A Software Engineer's Guide to the System Interface of the DP83266 MACSI™ Device

National Semiconductor
Application Note 964
Robert Macomber
August 1994

1.0 INTRODUCTION

This document describes features and mechanisms of the System Interface portion of National's DP83200 FDDI chipset that are important for software designers to know about. The information presented corresponds to the DP83256A BSI-2™ Device. However, this information also applies to the System Interface portion of the DP83266 MACSI Device as well as the DP83256 BSI™ Device. Throughout this document we use the generic term System Interface (SI) to refer to all three of these devices.

The goal of this document is to reduce your learning curve (and implementation time) as a software designer using the SI by clearly describing how the SI operates and suggesting methods of interacting with the chip. Since the SI can be used in many different system environments (e.g., direct bus connection, local memory) an effort has been made to make the discussion as general as possible. This document includes:

- An overview of the System Interface (SI)
- A description of the data structures used to transmit and receive FDDI frames
- A complete tutorial on the SI queues
- A description of the steps involved with sending and receiving FDDI frames
- An examination of various software design issues (i.e., memory mgmt., performance)
- An overview of some low-level SI operations
- A description of the steps involved with initializing the SI
- A description of the exception conditions and ideas for handling them

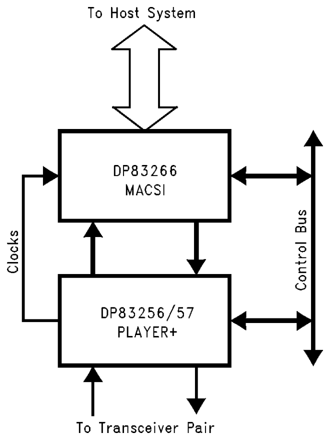
This document should be read in conjunction with the appropriate device datasheet; though it should be possible to get a good understanding of the chip using this document alone. An example of SI interface routines, the BSI Device Primitives, is also available from National Semiconductor. These example routines are free of charge. Please contact your local Sales Engineer for more information.

2.0 AN INTRODUCTION TO THE SYSTEM INTERFACE (SI)

This section of the document presents a basic overview of the System Interface portion of the FDDI chipset, discusses the design philosophy of the SI and offers a survey of features that are important for software designers to know about.

2.1 Overview of the System Interface (SI)

The SI architecture provides a high-level, high-performance system interface for National Semiconductor Corporation's DP83200 FDDI (Fiber Distributed Data Interface) chip set. It provides a simple, powerful interface for sending and receiving frames on FDDI networks. It includes features important to software designers who are implementing high performance interfaces to FDDI networks. *Figure 2-1* shows the MACSI Device (which integrates both the SI functions and the BMAC™ functions) as well as the PLAYER+™ Physical Layer Controller.



TL/F/12304–1

FIGURE 2-1. FDDI Chip Set Block Diagram

The basic operation of the SI is straight forward. It receives and sends FDDI frames. More specifically, it transfers FDDI frames received from the BMAC Device into buffers previously defined by the host and generates status information that describes where the received frames have been placed in memory. It also transfers FDDI frames from buffers supplied by the host to the BMAC Device and generates status information about the frame transmission. At the center of this activity is a set of circular queues located in memory that is shared by the host and the SI. With these memory resident queues, the SI and host engage in classic consumer/producer relationships. In particular:

- The host produces transmit requests and consumes transmission status information

- The host produces buffers that will hold received frames and consumes received frames
- The SI consumes empty receive buffers and produces received frames
- The SI consumes transmit requests and produces transmission status information

The host software controls the operation of the SI by manipulating:

- directly accessible registers (called Control Bus Registers)
- registers that are internal to the SI and accessed indirectly by the host (called Pointer RAM and Limit RAM Registers)
- a memory resident mailbox (used to load and store some internal registers)
- several queue related data structures that reside in memory accessible by both the host processor and the SI.

See *Figure 2-2* for an illustration of where these registers and data structures exist. It should be noted that some memory must be mutually accessible by both the host processor and the SI. The host accesses memory directly, (though at the hardware level, the host and SI may arbitrate for memory bandwidth). It does not use the FDDI chip set to reach this memory (as is done with some other FDDI chip sets).

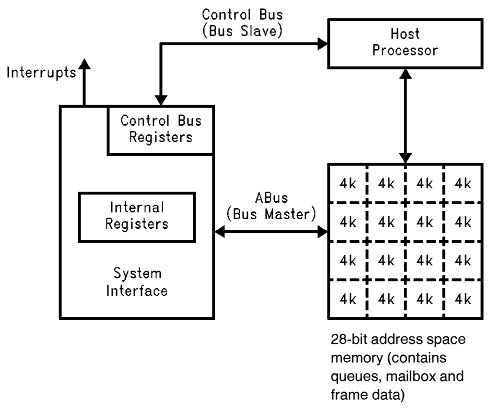


FIGURE 2-2. SI Device, Host Processor and Memory Relationships

2.2 Design Philosophy of the SI

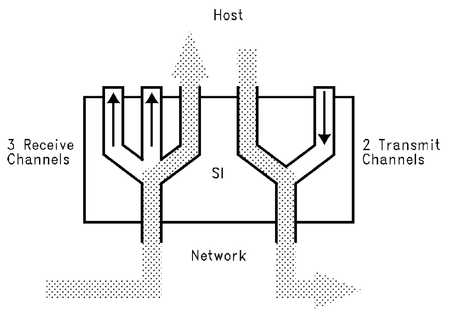
The architecture of the SI differs from many analogous Ethernet chips (including National Semiconductor's own SONIC™ Device). Why is this so? FDDI and Ethernet are very different types of networks. FDDI is an order of magnitude faster than Ethernet. The way that stations arbitrate for network bandwidth is completely different, albeit much more equitable in FDDI. FDDI's maximum frame size is roughly three times bigger than Ethernet. FDDI requires that each station engage in a rather complicated set of protocols for network connection and operation (collectively called FDDI Station Management or SMT), which greatly increase the robustness of the network at the cost of more network software/firmware. FDDI supports different classes of frames that are intended for different entities within a station (SMT, MAC, LLC, etc.). The SI was designed to meet these challenges of FDDI and provide features that facilitate the design of robust, high performance network products. Because FDDI is more demanding than Ethernet, different tradeoffs were made.

The design of the SI was driven by many goals. The three most important goals were 1) performance, 2) robustness and 3) FDDI specific functionality.

- Performance. FDDI is ten times faster than Ethernet (100 Megabits per second). Things happen much more quickly and happen much more often than with Ethernet. Also, due to the way that network bandwidth is distributed among stations, traffic on an FDDI network can be very ''bursty'', so that a station can experience periods of very intense network activity. Frames can arrive with very little idle time between each frame. To handle this type of traffic the SI is capable of simultaneously producing received frame status for one frame while processing the next incoming frame. The system interface is capable of generating and receiving frames at full FDDI bandwidth. The SI also has mechanisms in place to minimize the volume of status information that the host needs to process and make the best use of available memory bandwidth. The data structures were designed to be efficient for both the host and SI (i.e., minimize fetches). The concept of minimizing the volume of status information is also manifested in an interrupt batching scheme, where a single interrupt is generated after multiple frames have been received. Interrupt batching is very important when dealing with FDDI speeds. For example, a series of 100 byte frames can arrive only 8 to 10 microseconds apart.
- Robustness. The FDDI protocols have a lot of inherent reliability features (e.g., dynamic reconfiguration and monitoring of link quality). This emphasis on robustness is also pervasive throughout the entire DP83200 FDDI chip set. The SI was designed to be consistently reliable and operate deterministically. The data structures and queuing mechanisms, in particular, were designed to be extremely robust. For example, all queues are unidirectional with clearly defined producers and consumers; which makes the debugging of queue logic quite straightforward.
- FDDI Specific Functionality. To make FDDI work well, there are a number of features that should be available. For example, there are several different types of frames that may each need to be handled by different software or hardware agents (i.e., SMT frames, synchronous frames, restricted dialogs, LLC Frames of different priorities, etc.). The system interface needs to help the network driver(s) multiplex and de-multiplex this frame traffic. Also, FDDI requires that stations limit the amount of bandwidth consumed under certain circumstances (e.g., synchronous frames) and control the capture and issuance of different kinds of tokens (e.g., for restricted dialogs). National's FDDI chip set also provides features that take advantage of the fact that each station ''strips'' the frames that it transmits. The host may receive status information that includes information gathered after the frame has traversed the ring and is being stripped from the FDDI network. For example, it is possible to determine if the receiving station has ''dropped'' a frame; perhaps due to congestion.

2.3 SI Features

The SI supports two independent Transmit Channels and three independent Receive Channels. Concurrently queued transmit requests are serviced on a priority basis and incoming FDDI frames are sorted to one of three Receive Channels (Receive Channel 0 is dedicated to SMT usage). Channels allow frames to be sent and received by different pieces of software or they can be used to help software meet performance goals by sorting incoming frames onto different Receive Channels based upon the frame class. In addition, there is an option for splitting frames where the headers are placed in one buffer pool and the rest of the frames are, optionally, placed into another buffer pool. See *Figure 2-3* for an illustration of the SI Channels.



TL/F/12304–3

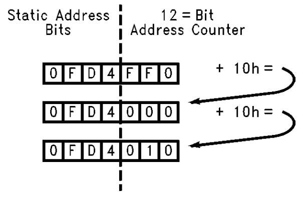
FIGURE 2-3. SI Channels

A skip filter is available on Receive Channel 0 that causes the SI to automatically discard duplicate MAC frames. During FDDI ring initialization thousands of duplicate MAC frames (Claim and/or Beacon) can be circulating on the network. When the skip filter is enabled the first instance of the frame is received while redundant frames are ignored. Thus the station can examine the MAC frames used in ring initialization without the penalty of having to perform unnecessary frame processing.

The data structures used in frame transmission and reception map nicely to data structures frequently used with layered protocols. The SI is capable of doing ''gather reads'' when processing transmit requests and a limited form of ''scatter writes'' when receiving frames from the network. On systems where the SI is attached to system memory it may be possible to avoid copying frame data by configuring the SI to read and write directly from system dependent data structures. For example, on systems where the protocol stack uses mbuf/mcluster data structures the SI may be configured to write frame data directly into mclusters (provided that the cluster size is 4 kb or larger).

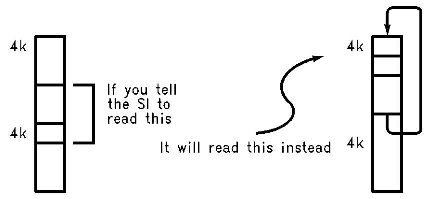
The SI eliminates the need for network drivers to maintain a separate link layer transmit queue. (The SI datasheet uses OSI terminology where ''Request'' is equivalent to ''Transmit'' and ''Indicate'' is equivalent to ''Receive''. For the sake of readability the more conventional terms are used in this document.) Instead frame transmission requests from upper-level software can be directly appended to one of the SI's Transmit Channel Queues. Frames can be dynamically queued for transmission while the SI is concurrently sending other frames from the same queue (without any race conditions). In addition, there are mechanisms available to provide link layer flow control. This is very important when sending FDDI synchronous frames, as each station must explicitly limit the amount of synchronous network bandwidth it uses at each token opportunity.

The MACSI and BSI-2 versions of the SI can be configured to automatically flip address bits when sending and receiving FDDI frames (the ordering of bits within each FDDI address octet is reversed from the IEEE canonical form used on Ethernet networks). This option can be enabled on some Channels and disabled on others; which is useful when SMT doesn't want the bits reversed and the ''normal'' LLC data path does want the bits reversed (actually this is the most common situation). This feature is not available on previous versions of the BSI Device. The SI is page oriented. It logically divides memory into 4 kb pages. This notion permeates the internal design of the SI. As a general rule, the host must not supply data to the SI that crosses a 4 kb page boundary. (Frame data larger than 4k can be described to the SI as multiple independent chunks of memory that happen to be contiguous. This concept is more fully explained in Section 4 ''Sending a Frame''.) This is due to the way that the SI internally generates addresses. The SI has a 28-bit address space, but the internal address counter that the SI uses to step through memory is 12 bits wide. The upper 16 address bits remain static during memory accesses of a given piece of data (i.e., descriptor queues, frame data, arrays of descriptors, etc.). If the SI is instructed to read across a 4 kb boundary the 12 bit address counter will roll-over and the effective address will ''wrap'' back to the beginning of the 4 kb page. See *Figure 2-4* for an illustration of address ''wrapping''. The SI presents interrupt status information in a hierarchical manner. This decreases the amount of time required to determine what event caused a given interrupt to occur.



TL/F/12304–4

a. Address Counter Roll-Over



TL/F/12304–5

b. Effect of Address Counter Roll-Over

FIGURE 2-4. Four kb Address Counter Roll-Over

The SI is a full-duplex device; meaning that it can simultaneously send and receive data. Full duplex operation is implemented throughout the DP83200 chip set. This feature, along with four available loopback paths in the MACSI and PLAYER+ devices, significantly increases the "testability" of a station's network interface. As a Power Up Self Test, the network interface software can test for good connectivity between the various DP83200 devices and proper operation of the entire chip set by sending and receiving frames across the four available internal loopback paths.

3.0 DATA STRUCTURES

This section covers the data structures shared between the host and the SI. As these data structures are all, in some fashion, related to the SI queues, the discussion will start with an overview of the SI queues, followed by an exposition of the various data structures and then finish up with a thorough tutorial about the SI queues. A full understanding of the operation of these queues is absolutely crucial for any software engineer who is developing software that will interface with the SI.

3.1 Introduction to the SI Queues

The SI queues are the primary interface for communication between the host and the SI. The host uses the queues to produce frames for transmission (and consume transmit status information) and consume frames that have been received from the network (and produce empty receive buffers). All SI queues are unidirectional; meaning that the SI either consumes descriptors from a given queue or it produces descriptors for a given queue.

The SI operates independently and asynchronously from the host processor. Many of the older network interface devices use a "batch oriented" approach when transmitting frames. In this scheme a limited number of frames are placed in a staging area, a "Go" command is issued and the host waits for all of the frames to be sent before attempting any further frame transmission. This method of operation sometimes requires that the network interface software provide a link layer queue to hold transmit requests that arrive when the hardware is busy. In contrast the SI is designed to dynamically accept transmit requests at any time (i.e., like a FIFO). The host doesn't have to "spoon feed" frames to the SI.

A crude analogy can be made between network interface hardware (i.e., the SI) and ovens for baking bread. A "batch oriented" oven holds a limited number of loaves for a fixed period of time, while a "continuous process" oven is constantly moving loaves through the heated area. A "continuous process" oven, of the same internal capacity, can bake many more loaves of bread than a "batch oriented" oven over the same span of time. To complete the analogy, the SI may be thought of as a "continuous process" network interface device with the queues acting as "conveyor belts" that carry frames. The net result is that a station can transmit and receive more frames with less overhead. This is an important concept and one that should be exploited by software designers.

Where do the SI queues exist?

The SI queues exist in memory that is shared between the host and the SI. The actual data of each queue does not reside within the SI, although it does internally retain a unique Queue Pointer for each queue.

What types of queues does the SI support?

There are four kinds of queues. Each queue is defined by the type of descriptor used within it. Each queue type supports one and only one kind of descriptor.

- Request Descriptor (REQ) queues contain Request Descriptors (REQs). This type of queue is used by the host to queue frames for network transmission.
- Confirmation Descriptor (CNF) queues contain Confirmation Status Message Descriptors (CNFs). This type of queue is used by the host to obtain confirmation about previously queued (and sent) frames.
- Input Data Unit Descriptor (IDUD) queues contain Input Data Unit Descriptors (IDUDs). This type of queue is used by the host to find out about frames that have been received from the network.
- Pool Space Descriptor (PSP) queues contain Pool Space Descriptors (PSPs). This type of queue is used by the host to declare empty buffers where the SI may store incoming frames.

Each Channel has two queues associated with it. Transmit Channels have a REQ queue and a CNF queue. Receive Channels have an IDUD queue and a PSP queue. *Figure 3-1* shows how descriptors flow between the host and the SI using these four types of queues.

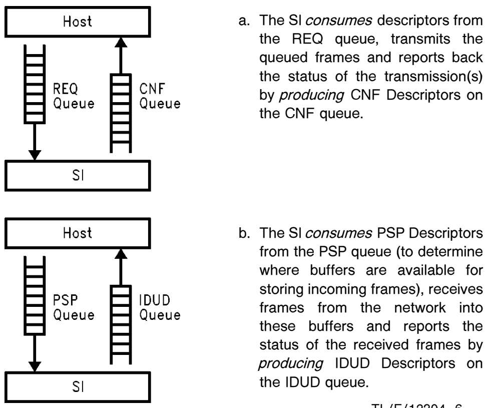
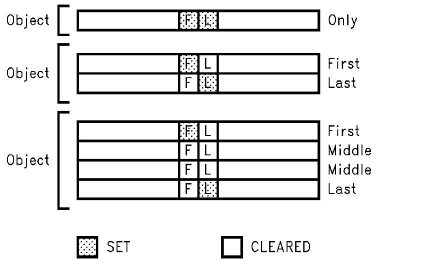

Host
REQ Queue
CNF Queue
SI
a. The SI *consumes* descriptors from the REQ queue, transmits the queued frames and reports back the status of the transmission(s) by *producing* CNF Descriptors on the CNF queue.
Host
PSP Queue
IDUD Queue
SI
b. The SI *consumes* PSP Descriptors from the PSP queue (to determine where buffers are available for storing incoming frames), receives frames from the network into these buffers and reports the status of the received frames by *producing* IDUD Descriptors on the IDUD queue.
TL/F/12304–6

FIGURE 3-1. Flow of Descriptors between Host and SI

If the SI can do "gather reads" of frame data this implies that more than one descriptor may be required to completely describe a packet. How is this done?

