SHAC

- SINGLE HOST ADAPTER CHIP -

ENGINEERING SPECIFICATION

-- VERSION 4.0 --

This document specifies all of the externalities of the SHAC chip. It is intended to serve as a guide to those who must interface drivers or devices which interact with the SHAC.

Part number DC-542

revision	date	description
1.0	16 Oct 86	Initial version.
1.1	3 Dec 86	Changes to reflect initial comments.
2.0	12 June 87	Change from UQ to CI Port.
3.0	10 August 88	First silicon updates.
4.0	6 September 89	Third silicon updates.

This document was prepared using VAX DOCUMENT, Version 1.0

Contents

Contents

Contents

FIGURES

TABLES

Preface

Intended Audience

This engineering specification is directed to all potential users of DSSI as well as those working on other embodiments of the same architectural definition. Comments and suggestions are solicited. Send them to:


```
    Michael Ben-Nun
    Mail Stop: ISV
Digital Technical Center
37 Pierre Koenig Street
   Jerusalem, ISRAEL

    JEREMY::MICHAEL

         tel:
 country code:     972
 area code:          2
 local number: 782 551
```


The group responsible for the preparation of this document comprises:

Michel Assayag
Michael Ben-Nun, Project Leader
Moshe De-Leon
James Feldman
Maurene Fritz
Eric Goldstein
Udi Kra
Pinchas Lozowick
Eyal Yatzkan
Eitan Zmora

SHAC is being designed at the Digital Technical Center in Jerusalem.

Changes Incorporated in This Version

All changes made since version 3.0 are marked with change bars.

Chapter	Important Changes
2	Updated document revisions.
3	Add VDD/VSS_CLEAN. Correct the error in pins 44,45
4	Host writes to PSR allowed as described
5	Minor changes and clarifications
6	Details on interlock operations and Burst limit
7	New behavior during the selection phase and modifications due to changes in the DSSI spec
A	Updated document revisions.
B	
C	
D	
E	New appendix.

Chapter 1

OVERVIEW

1.1 Introduction

SHAC (Single Host Adapter Chip) is a single-chip, VLSI version of an SCA port that uses a DSSI bus as the physical interconnect. Another SCA realization, CI, has defined a port-driver/port interface which has been used to connect VAXs in clusters. DSSI has adopted the same interface, so the same VMS port driver will be able to drive either a CI-port or SHAC. SHAC can be used to connect a host to any other device that can communicate through the CI-DSSI protocol. In particular, it provides a solution to:

1. The problem of interfacing a group of mass-storage device controllers (MSDCs) to a VAX.
2. The problem of interfacing several VAXs to a common group of MSDCs and, if higher level protocols support this option, to one another.

Where two or more VAXs connect to a group of MSDCs (or to one another) through DSSI, each has a SHAC (or another DSSI port) at the DSSI bus to serve as a port. Where a group of MSDCs connect to the DSSI bus, the controllers provide both the bus interface and the intelligent control required to respond to the CI commands received over the DSSI.

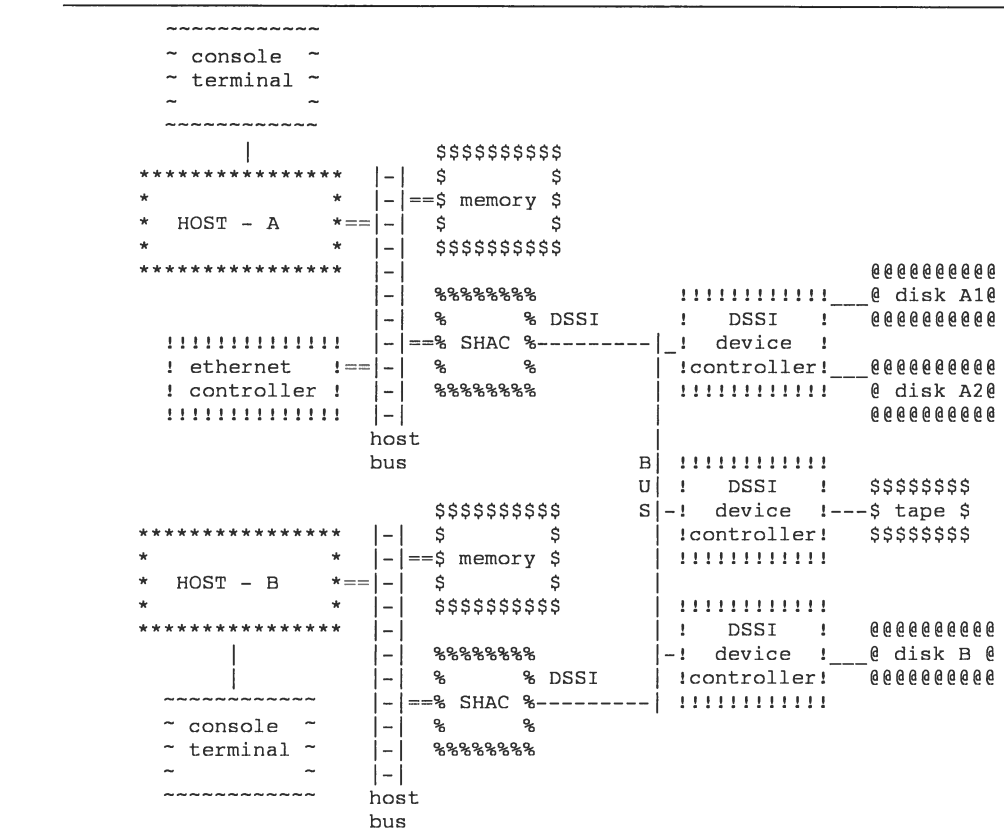
Figure 1–1 on the next page is the physical model of a system in which the SHAC will sit. Both the MSDCs and the several VAXs will communicate over a high-speed, 1-byte wide bus (DSSI) with a 4 to 5 MB/s burst transfer rate. The SHAC will handle the problem of providing effective, efficient and reliable interfacing between this DSSI bus and the host VAX, having direct host memory access (DMA) over the host's 32-bit wide, 16 MB/s bus. All communications between those connected to the DSSI will follow the CI protocol with the DSSI protocols providing handshaking in the transactions.

Structural parameters limit the number of possible combinations that can be realized with DSSI and SHAC.

1. A single DSSI bus has room for 8 nodes which may be partitioned among host adapters (e.g., SHACs) and MSDCs.
2. Since there must be a host, there can be up to 7 MSDCs on a single DSSI.

The SHAC provides a small amount of buffering on chip to improve bus utilization on both sides, but SHAC is designed to pass data through from one bus to the other as rapidly as the two busses permit. DMA services to and from the host reside in the SHAC, which responds to requests for transfers between the host and the remote nodes.

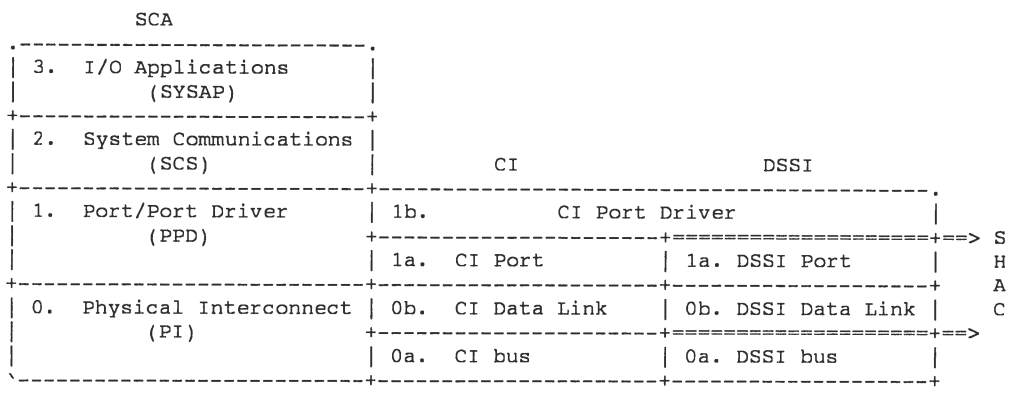
Figure 1–1: Sample Physical Layout



The SHAC is operated by an on-chip RISC processor that obtains its code and internal data from on-chip RAM and ROM. Part of the RAM may be loaded from host memory both during initialization and as circumstances require during normal run time. With this capability, it can read in new code and data from the host and thus adapt its behavior to new circumstances. This will permit inexpensive upgrades of SHACs after they are installed in the field. Furthermore, it will allow the SHAC to store infrequently accessed code in host memory, providing more capability than could be included in on-chip ROM.

The overall communication architecture under which SHAC works is Digital's SCA (Systems Communications Architecture). In this general architecture, four layers are defined, as shown in Figure 1–2. The architecture can be realized in a variety of ways. Two particularizations of the lowest two levels in the diagram are CI (*Computer Interconnect*) and DSSI (*Digital Small System Interconnect*). They share the same lowest host layer (CI Port driver) but have distinctly different physical interconnects. The layers between the Port Driver and the DSSI bus itself can be realized at both board and chip level and products at both levels are in design within Digital. The SHAC is a chip-level product which connects the host-bus to the DSSI bus, controlled by the host through a CI port driver and accepting and delivering CI-defined packets over the DSSI bus. Layers above the port driver are invisible to SHAC.

Figure 1-2: Relationship of DSSI to SCA and CI



The port driver maintains a set of 7 queues in its system space. Four of these contain commands for SHAC to execute. The priority of the command is determined by the queue it is on; order is determined by the position in the queue. Another queue contains all of the responses for the host (from SHAC or the remote nodes). Finally, there are two queues of "empty envelopes" for the host and the SHAC to use to fill with commands and responses and then to queue them on the other queues.

These "envelopes" are simply standard-sized "queuable" blocks of host memory. All commands and responses are copied into one of these standard-sized blocks. Included in the header on each block are a pair of queue pointers (for a doubly linked queue) and various standard identifiers which specify what is contained in the block and how much of the block represents the actual command or response. To be visible, a block must be on a queue, where pointers from other elements or the queue header show its presence. Once a block is removed from a queue, it is visible only to the entity which removed it.

SHAC's principal task is in accepting and delivering "mail" to other nodes. Externally (i.e., on DSSI) SHAC deals only in standard CI formats. Internally, SHAC deals with the envelopes just described and with blocks of data. Since DSSI deals with bytes and the host bus generally deals in longwords, SHAC must frequently do byte alignment tasks during transcription.

SHAC deals with the port driver in the virtual-address mode, unloading from the host the obligation to do virtual-to-physical address translation and to be aware of page crossings in virtually-contiguous blocks of information. SHAC supports full virtual address translation including the use of global I/O pages (to a depth of 1).

The rest of this overview chapter describes a typical set of steps that SHAC goes through in serving its role as the CI Port, with "mail" in both directions. Then it concludes with a brief description of what is in the other chapters.

1.2 CI-DSSI Overview

At start-up, the host provides the SHAC with a number of pointers to internal host structures, one of which, the *Port Queue Block* (PQB), contains pointers and data on all of the queues that the host maintains for CI. The SHAC uses this data to carry on its normal business in the following way.

If traffic is not coming in on the DSSI bus, SHAC goes to the highest command queue which has something enqueued. Choices are CMDQ0..CMDQ3, with 3 being most urgent. It dequeues an element from the queue and examines its header to see what it must do with the queue entry. It could be a command for the SHAC or an item to be delivered to one of the nodes on the DSSI. A command might be an order to deliver a block of data to a remote node. An item to be delivered would be either a datagram or a message.

A **datagram** is a "one-sided" communication—that is, one which will be sent without any assurance of either receipt or reply. An obvious application for such a communication is a request for the party at the other node to identify itself. If the host does not know if anything at all is out there, it must transmit its request without expectation. For this or any similar purpose, it employs a datagram. Datagrams are all of lengths guaranteed to fit in a datagram envelope.

A **message** is a "two-sided" communication used when a virtual circuit (an established formal relationship) between members of the bus exist. Once such a virtual circuit is established, the host(s) understand how to make requests of the other side. Such a request could be an order for a data transfer in either direction. The message itself (*move data*) is contained in a command (*deliver this message to ...*). Messages are all of lengths guaranteed to fit in a message envelope. Messages are always delivered sequentially to a given node—that is, in the order in which they were enqueued on a particular queue. While the SHAC supports retries if a message fails to get through, once the retry limit is reached without successful delivery, SHAC returns the command to the host, marking it as *undeliverable*, and then breaks the virtual circuit to that node.

A full transaction might go something like this:

1. The host queues a message for node 3 (say, a disk controller) to copy a block of 16 KB from host memory, starting at location X and to be stored in location Y on disk. The queues are doubly-linked, so at the top of every envelope there is a forward link FLINK and a backward link BLINK. Enqueuing involves putting link values into the new element's FLINK and BLINK and making the previous last-element's FLINK and the queue header's BLINK point to the new element.
2. When this message gets to the head of the queue, the SHAC dequeues it [1] , reads the header and finds that it should "dial up" node 3. To do this, SHAC goes through the DSSI protocols, contending for the DSSI bus and then, if successful in getting bus, specifying node 3 as the target. These steps are called *arbitration* and *selection*.
3. Node 3 responds by asking for the DSSI-command (*command-out phase*). In this phase, the SHAC tells node 3 how many bytes are coming and repeats the identification information to confirm a proper selection. Node 3 then tells the SHAC to switch to the *data-out phase*. SHAC sends a pair of CI header bytes to identify what type of message this is, and then proceeds to transmit the actual message read from the message block in host memory. The step-by-step details of the transfer are handled by hardware in the SHAC which permits simultaneous, buffered reading and writing on the two busses to which SHAC is

[1] Note that SHAC ends up holding the only pointer to the dequeued block of memory that constitutes the queue element. The port driver no longer "knows" where it is.

connected. Upon proper completion of the transmission, node 3 responds with a 1-byte acknowledgement of success (parity and check-sum proper and no other errors).

4. The SHAC is still holding the only pointer to the message block in host memory. It returns this to the host in one of two ways. If the host has requested a "return receipt", the SHAC puts the block on the Response Queue (RSPQ) to indicate proper delivery. This is where the port-driver software in the host will look for responses.

   Alternatively, the SHAC simply puts it back on the MFREEQ which holds the standard envelopes for messages. At this point the single message has been delivered and the message envelope is back in circulation.

5. After whatever delay node 3 needed to process the message, it contends for the bus and upon winning it, selects the SHAC as its target. It then sends a standard CI message as above telling SHAC to transmit the required data. In general, SHAC does not do this immediately, since it is obliged to handle traffic according to position in the queue and according to queue priority. Instead, it takes an empty envelope from MFREEQ, writes into it the message it is receiving and puts it on the proper CMDQ as specified in the message it just received.

6. When that message gets to the head of its queue, SHAC dequeues it once more, carries out its command (in transmissions of 4 KB whenever possible—a 4 KB transmission takes about 1 msec on the DSSI), possibly interleaving other transmissions of higher priority to this node or any priority to other nodes, until the last byte is sent. Once SHAC has completed this operation, it returns the message block to the MFREEQ.

7. Node 3 has put its data on the disk and must report to the host the successful completion of the transaction. Again it contends for the bus and upon winning specifies the SHAC as its target. Then it sends a message to the port-driver through SHAC confirming the successful transaction. SHAC dequeues another free envelope and writes this message into that block. Then it queues it on the host's RSPQ. Except for higher level responses in the host, that concludes a whole transaction.

The enqueue/dequeue operations represent a considerable fraction of the effort in delivering a message or datagram. To minimize this effort, SHAC caches a small number of the envelopes (that is, it keeps the pointers to the memory blocks) as they become free in its normal activity. It only fetches an envelope from the free queues when its own supply has disappeared, and it only returns them to the free queues when it has a full supply (4 of a type). By this and other attention to effort reduction and traffic conservation SHAC attempts to optimize its rate of doing useful work.

1.3 Document Scope

The remainder of this document concerns itself with the hardware, logical and software interfaces which support the sorts of activities described above. At the time of the writing of this document, some details are still in negotiation. These are marked TBD (to be decided) and will be specified in later versions of this specification.

The remaining chapters comprise:

- Chapter 2 provides a list of those documents from which SHAC has been specified and refined.
- Chapter 3 provides a description of the external connections to SHAC, including both electrical and physical data.

- Chapter 4 and Chapter 5 together specify all of the SHAC's host-addressable registers as well as the behavior of SHAC with respect to these registers. Details of initialization are included therein.
- Chapter 6 describes SHAC's use of and interactions with the host bus.
- Chapter 7 describes SHAC's use of and interactions with the DSSI bus.
- Appendix A provides an extensive list of related documents.
- Appendix B describes the "fair" aribitration scheme implemented in SHAC.
- Appendix C describes the SHAC shared host memory area.
- Appendix D describes the SHAC testing features and modes.
- Appendix E describes the errors which SHAC may report to the host.

Chapter 2

ORIGINS OF THE SHAC SPECIFICATION

This current SHAC specification is tightly coupled to the following documents:

- CVAX CPU chip engineering specification rev 4.0.
- CMCTL - CVAX memory controller engineering specification rev 1.1.
- VAX CI Port Architecture Rev 5.0 and subsequent ECO's Y87M11-2, Y87M11-1, Y87M08-1, and Y87M9-1.
- DEC STD 161-0, Rev A, Computer Interconnect Specification and subsequent ECO #2.
- DSSI, Digital's Small Storage Interconnect, An Addendum to DEC STD 161, Version X1.3.

The SHAC is specified and designed according to these documents. In order to prevent the overhead and conflicts between these documents and this specification, this specification does **not** repeat the content of these documents. A reader who needs details or better understanding should refer to these documents. Any discrepancy between these documents and the operation of the SHAC should be considered as a bug, unless otherwise explicitly stated.

This specification describes the implementation-specific details and the functions that are different in the SHAC from the definition in the above documents.

Chapter 1 is provided just for background information and does not replace either the DSSI or the other documents.

Note the revision numbers of the above documents. They are not necessarily the most recent ones.

Chapter 3

PINOUT

The chip is packaged in a 164-pin surface mount package. Signals that are active low are suffixed with "_L". Following are a pinout diagram of the chip, name, number and purpose of each pin and pinout signal descriptions. The sequence starts with the first CP_bus pin (#159) and continues clockwise. Pin location is subject to change.

