BSI™ Device Software Design Guide

National Semiconductor
Application Note 730
Robert Macomber, Mark Travaglio
November 1990

Table of Contents

1.0 INTRODUCTION

This application note describes how to initialize the National Semiconductor BSI device (DP83265) and interact with it. Initialization and data service support occur through the Control Bus and via memory that is accessible by both the BSI device and the host. The host processor must be able to respond to interrupts and have access to both the BSI device Control Bus and some mutually accessible memory. This application note should be read in conjunction with the BSI datasheet.

2.0 INITIALIZATION

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

- Put the BSI Device in Stop Mode
- Set the Mailbox Address Register
- Load the Pointer RAM
- Set the Event Notify Registers
- Set the Mode Register
- 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
- Load the Limit RAM
- Clear the Attention Register
- Put the BSI Device in Run Mode

2.1 Put the BSI Device in Stop Mode

During initialization the BSI device 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 to the completion of the initialization sequence. The BSI device may be placed in Stop Mode by setting three bits in the State Attention Register (STAR).

Since the State Attention Register (STAR) is a conditional write register, it must be read before it is written. By doing so the original contents of the STAR register are loaded into the Compare Register (CMP). When a subsequent write to the STAR register occurs, only those bits that match the corresponding bits in the CMP register will be actually stored in STAR. With the BSI device in ''Stop Mode'', and no intervening accesses to the BSI device Control Bus, it is guaranteed that all 8 bits will match.

Finally, write 0x07 to the STAR. This clears all the error bits and sets all the stop bits (the STAR is automatically loaded with 0x07 upon reset of the BSI device).

2.2 Set the Mailbox Address Register

When loading Pointer RAM data into the BSI device, a ''mailbox'' mechanism is used. The mailbox is a 32-bit word in off-chip memory which the BSI device uses to load or dump the Pointer RAM Registers. This mailbox may be located anywhere within the 28-bit ABus address space of the BSI device and accordingly its address must be explicitly defined. This is accomplished via the 8-bit Mailbox Address Register (MBAR).

To load the Mailbox Address Register (MBAR) first load 0x00 into the Pointer RAM Control and Address Register (PCAR). This tells the BSI device to internally point to the first byte of the mailbox address. Then execute four successive writes to the MBAR to load the full mailbox address; writing the most significant bytes first. Each write automatically increments the byte pointer to the next byte.

When the BSI device is reset, the Mailbox Address Register (MBAR) is loaded with a hardware revision code. The host may obtain this revision code by sequentially reading four bytes from the MBAR before loading the mailbox address.

2.3 Load the Pointer RAM

The BSI device maintains pointer registers for accessing and manipulating the various queues. Prior to normal operation, some of these pointer registers must be initialized with queue addresses (see Table I). For active Request queues the host software must load the Confirmation Message (CNF) Queue Pointer Register and the Request (REQ) Queue Pointer Register. For Indicate queues the host software must load the Indicate Data Unit Descriptor (IDUD) Queue Pointer Register and the Pool Space (PSP) Queue Pointer Register. For information about choosing initial Pointer RAM values see Section 7 on Queue Manipulation.

TABLE I. Pointer Registers Used during Queue Initialization

Pointer Registers		Addr
CNF	Queue Pointer Register (RCHN1)	0x02
REQ	Queue Pointer Register (RCHN1)	0x03
CNF	Queue Pointer Register (RCHN0)	0x06
REQ	Queue Pointer Register (RCHN0)	0x07
IDUD	Queue Pointer Register (ICHN2)	0x09
PSP	Queue Pointer Register (ICHN2)	0x0a
IDUD	Queue Pointer Register (ICHN1)	0x0d
PSP	Queue Pointer Register (ICHN1)	0x0e
IDUD	Queue Pointer Register (ICHN0)	0x11
PSP	Queue Pointer Register (ICHN0)	0x12

Before loading a Pointer RAM Register, first read the Service Attention Register (SAR) to verify that the PTOP bit is

RRD-B30M75/Printed in U. S. A.

set; signifying that a previous Pointer RAM operation has completed. If this bit is not set wait for the previous operation to finish.

Write the value one wishes to store in the Pointer RAM Register (i.e., the base address of the relevant queue) in the memory location selected as the BSI device Mailbox.

Next configure the Pointer RAM Control and Address Register (PCAR) with the PTRW bit cleared and the address of the Pointer RAM Register placed in the least significant five bits. A zero value in the PTRW bit specifies that the next Pointer RAM operation will read from the Mailbox and write to the Pointer RAM Register. The two most significant bits in the PCAR (BP0, BP1) are not used in this context and may be loaded with 0's. For example, when loading the PSP Queue Pointer for Indicate Channel 0, one would write 0x12 to the PCAR.

Finally, clear the PTOP bit in the Service Attention Register (SAR). The SAR is a conditional write register, so it is necessary to read it immediately before writing to it. Clearing the PTOP bit causes the BSI device to perform the actual Pointer RAM operation. The device signals the completion of the operation by setting the PTOP bit in the SAR.

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

2.4 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).

See Section 3, Servicing Interrupts, for a more complete treatment of this subject.

