filter structures usually introduce more phase delay than first-order sections. Phase delays[16] are critical in the operation of a phase-locked loop, and their effects are difficult to analyze.

The time-domain equivalent of (33) is

$$y(n) = B_1 x(n) + A_1 y(n-1) \tag{34}$$

where x(n) is the input to the filter and y(n) the output. The signal flowgraph of the carrier recovery loop filter is shown in Figure 16.

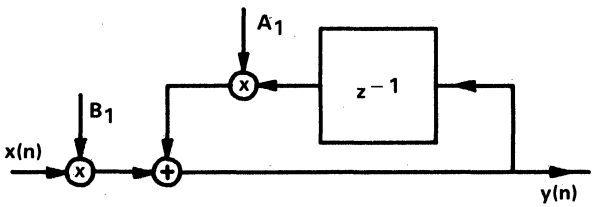



**Figure 16. Carrier Recovery Loop Filter**

The TMS32010, with a hardware on-chip multiplier, is most efficient in the implementation of such filter structures.[13] The code used to implement the carrier recovery loop filter (equation (34)) is shown below. The filter's input x(n) is stored in ERROR, and the filter's output y(n) is stored in ERRSIG.

```
*
* LOOP FILTER
*
        LT      PLL2        * T = PLL2
        MPY     ERROR       * P = PLL2*ERROR
        LTP     PLL1        * ACC = PLL2 ERROR, T = PLL1
        MPY     ERRSIG      * P = PLL1*ERRSIG
        APAC                * ACC = PLL2*ERROR + PLL1*ERRSIG
        SACH    ERRSIG,1    * STORE IN ERRSIG
```

The effect of the loop filter's bandwidth in the modem performance is considered in the following discussion where the bandwidth of the loop filter is defined as the frequency at which the magnitude of the filter's transfer function is 3 db below its maximum value. Therefore, the bandwidth of the loop filter is the frequency $\omega_b$ at which

$$20\log \frac{|H_1|\max}{|H_1(\omega_b)|} = 3 \tag{35}$$

where $|H_1|\max$ is the maximum value of the magnitude of the filter's transfer function. Substituting $z = e^{j\omega}$ in (33) gives

$$|H_1(\omega)| = \frac{|B_1|}{\{1 + A_1^2 - 2A_1 \cos(\omega)\}^{1/2}} \tag{36}$$

Equation (36) is maximum when the denominator is minimum. This is true for $\omega=0$, i.e., at DC. Substituting $\omega=0$ in (36) gives

$$|H_1|\max = \frac{|B_1|}{1-A_1} \qquad \text{where } 0 < A_1 < 1 \tag{37}$$

Substitution of (36) and (37) into (35) gives the following quadratic equation that relates the bandwidth of the loop filter $\omega_b$ to the coefficient $A_1$.

$$A_1^2 + 2A_1 \{\cos(\omega_b) - 2\} + 1 = 0 \tag{38}$$

Therefore, the value of the coefficient $A_1$ determines the bandwidth of the loop filter. Figure 17 shows a plot of the values of $A_1$ versus the bandwidth $\omega_b$, i.e., a plot of (38).

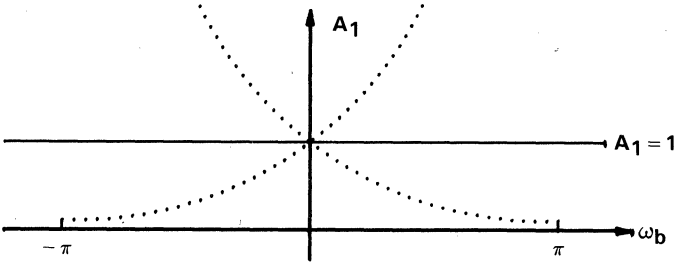



**Figure 17. Parameter $A_1$ vs. the Bandwidth of $H_1(z) = \dfrac{B_1}{1 - A_1\ z^{-1}}$**

The two curves in Figure 17 represent the solutions of the quadratic equation (38). From this figure, it can be seen that the closer $A_1$ is to unity, the narrower the bandwidth of the filter. If $A_1 = 1$, the magnitude response begins rolling off at zero frequency ($\omega = 0$). However, this situation must be avoided since $A_1 = 1$ results in placing a pole on the unit circle in the z-domain, thereby causing the filter to oscillate. Since only values less than unity of the coefficient $A_1$ result in a stable filter structure, Q15 format[15] was used to represent $A_1$.

The bandwidth $\omega_b$ is expressed in radians. Since the sampling frequency $f_b$ corresponds to $2\pi$ radians, the bandwidth of the loop filter in Hz is given by

$$\text{Bandwidth} = \frac{\omega_b\ f_b}{2\pi} \quad \text{Hz} \tag{39}$$

Since the loop filter runs once every baud, the sampling frequency $f_b$ is

$$f_b = \frac{1}{T_b} = 600 \text{ Hz}$$

with $T_b$ as the baud interval. This frequency should not be confused with the A/D and D/A sampling frequency designated by $f_s$ and having the value of 9600 Hz.

The bandwidth of the loop filter affects the Bit Error Rate (BER) and the time it takes for the modem receiver to lock-on to the incoming carrier. Initially, a large bandwidth results in a fast lock-on while a narrow bandwidth provides a good BER. Therefore, the ability to switch from a large bandwidth to a narrow one results in a better modem design. With the TMS32010, this is easily implemented using the TBLR instruction that transfers data from program memory to data memory.[15] On startup, $A_1$ (PLL1) is 0.539 or >4500 in Q15 format. This corresponds to a bandwidth of approximately 63 Hz. Once locked-on with the use of the TBLR instruction, the value of $A_1$ (PLL1) is changed to 0.953 or >7A00 in Q15 format. This corresponds to a bandwidth of approximately 6 Hz. Lock-on criterion is based on the magnitude of the error function

calculated by (32) being less than a certain threshold. The need and calculation of this threshold is covered later in this subsection. The TMS32010 code used to switch the loop filter's bandwidth is given below. The fifth bit of RECST is used as a flag, which if set indicates that the local carrier is locked-on to the incoming carrier.

```
        LAC     ONE,4       *  CHECK IF LOCAL CARRIER
        AND     RECST       *  IS LOCKED. IF SO, SWITCH
        BNZ     CARLCK      *  PLL FILTERS' BANDWIDTH
        B       NORMAL      *  EXECUTE NORMAL SEQUENCE
*
CARLCK  LACK    PLLC        *  CHANGE CARRIER PLL COEF. 1
        TBLR    PLL1
```

***Voltage-Controlled Oscillator***

Both the carrier used in the transmitter to modulate the data and the one used in the receiver for the demodulation (local carrier) were implemented in the TMS32010 using a 128-point sine table and a routine to drive it.[17] The voltage-controlled oscillator in the phase-locked loop for the carrier recovery generates the local carrier using this 128-point sine table. The frequency of this digital sine wave is 2400 Hz for an originate modem and 1200 Hz for an answer modem.

***Carrier Recovery Threshold***

The lowpass-filtered value of the error signal generated by the phase detector contains the information about the phase and frequency difference between the local and incoming carriers. If this value (ERRSIG) is positive, the local carrier must be advanced in phase. If negative, the local carrier must be delayed (see the Modem Receiver subsection). Since there are 128 points in the sine table, there is a 360/128 or 2.8125-degree jump going from one table entry to the next. This implies that corrections should not be made unless the magnitude of the error signal is greater than one table entry because redundant corrections introduce inaccuracies and noise. Therefore, the value of this threshold should correspond to the magnitude of the error signal when there is a 2.8125-degree phase error.

An estimate of the threshold can be obtained as described below. The relation of the phase error signal $E(nT_b)$ to the phase error $\theta_e$ is given by (18). Substituting 2.8125 for $\theta_e$ in (18) and taking the magnitude of both sides gives

$$|E(nT_b)| = |K \sin(2.8125)| \tag{40}$$

$K(I^2 + Q^2)$ is the signal energy, i.e., the maximum value of the I and Q channels. This value is set by the Automatic Gain Control. Since the software AGC used in this implementation of the Bell 212A/V.22 limits the signal maxima between 0.78 and 0.82 (see Automatic Gain Control Implementation in the Modem Receiver subsection), K is between 0.78 and 0.82. Using the average value of 0.80 for K, (40) gives

$$|E(nT_b)| = 0.039.$$

The threshold level should be at 0.039 if the gain of the loop filter given by (33) is unity. For DC, the gain G1 of the loop filter is given by (37), repeated below for convenience.

$$G_1 = |H_1|\max = \frac{B_1}{1 - A_1} \qquad \text{where } 0 < A_1 < 1$$

The coefficient $B_1$ (PLL2) was chosen to be 0.0039 (or >50 in the Q15 format). As explained earlier, once the receiver is locked, the value of coefficient $A_1$ (PLL1) is 0.953. From (37), the gain $G_1$ of the loop filter is $G_1 = 0.082$. Therefore, the threshold is scaled down to

$$\text{Effective Threshold} = 0.039 \times 0.082 = 0.0032.$$

This corresponds to >D in Q12, the format used for the threshold (designer's choice). After this initial estimate of the threshold was obtained, the final value of the carrier recovery threshold (TRSHD1), >7, was determined by trial and error. The calculated threshold is greater than the one obtained by trial and error, because of the use of the maximum value of the loop filter's gain in the threshold calculation.

To improve the lock-on characteristics of the modem, a two-level correction was used for the carrier recovery. If the magnitude of the error (ERRSIG) is less than the threshold (TRSHD1), no correction is applied. If the magnitude of the error is greater than the threshold but less than twice the threshold, one sine-table entry correction is applied by incrementing or decrementing the table entry pointer (RALPHA) by one. If the magnitude of the error is greater than twice the threshold value, then a two-table entry correction is applied by incrementing or decrementing the table entry pointer (RALPHA) by two. All of the corrections are applied to advance or delay the local carrier according to the algorithm described in the Modem Receiver subsection.

## Baud Clock Alignment Implementation

The purpose of the clock recovery is to identify the baud boundaries and inform the decision block when the middle of each baud occurs and therefore the optimum time to make an error-free decision (see Figure 5). As explained in the Modem Receiver subsection, one approach for clock recovery (adjustment of the baud clock) is to use the energy of the incoming signal. The energy is the sum of the squares of the demodulated I and Q channels (see equation (20)). As implied by (21), this quantity is independent of any phase and/or frequency difference between the incoming and local carriers.

The minima of the signal energy indicate the beginning of a new baud. This can be seen in Figure 18 where the signal energy is plotted every sample for several consecutive baud intervals.

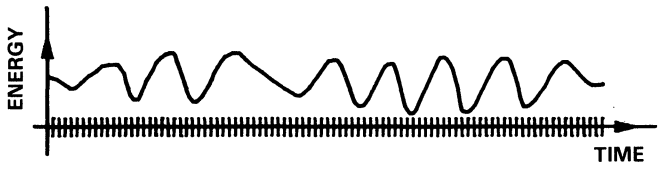



**Figure 18. Signal Energy Plotted Every Sample For Several Baud Intervals**

Each of the short vertical lines along the horizontal axis in Figure 18 corresponds to a sample time. This data was obtained using the XDS/22 emulator for the TMS32010. The block diagram for the clock recovery algorithm is shown in Figure 19. The functional blocks to be implemented are the error signal generator, loop filter, and baud clock.

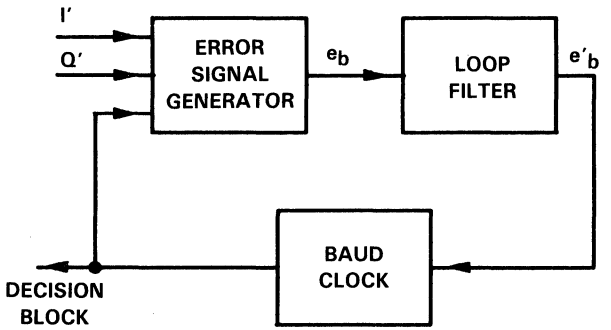



**Figure 19. Baud Clock Alignment Block Diagram**

***Error Signal Generator***

The error signal generator calculates the signal energy and from it generates an error signal $e_b$. This error signal contains the information about how close the local baud boundaries are to the incoming baud boundariers. The error signal is then lowpass-filtered so that noise and high-frequency components are removed. The output of the loop filter corrects the local baud clock.

The critical issue is how to calculate this error signal. Figure 20 shows the signal energy for a single baud interval. This figure was motivated from the realtime data of Figure 18. The 16 energy samples for this baud are indicated as E(0),E(1),...,E(15).

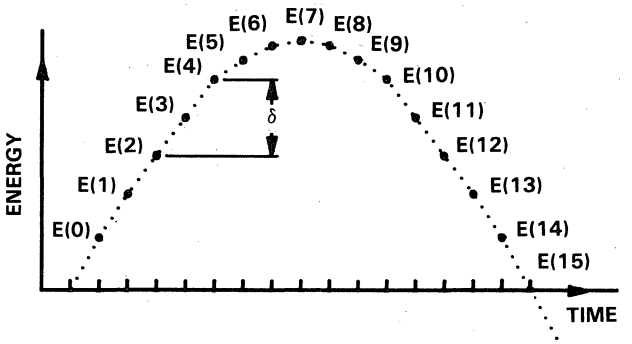



**Figure 20. Signal Energy Samples over a Baud**

From Figure 20, it can be seen that the energy sample E(7) is located at the middle of the baud (top of the 'energy hill'), and the rest of the samples are located symmetrically around it, i.e., E(6) = E(8), E(5) = E(9), and so on. Therefore, E(7) is taken to be the middle of the local baud. Consider now the difference between the energy sample E(11) that is four samples after E(7) and the energy sample E(3) that is four samples before E(7). If the local baud clock is correctly aligned so that E(7) corresponds to the middle of the incoming baud, then

$$E(11) - E(3) = 0.$$

If

$$E(11) - E(3) > 0,$$

then the sample E(7) is located to the left of the middle of the baud. This means that the middle of the local baud occurred earlier than the middle of the incoming baud. Therefore, the local baud clock must be delayed. On the other hand, if

$$E(11) - E(3) < 0,$$

the middle of the local baud occurred later than the middle of the incoming baud. Therefore, the local baud clock must be advanced.

In summary, the error signal generator computes the signal energy at sample points 3 (PENRGY) and 11 (ENRGY). The sample count information (SAMPLE) is provided by the baud clock shown in Figure 19. The error signal generator then calculates the error signal $e_b$ (BERROR) defined by

$$e_b = E(11) - E(3) \tag{41}$$

The subscript b represents baud since this signal is calculated once every baud.

### *Loop Filter*

Perturbations that may occur in the communications medium pass onto the demodulated I and Q channels. This can be seen from the data of Figure 18 where even with the presence of the automatic gain control, the energy levels are not exactly the same for every baud. Also, the duration of each baud in Figure 18 is not exactly sixteen samples (sixteen short vertical lines) as it theoretically should be. These perturbations result in abrupt changes of the signal generated by the error signal generator. Therefore, the error signal is not directly fed into the baud clock. Instead, it is lowpass-filtered by the loop filter. This removes noise and high-frequency components and results in a stable clock recovery.

The loop filter was implemented as a first-order recursive filter. The transfer function is of the same form as (33).

$$H_2(z) = \frac{B_2}{1 - A_2 z^{-1}} \tag{42}$$

Just as with the loop filter used for the carrier recovery, the most important characteristic of the loop filter used for the clock recovery is its bandwidth. A wide bandwidth results in a quick adjustment of the local baud boundaries to the incoming baud boundaries. A narrow bandwidth results in a more stable clock recovery. A good approach for this filter's design is to start with a wide bandwidth and then switch to a narrow one. All of the information provided in the Carrier Recovery Implementation subsection relating the coefficient $A_1$ to the loop filter's bandwidth apply here as well. With the use of the TBLR instruction, after the receiver is locked-on to the incoming carrier, the initial wide bandwidth is switched to a narrow one. The initial value of $A_2$ is 0.5, which is >4000 in Q15 format and corresponds to approximately a 70-Hz bandwidth. After the receiver is locked, this value changes to 0.91 (>7500 in Q15 format), which corresponds to a bandwidth of approximately 10 Hz. The criterion used for the receiver being locked-on is the magnitude of the error function calculated by (32) being less than the threshold used for the carrier recovery (TRSHD1).

### *Baud Clock*

The output of the loop filter, designated by $e'_b$ in Figure 19, drives the local baud clock. The baud clock tracks the sample count (SAMPLE) and thus informs:

1. The decision block when it is the middle of the baud (sample 7) and thus the optimum time for demodulation, and
2. The error signal generator when the sample count is 3 and 11 so that the error signal $e_b$ can be calculated.

These two objectives are achieved with the use of a 16-entry table in the program memory. Each table entry contains the address of a subroutine task to be performed between two consecutive samples. The tasks are numbered 0, 1,..., 15. Table 6 shows the memory map of the 16 tasks performed by the modem receiver.

**Table 6. Memory Map of Tasks Performed by the Modem Receiver**

```
*
* TASK MASTER SEQUENCE TABLE  (RECEIVE)
* TASKS ARE EXECUTED FROM BOTTOM TO TOP
*
TSKSEQ  EQU     $
        DATA    DUMMY   * UNUSED CYCLE                    15
        DATA    DUMMY   * UNUSED CYCLE                    14
        DATA    DUMMY   * UNUSED CYCLE                    13
        DATA    DUMMY   * UNUSED CYCLE                    12
        DATA    BDCLK2  * COMPUTE ENERGY E(11)            11
        DATA    DUMMY   * UNUSED CYCLE                    10
        DATA    OUT     * COMMUNICATE WITH TMS7742         9
        DATA    DECODE  * DECODE/GET SCRAMBLED DIBIT       8
        DATA    DEMODB  * DEMODULATE IN MIDDLE OF BAUD     7
        DATA    DUMMY   * UNUSED CYCLE                     6
        DATA    AGCUPT  * UPDATE THE AGC EVERY 3RD BAUD    5
        DATA    DUMMY   * UNUSED CYCLE                     4
        DATA    BDCLK1  * COMPUTE ENERGY E(3)              3
        DATA    DUMMY   * UNUSED CYCLE                     2
        DATA    DUMMY   * UNUSED CYCLE                     1
        DATA    DUMMY   * UNUSED CYCLE                     0
```

Task 3 (BDCLK1) calculates the signal energy E(3) (PENRGY). Task 5 updates (once every three bauds) the automatic gain control value. Task 7 (DEMODB) implements the demodulation in the middle of the baud. Task 8 (DECODE) makes the channel decisions based on the demodulated (from Task 7) I and Q values, and decodes the decisions to obtain the scrambled dibits. Task 9 (OUT) performs the data exchange between the TMS32010 and the TMS7742. Task 11 calculates the signal energy E(11) (ENRGY). The TMS32010 code used to drive the table of the modem receiver tasks is shown below.

```
*
* RECEIVER TASK SEQUENCE DRIVER ROUTINE
*
        LAC     SAMPLE  * DECREMENT THE SAMPLE COUNT
        SUB     ONE     * TO CHECK FOR END OF BAUD
        BGEZ    OVRSAM  * IF NOT, THEN SKIP COUNT RESET
        LACK    15      * RESTART THE SAMPLE COUNTER AT 15
```

```
OVRSAM   SACL  SAMPLE   * SAVE NEW COUNT VALUE
         LACK  TSKSEQ   * GET ADDRESS OF TOP OF TABLE
         ADD   SAMPLE   * ADD IN OFFSET
         TBLR  TEMP     * GET THE PROGRAM ADDRESS
         LAC   TEMP     * FOR THE TASK CALL
         CALA           * EXECUTE THE APPROPRIATE TASK
```

Initially, the sample count (SAMPLE) contains the task number of the previous task performed. This number is decremented so that the next task in the sequence is performed. If the sample count becomes negative, it is reset to 15. The sample count is then added to the address of the top of the task table (TSKSEQ). With the use of the TBLR instruction, the table entry is transferred to the data memory. Each table entry is the address of the subroutine task to be performed. Using the CALA instruction, the equivalent of the 'computed GOTO' used in FORTRAN, the program control transfers to the selected subroutine. For a 9.6-kHz sampling rate, the TMS32010 with a 200-ns cycle time has 512 cycles available to implement each of these tasks. This number of cycles is more than enough since the worst-case task takes approximately 300 cycles. Also, since only 6 out of the 16 tasks are used, 10 more tasks are available for the designer to incorporate additional functions such as an adaptive equalizer, scrambling/descrambling, and synchronous-to-asynchronous and asynchronous-to-synchronous conversions.

The algorithm of adjusting the baud clock based on the filtered error signal $e'_b$ (BEROUT) is the same as the one described earlier for the unfiltered error signal $e_b$ (BERROR), and is summarized below.

$$\begin{aligned} e'_b > 0 \quad & \text{delay local baud clock} \\ e'_b < 0 \quad & \text{advance local baud clock} \end{aligned} \tag{43}$$

The advance or delay of the baud clock is implemented by changing the sample count (SAMPLE) appropriately. In the case of delaying the clock, the middle of the local baud clock (sample 7) occurs earlier than the middle of the incoming baud. Geometrically, sample 7 is located on the left side of the 'energy hill' of Figure 20 instead of at the top. If the sample count does not change, then 16 samples later, sample 7 of the next local baud will again be located on the left side of the 'energy hill' of the next incoming baud. Therefore, the sample count must be decremented by one. Instead of 16 samples, the middle of the next baud is taken to be 17 samples later. Hopefully then, the middle of the local baud is on or at least closer to the top of the 'energy hill.'

The case of advancing the clock is similar, except that the sample count is incremented by one, and thus the middle of the next baud is taken 15 samples after the middle of the current baud.

### *Clock Recovery Threshold*

One more issue, the clock recovery threshold, is associated with the alignment of the baud clock. Since there is a finite number of samples in each baud interval, the

baud clock has a finite resolution. Therefore, if the middle of the local baud (sample 7) is within one sample of the middle of the incoming baud, no correction must be applied. A threshold can be used so that corrections are applied only if the magnitude of the filtered error signal is greater than the threshold value. An initial estimate of this threshold is obtained by computing the magnitude of the error signal that corresponds to a one-sample change in the local baud clock. Consider the effect of a one-sample change in Figure 20. The middle of the local baud clock E(7) is translated to E(6) (or E(8)); E(3) is translated to E(2) (or E(4)); and E(11) is to E(10) (or E(12)). Therefore, a one sample change results in an error signal $e_b$ given by (41) of magnitude $\delta$ as indicated in Figure 20. Approximating the 'energy hill' with the positive half of a sine wave (see Figure 20), results in $\delta = 0.12$. This would be the threshold if the gain of the clock recovery loop filter were unity. For DC, the gain of this filter is (see equation (37))

$$G_2 = |H_2|\max = \frac{B_2}{1 - A_2} \qquad \text{where } 0 < A_2 < 1$$

The value chosen for the coefficient $B_2$ (BPLL2) is 0.0024 or >50 in Q15 format. After the receiver is locked-on to the incoming carrier, the coefficient $A_2$ (BPLL1) is 0.91. The gain $G_2$ of the loop filter is computed to be $G_2 = 0.026$.

The gain $G_2$ results in an 'effective threshold' of $\delta = 0.00312$. This corresponds to >33 in Q14 format used for the clock recovery threshold by designer's choice. However, this is just an initial estimate since the mathematical model used is only an approximation. After this estimate was obtained, the final value of the clock recovery threshold (TRSHD2), >8, was determined by trial and error.

The calculation of the thresholds for both the clock and carrier recoveries was performed based on the DC gain of the loop filters. A reason why the calculated thresholds are greater than those obtained by trial and error is that the filter gain is maximum at DC.

Just as in the carrier recovery, a two-level correction is used for the baud clock. If the magnitude of the error signal is less than the threshold, no correction is applied. If the magnitude of the error signal (BERROUT) is greater than the threshold (TRSHD2) but less than twice the threshold, the baud clock is advanced or delayed by one sample. If the magnitude of the error is greater than twice the threshold, then the baud clock is adjusted by two samples.

## Functions Implemented in the TMS7742

The Texas Instruments TMS7742 is a microcomputer with an on-chip UART and 4K bytes of internal EPROM. It was included in the modem design to increase its flexibility and upgradability. With the use of the TMS7742, both serial and parallel interfaces with the DTE can be efficiently implemented. The TMS7742 can also perform some of the modem functions, thus allowing the TMS32010 to do more complicated tasks. This flexibility allows the hardware design to be upgradable to 2400-bps splitband modems

(V.22 bis). The TMS7742 acts as a modem controller and performs the asynchronous-to-synchronous and synchronous-to-asynchronous data conversions. It also scrambles the data from the DTE and descrambles the decoded dibits received from the TMS32010 before sending them to the DTE. The TMS7742 code is given in Appendix E.

### Asynchronous-to-Synchronous and Synchronous-to-Asynchronous Conversions

Asynchronous data received from the DTE may include a start bit, seven or eight data bits, and one or more stop bits. When the DTE is not sending any data, the modem must still continue to transmit scrambled marks. Even though the DTE can send faster than 1200 bits per second, the modem must transmit only 1200 bits per second to the telephone line. This means that the modem must delete some of the bits received from the DTE. The Bell 212A protocol permits deleting one stop bit every nine characters. The data received from the TMS32010 demodulator may have characters with a deleted stop bit. The TMS7742 must detect the deleted stop bit and add it to the character before sending it to the DTE. The TMS7742 assembles the descrambled dibits into a character, checks for missing stop bits, and adds the missing stop bit if detected. The speed of the UART is set to enable inserting one stop bit in every nine characters; i.e., when transmitting 10 bits per character, adding one bit in nine characters (a total of 90 bits) should not change the speed. Thus, the UART is set to 1/90th of a bit interval faster.

### Scrambler/Descrambler

The data that has been converted into synchronous dibits is scrambled using equation (2), which is repeated below.

$$d_S(n) = d(n) \text{ XOR } d_S(n-14) \text{ XOR } d_S(n-17)$$

The TMS7742 holds the previous 17 scrambler outputs in its internal registers and uses the XOR instruction to exclusively-OR the proper bits to generate the new scrambled output. After scrambling each bit, these registers are shifted by one and saved to provide the $(n-7)$ outputs for the next bit.

A similar routine is used to descramble the decoded data received from the TMS32010. The descrambling is performed using equation (3) repeated below.

$$d(n) = d_S(n) \text{ XOR } d_S(n-14) \text{ XOR } d_S(n-17)$$

## Performance

The performance of the modem implemented using the TMS32010 was evaluated using automatic modem testing equipment. A block diagram of this testing equipment is shown in Figure 21.

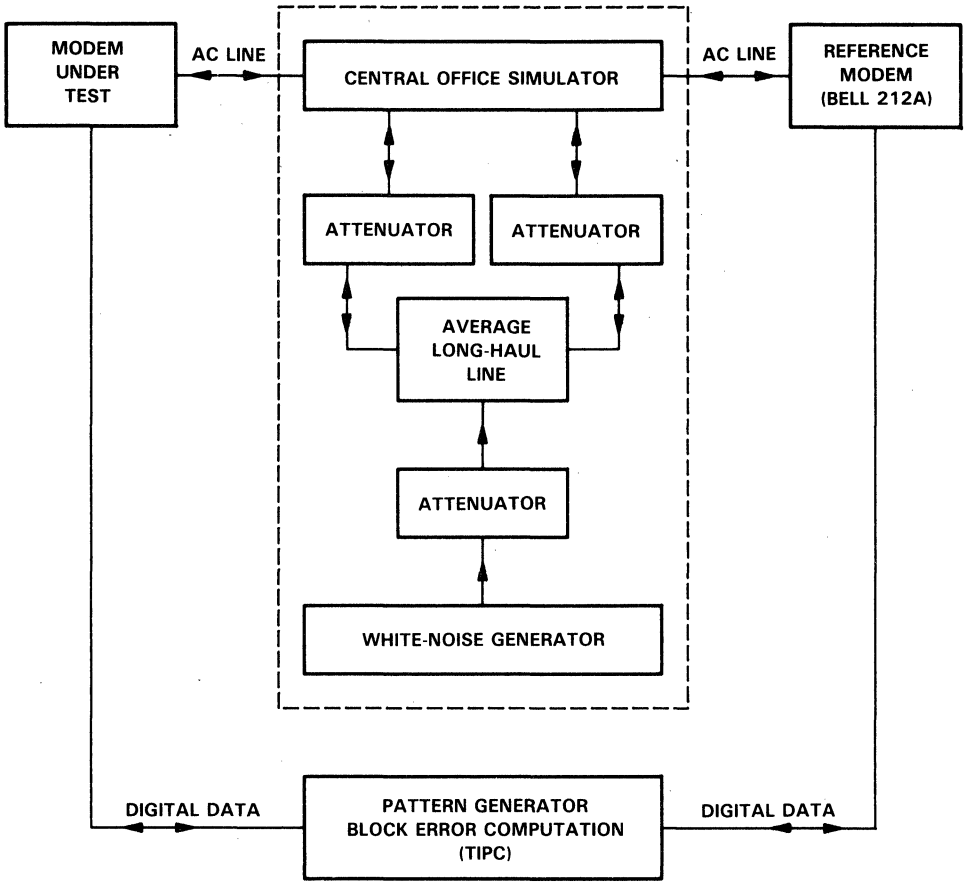
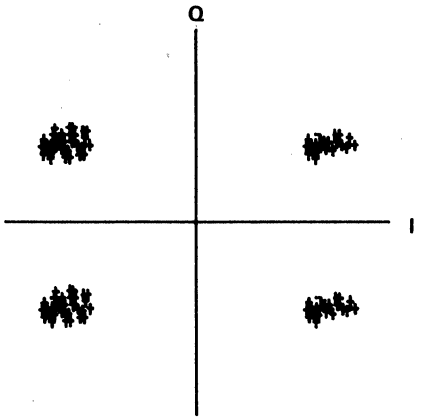


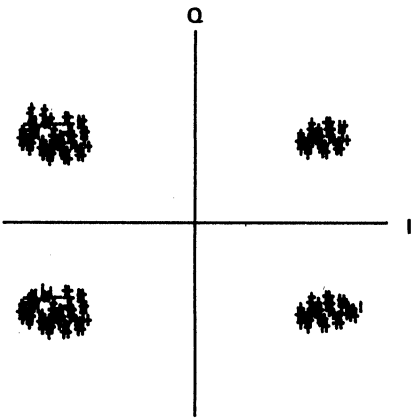

**Figure 21. Modem Testing Equipment**

The testing environment in Figure 21 provides a Central Office simulator, an average long-haul line simulator, and a C-notched white-noise generator. The attenuators provide signal-level and noise-level attenuation. The testing is performed under full-duplex and maximum data throughput conditions.

The average long-haul line effects are evident from the differences between the signal constellation diagrams of Figures 22(a) and 22(b). Figure 22(a) shows the signal constellation with the TMS32010 modem in the analog loop-back mode. Figure 22(b) shows the signal constellation with the TMS32010 modem operating over an average long-haul line at a 14-db signal-to-noise ratio. The presence of the average long-haul line results in a 'spreading' of the signal constellation points. This spreading implies a higher probability of error since the signal points used to make the decisions approach the decision boundaries.




(a) SIGNAL CONSTELLATION IN ANALOG LOOP-BACK MODE




(b) SIGNAL CONSTELLATION OVER AVERAGE LONG-HAUL LINE

**Figure 22. Signal Constellation Diagrams**

Referring to Figure 21, the Texas Instruments Professional Computer generates random characters. These characters are sent to the reference modem and the modem under testing. The modems transmit the characters they receive to each other, and each modem sends the characters received to the Professional Computer. The computer then compares the received characters with the ones originally created to determine the error rate. The error rate is determined in terms of percent error-free blocks. Each block consists of 512 characters (5120 bits) and is considered to be error-free only if all of the bits in the block are received with no error.

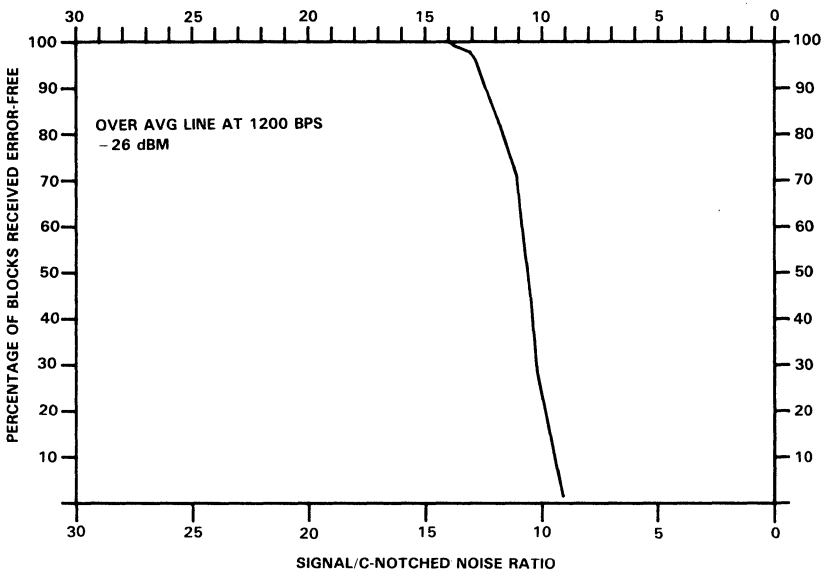
In all the tests performed, the Bell 212A modem was the reference modem configured in the answer mode. The reason for this is that only an originate

TMS32010-based modem is implemented. The answer mode is not included because, as mentioned in the "Introduction," this is beyond the purpose of this report. To incorporate the answer mode, two tables must be added in the TMS32010 code presented in Appendix D. The first table should contain the coefficients of the two receiver input bandpass filters with a passband centered around 1200 Hz. The second table should contain the increments used by the sine-table driver routine so that a 2400-Hz carrier is generated for the transmitter and a 1200-Hz carrier is generated for the receiver. When the TMS7742 configures the TMS32010 in the answer mode, the filter coefficients and the sine-table increments can be transferred from the program memory to the data memory with the use of the TBLR instruction. No performance difference is expected between the answer and originate modes.

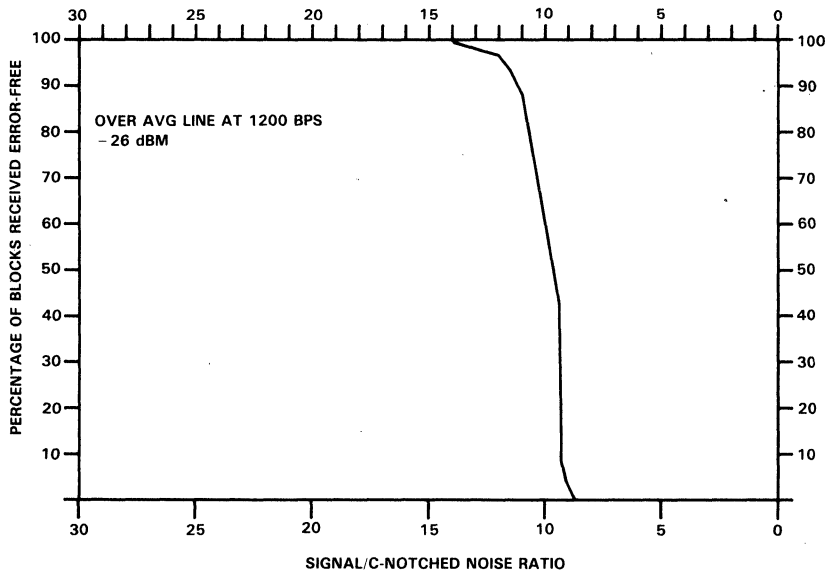
In Figure 23, the vertical axis indicates the percentage of blocks received error-free and the horizontal axis is the signal-to-noise ratio in db. The percentage of error-free blocks is calculated at each signal-to-noise ratio level (30, 29, 28,...) based on the number of error-free blocks received out of 1024 transmitted. All tests were performed at a $-26$ dbm (0.1 V) signal level. Figure 23(a) shows the test results with the TMS32010-based modem as the modem under testing. The vertical axis of Figure 23(a) is the percentage of blocks received error-free by the Bell modem. Figure 23(b) shows the results when the AT&T Dataphone II is used instead of the TMS32010-based modem.

Since the Bell modem is used as a reference modem, the above results indicate how well the transmitters of the TMS32010-based modem and the AT&T modem are performing. From Figures 23(a) and 23(b), it can be seen that for both the TMS32010 and AT&T modems, block errors start occurring at a signal-to-noise ratio of approximately 13 db and that the curve corresponding to the TMS32010 modem falls slightly faster. Therefore, the performance of both modem transmitters is approximately the same with the AT&T transmitter performing slightly better than the TMS32010 transmitter. Figure 24(a) shows the percentage of blocks received error-free by the TMS32010-based modem. The Bell modem (reference modem) is used to transmit these blocks. Figure 24(b) shows the percentage of blocks received error-free by the AT&T modem with the Bell modem transmitting.

It can be seen that the AT&T receiver performs approximately 2 db better than the TMS32010 receiver. The performance of the TMS32010 modem receiver could be improved with the inclusion of more filter taps in the receiver input bandpass filters.

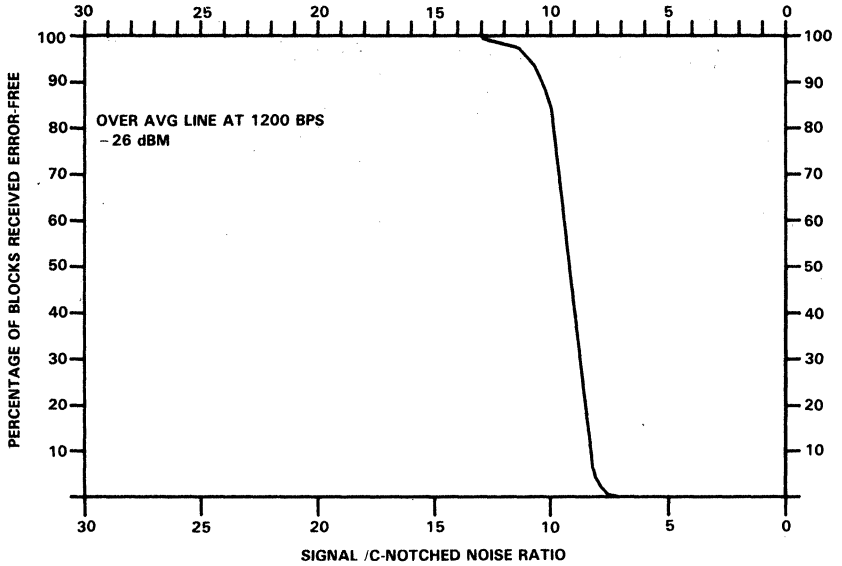



(a) PERCENTAGE OF BLOCKS RECEIVED ERROR-FREE BY THE BELL 212A MODEM VS. SNR WITH THE TMS32010 MODEM ORIGINATING

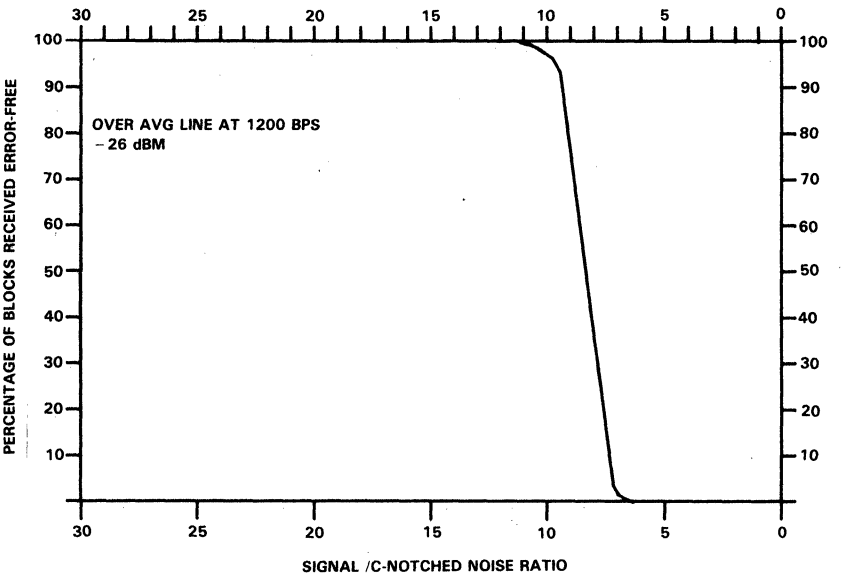



(b) PERCENTAGE OF BLOCKS RECEIVED ERROR-FREE BY THE BELL 212A MODEM VS. SNR WITH THE AT&T MODEM ORIGINATING

**Figure 23. Performance of TMS32010 and AT&T Modem Transmitters**




(a) PERCENTAGE OF BLOCKS RECEIVED ERROR-FREE BY THE TMS32010 MODEM VS. SNR WITH THE BELL MODEM TRANSMITTING




(b) PERCENTAGE OF BLOCKS RECEIVED ERROR-FREE BY THE AT&T MODEM VS. SNR WITH THE BELL MODEM TRANSMITTING

**Figure 24. Performance of TMS32010 and AT&T Modem Receivers**

## Other Implementation Considerations

The implementation approach of the Bell 212A/V.22 modem presented in the previous sections is not unique. There are other and possibly more efficient ways of implementing the modem.

Drastic reduction of the hardware cost results from the use of a codec for the A/D and D/A conversions instead of the 12-bit linear A/D and D/A converters used in this implementation. This approach becomes even more attractive with the use of the TMS32011 digital signal processor in place of the TMS32010. The TMS32011 is a microcomputer (no external memory expansion) having the same architecture as the TMS32010 with the additional feature of containing the necessary logic for interfacing to a codec. In this implementation, the necessary input bandpass filtering for the modem receiver can be performed with an AMI S35212A analog filter chip. The modem hardware block diagram of this implementation is shown in Figure 25.

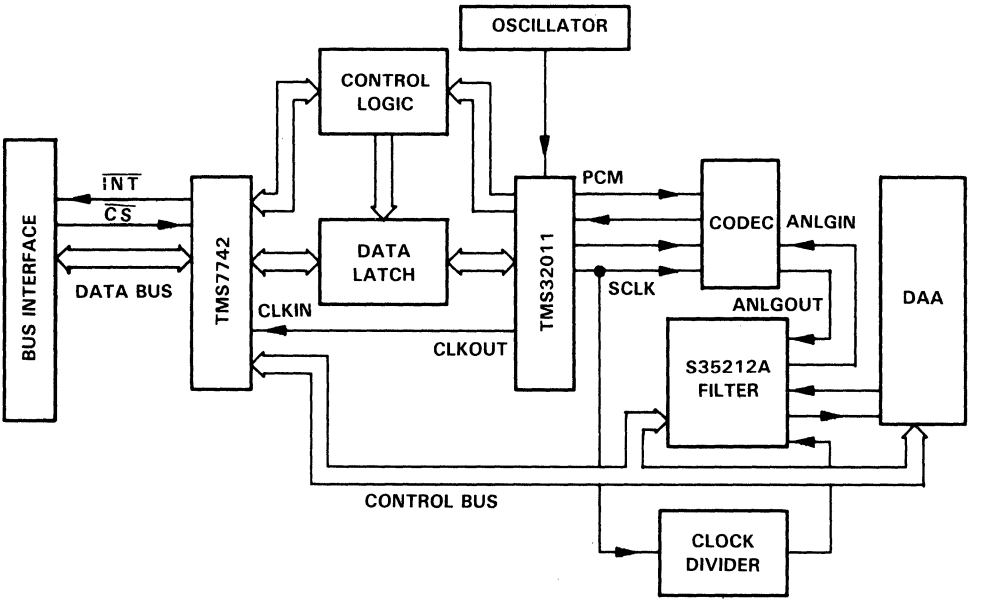



**Figure 25. Modem Hardware Block Diagram Using a Codec for the A/D and D/A Conversions**

If this approach is used, the receiver input has the configuration shown in Figure 26. The bandpass filtering is implemented in the analog domain and the Automatic Gain Control and Hilbert Transformer Pair implemented in the digital domain inside the TMS32011. Implementing the bandpass filtering in the analog domain should save adequate program memory, data memory, and processing power to allow the design to be upgraded to the V.22 bis specification. If only the Bell 212A is of interest, the bandpass filtering could be performed digitally within the TMS32011.

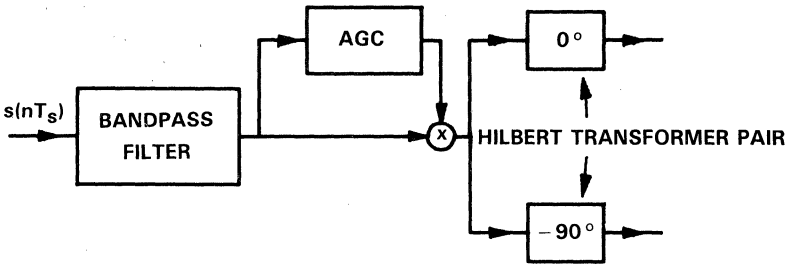



**Figure 26. Alternative Modem Receiver Input Configuration**

## Conclusions

This application report discussed the digital implementation of splitband modems using the TMS32010 general-purpose high-speed digital signal processor. The theory and implementation of the Bell 212A/V.22 full-duplex modem was covered in detail. With a modification of some of the functional blocks of the Bell 212A/V.22, 2400-bps splitband modems (V.22 bis) can be implemented.

Modems are sophisticated devices, consisting of many functional blocks. This implies the need of special features for the microprocessor to be used. The TMS32010 with a 200-ns cycle, an on-board single-cycle multiplier, and a special instruction set tailored for digital signal processing is able to implement the modem functional blocks (see Table 5) with approximately 60-percent use of the available processing power. The modem program utilizes 103 words of data memory out of the 144 words available. This corresponds to approximately 71 percent of the data memory. The program also utilizes 954 words of program memory out of the 1536 words available, corresponding to approximately 62 percent of the on-chip program memory. Therefore, the use of the full-speed off-chip memory feature of the TMS32010 was not utilized. Since a large portion of the power of the TMS32010 is still available, additional functions, such as an adaptive equalizer and the Data Encryption Standard (DES)[1], can be implemented with the inclusion of new code. With a 6-percent loading of the TMS32010, the DES can provide secure communication between 1200-bps full-duplex modems.

The TMS32010 is one of many digital signal processors in the TMS320 family. The flexibility and processing power of the TMS320 family provide high performance, high reliability, and cost-effective solutions for medium- and high-speed modems.

## References


1. P.E. Papamichalis and J. Reimer, "Implementation of the Data Encryption Standard with the TMS32010," *Digital Signal Processing Applications with the TMS320 Family*, Texas Instruments (1986).
2. C.F. Foschini, R.D. Gitlin, and S.B. Weinstein, "On the Selection of a Two-Dimensional Signal Constellation in the Presence of Phase Jitter and Gaussian Noise," *Bell System Technical Journal*, Vol. 52, 927-965 (July-August 1973).
3. P.J. Van Gerwen, N.A.M. Verhoeckx, H.A. Van Essen, and F.A.M. Snijders, "Microprocessor Implementation of High Speed Data Modems," *IEEE Trans. on Communications*, Vol. COM-25, No. 2, 238-250 (February 1977).
4. M.J. Di Toro, "Communication in Time Frequency Spread Using Adaptive Equalization," *Proceedings of the IEEE*, Vol. 56, No. 10, 1653-1679 (October 1968).
5. "Data Communications Using Voice Band Private Line Channels," *Bell Systems Technical Reference*, No. 41004 (1973).
6. F.M. Gardner, *Phaselock Techniques*, John Wiley & Sons (1979).
7. M.K. Simon and J.D. Smith, "Carrier Synchronization and Detection of QASK Signal Sets," *IEEE Trans. on Communications*, Vol. COM-22, 98-106 (February 1974).
8. W.C. Lindsey and M.K. Simon, "Carrier Synchronization and Detection of Polyphase Signals," *IEEE Trans. on Communications*, Vol. COM-20, 441-454 (June 1972).
9. H.L. Van Trees, *Detection, Estimation and Modulation Theory*, John Wiley and Sons (1968).
10. *TMS7742 Data Sheet*, Texas Instruments (1985).
11. *TMS7000 Family Data Manual*, Texas Instruments (1986).
12. M. Schwartz, *Information Transmission, Modulation, and Noise*, McGraw-Hill (1970).
13. A. Lovrich and R. Simar, "Implementation of FIR/IIR Filters with the TMS32010/TMS32020," *Digital Signal Processing Applications with the TMS320 Family*, Texas Instruments (1986).
14. T.W. Parks and C.S. Burrus, *Digital Filter Design*, John Wiley and Sons (1987).
15. *TMS32010 User's Guide*, Texas Instruments (1983).
16. A. Papoulis, *The Fourier Integral and Its Applications*, McGraw-Hill (1962).
17. D. Garcia, "Precision Digital Sine-Wave Generation with the TMS32010," *Digital Signal Processing Applications with the TMS320 Family*, Texas Instruments (1986).
18. H. Stark and F.B. Tuteur, *Modern Electrical Communications*, Prentice-Hall (1979).

# Appendix A
# Derivation of Demodulator Structure Equations

The equations that describe the demodulator structure (see Figure 6) of the modem receiver are derived in this Appendix. The background material required for this derivation is presented first. The following discussion requires a basic knowledge of complex variables.

The baseband signal, at the output of the transmitter digital lowpass filters (see Figure 3), can be expressed as a complex value

$$c(nT_S) = I(nT_S) - j\,Q(nT_S) \tag{A-1}$$

For transmission through the telephone network, this signal is modulated to the voice frequencies. Modulation involves multiplication by a complex exponential.[18] The modulated signal is then given by

$$m(nT_S) = c(nT_S)\,e^{j\omega_c nT_S} \tag{A-2}$$

where $\omega_c$ is the carrier frequency. Substitution of (A-1) into (A-2), and the use of the identity

$$e^{j\omega_c nT_S} = \cos(\omega_c nT_S) + j\,\sin(\omega_c nT_S)$$

give

$$\begin{aligned} m(nT_S) = {} & \{I(nT_S)\cos(\omega_c nT_S) + Q(nT_S)\sin(\omega_c nT_S)\} \\ & + j\,\{I(nT_S)\sin(\omega_c nT_S) - Q(nT_S)\cos(\omega_c nT_S)\} \end{aligned} \tag{A-3}$$

The real and imaginary parts of (A-3) are later shown to be a Hilbert transform pair. Two signals are referred to as a Hilbert transform pair if they are related with a Hilbert transform. A Hilbert transform is implemented with a filter called a Hilbert transformer. The Hilbert transform pair property that relates the real and imaginary parts of (A-3) allows the transmission of the real part of (A-3) only. The imaginary part is recovered at the receiver by Hilbert transforming the incoming signal. Figure A-1 shows the spectrum of the complex baseband information $c(nT_S)$. Figure A-2 shows the spectrum after modulation by the complex exponential (see equation (A-2)). This is the spectrum of $m(nT_S)$. Figure A-3 shows the spectrum of the transmitted signal, i.e., the spectrum of the real part of $m(nT_S)$.

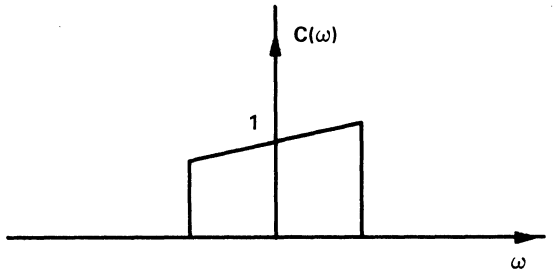



**Figure A-1. Spectrum of Complex Baseband Information**

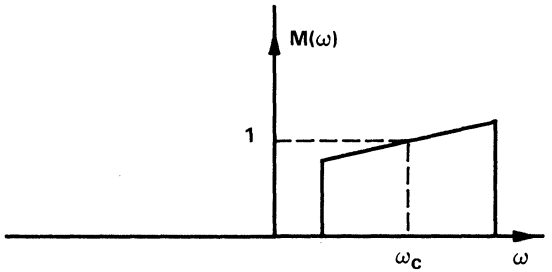



**Figure A-2. Spectrum after Modulation**

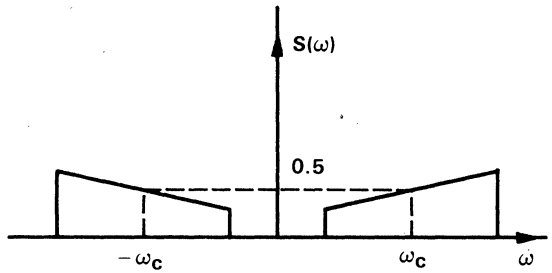



**Figure A-3. Transmitted Spectrum**

A Hilbert transformer is defined to be a filter with the transfer function[18]

$$H_t(\omega) = -e^{j\frac{\pi}{2}} \operatorname{sgn}(\omega) = -j \operatorname{sgn}(\omega) \quad \text{(A-4)} \qquad \text{(A-4)}$$

where sgn is the sign function defined by equation (29). The transfer function characteristics of the Hilbert transformer are shown in Figure A-4, where it is seen that the Hilbert transformer introduces a $-90$ degree phase shift for positive frequencies ($\omega > 0$), and a $+90$ degree phase shift for negative frequencies ($\omega < 0$).

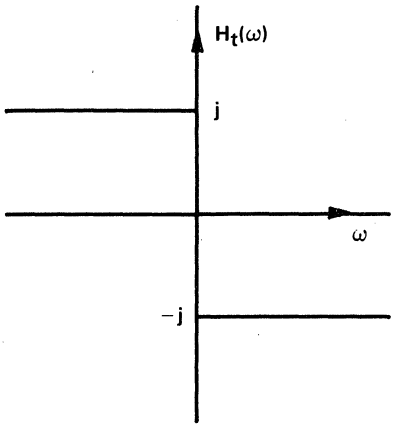



**Figure A-4. Hilbert Transformer Transfer Function**

The Hilbert transform pair relationship between the real and imaginary parts of (A-3) is discussed next. It is shown that the imaginary part of $m(nT_S)$ is the output of a Hilbert transformer with the input being the real part of $m(nT_S)$. The analysis is performed in the frequency domain where multiplication is replaced by convolution. Let $S(\omega)$ and $\hat{S}(\omega)$ be the Fourier transforms of the real and imaginary parts of $m(nT_S)$, respectively. Then (see equation (A-3))

$$S(\omega) = \frac{1}{2}\{I(\omega-\omega_C) + I(\omega+\omega_C)\} + \frac{j}{2}\{Q(\omega+\omega_C) - Q(\omega-\omega_C)\} \quad \text{(A-5)}$$

$$\hat{S}(\omega) = \frac{j}{2}\{I(\omega+\omega_C) - I(\omega-\omega_C)\} - \frac{1}{2}\{Q(\omega-\omega_C) + Q(\omega+\omega_C)\} \quad \text{(A-6)}$$

where $I(\omega)$ and $Q(\omega)$ are the Fourier transforms of $I(nT_S)$ and $Q(nT_S)$, respectively.

With $S(\omega)$ as the input to the Hilbert transformer (transfer function $H_t(\omega)$), the output in the frequency domain is given by

$$O(\omega) = S(\omega)\, H_t(\omega) = -j\, S(\omega)\, \mathrm{sgn}(\omega) \quad \text{(A-7)}$$

Substitution of (A-5) into (A-7) gives

$$O(\omega) = -j\left\{\frac{1}{2}[I(\omega-\omega_C) + I(\omega+\omega_C)] + \frac{j}{2}[Q(\omega+\omega_C) - Q(\omega-\omega_C)]\right\}\mathrm{sgn}(\omega) \quad \text{(A-8)}$$

Since for positive frequencies ($\omega > 0$),

$$\begin{aligned} I(\omega+\omega_C) &= 0 \\ Q(\omega+\omega_C) &= 0 \end{aligned} \quad \text{(A-9)}$$

and for negative frequencies ($\omega < 0$),

$$\begin{aligned} I(\omega - \omega_c) &= 0 \\ Q(\omega - \omega_c) &= 0 \end{aligned} \qquad \text{(A-10)}$$

equation (A-8) simplifies to

$$O(\omega) = \begin{cases} -\frac{j}{2} I(\omega - \omega_c) - \frac{1}{2} Q(\omega - \omega_c) & \text{where } \omega > 0 \\ \frac{j}{2} I(\omega + \omega_c) - \frac{1}{2} Q(\omega + \omega_c) & \text{where } \omega < 0 \end{cases} \qquad \text{(A-11)}$$

Substitution of (A-9) and (A-10) into (A-6) and comparison of the result with (A-11) shows that $S(\omega) = O(\omega)$.

Therefore, the real and imaginary parts of $m(nT_S)$ (see equation (A-3)) represent a Hilbert tranform pair. With $s(nT_S)$ and $\hat{s}(nT_S)$ denoting the real and imaginary parts of m(nTs), respectively, (A-3) can be written as

$$m(nT_S) = s(nT_S) + j\,\hat{s}(nT_S) \qquad \text{(A-12)}$$

At the receiver end, recovery of the imaginary part $\hat{s}(nT_S)$ involves Hilbert transforming the real part $s(nT_S)$ (incoming signal), as shown in Figure A-5.

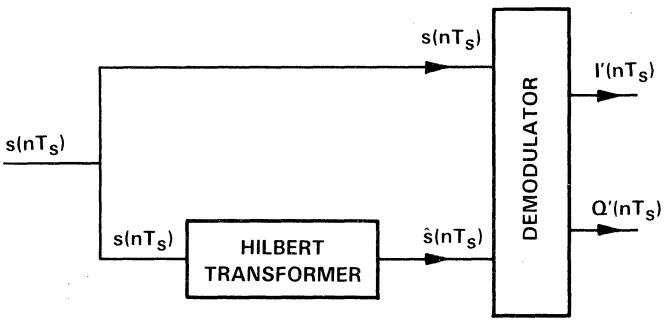



**Figure A-5. Recovery of Complex Information by Hilbert Transforming the Incoming Signal**

Consider the Fourier transform of (A-12)

$$\begin{aligned} M(\omega) &= S(\omega) + j\,\hat{S}(\omega) \\ &= S(\omega) + j\,\{-j\,S(\omega)\,\mathrm{sgn}(\omega)\} \\ &= S(\omega) + S(\omega)\,\mathrm{sgn}(\omega) \end{aligned}$$

Therefore,

$$M(\omega) = \begin{cases} 2\ S(\omega) & \text{where } \omega > 0 \\ 0 & \text{where } \omega < 0 \end{cases} \quad \text{(A-13)}$$

i.e., the spectrum of $m(nT_S)$ is zero for negative frequencies (see Figure A-2). If this property does not hold due to the use of a nonideal Hilbert transformer, harmonics appear at the output of the receiver demodulator (see Appendix B).

The equations that describe the receiver demodulator are derived next. The demodulator translates the recovered complex modulated information back to the baseband. This is accomplished by multiplying the passband information with a complex exponential.

$$c'(nT_S) = m(nT_S)\ e^{-j\omega_c nT_S} \quad \text{(A-14)}$$

where $c'(nT_S)$ is the recovered baseband signal, $m(nT_S)$ is the passband signal given by (A-12), and $\omega_c$ is the carrier frequency recovered at the receiver by the carrier recovery algorithm.

Substitution of (A-12) into (A-14) gives

$$\begin{aligned} c'(nT_S) = {} & \{s(nT_S)\cos(\omega_c nT_S) + \hat{s}\ (nT_S)\sin(\omega_c nT_S)\} \\ & + j\ \{\hat{s}(nT_S)\cos(\omega_c nT_S) - s(nT_S)\sin(\omega_c nT_S)\} \end{aligned} \quad \text{(A-15)}$$

The complex baseband information $c'(nT_S)$ is also given by (see equation (A-1) and Figure 5)

$$c'(nT_S) = I'(nT_S) - j\ Q'(nT_S) \quad \text{(A-16)}$$

Equating the real and imaginary parts of (A-15) to those of (A-16) results in

$$I'(nT_S) = s(nT_S)\cos(\omega_c nT_S) + \hat{s}(nT_S)\sin(\omega_c nT_S) \quad \text{(A-17)}$$

$$Q'(nT_S) = s(nT_S)\sin(\omega_c nT_S) - \hat{s}(nT_S)\cos(\omega_c nT_S) \quad \text{(A-18)}$$

Equations (A-17) and (A-18) describe the receiver demodulator of Figure 6.

# Appendix B

## Effects of Nonideal Hilbert Transformers

The effect of nonideal Hilbert Transformers in modem design is studied in this Appendix. The following discussion requires a basic knowledge of complex variables.

The nonideal Hilbert transformer characteristics differ from the ideal ones shown in Figure 28 and described by equation (A-4) in Appendix A. The phase shift introduced by the nonideal filter is not exactly 90 degrees. The transfer function characteristics of such a filter are given by

$$H'(\omega) = -e^{j\left(\frac{\pi}{2}+\alpha\right)} \operatorname{sgn}(\omega) = -j\, e^{j\alpha} \operatorname{sgn}(\omega) \tag{B-1}$$

where '$\alpha$' is a nonzero constant indicating the deviation from the ideal filter.

Consider the effect of a nonideal Hilbert transformer described by equation (B-1). The incoming signal $s(nT_S)$ is the real part of $m(nT_S)$. This signal is filtered by the nonideal Hilbert transformer to generate at the output a signal $\hat{s}'(nT_S)$ different from $\hat{s}(nT_S)$ (see Appendix A). With $\hat{S}'(\omega)$ as the Fourier transform of $\hat{s}'(nT_S)$, the output of the nonideal Hilbert transformer can be described in the frequency domain by

$$\hat{S}'(\omega) = H'(\omega)\, S(\omega) = -j\, e^{j\alpha} \operatorname{sgn}(\omega)\, S(\omega) \tag{B-2}$$

The complex signal at the input of the receiver demodulator is described by

$$m'(nT_S) = s(nT_S) + j\, \hat{s}'(nT_S) \tag{B-3}$$

The frequency-domain equivalent of (B-3) is

$$M'(\omega) = S(\omega) + j\, \hat{S}'(\omega) \tag{B-4}$$

Substitution of (B-2) into (B-4) gives

$$M'(\omega) = S(\omega) + e^{j\alpha} \operatorname{sgn}(\omega)\, S(\omega) \tag{B-5}$$

Equation (B-5) can be written as

$$M'(\omega) = \begin{cases} S(\omega)\, \{\, 1 + e^{j\alpha} \,\} & \text{where } \omega > 0 \\ S(\omega)\, \{\, 1 - e^{j\alpha} \,\} & \text{where } \omega < 0 \end{cases} \tag{B-6}$$

For a nonideal Hilbert transformer, the parameter '$\alpha$' is nonzero. This results in $M'(\omega)$ having nonzero components at negative frequencies as indicated by (B-6). The spectrum of the signal at the input of the receiver demodulator is shown in Figure B-1. Comparison of Figures A-2 and B-1 indicates that the effect of the nonideal Hilbert transformer is the generation of nonzero spectral components at negative frequencies.

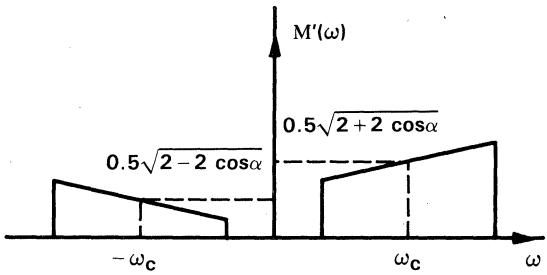



**Figure B-1. Effect of Nonideal Hilbert Transformer on the Spectrum of the Complex Signal at the Input of the Demodulator**

The effect of the receiver demodulator on the spectrum of Figure B-1 is shown in Figure B-2.

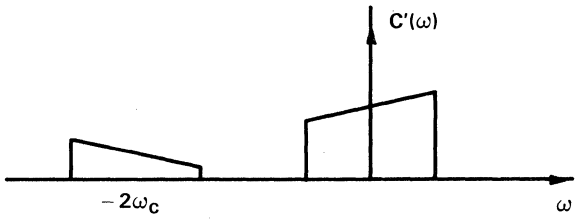



**Figure B-2. Effect of Nonideal Hilbert Transformer on the Spectrum of the Baseband Complex Signal**

Figure B-2 indicates that harmonics appear at the output of the demodulator. The frequency of these harmonics is twice the carrier frequency. Their elimination involves the use of lowpass filters at the output of the demodulator. These filters, however, introduce group delay and possibly phase delay effects that affect the carrier recovery and decision algorithms. Compensation for the lowpass filter side-effects results in a more complicated modem receiver design. Such nonideal Hilbert transformers are encountered in analog modems. This appendix has demonstrated one more advantage of a digital implementation of a modem using the TMS32010 digital signal processor.

# Appendix C
# Automatic Gain Control Table Generator Code

```
10 '     THIS PROGRAM GENERATES THE GAIN TABLE FOR THE AUTOMATIC
20 '     GAIN CONTROL ALGORITHM IN THE MODEM CODE
30 '
40 '     THE PROGRAM PROMPTS THE USER IN THE FOLLOWING MANNER:
50 '
60 '           AGC TABLE ADJUST FACTOR ?
70 '            This feature allows the AGC to gain to a level lower
80 '           than unity.  The entry for unity gain is 256, to set
90 '            the gain lower than unity enter the appropriate per-
100 '            centage of 256.
110 '
120 '          ENTER NAME OF OUTPUT FILE =
130 '            This prompt request the name of a MSDOS format file
140 '            name to store the generated table.
150 '
160 '          TABLE LENGTH =
170 '            This feature allows the user to generate different
180 '            length AGC tables.  This allows the accuracy of the
190 '            table to vary by the number of entries.  The number
200 '            of entries is tied to the number of bits used in the
210 '            table lookup.  In the modem algorithm six bits were
220 '            used in the lookup, therefore the table length will be
230 '            64 words.
240 '
250 '     THE TABLE GENERATED WILL INCLUDE DESCRIPTIVE COMMENTS AND WILL
260 '     BE IN A FORM READY TO BE ADDED DIRECTLY INTO THE ASSEMBLY CODE
270 '     FOR AN ALGORITHM.  SINCE THE AGC SOFTWARE SHIFTS THE LOOKUP
280 '     VALUE TO THE MOST SIGNIFICANT BIT THE FIRST HALF OF THE AGC TABLE
290 '     (THE LESS ACCURATE HALF) WILL NOT BE USED.  THEREFORE THE USER
300 '     CAN DELETE THE FIRST HALF AND SAVE A CONSIDERABLE AMOUNT OF PROGRAM
310 '     MEMORY SPACE.
320 '
330 '     THIS PROGRAM WAS WRITTEN BY PETER EHLIG FOR USE ON A
340 '       TEXAS INSTRUMENTS PROFESSIONAL COMPUTER
350 '     THE CODE TO MY KNOWLEDGE IS WRITTEN IN STANDARD MS-BASIC AND
360 '     SHOULD OPERATE ON ANY MSDOS SYSTEM.
370 '
380 PRINT 'PROGRAM STARTED"
390 DIM TBLD(500),HTB$(500)
400 OPEN "LPT1:" FOR OUTPUT AS #1
410 INPUT "AGC TABLE ADJUSTMENT FACTOR ? ",GAINADJ
420 INPUT "ENTER NAME OF OUTPUT FILE = ",OUTFILE$
430 OPEN OUTFILE$ FOR OUTPUT AS #3
440 PI = 3.1415927#
450 PI2 = PI * 2
460 INPUT "TABLE LENGTH = ",TBLEN
470 GOSUB 820  ' GENERATE TABLE HEADER
480 DELTA = 327681 / TBLEN
490 FOR I = 1 TO TBLEN
500 TBL = INT(32767 / (I * DELTA) * GAINADJ)
510 TBLD(I) = TBL
520 HTBL$ = HEX$(TBL)
530 HTB$(I) = HTBL$
540 GOSUB 690  ' DISPLAY RANGE ACCURACY  (OPTIONAL)
550 NEXT
560 GOTO 650
570 ' SAVE AGC TABLE TO DISK
580 PRINT#3, "       DATA ";
590 PRINT#3, USING ">\  \";HTB$(I);
600 PRINT#3, "              ";
610 TBLD1 = TBLD(I) / 256
620 PRINT#3, USING "###.#######";TBLD1
630 RETURN
640 ' END OF AGC TABLE SAVE ROUTINE
650 GOSUB 940  ' DISPLAY SECOND LEVEL LOOKUP
660 GOSUB 880  ' GENERATE TABLE TERMINATION COMMENTS
670 PRINT "PROGRAM FINISHED"
680 END
```

```
690 ' THIS ROUTINE DISPLAYS INFORMATION ABOUT THE RANGE
700 ' ACCURACY OF EACH STEP OF THE TABLE
710 TBLRL = (I - 1) * DELTA - 256
720 IF TBLRL < 0 THEN TBLRL = 0
730 SH1$ = HEX$(TBLRL)
740 SH1A$ = HEX$(TBLRL * TBL / 256)
750 TBLRH = (I - 1) * DELTA + 255
760 SH2$ = HEX$(TBLRH)
770 SH2A$ = HEX$(TBLRH * TBL / 256)
780 PRINT I;TBL;HTBL$;" ";SH1$;" ";SH1A$;" ";SH2$;" ";SH2A$
790 ' PRINT#1,I;TBL;HTBL$;" ";SH1$;" ";SH1A$;" ";SH2$;" ";SH2A$
800 RETURN
810 ' END OF RANGE INFORMATION
820 ' THE ROUTINE GENERATES THE HEADER COMMENTS FOR THE TABLE
830 PRINT#3,"*************************************************************"
840 PRINT#3,"AGCTBL EQU  $              AGC TABLE LENGTH = ";
850 PRINT#3, USING "###";TBLEN
860 RETURN
870 ' END OF HEADER ROUTINE
880 ' THIS ROUTINE GENERATES THE TABLE TERMINATION COMMENTS
890 PRINT#3,"*************************************************************"
900 PRINT#3, "       PAGE"
910 CLOSE
920 RETURN
930 ' END OF TERMINATOR ROUTINE
940 ' TRY SECOND LEVEL LOOKUP
950 DELTA1 = DELTA * 8
960 FOR I = 1 TO 64
970 GOSUB 570  ' SAVE AGC TABLE TO DISK
980 TBLRL = (I - 1) * DELTA - 256
990 IF TBLRL < 0 THEN TBLRL = 0
1000 TBLRH = (I - 1) * DELTA + 255
1010 SH1$ = HEX$(TBLRL)
1020 SH2$ = HEX$(TBLRH)
1030 GOSUB 1100 ' CALCULATE ACCURACY STEPS
1040 SH1A$ = HEX$(TBLRL * TBLD(TBLR1) / SHF1)
1050 SH2A$ = HEX$(TBLRH * TBLD(TBLR2) / SHF1)
1060 PRINT I;TBL;HTBL$;" ";SH1$;" ";SH1A$;" ";SH2$;" ";SH2A$;TBLR1;TBLR2;SHF1
1070 ' PRINT#1,I;TBL;HTBL$;" ";SH1$;" ";SH1A$;" ";SH2$;" ";SH2A$;TBLR1;TBLR2;SHF
1
1080 NEXT
1090 RETURN
1100 'TABLE LOOKUP SHIFTER
1110 TBLEV = TBLRH - 4096
1120 IF TBLEV > 0 GOTO 1180
1130 TBLEV = TBLEV + 2048
1140 IF TBLEV > 0 GOTO 1220
1150 TBLEV = TBLEV + 1024
1160 IF TBLEV > 0 GOTO 1260
1170 GOTO 1300
1180 TBLR1 = I
1190 TBLR2 = I
1200 SHF1 = 256
1210 RETURN
1220 TBLR2 = FIX(TBLRH / 64) + 1
1230 TBLR1 = FIX(TBLRL / 64) + 1
1240 SHF1 = 32
1250 RETURN
1260 TBLR2 = FIX(TBLRH / 32) + 1
1270 TBLR1 = FIX(TBLRL / 32) + 1
1280 SHF1 = 16
1290 RETURN
1300 TBLR2 = FIX(TBLRH / 16) + 1
1310 TBLR1 = FIX(TBLRL / 16) + 1
1320 SHF1 = 8
1330 RETURN
```

The following 64-point table was generated using the 124 for the AGC table adjust factor.

```
*************************************************************
AGCTBL EQU  $              AGC TABLE LENGTH =  64
       DATA >1EFF                  30.9961000
       DATA >F7F                   15.4960900
       DATA >A55                   10.3320300
       DATA >7BF                    7.7460940
       DATA >633                    6.1992190
       DATA >52A                    5.1640630
       DATA >46D                    4.4257820
       DATA >3DF                    3.8710940
       DATA >371                    3.4414060
       DATA >319                    3.0976560
       DATA >2D1                    2.8164060
       DATA >295                    2.5820310
       DATA >262                    2.3828130
       DATA >236                    2.2109380
       DATA >211                    2.0664060
       DATA >1EF                    1.9335940
       DATA >1D2                    1.8203130
       DATA >1B8                    1.7187500
       DATA >1A1                    1.6289060
       DATA >18C                    1.5468750
       DATA >179                    1.4726560
       DATA >168                    1.4062500
       DATA >159                    1.3476560
       DATA >14A                    1.2890630
       DATA >13D                    1.2382810
       DATA >131                    1.1914060
       DATA >125                    1.1445310
       DATA >11B                    1.1054690
       DATA >111                    1.0664060
       DATA >108                    1.0312500
       DATA >FF                     0.9960938
       DATA >F7                     0.9648438
       DATA >F0                     0.9375000
       DATA >E9                     0.9101562
       DATA >E2                     0.8828125
       DATA >DC                     0.8593750
       DATA >D6                     0.8359375
       DATA >D0                     0.8125000
       DATA >CB                     0.7929688
       DATA >C6                     0.7734375
       DATA >C1                     0.7539063
       DATA >BC                     0.7343750
       DATA >B8                     0.7187500
       DATA >B4                     0.7031250
       DATA >B0                     0.6875000
       DATA >AC                     0.6718750
       DATA >A8                     0.6562500
       DATA >A5                     0.6445313
       DATA >A1                     0.6289063
       DATA >9E                     0.6171875
       DATA >9B                     0.6054688
       DATA >98                     0.5937500
       DATA >95                     0.5820313
       DATA >92                     0.5703125
       DATA >90                     0.5625000
       DATA >8D                     0.5507813
       DATA >8B                     0.5429688
       DATA >88                     0.5312500
       DATA >86                     0.5234375
       DATA >84                     0.5156250
       DATA >82                     0.5078125
       DATA >7F                     0.4960938
```

```
       DATA >7D                    0.4882813
       DATA >7B                    0.4804688
*************************************************************
       PAGE
```

```
10 '
20 '    This program generates sine table in a format compatible
30 '    to the 320 assembler.  This allows the user to generate
40 '    any length sine table and this program will calculate the
50 '    table entries, configure them in a format compatible to
60 '    the assembler, and document the code.
70 '
80 '    The program prompts the user in the following manner:
90 '
100 '          ENTER NAME OF OUTPUT FILE =
110 '            This prompt request the name of a MSDOS format file
120 '            name to store the generated table.
130 '
140 '          TABLE LENGTH =
150 '            This feature allows the user to select the length of
160 '            the sine table to be generated and therefore the
170 '            accuracy of the table steps.
180 '
190 '
200 '    This program was written by Peter Ehlig for use on a
210 '      Texas Instruments Professional Computer
220 '    The code to my knowledge is written in standard MS-BASIC and
230 '    should operate on any MSDOS system.
240 '
250 PRINT 'PROGRAM STARTED"
260 INPUT "ENTER NAME OF OUTPUT FILE = ",OUTFILE$
270 OPEN OUTFILE$ FOR OUTPUT AS #3
280 PI = 3.1415927#
290 PI2 = PI * 2
300 INPUT "TABLE LENGTH = ",TBLEN
310 DELTA = PI2 / TBLEN
320 INDX1 = -DELTA
330 NETDEG = 360 / TBLEN
340 PRINT#3,"*************************************************************"
350 PRINT#3,"SINE    EQU  $              SINE TABLE LENGTH = ";
360 PRINT#3, USING "###";TBLEN
370 FOR I = 1 TO TBLEN
380 INDX1 = INDX1 + DELTA
390 TBL = SIN(INDX1)
400 HTBL$ = HEX$(TBL*16384)
410 RADS = INDX1 / PI
420 DEGR = NETDEG * (I - 1)
430 PRINT#3, "        DATA ";
440 PRINT#3, USING ">\   \";HTBL$;
450 PRINT#3, "          ANGLE = ";
460 PRINT#3, USING "###.####";DEGR;
470 PRINT#3, "  SINE = ";
480 PRINT#3, USING "#.######";TBL
490 NEXT
500 PRINT#3,"*************************************************************"
510 PRINT#3, "        PAGE"
520 CLOSE
530 PRINT "PROGRAM FINISHED"
540 END
```

```
*************************************************************
SINE    EQU  $              SINE TABLE LENGTH =  32
        DATA >0             ANGLE =   0.0000  SINE = 0.000000
        DATA >C7C           ANGLE =  11.2500  SINE = 0.195090
        DATA >187E          ANGLE =  22.5000  SINE = 0.382683
        DATA >238E          ANGLE =  33.7500  SINE = 0.555570
        DATA >2D41          ANGLE =  45.0000  SINE = 0.707107
        DATA >3537          ANGLE =  56.2500  SINE = 0.831470
        DATA >3B21          ANGLE =  67.5000  SINE = 0.923880
        DATA >3EC5          ANGLE =  78.7500  SINE = 0.980785
        DATA >4000          ANGLE =  90.0000  SINE = 1.000000
        DATA >3EC5          ANGLE = 101.2500  SINE = 0.980785
        DATA >3B21          ANGLE = 112.5000  SINE = 0.923880
        DATA >3537          ANGLE = 123.7500  SINE = 0.831470
        DATA >2D41          ANGLE = 135.0000  SINE = 0.707107
        DATA >238E          ANGLE = 146.2500  SINE = 0.555570
        DATA >187E          ANGLE = 157.5000  SINE = 0.382683
        DATA >C7C           ANGLE = 168.7500  SINE = 0.195090
        DATA >0             ANGLE = 180.0000  SINE = -.000000
        DATA >F384          ANGLE = 191.2500  SINE = -.195091
        DATA >E782          ANGLE = 202.5000  SINE = -.382684
        DATA >DC72          ANGLE = 213.7500  SINE = -.555571
        DATA >D2BF          ANGLE = 225.0000  SINE = -.707107
        DATA >CAC9          ANGLE = 236.2500  SINE = -.831470
        DATA >C4DF          ANGLE = 247.5000  SINE = -.923880
        DATA >C13B          ANGLE = 258.7500  SINE = -.980786
        DATA >C000          ANGLE = 270.0000  SINE = %-1.000000
        DATA >C13B          ANGLE = 281.2500  SINE = -.980785
        DATA >C4DF          ANGLE = 292.5000  SINE = -.923879
        DATA >CAC9          ANGLE = 303.7500  SINE = -.831469
        DATA >D2BF          ANGLE = 315.0000  SINE = -.707106
        DATA >DC72          ANGLE = 326.2500  SINE = -.555569
        DATA >E782          ANGLE = 337.5000  SINE = -.382682
        DATA >F384          ANGLE = 348.7500  SINE = -.195089
*************************************************************
        PAGE
```

# Appendix D
# TMS32010 Source Code

```
*
*************************************************************
***                 -  DSP MODEM PROGRAM  -               ***
*************************************************************
*** THIS CODE IMPLEMENTS A BELL 212A / V.22 MODEM         ***
*** ON THE TMS32010.                                      ***
***                                                       ***
*** SCRAMBLING AND DESCRAMBLING  ARE IMPLEMENTED          ***
*** ON THE TMS7742.                                       ***
*************************************************************
        IDT     'TASK6212'
        OPTION  XREF
        AORG    0
        B       START
********-----------------------------------------------********
*****       DATA MEMORY USED.                           *****
********-----------------------------------------------********
XDELTA  EQU  0           * SWAVE MACRO CARRIER RATE
XALPHA  EQU  1           * SWAVE MACRO CARRIER ANGLE
SINA    EQU  2           * XMIT SIN CARRIER MAGNETUDE
COSA    EQU  3           * XMIT COS CARRIER MAGNETUDE
ONE     EQU  4           * VALUE 1 HELD FOR MASKING
MASK1   EQU  5           * SWAVE MACRO TBL RANGE ADJ >7F
MASK2   EQU  6           * SWAVE MACRO TBL RANGE ADJ >7FFF
MASK3   EQU  7           * XMIT PHASE ENCODE MASK >0006
OFSET0  EQU  8           * SWAVE MACRO POINT TO COS TABLE
OFSET1  EQU  9           * XMIT POINT TO DIBIT ENCODE TABLE
XPTR    EQU  10          * XMIT POINT TO RAISED COS TABLE
CX0     EQU  11          * XMIT COEF FOR RAISED COS
CX1     EQU  12          * XMIT COEF FOR RAISED COS
CX2     EQU  13          * XMIT COEF FOR RAISED COS
XIBUF0  EQU  14          * XMIT STORE DATA FOR RAISED COS
XIBUF1  EQU  15          * XMIT STORE DATA FOR RAISED COS
XIBUF2  EQU  16          * XMIT STORE DATA FOR RAISED COS
XQBUF0  EQU  17          * XMIT STORE DATA FOR RAISED COS
XQBUF1  EQU  18          * XMIT STORE DATA FOR RAISED COS
XQBUF2  EQU  19          * XMIT STORE DATA FOR RAISED COS
XIOUT   EQU  20          * XMIT HOLD FILTERED I VALUE
XQOUT   EQU  21          * XMIT HOLD FILTERED Q VALUE
XMTOUT  EQU  22          * XMIT HOLD FOR TRANSMIT OUTPUT
XOLDPH  EQU  23          * XMIT HOLD LAST PHASE
XNEWPH  EQU  24          * XMIT HOLD NEW PHASE
RDIBIT  EQU  25          * DECODED DIBIT
INDXPH  EQU  26          * XMIT POINT TO PHASE ENCODE TABLE
XDIBIT  EQU  27          * XMIT DIBIT ISOLATION MASK
PLUS1   EQU  28          * +1 Q12 >FFF & MASK VALUE
XMTD    EQU  29          * XMIT HOLD DTE INPUT
RBUF0   EQU  30          * HOLD LOWPASS FILTERED SAMPLE
RBUF1   EQU  31          % RECEIVE BPF  COEFFICIENT
RBUF2   EQU  32          % RECEIVE BPF  COEFFICIENT
RBUF3   EQU  33          % RECEIVE BPF  COEFFICIENT
RBUF4   EQU  34          % RECEIVE BPF  COEFFICIENT
RBUF5   EQU  35          % RECEIVE BPF  COEFFICIENT
RBUF6   EQU  36          % RECEIVE BPF  COEFFICIENT
RBUF7   EQU  37          % RECEIVE BPF  COEFFICIENT
RBUF8   EQU  38          % RECEIVE BPF  COEFFICIENT
RBUF9   EQU  39          % RECEIVE BPF  COEFFICIENT
RBUF10  EQU  40          % RECEIVE BPF  COEFFICIENT
RBUF11  EQU  41          % RECEIVE BPF  COEFFICIENT
RBUF12  EQU  42          % RECEIVE BPF  COEFFICIENT
RBUF13  EQU  43          % RECEIVE BPF  COEFFICIENT
RBUF14  EQU  44          % RECEIVE BPF  COEFFICIENT
RBUF15  EQU  45          % RECEIVE BPF  COEFFICIENT
RBUF16  EQU  46          % RECEIVE BPF  COEFFICIENT
RBUF17  EQU  47          % RECEIVE BPF  COEFFICIENT
RBUF18  EQU  48          % RECEIVE BPF  COEFFICIENT
RBUF19  EQU  49          % RECEIVE BPF  COEFFICIENT
RBUF20  EQU  50          % RECEIVE BPF  COEFFICIENT
RBUF21  EQU  51          % RECEIVE BPF  COEFFICIENT
```

```
RBUF22 EQU  52          % RECEIVE BPF  COEFFICIENT
RBUF23 EQU  53          % RECEIVE BPF  COEFFICIENT
RBUF24 EQU  54          % RECEIVE BPF  COEFFICIENT
RBUF25 EQU  55          % RECEIVE BPF  COEFFICIENT
RBUF26 EQU  56          % RECEIVE BPF  COEFFICIENT
RBUF27 EQU  57          % RECEIVE BPF  COEFFICIENT
RBUF28 EQU  58          % RECEIVE BPF  COEFFICIENT
RBUF29 EQU  59          % RECEIVE BPF  COEFFICIENT
RBUF30 EQU  60          % RECEIVE BPF  COEFFICIENT
RBUF31 EQU  61          % RECEIVE BPF  COEFFICIENT
RBUF32 EQU  62          % RECEIVE BPF  COEFFICIENT
RBUF33 EQU  63          % RECEIVE BPF  COEFFICIENT
RBUF34 EQU  64          % RECEIVE BPF  COEFFICIENT
RBUF35 EQU  65          % RECEIVE BPF  COEFFICIENT
RBUF36 EQU  66          % RECEIVE BPF  COEFFICIENT
RBUF37 EQU  67          % RECEIVE BPF  COEFFICIENT
AGC    EQU  68          * AUTOMATIC GAIN FACTOR
AGCRA  EQU  69          * SIGNAL MAX RUNNING AVERAGE FOR AGC
RECST  EQU  70          * RECEIVER STATUS
AGCOFF EQU  71          * AGC CALCULATION LOOKUP TABLE
BSMAX  EQU  72          * BAUD SIGNAL MAX
AGCNT  EQU  73          * BAUD SAMPLE COUNT
AGCLEV EQU  74          * TEMPORARY AGC LEVEL (AGCUPT)
SAMPLE EQU  75          * BAUD LIMIT SAMPLE COUNT
SAMXMT EQU  76          * TRANSMITTER SAMPLE COUNT
BITOUT EQU  77          * DIBIT POSITIONED TO XMIT TO 7041
RPHSE  EQU  78          * OFFSET FOR RECEIVE PHASE DECODE
TRSHD1 EQU  79          * THRESHOLD FOR CARRIER RECOVERY
RALPHA EQU  80          * RECEIVE CARRIER POINTER
RDELTA EQU  81          * DELTA TO GENERATE RECEIVE CARRIER
ISUM   EQU  82          * FILTERED/PHASE SHIFTED SAMPLE
QSUM   EQU  83          * FILTERED/PHASE SHIFTED SAMPLE
RECI   EQU  84          * BASEBAND I CHANNEL
ROLDPH EQU  85          * PREVIOUS ABSOLUTE PHASE (QUADRANT)
RNEWPH EQU  86          * CURRENT  ABSOLUTE PHASE (QUADRANT)
ERRSIG EQU  87          * FILTERED CARRIER ERROR SIGNAL
MINUS1 EQU  88          * MINUS 1 IN THE Q12 FORMAT
PLL1   EQU  89          * CARRIER RECOVERY PLL FILTER COEFFICIENT 1
PLL2   EQU  90          * CARRIER RECOVERY PLL FILTER COEFFICIENT 2
FOUR   EQU  91          * >4 ( MASK VALUE FOR PHASE CODE/DECODE)
SIGNI  EQU  92          * SIGN OF I CHANNEL (TO COMPUTE CARRIER ERROR)
SIGNQ  EQU  93          * SIGN OF Q CHANNEL (TO COMPUTE CARRIER ERROR)
ERROR  EQU  94          * CARRIER PHASE ERROR
TEMP   EQU  95          * MISC. TEMPERORY REGISTER
RECQ   EQU  96          * BASEBAND Q CHANNEL
*-----DEFINE REGISTERS FOR BAUD CLOCK
ENRGY  EQU  97          * CURRENT ENERGY
PENRGY EQU  98          * PREVIOUS ENERGY
BERROR EQU  99          * BAUD CLOCK ERROR
BEROUT EQU  100         * OUTPUT OF BAUD PLL LOOP FILTER
BPLL1  EQU  101         * CLOCK RECOVERY PLL FILTER COEFFICIENT 1
BPLL2  EQU  102         * CLOCK RECOVERY PLL FILTER COEFFICIENT 2
TRSHD2 EQU  103         * CLOCK RECOVERY TRESHOLD
*---------------
*****       TRANSMITTER DIBIT ENCODER TABLE.          *****
********-------------------------------------------********
ENCODE  DATA    >0002       * DIBIT '01' = 90 deg.
        DATA    >0000       * DIBIT '00' = 0 deg.
        DATA    >0004       * DIBIT '10' = 180 deg
        DATA    >0006       * DIBIT '11' = 270 deg.
XPHASE  DATA    >7FFF       * 0 deg.    I CHANNEL =  1
        DATA    >0000       *           Q CHANNEL =  0
        DATA    >0000       * 90 deg.   I CHANNEL =  0
        DATA    >8000       *           Q CHANNEL = -1
        DATA    >8000       * 180 deg.  I CHANNEL = -1
        DATA    >0000       *           Q CHANNEL =  0
        DATA    >0000       * 270 deg.  I CHANNEL =  0
        DATA    >7FFF       *           Q CHANNEL =  1
********-------------------------------------------********
```

```
*****       RECEIVER DIBIT ENCODER TABLE.                    *****
*****       DIBITS are formed as 'MSB,LSB'.                  *****
*********-------------------------------------------*********
RPHASE  DATA    >0001       * 0  deg.,  DIBIT = '01'
        DATA    >0000       * 90 deg.,  DIBIT = '00'
        DATA    >0002       * 180 deg., DIBIT = '10'
        DATA    >0003       * 270 deg., DIBIT = '11'
*
M1      DATA    >7FFF       * MASK 1
M2      DATA    >007F       * MASK 2
M3      DATA    >0006       * MASK 3
CK      DATA    >0208       * CLOCK FOR AIB
MD      DATA    >000A       * MODE FOR AIB
ST      DATA    >1800
        DATA    >0000
DT      DATA    >1000       * TRANSMIT DELTA.
        DATA    >2000       * RECEIVE DELTA.
TH1     DATA    >0007       *  0.01  Q12 TRSHD FOR CARRIER
TH2     DATA    >0008       *  0.01  Q12 TRSHD FOR BAUD CLOCK
MIN1    DATA    >F000       * -1 Q12
PLS1    DATA    >0FFF       *  1 Q12
*
*********-------------------------------------------*********
*****   PLL LOOP FILTER COFFECIENTS.                     *****
*********-------------------------------------------*********
*
PLLC1   DATA    >4500   * Q15   CARRIER PLL INITIAL COEF. 1
PLLC2   DATA    >80     * Q15   CARRIER PLL COEFFICIENT 2
BPLLC1  DATA    >4000   * Q15   BAUD CLOCK PLL INITIAL COEF. 1
BPLLC2  DATA    >50     * Q15   BAUD CLOCK PLL COEFFICIENT 2
PLLC    DATA    >7A00   * Q15   CARRIER PLL STEADY STATE COEF. 1
BPLLC   DATA    >7500   * Q15   BAUD CLOCK PLL STEADY STATE COEF. 1
*
**********------------------------------------------********
*****   TASK MASTER SEQUENCE TABLE  (RECEIVE)           *****
*****   TASKS ARE EXECUTED FROM BOTTOM TO TOP           *****
**********------------------------------------------********
*
TSKSEQ  EQU  $
        DATA DUMMY    UNUSED CYCLE                        15
        DATA DUMMY    UNUSED CYCLE                        14
        DATA DUMMY    UNUSED CYCLE                        13
        DATA DUMMY    UNUSED CYCLE                        12
        DATA BDCLK2   COMPUTE ENERGY E(11)                11
        DATA DUMMY    UNUSED CYCLE                        10
        DATA OUT      COMMUNICATE WITH TMS7742             9
        DATA DECODE   DECODE/GET SCRAMBLED DIBIT           8
        DATA DEMODB   DEMODULATE IN THE MIDDLE OF BAUD     7
        DATA DUMMY    UNUSED CYCLE                         6
        DATA AGCUPT   UPDATE THE AGC (IF NECESSARY)        5
        DATA DUMMY    UNUSED CYCLE                         4
        DATA BDCLK1   COMPUTE ENERGY E(3)                  3
        DATA DUMMY    UNUSED CYCLE                         2
        DATA DUMMY    UNUSED CYCLE                         1
        DATA DUMMY    UNUSED CYCLE                         0
*
**********------------------------------------------********
*****   TASK MASTER SEQUENCE TABLE  (TRANSMIT)          *****
*****   TASKS ARE EXECUTED FROM BOTTOM TO TOP           *****
**********------------------------------------------********
*
TSKXMT  EQU  $
        DATA GETDBT   GET THE NEXT DIBIT                  16
        DATA DUMXMT   NO CYCLE                            15
        DATA DUMXMT   NO CYCLE                            14
        DATA DUMXMT   NO CYCLE                            13
        DATA DUMXMT   NO CYCLE                            12
        DATA DUMXMT   NO CYCLE                            11
        DATA DUMXMT   NO CYCLE                            10
```

```
        DATA DUMXMT   NO CYCLE                          9
        DATA DUMXMT   NO CYCLE                          8
        DATA DUMXMT   NO CYCLE                          7
        DATA DUMXMT   NO CYCLE                          6
        DATA DUMXMT   NO CYCLE                          5
        DATA DUMXMT   NO CYCLE                          4
        DATA DUMXMT   NO CYCLE                          3
        DATA DUMXMT   NO CYCLE                          2
        DATA DUMXMT   NO CYCLE                          1
        PAGE
********-----------------------------------------********
*****       RAISED COSINE COEFFICIENT TABLE.          *****
********-----------------------------------------********
COEF    DATA    >1
        DATA    >49A
        DATA    >394
        DATA    >FFD9
        DATA    >5A2
        DATA    >29A
        DATA    >FFAB
        DATA    >6A0
        DATA    >1B5
        DATA    >FF7A
        DATA    >789
        DATA    >ED
        DATA    >FF4C
        DATA    >853
        DATA    >45
        DATA    >FF27
        DATA    >8F4
        DATA    >FFC3
        DATA    >FF11
        DATA    >963
        DATA    >FF65
        DATA    >FF10
        DATA    >99C
        DATA    >FF2A
        DATA    >FF2A
        DATA    >99C
        DATA    >FF10
        DATA    >FF65
        DATA    >963
        DATA    >FF11
        DATA    >FFC3
        DATA    >8F4
        DATA    >FF27
        DATA    >45
        DATA    >853
        DATA    >FF4C
        DATA    >ED
        DATA    >789
        DATA    >FF7A
        DATA    >1B5
        DATA    >6A0
        DATA    >FFAB
        DATA    >29A
        DATA    >5A2
        DATA    >FFD9
        DATA    >394
        DATA    >49A
        DATA    >1
*
*************************************************************
*****  AGC DIVIDE LOOKUP TABLE                          *****
*****    STANDARD GAIN RANGE -- >3CC3 - >3F79           *****
*****    WITH 5% SIGNAL VARIATION -- >3966 - >41D6      *****
*****                                                   *****
*************************************************************
```

```
AGCTBL EQU  $-32          AGC TABLE LENGTH =  32
       DATA >F8                    0.9687500
       DATA >F0                    0.9375000
       DATA >EA                    0.9140625
       DATA >E3                    0.8867188 35 -
       DATA >DD                    0.8632812
       DATA >D7                    0.8398438
       DATA >D2                    0.8203125
       DATA >CC                    0.7968750 39 -
       DATA >C7                    0.7773438
       DATA >C3                    0.7617188
       DATA >BE                    0.7421875
       DATA >BA                    0.7265625 43 -
       DATA >B6                    0.7109375
       DATA >B2                    0.6953125
       DATA >AE                    0.6796875
       DATA >AA                    0.6640625 47 -
       DATA >A7                    0.6523438
       DATA >A3                    0.6367188
       DATA >A0                    0.6250000
       DATA >9D                    0.6132813 51 -
       DATA >9A                    0.6015625
       DATA >97                    0.5898438
       DATA >94                    0.5781250
       DATA >92                    0.5703125 55 -
       DATA >8F                    0.5585938
       DATA >8D                    0.5507813
       DATA >8A                    0.5390625
       DATA >88                    0.5312500 59 -
       DATA >86                    0.5234375
       DATA >84                    0.5156250
       DATA >82                    0.5078125
       DATA >7F                    0.4960938 63 -
       PAGE
*
********------------------------------------------********
*****              SINE(COSINE) TABLE                *****
********------------------------------------------********
SINE   DATA   >0
       DATA   >648
       DATA   >C8C
       DATA   >12C8
       DATA   >18F9
       DATA   >1F1A
       DATA   >2528
       DATA   >2B1F
       DATA   >30FC
       DATA   >36BA
       DATA   >3C57
       DATA   >41CE
       DATA   >471D
       DATA   >4C40
       DATA   >5134
       DATA   >55F6
       DATA   >5A82
       DATA   >5ED7
       DATA   >62F2
       DATA   >66D0
       DATA   >6A6E
       DATA   >6DCA
       DATA   >70E3
       DATA   >73B6
       DATA   >7642
       DATA   >7885
       DATA   >7A7D
       DATA   >7C2A
       DATA   >7D8A
       DATA   >7E9D
       DATA   >7F62
```

```
        DATA    >7FD9
COSINE  DATA    >7FFF
        DATA    >7FD9
        DATA    >7F62
        DATA    >7E9D
        DATA    >7D8A
        DATA    >7C2A
        DATA    >7A7D
        DATA    >7885
        DATA    >7642
        DATA    >73B6
        DATA    >70E3
        DATA    >6DCA
        DATA    >6A6E
        DATA    >66D0
        DATA    >62F2
        DATA    >5ED7
        DATA    >5A82
        DATA    >55F6
        DATA    >5134
        DATA    >4C40
        DATA    >471D
        DATA    >41CE
        DATA    >3C57
        DATA    >36BA
        DATA    >30FC
        DATA    >2B1F
        DATA    >2528
        DATA    >1F1A
        DATA    >18F9
        DATA    >12C8
        DATA    >C8C
        DATA    >648
        DATA    >0
        DATA    >F9B8
        DATA    >F374
        DATA    >ED38
        DATA    >E707
        DATA    >E0E6
        DATA    >DAD8
        DATA    >D4E1
        DATA    >CF04
        DATA    >C946
        DATA    >C3A9
        DATA    >BE32
        DATA    >B8E3
        DATA    >B3C0
        DATA    >AECC
        DATA    >AA0A
        DATA    >A57E
        DATA    >A129
        DATA    >9D0E
        DATA    >9930
        DATA    >9592
        DATA    >9236
        DATA    >8F1D
        DATA    >8C4A
        DATA    >89BE
        DATA    >877B
        DATA    >8583
        DATA    >83D6
        DATA    >8276
        DATA    >8163
        DATA    >809E
        DATA    >8027
        DATA    >8000
        DATA    >8027
        DATA    >809E
        DATA    >8163
```

```
        DATA    >8276
        DATA    >83D6
        DATA    >8583
        DATA    >877B
        DATA    >89BE
        DATA    >8C4A
        DATA    >8F1D
        DATA    >9236
        DATA    >9592
        DATA    >9930
        DATA    >9D0E
        DATA    >A129
        DATA    >A57E
        DATA    >AA0A
        DATA    >AECC
        DATA    >B3C0
        DATA    >B8E3
        DATA    >BE32
        DATA    >C3A9
        DATA    >C946
        DATA    >CF04
        DATA    >D4E1
        DATA    >DAD8
        DATA    >E0E6
        DATA    >E707
        DATA    >ED38
        DATA    >F374
        DATA    >F9B8
*
********-----------------------------------------********
*****  RECEIVER I CHANNEL BPASS FILTER COEFFICIENTS  *****
********-----------------------------------------********
*
ICF0    EQU     58      * 3A 0.014064
*ICF1   EQU     0       * 0 0.000000
ICF2    EQU     -58     * FFC6 -0.014067
*ICF3   EQU     0       * 0 0.000000
ICF4    EQU     28      * 1C 0.006883
*ICF5   EQU     0       * 0 0.000000
ICF6    EQU     37      * 25 0.009069
*ICF7   EQU     0       * 0 0.000000
ICF8    EQU     -137    * FF77 -0.033477
*ICF9   EQU     0       * 0 0.000000
ICF10   EQU     262     * 106 0.063862
*ICF11  EQU     0       * 0 0.000000
ICF12   EQU     -393    * FE77 -0.095882
*ICF13  EQU     0       * 0 0.000000
ICF14   EQU     509     * 1FD 0.124198
*ICF15  EQU     0       * 0 0.000000
ICF16   EQU     -588    * FDB4 -0.143676
*ICF17  EQU     0       * 0 0.000000
ICF18   EQU     617     * 269 0.150616
*ICF19  EQU     0       * 0 0.000000
ICF20   EQU     -588    * FDB4 -0.143676
*ICF21  EQU     0       * 0 0.000000
ICF22   EQU     509     * 1FD 0.124198
*ICF23  EQU     0       * 0 0.000000
ICF24   EQU     -393    * FE77 -0.095882
*ICF25  EQU     0       * 0 0.000000
ICF26   EQU     262     * 106 0.063862
*ICF27  EQU     0       * 0 0.000000
ICF28   EQU     -137    * FF77 -0.033477
*ICF29  EQU     0       * 0 0.000000
ICF30   EQU     37      * 25 0.009069
*ICF31  EQU     0       * 0 0.000000
ICF32   EQU     28      * 1C 0.006883
*ICF33  EQU     0       * 0 0.000000
ICF34   EQU     -58     * FFC6 -0.014067
*ICF35  EQU     0       * 0 0.000000
ICF36   EQU     58      * 3A 0.014064
```

```
*
*********----------------------------------------*********
*****  RECEIVER Q CHANNEL BPASS FILTER COEFFICIENTS  *****
*********----------------------------------------*********
*
*QCF0    EQU     0       * 0 0.000000
QCF1     EQU     61      * 3D 0.014809
*QCF2    EQU     0       * 0 0.000000
QCF3     EQU     -47     * FFD1 -0.011510
*QCF4    EQU     0       * 0 0.000000
QCF5     EQU     0       * 0 0.000034
*QCF6    EQU     0       * 0 0.000000
QCF7     EQU     83      * 53 0.020321
*QCF8    EQU     0       * 0 0.000000
QCF9     EQU     -197    * FF3B -0.048158
*QCF10   EQU     0       * 0 0.000000
QCF11    EQU     328     * 148 0.079991
*QCF12   EQU     0       * 0 0.000000
QCF13    EQU     -454    * FE3A -0.110844
*QCF14   EQU     0       * 0 0.000000
QCF15    EQU     554     * 22A 0.135320
*QCF16   EQU     0       * 0 0.000000
QCF17    EQU     -610    * FD9E -0.148859
*QCF18   EQU     0       * 0 0.000000
QCF19    EQU     610     * 262 0.148859
*QCF20   EQU     0       * 0 0.000000
QCF21    EQU     -554    * FDD6 -0.135320
*QCF22   EQU     0       * 0 0.000000
QCF23    EQU     454     * 1C6 0.110844
*QCF24   EQU     0       * 0 0.000000
QCF25    EQU     -328    * FEB8 -0.079991
*QCF26   EQU     0       * 0 0.000000
QCF27    EQU     197     * C5 0.048158
*QCF28   EQU     0       * 0 0.000000
QCF29    EQU     -83     * FFAD -0.020321
*QCF30   EQU     0       * 0 0.000000
QCF31    EQU     0       * 0 -0.000034
*QCF32   EQU     0       * 0 0.000000
QCF33    EQU     47      * 2F 0.011510
*QCF34   EQU     0       * 0 0.000000
QCF35    EQU     -61     * FFC3 -0.014809
*QCF36   EQU     0       * 0 0.000000
*
         PAGE
*********************************************************
******     Initializtion routine                  ******
*********************************************************
START    DINT
         LDPK    0
         ROVM
         LACK    1
         SACL    ONE
         LACK    M1      * INITIALIZE MASK 1
         TBLR    MASK1
         LACK    M2      * INITIALIZE MASK 2
         TBLR    MASK2
         LACK    M3      * INITIALIZE MASK 3
         TBLR    MASK3
*
         LACK    MD          * AIB BOARD INITIALIZATION.
         TBLR    TEMP        * MD IS MODE CNTRL FOR AIB.
         OUT     TEMP,PA0
         LACK    CK          * CK IS SAMPLE RATE FOR AIB.
         TBLR    TEMP
         OUT     TEMP,PA1    * SENT CLOCK VALUE TO PORT 1(NEW AIB)
*
         LACK    SINE        * TABLE OFFSET INITIALIZATION.
         SACL    OFSET0      * SINE TABLE OFFSET
         LACK    ENCODE
```

```
        SACL    OFSET1          * <DIBIT TO PHASE> TABLE
        LACK    COEF
        SACL    XPTR            * RAISED COS COEF. TABLE.
        LACK    XPHASE
        SACL    INDXPH          * OFSET FOR XMIT PHASE TABLE
        LACK    RPHASE
        SACL    RPHSE           * OFSET FOR RCVR PHASE TABLE
        LACK    4
        SACL    FOUR            *
        ZAC                     * MISC. INITIALIZATIONS.
        SACL    ROLDPH          * INITIALIZE PREVIOUS TOTAL PHASE
*
        SACL    RALPHA          * SWAVE INITIALIZATIONS.
        SACL    XALPHA
        LACK    DT
        TBLR    XDELTA          * READ SWAVE DELTAs
        ADD     ONE
        TBLR    RDELTA
*
*
        LACK    TH1             * CARRIER PLL THRESHOLD
        TBLR    TRSHD1
        ADD     ONE
        TBLR    TRSHD2          * BAUD CLOCK PLL THRESHOLD
        LACK    MIN1
        TBLR    MINUS1          * -1 IN Q12
        LACK    PLS1
        TBLR    PLUS1           * +1 IN Q12
*-------------------------------
        LACK    PLLC1           * CARRIER PLL INITIAL COEF. 1
        TBLR    PLL1
        ADD     ONE
        TBLR    PLL2            * CARRIER PLL COEF. 2
        ADD     ONE
        TBLR    BPLL1           * BAUD CLOCK PLL INITIAL COEF. 1
        ADD     ONE
        TBLR    BPLL2           * BAUD CLOCK COEF. 2
*-------------------------------
        LACK    AGCTBL          * SET THE AGC TABLE LOOKUP
        SACL    AGCOFF          * OFF SET VALUE
        LAC     ONE,13          * INITIALIZE RUNNING AVERAGE
        SACL    AGCRA           * TO >2000
        LACK    >FF             * INITIALIZE THE AGC FACTOR
        SACL    AGC             * TO ONE
        ZAC                     * INITIALIZE THE
        SACL    BSMAX           * BAUD SIGNAL MAX TO ZERO
        LACK    3               * RUNNING AVERAGE COUNT
        SACL    AGCNT           * TO THREE
        LACK    >20             * SET THE ENERGY DETECT
        SACL    RECST           * BIT IN THE STATUS FLAG WORD
*
        LACK    15              * SET THE REC SAMPLE COUNT
        SACL    SAMPLE          * TO 16
        ZAC                     * SET THE XMT SAMPLE COUNT
        SACL    SAMXMT          * TO ZERO
*
        PAGE
*-----------------------------------------------------------------*
*   THE FOLLOWING CODE HANDLES COMMANDS FROM THE 7042             *
*-----------------------------------------------------------------*
       LAC  ONE,4       SET COUNTER VALUE TO RUN
       SACL TEMP        DLB AT 600 BAUD
COMD   BIOZ LOOK         WAIT FOR 9600HZ SAMPLE PULSE
       B    COMD
LOOK   NOP
       IN   RBUF0,PA2   *--- DUMMY READ TO GET COUNTER GOING
LOOK1  IN   XMTD,PA6    LOOK FOR COMMAND
       LACK >30         MASK OFF ALL BUT COMMAND BITS
       AND  XMTD        CHECK COMMAND BITS FOR NEW COMMAND
```

```
        BZ    COMD          IF ZERO THEN NO COMMAND YET
        SUB   ONE,4         CHECK FOR DIGITAL LOOP BACK TEST
        BZ    LDLB          IF SO THEN EXECUTE TEST
        SUB   ONE,4         CHECK FOR MODEM RUN COMMAND
        BZ    WAIT          IF SO THEN RUN MODEM
        LACK  >F            MASK OFF COMMAND BITS
        AND   XMTD          TO GET SPECIFIC CONFIGURATION
*    THIS IS FOR CONFIGURATION CODES
        BZ    COMD          ZERO IS NOT VALID COMMAND
        SUB   ONE           CHECK FOR COMMAND ONE
        BZ    SETALB        SETUP THE MODEM TO RUN ALB
        SUB   ONE           CHECK FOR COMMAND TWO
        BZ    SETORG        SETUP THE MODEM TO RUN ORIGINATE
        SUB   ONE           CHECK FOR COMMAND THREE
        BZ    SQTREC        SHUT DOWN RECEIVER TO RUN XMIT ONLY
        B     COMD          CHECK FOR NEXT COMMAND
SETALB  LAC   ONE,13        LOAD ACC WITH 2000 TO PUT
        SACL  XDELTA        XMIT IN SAME BAND AS RECEIVE
        B     COMD          CHECK FOR NEXT COMMAND
SETORG  LAC   ONE,12        LOAD ACC WITH 1000 TO PUT
        SACL  XDELTA        XMIT IN ORIGINATE MODE
        B     COMD          CHECK FOR NEXT COMMAND
SQTREC  LAC   ONE,8         SET RECEIVER SQUELCH BIT
        OR    RECST         IN THE RECEIVE STATUS REG
        SACL  RECST         TO DISABLE RECEIVER CODE
        B     COMD          CHECK FOR NEXT COMMAND
*
LDLB    BIOZ  DLBOUT        WAIT FOR NEXT SAMPLE PERIOD
        B     LDLB          LOOP ON TIMER
DLBOUT  IN    RBUF0,PA2     *--- DUMMY READ TO GET COUNTER GOING
        LAC   TEMP          GET 16 SAMPLE BAUD COUNTER
        SUB   ONE           DECREMENT IT
        SACL  TEMP          SAVE COUNT
        BNZ   LDLB          COUNT ANOTHER SAMPLE PERIOD
        LAC   ONE,4         RESET COUNTER VALUE TO RUN
        SACL  TEMP          DLB AT 600 BAUD
        LAC   XMTD,10       ADJUST FOR OUTPUT RANGE
        OR    MINUS1      * MASK COMMAND BITS(15-12)TO 1'S
        SACL  XMTD          STORE IT FOR OUTPUT
        OUT   XMTD,PA6      ECHO INPUT
        B     LOOK1         REPEAT LOOP BACK TEST
*-------------------------------------------------------------*
********-------------------------------------------------********
*****    THE FOLLOWING SECTION IMPLEMENTS MODEM FUNCTIONS    *****
********-------------------------------------------------********
*-------------------------------------------------------------*
WAIT    BIOZ    GO          * WAIT FOR 9600HZ SAMPLE PULSE
        B       WAIT
GO      NOP
        OUT     XMTOUT,PA2  * OUTPUT TO D/A
        IN      RBUF0,PA2   * INPUT FROM A/D
********-----------------------------------------********
*****    TRANSMITER SECTION STARTS HERE.                *****
********-----------------------------------------********
XMITER  EQU     $
*
********-----------------------------------------********
*****       SINE(COSINE) WAVE GENERATION                *****
********-----------------------------------------********
SWAVE   EQU     $
        LAC     XALPHA,8        * DELTA IS THE INCREMENT.
        SACH    TEMP            * ISOLATE INTEGER PORTION.
        LAC     TEMP
        ADD     OFSET0          * ADD INDEX TO SINE TABLE.
        TBLR    SINA            * SINE VALUE,  (Q15).
        LACK    >20             * OFFSET TO COSINE VALUE (Q15).
        ADD     TEMP
        AND     MASK2
        ADD     OFSET0          * ADD INDEX TO COSINE TABLE.
```

```
        TBLR    COSA            * COSINE VALUE, (Q15).
        LAC     XALPHA          * COMPUTE ADDRESS OF NEXT
        ADD     XDELTA          * POINT FOR TABLE.
        AND     MASKI           * KEEP MOD128, MASK=>7FFF.
        SACL    XALPHA          * SAVE NEXT ADDRESS
*
*********-------------------------------------------*********
*****   TRANSMITTER 48 TAP RAISED COSINE FILTER.      *****
*****   INPUTS UPDATED AT 600HZ RATE.                 *****
*****   OUTPUT UPDATED AT 9600HZ RATE.                *****
*********-------------------------------------------*********
RACS    EQU     $
        LAC     XPTR
        TBLR    CX0             * RETRIEVE COEFFICIENTS
        ADD     ONE
        TBLR    CX1
        ADD     ONE
        TBLR    CX2
        ADD     ONE
        SACL    XPTR
*
        ZAC
        LT      XIBUF2          * COMPUTE FILTER TAPS ICHAN.
        MPY     CX2
        LTA     XIBUF1
        MPY     CX1
        LTA     XIBUF0
        MPY     CX0
        APAC
        SACH    XIOUT,1
*
        ZAC
        LT      XQBUF2          * COMPUTE FILTER TAPS QCHAN.
        MPY     CX2
        LTA     XQBUF1
        MPY     CX1
        LTA     XQBUF0
        MPY     CX0
        APAC
        SACH    XQOUT,1
*
XMIT    ZAC
        LT      XIOUT           * ICHAN*cos(wt)+ QCHAN*sin(wt)
        MPY     COSA
        LTA     XQOUT
        MPY     SINA
        APAC
        SACH    XMTOUT,1
*
        PAGE
*********-------------------------------------------*********
*****   RECEIVER I CHANNEL BANDPASS FILTER.           *****
*****   SAMPLING RATE IS 9600HZ.                      *****
*********-------------------------------------------*********
CONT6   ZAC
        LT      RBUF36
        MPYK    ICF36
        LTA     RBUF34
        MPYK    ICF34
        LTA     RBUF32
        MPYK    ICF32
        LTA     RBUF30
        MPYK    ICF30
        LTA     RBUF28
        MPYK    ICF28
        LTA     RBUF26
        MPYK    ICF26
        LTA     RBUF24
```

```
        MPYK    ICF24
        LTA     RBUF22
        MPYK    ICF22
        LTA     RBUF20
        MPYK    ICF20
        LTA     RBUF18
        MPYK    ICF18
        LTA     RBUF16
        MPYK    ICF16
        LTA     RBUF14
        MPYK    ICF14
        LTA     RBUF12
        MPYK    ICF12
        LTA     RBUF10
        MPYK    ICF10
        LTA     RBUF8
        MPYK    ICF8
        LTA     RBUF6
        MPYK    ICF6
        LTA     RBUF4
        MPYK    ICF4
        LTA     RBUF2
        MPYK    ICF2
        LTA     RBUF0
        MPYK    ICF0
        APAC
        SACH    ISUM,4      * OUTPUT OF I CHAN.
*
********-------------------------------------------********
*****   RECEIVER Q CHANNEL BANDPASS FILTER.          *****
*****   SAMPLING RATE IS 9600HZ.                     *****
********-------------------------------------------********
        LTD     RBUF35
        ZAC
        MPYK    QCF35
        DMOV    RBUF34
        LTD     RBUF33
        MPYK    QCF33
        DMOV    RBUF32
        LTD     RBUF31
        MPYK    QCF31
        DMOV    RBUF30
        LTD     RBUF29
        MPYK    QCF29
        DMOV    RBUF28
        LTD     RBUF27
        MPYK    QCF27
        DMOV    RBUF26
        LTD     RBUF25
        MPYK    QCF25
        DMOV    RBUF24
        LTD     RBUF23
        MPYK    QCF23
        DMOV    RBUF22
        LTD     RBUF21
        MPYK    QCF21
        DMOV    RBUF20
        LTD     RBUF19
        MPYK    QCF19
        DMOV    RBUF18
        LTD     RBUF17
        MPYK    QCF17
        DMOV    RBUF16
        LTD     RBUF15
        MPYK    QCF15
        DMOV    RBUF14
        LTD     RBUF13
        MPYK    QCF13
        DMOV    RBUF12
```

```
        LTD     RBUF11
        MPYK    QCF11
        DMOV    RBUF10
        LTD     RBUF9
        MPYK    QCF9
        DMOV    RBUF8
        LTD     RBUF7
        MPYK    QCF7
        DMOV    RBUF6
        LTD     RBUF5
        MPYK    QCF5
        DMOV    RBUF4
        LTD     RBUF3
        MPYK    QCF3
        DMOV    RBUF2
        LTD     RBUF1
        MPYK    QCF1
        DMOV    RBUF0
        APAC
        SACH    QSUM,4      * OUTPUT OF Q CHAN.
*
        PAGE
*************************************************************************
*       DETECT MAXIMUM SIGNAL STRENGTH OF RECI PER BAUD
AGCAL   EQU     $
        LAC     ISUM        * AGC VALUE CALCULATED USING ISUM
        ABS                   GET MAGNETUDE OF SIGNAL
        SUB     BSMAX         COMPARE TO PREVIOUS MAX VALUE
        BLZ     OVRMAX        IF LESS THAN THEN JUMP OVER UPDATE
        ADD     BSMAX         RESTORE VALUE AND
        SACL    BSMAX         STORE AS NEW MAX
*       MULTIPLY IN AGC FACTOR TO FILTERED SIGNAL
OVRMAX  LT      AGC           MULTIPLY THE AGC FACTOR
MPYI    MPY     ISUM          BY THE FILTERED DATA ELEMENT
        PAC                   MOVE THE PRODUCT TO THE ACC
        SACH    TEMP,4        SAVE TOP HALF OF ACC
        AND     PLUS1         MASK OFF UNUSABLE BITS
        SACL    ISUM          SAVE BOTTOM HALF OF ACC
        ZALH    TEMP          RELOAD HIGH ACC VALUE
        ADD     ISUM,4       SHIFT LOW HALF INTO POSITION
        SACH    ISUM,4       STORE Q15 GAINED FILTERED DATA
MPYQ    LT      AGC           MULTIPLY THE AGC FACTOR
        MPY     QSUM          BY THE FILTERED DATA ELEMENT
        PAC                   MOVE THE PRODUCT TO THE ACC
        SACH    TEMP,4        SAVE TOP HALF OF ACC
        AND     PLUS1         MASK OFF UNUSABLE BITS
        SACL    QSUM          SAVE BOTTOM HALF OF ACC
        ZALH    TEMP          RELOAD HIGH ACC VALUE
        ADD     QSUM,4       SHIFT LOW HALF INTO POSITION
        SACH    QSUM,4       STORE Q15 GAINED FILTERED DATA
*
        PAGE
*****************************************************************
*  The following code is the time sliced code task master.      *
*  The routine monitors the status of the modem operations      *
*  and sequences the code appropriately.                        *
*****************************************************************
MASTER  EQU  $
        LAC  ONE,5      *  CHECK OPERATING STATUS FOR
        AND  RECST      *  ENERGY DETECT
        BZ   HANGUP     *  IF NO ENERGY DETECT THEN HANG UP
        LAC  ONE,4      *  CHECK IF LOCAL CARRIER
        AND  RECST      *  IS LOCKED. IF SO SWITCH
        BNZ  CARLCK     *  PLL FILTERS BANDWIDTH
        B    NORMAL     *  EXECUTE NORMAL SEQUENCE
*
CARLCK  LACK PLLC       * CHANGE CARRIER PLL COEF. 1
        TBLR PLL1
        LACK BPLLC      * CHANGE BAUD CLOCK PLL COEF. 1
```

```
        TBLR BPLL1
NORMAL  EQU  $
        LAC  SAMPLE        * DECREMENT THE SAMPLE COUNT
        SUB  ONE           * TO CHECK FOR END OF BAUD
        BGEZ OVRSAM        * IF NOT THEN SKIP COUNT RESET
        LACK 15            * RESTART THE SAMPLE COUNTER AT 15
OVRSAM  SACL SAMPLE        * SAVE NEW COUNT VALUE
        LACK TSKSEQ        * GET ADDRESS OF TOP OF TABLE
        ADD  SAMPLE        * ADD IN OFFSET
        TBLR TEMP          * GET THE PROGRAM ADDRESS
        LAC  TEMP          * FOR THE TASK CALL
        CALA               * EXECUTE THE APPROPRIATE TASK
*
*     UPDATE CARRIER ANGLE AT SAMPLE RATE
*
        LAC  RALPHA        * COMPUTE ADDRESS OF NEXT
        ADD  RDELTA        * POINT FOR TABLE.
        AND  MASK1         * KEEP MOD128, MASK=>7FFF.
        SACL RALPHA        * SAVE NEXT ADDRESS
*
MASXMT  EQU  $             * EXECUTE TRANSMIT TASK SEQUENCE
        LAC  SAMXMT        * DECREMENT THE SAMPLE COUNT
        SUB  ONE           * TO CHECK FOR END OF BAUD
        BGEZ OVRSM1        * IF NOT THEN SKIP COUNT RESET
        LACK 15            * RESTART THE SAMPLE COUNTER AT 15
OVRSM1  SACL SAMXMT        * SAVE NEW COUNT VALUE
        LACK TSKXMT        * GET ADDRESS OF TOP OF TABLE
        ADD  SAMXMT        * ADD IN OFFSET
        TBLR TEMP          * GET THE PROGRAM ADDRESS
        LAC  TEMP          * FOR THE TASK CALL
        CALA               * EXECUTE THE APPROPRIATE TASK
        B    WAIT          * WAIT FOR NEXT SAMPLE TIMEOUT
        PAGE
*****************************************************************
*  This is the software automatic gain control factor update.   *
*  The routine keeps a running average plus three baud max's    *
*  to generate each new AGC update.  Once the value is gained   *
*  the routine uses a table lookup devide to force the filter   *
*  data max's into a tight range.                               *
*****************************************************************
AGCUPT ZALH AGCRA          ADD THE NEW BSMAX VALUE
       ADD  BSMAX,14       TO THE RUNNING AVERAGE
       SACH AGCRA          AND SAVE IT
       LAC  AGCNT          DECREMENT RUNNING AVERAGE COUNT
       SUB  ONE            SAVE IT AND
       SACL AGCNT          CHECK FOR ZERO
       SACH BSMAX          ZERO OUT RUNNING SIGNAL MAX
       BZ   OVROUT         IF ZERO THEN UPDATE AGC
       RET                 ELSE RETURN TO CALLING SEQUENCE
OVROUT LACK 3              RESET RUNNING AVERAGE COUNT
       SACL AGCNT          TO THREE
       LAC  AGCRA          MOVE AGCRA
       SACL AGCLEV         TO THE CALCULATION LEVEL
       LAC  AGCRA,14       DIVIDE RUNNING AVERAGE SUM
       SACH AGCRA          BY 4 TO GET NEW RUNNING AVERAGE
       LAC  AGCLEV         GET AVERAGE MAX SIGNAL LEVEL
       SUB  ONE,14         COMPARE TO 16384
       BLZ  ASHF1          IF LESS THAN SHIFT TABLE LOOKUP
       LAC  AGCLEV,7       GET LOOKUP VALUE
       SACH TEMP           MOVE LOOKUP VALUE TO
       LAC  TEMP           THE LOW HALF OF THE ACC
       ADD  AGCOFF         ADD IN TABLE OFFSET
       TBLR AGC            AND GET AGC VALUE
       LAC  AGC,15         DIVIDE THE AGC VALUE
       SACH AGC            BY 2 TO FORCE TO Q14 MODE
       RET                 RETURN TO CALLING SEQUENCE
ASHF1  ADD  ONE,13         COMPARE TO 8192
       BLZ  ASHF2          IF LESS THAN SHIFT TABLE LOOKUP
```

```
        LAC   AGCLEV,8      GET LOOKUP VALUE
        SACH  TEMP          MOVE LOOKUP VALUE TO
        LAC   TEMP          THE LOW HALF OF THE ACC
        ADD   AGCOFF        ADD IN TABLE OFFSET
        TBLR  AGC           AND GET AGC VALUE
        RET                 RETURN TO CALLING SEQUENCE
ASHF2   ADD   ONE,12        COMPARE TO 4096
        BLZ   ASHF3         IF LESS THAN SHIFT TABLE LOOKUP
        LAC   AGCLEV,9      GET LOOKUP VALUE
        SACH  TEMP          MOVE LOOKUP VALUE TO
        LAC   TEMP          THE LOW HALF OF THE ACC
        ADD   AGCOFF        ADD IN TABLE OFFSET
        TBLR  AGC           AND GET AGC VALUE
        LAC   AGC,1         AGC VALUE * 2 TO ADJUST
        SACL  AGC           FOR LOWER SIGNAL STRENGTH
        RET                 RETURN TO CALLING SEQUENCE
ASHF3   ADD   ONE,11        COMPARE TO 2048
        BLZ   ASHF4         IF LESS THAN SHIFT TABLE LOOKUP
        LAC   AGCLEV,10     GET LOOKUP VALUE
        SACH  TEMP          MOVE LOOKUP VALUE TO
        LAC   TEMP          THE LOW HALF OF THE ACC
        ADD   AGCOFF        ADD IN TABLE OFFSET
        TBLR  AGC           AND GET AGC VALUE
        LAC   AGC,2         AGC VALUE * 4 TO ADJUST
        SACL  AGC           FOR LOWER SIGNAL STRENGTH
        RET                 RETURN TO CALLING SEQUENCE
ASHF4   ADD   ONE,10        COMPARE TO 1024
        BLZ   ASHF5         IF LESS THAN SHIFT TABLE LOOKUP
        LAC   AGCLEV,11     GET LOOKUP VALUE
        SACH  TEMP          MOVE LOOKUP VALUE TO
        LAC   TEMP          THE LOW HALF OF THE ACC
        ADD   AGCOFF        ADD IN TABLE OFFSET
        TBLR  AGC           AND GET AGC VALUE
        LAC   AGC,3         AGC VALUE * 8 TO ADJUST
        SACL  AGC           FOR LOWER SIGNAL STRENGTH
        RET                 RETURN TO CALLING SEQUENCE
ASHF5   ADD   ONE,9         COMPARE TO 512
        BLZ   ASHF5         IF LESS THAN SHIFT TABLE LOOKUP
        LAC   AGCLEV,12     GET LOOKUP VALUE
        SACH  TEMP          MOVE LOOKUP VALUE TO
        LAC   TEMP          THE LOW HALF OF THE ACC
        ADD   AGCOFF        ADD IN TABLE OFFSET
        TBLR  AGC           AND GET AGC VALUE
        LAC   AGC,4         AGC VALUE * 16 TO ADJUST
        SACL  AGC           FOR LOWER SIGNAL STRENGTH
        RET                 RETURN TO CALLING SEQUENCE
ASHF6   ADD   ONE,5         COMPARE TO 32
        BLZ   NOEDT         LOST MINIMUM ENERGY LEVEL
        LAC   AGCLEV,13     GET LOOKUP VALUE
        SACH  TEMP          MOVE LOOKUP VALUE TO
        LAC   TEMP          THE LOW HALF OF THE ACC
        ADD   AGCOFF        ADD IN TABLE OFFSET
        TBLR  AGC           AND GET AGC VALUE
        LAC   AGC,5         AGC VALUE * 32 TO ADJUST
        SACL  AGC           FOR LOWER SIGNAL STRENGTH
        RET                 RETURN TO CALLING SEQUENCE
NOEDT   LACK  >DF           PASSBAND SIGNAL TOOL LOW
        AND   RECST         DISABLE SIGNAL ENERGY DETECT
        SACL  RECST         AND CARRIER DETECT SIGNAL
        RET                 RETURN TO CALLING SEQUENCE
*
        PAGE
HANGUP  B     WAIT
DUMXMT  EQU   $
        RET
SMARK   EQU   $
        RET
*
```

```
*********************************************************
*
GETDBT EQU $
       IN    XMTD,PA6     * GET NEW DIBIT
       LACK  >30
       AND   XMTD         * CHECK COMMAND BITS
       BZ    COMD         * IF ZERO SQT MODEM, IDLE
*
       LACK  COEF         * RECYCLE IF FINISHED
       SACL  XPTR
       DMOV  XIBUF1       * SHIFT UP THE FILTER
       DMOV  XIBUF0       * TO MAKE ROOM FOR
       DMOV  XQBUF1       * FOR THE NEW DATA VALUE
       DMOV  XQBUF0       * JUST INPUT
*
       LACK  3
       AND   XMTD         * NEW DIBIT FROM 7000
       ADD   OFSET1       * LOOKUP NEWPHASE
       TBLR  XNEWPH
       LAC   XOLDPH       * GET OLDPHASE.
       ADD   XNEWPH       * ADD NEW PHASE.
       AND   MASK3        * MASK WITH >0006.
       SACL  XOLDPH       * STORE BACK 'NEW' OLDPHASE.
       ADD   INDXPH       * LOOKUP I & Q INPUTS.
       TBLR  XIBUF0
       ADD   ONE
       TBLR  XQBUF0
       RET
*
*
DUMMY  CALL DEMOD         ATTEMPT DEMODULATION
       RET                RETURN TO TASK MASTER
*
*
DEMODB EQU  $             MIDDLE OF THE BAUD
       LACK >FE           RESET THE CURRENT BAUD CLOCK
       AND  RECST         CORRECTION FLAG IN THE STATUS
       SACL RECST         REGISTER AND SAVE IT
*********************************************************
*****   DEMODUATE THE PASSBAND SIGNAL.                *****
********-------------------------------------------*********
*****   RCVR. CARRIER SINE(COSINE) WAVE GENERATOR     *****
*********************************************************
DEMOD  EQU  $
       LAC   RALPHA,8     * DELTA IS THE INCREMENT.
       SACH  TEMP         * ISOLATE INTEGER PORTION.
       LAC   TEMP
       ADD   OFSET0       * ADD INDEX TO SINE TABLE.
       TBLR  SINA         * SINE VALUE,  (Q15).
       LACK  >20
       ADD   TEMP
       AND   MASK2
       ADD   OFSET0       * ADD INDEX TO COSINE TABLE.
       TBLR  COSA         * COSINE VALUE, (Q15).
*
*********************************************************
*
CONTI  LT    ISUM         * DEMOD. I CHANNEL
       MPY   COSA         * A=(Yi * cosA)/2
       PAC
       LT    QSUM
       MPY   SINA
       APAC               * A=(Yi * cosA)/2 + (Yq * sin A)/2
       SACH  RECI,1       * RECI= (Yi * cosA) + (Yq * sinA)
*
       LT    ISUM         * DEMOD. Q CHANNEL
       MPY   SINA
       PAC                * A = (Yi * sinA)/2
       LT    QSUM
       MPY   COSA
```

```
        SPAC                    * A =[(Yi * s
        SACH   RECQ,1           * RECQ = (Yi
*
*---MUST DETERMINE ENERGY FOR BAUD CLOC
*
        LT     RECI
        MPY    RECI         * FIND I**2
        PAC
        LT     RECQ
        MPY    RECQ         * FIND Q**2
        APAC
        SACH   ENRGY        * ENERGY = (I**
*
*---MUST DETERMINE SIGN OF I AND Q FOR
*
        LAC    RECI           * DETERMINE
        BGZ    DM1
        LAC    MINUS1
        B      DM2
DM1     LAC    PLUS1
DM2     SACL   SIGNI          * SAVE SIGN (
*
        LAC    RECQ           * DETERMINE
        BGZ    DM3
        LAC    MINUS1
        B      DM4
DM3     LAC    PLUS1
DM4     SACL   SIGNQ
        RET                   RETURN TO CA
*****************************************
* INOUT GET DIBIT FROM 7000 AND XMIT N
* TO THE 7000
*****************************************
OUT     EQU    $
        LAC    RDIBIT,10
        OR     MINUS1         * MASK D15-DI
        SACL   BITOUT         *AND SAVE THE
        OUT    BITOUT,PA6     *XMIT TO 7000
        RET                   * BACK TO CAL
********------------------------------
*****    PHASE DECODING - BINARY TO GR
* THIS ROUTINE CALCULATES PHASE SHIFT
* CURRENT ABSOLUTE PHASE, GREY CODE RE
********------------------------------
DECODE LAC    RECI         * DETERMINE ABS
       BGZ    ABS1
       LAC    RECQ
       BGZ    ABS2
       LACK   2            * PHASE IS 2 (0
       B      DIFFER       *
ABS2   LACK   3            * PHASE IS 3 (2
       B      DIFFER       *
ABS1   LAC    RECQ
       BGZ    ABS3
       LACK   1            * PHASE IS 1 (1
       B      DIFFER       *
ABS3   LACK   0            * PHASE IS 0 (9
*                          *
DIFFER SACL   TEMP
       SUB    ROLDPH       * SUBTRACT PREV
       BGEZ   DF1          * LUTE PHASE (E
       ADD    FOUR
DF1    ADD    RPHSE        * MAP PHASE CH/
       TBLR   RDIBIT
       LAC    TEMP
       SACL   ROLDPH
********------------------------------
*****    COMPUTE CARRIER ERROR SIGNAL.
*****    e(t) = RECI*SIGNQ - RECQ*SIGNI
********------------------------------
```

```
COMERR ZAC
       LT      REC1
       MPY     SIGNQ
       LTA     RECQ
       MPY     SIGNI
       SPAC
       SACH    ERROR,1      * ERROR IS IN Q12
********-------------------------------------------********
*****   LOOP FILTER                                  ******
********-------------------------------------------********
       ZAC
       LT      PLL2
       MPY     ERROR
       LTA     PLL1
       MPY     ERRSIG
       APAC
       SACH    ERRSIG,1     * ERRSIG IS IN Q12
*****************************************************************
* CORRECT PHASE ERROR ONLY AT MIDDLE OF BAUD                    *
*****************************************************************
*  Adjust carrier phase +/-                                     *
*  one table entry if     - (2*trshld) > error > trshld         *
*  two table entries if   - (2*trshld) < error >> trshld        *
*  RALPHA is current local carrier table index.(in MSB )        *
*****************************************************************
CKEROR  LAC    ERRSIG       *
        BGZ    ERR1         * If error is -ve add threshold
        ADD    TRSHD1       *
        BGZ    ERRETN       * Still -ve?... add again
        ADD    TRSHD1       *
        BGZ    SUB1A        * still -ve?...
        LAC    RALPHA       * Error >> trshld; add 2 to index
        SUB    ONE,9        *
        B      ERR2         *
SUB1A   LAC    RALPHA       * Error > trshld; add 1 to index
        SUB    ONE,8        *
        B      ERR2         *
ERR1    SUB    TRSHD1       * Error ia +ve; subtract threshold
        BLZ    ERRETN       * Error  > trshld
        SUB    TRSHD1       * see if error >> trshld
        BLZ    ADD1A        * No...add one to index
        LAC    RALPHA       * Yes...add 2 to index
        ADD    ONE,9        * SUB 2 same as ADD >7E in modulo 128
        B      ERR2         *
ADD1A   LAC    RALPHA       *
        ADD    ONE,8        *
ERR2    AND    MASK1        * Keep RALPHA modulo 128
        SACL   RALPHA       * save new index
        RET                 * Return with corrected RALPHA
ERRETN  LAC    ONE,4        *
        OR     RECST        * If |error| less than threshold
        SACL   RECST        * set flag in status register
RETA    RET
*****************************************************************
*****    BAUD  CLOCK  ALLIGNMENT                          ******
*****************************************************************
BDCLK1 CALL   DEMOD
       LAC    ENRGY        * ENRGY = E(3)
       SACL   PENRGY       * STORE IT IN PENRGY
       RET
BDCLK2 CALL   DEMOD
       LAC    RECST        * TEST IF CORRECTION OF THE
       AND    ONE          * BAUD CLOCK IS MADE
       BNZ    RETB         * IF SO THEN RETURN
       LAC    ENRGY        * ENRGY = E(11), PENRGY = E(3)
       SUB    PENRGY       * FORM ERROR SIGNAL
       SACL   BERROR       * BERROR = E(11)-E(3)
********-------------------------------------------**********
****    LOOP  FILTER                                 ********
```

```
*********-------------------------------------------**********
        ZAC
        LT    BPLL2
        MPY   BERROR
        LTA   BPLL1
        MPY   BEROUT
        APAC
        SACH  BEROUT,1        * BEROUT IN Q14
*
*---APPLY  CORRECTION
*
        LAC   BEROUT
        BGEZ  POS             * TEST BERROUT SIGN.
        ADD   TRSHD2
        BGEZ  RETB            * IF |BERROUT|<TRSHD RETURN.
        ADD   TRSHD2          * BERROUT IS NEGATIVE. THEREFORE
        BGEZ  SUB1B           * ADJUST CLOCK BY DELAYING SAMPLE COUNT.
        LAC   SAMPLE          * IF |BERROUT|>2*TRSHD
        SUB   ONE,1           * MAKE TWO SAMPLE ADJUSTMENT
        SACL  SAMPLE          * OF THE SAMPLE (BAUD CLOCK)
        B     RETB            * COUNT.
SUB1B   LAC   SAMPLE          * IF TRSHD<|BERROUT|<2*TRSHD
        SUB   ONE             * MAKE ONE SAMPLE ADJUSTMENT
        SACL  SAMPLE          * OF THE SAMPLE (BAUD CLOCK)
        B     RETB            * COUNT.
POS     SUB   TRSHD2          * BERROUR IS POSITIVE. THEREFORE
        BLZ   RETB            * ADJUST CLOCK BY ADVANCING SAMPLE
        SUB   TRSHD2          * COUNT.
        BLZ   ADD1B           * IF |BERROUT|>2*TRSHD
        LAC   SAMPLE          * MAKE TWO SAMPLE ADJUSTMENT
        ADD   ONE,1           * OF THE SAMPLE (BAUD CLOCK)
        SACL  SAMPLE          * COUNT.
        B     RETB
ADD1B   LAC   SAMPLE          * IF TRSHD<|BERROUT|<2*TRSHD
        ADD   ONE             * MAKE ONE SAMPLE ADJUSTMENT
        SACL  SAMPLE          * OF THE SAMPLE (BAUD CLOCK) COUNT.
RETB    LAC   RECST           * SET FLAG TO INDICATE THAT THE BAUD
        OR    ONE             * CLOCK ADJUSTMENT IS MADE.
        SACL  RECST
        RET
*******************************************************************************
        END
```

