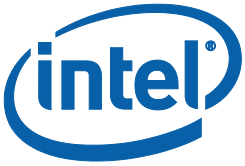


Enhanced Serial Peripheral Interface (eSPI)

Interface Specification (for Client Platforms)

May 2012

Revision 0.6

Document Number: 327432-001EN

Contents

Figures

Tables

Revision History

Document Number	Revision Number	Description	Revision Date
31288	0.4	• Initial release.	February 2012
31312	0.45	• Updated legal disclaimer.	February 2012
327432-001EN	0.6	• Updated with review feedback.	May 2012

§ §

1 Glossary

Table 1: Table of Glossary

Term	Definition
TBD	**TBD**

§ §

2 Introduction

This specification describes the architecture details of the Enhanced Serial Peripheral Interface (eSPI) bus interface. The devices that can be supported over this interface includes but not necessary limited to Embedded Controller (EC), Baseboard Management Controller (BMC), Super-I/O (SIO) and Port-80 debug card.

Prior to this specification, Embedded Controller (EC), Baseboard Management Controller (BMC) and Super I/O (SIO) are connected to the chipset through the Low Pin Count (LPC) bus. Low Pin Count (LPC) bus is a legacy bus developed as the replacement for Industry Standard Architecture (ISA) bus.

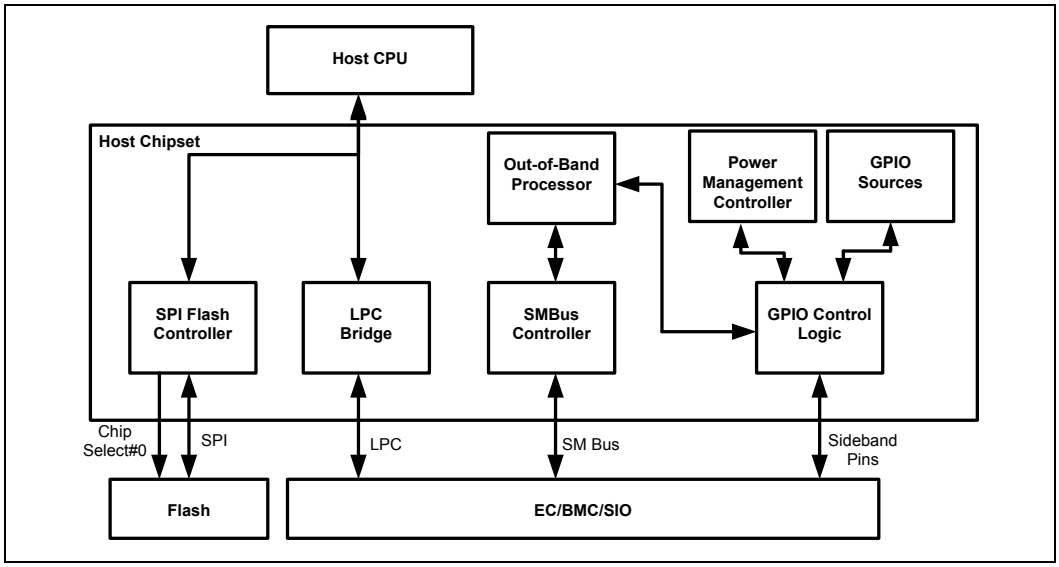
The specification generally refers to EC/BMC/SIO as the LPC device for the purpose of illustrating the eSPI bus capabilities and the comparison to LPC bus. However, EC/SIO is applicable for client platforms whereas BMC is generally associated with server platforms.

Here are some LPC bus limitations which led to the development of eSPI:

- LPC consists of 7 required pins and 6 optional pins that makes up to a total of 13 pins to implement.
- Present implementations of the LPC include a fabrication process cost burden as it is based on 3.3V I/O signaling technology.
- The frequency of the bus clock is fixed at 33 MHz. The fix LPC bandwidth of 133 Mbps is deemed insufficient to cater for the demands of new devices. Connecting these devices to high speed interfaces such as PCI Express and USB3 is prohibitive from cost perspective.
- There exist a significant number of sideband signals used for communication between chipset and EC, BMC and SIO that amounts to significant pin cost.

The diagram below shows how a EC/BMC/SIO is connected to the LPC bus.

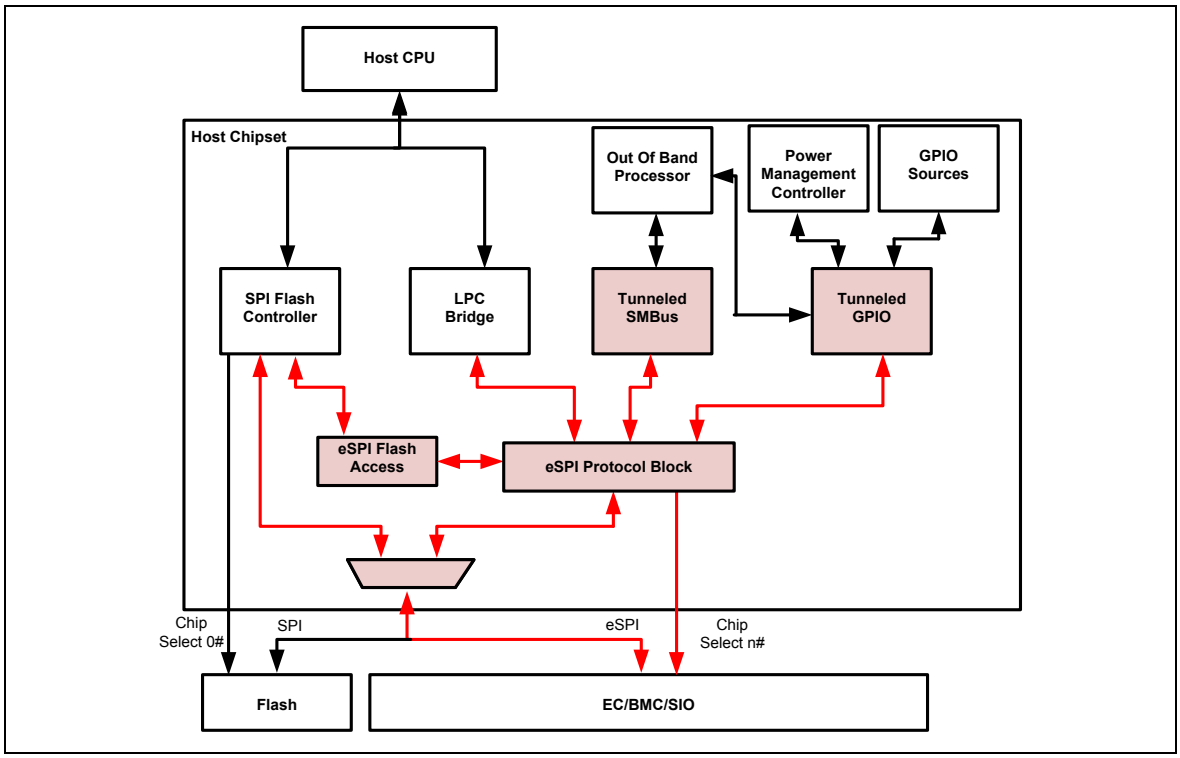
Figure 1: EC/BMC/SIO Communication over LPC