All SI descriptors can be bound together into "multi-descriptor objects". To achieve this grouping all SI descriptors have two bits defined: the "First" and "Last" bits. The first descriptor of an object must have the "First" bit set and, not surprisingly, the last descriptor of an object must have the "Last" bit set. Any in-between descriptors are termed "Middle" and must have both the "First" and "Last" bits cleared. A descriptor with both the "First" and "Last" bits set is termed an "Only" and describes a complete object. See *Figure 3-2* for a graphical representation of a few of the possible descriptor groupings using the First (F) and Last (L) bits.



TL/F/12304–7

FIGURE 3-2. Sample Descriptor Groupings

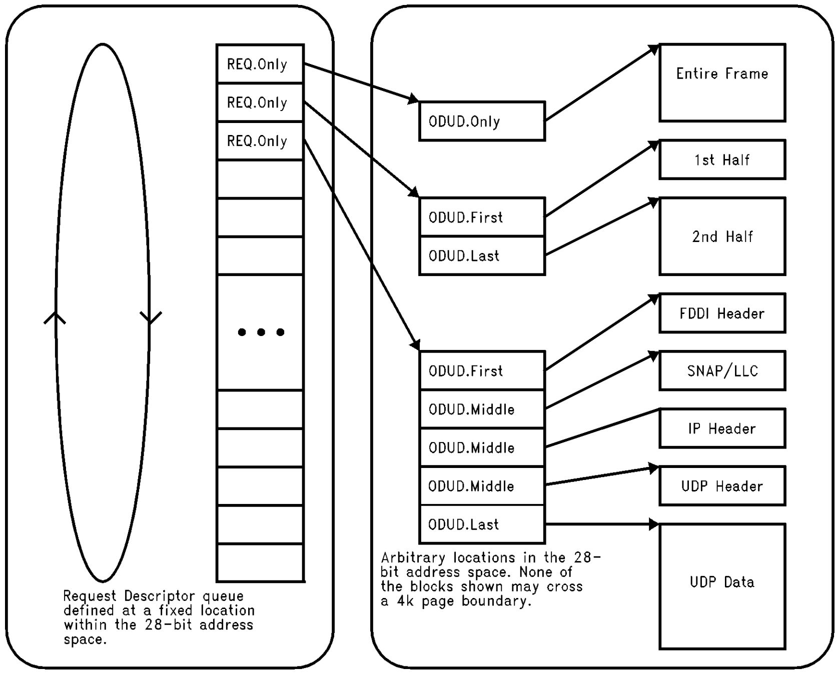
This document uses the term "object" to denote one or more descriptors that logically belong together. Groups of REQ Descriptors are termed "Request Objects". Groups of IDUD Descriptors are termed "Indicate Objects". Groups of CNF Descriptors are termed "Confirmation Objects". Multi-descriptor PSP objects are not currently processed in groups, but rather on an individual basis. For future compatibility it is recommended that all PSP Descriptors be configured as PSP.Only (both First and Last bits set). The following sections on SI data structures supply concrete examples of how the First and Last bits are used.

3.2 Transmit Data Structures

3.2.1 An Overview of the Transmit Data Structures

The SI has a three level transmit data structure. It consists of two levels of descriptors and buffers of frame data. The first level consists of Request Descriptors (REQ) that reside in a Request Descriptor queue. Each REQ Descriptor includes a frame count and an address pointer to the second level of the transmit data structure. The second level consists of an array of Output Data Unit Descriptors (ODUDs) that, in turn, each define the address and length of frame data buffers (called Output Data Units (ODUs) in the datasheet). The third level consists of buffers of frame data. See *Figure 3-3* for an example of some single frame transmit data structures.

The three level transmit data structure employed by the SI offers a flexible vehicle for building multi-frame Request Objects. It supports applications that need to dynamically batch single frame transmit requests into multi-frame Request Objects as well as applications that need to queue a group of frames (i.e., a TCP window) in a single atomic operation. Building multi-frame Request Objects is important for performance reasons. See *Figures 3-4* and *3-5* for examples of some multi-frame transmit data structures.


REQ.Only
REQ.Only
REQ.Only
• • •
Request Descriptor queue defined at a fixed location within the 28-bit address space.
ODUD.Only
ODUD.First
ODUD.Last
ODUD.First
ODUD.Middle
ODUD.Middle
ODUD.Middle
ODUD.Last
Arbitrary locations in the 28-bit address space. None of the blocks shown may cross a 4k page boundary.
Entire Frame
1st Half
2nd Half
FDDI Header
SNAP/LLC
IP Header
UDP Header
UDP Data

TL/F/12304–8

FIGURE 3-3. Some Example Single Frame Transmit Data Structures

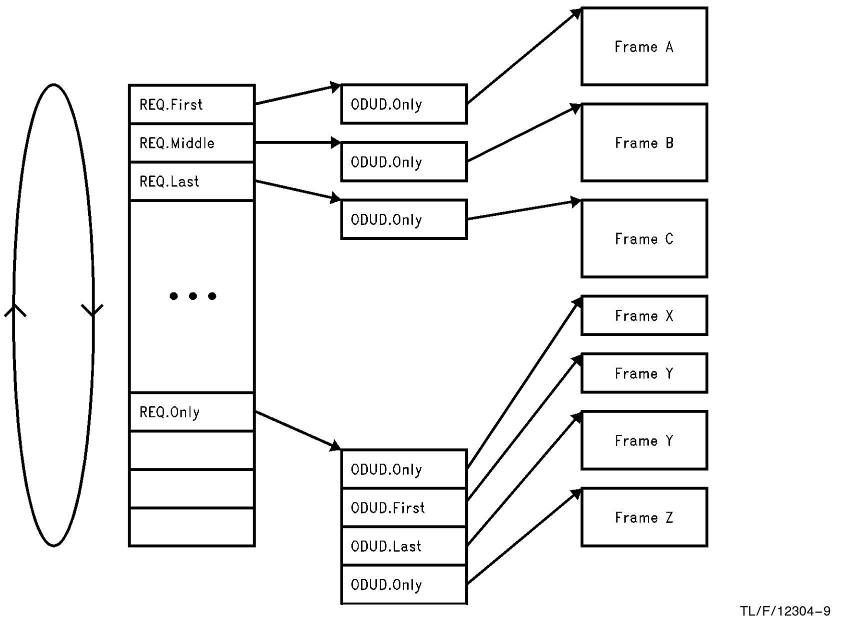


FIGURE 3-4. Some Example Multi-Frame Transmit Data Structures (Shows the Two Mechanisms for Building Multi-Frame Request Objects)

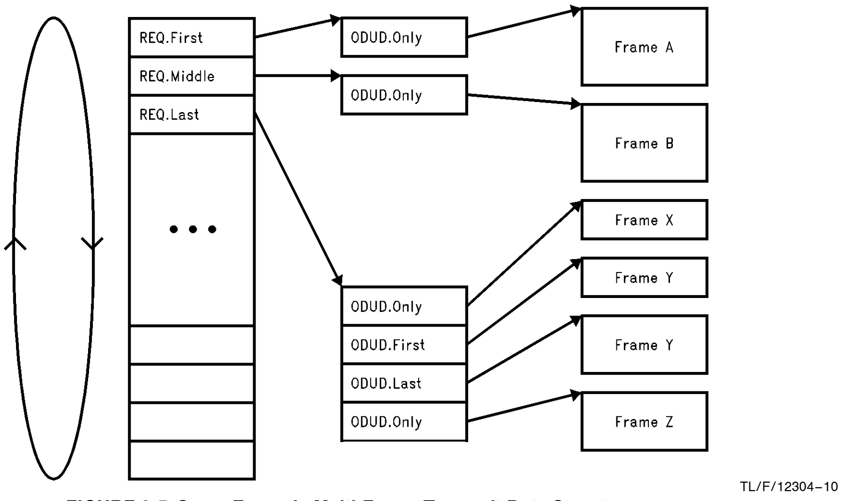


FIGURE 3-5. Some Example Multi-Frame Transmit Data Structures (Combines the Two Mechanisms for Building Multi-Frame Request Objects)

A Request Object consists of one or more REQ Descriptors grouped together using First and Last bits and all of the ODUDs and ODUs referenced by the REQ Descriptors. (During normal operation Request Objects should be limited to a maximum size of 255 frames.) The Request Object is the unit of consumption used by the SI when consuming transmit requests. The general flow control rule is that the SI processes one Request Object per Channel per service opportunity, thus the host may implement fine grained link layer flow control (at the network service opportunity level) by modulating the size of the Request Objects that it produces. If both of the two Transmit Channels each have at least one Request Object queued, then two Request Objects will be consumed upon the next service opportunity. (FDDI service opportunities are briefly described in the glossary at the end of this document.)

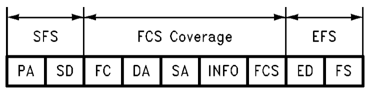
The SI automatically performs a series of consistency checks when consuming a Request Object. For example, if the SI consumes a REQ.First followed immediately by another REQ.First it will report a consistency failure and attempt to discard REQ Descriptors until a REQ Descriptor is found with the Last bit set.

Many Ethernet system interfaces, including National Semiconductor's SONIC Device, chain together descriptors into a linked list. The SI uses an array of ODUD Descriptors instead of a linked list. This was done to minimize the number of memory fetches needed to consume a transmit request (no "next descriptor" pointer to fetch). Plus, the use of First/Last bits allows the SI to perform consistency checks when processing a stream of ODUD Descriptors.

3.2.2 Transmit Frame Layout

When the frame data buffers (ODUs) are concatenated they present an entire FDDI frame to the SI. What frame format does the SI expect to find inside these ODUs?

The FDDI MAC Standard (X3.139-1987) defines the format of an FDDI frame as shown in *Figure 3-6*.



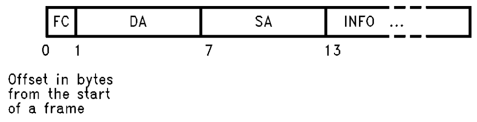
TL/F/12304–11

SFS	Start of Frame Sequence	INFO	Information
PA	Preamble	FCS	Frame Check Sequence
SD	Starting Delimiter	ED	Ending Delimiter
FC	Frame Control	FS	Frame Status
DA	Destination Address	EFS	End of Frame Sequence
SA	Source Address		

FIGURE 3-6. FDDI Frame Format

The SI expects to be presented with FDDI frames structured with place holders for the Frame Control, Destination Address and Source Address fields. The INFO field may be zero or more bytes and the host may optionally supply its own FCS field. The start of the frame data (or any of the ODUs for that matter) need not be aligned in any special way (i.e., frame data can start on any byte boundary). None of the blocks of frame data may cross a 4 kb page boundary, though any single, contiguous block of frame data can be represented by multiple ODUDs. The frame must always include a place for the Frame Control, whether or not Frame Control Transparency is used. (Frame Control Transparency is useful for SMT usage, which routinely uses different FDDI Frame Control values. This mode is enabled via the Transmit Channel's Request Configuration Register (R0CR0 or R1CR0).) The frame must always include a place for the Source Address, whether or not Source Address Transparency is used. Source Address Transparency is useful in transparent bridging applications. This mode is enabled via the Transmit Channel's Request Configuration Register (R0CR0 or R1CR0) and is normally used with "Void Stripping" (also enabled via the Request Configuration Register).

A typical frame (long addresses and FCS automatically generated by the BMAC device) will have the format shown in *Figure 3-7*.



TL/F/12304–12

FIGURE 3-7. Format of FDDI Frame within Output Data Units

3.2.3 The Request Descriptor

Request Descriptors (REQ) exist on REQ queues and are produced by the host. They function as the root of the three-level transmit data structure. In addition to describing what FDDI frames to transmit, the REQ Descriptor describes how the FDDI frames should be transmitted and what type of transmit status information the SI should produce. REQ Descriptors can be grouped together into a multi-descriptor Request Object by using the First/Last bits found in all of the SI descriptors. See *Figure 3-8* for an illustration of the REQ Descriptor.

31 30	29 28	27 24	23 16	15 12	11 8	7 0	
RES	UID	UID	SIZE	CNFCLS	RQCLS	FC	**Word 0**
F-L	RES	LOC	LOC	LOC	LOC	LOC	**Word 1**

FIGURE 3-8. Request Descriptor (REQ)

The SI examines the UID, CNFCLS, RQCLS and FC fields when consuming the first REQ Descriptor in a Request Object (i.e., the First bit is set). These values are then used for all REQ Descriptors in the rest of the Request Object. The LOC (address of ODUD array) and SIZE (frame count) fields of all REQ Descriptors are always used by the SI. A brief explanation of each field follows. The databook should be examined for a full explanation of these fields.

- The UID, User Identification, field (6 bits) may be programmed by the host with any value. This value reappears in the transmit status information (CNF Descriptors) that correspond to the transmit request. This field can be used by the host to match up transmit requests with transmit status.
- The SIZE, frame count, field (8 bits) declares the number of frames described by the current REQ Descriptor; not the number of ODUDs. The SI counts ODUD Descriptors that have the Last bit set and compares this value with SIZE.
- The CNFCLS, Confirmation Class, field (4 bits) declares the class of transmit status information that the SI should produce. This class describes three different kinds of status generating behavior, each represented by a bit, that can be combined in the following ways (the device datasheets includes a fourth bit, repeat frame (R) that is not related to transmit status generation). Many applications use a confirmation class of Tend to generate transmit status information solely for the purpose of transmit data memory reclamation. See Table 3-1.

TABLE 3-1. CNFCLS Components

CNFCLS	End	Intermediate	Full/Transmit	Description
NONE	False	False	Don't Care	**NONE.** CNF Descriptors are *produced* only when a transmit exception occurs.
TEND	True	False	False	**TEND.** A single CNF Descriptor is *produced* for each *consumed* Request Object (NOT one for each REQ Descriptor). CNF Descriptors are *produced* at frame transmission time.
TINT	True	True	False	**TINT.** The same as **TEND**, plus CNF Descriptors are *produced* at the end of each transmission (token) opportunity (provided that the Request Object spans more than one transmission opportunity).
FEND	True	False	True	**FEND.** A single CNF Descriptor is *produced* for each *consumed* Request Object (NOT one for each REQ Descriptor). CNF Descriptors are *produced* at frame stripping time and populated with additional status information about the returning frames.
FINT	True	True	True	**FINT.** The same as **FEND**, plus CNF Descriptors are *produced* at the end of each transmission (token) opportunity (provided that the Request Object spans more than one transmission opportunity).

- The RQCLS, Request Class, field (4 bits) denotes how the BMAC device should behave when transmitting the frames of the Request Object. It describes the type of token (if any) that must be captured before sending the frames, how the frames should be sent (e.g., synchronous, asynchronous), the type of token (nonrestricted, restricted, none) that will be issued after sending the frames and whether or not the BMAC device should use the normal (asynchronous) rules for determining token usability in a restricted dialog. (Different criterion are used for asynchronous and synchronous frames when determining if a frame may be transmitted at a given token opportunity.) Restricted dialogs can be thought of as a supervisor (kernel) mode for access to an FDDI network. This capability is *not* currently used by FDDI Station Management (SMT). Logical Link Control (LLC) traffic would typically use a value of 1000 binary (Asyn) to denote that the BMAC device should capture a non-restricted token, send asynchronous frames in a manner that respects the integrity of the FDDI timed token protocol and issue a non-restricted token after frame transmission. The network driver should enforce the use of appropriate request classes (e.g., don't allow IP packets to go out with a RQCLS of ''immediate'').
- The FC, Frame Control, field (8 bits) can be used to optionally override, at network transmission time, the FDDI Frame Control value of the FDDI frames of the Request Object. This will be done if Frame Control Transparency has not been enabled in the Transmit Channel's Request Configuration Register (R0CR or R1CR).
- The F-L, First-Last, fields (2 bits) are used to define groupings of one or more descriptors.
- The LOC, Location, field (28 bits) holds the address of an array of ODUD Descriptors.

What is the difference between ''Full'' and ''Transmit'' Confirmation?

Both ''Full'' and ''Transmit'' confirmation result in the generation of CNF Descriptors on the Transmit Channel's CNF queue. They differ in the time at which CNF Descriptors are produced and in the type of valid information contained in the CNF Descriptors.

Transmit confirmation is generated at frame transmit time and, therefore, includes only information that can be gathered when transmitting frames. The UID is always valid as is the Transmitted Frame Count (TFC) and the Request Status (RS). When using transmit confirmation the Frame Attributes (FRA), Frame Status (FRS), Confirmed Frame Count (CFC), and some bits of the Confirmation Status (CS) are invalid.

Full confirmation is generated at frame stripping time. The information that is valid under full confirmation is a super set of transmit confirmation (All of the fields of the CNF Descriptor are valid). While full confirmation does provide more information, the real power of this option is that it provides the ability to terminate wasteful transmissions.

When sending a large amount of data to another station (i.e., a TCP window) the receiving station may be receiving more frames than it can handle. In this case the sending station is wasting network bandwidth and system resources (e.g., bus bandwidth) by transmitting frames that will be ''dropped''. When using full confirmation the host may define certain conditions that will halt transmission and flush the remainder of the current Request Object. This is accomplished by configuring the Transmit Channel's Expected Frame Status Register (R0FSR or R1EFSR). For example, the host may specify that transmission will halt if the receiving station doesn't copy a frame or if a frame becomes corrupted on the network.

What is the difference between End and Intermediate Confirmation?

The End confirmation classes, Transmit End (Tend) and Full End (Fend), cause the SI to produce a single CNF Descriptor for each Request Object.

The Intermediate confirmation classes, Transmit Intermediate (Tint) and Full Intermediate (FINT) cause the SI to produce a CNF Descriptor for each service opportunity in which frames, of a given Request Object, have been sent. The confirmation count fields of the CNF Descriptors contain a cumulative, running count of what has been sent. For example, if a Request Object includes 20 frames which are actually transmitted in three service opportunities, then the SI will produce three CNF Descriptors perhaps with Transmitted Frame Count (TFC) values of 7, 14 and 20. The actual value depends upon the duration of each service opportunity.

3.2.4 The Output Data Unit Descriptor

Output Data Unit Descriptors (ODUDs) are the middle level of the three-level transmit data structure. They do not exist in a queue, but rather in arrays placed at arbitrary locations within the 28-bit addressable memory range of the SI. Each ODUD defines the address and length of part of a frame (perhaps the entire frame) queued for transmission. Those ODUD Descriptors that refer to a given frame are grouped together via First and Last bits. See *Figure 3-9* for a picture of the ODUD. A brief explanation of each field follows. The databook should be examined for a full explanation of these fields.

