SCSI Inquiry Path

Version 0.5

Prepared by Garry Ng

Hardware Engineering

04/21/2015

Revision History

Name	Date	Changes	Version
Garry Ng	10/7/14	Created	0.1
Garry Ng	10/14/14	Updated versioning, added single inquiry implementation, gave integration overview and some more details of the SCSI inquiries	0.2
Garry Ng	12/8/14	Changed architecture to separate path	0.3
Garry Ng	12/15/14	Changed inquiry detection architecture, updated RR arbiter behaviour and DPL buffer transfer behaviour.	0.4
Garry Ng	04/24/15	Added wrapper. Updated microarchitecture.	0.5

Table of Contents

Table of Figures

1. Introduction

One type of SCSI command that is currently not examined in depth is the SCSI inquiry command, which returns physical and logical information about the device being queried (such as type and manufacturer information). All devices must respond to commands with the opcode 0xC. The response can be decoded and provided as additional data for the user, such as their T10 vendor information, serial number, EUI-64, or NAA. Vendor-specific information would be useful as well depending on the product being examined.

The outgoing request can be easily detected with the unique opcode 0xC in the first 8 bytes, but the response data contains no unique identifying information that is common to all vendors and devices. The challenge is hence to correctly identify and match outgoing responses to their corresponding incoming responses when their sequence order cannot be known ahead of time.

The document also goes over proposed alternatives and their tradeoffs, as well as considerations for partially off-chip solutions. One on-chip implementation of the SCSI inquiry frame decoder is shown in the detail this document.

1.1. Reference

- Fiber Channel Framing and Signaling – 4 (FC-FS-4) (Rev 0.10)
- JIRA BAL-63 (http://jira.vi.local/browse/BAL-63?jql=text%20~%20%22inquiry%22)
- SCSI Commands Reference Manual Section 3.6.2 (http://www.seagate.com/staticfiles/support/disc/manuals/scsi/100293068a.pdf)
- Hash Collision Probabilities by pershing on programming (http://preshing.com/20110504/hash-collision-probabilities/)
- SCSI Primary Commands 3 (SPC-3) 2003 Sections 6.4 and 7.6

1.2. Acronyms

- **SOF:** Start of frame
- **EOF:** End of frame
- **SOP:** Start of packet
- **EOP:** End of packet
- **CAM:** Content-addressable memory

2. SCSI Inquiry Frame Overview

The structure of the SCSI inquiry frame is described by the SCSI Primary Commands – 3 (SPC-3) specifications. The exchange consists of a command and accompanying response. While this document does not cover all the possible mandatory and optional supported requests, an overview can be found in Figure 1. This overview covers the 4 responses that are of interest to the user, and what major statuses or device information can be extracted from them. Standard data and vital product data (VPD) are the two main forms of responses that will be dealt with, while command support data is not used for our purposes.

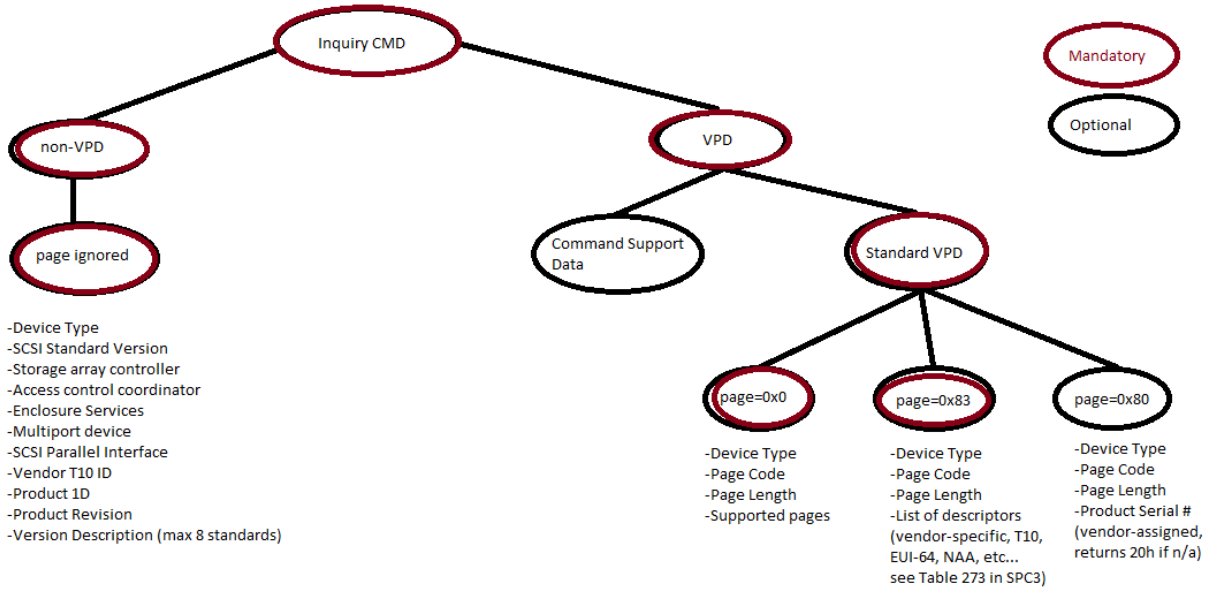

Figure 1 - Overview of SCSI Inquiry Response variations.

To uniquely associate a LUN with a globally unique identifier, the product descriptors used in the mandatory page 0x83 VPD response will be used. Specifically the descriptor of interest is the standardized NAA identifier format. For more detail, refer to 7.6.4.5 of the SPC-3 specification.

On the Fiber Channel header side, we need require R_CTL to identify whether something is a command or response, the TYPE field to identify a SCSI-type command, and F_CTL to identify whether a frame is an exchange originator or responder in order to corroborate that an opcode of 0xC, enabled VPD, and page code of 0x83 in the SCSI CMD data is a unique SCSI Inquiry frame.

3. Top Level (scsi_inq w/ single pending inquiry)

Figure 2 shows the high level overview while **Error! Reference source not found.** shows a predicted timing diagram for the scsi_inq module (the primary detector) when an inquiry frame and its corresponding response are passed though the link (extended headers are currently not supported). The latency of the module does not affect the primary datapath. There is one inquiry link FIFO per link, and 1 DPL buffer per FPGA.