2.5 Set the Mode Register

Load the BSI device Mode Register (MR) to configure the BSI device with global bus and queue parameters. For example a value of 0x52 causes the BSI device 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.

2.6 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.

2.7 Set the Request Expected Frame Status Registers

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.

2.8 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 BSI device to copy frames addressed for the owned MAC address or to an externally matched group address.

2.9 Set the Indicate Mode Register

Load the Indicate Mode Register (IMR) to set the frame sorting mode, skip option and the desired Indicate breakpoints.

Indicate breakpoints are instances that generate interrupts. You may configure the BSI device 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 BSI device.

2.10 Set the Indicate Threshold Register

The Indicate Threshold Register (ITR) specifies how many frames must be received before a threshold breakpoint is realized. The value in this register is only used when the appropriate bits 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 state of any Indicate Channels change.

2.11 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.

2.12 Load the Limit RAM

During normal operation of the BSI device, the CNF and IDUD queues must be given status space. This may be done as part of the initialization procedure. For information about choosing initial Limit RAM values see Section 7 on Queue Manipulation.

Before loading a Limit RAM Register, first read the Service Attention Register (SAR) to verify that the LMOP bit is set (signifying that the previous Limit RAM operation has completed). If this bit is not set wait for the previous operation to finish.

Next load the Limit Address Register (LAR). The top four bits of the LAR define the target Limit RAM Register, the LMRW bit specifies what the next Limit RAM operation will be (LMRW = 0 means a write to the Limit RAM) and the MSBD bit contains the most-significant data bit of the 9-bit Limit value.

Next load the Limit Data Register (LDR) with the lower 8 bits of the limit value.

Finally, write a 0 into the LMOP bit in the Service Attention Register (SAR). The SAR is a conditional write register, making it necessary to read it immediately before writing to it. Clearing the LMOP bit causes the BSI device to perform the actual Limit RAM operation. The BSI device signals the completion of the operation by setting the LMOP bit in the SAR.

Repeat the above steps for all desired limits.

2.13 Clear the Attention Registers

Clear the Request Attention (RAR) and Indicate Attention Registers (IAR) by first reading the register, to load the Compare Register (CMP), and then writing a 0x00 value to the register. Both of these registers are automatically initialized to 0 upon BSI device reset.

The No Space Attention Register (NSAR) should be initialized to reflect the state of space of all the queues. If space was given to all of the CNF and PSP queues, read and write 0x00 into NSAR.

2.14 Put the BSI Device in Run Mode

Initialization of the BSI device 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 BSI device 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 BSI device to begin fetching REQ Queue Descriptors for frame transmission.

3.0 SERVICING INTERRUPTS

The BSI device 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.

3.1 Event Registers

The BSI device 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 attention bits in the lower level 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 attention lower level registers. Those registers are listed in Table II.

TABLE II. 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)

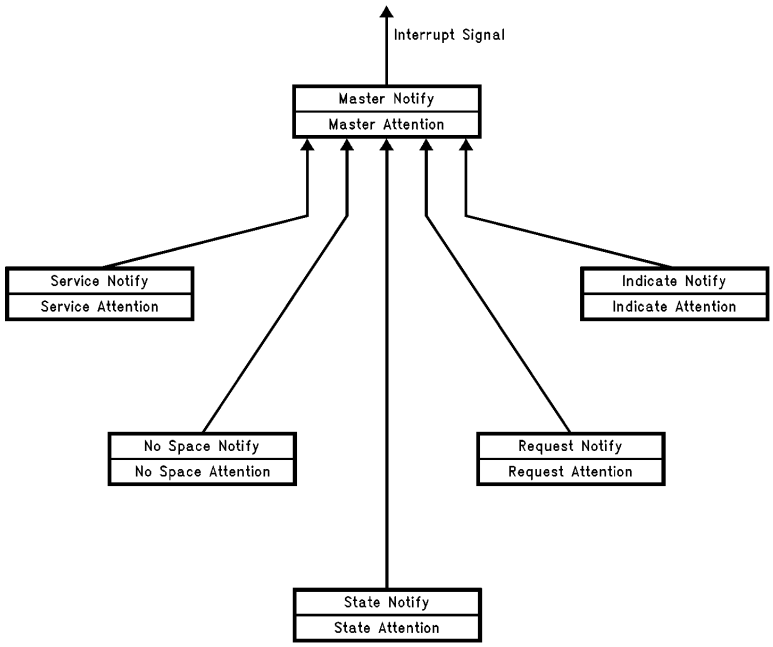
The host may control which attention bits will generate an interrupt by configuring the Notify Registers (see Table III).

TABLE III. 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)

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 1*).

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).



TL/F/11088–1

FIGURE 1. BSI Device Event/Notify Registers

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, to disable an Indicate Channel one may wish to leave its PSP queue empty and mask off the ''Low Data Space'' attention bit for that channel; via the Indicate Notify Register (INR). Under these circumstances the IAR, by itself, may contain misleading information.