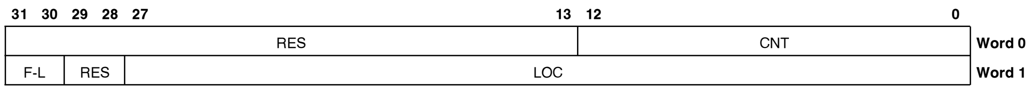


FIGURE 3-9. Output Data Unit Descriptor (ODUD)

- The CNT, count, field (13 bits) tells the SI the length of a chunk of frame data referred to by the ODUD.
- The F-L, First-Last, fields (2 bits) are used to define groupings of one or more descriptors.
- The LOC, location, field (28 bits) holds the address of a chunk of frame data referred to by the ODUD (called an Output Data Unit (ODU)).

ODUD arrays may not cross 4 kb page boundaries. If the SI is ‘‘asked‘’ to fetch ODUDs from an array that spans two 4k pages, the SI will ‘‘wrap’’ back to the beginning of the first page. This behavior can be exploited to implement a very simple memory management scheme for ODUDs. For example, the host software could maintain a 4 kb page of memory (aligned on a 4 kb page boundary) for ODUD arrays. The same type of logic used to acquire space on a REQ or PSP queue could be applied to managing such a block of ODUD memory.

3.2.5 The Confirmation Status Message Descriptor (CNF)

The SI produces transmit status information in the form of CNF Descriptors on a CNF Descriptor queue. The host can determine the success or failure of a transmit request by consuming CNF Descriptors from the queue. Some of the fields are not valid unless Full Confirmation is specified in the first REQ Descriptor of the Request Object See *Figure 3-10* for a picture of the CNF Descriptor. A brief explanation of each field follows. The databook should be examined for a full explanation of these fields.

- The RS, Request Status, field (4 bits) enumerates the status of the transmit request. This field is always valid.
- The FRA, Frame Attributes, field (4 bits) describes the terminating condition and address matching flags (Destination or Source Address equal to MAC address). These values are collected as the frame is stripped off of the ring. This field is only valid when Full Confirmation is used.
- The FRS, Frame Status, field (8 bits) contains the FDDI E, A, and C Indicators (Error Detected, Address Recognized and Frame Copied) and two flags that indicate if the FCS on the frame agrees with frame data and if the frame is well formed. These values are collected as the frame is stripped off of the ring. This field is only valid when Full Confirmation is used.
- The TFC, Transmitted Frame Count, field (8 bits) records the number of frames transmitted. This field is always valid. When using Tint or Fint confirmation the SI will produce a CNF Descriptor at the end of each service opportunity and this field will contain a running total of the number transmitted frames. This field is not accurate for Request Objects containing more than 255 frames.
- The CFC, Confirmed Frame Count, field (8 bits) records the number of frames confirmed. This count is based upon frames stripped off of the ring. This field is only valid when Full Confirmation is used. When using Tint or Fint confirmation the SI will produce a CNF Descriptor at the end of each service opportunity and this field will contain a running total of the number of confirmed frames. This field is not accurate for Request Objects containing more than 255 frames.
- The F-L, First-Last, fields (2 bits) are used to define groupings of one or more descriptors.
- The UID, User Identification, field (6 bits) matches the UID field programmed by the host in the first REQ Descriptor of the Request Object. The basic idea is that all CNF Descriptors produced by the SI include a tag that identifies which Request Object the status information is all about. The host can use the UID field to relate transmit status information to a given Request Object. Since this field is 6 bits wide, 26 (64) Request Objects can be uniquely paired with CNF Descriptors. Request objects may include multiple REQ Descriptors.

31 30	29 28	27 24	23 16	15 8	7 0	
RS		FRA	FRS	TFC	CFC	**Word 0**
F-L	UID		FC	CS	RES	**Word 1**

FIGURE 3-10. Confirmation Status Message Descriptor (CNF)

- The FC, Frame Control, field (8 bits) contains the FDDI Frame Control value of the last confirmed frame. This field is valid only when using Full Confirmation (Fend, Fint).
- The CS, Confirmation Status, field (8 bits) contains several miscellaneous flags related to the confirmation process. Some of these bits relate only to Full Confirmation.

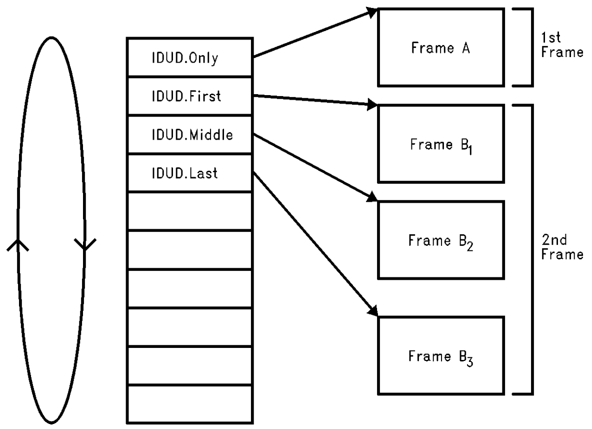
3.3 Receive Data Structures

3.3.1 Overview of Receive Data Structures

The SI uses a two level data structure to represent frames that have been received from the network. The receive data structure is extremely straightforward. The top level consists of Input Data Unit Descriptors (IDUDs) organized as a queue and the bottom level consists of frame data that has been placed in memory by the SI. See *Figure* 3-11 for a depiction of the receive data structure generated by the SI. The format of the IDUD is a super set of the ODUD. This was done to allow FDDI-to-FDDI bridges to reuse IDUDs as ODUDs.

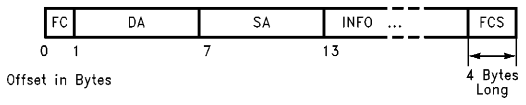
3.3.2 Received Frame Layout

Frames that are received from the network contain the FDDI Frame Control (FC), Destination Address (DA), Source Address (SA), the FDDI Info field (i.e., packet data) and the FDDI Frame Check Sequence (FCS, 4 bytes long). See *Figure 3-12* for an illustration of the received frame format.



TL/F/12304–13

FIGURE 3-11. Receive Data Structure



TL/F/12304–14

FIGURE 3-12. Received Frame Layout

3.3.3 The Input Data Unit Descriptor (IDUD)

The SI produces IDUD Descriptors and the host consumes them. Each IDUD Descriptor refers to part of a frame (possibly an entire frame). IDUD Descriptors that describe a single frame are grouped together via First and Last bits. The last IDUD of an Indicate Object (one or more IDUD Descriptors grouped together to completely describe a received frame) contains status for the entire frame. See *Figure 3-13* for a picture of the Input Data Unit Descriptor. A brief explanation of the fields in the IDUD Descriptor follows. The databook should be examined for a full explanation of these fields.

- The IS, Indicate Status, field (4 bits) reports the status of the received frame. This field should be queried to determine if the received frame is suitable for processing.
- The FRA, Frame Attributes, field (4 bits) describes the terminating condition and address matching flags (Destination or Source Address equal to MAC address). This field should be queried to determine if the received frame is suitable for processing.
- The FRS, Frame Status, field (8 bits) contains the FDDI E, A, and C Indicators (Error Detected, Address Recognized and Frame Copied) and two flags that indicate if the FCS on the frame agrees with frame data and if the frame is well formed. This field should be queried to determine if the received frame is suitable for processing.
- The VC (Valid Copy) field (1 bit) records how the SI signaled the BMAC device with regard to copying the frame off of the network. If VC is false then the BMAC device will increment its Frames Not Copied counter.
- The CNT (Count) field (13 bits) holds the length of the part of the received frame data (Input Data Unit (IDU)) to which the IDUD refers.
- The F-L, First-Last, fields (2 bits) are used to define groupings of one or more descriptors.
- The LOC (Location) field (28 bits) holds the address of the part of the received frame data (IDU) to which the IDUD refers.

3.3.4 The Pool Space Descriptor (PSP)

For the SI to produce received frames it must be told where, in memory, the incoming frame data can be stored. To declare receive buffers the host must produce Pool Space Descriptors (PSP) for the SI to consume. See *Figure 3-14* for a picture of the PSP Descriptor.

The current implementation of the SI is heavily page oriented. It does not currently use the PSP's CNT field at all (it does not even fetch the first word of the PSP Descriptor). It uses the 28-bit address of the LOC field to define the start of a receive buffer and the next 4 kb page boundary as the end of a receive buffer. Also, the buffer memory defined by the LOC field must be aligned on a "burst boundary"; either 16 or 32 bytes. When dealing with frame data the SI accesses memory in "bursts" of bus usage. It acquires the bus and then reads or writes in 4 or 8 word (16 or 32 byte) chunks. The SI can be configured to use only 4 word bursts or both 4 and 8 word bursts.

31 30	29 28	27 24	23 16	15	14 13	12 0	
IS		FRA	FRS	VC	RES	CNT	**Word 0**
F-L	RES	LOC					**Word 1**

FIGURE 3-13. Input Data Unit Descriptor (IDUD)

31 30	29 28	27 13	12 0	
RES			CNT	**Word 0**
F-L	RES	LOC		**Word 1**

FIGURE 3-14. Pool Space Descriptor (PSP)

How are received frames stored in memory? One frame per page? Multiple frames packed in a page?

The SI has two modes of operation that affect the way that incoming frames are stored: "normal" mode and "Frame per Page" mode (FPP). When Frame per Page mode is enabled the SI will only put one frame in a given page; though a single frame may split up into fragments that are stored in multiple pages (i.e., frames greater than the page size). When Frame per Page mode is not enabled the SI will pack as many frames as possible into each page. Currently Frame per Page mode affects all Receive Channels. See *Figure 3-15* for a picture showing the difference between the two modes.

The SI writes frame data out in bursts of 4 or 8 words (16 or 32 bytes). When transferring a new frame to memory the SI always starts writing at a burst boundary. The first word is filled with four copies of the FDDI Frame Control and the Destination Address starts with the second word. The LOC field of the first IDUD points to the last byte in the first word. This has the effect of starting the FDDI Info field with a 32-bit word alignment. This alignment is important (for performance reasons) for applications that want to avoid copying frame data as it allows the host to use native accesses (i.e., 16 and 32 bit integers) to parse the packet contents. When a frame is split across multiple pages the SI begins the next IDU on a burst boundary. Therefore, the bottom two bits of the IDUD.LOC field will either contain 3 or 0, but never 1 or 2 (this makes an IDUD with 0x00000001 or 0x00000002 in the LOC field a good candidate for marking an IDUD queue slot as a "null descriptor"). A "null descriptor" is a descriptor value that signifies that a queue slot is empty. This concept is presented more fully in the following discussion about SI queues. See *Figure 3-16* for an illustration of IDU alignments.

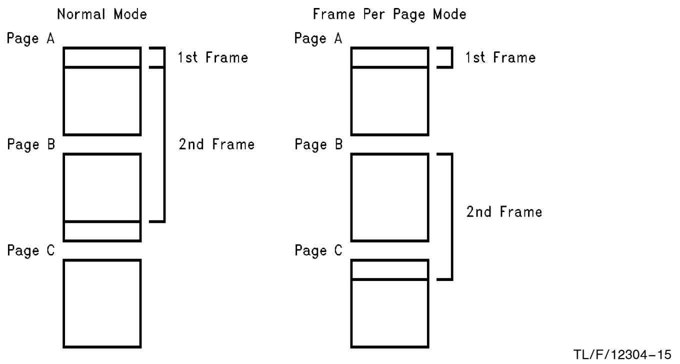


FIGURE 3-15. SI Memory Storage Modes

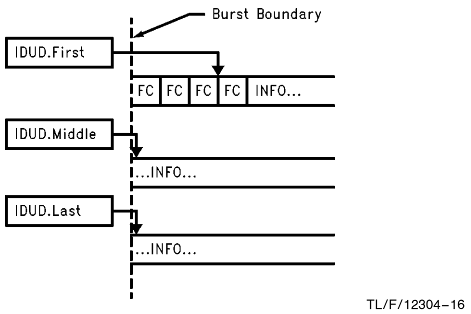


FIGURE 3-16. IDU Alignments

3.4 The SI Queues in Depth

This area of the document focuses in on the operational details of the SI queues. Although there are four types of descriptor queues, when examining the operation of the queues we need only consider two basic types:

1. queues for which the host produces descriptors and from which the SI consumes descriptors (REQ and PSP queues) and
2. queues from which the host consumes descriptors and for which the SI produces descriptors (CNF and IDUD queues).

In the discussion that follows the REQ and PSP queues are treated as equivalent and the IDUD and CNF queues are treated as equivalent. The steps that the host performs to produce or consume descriptors is the same.

How are the SI queues organized?

An SI queue is a 1 kb or 4 kb block of memory divided up into 8 byte (descriptor sized) queue slots. The size of the queues is defined at SI initialization time and is specified via the Mode Register (SIMR0) Control Bus Register. A 1 kb queue has 128 queue slots and a 4 kb queue has 512 queue slots. This block of memory is reused in a circular fashion (i.e., like a ring buffer); such that as descriptors are being processed, the lowest addressed descriptor is considered to logically follow after the highest addressed descriptor. This behavior is termed a ''queue wrap'' in this document. The point at which the queue wrap occurs is fixed at either a 1 kb or 4 kb boundary. All of the active SI queues are the same size (either 1 kb or 4 kb). Also, since boundaries are used to define the ''wrap point'' all SI queues must be aligned on either 1 kb or 4 kb boundaries. See *Figure 3-17* for illustrations of a queue before and after a queue wrap.

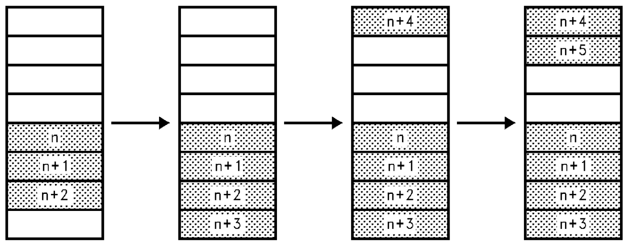
The SI uses two internal registers to represent the current progress of a queue: a Queue Pointer that indicates where the SI ''is'' in the queue and a Queue Limit that indicates how far the SI can advance. The consumption and production of descriptors is totally controlled via the Queue Pointer and Queue Limit values. The SI does not use ''ownership'' bits. See *Figure 3-18* for an illustration of how Queue Pointers and Queue Limits are logically used in queues.

The SI maintains a 28-bit Queue Pointer (internal Pointer RAM Register) for each queue. This pointer indicates the next queue slot that the SI will access. Also, because all queues are aligned on 1 or 4 kb boundaries, the upper most bits in the pointer (16 bits for 1 kb queues or 14 bits for 4 kb queues) define the base location of the queue. The host must load the Queue Pointer during initialization to tell the SI where the queue is located. Thereafter it is totally maintained by the SI.

The SI uses a 9-bit Queue Limit (internal Limit RAM Register) for each queue. The Queue Limit defines which queue slots are available for either reading or writing by limiting how far the SI can advance in the queue. A Queue Limit value can be thought of as an offset from the base address of the queue in units of 8 byte descriptors. Queue limits are totally maintained by the host. The SI only looks at Queue Limit values to determine when to stop queue processing. *Figure 3-19* shows how Pointer RAM Register and Limit RAM Register data types are related.

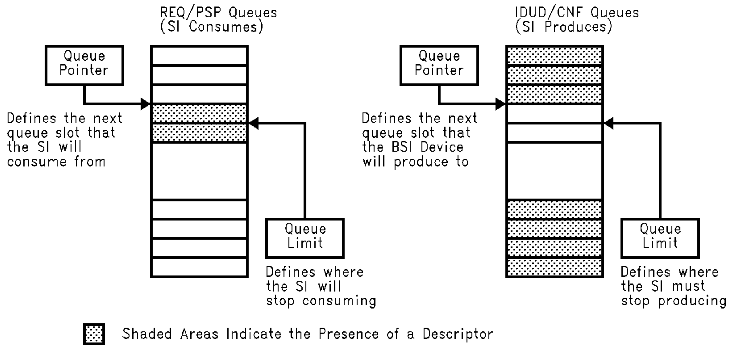
How does the SI ''know'' when to examine the REQ and PSP queues for new descriptors to consume?

Rather than implementing a special ''queue ready'' command or making the system interface repeatedly check for the presence of a descriptor (i.e., an ''ownership'' bit), the designers of the SI incorporated special logic in the portion of the chip that deals with updates to Limit RAM Registers. The act of updating a REQ or PSP Queue Limit is detected and interpreted as a signal to begin consuming descriptors from the queue. Thus the host may simply queue descriptors without having to be concerned about whether the Channel is active or inactive. If the host is queuing a group of descriptors it may, for the sake of efficiency, update the queue's Queue Limit once for all of the descriptors. See *Figure 3-20* for a depiction of the actions necessary to queue a single REQ or PSP Descriptor.



