15.4.2.2 Frame errors

Frame errors generally are handled in one of two ways depending on whether the error is expected to be transient or persistent and whether system state has been perturbed. For error conditions expected to be transient (such as a CRC error), and for which the system state has not been perturbed, the Transport layer may retry the corresponding transaction any number of times until ultimately a host timeout and software reset, or other error recovery attempt is made. For error conditions that are not a result of a transient error condition (such as an invalid Type field in a received FIS), the error response is generally to fail the transaction and report the failure.

The following are specific frame error scenarios and their corresponding Transport layer error control actions.

If the Transport layer receives a FIS with an invalid CRC signaled from the Link layer, the Transport layer shall signal the Link layer to negatively acknowledge frame reception by asserting R_ERR_P during the frame acknowledgement handshake.

The transmitter of a negatively acknowledged frame may retry the FIS transmission provided the system state has not been perturbed. Frame types that can be retransmitted are:

Register – Host to Device
Register – Device to Host
DMA Activate – Device to Host
DMA Setup – Device to Host
PIO Setup – Device to Host
Set Device Bits – Device to Host
BIST Activate – Host to Device or Device to Host

Because data transmission FISes result in a change in the host bus adapter's internal state, either through the DMA controller changing its state or through a change in the remaining PIO repetition count, data transmission FISes should never be retried.

The Transport layer is not required to retry those failed FIS transmissions that do not change system state, but the Transport layer may attempt retry any number of times. For conditions that are not addressed through retries, such as persistent errors, host software should eventually time out the transaction and reset the interface.

If the Transport layer detects reception of a FIS with unrecognized TYPE value, the Transport layer shall signal the Link layer to negatively acknowledge the frame reception by asserting R_ERR_P during the frame acknowledgement handshake.

If the Transport layer detects reception of a malformed frame, such as a frame with incorrect length, the Transport layer shall signal the Link layer to negatively acknowledge the frame reception by transmitting R_ERR_P during the frame acknowledgement handshake.

15.4.2.3 Protocol and state transition errors

Protocol and state errors stem from ill-behaved devices not following the Serial ATA protocol. Such errors are generally handled by failing the corresponding transactions and returning to a known state. Since such errors are not caused by an environmental transient, no attempt to retry such failed operations should be made. The following are specific frame error scenarios and their corresponding Transport layer error control actions.

If the PIO transfer count expires, and two Dwords later is not EOF_P (the CRC falls between the last data Dword and EOF_P), the transfer count stipulated in the PIO Setup FIS did not match the

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199

size of the subsequent data payload. For this data-payload/transfer-count mismatch, the Transport layer shall signal the Link layer to negatively acknowledge frame reception by transmitting R_ERR_P during the frame acknowledgement handshake.

15.4.3 Error reporting

The Transport layer reports errors to host software via the Serial ATA Status and Control registers. Devices communicate Transport layer error information to host software via transmitting a Register FIS to update the ATA Shadow Register Block Status and Error register values.

All Transport layer error conditions that are not handled/recovered by the Transport layer shall set the error bit in the Shadow Register Block Status register, and update the value in the Error register through transmission of an appropriate Register FIS.

Host Transport layer error conditions shall result in the status and error values in the SStatus and SError registers being updated with values corresponding to the error condition and shall result in the Link layer being notified to negatively acknowledge the offending FIS during the frame acknowledgement handshake.

15.5 Application layer error handling overview

The Application layer error handling is in part defined by behavior of software written for Parallel ATA. Superset error reporting capabilities are supported by the Transport layer through the Status and Control registers, and software can take advantage of those error reporting capabilities to improve error handling for Serial ATA.

15.5.1 Error detection

There are three overall error detection mechanisms by which software identifies and responds to Serial ATA errors

- Bad status in the Command Block Status register
- Bad status in the SError register
- Command failed to complete (timeout)

Conditions that return bad status in the Command Block Status register, but for which no Serial ATA interface error information is available, correspond to the error conditions specified in the ATA standard. Such error conditions and responses are defined in the ATA standard and there is no unique handling of those in Serial ATA. Errors in this category include command errors, such as attempts to read from an LBA past the end of the disk, as well as device-specific failures such as data not readable from the given sector number. These failures are not related to the Serial ATA interface, and thus no Serial ATA specific interface status information is available for these error conditions. Only the status information returned by the device is available for identifying the source of the problem, plus any available SMART data that might apply.

Transport layer error conditions, whether recovered or not, are reflected in the SStatus and SError registers as defined in section 14.1.2. The host Transport layer is responsible for reflecting error information in the SStatus and SError registers, while the device Transport layer is responsible for reflecting unrecovered errors in the Shadow Register Block Status and Error registers through transmission of appropriate Register FISes.

Commands that fail to complete are detected by host driver software through a timeout mechanism. Generally such timeouts result in no status or error information for the command being conveyed to host software and software may not be able to determine the source or cause of such errors.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 501 of 587

15.5.2 Error control actions

Conditions that return bad status in the Shadow Register Block Status register but for which no interface error information in the SError register is available shall be handled as defined in the ATA standard.

Conditions that return interface error information in the SError register are handled through four basic responses:

- Freeze
- Abort/Fail
- Retry (possible after reset)
- Track/ignore

Error conditions that result in catastrophic system perturbation that is not recoverable should result in the system halting. Serial ATA does not define any explicit halting conditions, however, software may be able to infer such conditions.

Error conditions that are not expected to be transient and which are not expected to succeed with subsequent attempts should result in the affected command being aborted and failed. Failure of such commands should be reported to higher software layers for handling. Scenarios in which this response is appropriate include attempts to communicate with a device that is not attached, and failure of the interface to successfully negotiate communications with an attached device.

Error conditions that are expected to be transient should result in the affected command being retried. Such commands may either be retried directly or may be retried after an interface and/or device reset, depending on the particular error value reported in the SError register. Scenarios in which this response is appropriate include noise events resulting in CRC errors, 8b/10b code violations, or disparity errors.

Conditions that are recoverable and for which no explicit error handling is required may be tracked or ignored. Tracking such errors allows subsequent fault isolation for marginal components and accommodates possible recovery operations. Scenarios in which this response is appropriate include tracking the number of Phy synchronization losses in order to identify a potential cable fault or to accommodate an explicit reduction in the negotiated communications rate.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 502 of 587

16 Port Multiplier

16.1 Introduction

A Port Multiplier is a mechanism for one active host connection to communicate with multiple devices. A Port Multiplier can be thought of as a simple multiplexer where one active host connection is multiplexed to multiple device connections, as shown in Figure 209.

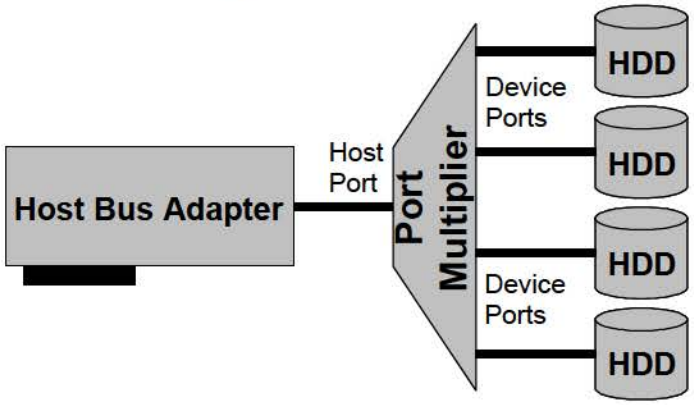


Figure 209 – Port Multiplier Overview

Only one active host connection to the Port Multiplier is supported. The Port Multiplier is an extensible design that supports up to 15 endpoint device connections and utilizes the full bandwidth of the host connection. A Port Multiplier shall not be connected to another Port Multiplier (i.e. no cascading).

16.2 Overview

The Port Multiplier uses four bits, known as the PM Port field, in all FIS types to route FISes between the host and the appropriate device. Using the PM Port field, the Port Multiplier can route FISes to up to 15 Serial ATA devices from one active host. The PM Port field is filled in by the host on a host-to-device FIS with the port address of the device to route the FIS to. For a device-to-host FIS, the PM Port field is filled in by the Port Multiplier with the port address of the device that is transmitting the FIS. Device port addresses start at zero and are numbered sequentially higher until the last device port address has been defined. The control port, port address Fh, is used for control and status communication with the Port Multiplier itself.

In order to utilize all devices connected to a Port Multiplier, the host needs to have a mechanism to set the PM Port field in all transmitted FISes.

The Port Multiplier maintains a set of general purpose registers and also maintains the Serial ATA superset Status and Control registers for each device port. The control port supports two commands, READ PORT MULTIPLIER and WRITE PORT MULTIPLIER, that are used to read and write these registers.

Some additional Port Multiplier features include:

- Supports booting with legacy software on device port 0h
- Supports staggered spin-up
- Supports hot plug

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199

16.3 Definition

16.3.1 Addressing Mechanism

The Port Multiplier uses four bits, known as the PM Port field, in all FIS types to route FISes between the host and the appropriate device. Using the PM Port field, the Port Multiplier can route FISes to up to 15 Serial ATA devices from one active host. The PM Port field is filled in by the host on a host-to-device FIS with the port address of the device to route the FIS to. For a device-to-host FIS, the PM Port field is filled in by the Port Multiplier with the port address of the device that is transmitting the FIS.

The PM Port field directly follows the FIS Type field in all FISes, refer to section 10.3.

16.3.2 Device Port Requirements

A device port is a port that can be used to connect a device to the Port Multiplier. The Port Multiplier may support up to 15 device ports. The device port addresses shall start at zero and shall be numbered sequentially until all device ports have port addresses. A valid device port shall have a port address that has a value less than the total number of device ports supported by the Port Multiplier.

16.3.2.1 Transmission from Host to Device

To transmit a FIS to a device connected to a Port Multiplier, the host shall set the PM Port field in the FIS to the device's port address. Then the host shall start transmitting the FIS to the Port Multiplier in accordance with the Transport, Link, and Phy state machines.

When a Port Multiplier receives a FIS over the host port, the Port Multiplier shall check the PM Port field in the FIS to determine the port address that the FIS should be transmitted over. If the FIS is destined for the control port, the Port Multiplier shall receive the FIS and perform the command or operation requested. If the FIS is destined for a device port, the Port Multiplier shall perform the following procedure:

1. The Port Multiplier shall determine if the device port is valid. If the device port is not valid, the Port Multiplier shall issue a SYNC Escape to the host and terminate reception of the FIS. Refer to section 16.3.3.8.3.

2. The Port Multiplier shall determine if the X bit in the DIAG field of the device port's PSCR[1] (SError) register is set to one. If set to one, the Port Multiplier shall issue a SYNC Escape to the host and terminate reception of the FIS. Refer to section 16.3.3.8.2.

3. The Port Multiplier shall determine if a collision has occurred. A collision is when a reception is already in progress from the device that the host wants to transmit to. If a collision has occurred, the Port Multiplier shall finish receiving the FIS from the host and shall issue an R_ERR_P primitive to the host as the ending status. The Port Multiplier shall follow the procedures outlined in section 16.3.3.2 to clear the collision condition.

4. The Port Multiplier shall initiate the transfer with the device by issuing an X_RDY_P primitive to the device. A collision may occur as the Port Multiplier is issuing the X_RDY_P to the device if the device has just decided to start a transmission to the host. If the device starts transmitting X_RDY_P to the Port Multiplier, a collision has occurred. If a collision has occurred, the Port Multiplier shall finish receiving the FIS from the host and shall issue an R_ERR_P primitive to the host as the ending status. The Port Multiplier shall follow the procedures outlined in section 16.3.3.2 to clear the collision condition.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 504 of 587

5. After the device issues R_RDY$_P$ to the Port Multiplier, the Port Multiplier shall transmit the FIS from the host to the device. The Port Multiplier shall not send R_OK$_P$ status to the host until the device has issued an R_OK$_P$ for the FIS reception. The R_OK$_P$ status handshake shall be interlocked from the device to the host. Refer to section 16.3.3.1.

If an error is detected during any part of the FIS transfer, the Port Multiplier shall ensure that the error condition is propagated to the host and the device.

The transfer between the host and Port Multiplier is handled separately from the transfer between the Port Multiplier and device; only the end of frame R_OK$_P$ handshake is interlocked. The Port Multiplier shall ensure that the flow control signaling latency requirement specified in section 9.4.7 is met for all FIS transfers on a per link basis. Specifically, the Port Multiplier shall ensure that the flow control signaling latency is met between:

1. The host port and the host it is connected to
2. Each device port and the device that it is connected to

If there is not an error detected during the FIS transfer, the Port Multiplier shall not alter the FIS transmitted to the device. The Port Multiplier is not required to check or recalculate the CRC.

16.3.2.2 Transmission from Device to Host

To transmit a FIS to the host, the device shall proceed with the transmission in accordance with the Transport, Link, and Phy state machines. The device behavior is the same whether it is connected directly to the host or is connected to the host via a Port Multiplier.

When a device wants to transmit a FIS to the host, the Port Multiplier shall perform the following procedure:

1. After receiving an X_RDY$_P$ primitive from the device, the Port Multiplier shall determine if the X bit is set in the DIAG field of the device port's PSCR[1] (SError) register. The Port Multiplier shall not issue an R_RDY$_P$ primitive to the device until the X bit in the DIAG field is cleared to zero.

2. The Port Multiplier shall receive the FIS from the device. The Port Multiplier shall fill in the PM Port field with the port address of the transmitting device. The Port Multiplier shall transmit the modified FIS to the host with a recalculated CRC. The Port Multiplier shall check the CRC received from the device. If the CRC from the device is invalid the Port Multiplier shall corrupt the CRC sent to the host to ensure that the error condition is propagated. Refer to section 16.3.3.8.4 for specific details on how the Port Multiplier corrupts the CRC in this error case.

3. The Port Multiplier shall issue an X_RDY$_P$ primitive to the host to start the transmission of the FIS to the host. After the host issues R_RDY$_P$ to the Port Multiplier, the Port Multiplier shall transmit the FIS from the device to the host. The Port Multiplier shall not send R_OK$_P$ status to the device until the host has issued an R_OK$_P$ for the FIS reception. The R_OK$_P$ status handshake shall be interlocked from the device to the host. Refer to section 16.3.3.1.

If an error is detected during any part of the FIS transfer, the Port Multiplier shall ensure that the error condition is propagated to the host and the device.

The Port Multiplier may wait for an X_RDY$_P$/R_RDY$_P$ handshake with the host prior to issuing an R_RDY$_P$ to the device to minimize buffering. The transfer between the device and Port Multiplier is handled separately from the transfer between the Port Multiplier and the host; only the end of frame R_OK$_P$ handshake is interlocked. The Port Multiplier shall ensure that the flow control signaling latency requirement specified in section 9.4.7 is met for all FIS transfers on a per link

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 505 of 587

basis. Specifically, the Port Multiplier shall ensure that the flow control signaling latency is met between:

1. The host port and the host it is connected to
2. Each device port and the device that it is connected to

16.3.3 Policies

16.3.3.1 FIS Delivery

The end of frame handshake shall be interlocked between the host and the device. Specifically, the Port Multiplier shall not issue an R_OK_P to the initiator of a FIS before the target of a FIS has issued an R_OK_P. The Port Multiplier shall propagate R_OK_P and R_ERR_P from the target of the FIS to the initiator of a FIS.

If a transmission fails before the R_OK_P handshake is delivered to the initiator, the Port Multiplier is responsible for propagating the error condition. Specifically, the Port Multiplier shall propagate $SYNC_P$ primitives received during a FIS transmission end-to-end to ensure that any error condition encountered in the middle of a FIS is propagated. Reference sections 16.3.3.8.4 and 16.3.3.8.5 on the appropriate actions to take when CRC calculation errors or possible data corruption occurs in a FIS transmission.

If there is a FIS transfer ongoing and the link between the Port Multiplier and the active device becomes inoperable, the Port Multiplier should issue $SYNC_P$ primitives to the host until the host responds with a $SYNC_P$ primitive in order to fail the transfer. Failing the transfer upon detecting an inoperable link allows the host to proceed with recovery actions immediately, thereby eliminating latency associated with a timeout.

16.3.3.1.1 Port Priority

The Port Multiplier shall ensure that an enabled and active device port is not starved. The specific priority algorithm used is implementation specific.

The control port shall have priority over all device transfers. While a command is outstanding to the control port, no device transmissions shall be started by the Port Multiplier until the command outstanding to the control port is completed.

16.3.3.1.2 FIS Delivery Mechanisms (Informative)

This section provides an informative reference for one method that a Port Multiplier may use to satisfy the FIS Delivery policies outlined.

16.3.3.1.2.1 Starting a FIS Transmission

If a device on a Port Multiplier asserts X_RDY_P and the Port Multiplier has selected that device for transmission next and the host port is not busy, the Port Multiplier shall:

1. Issue X_RDY_P to the host.
2. Wait for the host to respond with R_RDY_P.
3. After the host issues R_RDY_P, the Port Multiplier issues R_RDY_P to the device.

Then the transmission to the host may proceed.

If the host asserts X_RDY_P to the Port Multiplier and the Port Multiplier does not have X_RDY_P asserted to the host, the Port Multiplier responds with R_RDY_P to the host. The Port Multiplier receives the first Dword of the FIS payload from the host. If the Port Multiplier Port specified is a device port that is enabled on the Port Multiplier, the Port Multiplier issues an X_RDY_P over the device port specified and proceeds to transmit the entire FIS to the device. If a collision occurs during this process, the Port Multiplier shall follow the procedures outlined in section 16.3.3.2.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 506 of 587

16.3.3.1.2.2 Status Propagation

If there is an on-going FIS transmission between the host and a device, the Port Multiplier only issues R_OK_P, R_ERR_P and $SYNC_P$ if it has first received that primitive from the host or device, unless a collision occurs or an invalid port is specified.

The Port Multiplier shall not convey R_OK_P or R_ERR_P to the initiator of a FIS until the target of the FIS has issued R_OK_P or R_ERR_P once the end-to-end transmission has commenced.

If the initiator or target of a FIS transmission issues a $SYNC_P$ primitive during a FIS transfer, this primitive shall be propagated in order to ensure that the error condition is propagated to either end.

16.3.3.2 Collisions

A collision is when the Port Multiplier has already started a reception from the device that the host wants to transmit to. A collision also occurs when the device issues an X_RDY_P primitive at the same time that the Port Multiplier is issuing an X_RDY_P primitive to that device. All collisions are treated as an X_RDY_P/X_RDY_P collision; in accordance with the Link layer state machine, the host loses all such collisions and should retransmit its FIS at a later time.

A collision only occurs when the host is trying to issue another native queued command to a device that has native queued commands outstanding. The Native Command Queuing protocol guarantees that the host is never transmitting a Data FIS when the collision occurs. This means that the Port Multiplier can safely issue an error to the host and that the host should retry the failed FIS transmission at a later point in time.

When the Port Multiplier detects a collision, the Port Multiplier shall finish reception of the FIS from the host and shall issue an R_ERR_P to the host for the end of frame handshake. The Port Multiplier shall discard the FIS received from the host. The host should attempt to retry the FIS transfer that failed.

PMC1: PmColl_Idle	Perform normal operation. Wait for FIS reception from host.		
	1. Reception started from host for device port X	→	PmColl_ChkRecv
	2. Reception not started from host for device port X	→	PmColl_Idle

PMC2: PmColl_ChkRecv	Determine if a FIS reception is in progress on device port X.		
	1. Reception from device port X in progress	→	PmColl_Collision
	2. Reception from device port X not in progress	→	PmColl_XmitToDP

PMC3: PmColl_Collision	Collision is detected. Finish reception of the FIS from the host and discard the FIS contents.		
	1. $WTRM_P$ or $SYNC_P$ not received from host	→	PmColl_Collision
	2. $WTRM_P$ received from host	→	PmColl_FailHostFIS
	3. $SYNC_P$ received from host	→	PmColl_Idle

PMC4: PmColl_FailHostFIS	Transmit R_ERR_P to the host.		
	1. Unconditional	→	PmColl_Idle

PMC5: PmColl_XmitToDP	Transmit X_RDY_P to the device port X.	
1. X_RDY_P or R_RDY_P not received from device port X	→	PmColl_XmitToDP
2. X_RDY_P received from device port X	→	PmColl_Collision
3. R_RDY_P received from device port X	→	PmColl_Idle

X_RDY_P/X_RDY_P collisions that occur on the host port shall be handled in accordance with the Link layer state diagrams in section 9.6.

16.3.3.3 Booting with Software that is not Port Multiplier Aware

Booting is accommodated off of the first port of the Port Multiplier, device port 0h, without any special software or hardware support. An HBA that does not support attachment of a Port Multiplier shall work with the device on device port 0h.

If a system requires fast boot capability it should ensure that both the BIOS and OS driver software are Port Multiplier aware. If software that is not Port Multiplier aware is used in the presence of a Port Multiplier and there is no device present on device port 0h, that software detects a device as present but never receives a Register FIS with the device signature. Waiting for the device signature may cause a BIOS that is not Port Multiplier aware to not meet fast boot timing requirements.

16.3.3.4 Staggered Spin-up Support

The Port Multiplier shall disable all device ports on power-up or upon receiving a COMRESET signal over the host port. This feature allows the host to control when each device spins up. Refer to section 13.13.4.2 on how to enumerate a device, including devices that may be spun down because the port is disabled.

16.3.3.5 Hot Plug Events

Port Multiplier handling of hot plug events is defined by the hot plug state machines for the host port and device port, in sections 16.3.3.5.1 and section 16.3.3.5.2 respectively.

Upon receiving a COMRESET signal from the host, the Port Multiplier shall perform an internal reset. As part of the internal reset, the Port Multiplier shall update the Serial ATA superset Status and Control registers for each device port. A more detailed description of COMRESET handling is in section 13.13.2.1.

Upon receiving a COMINIT signal from a device, the Port Multiplier shall update the Serial ATA superset Status and Control registers for that port as defined in section 14. The Port Multiplier shall set the X bit in the DIAG field of the device port's PSCR[1] (SError) register to mark that device presence has changed as defined in section 14.1.2. If the Port Multiplier has not received a FIS for the control port after the last COMRESET and an unsolicited COMINIT signal was received over device port 0h, the Port Multiplier shall propagate the COMINIT signal to the host as specified in the hot plug state machine for the host port in section 16.3.3.5.1. For all other cases, the COMINIT signal shall not be propagated to the host.

If the X bit in the DIAG field of the PSCR[1] (SError) register is set for a device port, the Port Multiplier shall disallow FIS transfers with that port until the X bit has been cleared. When the Port Multiplier is disallowing FIS transfers with a device port, the Port Multiplier shall ensure FISes the device is attempting to transmit are not dropped.

It is recommended that host software frequently query GSCR[32], defined in section 16.4.1.2, to determine if there has been a device presence change on a device port.

16.3.3.5.1 Hot Plug State Machine for Host Port

Cabled hot plug of the Port Multiplier host port shall be supported since Port Multipliers may not share the same power supply as the host. Therefore the Port Multiplier shall periodically poll for

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 508 of 587

host presence by sending periodic COMINIT signals to the host after transitioning to the HPHP1:NoComm state.

The state machine enables device port 0h after communication has been established over the host port and also clears the X bit in the DIAG field of the PSCR[1] (SError) register for device port 0h after it is set when operating with software that is not Port Multiplier aware. In addition, the state machine shall propagate unsolicited COMINIT signals received from device port 0h to the host when no control port register accesses have yet occurred. These accommodations allow device port 0h to work with host software that is not Port Multiplier aware.

When operating with Port Multiplier aware software, the state machine treats device port 0h exactly like every other device port.

HPHP1: NoComm[1]	Set Port Multiplier to initial state as specified in section 13.13.1 and section 13.13.2.1	
1. Unconditional	→	StartComm
NOTE: 1. This state is entered upon power-on reset or in response to a received COMRESET		

HPHP2: StartComm	Transition Phy state machine to state DP1: DR_RESET. Communications retry interval reset to initial value.[1]	
1. Unconditional	→	WaitComm
NOTE: 1. The communications retry interval shall be greater than or equal to 10 ms.		

HPHP3: WaitComm		
1. PHYRDY asserted	→	EnablePort0
2. PHYRDY not asserted and communications retry interval not expired	→	WaitComm
3. PHYRDY not asserted and communications retry interval expired	→	NoComm

HPHP4: EnablePort0	Enable device port 0h if device port 0h is currently disabled	
1. PHYRDY asserted	→	Port0ComInit
2. PHYRDY not asserted	→	NoComm

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 509 of 587

HPHP4b: Port0ComInit		
1. Communications lost and interface not in power management state (i.e. unplug) and FIS not received for control port	→	NoComm
2. FIS received for control port	→	CommOK[1]
3. Communications established and FIS not received for control port and (COMINIT received from device port 0h or >= 10ms elapsed since entry into HPHP4b:Port0ComInit state)	→	Port0ComInitWait
4. Communications established and FIS not received for control port and COMINIT not received from device port 0h and < 10 ms elapsed since entry into HPHP4b:Port0ComInit state	→	Port0ComInit
NOTE: 1. If bit 0 of the DET field in PSCR[0] (SStatus) for Port 0 is set to one then the X bit in the DIAG field of PSCR[1] (SError) for Port 0 shall be set prior to making the transition		

HPHP4c: Port0ComInitWait		
1. Communications lost and interface not in power management state (i.e. unplug) and FIS not received for control port	→	NoComm
2. FIS received for control port	→	CommOK[1]
3. Communications established and FIS not received for control port and COMINIT de-asserted from device port 0h	→	LegacyCommOK[2]
4. Communications established and FIS not received for control port and COMINIT asserted from device port 0h	→	Port0ComInitWait
NOTE: 1. If bit 0 of the DET field in PSCR[0] (SStatus) for Port 0 is set to one then the X bit in the DIAG field of PSCR[1] (SError) for Port 0 shall be set prior to making the transition 2. The X bit in PSCR[1] (SError) for device port 0h shall be cleared to zero prior to making the transition.		

HPHP5: LegacyCommOK		
1. Communications lost and interface not in power management state (i.e. unplug) and FIS not received for control port	→	NoComm
2. Communications established or interface in power management state and COMINIT not received on device port 0h and FIS not received for control port	→	LegacyCommOK
3. Communications established or interface in power management state and COMINIT received on device port 0h and FIS not received for control port	→	StartComm[1]
4. FIS received for control port	→	CommOK[2]
NOTE: 1. The X bit in PSCR[1] (SError) for Port 0 shall be set to one prior to making the transition 2. If bit 0 of the DET field in PSCR[0] (SStatus) for Port 0 is set to one then the X bit in the DIAG field of PSCR[1] (SError) for Port 0 shall be set to one prior to making the transition		

HPHP6: CommOK		
1. Communications lost and interface not in power management state (i.e. unplug)	→	NoComm
2. Communications established or interface in power management state	→	CommOK

16.3.3.5.2 Hot Plug State Machine for Device Port

The device port hot plug behavior is exactly the same as the behavior for a host controller supporting device hot plug directly. There is no change to the device and there is no change to the usage model.

DPHP1: NoComm		
1. PHYRDY asserted	→	CommOK
2. PHYRDY not asserted	→	NoComm

DPHP2: CommOK		
1. Communications lost and interface not in power management state	→	NoComm
2. Communications established or interface in power management state	→	CommOK

16.3.3.6 Link Power Management

The Port Multiplier shall support reception of $PMREQ_P$ primitives from the host and from attached devices, in accordance with the Link layer state machine. If the Port Multiplier does not support the power management state requested, the Port Multiplier shall respond to the $PMREQ_P$ primitive with a $PMNAK_P$ primitive.

Before the Port Multiplier delivers a FIS, the Port Multiplier shall check the state of the link and issue a COMWAKE signal if the link is in Partial or Slumber state. The Port Multiplier shall accurately reflect the current state of each device port link in the port specific registers (specifically PSCR[0] (SStatus) for each port) defined in section 16.4.2.

The Port Multiplier shall not propagate a $PMREQ_P$ primitive received on a device port. A $PMREQ_P$ primitive received from a device only affects the link between that device and the Port Multiplier. If the Port Multiplier receives a $PMREQ_P$ primitive over a device port, the Port Multiplier shall perform the following actions:

- The Port Multiplier shall respond to the device with $PMACK_P$ or $PMNAK_P$
- If the Port Multiplier responds to the device with $PMACK_P$:
  - The Port Multiplier shall transition the link with that device to the power state specified
  - The Port Multiplier shall update the port specific registers for that device port to reflect the current power management state of that link

The Port Multiplier shall propagate a $PMREQ_P$ primitive received from the host to all active device ports if the Port Multiplier responds to the $PMREQ_P$ primitive with $PMACK_P$. In this case, the Port Multiplier shall propagate the $PMREQ_P$ primitive to all device ports that have PHYRDY asserted. If a device responds to a $PMREQ_P$ primitive with $PMNAK_P$, this event shall only affect the link with that device. The host may interrogate the Port Multiplier port specific registers to determine which device ports are in a power managed state.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 511 of 587

If the Port Multiplier receives a $PMREQ_P$ primitive from the host, the specific actions the Port Multiplier shall take are:

- The Port Multiplier shall respond to the host with $PMACK_P$ or $PMNAK_P$
- If the Port Multiplier responds to the host with $PMACK_P$:
  - The Port Multiplier shall transition the link with the host to the power state specified
  - The Port Multiplier shall issue $PMREQ_P$ primitives to all device ports that have PHYRDY asserted
    - If a device responds with $PMACK_P$, the Port Multiplier shall transition the link with that device to the power state specified and shall update the Port Multiplier port specific registers for that port
    - If a device responds with $PMNAK_P$, the Port Multiplier shall not take any action with that device port

The Port Multiplier shall wake device links on an as-needed basis. A COMWAKE signal from the host is not propagated to the devices. The Port Multiplier shall issue a COMWAKE signal to a device when a FIS needs to be delivered to that device.