# Appendix E
# TMS7742 Source Code

```
        TITL      TMS 7742 MODEM INTERFACE PROGRAM'
        OPTION  XREF,TUNLST
*               7042 PORT ASSIGNMENTS
*
*
* APORT
*           A7   A6   A5   A4   A3   A2   A1   A0
*           OHR_ N.C. RCVD ATE_ A_/O SQT  DTR  DCD
*           (O)  (X)  (I)  (O)  (O)  (O)  (I)  (I)
*
* BPORT
*           B7   B6   B5   B4   B3   B2   B1   B0
*           NB8  NB4  NB2  NB1  TXD  DP   DSR  CTS
*           (O)  (O)  (O)  (O)  (O)  (O)  (O)  (O)
*
* CPORT
*           C7   C6   C5   C4   C3   C2   C1   C0
*           ACKW ACKR CMD2 CMD1 TDB3 TDB2 TDB1 TDB0
*           (O)  (O)  (O)  (O)  (O)  (O)  (O)  (O)
*
* DPORT
*           D7   D6   D5   D4   D3   D2   D1   D0
*           NEWO NEWI CDT  ENB  RDB3 RDB2 RDB1 RDB0
*           (I)  (I)  (I)  (I)  (I)  (I)  (I)  (I)
*
*-------------------------------------------------------
*           +-------------------------------+
* SWSTAT    | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
*           +-------------------------------+
*
* BIT7: modem type        0= B103 mode
*                         1= B212 mode
*
* BIT6: timer flag        0= carrier wait timer enabled
*                         1= 1200 Hz timer enabled
*
* BIT5: 1st dibit flag    0= flag reset
*                         1= flag set
*-------------------------------------------------------
          IDT 'DSPMODM'
*         OPTION  XREF
          AORG    >F006
*
* 7041 Peripheral Memory Symbols
*
IOCNT0 EQU P0
T1DATA EQU P2
T1CNTL EQU P3
APORT  EQU P4
ADDR   EQU P5
BPORT  EQU P6
CPORT  EQU P8
CDDR   EQU P9
DPORT  EQU P10
DDDR   EQU P11
IOCNT1 EQU P16
SMODE  EQU P17
SCTL0  EQU P17
SSTAT  EQU P17
T2DATA EQU P18
T2CNTL EQU P19
T3DATA EQU P20
SCTL1  EQU P21
RXBUF  EQU P22
TXBUF  EQU P23
MPRTC  EQU >108
MPRTD  EQU >10A
*
* Bit Masks.
*
```

```
BIT0    EQU >01
BIT1    EQU >02
BIT2    EQU >04
BIT3    EQU >08
BIT4    EQU >10
BIT5    EQU >20
BIT6    EQU >40
BIT7    EQU >80
*
NOT0    EQU >FE
NOT1    EQU >FD
NOT2    EQU >FB
NOT3    EQU >F7
NOT4    EQU >EF
NOT5    EQU >DF
NOT6    EQU >BF
NOT7    EQU >7F
*
* Ascii constants
*
TAB     EQU >09                 ; tab character
BLANK   EQU >20                 ; space character
COMMA   EQU >2C                 ; ','
LF      EQU 10
CR      EQU 13
BS      EQU 8                   ; BACKSPACE CHARACTER
POUND   EQU >23                 ; '#'
STAR    EQU >2A                 ; '*'
*
ISA     EQU >41                 ; 'A'
ISZ     EQU >5A                 ; 'Z'
*
* 7041 RAM map
*
RDIBIT EQU R2
DIBIT2 EQU RDIBIT+1             ; receiver input sequence
DIBIT1 EQU DIBIT2+1             ; from the 32010
TEMP1  EQU DIBIT1+1             ; temporary register (receiver)
TEMP2  EQU TEMP1+1              ; temporary register (xmitter)
RBIT14 EQU TEMP2+1
RBIT17 EQU RBIT14+1
DESREG EQU RBIT17+1
FLAG   EQU DESREG+1
COUNTE EQU FLAG+1
BITCNT EQU COUNTE+1
CHRCNT EQU BITCNT+1
TRNMIT EQU CHRCNT+1             ; character to be transmitted
STPFLG EQU TRNMIT+1             ; stop bit deleted flag
MCOUNT EQU STPFLG+1             ; mark counter
XDBIT2 EQU MCOUNT+1
XDBIT1 EQU XDBIT2+1
XBIT17 EQU XDBIT1+1
XBIT14 EQU XBIT17+1
SCMREG EQU XBIT14+1
XDIBIT EQU SCMREG+1             ; xmitter input dibit
CONTER EQU XDIBIT+1             ; counter register
RBTCNT EQU CONTER+1             ; character bit counter (xmt)
XMTCHR EQU RBTCNT+1             ; input character buffer
ADDRES EQU XMTCHR+2             ; command buffer address pointer
PNTR   EQU ADDRES+1             ; counter register
SWSTAT EQU PNTR+1               ; software status flag
LOCHI  EQU SWSTAT+1
LOCLO  EQU LOCHI+1
ADDR1  EQU LOCLO
INDEX1 EQU ADDR1+1              ; general purpose register
INDEX2 EQU INDEX1+1             ; general purpose register
COUNT1 EQU INDEX2+1
COUNT  EQU COUNT1+1             ; general use double register counter
```

```
COMBUF  EQU  COUNT+1              ; beginning of the command buffer
S0      EQU  COMBUF+40           ; carriage return register
S1      EQU  S0+1                ; line feed register
S2      EQU  S1+1                ; backspace register
S3      EQU  S2+1                ; # of rings to answer on
S4      EQU  S3+1                ; # of rings detected
S5      EQU  S4+1                ; escape code character
INT5TM  EQU  S5+1                ; interrupt 5 timer register
VALUE   EQU  INT5TM+1            ; contains numerical value of parameters
MSGM    EQU  VALUE+1
MSGL    EQU  MSGM+1              ; masseage address register
CWT1    EQU  MSGL+1
CWT2    EQU  CWT1+1              ; carrier wait abort timer
MSTIME  EQU  CWT2+2              ; millisec timing register
DELYR1  EQU  MSTIME+1
STACK   EQU  R100
*
ALL     EQU  >FF
ZERO    EQU  >00
ONE     EQU  >01
TWO     EQU  >02
THREE   EQU  >03
EIGHT   EQU  >08
NINE    EQU  >09
TEN     EQU  >0A
CNTVAL  EQU  >DC
ADDTOP  EQU  S0-1
ADDBOT  EQU  COMBUF
*******-----------------------------------------*******
*****              Initialization                *****
*******-----------------------------------------*******
INIT    MOVP   %>0F,IOCNT0       ; s.c. mode, enable int2 and int1
        MOVP   %>00,IOCNT1       ; disable int4 and int5
        MOVP   %>0C,APORT
        MOVP   %>9C,ADDR         ; set direction of APORT
        MOVP   %>FB,BPORT
        MOVP   %>CF,CPORT
        MOVP   %>FF,CDDR         ; set direction of CPORT
        MOVP   %>CF,DPORT
        MOVP   %>00,DDDR         ; set direction of DPORT
        MOVP   %155,T2DATA
        MOVP   %>81,T2CNTL
        ORP    %BIT2,BPORT       ; reset the 99531 dialer
        ANDP   %NOT2,BPORT
        MOV    %STACK,B
        LDSP                     ; load stack pointer
        MOV    %ALL,DIBIT1
        MOV    %ALL,DIBIT2
        MOV    %ONE,RBIT17
        MOV    %ONE,RBIT14
        MOV    %ALL,XDBIT1
        MOV    %ALL,XDBIT2
        MOV    %ONE,XBIT17
        MOV    %ONE,XBIT14
*
        MOV    %CR,S0            ; carriage return character
        MOV    %LF,S1            ; line feed character
        MOV    %BS,S2            ; backspace character
        MOV    %ONE,S3           ; # of rings to answer on
        CLR    S4                ; # of rings detected
        MOV    %'+',S5           ; escape code character
        MOV    %>C0,SWSTAT       ; software flag default conditions
*
* main routine
*
        ANDP   %NOT0,BPORT       ; set CTS_
        EINT
*
```

```
        CALL    @AUTOBD             ; Autobaud to terminal speed
TOP     MOVD    %HELLO,MSGL         ; Send hello message
        CALL    @PRINT
*
* look for input commands.
*
LOOK    CALL    @CLEAR              ; clear the command buffer
        MOVD    %ADDTOP,ADDRES      ; point to top of the buffer
        CLR     PNTR                ; clear buffer command pointer
LK4COM  BTJZP   %BIT1,SSTAT,LK4COM  ; command received?
        MOVP    RXBUF,A
        MOVP    A,TXBUF             ; echo
WAIT4   BTJZP   %BIT2,SSTAT,WAIT4
        CMP     %CR,A               ; last character?
        JEQ     EXEC                ; yes, go execute command
*
        CMP     %'(',A              ; ignore
        JEQ     LK4COM
        CMP     %')',A              ; ignore
        JEQ     LK4COM
        CMP     %'-',A              ; ignore
        JEQ     LK4COM
        CMP     %' ',A              ; ignore
        JEQ     LK4COM
        CMP     %'/',A              ; ignore
        JEQ     LK4COM
        CMP     %BS,A               ; backspace?
        JNE     NXTSTG              ; yes, go get new command
        DEC     PNTR                ; decrement pointer
        CLR     A                   ; CLEAR OUT THE BUFFER
        STA     *ADDRES             ; AT THE CURRENT LOCATION
        INC     ADDRES              ; point to the previous location
        JMP     LK4COM
*
NXTSTG  INC     PNTR                ; command buffer pointer
        STA     *ADDRES             ; location for command
        DECD    ADDRES              ; location for next command
        CMP     %40,PNTR            ; allow 40 chars maximum
        JEQ     ERR                 ; more than 40..clear buffer
        JMP     LK4COM              ; keep going till <CR>
*
ERR     CALL    @CLEAR              ; clear command buffer
        MOVD    %ERROR,MSGL         ; send error message
        CALL    @PRINT
        MOV     %STACK,B            ; reset the stack pointer
        LDSP
        BR      @LOOK
*
EXEC    MOVD    %ADDTOP,ADDRES      ; initialize address point
        LDA     *ADDRES             ; get command
        CMP     %'A',A
        JL      ERR                 ; Check for A thru Z
        CMP     %'Z'+1,A
        JHS     ERR
        CLR     B                   ; Parameter buffer pointer
        DECD    ADDRES
        SUB     %'A',A
        MOV     A,B
        RL      B                   ; B*2
        LDA     @COMLIS(B)
        MOV     A,LOCHI             ; MSB address
        INC     B
        LDA     @COMLIS(B)
        MOV     A,LOCLO             ; LSB address
        BR      *ADDR1              ; execute command
*
        PAGE
```

```
*******-----------------------------------------*******
*****     Local Digital Loopback Test             *****
*******-----------------------------------------*******
*
LDLB    EQU   $
        MOVD  %LDLBM,MSGL          ; RESPOND TO COMMAND TO DTE
        CALL  @PRINT               ; BY PRINTING TEST CODE
        MOV   %>10,R23             ; SET COMMAND TO LDLB MODE
        BR    @GO320               ; AND RUN THE 320
*
        PAGE
*******-----------------------------------------*******
*****                Dial Blind                   *****
*******-----------------------------------------*******
*
DB      OR    %BIT6,SWSTAT
        MOVP  %>2A,IOCNT0          ; disable RI interrupt
        ORP   %BIT0,BPORT          ; turn off CTS_
        ORP   %>8C,APORT           ; originate mode, squelch 532,
*                                    and go off hook
        CALL  @DIAL                ; dial
        MOV   %18,CWT1             ; initialize carrier abort timer.
        CLR   CWT2
        ORP   %BIT2,IOCNT0         ; enable carrier abort interrupt
CHKDCD  BTJOP %BIT0,APORT,CHKDCD   ; wait for DCD_
        AND   %NOT6,SWSTAT
        BTJZ  %BIT7,SWSTAT,B103    ; check for modem type
        ORP   %BIT3,CPORT          ; 32010 in B212 originate mode
        ORP   %BIT2,CPORT
        BR    @B212
*******-----------------------------------------*******
*****       Bell 103 Call Initiation.             *****
*******-----------------------------------------*******
B103    BTJOP %BIT0,APORT,B103     ; Wait for DCD_
*
* Send originate tone.
*
        ORP   %BIT4,APORT          ; ATE = 1
        ANDP  %NOT2,APORT          ; unsquelch 532.
*
        MOVP  %>4A,IOCNT0          ; got DCD_, disable abort interrupt
*
* Wait 800ms
*
        MOVD  %800,MSTIME
        CALL  @MSDLY
        ANDP  %NOT0,BPORT          ; activate CTS_
        MOVD  %CONN3,MSGL          ; send connect 300 message
        CALL  @PRINT
        BR    @DAT103              ; enter data mode
*******-----------------------------------------*******
*****   DIAL - Dial number stored in ADDRES.      *****
*******-----------------------------------------*******
DIAL    ANDP  %NOT4,APORT          ; ATE_ = 0, enable EXI mode
*
* Execute dialing.
*
        MOVD  %4000,MSTIME         ; Initial dial tone wait of 2 second
        CALL  @MSDLY
NXTDIG  LDA   *ADDRES              ; Load subcommand
        CMP   %ZERO,A              ; is it the last command?
        JNE   NOTEND
*
* End of dialing.
*
        RETS
*
```

```
* Case statement to determine subcommand.
*
NOTEND DECD   ADDRES              ; update address
       CMP    %'0',A              ; check less than '0'
       JL     NOTNUM
       CMP    %'9'+1,A            ; check greater than '9'
       JHS    NOTSPC
       BR     @ISANUM
NOTNUM CMP    %',',A              ; ',' - dial tone wait
       JEQ    DPAUSE
       CMP    %STAR,A             ; '*' - tone dial *
       JEQ    ISSTAR
       CMP    %POUND,A            ; '#' - tone dial #
       JEQ    APOUND
NOTSPC BR     @NXTDIG
*
* Wait for a dial tone.
*
DPAUSE MOV    %TWO,VALUE          ; Blind delay
       CALL   @SECDLY
       BR     @NXTDIG
*
* Dial a digit.
*
ISSTAR MOV    %TEN,A              ; dial * if tone dial
       JMP    OUTDIG
APOUND MOV    %11,A               ; dial # if tone dial
       JMP    OUTDIG
ISANUM SUB    %'0',A              ; dial a number
OUTDIG ANDP   %>0F,BPORT          ; clear old digit
       RL     A                   ; get the correct value
       RL     A
       RL     A
       RL     A
       ORP    A,BPORT             ; send new digit
PNDWT0 BTJZP  %BIT1,APORT,PNDWT0  ; wait for acceptance
       ORP    %BIT2,BPORT         ; set DP
PNDWT1 BTJOP  %BIT1,APORT,PNDWT1  ; wait for PND low
       ANDP   %NOT2,BPORT         ; clear DP
PNDWT2 BTJZP  %BIT1,APORT,PNDWT2  ; wait for PND high
       BR     @NXTDIG
*
       PAGE
*******-----------------------------------------*******
*****      BELL 1200 BPS MODEM ALGORITHM          *****
*******-----------------------------------------*******
*
B212   EQU    $
       MOV    %>20,R23            ; SET COMMAND TO MODEM RUN
GO320  CLR    R2                  ; CLEAR COM STATUS REG
       CLR    R11                 ; INITIALIZE SCRAMBLER HISTORY
       CLR    R12                 ; AS ALL ZEROS
       CLR    R18                 ; INITIALIZE DESCRAMBLER HISTORY
       CLR    R19                 ; AS ALL ZEROS
       CLR    R13                 ; INITIALIZE DESCRAMBLER HISTORY
       CLR    R20                 ; AS ALL ZEROS
       ANDP   %NOT0,BPORT         ; ACTIVATE CTS TO DTE
       CLR    A                   ; CYCLE THE CLEAR LINES
       STA    @MPRTC              ; OF THE I/O CONTROL
       ORP    %>C0,CPORT          ; RESET 320 ACK LINES
*
*    START UP MODEM OR DLB TEST
*
STOPB2 MOV    %3,R10              ; SET DIBIT TO MARKS
       CALL   @SCRAM              ; AND SCRAMBLE IT
       MOV    R10,A               ; HOLD IT FOR TRANSMIT
       OR     R23,A               ; OR IN COMMAND BITS
```

```
        ANDP  %>C0,CPORT          ; CLEAR OFF CURRENT BITS
        ORP   A,CPORT             ; SEND OUT SCRAM MARKS
*   TRANSMIT UNSCRAMBLED MARKS AND RECEIVE
MRC1    BTJZP %BIT7,DPORT,MRC2    ; WAIT FOR WRITE FROM 320
CHKTCH  BTJOP %BIT6,DPORT,RECDTE  ; WAIT FOR READ FROM 320
        BR    @MRC3               ; PROCESS READ FROM 320
RECDTE  BTJOP %BIT1,SSTAT,DTEGET  ; IS DTE REC BUF FULL
XMTDTE  BTJOP %BIT0,SSTAT,DTEPUT  ; IS DTE TRANS BUF EMPTY
        JMP   MRC1                ; LOOK AGAIN
*
*     CODE INTERFACE TO DTE
*
DTEGET  EQU   $
        MOVP  RXBUF,A             ; YES, GET THE CHARACTER?
        CMP   %>1B,A              ; IF A <> ESCAPE
        JMP   OVRSQT              ; THEN CONTINUE
        CLR   A                   ; ELSE SQUELCH THE
        STA   @MPRTC              ; THE 320 MODEM AND
        BR    @TOP                ; AND RETURN TO MONITOR
OVRSQT  INC   R24                 ; INCREMENT BYTE COUNT
        BTJO  %BIT5,R2,DTEGER     ; CHECK FOR BUF2 FULL
        BTJO  %BIT3,R2,DTEG1      ; CHECK IF 1ST CHAR
        OR    %BIT7,R2            ; FLAG FOR START BIT
        MOV   A,R7                ; IF SO THEN RESTART
        MOV   %>A,R21             ; RESET XMT COUNT
        OR    %BIT3,R2            ; SET TRANS ACTIVE
        JMP   XMTDTE              ; CHECK OUTPUT
DTEG1   MOV   A,R28               ; SAVE IT IN THE BUF2
        OR    %BIT5,R2            ; SET BUF2 FULL FLAG
        JMP   XMTDTE              ; CHECK OUTPUT
*
DTEGER  CLR   A                   ; SQUELCH THE
        STA   @MPRTC              ; 320 MODEM
        MOVD  %BUFERR,MSGL        ; SEND ERROR MESSAGE
        CALL  @PRINT              ; TO USER TERMINAL
        BR    @TOP                ; EXIT ROUTINE
*
DTEPUT  EQU   $
        BTJZ  %BIT4,R2,MRC1       ; CHECK FOR CHARACTER READY
        MOV   R29,A               ; GET BUFFERED CHARACTER
        MOVP  A,TXBUF             ; SEND IT TO THE DTE
        AND   %NOT4,R2            ; RESET BUFFER FULL FLAG
        JMP   MRC1                ; RETURN TO FLAG LOOP
*
        PAGE
*
*     RECEIVE DIBITS FROM THE 320
*
MRC2    ANDP  %>7F,CPORT          ; RESET WRITE ACKNOWLEDGE
        ORP   %>80,CPORT          ; BY TOGGLING LINES
        MOVP  DPORT,A             ; GET THE RETURNED DATA
        MOV   A,R10               ; AND HOLD IT IN R10
        BTJZ  %BIT5,A,CHKTCH      ; IF NO CARRIER THEN DONE
        AND   %3,R10              ; AND OFF STATUS
        CALL  @DSCRAM             ; DESCRAMBLE IT
        BTJO  %BIT2,R2,RCHAR1     ; CHECK FOR REC CHAR ACTIVE
        RRC   R10                 ; CHECK DIBIT0
        JC    RNB                 ; IF HIGH THEN CHECK NEXT
        RRC   R10                 ; SAVE LSB OF RECEIVE CHAR
        RRC   R5                  ; IN CHAR HOLD REG
        MOV   %7,R22              ; SET REC BIT COUNT REG
        JMP   RCHAR0              ; SKIP OVER NEXT CHECK
*
RNB     RRC   R10                 ; CHECK DIBIT1
        JC    CHKTCH              ; IF HIGH THEN CHECK XMTCHAR
        MOV   %8,R22              ; SET REC BIT COUNT REG
RCHAR0  OR    %BIT2,R2            ; SET REC CHAR ACTIVE
        BR    @RECDTE             ; CHECK DTE
```

```
*
RCHAR1 SUB    %2,R22            ; CHECK BIT POSITION
       JP     RCHAR3            ; IF > 0 GET 2 BITS
       JZ     RCHAR2            ; IF = 0 GET 1 BIT
       RRC    R10               ; PUT BIT7 INTO
       RRC    R5                ; REC CHAR HOLD REG
       MOV    R5,R29            ; PUT CHAR IN OUT BUFFER
       OR     %BIT4,R2          ; SET BUFFER FULL FLAG
       CLR    R5                ; CLEAR BUFFER FOR NEXT CHAR
       AND    %NOT2,R2          ; RESET REC CHAR ACTIVE
       JMP    RNB               ; CHECK DIBIT1 FOR START BIT
*
RCHAR2 RRC    R10               ; SAVE MSB OF RECEIVE CHAR
       RRC    R5                ; INTO REC CHAR HOLD REG
       RRC    R10               ; PUT BIT7 INTO
       RRC    R5                ; REC CHAR HOLD REG
       MOV    R5,R29            ; PUT CHAR IN OUT BUFFER
       OR     %BIT4,R2          ; SET BUFFER FULL FLAG
       CLR    R5                ; CLEAR BUFFER FOR NEXT CHAR
       AND    %NOT2,R2          ; RESET REC CHAR ACTIVE
       BR     @RECDTE           ; CHECK DTE
*
RCHAR3 RRC    R10               ; MOVE DIBIT0 TO
       RRC    R5                ; REC CHAR HOLD REG
       RRC    R10               ; MOVE DIBIT1 TO
       RRC    R5                ; REC CHAR HOLD REG
       BR     @RECDTE           ; CHECK DTE
*
       PAGE
*
*   SEND DIBITS TO THE 320
*
MRC3   ANDP   %>BF,CPORT        ; RESET ACKNOWLEDGE
       ORP    %>40,CPORT        ; BY TOGGLING LINES
       BTJO   %BIT3,R2,TCHAR0   ; CHECK FOR TRANS CHAR ACTIVE
       BR     @STOPB2           ; IF NOT SEND STOPBITS
TCHAR0 CLR    R10               ; CLEAR OUT DIBIT REG
       SUB    %2,R21            ; CHECK POSITION
       JP     TCHAR6            ; > 2 MEANS TRANSMIT BITS
       JNZ    TCHAR3            ; IF PATTERN ONE THEN ODD
       RRC    R7                ; GET BIT 7 FROM CHAR
       JNC    TCHO0             ; IF NO CARRY DIBIT0=0
       OR     %BIT0,R10         ; ELSE DIBIT0=1
TCHO0  BTJO   %BIT5,R2,TCHAR1   ; IF BUF2 EMPTY
       AND    %NOT3,R2          ; RESET TRAN ACTIVE BIT
       OR     %BIT1,R10         ; SET DIBIT1 TO STOP
       JMP    TCHSND            ; AND SEND DIBIT
TCHAR1 CMP    %9,R24            ; CHECK CHAR COUNT
       JL     TCHAR2            ; IF < DON'T DELETE STOPBIT
       CLR    R24               ; CLEAR BYTE COUNT
       AND    %NOT1,R10         ; SEND DIBIT1 TO START
       MOV    R28,R7            ; LOAD IN NEW CHAR
       MOV    %9,R21            ; SET BIT COUNT
       AND    %NOT5,R2          ; RESET BUF2 FULL FLAG
       JMP    TCHSND            ; SEND THE DIBIT
TCHAR2 OR     %BIT1,R10         ; SEND DIBIT1 TO STOP
       MOV    R28,R7            ; LOAD IN NEW CHAR
       OR     %BIT7,R2          ; FLAG IN START BIT
       MOV    %>A,R21           ; SET BIT COUNT
       AND    %NOT5,R2          ; RESET BUF2 FULL FLAG
       JMP    TCHSND            ; SEND THE DIBIT
TCHAR3 BTJO   %BIT5,R2,TCHAR4   ; IF BUF2 EMPTY
       AND    %NOT3,R2          ; RESET TRAN ACTIVE BIT
       BR     @STOPB2           ; AND SEND MARKS
TCHAR4 CMP    %9,R24            ; CHECK CHAR COUNT
       JL     TCHAR5            ; IF < DON'T DELETE STOPBIT
       CLR    R24               ; CLEAR BYTE COUNT
```

```
        MOV   R28,R7              ; LOAD IN NEW CHAR
        OR    %BIT7,R2            ; FLAG IN START BIT
        MOV   %8,R21              ; SET BIT COUNT
        AND   %NOT5,R2            ; RESET BUF2 FULL FLAG
        JMP   TCHAR6              ; SEND THE DIBIT
TCHAR5  MOV   %1,R10              ; SEND STOP THEN START
        MOV   R28,R7              ; LOAD IN NEW CHAR
        MOV   %9,R21              ; SET BIT COUNT
        AND   %NOT5,R2            ; RESET BUF2 FULL FLAG
        JMP   TCHSND              ; SEND THE DIBIT
TCHAR6  BTJZ  %BIT7,R2,TCHAR7     ; START BIT NEEDED
        AND   %NOT7,R2            ; RESET START BIT FLAG
        JMP   TCHO1               ; SKIP DIBIT1
TCHAR7  RRC   R7                  ; GET NEXT BIT OF CHAR
        JNC   TCHO1               ; IF LOW SKIP BIT SET
        OR    %1,R10              ; ELSE SET DIBIT0 TO ONE
TCHO1   RRC   R7                  ; GET NEXT BIT OF CHAR
        JNC   TCHSND              ; IF LOW SKIP BIT SET
        OR    %2,R10              ; ELSE SET DIBIT1 TO ONE
TCHSND  EQU   $
        CALL  @SCRAM              ; AND SCRAMBLE IT
        MOV   R10,A               ; HOLD IT FOR TRANSMIT
        ANDP  %>F0,CPORT          ; CLEAR OUT DIBIT VALUE
        ORP   A,CPORT             ; SEND TO PORT
        BR    @RECDTE             ; WAIT FOR RETURN LOOP
*
        PAGE
********-------------------------------------------********
*****         Receiver descrambler                    *****
*****      X(N) = Y(N-17) XOR Y(N-14) XOR Y(N)        *****
********-------------------------------------------********
DSCRAM  EQU    $
*
        MOV   R10,B         ; SAVE SCRAMBLED DIBIT
        CLR   R16           ; CLEAR THE Y(N-14) REFERENCE
        CLR   R17           ; CLEAR THE Y(N-17) REFERENCE
        MOV   R11,A         ; GET THE DESCRAMBLER HISTORY
        RL    A             ; SHIFT OUT Y(N-18)
        RLC   A             ; GET HISTORY Y(N-17)
        RLC   R17           ; AND PUT INTO REFERENCE
        RLC   A             ; SHIFT OFF TWO MORE BITS
        RLC   R17           ; SAVE Y(N-16) REFERENCE
        RLC   A             ; TO GET TO THE Y(N-14)
        RLC   A             ; AND GET HISTORY
        RLC   R16           ; AND PUT INTO REFERENCE
        RLC   A             ; GET HISTORY Y(N-13)
        RLC   R16           ; AND PUT INTO REFERENCE
        XOR   R16,R10       ; R10=X(N) XOR Y(N-14)
        XOR   R17,R10       ; R10=X(N) XOR Y(N-14) XOR Y(N-17)
        CLRC                ; CLEAR OUT THE CARRY BIT
        RRC   R10           ; REVERSE THE DIBITS FOR
        JNC   OVRSW1        ; ALLIGNMENT WITH SCRAMBLER
        OR    %2,R10        ; IF CARRY THEN BIT HIGH
OVRSW1  EQU   $
*
        RLC   R13           ; SHIFT UP THE LSB HISTORY BITS
        RLC   R12           ; AND CARRY TO CSB HISTORY BITS
        RLC   R11           ; AND CARRY TO MSB HISTORY BITS
        CLRC                ; CLEAR THE CARRY BIT
        RLC   R13           ; SHIFT UP THE LSB HISTORY BITS
        RLC   R12           ; AND CARRY TO CSB HISTORY BITS
        RLC   R11           ; AND CARRY TO MSB HISTORY BITS
        RRC   B             ; GET DIBIT0 AND
        RRC   R13           ; AND SHIFT IT INTO R13
        RRC   B             ; GET DIBIT0 AND
        RRC   R13           ; AND SHIFT IT INTO R13
*
        RETS
```

```
*
        PAGE
********-----------------------------------------********
*****          Transmitter Scrambler                *****
*****       Y(N) = Y(N-17) XOR Y(N-14) XOR X(N)     *****
********-----------------------------------------********
SCRAM   EQU   $
        CLRC                ; CLEAR OUT THE CARRY BIT
        RRC   R10           ; REVERSE THE DIBITS FOR
        JNC   OVRSW2        ; ALLIGNMENT WITH SCRAMBLER
        OR    %2,R10        ; IF CARRY THEN BIT HIGH
OVRSW2  EQU   $
        CLR   R16           ; CLEAR THE Y(N-14) REFERENCE
        CLR   R17           ; CLEAR THE Y(N-17) REFERENCE
        MOV   R18,A         ; GET THE SCRAMBLER HISTORY
        RL    A             ; SHIFT OUT Y(N-18)
        RLC   A             ; GET HISTORY Y(N-17)
        RLC   R17           ; AND PUT INTO REFERENCE
        RLC   A             ; SHIFT OFF TWO MORE BITS
        RLC   R17           ; SAVE Y(N-16) REFERENCE
        RLC   A             ; TO GET TO THE Y(N-14)
        RLC   A             ; AND GET HISTORY
        RLC   R16           ; AND PUT INTO REFERENCE
        RLC   A             ; GET HISTORY Y(N-13)
        RLC   R16           ; AND PUT INTO REFERENCE
        XOR   R16,R10       ; R10=X(N) XOR Y(N-14)
        XOR   R17,R10       ; R10=X(N) XOR Y(N-14) XOR Y(N-17)
*
        MOV   R10,B         ; HOLD SCRAMBLED DIBIT FOR HISTORY
        RLC   R20           ; SHIFT UP THE LSB HISTORY BITS
        RLC   R19           ; AND CARRY TO CSB HISTORY BITS
        RLC   R18           ; AND CARRY TO MSB HISTORY BITS
        CLRC                ; CLEAR CARRY BIT
        RLC   R20           ; SHIFT UP THE LSB HISTORY BITS
        RLC   R19           ; AND CARRY TO CSB HISTORY BITS
        RLC   R18           ; AND CARRY TO MSB HISTORY BITS
        RRC   B             ; GET DIBIT0 AND
        RRC   R20           ; AND SHIFT IT INTO R20
        RRC   B             ; GET DIBIT0 AND
        RRC   R20           ; AND SHIFT IT INTO R20
*
        RETS
*
        PAGE
********-----------------------------------------********
***** MSDLY - Wait MSTIME number of milliseconds *****
********-----------------------------------------********
MSDLY   EQU   $
        MOV   %CNTVAL,DELYR1    ; load the inner counter (9)
HERE2   DJNZ  DELYR1,HERE2      ; (9+2)
        DECD  MSTIME            ; (11)
        JC    MSDLY             ; (7)
        RETS
********-----------------------------------------********
*****   SECDLY - Wait VALUE number of seconds     *****
********-----------------------------------------********
SECDLY  CMP   %0,VALUE
        JEQ   NODLY
NXTSEC  MOVD  %1001,MSTIME
        CALL  @MSDLY
        DJNZ  VALUE,NXTSEC
NODLY   RETS
********-----------------------------------------********
*****              PRINT  subroutine              *****
********-----------------------------------------********
* MSGM and MSGL contain the address of text to print
* for messages to the screen
*
```

```
PRINT   CALL    @CRLF
PRINTI  LDA     *MSGL
        JZ      WAIT6
        MOVP    A,TXBUF             ; print each character in text statement
WAIT5   BTJZP   %BIT0,SSTAT,WAIT5 ; wait for txbuf ready
        INC     MSGL
        ADC     %0,MSGM
        JMP     PRINTI
WAIT6   CALL    @CRLF
        RETS
*
* send carriage return/line feed
*
CRLF    MOV     S0,A
        MOVP    A,TXBUF             ; send carriage return
CRWAIT  BTJZP   %BIT0,SSTAT,CRWAIT
        MOV     S1,A
        MOVP    A,TXBUF             ; send line feed
LFWAIT  BTJZP   %BIT2,SSTAT,LFWAIT
        RETS
*
        PAGE
********-----------------------------------------********
*****                PRINT  subroutine                *****
********-----------------------------------------********
AUTOBD  EQU     $
        MOV     %>20,A              ; SET BAUD CLOCK FOR
        MOVP    A,T3DATA            ; FOR OVERSPEED DTE
SETMOD  MOVP    %0,P17              ; Write to P17 to guarantee
        MOVP    %>60,SCTL0          ; we are talking to SCTL0, then reset
*                                   ; serial port
        MOVP    B,SMODE             ;
        MOVP    %>15,SCTL0          ;
        MOVP    %>40,SCTL1          ;
*
        MOVP    %BIT6,SCTL0         ; Parity error, parity is disabled in DTE.
        MOVP    %>6E,SMODE          ; Disable parity of port
        MOVP    %>15,SCTL0
        MOVP    %>40,SCTL1
        RETS
*
********-----------------------------------------********
*****     screen messages - text statements        *****
********-----------------------------------------********
ERROR   TEXT    'ERROR'
        BYTE    0
*
BUFERR  TEXT    'DTE BUFFER OVERFLOW ERROR'
        BYTE    0
*
CONN12  TEXT    'CONNECT 1200'
        BYTE    0
*
CONN3   TEXT    'CONNECT 300'
        BYTE    0
*
NOCAR   TEXT    'NO CARRIER'
        BYTE    0
*
RCALL   TEXT    'RING'
        BYTE    0
*
RESET   TEXT    'OK'
        BYTE    0
*
LDLBM   TEXT    'EXECUTE LDLB, ENTER CHARACTERS'
        BYTE    0
*
```

```
IALBM   TEXT    'INITIALIZE 320 FOR ALB TEST'
        BYTE    0
*
IORGM   TEXT    'INITIALIZE 320 FOR ORIGINATE MODE'
        BYTE    0
*
IENBM   TEXT    'INITIALIZE 320 TO REENABLE RECEIVER'
        BYTE    0
*
ISQTM   TEXT    'INITIALIZE 320 TO SQUELCH RECEIVER'
        BYTE    0
*
ANSM    TEXT    'INITIALIZE 320 TO ANSWER MODE'
        BYTE    0
*
HONM    TEXT    'PUT LINE ON HOOK'
        BYTE    0
*
HOFFM   TEXT    'TAKE LINE OFF HOOK'
        BYTE    0
*
HELPM   TEXT    'TABLE OF COMMANDS'
        BYTE    >0D,>0A
        TEXT    'A ==> PUT MODEM IN ANSWER MODE'
        BYTE    >0D,>0A
        TEXT    'D ==> BLIND DIAL FOLLOWING DIGITS'
        BYTE    >0D,>0A
        TEXT    'E ==> ENABLE 320 RECEIVER'
        BYTE    >0D,>0A
        TEXT    'H ==> DISPLAY HELP LIST'
        BYTE    >0D,>0A
        TEXT    'J ==> PUT LINE ON HOOK'
        BYTE    >0D,>0A
        TEXT    'K ==> TAKE LINE OFF HOOK'
        BYTE    >0D,>0A
        TEXT    'L ==> RUN DIGITAL LOOP BACK TEST'
        BYTE    >0D,>0A
        TEXT    'M ==> RUN ANALOG LOOP BACK TEST'
        BYTE    >0D,>0A
        TEXT    'O ==> PUT MODEM IN ANSWER MODE'
        BYTE    >0D,>0A
        TEXT    'R ==> RUN THE 320 MODEM'
        BYTE    >0D,>0A
        TEXT    'S ==> SQUELCH THE 320 RECEIVER'
        BYTE    >0D,>0A
        TEXT    'Z ==> RESTART THE 7000'
        BYTE    >0D,>0A
        BYTE    0
*
HELLO   TEXT    'DSP MODEM. VERSION 1.0'
        BYTE    0
********-----------------------------------------********
*****           command address table             *****
********-----------------------------------------********
COMLIS  DATA ANSMDM              ; INITIALIZE TO ANSWER
        DATA ERR
        DATA ERR
        DATA DB                  ; dial command
        DATA ENBREC              ; REENABLE RECEIVER ON 320
        DATA ERR
        DATA ERR
        DATA HELP                ; HELP LIST
        DATA ERR
        DATA HOOKON              ; TAKE LINE ON HOOK
        DATA HOOKOF              ; TAKE LINE OFF HOOK
        DATA LDLB                ; LOCAL DIGITAL LOOP BACK
        DATA IALB                ; INITIALIZE TO ALB MODE
```

```
        DATA ERR
        DATA IORIG              ; INITIALIZE TO ORIGINATE
        DATA ERR
        DATA ERR
        DATA B212               ; RUN MODEM ROUTINE
        DATA SQTREC             ; SQUELCH THE RECEIVER
        DATA ERR
        DATA ERR
        DATA ERR
        DATA ERR
        DATA ERR
        DATA ERR
        DATA INIT               ; reset command
********-----------------------------------------********
*****     INITIALIZE TO ALB MODE                  *****
********-----------------------------------------********
IALB    EQU   $
        CLR   A             ; CYCLE THE CLEAR LINES
        STA   @MPRTC        ; OF THE I/O CONTROL
        ORP   %BIT0,CPORT   ; SET ALB INIT COMMAND
        ORP   %>F0,CPORT    ; PUT 320 IN INIT COMMAND MODE
IALB1   BTJOP %BIT6,DPORT,IALB1    ; CHECK 320 RESPONSE
        MOVD  %IALBM,MSGL       ; GET CONFIRMATION MESG
        CALL  @PRINT            ; AND SEND IT
        CLR   A                 ; CLEAR OUT THE COMMAND
        STA   @MPRTC            ; FROM I/O LINES
        BR    @TOP              ; EXIT ROUTINE
********-----------------------------------------********
*****     INITIALIZE TO ORIGINATE MODE            *****
********-----------------------------------------********
IORIG   EQU   $
        CLR   A             ; CYCLE THE CLEAR LINES
        STA   @MPRTC        ; OF THE I/O CONTROL
        ORP   %BIT1,CPORT   ; SET ORIG INIT COMMAND
        ORP   %>F0,CPORT    ; PUT 320 IN INIT COMMAND MODE
IORG1   BTJOP %BIT6,DPORT,IORG1    ; CHECK 320 RESPONSE
        MOVD  %IORGM,MSGL       ; GET CONFIRMATION MESG
        CALL  @PRINT            ; AND SEND IT
        CLR   A                 ; CLEAR OUT THE COMMAND
        STA   @MPRTC            ; FROM I/O LINES
        BR    @TOP              ; EXIT ROUTINE
********-----------------------------------------********
*****     INITIALIZE TO RECEIVER SQUELCHED        *****
********-----------------------------------------********
SQTREC  EQU   $
        CLR   A             ; CYCLE THE CLEAR LINES
        STA   @MPRTC        ; OF THE I/O CONTROL
        ORP   %3,CPORT      ; SET SQT INIT COMMAND
        ORP   %>F0,CPORT    ; PUT 320 IN INIT COMMAND MODE
ISQT1   BTJOP %BIT6,DPORT,ISQT1    ; CHECK 320 RESPONSE
        MOVD  %ISQTM,MSGL       ; GET CONFIRMATION MESG
        CALL  @PRINT            ; AND SEND IT
        CLR   A                 ; CLEAR OUT THE COMMAND
        STA   @MPRTC            ; FROM I/O LINES
        BR    @TOP              ; EXIT ROUTINE
********-----------------------------------------********
*****     INITIALIZE TO REENABLE RECEIVER         *****
********-----------------------------------------********
ENBREC  EQU   $
        CLR   A             ; CYCLE THE CLEAR LINES
        STA   @MPRTC        ; OF THE I/O CONTROL
        ORP   %4,CPORT      ; SET ENB INIT COMMAND
        ORP   %>F0,CPORT    ; PUT 320 IN INIT COMMAND MODE
IENB1   BTJOP %BIT6,DPORT,IENB1    ; CHECK 320 RESPONSE
        MOVD  %IENBM,MSGL       ; GET CONFIRMATION MESG
        CALL  @PRINT            ; AND SEND IT
        CLR   A                 ; CLEAR OUT THE COMMAND
```

```
        STA   @MPRTC          ; FROM I/O LINES
        BR    @TOP            ; EXIT ROUTINE
*******-----------------------------------------*******
*****     INITIALIZE TO ANSWER MODE               *****
*******-----------------------------------------*******
ANSMDM  EQU   $
        CLR   A           ; CYCLE THE CLEAR LINES
        STA   @MPRTC      ; OF THE I/O CONTROL
        ORP   %5,CPORT    ; SET ANS INIT COMMAND
        ORP   %>F0,CPORT  ; PUT 320 IN INIT COMMAND MODE
IANS1   BTJOP %BIT6,DPORT,IANS1   ; CHECK 320 RESPONSE
        MOVD  %ANSM,MSGL        ; GET CONFIRMATION MESG
        CALL  @PRINT            ; AND SEND IT
        CLR   A                 ; CLEAR OUT THE COMMAND
        STA   @MPRTC            ; FROM I/O LINES
        BR    @TOP              ; EXIT ROUTINE
*******-----------------------------------------*******
*****     PUT LINE ON HOOK                        *****
*******-----------------------------------------*******
HOOKON  EQU   $
        ANDP  %NOT7,APORT       ; PUT MODEM BACK ON HOOK
        MOVD  %HONM,MSGL        ; GET CONFIRMATION MESG
        CALL  @PRINT            ; AND SEND IT
        BR    @TOP              ; EXIT ROUTINE
*******-----------------------------------------*******
*****     TAKE LINE OFF HOOK                      *****
*******-----------------------------------------*******
HOOKOF  EQU   $
        ORP   %BIT7,APORT       ; TAKE OFF HOOK
        MOVD  %HOFFM,MSGL       ; GET CONFIRMATION MESG
        CALL  @PRINT            ; AND SEND IT
        BR    @TOP              ; EXIT ROUTINE
*******-----------------------------------------*******
*****     DISPLAY HELP LIST                       *****
*******-----------------------------------------*******
HELP    EQU   $
        MOVD  %HELPM,MSGL       ; GET CONFIRMATION MESG
        CALL  @PRINT            ; AND SEND IT
        BR    @TOP              ; EXIT ROUTINE
*******-----------------------------------------*******
*****           Clear command buffer              *****
*******-----------------------------------------*******
CLEAR   CLR   A
        CLR   B
MORE    STA   @ADDBOT(B)          ; zero command register
        INC   B
        CMP   %40,B               ; are we done yet?
        JNE   MORE
        RETS
*******-----------------------------------------*******
*****           Auto-answer routine               *****
*******-----------------------------------------*******
INT1    BTJZP  %BIT1,APORT,ANSMOD ; DTR_ must be active, else return
        RETI
*
ANSMOD  CLR    S4
        ORP    %BIT0,BPORT        ; Turn off CTS_
        MOVP   %>2A,IOCNT0        ; activate timer interrupt
        EINT
RIHIGH  ORP    %BIT1,IOCNT0
        BTJOP  %BIT1,IOCNT0,RIHIGH ; Wait RI to fall
        ORP    %BIT1,IOCNT0
        MOVD   %50,COUNT1
STALOW  MOVD   %10,MSTIME
        CALL   @MSDLY
        BTJOP  %BIT1,IOCNT0,RIHIGH ; separate rings
        DJNZ   COUNT1,STALOW
```

```
*
        MOVD    %RCALL,MSGL         ; send ring message
        CALL    @PRINT
        ORP     %BIT1,IOCNT0
*
LABEL0  INC     S4                  ; increment ring counter
        CMP     S3,S4
        JZ      PICKUP
NXTRNG  MOVD    %100,COUNT1
RILOW   MOVD    %100,MSTIME
        CALL    @MSDLY
        BTJOP   %BIT1,IOCNT0,RIHIGH ; check RI_ every 100 msecs
        DJNZ    COUNT1,RILOW
*
* no rings, abort answer
*
        ANDP    %NOT0,BPORT
        RETI
*
* Pickup the phone and go through answer procedures.
*
PICKUP  ORP     %BIT7,APORT         ; Go off hook
        ORP     %BIT1,BPORT         ; DSR is active
*
*  wait at least 2 seconds for billing delay
*
        MOV     %2,VALUE            ; must wait at least 2 secds
BDELAY  CALL    @SECDLY             ; Wait 2 seconds
        MOV     %18,CWT1            ; Initialize carrier abort timer.
        CLR     CWT2
        ORP     %BIT2,IOCNT0        ; Enable carrier abort interrupt
*
* determine if B212A or B103J mode
*
        ANDP    %BIT4,CPORT         ; answer mode (to 32010)
        ORP     %BIT4,APORT         ; ATE=1
        ANDP    %NOT2,APORT         ; Unsquelch 532, send 2225hz tone
*
        MOVD    %600,INT5TM         ; load timer
*
ORGWT0  BTJZP   %BIT5,DPORT,BE212   ; check for EDT_
        BTJZP   %BIT0,APORT,BE103   ; check for DCD_
        JMP     ORGWT0              ; keep looping till carrier timer aborts
BE212   MOVP    %>0C,IOCNT1         ; enable INT5
        JMP     GOTEDT              ; BELL 212 selected
BE103   MOVP    %>0C,IOCNT1         ; enable INT5
*
* Bell 103J selected
*
        MOV     %150,COUNT
DCDWT0  BTJOP   %BIT0,APORT,ORGWT0  ; check for DCD_
        MOVD    %1,MSTIME
        CALL    @MSDLY
        DJNZ    COUNT,DCDWT0
*
        MOVP    %>00,IOCNT0         ; Got DCD_, disable abort interrupt
        MOVP    %>00,IOCNT1
*
        MOVD    %CONN3,MSGL
        CALL    @PRINT
*
        MOVD    %765,MSTIME
        CALL    @MSDLY
        BR      @DAT103
*
GOTEDT  MOV     %150,COUNT          ; EDT_ active for at least 150 ms
EDTWT2  BTJOP   %BIT5,DPORT,ORGWT0
```

```
        MOVD    %1,MSTIME
        CALL    @MSDLY
        DJNZ    COUNT,EDTWT2
*
        MOVP    %>00,IOCNT0         ; Got EDT_, disable abort interrupt
        MOVP    %>00,IOCNT1
        ORP     %BIT2,APORT         ; Squelch 532
        ANDP    %NOT4,APORT         ; ATE=0 (EXI MODE)
        MOVD    %CONN12,MSGL
        CALL    @PRINT              ; CONNECT 1200
        ANDP    %NOT0,BPORT         ; CPORT is active (CTS_=0)
*
        MOVD    %765,MSTIME         ; Wait 765 ms
        CALL    @MSDLY
        BR      @B212               ; 212A mode, act as 32010 to DTE interface
*
*     Call Initiation Routines.
*
* We are now in data mode.  Wait for a disconnect.
*
DAT103 ANDP     %NOT0,BPORT         ; Activate CTS_
*
* look for escape character
*
LP103A MOV      %3,TEMP1
LP103B EQU      $
*        BTJOP  %BIT1,APORT,NODTR0 ; no DTR_
        BTJOP   %BIT0,APORT,DIS103 ; no DCD_
LP103E BTJZP    %BIT1,SCTL0,LP103E ; received char?
        MOVP    RXBUF,A
        CMP     S5,A                ; escape character?
        JNE     LP103A
        DJNZ    TEMP1,LP103B
*
* we now have three escape characters. start escape code timer
*
        MOV     %50,COUNT1
LP103C MOVD     %20,COUNT
LP103D MOVD     %1,MSTIME
        CALL    @MSDLY
        BTJOP   %BIT1,APORT,NODTR0 ; no DTR_
        BTJOP   %BIT0,APORT,DIS103 ; no DCD_
        BTJOP   %BIT1,SCTL0,LP103A
        DJNZ    COUNT,LP103D
        DJNZ    COUNT1,LP103C
*
* everything checked out O.K.
*
        JMP     CM103
*
NODTR0 MOV      %5,COUNT            ; 5 m/s check of DTR_
NODTR1 MOVD     %1,MSTIME
        CALL    @MSDLY
        BTJZP   %BIT1,APORT,LP103B
        DJNZ    COUNT,NODTR1
*
* Disconnect  from 103 data mode
*
DIS103 ORP      %BIT2,APORT         ; Squelch 532
        ANDP    %NOT7,APORT         ; Go on hook
        MOVP    %>03,IOCNT0         ; Enable interrupt 1
        MOVD    %NOCAR,MSGL         ; Send disconnect message
        CALL    @PRINT
TCODE2 BTJZP    %BIT0,SSTAT,TCODE2
        EINT
        BR      @INIT
*
```

```
* 103 COMMAND MODE
*
CM103   ANDP    %NOT0,BPORT         ; Activate CTS_
        MOVD    %RESET,MSGL
        CALL    @PRINT
        BR      @LOOK               ; look for new command
*
*************************************************************
****  TIMOUT INTERRUPT OF CARRIER DETECT                  ****
*************************************************************
INT2    EQU     $
        DECD    CWT2                ; DECREMENT SECONDARY COUNTER
        JNC     CABORT              ; IF COUNTED OUT THEN APORT
        RETI                        ; TIMOUT NOT COMPLETE CONTINUE
CABORT  ANDP    %NOT7,APORT         ; GO ON HOOK
        ORP     %BIT2,APORT         ; SQUELCH 532
        ANDP    %NOT0,BPORT         ; ACTIVATE CTS
        ORP     %BIT3,IOCNT0        ; DISBLE TIMER
        EINT
        MOVD    %NOCAR,MSGL         ; SEND NO CARRIER
        CALL    @PRINT              ; MESSAGE TO DTE
        BR      @LOOK               ; LOOK FOR NEXT COMMAND
*
INT3    RETI
INT4    RETI
INT5    RETI
        AORG    >FFF4
VECT5   DATA    INT5
VECT4   DATA    INT4
VECT3   DATA    INT3
VECT2   DATA    INT2
VECT1   DATA    INT1
VECT0   DATA    INIT
```

# Implementation of an FSK Modem Using the TMS320C17

**Phil Evans**

**Regional Technology Center — Ottawa, Canada**
**Texas Instruments**

**Al Lovrich**

**Digital Signal Processor Products — Semiconductor Group**
**Texas Instruments**

# Introduction

This application report presents an implementation of a 300-bit-per-second (BPS) modem conforming to the V.21 and Bell 103 standards, using a TMS320C17 Digital Signal Processor (DSP).

The purpose of this application report is, with references [1], [2], [3], to provide a complete hardware design for a splitband modem and the software to implement a V.21/Bell 103 300-bps modem. The designer can then concentrate on developing value-added functions, such as V.22bis or V.22 standard modems, encryption algorithms, etc. These value-added functions are implemented in software and can be easily incorporated into the TMS320C17 software provided in Appendix B.

The structure of this report is as follows:

- The first section reviews basic modem concepts and definitions and introduces the reader to frequency shift keying (FSK) data modulation.
- The second section describes the major functional blocks of the FSK modem system presented in this report:

  — Host interface,
  — Modem controller,
  — Digital signal processor, and
  — Analog front end.
- References to documents describing the actual hardware implementation are provided.
- The third section discusses the DSP software implementation of the V.21/Bell 103 modulator/demodulator using the TMS320C17 DSP.
- The fourth section reviews some of the issues involved with incorporating additional code into DSP software provided in Appendix B.
- The fifth section concludes this report.
- Appendix A is a derivation of the filter coefficient value required for the sample fraction time delay.
- Appendix B is the source code listing for the TMS320C17 modulator and demodulator implementation.

## Background

Over the past decade there has been a proliferation in the number and the use of computer systems. Accompanying this growth, there has been an increased demand for data communications between the various computer systems and terminals.

One of the most convenient and frequently used methods of data communications between geographically separated computer equipment is via the Public Switched Telephone Network (PTSN). The essential element for this method of data communication is the modem.

The modem converts the digital data it receives from the computer system or terminal into a modulated analog signal that is transmitted via the telephone network to the destination computer system or terminal. At the destination, the receive modem demodulates the received signal and transfers the digital data to the receiving terminal or computer system.

Table 1 shows a number of popular modem standards as specified by either the International Telegraph and Telephone Consultive Committee (CCITT) or the Bell System.

**Table 1. Bell and CCITT Modem Standards**

| Modem | Standard | Type* | Modulation | Data Rate (BPS) | Duplex |
|---|---|---|---|---|---|
| Bell | 103 | S/B | FSK | 300 | Full |
| | 202 | S/B | FSK | 1200 | Half |
| | 212A | S/B | DPSK | 1200 | Full |
| | 201 | S/B | DPSK | 2400 | Half |
| CCITT | V.21 | S/B | FSK | 300 | Full |
| | V.22 | S/B | DPSK | 1200 | Full |
| | V.22bis | S/B | QAM | 2400 | Full |
| | V.32 | E/C | QAM | 9600 | Full |

* S/B = Split band E/C = Echo Cancelling

Modems can be either half-duplex or full-duplex. In a half-duplex system, the transmission can be in either direction; however, only one direction is possible at a time. A half-duplex modem cannot simultaneously transmit and receive information. At the end of its transmission sequence, the modem must advise the receiving modem that the sequence is complete. The receiving modem may then begin transmitting data.

In a full-duplex system, the data transmission is bidirectional. Both modems may simultaneously transmit and receive data. Bidirectional (simultaneous data transmission) is achieved by either splitband or echo cancellation techniques.

Figure 1 shows the spectral response of a typical telephone channel. A splitband modem uses a filtering scheme to separate the telephone channel into two distinct frequency bands. One band is dedicated to the transmissions of the originate modem, the other band is dedicated to transmissions of the answer modem. To separate the received signal from the received and transmitted signal that is detected on the two-wire telephone line, the modem removes the transmitted signal frequency band using a splitband filter [1], [4], or by other means (such as software implemented on the DSP). Dividing the telephone channel into two separate non-overlapping frequency bands limits the maximum baud rate.

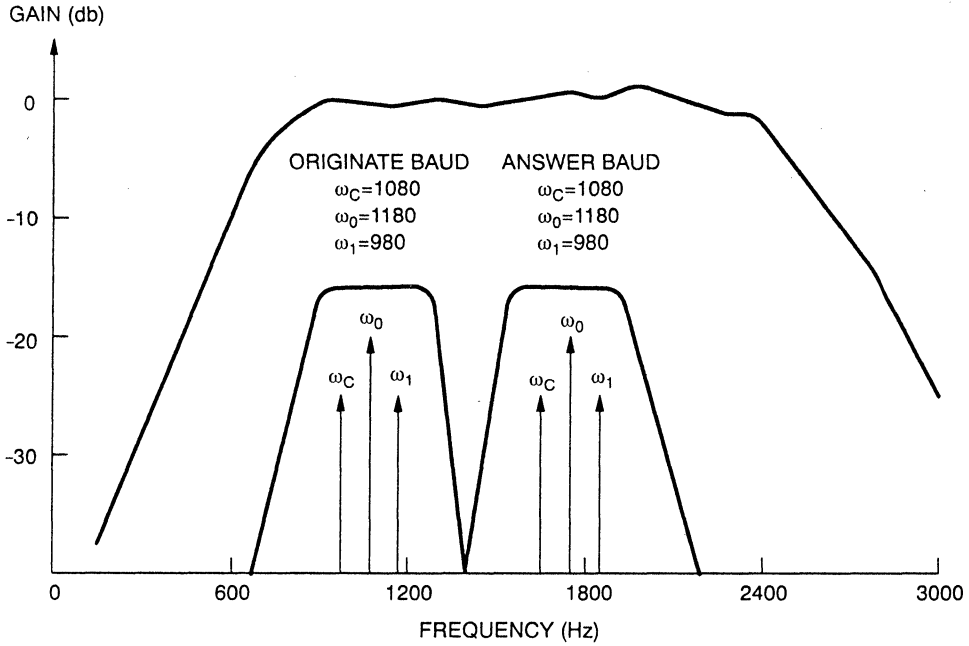



**Figure 1. Spectral Response of a Typical Telephone and a V.21 Splitband Modem**

The actual bit rate of the channel is determined by the baud rate and the data modulation scheme that is employed. Splitband type modems are typically used in low- to moderate-speed applications. As shown in Table 1, each modem standard uses a particular modulation scheme. For example, CCITT V.21, V.22, and V.22bis standards specify the frequency shift keyed (FSK), phase shift keyed (PSK) and quadrature amplitude modulation (QAM) schemes respectively.

Echo cancellation type modems, such as V.32, transmit both the originate and answer signals on the same channel. This allows both the originate and answer modems to utilize the complete bandwidth of the channel and to maximize the data baud rate. It is still necessary to separate the receive signal from the receive and transmit signal detected on the two-wire telephone line. However, the originate and answer signals are superimposed on the same channel band, and separating techniques that are more sophisticated than those found in splitband-type modems are required. The fact that transmit signal is typically 20 dB stronger than the receive signal, as measured on the transmit Tip and Ring, further complicates the extraction of the receive signal.

Echo cancellation type modems use algorithms that subtract an estimate of the transmit signal from the signal sampled from the two-wire telephone line, to determine the receive signal. Refer to [5] and [6] for further information on Echo cancellation type modems.

Table 2 shows the transmission frequencies for answer and originate modes for both the binary FSK modulated 300-bps V.21 and Bell 103 standards. It also shows details of the V.23 and Bell 202 1200-bps half-duplex standards.

**Table 2. Binary FSK Transmission Frequencies**

| Modem Standard | | Carrier (Hz) | 1(Mark) (Hz) | 0(Space) (Hz) |
|---|---|---|---|---|
| V.21 | Originate | 1080 | 980 | 1180 |
| | Answer | 1750 | 1650 | 1850 |
| BELL 103 | Originate | 1170 | 1270 | 1070 |
| | Answer | 2125 | 2225 | 2025 |
| V.23 | | 1700 | 1300 | 2100 |
| BELL 202 | | 1700 | 1200 | 2200 |

Since this report is primarily concerned with the 300-bps V.21 and Bell 103 standard modems, it is worthwhile to review FSK data communication.

These are the primary advantages of an FSK system:

1. There is no requirement for carrier phase recovery; this reducing system complexity.
2. Increased immunity to amplitude nonlinearities. FSK is a constant envelope signal, with the information transmitted in the zero crossings. It is less affected by amplitude nonlinearities than amplitude modulated schemes, and
3. The modulator and demodulator architectures are easily implemented in software.

The primary disadvantage of FSK modulation is its low spectral efficiency. Because the telephone network is bandlimited to 4KHz, only moderate data transmission rates over the telephone network are supported by an FSK modulation scheme. As a consequence, FSK is often the favored modulation scheme for very low cost, low-to-moderate speed data communication systems.

Subsequent sections of this report discuss FSK modulation and demodulation in some detail. It is important that you understand the mathematical representations of FSK signals. FSK modulation is represented in the following manner:

$$S(t) = \cos((\omega_c \pm \delta\omega)*t + \phi) \quad (1)$$

where $S(t)$ = Transmitted signal
$\omega_c$ = Carrier frequency
$\delta\omega$ = Frequency shift
$t$ = Time
$\phi$ = Phase shift

For a given baud period T, S(t) is at a frequency $f_1=(f_c+\delta f)$ or $f_0=(f_c-\delta f)$, corresponding to the transmission of a 1 or 0, respectively, for the duration of the baud period. In some cases, it is convenient to represent

$$\omega_0 = \omega_c - \delta_\omega \quad (2)$$
$$\omega_1 = \omega_c - \delta_\omega$$

Thus the following identities are true:

$$\omega_c = (\omega_1 + \omega_0)/2 \quad (3)$$
$$\delta\omega = (\omega_1 - \omega_0)/2$$

Some binary FSK modulation schemes, such as V.21, have $\omega_0$ greater than $\omega_1$; so by (3), $\delta\omega$ would be negative. Figure 2 shows an FSK signal transmission.

Note that the telephone channel provides limited spectral bandwidth. To achieve progressively higher data rates, more spectrally efficient modulation schemes, such as PSK and QAM, must be used. As spectral efficiency increases, typically, the complexity of the signal modulation and demodulation schemes increase. Additional information on modulation schemes can be found in references [4], [5], [6] and [7].

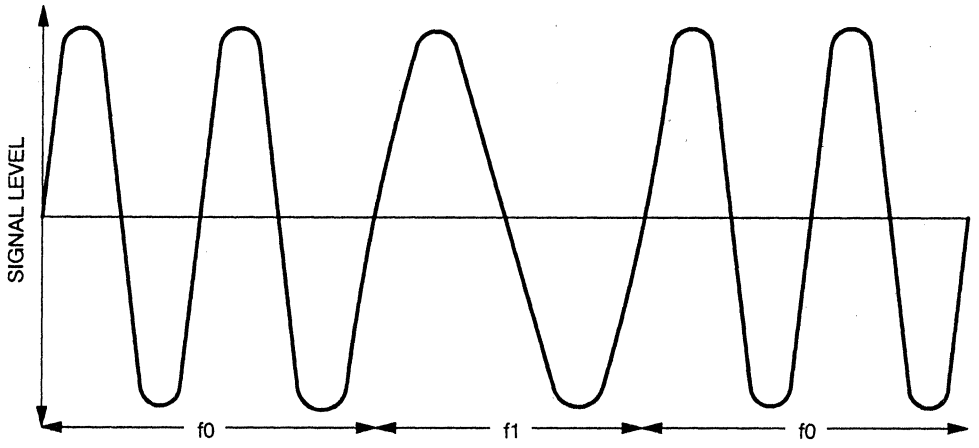



**Figure 2. FSK Signal Transmission**

## System Description

As discussed in the introduction, this application report presents the implementation of a V.21/Bell 103 300-bps FSK modem using a TMS320C17 Digital Signal Processor. The system hardware is identical to that of the Texas Instruments DSP2400 modem [1].

There are significant functional differences between the modem design provided here and the DSP2400 modem. These result from the differences between the TMS320 code provided in Appendix B and the DSP2400 code. The software found in Appendix B implements a V.21/Bell 103 FSK modem. The DSP2400 also implements V.22, Bell 212A, and V.22bis standard modems that implement PSK and QAM modulation/demodulation and the associated carrier recovery, clock recovery, and adaptive equalization functions.

The software in Appendix B provides all the necessary hooks so that the designer can easily incorporate his own custom value-added features (such as V.22 and V.22bis standard modems). Nevertheless, the reader should be aware of the difference between the DSP2400 software implementation and the software in Appendix B, particularly when referring to any DSP2400 related literature [1], [2], [3].

Figure 3 is a block diagram showing the components of the modem system. The modem consists of the following subsystems:

1. Host interface
2. Modem controller
3. Digital signal processor
4. Analog front end

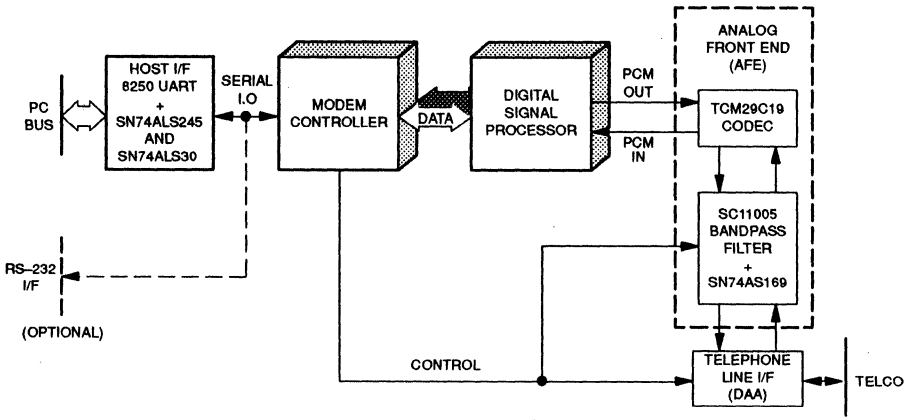



**Figure 3. Block Diagram of Modem System Components**

The designer must provide an interface between the host data terminal equipment and the modem controller. The DSP2400 uses an 8250 UART (plus a 74LS245 buffer and a 74ALS30 NAND Gate) to interface between a standard PC-AT and the modem controller. A standard RS-232C interface is used between the UART and the modem controller. The circuit diagram and additional information on the host interface used for the DSP2400 Modem can be found in [1].

The modem controller (80C51, TMS70C42, etc.) handles the overall modem control [3], directs the handshaking sequences, etc. It specifically performs the following functions:

1. AT command set interpretation
2. Scrambling/descrambling
3. Pulse dialing
4. Synchronous/asynchronous conversion
5. Modem configuration control
6. Protocol initialization

The modem controller sends a command to the DSP once per baud. Table 3 is a complete list of the commands, showing the structure and functions that are implemented.

## Table 3. Modem Controller Commands for the DSP

| Command | Code | Description |
|---|---|---|
| Protocol Select | Fxh | Select protocol<br>Bits 1, 0 – Speed select<br>0 0 = 300 BPS<br>0 1 = Reserved<br>1 0 = Reserved<br>1 1 = Reserved<br>Bit 2 – CCITT/Bell<br>0 = CCITT<br>1 = Bell<br>Bit 3 – Answer/originate<br>0 = Answer<br>1 = Originate |
| Reserved | Exh | Reserved command |
| Operation Select | Dxh | Select operating mode<br>(bits 3, 2 reserved)<br>0 0 = Line mode<br>0 1 = Analog loopback<br>1 0 = Reserved<br>1 1 = Reserved |
| Reserved | Cxh | Reserved command |
| Reserved | Bxh | Reserved command |
| Reserved | Axh | Reserved command |
| Transmit DTMF Tones | 9xh | Dial DTMF and return to configuration mode<br>xxxx = D3-D0; numbers 0-9, A, B, C, D, *, and # |

## Table 3. Modem Controller Commands for the DSP (Concluded)

| Command | Code | Description |
|---|---|---|
| Transmit Mode Select | 8xh | Enable answer tone/data select<br>Bits 1, 0 = Transmit select<br>0 0 = Transmit idle<br>0 1 = Transmit answer tone<br>1 0 = Transmit data mode enable<br>1 1 = Reserved<br>Bits 3, 2 = Select answer tone frequency<br>0 0 = 2100 Hz answer tone (V.21)<br>0 1 = 2225 Hz answer mark (Bell 103)<br>1 0 = 2025 Hz answer space (Bell 103)<br>1 1 = Reserved |
| Receive Mode Select | 7xh | Select receive configuration<br>(bits 3,2 reserved)<br>0 0 = Receive idle mode<br>0 1 = Reserved<br>1 0 = Receive data mode<br>1 1 = Reserved |
| Reserved | 6xh | Reserved command |
| FSK Mode | 5xh | Select 300 BPS mode<br>(bits 3,2,1 reserved)<br>0 = 300 BPS mode deselect<br>1 = 300 BPS mode select |
| Reserved | 4xh | Reserved command |
| Reserved | 3xh | Reserved command |
| Reserved | 2xh | Reserved command |
| Reserved | 1xh | Reserved command |
| Reserved | 0xh | Reserved command |

As an example, the DSP2400 uses a masked ROM version of the TMS70C42 microcontroller (denoted as a TMS70C2400A) as the modem controller. The TMS70C2400A source code is available from Texas Instruments and includes provisions for the V.22bis and V.22 standard modems.

One noteworthy advantage of the TMS70C42/TMS320C17 interface is that it requires no external glue logic [7]. For complete information on the TMS70C2400 Modem Controller, including the call originate and answer sequences, refer to [2].

The TMS320 Digital Signal Processor performs the computationally intensive tasks such as modulation, demodulation, and tone generation and detection. It does not perform any control functions. Specifically, the TMS320 DSP performs the following functions:

1. Modulation/demodulation (V.21/Bell 103)
2. Data encoding/decoding
3. Filtering
4. Automatic gain control
5. Tone dialing
6. Call progress monitoring

The DSP is discussed in further detail in the next section of this application report. The DSP source code in Appendix B was originally part of the code developed for the TMS320A2400 Modem Digital Signal Processor (a ROM coded TMS320C17 DSP). The TMS320A2400 source code also includes V.22bis, V.22, and Bell 212A standard modems, with the software implementing the QAM and PSK modulation and demodulation schemes, carrier recovery, clock recovery, automatic gain control, and adaptive equalization functions. The TMS320A2400 and the source code is available from Texas Instruments.

Despite the differences between the code provided in Appendix B and the TMS320A2400 code, [1] and [3] are useful references, providing technical information about TMS320C17 modem applications.

The analog front end is composed of a TCM29C19 combo codec [9], a SC11005 bandpass filter [10] and a data access arrangement (DAA) telephone line interface composed of discrete components. The codec converts an 8-bit $\mu$-law companded bit stream to an analog waveform and vice versa, at a 9.6-KHz sampling frequency. The SC11005 is a splitband filter that separates the transmit and receive carriers and performs the required signal shaping to the analog waveform. The DAA section is composed of a number of discrete components and is required to interface the modem to the public telephone network as dictated by FCC Rules Part 68. The analog front end circuit diagram is found in [1]. Further technical details are found in [2].

## The DSP Software Implementation

The code provided in Appendix B is written specifically for a Texas Instruments TMS320C17 Digital Signal Processor. The key architectural features of the TMS320C17 are these:

1. 4 Kwords (8 Kbytes) of on-chip maskable ROM
2. 256 words of on-chip data RAM
3. Two full-duplex serial ports
4. On-chip companding hardware ($\mu$- or A-Law)
5. On-chip sign magnitude/two's complement conversion hardware
6. A coprocessor port
7. 6.25-MIPS maximum execution speed

TMS320E17, with 4 Kwords of on-chip EPROM substituted for the 4 Kwords of maskable ROM, is also available for development and prototyping purposes. Refer to [8] and [11] for additional information on the TMS320C17 and TMS320E17.

The TMS320C17 source code listing file is found in Appendix B. The code requires approximately 50 words of data RAM and occupies 1100 words of program ROM. Of the 1100 words of program memory, 390 are coefficients, and the remaining 710 words are the program instructions. The software consists of a main program that references various subroutines. These are the main subroutines found in the program:

1. Command control interpreter (CCI)
2. FSK transmitter (FSKTX)
3. Dual-tone multifrequency transmitter (Part of FSKTX)
4. Automatic gain control (AGC)
5. FSK receiver (RSTSK)

The next section of text describes the main program. The subroutines are discussed in subsequent sections.

Figure 4 is a block diagram of the main program (code starting at beginning of main program label and ending at start of subroutines label) in Appendix B. Once the initialization of the data RAM and control registers (code beginning at start of additional tables label and ending at start of main program sequencer label) is complete, the main program loop is executed. The device remains in the WAIT loop (first four lines of code of main program sequencer routine) until the FR flag in the control register is raised. Control register bits 27-24 and 23-16 are set so the main program and data samples are transmitted/received to/from the TCM2919 codec at a rate of 9.6 KHz.

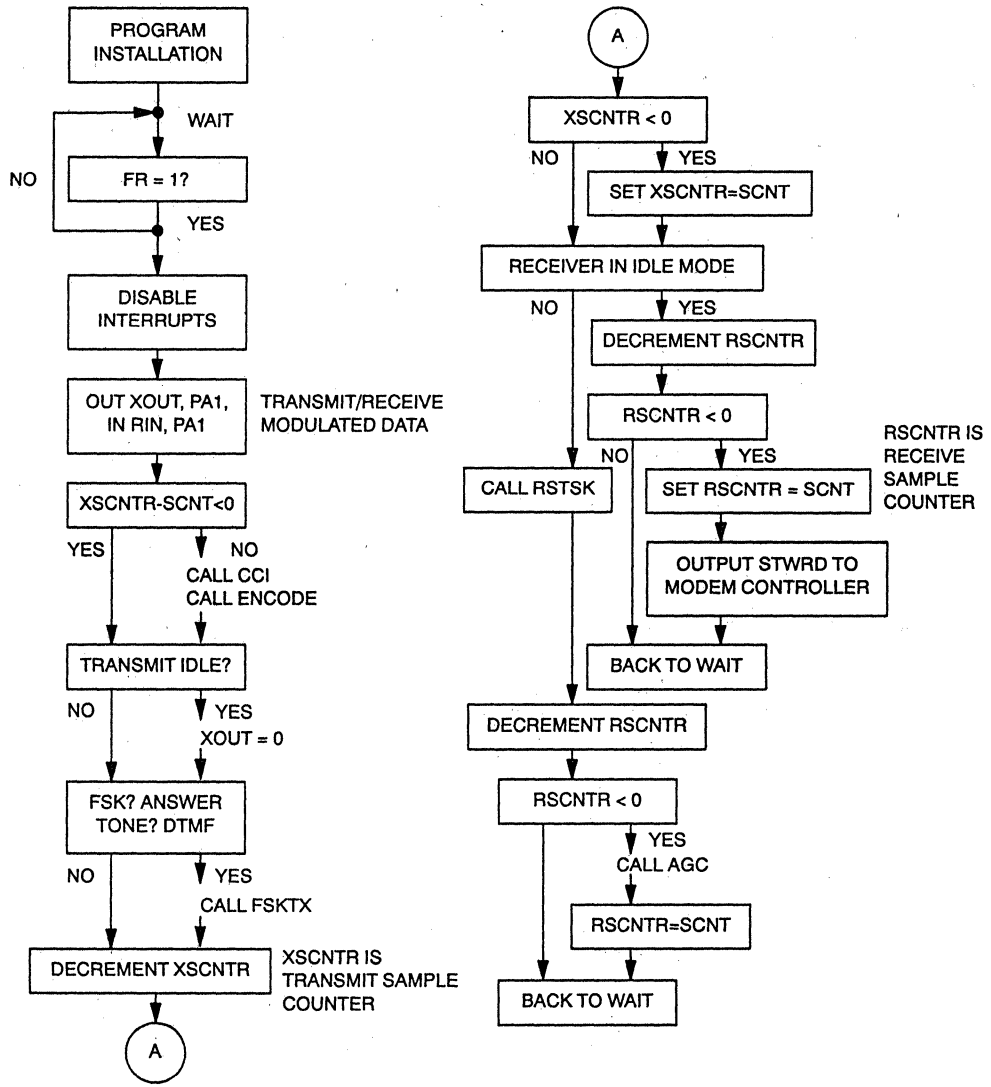



**Figure 4. Flowchart of Main Program (Appendix B)**

As the V.21/Bell 103 standard modems transmit data at 300 bps, a 9.6-KHz sampling rate results in 32 samples/baud. The 9.6-KHz sampling rate is very practical for several reasons:

- It is higher than the Nyquist sampling frequency of approximately 8 KHz for a telephone channel, and
- It is a convenient multiple of the popular modem transmission frequencies (300, 1200, and 2400 bps).

The TMS320C17 is clocked by a 18.432-MHz oscillator. To satisfy the 9.6-KHz sampling frequency, the number of instructions executed per sample must be less than 480. To implement the various functions required by the FSK modulator/demodulator, it is necessary to distribute the tasks among the various samples within the baud. The command control interpreter (CCI) is executed during the first sample of the baud, and the AGC routine is implemented during the final sample baud.

When the raised FR flag is detected, the processor exits the WAIT loop and executes the main program. Refer to [8], Sections 3.8 and 3.9 for additional details on the FR flag, interrupts, and serial port. Table 4 describes the variables that are referenced in the main program.

**Table 4. Variables Referenced in Main Program Variable**

| Variable Name | Description |
|---|---|
| XSCNTR | The transmit counter; equals the number of samples that have been transmitted in the current baud. |
| SCNT | Number of samples in a baud, i.e., 9.6 KHZ/300 HZ = 32 samples/baud. |
| XOUT | Output sample sent via the TX serial port to the combo codec. |
| RIN | Input sample sent via RX serial port from the combo codec. |
| STATUS | An 8-bit number used internally by the DSP. Indicates present operating mode of the modem. |
| STWRD | 8-bit status word sent to the modem controller by the DSP. See Table 5. |
| OAFLAG | Indicates if modem is in originate or answer mode. OAFLAG = 0 → originate mode. |
| DTFLAG | Indicates if the modem is transmitting DTMF tones. DTFLAG = 1 → transmitting DTMF data. |

Table 5 shows the organization of STWRD (the DSP status word that is written to the microcontroller).

**Table 5 STWRD - DSP Status Word Written to the Modem Controller**

| Bit No. | Description |
|---|---|
| 7 | Enable/disable automatic gain control.<br>0 = Enable<br>1 = Disable |
| 6 | EDT (in band energy)<br>0 = Not detected<br>1 = Detected |
| 5 | Reserved |
| 4 | Reserved |
| 3 | Received data bit (0,1) |
| 2 | Reserved, set to 1 |
| 1 | Reserved, set to 1 |
| 0 | Reserved, set to 1 |

When the program exits the wait loop, it disables all interrupts and reads a data sample RIN from the receive buffer or writes a data sample XOUT to the transmit buffer of serial port #1.

At the first sample of a baud, when XSCNTR = SCNT (=31), the program implements the command control interpreter (CCI) subroutine as shown in the following code. Note that SCNT = 31, and XSCNTR is initially set at 31 and decremented by 1 every sample. When XSCNTR equals 0, it is reset to 31, for a total of 32 samples.

```
        LAC   XSCNTR
        SUB   SCNT      ; ACCUM = XSCNTR-SCNT
        BLZ   SEQU      ; BRANCH TO SEQU IF ACCUM <0
        CALL  CCI
SEQU:   LACK  030h
```

The CCI subroutine reads the next 8-bit command from the modem controller (TMS70C42400A or equivalent), performs the required program control functions, and returns to the main program.

If the DSP is in transmit idle mode, the data sample XOUT is set to 0 and sent to serial port #1 transmit buffer.

If the DSP is not in transmit idle, the FSK transmit subroutine FSKTX is called. Depending on the present value of STATUS as determined by the modem controller and the CCI subroutine, the FSKTX subroutine will transmit FSK encoded data, DTMF tones, or an answer tone. Upon completing the FSKTX subroutine, the program decrements the transmit sample counter XSCNTR by 1 and checks to see if it is less than 0. If so, XSCNTR is reset to 31. Otherwise, the program proceeds without any further modifications to XSCNTR.

At this point, the main program checks to see if the receiver is in idle mode. If the receiver is in idle mode, the receive sample counter RSCNTR is decremented. If RSCNTR is not less than 0, the program returns to the WAIT loop. If RSCNTR is now less than 0, it is reset to 31, and the program then returns to the WAIT loop.

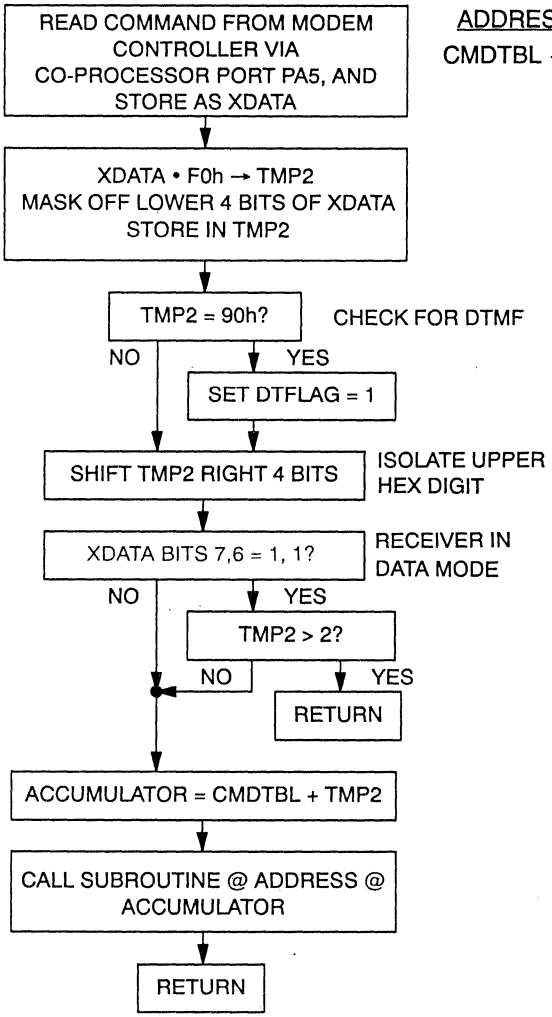
If the receiver is not in idle mode, the receiver decode/demodulation subroutine RSTSK (receiver per sample task) is called. This subroutine demodulates the receiver signal and estimates the value of the received data. When the subroutine is complete, the main program decrements RSCNTR and resets it to 31, if required.

After the RSTSK subroutine is complete, the program decrements RSCNTR. If RSCNTR is greater or equal to 0, the program returns to the wait loop. For the sample, when RSCNTR is less than 0, the automatic gain control subroutine (AGC) is called once per baud. The AGC subroutine monitors and compensates for any significant variation of the received signal level caused by telephone line fluctuations and other dynamic effects. RSCNTR is then RESET to 31, and the program returns to the WAIT loop.

The main program calls the following subroutines:

- CCI—Command control interpreter
- DTMF—DTMF setup
- FSKSET—Set up FSK transmit frequency
- FSKTX—Transmitter mode select
- OPER—Set operating mode
- PROTO—Protocol select
- RESET—Reset and equalizer enable
- RMODE—Receiver mode select
- RSTSK—FSK demodulation
- XMODE—Transmitter mode select

Figure 5 shows a block diagram of the CCI subroutine. The CCI reads the setup command from the modem controller (through the co-processor port PA5) and stores it in data RAM location XDATA (The structure of XDATA is shown in Table 3). The CCI then calls the appropriate subroutine to modify the system control bits (OAFLAG and DTFLAG) and status register (STATUS). The CCI, depending whether the modem configuring the DSP is in answer, originate, or transmit DTMF, loads the required nominal frequency values into TXFRQ and RXFRQ. Table 6 shows the organization of the STATUS register.




| ADDRESS | | SUBROUTINE | DESCRIPTION |
|---|---|---|---|
| CMDTBL + | 0h | RESERVED | |
| | 1h | RESERVED | |
| | 2h | RESERVED | |
| | 3h | RESET | |
| | 4h | RESERVED | |
| | 5h | FSKSET | FSK MODE |
| | 6h | RESERVED | |
| | 7h | RMODE | RECEIVED MODE SELECT |
| | 8h | XMODE | TRANSMIT MODE SELECT |
| | 9h | DTMF | DTMF TONES |
| | Ah | RESERVED | |
| | Bh | RESERVED | |
| | Ch | RESERVED | |
| | Dh | OPER | OPERATION SELECT |
| | Eh | RESERVED | |
| | Fh | PROTO | PROTOCOL SELECT |

**Figure 5. Flowchart of the CCI Subroutine**

**Table 6. The Status Register Organization**

| Bits | Description |
|---|---|
| 7,6 | Indicate Receiver Mode:<br>00 = Receiver in Idle Mode<br>01 = Call Progress Monitoring Mode<br>10 = Data Mode<br>11 = Reserved |
| 5,4 | Indicate Transmitter Mode:<br>00 = Transmitter in Idle Mode<br>01 = Transmit Answer Tone<br>10 = Data Mode<br>11 = Reserved |
| 3 | Answer/Originate Mode:<br>0 = Originate Mode<br>1 = Answer mode |
| 2 | CCITT/Bell Mode:<br>0 = CCITT (V.21)<br>1 = Bell (103) |
| 1,0 | Speed status:<br>00 = 300 BPS<br>01 = Reserved<br>10 = Reserved<br>11 = Reserved |

The setup commands from the modem controller and subroutines called by the CCI subroutine are shown in Table 3.

The RESET subroutine loads 81h into the STWRD word that is sent to the modem controller via the co-processor port PA5. This advises the modem controller that the DSP has been reset. The DSP program then branches to START, and the DSP is reinitialized.

The FSKSET subroutine reads the XDATA word to determine if the next bit to be transmitted is 0 or 1 and then loads the appropriate 0 or 1 frequency F0ADD or F1ADD into the TXFRQ register.

When setup in answer mode, XDATA bits 3 and 2 are loaded into the STATUS register bits 7 and 6, respectively, by the RMODE subroutine. These bits determine what tasks the FSK receiver subroutine RSTSK will perform, as shown in Table 3 and Figure 5.

The XMODE subroutine reads XDATA bits 0 and 1. These bits determine what tasks the FSK transmitter subroutine FSKTX will perform as shown in Figure 4. If the transmit answer tone function is selected, bits 2 and 3 of XDATA indicate what the answer tone frequency will be:

| | | |
|---|---|---|
| XDATA Bits 3,2 = | 0,0 | 2100 Hz |
| | 0,1 | 2225 Hz |
| | 1,0 | Reserved |
| | 1,1 | Reserved |

The program loads the appropriate answer tone value into register TXFRQ. XMODE then loads XDATA bits 1 and 0 into STATUS bits 5 and 4, respectively. STATUS bits 5 and 4 determine what tasks the transmitter subroutine FSKTX will perform.

The DTMF subroutine determines what number or symbol needs to be transmitted by reading XDATA bits 3 through 0. DTMF then loads the appropriate high-frequency phase step, low-frequency phase step, high-frequency gain, and low-frequency gain into the RXFRQ, TXFRQ, DTMFH, and DTMFL registers, respectively, from the Table TONTBL.

The OPER subroutine checks bits 1 and 0 of XDATA. If bits 1 and 0 equal 0 and 1 bit 3 of STATUS is set to 1, indicating that the modem is in analog loopback mode. If bits 1 and 0 are not equal to 0 and 1, OPER returns without performing any operations.

The PROTO subroutine selects the mode and protocol of the DSP based on XDATA bits 3 through 0. PROTO first sets bits 1 and 0 of STATUS (indicating the modem data rate), based on the value of bits 1 and 0 of XDATA (see Figure 7).

While the software provided in Appendix B supports only a 300-bps data rate, it does provide the necessary hooks so that different standard modems (ie V.22, V.22bis) can easily be incorporated into the code.

