7814/7851/7854
Streaming Bus
Application Note

Hifn

Intelligent Secure Networking

Hifn™ supplies the Internet's most important raw materials for the creation of intelligent and secure networks: compression, encryption, and flow classification. This is central to the growth of the Internet, helping to make electronic mail, web browsing, Internet shopping and multimedia communications better, faster and more secure.

750 University Avenue
Los Gatos, CA 95032
info@hifn.com
http://www.hifn.com
Tel: 408-399-3500
Fax: 408-399-3501

For technical support, please contact your local Hifn sales office, representative or distributor. For locations check: www.hifn.com.

Disclaimer

Hifn reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

Hifn warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with Hifn's standard warranty. Testing and other quality control techniques are utilized to the extent Hifn deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

HIFN SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of Hifn products in such critical applications is understood to be fully at the risk of the customer. Questions concerning potential risk applications should be directed to Hifn through a local sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

Hifn does not warrant that its products are free from infringement of any patents, copyrights or other proprietary rights of third parties. In no event shall Hifn be liable for any special, incidental or consequential damages arising from infringement or alleged infringement of any patents, copyrights or other third party intellectual property rights.

"Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals," must be validated for each customer application by customer's technical experts.

The use of this product in stateful compression protocols (for example,. PPP or multi-history applications) with certain configurations may require a license from Motorola. In such cases, a license agreement for the right to use Motorola patents may be obtained through Hifn or directly from Motorola.

This product must be exported from the United States in accordance with the Export Administration Regulations. Diversion contrary to U.S. law prohibited.

Table of Contents

1 Overview

1.1 Scope of this Document

This application note contains information about the 7814, 7851, and 7854 streaming bus interface which may be helpful in understanding the streaming bus and determining its suitability for a particular application. Interface architectures and system level design considerations discussed within this document are intended for use as a general guide when designing with the 78xx security processors.

Hifn introduced the streaming bus host interface beginning with the 7851 security processor. Subsequently, the entire 78xx family of security processors adopted the streaming bus interface. The 7814, 7851, and 7854 all share a common architecture, a common package, and a common pinout.

The reader is assumed to have a general knowledge of the architecture of the 7814, 7851 and 7854. Refer to the *7814_7851_7854 Device Specification*, DS-0030 for more information on these devices.

1.2 Streaming Bus Description

The streaming bus is a synchronous FIFO based interface that contains two independent 32-bit buses, one for input and one for output. The interface is designed so that a host network processor may interface to as many as four 78xx devices which may share a common streaming bus interface. From the host processors perspective, the 78xx streaming bus interface is a bus slave.

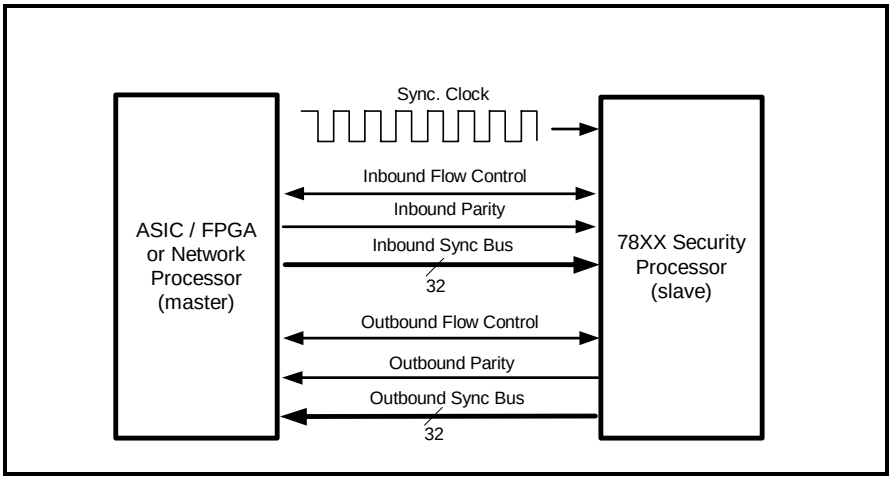

Figure 1. The Streaming Bus Interface

In 78xx security processors the streaming bus interface is considered an alternate to the PCI bus since both interfaces share common pins. Only one interface can be active at the same time. If the streaming bus is enabled, packets must flow over the streaming interface. Otherwise, they flow across PCI, controlled by the inbound and outbound DMA units.

Since only one interface may be used at a time, there are bound to be unused pins in the 78xx application. In streaming bus mode the unused PCI inputs should be pulled to their inactive state or pulled up to VDD through resistors. Any value between 1K and 50K ohms will work. The unused outputs may be left unconnected.

The most important distinction between the streaming bus interface and the PCI or private CPU interface is that the streaming bus is a specialized mechanism for transferring packets and requests to the security processor. The streaming bus does not directly support read or write access to processor registers, private memory, or the public-key core. Unlike the PCI interface, the streaming bus does not contain DMA units.

The following figure illustrates how data flows through the Inbound and Outbound ports of the 78xx streaming bus interface. Incoming packets ingress through the Inbound streaming port and consist of a command message, followed by source descriptors, and source data fragments. The processed packets egress through the Outbound port and consist of a result messages followed by destination data. Notice that the command message contains a field NS which defines the number of source fragments in the packet.

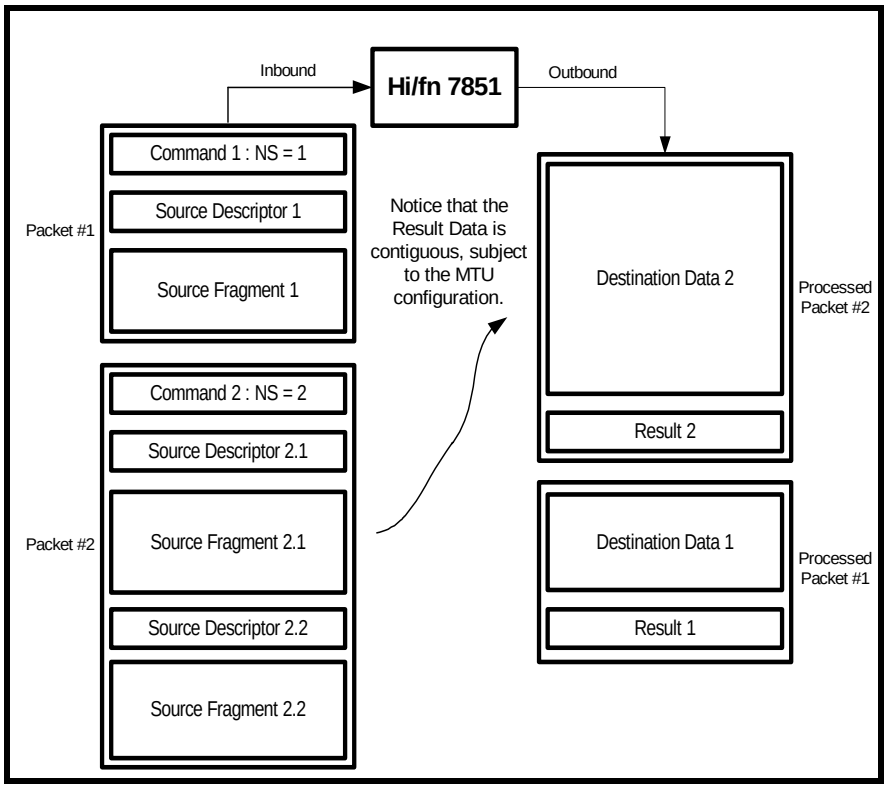

Figure 2. Packet Flow in Streaming Bus Mode

In streaming bus mode, the inbound bus is designed to transfer command messages and unprocessed packet data into the 78xx processor. The outbound bus transfers the processed packet data and result messages back to the host. The streaming bus interface is considered to be high-speed since it eliminates

the overhead of PCI and supports simultaneous bi-directional data transfer. The figure below depicts a single 7854 application using the streaming bus interface.

