Digital Signal Processing AND THE MICROCONTROLLER MOTOROLA

Digital Signal Processing

AND THE MICROCONTROLLER

1 0 0 1 0 0 0

1 0 0 0 0 1 1

0 1 1 0 0 0 1

0 1 1 0 1 1 0

MOTOROLA

Digital Signal Processing and The Microcontroller

Mark McQuilken
Microprocessor and Memory Technologies Group
Motorola, Inc.
Austin, Texas

James P. LeBlanc
Electrical Engineering Department
Cornell University
Ithaca, New York

ACKNOWLEDGMENTS

The Motorola Advanced Microcontroller Division gratefully acknowledges the contributions of the following people in the creation of this book. Many thanks to:

Scott Anderson and Tom Marischen for their commitment to "doing the right thing" *regardless* of division boundaries.

Arie Brish for reviewing many of the chapters, even over the holidays, for technical accuracy. His responsiveness and willingness to "dig-in" and help resolve production schedule crises made delivery of this work possible. Arie, should we have reserved *this acknowledgement* for the next edition?

Francis Christian for sharing his field experience and insights, and for technically reviewing some of the material.

James LeBlanc, former Motorolan, who left the "rat race" to obtain his PhD. and further his already extensive knowledge of DSP. Despite a myriad of personal pressures, he still managed to fulfill his commitment to Motorola and provide us with high quality material for chapters four through seven (and then some!). Without his contributions, this book would not have been possible. Now if we could just get him to stop listening to "The Buzzcocks".

Steve Marsh, who mandated that this book be written, left for "greener pastures", and then stood back and watched the fun! Despite this, he and his wife Jackie most graciously edited, proofread, and critiqued various versions of the manuscript.

Mark McQuilken, who deserves much more than a thank you for the enormous amount of effort and time spent, much of it at his family's expense, in authoring chapters one through three, editing and reviewing all material, preparing figures and drawings, and coordinating the whole project. His never-ending patience was much appreciated. And, special thanks go to Mark's wife, Beth, for so graciously putting up with it all, and for her help in editing some of the material.

Judy Racino, who got saddled, during the last two weeks of the production cycle, with the responsibility to complete this work *on-time*. Her honest, clear-minded approach to problem solving avoided many potential obstacles. She willingly spent many long hours and did "whatever it took" (e.g., writing, editing, proofreading, going for sandwiches, etc.) to deliver this book to the printer *on-time*.

Tanya Tussing, who spent untold time and effort in editing and reviewing, including the removal of thousands of prepositional phrases, to ensure technical accuracy and understandability.

John Waite, who provided the glossary of technical terms.

Brian Wilkie for his negotiating skills, financial *and* moral support of this project.

A "tip of the hat" to Tim Ahrens, Robert Chretien, Mark Glenewinkel, Mark Johnson, and Craig Haller for helpful edits, suggestions, and moral support.

Special thanks go to Staats Falkenberg & Partners for their assistance with many of the figures and for designing the cover art: especially to Jack Robberson for his willingness to always help in tight situations and for being so very organized.

Admundson and Associates for providing the required last minute resources, and especially to (last-but-not-least) Wendy Heath for her editing/layout skills, creativity, shortened honeymoon, and for sticking with us to the end (well, at least the end of *this* edition). Wendy, what time did you finish last night?!

PREFACE

Microcontrollers are signal processors. This may come as a surprise to some, but in the strictest interpretation it's true: microcontrollers (MCUs) are almost always used to evaluate inputs (signals) from sensors and the like, then manipulate (process) the digital form of those inputs and produce some relevant output. If you're like most of us however, we don't think of MCUs when we think of digital signal processing (DSP). That's probably because of two general misconceptions: 1) DSP may only be accomplished with ultra-fast processors that have been designed for the purposes of doing DSP, and 2) you have to be a brilliant person just to understand DSP, let alone use it. This guidebook will show that both of these are truly misconceptions.

The smaller geometries achieved by the semiconductor industry has made possible more highly integrated microcontrollers; practically a full computer system on one chip. Because of these advances in technology, digital signal processing (DSP) functions are being integrated as part of the microcontroller. As a result of high integration levels and diversity of functions now available on a single chip, electrical engineers are becoming exposed to areas that were previously covered by specialists. Digital signal processing is a prime example.

This book is intended to help bridge the knowledge gap between the two technologies: digital signal processors and microcontrollers. The target audience is design engineers, programmers, technicians, and students who deal with embedded control applications, but need to have a working knowledge of DSP concepts. Most digital signal processing books are highly theoretical and focus on the mathematics behind DSP. The style of this book is a little different - it is written in a more casual tone. We believe this type of presentation will help the reader enjoy and understand the topics discussed.

Chapter Descriptions

Chapter 1 introduces the microcontroller in general, and in particular, discusses the traits, and extols the space-and-cost-saving virtues of the Motorola MC68HC705C8 (C8). The C8 is then used in a proto-typical application, a household thermostat, to illustrate an MCU's usage in the "real-world". As part of this illustration, a noise problem is hypothetically encountered which is resolved by having the MCU average the readings coming from the Analog-to-Digital Converter (ADC). This averaging process is then graphically developed into a "lattice diagram", a graphic device which displays the process of traditional *digital filtering*. The rudiments of Analog-to-Digital conversion and specific ADC types are then briefly presented.

Chapter 2 discusses the strengths and weaknesses of "doing things digitally" with MCUs. The digital domain strengths are shown two ways: a) through the development of an MCU-based magnetic tape noise reduction system, and, b) through the implementation of an MCU-based "digital delay". Although both the noise reduction and delay units have typically and traditionally been implemented using analog circuits, the digital MCU versions are shown to be superior because of the inherent strengths of the digital domain. The details, of each application, even including assembly code segments are shown. This chapter is closed with a brief overview of the disadvantages of working in the digital domain.

Chapter 3 shows the benefits and practical details of remapping our "oscilloscope-view" of signals (classically referred to as the "time-domain") to a frequency response view (classically referred to as the "frequency domain") for use by an MCU. This remapping is accomplished by using the Fourier transform (FT). The FT is presented as a set of paired signals (transform pairs; time domain and frequency domain versions of a given signal) and a set of rules, including FT properties, rather than as mathematical manipulations of the messiest kind. The transform pairs and rules are then used to review the thermostat example of Chapter 1 as a digital filter. In addition, the Nyquist criteria for digitally sampled systems and signal aliasing are "proven" by intuitively applying the FT. The chapter is closed with a brief list of recommendations for continuing to learn Digital Signal Processing.

Chapter 4 discusses the fundamental concepts of signal processing by reviewing some of the basic building blocks required to apply signal processing: the characteristics of sinusoids and their spectra, the delta function, real and complex modulation, imaginary and complex numbers, Euler's relation, and phasors. These basic ideas are then applied to produce a heuristic presentation of the Fourier transform and its basic properties which are presented, unlike Chapter 3, with some of the mathematics of the FT. The chapter is closed with a discussion of FT transform pairs — all necessary fare to understand the contents of chapter 5.

Chapter 5 presents elements of signal filters, both ideal and "real-world" implementations, and transfer functions. Topics include: using the time domain to describe filters, signal convolution, and the duality of the FT. The specific characteristics of an ideal filter are presented with an eye toward defining the critical parts of a filter's response. Since ideal filters, like perfect circles, are not realized in the real-world, the topic shifts from ideal to real-world filters and the considerations which must be made, like "windowing", to understand and develop real filters. The chapter ends with a review of typical signal classifications band-widths one is likely to encounter when developing filters.

Chapter 6 begins the discussion of digital signal processing with an emphasis, as with previous chapters, on filtering. The moving average filter, like that of Chapter 1's thermostat example, is presented along with the mathematics. The relationship between this filter type and the digital version of convolution (the convolution sum) is defined. A connection between the concept of impulse response and frequency response is presented. By combining the impulse and frequency response concepts with the transfer function, two classes of filter types are constructed which are expressed through the lattice diagram and the "z^{-1}" unit time delay. The last item covered in this chapter is a characterization of the frequency response of a discrete time system.

Chapter 7 discusses the development of digital filters. A comparison between FIR and IIR filter types is presented. This chapter is the culmination of basic signal processing building blocks and requires some understanding of the topics in previous chapters.

Notes

Some microcontroller prices are mentioned in this book. These are general price ranges and not actual price quotes. If you are interested in obtaining prices for Motorola microcontrollers, please contact your nearest Motorola sales office or Motorola authorized distributor.

Practical implementations of DSP theory sometimes requires a few extra steps, which frequently are not included in this book. For more rigorous discussions of 1) all the required assumptions about signals and systems for DSP theory to be true, and 2) additional transfer functions that do front-end signal manipulations required for real-time DSP functions, consult one or more of the books listed in our reference section (at the end of this book).

Digital Signal Processing And The Microcontroller is a preliminary edition especially developed as an educational tool and specifically to be part of the "HC16 Tool Kit." Motorola plans to have future editions available, which will be published by Prentice-Hall. We appreciate your constructive comments and suggestions. After reviewing this guidebook, please take time to fill out the business reply card included with the kit and return it to Motorola.

CONTENTS

CHAPTER 1

The Microcontroller

What Is An MCU?

Today, the microcontroller (MCU) is regarded as a single piece of silicon that contains all of the major elements of a computer *system*. The MCU not only includes the central processing unit (CPU), comprised of the arithmetic logic unit (ALU) and execution unit (EU), but the peripheral devices and memory arrays as well. The common MCU peripherals and memories coresident with the CPU include such diverse devices as:

- Analog-to-Digital Converters (**ADCs**)
- Digital-to-Analog Converters (**DACs**)
- Digital input/output lines
- **H-bridge motor drivers**
- Voltage comparators
- Math coprocessors
- Light-emitting Diode (LED) drivers
- Random-Access-Memory (**RAM**)
- Read-Only-Memory (**ROM**)
- Electrically-Erasable Programmable ROM (**EEPROM**)
- Electrically-Programmable ROM (**EPROM**)
- Timer/counters
- **Watchdog timers**
- **Low-voltage detectors**
- High-speed Synchronous Peripheral Interface (**SPI**)
- Universal Asynchronous Receiver/Transmitters (**UART**)

This list is certainly not exhaustive and changes almost weekly. It does, however, represent a good cross section of the diversity of peripherals available on today's MCUs.

Typically, these peripherals and memories are not on a single chip; although this is technically feasible, each peripheral/memory adds more costs to the MCU. As we will reveal later, "an expensive MCU" is an **oxymoron** like "jumbo shrimp."

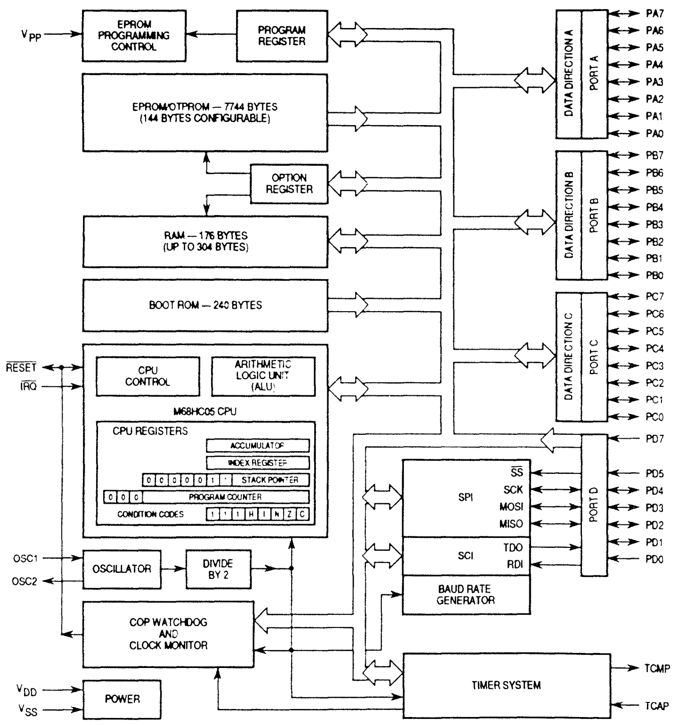


Figure 1-1
C8 Block Diagram

A Sample MCU

A typical combination of peripherals is one of the most popular MCUs these days, the Motorola MC68HC705C8 (see Figure 1-1). To understand why the MCU is becoming increasingly popular, we'll tabulate the approximate number and type of peripheral devices displaced by on-chip peripherals as we look at MCU features. Keep in mind that all of the features outlined below are on a single chip whose package, a 44-pin plastic leaded chip carrier (**PLCC**) in this case, requires no more area than .5 square inch.

Features Already Built-in to This MCU:

- On-chip driver for crystal or ceramic resonator to build a stable oscillator. This feature is very common today, so a separate, external crystal oscillator, which requires a 14-pin DIP inverter and some discrete components, is not needed.

- Memory-mapped digital I/O: 24 bidirectional and 7 input-only lines. The bidirectional lines are software configurable, one bit at a time. This is pretty nice if the exact mix of digital I/O is unknown at design time. As long as you don't exceed the maximum count, your software can determine the I/O mix, on-the-fly. This function used to be provided by such devices as the MC6821 Peripheral Interface Adapter (PIA), a 40-pin DIP.
- 7,744 bytes of EPROM for user program/data storage. This is roughly equivalent to having a 27C64 type EPROM, a 28-pin DIP, on the same chip as the CPU.
- Up to 304 bytes of fully static RAM. Since this amount of RAM is small even when compared to some of the smallest available, we'll not count this in the package savings.
- Software-selectable memory configurations which allow RAM/ROM space tradeoff under program control. This is a neat feature if certain parts of your program need more RAM at times and more ROM at other times.
- Watchdog timer. This is a safeguard against a runaway program and potentially catastrophic system performance. Until it was integrated on devices like this MCU, the watchdog function was handled by an external circuit, a retriggerable monostable timer. This on-chip retriggerable monostable is digitally implemented. Its timebase is derived directly from the crystal oscillator, which avoids drift problems associated with analog-based retriggerable monostables. I've often seen analog retriggerable monostables implemented with a 555-type timer, an 8-pin mini-DIP.
- Serial communications interface (SCI). This is equivalent to something like the old tried and true MC6850, a 24-pin DIP. This self-contained system uses the CPU crystal timebase to derive various standard baud rates. With the addition of RS-232 level shifters connected to the receive and transmit pins, this is a complete system for communicating to terminal-like devices or other MCUs over long wires.
- Serial peripheral interface (SPI). This high-speed (up to 1Mbits/second on some devices) interface is a four-wire system: 2 data lines, a serial clock line, and a select line. This is useful for expanding I/O capability by hanging Serial-In, Parallel-Out (**SIPO**) or Parallel-In, Serial-Out (**PISO**) shift register devices on this SPI "bus." Plus, there are many peripherals available, such as EEPROM or ADC, that "**hang**" directly on this interface, expanding the capabilities of this MCU even further. The SPI system

peripheral was not really used in the old days, probably because CPUs had their buses laid bare for the world to use. These buses could be easily and directly connected to the peripheral, rather than through an intermediate interface system like the SPI. Since a single-chip MCU is self-contained when used as a single-chip, and the internal CPU bus is not available to the off-chip world, there is a considerable benefit to expanding I/O through such an SPI system.

- 16-bit timer subsystem. This system gives us the ability to time external events relative to CPU activity as well as to generate events. This is similar in function to a MC6840 Programmable Timer Module...yes, another old part and a 28-pin DIP.

Let's count up the packages we can eliminate by using an MCU along with the type, size, pin count, and area each package occupies. Table 1-1 contains this information.

Table 1-1
Multi-chip equivalent of 68HC705 C8's peripherals

Chip Size	Type	Length x Width	Area
8-pin DIP	Analog timer	0.4 x 0.3	0.12
14-pin DIP	Inverter	0.7 x 0.3	0.21
24-pin DIP	UART	1.2 x 0.6	0.72
28-pin DIP	EPROM	1.4 x 0.6	0.84
28-pin DIP	Timer	1.4 x 0.6	0.84
40-pin DIP	PIA	2.0 x 0.6	1.20

Using individual peripherals and memories, a minimum total area of 3.93 *inches*^{2} would be used. This excludes the **real-estate** of the printed circuit board (PCB) which would be consumed just to electrically connect these components. Compare 3.93 *inches*^{2} to the .5 *inches*^{2} the entire MCU consumes (see Figure 1- 2). Based on this comparison alone, the MCU is very useful and economical.

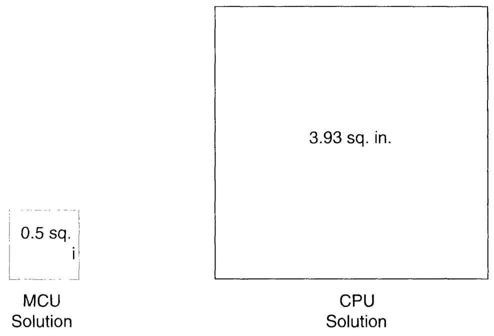


Figure 1-2
A comparison of areas required by like-functioned MCU and CPU systems.

What An MCU Is Not

Now that we've mentioned the remarkable features of a specific MCU and what an MCU is in general, we will discuss what an MCU is not:

It is not expensive. Although "expensiveness" is relative, MCUs would not be considered expensive by most. Some Motorola MCUs, for example, are available for as little as 90¢ in large quantities (yes, for the complexity of even this smaller MCU, that is a *very* reasonable price). Of course, for a logic gate like the MC74HC00, 90¢ is a lot of money. But if you find a MC68020 for that price (and it's still functional), buy it! The MC68020 CPU sells routinely for as much as $170. By comparison, the MCU is an *inexpensive* problem solver!

It is not bulky. Offered in various packages, a reasonably complex MCU can be obtained in a package as small as a 16-lead **SOIC** (.4 in. x .3 in., or .12 in^2). Even very complex microcontrollers, like the **MC68332** 32-bit MCU, similar in performance to some older mainframe computers like the VAX PDP-11, are still available in packages that take up no more room than 1.2 in^2. This is, literally, mainframe computing power in the palm of your hand!

It is not a "power pig". Advances in CMOS technology have continued to increase MCU speeds while reducing power consumption. One measure of microcontroller/microprocessor performance is the "instructions per second per watt"(IPS/W). The higher this quantity, the more efficiently the MCU is using each unit of power. Calculating this quantity for a popular NMOS CPU of days gone by, the Motorola **MC6800**, we get 1 million IPS/W maximum. Compared to the same calculation for the newer **MC68HC11A8**, an even more complex product than the relatively primitive 6800, we get 13.3 million IPS/W — over 13 times more efficient in the use of available power! Of course, at this point we could argue about the technical term "power pig" and the classification of one or another MCU as such a pig, but we would miss the point: The MCU of today uses power more efficiently than any of its predecessors.

Is not difficult to use. Building a microcontroller application is still a technically oriented process. It requires many levels of understanding in highly technical disciplines. It requires a leap of faith to mentally follow the process that generates ASCII data on a word processor and then assembles, downloads, and emulates the ASCII text file — almost incomprehensible to most non-engineers. It's a lot easier and less expensive to develop MCU applications today than 15 years ago. In these modern times, a complete in-circuit emulator with software, both assembler and PC-host software for the emulator, whose performance and features were available ten years ago for $10,000,.can now be bought for about $500. I've seen some people work with the new **Evaluation Module (EVM)**, Motorola's answer to low-cost, quick-learning environments, and program small applications on MCUs the same day! Pretty amazing considering this was accomplished by a non-electrical engineer.

It is not a panacea. You should be aware that an MCU can't be everything to everyone. Perhaps this isn't the time to question the MCU's potential usefulness, but we will discuss this later in the chapter. The limiting factor for the performance of an MCU-based system is usually the MCU's maximum bandwidth.

The Human Factor

At this point, you may have a suspicion that this MCU stuff actually may be useful. From where I sit, I'd say that you're right. But usefulness, like other things, depends on the observer. Consequently, as we discuss the potential usefulness of MCUs, we will compare it to the **vantage points** considered important in the electronics industry. Let's start with the most important points:

An Engineering View

If you've been engineering for a while, you know engineers never make mistakes. In fact, this may actually be Newton's long-lost fourth law. In any case, whenever there is a schedule slip:

a) It's usually the technician's fault, and/or,

b) The MegaBug 1000X compiler/assembler keeps randomly inserting HCF (Halt and Catch Fire) opcodes in your code, and/or,

c) The purchasing department still hasn't placed the part order you submitted a year ago last Tuesday.

Whatever the cause, the engineer always has to modify the design at the last minute. Fortunately, one of the advantages of MCU-based designs is they are easily modified. This is because, like most computers, the bulk of the MCU systems' performance lies in its programming. Consequently, from an engineer's perspective, the MCU is a hedge against schedule slips from other people's screw-ups.

Trying New Things

During the times engineers are not saving projects by our ingenious "on-the-fly" redesigns, we are often looking for new, innovative, and creative ways to accomplish old, boring, and unnecessary functions. For example, a colleague of mine recently designed a programmable power supply for generating EPROM and EEPROM programming voltages for some Motorola MCUs. He worked very hard to design in an MCU at the core of the power supply. His boss later pointed how much easier his job would have been had he used an adjustable linear regulator, a potentiometer instead of an MCU, and a half dozen other ICs and components. In all seriousness, the design did have some advantages - the output voltage could be set under computer (PC type) control by using the MCUs built-in serial interface. That's a feature we pay a lot for these days in commercial power supply units.

Easy Prototyping

Actually, the reason I mentioned this power supply is to illustrate the MCU's usefulness to the engineer as a very friendly and quick-prototyping environment. Even in the case of the programmable power supply, the real value of that approach is to provide an experimental platform for similar projects whose characteristics, such as rise and fall times, could be quickly modified. This quick turn-around of ideas in solving problems can make or break its success. During the early development stages of production test stations, we would often use the Motorola EVM in conjunction with "experimenter's boards" to generate signals which looked similar to the device under test (**DUT**). This allowed us to easily validate the performance of the production tester with known, measurable excitation. It took us no more than an hour with this approach compared to the time to do the same thing by wiring a bunch of counters and gates together.

One Part Fits All... Almost

Imagine having to stock only one IC in your engineering stockroom/workshop. It's an exaggeration to say an MCU is so flexible it would replace *all* the other ICs in your parts bins. If you've got to choose only one digital IC for prototyping, however, the MCU should be that device. Looking in the parts bins in my lab, I have devices like digital-to-analog converters (e.g., AD7524 CMOS DAC), analog-to-digital converters (like National's ADC0800), display drivers (MC14499 for example), analog multipliers (AD536), op-amps (TL074s, NE5534, etc.), EPROMs (2732s, 2764s), LSTTL and HCMOS logic ('138s,'00s, etc.), a few generic PALs (16R8s), and, of course, some MCUs (MC68HC705C8s, MC68705R3s, MC68HC811E2s, etc.). When trying out ideas, I use the MCUs most frequently. Let's say we need a two-digit LED display for a new product. I can either hook up an MCU, which feeds the MC14499 display driver via the SPI, or I can hook up the MCU in the configuration shown in Figure 1-3. In this form, the MCU drives the seven-segment LEDs directly, and the software instructs the MCU to decode and multiplex the

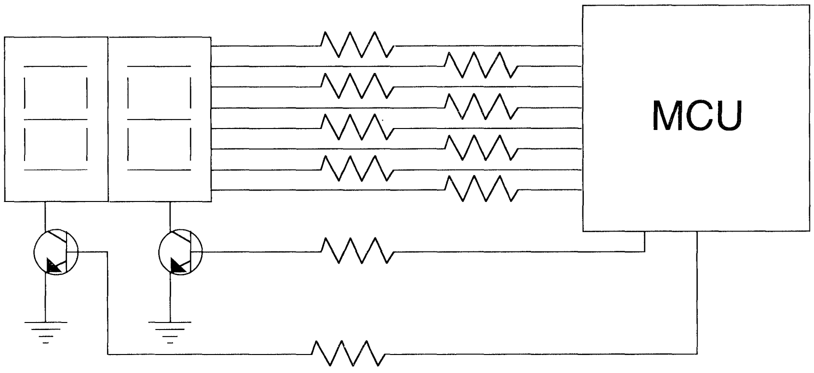


Figure 1-3
MCU drives LED directly

display. This eliminates the middleman, that is, the middle*circuit*, consisting of the MC14499 and its supporting components. Since the MCU software decides the decoding sent to the display, characters appear that aren't supported by the MC14499.

Despite the flexibility of the MCU in a prototyping and brainstorming environment, don't throw away all your non-MCU components. There are some things other components can do that the MCU can't — it's just the MCU may be the easiest, most flexible component in your bag of tricks. Some feature or function implemented with the MCU, to one degree or another, can replace each of the devices already mentioned.

Product Enhancements

Designing an MCU into a product affords the same flexibility we had for validating our test equipment. A typical situation might be to use an MCU to interface to an "aural annunciator" (that's a 25-cent word for the 10-cent word, "buzzer") like that shown in **Figure 1-4**. The characteristics of the buzzer will be changed sometime during the life of the product (see the preceding reasons a, b, and c for schedule slips, page 1-6). If this circuit had been done with the ubiquitous 555 monolithic timer, one or both of the timing elements (i.e., resistor and/or capacitor) will need to be physically changed sometime during its lifetime. Replacing a single resistor and/or capacitor is probably not a big deal. But imagine having to change some resistors and

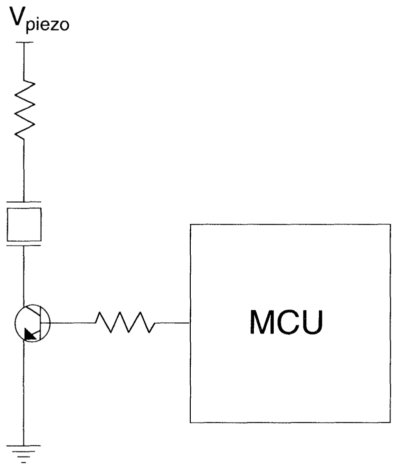


Figure 1-4
Very often a product change means software change only

capacitors in 10,000 final assemblies before shipping your product! An MCU can avoid many, but not all, uncomfortable situations like this one: an MCU-based unit could be easily removed and replaced with updated software without risking any damage the to PCB/assembly. There are many other situations where easily-altered MCUs can make a lot of cost-effective and *face-saving* sense.

MCUs Promote Proper Design

Most of us dread the day we make a mistake...that is, design-in *an undocumented feature*. Fortunately, this is more difficult when using an MCU. Many processes that can be brainlessly put together with other technologies (discrete, linear, small-scale integration logic, etc.) have to be carefully considered with an MCU. That is not to say an MCU is difficult, it just requires some thought. If you think about it in advance, there is another advantage to using MCUs: performance is more bounded and predictable than other choices. This is due to two reasons:

a) One of the advantages of the digital domain is bounded and predictable performance, and

b) The MCU, like most standard computational machines, is a sequential machine...basically, it can only make one mistake at a time.

We'll revisit this topic when discussing the advantages of the digital domain in the next chapter.

The View From Production

Production organizations appear to have two common traits: a) production people intensely dislike rework, and b) production managers typically don't like to inventory vast amounts of parts stock. The features of the MCU can turn your production people into "shiny, happy people."

A product can be designed around the MCU almost exclusively. The production manager will likely have fewer problems with a product having few components compared to one with many.

A Typical MCU Application: A Household Thermostat

Let's take a look at the MCU in action. This way you can experience first hand the MCU's value. Most importantly, however, this provides a good transition to describe some advantages of working in the digital domain.

Functionally, the household thermostat is nothing more than a "window comparator," a mechanism that evaluates the temperature of a room and operates the heating or cooling system (see Figure 1-5). In other words, if a room's temperature is greater or less than a predefined range of temperatures (called the "comfort zone"), the thermostat switches the heat or air conditioner on or off until the temperature is again within the comfort zone.

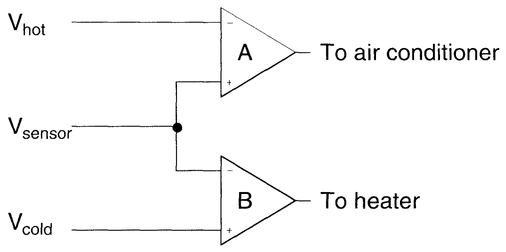


Figure 1-5
A Window Comparator

Comparators may be used as the main measuring function of the thermostat as indicated in Figure 1-5.

By adding a few more components, this function diagram could become the “real thing" in Figure 1-. The simplest version

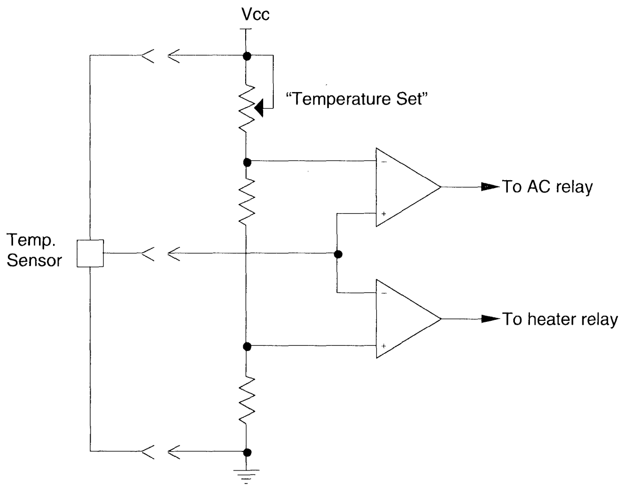


Figure 1-6
A typical household thermostat circuit

of a thermostat is a metallic strip usually wound in a coil used as a lever (contractor) in a single-pole double-throw switch. When the metal in the strip contracts from the cold, the lever is pulled one direction toward another contact point, which then completes a circuit and turns on the heat. The heat expands the metallic strip, and the lever moves in the opposite direction completing another circuit to turn on the air conditioner. Although this will vary from design to design, this describes the basic operation of a mechanical thermostat.

When such simple mechanisms are so reliable, an engineer must have *many* compelling reasons to use more complex and costly circuitry. Or the engineer has to be just plain crazy. Jim Sibigtroth, the original designer of an MCU-based thermostat, had some compelling reasons to design such an application. The block diagram for this thermostat is shown in Figure 1-7 .

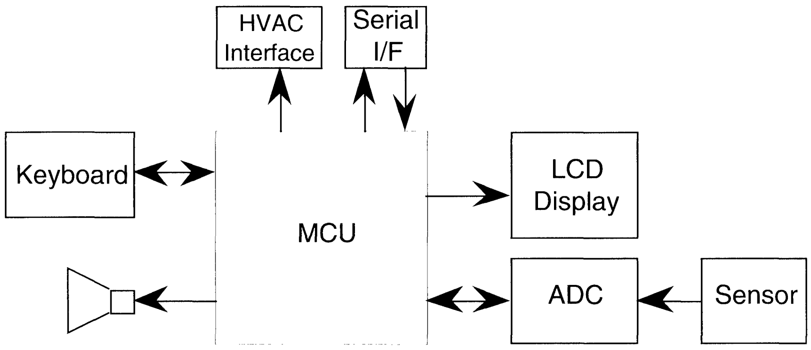


Figure 1-7
An MCU-based thermostat

Basic MCU Thermostat Operation

The microcontroller (MCU) reads the current room temperature through a solid-state temperature sensor whose output is translated by an Analog-to-Digital Converter (ADC) for the MCU. The ADC's output is interpreted by the MCU's software to determine the appropriate action for any temperature. Through a keypad, someone specifies the "trip points" that define the "comfort zone" for the heater and air conditioner which is indicated by liquid crystal display (LCD). The MCU acknowledges this operation through a **piezobuzzer**.

Furthermore, the MCU's serial interface is made accessible to allow some debugging without affecting the entire assembly. This is a very typical MCU application. Figure 1- shows the flowchart of the main control program.

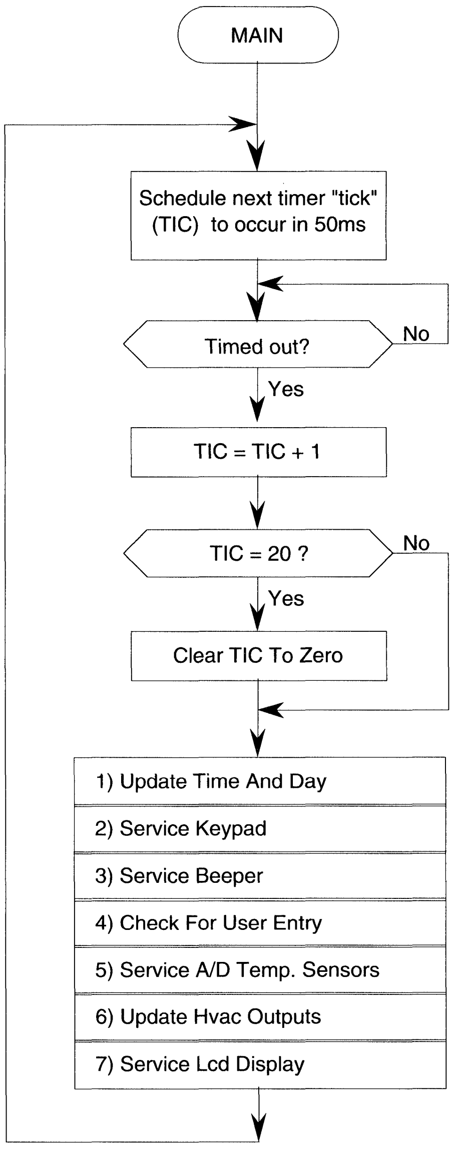


Figure 1-8
Main Control Program for Household Temperature Control

A Hypothermal Hypothetical

Let's say doing an MCU-based thermostat appeals to your sense of HTF (High Toy Factor) and has enticed you to try to build this thermostat project. So you send away to your area's Motorola sales office for the M68HC05 Applications Guide (M68HC05AG/AD) that has the details for the project. After painstakingly building the hardware, writing the software, and installing the thermostat in your wall, you discover the thermostat is acting strangely. At first, you're not even sure anything is wrong...but then...you notice: it's December, you live in Alaska, and your air conditioner has just kicked on! But you can to hook up your PC to the thermostat through a serial port and RS-232 level-shifters. Consequently, you can debug the problem without having to *gouge* the thermostat out of the wall. Fortunately, you are able to "look at" the data from the ADC and thus can ascertain, at the very least, if the problem is a malfunction with the temperature sensor or ADC. So, you set up your PC to take readings for a 12-hour period and retire to your igloo with a few extra dogs that night (a *three*-dog night, thanks to the air conditioner).

After a bone-chilling night of restless sleep, you get up to compile and graph your data from the previous 12 hours. This is the graph in Figure 1- . Being an astute observer, you notice two things: a) the word "comfort" is misspelled due to your typing "i" instead of "o" because of your hypothermally-induced shaking, and b) the readings from the sensor are

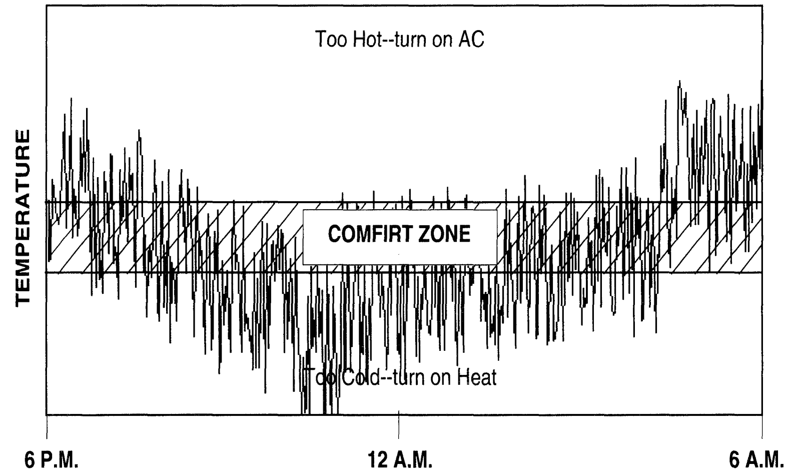


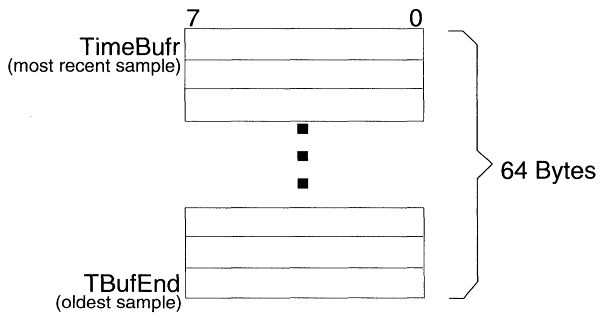
Figure 1-9
A temperature plot of a malfunctioning air condtioner system

rapidly changing. Based upon a rough thermodynamic calculation (done on the back of a napkin, of course) and your personal experience, you conclude the actual temperature can't be varying as wildly nor as frequently as indicated by your graph. It must be noise. What's an engineer to do? Here are some options:

- Low-pass filter the voltage(s) from the sensors. This makes sense but isn't indicated since you can't determine whether the noise is coming from the sensor(s) or there is a malfunction in the ADC. Besides it's going to get awfully messy to try and remove the unit from your igloo wall to put the resistors and capacitors needed to low-pass the sensor output(s). For these reasons, you decide this isn't a reasonable alternative.
- Move into another igloo. This sounds more like a management decision than an engineering one. Consequently, it isn't a reasonable alternative.
- Reduce the variations over time in the ADC data. This makes sense, assuming you can change the MCU code without changing the hardware. Since this is *my* hypothetical example, we'll choose this option. Figure 1- shows the ADC service routine for the thermostat with a call to the low-pass filter routine inserted.

Some Average(ing) Software

A simple jump-to-subroutine placed within the ADC read routine won't get rid of our noise problem: it's the subroutine that's going to do the work. What's in it? Look at the listing in Figure 1- . We'll take it one chunk of code at a time. Before starting, however, I want to introduce a graphic aid to help visualize the movement of data in the averaging routine. We'll use the following diagram to symbolize the MCU memory:



```
*******************************************************
* A2D - Check temp. sensors (via SPI and MC145041)      *
* If TIC = 0, send addr 0 ignore return data      *
* If TIC = 1, send addr 1 return data is ch.0 val *
* If TIC = 2, send addr 2 return data is ch.1 val *
* If TIC > 2, skip A2D routine                    *
*  To compensate for sensor & op-amp offset, A/D result *
*   will be modified by subtracting an offset constant  *
*******************************************************
A2D EQU *                      Check temp. sensors
 LDA  TIC                      If Tic = 0, 1, or 2 write to SPI
 CMP #2
 BHI XA2D                      If Tic > 2; Exit
 ASLA                          Move TIC # 0-2 to upper nibble
 ASLA
 ASLA
 ASLA                          4 bit left shift
 TST SPSR                      Reads SPIF (part of SPIF clear)
 BCLR 3,PORTC                  Drive low true SA/D CE* to 0
 STA SPDR                      Initiates a transfer

* Requests conversion of next channel and returns data
*   from previous channel Ch.0=Indoor Ch.1=Outdoor

SPIFLP BRCLR 7,SPSR,SPIFLP     Wait for SPI Xfer complete
 BSET 3,PORTC                  Drive low true SA/D CE* to 1
 LDA TIC                       If 0-Exit, 1 or 2 Read A/D data
 BEQ XA2D                      0 so exit
 LDA SPDR                      Get A/D data
 JSR LOWPASS                   Filter input with 64pt avg.
 BRSET 1,TIC,ADCH1             If Tic=2, data is Ch.1
 SUB OFF0                      A/D Ch.0; subtract offset
 STA INTMP                     Update indoor temperature
 BRA XA2D                      & Exit
ADCH1 SUB OFF1                 A/D Ch.1; subtract offset
 STA OUTMP                     Update outdoor temperature
XA2D RTS                       ** RETURN from A2D **
```

Figure 1-10
ADC routine for thermostat

The left-right width of the block is the data width, which is 8-bits in this case. The top-bottom height of the block is the number of bytes. Now...back to the code.

There are two pieces of the code that actually do the work in averaging the readings. The first is from the label "MoveSmpls" up to, but not including, the label "AvgLoop." The second is from AvgLoop to the label "AvgDone."The first block of code basically prepares the data to calculate the average by making sure that the 64 most-recent ADC samples are placed within the RAM buffer. The second chunk of code, the one that actually averages the readings, starts at AvgLoop. Starting at the instruction before MoveSmpls, we note the x-register is initialized with a value equal to the buffer RAM location with the highest numerical value minus one: "TBufEnd-1." The first two instructions at the label MoveSmpls move the second-oldest sample from TBufEnd-1 to "TBufEnd." The sample at TBufEnd is now the oldest ADC sample in memory. The third instruction after MoveSmpls decrements the x-register so it points to TBufTop-2. Afterwards, the x-register is checked to see if we've averaged all of our samples, which is, in this case, 64. Since we haven't done that many yet, our code branches back to MoveSmpls.

If viewed as data movement within our rectangular memory block, there are 64 data movements sequentially moving toward the part of the memory block labelled "TimeBufr." Each data movement copies the contents of each RAM location to the RAM location just below. The most current ADC sample is always located at TimeBufr. This is schematically represented in Figure 1- .