Next, PROTO checks XDATA bits 3 and 2 to determine if the modem should be in originate/answer mode and Bell/CCITT mode.

| | | |
|---|---|---|
| Bit 3: | 0 = | Originate |
| | 1 = | Answer |
| Bit 2: | 0 = | Bell |
| | 1 = | CCITT |

As shown in Table 2, the transmission frequencies of the Bell 103 and V.21 originate and answer modes are unique. PROTO loads registers used by the FSK transmitter

subroutine (FSKTX) and the FSK receiver subroutine (RSTSK) with values stored in table TONTBL in data ROM and corresponding to transmit and receive frequencies.

PROTO then uses the XDATA bits 3 and 2 to determine which constants are transfered from table FSKTBL into addresses F1ADD (transmit 1 phase step), F0ADD (transmit 0 phase step), B1FSK (FSK delay filter coefficient), and GAIN (FSK mode gain). PROTO also loads addresses SCNT (baud counter=32), TRANS (FSK data transmition N=15), A1FSK (A1 demodulator filter coefficient), A2FSK (A2 demodulator filter coefficient), and DZONE (dead zone of window comparator) with the appropriate values.

If bit 3 of the STATUS word equals 1, the modem is set to analog loopback mode, and the modem should receive the information that it transmits. PROTO checks to see if bit 3 of STATUS equals 1; if so, the receiver parameters are modified to be the same band as the transmitter.

The FSK modulator is implemented in the FSKTX subroutine. Figure 6 is a block diagram of the FSKTX subroutine. The primary function of the FSK modulator is the following: Given a stream of binary data $a_0$, $a_1$, $a_2$, . . ., $a_{k-1}$, ak for each data element $a_k = \{0,1\}$, generate a corresponding signal of frequency $f_0$ or $f_1$ for the duration of $a_k$'s baud period.

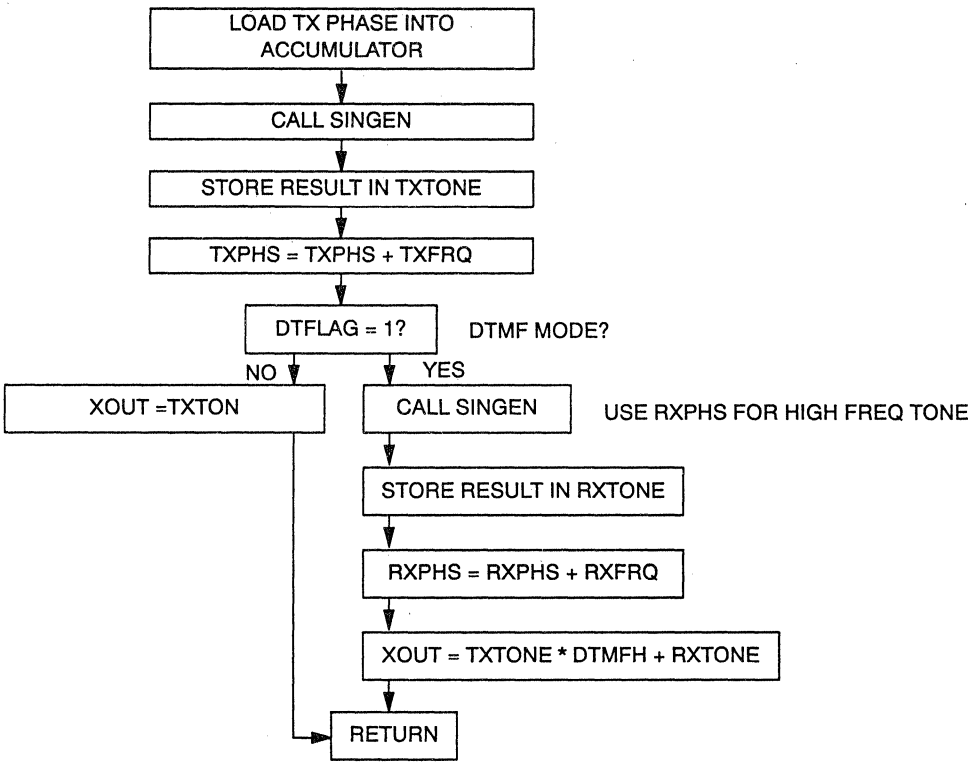



**Figure 6. Flowchart of Subroutine FSKTX**

Figure 7 shows a functional model of the FSK modulator. The TMS320 software implementation of the FSK modulator generates tones by stepping through a cosine table. The size of the phase step determines the output signal frequency. You should pay particular attention how phase angles, phase steps, cosines, and sines are represented as 16- and 32-bit integer numbers.

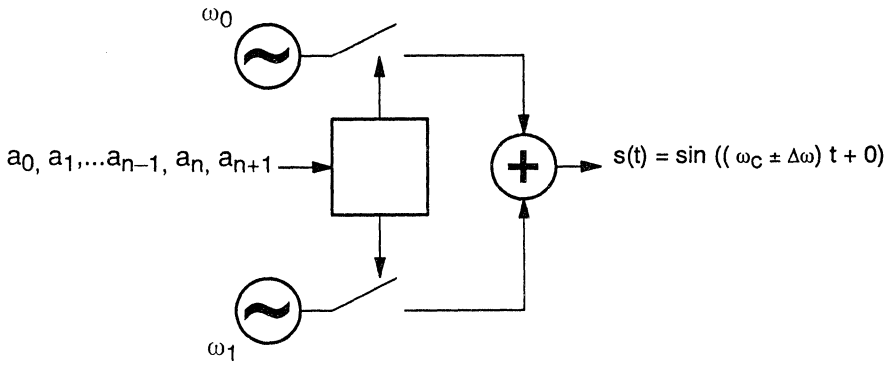



**Figure 7. Functional Model of an FSK Modulator**

Table 7 describes the significant variables used in the FSKTX subroutine.

**Table 7. Variables Referenced in the FSK Transmitter Subroutine FSKTX**

| Variable Name | Description |
|---|---|
| TXPHS | Present value of the transmit signal phase. Also used as present phase of the low frequency DTMF tones. |
| TXFRQ | Phase step between consecutive TXPHS samples. |
| RXPHS | Normally used in the FSK demodulator subroutine RXTSK. Used as present phase for the high frequency DTMF tone. |
| RXFRQ | Normally used in RXTSK subroutine. Also used as phase step for high-frequency tone when transmitting DTMF tones. |
| DTMFL | Scaling factor for low-frequency DTMF tones. |
| DTMFH | Scaling factor for high-frequency DTMF tones. |
| SINGEN | A subroutine called by FSKTX. Given a 16-bit number representing an angle from 0 to Pi, the SINGEN routine determines the sine of the angle and stores the result at address TMP3. |

The software FSK Modulation routine receives data at a rate of 300 bps and generates 12-bit, two's complement data samples at a rate of 9.6 KHz. The TMS320C17's on-chip hardware compander reduces the sample to 8 bits before it is sent to the Codec via the serial port.

The most recent phase of the output signal is stored in data memory location TXPHS, and the amplitude is read from the COSOFF table by the SINGEN subroutine. The frequency of the transmitted signal is determined by the size of the phase step TXFRQ between successive output samples:

$$TXPHS[(N+1)T] = TXPHS[NT] + TXFRQ[NT]$$

The value of TXFRQ is determined by the FSKSET subroutine referenced by the CCI subroutine. Recall that, depending on the instruction received from the modem controller at the beginning of the baud, the CCI subroutine loaded data memory location TXPHS with either F0ADD or F1ADD. Table 8 shows the FSK frequencies and phase steps (TXFRQ) for the V.21 and Bell 103 modem standards.

**Table 8. Frequencies and Phase steps for V.21 and Bell 103 Modems**

| Modem Standard | | Frequency (Hz) | Phase Step @9.6 KHz | Phase Step TXFRQ, Q15 hex |
|---|---|---|---|---|
| V.21 | Originate 1 | 980 | 0.2042*Pi | 1A22h |
| | Originate 0 | 1180 | 0.2458*Pi | 1F77h |
| | Answer 1 | 1650 | 0.3428*Pi | 2C00h |
| | Answer 0 | 1850 | 0.3854*Pi | 3155h |
| Bell 103 | Originate 1 | 1270 | 0.2646*Pi | 21DDh |
| | Originate 0 | 1070 | 0.2229*Pi | 1C89h |
| | Answer 1 | 2225 | 0.4635*Pi | 3B55h |
| | Answer 0 | 2025 | 0.4219*Pi | 3600h |

The magnitude of the phase step is determined by

$$[(\text{Desired Frequency})/(\text{Sampling Frequency})] * 2\pi$$

In the case of the originate 1 of the V.21 modem, the phase step equals

$$(1270/9600) * 2\pi = .2646\ \pi \text{ Radians}$$

Both TXPHS and TXFRQ data memory locations are 16-bit binary numbers in Q15 two's complement notation equal to

$$(\text{Output Signal Phase})/\pi.$$

Thus TXPHS hex values

$$\begin{aligned} 2000h &= \pi/4 \\ 4000h &= \pi/2 \\ 6000h &= 3\pi/4 \\ 8000h &= -\pi \\ A000h &= -3\pi/4 \end{aligned}$$

An advantage of this approach is that the phase of the output signal is continuous. This provides a higher spectral efficiency than that of a discontinuous phase FSK implementation.

The sine generation subroutine SINGEN subtracts $\pi/2$ (4000h) from TXPHS and uses this phase to read the amplitude from the COSOFF table. The symmetry of the cosine function has been used to reduce the table size from 513 to 257 elements, with data memory addresses COSOFF, COSOFF+128, and COSOFF+256 corresponding to 0, $\pi/2$, and $\pi$ radians, repectively. To determine the cosine of an angle outside the 0-to-$\pi$ range, the program utilizes the two's complement format of the data and the absolute value function ABS. As an example, assume that the present phase TXPHS is

$$\text{TXPHS(N)} = (-170/256) * \pi = -.6640625 * \pi = \text{A600h}$$

If we are transmitting a 1 in V.21 Originate mode, the phase step is

$$\text{TXFRQ} = .26448 * \pi = \text{21DDh}$$

The next value of:

$$\begin{aligned} \text{TXPHS(N+1)} &= \text{TXPHS(N)} + \text{TXFRQ} \\ &= -.6640625\ \pi + .26448\ \pi \\ &= -.3995825\ \pi \\ &= \text{A600h} + \text{21DDh} = \text{C7DDh} \end{aligned}$$

The subroutine then subtracts $\pi/2$ (4000h) from TXPHS, so the sine of angle TXPHS can be determined from the Cosine table:

$$\begin{aligned} \text{ANGLE} &= \text{TXPHS(N+1)} - \pi/2 \\ &= -.3995825\ \pi - .5\ \pi \\ &= -.8995825\ \pi \\ &= \text{C7DDh} - \text{4000h} = \text{87DDh} \end{aligned}$$

Note that TXRFQ is added to TXPHS(N), and $\pi/2$ is subtracted from TXPHS(N+1) with the sign extension suppressed, so TXPHS(N+1) = 87DDh. This represents 1.06143 $\pi$ as an unsigned number or $-.93857\ \pi$ as a signed number. If we now consider TXPHS(N+1) a signed and take the absolute value:

$$\text{ABS[TXPHS]} = \text{ABS[87DDh]} = \text{7823h representing } .93857\ \pi$$

Note that:

$$\text{Cos}(1.06143\pi) = \text{Cos}(.93857\pi) = -.98144$$

The cosine table address is generated:

COSSOFF + (7823h/80h) = COSOFF + F0h

The value at Data Memory address COSOFF + F0h is

$$\text{Cos}((240/256)\pi) = -.980786 = 8276\text{h, Q15 2's complement notation}$$

Within the limits of the cosine table precision, the calculated output value equals the value read from the table.

The structure of the FSK Demodulator is shown in Figure 8.

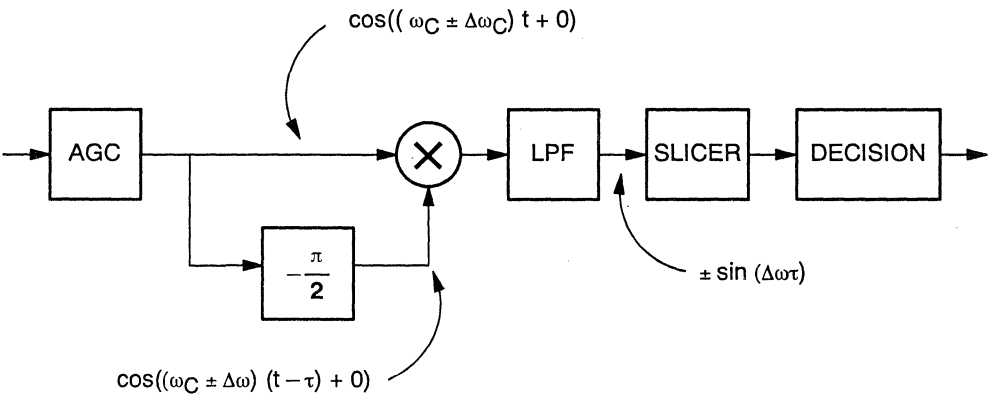



**Figure 8. FSK Demodulator**

The received FSK signal is sent to the DSP from the Codec via the serial port. The on-chip companding hardware expands the signal from an 8- to 13-bit value. The automatic gain control routine compensates for transient signal level variations and sends the amplitude adjusted received signal R(t) to the software demodulator.

$$R(t) = \cos[(\omega_c \pm \delta\omega) * t + \phi] \quad (4)$$

As this is a binary FSK system, the frequency of this signal is either $\omega_c - \delta\omega$ or $\omega_c + \delta\omega$, depending on whether a 0 or 1 was sent. (Recall from the V.21 signal that $\delta\omega$ is less than 0.)

The received signal R(t) is multiplied by a delayed version of itself:

$$R(t - \tau) = \cos[(\omega_c \pm \delta\omega) * (t - \tau) + \phi] \quad (5)$$

Where $\tau$ is the signal delay.

The product of the received signal (4) and delayed received signal (5) is

$$2 * \cos[(\omega_c \pm \delta\omega) * t + \phi] * \cos[(\omega_c \pm \delta\omega) * (t - \tau)] \quad (6)$$

$$= \cos[2(\omega_c \pm \delta\omega) * t - (\omega_c \pm \delta\omega) * \tau + 2 * \phi] + \cos[(\omega_c \pm \delta\omega) * \tau] \quad (7)$$

If $\omega_c \tau$ is set to equal $\pi/2$, and (7) is lowpass filtered to remove the double frequency component, the resulting signal is

$$\cos(\pi/2 \pm \delta\omega * \tau) = \sin(\pm\delta\omega\tau) = \pm \sin(\delta\omega) \quad (8)$$

If $\delta\omega$ is greater than 0, then the sign of the lowpass filter output will be positive or negative, depending on whether $\omega_c + \delta\omega$ or $\omega_c - \delta\omega$ is originally transmitted. If $\delta\omega$ is less than 0, obviously the opposite relationship is true. The sign of the lowpass filter output indicates the value of the received data.

The TMS320 software implementation of the 300-bps FSK Demodulator is found in Subroutine RSTSK, Subroutine CCITT, and Subroutine FDEM20 in Appendix B.

The AGC subroutine provides the RSTSK subroutine with a Q11 two's complement format received signal sample at a rate of 9.6 K samples per second.

As previously discussed, the data is extracted from the received signal by multiplying the received signal by a $\pi/2$ delayed version of itself, $\cos[(\omega_c \pm \delta\omega) * t + \phi - \pi/2 \pm \delta\omega * \tau]$. The product is then passed through a lowpass filter to remove the high frequency components.

If the desired phase delay is

$$\omega_c * \tau = \pi/2, \quad (9)$$

then

$$\tau = 1/(4 * f_c) \quad (10)$$

The sample rate is 9.6 KHz, or a period T = 104.167 $\mu$s. Table 9 shows the carrier frequencies, for both the V.21 and Bell 103 standards, the time delays corresponding to a $\pi/2$ phase delay and the equivalent number of 9.6-Khz samples. Note that none of the delays are exact multiples of the 9.6-KHz sampling period; each delay has an integer and fractional part.

**Table 9. Carrier Frequency and Time Delays**

| Modem Standard | | Frequency (Hz) | $\tau(\mu s)$ | # of 9.6-KHz Samples |
|---|---|---|---|---|
| V.21 | Originate | 1080 | 231.481 | 2.2222 |
| | Answer | 1750 | 142.857 | 1.3714 |
| Bell 103 | Originate | 1170 | 213.675 | 2.0513 |
| | Answer | 2125 | 117.647 | 1.1294 |

To minimize the probability of error, it is necessary that the phase delay be as close to $\pi/2$ as possible. An accurate estimate of the fractional part of the delay must be total phase delay. This is achieved by using a single zero FIR filter.

$$R((n - \alpha)T) = GAIN * [R(nT) + B1FSK * R((n-1)T)]) \quad (11)$$

where R(nT) is the nth sample of the received signal R(t)
R((n − α)T) is the estimate of the fractionally delayed signal
n is an integer
α is the desired fractional delay , $0 < \alpha < 1$

The filter coefficient B1FSK and GAIN for the fractional delay filter of each V.21 and Bell 103 carrier are shown in Table 10. The derivation of the gain and filter coefficients are shown in Appendix A.

**Table 10. Time Delay and FIR Filter Coefficients**

| Modem Standard | | Frequency | Fractional Delay 9.6-KHz Sample(__) | Gain | B1FSK |
|---|---|---|---|---|---|
| V.21 | Originate | 1080 | .2222 | .69753 | .32796 |
| | Answer | 1750 | .3714 | 1.00000 | .68889 |
| Bell 103 | Originate | 1170 | .0518 | .57731 | .07175 |
| | Answer | 2125 | .1294 | 1.00000 | .31678 |

B1 and GAIN are stored in data memory locations B1FSK and GAIN, resepectively. The actual implementation is

$$PDEL1 = AGCOUT + B1FSK * PDEL0$$

where AGCOUT is the received signal after the signal level has been compensated by the automatic gain control routine.

$$
\begin{aligned}
\text{AGCOUT} &= \cos[(\omega_c \pm \delta\omega) * nT + \phi\ ] \\
\text{PDEL0} &= \cos[(\omega_c \pm \delta\omega) * (n-1)T + \phi\ ] \\
\text{PDEL1} &= \cos[(\omega_c \pm \delta\omega) * (n-1-\alpha)T + \phi\ ],\ 0 < \alpha < 1 \\
\text{PDEL2} &= \cos[(\omega_c \pm \delta\omega) * (n-2-\alpha)T + \phi\ ]
\end{aligned}
$$

Since AGCOUT, PDEL0, PDEL1, and PDEL2 are consecutive data memory locations, the integer multiples of the 9.6-KHz sample delays are easily achieved by using the data move (DMOV) instruction. PDEL1 is calculated after the demodulator product operation and is not used until the next sample period, a delay of one sample period.

For the low-frequency carriers of the V.21 and Bell 103 standards, a second delay is required and is implemented as DMOV PDEL1, moving the contents of PDEL1 into data memory PDEL2.

When the sample delayed signal (PDEL1 or PDEL2 for the high- or low-frequency carriers, respectively) is generated, it is multiplied by the most recent sample AGCOUT. The product of the multiply is stored in data memory location PROD. PROD is multiplied by GAIN and then filtered by a second-order direct-form, lowpass IIR filter, and the result is stored in location LPFOUT. Further information on digital filters can be found in [12], [13].

Given the lowpass filter output LPFOUT, the FSK demodulator must now estimate the value of the received signal.

In the Data Estimation routine, the following memory location addresses are called:

BDATA — The data estimation for the previous baud.

FSKDAT — Data estimation of the current sample.

BAUDCK — A record of the number of samples presently taken in the current baud. Recall that the sample rate is 9.6 KHz and the baud rate is 300 Hz; so there are 32 samples/baud.

COUNTR — The data estimations of each sample in the current baud are compared to the decision of the previous baud. If these are different, then COUNTR is incremented. If COUNTR reaches 32 before BAUDCK reaches 32, it is assumed that a data transition has occurred, and BDATA is set to the opposite value:

$$\text{BDATA}(N+1) = \text{ABS}[\text{BDATA}(N) - 1]$$

Figure 9 is a flowchart of the data decision source code implementation.

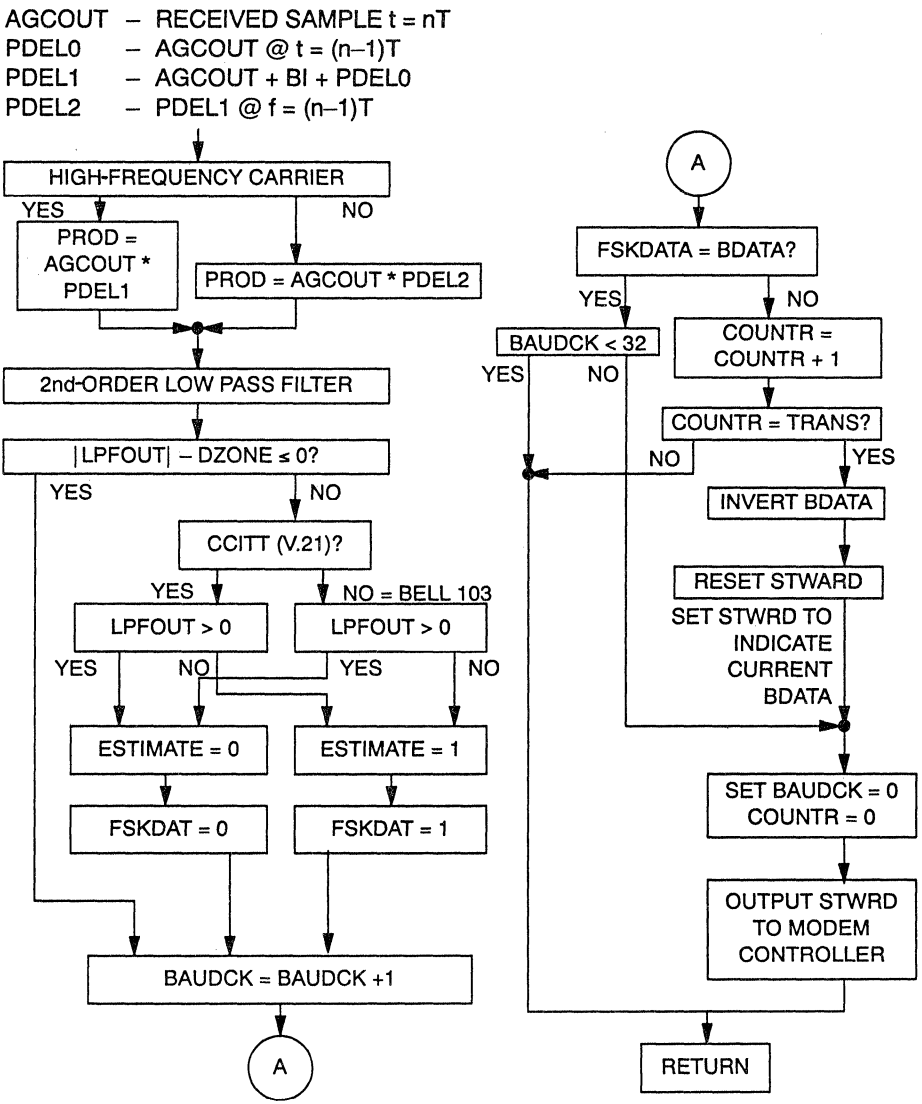



**Figure 9. Data Decision Algorithm Flowchart**

The function of the automatic gain control subroutine AGC is to compensate for amplitude distortions introduced by the telephone system, etc. References [5], [14] provide additional information on AGC.

## Incorporating Additional Functions into the DSP

One of the important tasks the designer faces is incorporating value-added software functions into the DSP source code found in Appendix B.

The software presented here uses only 1.1 Kwords of the 4 Kwords of maskable ROM available on the TMS320C17. This provides you with a significant amount of code space to implement value-added functions.

This software offers a number of hooks that facilitate the easy inclusion of additional software. Note in Table 3 (Modem Controller Commands for the DSP), that the following commands are presently reserved : E, C, B, A, 6, 4, 2, 1, and 0. Each of these commands have bits 0 through 3 undefined. All of these commands can be used by the designer to call additional functions.

You must ensure that the correct modifications are made to the modem controller and modem DSP software. The DSP control command interpreter (CCI) must be modified to recognize and respond to the new commands. The additional functions should be implemented in either a new or the appropriate existing subroutine. The option indicating to the main program that the new subroutine should be called, needs to be provided. This can be done using the STATUS register, or you can define a new register.

You must also ensure that the XDATA word will indicate the present status of the DSP to the modem controller. There are presently a number of unused bits in the XDATA word, so incorporating the modifications in the DSP is straightforward.

Finally, you must ensure that the additional software functions do not exceed the timing requirements imposed by the 9600-KHz sampling frequency.

## Conclusions

This application report presented you with the information required to implement a 300-bps V.21/Bell 103 FSK modem based on a TMS320C17 Digital Signal Processor. Both hardware and software issues were discussed. A summary of the FSK modulation and demodulation algorithms and a basic review of modems were also provided. A discussion about incorporation of additional functions and software into the code provided concluded this report.

Appendix A is a derivation of the FSK demodulator fractional delay filter coefficients. Appendix B is the TMS320C17 source code listing.

## Acknowledgements

The author wishes to acknowledge the contribution of Dr. Amin Haoni of Technekron, Inc., and George Troullinos, and Raj Chirayil of Texas Instruments. This report is based on their work.

## Glossary of Symbols and Abbreviations

bps — Bits per second
FSK — Frequency shift keying
$\omega_c$ — Carrier signal angular velocity
$\delta\omega$ — Modulation shift of angular velocity
t — Time
$\phi$ — Phase shift
$\omega_0$ — Angular velocity transmitted to indicate a 0
$\omega_1$ — Angular velocity transmitted to indicate a 1
$\tau$ — The amount of time the received signal is delayed in the FSK demodulator
$f_0$ — Frequency transmitted to indicate a 0
$f_1$ — Fequency transmitted to indicate a 1
$f_c$ — Carrier frequency
$\alpha$ — Sample fractional delay created by the single FIR filter

## References


[1] *DSP2400 Modem User's Guide*, Texas Instruments Inc. (1988).

[2] "TMS320A2400A Modem Digital Signal Processor Data Sheet", Texas Instruments Inc., (1988).

[3] "TMS70A2400A Modem Controller Data Sheet", Texas Instruments Inc., (1988).

[4] Lee, E.A., and Messerschmitt, D.G., "Digital Communications", Kluwer Academic Publishers (1988).

[5] Bingham, J.A.C., "The Theory and Practice of Modem Design", John Wiley and Sons, (1988).

[6] Proakis, J.G., "Digital Communications", McGraw-Hill (1983).

[7] Troullinos, G., et al., "Theory and Implementation of a Splitband Modem Using the TMS32010" (document number SPRA013), Texas Instruments Inc. (1986).

[8] "TCM29C18,'C19 PCM Codec Data Sheet" (document number SCTS021), Texas Instruments Inc., (1987).

[9] "SC11005 Splitband Filter Data Sheet", Sierra Semiconductors (1986).

[10] *First-Generation TMS320 User's Guide* (document number SPRA013A), Texas Instruments Inc., (1988).

[11] "First-Generation Digital Signal Processors Data Sheet" (document number SPRS009), Texas Instruments Inc., (1987).

[12] "Digital Signal Processing Applications with the TMS320 Family" (document number SPRA012A), pp 27-69, Texas Instruments Inc., (1986).

[13] Parks,T.W., and Burrus, C.S., "Digital Filter Design", John Wiley and Sons Inc. (1987).

[14] Lovrich, A., Troullinos, G., Chirayil, R. "An All Digital Automatic Gain Control", Texas Instruments Inc., *ICASSP Conference Proceedings*, (1988).

# Appendix A

## Calculation of Phase Delay Filter Coefficients

A key element of the FSK demodulator implementation is the $\pi/2$ phase delay of the carrier signal. The effectiveness of the demodulator is highly dependent on the accuracy of the $\pi/2$ phase delay.

In a digital system, it is highly unlikely that the time delay required for the $\pi/2$ phase delay is an exact multiple of the signal sampling period. It will be necessary to introduce phase delays that are a fraction of the sampling period.

To accurately generate the fractional delay, the digital signal processor uses a single zero FIR filter. This appendix derives the coefficients for the single zero FIR.

Given the one zero FIR filter shown in Figure A-1:

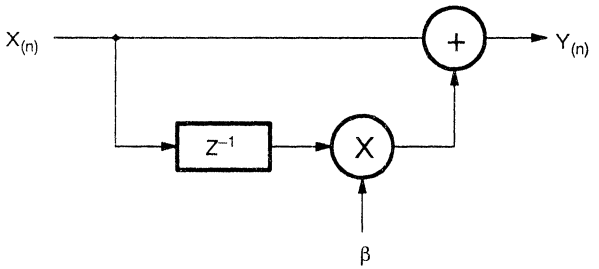



**Figure A-1. One Zero FIR Filter.**

$$Y(n) = X(n) + \beta X(n-1) \tag{A1}$$

therefore

$$\begin{aligned} Y(z) &= X(z) + \beta * z^{-1} X(z) \\ &= X(z) * (1 + \beta z^{-1}) \end{aligned} \tag{A2}$$

The transform of the filter is F(z)

$$F(z) = Y(z)/X(z) = (1 + \beta z^{-1}) \tag{A3}$$

The purpose of this filter is to introduce a precise group delay $\tau$ (delay of the signal envelope) to the received signal $\tau$. is defined as

$$\tau = \frac{-d0(\omega)}{d\omega} \stackrel{\Delta}{=} \text{group delay} \tag{A4}$$

Evaluate F(z) at $z = e^{j\omega}$ to obtain the frequency response.

$$F'(\omega) = F(e^{j\omega}) = 1 + \beta\, e^{-j\omega} \tag{A5}$$

$$F'(\omega) = R(\omega) + jI(\omega) = A(\omega)e^{j\phi(\omega)} \tag{A6}$$

Where $R(\omega)$, $I(\omega)$, $A(\omega)$, and $\phi(\omega)$ are real functions of $\omega$.

$$A(\omega) = |F'(\omega)| = [R(\omega)^2 + I(\omega)^2]^{1/2} \tag{A7}$$

and

$$\phi(\omega) = \arctan\,(I(\omega)/R(\omega)) \tag{A8}$$

Given

$$e^{-j\omega} = \cos\omega - j\sin\omega \tag{A9}$$

Substituting (A9) into (A5)

$$F'(\omega) = 1 + \beta\cos\omega - j\beta\sin\omega \tag{A10}$$

From (A6), (A8), and (A10)

$$\phi(\omega) = \left(\arctan\ \frac{-\beta\sin\omega}{1 + \beta\cos\omega}\right) \tag{A11}$$

Substituting (A11) into (A5)

$$\tau = \frac{-d\ \phi(\omega)}{d} = \frac{-d}{d\omega}\left((\arctan\left(\frac{-\beta\sin\omega}{1 + \beta\cos\omega}\right)\right) \tag{A12}$$

now

$$\frac{d}{dx}(\arctan(u)) = \frac{1}{1 + u^2} * \frac{du}{dx} \tag{A13}$$

therefore

$$\tau = \frac{-1}{1 + \left(\frac{-\beta\sin\omega}{1 + \beta\cos\omega}\right)^2} * \frac{d}{d\omega}\left(\frac{-\beta\sin\omega}{1 + \beta\cos\omega}\right) \tag{A14}$$

$$= \left(\frac{-(1 + \beta\cos\omega)^2}{1 + \beta^2 + 2\beta\cos\omega}\right) * \left(\frac{-\beta\cos\omega - \beta}{(1 + \beta\cos\omega)^2}\right) \tag{A15}$$

$$= \frac{+\beta\,(\beta + \cos\omega)}{1 + \beta^2 + 2\,\beta\cos\omega}$$

Assuming $\tau$ is expressed in terms of sample delays D

$$D = \frac{+ \beta\ (\beta\ +\ \cos\omega)}{(1\ +\ \beta^2\ +\ 2\ \beta\cos\omega)} \tag{A17}$$

Rearranging (A17) and using the quadratic equation to solve for

$$\beta = -\ \frac{(1-2D)\cos\omega\ \pm\ ((1-2D)^2\cos^2\omega\ +\ 4D(1-D))^{1/2}}{2(1-D)} \tag{A18}$$

Given the desired group delay D, and the frequency $f = \omega/2\pi$, the filter coefficient $\beta$ can be determined using equation (A18).

```
*****************************************************************
*
*   APPENDIX B
*
*****************************************************************
*
*   V21/BELL 103 MODEM
*
*   ASSEMBLY LANGUAGE SOFTWARE FOR TMS320C17 IMPLEMENTATION.
*
*   ALL RIGHTS RESERVED BY TEXAS INSTRUMENTS  (C)
*****************************************************************
*
*   VERSION 1.0       01/SEP/89
*
*****************************************************************
*
        .option   x
*
*****************************************************************
*
*   CONSTANT DEFINITIONS
*
*****************************************************************
*
RMMSK:    .set      0C0h
ANMSK:    .set      08h
LMMSK:    .set      0F3h
LDMSK:    .set      04h
NTXMSK:   .set      0CFh
NRCMSK:   .set      03Fh
NSPMSK:   .set      0FCh
TXSH:     .set      04h
RCSH:     .set      06h
*
*****************************************************************
*
*   AGC EQUATES
*
*****************************************************************
*
AGCREF:   .set      05B6h
*
*****************************************************************
*
*   HIGH PASS FILTER CONSTANT
*
*****************************************************************
*
TAU:    .set    14                ; *16 - 14
*
```

# Appendix B

```
*****************************************************************
*
*   DATA MEMORY (RAM) ASSIGNMENTS
*
*****************************************************************
*
STATUS:   .set      0
XDATA:    .set      1
DTFLAG:   .set      2             ; DTMF FLAG
ONE:      .set      3             ; CONSTANT 1
S1CNT:    .set      16            ; S1 DETECTION COUNTER
STWRD:    .set      ONE+1
XSCNTR:   .set      STWRD+1
RSCNTR:   .set      XSCNTR+1      ; REC. BAUD COUNTER
SCNT:     .set      RSCNTR+1      ; NOMINAL BAUD COUNTER
XBITS:    .set      SCNT+1
XOUT:     .set      XBITS+1       ; OUTPUT SAMPLE
RIN:      .set      XOUT+1        ; INPUT SAMPLE
TXFR0:    .set      RIN+1
TXPHS:    .set      TXFR0+1
RXFR0:    .set      TXPHS+1
RXPHS:    .set      RXFR0+1       ; RECEIVE DEMODULATION ANGLE (KH)
TMP4:     .set      RXPHS+1
TMP5:     .set      TMP4+1
*
TMP2:     .set      TMP5+1        ; PARTIAL FILTERED SIGNAL w(n)
TMP0:     .set      TMP2+1
*
TMP3:     .set      TMP0+1
TMP1:     .set      TMP3+1
*
IRCNT:    .set      TMP1+1
TIMIND:   .set      IRCNT+1
*
IR0:      .set      TIMIND+1
*
*****************************************************************
*
*   IN THE FSK MODE, THE TX/RX PARAMETERS SHARE THE SANE MEMORY AS THE
*   EQUALIZER DELAY LINE
*
*****************************************************************
*
OD:       .set      IR0           ; OUTPUT OF PRODUCT DEMODULATOR
AGCOUT:   .set      PROD+1
PDEL0:    .set      AGCOUT+1      ; PRODUCT DEMODULATOR DELAY LINE
PDEL1:    .set      PDEL0+1
PDEL2:    .set      PDEL1+1
LPDEL0:   .set      PDEL2+1       ; FSK LOWPASS DEMOD DELAY LINE
LPDEL1:   .set      LPDEL0+1
LPDEL2:   .set      LPDEL1+1
GAIN:     .set      LPDEL2+1      ; GAIN OF FSK DEMOD FILTER (0.5 OR 1.0)
FSKDAT:   .set      GAIN+1        ; OUTPUT OF FSK SLICER (X111)
BAUDCK:   .set      FSKDAT+1      ; BAUD CLOCK FOR FSK TIMING RECOVERY
```

```
BDATA:   .set    BAUDCK+1      ; CURRENT BAUD FSK DATA (X111)
TRANS:   .set    BDATA+1       ; = 20 USED IN TIMING DECISION
COUNTR:  .set    TRANS+1       ; TRANSITION COUNTER FOR RX TIMING
B1FSK:   .set    COUNTR+1      ; COEF B1 OF PHASE ADJUST FIR
A1FSK:   .set    B1FSK+1       ; COEF A1 OF FSK DEMOD FILTER
A2FSK:   .set    A1FSK+1       ; COEF A2 OF FSK DEMOD FILTER
F0ADD:   .set    A2FSK+1       ; ADDRESS OF 0 FREQUENCY
F1ADD:   .set    F0ADD+1       ; ADDRESS OF 1 FREQUENCY
FSKFLG:  .set    F1ADD+1       ; FLAG TO INDICATE FSK OPERATION
OAFLAG:  .set    FSKFLG+1      ; ORIGINATE_/ANSWER MODE FLAG
DZONE:   .set    OAFLAG+1      ; DEAD ZONE OF SLICER
LPFOUT:  .set    DZONE+1
CCITT:   .set    LPFOUT+1
*
*******************************************************************************
*
*    AGC RAM
*
*******************************************************************************
*
ALPHA:   .set    CCITT+1
AVESQR:  .set    ALPHA+1
GN:      .set    AVESQR+1
HYST:    .set    GN+1
*
*******************************************************************************
*
*    BAUD COUNTER
*
*******************************************************************************
*
BDCNTR:  .set    HYST+1
*
*******************************************************************************
*
*    PAGE 1 RAM ASSIGNMENTS
*
*******************************************************************************
*
X1:      .set    0
X2:      .set    X1+1
ST:      .set    X2+1
STLSB:   .set    ST+1
POSSM:   .set    STLSB+1
NEGSM:   .set    POSSM+1
*
*******************************************************************************
*
*    DIAGNOSTICS!
*
*******************************************************************************
*
DTMFL:   .set    IRCNT
DTMFH:   .set    TIMIND
*
         .text
         B       START
*
*******************************************************************************
*
*    COEFFICIENTS STORED IN PROGRAM ROM
*
*******************************************************************************
*    PHASE ANGLE LOOK-UP TABLE
*
*    CONTAINS THE INCREMENT USED AS XDELTA IN THE CARRIER GENERATION. THE
*    TABLE GIVES THE ANGLES FOR 300 BPS, (V21, BELL 103) ORIG/ANSW MODES.
*
*    NOTE: INCREMENTS ARE IN UNITS OF PI/128 TIMES 256 (UPPER 8 BITS OF DATA
*          EVENTUALLY REPRESENT TABLE INDEX)
*
*******************************************************************************
*
FSKTBL:  .set    $
         .word   021ddh        ; FSK, 103, ORIGINATE, 1  1270 HZ
         .word   01c89h        ; FSK, 103, ORIGINATE, 0  1070 HZ
         .word   01852h        ; COEFF B1 FOR 2125 HZ FREQ.(0.4293)
         .word   07fffh        ; GAIN FOR FSK DEMOD LPF (1)
*
         .word   01a22h        ; FSK, V.21, ORIGINATE, 1  1180 HZ
         .word   01f77h        ; FSK, V.21, ORIGINATE, 0  980 HZ
         .word   22574h        ; COEFF B1 FOR 1750 HZ FREQ.(0.63)
         .word   07fffh        ; GAIN FOR FSK DEMOD LPF (0.5)
*
         .word   03b55h        ; FSK, 103, ANSWER, 1  2225 HZ
         .word   03600h        ; FSK, 103, ANSWER, 0  2025 HZ
         .word   0666h         ; COEFF B1 FOR 1170 HZ FREQ.(0.3891)
         .word   049e5h        ; GAIN FOR FSK DEMOD LPF (0.8323)
*
         .word   02c00h        ; FSK, V.21, ANSWER, 1  1850 HZ
         .word   03155h        ; FSK, V.21, ANSWER, 0  1650 HZ
         .word   10747         ; COEFF B1 FOR 1080 HZ FREQ.(0.3)
         .word   22857         ; GAIN FOR FSK DEMOD LPF (0.5)
*
F21:     .word   03800h        ; 2100 HZ ANSWER TONE
F22:     .word   03b55h        ; 2225 HZ ANSWER TONE
ZONE:    .word   50            ; DEAD ZONE OF FSK DEMOD SLICER
FSKA1:   .word   04989h        ; COEF A1 OF FSK DEMOD FILTER
FSKA2:   .word   0aadbh        ; COEF A2 OF FSK DEMOD FILTER
*
```

```
*************************************************************************
*
*    COMMAND MODE SUBROUTINE LOOK-UP TABLE
*
*    IN COMMAND MODE, EACH COMMAND BYTE CORRESPONDS TO AN ENTRY HERE WHICH
*    CONTAINS THE NAME OF THE APPROPRIATE SUBROUTINE TO CALL TO EXECUTE THE
*    COMMAND. THE NUMBER OF UNDEFINED SUBROUTINES (AT ADDRESSES 00, 01, 02,
*    04, 06, 0A, 0B, 0C, AND 0Eh) MEANS INCREASED SYSTEM EXPANSION IS EASILY
*    ACCOMODATED.
*
*************************************************************************
*
CMDTBL:  .set     $
         .word    NONE          ; 00h NOT DEFINED
         .word    NONE          ; 1Xh NOT DEFINED
         .word    NONE          ; 2Xh NOT DEFINED
         .word    RESET         ; 3Xh  RESET THE DSP
         .word    NONE          ; 4Xh NOT DEFINED
         .word    FSKSET        ; 5Xh  FSK DATA MODE
         .word    NONE          ; 6Xh NOT DEFINED
         .word    RMODE         ; 7Xh  RECVR. MODE SETUP
         .word    XMODE         ; 8hX  TRANS. MODE SETUP
         .word    DTMF          ; 9Xh  TRANSMIT DTMF CHARACTER
         .word    NONE          ; AXh NOT DEFINED
         .word    NONE          ; BXh NOT DEFINED
         .word    NONE          ; CXh NOT DEFINED
         .word    OPER          ; DXh  SET OPERATING MODE
         .word    NONE          ; EXh NOT DEFINED
         .word    PROTO         ; FXh  PROTOCOL SELECT
*
*************************************************************************
*
*    RECEIVER SUBROUTINE TABLE
*
*    USED TO SELECT DISTRIBUTED RECEIVER TASKS. THIS IS IMPORTANT IN HIGHER
*    FUNCTIONALITY SYSTEMS THAT CANNOT IMPLEMENT TASKS IN A SINGLE SAMPLE
*    PERIOD.
*
*************************************************************************
*
RSEQTB:  .set     $
         .word    NONE
         .word    NONE
         .word    NONE
         .word    NONE
         .word    NONE
         .word    NONE
         .word    NONE
         .word    NONE
         .word    NONE
         .word    NONE
         .word    RTSK10
         .word    NONE
         .word    NONE
         .word    NONE
         .word    NONE
*
*************************************************************************
*
*    SERIAL PORT CONTROL REGISTER DATA
*
*************************************************************************
*
DA1:     .word    0390Fh
DA2:     .word    02CBEh        ; MODIFIED TO USE H/W COMPANDING
DA3:     .word    0380Fh
*
*************************************************************************
*
*    AGC DATA
*
*    THE FOLLOWING VALUES ARE THE CALCULATED VALUES WITH NO WINDOW
*    IMPLEMENTATION AROUND THE THEORETICAL ENERGY BAND OF QAM SIGNALS.
*    HOWEVER, DUE TO ISI EFFECTS, WINDOWS ARE REQUIRED. FOR PROPER OPERATION,
*    THESE VALUES MUST BE ADJUSTED FOR APPROXIMATELY 3 DB DIFFERENCE IN SIGNAL
*    ENERGY LEVELS TO COMPENSATE FOR THE WINDOWS.
*
*************************************************************************
*
THRES1   .word    23BBh         ; -48.0 DBM REC. LEV. (A=35.73)
THRES2   .word    1548h         ; -43.5 DBM REC. LEV. (A=21.28)
THRES3   .word    240h          ; -24 DBM REC. LEV. (AFE=ON,A=2.25)
THRES4   .word    8FBh          ; -24 DBM REC. LEV. (AFE=OFF,A=8.98)
THRES5   .word    1417h         ; -31 DBM REC. LEV. (AFE=OFF,A=20.09)
THRES6   .word    50Dh          ; -31 DBM REC. LEV. (AFE=ON,A=5.05)
*
MAXALP:  .word    038A1h        ; -52.0 DBM RECEIVE LEVEL
PSM:     .word    01EA6h        ; BAUD ENERGY ERROR LEVEL 10628
NSM:     .word    0113h         ; BAUD ENERGY ERROR LEVEL 1
THRES1:  .word    0194Ch        ; -48.0 DBM REC. LEV. (A=25.30)
THRES2:  .word    0FF5h         ; -43.5 DBM REC. LEV. (A=15.96)
THRES3:  .word    0199h         ; -24 DBM REC. LEV. (AFE=ON,A=1.60)
THRES4:  .word    06B0h         ; -24 DBM REC. LEV. (AFE=OFF,A=6.69)
THRES5:  .word    0E00h         ; -31 DBM REC. LEV. (AFE=OFF,A=14.00)
THRES6:  .word    0490h         ; -31 DBM REC. LEV. (AFE=ON,A=4.56)
HYSINC:  .word    0F0Fh         ; 1/17 OF 65536
*
```

```
*************************************************************************
*
*   DTMF TONE TABLE:
*
*                       FIRST ENTRY REPRESENTS LOW FREQUENCY
*                       SECOND ENTRY REPRESENTS HIGH FREQUENCY
*
*   DELTA = (F / F ) * N
*               S
*
*   WITH N = 256  TABLE SIZE
*        F = 9600 HZ
*         S
*        F = FREQUENCY OF INTEREST
*
*************************************************************************
*
*   DATA FORMAT IS S7.8 TO BE AS STEP SIZE. THE TABLE ENTRIES ARE HOWEVER,
*   TREATED AS 16 BIT UNSIGNED INTEGERS. A MULTIPLICATION OF DELTA BY 256
*   DOES THE NECESSARY CONVERSION IN FORMATS.
*
*************************************************************************
*
TONTBL:  .word    01918h        ; 0 LOW FREQ
         .word    023A0h
         .word    02A0h         ; LOW FREQ GAIN
         .word    03A0h         ; HI FREQ GAIN
*
         .word    01296h        ; 1
         .word    0203Eh
         .word    0800h         ; LOW FREQ GAIN
         .word    0493h         ; HI FREQ GAIN
*
         .word    01296h        ; 2
         .word    023A0h
         .word    0800h         ; LOW FREQ GAIN
         .word    0493h         ; HI FREQ GAIN
*
         .word    01296h        ; 3
         .word    02762h
         .word    08A0h         ; LOW FREQ GAIN
         .word    0493h         ; HI  FREQ GAIN
*
         .word    01488h        ; 4
         .word    0203Eh
         .word    02E0h         ; LOW FREQ GAIN
         .word    03A0h         ; HI FREQ GAIN
*
         .word    01488h        ; 5
         .word    023A0h
         .word    0340h         ; LOW FREQ GAIN
         .word    0420h         ; HI FREQ GAIN
*
         .word    01488h        ; 6
         .word    02762h
         .word    0340h         ; LOW FREQ GAIN
         .word    03A0h         ; HI FREQ GAIN
*
         .word    016B8h        ; 7
         .word    0203Eh
         .word    02A0h         ; LOW FREQ GAIN
         .word    0370h         ; HI FREQ GAIN
*
         .word    016B8h        ; 8
         .word    023A0h
         .word    02A0h         ; LOW FREQ GAIN
         .word    03D0h         ; HI FREQ GAIN
*
         .word    016B8h        ; 9
         .word    02762h
         .word    0290h         ; LOW FREQ GAIN
         .word    0300h         ; HI FREQ GAIN
*
         .word    01296h        ; A
         .word    02B8Ch
         .word    0493h         ; LOW FREQ GAIN
         .word    0493h         ; HI FREQ GAIN
*
         .word    01488h        ; B
         .word    02B8Ch
         .word    0493h         ; LOW FREQ GAIN
         .word    0493h         ; HI FREQ GAIN
*
         .word    016B8h        ; C
         .word    02B8Ch
         .word    0493h         ; LOW FREQ GAIN
         .word    0493h         ; HI FREQ GAIN
*
         .word    01918h        ; D
         .word    02B8Ch
         .word    0493h         ; LOW FREQ GAIN
         .word    0493h         ; HI FREQ GAIN
*
         .word    01918h        ; E (*)
         .word    0203Eh
         .word    0300h         ; LOW FREQ GAIN
         .word    0300h         ; HI FREQ GAIN
*
         .word    01918h        ; F (#)
         .word    02762h
         .word    0300h         ; LOW FREQ GAIN
         .word    0300h         ; HI FREQ GAIN
*
```

```
*************************************************************************
*
*    ADDITIONAL TABLES
*
*************************************************************************
*
        .copy     "COSTBL.A00"   ; COSINE FUNCTION TABLE
*
*************************************************************************
*
*    COSINE LOOKUP TABLE:
*
*    257 ENTRIES OVER THE RANGE [0,PI]. THE RESOLUTION OF THE TABLE IS
*    THEREFORE:
*
*    (180 / 256 ) = 0.703125 DEGREES
*
*************************************************************************
*
COSOFF:   .set      $              ; COSINE TABLE LENGTH = 512
          .word     07FFFh         ; ANGLE =   0.0000  COSINE = 1.000000
          .word     07FFEh         ; ANGLE =   0.7031  COSINE = 0.999925
          .word     07FF6h         ; ANGLE =   1.4063  COSINE = 0.999699
          .word     07FEAh         ; ANGLE =   2.1094  COSINE = 0.999322
          .word     07FD9h         ; ANGLE =   2.8125  COSINE = 0.998795
          .word     07FC2h         ; ANGLE =   3.5156  COSINE = 0.998118
          .word     07FA7h         ; ANGLE =   4.2188  COSINE = 0.997291
          .word     07F87h         ; ANGLE =   4.9219  COSINE = 0.996313
          .word     07F62h         ; ANGLE =   5.6250  COSINE = 0.995185
          .word     07F38h         ; ANGLE =   6.3281  COSINE = 0.993907
          .word     07F0Ah         ; ANGLE =   7.0313  COSINE = 0.992480
          .word     07ED6h         ; ANGLE =   7.7344  COSINE = 0.990903
          .word     07E9Dh         ; ANGLE =   8.4375  COSINE = 0.989177
          .word     07E60h         ; ANGLE =   9.1406  COSINE = 0.987301
          .word     07E1Eh         ; ANGLE =   9.8438  COSINE = 0.985278
          .word     07dDDh         ; ANGLE =  10.5469  COSINE = 0.983106
          .word     07D8Ah         ; ANGLE =  11.2500  COSINE = 0.980785
          .word     07D3Ah         ; ANGLE =  11.9531  COSINE = 0.978317
          .word     07CE4h         ; ANGLE =  12.6563  COSINE = 0.975702
          .word     07C89h         ; ANGLE =  13.3594  COSINE = 0.972940
          .word     07C2Ah         ; ANGLE =  14.0625  COSINE = 0.970031
          .word     07BC6h         ; ANGLE =  14.7656  COSINE = 0.966976
          .word     07B5Dh         ; ANGLE =  15.4688  COSINE = 0.963776
          .word     07AEFh         ; ANGLE =  16.1719  COSINE = 0.960431
          .word     07A7Dh         ; ANGLE =  16.8750  COSINE = 0.956940
          .word     07A06h         ; ANGLE =  17.5781  COSINE = 0.953306
          .word     0798Ah         ; ANGLE =  18.2813  COSINE = 0.949528
          .word     0790Ah         ; ANGLE =  18.9844  COSINE = 0.945607
          .word     07885h         ; ANGLE =  19.6875  COSINE = 0.941544
          .word     077FBh         ; ANGLE =  20.3906  COSINE = 0.937339
          .word     0776Ch         ; ANGLE =  21.0938  COSINE = 0.932993
          .word     076D9h         ; ANGLE =  21.7969  COSINE = 0.928506
          .word     07642h         ; ANGLE =  22.5000  COSINE = 0.923880
          .word     075A6h         ; ANGLE =  23.2031  COSINE = 0.919114
          .word     07505h         ; ANGLE =  23.9063  COSINE = 0.914210
          .word     07460h         ; ANGLE =  24.6094  COSINE = 0.909168
          .word     073B6h         ; ANGLE =  25.3125  COSINE = 0.903989
          .word     07308h         ; ANGLE =  26.0156  COSINE = 0.898675
          .word     07255h         ; ANGLE =  26.7188  COSINE = 0.893224
          .word     0719Eh         ; ANGLE =  27.4219  COSINE = 0.887640
          .word     070E3h         ; ANGLE =  28.1250  COSINE = 0.881921
          .word     07023h         ; ANGLE =  28.8281  COSINE = 0.876070
          .word     06F5Fh         ; ANGLE =  29.5313  COSINE = 0.870087
          .word     06E97h         ; ANGLE =  30.2344  COSINE = 0.863973
          .word     06DCAh         ; ANGLE =  30.9375  COSINE = 0.857729
          .word     06CF9h         ; ANGLE =  31.6406  COSINE = 0.851355
          .word     06C24h         ; ANGLE =  32.3438  COSINE = 0.844854
          .word     06B4Bh         ; ANGLE =  33.0469  COSINE = 0.838225
          .word     06A6Eh         ; ANGLE =  33.7500  COSINE = 0.831470
          .word     0698Ch         ; ANGLE =  34.4531  COSINE = 0.824589
          .word     068A7h         ; ANGLE =  35.1563  COSINE = 0.817585
          .word     067BDh         ; ANGLE =  35.8594  COSINE = 0.810457
          .word     066D0h         ; ANGLE =  36.5625  COSINE = 0.803208
          .word     065DEh         ; ANGLE =  37.2656  COSINE = 0.795837
          .word     064E9h         ; ANGLE =  37.9688  COSINE = 0.788347
          .word     063EFh         ; ANGLE =  38.6719  COSINE = 0.780737
          .word     062F2h         ; ANGLE =  39.3750  COSINE = 0.773011
          .word     061F1h         ; ANGLE =  40.0781  COSINE = 0.765168
          .word     060ECh         ; ANGLE =  40.7813  COSINE = 0.757209
          .word     05FE4h         ; ANGLE =  41.4844  COSINE = 0.749137
          .word     05ED7h         ; ANGLE =  42.1875  COSINE = 0.740951
          .word     05DC8h         ; ANGLE =  42.8906  COSINE = 0.732655
          .word     05CB4h         ; ANGLE =  43.5938  COSINE = 0.724247
          .word     05B9Dh         ; ANGLE =  44.2969  COSINE = 0.715731
          .word     05A82h         ; ANGLE =  45.0000  COSINE = 0.707107
          .word     05964h         ; ANGLE =  45.7031  COSINE = 0.698377
          .word     05843h         ; ANGLE =  46.4063  COSINE = 0.689541
          .word     0571Eh         ; ANGLE =  47.1094  COSINE = 0.680601
          .word     055F6h         ; ANGLE =  47.8125  COSINE = 0.671559
          .word     054CAh         ; ANGLE =  48.5156  COSINE = 0.662416
          .word     0539Bh         ; ANGLE =  49.2188  COSINE = 0.653173
          .word     05269h         ; ANGLE =  49.9219  COSINE = 0.643832
          .word     05134h         ; ANGLE =  50.6250  COSINE = 0.634394
          .word     04FFBh         ; ANGLE =  51.3281  COSINE = 0.624860
          .word     04EC0h         ; ANGLE =  52.0313  COSINE = 0.615232
          .word     04D81h         ; ANGLE =  52.7344  COSINE = 0.605512
          .word     04C40h         ; ANGLE =  53.4375  COSINE = 0.595700
          .word     04AFBh         ; ANGLE =  54.1406  COSINE = 0.585799
          .word     049B4h         ; ANGLE =  54.8438  COSINE = 0.575809
          .word     0486Ah         ; ANGLE =  55.5469  COSINE = 0.565733
          .word     0471Dh         ; ANGLE =  56.2500  COSINE = 0.555571
          .word     045CDh         ; ANGLE =  56.9531  COSINE = 0.545326
          .word     0447Bh         ; ANGLE =  57.6563  COSINE = 0.534998
          .word     04326h         ; ANGLE =  58.3594  COSINE = 0.524590
          .word     041CEh         ; ANGLE =  59.0625  COSINE = 0.514103
          .word     04074h         ; ANGLE =  59.7656  COSINE = 0.503539
          .word     03F17h         ; ANGLE =  60.4688  COSINE = 0.492899
          .word     03DB8h         ; ANGLE =  61.1719  COSINE = 0.482184
```

```
        .word   03C57h      ; ANGLE =  61.8750  COSINE = 0.471397
        .word   03AF3h      ; ANGLE =  62.5781  COSINE = 0.460539
        .word   0398Dh      ; ANGLE =  63.2813  COSINE = 0.449612
        .word   03825h      ; ANGLE =  63.9844  COSINE = 0.438617
        .word   036BAh      ; ANGLE =  64.6875  COSINE = 0.427556
        .word   0354Eh      ; ANGLE =  65.3906  COSINE = 0.416430
        .word   033DFh      ; ANGLE =  66.0938  COSINE = 0.405242
        .word   0326Eh      ; ANGLE =  66.7969  COSINE = 0.393992
        .word   030FCh      ; ANGLE =  67.5000  COSINE = 0.382684
        .word   02F87h      ; ANGLE =  68.2031  COSINE = 0.371318
        .word   02E11h      ; ANGLE =  68.9063  COSINE = 0.359895
        .word   02C99h      ; ANGLE =  69.6094  COSINE = 0.348419
        .word   02B1Fh      ; ANGLE =  70.3125  COSINE = 0.336890
        .word   029A4h      ; ANGLE =  71.0156  COSINE = 0.325310
        .word   02827h      ; ANGLE =  71.7188  COSINE = 0.313682
        .word   026A8h      ; ANGLE =  72.4219  COSINE = 0.302006
        .word   02528h      ; ANGLE =  73.1250  COSINE = 0.290285
        .word   023A7h      ; ANGLE =  73.8281  COSINE = 0.278520
        .word   02224h      ; ANGLE =  74.5313  COSINE = 0.266713
        .word   0209Fh      ; ANGLE =  75.2344  COSINE = 0.254866
        .word   01F1Ah      ; ANGLE =  75.9375  COSINE = 0.242980
        .word   01D93h      ; ANGLE =  76.6406  COSINE = 0.231058
        .word   01C0Ch      ; ANGLE =  77.3438  COSINE = 0.219101
        .word   01A83h      ; ANGLE =  78.0469  COSINE = 0.207111
        .word   018F9h      ; ANGLE =  78.7500  COSINE = 0.195090
        .word   0176Eh      ; ANGLE =  79.4531  COSINE = 0.183040
        .word   015E2h      ; ANGLE =  80.1563  COSINE = 0.170962
        .word   01455h      ; ANGLE =  80.8594  COSINE = 0.158858
        .word   012C8h      ; ANGLE =  81.5625  COSINE = 0.146730
        .word   0113Ah      ; ANGLE =  82.2656  COSINE = 0.134580
        .word   0FABh       ; ANGLE =  82.9688  COSINE = 0.122410
        .word   0E1Ch       ; ANGLE =  83.6719  COSINE = 0.110222
        .word   0C8Ch       ; ANGLE =  84.3750  COSINE = 0.098017
        .word   0AFBh       ; ANGLE =  85.0781  COSINE = 0.085797
        .word   096Bh       ; ANGLE =  85.7813  COSINE = 0.073564
        .word   07D9h       ; ANGLE =  86.4844  COSINE = 0.061320
        .word   0648h       ; ANGLE =  87.1875  COSINE = 0.049067
        .word   04B6h       ; ANGLE =  87.8906  COSINE = 0.036807
        .word   0324h       ; ANGLE =  88.5938  COSINE = 0.024541
        .word   0192h       ; ANGLE =  89.2969  COSINE = 0.012271
SINE0:  .set    $
        .word   00h         ; ANGLE =  90.0000  COSINE = -.000001
        .word   0FE6Eh      ; ANGLE =  90.7031  COSINE = -.012272
        .word   0FCDCh      ; ANGLE =  91.4063  COSINE = -.024542
        .word   0FB4Ah      ; ANGLE =  92.1094  COSINE = -.036808
        .word   0F9B8h      ; ANGLE =  92.8125  COSINE = -.049069
        .word   0F827h      ; ANGLE =  93.5156  COSINE = -.061322
        .word   0F695h      ; ANGLE =  94.2188  COSINE = -.073566
        .word   0F505h      ; ANGLE =  94.9219  COSINE = -.085798
        .word   0F374h      ; ANGLE =  95.6250  COSINE = -.098018
        .word   0F1E4h      ; ANGLE =  96.3281  COSINE = -.110223
        .word   0F055h      ; ANGLE =  97.0313  COSINE = -.122412
        .word   0EEC6h      ; ANGLE =  97.7344  COSINE = -.134582
        .word   0ED38h      ; ANGLE =  98.4375  COSINE = -.146732
        .word   0EBAAh      ; ANGLE =  99.1406  COSINE = -.158859
        .word   0EA1Eh      ; ANGLE =  99.8438  COSINE = -.170963
        .word   0E892h      ; ANGLE = 100.5469  COSINE = -.183041
        .word   0E707h      ; ANGLE = 101.2500  COSINE = -.195092
        .word   0E57Dh      ; ANGLE = 101.9531  COSINE = -.207113
        .word   0E3F4h      ; ANGLE = 102.6563  COSINE = -.219103
        .word   0E26Dh      ; ANGLE = 103.3594  COSINE = -.231060
        .word   0E0E6h      ; ANGLE = 104.0625  COSINE = -.242982
        .word   0DF61h      ; ANGLE = 104.7656  COSINE = -.254867
        .word   0DDDCh      ; ANGLE = 105.4688  COSINE = -.266714
        .word   0DC59h      ; ANGLE = 106.1719  COSINE = -.278521
        .word   0DAD8h      ; ANGLE = 106.8750  COSINE = -.290286
        .word   0D958h      ; ANGLE = 107.5781  COSINE = -.302008
        .word   0D7D9h      ; ANGLE = 108.2813  COSINE = -.313683
        .word   0D65Ch      ; ANGLE = 108.9844  COSINE = -.325312
        .word   0D4E1h      ; ANGLE = 109.6875  COSINE = -.336892
        .word   0D367h      ; ANGLE = 110.3906  COSINE = -.348420
        .word   0D1EFh      ; ANGLE = 111.0938  COSINE = -.359897
        .word   0D079h      ; ANGLE = 111.7969  COSINE = -.371319
        .word   0CF04h      ; ANGLE = 112.5000  COSINE = -.382685
        .word   0CD92h      ; ANGLE = 113.2031  COSINE = -.393994
        .word   0CC21h      ; ANGLE = 113.9063  COSINE = -.405243
        .word   0CAB2h      ; ANGLE = 114.6094  COSINE = -.416431
        .word   0C946h      ; ANGLE = 115.3125  COSINE = -.427557
        .word   0C7DBh      ; ANGLE = 116.0156  COSINE = -.438618
        .word   0C673h      ; ANGLE = 116.7188  COSINE = -.449613
        .word   0C50Dh      ; ANGLE = 117.4219  COSINE = -.460541
        .word   0C3A9h      ; ANGLE = 118.1250  COSINE = -.471399
        .word   0C248h      ; ANGLE = 118.8281  COSINE = -.482186
        .word   0C0E9h      ; ANGLE = 119.5313  COSINE = -.492900
        .word   0BF8Ch      ; ANGLE = 120.2344  COSINE = -.503540
        .word   0BE32h      ; ANGLE = 120.9375  COSINE = -.514105
        .word   0BCDAh      ; ANGLE = 121.6406  COSINE = -.524592
        .word   0BB85h      ; ANGLE = 122.3438  COSINE = -.535000
        .word   0BA33h      ; ANGLE = 123.0469  COSINE = -.545327
        .word   0B8E3h      ; ANGLE = 123.7500  COSINE = -.555572
        .word   0B796h      ; ANGLE = 124.4531  COSINE = -.565734
        .word   0B64Ch      ; ANGLE = 125.1563  COSINE = -.575810
        .word   0B505h      ; ANGLE = 125.8594  COSINE = -.585800
        .word   0B3C0h      ; ANGLE = 126.5625  COSINE = -.595701
        .word   0B27Fh      ; ANGLE = 127.2656  COSINE = -.605513
        .word   0B140h      ; ANGLE = 127.9688  COSINE = -.615234
        .word   0B005h      ; ANGLE = 128.6719  COSINE = -.624862
        .word   0AECCh      ; ANGLE = 129.3750  COSINE = -.634395
        .word   0AD97h      ; ANGLE = 130.0781  COSINE = -.643834
        .word   0AC65h      ; ANGLE = 130.7813  COSINE = -.653175
        .word   0AB36h      ; ANGLE = 131.4844  COSINE = -.662418
        .word   0AA0Ah      ; ANGLE = 132.1875  COSINE = -.671561
        .word   0A8E2h      ; ANGLE = 132.8906  COSINE = -.680603
        .word   0A7BDh      ; ANGLE = 133.5938  COSINE = -.689543
        .word   0A69Ch      ; ANGLE = 134.2969  COSINE = -.698378
        .word   0A57Dh      ; ANGLE = 135.0000  COSINE = -.707109
        .word   0A463h      ; ANGLE = 135.7031  COSINE = -.715733
        .word   0A34Ch      ; ANGLE = 136.4063  COSINE = -.724249
```

```
        .word   0A238h      ; ANGLE = 137.1094  COSINE = -.732656
        .word   0A128h      ; ANGLE = 137.8125  COSINE = -.740953
        .word   0A01Ch      ; ANGLE = 138.5156  COSINE = -.749138
        .word   09F14h      ; ANGLE = 139.2188  COSINE = -.757211
        .word   09E0Fh      ; ANGLE = 139.9219  COSINE = -.765169
        .word   09D0Eh      ; ANGLE = 140.6250  COSINE = -.773012
        .word   09C11h      ; ANGLE = 141.3281  COSINE = -.780739
        .word   09B17h      ; ANGLE = 142.0313  COSINE = -.788348
        .word   09A22h      ; ANGLE = 142.7344  COSINE = -.795839
        .word   09930h      ; ANGLE = 143.4375  COSINE = -.803210
        .word   09843h      ; ANGLE = 144.1406  COSINE = -.810459
        .word   09759h      ; ANGLE = 144.8438  COSINE = -.817587
        .word   09674h      ; ANGLE = 145.5469  COSINE = -.824591
        .word   09592h      ; ANGLE = 146.2500  COSINE = -.831472
        .word   094B5h      ; ANGLE = 146.9531  COSINE = -.838227
        .word   093DCh      ; ANGLE = 147.6563  COSINE = -.844856
        .word   09307h      ; ANGLE = 148.3594  COSINE = -.851357
        .word   09236h      ; ANGLE = 149.0625  COSINE = -.857730
        .word   09169h      ; ANGLE = 149.7656  COSINE = -.863975
        .word   090A1h      ; ANGLE = 150.4688  COSINE = -.870089
        .word   08FDDh      ; ANGLE = 151.1719  COSINE = -.876072
        .word   08F1Dh      ; ANGLE = 151.8750  COSINE = -.881923
        .word   08E62h      ; ANGLE = 152.5781  COSINE = -.887641
        .word   08DABh      ; ANGLE = 153.2813  COSINE = -.893226
        .word   08CF8h      ; ANGLE = 153.9844  COSINE = -.898676
        .word   08C4Ah      ; ANGLE = 154.6875  COSINE = -.903991
        .word   08BA0h      ; ANGLE = 155.3906  COSINE = -.909170
        .word   08AFBh      ; ANGLE = 156.0938  COSINE = -.914211
        .word   08A5Ah      ; ANGLE = 156.7969  COSINE = -.919115
        .word   089BEh      ; ANGLE = 157.5000  COSINE = -.923881
        .word   08927h      ; ANGLE = 158.2031  COSINE = -.928508
        .word   08894h      ; ANGLE = 158.9063  COSINE = -.932994
        .word   08805h      ; ANGLE = 159.6094  COSINE = -.937341
        .word   0877Bh      ; ANGLE = 160.3125  COSINE = -.941546
        .word   086F6h      ; ANGLE = 161.0156  COSINE = -.945609
        .word   08676h      ; ANGLE = 161.7188  COSINE = -.949530
        .word   085FAh      ; ANGLE = 162.4219  COSINE = -.953307
        .word   08583h      ; ANGLE = 163.1250  COSINE = -.956942
        .word   08511h      ; ANGLE = 163.8281  COSINE = -.960432
        .word   084A3h      ; ANGLE = 164.5313  COSINE = -.963777
        .word   0843Ah      ; ANGLE = 165.2344  COSINE = -.966978
        .word   083D6h      ; ANGLE = 165.9375  COSINE = -.970032
        .word   08377h      ; ANGLE = 166.6406  COSINE = -.972941
        .word   0831Ch      ; ANGLE = 167.3438  COSINE = -.975703
        .word   082C6h      ; ANGLE = 168.0469  COSINE = -.978318
        .word   08276h      ; ANGLE = 168.7500  COSINE = -.980786
        .word   0822Ah      ; ANGLE = 169.4531  COSINE = -.983106
        .word   081E2h      ; ANGLE = 170.1563  COSINE = -.985278
        .word   081A0h      ; ANGLE = 170.8594  COSINE = -.987302
        .word   08163h      ; ANGLE = 171.5625  COSINE = -.989177
        .word   0812Ah      ; ANGLE = 172.2656  COSINE = -.990903
        .word   080F6h      ; ANGLE = 172.9688  COSINE = -.992480
        .word   080C8h      ; ANGLE = 173.6719  COSINE = -.993907
        .word   0809Eh      ; ANGLE = 174.3750  COSINE = -.995185
        .word   08079h      ; ANGLE = 175.0781  COSINE = -.996313
        .word   08059h      ; ANGLE = 175.7813  COSINE = -.997291
        .word   0803Eh      ; ANGLE = 176.4844  COSINE = -.998118
        .word   08027h      ; ANGLE = 177.1875  COSINE = -.998796
        .word   08016h      ; ANGLE = 177.8906  COSINE = -.999323
        .word   0800Ah      ; ANGLE = 178.5938  COSINE = -.999699
        .word   08002h      ; ANGLE = 179.2969  COSINE = -.999925
        .word   08000h      ; ANGLE = 180.0000  COSINE = -1.000000
*
*************************************************************************
*
*   MAIN PROGRAM
*
*************************************************************************
*
*   INITIALIZATION CODE
*
*************************************************************************
*
START   DINT
*
*************************************************************************
*
*   CLEAR ALL RAM
*
*************************************************************************
*
        LDPK    0
        LARP    AR1
        LARK    AR1,143
        ZAC
CRAM    SACL    *
        BANZ    CRAM
        SACL    0           ; CLEAR RAM 0
*
*************************************************************************
*
*   START INITIALIZATION CODE - FIRST INITIALIZE PAGE 1 DATA
*
*************************************************************************
*
        LDPK    1           ; WINDOW FOR SLEW MODE IN
        LACK    PSM
        TBLR    POSSM       ; POSITIVE DIRECTION
        LACK    NSM
        TBLR    NEGSM       ; WINDOW FOR SLEW MODE IN
*
```

```
*****************************************************************
*
*   INITIALIZE PAGE 0 DATA
*
*   SYSTEM IS ORIGINALLY INITIALIZED AT A PSEUDO 1200 BPS, TX, RX IDLE LINE
*   MODE, 16 SAMPLES/BAUD TO ACCOMODATE THE START-UP CONDITION OF THE
*   TMS70A2400 MODEM CONTROLLER.
*
*****************************************************************
*
        LDPK    0
        LACK    8
        SACL    IRCNT
        LACK    2
        SACL    STATUS
        LACK    15
        SACL    SCNT
        SACL    XSCNTR
        SACL    RSCNTR
        LACK    1
        SACL    ONE
*
*****************************************************************
*
*   INITIALIZE SERIAL PORT CONTROL REGISTERS
*
*****************************************************************
*
        LACK    DA1
        TBLR    TMP0
        OUT     TMP0,PA0
        LACK    DA2
        TBLR    TMP0
        OUT     TMP0,PA1
        LACK    DA3
        TBLR    TMP0
        OUT     TMP0,PA0
*
*****************************************************************
*
*   INITIALIZE AGC GAIN AND LOCK INDICATOR. (MOD. 5/29)
*
*****************************************************************
*
        LACK    MAXALP
        TBLR    ALPHA           ; SET ALPHA TO ITS MAX POSSIBLE VALUE
*
*****************************************************************
*
*   INITIALIZE STATUS WORD TO SET AFE GAIN STAGE ON
*
*****************************************************************
*
        LACK    080h
        SACL    STWRD
*
*****************************************************************
*
*   INITIALIZE HYSTERESIS COUNTER TO 8000h
*
*****************************************************************
*
        LAC     ONE,15
        SACL    HYST
*
*****************************************************************
*
*   MAIN PROGRAM SEQUENCER
*
*   THE MAIN PROGRAM SEQUENCER PROVIDES THE TIMING FOR THE MAIN PROGRAM LOOP
*   AND CALLS THE VARIOUS SUBROUTINES AT THE APPROPRIATE TIMES. THE MAIN LOOP
*   IS COMPLETED ONCE EVERY BAUD INTERVAL OR EQUIVALENTLY 300 TIMES/SEC FOR
*   FSK.
*
*   THE PCM CODEC HAS A FIXED SAMPLING RATE OF 9.6 KHZ, WHICH MEANS THAT IN
*   THE FSK MODE THE DSP BAUD PERIOD CORRESPONDS TO 32 PCM SAMPLES.
*
*****************************************************************
*
WAIT    IN      TMP0,PA0        ; WAIT FOR FR INTERRUPT FLAG
        LACK    8               ; IF NOT LOOP HERE
        AND     TMP0            ; IF YES TRANSMIT AND RECEIVE FROM PORT 1
        BZ      WAIT            ; AND RESET THE INTERRUPT FLAG
*
SEQU    OUT     XOUT,PA1
        IN      RIN,PA1
        LACK    DA3
        TBLR    TMP0
        OUT     TMP0,PA0
*
*****************************************************************
*
*   EXECUTE TRANSMITTER TASK FOR TIME SLOT AND UPDATE SAMPLE COUNTER.
*
*****************************************************************
*
        LAC     XSCNTR
        SUB     SCNT
        BLZ     SEQU2           ; IF ZERO CALL CCI
*
SEQU01  CALL    CCI             ; CALL COMMAND INTERPRETER
*
*****************************************************************
*
*   EXECUTE THE SAMPLE TASKS
*
*****************************************************************
*
```

```
SEQU2   LACK    030h            ; CHECK FOR ANSWER TONE
        AND     STATUS
*
*************************************************************************
*
*   CHECK IF IDLE: IF BITS 4 & 5 OF STATUS ARE 0 THEN TX IN IDLE MODE =>
*   TRANSMIT UNMODULATED
*
*************************************************************************
*
        BZ      SEQU20
*
        SUB     ONE,4           ; IF 0 => ANSWER TONE
        BZ      SEQU3           ; ANSWER TONE => CALL FSK
*
        LACK    3
        AND     STATUS          ; CHECK BITS 1 AND 0
        BZ      SEQU3
*
        LAC     DTFLAG          ; CHECK FOR DTMF DIAL MODE DUAL TONE
*                               ; TRANSMISSION
        BNZ     SEQU3
        B       SEQU4
*
SEQU3   CALL    FSKTX           ; FSK OR ANSWER TONE OR DTMF
*
        B       SEQU4
*
*************************************************************************
*
*   IN IDLE MODE TRANSMIT A 0
*
*************************************************************************
*
SEQU20  ZAC
        SACL    XOUT
*
SEQU4   LAC     XSCNTR
        SUB     ONE
        BGEZ    SEQU5
        LAC     SCNT
SEQU5   SACL    XSCNTR
*
*************************************************************************
*
*   EXECUTE RECEIVER TASK FOR TIME SLOT AND UPDATE RECEIVER SAMPLE COUNTER.
*   IF RECEIVER IN IDLE MODE RETURN TO WAIT STATE
*
*************************************************************************
*
        LACK    RMMSK
        AND     STATUS
        BNZ     SEQU9
        LAC     RSCNTR
        SUB     ONE
        BGEZ    SEQU22
        LAC     SCNT
        OUT     STWRD,PA5
SEQU22  SACL    RSCNTR
        B       WAIT
*
*************************************************************************
*
*   USE ROUTINE 'RSTSK' TO PERFORM FSK DEMODULATION
*
*************************************************************************
*
SEQU9   CALL    RSTSK
        LACK    3
        AND     STATUS          ; CHECK FSK OPERATION
        BZ      DECRS           ; IF SO, JUST DECREMENT RX SAMPLE COUNTER
*
        LACK    RSEQTB          ; IN HANDSHAKING MODE CALL RTASK
        ADD     RSCNTR          ; SUBROUTINE ONCE PER SAMPLE. ONLY ONE
        TBLR    TMP1            ; NON-TRIVIAL FUNCTION, RTSK10, IS
        LAC     TMP1            ; ACTUALLY CALLED.
        CALA
*
DECRS   LAC     RSCNTR
        SUB     ONE
        BGEZ    SEQU6           ; IF NOT THE END OF BAUD, JUST CONTINUE
        CALL    AGC             ; ELSE DO AGC ONCE PER BAUD.
*
SEQU66  LAC     SCNT            ; RESET BAUD COUNTER
SEQU6   SACL    RSCNTR
*
        B       WAIT
*
*************************************************************************
*
*   SUBROUTINES
*
*************************************************************************
*
FSKTX:  .set    $
        ZALS    TXPHS           ; BRING IN TX ANGLE
        CALL    SINGEN          ; GENERATE TONE AT APPROPRIATE FREQ
```

```
*                                   ; RESULT RETURNED IN TMP3
        LAC     DTFLAG
        BNZ     ISDTMF
*
        LAC     TMP3,13             ; 4S12 FORMAT
        SACH    XOUT
        LT      XOUT
        MPYK    0700h
        PAC
        SACH    XOUT,4
        B       NODTMF
*
ISDTMF: .set    $
        LAC     TMP3,15             ; 2S30 FORMAT - LOWER FREQUENCY
        SACH    TMP5                ; 2S14 FORMAT - LOWER FREQUENCY
*
        ZALS    RXPHS               ; IN DTMF MODE, HIGH FREQ IS HANDLED BY
*                                   ; BY RXPHS
        CALL    SINGEN              ; GENERATE HIGH FREQUENCY TONE
        ZALS    RXPHS               ; INCREMENT HIGH FREQUENCY PHASE ANGLE
        ADDS    RXFRQ               ; BY SECOND TONE FREQUENCY
        SACL    RXPHS               ; STORE AWAY
*
        LAC     TMP3,15
        SACH    TMP3
        LT      TMP5
        MPY     DTMFL
        PAC
        LT      TMP3
        MPY     DTMFH
        APAC
        SACH    XOUT,4
*
NODTMF: .set    $
        ZALS    TXPHS               ; INCREMENT TX ANGLE BY APPROPRIATE 0 OR 1
        ADDS    TXFRQ               ; FREQUENCY AND STORE IN TXPHS
        SACL    TXPHS
*
        NONE    RET
*
        .copy   "CCIDTM.A00"        ; INCLUDES CODE FOR DTMF
*
*******************************************************************************
*
*   CONTROLLER COMMAND INTERPRETER (CCI)
*
*   THE FOLLOWING CODE READS A COMMAND FROM THE TMS70A2400 ON PORT 5 AND
*   INTERPRETS IT ACCORDING TO THE RULES SPECIFIED IN THE CONTROLLER-DSP
*   INTERFACE DOCUMENT. THE 320 READS ONE COMMAND EVERY BAUD PERIOD. THE BAUD
*   RATE IS INITIALLY SET TO 600, AND THE BAUD CLOCK IS DERIVED FROM THE
*   SERIAL PORT FR SIGNAL.
*
*******************************************************************************
*
CCI     IN      XDATA,PA5           ; READ COMMAND
*
        LACK    0F0h
        AND     XDATA               ; MASK OFF 4 LSBS OF COMMAND
        SACL    TMP2
        LACK    090h                ; CHECK FOR DTMF
        SUB     TMP2
        BZ      DTCONT
*
        ZAC                         ; IF NOT,
        SACL    DTFLAG              ; CLEAR DTMF DIAL FLAG
*
CONT    LAC     TMP2,12             ; COMMAND BITS TO ACCH LSBS
        SACH    TMP2
*
*******************************************************************************
*
*   NOW THE ACCUMULATOR VALUE CORRESPONDS TO THE FOLLOWING COMMANDS AND
*   CORRESPONDING SERVICE SUBROUTINES
*
*******************************************************************************
*
*   ACC             COMMAND                 SUBROUTINE
*
*   Fh              PROTOCOL SELECT         PROTO
*   Dh              SET OPERATING MODE      OPER
*
*   9h              DIAL DTMF               DTMF
*   8h              XMIT MODE               XMODE
*   7h              RECV MODE               RMODE
*
*   5h              FSK DATA MODE           FSKSET
*   4h              RESET                   RESET
*
*******************************************************************************
*
*   CHECK FIRST IF RECEIVER IS IN DATA MODE. IN WHICH CASE IGNORE ALL
*   COMMANDS EXCEPT 2Xh, 1Xh AND 00h.
*
*******************************************************************************
*
        LACK    RMMSK               ; REC. MODE MASK
        AND     STATUS
        SUB     ONE,7               ; CHECK IF BITS 7 AND 6 ARE ONES
        SUB     ONE,6
        BNZ     CCI1
*
*******************************************************************************
*
*   REC. IN DATA MODE => IGNORE COMMANDS >2
*
*******************************************************************************
*
        LAC     TMP2
```

```
        SUB     ONE,2
        BLEZ    CCI1            ; IF COMMAND LARGER THAN >2 EXIT COMMAND
        RET                     ; INTERPRETER.
*
*****************************************************************
*
*   CALL THE APPROPRIATE SERVICE SUBROUTINE (REFER TO CMDTBL TABLE).
*
*****************************************************************
*
CCI1    LACK    CMDTBL          ; BASE OF COMMAND TABLE
        ADD     TMP2            ; ADD COMMAND OPCODE
        TBLR    TMP2            ; READ ADDRESSS FROM TABLE
        LAC     TMP2            ; LOAD SUB. ADDR. INTO ACC.
        CALA                    ; CALL SERVICE SUBROUTINE
*
        RET                     ; EXIT COMMAND INTERPRETER.
*
*****************************************************************
*
*   COMMAND INTERPRETER SUBROUTINES
*
*****************************************************************
*
*   PROTOCOL SELECT COMMAND
*
*****************************************************************
*
PROTO:  .set    $
*
*****************************************************************
*
*   EXTRACT TWO LSB'S OF COMMAND
*
*****************************************************************
*
        LACK    03h             ; MASK OFF BITS 2 AND 3 OF COMMAND
        AND     XDATA
*
*****************************************************************
*
*   SET SPEED BITS STATUS REGISTER:
*
*   BITS 1 AND 0 = 00 FOR 300 BPS (FSK)
*                = 01 RESERVED
*                = 10 RESERVED
*                = 11 RESERVED
*
*****************************************************************
*
        SACL    TMP1
        LACK    0FCh
        AND     STATUS          ; ZERO BITS 0 AND 1
        SACL    STATUS
*****************************************************************
*
*   DETERMINE FSK FREQUENCIES AND SET BAUD COUNTER. ALSO SET OTHER FSK
*   SIGNAL PROCESSING PARAMETERS
*
*****************************************************************
*
PROTO2  LACK    31
        SACL    SCNT            ; BAUD COUNTER IS 32
*
        LAC     ONE,2
        AND     XDATA
        SACL    CCITT
*
        LAC     ONE,3
        SACL    FSKFLG
*
*****************************************************************
*
*   ACCUMULATOR NOW CONTAINS THE NUMERAL 8 LOGICAL AND HENCE IDENTIFIES
*   ORIGINATE_/ANSWER MODES
*
*****************************************************************
*
        AND     XDATA           ; ISOLATE ORIGINATE_/ANSWER BIT
        SACL    OAFLAG          ; SET OAFLAG <> 0 IN ANSWER MODE = 0 IN
*                               ; ORIGINATE MODE
        LACK    0Ch             ; MASK TWO LSBS OF COMMAND
        AND     XDATA
        SACL    TMP1
        LACK    FSKTBL          ; ADD BASE OF FSK TABLE
        ADD     TMP1
        SACL    F1ADD           ; 1 FREQUENCY
        ADD     ONE
        SACL    F0ADD           ; 0 FREQUENCY ADDRESS
        ADD     ONE
        TBLR    B1FSK           ; FSK LOWPASS FILTER COEFFICIENT
        ADD     ONE
        TBLR    GAIN            ; FSK MODE GAIN
        SUB     ONE
        SACL    TMP1            ; TMP1 NOW POINTS TO B1FSK
*
*****************************************************************
*
*   SET FSK TIMING RECOVERY PARAMETERS.
*
*****************************************************************
*
        LACK    0Ch             ; ACC = 12
        SACL    TRANS           ; TRANS = 12
*
*****************************************************************
*
*   SET FSK RECEIVE FILTER COEFFICIENTS AND SLICER DEAD ZONE
```

```
*
*****************************************************************
*
        LACK    FSKA1           ; LOAD ADDRESS OF A1 COEFFICIENT
        TBLR    A1FSK
        ADD     ONE
        TBLR    A2FSK           ; A2FSK CONTAINS A2 COEFFICIENT
        LACK    ZONE            ; ACC = ADDRESS OF DEAD ZONE
        TBLR    DZONE           ; DEAD ZONE OF WINDOW COMPARATOR FOR FSK
*                               ; DECISION
*
*****************************************************************
*
*   NOW CHECK FOR ANALOG LOOPBACK. IF IT IS IN ANALOG LOOPBACK MODE, THEN WE
*   NEED TO MODIFY THE RECEIVER PARAMETERS TO CONFIGURE RECEIVER IN THE SAME
*   BAND AS THE TRANSMITTER.
*
*****************************************************************
*
        LACK    ANMSK
        AND     STATUS          ; STATUS BIT IS 1 FOR ANALOG
        BZ      PROTO5          ; AND ZERO OTHERWISE
*
*****************************************************************
*
*   CHECK IF ANSWER OR ORIGINATE. IF IN ORIGINATE MODE THE THE RECEIVER MUST
*   ALSO BE CONFIGURED FOR LOW BAND. IN WHICH CASE THE PARAMETERS REQUIRED
*   ARE AT ADDRESS TMP1+8 IF IN ANSWER MODE, THE RECEIVER MUST BE PUT IN HIGH
*   BAND AND THE PARAMETERS ARE AT ADDRESS TMP1-8.(PLEASE REFER TO THE
*   FSKTBL)
*
*****************************************************************
*
        LACK    08h             ; CHECK ORIG/ANS BIT 3 OF XDATA
        AND     XDATA
        BZ      PROTO6
*
        LACK    0
        SACL    OAFLAG
        LAC     TMP1            ; IN ANSWER MODE SUBTRACT 8 FROM TMP1
        SUB     ONE,3
        B       PROTO7
*
PROTO6  LACK    8               ; WE'RE IN ORIGIN MODE,
        SACL    OAFLAG          ; SET OAFLAG <> 0 FOR LOWBAND
        LAC     TMP1            ; IN ORIGINATE MODE ADD 8
        ADD     ONE,3           ; TO TMP1 (THIS ONLY HAPPENS IN ANALOOP)
PROTO7  TBLR    B1FSK           ; READ B1 COEFF
        ADD     ONE
        TBLR    GAIN            ; READ GAIN
PROTO5  RET
*
*****************************************************************
*
*   SELECT GUARD TONE
*
*   GUARD   RET
*
*****************************************************************
*
*   SET OPERATING MODE
*
*****************************************************************
*
OPER    LACK    LMMSK           ; ZERO ANALOG LOOPBACK
        AND     STATUS          ; LOC. DIG. LOOPBACK BITS
*
*****************************************************************
*
*   CHECK OPERATING MODE
*
*****************************************************************
*
        LACK    03h             ; MASK OFF BITS 2 AND 3 OF COMMAND
        AND     XDATA
        BZ      OPER1           ; IF ZERO => LINE MODE (RET)
        SUB     ONE
        BZ      ANLB            ; IF ONE => ANALOG LOOPBACK
*
OPER1   RET
*
ANLB    LACK    ANMSK           ; SET ANALOG LOOPBACK
        OR      STATUS          ; STATUS BIT
        SACL    STATUS
        RET
*
*****************************************************************
*
*   DIAL DTMF
*
*   DTMF SET-UP ROUTINE : LOOKUP THE LOW AND HIGH FREQUENCIES CORRESPONDING
*   TO EVERY DIGIT AND PLACE IN TXFRQ AND RXFRQ RESPECTIVELY.
*
*****************************************************************
*
DTMF    LACK    15              ; MASK FOR ISOLATING THE DIGIT
        AND     XDATA
        SACL    TMP0            ; STORE AWAY TEMPORARILY
        LACK    TONTBL          ; BRING IN BASE ADDRESS OF TONE TABLE
        ADD     TMP0,2          ; LEFT SHIFT IS REQUIRED AS THERE ARE FOUR
*                               ;   ENTRIES PER DIGIT
        TBLR    TXFRQ           ; READ LOW FREQ INTO TXFRQ
        ADD     ONE             ; INCREMENT POINTER TOWARDS HIGH FREQ
        TBLR    RXFRQ           ; READ HIGH FREQ INTO RXFRQ
        ADD     ONE             ; READ IN LO FRQ GAIN
        TBLR    DTMFL
        ADD     ONE             ; READ IN HI FRQ GAIN
        TBLR    DTMFH
```

```
*
        LAC     ONE             ; SET DTMF DIAL MODE FLAG
        SACL    DTFLAG
*
        RET
*
*****************************************************************
*
*   TRANSMITTER MODE SELECT
*
*****************************************************************
*
XMODE   LACK    03h             ; MASK OFF BITS 2 AND 3
        AND     XDATA
        SACL    TMP1            ; SAVE MODE BITS TO SET STATUS
        SUB     ONE             ; CHECK IF TRANSMIT ANSWER TONE
        BNZ     XMODE1
*
*****************************************************************
*
*   FOR ANSWER TONE, LOAD TONE FREQ IN TXFRQ LOCATION AND 0001h IN XBITS
*   LOCATION IN RAM, AND CALL FSK SUBROUTINE WHICH WILL TRANSMIT TXFRQ (I.E.
*   (THE ANSWER TONE).
*
*****************************************************************
*
        LACK    01h
        SACL    XBITS
        LACK    F21             ; ADDRESS OF 2100 ANS. TONE
        TBLR    TXFRQ
        LACK    0Ch             ; DETERMINE ANS. TONE FREQ.
        AND     XDATA
        BZ      XMODE2          ; IF 0 => 2100 IS RIGHT
*
        LACK    F22             ; OTHERWISE LOAD TXFREQ REG WITH 2225
        TBLR    TXFRQ           ; ANSWER TONE PHASE INC.
        B       XMODE2
*
*****************************************************************
*
*   CHECK FOR REMOTE DIGITAL LOOPBACK
*
*****************************************************************
*
XMODE1  SUB     ONE,1           ; SUBTRACT 2 MORE FROM TMP1
        BNZ     XMODE2          ; IF ZERO => REM. DIG. LOOPBACK
*
*****************************************************************
*
*   HANDLE RDL: (ASSUME THAT THE RECEIVER IS ENABLED)
*
*   PLACE RECEIVED QUADBITS IN XBITS. CHECK FOR 1200 BPS OPERATION. IF SO
*   FORCE A 1 INTO LSB OF XBITS. SET BITS 5 & 4 IN STATUS = 10. PLACE
*   RECEIVED QUADBITS IN XBITS
*
*****************************************************************
*
        LACK    0Fh             ; MASK OFF QUADBITS
        AND     STWRD
        SACL    XBITS           ; STORE AWAY
*
*****************************************************************
*
*   CHECK FOR 1200 BPS OPERATION
*
*****************************************************************
*
        LACK    0FCh            ; MASK FOR SPEED BITS
        AND     STATUS          ; SPEED BITS
        SUB     ONE,1           ; CHECK FOR 10
        BNZ     XMODE2          ; IF NON-ZERO, JUST CONTINUE
*
        LAC     XBITS           ; ADD A 1 TO LSB FOR 1200 BPS OPERATION
        ADD     ONE
*
*****************************************************************
*
*   SET THE TRANSMITTER MODE BITS IN STATUS REGISTER.
*
*****************************************************************
*
XMODE2  LACK    NTXMSK          ; NEGATION OF TRANSMISSION BITS MASK
        AND     STATUS          ; ZERO THE TX STATUS BITS
        ADD     TMP1,TXSH       ; ADD TX STATUS BITS IN RIGHT POS.
        SACL    STATUS
        RET
*
*****************************************************************
*
*   RECEIVER MODE SELECT
*
*   SET THE RECEIVER STATUS BITS (BITS 6 AND 7) OF STATUS REGISTER TO:
*
*       00 IF RECEIVER IS IDLE
*       01 FOR CALL PROGRESS MONITORING
*       10 FOR DATA MODE
*
*****************************************************************
*
RMODE   LACK    03h             ; MASK OFF BITS 2 AND 3
        AND     XDATA
        SACL    TMP1
        LACK    NRCMSK          ; NEG. OF REC. BITS MASK
        AND     STATUS          ; ZERO REC. STATUS BITS
        ADD     TMP1,RCSH       ; ADD REC STATUS IN RIGHT POS.
        SACL    STATUS
RMODE1  RET
*
```

```
*****************************************************************
*
*   FSK DATA MODE
*
*   SET UP FSK TRANSMIT FREQ ACCORDING TO THE TX DATA
*
*****************************************************************
*
FSKSET  LACK    8               ; CHECK THE TRANSMITTED BIT
        AND     XDATA
        BZ      DATA0           ; IF ZERO, DATA MUST BE 0
        LAC     F1ADD           ; POINT ACC TO 1 FREQ
        TBLR    TXFRQ           ; SET TX FREQ TO APPROPRIATE 1 FREQ
        RET
*
DATA0   LAC     F0ADD           ; POINT ACC TO 0 FREQ
        TBLR    TXFRQ           ; SET TX FREQ TO APPROPRIATE 0 FREQ
*
NOFSK   RET
*
*****************************************************************
*
*   RESET AND EQUALIZER ENABLE ROUTINES
*
*****************************************************************
*
RESET   LACK    081h
        SACL    STWRD
        OUT     STWRD,PA5
*
        B       START
*
*****************************************************************
*
*   END CONTROLLER COMMAND INTERPRETER SUBROUTINES
*
*****************************************************************
```

```
*
        .copy   "SINGEN.A00"
*
*****************************************************************
*
*   SUBROUTINE : SINGEN
*
*   PURPOSE  : SINE GENERATION
*
*   TASK : GIVEN A COSINE TABLE WITH 257 VALUES AND START ADDRESS COSOFF, AND
*   GIVEN AN ANGLE INDEX IN THE ACCUMULATOR, DETERMINE THE SINE OF THE ANGLE.
*
*   ENTRY CONDITION : THE ANGLE INDEX MUST BE IN THE LOWER ACCUMULATOR.
*
*   EXIT CONDITION : THE SINE OF THE ANGLE IS RETURNED IN TEMPORARY LOCATION
*                    TMP3.
*
*   DESCRIPTION : THE COSINE LOOKUP TABLE CONTAINS 257 VALUES WITH;
*
*        COS[0] = 1.0 AND COS[256] = -1.0
*
*   HENCE ANGLE INDEX 0 MAPS TO  ANGLE 0 AND ANGLE INDEX 256 MAPS TO PI. THE
*   SINE VALUE IS GENERATED BY SUBTRACTING FROM THE ANGLE INDEX THE INDEX
*   CORRESPONDING TO PI/2, TAKING THE ABSOLUTE VALUE, AND HENCE FORMING AN
*   ADDRESS INTO THE LOOKUP TABLE.
*
*   NO OF CYCLES: 17
*
*   NO OF STACK LEVELS USED: 1
*
*   THE ANGLE INDEX IS THE LOWER ACCUMULATOR
*
*   ANGLE INDEX HAS S15.0 FORMAT. MUST SUBTRACT PI/2 VALUE WHICH LAYS AT THE
*   MIDDLE OF THE TABLE AND HAS SS14.0 FORMAT AS VIEWED IN S15.0 FORMAT
*
*****************************************************************
*
SINGEN  SUB     ONE,14          ; SUBTRACT INDEX OF PI/2
        SACL    TMP3            ; PUT AWAY TEMPORARILY
        ZALH    TMP3            ; PREPARE FOR ABSOLUTE VALUE
        ABS                     ; TAKE ABSOLUTE VALUE
        SACH    TMP3            ; PUT AWAY BEFORE RIGHT SHIFT
*
*****************************************************************
*
*   THE VALUE STORED IN TMP3 HAS S15.0 FORMAT -- ALBEIT A POSITIVE NUMBER
*
*   A LEFT SHIFT OF 9 BITS CORRESPONDS TO 8S24.0 FORMAT AND SAVING THE HIGH
*   ACCUMULATOR HAS A 8S8.0 FORMAT
*
*****************************************************************
*
        LAC     TMP3,9          ; ISOLATE 8 MSB'S IN HIGH ACC
        SACH    TMP3            ; PUT AWAY THE 8 MSB'S TEMPORARILY
```

```
*
*************************************************************************
*
*    THE NEXT THREE INSTRUCTIONS ELIMINATE ANY SIGN EXTENSION BITS THAT MIGHT
*    HAVE PROPAGATED
*
*************************************************************************
*
        LAC     TMP3
        ABS
        SACL    TMP3
*
*************************************************************************
*
*    FORM THE FINAL LOOK-UP ADDRESS
*
*    BRING IN THE ADDRESS OFFSET.
*
*    THE BASE ADDRESS IS IN 8S8.0 FORMAT, WHILE THE INDEX IS ALSO IN 8S8.0
*    FORMAT.
*
*************************************************************************
*
        LACK    COSOFF
        ADD     TMP3            ; FORM FINAL LOOK-UP ADDRESS
        TBLR    TMP3            ; READ SINE VALUE INTO TMP3
*
        RET
*
*************************************************************************
*
*    FSK DEMODULATION FILES
*
*************************************************************************
*
        .copy   "RSTSKF.A00"
*
*************************************************************************
*
*    DATE: 5-29-86
*
*    SUBROUTINE: RSTSK
*
*    INCLUDES FSK RECEIVER/TIMING RECOVERY
*
*    PURPOSE: RECEIVER PER SAMPLE TASK
*
*    TASK: THIS SUBROUTINE COMBINES SMALLER MODULES TO PERFORM THE SIGNAL
*          PROCESSING FUNCTIONS THAT ARE REQUIRED ON A PER SAMPLE BASIS
*          (9600 Hz).
*
*    ENTRY CONDITION: THE RECEIVED S/M SAMPLE IS IN RAM LOCATION RIN.
*
*************************************************************************
*
RSTSK:  .set    $
        LAC     RIN             ; INPUT 14-BIT S/M SAMPLE
*       SACL    TMP1            ; ARL INTO TMP1.
*
*************************************************************************
*
*    USE HIGH PASS FILTER. MAKE SURE INCOMING SAMPLE HAS NO SHIFT.
*
*************************************************************************
*
*       LAC     TMP1,0          ; ARL
*
*************************************************************************
*
*    HIGH PASS FILTER THE INPUT
*
*************************************************************************
*
MLHP:   .set    $
        LDPK    1
        SACL    X1
        ZALS    STLSB
        ADDH    ST
        SUB     ST,TAU
        ADDH    X1
        SUB     X1,TAU-1
        SUBH    X2
        ADD     X2,TAU-1
        SACL    STLSB
        SACH    ST
        DMOV    X1
        LDPK    0
        SACH    TMP1
*
*************************************************************************
*
*    MLHP LEAVES THE SAMPLE IN TMP1. MULTIPLY IT BY AGC GAIN ALPHA. THE OUTPUT
*    FORMAT IS S4.11 REQUIRING SOME MANIPULATIONS
*
*************************************************************************
*
        LT      TMP1            ; MULTIPLY BY AGC WORD
        MPY     ALPHA
        PAC
*
*************************************************************************
*
*    SHIFT ACCUMULATOR EIGHT 4 TIMES BEFORE STORING
*
*************************************************************************
*
        SACL    TMP0
        SACH    TMP1
```

```
        LAC     TMP0,8
        SACH    TMP0
        LAC     ONE,8           ; MASK OFF ANY SIGN EXTENSION
        SUB     ONE             ; 00FF -> ACC
        AND     TMP0
        SACL    TMP0
        LAC     TMP1,8
        ADD     TMP0
        SACL    TMP1
*
*****************************************************************
*
*   IN BOTH TMP1 AND TMP2 UPDATE THE SIGNAL POWER ESTIMATE AVESQR
*
*   AVESQR = AVESQR + TMP1^2
*
*   AVESQR IS ZEROED BY THE AGC ROUTINE ONCE PER BAUD.
*
*****************************************************************
*
        LAC     TMP1,15
        SACH    TMP0            ; TMP0 IN S5.10
        LT      TMP0
        MPY     TMP1
        PAC
        ADDH    AVESQR          ; AVESQR IN S10.5
        SACH    AVESQR
*
*****************************************************************
*
*   INCREASE SIGNAL ENERGY BY A FACTOR OF 4 FOR COMPATIBILITY WITH THE REST
*   OF THE RECEIVER
*
*****************************************************************
*
        LAC     TMP1,1          ; MULTIPLY TIMES 2
        SACL    TMP1
        SACL    TMP2
*
*****************************************************************
*
*   CHECK FOR FSK OPERATION
*
*       LACK    3
*       AND     STATUS
*
*       LAC     TMP1,0
*       SACL    AGCOUT          ; PLACE RECEIVE SIGNAL IN AGCOUT
*       CALL    RXFSK           ; CALL FSK RECEIVER/TIMING RECOVERY
*       RET
        .copy   CCITT.A00
*
*****************************************************************
*
*   CHECK FOR FSK OPERATION
*
*****************************************************************
*
        LACK    3
        AND     STATUS
        LT      TMP1            ; ASSUME V.21
        MPYK    013h            ; MULTIPLY BY 2.5
        LAC     CCITT
        BNZ     V21ORG
        MPYK    018h            ; MULTIPLY BY 3.0
V21ORG  PAC
*
        SACL    TMP0
        SACH    TMP1
        LAC     TMP0,13
        SACH    TMP0
        LAC     ONE,13
        SUB     ONE
        AND     TMP0
        SACL    TMP0
        LAC     TMP1,13
        ADD     TMP0
        SACL    AGCOUT
        CALL    RXFSK
        RET
        .copy   "FDEM20.A00"
*
*****************************************************************
*
*   DEMODULATOR SECTION
*
*   THIS DESIGN IMPLEMENTS A DELAY OF 5(PI)/2
*
*   AGC
*
*   FSK DEMODULATOR
*
*   MEMORY CONFIGURATION: (CONSECUTIVE ADDRESSES)
*       AGCOUT
*       PDEL0
*       PDEL1
*       PDEL2
*       LPDEL0
*       LPDEL1
*       LPDEL2
*
*****************************************************************
*
RXFSK   SOVM                    ; SET OVERFLOW MODE
*
*****************************************************************
*
*   TAKE PRODUCT FOR PRODUCT DEMOD SCHEME. ASSUME ANSWER MODE (2 SAMPLES)
```

```
*
*************************************************************************
*
        LT      AGCOUT          ; LOAD T WITH AGC STAGE OUTPUT
        MPY     PDEL2           ; TAKE PRODUCT WITH DELAY LINE OUTPUT
*
*************************************************************************
*
*   ORIGINATE/ANSWER FLAG = 0 WHEN TX ORIGINATES THE CALL HENCE RX RECEIVES
*   IN THE HIGH BAND
*
*************************************************************************
*
        LAC     OAFLAG          ; CHECK FOR HIGH BAND/LOW BAND
        BNZ     ANSWER          ; LOW BAND, DONE.
        MPY     PDEL1           ; HIGH BAND, USE ONE LESS SAMPLE DELAY
ANSWER  PAC                     ; ACCUMULATE THE PRODUCT OUTPUT
        SACH    PROD,1          ; STORE IN PROD (1,15)
*
*************************************************************************
*
*   PRODUCT DEMOD DELAY LINE                    |---------------> FOR HIGH BAND
*               -------------------     -------|     -------
*    AGCOUT--->| 1 ZERO FIR FILTER |-->| Z**-1 |-->| Z**-1 |---> FOR LOW band
*               -------------------     -------     -------
*
*   PRODUCT = AGCOUT * OUTPUT OF DELAY LINE
*   PRODUCT IS LOWPASS FILTERED AND IT'S SIGN INDICATES 0 OR 1 DATA...
*
*************************************************************************
*
*   ONE ZERO FILTER TO MOP-UP PHASE SHIFT OF FLAT DELAY LINE TO BE PI/2 IN
*   PRODUCT FSK DEMODULATOR:
*
*   COEFFICIENTS FOR ZERO:  (B1)
*
*           BELL 103:       GAIN 103:

*
*   1170 HZ  0.07175        0.57731
*   2125 HZ  0.31678        1.0
*
*           V.21 :          GAIN V21:
*
*   1080 HZ  0.32796        0.69753
*   1750 HZ  0.68889        1.0
*
*   START UPDATING THE DELAY LINE
*
*************************************************************************
*
        DMOV    PDEL1           ; START SHIFTING DATA IN DELAY LINE
*
*************************************************************************
*
*   GENERATE THE EXACT DELAY REQUIRED FOR RECEIVER.
*
*************************************************************************
*
        LAC     AGCOUT,15       ; PREPARE FOR FIR ACCUMULATION
        LT      PDEL0           ; STATE OF ONE ZERO FIR FILTER
        MPY     B1FSK           ; MULT BY COEFFICIENT
        APAC                    ; OUTPUT OF ONE ZERO FIR FILTER
        BV      RSFLAG          ; RESET OVERFLOW FLAG
*
RSFLAG  .set    $
        SACH    PDEL1,1         ; STORE AT NEXT STAGE OF DELAY FILTER
        DMOV    AGCOUT          ; SHIFT SAMPLE IN FILTER
*
*************************************************************************
*
*   PREPARE TO LOW PASS FILTER THE PRODUCT.
*
*************************************************************************
*
        ZAC                     ; 2ND ORDER DIRECT FORM LP FILTER
        LT      PROD            ; 500 HZ, 0.25 DB RIPPLE
        MPY     GAIN            ; COEFFICIENTS:
        ADD     LPDEL1,15       ;   A1 = 1.392 = 1.0 + 0.392
        LTA     LPDEL1          ;   A2 = -.562
        MPY     A1FSK           ;   B0 = 1
        LTA     LPDEL2          ;   B1 = 2
        MPY     A2FSK           ;   B2 = 1
        APAC                    ;   GAIN = 1 HIGH BAND
        SACH    LPDEL0,1        ;   GAIN = 0.5 LOW BAND
*
        LAC     LPDEL0,14
        ADD     LPDEL1,15       ; B1 * LPDEL1
        ADD     LPDEL2,14       ; B2 * LPDEL2
        BV      OVRFLW
        B       NOVRFL          ; CHECK FOR OVERFLOW
OVRFLW  .set    $
        ZAC                     ; CLEAR DELAY LINE
        SACL    LPDEL1
        SACL    LPDEL2
NOVRFL  .set    $
        SACH    LPFOUT,1        ; STORE FILTERED
        OUT     LPFOUT,PA6
*
        DMOV    LPDEL1
        DMOV    LPDEL0
```

```
*****************************************************************
*
*    DECISION: SLICER WITH DEAD ZONE IN MIDDLE (WINDOW COMPARATOR)
*
*****************************************************************
*
        ABS
        SUB     DZONE
        BLZ     DONE
*
        LAC     CCITT           ; CHECK FOR BELL VS CCITT
        BNZ     ITSV21
        LAC     LPFOUT
        BLEZ    HIGH
        LACK    0               ; SET BAUD DATA = 0 (0xxx)
        SACL    FSKDAT
        B       DONE
HIGH    LACK    8               ; SET BAUD DATA = 1 (1xxx)
        SACL    FSKDAT
        B       DONE
*
*****************************************************************
*
*    FOR V.21, NEGATIVE SIGNAL MEANS THAT WE HAVE RECEIVED A 0 AND A POSITIVE
*    SIGNAL MEANS THAT WE HAVE A 1.
*
*****************************************************************
*
ITSV21  .set    $
        LAC     LPFOUT
        BLEZ    LOW
        LACK    8               ; SET BAUD_DATA TO 1
        SACL    FSKDAT
        B       DONE
LOW     LACK    0               ; SET BAUD_DATA TO 0
        SACL    FSKDAT
*
*****************************************************************
*
*    THE FOLLOWING IMPLEMENTS FSK TIMING RECOVERY
*    ADDED BY K.H 10-3-86
*
*****************************************************************
*
DONE    LAC     BAUDCK          ; INCREMENT BAUD CLOCK
        ADD     ONE
        SACL    BAUDCK
        LAC     FSKDAT          ; CHECK FOR TRANSITIONS
        SUB     BDATA           ; COMPARE TO PREVIOUS DATA
        BNZ     TRAN            ; IF NOT THE SAME, HANDLE TRANSITION
        LAC     BAUDCK          ; ELSE CHECK FOR END OF BAUD
        SUB     ONE,5           ; PERIOD = 32 AND END OF BAUD
        BLEZ    ERFSK           ; IF NOT END OF BAUD, JUST RETURN
        B       CLEAR           ; ELSE OUTPUT PREVIOUS DATA AND CLEAR
*                               ; COUNTERS
*
TRAN    LAC     COUNTR          ; LOAD TRANSITION COUNTER
        ADD     ONE             ; INCREMENT
        SACL    COUNTR
        SUB     TRANS           ; CHECK FOR 15 TRANSITIONS
        BNZ     ERFSK           ; IF NOT ENOUGH TRANSITIONS, RETURN
        LAC     BDATA           ; ELSE INVERT PREVIOUS DATA
        SUB     ONE,3
        ABS
        SACL    BDATA           ; STORE BAUD_DATA
        LACK    0F0h
        AND     STWRD           ; GET RID OF WHAREVER WAS IN LSBs
        ADD     BDATA           ; FORCE BAUD_DATE INTO BIT 3
        SACL    STWRD
*
CLEAR   OUT     STWRD,PA5       ; OUTPUT TO MICROCONTROLLER
        ZAC
        SACL    BAUDCK          ; CLEAR BAUD CLOCK
        SACL    COUNTR          ; CLEAR TRANSITION COUNTER
ERFSK   RET                     ; END OF FSK DEMOD
*
        .copy   "RTASKC.A00"
*
*****************************************************************
*
*    RTASKS.A00
*
*    RTASK's ALLOW LARGE TIME INTENSIVE TASKS TO BE SPLIT BETWEEN THE SAMPLING
*    PERIODS OF A BAUD. THIS IS A PRACTICAL MEANS OF IMPLEMENTING COMPLEX
*    FUNCTIONS ON THE DSP, WITH THE TIME RESTRICTION PLACED BY THE 9.6 KHZ
*    SAMPLING TRACE
*
*****************************************************************
*
*    RTSK10
*
*****************************************************************
*
RTSK10  LACK    0F0h
*
*****************************************************************
*
*    OUTPUT THE STATUS WORD TO THE CONTROLLER BIT 6 (IN-BAND ENERGY DETECT)
*    INDICATES DIALTONE, BUSY SIGNAL, ETC.
*
*****************************************************************
*
        LACK    0F0h
        AND     STWRD
        ADD     ONE
        SACL    STWRD
        OUT     STWRD,PA5
```

```
        RET
*
*****************************************************************
*
*   AGC FILES
*
*****************************************************************
*
        .copy    "AGC.A00"
*
*****************************************************************
*
*   AGC.ASM
*
*   FRONT END AGC FUNCTION.
*
*   THIS AGC WAS REDESIGNED TO INCORPORATE THE FREEZE OF EQUALIZATION 5/29/87
*
*   THE AVERAGE SIGNAL SQUARED IS COMPUTED BY THE MAIN PROGRAM AND STORED IN
*   AVESQR, WHICH IS CLEARED BY THIS ROUTINE AFTER AVESQR IS USED. THE
*   ROUTINE USES A WINDOW WHOSE WIDTH DEPENDS ON THE MODULATION (1200, 2400)
*   AND AN ERROR WEIGHTING WHICH ALSO DEPENDS ON THAT RATE. WE FIRST SET
*   THOSE VALUES:
*
*****************************************************************
*
AGC     SOVM
        LAC     GN              ; CHECK FOR AFE SWITCHING
        BNZ     SWITCH
*
*****************************************************************
*
*   CHECK IF 2400 AND CHANGE THOSE VALUES
*
*****************************************************************
*
        LACK    3
        AND     STATUS          ; IF STATUS BITS 0 AND 1 > 2 => 2400
        SUB     ONE,1
        BLEZ    AGC0            ; IF <= 2 , DO NOT MODIFY TMP0 AND TMP1
*
*****************************************************************
*
*   FOR 2400 , 2 -> TMP0 AND 1320 -> TMP1
*
*****************************************************************
*
        LACK    2               ; IT IS 2400
        SACL    TMP0
        LT      ONE
        MPYK    1320
        PAC
        SACL    TMP1
        B       AGC1
*
AGC0    LACK    1               ; IT IS 1200
        SACL    TMP0            ; WEIGHTING FACTOR -> TMP0
        LT      ONE
        MPYK    950
        PAC
        SACL    TMP1            ; WINDOW -> TMP1
*
*****************************************************************
*
*   NOW SUBTRACT REFERENCE FROM BAUD ENERGY TO GET ERROR. THE BAUD ENERGY IS
*   IN S10.5 FORMAT. THE AGC MAINTAINS THAT LEVEL AT 2.8&*16 = 46.7  (H'5B6
*   IN S10.5). THE AGCREF IS THEREFORE H'5B6
*
*****************************************************************
*
AGC1    LAC     AVESQR
        BGEZ    CONT1           ; FOR NEGATIVE ENERGY SET TO MAX POSITIVE
        LAC     ONE,15          ; ENERGY LEVEL - FORCED SLEW MODE
        SUB     ONE
CONT1   LT      ONE
        MPYK    AGCREF          ; AGCREF = H'5B6
        SPAC                    ; AVESQR - AGCREF -> ACC
*
*****************************************************************
*
*   COMPARE THE ERROR TO WINDOW (TMP1).
*
*       IF ERROR > WINDOW => ERROR - WINDOW -> ERROR
*       IF -WINDOW < ERROR < WINDOW => 0 -> ERROR
*       IF ERROR < -WINDOW  => TMP0 * (ERROR + WINDOW) -> ERROR
*
*   IF THE AVERAGE BAUD ENERGY IS A, THE PEAK BAUD ENERGY FOR QAM SIGNALS IS
*   1.8 A AND THE MINIMUM IS 0.2 A. THE WINDOW IS THEREFORE CHOSEN TO BE 0.8
*   A IN EITHER DIRECTION. WITH AGCREF = H'5B6, THE WINDOW IS H'492. FOR DPSK
*   SIGNALS, THE VARIATIONS IN BAUD ENERGY ARE ENTIRELY DUE TO ISI AND
*   DISTORTION AND THEREFORE THE WINDOW IS MUCH SMALLER (H'A). FIRST CHECK IF
*   ERROR > WINDOW
*
*****************************************************************
*
        SUB     TMP1
        SACL    TMP3            ; ERROR - WINDOW -> TMP3
        BGEZ    AGC2
*
```

```
*****************************************************************
*
*   ERROR < WINDOW => CHECK IF ERROR > -WINDOW. IN WHICH CASE, ZERO THE
*   ERROR. FIRST, ZERO THE ERROR (I.E. ASSUME ERROR > -WINDOW) AND MODIFY IF
*   WRONG ASSUMPTION.
*
*****************************************************************
*
        LARK    AR1,0
        SAR     AR1,TMP3        ; ASSUME ERROR > -WINDOW
*
*****************************************************************
*
*   CHECK ASSUMPTION
*
*****************************************************************
*
        ADD     TMP1,1          ; ERROR + WINDOW -> ACC
        BGEZ    AGC2            ; ASSUMPTION IS RIGHT
*
*****************************************************************
*
*   ERROR < -WINDOW => TMP0*(ERROR+WINDOW) -> TMP3
*
*****************************************************************
*
        SACL    TMP3
        LT      TMP3
        MPY     TMP0
        PAC
        SACL    TMP3
*
*****************************************************************
*
*   AT THIS POINT, THE WEIGHTED WINDOWED ERROR IS CONTAINED IN TMP3. WE
*   CONSIDER IT AN S.15 NUMBER AND USE IT TO UPDATE THE AGC GAIN ALPHA.
*   FIRST, WE DETERMINE WHETHER TO SLEW OR NOT. IF THE ERROR IS LARGER THAN
*   1EA6h OR SMALLER THAN F5E7h, GO INTO SLEWING MODE BY SETTING ERROR TO
*   7FFFh or 8000h RESPECTIVELY. OTHERWISE, LEAVE IT UNCHANGED.
*
*   ALSO SET STAT2[7] APPROPRIATELY TO FREEZE THE UPDATE OF THE EQUALIZER
*
*       STAT2[7]=1 UPDATE EQUALIZER
*
*       STAT2[7]=0 FREEZE EQUALIZER
*
*****************************************************************
*
*AGC2   LAC     ONE,7
*       OR      STAT2           ; ASSUME EQUALIZER UPDATE
*       SACL    STAT2
*
AGC2    LAC     TMP3
        LDPK    1
        SUB     POSSM
        BLEZ    AGC3            ; DO NOT SLEW
*
        LDPK    0               ; ENTER SLEW MODE
        LAC     ONE,15
        SUB     ONE
        SACL    TMP3            ; TMP3 <- 7FFF
*
*       LACK    7Fh
*       AND     STAT2
*       SACL    STAT2           ; FREEZE EQUALIZER UPDATE
        B       AGC4
*
AGC3    ADD     POSSM           ; ACC <- TMP3
        ADD     NEGSM
        LDPK    0
        BGEZ    AGC4            ; DO NOT SLEW
*
        LAC     ONE,15          ; ENTER SLEW MODE
        ADD     ONE
        SACL    TMP3            ; TMP3 <- 8000
*
*       LACK    7Fh
*       AND     STAT2
*       SACL    STAT2           ; FREEZE EQUALIZER
*
*****************************************************************
*
*   THE FOLLOWING LINES UPDATE THE GAIN ALPHA USING AN EXPODENTIAL INTEGRATOR
*
*       ALPHA = ALPHA(1-K*ERROR)   (ERROR = TMP3)
*
*   WHERE ALPHA IS OF FORMAT S7.8, ERROR IS S0.15, AND K = 0.5.
*
*   ALPHA * ERROR: S7.8 * S.15 = S7.24.
*
*   BY KEEPING ACCH WITHOUT LEFT SHIFT THE MULTIPLICATION BY K IS
*   ACCOMPLISHED.
*
*   ALPHA IS UPPERBOUNDED TO 35.73 IN S7.8
*
*****************************************************************
*
AGC4    LACK    MAXALP
        TBLR    TMP0
        ZALH    ALPHA
        LT      TMP3            ; ERROR -> T
        MPY     ALPHA
        SPAC                    ; ALPHA (1 - 0.5*ERROR) -> ACC
        SACH    ALPHA
*
*****************************************************************
*
*   CHECK IF ALPHA > MAX ALPHA
```

```
*
*****************************************************************
*
        SUBH    TMP0
        BLEZ    AGC5
*
        LAC     TMP0
        SACL    ALPHA
*
*****************************************************************
*
*   ZERO BAUD ENERGY REGISTER
*
*****************************************************************
*
AGC5:   .set    $
*ARL    OUT     ALPHA,PA2       ; ARL DIAGNOSTIC
        ZAC
        SACL    AVESQR
*
*****************************************************************
*
*   ENERGY DETECT LOOP
*
*   START BY READING IN HYATERESIS COUNTER INCREMENT CONSTANT
*
*****************************************************************
*
       LACK     HYSINC
       TBLR     TMP5
*
*****************************************************************
*
*   CHECK IF AFE GAIN IS ON OR OFF
*
*****************************************************************
*
EDT     LACK    080h
        AND     STWRD
        BZ      EDT1
*
*****************************************************************
*
*   AFE GAIN IS ON, CHECK IF ENERGY DETECT IS ON (STWRD[6] = 1)
*
*****************************************************************
*
        LAC     ONE,6
        AND     STWRD
        BZ      EDT01           ; IF ZERO => ENERGY IS NOT DETECTED =>
*                               ; CHECK IF LEVEL IS LARGER THAN -43.5 DBM
*                               ; If STWRD[6] IS ONE, CHECK IF LEVEL LESS
*                               ; THAN -48 DBM
       LACK     THRES1
       TBLR     TMP0
       LAC      TMP0
       SUB      ALPHA
       BLZ      EDT2            ; IF < 0 THEN NO ENERGY DETECT
*
*****************************************************************
*
*   CHECK IF AFE GAIN STAGE SHOULD BE BYPASSED
*
*****************************************************************
*
        LACK    THRES3
        TBLR    TMP0
        LAC     TMP0
        SUB     ALPHA
        BLZ     EDT3
*
*****************************************************************
*
*   IS AFE GAIN ON ?
*
*****************************************************************
*
        LACK    080h
        AND     STWRD
        BZ      EDT3            ; IF GAIN IS OFF, EXIT
*
*****************************************************************
*
*   BYPASS AFE GAIN
*
*****************************************************************
*
        LACK    07Fh
        AND     STWRD
        SACL    STWRD
        LACK    04h
        SACL    GN
        RET
*
```

```
*************************************************************************
*
*    DECREMENT HYSTERESIS COUNTER
*
*************************************************************************
*
EDT2    BV      EDT21           ; CLEAR OVERFLOW BIT
EDT21   ZALH    HYST            ; HYSTERESIS COUNTER
        SUBH    TMP5            ; TMP5 = 1927 = 32768/15
        SACH    HYST
*
*************************************************************************
*
*    IN CASE OF OVERFLOW, DECLARE LOSS OF ENERGY DETECT
*
*************************************************************************
*
        BV      EDT02
        RET
EDT02   LACK    0BFh
        AND     STWRD
        SACL    STWRD
        RET
*
*************************************************************************
*
*    FOLLOWING LINES ARE EXECUTED IF AFE GAIN IS HIGH, BUT NO ENERGY DETECT.
*    CHECK IF ALPHA < 21.28 (I.E., RECEIVE LEVEL >-43.5 DBM) AND INCREMENT
*    HYSTERESIS COUNTER IF IT IS. OTHERWISE, EXIT.
*
*************************************************************************
*
EDT01   LACK    THRES2          ; 21.28 IN S7.8
        TBLR    TMP0
        LAC     TMP0
        SUB     ALPHA
        BLZ     EDT3
*
*************************************************************************
*
*    ALPHA < 21.28 => INCREMENT HYSTERESIS COUNTER
*
*************************************************************************
*
        BV      EDT011          ; CLEAR OVERFLOW BIT
EDT011  ZALH    HYST
        ADDH    TMP5            ; TMP5 CONTAINS INC. >F0F
        SACH    HYST
*
*************************************************************************
*
*    DETECT BIT STWRD[6]  = 1.
*
*************************************************************************
*
        BV      EDT04           ; IN CASE OF OVERFLOW, SET ENERGY
        RET
EDT04   LAC     ONE,6
        OR      STWRD
        SACL    STWRD
        RET
*
*************************************************************************
*
*    IF AFE GAIN STAGE IS BYPASSED, CHECK LEVEL OF ALPHA
*
*************************************************************************
*
EDT1    LACK    THRES5
        TBLR    TMP0
        LAC     TMP0
        SUB     ALPHA
        BGZ     EDT3
*
*************************************************************************
*
*    IF ALPHA > THRES5 (20.09 IN S7.8) THEN TURN AFE GAIN STATUS WORD BIT ON.
*
*************************************************************************
*
        LACK    080h
        OR      STWRD
        SACL    STWRD
        LACK    014h
        SACL    GN
EDT3    RET
*
*************************************************************************
*
*    ROUTINE FOR SWITCHING THE AFE ON/OFF
*
*************************************************************************
```

```
*************************************************************************
*
*    ZERO BAUD ENERGY REGISTER
*
*************************************************************************
*
SWITCH    ZAC
          SACL      AVESQR
*
          LACK      010h            ; MASK OFF UNWANTED BITS
          AND       GN
          BZ        AFEOFF
*
*************************************************************************
*
*    CHECK IF THE GAIN SHOULD BE ON
*
*************************************************************************
*
          LACK      0Fh             ; MASK OFF THE AFE ON BIT
          AND       GN
          SUB       ONE             ; DECREMENT THE COUNTER
          BZ        SWTCH1
*
          SACL      GN              ; SAVE GN VALUE
          LACK      010h            ; LOAD THE AFE ON BIT
          OR        GN
          SACL      GN              ; RESTORE AFE ON BIT
          RET
*
SWTCH1    SACL      GN              ; RESET THE GN VALUE TO ZERO
          LACK      THRES6          ; LOAD THE NEW ALPHA VALUE
          TBLR      ALPHA           ; RESET ALPHA TO 5.05
          RET
*
AFEOFF    LAC       GN
          SUB       ONE             ; DECREMENT THE COUNTER
          BZ        SWTCH2
*
          SACL      GN
          RET
*
SWTCH2    SACL      GN
          LACK      THRES4          ; LOAD NEW ALPHA VALUE
          TBLR      ALPHA           ; RESET ALPHA TO 8.98 IN S7.8
          RET
*
```

# An All-Digital Automatic Gain Control

Al Lovrich
Raj Chirayil

Digital Signal Processor Products — Semiconductor Group
Texas Instruments

One of the basic structural blocks of a modem receiver is the Automatic Gain Control (AGC). The AGC is an adaptive system that operates over a wide dynamic range while maintaining the output signal at a constant level. This is necessary for the proper operation of the carrier recovery and clock recovery algorithms of the modem receiver.