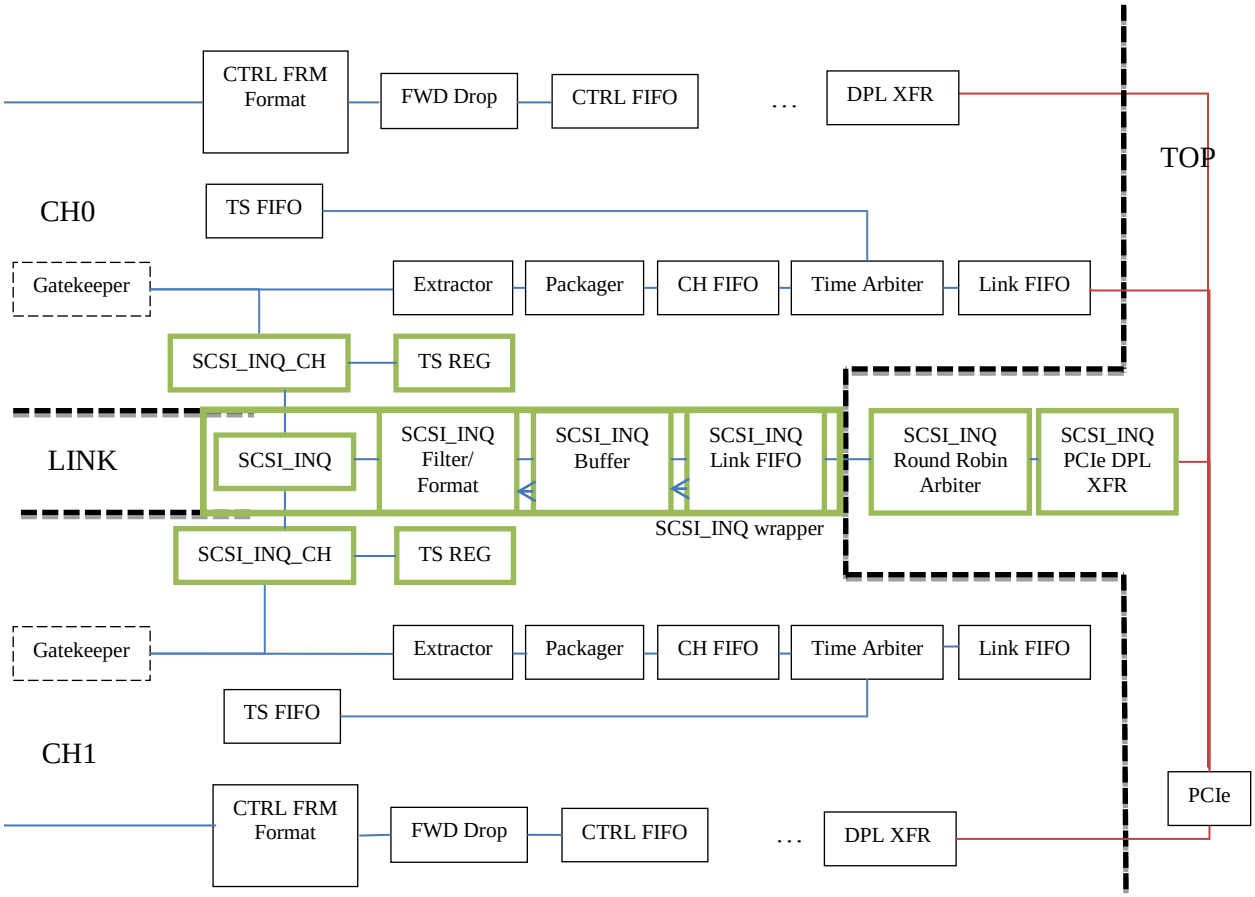

Figure 2 - Integrated modules for new SCSI inquiry path. Green modules are new modules to be added. Backpressure occurs from the SCSI_INQ link FIFO and ping-pong buffers.

3.1. Specification Summary

- Must detect SCSI inquiry frames based on criteria in Section 3.2
    - Must be configurable to accommodate frame padding in FCoE
    - Uses a latched pipeline to accommodate any number of cycles between two valid cycles on the streaming interface
- Must detect corresponding SCSI response frames based on stored D_ID, S_ID, and OX_ID
    - Matches on only the latest CMD seen per link
- Detected SCSI_INQ frames are forked into a separate SCSI_INQ path
    - Original data path must not be affected
- Metadata is added according to the output in Section 3.3
- All packets are 512 bytes long and contain both the CMD and RSP
- Packets are stored inside a FIFO RAM 512 bytes deep and 128 bits wide while it is being constructed
- Once a complete valid set is detected, the packet is zero-filled at the end and forwarded to an inquiry link FIFO
- The inquiry link FIFO is 4K bytes deep
- The inquiry link FIFO writes data on a 128b bus and is read from on the PCIe clock domain on a 256b bus
- A round robin arbiter arbitrates between all links on an FPGA and connects the active link FIFO to a buffer which connects to the PCIe arbiter (or the DPL_XFR module in FCoE)
    - The link FIFO that has control over the arbiter transfers all its stored packets to the arbiter's buffer, which in turn transfers its contents to the PCIe app
    - When one link FIFO is depleted the next link with a request (ie. not full) takes control
    - If less than 8 packets are passed before all FIFOs are empty, then the arbiter zero-fills the remaining difference
    - If more than 8 packets have been passed, the 4K transfer is completed, the arbiter must re-requests the PCIe bus to grant control before it continue to pop data from the link FIFOs
    - The round robin arbiter operates on a shifting priority basis (each transfer a different link has priority)
    - On every time interval boundary, an interval packet is sent by the arbiter as the last packet in a <4K transfer, or the 1st packet of the next 4K transfer
    - If no requests are made at the interval boundary, a 4K transfer occurs with only the interval stat packet
    - The arbiter will disable transfers to the PCIe bus if the DPLBUF is full as indicated by pointers (implicitly done by the DPLBUF not issuing grants to the arbiter)
- The packets are time order agnostic (no time arbiter)
- Error signals (MAC or FC2) detected during a frame that contains a CMD or RSP will invalidate both frames and cause a flush of the buffer
- If the length of the packet exceeds 512 bytes, it is dropped by the buffer