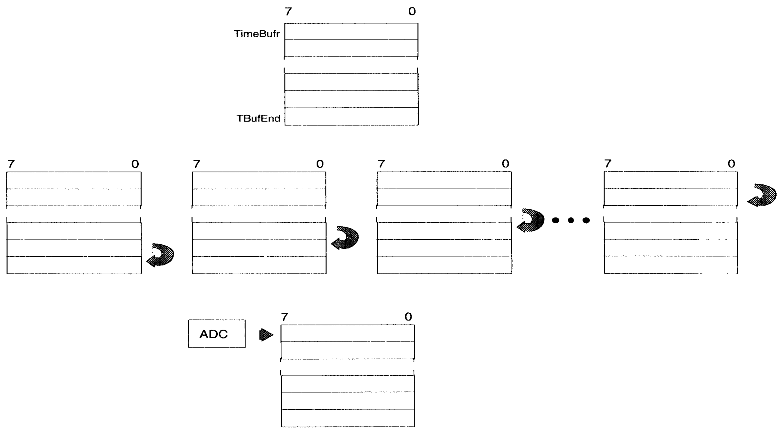


Figure 1-11
ADC Samples memory locations

```
***************************************************************
* Name:LowPass
*
* Purpose: To reduce variations in ADC data from thermal
*          sensors. A 64-point moving average is used.
*
* Use: pass latest sample from ADC to this routine via ACCA.
*
***************************************************************
 seg PAGEORAM

TempSmpl     rmb 1
AccMSB       rmb 1
TempCalc     rmb 1
TimeBufr     rmb 64
TBufTop      equ *

             seg Program

LowPass      sta TempSmpl    ;this contains the latest
                             ; ADC sample. Save it.
             ldx #TBufTop-1  ;start moving data from here.
MoveSmpls    lda 0,x         ;get sample
             sta 1,x         ;move it down the line.
             dex
             cpx #TimeBufr-1 ;check for latest sample location.
             bne MoveSmpls
             inx             ;leave x pointing to TimeBufr
             sta 0,x
             clr AccMSB      ;this preps the MSB of the
                             ; average result.
AvgLoop      inx             ;point to next location.
             cpx #TBufTop+1  ;check for last location
                             ; in buffer.
             beq Scale       ;leave if last sample.
             add ,x          ;begin avg calc.
             bcc AvgLoop     ;if no ripple to MSB thru
                             ; cy flag, then add again.
             sta TempCalc
             clra
             adc AccMSB ;this propagates cy flg
                             ; into AccMSB
             sta AccMSB
             lda TempCalc
             bra AvgLoop
Scale        lsr AccMSB
             rora
             lsr AccMSB
             rora
             lsr AccMSB
             rora
             lsr AccMSB
             rora
             lsr AccMSB
             rora
             lsr AccMSB
             rora
AvgDone      rts
```

Figure 1-12

64-point moving average routine

Following the flow of the code, if our MCU has made it to AvgLoop, then the x-register contains the value TimeBufr-1. Because we want to confine our averaging routine to the ADC data, which resides between TimeBufr and TBufTop inclusive, we first increment the x-register so it is pointing to TimeBufr instead of TimeBufr-1. The next instruction compares the value of the x-register to see if we've made it to the top of the RAM buffer where the newest ADC sample resides. We branch to the next chunk of code IF the x-register is pointing to the newest ADC sample located at the "top" of the RAM buffer (TimeBufr). In our next chunk of code, we begin the process of adding the ADC samples together. The HC05 CPU is an 8-bit machine that requires some extra steps for doing 16-bit addition. These steps are the instructions after the "add,x" and the instruction just before the label "Scale."

Averaging consists of two separate processes, addition of a list of numbers and division of the sum by the count of the numbers in the list. Now it is time to do the division. That's what the lines inclusively between the labels "Scale" and "AvgDone" do. The 8-bit architecture of the HC05 requires many instructions to divide. Following the instructions from Scale to the end, you'll see these perform six 16-bit shifts on the sum of the ADC values added in the previous part of the code.

To give you a sneak preview of things to come, let's mess around with the symbols we use to show data movements. This will show you that what we are talking about now has a direct application to the fun stuff later on. So, first separate each block and connect with arrows as shown in Figure 1-.

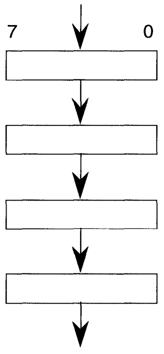


Figure 1-13

Now, turn the entire string of rectangles 90° counter-clockwise and reduce the size of the rectangles to make it easier to display. This results in Figure 1-.

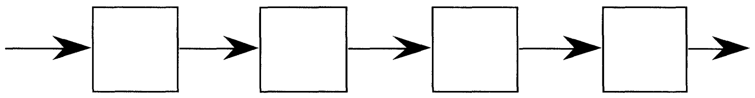


Figure 1-14

This view of the memory roughly represents the movement of the data by the averaging routine where each memory location is represented by a rectangle. At the output of each block, which represents each byte of data, draw arrows to a circle with a "plus" sign in it. One way of drawing it is in Figure 1- (a).

Below the summation bubble, draw another circle with an "x" in it...the symbol for multiplication (Figure 1- (b)).

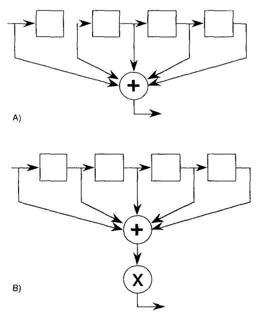


Figure 1-15

Let's pause at this point to review a rule of arithmetic we learned long ago but probably forgot. A rule in arithmetic says multiplication is distributive over addition. When multiplying a number by the sum of two other numbers it doesn't matter whether you multiply the individual numbers first and then add or add them first and then multiply. The result is the same. In symbols this is:

$$A(B+C) = AB + AC$$

For our diagram, move the multiplication bubble from its current location to each line entering the addition bubble, like Figure 1-.

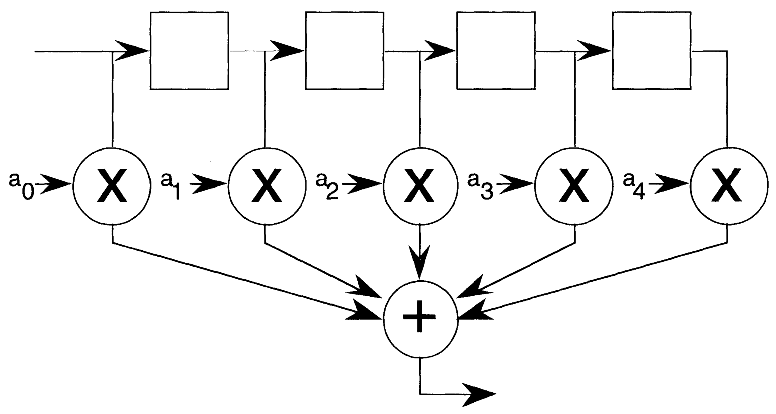


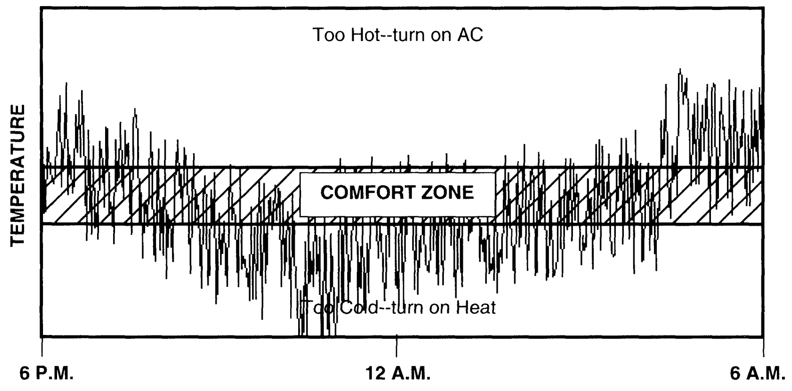
Figure 1-16
Lattice Diagram

Do you recognize this diagram? If you don't, don't worry. You'll see plenty of it if you work with DSP stuff. For those who recognized it, you know this as a **lattice diagram**. This type of diagram is often used to show the structure of digital filters. Does that mean that the averaging we just did is in fact a digital filter? Stay tuned.

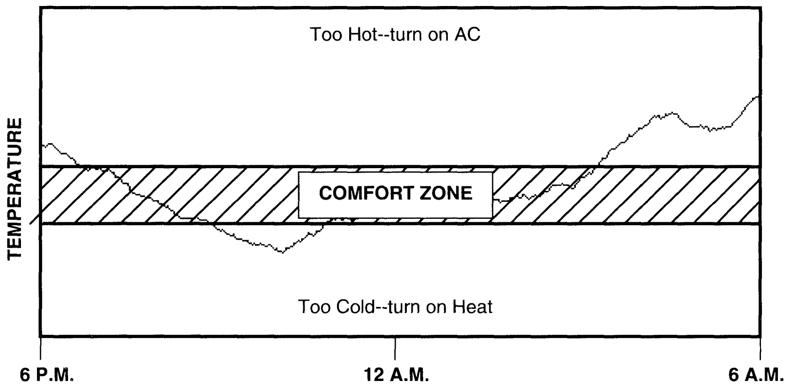
Thermostat Performance Results

After walking through the code, let alone writing it to begin with, we want to know "Does it work?". The answer to that question indicates writing a book has something in common with writing television sitcoms: the writer *never* introduces a problem or question without an answer...and usually the answer is the *obvious* one. So yes...of course our averaging routine works! To see how *e*ffectively the averaging process works on the actual thermostat data, look at the before and after graphs of the data in Figure 1- .

Had this thermostat been constructed with just any other technology, the solutions would have required more extensive hardware changes. *Although we could have opted for a pure hardware fix in this case, we didn't have to.* That's the whole point of this last exercise: it only took a few strokes of the computer keyboard to rewrite the source code to the MCU,



WithoutAverating



WithAveraging

Figure 1-17
Thermal sensor before and after averaging routine

program a new HC05 with the improved software, and then trade the socketed MCU for the new version. Compare this to pulling the thermostat out of the wall, desoldering discrete components, and spending untold hours changing resistor-capacitor values until the problem disappeared. Granted, you have to know about software, but the ease of working in the digital domain justifies the effort to becomes fluent with this software stuff.

ADC Basics

We could capture signals in digital form so the MCU's software could work on it through an **analog-to-digital converter** (**A-to-D**, or **ADC**). Most of us are familiar with these devices and many of us have used one. Even so, we need to discuss these in more detail to capitalize on the strengths of the digital domain and minimize construction problems. Digitizing analog signals creates artifacts in the process not present in the input analog waveforms.

Now, we will begin our discussion about analog-to-digital conversion by describing the constituent parts of the process, followed by a brief discussion about various converters.

The Sample-Hold Function

In most modern ADCs, the first step in our conversion is usually the **sample-hold function**. Although not absolutely necessary, this function is, to some degree, a function of the selected ADC as well as the signal we want to convert. Is this function required for your application? What happens if we don't use it? We will answer these questions in a minute. First, let's look at the process of sampling the analog world with our **sample/hold amplifier**.

Figure 1- shows an analog wave form. The sample/hold amplifier function is represented graphically by the following steps:

1) Move directly up from the "t" axis with a straight line until intersecting the input wave form.
2) Draw a rectangle, with a width determined by the distance between t-axis "ticks" and the height determined by (1), with its left-hand corner touching the analog wave form.
3) Repeat this process for each successive tick on the horizontal axis or until you run out of input wave form.

We have a wave form that looks like stairsteps representing the general shape of the input waveform. This stairstepped waveform is the sample-holdvalue that is fed to the input stage of the ADC.

Figure 1-18
Sampling an Analog Signal with Sample and Hold Amplifier

We assume most of the signals we want to convert to digital numbers include both **DC** and **AC** components. Let's feed a sine wave into an ADC. Taking the derivative of this function, we obtain an expression for the the sine wave's rate of voltage change per unit time. The maximum rate of voltage change occurs when a sine wave crosses the horizontal time axis. In quantitative terms, the rate of change of an A volt peak-valued sine wave amounts to:

$$V(t) = A\sin(2\pi ft)$$

$$\textit{Maximum rate of change} \equiv \left(\frac{dV}{dt}\right)_{t=0} = 2\pi Af$$

$$f_{\max} = \left(\frac{1}{2\pi A}\right)\frac{dV}{dt} = \left(\frac{1}{2\pi(16 \times 10^{-6})}\right)\left(\frac{5}{2^8}\right) \approx 194 \textit{ Hertz}$$

If we are using an 8-bit converter with a conversion time of 16 microseconds and a reference voltage of 5V, the maximum undistorted sine wave frequency that can be converted is less than 40 Hz, although our conversion frequency is much greater than the frequency of the sine wave!

The Generalized ADC

The analog-to-digital conversion process is nothing more than a translation of continuous-time, real-world signals into a granular, digital approximation. This digital representation is, in most cases, only an approximation of the actual voltage. In doing this translation, regardless of the specific ADC device used, a voltage comparator compares the input's analog voltage to a converging voltage internally generated by the ADC. When these two inputs to the comparator are equal, the analog voltage generator stops and the digital code associated with the internal analog voltage generator at that moment represents the analog signal. Let's see how this mechanism is applied with the various converters.

Successive Approximation Converter

One ADC is the **successive approximation converter (SAC)**. A device responsible for the character of the SAC is the **successive approximation register (SAR)**, which determines how the voltage is generated by the ADC. The elements of the SAC are: a) a voltage comparator, b) a **digital-to-analog converter (DAC)** and c) a clocked digital counter. The output of the comparator tells the digital counter to stop once the D/A output and the input voltage are roughly equivalent. The SAC's distinctive characteristic is the manner in which the DAC voltage is created: each "guess" at the input voltage, created by the DAC, is successively halved until equivalence is achieved. This technique allows a moderately fast (conversion rates > 1 µsecond) and moderate resolution (12-bits or less) of the unknown analog voltage.

IntegratingADC

Another popular converter used for much slower but typically much higher resolution conversions is the **integrating type converter**. This converter applies our generalized converter form in a unique way: it substitutes a charging capacitor for the DAC of our generalized ADC. The capacitor is charged by the input signal voltage and discharged by a constant current source. The time it takes for the voltage ramp, created from discharging the capacitor, to reach zero voltage is directly proportional to the analog input voltage that originally charged the capacitor. The linear voltage ramp is timed by a digitally clocked counter until reaching zero, at which time the digital counter stops. The resulting count shown by the counter is directly proportional to the input voltage. This type of A/D process is good for 10 µsec or higher conversion times and resolutions typically in the 18- and 20-bit range.

Flash Converter

Another conversion process is the **"flash" converter**, which uses a string of voltage comparators. Each comparator has one input tied to successively increasing reference voltages. The input voltage is applied to all of the other comparator's inputs that have all been tied together. The outputs of the comparators are sent through a digital logic array to produce a convenient output code as a function of the input voltage. Let's say we have three voltage comparators. The comparator's individual inverting inputs are connected to 1.25V, 2.5V, and 5V, respectively, so the comparators are referenced to 1.25V, 2.5V, and 5V. Let us then apply our input voltage to the other input. If the input voltage is less than the reference voltages, none of the comparators will trip. Once the input voltage increases above 1.25V, the comparator having the 1.25V reference will change from logic 0 to logic 1. The other comparators will remain unchanged until the input voltage rises above their respective reference voltages. Because its performance depends primarily upon the comparator's response speed, the flash converter is very useful for high-speed concertinos.

CHAPTER 2

The Digital Domain

MCUs processing digitized analog signals are powerful new tools for solving old and new problems alike. In Chapter 1 the microcontroller was introduced as a set of very useful features, including A-to-D conversion. Using these features we solved a noise problem in a common household thermostat without changing the hardware. In this chapter, we'll investigate some traits in the digital domain that make it a very appealing solution.

Listing the strengths and weaknesses of the digital domain is a straight-forward task. Some of the important subtleties of those strengths and weaknesses, however, are unclear. Consequently, we will briefly discuss and then illustrate each strength with actual applications. Afterward, we will discuss the weaknesses.

The Advantages

We'll begin by discussing benefits and strengths of the digital domain. Remember the example of the thermostat in Chapter 1 and how the digital domain worked to our advantage.

Doesn't Vary Over Time

It's hard *not* to notice the consistent performance from your unit once you've designed in the digital domain. For example, designing a remotely controllable "digital pot" — a potentiometer substitute that varies the amplitude of a digitized signal — you notice *the performance of the unit doesn't vary over time or temperature*. The process of varying a signal's amplitude consists of multiplying an incoming signal by a value representing the amplitude, or "volume" setting, you want. There's not much to drift or change in that situation. Multiplication is multiplication today, tomorrow, or next year. That's important to designers who want their designs to work "first time and every time" for everyone else as well as themselves. The same cannot be said for an analog implementation, which uses something like a transconductance multiplier for multiplication. Every example of such devices requires the trimming of offsets for many applications which can drift with time, temperature, and vibration. In addition, the strategic placement of a temperature-sensitive resistor is often required to compensate for the transistors' inherent drift that occurs with changes in temperature. Such analog circuit traits are manageable, but *if* we can reduce drifting parameters, then let's do it. With any carefully implemented digital domain design, sources of parametric drift are eliminated completely within specified error bounds.

Doesn't Vary Between Units

The next step with our digital pot design is to produce the product. It's in this next phase where the advantages of working in the digital domain are apparent. Even your company's accountant could appreciate this point: *parametric changes between units are predictable and bounded.* Translated into English, this means the differences between products can be negligible or, at best, known in the design phase. From an accounting perspective this means more profit from more sales of a better product, fewer service returns, and a better reputation for your company.

Less Expensive Than Analog

Many times *implementing a design in the digital domain can be less expensive than the equivalent analog implementation.* Using our digital pot example, however, the analog implementation is less costly to build than the digital version.

Unnatural Responses

We have assumed the digital pot example is just a straight translation of the analog pot function. Another advantage of the digital domain is *the ability to achieve "unnatural" responses including extraordinary analog-like responses.* The word "unnatural" in this case is meant to imply a system response not easily implemented with analog components. So, even though a single $1 pot could replace our more expensive digital example, would the analog pot be an equally good solution if there were additional "unnatural" performance criterion? That depends on what it is. If the analog pot must respond to the commands from a personal computer (PC) via its RS-232 port, which of the two makes sense? Right, our digital pot. An analog pot doesn't to respond to your PC.

Performance Easily Changed

A final advantage of the digital domain is illustrated with this example. Let's assume that your customers said they preferred a logarithmic taper to the digital pot instead of the linear taper product you currently ship. No problem. Just change the software in the MCU. *Perform-*

ance is easily altered with a digital domain implementation. Whether you're addressing feature enhancements or correcting problems with a product's operation, you will appreciate the easily altered performance of the digital domain. Although the performance of some analog circuits can be easily modified, component values must be changed to alter the circuit's performance. It is possible with a digital domain implementation to make significant enhancements without changing a single component. Try *that* with an analog circuit!

Digital Domain Strengths Illustrated

We've been somewhat superficial and academic in our discussion about the strengths of the digital domain. Let's now bring it down to earth with a real-world application: creating a magnetic tape noise reduction system.

"Noise 'R' Us" Recording Studios

Suppose I am an owner of a small sound recording studio that friends and clients use to tape their musical talents. The studio has a 16-track magnetic tape machine suffering from the evils of tape hiss, a high frequency sound produced from the tape's physically passing over the tape machine's playback head. Digitally-based recording systems with performance like a compact disc make tape hiss a problem of the past.

A digital recording system, however, is still too expensive for me to justify. We want our system, however, to sound as much like commercially available CDs.

Fortunately, in recent years much headway has been made in reducing the type of tape noise plaguing our system: Dolby Labs and a company by the name of dbx have both created noise reduction systems that effectively reduce tape hiss so it's inaudible. There's only one minor hitch: money, *my lack of money*. As nice as the commercially available noise reduction circuits are, they are expensive. Dolby Labs makes a system that is excellent at reducing the tape hiss and, at $1200 per tape channel, very good at reducing my available funds. It was the need for low-noise recordings and the high cost of the commercial noise reduction systems that is leading me to design my own noise reduction system. To design a workable system, I had to first understand how noise reduction works and then choose a specific implementation.

Simple Noise Reduction

Tape hiss is a function of the current state of the music's signal-to-noise ratio (SNR). If the music is loud, a listener can't hear the tape hiss through the music — the tape hiss is "masked." If the volume of the music is low, compared to the tape hiss, then a listener hears the tape hiss. Consequently, the masking effect from a high signal-to-noise ratio (SNR) is an effective way of reducing the audible impact of tape hiss.

The SNR is a way to compare the level of the music to the level of the noise...tape hiss, in this case. Several record producers believe noise reduction is *not necessary* for rock and roll. Why? Because rock'n'roll tends to have consistently high signal levels. This is particularly true of the "heavier" rock'n'roll tunes; there's so much music that the music effectively covers up or "masks" the tape hiss at any given time. You could say the SNR of rock'n'roll is typically very high (some classical music fans would refer to the rock'n'roll as noise). Recording only high-energy rock'n'roll is one way, and perhaps the simplest, to reduce the perceptible noise in recordings. But this doesn't work in our hypothetical situation.. Some of my clients like "bizarre," i.e., non-rock'n'roll, music such as jazz and R&B. Consequently, I had to find some other way to maximize the SNR of my tape system.

Another method that can be used to address the tape hiss issue is shown in Figure 2-1. In this noise reduction (NR) system, the high frequencies are boosted, for example, +10 dB before the signal is recorded to tape. On playback if we introduce a 10 dB loss (-10 dB) at the same frequencies we boosted, the net result is the signal on the tape will sound the same as before. This is because the net effect of boosting and then cutting the same frequencies in this complementary way produces the original signal. This is analogous to an amplifier stage which boosts overall signal gain by a factor of two only to have a successive stage introduce a gain factor of .5. The net effect is the signal emerges apparently unaffected, i.e., at unity gain, at the output of the last stage. This "boost then cut" form of noise reduction does the same thing except we are boosting and cutting only *parts* of the signal, only in the same area of the frequency spectrum where the tape hiss resides. In this way, during the playback the tape hiss levels are cut back when the frequencies are cut. Since we have effectively restored the original signal and cut the tape hiss, by 10 dB in this example, *we have effectively increased the SNR of our tape recording system by 10 dB*. This looks like a practical way to reduce tape hiss when recording clients don't want to record high-energy rock'n'roll.

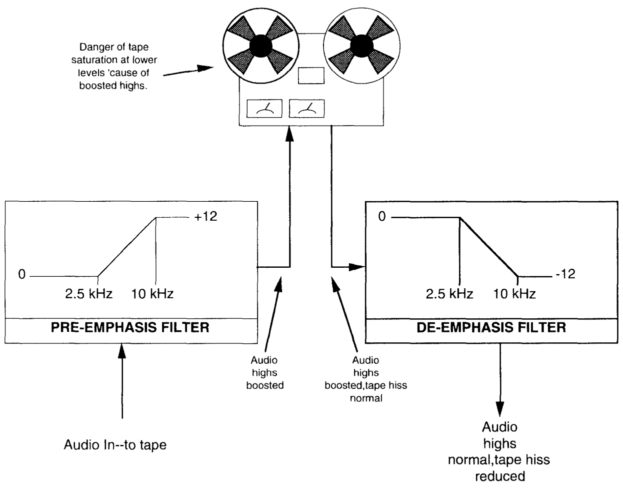

Danger of tape saturation at lower levels 'cause of boosted highs.
+12
0
2.5 kHz 10 kHz
PRE-EMPHASIS FILTER
Audio highs boosted
Audio highs boosted,tape hiss normal
0
-12
2.5 kHz 10 kHz
DE-EMPHASIS FILTER
Audio In--to tape
Audio highs normal,tape hiss reduced

Figure 2-1
Tape Noise Example

There is a significant flaw with this boost/cut noise reduction system: the tape's limited dynamic range as a storage medium. This means the boosted high frequencies will distort the tape at a lower absolute level than the lower, unboosted frequencies. So although the boost/cut scheme maximizes the SNR, we reduce the effective **dynamic range** of the system because of the boost. Fortunately, there is a way around this problem to make the boost/cut method usable. An **amplitude compression** can be used to record the signal with the boosted high frequencies while reducing the chance of overloading and distorting the tape. Here's how it works. As the energy of the music increases, an automatic volume control, the amplitude compressor, turns down the volume of the music in a predetermined way. As an example, if the music increased its loudness by a factor of two (6 dB for voltage measurement), we could have this compressor decrease the volume by 1.414 (3 dB for voltage measurement). The net result is the signal going to the tape has increased by only a factor of approximately 1.414 (a 2:1 ratio of input to output change in dB) instead of by 6 dB before we compressed the signal.

What does this all mean? Simply this: by using a compressor we can reduce the chance of tape overload from boosted high frequencies. Good, but compressing the signal like this introduces a new problem. It sounds "unnatural." Fixing this is analogous to boosting and cutting specific frequencies of our signal during recording and playback. While compressing the signal during record to avoid premature distortion, we use the complement of compression, called expansion, to restore the original signal's dynamic range on playback.

Encode/Decode NR System

Figure 2-2 shows a block diagram for implementing an encode/decode type of tape noise reduction (NR) system. Within the box marked "process" is the compressor and preemphasis filter to encode the signal going to the tape. This system view of the NR system allows you to see the actual mechanics of how the NR works without getting bogged down in the details. Basically, the process block produces a known output, a(t), when presented with the signal, s(t), that we want to record. The signal that actually gets recorded onto tape, however, is the sum of these two, that is, a(t) + s(t). During playback, we change the configuration slightly of the process and summing amplifier. The point of the decode process is to retrieve the original unencoded signal s(t). By inverting the phase of the process block and taking its input from the output of the summer, we get a decode process that is the mirror image of the encode process and that strips the encoded part of the signal, a(t), from the signal stored on the tape. Remember, the process function produces a(t) if s(t) is presented at its input. It is also true if s(t) is at the output of the summer and the phase of the process is inverted that the system-level equations work out:

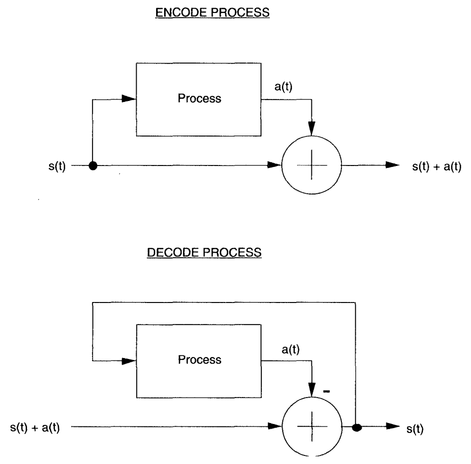


Figure 2-2
Encode/Decode Tape Noise Reduction

Once I determined the overall configuration of the NR system by similar reasoning, I needed to identify the characteristics of the process block. This was relatively easy because the process employs both compression and **preemphasis** to do what I needed it to do. The details of the encoder is shown in Figure 2-3 .

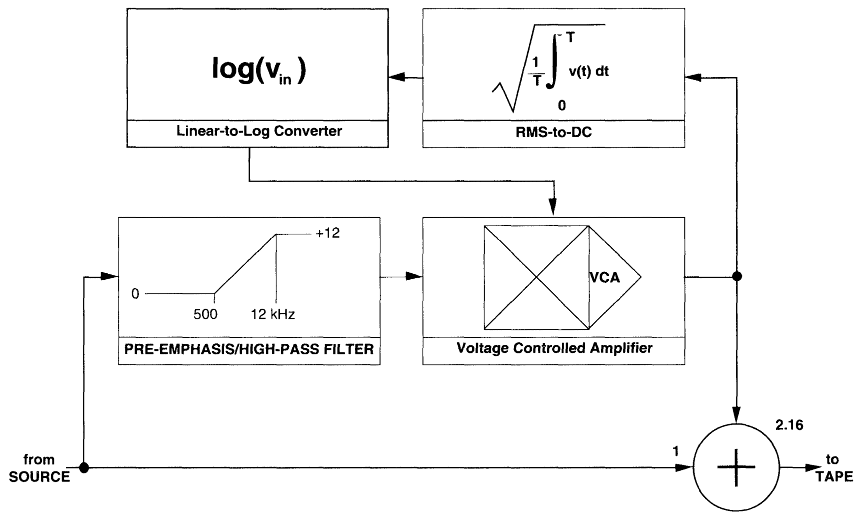

log(v_{in})
Linear-to-Log Converter
$\sqrt{\frac{1}{T}\int_0^T v(t)\,dt}$
RMS-to-DC
+12
0
500
12 kHz
PRE-EMPHASIS/HIGH-PASS FILTER
VCA
Voltage Controlled Amplifier
from SOURCE
1
2.16
to TAPE

Figure 2-3
Noise Reduction Encoder Detailed Block Diagram

The function blocks marked **VCA**, **RMS-to-DC**, and Log() make up the compressor. The preemphasized signal is fed to the input of the compressor, while the output of the compressor is summed with the unaltered signal. This produces the composite signal, s(t) + a(t), which is recorded on tape. Analogously, the decode loop uses the same elements; the only difference is the input to the compressor is now the output of the summer rather than the signal to be encoded.

Performance Requirements

After identifying the key concepts for a noise reduction system, I established a set of performance standards for the NR system. They were:

- The NR system shall reduce audible tape playback noise to a level competitive with commercially available units.
- The artifacts from mistracking between the encode/decode processes will be kept to an absolute minimum or eliminated completely.
- The NR system will have the lowest possible cost, specifically, less than $25 per channel.
- The tape shall be "listenable" without the NR decode process. Listenable means that any equalization funnies can be corrected with standard studio gear. This is a precaution in case a client takes one of our master tapes to another studio. I'd sure hate to ship my NR system with the client's tape!

The next step in the design process was to identify a specific circuit implementation.

The NR Prototype

The analog-based prototype has two noteworthy features: an RMS-to-DC converter, with built-in logarithmic output for driving the VCA, and a thermal tracking resistor. The resistor is required to keep the RMS-to-DC converter's output voltage from drifting.

Performance Of The Prototype

I breadboarded the circuit to verify the performance of the unit. All the testing I did was subjective in nature. The dynamic nature of noise reduction makes it difficult to test with standard lab equipment. The important performance criterion is how good the NR sounds to the ear. I used a cassette deck as the tape machine under test. The results were very gratifying because it performed to my *subjective* expectations.

NR Implementation Problems

If the subjective performance of the NR system is so good, then the design must be O.K., right? No. Unfortunately, the performance of the circuit *could* falter sometime in the future because of mistracking between the encode and decode modes of operation. This type of mistracking can be induced by environmental conditions and results in certain circuit parameters drifting. Remember our encode/decode equations in Figure 2-2 on page 34 assumed perfect encode/decode arithmetic; there were no error terms shown in the math.

A tracking error between the encode and decode process is exhibited as a loss or accentuation of certain frequencies during tape playback. Nonetheless, I wanted to reduce as much as practical. You have probably experienced this type of phenomena with your home and car stereos, if you record tapes at home and listen to them in your car. The mismatch between the encode and decode filters of the two different machines causes the differences in the frequency response between your home cassette recorder and your car's playback unit.

One way to correct this problem is to use components or processes having low errors. Another way is to trim out the error. I prefer using circuits that don't require trimming because the components used to trim can become sources of drift themselves. This left me to find devices or techniques without the inherent errors. While I was surveying the industry for such devices, I realized that there was yet another problem to consider even while looking for solutions to the drift problem: price.

In many ways, the drift issue may be trivial when compared to the price issue. Even with a perfect solution to my tape hiss problem, like a digital recording system, the availability of that solution is a function my VISA credit line. Like it or not, the price issue as legitimate a consideration in product design as the selection of technologies.

So, here was my dilemma: my prototype worked well, but included a $23 RMS-to-DC converter inherently prone to temperature-related drift. Both the cost and inclination for drift were at odds with my performance specifications. That spec included a total product cost per channel of $25 or less for all the electronics. The spec also required the design must minimize, if not completely eliminate, tracking errors. An additional, "hidden" spec requirement was I didn't want to spend all my time selecting the components for this project. This project was beginning to look too much like work! But I stumbled upon the solution in my "defunct" projects box.

Several months ago, I developed an MCU-based audio device using one of the HC11 variants I obtained for around $11 apiece in some quantity. This device was to control a Voltage Controlled Amplifier (VCA) using upon certain information contained in the audio. The device, an audio compressor/limiter, was the functional equivalent to the amplitude compressor needed in the NR system. The question is: if I use it in the NR system, would it address the performance issues?

The MCU As An Analog Circuit

Figure 2-4 gives a visual hint about how an HC11 applies to this application. By replacing the RMS-to-DC chip with the HC11, both the RMS-to-DC conversion is replaced as well as the linear-log weighting of the signal required to properly drive the VCA stage. Would this part substitution address the drift and cost issues? Yes. Here's how: the RMS-to-DC process is implemented by performing simple math on ADC results. The results of these calculations are causal, bounded, possibly convergent, and possess error margins that are calculable. This on account of the nature of the arithmetic — even with quantifying the arguments. If you have any doubt about this, perform an RMS calculation on a set of numbers tomorrow in your spare time. Repeat this same calculation several times thereafter under different conditions, but using the same set of numbers. It doesn't matter when or where you perform the calculations, you'll get the same answer. It's almost absurd to suggest there would be a difference from the original calculation. In addition, even the numbers in the calculation are changed or the precision of the arithmetic, the results may be predicted to lie within particular outer bounds. If so, then the question is, "Is the bounded error sufficiently low or can it be made sufficient for our application?." By subjective test, under a variety of conditions and after guesstimating a quantitative impact, the answer is a resounding "Yes!"

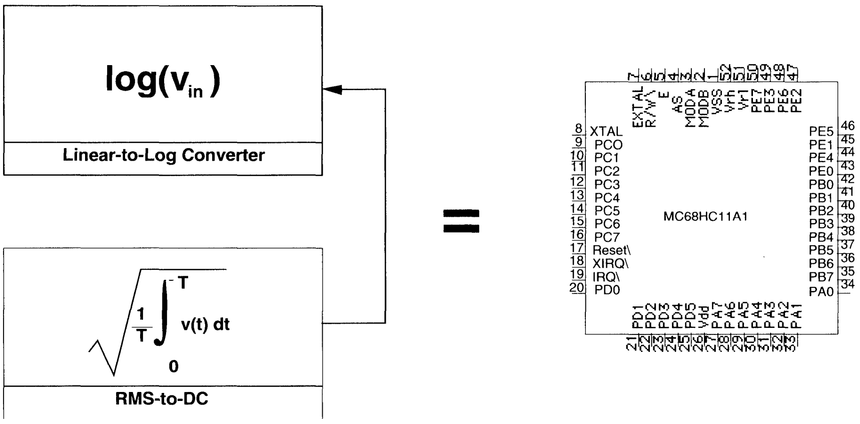


Figure 2-4
68 HC11 Can Replace Both the RMS-T0-DC Chip and the Linear Log Weighting

The linear-to-log converter has the same features the RMS-to-DC process does: the process is bounded and has a predictable error contribution. This, too, turned out to be acceptable for this application even when combined with the errors of the RMS-to-DC conversion process.

HC11 NR And Digital Domain Advantages

Before our discussion about the NR system, we talked about the strengths of the digital domain. These strengths include:

a) Minimizing parametric changes within a unit over time and temperature,

b) Predictable and bounded performance between units,

c) Lower costs than an equivalent analog version, usually having a subset of analog performance,

d) “Unnatural” responses, including extraordinary analog-like responses, and

e) Flexible performance.

The digitally-controlled NR system certainly takes advantage of these features. If you agree our RMS-to-DC converter and linear-to-log converter functions will not drift or change original performance when implemented digitally, then we can agree that *parametric changes relative to time and temperature are minimized within the same device* because of the digital implementation. It's not a large leap in logic to assert the same characteristics that give us (a) also give us (b). *Parametric changes between units are predictable and bounded.* The latter point is really a restatement of the former.

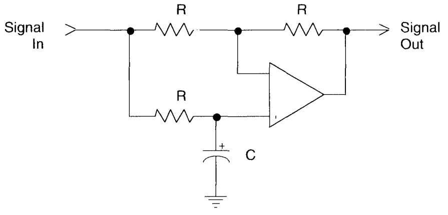
The cost of the HC11-based NR system per channel, around $15, is much less than the analog counterpart even though it includes the cost of the DAC required to interface to the VCA. Compare this $15 price tag with the $23 price tag for just the RMS-to-DC converter. This exemplifies another advantage of the digital domain: *a digital implementation is less expensive than analog but usually with a subset of analog performance.* In this case, the digitally implemented RMS-to-DC converter does not have the 10 MHz bandwidth of the monolithic RMS-to-DC converter. The **bandwidth** of the digitally implemented system is lower than the analog version...but so is the price.

The HC11-based NR system is also a good example of how the digital domain can be used to obtain "unnatural" responses. In the NR system, the unnatural response is a nonlinear, logarithmic response. The same function may be accomplished using analog components like the RMS-to-DC chip and its log output, but the accuracy of the output is affected by temperature. This is only one example. Many functions in the analog domain could be accomplished with great difficulty or many parts that are much more easily accomplished in the digital domain. This is because of the nonlinear or complex nature of the required function. Functions like waveform generation fall in the "it's-easier-to-do-in-digital-than-in-analog" category. Good examples are waveforms that are not particularly periodic or that have other characteristics not be easily processed or generated by continuous-time devices.

Finally, digital system performance is more easily changed than that of an analog system. While the NR system we've described is useful to illustrate several advantages of using the digital domain, another process is useful to illustrate how a digital system is more easily modified. And that's an analog building-block, the all-pass filter.

Easily Modified Performance

The **operational amplifier** schematic in Figure 2-5 shows an **"all-pass" filter**. We won't discuss the practical uses of the all-pass filter. Instead, we will contrast the circuit's ability to manipulate certain parameters of its input signal. Then we will indicate some possible uses for the circuit.



$$|T(jw)| = \frac{|1 - jwRC|}{|1 + jwRC|} = \frac{\sqrt{(1)^2 + (-jwRC)^2}}{\sqrt{(1)^2 + (jwRC)^2}} = 1$$

$$\boxed{T_{gd} = \frac{2RC}{(wRC)^2 + 1}}$$

Figure 2-5
All Pass Filter

The equations in the figure reveal two performance aspects of the all-pass network. One, quantity, the magnitude of the **network function**, is equated to a value of one. This means this circuit produces a signal whose amplitude is identical to the input. Normally, an amplifier with this characteristic would be called a **voltage follower**. But this is not a voltage follower. The second mathematical expression the figure shows is the time delay of a signal passing through this circuit will be affected as a function of the input frequency of that signal. The expression given is called the **group delay** of the network.

Let's ignore any reason why we would change the time delay of a signal — we'll get back to that later. How would we duplicate this same type of function in the digital domain? One answer is a simple, single-stage delay illustrated in Figure 2-6 . Let's walk through the description of this diagram. The input signal is digitized with an ADC, processed by the CPU, and output to a DAC. To realize a delay, the processor must perform three steps:

(a) Move data from a RAM location to the DAC.

(b) Move data from the ADC to the RAM location.

(c) Go back to step (a).

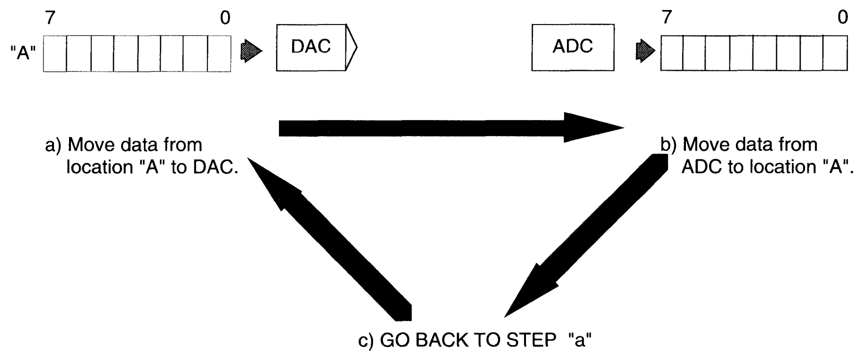

7
0
"A"
DAC
ADC
7
0
a) Move data from location "A" to DAC.
b) Move data from ADC to location "A".
c) GO BACK TO STEP "a"

Figure 2-6
Single Stage Delay

That approach is straightforward. How long is the time delay? It's however long it takes the CPU to perform steps (a), (b), and (c). How long is the delay of the analog circuit compared to this? Well, I'm not sure but *I* sure don't want to figure it out! But the expression for the analog circuit's time delay is much messier than the digital version. What if we wanted to change the time delay value? Basically we only have to shuffle the ADC samples in RAM around a bit more than we are currently doing.

As shown in Figure 2-7 , to achieve a longer time delay, lengthen the RAM buffer from what was previously one location to "n" locations and employ the following procedure:

(a) Transfer the last ("nth" sample) RAM location to the DAC,

(b) Move each previous sample into the next higher sample position of the RAM array, and

(c) Fill the top position with the latest ADC sample.

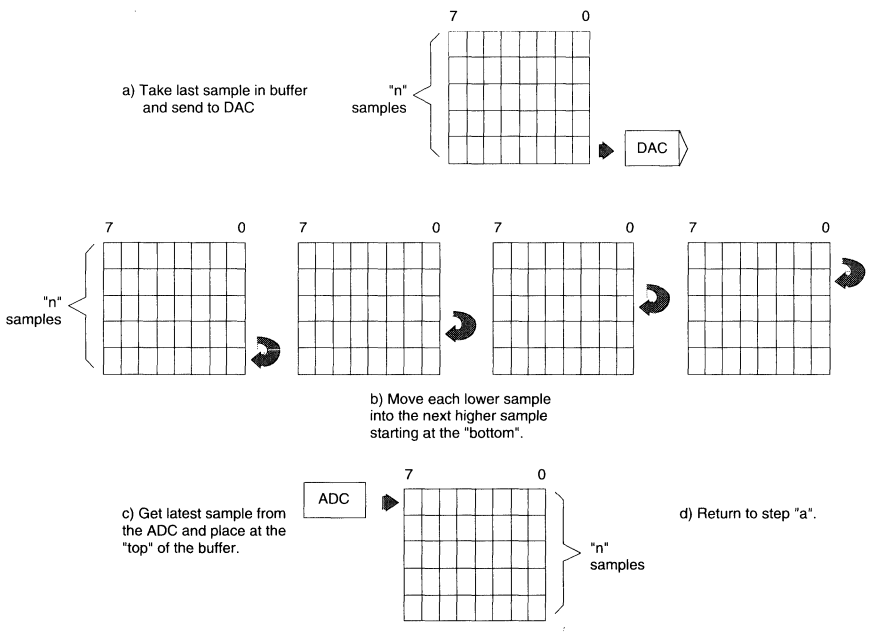

a) Take last sample in buffer and send to DAC
"n" samples
7
0
DAC
b) Move each lower sample into the next higher sample starting at the "bottom".
c) Get latest sample from the ADC and place at the "top" of the buffer.
ADC
d) Return to step "a".