Figure 3-1: SHAC Pinout

```
          1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
          4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6                   1 1 1 1 1 1 1 1 1 1 2 2
          5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
        .-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-.
        | V V Q Q Q Q Q V V Q Q Q V V D D D V V D D D V V D D V V D D D D D V V D D D D D V |
        | S D A A A A A S D A A A D S A A A D S A A A D S A A D S A A A A A D S A A A A A D |
        | S D D D D D D S D D D D D S L L L D S L L L D S L L D S L L L L L D S L L L L L D |
        | X X 8 9 1 1 1 X X 1 1 1     3 3 2 X X 2 2 2     2 2 X X 2 2 2 2 1 X X 1 1 1 1 1 X |
        | 9 9     0 1 2 9 9 3 4 5     1 0 9 1 1 8 7 6     5 4 1 1 3 2 1 0 9 1 1 8 7 6 5 4 1 |
        |                                                                                   |
140-144 + QAD<3-7>                                                                VSSX1     + 22
    139 + VDDX9                                                                   DAL<8-13> + 28-23
    138 + VSSX9                                                                   VDDX1     + 29
135-137 + QAD<0-2>                                                                VSSX1     + 30
131-134 + QINT<3-0>                                                               DAL<3-7>  + 35-31
    130 + VDDX9                                                                   VDDX1     + 36
    129 + VSSX9                                                                   VSSX1     + 37
    128 + QINT<4>                                                                 DAL<2>    + 38
    127 + QINT<5>                                                                 DAL<1>    + 39
    126 + QINT<6>                                                                 DAL<0>    + 40
    125 + QINT<7>                                                                 VSS       + 41
    124 + QINT<8>                                                                 RESET_L   + 42
    123 + VDDX9                                                                   CLKB      + 43
    122 + VSSX9                                                                   VDD       + 44
    121 + VDD                                                                     VSS       + 45
    120 + VSS                                                                     CLKA      + 46
    119 + QINT<9>                                                                 VSS       + 47
    118 + QHLD/SCAN_OUT_H                                                         VDD       + 48
    117 + QWR_L/TRISTATE_L                                                        CS/DP_L<0>+ 49
    116 + VSS                                                                     CS/DP_L<1>+ 50
    115 + VDD                                                                     VSSX2     + 51
    114 + QTM<2>                                                                  VDDX2     + 52
    113 + QTM<1>                                                                  CS/DP_L<2>+ 53
    112 + VSS_CLEAN                                                               CS/DP_L<3>+ 54
    111 + VDD_CLEAN                                                               DPE_L     + 55
    110 + SX                                                                      BM_L<0>   + 56
    109 + DB_RES                                                                  BM_L<1>   + 57
    108 + IO_L                                                                    VSSX3     + 58
    107 + VSSX8                                                                   VDDX3     + 59
    106 + REQ_L                                                                   BM_L<2>   + 60
    105 + CD_L                                                                    BM_L<3>   + 61
    104 + VSSX8                                                                   WR_L      + 62
        |                                                                                   |
        | S R V A B D V D V V D V D D V D D V D D V A D V V N I R D V I C V V D I E P C S S |
        | E S S C S B S B D S B S B B S B B S B B S S S S D S A D M S R C D S M A R W S E E |
        | L T S K Y P S 7 D S 6 S 5 4 S 3 2 S 1 0 S : : S D H K Y R S Q T D S G K R R : L L |
        | : : X : : : X :     :   : : X : : X : : X L L X X A E : : X : L     I E : F L 1 0 |
        | L L 8 L L L 7 L     L   L L 7 L L 7 L L 6     5 5 C 0 L L 4 L :     : I L L       |
        |                                                   : :         L     L :   :       |
        |                                                   L L                 L   L       |
        .-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-.
          1 1 1 1
          0 0 0 0 9 9 9 9 9 9 9 9 9 9 8 8 8 8 8 8 8 8 8 8 7 7 7 7 7 7 7 7 7 7 6 6 6 6 6 6 6
          3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3
```

Table 3–1: SHAC Pins

pin	in/out	no	purpose
		CP_Bus Interface Pins	
DAL<31:0>	IO	32	CP_bus data/address lines
RESET_L	I	1	RESET SHAC
CLKB,A	I	2	CPU clock input
CS/DP_L<0:3>	IO	4	Cycle status and data parity
DPE_L	IO	1	Data parity enable
BM_L<3:0>	O	4	Byte mask
WR_L	IO	1	Write
SEL0,1	I	2	Select address space
CS_L	I	1	Chip select
PWRFL_L	I	1	Power failure
ERR_L	I	1	Bus error
IAKEI_L	I	1	Interrupt acknowledge enable input
DMGI_L	I	1	DMA grant input
CCTL_L	O	1	Cache control (open drain)
IRQ_L	O	1	Interrupt request (open drain)
DMR_L	O	1	DMA request (open drain)
RDY_L	IO	1	Ready (open drain)
IAKEO_L	O	1	Interrupt acknowledge enable output
NSHAC_L	O	1	Not SHAC cycle
DS_L	IO	1	Data strobe
AS_L	IO	1	Address strobe
Total CP_bus Pins		**60**	
		DSSI Pins (Signals and Clocks)	
All outputs are open drain and active low			
DB_L<7:0>	IO	8	DSSI data bus lines
DBP_L	IO	1	Parity line
BSY_L	IO	1	Busy
ACK_L	IO	1	Acknowledge
RST_L	IO	1	Reset
SEL_L	IO	1	Select
CD_L	IO	1	Command (low) / data (high)
REQ_L	IO	1	Request

Table 3–1 (Cont.): SHAC Pins

pin	in/out	no	purpose
		DSSI Pins (Signals and Clocks)	
IO_L	IO	1	Input (low) / output (high)
DB_RES	O	1	DSSI bias resistor
SX	I	1	DSSI clock input
Total DSSI Pins		**18**	
		POWER Supply Pins	
V_{DD}	I	8	+5v power
V_{DD}X<1-3,5,9>	I	14	+5v buffer's power supply
V_{DD}_CLEAN	I	1	+5v Quiet power for DSSI pads
V_{SS}	I	10	Ground
V_{SS}X<1-9>	I	22	Buffer's ground
V_{SS}_CLEAN	I	1	Quiet ground for DSSI pads
Total Power Pins		**56**	
		Testing Pins	
These pins are for testing purposes only			
QTM<1:2>	I	2	QUIP test
QWR_L/TRISTATE_L	IO	1	QUIP write / Tristate SHAC pins
QHLD/SCAN_OUT_H	IO	1	QUIP hold/ Scan SHAC pins
QINT<9:0>	IO	10	QUIP interrupt lines
QAD<15:0>	IO	16	QUIP addresses and data
Total Testing Pins		**30**	

3.1 Pinout Signal Descriptions

The SHAC signals are divided into 4 main groups: the CP_bus signals, the DSSI bus signals, the power signals and the testing signals.

3.1.1 CP_Bus Signal Descriptions

The timing for the SHAC's CP_bus signals is derived from the CLKA and CLKB pins.

1. **DAL** - Data/Address Lines

   The Data/Address bus (DAL) is a 32-bit multiplexed bus used to transfer all data and address information between the SHAC, CPU and host memory. The strobe signals AS_L and DS_L determine whether the bus is transferring address or data information. Whoever is bus-master at a particular time controls the strobes and the address driver. Data may flow in either direction as specified by the bus-master on WR_L.

2. **RESET_L** - Reset

   Resets the SHAC to its initial state. Interrupts are disabled and all registers are reset to their default states. RESET_L should be asserted for at least six system clock cycles.

3. **CLKB,CLKA** - CVAX Clock Inputs

   These signals are used as clocks internally and are used as a reference for timing bus operations on the CP_bus. CLKA and CLKB should be connected to the CVAX Clock Chip pins to ensure that the timing of the bus signals when the SHAC is bus master matches the CVAX timing.

4. **CS/DP_L<0:3>** - Cycle Status and Data Parity

   a. The cycle status lines determine the type of cycle occurring on the CPU bus. When the SHAC is bus master it may assert Demand_D_stream_Read, Read_lock, Write_unlock or Write_No_Unlock on the CS lines. Current software always combines the lock/unlock pair as a single bus access. CS/DP_L<3> is used by the bus master to flag whether the current bus cycle is synchronous (low) or asynchronous (high). Following are the CS/DP_L<2:0> line values supported by the SHAC:

WR_L	CS/DP_L<2:0>	description
1	011	Interrupt acknowledge (slave only)
1	111	Demand_D_stream_read
0	111	Write_no_unlock
1	101	Read_lock (master only)
0	101	Write_unlock (master only)

   b. During a write cycle the SHAC drives the parity of the data on these lines. During read, if DPE_L is asserted, the SHAC will read the parity from these lines.

5. **DPE_L** - Data Parity Enable

   The SHAC asserts this pin on write to signify parity is generated. When this pin is asserted during read, the SHAC will check parity.

6. **BM_L<3:0>** - Byte Mask

   The byte mask signals specify which data bytes of the current transfer contain valid information during the data phase of the bus cycle. During a *write cycle* they indicate which of the data bytes in memory should and should not be altered, and for a *read cycle* they indicate which data bytes should be supplied by the external device (the SHAC always reads longwords). The BM signals are valid at the time that AS_L is asserted.

7. **WR_L** - Write

   The Write signal is used to specify the direction of the current bus transfer. If WR_L is asserted, the current bus master (either the SHAC or the CPU) will drive the DAL lines at data time. When WR_L is not asserted some external device is expected to supply data. WR_L can be used to control the direction of DAL transceivers. It is valid at the time AS_L is asserted.

8. **SEL0,1** - Select SHAC Address Space

   Using the SEL0,1 Pins, three pages can be assigned in the host address space in order to address up to three SHACs. The select lines assign a page to each of the SHACs. It is also possible to address any number of SHACs, this is done with SEL0,1 pulled up. In this case each SHAC can be addressed with the assertion of the appropriate CS_L pin.

9. **CS_L** - Chip Select

   When SEL0 and SEL1 are set the SHAC checks the CS_L line. When asserted, the SHAC ignores address lines <9:31> and the SHAC is selected.

10. **PWRFL_L** - Power Fail

    Power Fail input is monitored by the SHAC to recognize a power fail condition. Power Fail handling is described in Chapter 5.

11. **ERR_L** - Bus Error

    This signal is used by external logic to signal a hardware error condition on the CP_bus to the current bus master. Typically this is due to non-existent memory being accessed, but it may be asserted for other reasons. When the SHAC is the bus master it monitors the ERR_L line to see if the cycle it is performing is in error. Chapter 6 describes the SHAC response to an error.

12. **IAKEI_L** - Interrupt AcKnowledge Enable Input

    The CPU initiates an interrupt acknowledge cycle by asserting IAKEO_L, driving the IPL on DAL<6:2> with WR_L deasserted and writing 011 on the CS/DP_L<2:0> pins.

    The SHAC responds to an interrupt acknowledge cycle if the following conditions are met:

    - The SHAC has requested an interrupt (= IRQ_L asserted)

- IAKEI_L is asserted
- The IPL driven on DAL<6:2> matches the SHAC programmed IPL

The SHAC then drives DAL<15:0> with the appropriate interrupt vector, and asserts RDY_L to indicate to the CPU that a valid vector is present on the DAL bus.

The CPU reads the interrupt vector, and resumes the cycle as for a CPU read cycle. The SHAC deasserts IRQ_L, RDY_L and tri-states (releases) the DAL bus.

Where IAKEI_L is asserted with one or more of the other conditions not met, the SHAC asserts IAKEO_L to pass the interrupt acknowledge to the next device in the interrupt-acknowledge daisy chain.

13. **DMGI_L** - DMA Grant Input

    DMGI_L is the grant-in of the SHAC. When the SHAC is asserting DMR_L and DMGI_L is then asserted, the SHAC becomes the bus master. DMGI_L for each device comes from a central arbiter.

14. **CCTL_L** - Cache Control

    CCTL_L is asserted during SHAC DMA write cycles, to invalidate corresponding CVAX cache entries whenever a hit occurs. As one quadword is invalidated at a time by the CVAX, CCTL_L is asserted twice during a SHAC DMA octaword write access.

15. **IRQ_L** - Interrupt Request

    This line is used to signal interrupts from the SHAC to the CPU. The priority and vector for the requests is written into the SHAC by the host software. Since several peripheral chips may be connected to one CPU, the Interrupt Acknowledge (IACK) cycle must be decoded by each chip and used with a daisy-chain of grants to arbitrate among the peripherals. This process is described for the IAKEx signals.

16. **DMR_L** - DMA Request

    Used to request DMA cycles on the CPU bus. The SHAC waits for the DMGI_L to be de-asserted before it asserts the DMR_L. The SHAC owns the bus when DMGI_L is asserted. When the SHAC is done with the bus it releases the bus and then deasserts DMR_L.

17. **RDY_L** - Ready

    RDY_L is used to synchronize data transfers between CPU and memory or peripherals that operate at different speeds. During the data phase of the bus cycle the master must wait for RDY_L to be asserted by the external device (memory or IO device) before terminating the current cycle and latching (or removing) data from the bus.

    RDY is used by the SHAC in several situations:

    a. When the host accesses internal registers of the SHAC during a bus cycle, the SHAC performs its required actions and then asserts RDY_L to terminate the bus cycle.

b. When the SHAC is the master of the CP_bus, it must wait for an external device (usually memory) to assert RDY_L before terminating the transfer.

18. **IAKEO_L** - Interrupt AcKnowledge Enable Output

    IAKEO_L is used to daisy-chain interrupting devices. It is connected to IAKEI_L of the next device in the chain. If the SHAC is not selected to acknowledge the interrupt, then if IAKEI_L is asserted, SHAC asserts IAKEO_L. This will permit subsequent devices in the chain to acknowledge the interrupt.

19. **NSHAC_L** - Not SHAC

    NSHAC_L is asserted by the SHAC while it is not master, and AS_L assertion has latched into the SHAC an address which is not in its address space. NSHAC_L is deasserted by AS_L deassertion.

20. **DS_L** - Data Strobe

    DS_L provides timing information for the data transfer portion of the cycle. During a *read cycle* the asserting edge of DS_L indicates that the DAL lines are free to receive data, and the deasserting edge indicates that it has been latched by the SHAC and can be removed. In a *write cycle* the asserting edge indicates that data is present on the DAL lines, and the deasserting edge indicates that the data will be removed.

21. **AS_L** - Address Strobe

    The bus-master asserts AS_L when the DAL lines contain valid address information. The bus-slaves latch the DAL information on the asserting edge. The AS_L remains asserted until the end of the bus cycle, but the address is removed on or before DS_L assertion.

3.1.2 DSSI Signal Descriptions

The timing for the SHAC's DSSI signals is derived from the SX1 pin. All the DSSI-bus signals are open-drain and active-low.

1. **DB_L<7:0>** - Data Bus

   Eight data-bit signals. DB_L<7> is the most significant bit and has the highest priority during the Arbitration phase. Bit number, significance and priority decrease downward to DB_L<0>. A data bit is defined as one when the signal value is low and is defined as zero when the signal value is high.

2. **DBP_L** - DATA BUS PARITY

   Data parity DBP_L is odd (i.e. number of '0's (negative logic) in the data byte including DBP_L is odd). The use of parity is mandatory in DSSI. Parity is not valid during the Arbitration phase.

3. **BSY_L** - BUSY

   An "OR-tied" signal indicating the bus is being used.

4. **ACK_L** - ACKNOWLEDGE

   A signal driven by an initiator to indicate an acknowledgment for a REQ/ACK data transfer handshake.

5. **RST_L** - RESET

   An "OR-tied" signal that indicates the Reset condition.

6. **SEL_L** - SELECT

   A signal used by an initiator to select a target.

7. **C/D_L** - Command/Data

   A signal driven by a target that indicates whether Control or Data information is on the DATA BUS. Asserted (low) indicates Control.

8. **REQ_L** - REQUEST

   A signal driven by a target to indicate a request for a REQ/ACK data transfer handshake.

9. **IO_L** - Input/Output

   A signal driven by a target that controls the direction of data movement on the DATA BUS with respect to an initiator. Asserted (low) indicates input.

10. **DB_RES** - DSSI Bias Resistor

    A 5.62K$\Omega \pm 1\%$ resistor should be connected from this pin to ground. This Pin is used for the reference voltage for the DSSI input buffers. It is suggested that the ground of this resistor be connected as close as possible to the V_{SS}_CLEAN pin.

11. **SX** - DSSI clock input.

    This is a TTL level input. The frequency should be 4 times the required DSSI throughput; 16 Mhz for 4 Mbyte per second.

3.1.3 Power Signal Descriptions

1. **V_{DD} / $V_{DD}X$ / V_{DD}_CLEAN** - Power
   4.75V - 5.25V. These pins are used to distribute the transient current demands for the chip.

2. **V_{SS} / $V_{SS}X$/ V_{SS}_CLEAN** - Ground
   Ground.

3.1.4 Testing Signal Description

It is out of the scope of this document to describe the function of the testing pins. Output testing pins should not be connected, and input pins should be grounded (except for QWR_L/TRISTATE_L which should be pulled up). For information about the function of the testing pins refer to Appendix D.

1. **QTM<1:2>** - QUIP test
   Input signals for testing purposes only. Should be grounded in the normal use of the SHAC.

2. **QWR_L/TRISTATE_L** - QUIP write / Tristate SHAC Pins
   When this pin is asserted during normal use of the SHAC (QTM <1:2> grounded,) it tristates all SHAC pins. After using this pin the operation of the SHAC is **unpredictable**. A *reset* sequence is expected.

3. **QHLD/SCAN_OUT_H** - QUIP hold / Scan SHAC Pins
   When set, during normal use of the SHAC (QTM<1:2> grounded,) this pin enables pin continuity check. TRISTATE_L should be asserted for this test. Note that the following pins do not have continuity devices: **QTM<1:2>**, **QWR_L/TRISTATE_L**, **QHLD/SCAN_OUT_H** and **DB_RES**.

4. **QINT<9:0>; QAD<15:0>** - QUIP Interrupt Lines; QUIP Addresses and Data
   Input/Output signals for testing purposes only. May remain disconnected during normal use of the SHAC.

Chapter 4

HOST-ADDRESSABLE DEVICE REGISTERS

The host processor communicates directly with a SHAC through a set of device registers in the SHAC. These registers occupy a one-page (512-byte) region in the host I/O address-space, aligned on a page boundary. Each register has a fixed offset within the region; the base address of the region is as described in Chapter 6.

All of the registers are longword registers. They may be accessed only through longword operations.

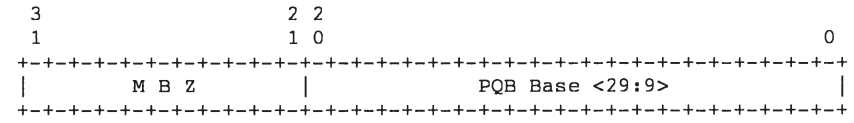
In addition to the access restrictions listed for specific registers, no register other than SHAC Software Chip Reset (SSWCR) may be read or written while certain chip initialization functions are being executed. The results of such an access during the 100 milliseconds following chip reset (power-up or a write to SSWCR), or during the 50 microseconds following a MIN-bit reset, are UNPREDICTABLE.

The registers can be divided into two categories:

- the CI Port registers defined in the CI Port Architecture specification;
- the SHAC specific registers.

4.1 CI Port Registers

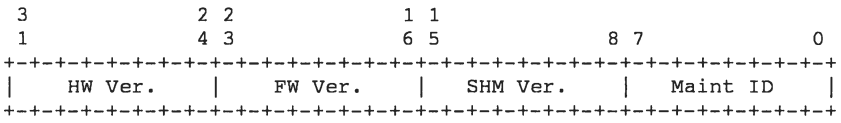
4.1.1 Port Queue Block Base Register (PQBBR)



This register contains the uppermost bits of the physical address of the base of the Port Queue Block. The PQB must be page-aligned, so the remaining bits of the address are assumed to be 0. PQBBR<31:21> MBZ.