This application report describes an all-digital implementation of an AGC on a TMS320C17 Digital Signal Processor (DSP). The AGC is designed specifically for modem applications. The structure of this application report is as follows:

- The first section provides an overview of modem receiver structure and implementation.
- Section two discusses the AGC block diagram and the motivation for using an AGC in a modem receiver.
- The last section covers the AGC hardware and software implementation aspects on a TMS320C17 DSP.

## Introduction

A modem (MOdulator/DEModulator) is a device that modulates baseband signals at the transmitter and demodulates the received data at the receiver. To achieve full-duplex operation, frequency division multiplexing is employed, in which both modems simultaneously transmit and receive information over a single channel by dividing the telephone bandwidth into separate frequency bands: one for transmit with a carrier frequency of 1200 Hz and one for receive with a carrier frequency of 2400 Hz. A modem receiver consists of several functional blocks, which include answer/originate bandpass filters, AGC, demodulator, adaptive equalizer, clock recovery, carrier recovery, decision block, decoder, and descrambler.

In this report, we are concerned with the implementation of a DSP-based AGC for a V.22 bis modem product[1]. One of the basic structural blocks of a modem receiver is the AGC. The AGC is an adaptive system that operates over a wide dynamic range while maintaining the output signal at a constant level. The AGC is needed because several modules within the receiver use amplitude thresholds to make their decisions. These threshold levels must remain constant over the entire dynamic range of input signals, typically from –9 dbm to –43 dBm[2]. This is achieved through use of a software AGC, which multiplies the input signal with a gain factor, depending on the actual received signal level.

### Modem Transmitter

The CCITT V.22 bis standard is a 2400-bps modem that uses Quadrature Amplitude Modulation (QAM) technique to transmit and receive data through the communications channel. This section presents an overview of QAM systems and the equations governing their operations.

In Quadrature Amplitude Modulation, the information is encoded as phase changes of the transmitted carrier and amplitude variations. With R denoting the amplitude and $\phi$ the phase change, the transmitted signal s(n) is mathematically represented as

$$s\ (n)\ =\ R\ \cos\ (\omega_c\ +\ \phi\ ) \qquad (1)$$

where $\omega_c$ is the carrier frequency. Simplifying (1) and substituting $I_n = R \cos(\phi)$ and $Q_n = -R \sin(\phi)$ into it results in (2); this is used to describe QAM modulation systems.

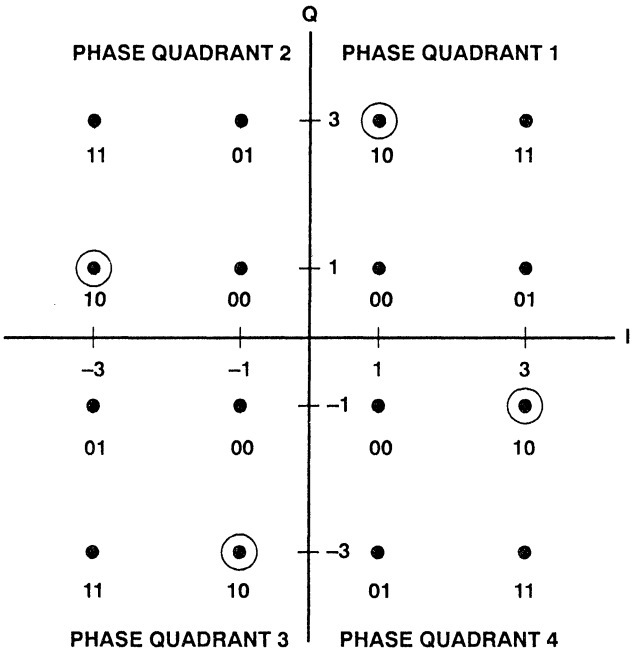
$$s(n) = I_n \cos(\omega_n) + Q_n \sin(\omega_n) \quad (2)$$

Transmission of a baseband sequence {In,Qn} is called *quadrature* transmission, with two carriers in phase quadrature to one another (cos $\omega_c t$ and sin $\omega_c t$) transmitted simultaneously over the same communications channel. Figure 1 shows a two-dimensional diagram of the signals of form (2) with the horizontal axis corresponding to the *in-phase* signal ($I_n$) and the vertical axis representing the *quadrature* signal ($Q_n$). These signal points are referred to as a 16-symbol QAM-signal constellation.

Each value of the $\{I_n,Q_n\}$ corresponds to one signaling element transmitted. The number of signaling elements per second is referred to as the baud rate. The *baud rate* is set by the CCITT V.22 bis recommendation to 600. By encoding four incoming bits (*quadbits*) in a single baud, transmission of 2400 bps is accomplished.

The encoding of the incoming data stream $d_s(n)$ into values of the sequence $\{I_n,Q_n\}$ is accomplished by the encoder. The encoder maps the first two bits of a quadbit as a phase quadrant change relative to the quadrant occupied by the preceding signal element. The last two bits of the quadbit define one of four signaling elements associated with the new quadrant[3].

Figure 1. V.22 bis Signal Constellation

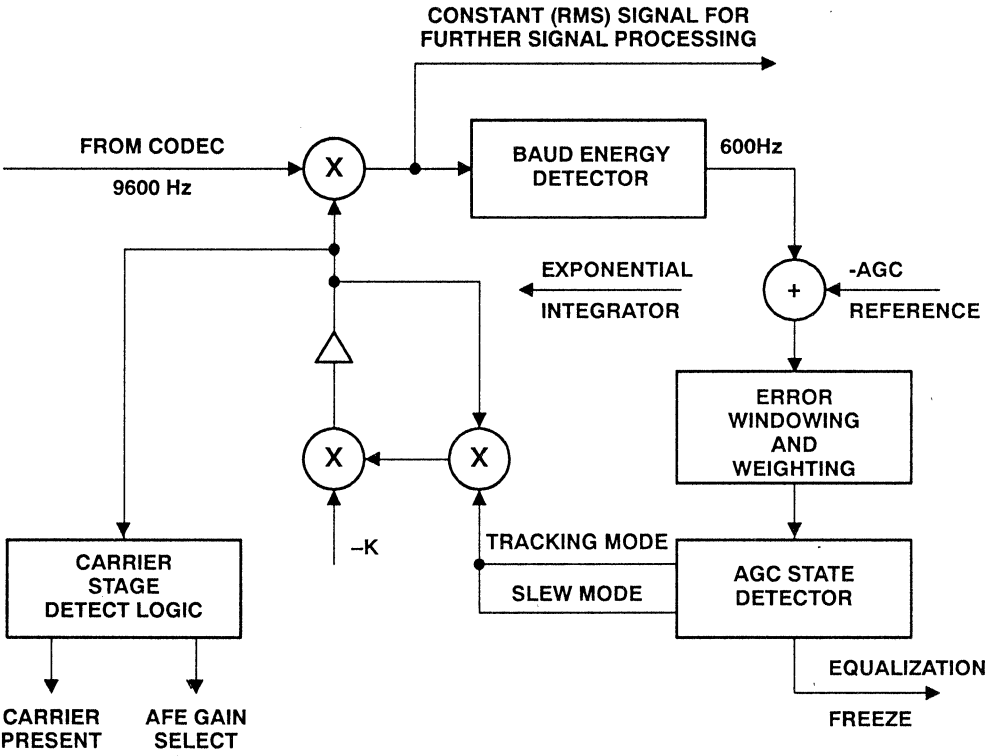



## The AGC Algorithm

The AGC circuit is a closed-loop regulating system that maintains the output level of an amplifier at a constant level, even though the input signal may vary substantially. The AGC modeling and design techniques based on linear system design have been studied in detail[4]. The global stability of AGC loops assures the designer that the overall loop will stay stable under considerable weaker conditions if the proper design rules are followed[5].

Figure 2 is a block diagram of the modem automatic gain control. The AGC algorithm is partitioned into tasks performed once per sampling interval, and tasks performed once per baud interval. The sampling rate for the overall system is the designer's choice as long as it satisfies the Nyquist's criterion. A widely used sampling rate for the communications channel is 8 kHz. In the system in Figure 2, the sampling rate is chosen to be an integer multiple of the baud rate. Therefore, a sampling rate of 9.6 kHz is selected. This value is divisible by the master crystal frequency of 18.432 MHz.

Figure 2. Modem AGC Block Diagram




## Baud Energy Detector

In Figure 2, every incoming linearized PCM sample is multiplied by the AGC gain factor. The result is available to the modem reciever for further signal processing. It is also used to update the baud energy detector. The energy of a baud interval is computed according to

$$E = \sum x_n^2 \tag{3}$$

where $x_n$ represents the incoming samples. The accumulated baud energy is then compared against a reference level, which depends on the modulation scheme. This comparison is necessary to compute the AGC loop error signal. It is this error that the AGC is trying to minimize.

The QAM transmitted signal shown in (2) can be rewritten, taking waveform shaping into account as follows

$$s(t) = \sum I_n \, g(t-nT) \cos \omega_c t + \sum Q_n \, g(t-nT) \sin \omega_c t \tag{4}$$

where $\omega_c = 2\pi f_c$, where $f_c$ = carrier frequency
$g(t)$ = shaping waveform
$T$ = sampling interval
$I_n, Q_n$ = data symbols

## AGC Reference Energy

The signal energy for a particular constellation point $(I_n, Q_n)$ is given by (see Appendix A)

$$E_n = 1/2\ (I_n^{\,2} + Q_n^{\,2}) \tag{5}$$

The energy reference level is chosen to be

$$E_{ref} = E\ \{\ E_n\ \} \tag{6}$$

where E{ } denotes the expectation operation. The V.22 bis modem standard requires the transmitter to scramble the incoming digital sequence from the DTE and descramble the decoded data in the receiver[2,3]. The use of scrambler in the modem transmitter effectively randomizes the data and avoids data-dependent patterns in the transmitted sequence. This allows the constellation point sequences to be modeled as a random sequence, with each point having an equal probability of occurrence of $E\{(I_n, Q_n)\} = 1/N$. Therefore, (6) can be written as

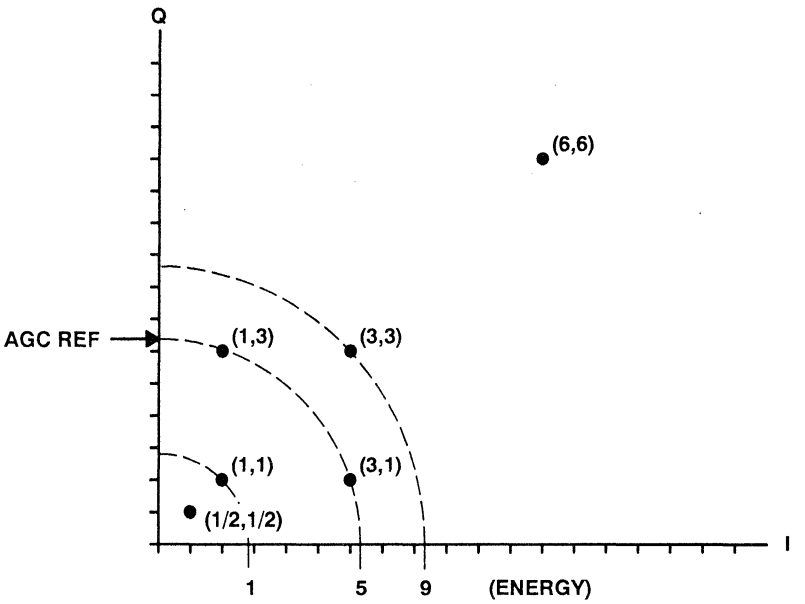
$$E_{ref} = \sum_{n=1}^{N} 1/N\ (\ E_n\ ) \tag{7}$$

Figure 3 shows a portion of the signal constellation diagram of a V.22 bis modem.

Applying (7) to all 16 constellation points results in

$$\begin{aligned} E_{ref} &= 1/16\ \{\ 4\ [\ (\ 1^2 + 3^2\ ) + 1/2\ (\ 1^2 + 1^2\ ) + 1/2\ (3^2 + 3^2\ )\ ]\ \} \\ &= 1/16\ \{\ 4\ [\ (\ 10\ ) + (\ 1\ ) + (\ 9\ )\ ]\ \} \\ &= 5 \end{aligned} \tag{8}$$

**Figure 3. Signal Energy Constellation Diagram**




In Figure 3, constellation points (3,3) and (1,1) with respective energy contents of 9 and 1 lie outside the reference level of 5. A window function is then necessary so that the AGC does not treat these energy variations around the nominal energy as distortions induced by the communication channel.

Therefore, the AGC should apply corrections when the incoming signal level is outside the interval (1,9)(see Figure 3). Such implementation, however, neglects the effects of intersymbol interference (ISI). ISI arises in systems whenever pulses are transmitted in a band-limited channel. In such channels, pulses tend not to die out immediately, and the tail from one pulse interferes with the next pulse. ISI-related effects are more easily shown when constant amplitude modulation techniques, such as DPSK, are considered. In a DPSK modem receiver, the received signal exhibits gain variations, that are entirely due to ISI. Since the modem equalizer compensates for ISI, the AGC should not act upon ISI-related signal-level variations, because this would introduce noise into the modem receiver and degrade the overall performance.

The received signal r(t) at the input of the receiver is the convolution of the channel impulse response h(t) with the transmitted symbols $x_j$ in

$$r\,(\,t\,) = \sum_{j} x_j\, h\,(\,t - jT\,) + \mu(t) \tag{9}$$

where $\mu(t)$ is the additive white Gaussian noise. For the effects of ISI to be seen, the received signal must be sampled at the instant $t_0+kT$ with $t_0$ incorporating the sampler phase and delay effects.

$$r\,(\,t_0 + kT\,) = x_k h\,(\,t_0\,) + \sum_{j \neq k} x_j h\,(\,t_0 + kT - jT\,) + \mu\,(\,t_0 + kt\,) \tag{10}$$

The first term of the right-hand side of (10) is the desired signal and is used to determine the transmitted symbol, while the middle term is ISI, which arises from the neighboring symbols [6]. With $x_k$, a constant amplitude sequence, the middle term in (10) results in received signal amplitude variations. Thus, the AGC design must incorporate an energy window around the energy reference level as defined by $x_k$'s.

## DSP Implementation

### Hardware

This section describes the hardware requirements of the modem. The modem hardware consists of the following functional blocks:

1) Host Interface
2) DSP
3) Controller
4) Controller-DSP Interface
5) Analog Front-End
6) Telephone Line Interface

For the purpose of understanding the operation of the Automatic Gain Control (AGC), the discussion is limited to only the analog front end.

### Modem Analog Front End

The function of the analog front end (AFE) in the modem is to convert the analog signals received on the telephone line to digital data that can be processed by a digital signal processing device, in this case the TMS320C17. Depending on the modem standard that is implemented, the modem AFE could further assist the DSP by preventing as many of the unwanted signals as possible from being received by the DSP. This reduces the signal conditioning and preprocessing required by the DSP, which, in turn, reduces the computational requirement.

In the implementation described here, the modem AFE performs the bandpass filtering, a single-step gain stage, and the A/D-D/A conversions. Although the modem hardware also includes the two-to-four wire conversion and the proper telephone line interface and impedance matching, it will not be considered in this discussion.

### Split-band Filtering

In Frequency Division Multiplexing (FDM) modems, the originating and answering stations use different carrier frequencies to transmit data[2]. For V.22 bis modems, the originating modem transmits data using a 1200-Hz carrier and receives signals from the remote modem at 2400 Hz. Since these signals are carried over the two-wire Public Switched Telephone Network (PSTN) for a full duplex communication, both signals are present in the telephone line simultaneously. For a modem to prevent its transmitted signal from interfering with its received signal, it must eliminate its own transmit signal at its receiver. Since the two modems use separate carrier frequencies to

transmit, this task becomes relatively easy. It is done by bandpass filtering the received signal with the passband filter being centered at the transmit carrier frequency of the remote modem.

This implementation uses a commercially available modem filter that has special modes to allow call-progress signal monitoring. This filter must provide adequate adjacent channel rejection while maintaining linear phase. The filter must operate over the entire dynamic range required by the modem, typically from 0 dBm to –43 dBm. For better Signal-to-Noise Ratio (SNR) and linear phase, it is desirable not to operate the filter and the Analog-to-Digital converter at very low signal levels. If signals are weak, an external gain stage (turned on/off under software control) in the receive signal path easily accomplishes this goal.

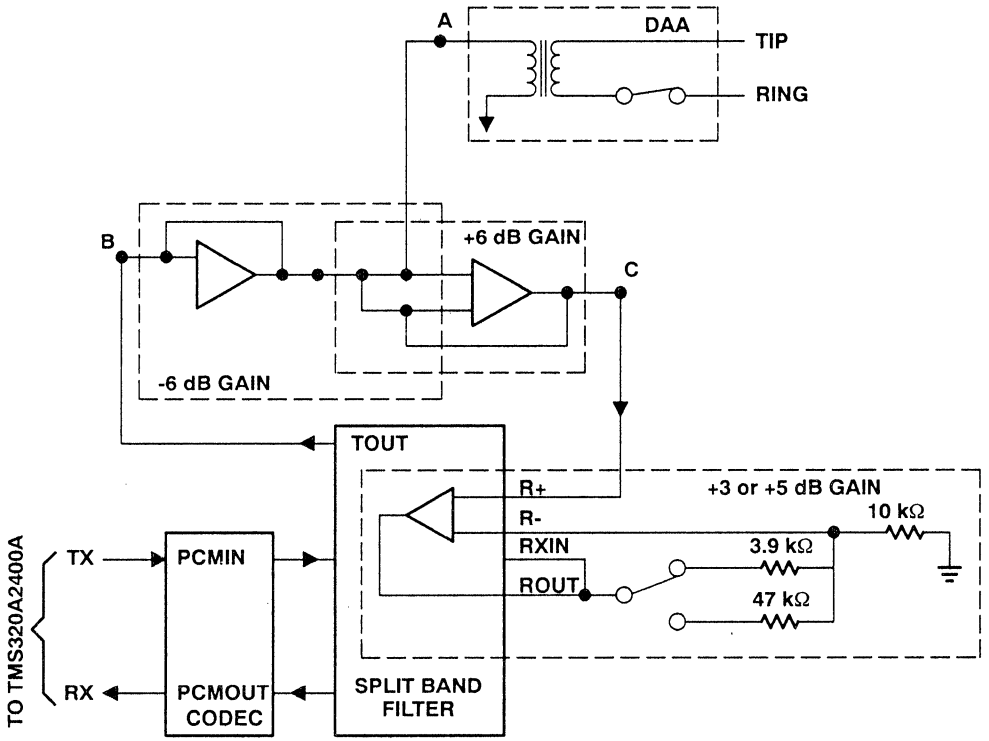
## Hardware Gain Control

The hardware gain switch is implemented by changing the gain in the analog input buffer to the filter. When the average signal energy falls below –28 dBm, the DSP sets a status line to the modem controller. The controller, in turn, switches on a different resistor in the feedback circuit of the op-amp, increasing the gain by 12 dB. This switching is normally done only once during call initialization. However, if the connection starts with low-level signals and later the signals become stronger due to change in line impedance, the DSP resets this status line to the controller. The modem controller then turns off the external gain stage.

When the modem received signal is actually at the threshold level, it is possible that the external gain could frequently be turned on and off by slight changes in signal level. To prevent this, a 4-dB hysteresis has been established between external gain On and Off. This means the external gain will be turned On when the average signal level is less than –24 dBm and will be turned Off when the level is more than –28 dBm. Figure 4 shows the AFE schematic of the modem.

| | LINE (dBm) | AFE GAIN (dB) | CODEC (dBm) |
|---|---|---|---|
| Rx level | –12 | 0 | – 9 |
| | –24 | 0 | –21 |
| | –25 | 12 | –10 |
| | –43 | 12 | –28 |

Figure 4. Modem AFE Schematic




## Codec Interface

The TMS320A2400A features hardware companding logic to interface directly to a μ-law codec[1]. The SCLK output provides the master clock frequency for the codec, and the FR provides the transmit and receive framing signal to the codec. Since the modem algorithm uses a 9.6-kHz sampling frequency, the codec must complete one A/D,D/A conversion at this rate.

The DSP serial port control register was programmed to provide an SCLK which is generated by dividing the DSP's input clock by ten. Thus, using an 18.432–MHz crystal as the DSP's clock input, a 1.8432-MHz SCLK was generated. The TCM29C19 uses an internal divide ratio of 192 to generate the 9.6-kHz sampling rate.

## Software

The previous section provided a brief overview of the hardware design issues associated with the AGC for a V.22 bis modem. DSP implementation issues are the focus throughout the rest of this report. All values are represented in decimal format unless otherwise noted. Data values in a digital system are not integers, but they must be manipulated as such on an integer processor. Appendix B provides an overview of fractional number representation on a two's-complement fixed-point device.

We choose to represent the signal within the AGC loop in S4.11 format. Recall that the $\{I_n,Q_n\}$ sequence can assume any value from the sequence $\{\pm1, \pm3\}$. This means that the sequence is bound in the ±3 range. We use three bits to represent the values in the given range, while the rest of the 12 bits can be treated as the fractional part that accommodates noise. Allocating an extra bit to the $\{I_n,Q_n\}$ sequence fully represents the RMS signal and allows for some gain hit.

For QAM signals, experimentation has shown that the ratio of peak signal to RMS signal is approximately 3 to 1. The maximum peak signal that can be represented using S4.11 notation is 16 (see Appendix B); therefore, 16 represents the peak value a QAM signal can attain using this notation. The $RMS_{max}$ is hence 5.33, which corresponds to approximately 14.5 dB (20 log 5.33). We design the system to work with a 10-dB gain hit. It follows that the AGC should maintain the signal level at approximately 4.5 dB or 1.69 RMS level.The constant level of 1.69 RMS represented in S4.11 format is 3461.12. The AGC loop maintains an average squared level of 2.86, or $(1.69)^2$, per sample. Therefore, to determine the average baud energy, the sample energy must be multiplied by 16. The resultant value (45.8) is represented in S10.5 format (corresponding to 1466 (05BAh) in S15.0 format), the actual value used in the implementation (see Appendix E for the code listing).

As shown in the previous section, the reference energy for a V.22 bis modem is 5. This corresponds to the energy level of the constellation points (1,3) and (3,1), shown in Figure 3. Hence it is possible to map the average baud energy of 5 into 45.8. Extending the mapping to the other energy levels results in the following:

| Average Baud Energy | S10.5 Format | S15.0 Format |
|---|---|---|
| 1 maps into | 9.16 | 292 |
| 5 maps into | 45.8 | 1466 |
| 9 maps into | 82.4 | 2632 |

## Error Windowing and Weighting

In the previous section, the need was established for an energy window around the nominal baud energy level to compensate for the effects of intersymbol interference. The AGC is not designed to, and should not be expected to, compensate for ISI. The equalizer in the modem receiver is designed for this purpose [6]. Experimental window values of 1320 and 950 were chosen for QAM and DPSK modes of operation, respectively.

The windowed error signal must be weighted appropriately to provide an approximate one-to-one relationship between the positive and negative energy errors. In Figure 3, the disparity between the positive and negative errors can be observed. Assume that the received points are (6,6) and (0.5,0.5). The QAM signal energy can be calculated as

$$E_{QAM} = 1/2 \ ( \ I_n^{\ 2} + Q_n^{\ 2} \ ) \qquad (11)$$

Therefore, the energy values of the received points are 36 and 0.25, respectively. When these energy values are represented in S10.5 (10552 and 73, respectively) and the deviation from the nominal energy level of 1466 is calculated, full scale error values of 9086 and –1393, respectively, are obtained. This indicates a nonlinear relationship between the received constellation points signal energy with respect to the nominal baud energy level. It is important to determine the weighting

factor to provide a parity between positive and negative errors while the AGC operates in the steady state or tracking mode. Appendix D provides a Fortran program to determine the best value for the expansion ratio of negative and positive energy values.

## AGC State Detector

The AGC always operates in one of two modes:

- Slew – (fast tracking mode) AGC uses a large step size to track the signal.
- Tracking – AGC adjusts the signal level by adjusting the gain factor via an exponential integrator loop.

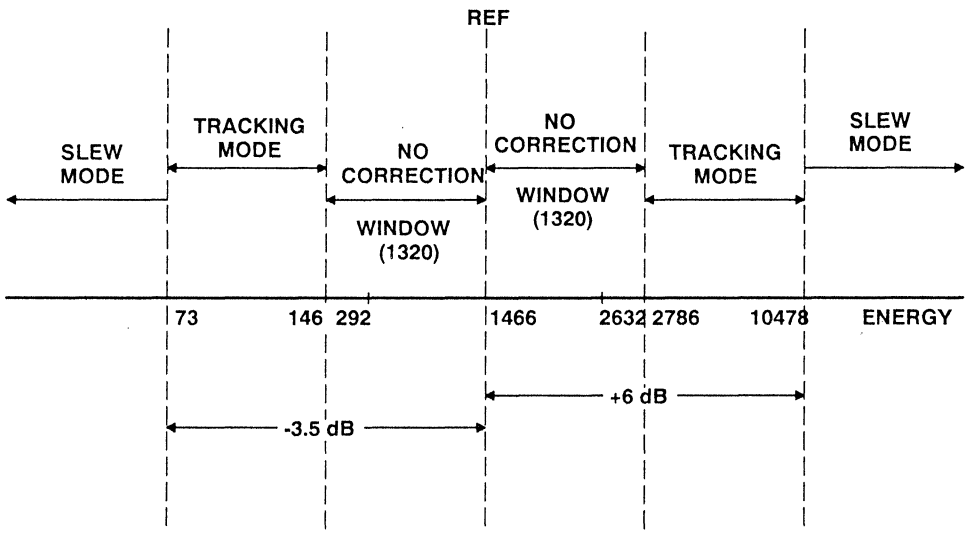
It is important to design the AGC to ignore relatively small gain changes on the telephone line. Otherwise, the AGC loop responds to the smallest variation in the signal level by switching to the slew mode. In this application, the AGC is designed to simply track the incoming signal when the received signal level varies by not more than ± 6 dB from the window values. These levels are calculated as follows:

$$10\log(x/2632) = +6\ dB \rightarrow x = 10478 \tag{12}$$

$$10\log(x/292) = -6\ dB \rightarrow x = 73 \tag{13}$$

As long as the incoming signal stays within these boundaries, the AGC simply adjusts the gain factor; otherwise, it will switch to the slew mode. Once the AGC determines that the error signal is within the tracking mode boundary, it switches back to the slow tracking mode as shown in 5.

**Figure 5. AGC Operating Modes**




Appendix C provides a FORTRAN program that determines the best weighting factor for a given QAM signal range. A weighting factor of 2 provided the approximate one-to-one relation-

ship. Since DPSK signals do not have amplitude variations, a value of 1 was chosen for the weighting factor when the modem operates in the V.22/Bell 212A mode.

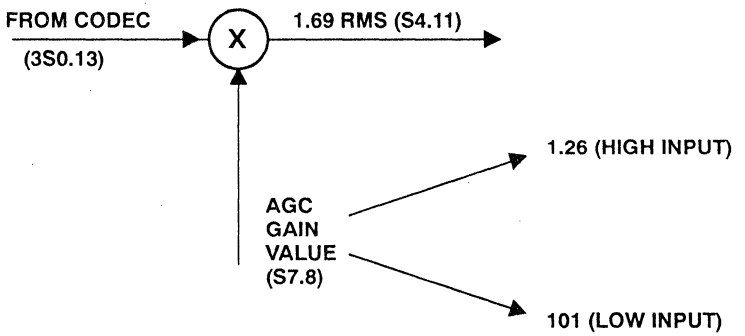
An upper and lower boundary for the AGC gain value must be determined. The V.22bis standard[3] requires the modem to operate at a signal level of –43 dBm. Therefore, the AGC is designed to work from the 0-dBm signal level to –50 dBm.

The DSP2400 contains a DSP-activated 12-dB gain switch. Therefore, our design should really have to cover only the range of 0 dBm to –38 dBm levels. The maximum codec output value is 1FFEh (8190 decimal) because the codec output is converted from 8-bit log value to 13-bit two's-complement value. When this value is saved in a data memory location of the TMS320C17 DSP, the number is sign-extended and is represented in 3S0.13 format. The $RMS_{max}$ is therefore 2730, which corresponds to a signal level of 0 dBm in our system. The minimum acceptable signal level from the codec corresponding to the –38-dBm level is computed as follows:

$$\begin{aligned} -38 &= 20 \log (RMS_{min} / 2730) \\ RMS_{min} &= 34.4 \end{aligned} \tag{14}$$

Given the maximum and minimum codec output values and the constant RMS output, it follows that $\alpha_{min} = 1.26$ and $\alpha_{max} = 101$ as shown in Figure 6.

**Figure 6. AGC Gain Value Computation**




The gain value requires 7 bits to represent; therefore, the S7.8 format is used to represent the $\alpha$ values.

## Exponential Integrator Loop

When the total baud energy stays within the window limits, the AGC is in the tracking mode and simply compensates for the changes in the signal levels by adjusting the gain factor appropriately. The gain factor is computed and updated via an exponential integrator loop. The exponential integrator loop implements the following function:

$$\alpha_{n+1} = \alpha_n \times (1 - Ke) \tag{15}$$

where the constant K determines the speed of convergence of the AGC closed loop. In our implementation, K is set to 1/2. This value corresponds to step sizes of ± 6 dB when the AGC is in the

slew mode. The error signal is in S0.15 format while $\alpha_n$ is in S7.8 format with the multiplication result in 2S7.23 format. When the upper half of the accumulator (ACCH) is saved with a left shift, the result is in S7.8 format. A further multiplication by 0.5 is necessary before carrying out the subtraction operation. Note that a divide by 2 is equivalent to a right shift, which cancels out the effect of the previous left shift. Therefore, saving ACCH with no shift accomplishes multiplication by K as shown in Appendix E.

The AGC is designed to declare carrier present when signal levels greater than –43 dBm appear at the input of the receiver. The response time for tone detection depends on the AGC design. The AGC uses a constant that is subtracted from a hysteresis counter, and presence of energy is declared when the counter underflows. It takes 9 bauds for the energy to be detected, corresponding to a response time of 15 ms.

## Conclusion

This application report has presented design and implementation techniques for an all-digital automatic gain control. The AGC has been implemented on a TMS320C17 digital signal processor as part of a commercial modem product (DSP2400). The approach of using a programmable processor resulted in minimal hardware configuration with excellent performance. The DSP implementation allows you to fine tune the AGC for your particular modem design, regardless of the modulation technique used.

## Acknowledgements

The author wishes to acknowledge the contribution of Technekron Communications Systems and George Troullinos of Texas Instruments. This report is based on their work.

## References

1) *DSP2400 Modem User's Guide*, Texas Instruments, 1987.
2) Troullinos, G., et al, *Theory and Implementation of a Splitband Modem Using the TMS32010* (literature number SPRA013), Texas Instruments, 1986.
3) Recommendation V.22 bis, "2400 bits per second duplex modem using the frequency division technique standardized for use on the general switched telephone network and on point-to-point 2-wire leased telephone-type circuits," *CCITT Redbook*, Volume III, 1984.
4) Mercy, D.V., "A Review of Automatic Gain Control Theory," *The Radio and Electronic Engineer*, Volume 51, Number 11/12, November/December 1981.
5) Green, D.N., "Global Stability Analysis of Automatic Gain Control Circuits," *IEEE Transactions on Systems and Circuits*, Volume CAS-30, Number 2, February 1983.
6) Falconer, D.D., "Jointly Adaptive Equalization and Carrier Recovery in Two-Dimensional Digital Communication Systems," *Bell System Technical Journal*, Volume 55, Number 3, March 1976.
7) Antoniou, A., *Digital Filters: Analysis and Design*, John Wiley & Sons, 1986.
8) Lovrich, A., et al, "An All Digital Automatic Gain Control," *Proceedings of ICASSP 88*, Pages 1734–1737.

# Appendix A

## QAM Signal Energy

The general form of a QAM signal is written as

$$\begin{aligned} s\ (\ t\ ) &= R\ (\ t\ )\ \cos\ [\omega_c t + \phi\ (\ t\ )] \\ &= I_n\ \cos\ \omega_c\ t + Q_n \sin \omega_c\ t \end{aligned} \tag{16}$$

The energy in a signal s(t) is defined as

$$E_{QAM} = \int_{-\infty}^{\infty} s^2\ (\ t\ )\ dt \tag{17}$$

Substituting (16) into (17) results in

$$\begin{aligned} E_{QAM} &= \int_0^T s^2\ (\ t\ )\ dt \\ &= \frac{1}{2\pi} \int_0^{2\pi} (\ I_n{}^2\ \cos^2 \omega_c\ t + Q_n{}^2\ \omega_c\ t + 2 I_n\ Q_n\ \sin\ \omega_c\ t\ \cos\ \omega_c\ t\ )\ dt \\ &= \frac{1}{2\pi} \int_0^{2\pi} 1/2\ [\ I_n{}^2\ (\ 1 + \cos\ 2\omega_c\ t\ )\ ]\ dt + \frac{1}{2\pi} \int_0^{2\pi} 1/2\ [Q_n\ 2^2\ (\ 1 - \cos 2\omega_c\ t)\ ] \\ &\quad + \frac{1}{2\pi} \int_0^{2\pi} I_n\ Q_n\ \sin 2\omega_c\ t\ dt \end{aligned} \tag{18}$$

When the three terms in (18) are integrated, the sine and cosine terms drop out since the average energy of sinusoidal signals is zero. Therefore, (18) simplifies to

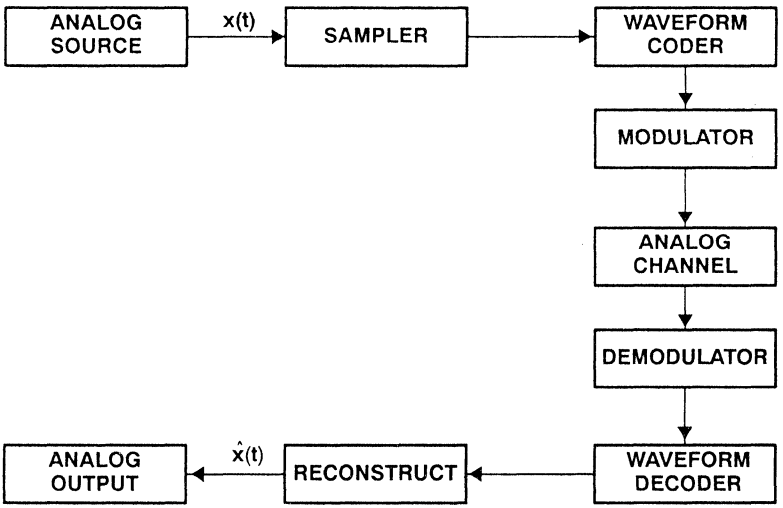
$$E_{QAM} = 1/2\ (\ I_n{}^2\ +\ Q_n{}^2\ ) \tag{19}$$

# Appendix B

## Fractional Number Representation Overview

A typical digital communication system is shown in Figure 7. Two blocks (marked as waveform coder and waveform decoder) are of interest. These blocks are collectively referred to as a codec, especially when both coder and decoder are implemented on a single device. An example is the TCM29C13 PCM codec, which consists of an amplitude quantizer and binary codeword generator.

**Figure 7. A Typical Communication Channel**




The quantized data represent instantaneous values of a continuous-time signal in digital form. On the TMS320C17, these data values are represented in two's-complement arithmetic[7]. The binary representation of a two's-complement value is as follows:

$$A = a_0 + \sum_{i=1}^{15} a_i \, 2^{-i} \tag{20}$$

Consider that the incoming samples are coming from a 16-bit linear ADC. The data coming out of the ADC consist of a sign bit at the most significant location, followed by the binary point. This information can be represented in Q15 format or, alternately, S0.15 format. This translates into the following upperbound and lowerbound limits with increments of $2^{-15}$ (0.00003051):

$$\begin{aligned} (\,2^{15}-1\,) \,/\, 2^{15} &= 0.99996948 \\ -2^{15} \,/\, 2^{15} &= -1 \end{aligned} \tag{21}$$

If two Q15 (S0.15) numbers are multiplied, the result is a number in Q30 (SS0.30) format. When the Q30 number resides in the 32-bit accumulator of the TMS320C17, the binary point fol-

lows the second most-significant bit. Assuming that the output of the encoder section is also Q15 format, the Q30 number must be adjusted by left-shifting by one while maintaining the most-significant 16 bits of the result. This is accomplished with a sach y,1. This instruction shifts the Q30 (SS0.30) number to the left by one and, following the shift, stores the upper 16 bits of the accumulator. The y value is in Q15 (S0.15) format.

The S notation is used consistently throughout this application report. The following table should assist you with the conversion between Q notations, S notations, and equivalent decimal representations.

**Table 1. S Notation, Q Notation, and Decimal Conversion Information**

| Q Notation | S Notation | Decimal Equivalent |
|---|---|---|
| Q15 | S0.15 | $-1 \le N \le 0.9999695$ |
| Q14 | S1.14 | $-2 \le N \le 1.9999390$ |
| Q13 | S2.13 | $-4 \le N \le 3.9998779$ |
| Q12 | S3.12 | $-8 \le N \le 7.9997559$ |
| Q11 | S4.11 | $-16 \le N \le 15.9995117$ |
| Q10 | S5.10 | $-32 \le N \le 31.9990234$ |
| Q9 | S6.9 | $-64 \le N \le 63.9980469$ |
| Q8 | S7.8 | $-128 \le N \le 127.9960938$ |
| Q7 | S8.7 | $-256 \le N \le 255.9921875$ |
| Q6 | S9.6 | $-512 \le N \le 511.9804375$ |
| Q5 | S10.5 | $-1024 \le N \le 1023.96875$ |
| Q4 | S11.4 | $-2048 \le N \le 2047.9375$ |
| Q3 | S12.3 | $-4096 \le N \le 4096.875$ |
| Q2 | S13.2 | $-8192 \le N \le 8191.75$ |
| Q1 | S14.1 | $-16384 \le N \le 16383.5$ |
| Q0 | S15.0 | $-32768 \le N \le 32767$ |

# Appendix C

The following is a Fortran program listing that creates a table of AGC gain values and its relation to the input signal strength. The table also includes the corresponding peak input signal level and its RMS equivalent.

dbm.for

```
c  This program determines the values of dBm signal levels in a QAM system:
c  Double Precision Peak, RMS, Alpha, dBm.
c
c     - Peak is the peak signal input to the receiver side.
c
c     - RMS is the RMS signal input to the receiver, where in a QAM system is
c       equal to on third of the peak value.
c
c     - Alpha is the gain value.
c
c  In this AGC design, the max codec input is actually +2.2 dBm (equivalent to
c  8190 peak.

        open    (1,file = 'dbm.dat',status = 'new')

        write   (1,8)
    8   format  (5x,' dBm',15x,'peak',16x,'rms',17x,'alpha')
        write   (1,9)
    9   format  (5x,' ===',15x,'====',16x,'===',17x,'=====')
                dbm = -27.1
                max = 2.
  999   if      (dbm .gt. max) goto 1000
                dbm = dbm + 0.1
                peak = 8190. * (10 ** ((dbm - max) / 20.))
                rms = peak / 3.
                alpha = 3461.12 / rms
        write   (1,10) dbm,peak,rms,alpha
   10   format  (f10.1,10x,f10.4,10x,f10.4,10x,f10.2)
        goto    999
c
 1000   stop
        end
```

| dBm | Peak | RMS | Alpha |
|---|---|---|---|
| -27.0 | 290.5922 | 96.8641 | 35.73 |
| -26.9 | 293.9571 | 97.9857 | 35.32 |
| -26.8 | 297.3609 | 99.1203 | 34.92 |
| -26.7 | 300.8042 | 100.2681 | 34.52 |
| -26.6 | 304.2873 | 101.4291 | 34.12 |
| -26.5 | 307.8108 | 102.6036 | 33.73 |
| -26.4 | 311.3751 | 103.7917 | 33.35 |
| -26.3 | 314.9807 | 104.9936 | 32.97 |
| -26.2 | 318.6280 | 106.2093 | 32.59 |
| -26.1 | 322.3175 | 107.4392 | 32.21 |
| -26.0 | 326.0498 | 108.6833 | 31.85 |
| -25.9 | 329.8252 | 109.9417 | 31.48 |
| -25.8 | 333.6444 | 111.2148 | 31.12 |
| -25.7 | 337.5079 | 112.5026 | 30.76 |
| -25.6 | 341.4160 | 113.8053 | 30.41 |
| -25.5 | 345.3694 | 115.1231 | 30.06 |
| -25.4 | 349.3686 | 116.4562 | 29.72 |
| -25.3 | 353.4141 | 117.8047 | 29.38 |
| -25.2 | 357.5065 | 119.1688 | 29.04 |
| -25.1 | 361.6462 | 120.5487 | 28.71 |
| -25.0 | 365.8338 | 121.9446 | 28.38 |
| -24.9 | 370.0700 | 123.3567 | 28.06 |
| -24.8 | 374.3552 | 124.7851 | 27.74 |
| -24.7 | 378.6900 | 126.2300 | 27.42 |
| -24.6 | 383.0751 | 127.6917 | 27.11 |
| -24.5 | 387.5109 | 129.1703 | 26.80 |
| -24.4 | 391.9980 | 130.6660 | 26.49 |
| -24.3 | 396.5372 | 132.1791 | 26.19 |
| -24.2 | 401.1288 | 133.7096 | 25.89 |
| -24.1 | 405.7737 | 135.2579 | 25.59 |
| -24.0 | 410.4723 | 136.8241 | 25.30 |
| -23.9 | 415.2254 | 138.4085 | 25.01 |
| -23.8 | 420.0335 | 140.0112 | 24.72 |
| -23.7 | 424.8972 | 141.6324 | 24.44 |
| -23.6 | 429.8173 | 143.2724 | 24.16 |
| -23.5 | 434.7943 | 144.9314 | 23.88 |
| -23.4 | 439.8290 | 146.6097 | 23.61 |
| -23.3 | 444.9220 | 148.3073 | 23.34 |
| -23.2 | 450.0740 | 150.0247 | 23.07 |
| -23.1 | 455.2856 | 151.7619 | 22.81 |
| -23.0 | 460.5575 | 153.5192 | 22.55 |
| -22.9 | 465.8905 | 155.2968 | 22.29 |
| -22.8 | 471.2853 | 157.0951 | 22.03 |
| -22.7 | 476.7425 | 158.9142 | 21.78 |
| -22.6 | 482.2629 | 160.7543 | 21.53 |
| -22.5 | 487.8473 | 162.6158 | 21.28 |
| -22.4 | 493.4963 | 164.4988 | 21.04 |
| -22.3 | 499.2107 | 166.4036 | 20.80 |
| -22.2 | 504.9913 | 168.3304 | 20.56 |
| -22.1 | 510.8388 | 170.2796 | 20.33 |
| -22.0 | 516.7540 | 172.2513 | 20.09 |
| -21.9 | 522.7378 | 174.2459 | 19.86 |

| | | | |
|---|---|---|---|
| -21.8 | 528.7908 | 176.2636 | 19.64 |
| -21.7 | 534.9139 | 178.3046 | 19.41 |
| -21.6 | 541.1079 | 180.3693 | 19.19 |
| -21.5 | 547.3736 | 182.4579 | 18.97 |
| -21.4 | 553.7119 | 184.5706 | 18.75 |
| -21.3 | 560.1236 | 186.7079 | 18.54 |
| -21.2 | 566.6095 | 188.8698 | 18.33 |
| -21.1 | 573.1706 | 191.0569 | 18.12 |
| -21.0 | 579.8076 | 193.2692 | 17.91 |
| -20.9 | 586.5214 | 195.5071 | 17.70 |
| -20.8 | 593.3130 | 197.7710 | 17.50 |
| -20.7 | 600.1833 | 200.0611 | 17.30 |
| -20.6 | 607.1331 | 202.3777 | 17.10 |
| -20.5 | 614.1633 | 204.7211 | 16.91 |
| -20.4 | 621.2750 | 207.0917 | 16.71 |
| -20.3 | 628.4690 | 209.4897 | 16.52 |
| -20.2 | 635.7464 | 211.9155 | 16.33 |
| -20.1 | 643.1080 | 214.3693 | 16.15 |
| -20.0 | 650.5548 | 216.8516 | 15.96 |
| -19.9 | 658.0879 | 219.3626 | 15.78 |
| -19.8 | 665.7082 | 221.9027 | 15.60 |
| -19.7 | 673.4167 | 224.4722 | 15.42 |
| -19.6 | 681.2145 | 227.0715 | 15.24 |
| -19.5 | 689.1026 | 229.7009 | 15.07 |
| -19.4 | 697.0820 | 232.3607 | 14.90 |
| -19.3 | 705.1539 | 235.0513 | 14.72 |
| -19.2 | 713.3192 | 237.7731 | 14.56 |
| -19.1 | 721.5790 | 240.5263 | 14.39 |
| -19.0 | 729.9345 | 243.3115 | 14.23 |
| -18.9 | 738.3867 | 246.1289 | 14.06 |
| -18.8 | 746.9369 | 248.9790 | 13.90 |
| -18.7 | 755.5860 | 251.8620 | 13.74 |
| -18.6 | 764.3352 | 254.7784 | 13.58 |
| -18.5 | 773.1858 | 257.7286 | 13.43 |
| -18.4 | 782.1389 | 260.7130 | 13.28 |
| -18.3 | 791.1956 | 263.7319 | 13.12 |
| -18.2 | 800.3573 | 266.7858 | 12.97 |
| -18.1 | 809.6250 | 269.8750 | 12.82 |
| -18.0 | 819.0000 | 273.0000 | 12.68 |
| -17.9 | 828.4835 | 276.1612 | 12.53 |
| -17.8 | 838.0769 | 279.3590 | 12.39 |
| -17.7 | 847.7814 | 282.5938 | 12.25 |
| -17.6 | 857.5983 | 285.8661 | 12.11 |
| -17.5 | 867.5288 | 289.1763 | 11.97 |
| -17.4 | 877.5743 | 292.5248 | 11.83 |
| -17.3 | 887.7361 | 295.9120 | 11.70 |
| -17.2 | 898.0156 | 299.3385 | 11.56 |
| -17.1 | 908.4141 | 302.8047 | 11.43 |
| -17.0 | 918.9331 | 306.3110 | 11.30 |
| -16.9 | 929.5738 | 309.8579 | 11.17 |
| -16.8 | 940.3378 | 313.4459 | 11.04 |
| -16.7 | 951.2264 | 317.0755 | 10.92 |
| -16.6 | 962.2411 | 320.7470 | 10.79 |
| -16.5 | 973.3833 | 324.4611 | 10.67 |
| -16.4 | 984.6545 | 328.2182 | 10.55 |
| -16.3 | 996.0563 | 332.0188 | 10.42 |
| -16.2 | 1007.5901 | 335.8634 | 10.31 |
| -16.1 | 1019.2574 | 339.7525 | 10.19 |
| -16.0 | 1031.0599 | 343.6866 | 10.07 |
| -15.9 | 1042.9990 | 347.6663 | 9.96 |
| -15.8 | 1055.0764 | 351.6921 | 9.84 |
| -15.7 | 1067.2936 | 355.7645 | 9.73 |
| -15.6 | 1079.6522 | 359.8841 | 9.62 |
| -15.5 | 1092.1540 | 364.0513 | 9.51 |
| -15.4 | 1104.8006 | 368.2669 | 9.40 |
| -15.3 | 1117.5936 | 372.5312 | 9.29 |
| -15.2 | 1130.5347 | 376.8449 | 9.18 |
| -15.1 | 1143.6257 | 381.2086 | 9.08 |
| -15.0 | 1156.8682 | 385.6227 | 8.98 |
| -14.9 | 1170.2641 | 390.0880 | 8.87 |
| -14.8 | 1183.8151 | 394.6050 | 8.77 |
| -14.7 | 1197.5231 | 399.1744 | 8.67 |
| -14.6 | 1211.3897 | 403.7966 | 8.57 |
| -14.5 | 1225.4170 | 408.4723 | 8.47 |
| -14.4 | 1239.6066 | 413.2022 | 8.38 |
| -14.3 | 1253.9606 | 417.9869 | 8.28 |
| -14.2 | 1268.4808 | 422.8269 | 8.19 |
| -14.1 | 1283.1691 | 427.7230 | 8.09 |
| -14.0 | 1298.0275 | 432.6758 | 8.00 |
| -13.9 | 1313.0579 | 437.6860 | 7.91 |
| -13.8 | 1328.2624 | 442.7541 | 7.82 |
| -13.7 | 1343.6430 | 447.8810 | 7.73 |
| -13.6 | 1359.2016 | 453.0672 | 7.64 |
| -13.5 | 1374.9405 | 458.3135 | 7.55 |
| -13.4 | 1390.8615 | 463.6205 | 7.47 |
| -13.3 | 1406.9669 | 468.9890 | 7.38 |
| -13.2 | 1423.2589 | 474.4196 | 7.30 |
| -13.1 | 1439.7394 | 479.9131 | 7.21 |
| -13.0 | 1456.4108 | 485.4703 | 7.13 |
| -12.9 | 1473.2753 | 491.0918 | 7.05 |
| -12.8 | 1490.3350 | 496.7783 | 6.97 |
| -12.7 | 1507.5922 | 502.5307 | 6.89 |
| -12.6 | 1525.0493 | 508.3498 | 6.81 |
| -12.5 | 1542.7086 | 514.2362 | 6.73 |
| -12.4 | 1560.5723 | 520.1908 | 6.65 |
| -12.3 | 1578.6429 | 526.2143 | 6.58 |
| -12.2 | 1596.9227 | 532.3076 | 6.50 |
| -12.1 | 1615.4142 | 538.4714 | 6.43 |
| -12.0 | 1634.1198 | 544.7066 | 6.35 |
| -11.9 | 1653.0420 | 551.0140 | 6.28 |
| -11.8 | 1672.1833 | 557.3944 | 6.21 |
| -11.7 | 1691.5463 | 563.8488 | 6.14 |
| -11.6 | 1711.1335 | 570.3778 | 6.07 |
| -11.5 | 1730.9475 | 576.9825 | 6.00 |
| -11.4 | 1750.9909 | 583.6636 | 5.93 |
| -11.3 | 1771.2664 | 590.4221 | 5.86 |
| -11.2 | 1791.7767 | 597.2589 | 5.80 |
| -11.1 | 1812.5245 | 604.1748 | 5.73 |

| | | | |
|---|---|---|---|
| -11.0 | 1833.5126 | 611.1709 | 5.66 |
| -10.9 | 1854.7437 | 618.2479 | 5.60 |
| -10.8 | 1876.2206 | 625.4069 | 5.53 |
| -10.7 | 1897.9462 | 632.6487 | 5.47 |
| -10.6 | 1919.9234 | 639.9745 | 5.41 |
| -10.5 | 1942.1550 | 647.3850 | 5.35 |
| -10.4 | 1964.6441 | 654.8814 | 5.29 |
| -10.3 | 1987.3936 | 662.4645 | 5.22 |
| -10.2 | 2010.4066 | 670.1355 | 5.16 |
| -10.1 | 2033.6860 | 677.8953 | 5.11 |
| -10.0 | 2057.2350 | 685.7450 | 5.05 |
| -9.9 | 2081.0566 | 693.6855 | 4.99 |
| -9.8 | 2105.1541 | 701.7180 | 4.93 |
| -9.7 | 2129.5307 | 709.8436 | 4.88 |
| -9.6 | 2154.1895 | 718.0632 | 4.82 |
| -9.5 | 2179.1338 | 726.3779 | 4.76 |
| -9.4 | 2204.3670 | 734.7890 | 4.71 |
| -9.3 | 2229.8923 | 743.2974 | 4.66 |
| -9.2 | 2255.7133 | 751.9044 | 4.60 |
| -9.1 | 2281.8332 | 760.6111 | 4.55 |
| -9.0 | 2308.2556 | 769.4185 | 4.50 |
| -8.9 | 2334.9839 | 778.3280 | 4.45 |
| -8.8 | 2362.0218 | 787.3406 | 4.40 |
| -8.7 | 2389.3727 | 796.4576 | 4.35 |
| -8.6 | 2417.0403 | 805.6801 | 4.30 |
| -8.5 | 2445.0283 | 815.0094 | 4.25 |
| -8.4 | 2473.3404 | 824.4468 | 4.20 |
| -8.3 | 2501.9804 | 833.9935 | 4.15 |
| -8.2 | 2530.9519 | 843.6506 | 4.10 |
| -8.1 | 2560.2590 | 853.4197 | 4.06 |
| -8.0 | 2589.9054 | 863.3018 | 4.01 |
| -7.9 | 2619.8951 | 873.2984 | 3.96 |
| -7.8 | 2650.2320 | 883.4107 | 3.92 |
| -7.7 | 2680.9203 | 893.6401 | 3.87 |
| -7.6 | 2711.9639 | 903.9880 | 3.83 |
| -7.5 | 2743.3669 | 914.4556 | 3.78 |
| -7.4 | 2775.1336 | 925.0445 | 3.74 |
| -7.3 | 2807.2681 | 935.7560 | 3.70 |
| -7.2 | 2839.7748 | 946.5916 | 3.66 |
| -7.1 | 2872.6578 | 957.5526 | 3.61 |
| -7.0 | 2905.9216 | 968.6405 | 3.57 |
| -6.9 | 2939.5706 | 979.8569 | 3.53 |
| -6.8 | 2973.6092 | 991.2031 | 3.49 |
| -6.7 | 3008.0420 | 1002.6807 | 3.45 |
| -6.6 | 3042.8735 | 1014.2912 | 3.41 |
| -6.5 | 3078.1083 | 1026.0361 | 3.37 |
| -6.4 | 3113.7511 | 1037.9170 | 3.33 |
| -6.3 | 3149.8067 | 1049.9356 | 3.30 |
| -6.2 | 3186.2797 | 1062.0932 | 3.26 |
| -6.1 | 3260.4977 | 1086.8326 | 3.18 |
| -5.9 | 3298.2525 | 1099.4175 | 3.15 |
| -5.8 | 3336.4445 | 1112.1482 | 3.11 |
| -5.7 | 3375.0787 | 1125.0262 | 3.08 |
| -5.6 | 3414.1602 | 1138.0534 | 3.04 |
| -5.5 | 3453.6943 | 1151.2314 | 3.01 |
| -5.4 | 3493.6862 | 1164.5621 | 2.97 |
| -5.3 | 3534.1412 | 1178.0471 | 2.94 |
| -5.2 | 3575.0646 | 1191.6882 | 2.90 |
| -5.1 | 3616.4619 | 1205.4873 | 2.87 |
| -5.0 | 3658.3386 | 1219.4462 | 2.84 |
| -4.9 | 3700.7002 | 1233.5667 | 2.81 |
| -4.8 | 3743.5523 | 1247.8508 | 2.77 |
| -4.7 | 3786.9005 | 1262.3002 | 2.74 |
| -4.6 | 3830.7508 | 1276.9169 | 2.71 |
| -4.5 | 3875.1088 | 1291.7029 | 2.68 |
| -4.4 | 3919.9804 | 1306.6601 | 2.65 |
| -4.3 | 3965.3717 | 1321.7906 | 2.62 |
| -4.2 | 4011.2885 | 1337.0962 | 2.59 |
| -4.1 | 4057.7370 | 1352.5790 | 2.56 |
| -4.0 | 4104.7234 | 1368.2411 | 2.53 |
| -3.9 | 4152.2539 | 1384.0846 | 2.50 |
| -3.8 | 4200.3347 | 1400.1116 | 2.47 |
| -3.7 | 4248.9723 | 1416.3241 | 2.44 |
| -3.6 | 4298.1731 | 1432.7244 | 2.42 |
| -3.5 | 4347.9436 | 1449.3145 | 2.39 |
| -3.4 | 4398.2904 | 1466.0968 | 2.36 |
| -3.3 | 4449.2202 | 1483.0734 | 2.33 |
| -3.2 | 4500.7397 | 1500.2466 | 2.31 |
| -3.1 | 4552.8559 | 1517.6186 | 2.28 |
| -3.0 | 4605.5754 | 1535.1918 | 2.25 |
| -2.9 | 4658.9055 | 1552.9685 | 2.23 |
| -2.8 | 4712.8531 | 1570.9510 | 2.20 |
| -2.7 | 4767.4253 | 1589.1418 | 2.18 |
| -2.6 | 4822.6295 | 1607.5432 | 2.15 |
| -2.5 | 4878.4729 | 1626.1576 | 2.13 |
| -2.4 | 4934.9630 | 1644.9877 | 2.10 |
| -2.3 | 4992.1072 | 1664.0357 | 2.08 |
| -2.2 | 5049.9131 | 1683.3044 | 2.06 |
| -2.1 | 5108.3883 | 1702.7961 | 2.03 |
| -2.0 | 5167.5406 | 1722.5135 | 2.01 |
| -1.9 | 5227.3779 | 1742.4593 | 1.99 |
| -1.8 | 5287.9081 | 1762.6360 | 1.96 |
| -1.7 | 5349.1392 | 1783.0464 | 1.94 |
| -1.6 | 5411.0793 | 1803.6931 | 1.92 |
| -1.5 | 5473.7367 | 1824.5789 | 1.90 |
| -1.4 | 5537.1196 | 1845.7065 | 1.88 |
| -1.3 | 5601.2364 | 1867.0788 | 1.85 |
| -1.2 | 5666.0957 | 1888.6986 | 1.83 |
| -1.1 | 5731.7060 | 1910.5687 | 1.81 |
| -1.0 | 5798.0760 | 1932.6920 | 1.79 |
| -0.9 | 5865.2145 | 1955.0715 | 1.77 |
| -0.8 | 5933.1305 | 1977.7102 | 1.75 |
| -0.7 | 6001.8329 | 2000.6110 | 1.73 |
| -0.6 | 6071.3309 | 2023.7770 | 1.71 |
| -0.5 | 6141.6336 | 2047.2112 | 1.69 |
| -0.4 | 6212.7504 | 2070.9168 | 1.67 |
| -0.3 | 6284.6906 | 2094.8969 | 1.65 |
| -0.2 | 6357.4639 | 2119.1546 | 1.63 |

| | | | |
|---|---|---|---|
| -0.1 | 6431.0799 | 2143.6933 | 1.61 |
| 0.0 | 6505.5483 | 2168.5161 | 1.60 |
| 0.1 | 6580.8790 | 2193.6263 | 1.58 |
| 0.2 | 6657.0819 | 2219.0273 | 1.56 |
| 0.3 | 6734.1673 | 2244.7224 | 1.54 |
| 0.4 | 6812.1453 | 2270.7151 | 1.52 |
| 0.5 | 6891.0262 | 2297.0087 | 1.51 |
| 0.6 | 6970.8206 | 2323.6069 | 1.49 |
| 0.7 | 7051.5389 | 2350.5130 | 1.47 |
| 0.8 | 7133.1918 | 2377.7306 | 1.46 |
| 0.9 | 7215.7903 | 2405.2634 | 1.44 |
| 1.0 | 7299.3452 | 2433.1150 | 1.42 |
| 1.1 | 7383.8676 | 2461.2892 | 1.41 |
| 1.2 | 7469.3688 | 2489.7896 | 1.39 |
| 1.3 | 7555.8600 | 2518.6200 | 1.37 |
| 1.4 | 7643.3528 | 2547.7843 | 1.36 |
| 1.5 | 7731.8586 | 2577.2862 | 1.34 |
| 1.6 | 7821.3893 | 2607.1298 | 1.33 |
| 1.7 | 7911.9567 | 2637.3189 | 1.31 |
| 1.8 | 8003.5729 | 2667.8576 | 1.30 |
| 1.9 | 8096.2499 | 2698.7500 | 1.28 |
| 2.0 | 8190.0000 | 2730.0000 | 1.27 |

# Appendix D

Appendix D provides a Fortran program that calculates an optimal value for the expansion ratio of negative and positive energy values, subject to some constraints (maximum signal levels). The program searches expansion ratios with their corresponding error values up to a maximum value defined by the user. The value that produces the least error is chosen as the optimal value. In this implementation, the tracking mode window is 6 dB for positive errors and at least 3.5 dBs wide for negative errors. The program, however, calculates the expansion window in 6-dB range. Error values are calculated using no-worse windows data. The index value for positive and negative errors correspond to the actual signal level in tenths of dBs.

```
c
c
c   Program to determine the best value for the expansion ratio of negative
c   energy values and that of positive ones.
c
        double precision neg(60), poserr(60), negerr(60)
        double precision sigma(60), maxerr, minerr, bingo
        double precision total(400)
        open  (1,file = 'n1.dat',status = 'new')
c
c   clear all the total values
c
        do 100  n = 1,400
  100   total (n) = 0.

        write(*,1)
    1   format(1x, 'enter positive dbm level')
        read (*,*) dbmpos
        write(*,2)
    2   format(1x, 'enter negative dbm level')
        read (*,*) dbmneg
        write(*,8)
    8   format(1x, 'enter maximum value for N')
        read (*,*) nn
c
c   determine positive errors
c
c   Since the AGC operates in the tracking mode close to the  boundary, more
c   weight must be given to these regions.
c
c   0.0 to 1.0 dB       10 pts
c   1.1 to 2.0 dB       10 pts
c   2.1 to 3.0 dB  5 pts
c   3.1 to 4.0 dB  1 pts
c   4.1 to 5.0 dB  1 pts
c   5.1 to 6.0 dB  1 pts
c
        do 200 i = 1,20
  200   poserr (i) = 2632. * (( 10. ** ( float(i) / 100.)) - 1. )
c 200   write     (1,5) i,poserr(i)
c
        do 201 i = 22,30,2
  201   poserr  (i) = 2632. * (( 10. ** ( float(i) / 100.)) - 1. )
c 201   write (1,5) i,poserr(i)
c
        do 201 i = 22,30,2
  202   poserr (i) = 2632. * (( 10. ** ( float(i) / 100.)) - 1. )
c 202   write(1,5) i,poserr(i)
c
c   determine negative errors
c
c   We do the same thing with the negative errors.
c
        do 300 k = 1,20
  300   negerr (k) = 292.*(1. - (10. ** (float(-k) /100.)))
c 300   write(1,9) k,negerr(k)
    9   format (1x, 'negative error(',i2,') = ',1x,f20.4)
c
        do 301 k = 22,30,2
  301   negerr (k) = 292.*(1. - (10. ** (float(-k) /100.)))
c 301   write(1,9) k,negerr(k)
c
        do 302 k = 40,60,10
  302   negerr (k) = 292.*(1. - (10. ** (float(-k) /100.)))
c 302   write(1,9) k,negerr(k)
c
c   Assuming that the mapping is actually linear, then the following criteria
c   is used to determine the optimum value for N.
c
c
c   total(n) = Sigma [ e  - e  * n ]
c       k   +k -k
c
        do 400 n = 1,nn
        do 400 k = 1,60
        sigma(k) = poserr(k) - float(n) * negerr(k)
  400   total(n) = total(n) + sigma(k)
c
c   Now it is time to determine the minimum value of the error.
c
        do 500 n = 1,nn
        if ( bingo .lt. 0.) goto 504
        if ( total(n) .le. total(n+1)) goto 501
        bingo = total(n+1)
        itr = n+1
        goto 502
  501   bingo = total(n)
        itr = n
  502   if ( n+1 .gt. nn) goto 503
  500   continue
  503   itr = n
        bingo = total(n)
        goto 510
  504   itr = n-1
        bingo = total(n-1)
c
c   Calculate maximum and minimum energy levels
c
        do 600 i = 1,60
  600   write (1,5) i, poserr(i)
    5   format(1x, 'positive error(',i2,') = ',1x,f20.4)
c
        do 601 k = 1,60
        neg(k) = itr * negerr(k)
  601   write (1,6) k, negerr(k), neg(k)
    6   format(1x, 'negative error(',i2,') = ',1x,f20.4,
       +' equivalent to',f20.4)
        maxerr = 2632. * (( 10. ** ( dbmpos / 10. )) - 1.)
```

```
            minerr = 292. * float(itr) * ( 1. - ( 10. ** (dbmneg / 10. )))
            write  (1,3) maxerr
    3       format (1x, ' Maximum energy level is ',f20.4)
            write  (1,4) minerr
    4       format (1x, ' Minimum energy level is ',f20.4)
c
c    Output the N value
c
  510       write(1,7) itr, bingo
    7       format(1x,'N =',i3,' with the corresponding error of ',f20.4)
c
            end
```

# Appendix E

Appendix E

```
;*****************************************************************************
;
;    agc.asm
;

;*****************************************************************************
;
;    front end agc function.
;
;
;*****************************************************************************
;
;    the average signal squared is computed by the main program and stored in
;    avesqr, which is cleared by this routine after using it. the routine uses
;    a window whose width depends on the modulation (1200,2400) and a error
;    weighting, which also depends on that rate. we first set those values:
;
agc:
        sovm
        lac     gn              ; check for afe switching
        bnz     switch
;
;    check if 2400 and change those values
;
        lack    3
        and     status          ; if status bits 0 and 1
2 = 2400
        sub     one,1
        blez    agc0            ; if 2 , do not modify tmp0 and tmp1
;
;    for 2400 , 2 - tmp0 and 1320 - tmp1
;
        lack    2               ; it is 2400
        sacl    tmp0
        lt      one
        mpyk    1320
        pac
        sacl    tmp1
        b       agc1
;
agc0:
        lack    1               ; it is 1200
        sacl    tmp0            ; weighting factor - tmp0
        lt      one
        mpyk    950
        pac
        sacl    tmp1            ; window - tmp1
;
;    now subtract reference from baud energy to get error. the baud energy is
;    in s10.5 format. the AGC maintains that level at 2.86x16 = 46.7  (h'5b6
;    in s10.5). the agcref is therefore h'5b6
;
agc1:
        lac     avesqr
        bgez    cont1           ; for negative energy
        lac     one,15          ; set to max positive
        sub     one             ; energy level - forced
        slew    mode
cont1:
        lt      one
        mpyk    agcref          ; agcref = h'5b6
        spac                    ;avesqr - agcref - acc
;
;    compare the error to window (tmp1).
;    if error  window = error - window - error
;    if -window  error  window = 0 - error
;    if error  -window  = tmp0 x (error + window) - error
;    if the average baud energy is a, the peak baud energy for QAM signals is
;    1.8 a and the minimum is 0.2 a.
;    the window is therefore chosen to be 0.8 a in either direction. with
;    agcref = h'5b6 the window is h'492
;    for dpsk signals the variations in baud energy are entirely due to isi
;    and distortion and therfore the window is much smaller (h'a).
;    first check if error  window
;
        sub     tmp1
        sacl    tmp3            ; error - window - tmp3
        bgez    agc2
;
;    error  window = check if error  -window. in which case zero the error.
;    first zero the error (i.e. assume error  -window) and modify if wrong
;    assumption.
;
        lark    ar1,0
        sar     ar1,tmp3        ; assume error  -window
;
;    check asumption
;
        add     tmp1,1          ; error + window - acc
        bgez    agc2            ; assumption is right
;
;    error  -window = tmp0x(error+window) - tmp3
;
        sacl    tmp3
        lt      tmp3
        mpy     tmp0
        pac
        sacl    tmp3
```

```
;
;    at this point, the weighted windowed error is contained in tmp3. we
;    consider it an s.15 number and use it to update the agc gain alpha. first
;    we determine whether to slew or not. if error is larger than 1EA6h or
;    smaller than F5E7h, go into slewing mode by setting error to 7FFFh or
;    8000h respectively. otherwise leave it unchanged.
;
agc2:
        lac     tmp3
        ldpk    1
        sub     possm
        blez    agc3            ; do not slew
;
        ldpk    0               ; enter slew mode
        lac     one,15
        sub     one
        sacl    tmp3            ; tmp3  7FFFh
        b       agc4
;
agc3:
        add     possm           ; acc  tmp3
        add     negsm
        ldpk    0
        bgez    agc4            ; do not slew
;
        lac     one,15          ; enter slew mode
        add     one
        sacl    tmp3            ; tmp3  8000h
;
;    the following lines update the gain alpha using an exponential integrator
;    alpha = alpha(1-kxerror)   (error = tmp3) where alpha is of format s7.8
;    and error is s0.15 and k = 0.5. alpha * error : s7.8 * s.15 = s7.24. by
;    keeping acch without left shift the multiplication by k is accomplished.
;    alpha is upperbounded to 35.73 in s7.8
;
agc4:
        lack    maxalp
        tblr    tmp0
        zalh    alpha
        lt      tmp3            ; error - t
        mpy     alpha
        spac                    ; alpha (1 - 0.5*error) - acc
        sach    alpha
;
;    check if alpha  max alpha
;
        subh    tmp0
        blez    agc5
;
        lac     tmp0
        sacl    alpha
;
;    zero baud energy register
;
agc5:
        zac
        sacl    avesqr
;
;*****************************************************************
;
;    energy detect loop
;
;*****************************************************************
;
;    start by reading in hysterisis counter increment constant
;
        lack    hysinc
        tblr    tmp5
;
;    check if afe gain is on or off
;
edt:
        lack    080h
        and     sturd
        bz      edt1
;
;    afe gain is on, check if energy detect is on (sturd[6] = 1)
;
        lac     one,6
        and     sturd
        bz      edt01           ; if zero = energy is not
;
;    detected = check if level is larger than -43.5 dbm. if sturd[6] is one,
;    check if level less than -48 dbm
;
        lack    thres1
        tblr    tmp0
        lac     tmp0
        sub     alpha
        blz     edt2            ; if  0 then no energy detect
;
;    check if afe gain stage should be bypassed
;
        lack    thres3
        tblr    tmp0
        lac     tmp0
        sub     alpha
        blz     edt3
;
;    is afe gain on?
;
        lack    080h
        and     sturd
        bz      edt3            ; if gain is off exit
```

```
;
;    bypass afe gain
;
         lack      07fh
         and       sturd
         sacl      sturd
         lack      04h
         sacl      gn
         ret
;
;    decrement hysterisis counter
;
edt2:
         bv        edt21          ; clear overflow bit
edt21:
         zalh      hyst           ; hysteresis counter
         subh      tmp5           ; tmp5 = i927 = 32768/15
         sach      hyst
;
;    in case of overflow declare loss of energy detect
;
         bv        edt02
         ret
edt02:
         lack      0bfh
         and       sturd
         sacl      sturd
         ret
;
;    following lines are executed if afe gain is high but no energy detect.
;    check if alpha  21.28 (i.e. receive level -43.5 dbm) and increment
;    hysteresis counter if it is. otherwise, exit.
;
edt01:
         lack      thres2         ; 21.28 in s7.8
         tblr      tmp0
         lac       tmp0
         sub       alpha
         blz       edt3
;
;    alpha  21.28 = increment hyst. counter
;
         bv        edt011         ; clear overflow bit
edt011:
         zalh      hyst
         addh      tmp5           ; tmp5 contains inc. f0f
         sach      hyst
;
;    detect bit sturd[6] =1.
;
         bv        edt04          ; in case of overflow set energy
         ret
edt04:
         lac       one,6
         or        sturd
         sacl      sturd
         ret
;
;    if afe gain stage is bypassed, check level of alpha
;
edt1:
         lack      thres5
         tblr      tmp0
         lac       tmp0
         sub       alpha
         bgz       edt3
;
;    if alpha  thres5 (20.09 in s7.8) then turn afe gain status word bit on.
;
         lack      080h
         or        sturd
         sacl      sturd
         lack      014h
         sacl      gn
edt3:
         ret

;*****************************************************************
;
;    routine for switching the afe on/off
;
;*****************************************************************
;    zero baud energy register
;
switch:
         zac
         sacl      avesqr
;
         lack      010h           ; mask off unwanted bits
         and       gn
         bz        afeoff
;
;    check if the gain should be on
;
         lack      0fh            ; mask off the afe on bit
         and       gn
         sub       one            ; decrement the counter
         bz        swtch1
;
         sacl      gn             ; save gn value
         lack      010h           ; load the afe on bit
         or        gn
         sacl      gn             ; restore afe on bit
         ret
```

```
;
swtch1:
        sacl    gn              ; reset the gn value to zero
        lack    thres6          ; load the new alpha value
        tblr    alpha           ; reset alpha to 5.05
        ret
;
afeoff :
        lac     gn
        sub     one             ; decrement the counter
        bz      swtch2
;
        sacl    gn
        ret
;
swtch2:
        sacl    gn
        lack    thres4          ; load new alpha value
        tblr    alpha           ; reset alpha to 8.98 in s7.8
        ret
;
;***************************************************************************
*
;
;    RECEIVER PER SAMPLE PROCEDURE
;
rstskf:
        lac     rin             ; input 2's complement sample
;
;    high pass filter the incoming signal to remove the dc component.
;
        ldpk    1
        sacl    x1
        zals    stlsb
        addh    st
        sub     st,tau
        addh    x1
        sub     x1,tau-1
        subh    x2
        add     x2,tau-1
        sacl    stlsb
        sach    st
        dmov    x1
        ldpk    0
        sach    tmp1
;
;    multiply input sample by agc gain alpha. the output format is s4.11
;    requiring some manipulations
;
        lt      tmp1            ; multiply by agc word
        mpy     alpha
        pac
;
;    shift accumulator eight four times before storing
;
        sacl    tmp0
        sach    tmp1
        lac     tmp0,8
        sach    tmp0
        lac     one,8           ; mask off any sign extension
        sub     one             ; 00ff - acc
        and     tmp0
        sacl    tmp0
        lac     tmp1,8
        add     tmp0
        sacl    tmp1
;
; update the signal power estimate avesqr. avesqr = avesqr + (tmp1)2
; avesqr is zeroed by the agc routine once per baud.
;
        lac     tmp1,15
        sach    tmp0            ; tmp0 in s5.10
        lt      tmp0
        mpy     tmp1
        pac
        addh    avesqr          ; avesqr in s10.5
        sach avesqr
        .
        .
        .
;
;    end of receiver per sample task
;
```

# Part IV. Telecommunications

# General-Purpose Tone Decoding and DTMF Detection

**Craig Marven**

**Regional Technology Center — Bedford, England**
**Texas Instruments**

# Introduction

The use of the Dual-Tone Multi-Frequency (DTMF) signaling scheme within telecommunications systems has become widespread over the past few years. It is replacing the older type of pulse oriented dialing methods in telephones worldwide, and also finds application in a number of other equipment types, such as personal computer (PC) telephone peripherals, remote signaling schemes etc.

In parallel with the universal DTMF standard, the various telecommunications companies or public authorities (PTTs) around the world use a number of different tones to signal call progress parameters. Examples include busy tones, number unobtainable, timing tones, etc. Although DTMF operates to an internationally recognized standard, these additional tones do not. Therefore, there is often a need for a programmable tone detection capability operating concurrently with standard format DTMF decoding. Alternatively, there are also many possible areas of application for an expanded programmable tone decoding facility without DTMF capability.

This document describes a single-chip solution to fulfill the requirement for concurrent DTMF and general-tone decoding or expanded, general-tone decoding only. These facilities are provided by a special program on the TMS320C17 or TMS320E17 first-generation digital signal processor (DSP). The term TMS320C17 should be taken to apply to both the TMS320C17 and TMS320E17 for the remainder of this report. See Reference [6] for full information on these devices.

The TMS320C17 is particularly suited to tone detection as it possesses on-chip serial ports, a hardware multiplier and a 200 nanosecond (ns) instruction cycle time. These last two features allow high-speed calculation of the digital filter equations which implement the core of the tone decoding function.

The main functions of the tone detector described in this report are as follows:

1. DTMF tone decoding to international standards
2. Power measurement at six selectable frequencies in the band 300-3400 Hz
3. Power measurement at three selectable frequencies simultaneously with DTMF tone decoding
4. Selectable bandwidth and resolution of frequency selection
5. Timestamping of tone arrival and departure
6. Selectable thresholds to define tone arrival and departure
7. Interrupt generation on tone arrival, departure or change
8. Interrupt generation on unidentified tone
9. Interrupt generation on validation of DTMF digits

10. Variable gain setting on input to receivers

11. Self test

In addition to a detailed description of the operation of the software within the TMS320C17, a complete solution to a tone detection peripheral for an IBM XT or AT compatible PC is presented. Remember that this is just one possible application for the tone detection TMS320C17, it could equally be paired with any other host CPU.

This report is divided into seven sections and three appendices. A brief outline of the contents of each section serves as a useful guide. Although some sections refer to general principles of DTMF and tone decoding, keep in mind that the primary objective is to discuss a particular implementation of a tone detector.

**Theory of Operation**

Describes the basic theory of operation of the tone detector, describing total system scope and functionality, and giving a brief introductory description of each functional block. For this purpose the tone detector is considered as a set of software functions with supporting hardware. The high suitability of the TMS320C17 DSP for tone detection is also discussed.

**Implementation**

Deals in detail with the implementation of both the software within the TMS320C17, and its supporting hardware. Each is split into its main functional blocks and then further subdivided into individual tasks. The description of software implementation is accompanied by a series of flow charts, allowing the reader to follow the description from the top functional level right down to the detail of individual tone detector features. This section also covers in detail how the tone detector program controls, and benefits from, some of the resources provided by the TMS320C17.

**Host Interface**

Describes the host interface of the tone detector. This has been designed for easy connectability to a variety of host CPUs, and is essentially a single physical 8-bit read/write register. The host interface software is implemented by an interrupt routine in the TMS320C17, allowing host access at any time as required.

**Applications and Customization**

Briefly outlines some possible applications for the tone detector including traditional telephony applications along with some innovative approaches. These include a method for secure off-site remote control of equipment via telephone lines, a tester for telephone equipment, etc. For many applications it may be necessary to customize the program to some extent. A number of examples of this are discussed.

## Conclusion

Within the appendices are a full listing of the source code for the tone detector in COFF (common object file format) source format, and a demonstration program for IBM or compatible PCs. This program is written in Turbo Pascal and is for use with the design example included in this report.

## History of DTMF

There are two standard dialing conventions used in telephone systems throughout the world. The most common, and by far the oldest is known as pulse or loop-disconnect dialing. DTMF is a relatively new all-electronic method which is rapidly replacing the older electro-mechanical system. Figure 1 represents a highly simplified pulse dialing telephone terminal. There are other circuits required to make a practical telephone, but this diagram serves to illustrate several key points.

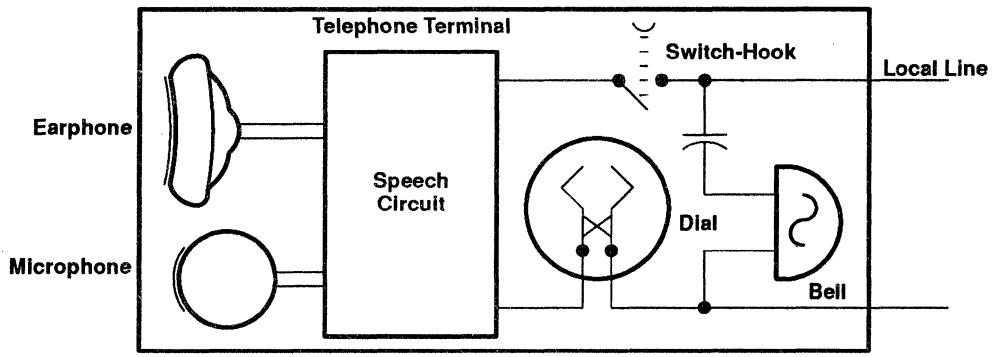



**Figure 1. Pulse Dialing Telephone**

When the receiver of a pulse dialing telephone is lifted, the hook-switch closes and a DC loop current of a few milliamperes flows from the central office or local exchange. The dial is arranged so that the switch within it opens and closes as it returns to its rest position. When the switch opens it causes the loop current to be interrupted, hence the alternative name of loop-disconnect dialing. The dial is arranged so that one disconnect period or pulse is created for the digit 1, two for the digit 2, up to ten pulses for the digit 0.

Dial pulses originally operated electromechanical switching systems, and still do in many countries. These systems have an upper limit of about ten operations per second and pulse dialing systems therefore produce pulses of a 100 millisecond (ms) duration. Nominal operation in the U.S. gives a break period of 61 ms and a make period of 39 ms. This is different from other countries which use a 2:1 ratio (67 ms break, 33 ms make). An inter-digit pause is indicated by an absence of pulses of nominally 700 ms for U.S. systems, or as short as 200 ms in other countries.

The time required to send the dial pulses needed for one digit can be up to 1.7 seconds (ten pulses for the digit 0 and a 700 ms inter-digit pause) which can make the dialing of a long international number very time consuming. For example, the international number (from the U.S.) for Texas Instruments in Bedford, England is:

0 1 1  4 4  2 3 4  2 7 0 1 1 1

This would take 15.1 seconds to dial with a U.S. pulse dialing system. It is not difficult to see why the method is now regarded as out-dated.

In order to reduce costs, increase reliability, and improve service, the electromechanical switching systems used at central offices or local exchanges are being replaced with fully electronic systems. In most advanced countries this upgrading process is virtually complete. With the new equipment it is no longer necessary to have a slow dialing mechanism to accommodate the response time of the old switching mechanisms. A new dialing scheme thus becomes possible using purely electronic means. The DTMF system has been adopted as the universal standard through the CCITT (Comite Consultatif International de Telephonie et de Telegraphie) which is a committee of the International Telecommunication Union (ITU), now part of the United Nations.

## The Use and Characteristics of DTMF

The full name for DTMF is Dual-Tone Multi-Frequency which describes its operating characteristics very well. Consider that a telephone is equipped with a keypad as shown in Figure 2, instead of a dial. The A,B,C and D keys are usually not present, but are part of the full CCITT specification and can be decoded by the programmed TMS320C17 used here.

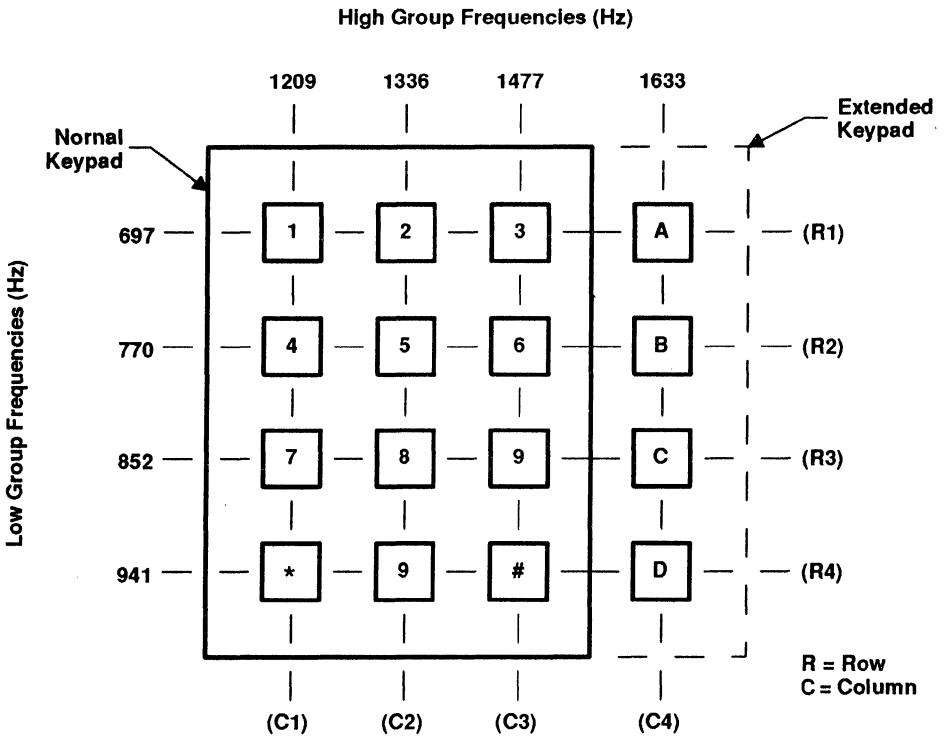



**Figure 2. DTMF Keypad**

Pressing any key causes an electronic circuit to generate a tone which is a summation of the two individual frequencies related to the row and column of that key.

The frequencies used in DTMF dialing have been carefully selected so that any DTMF decoding circuit will not confuse them with other tones that may occur on the line. As the tone generation does not involve a disconnect of the telephone circuit, DTMF tones may be sent down the line during a call just by pressing any key on the keypad. When this method is used as a form of low speed data transmission, it is important that speech is not accidentally interpreted as a DTMF tone. In order to reduce the risk of this happening, tones must be present continuously for a minimum period of about 50 ms, with an interdigit pause of similar length.

With a minimum dialing time of 100 ms per digit, irrespective of its value, our previous example number would take 1.4 seconds to dial. This represents a saving of 13.7 seconds or 91% of the time taken by a pulse dialer. Additional advantages of DTMF dialing include the use of solid-state electronic circuits and compatibility with electronically controlled exchanges.

## Theory of Operation

This section briefly describes the operation of the tone detection system presented in this report. A functional block diagram for the complete system is shown in Figure 3.

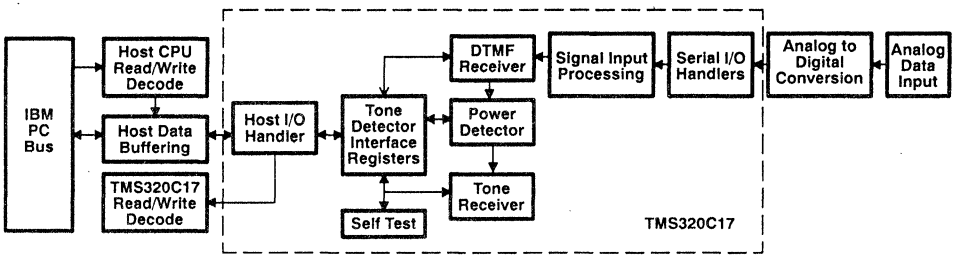



**Figure 3. Tone Detector Functional Block Diagram**

As is clear from examination of Figure 3, the tone detector may be viewed as comprising a set of software routines within the TMS320C17, plus associated external hardware to provide interfaces between the TMS320C17 and both the incoming analog signal and a host CPU.

The following paragraphs briefly describe the major software and hardware features of the tone detection system, and some of the features of the TMS320C17 which are of special benefit to this application.

### Software

The tone detection system described in this report comprises six groups of functions within the TMS320C17. These provide a powerful tone detection capability for either DTMF decoding, general tone identification or a combination of both. These six functional groups are as follows:

1. Input signal processing
2. DTMF receiver
3. Power (envelope) detector
4. Tone receiver - comprising five sub-sections
5. I/O routines (Interrupt Handler)
6. Self test

Figure 4 shows how the first four of these functions interrelate during normal operation of the the tone detector. Each block within Figure 4 is explained in detail in the Implementation section and each also has a detailed flowchart associated with it. The number of the figure for the associated detailed flow chart is shown inside each block in Figure 4.

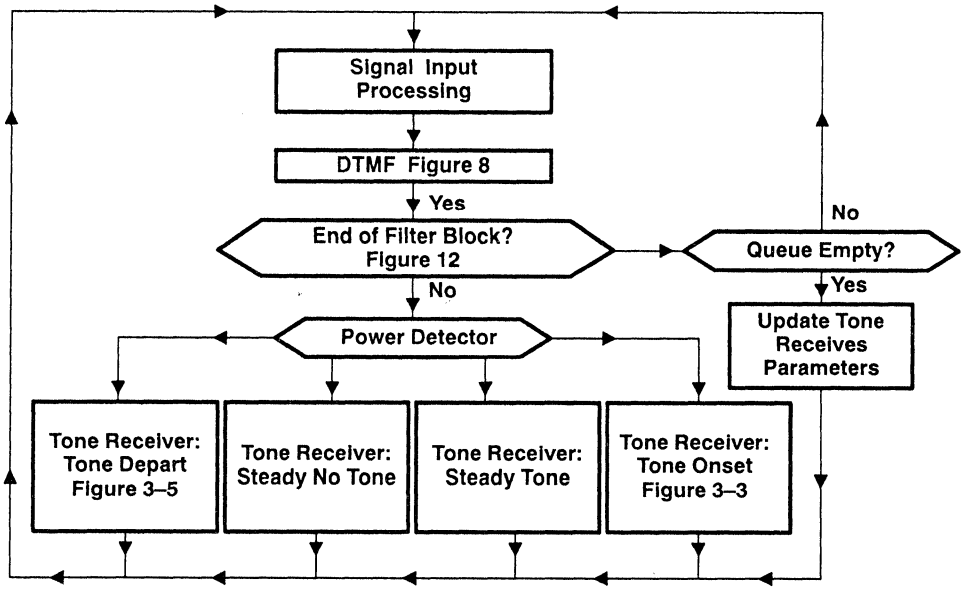



**Figure 4. Tone Detector Flow Chart - Top Level**

Program execution remains within the flow shown in Figure 4 unless interrupted by either an I/O request or a self-test command, which are independent functions. Self-test is merely a special case of a host CPU I/O request. Both serial I/O and host CPU I/O cause an interrupt to the TMS320C17 and therefore function outside the normal program flow. Self-test additionally destroys all temporary data storage, leaving the tone detector in the same state as after a hardware reset (see the Register Read Functions section). The following sections briefly describe the relationship betwen the above six functional groups. A more detailed description of the operation of each is contained in the Implementation section.

### *Input Signal Processing*

This ensures that the incoming data samples are within the optimum working range of the tone detector. Software limiting of the incoming signal is applied if it exceeds the maximum signal input level (see the Signal Input Processing section). Program control passes to the DTMF receiver if it is enabled, otherwise control passes to the power detector.

### *DTMF Receiver*

Using the signaling plan outlined in CEPT (Conference Europeenne des Administrations des Postes et des Telecommunications) recommendations T/CS 46-02, the DTMF receiver validates and decodes DTMF tone pairs against a template of acceptable frequency deviation. The DTMF receiver may be enabled or disabled under software control by the host CPU. Once the operation of the DTMF receiver is complete, program flow passes to the power detector.

### *Power (Envelope) Detector*

The power detector performs a simple smoothing operation on the incoming signal and, using thresholds programmed by the user, directs program flow among one of the four possible tone receiver flow paths shown in Figure 4:

1. Tone onset
2. Tone depart
3. Steady no tone
4. Steady tone

Separate threshold levels may be programmed for detection of the onset and departure of the input signal.

### *Tone Receiver Power Level Determination*

The tone receiver determines the overall power level of the incoming signal and the individual power level at up to six selectable frequencies. In addition, it validates the signal onset or signal departure indication from the envelope detector to change the tone arrival or tone departure status bits (see Status section). The tone receiver operates independently of the DTMF receiver and provides programmable center frequency, bandwidth, resolution and thresholds for the recognition of general tones in the band 300 Hz to 3400 Hz (e.g., call progress tones).

When the DTMF receiver is disabled the tone receiver monitors six programmable frequencies in the range 300-3400 Hz and reports the power levels received at each of those frequencies. When the DTMF receiver is enabled the tone receiver monitors only three frequencies. The power level of the three unused frequencies is registered as zero. The tone receiver also has an additional power measurement which reports the received power across the telephony band of 300-3400 Hz allowing the system to detect the presence of frequencies outside those programmed individually.

When the tone receiver is enabled, filtering begins upon the recognition of a tone by the envelope detector. The host may be interrupted at the end of the first block of filtering as a result of the tone arrival bit in the status register being set. At this time level information for the new tone is available at each of the search frequencies. The host may also be interrupted by tone departure. The tone receiver is also able to detect any change in signal content and may optionally generate an interrupt as a result. Host interrupt is described in detail in Host Interrupt section.

The flow of program execution around the tone receiver is dependent upon the results of tests at a number of points. The most important of these is at the output of the power detector. As mentioned above there are four possible conditions the power detector can indicate:

1. Tone onset
2. Tone depart
3. Steady no tone
4. Steady tone

The operations performed within these blocks are described in detail in Software Implementation section.

The second most important decision point in the tone receiver program flow is represented by the end of filter block test. When the tone receiver is enabled, incoming samples are filtered in blocks. The block size is dependent upon the value written to the filter length register (see Filter Length section). If a filtering block has been completed, housekeeping functions must be performed.

***I/O Handler (Serial and Parallel)***

Any external I/O access will cause an interrupt to the TMS320C17. External I/O can come from one of three possible sources:

- A new data sample being input from the serial port
- A host CPU write access
- A host CPU read access

The source of the interrupt is checked by the program and control passed to the appropriate portion of the interrupt handler code. A comprehensive discussion on the use of interrupts within the tone detector is given in Hardware Implementation section, including a detailed examination of some parts of the interrupt handler code.

### *Self-Test*

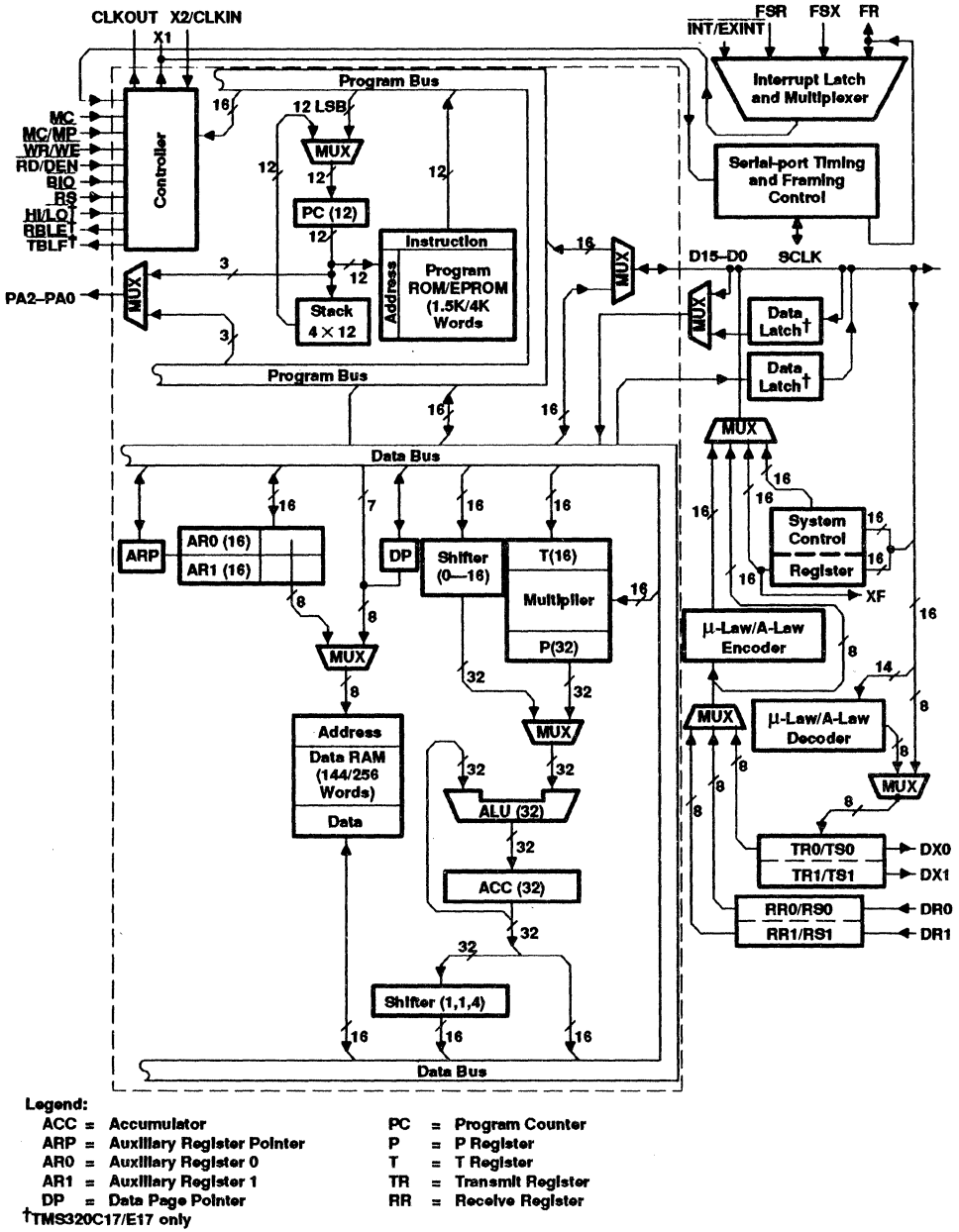
One special case of a host CPU write access is a self-test request. The TMS320C17 responds to this by immediately performing a ROM checksum test, a RAM data test and a codec interrupt check. After these have been performed the host CPU may release the TMS320C17 from self-test mode. The TMS320C17 is then left in a state similar to that after a hardware reset (see Register Read Functions section).

## TMS320C17 Features

The TMS320 family utilizes a modified Harvard architecture for speed and flexibility. In a strict Harvard architecture, program and data memory lie in two separate spaces, permitting a full overlap of instruction fetch and execution. The TMS320 family's modification allows transfers between program and data spaces. This permits coefficients stored in program ROM to be read into RAM, eliminating the need for a separate coefficient ROM. It also makes available immediate operand instructions and subroutine calls to computed addresses.

The TMS320C17 provides all the basic features of the industry-standard TMS320C10. Two serial ports, expanded data memory to 256 words, expanded program memory to 4K words on-chip, and a coprocessor mode are added to provide a powerful processor for a variety of communications-oriented applications. The TMS320C17 is a microcomputer device only, with no external program memory facility. The TMS320E17, a 4K-word EPROM version of the TMS320C17 is available for prototyping or low volume production.

The Tone Detection application takes advantage of the full set of processor resources shown in Figure 5. A few examples from the code, and a description of each, are given in Utilization of TMS320C17 Resources section to illustrate this.




**Legend:**

| | | | |
|---|---|---|---|
| ACC | = Accumulator | PC | = Program Counter |
| ARP | = Auxiliary Register Pointer | P | = P Register |
| AR0 | = Auxiliary Register 0 | T | = T Register |
| AR1 | = Auxiliary Register 1 | TR | = Transmit Register |
| DP | = Data Page Pointer | RR | = Receive Register |

†TMS320C17/E17 only

**Figure 5. TMS320C17/E17 Block Diagram**

The Tone Detector program uses less than 50% of the available 4K-words of program memory and less than 70% of the available 256 words of data memory within the TMS320C17. Of the 174 words of data memory used, 75 are in page 0, and 99 in page 1. A detailed list of program and data memory utilization is shown in Table 1.