- If a manual flush or link down is encountered, completed sets in the buffer that are being zero-filled are accepted into the link FIFO. Incomplete sets are dropped from the buffer immediately.
- If the link FIFO is full, data can still be written to the buffer, and wait for the link FIFO to free up. If the buffer is also full, frames are dropped (and accounted for).
- A drop counter is implemented to show the number of inquiries dropped by the buffer (for any reason) and by the link FIFO (for any reason)
- Resettable and readable counters for # CMDs and # RSPs found per link
- A LinkCtrl interface will need to be added for the additional DPL buffer – it shall have the same monitor mode control bits (disabling this bit on the DPL will cease all transfers to the DPL buffer, while disabling this on the individual links will cease inquiry detection on only that link)
- A status bit to indicates the flush status each individual link FIFOs
- A link down has the same effect as the manual flush on that link

3.2. Inquiry Frame Match Criteria

There are 4 8-byte buffers required to catch all the relevant data to determine whether an incoming frame is an inquiry command (extended headers would require an additional cycle, but they are not supported). Refer to the FCP-SCSI and SPC-3 specs for details on the frame contents. At the appropriate cycle, each of the following fields are checked:

- R_CTL: Arrived on valid cycle 0, in top buffer input bits
- TYPE: Arrived on valid cycle 1, in top buffer input bits
- F_CTL[top bit]: Arrived on valid cycle 1, in top buffer input bits
- SCSI CMD OPCODE: Arrived on current valid cycle, check input bits 24 to 31
- EVD: Arrived on current valid cycle, check input bit 39
- Page Code: Arrived on current valid cycle, check input bit 40 to 47

The response frame is matched when the previous CMD frame's D_ID, S_ID, and OX_ID are matched to the incoming frame's own. Note that D_ID and S_IDs are switched on the response.

3.3. Output DPL Packet Format

The format of the output DPL packet produced by the inquiry path is shown in Table 1. The packet has a fixed size of 512 bytes and contains both the command and its associated response. The command portion always has a fixed length, but the response will have a variable length. The packet is padded with zeros at the end. Refer to SPC-3 7.6.4.5 NAA identifier for the SCSI VPD ID descriptor of interest.

Packet Type	Heading	Contents	Size
SCSI INQ (CMD)	Packet Type	Bits 3:0 = 6 (for CMD)	1B
	Timestamp		7B
	Link		1B

	Channel		1B
	Reserved	Reserved	6B
	FC Header		24B
	FCP_CMND	LUN	2B
		[Don't Care]	10B
	SCSI Inquiry	Opcode	1B
		CMDDT/EVPD	1B
		Page	1B
		Allocation Length	2B
		Control	1B
		0-fill	10B
	FCP_CMND	FCP_DL	4B
(RSP)	Packet Type	Bits 3:0 = 7 (for RSP)	1B
	Timestamp		7B
	Link		1B
	Channel		1B
	Reserved	Reserved	6B
	FC Header		24B
	SCSI VPD Payload Header	Device Type	1B
		Page Code	1B
		Page Length (n-3)	2B
Repeats ->	SCSI VPD ID Descriptor	Protocol ID	4b
		Code Set	4b
		PIV	1b
		Reserved	1b
		Association	2b
		Identifier Type	4b
		Reserved	1B
		Identifier Length (m-3)	1B
		Identifier	(m-3)B

Table 1 – Output DPL format of SCSI Inquiry Frames. Red indicates direct input from the data path.

4. misc_le_wrap

The wrapper serves to organize all the link engine-specific components under one module. Only the final PCIe arbiter and the per-channel detectors do not belong in this module.

4.1. Signals

Input:

- iCLK: the core clock
- iCLK_PCIE: the pcie clock
- iRST_N: the reset signal
- iRST_PCIE_N: the reset signal
- iMISC_ARB_GRANT: the grant signal from the misc_rr_arbiter to this link
- iMISC_FIFO_CTRL[3:0]: the Ctrl signal from the global register set (global.g.misc) – operates similar to MonitorMode within the local link, but is the global version for scsi_inq
- iCHAN_LINKUP[1:0]: the linkup status of each channel (hold high for linkup)
- iREG_LINKCTRL_MONITORMODE[3:0]: the MonitorMode register set within LinkCtrl of the CSR
- iD_ID_OUT[1:0] [23:0]: the D_ID pulled from a standard frame header from each channel (valid only with IS_CMD/IS_RSP)
- iS_ID_OUT [1:0][23:0]: the S_ID pulled from a standard frame header from each channel (valid only with IS_CMD/IS_RSP)
- iOX_ID_OUT [1:0][23:0]: the OX_ID pulled from a standard frame header from each channel (valid only with IS_CMD/IS_RSP)
- iINQ_IS_CMD[1:0]: indicates valid SCSI inquiry command frame on each channel
- iINQ_IS_RSP[1:0]: indicates response candidate on each channel
- iINQ_*: a set of signals that contain the data path's signals
- iINQ_LAST_TS [55:0]: the timestamp from the latest packet

Output:

- oMISC_FIFO_DATA[255:0]: data from the misc_fifo packet to the arbiter
- oMISC_FIFO_DATA_V: data valid from the misc_fifo packet to the arbiter
- oMISC_FIFO_FULL: indicates that the misc_fifo is full (scsi_inq packets will start getting dropped soon as no more can be added)
- oMISC_FIFO_EMPTY: indicates that the misc_fifo is empty
- oMISC_FIFO_AEMPTY: indicates that the msic_fifo is almost empty (1 word left)
- oMISC_FIFO_RD_REQ: a request to the misc_rr_arbiter to be read from (packet stored)

4.2. Logic

This wrapper contains all the logic that belongs at the link engine level and serves as the “virtual link” for scsi_inq in a port channel interface.

5. scsi_inq_ch

The SCSI inquiry detector serves to extract the D_ID, S_ID, and OX_ID located in a frame header, as well as specifying the whether frame contains an inquiry command. The data inputs on the channel are examined, but not operated on so the data outputs are simply the pipelined version of the input in most cases. However, the pipelined behavior turns into a latched buffer for 4 valid data cycles when an SOP is detected. The module does not support extended frame headers (Virtual Fabric Tagging, Inter-Fabric Routing, and Encapsulation). The output of this module is not connected to the normal datapath.