Figure 2-7
Longer RAM Buffer is Used to Achieve Longer Delay

To change the delay time of this digital delay, we only have to change the number of bytes in the RAM array for the ADC values. The source code for implementing this digital delay line on the Motorola MC68HC05 MCU is shown in Figure 2-8 .

```
*************************************************************
* Name:TimeDelay
*
* Purpose: To illustrate a 64 sample digital delay.
*
*************************************************************
            seg PAGEORAM

TempSmpl    rmb 1
TimeBufr    rmb 64
TBufEnd     equ *

            seg Program

TimeDelay   sta TempSmpl     ;this contains the latest
                             ; ADC sample. Save it.
            lda TBufEnd      ;Get very last sample.
            sta DAC          ;Send it to DAC.
            ldx #TBufEnd-1   ;start moving data from here.
MoveSmpls   lda 0,x          ;get sample
            sta 1,x          ;move it down the line.
            dex
            cpx #TimeBufr-1 ;check for latest sample location.
            bne MoveSmpls
            inx              ;leave x pointing to TimeBufr
            lda ADCResult
            sta 0,x          ;put latest result into buffer.
            rts
```

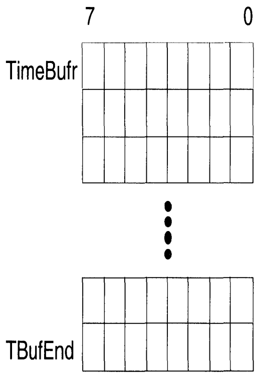


Figure 2-8
Implementing Digital Delay Line with 68HC05

How do we change the time delay characteristics in the analog all-pass filter in Figure 2-5 (page 2-13)? A circuit with an approximate time delay of 100µsec, for most frequencies, is shown in Figure 2-9 . Although the expression for the first analog all-pass filter's time delay (Figure 2-5) wasn't too complex, it was nothing compared to *this* all-pass network's expression. The transfer function for this multi-opamp circuit is so messy we won't present it here, let alone attempt to modify the circuit elements to obtain a different time delay! Can it be done? Yes. Would *I* want to do it? No. The digital implementation is much more convenient to modify.

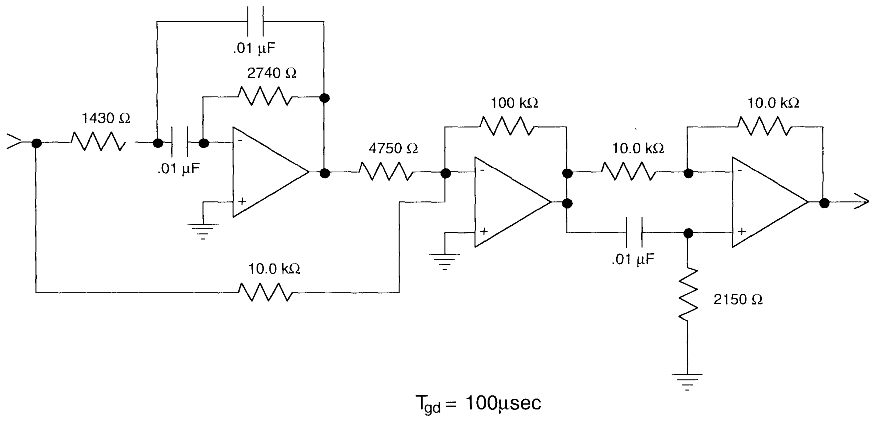


Figure 2-9
100 μsec delay all-pass filter

Uses Of Time Delay

We've skirted answering the question, "Why would anyone want to delay a signal?" Here's One answer that's to the point is: to delay signals. Another, is many analog filters mess up a signal's group delay that can be corrected by an all-pass network. In the past when passive audio signal equalizers were the only game in town, some people believed the signal would sound better if the weird time delay characteristics of some equalizers were corrected.

A more contemporary use of the time delay function is to hide the fact that most rock'n'roll singers can't sing. Yet, even this is not the primary use for such a device.

Time delay is an integral part of the process of averaging data. We saw this first hand with the thermostat described earlier in this chapter. But at that point we didn't call the data movement "time delay," and that's exactly what we're doing with the data. Look at Figure 2-10 . The left side of this figure is the HC05 source code from Figure 1-11 in Chapter

1, and the right side is the averaging routine from our discussion on filtering thermostat data. The heavy-lined rectangle highlights the common code segments between the time delay routine and the averaging routine. The time delay code is exactly duplicated in the lowpass filtering routine. ***Time delay is an important and integral part of the filtering process***.

```
**************************************************************
* Name:TimeDelay
*
* Purpose: To illustrate a 64 sample digital delay.
*
**************************************************************
             seg PAGEORAM

TempSmpl     rmb 1
TimeBufr     rmb 64
TBufEnd      equ *

             seg Program

TimeDelay    sta TempSmpl     ;this contains the latest
                              ; ADC sample. Save it.
             lda TBufEnd      ;Get very last sample.
             sta DAC          ;Send it to DAC.

             ldx #TBufEnd-1   ;start moving data from here.
MoveSmpls    lda 0,x          ;get sample
             sta 1,x          ;move it down the line.
             dex
             cpx #TimeBufr-1 ;check for latest sample location.
             bne MoveSmpls
             inx              ;leave x pointing to TimeBufr

             lda ADCResult
             sta 0,x          ;put latest result into buffer.
             rts
```

Figure 2-10
Time Delay is an Integral Part of Averaging Data

A Perfect Solution?

We hope you are convinced, if not enthusiastic, about "doing things digitally." After all, digital techniques appear to have a lot of advantages over many analog methods. So, it appears there is an ideal solution to most problems. Well...not quite. There are some disadvantages to using digital processes. Some are practical considerations like cost, and others are problems inherent in the process of digital conversion and manipulation. Let's take them one at a time. These are not listed in order of importance since their significance depends on your system.

Limited resolution of data. Even the output of high resolution A-to-D converters, like 18- and 20-bit converters, are still only approximations of the real-world. The process of sampling with limited resolution and finite sampling intervals produces artifacts not present before converting the analog signal. Usually, the artifacts are unwanted and must be minimized to avoid corrupting the system's specified performance.

Limited bandwidth. The maximum signal frequency that may be processed by a digitally sampled system is considerably lower than that of an analog system. Many analog system frequency responses run into the gigahertz region, while high-resolution (8-bits or higher) systems are pushing the current technology to run in the area of tens of megahertz. Because of this technology/performance ceiling, we must deliberately minimize the bandwidth of the incoming signal to avoid sampling artifacts. This may severely limit a purely digital implementation, depending on the application. The bandwidth issue is commonly linked to the issue of limited resolution. As the speed of data conversion goes up, the resolution tends to go down. Whereas you may obtain 30-megasamples per second from an 8-bit converter, a 16-bit converter may not produce the same conversion rate. The two issues are inversely related.

Limited dynamic range. Dynamic range is the measure of a system's ability to handle very small and very large amplitude signals. There are really two reasons the dynamic range is limited with digital systems: the inherent resolution limitations of the ADC process and the arithmetic characteristics of the given CPU. If an 8-bit converter is used, for example, then the maximum ratio of the smallest to largest signals that can be digitized is 255:1. A 16-bit system, on the other hand, can handle a maximum ratio of 65535:1! The dynamic range is significant. If an input signal is larger or smaller than the dynamic range, the signal will be either completely ignored or severely distorted since the signal isn't large enough for the digital system to "see" it.

Dynamic range is also determined by the arithmetic capabilities of the CPU. Given identical throughput and functional performance requirements, an 8-bit CPU will handle a wider input signal dynamic range than a 4-bit CPU and produce less erosion of the dynamic range. For example, let's add the number "1" twenty times to itself. After the sixteenth addition, the 4-bit CPU would either overflow or invoke a special overflow handling routine. Doing the same thing on an 8-bit CPU requires nothing special until the 255th addition! Although there are other, more subtle arithmetic processes like convergent rounding that affect the dynamic range and precision of a digital system, quantizing the arguments degrades dynamic range the most.

High cost for high resolution and throughput systems. Basically, the more closely we're able to digitally represent a signal and/or at a higher bandwidth, the more money it is going to cost. When conversion time is constant, then the higher the resolution conversion, the greater the cost. With system throughput remaining constant, the higher the bandwidth you need, the greater the cost. These are determined by the practicalities of making high-speed, high resolution ADCs and CPUs.

CHAPTER 3

The Representation of Signals

Repre-sentationof Signals

Now that we know how various ADCs perform their magic, it's time to talk about some different ways of looking at wave forms. This is a topic that many of us take for granted, and yet it is as fundamentally important to understanding signal processing as Ohm's law is to understanding AC and DC circuits. If you're a practicing engineer or even just have some experience poking around a circuit with an oscilloscope, you've seen a signal's familiar voltage versus time representation on the 'scope. Well, whether you're experienced or not, we will now *stop* taking this representation for granted. There is information we've overlooked to help us understand digital manipulation of signals a bit better. At the worst (or at best, depending on your view), we'll avoid some problems because we won't make certain assumptions.

As just mentioned, voltage versus time is probably the most common view. Our lab instruments continually enforce this view by showing us how signals change over time. Consequently, we develop skills and, particularly, thought processes centered around this view. How many times have you watched the rise time of a square wave degraded by an "integrator" similar to Figure 3-1? Or observed the characteristic tilt of the high-pass differentiator on a square wave or a tone burst as shown in Figure 3-2? In these cases, the type and extent of filtering is evaluated according to the time domain view. There is another way. You might ask, "Why do we need another view? After all, our trusty 'scope usually helps us solve the problem and what's wrong with my thought processes anyway?!" As we will see, the idea is to make things easier for *us* and to make *us* more effective. The concepts we are about to discuss will be very helpful as we start to move from the analog realm into the digital world.

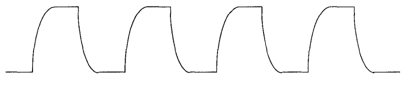


Figure 3-1
An Integrated Square Wave

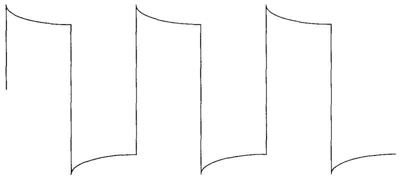


Figure 3-2
A Square Wave Through A High Pass Filter

There are really many different ways of looking at the same signals. In fact, in many cases different views can provide much more information about the signals in question than the 'scope view or, as it is also called, the time domain description. There is at least one new and exciting way of looking at signals in addition to the standard time domain view. For example, let's recall the log function. Aside from giving us grief during our first calculus courses, this function proved to be very useful in the early days of computers. Then a computer could only add and subtract. (According to all you purists out there, it is really only addition, since subtraction is addition with signed numbers.) Any multiplication and division had to be performed by sequential additions or subtractions. You may even recall the restoring and nonrestoring division algorithms, which used iterative subtractions and were popular for such machines.

In any case, a much faster and non-iterative way to do the same type of functions was to map the arguments into the *log domain*, that is, to use the identity: Log(xy)=Log(x) + Log(y). By taking the exponential of both sides, we're left with $xy=10^{(\mathrm{Log}(x)+\mathrm{Log}(y))}$. Roughly translated into English, this means that we can determine the product of two numbers (x and y in this case) by *adding* the log of the numbers and exponentiating the result. The advantage to this approach, in this example, is we can *multiply* two numbers by using the *addition* capability of our computers. "So what?!", you say. What does this have to do with the representation of signals? Simply this: by changing our view of the numbers, i.e., by *mapping* into the log domain, we are able to quickly perform a function that our computer otherwise isn't capable of performing quickly. In other words, we can do something relatively easily with the numbers which we couldn't do easily before, *just by remapping the problem* from one domain to the other.

So, what is this new and exciting way of viewing signals? Sad to say, it's neither new nor very exciting, but it is extremely useful. It's the *frequency response* view. Yes, the frequency response view can be a very compact and useful way of characterizing a system's performance. If you're a stereo enthusiast, you know your equipment's frequency response is a key measurement of your system's ability to accurately reproduce your favorite music. One thing you may not have realized, however, is the frequency response of your stereo is a shorthand method of describing the stereo's response to a whole bunch of time domain signals. For example, my stereo, if excited with a 50 hertz sine wave of .775 Vrms, will put out a 50 hertz sine wave of such-and-such size. Imagine doing this for a collection of sine waves of various frequencies. It might look something like Table 3-1.

Table 3-1

25Hz	50Hz	100Hz	200Hz	1kHz	4kHz	10kHz	20kHz
.700	.780	.905	.825	.790	.803	.653	.420

Table 3-1 certainly illustrates the ability of my stereo to handle each frequency, but it also took a while to fill in this table. The table is also hard to read and understand. A better way to show the same information might have been to graph the magnitude of the sine waves, at the output of my stereo, versus its frequency value, as shown in Figure 3-3. Easier to read, isn't it? These two ways of showing stereo performance are also equivalent in meaning. Notice that the *frequency response plot* is merely another way of describing how the stereo responds to sine waves.

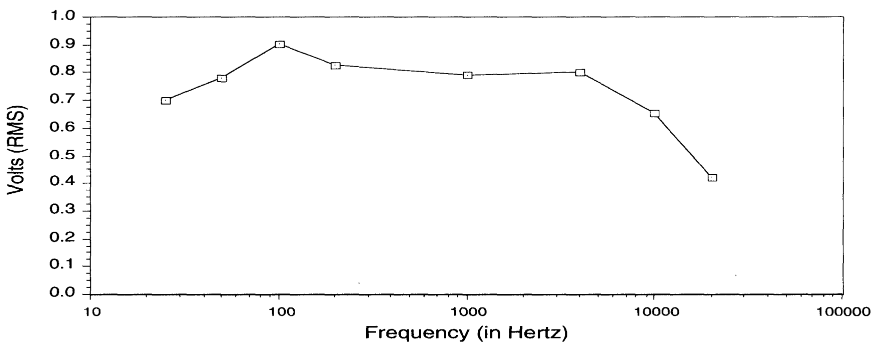


Figure 3-3
Frequency Plot of a Stereo Amplifier

If there were nothing to be gained by the frequency response view, it would not be worth discussing. Since *we* would never put anything wasteful in this book, this view must offer a lot of potential gain. Earlier in this chapter, we discussed the use of averaging to effectively filter noisy data. Would you believe there is an easier way to filter the noise which uses our new frequency response representation? The answers to the following two questions give a clue to this easier way:

- What answer do you get when you multiply a number by 1?
- What answer do you get when you multiply a number by 0?

The answer to the first question is, of course, the number itself, and the second answer is zero. Now, imagine taking a frequency response, such as shown in Figure 3-4(a), and multiplying it by the rectangular shape, as shown in Figure 3-4(b). In other words, for any frequency that has nonzero amplitude in both figures, we plot the first frequency response in the resultant graph. Otherwise, we plot a 0 in the resultant graph. After we perform this short procedure, we end up with the graph shown in Figure 3-5.

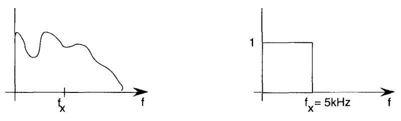


Figure 3-4
(a) An Input Signal Spectrum and
(b) A Low Pass Filter Spectrum

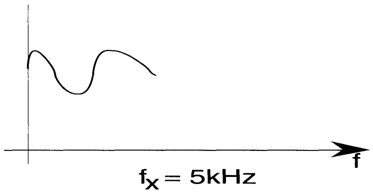


Figure 3-5
Result of Multiplying Fig. 3-4
(b) by Fig.3-4 (b)

The plot of Figure 3-4(a) contains only those frequency components at or below 5 kHz. After multiplying the original frequency response, containing frequencies up to 15 kHz, by a rectangular pulse with a width of 5 kHz, we end up with a frequency response containing frequency components no greater than 5 kHz. In other words, we have *low-pass filter*ed the original frequency response plot. At this point, you would probably guess that it is possible to filter a signal by simply multiplying it's frequency representation by a rectangular shape. Your guess would be perfectly correct! Couple this idea with your knowledge of how effectively and easily we can multiply numbers with microprocessors these days.

Before we start multiplying, there are actually two challenges at this point:

- Transform the real-world signals from their normal, analog state into a form that a microprocessor can manipulate.
- Transform the digital time domain data into a frequency domain representation.

We already covered converting analog signals into digital ones a little earlier. Let's discuss time and frequency domain conversions.

We have, at least *implicitly*, agreed that the scope and *frequency response* representations are equivalent. If they really are (and they are!), then the process of going from time-like data into frequency-like data must also be reversible. So maybe instead of transforming the time-based data into the frequency domain, we could translate our rectangular frequency response into time-based data. Either approach is valid because the time and frequency domain representations of the signal are equivalent. From a practical view, the point is to identify which representation is easier and/or cost effective to manipulate. In the earlier thermostat example, we actually implemented the filtering function in the time domain. We transformed (unknown to most of us at the time) the desired frequency response shape into a set of time-based procedures consisting of additions and multiplications. If we had graphically presented the actual frequency response shape (which we hadn't done at that point), you would have seen something like Figure 3-6 - much less stark and extreme than the rectangular frequency shape. This "rounder" frequency response shape ultimately translates to a more gradual slope in the transition band — a less effective filter.

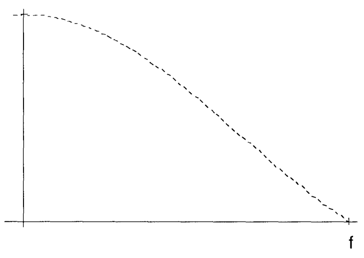


Figure 3-6
Low-pass Frequency Response of Thermostat Averaging Filter of

Time To Frequency And Back Again

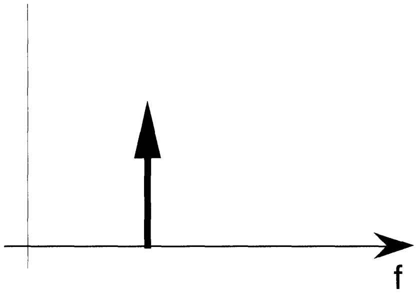
What is it that allows us to move from the time-data to frequency-data and vice versa? As we saw with the logarithm example, there are many potential remapping techniques. However, the most widely accepted mapping technique between the frequency and time domains is the *Fourier Transform* (FT). A function has two representations: time domain and frequency domain. The combination of these two representations is called a *transform pair.*

Believe it or not, you don't really have to know the details of the Fourier Transform for it to be useful to you. Though it would probably benefit you to work through the math of an FT at least once, we won't go through it here. You may review the FT in **Chapter 4** which shows the actual mechanics of the Fourier Transform. To become somewhat conversant in manipulating signals in the digital domain, we need to discuss some of the results of applying the FT.

By now, we've presented several concepts possibly new to you. Without jumping ahead in the book, you may have to take the concepts and their manipulation on faith. However, if you are squeamish about "taking things on faith," then please read ahead to **Chapter 4**. There you will find the detailed logical argument to thoroughly understand and accept the concepts. Otherwise, continuing here will at least expose you to some very important concepts and "grease the skids" for later. Just seeing the application of the FT, without the nitty-gritty mathematical details, can be immediately useful to you. We will now present some of the more useful *transform pairs* along with some important signal constructs.

The Impulse

An *impulse* is a concept represented by a vertical arrow:



The impulse is a periodic event the converter uses to initiate taking a sample of the incoming analog wave form. It also can represent a sample clock, like that used by the ADC process. Actually, a single impulse is not used to model the ADC, but a "train" of impulses is used. The train of impulses is usually regarded as stretching infinitely in time. Two useful characteristics of these impulses are:

1) The magnitude can be 1, and when multiplied by the input signal gives us a rudimentary understanding of the process of digitally sampling analog signals.
2) Also, the time-based infinite *train of impulses* gives us a corresponding infinite *train of impulses* in the frequency domain, and the distance between impulses is given by the inverse relationship f=1/T, where "f" denotes the frequency and "T" the period of the time signal.

Looking at this graphically as in Figure 3-7 , we see as the distance in the time-based train of impulses is decreased, the distance of the frequency-based train of impulses increases. Conversely, as the distance between the time domain impulses increases, the distance between the frequency domain impulses decreases. The importance of this characteristic will become evident shortly. Keep it in mind.

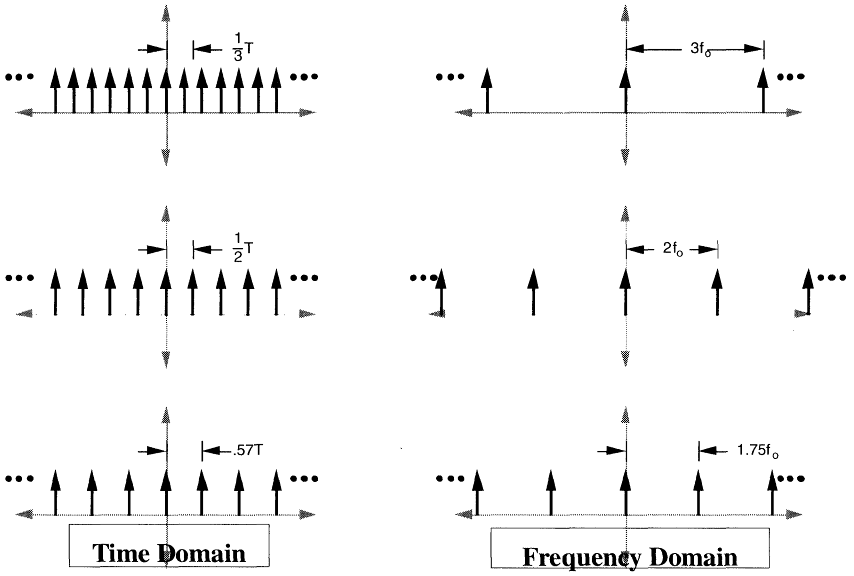


Figure 3-7
Relationship Between Time & Frequency Impulses

Multiplication

Earlier we discussed the frequency response of a stereo system. We went through a brief exercise of multiplying a frequency response by a rectangular shape. We just discussed how multiplying a train of impulses by an analog signal models the process of sampling that signal. Based on just these two scenarios, the process of multiplication in either the time or frequency domain is useful. But are they really the same thing? That is, is multiplication in the frequency domain the same type of multiplication in the time domain? Let me say definitively: yes and no. An explanation is in order.

The mechanics of multiplication are the same regardless of the domain you are working in. The same old multiplication you and I have been doing for many years now is implemented the same in either domain. That is not to say if we multiply something in one domain we are also simultaneously multiplying in the other domain. On the contrary. Remember we are using the FT to go between the time and frequency domains. Because of this mapping process, multiplication in one domain translates into a different operation in the other domain, just as adding in the log domain results in multiplication in the linear domain. The results are always the same. It's just that the steps, or mechanics of the processes are different depending upon which domain you are working in.

Convolution

If you derive an output in the time domain by multiplying a signal by a function, what is the equivalent operation used on that signal and function in the frequency domain? It's called convolution. If you applied convolution to an input and a function in the time domain, what is the corresponding operation used in the frequency domain? It's multiplication. In DSP lingo:

$$y(t) = x(t) * h(t) \Rightarrow Y(f) = X(f) * H(f)$$

and

$$Y(f) = X(f) * H(f) \Rightarrow y(t) = x(t) * h(t)$$

Armed with this technotrivia, you can liven up your convoluted technical discussions with all kinds of bad puns like this one. Once you know the mechanics of a convolution, you can put it to work livening up your MCU-based product designs. Let's look at the mechanics of convolution by looking at a special case involving a train of impulses.

Simply stated, *convolving a wave shape with a train of impulses duplicates that wave shape for every frequency there is an impulse*. For example, let's convolve, in the time domain, the following two signals: a rectangular pulse and a train of impulses as shown in Figure 3-8. Now apply the convolution "rule" we just stated on the rectangular pulse and the train of impulses. Take the rectangular pulse of Figure 3.8(a)and "paste" its shape at every frequency having an impulse (Figure 3.8(b). Notice that just as the input rectangular pulse was centered around the vertical axis, a rectangle is centered at each impulse after convolution. If done properly, we end up with the graph in Figure 3-9. Not very informative or useful you say? True...but that's just an example. Let's take it one step further with an example with a more useful outcome.

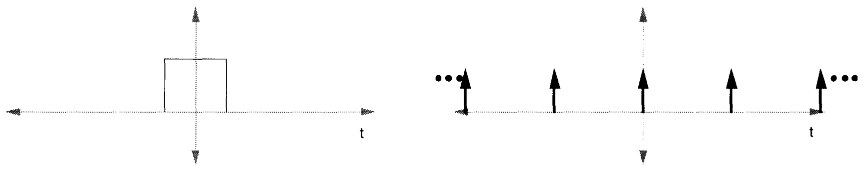


Figure 3-8
Rectangular Pulse (a) and a Train of Impulses (b)

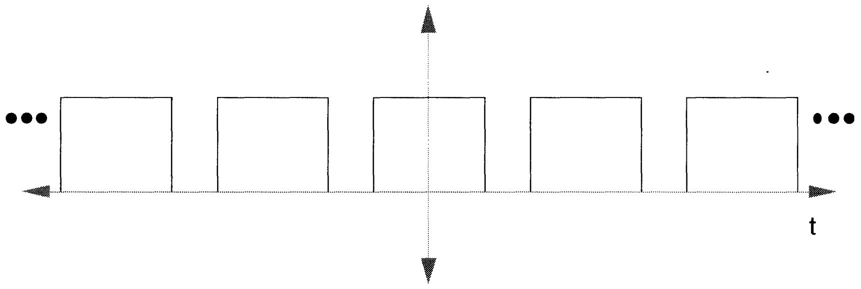


Figure 3-9
Convolution of Fig. 3-8 (a) & 3-8 (b)

Take the sine wave in Figure 3-10(a) and multiply it by the train of impulses Figure 3-10(b). The result is like that in Figure 3-11. Obviously we have a train of impulses with the original sine wave's shape. Now, convolve this sinusoidally-weighted train of impulses with the rectangular pulse shown in Figure 3-12. What's the result? The time-domain plot shown in Figure 3-13. You astute observers will recognize this as the sample-and-hold function applied to a sine wave. The rest of us, particularly the management types, will describe the resultant wave form as "a bumpy sine wave." Who am I to argue?

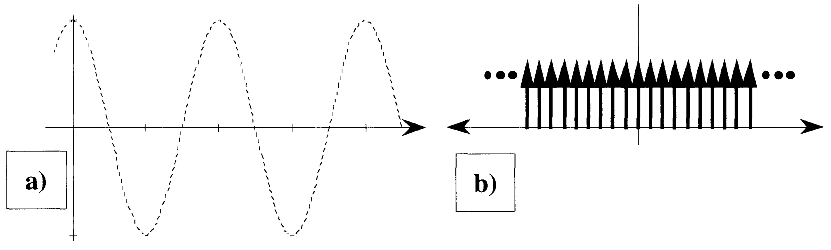


Figure 3-10
A sine wave (a) and a train of impulses (b) to sample it

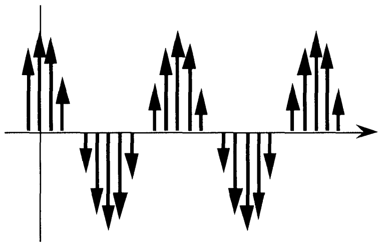


Figure 3-11
The Sine Wave of Fig. 3-10 Multiplied by a Train of Impulses

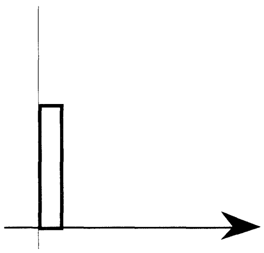


Figure 3-12
A Short Rectangular Pulse

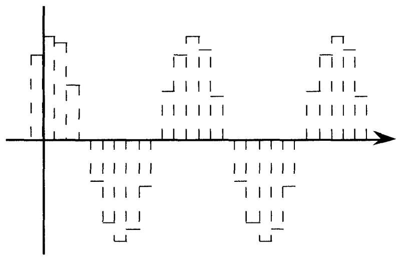


Figure 3-13
Sine Wave of Fig. 3-10 (a) after Sam-

Nevertheless, this convolution's result closely represents a real world sample-hold function sampling a sine wave. As we discussed earlier, the sample-hold function is usually an analog to digital converter's first step in the process of going from the analog to the digital world. By using the impulse's characteristics and convolution mechanics, we can model what happens to a signal in the analog to digital process.

A Quick Convolution Example

If you understand the concepts presented thus far, you actually have enough information to start processing real-world signals. Convolving functions in the frequency domain is the real-world processing for which we're prepared. Summarized below are the FT's features and relevant facts discussed so far about the frequency domain :

Important point #1: Convolving a rectangular shape with a train of impulses results in duplicating that rectangular shape everywhere there is an impulse.

Important point #2: A rectangular shape centered around the vertical frequency axis in the frequency domain is a low-pass filter.

Important point #3: Multiplying a signal by a train of impulses in the time domain represents the digital sampling process for an analog wave form.

Given these tidbits of information, try to determine the answer to the following problem:

> Convolve a 1000-hertz wide rectangular shape centered at zero frequency, with an impulse train in the frequency domain. What minimum distance between frequency impulses is required to guarantee no overlap between the resultant rectangular shapes?

The answer follows this reasoning:

First, the 1000-hertz wide rectangular shape is centered around zero frequency, so it stretches 500 hertz on either side of the vertical axis. Once the impulse train is convolved with the rectangular shape, it will stretch 500-hertz on either side of each impulse.

We know that convolving a train of impulses with another function basically duplicates that function everywhere there is an impulse. Therefore, the result of this impulse train convolved with this rectangular shape is a series of 1000-Hz rectangular shapes, centered at each impulse of the impulse train. The minimum distance between the duplicated rectangular pulses will be 500 hertz, for a given impulse, plus 500 Hz for each adjacent impulse. As you probably know, 500 Hz plus 500 Hz equals 1000 Hz. Thus, in this case, the minimum distance between impulses, to avoid overlap of the rectangular shapes, must be *at least* 1000 Hertz.

What have we just reasoned through? It's called the *Nyquist Criteria* for digitally sampled systems. You've probably heard of it. It's an important operating consideration when working with digitally sampled systems. A typical textbook definition of the Nyquist Criteria states that our system sampling frequency must be greater than twice the highest frequency component of the input signal. Why? When this "rule" is not followed (as in the example above), the frequency spectrum associated with one impulse starts to "collide" with the frequency spectrum associated with the impulse next to it. Overlapping spectra is not usually something that we want, because overlapping spectra means the spectra are interfering with each other. In fact, the overlap creates frequencies not present in the original spectrum. These products of overlapping spectra are called aliased components.

By applying the rudiments of the Fourier transform, we've seen all digitally sampled systems are bound by two important concepts: The Nyquist Criteria and aliasing. Despite their importance, we have been able to deduce their existence, and we've even guesstimated a quantitative system impact.

The Rectangular Pulse

Most of this material may appear to be "black magic," but I assure you the rules and transform pairs we have discussed are a direct consequence of the FT. We've only looked at signals graphically, but a mathematical approach supports the FT as well. Without actually performing the mathematics of the FT, it's pretty tough to predict the *time-frequency pairs*. Time-frequency pairs are the two functions that represent a signal or operation in both the time and frequency domain. For example, given a rectangular pulse in the time domain, the FT transformation results in a rather strange-looking, obtuse function called the sinc function (pronounced "sink"). It looks like the graph of

Figure 3-14. Do you recall our discussion about the frequency response of the stereo system? We talked about a frequency response multiplied by a rectangular shape. Although at that point we hadn't explicitly discussed the FT, we mentioned filtering in the frequency domain by multiplying a signal's frequency response by a rectangular shape. We ignored at that point *how* to get into the frequency domain to begin with. We did hint, however, that this filter could be performed in the time domain if a time domain version of the frequency domain's rectangle function could be multiplied by a signal.

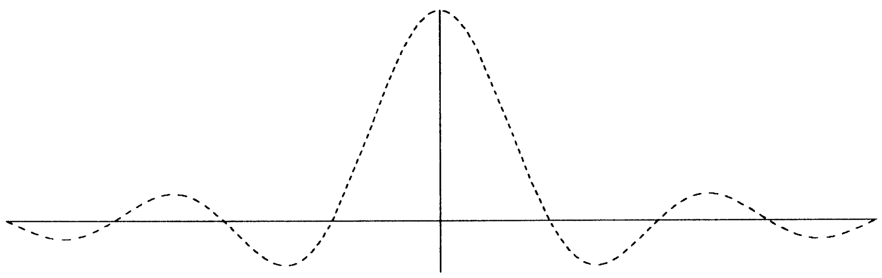


Figure 3-14
A Graph of the Sinc Function

Based upon our discussion up until now, the following two operations will lowpass filter a signal:

- Multiplying a signal's frequency domain representation by a rectangular shape, or
- Multiplying a signal's time domain representation by the sinc function.

As stated earlier, we usually choose the filter implementation by which representation is easier to manipulate and/or is cost effective. A digital filter, however, is usually implemented in the time domain.

From the discussion on transform pairs, we have the basic background required to build a digital filter.

The Sine And Cosine Functions

This transform pair is one you've already seen but probably didn't realize it. In fact, we opened our discussion on transform pairs with it: the FT of the sine or cosine function results in an impulse in the other domain. For a time-based sinusoid, the frequency plot will be an impulse located at the frequency of the sinusoid like that shown in Figure 3-15. As you may remember, the difference between the sine and cosine functions is slight. They are really the same wave form but 90° out-of-phase with respect to one another. To be truthful, both sine and cosine may simply be represented with a single impulse arrow as shown in Figure 3-15. That's only part of the story, however. There *is* an equivalent frequency domain representation to designate the phase differences between the two, but we'll leave that for further discussion until **ChapterXX**. It is sufficient at this time to represent either time-based function by a single impulse in the frequency domain.

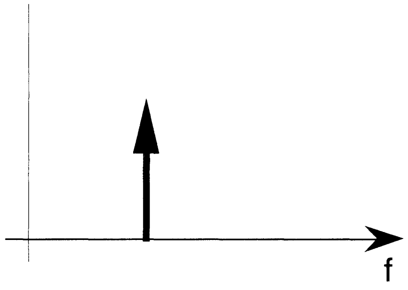


Figure 3-15
The Fourier Transform of a Sine or Cosine Wave

Frequency Representation of Direct Current (DC)

A DC wave form is really no wave form at all. Electrical engineers understand DC to mean the current or voltage amplitude does not vary with time—there is zero frequency. This produces a horizontal line on an amplitude versus time plot as shown in Figure 3-16. The equivalent frequency domain is a single impulse located at the zero frequency point.

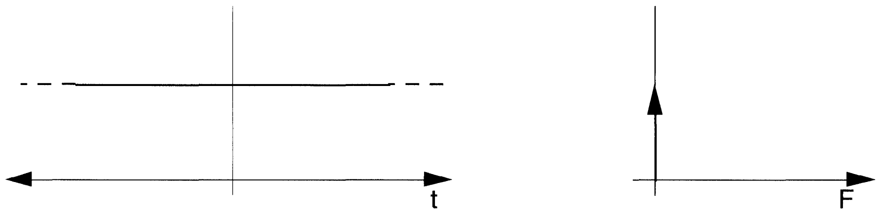


Figure 3-16
DC and Its Spectrum

The consequence of this is twofold: a) A train of impulses that has an impulse located at the zero frequency point means there is a DC component in the time domain signal, b) A horizontal line in the frequency domain is plotted in the time domain as an impulse at t=0. This is an important point. The implication is a horizontal line in the frequency domain suggests that all frequencies are at the same amplitude. We interpret the single time domain impulse as a very short-lived signal. A short time domain impulse produces a frequency plot containing all frequencies.

Time Or Frequency: It's All The Same

Up to this point we've freely interchanged a signal's representation in the time and frequency domains and vice versa. That is, if a rectangular pulse in the time domain results in a *sinc function* in the frequency domain, we've also assumed the opposite was true: a sinc function in the time domain produces a rectangular wave shape in the frequency domain. We haven't attempted to prove this, we've just casually presented this as acceptable. Fortunately, it is not only acceptable, it's correct. This pairing of wave shapes, regardless of the domains in which they reside, is called *duality* in signal processing lingo. This is actually one of the most important consequences of the FT. Keep it in mind as you would the transform pairs we've discussed - it's very important.

Other Important FT Stuff

We've only talked about one of the properties of the FT so far: duality. There are many more. We can get along well, for most day-to-day engineering work, without these additional details. This is primarily because some of the properties, like the linearity property, are so common in engineering that we assume it applies here too. Most of us haven't been concerned with other properties, like the time and frequency shifting properties, until now. What follows is a list of the properties with a short explanation.

Linearity property. The FT of the sum of two functions is equivalent to the sum the FT of each function.

Because the FT is a linear function, many of the other arithmetic properties also apply and often come in handy when solving problems. Two of those properties I've found particularly useful are the commutative and associative properties.

Symmetry property. This is another term for duality.

Time scaling property. This is one of the properties establishing the inverse relationship between the time and frequency domains. Basically, this property states that if our time variable "t" is multiplied by a constant "k" then the transformed "f" variable is divided by the constant "k." As it turns out, the amplitude of the transformed function is also affected by this property.

Frequency scaling. A complementary statement of the time scaling property starting at the frequency domain. You could "prove" this by applying the time scaling and the symmetry properties. In symbols, that is, frequency scaling symbols.

Time shifting property. This property is responsible for the fact that time delay in the time domain is equivalent to phase shifting in, you guessed it, the frequency domain.

Frequency shifting property. The concept of *modulation* is a direct consequence of this property, which states a frequency may be shifted by multiplying the corresponding time domain function by a certain exponential function.

Other properties. Even functions, odd functions, real functions, and imaginary functions are properties that, at this point, require little more than this quick acknowledgment. Although ultimately very important to those of us who will become deeply involved in signal processing, these are not absolutely necessary to apply signal processing to many engineering problems.

Putting It All Together

At the end of this book is a list of some really informative books (not including this one of course), where the curious among you may delve more deeply into these things.

We will now put our new knowledge to work by taking the concepts we have learned so far to develop a digital filter. We will develop the details of the digital filter we constructed for the thermostat that is the averaging filter. Although this is a very specific filter, the mechanics presented are useful for constructing any type of digital filter that does not utilize feedback in the filtering process.

Earlier in the chapter we said filtering an input signal could be accomplished by multiplying the input signal spectrum by a rectangular wave shape. But, while we didn't indicate how to obtain the input signal spectrum, we did infer that maybe we didn't have to. Instead, we assumed we could work in the time domain. By working in the time domain we avoid the transformation of an input signal into a frequency spectrum. Then we found the rectangular low-pass filter shape in the frequency domain was identical to the sinc function in the time domain. The rectangular shape and the sinc function are transform pairs. These concepts give us a good starting point to build a digital filter.

Let's recount these principles and state them the way you might see them in a signal processing textbook.

Multiplication of two frequency spectra in the frequency domain is equivalent to the convolution in the time domain of tthe inverse FT of each spectrum . This is the kind of statement you see in a signal processing textbook. It's short, concise, and confusing. We'll translate it.

Multiplication of two frequency spectra in the frequency domain. In the case of our low-pass filter, this means we will multiply the rectangular wave shape by the input signal spectrum. But we don't know what the input signal's spectrum is or how to get it. So, we're much better off working in the time domain. How?...

Is convolution in the time domain of the inverse FT of each spectrum. Earlier we discussed specific Fourier Transform pairs. We said the FT of a rectangular time-based pulse is a sinc function in the frequency domain. Because of the symmetry property, we also are able to say a rectangular shape in the frequency domain is a sinc function in the time domain. The process of translating a time domain function to its frequency domain version is usually referred to as "taking the Fourier transform" of the time-based functions. The process of translating the frequency domain function into its time domain version is referred to as "taking the inverse FT" of the function. We have been freely going both directions because we have been applying the symmetry property. Thus, all we have to do is convolve the sinc pulse with a time-based input signal to filter it.

The Thermostat Example Revisited

Simply stated, if our lowpass filter is the multiplication of a rectangular wave shape with the input signal's frequency spectra, then we can filter in the time domain by convolving the time-based input signal by time domain representation of the rectangle wave. In this case, it is the sinc function. But how do we convolve an input signal with a sinc function? We already have. We have already convolved an input signal with another function. Remember the thermostat averaging routine and the lattice diagram? Both the routine and the diagram demonstrate convolution in the time domain! Let's walk through the details of this thermostat filter implementation.