TL/F/12304–17

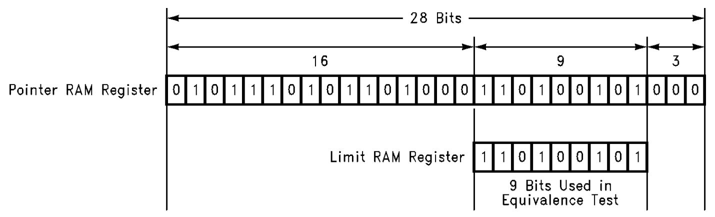
FIGURE 3-17. Queue Wrap at 1 kb or 4 kb Boundary (Snapshots of a Queue Over Time)



REQ/PSP Queues (SI Consumes)
IDUD/CNF Queues (SI Produces)
Queue Pointer
Defines the next queue slot that the SI will consume from
Queue Limit
Defines where the SI will stop consuming
Queue Pointer
Defines the next queue slot that the BSI Device will produce to
Queue Limit
Defines where the SI must stop producing
Shaded Areas Indicate the Presence of a Descriptor

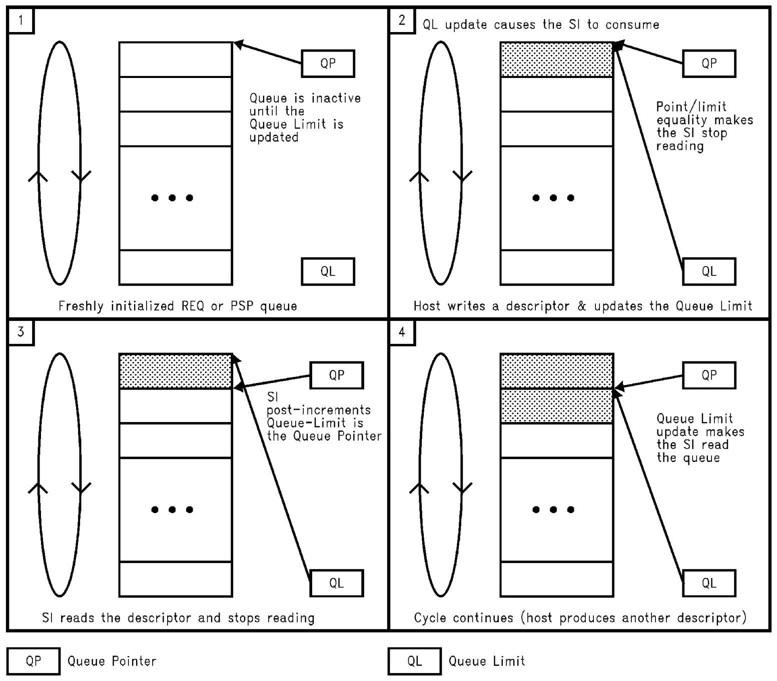
TL/F/12304–18

FIGURE 3-18. Queue Pointers and Queue Limits



TL/F/12304–19

FIGURE 3-19. Comparison of Pointer RAM and Limit RAM Registers (The Register Values Shown Are Considered to be Equivalent)



1
QP
Queue is inactive until the Queue Limit is updated
QL
Freshly initialized REQ or PSP queue
2
QL update causes the SI to consume
QP
Point/limit equality makes the SI stop reading
QL
Host writes a descriptor & updates the Queue Limit
3
QP
SI post-increments Queue-Limit is the Queue Pointer
QL
SI reads the descriptor and stops reading
4
QP
Queue Limit update makes the SI read the queue
QL
Cycle continues (host produces another descriptor)
QP Queue Pointer
QL Queue Limit

TL/F/12304–20

FIGURE 3-20. REQ or PSP Descriptor Queuing Example

How does the SI "know" when to stop consuming REQ and PSP Descriptors?

As each descriptor is fetched the SI compares the Queue Pointer with the Queue Limit. When these two registers are equivalent the descriptor currently being fetched is consumed and queue processing is stopped and remains inactive until the Queue Limit is updated. If the Queue Limit is updated before the above mentioned equivalence is detected the SI will automatically continue consuming descriptors from the queue. This mechanism is very robust and allows the host to dynamically produce transmit requests without concern for race conditions between the host and SI.

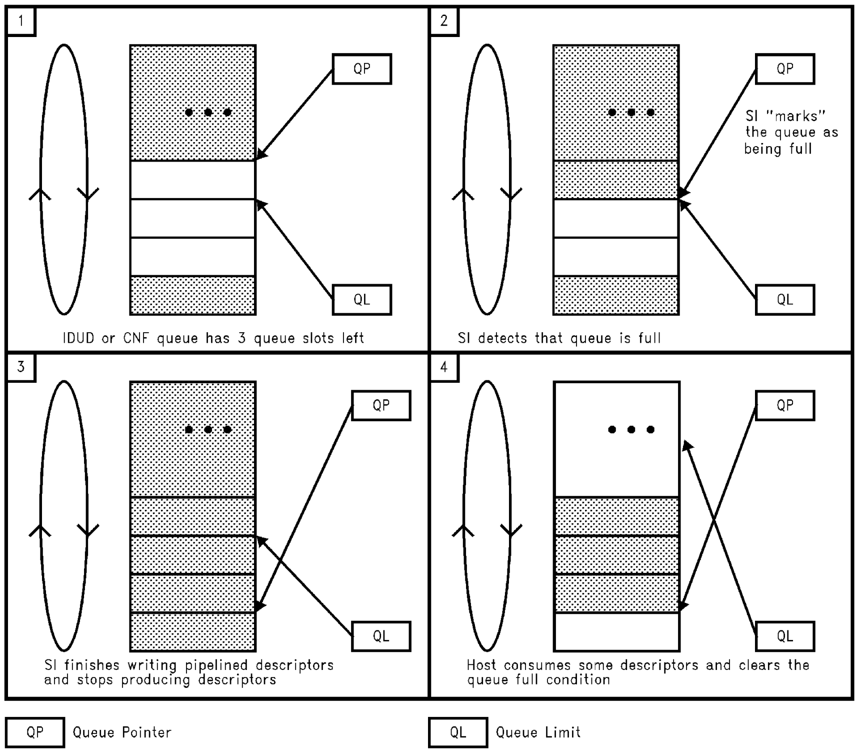
What causes the SI to produce descriptors on CNF and IDUD queues?

The CNF and IDUD queues are status queues. The SI produces CNF or IDUD Descriptors to report status information back to the host. CNF Descriptors are produced when a transmission request has completed or, optionally, a network transmission opportunity has ended before the entire Request Object has completed. (If a confirmation class of "none" is specified on a given Request Object then CNF Descriptors will only be produced when an error occurs while processing that Request Object.) IDUD Descriptors are produced every time a frame is received from the network. Multiple IDUD Descriptors are produced to describe a frame that is split across 4 kb pages.

How does the SI "know" when to stop producing CNF and IDUD Descriptors so that previously produced descriptors are not overwritten before the host has consumed them?

The SI compares the queue's Queue Pointer and Queue Limit after producing a CNF or IDUD Descriptor (the SI post-increments the Queue Pointer after producing a descriptor). When these two registers are found to be equivalent then the SI "marks" the queue as full, by asserting a bit in one of the directly accessible Control Bus Registers (No Space Attention Register (NSAR)), and stops Channel operation. The user may configure the SI to generate an interrupt when this bit is asserted. The Channel will remain stopped until the host has cleared the bit. This mechanism keeps the SI from overwriting previously produced descriptors before the host has consumed them. The host should update the queue's Queue Limit (to denote which queue slots may be overwritten) before clearing the bit.

Due to internal "pipelining" of descriptors within the SI, it is possible for two additional descriptors to be produced after pointer/limit equality has been detected. This behavior must be considered by the host when calculating Queue Limit values. The Queue Limit should be set such that it refers to the "next to the last" writable queue slot. Also, when responding to a full queue condition the host must ensure that the Queue Limit is set to be logically "ahead" of the Queue Pointer before clearing the bit that signals the queue full condition. See *Figure 3-21* for an illustration of the queue full condition.


1
QP
QL
IDUD or CNF queue has 3 queue slots left
2
QP
SI "marks" the queue as being full
QL
SI detects that queue is full
3
QP
QL
SI finishes writing pipelined descriptors and stops producing descriptors
4
QP
QL
Host consumes some descriptors and clears the queue full condition
QP Queue Pointer
QL Queue Limit

TL/F/12304–21

FIGURE 3-21. IDUD or CNF Queue Full Condition

How does the host "know" when to start consuming IDUD or CNF Descriptors?

The host may either poll for new descriptors or respond to an interrupt signifying the arrival of a single or group of descriptors.

The SI can be configured to generate interrupts when producing a CNF Descriptor and when frames have been received (it can generate a single interrupt for a group of frames or an interrupt after each frame is received). The host may enable and disable specific interrupt causing attentions at any time.

The host may poll for new descriptors in two different ways. One way is for the host to repeatedly read the queue memory looking for a new descriptor to be written into memory. An alternate method is for the host to configure the SI to generate an attention, yet disable that specific attention from causing an actual interrupt. The host may then repeatedly read one of the directly accessible Control Bus Registers, waiting for the attention bit to become asserted.

How does the host "know" when to stop consuming IDUD or CNF Descriptors? How does the host recognize an "End of Queue" condition?

There are two different methods of dealing with this issue. The most direct way for the host to determine which descriptors it should consume is to read the value of the queue's Queue Pointer. The Queue Pointer is maintained by the SI and defines the 28-bit address at which the next descriptor will be written; thus the host may consume IDUD or CNF Descriptors until reaching this point in memory. Since the Queue Pointer is maintained by the SI as an internal Pointer RAM Register the host must initiate an operation that causes the SI to write out the Queue Pointer into a predefined memory location (termed the mailbox). However, this operation competes with other SI functions for access to the SI's ABus interface.

Another method is for the host to populate each empty queue slot with a specially marked descriptor referred to as a "null descriptor". When processing the queue the host can recognize the "End of Queue" condition by detecting the presence of a "null descriptor". In most environments the "null descriptor" method will be fastest.

What is a good bit pattern for denoting a "null descriptor"?

The value used to denote a null descriptor must be one that is impossible for the SI to produce and must be placed in the second word of the descriptor. Null descriptors are only needed for IDUD and CNF queues, since these are the kinds of queues that the host uses to consume descriptors. The SI writes descriptors out to memory using two distinct bus request/grant cycles, so the host should look for "nullness" in the second word (which is written out last). There is no single value that, in general, can be used for both IDUD and CNF queues. Some ideas follow.

The SI will never produce an IDUD Descriptor with a value of x′1′ or x′2′ in the least significant two bits of the LOC field. It will only produce IDUDs with a value of x′0′ or x′3′ for these two bits. It is recommended to use an IDUD with a LOC field value of x′000001′ or x′000002′ to denote a null descriptor on an IDUD queue. If the system environment precludes the use of page 0 (e.g., that's where the system's interrupt vectors are located), then the act of testing for "nullness" may be made more efficient by using an IDUD with binary zeroes in the second word as the null descriptor; which may permit a "Branch Not Zero" instruction to be used.

The choice of a null descriptor for CNF queues is, unfortunately, dependent upon the way that the SI is programmed. Here are two suggestions.

- When consistently using a confirmation class of None, Tend or Fend the F-L field may be used. In particular, a descriptor with both First and Last bits cleared could be used to denote a null descriptor on a CNF queue.
- If using a confirmation class of Tint or Fint the UID field could be used. This requires that the host never create a Request Object using a particular UID value (i.e., zero) that is being used to denote a null descriptor on a CNF queue.

The host produces REQ Descriptors when queuing a frame for transmission, but how does the host "know" when to produce a PSP Descriptor?

The host produces PSP Descriptors to declare where the SI can store incoming frames. So, the host should either attempt to maintain a certain number of receive buffers (by detecting when buffers are consumed by the SI) or respond to the attention bit (in one of the directly accessible Control Bus Registers) that signals the "Low Data Space" condition. This issue is discussed in greater detail in Section 7.1, Memory Issues.

How can the host tell when it should stop producing REQ or PSP Descriptors (to avoid overwriting descriptors not yet consumed by the SI)? What is the "Queue Full" condition for REQ and PSP queues?

The SI queues are circular, so it makes sense that the host should consider a REQ or PSP queue full when producing one more descriptor would cause the SI to treat the queue as empty. The SI recognizes the queue as empty when it detects pointer/limit equality while consuming a descriptor, so the host should recognize the "Queue Full" condition when the queue's Queue Pointer is logically one descriptor "ahead" of the queue's Queue Limit. The host should test for the "Queue Full" condition before queuing an additional REQ Descriptor.

If, for some reason, the host also wants to be able to determine if a REQ or PSP queue is empty by comparing the Queue Pointer and Queue Limit values the above test for a full queue will make a test for an empty queue impossible. In this case the host should recognize the "Queue Full" condition when the queue's Queue Pointer is logically two descriptors "ahead" of the queue's Queue Limit and recognize an empty queue when the Queue Pointer is logically one descriptor "ahead" of the Queue Limit. This is a non-typical requirement, since there are more intuitive ways for the host to determine when a REQ Queue is empty.

How does the host "know" which queue slot to access when either producing or consuming a descriptor?

The SI does not maintain pointers to tell the host where to read or write descriptors. The host must maintain these pointers by itself, but it is a simple task. When the SI is initialized the Queue Pointers are established with known values (typically the base address of the queue). The SI always advances sequentially in all queues; therefore the host should start with the initialized value and proceed sequentially from then on (taking care to handle queue wraps). Some example C macros for calculating the address of the next queue slot can be found in *Figure 3-22*.

The host needs to "know" the value of Queue Pointers to check for the "Queue Full" condition on REQ and PSP queues (to avoid overwriting descriptors before the SI consumes them). Does the host need to perform Pointer RAM Operations each time it produces a descriptor?

While directly reading Queue Pointers from the SI's internal Pointer RAM Registers is logically simple, it may be unacceptable (for performance reasons) for the host to frequently perform Pointer RAM Operations in a synchronous manner. (An alternate and efficient method is to asynchronously schedule Pointer RAM Operations and update the Software Queue Pointer when the read operation has completed.) In stead the host may infer the value of an SI's REQ or PSP Queue Pointer when consuming the complementary status queue. This requires that the host maintain a "Software Queue Pointer" (SQP) that is a lower bound approximation of the true value of the "Hardware Queue Pointer" (HQP, located in a register internal to the SI). Note that this exercise is not necessary for Queue Limits, since these are totally maintained by the host. The Software Queue Pointer indicates how far the Queue Limit can be advanced before overwriting descriptors that the SI has not yet consumed.

To maintain a REQ Software Queue Pointer (SQP) by inference the host must:

- use a confirmation class other than none (Tend works quite well for this purpose)
- initialize the SQP and SI's Queue Pointer with the same value (at power-up time)
- increment the REQ.UID field value for each REQ.Only or REQ.First Descriptor
- consume CNF Descriptors and use the CNF.UID field to see which REQ Descriptors have been consumed by the SI, incrementing the REQ queue's SQP when scanning through the REQ queue for matching UID values. See *Figure 3-23* for the pseudo-code of this step.

To maintain a PSP Software Queue Pointer (SQP) the host must:

- initialize the SQP and HQP with the same value (at power-up time)
- when consuming IDUD Descriptors increment the SQP when a page change is seen in the IDUs (input data buffers). See *Figure 3-24* for the pseudo-code of this step.

```
#define BSIQ_NEXT_1K(p) ((((p) + 8) & 0x3ff) | ((p) & 0x0ffffc00))
#define BSIQ_NEXT_4K(p) ((((p) + 8) & 0xfff) | ((p) & 0x0ffff000))
/* increment a Software Queue Pointer q_ptr is 32 bit variable */
q_ptr = BSIQ_NEXT_4K(q_ptr);
```

FIGURE 3-22. C Preprocessor Macros for Incrementing Queue Pointers

```
Get CNF Descriptor
WHILE CNF QUEUE NOT EMPTY
      Get REQ Descriptor (at SQP location)
      WHILE REQ.UID = CNF.UID
            <reclaim transmit request memory>
            increment REQ queue's SQP
            Get next REQ Descriptor (at SQP location)
      END WHILE
      Get next CNF Descriptor
END WHILE
```

FIGURE 3-23. Maintaining a SQP for a REQ Queue

```
Get IDUD Descriptor
<normal IDUD processing>
IF (IDUD.LOC & PAGE_MASK) not = Prev_Page
     <release page or produce new PSP>
     increment PSP queue's SQP
     Prev_Page = IDUD.LOC & PAGE_MASK
END IF
```

FIGURE 3-24. Maintaining a SQP for a PSP Queue

4.0 SENDING A FRAME

The steps for producing a frame for transmission are:

1. Make sure REQ queue isn't full
2. Build the ODU(s)
3. Build the ODU Descriptor(s)
4. Produce a REQ Descriptor
5. Update the REQ queue's Queue Limit

4.1 Make Sure REQ Queue Isn't Full

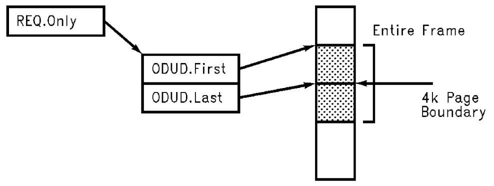
The host may produce transmit requests faster than the network can service them (i.e., a heavily loaded network), so the host must make sure that is doesn't overlay previously produced REQ Descriptors before the SI has consumed them (the SI queues are circular). This is accomplished by doing a comparison between the Queue Limit and Software Queue Pointer. See the discussion on SI queues in Section 3.4 for information on how to detect the queue full condition.

4.2 Build the ODU(s)

If the SI is directly attached to system memory this step may not be necessary, since the SI can directly access the same memory that the host uses for system dependent data structures (i.e., mbufs/mclusters). (It may still be necessary for the device driver to provide FDDI and/or LLC/SNAP encapsulation, which may be placed in an ODU that is separate from the rest of the frame data.) If the SI is not directly attached to system memory, then the host must allocate one or more buffers (none of which may span across a four kb page boundary) and populate the ODU(s) with frame data.

4.3 Build the ODU Descriptor(s)

Each frame data buffer (ODU) must be "described" to the SI with an ODU Descriptor (ODUD). The ODUDs must be placed contiguously (back to back) in memory within a 4 kb page. When dealing with system dependent data structures some care must be taken to avoid asking the SI to read across a 4 kb page boundary when fetching frame data. For example, if a particular host buffer does span across a 4 kb page boundary then two ODUDs must be used to describe that buffer. See *Figure 4-1* for picture of how to configure ODUDs to handle this situation.



TL/F/12304–22

FIGURE 4-1. Two ODUDs Needed for a Buffer that Crosses a 4k Page Boundary

4.4 Produce A REQ Descriptor

At this point there exists one or more buffers of frame data and an array of ODUD Descriptors that define the address and length of each buffer (termed ODUs in the datasheet). A REQ Descriptor must be created on a REQ queue that defines the address and frame count of the ODUD array. This completes the three level data structure. If the REQ Descriptor is marked as REQ.First or REQ.Only the host must configure the REQ.UID, REQ.CNFCLS, REQ.RQCLS and REQ.FC fields.