The eSPI specification provides a path for migrating LPC devices over to the new eSPI interface. eSPI reuses the timing and electrical specification of Serial Peripheral Interface (SPI) but with different protocol to meet a set of different requirements.

The diagram below shows how a EC/BMC/SIO can be connected to the eSPI bus.

Figure 2: EC/BMC/SIO Communication over eSPI



Sideband pin communications between chipset and these devices will be converted to in-band messages through the eSPI interface as part of the effort to reduce the component pin count and provide a migration path towards elimination of high-voltage 3.3V I/O pins.

Out-Of-Band (OOB) messaging between Out-Of-Band Processor in the chipset and Embedded Controller (EC) or Baseboard Management Controller (BMC) is also tunneled through the new eSPI interface as in-band messages, thus replacing the SMBus interface for this purpose.

Run-time flash sharing between chipset and slave devices will be supported over this new interface. The slave devices would be able to access the corresponding Flash partition through the Flash Access channel.

Depending on applications, eSPI bus may be active in all the S0-S5 system states. To lower the system power, the eSPI bus frequency and data pins may be a function of the system state.

2.1 Requirements

eSPI is defined to meet the following requirements:

- **Low Power:** The interface may be active in all S0-S5 system states. The power consumed when the bus is operating in S3-S5 system states must be very low to meet the power requirements of these low power system states. When the interface is not transmitting or receiving, it should consume a negligible amount of power (at system level).
- **Pin Count Reduction:** Moving LPC devices over to the eSPI interface facilitates the removal of LPC pins in the longer term. On top of that messaging through sideband pins needed for communication between the chipset and slave devices (such as EC, BMC and SIO) is converted to in-band messages, resulting in further pin count reduction.
- **Medium Bandwidth:** The bus bandwidth needs to be higher than that of the Low Pin Count (LPC) bus.
- **LPC Replacement:** Supports all the capabilities needed to replace the parallel LPC interface. However, 8237 DMA and Firmware Hub (FWH) are not supported over this interface.
- **Sideband Pins as In-Band Messaging**: Facilitates the removal of sideband pins for communication between chipset and slave devices by converting this communication into in-band messages sent over the eSPI bus.
- **Real Time Flash Sharing:** Supports flash sharing based on partition-able memory mapping. Allows real-time operational access by chipset and slave devices.
- **Chipset and Slave Devices SMBus Replacement:** Supports tunneling of all SMBus communication between chipset and slave devices over the new interface as in-band messages.

- **Scalable bandwidth:** Allows the bandwidth to be scaled based on application needs to optimize power versus performance. This could be done through frequency scaling or varying the number of active data pins.

- **Low Voltage I/O Buffer:** eSPI uses the same I/O buffer as Serial Peripheral Interface (SPI). The I/O buffer will support only 1.8V mode of operation for the eSPI bus.

§ §

3 Architecture Overview

3.1 System Topology

The Enhanced Serial Peripheral Interface (eSPI) operates in master/slave mode of operation where the eSPI master dictates the flow of command and data between itself and the eSPI slaves by controlling the Chip Select# pins for each of the eSPI slaves. At any one time, only one of the Chip Select# pins can be asserted allowing transactions to flow between the eSPI master and the corresponding eSPI slave associated with the Chip Select# pin. The eSPI master is the only component that is allowed to drive Chip Select# when eSPI Reset# is de-asserted.

For an eSPI bus, there is only one eSPI master and one or more eSPI slaves.

In Single Master-Single Slave configuration, a single eSPI master will be connected to a single eSPI slave. In one configuration, the eSPI slave could be the device that generates the eSPI Reset# such as EC or BMC. In this case, the eSPI Reset# is driven from eSPI slave to eSPI master. In other configuration, the eSPI Reset# could be generated by the eSPI master and thus, it is driven from eSPI master to eSPI slave.