The Port Multiplier may issue a $PMREQ_P$ to the host if all device ports are in the Partial state, Slumber state, or are disabled.

16.3.3.7 Reducing Context Switching Complexity

It may complicate some host controller designs if traffic from another device is received in the middle of certain FIS sequences. For example, if a host receives a DMA Activate FIS from one device, it may be awkward for the host to receive a FIS from another device before it is able to issue the Data FIS to the first device.

The Port Multiplier shall provide the host with the opportunity to transmit before initiating any pending device transmissions. The Port Multiplier shall not assert an X_RDY_P primitive to the host until the Port Multiplier has received at least two consecutive $SYNC_P$ primitives from the host. In the case of a collision, the Port Multiplier shall ignore this requirement and perform the actions detailed in section 16.3.3.2.

If the host wants to transmit prior to receiving another FIS, the host should issue one $SYNC_P$ primitive between the end of the last FIS transmission and the start of the next FIS transmission. One possible implementation is to transition to the host Link layer state HL_SendChkRdy defined in section 9.6.3 immediately, regardless of whether the host is ready to proceed with the next transmission. The host would then only transition out of HL_SendChkRdy state when the host is ready to proceed with the transmission and the R_RDY_P primitive is received.

16.3.3.8 Error Handling and Recovery

The host is responsible for handling error conditions in the same way it handles errors when connected directly to a device. The host is responsible for detecting commands that do not finish and performing error recovery procedures as needed. The exact host software error recovery procedures are implementation specific.

The Port Multiplier is not responsible for performing any error recovery procedures. The Port Multiplier shall return R_ERR_P for certain error conditions as described by the Link layer state machine. The Port Multiplier shall update GSCR[32], defined in section 16.4.1.2, and PSCR[1] (SError) for the device port that experiences an error.

16.3.3.8.1 Command Timeout

If a command times out, the host may check PSCR[1] (SError), defined in section 16.4.2, to determine if there has been an interface error condition on the port that had the error. If there has been a device presence change on that port as indicated by the X bit in the DIAG field of

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 512 of 587

PSCR[1] (SError) for the port, the host should re-enumerate the device on that port. The host software error recovery mechanism after a command timeout is implementation specific.

16.3.3.8.2 Disabled Device Port

If the host transmits a FIS to a device port that is disabled or that has FIS transfers disallowed due to the X bit being set in the DIAG field of PSCR[1] (SError) for that port, the Port Multiplier shall not perform an R_OK_P or R_ERR_P handshake at the end of FIS reception and shall instead terminate the FIS reception by issuing SYNC Escape to the host. The host is responsible for detecting that the command did not finish and performing error recovery procedures, including clearing the X bit in the DIAG field of the PSCR[1] register as necessary.

16.3.3.8.3 Invalid Device Port Address

An invalid device port address is a device port address that has a value greater than or equal to the number of device ports that the Port Multiplier supports. If the host specifies an invalid device port address as part of a FIS transmission, the Port Multiplier shall issue a SYNC Escape and terminate reception of the FIS. The host is responsible for detecting that the command did not finish and performing normal error recovery procedures.

16.3.3.8.4 Invalid CRC for Device Initiated Transfer

On a device initiated transfer, the Port Multiplier shall recalculate the CRC since it modifies the first Dword of the FIS. The Port Multiplier shall check the original CRC sent by the device. If the original CRC is invalid, the Port Multiplier shall invert the recalculated CRC to ensure that the CRC error is propagated to the host. The inversion can be done by XORing the recalculated CRC with FFFFFFFFh. The Port Multiplier shall also update the error information in PSCR[1] (SError) for the device port that experienced the error.

16.3.3.8.5 Data corruption

If the Port Multiplier encounters an 8b/10b decoding error or any other error that could affect the integrity of the data passed between the transmitter and receiver of a FIS, the Port Multiplier shall ensure that the error is propagated to the receiver. The Port Multiplier may propagate the error by corrupting the CRC for the FIS. The Port Multiplier shall also update the error information in PSCR[1] (SError) for the device port that experienced the error.

16.3.3.8.6 Unsupported Command Received on control port

If an unsupported command is received on the control port, the Port Multiplier shall respond with a Register FIS that has the values shown in Figure 210 for the Status and Error registers.

Register	7	6	5	4	3	2	1	0
Error	Reserved					ABRT	Reserved	
Status	BSY	DRDY	DF	na	DRQ	0	0	ERR

Figure 210 – Register values for an unsupported command

ABRT = 1
BSY = 0
DRDY = 1
DF = 0
DRQ = 0
ERR = 1

16.3.3.9 BIST Support

A Port Multiplier may optionally support BIST. A Port Multiplier that supports BIST shall only support BIST in a point-to-point manner. A Port Multiplier that supports BIST shall not propagate a BIST Activate FIS received on one port over another port. The host determines that a Port Multiplier supports BIST by checking GCSR[64], defined in section 16.4.1.3.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199

To enter BIST mode over the host connection with a Port Multiplier, the host shall issue a BIST Activate FIS to the Port Multiplier control port. The host shall not issue a BIST Activate FIS to a device port.

To enter BIST mode over a device connection, the device shall issue a BIST Activate FIS to the Port Multiplier. The Port Multiplier shall intercept the BIST Activate FIS and enter BIST mode. Upon entering BIST mode with the device, the Port Multiplier shall update the PSCR[0] (SStatus) register for that port to reflect that the link has entered BIST mode, as specified in section 14.1.1.

Initiation of BIST by a Port Multiplier is vendor unique.

16.3.3.10 Asynchronous Notification

A Port Multiplier may optionally support asynchronous notification as defined in section 13.6. If asynchronous notification is enabled, a Port Multiplier shall send an asynchronous notification to the host when a bit transitions from zero to one in GSCR[32] of the Global Status and Control Registers. The Port Multiplier may send one notification for multiple zero to one bit transitions. The asynchronous notification shall only be sent when there is no command currently outstanding to the control port. If there is a command outstanding to the control port when an asynchronous notification needs to be sent, the Port Multiplier shall first complete the command and then send the asynchronous notification. The Port Multiplier shall set the PM Port field in the Set Device Bits FIS to the control port to indicate that the Port Multiplier itself needs attention.

Support for the asynchronous notification feature is indicated in GSCR[64] and it is enabled using GSCR[96].

16.3.3.10.1 Command-based switching (Informative)

Host designs should give careful consideration to support of asynchronous notification in command-based switching designs (outlined in section 16.7.1). If a command-based switching HBA has no explicit accommodation for asynchronous notification, then the host should not enable asynchronous notification on the control port or on any attached device.

16.3.3.11 Phy Event Counters

A Port Multiplier may optionally support the Phy event counters feature that is defined in section 13.7. If Phy event counters is enabled, a Port Multiplier shall store supported counter information for all of the ports that are enabled, including the host port. It is not required that the same list of Phy event counters be implemented on every port.

Support for the Phy event counters feature is indicated in GSCR[64] and it is enabled using bit 0 in GSCR[34]. The counter values shall not be retained across power cycles. The counter values shall be preserved across COMRESET and software resets that occur on any port.

16.3.3.11.1 Counter Identifiers

A Port Multiplier may support any of the counter identifiers described in Figure 181 in section 13.7.2. Support for some counters may not be logical for all ports due to the Port Multiplier architecture; examples of this would be the counters with identifiers 001h and 00Ah. For the Port Multiplier implementation, all counters other than 000h are optional. A counter identifier value of 000h shall indicate the end of the Phy event counter list implemented in the GSCR or PSCR registers. The counter with identifier 000h shall have no counter value.

All counter values consume a multiple of 16-bits, with a maximum of 64-bits. Each counter is allocated a single register location. The register location contains both the identifier for the counter implemented along with the value of the Phy event counter.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 514 of 587

16.3.3.11.2 Reading Counter Values

Initially, the host may obtain the mapping of counters implemented along with their sizes by submitting reads starting at GSCR[256] (or PSCR[256]) to obtain the identifiers for the counters on a per port basis. Once the identifier of 000h is reached, this signifies the end of the list of counters implemented. The format of this read is a READ PORT MULTIPLIER command with the RS1 bit of the Device register cleared to zero. The value in the PortNum field determines which set of counters is to be accessed. Device port counter information for ports 0h-Eh can be retrieved by using port numbers 0h-Eh respectively. Host port counter information can be retrieved by using the control port number (Fh). RegNum[15:0] shall be set to the specific register location to be read. The output of the READ PORT MULTIPLIER command contains the identifier of the counter. When Phy event counters are enabled, a Port Multiplier shall return identifier 000h as the first counter for a port on which no Phy event counters are implemented.

To read the counter value itself, the READ PORT MULTIPLIER command is sent with the RS1 bit of the Device register set to one. The value in the PortNum field determines which set of counters are to be accessed. Device port counter information for ports 0h-Eh can be retrieved by using port numbers 0h-Eh respectively. Host port counter information can be retrieved by using the control port number (Fh). RegNum[15:0] shall be set to the specific register location to be read. The output of the READ PORT MULTIPLIER command contains the value of the counter, up to 64-bits in length. If the counter value read is less than 64-bits in length, the value returned by the Port Multiplier shall have the upper bits padded with zeroes.

All counters are one-extended up to a 16-bit multiple (i.e. 16, 32, 48, 64-bit), once the maximum counter value has been reached. For example, when returned as a 16-bit counter, if a counter is only physically implemented as 8-bits when it reaches the maximum value of FFh, it shall be one-extended to FFFFh. The counter shall stop (and not wrap to zero) after reaching its maximum value.

Upon any read to Phy event counter register space that is at or beyond the identifier 000h location, the Port Multiplier shall return error status with the ERR bit set to one, and the BSY and DRQ bits cleared to zero in the Status field of the FIS. The ABRT bit shall also be set to one in the Error field.

16.3.3.11.3 Counter Reset Mechanisms

There are three mechanisms the host can use to explicitly cause the Phy event counters to be reset. The first mechanism uses the WRITE PORT MULTIPLIER command to clear counters on an individual counter basis. The PortNum field determines which set of counters are to be accessed. Device port counter information for ports 0h-Eh can be written by using port numbers 0h-Eh respectively. Host port counter information can be written by using the control port number (Fh). The RegNum field shall be set to the register location (counter) which is to be written. The Value field within the command contains what shall be written into the register, in this case all zeroes.

The second mechanism allows for a counter to be reset following a read to that counter register. If the RS2 bit in the Device register is set to one for a read to any counter, the Port Multiplier shall return the current counter value for the command and then reset that value upon successful transmission of the Register FIS. If retries are required, upon unsuccessful transmission of the FIS it is possible that the counter value may be changed before the re-transmission of the counter value.

The third mechanism is a global reset function by writing appropriate bits in GSCR[34] to reset all Phy event counters for specific ports. A host may reset all Phy event counters by writing FFFFh to bits 31-16 of GSCR[34].

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 515 of 587

16.4 Port Multiplier Registers

The Port Multiplier registers are accessed using READ PORT MULTIPLIER and WRITE PORT MULTIPLIER commands issued to the control port.

16.4.1 General Status and Control Registers

The control port address is specified in the PortNum field of the READ PORT MULTIPLIER and WRITE PORT MULTIPLIER commands defined in section 16.5.1 and section 16.5.2 in order to read/write the General Status and Control registers.

16.4.1.1 Static Configuration Information

The Static Configuration Information section of the General Status and Control Registers contains registers that are static throughout the operation of the Port Multiplier. These registers are read-only and may not be modified by the host.

Register	O/M	F/V	Description
GSCR[0]	M		Product Identifier
		F	31-16 Device ID allocated by the vendor
		F	15-0 Vendor ID allocated by the PCI-SIG
GSCR[1]	M		Revision Information
		F	31-16 Reserved
		F	15-8 Revision level of the Port Multiplier
		F	7-4 Reserved
		F	3 1 = Supports Port Multiplier specification 1.2
		F	2 1 = Supports Port Multiplier specification 1.1
		F	1 1 = Supports Port Multiplier specification 1.0
		F	0 Reserved
GSCR[2]	M		Port Information
		F	31-4 Reserved
		F	3-0 Number of exposed device fan-out ports
GSCR[3] – GSCR[31]	O	F	Reserved

Key:
O/M = Mandatory/optional requirement.
M = Support of the register is mandatory.
O = Support of the register is optional.
F/V = Fixed/variable content
F = the content of the register is fixed and does not change.
V = the contents of the register is variable and may change.
X = the content of the register may be fixed or variable.

Figure 211 – Static Information Registers

Register 0: Product Identifier

Support for this register is mandatory. The register identifies the vendor that produced the Port Multiplier and the specific device identifier.

Bit 15-0 shall be set to the vendor identifier allocated by the PCI-SIG of the vendor that produced the Port Multiplier.

Bit 31-16 shall be set to a device identifier allocated by the vendor.

Register 1: Revision Information

Support for this register is mandatory. The register specifies the specification revision that the Port Multiplier supports; the Port Multiplier may support multiple specification revisions.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 516 of 587

The register also specifies the revision level of the specific Port Multiplier product identified by Register 0.

Bit 0 is reserved and shall be cleared to zero.

Bit 1 when set to one indicates that the Port Multiplier supports Port Multiplier specification version 1.0.

Bit 2 when set to one indicates that the Port Multiplier supports Port Multiplier specification version 1.1.

Bit 3 when set to one indicates that the Port Multiplier supports Port Multiplier specification version 1.2.

Bit 7-4 are reserved and shall be cleared to zero.

Bit 15-8 identifies the revision level of the Port Multiplier product identified by Register 0. This identifier is allocated by the vendor.

Bit 31-16 are reserved and shall be cleared to zero.

Register 2: Port Information

Support for this register is mandatory. The register specifies information about the ports that the Port Multiplier contains, including the number of exposed device fan-out ports. The number of exposed device ports is the number of device ports that are physically connected and available for use on the product.

Bit 3-0 specifies the number of exposed device fan-out ports. A value of zero is invalid. The control port shall not be counted.

Bit 31-4 are reserved and shall be cleared to zero.

Register 3-31: Reserved

Registers 3-31 are reserved for future Port Multiplier definition and shall be cleared to zero.

16.4.1.2 Status Information and Control

The Status Information section of the General Status and Control Registers contains registers that convey status information and control operation of the Port Multiplier.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 517 of 587

Register	O/M	F/V	Description
GSCR[32]	M		Error Information
		F	31-15 Reserved
		V	14 OR of selectable bits in Port 14 PSCR[1] (SError)
		V	13 OR of selectable bits in Port 13 PSCR[1] (SError)
		V	12 OR of selectable bits in Port 12 PSCR[1] (SError)
		V	11 OR of selectable bits in Port 11 PSCR[1] (SError)
		V	10 OR of selectable bits in Port 10 PSCR[1] (SError)
		V	9 OR of selectable bits in Port 9 PSCR[1] (SError)
		V	8 OR of selectable bits in Port 8 PSCR[1] (SError)
		V	7 OR of selectable bits in Port 7 PSCR[1] (SError)
		V	6 OR of selectable bits in Port 6 PSCR[1] (SError)
		V	5 OR of selectable bits in Port 5 PSCR[1] (SError)
		V	4 OR of selectable bits in Port 4 PSCR[1] (SError)
		V	3 OR of selectable bits in Port 3 PSCR[1] (SError)
		V	2 OR of selectable bits in Port 2 PSCR[1] (SError)
		V	1 OR of selectable bits in Port 1 PSCR[1] (SError)
		V	0 OR of selectable bits in Port 0 PSCR[1] (SError)
GSCR[33]	M		Error Information Bit Enable
		V	31-0 If set, bit is enabled for use in GSCR[32]
GSCR[34]	O	V	Phy Event Counter Control
			31 Host port global counter reset
			30 Port 14 global counter reset
			29 Port 13 global counter reset
			28 Port 12 global counter reset
			27 Port 11 global counter reset
			26 Port 10 global counter reset
			25 Port 9 global counter reset
			24 Port 8 global counter reset
			23 Port 7 global counter reset
			22 Port 6 global counter reset
			21 Port 5 global counter reset
			20 Port 4 global counter reset
			19 Port 3 global counter reset
			18 Port 2 global counter reset
			17 Port 1 global counter reset
			16 Port 0 global counter reset
			15-1 Reserved
			0 Phy event counters enabled
GSCR[35] – GSCR[63]	O	F	Reserved

Key:
O/M = Mandatory/optional requirement.
M = Support of the register is mandatory.
O = Support of the register is optional.
F/V = Fixed/variable content
F = the content of the register is fixed and does not change.
V = the contents of the register is variable and may change.
X = the content of the register may be fixed or variable.

Figure 212 – Status Information and Control Registers

Register 32: Error Information

Support for this register is mandatory. This register reflects whether specific bits have been set in any of the device port's PSCR[1] (SError) register. A set bit may reflect an error or

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 518 of 587

that device presence has changed. The host selects the device port's PSCR[1] (SError) bits to reflect using GSCR[33]. The Port Multiplier algorithm for updating the register contents is:

```
for (n=0; n<NumPorts; n++)
{
      if (Port[n].PSCR[1] & GSCR[33]) == 0)
            GSCR[32].Bit[n]=0
      else
            GSCR[32].Bit[n]=1
}
```

This register is read-only. The specific device port's PSCR[1] (SError) register shall be written in order to clear values in the affected PSCR[1] (SError) register and by reflection in this register. Refer to section 14.1.2 for the definition of the SError register.

Bit 0 shall be set to the logically OR-ed value of selected bits in Port 0 PSCR[1] (SError). The bits to be OR-ed are selected using GSCR[33].

Bit 1 shall be set to the logically OR-ed value of selected bits in Port 1 PSCR[1] (SError). The bits to be OR-ed are selected using GSCR[33]. If Port 1 is not implemented by the Port Multiplier, this bit shall be cleared to zero.

Bit 2 shall be set to the logically OR-ed value of selected bits in Port 2 PSCR[1] (SError). The bits to be OR-ed are selected using GSCR[33]. If Port 2 is not implemented by the Port Multiplier, this bit shall be cleared to zero.

Bit 3 shall be set to the logically OR-ed value of selected bits in Port 3 PSCR[1] (SError). The bits to be OR-ed are selected using GSCR[33]. If Port 3 is not implemented by the Port Multiplier, this bit shall be cleared to zero.

Bit 4 shall be set to the logically OR-ed value of selected bits in Port 4 PSCR[1] (SError). The bits to be OR-ed are selected using GSCR[33]. If Port 4 is not implemented by the Port Multiplier, this bit shall be cleared to zero.

Bit 5 shall be set to the logically OR-ed value of selected bits in Port 5 PSCR[1] (SError). The bits to be OR-ed are selected using GSCR[33]. If Port 5 is not implemented by the Port Multiplier, this bit shall be cleared to zero.

Bit 6 shall be set to the logically OR-ed value of selected bits in Port 6 PSCR[1] (SError). The bits to be OR-ed are selected using GSCR[33]. If Port 6 is not implemented by the Port Multiplier, this bit shall be cleared to zero.

Bit 7 shall be set to the logically OR-ed value of selected bits in Port 7 PSCR[1] (SError). The bits to be OR-ed are selected using GSCR[33]. If Port 7 is not implemented by the Port Multiplier, this bit shall be cleared to zero.

Bit 8 shall be set to the logically OR-ed value of selected bits in Port 8 PSCR[1] (SError). The bits to be OR-ed are selected using GSCR[33]. If Port 8 is not implemented by the Port Multiplier, this bit shall be cleared to zero.

Bit 9 shall be set to the logically OR-ed value of selected bits in Port 9 PSCR[1] (SError). The bits to be OR-ed are selected using GSCR[33]. If Port 9 is not implemented by the Port Multiplier, this bit shall be cleared to zero.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 519 of 587

Bit 10 shall be set to the logically OR-ed value of selected bits in Port 10 PSCR[1] (SError). The bits to be OR-ed are selected using GSCR[33]. If Port 10 is not implemented by the Port Multiplier, this bit shall be cleared to zero.

Bit 11 shall be set to the logically OR-ed value of selected bits in Port 11 PSCR[1] (SError). The bits to be OR-ed are selected using GSCR[33]. If Port 11 is not implemented by the Port Multiplier, this bit shall be cleared to zero.

Bit 12 shall be set to the logically OR-ed value of selected bits in Port 12 PSCR[1] (SError). The bits to be OR-ed are selected using GSCR[33]. If Port 12 is not implemented by the Port Multiplier, this bit shall be cleared to zero.

Bit 13 shall be set to the logically OR-ed value of selected bits in Port 13 PSCR[1] (SError). The bits to be OR-ed are selected using GSCR[33]. If Port 13 is not implemented by the Port Multiplier, this bit shall be cleared to zero.

Bit 14 shall be set to the logically OR-ed value of selected bits in Port 14 PSCR[1] (SError). The bits to be OR-ed are selected using GSCR[33]. If Port 14 is not implemented by the Port Multiplier, this bit shall be cleared to zero.

Bit 31-15 are reserved and shall be cleared to zero.

Register 33: Error Information Bit Enable

Support for this register is mandatory. This register selects/enables bits to be used for the OR operation in GSCR[32]. If a bit is set to one, that bit shall be reflected for each device port in GSCR[32]. This is a global enable and is not device port specific. The default value of this register shall be 0400FFFFh; that corresponds to all bits in the ERR field and the DIAG field's X bit being OR-ed together for each device port in GSCR[32].

Register 34: Phy Event Counter Control

Support for this register is mandatory if Phy event counters is supported by the Port Multiplier as indicated by bit 4 in GSCR[64]. This register is used for Phy event counter control mechanisms.

Bit 0 if set to one indicates that the Port Multiplier supports Phy event counters and that the feature is currently enabled. If the Port Multiplier supports Phy event counters, it shall support counters for all ports implemented on the Port Multiplier, including the host port. It is not required for the same list of Phy event counters be implemented on every port. If this bit is cleared to zero, then the current values within the counters shall be retained and counting shall stop. This bit shall be cleared to zero on power-up. This bit shall not be affected by a COMRESET or software reset that has occurred on any port. The values within the Phy event counters are not affected by this bit.

Bits 1-15 are reserved and shall be cleared to zero.

Bit 16 if set to one shall result in an immediate reset of all Phy event counters associated with Port 0. Once the reset of all counters is complete, this bit shall be cleared to zero automatically by the Port Multiplier. If written to zero, no action is taken.

Bit 17 if set to one shall result in an immediate reset of all Phy event counters associated with Port 1. Once the reset of all counters is complete, this bit shall be cleared to zero automatically by the Port Multiplier. If written to zero, no action is taken.

Bit 18 if set to one shall result in an immediate reset of all Phy event counters associated with Port 2. Once the reset of all counters is complete, this bit shall be cleared to zero automatically by the Port Multiplier. If written to zero, no action is taken.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 520 of 587

Bit 19 if set to one shall result in an immediate reset of all Phy event counters associated with Port 3. Once the reset of all counters is complete, this bit shall be cleared to zero automatically by the Port Multiplier. If written to zero, no action is taken.

Bit 20 if set to one shall result in an immediate reset of all Phy event counters associated with Port 4. Once the reset of all counters is complete, this bit shall be cleared to zero automatically by the Port Multiplier. If written to zero, no action is taken.

Bit 21 if set to one shall result in an immediate reset of all Phy event counters associated with Port 5. Once the reset of all counters is complete, this bit shall be cleared to zero automatically by the Port Multiplier. If written to zero, no action is taken.

Bit 22 if set to one shall result in an immediate reset of all Phy event counters associated with Port 6. Once the reset of all counters is complete, this bit shall be cleared to zero automatically by the Port Multiplier. If written to zero, no action is taken.

Bit 23 if set to one shall result in an immediate reset of all Phy event counters associated with Port 7. Once the reset of all counters is complete, this bit shall be cleared to zero automatically by the Port Multiplier. If written to zero, no action is taken.

Bit 24 if set to one shall result in an immediate reset of all Phy event counters associated with Port 8. Once the reset of all counters is complete, this bit shall be cleared to zero automatically by the Port Multiplier. If written to zero, no action is taken.

Bit 25 if set to one shall result in an immediate reset of all Phy event counters associated with Port 9. Once the reset of all counters is complete, this bit shall be cleared to zero automatically by the Port Multiplier. If written to zero, no action is taken.

Bit 26 if set to one shall result in an immediate reset of all Phy event counters associated with Port 10. Once the reset of all counters is complete, this bit shall be cleared to zero automatically by the Port Multiplier. If written to zero, no action is taken.

Bit 27 if set to one shall result in an immediate reset of all Phy event counters associated with Port 11. Once the reset of all counters is complete, this bit shall be cleared to zero automatically by the Port Multiplier. If written to zero, no action is taken.

Bit 28 if set to one shall result in an immediate reset of all Phy event counters associated with Port 12. Once the reset of all counters is complete, this bit shall be cleared to zero automatically by the Port Multiplier. If written to zero, no action is taken.

Bit 29 if set to one shall result in an immediate reset of all Phy event counters associated with Port 13. Once the reset of all counters is complete, this bit shall be cleared to zero automatically by the Port Multiplier. If written to zero, no action is taken.

Bit 30 if set to one shall result in an immediate reset of all Phy event counters associated with Port 14. Once the reset of all counters is complete, this bit shall be cleared to zero automatically by the Port Multiplier. If written to zero, no action is taken.

Bit 31 if set to one shall result in an immediate reset of all Phy event counters associated with the host port. Once the reset of all counters is complete, this bit shall be cleared to zero automatically by the Port Multiplier. If written to zero, no action is taken.

Register 35-63: Reserved

Registers 34-63 are reserved for future Port Multiplier definition and shall be cleared to zero.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 521 of 587

16.4.1.3 Features Supported

The Features Supported section of the General Status and Control Registers contains registers that convey the optional features that are supported by the Port Multiplier. All Features Supported registers are read-only.

Register	O/M	F/V	Description
GSCR[64]	M		Port Multiplier Revision 1.X Features Support
		F	31-5 Reserved
		F	4 1 = Supports Phy event counters
		F	3 1 = Supports asynchronous notification
		F	2 1 = Supports dynamic SSC transmit enable
		F	1 1 = Supports issuing PMREQ$_P$ to host
		F	0 1 = Supports BIST
GSCR[65] – GSCR[95]	O	F	Reserved

Key:
O/M = Mandatory/optional requirement.
M = Support of the register is mandatory.
O = Support of the register is optional.
F/V = Fixed/variable content
F = the content of the register is fixed and does not change.
V = the contents of the register is variable and may change.
X = the content of the register may be fixed or variable.

Figure 213 – Features Supported Registers

Register 64: Port Multiplier Revision 1.X Features Support

Support for this register is mandatory. This register specifies the optional Port Multiplier specification revision 1.X features that the Port Multiplier supports.

Bit 0 if set to one indicates that the Port Multiplier supports BIST as outlined in section 16.3.3.9. If this bit is cleared to zero the Port Multiplier does not support reception of the BIST Activate FIS.

Bit 1 if set to one indicates that the Port Multiplier supports issuing PMREQ$_P$ requests to the host when all device ports are disabled or in a Partial/Slumber state. If this bit is cleared to zero, the Port Multiplier does not support issuing PMREQ$_P$ requests to the host.

Bit 2 if set to one indicates that the Port Multiplier supports dynamically enabling and disabling spread-spectrum clocking transmit. If this bit is cleared to zero, the Port Multiplier does not support dynamically enabling and disabling spread-spectrum clocking transmit.

Bit 3 if set to one indicates that the Port Multiplier supports asynchronous notification. If the Port Multiplier supports asynchronous notification, it shall be capable of sending a Set Device Bits FIS to the host with the Interrupt 'I' bit set to one and the Notification 'N' bit set to one when a bit in GSCR[32] transitions from zero to one. If this bit is cleared to zero then the Port Multiplier does not support asynchronous notification.

Bit 4 if set to one indicates that the Port Multiplier supports Phy event counters, and GSCR[34] shall be implemented. If this bit is cleared to zero, the Port Multiplier does not support Phy event counters.

Bit 31-5 are reserved and shall be cleared to zero.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 522 of 587

Register 65-95: Reserved

Registers 65-95 are reserved for future Port Multiplier definition and shall be cleared to zero.

16.4.1.4 Features Enabled

The Features Enabled section of the General Status and Control Registers contains registers that allow optional features to be enabled. The Features Enabled registers have a one-to-one correspondence with the Features Supported registers defined in section 16.4.1.3. All Features Enabled registers are read/write.