PQBBR is read/write by the port driver and writable only when the port is in the *disabled* or *disabled/maintenance* state. (See Figure 5–1 for state diagram.)

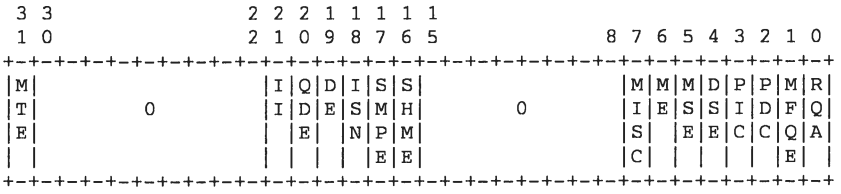
Following chip reset, PQBBR contains the following configuration information:



bits	name	description
31:24	HW Ver.	Hardware Version. Always greater than 0.
23:16	FW Ver.	Firmware Version. Always greater than 0.
15:8	SHM Ver.	Shared Host Memory Version. (0 until the Shared Host Memory Data Area has been read in.)
7:0	Maint ID	CI Port Maintenance ID. Always 22 (hex).

This information remains in PQBBR until the host overwrites it with the address of the Port Queue Block.

4.1.2 Port Status Register (PSR)



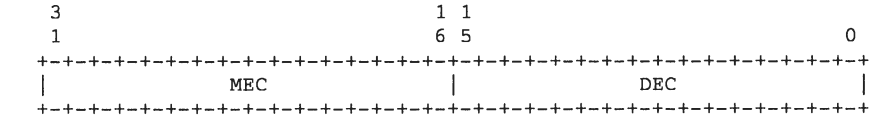
This register contains status information for the port driver. If interrupts are enabled (PMCSR<IE> is set), the port interrupts the host each time that it writes to this register. Once an interrupt is requested by the port, the value of PSR is fixed and is not changed until the port driver releases the register by writing the Port Status Release Control Register (PSRCR).

PSR may be read by the port driver at any time, except during chip initialization as noted above. Although CI Port does not permit it, the host may write to PSR after SHAC has loaded a non-zero value into it, but before writing to PSRCR. This makes it easier for the host to run the SHAC with interrupts disabled, as described in Section 5.1.17.

bit	name	description of status when the bit is set
31	MTE	Maintenance Error. The port has detected an implementation specific error (or hardware status condition). The source of the error may be more accurately determined from other bits in the upper half of this register and from the contents of other registers, as specified by Appendix E. The port is in the *uninitialized* state (port is non-functional). Maintenance Errors normally indicate a severe SHAC hardware or software failure. If the host believes that the SHAC is usable, despite this error, it can proceed with a MIN-Bit Reset.
21	II	Illegal Interrupt. This bit indicates a SHAC internal error, detected when the SHAC's internal microprocessor received an interrupt from an invalid source. This causes a Maintenance Error, so the MTE bit also will be set and the port about to enter the *uninitialized* state.
20	QDE	QUIP-Detected Error. This bit indicates a SHAC internal error, detected when the SHAC's internal microprocessor was given an invalid instruction. This causes a Maintenance Error, so the MTE bit also will be set and the port about to enter the *uninitialized* state.
19	DE	Diagnostic Error. An error was detected while running the SHAC's internal self-test. This causes a Maintenance Error, so the MTE bit also will be set and the port about to enter the *uninitialized* state. As described in Appendix C, diagnostics may be run when Shared Host Memory is loaded, optionally continuing if certain ones fail. When such a diagnostic fails, but the SHAC has been told to continue, DE will be set; however, MTE will be clear.
18	ISN	Illegal Segment Number. This bit indicates a SHAC internal error, in which it attempts to load a nonexistent External Segment from the SHAC Shared Host Memory. This causes a Maintenance Error, so the MTE bit also will be set and the port about to enter the *uninitialized* state.
17	SMPE	Slave Mode Parity Error. This bit is set by the occurrance of a parity error on a host access of a SHAC device register. This causes a Maintenance Error, so the MTE bit also will be set and the port about to enter the *uninitialized* state.
16	SHME	Shared Host Memory Error. This bit is set by the occurrance of an error involving the SHAC Shared Host Memory. This causes a Maintenance Error, so the MTE bit also will be set and the port about to enter the *uninitialized* state.

bit	name	description of status when the bit is set
7	MISC	Miscellaneous Error Detected. Indicates that the port microcode has detected one of the miscellaneous errors and the port is about to enter the *disabled/maintenance* state. The actual error code is stored in the Port Error Status Register.
6	ME	Maintenance Timer Expiration. The maintenance timer has expired. The port is in the *uninitialized/maintenance* state.
5	MSE	Memory System Error. Port has encountered an uncorrectable data or non-existent memory error in referencing memory. Port is in the *disabled* or *disabled/maintenance* state. See PFAR for further information.
4	DSE	Data Structure Error. The port has encountered an error in a port data structure (i.e., queue entry, PQB, BDT or page table). Port is in the *disabled* or *disabled/maintenance* state. See the Port Error Status Register (PESR) and the Port Failing Address Register (PFAR) for further information. Note that errors in queue structures leave the queues locked.
3	PIC	Port Initialization Complete. The port has completed internal initialization. The port is in the *disabled* or *disabled/maintenance* state.
2	PDC	Port Disable Complete. The port is in the *disabled* or *disabled/maintenance* state.
1	MFQE	Message Free Queue Empty. The port attempted to remove an entry from the MFREEQ and found it empty.
0	RQA	Response Queue Available. Indicates port has inserted an entry on an empty Response Queue.

4.1.3 Port Error Status Register (PESR)

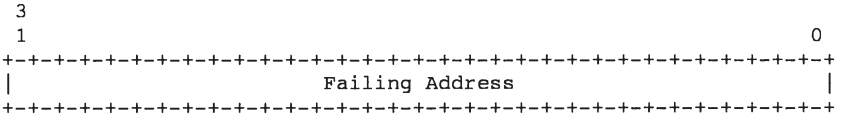


This register indicates the type of error which resulted in a DSE or an MISC (PSR bit) error.

PESR is read only by the port driver and valid only after either a DSE or MISC error, or after certain MTE and DE errors (as noted in Appendix E). Its value at any other time, or following a write to it, is UNPREDICTABLE.

bits	name	description
31:16	MEC	Miscellaneous Error Code. This code comprises two fields: bits <31:24> define the the module within the SHAC code where the error occurred, and bits <23:16> contain the specific error that occurred. These codes are implementation specific; those defined for SHAC are in Appendix E.
15:0	DEC	Data Structure Error Code. Appendix D of the CI Port Architecture specification describes the errors that are indicated by the contents of this field.

4.1.4 Port Failing Address Register (PFAR)

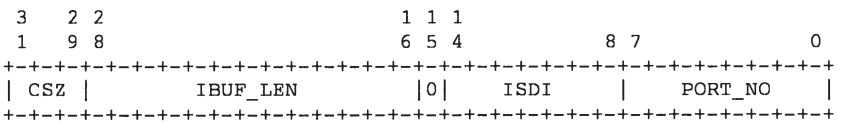


After an DSE, MSE, MTE, or DE error, or after a response with Buffer Memory System Error status, the PFAR contains the memory address at which the failure occurred. The address may be the exact failing address, an address in the same page as the exact failing address, or, in the case of DSE, an address in some part of the data structure. For DSE, PFAR contains a virtual address or offset, while for MSE and Buffer Memory System Errors, the PFAR contains a physical address. For MTE and DE, the interpretation of the address is error-dependent; see Appendix E for details.

Since the port continues command execution and packet processing after Buffer Memory System Errors, the PFAR is overwritten if subsequent errors occur. For DSE, MSE, and MTE errors the PFAR is effectively fixed since the port enters the *disabled*, *disabled/maintenance*, or *uninitialized* state.

PFAR is read only by the port driver and readable after a DSE, MSE, MTE, or DE error, or after a response with Buffer Memory System Error status. Its value at any other time, or following a write to it, is UNPREDICTABLE.

4.1.5 Port Parameter Register (PPR)

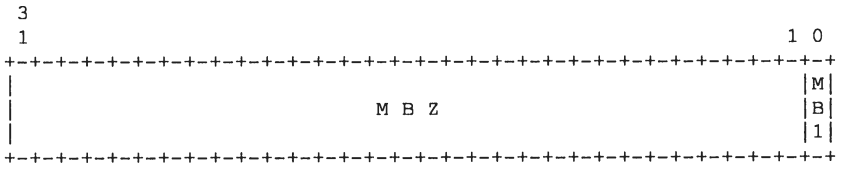


This register contains port implementation parameters and the port number. The value of the PPR is set by the port during initialization and is valid when PSR<PIC> is set.

PPR is read only by the port driver. Its value at any other time, or following a write to it, is UNPREDICTABLE.

bits	name	description
31:29	CSZ	Cluster Size. For SHAC, the value always is 0, indicating a maximum of 16 ports on the DSSI bus. (Note that the DSSI architecture only allows up to 8 ports on the bus, but 16 is the smallest size defined for the CSZ field.)
28:16	IBUF_LEN	Internal Buffer length. Indicates size of internal buffers available for message and data transfers. Maximum data packet =IBUF_LEN - 16 bytes. Maximum message or datagram length =IBUF_LEN. For SHAC, the value is 4112 (1010 hexadecimal).
14:8	ISDI	Implementation Specific Diagnostic Information. The bits in this field contain information about the local adapter's link layer configuration. For SHAC, the definitions of these bits are TBD.
7:0	PORT_NO	Port number. This is the same as the SHAC's DSSI ID.

4.1.6 Port Control Registers



The port control registers are 32-bit registers which are write-only by the port driver. To invoke the function provided by any of the control registers, the port driver writes a "1" to the register.

The result of writing any other value to any of these registers is UNPREDICTABLE. The value read from any of them is also UNPREDICTABLE.

4.1.6.1 Port Command Queue 0 Control Register (PCQ0CR)

When the port driver inserts an entry in an empty CMDQ0, the port driver writes PCQ0CR to initiate port execution of the Command Queue. PCQ0CR can be written only when the port is in the *enabled* or *enabled/maintenance* state. Writing to PCQ0CR when the port is in any other state has no effect.

4.1.6.2 Port Command Queue 1 Control Register (PCQ1CR)

Same as PCQ0CR except refers to CMDQ1.

4.1.6.3 Port Command Queue 2 Control Register (PCQ2CR)

Same as PCQ0CR except refers to CMDQ2.

4.1.6.4 Port Command Queue 3 Control Register (PCQ3CR)

Same as PCQ0CR except refers to CMDQ3.

4.1.6.5 Port Datagram Free Queue Control Register (PDFQCR)

When the port driver inserts an entry on the DFREEQ and the latter was previously empty, the port driver writes PDFQCR to indicate the availability of DFREEQ entries. PDFQCR can be written only if the port is in the *enabled* or *enabled/maintenance* State. Writing to PDFQCR when the port is in any other state has no effect.

4.1.6.6 Port Message Free Queue Control Register (PMFQCR)

Same as PDFQCR except refers to MFREEQ.

4.1.6.7 Port Status Release Control Register (PSRCR)

After the port driver has received an interrupt and read the PSR, it returns the PSR to the port by writing PSRCR.

4.1.6.8 Port Enable Control Register (PECR)

The port driver enables the port by writing PECR. PECR is ignored if the port is in the *uninitialized , uninitialized/maintenance , enabled ,* or *enabled/maintenance* state.

4.1.6.9 Port Disable Control Register (PDCR)

The port driver disables the port by writing PDCR. When the port is disabled, the port sets PSR<PDC> and (if interrupts are enabled) requests an interrupt. PDCR is ignored if the port is in the *uninitialized , uninitialized/maintenance , disabled,* or *disabled/maintenance* state.

4.1.6.10 Port Initialize Control Register (PICR)

The port driver initializes the port by writing PICR. When the initialization is complete, the port sets PSR<PIC> and (if interrupts are enabled) requests an interrupt. As part of the initialization, the maintenance timer is set to expire in 100 seconds.

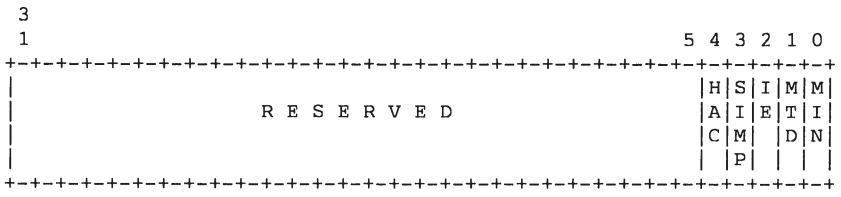
4.1.6.11 Port Maintenance Timer Control Register (PMTCR)

The port driver forces the maintenance timer to reset its expiration time by writing the PMTCR. If the PMTCR is not written again before the expiration time, the port will enter the *uninitialized/maintenance* state, setting PSR<ME> and (if interrupts are enabled) requesting an interrupt. PMTCR is ignored if the maintenance timer is not running.

4.1.6.12 Port Maintenance Timer Expiration Control Register (PMTECR)

The port driver forces a Maintenance-Timer-Expiration Interrupt by writing the PMTECR. This register may be written only when the port is in the *enabled , enabled/maintenance, disabled,* and *disabled/maintenance* states and only while the Maintenance Timer is not disabled.

4.1.7 Port Maintenance Control And Status Register (PMCSR)



This register is used for maintenance level control and status reporting. The CI Port specification defines all but the 2 least significant bits as implementation-specific. The bits can be divided into the following catagories:

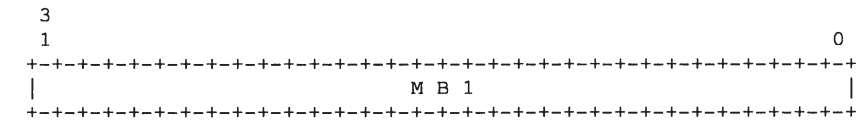
- Status bits are set by the port to report various conditions. They are cleared by maintenance initialization or clearing the condition in another register. SHAC's PMCSR does not include any status bits at this time.
- Function control bits are read/write by the port driver only, and read only by the port. They are cleared by chip reset. These bits are of two classes:
  1. Init: this type of bit invokes a function (e.g. initialization) by setting it. It always reads as zero, except while the function is active.
  2. Enable/disable: this type of bit causes an activity or state to exist while the bit is set. Clearing the bit stops the activity or changes the state. The bit always reads the most recently written value. The bit is never changed by the port.

bit	name	description
4	HAC	Host Access feature. MBZ, except for diagnostic purposes. Its use is described in Appendix D. This is an enable/disable class control bit.
3	SIMP	Simple SHAC mode. MBZ, except for diagnostic purposes. This is an enable/disable class control bit.
2	IE	Interrupt Enable. When set, interrupts from the port to the host are enabled. Power-up state is clear (interrupts disabled). This is an enable/disable class control bit.
1	MTD	Maintenance Timer Disable. Read/write by driver. If set, the maintenance timer is turned off. Timer is set to the initial value and suspended. If clear, timer functions normally. Power-up state is clear (timer enabled). This is an enable/disable class control bit.
0	MIN	Maintenance Init. Writing a "1" to this bit resets the port. Upon completion, the port is in the *uninitialized* state and MIN is clear. Writing a "0" to this bit has no effect. It always reads as zero, except while the reset function is active. Although Maintenance Init resets the port, it is not equivalent to a write to the SHAC Software Chip Reset register. See Chapter 5 for further information.

4.2 SHAC Specific Registers

These registers, which are not defined in the CI Port Architecture, are used for additional maintenance level control.

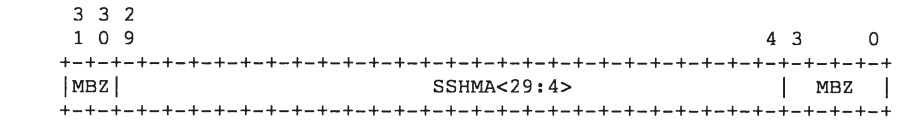
4.2.1 SHAC Software Chip Reset (SSWCR)



When the port driver writes FFFF FFFF (all bits set) to this register, a chip reset is performed. The result is equivalent to that of the reset that occurs following system power-up. On completion, all device registers are reset to their power-up state, and the port is in the *uninitialized* state.

SSWCR is write only by the port driver and may be written to at any time. Its value when read is UNPREDICTABLE. The results if other values are written to it are also UNPREDICTABLE.

4.2.2 SHAC Shared Host Memory Address (SSHMA)

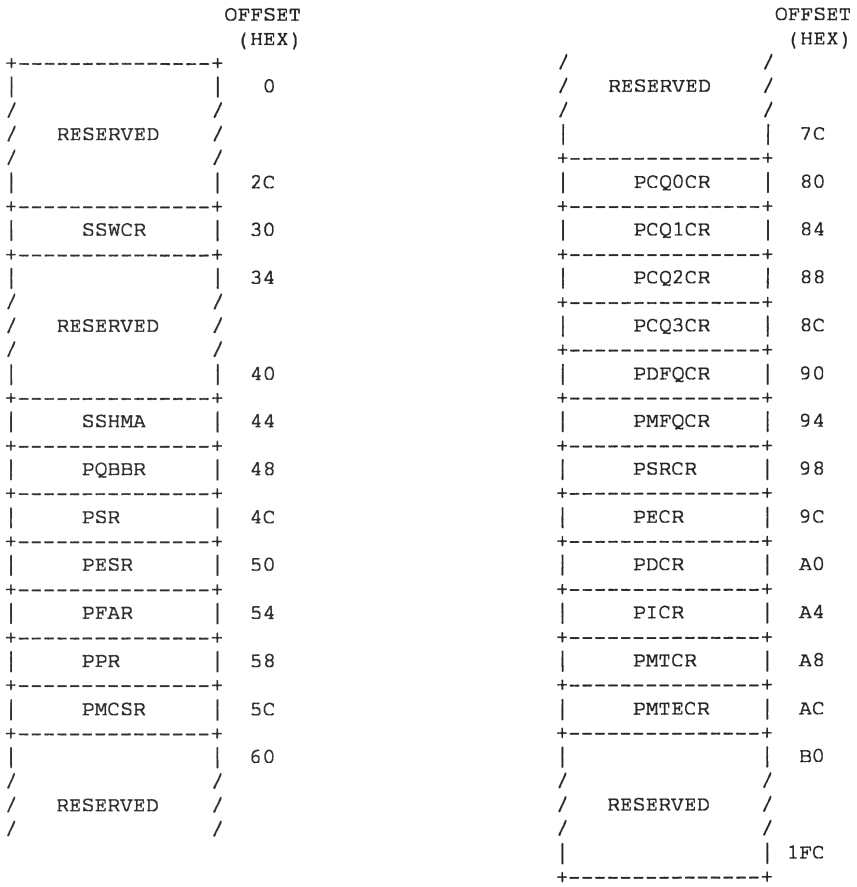


Following chip reset, the host writes into this register the physical address of the Shared Host Memory Header that is described in Chapter 5. This header must be octaword aligned and contiguous in physical memory.

SSHMA is read/write by the port driver, but may be written only when the port is in the *uninitialized* state. Writing when the port is in any other state can produce unpredictable results.