First, we digitally sample the output of the analog temperature sensor. We now know we may approximate a signal's digitization by multiplying the input waveform by a train of time domain impulses. Because we don't really know what the time or frequency domain output of the sensor looks like, we'll make something up. We'll assume temperature slowly varies, and for all intents and purposes, it's a DC value. To demonstrate the characteristics of the filter, we'll superimpose a 60 Hz sine wave on top of the DC output of the sensor. The output of the sensor would then look like this:

From the view of the sensor output, the functional interpretation of Figure 3-17 is the temperature is relatively stable. Unfortunately, "riding on top" of the sensor output voltage is some 60-cycle noise, possibly induced by the house wiring. The corresponding spectrum for this time domain wave shape is:

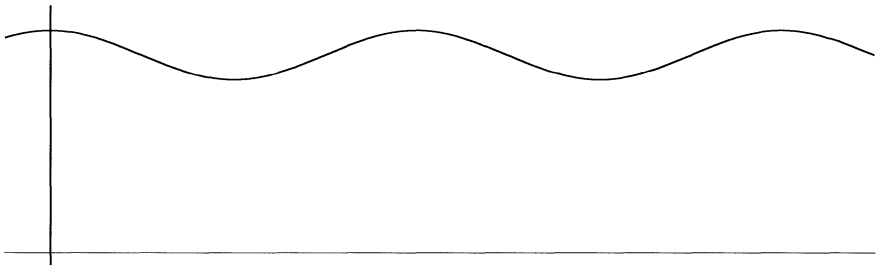


Figure 3-17
An Imagined Output of a Thermal Sensor Plus Noise (60Hz)

For the sake of brevity, we will designate this composite function TempNnoyz(t). The plot in Figure 3-18 shows an FT property and two of the transform pairs we looked at earlier. The figure reveals the FT's linearity property. Briefly, the linearity property states the FT of the sum of two signals is equivalent to the sum of each function's FT.

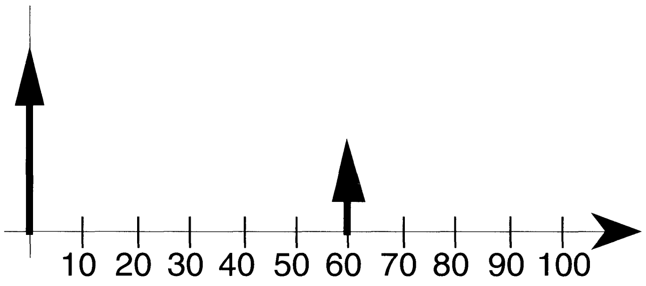


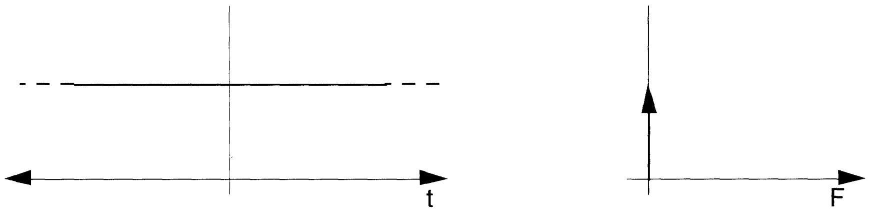
Figure 3-18
Spectrum of Thermal Sensor Output Plus Noise of Fig. 3-17

The first transform pair is shown in Figure 3-16. It is a constant time function, and its corresponding impulse is located at the frequency plot's origin, like so:

The second transform pair is a sine wave and its corresponding FT, an impulse located at the frequency of the sine wave:

By applying the linearity property, we end up with the composite spectrum TempNnoyz(f) in Figure 3-18.

Now that the output voltage of the sensor



has been defined, the signal must be digitized. In this case, we will use a sampling rate 10 times greater than the frequency of the 60 Hz sine wave. Remember,

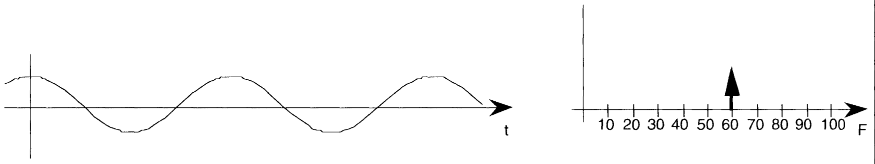


Figure 3-19
A 60 Hz Sine Wave and Its Spectrum

digitally sampling a signal can, for our purposes, be adequately modeled by multiplying the input signal by a train of impulses in the time domain. The time domain plot of the sampled temperature profile looks like Figure 3-20 on the following page:

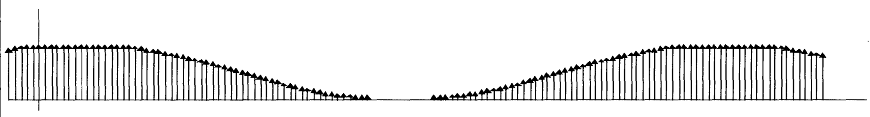


Figure 3-20
Digitally Sampled Thermal Sensor Output

We'll call this TempNnoyz(nT). Notice the use of "nT" to distinguish the signal's sampled timebase from the continuous time designation of "t." What does the spectrum of this digitized function look like? You are correct if you guessed the composite function of Figure 3-18 duplicated wherever there is an impulse in the frequency domain. This function is Figure 3-21 and occurs because time domain multiplication translates to frequency domain convolution. And convolving a wave shape by a train of impulses duplicates the wave shape everywhere there is an impulse. To sample TempNnoyz(t), we multiplied it by a train of time domain impulses which resulted in TempNnoyz(nT). The frequency domain translation of this time domain activity is to convolve the FT TempNnoyz(t), which is given by TempNnoyz(f), by a frequency domain impulse train. This process of sampling translates the continuous time function into a stream of numbers. We then need to feed these into the process represented by the lattice diagram. Let's look at the lattice diagram we developed earlier:

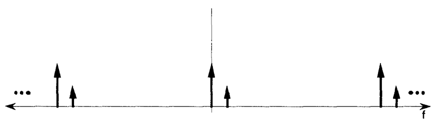


Figure 3-21
Spectrum of Sampled
Sensor Output

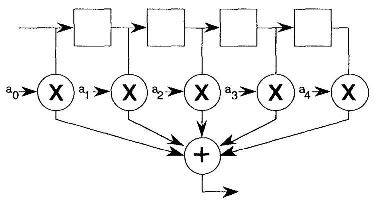


Figure 3-22
A Lattice Diagram

Data moves from left to right. The samples coming from the ADC enter on the left, and they leave our averaging process on the right. As shown in the lattice diagram, each piece of data held in RAM is first multiplied by a number or *coefficient*. If you'll remember from our discussion placement of this coefficient is a consequence of the distributive property of multiplication over addition. Suffice it to say in our thermostat averaging example:

a) There are 64 "blocks" in our main data flow in the lattice diagram.

b) All 64 of the coefficients happen to be the same. They all had the value of 1/64.

The data moves through the lattice diagram so that each data sample is shifted left to right, once each sample clock. We could view the output of each block, called *taps*, as being one sample clock apart. Since each sample coefficient has an amplitude of 1/64 and a sample clock delay, we could picture combining these two basic elements for every tap. The plot would look like this:

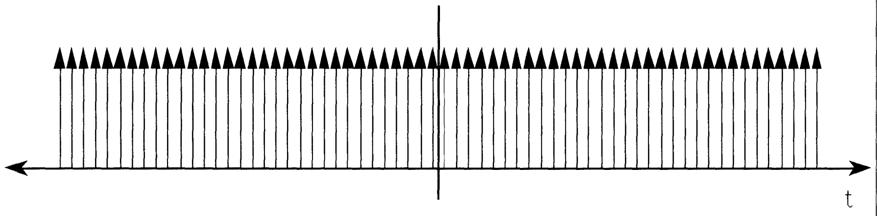


Figure 3-23
Representation of Lattice Diagram Coefficients as a Sampled Rectangular Pulse

Does this look familiar? It looks like a rectangular pulse multiplied by a train of impulses. Since we are convolving this wave shape by the digitized input signal, TempNnoyz(nT), this convolution also looks like the input signal's FT being multiplied by the FT of the sampled rectangular pulse. The spectra for a sampled rectangular pulse looks like this:

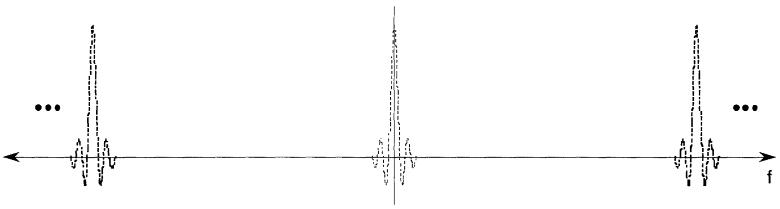


Figure 3-24
Spectrum of Sampled Rectangular Pulse of Fig. 3-23

So, how does this filter work? If we look more closely at Figure 3-24 , we can see the details. Figure 3-25 shows figure3-14 (the sinc function) magnified a little. The graph's left hand side, the "negative" frequencies, is cropped from view. In addition, it only shows its characteristic sinc shape up to where it first passes through the horizontal axis. We did this to simplify the final step of multiplying the input spectra by this wave shape. Compare Figure 3-25 sinc-type spectrum with the shape of the ideal lowpass filter:

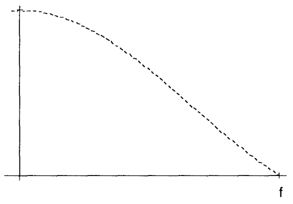


Figure 3-25
Sinc Function Low-pass Filter

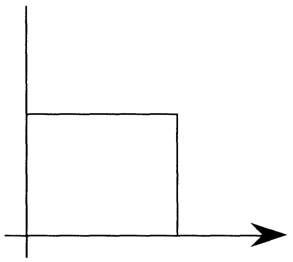


Figure 3-26
Ideal Low-pass Filter Frequency Response

The two spectra in this figure are similar. The amplitudes in the lower frequency are affected less that those in the higher frequency. But the sinc function's gradual slope implies it is going to be less effective as a filter. The final step in understanding this averaging type filter is to multiply the input signal spectrum by the filter's spectrum:

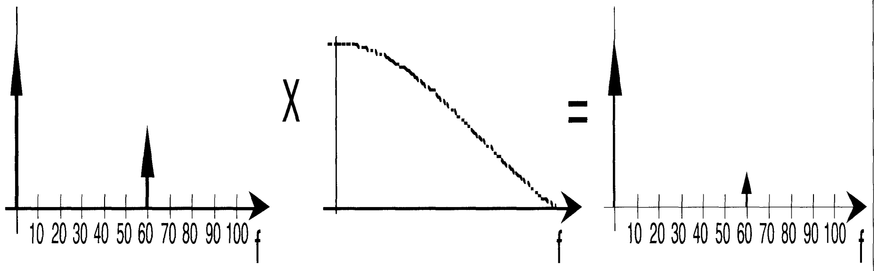


Figure 3-27
Multiplying Thermal Sensor Output Spectrum (Plus Noise) by Sinc Function Results in Reduced Noise Component

We can see from the graph that applying this averaging process has significantly reduced the magnitude of unwanted noise in our system.

Digital Strengths Revisited

It may not be obvious that there is a lot of underlying mathematics involved in building the averaging filter. Yet it is precisely the mathematics that gives this averaging filter its performance and appeal. Solving the problem of the noisy thermostat illustrates its performance, but how about its appeal? The following indicates the appeal of using the averaging filter as well as other digital filters:

- *The performance of the filter doesn't vary over time or temperature.*
- *Parametric changes between units are predictable and bounded.*
- *Performance of a digital domain implementation is easily altered.*

This averaging filter exemplifies these "strengths" perfectly. Here's why: the point where the sinc function passes through the horizontal axis is set by the relationship f=1/(2T), where T is the width of the rectangular pulse either side of the vertical axis in the time domain. By increasing the "T" value, i.e., making the width of the pulse wider, the corresponding width of the sinc pulse becomes narrower. If you needed more *roll-off* of the noise than we obtained, you would widen the rectangular pulse. Since the width of the rectangular pulse is determined by the number of averaging points, we would have to increase the number of averaging points to narrow the *sinc pulse*. In any case, the characteristics of the filter are a function of the mathematics, the number of points, and the arithmetic used to combine the data and are unaffected by the ravages of time and temperature.

Averaging Filter Summary

We have just constructed an averaging filter by doing two major things:

Learning new concepts. We learned how to look at signals from a different perspective; the FT has given us a new way of looking at signals. The FT has also given us a new methods, like convolution, to give us a very effective means of manipulating signals.

Reinterpreting what we already knew. The same old process of averaging which many of us have done time and again was reviewed and translated to include the new concepts brought to us by the Fourier Transform.

Where Do We Go From Here?

Before we started to talk about the details of the FT, I said we could do useful things without getting bogged down in the mathematics of the FT. We built a digital filter and have "proven" the Nyquist criteria and aliasing. Both of these are substantial accomplishments. They indicate a true understanding of some of the more difficult signal processing concepts like convolution.

From here, you can apply the type of reasoning we learned from the averaging filter. This style of thought and problem solving will always have its place no matter how detailed your involvement in the mathematics of signal processing. I must give you a word of caution: although our accomplishments to this point are very real and very useful, you must progress to the next level of understanding to be truly creative, and wholly accurate, in applying signal processing techniques. In order to increase your understanding of signal processing, I recommend that you read **Chapters 4-7** of this book. These chapters review the concepts we've discussed and presents many others like Infinite Impulse Response (IIR) filters and the discrete Fourier Transformjust to name two in more detail.

Then read every signal processing text you can get your hands on. At first, the mathematics can be intimidating. If you try to relate the math to the graphical and intuitive treatment of the subject in this book, you won't get so bogged down or lost. While many of us may be involved in signal processing for recreation, I believe signal processing is a problem-solving tool which could enrich all our lives.

Talk to anyone and everyone about what you are learning. By hearing yourself talk about what you know and what you're learning, you will solidify your understanding.

CHAPTER 4 Fundamentals of DSP

The purpose of this chapter is to present some basic concepts needed for the use and apply **Digital Signal Processing** (DSP). The presentation of this chapter may be different than the reader is accustom to. I have attempted to present topics through the use of arguments and concepts. Mathematics is omitted where verbal and graphical presentations may better convey an idea. For more in-depth applications, you are directed to one of several texts on the subject.

Continuous Time Systems

Before leaping into the **digital domain** of DSP, take a minute to look at the world around us, a continuous real-time world, the analog domain. By continuous, we mean things move from one state to another by passing through all the states in between. At all times there can be a state associated with this process. The way the sun moves from sunrise to sunset is a good example of a continuous time process. The sun does not stay in the same position all morning and then jump instantaneously to its sunset position. Nor does it jump 1/12 of its journey at the top of each hour. It appears to move continuously across the sky. This is **analog**. *What we see, hear and touch are analog signals.* This being the case, many readers may wonder why anyone would be interested in digital (non-analog) signal processing. Digital signal processing's advantages are made clearer later, but first let's investigate continuous time analog signals and the concepts of the analog domain.

Sinusoids And Spectra

Much literature is dedicated to the study of sinusoids. Sinusoids go by many names: sine waves, pure tones, simple harmonic signals, etc. Taking our lead from this, perhaps we should look at the sinusoids and their **spectra**. Many of the everyday events involve sinusoids, vibrating guitar strings, rotating motors, even the apparent motion of the sun. The most convenient example might be the ordinary dial tone of the telephone. This signal is a sine wave which oscillates back and forth. The word "tone" is used because a person perceives the signal as a single pitch at 440 cycles each second (440 Hz).

Notice that the dial tone was described two different ways. The first, as an oscillating sine wave, describes what is happening with respect to time. This is known as the **time domain** description of the signal. The second description indicated something is going back and forth in a cyclic manner and as a tone, tells us how fast or how often it is moving. This is an example of the **frequency domain** description of the signal. *Both the time and frequency domain descriptions accurately described the phenomena.* ***To understand digital signal processing, we will need to accept, understand, and use the two different yet intimately related descriptions of signals in the time and frequency domains.***

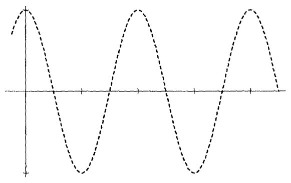


Figure 4-1 Domain Description of Cosine Function

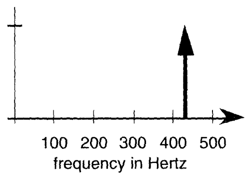


Figure 4-2 Frequency Domain Description of Cosine Signal

Let x(t) represent the dial tone as a function of time in seconds.

$$x(t) = A\cos(2\pi f t)$$

Alternately, in the frequency domain description (by convention we use capital letters):

$$X(f) = 0$$

Everywhere except where f = 440 Hz

The Cosine Function

The **cosine function** is a sine wave with a phase of 90 degrees. *The term "sinusoid" is often used to refer to a cosine function as well as a sine function.* This reference is generally accepted. Thus, for the remainder of the book, the term sinusoid may refer to either a sine or cosine. At times one may be preferable to the other; some mathematical relationships are more elegantly stated using one instead of the other.

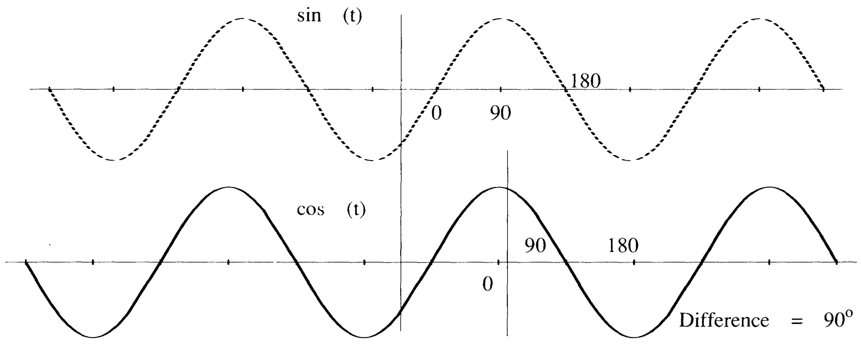


Figure 4-3

Cosine Function is equal to sine wave with a phase shift of 90 degrees

More mathematically stated:

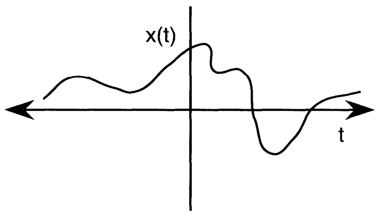
$$\sin(t) = \cos(t - \frac{\pi}{2})$$

$$\cos(t) = \sin(t + \frac{\pi}{2})$$

The Frequency Description

Few engineers would argue that we describe a dial tone by a time domain description or a frequency domain description, that is, a tone with a frequency of 440 Hertz and no other frequencies. Some readers may not be so sure we can give a frequency domain description for any arbitrary signal. For example, take the signal coming from the needle of your stereo when playing a favorite record. It has a time domain description that can be viewed on a scope. We may guess that it also has a frequency domain description since that's what the bass and treble adjusts on the stereo. What is really meant when we talk about the frequency domain description of this complicated signal? Let's use a hand waving argument for a moment. Let x(time) describe what happens to x over time, that is, how the recording varies with time. Now let X(frequency) describe how *often* these "things" happen to the recording x(time). ***Frequency is synonymous with "how often something happens."*** The question remains, "What should we use as a basis to measure how often these *things* occur?"

We are able to accept that a sinusoid consists of a single frequency. ***The simplicity of this description makes the sinusoid the ideal concept to use as a basis for frequency domain descriptions.*** This means that ***we may think of the complete frequency domain description as a collection of sinusoids***, just as we view the time domain description as a collection of points in time. It is also possible to view this the other way around: a collection of sinusoids (a frequency domain description) may be used to construct a collection of time points. ***The time or frequency domain descriptions are equally valid and there is nothing lost going from one to the other***. Based upon this reasoning, the highly complicated waveform of our favorite record does indeed have a valid and unique frequency domain description.



<=>

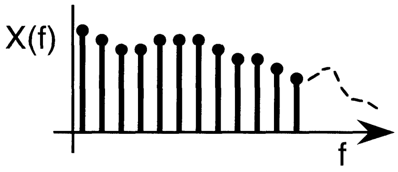


Figure 4-4 Time Domain Description of Signal x(t)

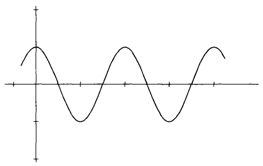
Figure 4-5 Frequency Domain Description of Signal X(f)

The Sinusoid And The Delta Function

Continuing with the frequency domain description of the sinusoid, it can be said that the ***frequency domain description*** (or **spectra)** *of a sinusoid is zero everywhere except at the frequency of that sinusoid.* At this point we have skirted the issue of what the <u>value</u> of the spectra is at the frequency of the sinusoid.

Let's not worry about the height of the spectrum at the sinusoid's frequency. Our concern is with the area under the curve at the sinusoid's frequency. Let's ***define the spectra of a sinusoid as having an amplitude of zero everywhere except at the sinusoid's frequency, where the area under the curve of the spectra has an area of one***. See Figures 4-6 and 4-7 for an illustration.

A sinusoid with an amplitude of two can be considered as having zero spectra everywhere except at the sinusoid's frequency where the area under the curve is equal to two. The mathematical notation for this type of function is known as the **Delta function.**



<=>

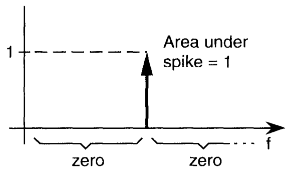
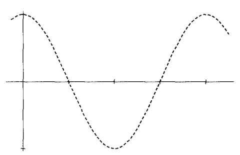


Figure 4-6 Sine Wave with

Figure 4-7 Spectra



<=>

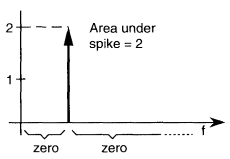


Figure 4-8

Sine Wave with Amplitude 2 and Frequency 1/2 of Fig. 4-6

Figure 4-9 Spectra of Sine Wave from Figure 4-8

$$\delta(f) \equiv \begin{cases} 0 & \text{for } f \neq 0 \\ \infty & \text{for } f = 0 \end{cases} \quad \textit{implies} \quad \delta(f-f_c) = \begin{cases} 0 & \text{for} f \neq f_c \\ \infty & \text{for} f = f_c \end{cases}$$

Additionally, the area under the spike is given by:

$$\textbf{Area} = \int_{-\infty}^{+\infty} \delta(f)df = 1$$

Also,

$$\int_{-\infty}^{+\infty} \delta(f)x(f)df = x(0)$$

$$\int_{-\infty}^{+\infty} \delta(f-f_c)x(f)df = x(f_c)$$

The Delta Function

The Delta function equals zero everywhere except when its argument is zero. *At the zero point, the value of the function is not specified, but rather, the area underneath the curve is specified. This area is specified with a value of one.* One way to understand this is by looking at various rectangles. The area of a rectangle is equal to the base times the height. Consider a sequence of rectangles all with an area of one. As shown in Figure 4-10, start with a square (base = 1, height = 1, thus area = 1 x 1). Notice that for each successive rectangle we have shortened the base and are forced to increase the height in order to maintain a total area of one. It is in our interest to continue this process until our rectangle is so narrow that its base is zero. How tall would the last rectangle have to be to maintain its area of one? In a rough sense, it would have to have infinite height to make up for its zero width.

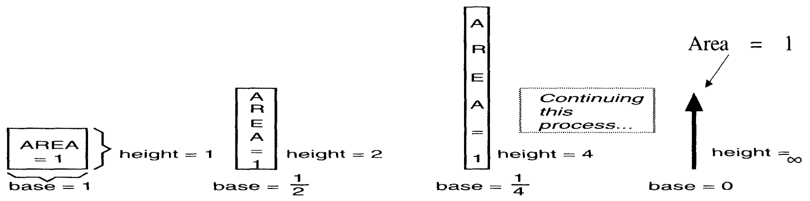


Figure 4-10

Delta Function has zero width, infinite height, and area one

If we place the last rectangle on an axis, we can consider this as a function that was zero everywhere except at a certain point where the area under the *curve* is one. This is precisely the Delta function.

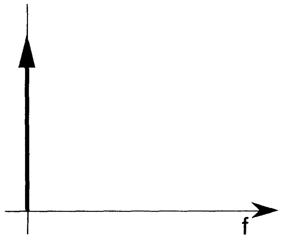


Figure 4-11 Example of the Delta Function

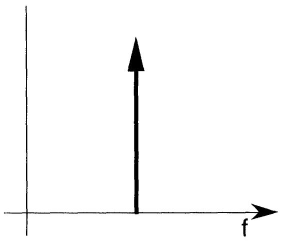


Figure 4-12 Example of the Delta Function

It is also worth noting that typically the Delta function is plotted with an arrow at the top of the spike. This arrow helps remind us the height is infinite.

Earlier we discussed that the sinusoid with amplitude one has an area of one under the spectral spike, and the sinusoid with amplitude two has an area of two. Since we know the arrow reminds us the height is infinite, we are free to draw the length of the line to be proportional to the area under the spike. This is exactly the convention used.

You may now, for the time being, forget about the Delta function and breathe a sigh of relief. It was included so that some loose ends can be avoided when the Fourier transform is presented.

Amplitude And Phase Descriptions

The process of specifying a signal is a bit more complicated than saying there is a tone at a particular frequency. To be thorough, we need to discuss two other parameters: **amplitude** and **phase**. ***Amplitude is the maximum level a sinusoid reaches anywhere in its cycle***. Think of the sinusoid's height in Figure 4-13. For physical phenomena, the amplitude can be related to qualities such as loudness, brightness, and voltage. The higher the amplitude, as seen in the time domain, the higher the amplitude is in the frequency domain represented as a "spike".

Let's look at phase, a simple idea so often made difficult. Take a look at the signals shown in Figures 4-13. These are all sinusoids of the same frequency and amplitude, yet they are obviously different. Not only is the time domain description we were using up to now inadequate to resolve these differences, Figure 4-14 shows the frequency domain description is also lacking. *The last descriptor we need to help us out of this quandary is the descriptor of phase.*

Looking once more at Figure 4-13, we see the only difference between the sinusoids is each starts at a different point in its cycle. Wave one starts at zero degrees (at t = 0), wave two starts at 90 degrees, and wave three starts at 180 degrees. ***Calling this starting point "the phase" of the sinusoid completes the description nicely.*** We will also need to provide for phase in the frequency description as shown in Figures 4-15 through 4-24.

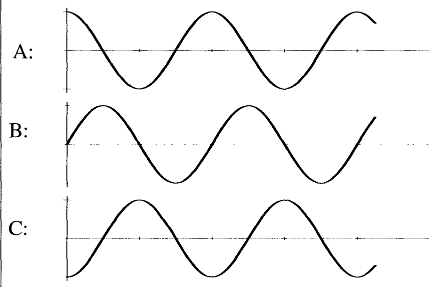


Figure 4-13

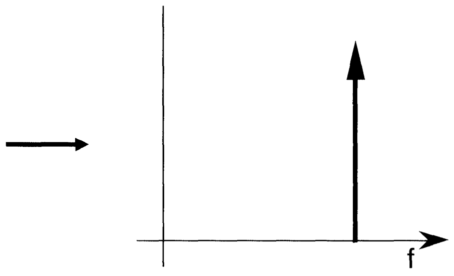


Figure 4-14

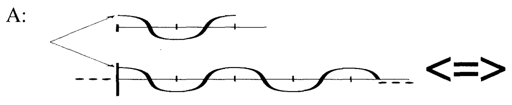


Figure 4-15

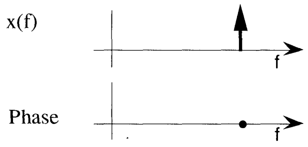


Figure 4-16

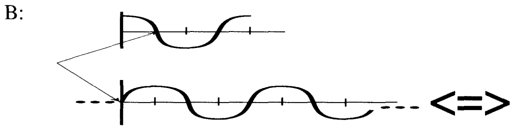


Figure 4-17

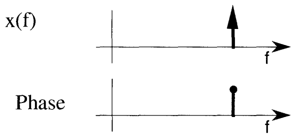


Figure 4-18

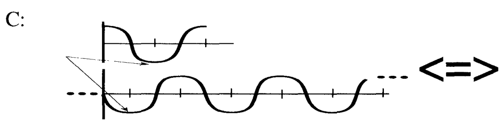


Figure 4-19 Time Domain

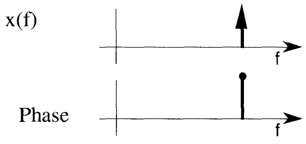


Figure 4-20 Frequency Domain

Importance Of Phase

If the previous description of phase is any indication, the concept of phase doesn't seem too difficult, then why is it so misunderstood? The difficulty doesn't begin with the phase, but with the question of when does t = 0. Viewing Figure 4-13, it is easy to know when t = 0. Simply look at the plot, point to where t = 0, and you have it. Look at the dial tone example in Figures 4-21 through 4-23. Let's define t = 0 as the exact moment you pick up the receiver. When the receiver is picked up you can "look" at the waveform, determine its phase as shown in the figures, and complete the amplitude, phase and frequency descriptions. Sometimes it can be picked up exactly when the waveform equals zero and increasing (0 degrees phase, Figure 4-21), and other times when it equals one and decreasing (90 degrees phase). Simple enough? Suppose someone left the receiver off the hook before you arrived. You are asked to give the waveform description as before. How do you determine the phase? Well, you can't. ***Sometimes phase doesn't matter, only the amplitude and frequency are important.***

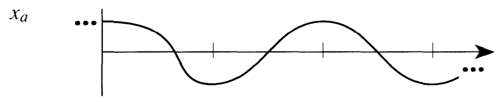


Figure 4-21

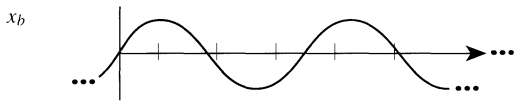


Figure 4-22

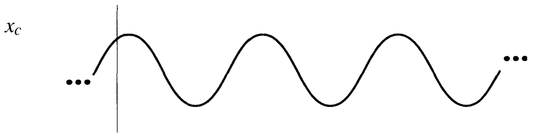


Figure 4-23

Expression for figure 4-21:

$$x_a(t) = A_a\cos(2\pi f_a t + 0)$$

Similarly for figure 4-22:

$$x_b(t) = A_b\cos(2\pi f_b t - \frac{\pi}{2})$$

For figure 4-23 we can write

$$x_c(t) = A_c\cos(2\pi f_c t + \phi)$$

Relative Phase

As we just discussed, sometimes it is not possible to determine the absolute phase of a signal due to the ambiguity of when t = 0. Therefore, we must accept the fact that we cannot determine phase of a signal and go about our business. However, there's more to consider than just the absolute phase. We must also consider **relative phase**. The idea comes into play when there is more than one signal. Let's use three for illustrative purposes. *We may not know the absolute phase of any of the three, but we can easily see a phase difference between them.* ***This phase difference is known as relative phase.***

The three signals of Figures 4-24 through 4-26 have the same frequency but different phases. We know this because we can see their peaks and valleys do not coincide. We do not know the absolute phase of A, B, or C, but we can determine their relative phase. Let's arbitrarily pick signal A as our reference signal. We can find a point on A where the wave equals zero and is headed positive. Use this point as a time reference and determine the other two signals' phases in relation to this point. *Remember that since we chose the reference point, it is the relative phase we are determining, not the absolute phase.*

Now a quick look at the importance of relative phase. In Figures 4-27 and 4-28 we added signals A and B (Figures 4-24 through 4-28 to get signal X (Figure 4-28). Notice how signal X has the same frequency, but the amplitude has become larger. We also added signals B and C to get signal Y (Figure 4-28). This equals zero all the time. We have two very different results from adding signals of like frequency but of different relative phases. Therefore, relative phase is important.

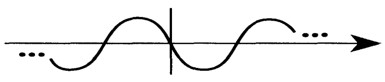


Figure 4-24

Choose a reference point $t = t_{ref}$ (shown by vertical line). From the t_{ref} we can find phase:

$$\phi_a = \frac{\pi}{2}$$

$$\phi_b = 0$$

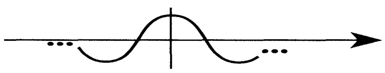


Figure 4-25

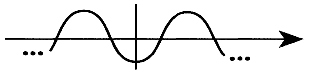


Figure 4-26

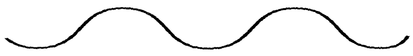


Figure 4-27

The sum of two sinusoids of the same frequency is a sinusoid of the same frequency.

Notice how drastic the effect due to different relative phases can be.

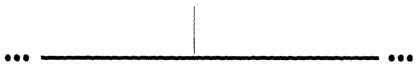


Figure 4-28

$$A_a \cos(2\pi f_c t + \phi_a) + A_b \cos(2\pi f_c t + \phi_b) = R\cos(2\pi_c t + \phi_r)$$

where,

$$R=\sqrt{A_a^2 + A_a^2 + 2A_aA_b\cos(\phi_a - \phi_b)}$$

$$\phi=Tan^{-1}[A_a\sin(\phi_a) + A_b\sin(\phi_b)]$$

Time Delay And Phase Shift

By now, you've had it with sinusoids, phase, etc. What has this all have to do with DSP? It is time for a technical teaser, a simple concept that is at the very root of filtering (including **digital filtering**). It also starts us thinking about the interplay between time and frequency. Consider Figures 4-29. The sinusoid shows a phase equal to zero (solid line). Also found is a delayed version of the same (dotted line). The time delay between the two is "t" seconds. We can visually estimate the relative phase between the two to be approximately 3/4 π (about 120 degrees). *Let's add the delayed version to its original and look at the output. Notice the amplitude has been effected.* The original and delayed amplitude both had an amplitude of one, but their sum has a smaller amplitude.

Continuing the teaser, observe the same experiment in figures 4-31 and 4-32. The only thing different is the frequency of the sinusoid. *The time delay is the same but the relative phase is different!* We will estimate the relative phase to be $\frac{\pi}{2}$ here. Again, summing the two, we find the amplitude of the result is different from the inputs (Figure 4-32). But more importantly, we see the resulting amplitude is different from that of figures 4-30.

The frequency dependency of the relationship between the inputs and outputs of our simple setup shows the characteristics of a **filter**. For our constant time delay of "t," there are input frequencies for which constructive interference occurs which results in larger amplitudes, and there are other input frequencies for which destructive interference occurs which results in smaller amplitudes. ***This filtering effect is made possible by the fact that for a constant time delay, the relative phase is dependent upon the input frequency***. The frequency dependency of the relative phases can be used to affect different frequencies in different manners.

What about DSP? *The experiment shows the makings of a filter.* The first operation was a delay, the second the summation. Although DSP hasn't actually been discussed yet, those of you with an inkling may see the two operations (delay and addition) are the types a computer (or DSP) would be very good at. To delay, one could store something in memory to be used again later. To add, well, it should be obvious that computers can be somewhat useful with addition and storage.

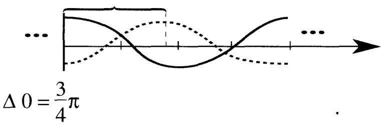


Figure 4-29

Figure 4-30

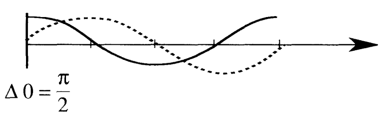


Figure 4-31

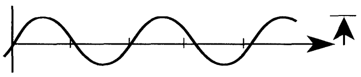
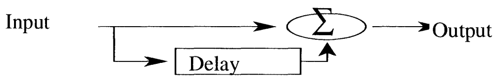


Figure 4-32



$\cos(2\pi f_1 t_1 + \phi_1)$ *can be written as* $\cos(\theta_1)$ *where* $\theta = 2\pi f_1 t_1 + \phi_1$

$\cos(2\pi f_1 t_2 + \phi_1)$ *can be written as* $\cos(\theta_2)$ *where* $\theta_2 = 2\pi f_1 t_2 + \phi_1$

Frequency Descriptions

We have been adding signals of the same frequency to each other without a problem. Let's investigate two signals of different frequencies. Figures 4-33, 4-35, and 4-37 show signal A, a low frequency sinusoid, added to B, a higher frequency sinusoid, to produce C which exhibits sinusoids of both frequencies. The time domain plot of C visibly shows the two frequencies. The large amplitude-slow fluctuations are due to A, while the small and fast fluctuations are due to signal B. The frequency domain representations may also describe the phenomena here Figures 4-34 through 4-36).

$x_a(t)$

A:

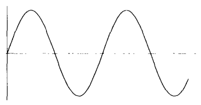


Figure 4-33

<=>

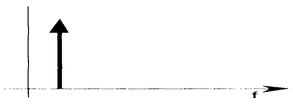


Figure 4-34

$x_b(t)$

B:

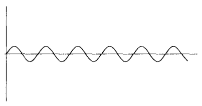


Figure 4-35

<=>

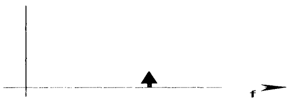


Figure 4-36

$x_c = x_a + x_b$

C:

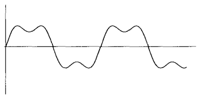


Figure 4-37

<=>

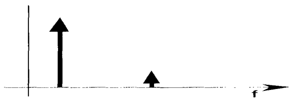
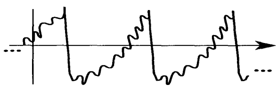


Figure 4-38

D:



<=>

Figure 4-39

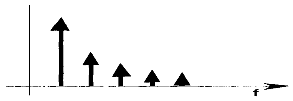


Figure 4-40

In general, additive signals demonstrate additive spectra.

$$x_c(t) = x_a(t) + x_b(t) \iff X_c(f) = X_a(f) + X_b(f)$$

For the plots above we have:

$$x_c(t) = A_a\cos(2\pi f_a t) + A_b\cos(2\pi f_b t) \iff X_c(f) = \delta(f - f_a) + \delta(f - f_b)$$

Notice how the frequency domain plots are very simple and quickly show signal spectral components. To further the point, consider signal D (Figure 4-33). Can we determine what's really happening here? Not easily. However, the frequency domain representation of D clearly shows its components (Figure 4-40)

Sinusoidal Basis of Signals

Until now, we have been discussing the sinusoid. The greatest advantage of this signal is it is so easy to determine the frequency domain description: a single spectral line with its height proportional to its amplitude. But, hasn't this been a waste of time since "real world" signals are so much more complicated? Not really.

A classic example of this is the **square wave**. Figure 4-41 shows how a square wave can be built by adding the sinusoids of the proper frequency, phase, and amplitude. Moreover, from this construction we are given a sense of what spectral content of the square wave is. Notice how we get closer to the desired square wave with each additional step. Using all of the *odd harmonics* of the original sinusoid, we eventually get a square wave. This means that a perfect square wave needs infinite terms, thus, has infinite spectral content. But as you can see, the amplitude needed for each addition gets smaller.

So, for many purposes, a square wave which is "good enough" can be achieved with a limited number of spectral components. The need for an infinite number of sinusoids where only a limited number may be physically tolerated is called the **Gibb's Phenomena**. This is why many square waves we see on oscilloscopes actually look more like C, D or F than G. ***Many non-sinusoidal signals can be treated as the summation of sinusoids***. This is how we are able to think of frequency components of non-sinusoidal signals.

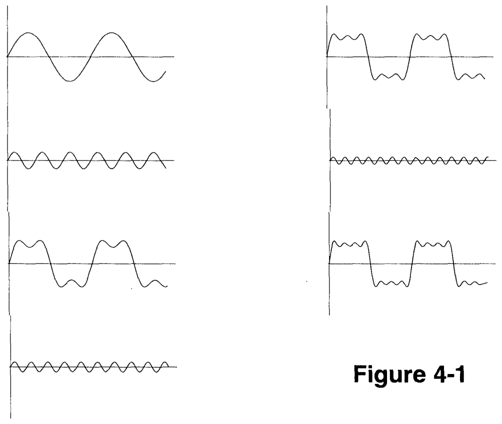


Figure 4-1

A square wave can be built by adding the sinusoids of the proper frequency, phase, and amplitude.

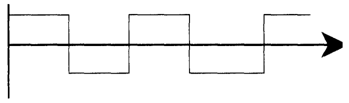


Figure 4-42

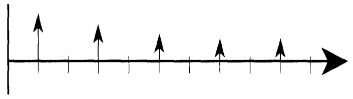


Figure 4-43

$$x_{square}(t) = \sum_{n=1}^{\infty} \frac{1}{n} \cos(2\pi f_n t - \frac{\pi}{2}) \Leftrightarrow X(f) = \sum_{n=1}^{\infty} \frac{1}{n} \delta(f - n)$$

Real And Complex Modulation

We added various sinusoids and saw what can happen in both the time and frequency descriptions. What happens when we *multiply* various sinusoids? Before we answer this, let's walk through a thought experiment. If we took a constant and multiplied it by a time varying signal, our result will be another time varying signal. We can say that by multiplying the constant by this time varying signal we affected the frequency of the original constant. Obviously, if this is true, our result is no longer constant. Now, what happens if the original signal wasn't constant, but a time varying signal as well? It seems completely likely that we will still **affect its frequency by multiplying it with another time varying signal. This idea is known as** *modulation.*

Our thought experiment is graphically shown in Figure 4-44 with a teaser of things to come. The left columns are time plots, the right columns are the frequency domain plots. We can think of constant's original spectra (located at zero) as jumping up to a new location determined by the multiplying signal. The real question is what happens in the second case when there are two time-varying signals?