Figure 3: Single Master-Single Slave with eSPI Reset# from Slave to Master

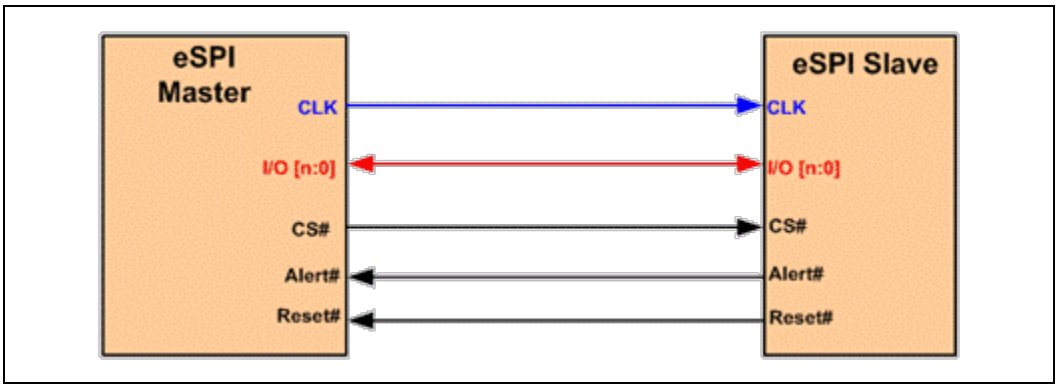


Figure 4: Single Master-Single Slave with eSPI Reset# from Master to Slave

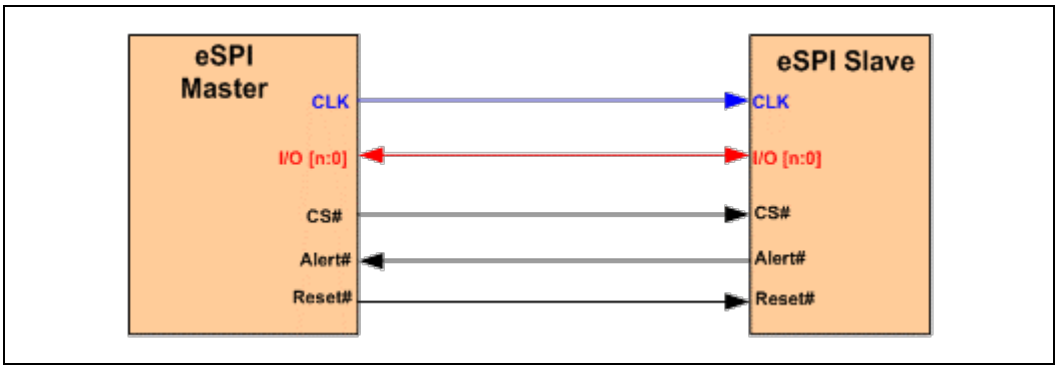
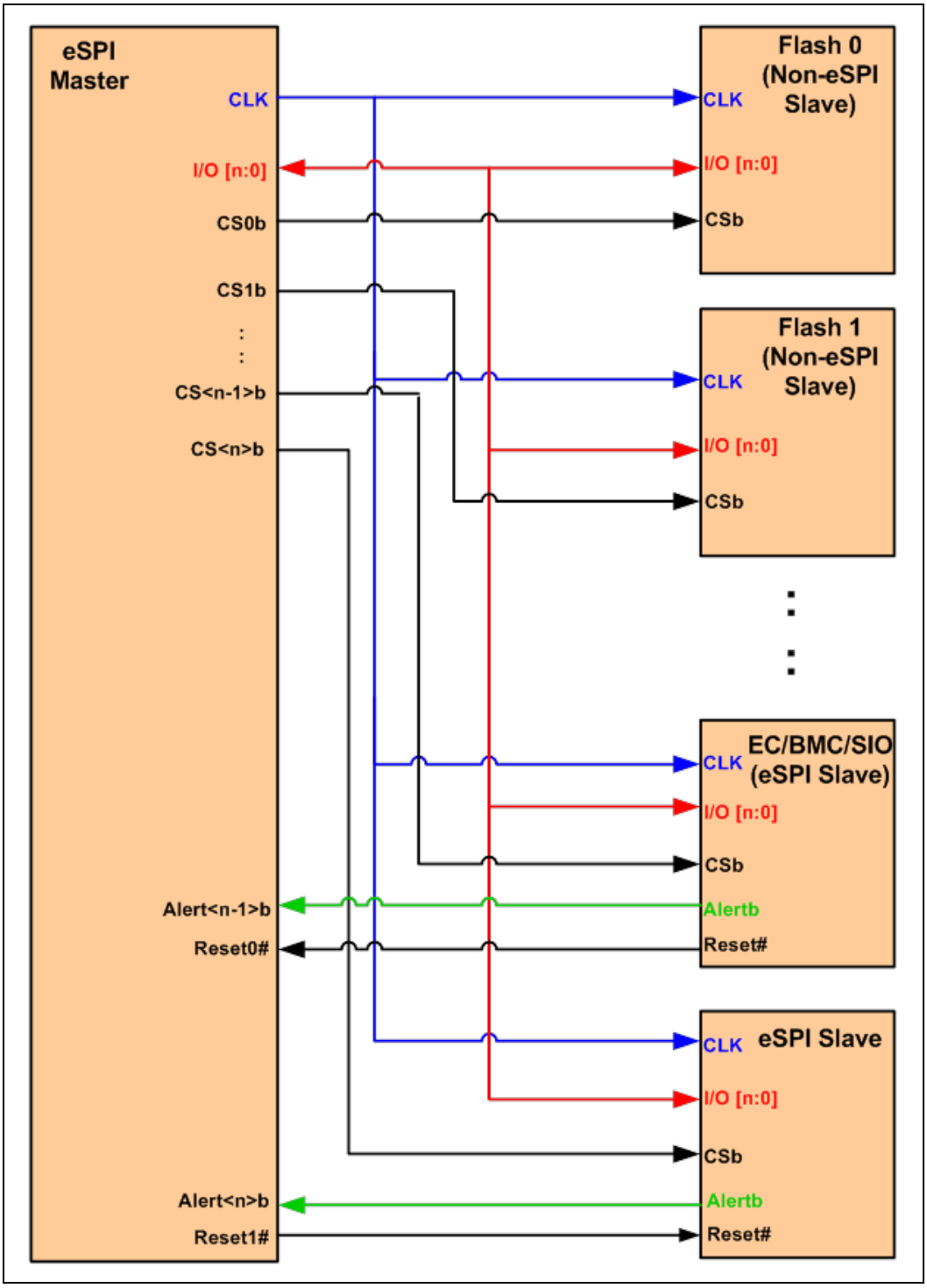


Figure 5: Single Master-Multiple Slaves with Two eSPI Reset#



Multiple SPI and eSPI slaves could be connected to the same eSPI bus interface in a multi-drop Single Master-Multiple Slaves configuration. The number of devices that can be supported over a single eSPI bus interface is limited by bus loading and signals trace length.

In this configuration, the clock and data pins are shared by multiple SPI and eSPI slaves. Each of the slaves has its dedicated Chip Select# and Alert# pins.