Register	O/M	F/V	Description
GSCR[96]	M		Port Multiplier Revision 1.X Features Enable
		F	31-4 Reserved
		V	3 1 = Asynchronous notification enabled
		V	2 1 = Dynamic SSC transmit is enabled
		V	1 1 = Issuing PMREQ$_P$ to host is enabled
		V	0 1 = BIST support is enabled
GSCR[97] – GSCR[127]	O	F	Reserved

Key:
O/M = Mandatory/optional requirement.
M = Support of the register is mandatory.
O = Support of the register is optional.
F/V = Fixed/variable content
F = the content of the register is fixed and does not change.
V = the contents of the register is variable and may change.
X = the content of the register may be fixed or variable.

Figure 214 – Features Enabled Registers

Register 96: Port Multiplier Revision 1.X Features Enable

Support for this register is mandatory. This register controls whether the optional Port Multiplier specification revision 1.X features that the Port Multiplier supports are enabled.

Bit 0 if set to one indicates that the Port Multiplier supports BIST and that BIST support is enabled. If this bit is cleared to zero reception of the BIST Activate FIS is not enabled. If the Port Multiplier supports BIST, this bit shall be set to one on power-up or after a reset, otherwise it shall be cleared to zero on power-up and reset.

Bit 1 if set to one indicates that the Port Multiplier supports issuing PMREQ$_P$ requests to the host when all device ports are disabled or in a Partial/Slumber state and the feature is enabled. If this bit is cleared to zero, the Port Multiplier shall not issue PMREQ$_P$ requests to the host. This bit shall be cleared to zero on power-up and reset.

Bit 2 if set to one indicates that the Port Multiplier supports dynamically enabling and disabling spread-spectrum clocking transmit and that the feature is currently enabled. If this bit is cleared to zero and the feature is supported as specified in GSCR[64], the Port Multiplier shall not use spread-spectrum clocking transmit. If this feature is supported as specified in GSCR[64], the bit shall be set to one on power-up and reset. If this feature is not supported as specified in GSCR[64], then the bit shall be cleared to zero on power-up and reset.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 523 of 587

Bit 3 if set to one indicates that the Port Multiplier supports asynchronous notification and that the feature is currently enabled. If the Port Multiplier supports asynchronous notification, it shall be capable of sending a Set Device Bits FIS to the host with the Interrupt 'I' bit set to one and the Notification 'N' bit set to one when a bit in GSCR[32] transitions from zero to one. If this bit is cleared to zero and the corresponding bit in GSCR[64] is set to one, then asynchronous notification is not enabled. This bit shall be cleared to zero on power-up and reset.

Bit 31-4 are reserved and shall be cleared to zero.

Register 97-127: Reserved

Registers 97-127 are reserved for future Port Multiplier definition and shall be cleared to zero.

16.4.1.5 Vendor Unique

The Vendor Unique section of the General Status and Control Registers contains registers that are vendor unique.

Register	O/M	F/V	Description
GSCR[128] – GSCR[255]	O	X	Vendor Unique

Key:
O/M = Mandatory/optional requirement.
M = Support of the register is mandatory.
O = Support of the register is optional.
F/V = Fixed/variable content
F = the content of the register is fixed and does not change.
V = the contents of the register is variable and may change.
X = the content of the register may be fixed or variable.

Figure 215 – Vendor Unique Registers

Register 128-255: Vendor unique

16.4.1.6 Phy Event Counters

The Phy event counters section of the General Status and Control Registers contains registers that store the data for each of the Phy event counters supported by the Port Multiplier for the host port. The Phy event counters registers contain both the identifier and counter values. All Phy event counter registers are read/write.

A value of zero returned for a counter means that there have been no instances of that particular event.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 524 of 587

Register	O/M	F/V	Description
GSCR[256] – GSCR[2303]	O	V	Phy event counter registers

Key:
O/M = Mandatory/optional requirement.
M = Support of the register is mandatory.
O = Support of the register is optional.
F/V = Fixed/variable content
F = the content of the register is fixed and does not change.
V = the contents of the register is variable and may change.
X = the content of the register may be fixed or variable.

Register 256-2303: Used for Phy event counters. Phy event counters are not allocated specific registers. Each register contains both the identifier and value for the counter implemented.

16.4.1.7 Reserved

The section of the General Status and Control Registers starting at address 2304 are reserved.

Register	O/M	F/V	Description
GSCR[2304] – GSCR[65535]	O	F	Reserved

Key:
O/M = Mandatory/optional requirement.
M = Support of the register is mandatory.
O = Support of the register is optional.
F/V = Fixed/variable content
F = the content of the register is fixed and does not change.
V = the contents of the register is variable and may change.
X = the content of the register may be fixed or variable.

Register 2304-65535: Reserved for future Port Multiplier definition and shall be cleared to zero.

16.4.2 Port Status and Control Registers

The Port Multiplier shall maintain a set of Port Status and Control registers (PSCRs) for each device port that it supports. These registers are port specific and contain the Serial ATA superset Status and Control registers, along with the Phy event counters information for each port. The host specifies the device port to read or write registers for in the PortNum field of the READ PORT MULTIPLIER command or WRITE PORT MULTIPLIER command defined in section 16.5.1 and section 16.5.2. The registers are defined in the following table.

PSCR[0]	SStatus register (see section 14.1.1)
PSCR[1]	SError register (see section 14.1.2)
PSCR[2]	SControl register (see section 14.1.3)
PSCR[3]	SActive register (not implemented)
PSCR[4]	SNotification register (not implemented)
PSCR[5] - PSCR[255]	Reserved
PSCR[256] – PSCR[2303]	Phy event counter registers
PSCR[2304] – PSCR[65535]	Reserved

Figure 216 – PSCR definition

The reset value for the DET field of PSCR[2] (SControl) register shall be 4h (Phy disabled).

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 525 of 587

16.4.2.1 Phy Event Counters

The Phy event counter information for each of the device ports within a Port Multiplier is contained in the Port Status and Control registers starting at PSCR[256]. The Phy event counters registers contain both the identifier and counter values. All Phy event counter registers are read/write.

A value of zero returned for a counter means that there have been no instances of that particular event.

16.5 Port Multiplier Command Definitions

16.5.1 READ PORT MULTIPLIER

The READ PORT MULTIPLIER command is used to read a register on a Port Multiplier. The READ PORT MULTIPLIER command shall be issued to the control port.

16.5.1.1 Inputs

Register	7	6	5	4	3	2	1	0
Features	RegNum[7:0]							
Features (exp)	RegNum[15:8]							
Sector Count	Reserved							
Sector Count (exp)	Reserved							
LBA Low	Reserved							
LBA Low (exp)	Reserved							
LBA Mid	Reserved							
LBA Mid (exp)	Reserved							
LBA High	Reserved							
LBA High (exp)	Reserved							
Device	na	RS1	RS2	na	PortNum			
Command	E4h							

Figure 217 – READ PORT MULTIPLIER command definition

PortNum — Set to the port address that has register to be read

RegNum — Set to number of register to read

RS1 — Register Specific 1: This bit is register specific.
Phy event counter usage: Used to determine access to Phy event counter identifier or value. If cleared to zero upon a read to a Phy event counter register, the Port Multiplier shall return the Phy event counter identifier for that register. If set to one, the Port Multiplier shall return the Phy event counter value for that register, up to 64-bits in length.
All other usage: This bit shall be treated as 'na' on all READ PORT MULTIPLIER commands to registers other than Phy event counter registers.

RS2 — Register Specific 2: This bit is register specific.
Phy event counter usage: Used in reads to Phy event counter values. If set to one upon a read to a Phy event counter register, the Port Multiplier shall return the value of the counter, followed by a reset of the counter value for the register supplied in RegNum.
All other usage: This bit shall be treated as 'na' on all READ PORT MULTIPLIER commands to registers other than Phy event counter registers.

na — Field or register is not used

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 526 of 587

16.5.1.2 Success Outputs

Upon successful completion, the ERR bit in the Status register shall be cleared and the value in the Error register shall be zero, and the value of the specified register shall be returned.

Register	7	6	5	4	3	2	1	0
Error	0							
Sector Count	Value [7:0]							
Sector Count (exp)	Value [39:32]							
LBA Low	Value [15:8]							
LBA Low (exp)	Value [47:40]							
LBA Mid	Value [23:16]							
LBA Mid (exp)	Value [55:48]							
LBA High	Value [31:24]							
LBA High (exp)	Value [63:56]							
Device	Reserved							
Status	BSY	DRDY	DF	na	DRQ	0	0	ERR

Figure 218 – READ PORT MULTIPLIER success status result values

Value Set to the 64-bit value read from the register
na Field or register is not used

BSY = 0
DRDY = 1
DF = 0
DRQ = 0
ERR = 0

16.5.1.3 Error Outputs

Upon encountering an error, the Port Multiplier shall set the ERR bit in the Status register and identify the error code in the Error register.

Register	7	6	5	4	3	2	1	0
Error	Reserved					ABRT	REG	PORT
Sector Count	Reserved							
Sector Count (exp)	Reserved							
LBA Low	Reserved							
LBA Low (exp)	Reserved							
LBA Mid	Reserved							
LBA Mid (exp)	Reserved							
LBA High	Reserved							
LBA High (exp)	Reserved							
Device	Reserved							
Status	BSY	DRDY	DF	na	DRQ	0	0	ERR

Figure 219 – READ PORT MULTIPLIER error status result values

ABRT = 0
REG Set to one if the register specified is invalid
PORT Set to one if the port specified is invalid
na Field or register is not used

BSY = 0
DRDY = 1
DF = 0

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 527 of 587

DRQ = 0
ERR = 1

16.5.2 WRITE PORT MULTIPLIER

The WRITE PORT MULTIPLIER command is used to write a register on a Port Multiplier. The WRITE PORT MULTIPLIER command shall be issued to the control port.

16.5.2.1 Inputs

Register	7	6	5	4	3	2	1	0
Features	RegNum[7:0]							
Features (exp)	RegNum[15:8]							
Sector Count	Value [7:0]							
Sector Count (exp)	Value [39:32]							
LBA Low	Value [15:8]							
LBA Low (exp)	Value [47:40]							
LBA Mid	Value [23:16]							
LBA Mid (exp)	Value [55:48]							
LBA High	Value [31:24]							
LBA High (exp)	Value [63:56]							
Device	na	RS1	na	na	PortNum			
Command	E8h							

Figure 220 – WRITE PORT MULTIPLIER command definition

Value	Set to the 64-bit value to write to the register
RegNum	Set to number of register to write
PortNum	Set to the port address that has register to be written
RS1	Register Specific 1: This bit is register specific. Phy event counter usage: The WRITE PORT MULTIPLIER command may not be used to write to a Phy event counters' identifier value. If this bit is set to one during a write to a Phy event counter register, the counter addressed by RegNum shall be written with Value. If cleared to zero during a write to a Phy event counter register, the Port Multiplier shall return error status with the ERR bit set to one, and the BSY and DRQ bits cleared to zero in the Status field of the FIS. The ABRT bit shall also be set to one in the Error field. All other usage: This bit shall be treated as 'na' on all READ PORT MULTIPLIER commands to registers other than Phy event counter registers.
na	Field or register is not used

16.5.2.2 Success Outputs

Upon successful completion, the Port Multiplier shall write the specified value to the specified register. The ERR bit in the Status register shall be cleared and the value in the Error register shall be zero.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 528 of 587

Register	7	6	5	4	3	2	1	0
Error	0							
Sector Count	Reserved							
Sector Count (exp)	Reserved							
LBA Low	Reserved							
LBA Low (exp)	Reserved							
LBA Mid	Reserved							
LBA Mid (exp)	Reserved							
LBA High	Reserved							
LBA High (exp)	Reserved							
Device	Reserved							
Status	BSY	DRDY	DF	na	DRQ	0	0	ERR

Figure 221 – WRITE PORT MULTIPLIER success status result values

na Field or register is not used

BSY = 0
DRDY = 1
DF = 0
DRQ = 0
ERR = 0

16.5.2.3 Error Outputs

Upon encountering an error, the Port Multiplier shall set the ERR bit in the Status register and identify the error code in the Error register.

Register	7	6	5	4	3	2	1	0
Error	Reserved					ABRT	REG	PORT
Sector Count	Reserved							
Sector Count (exp)	Reserved							
LBA Low	Reserved							
LBA Low (exp)	Reserved							
LBA Mid	Reserved							
LBA Mid (exp)	Reserved							
LBA High	Reserved							
LBA High (exp)	Reserved							
Device	Reserved							
Status	BSY	DRDY	DF	na	DRQ	0	0	ERR

Figure 222 – WRITE PORT MULTIPLIER error status result values

ABRT = 0
REG Set to one if the register specified is invalid
PORT Set to one if the port specified is invalid
na Field or register is not used

BSY = 0
DRDY = 1
DF = 0
DRQ = 0
ERR = 1

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 529 of 587

16.5.3 Interrupts

The Port Multiplier shall generate an interrupt in the status response to a command for the control port by setting the I bit in the Register FIS. The Port Multiplier shall not generate an interrupt in response to a software reset for the control port.

16.6 Controlling PM Port value and Interface Power Management

The host controller needs to provide a means by which software can set the PM Port field in all transmitted FISes. This capability may be exposed to software by supporting the PMP field in the SControl register defined in section 14.1.3. In addition a means by which software can cause a specific device port to transition to a low power management state needs to be provided. This capability may be exposed to software by supporting the SPM field in the SControl register defined in section 14.1.3.

16.7 Switching Types (Informative)

The host may use two different switching types depending on the capabilities of the host controller. If the host controller supports hardware context switching based on the value of the PM Port field in a received FIS, then the host may have commands outstanding to multiple devices at the same time. This switching type is called FIS-based switching and results in FISes being delivered to the host from any of the devices that have commands outstanding to them. If the host controller does not support hardware context switching based on the value of the PM Port field in a received FIS, then the host may only have commands outstanding to one device at any point in time. This switching type is called command-based switching and results in FISes being delivered to the host from only the one device that has commands outstanding to it. The Port Multiplier's operation is the same regardless of the switching type used by the host.

16.7.1 Command-based switching

Host controllers that do not support hardware context switching utilize a switching type called command-based switching. To use command-based switching, the host controller has commands outstanding to only one device at any point in time. By only issuing commands to one device at a time, the result is that the Port Multiplier only delivers FISes from that device.

A host controller may support command-based switching by implementing the Port Multiplier Port (PMP) field in the SControl register as detailed in section 14.1.3. In order to use this mechanism, host software would set the PMP field appropriately before issuing a command to a device connected to the Port Multiplier. When host software had completed the commands with a particular device port, it would modify the PMP field before issuing commands to any other device port. Note that the PMP field shall be set to the control port when host software issues commands to the Port Multiplier itself (such as READ PORT MULTIPLIER or WRITE PORT MULTIPLIER).

16.7.2 FIS-based switching

Host controllers that support hardware context switching may utilize a switching type called FIS-based switching. FIS-based switching allows the host controller to have commands outstanding to multiple devices at any point in time. When commands are outstanding to multiple devices, the Port Multiplier may deliver FISes from any device with commands outstanding.

16.7.2.1 Host Controller Requirements

To support FIS-based switching, the host controller shall have context switching support. The host controller needs to provide a means for exposing a programming interface for up to 16 devices on a single port. The context switching support needs to comprehend not only Control Block register context, but also DMA engine context and the SActive register. The host controller needs to be able to update context for a particular device even if the programming interface for that device is not currently selected by host software.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 530 of 587

The host controller needs to fill in the PM Port field in hardware assisted FIS transmissions. For example, if the host controller receives a DMA Activate from a device, it needs to construct a Data FIS with the PM Port field set to the value in the received DMA Activate FIS.

Refer to section 16.3.3.7 for a mechanism to reduce the complexity of host context switching.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 531 of 587

17 Port Selector

A Port Selector allows two different host ports to connect to the same device in order to create a redundant path to that device. In combination with RAID, the Port Selector allows system providers to build fully redundant solutions. The upstream ports of a Port Selector can also be attached to a Port Multiplier to provide redundancy in a more complex topology. A Port Selector can be thought of as a simple multiplexer as shown in Figure 223.

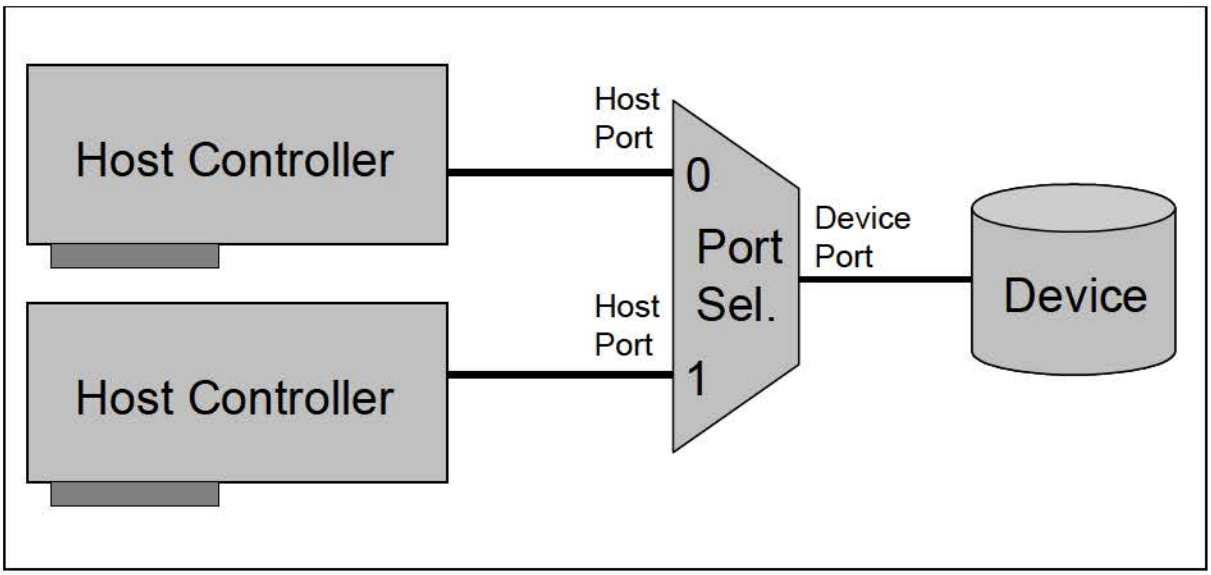


Figure 223 – Port Selector Overview

Exactly two host connections are provided by a Port Selector. Only one of the two host ports is active at a time – the specification does not define mechanisms for active/active solutions. Cascading Port Selectors to eachother is not supported.

17.1 Example Applications

One example application of a Port Selector, as shown in Figure 224, is to provide a means for redundant access to a device. This ingredient, along with RAID, allows a system with no single point of failure to be built. Typically the Port Selector would be packaged in the hard drive carrier to create a single serviceable unit in case the hard drive failed. The total system would consist of two hosts each connected to a RAID array where each drive in the system had a Port Selector attached that was connected to each host. One host could be considered the live host and the other host may be the spare. In this configuration, the live host would maintain access to all of the devices and the spare host would only take over access to the devices if the live host had a failure.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199

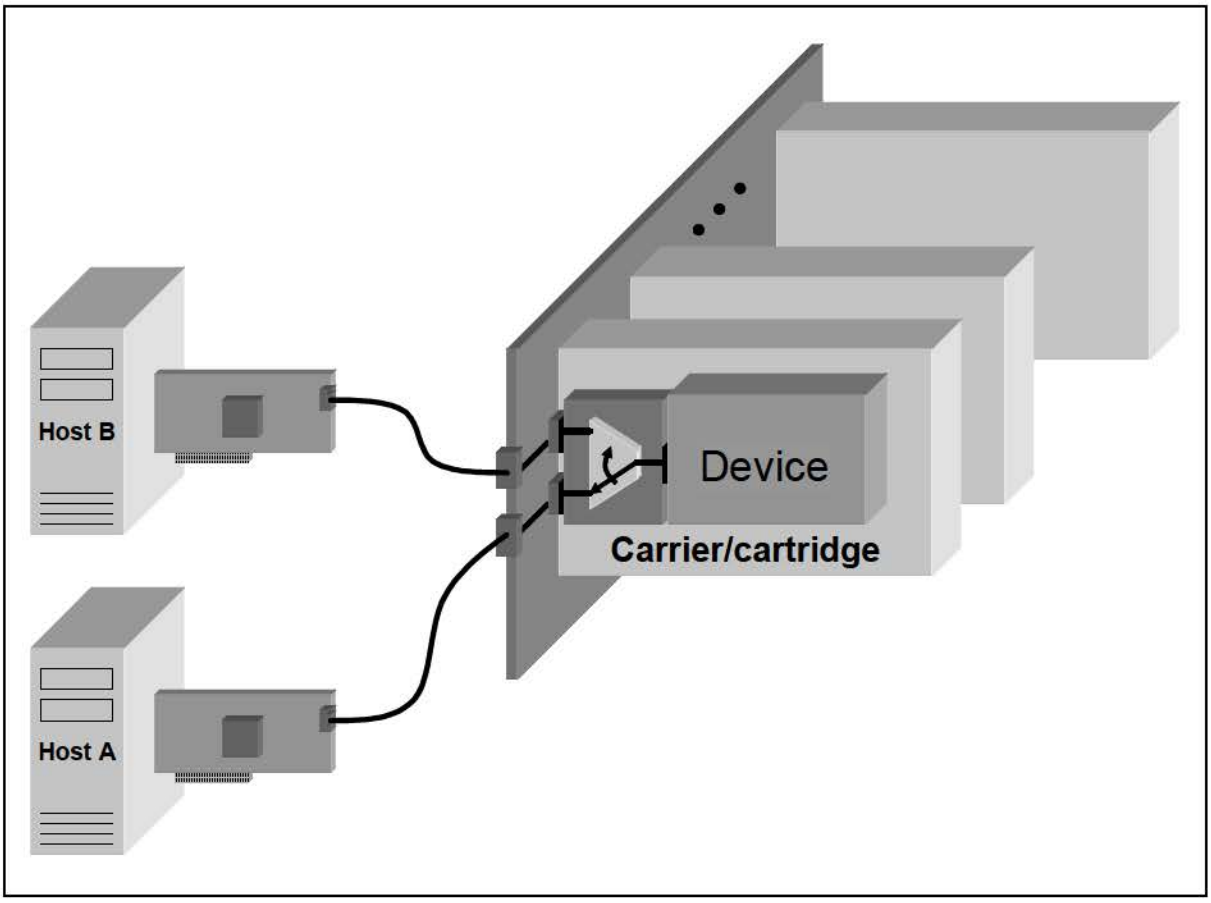


Figure 224 – Example Failover Application with Two Hosts

17.2 Overview

Port Selector is a mechanism that allows two different host ports to connect to the same device in order to create a redundant path to that device. Only one host connection to the device is active at a time. Effective use of a Port Selector requires coordinated access to the device between the two host ports. The host(s) shall coordinate to determine which host port should be in control of the device at any given point in time. Definition of the coordination mechanism or protocol is beyond the scope of this specification.

Once the host(s) determines the host port that should be in control of the device, the host that contains the host port to be made active takes control of the device by selecting that host port to be active. The active host selects a port to be active by using either a protocol-based or side-band port selection mechanism. A side-band port selection mechanism can be as simple as a hardware select line that is pulled high to activate one host port and low to activate the other. The side-band port selection mechanism is outside the scope of this specification. A protocol-based port selection mechanism uses the Serial ATA protocol to cause a switch of active port. This specification defines a protocol-based port selection mechanism that uses a particular Morse coding of COMRESET signals to cause a switch of active host port. A Port Selector shall only support one selection mechanism at any point in time. The externally visible behavior of a Port Selector is the same regardless of whether a protocol-based or side-band port selection mechanism is used.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 533 of 587

A Port Selector that supports protocol-based port selection can be detected in the signal path if the optional presence detection feature is supported by the Port Selector and the host has an enhanced SError register that can latch this event. The detection mechanism for a Port Selector that supports side-band port selection is outside the scope of this specification.

17.3 Active Port Selection

The Port Selector has a single active host port at a time. The Port Selector shall support one of two mechanisms for determining which of the two host ports is active. The first mechanism is called side-band port selection. Side-band port selection uses a mechanism outside of the Serial ATA protocol for determining which host port is active. The second mechanism is called protocol-based port selection. Protocol-based port selection uses a sequence of Serial ATA OOB Phy signals to select the active host port.

Whether a protocol-based or side-band port selection mechanism is used, the Port Selector shall exhibit the behavior defined within this specification.

After selection of a new active port, the device and is in an unknown state. The device may have active commands outstanding from the previous active host that need to be flushed. After an active port switch has been performed it is strongly suggested that the active host issue a COMRESET to the device to ensure that the device is in a known state.

17.3.1 Protocol-based Port Selection

Protocol-based port selection is an active port selection mechanism that uses a sequence of Serial ATA OOB Phy signals to select the active host port. A Port Selector that supports protocol-based port selection shall have no active host port selected upon power-up. The first COMRESET or COMWAKE received over a host port shall select that host port as active. The host may then issue explicit switch signals to change the active host port.

Reception of the protocol-based port selection signal on the inactive host port causes the Port Selector to deselect the currently active host port and select the host port over which the selection signal is received. The protocol-based port selection signal is defined such that it can be generated using the Status and Control registers and such that it can be received and decoded without the need for the Port Selector to include a full Link or Transport layer (i.e. direct Phy detection of the signal).

17.3.1.1 Port Selection Signal Definition

The port selection signal is based on a pattern of COMRESET OOB signals transmitted from the host to the Port Selector. As illustrated in Figure 225, the Port Selector shall qualify only the timing from the assertion of a COMRESET signal to the assertion of the next COMRESET signal in detecting the port selection signal.

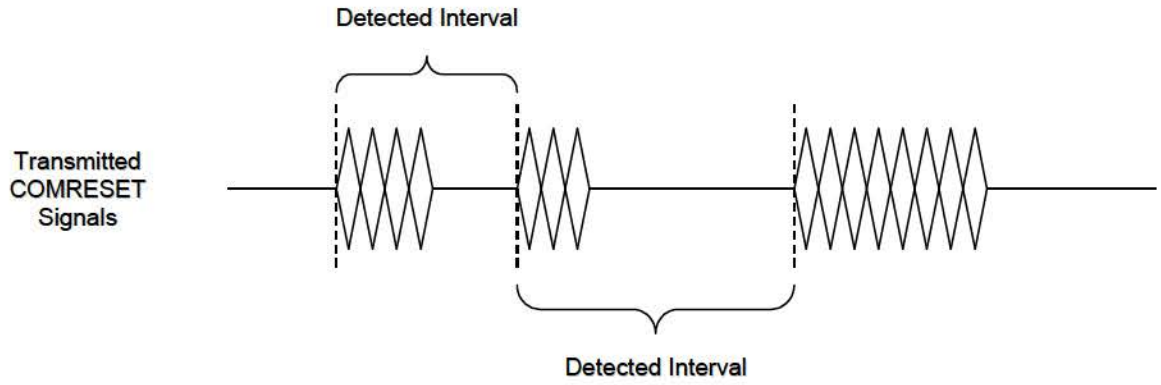


Figure 225 – Port selection signal based on assertion of COMRESET to assertion of following COMRESET

The port selection signal is defined as a series of COMRESET signals with the timing from the assertion of one COMRESET signal to the assertion of the next as defined in Table 44 and illustrated in Figure 226. The Port Selector shall select the port, if inactive, on the de-assertion of COMRESET after receiving two complete back-to-back sequences with specified inter-burst spacing over that port (i.e. two sequences of two COMRESET intervals comprising a total of five COMRESET bursts with four inter-burst delays). Specifically, after receiving a valid port selection signal, the Port Selector shall not select that port to be active until the entire fifth COMRESET burst has been de-asserted. The Port Selector is only required to recognize the port selection signal over an inactive port. Reception of COMRESET signals over an active port is propagated to the device without any action taken by the Port Selector, even if the COMRESET signals constitute a port selection signal. This may result in multiple device resets.

The timings detailed in Table 44 shall be independent of the signaling speed used on the link. For example, the inter-reset timings are the same for links using Gen1 or Gen2 speeds.

	Nom	Min	Max	Units	Comments
T1	2.0	1.6	2.4	ms	Inter-reset assertion delay for first event of the selection sequence
T2	8.0	7.6	8.4	ms	Inter-reset assertion delay for the second event of the selection sequence

Table 44 – Port selection signal inter-reset timing requirements

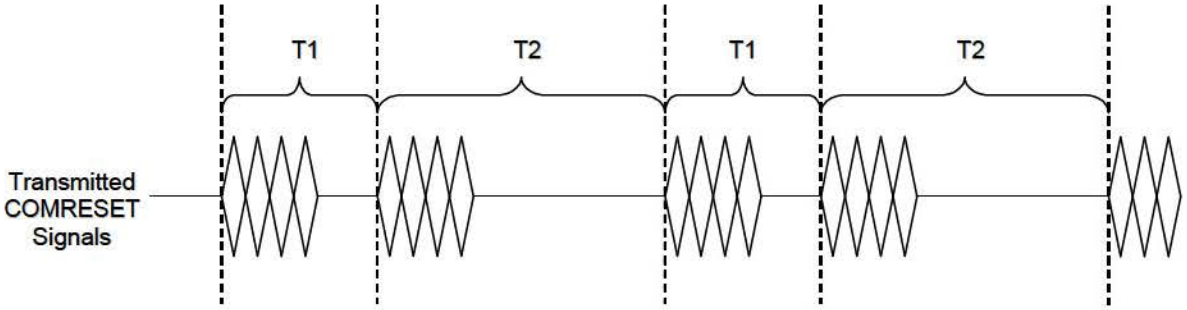


Figure 226 – Complete port selection signal consisting of two sequences with requisite inter-reset spacings

The interpretation and detection of the COMRESET signal by the Port Selector is in accordance with the Phy layer electrical specifications, i.e. the COMRESET signal is detected upon receipt of the fourth burst that complies with the COMRESET signal timing definition. The inter-reset timings referred to here for the port selection signal are from the detection of a valid COMRESET signal to the next detection of such a signal, and are not related to the bursts that comprise the COMRESET signal itself.