5.1. Signals

Parameters:

- FCOE: 0 for FC, 1 for FCOE

Input:

- iCLK: the core clock
- iRST_N: the reset signal
- iFC_INQ_*: a set of signals that contain the data path’s signals
- iGLOBAL_TIMESTAMP [55:0]: the current global timestamp

Output:

- oD_ID [23:0]: the D_ID pulled from a standard frame header (valid only with oIS_CMD/oIS_RSP)
- oS_ID [23:0]: the S_ID pulled from a standard frame header (valid only with oIS_CMD/oIS_RSP)
- oOX_ID [23:0]: the OX_ID pulled from a standard frame header (valid only with oIS_CMD/oIS_RSP)
- oFC_INQ_*: a set of signals that contain the data path’s signals
- oIS_CMD: indicates valid SCSI inquiry command frame
- oIS_RSP: indicates valid response candidate frame
- oLAST_TS [55:0]: the timestamp from the latest packet

5.2. Logic

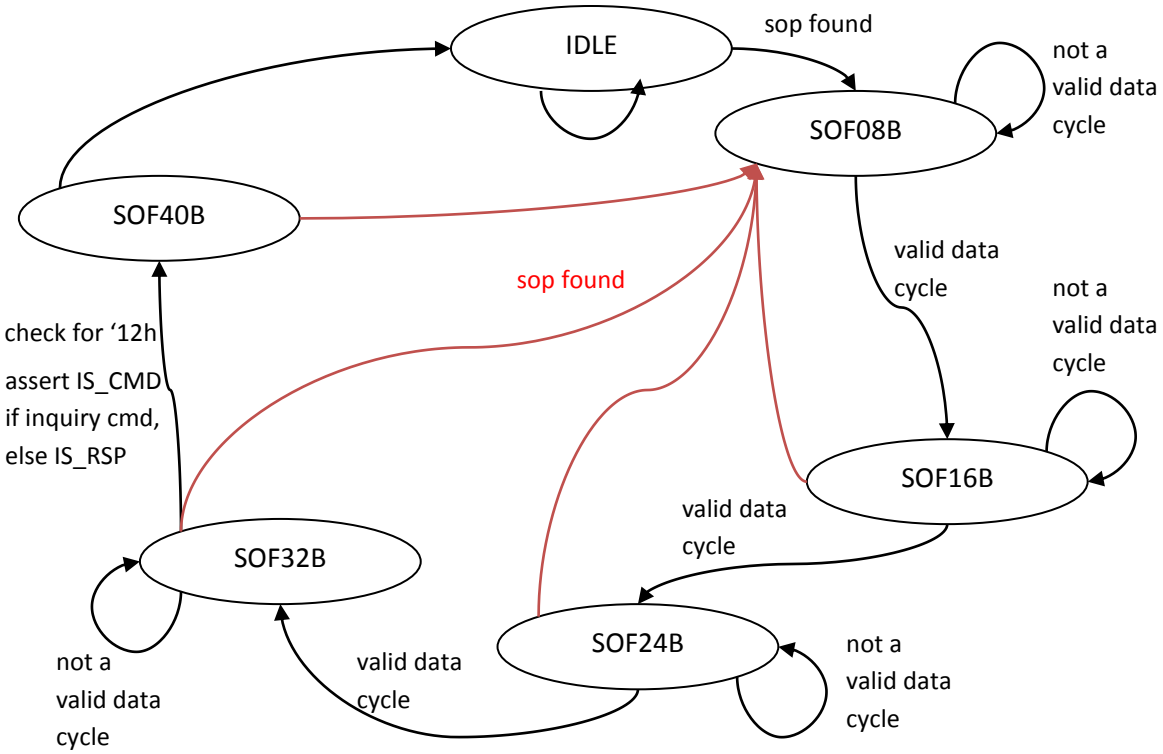

Figure 3 - The internal state machine of inquiry. All states have a

The incoming data is sent into 4 separate register blocks in order to have all the required frame information present on the same clock cycle. The system uses the finite state machine shown in Figure 3 in order to correctly extract the D_ID, S_ID, and OX_ID and check for the SCSI inquiry-specific codes.

It is assumed that the start of frame word is dropped when the SOP signal is asserted in standard FC mode. It is also assumed that an SOP signal is mutually exclusive from an error signal and that it will be asserted with data_valid. The timestamp is immediately stored in a register and made available downstream to the inquiry path. Every cycle after the SOP is seen, the appropriate checks mentioned in Section 3.2 are performed, and the module will assert IS_CMD if all criteria match, or IS_RSP otherwise. IS_RSP is only an indicator for a response candidate, and will require a module downstream to check for a D_ID, S_ID, and OX_ID match.

When the module is in its SOF40B or IDLE state, it will pipeline the information through all the registers, whether it is valid or invalid. However when an SOP is detected, the data oFC_INQ_VALID and oFC_INQ_SOP signals are forced low to start collecting all the valid data into 4 register sets. Each cycle that the iFC_INQ_VALID signal is held high, the data is pushed into a set of 64 bit registers, and each cycle it is not, the data is dropped. When all 4 sets of registers are filled, the SCSI_INQ CMD check is performed (see Section 3.2), and the latch is released to return to its normal pipeline behavior. Thus, the

output will always be 5 back-to-back valid cycles of data (with all invalid data dropped), followed by the replicated input thereafter.

In the case that the SOP signal is asserted before the expected end of the header information is reached, the FSM will defer to the signal and reset the state back to that of the first set of 8 bytes, effectively discarding the current information as a valid candidate for either inquiry commands or responses. This priority condition also acts as the timeout mechanism for incomplete frames.

In FCoE mode, the preexisting padding between the Ethernet header and the Start-of-Frame word must be accounted for when looking for the OPCODE by dropping or masking the padding.