4.5 Update the REQ Queue's Queue Limit

The SI will not consume REQ Descriptors beyond the queue slot defined by the queue's Queue Limit. (Queue limits are updated via Limit RAM Operations (LMOP). LMOPs are described in Section 6.0, "Low Level Operations".) So the host must update the Queue Limit to denote how far in the queue the SI will progress. If the queue is quiescent, the act of updating the Queue Limit, located in the SI's internal Limit RAM, will "wake up" the Channel and cause the SI to start consuming REQ Descriptors. If the SI is already actively consuming REQ Descriptors from the queue, the newly queued REQ Descriptor will be automatically consumed.

5.0 RECEIVING A FRAME

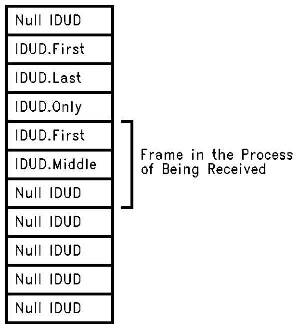
The steps for consuming a frame that has been received from the network are:

1. Collect the IDUD Descriptors that describe the next received frame
2. Check status fields in the last IDUD of those collected
3. Process the frame data
4. Reclaim Pool Space (receive data buffers)
5. Update the IDUD queue's Queue Limit

5.1 Collect the IDUD Descriptors that Describe the Next Received Frame

As a first step it is necessary to gather up all of the IDUD Descriptors that describe the next frame. This is necessary as the SI only includes status information in IDUDs with the "Last" bit set (IDUD.Only and IDUD.Last). As a general rule the host software should be capable of dealing with up to three IDUDs per frame. The host software should plan on handling three IDUDs per frame; even when operating the SI in "Frame per Page" mode. If buffers smaller than 4 kb are used (i.e., PSP.LOC points to an offset within a 4 kb page), then more than three IDUDs may be produced for a single frame.

It is possible for the host to detect an "end of queue" while collecting IDUDs. This can happen when the SI is in the process of receiving a frame and the host is asynchronously examining the queue. The host may either consider the queue as currently empty or wait for the frame reception to complete (note that IDUDs are produced after the corresponding part of frame data has been copied to memory). If the host considers the queue as empty in this situation and interrupts are used to trigger received frame processing, then the host should clear the attention only at the beginning of the Interrupt Service Routine (ISR) code (so that the frame reception in progress will not be missed). See *Figure 5-1* for a snapshot of an IDUD queue displaying this transient condition.



TL/F/12304–23

FIGURE 5-1. Frame Reception in Progress

5.2 Check Status Fields in the Last IDUD of Those Collected

The host must decide if the frame should be processed or "dropped". There are several status items in the IDUD that provide all of the information required to make this decision. These status indicators are only valid in an IDUD Descriptor with the Last bit set (IDUD.Last or IDUD.Only). The host should always check the following indicators.

- Indicate Status (IDUD.IS). This field must be examined to check for errors that may occur during frame reception.
- Valid Frame Check Sequence (VFCS). As the frame is being received a CRC calculation is done. If the calculated value doesn't agree with the FCS field of the frame, this bit in the IDUD will be zero to indicate a problem.
- Valid Data Length (VDL). This bit will be zero if a length problem has been detected (basically an odd number of FDDI symbols (kind of like 4 bit nibbles)).
- Terminating Condition (TC). This two bit field reports how the received frame ended. The host must verify that the frame ended with an FDDI Ending Delimiter. If the sending station isn't stripping the frame properly or if there is a lot of noise on one of the links in the ring the frame may be partially stripped. This fact is only recorded in this field (just checking the IDUD.IS field isn't enough).

5.3 Process the Frame Data

Clearly this step is very system dependent. On some systems this may only involve the creation of a few mbufs and a scheduling of the protocol stack processing. On others the frame may need to be actively copied into different data structures.

5.4 Reclaim Pool Space (receive data buffers)

Again this is a fairly system dependent step. One thing is true in all cases, though; the host can be sure that the SI has finished using a receive buffer when an IDUD is consumed that refers to a new buffer. (Assuming that the host is processing the IDUD queue in a sequential manner.) The process of determining when the host has finished using a receive buffer is system dependent. Many systems use a mechanism in which frames are passed to upper level protocols and may be relinquished in an arbitrary order. Section 7.4, Reclaiming Receive Buffer Memory, presents a mechanism that deals with this situation by using buffer reference counts.

5.5 Update the IDUD Queue's Queue Limit

The SI will consider an IDUD queue as full when queue pointer/limit equality is detected. Therefore, the host should update the Queue Limit (in the SI's internal Limit RAM) to grant additional status space (i.e., queue slots) to the SI. This need not be done for each frame (it can be done as infrequently as almost once per queue wrap). For performance reasons, it is advantageous to delay this operation until many frames have been consumed by the host. An example of delaying Queue Limit updates can be found in the nf__ret__idud( ) routine of the SI Primitives. National Semiconductor makes the source code to these example routines available to National's FDDI customers.

6.0 LOW LEVEL OPERATIONS

At the lowest, most primitive level the host must be able to perform read and write operations to the SI's Control Bus Registers and memory that is accessible by both the host and the SI. These operations are highly system dependent. For example, the Control Bus Registers might be memory mapped in one implementation and reached via some form of programmed I/O in another system.

At the next higher level (where the host software has control) there are several low level operations.

- Pointer RAM Operations (PTOP)
- Limit RAM Operations (LMOP)
- Mailbox definition
- Reading the SI's silicon revision number
- Updating Conditional Control Bus Registers
- Updating "Stop Mode Only" Control Bus Registers

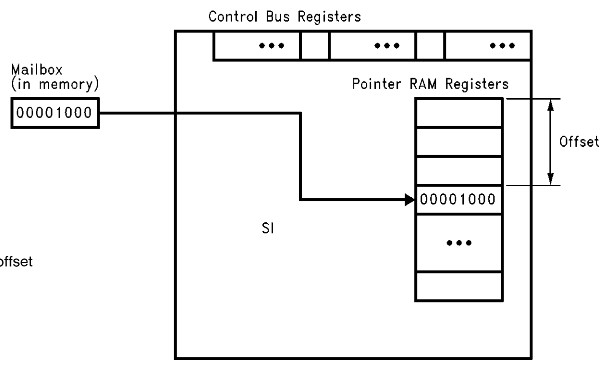
6.1 Pointer RAM Operations (PTOP)

All Queue Pointers are maintained by the SI in Pointer RAM Registers that are internal to the chip. These registers are not directly accessible via the SI's Control Bus. Instead the host performs a Pointer RAM Operation (PTOP) to load or store one of the Pointer RAM Registers. The SI uses a special memory location (a 32-bit word termed the mailbox) to load and store 28-bit Pointer RAM Registers. The SI reads the mailbox when loading a Pointer RAM Register and writes to the mailbox when storing a Pointer RAM Register. The steps required to perform a PTOP are:

1. make sure that the SAR.PTOP bit is asserted (SI able to do a PTOP)
2. if loading a register from memory, put the new value in the mailbox
3. configure the Pointer Control and Address Register (PCAR) (defines load/store and which Pointer RAM Register)
4. clear the PTOP bit in the Service Attention Register (SAR.PTOP) (causes the SI to do the PTOP)
5. if storing a register to memory, when the SAR.PTOP bit becomes asserted get the value out of the mailbox.

It is not necessary to "baby-sit" the SI during a Pointer RAM Operation. This behavior can be used to reduce the cost of reading the Queue Pointers of REQ and PSP queues. It is possible to "schedule" a Pointer RAM "Read" Operation, go perform other work and detect the completion of the operation at a later time. Also, the act of detecting the completion of the PTOP can be made extremely efficient by preloading the mailbox location with a value of x′00000003′ (an impossible value for Pointer RAM Registers) prior to starting the PTOP. This scheme avoids accessing the Service Attention Register (SAR); which, on most systems, is more expensive than accesses to shared memory. Seeing a value other than x′00000003′ in the mailbox signifies that the operation has completed.

See *Figures 6-1* and *6-2* for illustrations of the steps involved when loading a Pointer RAM Register from memory and when storing a Pointer RAM Register to memory.

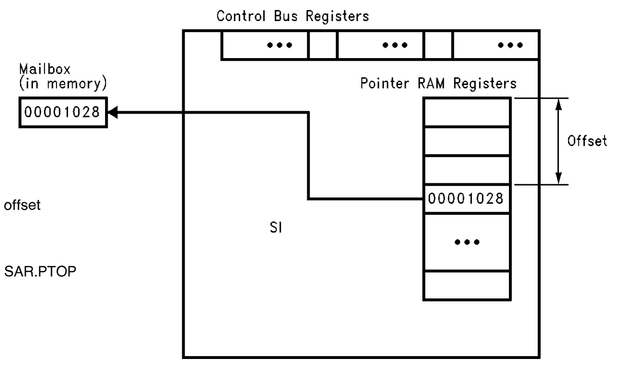


Steps:

1. Make sure SAR.PTOP is set
2. Place value in mailbox
3. Configure PCAR to define the offset and type of operation (write)
4. Clear SAR.PTOP

TL/F/12304–24

FIGURE 6-1. Pointer RAM Operation (“Write”) (Loading Register from Memory)



Steps:

1. Make sure SAR.PTOP is set
2. Configure PCAR to define the offset and type of operation (read)
3. Clear SAR.PTOP
4. Get value out of mailbox *after* SAR.PTOP becomes set again

TL/F/12304–25

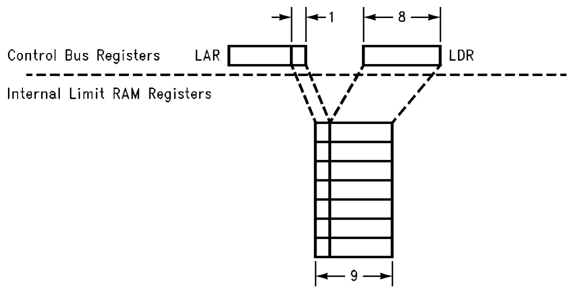
FIGURE 6-2. Pointer RAM Operation (“Read”) (Storing Register Contents to Memory

6.2 Limit RAM Operations (LMOP)

The host maintains all Queue Limit values by updating the Limit RAM Registers that are internal to the SI. The host must perform a Limit RAM Operation (LMOP) to load or store one of the Limit RAM Registers. Limit RAM Operations are accomplished solely via the directly accessible Control Bus Registers. Two Control Bus Registers (8-bit) are needed to hold a Limit RAM Register value (9-bit).The steps required to perform an LMOP are:

1. make sure that the SAR.LMOP bit is asserted (SI able to do an LMOP)
2. configure the LAR (defines load/store, which Limit RAM Register and one bit of data)
3. if loading a register, configure the LDR (bottom 8 bits of the Limit RAM value)
4. clear the SAR.LMOP bit (causes the SI to do the LMOP)
5. if reading a register, poll for the SAR.LMOP bit to become asserted and examine the LAR and LDR Control Bus Registers to get the Limit RAM value.

The act of updating Limit RAM Registers is done quite frequently by the host thus this operation should be coded as efficiently as possible. Limit RAM updates can be done with four accesses of the Control Bus. See *Figure 6-3* for a depiction of the relationship between the LAR, LDR and Limit RAM Registers. See *Figures 6-4* and *6-5* for illustrations of the steps involved in reading and writing to Limit RAM Registers. The "cost" of Limit RAM Operations is discussed in Section 7.5, Performance Issues.

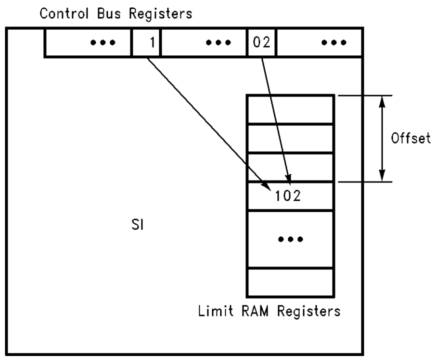


TL/F/12304–26

FIGURE 6-3. How LAR and LDR Create a 9-Bit Value

Steps:

1. Make sure SAR.LMOP is set
2. Place lower 8 bits in LDR
3. Configure LAR to define the offset, type of operation (write) and top bit of value
4. Clear SAR.LMOP

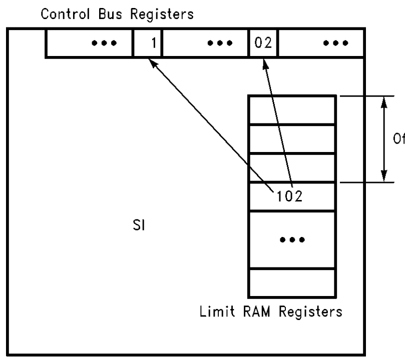


TL/F/12304–27

FIGURE 6-4. Limit RAM Operation ("Write")

Steps:

1. Make sure SAR.LMOP is set
2. Configure LAR to define the offset, type of operation (read)
3. Clear SAR.LMOP
4. After SAR.LMOP becomes set read LDR and LAR to get the value



TL/F/12304–28

FIGURE 6-5. Limit RAM Operation ("Read")

6.3 Reading the SI's Silicon Revision Number

All SIs contain a silicon revision number. This value can be obtained immediately after resetting the SI by:

- clearing the Pointer Control and Address Register (PCAR) Control Bus Register. This setp is important because the PCAR is not initialized to known values at reset time.
- reading the Mailbox Address Register (MBAR) Control Bus Register four times.

A 32-bit revision number is returned in most significant byte order. This operation is typically done as part of the initialization of the SI. See Table 6-1 for a list of defined silicon revision numbers and how they correspond to the revisions of the SI.

TABLE 6-1. Correlation of Silicon Revision Numbers to SI Versions

Silicon Revision Number	SI Release Level
0x00000004	BSI-1 Rev-B
0x0000004C	BSI-2 Rev B
0x00000054	MACSI Rev C
0x00000058	MACSI Rev D

6.4 Mailbox Definition

The mailbox location must be defined before any Pointer RAM Operations can be performed. The mailbox address is defined by:

- clearing the Pointer RAM Control and Address Register (PCAR) Control Bus Register. This step is important because the PCAR is not initialized to known values at reset time.
- writing the Mailbox Address Register (MBAR) Control Bus Register four times to define a 28-bit address (most significant byte first).

This operation is typically done as part of the initialization of the SI.

6.5 Updating Conditional Control Bus Registers

Some of the directly accessible Control Bus Registers are Conditional Registers that are used to report events or conditions to the host (Attention Registers). These registers may be read and written like regular Control Bus Registers except that some extra logic is included to make sure that the events are not missed. There is one Compare Register on the SI (SICMP) that is shared by all SI Conditional Registers. Read operations for non-Conditional Registers do not affect the contents of the Compare Register. When a Conditional Register is read the contents are also cached in the Compare Register. When a Conditional Register is written the original contents of the register are compared bit by bit with the contents of the Compare Register and only those bits that are the same are updated with the new values. If any of the bits don't "compare" then the CWI attention bit is asserted in the State Attention Register (STAR).

Why are Conditional Registers necessary?

The host and SI are operating independently and asynchronously. An event may occur (i.e., a frame arrival) between the time that the host has read an Attention Register and the time that the host later writes back to the same Attention Register (e.g., to clear an attention bit). Without Conditional Registers it is possible for some events to be missed.

The National Semiconductor FDDI chip set uses Conditional Registers to solve this problem. There are other ways of dealing with this issue, but Conditional Registers are used because they make it easier to test software and hardware. In many cases, the attention bits may be manually asserted by the host (perhaps a debugger) to exercise the Interrupt Service Routine (ISR) logic of the host software. Explicitly setting a bit in a Conditional Register is much often much easier than attempting to recreate the circumstances that would cause the SI to generate a particular attention.

During normal operation, the host should:

- read a Conditional Register at the start of Interrupt Service Routine (ISR) logic
- process the events
- clear the processed events by:

1. loading the Compare Register with the original value and
2. updating the Conditional Register with the processed event bits cleared.

— or —

- read a Conditional Register at the start of ISR logic
- individually clear each event by:

1. reading the Conditional Register,
2. masking the bit(s) associated with the event and
3. writing the modified value back to the Conditional Register (without any intervening accesses of other Conditional Registers).

— or —

- read a Conditional Register at the start of ISR logic to determine what events have occurred
- clear attention bits by:

1. loading the Compare Register with a bit pattern that defines the bits to be cleared (1's for those bits that will be zeroed),
2. write all zeros to the Conditional Register and, optionally,
3. check the CWI bit in the State Attention Register (STAR) to see if any other events need to be processed.

6.6 Updating "Stop Mode Only" Control Bus Registers

Several of the Control Bus Registers of the SI may only be updated by the host when the Indicate State Machine is in "stop mode". This restriction exists because the configuration of these registers affects indicate operation for all Receive Channels. These registers are the Indicate Mode Register (IMR), the Indicate Threshold Register (ITR) and the Indicate Header Length Register (IHLR).

7.0 IMPLEMENTATION ISSUES

This section of the document discusses some typical issues that need to be addressed when designing a network interface that uses the SI.

7.1 Memory Issues

The SI was designed to "fit" well in a wide variety of implementations. As such the way that the memory used by the SI is allocated and released is very system dependent. On systems where the SI directly accesses system memory, the network interface software may be able to employ the same memory management subsystem used by the upper layer protocols (e.g., mbufs/mclusters); or an entire, additional memory management subsystem may need to be developed.

There are two cycles of memory usage that correspond to the flow of FDDI frame data. On the transmit side, memory is allocated before transmission and released after transmission. On the receive side, memory is allocated before frame reception and released after the received data has been processed.

What items must memory management support? What alignment, if any, is required for each item?

The host must supply memory management support for descriptor queues (REQ, CNF, IDUD and PSP), receive buffers (Pool Space), transmit data chunks (Output Data Units (ODU)) and ODUD Descriptors.