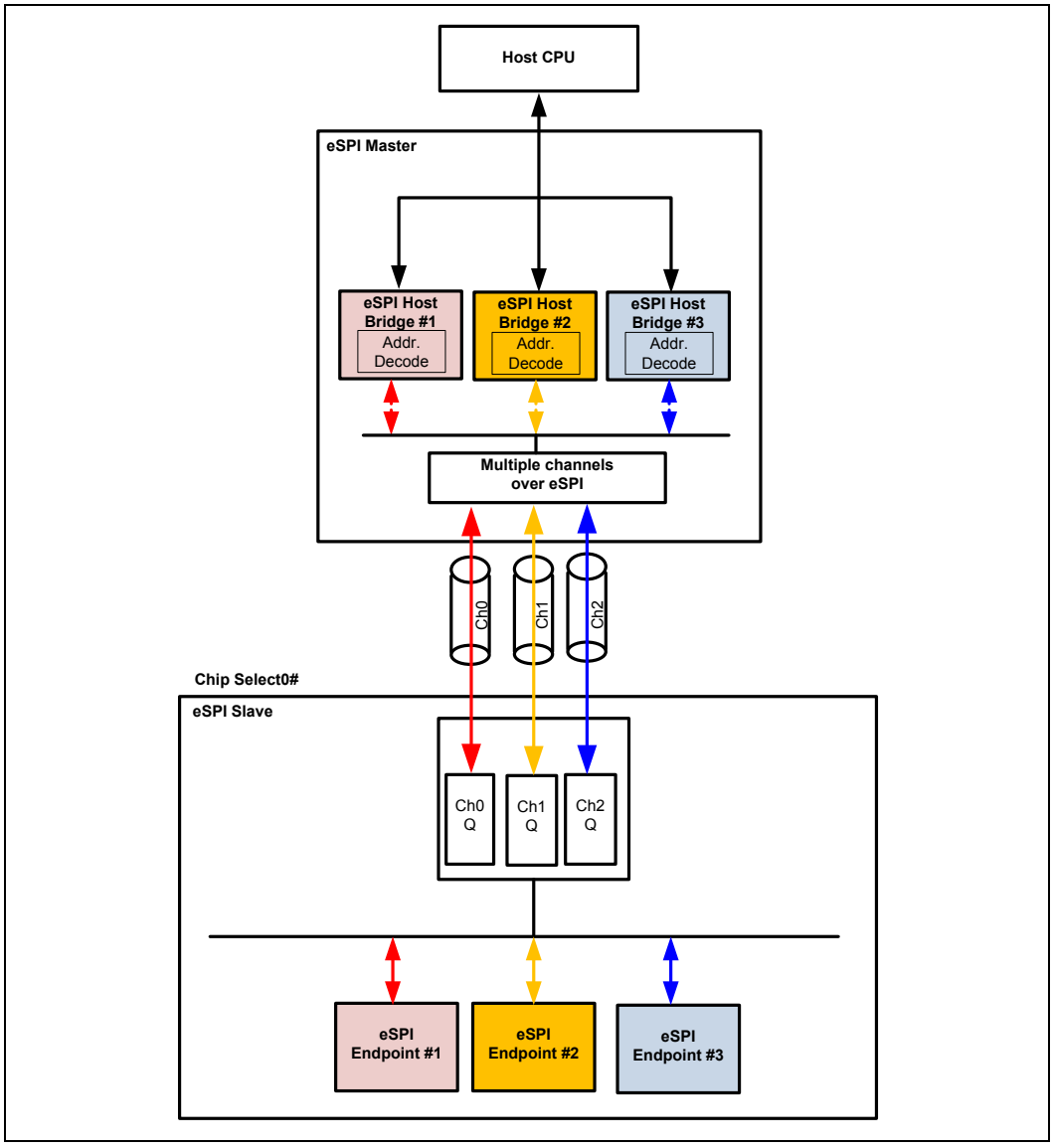
In an eSPI bus configuration with multiple slaves present, the eSPI master may support 2 eSPI Reset# pins, one from eSPI slave such as EC or BMC to eSPI master and another one from eSPI master to eSPI slaves. In this case, the master's eSPI interface will only be reset if all the slaves' eSPI interfaces are reset.

SPI slaves such as Flash and TPM are allowed to share the same set of clock and data pins with eSPI slaves. These non-eSPI slaves are selected using the dedicated Chip Select# pins and they communicate with the eSPI master through SPI specific protocols ran over the eSPI bus.

3.2 Architecture Descriptions

In a Single Master-Single Slave configuration as shown in the diagram below, there could be multiple eSPI host bridges within a single eSPI master and there could be multiple eSPI endpoints within a single eSPI slave.

Figure 6: Single Master-Single Slave (Multiple Channels)



When Chip Select# corresponding to the eSPI slave is asserted, command and data transfer happens between the eSPI master and eSPI slave, which could be a result of the eSPI host bridge and eSPI endpoint communications.

Each of the eSPI host bridges communicates with its corresponding eSPI endpoint through dedicated channel.