4.3 SHAC Device Register Map

This diagram shows the layout of the entire set of SHAC registers as they appear in the host bus address space. The offsets shown are from the base of the 512-byte SHAC register region. (Chapter 6 describes the region base addresses.)



Chapter 5

DETAILS OF HOST-SHAC COMMUNICATION

5.1 CI Port Functionality

This section describes various aspects of SHAC implementation of the CI Port Architecture.

5.1.1 Port States

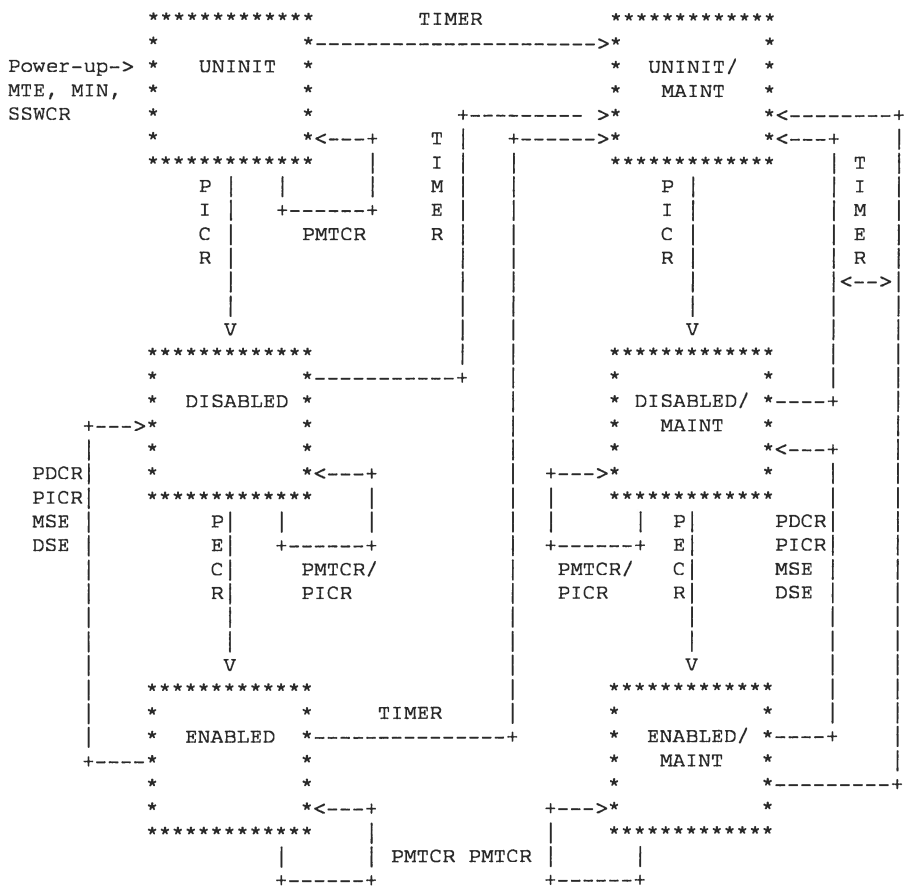
SHAC implements all 6 of the CI Port states, as shown in Figure 5–1. In addition to the state changes listed in the architecture, a port driver write to the SSWCR at any time causes a chip reset, following which the port is in the *uninitialized* state.

5.1.2 Chip Initialization

Following power-up, a port driver write to SSWCR, or a port driver write of 1 to PMCSR<MIN> (a MIN-bit reset), the SHAC and port driver perform certain initialization functions. This Chip Initialization is done in the *uninitialized* state, and concludes with CI Port initialization, which is triggered by a host write to PICR.

This section describes the sequence of Chip Initialization, which starts with either Chip Reset or MIN-Bit Reset.

Figure 5-1: SHAC Port State Diagram



5.1.2.1 Chip Reset

Chip Reset occurs upon power-up, and also when the host writes FFFF FFFF to SSWCR. The results of these 2 events are identical; in fact, the SHAC internal code has no way of distinguishing between them.

Following Chip Reset, the SHAC performs various self-test functions. These may take up to 100 milliseconds, during which time the host may not read or write any SHAC register, except SSWCR.

After waiting for at least 100 milliseconds following Chip Reset, the host can determine whether or not the SHAC passed its self-test by examining the Port Status Register. If it passed, PSR<MTE> will be clear, and the host can proceed as described in Section 5.1.2.3 below. If an error was detected, PSR<MTE> and PSR<DE> will be set, with further information supplied in other registers as specified by Appendix E.

Following the occurrance of a Diagnostic Error, the SHAC is in the *uninitialized* state. Such an error normally indicates a SHAC hardware failure. If the host believes that the SHAC is usable, despite this error, it can proceed with a MIN-Bit Reset.

5.1.2.2 MIN-Bit Reset

This is triggered by the host writing 1 to the MIN bit in PMCSR. The initialization that follows that may take up to 50 microseconds, during which time the host may not read or write any SHAC register, except SSWCR.

After waiting for at least 50 microseconds following MIN-Bit Reset, the host can proceed as described in Section 5.1.2.3 below.

5.1.2.3 Common Initialization

This section describes the initialization done following that for Chip Reset or MIN-Bit Reset. Throughout this initialization, the port is in the *uninitialized* state.

The port driver must write to the SSHMA register the physical address of the Shared Host Memory Header Area. Shared Host Memory is described in detail in Appendix C. A value of 3FFF FFFF may be written if there is no Shared Host Memory, as discussed there.

NOTE

Shared Host Memory must be present for full normal operation. If 3FFF FFFF is written to SSHMA, an attempt to read in an External Segment will cause SHAC to report an Illegal Segment Number Maintenance Error, as described in Appendix E. All of the following involve External Segments:

- generating an ID packet;
- responding to a host write to PDCR;
- processing a RDCNT command;
- processing a SETCKT command;
- processing any self-directed command.

Following the write to SSHMA, the port driver can request initialization of the port (as distinguished from Chip Initialization) by writing 1 to PICR. Port initialization, if successful, results in a change to the *disabled* state.

Following the writes to SSHMA and PICR, the port should poll PSR until the operations have completed or an error occurs. The maximum amount of time for this will be no more than 100 milliseconds; the actual time will be far less, however, unless the Shared Host Memory includes time-consuming External Code Segments that are executed at this time.

PSR should be polled until one of the following occurs:

1. If PSR<MTE> is set, a Maintenance Error has occurred. Further information about the error can be found as described in Chapter 4. This normally indicates a SHAC hardware or software failure. If the host believes that the SHAC is usable, despite this error, it can proceed with a MIN-Bit Reset.

2. If PSR<PIC> is set, the Shared Host Memory has been read and the port initialized. The port is now in the *disabled* state, and the port driver should proceed as follows.

At this point, the port driver may enable interrupts from the port to the host, by setting PMCSR<IE>. It then should check PSR, making sure that no bits other than PIC were set in it before the interrupts were enabled.

Since the port is in the *disabled* state, the port driver should now write PQB<29:9> to PQBBR, and then 1 to PECR to enable the port. Once this is done, normal operation begins.

5.1.3 Subnode and Full Sequence Number Support

These functions are quite independent of each other, but both are defined in ECO #2 to DEC Standard 161 and ECO Y87M9-1 to CI Port.

As permitted by the two ECO's, SHAC does not support subnode addressing, but does support Full (3-bit) Sequence Numbers.

5.1.3.1 SETCKT Command

Y87M9-1 specifies how a node supporting Full Sequence Numbers should implement Virtual Circuit Descriptors and the SETCKT command. It does not, however, specify how a port should respond if the host attempts, via a SETCKT command, to put an illegal value in the VCD.

For SHAC, an illegal value is one in which

- VCD<14:13> would be non-zero;
- VCD<11> would be non-zero; and/or
- VCD<FSN> would be zero.

SHAC responds to such a situation by returning an UNRCMD response.

5.1.4 Supported Opcodes

The following CI Port command and CI packet opcodes are supported:

- send DG/receive DG
- send MSG/receive MSG
- send SNTDAT/receive CNF
- receive SNTDAT/send CNF
- send DATREQ{0,1,2}/receive RETDAT
- receive DATREQ{0,1,2}/send RETDAT
- send IDREQ/receive ID
- receive IDREQ/send ID

- send RST
- send STRT
- send/receive LB
- INVTC
- SETCKT
- RDCNT

5.1.5 Unsupported Opcodes

CI Port ECO Y87M11-2 specifies that the following CI Port command and CI packet opcodes are optional; SHAC treats commands and packets with these opcodes as unrecognized.

- receive RST-STRT
- send SNTMDAT/receive MCNF
- receive SNTMDAT/send MCNF
- send MDATREQ/receive RETMDAT
- receive MDATREQ/send RETMDAT

CI Port ECO Y87M9-1 and DEC Standard 161 ECO #2 specify that the following CI Port command and CI packet opcodes are optional; SHAC treats commands and packets with these opcodes as unrecognized.

- send PSREQ/receive PSRET
- receive PSREQ/send PSRET

5.1.6 Maintenance/Sanity Timer

The Maintenance/Sanity Timer is implemented as specified in CI Port, with one modification. When the port enters the *uninitialized* state, the Timer is not started until after Shared Host Memory has been read (or the host has written 3FFF FFFF to SSHMA). Thus, a timer expiration and transition to the *uninitialized/maintenance* state cannot occur until after that.

5.1.7 ID Packet Configuration Information

The values for some of the fields of an ID packet are implementation-dependent; the following are placed in ID packets generated by SHAC:

name	bits	value (hex)	description
MAINT_ID	31	0	indicates single path interface
	30:0	22	SHAC's maintenance ID
CODE_REV	31:0		SHAC version information. This is the same as that displayed in PQBBR following chip reset, except that the SHM Version will never be 0 (since an ID packet cannot be generated unless Shared Host Memory is present).
PORT_FCN	31:0	FFFF 0D00	port functionality supported by SHAC
PORT_FCN_EXT (1st longword)	31:29	0	Cluster Size. This is the same as that displayed in PPR<31:29> following port initialization.
	28:16	1012	Maximum Body Length. This is 2 more than the IBUF_LEN displayed in PPR<28:16> following port initialization.
	13:11	0	CI configuration information not applicable to SHAC.
	7:0	0	Subnode Member which last reset the port; not applicable to SHAC.
PORT_FCN_EXT (2nd longword)	15	0	Subnode Addressing is not supported.
	14	1	Full Sequence Number is supported.
	13	0	Subnode Map is not valid.
	7:0	0	Number of Polling Group Members implemented on this node; not applicable to SHAC.

5.1.8 Data Packet Base Size

CI Port ECO Y87M11-1 specifies that support for the 576-byte Base Size is optional; SHAC does not support it. A command with P=1 in the flags byte will be treated as an Unrecognized Command; a packet with P=1 will be treated as Unrecognized Packet.

5.1.9 Path Select

SHAC supports only a single path, path 0. In processing commands, the Path Select field is ignored, as specified by section 6.1 of the CI Port Architecture. The portion of section 2.4 dealing with single path ports, which contradicts 6.1, is not followed.

5.1.10 Self-Directed Commands

SHAC fully supports *self-directed* commands. Such a command causes SHAC to transfer the command or data from the host memory source, first to SHAC internal RAM, and then to the host memory destination.

Because there is no hardware support for self-directed commands, they will generally take longer to execute than commands directed to a remote node. Processing of self-directed commands, as with non-self-directed ones, may be interrupted by reception of packets on the DSSI bus.

5.1.11 Loopback Commands

Neither the SHAC nor the DSSI hardware includes a specific loopback mechanism. Therefore, SHAC implements self-directed SNDLB commands in a way similar to that of other self-directed commands, transferring the loopback text from the host memory source to SHAC internal RAM, and from there to the host memory destination.

5.1.12 Sequentiality and Prioritization

SHAC follows the rules for sequentiality and prioritization specified by DEC Standard 161 and the CI Port architecture, including ECO Y87M11-1 to the latter.

5.1.13 Caching of Host Queue Entries

To minimize the time spent in queuing and dequeuing of host queue elements, SHAC will avail itself of the CI privilege of caching both empty and "in process" elements.

At any given moment, a SHAC may hold up to 31 queue entries that have been removed from host command and free queues, including up to 16 each of the datagram (unsequenced) and message (sequenced) types.

There may be up to 4 empty entries of each type, containing commands which have completed successfully, that could be placed on the DFREEQ or MFREEQ. SHAC pre-fetches free queue entries when it is otherwise idle, in addition to saving "used" entries.

5.1.14 Power Failure

SHAC detects assertion of the Power Fail line on the host bus. Its response depends on what state it is in at the time.

- If the port is in the *enabled* or *enabled/maintenance* state, SHAC writes the addresses of all internally held datagram and message queue entries to the corresponding logout areas of the Port Queue Block. It then enters the *uninitialized* state.
- If the port is in the *disabled*, *disabled/maintenance*, *uninitialized*, or *uninitialized/maintenance* state, it is not permitted to access host data structures. Therefore, it could not have been holding any such entries, so simply enters (or remains in) the *uninitialized* state.

5.1.15 Memory Management Mode

CI Port ECO Y87M08-1 defines different modes of host bus addressing, any or all of which may be supported by a given port. SHAC supports only the 30-bit VAX addressing mode.

When the SHAC reads the contents of the Port Queue Block, following a host write to the Port Enable Control Register, it checks the value of the FUNCTION_MASK field defined in Y87M08-1. If the MMM field is not zero, an Unsupported Memory Mapping Mode Data Structure Error is reported, and the port is disabled.

5.1.16 Event/Performance Counters

SHAC maintains all of the counters defined in section 4.9.1 of the DSSI specification. However, only those counters listed in the CI Port spec are returned in the CNTRD response to a RDCNT command. The remaining counters are accessible only via the Host Access Feature.

5.1.17 SHAC Operation with Interrupts Disabled

It is assumed that under normal cirumstances, interrupts from the SHAC will be enabled during initialization, as described in Section 5.1.2.3. However, it is possible to run SHAC with interrupts disabled, instead.

To do this, the host periodically polls PSR to see if any bits are set, instead of waiting for an interrupt. When it detects one or more PSR bits set, it then reads any other SHAC registers that may contain information regarding the event that caused the bit(s) to be set. Then, before writing 1 to PSRCR, the host writes 0 to PSR; this will allow it to detect the next time that SHAC sets a bit in that register.

5.1.18 Last Packet Implicit Buffer Invalidation

Section 3.3.1 of CI Port states that after processing a (sent or received) data packet with the Last Packet flag set, any cached address translations for that buffer are invalidated.

SHAC clears the translations only for the transaction associated with that particular data packet. If there are other transactions using that same buffer, address translations for them are not invalidated. This should not cause any problems.

5.2 DSSI Sequence Completion Status

In a CI Port response, the STATUS field sometimes includes the "result of last use of path", which may be ACK, NAK, NO_RSP, or ARB_TIMEOUT. In addition, both CI Port and DSSI specify counters for occurrances of ACK, NAK, and NO_RSP status. However, the DSSI spec does not define in detail the difference between NAK and NO_RSP.

The following pseudo-code describes how SHAC (as initiator) defines a sequence's completion status, based on the information reported by the SHAC hardware.

```
IF (the sequence does not reach Status In)
    THEN the status is NO_RSP
    ELSE
        IF (SHAC receives a Status In byte) AND (the bus is released) AND
                (there are no errors on the bus)
            THEN
                IF (the target received all Data Out bytes before going
                            to Status In)
                    THEN
                        IF the Status In byte is 61 hex
                            THEN the status is ACK
                            ELSE the status is NAK
                    ELSE the status is NAK
            ELSE
                IF (a parity error occurs) AND (the bus is released) AND
                        (there are no other errors on the bus)
                    THEN the status is NAK
                    ELSE the status is NO_RSP
```

5.3 DSSI Retries

When an error occurs in sending a packet on the DSSI bus, SHAC will retry the transmission. The retry algorithm conforms to the requirements agreed upon in the DSSI conference in June, 1989.

The algorithm includes a number of parameters, which are read in from Shared Host Memory at initialization time. (Appendix C includes a list of these and other parameters stored in Shared Host Memory.) This will allow system designers to tune SHAC's retry algorithm without modifying the SHAC itself.

While a packet is awaiting retry, no other packet will be transmitted to the same node. Additional commands to send packets to that node will not be processed during that time, regardless of their priority with respect to that of the packet awaiting retry.

While a packet is awaiting retry, packets will be transmitted to other nodes. Commands to send packets to those nodes will be processed during that time, regardless of their priority with respect to that of the packet awaiting retry.

5.3.1 Immediate Retries

The first "Number of Immediate DSSI Retries" times that the packet fails with a NAK status, it will be requeued for Immediate Retry. The term Immediate Retry means that the minimum delay between retries of a packet is on the order of 100 microseconds, though packets may be sent to other nodes between those retries. This minimum can be increased by means of the Idle Counter for Immediate Retries parameter in Shared Host Memory.

If a packet fails with a NO_RSP status during the first "Number of Immediate DSSI Retries" times, no further immediate retries are done. Instead, the packet is requeued for Delayed Retry.

The parameter "Idle Counter for Immediate Retries" may be set to a nonzero value, which will force a minimum delay between these retries. The precise correlation between this parameter and the delay time depends on the SHAC's internal code and the hardware environment around the chip.

5.3.2 Delayed Retries

If the final Immediate Retry fails, the packet will be requeued for Delayed Retry up to "Number of Delayed DSSI Retries" times. Each Delayed Retry will be done as follows:

1. The packet will be held for a period of up to 10 milliseconds. During this time, packets may be sent to other nodes.
2. A "coin-flip" decision be made, determining whether or not the packet should be retried. If the decision is "no", and the number of consecutive "no's" for this packet does not exceed the "Maximum Number of Coin-Flips" parameter, the packet is held for another 10 milliseconds.
3. If the decision is "yes", or if the number of consecutive "no's" for the packet does exceed the limit, the packet is retried.

Chapter 6

HOST BUS OPERATION

This chapter discusses the logical interactions between SHAC and the host bus.

6.1 Description of the Bus Cycles

Note that there is a differentiation between logical cycles and physical cycles. In this chapter logical cycles will always be prefixed with the cycle type (such as Bus Cycle, Read Cycle etc.). A physical cycle (which is nominally a 100nsec period which consists of four internal phases) will be reffered to simply as a cycle.

The SHAC is directly compatible with the CP_bus, and supports the following bus cycles:

- *longword* DMA read and write
- *octaword* DMA read and write
- *longword* DMA read_lock/write_unlock
- CPU read and write of SHAC registers
- interrupt acknowledge
- DMA grant

The SHAC will support a *burst mode* wherein the SHAC may transfer (*burst_size*) longwords before releasing the bus. The burst size may be written into the SHAC by the host during initialization of the SHAC.

6.1.1 Longword DMA Read Cycle

In a longword DMA read cycle, the SHAC reads one longword from host memory. A longword DMA read cycle requires a minimum of three cycles.

The steps in the longword DMA read-cycle are laid out in Table 6.1.