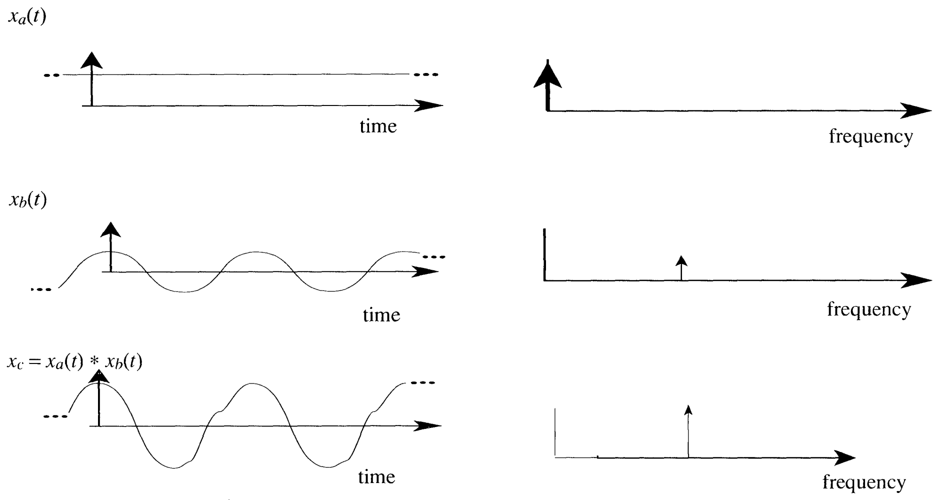


Figure 4-44

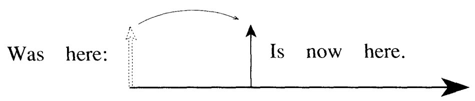


Figure 4-45

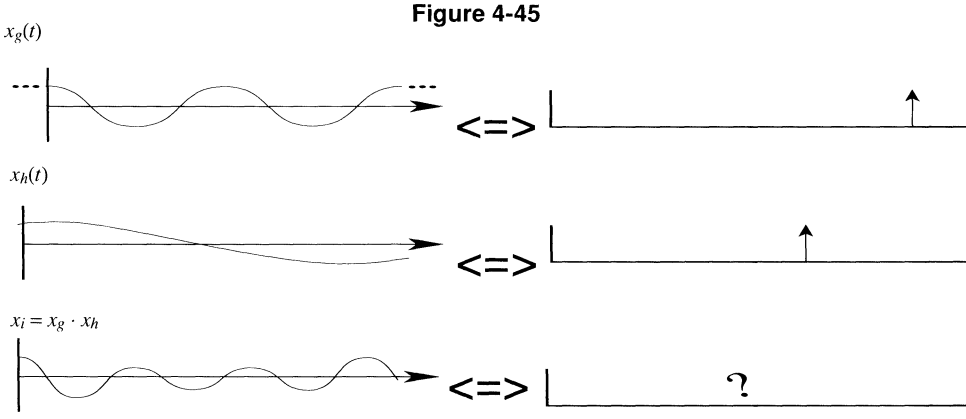


Figure 4-46

Real Modulation

The mathematical basis for exactly how the frequency is affected is given by the trigonometric identity found in at the bottom of the page. However, even without using the identity, we can visualize some of its qualities. We start with two sinusoids of frequency f and 3f Figures 4-47 through 4-52. These two are multiplied together. Notice the resulting wave looks a bit different, it doesn't really match either of the original two waves.

Take a closer look. Its not too hard to imagine that within the resulting wave we see something happening twice as often as f and maybe something else happening at four times as often as f. Just for grins, let's plot waves with frequencies of 2f and 4f. As a check, we *added* the waves of f = 2 and f = 4 and verified their sum to be the same as when we *multiplied* waves with f = 1 and f = 3.

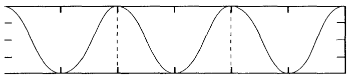


Figure 4-47

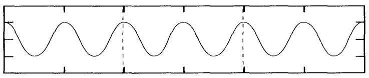


Figure 4-48

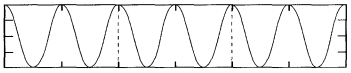


Figure 4-49

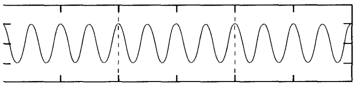


Figure 4-50

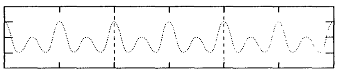


Figure 4-51

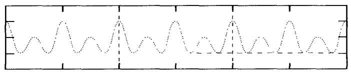


Figure 4-52

$$\cos(\alpha)\cos(\beta) = \frac{1}{2}[\cos(\alpha - \beta) + \cos(\alpha + \beta)]$$

or

$$\cos(2\pi f_1 t + \phi_1)\cos(2\pi f_2 t + \phi_2) = \frac{1}{2}\cos[2\pi(f_1 - f_2)t + (\phi_1 - \phi_2)] + \frac{1}{2}\cos[2\pi(f_1 + f_2)t + (\phi_1 + \phi_2)]$$

Frequency Shifting Properties

Earlier we graphically demonstrated how modulation can be interpreted in the time domain. In most cases, the goal of modulation is to make use of its frequency shifting properties. To illustrate this, the corresponding frequency domain plots are shown in figure 4-53 - 4-59. To keep things simple, only the cases where alpha - beta and alpha + beta results in positive numbers as shown. We will discuss negative frequency cases later.

Frequency shifting is most relevant to modulation. It is this modulation effect that allows audio signals (voice, music, etc.) with frequencies between 20 to 20,000 Hertz to be shifted up to the hundreds of thousands of Hertz or even megahertz where these signals may be transmitted in the form of radio waves.

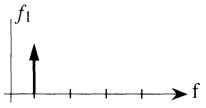


Figure 4-53

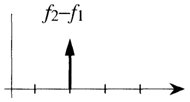


Figure 4-54

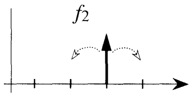


Figure 4-55

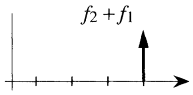


Figure 4-56

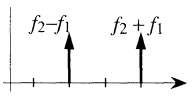


Figure 4-57

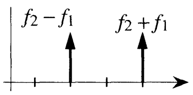


Figure 4-58

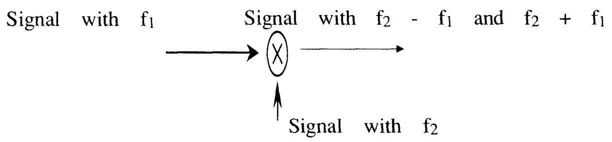


Figure 4-59

Imaginary Numbers

The philosophy adopted for this book has been to provide conceptual examples and heuristic arguments first to create an understanding. Mathematics should follow where they cement these concepts into a firm foundation. However, this is not so easy with the introduction of imaginary or complex numbers. Historically, there has been some debate of the philosophical aspects of the use of imaginary numbers. The reader is asked to take a slight leap of faith concerning the use of imaginary numbers. Please believe that their use can be valid and they are not uncomfortable to work with. More importantly, they allow us to perform analysis in an easier manner.

The basis of imaginary numbers is the unit i (or sometimes j - used by electrical engineers who wish to avoid confusion with the symbol (i) which is used to represent current). "i" represents the square root of -1.

If we define $\sqrt{-1}$ to be equal to a number we will call "i," then "i" exists by definition. First we will view some of the basic properties of "i." This begans to make more sense after it's been explained with some of the examples on the next few pages.

$$\text{If } i = \sqrt{-1}\text{, then } i^2 = -1 \text{ and } \frac{1}{i} = -i$$

Some imaginary numbers: 5i -4i 2.21i sqrt(2)i

Q: What is 2i + 3i ?

A: 5i.

Q: What is 5i · i ?

A: $5 \cdot i^2 = 5 \cdot (-1) = -5$.

Let's look at the next step - **complex numbers**. Complex numbers are numbers which may have real and imaginary parts. Generally, we write complex numbers as the sum of these two parts.

Some complex numbers: 2 + i 4 + 3i -2 - 2i

Q: What is (2+i) + (4+3i) ?

A: Add the real parts together and add the imaginary parts together to get 6+4i.

Q: What is (2+i) · (4+3i) ?

A: $2 \cdot 4 + 2 \cdot 3i + i \cdot 4 + i \cdot 3i$

$$= 8 + 6i + 4i + 3i^2$$

$$= 8 + 10i + (-3)$$

$$= 5 + 10i$$

The letter z is often used to denote a complex variable. Further convention uses x to represent the real component and y to represent the imaginary component (foreshadowing?):

$$z = x + iy$$

$$\text{Real}(z) = x$$

$$\text{Imag}(z) = y$$

The Z-Plane (Cartesian Form)

Okay, the foreshadowing relates to how conveniently the components of complex numbers can be represented on a plane. This Cartesian plane is known as the Z-plane (or the Complex plane). Here a complex number's real component is the x-coordinate, and the imaginary component is the y-coordinate. Therefore each complex number can be represented as a point on this plane.

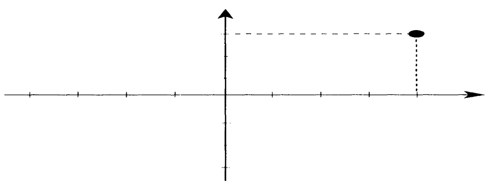


Figure 4-60

The Z-Plane (Polar Form)

Every point on the Z-plane can be expressed in Cartesian (XY) coordinates. Alternatively polar coordinates could be used. Here, the point is defined by its distance from the origin (magnitude or "r") and the angle from the positive real axis (angle or ϕ).

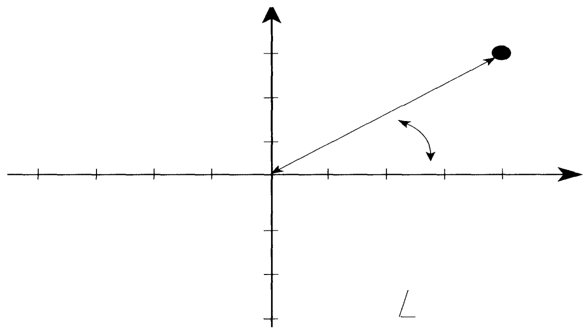


Figure 4-61

Cartesian And Polar Relationships

Points on the Z-plane can be represented in either coordinate system. Both are used. The relationship between the can be expressed by Figure 4-.0 The superscribed asterisk denotes complex conjugate.

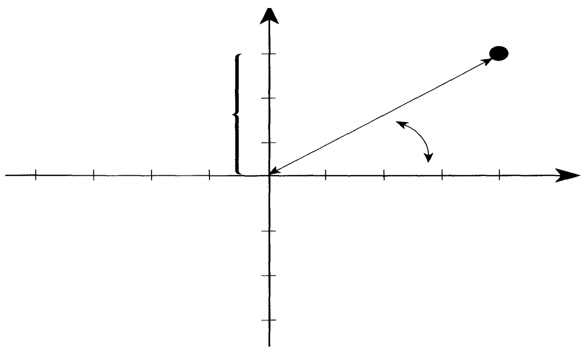


Figure 4-62

Euler's Formula

We're going to need to look at one more mathematical tidbit before we can complete this section. It's a version of euler's formula, named after Leonhard Euler 18th century mathematician and physicist. This tidbit is quite useful. But, more than that it is really elegant. So elegant that we'll provide a summary of its derivation. Consider the exponential function and its series expansion,

$$e^a = 1 + a + \frac{a^2}{2!} + \frac{a^3}{3!} + \frac{a^4}{4!} + \frac{a^5}{5!} + \dots$$

Also remember:

$$\mathbf{i^0 = 1,\quad i^1 = i,\quad i^2 = -1,\quad i^3 = -i,\quad i^4 = 1,\quad i^5 = i\quad \dots}$$

Letting "a" be replaced with "ia" we have:

$$e^{ia} = 1 + ia + \frac{i^2a^2}{2!} + \frac{i^3a^3}{3!} + \frac{i^4a^4}{4!} + \frac{i^5a^5}{5!} + \dots$$

$$= 1 + ia - \frac{a^2}{2!} - \frac{ia^3}{3!} + \frac{a^4}{4!} + \frac{ia^5}{5!} + \dots$$

Grouping real and imaginary components together:

$$e^{ia} = \left(1 - \frac{a^2}{2!} + \frac{a^4}{4!} - \dots\right) + i\left(a - \frac{a^3}{3!} + \frac{a^5}{5!} - \dots\right)$$

The real grouping is the series expansion for the cosine function and the imaginary grouping is the expansion for the sine function! So:

$$e^{ia} = \cos a + i \sin a \quad (\textit{Euler's Formula})$$

Now does it seem like it might be useful to us? Sure. Previous page tells us we can describe the complex number z by cartesian (XY) or polar (magnitude angle) systems:

$$z = x + jy$$

but,

$$x = r\cos(\phi)$$
$$y = r\sin(\phi)$$

so,

$$z = r\,[\underbrace{\cos(\phi) + i\sin(\phi)}_{e^{j\phi}}]$$

Since,

$$e^{i(\phi)} = \cos\phi + i\sin\phi$$

Then,

$$z = r\,e^{i(\phi)}$$

Remember Euler now?

With Euler's formula we can see that for any z we have a concise representation of z:

$$z = r\,e^{i(\phi)}$$

Not only do we have this concise representation, but since we've shown that this is mathematically valid as well, we know that we can perform mathematical operations on this representation and expect them to be true.

Phasors And Unit Circle

Let's plot a group of z's on the complex plane. For the plot we choose z's with a constant unit magnitude (r=1) and allow for different (and increasing ϕ's.)

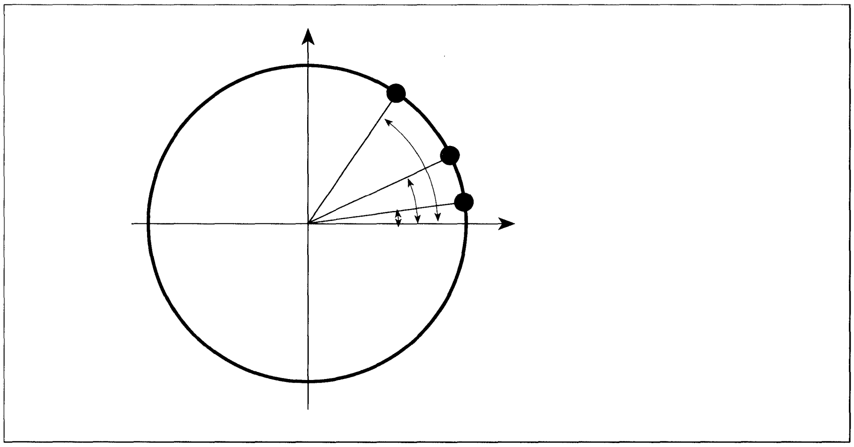


Figure 4-63

With ϕ increasing with time, we have the concept of a phasor. This phasor can be thought of as rotating around and around at some frequency. The frequency of rotation depends on how quickly φ increases. For our chosen z's with r=1, this phasor will trace out a set of points known as the unit circle. The points 1, -1, i, -i all lie on the unit circle.

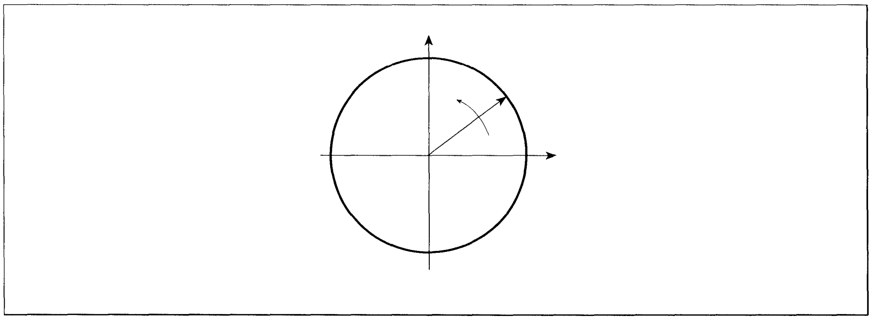


Figure 4-64

Some points to consider about phasor's. They are described by quantities such as magnitude (r) and frequency (rate of rotation). This sounds a bit familiar to the qualities of sinusoids, true? Yeah, maybe.

Another thought, if a phasor has a frequency of "f" turning counterclockwise, then a phasor with "-f" would turn the other way wouldn't it? Yes.

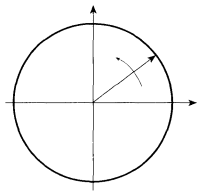


Figure 4-65

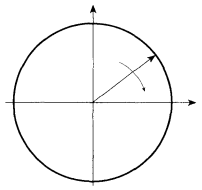


Figure 4-66

What About Phasors?

Let's draw a picture using two counter-rotating (one with +f, one with -f) phasors A and B.

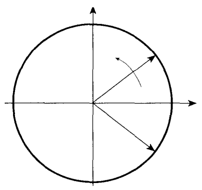


Figure 4-66

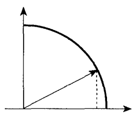


Figure 4-67

Also, review the relationships

Now, let's add the phasors A and B. Graphically we can see that this results in cancellation of the imaginary part (one is positive, the other negative), and the real part is doubled. From above we also know the real part (x) is r cos phi.

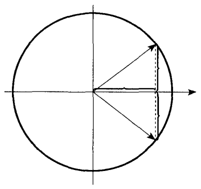


Figure 4-68

In a similar manner, B could be subtracted from A. Now the real parts cancel, leaving two time the imaginary components. But, the imaginary component (y) is equal to sin(ϕ).

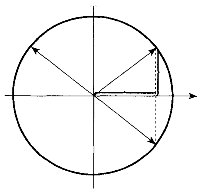


Figure 4-69

Negative Frequencies In Sinusoids?

Rewriting and restating we have:

$$\cos(2\pi ft) = \left(\frac{e^{j2\pi f} + e^{-j2\pi f}}{2}\right)$$

$$\sin(2\pi ft) = j\left(\frac{e^{j2\pi f} - e^{-j2\pi f}}{2}\right)$$

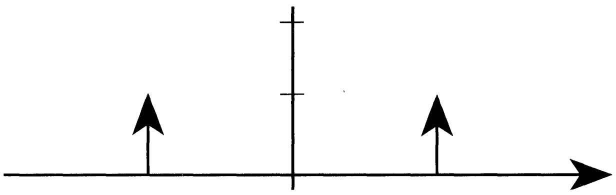


Figure 4-70

Remembering our sinusoid we see that this could now be expressed as consisting of two phasors. One phasor has positive frequency, one has negative frequency. Well, in our earlier plots for the frequency domain we haven't shown that. Let's correct for that omission now and redraw the frequency domain plot for a sinusoid with frequency of "f."

What about the phase for this negative frequency? Let's take a look at the two phasors at t=0. (Which seems fair enough since we look at the points around t=0 to determine the phase of a sinusoid in a time plot.)

$$\cos(2\pi ft) = \left(\frac{e^{j2\pi f} + e^{-j2\pi f}}{2}\right)$$

but t=0,

$$\cos(0) = \left(\frac{e^{i0} + e^{-i0}}{2}\right)$$

So one phasor is at e^{i0} (but $e^{i0}=1$) and the other is at e^{-i0} (but $e^{-i0}=1$), so plotting these phasors we have the following:

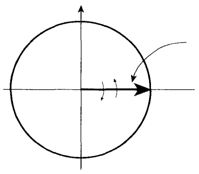


Figure 4-71

From our definition of angles we can agree the angles are 0 for both.

Thus, representing both the phase and magnitude of the particular sinusoid, we have the given phase for the positive frequency and the opposite phase for the negative frequency.

This page intentionally left blank.

Complex Reprsentation

Let's put the complex representation of the sinusoid to use in a quick review of the modulation theorem and verify that the new representation works.

When a sine wave is multiplied with another sine wave, we still get waveforms of the sum and differences of the two original frequencies.

$$\cos(x)\cos(y) = \frac{1}{2}[\cos(x-y) + \cos(x+y)]$$

This product can be written as:

$$\left(\frac{e^{j2\pi x} + e^{-j2\pi x}}{2}\right)\left(\frac{e^{j2\pi y} + e^{-j2\pi y}}{2}\right)$$

multiplying term by term,

$$\left(\frac{e^{jx+jy} + e^{jx-jy} + e^{-jx+jy} + e^{-jx-jy}}{4}\right) =$$

$$\left(\frac{e^{j(x+y)} + e^{j(x-y)} + e^{-j(x-y)} + e^{-j(x+y)}}{4}\right)$$

rearranging,

$$\frac{1}{2}\left[\left(\frac{e^{j(x+y)} + e^{-j(x-y)}}{2}\right)\left(\frac{e^{j(x-y)} + e^{-j(x-y)}}{2}\right)\right] =$$

$$\frac{1}{2}[\cos(x+y) + \cos(x-y)]$$

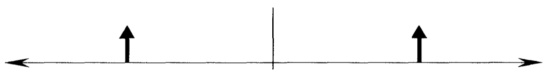


Figure 4-72 Spectrum of cosx

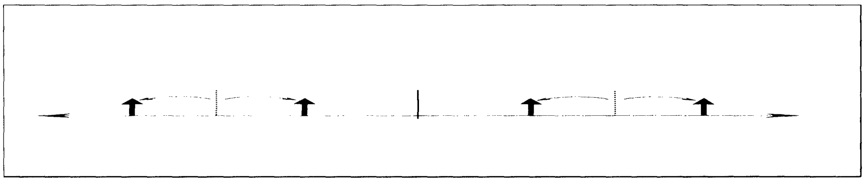


Figure 4-73 Spectrum of cosx cosy

So why bother to use this complex representation if you get the same thing? We have used complex number notations to represent strictly real sinusoids. Remember that for the **complex phaser** of the sinusoid, the imaginary parts exactly cancel and what is left is the real part. Since only the representation changed, one would not expect anything different to happen. However, if we want to, we can do something different.

With the use of complex representation, we now have the freedom to multiply our real sinusoid by a complex waveform; one with both real and imaginary components where the imaginary components don't have to cancel. Notice that both of the resulting phasers had their frequencies reduced by a factor of y. We no longer get the sum and the difference, only the sum; for example, (x + -y) and (-x + -y). The result shows that we have one phaser rotating in the positive direction at rate (x+y), and one phaser rotating in the negative direction (x-y). What have we produced? The multiplication by the single phaser gave us a means to perform a frequency shift in only one direction. We can choose this shift to be up or down by the sign of the exponential.

Looking at the spectral plot in figure 4-74, we see that it is *not* symmetric about f = 0. This means that for each spike on the left there is *not* a corresponding spike on the right. The lack of pairing means that the phaser does not have a counter-rotating pair which cancels the imaginary part. The resulting waveform is complex.

What happens if we take the real part of the resulting waveform? Let's toss out the 1/2 for clarity.

$$\frac{1}{2}Real\,[e^{jx-jy}+e^{-jx-jy}] = \frac{1}{2}Real\,[e^{j(x-y)}+e^{-j(x+y)}]$$

p23b

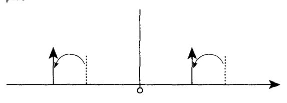


Figure 4-74

$$\frac{1}{2}Real\,[e^{j(x-y)} + e^{-j(x+y)})] = \{\{\frac{1}{2}$$

$$Real\,[cos(x-y) + j\,sin(x-y) + above\_\ cos(x+Y) - j\,cos(x+y)\,]\}$$

We have the sum and difference again, just as we would if we had done a strictly real modulation.

Integral Of Sine Wave

The last bit of information you need to truly appreciate and comprehend the Fourier Transform is below. This transform is the basis of **spectral analysis**. Bear in mind that the figure on this page is *a stepping stone to greater things and is not introducing a concept that stands alone*. By the way, for those who get queasy with the mention of integral calculus, breath easy ... this material is straight forward.

We think of the integral of a waveform as, on a XY plot, the integral of the curve Y as it is plotted along the X axis is the area under the curve. An addendum is that the area above the x axis is + and the area below is -.

Let's use a sinusoid for the curve. What's the integral, the area under the curve, for one complete cycle of a sinusoid? In

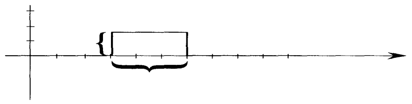


Figure 4-75

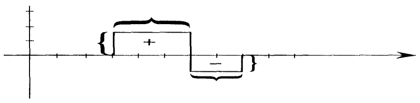


Figure 4-76

figure 4-77 we can see that there is the same amount of positive area above the x axis as there is negative area below the axis. These two are equal; however, opposite sign areas cancel each other. The integral of the sinusoid's cycle is zero. We can extend this reasoning for a sinusoid that continues to infinity. Still there are equal positive and negative areas that cancel each other; thus, the integral of the infinite sine wave is zero. Notice, that this result is true regardless of the frequency, amplitude or phase of the sine wave. ***All infinite duration sinusoids have an integral of zero.***

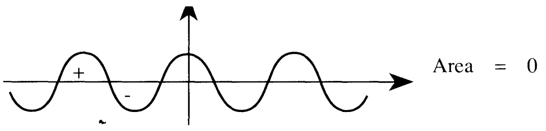


Figure 4-77

All Sinusoids Have Zero Valued Integers?

One clarification needs to be made for a special case. What if the sinusoid's frequency is zero? ***A zero frequency means that it is unchanging or a constant. Electrical Engineers recognize this as another way of saying DC.*** Figure 4-78 shows us that a constant integrated over infinity is not zero, it is infinite. So a sinusoid with zero frequency, a constant or DC if you prefer, is different.

Fourier Transforms

We have all the arguments in place to understand a most magical concept called the **Fourier Transform**. We do hope, however, that it will not seem too mysterious after the explanation.

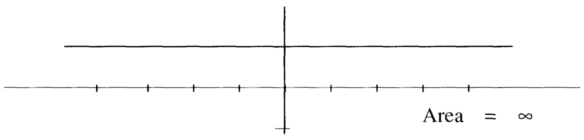


Figure 4-78

The Fourier Transform is what enables us to look at signals in either the time or frequency domain. It allows us to switch back and forth or perform, as it is usually said, a transformation between the two domains. Before actually defining the Fourier Transform and proceeding with an example, let's introduce some of the language and symbols associated with the transform.

If we describe a signal by its time domain description, we use a small letter followed by *(t)*; for example, x(t). Alternatively, when a signal is described by its frequency domain representation, a capital letter is used followed by *(f)*; for example, X(f). The same letter is used in both the time and frequency descriptions if both descriptions refer to the same signal. For instance, x(t) ≡ time domain description of signal x and X(f) = frequency domain description of the *same* signal x.

The variables x(t) and X(f) are intimately related since they both describe the same signal. The relationship between the two is given a name, **Fourier Transform pairs**, or just *transform pairs.* In fact, the relationship is also given a symbol. The symbol means that the variables on each side are indeed transform pairs. x(t) <=> X(f) means that x(t) is a transform pair with X(f). This is written as:

X(f) = F [x(t)] where F means Fourier Transform

Taking the Fourier Transform of something really means that you give me the time domain description of x, and I'll give you the Frequency Domain description. More explicitly, we can say X(f) is described by taking the Fourier Transform of x(t). See figure 4-79 for an illustration.

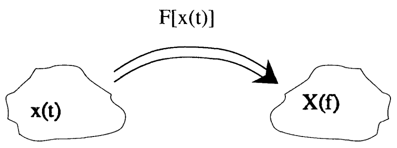


Figure 4-79

Alternatively, it can be described as you give me the frequency domain description and I'll give you the time domain description. In other words, going the other direction. This is called the **inverse Fourier Transform**. This is written as:

$x(t) = F^{-1}$ [X(f) } where F^{-1} means Inverse Fourier Transform

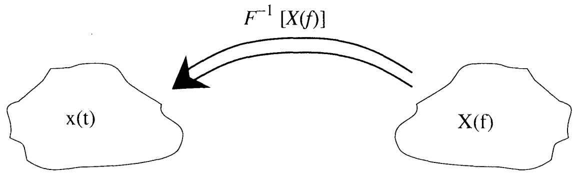


Figure 4-80

Why the Fourier Transform?

Good question. Here are a few reasons why.

Reason 1: In signal processing, a common problem involves the transmission of signals or their reception. This transmission and reception takes place through a **channel**. For radio, the channel is the sky; for a musical concert, the channel is the room in which the concert takes place; for underwater sonar, it is the water. The point is, each one of these channels has its own characteristics. Some may really corrupt the signal of interest.The channel characteristics may have echoes (and other frequency-dependent effects). Time delay, which we have seen, inherently effects frequencies. With the Fourier Transform we have a better understanding of the frequency content of the signals we are sending and receiving. It provides a quantitative measure of the spectral corruption of the channel. With this measure, we are better equipped to use our signal processing to correct for the corruption, analyze the effects of such corruption, or assess performance.

Reason 2: In many applications we look for the presence of a signal or the lack thereof. This sounds simple enough, but in a noisy environment or channel it may be difficult. The Fourier Transform can help. Assume that our signal of interest is a sinusoid with a frequency of f. With time plot 4-81, noise has been added. However, looking at the Fourier Transform of plot 4-82, it is apparent that the sinusoid is there. Voila! The Fourier Transform in action.

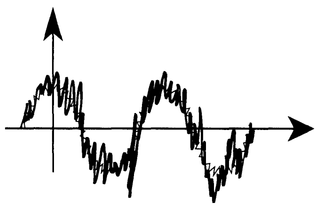


Figure 4-81

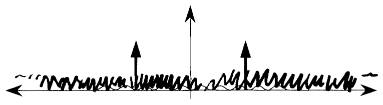


Figure 4-82

The Mathematics Of The Fourier Transform

Now it's time to mathematically define this transform. Don't get caught up with the form of the equation itself, the concepts needed to understand it have already been covered. The Fourier Transform is defined as:

$$X(f) = F[x(t)]$$

$$X(f) = \int_{-\infty}^{+\infty} x(t)\, e^{-j2\pi ft}\, dt$$

It's not as bad as it looks. Let's dissect it a little. First, ignore the integral and look at the inside. We have a signal x(t) multiplied by a complex exponential.

$$x(t)\, e^{-j2\pi ft}\, dt$$

Do you recognize this expression? Isn't that **complex modulation**? Yes, it is. ***Remember that complex modulation is a one-way shift of the spectra.*** *In fact, the value of "f" chosen is what determines how much the spectra shifted and in which direction. In fact, after the modulation is complete,* ***the spectra that was originally at f now resides at the frequency of zero.***

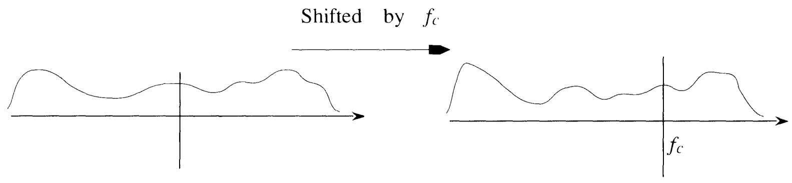


Figure 4-83 Spectrum of x(t) **Figure 4-84**

Let's look at the integral in figure 4-85 and 4-86. We are taking the integral over infinite time which is the area under the curve

Let's look at the integral in figure 4-85 and 4-86. We are taking the integral over infinite time which is the area under the curve of a signal composed of sinusoids. Didn't we see that the integral for sinusoids over infinite time equals zero? Yes, *unless* the frequency of the sinusoid is zero. *First the complex modulation allows us to* choose f which performs a shift of the spectra. The chosen *f* is now at zero frequency. ***The integration makes everything zero except for what was shifted down to zero frequency. Basically, that's all the Fourier Transform is doing***. We pick a frequency of interest and modulate the signal by that frequency so that after integration we have a measure of how much of the signal was at the chosen frequency. kes everything zero except for what has been shifted down to zero frequency.

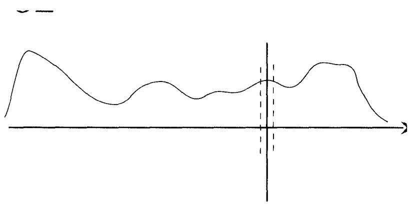


Figure 4-85

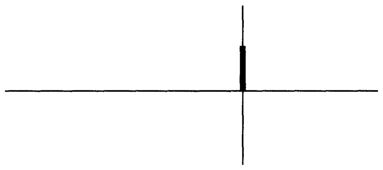


Figure 4-86

Intuitive Example Of The Fourier Transform

Basically, that's all the Fourier Transform is doing, we select a frequency of interest, modulate the signal by that frequency, so that after we integrate (cancel all the non-zero frequency content), we have a measure of how much of the signal was at our chosen frequency.

Some may wish to accuse us of circular reasoning so let's look at a different example. For this example, we will not do the actual calculus to perform the integration. Instead, keep in mind an assumption and two rules previously learned to follow an intuitive approach.

<u>**Assumption**</u>: *Any signal describable in the time domain has a description in the frequency domain.* We may not know exactly what its frequency domain description is right now, but we can assume that it does exist.

Rule #1) ***Multiplication by a complex sinusoid represents a one-way shift of a spectrum.*** The process is called complex modulation.

Rule #2) ***Integration of non-zero frequency sinusoids over all time equals 0.*** By choosing to integrate something over all time, we deliberately force ourselves to look at the DC component which is the only component that may have a non-zero component. We effectively blind ourselves to any time-varying quantities. We can now follow a simple procedure to construct the spectra of any given real waveform x(t).

Let's integrate the given x(t) over all time. Using rule two, we know the zero frequency component and can fill in the f = 0 position on the spectral plot.

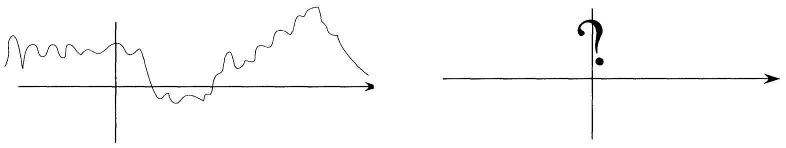
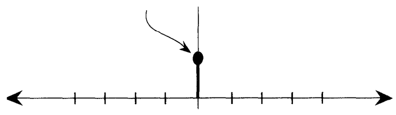


Figure 4-87



We can choose *f*, say f = 3, and modulate, using rule one, the given signal x(t) and call the result x′(t). By using rule two, we can integrate x′(t) and determine the DC component of x′(t). Furthermore, we know that by rule number one that the modulation shifted the spectra of x(t) by f=3. So that the DC component of x ′(t) is the f=3 component of the original x(t). Thus, we can now fill in the point on the spectral plot for f=3.

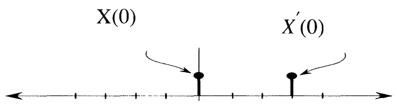
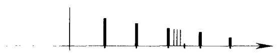


Figure 4-88



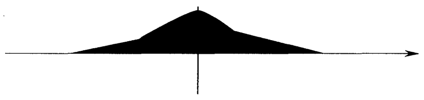


Figure 4-89

Figure 4-90 Entire spectrum of x(t)

This procedure can be continued for other values of *f* (f = 1, f = 2, f = 2.002, f = 2.0024, f = 3.14159..., etc.) Once all the values of *f* are viewed, and the entire spectrum of x(t) is determined, we find X(f) from x(t).

Very nice, but there are a couple of problems with this approach; for example, we have to integrate a signal over all time. Who can wait that long? Furthermore, does this have to be done for each frequency component? Keep in mind that this presentation has tried to heuristically show what the mathematical operations that comprise the Fourier Transform are actually doing, and how these operations may give us the spectral composition of a signal from its time domain description.

Determining the Fourier Transform of a simple mathematical formula may be done analytically. Finding the Fourier Transform of a real-life signal may be different. We will address these differences later under the heading *Fourier Transform of Real-Life Signals*. If you would like more math, continue with the *Fourier Transform Pairs* section.

A More Mathematical Approach To The Fourier Transform

This page presents a slightly more mathematical demonstration of the Fourier transform. If you've no interest please turn the page, you will not need this for following material.

The term "slightly more mathematical" was used because the development will utilize some straight forward elementary calculus. We're not going to provide a development of the delta function.

Let's look at the Fourier Transform of a sinusoid with phase ϕ and non-zero frequency f_c:

$$x(t) = \cos(2\pi f_c t + \phi)$$

$$X(f) = \int_{-\infty}^{+\infty} \cos(2\pi f_c t + \phi)\, e^{-j2\pi f t}\, dt$$

$$= \frac{1}{2}\int_{-\infty}^{+\infty} [e^{j(2\pi f_c t + \phi)} + e^{-j(2\pi f_c t + \phi)}]\, e^{-j2\pi f t}\, dt$$

$$= \frac{1}{2}\int_{-\infty}^{+\infty} [e^{j(2\pi(f_c - f)t + \phi)} + e^{-j(2\pi(f_c - f)t + \phi)}]\, dt$$

$$= \frac{1}{2}\int_{-\infty}^{+\infty} \cos[2\pi(f_c - f)t + \phi]\, dt + \frac{1}{2} j\int_{-\infty}^{+\infty} \sin[2\pi(f_c - f)t + \phi] dt\_$$

$$+ \frac{1}{2}\int_{-\infty}^{+\infty} \cos[2\pi(f_c + f)t + \phi]\, dt + \frac{1}{2} j\int_{-\infty}^{+\infty} \sin[2\pi(f_c + f)t + \phi] dt\}$$

Notice that if "f" is not equal to +/- f_c, we've got the infinite time integral of sinusoids which we know are zero. What about if f = +/- f_c?

For f = +f_c we have,

$$X(f) = \frac{1}{2}\int_{-\infty}^{+\infty} e^{j\phi}\, dt + \frac{1}{2}\int_{-\infty}^{+\infty} e^{-j(2\pi f_c t + \phi)}\}\, dt$$

$$= \frac{1}{2} e^{j\phi}\int_{-\infty}^{+\infty} 1\, dt + \frac{1}{2}\int_{-\infty}^{+\infty} \cos(2\pi f t + \phi)\, dt$$

$$+ \frac{1}{2}\int_{-\infty}^{+\infty} \sin(2\pi f t + \phi)\, dt$$

Similarly for f = -f_c we have,

$$X(f) = \frac{1}{2}\int_{-\infty}^{+\infty} e^{-j\phi}\, dt + \frac{1}{2}\int_{-\infty}^{+\infty} e^{j(2\pi f_c t + \phi)}\, dt$$

$$X(f) = \frac{1}{2}\, e^{-j\phi} \int_{-\infty}^{+\infty} 1\; dt + \frac{1}{2} \int_{-\infty}^{+\infty} \sin(2\pi f t + \phi)\; dt + \frac{1}{2} \int_{-\infty}^{+\infty} \cos(2\pi f t + \phi)\; dt$$

Notice the sinusoidals terms are zero since they are the infinite integral of sinusoids with non-zero frequency.

Now we're left with the just the first term for f=+f$_C$ and the last term when f=-f$_C$

$$X(f) = \frac{1}{2}\, e^{j\phi} \int_{-\infty}^{+\infty} 1\; dt = \frac{1}{2}\, e^{j\phi} \qquad ;t(+\infty) - t(-\infty) = \infty \quad \text{for } f = f_C$$

$$+ \frac{1}{2}\, e^{-j\phi} \int_{-\infty}^{+\infty} 1\; dt = \frac{1}{2}\, e^{j\phi} \qquad ;t(+\infty) - t(-\infty) = \infty \quad \text{for } f = -f_C$$

So we have infinity? What the does that mean?

Well, first let's look at the whole thing. True we've infinity when f is the frequency of the sinusoid (f$_C$ or -f$_C$) but we also figured that the spectrum if zero for all f not equal to +/-f$_C$. Putting these two ideas together, what does this sound like? It's the Delta function we saw earlier. Using this substition we can see:

$$F[\cos(2\pi f_C t + \phi)] = e^{j\phi}\, \delta(f - f_C) + e^{-j\phi}\, \delta(f - f_C)$$

To illustrate, let $\phi = 0$. Then we have the famaliar transform of the cosine function below:

$$\cos(2\pi f_C t) \Longleftrightarrow \delta(f - f_C) + \delta(f + f_C)$$

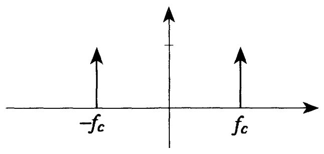


Figure 4-3

What about using the sine function instead of the cosine function?

$$\sin(2\pi ft) = ??$$

We could start from scratch again and follow the same path as we did for cosine or we take a short cut and can use the trigonometric identity $\sin x = \cos(x - \frac{\pi}{2})$.

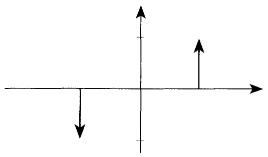


Figure 4-91

$$F[\sin(2\pi f_c t)] = F[\cos(2\pi f_c t - \frac{\pi}{2})]$$

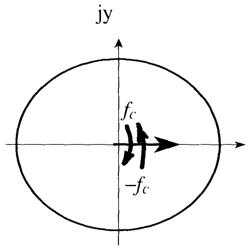
$$= \left[\frac{e^{-j\frac{\pi}{2}}\delta(f - f_c) + e^{j\frac{\pi}{2}}\delta(f + f_c)}{2}\right]$$

$$= \left[\frac{-j\,\delta(f - f_c) + j\,\delta(f + f_c)}{2}\right]$$

$$= j\left[\frac{\delta(f - f_c) + \delta(f + f_c)}{2}\right]$$

Notice the Fourier transform of the sine function is purely imaginary. Don't let this bother you, the imaginary parts and real parts are just used to describe the phase. Notice that the only difference between the cosine and sine function is phase. In fact, we just made use of this fact!

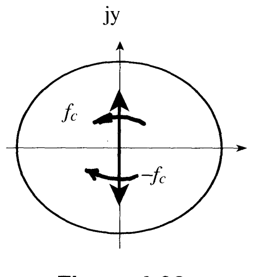
Phase And Magnitudes Of The Fourier Transforms



$$\cos(2\pi f_c t) = \frac{1}{2} e^{j2\pi f_c t} + \frac{1}{2} e^{-j2\pi f_c t}$$

Figure 4-92

The preceding pages plots the Fourier transforms for the sine function and the cosine function. The cosine was strictly real and the sine was stricly imaginary. Typically, however, we tend to view things in terms of amplitude and phase. Especially when we aren't lucky enough to have just strictly real or just strictly imaginary. Viewing the cosine and sine functions Fourier transforms with amplitude and phase we get...