**Table 1. Program and Data Memory Utilization**

| Routine | Code Listing Page | Description | Program Memory Locations | Data Memory Locations |
|---|---|---|---|---|
| | 489 | Reset and interrupt vectors | 4 | |
| | 490 | DTMF Constants and filter coefficients | 28 | |
| | 490 | Tone detector constants | 62 | |
| | 491 | Tone detector filter coefficients | 129 | |
| MAIN | 492 | Read sample from input queue and update current time, scale the input sample and call DTMF if it is switched on. | 55 | 14 |
| ENVDET | 494 | Detect changes in signal envelope relative to user-programmed upper and lower thresholds | 41 | 4 |
| TONSET | 495 | Handle occurrence of tone onset | 11 | 2 |
| TDEPT | 495 | Handle tone departure | 41 | 1 |
| $FILTER_4$ | 496 | Routine for filtering and accumulating the input samples | 172 | 52 |
| LEVCAL | 499 | Calculates the levels at the end of each block of filtering | 109 | 1 |
| CHNGS | 501 | Check for level changes during a toneburst | 31 | 0 |
| LVLS | 501 | Write levels into registers | 13 | 3 |
| COMPLT | 502 | Complete operations ready for next filtering operation | 39 | 1 |
| RSTFIL | 502 | Clear down filter accumulators and reset pointers ready for another filter operation | 61 | |
| SQRT | 504 | Generates the square root of an integer | 32 | |
| DTMF | 504 | Detect DTMF digits | 508 | 83 |
| INTHDL | 510 | Interrupt handler | 194 | 8 |
| CRESET | 514 | Cold reset handler | 14 | 1 |
| WRESET | 514 | Warm reset handler | 33 | 3 |
| ATTEN | 515 | Write out status to draw attention to change in one or more of the status bits | 4 | |
| XFUPD | 515 | Update the XF flag | 17 | |
| SLFTST | 516 | Self test of processor | 111 | 4 |
| Total | | | 1709 | 177 |

## Hardware

In order for the TMS320C17 to receive its input signal and communicate with a host CPU it requires a small amount of support circuitry. This comprises just three devices, as shown in Figure 6. This example is specifically for interfacing the tone detection system to an IBM XT or AT compatible PC bus. A detailed description of this circuit is given in Hardware Implementation section.

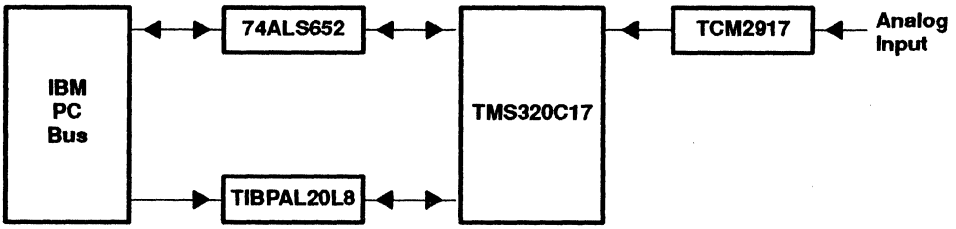



**Figure 6. PC Tone Detector Circuit Diagram—Block Level**

### *Analog to Digital Conversion*

The analog signal is converted to a serial pulse code modulated (PCM) serial data stream by an industry standard combined codec and line filter (COMBO), the TCM2917. This interfaces directly to the TMS320C17 with no support circuitry.

### *Host Interface*

A programmable logic array (PAL) provides read and write decoding for both the host CPU and the TMS320C17, including full address decoding of the host CPU bus. A 74ALS652 provides a two way latched data buffer between the host CPU and the TMS320C17. The TMS320C17 has a special coprocessor mode which can also perform the latched data buffer function in a wide variety of applications. The coprocessor mode is described in greater detail in Use of Coprocessor Port for Parallel I/O section.

## Implementation

This section describes in greater detail how the tone detector functions described in the Theory of Operation section are implemented. It is intended for non-mathematical readers, and equations have only been included where they can aid understanding for readers familiar with general DSP techniques. It is not necessary to understand the derivation or purpose of these equations in order to gain a basic understanding of system operation.

## Software Implementation

As described in Software section the software within the tone detector may be conveniently split into the following six groups:

1. Signal input processing
2. DTMF receiver
3. Power detector
4. Tone receiver comprising five sub-sections
5. I/O routines (Interrupt Handler)
6. Self test

A detailed description of the performance and implementation of these functions follows.

In all of the detailed explanations in this section of the report, references are provided to a page of the program listing included as Appendix A.

### *Signal Input Processing*

This block contains only two straightforward tasks:

1. Read queue, increment time (program listing page 492)—Codec samples sent to the TMS320C17 are received via its serial port and then queued. The maximum queue length is eight samples. Under normal circumstances the queue will not contain more than one sample. However, at the end of each block of filtering or DTMF detection, there is a series of computations which must be completed before the handling of the next codec sample. Operation of both the DTMF code and the tone filtering code are suspended during this period and new codec samples accumulate on the queue. At all times, information arriving at the TMS320C17 via its serial port is handled with first priority, so that no samples or requests are missed.

2. Scale and limit (program listing page 492)—In this report the TMS320C17 is programmed to accept A-law input samples. The TMS320C17 can also be programmed to accept the u-law samples in North American applications. The output from the on-board compander is scaled to a number range which affords the maximum precision for the range of signal magnitudes allowed. The tone receiver is specified to provide linear detection of tones in three ranges. The dynamic range of the tone receiver is between 35 and 40 decibels (dB). Provision of three software selectable scale factors allows this dynamic range to be shifted so that the top of the range is at either +2, −10 or −22 dBmO. Where dBmO is defined as the zero reference point of the channel. The overall detection range is thus +2 to −60 dB approximately (see Figure 7).

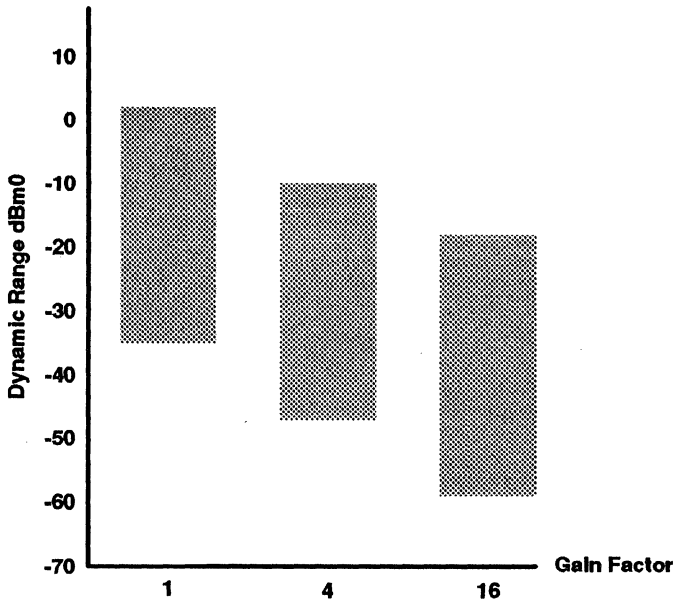



**Figure 7. Tone Detector Active Dynamic Range vs Gain Factor**

The output from this block is the next sample to be dealt with by the DTMF code and the power detector.

### *DTMF Receiver*

A brief specification is given in Table 2. For full details, refer to CEPT recommendation T/CS 46-02. The operation of the TMS320C17 algorithm to this specification has been verified by use of the standard Mitel DTMF test tape.

**Table 2. DTMF Decoder Specification**

| Measurement | Breakdown | Value |
|---|---|---|
| Signal frequencies | Low Group<br>High Group | 697, 770, 852, 941 Hz<br>1209, 1336, 1477, 1633 Hz |
| Frequency deviation for correct operation | | ≤1.9% |
| Power levels per frequency | Operation<br><br>Non-operation | (−6 dBm0 − G dB) to<br>(−36 dBm0 − G dB)*<br>−45 dBm0 − G dB* |
| Power level difference between frequencies for operation | | 0 dB to 10 dB |
| Tone duration | Recognition<br>Non-Recognition | ≥40 mS<br>≤20 mS |
| Silence duration | Recognition<br>Non-Recognition | ≥40 mS<br>≤20 mS |
| Signal to noise ratio required for correct operation | | 12 dB |
| Talk-off performance | | 15 hits in 30 minutes of condensed speech |

*See Mode subsection in Host Interface section for an explanation of the gain control factor GdB.

The DTMF receiver may be used to receive and recognize tones from a remote handset, e.g. in a PABX, or from a telephone set at a remote point on the public telephone network. The distortion of tones over the public network is often severe; for example, the attenuation of the signal from the remote transmitter could vary from 0 dB to 30 dB or more. The specification shown in Table 2 provides correct operation across the normal range of signals received over the public network.

The range of received signal levels at which the DTMF receiver will correctly decode signals can be varied by altering the gain of the tone detector module under software control (see Mode section).

Validation of a DTMF digit while the DTMF receiver is enabled (see Mode section) causes a DTMF interrupt to be generated and suppresses the generation of any short tone interrupt which might otherwise have been generated by the tone receiver code. The arrival time of the tone is stored for the host to read if required.

The following description of the operation of the DTMF block relates directly to the detailed flow chart shown in Figure 8.

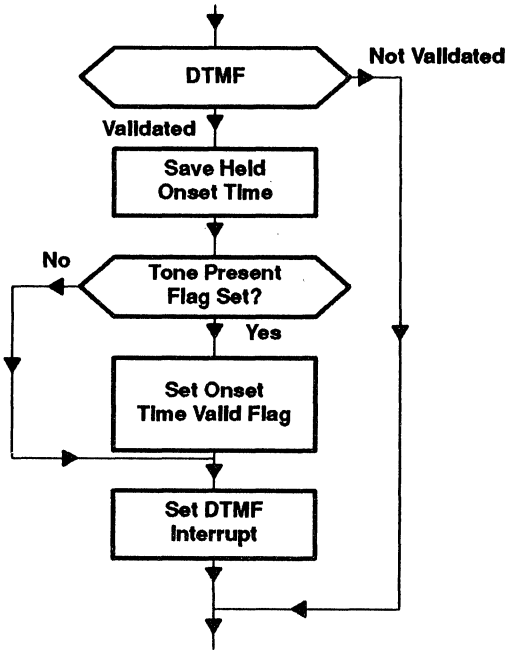



**Figure 8. DTMF Receiver Flow Chart**

DTMF (program listing page 508)—This revolves around a set of eighth order narrow bandpass filters at each of the individual tone frequencies which may be combined to produce a DTMF digit.

The simple eighth order filtering process is executed on the incoming sample automatically when the DTMF receiver is enabled. If a valid DTMF digit is found, its value is stored in the DTMF digit register and execution passes along the 'validated' path. If the DTMF receiver is not enabled, program execution passes onto the tone receiver.

Save Held Onset Time—The onset time of all detected signals is saved in a holding register. This is transferred to the tone arrival register only if the tone receiver is not already indicating the presence of a tone, in which case the tone arrival register will already have been loaded.

Set Onset Time Valid Flag, Set DTMF Interrupt—The DTMF tone onset time is saved in a register for the host to read. The host is informed by interrupt (if implemented) that a tone onset has occurred and that timer registers containing information about the tone are available to be read.

### *Power (Envelope) Detector (see program listing page 494)*

As described above the power detector performs an envelope detection operation on the incoming signal, and directs flow to one of four tone receiver paths.

The smoothing filter applied to the incoming signal has the form:

$$ENVEL = \frac{((2^{15} \times ENVEL) + ABS(32 \times EDF \times SAMPLE) - (32 \times EDF \times ENVEL))}{2^{15}}$$

Where EDF is the user programmed envelope decay factor (see Envelope Decay Factor section). This is equivalent to:

ENVEL = ((1−k) × ENVEL) + (k × ABS(SAMPLE)) where EDF is k X $2^{10}$ where EDF is k × $2^{10}$ k positive.

The envelope decay factor may be programmed to provide a range of time constants for the envelope detector. There is generally a trade-off between the rejection of a glitch if a long time constant is used and increased accuracy of time-stamping with a short time constant.

When the power detector identifies the departure of the input signal, a status register bit (see Status section) may be set, and the time of departure written into a register. This depends upon the signal having been recognized as a DTMF digit or a valid tone within the tone receiver search bands.

Due to the method of implementation of the envelope detector, it should be kept in mind that there are two areas of operation when using the tone receiver: the arrival and departure time skew and the sampling frequency. These are explained in detail in Appendix C.

### *Tone Receiver Band Pass Filter Generation*

The tone receiver generates a band pass filter for each of the chosen frequencies and uses these filters to select the desired frequencies from the incoming signal. The steepness of cut-off of each bandpass filter is defined by the length of time over which the received signal is filtered. This is programmed via a register and applies to all the filters in operation. The passband width of each filter is specified via a separate register, and the maximum value for passband width for any single filter is 492 Hz. Each of the filters in use may be selected to adopt either the passband width specified in the register (wide filter) or a passband width of zero (narrow filter).

As described in the Tone Receiver Power Level Determination section, the power detector directs the flow of the tone receiver along one of four paths:

1. Tone onset
2. Tone departure
3. Steady tone
4. Steady no tone

A detailed description of the operation of each of these follows.

*Tone Onset*

Figure 9 shows the flow chart associated with a tone onset indication from the power detector.

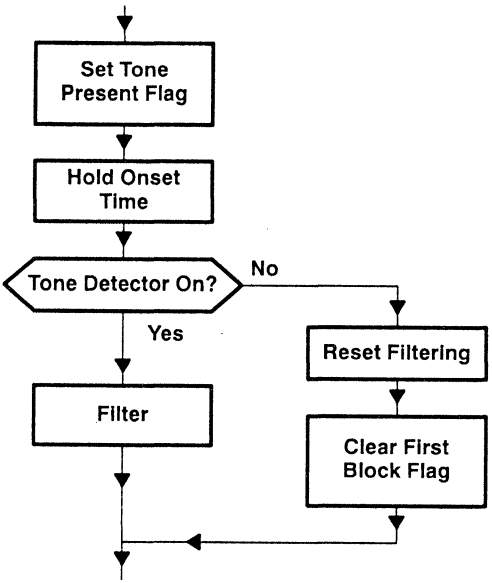



**Figure 9. Power Detector Flow Chart—Tone Onset**

Set Tone Present Flag—This flag is used to indicate the presence or absence of a tone on the line.

Hold Onset Time—The onset time of all detected signals is saved in a holding register.

Filter (program listing page 496)—This routine is the heart of the tone receiver algorithm. The FIR filters are of the lowpass type and there is one for each of the six search frequencies. A range of filter lengths may be specified, from 61 to 1025 samples, allowing filters of extremely steep cut-off to be implemented. With the maximum filter length of 1025 samples, the shortest quantifiable tone is one of at least 128 ms duration. The input signal is demodulated using a sine and cosine wave at each of the six search frequencies. The result of the demodulation is that any signal present at one of the search frequencies is transposed into the passband of the lowpass filter. Figure 10 shows the filter structure.

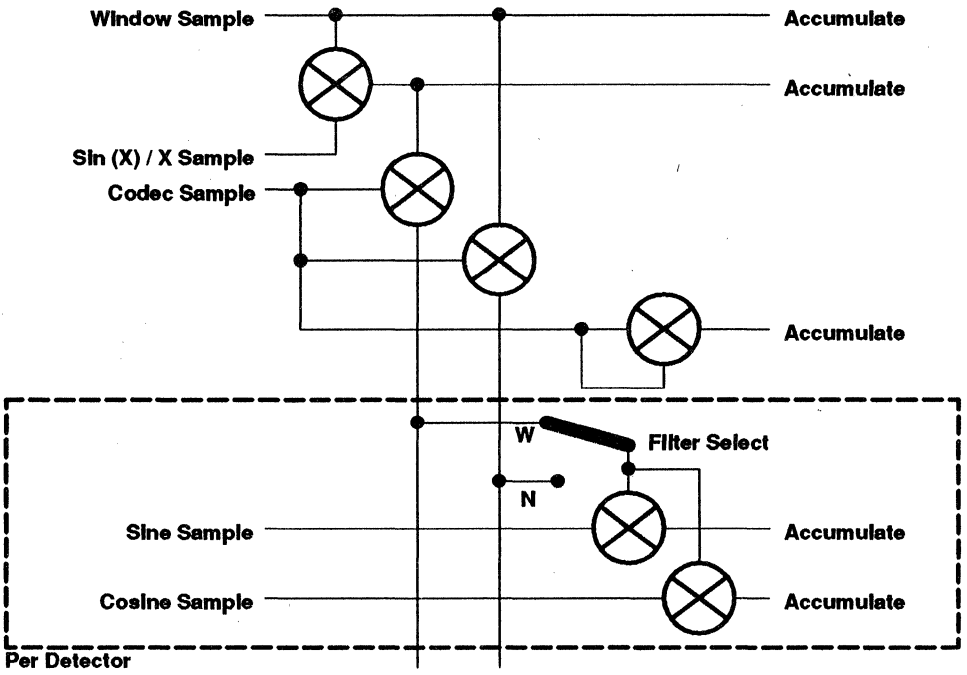



**Figure 10. FIR Filter Structure**

The coefficients of the filter are samples taken from a window function stored in ROM. The function is a Kaiser window, chosen to give the narrowest lowpass response with the given stopband rejection. Where a wide filter response is specified, each filter coefficient is multiplied by a sample of a $\sin(\times)/\times$ function to provide a second wide filter coefficient. This has the effect of widening the filter passband in a definable and convenient manner. The input sample is multiplied by the normal (narrow) and wide filter coefficients to produce both a narrow and wide intermediate sample. Each of the six filters is specified to be either narrow or wide according to the value in the filter select register. Depending on this value, the appropriate intermediate sample is multiplied by a sine sample and cosine sample at the required search frequency. The sine and cosine samples are generated as required by a special routine. The twelve products are separately accumulated to 32-bit accuracy.

In addition to this, accumulations are kept of the wide and narrow filter coefficients so that the filter accumulations can later be normalized. An accumulation is also kept of the square of the input sample, so that the total signal level in the telephony band can be calculated.

Reset Filtering—Clears down all the accumulators and registers used by the filters.

Clear First Block Flag—Clears a flag set to indicate that the first block of data was being filtered.

*Tone Depart*

Figure 11 shows the flow chart associated with a tone departure indication from the power detector.

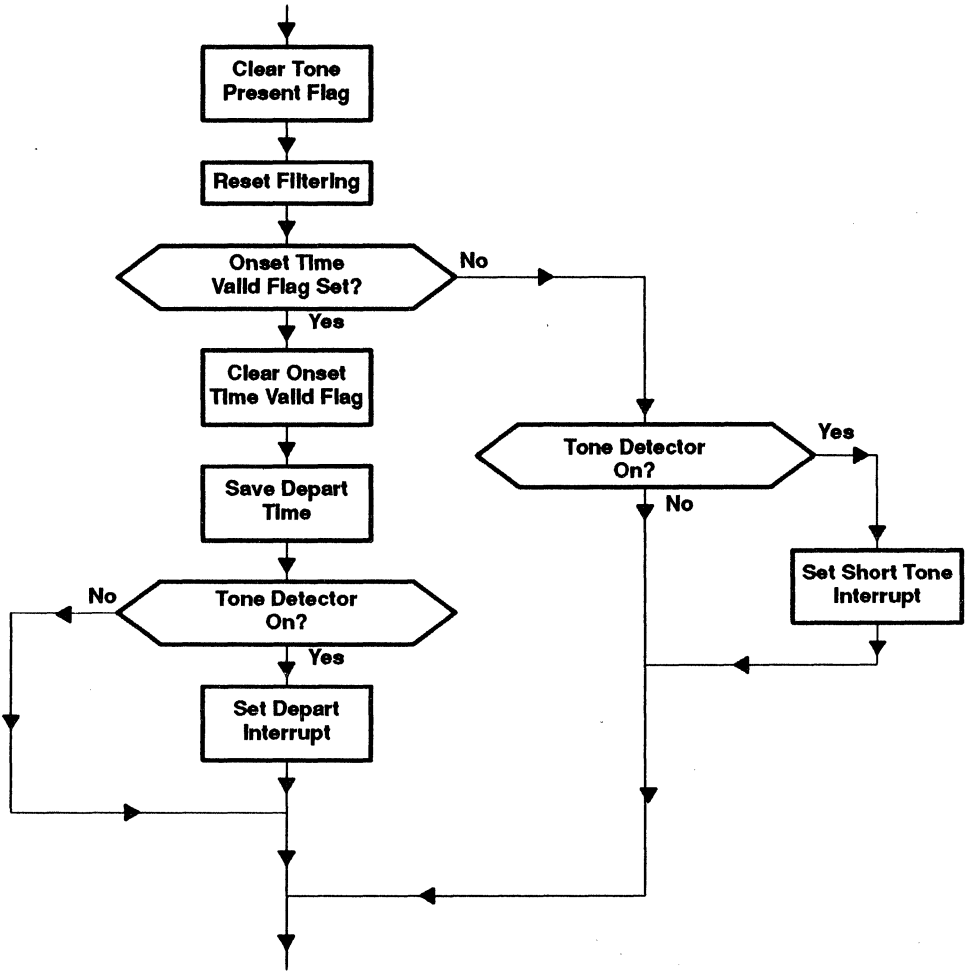



**Figure 11. Power Detector Flow Chart—Tone Departure**

Clear Tone Present Flag—This flag is used to indicate the presence or absence of a tone on the line.

Reset Filtering—Clears down all the accumulators and registers used by the filters.

Onset Time Valid Flag Set?—The program tests to see if a flag has been set at this point to indicate that the stored onset time is valid. This will be the case only if a complete block of filtering has been performed on the tone, or the tone has been recognized as a DTMF digit. If the flag is not set the program further checks to see if the tone detector is enabled. If not this section terminates. Timer registers are not updated and contain onset and departure times for the previous valid tone or digit. However, the current time register is available for the host to read if it wishes to timestamp the short tone. If the tone detector is on, the short tone bit in the status register is set which can optionally generate an interrupt (see Status section).

Clear Onset Time Valid Flag—Clears the above flag.

Save Depart Time—Provided that a valid tone or digit has been recognized, the current time is saved directly into the tone departure register.

Set Depart Interrupt—If the tone detector is enabled, the tone depart bit in the status register is set. This may optionally generate an interrupt.

*Steady No Tone*

In this case, the only operation performed is Reset Filtering which clears down all the accumulators and registers used by the filters.

*Steady Tone*

This condition causes execution from just above the "Tone Detector On" decision point in the tone onset flow chart (Figure 9).

*End of Filter Block?*

When the tone receiver is enabled, incoming samples are filtered in blocks. The number of samples in a block is set by the filter length selected, and may be between 61 and 1025 samples. After each complete block of filtering, much housekeeping must be done. Figure 12 shows the flow chart for this process.

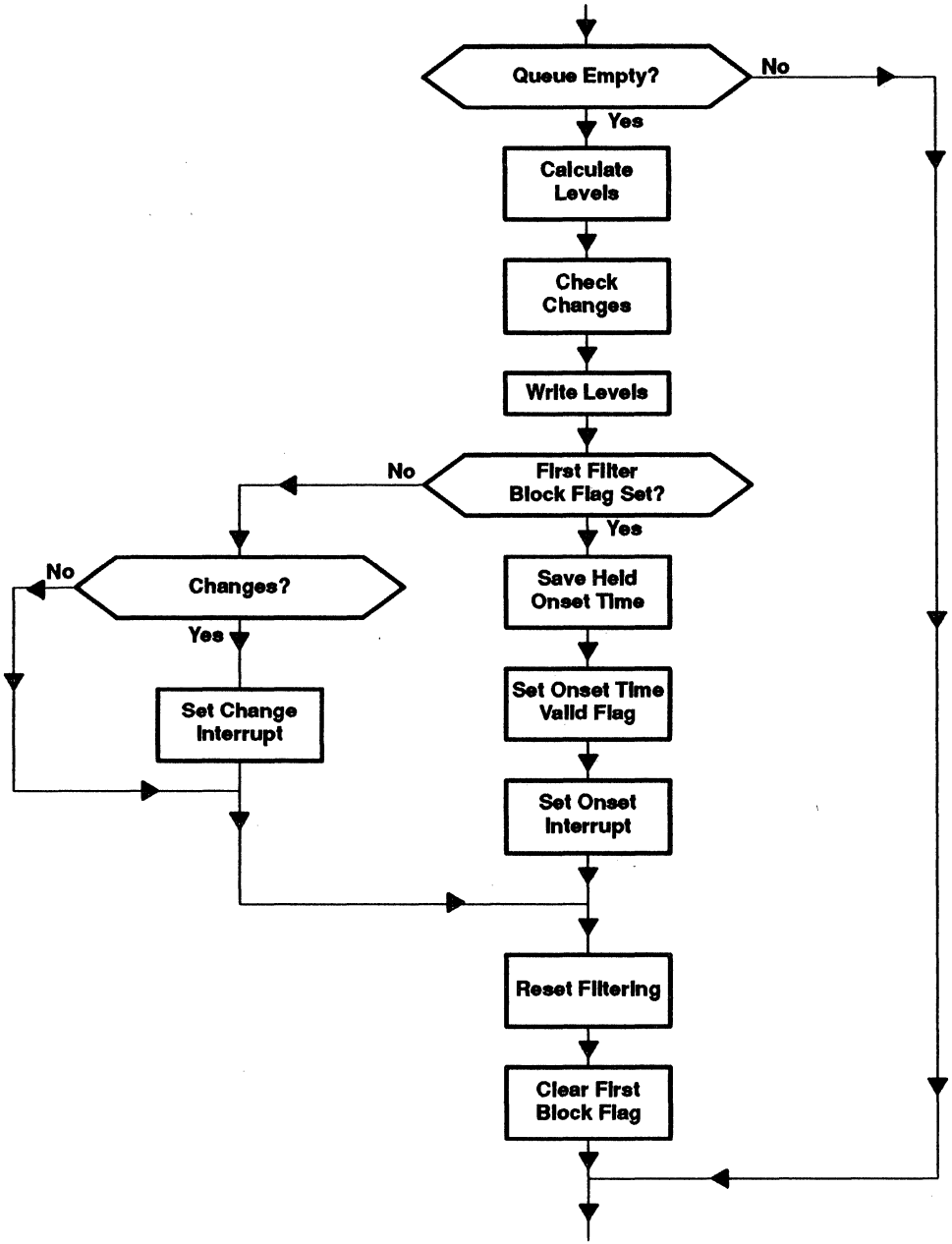



**Figure 12. Tone Receiver Flow Chart—End of Filter Block**

Calculate Levels—For each filter, the root of the sum of the squares of the corresponding sine and cosine accumulations is calculated and normalized using the appropriate filter-coefficient accumulation. The result represents the signal level falling within the pass-band of the filter. The square root of the signal-squared accumulator represents the total signal level present within the telephony band. Provided that the filters have been correctly placed, the root of the sum of the squares of the filter outputs should equal the total signal level. This allows a check to be made for tones present but not registered by the filters in use.

Check Changes, Write Levels—The output level of each of the six filters is checked to see whether any of them has crossed the change threshold programmed by the user. The signal levels in the six bands are then written to registers for the host to read. The second three filters will be zero if DTMF is switched on.

Save Held Onset Time, Set Onset Time Valid Flag, Set Onset Interrupt—If the block of filtering that has just been completed was the first one performed on the current tone there are a few other tasks to perform. The tone onset time is saved in a register for the host to read and then the host is informed by interrupt that a tone onset has occurred and that timer registers containing information about the tone are available to be read.

Changes?, Set Change Interrupt—If the completed filter block was not the first block after tone arrival, it is necessary to check for any changes to the tone. If any signal levels have crossed the change threshold in a filtering block other than the first block, then a change interrupt is asserted. Registers containing information about the tone may contain misleading information due to the likelihood of the change having occurred in the middle of a filtering operation.

Reset Filtering—Clears down all the accumulators and registers used by the filters.

Clear First Block Flag—Clears a flag set to indicate that the first block of data was being filtered.

### *I/O Routines (Interrupt Handler)*

Both host and signal (serial) I/O are dealt with by the interrupt handler. Host read or write accesses cause an external hardware interrupt to the TMS320C17. The availability of a new codec sample within the serial port receive register causes an internal hardware interrupt. A flow chart of the interrupt handler is shown in Figure 13. A detailed description of some parts of the code within the interrupt handler are contained in Interrupts section.

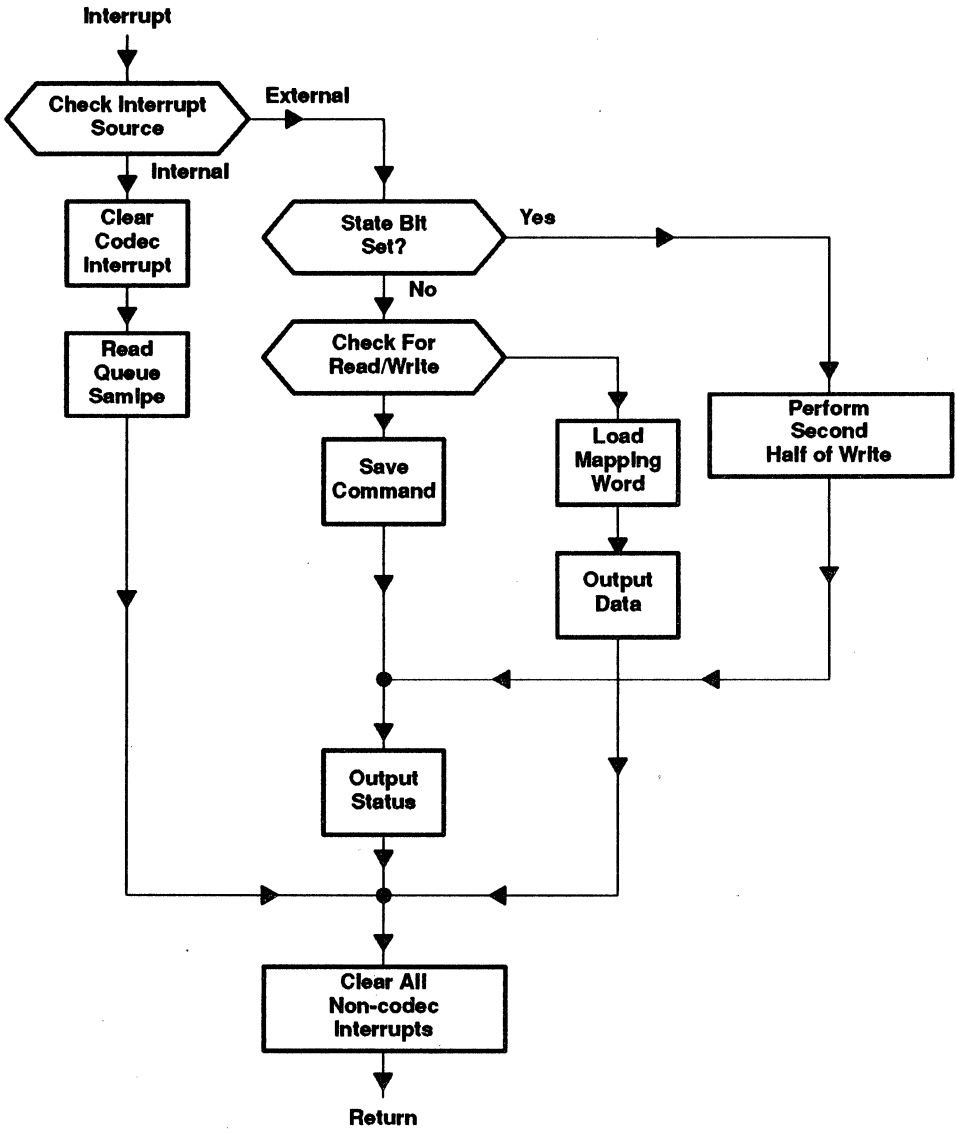



**Figure 13. I/O (Interrupt Handler) Flow Chart**

### *Self Test*

The tone detector system can be instructed to carry out a self-test operation at any time by writing to a bit in the mode register. The flow chart for the self test routine is shown in Figure 14. The duration of the test is 6 ms. No access should be made to the tone detector until the end of this period when the result of the self test is available in the mode register.

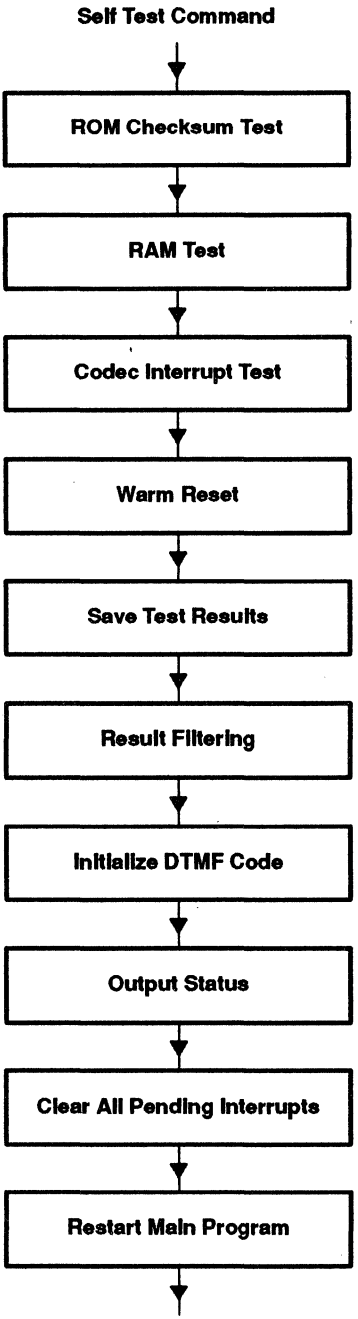



**Figure 14. Tone Receiver Flow Chart - Self Test**

Once the self-test is complete the tone detector enters a state where normal functions are inoperative, but the host data path may be tested. In this mode a write to any register other than mode or control will access a holding register inside the tone detector, rather than the register specified. This holding register may then be read by accessing any register other than mode or status, thus checking the integrity of the host data path.

Self-test is terminated by a further write to the mode register. When this has been done, the tone detector is left in the default state as though it had received a hardware reset.

### *Program Overview*

An integrated flowchart for the tone detector program is shown in Figure 15. I/O routines and self test are not included as they do not form part of the normal tone detector program flow.

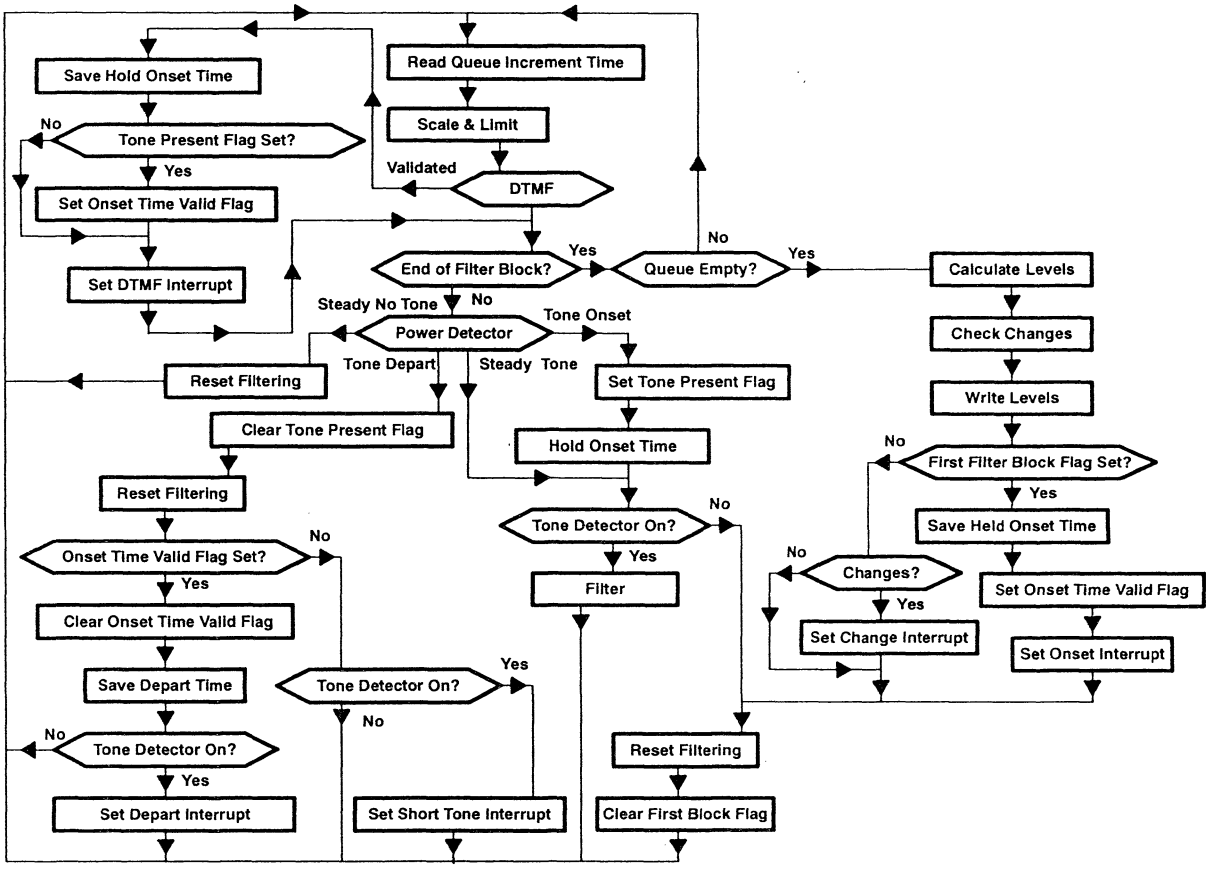



**Figure 15. Tone Detector Flow Chart (Detailed)**

## Utilization of TMS320C17 Resources

### *Central Arithmetic Logic Unit (CALU)*

The throughput capability of the CALU is one of the keys to the success of the TMS320 family. At the center of the CALU is a two's-complement 16 by 16 hardware multiplier with a 32-bit product register, which provides a result in a single cycle. Other features interfacing directly to the multiplier are the 32-bit ALU, 32-bit accumulator (ACC), two shifters and the data bus as shown in Figure 16. One input of the multiplier is provided directly from data memory via the data bus, the other is from the previously loaded temporary (T) register.

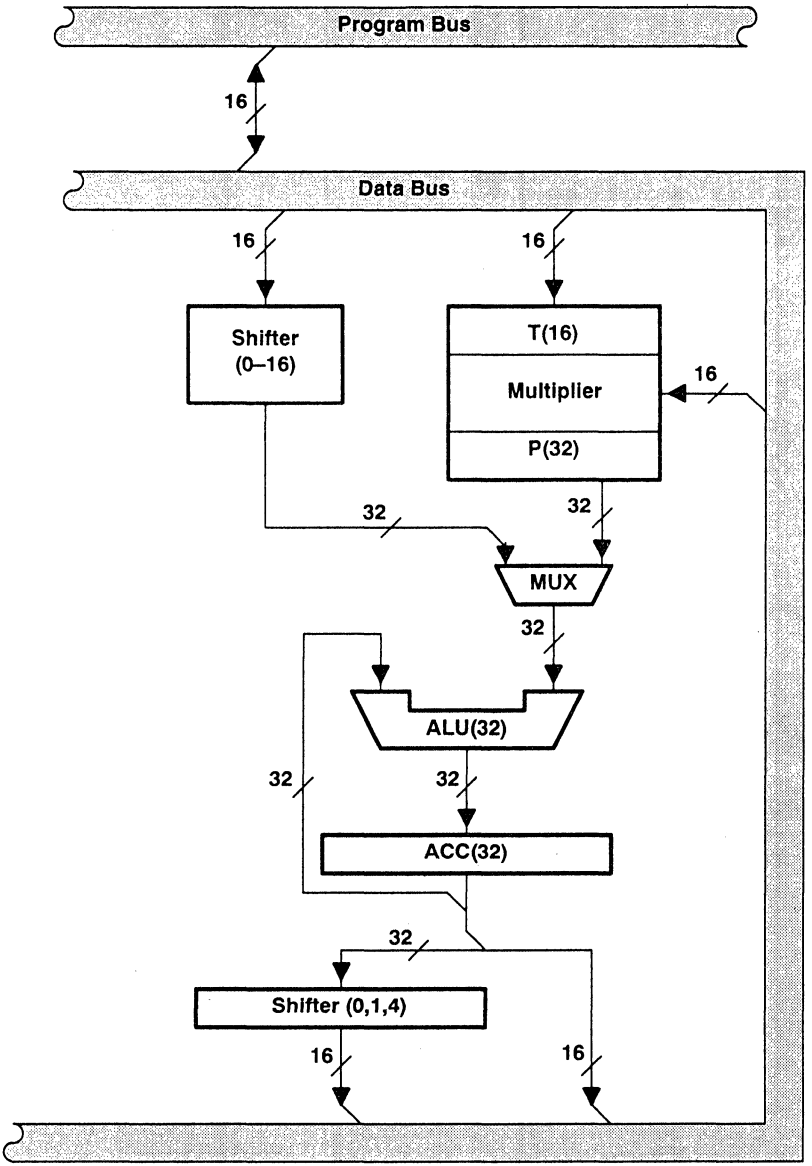



**Figure 16. Central Arithmetic Logic Unit (CALU)**

The hardware intensive approach of the CALU allows mathematically intensive algorithms to be performed very efficiently. To show its performance, the following example is taken from the ENVDET (envelope detector) routine in the source listing. Its function is to implement a smoothing filter of the form:

$$\text{ENVEL} = \frac{((2^{15} \times \text{ENVEL}) + \text{ABS}(32 \times \text{EDF} \times \text{SAMPLE}) - (32 \times \text{EDF} \times \text{ENVEL}))}{2^{15}}$$

Initial conditions are that EDF is stored in data memory location TEMP and the current envelope detector output is stored in ENVEL.

| | | |
|---|---|---|
| LAC | TEMP,5 | Puts EDF × ($2^5$) into the accumulator, using the barrel shifter to shift EDF from data RAM location TEMP left by 5 bits. |
| SACL | TEMP | Stores 32 × EDF back into TEMP. |
| LT | TEMP | Loads 32 × EDF from TEMP into T register. |
| MPY | SAMPLE | Multiplies the data value from SAMPLE by 32 × EDF and puts result into the P register. |
| PAC | | Copies P register result into accumulator. Note that an instruction which transfers the P register into the accumulator must always follow a multiply in order to ensure the contents of the P register are not lost if an interrupt occurs during the multiply instruction.<br>ACC = 32 × EDF × SAMPLE |
| ABS | | The absolute value (magnitude) of the result is left in the accumulator. |
| MPY | ENVEL | Multiplies the data value from ENVEL by 32 × EDF and puts result into P register. Note that it is not necessary to reload the T register. |
| SPAC | | Subtracts P register contents from accumulator. ACC = ABS(32 × EDF × SAMPLE) − (32 × EDF × ENVEL) |
| ADD | ENVEL,15 | Adds current value from ENVEL to accumulator with a left shift of 15 (i.e. multiplied by $2^{15}$).<br>ACC = ABS(32 × EDF × SAMPLE) − (32 × EDF × ENVEL) + (EDF × $2^{15}$) |
| ADD | ONE,14 | Adds the value $2^{14}$ to the accumulator to round up the result. |

SACH ENVEL,1 Stores the upper 16 bits of the accumulator in ENVEL with a left shift of one to remove the extra sign bit (caused by multiplying two two's-complement numbers). As it is storing the high-order accumulator, the result is effectively divided by $2^{15}$.

Thus we now have the result:

$$ENVEL = \frac{((2^{15} \times ENVEL) + ABS(32 \times EDF \times SAMPLE) - (32 \times EDF \times ENVEL))}{2^{15}}$$

This calculation takes 11 instructions and executes in 11 cycles or approximately 2.15 $\mu$s with a 20.48 MHz operating frequency.

### *Interrupts*

The TMS320C17 has an extended interrupt capability to handle a number of possible sources. These are external interrupt and serial port interrupts for any of FSR (external receive framing input), FSX (external transmit framing input) and FR (internal framing output).

Two steps are required to enable an active interrupt to the device. First, the individual interrupt must be enabled by writing to the appropriate bits in the system control register. Secondly the master interrupt circuitry should be enabled by the EINT instruction.

When an interrupt occurs, its source can be determined by reading the interrupt flag bits in the system control register. Program control can then branch to the appropriate interrupt handler.

For a full explanation of TMS320C17 interrupts refer to sections 3 and 5 of the *First-Generation TMS320 User's Guide* (Reference [6]).

#### *Interrupt Initialization*

In our example interrupts are initialized by the WRESET (warm reset handler) routine as follows. CTLPRT and CTLUPR are equated to 0 and 1 respectively to point to the I/O locations of the lower and upper 16 bits of the 32-bit system control register. Some data RAM locations are also previously set up as shown.

| | | |
|---|---|---|
| CTL320 | contains | FD9Fh |
| MS00FF | contains | 00FFh |
| ONE | contains | 0001h |

The interrupt initialization code also includes the serial port initialization. The use of the serial ports within this application is covered briefly in DTMF Telephone Tester section. The following listing should also be referred to when reading that section.

| | | |
|---|---|---|
| OUT | CTL320,CTLPRT | Sets lower 16 control bits to FD9Fh. This resets all interrupt flags, enables external and FR interrupts only, connects I/O port 1 to the upper control register, sets the XF output low, enables the serial port, selects and enables A-law encoding/decoding and selects SCLK (serial clock) as an input. |
| OUT | CTL32U,CTLUPR | Sets upper control bits to 0CFEh. This sets SCLK to 2.048MHz, sets FR to 8KHz, selects sign magnitude companding and selects FR for fixed data rate operation. |
| LAC | CTL322 | ACC = 7C90h. |
| SACL | CTL320 | Stores 7C90h back into CTL320, for future use. |
| OUT | CTL320,CTLPRT | Sets lower control bits to 7C90h. This sets SCLK to be an output, connects I/O port 1 to the serial port companding hardware, selects internal framing and leaves other options unchanged. Note it does not clear interrupt flags. |

*Interrupt Handler - Entry*

When a valid enabled interrupt is received, program execution jumps to program memory location 2. In our code, this contains a branch to label INTHDL which is at the start of the Interrupt Handler routine.

This routine contains the detailed steps for handling a serial port interrupt or an external (host interface) interrupt. All that is explained here is the code concerned with interrupt management.

| | | |
|---|---|---|
| SST | SRSAVE | Saves the current contents of the status register in data memory location SRSAVE. This is automatically in page 1 of data RAM, regardless of the value of the data page pointer. |
| LDPK | 1 | Sets the data page pointer to page 1. |
| SACH | ACCUHI | Saves the current contents of the accumulator in data memory location ACCUHI (data page 1). |
| SACL | ACCULO | As above. |
| LDPK | 0 | Resets the data page pointer to page 0. |
| SAR | AR0,ARSAVE | Saves the contents of AR0 in ARSAVE (data page 0). |
| LARP | 0 | Ensures auxillary register pointer is 0 for future indirect memory accesses. |
| IN | ITEMP,CTLPRT | Stores lower order system control register in data memory location ITEMP (data page 1). |

| | | |
|---|---|---|
| LAC | ONE,3 | Loads $2^3$ into accumulator, ACC = 0004h. |
| AND | ITEMP | ANDs data in ITEMP with 0004h in order to test whether bit 2 in system control register is 1, (i.e. is it a serial port interrupt?). |
| BZ | NOTCDC | If bit 2 not set, it is not a serial port (codec) interrupt and execution branches to the routine for external (host interface) interrupts. |

*Interrupt Handler - Exit*

All external interrupts return through the following path

| | | |
|---|---|---|
| LACK | 7 | Loads 7 into accumulator. |
| ADDS | CTL320 | Adds CTL320 (7C90h) to accumulator with sign extension suppressed as we are not dealing with two's-complement numbers.<br>ACC = 7C97h |
| SACL | ITEMP | Store accumulator into ITEMP. |
| OUT | ITEMP,CTLPRT | Clears all interrupts except internal framing, leaves all other bits in system control register unchanged. |

Note only non-codec interrupts are cleared here. Codec (serial port) interrupts are cleared at the start of the codec interrupt routine. This is because the two interrupt sources are asynchronous. Thus it is quite possible for a serial port interrupt to occur during the external interrupt routine and vice-versa. It is essential that these "pending" interrupts are not lost during the handling of the previous interrupt.

The codec interrupts join the external interrupt exit path here

| | | |
|---|---|---|
| LAR | AR0,ARSAVE | Restores AR0 value to that prior to entering interrupt routine. |
| LDPK | 1 | Sets data page pointer to page 1. |
| ZALH | ACCUHI | Loads high accumulator with exact copy of ACCUHI. |
| ADDS | ACCULO | Loads low accumulator with exact copy of ACCULO with sign extension suppressed to leave high accumulator unaffected. |
| LST | SRSAVE | Restores status register value with that prior to entering interrupt routine. |
| EINT | | Enables interrupts. This instruction always waits until the following instruction has completed execution so that interrupts are not nested. |

| | | |
|---|---|---|
| RET | | Returns program control to the point at which the interrupt occurred. |

***Serial Ports***

Serial port initialization occurs at the same time as interrupt initialization as both involve the use of the TMS320C17 Control Registers. This is covered in detail in the interrupt section above.

This application uses a single serial input only. A TCM2917 codec chip operated in the fixed data rate mode is used to provide analog to digital conversion. A 2.048 MHz clock (SCLK) is provided by the TMS320C17 along with a framing signal (FR) giving a sampling rate of 8 KHz. With CDCPRT having been equated to one, data transfer is simply by the use of the following instruction

| | | |
|---|---|---|
| IN | *,CDCPRT | Inputs data from I/O port 1 which has been switched to accept serial input from the companding hardware by a previous write of a one to control register bit 8. |

## Hardware Implementation

The example outlined below is a possible design for a tone detection system as a peripheral to an IBM XT or AT compatible PC bus. Figure 17 shows the complete circuit schematic for this design. The circuit uses only four integrated circuits to implement a full-functionality tone detector. The signals required from the PC bus are SA0 - SA9 (latched address bus), D0 - D7 (8-bit data bus), $\overline{\text{IOW}}$ (I/O Write), $\overline{\text{IOR}}$ (I/O Read), RESET DRV (System Reset), and AEN (Address enable for DMA). Figure 18 shows the PC bus activity for these signals during an I/O operation. For more detailed information on the function and behaviour of these signals see References [3] and [4].

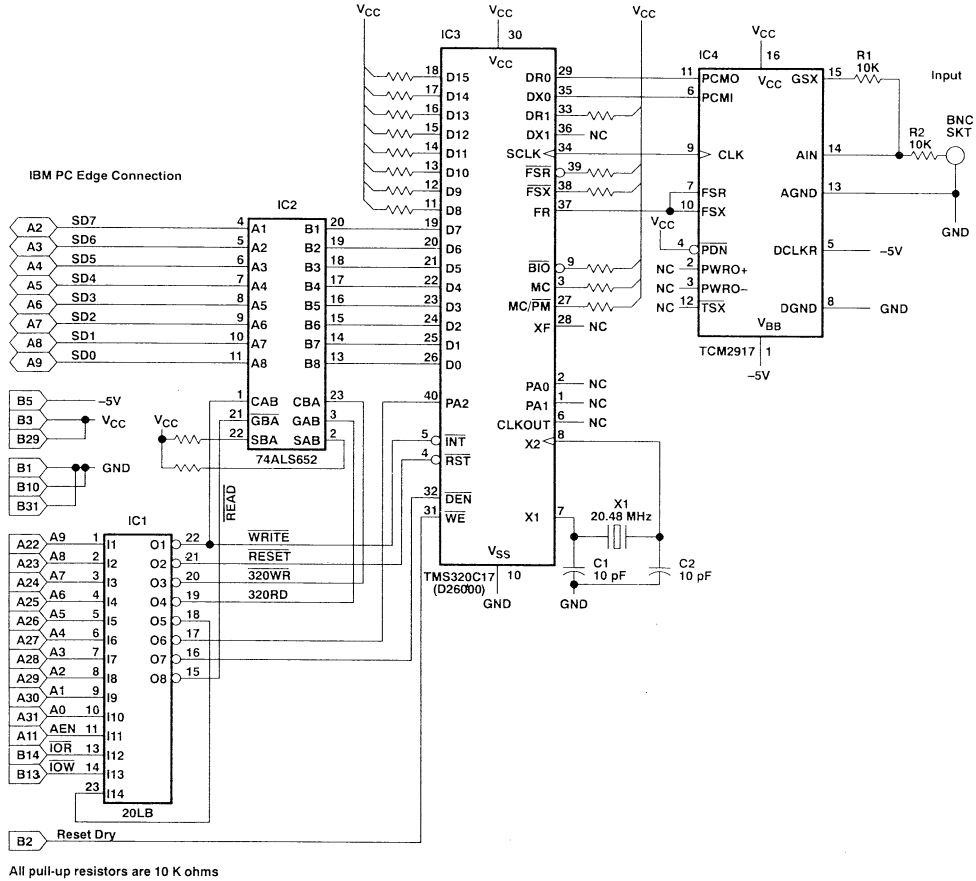



**Figure 17. Tone Detector PC Application Circuit Diagram**

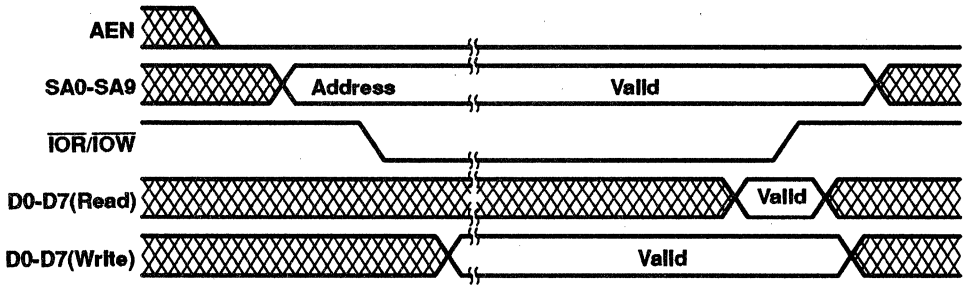



**Figure 18. PC Bus Activity I/O Read or Write**

The XF (external flag) pin of the TMS320C17 may also be used to signal an interrupt on one of the PC bus lines IRQ3 - IRQ7 (Interrupt requests), if it is desired to have an interrupt driven and not a polled interface. The example shown is based on a polled interface and does not utilize host interrupt.

***Host Read/Write Decode***

The PAL (programmable logic array) can give a host read or write function at any address in the range 0 to 03FFh (hexadecimal). Only one I/O address is used by the tone detection system in this example. For use in a PC, any free address in the I/O space could be chosen. The AEN signal is also passed to the PAL to ensure that the system is not mistakenly accessed during a direct memory access (DMA) cycle.

Assume that the I/O address of the tone detector is 0300h. The equations for the host read and write strobes would be as follows:

$$\overline{\text{READ}} = \overline{\text{A9}} \# \overline{\text{A8}} \# \text{A7} \# \text{A6} \# \text{A5} \# \text{A4} \# \text{A3} \# \text{A2} \# \text{A1} \# \text{A0} \# \overline{\text{IOR}} \# \text{AEN}$$

$$\overline{\text{WRITE}} = \overline{\text{A9}} \# \overline{\text{A8}} \# \text{A7} \# \text{A6} \# \text{A5} \# \text{A4} \# \text{A3} \# \text{A2} \# \text{A1} \# \text{A0} \# \overline{\text{IOW}} \# \text{AEN}$$

where # represents the logical OR function.

***TMS320C17 I/O Read/Write Decode***

The PAL also provides the decode function for TMS320C17 IN and OUT (read and write) operations. A TMS320C17 read and a data write always use I/O port 4. A status write is made to port 6. Ports 0 and 1 are reserved for internal functions of the TMS320C17. Other ports are not implemented in this system.

The equations for a TMS320C17 read and write are as follows:

$$\text{320RD} = \text{DEN} \ \& \ \text{PA2}$$

$$\overline{\text{320WR}} = \overline{\text{WE}} \# \overline{\text{PA2}}$$

### *Host Data Write*

Upon receipt of the correct I/O address and the I/O Write strobe, the data present on the PC bus is latched into the 74ALS652 on the rising edge of I/O Write. Simultaneously, an interrupt is given to the TMS320C17. As previously described, the TMS320C17 responds to this interrupt by performing a read operation from its input port 4.

The TMS320C17 read is implemented by PA2 being set high and $\overline{\text{DEN}}$ (data enable) acting as a read strobe. While data enable is low, the high-impedance outputs of the 74ALS652 are enabled and the TMS320C17 reads an 8-bit value. This contains the address of the register to be accessed and the read/write bit which is set to indicate a host write in this case.

The read of port 4 is then followed by a write of the current contents of the status register from the TMS320C17 to output port 6. This is implemented by PA2 and PA1 being set high and $\overline{\text{WE}}$ (write enable) being used as a write strobe. When write enable goes high to signify the end of the write, the data on the low order data bus (D7 to D0) of the TMS320C17 is latched into the 74ALS652.

The second part of the host data write operation is an exact duplication of the above sequence of events. It would then be normal to read the status information returned at the end of the cycle. This is done by a simple I/O read from the address of the board which enables the contents of the 74ALS652 onto the PC data bus.

### *Host Data Read*

This operation is based on the same sequence of events as above, as indicated in Host Read Cycle section.

### *Host Reset*

The active high RESET DRV signal is taken from the PC bus, inverted and applied to the TMS320C17 RS input (pin 4).

### *Host Interrupt*

As mentioned briefly above, the TMS320C17 uses the external flag (XF) pin (pin 28) to signal an interrupt to the host. This interrupt may come from a number of sources as described in Control section. This signal is active low and is set to a high level after a reset to the TMS320C17. There is a period of 2 ms after the release of reset for which the state of the interrupt should be ignored, as it is set inactive only by execution of the appropriate instruction. The state of XF is therefore undefined for the period between the application of reset and the execution of the instruction which initializes it to the inactive state.

The easiest method to overcome this would be only to enable the appropriate host interrupt line at least 2 ms after the release of reset.

### *Host Handshake*

There is no host handshake implemented on the example application described here. The maximum length of time which a single read or write can occupy is 20 $\mu$s. The host should ensure that consecutive accesses do not occur more closely than this.

### *Analog Interface*

This function is performed by an industry-standard combined PCM codec and line filter (COMBO), the TCM2917 (see Reference [5] which provides A/D and D/A conversion as well as transmit and receive filtering. In this application the codec is set to a gain of 1. The TCM2917 performs A-law companding and operates in this circuit in the fixed data rate mode of 2.048 MHz. As this application was developed in Europe, the A-law companding TCM2917 was used. For applications in North America this may be replaced with the TCM2916 which provides $\mu$-law companding and is pin-for-pin compatible with the TCM2917. There is a small change to be made to the area of program which initializes the control registers in the TMS320C17. This is covered in detail in Substitution of TCM2916 for TCM2917 subsection.

The TCM2917 interfaces directly to one of the two serial ports on the TMS320C17 which were designed to facilitate the use of this type of device (see References [1] and [6] for further information).

## Host Interface

The tone detector function described in this application note appears to the host CPU bus as a single 8-bit parallel port. This port is used as shown below to give access to the sixteen read and write registers within the TMS320C17.

In the particular example presented here the interface is of the polled access type. An interrupt driven interface can be implemented by setting the appropriate bits of the tone detector control register and connecting the XF pin of the TMS320C17 (pin 28) to a host interrupt input.

### Host Write Cycle

The host CPU writes to one of the 16 available registers by a four step process as shown in Figure 19.

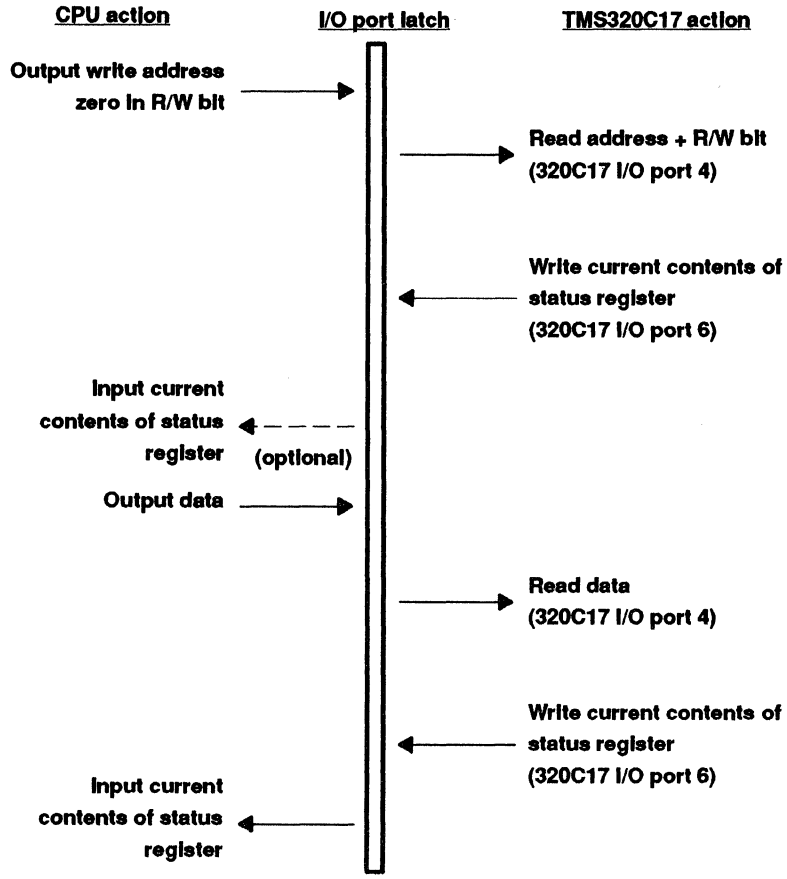



**Figure 19. Host Write Cycle**

The write cycle is initiated by an output from the host CPU to the I/O port or memory location occupied by the tone detector. The first byte of data transferred is a command byte which contains the address of the register to be written to and the read/write bit set to a zero to indicate a write operation. The bit assignment is as shown below.

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | A3 | A2 | A1 | A0 |
| | | | R/W | Register address | | | |

A3 is the most significant bit of the tone detector register number and A0 is the least significant bit.

Following this host CPU command the TMS320C17 will make the current contents of its status register available for input by the CPU. It is not usual for the host CPU to read the status information at this point.

This is followed by a host CPU write of the data to be transferred into the tone detector register. The operation is completed by the TMS320C17, which again makes the current contents of its status register available. It would be normal for the host CPU to read this status byte from the I/O port at this time.

## Host Read Cycle

The read cycle is initiated by the host CPU in a similar way to the write cycle above, and is shown in Figure 20.

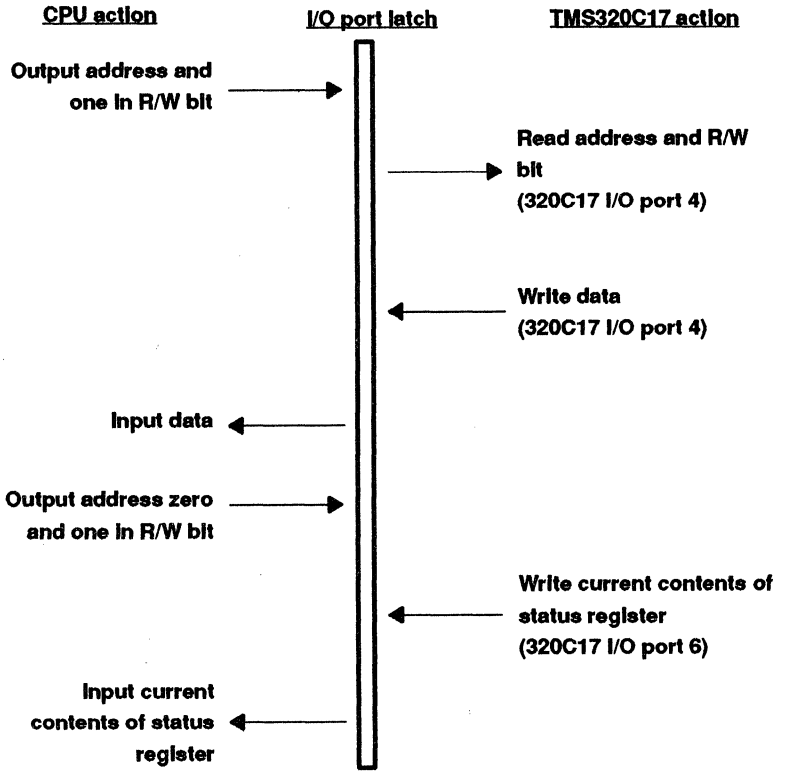



**Figure 20. Host Read Cycle**

In this case, the read/write bit is set to a one to indicate a read. Following the initial host CPU write of the address, the TMS320C17 makes the contents of the addressed register available for the host CPU to read. The cycle is completed when the host CPU issues a second register read request with an address of zero (status register) and the TMS320C17 makes available the current contents of its status register for the host CPU to read.

## Host Access Considerations

The host CPU may not attempt to perform any new access of a tone detector register before the previous access is complete. A read operation must be fully completed before a write is initiated and vice versa. Additionally, neither read nor write operations should be nested. Both the host read and host write should be regarded as discrete tasks to be executed in isolation from any other host access.

A delay should be allowed between the host CPU writing the register address to the tone detector and reading the subsequent response (data for a read cycle, status for a write cycle). This delay should be a minimum of 20 $\mu$s. No delay is necessary between reading the response and performing a subsequent write operation, but a further minimum 20 $\mu$s delay should be allowed prior to the next read. This delay allows the TMS320C17 to retrieve the correct data from its data memory, perform any necessary calculations and output it to the interface latch.

## Host Interface Registers

Although the tone detector only occupies one physical 8-bit read/write host location, the full interface is implemented by sixteen read and write registers within the TMS320C17. Their allocation is shown below:

| Address | Read Register | Write Register |
|---|---|---|
| 0 | Status | Control |
| 1 | Mode | Mode |
| 2 | DTMF digit | Envelope decay factor |
| 3 | Tone arrival (MS byte) | Upper threshold |
| 4 | Tone arrival (LS byte) | Lower threshold |
| 5 | Tone departure (MS byte) | Filter length |
| 6 | Tone departure (LS byte) | Passband width |
| 7 | Current time (MS byte) | Change threshold |
| 8 | Current time (LS byte) | Frequency (MS byte) |
| 9 | Band 1 signal level | Band 1 frequency (LS byte) |
| A | Band 2 signal level | Band 1 frequency (LS byte) |
| B | Band 3 signal level | Band 1 frequency (LS byte) |
| C | Band 4 signal level | Band 4 frequency (LS byte) |
| D | Band 5 signal level | Band 5 frequency (LS byte) |
| E | Band 6 signal level | Band 5 frequency (LS byte) |
| F | Total signal level | Filter select |

Where MS byte refers to the most significant (upper) byte of a 16-bit word, and LS refers to the least significant (lower) byte of a 16-bit word.

## Register Read Functions

Except where specified all of the following read registers are set to zero by a hardware reset.

***Status***

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| | ST | DT | TC | TA | TD | | |

ST— This bit is set to zero when the departure of a tone is detected by the envelope detector before it has been validated as a DTMF tone or a filtering operation has been completed on the tone.

DT— This bit is set to zero when the occurence of a valid DTMF tone pair is detected.

TC— This bit is set to zero when a change in tone is detected.

TA— This bit is set to zero when the arrival of a tone is detected by the envelope detector, and the tone has been validated as a DTMF digit or a filtering operation has been completed on the tone.

TD— This bit is set to zero when the departure of a tone is detected by the envelope detector.

Each of the bits in the register are set to one by writing the appropriate value to the IACK bits (bits 2-4) of the MODE register (see Mode section).

A reset will cause all of the bits of the status register to be set to one.

***Mode***

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| TEST | DTMF | TONE | IACK2 | IACK1 | IACK0 | RC1 | RC0 |

Each of the bits in this register, except RC1 and RC0, simply reflect the last value written to the corresponding bit in the mode register.

RC1 These two bits together form the result code generated by a self
RC0 test operation by the tone detector.

The meanings of the result codes are as follows:

| RC1 | RC0 | Meaning |
|---|---|---|
| 0 | 0 | Clock failure |
| 0 | 1 | Test successfully completed |
| 1 | 0 | RAM failure detected |
| 1 | 1 | ROM failure detected |

***DTMF Digit***

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| OVRUN | | | | DD3 | DD2 | DD1 | DD0 |

OVRUN— This bit is set to one when there has been an overrun of received DTMF digits, ie. a new digit has been received when the DT bit in the status register was set to zero (before the host has acknowledged the receipt of a previous digit). OVRUN remains set to one until a DTMF digit is received while the DT bit in the status register has a value of one. The digit indicated by DD3-DD0 is the last received digit regardless of the state of OVRUN.

DD3 to— DD0 These four bits together identify the last valid received DTMF digit.

The digits are identified as follows:

| DD3 | DD2 | DD1 | DD0 | Received Digit |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 2 |
| 0 | 0 | 1 | 0 | 3 |
| 0 | 0 | 1 | 1 | A |
| 0 | 1 | 0 | 0 | 4 |
| 0 | 1 | 0 | 1 | 5 |
| 0 | 1 | 1 | 0 | 6 |
| 0 | 1 | 1 | 1 | B |
| 1 | 0 | 0 | 0 | 7 |
| 1 | 0 | 0 | 1 | 8 |
| 1 | 0 | 1 | 0 | 9 |
| 1 | 0 | 1 | 1 | C |
| 1 | 1 | 0 | 0 | * |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | # |
| 1 | 1 | 1 | 1 | D |

***Tone Arrival (MS Byte and LS Byte)***

The two tone arrival registers are read by the host CPU in conjunction. They report the time at which the arrival of a tone was detected. The 16-bit value formed by (256 × MS) + LS is treated as an unsigned integer giving the time at which tone arrival was detected in milliseconds. This time is taken from the contents of the current time register (see Current Time section) at the moment of the tone arrival being recognised by the power detector.

The tone arrival registers are updated when either a DTMF digit is detected, or a filtering operation is completed.

***Tone Departure (MS Byte and LS Byte)***

The two tone departure registers are read by the host CPU in conjunction. They report the time at which the departure of a tone was detected. The 16-bit value formed by (256 × MS) + LS is treated as an unsigned integer giving the time at which tone departure was detected in milliseconds, as taken from the current time register.

Neither the tone arrival or tone departure registers are updated by the arrival or departure of a short tone, i.e. one which had departed before being recognised as a DTMF digit, and before a tone receiver filtering operation had been completed on it.

***Current Time (MS Byte and LS Byte)***

The two current time registers are read by the host CPU in conjunction. They report the current time indicated by the tone detector module. The 16-bit value formed by (256 × MS) + LS is treated as an unsigned integer giving the current time in milliseconds. Reading the current time (MS byte) register causes the value of the current time (LS byte) register to be copied into a holding register. In order to get a correct reading of the full 16-bit value of the current time the MS byte should therefore be read first.

When current time reaches the maximum value of 65535, the next increment takes it to zero. The current time increments every millisecond upon release of hardware reset.

***Band 1-6 Signal Level***

The signal levels received in each of the frequency bands specified are reported in these six frequency band signal level registers. The values read from these registers are to be interpreted as 8-bit unsigned integers, SL. If a value of SL is read from a register, then the signal level represented is:

$$\frac{(5.30 \times SL)}{GAIN} \text{ mV rms (root mean square)}$$

See Mode subsection in Host Interface section for a description of the gain factor (GAIN).

Typical values which may be read are as follows:

| SL | Signal Level | Codec Level |
|---|---|---|
| 40 | 212.0/GAIN mV rms | −14.1 dBm0 − G dB |
| 254 | 1346.0/GAIN mV rms | + 2.0 dBm0 − G dB |

An input signal level of greater than 1346/GAIN mV rms will result in a value of SL = 255.

When the DTMF bit in the mode register is set to one, the values read from Band Signal Level Registers 4 to 6 are all zero as only three frequency bands can be monitored while the DTMF receiver is active. The DTMF bit must be set to a zero if bands 4 to 6 are to be monitored.

### *Total Signal Level*

The signal level received over the frequency range 300 Hz to 3400 Hz is reported in the total signal level register. The number format is identical to that described for the band 1-6 signal level registers.

## Register Write Functions

Except where explicitly stated, a hardware reset will set each register to zero and, when the contents of any register are changed, the tone detector uses the new value immediately.

### *Control*

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| | INTST | INTDT | INTTC | INTTA | INTTD | | |

Writing a one to any of the bits in the control register enables an interrupt to be signalled on the XF pin of the TMS320C17 when the corresponding bit in the status register is set to zero.

### *Mode*

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| TEST | DTMF | TONE | IACK2 | IACK1 | IACK0 | GF1 | GF0 |

The functions of the bits in the mode register are as follows:

TEST— Writing a one to this bit starts a self test operation. The result of the test is reported in the lower bits of the mode register. As long as TEST is a one the tone detector remains in the TEST mode and no register accesses may take place. The self-test is terminated by writing a zero to TEST after which the tone detector is left in the default state assumed after a reset. A self test operation takes approximately 6 ms.

DTMF— Writing a one to this bit enables the detection of DTMF digits. On entering the active state, the DTMF receiver begins looking for DTMF digits as though it had been monitoring a silent line in the recent past.

TONE— Writing a one to this bit enables the detection of tones. When the tone detector is turned on, it will wait for the envelope detector to indicate that a tone is present before starting filtering operations.

IACK0— to IACK2 The pattern written to these bits selects which of the five possible interrupt conditions from the tone detector module is being acknowledged. The acknowledgement of an interrupt causes the corresponding status bit to be set to the one state.

The selection patterns are as follows:

| IACK2 | IACK1 | IACK0 | Interrupt to be acknowledged |
|---|---|---|---|
| 0 | 1 | 0 | Tone Departure (TD) |
| 0 | 1 | 1 | Tone Arrival (TA) |
| 1 | 0 | 0 | Tone Change (TC) |
| 1 | 0 | 1 | DTMF Digit Arrival (DT) |
| 1 | 1 | 0 | Short Tone (ST) |

Other patterns have no effect

GF1-GF0— The two bit pattern written to these bits selects which of three gain factors is applied to the input signal before it is passed to the DTMF and tone receivers and the envelope detector. By writing a suitable value to these bits, it is possible to adjust the tone detector module to accommodate very loud or very quiet signals. The selection patterns are as follows:

| GF1 | GF0 | Gain Factor (GAIN) | Relative Gain (G dB) |
|---|---|---|---|
| 0 | X | 4 | 12 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 16 | 24 |

***Envelope Decay Factor***

The time constant of the envelope detector is the time taken for the output of the detector to reach 63% of its final value. The value written to the envelope decay factor register is treated as an 8-bit unsigned integer, EDF. If the time constant required for the envelope detector is t, then EDF should be specified as

$$EDF = 1024 \times [1 - \exp(-1/(8000t))].$$

For example for a time constant of 1.0 ms, EDF should be set to 120.

A reset will cause this register to be set to a value of 120.

### *Upper Threshold*

The upper threshold is the signal level at the output of the envelope detector at which the arrival of a tone is recognized. The number written to the upper threshold register is treated as an 8-bit unsigned integer, UT. If the signal level required for this threshold is $V_{ut}$ Volts rms, then UT should be specified as

$$UT = 254 \times (0.743 \times GAIN \times V_{ut})$$

For example for an upper threshold of 425/GAIN mV, UT should be set to a value of 80. This represents a codec input of $-8.0$ dBmO $-$ G dB.

A reset will cause this register to be set to a value of 255.

### *Lower Threshold*

The lower threshold is the signal level at the output of the envelope detector at which the departure of a tone is recognized. The lower threshold is specified in exactly the same way as the upper threshold described above.

If the value programmed into the lower threshold register is larger than the value programmed to the upper threshold register, the value in the lower threshold register is taken as the threshold for both tone arrival and tone departure.

A reset will cause this register to be set to a value of 255.

### *Filter Length*

The filter length register defines the number of samples of the input signal which are required to produce one result from the tone detector. The rate at which the codec feeds samples to the tone detector is 8000 samples per second, or one sample every 125$\mu$s. The value which is written to this register is treated as an 8-bit unsigned integer, FL. The length of filter specified by the value FL is

$$\frac{16384}{FL + 16} + 1 = N \text{ samples}$$

For example, for a filter length of 410 samples, FL should be set to 24, giving a filter duration of 51.3 ms.

The filter length defines the steepness of cutoff at the filter band edge. Figures 21 and 22 give an indication of the filter band edge shape for both wide filters and narrow filters of different lengths. They should be treated as indicative of the performance of the tone detector.

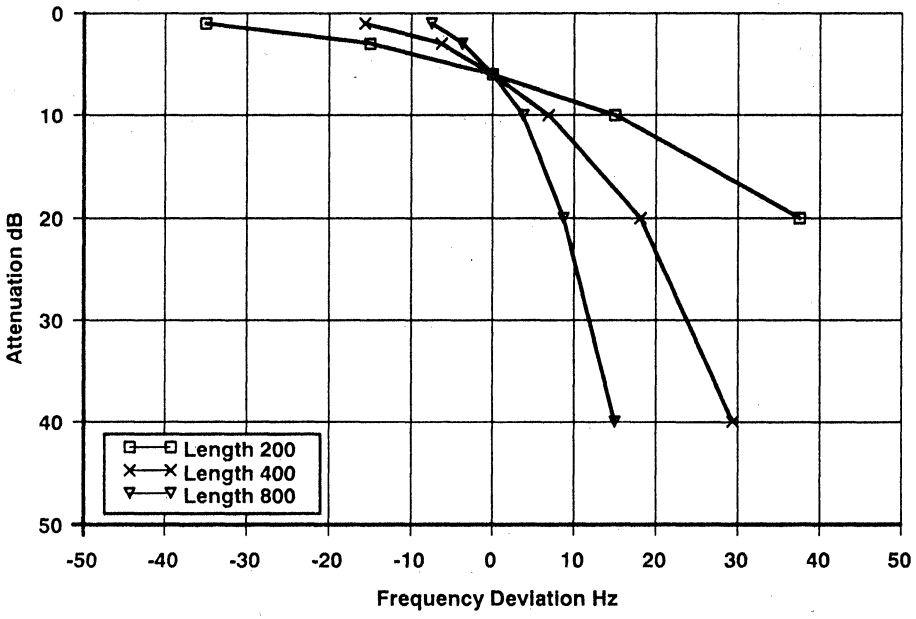



**Figure 21. Filter Band Edge Shape - Wide Filter**

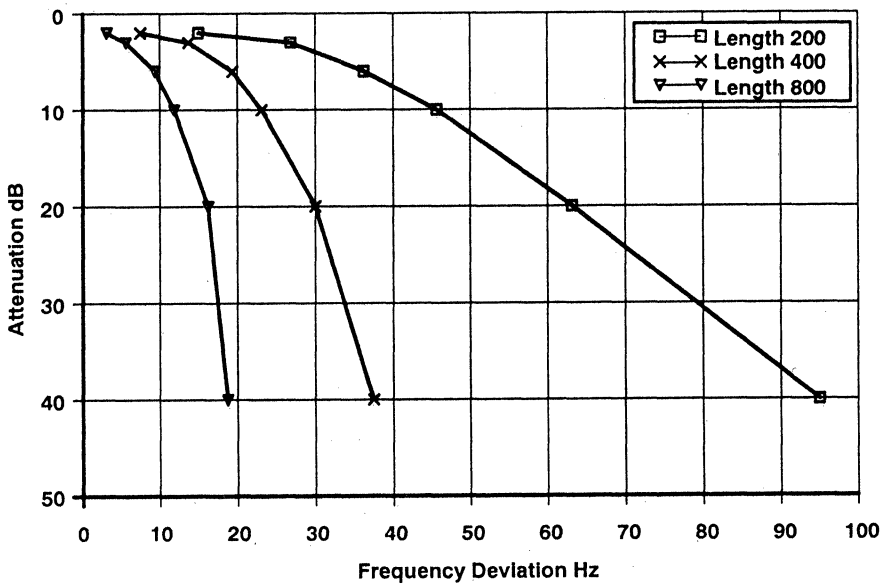



**Figure 22. Filter Band Edge Shape - Narrow Filter**

When contents of this register are changed, the tone detector waits until the start of the next filtering operation before using a new filter-length value.

A reset will cause this register to be set to a value of 50 (250 samples).

***Passband Width***

The bandwidth of the bandpass filters used by the tone detector is specified by the passband width register. The value which is written to this register is treated as a 6-bit unsigned integer, PW. If a bandwidth of Y Hz is required, then PW should be specified as:

$$PW = Y \times 0.128$$

The maximum permitted value for PW is 63, giving a passband width of 492 Hz.

The bandpass filters used by the tone detector are symmetrical about the center frequency, i.e. a bandwidth of X Hz defines that frequencies which deviate by up to X/2 Hz from the center frequency fall within the passband.

***Change Threshold***

At the end of each filtering operation (except the first after tone onset) the signal received at each of the monitored frequencies is compared against the signal received during the previous filtering operation. If the signal level at any one of the monitored frequencies has crossed the signal level threshold defined in the change threshold register, then the Tone Change status bit is set in the status register. The change threshold is defined in an identical manner to the upper threshold described above.

When the contents of this register are changed, the tone detector uses the new value of change threshold on the next signal level comparison.

***Frequency (MS Byte)***

The 8-bit value, FMS, written to the frequency register (MS byte) forms the most significant byte of the 16-bit specifier of a filter center frequency. When a value is written to one of the frequency (LS) registers, the current contents of frequency (MS) is concatenated with the 8-bit LS value defined below to form the 16-bit frequency specifier. The frequency (MS byte) register must therefore contain the desired value when the LS byte is written.

***Band 1-6 Frequency (LS Byte)***

The 8-bit value, FLS, written to one of the band 1-6 frequency (LS byte) registers is concatenated with the 8-bit value most recently written to the frequency (MS byte) register to form the 16-bit specification for the filter center frequency. If a center frequency of G Hz is required, then FMS and FLS should be specified as follows:

$$(FMS \times 256) + FLS = 8.192 \times G$$

or $FMS = (8.192 \times G)$ div 256
and $FLS = (8.192 \times G)$ mod 256

***Filter Select***

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| | | F1 | F2 | F3 | F4 | F5 | F6 |

Writing a one to any of the bits in the filter select register causes the corresponding filter to adopt the passband width specified by the passband width register (wide filter). Writing a zero causes the filter to adopt a zero passband width (narrow filter).

A reset will cause each of the bits in this register to be set to one.

## Applications and Customization

The combination of a programmable tone receiver and a CEPT DTMF decoder in a single chip opens up a wide range of potential applications. The operation of the device across the 300-3400 Hz band targets its use towards telephony, but this is by no means the only area to which it can be applied.

The examples shown here are chosen from the more obvious potential applications. Some examples do not utilize the full power of the system, but they will hopefully serve to illustrate the capabilities of the tone detector and act as a stimulus for the development of innovative designs.

### Secure Off-Site Control

The tone detection system described may be used within a secure off-site control system. An increasing amount of such equipment is now available, designed to respond to commands given remotely via a telephone line, as shown in Figure 23. These commands are typically a single or a sequence of DTMF tone(s), and may be supplemented by special tones.

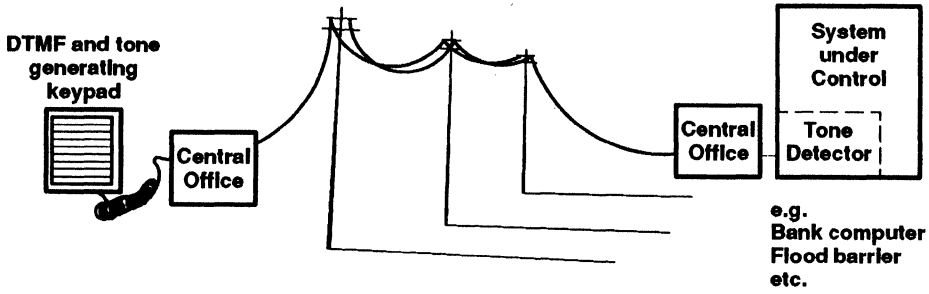



**Figure 23. Secure Remote Controller**

The level of security required varies with each type of equipment depending upon its function. For example, a home answering machine does not require a high level of security to protect its stored messages from being replayed to a remote telephone. At the other end of the scale, it is clearly important that financial information or transactions be heavily protected in the new remote banking systems now becoming available.

Sequences of DTMF tones of varying lengths with various intervals provide one level of security which would be more than adequate for remote activation in the case of the home answering machine. However, DTMF tones are limited by definition to a set of sixteen tones making computer controlled attack (hacking) of any equipment relying on them for protection relatively easy. The method of protection used for cash-cards, etc., where three unsuccessful attempts at breaking a code (the personal identification number, or PIN) result in a machine refusing to return the card is not feasible in that any remotely accessed system must be ready to respond to its authorized user at all times.The system cannot just shut down if it suspects it is under attack from an unauthorized source.

One way of providing the protection needed would be to make the number of possible combinations of activating tones impractically large for any systematic hacking. This could easily be achieved by extending the number of tones capable of detection beyond the sixteen provided by DTMF.

The tone detector presented here makes just such a scheme possible by providing capability for the accurate detection of a single frequency or any combination of up to six simultaneous frequencies within the telephony band. With the added variety of variable lengths of tone presence and absence, and sequential combinations of different tones, it is clear that a very high level of security can be offered. The tone detector offers time stamping of tone arrival, tone change, and tone departure and would thus make it easy for any equipment to which it is attached to decide whether or not to allow access.

## Call Monitoring

Call monitoring functions may be implemented using the tone detection system described here. Across the various telephone companies in the world, there is a large variety of call progress tones used. It may also be useful to decode other tones received down a telephone line. An example might be for an answering machine to detect the fact that it is accidentally being called by a modem, or for auto dialing equipment to detect that it has accidentally called a modem. The ability to decode national call progress tones and other random tones received is of particular use in, for example, a PC with an integral telephony function. Here a range of actions may be expected of the PC depending upon the exact nature of the received tone. This application relates directly to the design example presented in the Host Interface section where a four-chip solution is shown for a PC tone detection peripheral.

## DTMF Telephone Tester

Using the general purpose tone detection function, a low-cost DTMF telephone tester could be built to check the conformance of a telephone, or any other fixed tone generator, to a particular standard.

With programmable center frequency (to a resolution of 0.12 Hz), programmable passband width and filter cut-off, a precise measure of an incoming tone for conformity is easily made. In a laboratory environment this could again be implemented as a peripheral to a PC. If required, the TMS320C17 could also easily be controlled by any general-purpose 8-bit microcomputer to provide a low cost portable programmable tone tester.

## Customization for User Applications

The source code for the TMS320C17 program described here is presented as Appendix A. The code takes up less than half of the on-chip ROM and allows space for user application code to be included on-chip for low chip-count solutions to a number of complex tone decoding tasks.

The TMS320E17 EPROM digital signal processor can be used for the development phase and low-volume manufacturing. For high-volume production, code can be masked onto the TMS320C17 to provide a custom DSP.

To aid integration of additional application code, certain functions of the device are not utilized by the existing source code. Of most importance is the BIO pin (pin 9) which is effectively a software interrupt. By simple insertion of a BIOZ instruction, code execution could branch to special application routines. The XF (external flag) pin of the device is used to signal an interrupt to the host. If (as is the case in the design example in this report) this function is not used, it is simple to reprogram the function of this pin for any desired purpose.

The following notes apply to any customization of the tone detector source code:

1. The correct execution of both the DTMF receiver and tone receiver functions is dependent upon certain time critical functions. Care should be taken to ensure that any change made to the code does not affect the clean handling of the continuous stream of samples from the codec.

2. Any change to the ROM code will require a corresponding change to the checksum word at program memory location 0004h (label CHECKS at bottom of page 489 of the source listing in Appendix A). The checksum test routine (see page 516 of Appendix A) sums all the program memory locations in the code and tests the lower 16 bits of the final sum for zero. It is important to maintain this zero result by adjusting the checksum word.

Alterations that may be made to the tone detector include:

- Substitution of TCM2916 for TCM2917 in North American applications
- Use of the coprocessor port for parallel I/O
- Use of either DTMF or tone receiver code in isolation

***Substitution of TCM2916 for TCM2917***

To change the TCM2917 codec for a TCM2916 requires only a small alteration in the program code. The only difference between the TCM2917 and the TCM2916 is that the TCM2917 performs A-law compression of its serial PCM data prior to output, and the TCM2916 performs $\mu$-law compression. The TMS320C17 can decode either $\mu$-law or A-law encoded data. The choice between $\mu$-law and A-law is made by the value written to bit 14 in the TMS320C17 control register.

The lower 16 bits of the control register are set by writing data memory location CTL320 to output port zero. CTL320 is initialised with a value of FD9Fh in the existing code, with bit 14 set to a one (A-law conversion). Changing this initial value to BD9Fh will ensure bit 14 is set to a zero ($\mu$-law conversion).

The change to a value of BD9Fh should be made by altering the statement

```
.word     FD9Fh     ; CTL320
```

(second statement below label CONST1 at the bottom of page 490 in Appendix A) to

```
.word     BD9Fh     ; CTL320
```

within the source file.

### *Use of Coprocessor Port for Parallel I/O*

The TMS320C17 features a coprocessor port which provides a direct interface to most 4/8-bit microcomputers and 16/32-bit microprocessors. It is possible for the tone detection system to make use of this port for connection to a variety of possible host CPUs. Figure 24 shows a simplified logic diagram for the coprocessor port. Note that RBLE, TBLF and BIO are not necessary to the tone detector interface as it uses single byte transfers only.

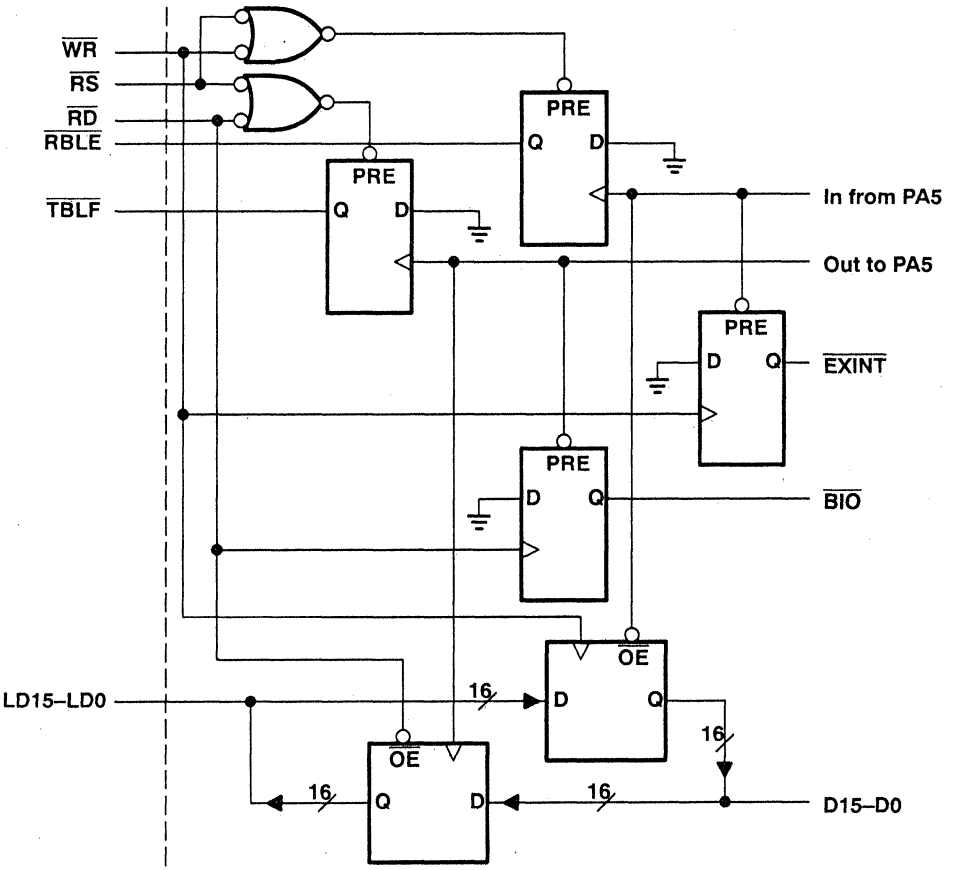



**Figure 24. TMS320C17/E17 Simplified Coprocessor Port Logic Diagram**

For full details of the coprocessor port refer to the *First Generation TMS320 User's Guide* (Reference [1]).

As an example this section considers an 8-bit interface, as may be required by a TMS7000 8-bit microcomputer.

Coprocessor mode is selected by setting both the MC/$\overline{PM}$ input (pin 27) and the MC input (pin 3) to low. Bit 30 in the TMS320C17 control register selects either a 16-bit or an 8-bit interface. This should be set to zero for an 8-bit interface. Connections to the TMS320C17 coprocessor port should be as shown in Figure 25.

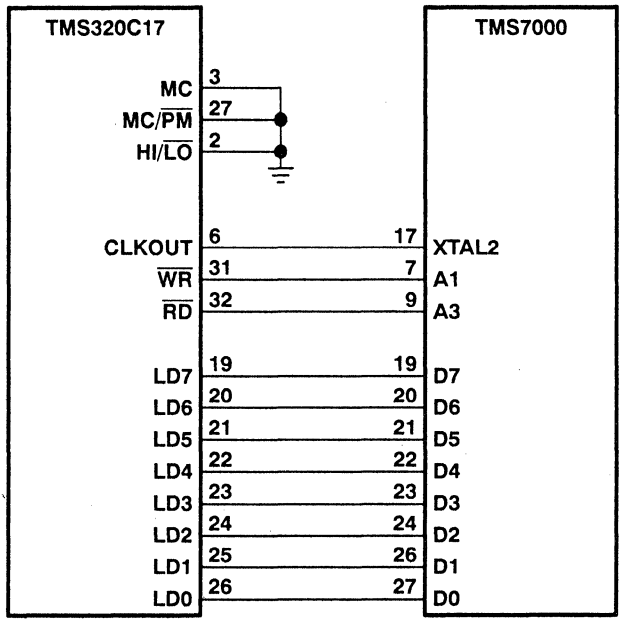



**Figure 25. TMS320C17 to 8-Bit Microcomputer (TMS7000) Interface**

The coprocessor port is accessed through I/O port 5 in the TMS320C17, and all parallel I/O IN and OUT instructions should be changed to access this port. In the listing file in Appendix A, IN instructions are from port 4 and OUT instructions are to either port 4 or port 6. All of these operations are within the interrupt handler section, INTHDL (see page 510 of the listing).

Data transfers in coprocessor mode operate on the same basis as presented in Host Write Cycle and Host Read Cycle sections, but the host CPU write and read sub-cycles operate differently. Transfers to the TMS320C17 operate as follows:

1. The $\overline{WR}$ signal is driven low by the microcomputer using a single I/O bit.
2. Data present on the LD7-LD0 bus is written to the receive buffer latch (D7-D0) when the WR signal is driven high by the microcomputer.
3. An internal $\overline{EXINT}$ signal is generated, causing the interrupt flag to be set in the TMS320C17.

4. The TMS320C17 responds to this interrupt condition in exactly the same way as the present code does, by executing the interrupt handler and executing an IN instruction (from port 5 in this case).

Transfers from the TMS320C17 use the following sequence:

1. The TMS320C17 writes 8 bits of data to the transmit buffer latch (D7-D0) with an OUT instruction to port 5.
2. At some point after this, the $\overline{RD}$ signal is driven low by the micro-computer using a single I/O bit.
3. Data is driven from the transmit buffer latch (D7-D0) to the LD7-LD0 bus until the RD signal is driven high by the microcomputer.

This interface may be further enhanced by implementing hardware handshaking between the TMS320C17 and the microcomputer, using the RBLE and TBLF signals from the TMS320C17.

***Use of DTMF Receiver or Tone Receiver in Isolation***

This application report describes an integrated DTMF and tone detection system. Both the DTMF receiver and tone receiver may separately be enabled or disabled (see Mode section), but the code for both is resident at all times. For any application requiring only the DTMF receiver function or only the general-tone function, ROM space can be saved by removing the unwanted code. Due to the complexity of functions such as time-stamping which are shared by both the DTMF receiver and the tone receiver, it is not feasible to describe a complete solution, but some of the major considerations are outlined below.

Note all subsequent page references are to the page number of the listing file given in Appendix A.

The DTMF code section can be removed from the program without significant modification. The DTMF code is very self-contained and is executed as a single block, with few external calls to subroutines within it. The test for the DTMF bit in the mode register should be removed from the end of the routine MAIN (see page 492). Calls to the DTMF reset routine RSDTMF should be removed from the cold reset routine (CRESET on page 514) and the self-test routine (SLFTST on page 516). The DTMF routine may then be removed completely (pages 504 to 510). DTMF constants may be removed, and the data memory locations they were loaded into used for other purposes. Care should be taken to ensure that the correct initialization of locations required by the tone receiver is not disturbed. The section in the warm reset routine (WRESET on page 514) which initializes DTMF data memory locations in page 1 should also be removed.

The tone receiver section is far more complex and cannot be removed as easily. Because the DTMF receiver relies upon certain of the ancillary functions of the tone receiver, these must be left intact. The routines which can be removed are:

| | |
|---|---|
| FILTER | (pages 496 to 499) |
| CHNGS | (page 501) |
| LVLS | (pages 501 and 502) |
| COMPLT | (page 502) |
| RSTFIL | (pages 502 and 503) |
| SQRT | (pages 503 and 504) |

Associated data memory locations and initialisation values may also be removed. Care should be taken to check all remaining sections of the code for references to code or memory locations which have been removed. This applies particularly to the following routines:

| | |
|---|---|
| CRESET | (pages 514) |
| WRESET | (pages 514 and 515) |
| SLFTST | (pages 516 and 517) |
| INTHDL | (pages 510 to 513) |
| ENVDET | (pages 494 and 495) |

It is recommended that these changes are not attempted without an in-circuit emulator for the TMS320C17. This can be used to trace program execution and, with its powerful hardware breakpoint facilities can readily debug the modified source code.

For anyone who wishes to investigate the possibility of customizing the code presented here and does not feel capable of taking on the development work, there is a U.K. company who may be willing to help on a consultancy basis:

Ensigma Ltd.
Archway House
Welsh Street
Chepstow
Gwent
NP6 5LL
Wales

Contacts:
Dr. Mike Carey
Adrian Anderson

Telephone: (44) 291 625422 (International)
0291 625422 (Within U.K.)

## Flexibility Through Programmability

Due to the programmability of the tone detector, this solution is not bound by the constraints of a custom hardware solution. Although the DTMF decoder performance is targeted to the CEPT recommendations, the tone receiver is dynamically re-programmable to suit a wide variety of incoming tones across a range of applications.

A simple tone detection system comprising no more than four chips may thus be controlled by a PC or a single chip 8-bit microcontroller to perform any of the tasks described by merely re-programming the on-chip registers of the TMS320C17.

## Conclusion

This report has presented a high-functionality DTMF and general tone decoder. The application as described has been fully tested and incorporated into a commercially available telephony peripheral.

Information has been presented which allows a designer to incorporate the tone detector function into a product. A full source listing is included in this report for customization. Performance characteristics for any customized version may vary from those given here.

The objective has been to describe both a particular implementation of the tone detector and provide a level of insight for further development. In order to keep this last part as simple as possible the mathematical detail has been kept to a minimum. If a detailed explanation of this aspect is required Ensigma Ltd. should be approached (see Use of DTMF Receiver or Tone Receiver in Isolation section).

## References


[1] *TMS320 First Generation Digital Signal Processors Data Sheet*, Texas Instruments Incorporated. Literature # SPRS009A, January 1987.

[2] *Understanding Telephone Electronics*, W.Sams Inc.

[3] *Technical Reference - Personal Computer AT*, International Business Machines Corporation, September 1985.

[4] *Technical Reference - Personal Computer XT*, revised edition, International Business Machines Corporation, March 1986.

[5] *Telecom Circuits Data Book*, Literature # SCTD001A, Texas Instruments Incorporated, 1987.

[6] *First Generation TMS320 User's Guide*, Literature # SPRT013A, Texas Instruments Incorporated, 1987.