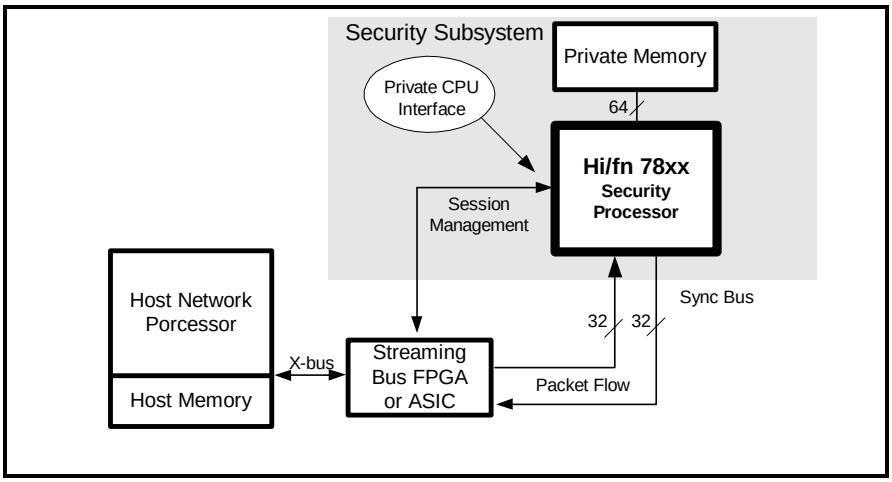

Figure 3. 78xx Application with Streaming Bus Interface

In this figure packets flow over the Inbound and Outbound streaming bus while session management takes place over the private CPU interface. In streaming bus applications the private processor interface should be used to initialize 78xx registers, load DPU programs, perform all session management functions, and handle security processor exceptions. In 7854 and 7814 streaming bus applications the private processor interface should also be used for host access to the public key engine.

The following sections of this document will discuss architectural considerations which apply to 78xx systems which may utilize the streaming bus interface. These considerations include host interface methodology, the placement of the FIPS security boundary, optional use of the private processor, options for Hifn's HSP session management software, and recommendations for implementing multiple 78xx devices on a shared streaming bus interface. This document also contains information about streaming bus interface signaling, flow control, and session setup. The last section of this document contains a streaming bus example.

2 Designing with the Streaming Bus

2.1 Choosing between the PCI and Streaming Interface

System performance and host interface architecture are the primary considerations when choosing between either the PCI or streaming bus modes. Systems which employ PCI architectures may naturally lend themselves toward using the PCI interface. On the other hand, systems which require several 78xx security processors and performance beyond PCI may be better suited for using a shared streaming bus.

In PCI systems packets ingress and egress over the same bus. In streaming bus systems packets simultaneously flow through separate 32-bit wide Inbound and Outbound data paths at speeds up to 2 Gigabits per second (4 Gigabits/sec. full duplex) maximum at 66 Mhz. Additionally, the streaming bus is a fully dedicated interface. Packets flow on demand. And, unlike PCI, there is no bus arbitration.

Either the streaming bus or PCI may be used in multi-chip 78xx systems. Systems with more than two 78xx processors may be forced to use the streaming bus mode. Of course this assumes that multiple 78xx processors will share either a single PCI or streaming bus. While multiple PCI and/or streaming busses could be used in the system, a single shared bus is assumed to be more desirable.

2.2 Multi-chip Streaming Bus Applications

Applications which require higher overall performance may connect up to four 78xx devices to a single streaming bus interface. The Outbound streaming bus interface is tri-statable and the Inbound bus is inactive if the sync_write signal is not asserted. The following figure contains an example application which utilizes multiple 78xx processors which share a single streaming bus interface.

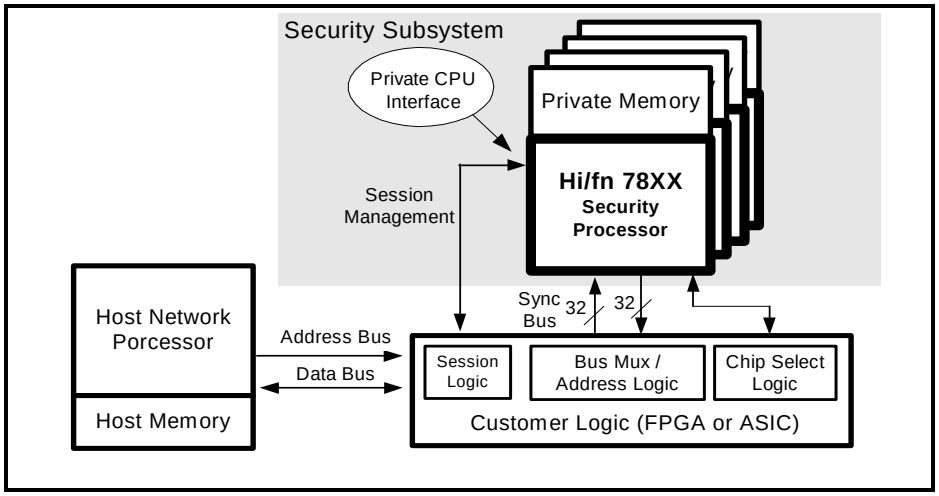

Figure 4. Multichip Application with shared Streaming Bus Interface

Shared streaming bus applications require some provision for the host to address individual devices on the shared bus. This may be accomplished several ways. The two most common ways are to memory map the individual 78xx devices (as shown) or utilize the pass-through field in the command message. Again, host architecture may lend itself more strongly toward one or the other. If the host interface to FPGA/ASIC is similar to the streaming bus, the host could easily embed an address into the pass-through field. These bits could subsequently be decoded by glue logic and used to select a 78xx device from the shared streaming bus by asserting the sync_read and sync_write signals for the desired target. If the host interface to FPGA/ASIC is a traditional address and data bus, a memory mapped addressing mechanism may be more appropriate.

2.3 Streaming Mode Boot Sequence

In streaming bus mode the host is required to perform all of the private processor functions including the 78xx device initialization sequence. As with session management in streaming bus mode, the initialization process takes place over the private processor interface. This process must be performed before any sessions are opened in 78xx private memory. The sequence involves programming all 78xx registers to their desired operating state, which may reset, enable, and disable certain logic blocks. Then all required DPU programs must be loaded into private memory.

The actual register initialization values are application specific. Refer to the *7814_7851_7854 Device Specification* for register descriptions. Also, the *7851 Verilog Model Application Note*, AN-0022 and the 7814/7854 Verilog Model Application Note, AN-0048 provide complete examples of initialization and packet processing in both PCI and streaming modes.

2.4 Session Management in Streaming Bus Mode

This section describes how session handling may be accomplished over the streaming bus interface (without a private processor). Session setup must be performed by either the private processor or the host network processor. Session setup is never performed by the 78xx. The following session setup tasks must be performed before any packets for the session are sent across the streaming bus :

- An unused session number must be allocated to the session.
- The session context must be initialized to appropriate values. Session context resides in private SDRAM memory and consists of small session context and optional large session and compression context(s).
- The DPU program(s) that are going to process the packets in the session must be loaded into private memory. This task is typically done only once at the beginning of several similar sessions.