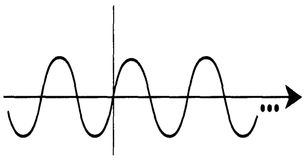
$$sin(2\pi f_c t) = \frac{1}{2j} e^{j2\pi f_c t} - \frac{1}{2j} e^{-j2\pi f_c t}$$

Figure 4-93

We have come full circle at this point. This is exactly the description for sinusoids that was at the very begining of the chapter when time and frequency domain were first mentioned!

What about those phasors, do they still have relevence? Yes, in fact they tie everything together quite nicely. Below are the phasor diagrams for both the cosine and sine function.

In coming full circle we have now cemented the relationship between the two domains with a mathematical relationship, the Fourier transform. The main idea here is that the heuristic presentation that worked in the first few pages continues to work after a more rigouress development. Inspite of the integral calculus and complex mathematics of this transform, it is indeed possible to make sense and utilize Fourier transforms.



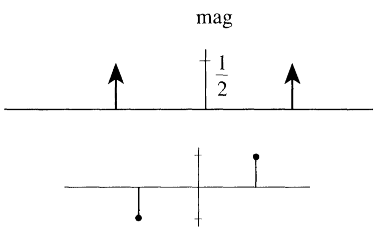


Figure 4-94

$\sin(2\pi f_c t) \Longleftrightarrow \frac{1}{2} j\,[(f-f_c) - (f+f_c)]$

Pair: Sine Function

$\sin(2\pi f_c t) \Longleftrightarrow \frac{1}{2} j\,[(f+f_c)]$

Duality

A quick note on a concept that can save us tim. This concept is called **duality**.

Two points to consider:

Point 1: Did you notice that the Fourier transform of a Delta function ((t)) in time resulted in a constant values in the frequency domain? In other words, a spike at t=0 has frequency components of equal magnitudes for every frequency f.

Point 2: Did you notice the the Inverse fourier transform of a spike in the frequency domain at f=0 results in a constant in time domain? In other words a DC component remains constant over all time.

Notice how if one ignores the axis labels they are the same? Notice how it doesn't matter if we are talking about time or frequency. The fact remains that the "shape" on the left side of the plots transforms to the same "shape" on the right side of the plots regardless of the name we give to the domain? Well, this is an example of what is meant by duality.

We can be more specific andmore mathematical:

$$F[x(t)] = \int_{-\infty}^{+\infty} x(t)\ e^{-j2\pi ft}\ dt = X(f)$$

(We already knew this, just a review)

$$F^{-1}[X(f)] = \int_{-\infty}^{+\infty} X(f)\ e^{j2\pi ft}\ df = x(t)$$

(We knew this too)

Now, not only is the above true, but now we also have:

$$x(-t) = \int_{-\infty}^{+\infty} X(f)\ e^{-j2\pi ft}\ df$$

and

$$X(-f) = \int_{-\infty}^{+\infty} x(t)\, e^{j2\pi ft}\, dt$$

The concept of a transform pair is valid regardless which domain is actually being considered. The transform of a spike is a constant, regardless of whether the spike is in the frequency or the time domain. A rectangular pulse transforms into a sinc pulse, regardless of whether the rectangular pulse refers to a time signal or a spectral content. Etc, etc, etc...

This tends to make a lot of sense when one realizes that the only difference between the Fourier transform and the Inverse Fourier transform is the sign of the exponential (direction of (de)modulation).

The amplitude, phase and frequency are useful measures of the sinusoid. However, not all signals are simple sinusoids, so there are other metrics that are useful in describing these other signals.

CHAPTER 5

Filters and Transfer Functions

Filters and Transfer Functions

We briefly mentioned (signal) filters earlier. The signal filter in concept is no different than a coffee filter or an oil filter. A coffee filter lets the coffee through but filters out the grinds. A signal filter lets some components of the signal through but filters out the other components of the signal (hopefully the noise.) But let's make it a little more formal.

A signal filter (from now on called a filter) is typically drawn as a box (Figure 5-1). By convention *x(t)* represents the filter's input signal, y(t) the output signal. The filter is described by a function that is usually given the letter *h(t)*. Filter impulse response is the name given to *h(t)*.

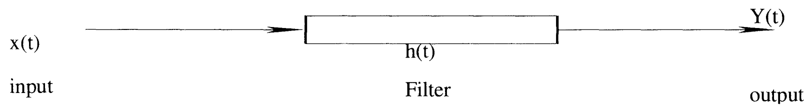


Figure 5-1 A signal Filter in Time Domain

More on this and how exactly the three (x(t), y(t) and h(t)) are related later. How are x(t),y(t) and h(t) related? Typically the frequency domain makes more sense. Often it is the frequency components of a signal that we desire to affect. (remember bass/treble controls on the stereo?).

Let's use the frequency domain, now we have the input spectra labeled by X(f), the output spectra by Y(f). The filter is now described by a function H(f). This is exactly the transform of h(t). H(f) is known as the filter frequency response.

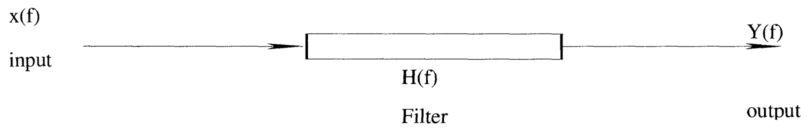


Figure 5-2 A signal Filter in the Frequency Domain

The advantage of using the frequency domain is that the relationship between X(f), Y(f), and H(f) is simple. Use multiplication. As shown below, Y(f) is equal to X(f) times H(f).

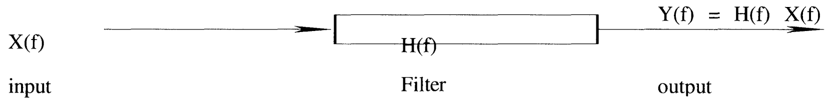


Figure 5-3

You'll see a good reason as to why H(f) is called the transfer function if the equation is re-written as...

$$H(f) = \frac{Y(f)}{X(f)}$$

It's understood that H(f) describes how the input (X(f)) is related to the output (Y(f)). In other words "how the input is 'transferred' to the output by the function H(f)."

Remember, since H(f) is a Fourier transform of h(t), we know that H(f) could well be complex even if h(t) is strictly real. Thus, we'll be interested in the magnitude of H(f) as well as the phase of H(f). Discussions often focus more on the magnitude. This is common, but don't get lulled into thinking the phase doesn't matter, as we know, sometimes it does.

What Exactly Is h(t)?

Let's try a thought experiment, if someone handed us an unknown filter (H(f) is unknown), how could we figure out what it was?

A simple way would be to put in a waveform that consisted of equal amounts of all frequencies. By doing that, we could look at the resulting output from the unknown filter and see how each of these input frequencies were affected. We could identify the filter's frequency response, since it's the ratio of the input and the output. In fact, for X(f)=1, then H(f)=Y(f)! Thus, we've directly measured the filter's frequency response H(f).

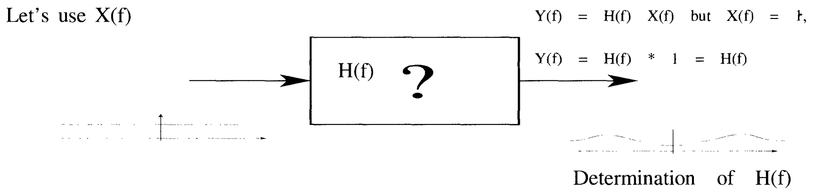


Figure 5-4

Simple enough, but how can we put in such an input spectrum (X(f)=1 for all f)? Looking at the Fourier transform pairs, we find that such a spectrum exists if the input is a Delta function. So, using an impulse as input, the time domain description of the above block diagram is...

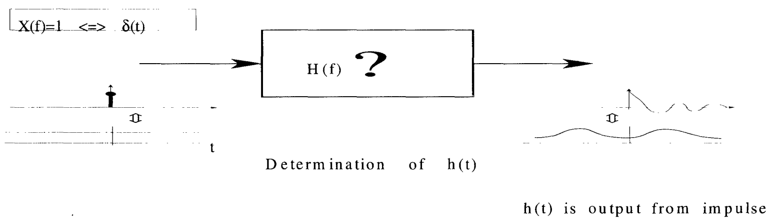


Figure 5-5

Two Views Of The Same Thing

When using frequency domain we used *X(f)=1* as an input and got the frequency description of the filter as the output is *H(f)*. *H(f)* has been called the filter's "transfer function."

Alternatively, the time domain descriptions could be viewed. Here we used an input Delta function to give us the time domain description of the filter. Because of this, the time descriptor *h(t)* is known as the filter's impulse response (the response due to an impulse being used as input).

Either description, *h(t)* or *H(f)*, may be used to completely describe a filter. Essentially, they describe the same thing since they are uniquely related to each other by the Fourier transform. Both views have there merits and uses.

What About Using The Time Domain To Describe Filters ?

The question remains, "If *Y(f) = H(f) X(f)*, what can we say about how the time domain descriptions *x(t)*, *h(t)*, and *y(t)* are related."

We can figure this out. Remember the Inverse Fourier transform? Let's use that and what we learned previously. It's not necessarily evident what's going on here for those with a dislike for calculus, but it's not too terribly difficult if you are interested.

Mathematical Derivation Of The x(t), y(t), and h(t) Relationship

Let's start with what we have accepted so far:

$x(t) \quad X(f), \quad y(t) \quad Y(f), \quad h(t) \quad H(f)$ and,

$Y(f) = H(f)\,X(f)$ let's take the IFT (Inverse Fourier transform)

$y(t) = F^{-1}[Y(f)] = F^{-1}[H(f)\,X(f)]$

by definition, we get:

$$y(t) = \int_{-\infty}^{\infty} y(f)\, e^{j2\pi ft}\, df = \int_{-\infty}^{\infty} H(f)\, X(f)\, e^{j2\pi ft} dt$$

but we know $X(f)=F[x(t')]$ is used for dummy variable

so rewriting we have,

$$y(t) = \int_{-\infty}^{\infty} H(f)\, [\int_{-\infty}^{\infty} x(t')\, e^{-j2\pi ft'}\, dt'\,]\, e^{j2\pi ft} df$$

rewriting again,

$$y(t)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} H(f)\, x(t')\, e^{-j2\pi ft'}\, e^{j2\pi ft}\, dt'\, df$$

now let's relate t' and t by $\tau = t - t'$, substitute and rewrite

$$y(t) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} H(f)\, x(t-\tau)\, e^{-j2\pi f\tau}\, d\tau\, df$$

separate and reorder integral,

$$y(t) = \int_{-\infty}^{\infty} x(t-\tau)\, [\int_{-\infty}^{\infty} H(f)\, e^{j2\pi f\tau}\, df\,]\, d\tau$$

this is the IFT of H(f), so it equals h(τ).

Then let's rewrite now and obtain,

$$y(t) = \int_{-\infty}^{\infty} x(t-\tau)\, h(\tau)\, d\tau$$

Certainly we are getting close. What else can we do? Unfortunately (or maybe fortunately), this is the end of the road, we're stuck with this. However, this integral is so useful its given a name and a symbol of its own. It's known as the convolution integral. The asterisk (*) is used to denote convolution, please don't confuse this with multiplication, it's not the same.

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(t-\tau)\, h(\tau)\, d\tau$$

Let's redraw the filter block diagram with this new information to tie it together:

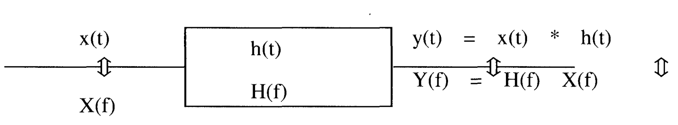


Figure 5-6

Convolution Integral, Intuitive Perspective

The Fourier transform was an integral equation, yet it made sense on a basic level. The same can be said for the convolution integral. This time we'll work backwards. Iinstead of dissecting the integral equation to understand it, let's synthesize it from a string of arguments.

Argument #1) Input an impulse into a filter and the output is the impulse response, h(t). We established that a filter is completely defined by its impulse response.

Argument #2) Input signal as a collection of impulses:

Imagine the waveform x(t) as shown -

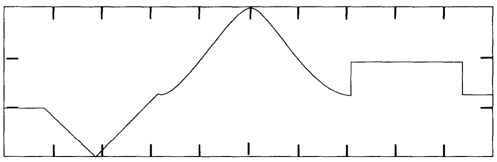


Figure 5-8

Let's draw some impulses, say every second, equal to the value of x(t) at that time (in other words, weighted impulses).

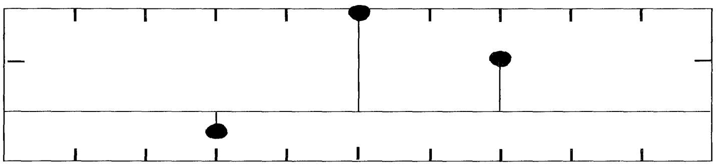


Figure 5-7

Try drawing more of these impulses this time, say on every half second. This still doesn't look like much.

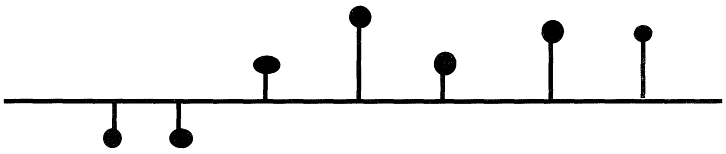


Figure 5-9

Now, try every tenth of a second. This is starting to look pretty good. It's not too bad of a representation of the original signal x(t).

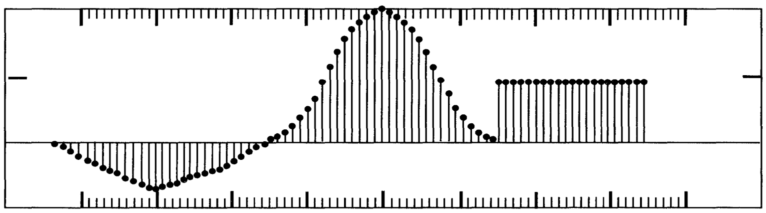


Figure 5-10

What if we drew the impulses every infinitesimal time interval? We would get exactly the original x(t).

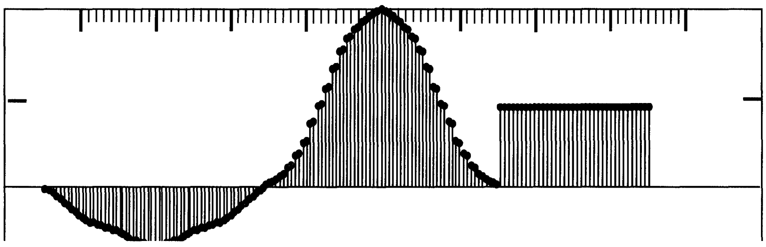


Figure 5-11

We can indeed view the signal x(t) as a collection of individual impulses.

Argument #3) Without explicitly stating it, we assumed our discussion is based on linear time invariant systems. (Non-linear systems and their analysis can get quite complicated and are beyond the scope of this book.) What linear means to us is a linear system that has the property that "the sum of the input components produce an output equal to the sum of the outputs if the individual components." This is known as **superposition**. Pictorially it is simple:

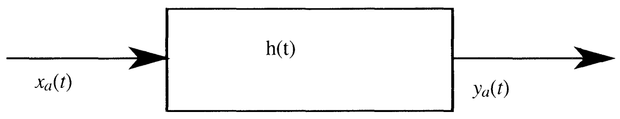


Figure 5-12

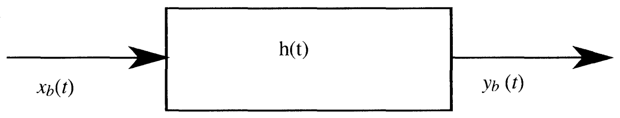


Figure 5-13

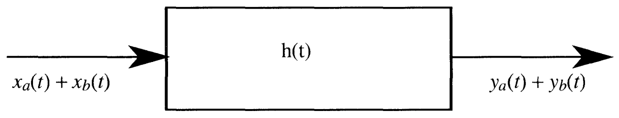


Figure 5-14

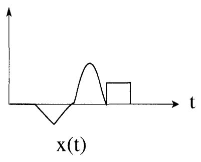
Time invariant means the systems impulse response does not change with time. Impulses coming in at different times would still have the same response (assuming equivalent states).

Combine #1 and #2 and #3 to form the logical string. Input an impulse into the filter and the output is the impulse response (argument #1).

Every impulse train that the input signal x is constructed from, argument #2, will have a corresponding impulse response generated by it, arguments #1 and #2.

The actual output of the filter due to signal x(t) is the additive action of all these impulse responses (argument # 3).

Assume the original signal x(t) and imagine an impulse response that looks like the example below.



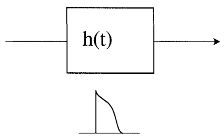


Figure 5-15

Figure 5-16

Let's recreate the "a signal is a collection of impulse" plots with the filter's output y(t) represented by the addition of these impulse responses.

Draw some impulse responses, say every second, equal to the value of x(t) at that time. As expected, there is not too much to say about this.

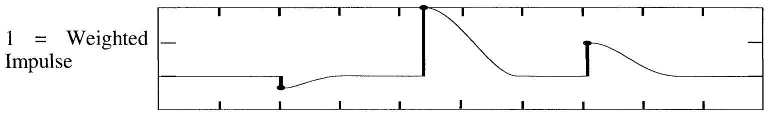


Figure 5-17

Try drawing more of these impulse responses this time, say on every half second. Again, it still doesn't look like much.

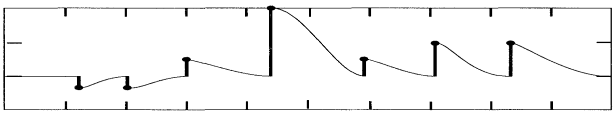


Figure 5-18

Now try every tenth of a second. This is starting to look pretty good.

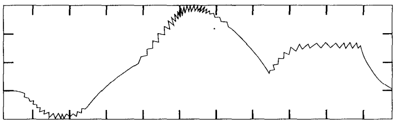


Figure 5-19

What was meat when we drew the impulses every infinitesimal time interval?Actually we would get exactly the true output signal y(t).

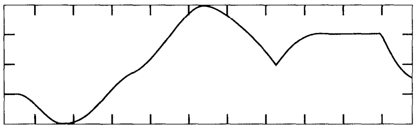


Figure 5-20

Take a minute to review how these presented arguments and accompanying plots lead us to the convolution integral. After that, we'll step up the above argument with a little more mathematics.

Each point of our input signal *x(t)* can be represented by an weighted impulse (by weighted we mean an impulse whose area under the curve (i.e. integral) is equal to *x(t)* at that time. Say, for any time = tau, we get:

$$y(\tau) = \int_{-\infty}^{\infty} \delta\,(t-\tau)\;\; h(t)\; dt$$

One impulse at time t=τ ((t-τ)) into the filter will generate one impulse response starting at time t=τ (h(t-τ)).

By superposition, the output of a collection of inputs equals the collection of the outputs of the individual inputs. This

This means the output signal y(t) is the collection of the outputs of each of the impulse inputs. (Since again we've got an infinite number, we'll get an integral.)

Voila, the convolution integral. In spite of the looks of this integral equation, it really does make sense.

Convolution In The Time Domain = Multiplication In The Frequency Domain

In describing a filter we just naturally assumed that a filter would have the characteristics that Y(f) = H(f) X(f). This is because we tend to think as filters as acting on frequencies, by attenuating or amplifying certain frequency components. This characteristic come about from our definition and concept of a filter.

In order to have the frequency domain characteristic of Y(f) = H(f)· X(f), we found that the time domain characteristics were given by the convolution integral. This was done by use of the definition of the relation between the time and frequency domains (the Fourier transform).

As seen earlier, convolution is so prevalent that it' given a shorthand notation, the asterisk:

$$x(t) * h(t) = \int_{-\infty}^{\infty} h(t-\tau)x(f)df$$

Putting this more succinctly we have:

If y(t) = x(t) * h(t), then Y(f) = H(f)·X(f)

"Convolution in the time domain equals multiplication in the Frequency domain!"

Duality Revisited

Could duality come into play with regards to the above quote? If duality means that we swap "f" and "t," does it also mean that "convolution" and "multiplication" could get swapped as well? Does "multiplication in the time domain equal convolution int the frequency domain?"

Yes. The proof of this is left as an exercise to the reader.

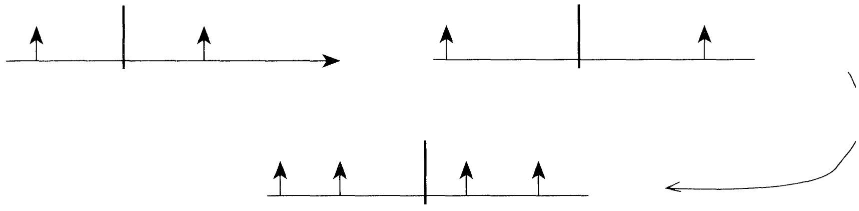


Figure 5-21

A familiar example of this would be the multiplication of two sinusoids x(t) and y(t).

The reason that convolution came up was because we were talking about filters. Let's get back to that.

Ideal Filter Types

It is impossible to draw a perfect circle, nonetheless, the concept of a perfect circle is commonly used.

It is much the same with the ideal filters presented. They are impossible to construct, but they are useful idealizations. The basic ideal filter types are; low-pass, high-pass, bandpass, and bandstop.

Ideal Low-Pass Filter: A ideal lowpass filter will allow all frequency components below the cutoff frequency to pass unharmed, but prevent the passage of frequency components above the cutoff frequency. The set of frequencies allowed to pass through is called the *Passband.* The set of frequencies stopped is the *Stopband.*

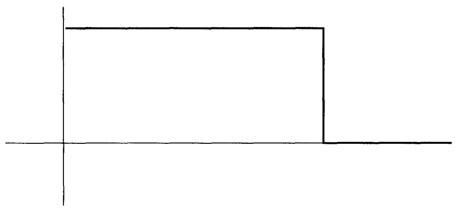


Figure 5-22

Ideal High-Pass Filter: Merely the opposite of a lowpass filter. Highs go through but the low frequencies get stopped. Again the boundary between these two is determined by the cutoff frequency.

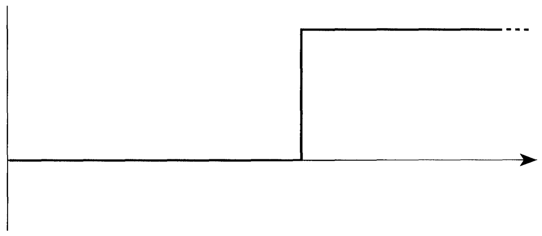


Figure 5-23

Ideal Bandpass Filter: Here only the frequencies that lie between the upper bandedge and the lower bandedge get through. Thus, there are two stopbands and one passband.

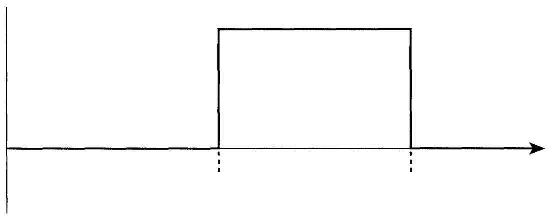


Figure 5-24

Ideal Bandstop Filter: As you probably guessed, it is the opposite of the bandpass. Frequencies between the lower bandedge and upper bandedge get stopped. Now there's two passbands and one stopband.

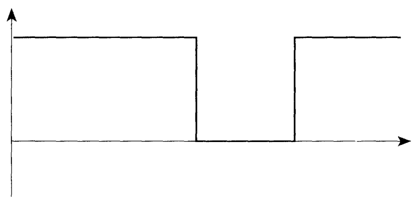


Figure 5-25

Why Can't You Make An Ideal Filter? (Causality)

It's a good question. Also, its gives us one of the first opportunities to use what we learned about Fourier transforms.

Let's look closer at the ideal lowpass filter.

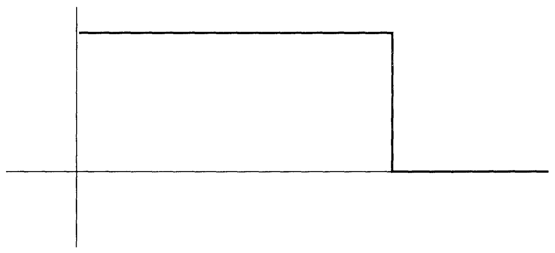


Figure 5-26

We know that we can determine the impulse response h(t) from its frequency description H(f). We can do so by calculating the inverse Fourier transform, but let's save ourselves the bother and refer back at the Transform Pair section where we'd find the pair:

Thus, from transform pairs and duality we now know that the ideal filter has an impulse response defined by the sinc pulse (shown below)

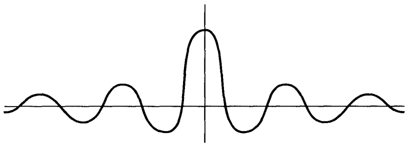


Figure 5-27

Great, so now we can go ahead and build the ideal filter, right? No. Take a look at the sinc pulse plot, notice the dots at the left and right. They signify that the impulse response continues for ever and ever. Well, this is one reason the ideal filter is impossible. Eternal time systems are a bit difficult to build.

Remember, when we plot impulse responses, by convention, we say the input impulse "hits" the filter at time t=0. Look again at the impulse response and notice how it has appreciable content to the left of t=0. This corresponds to the time before the input impulse even get to the filter! More specifically, the ideal filter would have to start reacting to something that hasn't even occurred yet! (A difficult task at best). A term to describe this task is "**non-causal.**" To build a non-causal filter is beyond our means.

We'd really like the filters to be causal, which is to say h(t) = 0 for t = 0.

The observant reader may say, "We could just build a delay in the filter." This could be thought of as shifting the impulse response plot to the right. If we could delay this enough, maybe we could move the whole impulse response to the right of t=0, then it'd be causal and we'd be able to build it. Great! But, don't forget the impulse response needed for the ideal LPF is infinite in duration. Thus, we'd have to add a lot of delay, infinite in fact. So much for that idea.

However, the idea of putting delay into the impulse response is not an entirely bad idea. It does, indeed, get us an impulse response closer to what we needed.

Almost Realistic Filter Types

Here we take a step towards a little more realism in our description of filters. One thing that made the ideal filters so impossible was the discontinuity, or jump in the filter's frequency response from 0 in the stopband to 1 in the passband. Let's get over this hurdle by allowing values in between 0 and 1. This added concept is the transition band, bordered by the stopband edge and the passband edge.

Almost Realistic Low-Pass:

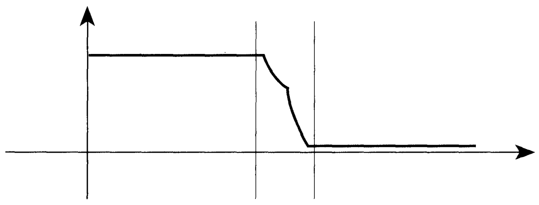


Figure 5-28

Almost Realistic High-Pass:

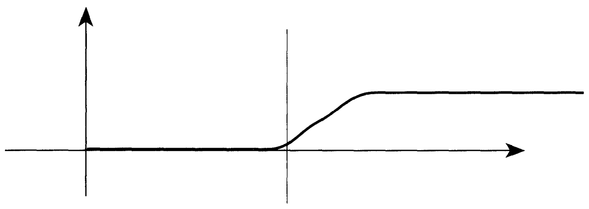


Figure 5-29

Realistic Filters

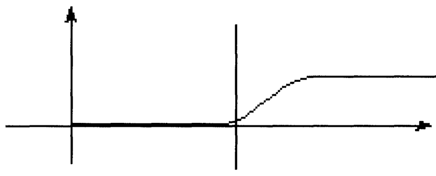


Figure 5-30

To be able to build a filter we must allow some additional concessions.

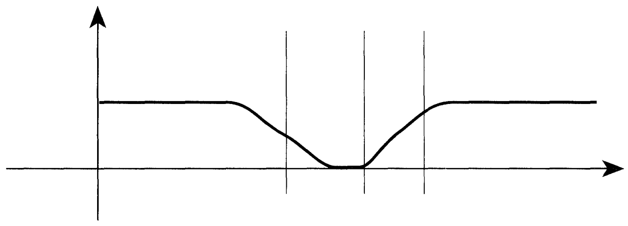


Figure 5-31

First, we allow for variation in the pass band. The pass-band response is allowed to vary somewhat around the desired response of H(f)=1. It could be greater than 1 or a less (of course we'd generally like it to be really close to 1). This is sometimes called **passband ripple**.

Secondly, we relax the constraint of the stop-band. Now, it doesn't have to exactly equal zero (of course we'd like generally like it to be really close to 0). The amount of signal allowed through the stopband is called **stopband attentuation**.

Filter Specification

Now, we have all the basic parameters we need to specify a basic filter that we wish to build. These are:

These are the parameters generally regarded as filter design specifications. (Some may argue that I've left out such things as phase response, overshoot, etc. Fine, but I will still call the above parameters the basic filter design parameters.)

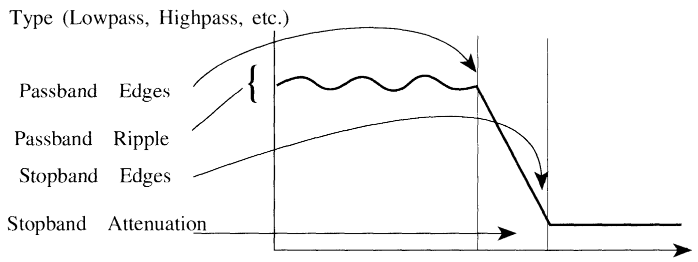


Figure 5-32

Implementing The Filter From These Parameters:

From a set of filter design parameters, there are a variety of techniques for implementing an actual filter. Some of the classical filter implementation types for continuous time filters are Chebychev, Butterworth, Elliptic. Traditionally, continuous time implementations of these filters used capacitors, resistor, inductors in passive filters or active filters with the inclusion of op-amps.

Our goal is Digital Signal Processing. I know we still haven't formally started the DSP discussion yet - hang in there, the continuous time basics are important. Let's delve into filter implementation issues as digital filter implementation issues and save this for when we get to the strict DSP sections.

Return To Fourier Transforms

We defined the Fourier transform with an equation. We investigated the motives for such an equation and it seemed to make sense. Next, we took a survey of Fourier transform pairs of some simple mathematical functions of interest.

Let's apply the concept to common signals. Let's use our knowledge to see what happens when we want use the Fourier transform to determine the spectra of signals that we'll find in real-life. By real-life, we simply mean something that we might measure over finite time.

Finite Measuring Time

We actually give a technical term and discussion about the fact that we'll use a finite measurement time on the signal we wish to apply Fourier analysis. The Fourier integral has plus infinity and and minus infinity as the limits of integration . When we limit our time "window" we see that we aren't doing a strict Fourier analysis. We're kind of "fudging" a bit. Thus, the results we get from our limited Fourier analysis will be somewhat different. Let's demonstrate. Below, you'll find the sinusoid in its infinitely long time representation along with its spikey Fourier representation.

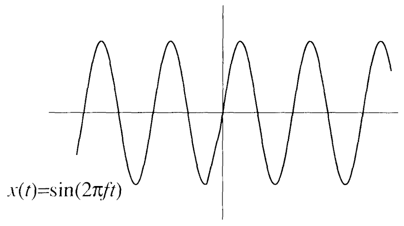


Figure 5-33

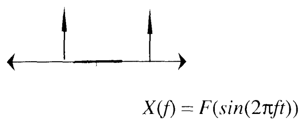


Figure 5-34

Let's consider what happens if we take only a small portion of that sinusoid. In other words, we'll use a finite window instead of minus infinite to plus infinite. Let's say we'll examine the "sinusoid" from 8:21 AM to 8:23 AM. We might draw this BC (before coffee) time domain waveform as shown below:

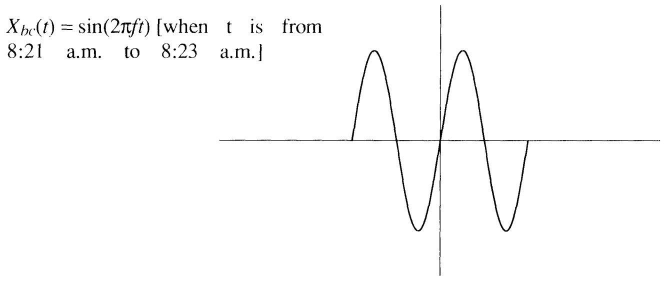


Figure 5-35

The question remains, what's its frequency domain description?

More On Windowing

Let's define xbc(t) a little more properly.

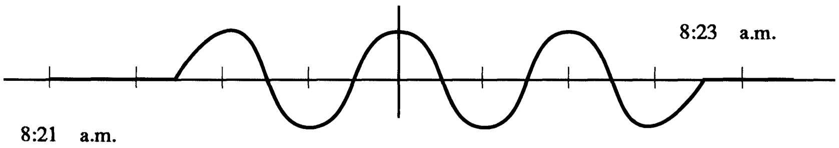


Figure 5-36

Now with this proper definition we could calculate the Fourier transform of $x_{bc}(t)$ by direct application of the Fourier Transform Equation and get an answer. Who wants to go through the steps of calculus? There is no need to do that, there's a more illustrative solution. We'll consider $x_{bc}(t)$ to be made up by the product of two other "simple" signals, w(t) and s(t).

$$x_{bc}(t)=w(t)\cdot s(t)$$

where: w(t) is a rectangular pulse (we've examined these before)

s(t) is an infinite sinusoid

The three signals xbc(t), w(t), and s(t) are plotted below to verify that, indeed, the product of x(t) and s(t) is a valid representation.

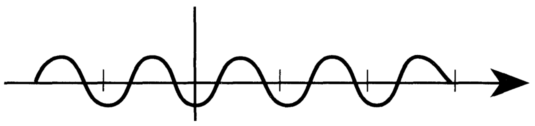


Figure 5-37

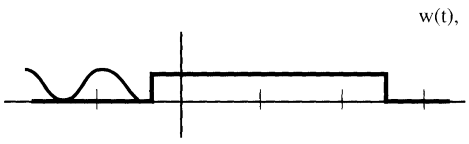


Figure 5-38

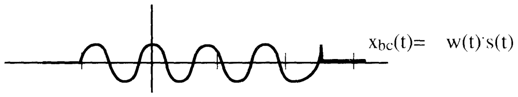


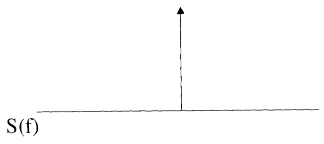
Figure 5-39

By viewing $x_{bc}(t)$ composed in this way, we can take an shortcut around the calculus computations using the dual of "convolution in the time domain equals multiplication in the frequency domain." The dual of this states:

"Multiplication in the time domain is convolution in the frequency domain."

Knowing this, we can directly construct the frequency domain representation of $x_{bc}(t)$ is $(X_{bc}(f))$.

To convolve the spectra of s(t) with w(t), written as S(f)*W(f). We'll need to know S(f) and W(f), luckily we've already seen them in the Fourier transform pairs sections. Here they are,



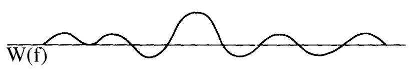


Figure 5-40

Figure 5-41

Think of convolution as it was presented in the filtering section paralleled below. Signal S is convoloved with W by considering W to be the "impulse response" of a filter. (If the thought of sticking frequency domain signals into filters bothers you, just forget about what domain they are in and consider them by their shapes, that's why we've stripped off the "f" in S(f). Remember we're only interested in the convolution process.

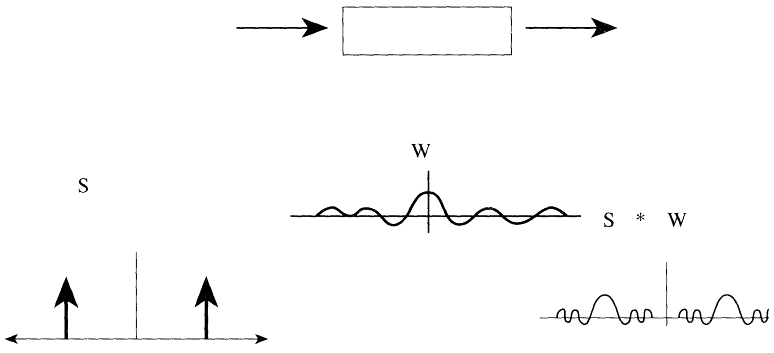


Figure 5-42

Finally, we can see what the spectrum of this windowed sinusoid look like.

Observation #1: Notice that $X_{bc}(f)$ is not a simple spike like the infinite time sinusoid was. It's been spread out to infinity.

Observation #2: Here we chose to view the windowing effect as convolving the frequency responses of the window and the input sinusoid to get the resulting plot. Since we chose the input as a sinusoid we could have looked at it in a different way. We know multiplication of a signal by a sinusoid is modulation. Verify that the spectrum of the window could also be viewed as having been modulated up and down by the sinusoid. It's amazing how you can view the same thing from different angles and it still makes sense.

Observation #3: There's an interesting interplay between time and frequency stemming from the fact that time and frequency are reciprocals of each other. Remember the duality issue between convolution and multiplication? Well, here's a close relative - "A finite time signal has infinite frequency content." We used S to stand for the the signal and W for the window.

Windowing Interplay

Use the same setup as before but change the length of the time window that we'll use to examine the sinusoid. Instead of a couple of minutes before coffee, let's examine it all morning from 9am to 11:30 am. We'll call this $x_{am}(t)$, what's its spectrum look like now?

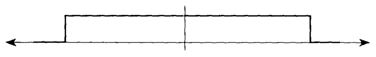


Figure 5-45

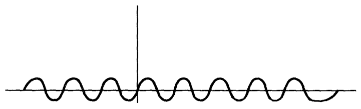


Figure 5-47

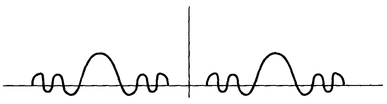


Figure 5-48

With the a different window function w we'll have to re-work our determination of its frequency description W. We see that the longer w(t) is "on" (has an amplitude of 1), the narrower its frequency domain description is. Since we end up convolving our spikey (S) with a narrower (W), see below. Notice now we get a representation closer to what the actual Fourier determination gives us, this makes sense.

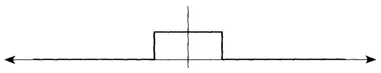


Figure 5-49

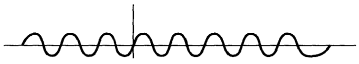


Figure 5-50

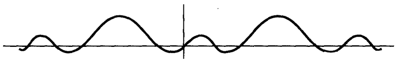


Figure 5-51

Now let's go the other way. Let's choose to examine the signal over a one second intervals call this signal $x_{1sec}(t)$. Again we guess what the new window's frequency description will look like and do the convolution. With such a short duration window , w(t) we get a very broad spectral representation, W(f). Thus, we might get something like this:

Now the frequency determination is not as good. We can still determine the sinusoids exact frequency by picking the biggest the "peak" of the very broad bump.

But, if there was the slightest amount of noise in our measurements we'd have something resembling the plot below. Try to tell the exact frequency of the signal.

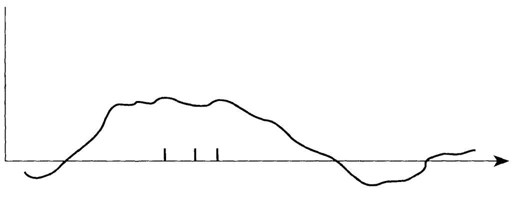


Figure 5-52

This bit of noise wouldn't have bothered us as much in the signal $x_{am}(t)$.

Windows Wrap-up

Hopefully the preceding pages have driven home the point that the chosen window can have a big impact on your spectral analysis. It's a simple point, but one that is sometimes forgotten. For instance, forgetful engineers have been known to try to do spectral analysis in the 20-50 Hz region using a window only a few millisecond long. The figure below may give an intuitive feel to how accurate their results might have been.

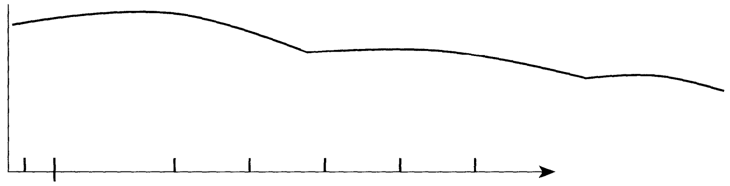


Figure 5-53

Alternative Windows

We may have lead the reader erroneously to believe that the only window function to use is the rectangular window that we've been discussing. This is not true. It just happens to be the simplest to use and easiest for illustrative purposes. In the rectangular window we presented, you're either looking at the signal or you're not. There's a negative point about the rectangular window and that is its frequency domain description. W(f) is bumpy with many local minimas and local maximas (sidelobes).

A better approach may be to avoid using the rectangular window and its "I'm looking, I'm not looking" approach. A different windowing function will allow for a more gradual transition between *looking and not looking*.

See the example of the Hamming Window. Instead of multiplying the signal s(t) by the rectangular window w(t) of duration of two minutes, we'll use the hamming window $w_{hamm}(t)$ of the same duration. Start viewing at the signal a little bit at first, then more and more till we taper the view back down to zero.