17.3.1.2 Presence Detection

Presence detection is the ability for a host to detect that a Port Selector is present on a port. If a Port Selector supports presence detection capabilities, a host will be able to determine whether a Port Selector is connected to a host port. The host may determine this regardless of whether the Port Selector port to which it is connected is the active or inactive link.

Presence detection capabilities are defined for Port Selectors utilizing protocol-based port selection only. Systems utilizing side-band port selection shall be preconfigured for side-band port selection and therefore those systems can also be preconfigured to support presence detection. Presence detection is an optional feature of protocol-based Port Selectors.

17.3.1.2.1 Host port Phy state machine enhancements

If presence detection is supported, the Port Selector host port Phy state machine as described in the section 8.3.2 shall be modified as shown. A Port Selector shall remain in the DR_PS_Wait state when the Phy is offline and presence detection is enabled.

DP1: DR_Reset[1]	Interface quiescent	
1. COMRESET not detected and power-on reset deasserted ***and presence detection not enabled and Phy not offline***	→	DR_COMINIT
2. COMRESET not detected and power-on reset deasserted and presence detection enabled	→	***DR_PS_Presence***
3. COMRESET detected or power-on reset asserted	→	DR_Reset
NOTE : 1. This state is entered asynchronously any time in response to power-on reset or receipt of a COMRESET signal from the host.		

DP12: DR_PS_Presence	***Transmit COMWAKE***	
1. Phy online	→	***DR_COMINIT***
2. Phy offline	→	***DR_PS_Wait***

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199

DP13: DR_PS_Wait	*Interface quiescent*	
1. Phy online	→	*DR_COMINIT*
2. Phy offline	→	*DR_PS_Wait*

17.3.1.2.2 Host Phy state machine impact (Informative)

A host connected to a Port Selector performing presence detection receives a COMWAKE signal while in the HP2: HR_AwaitCOMINIT state of the Phy state machine defined in section 8.3.1. This state is insensitive to receiving a COMWAKE and only performs a transition to a new state when COMINIT is received. Host designers should ensure that their implementations are insensitive to receiving a COMWAKE in this state. If the host design is capable of latching the event of receiving a COMWAKE while in this state, the host should expose that event using the SError register enhancement detailed in section 17.3.1.2.3.

17.3.1.2.3 SError Register Enhancement for Presence Detection

The Serial ATA interface error register (SError) as defined in section 14.1.2 includes indications for various events that may have occurred on the interface such as a change in the PHYRDY state, detection of disparity, errors, and so on. In order to facilitate a means for notifying host software that a Port Selector presence detection signal was received, the A bit in the DIAG field of the SError register is set to one when COMWAKE is received while the host is in state HP2: HR_AwaitCOMINIT.

17.3.1.3 Host Transmission Considerations (Informative)

In order to ensure the port selection signal is reliably conveyed to the Port Selector, the host should account for any other interface activity that may interfere with the transmitted COMRESET port selection sequence. For example, if the host periodically issues a COMRESET signal as part of a hardware-polled device presence detection mechanism, a periodic COMRESET signal could occur during the port selection signaling sequence, thereby corrupting the port selection sequence. In order to avoid such interactions, the host may elect to continually transmit the port selection sequence while monitoring the Phy status in the associated superset Status and Control register. When the port selection signal is recognized by the Port Selector and has taken effect, the host can detect a change in the PHYRDY status since the associated port is activated and communications with it are established. It is recommended that the host check the PHYRDY signal immediately before issuing each COMRESET burst in the protocol-based selection signal and only issue the next COMRESET burst if PHYRDY is not asserted.

17.3.2 Side-band Port Selection

The active host port may be selected by a side-band mechanism. Side-band port selection uses a mechanism outside of the Serial ATA protocol for selecting which host port is active. One example of a side-band port selection mechanism is a hardware select line. The side-band selection mechanism used is outside the scope of this specification. A Port Selector that supports side-band port selection shall exhibit the behavior defined within this specification.

17.3.3 Behavior during a change of active port

During a change of active port, the previous host connection is broken and all internal state other than the active host port is initialized before the connection with the new active host is made. When a new active host port is selected, the Port Selector shall perform the following procedure:

1. The Port Selector shall stop transmitting and enter the quiescent power condition on the previously active host port Phy (now the inactive host port).
2. The Port Selector shall initialize all internal state other than the state of the selection bit for the active host port.
3. The Port Selector shall enter the active power condition on the new active host port.
4. The Port Selector shall allow OOB and normal traffic to proceed between the new active host port and the device.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 537 of 587

17.3.3.1 Device State after a change of active port (Informative)

A Port Selector may support an orderly switch to a new active host port. A Port Selector that supports an orderly switch ensures that primitive alignment with the device Phy is maintained during the switch to the new active host port. Maintaining primitive alignment ensures that PHYRDY remains present between the Port Selector and the device throughout the switch to the new active host port.

After selection of a new active port, the device may be in an unknown state. The device may have active commands outstanding from the previously active host port that need to be flushed. The new active host can issue a COMRESET to the device in order to return the device to a known state.

17.4 Behavior and Policies

17.4.1 Control State Machine

The Port Selector Control state machine is based on a model of a Port Selector consisting of three Serial ATA Phys interconnected and controlled by an overall control logic block as depicted in Figure 227. For convenience, the three ports of a Port Selector are abbreviated "A," "B," and "D" corresponding to Host Port A, Host Port B, and Device Port respectively.

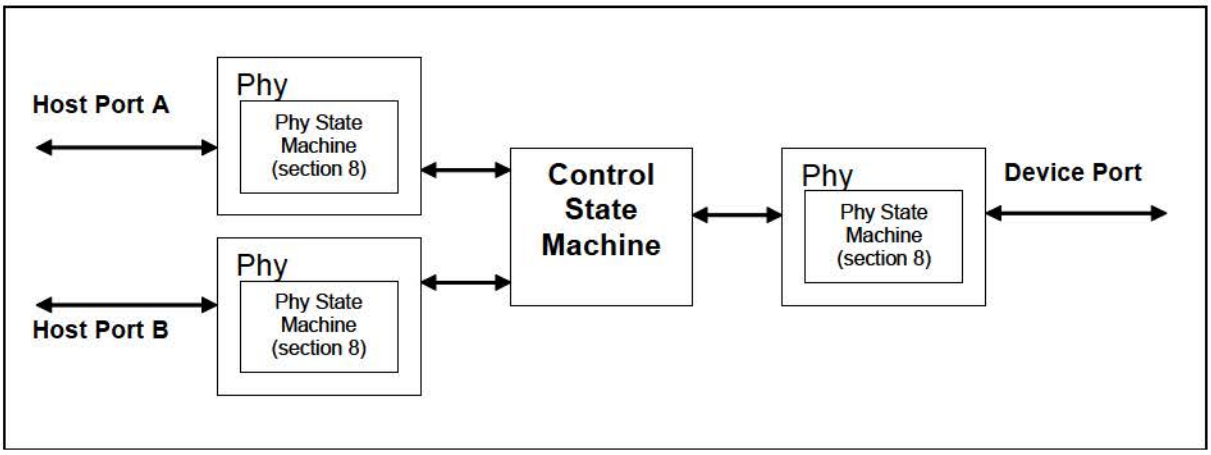


Figure 227 – Control State Machine

The Phys depicted in Figure 227 are presumed to have the basic capabilities and controls indicated in Figure 228. Figure 228 is a variant of Figure 64. The differences are:

- Loopback controls were removed since those controls were not relevant in this state machine.
- Explicit ONLINE and OFFLINE signals were added, including a register latch for these signals so that the signals do not need to be asserted in every state. The SControl register specifies a mechanism for the host to put the Phy in offline mode. Therefore, it is reasonable to expect that the Phy has signals ONLINE and OFFLINE that can be utilized.
- A PORTSELECT signal was added. A Port Selector using protocol-based port selection shall set the PORTSELECT signal to the output of the Sequence Detect block. The Sequence Detect block is asserted when the protocol-based selection signal is received, otherwise the signal is de-asserted. A Port Selector using side-band port selection shall set the PORTSELECT signal when a change in active port is requested.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 538 of 587

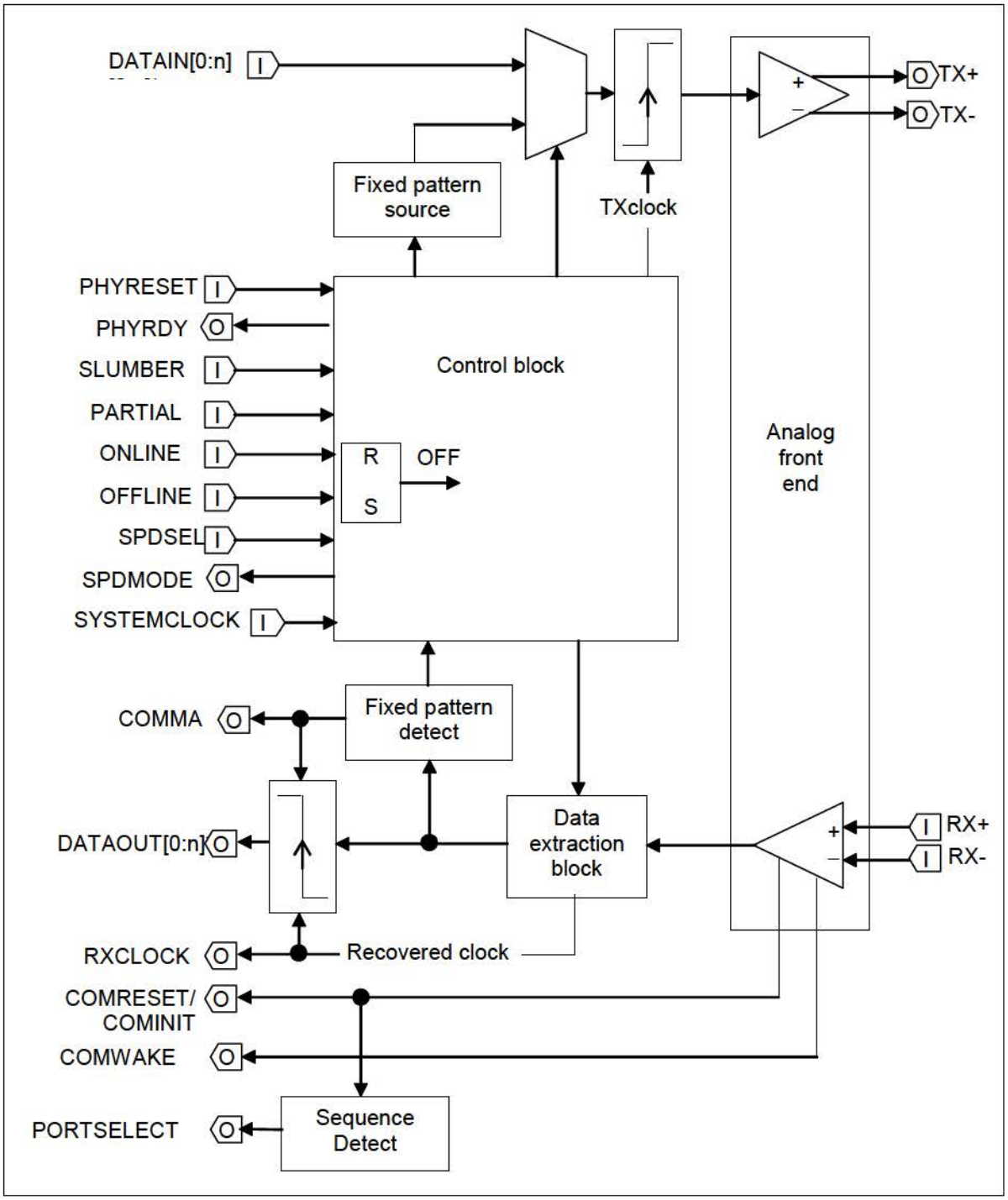


Figure 228 – Phy Block Diagram

The following state diagram specifies the behavior of the Port Selector control logic block. The Port Selector shall have the externally visible behavior described by this state machine.

Reception of a COMRESET signal from the selected host port shall unconditionally force the control state machine to transition to state PS15: ResetDevice. Reception of a COMINIT signal

from the device shall unconditionally force the control state machine to transition to state PS14: ResetHost if not in the ResetDevice state. For the sake of brevity, this implied transition has been omitted from most states.

PS1: PORReset[1]	Set internal PS state to initial conditions including MaxNegSpeed=Max and SelHostPort=none. Assert A.OFFLINE. Assert B.OFFLINE. Assert D.OFFLINE.	
1. Power-on reset and explicit reset request deasserted and Mode = SideBand	→	SetHostPortSideBand
2. Power-on reset and explicit reset request deasserted and Mode = ProtocolBased	→	AwaitHostSelection
3. Power-on reset or explicit reset request asserted	→	PORReset
NOTE : 1. This state is entered asynchronously any time in response to power-on reset or an explicit reset request. An explicit reset request is a reset line/button on the Port Selector itself, not a COMRESET signal from a host.		

PS2: SetHostPortSideBand	Set SelHostPort based on value of side-band selection signal.	
1. COMRESET received from SelHostPort		ResetDevice
2. COMINIT received from D and COMRESET not received from SelHostPort		ResetHost
3. COMRESET not received from SelHostPort and COMINIT not received from D	→	SetHostPortSideBand

PS3: ComInitPropToBoth	Assert A.ONLINE. Assert A.PHYRESET. Assert B.ONLINE. Assert B.PHYRESET.	
1. Unconditional	→	AwaitHostSelection

PS4: AwaitHostSelection		
1. COMINIT received from D and (COMRESET or COMWAKE) not received from (A or B)	→	ComInitPropToBoth
2. (COMRESET or COMWAKE) received from A	→	SelectA
3. (COMRESET or COMWAKE) not received from A and (COMRESET or COMWAKE) received from B	→	SelectB
4. No OOB signal detected	→	AwaitHostSelection

PS5: SelectA	Assert B.OFFLINE. Set SelHostPort=A. Assert SelHostPort.ONLINE. Assert D.ONLINE. Assert SelHostPort.PHYRESET. Assert D.PHYRESET.	
1. Unconditional	→	WaitforComm

PS6: SelectB	Assert A.OFFLINE. Set SelHostPort=B. Assert SelHostPort.ONLINE. Assert D.ONLINE. Assert SelHostPort.PHYRESET. Assert D.PHYRESET.	
1. Unconditional	→	WaitforComm

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 540 of 587

PS7: WaitForComm		
1. SelHostPort.PHYRDY and D.PHYRDY and no change in PORTSELECT signal	→	CheckSpeeds
2. (!SelHostPort.PHYRDY or !D.PHYRDY) and timeout[1] not exceeded and no change in PORTSELECT signal	→	WaitForComm
3. (!SelHostPort.PHYRDY or !D.PHYRDY) and timeout[1] exceeded and no change in PORTSELECT signal	→	PowerManageCheck
4. PORTSELECT signal received for non-selected host port	→	ChangePort
NOTE: 1. The timeout is vendor specific but shall be larger than 1760 microseconds.		

PS8: CheckSpeeds[1]		
1. SelHostPort.SPDMODE = D.SPDMODE	→	Online
2. SelHostPort.SPDMODE > D.SPDMODE	→	SetDeviceSpeed
3. SelHostPort.SPDMODE < D.SPDMODE	→	SetHostSpeed
NOTE: 1. A larger value for the SPDMODE signal shall indicate a higher speed than a smaller value for SPDMODE.		

PS9: Online	Transfer DATA received on D to SelHostPort. Transfer DATA received on SelHostPort to D.	
1. PORTSELECT signal received for non-selected host port	→	ChangePort
2. SelHostPort.PHYRDY deasserted or D.PHYRDY deasserted	→	PowerManageCheck

PS10: SetDeviceSpeed	Set MaxNegSpeed=D.SPDMODE	
1. Unconditional	→	ReComm

PS11: SetHostSpeed	Set MaxNegSpeed=SelHostPort.SPDMODE	
1. Unconditional	→	ReComm

PS12: ReComm	Assert D.PHYRESET	
1. Unconditional	→	WaitForComm

PS13: ChangePort	Assert SelHostPort.OFFLINE. Assert !SelHostPort.ONLINE. Set SelHostPort=!SelHostPort. Set MaxNegSpeed=Max.	
1. Unconditional	→	WaitForComm

PS14: ResetHost	Assert SelHostPort.ONLINE. Assert D.ONLINE. Assert SelHostPort.PHYRESET	
1. Unconditional	→	WaitForComm
NOTE: 1. This state is entered unconditionally upon receipt of D.COMINIT if SelHostPort != none and the Port Selector is not in state ResetDevice.		

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 541 of 587

PS15: ResetDevice[1]	Assert SelHostPort.ONLINE. Assert D.ONLINE. Assert D.PHYRESET.		
	1. Unconditional	→	WaitForComm
	NOTE: 1. This state is entered unconditionally upon receipt of SelHostPort.COMRESET if SelHostPort != none.		

PS16: PowerManageCheck			
	1. Port Selector determined that low power state entered is SLUMBER	→	PowerManageSlumber
	2. Port Selector has not determined that low power state entered is SLUMBER	→	PowerManagePartial

PS17: PowerManagePartial	Assert SelHostPort.Partial. Assert D.Partial.		
	1. SelHostPort.COMWAKE detected or D.COMWAKE detected	→	WaitForComm
	2. PORTSELECT signal received for non-active host port	→	ChangePort
	3. COMWAKE not received and no change in PORTSELECT signal	→	PowerManagePartial

PS18: PowerManageSlumber	Assert SelHostPort.Slumber. Assert D.Slumber.		
	1. SelHostPort.COMWAKE detected or D.COMWAKE detected	→	WaitForComm
	2. PORTSELECT signal received for non-active host port	→	ChangePort
	3. COMWAKE not received and no change in PORTSELECT signal	→	PowerManageSlumber

17.4.2 BIST support

A Port Selector is not required to support the BIST Activate FIS. The resultant behavior of sending a BIST Activate FIS through a Port Selector is undefined.

17.4.3 Flow control signaling latency

The Port Selector shall satisfy the flow control signaling latency specified in section 9.4.7. The Port Selector shall ensure that the flow control signaling latency is met on a per link basis. Specifically, the Port Selector shall ensure that the flow control signaling latency is met between:

1. The Port Selector active host port and the host it is connected to
2. The Port Selector device port and the device it is connected to

The Port Selector shall not reduce the flow control signaling latency budget of the active host it is connected to or the device it is connected to.

17.4.4 Power Management

The Port Selector shall maintain the active host port across power management events and only allow an active host port change after receiving a valid port selection signal.

The Phy on the inactive host port shall be in the quiescent power condition. Upon detecting that the PHYRDY signal is not present for the active host port or device port, the Port Selector shall place that Phy in a quiescent power condition.

If the PHYRDY signal is not present between the device and the Port Selector, the Phy connected to the active host port shall enter the quiescent power condition and squelch the Phy transmitter. If the PHYRDY signal is not present between the active host and the Port Selector,

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 542 of 587

the Phy connected to the device shall enter the quiescent power condition and squelch the Phy transmitter. During these periods while PHYRDY is not present, OOB signals shall still be propagated between the active host and the device to ensure that communication can be established.

If the Port Selector is able to determine that the active host and device negotiated a Slumber power management transition, the Port Selector may recover from the quiescent power condition in the time defined by the Slumber power state. If the Port Selector is not able to determine the power state entered by the host and device, the Port Selector shall recover from the quiescent power condition in the time defined by the Partial power state.

17.4.4.1 Wakeup Budget

The wakeup budget out of Partial or Slumber may increase when a Port Selector is connected to a device. When the active host Phy comes out of low power condition, the Port Selector active host Phy may wakeup before causing the Port Selector device Phy to wakeup which in turn wakes the device. The host shall allow the device at least 20 microseconds to wakeup from the Partial power management state.

17.4.5 OOB Phy signals

The Port Selector shall propagate COMRESET received from the active host to the device as specified in the Control State Machine in section 17.4.1. The Port Selector shall propagate COMINIT received from the device to the active host port as specified in the Control State Machine in section 17.4.1. If no active host port is selected, the Port Selector shall propagate COMINIT received from the device to the active host port as specified in the Control State Machine in section 17.4.1. The Port Selector is allowed to delay delivery of propagated OOB signals.

The Port Selector shall not respond to COMRESET signals received over the inactive host port. The inactive host port Phy shall remain in the quiescent power condition when COMRESET is received over the inactive host port.

17.4.6 Hot Plug

The Port Selector shall only generate a COMINIT over a host port when a COMINIT signal is received from the device or as part of an active speed negotiation as specified in the Control State Machine in section 17.4.1. If a drive connected to a Port Selector is hot plugged, the drive issues a COMINIT sequence as part of its normal power-up sequence in accordance with the Phy layer state machine. The Port Selector shall propagate the COMINIT over the active host port (or both host ports if both host ports are inactive). When the host detects the COMINIT signal, the host then interrogates the port to determine whether a drive is attached.

If a drive connected to a Port Selector is hot unplugged, the Port Selector shall squelch the transmitter for the active host port as detailed in section 17.4.4. The active host should then determine that the PHYRDY signal is no longer present and therefore that a drive is no longer present.

17.4.7 Speed Negotiation

Speed is negotiated on a per link basis. Specifically, the Port Selector shall negotiate speed between:

1. The Port Selector active host port and the host to which it is connected
2. The Port Selector device port and the device to which it is connected

The Port Selector starts speed negotiation at the highest speed rate that it supports. The Port Selector then negotiates speed on each link to the appropriate supported speed. After negotiating speed on the active host link and on the device link, the Port Selector shall check whether the two speeds match. If the speeds do not match, the Port Selector limits the maximum

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 543 of 587

speed rate it supports to the lower of the two speeds negotiated. Then the Port Selector forces speed to be renegotiated to reach a common speed rate.

17.4.8 Spread spectrum clocking

The Port Selector shall support spread spectrum clocking receive on all of its ports. The Port Selector may support spread spectrum transmit. It is recommended that a configuration jumper be used to enable/disable spread spectrum clocking if it is settable. There is no means within the Serial ATA protocol provided to enable/disable spread spectrum clocking if it is statically configurable.

If spread spectrum clocking is used, the spreading domain between the host and the Port Selector is not required to be the same as the spreading domain between device and the Port Selector. The signals passing through a Port Selector may be re-spread.

17.5 Power-up and Resets

17.5.1 Power-up

Upon power-up, the Port Selector shall reset all internal state, including the active host port. This causes no active host port to be selected when protocol-based port selection is used.

17.5.1.1 Presence detection of Port Selector

For protocol-based port selection, presence detection can be performed using the optional mechanism outlined in section 17.3.1.2. Presence detection for Port Selectors implementing side-band port selection is outside the scope of this specification.

17.5.2 Resets

17.5.2.1 COMRESET

When COMRESET is received over the active host port the Port Selector shall reset all internal state, the active host port shall remain unchanged, the maximum speed shall remain unchanged, and the COMRESET signal shall be propagated to the device. The Port Selector shall take no reset action upon receiving a COMRESET signal over the inactive host port except as specified in section 17.3.1 when there is no active host port selected after power-up.

17.5.2.2 Software reset and DEVICE RESET

The Port Selector shall not reset in response to receiving a Software reset or the DEVICE RESET command.

17.6 Host Implementation (Informative)

17.6.1 Software Method for Protocol-based Selection (Informative)

The preferred software method for producing a protocol-based port selection signal is detailed in this section. Software for HBAs that implement the SControl and SStatus registers may use this method to create the protocol-based port selection signal.

If the Phy is left on for long periods during the generation of the sequence, a hardware based COMRESET polling algorithm may interfere and corrupt the sequence. This method tries to minimize any impact of host based COMRESET polling algorithms by leaving the Phy online for very short sequences during the creation of the signal. The procedure outlined is appropriate for any HBA design that has a COMRESET polling interval greater than 25 microseconds.

1. Set Phy to offline by writing SControl.DET to 4h.
2. Set Phy to reset state by writing SControl.DET to 1h.
3. Wait 5 microseconds to allow charging time for DC coupled Phy designs.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 544 of 587

4. Set Phy to online state by writing SControl.DET to 0h.
5. Wait 20 microseconds to allow COMRESET burst to be transmitted to the device.
6. Set Phy to offline by writing SControl DET to 4h.
7. Wait 1.975 milliseconds to satisfy T1 timing as specified in Table 44.

8. Repeat steps 2-6.
9. Wait 7.975 milliseconds to satisfy T2 timing as specified in Table 44.

10. Repeat steps 2-6.
11. Wait 1.975 milliseconds to satisfy T1 timing as specified in Table 44.

12. Repeat steps 2-6
13. Wait 7.975 milliseconds to satisfy T2 timing as specified in Table 44.

14. Set Phy to reset state by writing SControl.DET to 1h.
15. Wait 5 microseconds to allow charging time for DC coupled Phy designs.
16. Set Phy to online state by writing SControl.DET to 0h.
17. Wait up to 10 milliseconds for SStatus.DET = 3h
18. If SStatus.DET != 3h, go to step 1 to restart the process.

The procedure is also outlined in pseudocode on the following page.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 545 of 587

```
//
// Continue to perform this procedure until SStatus.DET == 3.
//
while (SStatus.DET != 3)
{
   SControl.DET = 4;             // Turn off Phy

   //
   // Mimic out the COMRESET bursts at the appropriate T1/T2 timing intervals.
   //
   for (i = 0; i < 4; i++)
   {
      SControl.DET = 1;          // Place HBA in reset state
      Sleep(5);                  // Wait for Phy to charge, in microseconds
      SControl.DET = 0;          // Issue COMRESET
      Sleep(20);                 // Wait for COMRESET to be sent
      SControl.DET = 4;          // Turn off Phy

      if ((i == 0) || (i == 2))
      {
         Sleep(1975);            // Wait T1 time minus time already waited
      }
      else if ((i == 1) || (i == 3))
      {
         Sleep(7975);            // Wait T2 time minus time already waited
      }
   }

   //
   // Issue final COMRESET of the burst.
   //
   SControl.DET = 1;             // Place HBA in reset state
   Sleep(5);                     // Wait for Phy to charge, in microseconds
   SControl.DET = 0;             // Issue COMRESET


   //
   // Wait up to 10 milliseconds for PHYRDY.
   //
   for (i = 0; i < 10000; i++)
   {
      if (SStatus.DET == 3)
      {
         break;                  // Stop procedure if SStatus.DET == 3
      }
      Sleep(1);                  // Wait 1 microsecond
   }
```

APPENDIX A. SAMPLE CODE FOR CRC AND SCRAMBLING (INFORMATIVE)

A.1 CRC calculation

A.1.1 Overview

The following section provides an informative implementation of the CRC polynomial. The example is intended as an aid in verifying a HDL implementation of the algorithm.

A.1.2 Maximum frame size

The 32-bit CRC used by Serial ATA can be shown to provide detection of two 10-bit errors up to a maximum frame size of 16384 bytes. This provides for future expansion of FISes to a maximum of 64 bytes of fixed overhead while still permitting a maximum user data payload of 8192 bytes.

A.1.3 Example code for CRC algorithm

The following code, written in C, illustrates an implementation of the CRC algorithm. A Register Host to Device FIS containing a PIO write command is used as example input. The CRC calculated is the check Dword appended to a transmitted serial stream immediately preceding the EOF_P primitive.. The code displays both the resulting command FIS and the intermediate CRC polynomial values. To compile the code with the GNU tool chain, use the following command:

"gcc –o crc.exe crc.c"

This code is supplied for illustrative purposes only.