The memory occupied by descriptors queues must be aligned on 1 kb or 4 kb page boundaries (depending upon the size of the queues). Any memory beyond the 1 kb or 4 kb page boundary is ignored by the SI. The memory used to contain an array of ODUD Descriptors must be aligned on an 8 byte boundary and may not cross a 4 kb page boundary (since the address counter will "wrap" at a 4 kb page boundary). The memory used to store in-coming frame data (Pool Space) must begin on an SI burst boundary (16 or 32 bytes) and continue to a 4 kb page boundary. If the buffer spans across a 4 kb page boundary the SI will ignore that portion of the buffer after the boundary. If the buffer, as described in a Pool Space Descriptor (PSP), does not begin on a burst boundary the SI, currently, will not operate correctly.

The memory used to contain a chunk of transmit data (ODU) need not follow any special alignment, but it may not cross a 4 kb page boundary. If the SI is presented with an ODU that spans a 4 kb page boundary, the data will not be transmitted as intended (since the address counter will "wrap" at a 4 kb page boundary). A buffer that crosses one or more 4 kb page boundaries may be successfully described to the SI using multiple ODUDs.

If a special memory management subsystem must be developed for the SI, what is the simplest scheme?

A specially developed memory management system can be as complicated or as simple as desired. An extremely simple, yet workable scheme would allocate all memory in 4 kb pages (aligned on 4 kb page boundaries) for all purposes. Small, 1 kb, queues would be statically packed into 4 kb pages and, of course, 4 kb queues would occupy an entire page. ODUD Descriptors could be stuffed into the first 16 bytes of a page also used to hold a fragment of transmit data (See *Figure 7-1*) or serially allocated within one or two statically allocated pages.

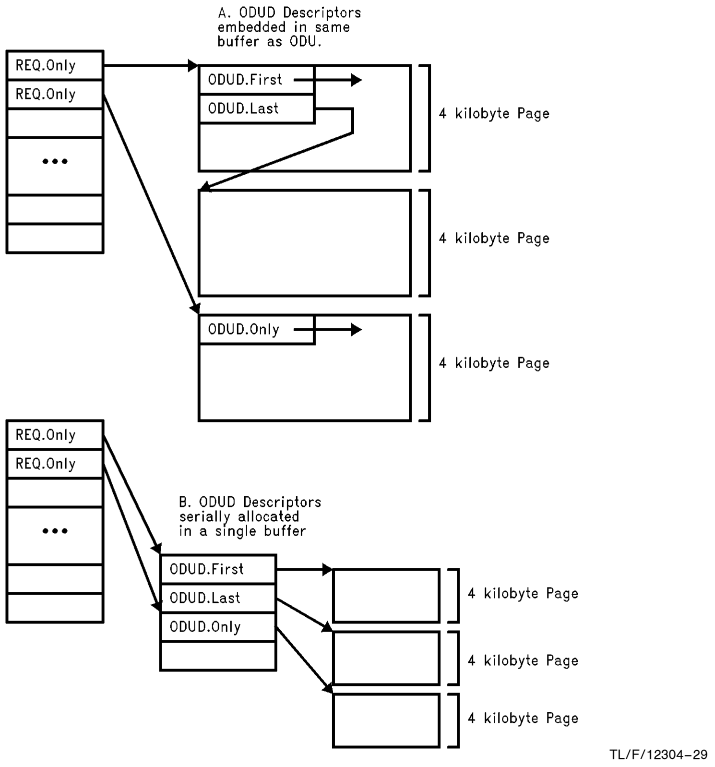


FIGURE 7-1. Example ODUD Allocation Methods

7.2 Reclaiming Transmit Data Structure Memory

The SI consumes transmit requests, but cannot complete the buffer usage cycle by releasing the memory occupied by each transmit request data structure (i.e., REQ and ODUD Descriptors, frame data pieces). The host must explicitly reclaim this memory after the SI has finished transmitting the frames. There are at least two methods of determining which transmit requests have been consumed by the SI and are thus available for space reclamation.

The simplest method is to examine the REQ queue's Queue Pointer value (Pointer RAM Register). The host then scans forward through the REQ queue (until reaching the Queue Pointer's memory location), parses the transmit data structures and releases those memory resources held by each transmit request. This is simple and straightforward. Performing synchronous PTOPs may not be suitable on some systems, however, due to the system dependent nature of PTOP performance. The cost of PTOPs can be minimized by asynchronously ‘‘scheduling’’ a PTOP and delaying the reclamation process until after the PTOP has completed (see Section 6.1, ‘‘Pointer RAM Operations’’ for more information).

Another method is to use a confirmation class (REQ.CNFCLS) of ‘‘Tend’’ (Transmit-End) and increment the REQ.UID field for each REQ Descriptor produced with the REQ.First bit set (i.e., REQ.First, REQ.Only). This causes the SI to produce a CNF Descriptor (with a matching UID value) for each consumed Request Object. The UID field of the CNF can be used to reclaim all of those REQ Descriptors with a matching UID value. This method is somewhat more complex than the first method, but it performs consistently well in many different system environments.

7.3 Reclaiming Receive Buffer Memory

The exact mechanism used to manage receive buffer memory is highly system dependent. However, many of the different possible implementations fall into two categories: those implementations that do copy frame data and those implementations that do not copy frame data.

7.3.1 Receive Buffer Management when Copying Frame Data

When the SI data structures are completely distinct from the system data structure the host must explicitly copy frame data from the SI addressable memory to the host addressable memory. Under these circumstances (and assuming that the host consumes IDUDs sequentially) the task of reclaiming receive buffer space is trivial. It is simply a matter of detecting that the SI is using a new receive buffer and reusing (e.g., returning to a free list or requeuing on a PSP queue) the ‘‘old’’ buffer (after copying out frame data, of course). See *Figure 7-2* for the pseudo-code required to implement this simple algorithm.

```
/* at Channel initialization time */
previous_page = <first page on the PSP
queue>
/* when processing IDUDs, after copying
frame data */
current_page = IDUD.LOC AND 0x0ffff000
if previous_page not = current_page then
    <reuse previous_page>
    previous_page = current_page
endif
```

FIGURE 7-2. Simple Receive Buffer Reclamation when Copying Frame Data

7.3.2 Receive Buffer Management when Not Copying Frame Data

When the SI directly shares its receive buffers with the host system, significant performance advantages may be gained by avoiding the copying of frame data. In such an environment the software that "drives" the SI will typically maintain a pool of buffers that the SI may transfer frame data to and from which frame data will be "loaned" to upper level protocols. The upper level protocols later invoke a registered routine to release the frame memory, but may release the frames in a different order from which they were presented.

If "Frame-per-Page" (FPP) mode is used, then the receive buffer(s) occupied by the frame may be immediately reused. However, if FPP is not used there may be multiple frames resident within a single receive buffer. A simple mechanism for this environment uses reference counts to keep track of the number of "active" frames of a given receive buffer. In a nutshell, a buffer's reference count is

1. Set to 1 when the buffer is queued on a PSP queue,
2. Incremented for each frame "loaned out" that includes a reference to that buffer,
3. Decremented by 1 when it is recognized that the SI is done using that page (see the pseudo-code in *Figure 7-2*),
4. Decremented by 1 for each release call received from upper level protocols that includes a reference to that buffer and
5. Finally reused when the reference count equals 0.

7.4 Synchronization Issues

The host and SI produce and consume descriptors independently and asynchronously from each other, so it is important for the designer of SI interface software to understand which areas need special attention to avoid race conditions between the SI and the host.

The host need not be concerned about race conditions when producing a REQ or PSP Descriptor. The SI handles the synchronization of Queue Limit updates and queue processing internally. The host may produce REQ or PSP Descriptors without having to double check to make sure that the queued frame is actually processed. The host doesn't have to "spoon feed" the SI as is required with many other network interface devices.

When the host is consuming descriptors and reading the Queue Pointer values from the SI's internal Pointer RAM Registers to detect the "End of Queue" condition, the host need not be concerned about race conditions. This method, while simple and straightforward, may not be suitable for many system environments. However, when the host is consuming descriptors and using "null descriptors" to detect the "End of Queue" condition some care must be taken to avoid a race condition. The SI uses single word (32-bit) bus cycles when producing a CNF or IDUD Descriptor; thus two distinct bus Request/Grant cycles are required for the SI to write out a descriptor into memory. This makes it theoretically possible for the host to fetch both descriptor words after the SI has written the first word, but before the SI has written the second word. This race condition can be avoided by:

1. using the second word of the CNF or IDUD Descriptor to denote a "null descriptor" and
2. adopting a "Probe and Fetch" mechanism for consuming descriptors (i.e., check for "End of Queue" by probing the second word of a descriptor and then, if it is not a null descriptor, read entire descriptor).

Also, when using null descriptors the software designer should scrutinize the way that the host interfaces with the SI addressable memory. For example, on the SI Evaluation Card the host has an 8-bit interface to SI addressable memory; which introduces a race condition with a much larger window than the one just described.

7.5 Performance Issues

The performance of an FDDI interface is clearly very important. There are several issues that need to be considered during the design of SI interface software. These issues are presented in the form of questions and answers.

How expensive are Control Bus read and write operations?

Control Bus accesses are internally synchronized with the ring clock (the SI spans two timing domains using both system and ring clocks). It takes 4 or 5 ring clock cycles (the FDDI ring clock runs at 12.5 MHz) for a Control Bus access to complete; thus each access takes approximately 320 ns to 400 ns.

How expensive are Limit RAM Operations?

In relative terms, Limit RAM Operations (LMOP) are expensive. LMOPs usually take a little less than 2 ms to complete (four Control Bus accesses = 400 ns × 4 = 1.6 ms; plus some host cycles). There are times when LMOPs must be done (i.e., queuing a Request Object) and there are times when LMOPs may be delayed (i.e., granting queue space on IDUD and CNF queues, queuing a PSP Descriptor); such that instead of performing many incremental LMOPs a single LMOP is done to reach the same value. Whenever possible, LMOPs should be delayed. A significant increase in performance can be realized by delaying LMOPs for CNF, IDUD and PSP queues. Also, there is absolutely no reason for the host to read Limit RAM Registers during normal operation, since the SI never changes Limit RAM values.

How expensive are Pointer RAM Operations?

The performance of Pointer RAM Operations (PTOP) is extremely dependent upon the bandwidth of the memory interface to which the SI is attached (i.e., dedicated RAM, connected to system bus) and the volume and pattern of frame traffic, because PTOPs use the same bus as frame data and PTOPs are given lower priority than frame data and descriptor transfers. A best case estimate is somewhat less than 2 ms (3 Control Bus accesses = 400 ns × 3 = 1.2 ms; plus an ABus access + some host cycles). The cost of reading Pointer RAM Registers can be minimized by scheduling asynchronous PTOPs (useful for reading REQ and PSP Queue Pointer values, but not appropriate for reading IDUD and CNF Queue Pointer values).

Why build multi-frame Request Objects?

The SI uses Request Objects to implement flow control at the token opportunity level. This is required for stations that send FDDI Synchronous frames. However, if the host generates only single frame Request Objects the station's effective bandwidth can suffer on heavily loaded FDDI networks (if the network can keep up with the host, then single frame Request Objects are fine). This is due to the way that transmission time is allocated among the stations on the network (timed token protocol); which favors those stations that send bursts of many frames at each service opportunity. Since today's typical implementations of protocol stacks produce single frame requests, it is important for the network interface software to bundle together frame transmission requests from upper level software into multi-frame Request Objects whenever possible (e.g., when the host produces frames faster than the ring can consume them).

In systems where the network device driver accepts single frame requests (most of the world), how can the network software present multi-frame Request Objects to the SI without any race conditions and without increasing the "end-to-end" delay?

The recommended algorithm takes advantage of the fact that multi-frame Request Objects are only important when the station is generating frames faster than the network can send them. If the network can keep up with the host, then single frames Request Objects are fine. This algorithm is similar, in principle, to a scheme used to avoid small TCP packets sizes proposed by J. Nagle. (J. Nagle, "Congestion Control in IP/TCP Internetworks", RFC 896, January 1984.) The basic idea is that when the REQ queue is empty, a REQ.Only Descriptor is generated (single frame request made immediately available for transmission). When the REQ queue is not empty a multi-frame Request Object is dynamically, incrementally built by producing suitable REQ Descriptors (i.e., REQ.First, REQ.Middle and REQ.Last Descriptors), but delaying the updating of the REQ queue's Queue Limit until a REQ.Last has been produced. To make this scheme work well the host needs to:

- Use the Transmit End (Tend) Confirmation Class. This is done so that the completion of a Request Object can be detected. Transmit Intermediate could be used as well, but it increases the complexity of the CNF processing logic and does not provide any additional information related to this task. Similarly, Fend and Fint could be used.
- Provide a method that notifies the driver that the CNF queue should be checked. This can be done by enabling interrupts for CNF Descriptor arrivals (CNF breakpoint) or, if a watchdog timer is already being used, check for CNF Descriptors when the watchdog timer expires. The method chosen here has an effect on the "end- to-end" delay.
- Maintain three variables per Transmit Channel:

1. a count of the number of Request Objects currently queued,
2. a Boolean variable that indicates the presence of an "open" Request Object (no REQ.Last yet) and
3. a count of the number of frames in the currently "open" Request Object.

A nice feature of this algorithm is that the overhead involved with frame transmission decreases as the frame traffic load increases. When the host gets ahead of the ring, fewer Limit RAM Operations are necessary to send the same number of frames and fewer CNF Descriptors are produced by the SI for the same number of frames. See *Figure 7-3* for the algorithm in pseudo-code.

The "end-to-end" delay is increased in only one circumstance. This occurs when an "open" Request Object exists on a REQ queue and the ring load drops enough to empty out the REQ queue. In this situation there will be frames awaiting transmission that the SI doesn't "know" about. The host must detect that the queue has emptied out and "close" the Request Object by producing a REQ Descriptor with a SIZE field set to zero and the Last bit set. Note that this requires that the test for the Queue Full condition be modified to reserve a position in the queue for this REQ.Last Descriptor.

```
IF Queued_Cnt = 0 THEN
     produce REQ.Only
     Open_Req = FALSE
     Queued_Cnt = 1
     <update Queue Limit>
ELSE
     IF Open_Req = FALSE THEN
          produce REQ.First
          Open_Req = TRUE
          Queued_Cnt = Queued_Cnt + 1
          Req_Size = 1
          <delay Queue Limit update>
     ELSE
          IF Req_Size < MAX_FRAMES_PER_REQ_OBJECT THEN
                produce REQ.Middle
                Req_Size = Req_Size + 1
                <delay Queue Limit update>
          ELSE
                produce REQ.Last
                Open_Req = FALSE
                <update Queue Limit>
          ENDIF
     ENDIF
ENDIF
```

CNFs are consumed by the host to determine when transmit requests have completed.

```
Get CNF Descriptor
WHILE CNF QUEUE NOT EMPTY
     Get REQ Descriptor (at SQP location)
     WHILE REQ.UID = CNF.UID
             <reclaim transmit request memory>
             increment REQ queue's SQP
             if REQ.Last or REQ.Only then
                    Queued_Cnt = Queued_Cnt - 1
                    if Queued_Cnt = 1 and Open_Req = TRUE then
                         produce null REQ.Last
                         <update Queue Limit>
                    endif
             Get next REQ Descriptor (at SQP location)
     END WHILE
     Get next CNF Descriptor
END WHILE
```

FIGURE 7-3. Dynamic Construction of Multi-Frame Request Objects

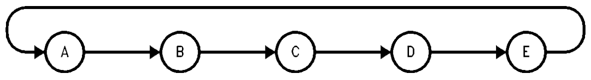
Under heavy load the number of interrupts coming out of some network devices can be a real burden for the host. What mechanisms does the SI have for minimizing the number of interrupts generated?

Many other network interface devices have a rather rigid set of rules for the generation of interrupts. Typically, the host gets a choice between no interrupts or an interrupt for each and every frame. Polling for events is fairly common on bridges and routers. The DP83200 FDDI chipset can be configured to work well in this environment as the host can choose exactly which events will generate an interrupt (perhaps none). The SI can be configured to "batch" events and generate interrupts at reasonable times. For example, multi-frame Request Objects can be structured to generate a single interrupt to signal that an entire group of frames has been transmitted and various interrupt batching rules can be applied to cause the SI to signal the arrival of groups of frames. See Table 7-1 for information on batching interrupts that signal the arrival of incoming frames (indicate breakpoints).

Breakpoint on Service Opportunity (BOS) generates the coarsest granularity of interrupts, Breakpoint on Burst (BOB) has a somewhat finer control and Breakpoint on Threshold (BOT) has the finest granularity (can be configured with a threshold of 1 to cause an interrupt for each frame received). See *Figure 7-4* for an illustration of how the SI indicate breakpoints work.

TABLE 7-1. Indicate Breakpoint (Interrupt) Rules

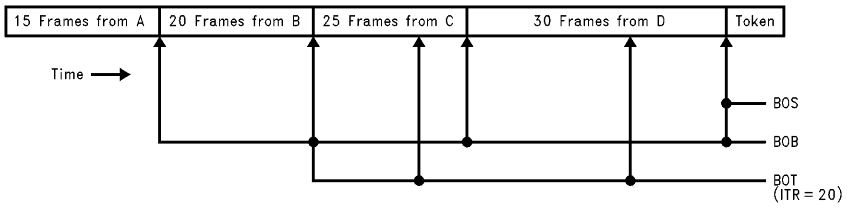
Breakpoint Rule	Description
Breakpoint on Service Opportunity (BOS)	An indicate breakpoint is generated when an end of a service opportunity, for other stations, has been detected (i.e., a token was received) and one or more frames have been received. This option must be enabled to cause the SI to generate breakpoints on Receive Channel 0 (dedicated to SMT/MAC frames).
Breakpoint on Burst (BOB)	An indicate breakpoint is generated when a change is detected in the Destination Address, Source Address, first four octets of the FDDI information field or Indicate Channel. The basic idea is that the SI generates a single interrupt for a "burst" of packets (i.e., a TCP window).
Breakpoint on Threshold (BOT)	An indicate breakpoint is generated after a predefined number of frames have been received. This option is designed to make BOB and BOS behave reasonably when a huge number of frames are received from a single station in a single service opportunity. The count of received frames used to compare against the threshold is reset after each service opportunity or burst.