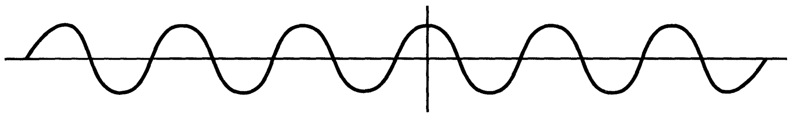


Figure 5-54

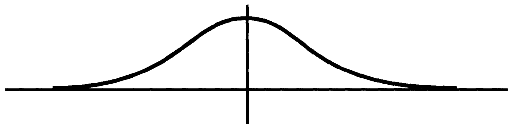
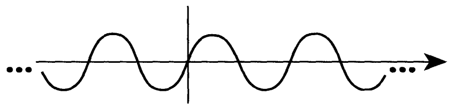


Figure 5-55



Figures 5-56

Notice that the frequency description of $w_{hamm}(t)$ ($W_{hamm}(f)$) is a smoother function.

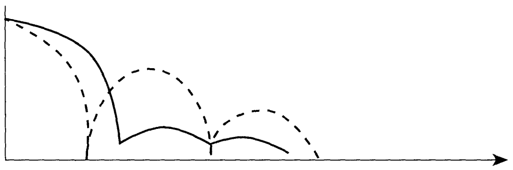
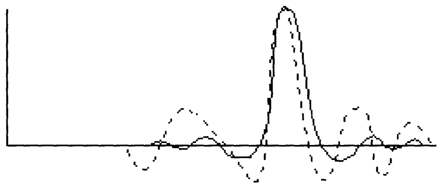


Figure 5-57

$W_{hamm}(f)$

Thus, when we convolve S(f) and $W_{hamm}(f)$ to get the spectral estimate, we get a better plot. By better, we mean lower sidelobe levels.

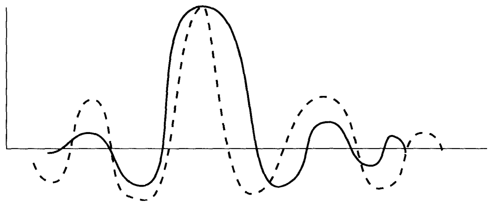


Figures 5-58

Spectrum of x using a Hamming Window

Windows Caveat

Compare the plots of the spectral estimates for the two minute rectangular window and the two minute hamming window on the next page.



Figures 5-59

Indeed, the Hamming Window signal's spectra have lower sidelobes, but notice how it has broadened as well.

Conversely the rectangular windowed signal's spectra is narrower but the sidelobes are higher.

That's the window tradeoff: sidelobe suppression vs. spectral widening. There are many possible window functions to choose from. Each falls in its own niche within the window tradeoff. Some of the more common window functions are, rectangular, hamming, hanning, gaussian, blackman harris.

The Bandwidth Concept

Armed with the Fourier transform and choosing our windows properly, we may determine the spectra of various signals.

Voice Signals

Let's look at an arbitrary voice spectra. We'll find most of its content is located in the maybe 200Hz to 3200 Hz range. This total span of frequencies where there are appreciable spectral components, 3200 Hz, is called the **Signal Band Width** . Thus, any system which is used to carry voice signals should have the ability to carry all these frequencies. (In fact, the frequencies listed here are the frequencies generally used for telephone applications). If this is so, we say that this system has a System Band Width of 3200 Hz. In general, it's a good idea for the system bandwidth at least as large as the signal bandwidth for good results.

Musical Signals

The range of human hearing is generally 20 to 20,000 Hz. So, a good musical reproduction system should have a bandwidth of 20,000 Hz.

Why does an (mono) FM radio signal have only a bandwidth of 15000 Hz? It would be better if it was 20,000 Hz. However, to allow for this much bandwidth, the broadcast station would have to use up more of the radio spectra (or radio band). The bandwidth of 15000 Hz was seen as a good compromise between getting a "good" quality signal to broadcast and conserving more of the band. So by it's bandlimiting approach (system bandwidth smaller than the input signal bandwidth) we don't get a "perfect" musical reproduction.

The Baseband Concept:

The term baseband is used to describe an original signal before any modulation takes place. In the above example of musical signals in a radio broadcast we can regard 0 to 15,000 Hz as the baseband of the signal. After this music is modulated up to radio frequencies (in the hundreds of megahertz range) it is no longer considered in the baseband. In order for the listeners to enjoy the music, their radios must de-modulate the radio signal to bring the music back to baseband.

Modulation does not effect the signal bandwidth. This bandwidth remains constant throughout the modulation and demodulation process and it is not effected by where it is modulated.

A general rule of thumb: For a signal at baseband, its bandwidth can generally be regarded as the frequency for which the signal has any appreciable content.

CHAPTER 6

General DSP Concepts

What Exactly Is DSP?

In dissecting the term *digital signal processing*, we know that digital is a given. Let's examine what is meant by **signal processing.**

Signal processing is a general term used to describe the various functions are performed on signals usually to improve or enhance the original signal. The specific improvement or enhancement differs greatly from application to application and the type of signals involved. Perhaps a quick sketch of various applications of signal processing, specific signals and types of processing will help. Let's start with a few classic examples.

General Signal Processing

Communications Example: A communications engineer may be foremost concerned with the proper transfer and reception of information from point A to point B over a channel or medium. Any signal will is affected in some way by the channel. To extract the maximum information, it is necessary to counteract the channel effects. That is, filtering may be used to filter out the unwanted added noise or interference. Automatic gain control may possibly be used to compensate for fading.

Control Example: In control engineering (e.g., manufacturing quality/process control, chemical process control, vehicle stabilization) the actual control performances done by taking measurements of the process, then initiating a corrective action based upon the measurement. Signal processing becomes involved when this measurement may be affected by noise and require filtering for accurate, stable results. Further, signal generation may be required to initiate the desired corrective action.

Storage or Retrieval Example: storage or Retrieval is a fancy way of saying *recording* or signal storage. Here, the goal is to archive the signal so that extracting information is maximized on storage and on playback. Again, filtering is helpful.

The word "classic" is used to describe the signal processing used in the examples because the problems with their theory and solutions have been known for a long time, that is since the beginning of radio. Simple components such as tubes, transistors, inductors, capacitors, resistors and vinyl or magnetic tape, are used to compose the elements of the signal processing gear. The functional elements of signal processing are filters, variable gain amplifiers, mixers, and oscillators.

DSP composes the most elemental and most important aspects of signal processing today and it offers solutions to these classic problems as well. However, with the advantages that DSP offers, signal processing has moved beyond the "same old thing." Signal processing ideas dabbled with in research labs and labeled "interesting but unrealistic" have found new life in a variety of consumer as well as military and industrial products.

DSP's Newer Uses

We will look at those necessary and effective classic approaches to see how DSP is used to implement those functions in later sections.

Communications Example: In addition to the noise and interference introduced by the channel, an echo is often introduced as well. Have you ever seen the "ghosting" of images on your TV set or heard an echo on a long distance telephone call? DSP made possible the implementation of "echo cancellers" of quality previously beyond the reach of the analog electronics.

Storage/Retrieval Example: Think of the compact audio disc. This is perhaps one of the most visible examples of a digital storage and playback system. Earlier forms of storage and playback such as the LP or cassette tapes demanded, accurate and expensive motors to minimize the warbling sound resulting from motor speed variations. Each time a LP or cassette is played back, mechanical wear degrades the quality of the achieving medium. Additionally, the medium itself generates additional components of noise; surface noise for LPs and tape hiss. Storing digitally eliminates the need for expensive motors and no archival degradation takes place on playback. The medium itself introduces no additional noise to the system. The market acceptance of the compact disc demonstrates that these advantages can be achieved within market cost constraints.

A major factor in the introduction of DSP into a variety of areas has been the decreasing cost of digital micro-electronics and digital memories. The presence of new DSP implementations will likely continue as micro-electronic components continue to increase in performance while decreasing in cost.

We may be accused of skipping over the tempting "wow factor" and "wonder toys" associated with DSP; those machines that recognize and respond to the spoken word, machines that speak for themselves; real time signature analysis; and machine vision. While such DSP functioning does exist today it is to a large degree, so application-specific that it's understanding would not be very portable from one to the next. This is a publication concerning DSP and not the complexities of voice recognition algorithms. Realize that DSP applications are based, to a great extent, on the classical techniques already introduced such as with filtering.

Defining Digital Filtering

It should be made clear exactly what is meant by the term *digital filtering*. The **digital filtering** is filtering which takes place by the mathematical manipulation of the numbers which representing the signal. In fact, for the balance of the presentation, we assume that the sampling and reconstruction conditions are properly met by the data conversion sub-systems. Additionally, the appropriate choice of sample word-length is assumed based upon the sampled signals SNR and the dynamic range. Focusing strictly on the digital portion of the system, we will deal with the effects of numerical manipulations on the signal samples.

Introduction To Digital Filtering

The *moving average operator* is the first hint of a digital filter. The filter needs an input signal to operate on, an operation to perform, and an output signal. We chose an arbitrary sample sine wave of 0.1fs as the input. See figure 6-1.

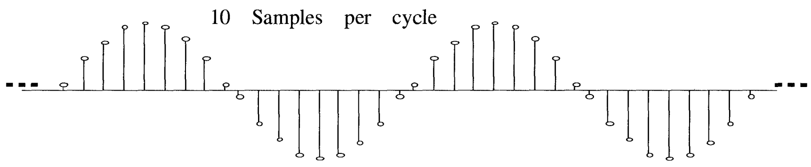


Figure 6-1 An Arbitrary Sampled Input Signal

For the moving average operator, the operation performed is an averaging of the samples. We sum up a few samples, then divide the sum by the number of samples that we added together, and voila, an average! *Moving* means that we find an average for one sample, then move to another, find an average, and so on. The output is formed by looking at the sequential values obtained by each step of the averaging process.

Deciding how many samples are averaged, determines *the order of the filter*. Imagine this with the input and output signals below.

x(n) = input signal

y(n) = output signal = moving average of x(n)

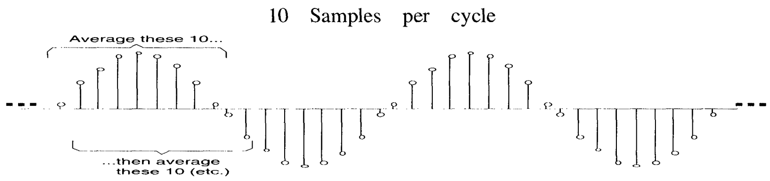


Figure 6-2

The input and output signals differ. Is this really a filter in terms of what we have previously discussed? If it is, why haven't we discussed its frequency characteristics? Where is the convolution in the time domain? To answer these questions, we step up our notation used to represent and manipulate sampled signals.

Representation Of Sampled Signals

Sampled signals, or more accurately, discrete signals, have acquired a representation of their own. Below is an overview.

x(n): This is a signal called *x*. With a value it takes at integer value of *n*. If *n* is explicitly stated, (i.e., x(5)) means the fifth sample of signal x. If *n*, is left as an unspecified integer value (i.e., we have x(n)), then we are generally speaking of the entire signal *x*.

x(n-k): This is a delayed version of the signal *x(n)*. It is delayed by *k* samples. To see this, view the sketch in figure 6-3 and let's use k = 4. Now, compare the signals x(n) and x(n-4). Notice that x(n-4) eventually takes on the same values of x(n) just four samples later. For instance, for n = 10, k = 4 we have if, x(10) = 100, then x(10) = x(14-4) = 100.

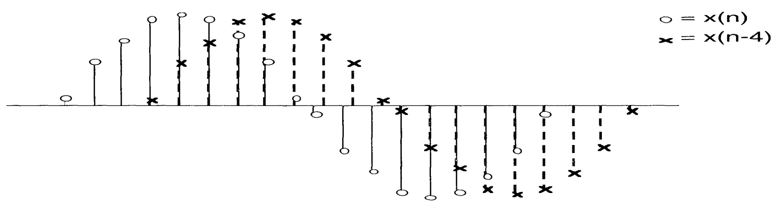


Figure 6-3 x(n) Delayed by Four Units

N yx(i): Summation of samples of signal x. Here, the samples x(1), x(2), x(3), ..., N are summed together.

Moving Average Operator

We can now continue the investigation of the moving average operation adding a more mathematical description. For the input, we chose a sine wave sampled at 0.1fs. This can be written as:

x(n) $= \sin(2\pi n T_S)$, where Ts is the sampling period

$= \sin(2\pi n f/f_S)$, but f=0.1fs so this is also

$= \sin(2\pi n 0.1)$

Defining the output waveform y(n) be the moving average of ten samples of x(n.) Rewriting this mathematically gives insight.

$$y(n) = 1/10\ [\ x(n) + x(n\text{-}1) + x(n\text{-}2) + x(n\text{-}3) + \ldots + x(n\text{-}8) + x(n\text{-}9)\ ]$$

It should be obvious that this is an averaging process. x(n-1) is the sample that occurred just before x(n). Similarly, x(n-2) is the sample value that occurred two sample periods before x(n). So the formula shows that the output is simply formed by adding the present input sample, x(n), to its nine preceding sample's value, then dividing by ten. Let's use the summation version of this equation.

$$y(n) = \frac{1}{10} \sum_{k=0}^{9} x(n-k)$$

Let's take a not-so-obvious mathematical turn. Instead of adding the samples x(n-k)s and then dividing by ten, let's bring the 1/10 inside the summation sign. So we multiply all of the samples by 1/10 first. This doesn't change a thing, we still have the same equation.

$$y(n) = \frac{1}{10} x(n) + \frac{1}{10} x(n-1) + \frac{1}{10} x(n-2) + \frac{1}{10} x(n-3) + \ldots + \frac{1}{10} x(n-9)$$

In fact, let's give the 1/10's a variable name. How about calling it "h?" Better yet, let's have ten h's all with the value of 1/10. Again, using the summation method of writing the exact same thing:

$$y(n) = \sum_{k=0}^{9} h(k) * x(n-k) \textit{ where } h(k) = \frac{1}{10} \text{ for } k=0,\ 1,\ 2,\ \ldots\ 9$$

Nothing has changed by rewriting. However, now we have the discrete version of something with which we are very familiar. Remember the convolution integral from the continuous time and filtering discussion? We will rewrite it with a generalized version of our new formula.

$$y(t) = \int_{-\infty}^{\infty} h(\tau) * x(t-\tau)\, d\tau$$

$$y(n) = \sum_{k=0}^{\infty} h(k) * x(n-k)$$

(For a particular pattern we have $h(k) = \frac{1}{10}$ for k=0, 1, ... 9 and zero everywhere else.)

Notice how strong the parallel is between the convolution integral and the summation formula above. In both cases, we have the input signal x and its past values, multiplied by a function h. We then take the sum. Remember that the integral is nothing more than an infinite sum. Indeed, the summation formula is precisely the discrete version of the convolution integral. It is known as the **convolution sum**.

Convolution Sum And The Moving Average

How far can we push the parallel between the convolution integral, continuous time filters, convolution sum, and discrete time filters? Let's play some games with this moving average.

For the summation formula representation of the moving average, we chose $h(k)$ = 1/10 for k = 0, 1, 2, ...9. If this is truly like the convolution formula we saw earlier, we would expect that $h(k)$ represents the impulse response of the filter. That's what is represented for the continuous time filters. Let's check that out. The impulse input means:

x(n) = 1 for n=0
0 else

Notice that this implies,

x(n-k) = 1 for n=k (since if n=k, then n-k=0)
0 else

So our output y(n) is,

y(n) = average of ten x(n)'s
= [1/10 x(n) + 1/10 x(n-1) + 1/10 x(n-2) + 1/10 x(n-3) + ... + 1/10 x(n-9)]

So we can determine $y(n)$ for each n from our input $x(n)$.

If *n<0*, then all the n-1, n-2, n-3, ... indices are strictly less than zero. This means all *x(n-k)*'s are zero.

For n=0, only x(n) is not zero,
y(0) = [1/10 * 1 + 1/10 * 0 + 1/10 * 0 + 1/10 * 0 + ... + 1/10 * 0]

For n = 1, only x(n-1) is not zero,
y(1) = [1/10 * 0 + 1/10 * 1 + 1/10 * 0 + 1/10 * 0 + ... + 1/10 * 0]

For n = 2, only x(n-2) is not zero,
y(2) = [1/10 * 0 + 1/10 * 0 + 1/10 * 1 + 1/10 * 0 + ... + 1/10 * 0]

and so on through n = 9.

If n = 9, then all the n-1, n-2, n-3, ..., n-9, are now strictly greater than zero, thus all other x(n-k)'s are again zero.

Basically, we input an impulse and got ten

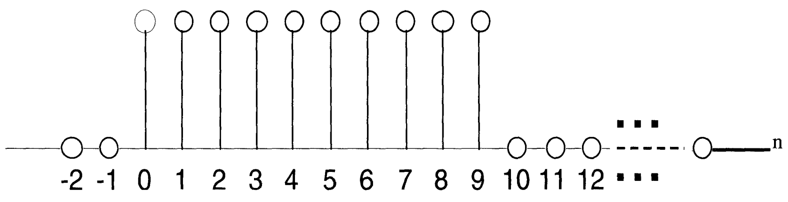


Figure 6-4 Impulse Response of y(n)

values. The actual impulse response is ten values, each equal to 1/10. These output values are exactly described by the terms of *h(i)*. Indeed, *h(i)* is the impulse response. So what about frequency responses?

Frequency Response Of The Moving Average Operator

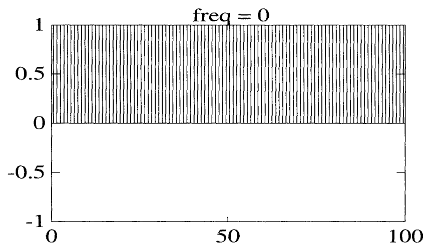
We saw from continuous time filters that if we have the impulse response of a filter (*h(t)*), then the frequency response (H(f)) of the filter is determined by the Fourier transform of *h(t)*. We are not ready to make bold statements about our moving average operator. It is appropriate to start looking at the moving average operation from a frequency domain perspective.

Averaging is performed to get a better view of longer term trends, in other words, to remove the day-to-day or moment-to-moment fluctuations by filtering out the higher frequency noise.

The moving averaging process is a digital low pass filter. We can use our Fourier transform process to see the frequency response.

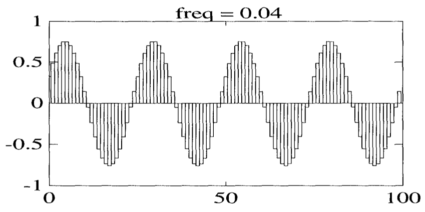
Since averaging is easy to compute, let's take a look at how various frequency inputs are affected by this averaging. For the following figures, the input amplitude remained constant with a value of one; only the input frequencies were varied. All that has been done here is the computation of the moving average (averaging ten consecutive samples) of the input sinusoid for the various frequencies.

We can take a shortcut and remember that the average of a constant value is merely the constant value. So it's no surprise to see that we get the same amplitude on the output as we had for the input.



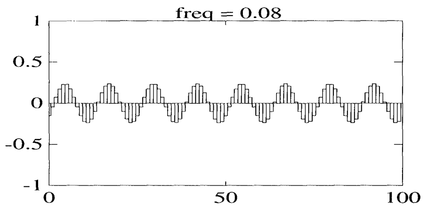
f = 0 We could take a shortcut and remember that the average of a constant value is merely the constant value. Thus, it's no surprise to see that we get the same amplitude on the the

Figure 6-5



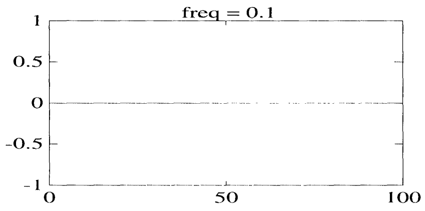
f = 0.04 Here, it becomes noticeable that the amplitude of the output waveform has been reduced by the averaging process. Estimating the amplitude, we will

Figure 6-6



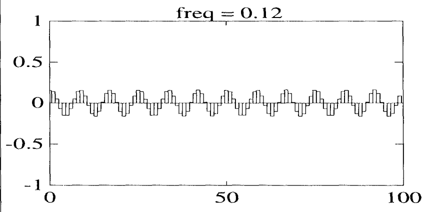
f = 0.08 Once again the output amplitude is reduced.

Figure 6-7



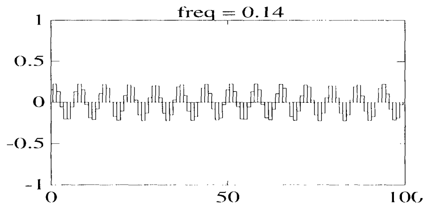
f = 0.1 The amplitude is exactly zero.

Figure 6-8



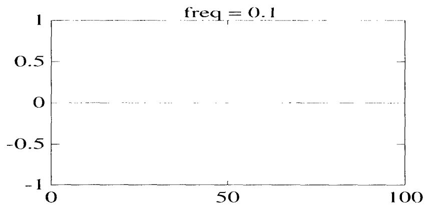
f = 0.12 The output has returned from the zero value. Once again, we have a visible output waveform. The amplitude

Figure 6-9



f = 0.14 Output continues to increase in amplitude.

Figure 6-10



f = 0.2 Zero output again.

Figure 6-11

We could continue, but let's assume we have actually run a set of sampled sine waves of various frequencies through our moving average operator. We chose to let sampled sinusoids have frequencies from zero to half the sampling frequency our limit given by the Nyquist criterion. We can now present a plot of the resulting amplitudes for the various frequency inputs. We now have a rough plot of the frequency response of the moving average operator.

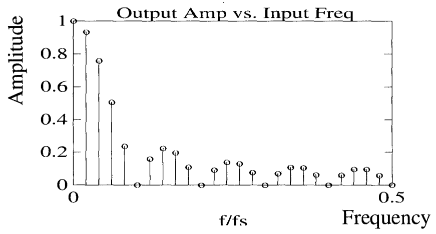


Figure 6-12
Frequency Response of a Moving Average Operator

Since we can discuss the frequency response of this moving average operator, we will throw out the word *operator* and call it *a moving average filter.*

The Moving Average Filter

Let's reflect for a moment on this moving average filter. In fact, since we are interested in exactly how the output waveforms are produced, let's zoom in on ten samples and overlay a plot of the *h(k)* values on the plot of input samples *x(n).*

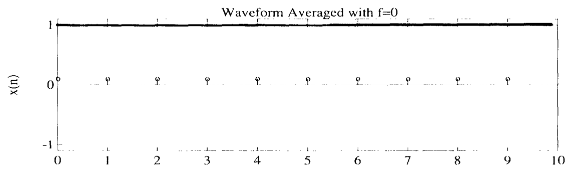


Figure 6-13

This type of diagram comes in handy for understanding many types of filters and gives a great intuitive feel for exactly how the filtering process works. Please become familiar with it! Notice that it shows the actual convolution process. The figure below shows values of x(n) for an input frequency of 0. We can use this graphic for other input frequencies to verify the frequency dependence of the averaging process.

Since an interesting aspect of the moving average filter's frequency response is the value of zero it takes on at 0.1fs, let's re-draw the diagram using an input frequency of 0.1fs.

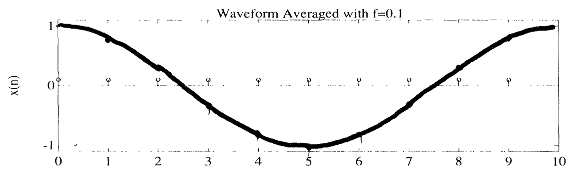


Figure 6-14

It is now possible to see how the averaging process produces a zero output. There are waveform samples that are positive, and five samples that are negative. In fact, the pairs of fives are identical except for being opposite in sign. Thus, when all ten samples are added together, a value of zero is produced. In a like fashion, the other frequencies that lead to zero amplitude exhibit a similar kind of symmetry. The sum of the positive samples in the averaging window is equal and opposite in sign to the sum of the negative samples in the averaging window. Thus, the positive samples cancel the negative samples and the sum is zero. For this perfect cancellation to take place, the input waveform must conform to the moving average window.

Perhaps the easiest case to see why multiples of 0.1fs are zero, is the case of f=0.5fs. Here, we recognize that the sample values alternate between +1 and -1. Therefore, the moving average is always adding five samples of +1 with five values of -1. The sum equals 0. Thus, the moving average always equals zero.

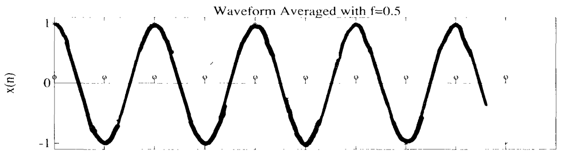


Figure 6-15

Construction vs. Destruction

The averaging process is a low pass process and part of the averaging requires the summation of neighboring samples. In a low frequency input, neighboring samples have the same sign. Each of these samples is multiplied by an h(n) value. Since these are all the same sign, the summation of these values is always constructive in nature.

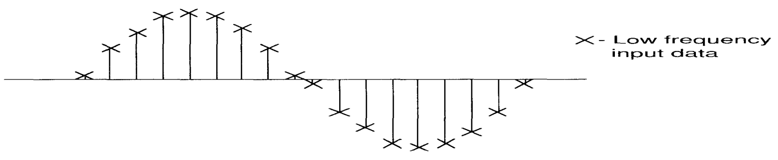


Figure 6-16

Conversely, a high frequency signal has samples of both positive and negative signs over the same size neighborhood. Multiplying by the constant h(n) values will have no effect on the sign of these samples. Therefore, we are left with both positive valued and negative valued samples. As such, their summation is destructive in nature.

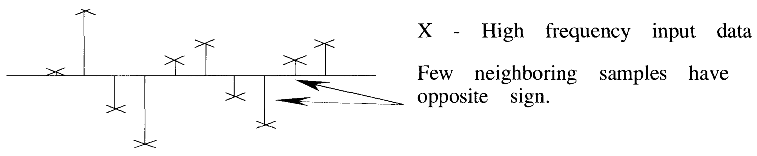

X - High frequency input data
Few neighboring samples have opposite sign.

Figure 6-17

It is exactly the **frequency dependent** nature of the constructive or destructive properties of the summation calculation that is exploited to yield a filtering process.

Conjecture Relating Impulse Response To Frequency Response

In the previous section, the *h(n)*'s were all of the same sign. This acted to produce a low pass filter. What happens if we were to change the h(n) values to something that alternated signs? Can we conjecture the following?

Low Frequency Signal Same Signed Neighbors	Impulse Response ====> Constructive Addition Of Alternating Sign
High Frequency Signal Alternating Signed Neighbors	Impulse Response ====> Destructive Addition Of Alternating Sign

Conjecture Investigated

This conjecture certainly has an attractive, symmetric quality to it. To investigate, use two standard waveforms, a DC signal (constant) and one with a frequency of fs/2. These are chosen simply because they are so easy to work with and still provide useful information.

What happens if the impulse response alternates in sign? Let's take the previous impulse response and change the sign of *h(n)* when n is an odd number. Thus, we have h(n) when n is an odd number.

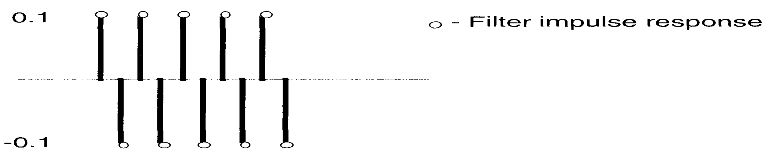


Figure 6-18
Impulse Response that Alternates in Sign

We can easily see that a DC waveform undergoes a destructive summation effect in this case, due to the alternating sign of the *h(n)*. What about the case of fs/2? Let's look again using the signal superimposed on the impulse response diagram.

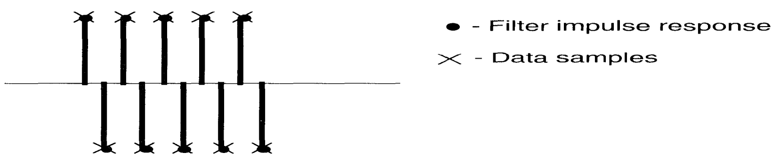


Figure 6-19
h(n) and x(n-k) lined up so each pair is either positive or negative

Here we have the *h(n)*'s and *x(n-k)*'s "lining up" so that each pair is either both positive or both negative. Thus, the result of the multiplication step is always positive, and the sum is constructive. The output sample has a positive value. Its appreciable magnitude conveys a high pass filter. That may be correct for figure 6-19, but look at what happens for the next output sample.

For the next sample, we can view the input as delayed, or moved to the right, within the filter window. The last sample from the previous diagram has exited to the right, while the new sample has appeared to the left.

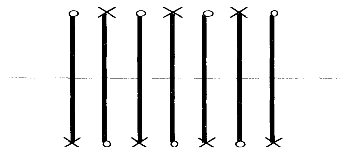

o - Filter impulse response
✕ - Data samples (1 sample later than previous plot)

Figure 6-20
What occurs when h(n) and x(n-k) are lined up with opposite signs.

In cases like this, we always have positive samples lined up with negative h(n)'s or negative samples lined up with positive h(n)'s. So now the result of each of the multiplications is negative. The sum of a collection of negative numbers is negative. We still have constructive addition and an appreciable magnitude on the output sample. It just has a negative sign, that's all.

There is absolutely no problem that the output signal alternates signs every other sample. This is exactly what the input signal did.

We have seen how this alternating sign, *h(n)*, can indeed attenuate low frequency inputs while passing high frequencies. Our experiment, based upon rudimentary examples, demonstrates the underlying principles involved with digital filtering. However, few filters are simple *moving average* or *alternating sign filters*.

The moving average filter discussion was used to introduce the topic of digital filtering. Our everyday idea of averaging was expanded to a moving average. This lead to the notion of a convolution summation. This convolution sum was shown to possess a frequency-dependent character which certainly appears suitable for a filtering operation. Moreover, it has become apparent that different choices for the impulse response *(h(n)'s)* can greatly effect the frequency characteristics of the convolution sum. To design a digital filter of a desired frequency response, you must determine the corresponding impulse response.

Determining The Right Impulse Response

The moving average filter had a given impulse response and determined frequency response as shown in figure 6-12. If we were to make a continuous time analogy to the *h(n)*, by connecting the dots to get *h(t)*, we would have a *continuous time version* of the impulse response. Since this is now a continuous time impulse response, the dots have become connected. We can apply the Fourier transform to determine the frequency response.

In our familiarity with the continuous time Fourier transform, we realize that the filter's needed impulse response can be found simply by taking the inverse Fourier transform of the desired frequency response.

In light of the similarity between discrete and continuous convolutions we can determine the digital filter's impulse response. In other words, take a desired continuous frequency response, call it *H(f)* and compute its impulse response, *h(t)* by the inverse Fourier transform. We merely take a *sample* value of *h(t)*. These samples are taken every sample period corresponding to the rate the digital system is intended to use. We call these samples *h(n)*, and use *h(n)* to implement the digital filter. This is known as the **impulse invariant filter** design.

Implementing Response

There are a variety of ways to determine a proper filter given the desired frequency response. We will look at other methods of impulse response determination later.

We presently need to address another important aspect of digital filtering; **implementation**. It is correct to do so at the present time since the eventual implementation will invariably impact the filter determination as well.

Finite Impulse Response Filters

The **finite impulse response filter** (FIR) is the first filter type we will view. Actually, the FIR filter is nothing more than a generalization of the moving average filter we discussed. The difference is that we no longer require the *h(n)*'s to have the same value.

The limits of the summation go from zero to N-1. This helps to denote the that the impulse response is finite. Since there are N terms in this summation, N is known as the order of the filter. This convolution sum becomes more penetrable in its block diagram form. The block diagram is a direct implementation of the actual convolution sum above. For this reason, the block diagram, shown in figure 6-21, is known as the **direct form** or **canonical** form of the FIR.

Z^{-1} has the same meaning as Delay and is discussed on page 6-20.

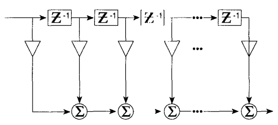


Figure 6-21
The Direct Form or Canonical Form of FIR Filter

We can see that each delayed sample is put through a gain (multiplied by *h(n)*). The output consists of the summation of N of these delayed and gained samples. The direct form implementation is closely related to analog tap delay lines used in communications engineering and has borrowed much of its nomenclature. The string of digital delays along the top is known as a **delay line**. The points available within this delay line are called **taps**. The values the samples are multiplied by are the **tap gains**, **tap coefficients**, and **tap weights**.

The FIR filter structure is a very standard and common filter structure. The most notable being:

1) They are inherently stable (as opposed to the next filter type to be presented).

2) The dynamic range of the states of the filter is easily computed. This is important to prevent clipping of the signal.

3) It can easily be constructed to have the desirable property of linear phase.

4) FIR tap gains for a given desired frequency response and can be easily computed by Fourier transform techniques.

Let's look at another standard digital filter implementation.

Infinite Impulse Response Filters

We do not need an infinite number of taps to construct an **infinite impulse response** (IIR). The reason for this is that the IIR does not use the same structure as the FIR. The concept of feedback is used for the IIR. Here, the idea is to re-circulate some of the input signal. It should be obvious what is meant by looking at the IIR block diagram shown in figure 6-23. To simplify things, we drew an elementary, IIR filter.

Z^{-1} has the same meaning as Delay and is discussed on page 6-20.

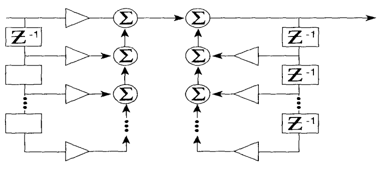


Figure 6-23 IIR Direct Form

To realize the infinite nature of the filter's impulse response, look at figure 6-23 using a feedback gain of 0.9 for the filter. For the first output of the impulse response we have the same value as the impulse. The next output value is the sum of the input plus 0.9 times the previous output. However, all following input values are zero, so we keep multiplying the present output by 0.9 to get the next output value. The frequency response of this IIR filter can be computed in the same manner as was done for the moving average filter in figure 6-2.

The Delay

In both the FIR and IIR filter descriptions, we made casual use of an element labeled *delay* or *sample delay.* Whatever sample value you enter eventually comes out. Often we talk of the **unit sample delay**, which means the time between input and output is exactly one sample period of the digital system.

This sample delay appeared in both of the block diagrams and in the convolution summation. The only reason we bring the delay up again is to introduce an alternate description. This new description gives us a more abstract and powerful representation. It provides us with a direct link between the **digital impulse response** and the **digital frequency response**. In other words, we won't have to use continuous Fourier Transforms and sampling as a bridge to determine the digital filter.

The z^{-1} Description Of Delay

Although the actual reasoning behind the use of the z^{-1} to represent a delay is not completely intractable, it is one of those discussions that is beyond our scope. Let's agree to use it as an alternative and identical description of the sample delay. Using this notation, instead of writing the sample delay of x(n) as x(n-1), we will now write the delayed version as $^{z^-}1$x(n).

Although we have glossed over the how's and whys of this notation, we can still use it as an effective device (albeit a blind rule) to gain insight into DSP. This shouldn't bother us too much since the idea of following blind recipes is exactly what quantum physicists have been doing for years. To acquaint ourselves with the use of this new notation, let's rewrite a familiar equation, the convolution sum of the moving average filter, using this z^{-1} form.

The Frequency Response Of Discrete Time Signals And Systems

The previous pages have begun to discuss digital filtering. As filters are, for the most part, thought of as effecting the frequency components of signals, we will need to present some further ideas on what frequency really means in terms of digital (or discrete) signals.

Up to this point, we have allowed to ourselves to think of a discrete sequence having certain frequency components by imagining that we are merely talking about the sampled version of some continuous waveform with those frequency components. This has an intuitive appeal. A true mathematical expression for the frequency description of the actual discrete time signals was deliberately delayed until now. We now seek to rid ourselves of this excess baggage by re-examining the Fourier transform. Consider the continuous time signal *x(t)* and its frequency description *X(f)* given by the Fourier transform of *x(t)*:

$$X(f) = F[x(t)] = \int_{-\infty}^{\infty} x(t)\, e^{-j2\pi ft}\, dt$$

Now, consider the sampled version of x(t) obtained by sampling x(t) every T_S seconds, call this $x_S(n)$. Here $n=kT_S$ where k takes integer values. This is just the same sampling process we've used all along.

$$x_S(n) = x_S(kT_S) = x(t)\ \delta\ (t-kT_S)$$

We are still at liberty to take the Fourier transform of this, so let's do just that:

$$X(f) = F[x_{S(n)}] = F[x(t) * \delta\ (t-kT_S)] = \int_{-\infty}^{\infty} x(t) * \delta\ (t-kT_S)\ e^{-j2\pi ft}\, dt$$

Since $\delta(t-kT_S)$ is zero everywhere except for when $t-kT_S$ is an integer. For simplicity let's call this integer *n*. We need to consider only the values in the integral. This reduces the integral merely to the summation:

$$X(f) = \sum_{n=-\infty}^{\infty} x(kT_S)\ e^{-j2\pi fkT_s} = \sum_{n=-\infty}^{\infty} x(n)\ e^{-j2\pi fn}$$

This expression gives us the frequency components of the discrete, or digital, signal directly from the samples of the signal *x(n)*. We no longer need to imagine a link to the continuous signal; the frequency components are strictly determined by the samples. This is the Fourier transform of the sequence *x(n)* now given by an infinite sum.

Often in DSP, engineers talk not of the Fourier transform, but of the "Z-transform." Indeed, the Z-transform is prevalent and will be presented and defined next. Although the reasons behind the use of the Z-transform and Z notation are not completely intractable, it is just one of those discussions that is beyond the scope of this book.

CHAPTER 7

Digital Filter Design

How To Get The Filter You Desire

To construct this digital filter there are a variety of well-defined options available once the desired frequency response is decided upon. To a large extent, the filter design methods used will depend on the filter parameters that have been specified for the desired response.

For instance, we've seen from our discussion on continuous time filters that the general parameters needed to specify an analog filter (and digital) consist of:

- Pass band Edge(s)
- Stopband Edge(s)
- Stopband Attenuation

A bit simplified. For the analog filter, we also have to decide the order of the filter, that is how many inductors and capacitors will be required. This order determination is required for our digital filter as well. For the digital implementation, the filter order may be thought of as the number of "delay" operators (or the highest power of z in the H(z) polynomial --- it's the same thing.) So, we'll add that to our list:

- Filter Order

Notice that the first three quantities are generally readily apparent to us based on our filtering needs. Whereas, the value of our filter order is dependent on our first three constraints, as well as our chosen filter design technique and filter implementation. Generally, from a cost consideration, we seek to keep the filter order as small as possible while still doing the job.

The problem is the value of our filter order depends on the design method we use. It also depends on the type of filter we choose to build, IIR or FIR.

In general, the first decision to make on the filter implementation is to choose either an IIR or FIR. Each has its own set of advantages and disadvantages and the two can be quite different. Faced with this, we are required to present some basic facts and generalities on the issue of IIR vs. FIR.

FIR vs. IIR "FACT" SHEET

FIR Filter	**IIR Filter**
#1) - may require high order for relatively modest filtering requirements	+ impressive performance achievable with modest order
#2) + always stable	-(?) stability issues must be considered
#3) + simple limits to state dynamic rang	- internal states may acquire very high dynamic range
#4)	- sensitivity to coefficient quantization
#5)	- susceptibility limit cycles
#6) +(?) linear phase response	-(?) non-linear phase response

Here are a few points.

#1) The first point is short and sweet. When it comes to filtering performance, as measured in great stopband attenuation, sharp transition regions, the IIR generally can out perform the FIR, for any given filter order. The performance difference (or equivalently the order difference) can be dramatic. However, the advantage of IIR's in this area may be offset by the other factors. Obviously, the continued popularity of the FIR filter demonstrates that filtering efficiency is not the only metric used to decide between IIR or FIR.

#2) Unstable filters (or systems) are to be avoided. Instability *may* arise only from a feedback mechanism. This feedback is inherent in IIR designs. The stability liability of the IIR may be a bit misleading at first glance. To be sure, we don't want an unstable filter. With proper design, all filter poles can be made to fall inside the unit circle. Thus, proper stability will be guaranteed. Conversely, FIR filters have no feedback, and are thus inherently stable. The FIR transfer function has no poles, consisting strictly of zeros.

#3) The state of a filter is just another way of saying a filter memory location (or register) used to hold the delayed samples. These states have a dynamic range, just as we say a signal possesses a dynamic range. We've already seen that the dynamic range of a signal is limited by the number of bits used in this signal's representation. Let's examine the state's of both the FIR and IIR filter, keeping the dynamic range in mind. We'll use a canonical form of the FIR and IIR block diagrams to make our point here.

Note: Quotation marks around the word 'facts' is deliberate; this table is formed by generalities. The +'s and -'s above are meant to give some indication of the desirability, or , of the listed filter qualities.

FIR:

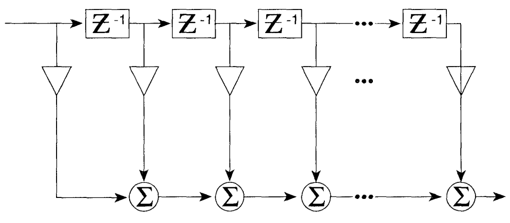


Figure 7-1 The Direct Form Realization of FIR Filter

This case is straight forward. Here, the dynamic range of the states is exactly the same as the dynamic range of the signal. (This is seen by realizing that the signal itself passes through the delay line.) The same number of bits may be used in the filter state storage as is used in the signal representation. In fact, the only point of concern here may be at the summer.

Since we could be adding a potentially large number of delayed samples (obviously this number depends on the filter order). The concern for possible overflow at summer stage can be mitigated in many situations by appropriate scaling of the tap gain values.