3.2 Example Procedure

A typical procedure for servicing BSI device 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

4.0 MEMORY MANAGEMENT SCHEMES

The BSI device may be configured to use memory shared between itself and the host or it may be configured to use the host's memory. In addition, it can be made to operate in a vitural memory environment.

Although the BSI device manages space for incoming data (from channel specific Pool Space (PSP) queues); the host must implement a memory management mechanism to replenish the PSP queues and manage the space needed to hold output data (ODU) and ODU Descriptors (ODUDs).

4.1 Memory Requirements

Up to ten distinct queues may be established; two for each channel. Depending upon the value of the Small Queue (SMLQ) bit in the Mode Register (MR), these queues will each consume 1k or 4k of memory; collectively occupying either 10k or 40k of memory.

A 4 byte word must be allocated as the BSI device Mailbox. This word is only used when accessing the BSI device Pointer RAM Registers.

Space must also be allocated for buffering frames. Any buffers drawn from this space must be no larger than 4 kbytes and may not cross a 4 kbyte boundary. At the current time the Count (CNT) field in the PSP Descriptor is ignored by the BSI device. Pool space is intended to be allocated in 4 kbyte pages.

Space must be allocated for buffering ODU Descriptors (ODUDs). As with frame data, buffers drawn from this space must be no larger than 4 kbytes and may not cross a 4 kbyte boundary.

4.2 Dedicated Buffer Pools

To simplify memory management, buffer pages may be dedicated to individual PSP queues. When using dedicated buffers, the Indicate buffer management task becomes a matter of:

- detecting page boundary crosses; as an indication that the BSI device has finished filling the previous page
- obtaining confirmation that the host has finished processing all frames in the previous page
- attaching the previous page to a PSP Descriptor of the same queue
- incrementing the PSP Limit Register for that queue

The management of space for ODUs (outgoing frame data) and ODU Descriptors (ODUDs) must be done by the host.

A fully allocated 1k PSP queue consumes 512 kbytes of buffer space. A fully allocated 4k PSP queue uses 2 MB of buffer space.

4.3 Shared Buffer Pool

To maximize memory utilization, multiple Indicate Channels may share a single pool of data buffers. This does **not** mean that Indicate Channels can be made to share a Pool Space (PSP) queue, but rather that data buffers attached to the various PSP queues are allocated and freed from a global buffer pool on an ''as needed'' basis. When using a shared buffer pool, the Indicate buffer management becomes the following:

- Detecting page boundary crosses; to determine when the BSI device is finished filling the previous page
- Obtaining confirmation that the host has finished processing all frames in that page
- Returning that page to a shared buffer pool or, upon determining that the page has been dedicated to a given channel, reattach the page to the channel's PSP queue
- Responding to interrupts caused by a ''Low Space'' condition by allocating buffer space to one or more PSP Descriptors and incrementing the PSP Limit Register for that queue

Again, the management of space for ODUs (outgoing frame data) and ODU Descriptors (ODUDs) must be done by the host.

One danger with sharing pool space is that a heavily used low priority channel may starve a high priority channel by consuming all of the buffer space. This is contrary to the idea of priority. It is recommended that some mechanism be implemented for reserving memory for a given channel and that at least four buffer pages be dedicated to Indicate Channel 0 (ICHN0). This is to ensure that FDDI-SMT frames will not be dropped when there is a greal deal of activity on the other Indicate Channels.

5.0 SENDING FRAMES

This section describes how to use the BSI device to queue a frame for transmission on an FDDI ring. It is assumed that the BSI device has been initialized and that the Request Configuration Registers (R0CR and R1CR) and the Request Expected Frame Status Registers (R0EFSR and R1EFSR) have been previously loaded with the desired values.

The mechanism for sending a frame is as follows:

- Obtain space for data structures
- Load the ODU(s)
- Process previous CNFs (optional)
- Build the ODUD(s)
- Build the REQ
- Signal the BSI device

Subsection 5.7 describes some special considerations for sending multiple frames in a single request object.

5.1 Obtain Space for Data Structures

BSI device addressable memory must be obtained to:

- hold the frame data
- hold the ODU Descriptor(s)
- hold the Request Descriptor

It is the responsibility of the host software to manage space for frame data and the ODUDs. Section 4, Memory Management Schemes, describes two simple memory allocation methods. If multiple ODUDs are required, these must be contiguously allocated in the form of an array of ODUDs.

Space for the Request Descriptor must be located within the area designated as the Request queue (see Section 2.3). It must also be allocated in a serially contiguous fashion; immediately following the previously allocated descriptor. All BSI device queue pointers ''wrap'' to the first location upon reaching the end of the queue area.

The frame data may need to be divided between multiple ODUs. Each ODU may start anywhere within a 4k page, but it must end at or before the next 4k boundary. Multiple ODUs must be generated for frames over 4k in length.

For each ODU, space must be allocated for a corresponding ODU Descriptor (ODUD). System configurations in which the BSI device directly addresses host memory, may be able to contrive ODU Descriptors (ODUDs) that refer directly to host specific memory buffers.

5.2 Process Confirmation Status Message Descriptors (CNF)—Optional

When the BSI device processes a request it places confirmation messages in the CNF queue for that Request Channel. By examining these messages, the host may determine when the BSI device has finished using ODU, ODUD and REQ queue space.