Table 6–1: Longword DMA Read Cycle

Cycle	Typical Phase	Changes on the bus
1	3	The SHAC drives the address onto DAL<29:02>. DAL<31:30> are driven[1] to 01 to indicate longword transfer. WR_L is deasserted.CS/DP_L<2:0> are driven to 111 (*demand D stream read*), CS/DP_L<3> is driven to one if synchronous mode and to zero otherwise.
	4	BM_L<3:0> are all asserted.
	1	The SHAC asserts AS_L, indicating that the address is valid.
	2	The SHAC deasserts DAL<31:0>.
2	3	The SHAC asserts DS_L, indicating that the DAL bus is free to receive incoming data.
3...	1	The SHAC then tests for *cycle complete* (RDY_L or ERR_L asserted) once every cycle. In a normal, error-free transfer, data is driven onto DAL<31:0>. RDY_L is asserted with ERR_L deasserted and SHAC reads the data from the DAL bus , data parity from the CS/DP_L and samples DPE_L. If DPE_L is asserted,the SHAC checks for a parity error. Should a bus error occur (e.g., time_out), external logic will respond by asserting ERR_L with RDY_L deasserted[2]. The SHAC shall ignore the data on DAL<31:0>.
	2	Without regard to how a longword DMA read cycle is terminated, the SHAC finishes the cycle by deasserting AS_L and DS_L.

[1]The terms driven, assertion and deassertion will be extensively used in this chapter. *Assertion* is used to indicate that a signal is TRUE, independent of its polarity. *Deassertion* is used to indicate that a signal is inactive. *Driven* is used to indicate that a particular set of bit values are put on the set of lines.

[2]This will be reported by SHAC as an error.

6.1.2 Longword DMA Write Cycle

In a longword DMA write cycle, the SHAC writes a single longword of information to host memory. A longword write cycle requires a minimum of three cycles.

The steps in the longword DMA write cycle are laid out in Table 6.2.

Table 6–2: Longword DMA Write Cycle

Cycle	Typical Phase	Changes on the bus
1	3	The SHAC drives the address onto DAL<29:02>. DAL<31:30> are driven to 01 to indicate longword transfer. CCTL_L and WR_L are asserted.CS/DP_L<2:0> are driven to 111 (*write no unlock*), CS/DP_L<3> is driven to one if synchronous mode and to zero otherwise.
	4	BM_L<3:0> are asserted as required.
	1	The SHAC asserts AS_L, indicating that the address is valid.
2	3	The SHAC drives DAL<31:0> with valid data, CS/DP_L<3:0> with valid parity and asserts DS_L and DPE_L.
3...	1	The SHAC then tests for *cycle complete* (RDY_L or ERR_L asserted) once every cycle. RDY_L is asserted with ERR_L deasserted. Should an error occur (e.g., time_out), external logic will respond by asserting ERR_L with RDY_L deasserted.
	2	Without regard to how a longword DMA write cycle is terminated, the SHAC finishes the cycle by deasserting AS_L and DS_L.

6.1.3 Octaword DMA Read Cycle

In an octaword DMA read cycle, the SHAC reads four consecutive longwords but specifies the address only for the first data transfer. This mode will be used by SHAC only if bursts of 4 longwords or more are allowed and if the *OE bit* in the HCR register is set.

An octaword DMA read cycle requires a minimum of nine cycles.

The steps in the octaword DMA read-cycle without error are laid out in Table 6.3. In case of an error response (ERR_L asserted with RDY_L deasserted) in either of the four longwords transfers, the SHAC will complete only the current transfer and release the bus.

Table 6-3: Octaword DMA Read Cycle

Cycle	Typical Phase	Changes on the bus
1	3	The SHAC drives the address of the first longword onto DAL<29:02>. This address will always be octaword aligned. DAL<31:30> are driven to 11 to indicate octaword transfer. WR_L is deasserted.CS/DP_L<2:0> are driven to 111 (*demand D stream read*), CS/DP_L<3> is driven to one if synchronous mode and to zero otherwise.
	4	BM_L<3:0> are all asserted.
	1	The SHAC asserts AS_L, indicating that the address is valid.
	2	The SHAC deasserts DAL<31:0>.
2	3	The SHAC asserts DS_L, indicating that the DAL bus is free to receive incoming data.
3...	1	The SHAC then tests for *transfer complete* (RDY_L or ERR_L asserted) once every cycle. In a normal, error-free transfer, data is driven onto DAL<31:0>. RDY_L is asserted with ERR_L deasserted and SHAC reads the data from the DAL bus, data parity from the CS/DP_L and samples DPE_L. If DPE_L is asserted,the SHAC checks for a parity error.
	2	The SHAC finishes this transfer by deasserting DS_L.
5,7,9		The SHAC tests again for next *transfer complete* (RDY_L or ERR_L asserted) once every cycle and reads the next three longwords from the DAL bus as it did for the first, finishing each transfer by deasserting DS_L.
9	2	The SHAC finishes the octaword transfer by deasserting AS_L.

6.1.4 Octaword DMA Write Cycle

In an octaword DMA write cycle, the SHAC writes four consecutive longwords but specifies the address only for the first data transfer. This mode will be used by SHAC only if bursts of 4 longwords or more are allowed and if the *OE bit* in the HCR register is set.

An octaword DMA write cycle requires a minimum of nine cycles.

The steps in the octaword DMA write cycle without error are laid out in Table 6.4. In case of an error response (ERR_L asserted with RDY_L deasserted) in either of the four longwords transfers, the SHAC will complete the whole octaword DMA write cycle, and only then release the bus.

Table 6–4: Octaword DMA Write Cycle

Cycle	Typical Phase	Changes on the bus
1	3	The SHAC drives the address of the first longword onto DAL<29:02>. This address will always be octaword aligned. DAL<31:30> are driven to 11 to indicate octaword transfer. CCTL_L and WR_L are asserted,CS/DP_L<2:0> are driven to 111 (*write no unlock*). CS/DP_L<3> is driven to one if synchronous mode and to zero otherwise.
	4	BM_L<3:0> are asserted as required.
	1	The SHAC asserts AS_L, indicating that the address is valid.
2	3	The SHAC drives DAL<31:0> with valid data, and asserts DS_L and DPE_L.
3...	1	The SHAC then tests for *transfer complete* (RDY_L or ERR_L asserted) once every cycle. In a normal, error-free transfer, data is driven onto DAL<31:0> and parity data is driven onto CS/DP_L<3:0>. RDY_L is asserted with ERR_L deasserted and SHAC reads the data from the DAL bus.
	2	The SHAC finishes this transfer by deasserting DS_L.
5,7,9		The SHAC tests again for next *transfer complete* (RDY_L or ERR_L asserted) once every cycle and writes the next three longwords from the DAL bus as it did for the first, finishing each transfer by deasserting DS_L. CCTL_L is asserted again when DS_L is asserted for the 3rd longword.
9	2	The SHAC finishes the octaword write cycle by deasserting AS_L.

6.1.5 Longword DMA read_lock/write_unlock Cycle

The longword DMA read_lock/write_unlock cycle, is used by the SHAC for semaphore operations. In the read_lock part of the cycle, the SHAC attempts to read one longword from host memory. The system will either accept the read_lock attempt by asserting ready, or refuse it, due to memory already locked, or for any other reason, by asserting RDY_L with ERR_L asserted (*retry response*),

In case of a successful read_lock, the SHAC will continue to assert DMR_L, thus holding the bus while it processes the read data. After a maximum of 20 cycles, the SHAC performs a write_unlock cycle and releases the bus.

In case of *retry response*, the SHAC finishes the read_lock cycle and releases the bus. After a maximum of 20 cycles, the SHAC will retry the read_lock/write_unlock cycle.

The steps in the longword DMA read_lock/write_unlock cycle without error are laid out in Table 6.5.

Table 6-5: Longword DMA Read_lock/Write_unlock Cycle

Cycle	Typical Phase	Changes on the bus
1	3	The SHAC drives the address onto DAL<29:02>. DAL<31:30> are driven to 01 to indicate longword transfer. WR_L is deasserted.CS/DP_L<2:0> are driven to 101 (*read_lock*). CS/DP_L<3> is driven to one if synchronous mode and to zero otherwise
	4	BM_L<3:0> are all asserted.
	1	The SHAC asserts AS_L, indicating that the address is valid.
	2	The SHAC deasserts DAL<31:0>.
2	3	The SHAC asserts DS_L, indicating that the DAL bus is free to receive incoming data.
3...	1	The SHAC then tests for *cycle complete* (RDY_L and/or ERR_L asserted) once every cycle. In a normal, error-free transfer, data is driven onto DAL<31:0>. RDY_L is asserted with ERR_L deasserted and SHAC reads the data from the DAL bus, data parity from the CS/DP_L and samples DPE_L. If DPE_L is asserted,the SHAC checks for parity error.
	2	The SHAC finishes the cycle by deasserting AS_L and DS_L.
		The SHAC takes at most 20 cycles (2.0 μsec typical) to process the input data and prepare it to return to the host memory. During that interval it holds the host bus. As soon as it finishes, it performs a write_unlock cycle.
1	3	The SHAC drives the address onto DAL<29:02>. DAL<31:30> are driven to 01 to indicate longword transfer. CCTL_L and WR_L are asserted. CS/DP_L<2:0> are driven to 101 (*write unlock*). CS/DP_L<3> is driven to one if synchronous mode and to zero otherwise.
	4	BM_L<3:0> are asserted as required.
	1	The SHAC asserts AS_L, indicating that the address is valid.
2	3	The SHAC drives DAL<31:0> with valid data, and asserts DS_L and DPE_L.

Table 6–5 (Cont.): Longword DMA Read_lock/Write_unlock Cycle

Cycle	Typical Phase	Changes on the bus
3...	1	The SHAC then tests for *cycle complete* (RDY_L or ERR_L asserted) once every cycle. RDY_L is asserted with ERR_L deasserted.
	2	The SHAC finishes the cycle by deasserting AS_L and DS_L.

6.2 CPU Cycles (SHAC slave)

The CPU may access any of the device registers described in Chapter 4,using either one of the three pre-allocated SHAC base addresses, or the Chip Select pin.

To define the slave addressing mode, the user should tie pins SEL0 and SEL1 as follows :

SEL1	SEL0	slave addressing mode/base address
0	0	2000 4000
0	1	2000 4200
1	0	2000 4400
1	1	Chip Select mode

Each SHAC is allocated one page in the host address space. The specific device register to be accessed is identified by the value of DAL<6:2> during the address phase, in both the SHAC address mode and the chip-select mode.

6.2.1 CPU Read Cycle

The CPU initiates a read cycle by asserting AS_L with WR_L deasserted, and driving the DAL bus with the address of the device register to be read. The SHAC latches the address, CS/DP_L<2:0> and WR_L with the asserting edge of AS_L . The SHAC keeps the NSHAC_L pin deasserted, and drives DAL<31:0> with the required data, CS/DP_L<3:0> with the data parity and asserts DPE_L to signify valid data parity. The SHAC then asserts RDY_L to inform the CPU that valid data is driven on the DAL bus. The CPU completes the cycle and deasserts the control lines. The SHAC deasserts RDY_L.

A CPU read cycle may last from five to six cycles.

6.2.2 CPU Write Cycle

The CPU initiates a write cycle by asserting AS_L, asserting WR_L, and driving the DAL bus with the address of the device register to be written. The SHAC latches the address, CS/DP_L<2:0> and WR_L with the asserting edge of AS_L . The SHAC keeps the NSHAC_L pin deasserted, and after DS_L has been asserted, latches the data on DAL<31:0>,data parity on CS/DP_L<3:0> and DPE_L and asserts RDY_L to inform the CPU that the data has been sampled. If DPE_L was asserted, the SHAC checks for parity error. The CPU completes the cycle and deasserts the control lines. The SHAC deasserts RDY_L.

A CPU write cycle may last from five to six clock cycles.

6.2.3 Interrupt Acknowledge Cycle

An interrupt acknowledge cycle has the same structure as a CPU Read cycle.

The CPU initiates an interrupt acknowledge cycle by asserting IAKEO_L, driving the IPL on DAL<6:2> with WR_L deasserted and writing 011 on the CS/DP_L<2:0> pins.

The SHAC responds to an interrupt acknowledge cycle if the following conditions are met.

- The SHAC has requested an interrupt (= IRQ_L asserted)
- IAKEI_L is asserted
- The IPL driven on DAL<6:2> matches the SHAC programmed IPL

The SHAC then drives DAL<15:0> with the appropriate interrupt vector, and asserts RDY_L to indicate to the CPU that a valid vector is present on DAL bus.

The CPU reads the interrupt vector, and resumes the cycle as for a CPU read cycle. The SHAC deasserts IRQ_L, RDY_L and tri-states (releases) the DAL bus.

Where IAKEI_L is asserted with one or more of the other conditions not met, the SHAC asserts IAKEO_L to pass the interrupt acknowledge to the next device in the interrupt-acknowledge daisy chain.

6.3 Bus Arbitration Cycle

After checking that DMGI_L is not asserted, the SHAC requests the host-bus mastership by asserting DMR_L. SHAC starts the first bus cycle two to three cycles after DMGI_L has been asserted. The SHAC releases the bus by deasserting DMR_L.

6.4 Bus Operating Modes

While master the SHAC can operate in either synchronous or asynchronous mode. Selection of the bus operating mode is made by setting the SYN bit in HCR register (see Chapter 4). While slave ,the SHAC always synchronizes input signals but responds synchronously with the clocks and thus can be accessed either synchronously or asynchronously.

6.4.1 Synchronous Mode

When the SHAC is master of the host bus in synchronous mode, CS/DP_L<3> is asserted during the first part of the bus cycle. The signals RDY_L, ERR_L, DMGI_L are not synchronized.

6.4.2 Asynchronous Mode

When the SHAC is master of the host bus in the asynchronous mode, CS/DP_L<3> is deasserted in the first part of the bus cycle. The signals RDY_L, ERR_L, DMGI_L are synchronized.

6.4.3 Burst Limit

The SHAC may be programmed during initialization to a burst limit. The burst limit is defined as the maximum number of longwords transfers the SHAC will perform within one bus arbitration. The burst limit must be a power of two, but less or equal to 64 longwords.

A special burst mode, is when burst limit is set to 1, and the octal access mode is enabled. In this mode, the SHAC will perform one bus access per arbitration (i.e. AS_L is asserted only once), but this access may be a single access (one longword) or an octaword access (four longwords).

Chapter 7

DSSI BUS OPERATION

7.1 Basic Operation

While eight devices can share a common DSSI bus, during information-transfer phases, the bus may be used only by a pair of them. When two DSSI devices communicate on the bus, one device acts as initiator and the other device acts as target. Devices can assume either role. Each device is assigned a unique address or ID on the bus. Each ID is represented by a single bit in an 8-bit byte. This byte is used both when an initiator contends for the bus and when it selects a target.

Certain functions are assigned to the initiator and others are assigned to the target. An initiator arbitrates for the bus and selects a particular target. Once selected the target requests the transfer of Command, Data or Status. In DSSI the target requests must be given in a specific order.

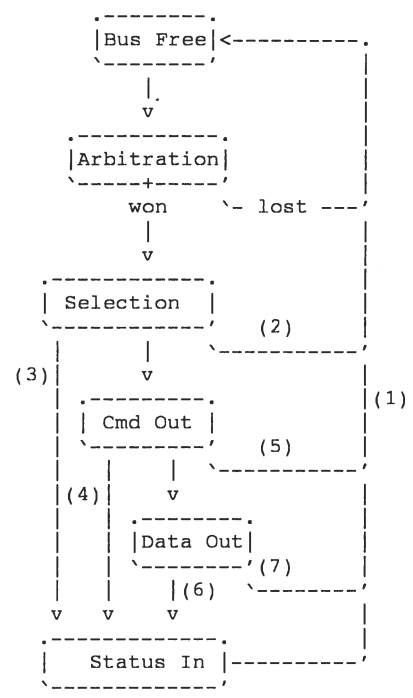
During the three data-transfer phases, information bytes (+parity) are sent on the 9 data lines DB_L<7:0,P> with handshaking. For Command-Out and Status-In, these transfers are asynchronous in the sense that each byte is surrounded by a conventional request(REQ)/acknowledge(ACK) handshake-pair. In the Data-Out phase, the transmission is termed *synchronous* in the sense that data is asserted in synchronism with the ACK signal (and held for a period determined only by the sender) rather than being held for periods determined by the handshakes. The target may send a number of REQs (up to the REQ/ACK offset) which are then followed by initiator data bytes and ACKs.

7.2 DSSI-Bus Phase Transitions

The DSSI goes through 5 phases for each transfer of information between any two nodes on the bus. The normal sequence of phases and exceptions are defined in Figure 7–1.

The sequence in Figure 7–1 always begins from the Bus-Free state. Several nodes may contend for the bus during the Arbitration phase, but only one (the node with the highest ID) will win the bus. The others must wait until the next Bus-Free phase to try again. In the normal, downward path in Figure 7–1, the node that wins the bus during Arbitration selects a target during the Selection phase, and then the pair exchanges three pieces of information during the next three phases. The target signals *Command-Out* and the initiator issues a command. The

Figure 7–1: DSSI Phase Sequence



command contains both identity information and the length of the coming data. The target then signals *Data-Out* and then data are transferred from the initiator to the target. Finally, the target asserts *Status_In* and returns the status (ACK or NAK—not to be confused with the ACK of REQ/ACK) to the initiator.

The **exception paths** in the DSSI transfer sequence are as follows:

1. A third party can always assert RST_L and cause a return to Bus-Free. The most likely reason for this to occur is that the third party has "timed out". Each party has a clock which it uses when it wants to use the bus and which it starts at the end of Bus-Free. If the bus is not released before that clock runs out, the third party will then attempt to reset the bus.
2. If a target fails to respond to selection (selection timeout) SHAC as initiator will do a smooth release of the bus without asserting RST_L (see Section 7.3.5). If the target responds with an illegal information phase (an information phase other than Command-Out or Status-In) the SHAC will respond by asserting RST_L on the bus, thereby forcing the bus to the Bus-Free phase.
3. Target does not have a buffer ready to receive the communication and returns a NAK.

4. The target issues a Status-In instead of Data-Out because it detects a command parity error or some other command validation error (length, operation code, illegal REQ/ACK offset, or source or destination port errors).

5. If a target fails to respond with the expected phase (a phase other than Data-Out or Status-In) or the Initiator-timeout occurs or there is a REQ/ACK offset error the SHAC as initiator shall respond by asserting RST_L on the bus, thus forcing the bus to the Bus-Free phase.

   SHAC as target will release the bus if a target-timeout occurs or there is a REQ/ACK offset error. The release of the bus puts the DSSI in the Bus-Free phase. The initiator shall stop its transmission and release the bus as well.

6. Target detects parity or checksum error. SHAC as target will wait until ACKs are received for all outstanding REQs and will switch phases to Status-In sending a NAK. This path is also the normal one, in which case the status returned is ACK.

7. SHAC as initiator will assert RST_L if an Initiator-timeout occured or the target responded with an unexpected phase (a phase other than Status-In) or there was a REQ/ACK offset error.

   SHAC as target will release the bus to enter the Bus-Free phase when target-timeout occurs or when there is a REQ/ACK offset error.

If the target enters the Bus-Free phase prematurely, SHAC will assume that the information frame was not delivered and will retransmit it.