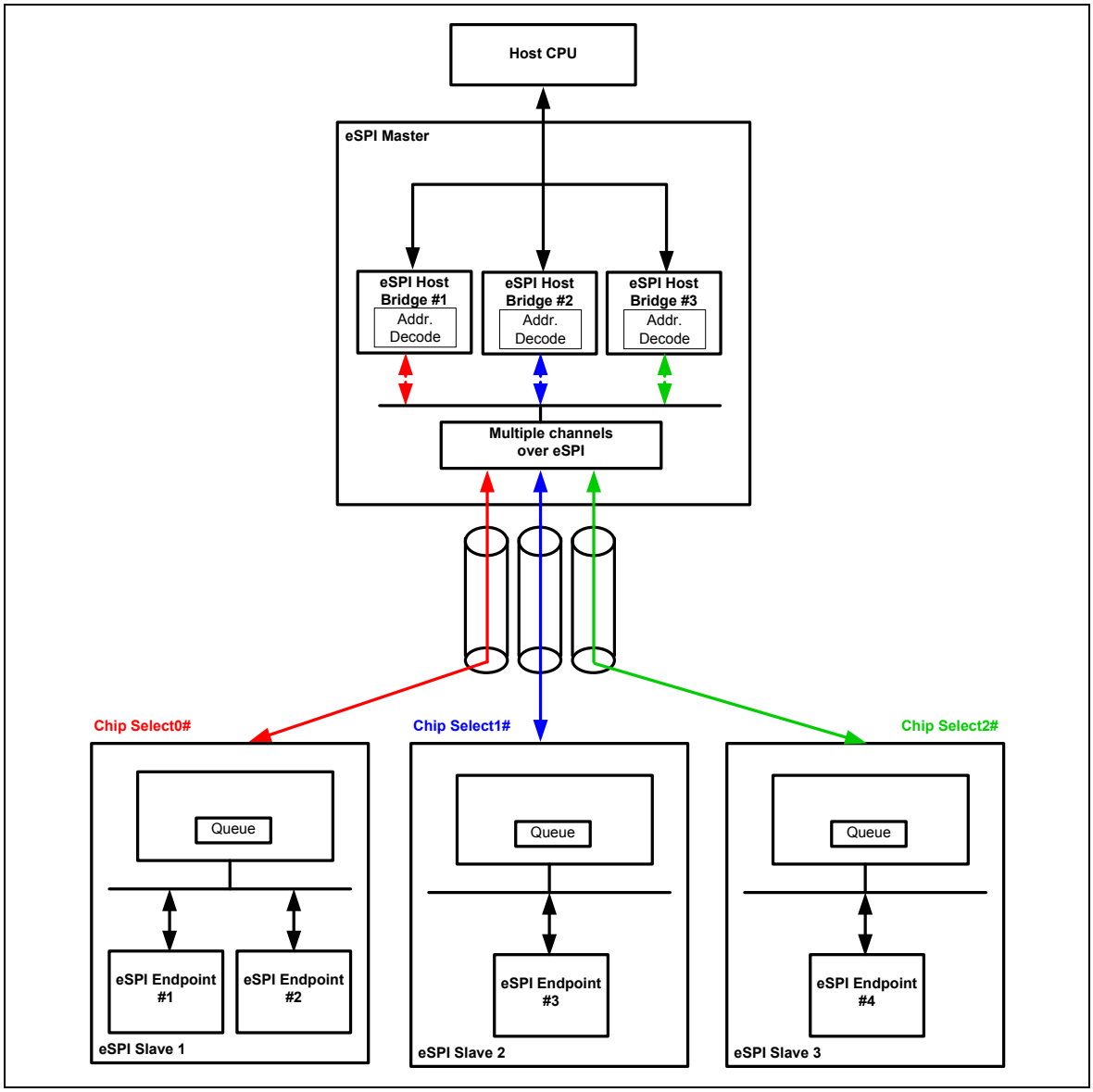
The use of channels allows multiple independent flows of command and data to be transferred over the same bus between the eSPI master and eSPI slave with no ordering requirement.

Resources such as flow control, command and data queues are dedicated for each of the channels to provide independent command and data flows.

In Single Master-Multiple Slaves configuration shown in the diagram below, multiple discrete eSPI slaves can be dropped onto the eSPI bus. Each of the eSPI slaves should have a dedicated Chip Select# pin. On the master side, there are eSPI host bridges corresponding to each of the discrete slaves respectively, each driving the Chip Select# pin of the corresponding discrete slave.

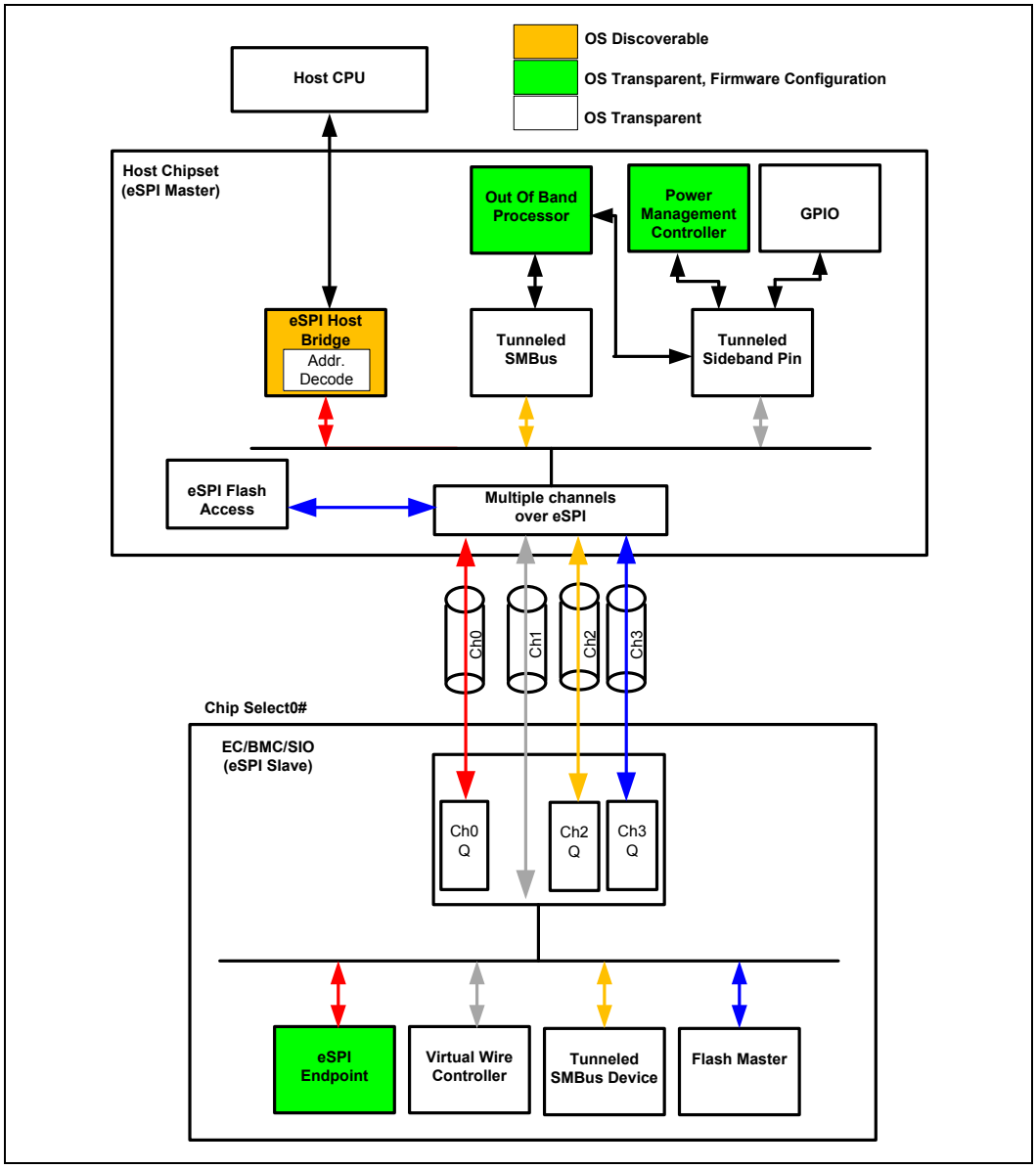
At any one time, only one of the Chip Select# pins can be asserted. Command and data transfer can then happen between the eSPI host bridge and the corresponding eSPI slave.