If the BSI device has been instructed to generate interrupts after writing confirmation messages, then an autonomous interrupt handler should be available to asynchronously process CNFs. Conversely, if these interrupts have been disabled, then CNFs should be processed when attempting to send a frame.

5.3 Copy Frame Data to Buffer

If the ODU buffers are distinct from the host specific memory buffers, copy the frame data from the host buffer(s) to the ODU buffer(s).

5.4 Build the ODU Descriptor(s)

An ODU Descriptor (ODUD) must be written for each ODU. The address and size of the ODU must be recorded in the ODUD.LOC and ODUD.CNT fields, respectively.

When only a single ODUD is needed both the First and Last bits should be set (only). With multiple ODUDs the first ODUD should has the just First bit set (first) and the last ODUD should have the just Last bit set (last). Any intervening ODUDs should have both bits cleared (middle).

5.5 Build the Request Descriptor

A Request Descriptor must be constructed which references the ODUD(s) that were just built.

The User Identification (UID) field may be assigned a host defined value. This UID value will reemerge in one or more CNFs and may be useful when processing a CNF (i.e., deallocating ODUD and buffer space). The SIZE field should be set to a value of 1; since, in this case, only a single frame will be transmitted.

The Confirmation Class (CNFCLS) field defines the level of request confirmation that the BSI device will use and should be set as needed. To turn off request confirmation put a hex value of 0x4 in the CNFCLS field; although, the BSI device will **always** generate CNF Descriptors whenever an exception is encountered. Please note that request processing will halt for a given channel should that channel's CNF queue become full. Thus, a provision for processing CNF Descriptors must be included in all applications; even those applications that do not wish to receive confirmation for most requests.

The Request Class (RQCLS) field defines the class of the request (i.e., asynchronous, synchronous, restricted token, etc.). The FC field should be loaded with an appropriate FDDI Frame Control value.

When sending a single frame, both bits in the First and Last bits should be set; indicating that this is the only REQ Descriptor in this request object. Finally, the address of the first ODUD should be put into the LOC field.

5.6 Signal the BSI Device about the Request

The BSI device may be caused to examine either of its REQ queues by writing to the corresponding Limit RAM Register with a value that raises the limit to reference the new REQ Descriptor.

5.7 Sending Multiple Frames in a Single Request Object

The BSI device is capable of transmitting multiple frames in a single service opportunity. This feature becomes important on a heavily loaded FDDI ring, with relatively infrequent service opportunities. The BSI device can be caused to process multiple frames by:

- building an ODUD list that contains multiple frames
- building a request object that contains multiple REQ Descriptors or
- a combination of the above two methods.

The first method is extremely simple. Allocate and fill the ODU buffers for all of the frames. Build multiple ODU Descriptor objects (demarcated by the First and Last bits in each ODUD) and concatenate the ODUDs together into one array of descriptors. Build the REQ Descriptor, as before, except load the frame count into the SIZE field.

The second method consists of a creating a REQ Descriptor marked as being ''first'', zero or more REQ Descriptors marked as ''middle'' descriptors and an ending REQ Descriptor marked as being ''last''. The Limit RAM Register, for the given request queue, must be set beyond the last REQ Descriptor. The parameter fields in first REQ Descriptor are used for the entire request object.

5.8 Batching Single Frame Requests

On a heavily loaded FDDI ring service opportunities occur less frequently than on an FDDI ring with only light traffic. On a loaded network it makes sense to send multiple frames per service opportunity. However, many network communication systems send only a single frame at a time. This subsection tells how one may use the capabilities of the BSI device to batch single frame requests into a larger request object.

The BSI device will only attempt to send a single request object in any given service opportunity. A request object is defined here to consist of one or more REQ Descriptors delimited using the First and Last bits found inside each

descriptor. The BSI device interface software needs to build different types of REQ Descriptors when queuing a frame such that:

- A single frame request object is generated when the queue is empty
- The resulting request object is limited to a maximum size
- Optionally the resulting request object is closed whenever a service opportunity is detected.

The following pseudo-code may be used to satisfy the above requirements.

Transmit Logic

```
Req__Size = Req__Size + 1
if Queued__Cnt = 0
 mark as REQ.ONLY
 Open__Req = FALSE
 Queued__Cnt = 1
 REQ__Size = 0
else
 if Open__Req = FALSE
  mark as REQ.FIRST
  Open__Req = TRUE
  Queued__Cnt = Queued__Cnt + 1
  Req__Size = 1
 else
  if Req__Size ≥ Max__Req
   mark as REQ.LAST
   Open__Req = FALSE
   Req__Size = 0
  else
   mark as REQ.MIDDLE
  endif
 endif
endif
```

Interrupt Service Routine Logic (Optional)

```
if Open__Req = TRUE
 generate empty REQ.LAST
 Open__Req = FALSE
 Req__Size = 0
endif
```

Queue__Cnt is the number of queued request objects on the request queue and is set to 0 queue initialization time. **Req__Size** is the number of frames in the currently open request object and is set to 0 at queue initialization time. **Open__Req** is a boolean variable indicating the presence or absence of an open request object and is set to FALSE at queue initialization time. **Max__Req** is a maximum number of frames per request object defined by the host software.