TL/F/12304–30

1. Station A captures the token and sends 15 frames to station E
2. Station B captures the token and sends 20 frames to Station E
3. Station C captures the token and sends 25 frames to Station E
4. Station D captures the token and sends 30 frames to Station E
5. Station E receives 90 frames followed by the token!

Station E receives the following frames and indicate breakpoint attentions:



TL/F/12304–31

FIGURE 7-4. Example of the Various Indicate Breakpoints

7.6 Miscellaneous Issues

How are the station's addresses (MAC, broadcast and multi-cast) defined to the SI?

The SI does not recognize addresses, per se. The BMAC Device includes address matching logic for the MAC address, broadcast address and a range of multi-cast addresses. When an address match occurs, the BMAC Device asserts a signal on one of the pins that connect the SI and BMAC Devices.

What issues are involved when implementing FDDI Station Management (SMT) with the SI?

The majority of SMT can safely ignore the SI and concentrate on the BMAC and PLAYER Devices. There are two exceptions however.

FDDI SMT sends and receives frames as part of normal operation. It is necessary, therefore, to provide a facility for frame transmission and reception to SMT. This can be accomplished in basically two ways. The simplest way is to dedicate one of the Transmit Channels to SMT traffic. Since Receive Channel 0 is already dedicated to SMT, this provides SMT with guaranteed access to the FDDI ring. If the station cannot dedicate a Transmit Channel to SMT (e.g., has already set aside one Channel for synchronous frame transmission), then it will be necessary to multiplex (in software) SMT frame transmission requests with ''regular'' asynchronous LLC frame transmission requests.

The Ring Management (RMT) state machine of SMT includes a state (RMT:DIRECTED) that requires the station to send a special type of MAC frame called a ''Directed Beacon''. These frames must be sent out ''back to back'' for at least 370 ms. The BMAC Device, which does send out the ''normal'', Undirected Beacon frames, cannot (by itself) generate the Directed Beacon frames mandated by the SMT standard. So, the SI must be used to generate Directed Beacons. A User Information Note exists that describes how to configure the SI to send approximately 370 ms of Directed Beacon frames. Please refer to this note for detailed information.

What is Header/Info Splitting Mode and how does it work?

Header/Info splitting mode is intended for use by high performance protocol processing applications and by network protocol monitors. This mode of operation is selected via the Indicate Mode Register (IMR). Header/Info splitting mode ''cracks'' incoming frames into two or more pieces at a user defined offset into the frame (defined via the Indicate Header Length Register (IHLR)). The point at which frames are split can range from 4 words to 255 words with the FDDI Frame Control byte occupying an entire word; so the net effect is that the radix can range from the beginning of the FDDI Info field through 1004 bytes into the FDDI Info field.

When using this mode the Header portion is routed to Receive Channel 1's buffers, the Info portion is routed to Receive Channel 2's buffers and all of the IDUDs are produced on Receive Channel 1's IDUD queue. Thus the host declares two different sets of buffers for frame data reception (header buffers and info buffers) and processes the frames from a single IDUD queue.

Some network monitoring applications are only interested in examining frame headers (perhaps the first 60 bytes of the FDDI info). In this environment the host need only supply buffers to accept the header portion (using Receive Channel 1's PSP queue) and the Info portion of received frames will be discarded.

The SI uses two Indicate Breakpoint attention bits (in the Indicate Attention Register (IAR)) when operating in Header/Info splitting mode. A breakpoint may be generated for either Channel 1 or 2 depending upon the length of the frame being received. The host must accept either attention bit as a signal that frames have arrived and clear both bits when handling the interrupt. In other words, enable interrupts for both bits and clear both bits when clearing frame arrival events.

8.0 SERVICING INTERRUPTS

The SI provides facilities for selecting which events will generate an interrupt and a mechanism for determining which events are present after an interrupt has been raised.

8.1 Event Registers

The SI supports a two-level hierarchy of Event Registers; where the presence of attention signals in lower level Attention Registers is recorded in a single upper level Attention Register. Attention signals may be disabled at either of the two levels. Events may only be cleared by resetting the appropriate bits in the lower level Attention Registers.

The upper level Attention Register is called the Master Attention Register (MAR). It contains five attention bits that indicate the presence or absence of any events recorded in each of the five corresponding lower level Attention Registers. Those registers are listed in Table 8-1.

TABLE 8-1. Attention Registers

Attention Registers
Master Attention Register (MAR)
State Attention Register (STAR)
No Space Attention Register (NSAR)
Service Attention Register (SAR)
Request Attention Register (RAR)
Indicate Attention Register (IAR)

The host may control which attention bits will generate an interrupt by configuring the Notify Registers (see Table 8-2). For each Attention Register a corresponding Notify Register exists. Each Attention Register is ANDed with its corresponding Notify Register and then all of the resulting signals are ORed together and presented to the next level (see *Figure 8-1*).

TABLE 8-2. Notify Registers

Notify Registers
Master Notify Register (MNR)
State Notify Register (STNR)
No Space Notify Register (NSNR)
Service Notify Register (SNR)
Request Notify Register (RNR)
Indicate Notify Register (INR)

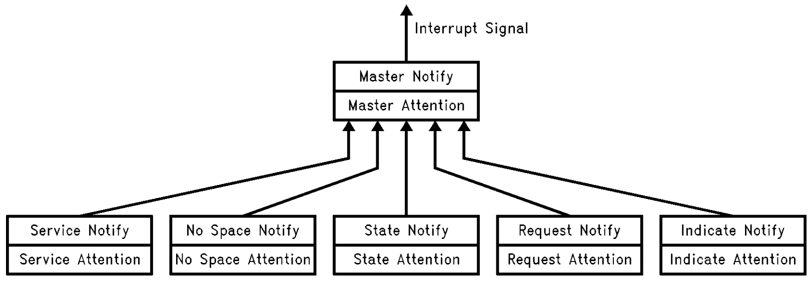


FIGURE 8-1. Attention/Notify Registers

For example, to disable all interrupts caused by service events: clear the Service Attention Register Notify (SVAN) bit in the Master Notify Register (MNR). To disable only interrupts caused by Pointer RAM Operations: set the SVAN bit in the MNR and clear the PTOPN bit in the Service Notify Register (SNR).

When checking Attention Registers for the cause of an interrupt, one should perform a bit-wise AND operation between the attention and notify Registers and examine the result. Just checking the Attention Registers may be misleading. For example, on an unused Indicate Channel one may wish to leave its PSP queue empty and mask off the ‘‘Low Data Space’’ attention bit for that Channel. This masking is accomplished via the Indicate Notify Register (INR). Under these circumstances the IAR, by itself, may contain misleading information.

8.2 Example Procedure

A typical procedure for servicing SI interrupts is as follows:

- disable host interrupts
- determine the event that triggered the interrupt by checking the Master Attention Register and then querying the appropriate lower level Attention Register
- process the event (or post the event to a service queue)
- clear the attention bit (or mask the attention bit)
- enable host interrupts.

8.3 The No Space Attention Register (NSAR)

The No Space Attention Register (NSAR) requires some special treatment. The host must never explicitly clear the ‘‘Low Data Space’’ attention bits. When additional pool space is given to the SI it will automatically clear these bits. In the current implementation of the SI, clearing the ‘‘Low Data Space’’ attention bits can cause the SI to fetch a PSP before an LMOP has occurred. For example, if the NSAR is cleared immediately after reset the SI will attempt to prefetch two PSP descriptors for all three Receive Channels; using whatever contents happen to be located in the Queue Pointers for each PSP queue. Also, the host must be careful to update a CNF or IDUD queue's Queue Limit before clearing the ‘‘No Status Space’’ attention bit for that Channel and the host must be careful to place the Queue Limit logically after the Queue Pointer. Section 3.4 discussed the ‘‘queue full’’ condition in detail.

9.0 INITIALIZATION

Before SI operation can begin, the device must be initialized. The BMAC and PLAYER devices must also be individually initialized. To initialize the SI, the steps shown below should be followed. Each action is explained further in the sub sections that follow:

- Reset the SI
- Read the Silicon Revision Number
- Set the Mailbox Address Register
- Set the Mode Register
- Load the Pointer RAM Registers
- Set the Event Notify Registers
- Set the Request Configuration Registers
- Set the Request Expected Frame Status Registers
- Set the Indicate Configuration Register
- Set the Indicate Mode Register
- Set the Indicate Threshold Register
- Set the Indicate Header Length Register (if using Header/Info splitting mode)
- Load the Limit RAM Registers
- Clear the Attention Registers. The No Space Attention Register (NSAR) cannot be simply cleared. The Low Data Space attention bits must not be explicity cleared by the host. The SI automatically clears these bits when consuming PSP Descriptors. See Section 8.3 for details.
- Put the SI in Run Mode.

9.1 Reset the SI

During initialization the SI must be in Stop Mode. This is necessary to prevent the device from attempting to perform any actions or respond to any external stimulus prior the completion of the initialization sequence. The SI may be placed in Stop Mode by setting three bits in the State Attention Register (STAR): SPSTOP, RQSTOP and INSTOP. However, the SI automatically places itself in Stop Mode after a reset. Since the entire device is being initialized anyway, it makes sense to go ahead and reset the entire SI. This is done by setting the MRST bit in the Mode Register (offset x′00′ of the Control Bus).

9.2 Read the Silicon Revision Number

The host may obtain the revision number of a given SI immediately after resetting the SI. The process of reading the revision number is discussed in detail in Section 6, Low Level Operations. Basically, the host clears the PCAR and reads the MBAR Control Bus Register four times.

9.3 Set the Mailbox Address Register

When loading Pointer RAM data into the SI, a ‘‘mailbox’’ mechanism is used. The mailbox is a 32 bit word in off-chip memory which the SI uses to load or dump the Pointer RAM Registers. This mailbox may be located anywhere within the 28-bit ABus address space of the SI and accordingly its address must be explicitly defined. The process of loading the mailbox address is discussed in detail in Section 6, Low Level Operations. It works a lot like reading the revision number; clear the PCAR and write to the MBAR four times (in most significant byte order).

9.4 Set the Mode Register

Load the SI Mode Register (MR) to configure the SI with global bus and queue parameters. For example a value of 0x52 causes the SI to generate 32 byte bursts when accessing the data bus, use 1k (small) queues, operate in a physical memory environment, use ‘‘big-Endian’’ data alignment, check parity on access to the ABus and Control Bus and optimize operation for clock speeds over 12.5 MHz.

9.5 Load the Pointer RAM Registers

As described in Section 3.4, the SI Queues in depth, the Queue Pointers define the memory location of the queues and which queue slot the SI will access next. The Queue Pointers are located in the internal Pointer RAM Registers (not directly accessible by the host). The actions required to load and store Pointer RAM Registers are covered in Section 6, Low Level Operations.

The above steps must be done for all pointers associated with those Channels that will be used.

9.6 Set the Event Notify Registers

You may specify which events will trigger an interrupt by setting the corresponding bit in the Notify Registers; where a 1 enables interrupts from that event and a 0 disables those interrupts. The Notify Registers may be written without being read previously (not conditional write registers).

9.7 Set the Request Configuration Registers

Load the Request Configuration Registers (R0CR and R1CR) for both Request Channels (RCHN0 and RCHN1) to establish Channel specific operating parameters; such as Source Address and Frame Control Transparency.

9.8 Set the Request Expected Frame Status Registers

This step is only required when using Full Confirmation. Load the Request Expected Frame Status Registers (R0EFSR and R1EFSR) for both Request Channels (RCHN0 and RCHN1) to set up the expected status for frame confirmation services. A value of 0x00 in these registers means that any frame status is acceptable. These registers are used for Full Confirmation of transmitted frames to declare the condition under which the SI should stop sending frames (i.e., the receiving station stopped copying them). If this behavior is not desired when using Full Confirmation, then the host may load the Request Expected Frame Registers with 0x00.

9.9 Set the Indicate Configuration Register

Load the Indicate Configuration Register (ICR) to establish copy control parameters for each Indicate Channel. A typical register value is 0x49; which instructs the SI to copy frames addressed for the station’s own MAC address or to an externally matched group address.

9.10 Set the Indicate Mode Register

Load the Indicate Mode Register (IMR) to set the frame sorting mode, skip option and the desired Indicate breakpoints. This register may only be updated when the frame reception logic (Indicate Machine) is stopped.

Indicate breakpoints are instances that generate interrupts. You may configure the SI to interrupt at the end of each service opportunity, at the end of a burst (i.e., Channel change) or after a user defined number of frames have been received. Prudent use of Indicate breakpoints can significantly reduce interrupt processing overhead by reducing the number of interrupts generated by the SI.

9.11 Set the Indicate Threshold Register

The Indicate Threshold Register (ITR) specifies the maximum number of frames that can be received before a threshold breakpoint is realized. The value in this register is only used when the appropriate bits (BOT1, BOT2) are set in the IMR.

Loading the ITR with 0x00 specifies a value of 256. This value is loaded into an internal working register each time the current Receive Channel changes (i.e., on a burst boundary). In this context ‘‘burst boundary’’ refers to a change in the stream of received frames (e.g., sent from a different station), not the transfer of frame data to and from memory.

9.12 Set the Indicate Header Length Register

If the Header/Info frame sorting mode is specified, one must load the Indicate Header Length Register (IHLR) with the length (in units of four byte words) of the header portion of the frame. The FC field occupies an entire word. For example, to separate an 8 octet header when using long, six-octet MAC addresses, one would load a value of 6 (FC=1, DA/SA=3, header=2) into this register.

9.13 Load the Limit Ram Registers

During normal operation of the SI, the CNF and IDUD queues must be given status space. This may be done as part of the initialization procedure. Though, the Limit RAM Registers assigned to the PSP and REQ queues should only be updated when actually placing descriptors on the given queues. The actions necessary to load Limit RAM Registers are covered in Section 6, Low Level Operations.

9.14 Clear the Attention Registers

Clear the Request Attention (RAR) and Indicate Attention Registers (IAR) by first reading the register, which has the side-effect of loading the Compare Register (CMP), and then writing a 0x00 value to the register. Both of these registers are automatically initialized to 0 upon SI reset.

The No Space Attention Register (NSAR) must be treated carefully. Only the No Status Space attention bit in the NSAR should be modified and only those bits that correspond to Channels that are being configured for operation.

Don't clear the Low Data Space bits. The Low Data Space bits are cleared automatically as PSP Descriptors are fetched.

9.15 Put the SI in Run Mode

Initialization of the SI is now complete. The device may be made fully operational by reading the State Attention Register (STAR) and immediately writing 0x00 to it. This will clear the stop bits for the Indicate, Request and Status/Space machines; putting them in Run Mode. The SI should immediately begin fetching PSP Descriptors for the Indicate Channels to use for frame reception. At this point a write to one of the REQ queue Limit RAM Registers would cause the SI to begin fetching REQ Queue Descriptors for frame transmission.

10.0 EXCEPTION HANDLING

This section of the document describes some of the error events that can be generated by the SI. It discusses some of the possible causes of various error conditions and suggests possible actions for the host to take in response to these exceptional conditions. A distinction is made between errors that are transient (i.e., trying to transmit asynchronous frames when the ring is non-operational) and errors that are serious enough to warrant resetting the SI.

Exception conditions are reported via several Attention Registers (accessible via the Control Bus) and descriptors in the status queues (CNF and IDUD).

10.1 Serious Errors

The State Attention Register (STAR) reports errors that software should consider to be serious. For example, parity errors on the Control Bus or BMAC Interface and errors on the ABus (address/data bus). A quick description of the "serious" attention bits in the STAR follows.

- INSTOP. When the INSTOP bit is set the Indicate Machine (frame reception logic) is stopped. The SI sets this bit when it is reset and when an internal error of the Indicate Machine has been detected. The host should reset the SI if this bit goes high during normal operation.
- RQSTOP. When the RQSTOP bit is set the Request Machine (frame transmission logic) is stopped. The SI sets this bit when it is reset, when an internal error of the Request Machine has been detected and when an ABus error has occurred during REQ, ODUD, ODU or CNF accesses. The host should reset the SI if this bit goes high during normal operation.
- SPSTOP. When the SPSTOP bit is set the Status/Space Machine is stopped. The SI sets this bit when it is reset and when an unrecoverable error occurs during a PSP fetch of a Pointer RAM Operation (PTOP). The host should reset the SI if this bit goes high during normal operation.
- ERR. The ERR bit is set whenever any ABus error occurs and if any internal state machine errors are detected. This is a good attention bit to use for getting notification of "serious" errors (interrupts). The host should reset the SI if this bit goes high anytime during normal operation.

10.2 Parity Errors

The State Attention Register (STAR) reports parity errors on the Control Bus and the interface between the SI and BMAC Devices. Parity checking is only done when the FLOW bit is set in the SI's Mode Register (MR). A quick description of these attention bits follows:

- CPE. The CPE bit is set when a parity error is detected on the Control Bus during a write operation. The host should then retry the write operation again.
- BPE. The BPE bit is set when a parity error is detected on the interface between the SI and BMAC Devices during frame reception. The host doesn't have to take any particular action since the parity error will be also reported in an IDUD Descriptor. However, this bit is a "sticky bit", so the host should either clear it or completely ignore it.