The initiator may receive the Status-In byte with a parity error. The initiator treats this condition as if a NAK Status-In byte was received correctly. As a rule, a sender of information should treat all Status-In values other than an ACK status as a NAK status.

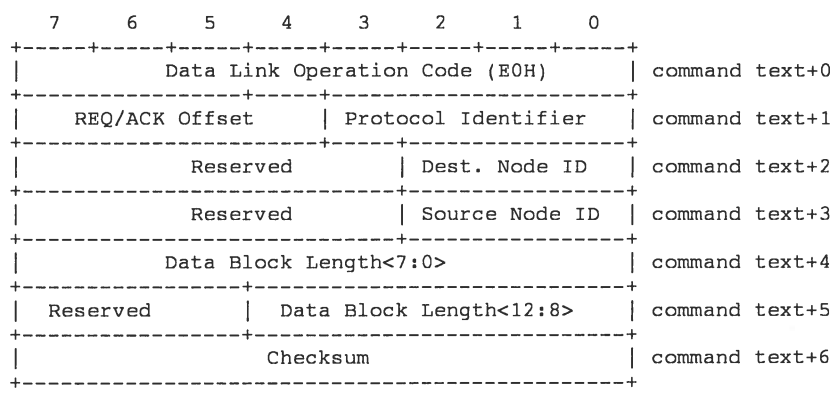
In borderline situations, SHAC behavior will be optimistic as target and pessimistic as initiator. An example of such a situation is when a reset occurs after the final release of ACK for the Status-In byte but before the bus is freed. In this case, SHAC as target will assume that the transfer was successful. SHAC as Initiator would assume that the transfer failed and will retransmit it.

The reasoning behind this behavior is as follows. There are delays on the DSSI bus which in such borderline cases may cause different devices to reach different conclusions as to what actually happened. It seems that the better choice is to have one too many packets (which should be discovered by the sequence number) than one packet missing (which will result in breaking a virtual circuit).

7.2.1 Formats of DSSI Data Exchanges

The information phases represent data exchanges that are set up and/or received by the intelligent controllers at the nodes. Each phase has its own rigid format for information exchange. These formats are presented below.

Figure 7–2: DSSI Command-Out Format



7.2.1.1 DSSI Command-Out Phase

The DSSI Command-Out phase utilizes the format illustrated in Figure 7–2.

The fields of this command are as follows:

- **Data Link Operation Code** is set to E0 (16).
- **Protocol Identifier** identifies a protocol layer, serviced by the Data Link, to receive the data block. A value of 0 specifies delivery to the CI port layer. Any other value should be treated as a reserved field which is not zero (see below).
- **REQ/ACK Offset** contains the maximum REQ/ACK offset to be used for this communication. This value is exactly the sender's maximum REQ/ACK offset. Proper values are 3, 7 or 15. The actual offset for the communication will be the smaller of the sender's offset and the target's offset. SHAC will always send 7 and use 7 or 3 as target. A number other than 3, 7 or 15 will be a validation error.
- **Destination Node ID** is the DSSI ID of the node receiving the information (i.e., the target). The destination port number serves as a double check of the DSSI selection sequence and must be checked at the target node as part of frame transmission command validation. The destination port ID is between 0..7.
- **Source Node ID** is the DSSI ID of the node transmitting the information (i.e., the initiator). The source port number serves as a double check of the DSSI selection sequence and must be checked at the target node as part of frame-transmission-command validation. The source port ID is between 0..7.
- **Data Block Length** is the number of bytes of data that will follow in the Data-Out phase excluding the checksum byte (the LS byte of the frame length word is delivered first). Frame length values less than 2, or more than 4114 are considered a length mismatch error.

- **Checksum** is the 8 bit XOR of the first six bytes of the command. This is generated by the initiator and checked by the target. This is simply the vertical parity.

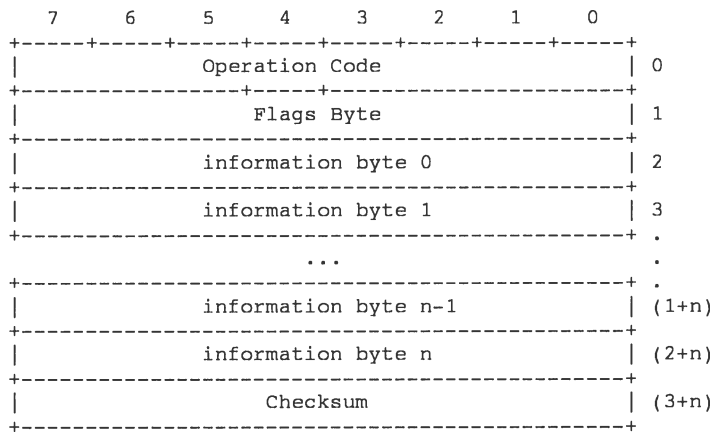
The treatment of Reserved fields is as follows. If the target detects a non-zero value in one of these fields, it shall ACK the packet if the packet is otherwise correct. However the target shall discard the packet, whether it was ACK'd or NAK'd. Note that the protocol identifier is treated as a reserved field.

Higher layers in the initiator or target are not notified when such a packet is discarded. A discarded packet is handled through provisions in higher layer protocols for detecting lost packets.

7.2.1.2 DSSI Data-Out Phase

The DSSI Data-Out phase utilizes the CI format illustrated in Figure 7–3.

Figure 7–3: DSSI Data-Out Format



The fields of the Data-Out phase are defined in the CI protocol. The invariant fields are:

- **Operation Code** defines the type of data that follows in the information bytes.
- **Flags Byte** is a qualifier on the operation code.
- **Checksum** is the XOR of the first n+2 bytes. This is simply the vertical parity. This byte is generated by the initiator and checked by the target.

7.2.1.3 DSSI Status-In Phase

The DSSI Status-In phase consists of one byte only which reports the success (ACK, 61 (16)) or failure (NAK, 1E (16)) of reception. The Status-In byte tells only if the data was received successfully or not and does not go into any more detail.

7.3 Detailed DSSI Bus Operation

7.3.1 DSSI Bus Signals

All the signals on the DSSI bus use negative logic. The following descriptions will assume negative logic. There are a total of 16 signals. 7 are used for control and 9 are used for data (including parity). See Chapter 3 for details.

7.3.2 DSSI Bus Phases

DSSI defines six bus phases:

- Bus-Free
- Arbitration
- Selection
- Command-Out
- Data-Out
- Status-In

The DSSI bus can never be in more than one phase at any given time. Unless otherwise noted in the following descriptions, signals that are not mentioned shall not be asserted.

The Command-Out, Data-Out, and Status-In phases are collectively termed the information transfer phases.

7.3.3 Bus-Free Phase

The Bus-Free phase is used to indicate that no device is actively using the bus and that it is available for subsequent users.

DSSI devices shall detect the Bus-Free phase after SEL_L and BSY_L have both been FALSE for at least 400 ns. All devices must be able to clear the bus within 400 ns after detecting a Bus-Free condition.

7.3.4 Arbitration Phase

The Arbitration phase allows all devices to contend for bus control. One device will gain control of the bus, becoming the initiator.

SHAC supports the fair arbitration scheme. In this scheme the procedure for a device to obtain control of the bus is a modified priority scheme. The basic concept is that DSSI nodes can be enabled or disabled. An enabled node may participate in Arbitration if it has a message to send, a disabled node may not. The normal sequence of events starts with all nodes enabled. Each node in turn wins arbitration, sends a message, and disables itself. Eventually all nodes are disabled, resulting in the DSSI bus becoming idle. All the nodes recognize that the bus is idle (BSY_L and SEL_L both false for a long enough time) and re-enable themselves (see Appendix B).

This modified priority scheme assures a fair distribution of bus privilege among the parties on the single DSSI bus.

7.3.5 Selection Phase

During the Selection phase the initiator will select a target for the purpose of sending it information.

The device that won arbitration (now the initiator) has both BSY_L and SEL_L asserted and has delayed at least 1200ns from the assertion of SEL_L to enable all other contenders to retire from the bus.

The initiator shall set DB_L<7:0> to a value which is the OR of its DSSI ID bit and the target's DSSI ID bit. Parity must be correct. The initiator shall then wait at least 90 ns and then release BSY_L. This starts the Selection phase. The initiator waits 400ns and then starts looking for the target's response. The target responds by asserting the BSY_L line.

SHAC implements the following selection timeout scheme. SHAC will wait a minimum of 20usec. If within this time the SHAC has received no response from the target, it shall release the data and parity lines. If within an additional 25usec, SHAC has still not received a response from the target, it will release SEL_L thus causing the bus to return to the Bus-Free state. Note that as long as SEL_L is asserted, if the target responds to selection, SHAC will consider the selection to have completed correctly.

Note that as initiator SHAC may wait much longer than the minimum 20usec (up to a few hundred usec) to check for a response from the target.

SHAC as target shall determine that it is selected when SEL_L and its DSSI ID bit are TRUE, and BSY_L is FALSE for at least 400 ns. SHAC will examine DB_L<7:0> both to determine the DSSI ID of the selecting initiator and to be sure that the selection is in order. If all is well, SHAC will respond to the selection within 1usec of the release of BSY_L by the initiator.

SHAC shall not respond to a selection if

- bad parity is detected, or
- more or less than two DSSI ID bits are on DB_L<7:0>.

No less than 90 ns after the initiator detects BSY_L is TRUE, it shall release SEL_L and may change DB_L<7:0>. This concludes the Selection phase. Control of the next phases transfers to the target.

7.3.6 Information Transfer Phases

The C/D_L and I/O_L signals are used to distinguish between the different information transfer phases. Once a target has been selected by an initiator it drives these two signals along with BSY_L and therefore controls all changes from one phase to another.

The target can cause the Bus-Free phase by releasing C/D_L, I/O_L and BSY_L. The initiator (or any other device on the bus) can cause the Bus-Free phase by asserting RST_L on the bus.

The information transfer phases use REQ/ACK handshakes to control the information transfer. Each REQ/ACK handshake allows the transfer of one byte of information.

During the information transfer phases, BSY_L shall remain TRUE and SEL_L shall remain FALSE. In addition the target shall maintain the control signals (C/D_L and I/O_L) valid for 400 ns before the assertion of REQ_L of the first handshake and they shall remain valid until the deassertion of ACK_L at the end of the last handshake.

The data transfer in Command-Out and Status-In is asynchronous. In Data-Out the data transfer is synchronous.

7.3.6.1 Asynchronous Information Transfer

If I/O_L is FALSE (transfer to the target), the target shall request information by asserting REQ_L. The initiator shall drive DB_L<7:0,P> to their desired values, delay at least 55 ns and then assert ACK_L. The initiator shall continue to drive DB_L<7:0,P> until REQ_L is FALSE.

When ACK_L becomes TRUE at the target, the target shall read DB_L<7:0,P>, then deassert REQ_L. When REQ_L becomes FALSE at the initiator, the initiator may change or release DB_L<7:0,P> and shall deassert ACK_L. The target may continue the transfer by once again asserting REQ_L, as described above.

If I/O_L is TRUE (transfer to the initiator), the target shall first drive DB_L<7:0,P> to their desired values, delay at least 55 ns then assert REQ_L. DB_L<7:0,P> shall remain valid until ACK_L is TRUE at the target.

The initiator shall read DB_L<7:0,P> after REQ_L is TRUE, then signal its acceptance of the data by asserting ACK_L. When ACK_L becomes TRUE at the target, the target may change or release DB_L<7:0,P> and shall deassert REQ_L. After REQ_L is FALSE, the initiator shall then deassert ACK_L. After ACK_L is FALSE, the target may release the bus (since this was the last step in Status-In).

7.3.6.2 Synchronous Data Transfer

Synchronous data transfers are used to send data to the target during the Data-Out phase.

The REQ/ACK offset specifies the maximum number of REQ_L pulses that can be sent by the target in advance of the number of ACK_L pulses received from the initiator.

If the number of REQ_L pulses exceeds the number of ACK_L pulses by the REQ/ACK offset, the target shall not assert REQ_L until the next ACK_L pulse is received. A requirement for successful completion of the Data_Out phase is that the number of ACK_L and REQ_L pulses be equal.

The target shall assert the REQ_L signal for a minimum of 90ns The initiator shall send one pulse on the ACK_L signal for each REQ_L pulse received. The initiator shall assert the ACK_L signal for a minimum of 90 ns.

During Data-Out the initiator shall transfer one byte for each REQ_L pulse received. After receiving a REQ_L pulse, the initiator shall first drive DB_L<7:0,P> to their desired values, delay at least 55 ns and then assert ACK_L.

The initiator shall hold DB_L<7:0,P> valid for at least 100 ns after the assertion of ACK_L. The initiator shall assert ACK_L for a minimum of 90 ns. The initiator may then deassert ACK_L and may change or release DB_L<7:0,P>.

The target shall read the value of DB_L<7:0,P> within 45ns of the transition of ACK_L to TRUE.

7.3.7 Command-Out Phase

The Command-Out phase is always the first information transfer phase in the information transfer sequence.

The target shall assert the C/D_L signal and deassert the I/O_L signal during the seven REQ/ACK handshakes of this phase.

7.3.8 Data-Out Phase

The Data-Out phase comes after the Command-Out phase. This phase is used to send data from the initiator to the target.

The target shall deassert the C/D_L and I/O_L signals during this phase.

7.3.9 Status-In Phase

The Status-In phase is the final phase of the information transfer sequence. In this phase the target sends status information about the previous phases to the initiator.

The target shall assert C/D_L and I/O_L during the REQ/ACK handshake of this phase.

7.3.10 Signal Restrictions Between Phases

When the DSSI bus is between two information transfer phases, the following restrictions shall apply to the bus signals. If these are violated, the results are UNPREDICTABLE (with the exception of the release of BSY_L, which will result in a switch to Bus-Free).

1. The BSY_L, SEL_L, REQ_L and ACK_L signals shall not change.
2. The C/D_L, I/O_L and DB_L<7:0,P> signals may change.

   When switching the direction of DB_L<7:0,P> from out to in (target to initiator), the target shall delay driving DB_L<7:0,P> by at least 800 ns after asserting the I/O_L signal. The initiator shall release DB_L<7:0,P> no later than 400 ns after the transition of the I/O_L signal to TRUE.
3. The RST_L signal may change as defined in Section 7.4.

7.4 Use of and Reaction to Reset

The Reset condition is used to immediately clear all DSSI devices from the bus. This condition shall take precedence over all other phases and conditions.

Any DSSI device may create the Reset condition by asserting RST_L for a minimum of 25 μsec. During the Reset condition, the state of all DSSI bus signals other than RST_L is not defined.

All DSSI devices shall release all DSSI bus signals (except RST_L) within 800 ns of the transition of RST_L to TRUE. The Bus-Free phase always follows the Reset condition.

SHAC will assert the DSSI bus RST_L signal only as initiator or when it is attempting to be initiator and when it detects one of the following conditions:

- SHAC as initiator detects an unexpected information phase on the bus.
- SHAC as initiator or potential initiator times out a target with the initiator timeout.
- SHAC as initiator detects a REQ/ACK offset error.

Note that the release of the bus by a target has the same end effect as a RST_L in that the bus returns to Bus-Free.

A reset on the bus detected by the SHAC does not cause the SHAC any loss of context.

7.5 DSSI Timeouts

Three timeouts are used to insure that the bus does not lock up, one as initiator, one as target, and one during selection when attempting to select a nonexistant device.

As inititator SHAC will reset it's initiator timeout clock whenever it detects a Reset or Bus-Free. When the bus leaves the Bus-Free state the timer will be started. If 2800 μsec have elapsed and a Bus-Free has not been once again detected, SHAC may decide to reset the bus.

SHAC always monitors the bus for initiator timeouts but it will only actually reset the bus if it is actively attempting to become initiator or if it is presently initiator.

SHAC as target will start it's target timeout clock whenever it is selected. If SHAC does not detect a Bus-Free within 2400 μsec from when it is selected it will release the bus thereby entering the Bus-Free phase.

Note that the Bus-Free phases will occur at intervals less than or equal to an initiator-timeout.

When SHAC starts selecting a target it starts timing the selection. The SHAC will wait a minimum of 20usec. If the target has not responded by this time the SHAC will begin its smooth release of the DSSI bus (see Section 7.3.5).

Appendix A

Related Documents

The SHAC specification is related to the following documents:

The documents listed below originate with the Jerusalem Technical Center. Copies may be obtained through Michael Ben-Nun. See the cover sheet for communication details.

- SHAC chip timing specification.
- SHAC design methodology.
- SHAC design specification.
- SGEC chip engineering specification.
- QUIP (QUickly Integrated Processor) engineering specification.

The documents listed below are maintained by other groups throughout DEC. Consult the appropriate group for copies.

- CVAX CPU chip engineering specification rev 4.0.
- CVAX clock chip engineering specification.
- CMCTL - CVAX memory controller engineering specification rev 1.1.
- VAX CI Port Architecture Rev 5.0 and subsequent ECO's Y87M11-2, Y87M11-1, Y87M08-1, and Y87M9-1
- DEC STD 161-0, Rev A, Computer Interconnect Specification and subsequent ECO #2.
- DSSI, Digital's Small Storage Interconnect, An Addendum to DEC STD 161, Version X1.3.

Appendix B

DSSI FAIR-ARBITRATION SCHEME

B.1 Basic Operation

This appendix describes the fair arbitration scheme which is implemented in SHAC.

The scheme assumes that each DSSI node will disable itself upon completion of a turn on the bus and reenable itself only after the DSSI-bus remains free for a *DSSI Idle Delay*. An enabled node may participate in arbitration; a disabled node may not. The normal sequence of events starts with all nodes enabled. Each contending node wins arbitration in its turn, uses the bus, and then disables itself. Eventually all contending nodes are disabled, resulting in the DSSI bus becoming idle. All the nodes recognize that the bus is idle (BSY_L and SEL_L both false for a long enough time) and re-enable themselves.

The following timing parameters are to be used:

- **DSSI Arbitration Skew Delay [600ns].** The maximum allowable skew for a DSSI node to assert BSY_L and its DSSI ID during arbitration.
- **DSSI Idle Delay [2200ns].** The minimum time a DSSI node must wait before concluding that no enabled nodes are arbitrating for the bus.

All timing measurements shall be calculated from the signal conditions existing at each DSSI node's own DSSI connection (that is, the DSSI bus side of drivers and receivers).

SHAC implements an internal flag indicating whether it is enabled or disabled. This arbitration algorithm specification is divided into three sections:

1. Behavior of an enabled SHAC. When SHAC is enabled and wishes to send a message it will participate in arbitration. Upon winning arbitration SHAC will disable itself and attempt to send the message.
2. Behavior of a disabled SHAC. When SHAC is disabled it does not participate in arbitration but it monitors the DSSI bus and enables itself upon detecting that the bus has become idle.
3. SHAC manipulation of it's enabled/disabled flag.