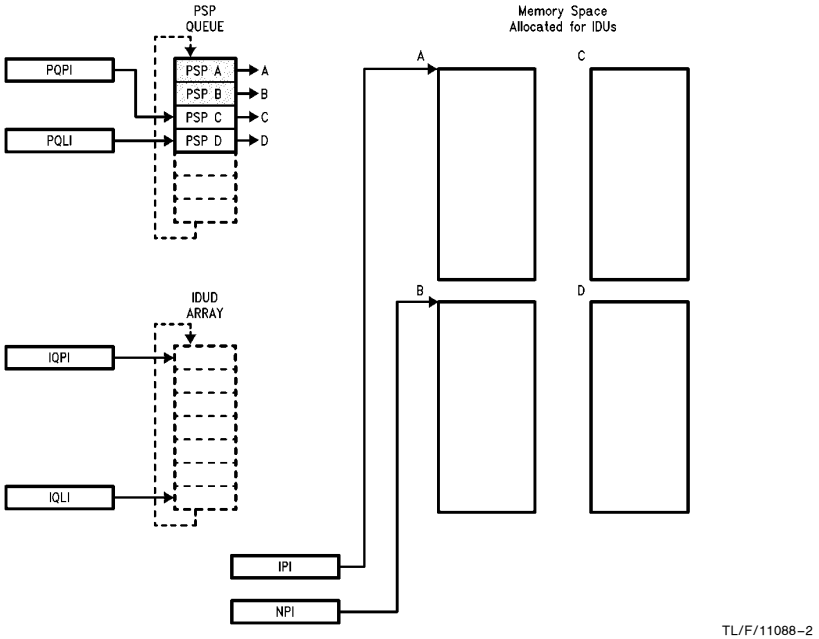
Please note that the BSI device will **not** hold the token unnecessarily when processing an open request object. It will only hold the token when explicitly instructed to do so, via the RQCLS field in the Request Descriptor (REQ).

6.0 RECEIVING FRAMES

This section describes how to process incoming frames, using the BSI device's data structures. It is assumed that the Indicate Mode Register (IMR), Indicate Threshold Register (ITR), Indicate Configuration Register (ICR) and Indicate Header Length Register (IHLR) have been previously configured. It is also assumed that the host has already selected a particular Indicate Channel for processing; perhaps by examining the attention register hierarchy.

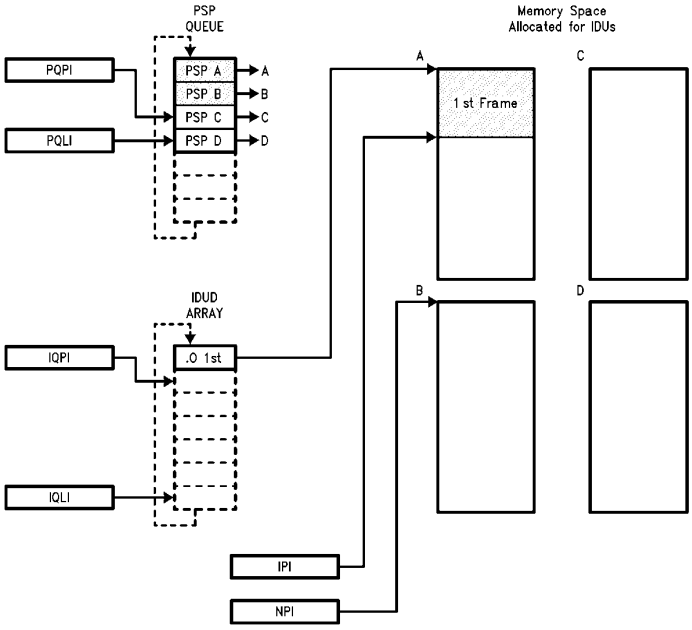
The host must maintain a minimum of two variables depicting the state of the Indicate machine: current IDUD queue pointer and the current buffer page address. To reduce accesses to the BSI device Control Bus, the host software may wish to also keep its own copy of the channel's Limit RAM Register.

For a visual description of the Indicate machine, see *Figures 2, 3,* and *4*.