10.3 Request (Transmit) Exceptions

Request (transmit) exceptions are reported in the Request Attention Register (RAR) and CNF Descriptors. There are two types of exceptions events that are reported in the RAR: "exceptions" and "unserviceable requests" (separate bits are defined for each Channel). A description of these types of attention bits follows:

- EXC. The EXC bit (EXC0 or EXC1) is set when an exception occurs on the corresponding Transmit Channel. There are a lot of different conditions that can cause this bit to go high. Some examples are: ABus error, MAC reset, RINGOP change during transmission, FIFO underrun, etc. Most of these errors can be considered to be of a transient nature, so the host should make note of the condition and clear the attention bit. The SI will also set this bit when an ABus error occurs during transmission (which generally should trigger the host to reset the SI). However, since the ERR bit in the State Attention Register (STAR) will also be set, the code that deals with this bit need not deal with the possibility of an ABus error.
- USRR. The USRR bit (USRR0 or USRR1) is set when the SI cannot service a transmit request. The most common reason for this is that the network goes down while there are active Request Objects on a REQ queue. For example, say there are three Request Objects on a REQ queue and while the SI is consuming the first Request Object the ring goes non-operational. The SI will flush the first Request Object, stop the Channel and set this bit. The other two Request Objects are still valid. When the host clears the USRR bit the SI will start consuming Request Objects again and attempt to send the second and third Request Object. For "just like Ethernet" behavior, the host should just clear this bit whenever it becomes set.
- CNF Descriptor. The Request Status field of the CNF Descriptor (CNF.RS) offers more detail on the causes of transmit exceptions. For example, if a FIFO underrun occurs the EXC bit in the RAR will be set and the CNF.RS field will report that a FIFO underrun is the cause of the exception.

10.4 Indicate (Receive) Exceptions

Indicate (receive) exceptions are reported in the Indicate Attention Register (IAR) and IDUD Descriptors. The IAR has an EXC bit for each of the three Receive Channels. These bits get set when an ABus error occurs when writing frame data or an IDUD Descriptor. On BSI-1 Devices if the host sees this bit go high, then it should consider the last frame in the corresponding Channel's IDUD queue to be in error (regardless of the status reporting in that frames IDUD Descriptors). The Indicate Status field of the IDUD Descriptor (IDUD.IS) offers detailed information on the quality of received frames. This field should always be checked when deciding whether or not to accept a frame from the network. The Terminating Condition, Valid FCS and Valid Data Length fields in the IDUD should also be checked when determining frame acceptabiity. See the appropriate Device datasheet for more information.

10.5 Low Data Space Attentions

When the SI reaches the ‘‘end of queue’’ condition on a PSP queue it notifies the host by raising a Low Data Space attention bit in the No Space Attention Register (NSAR). It is called ‘‘Low Data’’ instead ‘‘No Data’’, because it may still have one more page available for storing incoming frame data.

The host must never explicitly clear these attention bits in the NSAR (even when initializing the SI). The host must effectively clear these bits by producing PSP Descriptors. Thus the host must either produce more PSP Descriptors or mask off the attention from raising an interrupt.

10.6 No Status Space Attentions

When the SI detects that a given status queue (CNF and IDUD queues) is full it raises a ‘‘No Status Space’’ attention bit in the No Space Attention Register (NSAR). All SI Channels require status space (even Transmit Channels). So, when a status queue becomes full the Channel stops. The host must manually clear this attention bit for Channel operation to continue and must first update the status queue's Queue Limit to denote which queue slots the SI may use when producing descriptors.

10.6.1 A Suggested Method for Handling NSAR Attentions

Typically the NSAR attentions occur when receiving frames. For example, if the station is being flooded with lots of small frames the IDUD queue may fill up, because the host can't keep up with the network. Also, if the station is getting ‘‘pounded’’ with lots of big frames it may run out of receive buffer before the IDUD queue fills up. In both of these situations, the host can free up the needed resource by consuming frames, since the act of consuming frames frees up both queue and data space. Thus, the host may treat the NSAR attentions like yet another signal to go consume frames and handle the clearing of the No Status Space attention bit as part of the normal mechanism for granting more IDUD queue slots. This turns out to be a good place to clear No Status Space attentions, since the Queue Limit must be updated first anyway. In summary, the host may

- Use NSAR.LDIx and NSAR.LSIx like another indicate breakpoint or simply ignore them if some other mechanism is in place for scheduling received frame processing (e.g., Indicate Breakpoints or polling for frame arrivals).
- Produce PSP Descriptors as part of the frame reception logic (the SI clears Low Data Space attention bits automatically).
- Update the IDUD queue's Queue Limit (Limit RAM Register) to indicate that the queue space occupied by the consumed IDUDs is available for the SI. For performance reasons it is desirable to delay updates to IDUD Queue Limits until after a threshold has been reached (i.e., half the queue size). This threshold value may actually approach the size of the queue. How close the threshold gets to the queue size depends upon the latency between arrivals and Queue Limit updates. This improves the performance of the station and eliminates the need to include special logic to verify that the Queue Limit is logically ahead of the Queue Pointer.
- Make sure that the No Status Space attention bit is cleared after updating the IDUD queue's Queue Limit.

11.0 SI REGISTERS

The SI has three sets of registers: Control Bus Registers (which can be directly accessed by the host), Pointer RAM Registers (which are internal to the SI) and Limit RAM Registers (which are also internal to the SI). The SI datasheet should be examined for detailed information about each register.

11.1 SI Control Bus Registers

The MACSI device datasheets has a complete explanation of each of the Control Bus Registers. Here is a quick reference table for locating offsets and bit fields (useful when debugging).

TABLE 11-1. System Interface Registers

Addr	Name	D7	D6	D5	D4	D3	D2	D1	D0	Read	Write
100	SIMR0	SMLB	SMLQ	VIRT	BIGEND	FLOW	MRST	FABCLK	TEST	Always	Always
101	SIMR1	AB__A31	AB__A30	AB__A29	AB__A28	ATM	ASM	RES	EAM	Always	Always
102	PCAR	BP1	BP0	PTRW	A4	A3	A2	A1	A0	Always	Always
103	MBAR	Mailbox Address [27:24], [23:16], [15:8], [7:0]								Always	Always
104	MAR	STA	NSA	SVA	RQA	INA	RES	RES	RES	Always	Ignored
105	MNR	STAN	NSAN	SVAN	RQAN	INAN	RES	RES	RES	Always	Always
106	STAR	ERR	BPE	CPE	CWI	CMDE	SPSTOP	RQSTOP	INSTOP	Always	Conditional
107	STNR	ERRN	BPEN	CPEN	CWIN	CMDEN	SPSTOPN	RQSTOPN	INSTOPN	Always	Always
108	SAR	RES	RES	RES	RES	ABR0	ABR1	LMOP	PTOP	Always	Conditional
109	SNR	RES	RES	RES	RES	ABR0N	ABR1N	LMOPN	PTOPN	Always	Always
10A	NSAR	NSR0	NSR1	LDI0	NSI0	LDI1	NSI1	LDI2	NSI2	Always	Conditional
10B	NSNR	NSR0N	NSR1N	LDI0N	NSI0N	LDI1N	NSI1N	LDI2N	NSI2N	Always	Always
10C	LAR	LRA3	LRA2	LRA1	LRA0	LMRW	RES	RES	LRD8	Always	Always
10D	LDR	LRD7	LRD6	LRD5	LRD4	LRD3	LRD2	LRD1	LRD0	Always	Always
10E	RAR	USRR0	RCMR0	EXCR0	BRKR0	USRR1	RCMR1	EXCR1	BRKR1	Always	Conditional
10F	RNR	USRR0N	RCMR0N	EXCR0N	BRKR0N	USRR1N	RCMR1N	EXCR1N	BRKR1N	Always	Always
110	R0CR0	TT1	TT0	PRE	HLD	FCT	SAT	VST	FCS	Always	Always
111	R1CR0	TT1	TT0	PRE	HLD	FCT	SAT	VST	FCS	Always	Always
112	R0EFSR	VDL	VFCS	EE1	EE0	EA1	EA0	EC1	EC0	Always	Always
113	R1EFSR	VDL	VFCS	EE1	EE0	EA1	EA0	EC1	EC0	Always	Always
114	IAR	RES	RES	EXCI0	BRKI0	EXCI1	BRKI1	EXCI2	BRKI2	Always	Conditional
115	INR	RES	RES	EXC0N	BRK0N	EXC1N	BRK1N	EXC2N	BRK2N	Always	Always
116	ITR	THR7	THR6	THR5	THR4	THR3	THR2	THR1	THR0	Always	INSTOP = 1 or EXC = 1 Only
117	IMCR	SM1	SM0	SKIP	FPP	BOT2	BOT1	BOB	BOS	Always	INSTOP = 1 Only
118	ICCR	CC0		RES	CC1		RES	CC2		Always	Always
119	IHLR	HL7	HL6	HL5	HL4	HL3	HL2	HL1	HL0	Always	INSTOP = 1 or EXC = 1 Only
11A	ACR	PCKI2	PCKI1	PCKI0	RSWP1	RSWP0	ISWP2	ISWP1	ISWP0	Always	Always
11B	R0CR1	EFT	RES	RES	RES	RES	RES	RES	ETR	Always	Always
11C	R1CR1	EFT	RES	RES	RES	RES	RES	RES	ETR	Always	Always
11D–11E	Reserved	RES	RES	RES	RES	RES	RES	RES	RES	N/A	N/A
11F	SICMP	CMP7	CMP6	CMP5	CMP4	CMP3	CMP2	CMP1	CMP0	Always	Always
120–1FF	Reserved	RES	RES	RES	RES	RES	RES	RES	RES	N/A	N/A

Note: Bits in the conditional write registers are only written when the corresponding bit in the System Interface Compare Register is equal to the bit to be overwritten.

11.2 Pointer RAM Registers

The MACSI datasheet has a complete explanation of each of the Pointer RAM Registers. Here is a quick reference table for locating offsets. The defined Pointer RAM Registers are always readable and writable.

11.3 Limit RAM Registers

The SI datasheet has a complete explanation of each of the Limit RAM Registers. Here is a quick reference table for locating offsets. All of the defined Limit RAM Registers are always readable and writable.

TABLE 11-2. Pointer RAM Registers

Group	Address	Register Name	Access Rules Read	Access Rules Write
POINTER RAM	00	ODU Pointer RCHN1 (OPR1)	Always	Always
	01	ODUD List Pointer RCHN1 (OLPR1)	Always	Always
	02	CNF Queue Pointer RCHN1 (CQPR1)	Always	Always
	03	REQ Queue Pointer RCHN1 (RQPR1)	Always	Always
	04	ODU Pointer RCHN0 (OPR0)	Always	Always
	05	ODUD List Pointer RCHN0 (OLPR0)	Always	Always
	06	CNF Queue Pointer RCHN0 (CQPR0)	Always	Always
	07	REQ Queue Pointer RCHN0 (RQPR0)	Always	Always
	08	IDU Pointer ICHN2 (IPI2)	Always	Always
	09	IDUD Queue Pointer ICHN2 (IQPI2)	Always	Always
	0A	PSP Queue Pointer ICHN2 (PQPI2)*	Always	Always
	0B	Next PSP ICHN2 (NPI2)	Always	Always
	0C	IDU Pointer ICHN1 (IPI1)	Always	Always
	0D	IDUD Queue Pointer ICHN1 (IQPI1)	Always	Always
	0E	PSP Queue Pointer ICHN1 (PQPI1)*	Always	Always
	0F	Next PSP ICHN1 (NPI1)	Always	Always
	10	IDU Pointer ICHN0 (IPI0)	Always	Always
	11	IDUD Queue Pointer ICHN0 (IQPI0)	Always	Always
	12	PSP Queue Pointer ICHN0 (PQPI0)*	Always	Always
	13	Next PSP ICHN0 (NPI0)	Always	Always
	14	IDUD Shadow Register (ISR)	Always	Always
	15	ODUD Shadow Register (OSR)	Always	Always
	16–1F	Reserved	N/A	N/A

*Note: Bit position D2 of these Pointer RAM Locations is always forced to a 1, (The first word of a PSP is not fetched).

TABLE 11-3. Limit RAM Registers

Group	Address	Register Name	Access Rules Read	Access Rules Write
LIMIT RAM	0	REQ Queue Limit RCHN1 (RQLR1)	Always	Always
	1	CNF Queue Limit RCHN1 (CQLR1)	Always	Always
	2	REQ Queue Limit RCHN0 (RQLR0)	Always	Always
	3	CNF Queue Limit RCHN0 (CQLR0)	Always	Always
	4	IDUD Queue Limit ICHN2 (IQLI2)	Always	Always
	5	PSP Queue Limit ICHN2 (PQLI2)	Always	Always
	6	IDUD Queue Limit ICHN1 (IQLI1)	Always	Always
	7	PSP Queue Limit ICHN1 (PQLI1)	Always	Always
	8	IDUD Queue Limit ICHN0 (IQLI0)	Always	Always
	9	PSP Queue Limit ICHN0 (PQLI0)	Always	Always
	A-F	Reserved	N/A	N/A

12.0 GLOSSARY OF TERMS

ABus	The Address/Data bus interface of the SI. This interface is 32 bits wide. The SI always acts as a Bus Master on this interface. [2]
Asynchronous	A class of data transmission service whereby all requests for service contend for a pool of dynamically allocated ring bandwidth and response time. [1]
Channel	A logical I/O port on the SI. All channels are either used to transmit frames or receive frames. Each channel has two queues associated with it. [2]
Control Bus	The Control Bus Interface of the SI (as well as the BMAC and PLAYER Devices). This interface is 8 bits wide. The SI always acts as Bus Slave on this interface. [2]
CNF Descriptor	An 8-byte descriptor that reports the status of previously transmitted frames. [2]
FDDI	Fiber Distributed Data Interface. FDDI provides a high-bandwidth (100 megabits per second), general purpose interconnect among computers and peripheral equipment using fiber optics as the transmission medium in a ring configuration. [1]
FIFO	First-In-First-Out
Frame	A PDU transmitted between cooperating MAC entities on a ring, consisting of a variable number of octets and control symbols. [1]
Indicate	OSI terminology for receive.
Input Data Unit (IDU)	A contiguous range of memory which contains part or all of a Frame that has been received from the FDDI network and transferred to memory by the SI. IDUs will never cross a 4 kb page boundary and will always be aligned on an ABus burst boundary (either 16 or 32 bytes). The first IDU of a frame will begin on the last byte of the first word. [2]
IDU Descriptor (IDUD)	An 8-byte descriptor that describes a particular IDU. The format of the IDU Descriptor (IDUD) is a superset of the ODUD. The IDUD informs the host of the IDU's location in memory, the size of the IDU and, in the last IDUD for a frame, status information regarding frame reception. [2]
Limit RAM Operation (LMOP)	A sequence of actions taken by a host processor to access (read or write) a Limit RAM Register (internal to the SI). LMOPs are accomplished strictly via several Control Bus Registers. [2]
Logical Link Control (LLC)	Part of the IEEE 802.2 standard.
Media Access Control (MAC)	The Data Link Layer responsible for scheduling and routing data transmissions on a shared medium Local Area Network (e.g., an FDDI ring).[1]
Octet	A data unit composed of eight ordered bits (a pair of data symbols). [1]
Output Data Unit (ODU)	A contiguous range of memory which contains part or all of a Frame to be transmitted by the SI. This memory may not cross a four kb page boundary. ODUs are presented to the SI as a complete Frame via one or more ODU Descriptor s (ODUD). [2]
ODU Descriptor (ODUD)	An 8-byte descriptor that describes a particular ODU. It informs the SI of the ODU's location in memory, the ODU's size and (together with zero or more other ODU Descriptors (ODUD)) the ODU's position within the entire Frame to be transmitted. [2]
Pointer RAM Operation (PTOP)	A sequence of actions taken by a host processor to access (read or write) a Pointer RAM Register (internal to the SI). PTOPs involve the use of a memory mailbox word and some Control Bus Registers. [2]
Physical Layer (PHY)	The Physical Layer responsible for delivering a symbol stream produced by an upstream MAC Transmitter to the logically adjacent downstream MAC Receiver in an FDDI ring. [1]
PMD	The Physical Medium Dependent Layer.
Protocol Data Unit (PDU)	The unit of data transfer between communicating peer layer entities. It may contain control information, address information, data (e.g., an SDU from a higher layer entity), or any combination of the three. The FDDI MAC PDUs are Tokens and Frames. [1]
Pool Space Descriptor (PSP)	An 8-byte descriptor that is used to declare a pool of buffers into which received frame data will be copied. It is also used to declare the sequence in which these buffers will be used. [2]
Queue	A block of memory registered to act as a descriptor FIFO between the host and SI. Each queue has a Queue Pointer and a Queue Limit associated with it. There are two queues per channel.
Queue Limit	An internal register of the SI that defines how far the SI may advance when producing or consuming descriptors in a descriptor queue. Queue Limits are maintained solely by host software.
Queue Pointer	An internal register of the SI that defines the next position in a queue that the SI will access when next producing or consuming a descriptor. Queue Pointers are initialized by host software and maintained by the SI thereafter.
Repeat	The action of a station in receiving a Token or Frame from the adjacent upstream station and simultaneously sending it to the adjacent downstream station. The FDDI MAC may repeat received PDUs (Tokens and Frames), but does not repeat the received symbol stream between PDUs. While repeating a Frame, a MAC may copy the data contents and modify the control indicators as appropriate. [1]

Request	OSI terminology for transmit.
Request Descriptor (REQ)	An 8-byte descriptor that describes an array of ODU Descriptors; which in turn describe one or more frames. The REQ Descriptor acts as the top level of the three level transmit data structure used by the SI. [2]
Service Opportunity	A span of time triggered by the capturing of a usable Token during which a station may transmit Frames onto the ring.
Station	An addressable logical and physical attachment in a ring, capable of transmitting, receiving and repeating information. An FDDI station has one or more PHY entities, one or more MAC entities and one SMT entity. [1]
Station Management (SMT)	The supervisory entity within an FDDI station that monitors and controls the various FDDI entities including PMD, MAC and PHY. [1]
Symbol	The smallest signaling element used by MAC, i.e., the PHY Service Data Unit. The symbol set consists of 16 data symbols and 8 control symbols. Each symbol maps to a specific sequence of five code bits as transmitted by the Physical Layer. [1]
Synchronous	A class of data transmission service whereby each requester is pre-allocated a maximum bandwidth and guaranteed a response time not to exceed a specific delay. [1]
Token	An explicit indication of the right to transmit on a shared medium. On a Token Ring, the Token circulates sequentially through the stations in the ring. At any time, it may be held by zero or one station. FDDI uses two classes of Tokens: restricted and nonrestricted. [1]

REFERENCES

1. ANSI X3.139-1987
2. National Semiconductor 1991 FDDI Databook