These tasks will all take place through the private processor interface. They are essentially made up of private memory reads and writes which take place through the private processor interface. For this to be accomplished in a streaming bus application, interface logic between the host and 78xx device must drive memory transactions through the private processor interface. In the most basic sense, this may be accomplished through the generation of address, data, and control signals which mimic session handling as it would be performed by a private processor. The following figure illustrates the concept of a host interface to 78xx private processor interface through a FPGA device.

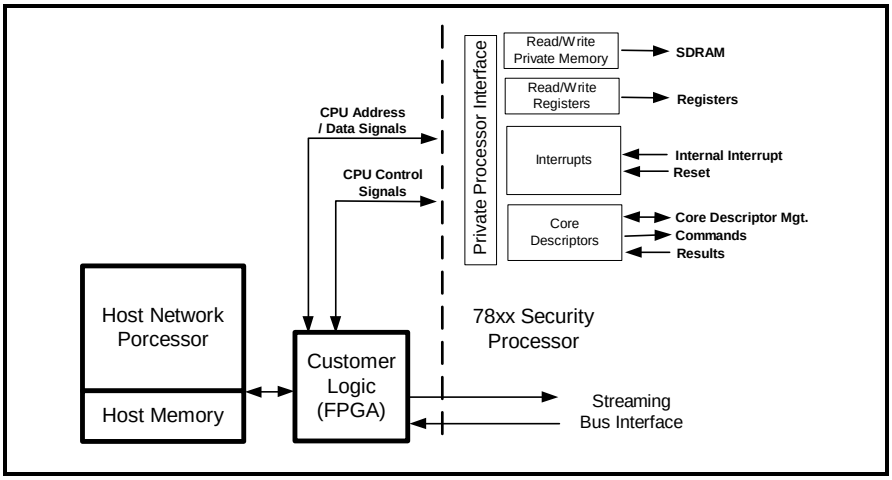

Figure 5. Host Interface to 78xx Private Processor Interface

2.4.1 Using the Trap Queues for Session Management

The Trap_out_q and Trap_in_q registers are designed for moving core descriptor indexes to and from the private processor and 78xx DPU core. These queues enable the private processor to transfer and receive security/compression tasks to and from the 78xx DPU core.

There can be hidden difficulties when using these queues in streaming bus applications where the host is required to perform all of the regular private processor duties. For example, assume that the host traps an operation to the DPU core. The DPU then completes the required processing and initiates a trap back to the private cpu (host processor in this example). If the host doesn't respond immediately, or mishandles the return trap, all processing for the associated session may be stalled. As a result, Hifn strongly recommends that our customers either do not try to manipulate the trap queues or use Hifn reference software for these functions. Such software is subject to Hifn availability. Contact your Hifn representative for more information.

2.5 Private Processor Interface in Streaming Bus Mode

The following diagram illustrates how host interface logic may be connected to the 78xx private processor interface. A bus controller must be generated in the customer logic block if more than one 78xx device is being addressed on the same private interface bus. Example bus control logic is shown.

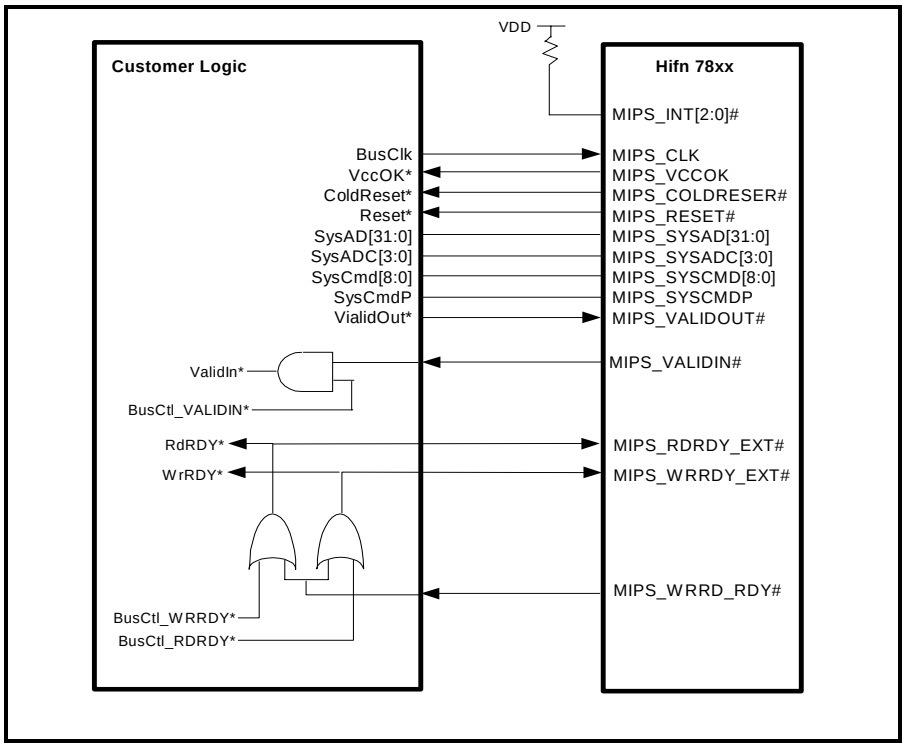

Figure 6. Private Processor Interface in Streaming Bus Mode

2.6 Load Balancing with multiple 78xx devices

When multiple 78xx security processors share a single streaming bus, various load balancing schemes may be used to help ensure a high degree of overall system utilization. One of the simplest load balancing methods is to statically assign certain blocks of session numbers to each 78xx device on the shared streaming bus. Using this method, incoming packet session numbers are used to route entire packets to the appropriate 78xx device. This method alone may be sufficient for systems which have no requirement for the ordering of processed packets.

In some systems it may be necessary to export packets in the same order they are received. In this situation the host may build and maintain an index table whereby Inbound packets may be assigned an identifier which records which packet processor was assigned each particular command. Head and tail pointers in the index table may be used to read result messages and retire packets in the same order they are received. And, if the pass through field in the command message contains the index number, the result message, which will also contain that index number, may be used as an error checking mechanism to ensure synchronicity is maintained with the stream. Other more elaborate methods exist, but depend largely on the software and hardware capabilities of the end application.

2.7 Signaling Considerations in Streaming Bus Mode

2.7.1 Deasserting the sync_write signal mid-stream

Streaming bus operation and timing requirements are described in the *7814_7851_7854 Device Specification*, DS-0030. Streaming bus writes can be contiguous or non-contiguous. For example, when the host processor is writing words that make up a command descriptor it is not necessary to deassert the write signal before writing data words. It is, however, acceptable to pause mid-cycle by deasserting the write signal. The write process may resume on any future clock cycle by reasserting the write signal.

The following figure illustrates a typical write cycle for a command message. In this example one of the SYNC_IN_LEVEL signals indicates that the Inbound FIFO has reached some predefined threshold after the first word is written. The host subsequently deasserts the write signal and waits for the Inbound level indicator to transition back to zero and signify that there is room in the Inbound FIFO for writing additional words. In the given example, the SYNC_IN_LEVEL signal may be assumed to be set at some unknown arbitrary value. Also, it is assumed that a sufficient amount of data has already been written to the Inbound FIFO so that the threshold for the level signal is met or exceeded by the first word of the write transaction. The next section describes the FIFO level signals in greater detail.