6. scsi_inq

The scsi_inq module serves as cross-channel communication between the two inquiry detectors on each channel per link. Its job is to hold the last known SCSI inquiry command IDs and make the comparison

6.1. Signals

Input:

- iCLK: the core clock
- iRST_N: the reset signal
- iD_ID_OUT[1:0] [23:0]: the D_ID pulled from a standard frame header from each channel (valid only with IS_CMD/IS_RSP)
- iS_ID_OUT [1:0][23:0]: the S_ID pulled from a standard frame header from each channel (valid only with IS_CMD/IS_RSP)
- iOX_ID_OUT [1:0][23:0]: the OX_ID pulled from a standard frame header from each channel (valid only with IS_CMD/IS_RSP)
- iINQ_IS_CMD[1:0]: indicates valid SCSI inquiry command frame on each channel
- iINQ_IS_RSP[1:0]: indicates response candidate on each channel
- iINQ_*: a set of signals that contain the data path's signals
- iINQ_LAST_TS[55:0]: the timestamp from the latest packet
- iINQ_MATCH_EXPECTED: indicates a CMD is stored downstream and a RSP is expected

Output:

- oINQ_IS_CMD: indicates valid SCSI inquiry command frame on each channel
- oINQ_IS_MATCH: indicates valid SCSI inquiry response frame on each channel
- oINQ_*: a set of signals that contain the data path's signals
- oINQ_LAST_TS [55:0]: the timestamp from the latest packet

6.2. Logic

This module mainly serves to finalize the detection of the response frame, as inquiries can originate on one channel and have its response present on either channel in the link. When a command frame is detected, its IDs are stored in registers as the latest pending inquiry. If new inquiry commands are detected without resolving the previous one, then they are always overwritten with the newest set of IDs. When a matching response is found while the iINQ_MATCH_EXPECTED signal is high, the INQ_IS_MATCH signal is asserted on the correct bit associated with that channel.

All other inputs used later are passed through the module for hierarchy purposes.

7. scsi_inq_frmt

The scsi_inq formatter module's purpose is to package the raw data into the output format shown in Section 3.3. It also converts the bus width from 64b to 128b and is the first point of entry for external FIFO flush requests.

7.1. Signals

Input:

- iCLK: the core clock
- iRST_N: the reset signal
- iINQ_IS_CMD[1:0]: indicates valid SCSI inquiry command frame on each channel
- iINQ_IS_MATCH[1:0]: indicates valid SCSI inquiry response frame on each channel
- iINQ_*: a set of signals that contain the data path's signals
- iINQ_LAST_TS [55:0]: the timestamp from the latest packet
- iINQ_BUFFER_BUSY: indicates that the buffer is busy and not accepting any input
- iLINK_ID [3:0]: link #
- iFLUSH: a general signal that resets the state of the formatter and cause it to drop any further incoming packets (MonitorMode, Linkup events connect to this)

Output:

- oFLUSH: indicates buffer needs to be flushed (not link FIFO though)
- oWR_BANK_SLOT[1:0]: indicates which of the 64b of oINQ_DATA are valid (128b output)
- oINQ_START_SET: indicates beginning of a SCSI_INQ cmd/rsp set
- oINQ_END_SET: indicates valid end of a SCSI_INQ cmd/rsp end
- oINQ_DATA[127:0]: frame data for the inquiry
- oINQ_MATCH_EXPECTED: indicates the state machine of the formatter is looking for the RSP

- oINQ_PKT_ERR: inquiry frame error detect event
- oINQ_PKT_DROP: inquiry frame drop event (buffer is full)
- oINQ_PKT_OVERWRITE: inquiry packet overwrite event (CMD overwritten before completion of packet)

7.2. Logic

This module contains two state machines: one to track the state of the command and response pair, and another to track the writing of 64b data to 4 possible slots.

The writing state machine (wr_state) simply converts the 64b input to a 128b output by populating the 2 possible slots within a "bank" with data from the right channel. There are two banks (the HI and LO banks) which contain data. By default, both banks will have zeros written to them, but when the CMD/RSP state machine finds a command or response, it will pipe the data from the right channel (based on a sticky bit for last found CMD and RSP) to the HI bank [63:0] and go in a round robin. The LO bank will be initially filled with the DPL packet header information, and gets pushed immediately to the buffer as the HI bank starts getting filled with inquiry data. When the state machine starts writing to the LO bank, the HI bank is ready and its data is pushed. Pushes are made every 2 cycles of valid data until an EOP is encountered. If a cycle has invalid data, the state machine retains its state and overwrites the contents of the data bank (hence the valid data is the last thing written before a state machine change).

When an EOP is encountered in a valid CMD or RSP state, the remaining data is sent to the buffer, with the correct oWR_BANK bits asserted and proceeds to either IDLE or start writing to the HI bank again for back-to-back received frames.

The purpose of the CMD/RSP pair state machine is to track the correct number of bits to write to the buffer by tracking the EOP of the valid frame that a channel originates from. It moves between a number of 4 unique states for CMD and 4 for RSP. The idle state waits for either CMD to be asserted, the INQ_CMD_0 state is a special state that fills the LO bank with the packet header metadata, the INQ_CMD_1 state simply pushes all the FC data into the buffer and waits for a valid EOP to stop pushing data. The RSP versions of all the states do the same thing except wait for the INQ_IS_MATCH signal to transition.

In the event of a IS_CMD assertion in an unexpected state, the state machine defers to the new CMD and resets its state to CMD_0 and assumes the old inquiry command has been overwritten. In any case where there is an unexpected signal found or an error signal asserted upstream during a non-IDLE phase, it is assumed that the data is now junk, and a local flush is sent to the buffer to discard the accumulated data via oFLUSH. This does not cause the link FIFO to be pushed however.

8. scsi_inq_buffer

The scsi_inq buffer stores the current 512 byte CMD/RSP set being processed in a FIFO.

8.1. Signals

Input:

- iCLK: the core clock
- iRST_N: the reset signal
- iFLUSH: indicates buffer needs to be flushed (not link FIFO though)
- iFLUSH_ALL: indicates the buffer and the link FIFO needs to be flushed
- iWR_BANK_SLOT[1:0]: indicates which of the 64b of oINQ_DATA are valid
- iINQ_DATA[127:0]: frame data for the inquiry
- iINQ_START_SET: indicates beginning of a SCSI_INQ cmd/rsp set
- iINQ_END_SET: indicates valid end of a SCSI_INQ cmd/rsp end
- iINQ_LINK_FIFO_BUSY: indicates that the link FIFO is busy and won't accept new data

Output:

- oFLUSH_ALL: signal to the link FIFO to flush all stored packets
- oINQ_PKT_DROP: pulse to indicate a packet was dropped
- oINQ_DATA[127:0]: frame data for the inquiry
- oINQ_PUSH_LINK_FIFO: push valid inquiry packet data into the link FIFO

8.2. Logic

The link buffer is the point at which an inquiry packet is stored before it is released. This buffer is the cutoff point between what is guaranteed to be attempted to be sent to the DPL buffer and what is not.