Figure 7: Single Master-Multiple Slaves



The next diagram shows one of the ways the specification can be used to support EC/BMC/SIO communication over the eSPI interface.

Figure 8: EC/BMC/SIO Communication Over eSPI Channels



In this example, the eSPI host bridge and the corresponding eSPI endpoint communicate through Channel 0. The Sideband Pins are tunneled as in-band messages through Channel 1. SMBus OOB messages are tunneled through Channel 2. Flash access transactions are accomplished through Channel 3. The transactions for different channels flowing between the eSPI master and EC/BMC/SIO share the same Chip Select# pin, and the same set of data and clock pins.

3.3 Pin Descriptions

eSPI uses the existing SPI I/O buffer. The electrical specification for this new interface is the same as SPI.

eSPI Reset# is typically driven from eSPI master to eSPI slaves. The exception is when eSPI Reset# is generated by eSPI slave such as EC or BMC, which drives the eSPI Reset# to the eSPI master. eSPI Reset# is the reset to the eSPI interface on both sides.

eSPI master and eSPI slaves must tri-state the interface pins when their respective eSPI Reset# is asserted. These pins include Chip Select#, Serial Clock, I/O[n:0] and Alert# pins which have weak pull-up enabled during the reset. Weak pull-up should be implemented either as an integral part of the eSPI master buffer or on the board. eSPI slaves must not implement the weak pull-up.

Refer to Section 8.1 - Electrical Specification for the value of the weak pull-up resistor.

After eSPI Reset# is deasserted on the eSPI master, the eSPI master begins driving Chip Select# and Serial Clock pins to their idle state appropriately. The weak pull-up on the Chip Select# and Serial Clock are allowed to be disabled after the eSPI Reset# deassertion. However, I/O[n:0] and Alert# pins continue to have the weak pull-up enabled for the proper operation of the eSPI bus.

Figure 9: eSPI Pin List

Pin Name	Direction	Clock	Description
eSPI Reset#	Master to Slave[1] or Slave to Master[2]	Asynchronous	**Reset#:** Reset the eSPI interface for both master and slaves. **Note:** 1. eSPI Reset# is typically driven from eSPI master to eSPI slaves. 2. eSPI Reset# is generated by eSPI slave such as EC or BMC, driven from eSPI slave to eSPI master.
Chip Select#	Master to Slave	Asynchronous	**Chip Select#:** Driving Chip Select# low selects a particular eSPI slave for the transaction. Each of the eSPI slaves is connected to a dedicated Chip Select# pin.
Serial Clock	Master to Slave	-	**Clock:** This pin provides the reference timing for all the serial input and output operations.
I/O [n:0]	Bi-directional	Serial Clock	**I/O:** These are bi-directional input/output pins used to transfer data between master and slaves. The value of 'n' may be 0, 1 or 3 depending on the I/O mode.

Pin Name	Direction	Clock	Description
			In Single I/O mode (n=0), I/O[0] is the eSPI master output/eSPI slave input (MOSI) whereas I/O[1] is the eSPI master input/eSPI slave output (MISO).
Alert#	Slave to Master	Asynchronous	**Alert#:** This pin is used by eSPI slave to request service from eSPI master. Alert# is an open-drain output from the slave. This pin is optional for Single Master-Single Slave configuration where I/O[1] can be used to signal the Alert event.

§ §

4 Bus Protocol

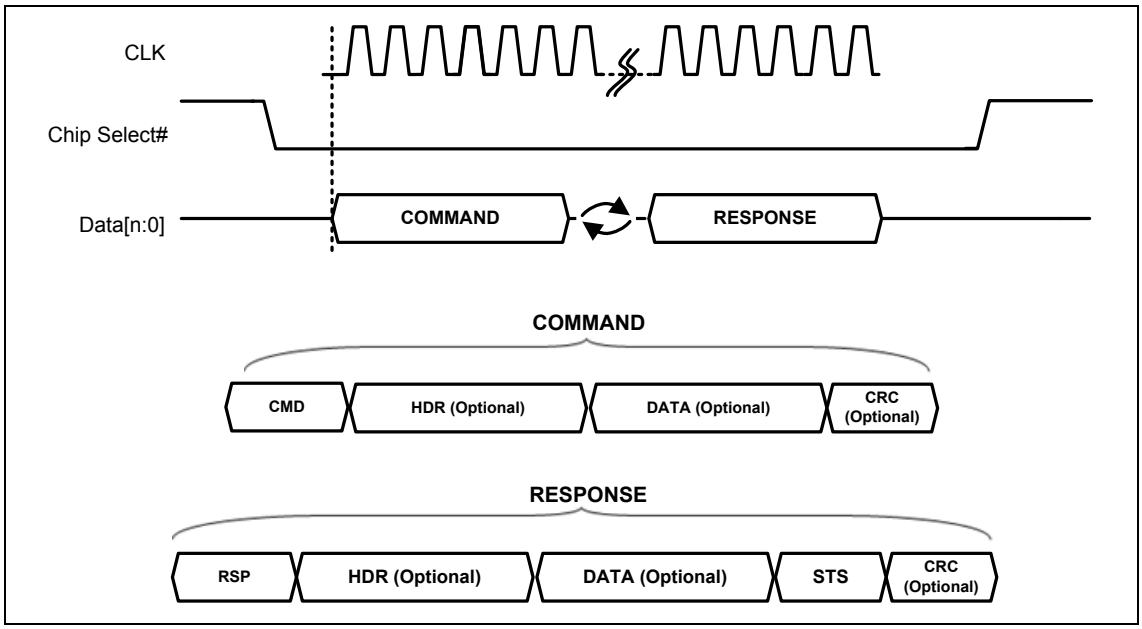
The details of the Enhanced Serial Peripheral Interface (eSPI) protocol are described in this section. The electrical of eSPI bus is similar to SPI bus with deviations specifically called out in this specification.