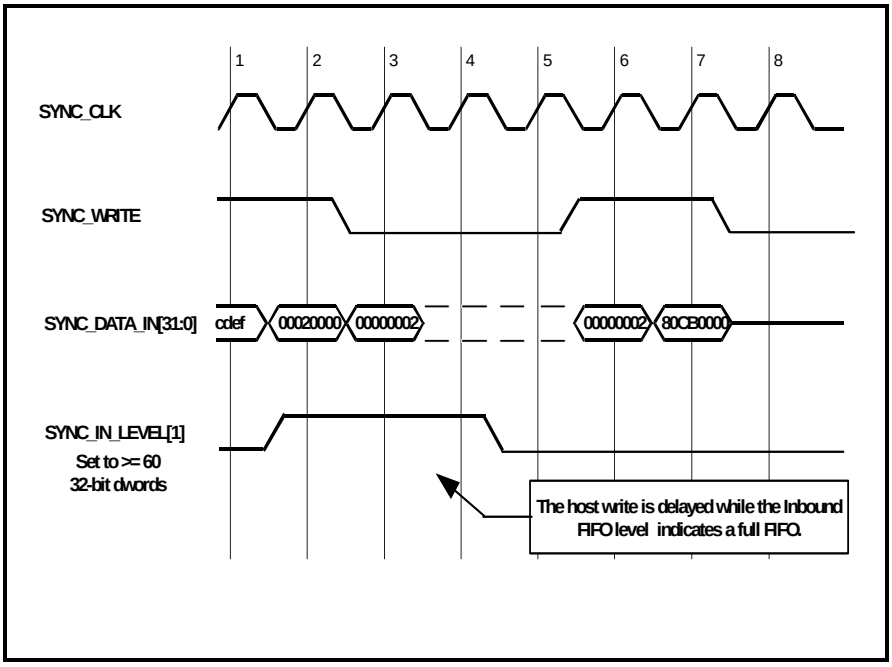

Figure 7. Example Streaming Bus WRITE with 3-cycle pause.

The same conditions apply to reading data from the streaming bus. The read signal may be deasserted at any time during the sequence when words are being read. The read process may resume once the read signal is reasserted at a later time. The following figure illustrates a typical streaming bus read transaction

where the Outbound FIFO level signals are used to gate single and multi cycle read sequences.

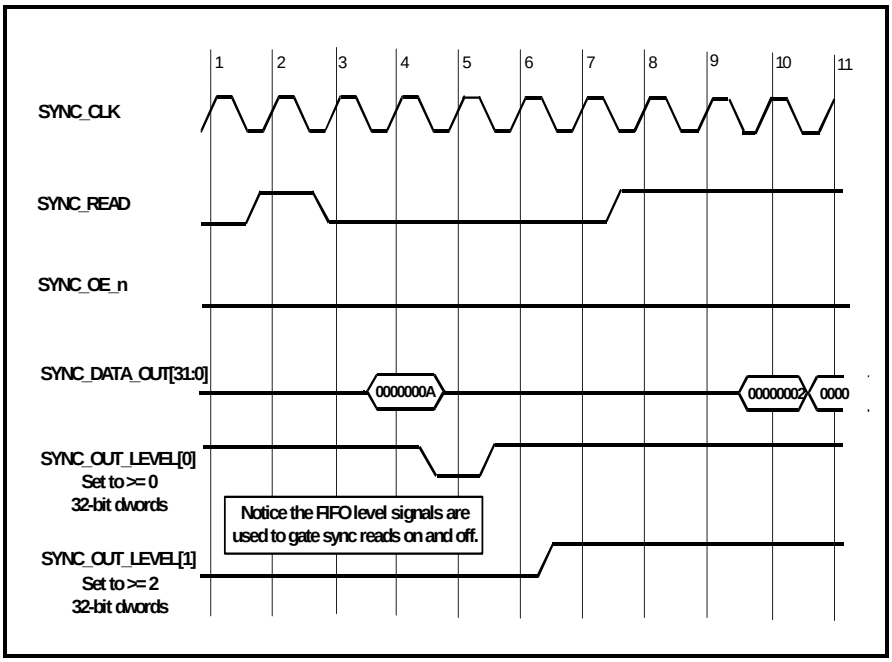

Figure 8. Example Streaming Bus READ with pause.

Notice that the read signal is asserted for one cycle and then deasserted for 5 subsequent cycles. In this example the Outbound FIFO initially has only one word available, which is read at cycle 4. Later, the SYNC_READ signal is reasserted in cycle 8 since the FIFO level signals, which are described in the next section, indicate that there are more words in the Outbound FIFO.

2.7.2 FIFO level signals

The host may use register programmable FIFO fullness signals to determine when there is data in the Inbound or Outbound FIFOs. The 78xx provides 3 signals for Inbound FIFO level detection and 3 signals for Outbound FIFO level detection. These signals may be used by the host to gate inbound and outbound read and write bursts for maximum throughput with minimum host intervention. A typical implementation may program one level signal to indicate a FIFO full condition which would gate off all host transfers to prevent FIFO overrun. A second FIFO level indicator may be programmed to signal that the FIFO is at some intermediate fullness which may correspond to the maximum burst transfer size of the host. Then the third FIFO fullness signal could be programmed to signal a FIFO empty condition that may interrupt the host and trigger a burst read or write so that the 78xx processor achieves 100% utilization.

It is important to recognize that there is a cycle delay between a level change in the Outbound FIFO and the update to the associated level signal(s). On the Inbound FIFO there is a 3-cycle delay between writing a dword and any updates to the associated FIFO level signals. Assuming the output enable is active, on every clock in which the read signal is active, data is read from the Outbound FIFO and is driven onto the output pins two cycles later. The Outbound FIFO

levels are driven three cycles after the sync_read is asserted. See the 78xx device spec for the timing diagram of a Sync read.

Software routines that service the Inbound and Outbound FIFOs should be written so that command, source and destination data, and result messages are written and read in their entirety. Since destination data length is independent of the command or source data length, Outbound read routines should not be written solely for fixed length multi-word block reads. For optimum Outbound efficiency, it may be helpful to program one of the Outbound FIFO level indicators to four (level=4) and another to one (level=0). The host may read blocks of four when the level=4 signal is asserted. When the level=4 signal subsides, the host may poll the level=1 signal to finish up reading the result message. This may prevent a single destination word from remaining in the Outbound FIFO without being read by the host.

Managing the Outbound level indicator signals and associated read cycles may be somewhat more complicated than those of the Inbound side. This is because it is generally easier to write large blocks to the Inbound FIFO at one time without having to manage individual dword write sequences. In fact the host may make burst writes to the Inbound FIFO until it is gated off by the SYNC_IN_LEVEL signal(s). The Outbound FIFO works the same way except that the host must take into consideration that there are delays between asserting the read signal and reading from the Outbound FIFO and the update to the Outbound FIFO level signal(s). The following figure illustrates how the Outbound FIFO level signals may be used to gate a block read and subsequent individual read cycles.