The module contains a FIFO RAM that is 512 bytes in size, the exact size of a packet. Data is pushed into it and forced out into the link buffer when it is full.

The other component of this module is a 4-state FSM. While it IDLEs, it waits for an INQ_START_SET signal to push it into the PEND state, which records INQ_DATA based on the WR_BANK_SLOT signal. If an error occurs during this state, or it buffer fills before the INQ_END_SET signal is asserted, then it is too large and invalid. The packet would be dropped by flushing the buffer. Otherwise, it will enter the FILL state once INQ_END_SET is seen and zero-fill until the fifo_full signal is asserted from the FIFO. This is followed by the DONE state which essentially pushes the FIFO into the link FIFO until the aempty (almost empty means 1 word remaining) is asserted. If the link FIFO is full or is BUSY (both asserted by iINQ_LINK_FIFO_BUSY), then this returns the FSM into the IDLE state.

If during the buffer fill or push into link FIFO another valid set is detected, the state machine will opt to ignore it and drop it as it is locked with the current set. Valid sets will again be processed when it returns to the IDLE state.

9. misc_fifo_wrap

The misc fifo (or scsi_inq link fifo) holds 4K worth of data, and is connected via an round-robin arbiter to the DPL transfer module. It has the ability send requests to the round robin arbiter to send data to the DPL buffer.

9.1. Signals

Input:

- iCLK: the core clock (write side domain)
- iCLK_PCIE: the PCIe clock (read side domain)
- iRST_N: the reset signal
- iRST_PCIE_N: the reset signal for PCIe
- iFLUSH: a flush signal that causes the entire 4K FIFO to empty
- iMISC_DATA[127:0]: packet data (inquiry) for the misc FIFO
- iMISC_PUSH_LINK_FIFO: push valid inquiry packet data into the misc FIFO
- iMISC_POP_LINK_FIFO: pop valid inquiry 4K from the misc FIFO

Output:

- oMISC_DATA[255:0]: packet data for the inquiry
- oMISC_DATA_V: packet data valid for the inquiry
- oMISC_LINK_FIFO_EMPTY: empty indicator for read side clock domain
- oMISC_LINK_FIFO_AEMPTY: almost empty indicator for read side clock domain (1 word left)
- oMISC_LINK_FIFO_RD_FULL: full indicator for the read side clock domain
- oMISC_LINK_FIFO_RD_REQ: read request to the arbiter (when at least 1 pkt is available)
- oMISC_LINK_FIFO_BUSY: indicates that the misc FIFO is busy

9.2. Logic

The module contains a small state machine and a dual clock FIFO. When the FIFO is full, it will assert its RD_FULL signal out. It also has a counter for the number of packets stored, which when >=1, acts as a request downstream to the arbiter.

The small state machine controls the BUSY state and the full state of the FIFO form the write-side clock domain. When the arbiter grants the request, the FIFO state machine will enter the BUSY state. In such a case, or if the link FIFO is in the FULL state, the BUSY signal is asserted to prevent more writes. The BUSY signal is not deasserted until the FIFO is empty again. The deassertion of the BUSY signal will coincide with the state machine returning to the IDLE state. The buffer should hold one additional packet at most before it starts dropping while waiting for the buffer to free.

10. misc_rr_arbiter

The misc fifo round robin arbiter is used to arbitrate between all the links on a device to see who gets to send data to the single DPL buffer once control has been granted. It is a priority-shifting arbiter which multiplexes the data signals from all the inquiry link FIFOs.

10.1. Signals

Parameters:

- NUM_LINKS: # of links
- DATA_WIDTH: width of the input data bus

Input:

- iCLK: the PCIe clock
- iRST_N: the reset signal
- iGLOBAL_TIMESTAMP [55:0]: the current global timestamp
- iEND_OF_INTERVAL: interval heartbeat
- iMISC_ARB_DATA [NUM_LINKS-1:0][DATA_WIDTH-1:0]: input data from each FIFO
- iMISC_ARB_DATA_V [NUM_LINKS-1:0]: input data valid from each FIFO
- iMISC_ARB_REQ [NUM_LINKS-1:0]: request signal from each misc FIFO
- iMISC_ARB_FULL [NUM_LINKS-1:0]: full indicator
- iMISC_ARB_EMPTY [NUM_LINKS-1:0]: empty indicator
- iMISC_ARB_AEMPTY [NUM_LINKS-1:0]: almost empty indicator
- iINQ_PCIE_GRANT: grant signal from the PCIe DPL arbiter
- iMISC_FIFO_CTRL[3:0]: the Ctrl signal from the global register set (global.g.misc) – operates similar to MonitorMode within the local link, but is the global version for scsi_inq

Output:

- oMISC_ARB_REQ : arbiter request to DPL XFR
- oMISC_ARB_DATA [DATA_WIDTH-1:0]: muxed inquiry packet data to PCIe DPL arbiter

- oMISC_ARB_DATA_V: muxed inquiry packet data valid to PCIe DPL arbiter
- oMISC_ARB_FIFO_POP [NUM_LINKS-1:0]: arbiter grant signal to each link FIFO
- oMISC_ARB_DATA_CNT: pulse for misc pkt
- oMISC_ARB_INTV_CNT: pulse for interval heartbeat pkt
- oMISC_ARB_ZERO_CNT: pulse for zero-fill pkt

10.2. Logic

The round robin (based on the common one in bali_lib_pkg) arbiter grants incoming requests from the link FIFOs. The highest priority is checked first, before moving to the subsequent link FIFOs in a circular fashion. Every transfer changes the FIFO with highest priority for fairness.

A request can only be made by a link FIFO that has at least one complete packet stored in it. When it is granted, this puts the arbiter's FSM is a locked state to complete the transaction. During this locked state, the arbiter will mux and select the FIFO output from the desired misc FIFO to be transferred to the DPL buffer. A transfer count keep tracks of the number of bytes sent to the DPL in the current transaction. If a link FIFO is empty before the 4K transfer is empty, the arbiter will grant the request of the next highest priority link, until no link has remaining requests or the counter has detect a 4K transfer size. If the transfer size at the end of a transaction is less than 4K and no further requests are seen, then zeros will be streamed from arbiter to the DPL buffer to fulfill the 4K transfer requirement.