B.2 Behavior of an Enabled SHAC

When SHAC is enabled and it has a message to send it shall arbitrate for the DSSI bus as follows:

1. Wait for the Bus-Free phase to occur. The Bus-Free phase is detected whenever both BSY_L and SEL_L are simultaneously and continuously false for a minimum of 400ns.
2. Wait an additional minimum of 800ns after detection of the Bus-Free phase (i.e. after BSY_L and SEL_L are both false for 400ns) before driving any signal.
3. Following the 800ns delay in step 2, the SHAC shall assert both BSY_L and its own DSSI ID within a DSSI *arbitration-skew* delay [600ns]. SHAC will not arbitrate (i.e. assert BSY_L and its DSSI ID) if more than 1400ns have passed since the Bus-Free phase was last observed.
4. After waiting at least an arbitration delay [2200ns] (measured from its assertion of BSY_L) SHAC shall examine DB_L<7:0>. If a higher priority DSSI ID bit is asserted on DB_L<7:0>. (DB_L<7> is the highest), then SHAC has lost the arbitration and will release its signals and return to step 1. If no higher priority DSSI ID bit is asserted then SHAC concludes that it has won the arbitration and asserts SEL_L. Any other DSSI node that is participating in the Arbitration phase has lost the arbitration and must release BSY_L and its DSSI ID bit within 800ns after the assertion of SEL_L.
5. SHAC will wait 1200ns after asserting SEL_L before changing any signals.
6. SHAC will disable itself prior to releasing both BSY_L and SEL_L regardless of the outcome of the subsequent Selection phase or message transmission attempt.

B.3 Behavior of a Disabled SHAC

- When SHAC is disabled it shall monitor the DSSI bus for an idle condition regardless of whether or not it has a message to send.
- A DSSI bus idle condition occurs whenever BSY_L and SEL_L are simultaneously and continuously false for a minimum of a DSSI Idle Delay [2200ns]. SHAC will enable itself upon detecting a DSSI bus idle condition.
- SHAC shall detect a DSSI bus idle condition within a DSSI Arbitration Skew Delay [600ns] of its occurence.
- If SHAC begins to monitor the bus and finds it in the Bus-Free state it will consider this time as though BSY_L and SEL_L were just released and continue the timing as defined above (i.e. it shall detect a DSSI bus idle condition no sooner than a DSSI IDle Delay [2200ns] and no later than a DSSI Idle Delay [2200ns] plus a DSSI Arbitration Skew Delay [600ns] after starting to monitor the bus).
- If SHAC has a message to send, once it has detected the idle condition it will continue with step 2 from Section B.2.

B.4 SHAC Manipulation of its Enabled/Disabled Flag

This section states the rules the SHAC uses in manipulating the enabled/disabled flag, repeating the rules stated in the previous two sections. The rules are:

1. When SHAC is newly initialized or has otherwise lost track of the DSSI bus context it shall initially be disabled.
2. A disabled SHAC will only become enabled after detecting the idle condition.
3. SHAC will disable itself after winning arbitration, prior to releasing BSY_L or SEL_L.

Appendix C

SHAC Shared Host Memory

C.1 Overview

Some of SHAC's read-only data and code are stored in a dedicated block of host memory. Because SHAC treats it as read-only, multiple SHACs on the host bus can share portions of the same area; hence the name.

SHAC shared host memory contains four areas:

- a header area, containing pointers to the other areas
- a parameter area
- a patch area used to load code patches into the SHAC as part of initialization
- an "external code segments" area containing code loaded into the SHAC when it is needed.

The SSHMA host-addressable register, when set by the host, points to the start of the header area. Each pointer in the Shared Host Memory data structures is a signed byte offset from the start of its area. (This allows Shared Host Memory to be inserted at different places in VAX address space, either in ROM or RAM.) Each area need not be physically contiguous with the other areas.

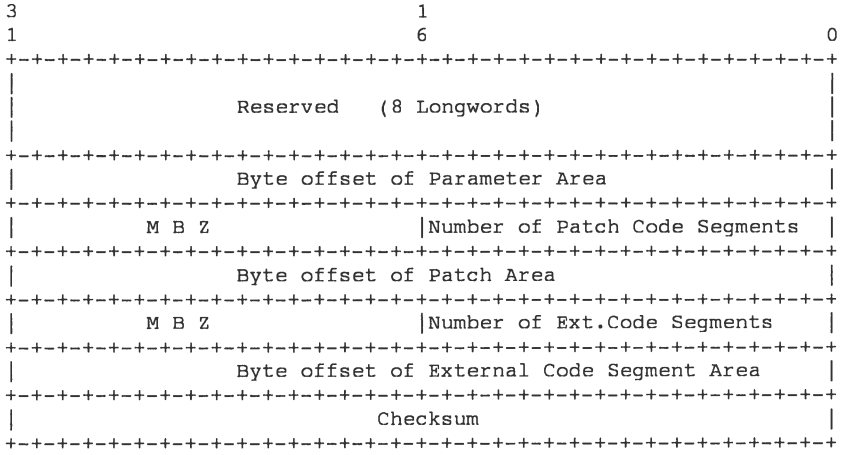
Each of the areas contains one or more "table" sections. The patch area and the External Code Segment area also contain code segments. Each area, table section, and code segment is octaword aligned.

Some portions of Shared Host Memory have checksums. Each checksum is the 32-bit sum of all 16-bit words in the section of memory being checked.

The total size of Shared Host Memory depends on the sizes of the Patch and External Code Areas. These, in turn, depend largely on the requirement of the system module containing the SHAC. The number of pages required for all of Shared Host Memory could be as small as 1; unless the system requirements are highly unusual, it would not be greater than 8.

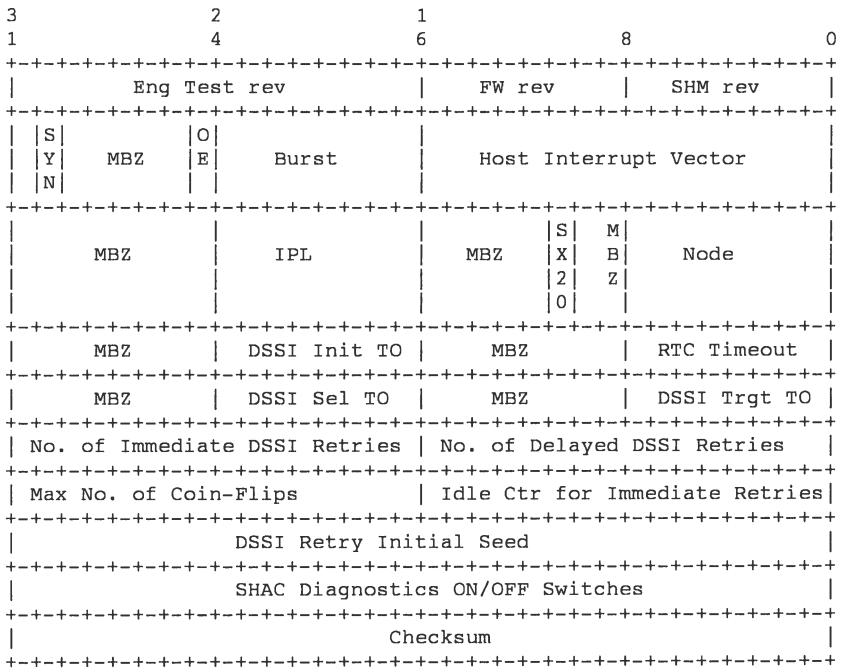
C.2 Shared Host Memory Header Area

This is the "root" area of the Shared Host Memory. It contains pointers to the other areas in Shared Host Memory. Note that the offsets to the other areas are offsets in Physical memory, NOT virtual memory. Note also that the checksum does NOT include the values in the reserved area. (It is recommended that the reserved area be zero.) This portion of the Shared Host Memory is specific to the particular SHAC, and cannot be common to multiple SHACs on the host bus.



C.3 Shared Host Memory Parameter Area

This area contains data required to initialize the SHAC . It has a checksum at the end. All fields are binary numbers. This portion of the Shared Host Memory is specific to the particular SHAC, and cannot be common to multiple SHACs on the host bus. The format of this area is:



There are default values for all the parameters in this area. If the host writes 3FFFFFFF (hex) to SSHMA, the SHAC uses the default values instead of loading Shared Host Memory.

Table C–1: Parameter Area Field Descriptions

Field	Default(hex)	Description
SHM rev	FF	Revision number of the Shared Host Memory, used as 8 bits of the CODE_REV field in ID packets
FW rev	NA	Revision number of the Firmware with which this Shared Host Memory may be used
Eng test rev	NA	reserved for additional revision information, for engineering test purposes only
Host Interrupt Vector	400	Used by the HIS in host interrupts (<1:0> MBZ)
Burst	1	Maximum burst size in longwords for host bus transfers (1,2,4,8,16,32,64)
OE	0	Octaword Enable bit (bit 24)
SYN	0	Synchronous operation (bit 30)
IPL	14	Interrupt Priority Level, used to set the HIS IPL register (0-1F)
Node	0	DSSI node number (0-7)
SX20	0	SX clock speed (0=16mh, 1=20mh); setting must be consistent with the SX pin; (bit 10)
RTC TO	9E	Real-time clock timeout period; it is the actual number that the SHAC puts in the internal RTC register. (It is based on the SX clock.)
DSSI Init TO	2D	DSSI initiator timeout period; it is the actual number that the SHAC puts in the internal TOC register.
DSSI Trgt TO	27	DSSI target timeout period; it is the actual number that the SHAC puts in the internal TOC register.
DSSI Sel TO	5	DSSI selection timeout period; this is the initial value of the counter for a software loop that checks for the timeout.
No. of Immediate DSSI Retries	8	Parameter for the DSSI retry algorithm, the maximum number of retries to be done before invoking the "delayed retry" portion of the algorithm.
No. of Delayed DSSI Retries	100	Parameter for the DSSI retry algorithm, maximum number of delayed retries to be done before failing a command.
Max No. of Coin-Flips	A	Parameter for the DSSI retry algorithm, the maximum number of negative coin-flip decisions that can be made before automatically retrying a command.

Table C–1 (Cont.): Parameter Area Field Descriptions

Field	Default(hex)	Description
Idle Ctr for Immediate Retries	0	Parameter for the DSSI retry algorithm, counter for an idle loop in the "immediate" retry algorithm. Initially set to zero, it is included in case that the minimum time between immediate retries needs tuning.
DSSI Retry Initial Seed	A7524B79	Initial seed for the random number generator for coin-flip decisions in the DSSI retry algorithm
SHAC Diagnostics ON/OFF Switches	15	There are 2 bits for each of the diagnostics ROM check, Device Register Check, and CAM check. See note below for details. The default is not to perform diagnostics. The purpose is to allow chip testing even if one part of the SHAC fails its diagnostic.

NOTE 1

OE, BURST, and SYN are in the format of the HIS Control Register.

NOTE 2

Some of the SHAC self-test diagnostics may be performed optionally after the load of the Shared Host Memory Parameter Area and before the load of the Shared Host Memory Patches. These diagnostics are: ROM check, CAM check and Device Register check. There are 2-bit switches to control the performance of diagnostics during the load of Shared Host Memory. These bits are as follows:

- bit 0 : skip ROM check
- bit 1 : continue after ROM check failure
- bit 2 : skip CAM check
- bit 3 : continue after CAM check failure
- bit 4 : skip Device Register check
- bit 5 : continue after Device Register check failure

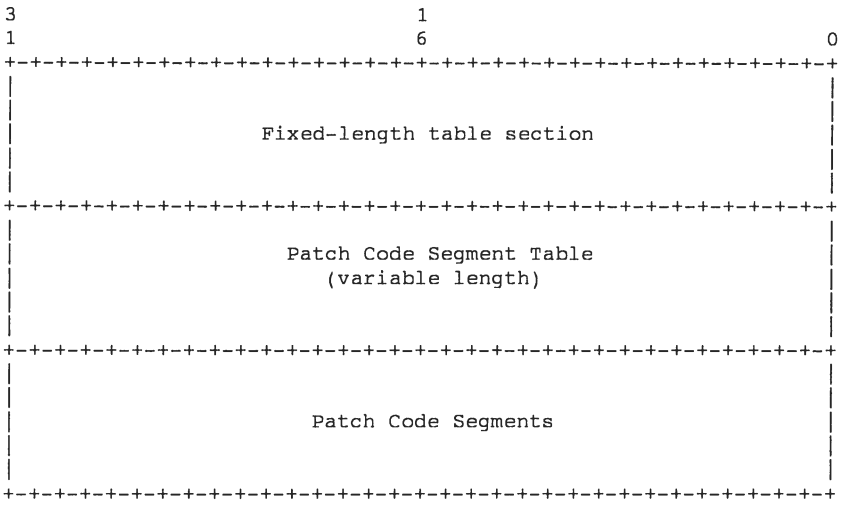
The purpose of these switches is to allow a diagnostic driver to detect a failure but to allow the SHAC to continue operation. It is expected that in normal operation no diagnostics will be performed during the load of Shared Host Memory.

C.4 Shared Host Memory Patch Area

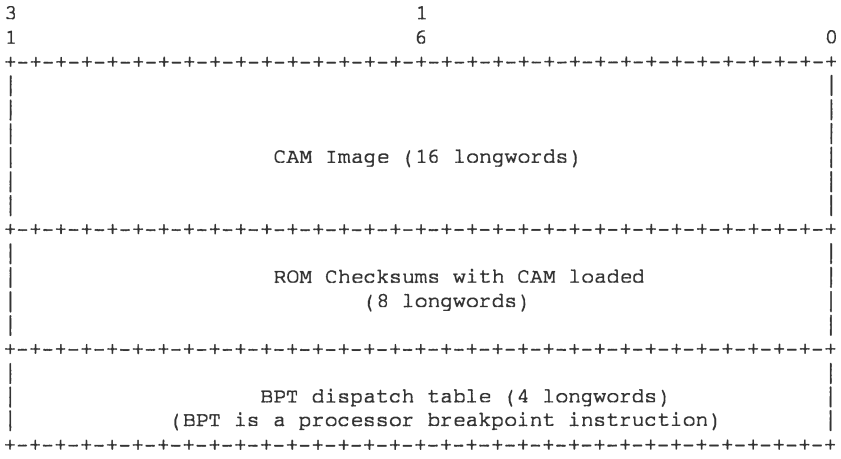
The patch area contains two table sections and a variable number of patch code segments. Note that each table and each code segment is octaword aligned.

Patching is done following chip reset or MIN bit reset, while the SHAC is still in the *uninitialized* state. Multiple SHACs on the host bus may share a common Patch Area.

The area's format is:

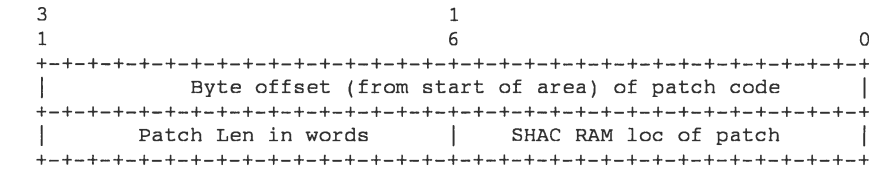


The Fixed Length Table section contains data which is used to load the CAM and BPT dispatch table. Note that the first and last ROM locations cannot be patched. The format of this area is:



The Patch Code Segment Table contains one code segment entry for each patch code segment. The format of the entry is shown in Figure C–1.

Figure C–1: Code Segment Table Entry



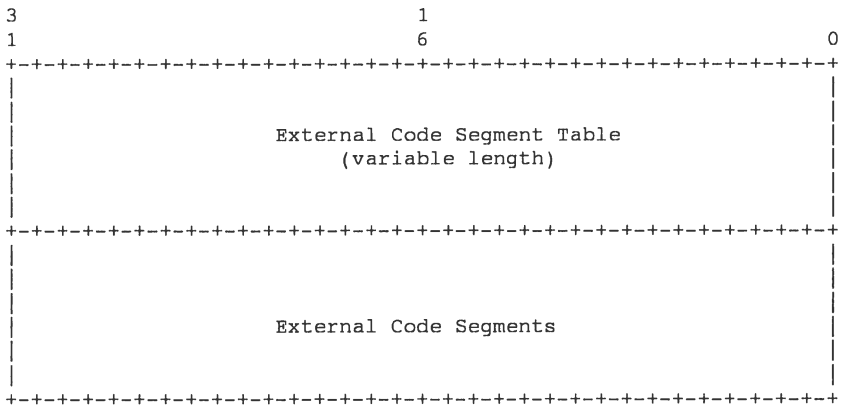
C.5 Shared Host Memory External Code Segment Area

This area contains modules that the SHACs on the host bus may load and execute at infrequent intervals. Multiple SHACs on the host bus may share a common External Code Area.

This area contains a variable number of External Code Segments. Each of these segments is self-contained, but may contain more than one routine. There is a table at the beginning of the area, with a fixed-length entry for each code segment. Each entry has the same format as that of a patch code segment (see Figure C–1). Note that each code segment is octaword aligned.

It is expected that External Code Segments will be used only for infrequently executed routines, where performance is not a critical issue. No data from this area is read by the SHAC until it is needed for execution.

The format of the area is:



C.6 Default Values for Shared Host Memory

If the host writes 3FFFFFFF (hex) to SSHMA, the SHAC does not load Shared Host Memory. Instead, it uses default values for the parameters in the Data Area, and it assumes there are no patches and no external segments. The default values for Shared Host Memory parameters are described above.

C.7 Boot Device on DSSI Bus

If the system boot device is on a given SHAC's DSSI bus, that SHAC will need Shared Host Memory parameter values before beginning to load the system. If the default parameters are appropriate (especially IPL and IVR) the host could tell SHAC to use the default parameters. Otherwise, a system will probably have a complete Shared Host Memory, or the code to set it up, in some kind of ROM.

C.8 Loading Shared Host Memory from a File

It may be desirable to update the Shared Host Memory in a system, in order to implement protocol changes or bug fixes in the SHAC. Such a system could be booted as described in the previous paragraph, with some or all of an updated Shared Host Memory then being loaded from a file into a dedicated area of host RAM. The port driver then would set the MIN bit in the PMCSR to force the port into the *uninitialized* state, and write the address of a new Shared Host Memory Header into the SSHMA register. Since the Shared Host Memory is read following each such write to SSHMA, this effectively would replace the previous Shared Host Memory.

Appendix D

SHAC TESTING

D.1 Overview

This appendix highlights some of the SHAC testing features and suggests a way of using them. This information is applicable for system level debug (hardware and software) and for supporting the following testing environments:

- Wafer testing.
- Packaged parts.
- Module testing.
- System level testing.

The following topics are discussed:

- SHAC testing modes.
- Vector oriented testing.
- Tristate and continuity transistor features.
- Host access feature.

D.2 SHAC Testing Modes

The testing port is the third port of the SHAC. This port is added to increase the observability and controllability of the SHAC. It should be used for testing purposes only.

Most of the SHAC logic and hardware control are done using an internal processor called QUIP. The QUIP has two main busses: the address bus (14 bits) and the data bus (16 bits). These busses are multiplexed and channeled to the IO pins called QAD(0:15). All the control lines of the QUIP are also accessible through additional pins in the testing port.

The testing port operates in four modes. The testing mode is selected by two pins - QTM1,2. Following are the four modes of operation:

- QTM1 = 0 ; QTM2 = 0 - Normal mode. Testing port is tristated.
- QTM1 = 1 ; QTM2 = 1 - Trace mode. The QUIP's address, data and control lines are driven out. The SHAC operates normally.