A.1.4 Example code for CRC algorithm

```
/*****************************************************************************/
/*                                                                           */
/* crc.c                                                                     */
/*                                                                           */
/* This sample code reads standard in for a sequence of 32 bit values       */
/* formatted in hexadecimal with a leading "0x" (e.g. 0xDEADBEEF).  The     */
/* code calculates the Serial ATA CRC for the input data stream.  The       */
/* generator polynomial used is:                                            */
/*         32   26   23   22   16   12   11   10   8   7   5   4   2        */
/* G(x) = x  + x  + x  + x  + x  + x  + x  + x  + x + x + x + x + x + x + 1 */
/*                                                                           */
/* This sample code uses a parallel implementation of the CRC calculation   */
/* circuit that is suitable for implementation in hardware.  A block        */
/* diagram of the circuit being emulated is shown below.                    */
/*                                                                           */
/*                      +---+          +---+          +---+                 */
/*   Data_In ---------->|   |          |   |          | R |                 */
/*                      | + |--------->| * |--------->| e |----+            */
/*                +---->|   |          |   |          | g |    |            */
/*                |     +---+          +---+          +---+    |            */
/*                |                                            |            */
/*                |                                            |            */
/*                +--------------------------------------------+            */
/*                                                                           */
/* The CRC value is initialized to 0x52325032 as defined in the Serial ATA   */
/* specification.                                                            */
/*                                                                           */
/*****************************************************************************/

#include <stdlib.h>
#include <stdio.h>
```


Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 547 of 587

```
main(argc,argv)
int argc;
char *argv[];
{
   int               i,
                     data_count;
   unsigned int      crc,
                     data_in;
   unsigned char     crc_bit[32],
                     new_bit[32];


   crc = 0x52325032;
   data_count = 0;

   while (scanf(" 0x%8x", &data_in) == 1)  {
      data_count++;
      /* Add the data_in value to the current value of the CRC held in the  */
      /* "register".  The addition is performed modulo two (XOR).           */
      crc ^= data_in;
      /* Expand the value of the CRC held in the register to 32 individual  */
      /* bits for easy manipulation.                                        */
      for (i = 0; i < 32; ++i)  {
         crc_bit[i] = (crc >> i) & 0x01;
      }

      /* The following 32 assignments perform the function of the box       */
      /* labeled "*" in the block diagram above.  The new_bit array is a    */
      /* temporary holding place for the new CRC value being calculated.    */
      /* Note that there are lots of shared terms in the assignments below. */
      new_bit[31] = crc_bit[31] ^ crc_bit[30] ^ crc_bit[29] ^ crc_bit[28] ^ crc_bit[27] ^ crc_bit[25] ^ crc_bit[24] ^
                    crc_bit[23] ^ crc_bit[15] ^ crc_bit[11] ^ crc_bit[9]  ^ crc_bit[8]  ^ crc_bit[5];
      new_bit[30] = crc_bit[30] ^ crc_bit[29] ^ crc_bit[28] ^ crc_bit[27] ^ crc_bit[26] ^ crc_bit[24] ^ crc_bit[23] ^
                    crc_bit[22] ^ crc_bit[14] ^ crc_bit[10] ^ crc_bit[8]  ^ crc_bit[7]  ^ crc_bit[4];
      new_bit[29] = crc_bit[31] ^ crc_bit[29] ^ crc_bit[28] ^ crc_bit[27] ^ crc_bit[26] ^ crc_bit[25] ^ crc_bit[23] ^
                    crc_bit[22] ^ crc_bit[21] ^ crc_bit[13] ^ crc_bit[9]  ^ crc_bit[7]  ^ crc_bit[6]  ^ crc_bit[3];
      new_bit[28] = crc_bit[30] ^ crc_bit[28] ^ crc_bit[27] ^ crc_bit[26] ^ crc_bit[25] ^ crc_bit[24] ^ crc_bit[22] ^
                    crc_bit[21] ^ crc_bit[20] ^ crc_bit[12] ^ crc_bit[8]  ^ crc_bit[6]  ^ crc_bit[5]  ^ crc_bit[2];
      new_bit[27] = crc_bit[29] ^ crc_bit[27] ^ crc_bit[26] ^ crc_bit[25] ^ crc_bit[24] ^ crc_bit[23] ^ crc_bit[21] ^
                    crc_bit[20] ^ crc_bit[19] ^ crc_bit[11] ^ crc_bit[7]  ^ crc_bit[5]  ^ crc_bit[4]  ^ crc_bit[1];
      new_bit[26] = crc_bit[31] ^ crc_bit[28] ^ crc_bit[26] ^ crc_bit[25] ^ crc_bit[24] ^ crc_bit[23] ^ crc_bit[22] ^
                    crc_bit[20] ^ crc_bit[19] ^ crc_bit[18] ^ crc_bit[10] ^ crc_bit[6]  ^ crc_bit[4]  ^ crc_bit[3]  ^
                    crc_bit[0];
      new_bit[25] = crc_bit[31] ^ crc_bit[29] ^ crc_bit[28] ^ crc_bit[22] ^ crc_bit[21] ^ crc_bit[19] ^ crc_bit[18] ^
                    crc_bit[17] ^ crc_bit[15] ^ crc_bit[11] ^ crc_bit[8]  ^ crc_bit[3]  ^ crc_bit[2];
      new_bit[24] = crc_bit[30] ^ crc_bit[28] ^ crc_bit[27] ^ crc_bit[21] ^ crc_bit[20] ^ crc_bit[18] ^ crc_bit[17] ^
                    crc_bit[16] ^ crc_bit[14] ^ crc_bit[10] ^ crc_bit[7]  ^ crc_bit[2]  ^ crc_bit[1];
      new_bit[23] = crc_bit[31] ^ crc_bit[29] ^ crc_bit[27] ^ crc_bit[26] ^ crc_bit[20] ^ crc_bit[19] ^ crc_bit[17] ^
                    crc_bit[16] ^ crc_bit[15] ^ crc_bit[13] ^ crc_bit[9]  ^ crc_bit[6]  ^ crc_bit[1]  ^ crc_bit[0];
      new_bit[22] = crc_bit[31] ^ crc_bit[29] ^ crc_bit[27] ^ crc_bit[26] ^ crc_bit[24] ^ crc_bit[23] ^ crc_bit[19] ^
                    crc_bit[18] ^ crc_bit[16] ^ crc_bit[14] ^ crc_bit[12] ^ crc_bit[11] ^ crc_bit[9]  ^ crc_bit[0];
      new_bit[21] = crc_bit[31] ^ crc_bit[29] ^ crc_bit[27] ^ crc_bit[26] ^ crc_bit[24] ^ crc_bit[22] ^ crc_bit[18] ^
                    crc_bit[17] ^ crc_bit[13] ^ crc_bit[10] ^ crc_bit[9]  ^ crc_bit[5];
      new_bit[20] = crc_bit[30] ^ crc_bit[28] ^ crc_bit[26] ^ crc_bit[25] ^ crc_bit[23] ^ crc_bit[21] ^ crc_bit[17] ^
                    crc_bit[16] ^ crc_bit[12] ^ crc_bit[9]  ^ crc_bit[8]  ^ crc_bit[4];
      new_bit[19] = crc_bit[29] ^ crc_bit[27] ^ crc_bit[25] ^ crc_bit[24] ^ crc_bit[22] ^ crc_bit[20] ^ crc_bit[16] ^
                    crc_bit[15] ^ crc_bit[11] ^ crc_bit[8]  ^ crc_bit[7]  ^ crc_bit[3];
      new_bit[18] = crc_bit[31] ^ crc_bit[28] ^ crc_bit[26] ^ crc_bit[24] ^ crc_bit[23] ^ crc_bit[21] ^ crc_bit[19] ^
                    crc_bit[15] ^ crc_bit[14] ^ crc_bit[10] ^ crc_bit[7]  ^ crc_bit[6]  ^ crc_bit[2];
      new_bit[17] = crc_bit[31] ^ crc_bit[30] ^ crc_bit[27] ^ crc_bit[25] ^ crc_bit[23] ^ crc_bit[22] ^ crc_bit[20] ^
                    crc_bit[18] ^ crc_bit[14] ^ crc_bit[13] ^ crc_bit[9]  ^ crc_bit[6]  ^ crc_bit[5]  ^ crc_bit[1];
      new_bit[16] = crc_bit[30] ^ crc_bit[29] ^ crc_bit[26] ^ crc_bit[24] ^ crc_bit[22] ^ crc_bit[21] ^ crc_bit[19] ^
                    crc_bit[17] ^ crc_bit[13] ^ crc_bit[12] ^ crc_bit[8]  ^ crc_bit[5]  ^ crc_bit[4]  ^ crc_bit[0];
      new_bit[15] = crc_bit[30] ^ crc_bit[27] ^ crc_bit[24] ^ crc_bit[21] ^ crc_bit[20] ^ crc_bit[18] ^ crc_bit[16] ^
                    crc_bit[15] ^ crc_bit[12] ^ crc_bit[9]  ^ crc_bit[8]  ^ crc_bit[7]  ^ crc_bit[5]  ^ crc_bit[4]  ^
                    crc_bit[3];
      new_bit[14] = crc_bit[29] ^ crc_bit[26] ^ crc_bit[23] ^ crc_bit[20] ^ crc_bit[19] ^ crc_bit[17] ^ crc_bit[15] ^
                    crc_bit[14] ^ crc_bit[11] ^ crc_bit[8]  ^ crc_bit[7]  ^ crc_bit[6]  ^ crc_bit[4]  ^ crc_bit[3]  ^
                    crc_bit[2];
      new_bit[13] = crc_bit[31] ^ crc_bit[28] ^ crc_bit[25] ^ crc_bit[22] ^ crc_bit[19] ^ crc_bit[18] ^ crc_bit[16] ^
                    crc_bit[14] ^ crc_bit[13] ^ crc_bit[10] ^ crc_bit[7]  ^ crc_bit[6]  ^ crc_bit[5]  ^ crc_bit[3]  ^
                    crc_bit[2]  ^ crc_bit[1];
      new_bit[12] = crc_bit[31] ^ crc_bit[30] ^ crc_bit[27] ^ crc_bit[24] ^ crc_bit[21] ^ crc_bit[18] ^ crc_bit[17] ^
                    crc_bit[15] ^ crc_bit[13] ^ crc_bit[12] ^ crc_bit[9]  ^ crc_bit[6]  ^ crc_bit[5]  ^ crc_bit[4]  ^
                    crc_bit[2]  ^ crc_bit[1]  ^ crc_bit[0];
      new_bit[11] = crc_bit[31] ^ crc_bit[28] ^ crc_bit[27] ^ crc_bit[26] ^ crc_bit[25] ^ crc_bit[24] ^ crc_bit[20] ^
                    crc_bit[17] ^ crc_bit[16] ^ crc_bit[15] ^ crc_bit[14] ^ crc_bit[12] ^ crc_bit[9]  ^ crc_bit[4]  ^
                    crc_bit[3]  ^ crc_bit[1]  ^ crc_bit[0];
      new_bit[10] = crc_bit[31] ^ crc_bit[29] ^ crc_bit[28] ^ crc_bit[26] ^ crc_bit[19] ^ crc_bit[16] ^ crc_bit[14] ^
                    crc_bit[13] ^ crc_bit[9]  ^ crc_bit[5]  ^ crc_bit[3]  ^ crc_bit[2]  ^ crc_bit[0];
      new_bit[9]  = crc_bit[29] ^ crc_bit[24] ^ crc_bit[23] ^ crc_bit[18] ^ crc_bit[13] ^ crc_bit[12] ^ crc_bit[11] ^
```

```
                crc_bit[9]  ^ crc_bit[5]  ^ crc_bit[4]  ^ crc_bit[2]  ^ crc_bit[1];
    new_bit[8]  = crc_bit[31] ^ crc_bit[28] ^ crc_bit[23] ^ crc_bit[22] ^ crc_bit[17] ^ crc_bit[12] ^ crc_bit[11] ^
                crc_bit[10] ^ crc_bit[8]  ^ crc_bit[4]  ^ crc_bit[3]  ^ crc_bit[1]  ^ crc_bit[0];
    new_bit[7]  = crc_bit[29] ^ crc_bit[28] ^ crc_bit[25] ^ crc_bit[24] ^ crc_bit[23] ^ crc_bit[22] ^ crc_bit[21] ^
                crc_bit[16] ^ crc_bit[15] ^ crc_bit[10] ^ crc_bit[8]  ^ crc_bit[7]  ^ crc_bit[5]  ^ crc_bit[3]  ^
                crc_bit[2]  ^ crc_bit[0];
    new_bit[6]  = crc_bit[30] ^ crc_bit[29] ^ crc_bit[25] ^ crc_bit[22] ^ crc_bit[21] ^ crc_bit[20] ^ crc_bit[14] ^
                crc_bit[11] ^ crc_bit[8]  ^ crc_bit[7]  ^ crc_bit[6]  ^ crc_bit[5]  ^ crc_bit[4]  ^ crc_bit[2]  ^
                crc_bit[1];
    new_bit[5]  = crc_bit[29] ^ crc_bit[28] ^ crc_bit[24] ^ crc_bit[21] ^ crc_bit[20] ^ crc_bit[19] ^ crc_bit[13] ^
                crc_bit[10] ^ crc_bit[7]  ^ crc_bit[6]  ^ crc_bit[5]  ^ crc_bit[4]  ^ crc_bit[3]  ^ crc_bit[1]  ^
                crc_bit[0];
    new_bit[4]  = crc_bit[31] ^ crc_bit[30] ^ crc_bit[29] ^ crc_bit[25] ^ crc_bit[24] ^ crc_bit[20] ^ crc_bit[19] ^
                crc_bit[18] ^ crc_bit[15] ^ crc_bit[12] ^ crc_bit[11] ^ crc_bit[8]  ^ crc_bit[6]  ^ crc_bit[4]  ^
                crc_bit[3]  ^ crc_bit[2]  ^ crc_bit[0];
    new_bit[3]  = crc_bit[31] ^ crc_bit[27] ^ crc_bit[25] ^ crc_bit[19] ^ crc_bit[18] ^ crc_bit[17] ^ crc_bit[15] ^
                crc_bit[14] ^ crc_bit[10] ^ crc_bit[9]  ^ crc_bit[8]  ^ crc_bit[7]  ^ crc_bit[3]  ^ crc_bit[2]  ^
                crc_bit[1];
    new_bit[2]  = crc_bit[31] ^ crc_bit[30] ^ crc_bit[26] ^ crc_bit[24] ^ crc_bit[18] ^ crc_bit[17] ^ crc_bit[16] ^
                crc_bit[14] ^ crc_bit[13] ^ crc_bit[9]  ^ crc_bit[8]  ^ crc_bit[7]  ^ crc_bit[6]  ^ crc_bit[2]  ^
                crc_bit[1]  ^ crc_bit[0];
    new_bit[1]  = crc_bit[28] ^ crc_bit[27] ^ crc_bit[24] ^ crc_bit[17] ^ crc_bit[16] ^ crc_bit[13] ^ crc_bit[12] ^
                crc_bit[11] ^ crc_bit[9]  ^ crc_bit[7]  ^ crc_bit[6]  ^ crc_bit[1]  ^ crc_bit[0];
    new_bit[0]  = crc_bit[31] ^ crc_bit[30] ^ crc_bit[29] ^ crc_bit[28] ^ crc_bit[26] ^ crc_bit[25] ^ crc_bit[24] ^
                crc_bit[16] ^ crc_bit[12] ^ crc_bit[10] ^ crc_bit[9]  ^ crc_bit[6]  ^ crc_bit[0];


    /* The new CRC value has been calculated as individual bits in the    */
    /* new_bit array.  Re-assembled it into a 32 bit value and "clock" it */
    /* into the "register".                                               */
    crc = 0;
    for (i = 31; i >= 0; --i)  {
       crc = crc << 1;
       crc |= new_bit[i];
    }
    printf("Running CRC value is 0x%08X\n", crc);
  }

  printf("\n\nThe total number of data words processed was %d\n", data_count);
  printf("The CRC is 0x%08X\n\n", crc);

  return 0;
}
```

A.1.5 Example CRC implementation output

The following is the sample data used as input for the example stored in file *sample*:

```
0x00308027
0xE1234567
0x00000000
0x00000002
0x00000000
```

Executing the command *./crc < sample* yields the following output:

```
Running CRC value is 0x11E353FD
Running CRC value is 0x0F656DA7
Running CRC value is 0x3D14369C
Running CRC value is 0x92D0D681
Running CRC value is 0x319FFF6F


The total number of data words processed was 5
The CRC is 0x319FFF6F
```

A.2 Scrambling calculation

A.2.1 Overview

The following section provides an informative implementation of the scrambling polynomial. The example is intended as an aid in verifying a HDL implementation of the algorithm.

A.2.2 Example code for scrambling algorithm

The following code, written in C, illustrates an implementation of the scrambling algorithm. A Register Host to Device FIS containing a PIO write command is used as example input. The code displays both the resulting scrambled data and the raw output of the scrambler. To compile the code with the GNU tool chain, use the following command:

"gcc –o scramble.exe scramble.c"

This code is supplied for illustrative purposes only.

A.2.3 Example scrambler implementation

```
/*****************************************************************************/
/*                                                                           */
/* scramble.c                                                                */
/*                                                                           */
/* This sample code generates the entire sequence of 65535 Dwords produced   */
/* by the scrambler defined in the Serial ATA specification.  The            */
/* specification calls for an LFSR to generate a string of bits that are     */
/* then packaged into 32 bit Dwords to be XORed with the data Dwords.  The   */
/* generator polynomial specified is:                                        */
/*                         16   15   13   4                                  */
/*                 G(x) = x  + x  + x  + x + 1                               */
/*                                                                           */
/* Parallelized versions of the scrambler are initialized to a value         */
/* derived from the initialization value of 0xFFFF defined in the            */
/* specification.  This implementation is initialized to 0xF0F6.  Other      */
/* parallel implementations have different initial values.  The              */
/* important point is that the first Dword output of any implementation      */
/* needs to equal 0xC2D2768D.                                                */
/*                                                                           */
/* This code does not represent an elegant solution for a C implementation,  */
/* but it does demonstrate a method of generating the sequence that can be   */
/* easily implemented in hardware.  A block diagram of the circuit emulated  */
/* by this code is shown below.                                              */
/*                                                                           */
/*         +-----------------------------------+                             */
/*         |                                   |                             */
/*         |                                   |                             */
/*         |     +---+                +---+    |                             */
/*         |     | R |                | * |    |                             */
/*         +---->| e |----------+---->| M |----+----> Output(31 downto 16)   */
/*               | g |          |     | 1 |                                  */
/*               +---+          |     +---+                                  */
/*                              |                                            */
/*                              |     +---+                                  */
/*                              |     | * |                                  */
/*                              +---->| M |---------> Output(15 downto 0)    */
/*                                    | 2 |                                  */
/*                                    +---+                                  */
/*                                                                           */
/* The register shown in the block diagram is a 16 bit register.  The two    */
/* boxes, *M1 and *M2, each represent a multiply by a 16 by 16 binary        */
/* matrix.  A 16 by 16 matrix times a 16 bit vector yields a 16 bit vector.  */
/* The two vectors are the two halves of the 32 bit scrambler value.  The    */
/* upper half of the scrambler value is stored back into the context         */
/* register to be used to generate the next value in the scrambler           */
/* sequence.                                                                 */
/*                                                                           */
/*****************************************************************************/
```

```
#include <stdlib.h>
#include <stdio.h>


main(argc,argv)
int argc;
char *argv[];


{
   int               i, j;
   unsigned short    context;       /* The 16 bit register that holds the context or state */
   unsigned long     scrambler;     /* The 32 bit output of the circuit    */
   unsigned char     now[16];       /* The individual bits of context      */
   unsigned char     next[32];      /* The computed bits of scrambler      */



   /* Parallelized versions of the scrambler are initialized to a value     */
   /* derived from the initialization value of 0xFFFF defined in the        */
   /* specification.  This implementation is initialized to 0xF0F6.  Other  */
   /* parallel implementations have different initial values.  The          */
   /* important point is that the first Dword output of any implementation  */
   /* needs to equal 0xC2D2768D.                                            */
   context = 0xF0F6;


   for (i = 0; i < 65535; ++i)  {
      /* Split the register contents (the variable context) up into its     */
      /* individual bits for easy handling.                                 */
      for (j = 0; j < 16; ++j)  {
         now[j] = (context >> j) & 0x01;
      }

      /* The following 16 assignments implement the matrix multiplication   */
      /* performed by the box labeled *M1.                                  */
      /* Notice that there are lots of shared terms in these assignments.   */
      next[31] = now[12] ^ now[10] ^ now[7]  ^ now[3]  ^ now[1]  ^ now[0];
      next[30] = now[15] ^ now[14] ^ now[12] ^ now[11] ^ now[9]  ^ now[6]  ^ now[3]  ^ now[2]  ^ now[0];
      next[29] = now[15] ^ now[13] ^ now[12] ^ now[11] ^ now[10] ^ now[8]  ^ now[5]  ^ now[3]  ^ now[2]  ^ now[1];
      next[28] = now[14] ^ now[12] ^ now[11] ^ now[10] ^ now[9]  ^ now[7]  ^ now[4]  ^ now[2]  ^ now[1]  ^ now[0];
      next[27] = now[15] ^ now[14] ^ now[13] ^ now[12] ^ now[11] ^ now[10] ^ now[9]  ^ now[8]  ^ now[6]  ^ now[1]  ^ now[0];
      next[26] = now[15] ^ now[13] ^ now[11] ^ now[10] ^ now[9]  ^ now[8]  ^ now[7]  ^ now[5]  ^ now[3]  ^ now[0];
      next[25] = now[15] ^ now[10] ^ now[9]  ^ now[8]  ^ now[7]  ^ now[6]  ^ now[4]  ^ now[3]  ^ now[2];
      next[24] = now[14] ^ now[9]  ^ now[8]  ^ now[7]  ^ now[6]  ^ now[5]  ^ now[3]  ^ now[2]  ^ now[1];
      next[23] = now[13] ^ now[8]  ^ now[7]  ^ now[6]  ^ now[5]  ^ now[4]  ^ now[2]  ^ now[1]  ^ now[0];
      next[22] = now[15] ^ now[14] ^ now[7]  ^ now[6]  ^ now[5]  ^ now[4]  ^ now[1]  ^ now[0];
      next[21] = now[15] ^ now[13] ^ now[12] ^ now[6]  ^ now[5]  ^ now[4]  ^ now[0];
      next[20] = now[15] ^ now[11] ^ now[5]  ^ now[4];
      next[19] = now[14] ^ now[10] ^ now[4]  ^ now[3];
      next[18] = now[13] ^ now[9]  ^ now[3]  ^ now[2];
      next[17] = now[12] ^ now[8]  ^ now[2]  ^ now[1];
      next[16] = now[11] ^ now[7]  ^ now[1]  ^ now[0];

      /* The following 16 assignments implement the matrix multiplication   */
      /* performed by the box labeled *M2.                                  */
      next[15] = now[15] ^ now[14] ^ now[12] ^ now[10] ^ now[6]  ^ now[3]  ^ now[0];
      next[14] = now[15] ^ now[13] ^ now[12] ^ now[11] ^ now[9]  ^ now[5]  ^ now[3]  ^ now[2];
      next[13] = now[14] ^ now[12] ^ now[11] ^ now[10] ^ now[8]  ^ now[4]  ^ now[2]  ^ now[1];
      next[12] = now[13] ^ now[11] ^ now[10] ^ now[9]  ^ now[7]  ^ now[3]  ^ now[1]  ^ now[0];
      next[11] = now[15] ^ now[14] ^ now[10] ^ now[9]  ^ now[8]  ^ now[6]  ^ now[3]  ^ now[2]  ^ now[0];
      next[10] = now[15] ^ now[13] ^ now[12] ^ now[9]  ^ now[8]  ^ now[7]  ^ now[5]  ^ now[3]  ^ now[2]  ^ now[1];
      next[9]  = now[14] ^ now[12] ^ now[11] ^ now[8]  ^ now[7]  ^ now[6]  ^ now[4]  ^ now[2]  ^ now[1]  ^ now[0];
      next[8]  = now[15] ^ now[14] ^ now[13] ^ now[12] ^ now[11] ^ now[10] ^ now[7]  ^ now[6]  ^ now[5]  ^ now[1]  ^ now[0];
      next[7]  = now[15] ^ now[13] ^ now[11] ^ now[10] ^ now[9]  ^ now[6]  ^ now[5]  ^ now[4]  ^ now[3]  ^ now[0];
      next[6]  = now[15] ^ now[10] ^ now[9]  ^ now[8]  ^ now[5]  ^ now[4]  ^ now[2];
      next[5]  = now[14] ^ now[9]  ^ now[8]  ^ now[7]  ^ now[4]  ^ now[3]  ^ now[1];
      next[4]  = now[13] ^ now[8]  ^ now[7]  ^ now[6]  ^ now[3]  ^ now[2]  ^ now[0];
      next[3]  = now[15] ^ now[14] ^ now[7]  ^ now[6]  ^ now[5]  ^ now[3]  ^ now[2]  ^ now[1];
      next[2]  = now[14] ^ now[13] ^ now[6]  ^ now[5]  ^ now[4]  ^ now[2]  ^ now[1]  ^ now[0];
      next[1]  = now[15] ^ now[14] ^ now[13] ^ now[5]  ^ now[4]  ^ now[1]  ^ now[0];
      next[0]  = now[15] ^ now[13] ^ now[4]  ^ now[0];

      /* The 32 bits of the output have been generated in the "next" array. */
      /* Reassemble the bits into a 32 bit Dword.                           */
      scrambler = 0;
      for (j = 31; j >= 0; --j)  {
         scrambler = scrambler << 1;
         scrambler |= next[j];
      }
      /* The upper half of the scrambler output is stored backed into the   */
```

```
      /* register as the saved context for the next cycle.                */
      context = scrambler >> 16;
      printf("0x%08X\n", scrambler);

   }

   return 0;

}
```

A.2.4 Example scrambler implementation output

The following lists the first 32 results generated by the scrambler and the C sample code listed above.

```
0xC2D2768D
0x1F26B368
0xA508436C
0x3452D354
0x8A559502
0xBB1ABE1B
0xFA56B73D
0x53F60B1B
0xF0809C41
0x747FC34A
0xBE865291
0x7A6FA7B6
0x3163E6D6
0xF036FE0C
0x1EF3EA29
0xEB342694
0x53853B17
0xE94ADC4D
0x5D200E88
0x6901EDD0
0xFA9E38DE
0x68DB4B07
0x450A437B
0x960DD708
0x3F35E698
0xFE7698A5
0xC80EF715
0x666090AF
0xFAF0D5CB
0x2B82009F
0x0E317491
0x76F46A1E
```


Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 553 of 587

A.3 Example frame

The following table shows the steps and values used in the transmission of a simple FIS.

For LBA = 1234567 and Sector Count = 2

Table 45 – CRC and scrambler calculation example - PIO Write Command

FIS Data (hex)	Scrambler Value (hex)	Scrambled Data (hex)	Accumulated CRC Value (hex)	Comments
3737B57C	N/A	3737B57C	N/A	SOF_P, primitives not scrambled, scrambler reset
00308027	C2D2768D	C2E2F6AA	11E353FD	Register – Host to Device FIS, Command = 30, Cbit set
E1234567	1F26B368	FE05F60F	0F656DA7	LBA 1234567
00000000	A508436C	A508436C	3D14369C	Extended LBA = 0
00000002	3452D354	3452D356	92D0D681	Device Control Register = 0, 2 sectors
00000000	8A559502	8A559502	319FFF6F	reserved = 0
319FFF6F	BB1ABE1B	8A854174	N/A	CRC
D5D5B57C	N/A	D5D5B57C	N/A	EOF, primitives not scrambled

Note: All values in hexadecimal, shown 31:0

The transmitted Dwords for this Register FIS, prior to 8b/10b encoding, are :

SOF_P
C2E2F6AAh
FE05F60Fh
A508436Ch
3452D356h
8A559502h
8A854174h
EOF_P

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 554 of 587

Appendix B. Command processing overview (Informative)

B.1 Non-data commands

When the Command register is written by the BIOS or software driver, the host adapter sets BSY in the shadow Status register and transmits a Command frame to the device.

When command actions are complete, the device transmits a Register frame to set ending content of the shadow registers.

B.2 DMA read by host from device

- Prior to the command being issued to the device, the host driver software programs the host adapter's DMA controller with the memory address pointer(s) and the transfer direction, and arms the DMA controller (enables the "run" flag).
- The host driver software issues the command to the device by writing the Shadow Register Block Registers (command register last).
- In response to the command Shadow Register Block Register being written, the host adapter sets the BSY bit in the status shadow register and transmits a Register – Host to Device FIS to the device with the Shadow Register Block contents.
- When the device has processed the command and is ready, it transmits the read data to the host in the form of one or more Data FISes. This transfer proceeds in response to flow control signals/readiness.
- The host adapter recognizes that the incoming frame is a Data FIS and the DMA controller is programmed, and directs the incoming data to the host adapter's DMA controller which forwards the incoming data to the appropriate host memory locations.
- Upon completion of the transfer, the device transmits a Register – Device to Host FIS to indicate ending status for the command, clearing the BSY bit in the Status register, and if the interrupt flag is set in the header an interrupt is asserted to the host.

In some error conditions there may be no Data FIS transmitted prior to the Register FIS being transmitted with the status information. If so, the host software driver shall abort the setup of the DMA controller.

B.3 DMA write by host to device

- Prior to the command being issued to the device, the host driver software programs the host adapter's DMA controller with the memory address pointer(s) and the transfer direction, and arms the DMA controller (enables the "run" flag). As a result the DMA controller becomes armed but remains paused pending a signal from the device to proceed with the data transfer.
- The host driver software issues the command to the device by writing the Shadow Register Block Registers (command register last).
- In response to the command Shadow Register Block Register being written, the host adapter sets the BSY bit in the status shadow register and transmits a Register – Host to Device FIS to the device with the Shadow Register Block contents.
- When the device is ready to receive the data from the host, the device transmits a DMA Activate FIS to the host, which activates the armed DMA controller. The DMA controller transmits the write data to the device in the form of one or more Data FIS. If more than one data FIS is required to complete the overall data transfer request, a DMA Activate FIS shall be sent prior to each and every one of the subsequent Data FISes. The amount of data

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 555 of 587

transmitted to the device is determined by the transfer count programmed into the host adapter's DMA controller by the host driver software during the command setup phase.
- Upon completion of the transfer, the device transmits a Register – Device to Host FIS to indicate ending status for the command, clearing the BSY bit in the Status register, and if the interrupt flag is set in the header an interrupt is asserted to the host.