If an end of interval signal is received, then a short interval packet must be sent out. This interval packet will have no statistical data other than the timestamp. The interval stat packet will always be sent out at the end of the current transfer (if the transfer size is less than 4K), or at the beginning of the next transfer (if the current transfer size is > 4K). If the arbiter is idle with no requests, it will send only the interval stat packet (of size 512 bytes), followed by a zero fill.

Turning off the global misc fifo ctrl bits in the global misc csr registers will cause the arbiter to shut off and not send out interval packets as well. A mode for interval heartbeat only is available as well.

11. Design Alternatives

This sections contain alternate routes investigated during design and is obsolete (included only for reference purposes). All FPGA estimates in this section are baselined from the single pending inquiry design. Some of these are outdated when compared to the requirements of the final design, but their principles remain.

11.1. Complete Lookups vs. Best Effort

The major consideration to take into account is whether to adopt a system that stores all the information required to match inquiry commands to responses versus a best effort approach. The former inherently supports n-pending inquiries and a guarantee that no collisions will occur, at the cost of system complexity and additional resources, as well as likely board layout changes. The best effort approach on the other hand is highly customizable to usage needs and much more optimized for utilization, but will not guarantee that n-number of inquiry commands will be stored in the system.

A best effort approach is affected by the hashing algorithm used, the type of IDs that commonly appear, the size of the encoded hash, the size of the unencoded data, and the maximum number of pending inquiry. For a given random data set and random hashing algorithm, the hash collision chance can be calculated as:

$$1 - e^{-\frac{k(k-1)}{2N}}$$

Where k=number of inputs, N=number of possible hash values.

Based on Figure 4, we can see that for a 14-bit address, we have a 10% chance of a collision at 50 pending (unresolved) SCSI inquiry commands. Note that the M20Ks required grow exponentially with larger address space.

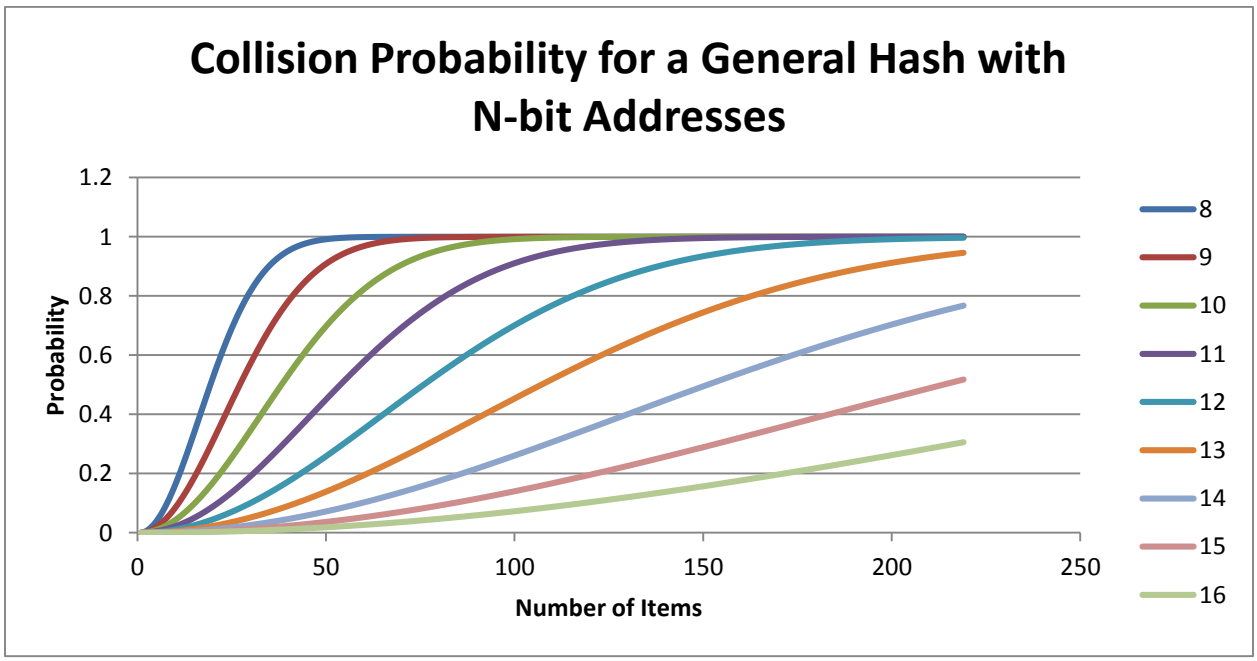

Figure 4 - Hash collision probability based on the hashed address size.

11.2. Content Addressable Memory

One solution for creating a robust lookup system would be to use content-addressable memory that is external to the FPGA and accessed by it. CAMs will allow us access the data directly by using the contents of the SCSI frame headers to attempt to match entries that have already been entered – however physical layout and latency issues come into play due to the nature of using external parts.

	ALUTs	Logic Reg	M20K
implementation	**165**	**218**	**64**
single_query	123	218	0

Figure 5 - Estimate of implementing CAM with current design (base estimate)

It is possible to adopt a similar CAM implementation to the one proposed in the Fiji project which uses TCAM (we can use the same part they use, but the size is completely dependent on the desired depth of the module).

At a minimum, the CAM would require extra available pins from the FPGA:

- Clock
- Reset
- 64-bit key bus
- Write enable
- Data valid
- 64-bit data out bus
- Hit signal (match/no match)