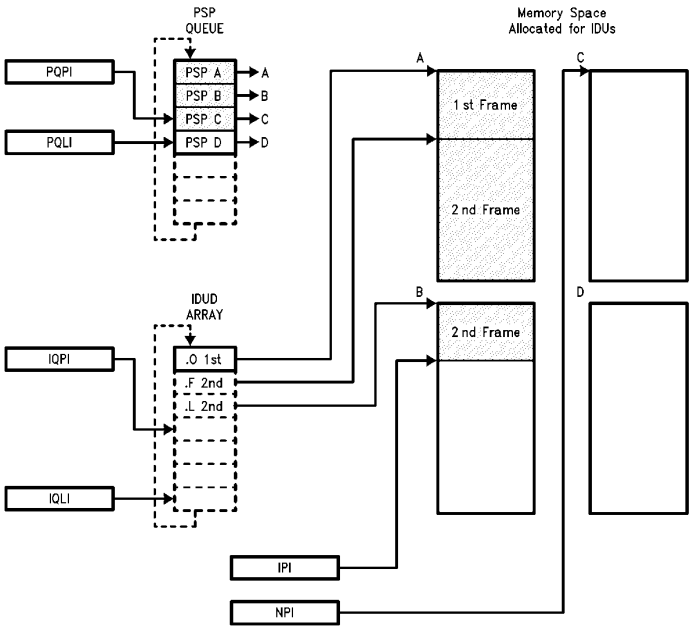
TL/F/11088–2

FIGURE 2. BSI Device Indicate Memory Structure



TL/F/11088–3

FIGURE 3. BSI Device Indicate Memory Structure



TL/F/11088–4

FIGURE 4. BSI Device Indicate Memory Structure

Figures 2 through *4* describe the operation of an Indicate Channel when receiving two frames.

Key for *Figures 2–4*

PQPI —PSP Queue Pointer
PQLI —PSP Queue Limit
IQPI —IDUD Queue Pointer
IQLI —IDUD Queue Limit
IPI —IDU Pointer
NPI —Next PSP Pointer
.O —Only
.F —First
.L —Last

6.1 Disable Interrupts for the Indicate Channel

Host manipulation of BSI device queues must be atomic; meaning, in this context, that only a single host agent (process, task, thread, etc.) may actively dequeue frames from a given Indicate Channel at a given time. In support of atomicity, four different granularities of interrupt masking may be achieved: host level, BSI device level, Indicate service level or Indicate Channel level. To mask interrupts at the Indicate Channel level, modify the Indicate Notify Register (INR) to clear the breakpoint and exception bits for the target channel.

6.2 Collect an Indicate Object

Indicate objects may be represented by one or more IDU Descriptors (IDUDs) on the given channel's IDUD queue. The host must maintain a queue pointer for the next queue position. When collecting an Indicate object the host should scan forward from this position until an entire Indicate object has been found. If a null descriptor is found in the first position, there are no Indicate objects on the queue. The beginning of an Indicate object is marked by an IDUD with the ''First'' bit set and, conversely, the end is marked by an IDUD with the ''Last'' bit set. An IDUD with both of these bits set (''Only'') completely describes an incoming frame. Note that it is the responsibility of the host to nullify descriptors after processing the frame.

6.3 Determine Acceptability of the Frame

There are three fields defined in the IDU Descriptor (IDUD) that are of use in determining the acceptability of a given frame. These fields are valid in the last IDUD in the Indicate object.

The Frame Status field may be examined to determine validity of the data length and FDDI FCS fields; as well as the values of the E, A, and C Indicators in the FDDI Frame Status field.

The Frame Attribute field can be queried to determine how the frame was recognized (MFLAG, AFLAG) and what the terminating condition was.

The Indicate Status field contains encoded status information (See Table IV).

TABLE IV. Indicate Status Codes

Code	Status
0x0	Last IDUD of Queue, Page Cross
0x1	Page Cross
0x2	Header End
0x3	Page Cross and Header End
0x4	Intermediate Frame
0x5	Burst Boundary
0x6	Threshold
0x7	Service Opportunity
0x8	Insufficient Data Space
0x9	Insufficient Header Space
0xa	Successful Header Copy, No Info Copy
0xb	No Info Space
0xc	FIFO Overrun
0xd	Bad Frame
0xe	Parity Error
0xf	Internal Error

6.4 Process the Frame Data

The Location (IDUD.LOC) and Byte Count (IDUD.CNT) fields in each IDUD describe the address and length of each Indicate Data Unit (IDU). There may be multiple IDUDs in a given Indicate object. The frame data referenced by these IDUDs must be logically concatenated to construct a single frame. The method of presenting frame data to upper level software is highly host dependent.

6.5 Reclaim Data Buffer Space

A 4 kbyte data page is available for reuse when the BSI device has filled it with IDUs **and** the host has finished processing all the IDUs on the page. The BSI device is guaranteed to consume space on a channel's PSP queue in a serial manner; thus it is possible to tell when the BSI device has finished using a page by detecting when the device starts filling a new page. When the host is also done processing all of the IDUs on the given page that data space may be reused. See Section 4 on Memory Management Schemes for ideas on managing buffer space for the BSI device.

6.6 Update IDUD Queue Pointers

After extracting all of the needed data from an Indicate Channel, that channel's IDUD queue may be updated to allow reuse of queue space. Three operations should be performed:

- Mark the processed IDU Descriptors (IDUDs) as null descriptors. A safe method is overlaying the eight byte queue slot with binary zeros.
- Update the host resident current IDUD queue pointer to point beyond the processed IDUDs.
- Update the Limit RAM Register for the given channel. If this value is maintained by the host, the value may be incremented and stored in the channel's Limit RAM Register; otherwise it is necessary to read the register first. The Limit RAM Operation (LMOP) is described above in Section 2.12.