In some error conditions the device may signal an ending status by transmitting a register frame to the host without having transmitted a DMA Activate FIS. In such cases the host driver software should abort and clean up the DMA controller.

B.4 PIO data read from the device

- The host driver software issues a PIO read command to the device by writing the Shadow Register Block Registers (command register last).
- In response to the command register being written, the host adapter sets the BSY bit in the status shadow register and transmits a Register – Host to Device FIS to the device with the Shadow Register Block contents.
- When the device has processed the command and is ready to begin transferring data to the host, it first transmits a PIO Setup FIS to the host. Upon receiving the PIO Setup FIS, the host adapter holds the FIS contents in a temporary holding buffer.
- The device follows the PIO Setup FIS with a Data – Device to Host FIS. Upon receiving the Data FIS while holding the PIO Setup FIS, the host adapter transfers the register contents from the PIO Setup FIS into the shadow registers including the initial status value, resulting in DRQ getting set and BSY getting cleared in the Status register. Also, if the interrupt flag is set, an interrupt is generated to the host.
- The host controller receives the incoming data that is part of the Data FIS into a speed matching FIFO that is conceptually attached to the Shadow Register data Block Register.
- As a result of the issued interrupt and DRQ being set in the Status register, host software does a REP INSW on the data register and pulls data from the head of the speed matching FIFO while the serial link is adding data to the tail of the FIFO. The flow control scheme handles data throttling to avoid underflow/overflow of the receive speed matching FIFO that feeds the data Shadow Register Block Register.
- When the number of words read by host software from the Data shadow register reaches the value indicated in the PIO Setup FIS, the host transfers the ending status value from the earlier PIO Setup FIS into the status shadow register resulting in DRQ being cleared and the ending status reported.
- If there are more data blocks to be transferred, the ending status will indicate BSY, and the process repeats from the device sending the PIO Setup FIS to the host.

B.5 PIO data write to the device

- The host driver software issues a PIO write command to the device by writing the Shadow Register Block Registers (command register last).
- In response to the command register being written, the host adapter sets the BSY bit in the status shadow register and transmits a Register – Host to Device FIS to the device with the Shadow Register Block contents.
- When the device is ready to receive the PIO write data, it transmits a PIO Setup FIS to the host to indicate that the target is ready to receive PIO data and the number of words of data that are to be transferred.
- In response to a PIO Setup FIS with the D bit indicating a write to the device, the host transfers the beginning Status register contents from the PIO Setup FIS into the shadow Status register, resulting in DRQ getting set, BSY cleared. Also, if the interrupt flag is set, an interrupt is generated to the host.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 556 of 587

- As a result of DRQ being set in the shadow Status register, the host driver software starts a REP OUTSW to the data register.
- The data written to the data register is placed in an outbound speed matching FIFO and is transmitted to the device as a Data – Host to Device FIS. The REP OUTSW pushes data onto the tail of the FIFO and the serial link pulls data from the head. The flow control scheme handles data throttling to avoid underflow of the transmit FIFO.
- When the number of words indicated in the PIO Setup FIS have been written to the transmit FIFO, the host controller transfers the final status value indicated in the PIO Setup frame into the shadow Status register resulting in DRQ being cleared, and closes the frame with a CRC and EOF_P. If additional sectors of data are to be transferred, the ending status value transferred to the status shadow register would have the BSY bit set and the state is the same as immediately after the command was first issued to the device.
- If there are more data blocks to be transferred, the ending status will indicate BSY, and the process repeats from the device sending the PIO Setup FIS to the host.
- When the number of sectors indicated in the Sector Count register have been transferred, the device shall send a Register – Device to Host FIS with the command complete interrupt and not BSY status.
- In the case of a write error, the device may, on any sector boundary include error status and a command complete interrupt in the PIO setup FIS, and there is no need to send the Register – Device to Host FIS.

B.6 ATA Tagged Command Queuing DMA read from device

NOTE – Serial ATA devices may choose to not implement ATA Tagged Command Queuing in favor of Native Command Queuing.

- Prior to the command being issued to the device, the host driver software programs the host-side DMA controller with the memory address pointer(s) and the transfer direction, and arms the DMA controller (enables the “run” flag).
- The host driver software issues the command to the device by writing the Shadow Register Block Registers (command register last).
- In response to the command Shadow Register Block Register being written, the host adapter sets the BSY bit in the status shadow register and transmits a Register – Host to Device FIS to the device with the Shadow Register Block contents.
- When the device has queued the command and wishes to release the bus, it transmits a Register FIS to the host resulting in the BSY bit being cleared and the REL bit being set in the Status register.
- When the device is ready to complete the transfer for the queued command, it transmits a Set Device Bits FIS to the host resulting in the SERV bit being set in the Status register. If no other command is active (i.e. BSY set to one), then an interrupt is also generated.
- In response to the service request, the host software deactivates the DMA controller (if activated) and issues a SERVICE command to the device by writing the Shadow Register Block Registers, resulting in the BSY bit getting set and a Register FIS being transmitted to the device.
- In response to the SERVICE request, the device transmits a Register FIS to the host conveying the TAG value to the host and clearing the BSY bit and settings the DRQ bit.
- When the DRQ bit is set, the host software reads the TAG value from the Shadow Register Block and restores the DMA controller context appropriate for the command that is completing.
- The device transmits the read data to the host in the form of one or more Data FIS. This transfer proceeds in response to flow control signals/readiness. Any DMA data arriving before the DMA controller has its context restored backs up into the inbound speed-matching FIFO until the FIFO is filled and thereafter is flow controlled to throttle the incoming data until the DMA controller has its context restored by host software.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 557 of 587

- The host controller recognizes that the incoming packet is a Data FIS and the DMA controller is programmed, and directs the incoming data to the host controller's DMA controller that forwards the incoming data to the appropriate host memory locations.
- Upon completion of the transfer, the target transmits a Register – Device to Host FIS to indicate ending status for the command, clearing the BSY bit in the Status register, and if the interrupt flag is set in the header an interrupt is asserted to the host.

B.7 ATA Tagged Command Queuing DMA write to device

NOTE – Serial ATA devices may choose to not implement ATA Tagged Command Queuing in favor of Native Command Queuing.

- Prior to the command being issued to the device, the host driver software programs the host-side DMA controller with the memory address pointer(s) and the transfer direction, and arms the DMA controller (enables the "run" flag).
- The host driver software issues the command to the device by writing the Shadow Register Block Registers (command register last).
- In response to the Shadow Register command Block Register being written, the host adapter sets the BSY bit in the status shadow register and transmits a Register – Host to Device FIS to the device with the Shadow Register Block contents.
- When the device has queued the command and wishes to release the bus, it transmits a Register FIS to the host resulting in the BSY bit being cleared and the REL bit being set in the Status register.
- When the device is ready to complete the transfer for the queued command, it transmits a Set Device Bits FIS to the host resulting in the SERV bit being set in the Status register. If no other command is active (i.e. BSY set to one), then an interrupt is also generated.
- In response to the service request, the host software deactivates the DMA controller (if activated) and issues a SERVICE command to the device by writing the Shadow Register Block Registers, resulting in the BSYT bit getting set and a Register FIS being transmitted to the device.
- In response to the SERVICE request, the device transmits a Register FIS to the host conveying the TAG value to the host and clearing the BSY bit and setting the DRQ bit.
- When the DRQ bit is set, the host software reads the TAG value from the Shadow Register Block and restores the DMA controller context appropriate for the command that is completing.
- When the device is ready to receive the data from the host, the device transmits a DMA Activate FIS to the host, which activates the armed DMA controller. The DMA controller transmits the write data to the device in the form of one or more Data – Host to Device FISes. If more than one data FIS is required to complete the overall data transfer request, a DMA Activate FIS shall be sent prior to each and every one of the subsequent Data FISes. The amount of data transmitted to the device is determined by the transfer count programmed into the host's DMA controller by the host driver software during the command setup phase. If the DMA Activate FIS arrives at the host prior to host software restoring the DMA context, the DMA Activate FIS results in the DMA controller starting the transfer as soon as the host software completes programming it (i.e. the controller is already activated, and the transfer starts as soon as the context is restored).
- Upon completion of the transfer, the target transmits a Register – Host to Device FIS to indicate ending status for the command, clearing the BSY bit in the Status register, and if the interrupt flag is set in the header an interrupt is asserted to the host.

B.8 ATAPI Packet commands with PIO data in

- The host driver software issues a PACKET command to the device by writing the Shadow Register Block Registers (command register last).

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 558 of 587

- In response to the command register being written, the host adapter sets the BSY bit in the status shadow register and transmits a Register – Host to Device FIS to the device with the Shadow Register Block contents.
- When the device is ready to receive the ATAPI command packet, it transmits a PIO Setup – Device to Host FIS to the host to indicate that the target is ready to receive PIO data and the number of words of data that are to be transferred.
- The host transfers the beginning Status register contents from the PIO Setup FIS into the shadow Status register, resulting in BSY getting deasserted and DRQ getting asserted.
- As a result of BSY getting cleared and DRQ being set in the shadow Status register, the host driver software writes the command packet to the Shadow Register Block data register.
- The data written to the data register is placed in an outbound speed matching FIFO and is transmitted to the device as a Data – Host to Device FIS. The writes to the data register push data onto the tail of the FIFO and the serial link pulls data from the head. The flow control scheme handles data throttling to avoid underflow of the transmit FIFO.
- When the number of words indicated in the PIO Setup FIS have been written to the transmit FIFO, the host controller transfers the final status value indicated in the PIO setup frame into the shadow Status register resulting in DRQ being cleared and BSY being set, and closes the frame with a CRC and EOF_P. This completes the transmission of the command packet to the device.
- When the device has processed the command and is ready to begin transferring data to the host, it first transmits a PIO Setup – Device to Host FIS to the host. Upon receiving the PIO Setup – Device to Host FIS, the host adapter holds the FIS contents in a temporary holding buffer.
- The device follows the PIO Setup – Device to Host FIS with a Data – Device to Host FIS. Upon receiving the Data FIS while holding the PIO Setup FIS context, the host adapter transfers the register contents from the PIO Setup FIS into the shadow registers including the initial status value, resulting in DRQ getting set and BSY getting cleared in the Status register. Also, if the interrupt flag is set, an interrupt is generated to the host.
- The host controller receives the incoming data that is part of the Data FIS into a speed matching FIFO that is conceptually attached to the data Shadow Register Block Register.
- As a result of the issued interrupt and DRQ being set in the Status register, host software reads the byte count and does a REP INSW on the data register to pull data from the head of the speed matching FIFO while the serial link is adding data to the tail of the FIFO. The flow control scheme handles data throttling to avoid underflow/overflow of the receive speed matching FIFO that feeds the data Shadow Register Block Register.
- When the number of words received in the Data FIS reaches the value indicated in the PIO Setup FIS and the host FIFO is empty, the host transfers the ending status value from the earlier PIO Setup into the status shadow register resulting in DRQ being cleared and, BSY being set.
- The device transmits final ending status by sending a Register FIS with BSY cleared to zero and the ending status for the command and the interrupt flag set.
- The host detects an incoming register frame that contains BSY cleared to zero and ending status for the command and the interrupt flag set and places the frame content into the shadow registers to complete the command.

B.9 ATAPI Packet commands with PIO data out

- The host driver software issues a PACKET command to the device by writing the Shadow Register Block Registers (command register last).
- In response to the command register being written, the host adapter sets the BSY bit in the status shadow register and transmits a Register – Host to Device FIS to the device with the Shadow Register Block contents.
- When the device is ready to receive the ATAPI command packet, it transmits a PIO Setup – Device to Host FIS to the host to indicate that the target is ready to receive PIO data and the number of words of data that are to be transferred.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 559 of 587

- The host transfers the beginning Status register contents from the PIO Setup – Device to Host FIS into the shadow Status register, resulting in BSY getting deasserted and DRQ getting asserted.
- As a result of BSY getting cleared and DRQ being set in the shadow Status register, the host driver software writes the command packet to the Shadow Register Block data register.
- The data written to the data register is placed in an outbound speed matching FIFO and is transmitted to the device as a Data – Host to Device FIS. The writes to the data register push data onto the tail of the FIFO and the serial link pulls data from the head. The flow control scheme handles data throttling to avoid underflow of the transmit FIFO.
- When the number of words indicated in the PIO Setup FIS have been written to the transmit FIFO, the host controller transfers the final status value indicated in the PIO Setup frame into the shadow Status register resulting in DRQ being cleared and BSY being set, and closes the frame with a CRC and EOF_P. This completes the transmission of the command packet to the device.
- When the device has processed the command and is ready to receive the PIO write data, it transmits a PIO Setup – Device to Host FIS to the host to indicate that the target is ready to receive PIO data and the number of words of data that are to be transferred.
- In response to a PIO Setup FIS with the D bit indicating a write to the device, the host transfers the beginning Status register contents from the PIO Setup FIS into the shadow Status register, resulting in DRQ getting set. Also, if the interrupt flag is set, an interrupt is generated to the host.
- As a result of DRQ being set in the shadow Status register, the host driver software reads the byte count and starts a REP OUTSW to the data register.
- The data written to the data register is placed in an outbound speed matching FIFO and is transmitted to the device as a Data – host to device FIS. The REP OUTSW pushes data onto the tail of the FIFO and the serial link pulls data from the head. The flow control scheme handles data throttling to avoid underflow of the transmit FIFO.
- When the number of words indicated in the PIO Setup FIS have been written to the transmit FIFO, the host controller transfers the final status value indicated in the PIO Setup FIS into the shadow Status register resulting in DRQ being cleared, BSY being set, and closes the frame with a CRC and EOF_P.
- The device transmits final ending status by sending a Register – Device to Host FIS with BSY cleared to zero and the ending status for the command and the interrupt flag set.
- The host detects an incoming Register – Device to Host FIS that contains BSY cleared to zero and ending status for the command and the interrupt flag set and places the frame content into the shadow registers to complete the command.

B.10 ATAPI Packet commands with DMA data in

- Prior to the command being issued to the device, the host driver software programs the host-side DMA controller with the memory address pointer(s) and the transfer direction, and arms the DMA controller (enables the “run” flag).
- The host driver software issues a PACKET command to the device by writing the Shadow Register Block Registers (command register last).
- In response to the command register being written, the host adapter sets the BSY bit in the status shadow register and transmits a Register – Host to Device FIS to the device with the Shadow Register Block contents.
- When the device is ready to receive the ATAPI command packet, it transmits a PIO Setup – Device to Host FIS to the host to indicate that the target is ready to receive PIO data and the number of words of data that are to be transferred.
- The host transfers the beginning Status register contents from the PIO Setup FIS into the shadow Status register, resulting in BSY getting deasserted and DRQ getting asserted.
- As a result of BSY getting cleared and DRQ being set in the shadow Status register, the host driver software writes the command packet to the Shadow Register Block data register.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 560 of 587

- The data written to the data register is placed in an outbound speed matching FIFO and is transmitted to the device as a Data – Host to Device FIS. The writes to the data register push data onto the tail of the FIFO and the serial link pulls data from the head. The flow control scheme handles data throttling to avoid underflow of the transmit FIFO.
- When the number of words indicated in the PIO Setup FIS have been written to the transmit FIFO, the host controller transfers the final status value indicated in the PIO setup frame into the shadow Status register resulting in DRQ being cleared and BSY being set, and closes the frame with a CRC and EOF_P. This completes the transmission of the command packet to the device.
- When the device has processed the command and is ready, it transmits the read data to the host in the form of a single Data FIS. This transfer proceeds in response to flow control signals/readiness.
- The host controller recognizes that the incoming packet is a Data FIS and the DMA controller is programmed, and directs the incoming data to the host controller's DMA controller that forwards the incoming data to the appropriate host memory locations.
- Upon completion of the transfer, the target transmits a Register – Device to Host FIS to indicate ending status for the command, clearing the BSY bit in the Status register, and if the interrupt flag is set in the header an interrupt is asserted to the host.

B.11 ATAPI Packet commands with DMA data out

- Prior to the command being issued to the device, the host driver software programs the host-side DMA controller with the memory address pointer(s) and the transfer direction, and arms the DMA controller (enables the "run" flag). As a result the DMA controller becomes armed but remains paused pending a signal from the device to proceed with the data transfer.
- The host driver software issues a PACKET command to the device by writing the Shadow Register Block Registers (command register last).
- In response to the command register being written, the host adapter sets the BSY bit in the status shadow register and transmits a Register – Host to Device FIS to the device with the Shadow Register Block contents.
- When the device is ready to receive the ATAPI command packet, it transmits a PIO Setup – Device to Host FIS to the host to indicate that the target is ready to receive PIO data and the number of words of data that are to be transferred.
- The host transfers the beginning Status register contents from the PIO Setup FIS into the shadow Status register, resulting in BSY getting deasserted and DRQ getting asserted.
- As a result of BSY getting cleared and DRQ being set in the shadow Status register, the host driver software writes the command packet to the Shadow Register Block data register.
- The data written to the data register is placed in an outbound speed matching FIFO and is transmitted to the device as a Data – Host to Device FIS. The writes to the data register push data onto the tail of the FIFO and the serial link pulls data from the head. The flow control scheme handles data throttling to avoid underflow of the transmit FIFO.
- When the number of words indicated in the PIO Setup FIS have been written to the transmit FIFO, the host controller transfers the final status value indicated in the PIO Setup frame into the shadow Status register resulting in DRQ being cleared and BSY being set, and closes the frame with a CRC and EOF_P. This completes the transmission of the command packet to the device.
- When the device is ready to receive the data from the host, the device transmits a DMA Activate FIS to the host, which activates the armed DMA controller. The DMA controller transmits the write data to the device in the form of one or more Data FIS. The transfer proceeds in response to flow control signals/readiness. The amount of data transmitted to the device is determined by the transfer count programmed into the host's DMA engine by the host driver software during the command setup phase.
- Upon completion of the transfer, the device transmits a Register – Device to Host FIS to indicate ending status for the command, clearing the BSY bit in the Status register, and if the interrupt flag is set in the header an interrupt is asserted to the host.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 561 of 587

B.12 Odd word count considerations

This section outlines special considerations required to accommodate data transfers of an odd number of 16-bit word quantities. The considerations are separately outlined for each of the data transaction types. No accommodation in Serial ATA is made for the transfer of an odd number of 8-bit byte quantities.

B.12.1 DMA read from target for odd word count

- Prior to the command being issued to the device, the host driver software programs the host-side DMA controller with the memory address pointer(s) and the transfer direction, and arms the DMA controller (enables the "run" flag). The count for the DMA transfer is an odd number of word (16-bit) quantities.
- When the device has processed the corresponding command and is ready to transmit the data to the host, it does so in the form of one or more data FIS. Because the transfer count is odd, the last 32-bit Dword transmitted to the host has the high order 16 bits padded with zeroes. The CRC value transmitted at the end of the FIS is computed over the entire FIS including any pad bytes in the final transmitted Dword.
- The host controller receives the incoming data and the DMA controller directs the received data from the receive FIFO to the appropriate host memory locations. The DMA controller has a transfer granularity of a 16-bit word (consistent with the ATA/ATAPI Host Adapters standard).
- Upon receiving the final 32-bit Dword of receive data, the DMA controller transfers the first half (low order 16 bits) to the corresponding final memory location at which point the DMA engine's transfer count is exhausted. The DMA controller drops the high-order 16 bits of the final received Dword since it represents data received beyond the end of the requested DMA transfer. The dropped 16 high order bits corresponds with the 16 bits of transmission pad inserted by the sender.

B.12.2 DMA write by host to target for odd word count

- Prior to the command being issued to the device, the host driver software programs the host-side DMA controller with the memory address pointer(s) and the transfer direction, and arms the DMA controller (enables the "run" flag). The count for the DMA transfer is an odd number of word (16-bit) quantities.
- When the device has processed the corresponding command and is ready to receive the data from the host, it signals readiness with a DMA Activate FIS.
- Upon receiving the DMA Activate signal, the host transmits the data to the device in the form of one or more data FIS. Because the transfer count is odd, the DMA controller completes its data transfer from host memory to the transmit FIFO after filling only the low order 16-bits of the last Dword in the FIFO, leaving the upper 16 bits zeroed. This padded final Dword is transmitted as the final Dword in the data frame. The CRC value transmitted at the end of the FIS is computed over the entire FIS including any pad bytes in the final transmitted Dword.
- Having awareness of the command set and having decoded the current command, the device that receives the transmitted data has knowledge of the expected data transfer length. Upon receiving the data from the host, the device removes the 16-bit pad data in the upper 16 bits of the final 32-bit Dword of received data.

B.13 PIO data read from the device

- In response to decoding and processing a PIO read command with a transfer count for an odd number of 16-bit words, the device transmits the corresponding data to the host in the form of a single Data FIS. The device pads the upper 16-bits of the final 32-bit Dword of the last transmitted FIS in order to close the FIS. The CRC value transmitted at the end of the FIS is computed over the entire FIS including any pad bytes in the final transmitted Dword.
- Host driver software responsible for retrieving the PIO data is aware of the number of words of data it expects to retrieve from the Shadow Command Block Register Data register and performs a REP INSW operation for an odd number of repetitions.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 563 of 587

- Upon exhaustion of the REP INSW operation by the host driver software, the receive FIFO that interfaces with the data register has one 16-bit word of received data remaining in it that corresponds to the pad that the device included at the end of the transmitted frame. This remaining word of data left in the data register FIFO is flushed upon the next write of the Shadow Command Block Register Command register or upon the receipt of the next data FIS from the device.

B.14 PIO data write to the device

- In response to decoding and processing a PIO write command with a transfer count for an odd number of 16-bit words, the device transmits a PIO Setup FIS to the host indicating it is ready to receive the PIO data and indicating the transfer count. The conveyed transfer count is for an odd number of 16-bit word quantities.
- Host driver software responsible for transmitting the PIO data is aware of the number of words of data it needs to write to the Shadow Command Block Register Data register and performs a REP OUTSW operation for an odd number of repetitions.
- After the final write by the software driver to the Shadow Command Block Register Data register, the transfer count indicated in the PIO Setup packet is exhausted, which signals the host controller to close the FIS. Since the transfer count was odd, the upper 16 bits of the final 32-bit Dword of data to transmit remains zeroed (pad) and the host controller closes the FIS after transmitting this final padded Dword. The CRC value transmitted at the end of the FIS is computed over the entire FIS including any pad bytes in the final transmitted Dword.
- Having awareness of the command set and having decoded the current command, the device that receives the transmitted data has knowledge of the expected data transfer length. Upon receiving the data from the host, the device removes the 16-bit pad data in the upper 16 bits of the final 32-bit Dword of received data.

B.15 Native Command Queuing Examples (Informative)

The following is an overview of macro operations and their sequencing for two typical Native Command Queuing scenarios. These illustrative sequences presume a host controller DMA implementation equivalent to current mainstream desktop implementations (hence references to PRD tables and other data structures typically referenced for such implementations) and these illustrative sequences are not intended to exclude other possible host controller implementations.

B.15.1 Queued Commands with Out of Order Completion

HOST Actions		***DEVICE Actions***
Host issues Read Command Tag=0 by presetting bit 0 in the SActive register by writing the value 00000001h (…00000001b) to it and transmitting a Register FIS to the device.		
		Device de-asserts BSY by transmitting a Register FIS to the host.
Host issues Read Command Tag=5 (if BSY not yet 0, host needs to wait) by presetting bit 5 in the SActive register by writing the value 00000020h		

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 564 of 587

(…00100000b) to it and transmitting a Register FIS to device. The resultant SActive register value is 21h.		
		Device de-asserts BSY.
		Device sends DMA Setup FIS, DMA Buffer ID=5 (in this example the second issued command is serviced first).
Host loads PRD pointer into DMA engine corresponding to buffer 5.		
		Device sends data for command corresponding to TAG=5.
Host DMA engine directs incoming data into buffer 5.		
		Device sends Set Device Bits FIS with I bit set and with SActive value of 00000020h (…00100000b), indicating that TAG=5 has finished.
Host receives Set Device Bits FIS with SActive field value of 20h and I bit set. Results in bit 5 in SActive register getting cleared yielding a value of 01h in the SActive shadow register and interrupt getting triggered.		
		Device sends DMA Setup FIS, DMA Buffer ID=0
Host loads PRD pointer into DMA engine corresponding to buffer 0. Host software processes the received interrupt. Reads SActive shadow register and determines that bit 5 is de-asserted and retires command with TAG=5		
		Device sends data for command corresponding to TAG=0
Host DMA engine directs incoming data into Buffer 0		
		Device sends Set Device Bits FIS with I bit set and with SActive value of 00000001h (…00000001b), indicating that TAG=0 has finished.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 565 of 587

Host receives Set Device Bits FIS with SActive field value of 01h and I bit set. Results in bit 0 in SActive register getting cleared yielding a value of 00h in SActive shadow register and interrupt getting triggered.		
		Device idle
Host software processes the received interrupt. Reads SActive shadow register and determines that bit 0 is de-asserted and retires command with TAG=0. Host idle.		

B.15.2Interrupt Aggregation

HOST Actions		***DEVICE Actions***
Host issues Read Command Tag=0 by presetting bit 0 in the SActive register by writing the value 00000001h (…00000001b) to it and transmitting a Register FIS to the device.		
		Device de-asserts BSY by transmitting a Register FIS to the host.
Host issues Read Command Tag=5 (if BSY not yet 0, host needs to wait) by presetting bit 5 in the SActive register by writing the value 00000020h (…00100000b) to it and transmitting a Register FIS to device. The resultnant SActive register value is 21h.		
		Device de-asserts BSY.
		Device sends DMA Setup FIS, DMA Buffer ID=5 (in this example the second issued command is serviced first).
Host loads PRD pointer into DMA engine corresponding to buffer 5.		
		Device sends data for command corresponding to TAG=5.
Host DMA engine directs		

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 566 of 587

incoming data into buffer 5.		
		Device sends Set Device Bits FIS with I bit set and with SActive value of 00000020h (…00100000b), indicating that TAG=5 has finished.
Host receives Set Device Bits FIS with SActive field value of 20h and I bit set. Results in bit 5 in SActive register getting cleared yielding a value of 01h in the SActive shadow register and interrupt getting triggered.		
		Device sends DMA Setup FIS, DMA Buffer ID=0
Host loads PRD pointer into DMA engine corresponding to buffer 0		
		Device sends data for command corresponding to TAG=0
Host DMA engine directs incoming data into buffer 0 Host is busy and is slow to respond to and clear the received interrupt. Host interrupt response time is slow relative to device completion rate for this example.		
		Device sends Set Device Bits FIS with I bit set and with SActive value of 00000001h (…00000001b), indicating that TAG=0 has finished.
Host receives Set Device Bits FIS with SActive field value of 01h and I bit set. Results in bit 0 in SActive register getting cleared yielding value of 00h in SActive shadow register and interrupt getting triggered. If the host had not already reset the pending interrupt from the completion of TAG=5, no new interrupt is triggered. If the previous interrupt has already been reset, then a new interrupt is triggered which is the previous example illustrated in section B.15.1		
Host had a long latency, but is now processing the interrupt and has reset the interrupt		

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 567 of 587

pending flag. Host is now doing command completion processing. The second interrupt issued by the device got aggregated because the first interrupt didn't get reset soon enough. Host reads SActive shadow register and sees that TAG=0 and TAG=1 commands have both completed (neither have their SActive bit set). Host processes command completions and retires both commands.		
		Device idle

Appendix C. Device Emulation of nIEN with Interrupt Pending (Informative)

This specification defines the I bit in Register Device to Host FISes as the interrupt pending state of the device, and it is not modified by the state of nIEN in received Host to Device FISes. In this specification, devices ignore the nIEN bit in received Host to Device FISes and always perform as if nIEN is cleared to zero. See sections 10.3.4 and 10.3.5.

Some devices implemented to prior Serial ATA specification revisions used the nIEN bit of the Register Host to Device FIS as a pre-condition to setting the interrupt pending flag (I bit) of the Register Device to Host and Set Device Bits FISes. The purpose of using nIEN to enable the I bit was to emulate the operation of the parallel implementation of ATA. In the parallel implementation, when nIEN is cleared to zero, the driver is enabled for the INTRQ line to the host. When nIEN is set to one, the INTRQ line is put into the high impedance state by the device. This function is typically used in devices that support Device 0 and Device 1 operation (see section 13.1.3), and it is also required for Overlapped operation.

In this specification, the implementation of Device 0 / Device 1 emulation is performed exclusively by the host (see section 13.1.3).