# Appendix A
# Tone Detector Source Code

```
*************************************************************************
*
*    TMS320C17 SOURCE CODE FOR TONE DETECTOR MODULE
*
*    COPYRIGHT (c) TEXAS INSTRUMENTS June 1986, May 1987
*
*    WRITTEN BY ENSIGMA LTD.
*
*    REVISION 2.08 NOV 1988
*
*************************************************************************
*
*    ASSEMBLER EQUATES FOR TONE DETECTOR
*
*************************************************************************
*
*    PORT DEFINITIONS
*
CTLPRT    .set     0              ; CONTROL PORT
CTLUPR    .set     1              ; UPPER CONTROL PORT
CDCPRT    .set     1              ; CODEC PORT
ATTPRT    .set     2              ; STATUS ATTENTION PORT
PRMPRT    .set     3              ; SIMULATOR INPUT PORT
DATPRT    .set     4              ; DATA PORT
STAPRT    .set     6              ; STATUS READ PORT
SIMPRT    .set     7              ; SIMULATOR FLAG PORT
*
*    FLAG POSITIONS IN FLAGS REGISTER
*
TPRFLG    .set     15             ; TONE PRESENT
ONSFLG    .set     14             ; ONSET TIME VALID
FSTFLG    .set     13             ; FIRST BLOCK OF FILTERING
STAFLG    .set     12             ; INTERRUPT HANDLER STATE BIT
DTMFLG    .set     11             ; DTMF ON FLAG
INTFLG    .set     10             ; INTERRUPT HAS OCCURRED
*
RES1      .set     7              ; RESERVED
RES2      .set     6              ; RESERVED
FILT1     .set     5              ; LEVEL 1 ABOVE CHANGE THR
FILT2     .set     4              ; LEVEL 2 ABOVE CHANGE THR
FILT3     .set     3              ; LEVEL 3 ABOVE CHANGE THR
FILT4     .set     2              ; LEVEL 4 ABOVE CHANGE THR
FILT5     .set     1              ; LEVEL 5 ABOVE CHANGE THR
FILT6     .set     0              ; LEVEL 6 ABOVE CHANGE THR
*
PAGE1     .set     080h           ; ADDRESS OFFSET FOR DATA PAGE 1
*
CPCBIT    .set     8              ; PORT 1 CONTROL BIT
CEFBIT    .set     9              ; EXTERNAL FRAMING BIT
CXFBIT    .set     10             ; XF OUTPUT LATCH BIT
CSPBIT    .set     11             ; SERIAL PORT ENABLE BIT
CEEBIT    .set     12             ; COMPANDER ENCODE ENABLE
CDEBIT    .set     13             ; COMPANDER DECODE ENABLE
CUABIT    .set     14             ; F-LAW, A-LAW SELECT BIT
CSCBIT    .set     15             ; SERIAL CLOCK CONTROL BIT
*
*    MODE REGISTER BITS
*
RC0       .set     0              ; SELF TEST RESULT
RC1       .set     1              ; SELF TEST RESULT
*         .set     2              ; RESERVED
*         .set     3              ; RESERVED
*         .set     4              ; RESERVED
TONEBT    .set     5              ; TONE DETECTOR ON/OFF BIT
DTMFBT    .set     6              ; DTMF DETECTOR ON/OFF BIT
TESTBT    .set     7              ; SELFTEST ONLY ON/OFF BIT
*
*    STATUS REGISTER BITS
*
RDRDY     .set     8              ; NA (interface only)
WRRDY     .set     9              ; NA (interface only)
DPINBT    .set     10             ; Tone Depart interrupt
OSINBT    .set     11             ; Tone Onset interrupt
CHINBT    .set     12             ; Tone Change interrupt
DTINBT    .set     13             ; DTMF Digit interrupt
STINBT    .set     14             ; Short Tone interrupt
*         .set     15             ; Reserved
*
*    EQUATES FOR SELF TESTS
*
ROMFAI    .set     3
RAMFAI    .set     2
PASS      .set     1
CDCFAI    .set     0
ACCHLD    .set     0
TOTAL     .set     1
ROMVAL    .set     2
*
*    (THE REGISTER MAPPING TABLE IS USED TO CONVERT INTERFACE COMMANDS TO
*    TMS320 DATA ADDRESSES.) FLAGS USED IN REGISTER MAPPING TABLE TO INDICATE
*    ACCESSES REQUIRING SPECIAL PROCESSING. THE FUNCTION OF EACH BIT IS
*    DESCRIBED BELOW.
*
L         .set     512
U         .set     1024
T         .set     2048
F         .set     4096
S         .set     8192
M         .set     16384
*
*    SHIFTS FOR TESTING THE REGISTER MAPPING BITS
*
RWBIT     .set     4              ; READ ACCESS OF REGISTER
LBIT      .set     9              ; ACCESS OF ADDRESS 0 OR 1
UBIT      .set     10             ; ACCESS OF AN UPPER BYTE
TBIT      .set     11             ; READ OF CURRENT TIME
FBIT      .set     12             ; WRITE OF FREQUENCY LOWER BYTE
SBIT      .set     13             ; READ OF STATUS REGISTER
```

```
MBIT    .set    14              ; WRITE TO MODE REGISTER
*
*   ALLOW BETWEEN 515 AND 715 IDLE CYCLES BETWEEN CODEC INTERRUPTS
*
INTMAX  .set    077h            ; MAXIMUM AND MINIMUM NUMBER OF VALID
INTMIN  .set    056h            ; TRANSITS OF LOOPB IF CODEC IS INTER-
*                               ; RUPTING PROPERLY
*
*   LOOP LENGTH IS 6 CYCLES
*
INTLFT  .set    (INTMAX-INTMIN) ; MINIMUM NUMBER OF REMAINING TRANSITS
*
*****************************************************************
*
*   ASSEMBLER EQUATES FOR DTMF PROGRAM
*
*****************************************************************
*
SD      .set    06A0h           ; SCALING FACTOR FOR INPUT
Z       .set    0380h           ; SCALING FACTOR FOR THRESHOLDS
SCNT    .set    120             ; MINIMUM SAMPLE COUNT (30MS)
MINTH   .set    0DAh            ; MINIMUM SIGNAL LEVEL
TWSTLO  .set    640             ; 2.5 * 2**8
TWSTHI  .set    2048            ; 8 * 2**8
THRHL1  .set    034h            ; THRESHOLD COUNT FOR 2ND ORDER
THRHL2  .set    02Dh
THRHL3  .set    02Bh
THRHL4  .set    02Dh
THRHH1  .set    029h
THRHH2  .set    027h
THRHH3  .set    024h
THRHH4  .set    022h
LOLIM   .set    14              ; MAX OVERSPILL LO BAND
HILIM   .set    7               ; MAX OVERSPILL HI BAND
MIN     .set    7               ; IDLE LINE DETECT
*
*****************************************************************
*
*   PAGE 0 DATA DEFINITIONS FOR TONE DETECTOR
*
*****************************************************************
*
*   UNINITIALIZED VARIABLES FOR TONE DETECTOR
*
*****************************************************************
        .bss    DUM,033h        ; DUMMY RAM LOCATIONS
        .bss    ONE,1           ; UNITY
        .bss    TEMP,1          ; SCRATCH LOCATION
        .bss    TEMP1,1         ; SCRATCH LOCATION
        .bss    TEMP2,1         ; SCRATCH LOCATION
        .bss    SAMPLE,1        ; CURRENT LINEARIZED INPUT SAMPLE SHARED
*                               ; WITH DTMF
*
*   THE FOLLOWING LOCATION MUST BE AT AN ADDRESS ENDING IN 8
*
        .bss    QUEUE,8         ; CIRCULAR QUEUE FOR LINEARIZED INPUT
*                               ; SAMPLES
        .bss    TEMP3,1         ; SCRATCH LOCATION
        .bss    ITEMP,1         ; INTERRUPT HANDLER SCRATCH LOCATION
        .bss    CMSAVE,1        ; INTERRUPT HANDLER COMMAND BYTE SAVE
        .bss    ARSAVE,1        ; INTERRUPT HANDLER AUXILIARY REGISTER 0
*                               ; SAVE
        .bss    WINDOW,1        ; WINDOW SAMPLE READ FROM ROM TABLE. 2**15
*                               ; FORM
        .bss    SNCWIN,1        ; SIN(X)/X . WINDOW PRODUCT. 2**15 FORM
        .bss    FILPOS,1        ; CURRENT BLOCK FILTERING POSITION [COUNTS
*                               ; UP FROM -16384 TO +16384 IN STEPS OF
*                               ; (2FL + 32)]
        .bss    ACWNHI,1        ; HIGH WORD OF 32BIT WINDOW ACCUMULATORS
        .bss    ACWNLO,1        ; LOW WORD OF 32BIT WINDOW ACCUMULATORS
        .bss    ACSWHI,1        ; HIGH WORD OF 32BIT SIN(X)/X. WINDOW
*                               ; PRODUCT ACCUMULATORS
        .bss    ACSWLO,1        ; LOW WORD OF 32BIT SIN(X)/X. WINDOW
*                               ; PRODUCT ACCUMULATORS
*
        .bss    ACSQHI,1        ; HIGH WORD OF 32BIT SIGNAL SQUARED
*                               ; ACCUMULATORS
        .bss    ACSQLO,1        ; LOW WORD OF 32BIT SIGNAL SQUARED
*                               ; ACCUMULATORS
*
*****************************************************************
*
*   INITIALIZED VARIABLES FOR TONE DETECTOR
*
*****************************************************************
*
        .bss    IVAR1,0
        .bss    CTL320,1        ; INITIAL VALUE FOR TMS320 CONTROL REGISTER
*                               ; (LOWER)
        .bss    CTL322,1        ; SECOND VALUE FOR TMS320 CONTROL REGISTER
*                               ; (LOWER)
        .bss    CTL32U,1        ; VALUE FOR UPPER CONTROL PORT
        .bss    QIN,1           ; POINTS TO NEXT FREE QUEUE INPUT LOCATION
        .bss    QOUT,1          ; POINTS TO NEXT AVAILABLE LINEARIZED
*                               ; SAMPLE ON QUEUE
        .bss    CORREC,1        ; CORRECTION FACTOR FOR SINE AND COS =
*                               ; 1/0.9050 * 2**12
        .bss    K1,1            ; CONSTANT USED IN THE SINE COSINE ROUTINE.
        .bss    SCALEF,1        ; SCALE FACTOR FOR SCALING A-LAW LINEARIZED
*                               ; INPUT SAMPLE INTO AN OPTIMAL NUMBER
*                               ; RANGE. THE INTERNAL SAMPLE HAS THE VALUE
*                               ; (SCALEF * 2.25 * LINEARIZED(ALAW) /4).
*                               ; THE STARTUP VALUE FOR SCALEF IS 4, BUT
*                               ; TWO OF THE BITS IN THE MODE REGISTER MAY
*                               ; BE USED TO SELECT SCALEF = 1,4, OR 16.
*                               ; SCALEF = 4 GIVES TONE DETECTOR DYNAMIC
*                               ; RANGE SUCH THAT OUTPUT OF 254 == -10dBm0.
*                               ; THE OTHER SCALEF VALUES MOVE THE DYNAMIC
```

```
*                               ; RANGE BY 12dBmO IN EACH DIRECTION.
*
        .bss    MSOOFF,1        ; BIT MASK VALUE >OOFF
*
*****************************************************************
*
*   REGISTERS FOR COMMUNICATION WITH INTERFACE
*
*****************************************************************
*
        .bss    STMODE,1        ; STATUS REGISTER (UPPER) AND MODE REGISTER
*                               ; (LOWER)
        .bss    FLSTOR,1        ; LAUUE OF FL SAVED TO PREVENT UPDATE
*                               ; DURING FILTERING
        .bss    UPRTHR,1        ; UPPER ENVELOPE DETECTOR THRESHOLD IN
*                               ; LOWER BYTE
        .bss    LWRTHR,1        ; LOWER ENVELOPE DETECTOR THRESHOLD IN
*                               ; LOWER BYTE
        .bss    EDFCT,1         ; ENVELOPE DECAY FACTOR (UPPER) CHANGE
*                               ; THRESHOLD (LOWER)
        .bss    FMSFL,1         ; UPPER BYTE OF DETECTOR CENTER FREQUENCY
*                               ; IN UPPER, AND FL IN LOWER BYTE. FILTER
*                               ; LENGTH IS (4 * FL + 1)
*
        .bss    FSPW,1          ; FILTER SELECT (UPPER) AND PASSBAND WIDTH
*                               ; (LOWER)
*
*                               ; IEND1-IVAR1 IS THE LENGTH OF THE PAGE 0
*                               ; INITIALIZED VARIABLES SECTION
*
*****************************************************************
*
*   UNINITIALIZED VARIABLES FOR TONE DETECTOR
*
*****************************************************************
*
        .bss    IEND1,0
        .bss    OSTIME,1
*
*   MODULO 65536 AND RELATIVE TO END OF LAST RESET
*
        .bss    DPTIME,1        ; TONE DEPART TIME IN MS
        .bss    CRTIME,1        ; CURRENT TIME IN MS
        .bss    OSHOLD,1        ; ONSET TIME LATCH REGISTER
        .bss    CRHOLD,1        ; CURRENT TIME LATCH REGISTER
*
*   DETECTED SIGNAL LEVELS
*
        .bss    LVL12,1         ; FILTER1 (UPPER) FILTER2 (LOWER)
        .bss    LVL34,1         ; FILTER3 (UPPER) FILTER4 (LOWER)
        .bss    LVL56,1         ; FILTER5 (UPPER) FILTER6 (LOWER)
*
        .bss    ENVEL,1         ; SMOOTHED SIGNAL ENVELOPE
        .bss    CTLTSL,1        ; CONTROL REGISTER (UPPER) AND TOTAL SIGNAL
*                               ; LEVEL (LOWER)
        .bss    FLAGS,1         ; MULTIPLE FLAG REGISTER. ALL FLAGS HIGH
*                               ; ASSERTED.
*                               ; SEE ASSEMBLER EQUATES FOR FLAG
*                               ; DEFINITIONS. UNUSED BITS READ AS ZERO.
*
        .bss    SINCPH,1        ; PHASE FOR SIN(X)/X FUNCTION
*
*****************************************************************
*
*   VARIABLES USED IN FILTER ROUTINE, CONTINUE INTO PAGE 1
*   DO NOT INSERT OR DELETE ANY VARIABLES AFTER THIS POINT
*
*****************************************************************
*
        .bss    FREQ1,1         ; FREQN = (FILTER N CENTER FREQUENCY) /
*                               ; 0.12207
        .bss    PHASE1,1        ; PHASE OF FREQN GENERATOR
        .bss    COS1HI,1        ; HIGH WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss    COS1LO,1        ; LOW WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss    SIN1HI,1        ; HIGH WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
        .bss    SIN1LO,1        ; LOW WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
*
        .bss    FREQ2,1         ; FREQN = (FILTER N CENTER FREQUENCY) /
*                               ; 0.12207
        .bss    PHASE2,1        ; PHASE OF FREQN GENERATOR
        .bss    COS1H2,1        ; HIGH WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss    COS2LO,1        ; LOW WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss    SIN2HI,1        ; HIGH WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
        .bss    SIN2LO,1        ; LOW WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
*
        .bss    FREQ3,1         ; FREQN = (FILTER N CENTER FREQUENCY) /
*                               ; 0.12207
        .bss    PHASE3,1        ; PHASE OF FREQN GENERATOR
        .bss    COS3HI,1        ; HIGH WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss    COS3LO,1        ; LOW WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss    SIN3HI,1        ; HIGH WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
        .bss    SIN3LO,1        ; LOW WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
*
        .bss    FREQ4,1         ; FREQN = (FILTER N CENTER FREQUENCY) /
*                               ; 0.12207
        .bss    PHASE4,1        ; PHASE OF FREQN GENERATOR
```

```
        .bss    COS4HI,1        ; HIGH WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss    COS4LO,1        ; LOW WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss    SIN4HI,1        ; HIGH WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
*
*****************************************************************
*
*   PAGE 1 DATA DEFINITIONS FOR TONE DETECTOR.
*
*****************************************************************
*
*   UNINITIALIZED VARIABLES FOR TONE DETECTOR (CONTINUED)
*
*****************************************************************
*
*   THIS LOCATION MUST REPRESENT THE BOUNDARY BETWEEN PAGE 0 AND PAGE 1
*   I.E., DATA MEMORY LOCATION 0080h
*
*****************************************************************
*
*                               ; CONTINUED VARIABLES USED IN FILTER
*                               ; ROUTINE
        .bss    SIN4LO,1        ; LOW WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
*
        .bss    FREQ5,1         ; FREQN = (FILTER N CENTER FREQUENCY) /
*                               ; 0.12207
        .bss    PHASE5,1        ; PHASE OF FREQN GENERATOR
        .bss    COS5HI,1        ; HIGH WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss    COS5LO,1        ; LOW WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss    SIN5HI,1        ; HIGH WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
        .bss    SIN5LO,1        ; LOW WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
*
        .bss    FREQ6,1         ; FREQN = (FILTER N CENTER FREQUENCY) /
*                               ; 0.12207
        .bss    PHASE6,1        ; PHASE OF FREQN GENERATOR
        .bss    COS6HI,1        ; HIGH WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss    COS6LO,1        ; LOW WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss    SIN6HI,1        ; HIGH WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
        .bss    SIN6LO,1        ; LOW WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
*
        .bss    SRSAVE,1        ; INTERRUPT HANDLER STATUS REGISTER SAVE
        .bss    ACCUHI,1        ; HIGH WORD OF INTERRUPT HANDLER
*                               ; ACCUMULATOR SAVE
        .bss    ACCULO,1        ; LOW WORD OF INTERRUPT HANDLER ACCUMULATOR
*                               ; SAVE
*
*****************************************************************
*
*   UNINITIALIZED VARIABLES OF THE DTMF PROGRAM (PAGE 1)
*
*****************************************************************
*
        .bss    UNITY,1
        .bss    DECIM,1         ; DECIMATION FLAG
        .bss    PAUSE,1         ; WE-ARE-IN-THE-GAP FLAG
        .bss    X1,1            ; NEWEST LINEAR DATA SAMPLE
        .bss    GN1,1           ; HIGHPASS/NOTCH-FILTER
        .bss    GN2,1           ; SAMPLE DELAYS
        .bss    Y5,1            ; HIGHPASS/NOTCH-FILTER OUTPUT
        .bss    GAP,1           ; GAP-MUST-FOLLOW FLAG
        .bss    SIGCNT,1        ; SAMPLE COUNTER
        .bss    STOP,1          ; FLAG FOR AGC/RWIST/TONE DECODE
        .bss    SEMA,1          ; FLAG SEMAPHORE FOR THE DECIMATION
        .bss    X,1             ; NEWEST CODEC SAMPLE
*
*   FILTER DELAY SAMPLES
*
        .bss    L1N1,1
        .bss    L1N2,1
        .bss    L2N1,1
        .bss    L2N2,1
*
*   LOW-BAND 8TH ORDER BANDPASS
*
        .bss    L3N1,1
        .bss    L3N2,1
        .bss    L4N1,1
        .bss    L4N2,1
*
*   LOW-BAND 8TH ORDER FILTER OUTPUT
*
        .bss    LY,1
*
*   LOW-BAND 2ND ORDER SUB-FILTER OUTPUTS
*
        .bss    LY1,1
        .bss    LY2,1
        .bss    LY3,1
        .bss    LY4,1
*
        .bss    F1,1            ; 697 HZ
        .bss    F2,1            ; 770 HZ
        .bss    F3,1            ; 852 HZ
        .bss    F4,1            ; 941 HZ
*
*   FILTER DELAY SAMPLES HIGH-BAND 8TH ORDER BANDPASS
*
```

```
        .bss    H1N1,1
        .bss    H1N2,1
        .bss    H2N1,1
        .bss    H2N2,1
        .bss    H3N1,1
        .bss    H3N2,1
        .bss    H4N1,1
        .bss    H4N2,1
*
*   HIGH-BAND 8TH ORDER FILTER OUTPUT
*
        .bss    HY,1
*
*   HIGH-BAND 2ND ORDER SUB-FILTER OUTPUTS
*
        .bss    HY1,1
        .bss    HY2,1
        .bss    HY3,1
        .bss    HY4,1
*
        .bss    F5,1            ; 1209 HZ
        .bss    F6,1            ; 1336 HZ
        .bss    F7,1            ; 1477 HZ
        .bss    F8,1            ; 1633 HZ
        .bss    TEMPD,1         ; SCRATCH-PAD REGISTER
        .bss    ADJL,1          ; HIGH-BAND THRESHOLD ADJUST VALUE
        .bss    ADJH,1          ; LOW-BAND THRESHOLD ADJUST VALUE
        .bss    CNTR1,1         ; SCRATCH COUNTER
        .bss    CNTR2,1         ; SCRATCH COUNTER
        .bss    CNTR,1          ; SAMPLE COUNTER FOR GAP SEARCH
        .bss    TESTG,1         ; GOOD DIGITS TEST COUNT
        .bss    TESTB,1         ; BAD DIGITS TEST COUNT
        .bss    TMP,1           ; TEMP REGISTER FOR CODEC SETTING
*
*****************************************************************
*
*   INITIALIZED VARIABLES OF THE DTMF PROGRAM (PAGE 1)
*
*****************************************************************
*
*   HIGHPASS/NOTCH-FILTER COEFFICIENTS
*
        .bss    IVAR0,0
        .bss    A1,1
        .bss    A2,1
        .bss    B0,1
        .bss    B1,1
        .bss    B2,1
*
*   FILTER COEFFICIENTS LOW-BAND 8TH ORDER BANDPASS
*
        .bss    L1C,1
        .bss    L1D,1
        .bss    L2C,1
        .bss    L2D,1
        .bss    L3C,1
        .bss    L3D,1
        .bss    L4C,1
        .bss    L4D,1
*
*   FILTER COEFFICIENTS HIGH-BAND 8TH ORDER BANDPASS
*
        .bss    H1C,1
        .bss    H1D,1
        .bss    H2C,1
        .bss    H2D,1
        .bss    H3C,1
        .bss    H3D,1
        .bss    H4C,1
        .bss    H4D,1
*
        .bss    THRSH8,1        ; 8TH ORDER THRESHOLD VALUE
        .bss    THRHI,1         ; 2ND ORDER THRESHOLD HIGH BAND
        .bss    THRLO,1         ; 2ND ORDER THRESHOLD LOW BAND
        .bss    DVLOW,1         ; STROBE LOW MASK
        .bss    DIGIT,1         ; DIGIT OUTPUT REGISTER
        .bss    MINTHH,1        ; ALTERNATIVE MINTH FOR LOW SIGNAL LEVELS
        .bss    MINTHL,1
*
*****************************************************************
*
*   INITIALIZED VARIABLE FOR CHECKSUM ROUTINE
*
*****************************************************************
*
        .bss    MARKER,1        ; REGISTER TO HOLD PROGRAM END ADDRESS
*
        .bss    IEND0,0
*
*****************************************************************
*
*   VECTORS AND CONSTANTS FOR TONE DETECTOR
*
*****************************************************************
*
        .text
*
        B       CRESET          ; COLD RESET VECTOR
0       B       INTHDL          ; INTERRUPT VECTOR
*
*****************************************************************
*
*   THIS IS THE TABLE OF CONSTANTS WHICH ARE NOT COPIED INTO RAM
*
*****************************************************************
*
CHECKS  .word   02CACh          ; ROM CHECKSUM LOCATION.
SXCENT  .word   23291           ; CENTRAL MAXIMUM OF SIN(X)/X FUNCTION
```

```
*
*****************************************************************
*
*    TABLE OF PAGE 1 DTMF CONSTANTS FOR COPYING TO DATA RAM BY RESET HANDLER
*
*****************************************************************
*
CONST0    .set      $
*
          .word     -27801          ; -30307
          .word     -13617          ; -13952
          .word     +20767          ; +18463
          .word     -28968          ; -25764
          .word     +20767          ; +18427
          .word     03880h          ; CENTER:705 BW:8
          .word     0832Dh
          .word     +11402
          .word     -31755          ; COEFFICIENTS FOR LOW BANDPASS
          .word     +7432
          .word     -31755
          .word     +2985
          .word     -31755
          .word     0D5D6h
          .word     085E4h          ; 1219/15
          .word     0C234h
          .word     087CCh          ; 1332/20
          .word     0AACEh          ; COEFFICIENTS FOR HIGH BANDPASS
          .word     087CCh          ; 1482/20
          .word     09853h
          .word     087CCh          ; 1630/20
          .word     07FFFh          ; THRESHOLD FOR 8TH ORDER OUTPUT
          .word     07FFFh          ; THRESHOLD FOR 2ND ORDER HIGH
          .word     07FFFh          ; THRESHOLD FOR 2ND ORDER LOW
          .word     0FF7Fh          ; MASK FOR DATA VALID STROBE
          .word     070h            ; OUTPUT DIGIT (INIT. INVALID)
          .word     MINTH           ; INITIAL VALUE FOR MINTHH
          .word     0               ; INITIAL VALUE FOR MINTHL
          .word     PRGEND
*
CONEN0    .set      $
*
*****************************************************************
*
*    TABLE OF PAGE 0 TONE DETECTOR CONSTANTS FOR COPYING TO DATA RAM BY RESET
*    HANDLER
*
*****************************************************************
*
CONST1    .set      $
*
          .word     0FD9Fh          ; CTL320
          .word     07C90h          ; CTL322
          .word     0CFEh           ; CTL32U
          .word     QUEUE           ; QIN
          .word     QUEUE           ; QOUT
*
          .word     011AEh          ; CORREC 0.9050 * 2**12
          .word     059A8h          ; K1     1.4008687 * 2**14
          .word     04h             ; SCALEF 4 (DEFAULTS TO 254==-10dBm0)
          .word     0FFh            ; MSOOFF
*
*****************************************************************
*
*    REGISTERS FOR COMMUNICATION WITH INTERFACE
*
*****************************************************************
*
          .word     0FF00h          ; STMODE
          .word     032h            ; FLSTOR
          .word     0FFh            ; UPRTHR
          .word     0FFh            ; LWRTHR
          .word     07800h          ; EDFCT
          .word     032h            ; FMSFL
          .word     03F00h          ; FSPW
*
CONEN1    .set      $
*
*****************************************************************
*
*    THIS IS THE TABLE OF MAPPINGS BETWEEN THE 16 LOGICAL READ AND WRITE BYTE
*    ADDRESSES IN THE INTERFACE AND THE PHYSICAL WORD LOCATIONS IN THE
*    TMS320C17
*
*****************************************************************
*
INTTAB    .set      $
*
*****************************************************************
*
*    WRITE REGISTERS
*
*****************************************************************
*
*     ;    Physical locations:  Logical locations:
*
          .word     (L+U+CTLTSL)    ; CONTROL
          .word     (L+H+STMODE)    ; MODE
          .word     (U+EDFCT)       ; ENVELOPE DECAY FACTOR
          .word     (UPRTHR)        ; UPPER THRESHOLD
          .word     (LWRTHR)        ; LOWER THRESHOLD
          .word     (FMSFL)         ; FILTER LENGTH SPECIFIER
          .word     (FSPW)          ; PASSBAND WIDTH SPECIFIER
          .word     (EDFCT)         ; CHANGE THRESHOLD
          .word     (U+FMSFL)       ; FREQUENCY MS BYTE
          .word     (F+FREQ1)       ; FILTER1 CENTER FREQUENCY LS BYTE
          .word     (F+FREQ2)       ; FILTER2 CENTER FREQUENCY LS BYTE
          .word     (F+FREQ3)       ; FILTER3 CENTER FREQUENCY LS BYTE
          .word     (F+FREQ4)       ; FILTER4 CENTER FREQUENCY LS BYTE
```

```
        .word   (F+FREQ5)       ; FILTER5 CENTER FREQUENCY LS BYTE
        .word   (F+FREQ6)       ; FILTER6 CENTER FREQUENCY LS BYTE
        .word   (U+FSPW)        ; FILTER WIDTH SELECT
*
*****************************************************************
*
*   READ REGISTERS
*
*****************************************************************
*
* ;     Physical locations:     Logical locations:
*
        .word   (L+S+STMODE)    ; STATUS
        .word   (L+STMODE)      ; MODE
        .word   (DIGIT)         ; DTMF DIGIT
        .word   (U+OSTIME)      ; TONE ARRIVAL (MS)
        .word   (OSTIME)        ; TONE ARRIVAL (LS)
        .word   (U+DPTIME)      ; TONE DEPARTURE (MS)
        .word   (DPTIME)        ; TONE DEPARTURE (LS)
        .word   (T+CRTIME)      ; CURRENT TIME (MS)
        .word   (CRHOLD)        ; CURRENT TIME (LS)
        .word   (U+LVL12)       ; FILTER1 SIGNAL LEVEL
        .word   (LVL12)         ; FILTER2 SIGNAL LEVEL
        .word   (U+LVL34)       ; FILTER3 SIGNAL LEVEL
        .word   (LVL34)         ; FILTER4 SIGNAL LEVEL
        .word   (U+LVL56)       ; FILTER5 SIGNAL LEVEL
        .word   (LVL56)         ; FILTER6 SIGNAL LEVEL
        .word   (CTLTSL)        ; TOTAL SIGNAL LEVEL
*
*****************************************************************
*
*   LOOKUP TABLE FOR CONVERTING 2 SC MODE BITS INTO A SCALE FACTOR
*
*****************************************************************
*
SCATAB  .set    $
*
        .word   4               ; SCALEF=4 (DEFAULT)
        .word   4               ; SCALEF=4          -10dBm0 RANGE
        .word   1               ; SCALEF=1           +2dBm0 RANGE
        .word   16              ; SCALEF=16         -22dBm0 RANGE
*
*****************************************************************
*
*   LOOKUP TABLE FOR CONVERTING 3 IACK MODE BITS INTO THE EQUIVALENT STATUS
*   BIT TO BE SET BY AN ACKNOWLEDGE.
*
*****************************************************************
*
ACKTAB  .set    $
*
        .word   0               ; IACK = 0
        .word   0               ; IACK = 1
        .word   1024            ; IACK = 2
        .word   2048            ; IACK = 3
        .word   4096            ; IACK = 4
        .word   8192            ; IACK = 5
        .word   16384           ; IACK = 6
        .word   0               ; IACK = 7
*
*****************************************************************
*
*   THIS IS THE TABLE OF WINDOW COEFFICIENTS. ONLY HALF OF THE WINDOW IS
*   STORED, STARTING IN THE MIDDLE.
*
*****************************************************************
*
WINTAB  .word   32767           ; CENTRAL COEFFICIENT.
        .word   32763
        .word   32749           ; LENGTH OF HALF-WINDOW = 129
        .word   32727
        .word   32696
        .word   32655
        .word   32607
        .word   32549
        .word   32482
        .word   32407
        .word   32323
        .word   32230
        .word   32129
        .word   32019
        .word   31901
        .word   31774
        .word   31639
        .word   31496
        .word   31345
        .word   31186
        .word   31019
        .word   30844
        .word   30662
        .word   30472
        .word   30274
        .word   30069
        .word   29857
        .word   29638
        .word   29412
        .word   29180
        .word   28940
        .word   28694
        .word   28442
        .word   28184
        .word   27920
        .word   27650
        .word   27374
        .word   27093
        .word   26807
        .word   26515
        .word   26218
```

```
        .word   25917
        .word   25611
        .word   25301
        .word   24987
        .word   24668
        .word   24346
        .word   24020
        .word   23691
        .word   23358
        .word   23022
        .word   22684
        .word   22343
        .word   21999
        .word   21654
        .word   21306
        .word   20956
        .word   20605
        .word   20252
        .word   19898
        .word   19543
        .word   19187
        .word   18830
        .word   18473
        .word   18115
        .word   17758
        .word   17400
        .word   17043
        .word   16686
        .word   16330
        .word   15975
        .word   15621
        .word   15268
        .word   14916
        .word   14566
        .word   14218
        .word   13871
        .word   13526
        .word   13184
        .word   12844
        .word   12506
        .word   12172
        .word   11839
        .word   11510
        .word   11184
        .word   10861
        .word   10541
        .word   10225
        .word   9913
        .word   9604
        .word   9299
        .word   8998
        .word   8701
        .word   8408
        .word   8119
        .word   7834
        .word   7554
        .word   7279
        .word   7008
        .word   6741
        .word   6480
        .word   6223
        .word   5971
        .word   5723
        .word   5481
        .word   5244
        .word   5012
        .word   4784
        .word   4562
        .word   4345
        .word   4134
        .word   3927
        .word   3725
        .word   3529
        .word   3338
        .word   3152
        .word   2972
        .word   2796
        .word   2626
        .word   2461
        .word   2301
        .word   2146
        .word   1996
        .word   1851
        .word   1712
        .word   1577
        .word   1448
        .word   1323
        .word   1203
*
ENDWIN  .set    $               ; END OF TABLE OF WINDOW COEFFICIENTS
*
;*****************************************************************
*
*   ROUTINE: MAIN
*
*   REFERENCE IN FLOWCHART: READ QUEUE
*                           INCREMENT TIME
*                           SCALE
*                           DTMF
*
*   FUNCTION: READ SAMPLE FROM INPUT QUEUE, AND UPDATE CURRENT TIME. SCALE
*             THE SAMPLE AND CALL DTMF IF IT IS SWITCHED ON.
*
;*****************************************************************
*
MAIN    .set    $               ; READ NEXT SAMPLE FROM QUEUE INTO SAMPLE.
*                               ; INCREMENT CURRENT TIME EVERY MS.
*
```

```
READQ   LAC     QIN             ; WAIT FOR SOMETHING ON QUEUE. THE QUEUE IS
        SUB     QOUT            ; EMPTY WHEN THE QUEUE INPUT POINTER EQUALS
        BZ      READQ           ; THE QUEUE OUTPUT POINTER.
*
QNEMPT  LAR     AR0,QOUT        ; LOAD AR0 WITH QUEUE OUTPUT POINTER
*
        LAC     *,0             ; READ SAMPLE FROM QUEUE
*
        BGEZ    POSSMP
*
        ADD     ONE,15          ; CONVERT FROM SIGNED-MAGNITUDE NEGATIVE
        SACL    SAMPLE          ; TO TWOS-COMPLEMENT NEGATIVE
        SUB     SAMPLE,1
POSSMP  SACL    SAMPLE          ; STORE IN SAMPLE
*
        LAC     QOUT
        SUB     ONE             ; DECREMENT OUTPUT POINTER MODULO 8
        SACL    TEMP
*
        LAC     ONE,3
        OR      TEMP            ; POINTER COUNTS 002Fh THROUGH 0028h WHERE
        SACL    QOUT            ; THE QUEUE STARTS AT 0028h
*
        LACK    QUEUE
        SUB     QOUT            ; EVERY TIME THE QUEUE OUTPUT POINTER GETS
        BNZ     SCALE           ; TO 0028h, ONE MS HAS ELAPSED.
*
INCR    ZALS    CRTIME          ; INCREMENT CURRENT TIME BY ONE,
*                               ; MODULO 65536
        ADD     ONE
        SACL    CRTIME
*
*****************************************************************
*
*   SCALE SAMPLE INTO WORKING RANGE. THE WORKING RANGE IS SET SO THAT NONE OF
*   THE ACCUMULATORS IN THE TONE DETECTOR WILL OVERFLOW UNDER ANY SIGNAL
*   CONDITIONS. THE PEAK-TO-PEAK SINUSOIDAL SWITCH WHICH CAUSES A FULL SCALE
*   (254) READING IN THE TOTAL SIGNAL OUTPUT REGISTER, HAS AN INTERNAL
*   AMPLITUDE OF 2030. WHEN THE DEFAULT FACTOR OF 4 IS SELECTED, 2030
*   CORRESPONDS TO AN INPUT SIGNAL LEVEL OF -10 dBm0. THE OTHER POSSIBLE
*   SCALE FACTORS SHIFT THIS VALUE BY 12dB EITHER WAY. SOFTWARE LIMITING OF
*   THE INTERNAL SIGNAL LEVEL OCCURS AT q 8191, WHICH, FOR THE DEFAULT SCALE
*   FACTOR, CORRESPONDS TO A SIGNAL LEVEL OF +2.1 dBm0. THIS LEVEL IS ALSO
*   SHIFTED BY 12 dB EITHER WAY BY SELECTING THE OTHER SCALE FACTORS, HOWEVER
*   THE CODEC WILL CLIP ANY SIGNALS LARGER THAN +3 dBm0.
*
*****************************************************************
*
SCALE   .set    $
*
        ZALH    SAMPLE          ; LOAD SAMPLE INTO HIGH ACCUMULATOR
        ADD     SAMPLE,13
        SACH    SAMPLE,1        ; 2.25 * SAMPLE IN SAMPLE
*
        LT      SAMPLE
        MPY     SCALEF          ; MULTIPLY SAMPLE BY SCALE FACTOR
*                               ; (1,4 OR 16)
        PAC
        SACH    TEMP2           ; SAVE SIGN
        SACL    TEMP1
*
        ABS                     ; CHECK FOR OVERFLOW OF THE <4 * 8192
        SUB     ONE,15          ; LIMIT BY SUBTRACTING 32768
        BLZ     SIZOK
*
OVRLOD  LAC     TEMP2
        SACH    TEMP2           ; EXTRACT PURE SIGN OF SAMPLE
*
        LAC     ONE,15          ; LOAD UP 32768
        SUB     ONE             ; 32767
        XOR     TEMP2           ; 32767 * SIGN(SAMPLE)
        SACL    TEMP1
*
SIZOK   LAC     TEMP1,14
        SACH    SAMPLE          ; DIVIDE BY 4, MAX VALUE IS 8191 OR -8192
*
*****************************************************************
*
*   CHECK WHETHER DTMF IS SWITCHED ON. DTMF IS SWITCHED ON WHEN THE
*   APPROPRIATE BIT IN THE FLAGS REGISTER IS SET. THIS BIT IS COPIED FROM THE
*   MODE REGISTER EVERY TIME THE RESET-FILTERING ROUTINE IS CALLED, WHICH MAY
*   BE DETERMINED FROM THE FLOWCHART. THIS ENSURES THAT THE ON/OFF STATUS OF
*   DTMF CANNOT CHANGE IN THE MIDDLE OF A FILTERING BLOCK. (DTMF AFFECTS THE
*   NUMBER OF FILTERS USED)
*
*****************************************************************
*
        LAC     ONE,DTMFLG
        AND     FLAGS
        BZ      FILCHK          ; IF DTMF IS OFF, BRANCH AROUNF THE CALL
*                               ; TO IT
*
        CALL    DTMF            ; CALL THE DTMF ROUTINE.
*
FILCHK  .set    $
*
        LAC     FILPOS
        SUB     ONE,14          ; CHECK FOR END OF FILTERING, FILTER
        BGZ     LEVCAL          ; POSITION GREATER THAN 16384.
*
*****************************************************************
*
*   IF FILTER POSITION HAS INCREMENTED PAST 16384, THEN A BLOCK OF FILTERING
*   HAS BEEN COMPLETED AND PROGRAM FLOW BRANCHES TO LEVEL CALCULATION, OTHER-
*   WISE IT CONTINUES WITH THE ENVELOPE DETECTOR.
*
*****************************************************************
```

```
*
*****************************************************************
*
*    ROUTINE: ENVDET
*
*    REFERENCE IN FLOWCHART: POWER DETECTOR
*
*    FUNCTION: DETECT CHANGES IN SIGNAL ENVELOPE RELATIVE TO THE USER-
*              PROGRAMMED UPPER AND LOWER THRESHOLDS. ENVELOPE DETECTOR ALWAYS
*              RUNS, REGARDLESS OF WHETHER TONE DETECTION IS ENABLED. THE
*              ENVELOPE DETECTOR IS USED FOR TIMESTAMPING.
*
*****************************************************************
*
ENVDET    .set      $
*
          LAC       EDFCT,8
          SACH      TEMP
          LAC       TEMP
          AND       MSOOFF          ; EXTRACT EDF FROM EDFCT
*
          SACL      TEMP
          LAC       TEMP,5
          SACL      TEMP            ; 32 * EDF IN TEMP
*
*    THIS OPERATION IMPLEMENTS A SMOOTHING FILTER OF THE FORM:
*
*         ENVEL = (2**15 * ENVEL) + ABS(32EDF * SAMPLE) - (32EDF * ENVEL)
*                 ------------------------------------------------------
*                                        2**15
*
*    WHICH IS THE SAME AS
*
*         ENVEL = ((1 - K) * ENVEL) + (K * ABS(SAMPLE))
*
*    WHERE EDF IS K * 2**10, K POSITIVE
*
*****************************************************************
*
          LT        TEMP
          MPY       SAMPLE
          PAC
          ABS
          MPY       ENVEL
          SPAC
          ADD       ENVEL,15
          ADD       ONE,14
          SACH      ENVEL,1
*
*****************************************************************
*
*    THE NEXT PIECE OF CODE CHECKS THE ENVELOPE LEVEL AGAINST THE UPPER OR
*    LOWER THRESHOLD, ACCORDING TO THE STATE OF THE SIGNAL PRESENT FLAG.
*
*****************************************************************
*
          LAC       ONE,TPRFLG      ; LOAD A 1 IN TONE PRESENT FLAG POSITION
          AND       FLAGS           ; AND WITH THE FLAG REGISTER
          BGZ       TPRSNT          ; BRANCH TO TONE PRESENT IF IT IS SET
*
NOTONE    LAC       UPRTHR,2        ; WE WANT TO COMPARE WITH
*                                   ; (8 * 2/c * UPRTHR)
          ADD       UPRTHR          ; WHICH IS VERY NEARLY FIVE * UPRTHR
*
          SUB       ENVEL           ; BRANCH TO NOSIG IF ENVEL IS THE UPPER
          BGEZ      NOSIG           ; THRESHOLD.
*
          LAC       LWRTHR,2        ; WE ALSO COMPARE WITH (8 * 2/C * LWRTHR)
*                                   ; BECAUSE THE HIGHER OF THE TWO THRESHOLDS
*                                   ; IS TAKEN AS THE UPPER ONE.
          ADD       LWRTHR
*
          SUB       ENVEL           ; DROP THROUGH TO NOSIG IF ENVEL IS THE
          BLZ       TONSET          ; LOWER THRESHOLD.
*
NOSIG     .set      $
*
          CALL      RSTFIL          ; RESET FILTERING TO INCORPORATE ANY
*                                   ; CHANGED PARAMETERS
*
          B         MAIN            ; RETURN TO START.
*
*****************************************************************
*
*    TONE PRESENT FLAG WAS SET.
*
*****************************************************************
*
TPRSNT    .set      $
*
          LAC       LWRTHR,2        ; WE WANT TO COMPARE WITH
*                                   ; (8 * 2/c * LWRTHR)
          ADD       LWRTHR          ; WHICH IS VERY NEARLY FIVE * LWRTHR
*
          SUB       ENVEL           ; BRANCH TO TDEPT IF ENVEL LOWER THAN THE
          BGZ       TDEPT           ; THRESHOLD.
*
*                                   ; OTHERWISE BRANCH TO TONCHK, AND IF THE
*                                   ; TONE DETECTOR IS ON, THEN BRANCH TO THE
*                                   ; FILTER ROUTINE.
*
          B         TONCHK
*
*****************************************************************
*
*    ROUTINE: TONSET
*
*    REFERENCE IN FLOWCHART: SET TONE PRESENT FLAG
```

```
*                       HOLD ONSET TIME
*
*   FUNCTION: HANDLE OCCURANCE OF TONE ONSET
*
*****************************************************************
*
TONSET  LAC     ONE,TPRFLG      ; LOAD A 1 IN TONE PRESENT FLAG POSITION
        OR      FLAGS
        SACL    FLAGS           ; THIS HAS SET THE TONE PRESENT FLAG
*
        LAC     CRTIME
        SACL    OSHOLD          ; SAVE ONSET TIME OF DETECTED SIGNAL IN
*                               ; ONSET TIME LATCH REGISTER
*
*****************************************************************
*
*   CHECK THAT THE TONE DETECTOR IS SWITCHED ON. IF IT IS NOT, THEN RESET THE
*   FILTER AND RETURN TO THE BEGINNING. IF IT IS, THEN BRANCH TO THE FILTER-
*   ING ROUTINE.
*
*****************************************************************
*
TONCHK  .set    $
*
        LAC     ONE,TONEBT
        AND     STMODE
        BGZ     FILTER          ; IF TONE DETECTOR IS ON, BRANCH TO FILTER
*                               ; ROUTINE
*
*****************************************************************
*
*   IF AT THE END OF THE ROUTINE A BLOCK OF FILTERING HAS BEEN COMPLETED,
*   PROGRAM FLOW BRANCHES TO LEVELS, OTHERWISE IT BRANCHES TO MAIN
*
*****************************************************************
*
TONOFF  B       CONT3           ; RESET THE FILTERS READY FOR WHEN IT IS
*                               ; SWITCHED ON AGAIN.
*
*****************************************************************
*
*   ROUTINE: TDEPT
*
*   REFERENCE IN FLOWCHART: CLEAR TONE PRESENT FLAG
*                           RESET FILTERING
*                           SET ONSET TIME VALID FLAG
*                           CLEAR ONSET FLAG
*                           SAVE DEPART TIME
*                           SET DEPART INTERRUPT
*                           SET SHORT TONE INTERRUPT
*
*   FUNCTION: HANDLE TONE DEPARTURE
*
*****************************************************************
*
TDEPT   LAC     ONE,TPRFLG      ; PUT A 1 IN THE TONE PRESENT FLAG POSITION
        XOR     FLAGS
        SACL    FLAGS           ; THIS HAS CLEARED THE TONE PRESENT FLAG.
*
        CALL    RSTFIL          ; CALL RESET FILTERING ROUTINE TO CLEAR
*                               ; DOWN ALL ACCUMULATORS AND SET UP FILTER
*                               ; READY FOR THE NEXT BLOCK.
*
        LAC     ONE,ONSFLG      ; LOAD A 1 IN THE ONSET TIME VALID FLAG
*                               ; POSITION
        AND     FLAGS
        BGZ     OSVAL           ; IF THE FLAG IS SET, BRANCH TO OSVAL
*
        LAC     ONE,TONEBT      ; ELSE IF THE TONE DETECTOR IS ON, SET A
        AND     STMODE          ; SHORT TONE INTERRUPT.
        BZ      MAIN            ; IF TONE DETECTOR IS OFF, GO HOME.
*
STINT   LAC     ONE,STINBT
        XOR     STMODE
        AND     STMODE          ; ASSERT SHORT TONE INTERRUPT
        SACL    STMODE
*
        CALL    ATTEN           ; WRITE OUT STATUS
*
        CALL    XFUPD           ; UPDATE XF FLAG
*
        B       MAIN
*
OSVAL   XOR     FLAGS           ; CLEAR ONSET TIME VALID FLAG
        SACL    FLAGS
*
        LAC     CRTIME
        SACL    DPTIME          ; TONE DEPARTURE TIME = CURRENT TIME.
*
        LAC     ONE,TONEBT      ; IF TONE DETECTOR IS ON, THEN SET A DEPART
        AND     STMODE          ; INTERRUPT.
        BZ      MAIN            ; ELSE GO HOME.
*
*                               ; SET A TONE DEPARTURE INTERRUPT
DPINT   LAC     ONE,DPINBT
        XOR     STMODE
        AND     STMODE          ; ASSERT DEPART INTERRUPT
        SACL    STMODE
*
        CALL    ATTEN           ; WRITE OUT STATUS
*
        CALL    XFUPD           ; UPDATE XF FLAG
*
        B       MAIN
*
*****************************************************************
*
*   ROUTINE: FILTER
```

```
*
*   REFERENCE IN FLOWCHART: FILTER
*
*   FUNCTION: ROUTINE FOR FILTERING AND ACCUMULATING THE INPUT SAMPLE
*
*****************************************************************
*
FILTER   .set     $
*
*****************************************************************
*
*   THE FIRST SECTION CALCULATES A SAMPLE OF SIN(X)/X. THE PHASE X IS STORED
*   IN 2**-5 FORM TO AVOID POSSIBLE OVERFLOW, AND THE RESULTING SAMPLE IS IN
*   2**14 FORM, MAXIMUM VALUE C/2 * 2**14. THE DIVISION ASSUMES NUMERATOR IN
*   2**14 FORM, HOWEVER THE SINE SECTION PRODUCES A 2**13 RESULT, AND THE
*   DENOMINATOR IS ACTUALLY IN 2**-5 FORM, THUS THE NUMERATOR IS SHIFTED LEFT
*   BY 10 PLACES INSTEAD OF 14.  (14-5+1 = 10)
*
*****************************************************************
*
        LACK    SXCENT          ; LOAD UP CENTRAL VALUE OF SIN(X)/X
        TBLR    TEMP1           ; FUNCTION INTO TEMP1
        LAC     SINCPH,5        ; SIN(X)/X WHERE THE PHASE X = ZERO IS A
        BZ      PHASE0          ; SPECIAL CASE.
*
*****************************************************************
*
*   ACCUMULATOR NOW CONTAINS FULL 2**0 REPRESENTATION OF SINCPHASE. THIS IS
*   TRUNCATED TO THE NORMAL q c REPRESENTATION.
*
*****************************************************************
*
        ADD     ONE,14          ; GENERATE SINE OF SINCPH
        SACL    TEMP
        LAC     TEMP
        ABS
        SUB     ONE,14
        SACL    TEMP
*
        LT      TEMP
        LAC     K1,15
        MPY     TEMP
        SPAC
        SACH    TEMP,1
*
        MPY     TEMP
        PAC
        SACH    TEMP1,1         ; SINE(X) IN TEMP1 IN 2**13 FORM
*
        LT      TEMP1           ; LOAD T WITH SIN(X)
        MPY     SINCPH          ; DUMMY MULTIPLY IN ORDER TO OBTAIN
        PAC                     ; SIGN OF SIN(X)/X QUOTIENT IN TEMP3
        SACH    TEMP3
*
        LAC     SINCPH
        ABS                     ; OBTAIN POSITIVE DIVISOR (PHASE X) IN TEMP
        SACL    TEMP            ; DIVISOR ANGLE STILL IN 2**-5 FORM
*
        LAC     TEMP1,10        ; SINE(X) IN 2**13 FORM * 2**10 IN
*                               ; ACCUMULATOR
        ABS                     ; READY FOR DIVISION. INITIAL RESULT IN
*                               ; 2**8.
*
DIV0    SUBC    TEMP            ; 8 CYCLE DIVIDE LOOP
        NOP
        SUBC    TEMP
        NOP
        SUBC    TEMP
        NOP
        SUBC    TEMP
        NOP
        SUBC    TEMP
        NOP
        SUBC    TEMP
        NOP
        SUBC    TEMP
        NOP
        SUBC    TEMP
        NOP
*
        AND     MSOOFF
        SACL    TEMP1           ; 8 BIT RESULT IN 2**8 FORM
        LAC     TEMP1,8
        ADD     ONE,7           ; ROUND BY ADDING 128
        SACL    TEMP1           ; FINAL RESULT IN 2**14 FORM
*
        LAC     TEMP3
        BGEZ    PHASE0          ; DONE IF REQUIRED SIGN IS POSITIVE
*
        ZAC
        SUB     TEMP1
        SACL    TEMP1           ; OTHERWISE INVERT SIGN OF RESULT
*
PHASE0  .set    $               ; SIN(X)/X SAMPLE IN TEMP1
*
*****************************************************************
*
*   NOW CALCULATE WINDOW POSITION AND READ WINDOW SAMPLE FROM ROM TABLE
*
*****************************************************************
*
        LAC     FILPOS,9
        ABS
        SACH    TEMP            ; DIVIDE ABS (FILTERPOSITION) BY 128
*                               ; THE RESULT IS THE OFFSET INTO THE HALF-
*                               ; LENGTH WINDOW TABLE.
        LACK    WINTAB
        ADD     TEMP            ; ADD IN WINDOW TABLE OFFSET
```

```
*
        LAC     ONE,DTMFLG
        AND     FLAGS
        BNZ     FLOOP           ; IF DTMF IS ON, ONLY DO THREE FILTERS,
*
        LARK    AR0,5           ; ELSE DO ALL SIX
*
FLOOP   .set    $               ; FILTERING LOOP
*
        LARP    1
*
*****************************************************************
*
*   THIS IS THE FILTERING LOOP. FOR EACH OF THE FILTERS, A SINE AND A COSINE
*   SAMPLE IS GENERATED, AND EACH IS MULTIPLIED BY THE PRODUCT OF THE INPUT
*   SAMPLE AND THE APPROPRIATE FILTER COEFFICIENT, WIDE OR NARROW. THE NARROW
*   FILTER COEFFICIENT IS JUST THE WINDOW SAMPLE WHEREAS THE WIDE FILTER
*   COEFFICIENT IS THE WINDOW SAMPLE MULTIPLIED BY A SIN(X)/X SAMPLE.
*
*****************************************************************
*
*****************************************************************
*
*   GENERATE SINE AND COSINE SAMPLE AT THE SPECIFIED SEARCH FREQUENCY OF THIS
*   FILTER
*
*****************************************************************
*
        LAC     *+              ; LOAD UP FREQN WHICH IS THE REQUIRED PHASE
*                               ; INCREMENT
        ADD     *               ; ADD CURRENT PHASE.
        SACL    *               ; CALCULATE NEW PHASE OF SEARCH FREQUENCY
*
*****************************************************************
*
*   SINE AND COSINE ROUTINE. REQUIRES ARGUMENT IN TEMP1 IN REPRESENTATION
*   WHERE q 2**15 REPRESENTS q c. RESULT IS -COS(TEMP1) SCALED IN LOCATION
*   TEMP2 AND SINE(TEMP1) SCALED IN LOCATION TEMP1 USES TEMP AS A SCRATCH
*   LOCATION.
*
*****************************************************************
*
*   K1 IS A CONSTANT EQUAL TO 1.4008687 * 2**14. THE OCCURRANCES OF ONE,14
*   REPRESENT C/2. THE RESULT IS A SINE AND A -COSINE SCALED BY 0.9050 *
*   2**13. THE ALGORITHM ACCEPTS ANGLES IN THE RANGE q 2 * 2**14, REPRESENT-
*   ING q c, AND CONVERTS THEM INTO 2 EQUIVALENT ANGLES IN THE RANGE q 1 *
*   2**14, ONE SHIFTED BY C/2.  IT THEN PERFORMS A SINE ALGORITHM ON EACH OF
*   THESE TWO VALUES TO YIELD THE DESIRED RESULTS.
*
*****************************************************************
*
SINCOS  .set    $
*
        LAC     *               ; MAKE ANGLE MODULO 65536
*
        TBLR    WINDOW          ; READ WINDOW SAMPLE
*
        LT      WINDOW
        MPY     TEMP1           ; TAKE WINDOW * SIN(X)/X PRODUCT
        PAC
*
        SACH    SNCWIN,1        ; STORE PRODUCT
*
*****************************************************************
*
*   ACCUMULATE WINDOW AND SNCWIN INTO THEIR RESPECTIVE ACCUMULATORS; THESE
*   WILL LATER BE USED TO NORMALIZE THE FILTER OUTPUTS TO COMPENSATE FOR THE
*   EFFECT OF THE WINDOWING AND THE MULTIPLICATION BY A SIN(X)/X FUNCTION.
*
*****************************************************************
*
        ZALH    ACSWHI
        ADDS    ACSWLO          ; LOAD UP SNCWIN ACCUMULATOR
*
        ADD     SNCWIN
        SACL    ACSWLO
        SACH    ACSWHI          ; STORE ACCUMULATION BACK.
*
        ZALH    ACWNHI
        ADDS    ACWNLO          ; LOAD UP WINDOW ACCUMULATOR
*
        ADD     WINDOW
        SACL    ACWNLO
        SACH    ACWNHI          ; STORE ACCUMULATION BACK.
*
        LAC     SAMPLE,2
        SACL    TEMP            ; MULTIPLY SCALED LINEARIZED SAMPLE BY FOUR
*
        LT      TEMP
        MPY     SNCWIN          ; GENERATE WIDE FILTER SAMPLE
        PAC
*
        SACH    SNCWIN,1        ; SNCWIN NOW CONTAINS WIDE FILTER SAMPLE
*
        MPY     WINDOW          ; GENERATE NARROW FILTER SAMPLE
        PAC
*
        SACH    WINDOW,1        ; WINDOW NOW CONTAINS NARROW FILTER SAMPLE
*
        LAC     MSOOFF,8
        AND     FSPW            ; MASK IN FILTERSELECT BYTE
*
        SACL    TEMP3
        LAC     TEMP3,2         ; THE SIX FILTERSELECT BITS ARE NOW IN THE
        SACL    TEMP3           ; TOP SIX BITS OF TEMP3
*
        LARK    AR1,FREQ1
        LARK    AR0,2           ; SET UP LOOP COUNTER FOR THE FIRST THREE
```

```
        ABS
        SUB     ONE,14
        SACL    TEMP
        LT      TEMP
        LAC     K1,15
        MPY     TEMP
        SPAC
        SACH    TEMP,1
        MPY     TEMP
        PAC
        SACH    TEMP2,1
        LAC     *+
        ADD     ONE,14
        SACL    TEMP
        LAC     TEMP
        ABS
        SUB     ONE,14
        SACL    TEMP
        LT      TEMP
        LAC     K1,15
        MPY     TEMP
        SPAC
        SACH    TEMP,1
        MPY     TEMP
        PAC
        SACH    TEMP1,1
*
*****************************************************************
*
*   END OF SINE COSINE ROUTINE
*
*****************************************************************
*
*****************************************************************
*
*   THIS SECTION GETS EITHER WIDE OR NARROW SAMPLE INTO THE T REGISTER.
*
*****************************************************************
*
SELECT  .set    $
*
        LT      SNCWIN          ; SNCWIN CONTAINS THE WIDE SAMPLE
*
        LAC     TEMP3,1         ; PICK OFF THE FILTERSELECT BIT FOR THIS
*                               ; FILTER
        SACL    TEMP3
        BLZ     WIDEF           ; IF THE FILTERSELECT BIT WAS SET, THEN
*                               ; THIS FILTER IS WIDE.
*
NARROF  .set    $               ; THIS FILTER IS NARROW
*
        LT      WINDOW          ; WINDOW CONTAINS THE NARROW SAMPLE
*
*****************************************************************
*
*   END OF SELECTION SECTION
*
*****************************************************************
*
WIDEF   .set    $               ; THIS FILTER IS WIDE
*
        MPY     TEMP2
        PAC
        ADD     ONE,14
        SACH    TEMP,1          ; SAMPLE*COSINE / 2**15
*
        LAC     TEMP,2          ; LOAD SAMPLE*COSINE PRODUCT, NOW
*                               ; ACCUMULATE
*
        ADDH    *+              ; COSNHI ADDED
        ADDS    *-              ; COSNLO ADDED
        SACH    *+              ; STORE BACK ACCUMULATOR.
        SACL    *+
        MPY     TEMP1
        PAC
        ADD     ONE,14
        SACH    TEMP,1          ; SAMPLE*SINE / 2**15
        LAC     TEMP,2          ; LOAD SAMPLE*SINE PRODUCT, NOW ACCUMULATE
*
        ADDH    *+              ; SINNHI ADDED
        ADDS    *-              ; SINNLO ADDED
        SACH    *+              ; STORE BACK ACCUMULATOR.
        SACL    *+,0,AR0        ; AR0 NOW POINTING AT FREQN FOR NEXT FILTER
*
        BANZ    FLOOP
*
*****************************************************************
*
*   THE NEXT SECTION ACCUMULATES (SAMPLE/4)} SO THAT A MEASURE OF THE TOTAL
*   SIGNAL ENERGY IN THE BAND CAN BE CALCULATED.
*
*****************************************************************
*
        LAC     SAMPLE,14       ; SAMPLE/4 IN HIGH ACCUMULATOR
        SACH    TEMP
        LT      TEMP
        MPY     TEMP
        PAC                     ; (SAMPLE/4)} IN ACCUMULATOR
        ADDH    ACS@HI          ; ACCUMULATOR
        ADDS    ACS@LO
        SACH    ACS@HI          ; STORE ACCUMULATOR BACK.
        SACL    ACS@LO
*
*****************************************************************
*
*   INCREMENT FILTER POSITION
*
*****************************************************************
```

```
*
        LAC     FLSTOR,1        ; CALCULATE FILTERPOSITION INCREMENT AS
        ADD     ONE,5           ; (2FL +32)
*
        ADD     FILPOS
        SACL    FILPOS
*
*****************************************************************
*
*   INCREMENT SIN(X)/X PHASE
*
*****************************************************************
*
        LAC     MSOOFF
        AND     FSPW            ; MASK IN PW THE PASSBANDWIDTH, WHICH
*                               ; EQUALS PHASE INCREMENT REQUIRED FOR THE
*                               ; Sin(X)/X PHASE.
        ADD     SINCPH
        SACL    SINCPH
*
        B       MAIN            ; RETURN TO BEGINNING OF MAIN LOOP.
*
*****************************************************************
*
*   ROUTINE: LEVCAL
*
*   REFERENCE IN FLOWCHART: CALCULATE LEVELS
*
*   FUNCTION: CAICULATES THE LEVELS AT THE END OF EACH BLOCK OF FILTERING
*
*****************************************************************
*
LEVCAL  .set    $
*
*****************************************************************
*
*   FIRST CHECK THAT THE QUEUE IS EMPTY. IF IT IS NOT, JUMP BACK TO THE
*   BEGINNING OF THE PROGRAM. DON'T DO ANY OF THIS PROCESSING UNTIL THE QUEUE
*   IS EMPTY; THIS WILL RESTORE THE INTERRUPT TIME OVERHEAD PROVIDED BY
*   HAVING AN EMPTY QUEUE.
*
*****************************************************************
*
        LAC     QIN             ; THE QUEUE IS EMPTY WHEN THE QUEUE INPUT
        SUB     QOUT            ; POINTER EQUALS THE OUTPUT POINTER.
        BNZ     MAIN
*
*****************************************************************
*
*   CALCULATE THE TOTAL SIGNAL LEVEL IN THE WHOLE BAND. WE HAVE AN ACCUMU-
*   LATION OF (SAMPLE/4)) IN ACSQRD. THIS IS DIVIDED BY (4*FILTERLENGTH),
*   WHERE FILTER LENGTH IS (16384/(FL + 16)) + 1, AND THEN MULTIPLIED BY 2
*   AND SQUARE-ROOTED.
*
*****************************************************************
*
        LAC     FLSTOR
        ADD     ONE,4           ; GIVES FL + 16
        SACL    TEMP            ; AS DIVISOR IN TEMP.
*
        LARK    AR0,15          ; SET UP4AR0 AS 16 CYCLE COUNTER
        ZALH    ONE
*
DLOOP   SUBC    TEMP
        BANZ    DLOOP
*
        ADD     ONE,2
        SACL    TEMP            ; 4 * FILTERLENGTH IN TEMP
*
        ZALH    ACSQHI
        ADDS    ACSQLO
*
        SUB     TEMP,15         ; LIMIT DIVIDEND TO LESS THAN (2**15 *
*                               ; DIVISOR) SO THAT RESULT OF DIVISION WILL
*                               ; WILL BE LESS THAN 2**15
        BLZ     SIZEOK
*
TOOBIG  ZAC
        SUB     ONE
*
SIZEOK  .set    $
*
        ADD     TEMP,15         ; DIVIDEND LIMITED.
*
        LARK    AR0,15          ; SET UP AR0 FOR 16 SUBC'S
*
ALOOP   SUBC    TEMP
        BANZ    ALOOP
*
        SACL    TEMP
        LAC     TEMP,3          ; RESULT NOW IN 2**2 FORM AS REQUIRED BY
*                               ; THE SQUARE ROOT ROUTINE.
        SACH    TEMP1
        SACL    TEMP2
        CALL    SQRT            ; SQUARE ROOT. RESULT IN TEMP3
*
        LAC     MSOOFF,8
        AND     CTLTSL          ; MASK IN CONTROL REGISTER BITS ONLY.
        ADD     TEMP3
        SACL    CTLTSL          ; STORE RESULT IN LOWER HALF OF CTLTSL
*
*****************************************************************
*
*   DIVIDE THE (WINDOW*SIN(X)/X) WIDE ACCUMULATION ANF THE WINDOW NARROW
*   ACCUMULATION BY 2**15.
*
*****************************************************************
*
```

```
        ZALH    ACSWHI
        ADDS    ACSWLO
        SACH    ACSWLO,1        ; RESULT IN ACSWLO
*
        ZALH    ACWNHI
        ADDS    ACWNLO
        SACH    ACWNLO,1        ; RESULT IN ACWNLO
*
*****************************************************************
*
*   NOW NORMALIZE THE SINE AND COSINE FILTER ACCUMULATORS BY DIVIDING THEM BY
*   EITHER THE (WINDOW * SIN(X)/X) ACCUMULATION OR THE WINDOW ACCUMULATION
*   DEPENDING ON WHETHER THE FILTER IS WIDE OR NARROW.
*
*****************************************************************
*
        LAC     MS00FF,8
        AND     FSPW            ; MASK IN FILTER SELECT BYTE
*
        SACL    FILPOS
        LAC     FILPOS,2        ; THE SIX FILTER SELECT BITS ARE NOT IN THE
        SACL    FILPOS          ; TOP SIX BITS OF FILPOS
*
        LARK    AR1,FREQ1
        LARK    AR0,2           ; SET UP LOOP COUNTER FOR THE FIRST THREE
*                               ; FILTERS ONLY
        LAC     ONE,DTMFLG
        AND     FLAGS
*
        BNZ     NLOOP           ; IF DTMF IS ON, ONLY DO THREE FILTERS,
*
        LARK    AR0,5           ; ELSE DO ALL SIX
*
NLOOP   .set    $
*
        LARP    1
*
*****************************************************************
*
*   THIS IS THE NORMALIZATION LOOP.
*
*   THIS SECTION GETS EITHER THE WIDE OR NARROW ACCUMULATION INTO TEMP
*
*****************************************************************
*
        LAC     ACSWLO          ; ACSWLO CONTAINS THE WIDE NORMALIZATION
        SACL    TEMP
*
        LAC     FILPOS,1        ; PICK OFF THE FILTERSELECT BIT FOR THIS
*                               ; FILTER
        SACL    FILPOS
        BLZ     WIDEFL          ; IF THE FILTERSELECT BIT WAS SET, THEN
*                               ; THIS FILTER IS WIDE.
*
NARROW  .set    $               ; THIS FILTER IS NARROW
*
        LAC     ACWNLO          ; ACWNLO CONTAINS THE NARROW NORMALIZATION
        SACL    TEMP
*
WIDEFL  .set    $               ; FIRST NORMALIZE THE COSINE ACCUMULATOR
        LAC     *+
        LAC     *+              ; TWO DUMMY READS TO INCREMENT POINTER TO
*                               ; COSNHI
*
        ZALH    *+
        ADDS    *               ; LOAD UP COSINE FILTERN ACCUMULATOR
        ABS
*
        SAR     AR1,TEMP1       ; SAVE AR1 IN TEMP1
        LARK    AR1,15          ; USE AR1 TO CONTROL A 16 CYCLE DIVIDE.
*
CDIV    SUBC    TEMP            ; DIVIDE BY THE SELECTED ACCUMULATOR
        BANZ    CDIV
*
        LAR     AR1,TEMP1       ; RESTORE AR1
*
        SACL    TEMP2
        LT      TEMP2           ; MULTIPLY RESULT OF DIVISION BY SINE
        MPY     CORREC          ; CORRECTION FACTOR.
        PAC
        SACH    *+,1,AR1        ; STORE 2*RESULT IN COSNLO
*
*****************************************************************
*
*   NOW REPEAT FOR THE SINE ACCUMULATOR
*
*****************************************************************
*
        ZALH    *+
        ADDS    *-              ; LOAD UP SINE FILTERN ACCUMULATOR
        ABS
*
        SAR     AR1,TEMP1       ; SAVE AR1 IN TEMP1
        LARK    AR1,15          ; USE AR1 TO CONTROL A 16 CYCLE DIVIDE.
*
SDIV    SUBC    TEMP
        BANZ    SDIV
*
        LAR     AR1,TEMP1       ; RESTORE AR1
*
        SACL    TEMP2
        LT      TEMP2           ; MULTIPLY RESULT OF DIVISION BY SINE
        MPY     CORREC          ; CORRECTION FACTOR
        PAC
        SACH    *,1,AR1         ; STORE 2*RESULT IN SINnHI
*
        LTC     *
        MPY     *-
```

```
        PAC                     ; SQUARE SINE RESULT
*
        LT      *
        MPY     **              ; SQUARE COS RESULT
        APAC                    ; AND ADD IT IN   -> 4* SUM OF SQUARES
*
        SACH    TEMP1
        SACL    TEMP2
        CALL    SQRT            ; NOTE THAT SQRT ALWAYS RETURNS WITH ARP=0
*
        LARP    1
*
        LAC     **              ; DUMMY READ TO INCREMENT POINTER TO END OF
*                               ; BLOCK.
        LAC     TEMP3           ; LOAD UP RESULT OF SQUARE ROOT
        SACL    **,0,AR0        ; STORE RESULT (FILTERN OUTPUT LEVEL) IN
*                               ; SINNLO
*
*****************************************************************
*
*   END OF NORMALIZATION SECTION. REPEAT FOR EACH FILTER.
*
*****************************************************************
*
        BANZ    NLOOP
*
*****************************************************************
*
*   ROUTINE: CHNGS
*
*   REFERENCE IN FLOWCHART: CHECK CHANGES
*
*   FUNCTION: CHECK FOR LEVEL CHANGES DURING A TONEBURST
*
*****************************************************************
*
*****************************************************************
*
*   NOW CHECK FOR CHANGES IN ANY OF THE FILTER LEVELS WHICH CROSS THE CHANGE
*   THRESHOLD. COPY THE FLAGS REGISTER INTO TEMP, ANF USE THAT TO CHECK FOR
*   CHANGES WHILE THE REAL FLAGS REGISTER IS MODIFIED TO REFLECT THE FILTER
*   OUTPUTS WHICH ARE CURRENTLY ABOVE THE CHANGE THRESHOLD.
*
*****************************************************************
*
CHNGS   .set    $
*
        LAC     FLAGS
        SACL    TEMP            ; COPY FLAGS INTO TEMP
*
        LAC     MSOOFF,8
        AND     FLAGS
        SACL    FLAGS           ; CLEAR OUT THE SIX FILTER LEVEL BITS.
*
        LAC     EDFCT
        AND     MSOOFF          ; LOAD UP CHANGE THRESHOLD
        SACL    TEMP1           ; STORE CT IN TEMP1
*
        LARK    AR0,FREQ1       ; START POINTING AT FREQ1
*
        LAC     ONE,FILT1
        SACL    TEMP2           ; INITIAL BIT TO WORK WITH IN FLAGS
*
CLOOP   MAR     **
        MAR     **
        MAR     **
        MAR     **
        MAR     **              ; NOW POINTING AT SINNLO
*
        LAC     TEMP1           ; LOAD UP CHANGE THRESHOLD
        SUB     **              ; AR1 NOW POINTING AT FREQN+1
*
        BGEZ    CEND            ; DROP TO END OF LOOP IF LEVEL IN THIS BAND
*                               ; IS BELOW THE CHANGE THRESHOLD.
        LAC     TEMP2           ; LOAD UP APPROPRIATE FLAG BIT MASK
        XOR     TEMP
        SACL    TEMP            ; FLIP THE BIT IN TEMP
*
        LAC     TEMP2
        OR      FLAGS
        SACL    FLAGS           ; SET THE BIT IN FLAGS
*
CEND    LAC     TEMP2,15
        SACH    TEMP2           ; SHIFT BIT OF INTEREST ALONG TO THE RIGHT
*
        LAC     TEMP2
        BNZ     CLOOP           ; REPEAT CLOOP FOR EACH FILTER
*
*****************************************************************
*
*   ROUTINE: LVLS
*
*   REFERENCE IN FLOWCHART: WRITE OUT LEVELS
*
*   FUNCTION: WRITE LEVELS INTO REGISTERS
*
*****************************************************************
*
*****************************************************************
*
*   COPY FILTER OUTPUT LEVELS INTO THEIR RESPECTIVE REGISTERS IN COMPRESSED
*   FORMAT, 2 LEVELS TO A WORD.
*
*****************************************************************
*
LVLS    .set    $
*
        LAC     SINILO,8
```

```
        ADD     SIN2LO
        SACL    LVL12
*
        LAC     SIN3LO,8
        LDPK    1
        ADD     SIN4LO
        LDPK    0
        SACL    LVL34
*
        LDPK    1
        LAC     SIN5LO,8
        ADD     SIN6LO
        LDPK    0
        SACL    LVL56
*
*****************************************************************
*
*   ROUTINE: COMPLT
*
*   REFERENCE IN FLOWCHART: FIRST BLOCK FLAG SET?
*                           CHANGES?
*                           SET CHANGE INTERRUPT
*                           CLEAR FIRST BLOCK FLAG
*                           SAVE HELD ONSET TIME
*                           SET ONSET TIME VALID FLAG
*                           SET ONSET INTERRUPT
*
*   FUNCTION: COMPLETE OPERATIONS READY FOR NEXT FILTERING OPERATION
*
*****************************************************************
*
COMPLT  .set    $
*
        LAC     ONE,FSTFLG
        AND     FLAGS
        BNZ     FSTTIM          ; IF IT IS THE FIRST BLOCK, SKIP OVER THE
*                               ; NEXT SECTION
        LAC     TEMP
        AND     MSOOFF
        BZ      CONT3           ; IF ALL THE 6 FLAG BITS ARE ZERO, THERE
*                               ; WAS NO CHANGE IN LEVEL ACROSS THE CHANGE
*                               ; THRESHOLD
*
CHINT   LAC     ONE,CHINBT      ; ELSE THERE WAS A CHANGE
        XOR     STMODE
        AND     STMODE          ; ASSERT CHANGE INTERRUPT
        SACL    STMODE
*
        CALL    ATTEN           ; WRITEOUT STATUSs
*
        CALL    XFUPD           ; UPDATE XF FLAG
*
        B       CONT3
*
```

```
FSTTIM  .set    $
*
        LAC     OSHOLD
        SACL    OSTIME          ; COPY ONSET TIME FROM HOLDING REGISTER
*                               ; INTO ONSET TIME REGISTER
        LAC     ONE,ONSFLG
        OR      FLAGS
        SACL    FLAGS           ; SET THE ONSET TIME VALID FLAG
*
*
*****************************************************************
*
*   SET AN ONSET INTERRUPT
*
*****************************************************************
*
OSINT   LAC     ONE,OSINBT
        XOR     STMODE
        AND     STMODE          ; ASSERT ONSET INTERRUPT
        SACL    STMODE
*
        CALL    ATTEN           ; WRITE OUT STATUS
        CALL    XFUPD           ; UPDATE XF FLAG
*
CONT3   .set    $
*
        CALL    RSTFIL          ; RESET THE FILTER
*
        LAC     ONE,FSTFLG
        XOR     FLAGS
        AND     FLAGS
        SACL    FLAGS           ; CLEAR THE FIRST BLOCK FLAG
*
        B       MAIN            ; RETURN TO BEGINNING
*
*****************************************************************
*
*   ROUTINE: RSTFIL
*
*   REFERENCE IN FLOWCHART: RESET FILTERING
*
*   FUNCTION: CLEAR DOWN FILTER ACCUMULATORS AND RESET POINTERS
*             READY FOR ANOTHER FILTERING OPERATION.
*
*****************************************************************
*
RSTFIL  .set    $               ; RESET FILTERING ROUTINE.
*
        LAC     MSOOFF          ; LOAD UP LOWER BYTE MASK
        AND     FMSFL           ; MASK IN FL
        SACL    FLSTOR          ; FL IN FLSTOR
*
        ZAC
        SUB     ONE,14
```

```
        SACL    FILPOS          ; RESET FILTER POSITION TO -16384
*
*****************************************************************
*
*   CALCULATE INITIAL SIN(X)/X PHASE
*
*   EXPRESSION FOR THIS IS:
*   (INITIAL FILTERPOSITION + SIN(X)/X INCREMENT)/HALF-FILTERLENGTH
*   WHICH IS THE SAME AS:
*   (-16384 * PW) / (2FL +32)
*
*****************************************************************
*
        LAC     FSPW            ; LOAD UP FILTER SELECT AND PASSBANDWIDTH
        AND     MSOOFF          ; MASK IN PASSBANDWIDTH SPECIFIER PW
        SACL    TEMP
*
        LAC     ONE,14          ; 16384
        SACL    TEMP1
*
        LAC     FLSTOR,1
        ADD     ONE,5           ; HALF FILTERLENGTH = (2FL +32)
        SACL    TEMP2           ; DIVISOR
*
        LT      TEMP1
        MPY     TEMP            ; 16384 * PW
        PAC                     ; DIVIDEND IN ACCUMULATOR
*
        LARK    0,15
*
SLOOP   SUBC    TEMP2
        BANZ    SLOOP
*
        SACL    SINCPH
        SUB     SINCPH,1
        SACL    SINCPH          ; INITIAL SINCPHASE
*
*****************************************************************
*
*   NOW ZERO ALL THE ACCUMULATORS
*
*****************************************************************
*
        LARK    AR1,FREQ1
        LARK    AR0,2           ; SET UP LOOP COUNTER FOR THE FIRST THREE
*                               ; FILTERS ONLY.
        LAC     ONE,DTMFLG
        AND     FLAGS
        BNZ     ZLOOP           ; IF DTMF IS ON, ONLY DO THREE FILTERS,
*
        LARK    AR0,5           ; ELSE DO ALL SIX
*
ZLOOP   LARP    AR1             ; FOR EACH FILTER, DO DUMMY INCREMENTS OF
        LAC     *+              ; AR1 OVER THE FREQN AND PHASEN LOCATIONS,
        LAC     *+
        ZAC                     ; THEN ZERO THE ACCUMULATOR AND PUT ZERO
        SACL    *+              ; INTO THE NEXT FOUR LOCATIONS.
        SACL    *+
        SACL    *+
        SACL    *+,0,AR0
*
        BANZ    ZLOOP           ; THEN ZERO THE REMAINING ACCUMULATORS
*
        SACL    ACSQHI          ; HIGH WORD OF 32BIT SIGNAL SQUARED
*                               ; ACCUMULATOR
        SACL    ACSQLO          ; LOW WORD OF 32BIT SIGNAL SQUARED
*                               ; ACCUMULATOR
        SACL    ACWNHI          ; HIGH WORD OF 32BIT WINDOW ACCUMULATOR
        SACL    ACWNLO          ; LOW WORD OF 32BIT WINDOW ACCUMULATOR
*
        SACL    ACSWHI          ; HIGH WORD OF 32BIT SIN(X)/X. WINDOW
*                               ; PRODUCT ACCUMULATOR
        SACL    ACSWLO          ; LOW WORD OF 32BIT SIN(X)/X. WINDOW
*                               ; PRODUCT ACCUMULATOR
*
*****************************************************************
*
*   CHECK INTO BIT IN MODE REGISTER AND SET THE DTMF ON/OFF FLAG IN THE FLAGS
*   REGISTER ACCORDINGLY
*
*****************************************************************
*
        LAC     ONE,DTMFLG
        AND     FLAGS
        SACL    TEMP            ; GET CURRENT DTMF FLAG INTO TEMP
*
        LAC     ONE,DTMFBT
        AND     STMODE
        SACL    TEMP1           ; GET STATE OF DTMF BIT INTO TEMP1
*
        LAC     TEMP1,(DTMFLG-DTMFBT)
        XOR     TEMP            ; RESULT IS A ONE IF FLAGS ARE DIFFERENT
*
        XOR     FLAGS
        SACL    FLAGS           ; DTMFLG NOW EQUALS DTMFBT
*
        LAC     ONE,FSTFLG      ; LOAD A 1 IN THE FIRST BLOCK FLAG POSITION
        OR      FLAGS
        SACL    FLAGS           ; THIS HAS SET THE FIRST BLOCK FLAG
*
        RET
*
*****************************************************************
*
*   ROUTINE: SQRT
*
*   REFERENCE IN FLOWCHART: NONE
*
```

```
*    FUNCTION: USED IN THE LEVEL CALCULATION ROUTINE. GENERATES THE SQUARE
*    ROOT OF AN INTEGER, WITH AN OUTPUT WHICH SATURATES AT 255.
*
*****************************************************************
*
SQRT     .set     $
*
*****************************************************************
*
*    THIS IS THE SQUARE ROOT ROUTINE. THE RESULT RANGE IS ZERO TO 255, AND IS
*    THE NEAREST INTEGER TO THE SQUARE ROOT OF THE INPUT NUMBER. ANY INPUT
*    NUMBER WHICH HAS A SQUARE ROOT > 254.5 WILL GIVE A RESULT IF 255. THE
*    INPUT NUMBER MUST BE STORED IN THE PAIR OF LOCATIONS TEMP1(HIGH) AND
*    TEMP2(LOW) IN 2**2 FORM, AND MUST BE POSITIVE. NEGATIVE NUMBERS WILL GIVE
*    THE RESULT ZERO. TEMP IS USED AS A TEMPORARY LOCATION, AND THE RESULT IS
*    RETURNED IN TEMP3. THE ROUTINE TAKES 111 CYCLES. SQRT ALWAYS RETURNS WITH
*    ARP = 0
*
*****************************************************************
*
         LAC      ONE,8          ; LOAD UP 128 * 2**1
*                                ; WORK IN 2**1 FORM THROUGHOUT.
         SACL     TEMP           ; INITIAL INCREMENT IS 128 * 2**1
         SUB      ONE
         SACL     TEMP3          ; INITIAL ROOT GUESS IS 127.5 * 2**1
*
         LARP     AR0
         SAR      AR0,SAMPLE     ; SAVE AR0 IN AN UNUSED LOCATION
*
         LARK     AR0,7          ; SET UP AR0 FOR 8 ITERATIONS
*
LOOP0    LAC      TEMP,15
         SACH     TEMP           ; HALVE THE INCREMENT
*
         LT       TEMP3          ; SQUARE THE ROOT
         MPY      TEMP3
         PAC
*
         SUBH     TEMP1
         SUBS     TEMP2
*
         BLEZ     RTOOSM         ; ROOT TOO SMALL
*
RTOOBG   .set     $              ; ROOT TOO BIG
*
         ZALS     TEMP3
         SUB      TEMP           ; SUBTRACT CURRENT INCREMENT FROM ROOT
         SACL     TEMP3
         BANZ     LOOP0
         B        END
*
RTOOSM   .set     $
*
         ZALS     TEMP3
         ADD      TEMP           ; ADD CURRENT INCREMENT TO ROOT
         SACL     TEMP3
         BANZ     LOOP0
*
END      .set     $
*
         LAC      TEMP3,15
         SACH     TEMP3          ; PUT ROOT INTO 2**0 FORM
*
         LAR      AR0,SAMPLE     ; RETRIEVE AR0 FROM ITS TEMPORARY STORE.
         RET
*
*****************************************************************
*
*    ROUTINE: DTMF
*
*    REFERENCE IN FLOWCHART: DTMF
*
*    FUNCTION: DETECT DTMF DIGITS
*
*****************************************************************
*
RSDTMF   .set     $
*
         LDPK     1
         ZAC
         SACLC    SIGCNT         ; ZERO VARIABLES
         SACL     CNTR
         SACL     CNTR1
         SACL     CNTR2
         SACL     STOP
         SACL     GAP
         SACL     PAUSE
         SACL     F1             ; ZERO FREQUENCY ARRAY
         SACL     F2
         SACL     F3
         SACL     F4
         SACL     F5
         SACL     F6
         SACL     F7
         SACL     F8
         SACL     ADJL
         SACL     ADJH
         SACL     SEMA           ; INITIALIZE SEMAPHORE
         SACL     TESTG
         SACL     TESTB
         LACK3
         SACLTMP
AGAIN    LDPK     0              ; RETURN TO PAGE 0
         RET                     ; END OF DTMF PROCESSING
*
DTMF     .set     $
*
         LDPK     1              ; DTMF PROCESSING ON PAGE 1
```

```
        LAC     UNITY           ; DECIMATE THE SAMPLES, DTMF USES
        XOR     DECIM           ; ALTERNATE ONES.
        AND     UNITY
        SACL    DECIM
        BZINP
*
        LDPK    0               ; RETURN TO PAGE 0
        RET
*
INP     .set    $
*
*****************************************************************
*
*   THIS SECTION ADDED TO IMPROVE THE DYNAMIC RANGE BY PROVIDING A DYNAMIC
*   THRESHOLD MINTHH WHICH COMES INTO PLAY DURING LOUD SIGNALS.
*
*****************************************************************
*
        LAC     MINTHH,2
        SUBADJL
        BGZ     MELSE
*
MTHEN   LAC     ADJL,14
        SACH    MINTHH
        ZAC
        SACL    MINTHL
        BMEND
*
MELSE   ZALH    MINTHH
        ADDS    MINTHL
        SUB     MINTHH,6        ; DECAY HALF LIFE OF 700 SAMPLES
        SACH    MINTHH
        SACL    MINTHL
*
MEND    .set    $
*
        LACK    MINTH
        SUB     MINTHH
        BLEZ    MNTHOK
*
        LACK    MINTH
        SACL    MINTHH
*
MNTHOK  .set    $
*
        LDPK0
        LAC     SAMPLE,2        ; SCALE INTO CORRECT RANGE
        LDPK1
        SACLX1
*
*****************************************************************
*
*   DTMF DECODER PROCESSING
*
*****************************************************************
*
        LT      X1              ; GET LINEAR INPUT SAMPLE
*
*****************************************************************
*
*   SCALE INPUT SAMPLE SO THAT THE 2ND ORDER SUB-FILTERS DO NOT OVERFLOW
*
*****************************************************************
*
        MPYK    SD              ; SCALE IT DOWN
        PAC
        SACH    X               ; AND STORE IT AS 8TH ORDER INPUT
*
*****************************************************************
*
*   8TH ORDER DETECTION WINDOW FOR DTMF LOW BAND
*
*****************************************************************
*
        LAC     X,15            ; N0
        LT      L1N2
        MPY     L1D             ; N2*D
        LTD     L1N1            ; N1-->N2
        MPY     L1C             ; N1*C
        APAC
        SACH    LY1,1           ; N0+N1*C+N2*D-->Y
        LTA     L2N2            ; N0+N1*C+N2*D+N1*C-->(ACC)
        SACH    L1N1,1          ; (ACC)-->N1
*
        LAC     X,15
        MPY     L2D
        LTD     L2N1
        MPY     L2C
        APAC
        SACH    LY2,1
        LTA     L3N2
        SACH    L2N1,1
*
        LAC     X,15
        MPY     L3D
        LTD     L3N1
        MPY     L3C
        APAC
        SACH    LY3,1
        LTA     L4N2
        SACH    L3N1,1
*
        LAC     X,15
        MPY     L4D
        LTD     L4N1
        MPY     L4C
        APAC
        SACH    LY4,1
```

```
        APAC
        SACH    L4N1,1
*
*****************************************************************
*
*   THE OUTPUT OF THE 8TH ORDER FILTER (SUM OF 2ND ORDER SUB-FILTERS) HAS
*   ONLY + THE MAGNITUDE (BY THEORY). SO THERE IS AN UPSCALE BY TWO
*
*****************************************************************
*
        ZALH    LY1             ; PROCESS RESULTS
        SUBH    LY2
        ADDH    LY3
        SUBH    LY4
        SACH    LY,1            ; UPSCALE BY TWO AND STORE RESULTS
*
*****************************************************************
*
*   8TH ORDER DETECTION WINDOW FOR DTMF HIGH BAND
*
*****************************************************************
*
        LAC     X,15
        LT      H1N2
        MPY     H1D
        LTD     H1N1
        MPY     H1C
        APAC
        SACH    HY1,1
        LTA     H2N2
        SACH    H1N1,1
*
        LAC     X,15
        MPY     H2D
        LTD     H2N1
        MPY     H2C
        APAC
        SACH    HY2,1
        LTA     H3N2
        SACH    H2N1,1
*
        LAC     X,15
        MPY     H3D
        LTD     H3N1
        MPY     H3C
        APAC
        SACH    HY3,1
        LTA     H4N2
        SACH    H3N1,1
*
        LAC     X,15
        MPY     H4D
        LTD     H4N1
        MPY     H4C
        APAC
        SACH    HY4,1
        APAC
        SACH    H4N1,1
*
        ZALH    HY1             ; PROCESS RESULTS
        SUBH    HY2
        ADDH    HY3
        SUBH    HY4
        SACH    HY,1            ; UPSCALE BT TWO AND STORE RESULT
*
        ZALS    PAUSE           ; LOOKING AT GAP ?
        BNZ     GAP1
*
        ZALS    GAP             ; AGC DOES NOT RUN DURING
        BNZ     T8              ; THE GAP
*
*****************************************************************
*
*   ACCUMULATE PEAKS
*
*****************************************************************
*
        LAC     LY              ; SAVE PEAK INFO FOR NOW
        ABS
        SACL    TEMPD
        SUB     ADJL
        BLEZ    PK
*
        LAC     TEMPD
        SACL    ADJL
PK      LAC     HY              ; SAVE PEAK INFO FOR HIGH
        ABS
        SACL    TEMPD
        SUB     ADJH
        BLEZ    PK1
*
        LAC     TEMPD
        SACL    ADJH
PK1     ZALS    CNTR1
        ADD     UNITY
        SACL    CNTR1
        NOP
        LACK    MIN
        SUB     CNTR1           ; SAVE PEAKS OVER MIN SAMPLES
        BGZ     T8
*
        ZAC
        SACL    CNTR1
*
        ZALS    CNTR2           ; SET AGC FLAG AFTER CNTR2 UPDATES
        ADD     UNITY
        SACL    CNTR2
                                ; LACK7
```

```
        LACK    6               ; 6 * 7 * 0.25 MS
        SUB     CNTR2
        BGZ     TSTW
*
        ZAC
        SACL    CNTR2
*
        LACK    1
        SACL    STOP            ; STOP WAITING FOR FILTER TRANSIENT
*
*****************************************************************
*
*   TEST FOR TWIST - SET LEVELS WITH TWSTHI (LO>HI) AND TWSTLO (HI>LO)
*
*****************************************************************
*
TSTW    LAC     ADJL
        SACL    TEMPD
        LAC     ADJL,8          ; TEST FOR TWIST BEFORE
        LTA     DJH
        MPYK    TWSTHI
        SPAC
        BGEZ    RSDTMF
*
        LAC     ADJH,8
        LTA     DJL
        MPYK    TWSTLO
        SPAC
        BGEZ    RSDTMF
*
ADJST   LAC     MINTHH          ; TEST FOR MINIMUM SIGNAL
        SUB     ADJH
        BGEZ    TCNT
*
        LAC     MINTHH          ; IN HI AND LO BAND
        SUB     ADJL
        BGEZ    TCNT
*
        LT      TEMPD           ; CALCULATE NEW THRESHOLDS
        MPYK    Z
        PAC
        SACH    THRSH8,4
        LT      ADJH
        MPYK    Z
        PAC
        SACH    THRHI,4
        LT      ADJL
        MPYK    Z
        PAC
        SACH    THRLO,4
*
*****************************************************************
*
*   THRESHOLD 8TH ORDER RESULT
*
*****************************************************************
*
T8      LAC     LY              ; GET LOW RESULT
        ABS                     ; MAKE POSITIVE
        SUB     THRSH8          ; APPLY THRESHOLD
        BLEZ    THR
*
        ZAC
        SACL    CNTR            ; ZERO TEMP CNTR
*
        B       SECND
*
THR     ZALS    CNTR            ; IF INPUT SIGNAL IS
        ADD     UNITY           ; GONE FOR MIN CONSECUTIVE
        SACL    CNTR            ; SAMPLES, RESET SYSTEM
        LACK    MIN
        SUB     CNTR            ; OR LOOK FOR GAP
        BGZ     TCNT
*
        ZALS    GAP             ; IS GAP REQUIRED ?
        BZ      RSDTMF
*
        ZAC
        SACL    GN1
        SACL    GN2
        SACL    Y5
*
        GAPC    ZAC
        SACL    SIGCNT          ; USE THIS FOR GAP COUNT
        SACL    CNTR
        LAC     THRSH8,13       ; GAP THRESHOLD = PREVS-
        SACH    TEMPD           ; 8TH THRESHOLD / 8  (DB)
        LACK1
        SACL    PAUSE           ; SET FLAG FOR GAP TRAP
*
GAP1    LAC     X1,12           ; THRESHOLD INPUT SAMPLE
        LT      GN1             ; HIGHPASS CHANNEL NOTAND
        MPY     A1              ; NOTCH OUT CP TONES
        LTA     GN2
        MPY     A2
        APAC
        SACH    Y5,1
        ZAC
        MPY     B2
        LTD     GN1
        MPY     B1
        LTD     Y5
        MPY     B0
        APAC
        SACH    Y5,1
        LAC     Y5
        ABS
        SUB     TEMPD
```

```
        BLEZ    GAP2
*
        ZALS    CNTR            ; ACCUMULATE SAMPLES
        ADD     UNITY           ; ABOVE GAP THRESHOLD
        SACL    CNTR
GAP2    ZALS    SIGCNT          ; INCREMENT GAP COUNT
        ADD     UNITY
        SACL    SIGCNT
*
*****************************************************************
*
*   INTER-DIGIT PAUSE IS DETERMINED BY FOLLOWING INSTRUCTION
*
*****************************************************************
*
CA1C    LACK    01Ch            ; LOAD GAP TIMER (07 MS + )
*
        SUB     SIGCNT          ; IS GAP TIME UP ?
        BGZ     AGAIN
*
        LACK    030h            ; SMOOTH OUT GLITCHES
        SUB     CNTR            ; IF THERE ARE TOO MANY SAMPLES
        BLEZ    GAPC            ; ABOVE THRESHOLD, DO GAP AGAIN
*
        LDPK    0
*
        ZAC
        SACL    GN1
        SACL    GN2
        SACL    Y5
        B       RSDTMF          ; RESET SYSTEM
*
*****************************************************************
*
*   THRESHOLD 2ND ORDER RESULTS
*
*****************************************************************
*
SECND   ZALS    GAP
        BNZ     AGAIN
*
        ZALS    STOP
        BZ      TCNT
*
        LAC     LY1
        ABS
        SUB     THRLO
        BLEZ    THR1
*
        ZALS    F1
        ADD     UNITY
        SACL    F1
THR1    LAC     LY2
        ABS
        SUB     THRLO
        BLEZ    THR2
*
        ZALS    F2
        ADD     UNITY
        SACL    F2
THR2    LAC     LY3
        ABS
        SUB     THRLO
        BLEZ    THR3
*
        ZALS    F3
        ADD     UNITY
        SACL    F3
THR3    LAC     LY4
        ABS
        SUB     THRLO
        BLEZ    THR4
*
        ZALS    F4
        ADD     UNITY
        SACL    F4
THR4    LAC     HY1
        ABS
        SUB     THRHI
        BLEZ    THR5
*
        ZALS    F5
        ADD     UNITY
        SACL    F5
THR5    LAC     HY2
        ABS
        SUB     THRHI
        BLEZ    THR6
*
        ZALS    F6
        ADD     UNITY
        SACL    F6
THR6    LAC     HY3
        ABS
        SUB     THRHI
        BLEZ    THR7
*
        ZALS    F7
        ADD     UNITY
        SACL    F7
THR7    LAC     HY4
        ABS
        SUB     THRHI
        BLEZ    TCNT
*
        ZALS    F8
        ADD     UNITY
        SACL    F8
```

```
*
TCNT    ZALS    GAP
        BNZ     AGAIN
*
        ZALS    SIGCNT          ; INCREMENT DTMFTONE COUNT
        ADD     UNITY
        SACL    SIGCNT
*
        LACK    SCNT            ; TEST FOR TIME UP
        SUB     SIGCNT
        BGZ     AGAIN
*
*************************************************************************
*
*   DETERMINE TONE DIGIT FROM SECOND ORDER COUNTERS
*
*   THE FREQUENCY IN EACH BAND WITH ENERGY ABOVE THE BIN THRESHOLD IS RECOG-
*   NISED AND THEN ZEROED. IN ORDER TO BE A VALID DTMF COMBINATION, ALL OTHER
*   BINS HAVE TO BE BELOW THRESHOLD.
*
*************************************************************************
*
        LARP    0
        LARK    0,0
        LARK    1,0
        LACK    THRHL1
        SUB     F1
        BGEZ    FI1
*
        SAR     1,F1
        B       FIND1
*
FI1     MAR     *+
        LACK    THRHL2
        SUB     F2
        BGEZ    FI2
*
        SAR     1,F2
        B       FIND1

FI2     MAR     *+
        LACK    THRHL3
        SUB     F3
        BGEZ    FI3
*
        SAR     1,F3
        B       FIND1
*
FI3     MAR     *+
        LACK    THRHL4
        SUB     F4
        BGEZ    NOFIND
*
        SAR     1,F4

FIND1   SAR     0,TEMPD
        LACK    LOLIM
        SUB     F1
        BLZ     NOFIND
*
        LACK    LOLIM
        SUB     F2
        BLZ     NOFIND
*
        LACK    LOLIM
        SUB     F3
        BLZ     NOFIND
*
        LACK    LOLIM
        SUB     F4
        BLZ     NOFIND
*
        LARK    0,0
        LACK    THRHH1
        SUB     F5
        BGEZ    FI5
*
        SAR     1,F5
        B       FIND2
*
FI5     MAR     *+
        LACK    THRHH2
        SUB     F6
        BGEZ    FI6
*
        SAR     1,F6
        B       FIND2
*
FI6     MAR     *+
        LACK    THRHH3
        SUB     F7
        BGEZ    FI7
*
        SAR     1,F7
        B       FIND2
 FI7    MAR     *+
        LACK    THRHH4
        SUB     F8
        BGEZ    NOFIND
*
        SAR     1,F8
 FIND2  .set    $
        LACK    HILIM
        SUB     F5
        BLZ     NOFIND
*
        LACK    HILIM
        SUB     F6
        BLZ     NOFIND
```

```
*
        LACK    HILIM
        SUB     F7
        BLZ     NOFIND
*
        LACK    HILIM
        SUB     F8
        BLZ     NOFIND
*
        LAC     TEMPD,2         ; + LOW-BAND OFFSET * 4
        SAR     0,TEMPD
        ADD     TEMPD           ;+ HIGH-BAND OFFSET
*
*****************************************************************
*
*   OUTPUT ROUTINE
*
*****************************************************************
*
DINT                            ; INTERRUPT PROTECTED BECAUSE THE OVERRUN
        SACL    DIGIT           ; BIT HAS NOT BEEN UPDATED YET
        LACK    070h
        ADD     DIGIT
        SACL    DIGIT
        LDPK    0
        LAC     ONE,DTINBT
        AND     STMODE
        BGZ     NOVRUN
*
        LDPK    1
        LACK    080h
        ADD     DIGIT
        SACL    DIGIT
        LDPK    0
*
NOVRUN  EINT
        LAC     OSHOLD
        SACL    OSTIME
*
        LAC     ONE,TPRFLG
        AND     FLAGS
*
        BZ      DTINT
*
        LAC     ONE,ONSFLG
        OR      FLAGS
        SACL    FLAGS
*
DTINT   LAC     ONE,DTINBT
        XOR     STMODE
        AND     STMODE
        SACL    STMODE
*
DATTEN  CALL    ATTEN
*
        CALL    XFUPD
*
        B       AGAIN
*
NOFIND  .set    $
        ZALS    TESTB
        ADD     UNITY                   ; INCREMENT BAD DIGITS
        SACL    TESTB
*
        LACK    1                       ; TONE NOT VALID
        SACL    GAP                     ; NOW LOOK FOR GAP
        B       AGAIN
*
*****************************************************************
*
*   ROUTINE: INTHDL
*
*   REFERENCE IN FLOWCHART: NONE
*
*   FUNCTION: INTERRUPT HANDLER
*
*****************************************************************
*
INTHDL  .set    $
*
        SST     SRSAVE                  ; SAVE STATUS REGISTER
        LDPK    1
        SACH    ACCUHI
        SACL    ACCULO                  ; SAVE CONTENTS OF ACCUMULATOR
        LDPK    0
        SAR     AR0,ARSAVE              ; SAVE CURRENT AUXILLIARY
*                                       ; REGISTER IN ARSAVE
        LARP    0                       ; POINT TO AR0
*
*****************************************************************
*
*   CHECK SOURCE OF INTERRUPT, EITHER CODEC OR PARALLEL INTERFACE.
*
*****************************************************************
*
        IN      ITEMP,CTLPRT            ; READ CONTROL REGISTER
        LAC     ONE,3
        AND     ITEMP                   ; CHECK FOR CODEC INTERRUPT BIT
*                                       ; SET
*
        BZ      NOTCDC
*
*****************************************************************
*
*   CODEC INTERRUPT HANDLER
*
*****************************************************************
*
```

```
CODEC   .set    $               ; CODEC INTERRUPT HANDLER
*
        ADDS    CTL320
        SACL    ITEMP
        OUT     ITEMP,CTLPRT    ; CLEAR CODEC INTERRUPT
*
        LAC     ONE,INTFLG
        OR      FLAGS
        SACL    FLAGS           ; SET CODEC INTERRUPT INDICATOR FLAG
*
QREAD   LAR     AR0,QIN         ; LOAD UP THE QIN POINTER
*
        IN      *,CDCPRT        ; READ NEXT LINEARIZED SAMPLE INTO
*                               ; QUEUE IN SIGNED MAGNITUDE FORM.
*
        LAC     QIN
        SUB     ONE             ; DECREMENT THE QIN POINTER.
        SACL    ITEMP
*
        LAC     ONE,3
        OR      ITEMP           ; POINTER COUNTS 003Fh THRU 0038h
        SACL    QIN             ; UPDATE QIN
        B       CINEND
INTEND  .set    $               ; COMMON EXIT PATH FROM INTERRUPT HANDLER
*
        LACK    7
        ADDS    CTL320
        SACL    ITEMP
        OUT     ITEMP,CTLPRT    ; CLEAR ALL LATCHED NON-CODEC INTERRUPTS
*
CINEND  LAR     AR0,ARSAVE      ; RESTORE AR0
        LDPK    1
        ZALH    ACCUHI
        ADDS    ACCULO          ; RESTORE ACCUMULATOR
        LST     SRSAVE          ; RESTORE STATUS REGISTER
        EINT
        RET
*
*****************************************************************
*
*   PARALLEL INTERFACE INTERRUPT HANDLER
*
*****************************************************************
*
NOTCDC  .set    $               ; PARALLEL INTERFACE INTERRUPT HANDLER
*
        LAC     ONE,STAFLG
        AND     FLAGS           ; CHECK STATE BIT, IF IT IS SET WE ARE HALF
*                               ; WAY THROUGH A WRITE OPERATION AND MUST
*                               ; DO THE SECOND PAIR OF TRANSFERS.
        BGZ     WRITE2
*
*****************************************************************
*
*   THE STATE BIT WAS NOT SET, SO WE ARE AT THE BEGINNING OF A TRANSFER
*   OPERATION, EITHER READ OR WRITE. READ OPERATIONS REQUIRE TWO TRANSFERS
*   ONE EACH WAY, WRITE TRANSFERS REQUIRE TWO TRANSFERS IN EACH DIRECTION AND
*   TWO INTERRUPTS, WHICH IS WHY A STATE BIT IS REQUIRED TO FLAG THE SECOND
*   HALF OF A WRITE OPERATION.
*
*****************************************************************
*
        IN      ITEMP,DATPRT    ; READ COMMAND FROM INTERFACE.
*
        LAC     ONE,5           ; MASK INTERFACE COMMAND TO 5 BITS
        SUB     ONE
        AND     ITEMP
        SACL    ITEMP
*
        LAC     ONE,RWBIT
        AND     ITEMP           ; CHECK RW BIT, IF IT IS SET, THIS IS A
*                               ; READ.
*
        BGZ     READOP
*
*****************************************************************
*
*   THIS IS THE FIRST PART OF A WRITE TRANSFER
*
*****************************************************************
*
WRITE1  .set    $
*
        LAC     ONE,STAFLG
        OR      FLAGS           ; SET THE STATE BIT TO FLAG THAT THE FIRST
        SACL    FLAGS           ; PART OF A WRITE OPERATION HAS BEEN DONE.
*
        LAC     ITEMP
        SACL    CMSAVE          ; SAVE THE COMMAND BYTE FOR THE SECOND PART
*                               ; OF THE WRITE TRANSFER.
ACKNOW  .set    $
*
        LAC     STMODE,8        ; ACKNOWLEDGE WRITE BY WRITING OUT STATUS
        SACH    ITEMP           ; TO STATUS PORT
        LAC     ITEMP
        AND     MSOOFF
        SACL    ITEMP
        OUT     ITEMP,STAPRT    ; ALSO CLEARS HARDWARE INTERRUPT SOURCE.
*
        B       INTEND
*
*****************************************************************
*
*   THIS IS THE FIRST PART OF A READ TRANSFER
*
*****************************************************************
*
READOP  .set    $
```

```
*
        LACK    INTTAB          ; LOAD UP THE START ADDRESS OF INTTAB
        ADDS    ITEMP           ; ADD ON THE INTERFACE COMMAND, WHICH IS AN
*                               ; OFFSET INTO THE INTTAB TABLE OF REGISTER
*                               ; MAPPINGS.
        TBLR    ITEMP           ; READ THE REGISTER MAPPING.
*
        LAR     AR0,ITEMP       ; MAPPING WORD IN AR0. THE MAP WORD
*                               ; INDICATES WHICH PHYSICAL ALLOCATION IS TO
*                               ; BE ACCESSED IN ITS LOWER 9 BITS, AND ALSO
*                               ; CONTAINS CONTROL FLAGS ON THE UPPER BYTE.
*                               ; THESE CONTROL FLAGS ARE USED BY THE
*                               ; INTERRUPT HANDLER TO INDICATE WHAT TYPE
*                               ; OF TRANSFER IS HAPPENING.
*
        LAC     ONE,TESTBT      ; CHECK FOR TEST BIT SET IN MODE REGISTER
        AND     STMODE          ; WHICH MEANS THAT TEST MODE IS CURRENTLY
*                               ; ON
        BZ      NOTEST
*
        LAC     ONE,LBIT        ; CHECK FOR L BIT SET IN MAP WORD
        AND     ITEMP           ; WHICH MEANS AN ACCESS OF ADDR 0 OR 1
*
        BGZ     NOTEST
*
        OUT     CMSAVE,DATPRT   ; ALL OTHER REGISTERS IN TEST MODE ARE
*                               ; MAPPED ONTO THE SAME REGISTER (CMSAVE).
        B       INTEND
*
NOTEST  LAC     ONE,TBIT        ; CHECK FOR T BIT SET IN MAP WORD WHICH
        AND     ITEMP           ; MEANS A READ OF CURRENT TIME
*
        BGZ     TBTSET
*
        LAC     ONE,SBIT        ; CHECK FOR S BIT SET IN MAP WORD WHICH
        AND     ITEMP           ; MEANS A READ OF THE STATUS REGISTER
*
        BGZ     SBTSET
*
        LAC     ONE,UBIT        ; CHECK FOR U BIT SET IN MAP WORD WHICH
        AND     ITEMP           ; MEANS A READ OF AN UPPER BYTE
*
        BGZ     UBITRD
*
*****************************************************************
*
*   THIS IS A READ OF THE LOWER BYTE OF THE LOCATION SPECIFIED IN THE LOWER 9
*   BITS OF THE MAP WORD AND ALSO IN AR0.
*
*****************************************************************
*
        OUT     *,DATPRT
*
        B       INTEND
*
*****************************************************************
*
*   THIS IS A READ OF THE CURRENT TIME MS REGISTER.
*
*****************************************************************
*
TBTSET  .set    $
*
        LAC     CRTIME
        SACL    CRHOLD          ; SAVEW CURRENT TIME
*
        B       UBITRD
*
*****************************************************************
*
*   THIS IS A READ OF THE STATUS REGISTER.
*
*****************************************************************
*
SBTSET  .set    $
*
        LAC     STMODE,8
        SACH    ITEMP
        OUT     ITEMP,STAPRT    ; WRITE OUT MS BYTE OF STMODE
*
        B       INTEND
*
*****************************************************************
*
*   THIS IS A READ OF THE UPPER BYTE OF THE LOCATION SPECIFIED IN THE LOWER 9
*   BITS OF THE MAP WORD
*
*****************************************************************
*
UBITRD  .set    $
*
        LAC     *,8
        SACH    ITEMP
        OUT     ITEMP,DATPRT
*
        B       INTEND
*
*****************************************************************
*
*   THIS IS THE SECOND PART OF A WRITE TRANSFER
*
*****************************************************************
*
WRITE2  .set    $
*
        XOR     FLAGS           ; CLEAR STATE BIT IN FLAGS REGISTER
        SACL    FLAGS
*
```

```
        LACK    INTTAB
        ADDS    CMSAVE          ; ADD ON THE SAVED INTERFACE COMMAND, WHICH
*                               ; IS AN OFFSET INTO THE INTTAB TABLE OF
*                               ; REGISTER MAPPINGS.
        TBLR    ITEMP           ; READ THE REGISTER MAPPING.
*
        LAR     AR0,ITEMP       ; MAPPING WORD IN AR0.
*
        IN      CMSAVE,DATPRT   ; READ THE DATA IN WHICH IS TO BE WRITTEN
*                               ; TO A REGISTER, USE CMSAVE FOR THIS.
        LAC     CMSAVE
        AND     MSOOFF
        SACL    CMSAVE          ; MASK OUT UNDEFINED BITS
*
        LAC     ONE,MBIT        ; CHECK FOR M BIT SET IN MAP WORD
        AND     ITEMP           ; WHICH MEANS A WRITE TO THE MODE REGISTER
*
        BGZ     MBTSET
*
        LAC     ONE,TESTBT      ; CHECK FOR TEST BIT SET IN MODE REGISTER
        AND     STMODE          ; WHICH MEANS A TEST MODE IS CURRENTLY ON
*
        BZ      CHECKF          ; TEST MODE NOT SET
*
        LAC     ONE,LBIT        ; CHECK FOR L BIT SET IN MAP WORD WHICH
        AND     ITEMP           ; MEANS A WRITE TO ADDRESS 0 OR 1
*
        BZ      ACKNOW          ; TEST MODE. LEAVE DATA IN CMSAVE
*
CHECKF  LAC     ONE,FBIT        ; CHECK FOR F BIT SET IN MAP WORD WHICH
        AND     ITEMP           ; MEANS A WRITE TO A FREQUENCY REGISTER
*
        BGZ     FBTSET
*
        LAC     ONE,UBIT        ; CHECK FOR U BIT SET IN MAP WORD WHICH
        AND     ITEMP           ; MEANS A WRITE TO AN UPPER BYTE
*
        BGZ     UBITWR
*
*****************************************************************
*
*   THIS IS THE SECOND PART OF A WRITE TO THE LOWER HALF OF A REGISTER
*
*****************************************************************
*
LOWRT   LAC     MSOOFF,8
        AND     *               ; ADD THE CURRENT CONTENTS OF THE UPPER
LOWR2   ADDS    CMSAVE          ; REGISTER TO THE NEW LOWER HALF
        SACL    *
*
        B       ACKNOW
*
*****************************************************************
*
```

```
*   THIS IS A WRITE TO THE MODE REGISTER
*
*****************************************************************
*
MBTSET  .set    $
*
*****************************************************************
*
*   OPERATIONS TO BE PERFORMED HERE ARE:
*
*   1. COPY THE BYTE WRITTEN, INTO THE MODE REGISTER, PRESERVING THE STATE
*      OF THE RC BITS WHICH ARE ALREADY IN THE MODE REGISTER IN POSITIONS 0
*      AND 1.
*   2. SET THE SAMPLE SCALE FACTOR ACCORDING TO THE SC BITS.
*   3. IF THE TEST BIT HAS BEEN SET, THEN CLEAR THE DTMF AND TONE BITS.
*   4. IF ANY INTERRUPTS HAVE BEEN ACKNOWLEDGED (IACK BITS SET) THEN SET THE
*      APPROPRIATE STATUS BIT BACK TO A 1 AND UPDATE THE STATUS OF THE XF
*      FLAG.
*   5. IF THE TEST BIT IS SET, THEN DO A SELF TEST, ACKNOWLEDGE THE WRITE,
*      THEN RESTART, ELSE ACKNOWLEDGE THE WRITE AND RETURN FROM INTERRUPT.
*
*****************************************************************
*
        LAC     MSOOFF,8
        ADD     ONE,1
        ADD     ONE             ; MASK IN STATUS BYTE AND RC BITS IN MODE
*                               ; BYTE.
*
        AND     STMODE          ; ZERO MODE BYTE, EXCEPT FOR THE BOTTOM TWO
        SACL    STMODE          ; BITS, AND STORE BACK IN STMODE
*
        LACK    3
        AND     CMSAVE          ; EXTRACT THE SC BITS FROM THE BYTE BEING
        SACL    ITEMP           ; WORKED
*
        LACK    SCATAB          ; ADD IN TABLE OFFSET
        ADD     ITEMP
        TBLR    SCALEF          ; READ THE DESIRED SCALE FACTOR INTO SCALEF
*
        LAC     ONE,8
        SUB     ONE,2           ; MASK IN LOWER BYTE, EXCEPT FOR THE BOTTOM
*                               ; TWO (SC) BITS, OF THE BYTE BEING WRITTEN
*                               ; TO MODE
        AND     CMSAVE
        ADDS    STMODE          ; ADD TP STMODE. MODE IS NOW UPDATED, BUT
        SACL    STMODE          ; THE BOTTOM TWO (RC) BITS HAVE BEEN LEFT
*                               ; INTACT IN THE READABLE VERSION OF MODE.
*
        LAC     ONE,TESTBT      ; CHECK FOR TEST BIT SET IN MODE REGISTER
        AND     STMODE          ; WHICH MENAS THAT TEST MODE IS CURRENTLY
*                               ; ON
*
        BZ      CLRACK
*
```

```
TESTW   B       SLFTST          ; BRANCH TO THE SELF TEST ROUTINE
*
CLRACK  LAC     CMSAVE,14
        SACH    CMSAVE
        LACK    7
        AND     CMSAVE          ; MASK IN THE THREE IACKBITS
        SACL    ITEMP
*
        LACK    ACKTAB          ; ADD IN TABLE OFFSET
        ADD     ITEMP
*
        TBLR    ITEMP
        LAC     ITEMP           ; STATUS BIT TO BE SET TO 1, IF ANY
        OR      STMODE
        SACL    STMODE
*
ENDSLF  .set    $               ; SELF TEST BRANCHES BACK TO HERE
*
        CALL    XFUPD           ; UPDATE XF FLAG
*
        B       ACKNOW
*
*****************************************************************
*
*   THIS IS THE SECOND PART OF A WRITE TO A FILTER CENTER FREQUENCY
*
*****************************************************************
*
FBTSET  .set    $
*
        LAC     MSOOFF,8
        AND     FMSFL           ; ADD THE CURRENT CONTENTS OF FREQUENCY MS
*                               ; BYTE
*
        B       LOWWR2
*
*****************************************************************
*
*   THIS IS THE SECOND PART OF A WRITE TO THE UPPER HALF OF A REGISTER
*
*****************************************************************
*
UBITWR  .set    $
*
        LAC     MSOOFF
        AND     *               ; ADD THE CURRENT CONTENTS OF THE LOWER
        ADD     CMSAVE,8        ; REGISTER TO THE NEW UPPER HALF
        SACL    *
*
        CALL    XFUPD           ; UPDATE THE XF FLAG (ONLY AFFECTED BY
*                               ; WRITES TO THE CONTROL TEGISTER.)
        B       ACKNOW
*
*****************************************************************
*
*   ROUTINE: CRESET
*
*   REFERENCE IN FLOWCHART: NONE
*
*   FUNCTION: COLD RESET HANDLER
*
*****************************************************************
*
*                               ; INITIALISE PROCESSOR
*
CRESET  .set    $               ; COLD RESET. BRANCH HERE FROM RESET
*                               ; VECTOR.
*
        LDPK    0               ; INITIALIZE DATA PAGE POINTER
*
        BV      CLROVF          ; CLEAR OVERFLOW FLAG
*
CLROVF  SOVM                    ; SET OVERFLOW MODE
*
        LARP    0
*
        CALL    WRESET
*
        CALL    RSTFIL          ; CALL RESET FILTERING ROUTINE TO CLEAR
*                               ; DOWN ALL ACCUMULATORS AND SET UP FILTER
*                               ; READY FOR THE FIRST BLOCK.
*
        CALL    RSDTMF          ; REINITIALIZE THE DTMF CODE
*
        EINT                    ; ENABLE INTERRUPTS
        B       MAIN            ; JUMP TO MAIN (MAINSTREAM CODE)
*
*****************************************************************
*
*   ROUTINE: WRESET
*
*   REFERENCE IN FLOWCHART: NONE
*
*   FUNCTION: WARM RESET HANDLER
*
*****************************************************************
*
*                               ; RE-INITIALIZE PROCESSOR
*
*****************************************************************
*
WRESET  .set    $               ; WARM RESET.  CALLED BY SELF-TEST ROUTINE
*                               ; AND BY COLD RESET HANDLER.
*
*****************************************************************
*
*   ZERO ALL RAM LOCATIONS IN PAGES 0 & 1
*
```

```
*****************************************************************
*
        LARK    AR0,0FFh        ; SET UP AR0 TO CONTROL COPYING LOOP
        ZAC
*
ZER00   LAR     PAR0
        SACL    *
        BANZ    ZER00
*
        LACK1
        SACL    ONE
        LDPK    1
        SACL    UNITY
        LDPK    0
*
*****************************************************************
*
*   INITIALIZE DTMF MEMORY LOCATIONS IN PAGE 1
*
*****************************************************************
*
        LARK    AR0,(IEND0-IVAR0-1) ; SET UP AR0 TO CONTROL COPYING LOOP
        LARK    AR1,(IEND0-1)  ; SET UP AR1 TO POINT TO DATA SUM
*
        LACK    CONEN0         ; LOAD ACCUMULATOR WITH (1 + END OF TABLE)
*
COPY0   SUB     ONE
        LAR     PAR1
        TBLR    *-,AR0
        BANZ    COPY0
*
*****************************************************************
*
*   INITIALIZE TONE DETECTOR LOCATIONS IN PAGE 0
*
*****************************************************************
*
        LARK    AR0,(IEND1-IVAR1-1) ; SET UP AR0 TO CONTROL COPYING LOOP
        LARK    AR1,IEND1-1    ; SET UP AR1 TO POINT TO DATA RAM.
*
        LACK    CONEN1         ; LOAD ACCUMULATOR WITH (1 + END OF TABLE)
*
COPY1   SUB     ONE
        LAR     PAR1
        TBLR    *-,AR0
        BANZ    COPY1
*
        OUT     CTL320,CTLPRT  ; SET UP LOWER CONTROL REGISTER BITS TO
*                              ; FD9Fh
        OUT     CTL32U,CTLUPR  ; WRITE VALUE OCFEh TO UPPER CONTROL PORT
*
        LAC     CTL322         ; RESET THE PORT 0 CONTROL BIT TO POINT AT
*                              ; THE LOWER CONTROL PORT, AND SET SCLK TO
*                              ; BE AN OUTPUT. CLEAR THE INTERRUPT ACKNOW-
*                              ; LEDGE BITS FROM CTL320.
        SACL    CTL320         ; KEEP THIS AS THE DEFAULT VAIUE OF CTL320
*                              ; IN RAM
        OUT     CTL320,CTLPRT  ; SET UP LOWER CONTROL REGISTER BITS TO
*                              ; 7C90h
*
        RET
*
*****************************************************************
*
*   ROUTINE: ATTEN
*
*   REFERENCE IN FLOWCHART: NONE
*
*   FUNCTION: WRITE OUT STATUS TO DRAW ATTENTION TO A CHANGE IN ONE OR MORE
*             OF THE STATUS BITS.
*
*****************************************************************
*
ATTEN   .set    $
*
        LAC     STMODE,8
        SACH    ITEMP
        OUT     ITEMP,ATTPRT   ; WRITE OUT MS BYTE OF STMODE
*
        RET
*
*****************************************************************
*
*   ROUTINE: XFUPD
*
*   REFERENCE IN FLOWCHART: NONE
*
*   FUNCTION: UPDATE THE XFFLAG. CALLED EVERY TIME A STATUS REGISTER
*             INTERRUPT FLAG IS UPDATED, AND EVERY TIME THE CONTROL REGISTER
*             IS WRITTEN TO.
*
*****************************************************************
*
XFUPD   .set    $
*
        LAC     MSOOFF,8
        XOR     STMODE
        AND     CTLTSL
*
        SACL    ITEMP
        LAC     MSOOFF,8
        AND     ITEMP          ; ANY BITS NOW SET MEAN PENDING ENABLED
*                              ; INTERRUPTS ?
        BZ      SETXF
*
        CLRXF   LACONE,CXFBIT  ; INTERRUPT(S) ASSERTED
        XOR     CTL320
        AND     CTL320         ; CLEAR XF BIT
```

```
        SACL    CTL320
*
        RET
*
SETXF   LAC     ONE,CXFBIT      ; NO INTERRUPT
        OR      CTL320          ; SET XF BIT
        SACL    CTL320
*                               ; CTL320 WILL GET WRITTEN TO THE CONTROL
*                               ; REGISTER DURING THE NEXT CODEC INTERRUPT.
        RET
*
*****************************************************************
*
*   ROUTINE: SLFTST
*
*   REFERENCE IN FLOWCHART: NONE
*
*   FUNCTION: SELF TEST OF PROCESSOR. PERFORM INTERNAL RAM TEST AND ROM
*             CHECKSUM TEST. SELF TEST USES THE STACK AS A HOLDING REGISTER.
*             AT THE END OF THE SELF TEST, THE PROCESSOR IS REINITIALIZED AND
*             BRANCHES INTO THE MAIN STREAM CODE. THE CONTENTS OF THE STACK
*             ARE DISCARDED. EACH TIME A VALUE IS PUSHED, THE STACK IS FIRST
*             POPPED, TO PREVENT STACK OVERFLOW MESSAGES FROM OCCURRING ON
*             SIMULATORS. THESE REDUNDANT POP INSTRUCTIONS MAY BE REMOVED;
*             THEY ARE MARKED (+) IN THE CODE.
*
*****************************************************************
*
SLFTST  .set    $
*
        LACK    ROMFAI
        PUSH                    ; PUT ROM FAIL RESULT ONTO STACK
*
ROMTST  .set    $
*
*****************************************************************
*
*   ROM CHECKSUM TEST
*
*****************************************************************
*
        ZAC
        SACL    TOTAL           ; SET CHECKSUM TO ZERO
*
        LACK    1
        SACL    ONE             ; RESTORE THE UNITY LOCATION
        LDPK    1
        SACL    UNITY
*
        LAC     MARKER          ; PROGRAM END ADDRESS IN ACCUMULATOR
*
        LDPK    0
ROMLP   SUB     ONE
        TBLR    ROMVAL
*
        SACL    ACCHLD
*
        ZALS    ROMVAL
LABEL2  ADDS    TOTAL
        SACL    TOTAL
*
        LAC     ACCHLD
        BNZ     ROMLP
*
        LAC     TOTAL           ; CHECK FOR ZERO CHECKSUM
        BNZ     RESULT
*
*****************************************************************
*
*   RAM TEST
*
*****************************************************************
*
        POP                     ; (+) SEE NOTE IN ROUTINE HEADER
        LACK    RAMFAI
        PUSH                    ; PUT RAM FAIL RESULT INTO STACK
        LARK    AR0,0FFh        ; AR0 COUNTS 0100h ITERATIONS OF OUTER
*                               ; LOOP
        LARK    AR1,15          ; AR1 COUNTS 16 ITERATIONS OF INNER LOOP
        LACK    1               ; START WITH ONE
INLP    LARP    AR0
        SACL    *               ; STORE IN RAM LOCATION
        SUBS    *               ; READ IT BACK AND CHECK IT BY SUBTRACTION
        BNZ     RESULT
*
        LAC     *,1,AR1         ; SHIFT OPERAND LEFT BY ONE BIT, REPEATING
        BANZ    INLP            ; FOR EACH BIT POSITION.
*
        BGEZ    RESULT          ; FINALLY CHECK THAT BIT HAS SHIFTED OUT OF
*                               ; LOWER ACCUMULATOR (FFFF0000h IN ACC)
        MAR     *,0             ; FLIP BACK TO AR0, REPEAT WHOLE FOR EACH
        SACL    *               ; LEAVE TESTED LOCATION AT ZERO RAM
        BANZ    OUTLP           ; LOCATION
*
CDCTST  .set    $
*
*****************************************************************
*
*   CODEC INTERRUPT CHECK
*
*****************************************************************
*
        POP                     ; (+) SEE NOTE IN ROUTINE HEADER.
*
        CALL    WRESET          ; DO A WARM RESET TO RE-INITIALIZE ALL
*                               ; VARIABLES
*
        ZALH    ONE
```

```
        SUB     ONE,7           ; CREATE MASK TO ENABLE ONLY CODEC
*                               ; INTERRUPTS
        AND     CTL320
        SACL    CTL320
*
        OUT     CTL320,CTLPRT   ; ENABLE CODEC INTERRUPTS ONLY, IN MASK.
*                               ; ONLY TAKES EFFECT AFTER THE NEXT EINT
        LARP    AR0
        LARK    AR0,INTMAX      ; LOAD AR0 WITH THE MAXIMUM NUMBER OF LOOPS
*                               ; EXPECTED BETWEEN INTERRUPTS. LOOPA IS 6
*                               ; CYCLES LONG.
        EINT                    ; CODEC INTERRUPTS NOW ENABLED.
*
        LAC     ONE,INTFLG
        AND     FLAGS
        XOR     FLAGS
        SACL    FLAGS           ; CLEAR INTERRUPT INDICATOR FLAG. AWAIT
*                               ; NEXT INTERRUPT, COUNTING LOOPS WITH AR0
LOOPA   LAC     ONE,INTFLG
        AND     FLAGS
        BNZ     FSTINT          ; BRANCH WHEN NEXT CODEC INTERRUPT ARRIVES
*
        BANZ    LOOPA           ; LOOP UNTIL AR0 = 0 OR UNTIL AN INTERRUPT
*
        B       CDCERR          ; IF AR0 REACHES 0 THEN CODEC ERROR.
*
FSTINT  XOR     FLAGS
        SACL    FLAGS           ; CLEAR INTERRUPT INDICATOR FLAG
*
        LARK    AR0,INTMAX      ; LOAD AR0 WITH THE MAXIMUM NUMBER OF LOOPS
*                               ; EXPECTED BETWEEN INTERRUPTS. LOOPB IS 6
*                               ; CYCLES LONG. AWAIT NEXT INTERRUPT,
*                               ; COUNTING LOOPS WITH AR0
*
LOOPB   LAC     ONE,INTFLG
LABEL3  AND     FLAGS
        BNZ     GOTINT          ; NEXT INTERRUPT HAS OCCURRED
*
        BANZ    LOOPB           ; AUX REGISTER NOT ZERO YET. LOOP
*                               ; AUX REGISTER ZERO. INTERRUPTS TOO
*                               ; INFREQUENT
*
CDCERR  .set    $
*
        LACK    CDCFAI
        PUSH                    ; PUT CODEC INTERRUPT FAILURE RESULT ON
*                               ; STACK
        B       RESULT
*
GOTINT  SAR     AR0,ITEMP       ; CHECK FOR INTERRUPTS TOO FREQUENT
        LACK    INTLFT
        SUB     ITEMP
        BLZ     CDCERR
*
```

```
*****************************************************************
*
*   IF PATH REACHES HERE, ALL THE TESTS HAVE BEEN SUCCESSFUL
*
*****************************************************************
*
        LACK    PASS
        PUSH                    ; PUT TEST PASS RESULT ONTO STACK
*
*****************************************************************
*
*   END OF CODEC INTERRUPT CHECK
*
*****************************************************************
*
RESULT  .set    $
*
        DINT                    ; DISABLE INTERRUPTS.
        CALL    WRESET          ; DO A WARM RESET TO REINITIALIZE THE
*                               ; PROCESSOR
*
        POP                     ; RETRIEVE TEST RESULT FROM STACK
        ADDS    STMODE
        ADD     ONE,7           ; RESTORE TEST MODE BIT.
        SACL    STMODE
*
        CALL    RSTFIL          ; CALL RESET FILTERING ROUTINE TO CLEAR
*                               ; DOWN ALL ACCUMULATORS AND SET UP FILTER
*                               ; READY FOR THE NEXT BLOCK.
*
        CALL    RSDTMF          ; REINITIALIZE THE DTMF CODE
*
        LAC     STMODE,8        ; ACKNOWLEDGE THE WRITE WHICH CAUSED THE
        SACH    ITEMP           ; SELFTEST BY WRITING OUT STATUS
*
        LAC     ITEMP
        AND     MSOOFF
        SACL    ITEMP
        OUT     ITEMP,STAPRT
*
        LAC     ONE,3           ; CLEAR ALL NON-CODEC INTERRUPTS SO THAT
        SUB     ONE             ; ANY SPURIOUS FSX OR FSR INTERRUPTS
        ADDS    CTL320          ; GENERATED BY SUB-STANDARD HARDWARE WON'T
        SACL    ITEMP           ; HANG UP THW SYSTEM.
        OUT     ITEMP,CTLPRT    ; CLEAR ALL LATCHED NON-CODEC INTERRUPTS
*
        EINT
        B       MAIN            ; IGNORE THE STACK AND RESTART.
PRGEND  .word   0               ; END OF PROGRAM MARKER FOR CHECKSUM
*                               ; ROUTINE
*
        .end
```

```
Program Tone_Detector_Demo (INPUT,OUTPUT);

{ Tone Detection Demonstration Program }

{ This program is written in Turbo Pascal and has been tested using an
  IBM PC-AT(X) with PC-DOS version 3.30 and Turbo Pascal version 3.02A

  The program implements the interface described in section 4 of this
  application report.

  The program is terminated by typing '0' at the PC keyboard.

  The program is included for illustrative purposes only. It may be used
  as a whole or in part for the evaluation of the tone detector. Certain
  changes may be necessary in order to ensure correct operation with a
  particular PC, operating system or Pascal Compiler.                    }

{ Turbo Pascal is a registered trademark of Borland International, Inc.
  IBM is a registered trademark of International Business Machines Corp }


{ Written by   Craig Marven   May 1988     }

{$C-,U-} {Compiler directive for correct operation of Keypressed function}

Const
   ERRORX  = 1;   {X coordinate for error message     }
   ERRORY  = 24;  {Y coordinate for error message     }
   OFFSET       = $300;  {Change this if board address is not 0300h}
   WRITEBLOCK  = 23;   {Column address for write data     }
   READBLOCK  = 66;   {Column address for read data      }
   DISP : Array[1..50] of Integer = (0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,
                                     2,0,0,0,0,0,0,0,0,
                                     0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,
                                     1,1,0,0,0,0,0,0,0);
     {Cursor Y displacement for any given row position}
   FIRSTCOL : Array[5..19] of Integer = (1,1,5,5,5,5,5,6,5,5,5,5,5,5,5,3);
     {Initial column for each write data row   }

Type
   Eightstring = Array[1..8] of Char;
   Errorstring = Array[1..24] of Char;
   Table       = Array[5..19] of Eightstring;

Var
   SAVEX,SAVEY : Integer;      {Temporary cursor location storage }
   I       : Integer;      {Temporary variable    }
   BLOCK,COL   : Integer;
   ROW,CODE    : Integer;
   GAIN_FACTOR : Real;         {Current gain setting of tone detector}
   CMD         : Char;         {Last character input from keyboard}
   ERRFLAG     : Boolean;      {Error flag}
   TEMP_CELL   : Eightstring;  {Temporary keyboard input string}
   WRITE_TABLE : Table;        {Array containing data values for update of
                                tone
                                    detector registers}

Procedure Soft_cursor (COMMAND : Integer);
{ This procedure places, or removes a software cursor at the current location
  of the screen cursor. This cursor remains in place while screen updates go
  on elsewhere }
Var
   CHARACTER : Char;
Begin
  Case COMMAND of
     1 : Begin
           CHARACTER := TEMP_CELL[COL];
           Textcolor(black);
           Textbackground(white);
           Write(CHARACTER);
           Textcolor(14);
           Textbackground(0);
           Gotoxy(Wherex-1,Wherey);
         End;
     0 : Begin
           CHARACTER := TEMP_CELL[COL];
           Textcolor(14);
           Textbackground(0);
           Write(CHARACTER);
           Gotoxy(Wherex-1,Wherey);
         End;
  End; {Case}
End; {Soft_cursor}

Procedure Error_message (MESSAGE : Errorstring);
{ This procedure places a message in red (blinking) at the normal error message
  location and produces a 'bleep'}
Begin
   Gotoxy(ERRORX,ERRORY);
   Textcolor(12+BLINK);
   Write(MESSAGE);
   Sound(1100);
   Delay(500);
   Nosound;
   Textcolor(7);
End; {Error_message}

Procedure Cell_check (CELL : Eightstring ; LROW : Integer ; Var LCOL : Integer
;
                      Var ERRFLAG : Boolean);
{ This procedure checks the contents of a new input from the keyboard before
  before allowing it to be passed on to the remainder of the program. It tests
  for invalid digits, multiple decimal points, and trailing blanks, none of
  which are allowed }
Var
   I           : Integer;
   DIGIT_FOUND : Boolean;
   POINT_FOUND : Boolean;
```

```
Begin
   ERRFLAG := False;
   DIGIT_FOUND := False;
   POINT_FOUND := False;
   LCOL := 0;
   Repeat
      LCOL := LCOL + 1;
      Case LROW of
         5,6,19 : Case CELL[LCOL] of
                     '0','1' : DIGIT_FOUND := TRUE;
                     Else
                     Begin
                        ERRFLAG := True;
                        Error_message('Invalid binary digit   ')
                     End; {Else}
                  End; {Case}
         7..18  : Case CELL[LCOL] of
                     '0'..'9': DIGIT_FOUND := True;
                     '.'     : If POINT_FOUND then
                               Begin
                                  Error_message('Multiple decimal points ');
                                  ERRFLAG := True;
                               End {If}
                               Else POINT_FOUND := True;
                     ' '     : If DIGIT_FOUND then
                               Begin
                                  Error_message('Trailing blanks invalid ');
                                  ERRFLAG := True;
                               End; {If}
                     Else
                     Begin
                        ERRFLAG := True;
                        Error_message('Invalid decimal digit   ');
                     End; {Else}
                  End; {Case}
      End; {Case}
   Until (ERRFLAG = True) or (LCOL = 8);
   If not DIGIT_FOUND then
   Begin
      Error_message('A number must be input  ');
      ERRFLAG := True;
   End; {If}
   If ERRFLAG = False then
   Begin
      Gotoxy(ERRORX,ERRORY);
      Write('                        ');
   End; {If}
End; {Cell_check}

Function Binary_to_int (Var CELL : Eightstring) : Integer;
{ This function converts an 8 digit binary number to a decimal integer}
Var
   I,TEMP,POWER : Integer;
Begin
   POWER := 1; TEMP := 0;
   For I := 8 downto 1 do
   Begin
      If CELL[I] = '1' then TEMP := TEMP + POWER;
      POWER := POWER + POWER;
   End; {For}
   Binary_to_int := TEMP;
End; {Binary_to_int}

Procedure Outbin (INT,NUMDIG : Integer);
{ This recursive procedure outputs an integer as a binary number of any
  length}
Begin
   If NUMDIG > 1 then outbin(INT div 2,NUMDIG-1);
   Textcolor(14);
   Write(chr(INT mod 2 + 48));
   Textcolor(0);
End; {Outbin}

Procedure Zero_fill (Var CELL : Eightstring);
{ This procedure replaces leading blanks on a keyboard input with zeroes}
Var  X : integer;
Begin
   For X := 1 to 8 do If CELL[X] = ' ' then CELL[X] := '0';
End; {Zero_fill}

Function Getreg (RNUM : Byte) : Byte;
{ This function reads the current value of any tone detector register}
Var
   STATUS : Integer;
Begin
   port[OFFSET] := RNUM + 16;
   Delay(1);
   Getreg := port[OFFSET];
   port[OFFSET] := 16;
   Delay(1);
   STATUS := port[OFFSET];
End; {Getreg}

Procedure Putreg (RNUM : Byte ; VALUE : Byte );
{ This procedure puts a new value into any tone detector register}
Var
   STATUS : Integer;
Begin
   port[OFFSET] := RNUM;
   Delay(1);
   STATUS := port[OFFSET];
   port[OFFSET] := VALUE;
   Delay(1);
   STATUS := port[OFFSET];
End; {Putreg}

Procedure Update_register(LROW : Integer ; Var ERRFLAG : Boolean);
{ Update register is called by the user pressing return after entering some
```

```
  data into the program, as long as the input value is valid. Depending upon
  cursor position update register calls one of its own procedures to convert
  the user input into a format understood by the tone detector, and writes the
  new value to the tone detector}
Var
   VALUE : Integer;

Procedure Change_control(CELL : Eightstring);
{ Writes the new value for the control register into the tone detector}
Begin
  Putreg(0,Binary_to_int(CELL));
End; {Change_control}

Procedure Change_mode(CELL : Eightstring);
{ Writes the new value for the mode register into the tone detector, and up-
  dates program variable GAIN_FACTOR used in other calculations}
Begin
   Putreg(1,Binary_to_int(CELL));
   Case CELL[7] of
      '0' : GAIN_FACTOR := 4;
      '1' : If CELL[8] = '0' then GAIN_FACTOR := 1
            Else GAIN_FACTOR := 16;
   End {Case}
End; {Change_mode}

Procedure Change_env_time_constant (CELL : Eightstring ; Var ERRFLAG :
Boolean);
{ Checks for valid range of new envelope time constant - if valid writes new
  value to tone detector, if not gives error message}
Var
   TEMP : Real;
Begin
   Zero_fill(CELL);
   Val(CELL,TEMP,CODE);
   If TEMP = 0 then
   Begin
      Error_message('Value out of range      ');
      ERRFLAG := True;
   End {If}
   Else
   Begin
      VALUE := Round(1024*(1-exp(-1/(8*TEMP))));
      If (VALUE > 255) or (VALUE < 1) then
      Begin
         Error_message('Value out of range      ');
         ERRFLAG := True;
      End {If}
      Else Putreg(2,VALUE);
   End; {Else}
End; {Change_env_time_constant}

Procedure Change_thresholds (CELL : Eightstring ; THRESHOLD_TYPE : Char ; Var
                             ERRFLAG : Boolean);
{ Checks for valid range of new threshold - if valid writes new value to tone
  detector, if not gives error message}
Var
   RNUM,TEMP : Integer;
Begin
   Zero_fill(CELL);
   Val(CELL,TEMP,CODE);
   VALUE := Round(254*(0.743*GAIN_FACTOR*TEMP)/1000);
   If VALUE > 255 then
   Begin
      Error_message('Value out of range      ');
      ERRFLAG := True;
   End {If}
   Else Case THRESHOLD_TYPE of
      'U' : RNUM := 3;
      'L' : RNUM := 4;
      'C' : RNUM := 7;
   End; {Case}
   Putreg(RNUM,VALUE);
End; {Change_thresholds}

Procedure Change_filter_length (CELL : Eightstring ; Var ERRFLAG : Boolean);
{ Checks for valid range of new filter length - if valid writes new value to
  tone detector, if not gives error message}
Var
   TEMP : Integer;
Begin
   Zero_fill(CELL);
   Val(CELL,TEMP,CODE);
   VALUE := Round((16384/(TEMP-1))-16);
   If (VALUE > 255) or (VALUE < 0) then
   Begin
      Error_message('Value out of range      ');
      ERRFLAG := True;
   End {If}
   Else Putreg(5,VALUE);
End; {Change_filter_length}

Procedure Change_passband_width (CELL : Eightstring ; Var ERRFLAG : Boolean);
{ Checks for valid range of new passband width - if valid writes new value to
  tone detector, if not gives error message}
Var
   TEMP : Integer;
Begin
   Zero_fill(CELL);
   Val(CELL,TEMP,CODE);
   VALUE := Round(0.128*TEMP);
   If VALUE > 63 then
   Begin
      Error_message('Value out of range      ');
      ERRFLAG := True;
   End {If}
   Else Putreg(6,VALUE);
End; {Change_passband_width}
```

```
Procedure Change_frequency (CELL : Eightstring ; BAND : Integer ; Var ERRFLAG
:
                            Boolean);
{ Checks for valid range of new freqency - if valid converts to the values to
  be programmed into the frequncy MS byte and LS byte registers and writes new
  values to tone detector, if not gives error message}
Var
   TEMP : Real;
Begin
   Zero_fill(CELL);
   Val(CELL,TEMP,CODE);
   If TEMP > 3400 then
   Begin
      Error_message('Value out of range      ');
      ERRFLAG := True;
   End {If}
   Else
   Begin
      VALUE := Round(8.192*TEMP) div 256;
      Putreg(8,VALUE);
      VALUE := Round(8.192*TEMP) mod 256;
      Putreg(8+BAND,VALUE);
   End; {Else}
End; {Change_frequency}

Procedure Change_filter_select (CELL : Eightstring);
{ Writes the new value for the fliter select register into the tone detector}
Begin
   VALUE := Binary_to_int(CELL);
   Putreg(15,VALUE);
End; {Change_filter_select}

Begin {Update_register}
   Case LROW of
      5      : Change_control(WRITE_TABLE[5]);
      6      : Change_mode(WRITE_TABLE[6]);
      7      : Change_env_time_constant(WRITE_TABLE[7],ERRFLAG);
      8      : Change_thresholds(WRITE_TABLE[8],'U',ERRFLAG);
      9      : Change_thresholds(WRITE_TABLE[9],'L',ERRFLAG);
      10     : Change_filter_length(WRITE_TABLE[10],ERRFLAG);
      11     : Change_passband_width(WRITE_TABLE[11],ERRFLAG);
      12     : Change_thresholds(WRITE_TABLE[12],'C',ERRFLAG);
      13..18 : Change_frequency(WRITE_TABLE[LROW],LROW-12,ERRFLAG);
      19     : Change_filter_select(WRITE_TABLE[19]);
   End; {Case}
End; {Update_register}

Procedure Update_read_value;
{ Update read value is called continuously as long as no keyboard input
  processing is pending. Update read value calls its own internal procedures
  in turn to convert values read from the tone detector into the display
  format}
Var
   I     : Integer;
```

```
Procedure Read_status;
{ Places cursor at start of status register display and outputs value read
  from tone detector}
Begin
   Gotoxy(BLOCK,5);
   Outbin(Getreg(0),8);
End; {Read_status}

Procedure Read_mode;
{ Places cursor at start of mode register display and outputs value read from
  tone detector}
Begin
   Gotoxy(BLOCK,6);
   Outbin(Getreg(1),8);
End; {Read_mode}

Procedure Read_DTMF;
{ Places cursor at start of DTMF digit display, cnverts and outputs value read
  from tone detector}
Var
   DTMF : Integer;
Begin
   Gotoxy(BLOCK,8);
   DTMF := Getreg(2);
   If DTMF > 127 then
   Begin
      DTMF := DTMF - 240;
      Textcolor(14);
      Write('OVRUN  ');
      Textcolor(0);
   End {If}
   Else
   Begin
      DTMF := DTMF - 112;
      Textcolor(0);
      Write('       ');
   End; {Else}
   Textcolor(14);
   Case DTMF of
      0..2  : Write(DTMF+1);
      3     : Write('A');
      4..6  : Write(DTMF);
      7     : Write('B');
      8..10 : Write(DTMF-1);
      11    : Write('C');
      12    : Write('*');
      13    : Write('0');
      14    : Write('#');
      15    : Write('D');
   End; {Case}
   Textcolor(0);
End; {Read_DTMF}
```

```
Procedure Read_Time (TIME : Char);
{ Places cursor at start of time display, reads and outputs present value of
  current time, departure time and arrival time from tone detector}
Var
   RESULT : Real;
Begin
   Case TIME of
     'A' : Begin
             Gotoxy(BLOCK+3,9);
             Result := Getreg(3) * 256 + getreg(4);
           End;
     'D' : Begin
             Gotoxy(BLOCK+3,10);
             Result := Getreg(5) * 256 + Getreg(6);
           End;
     'C' : Begin
             Gotoxy(BLOCK+3,11);
             Result := Getreg(7) * 256 + Getreg(8);
           End;
   End; {Case}
   If RESULT < 0 then RESULT := RESULT + 65536.0;
   Textcolor(14);
   Write(RESULT:5:0);
   Textcolor(0);
End; {Read_time}

Procedure Read_level (BAND : Integer);
{ Places cursor at start of signal level display for the required freqency
  band and outputs value read from tone detector. This procedure is called six
  times in succession with BAND incrementing from 1 to 6}
Var
   RESULT : Real;
   TEMP   : Real;
Begin
   Gotoxy(BLOCK+4,12+BAND);
   RESULT := Getreg(8+BAND);
   TEMP := (RESULT*5.3)/GAIN_FACTOR;
   If (TEMP < 4.5) then TEMP := 0;
   Textcolor(14);
   Write(TEMP:4:0);
   Textcolor(0);
End; {Read_level}

Begin {Update_read_value}
   BLOCK := READBLOCK;
   Textcolor(0);
   Read_status;
   Read_mode;
   Read_DTMF;
   Read_time('A');
   Read_time('D');
   Read_time('C');
   For I := 1 to 7 do Read_level(I);
   BLOCK := WRITEBLOCK;
   Textcolor(14);
End; {Update_read_value}

Procedure Initialise_tone_detector;
{ Sets up screen display, initial values for all tone detector registers and
  loads those registers}
Begin
   GAIN_FACTOR := 4;
   ROW := 5;
   COL := FIRSTCOL[ROW];
   Textcolor(11);
   Gotoxy(6,3);  Write('Write Registers');
   Textcolor(10);
   Gotoxy(1,5);  Write('CONTROL Register       '); Write('       b');
   Gotoxy(1,6);  Write('MODE   Register        '); Write('       b');
   Gotoxy(1,7);  Write('Envelope time constant '); Write('       mS');
   Gotoxy(1,8);  Write('Upper threshold        '); Write('       mV');
   Gotoxy(1,9);  Write('Lower threshold        '); Write('       mV');
   Gotoxy(1,10); Write('Filter length          '); Write('       samples');
   Gotoxy(1,11); Write('Passband width         '); Write('       Hz');
   Gotoxy(1,12); Write('Change threshold       '); Write('       mV');
   Gotoxy(1,13); Write('Band 1 frequency       '); Write('       Hz');
   Gotoxy(1,14); Write('Band 2 frequency       '); Write('       Hz');
   Gotoxy(1,15); Write('Band 3 frequency       '); Write('       Hz');
   Gotoxy(1,16); Write('Band 4 frequency       '); Write('       Hz');
   Gotoxy(1,17); Write('Band 5 frequency       '); Write('       Hz');
   Gotoxy(1,18); Write('Band 6 frequency       '); Write('       Hz');
   Gotoxy(1,19); Write('Filter select          '); Write('       b');
   Textcolor(11);
   Gotoxy(50,3);  Write('Read Registers');
   Textcolor(10);
   Gotoxy(45,5);  Write('STATUS Register       '); Write('       b');
   Gotoxy(45,6);  Write('MODE   Register       '); Write('       b');
   Gotoxy(45,8);  Write('DTMF Digit is         '); Write(' ');
   Gotoxy(45,9);  Write('Tone arrival time     '); Write('       mS');
   Gotoxy(45,10); Write('Tone departure time   '); Write('       mS');
   Gotoxy(45,11); Write('Current time is       '); Write('       mS');
   Gotoxy(45,13); Write('Band 1 signal level   '); Write('       mV');
   Gotoxy(45,14); Write('Band 2 signal level   '); Write('       mV');
   Gotoxy(45,15); Write('Band 3 signal level   '); Write('       mV');
   Gotoxy(45,16); Write('Band 4 signal level   '); Write('       mV');
   Gotoxy(45,17); Write('Band 5 signal level   '); Write('       mV');
   Gotoxy(45,18); Write('Band 6 signal level   '); Write('       mV');
   Gotoxy(45,19); Write('Total signal level    '); Write('       mV');
   Textcolor(14);
   Update_read_value;
   WRITE_TABLE[5] := '00000000';
   WRITE_TABLE[6] := '00100000';
   WRITE_TABLE[7] := '    1.00';
   WRITE_TABLE[8] := '     100';
   WRITE_TABLE[9] := '      50';
   WRITE_TABLE[10] := '     250';
   WRITE_TABLE[11] := '     400';
   WRITE_TABLE[12] := '     100';
```

```
    WRITE_TABLE[13] := '    500';
    WRITE_TABLE[14] := '   1000';
    WRITE_TABLE[15] := '   1500';
    WRITE_TABLE[16] := '   2000';
    WRITE_TABLE[17] := '   2500';
    WRITE_TABLE[18] := '   3000';
    WRITE_TABLE[19] := '00111111';
    For I := 5 to 19 do begin
       Gotoxy(BLOCK+1,I);
       Write(WRITE_TABLE[I]);
       Update_register(I,ERRFLAG);
    End; {For}
    ROW := 5;
    Gotoxy(BLOCK+1,ROW);
    TEMP_CELL := WRITE_TABLE[ROW];
    Soft_cursor(1);
End; {Initialise_tone_detector}

Procedure Cursor_up;
{ Moves current cursor location and soft cursor up on screen. Vertical
  displacement depends upon current location, and is given by array DISP}
Begin
    If not ERRFLAG then
    Begin
       Gotoxy(BLOCK+1,ROW);
       Write(WRITE_TABLE[ROW]);
       ROW := ROW - DISP[ROW+24];
       TEMP_CELL := WRITE_TABLE[ROW];
       COL := FIRSTCOL[ROW];
       Gotoxy(BLOCK+COL,ROW);
       Soft_cursor(1);
    End {If}
End; {Cursor_up}

Procedure Cursor_down;
{ As for cursor up, but moves down}
Begin
    If not ERRFLAG then
    Begin
       Gotoxy(BLOCK+1,ROW);
       Write(WRITE_TABLE[ROW]);
       ROW := ROW + DISP[ROW+26];
       COL := FIRSTCOL[ROW];
       TEMP_CELL := WRITE_TABLE[ROW];
       Gotoxy(BLOCK+COL,ROW);
       Soft_cursor(1);
    End; {If}
End; {Cursor_down}

Procedure Cursor_left;
{ Moves current cursor location and soft cursor to left on screen. Cursor
  remains within value window for each parameter, given by FIRSTCOL array}
Begin
    If COL > FIRSTCOL[ROW] then
    Begin
       Gotoxy(BLOCK+COL,ROW);
       Soft_cursor(0);
       COL := COL - 1;
       Gotoxy(BLOCK+COL,ROW);
       Soft_cursor(1);
    End; {If}
End; {Cursor_left}

Procedure Cursor_right;
{ Moves current cursor location and soft cursor to right on screen. Cursor
  remains wthin value window (column 8)}
Begin
    If COL < 8 then
    Begin
       Gotoxy(BLOCK+COL,ROW);
       Soft_cursor(0);
       COL := COL + 1;
       Gotoxy(BLOCK+COL,ROW);
       Soft_cursor(1);
    End; {If}
End; {Cursor_right}

Procedure Data_entry;
{ This procedure takes a keyboard numeric entry into the temporary string, and
  moves the soft cursor to the right if necessary}
Begin
    Gotoxy(BLOCK+COL,ROW);
    Soft_cursor(0);
    Write(CMD);
    TEMP_CELL[COL] := CMD;
    If COL < 8 then COL := COL +1;
    Gotoxy(BLOCK+COL,ROW);
    Soft_cursor(1);
End; {Data_entry}

Procedure Newline;
{ This procedure terminates data entry for a particular value, inputs the
  value to the cell check procedure, moves the soft cursor down, and loads the
  temporary string with the present data of the new parameter}
Begin
    Gotoxy(BLOCK+COL,ROW);
    Soft_cursor(0);
    Cell_check(TEMP_CELL,ROW,COL,ERRFLAG);
    If not ERRFLAG then
    Begin
       WRITE_TABLE[ROW] := TEMP_CELL;
       Update_register(ROW,ERRFLAG);
       If ERRFLAG then COL := FIRSTCOL[ROW]
       Else
       Begin
          ROW := ROW + DISP[ROW+26];
          COL := FIRSTCOL[ROW];
          TEMP_CELL := WRITE_TABLE[ROW];
```

```
        End; {Else}
    End; {If}
    Gotoxy(BLOCK+COL,ROW);
    Soft_cursor(1);
End; {Newline}

Begin
    Rewrite(OUTPUT);
    Reset(INPUT);
    Clrscr;
    Textcolor(15);
    CMD := ' ';
    Gotoxy(20,1);
    Writeln('TONE DETECTION DEMONSTRATION PROGRAM');
    Initialise_tone_detector;
    Repeat
        If KeyPressed then
        Begin
            Read(kbd,CMD);
            If (CMD = #27) and KeyPressed then
            Begin
                Read(kbd,CMD);
                Case CMD of
                    #72 : Cursor_up;
                    #75 : Cursor_left;
                    #77 : Cursor_right;
                    #80 : Cursor_down;
                End; {Case}
            End; {If}
            Case CMD of
                '0'..'9','A'..'F','a'..'f',' ' : Data_entry;
                '.'     : If ROW = 7 then Data_entry;
                #13     : Newline;
            End; {Case}
        End {If}
    Else Update_read_value;
    Until (CMD = 'Q') or  (CMD = 'q');
    Clrscr;
End.
```