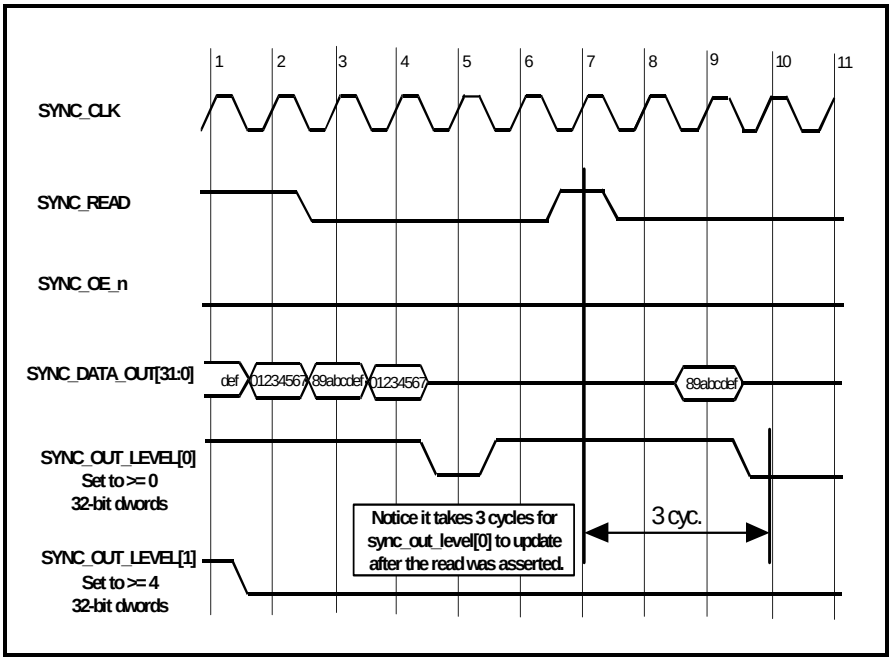

Figure 9. Example use of Outbound FIFO level signals.

Notice in cycle 1, the SYNC_OUT_LEVEL[1] signal transitions low to tell the host that there are 3 or less dwords remaining in the Outbound FIFO. Since there is a cycle delay between a FIFO level change and an update to the FIFO level signal, this condition must have began on the last cycle, which was a read

cycle. In response, the host must immediately deassert the read signal (in cycle 2) to prevent a read underflow condition. The last two dwords are read in cycles 3 and 4 due to the 2-cycle delay between deasserting the read signal and the final word being read. In this particular situation the SYNC_OUT_LEVEL[1] signal gated off the host read process just in time to read all of the data from the Outbound FIFO. In cycle 6 the SYNC_OUT_LEVEL[0] signal tells the host that there is at least one more dword waiting in the Outbound FIFO. It is read in cycle 9. Also notice that it takes 3 cycles for the SYNC_OUT_LEVEL[0] signal to update from the time the read signal was asserted by the host.

The Inbound FIFO level signaling is slightly simpler. There is a 3-cycle delay between writing the data and the update to the level signal(s). The following figure illustrates how the Inbound FIFO level signals may be used to gate off a burst write to the 78xx device.

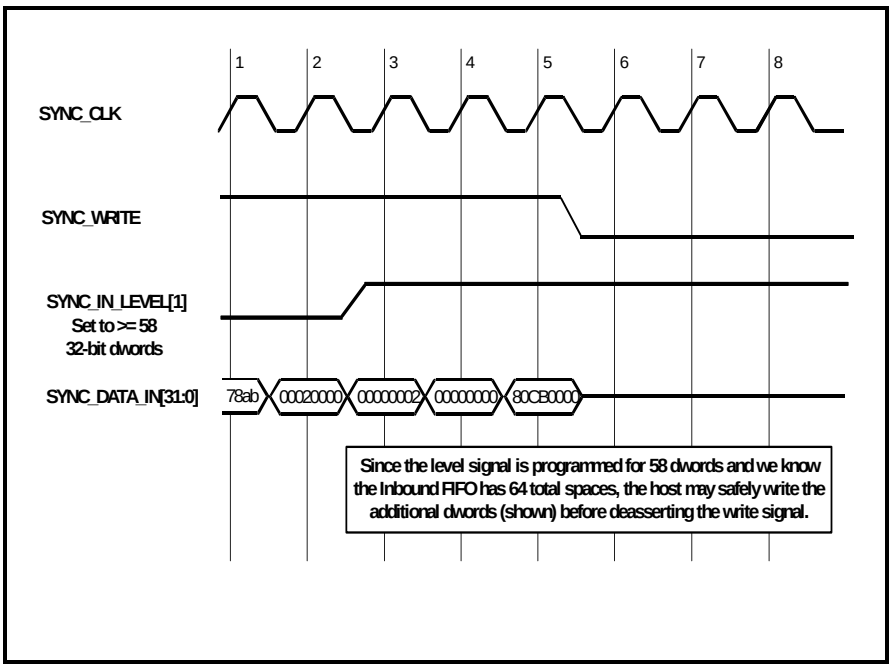

Figure 10. Example use of an Inbound FIFO level signal.

2.7.3 Overflow and Underflow

Inbound FIFO overflow and Outbound FIFO underflow both require a reset of the internal FIFO control logic. A status bit in the pci_stat register is set if the host tries to write data to the Inbound FIFO when it is full. If enabled in the pci_int_mask register, this status bit will generate an interrupt on the sync_interrupt pin to signal the overflow. Reading an empty outbound FIFO is handled the same way. When the host tries to read from an empty Outbound FIFO, a separate status bit is set in the pci_stat register.

Data is lost when the host tries to write to an already full Inbound FIFO. Similarly, invalid stale data will be read from a empty Outbound FIFO. In both situations, the internal pointing mechanism of the FIFO becomes corrupt and must be reset by writing to the appropriate reset bit in the Inbound and

Outbound control registers. For the Inbound FIFO this bit is located in the Cmd_ring_ctl register. For the Outbound FIFO this bit is located in the Rslt_ring_ctl register.

2.8 Commands and Headers

2.8.1 Zero Length Source Descriptors

A command message may contain zero associated source descriptors (NS=0). In this situation the command has no data and may immediately be followed by a new command. However, on Inbound FIFO writes it is important to realize that the number of bytes in a source descriptor cannot be zero. Software write routines that are designed to handle blocks of source fragments should not be written in such a way that zero sized source descriptors are used to fill up remaining words to finish out a block transfer or fill a boundary.

2.8.2 Pass a Header Through the 78xx

Some applications require in band communications between subsystems. In streaming bus mode, a 78xx DPU program could allow a variable length header to pass through the security processor unprocessed. Such software is subject to Hifn availability. Contact your Hifn representative for more details. The following diagram illustrates a customer header which is surrounded by command and packet data.

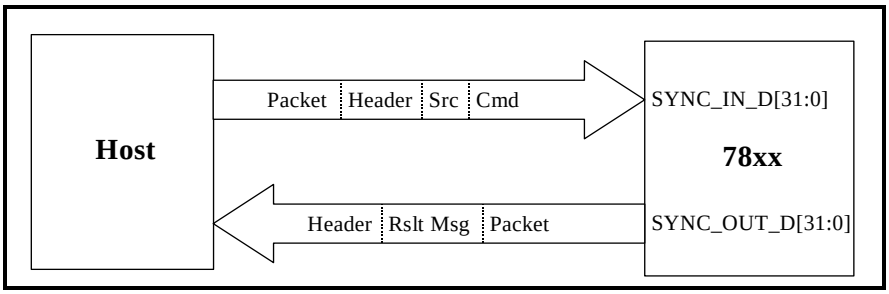

Figure 11. Example Header Use

3 HSP in Streaming Bus Mode

For most 78xx streaming bus implementations, customers will use Hifn's Software Development Kit (SDK).