One serious side effect of device emulation of nIEN is the possibility of lost interrupts. In the parallel implementation, a host can disable interrupts, and upon re-enabling interrupts (by clearing nIEN to zero) see the INTRQ line again asserted. If a serial device is performing I-bit masking based on the state of nIEN, a Register Device to Host FIS may be received with I=0 (since it is masked by nIEN). The device, however, may have an interrupt pending at that time. When the host writes the Device Control register with nIEN cleared to zero, it will not see the pending interrupt reflected by the assertion of INTRQ as in the parallel case. There is no way for the device to "re-send" the I=1 condition to the host. In this instance, the host has to resort to a polling operation to resume the operation.

The system designer should be aware of the following:

1) The "I" bit is the interrupt pending flag for Serial ATA devices.

2) The behavior of the "I" bit may be modified by the state of the nIEN bit in devices implemented to prior versions of the Serial ATA specification. Such devices may change the behavior of the "I" bit based on the current state of the nIEN bit, as last written by a Register Host to Device FIS with C=0 or C=1. Some devices do not observe the nIEN bit when C=1. Devices that do not observe the nIEN bit when C=1 have compatibility issues with hosts that only transmit a Control Register FIS when SRST changes state.

3) There is no defined behavior for a device when nIEN bit changes and the modification of the behavior of the "I" bit by nIEN is vendor specific.

4) The host should clear nIEN=0 in all Register Host to Device FISes. This results in the highest compatibility with devices that use the nIEN bit as a pre-condition to setting the interrupt pending flag.

5) If a device supports nIEN emulation, and the nIEN bit is set to one by the host, the host driver should accommodate the case of no interrupt generation when nIEN is cleared to zero and the device has a pending interrupt.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 569 of 587

APPENDIX D. I/O CONTROLLER MODULE (INFORMATIVE)

The purpose of this optional feature is to specify the IO Connectors (shown with heavy outline below) that provide interconnection for power, data and enclosure services between I/O controller(s) and a Serial ATA disk backplane. This specification defines interoperability between I/O modules from one manufacturer which work with backplanes from a different manufacturer. This interoperability includes electrical, mechanical and enclosure management connections. Compliant I/O controllers and Serial ATA backplanes shall adhere to the specified connector placement and connector pinout.

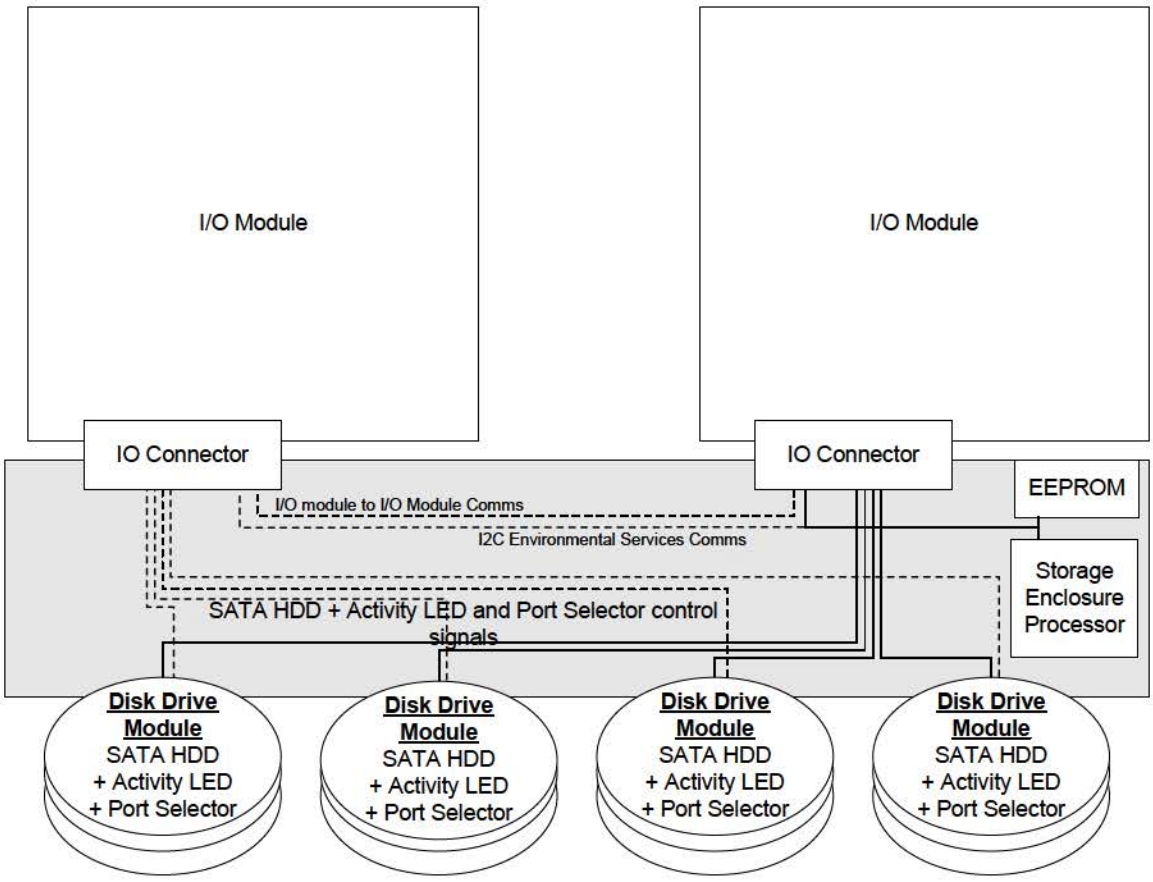


Figure 229 – Concept summary interconnect structure

The I/O connector(s), shown in
Figure 229 with the heavy outline, are the central items being specified. The I/O Module in dotted line format represents a secondary and optional I/O module. The shaded area represents the backplane. The Disk Drive Module is envisioned as a standard Serial ATA HDD within a carrier, including:

- power connection between the module and the backplane
- data connection between the module and the backplane
- HDD activity LED (see section 6.4.1), optional
- Port Selector with a secondary connection to the backplane, optional

D.1 Supported Configurations

Two configuration goals are supported by the connector specification:

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199

1. Single I/O Controller System
2. Dual I/O Controller System with redundancy features (optional)

Each implementation has mandatory and optional features supported through the use of the available signals.

D.1.1 Single I/O Controller Signals

- Serial ATA interface
- I/O controller to enclosure processor (SEP) communication
- Disk activity LED signals.
- I/O controller power and ground
- I/O controller identification and control (RESET)

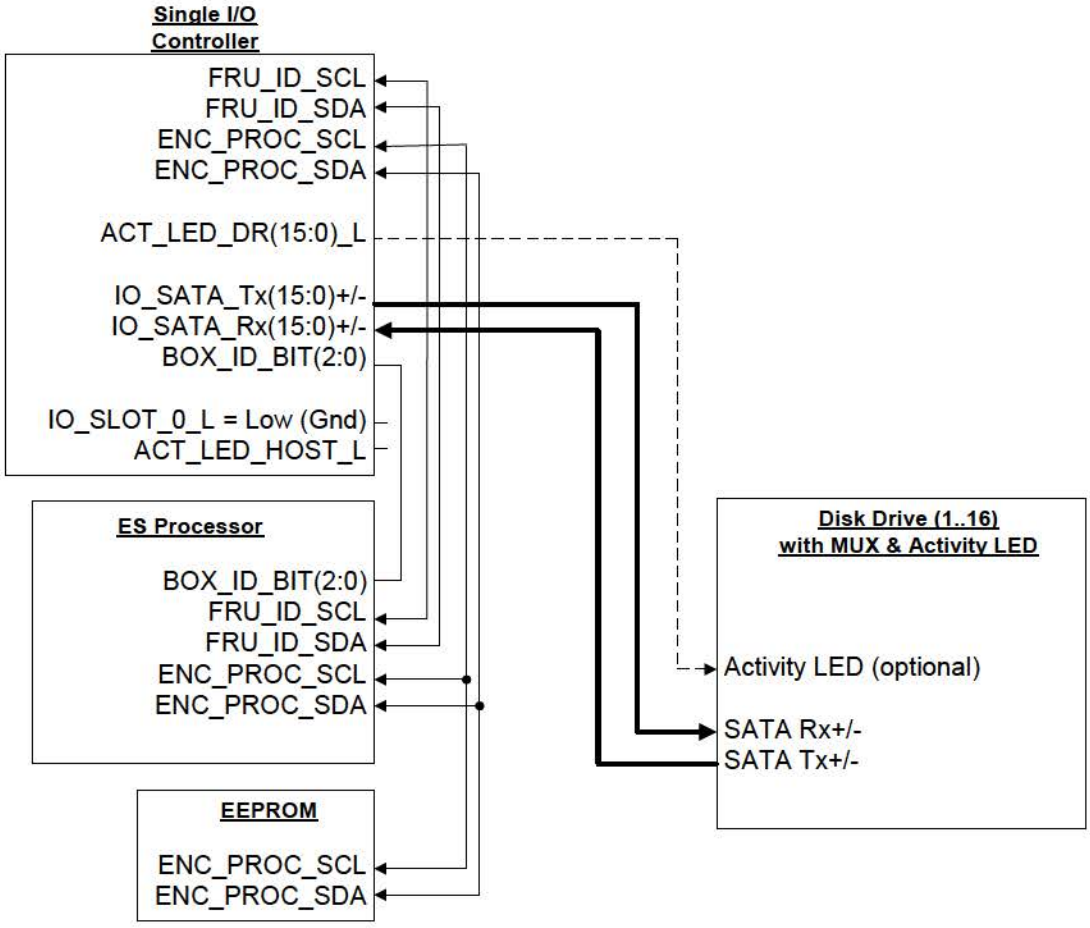


Figure 230 – An example of signal connections with one I/O Controller

D.1.2 Dual I/O Controller Signals

As for single I/O controller and in addition:

- Multiplexer control to provide dual access for I/O controller failover capabilities
- I/O controller to I/O controller communication

D.1.3 Further optional features

- RAID battery backup support
- Additional low and high speed communications for optional board-to-board communications. As this specification is not intended to define board-to-board for boards from different manufacturers; the actual implementation of signaling and protocol is left to the discretion of the I/O controller manufacturer.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 572 of 587

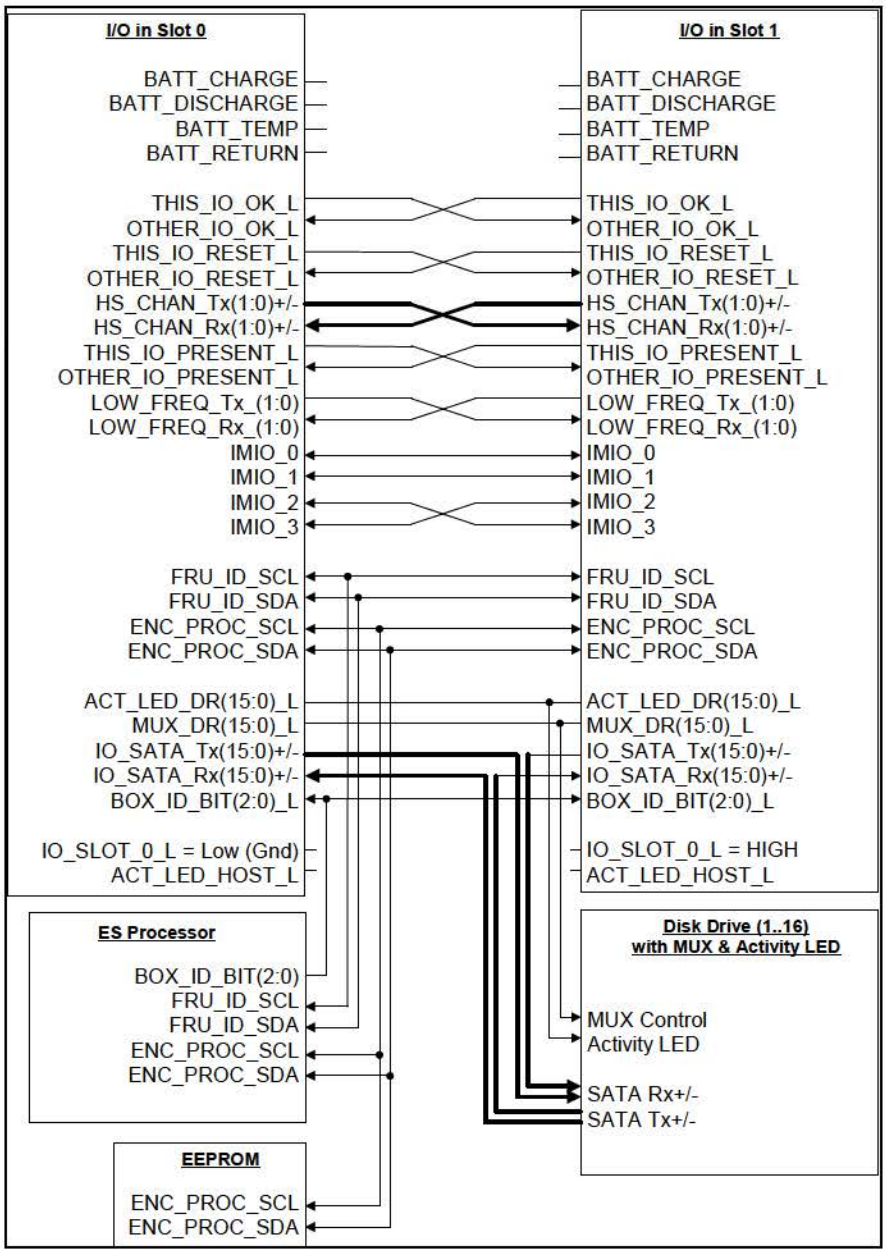


Figure 231 – Example of signal connections with two I/O Modules

D.2 Optional High Speed Channel configurations

The optional High speed communications channels can be used for high speed differential communications between the two I/O controllers, or for I/O controller to host communications such as inclusion into a Fibre Channel loop. It is recommended that these signals should be 100 ohms differential characteristic impedance.

Actual usage is open to the user definition however the normative backplane routing should be either:

- Configuration 0: Both channels link I/O controller to I/O controller
- Configuration 1: One channel links the two I/O Controllers, and the other is router to a host connector.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199

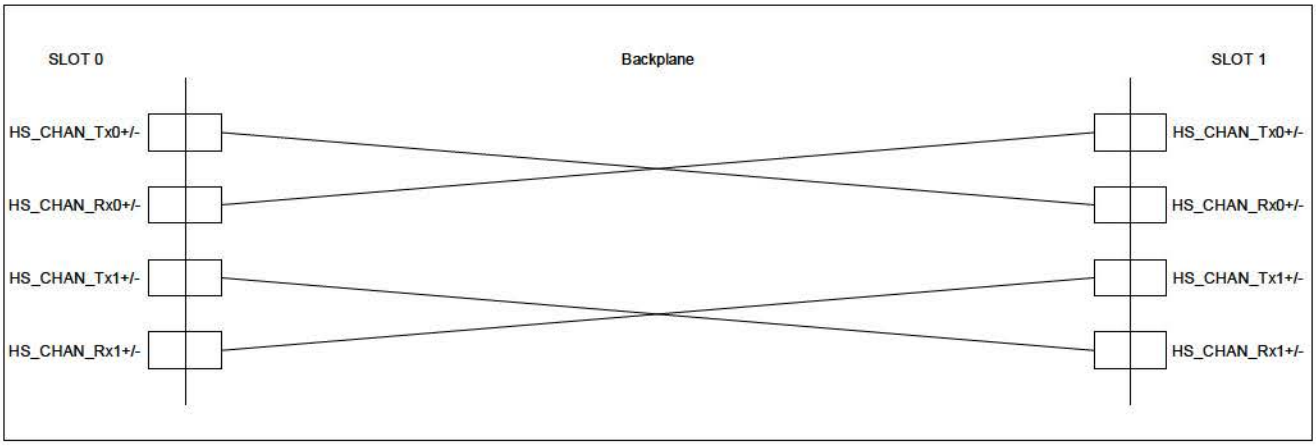


Figure 232 – High Speed Channels – Configuration 0

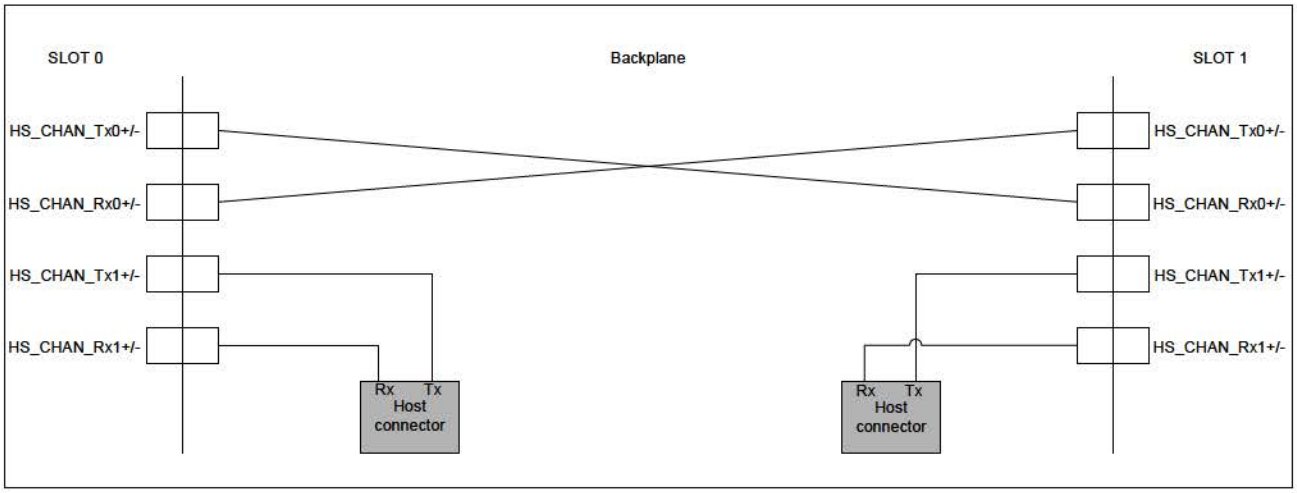


Figure 233 – High-Speed Channels – Configuration 1

D.3 Optional Low Speed Channel configurations

The optional Low speed communications channels can be used for end-to-end I/O controller to I/O controller communications. There is a pair transmit and receive signals for low frequency (<500 kHz) I/O controller to I/O controller communications.

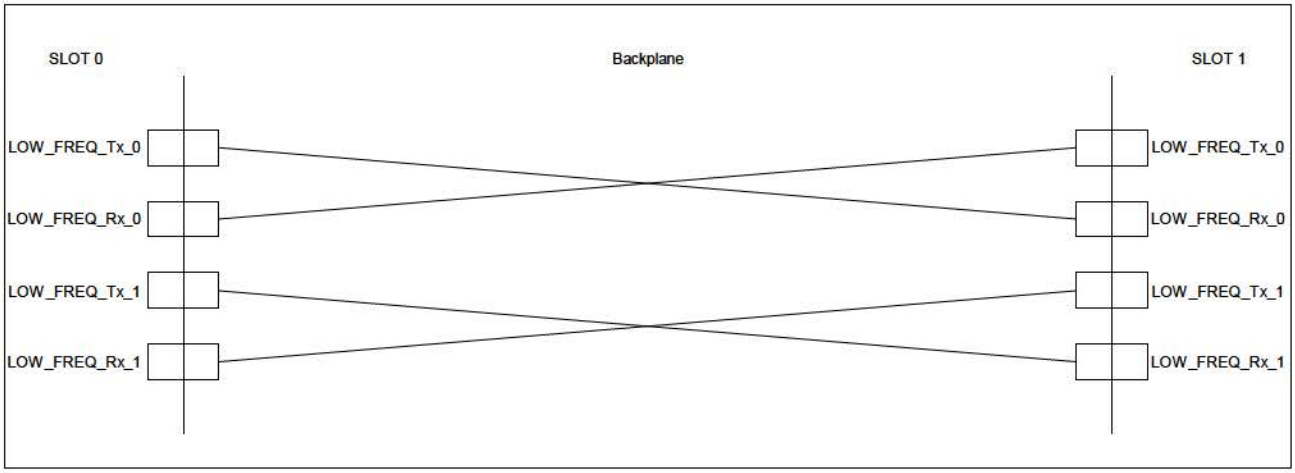


Figure 234 – Low Speed Channels

There are 4 end-to-end signals that can be used for intermodule I/O communications.

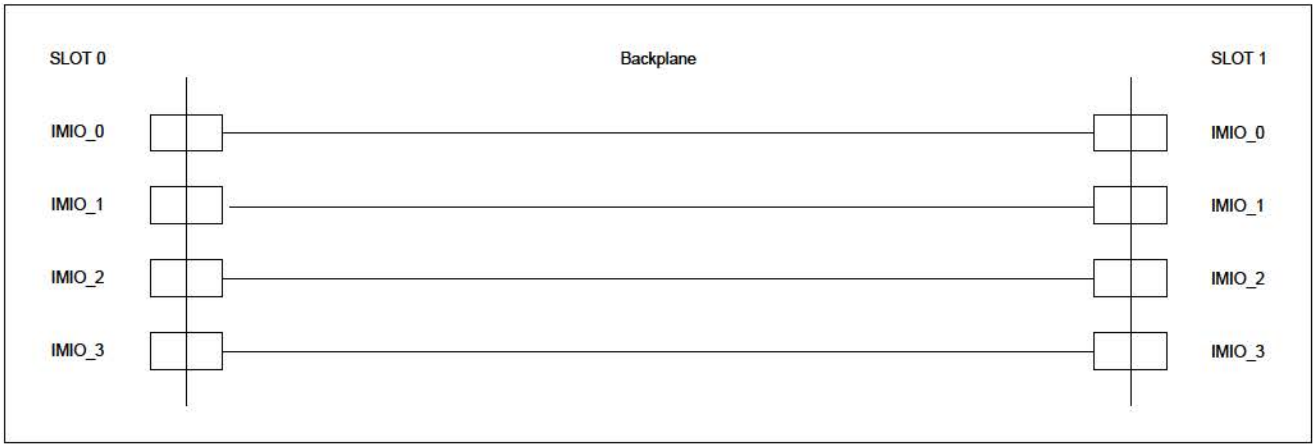


Figure 235 – Interconnect Channels

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 575 of 587

D.4 I/O Controller Module Connectors

The interface connector that shall be used on the backplane is based on a Berg-FCI High Speed Metral 5x6 30-pin male header connector part number 59566-1001 or equivalent. Equivalent parts from ITT Cannon: on the backplane 5-row 4000 male CBC20HS4000-030WXP5-5yy-x-VR; and on the I/O controller 5-row 4000 std-female CBC20HS4000-030FDP5-500-x-VR.

D.4.1 I/O Controller Module Connector

The interface connector that shall be used on the I/O controller is based on a Berg-FCI High Speed Metal 5x6 30-pin right angle female receptacle part number 52057-102. This connector is used for the 6 high-speed connectors.