6.7 Enable Interrupts for the Indicate Channel

Now that the given channel's queues have been updated, interrupts may be enabled again. If interrupts were masked at the Indicate Channel level, the Indicate Notify Register (INR) should be modified to set the breakpoint and exception bits for the target channel.

7.0 QUEUE MANIPULATION

The BSI device has two basic classes of queues: those that it uses to consume descriptors (REQ, PSP) and those that it uses to produce descriptors (IDUD, CNF). Conversely, the host software must consume descriptors (IDUD, CNF) and produce descriptors (REQ, PSP) on the opposite queues.

For Request Channel operation the BSI device **reads** from the channel's Request Descriptor (REQ) queue and **writes** to the channel's Confirmation Message (CNF) queue. For Indicate Channel operation the BSI device **reads** from the channel's Pool Space (PSP) queue and **writes** to the channel's Indicate Data Unit Descriptor (IDUD) queue.

It is necessary to understand how the BSI device interacts with each type of queue so that one may design host software that interacts with the queues in a complementary fashion. When writing software that interfaces with the BSI device, one minimally needs to understand the following:

- How the queues are organized
- How to initialize each type of queue

- How to handle queue ''wraps''
- How to detect boundary conditions (empty queue, full queue)

7.1 Queue Organization

A BSI device queue consists of a contiguous block of BSI device addressable memory logically sub-divided into eight byte queue slots. Queues sized at 1 kbytes must be aligned on 1 kbyte boundaries, while queues sized at 4 kbytes must be aligned on 4 kbytes boundaries.

The BSI device embodies two indexing variables for each queue: a pointer to the **next** available queue position to read or write (stored in BSI device Pointer RAM) and a queue limit (stored in BSI device Limit RAM). The BSI device increments the pointer variable after reading from a queue or writing to a queue. The host software may control channel operation by manipulating the value of the limit variable. With this in mind there are a few simple rules governing queue manipulation by the BSI device.

1. Pointer and limit variables reference a given queue slot by pointing to the first word of the descriptor.
2. The pointer variable always points to the **next** available position on the queue.
3. The BSI device always increments the pointer variable **after** a read or write operation.
4. The BSI device halts channel processing when the pointer and limit variables are logically **equal**.
5. The BSI device tests for pointer/limit equality **during** queue read operations (REQ, PSP) and **after** queue write operations (IDUD, CNF). When detecting pointer/limit equality during a read operation, the current read operation and no further read operations are made.
6. Read operations are triggered by Limit RAM updates.

There are also some special considerations caused by the pipelined processing of CNF, IDUD and PSP Descriptors.

- The BSI device may generate two additional CNF/IDUD Descriptors after detecting pointer/limit equality. It is necessary to set the limit value such that it references the penultimate available queue slot.
- Each active Indicate Channel should have at least two buffers placed on its PSP queue. With only one buffer, the BSI device will immediately raise the Low Data Space attention bit for that channel.

The host software must maintain its own pointer and limit variables for each queue. For REQ and PSP queues, the limit variable should reference the next available queue slot for writing a new descriptor; while the pointer variable should correspond to the pointer variable on BSI device. For CNF and IDUD queues, the software pointer should reference the next queue slot to be used when reading a new descriptor. The software limit variable must always reflect the limit variable on the BSI device. The software pointer variable must be maintained independently from the BSI device queue pointer.

7.2 Queue Initialization

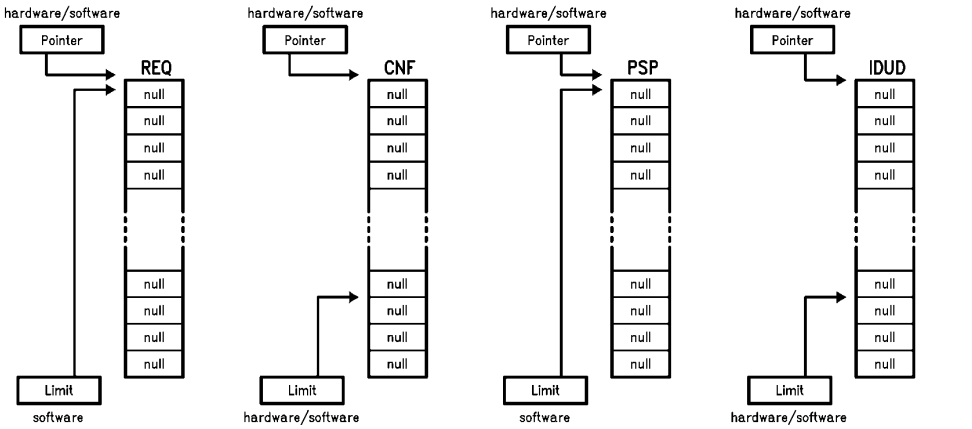
To initialize queues that the BSI device reads (REQ, PSP) simply set the pointer (BSI device and software versions) and software limit to reference the first queue slot. Do not update the queue's Limit RAM Register until actually queuing the queue's first descriptor.