78xx HSP software is designed only for PCI applications. When using HSP, the private processor is responsible for performing session setup and management through the private processor interface. Session setup involves programming session context into private memory prior to the security processor ever receiving or processing packets associated with the new sessions. This alleviates a significant amount of processing from the host CPU.

In some 78xx streaming bus applications it may be possible to use HSP. In this situation session management communications would likely flow through the private processor bus. However, HSP software in this type of application is subject to Hifn availability. Contact your Hifn representative for more information. The following diagram illustrates this type of system. To protect the security boundary afforded by HSP, the private processor interface block on the FPGA or ASIC should be designed to be a slave only. If it were a master block with arbitration, the security boundary would encompass the host network processor since it would have access to key material.

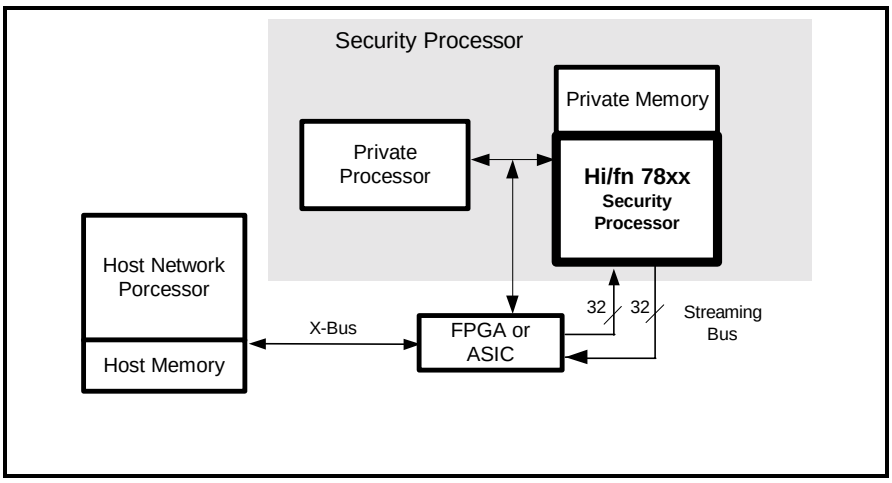

Figure 12. 78xx Streaming Bus w/ Private Processor and HSP

3.1 The Security Boundary

The private CPU and HSP software make a security boundary possible and offload exception processing and session setup and teardown overhead from the host. It also simplifies the host software by encapsulating most of the security software in the security subsystem. The following figure highlights a FIPS 140-1 security boundary in a streaming bus application which uses one 78xx device with private processor present.

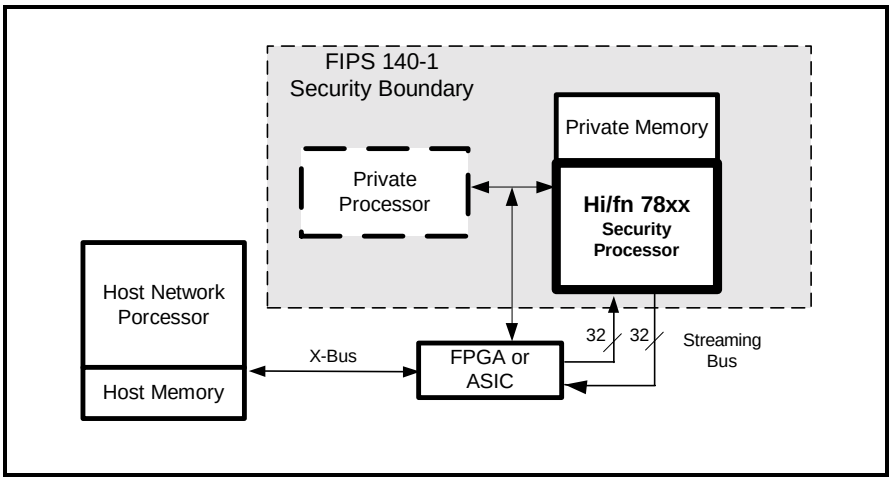

Figure 13. 78xx Security boundary (with private processor present)

The private processor, running HSP software, would have direct access to private memory session context, including key material. Assuming the private processor is present, this sensitive information is contained within the security boundary as shown. However, HSP in this type of application is subject to Hifn availability. Contact your Hifn representative for more information.

If the private processor is not present, the host is required to perform all of the private processor tasks. In this situation the security boundary expands to include the host CPU, host memory, and any interface logic, which is used to connect to the private processor interface. In applications without the private processor the host CPU and host memory will have access to session context and contain key materials. The following figure shows how the security boundary is expanded in systems which have no private processor.

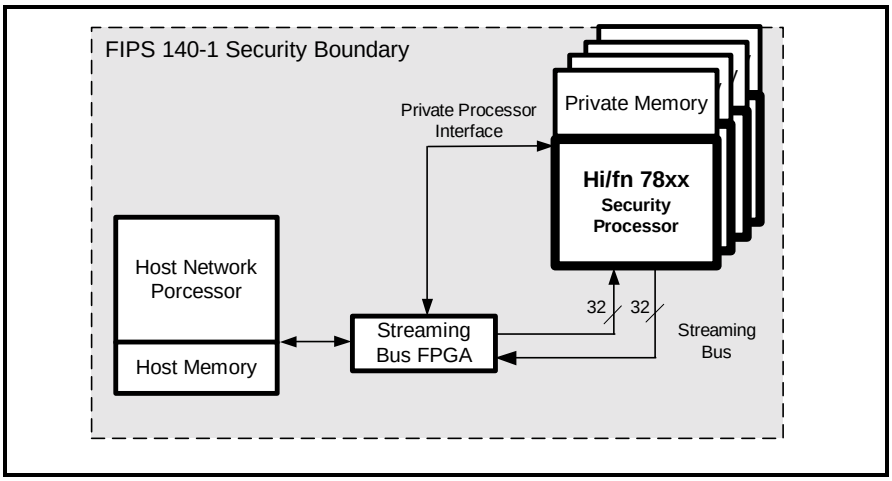

Figure 14. Security boundary (without a private processor present)

4 Streaming Bus Example

The 78xx streaming bus is described in this section using an example.

We assume the 78xx is configured in Streaming bus mode, is out of reset and a session is set up in the SDRAM for the desired compression or encryption operation. We assume the required DPU program is loaded in SDRAM at function code offset 0. We assume that all Sync interrupts are disabled and we ignore parity.

4.1 Streaming Bus Initialization

Initialization of the Sync Interface requires register writes to **cmd_ring_ctl** and **rslt_ring_ctl**.

cmd_ring_ctl is written with **32'h7C3C_0018.** Referring to the device specification we see that this value means the Inbound FIFO is out of reset, the command message endian is set to 32-bit big and the Inbound FIFO is enabled. Inbound parity is disabled and the inbound bus mode is 32. In_FIFO_Level_0 = 31, In_FIFO_Level_1 = 15 and In_FIFO_Level_2 = 0.

In_FIFO_Level_0 = 31 means **SYNC_IN_LEVEL[0]** is 0 when the Inbound FIFO is empty. It is not used in this example.

In_FIFO_Level_2 = 0 means **SYNC_IN_LEVEL[2]** is 0 when the Inbound FIFO is not full. It is not used in this example.

In_FIFO_Level_1 = 15 means **SYNC_IN_LEVEL[1]** is 0 when the Inbound FIFO has >= 30 dwords of free space in it. It is used in the following write rule:

- If **SYNC_IN_LEVEL[1] = 0** then write, else do not write.