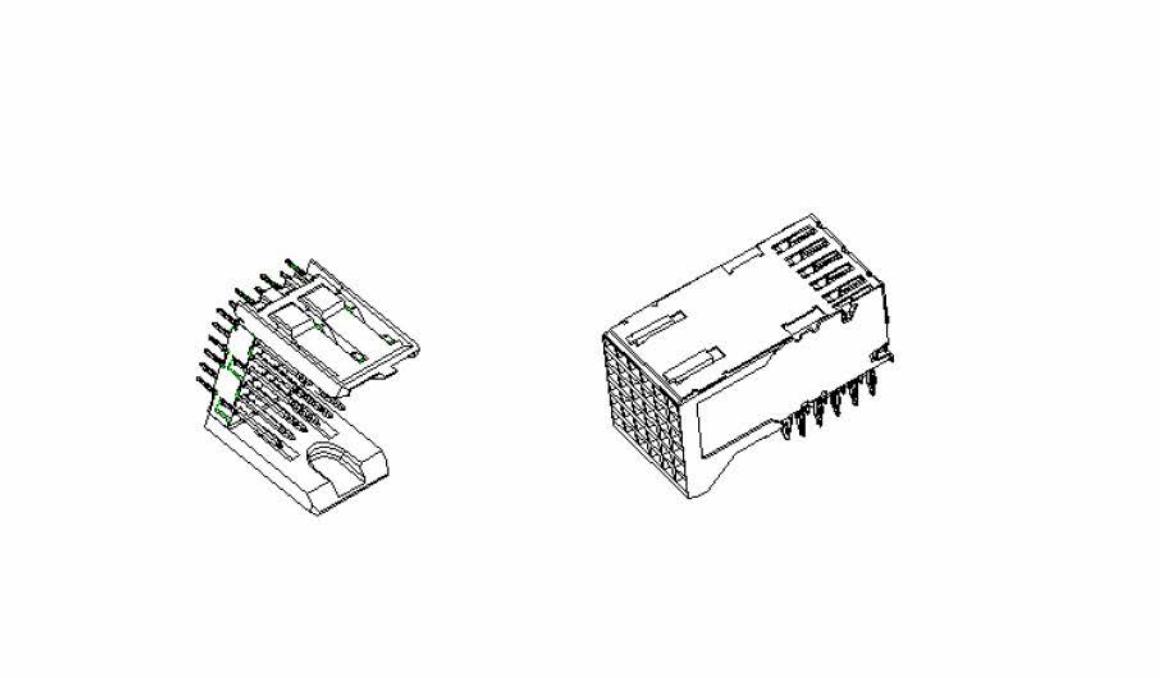


Figure 236 – I/O Controller Module Connector Rendering

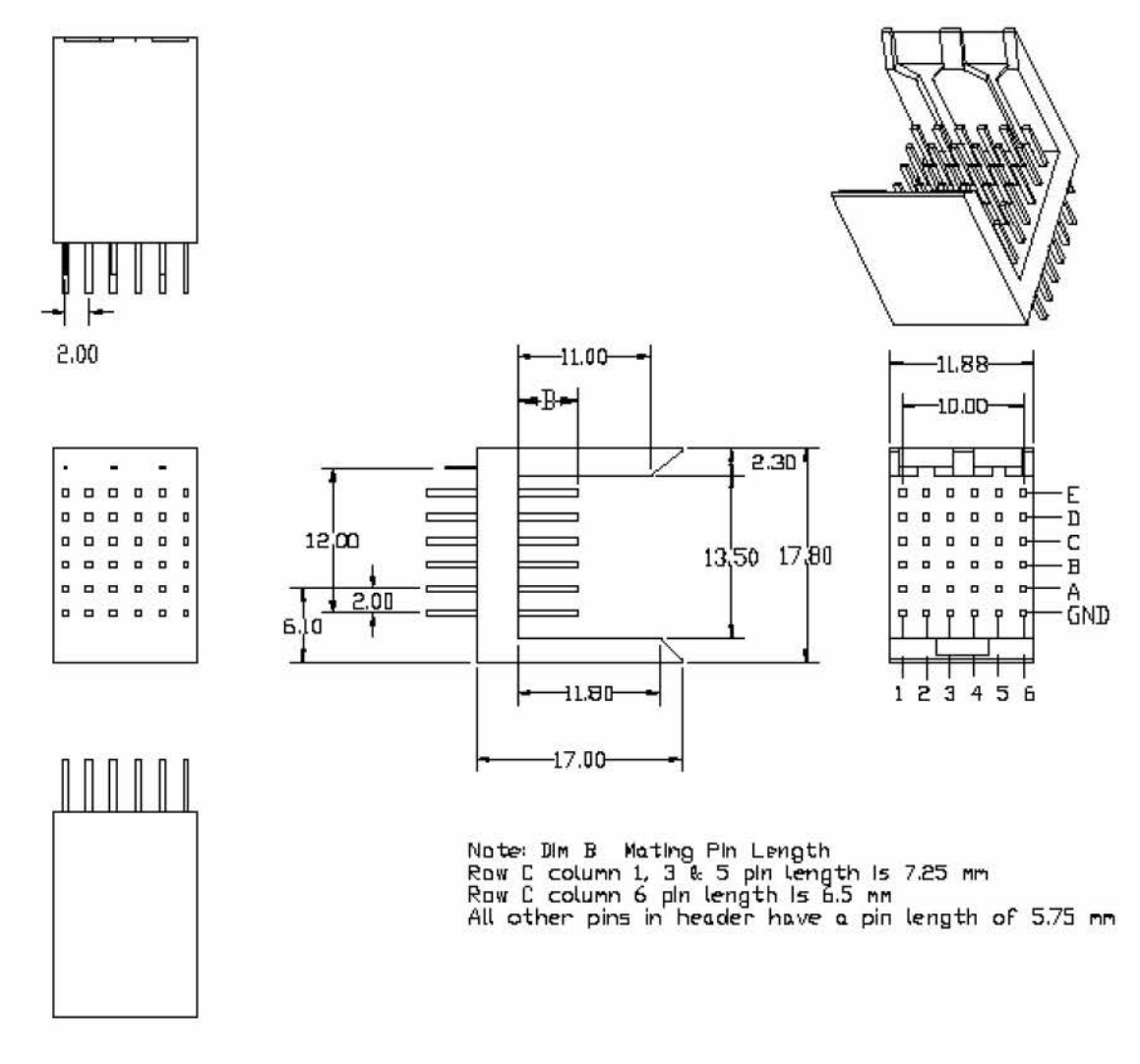


Figure 237 – Connector Pin Layout and Pin Lengths

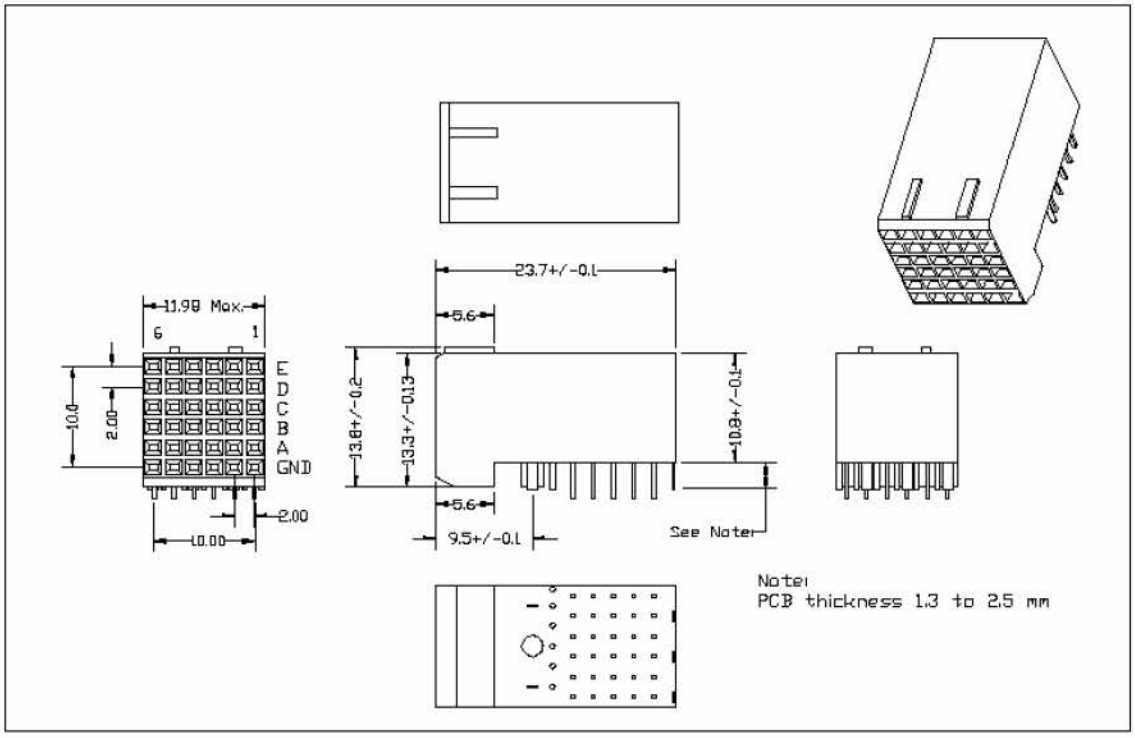


Figure 238 – I/O Controller Module Connector Receptacle Engineering Drawing

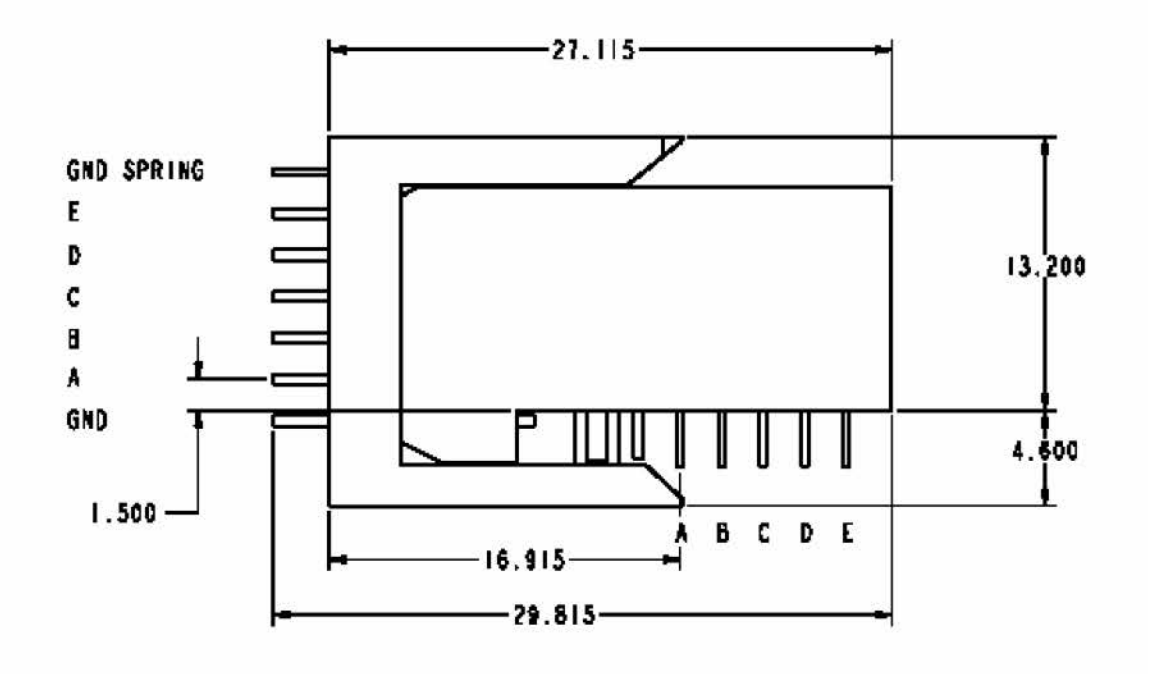


Figure 239 – Side View of Connector

D.5 I/O Controller Module Connector Locations

D.5.1 Purpose

This section defines the mating connector locations and connector alignment between I/O modules from one manufacturer and backplanes from a different manufacturer. This is an optional feature of the Serial ATA specification; however I/O controllers and Serial ATA backplanes in support of this industry normal Serial ATA I/O Controller-to-Backplane interface definition shall adhere to this specified connector placement, and connector pinout.
The width and length dimensions of the I/O Module shown in Figure 240 and Figure 241 are presented as examples and not mandatory requirements.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 579 of 587

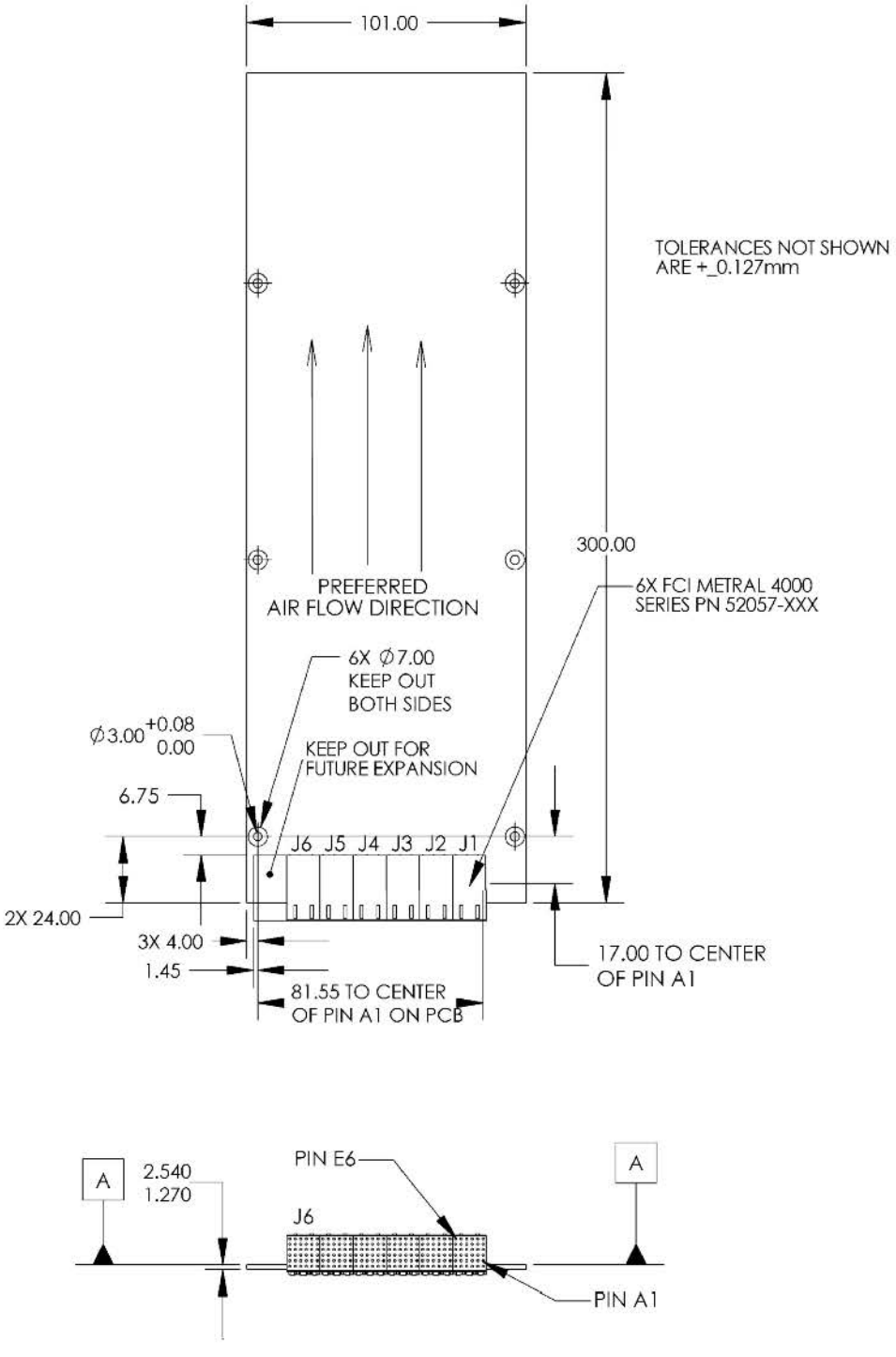


Figure 240 – I/O Controller Module Connector Locations on 1xWide I/O module

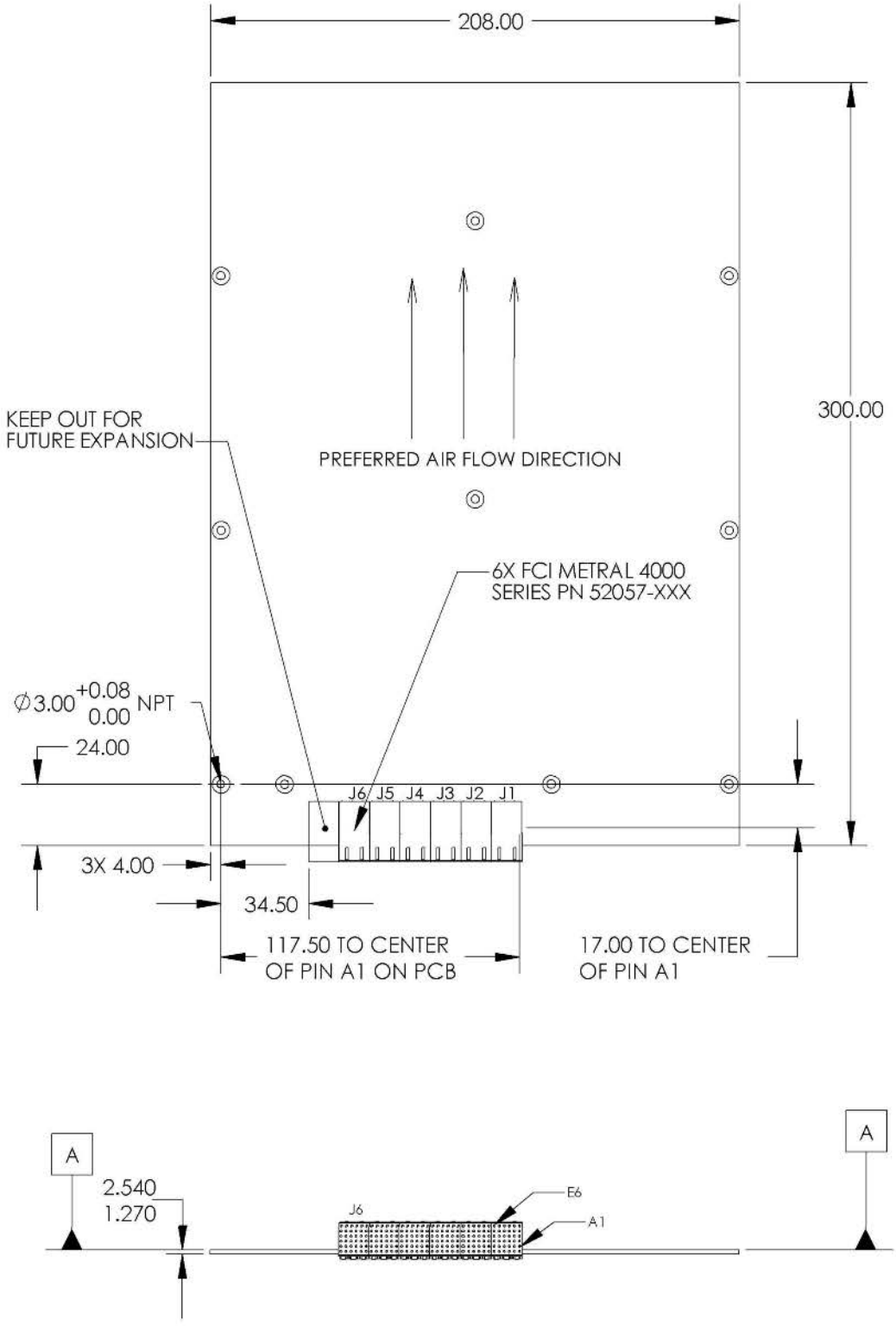


Figure 241 – I/O Controller Module Connector Locations on 2xWide I/O Module

D.6 Pinout Listing

These tables show the pin listing for the Metral 4000 connector system. Note that there are grounded shields between each connector row and signal, which are not shown in this matrix to aid clarity.

Table 46 – J1, J2, J3 Pin Assignments

J1	Row E	Row D	Row C	Row B	Row A	Ground
1	IO_SATA_RX0+	IO_SATA_RX0-		IO_SATA_TX0+	IO_SATA_TX0-	Shield
2	ACT_LED_DR0_L	MUX_DR0_L	5V	5V	IO_SLOT_0_L	Shield
3	IO_SATA_RX1+	IO_SATA_RX1-		IO_SATA_TX1+	IO_SATA_TX1-	Shield
4	ACT_LED_DR1_L	MUX_DR1_L	5V	5V	OTHER_IO_PRESENT_L	Shield
5	IO_SATA_RX2+	IO_SATA_RX2-		IO_SATA_TX2+	IO_SATA_TX2-	Shield
6	ACT_LED_DR2_L	MUX_DR2_L	5V PRECHARGE	ENC_PROC_SDA	ENC_PROC_SCL	Shield
J2	Row E	Row D	Row C	Row B	Row A	Ground
1	IO_SATA_RX3+	IO_SATA_RX3-		IO_SATA_TX3+	IO_SATA_TX3-	Shield
2	ACT_LED_DR3_L	MUX_DR3_L	5V	5V	BOX_ID_BIT_0	Shield
3	IO_SATA_RX4+	IO_SATA_RX4-		IO_SATA_TX4+	IO_SATA_TX4-	Shield
4	ACT_LED_DR4_L	MUX_DR4_L	5V	5V	BOX_ID_BIT_1	Shield
5	IO_SATA_RX5+	IO_SATA_RX5-		IO_SATA_TX5+	IO_SATA_TX5-	Shield
6	ACT_LED_DR5_L	MUX_DR5_L	5V PRECHARGE	FRU_ID_SDA	FRU_ID_SCL	Shield
J3	Row E	Row D	Row C	Row B	Row A	Ground
1	IO_SATA_RX6+	IO_SATA_RX6-		IO_SATA_TX6+	IO_SATA_TX6-	Shield
2	ACT_LED_DR6_L	MUX_DR6_L	5V	5V	BATT_CHARGE	Shield
3	IO_SATA_RX7+	IO_SATA_RX7-		IO_SATA_TX7+	IO_SATA_TX7-	Shield
4	ACT_LED_DR7_L	MUX_DR7_L	5V	THIS_IO_PRESENT_L	BATT_DISCHARGE	Shield
5	IO_SATA_RX8+	IO_SATA_RX8-		IO_SATA_TX8+	IO_SATA_TX8-	Shield
6	ACT_LED_DR8_L	MUX_DR8_L	12V PRECHARGE	12V	12V	Shield

Table 47 – J4, J5, J6 Pin Assignments

J4	Row E	Row D	Row C	Row B	Row A	Ground
1	IO_SATA_RX9+	IO_SATA_RX9-		IO_SATA_TX9+	IO_SATA_TX9-	Shield
2	ACT_LED_DR9_L	MUX_DR9_L	5V	12V	BATT_RETURN	Shield
3	IO_SATA_RX10+	IO_SATA_RX10-		IO_SATA_TX10+	IO_SATA_TX10-	Shield
4	ACT_LED_DR10_L	MUX_DR10_L	5V	THIS_IO_OK_L	BATT_TEMP	Shield
5	IO_SATA_RX11+	IO_SATA_RX11-		IO_SATA_TX11+	IO_SATA_TX11-	Shield
6	ACT_LED_DR11_L	MUX_DR11_L	Reserved (3_3V PRECHARGE)	Reserved (3_3V)	BOX_ID_BIT_2	Shield
J5	Row E	Row D	Row C	Row B	Row A	Ground
1	IO_SATA_RX12+	IO_SATA_RX12-		IO_SATA_TX12+	IO_SATA_TX12-	Shield
2	ACT_LED_DR12_L	MUX_DR12_L	5V	LOW_FREQ_RX_2	LOW_FREQ_TX_2	Shield
3	IO_SATA_RX13+	IO_SATA_RX13-		IO_SATA_TX13+	IO_SATA_TX13-	Shield
4	ACT_LED_DR13_L	MUX_DR13_L	5V	ACT_LED_HOST0	ACT_LED_HOST1	Shield

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 582 of 587

5	IO_SATA_RX14+	IO_SATA_RX14-		IO_SATA_TX14+	IO_SATA_TX14-	Shield
6	ACT_LED_DR14_L	MUX_DR14_L	Reserved (3_3V PRECHARGE)	Reserved	Reserved	Shield
J6	Row E	Row D	Row C	Row B	Row A	Ground
1	IO_SATA_RX15+	IO_SATA_RX15-		IO_SATA_TX15+	IO_SATA_TX15-	Shield
2	ACT_LED_DR15_L	MUX_DR15_L	5V	5V	OTHER_IO_RESET_L	Shield
3	HS_CHAN_TX(0)+	HS_CHAN_TX(0)-		HS_CHAN_RX(0)+	HS_CHAN_RX(0)-	Shield
4	LOW_FREQ_RX_1	LOW_FREQ_TX_1	5V	IMIO_0	IMIO_1	Shield
5	HS_CHAN_TX(1)+	HS_CHAN_TX(1)-		HS_CHAN_RX(1)+	HS_CHAN_RX(1)-	Shield
6	OTHER_IO_OK_L	THIS_IO_RESET_L	5V PRECHARGE	IMIO_2	IMIO_3	Shield

NOTES:

1 Connector as seen from side 1 of backplane. (I.e. Disk Side)

2

3 Medium pins shaded grey - Length 6.50 mm

4 All other pins short - Length 5.75 mm

D.7 Signal Descriptions

ACT_LED_DR(15:0)_L

Signal is used to drive the activity LED associated with each disk drive. This is an active low signal, i.e. the LED is on when the signal is low. The LEDs are pulled up to +5 Vdc. The LED control circuitry shall be capable of sinking 15 mA at 0.4 V steady-state.

This signal should be driven by an open-drain output in order to prevent damage should two I/O modules inadvertently attempt to drive the signal at the same time. During RESET the I/O controller output should be set to a high impedance state.

The number of activity LED signals shall match the number of SATA ports supported by the controller. It is not mandatory to support 16 ports, any number between 1 and 16 SATA ports can be supported by this configuration.

MUX_DR(15:0)_L

Signal is used to control the Serial ATA path through a front-end multiplexer to a disk drive (if implemented). When this signal is low, the multiplexer selects the Serial ATA path to the I/O controller in the option slot 0 identified with IO_Slot_0 connected to GND. This is a standard 3.3 VTTL level signal. This signal should be driven by an open-drain output in order to prevent damage should two I/O modules inadvertently attempt to drive the signal at the same time.

Serial ATA is a point-to-point, controller to disk drive, single initiator interface. Current Serial ATA drives do not support dual Serial ATA ports, therefore, no native capability exists to implement I/O controller failover (Active-Active or Active-Passive). An alternative method to provide a dual I/O controller access to the same disk drive is to implement a front-end multiplexer between the I/O controller and disk drive. This multiplexer allows only one I/O controller to own the complete access to the disk drive and implement an Active-Passive failover.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 583 of 587

This signal shall not be implemented in single I/O controller enclosures, as all Serial ATA signals shall be directly connected to the single slot. The I/O controller needs to ensure it is capable of operating in a single I/O controller configuration without the presence of these signals.

The MUX_DRx_L controls on the I/O controller shall be capable of being reset to an OPEN or high impedance state using the I/O controller RESET signal. This is to allow the failover I/O controller to have direct RESET capability over the MUX_DRx_L controls in the event of the I/O controller failure.

The number of activity MUX_DR signals shall match the number of SATA ports supported by the controller. It is not mandatory to support 16 ports, any number between 1 and 16 SATA ports can be supported by this specification.

IO_SATA_Tx(15:0)±

Differential SATA signal pair that originates at the I/O controller (Tx) and is received by the SATA disk drive (Rx). It is not mandatory to support 16 ports, any number between 1 and 16 SATA ports may be supported by this specification.

These signals shall be 100 Ohms differential characteristic impedance as per the nominal differential impedance documented in Table 20.

IO_SATA_Rx(15:0)±

Differential SATA signal pair that originates at the SATA disk drive (Tx) and is received by the I/O controller (Rx). It is not mandatory to support 16 ports, any number between 1 and 16 SATA ports may be supported by this specification.

These signals shall be 100 Ohms differential characteristic impedance. As per the nominal differential impedance documented in Table 20.

IO_Slot_0_L

Active low signal that identifies the I/O controller location, 0 or 1. This signal shall be pulled low (GND) on the backplane for IO Slot 0 and high (3.3 Vdc through a 10 kOhms resistor) for IO Slot 1.

In a single I/O controller enclosure, this signal shall be connected to GND. In a dual controller configuration, the I/O controller should sense this line to determine in which slot it is located.

Ground

Signal and power ground of the I/O module.

5V PRECHARGE

+5 Vdc power that is available on the extended pins. This is used for pre-charging the I/O module.

The enclosure shall provide for a current limit of 4.5 A peak on each 5V pre-charge pin (R=1.1 Ohms).

5V

+5 Vdc power that is available on standard length pins.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 584 of 587

The enclosure shall be capable to supplying 10 A of 5 V current per I/O module.

3V3 PRECHARGE

+3.3 Vdc power that is available on the extended pin. This is used for pre-charging the I/O controller 3.3 V circuitry. The enclosure shall provide for a current limit of 0.75 A on each 3.3 V pre-charge pin (R=4.4 Ohms).

3V3

+3.3 Vdc power that is available on standard length pins.

The enclosure shall be capable of supplying 0.75 A of 3.3 V current per I/O module.

12V PRECHARGE

+12 Vdc power that is available on the extended pins. This is used for pre-charging the 12V circuitry in the I/O Option slot module.

The enclosure shall be capable of supplying 2.4 A peak on each 12 V pre-charge pin (R=5 Ohms).

12V

+12 Vdc power that is available on standard length pins.

The enclosure shall be capable of supplying 1.0 A of current per I/O module.

FRU_ID_SCL

Clock signal of the FRU Identification 2-wire, serial bus. The backplane has a 1.1 kOhms resistor pulled up to 3.3 V on this signal. The devices on this bus shall have open collector outputs. This 2-wire serial bus is used by the enclosure processor to gather information from all the FRUs located within the enclosure.

FRU_ID_SDA

Data signal of the FRU Identification 2-wire, serial bus. The backplane has a 1.1 kOhms resistor pulled up to 3.3 V on this signal. The devices on this bus shall have open collector outputs. This 2-wire serial bus is used by the enclosure processor to gather information from all the FRUs located within the enclosure.

ENC_PROC_SCL

Clock signal of the SES processor 2-wire, serial bus. The SES processor has a 1.1 kOhms resistor pulled up to 3.3 V on this signal. The devices on this bus shall have open collector outputs. This 2-wire serial bus is used by the I/O controller to talk to the enclosure processor.

ENC_PROC_SDA

Data signal of the SES processor 2-wire, serial bus. The SES processor has a 1.1 kOhms resistor pulled up to 3.3 V on this signal. The devices on this bus shall have open collector outputs. This 2-wire serial bus is used by the I/O controller to talk to the enclosure processor.

HS_CHAN_Tx(1:0)

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 585 of 587

Differential transmit pair that can connect the I/O controller to either the other I/O controller or an I/O connector receive pair. Details of the uses of these signals are provided in Figure 232 and Figure 233. It is recommended that these signals be 100 Ohms differential characteristic impedance.

HS_CHAN_Rx(1:0)

Differential receive pair that can connect the I/O controller to either the other I/O controller or an I/O connector transmit pair. Details of the uses of these signals are provided in Figure 232 and Figure 233. It is recommended that these signals be 100 Ohms differential characteristic impedance.

ACT_LED_HOST_L

Host link activity LED control signal from the I/O controller for front panel mounted LED. Each port shall have a separate LED for the intended use of indication which controller port is active. Exact functionality may be defined by the vendor.

BATT_CHARGE

Signal is used to charge a battery back-up unit (BBU). The backplane shall support a maximum of 2 A on this trace.

BATT_DISCHARGE

Signal is used to discharge a BBU. The backplane supports a maximum of 2 A on this trace.

BATT_TEMP

Signal is used to report an analog voltage level which corresponds to a temperature level within a BBU. The backplane supports a maximum of 100 mA on this trace.

BATT_RETURN

Signal is used as the return (ground) path for a BBU. The backplane supports a maximum of 2 A on this trace.

THIS_IO_OK_L

Active low signal is used to determine if the I/O Option slot module is performing within specification. This signal is driven by the I/O controller. The I/O Option Slot module shall provide a pull-up for this line when it is inactive. The backplane should support a maximum of 100 mA on this trace.

OTHER_IO_OK_L

Active low signal is used to determine if the other I/O Option slot module in a dual controller configuration is performing within specification. This signal is an input to the I/O controller. The backplane should support a maximum of 100 mA on this trace.

OTHER_IO_RESET_L

Active low output signal which is used to reset the other I/O Option slot module in a dual controller system, located in the opposite slot. The backplane shall provide a pull-up for this line

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 586 of 587

when it is inactive. This signal is cross-wired with the THIS_IO_RESET_L signal on dual controller backplanes. The backplane should support a maximum of 100 mA on this trace.

THIS_IO_RESET_L

Active low input signal which is used to reset the I/O Option slot module located in this slot. This signal is cross-wired with the OTHER_IO_RESET_L signal on dual controller backplanes. The backplane should support a maximum of 100 mA on this trace.

LOW_FREQ_RX_(1:0)

Signal line used for intermodule, 2-wire communications. This signal is cross-wired with the LOW_FREQ_TX signal on dual controller backplanes as shown in Figure 234. This is a low-speed signal line. Maximum frequency is 500 kHz. The actual implementation of this signal is to be decided by the I/O controller vendor. The backplane should support a maximum of 100 mA on this trace.

LOW_FREQ_TX_(1:0)

Signal line used for intermodule, 2-wire communications. This signal is cross-wired with the LOW_FREQ_RX signal on dual controller backplanes as shown in Figure 234. This is a low-speed signal line. Maximum frequency is 500 kHz. The actual implementation of this signal is to be decided by the I/O controller vendor. The backplane should support a maximum of 100 mA on this trace.

OTHER_IO_PRESENT_L

Active low signal is used to denote the presence of an option slot module in the opposite I/O option slot module. The I/O controller shall provide a pull-up for this line when it is inactive. This signal is cross-wired with the THIS_IO_PRESENT_L signal on the dual backplane only. This is a low-speed signal line. The actual implementation of this signal is to be decided by the I/O controller vendor. The backplane should support a maximum of 100 mA on this trace.

THIS_IO_PRESENT_L

Active low signal is used to denote the presence of an option slot module in the local I/O option slot module. The I/O controller shall provide a pull-up for this line when it is inactive. This signal is cross-wired with the OTHER_IO_PRESENT_L signal on the dual backplane only. This is a low-speed signal line. The actual implementation of this signal is to be decided by the I/O controller vendor. The backplane should support a maximum of 100 mA on this trace.

IMIO_(3:0)

These are direct connecting signals used for intermodule communication. The backplane should support a maximum of 100mA on these traces. See Figure 235 for details.

BOX_ID_BIT_(2:0)

Signal is used to determine the ID of the enclosure in which it is installed.

Reserved (2 pins)

Two undefined pins J5 Row A pin 6, and J5 Row B pin 6, which are not available for use at this time. No signals should be connected to these signals on either the backplane or the I/O Option Slot Module.

Silicon Motion - Exhibit 1020
Silicon Motion v. UTL - IPR2024-00199
Page 587 of 587