IIR:

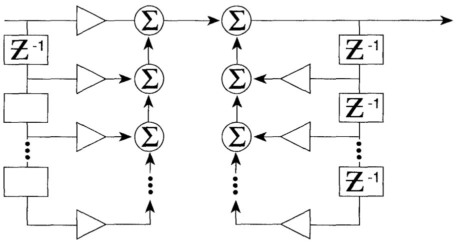


Figure 7-2 The IIR Direct Form

In contrast to the FIR filter, the IIR filter does not have only the original input signal flowing through its filter states. Because of the feedback mechanism of the IIR filter, many delayed and superimposed versions of the input signal circulate through the delay line. If this superimposition is constructive in nature the amplitudes of the filter states (dynamic range) may greatly exceed that of the input signal. This may allow for the filter states' dynamic range to be *much* larger than the original input signal. For such cases, the same number of bits may *not* be used in the filter state storage as is used in the signal representation. Poor results, such as clipping, may result. Care must be taken not only for the summer nodes but also the filter state.

#4) Sensitivity to Coefficient Quantization. In the following sections we'll discuss how to determine the proper coefficients for a desired frequency response. For now let's imagine we have determined the proper coefficients by one of these methods. Frequently, these

coefficient values will be determined through the use of a computer program (with rather high numerical precision - perhaps 32-bit floating point). Since our filter will require real-time operation (fast) at reasonable cost (cheap), the filter will probably be implemented on a machine (programmable DSP microprocessor, or fixed hardware state machine) with lesser numerical accuracy. In this case, we cannot implement the “true 32 bit floating point” desired coefficients, but are forced to accept something reasonably close. The term used for, not surprisingly, known as **coefficient quantization**. Typical quantization values for fixed point coefficients are 12, 14, 16, or 24 bits. In making this transition from high accuracy design to limited accuracy implementation we must take care. Some implementation forms are more sensitive to coefficient quantization than others. To be a bit simplistic, we’ll say here that FIR structure implementations are generally less susceptible to this quantization than IIR filters. Furthermore, among IIR implementations there are some forms more sensitive than others.

#5) Limit Cycles: Limit Cycles are a phenomena strictly limited to the IIR filters since they are not possible in the FIR implementations. This again is due to the feedback nature of the IIR. As we’ve seen, we fully expect a stable filter to eventually reach a zero output condition for the zero input case. [Think of the example where the impulse response was (1 -0.9 $(-0.9)^2$ $(-0.9)^3$ ...), here the output asymptomatically approaches zero.] Note that this has been discussed in reference to infinite precision processes, where we make no quantizations on either the coefficient (0.9) nor the state of the filter. The point here is that re-examining this with respect to quantized coefficients and quantized filter states we get an interesting result. This may be best shown by an example.

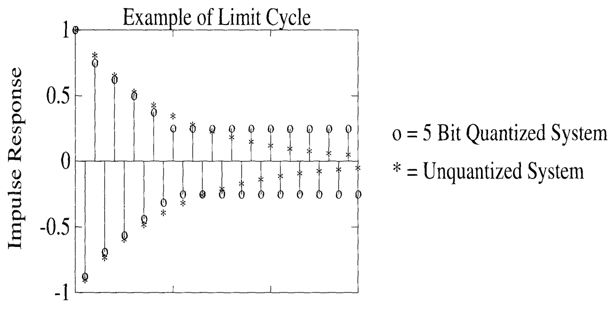


Figure 7 - 3 The Limit Cycle

In figure 7-3, we have resurrected a first order (at z=-0.9) IIR filter. However, now the state and coefficient quantization have been included. For illustrative purposes, we have chosen a 5-bit quantization (see other figure). The impulse response has been plotted for the first 25 output values. Notice how the quantized IIR's output "gets stuck" in a fixed value oscillation, while the unquantized version continues on its journey towards zero. This occurs because in the quantized system the result of the multiplication of the (quantized) state by the (quantized) alpha yields the same magnitude (after quantization) as the previous state. This is a simple example of a limit cycle. The point to note is that this limit cycle has produced an oscillation. Its behavior has been greatly magnified for our example here by using a severely limited quantization scheme of only five bits. However, even for more realistic quantizations limit cycles may still persist. These limit cycles may become more complex as the IIR structure increases in order.

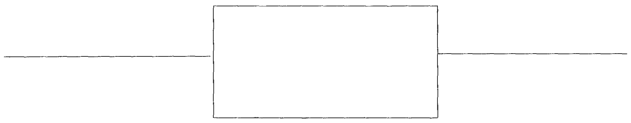
#6) In the preceding chart, the "Linear Phase vs. Non-linear Phase Response" of FIR versus IIR filters, a question mark was placed next to the entry. It's presence was meant to make the reader reflect on whether linear phase might be a good, bad, or even a "don't care" characteristic. The answer to this depends on the application. However, it might be said that linear phase response is given more credit than it deserves as a desirable filter quality. To be sure, some applications require linear phase. However, if you're not explicitly aware of phase properties of your signals, chances are you don't need a linear phase filter. First, some background:

- We haven't paid too much attention to the concept of phase since it was first discussed in the continuous-time section. Reviewing the basic points will remind us that is phase is always relative measure between two sinusoids. In a filter, this relative measurement is taken between the sinusoidal input and the sinusoidal output of a filter. The "Phase Response" of the filter is result of the collection of all such relative measurements for all possible input frequencies. As such, a filter's phase response defines a relative phase (or phase delay) as a function of frequency.

- In our continuous-time filter discussion, we concentrated on the magnitude response of ideal and non-ideal filters. The reason for this concentration (beyond the fact that magnitude response is a slightly simpler concept that phase response) is that most often it is exactly the magnitude response that is desired to be affected. Below is an example.

Example: You want the bass guitar on your stereo to be *louder*, so you turn up the bass knob (which increases the *magnitude* response of low frequencies of your stereo.) Nobody even asks about the phase response considerations involved in turning this knob.

- What does linear phase mean? Linear phase means that the phase delay between filter input and output is a linear function with respect to frequency. For example, the phase delay at 200 Hz will be some constant times the phase delay at 100 Hz. The actual meaning of linear phase becomes easier to see with a small excursion, looking at a few equations with our linear phase filter. Consider a simple, sinusoidal input:



input:		output:
$\cos(\omega \cdot t)$	----->	$A \cdot \cos(\omega \cdot t + \varphi)$

Here, the corresponding output is the expected sinusoid of the same frequency with a different amplitude and phase. We're interested only in the phase for this discussion. But, we actually know more than this. From the fact that this is a linear phase filter we know that φ is a linear function of frequency (ω), writing this out:

$$\varphi = -\alpha \cdot \omega$$

now, rewriting the above equations:

$$\cos(\cdot t) \quad \text{----->} \quad A\cos(3\omega \cdot t - \alpha \cdot 3 \cdot \omega)$$

$$\text{----->} \quad A\cos(\omega \cdot (t - \alpha))$$

Remember that for any function f(t), the function $f(t-t_d)$ is the same function delayed by some time t_d. The result is really no surprise here: We just get a delayed and amplitude scaled version of our original sinusoid as output. What happens if the input was of a different frequency? Let's see, using $3\cdot\omega$ instead of ω:

$$\cos(3\omega t) \text{ -----> } A\cos(3\omega t+\varphi)$$

but $\varphi=\alpha 3\omega$ (by linear phase filter),

$$\cos(3\omega\cdot t) \text{ ----> } A\cdot\cos(3\omega t-\alpha 3\omega)$$

$$\text{ -----> } A\cos(3\omega(t-\alpha))$$

We get the same time delay as the case before. For two different frequencies (ω and 3ω), we get the same time delay (α). In fact, for *any* input frequency, we'll get the same time delay between input and output, as long as the linear phase constraint of $\varphi = \alpha\ \omega$ is met! It may make more sense to call a linear phase filter a constant time delay filter since *all* spectral components are delayed by the same amount of time.

Okay, so with the above points in mind, we may consider the linear phase argument with respect to FIR and IIR filtering. It turns out that the linear phase, or constant time delay property can be easily met with an FIR filter implementation. This linear phase condition can be guaranteed by constraining the filter's impulse to be symmetric about the center tap. For instance, assuming an N tap filter where N is odd for our picture, we have:

- (N/2)-1 = (N/2)+1
- (N/2)-2 = (N/2)+2 picture of symmetric h(n)
- (N/2)-3 = (N/2)+3 etc.

The FIR can easily be made to have linear phase response. The same does not hold for IIR filters. One quick way to see this is to consider the symmetry argument of impulse response that guarantees linear phase. For the IIR we'd seek to answer the question, "How does one build a symmetric version of something with an infinite tail?." (Don't forget the first "I" in IIR stands for "infinite") Obviously, we'll require an infinite duration 'head' before this infinite tail. But, this would give us problems with respect to causality. So, the goal of finding a causal, linear phase IIR filter seems elusive.

Keep in mind that continuous-time (analog) filters have never been linear phase. And yet, in spite of their lack of linear phase, they have been useful tools for a variety of functions over the years. So, be careful not to relegate non-linear phase filters to the junk pile of obsolescence, it just isn't so.

Should You Choose An FIR Or An IIR Filter?

Wouldn't it be great if we could simply "always use an FIR filter? The truth is that there is no one, global answer. Each filter design must be considered within the framework of its implementation. The list presented some of the general tradeoffs involved in this selection.

Typically, the engineer may perform a rough filter design using estimates of the desired filter response for both an FIR implementation and an IIR implementation. The results of this initial design gives an estimate of the necessary filter orders. (And correspondingly, the number of numerical operations per second required for such a filter.) Sometimes the limited horsepower of DSP machinery used in the filter's implementation will dictate the use of the IIR filter at this point. However, in general, the selection may have to be based on other issues mentioned. This design process typically is an iterative process where the engineer unravels the desired filters implementation through a series of small investigations. Luckily, as we'll soon see, much of the drudgery of this series of small investigations is mitigated by the use of computer programs. A great variety of DSP filter design computer packages makes this job easier. This by no means that an exhaustive computer search should be used to find the "best" filter implementation!

"FIR vs. IIR", some readers may believe there is a prejudice against FIR filters. This is definitely not so, although, we have deliberately played down the linear phase advantage of the FIR over the IIR. This is because the "linear phase advantage" is truly an advantage only in those applications where linear phase is important. For a large number of applications this may not be the case. Thus, if one were to be limited to the choice of an FIR because of its linear phase properties (when it is of no consequence for the application at hand) the engineer has placed an unnecessary constraint on the design problem (and thus not obtained the optimal solution). The result of such a false constraint could have a tremendous impact on the system solution, as IIR and FIR filters of similar performance can be of considerably different order.

GLOSSARY

AC (Alternating Current): Refers to the cyclical nature of electric current when it has a frequency associated with it.

ACCOUNTANT-ESE: Terms which may be used by an accountant.

ADC: Analog to digital converter.

ALL-PASS FILTER: A filter that allows all frequencies to pass through the circuit.

AMPLITUDE: Amplitude refers to the intensity of an electrical signal in terms of its voltage (or potential), current or power level at any given point in time or frequency.

AMPLITUDE COMPRESSION: An action to shrink a signal's amplitude. Amplitude compression is performed on music signals before they are sent across the airwaves, for example.

AMPLITUDE 1 (OR, AMPLITUDE OF ONE): When this phrase is used, the author is referring to the maximum intensity of unit 1 that a signal reaches. It may refer, for example to voltage, current , or, power signals. In the case of a signal which repeats itself periodically, such as a sine wave, this phrase refers to the intensity that a signal may reach on its upward or downward swing.

ANALOG: This concept refers to an unbroken stream of any natural or man-made phenomena, such as a river, or the electrical current flowing through a high power line. In signal processing, the continuous action always refers to the continuous action of an electrical signal or signals in which we have taken interest. The idea of brokenness, or discontinuity, as it is associated to signal processing, is discussed under both "Discrete" and "Digital."

ANALOG DOMAIN: This phrase refers to the category under which all continuous actions fall; or to everything in the universe that can be said to be continuous and analog. For example, we can say that the motion of the sun is part of the analog universe. However, in the unlikely event that something breaks the motion of the sun from its normal path, we would cease to think of its motion as continuous and analog in that moment (see "Discrete" and "Digital").

ANALOG SIGNALS: An analog signal refers to something in the analog domain that contains information (see "Signal"). For example, the position of the sun is an analog signal if we view it as something which contains information about the time of day.

ANALOG-TO-DIGITAL-CONVERTER: This device does just as it says; it converts analog waveforms into digital numbers suitable for storage in computer memory.

ARITHMETIC LOGIC UNIT (ALU): The ALU is the portion of the Central Processing Unit (CPU) which interprets and performs arithmetic and logic commands which we issue to the CPU.

ASCII: This acronym stands for American Standard Code for Information Interchange. This is the standard which defines certain digital bit patterns to stand for common symbols such as our alphabet and our number system. The bit pattern, 0011000, for example, stands for the number zero.

ATTENUATION: This is what happens when a signal tapers off. We are attenuating the music on our radio when we turn down the volume.

BANDPASS FILTER: This is a filter which attenuates all frequencies higher than the upper bandedge, and lower than the lower bandedge; while allowing all those frequencies in between to "pass." It is the opposite of a bandstop filter. The ideal filter has an absolute transition to zero, rather than the non-ideal attenuation.

BANDSTOP FILTER: This filter passes frequencies higher than the upper bandedge, and lower than the lower bandedge, while attenuating all those in between the bandedges; this is commonly referred to as a "notch filter." This filter is the opposite of a bandpass Filter. The ideal filter has an absolute transition to zero, rather than the non-ideal attenuation:

BANDWIDTH: A bandwidth is a range of frequencies. For example, the lowest audible frequency is 20 Hertz, while the highest audible frequency is 20000 Hertz; together, these frequencies define the bandwidth of audible frequencies.

BASEBAND: This is an often-used term to refer to the original form of a signal before it was modified by a DSP filtering technique.

BLACKMAN HARRIS WINDOW: (See "Windows") -C- CERAMIC RESONATOR: This is a frequency-producing device whose ceramic material is so strong and responsive that it can be made to resonate, or vibrate, at a very high frequency when an electrical signal is applied (see "Natural Frequency").

BREADBOARDED: To build up the circuit with the intent of trying it to see if it works.

CARRIER FREQUENCY: This can best be understood in terms of the example given in the textbook. We know that the audible range of musical signals is from 20 Hertz to 20000 Hertz. However, it turns out that when FM radio stations want to broadcast these signals, they do so at a considerably higher frequency; somewhere between 88000 Hertz and 108000 Hertz. The musical signals are being 'carried' by another signal; hence the term "Carrier Frequency."

CAUSAL: This concept refers to the idea that any signal which we create, and which we begin to see at the output of our system, was created from previous inputs. That is, the causal system never anticipates what the input might be; but it depends on a new value at the input to create another output value.

CENTRAL PROCESSING UNIT (CPU): This is the "brain of the computer. The CPU analyzes and interprets the commands that we give it, and then it executes those commands in an orderly fashion.

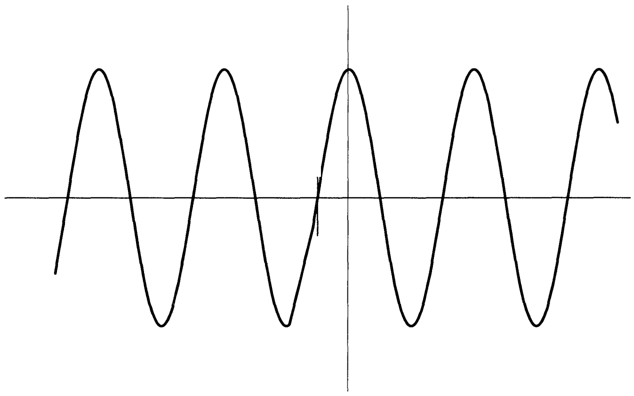
COEFFICIENT: This is also known as a Coefficient Multiplier. This is a number by which each point in a signal is multiplied in order to alter it in predetermined way. More than one coefficient is generally used to accomplish the signal processing objective.

CONVERSION TIME: Length of time to convert a signal from analog to digital.

CONVOLUTION: This is the process whereby 2 or more signals are mixed together to form a new signal.

CONVOLUTION INTEGRAL: This is the mathematical formula, or algorithm, which describes how 2 or more signals are mixed to synthesize a new one. In a more detailed sense, this integral describes how signals, themselves composed of 1 or more sinusoids, may be systematically meshed together for the purpose of creating a new set of finely integrated sinusoids, which together compose one new signal. This is also known as the Superposition Integral.

COSINE FUNCTION: Below we see a graphically description of a cosine function: Figure 1. A cosine function



D.C. (OR, DIRECT CURRENT): Whenever the term "dc," or "level," is used, it refers simply to the constant voltage level of a signal which exists without a frequency component.

DAC: Digital to analog converter.

DELAY: Delay is one of the main strengths of DSP. In an analog system, it is difficult to control how long one event must precede another, faithfully, time after time. A digital system, however, incorporates memory devices which can store a value for just about as long as we want to before using it again. Computers, and their ability to store and synchronize events, are the reason for Digital Signal Processing; and they give us the flexibility to use delay effectively.

DELTA FUNCTION: The delta function is a tool used in many areas of Electrical Engineering to describe a single, infinitely small occurrence of a signal whether it be in the Frequency Domain, or in the Time Domain. It can be viewed as the smallest, simplest element of any signal, out of which all signals are built. The delta function is to the atom, as signals are to matter.

DIGITAL: Digital simply means having a value of "1" or '0", as opposed to analog, which could have any value in between. It often implies computers or computing devices. The word digital is best thought of in connection with the words analog and discrete. A signal being received by a computer is at first analog in nature since it is generally a continuous waveform. The analog signal then passes through a device which converts the analog signal into a set of discrete quantifiable sub-signals. These sub-signals are then put into memory within the computer where they can officially be termed "digital."

DIGITAL SIGNAL PROCESSING, OR, DSP: This is what this book is all about and aims to explain. Briefly, DSP refers to the way in which computers are used to alter signals which have already passed out of the analog domain, into the discrete domain, and finally into the digital domain. Once in the digital domain, the DSP engineer uses the processing power of the computer to alter the properties of the signal before it is passed back out of the computer into the analog world again.

DIGITAL-TO-ANALOG-CONVERTER: A device which converts digital numbers into analog output.

DISCRETE: Discrete implies a broken stream of of a normally-thought -to-be continuous action or thing. For example, old timepieces had second hands that swept across the face of the watch in one continuous, analog motion. The newer watches, we noticed, had second hands that skipped across the face of the watch at regular intervals to count out the seconds. This action would be considered discrete (broken).

DOMAIN: In Digital Signal Processing, the engineer or enthusiast frequently speaks in terms of domains — for example, the frequency domain, or the digital domain. The word domain came from a branch of mathematics called Set Theory; but generally it is used define the boundaries of the set to which we are limiting ourselves.

DRIFT: Change in performance.

DUALITY: Without meaning to sound cosmic about it, this is a concept which refers to the "twoness" in the universe. But more important to DSP, it refers to the reciprocal relationship between the time and the frequency domains which makes possible the the very powerful tool of the Fourier Transform Pairs.

DUMPING: To get rid of; stop using.

DYNAMIC RANGE: This refers to the frequency range of a signal. Since most signals are composed of many sub- signals, it is helpful to speak of its dynamic range, or the range of frequencies of which it is composed.

EEPROM: Electrically, erasable programmable memory.

EPROM: Erasable programmable memory.

EXECUTION UNIT: This part of the Central Processing Unit (CPU) coordinates and synchronizes all actions which the Arithmetic Logic Unit (ALU) performs.

FEEDBACK: In signal processing. quite often the output of a system is re-used in order to help make it more stable, or simply to produce a desired effect; this is known as feedback. FILTER (OR, SIGNAL FILTER): In the broadest sense, a filter is anything that modifies whatever it is that we are trying to modify. For example, a singer modifies, or filters, sounds in order to make the tones pleasing to the ear. We can see that filtering is actually a point of view. However, in the DSP sense, filters are ways that we can change an incoming signal, such as the signal being received from a telephone line, or the music we hear from our CD player.

FLASH CONVERTER: This is a high speed Analog-to-Digital Converter (See "Successive Approximation Converter").

FOURIER TRANSFORM: The Fourier Transform is a process based on the mathematics of calculus which is used to move back and forth between the frequency and the time domains in examining the signal.

FOURIER TRANSFORM PAIRS: Table XX given on page YY lays them out for perusal. Such frequent use of these particular bridges, so to speak, between the frequency and the time domains (and back again)is made, that it finally became prudent simply to put one of these tables in just about every DSP reference manual or textbook in existence.

FREQUENCY: Intuitively, we know that this word means how often something happens. In the DSP sense, we mean how quickly a signal finishes one of its cycles (see "Sine Function").

FREQUENCY DOMAIN: This domain gives information about how often a signal completes one of its cycles, and how many major sub-signals there are contained in the signal. A signal can be composed of many individual signals where each one of those signals has its own cycle, or, frequency, associated with it (see "Sine Function," or, "Cosine Function"). So, one can imagine that, upon viewing a complicated signal in the frequency domain, one would see a full range of main lobes, indicating that the signal has many concentrations of single frequencies within it.

FREQUENCY DOMAIN DESCRIPTION: (see "Frequency Domain").

FREQUENCY RESPONSE: Completely defines the way a system responds to frequency input.

FUNDAMENTAL FREQUENCY: This refers to the dominant frequency among a group of related frequencies (See "Harmonics").

GAUSSIAN WINDOW: (See "Windows").

GIBB'S PHENOMENON: These are oscillations in the time or the frequency domain which occur at any sharp transition point in a signal. The famous mathematician, Josiah Gibbs, proved that when we are attempting to create a square wave (which is a wholly owned subsidiary of the human mind —they do not occur in nature) or, whenever we are trying to create a signal in the frequency or the time domain which exhibits sharp transitions, small oscillations start to occur. These oscillations are also known as ringing (See "Ringing").

GROUP DELAY: In the DSP sense, this takes each increment of phase change, and ratios it with the amount of frequency change during the same increment. When the signal is demonstrating Gibb's Phenomenon in the frequency domain, for example, the ratio is not constant; but for as long as the amplitude was flat, the ratio was a constant.

H- H-BRIDGE MOTOR DRIVER: This is a microcontroller peripheral which is a configuration of transistors used to control motors. Yes, the circuit does look like an "H."

HAMMING WINDOW: (See "Windows").

HANG: Connect; attach to.

HANNING WINDOW: (See "Windows").

HARMONICS: When we pluck a guitar string we produce a sound which is composed of many different sub- signals each of which has its own individual frequency. Each one of these frequencies is associated with an integer multiple of the main, or fundamental, frequency that the plucked string made. Harmonics, then, are the related sub-frequencies that a signal may produce. In a frequency domain plot, we would see the fundamental frequency as a main lobe, and each of its harmonics would be the side lobes, which taper away from it on either side.

HERTZ: The unit used to measure electrical signal frequency.

HEURISTIC: Refers to the exploratory method of engineering sciences. But it is really just a fancy name for "rules of thumb." Heuristic notions are also common sense notions, or notions about how to do things based on the experiences of life; for example, we learn not to touch a live wire by either being zapped, or being warned before it can happen. This is known as negative feedback, and it is the most important part of the heuristic method.

HIGHPASS FILTER: This is a software or hardware device which will block the low frequency signals of our choice, and allow us to pass those high frequency signals which interest us. The ideal filter has an absolute, sharp transition to zero, rather than the non-ideal attenuation.

IDEAL FILTER: The ideal filter blocks every frequency component of an incoming signal above, and/or, below, the specified bandedges of the filter. If it were a bandpass filter, it would look like a this:

Needless to say, the ideal does not occur naturally; but it is a convenient form to turn to since it is easier to draw when we are trying to explain something.

IDEAL BANDPASS FILTER: (See “Ideal Filter,” and “Bandpass Filter").

IDEAL BANDSTOP FILTER: (See “Ideal Filter,” and “Bandstop Filter").

IDEAL LOWPASS FILTER: (See “Ideal Filter,” and “Low Pass Filter").

IIR FILTER: Infinite Impulse Response Filter

IMPULSE: This is a single discrete sample of a signal (See “Delta Function").

IN-CIRCUIT-EMULATOR: This device is generally built into a computer for the purpose of allowing the systems designer to use external equipment to examine, alter, and control the MCU system.

INFINITE IMPULSE RESPONSE FILTER: This kind filter is produced using feedback. Its major drawback is that it has stability problems.

INTEGRATING CONVERTER: A Digital-to-Analog Converter whose specialty is making very accurate representations of the analog signal for use in the digital domain. The drawback is that this kind of converter is slow.

INVERSE FOURIER TRANSFORM: The inverse of the Fourier Transform simply refers to the process whereby a signal which we are viewing in the frequency domain, is transformed back to the time domain. (See “Fourier Transform").

LINEARITY: In the DSP sense, and in almost any other sense, this idea refers to the constant ratio associated with incremental changes in cause and effect. For example, if time changes and voltage changes with it, and the ratio between them remains a constant, then we have a linear relationship.

LINEAR-LOG WEIGHTING: (See “Weighting"). This is a conversion used to emphasize aspects of a signal that come out best when viewed on a logarithmic scale as opposed to a linear scale.

LOCAL MINIMAS: When viewing a graph of a waveform in the frequency, or in the time domain, we may choose to study one particular area of the waveform, ignoring all the rest. Local to that one particular area, the waveform may have a lowest, and a highest point: the local minima is, clearly enough, the lowest point (See “Local Maximas").

LOCAL MAXIMAS: When viewing a graph of a waveform in the frequency, or in the time domain, we may choose to study one particular area of the waveform, ignoring all the rest. Local to that one particular area, the waveform may have a lowest, and a highest point: the local maxima is, clearly enough, the highest point (See "Local Minimas").

LOW-PASS FILTER: This is a device which blocks frequencies higher than a predetermined point in the frequency spectrum; thereby allowing only those frequencies lower than that point to pass through, since all the points higher than that point would be multiplied by zero. The ideal filter has an absolute transition to zero, rather than the non-ideal attenuation. Please see the graphical description below:

LOW VOLTAGE DETECTORS: This is circuitry which alerts the Central Processing Unit, or Microcontroller Peripheral that the amplitude of the signal in question has dropped below a certain point.

MAGNITUDE: This is a measure of the intensity of a signal.

MAIN LOBE: The main lobe is located by searching, in the frequency domain, for the largest peak among a group of smaller peaks which taper up to it. The main lobe represents the area within which most of the signal's frequency is found. If the signal were composed of exactly one sinusoid operating at a frequency of exactly 440 Hertz, we would find, ideally speaking, a spike at 440 Hertz on the frequency plot.

MAPPING: Using an established mathematical relationship to transfer from one domain to another.

MC6800: A 4-bit Motorola MicroController.

MC68332: A Motorola 32-bit MCU.

MC68HC11A8: An 8-bit Motorola MicroController.

MICROCONTROLLER: A bunch of mostly digital and somewhat analog circuitry that helps us control mainly mechanical equipment; and, as we know, also helps us perform low-level DSP operations.

MICROCONTROLLER PERIPHERALS (MCU): These are functions through which the MCU communicate to the outside world. (Examples: A/D, Timers, Serial I/O ports, etc.)

MODULATION: This is the process whereby we modify the frequency of an existing signal. This word describes the act of multiplying two signals together with the purpose in mind of modifying the frequency properties which each originally had to arrive at one signal with an entirely new frequency domain signature — phew! that was a mouthful — (See "Carrier Signal" for a well known application of this method).

NETWORK FUNCTION: When we are talking about the design of filters in DSP, this idea refers to the ratio of input to output terms.

NATURAL FREQUENCY: Every material on earth has a frequency at which it vibrates best as a result of its own physical properties. This is known as its resonant, or natural frequency.

NON-CAUSAL: If a system is non-causal then it is a system whose output is independent of the inputs (see "Causal").

NON-CAUSAL FILTER: A device whose signal-altering capabilities do not depend upon previous inputs to the system.

NYQUIST CRITERIA: The rule which states that the frequency at which we sample the incoming signal must be twice that of the highest frequency inherent in that signal.

ODD HARMONICS: (See "Harmonics"). Odd harmonics are individual frequencies which are odd integer multiples of the fundamental or main frequency.

ONLY GAME IN TOWN: The only way of doing something.

OPAMP: Operational amplifier.

OPERATIONALAMPLIFIER: A bunch of analog circuitry which can filter, boost or otherwise process analog signals.

OSCILLOSCOPE: This is device which is used to view and analyze the time domain activity of any signal.

OUT-OF-PHASE: Shifted 90 degrees.

PARALLEL TRANSFER: The transfer of bit patterns, pattern by pattern. In other words, the bits are sent in parallel.

PARALLEL INTERFACE ADAPTOR(PIA): This is a device which can link an MCU to a MCU peripheral device.

PASSBAND EDGE: This is a boundary of a filter within which frequencies are passed. **PASSBANDS:** These are ranges of frequencies which are not filtered out by a digital, or analog, filter.

PASSBAND RIPPLE: (See "Gibbs Phenomenon', or, 'Ringing")

PHASE: Phase is a measure which helps to show us the difference between two cyclical signals. Below we see a graph of a sine wave and a cosine wave. If each cycle takes one second, we can see that the time delay between them is 1/4 of a second; however, the phase delay would be expressed not in terms of time, but in terms of cycle differences. In this case we see that we have a 1/4 cycle difference between the two signals. In other words, if we moved the sine wave over by 1/4 of a cycle, we would get the cosine wave.

PHASE, ABSOLUTE: Under this condition we know exactly when the signal started up(See "Phase").

PHASE, RELATIVE: This phrase refers to a common condition of not knowing when a signal has started up; but we do know the relationship between that signal relative to another signal (See "Phase").

PIEZOBUZZER: A piece of piezo-electric material which acts as a buzzer.

PISO: Parallel In Serial Out.

PLCC: Plastic leaded chip carrier.

PREEMPHASIS: A frequency enhancer; that is, a device that will boost the frequencies that we choose to boost (See "Weighting").

QUANTIZATION: When we turn a discrete signal into a digital one, this is known as quantizing.

RAM BUFFER: Random Access Memory (RAM) where we store the quantized signal and the coefficients we will be using to modify it.

REAL-ESTATE: Surface area, in this case referring to printed circuit board area.

RESONANCE: (See "Natural Frequency").

RIDING ON TOP: Superimposed on

RINGING: This is a phenomenon which results from the energy required to change from one intensity level, or, amplitude, to another higher or lower one. Intuitively we can see that as the intensity level shoots up from a lower to a higher level, it may, and quite often will, overshoot the mark — like letting go of one end of a loose spring which then will first overshoot the mark, and then undershoot it as it returns. Finally, it will settle down to the intended level. This is known as ringing(see "Gibbs Phenomenon").

RMS (ROOT MEAN SQUARE): The fundamental measurement of the magnitude of the A.C. signal. Mathematically, we compute this by squaring the signal, taking its average, and then taking the square root of the resulting number.

RMS-TO-DC FUNCTION: The method by which we arrive at the DC equivalent of an RMS value. The DC value that produces the same amount of heat as its RMS counterpart is the equivalent value.

ROM: Read only memory.

SAMPLE: One instantaneous value of an analog waveform.

SAMPLE-AND-HOLDAMPLIFIER: This is the analog circuit that takes an instantaneous value of an incoming waveform and holds it until it is time to take another sample (See "Sample-and-Hold Function").

SAMPLE-AND-HOLD FUNCTION: This is the bridge between the discrete form of a signal and the analog representation of it. This function describes what a Sample-And-Hold circuit would do; which is to take an instantaneous value of a waveform and hold it until such time as another sample is taken. During the time that it is held, the instantaneous value is converted (quantized) into a digital value and stored in a RAM Buffer (See "Sample-and-Hold Amplifier").

SERIAL: Sequential transfer of bit patterns, one bit at a time; and furthermore, one right after the other.

SERIAL COMMUNICATIONS INTERFACE: This device allows a bit-by-bit transfer of information outside the MCU.

SIDELOBES: Whenever we look at a signal's frequency domain plot, we see many small peaks each getting successively larger and larger until it reaches the largest one in that group; then, on the other side of the largest one, each peak gets successively smaller. Sidelobes are all those smaller ones. (See "Main Lobe").

SIGNAL: A signal is anything that has meaning or significance to the person or device receiving it. In other words, a signal is anything that contains information of value to the receiver, like an electrical signal, for instance.

SIGNAL BANDWIDTH: The bandwidth of a signal is defined by the number of different sinusoids that compose it. If a signal contains three sinusoids each operating at a different speed, then its bandwidth is defined by the highest to the lowest of those three contributing frequencies.

SIGNAL FILTER: (See "Filter").

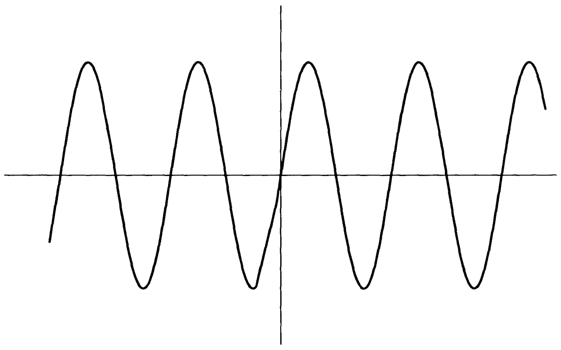
SIGNAL-TO-NOISE RATIO: In designing signal filters, the DSP engineer must account for noise in the system. This ratio is a characteristic of the system which the engineer is always trying to maximize by minimizing the noise with appropriate filtering techniques.

SIGNAL PROCESSING: This the general title for the branch of Electrical Engineering which is devoted finding new ways to alter electrical signals for useful ends whether it be by analog, or digital means.

SINC FUNCTION: (1) This function is fundamental to the field of signal processing. It is often used to show the duality concept when referring to the relationship between the frequency and the time domains. It is used this way since the time domain representation of the frequency domain sinc function is a rectangular pulse; and vice versa, the time domain sinc function is a frequency domain "rectangular pulse," such as a bandpass filter(See "Sinc Pulse"). (2) The function used to 'trace out' a set of discrete impulses at the output of a Digital-to- Analog Converter, thereby producing an analog waveform.

SINC PULSE: This refers to the dual of the sinc function in the opposite domain which is always a rectangular pulse(See "Sinc Function").

SINE WAVE: Below we see a graphical description of a sine wave:



For an explanation of the mathematical detail behind these descriptions, please refer to a Signal Processing text.

SINGLE FREQUENCY: When the author of this chapter refers to "single frequency" he means a sinusoid which has only one frequency.

SINUSOIDS: One idea in DSP which one should understand is that of the Sinusoid. Sinusoids are the signals which when combined in various ways compose all other signals that occur naturally or synthetically in the universe. These are the signals which the DSP enthusiast will filter or reshape for whatever reason. When the mathematician, Joseph Fourier, examined these signals, he made the remarkable discovery that all the signal he studied are part of the set of signals which are composed entirely of sine waves and cosine waves.

SIPO: Serial In Parallel Out.

SNR: Signal to noise ratio.

SOIC: Small Outline Integrated Circuit.

SPEC: Specification.

SPECTRA: This word is the plural form of the word, spectrum (see "Spectrum").

SPECTRALANALYSIS: In one sense, we refer here to the thought process whereby we study and dissect the frequency domain signature of a signal in an effort to gain some understanding of its characteristics. In another sense, it is the act whereby a frequency domain representation of a signal is broken down into its component parts in order to return it to its time domain representation; which, by another name, is the inverse fourier transformation.

SPECTRAL SPIKE: Please refer to the graphic under the glossary heading "Cosine Wave," or, "Sine Wave." A spectral spike in each of those cases is that which is described graphically in the second figure of each entry. It is the single frequency associated with a simple sinusoid.

SPECTRUM: This word refers to the way a signal looks when we view its frequency characteristics. For example, when we look at a rainbow, we are actually looking at how often particular light waves finish one cycle of their life. If one of them finishes one cycle in 15 seconds, then when we view its frequency domain description, we see that its spectrum, or range of frequencies fall into the category of blue colors. A spectrum is simply a range of frequencies.

SPI: Serial Port Interface.

SPIKE: In the frequency domain, this is an area, denoted by a main lobe, where we would find a major concentration of a particular frequency. In the time domain, this can be a representation of an instantaneous energy surge which appears suddenly in time, and disappears just as quickly; which we would view as a spike on an oscilloscope.

SQUARE WAVE: In this section of his chapter, the author is attempting to show how all signals are composed of one or more simple sinusoids. The square wave is a signal which looks like one might expect:

STOPBANDATTENUATION: Everything before the passband edge is passed frequencies, and everything after it until the stopband edge, is stopband attenuation. This is the part of the frequency domain representation of a filter which "rolls off" until it reaches zero amplitude. The major issue which filter designers contend with is how many frequency bands must the stopband attenuation pass through before it reaches zero? The fewer, the better. However, the "faster" stopband attenuation implies more computation, and therefore, more computing time and power to achieve it.

STOPBAND EDGE: Somewhere, just before the stopband attenuation reaches zero, we reach the stopband edge. As in the case of the passband edge, calculus describes how we reach one or the other edge just before we reach either the bottom or the top of our amplitude extremes.

SUCCESSIVE APPROXIMATION CONVERTER: This is an Analog-to-Digital Converter which is used in applications where the speed required is no more than about one million hertz (which by today's standards is not slow, but not blazingly fast either).

SUPERPOSITION: This is a mathematical principle which simply put, states that we can add two signals to get another. This allows us to describe and understand signals which are composed of many sub- signals. Convolution also uses the superposition principle.

SUPERPOSITION INTEGRAL: (See "Convolution Integral").

SYSTEM BANDWIDTH: We are referring here to the range of frequencies that are relevant to a particular system. For example, the system bandwidth of our telephone system is 4000 Hertz. This means that any signals from zero to 4000 Hertz in frequency will be heard by whoever is at the other end of the line.

TIME DELAY: The time delay is the difference in time between two signals.

TIME DOMAIN: When we speak of the time domain, we are signifying that we are interested in any signal behavior, which we observe over time. We could, for example, observe this behavior on an oscilloscope.

TIME DOMAIN DESCRIPTION: This kind of description can be a verbal, a graphical, or a mathematical model, but it must contain elements which always impinge upon the time-domain signal processing goals for which we have set out.

TIME-FREQUENCY PAIRS: (See "Fourier Transform Pairs").

TRAIN OF IMPULSES: This refers to a set of equally spaced delta functions each corresponding to a different point on a signal. This terminology is generally used to describe the effect of sample-and-hold circuitry upon the incoming waveform.

TRANSFER FUNCTION: We are referring, here, to a relationship between the input and the output of our Digital Signal Processing system, whatever that may be. It is generally expressed as a ratio of the mathematical expression of the output sequence, to the expression for the input sequence.

TRIP POINTS: A threshold at which something occurs.

UNIVERSAL ASYNCHRONOUS RECEIVER/TRANSMITTER (UART): This device converts bits that are being transferred one at a time, into a set of bits that are to be transferred as one packet. It can also convert a packet of bits into bits that are to be transferred in serial (See "Serial" and "Parallel Transfer").

VANTAGE POINTS: Views or Perspectives.

VAX PDP-11: A large mainframe computer.

VCA: Voltage controller amplifier.

VOLTAGE FOLLOWER: Circuitry whose voltage output tracks the voltage input. To be more precise, it makes a fine high current buffer because it has high input impedance, low output impedance and unity gain.

WATCHDOG TIMER: These timers are used to protect against faulty computer programs, or situations in which an instruction sequence begins to repeat itself indefinitely and in a way that is outside the control of the user. This countdown timer detects these situations by allowing only a certain number of repetitions to occur before timing out and giving control back to the user.

WEIGHTING: Basically, this is a constant assigned to, say, one frequency item, in order to indicate its importance. In general, it is any constant assigned to any part of a signal to emphasize or de- emphasize it.

WINDOWS: A window, in the Digital Signal Processing sense, is a type of aperture used to filter a given signal (see "Windowing"). The most famous windows are: Rectangular, Hamming, Hanning, Gaussian, and Blackman-Harris. When we get to the point where we want to select the right window for our filtering purposes, we would begin by exercising our ability to juggle with the tradeoffs inherent in each window's characteristics. In general, the characteristics which we would attempt to keep aloft in our heads are: (1) the width of the central peak, or main lobe; (2) the passband edge location; (3)

is relative to the main lobe, and how 'quickly' the other sidelobes taper off to nothing; and, (4) the amount of area beneath the window function's frequency domain description. (Adapted from DSP in VLSI, by Richard J. Higgins).

WINDOWING: Windowing is like taking a photograph. If we are very concerned about getting the most complete photograph that we can, we concern ourselves with getting the appropriate film for the occasion (color, or black and white; high-speed shutter sensitivity; and so on). Then, before actually pushing the shutter release button, we would try to get the best possible picture of the event given the size of our frame. It is the same way with windowing; that is, before we actually start snapping pictures, we choose the right film (the window) which will have all the right qualities given the kind of picture we are trying to take of the signal; then we try to get everything into the picture that we need (choose the width of the window properly); and then we start taking pictures.

References

The Fast Fourier Transform, E. Oran Brigham, Prentice-Hall, 1974.

Digital Signal Processing In VLSI, R. J. Higgins, Prentice-Hall, 1988.

The Fourier Transform And Its Applications, 2nd edition, R. N. Bracewell, McGraw-Hill, 1978.

Signals And Systems, Alan V. Oppenheim and Alan S. Willsky with Ian T. Young, Prentice-Hall, 1983.

Digital Signal Processing, A. V. Oppenheim and R. W. Schafer, Prentice-Hall, 1975.

NOTES