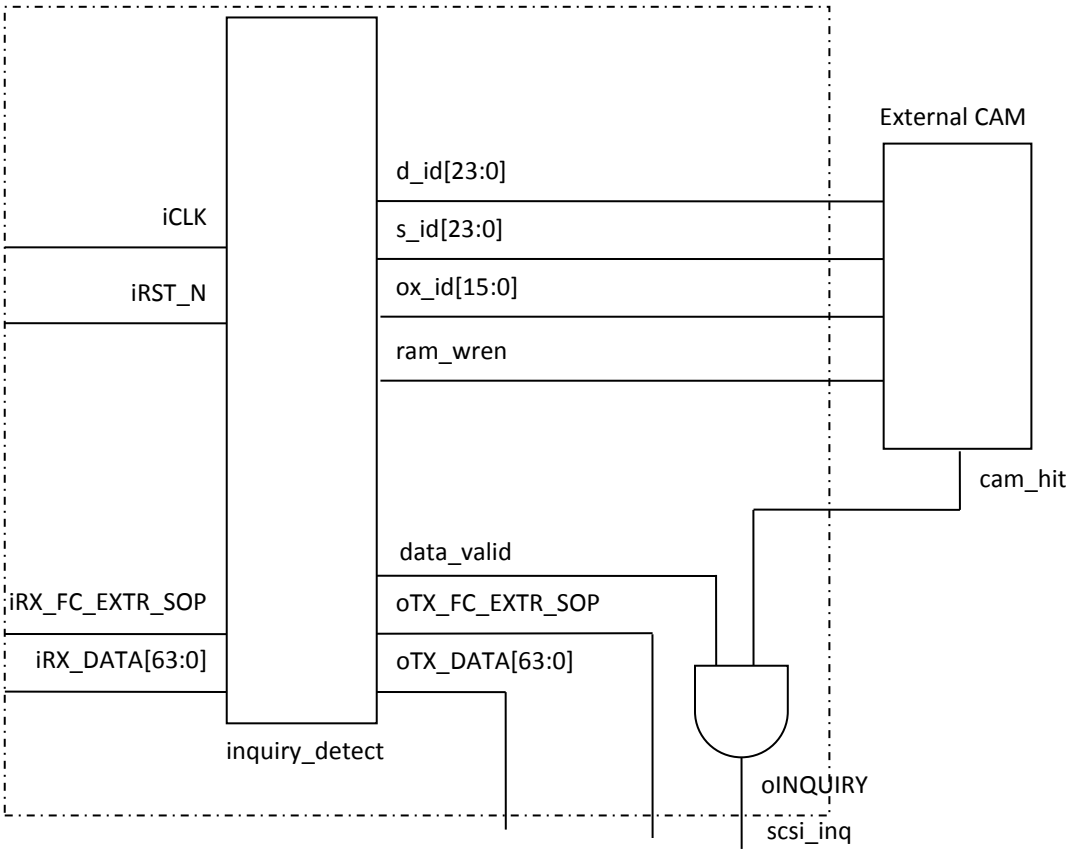

Figure 6 – Simple adoption of an external CAM into the current implementation

11.3. Hardware or DPL Check for Errors

If a best-effort approach is adopted, one question that arises is whether the DPL or the hardware itself should catch collision errors. In the standard case, there may be an occurrence where two set of IDs that occur hash to the same value. The frame that has nothing to do with SCSI inquiries would then be identified as a SCSI inquiry response and packaged as such. In order to rectify the situation, the hardware must filter the content by comparing the S_ID, D_ID, and OX_ID of the request and the response ahead of time, and correctly detect that this was a collision since the entries don't match. The alternative is to use the look-up table stored in the DPL to verify that the incoming SCSI responses match a known outgoing SCSI command from earlier. If it does it, the packet can simply be dropped.

	ALUTs	Logic Reg	M20K
implementation	**165**	**218**	**64**
no_hw_check	135	218	64

Figure 7 - Estimate of moving the SCSI inquiry response check to the DPL

11.4. Single Query vs N-Query

Single query is a very large simplification that allows only the last SCSI inquiry command to be stored in memory. The system will only be able to detect a matching SCSI inquiry response if it matches the IDs of the last command that went out. Expanding this, you could store the last n number of responses on the FPGA itself, but the logic requirement grows exponentially and favours external memory solutions after a small amount stored.

	ALUTs	Logic Reg	M20K
implementation	**165**	**218**	**64**
single_query	139	282	0

Figure 8 - Estimate of implementing single query mode

11.5. Hashing Algorithm

The choice of a hashing algorithm affects the robustness of the system by minimizing collisions. If the address space of the customer's data center does not change or is not reassigned regularly, existing collisions will always occur and may mask certain data from ever being shown in certain sequence of commands. A simpler algorithm (such as the XOR and XNOR system implemented above) requires fewer resources and has no extra latency, but does not do a very good job on sparse address spaces, such as one with a single switch. The D_ID and S_ID is a 24-bit ID that is assigned as follows:

23:16	15:8	7:0
Area	Domain	Port

Factors to consider:

- Latency (low priority – but must remain static).
- Fmax
- Utilization (logic use mostly)

11.6. Multiple Hashing Algorithms

One simple stopgap measure to stop frequent collisions on small datasets is to implement multiple hashing algorithms which can be toggled by the user to attempt to reduce the collision count for their particular setup. There are two possible ways to approach this:

- Have a toggle for the user to select **one** hashing algorithm to use
- Use **all** the hashing algorithms simultaneously

Using all the hashing algorithms gives the best coverage against collisions, but also requires dramatically more area and makes timing closure harder due to the extra logic generated from the data comparisons (assuming the hardware is guaranteeing error-free packets to the DPL). The toggle would require some playing around with settings before an optimal solution is discovered, and does not increase overall collision protection of a system.

	ALUTs	Logic Reg	M20K
implementation	**165**	**218**	**64**
hash_toggle	140 + 33*num_hash	218 + 64*extra_latency	64*num_hash
simultaneous_hash	140 + 33*num_hash	218 + 64*extra_latency	64*num_hash

Figure 9 - Estimate of implementing multiple hashes in hardware

11.7. Template Parsing per Vendor

After examining sample data from inquiry commands sent over SANBlaze with multiple vendor profiles, it has been determined that there are too many possible variations of responses from vendors due to the use of non-mandatory vendor-specific fields and variable lengths of some of these response fields and lists.

Possible candidates for vendor-specific information were as follows:

- Standard response, vendor-specific version descriptor: Describes the standards adhered to the device, but not implemented by some vendors – fixed response length if implemented (8 standard codes always with 0-fill)
- Serial number, 0x80 page from VPD: Optional to implement and variable length vendor-specific serial number
- Descriptor list, 0x83 page from VPD: Variable length list with both pre-formatted return blocks combined with vendor-specific return blocks. Does not suggest fixed return precedence so standardized blocks may not appear in the same place for each vendor.

11.8. Extended Extractor

It is possible to consider an alternative where the command set of the extractor could be increased to support type-length-value style variable length fields. However this idea was not pursued for the current chasis type (Bali, Emerald, Dominica) due to logic usage constraints.