This rule ensures that we can burst 16 qwords (32 dwords) to the 78xx without Inbound FIFO overflow. This is just one example. Many other write rules could be designed.

rslt_ring_ctl is written with **32'h0008_7C18**. Referring to the device specification we see that this value means the Outbound FIFO is out of reset, the result data endian is set to 32-bit big and the Outbound FIFO is enabled. Outbound parity is ignored and the outbound bus mode is 32. Out_FIFO_Level_0 = 0, Out_FIFO_Level_1 = 2 and Out_FIFO_Level_2 = 31.

Out_FIFO_Level_2 = 31 means **SYNC_OUT_LEVEL[2]** is 0 when the Outbound FIFO is full. It is not used in this example.

Out_FIFO_Level_1 = 2 means **SYNC_OUT_LEVEL[0]** is 0 when the Outbound FIFO contains 4 or more 32-bit dwords in it.

Out_FIFO_Level_0 = 0 means **SYNC_OUT_LEVEL[1]** is 0 when the Outbound FIFO is not empty. **SYNC_OUT_LEVEL[1]** will be 1 when the Outbound FIFO is empty. This signal is polled to prevent read underflow when the host is reading out the last dwords from the FIFO.

The read rule is:

- If **SYNC_OUT_LEVEL[1]** = 0 then read and do not wait.
- Else if **SYNC_OUT_LEVEL[0]** = 0 then read and wait 2 clocks.

- Else do not read.

This rule ensures that we can perform continuous reads while there are in excess of 2 dwords from the 7851 without the danger of reading from an empty FIFO. When there is only one dword in the Outbound FIFO, we only allow the host to read a single dword, wait for 2 cycles, and poll the level signals again. Again this prevents the host from reading from an empty FIFO. This is just one example. Many other read rules could be designed.

4.2 Writing the Command and Source Fragments

An operation begins by the host writing a command message over the Sync bus to the Inbound FIFO. Let the command message be given by

128'h0002_0000_0000_0002_0000_0000_80CB_0000

Referring to the device specification, we see that, for this command, the number of source descriptors, NS, is 2, the overflow bit is zero, the Session Number is 2, the Command Parameter is 0, the Valid Bit is 1, the Function Code is 0, and the pass through value is 8'hCB. The following figure shows the Sync Bus signals during this command write.

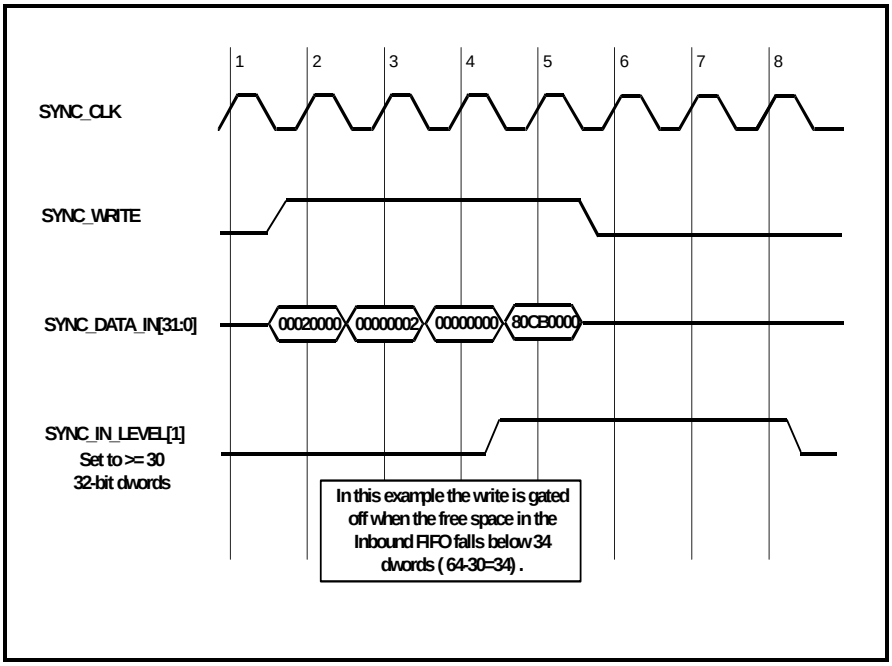

Figure 15. Host writes command.

The host then writes the first of the 2 source descriptors followed by its data, and then the second of the 2 source descriptor followed by its data. Let the first source descriptor be given by

64'h0000_0000_1000_0003

Referring to the device spec we see that, for this source descriptor, the byte alignment is 0, the source data endian is 32-bit big and the fragment size is 3 bytes. The following figure shows the Sync Bus signals during the write of the source descriptor and the 3 bytes of data given by

24'h1234_56

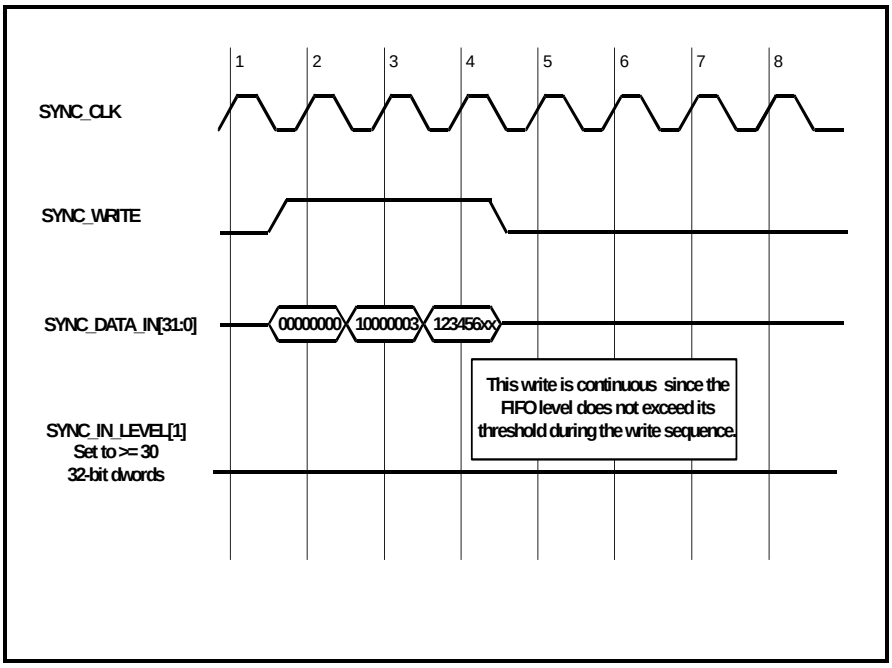

Figure 16. Host writes first source descriptor and 3 bytes of data.