To initialize queues that the BSI device writes (IDUD, CNF) one must set the limit variable to reference the penultimate available queue slot (required due to pipelining). For example, to make all but one of the queue slots available one could set the pointer variable to reference the first queue slot and the limit variable to reference the ''next to the next to the last'' queue slot. Also, one should clear the queue area by overwriting it with binary zeroes; effectively marking all queue slots as ''null descriptors''.

See *Figure 5* for a pictorial description of initialized queues.

7.3 Queue ''Wraps''

Upon reaching the end of the queue memory block, queue indexing variables ''wrap'' to the beginning of the queue memory block. The BSI device automatically performs queue ''wraps'' for pointer variables; while the host software must perform queue ''wraps'' for limit variables and software queue pointers. The method for calculating the next



TL/F/11088–5

FIGURE 5. Suggested Queue Initialization

queue position is dependent upon the form of data representation that the host software uses (i.e., BSI device addressable pointers, queue byte offset, . . .). For example, when representing the limit variable as a queue offset one could use simple modulo arithmetic. When the limit variable is maintained as a pointer into BSI device addressable memory the host software might use the following method to increment the variable (specified with C code using 4 kbyte queues).

new = ((old + 8)& 0xfff) + (old & 0x0ffff000)

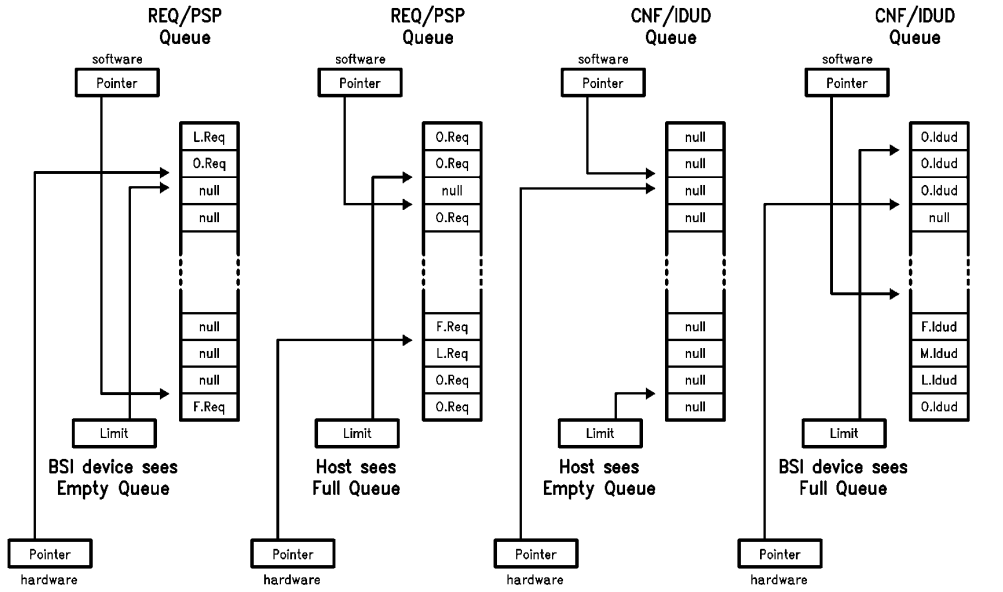
7.4 Detecting Boundary Conditions

The BSI device detects both queue empty (when reading) and queue full conditions (when writing) by testing for pointer/limit variable equality. As noted above, this test is done during read operations and after write operations.

When reading, a queue empty condition **cannot** be determined by comparing the pointer and limit variables. Instead the host software may recognize the presence of a ''null descriptor'' on the queue. To ensure that there will always be at least one ''null descriptor'' to demarcate the queue empty condition; the host software must never set the limit variable to indicate that all of the queue slots are available and must mark each available queue slot as a ''null description'' (binary zeroes are recommended).

The host software can detect a queue full condition using the same basic mechanism as the BSI device. When writing, a queue may be considered full when the software limit pointer references the queue slot immediately ''before'' the slot referenced by the software pointer variable. Queue ''wraps'' must be taken into account in the queue variable arithmetic.

See *Figure 6* for a graphical representation of the various queue boundary conditions.



TL/F/11088–6

FIGURE 6. Queue Boundary Conditions

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: 1(800) 272-9959
TWX: (910) 339-9240

National Semiconductor GmbH
Livry-Gargan-Str. 10
D-82256 Fürstenfeldbruck
Germany
Tel: (81-41) 35-0
Telex: 527649
Fax: (81-41) 35-1

National Semiconductor Japan Ltd.
Sumitomo Chemical
Engineering Center
Bldg. 7F
1-7-1, Nakase, Mihama-Ku
Chiba-City,
Ciba Prefecture 261
Tel: (043) 299-2300
Fax: (043) 299-2500

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductores Do Brazil Ltda.
Rue Deputado Lacorda Franco
120-3A
Sao Paulo-SP
Brazil 05418-000
Tel: (55-11) 212-5066
Telex: 391-1131931 NSBR BR
Fax: (55-11) 212-1181

National Semiconductor (Australia) Pty, Ltd.
Building 16
Business Park Drive
Monash Business Park
Nottinghill, Melbourne
Victoria 3168 Australia
Tel: (3) 558-9999
Fax: (3) 558-9998