- QTM1 = 0 ; QTM2 = 1 - QUIP mode. The QUIP is the only functional block in the SHAC, all other blocks are disabled. In this mode the SHAC is the master of the testing port. The address is driven out through the QAD lines. Data is driven in/out depending on the value of QWR_L; low = write, high = read.
- QTM1 = 1 ; QTM2 = 0 - External QUIP. All blocks function normally except for the QUIP. The QUIP is disabled. The SHAC testing port is slave. The SHAC memories and registers can be read/write through the testing port.

Note that:

1. A SHAC in QUIP mode connected to a SHAC in external QUIP mode will operate exactly (phase by phase) as one SHAC.
2. The normal mode is the only mode in which the SHAC can run at its full speed. In all other modes the SHAC will run at about 25% of its maximum speed.

D.3 Vector Oriented Testing

The following notes should be considered when doing any vector oriented testing (i.e. TAKEDA):

1. When the SHAC operates in the normal mode (QTM1 and QTM2 zero) the host interface clock (pins CLKA, CLKB) and the DSSI clock (pin SX) are not guaranteed to be synchronized. The synchronization can be done only if SHAC is in non-normal mode and when the RESET_L is deasserted. After the synchronization occurs (one CLKA cycle after the deassertion of RESET_L) it is possible to change the SHAC mode to the normal mode and the clocks will remain synchronized.
2. Production testing can achieve a very high coverage with a minimal vector count, when done in an 'external QUIP' mode (QTM1 is 1, QTM2 is 0). In this mode all SHAC storage nodes accessible by the QUIP are accessible from the testing port. These tests shall be done with a maximum frequency of 5Mhz (frequency measured on CLKA).

   In this mode the SHAC ROM, RAMs and registers can be tested. All the ROM transistors are checked simply by reading all ROM addresses. The registers and RAMs are checked by writing 5s to all even rows and As to all odd rows, and reading these locations. Then doing the same but with As to even rows and 5s to odd rows. The following pseudo code implements the chase board pattern:

```
RAMs (QRAM and ARAM) testing pseudo code
     (radix=16 for all values)
```

```
X=0000
Y=FFFF
i=2000                                  ;  QRAM test.
WHILE i<2550 DO
    BEGIN
        M(i)=X                          ;Odd line
        M(i+1)=Y
        M(i+2)=X
        M(i+3)=Y
        M(i+4)=X
        M(i+5)=Y
        M(i+6)=X
        M(i+7)=Y
        M(i+8)=Y                        ;Even line
        M(i+9)=X
        M(i+A)=Y
        M(i+B)=X
        M(i+C)=Y
        M(i+D)=X
        M(i+E)=Y
        M(i+F)=X
        i=i+10
    END

i=2800                                  ;  ARAM test.
WHILE i<2A80 DO
    BEGIN
        M(i  )=X                          ;ODD LINE
        M(i+1)=X
        M(i+2)=Y
        M(i+3)=Y
        M(i+4)=Y
        M(i+5)=Y
        M(i+6)=X
        M(i+7)=X
        i=i+8
    END
; Then read above locations and check the data.
```

3. The testing port (signals Q*) should be ignored when operating with clocks above 5Mhz (frequency measured on pin CLKA).
4. When measuring the chip power consumption (power supply current) the SHAC must be in normal mode. In any other mode the testing port consumes a significant amount of power.

D.4 Tristate and Continuity Transistor Features

The SHAC has two features to aid in testing the SHAC, embedded in an external environment.:

1. The tristate feature allows SHAC to effectively be removed from testing.
2. The continuity transistor feature (together with the tristate) allows testing for open and short connections between SHAC and the surrounding environment.

These features can help in the following testing phases:

1. During die test on a wafer, checks that all IO lines are connected properly. This is a basic 'opens' and 'shorts' test.
2. After chip packaging, to test that the package pins are connected properly to the chip pads.

3. After module assembly, to test that the board wires are connected properly to the chip pads.
4. During module test, all SHAC drivers can be tristated, so that the SHAC does not interfere while testing other functions on the board.

These features are based on the pins QWR_L/TRISTATE_L and QHLD/SCAN_OUT_H.These pins operate as follows:

1. Pins QTM1 and QTM2 should be Grounded (normal mode). In this mode the pin QWR_L/TRISTATE_L functions as a TRISTATE_L and the pin QHLD/SCAN_OUT_H functions as SCAN_OUT_H.
2. When TRISTATE_L is asserted (low) and SCAN_OUT_H is deasserted, ALL pins are tristated. This function does not depend on the presence of the clock signals or on the internal state of the chip.
3. When TRISTATE_L is asserted (low) and SCAN_OUT_H is asserted (high), all pins are connected to ground via a pull down resistor (open drain transistor)

The following memo is the Rigel Tristate and "Continuity Transistor" Specification and Usage During Loaded Module Test Process V2.0. Although this memo describes the use of the feature during the module test, this feature can be used also in the phases listed above.

D.4.1 Rigel Tristate and "Continuity Transistor" Specification

```
 ___ ___ ___ ___ ___ ___ ___
|   |   |   |   |   |   |   |
| d | i | g | i | t | a | l |   I N T E R O F F I C E   M E M O R A N D U M
|___|___|___|___|___|___|___|

To:  Distribution                        Date: 20 October 1986 14:38:39
                                         From: John Sweeney
                                         Dept: Advanced Test Technology
                                         DTN : 289-1783
                                         L/MS: AP0-2/C15
                                         ENet: HAZEL::SWEENEY

Subject: Rigel Tristate and "Continuity Transistor" specification and usage
      during loaded module test process V2.0
```

D.4.1.1 Overview

This memo will present the specifications and test plans for the tristate control pin and the continuity test transistor structure.

D.4.1.2 Tristate

We have requested a pin on each chip which will, when asserted, cause ALL signal outputs of that particular chip to be put in the high impedance (tristate) state.

D.4.1.2.1 Purpose

During many phases of the test process, it is desirable to test a subset (either a single chip, or a group of chips) of the module functionality. Unfortunately, when testing one subset of the module, other functions may interfere with this testing if active. Additionally, the tester may damage parts if it has to overdrive them to achieve the required logic levels.

The best method to de-activate the portions of the module which are NOT currently being tested is to cause all outputs of all the chips not being tested to cease driving. This can be accomplished by each chip having one pin which, when asserted causes all of that individual chip's outputs to become tristated.

D.4.1.2.2 Specification

Each chip shall have one pin which when asserted will tristate ALL of that chip's outputs. This pin should be active low so that it can be asserted during the power-up of the module and internally pulled up through a 6:2 n-channel transistor to VDD_INT. This pin will be called TRISTATE_L.

The tristate function must not depend on presence of any clocking or the state of any other pins or internal states, ie. when that pin is asserted, the outputs must be tristated even if the clocks on the module (into the chips) are not operating, or the chip is not initialized.

D.4.1.2.3 Test Plan For Usage During Module Test

The tristate feature will be used during both Assembly Verification Testing (AVT) and Product Performance Test (PPT). In both cases the module will be fixtured such that the tester has electrical access to internal module signals.

First all tristate pins will be grounded (asserted), then the module will be powered up. Next, one chip at a time or in groups of chips, the tristate control pin(s) will be deasserted, and the chip(s) will be initialized, and then tested for functionality. These test, because of the additional loading from the tester contact, will be run at less than full clock rate.

The source of the test vectors for initialization and functional testing will be either from a simulation of chip DVT's or a capture of a module DVT or self-test.

D.4.1.2.4 Considerations During Module Design

The tristate control pin of each chip should, once the module is assembled, be connected to separate signal networks and accessible to the tester so that each chip can selectively be tristated under tester control. The reason for this is so that while one chip is being tested (not tristated) the other chips which would normally drive the chip under test could be tristated.

D.4.1.3 Continuity Transistor Structure

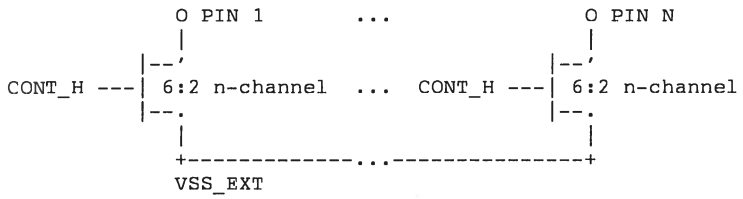
We have requested a transistor structure be designed into each chip to aid testing for opens between the module and the chips.

D.4.1.3.1 Purpose

Because the devices will be surface mounted to the module, open faults between the module and the chips are expected to be a large percentage of the manufacturing defects. Typically these open faults are difficult to detect and diagnose, usually requiring the development of a set of test vectors which will be applied to the chip. To simplify the test for opens, a Continuity Transistor structure should be designed into each chip.

D.4.1.3.2 Specification

A continuity transistor structure should be designed into each chip to facilitate testing for opens. This structure (shown below) will allow testing for opens by using simple instruments such as voltage sources and current meters.



The internal node CONT_H can only be asserted when the chip is tristated by asserting TRISTATE_L, otherwise it is pulled down. When TRISTATE_L is asserted, SCAN_OUT_H is enabled as an input and controls CONT_H through an input buffer.

Here PIN 1...PIN N represents ALL signal pins on the chip with the ONLY exceptions being:

- SCAN_OUT_H because this pin is already the common connection to the test structure.
- PHI_12, PHI_23, PHI_34, and PHI_41, because these pins will be driven as the system starts operation in phase 2 due to SYS_RESET_L being asserted and can't be overdriven.
- TRISTATE_L as this pin deactivates all output drivers and it's assertion would invalidate the continuity test.

The transistors should be sized such that as each pin is forced to 5.0 volts, the current into that pin will vary between 100 uA and 500 uA, depending on the process parameters. This will be accomplished by using 6:2 n-channel devices. The complete structure will be implemented in the external region of the chip.

This design does not require any dedicated pins as the test pin will be SCAN_OUT_H.

D.4.1.3.3 Test Plan for Usage During Module Test

The continuity structure will be used during Assembly Verification Testing (AVT) . The module will be fixtured such that the tester has electrical access to at least one point on each internal module signal network.

The test pin of each device (SCAN_OUT_H) will be fixtured such that the digital drivers can be connected to those pins.. All other pins will be fixtured such that the voltage source and current meter of the tester can be connected to that signal network as well as the digital drivers.

1. Check for any shorts on the module using the standard resistance test. If any are found, record them and check that they won't cause damage or impair further testing.

2. Force all pins to 0 volts, including VDD_INT, VDD_EXT, SYS_RESET_L, all TRISTATE_L signals and all SCAN_OUT_H signals. SYS_RESET_L is an input to the clock chip and will cause the clocks to be in the phase two state.
3. Next, power will be applied to the module using the normal (VDD_INT followed by VDD_EXT) power-up sequence. Because all the tristate and reset pins are active low and currently at 0.0 volts, all chip outputs will be tristated.
4. Now, for each chip to be tested:

   A. SCAN_OUT_H is currently low which will bring internal node CONT_H low, turning off all of the continuity transistors.

   B. For each pin on the chip under test:

   o Program the voltage source and current meter to connect to that pin of that device. Source a voltage of +5.0 volts, allow time for the current to settle, then measure the current. The current should be below 10 uA.

   o Drive SCAN_OUT_H to 5.0 volts, which brings internal node CONT_H high, turning on all of the continuity transistors.

   o Allow time for the current into the pin being tested to settle, then measure the current again. The current should now be above 300 uA.

   o Drive SCAN_OUT_H to 0.0 volts, which brings internal node CONT_H low, turning off all of the continuity transistors.

   o Remove the voltage source and current meter from the test pin of that device and attach a digital driver to that pin, driving 0.0 volts.
5. Opens testing is complete, either power down the module, or go on to more exhaustive testing.

D.4.1.3.4 Considerations During Module Design

The continuity test pin of each chip should, once the module is assembled, be connected to separate signal networks and accessible to the tester so that this continuity test can be properly performed.

D.5 Host Access Feature

D.5.1 Purpose

This feature enables access to all the QUIP address space from the host bus. This feature can be used for debug and diagnostics. Some possible uses are:

1. The host can activate the SHAC DMAs and perform data transfers over the CP bus and/or the DSSI bus. Such testing sequences can help to check the connections of the SHAC to the board and the connections of the DSSI wires and connectors.
2. The host can modify the SHAC code (by writing to the CAM and the RAM).
3. Perform an internal check of the ROM, RAM and device registers to check if the chip is functional.

D.5.2 Description

This feature may be invoked at the same time as a MIN bit reset, or during normal SHAC operation.

To invoke this feature, the host must first set Bit 4 in the PMCSR (this bit only needs to be set once). After that, host accesses are allowed until there is a chip reset or a MIN bit reset.

The host should then set SSHMA (host address XX44) as follows:

```
                ,-----------------------------------------------.
SSHMA           |OP|  Quip Address       |    Data               |
                `-----------------------------------------------'

  Bits <31:30> = OP

  IF host requests an operation (bits <31:30> ={10|11}) then

          Normal SHAC work is stopped until the host
          writes 01 to the two high-order bits of SSHMA.

             IF OP=11 THEN operation=READ; SSHMA<16:29>=QUIP address.
                           QUIP will write to SSHMA<0:15> the data.
                           And then writes OP=00.
             IF OP=10 THEN operation=WRITE; SSHMA<16:29>=QUIP address,
                           REG<0:15> data to write.
                           Quip will write the data to the Quip address
                           and then write OP=00.

             IF OP=01 THEN return to normal work

             (Note that if the host writes OP = 00, the SHAC will
                  also return to normal work.  However, the host
                  will not know when the SHAC wrote bits <31:30>.

          After the QUIP finishes with the operation it writes
          OP=00, to signal the host that it is done with the
          previous operation.  The host is not allowed to initiate
          an operation before OP=00.
```

D.5.3 Using the Host Access Feature After a MIN Bit Reset

Normally, the Host Access Feature operates from an interrupt service routine. If, for any reason, this becomes impossible, the host may reset the Quip and enter the Host Access Feature immediately by setting both the MIN bit and the Host Access Feature bit in PMCSR. The Quip stores an image of the Quip registers in a special area of RAM, and then processes host commands. This feature can be useful for debugging if the SHAC "hangs".

Appendix E

Error Codes

SHAC generates various types of error codes, which are reported via the host-addressable registers. In general, SHAC reports all errors as specified by CI Port; this appendix discusses those errors for which SHAC reporting differs from that spec.

E.1 Data Structure Errors

SHAC checks for and reports all Data Structure Errors as specified by Appendix D of CI Port, except as noted in this section.

- Q_INTL_FAIL—SHAC always attempts a queue interlock operation until it succeeds. Therefore, this error never is reported.
- ILL_PQB_FRM—CI Port specifies that the check for this error is optional. SHAC performs the check and reports the error if detected.
- REG_PROT_VIOL—This error is reported only when the port driver writes to PECR without having first written to PQBBR.

E.2 Miscellaneous Errors

Miscellaneous Errors should occur only as a result of a SHAC internal bug (almost certainly software). The error code is reported in the MISC field of PESR as specified by Chapter 8 of CI Port.

The SHAC-specific error codes are not listed here, since understanding them requires a level of familiarity with the SHAC internal software that is beyond the scope of this document.

E.3 Maintenance Errors

A Maintenance Error is reported when a very serious internal SHAC error occurs. In addition to setting PSR<MTE>, as required by CI Port, SHAC provides further information on the error in the host-addressable registers.

In addition to MTE, one other bit in PSR (as listed in Chapter 4) should be set, indicating the type of error. The following sections list these errors, and describe which other registers contains what additional information.

E.3.1 Shared Host Memory Error

This indicates the occurrance of an error in the contents or processing of Shared Host Memory. The specific error is indicated by a bit in PESR<31:16>:

PESR bit	error name	description
16	Channel Status	A host bus error occurred while reading from Shared Host Memory into the SHAC. PFAR contains the approximate host address at which the error occurred.
17	Header Checksum	The checksum for the SHM Header Area is incorrect. PFAR contains the approximate host address of the checksum.
18	Parameter Checksum	The checksum for the SHM Parameter Area is incorrect. PFAR contains the approximate host address of the checksum.
19	ROM Checksum	One of the ROM checksums in the SHM Patch Area is incorrect. The contents of other registers are as described in Section E.3.4.

Refer to Appendix C for information about the format of Shared Host Memory.

E.3.2 Slave Mode Parity Error

This indicates that a host bus parity error occurred on a host access of one of the SHAC registers. PFAR contains the register's byte offset within the set of SHAC registers, as listed in Section 4.3.

E.3.3 Illegal Segment Number

The SHAC software attempted to read an External Segment that is not present in the Shared Host Memory. PFAR<15:0> contains the number of the segment. PFAR<31:16> contains the QUIP address of the SHAC code that attempted to read in the segment.

This error also will occur if the SHAC software attempts to read any External Segment after the host has written 3FFF FFFF into SSHMA, as described in Section 5.1.2.3.

E.3.4 Diagnostic Error

An error was detected while running a SHAC internal self-test. The failing test is indicated by a bit in PESR<31:16> and additional information can be found in other registers, as shown:

PESR bit	test name	register	contents of register
16	RAM diagnostic	PFAR<15:0>	QUIP address at which the error occurred
		PFAR<31:16>	pattern written to that address
		PESR<15:0>	pattern read from that address after the write
17	ROM diagnostic	PFAR	a bit vector; each PFAR<7:0> set indicates that the checksum for the nth KW of ROM is incorrect
18	CAM diagnostic	PFAR<15:0>	QUIP address of ROM location that test was attempting to patch
		PFAR<31:16>	QUIP address of CAM entry that was being used
		PESR<15:0>	patch value that test was attempting to use
19	device register test	PFAR<15:0>	QUIP address of device register
		PFAR<31:16>	value written to that register
		PESR<15:0>	value read from that register after the write

All of these diagnostics are run following power-up, so any of these errors may be reported then.

Some or all of these tests (except for the RAM diagnostic) may be run following the read of Shared Host Memory, as described in Appendix C. If the SHM specified that tests should continue even after a failure, PSR<DE> will be set but PSR<MTE> will not, and no Maintenance Error interrupt will occur. If more than one test fails in this situation, more than one of the bits in PESR<31:16> will be set.

E.3.5 QUIP-Detected Error

This indicates that a SHAC internal software error was detected by the QUIP hardware. The following information can be found in other registers, though interpreting it requires a level of familiarity with the SHAC internal software that is beyond the scope of this document:

register	contents
PFAR<15:0>	QUIP address that triggered the error
PFAR<31:16>	may be the SHAC code location at which the error occurred
PESR<15:0>	the value in the QUIP internal ERR register
PESR<31:16>	the value in the QUIP internal SP register

Additional information can be found in the SHAC internal RAM, at a location that is dependent on the Firmware Version.

E.3.6 Illegal Interrupt

The QUIP received an interrupt that it shouldn't have. The following information can be found in other registers:

register	contents
PFAR<15:0>	the most significant bit set indicates the level of the interrupt
PFAR<31:16>	the SHAC code location at which the interrupt occurred