Let the second source descriptor be given by

64'h0000_0003_3000_0005

Referring to the device spec we see that, for this source descriptor, the 32-bit byte alignment is 3, the source data endian is 32-bit little and the fragment size is 5 bytes. The following figure gives the timing diagram for the write of the second source descriptor and data given by

40'habcd_ef01_23

which is 40'h80F7_B3D5_C4 in 32-bit little endian format

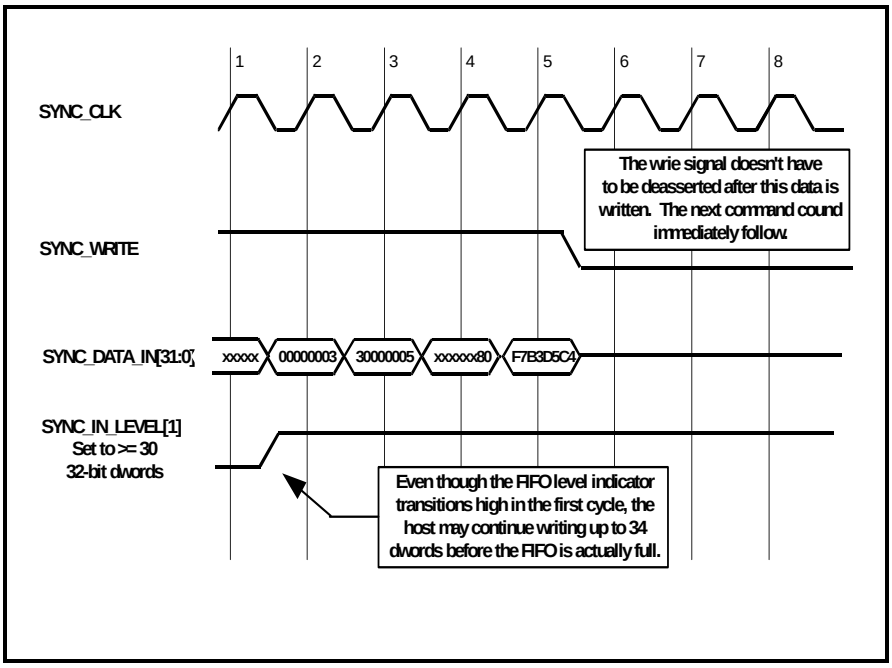

Figure 17. Host writes second source descriptor and 5 bytes of data.

4.3 Reading the Result Message and Destination Data

In parallel to writing, the host may read result and destination data from the Outbound FIFO as it becomes available.

The following figure gives the timing diagram for reading the result message:

128'h0000_0014_0000_0002_0000_0000_00CB_0000

Referring to the device spec this result message has Total Destination Count = 20, Session Number = 2, Result Parameter = 0, a pass through value of 8'hCB, A = 0, C = 0, Result Flags = 0 and Result Code = 0.

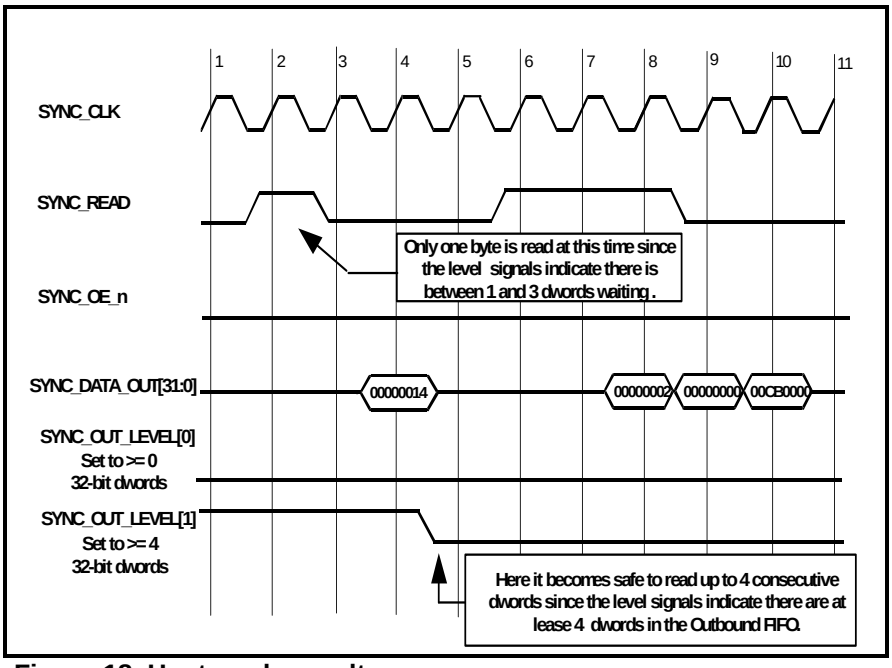

Figure 18. Host reads result message.

Figure 18 illustrates how the level signals are used to determine how many dwords are waiting to be read from the Outbound FIFO. In the first cycle the level signals indicate that there are between 1 and 3 dwords waiting to be read. To prevent a read underflow situation, the host reads a single dword and waits 2 cycles before polling the level signals again to determine if the FIFO is empty. Since the SYNC_OUT_LEVEL[1] signal transitions low for cycle 4, the 78xx must have dumped more dwords into the Outbound FIFO while the host was performing the single read. In this situation it safe for the host to start reading consecutive dwords since the FIFO level is greater than 3.

The result message is followed by destination data. The following figure illustrates the timing diagram for reading the data:

160'h0123_4567_89ab_cdef_0123_4567_89ab_cdef_0123_4567

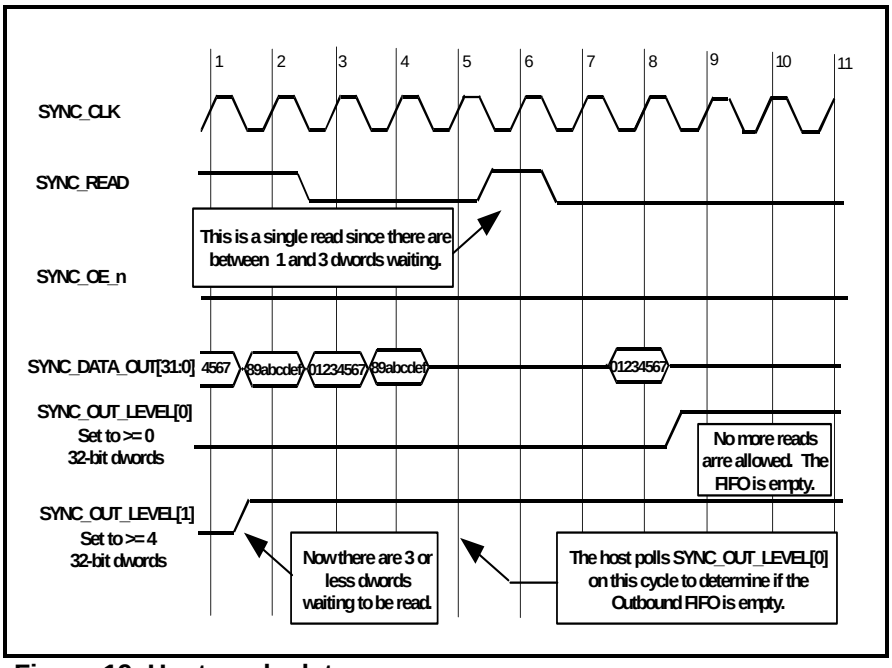

Figure 19. Host reads data.

Figure 19 illustrates how the level signals may be used to empty out the FIFO without reading while it is empty. The read process began prior to the first cycle and all of the result data is not shown in the figure. When SYNC_OUT_LEVEL[1] transitions high there are less than 4 dwords waiting to be read from the outbound FIFO. Since the level signals are updated one cycle after a change in the FIFOs, the host must deassert the read signal and begin reading only a dword at a time until the FIFO is empty or until there are 4 or more dwords in the FIFO. In this situation there was only one more dwords waiting to be read from the Outbound FIFO. Notice that after reading the single dword, the host must wait 2 cycles before polling the level signals to determine if there are any remaining dwords to be read.