# Appendix C

# Power Detector Operational Considerations

## C. 1 Arrival and Departure Time Skew

The use of a large time constant $\tau$ for the envelope decay factor results in a greater time delay between the appearance or departure of a tone and the envelope detector output crossing a threshold. See the section "Envelope Decay Factor" for details of the envelope decay detector.

If the signal level on the line changes to S at time t= 0 when the output of the envelope detector is EI, we may determine the time t taken for the envelope detector output to reach a pre-defined threshold level L as follows:

$$t = -\tau \times \ln[(S-L) / (S-EI)]$$

For example, at time t = 0, the output of the envelope detector EI = 0 and the upper (arrival) threshold level is set to −20 dBm, a signal appears on the line at a level of −10 dBm.

Therefore, when $t = 0$, $EI = 0$

$$S = 10^{(-10 / 20)} = 0.3162$$

$$L = 10^{(-20 / 20)} = 0.1$$

This gives a value of $t = 0.38\tau$ for the envelope detector output to cross the arrival threshold.

In the case for tone departure at time t = 0, the output of the envelope detector is stable at −10 dBm, and the lower (departure) threshold is set to −20 dBm, the signal on the line departs.

Therefore, when $t = 0$, $S = 0$, $L = 0.1$ as before

$$EI = 10^{(-10 / 20)} = 0.3162$$

This gives a value of $t = 1.15\tau$ for the envelope detector output to cross the departure threshold.

The skew between the delays for the envelope detector to cross the arrival and departure thresholds is thus $0.77\tau$ in this instance. This skew obviously increases as the envelope factor $\tau$ is increased.

## C.2 Sampling Frequency Considerations

Due to the envelope detector assessing the level of the signal on the line by rectification and smoothing, anomalies arise in its behavior when the signal being rectified is a pure tone at a frequency which is an integer sub-multiple of the sampling frequency of 8 kHz. This effect is most significant at 2 kHz, where the signal level assessed by the envelope detector may vary over the range of +0.91 dB to −2.1 dB relative to the true signal level. If the output signal differs from 2 kHz by a very small amount, the envelope detector output will vary over the above range as the phase of the signal slides past the sampling instants. Ensuring a hysteresis band between the upper and lower thresholds of at least 3 dB will avoid the possibility of a series of apparent tone arrivals and departures in the presence of a steady pure tone.

# Part V. Control

# Implementation of PID and Deadbeat Controllers with the TMS320 Family

**Irfan Ahmed**

**Digital Signal Processor Products — Semiconductor Group**
**Texas Instruments**

# Introduction

Control systems are a necessary part of modern manufacturing, industrial processes, and our daily lives. They range from simple controls like those on our air conditioning to more intricate controls like those for a missile guidance system. Control mechanisms have evolved from mechanical, pneumatic, and electromechanical systems to electronic control systems. Electronic controls have been implemented with analog components like resistors, capacitors and op-amps (operational amplifiers). However, with the availability of microprocessors, control systems are being implemented in digital form. The use of microprocessors in digital control systems has created not only some new opportunities due to the powerful processing capabilities of microprocessors, but also a need for a new body of knowledge that utilizes some of these processing capabilities.

This report discusses implementation of PID (proportional integral derivative) and deadbeat digital controllers with the TMS320 family of digital signal processors. These digital signal processors are application-specific processors designed to process signals, including control signals, very efficiently. In numerically intensive applications, digital signal processors are at least an order of magnitude higher in performance than conventional processors and minimize the numerical problems of processing signals digitally.

This report is arranged in the following order:

- *Control Systems* – Provides an introduction to digital controllers and discusses selection of processors for a digital controller.
- *Design of Digital Control Systems* – Discusses the design of digital controllers.
- *Implementing Digital Controllers* – Discusses implementation of digital controllers.
- *Applications* – Describes applications of digital controllers using the TMS320 digital signal processors.
- *Appendices A* through *C* – Lists the mathematical procedures needed to design the controllers.
- *Appendix D* – Lists the PC-Matlab programs used for simulation of these controllers.
- *Appendix E* – Lists the TMS320C15 assembly code required to implement the controller algorithms.

## Control Systems

A control system is a system that commands or regulates a process in order to achieve a desired output from the process. A control system consists of three main components:

- Sensors
- Actuators
- A controller

Sensors measure the behavior of the system or the process and provide feedback information. Typical sensors used in control systems are resolvers, shaft encoders, and potentiometers that provide position information; tachometers that provide speed information, and current sensors that provide current information. Actuators supply the driving and corrective forces to achieve a desired output. Typical actuators are DC and AC motors in electromechnical systems, and valves in hydraulic or pneumatic systems.

The controller generates actuator commands in response to the commands received from the system controller and in response to feedback information provided by the sensors. The controller consists of some computation elements that process the command and feedback signals to achieve a desired response from the entire system. The function of the controller is to ensure that the actuator responds to the commands as quickly as possible and, at the same time, that the system remains stable under all operating conditions. Typically, a controller will modify the frequency response of the system. The computational elements of the controller can be implemented with analog or digital components.

### Analog Controllers

Control systems have traditionally been implemented using analog components like operational amplifiers, resistors, and capacitors. The combination of these elements implements filter-like structures that modify the frequency response of the system. Although more powerful analog processing elements like multipliers are available, they are generally not used because of their high cost. In spite of their simpler processing elements, analog controllers can be used to implement high-performance systems.

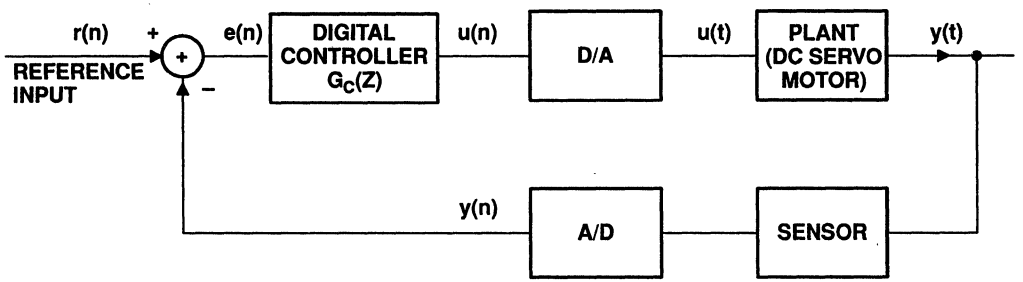
### Digital Control Systems

With the performance and reliability inherent to microprocessors and microcontrollers, digital controllers are taking over many applications from analog controllers. In a digital control system, the controller is implemented with a microprocessor or a microcontroller, which is responsible for processing the signal. However, the actuator commands from the controller are digital and may have to be converted into analog signals by a D/A (digital-to-analog) converter. Similarly, the measurements from the sensor may be analog and will have to be converted into a digital signal by an A/D (analog-to-digital) converter.

Figure 1 shows the block diagram of a digital control system. The D/A converter converts the digital output of the microcomputer, u(n), into an analog signal, u(t). The output, u(t), of the D/A needs power amplification and drives the motor to the desired or reference state, r(n). The out-

put of the motor, y(t), is measured by a sensor and converted into a digital signal, y(n), by the A/D. The feedback signal is subtracted from the reference signal r(n) to create an error signal e(n). The error signal, e(n), is used by the controller to issue the corresponding control action u(n).

**Figure 1. Digital Control System**




## Analog Versus Digital Controllers

Tradeoffs have to be made in the selection of controllers for a system. Analog controllers provide continous processing of the signal and can be used for very high bandwidth systems. They also give almost infinite resolution of the signal they are measuring, thus providing precise control. Analog controllers have been around for a long time; their behavior is well understood, and this makes them easy to design. They can be implemented with relatively inexpensive components and therefore are sometimes cheaper.

On the negative side, analog controllers suffer from component aging and temperature drifts. Thus, a perfectly designed controller will start to exhibit undesired characteristics after a while. Analog controllers are hardwired solutions; this makes modifications or upgrades in the design difficult. Analog controllers are also limited to simpler algorithms from classical control theory like PID and compensation techniques.

Digital controllers sample the signal at discrete time intervals. This limits the bandwidth (bandwidth is 1/6 to 1/10 sampling rate) that can be handled by the controller. The processing of the signal takes a finite amount of time, adding to phase delay in the system. In addition, the resolution of the signal is limited by the resolution or wordlength of the processor. Digital controllers also require additional components like A/D and D/A; although newer processors include these components on the same chip. Digital controllers are relatively new, and their behavior is not very well understood, thus making design of digital controllers relatively difficult in comparison to analog controllers.

However, digital controllers have some advantages also. They are not affected by component aging and temperature drift, and they provide stable performance. When the design is done in the z-domain, the behavior of digital controllers can be more precisely controlled. They can also be used to implement more sophisticated techniques from modern control theory, such as state controllers, optimal control, and adaptive control. Digital controllers are programmable, thus making them easy to upgrade and maintaining design investment. They can also be time shared to implement different functions in the system, like notch filters and system control, thus reducing system

cost. If digital controllers are designed properly, their advantages outweigh their disadvantages. Table 1 compares analog and digital controllers.

**Table 1. Comparison of Analog versus Digital Controllers**

| Controller | Advantages | Disadvantages |
|---|---|---|
| Analog | High bandwidth<br>High resolution<br>Ease of design | Component aging<br>Temperature drift<br>Hardwired design<br>Good only for simpler designs |
| Digital | Programmable solution<br>Insensitive to environment<br>Shows precise behavior<br>Implements advanced algorithms<br>Capable of additional functions | Creates numerical problems<br>Must use high-performance processor<br>Difficult to design |

## Processor Selection Issues

The choice of processor is critical in determing the performance and behavior of the digital controller. The usual choices are microcontrollers, general purpose microprocessors, and digital signal processors (DSP). RISC processors and bit-slice processors can also be used; however it is not practical to use them in most cases because of their cost.

Digital controllers monitor signals at discrete time intervals or finite sampling rates. If the signal is not sampled fast enough, some of the information may be lost. The processing of the signal takes a finite amount of time. The processing has to be completed before the arrival of the next sample, or preferably as soon as possible. Too much delay in the output can cause loss of information or excessive phase delay in the system, leading to instability. These conditions impose certain minimum performance requirements on the processor. Most of the processors currently used to implement controllers are usually not fast enough to process signals in real time; they rely upon lookup tables with precomputed results.

Digital controllers use discrete steps to represent a signal, which is limited to the wordlength of the processor. Coefficients or gain constants also have to be represented in the limited wordlength. This discretization or loss of resolution is referred to as quantization error. In addition, results of mathematical operations have to fit in a limited wordlength and may lose part of the result due to this limitation. This is referred to as truncation error. Both of these errors cause oscillations or limit cycles and can lead to instability.

Another problem that occurs in digital controllers is overflow of registers. Successive mathematical operations can cause registers to overflow. Registers in most processors wrap around, causing the result of a calculation to go from most positive to least negative, in turn causing the output to reverse directions. Most of these problems occur in microcontrollers and microprocessors because their architectures are not designed for signal processing.

Early microprocessors (μP) and microcontrollers (μC) were designed to replace hardwired logic, TTL gates, etc. Newer microprocessors and microcontrollers have retained most of the old

architectures. These processors are adequate for simple applications that require little or no signal processing. In a control system, the controller is responsible for processing the command and feedback signal. Thus, applications such as control systems, speech, and telecommunications require intensive numeric processing of analog signals. μPs and μCs are usually unable to do the processing correctly, so using them can cause significant numerical problems.

Most of these problems occur in processors that have 8/16-bit ALUs and registers. This 8/16-bit architecture limits the accuracy of intermediate and final results and generates truncation/quantization errors. Lack of scaling shifters to maintain the required significant bits can cause additional quantization/truncation errors. Most processors also lack the performance to perform real time processing, so they rely upon lookup tables, thus limiting precision to low-performance or low-bandwidth systems. Lack of processing capability also limits these processors to simpler control techniques. They are unable to take advantage of sophisticated techniques from modern control theory. If used in higher-performance systems, they can cause excessive loop delays, leading to instability. Most of the problems discussed can be eliminated with the use of digital signal processors as controllers.

## DSP Architectures

DSP architectures like those in the TMS320 family have been designed for signal processing systems. The TMS320 family not only has an architecture that minimizes numerical problems in signal processing, but also has the performance to meet the bandwidth requirements of high performance systems using sophisticated techniques.

DSP architectures are optimized to give the highest possible performance. To achieve high processing speed, the TMS320 DSPs perform all functions in internally hardwired logic. Thus, it takes a single clock cycle to execute most functions. Other processors perform the same functions in software or microcode, thus taking a large number of cycles for execution. To enhance the performance even further, the TMS320 architecture employs a multiple-bus internal architecture. This allows simultaneous fetch of instructions and data operands. Most instructions on the TMS320, including arithmetic operations, are executed in a single clock cycle.

In digital signal processing, most algorithms, including control algorithms, can be represented as difference equations consisting of multiply accumulates. The TMS320 DSPs contain a hardware multiplier that performs a $16 \times 16$ multiply in a single instruction cycle. This high speed allows the TMS320 to execute most control algorithms in real time. The fast processing speed minimizes the computation delay time for generating the output from the controller and also allows very fast sampling rates to be implemented for high bandwidth systems. Additional features in the TMS320 architecture include an instruction set that is optimized for data sampled systems. The DMOV instruction implements the $z^{-1}$ operator. The MACD instruction implements four operations simultaneously:

- A multiply
- Data move
- Accumulate previous result
- Load T register

For greater precision, the TMS320 has 32-bit registers for storing intermediate results. In addition, the TMS320 has multiple hardware parallel shifters to allow scaling and prevent overflows. These shifters enable shifting to take place simultaneously with other operations without additional overhead or execution time. The 32-bit registers and shifters minimize quantization and truncation errors because a very high precision can be maintained both for intermediate and final results. The TMS320 also contains an overflow mode, which, in case of overflow, allows the accumulator to saturate at most positive or least negative values (similar to analog circuits), instead of rolling over and varying between positive and negative values. For fast context saves, the TMS320 contains an on-chip hardware stack, reducing interrupt response time and minimizing stack pointer manipulations. Since the TMS320 is a family of microcontrollers, it also minimizes system costs with features such as on-chip RAM, on-chip ROM/EPROM, and on-chip peripherals like serial ports, timers, and parallel I/O. The high degree of on-chip functionality, flexible instruction set, pipelined architecture, and high performance make the TMS320 the preferred choice in most control and signal processing applications. Table 2 lists a comparison between the TMS320C14, TMS320C25, and several μCs and μPs.

**Table 2. Features Comparison**

| Feature | '320C14 | '320C25 | 80C196 | 68000 | 68020 |
|---|---|---|---|---|---|
| Instruction cycle time-ns | 160 | 100 | 333 | 400 | 120 |
| Frequency – Mhz | 25 | 40 | 12 | 10 | 24 |
| Multiply 16×16 → 32 (μs) | 0.16 | 0.1 | 2.2 | 7.0 | 1.0 |
| PID loop ( μs) | 2.2 | 1.3 | 27.0 | 25.0 | 4.8 |
| Matrix multiply (3×3, 1×3 — μs) | 4.3 | 2.7 | 24.3 | 65.2 | 9.7 |

### *TMS320 Digital Signal Processors*

The TMS320 digital signals processors can be divided into two families: a fixed-point arithmetic family and a floating-point arithmetic family. Each family is further divided into different generations and offers different performance ranges between generations. Within each generation, members are object code and, in some cases, pin compatible.

### *TMS320 Family of Fixed-Point Arithmetic DSPs*

The fixed-point arithmetic family is made up of three generations, TMS320C1x, TMS320C2x, and TMS320C5x. All members of the fixed-point arithmetic family have a 16-bit architecture with 32-bit ALU and accumulator. They are based on a Harvard architecture with separate buses for program and data, allowing both instructions and operands to be fetched simultaneously. They also feature a 16×16=32 hardware multiplier for a single-cycle multiply, and a hardware stack for fast context saves. An overflow saturation mode is included to prevent wraparound. All instructions (except branches) are executed in a single clock cycle. Performance ranges from 5 MIPS (million of instructions per second) to 28.5 MIPS among the three generations.

### *TMS320C1x*

The TMS320C1x generation is based on the TMS32010, the first DSP, introduced in 1982. It comes with 144 words of on-chip RAM and 4K-words of address space. Instruction cycle time goes down to 160 ns. Other members of the first generation include the TMS320C15 and its

EPROM version (TMS320E15), TMS320C17/E17, and TMS320C14/E14. All of these devices have an expanded memory of 256 words of on-chip RAM and 4K-words of on-chip ROM/EPROM. The TMS320C14/E14 is optimized for digital control applications and has 16 pins of bit I/O, four timers (including a watchdog timer), a USART, 6/4 channels of pulse width modulation (PWM), and 2/4 capture inputs for optical encoder interface and PWM[1,2].

*TMS320C2x*

The TMS320C2x generation is based on the TMS320C25. It comes with 544 words of on-chip RAM and 4K-words of on-chip ROM. Total address space is expanded to 64K words for both data and program. The instruction set is considerably enhanced from the C1x generation. Instruction cycle time is reduced to 80 ns. Other members include the TMS320E25, TMS32020, and TMS320C26[3].

*TMS320C5x*

The TMS320C5x generation is based on the TMS320C50. It features 8K-words of on-chip RAM and 2K-words of on-chip ROM. The instruction set is considerably enhanced from the TMS320C2x generation. Some of the new features include a separate PLU, shadow registers for fast context save, JTAG serial scan emulation, and software wait states. Instruction cycle time is 35 ns.

***TMS320 Family of Floating-Point Arithmetic DSPs***

The floating-point arithmetic DSPs consist of two generations: the TMS320C3x and the TMS320C4x. All members of the floating-point arithmetic family have a 32-bit architecture with 40-bit extended precision registers. The floating-point arithmetic family is based on Von Neuman architecture. However, multiple buses are included to give even faster throughput than traditional Harvard architectures. Features include hardware floating-point multiplier and a floating-point ALU.

*TMS320C3x*

The TMS320C3x family is based on the TMS320C30. It features 2K × 32 words of on-chip RAM, 4K × 32 of on-chip ROM, and 64 words instruction cache. Other features include a separate DMA, two serial ports and two timers. The TMS320C30 features two external 32-bit data buses and a 16M-word address space. Instruction cycle time is 60 ns, and performance is up to 33 MFLOPS (million floating-point operations/second)[4].

## Design of Digital Control Systems

Design of a control system involves two major steps:

1) The process or plant has to be put into mathematical form so that its behavior can be analyzed and evaluated (i.e., a plant model has to be derived).
2) An appropriate controller must be designed so that the plant gives the desired response under influence of the control system.

Designing a digital control system requires additional steps that convert the system into a discrete form. This conversion can be done in two different ways.

1) The design of the controller can be carried out entirely in the analog form in s-domain, and then converted into discrete form at the final stage for implementation.
2) The design of the controller may also be carried out entirely in discrete form. In this case, the plant model has to converted into discrete form or z-domain.

This section describes how to

- Discretize analog systems
- Reduce a plant into a mathematical form
- Design the controller.

## Discretization of Analog Systems

There are several ways to convert existing continuous or analog control systems into discrete or digital systems. However, the conversion from the s-domain to the z-domain causes some distortion in the location of the poles in the z-domain, and must therefore be taken into account.

### *Zero-Order Hold (ZOH)*

This technique assumes that the controller is preceded by a ZOH (D/A converter) and followed by a sampler (A/D converter), so that both input and output of the resulting system are digital. Both the ZOH and sampler are included in the conversion scheme. The conversion is given by the following equation:

$$H(z) = (1 - z^{-1})Z\,[L^{-1}(H(s)/s)] \tag{1}$$

It is assumed that the Laplace transform will be split up using partial fractions and z-transform tables will be used.

### *Matched Pole-Zero*

In this technique, the poles of the s-domain are directly mapped into the z-domain according to the relationship $z = e^{Ts}$, where T is the sampling period. To equal the number of poles and zeros, additional zeros are added at $z = -1$. The gain of the two systems is matched at a critical frequency by choosing an arbitrary gain constant. This method does not take into consideration any aliasing effects.

### *Bilinear Transformation*

This technique, also called the Tustin or trapezoidal approximation, uses the relationship

$$s = \frac{2}{T}(z-1)/(z+1) \tag{2}$$

to transform an s-domain function into the z-domain. The left half of the s-plane is mapped into the unit circle in the z-plane. However, this method warps the frequency response at the critical frequencies of the system. To overcome this problem, the critical frequencies of the original s-domain are prewarped so that the critical frequencies of the z-domain system end up where they belong. The critical frequency $W_0$ is prewarped to another frequency by the transformation

$$W^* = \frac{2}{T}\tan\left(W_o\frac{T}{2}\right) \tag{3}$$

where T is the sampling period.

Before these techniques can be used, an appropriate plant transfer function must be derived, or a controller may have to be designed. The next section describes derivation of a plant transfer function and the controller design.

## Plant Modeling

The first part of designing any control system is to describe the plant in a mathematical form or identify the plant's parameters. This section describes the derivation of a mathematical model for a plant.

A DC servo motor is used in the example, and a model is developed for the motor. The motor is an analog device, and the given electrical and mechanical characteristics describe its behavior in the continuous time form. This model must be transferred into a discrete form or the z-domain for use with a digital controller. The ZOH method is used to transform the model into a discrete form.

In general, the electrical characteristics of a DC motor are given by

$$L\frac{di}{dt} + Ri = V - emf \tag{4}$$

where

$L$ = inductance of motor
$R$ = resistance
$V$ = applied voltage
$i$ = current
$emf$ = back emf = $K_e \times \theta$

The mechanical characteristics are given by

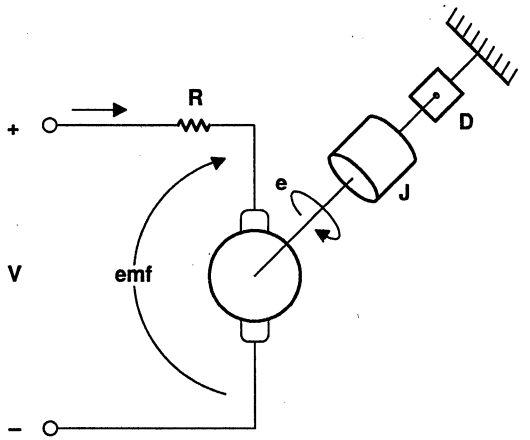
$$J_M\frac{d^2\theta}{d_t^2} + B\frac{d\theta}{d_t} + K\theta = T_L - J_L\frac{d^2\theta}{dt^2} \tag{5}$$

where

$J_M$ = motor inertia
$\theta$ = angular displacement
$K$ = stiffness constant
$B$ = damping constant
$J_L$ = load inertia
$T_L$ = load torque = $K_t \times i$
$K_t$ = torque constant

Figure 2 shows a representation of the equivalent electrical and mechanical model of a DC servo motor.

**Figure 2. Model of DC Motor**




The motor selected for this report is a Pittman model 9412G316. It has the following parameters:

$R$ = 6.4 Ω
$J$ = $1.54 \times 10^{-6}$ kg × m$^2$
$K_t$ = 0.0207 (N × m)/A
$K_e$ = 0.0206 volt/(rad/s)

The electrical and mechanical characteristics of the motor given by (4) and (5) can be combined. After the values of the motor parameters are substituted, the complete transfer function for the mathematical model of the motor as derived in Appendix A can be stated as

$$G_m(s) = \frac{53.906}{s\ (s + 1.116)} \tag{6}$$

This transfer function is then transformed into the z-domain using the zero-hold approximation as shown in Appendix A. The transfer function or the model in the discrete form is stated as

$$G_m(z) = \frac{.2694z^{-1} + .2693z^{-2}}{1 - 1.999z^{-1} + .999z^{-2}} \times K_m \tag{7}$$

where

$K_m$ = the numerator gain constant.

## Digital Controller Design

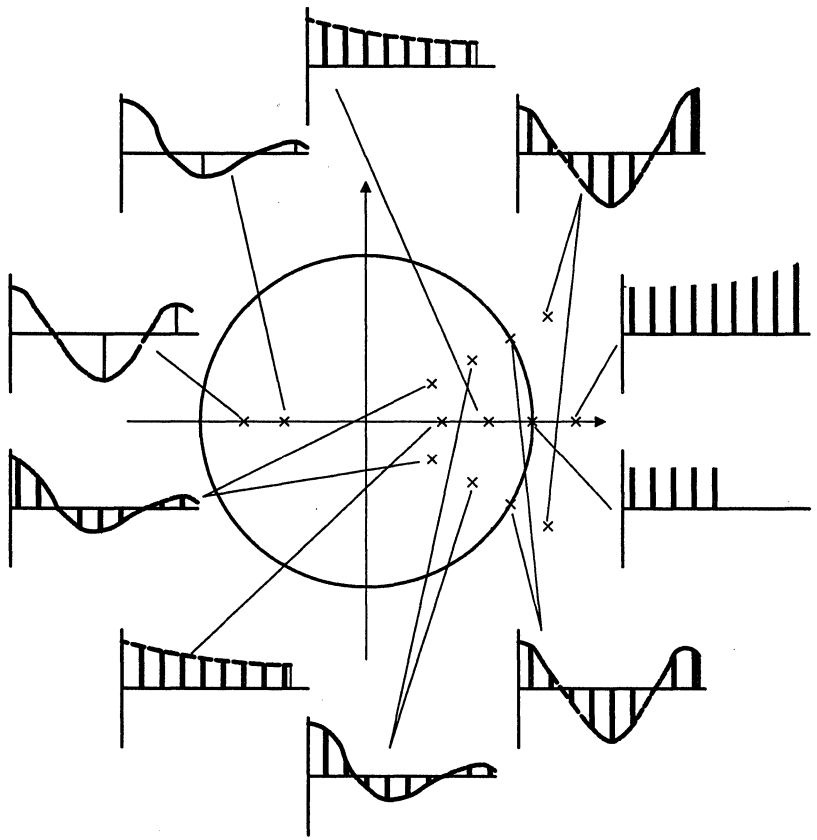
The next step in designing a digital control system is to design the controller. The controller may be designed in the continuous domain and then converted into discrete form. Alternatively,

the entire design may be carried out in the discrete domain. It is assumed that the design is carried out in the discrete domain. The next few sections give an overview of different types of control algorithms and discuss design of a PID controller and a deadbeat controller. However, before design of a digital controller is done, some discussion of behavior of poles in the z-domain is essential. The next section discusses behavior of poles in the z-domain.

### *Behavior of Poles in Z-domain*

Use of the conversion techniques allows the conversion of an existing analog design into a digital design. However, to insure successful implementation of the control system design, some knowledge of the behavior of the poles in the z-domain is essential. Any poles (real or imaginary) located outside the unit circle are unstable and have an unbounded response. Poles located inside the unit circle give a stable response. Poles that lie on the unit circle manifest oscillatory behavior. As poles move towards the origin, their response decays at a faster rate. Figure 3 shows the different pole locations and their corresponding responses.

**Figure 3. Behavior of Poles in Z-Domain**



### *Control Algorithms*

The next step in designing the controller is to select an appropriate algorithm or controller structure. The processing burden imposed upon the controller is directly dependent upon the complexity and type of controller structure.

#### *Compensation Techniques*

Compensation techniques are one of the most commonly used control techniques. In this technique, the controller adds poles and zeros to get a desired system response. For a continuous control system, the controller is designed in the s-domain by using some of the well known methods such as root locus, Bode plots, and Nyquist plots. The analog or s-domain design is then transferred into a discrete form (z- domain) by using a transformation technique. Alternatively, the compensator can be designed directly in the z-domain by using z-domain frequency response methods or the z-domain root locus method. Compensation techniques allow a precise modification to system behavior. Implementation of compensation techniques is described in an application report, Control System Compensation and Implementation with the TMS32010[5].

#### *PID*

The PID is a commonly used analog control technique. In a PID controller, terms proportional to the error term, its integral, and its derivative are summed to achieve the controller output. A PID controller may be designed in the s-domain, and then transferred into the z-domain by using one of the transformation methods. Alternatively, the PID algorithm is converted into a discrete form, and the design is carried out entirely in the z-domain. PID is probably the most commonly used algorithm. PID controllers are very robust; although the design of coefficients is somewhat arbitrary[6, 7].

#### *Deadbeat*

A deadbeat algorithm is used when a quick settling time is required. Deadbeat design is carried out entirely in the z-domain. A deadbeat controller replaces the poles of the system with poles at the origin of z-domain[8].

#### *State Models*

In a state model, a complete representation of the system is made in matrix form. This is accomplished by identifying and developing the relationship between the different states or variables of the plant. An appropriate feedback gain can be chosen to place the poles of the system at any desired location in the z-domain. State controllers are used where multiple variables or states need to be controlled. State controllers are sometimes impractical to implement because it may not be possible to measure all states. They are usually used in conjunction with observers. State controllers allow very precise control of system behavior[6, 9, 10].

#### *Observer Models*

Often, in control systems, some of the states of the system are not available for measurement. The measurement of known states in an observer model can be used to estimate unknown states in the control system. The estimated states, along with an appropriate feedback gain, can be used

to complete the control loop and place the poles at any desired location. Observers are typically used in conjunction with state controllers, where access to all state variables may not be available[6, 9, 10, 11].

*Optimal Control*

Optimal control synthesis is used when a specific performance or cost criteria (e.g., time or energy) must be minimized. The given criteria or cost function is used to derive an appropriate control law, which is then implemented with a controller or compensator[7, 12].

*Kalman Filter*

An observer model can be used in a system where an exact measurement of some states is available. However, in stochastic systems, the presence of noise or uncertainty makes it impossible to make an exact measurement. A Kalman filter is an observer model in a noisy or stochastic system[7, 13, 14].

*Adaptive Control*

Adaptive control is used in systems in which there is insufficient information about the plant parameters, making it impossible to derive a plant model. It is also used in systems where plant parameters or plant models change over time, making the controller unstable. An adaptive controller tracks changes in the plant by redesigning the controller to give an optimum control system[6, 8, 16].

## Design and Implementation of PID Controllers

This section describes design and implementation of a PID controller. PID is a commonly used technique in classical control. In designing controllers, it is often found that just minimizing a term proportional to the error is not sufficient. Including the integral of the error term reduces the steady-state error to zero because it represents the accumulated error. To further improve stability and plant dynamics, a differential of the error term is introduced. This term represents the error rate. A PID controller that includes all three terms can give very good results. This technique is also being used in discrete form with digital control systems.

Two different approaches are used for conversion of PID into discrete form: rectangular and trapezoidal approximations. For the rectangular approximation, the design is done in the analog domain and then converted into z-domain. For the trapezoidal approximation, the design is done directly in the z-domain, using pole placement techniques.

The analog PID algorithm is given by

$$u(t) = K_p e(t) + K_i \int d_t + K_d \frac{de}{dt} \tag{8}$$

where

| | |
|---|---|
| $K_p$, $K_i$, and $K_d$ | = PID constants |
| $u(t)$ | = output of controller |
| $e(t)$ | = error signal |

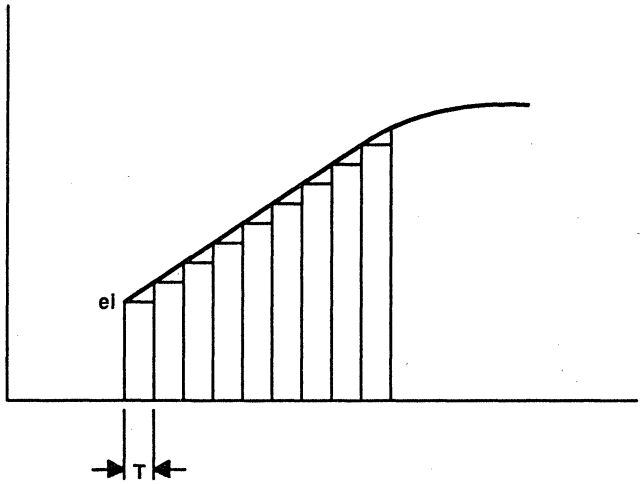
*Rectangular Approximation*

To discretize, assume that the sampling interval for the system is T. The rectangular approximation is the easiest to use and gives satisfactory results. For the rectangular approximation, assume the integral $\int edt$ is an accumulation of small rectangles given by $T \times \sum e(i)$ for i = 1 to n (see Figure 4). The differential de/dt (if T is sufficiently small) can be approximated by

$$\frac{e(n) - e(n-1)}{T} \tag{9}$$

where e(n) = the nth sample.

**Figure 4. Rectangular Approximation**




After conversion into its discrete form and transferring into time domain, the final form is shown in (7). The complete derivation is described in Appendix B.

$$u(n) = u(n-2) + K_1 e(n) + K_2 e(n-1) + K_3 e(n-2) \tag{10}$$

where

$K_1$ = $K_p + K_d/T + K_i T$
$K_2$ = $K_i T - 2K_d/T$
$K_3$ = $K_d/T - K_p$
$u(n)$ = control signal at time interval n
$u(n-2)$ = $(n-2)$th sample
$e(n)$ = error signal at time $n$
$e(n-1)$ = error signal at time $n-1$
$T$ = sampling interval

Appendix E shows the code to implement (7) on the PID controller with rectangular approximation. The code takes 11 instructions on the TMS320C15 and executes in 2.6 μs at 20 MHz. The

MPY instruction performs a multiply in a single cycle. The LTD instruction loads the T register, implements a data shift or $z^{-1}$ operation, and adds the result of the multiply to the previous value in the accumulator, all in a single cycle.

$K_1$, $K_2$, and $K_3$ are obtained by designing $K_p$, $K_i$, and $K_d$ and using conventional techniques from classical control[6, 7] (see Appendix B). Figure 7 through Figure 6 show the step response of the PID controller with different values of $K_p$, $K_i$, and $K_d$.

**Figure 5. Step Response of the PID Controller with First Set of $K_p$, $K_i$, and $K_d$ Values**

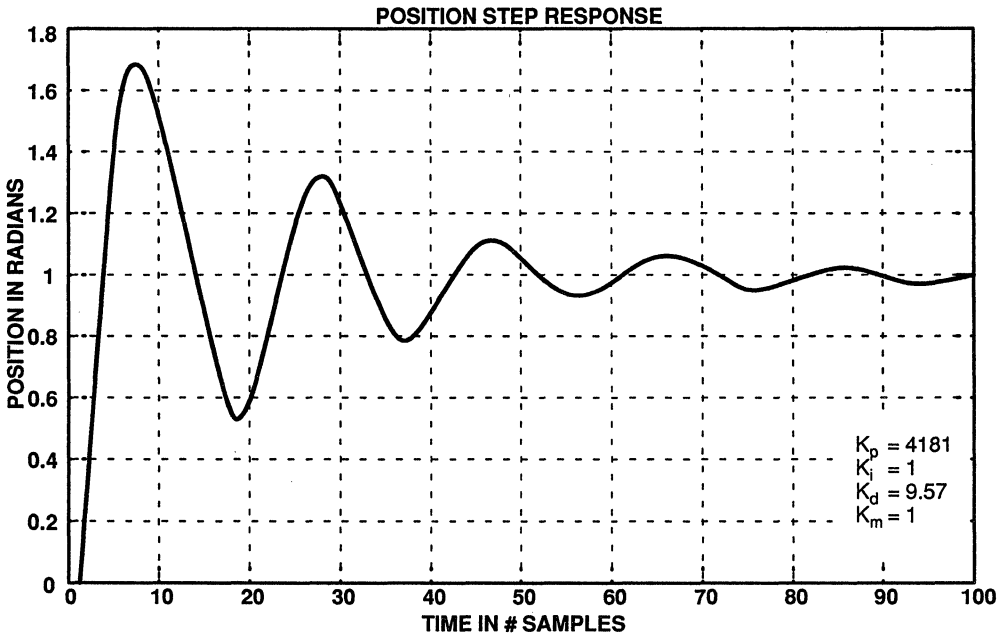


Figure 6. Step Response of the PID Controller with Second Set of $K_p$, $K_i$, and $K_d$ Values

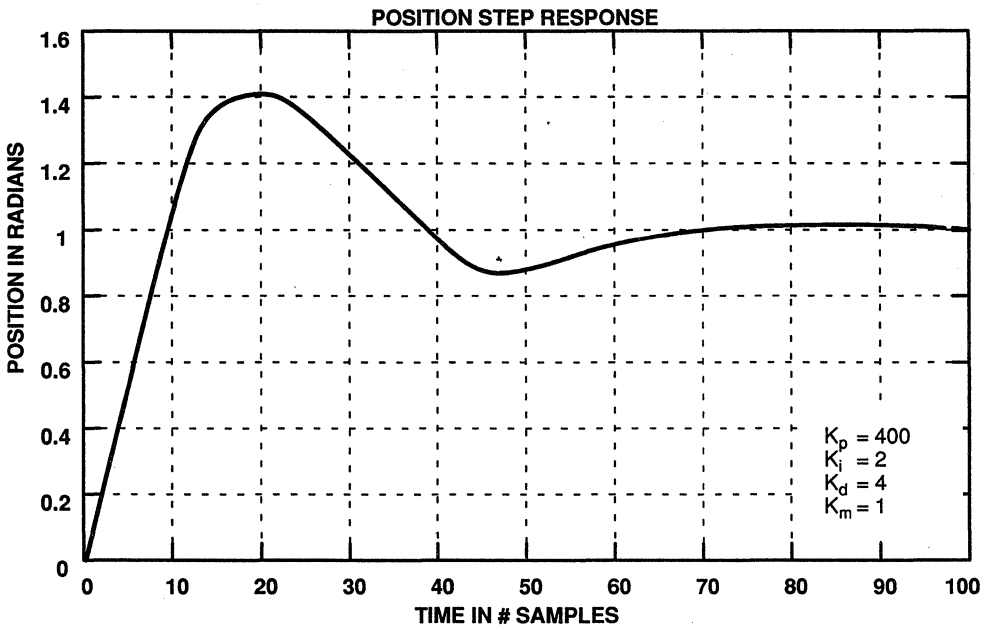
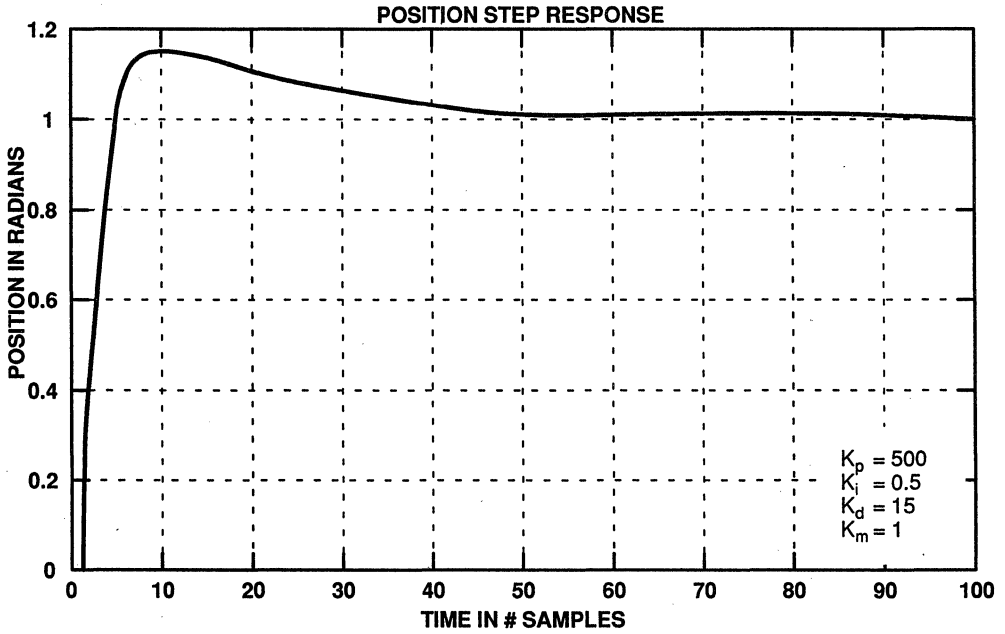


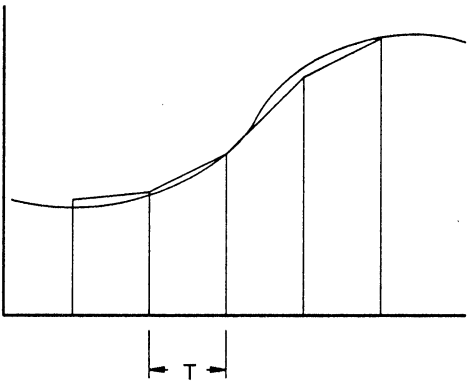

Figure 7. Step Response of the PID Controller with Third Set of $K_p$, $K_i$, and $K_d$ Values



*Trapezoidal Approximation*

If a more accurate conversion is required, the trapezoidal approximation or Tustin transformation is used. The area of the integral $\int edt$ is given by the summation of small trapezoids (see Figure 8).

**Figure 8. Trapezoidal Approximation**




The integral $\int edt$ can also be solved by taking the Laplace transform of (8) and substituting

$$s = 2/T \times (z-1/z+1) \tag{11}$$

After substitution and solving,

$$u(n) = u(n-2) + K_1e(n) + K_2e(n-1) + K_3e(n-2) \tag{12}$$

where

$K_1$ $= K_p + 2K_d/T + K_iT/2$
$K_2$ $= K_iT - 4K_d/T$
$K_3$ $= 2K_d/T - K_p + K_iT/2$
$u(n)$ = nth sample of output of controller
$u(n–2)$ = $(n–2)$nd sample of output

The complete derivation of this equation is described in Appendix B. The code to implement (12) on the PID controller is shown in Appendix E; it takes 12 instructions and executes in 2.8 μs.

The gain constants $K_1$, $K_2$, and $K_3$ are designed by selecting the poles for the system transfer function[15] (see Appendix B). The dominant poles are selected by choosing a desired characteristic equation. The rest of the poles can be selected by placing them near the origin. These polar locations are chosen to ensure system stability and a desired system response. However, some fine tuning may be necessary to achieve an optimum response from the system. As the poles move toward

the unit circle, the system response speed decreases while the overshoot decreases; the system may become unstable if the poles are selected too near or outside the unit circle. Figure 9 through Figure 11 show the step response of the PID controller for various pole locations for the system.

**Figure 9. PID Controller Step Response for System's First Pole Locations**

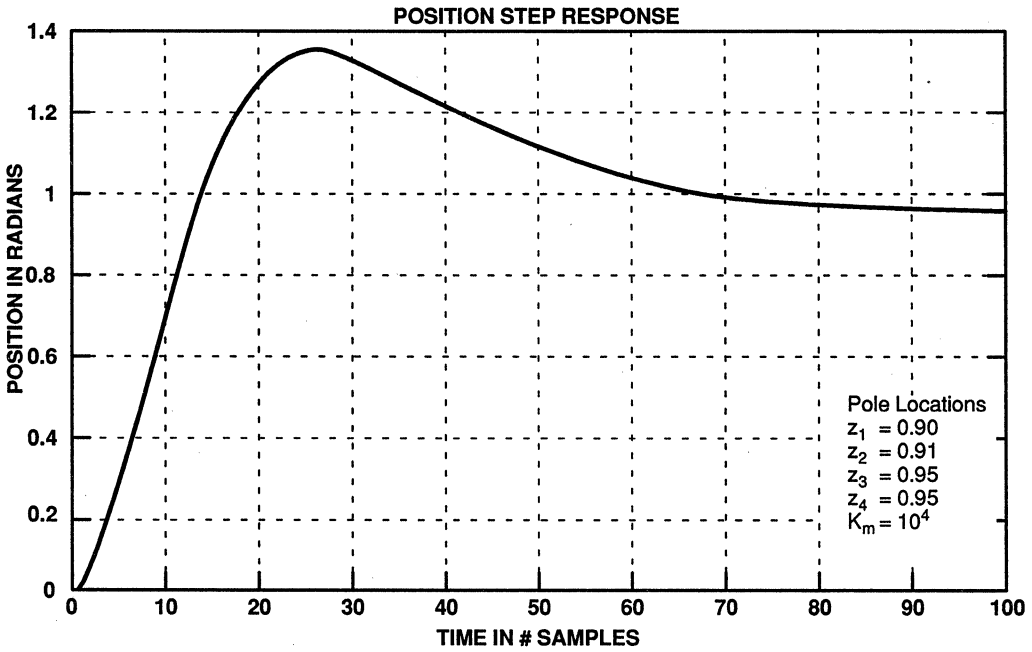


Figure 10. PID Controller Step Response for System's Second Pole Locations

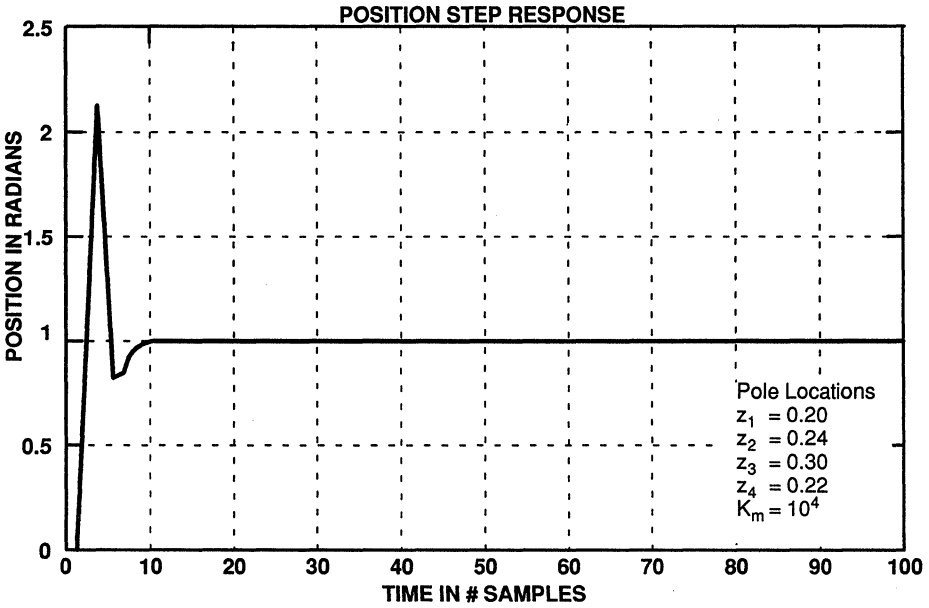
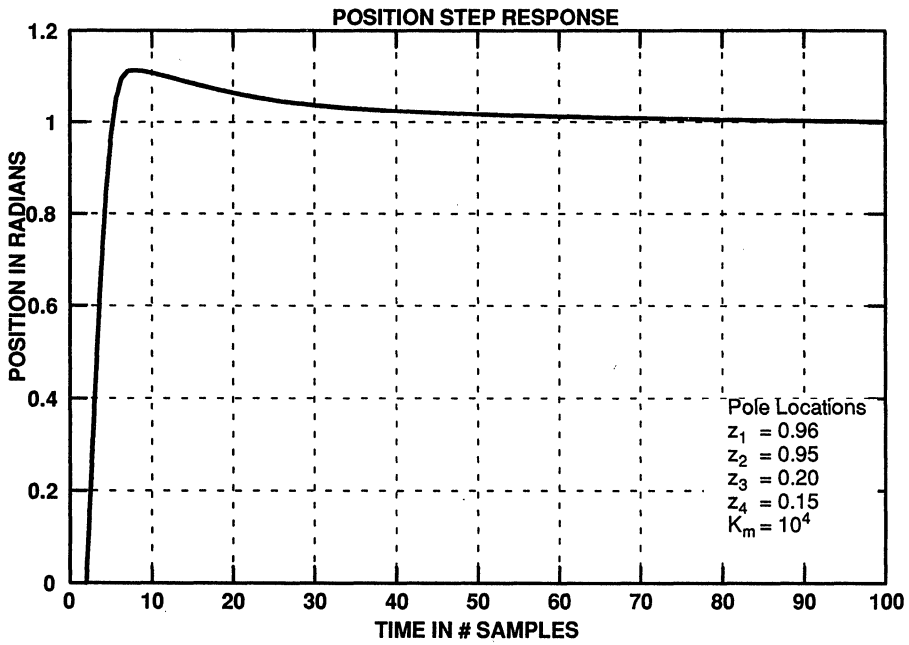



Figure 11. PID Controller Step Response for System's Third Pole Locations

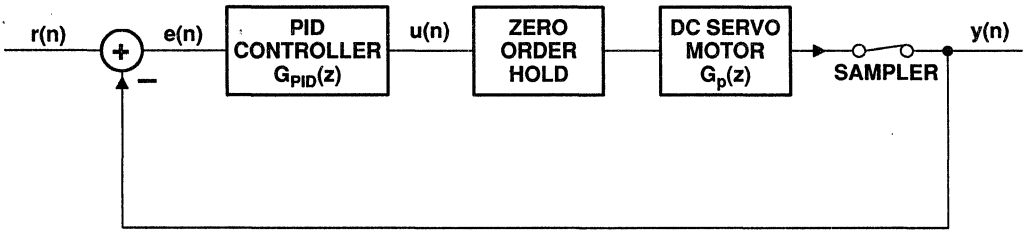


Our final algorithm comes out to

$$u(n) = u(n-2) + 0.162e(n) - 0.439e(n-1) + 0.3185e(n-2) \quad (13)$$

Figure 12 shows the the block diagram of a system using a PID controller. The zero-order hold represents the function of a D/A, and the sampler represents the function of an A/D.

**Figure 12. PID Controller System Block Diagram**




## Deadbeat

One of the desired characteristics in a control system design is a quick settling time. In an analog controller, it takes the system output an infinite time to settle exactly to the reference input signal. A deadbeat controller is used when a quick or finite settling time is required. A deadbeat controller reaches a steady state in n+1 samples, where n is the order of the controller. Essentially, a deadbeat controller cancels all the poles of the system and replaces them with poles at the origin. Another advantage of deadbeat controllers is that they require few calculations. Therefore, they can be used in systems where synthesis must be repeated frequently (e.g., in adaptive control systems).

Deadbeat controllers compensate for the poles of the system; therefore, they should not be applied to systems with poles outside or in the vicinity of the unit circle in the z-plane. Thus, deadbeat controllers should be used only with stable plants or processes; otherwise they may cause instability. Deadbeat controllers may also require a large amount of gain, thus leading to actuator saturation.

Deadbeat controllers also give a large overshoot. The only design parameter in deadbeat controllers is the sampling period that influences the magnitude of the control signal. When deadbeat control is used, the magnitude of the control signal increases as the sampling period decreases, otherwise, a larger overshoot occurs. Thus, it is important to choose the sampling period carefully when using deadbeat control. Besides increasing the sampling period, there are two other ways to reduce the overshoot.

1) One is to design an extended-order deadbeat controller[8], which allows u(0) or the initial control action to be specified. Since u(0) has the largest magnitude, this allows the overshoot to be controlled.
2) An alternate method is divide the r(t) (the desired final state) into two or three sublevels and reach final steady state in 2(n+1) or 3(n+1) sample times instead of n+1 sample times. This has the same effect as increasing the sample time. However, the final overshoot can be more precisely controlled, depending on how r(t) is subdivided.

The transfer function of a deadbeat controller is given by

$$G_{db} = \frac{p_o + p_1 z^{-1} + p_2 z^{-2} + \ldots p_n z^{-n}}{q_o + q_1 z^{-1} + q_2 z^{-2} + \ldots q_n z^{-n}} \tag{14}$$

The order n of the controller transfer function is the same as the order of the plant transfer function, or n=2. The deadbeat controller will reach final state in n+1 or three sample time intervals. The coefficients $p_0$, $p_1$, $p_2$, $q_0$, $q_1$, and $q_2$ are found from the plant parameters. Appendix C describes the complete equations to find these parameters. Solving for the parameters of the controller, the final transfer function for the controller is

$$G_{db} = \frac{0.1566 - 0.3129z^{-1} + 0.1564z^{-2}}{1 - 0.4218z^{-1} - 0.4216z^{-2}} \tag{15}$$

Figure 13 shows the block diagram of a deadbeat controller and Figure 16 through Figure 15 show the step response of the deadbeat controller. The code to implement the deadbeat controller is given in Appendix E.

**Figure 13. Deadbeat Controller Block Diagram**

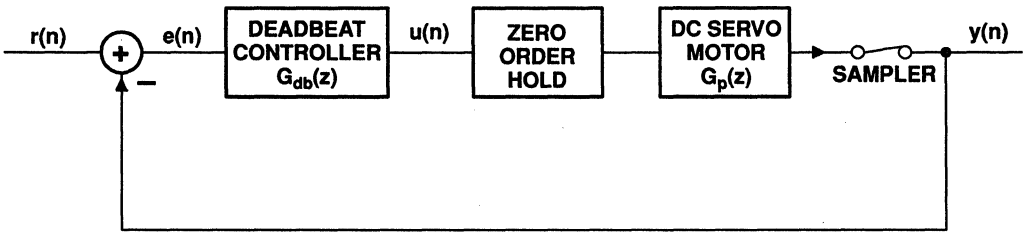


Figure 14. Step Response of the Deadbeat Controller

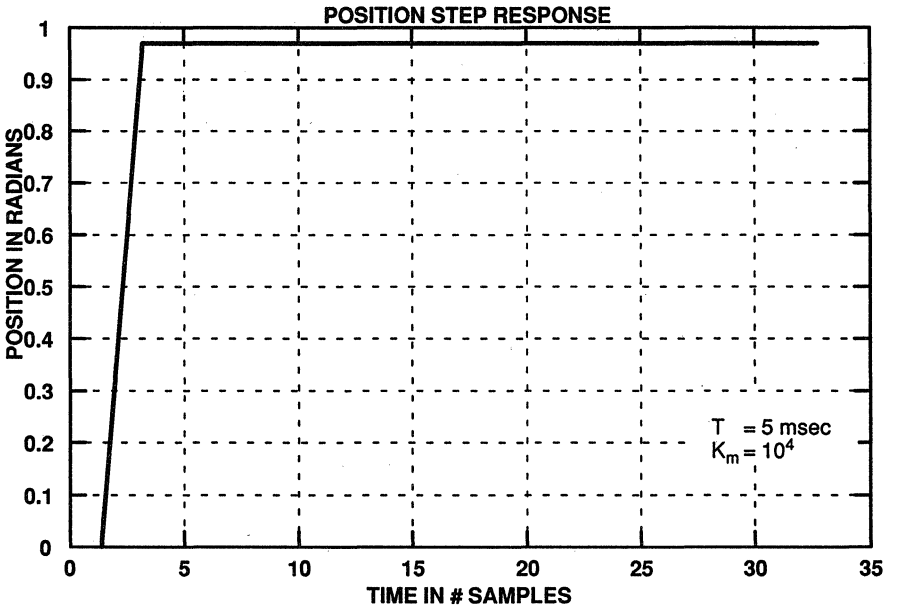



Figure 15. Step Response of the Deadbeat Controller

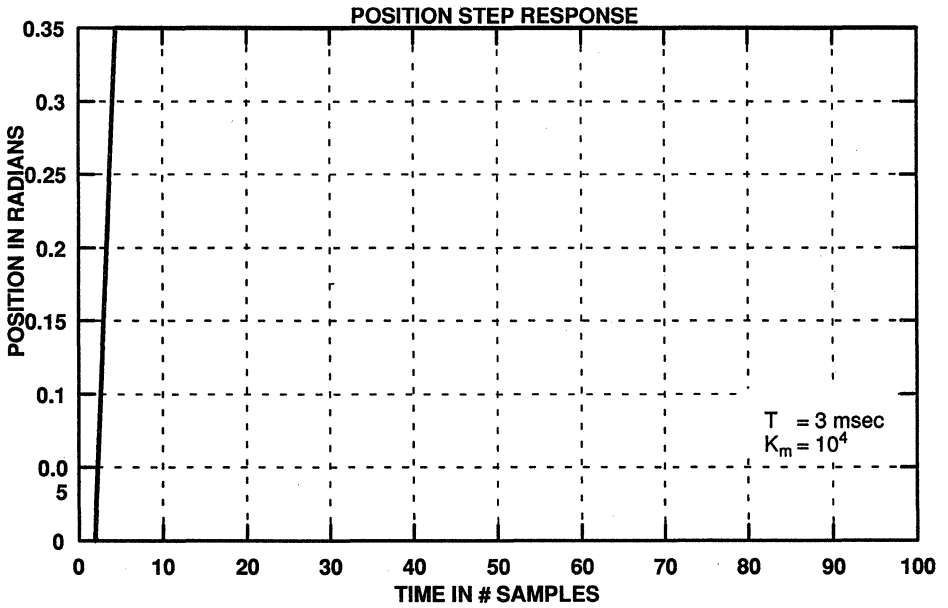
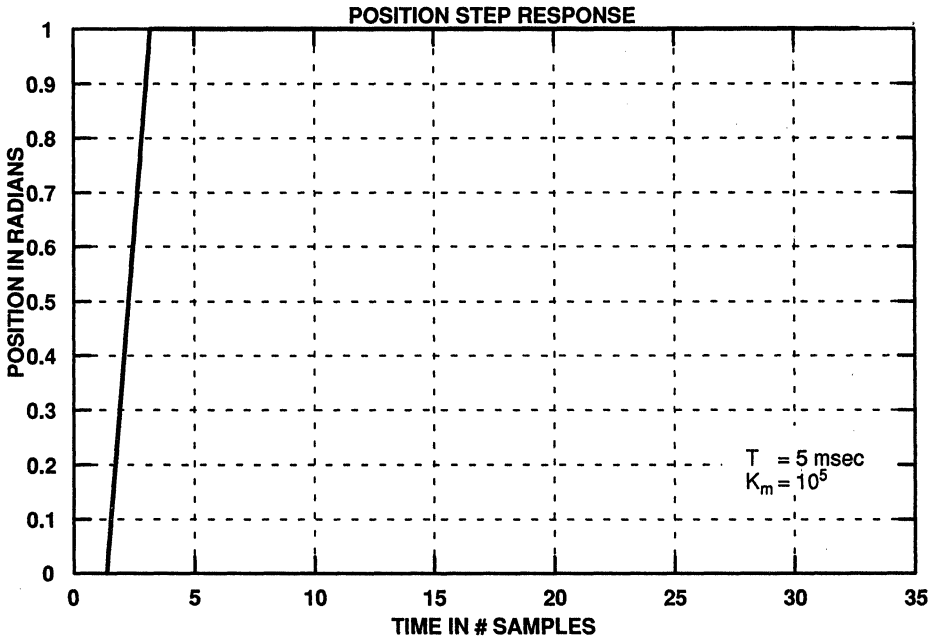


**Figure 16. Step Response of the Deadbeat Controller**



## Implementing Digital Controllers

This section will discuss some of the issues in implementing digital controllers[5, 17] without going into mathematical details. For mathematical models refer to references [6 through 11]. Perhaps the most critical issue in implementation is the effects of finite wordlength.

### Finite Wordlength Effects

Most digital controllers use fixed-point arithmetic processors. In a fixed-point arithmetic processor, only a finite amount of storage length (i.e., 4, 8, or 16 bits) is available to represent the magnitude of a signal or a gain constant. Signals and coefficients have to be scaled to fit in the dynamic range and wordlength of the processor. This limited storage capacity is referred to as the finite wordlength issue. The effects of finite wordlength show up as noise or limit cycles in the system, and may even cause instability. These effects are also referred to as quantization noise.

Another effect of finite wordlength shows up in the processing of signals. As intermediate calculations are carried out, higher precision is needed. For example, a 16×16 multiply needs a 32-bit register to store the result. If only 16 bits are available, the lower 16 bits are thrown away. This results in a loss of precision in the result, referred to as roundoff error. As successive calculations are carried out, these errors will accumulate. Another effect of finite wordlength is overflow management. Too often, registers will overflow during calculations. This usually causes registers to wrap around from most positive to most negative.

To minimize finite wordlength effects, a minimum of 16-bit wordlength is required, with 32-bit registers for internal precision. In addition, extensive simulations should be carried out to determine the dynamic range of the signals. Once system dynamics are known, proper scaling factors along with structure optimization techniques can reduce most of the effects of finite wordlength.

Selection of a proper scaling factor is critical in minimizing the effects of finite wordlength. The scale factor should support the full dynamic range of signals and coefficients. A large scale factor may cause overflows. Although overflow protection is built into the TMS320 architecture, it is advisable to minimize overflows. To minimize overflow, sometimes it may be necessary to choose a smaller (12–13 bits) scale factor. A small scale factor, on the other hand, may cause quantization noise or even underflow.

Usually, there is little choice in handling the dynamic range of signals. If the dynamic range is too big, it may dictate selection of a floating-point instead of a fixed-point arithmetic processor. Simulations are required to determine the dynamic range. However, in some cases, it may be possible to switch modes and change scale factors.

If gain coefficients have a large dynamic range, direct structures should be avoided and broken into smaller cascaded structures. Different scale factors can be chosen for different sections. Another approach is to use structure transformation techniques like Schur transformation or Modal transformation to optimize structures. These transformation techniques not only reduce the dynamic range of coefficients, but also reduce the number of nonzero elements in the structure. This minimizes processor calculations.

## Fixed-Point Versus Floating-Point Arithmetic Processors

One of the ways to avoid finite wordlength effects is to use a floating-point arithmetic processor. Floating-point arithmetic processors have a very large dynamic range. A 32-bit floating-point arithmetic processor has a dynamic range large enough for most control system applications. Floating-point arithmetic processors are usually expensive; their cost can be justified in only a few applications. Floating-point arithmetic processors may be needed in applications where either signals or gain coefficients are time varying and have a large dynamic range. Another case that justifies floating-point arithmetic processors is one in which cost of development cost is more significant than component cost and very low volumes are required.

If gain coefficients have a large dynamic range but are constant, their dynamic range can usually be reduced by structure optimization techniques. Floating-point arithmetic processors usually allow code to be developed in high-level languages and reduce the need to fully identify system dynamic range.

## Sampling Rate Selection

An important consideration is the selection of sampling rate. In signal processing, the sampling rate is chosen to be at least twice the bandwidth, or the highest frequency component in the systems. In control systems, the sampling rate chosen is six to ten times the system bandwidth. If lower sampling rates are selected, noise from higher frequency components may be introduced into the system and can be indistinguishable from the signal. Anti-aliasing filters are used before the controller to filter out high frequency components. A first-order filter should be used to minimize the phase shift.

## Controller Design Tools

A major consideration in using digital controllers is the design of hardware and software. One advantage of using digital controllers is that a large number of CASE (computer aided software engineering) tools are becoming available. These tools tremendously increase the productivity of the control designer.

### *Code Development*

Software or code development cost is a major concern in implementing digital controllers. The programmable approach to controllers allows easy upgrade and maintenance. It also protects development investment, but at the same time it requires more initial development effort. Six different approaches can be taken for software development:

- *High-Level Languages (HLL)* – Use of an HLL like C, Pascal, or FORTRAN can substantially cut development effort. Such languages are familiar and easy to program. However, they are not optimized with respect to signal processing or to a particular processor's architecture. Code compiled on a processor may be two to four times the size of assembly code. This is a high penalty in time-critical signal processing applications.
- *Assembly Language* – The most efficient coding occurs in assembly language. Even when an HLL language is used, it may be necessary to use assembly language for the more

time-critical sections. Assembly language programming requires an intimate knowledge of the processor architecture. However, the nature of performance requirements in signal processing requires maximum code efficiency, leaving very little choice in some cases.

- *Signal Processing Languages* – This may provide a mid-ground between the two approaches discussed above. Special languages designed for signal processing may give the ease of development of standard HLL languages. At the same time, they can give some of the efficiency of assembly language since they are designed for specific signal processing applications. One example of these special languages is DSPL from dSPACE[18]. However, there are no standards for these languages, and none of the languages are widely known.
- *Code Generation Software* – Code generation packages are becoming available that automatically generate assembly code for particular processors. For example, the Impex software package[19] from dSPACE will generate TMS320 assembly code from a mathematical description of the controller. The DFDP (Digital Filter Design Package)[20] from ASPI will generate assembly code for TMS320 processors from a description of a filter. These packages are becoming increasingly popular because they allow the control designer to focus on design issues instead of processor architecture in developing software.
- *Controller Design and Simulation Software* – Controller design and simulation packages are also becoming popular. Packages like Matrix-X[21] and PC-Matlab[22] can be used for simulation and design of controllers. These packages allow use of pole placement and other techniques. Packages like Simnon[23] are extremely good for simulation of continuous and discrete controllers. These packages greatly increase the productivity of the control engineer.
- *Device Simulators* – Another very useful tool in designing software is the device simulator. Simulators for the TMS320 family run on common platforms like PC or VAX and provide full simulation of the instruction set, along with instruction timing. These allow simulation of the controller software to fully check the effects of math on internal registers and memory without the necessity of building hardware.

## Hardware Design

A large variety of tools is available for designing the hardware for a controller. These include target systems and EVMs that plug into a PC or stand alone, and in-circuit emulators that can be used for complete system debugging. Also available are device behavioral models. These models can be used to simulate the timing and behavior of a complete target system without building any actual hardware. Logic Automation provides behavioral models for most members of the TMS320 family that run on popular workstations. Also available are logic analyzers from manufacturers like HP and Tektronix that can be used for extensive tracing. These logic analyzers can disassemble captured data to allow debugging of code[24].

## Applications

An increasing number of designers for control system applications are turning to DSPs to solve their problems. The capabilities of DSPs are also making applications practical that were previously impossible to implement or not cost effective. As costs of DSPs come down, they will replace microcontrollers and analog components in most control applications. Some of the control system applications in which DSPs are already cost effective are servo control in computer peripherals, vector control in AC motors, motion control in robotics and NC machines, and power control in power supplies and uninterruptible power supplies.

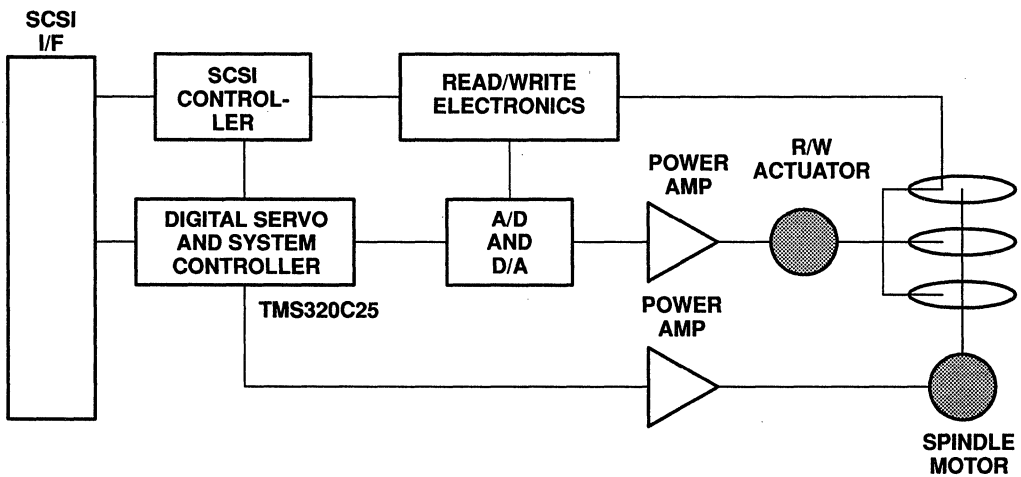
### Computer Peripherals

A large number of applications in computer peripherals are starting to use DSPs. These applications include read/write head control in winchester disk drives, tape control in tape drives, pen control in plotters, beam positioning and focusing in optical disks and in paper feed and print head control in printers.

#### *Disk Drives*

Disk drive designers were early adopters of DSPs. DSPs are used for servo control of the actuator driving the read/write head. Disk drives employ a voice-coil motor with high bandwidth. In addition, data is read from the disk at a very high rate. Sampling rates of up to 50 kHz are sometimes used. Besides implementing the compensator, DSPs implement notch filters to cancel mechanical resonances or vibrations[25]. Figure 17 shows the block diagram of a disk drive.

**Figure 17. Disk Drive Block Diagram**

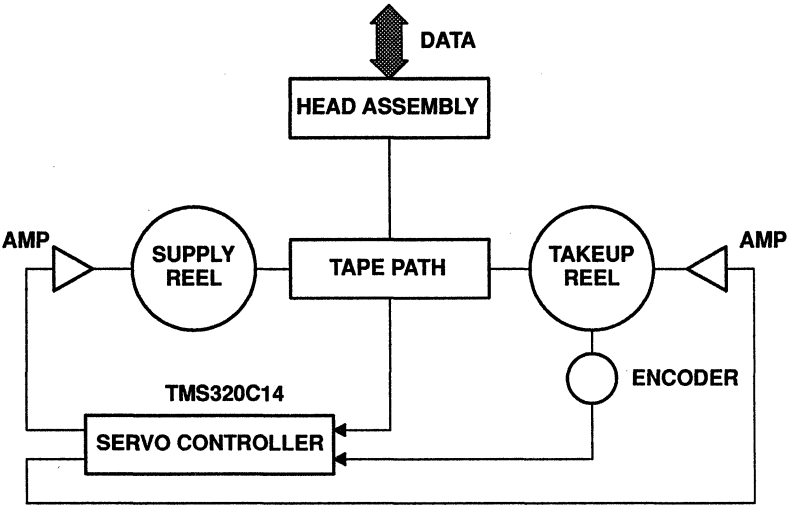



#### *Tape Drives*

In tape drives, DSPs are used for control of the tape mechanism. A tape drive has two servo loops that control tape speed and tension on the tape. Position feedback is obtained from an optical

encoder, and tension information is fed from a tension sensor. In addition, DSPs are also used to cancel mechanical resonances. Figure 18 shows the block diagram of a tape drive.

**Figure 18. Block Diagram of a Tape Drive Controller**



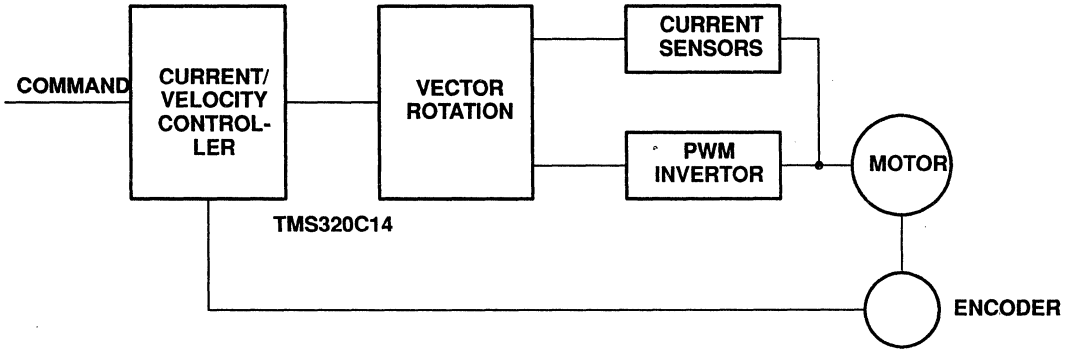

## Power Electronics

DSPs can be used for multiple applications in power electronics. These applications include AC servo drives, converter control, robotics, and motion control.

### *AC Servo Drives*

In AC servo drives, DSPs are used for vector control of AC motors. AC drives are less expensive and easier to maintain than DC drives. However, AC drives have complex control structures resulting from the cross-coupling of three-phase currents. Vector rotation techniques are used to transform three-phase to rotating two-phase d-q axes. This simplifies the analysis to a field-wound DC motor[26, 27, 28]. Figure 19 shows the block diagram of AC servo control.

Figure 19. AC Servo Control




*UPS/Power Converters*

In uninterruptible power supplies and power converters, DSPs are used for PWM generation along with power factor correction and harmonic elimination. Advanced mathematical techniques can be used to control the firing angles of the inverter to create low harmonic PWM with unity power factors[29, 30].

*Robotics/Motion Control*

DSPs are being used in large applications in robotics and other axis-control applications. DSPs allow high-precision control along with implementation of advanced techniques like state estimators and adaptive control. A single controller can handle speed/position control along with current control. Time-varying loads can be handled by using adaptive control techniques. Adaptive control techniques can also be used to create universal controllers that can be used with different motors. In addition to implementing controllers, DSPs can be used to implement notch filters to cancel resonances or vibrations[31].

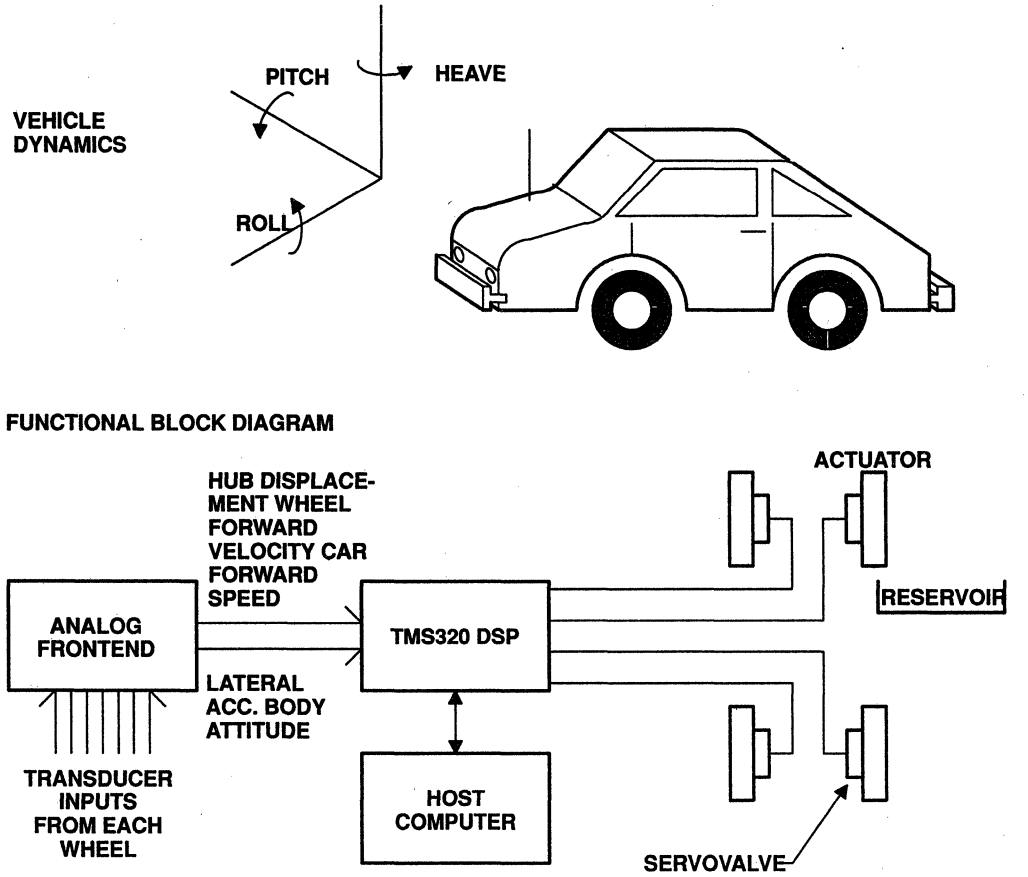
## Automotive

DSPs can be used for many applications in automotive design. These applications include active suspension, anti-skid braking, engine and transmission control, and noise cancellation [32, 33].

*Active Suspension*

Active suspension systems use four hydraulic actuators, one at each corner of the vehicle. DSPs can take into consideration body dynamics, such as pitch, heave, and roll. This information is used to control the four actuators independently, and to dynamically counter the external forces and car attitude changes[34]. Figure 20 shows block diagram of an active suspension system.

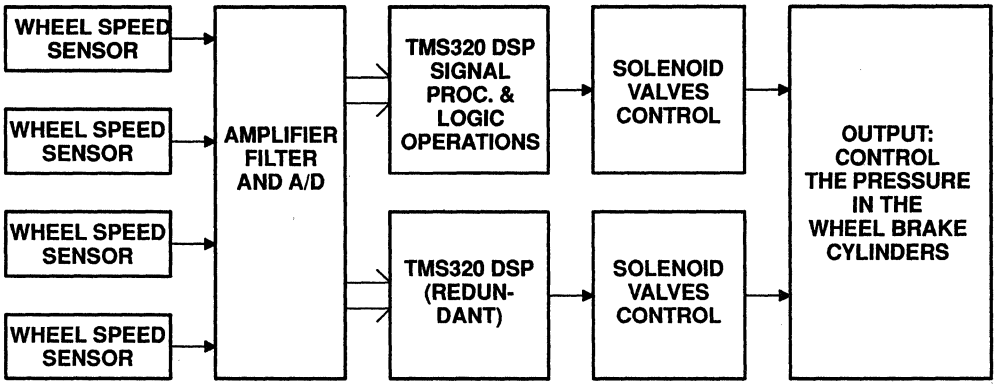
**Figure 20. An Active Suspension System**



*Anti-Skid Braking*

In anti-skid braking systems, DSPs can be used to read the wheel speed from sensors, calculate the skid, and control the pressure in the wheel brake cylinder. Traction control can be added to control the vehicle in extreme condition (wheel lock and spinning) and to further increase vehicle stability, steerability, and drivability. Figure 21 shows a block diagram of an ABS system.

**Figure 21. Anti-Lock Braking System (ABS) Block Diagram**




*Engine Control*

In engine control applications, DSPs can be used with in-cylinder pressure sensors to perform engine pressure waveform analysis. This information can be used to determine the best spark timing, firing angles, and the optimal air/fuel ratios. The closed-loop engine control scheme can tolerate external turbulences, aging, wearing, etc., and maintains optimum engine performance and fuel efficiency.

## Summary

This report has discussed implementation of digital control systems with the TMS320 family of processors, using various control algorithms. The report has focused on showing design procedures and implementation of generic digital control systems without going into specific applications or choosing a particular approach or algorithm. Obviously, selection of a specific approach depends on the requirements of a particular application. However, the procedures outlined in this report can be applied with minor modifications to a wide range of applications. This report has also attempted to provide a bridge for control system designers who have been trained in analog control design and want to convert their analog designs into digital designs for stability, higher performance, or other reasons.

TMS320-based digital control systems have numerous advantages over analog-based designs. The high processing speed of the TMS320 family allows sophisticated control techniques to be used to build a high-precision control system. Digital systems are insensitive to component aging and temperature drift, thus minimizing variation in controller gain coefficients. With the TMS320 an adaptive control system can also be designed, thus creating a truly robust system that is insensitive to plant parameter variation. A digital control system using the TMS320 can also be employed to control multiple devices or time shared between different processes. An observer system may also be designed with a TMS320-based system to eliminate expensive sensors.

In addition to the advantages outlined above for TMS320-based control systems, the tradeoffs and disadvantages in implementing digital control are no longer applicable. The 16-bit word length, and 32-bit ALU and 32-bit accumulator of the TMS320 make quantization errors negligible. The hardware scaling shifters of the TMS320 family further minimize errors due to quantization and truncation. The fast processing speed of the TMS320 allows high sampling rates to be used, thus giving analog-like performance and minimizing delay time.

## References


1) *TMS320C1x User's Guide*, literature number SPRU013B, Texas Instruments, 1989.
2) *TMS320C14/E14 Users Guide*, literature number SPRU032, Texas Instruments, 1988.
3) *TMS320C2x User's Guide*, literature number SPRU014A, Texas Instruments, 1989.
4) *TMS320C3x User's Guide*, literature number SPRU031, Texas Instruments, 1988.
5) *Digital Signal Processing Applications with the TMS320 Family*, literature number SPRA012, Texas Instruments, 1986.
6) Astrom, K., and Wittenmark, B., *Computer Controlled Systems*, Prentice-Hall Inc., 1984.
7) Phillips, C., and Nagel, H., *Digital Control System and Analysis and Design*, Prentice-Hall Inc., 1984.
8) Iserman, R., *Digital Control Systems*, Springer-Verlag, 1981.
9) Franklin, G., and Powell, D., *Digital Control of Dynamic Systems*, Addison-Wesley, 1980.
10) Jacquot, R., *Digital Control Systems*, Marcel Dekker, 1981.
11) Katz, P., *Digital Control Using Microprocessors*, Prentice-Hall Inc., 1981.
12) Lewis, F., *Optimal Control*, John Wiley and Sons, 1986.
13) Lewis, F., *Optimal Estimation*, John Wiley and Sons, 1986
14) Kyriakopoulos, N. and Tan, J., "Implementation of a Tracking Kalman Filter on a Digital Signal Processor," *IEEE Transactions on Industrial Electronics*, pp. 126–134, Volume 35, Number 1, February 1988.
15) Astrom, K. and Hagglund, T., "Automatic Tuning of PID Controllers," *Instrument Society of America*, 1988.
16) Astrom, K., and Wittenmark, B., "Adaptive Control," Addison-Wesley, 1988.
17) Hanselmann, H., "Implementation of Digital Controllers – A Survey," *Automatica*, Volume 23, Number 1, pp. 7–32, (1987).
18) Hanselmann, H., and Schwarte, A. "Generation of Fast Target Processor Code from High Level Controller Descriptions," *Proceedings of 10th IFAC World Congress*, Munich, July 1987.
19) Impex is a trademark of dSPACE, Paderborn, Germany.
20) DFDP is a trademark of Atlanta Signal Processors, Atlanta, GA.
21) Matrix-X is a trademark of Integrated Controls, Santa Clara, CA.
22) Matlab is a trademark of Mathworks, Inc, South Natick, Ma.
23) Simnon is a trademark of Lund Institute of Technology, Lund, Sweden.
24) *TMS320 Family Development Support Reference Guide*, literature number SPRA011A, Texas Instruments, 1989.
25) Hanselmann, H., and Engelke, A., "LQG – Control of a Highly Resonant Disc Drive Head Positioning Actuator," *IEEE Transactions on Industrial Electronics*, Volume 35, Number 1, pp. 100–104, February 1988.
26) Bose, B. K., and Szczesny, P., "A Microcomputer-Based Control and Simulation of an Advanced IPM Synchronous Machine Drive System for Electric Vehicle Propulsion," *Proceedings of IECON'87*, pp. 454–463, October 1987.
27) Matsui, N., and Ohasi, H., "DSP-Based Adaptive Control of a Brushless Motor," *Proceedings of IECON'88*, pp. 375–380, October 1988.

28) Leonhard, W., Lessmeier, R., and Schumacher, W., "Microprocessor-Controlled AC-Servo Drives With Synchronous or Induction Motors: Which is Preferable?," *IEEE Transactions on Industry Applications*, September/ October 1986.
29) Chance, R., and Taufiq, T., "A TMS32010 Based Near Optimized Pulse Width Modulated Waveform Generator," *Third International Conference on Power Electronics & Variable Speed Drives*, Conference Publication Number 291, July 1988.
30) Garate, I., Carrasco, R., and Bowden, A., "An Integrated Digital Controller for Brushless AC Motors Using a DSP Microprocessor," *Third International Conference on Power Electronics & Variable Speed Drives*, Conference Publication Number 291, pp. 249–252, July 1988
31) Tomizuka, M., Horowitz, R., and Anwar, G., "Implementation of a MRAC for a Two Axis Direct Drive Robot Manipulator Using a Digital Signal Processor," *Proceedings of American Control Conference*, pp. 658–660, 1988.
32) Costin, M. and Elzinga, D., "Active Reduction of Low-Frequency Tire Impact Noise Using Digital Feedback Control," *IEEE Control Systems Magazine*, pp. 3–6, August 1989.
33) Lin, K., " Trends of Digital Signal Processing In Automotive," *Proceedings of CONVERGENCE '88*, October 1988.
34) Williams, D. and Oxley, S., "Application of the Digital Signal Processor to an Automotive Control System," *Proceedings of the Sixth International Conference on Automotive Electronics*, Great Britain, October 1987.