The Serial Clock must be low when the Chip Select# is de-asserted and eSPI Reset# is de-asserted. The data is launched on the falling edge of the clock from master and sampled on the rising edge of the clock by slave. The data is launched on the falling edge of the clock from slave. The master could implement a more flexible sampling scheme since it controls the clock.

All transactions on eSPI must be in multiple of 8-bits (one Byte).

4.1 Basic Protocol

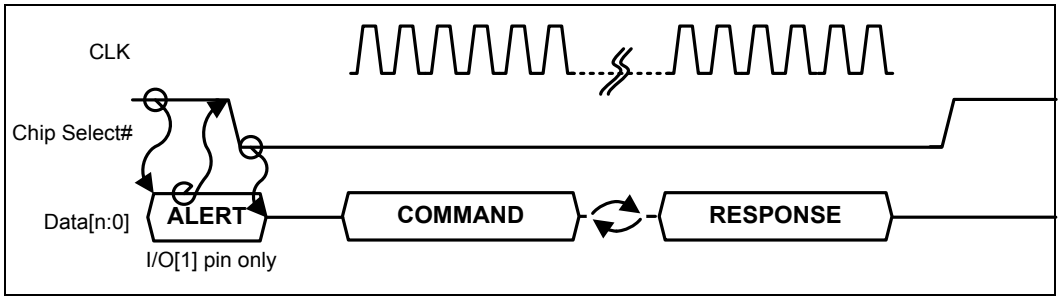
Figure 10: Basic eSPI Protocol



eSPI transaction consists of a Command phase driven by master, a Turn-Around (TAR) phase, and a Response phase driven by the slave. The Command phase consists of a CMD, an optional header (HDR), optional DATA and an optional CRC. The Response phase consists of a RSP, an optional header (HDR), optional data, Status and an optional CRC. When CRC support is disabled, the CRC bytes are not transmitted on the bus.

A transaction could be initiated by the master through the assertion of Chip Select#, start the clock and drive the command onto the data bus. The clock remains toggling until the complete response phase has been received from the slaves.

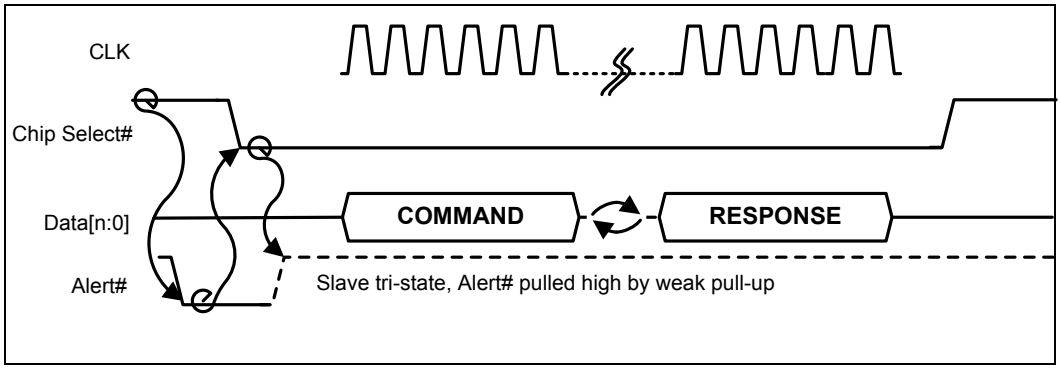
Figure 11: Slave Triggered Transaction (Single Master-Slave)



A transaction could be initiated by the slave by first signaling an Alert event to the master. The Alert event could be signaled through two ways. In the Single Master-Single Slave configuration, the I/O[1] pin could be used by the slave to indicate an Alert event. In the Single Master-Multiple Slaves configuration, a dedicated Alert# pin is required.

The Alert event can only be signaled by the slave when the Chip Select# is high. The pin, either I/O[1] or Alert# is toggled from tri-state to pulled low by the slave when it decides to request for service. The slave then holds the state of the pin until the Chip Select# is asserted by the master. Once the Chip Select# is asserted, the eSPI slave must release the ownership of the pin by tri-stating the pin and the pin will be pulled high by the weak pull-up. The master then continues to issue command to figure out the cause of the Alert event from the device and then service the request.

Figure 12: Slave Triggered Transaction (Multiple Slave)



The specification does not prevent the use of a dedicated Alert# pin for the Single Master-Single Slave configuration.

In the boundary case where the Alert event assertion aligns with Chip Select# assertion, the slave still tri-state the pin after sampling the corresponding Chip Select# assertion. The status is returned during the response phase and the master is then aware of the need to service the slave's outstanding requests.

The Alert event signaled on the pin is asynchronous to the Serial Clock.

The eSPI master and eSPI slaves must support both types of Alert mechanism. The method to determine which Alert mechanism to use for each of the eSPI slaves is implementation specific.

eSPI is defined to use packet-based split transaction protocol. On the transmit side, the packets are formed in the Transaction Layer based on the transaction to be sent. The Link Layer extends the packet with CRC when CRC support is enabled.