# Appendix A

## Plant Modeling

The discrete time model for a DC motor can be derived from the electrical and mechanical characteristics of the motor. The mechanical characteristics are:

$$J_m\ddot{\theta} + B\dot{\theta} + K\theta = T_L - J_L\ddot{\theta} \tag{16}$$

where

$Jm$ = motor inertia
$B$ = viscous damping
$K$ = stiffness
$J_L$ = load inertia
$\theta$ = position or angular displacement
$\dot{\theta}$ = $d\theta/dt$ = angular velocity
$\ddot{\theta}$ = $d^2\theta/dt^2$ = angular acceleration
$T_L$ = load torque = $K_t \times i$
$K_t$ = torque constant
$i$ = current

The electrical characteristics are given by

$$L\frac{di}{dt} + Ri = V - EMF \tag{17}$$

where

$L$ = inductance
$R$ = resistance
$V$ = applied voltage
$EMF$ = $K_e \times \dot{\theta}$ = back emf
$K_e$ = emf constant
$\dot{\theta}$ = angular velocity

The electrical time constant is given by L/R, and the mechanical time constant is given by B/J.

In practice, L/R << B/J; i.e., electrical steady-state conditions are reached quickly. Assuming steady-state current is reached, (17) is reduced to

$$Ri = V - EMF = V - K_e\dot{\theta} \tag{18}$$

Combining (16) and (18) gives

$$(J_m + J_L)\ddot{\theta} + B\dot{\theta} + K\theta = K_t\,(V - K_e\dot{\theta})/R \tag{19}$$

Assuming $J_m + J_L = J$ = system inertia, and K = 0 = stiffness constant, the system equation becomes

$$\ddot{\theta} + 1/J(B + K_t K_e/R)\,\dot{\theta} = 1/J(K_t/R)V \tag{20}$$

where

$a = 1/J\,(B + K_t\,K_e/R)$
$b = 1/J\,(Kt/R)$

The Laplace transform of (20) is

$$\text{or} \quad \begin{aligned} (s^2 + as)\theta(s) &= bV(s) \\ \theta(s)/V(s) &= b/s(s + a) \end{aligned} \tag{21}$$

(21) is the final form of the transfer function of the motor in continuous form. This must be converted into a discrete form. The ZOH transformation is used.

ZOH states that

$$G(z) = (1 - z^{-1})\mathbf{Z}(\mathbf{L}^{-1}G(s)/s) \tag{22}$$

Then,

$$\frac{G\,(s)}{s} = \frac{b}{s(s + a)(s)} = \frac{b}{s^2(s + a)} \tag{23}$$

Expanding as partial fractions, (23) is expressed as

$$\frac{G(s)}{s} = \frac{A_1}{s} + \frac{A_2}{s^2} + \frac{A_3}{s + a} \tag{24}$$

Solving for $A_1$, $A_2$, and $A_3$ gives

$$\frac{G(s)}{s} = \frac{(-b/a^2)}{s} + \frac{(b/a)}{s^2} + \frac{(b/a^2)}{s + a} \tag{25}$$

When multiplying by $(1 - z^{-1})$ and using tables to derive the z-transform,

$$G(z) = \frac{b/a^2\,(e^{-aT} - 1 + aT)z^{-1} + b/a^2(1 - e^{-aT} - aTe^{-aT})\,z^{-2}}{1 - (1 + e^{-aT})\,z^{-1} + e^{-a}Tz^{-2}} \tag{26}$$

where $T$ = sampling period.

Substituting values for $a$, $b$, and $T$ of

$a = 1.116$
$b = 53.906$
$T = 0.001$

the transfer function of the motor becomes

$$G_m(z) = \frac{\theta(z)}{V(z)} = \frac{0.269z^{-1} + 0.269z^{-2}}{1 - 1.999z^{-1} + 0.999z^{-2}} \times K_g 10^{-4} \tag{27}$$

where $K_g$ is a gain constant.

By introducing a numerator gain factor, (27) can be rewritten as

$$G_m(z) = \frac{\theta(z)}{V(z)} = \frac{0.269z^{-1} + 0.269z^{-2}}{1 - 1.999z^{-1} + 0.999z^{-2}} \times K_m \tag{28}$$

where $K_m$ is a numerator gain factor.

# Appendix B

## PID Controller

The PID algorithm is given by

$$u(t) = K_p e(t) + K_i \int edt + K_d \frac{de}{dt} \tag{29}$$

where

$K_p, K_i, K_d$ = PID gain constants
$u(t)$ = control signal
$e(t)$ = error signal

***Rectangular Approximation***

To convert to discrete form, the integral term edt is approximated by the summation of rectangles $\sum e_i \times T$, where $T$ is the sampling interval and $e_i$ is the value of the error signal at sample time $i$, written as

$$\int edt = \sum e_i T \tag{30}$$

If the sampling interval T is small enough, the differential term $d_e/d_t$ can be approximated by

$$\frac{de}{dt} = \frac{e\ (n) - e\ (n-1)}{T} \tag{31}$$

where $e(n)$ and $e(n–1)$ are values of the error signal $e$ at time intervals $n$ and $n–1$.

The PID algorithm can now be approximated in discrete form by

$$u(n) = K_p e(n) + K_i \sum e_i T + K_d[e(n) - e(n-1)]/T \tag{32}$$

To reduce (32) into a difference equation, (32) for time interval $n–2$ is written as

$$u(n-2) = K_p e(n-2) + K_i \sum_{i=1}^{n-2} e_i T + K_d[e(n-1) - e(n-2)]/T \tag{33}$$

Subtracting (33) from (32) gives

$$\begin{aligned} u(n) - u(n-2) &= K_p e(n) - e(n-2) + K_i[e(n) + e(n-1)]T \\ &\quad + (K_d/T)\{\ e(n) - 2e(n-1) + e(n-2)\ \} \end{aligned} \tag{34}$$

Combining similar terms gives

$$u(n) = u(n-2) + (K_p + K_d/T + K_iT)e(n) - (K_iT - 2K_d/T)e(n-1) + (K_d/T - K_p)e(n-2) \quad (35)$$

or

$$u(n) = u(n-2) + K_1e(n) + K_2e(n-1) + K_3e\ (n-2)$$

where

$K_1 = K_p + K_d/T + K_iT$
$K_2 = K_i\ T - 2K_d/T$
$K_3 = K_d/T - K_p$

$K_1$, $K_2$, and $K_3$ are obtained from the design of $K_p$, $K_i$, and $K_d$, which are designed by using conventional techniques. One way of designing is to use the Ziegler and Nichols ultimative-sensitivity method [6]. With this approach, a proportional controller is first used to control the system. The gain of the controller, Kmax, and the period time, $T_p$, when the closed loop is on the stability boundary, are measured. The parameters $K_p$, $K_i$, and $K_d$ can then obtained as follows:

$K_p = 0.6$ Kmax
$K_i = T_p/2$
$K_d = T_p/8$

Another tuning method is to use the phase margin and critical frequencies [7]. Using this technique, $K_p$, $K_i$, and $K_d$ can be computed as follows.

$$\theta = 180 + \phi_m - G_m\ (\ j\omega\ )\ |\ \omega = \omega_c$$

$$Kp = \frac{\cos(\theta)}{|G_m\ (\ j\omega\ )\ |\omega = \omega_c} \quad (36)$$

$$K_d\omega_c - K_i/\omega_c = \frac{\sin(\theta)}{|\ G_m(j\omega)\ |\ \omega = \omega_c}$$

By substituting an arbitrary value of $K_i$ in the above equation, we can obtain $K_d$. Using this technique and designing with the following parameters

$\phi_m = 55°$ (phase margin)
$\omega_c = 628.315$ radians (critical frequency = 100 hz)

we obtain

$$|\ G_m(\ j\omega\ |\ \omega = \omega_c = 0.0001365$$

$$G_m(j\omega)|\omega = \omega_c = -179.8982 \quad (37)$$

The PID constants are then found to be

$K_p = 4181$
$K_d = 9.569$
$K_i = 1$

$K_1$, $K_2$, and $K_3$ are then obtained as

$K_1$ = 13751
$K_2$ = −19138
$K_3$ = 5387

Both these methods give approximate answers. Further fine tuning of the parameters may be necessary to get the desired response from the system.

The controller is obtained as

$$u(n) = u(n-1) + K_1e(n) + K_2e(n-1) + K_3e(n-2) \tag{38}$$

***Trapezoidal Approximation***

Another method for converting the analog form of the PID algorithm into a discrete form is to use a trapezoidal approximation, sometimes referred to as the Tustin approximation.

The PID is again given as

$$u(t) = K_pe(t) + K_i \int edt + K_d\frac{de}{dt} \tag{39}$$

where

$$e(t) = \frac{de}{dt} \tag{40}$$

The Laplace transform of that is expressed as

$$U(s) = K_pE(s) + K_iE(s)/s + K_dsE(s) \tag{41}$$

Combining gives

$$U(s) = (K_p + K_ds + K_i/s)E(s) \tag{42}$$

Conversion into discrete form requires a transfer from s-domain to z-domain by using the Tustin approximation,

$$s = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} \tag{43}$$

where T = sampling period.

Substituting (43) in (42) gives

$$\frac{U(z)}{E(z)} = \frac{(2K_pT + 4K_d + T^2K_i) + (2T^2K_i - 8K_d)z^{-1} + (4K_d - 2TK_p + T^2K_i)z^{-2}}{2T(1 - z^{-2})} \tag{44}$$

Further computation yields

$$U(z) - U(z)z^{-2} = K_1E(z) + K_2E(z)z^{-1} + K_3E(z)z^{-2} \tag{45}$$

where

$K_1 = K_p + 2K_d/T + K_iT/2$
$K_2 = K_iT - 4K_d/T$
$K_3 = K_iT/2 + 2K_d/T - K_p$

In (45), $z^{-1}$ represents a delay of one sample time. Taking an inverse z-transform of (45) gives

$$\begin{aligned} & u(n) - u(n-2) = K_1e(n) + K_2e(n-1) + K_3e(n-2) \\ \text{or} \\ & u(n) = u(n-2) + K_1e(n) + K_2e(n-1) + K_3e(n-2) \end{aligned} \tag{46}$$

This is the final form of the PID controller. However, before implementing the controller, the constants $K_p$, $K_i$, and $K_d$ must be located. Alternatively, constants $K_1$, $K_2$, and $K_3$ have to be determined. These constants can be found by locating the poles of the equivalent system transfer function (i.e., controller + plant).

The transfer function for the controller can be stated as

$$G_c(z) = \frac{K_1 + K_2z^{-1} + K_3z^{-2}}{1 - z^{-2}} \tag{47}$$

The transfer function of the plant is given by

$$G_p(z) = \frac{0.2694z^{-1} + 0.2693z^{-2}}{1 - 1.999z^{-1} + 0.999z^{-2}} \times K_m \tag{48}$$

The overall system transfer function is expressed as

$$G_s(z) = \frac{G_p(z)G_c(z)}{1 + G_p(z)G_c(z)} \tag{49}$$

The denominator of the system transfer function provides the poles of the overall system. The stability and robustness of the system depends upon the location of these poles in the z-domain. Assuming pole locations of 0.96, 0.95, 0.20 and 0.15, a desired characteristic equation for the denominator is obtained. Program 4 for PC-Matlab, in Appendix D, lists the steps to obtain the characteristics equation from pole locations and obtain values of K1, $K_2$ and $K_3$. To solve for values of $K_1$, $K_2$, $K_3$, the coefficients of powers of z for the denominator of the system transfer function are compared with the desired polynomial.

Solving for $K_1$, $K_2$, and $K_3$ gives
$K_1 = 1.4795$,
$K_2 = -2.845$,
$K_3 = 1.3636$

The controller is

$$u(n) = u(n-2) + 1.4795e(n) - 2.8405e(n-1) + 1.3636e(n-2) \tag{50}$$

# Appendix C

## Deadbeat Controller

A deadbeat controller has the property of settling to a final state in a finite time. It has the form,

$$G_{db}(z) = \frac{p_0 + p_1 z^{-1} + p_2 z^{-2} \ \ldots\ldots\ p_n z^{-n}}{q_0 + q_1 z^{-1} + q_2 z^{-2} \ \ldots\ldots\ q_n z^{-n}} \tag{51}$$

To design the deadbeat controller, its coefficients $p_0$, $p_1$, ... $q_0$, $q_1$, .. have to be found from the parameters of the motor.

The general form of a plant (i.e., motor) is given by

$$G_{dp}(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} \ \ldots\ldots\ b_n z^{-n}}{a_0 + a_1 z^{-1} + a_2 z^{-2} \ \ldots\ldots\ a_n z^{-n}} \tag{52}$$

If R(z) is the reference input, the coefficients $p_n$ and $q_n$ are

$p_0 = r / \sum b_i = r / (b_0 + b_1 + b_2)$
$p_1 = a_1 p_0$
$p_2 = a_2 p_0$
.
.
.
$p_n = a_n p_0$
and

$q_0 = r - p_0 b_0$
$q_1 = -b_1 p_0$
$q_2 = -b_2 p_0$
.
.
.
$q_n = -b_n p_0$

The transfer function of the dc servo motor is

$$G_m(z) = \frac{0.269z + 0.269}{z^2 - 1.999z + 0.999} \times K_m \qquad (K_m = 4000) \tag{53}$$

Since the plant transfer function is a second-order system, the deadbeat controller is also a second-order system (n = 2).

From the plant transfer function,

$a_0 = 1, \quad a_1 = -1.999, \quad a_2 = 0.999$
$b_0 = 0, \quad b_1 = 0.269 \quad b_2 = 0.269$

The numerator and denominator of $G_{db}(z)$ are divided by $r$. Thus, r disappears from the calculations of coefficients.

Solving for the coefficients yields

$$
\begin{aligned}
p_0 &= 1/(b_0 + b_1 + b_2) = 0.1566 \\
p_1 &= a_1\, p_0 = -0.3129 \\
p_2 &= a_2\, p_0 = 0.1564 \\
q_0 &= 1 - p_0\, b_0/r = 1 \\
q_1 &= -b_1\, p_0 = -0.4218 \\
q_2 &= -b_2\, p_0 = -0.4216
\end{aligned}
$$

The controller becomes

$$G_{db}(z) = \frac{0.1566 - 0.3129z^{-1} + 0.1564z^{-2}}{1 - 0.4218z^{-1} - 0.4216z^{-2}} \tag{54}$$

or, in time domain, it can be represented as

$$
\begin{aligned}
u(n) &= 0.4218u(n-1) + 0.4216u(n-2) \\
&\quad + 0.1566e(n) - 0.3129e(n-1) + 0.1564e(n-2)
\end{aligned} \tag{55}
$$

# Appendix D

PC-Matlab is an interactive program developed by Mathworks for scientific and engineering numerical calculations. Included are many routines for design and simulations for signal processing and control systems. The programs in this appendix design the PID and deadbeat controllers and display step responses for the system. The programs are interactive and allow the user to change certain parameters. The programs use PC-Matlab and the Control Tool Box. Control Tool Box is a set of PC-Matlab utilities for control system design and simulation. PC-Matlab runs on MS-DOS computers. More information on PC-Matlab can be obtained from Mathworks. [22]

PC-Matlab Programs

Program 1

```
%    This program implements simulation of a dc servo motor. The motor
%    used in the example is a Pittman motor, model 9412.
%
Kt=0.0207;                  % Torque constant
Ke=Kt;
j=0.00006;                  % Armature inertia + assumed load inertia
R=6.4;                      % Resistance
input('input sampling period in milliseconds')
T=ans/1000;                 % get sampling period
a=(Kt^2)/(R*j)              % a, and b give transfer function in s-domain
b=Kt/(j*R)
ab=b/(a^2);                 % Calculate values to transfer into z-domain
c=exp(-a*T);
d1=a*T;
d=(c-1+d1);
e=(1-c-(c*d1));
input('input numerator gain   ')
Kp=ans;                     % get numerator gain
b1=ab*d*Kp;                 % numerator terms
b2=ab*e*Kp;
a1=-(1+c);                  % denominator terms
a2=c;
num=[0 b1 b2]               % numerator of transfer function in z-domain
den=[1 a1 a2]               % denominator of transfer function in z-domain
end
```

Program 2

```
%    This program implements design of a PID controller using classical
%    techniques. The design is then be converted into discrete form. This
%    design calculates values of Kp, Ki, and Kd or lets you enter these
%    values manually
%
' Enter 0 if you want to enter PID constants manually'
' Enter 1 if you want program to calculate these for you'
input('input 0 or 1   ')
X=ans
if X==1,
  N1=[0  0  b]              % numerator of transfer function in s-domain
  D1=[1  a  0]              % denominator of transfer function in s-domain
  input('input critical frequency in Hz  ')
  Wi=ans;                   % get critical frequency
  w=Wi*2*pi                 % Assuming critical frequency Wc=100Hz
  [mag,phase]=bode(N1,D1,w)  % calculate phase and magnitude of motor
  input(' input phase margin in degrees   ')
  Pm=ans;                   % get phase margin
  O=180 + Pm - phase - 360  % calculate theta
  Or=O/57                   % convert into radian
%
  Kp=cos(Or)/mag            % calculate  Kp
  Ki=1                      % assume Ki =1
  Kd =(Ki/w+sin(Or)/mag)/w  % Calculate Kd
  pause
elseif X==0,
  input('input p ')
  Kp=ans;
  input('input i ')
  Ki=ans;
  input('input d ')
  Kd=ans;
else 'No assigned values for constants, 0 or 1 was not input'
pause
end
K1 = Kp + Kd/T + Ki*T       % convert design into discrete form
K2 = Ki*T - 2*Kd/T
K3 = Kd/T - Kp
end
```

Program 3

```
%    This program simulates a closed loop system using a PID controller. The
%    PID controller has been designed using classical techniques and then
%    converted into discrete form using rectangular approximation.
%
motrans                          % call program to simulate motor
pidrect                          % call program to design PID constants
num1=[K1 K2 K3];                 % numerator of controller
den1=[1  0  -1];                 % denominator of controler
%
compnum=num1
compden=den1
procnum=num                      % numerator of motor
procden=den                      % denominator of motor
%
num5=conv(num1,num)              % multiply both numerators
den5=conv(den1,den)              % multiply both denominators
pause
input('specify the time in secs over which you want to see the step: ')
t=ans;
n=t/T;                           % calculate number of samples for simulation
input('input a loop gain: ')
g=ans;
u=ones(n,1);
closnum=g*num5                   % numerator of closed loop system
closden=g*num5+den5              % denominator of closed loop system
pause
y=dlsim(closnum,closden,u);      % do simulation
plot(y)
title('Position Step Response')
xlabel('Time in # samples')
ylabel('Position in radian')
grid
pause
end
end
```

Program 4

```
%    This program designs a PID controller using pole placement techniques.
%    Desired pole locations have to be input. The PID is converted into
%    discrete form using trapezoidal approximation. Enter desired pole
%    locations in the next step.
%
'Enter the location of your poles'
input('Input location of pole 1:    ')
p1=ans;
input('Input location of pole 2:    ')
p2=ans;
input('Input location of pole 3:    ')
p3=ans;
input('Input location of pole 4:    ')
p4=ans;
p=[p1 p2 p3 p4];
%
%    The desired characteristic polynomial is found as
%
Q(1:5)=poly(p)
%
%    The coefficients of different powers are given by
%
q2=Q(:,2);
q3=Q(:,3);
q4=Q(:,4);
q5=Q(:,5);
%
%    The system polynomial is given by
%
%    (K1z**2 + K2*z + K3)(b1*z + b2) + (z - 1)(z - r)(z**2 - a1*z +a2)
%
%    Equating coefficients of different powers, we get four linear equations.
%    The next few steps will solve for K1, K2, K3 and r, where r is a location
%    of one of the poles of the controller.
%
D = [ b1        0         0         -1
      b2        b1        0         1-a1
      0         b2        b1        a1-a2
      0         0         b2        a2 ];
%
D1= [ q2+1-a1   0         0         -1
      q3+a1-a2  b1        0         1-a1
      q4+a2     b2        b1        a1-a2
      q5        0         b2        a2 ];
%
D2= [ b1        q2+1-a1   0         -1
      b2        q3+a1-a2  0         1-a1
      0         q4+a2     b1        a1-a2
      0         q5        b2        a2 ];
%
```

```
D3= [ b1        0         q2+1-a1    -1
      b2        b1        q3+a1-a2   1-a1
      0         b2        q4+a2      a1-a2
      0         0         q5         a2 ];
%
D4= [ b1        0         0          q2+1-a1
      b2        b1        0          q3+a1-a2
      0         b2        b1         q4+a2
      0         0         b2         q5 ];
%
d=det(D);
d1=det(D1);
d2=det(D2);
d3=det(D3);
d4=det(D4);
K1=d1/d
K2=d2/d
K3=d3/d
r=d4/d
end
```

Program 5

```
%    This file simulates a closed loop deadbeat controller. If the
%    plant transfer function is
%
%         G(z) = A/B
%
%    and controller function is given by
%
%         H(z) = C/D
%
%    then the closed loop response is given by
%
%         G(z)H(z)        AC
%         ------------ = --------
%         1 + G(z)H(z)    AC + BD
%
ggg=1
while ggg==1                    %  Keep doing
motrans                         %  Call motor transfer function
dbeatdes                        %  Call deadbeat parameters
num1=[p0 p1 p2];                %  Numerator of the controller
den1=[q0 q1 q2];                %  denominator of controller
compnum=num1;
compden=den1;
procnum=num;
procden=den;
num5=conv(num1,num);            % multiply both numerators
den5=conv(den1,den);            % multiply both denominators
input('specify the time in secs over which you want to see the step: ')
t=ans;
n=t/T;                          % Calculate number of samples to see simulation
input('input a loop gain: ')
g=ans;
u=ones(n,1);
closnum=g*num5;                 % Enter additional closed loop gain
closden=g*num5+den5;            % Calculate denominator of closed loop system
y=dlsim(closnum,closden,u);     % Do closed loop simulation
plot(y)
title('Position Step Response')
xlabel('Time in # of samples')
ylabel('Position in radian')
grid
pause
end
```

Program 6

```
%    This file implements design of a deadbeat controller. The form of the
%    controller is given by the following equation
%
%                          -1        -2       -3           -n
%                  p0 + p1*z     + p2*z   + p3*z  ........pn*z
%        G  (z) = ------------------------------------------------
%         db               -1          -2       -3            -n
%                  q0 + q1*z      + q1*z   + q3*z  ........qn*z
%
%    If the plant transfer function is given by
%
%                        -1       -2       -3          -n
%                 b0 + b1*z   + b2*z   + b3*z  ........bn*z
%        G (z)= ------------------------------------------------
%         p               -1       -2       -3           -n
%                 a0 + a1*z   + a1*z   + a3*z  ........an*z
%
%    then the following procedure can be used to design a deadbeat controller
%
p0 = 1/(1 + b1 + b2)
p1 = a1*p0
p2 = a2*p0
q0 = 1
q1 = -b1*p0
q2 = -b2*p0
end
```

Program 7

```
%    This program simulates a PID controller using trapezoidal approximation
%    and a pole placement technique
%
%    If the plant transfer function is
%
%         G(z) = A/B
%
%    and controller function is given by
%
%         H(z) = C/D
%
%   then the closed loop response is given by
%
%          G(z)H(z)       AC
%         ----------- = -------
%         1 + G(z)H(z)   AC + BD
%
ggg=1
while ggg==1
motrans                 % Call program to calculate motor transfer function
pidtdes                 % Call program to calculate controller gains
num1=[K1 K2 K3];        % numerator of PID controller
R1=[1,r];               % poles of the PID controller
den1=poly(R1);          % calculate denominator
compnum=num1;
compden=den1;
procnum=num;
procden=den;
num5=conv(num1,num);    % Multiply numerators
den5=conv(den1,den);    % Multiply denominators
input('specify the time in secs over which you want to see the step: ')
t=ans;
n=t/T;                  % Calculate number of samples to see simulation
input('input a loop gain: ')  % Enter any additional loop gain
g=ans;
u=ones(n,1);            % Number of samples to see simulation
closnum=g*num5          % numerator of closed loop system transfer function
closden=g*num5+den5     % denominator of closed loop system transfer function
y=dlsim(closnum,closden,u);   % do discrete simulation
plot(y)
title('Position Step Response')
xlabel('Time in # of samples')
ylabel('Position in radian')
grid
pause
end
```

# Appendix E

```
*       Program 1
*
*       PID Rectangular Controller
*
*
        .title      'PID Controller'
        .def        PID
*
*       This routine implements a PID controller
*
RV      .set    0           ; reference value
XN      .set    1           ; input from A/D
E0      .set    2           ; Latest error sample
E1      .set    3           ; Previous error sample
E2      .set    4           ; oldest error sample
K1      .set    5           ; gain constant
K2      .set    6           ; gain constant
K3      .set    7           ; gain constant
UN      .set    8           ; output to controller
U1      .set    9           ; previous output
U2      .set   10           ; oldest output
*
*       Processor initialization
*
reset   B       init        ; RS- processing begins here
int     B       isr
*
        .word   5632        ; coefficient K1
        .word  -7839        ; coefficient K2
        .word   2206        ; coefficient K3
*
*
init    LDPK    0           ; set DP pointer
        SOVM
        LARK    0,255
*
        ZAC
        LARP    0
loop    SACL    *           ; initialize memory
        BANZ    loop
        LACK    4           ; set program memory pointer to 4h
        TBLR    K1          ; load coeffcients into data memory
        LACK    5
        TBLR    K2
        LACK    6
        TBLR    K3
        EINT
        B       self        ; wait for interrupts
*
*    Process input sample
*
*       e(n) = r - x(n)
*
isr     IN      RV,PA2      ; read reference command input
        IN      XN,PA0      ; read input position signal on upper 13 bits
        LAC     XN,13
        SACH    XN
        LAC     RV
        SUB     XN          ; subtract from reference to give error
        SACL    E0

*
*   PID routine
*
*       u(n) = u(n-2) + K1*e(n) + K2*e(n-1) + K3*e(n-2)
*
PID     LAC     U2          ; Transfer u(n-2) to accumulator
        LT      E2          ; load T register with oldest sample e(n-2)
        MPY     K3          ; Preg = K3*e(n-2)
        LTD     E1          ; ACC = u(n-2) + K3*e(n-2), Treg = e(n-1)
        MPY     K2          ; Preg = K2*e(n-1)
        LTD     E0          ; ACC = u(n-2) + K3*e(n-2) + K2*e(n-1)
        MPY     K1          ; Preg = K1*e(n)
        APAC                ; ACC=u(n-2) + K3*e(n-2) + K2*e(n-1) + K1*e(n)
        SACH    UN,4        ; shift out 4 sign bbits
        OUT     UN,PA1      ; write to D/A - two's complement form
        DMOV    U1          ; transfer u(n-1) --->  u(n-2)
        DMOV    UN          ; transfer u(n)   --->  u(n-1)
        EINT
self    NOP
        B       self        ; wait for next interrupt
```

```
*       Program 2
*
*       PID Trapezzoidal Controller
*
*
        .title      "PID Controller"
        .def        PID
*
*       This routine implements a PID controller
*
RV      .set    0           ; reference value
XN      .set    1           ; input from A/D
E0      .set    2           ; Latest error sample
E1      .set    3           ; Previous error sample
E2      .set    4           ; oldest error sample
K1      .set    5           ; gain constant
K2      .set    6           ; gain constant
K3      .set    7           ; gain constant
UN      .set    8           ; output to controller
U1      .set    9           ; previous output
U2      .set    10          ; oldest output
*
*       Processor initialization
*
reset   B       init        ; RS- processing begins here
int     B       isr
*
        .word   6060        ; coefficient K1 =  1.4795
        .word -11653        ; coefficient K2 = -2.8450
        .word   5585        ; coefficient K3 =  1.3636
*
*
init    LDPK    0           ; set DP pointer
        SOVM
        LARK    0,255       ; clear memory
*
        ZAC
        LARP    0
loop    SACL    *           ; initialize memory
        BANZ    loop
        LACK    4           ; set program memory pointer to 4h
        TBLR    K1          ; load coeffcients into data memory
        LACK    5           ; set program memory pointer to 5h
        TBLR    K2
        LACK    6           ; set program memory pointer to 6h
        TBLR    K3
        EINT
        B       self        ; wait for interrupt
*
*     Process input sample
*
*       e(n) = r - x(n)
*
isr     IN      RV,PA2      ; read reference command input
        IN      XN,PA0      ; read input position signal on upper 13 bits
        LAC     XN,13
        SACH    XN
        LAC     RV
        SUB     XN          ; subtract from reference to give error
        SACL    E0

*
*    PID routine
*
*       u(n) = u(n-2) + K1*e(n) + K2*e(n-1) + K3*e(n-2)
*
PID     LAC     U2          ; Transfer u(n-2) to accumulator
        LT      E2          ; load T register with oldest sample e(n-2)
        MPY     K3          ; Preg = K3*e(n-2)
        LTD     E1          ; ACC = u(n-2) + K3*e(n-2), Treg = e(n-1)
        MPY     K2          ; Preg = K2*e(n-1)
        LTD     E0          ; ACC = u(n-2) + K3*e(n-2) + K2*e(n-1)
        MPY     K1          ; Preg = K1*e(n)
        APAC                ; ACC=u(n-2) + K3*e(n-2) + K2*e(n-1) + K1*e(n)
        SACH    UN,4        ; store to memory and shift out 4 sign bits
        OUT     UN,PA1      ; write to D/A  - in two's complement form
        DMOV    U1          ; transfer u(n-1) --->  u(n-2)
        DMOV    UN          ; transfer u(n)   --->  u(n-1)
        EINT
self    NOP
        B       self        ; wait for next interrupt
```

```
*       Program 3
*
*       Deadbeat Controller
*
*
        .title      "Deadbeat Controller"
        .def        DBEAT
*
*       This routine implements a Deadbeat controller
*
RV      .set    0           ; reference value
XN      .set    1           ; input from A/D
E0      .set    2           ; Latest error sample
E1      .set    3           ; Previous error sample
E2      .set    4           ; oldest error sample
P0      .set    5           ; gain constant
P1      .set    6           ; gain constant
P2      .set    7           ; gain constant
Q1      .set    8           ; gain constant
Q2      .set    9           ; gain constant
UN      .set   10           ; output to controller
U1      .set   11           ; previous output
U2      .set   12           ; oldest output
*
*       Processor initialization
*
reset   B       init        ; RS- processing begins here
int     B       isr
*
        .word   5131        ; coefficient P0 =  .1566
        .word -10253        ; coefficient P1 = -.3129
        .word   5125        ; coefficient P2 =  .1564
        .word -13215        ; coefficient Q1 = -.4218
        .word -13815        ; coefficient Q2 = -.4216
*
*
init    LDPK    0           ; set DP pointer
        SOVM
        LARK    0,255
*
        ZAC
        LARP    0
loop    SACL    *           ; initialize memory
        BANZ    loop
        LACK    4           ; set program memory pointer to 4h
        TBLR    P0          ; load coeffcients into data memory
        LACK    5           ; set program memory pointer to 5h
        TBLR    P1
        LACK    6           ; set program memory pointer to 6h
        TBLR    P2
        LACK    7           ; set program memory pointer to 7h
        TBLR    Q1
        LACK    8           ; set program memory pointer to 8h
        TBLR    Q2
        EINT                ; enable interrupts
        B       self        ; wait for interrupt
*
*    Process input sample
*
*       e(n) = r - x(n)
```

```
*
isr     IN      RV,PA2          ; read reference command input
        IN      XN,PA0          ; read input position signal on upper 13 bits
        LAC     XN,13
        SACH    XN
        LAC     RV
        SUB     XN              ; subtract from reference to give error
        SACL    E0
*
*   Deadbeat Controller routine
*
*   u(n) = Q2*u(n-2) + Q1*u(n-1) + P0*e(n) + P1*e(n-1) + P2*e(n-2)
*
DBEAT   ZAC                     ; clear accumulator
        LT      E2              ; load T register with oldest sample e(n-2)
        MPY     P2              ; Preg = P2*e(n-2)
        LTD     E1              ; ACC =  P2*e(n-2), Treg = e(n-1)
        MPY     P1              ; Preg = P1*e(n-1)
        LTD     E0              ; ACC =  P2*e(n-2) + P1*e(n-1)
        MPY     P0              ; Preg = P0*e(n)
        LTA     U2              ; ACC =  P2*e(n-2)+P1*e(n-1)+P0*e(n), Treg=u(n-2)
        MPY     Q2              ; Preg = Q2*u(n-2)
        LTD     U1              ; ACC =  P2*e(n-2)+P1*e(n-1)+P0*e(n)+Q2*(n-1)
        MPY     Q1              ; Preg = Q1*u(n-1)
        APAC                    ; ACC=Q2*u(n-2)+Q1*u(n-1)+P0*e(n)+P1*e(n-1)+P2*e(n-2
        SACH    UN,4            ; write to memory and shift out 4 sign bits
        OUT     UN,PA1          ; write to D/A - in two's complement form
        DMOV    UN              ; transfer u(n)   --->  u(n-1)
        EINT
self    NOP
        B       self            ; wait for next interrupt
```

# Part VI. Tools

# TMS320 Algorithm Debugging Techniques

**Peter Robinson**

**Regional Technology Center — Waltham, Massachusetts**
**Texas Instruments**

## Introduction

Debugging a DSP algorithm is becoming more of an issue as our DSP tasks increase in complexity. It is easy enough to verify program flow for a filter or FFT, but it is an entirely different task to evaluate a discrete transfer function, H(n), over time and frequency.

In this report, a technique for debugging a coded transfer function, h(n), in a purely software environment is presented. The technique shows how traditional analog trouble-shooting methods can be applied to analyze and debug DSP algorithms on an IBM/PC-based TMS320C2x Software Development system and/or any of the TMS320 simulators.

## Data Logging

Data Logging is the ability to simulate an I/O device by using DOS files. Many, if not all, IBM/PC-based DSP software development tools and algorithm development packages, such as simulators, offer this ability or feature. In the case of the Texas Instruments TMS320C2x SWDS, the IN and OUT instructions can be equated or tied to a DOS file. Using files to simulate I/O devices can be extremely useful in analyzing performance of a transfer function such as a filter.

Assume that you have written an algorithm for the TMS320C25 for a first-order IIR filter, comprising one second-order element. Figure 1 shows how the code can be divided into four sections:

1. An initialization section where the coefficients are moved from program to data memory,
2. The aquisition of data, IN X0,PA1,
3. The IIR section, and
4. The output of y(n) via an OUT YN,PA0 (with a branch back to the IN instruction).

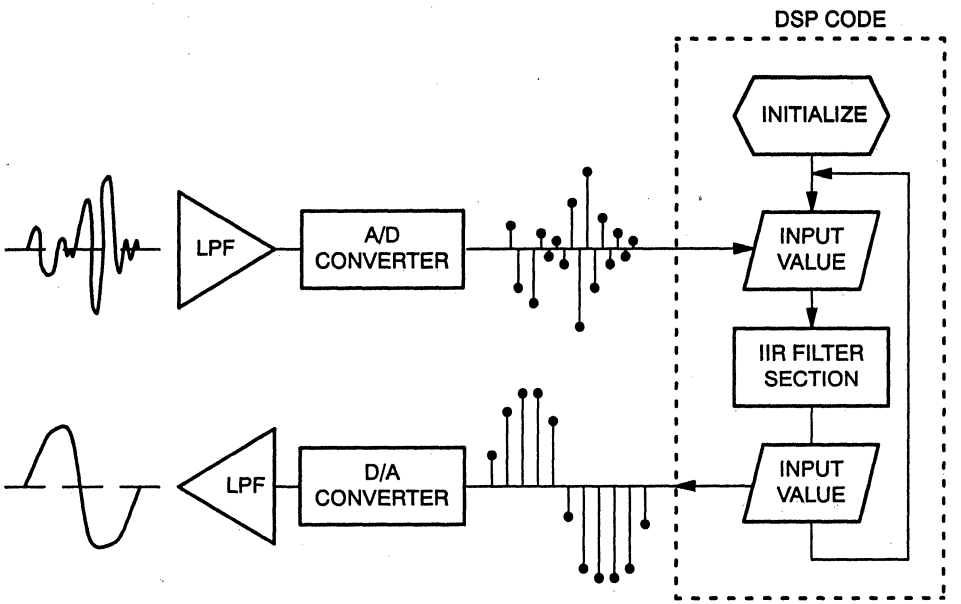



**Figure 1. DSP Code Division**

If this is an analog system made up of an op-amp, you can verify performance or response by sweeping the filter over frequency and observing the output on a spectrum analyzer. Since the example has no target system (this is entirely a software simulation), a file is used for both input (the A/D) and output (the D/A). To use the data logging feature as noted, you will need to create an input file.

## Creating the Input File with Lotus

Before you start, determine the format of the INPUT DOS file structure. In the case of the TMS320C2x SWDS, input files must be represented by four ASCII HEX characters (a 16-bit field), followed by a carriage return and line feed, <CR><LF>. An example for a file containing an impulse, emulating a 12-bit, two's-complement A/D, is shown in Figure 2.

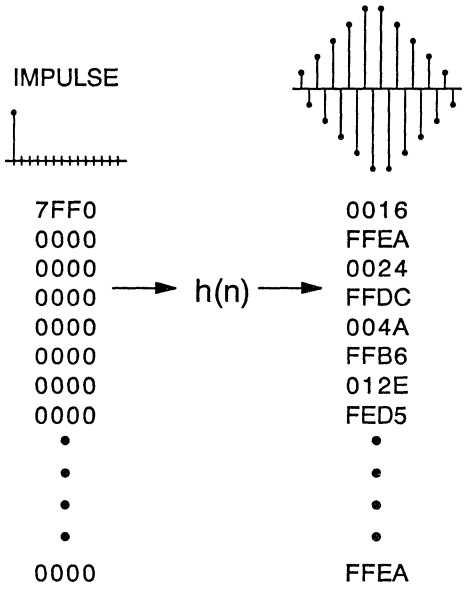
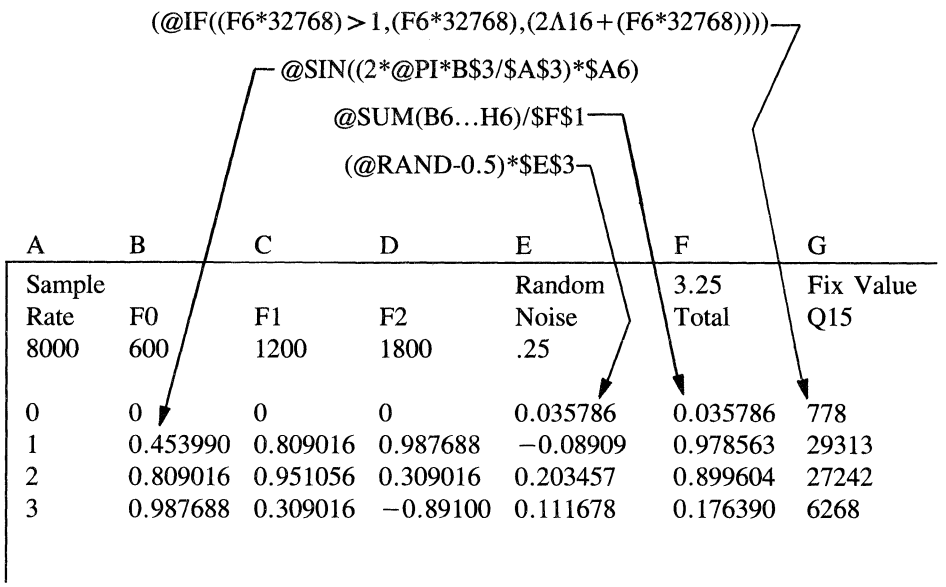



**Figure 2. Input Text Editor File**

Note: HEX values in this report are represented in Q15 format. In Q15 format, −1 = 8000h for both a 12- and 16-bit field, +1 = 7FF0h for a 12-bit field and 7FFFh for a 16-bit field. For further information on Qn notation, see Section 5.5.5 on page 5-33 of Reference [1].

The file shown in Figure 2 can easily be generated with a text editor and produces a near-ideal impulse response. This is seldom achieved with analog systems. However, what if you wanted to inject a more complex signal, such as several sine waves and/or random noise? Here, the generation of the input file can become a monumental task. One method of file generation, presented here, uses Lotus 1-2-3, a software package found on most PCs. Because Lotus 1-2-3 is a spreadsheet calculator, you can use a column to denote the input sequence, X(0), X(1), X(3) etc. Adjacent columns can be set up to calculate the desired x(N) values. An example of a spread sheet, which calculates three sine waves with a predetermined noise signal added in, is shown in Figure 3.




| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | Sample | | | | Random | 3.25 | Fix Value |
| 2 | Rate | F0 | F1 | F2 | Noise | Total | Q15 |
| 3 | 8000 | 600 | 1200 | 1800 | .25 | | |
| 4 | | | | | | | |
| 5 | 0 | 0 | 0 | 0 | 0.035786 | 0.035786 | 778 |
| 6 | 1 | 0.453990 | 0.809016 | 0.987688 | −0.08909 | 0.978563 | 29313 |
| 7 | 2 | 0.809016 | 0.951056 | 0.309016 | 0.203457 | 0.899604 | 27242 |
| 8 | 3 | 0.987688 | 0.309016 | −0.89100 | 0.111678 | 0.176390 | 6268 |
| 9 | | | | | | | |

**Figure 3. Spread Sheet Calculation of Three Sine Waves with Added Noise Signal**

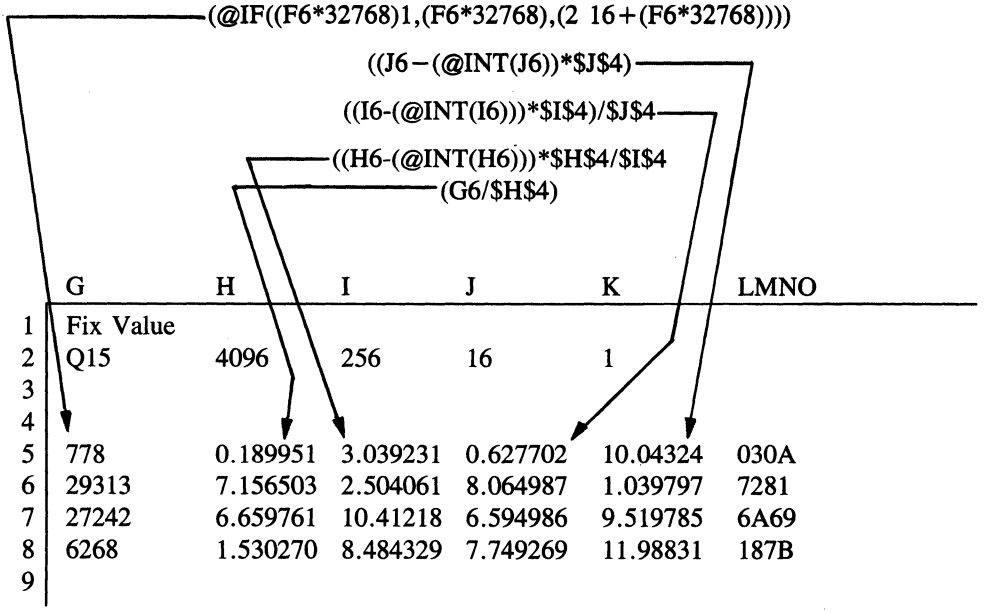
This spreadsheet approach allows you to specify a wide range of input conditions. You can add columns by copying previous column data and can extend the length of the array by copying rows. If you are going to use Lotus's random number generator for adding noise to your signal, you should note the following:

- Lotus's random number function generates uniform numbers or noise.
- Lotus appends an existing file when the print-to-file option is used.
- Each time you recalculate the spreadsheet with a random function, a new random seed is used, which will generate a new random array.

You can use this last feature to your advantage by writing to a file, recalculating the spreadsheet, then writing again. This sequence permits large input files with uniquely different file segments. You must exercise caution, however, to insure that all frequencies end at a zero crossing; if they don't, unwanted discontinuities will be introduced.

The end result of the above process is a column of decimal numbers scaled between −1 and +1. Using the Lotus graph utility, you can plot one or several full cycles of the wave form. When you get the desired results, you must convert the column data to a Qn HEX value of the form in the note that follows Figure 2. You can use the command structure noted in Figure 4.

@CHOOSE((@INT(H6)),"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F","0")




| | G | H | I | J | K | LMNO |
|---|---|---|---|---|---|---|
| 1 | Fix Value | | | | | |
| 2 | Q15 | 4096 | 256 | 16 | 1 | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | 778 | 0.189951 | 3.039231 | 0.627702 | 10.04324 | 030A |
| 6 | 29313 | 7.156503 | 2.504061 | 8.064987 | 1.039797 | 7281 |
| 7 | 27242 | 6.659761 | 10.41218 | 6.594986 | 9.519785 | 6A69 |
| 8 | 6268 | 1.530270 | 8.484329 | 7.749269 | 11.98831 | 187B |
| 9 | | | | | | |

**Figure 4. Lotus Command Structure**

## Saving the Lotus 1-2-3 File as an ASCII Text File

You can save a spreadsheet range as an ASCII file by using the Lotus PRINT utility. If you select the FILE option under PRINT, with the left column margin set to zero, Lotus will write the columns to a file with the required <CR><LF> ending. Once this is written out, you must edit out the blank lines between pages so that the file content is continuous. You must also ensure that the data in the file is fully left-justified and starts at the top of the file (no blank lines). You can do this by editing the file with a text editor.

## Running Your Program with Data Logging

After completion of the preceding steps, you can execute your program with data logging enabled. Name the input file INPUT.DAT, and the output file OUTPUT.DAT. Make certain the created LOTUS file is in the correct directory and is referenced by the correct file name; the DOS file name is the same as the data log file name."

Since each development tool has a unique procedure for enabling data logging, it is assumed that you know how to initialize file I/O. The data logging feature of the TMS320C25 SWDS is documented on page 3-29 of Reference [2], and the TMS320C30 Simulator on page 3-14 of Reference [3].

When your program is executed, the disk drive light will start to blink. With the TMS320C25 SWDS, each disk access is equal to 64 samples being written and/or read, while on the TMS320 simulators, there is one disk access for each sample. To control the number of samples written to and/or read from the file, you can either

1. Manually count the number of times the drive light flashes,
2. In the case of the TMS320C25 SWDS, use program control techniques, such as break points with count values (see page 4-108 of Reference [2], or
3. In the case of the TMS320C30 simulator, use the LOOP command (see page 5-103 of Reference [3].

When the above process is finished, there will be a new file in the working directory named OUTPUT.DAT (the name previously given to your file). This file contains a listing of *N* ASCII HEX character strings in which each line represents one output time sample, y(n). In the case of the first- and second-generation development tools, the file structure is identical to the input file structure: four HEX values represent a 16-bit field in which the sign bit is left-justified. However, the TMS320C30 Simulator outputs, and also requires for input, a form similar to the HEX syntax used in the C programming language. This is a 10-character HEX field with 0x as the first two characters. The impulse value shown in Figure 2 would be written as 0x7FF00000 for the TMS320C30 Simulator. You can generate an input file form, using Lotus 1-2-3, by simply adding two columns: a column containing the 0x prefix placed just before the calculated four-digit HEX field, and a column containing 0000 just after it.

## Plotting the Output Data

Several software programs can easily read and plot the output file as a continuous time signal or frequency domain; Mat Lab, DAPiSP, ILS, Math Cad, and Hypersignal. For further information on any of these products, contact the companies shown in Appendix A. This report shows how Hyperceptions' Hypersignal package is used to debug DSP algorithms.

The Hypersignal program can acquire and display all types of TMS320 files. This permits viewing numerical file data (input and output) in both time and frequency representation. Hypersignal offers an extensive list of DSP utilities, such as

- Waveform display/edit
- FFT generation
- FIR and IIR filter construction and code generation (assembly and C)
- Convolution
- LPC autocorrelation
- Recursive filtering for IIR filter types

- Generation of user-defined difference equations (which can generate files for use as input to any of the TMS320 development tools)
- Digital Oscilloscope

Hypersignal has several other functions for analyzing data files and filters in the frequency domain with utilities for creating or displaying

- Filter/file magnitude display (both log or linear)
- Filter phase display
- 3-D and 2-D frequency vs time vs amplitude-spectrogram display
- Inverse FFT function
- Filter pole-zero plot (both in s and z domains)
- Power spectrum generation

Hypersignal's powerful functions permit the evaluation of DSP tasks. For a first time user, they can prove to be extremely helpful in establishing a base line knowledge of DSP.

## Algorithms of High Complexity

Packages such as Hyperception's make DSP algorithm development manageable, even with N second-order cascaded sections. In Figure 3, there was one second-order section. If anything were to go wrong, it would do so within this section. How would data logging help if you have several cascade sections?

This can be answered by drawing an analogy between debugging a fourth-order analog system, which uses op-amps, and the equivalent DSP system implemented with four cascaded second-order IIR sections.

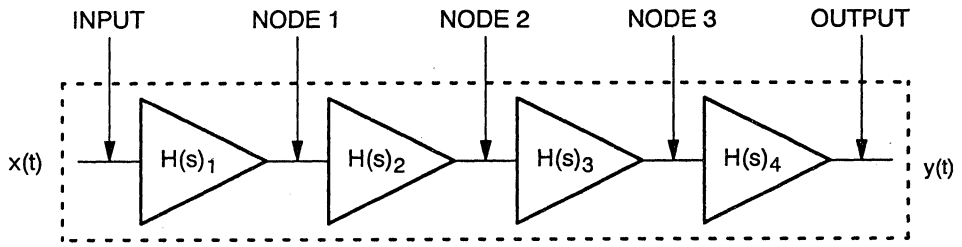



**Figure 5. Four Op-Amp Block Diagram**

Using traditional debugging techniques, i.e., an oscilloscope and function generator, you can examine the output versus the input on a stage-by-stage basis, correcting or adjusting each stage one-at-a-time. This process starts at node 1 and continues through to the output. When the system yields a satisfactory response for a given input condition

(not clipping, and amplifying at expected levels), use a spectrum analyzer to verify total frequency response. If the frequency response was not as expected, you can then examine each stage individually and adjust pole-zero placement to obtain the desired response.

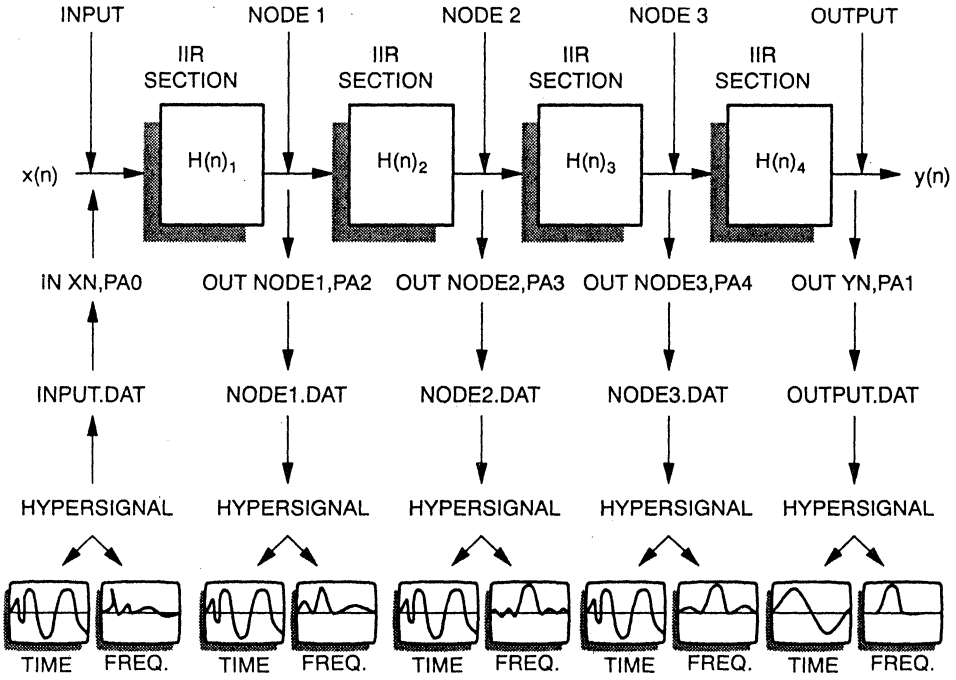



**Figure 6. Four Second-Order IIR Structures**

Figure 6 shows the same system as Figure 5 but uses four second-order IIR structures (a direct form II realization). When you use straight line code, it is a simple task to write a time sample to the DOS file by adding an OUT instruction. You can examine the feedback and/or feed forward signal within the IIR section as well. In addition to the obvious benefits of probing literally anywhere within the algorithm or system, there are some not-so-obvious benefits.

## The Advantages of Data Logging:

1. You can assume any sample frequency. Sample frequency in a hardwarebased DSP system is a function of the A/D, the D/A, and the clock cycle of the DSP. With data logging, you can arbitrarily assign any frequency to the data samples within the files and can further assume any operating frequency for the DSP. It is therefore possible to specify devices with speeds in excess of any presently available speed if your algorithm so requires.

2. You can specify any input condition. If you are doing a modem design, you can use a real-time data sampler to acquire a REAL signal to use as an input file. It is also possible to use a numerically generated input signal supplied by Lotus or any other software system/utility such as Hypersignal, HLL programs, math packages, etc.

3. You can probe your system without having to observe any location restrictions. In hardware systems, you are restricted to available pins. With DSP code, an OUT instruction can be put anywhere.

4. You must use use a scope probe with analog systems, thus adding resistance and capacitance to the signal being examined. Data logging is a perfect observation utility, since it places no load on the signal.

5. You can examine the input and output signals with any level of desired granularity. If you intend to use a 12-bit A/D, you can examine the signal at 16 bits, then truncate the data to 12 bits and compare results. If you can get by with 10 or even 8 bits of granularity, you will reduce system cost.

6. You can print your results using plotting packages such as Hypersignal. Results can be printed for both frequency and time, thus providing a greater level of documentation.

7. You can archive input and output files as part of your total documentation package.

8. You can't get burned; there are no soldering irons involved.

## Conclusion

DSP-based systems using data logging techniques demonstrate improved quality and shorter time to market. Using the TMS320 simulators and SWDS products in conjunction with graphic/data acquisition software packages, you can write and debug a large portion of an algorithm long before silicon or target platforms are available.

## References

[1] *First-Generation User's Guide* (literature number SPRU013B), Texas Instruments, 1989.

[2] *TMS320 Family Simulator User's Guide* (literature number SPRU009B), Texas Instruments, 1988.

[3] *The TMS320C30 Simulator User's Guide* (literature number SPRU017), Texas Instruments, 1988.

# Appendix A

## Software Package Sources

DADiSP
DSP Development Corp
One Kendall Square, Cambridge, MA 02139
(617) 577-1133

Hypersignal
Hyperception
9550 Skillman-LB125 Dallas, TX 75243
(214) 343-8525

ILS
STI Signal Technology Inc
5951 Encina Road, Goleta, CA 93117
(805) 683-3771

Lotus 1-2-3
Lotus Development Group
55 Wheeler St. Cambridge, MA 02138
(617) 492-7171

Math CAD
MathSoft
One Kendall Sq., Cambridge, MA 02139
(617) 577-1017

MATLAB
The Math Works Inc
South Natick, MA 01760
(508) 653-1415

# TMS320 Bibliography

Since the TMS32010 was disclosed in 1982, the TMS320 family has received an ever-increasing amount of recognition. The number of outside parties contributing to the extensive development support offered by Texas Instruments is rapidly growing. Many technical articles are being written about TMS320 applications in the field of digital signal processing.

The following articles and papers have been published since 1982 regarding the Texas Instruments TMS320 Digital Signal Processors. Readers who are interested in gaining further information about these processors and their applications may obtain copies of these articles/papers from their local or university library.

The articles are broken down into 12 different application categories. Articles in each category are in reverse chronological order (most recent first). Articles having the same publication date are shown in alphabetical order by authors name.

The application categories are:

1) General Purpose DSP
2) Graphics/Imaging
3) Instrumentation
4) Voice/Speech
5) Control
6) Military
7) Telecommunications
8) Automotive
9) Consumer
10) Industrial
11) Medical
12) Development Support


***General Purpose DSP***

1) R. Chassaing, "A Senior Project Course in Digital Signal Processing with the TMS320," *IEEE Transactions on Education*, USA, Volume 32, Number 2, pages 139–145, May 1989.
2) P.E. Papamichalis, C.S. Burrus, "Conversion of Digit-Reversed to Bit-Reversed Order in FFT Algorithms," *Proceedings of ICASSP 89*, USA, pages 984–987, May 1989.
3) P.E. Papamichalis, "Application, Progress and Trends in Digital Signal Processing," *Proceedings of Mikroelektronik Conference*, Baden-Baden, March 1989.
4) R. Chassaing, "Adaptive Filtering with the TMS320C25 Digital Signal Processor," *Proceedings of 1989 ASEE Conference,* USA, pages 215–217, 1989.
5) P.E. Papamichalis, R. Simar, Jr., "The TMS320C30 Floating-Point Digital Signal Processor," *IEEE Micro Magazine*, USA, pages 13–29, December 1988.
6) K. Rogers, "The Real-Time Thing (Digital Signal Controller)," *Electronic Engineering Times*, USA, Number 506, page 85, October 1988.
7) P.E. Papamichalis, "Impact of DSP Devices on Fast Algorithms," *Proceedings of the 1988 IEEE DSP Workshop*, USA, September 1989.

8) G. Umamaheswari, C. Eswaran, A. Jhunjhunwala, "Signal Processing with a Dual-Bank Memory," *Microprocessor Microsystems,* Great Britain, Volume 12, Number 4, pages 206–210, May 1988.
9) G. Castellini, P. Luigi, E. Liani, L. Pierucci, F. Pirri, S. Rocchi, "A Multiprocessor Structure Based on Commercial DSP," *Proceedings of ICASSP 88*, USA, Volume V, page 2096, April 1988.
10) M.R. Civanlar, R.A. Nobakht, "Optimal Pulse Shape Design Using Projections onto Convex Sets," *Proceedings of ICASSP 88,* USA, Volume D, p. 1874, April 1988.
11) L.J. Eriksson, M.C. Allie, C.D. Bremigan, R.A. Greiner, "Active Noise Control Using Adaptive Digital Signal Processing," *Proceedings of ICASSP 88*, USA, Volume A, page 2594, April 1988.
12) G. Mirchandani, D.D. Ogden, "Experiments in Partitioning and Scheduling Signal Processing Algorithms for Parallel Processing," *Proceedings of ICASSP 88*, USA, Volume D, page 1690, April 1988.
13) P. Papamichalis, "FFT Implementation on the TMS320C30," *Proceedings of ICASSP 88,* USA, Volume D, page 1399, April 1988.
14) A.C. Rotger-Mora, "An N-Dimensional SIMD Ring Architecture for Implementing Very Large Order Adaptive Digital Filters," *Proceedings of ICASSP 88*, USA, Volume V, page 2140, April 1988.
15) J. Santos, J. Parera, M. Veiga, "A Hypercube Multiprocessor for Digital Signal Processing Algorithm Research," *Proceedings of ICASSP 88*, USA, Volume D, page 1698, April 1988.
16) R. Simar, A. Davis, "The Application of High-Level Languages to Single-Chip Digital Signal Processors," *Proceedings of ICASSP 88*, USA, Volume D, page 1678, April 1988.
17) K. Bala, "Running on Embedded Power. (Dedicated 32-Bit Microprocessors Used in New Microcontrollers)(Technology Trends: Microprocessors and Peripherals)," *Electronic Engineering Times*, USA, Number 478, page 34, March 1988.
18) J. Cooper, "DSP Chip Speeds VME Transfer," *ESD: Electronic Systems Design*, USA, Volume 18, Number 3, pages 47,48,50,51, March 1988.
19) L. Vieira de Sa, F. Perdigao, "A Microprocessing System for the TMS32020," *Microprocessing Microprogramming,* Netherlands, Volume 23, Number 1–5, pages 221–225, March 1988.
20) G. Wade, "Offset FFT and Its Implementation on the TMS320C25 Processor," *Microprocessing Microsystems*, Great Britain, Volume 12, Number 2, pages 76–82, March 1988.
21) R. Chassaing, "Digital Broadband Noise Synthesis by Multirate Filtering Using the TMS320C25," *Proceedings of 1988 ASEE Conference*, USA, pages 394–397, 1988.
22) R. Chassaing, "A Senior Project Course on Applications in Digital Signal Processing with the TMS320," *Proceedings of 1988 ASEE Conference*, USA, pages 354–359, 1988.
23) L.N. Bohs, R.C. Barr, "Real-Time Adaptive Sampling with the Fan Method," *Proceedings of the Ninth Annual Conference of the IEEE Engineering in Medicine and Biology Society,* USA, Volume 4, pages 1850–1851, November 1987.
24) T. Kimura, Y. Inabe, T. Hayashi, K. Uchimura, K. Hamazato, "Dual-Chip SLIC Using VLSI Technology," *Conference Record of GLOBECOM Tokyo '87*, Volume 3, pages 1766–1770, November 1987.

25) W.S. Gass, R.T. Tarrant, T. Richard, B.I. Pawate, M. Gammel, P.K. Rajasekaran, R.H. Wiggins, C.D. Covington, "Multiple Digital Signal Processor Environment for Intelligent Signal Processing," *Proceedings of the IEEE*, USA, Volume 75, Number 9, pages 1246–1259, September 1987.

26) L. Johnson, R. Simar, Jr., "A High Speed Floating Point DSP," *Conference Record of MIDCON/87*, USA, pages 396–399, September 1987.

27) K.S. Lin, G.A. Frantz, R. Simar, Jr., "The TMS320 Family of Digital Signal Processors," *Proceedings of the IEEE*, USA, Volume 75, Number 9, pages 1143–1159, September 1987.

28) S.L. Martin, "Wave of Advances Carry DSPs To New Horizons. (Digital Signal Processing)," *Computer Design*, USA, Volume 26, Number 17, pages 69–82, September 1987.

29) C. Murphy, A. Coats, J. Conway, P. Colditz, P. Rolfe, "Doppler Ultrasound Signal Analysis Based on the TMS320 Signal Processor," *27th Annual Scientific Meeting of the Biological Engineering Society*, Great Britain, Volume 10, Number 2, pages 127–129, September 1987.

30) G.S. Kang, L.J. Fransen, "Experimentation With An Adaptive Noise-Cancellation Filter," *IEEE Transactions on Circuits and Systems*, USA, Volume CAS-34, Number 7, pages 753–758, July 1987.

31) R. Chassaing, "Applications in Digital Signal Processing with the TMS320 Digital Signal Processor in an Undergraduate Laboratory," *Proceedings of the 1987 ASEE Annual Conference*, USA, Volume 3, pages 1320–1324, June 1987.

32) D.W. Horning, "An Undergraduate Digital Signal Processing Laboratory," *Proceedings of the 1987 ASEE Annual Conference*, USA, Volume 3, pages 1015–1020, June 1987.

33) D. Locke, "Digitising In The Gigahertz Range," *IEE Colloguium on Advanced A/D Conversion Techniques*, Great Britain, Digest Number 48, 10/1–4, April 1987.

34) S. Orui, M. Ara, Y. Orino, E. Sazuki, H. Makino, "Realization of IIR Filter using the TMS320," *Resident Reports of Kogakuin University*, Japan, Number 62, pages 195–204, April 1987.

35) R. Simar, T. Leigh, P. Koeppen, J. Leach, J. Potts, D. Blalock, "A 40 MFLOPS Digital Signal Processor: The First Supercomputer on a Chip," *Proceedings of ICASSP 87*, USA, Catalog Number 87CH2396–0, Volume 1, pages 535–538, April 1987.

36) R. Simar, "TMS320: Texas Instruments Family of Digital Signal Processors," *Proceedings of SPEECH TECH 87*, USA, pages 42–47, April 1987.

37) G.Y. Tang, B.K. Lien, "A Multiple Microprocessor System For General DSP Operation," *Proceedings of ICASSP 87*, USA, Catalog Number 87CH2396–0, Volume 2, pages 1047–1050, April 1987.

38) L. Vieira de Sa, "Second MicroProcessor Enhances TMS32020 System," *EDN: Electronic Design News*, USA, Volume 32, Number 9, pages 230–232, April 1987.

39) T.J. Moir, T.G. Vishwanath, D.R. Campbell, "Real-Time Self-Tuning Deconvolution Filter and Smoother," *International Journal of Control*, Great Britain, Volume 45, Number 3, pages 969–985, March 1987

40) R. Simar, M. Hames, "CMOS DSP Packs Punch of a Supercomputer," *EDN: Electronic Design News*, USA, Volume 35, Number 7, pages 103–106, March 1987.

41) S. Sridharan, "On Improving the Performance of Digital Filters Designed Using the TMS32010 Signal Processor," *Journal of Electrical and Electronic Engineers of Australia, Australia*, Volume 7, Number 1, pages 80–82, March 1987.
42) R. McCammon, "Software Routine Probes TMS32010 Code," *EDN: Electronic Design News,* USA, Volume 32, Number 4, pages 200,202, February 1987.
43) J. Prado, R. Alcantara, "A Fast Square-Rooting Algorithm Using A Digital Signal Processor," *Proceedings of IEEE*, USA, Volume 75, Number 2, pages 262–264, February 1987.
44) T.G. Vishwanath, D.R. Camppbell, T.J. Moir, "Real-Time Implementation Using a TMS32010 Microprocessor," *IEEE Transactions on Industrial Electronics*, USA, Volume 1E-34, Number 1, pages 115–118, February 1987.
45) R Chassaing, "Applications in Digital Signal Processing with the TMS320 Digital Signal Processor in an Undergraduate Laboratory," *Proceedings of 1987 ASEE Conference*, USA, pages 1320–1324, 1987.
46) R.M. Sovacool, "EPROM Enhances TMS32020 Mu C's Memory," *EDN: Electronic Design News,* USA, Volume 32, Number 1, page 231, 1987.
47) F. Kocsis, F. Marx, "Fast DFT Modules For The TMS32010 Digital Signal Processor," *Meres and Automation, Hungary*, Volume 35, Number 1, pages 6–11, 1987.
48) Y.V.V.S. Murty, W.J. Smolinski, "Digital Filters for Power System Relaying," *International Journal of Energy Systems*, USA, Volume 7, Number 3, pages 125–129, 1987.
49) S. Wang, "The TMS32010 High Speed Processor and Its Applications," *Mini-Micro Systems*, China, Volume 8, Number 3, pages 24–32, 1987.
50) G.A. Frantz, K.S. Lin, J.B. Reimer, J. Bradley, "The Texas Instruments TMS320C25 Digital Signal Microcomputer," *IEEE Microelectronics,* USA, Volume 6, Number 6, pages 10–28, December 1986.
51) P. Renard, "A/D Converters: The Advantage of a Mixture of Techniques," *Mesures, France*, Volume 51, Number 16, pages 80–81, December 1986.
52) M. Ara, E. Suzuki, "Design of Real Time Filter Using DSP," *Resident Reports of Kogakuin University*, Japan, Number 61, pages 115–127 October 1986.
53) J. Reidy, "Connection of a 12-Bit A/D Converter to Fast DSPs," *Electronik, Germany*, Volume 35, Number 22, pages 132–134, October 1986.
54) G.R. Steber, "Implementation of Adaptive Filters on the TMS32010 DSP Microcomputer," *Proceedings of IECON 86*, Catalog Number 86CH2334–1, Volume 2, pages 653–656, September/October 1986.
55) D. Collins, M.A. Rahman, "Digital Filter Design Using The TMS320 Digital Signal Processor," *Proceedings of EUSIPCO-86*, Volume 1pages 163–166, September 1986.
56) R. Simar, Jr., J.B. Reimer, "The TMS320C25: A 100 ns CMOS VLSI Digital Signal Processor," *1986 Workshop on Applications of Signal Processing to Audio and Acoustics*, September 1986.
57) J. Dudas, A. Stipkovits, E. Simonyi, "On The recursive Momentary Discrete Fourier Transform," *Proceedings of EUSIPCO-86*, Volume 1, pages 303–306, September 1986.
58) E. Feder, "Digital Signal Processor – General Purpose or Dedicated?," Electronics Industry, France, Number 111, pages 74–82, September 1986.
59) K. Herberger, "The Use of Signal Processors For Simulating Data Circuits," *Proceedings of EUSIPCO-86*, Volume 2, pages 1109–1112, September 1986.

60) K. Kassapoglou, P. Hulliger, "Implementation of Recursive Least Squares Identification Algorithm on The TMS320," *Proceedings of EUSIPCO-86*, Volume 2, pages 1263–1266, September 1986.
61) G. Lucioni, "General Processor Application; CAD Tool For Filter Design," *Proceedings of EUSIPCO-86,* Volume 2, pages 1335–1338, September 1986.
62) R. Schapery, "A 10-MIP Digital Signal Processor From Texas Instruments," *Conference Record of Midcon 86,* USA, 1/2/1–11, September 1986.
63) "DSP Microprocessors," *Inf. Elettronica*, Italy, Volume 14, Number 7–8, pages 21–28,
64) R.L. Barnes, S.H. Ardalan, "Multiprocessor Architecture For Implementing Adaptive Digital Filters," *Conference Record of ICC-86*, Catalog Number 86CH2314–3, Volume 1, pages 180–185, June 1986.
65) A.D.E. Brown, "EPROMS Simplify TMS32010 Memory System," *EDN: Electronic Design News*, USA, Volume 31, Number 13, page 230, June 1986.
66) T. Kolehamainen, T. Saramaki, M. Renfors, Y. Neuvo, "Signal Processor Implementation of Computationally Efficient FIR Filter Structures – Theory and Practice," *2nd Nordic Symposium on VLSI in Computers and Communications,* 10 pages, June 1986.
67) T.G. Marshall Jr.,"Transform Methods For Developing Parallel Algorithms For Cyclic-Block Signal Processing," *Conference Record of ICC-86*, Catalog Number 86CH2314–3, Volume 1, pages 288–294, June 1986.
68) S. Abiko, M. Hashizume, Y. Matsushita, K. Shinozaki, T. Takamizawa, C. Erskine, S. Magar, "Architecture and Applications of a 100-ns CMOS VLSI Digital Signal Processor," *Proceedings of ICASSP 86*, USA, Catalog Number 86CH2243–4, Volume 1, pages 393–396., April 1986.
69) T.P. Barnwell, "Algorithm Development and Multiprocessing Issues for DSP Chips," *Proceedings of Speech Technology 86,* April 1986.
70) W. Gass, "TMS32020 – The Quick and Easy Solution to DSP Problems," *Proceedings of Speech Technology 86*, April 1986.
71) M. Hashizume, S. Abiko, Y. Matsushita, K. Shinozaki,T. Takamizawa, S. Magar, J. Reimer, "A 100-ns CMOS VLSI Digital Signal Processor Using Double Level Metal Structure," *Semiconductor Group 1986 Technical Meeting,* April 1986.
72) R.E. Morley, A.M. Engebretson, and J.G. Trotta, "A Multiprocessor Digital Signal Processing System for Real-Time Audio Applications," *IEEE Transactions on Acoustics, Speech and Signal Processing*, USA, Volume ASSP-34, Number 2, April 1986.
73) S.G. Smith, A. Fitzgerald, P.B. Denyer, D. Renshaw, N.P. Wooten, R. Creasey, "A Comparison of Micro-DSP And Silicon Compiler Implementations of a Polyphase-Network Filter Bank," *Proceedings of ICASSP 86*, USA, Catalog Number 86CH2243–4, Volume 3, pages 2207–2210, April 1986.
74) J. Reimer, M. Hames, "Next Generation CMOS Chip Stakes High-Performance Claim on 10-MIPS DSP Operations," *Electronic Design*, USA, Volume 34, Number 8, pages 141–146, April 1986.
75) W.W. Smith, "Playing to Win: Product Development with the TMS320 Chip," *Speech Technology Magazine*, March/April 1986.
76) D. Essig, C. Erskine, E. Caudel, and S. Magar, "A Second-Generation Digital Signal Processor," *IEEE Journal of Solid-State Circuits*, USA, Volume SC-21, Number 1, pages 86–91, February 1986.

77) W.K. Anakwa, T.L. Stewart, "TMS320 Microprocessor-Based System For Signal Processing," *Proceedings of the ISMM International Symposium*, pages 64–65, February 1986.
78) M. Omenzetter, "Universal Signal Processors Offers High Data Throughput," *Electronik*, Germany, Volume 35, Number 4, pages 71–77, February 1986.
79) P.F. Regamey, "Matched Filtering Using a Signal Microprocessor TMS320," *Mitt. AGEN*, Switzerland, Number 42, pages 31–35, February 1986.
80) "TI Set To Show 2nd-Generation DSP," *Electronics*, USA, pages 23–24, February 3, 1986.
81) "TI Preps CMOS Versions of Signal-Processor Chips," *Electronics Engineering Times*, USA, page 6, February 3, 1986.
82) D. Wilson, "Digital Signal Processing Moves on Chip," *Digital Design*, USA, Volume 16, Number 2, pages 33–34, February 1986.
83) "TI Chip Heads for Fast Lane of Digital Signal Processing," *Electronics*, USA, page 9, January 27, 1986.
84) R.D. Campbell and S.R. McGeoch, "The TMS32010 Digital Signal Processor – An Educational Viewpoint," *International Journal for Electrical Engineering Education*, Great Britain, Volume 23, Number 1, pages 21–31, January 1986.
85) P. Eckelman, "The Cascadable Signal Processor For Digital Signal Processing," *Electronics Industry*, Germany, Volume 17, Number 10, pages 26–27, 1986.
86) R. Cook, "Digital Signal Processors," *High Technology*, USA, Volume 5, Number 10, pages 25–30, October 1985.
87) C.F. Howard, "A High-Level Approach to Digital Processing Design," *Proceedings of MILCOMP/85*, USA, October 1985.
88) H.E. Lee, "Versatile Data-Acquisition System Based on the Commodore C-64/C-128 Microcomputer," *Proceedings of the Symposium of Northeastern Accelerator Personnel*, USA, Volume 57, Number 5, pages 983–985, October 1985.
89) N.K. Riedel, D.A. McAninch, C. Fisher, and N.B. Goldstein, "A Signal Processing Implementation for an IBM PC-Based Workstation," *IEEE Micro*, USA, Volume 5, Number 5, pages 52–67, October 1985.
90) K.E. Marrin, "VLSI and Software Move DSP Into Mainstream," *Computer Design*, USA, Volume 24, Number 9, pages 69–72, September 1985.
91) "Signal Processor ICs: Highly Integrated ICs Making DSP More Attractive," *Electronics Engineering Times*, USA, pages 37–38, September 2, 1985.
92) K.E. Marrin, "VLSI and Software Move DSP Techniques into Mainstream," *Computer Design*, USA, September 1985.
93) "High-Speed Four-Channel Input Board," *Electronics Weekly*, USA, Number 1277, p. 31, July 24, 1985.
94) "4-Channel Analog-Input Board Puts Signal-Processing on VMF Bus," *EDN: Electronic Design News*, USA, Volume 30, Number 17, page 74, July 1985.
95) R.H. Cushman, "Third-Generation DSPs Put Advanced Functions On-Chip," *EDN: Electronic Design News*, USA, July 1985.
96) W.W. Smith, Jr., "Agile Development System, Running on PCs, Builds TMS320-Based FIR Filter," *Electronic Design*, USA, Volume 33, Number 13, pages 129–138, June 6, 1985.

97) S. Magar, S.J. Robertson, and W. Gass, "Interface Arrangement Suits Digital Processor to Multiprocessing," *Electronic Design*, USA, Volume 33, Number 5, pages 189–198, March 7, 1985.
98) G. Kropp, "Signal Processor Offers Multiprocessor Capability," *Elektronik*, Germany, Volume 34, Number 6, pages 53–58, March 1985.
99) S. Magar, D. Essig, E. Caudel, S. Marshall and R. Peters, "An NMOS Digital Signal Processor with Multiprocessing Capability," *Digest of IEEE International Solid-State Circuits Conference*, USA, February 1985.
100) "TI 'Shiva' Chip Outlined," *Electronics Engineering Times*, USA, page 15, February 18, 1985.
101) S. Magar, E. Caudel, D. Essig, and C. Erskine, "Digital Signal Processor Borrows from P to Step up Performance, *Electronic Design*, USA, Volume 33, Number 4, pages 175–184, February 21, 1985.
102) C. Erskine, S. Magar, E. Caudel, D. Essig, and A. Levinspuhl, "A Second-Generation Digital Signal Processor TMS32020: Architecture and Applications," *Traitement de Signal*, France, Volume 2, Number 1, pages 79–83, January–March 1985.
103) S. Baker, "TI 'Shiva' Chip Outlined," *Electronic Engineering Times*, USA, Number 317, page 15, February 1985.
104) S. Baker, "Silicon Bits," *Electronic Engineering Times*, USA, Number 316, page 42, February 1985.
105) H. Bryce, "Board Arrives For Digital Signal Processing on the VMEbus," *Electronic Design*, USA, Volume 33, Number 2, page 266, 1985.
106) K. Marrin, "VME-Compatible DSP System Incorporates TMS320 Chip," *EDN: Electronic Design News*, USA, Volume 30, Number 2, page 122, January 1985.
107) C. Erskine and S. Magar, "Architecture and Applications of A Second-Generation Digital Signal Processor," *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing,* USA, 1985.
108) D.P. Morgan and H.F. Silverman, "An Investigation into the Efficiency of a Parallel TMS320 Architecture: DFT and Speech Filterbank Applications," *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing,* USA, Volume 4, pages 1601–1604, 1985.
109) P. Harold, "VME Bus Meeting Sparks Change in Standard, New Products," *EDN: Electronic Design News*, USA, Volume 29, Number 26, page 18, December 1984.
110) W. Loges, "A Code Generator Sets up the Automatic Controller Program for the TMS320," *Elektronik*, Germany, Volume 33, Number 22, pages 154–158, November 1984.
111) H. Volkers, "Fast Fourier Transforms with the TMS320 as Coprocessor," *Elektronik*, Germany, Volume 33, Number 23, pages 109–112, November 1984.
112) Keun-Ho Ryoo, "On the Recent Digital Signal Processors," *Journal of South Korean Institute of Electrical Engineering,* South Korea, Volume 33, Number 9, pages 540–549, September 1984.
113) D. Wilson, "Editor's Comment," *Digital Design*, USA, Volume 14, Number 9, page 14, September 1984.
114) "Signal Processors Will Squeeze Into One Chip, Says TI's French," *Electronics*, USA, Volume 57, Number 9, pages 14,20, May 1984.

115) S. Mehrgardt, "32-Bit Processor Produces Analog Signals," *Elektronik*, Germany, Volume 33, Number 7, pages 77–82, April 1984.
116) S. Magar, "Signal Processing Chips Invite Design Comparisons," *Computer Design*, USA, Volume 23, Number 4, pages 179–186, April 1984.
117) S. Mehrgardt, "General-Purpose Processor System for Digital Signal Processing," *Elektronik*, Germany, Volume 33, Number 3, pages 49–53, February 1984.
118) T. Durham, "Chips: Familiarity Breeds Approval," *Computing*, Great Britain, page 26, January 1984.
119) J. Bradley and P. Ehlig, "Applications of the TMS32010 Digital Signal Porcessor and Their Tradeoffs," *Midcon/84 Electronic Show and Convention*, USA, 1984.
120) J. Bradley and P. Ehlig, "Tradeoffs in the Use of the TMS32010 as a Digital Signal Processing Element," *Wescon/84 Conference Record*, USA, 1984.
121) E. Fernandez, "Comparison and Evaluation of 32-Bit Microprocessors," *Mini/Micro Southeast Computer Conference and Exhibition*, USA, 1984.
122) D. Garcia, "Multiprocessing with the TMS32010," *Wescon/84 Conference Record*, USA, 1984.
123) S. Magar, "Architecture and Applications of a Programmable Monolithic Digital Signal Processor – A Tutorial Review," *Proceedings of IEEE International Symposium on Circuits and Systems*, USA, 1984.
124) D. Quarmby (Editor), "Signal Processor Chips," *Granada*, England 1984.
125) R. Steves, "A Signal Processor with Distributed Control and Multidimensional Scalability," *Proceedings of IEEE National Aerospace and Electronics Conference*, USA, 1984.
126) V. Vagarshakyan and L. Gustin, "On A Single Class of Continuous Systems – A Solution to the Problem on the Diagnosis of Output Signal Characteristics Recognition Procedures," *IZV. AKAD. NAUK ARM. SSR, SER. TEKH. NAUK, USSR*, Volume 37, Number 3, pages 22–27, 1984.
127) J. So, "TMS320 – A Step Forward in Digital Signal Processing," *Microprocessors and Microsystems*, Great Britain, Volume 7, Number 10, pages 451–460, December 1983.
128) J. Elder and S. Magar, "Single-Chip Approach to Digital Signal Processing," *Wescon/83 Electronic Show and Convention*, USA, November 1983.
129) M. Malcangi, "VLSI Technology for Signal Processing. III," *Elettronica Oggi*, Italy, Number 11, pages 129–138, November 1983.
130) P. Strzelcki, "Digital Filtering," *Systems International*, Great Britain, Volume 11, Number 11, pages 116–117, November 1983.
131) W. Loges, "Digital Controls Using Signal Processors," *Elektronik, Germany*, Volume 32, Number 19, pages 51–54, September 1983.
132) "TI's Voice Chip Makes Debut," *Computerworld*, USA, Volume 17, Number 15, page 91, April 1983.
133) L. Adams, "TMS320 Family 16/32-Bit Digital Signal Processor, An Architecture for Breaking Performance Barriers," *Mini/Micro West 1983 Computer Conference and Exhibition*, USA, 1983.
134) R. Blasco, "Floating-Point Digital Signal Processing Using a Fixed-Point Processor," *Southcon/83 Electronics Show and Convention*, USA, 1983.

135) R. Dratch, "A Practical Approach to Digital Signal Processing Using an Innovative Digital Microcomputer in Advanced Applications," *Electro '83 Electronics Show and Convention*, USA, 1983.
136) C. Erskine, "New VLSI Co-Processors Increase System Throughput," *Mini/Micro Midwest Conference Record*, USA, 1983.
137) L. Kaplan, "Flexible Single Chip Solution Paves Way for Low Cost DSP," *Northcon/83 Electronics Show and Convention*, USA, 1983.
138) L. Kaplan, "The TMS32010: A New Approach to Digital Signal Processing," *Electro '83 Electronics Show and Convention*, USA, 1983.
139) S. Mehrgardt, "Signal Processing with a Fast Microcomputer System," *Proceedings of EUSIPCO-83 Second European Signal Processing Conference*, Netherlands, 1983.
140) L. Morris, "A Tale of Two Architectures: TI TMS 320 SPC VS. DEC Micro/J–11," *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, USA, 1983.
141) L. Pagnucco and D. Garcia, "A 16/32 Bit Architecture for Signal Processing," *Mini/Micro West 1983 Computer Conference and Exhibition*, USA, 1983.
142) J. Potts, "A Versatile High Performance Digital Signal Processor," *Ohmcon/83 Conference Record*, USA, 1983.
143) J. Potts, "New 16/32-Bit Microcomputer Offers 200-ns Performance," *Northcon/83 Electronics Show and Convention*, USA, 1983.
144) R. Simar, "Performance of Harvard Architecture in TMS320," *Mini/Micro West 1983 Computer Conference and Exhibition*, USA, 1983.
145) K. McDonough, E. Caudel, S. Magar, and A. Leigh, "Microcomputer with 32-Bit Arithmetic Does High-Precision Number Crunching," *Electronics*, USA, Volume 55, Number 4, pages 105–110, February 1982.
146) K. McDonough and S. Magar, "A Single Chip Microcomputer Architecture Optimized for Signal Processing," *Electro/82 Conference Record*, USA, 1982.
147) L. Kaplan, "Signal Processing with the TMS320 Family," *Midcon/82 Conference Record*, USA, 1982.
148) S. Magar, "Trends in Digital Signal Processing Architectures," *Wescon/82 Conference Record*, USA, 1982.

***Graphics/Imaging***

1) J.A. Lindberg, "Color Cell Compression Shrinks NTSC Images," *ESD: Electronic Systems Design Magazine*, USA, Volume 17, Number 10, pages 91–96, October 1987
2) S. Ganesan, "A Digitial Signal Processing Microprocessor Based Workstation For Myoelectric Signals," *Fifth International Conference on System Engineering*, USA, Catalog Number 87CH2480–2, pages 427–438, September 1987.
3) JU. Pokovny, O. Skoloud, "Digitisation of a Video Signal From a Television For a Microcomputer," *Sdelovaci Tech.*, Czechoslovakia, Volume 35, Number 6, pages 207–211, June 1987.
4) M.E. Bukaty, "A Vehicle Identification System For Surveillance Applications," *Topical Meeting on Machine Vision*. Technical Digest Series, USA, Volume 12, pages 106–109, March 1987.

5) K.N. Ngan, A.A. Kassim, H.S. Singh, "Parallel Image-Processing System Based on THe TMS32010 Digital Signal Processor," *IEE Proceedings in Electronics*, Great Britain, Volume 134, Number 2, pages 119–124, (March 1987.
6) K.N. Ngan, A.A. Kassim, H. Singh, "A TMS32010-Based Fast Parallel Vison Processor," *Proceedings of the International Workshop on Industrial Applications of Machine Vision and Machine Intelligence*, Catalog Number 87TH0166–9, pages 156–161, February 1987.
7) P. Bellamah, "Hardware-Software Increases Video Storage Capacity," *PC Week*, USA, Volume 4, Number 4, page 15, January 27 1987.
8) J.M. Younse, "Motion Detection Using the Statistical Properties of a Video Image," *Proceedings of SPIE International Society of Optical Engineering*, USA, Volume 697, pages 233–243, August 1986.
9) T. Gehrels, B.G. Marsden, R.S. McMillan, J.V. Scotti, "Astrometry With a Scanning CCD," *Astronomy Journal*, USA, Volume 91, Number 5, pages 1242–1248, May 1986.
10) S. Srinivasan, A.K. Jain, T.M. Chin, "Cosine Transform Block Codec For Images Using TMS32010," *IEEE International Symposium on Circuits and Systems*, USA, Catalog Number 86CH2255–8, Volume 1, pages 299–302, May 1986.
11) D.M. Holburn and I.D. Sommerville, "A High-Speed Image Processing System Using the TMS32010," *Software and Microsystems*, Great Britain, Volume 4, Number 5–6, pages 102–108, October–December 1985.
12) C. D. Crowell and R. Simar, "Digital Signal Processor Boosts Speed of Graphics Display Systems," *Electronic Design*, USA, Volume 33, Number 7, pages 205–209, March 1985.
13) J. Reimer and A. Lovrich, "Graphics with the TMS32020," *WESCON/85 Conference Record*, USA, 1985.
14) H. Megal and A. Heiman, "Image Coding System – A Single Processor Implementation," *MILCOM/85 IEEE Military Communications Conference Record*, USA, 1985.
15) G. Gaillat, "The CAPITAN Parallel Processor: 600 MIPS for Use in Real Time Imagery," *Traitement de Signal*, France, Volume 1, Number 1, pages 19–30, October–December 1984.

***Instrumentation***

1) G.R. Halsall, D.R. Burton, M.J. Lalor, C.A. Hobson, "A Novel Real-Time Opto-Electronic Profilometer Using FFT Processing," *Proceedings of ICASSP 89*, USA, pages 1634–1637, May 1989.
2) A.J. Pratt, R.E. Gander, B.R. Brandell, "Real-Time Median Frequency Estimator," *Proceedings of the Ninth Annual Conference of the IEEE Engineering in Medicine and Biology Society*, USA, Volume 4, pages 1840–1841, November 1987.
3) D.Y. Cheng, A. Gersho, "A Fast Codebook Search Algorithm For Nearest-Neighbor Pattern Matching," *Proceedings of ICASSP 86*, USA, Catalog Number 86CH2243–4, Vol 1, pages 265–268, April 1986.
4) Y. Chikada, M. Ishiguro, H. Hirabayashi, M. Morimoto, K. Morita, T. Kanazawa, H. Iwashita, K. Nakazima, S. Ishikawa, T. Takahashi, K. Handa, T.Kazuga, S. Okumura, T. Miyazawa, K. Miura, S. Nagasawa, "A Very Fast FFT Spectrum Analyzer For Radio

Astronomy," *Proceedings of ICASSP 86*, USA, Catalog Number 86CH2243–4, Volume 4, pages 2907–2910, April 1986.
5) R.C. Wittenberg, "Four Microprocessors Power Multifunction Analyzer," *Electronic Engineering Times*, USA, Number 306, page 30, November 1984.
6) D. Lee, T. Moran, and R. Crane, "Practical Considerations for Estimating Flaw Sizes from Ultrasonic Data," *Materials Evaluation*, Volume 42, Number 9, pages 1150–1158, August 1984.
7) S. Magar, R. Hester, and R. Simpson, "Signal-Processing c Builds FFT-Based Spectrum Analyzer," *Electronic Design*, USA, Volume 30, Number 17, pages 149–154, August 1982.

*Voice/Speech*

1) A. Aktas, H. Hoge, "Multi-DSP and VQ-ASIC Based Acoustic Front-End for Real-Time Speech Processing Tasks," *Proceedings of EUROSPEECH 89*, pages 586–589, September 1989.
2) D. Bergmann, D. Boillon, F. Bonifacio, R. Breitschadel, "Experimental Speech Input/Output System," *Proceedings of ICASSP 89*, USA, pages 1138–1141, May 1989.
3) J. DellaMorte, P.E. Papamichalis, "Full-Duplex Real-Time Implementation of the FED-STD-1015 LPC-10e Standard V.52 on the TMS320C25," *Proceedings of SPEECH TECH 89*, pages 218–221, May 1989.
4) B.I. Pawate, G.R. Doddington, "Implementation of a Hidden Markov Model-Based Layered Grammar Recognizer," *Proceedings of ICASSP 89*, USA, pages 801–804, May 1989.
5) P.E. Papamichalis, "High Quality Speech Coding: Some Recent Algorithms," *Proceedings of SPEECH TECH 89*, pages 329–333, May 1989.
6) J.C. Ventura, "Digital Audio Gain Control for Hearing Aids," *Proceedings of ICASSP 89*, USA, pages 2049–2052, May 1989.
7) N. Matsui, H. Ohasi, "DSP-Based Adaptive Control of a Brushless Motor," *IEEE Industry Application Society Annual Meeting*, USA, October 1988.
8) A. Albarello, R. Breitschaedel, A. Ciaramella, E. Lenormand, "Implementation of an Acoustic Front-End For Speech Recognition," *CSELT Technical Report*, Italy, Volume 16, Number 5, pages 455–459, August 1988.
9) D. Curl, "Voice Over Data Means More For Your Money," *Communications*, Great Britain, Volume 5, Number 8, pages 27–29, August 1988.
10) H. Hanselman, H. Henrichfreise, H. Hostmann, A. Schwarte, "Hardware/Software Environment for DSP-Based Multivariable Control," *12th. IMACS World Congress*, July 1988.
11) J.B. Attili, M. Savic, J.P. Campbell, Jr., "A TMS32020-Based Real Time Text-Independent, Automatic Speaker Verification System," *Proceedings of ICASSP 88*, USA, Volume S, page 599, April 1988.
12) D. Chase, A. Gersho, "Real-Time VQ Codebook Generation Hardware For Speech Processing," *Proceedings of ICASSP 88*, USA, April 1988.
13) T. Kohonen, K. Torkkola, M. Shozaki, J. Kangas, O. Venta, "Phonetic Typewriter for Finnish and Japanese," *Proceedings of ICASSP 88*, USA, Volume S, page 607, April 1988.

14) I. Lecomte, M. Lever, L. Lelievre, M. Delprat, A. Tassy, "Medium Band Radio Communications," Proceedings of ICASSP 88, USA, April 1988.
15) J.B. Reimer, K.S. Lin, "TMS320 Digital Signal Processors in Speech Applications," Proceedings of SPEECH TECH '88, April 1988.
16) M. Smmendorfer, D. Kopp, H. Hackbarth, "A High-Performance Multiprocessor System for Speech Processing Applications," *Proceedings of ICASSP 88*, USA, Volume V, page 2108, April 1988.
17) P. Vary, K. Hellwig, R. Hoffmann, R.J. Sluyter, C. Garland, M. Russo, "Speech Codec for the European Mobile Radio System," *Proceedings of ICASSP 88*, USA, Volume S, page 227, April 1988.
18) A. Hunt, "A Speaker-Independent Telephone Speech Recognition System: The VCS TeleRec," *Speech Technology*, USA, Volume 4, Number 2, pages 80–82, March–April 1988.
19) R.A. Sukkar, J.L. LoCicero, J.W. Picone, "Design and Implementation of a Robust Pitch Detector Based on a Parallel Processing Technique," *IEEE Journal of Selected Areas of Communications*, USA, Volume 6, Number 2, pages 441–451, February 1988.
20) A.Z. Baraniecki, "Digital Coding of Speech Algorithms and Architecture," *Proceedings of IECON '87*, November 1987.
21) G.R. Steber, "Audio Frequency DSP Laboratory on a Chip-TMS32010," *Proceedings of IECON '87*, Volume 2, pages 1047–1051, November 1987.
22) S.H. Kim, K.R. Hong, H.B. Han, W.H. Hong, "Implementation of Real Time Adaptive Lattice Predictor on Digital Signal Processor," *Proceedings of TENCON 87*, South Korea, Volume 3, pages 1131–1135, August 1987.
23) J.B. Reimer, M.L. McMahan, W.W. Anderson, "Speech Recognition For a Low Cost System Using a DSP," *Digest of Technical Papers for 1987 International Conference on Consumer Electronics*, June 1987.
24) A. Ciaramella, G. Venuti, "Vector Quantization Firmware For an Acoustical Front-End Using the TMS32020," *Proceedings of ICASSP 87*, USA, Catalog Number 87CH2396-0, Volume 4, pages 1895–1898, April 1987.
25) G.A. Frantz, K.S. Lin, "A Low Cost Speech System Using the TMS320C17," *Proceedings of SPEECH TECH '87*, pages 25–29, April 1987.
26) Z. Gorzynski, "Realtime Multitasking Speech Application on the TMS320," *Microprocessors and Microsystems*, Great Britain, Volume 11, Number 3, pages 149–156, April 1987.
27) P. Papamichalis, D. Lively, "Implementation of the DOD Standard LPC-10/52E on the TMS320C25," *Proceedings of SPEECH TECH '87*, pages 201–204, April 1987.
28) B.I. Pawate, M.L. McMahan, R.H. Wiggins, G.R. Doddington, P.K. Rajasekaran, "Connected Word Processor on a Multiprocessor System," *Proceedings of ICASSP 87*, USA, Catalog Number 87CH2396-0, Volume 2, pages 1151–1154, April 1987.
29) S. Roucos, A. Wilgus, W. Russell, "A Segment Vocoder Algorithm For Real-Time Implementation," *Proceedings of ICASSP 87*, USA, Catalog Number 87CH2396-0, Volume 4, pages 1949–1952, April 1987.
30) H. Yeh, "Adaptive Noise Cancellation For Speech With a TMS32020," *Proceedings of ICASSP 87*, USA, Catalog Number 87CH2396-0, Volume 2, pages 1171–1174, April 1987.

31) R. Conover, D. Gustafson, "VLSI Architecture For Cepstrum Calculations," *1987 IEEE Region 5 Conference*, USA, Catalog Number 87CH2383-8, pages 63–64, March 1987.
32) K. Field, A. Derr, L. Cosell, C. Henry, M. Kasner, J. Tiao, "A Single Board Multrate APC Speech Coding Terminal," *Proceedings of ICASSP 87*, USA, Catalog Number 87CH2396-0, Volume 2, pages 960–963, April 1987.
33) H. Brehm, W. Stammler, "Description and Generation of Sperically Invariant Speech-Model Signals," *Signal Processing, Netherlands*, Volume 12, Number 2, pages 119–141, March 1987.
34) A.Z. Baraniecki, "Digital Coding of Speech Algorithms and Architectures," *Proceedings of IECON '87*, Volume 2, pages 977–984, 1987.
35) B. Flocon, P. Lockwood, J. Sap, L. Sauter, "MARIPA: Speaker Independent Recognition of Speech on IBM-PC," *Eighth International Conference on Pattern Recognition*, Catalog Number 86CH2342-4, pages 893–895, October 1986.
36) M.T. Reilly, "A Hybridized Linear Prediction Code Speech Synthesizer," *Conference Records for MILCOM 86*, USA, Catalog Number 86CH2323-4, Volume 2, 32.5/1–5, October 1986.
37) K. Torkkola, H. Riittinen, T. Kohonen, "Microprocessor-Based Word Recognizer For a Large Vocabulary," *Eighth International Conference on Speech Recognition Proceedings*, Catalog Number 86CH2342-4, pages 814–816, October 1986.
38) C.H. Lee, D.Y. Cheng, D.A. Russo et al, "An Integrated Voice-Controlled Voice Messaging System," *Proceedings of Speech Technology 86*, April 1986.
39) Kun-Shan Lin and G.A. Frantz, "A Survey of Available Speech Hardware for Computer Systems," *Proceedings of Speech Technology 86*, April 1986.
40) L.R. Morris, "Software Engineering for an IBM PC/TI-SPEECH Realtime Digital Speech Spectrogram Production System," *Proceedings of Speech Technology 86*, April 1986.
41) K. Torkkola, H. Riittinen, "A Microprocessor-Based Recognition System For Large Vocabularies," *Proceedings of ICASSP 86*, USA, Catalog Number 86CH2243-4, Volume 1, pages 333–337, April 1986.1)
42) Z. Gorzynski, "Real Time Software Engineering on the TMS320: Application in a Pitch Detector Implementation," *International Conference on Speech Input/Output; Techniques and Applications*, Conference Publication Number 258, pages 270–275, March 1986.
43) S. Ganesan, M.O. Ahmad, "A Real Time Speech Signal Processor," *Proceedings of the ISMM Internal Symposium*, pages 46–49, February 1986.
44) L. Gutcho, "DECtalk-a Year Later," *Speech Technology*, Volume 3, Number 1, pages 98–102, August–September 1985.
45) B. Bryden, H.R. Hassanein, "Implementation of a Hybrid Pitch-Excited/Multipulse Vocoder for Cost-Effective Mobile Communications," *Proceedings of Speech Technology 85*, April 1985.
46) M. McMahan, "A Complete Speech Application Development Environment," *Proceedings of SPEECH TECH 85*, pages 293–295, April 1985.
47) H. Hassanein and B. Bryden, "Implementation of the Gold-Rabiner Pitch Detector in a Real Time Environment Using an Improved Voicing Detector," *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, USA, 1985.

48) K. Lin and G. Frantz, "Speech Applications with a General Purpose Digital Signal Processor," *IEEE Region 5 Conference Record*, USA, March 1985.
49) K. Lin and G. Frantz, "Speech Applications Created by a Microcomputer," *IEEE Potentials*, USA, December 1985.
50) M. Malcangi, "Programmable VLSI's for Vocal Signals," *Electronica Oggi*, Italy, Number 10, pages 103–113, October 1984.
51) V. Kroneck, "Conversing with the Computer," *Elektrotechnik*, Germany, Volume 66, Number 20, pages 16–18, October 1984.
52) P.K. Rajasekaran and G.R. Doddington, "Real-Time Factoring of the Linear Prediction Polynomial of Speech Signals," *Digital Signal Processing – 1984: Proceedings of the International Conference*, pages 405–410, September 1984.
53) M. Hutchins and L. Dusek, "Advanced ICs Spawn Practical Speech Recognition," *Computer Design*, USA, Volume 23, Number 5, pages 133–139, May 1984.
54) E. Catier, "Listening Cards or Speech Recognition," *Electronique Industrielle*, France, Number 67, pages 72–76, March 1984.
55) O. Ericsson, "Special Processor Did Not Meet Requirements – Built Own Synthesizer," *Elteknik Aktuell Elektronik*, Sweden, Number 3, pages 32–36, February 1984.
56) H. Strube, "Synthesis Part of a 'Log Area Ratio' Vocoder Implemented on a Signal-Processing Microcomputer," *IEEE Transactions on Acoustics, Speech and Signal Processing*, USA, Volume ASSP-32, Number 1, pages 183–185, February 1984.
57) B. Bryden and H. Hassanein, "Implementation of Full Duplex 2.4 Kbps LPC Vocoder on a Single TMS320 Microprocessor Chip," *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, USA, 1984.
58) M. Dankberg, R. Iltis, D. Saxton, and P. Wilson, "Implementation of the RELP Vocoder Using the TMS320," *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, USA, 1984.
59) A. Holck and W. Anderson, "A Single-Processor LPC Vocoder," *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, USA, 1984.
60) N. Morgan, "Talking Chips," McGraw-Hill, 1984.
61) A. Kumarkanchan, "Microprocessors Provide Speech to Instruments," *Journal of Institute of Electronic and Telecommunication Engineers*, India, Volume 29, Number 12, December 1983.
62) L. Dusek, T. Schalk, and M. McMahan, "Voice Recognition Joins Speech on Programmable Board," *Electronics*, USA, Volume 56, Number 8, pages 128–132, April 1983.
63) J.R. Lineback, "Voice Recognition Listens For Its Cue," *Electronics*, USA, Volume 56, Number 1, page 110, January 1983.
64) D. Daly and L. Bergeron, "A Programmable Voice Digitizer Using the TI TMS320 Microcomputer," *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, USA, 1983.
65) W. Gass, "The TMS32010 Provides Speech I/O for the Personal Computer," *Mini/Micro Northeast Electronics Show and Convention*, USA, 1983.
66) A. Holck, "Low-Cost Speech Processing with TMS32010," *Midcon/83 Conference Record*, USA, 1983.

67) H. Strube, R. Wilhelms, and P. Meyer, "Towards Quasiarticulatory Speech Synthesis in Real Time," *Proceedings of EUSIPCO-83 Second European Signal Processing Conference*, Netherlands, 1983.
68) T. Schalk and M. McMahan, "Firmware-Programmable μc Aid Speech Recognition," *Electronic Design*, Volume 30, Number 15, pages 143–147, July 1982.

*Control*

1) I. Ahmed, "16-Bit DSP Microcontroller Fits Motion Control System Application," *PCIM*, October 1988.
2) D. Bursky, "Merged Resources Solve Control Headaches," *Electronic Design*, USA, pp 157–159, October 1988.
3) I. Ahmed, "Implementation of Self Tuning Regulators with TMS320 Family of Digital Signal Processors," *MOTORCON '88*, pages 248–262, September 1988.
4) D. Dunnion, M. Stropoli, "Design a Hard-Disk Controller with DSP Techniques," *Electronic Design*, USA, pages 117–121, September 1988.
5) R. van der Kruk, J. Scannell, "Motion Controller Employs DSP Technology," *PCIM*, September 1988.
6) S.W. Yates, R.D. Williams, "A Fault Tolerant Multiprocessor Controller For Magnetic Bearings," *IEEE Micro*, USA, Volume 8, Number 4, page 6, August 1988.
7) I. Garate, R.A. Carrasco, A.L. Bowden, "An Integrated Digital Controller For Brushless AC Motors Using a DSP Microprocessor," *Third International Conference on Power Electronics and Variable-Speed Drive*, Conference Publication Number 291, Conference Publication Number 291, pages 249–252, July 1988.
8) J.M. Corliss, R. Neubert, "DSP Keeps Keep Disk Drive on Track," *Computer Design*, USA, pages 60–65, June 1988.
9) Y.V.V.S. Murty, W.J. Smolinski, S. Sivakumar, "Design of a Digital Protection Scheme For Power Transformers Using Optimal State Observers," *IEE Proc. C, Generation Transmission, Distribution*, Great Britain, Volume 135, Number 3, pages 224–230, May 1988.
10) R.D. Jackson, D.S. Wijesundera, "Direct Digital Control of Induction Motor Currents," *IEE Colloquim on 'Microcomputer Instrumentation and Control Systems in Power Electronics*, Great Britain, Digest Number 61, 1/1–3, April 1988.
11) A. Lovrich, G. Troullinos, R. Chirayil, "An All Digital Automatic Gain Control," *Proceedings of ICASSP 88*, USA, Volume D, page 1734, April 1988.
12) K. Bala, "Running on Imbedded Power," *Electronics Engineering Times*, USA, March 1988.
13) I. Ahmed, S. Meshkat, "Using DSPs in Control," *Control Engineering*, February 1988.
14) M. Babb (Editor), "Solving Control Problems With Specialized Processors," *Control Engineering*, February 1988.
15) S. Meshkat, "High-Level Motion Control Programming Using DSPs," *Control Engineering*, February 1988.
16) S. Meshkat, I. Ahmed, "Using DSPs in AC Induction Motor Drives," *Control Engineering*, February 1988.

17) J. Tan, N. Kyriakopoulos, "Implemmention of a Tracking Kalman Filter on a Digital Signal Processor," *IEEE Transactions of Industrial Electronics*, USA, Volume 35, Number 1, pages 126–134, February 1988.
18) H. Hanselman, "LQG-Control of a Highly Resonant Disc Drive Head Positioning Actuator," *IEEE Transactions on Industrial Electronics*, USA, Volume 35, Number 1, pages 100–104, February 1988.
19) I. Ahmed, "DSP Architectures for Digital Control Systems," *SATECH 1988*, 1988.
20) S. Meskat, "Advanced Motion Control Systems," *Intertec Communications* – Ventura, CA., 1988.
21) I. Ahmed, S. Lundquist, "DSPs Tame Adaptive Control," *Machine Design*, USA, Volume 59, Number 28, pages 125–129, November 1987.
22) B.K. Bose, P.M. Szczesny, "A Microcomputer-Based Control and Simulation of an Advanced IPM Synchronous Machine Drive System For Electric Vehicle Propulsion," *Proceedings of IECON '87*, Volume 1, pages 454–463, November 1987.
23) Y. Dote, M. Shinojima, R.G. Hoft, "Digital Signal Processor (DSP)-Based Novel Variable Structure Control For Robotic Manipulator," *Proceedings of IECON '87*, Volume 1, pages 175–179, November 1987.
24) J.P. Pratt, S. Gruber, "A Real-Time Digital Simulation of Synchronous Machines: Stability Consiferations and Implementation," *IEEE Transactions on Industrial Electronics*, USA, Volume 1E-34, Number 4, pages 483–493, November 1987.
25) I. Ahmed, "Deadbeat Controllers and Observers with the TMS320," *MOTORCON '87*, pages 22–33, September 1987.
26) I. Ahmed, S. Lindquist, "Digital Signal Processors: Simplifying High-Performance Control," *Machine Design*, September 1987.
27) R.D. Ciskowski, C.H. Liu, H.H. Ottesen, S.U. Rahman, "System Identification: An Experimental Verification," *IBM Journal of Research Developments*, Volume 31, Number 5, pages 571–584, September 1987.
28) J.A. Taufiq, R.J. Chance, C.J. Goodman, "On-Line Implementation of Optimised PWM Schemes For Traction Inverter Drives," *International Conference of 'Electric Railway Systems For a New Century*, Conference Publication Number 279, September 1987.
29) Y. Dote, M. Shinojima, H. Yoshimura, "Microprocessor-Based Novel Variable Structure Control For Robot Manipulator," *Proceedings of the 10th. IFAC World Congress*, July 1987.
30) H. Hanselmann, A. Schwarte, "Generation of Fast Target Processor Code From High Level Controller Descriptions," Presented at 10th. IFAC World Congress, July 1987.
31) E. Debourse, "Emergence of DSPs in Machine-Tool Axes Control Systems: Application of Distributed Interpolation Concepts," *Proceedings of the International Workshop on Industrial Automation*, February 1987.
32) C. Chen, "The Mathematical Model and Computer Simulation of an LCI Drive," *Electrical Machinery Power Systems*, USA, Volume 13, Number 3, pages 195–206, 1987.
33) R.D. Ciskowski, C.H. Liu, H.H. Ottesen, S.U. Rahman, "System Identification: An Experimental Verification," *IBM Journal Research Development*, USA, September 1987.
34) H. Hanselmann, "Implementation of Digital Controllers – A Survey," *Automatica*, Volume 23, Number 1, pages 7–32, 1987.

35) H. Henrichfreise, W. Moritz, H. Siemensmeyer, "Control of a Light, Elastic Manipulation Device," *Conference on Applied Motion Control*, 1987.
36) M.C. Stich, "Digital Servo Algorithm For Disk Actuator Control," *Conference on Applied Motion Control*, pages 35–41, 1987.
37) T. Takeshita, K. Kameda, H. Ohashi, N. Matsui, "Digital Signal Processor Based High Speed Current Control of Brushless Motor," *Electronic Engineering*, Japan, USA, Volume 106, Number 6, pages 42–49, November–December 1986.
38) R. Lessmeier, W. Schumacher, W. Leonard, "Microprocessor-Controlled AC-Servo Drives With Synchronous or Induction Motors: Which is Preferable?," *IEEE Transactions On Industry Applications*, USA, September/October 1986.
39) R. Alcantara, J. Prado, C Guegen, "Fixed-Point Implementation of the Fast Kalman Algorithm: Using the TMS32010 Microprocessor," *Proceedings of EUSIPCO-86*, Volume 2, pages 1335–1338, September 1986.
40) B. Nowrouzian, M.H. Hamza, "DC Motor Control Using a Switched-Capacitor Circuit," *Proceedings of the IASTED International Symposium on High Technology in the Power Industry*, pages 352–356, August 1986.
41) N. Matsui, T. Takeshita, "Digital Signal Processor-Based Controllers For Motors," *SICE*, July 1986.
42) H. Hanselmann, "Using Digital Signal Processors For Control," *Proceedings of EICON*, 1986.
43) H. Hanselman, W. Moritz, "High Banwidth Control of the Head Positioning Mechanism in a Winchester Disc Drive," *Proceedings of IECON*, pages 864–869, 1986.
44) R. Cushman, "Easy-to-Use DSP Converter ICs Simplify Industrial-Control Tasks," *Electronic Design*, USA, Volume 29, Number 17, pages 218–228, August 1984.
45) W. Loges, "Signal Processor as High-Speed Digital Controller," *Elektronik Industrie*, Germany, Volume 15, Number 5, pages 30–32, 1984.
46) W. Loges, "Higher-Order Control Systems with Signal Processor TMS320," *Elektronik*, Germany, Volume 32, Number 25, pages 53–55, December 1983.

*Military*

1) V. Lazzari, Quacchia, M. Sereno, E. Turco, "Implementation of a 16 Kbit/s Split Band-Adaptive Predictive Codec For Digital Mobile Radio Systems," *CSELT Technical Reports*, Italy, Volume 16, Number 5, pages 443–447, August 1988.
2) P. Papamichalis, J. Reimer, "Implementation of the Data Encryption Standard Using the TMS32010," *Digital Signal Processing Applications*, 1986.

*Telecommunications*

1) S. Casale, R. Russo, G.C. Bellina, "Optimal Architectural Solution Using DSP Processors for the Implementation of an ADPCM Transcoder," *Proceedings of GLOBECOM '89*, pages 1267–1273, November 1989.
2) A. Lovrich and J.B. Reimer, "A Multi-Rate Transcoder," *Transactions on Consumer Electronics*, USA, November 1989.
3) J.L. Dixon, V.K. Varma, N.R. Sollenberger, D.W. Lin, "Single DSP Implementation of a 16 Kbps Sub-Band Speech Coder for Portable Communications," *Proceedings of ICASSP 89*, USA, pages 184–187, May 1989.

# Index

# E



# F



# G



# H



# I



# K



# L

# M



# N



# P



# Q

TEXAS INSTRUMENTS

Printed in U.S.A., March 1990

SPRA016