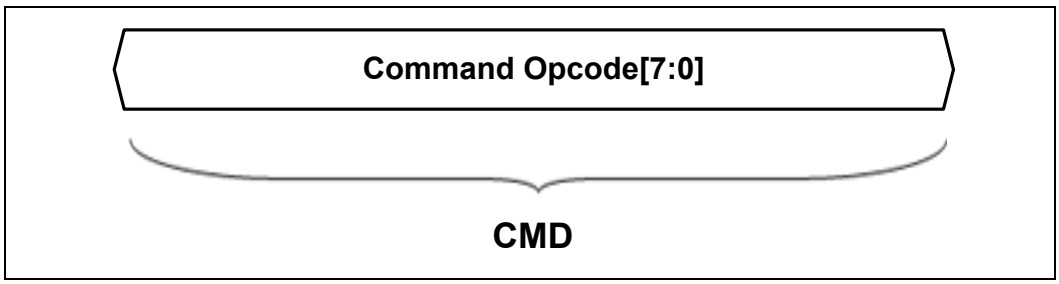
Similarly on the receive side, the CRC is checked at the receiving Link Layer when CRC support is enabled. Once the packet passes the CRC check, the packet is sent to Transaction Layer where it is decoded and acted upon.

4.2 Command Phase

The Command phase is used by the eSPI master to initiate a transaction to the slave or in response to an Alert event by the slave. It consists of a CMD, an optional header (HDR), optional DATA and an optional CRC.

The CMD field consists of Command Opcode.

Figure 13: Command Field



The Command Opcode is used to indicate channel specific commands and to communicate link management events.

Channels specific commands communicated over the bus include Command Put and Command Get for the respective channels.

Link management events include GET_STATUS, GET_CONFIGURATION and SET_CONFIGURATION.

The Command Opcode is 8-bits wide.

If the slave receives a Command Opcode on a packet which is not supported, the slave must not respond to the transaction. The transaction will be terminated with the default response (NO_RESPONSE) on the bus.

Table 2: Command Opcode Encodings

CMD Opcode	Encoding[7:0]	Description
eSPI Peripheral Channel		
PUT_PC	00000000	Put a posted or completion header and optional data. Note: It is illegal to issue a PUT_PC unless the slave has indicated that it is free to take the Posted or Completion packet.
PUT_NP	00000010	Put a non-posted header and optional data. Note: It is illegal to issue a PUT_NP unless the slave has indicated that it is free to take the Non-Posted packet.
GET_PC	00000001	Get a posted or completion header and optional data. Note: It is illegal to issue a GET_PC unless the slave has indicated that it has a Posted or Completion packet available
GET_NP	00000011	Get a non-posted header and optional data. Note: It is illegal to issue a GET_NP unless the slave has indicated that it has a Non-Posted packet available.
PUT_IORD_SHORT	$010000C_1C_0$[1]	Put a short (1-4 bytes) non-posted I/O Read packet. Note: It is illegal to issue a PUT_IORD_SHORT unless the slave has indicated that it is free to take the Non-Posted packet.
PUT_IOWR_SHORT	$010001C_1C_0$[1]	Put a short (1-4 bytes) non-posted I/O Write packet. Note: It is illegal to issue a PUT_IOWR_SHORT unless the slave has indicated that it is free to take the Non-Posted packet.
PUT_MEMRD32_SHORT	$010010C_1C_0$[1]	Put a short (1-4 bytes) non-posted Memory Read 32 packet. Note: It is illegal to issue a PUT_MEMRD32_SHORT unless the slave has indicated that it is free to take the Non-Posted packet.
PUT_MEMWR32_SHORT	$010011C_1C_0$[1]	Put a short (1-4 bytes) posted Memory Write 32 packet. Note: It is illegal to issue a PUT_MEMWR32_SHORT unless the slave has indicated that it is free to take the Posted or Completion packet.
Virtual Wire Channel		

CMD Opcode	Encoding[7:0]	Description
PUT_VWIRE	00000100	Put a Tunneled virtual wire packet.
GET_VWIRE	00000101	Get a Tunneled virtual wire packet.
OOB Message Channel		
PUT_OOB	00000110	Put an OOB (Tunneled SMBus) message. Note: It is illegal to issue a PUT_OOB unless the slave has indicated that it is free to take the OOB message.
GET_OOB	00000111	Get an OOB (Tunneled SMBus) message. Note: It is illegal to issue a GET_OOB unless the slave has indicated that it has an OOB message available to send.
Flash Access Channel		
PUT_FLASH_C	00001000	Put a Flash Access completion. Used in Master Attached Flash Sharing mode for the master to return a flash access completion to the slave. Note: It is illegal to issue a PUT_FLASH_C unless the slave has indicated that it is free to take the Flash Access completion.
GET_FLASH_NP	00001001	Get a non-posted Flash Access request. Used in Master Attached Flash Sharing mode for the slave to issue a flash access request to the master. It is illegal to issue a GET_FLASH_NP unless the slave has indicated that it has a non-posted Flash Access request available to send.
PUT_FLASH_NP	00001010	Put a non-posted Flash Access request. Used in Slave Attached Flash Sharing mode for the master to issue a flash access request to the slave. Note: It is illegal to issue a PUT_FLASH_NP unless the slave has indicated that it is free to take the non-posted Flash Access request.
GET_FLASH_C	00001011	Get a Flash Access completion. Used in Slave Attached Flash Sharing mode for the slave to return a flash access completion to the master. Note: It is illegal to issue a GET_FLASH_C unless the slave has indicated that it has a Flash Access completion available to send.

CMD Opcode	Encoding[7:0]	Description
Channel Independent[2]		
GET_STATUS	00100101	Command initiated by the master to read the status register of the slave.
SET_CONFIGURATION	00100010	Command to set the capabilities of the slave as part of the initialization. This is typically done after the master discovers the capabilities of the slave.
GET_CONFIGURATION	00100001	Command to discover the capabilities of the slave as part of the initialization.

Note:

1. The opcode encoding C_1C_0 indicates the length of the request. The address together with the length must not cross the DWord boundary.

Encoding[1:0] C_1C_0	Request Length
00	1 byte
01	2 bytes
10	3 bytes
11	4 bytes

2. Channel independent commands are enabled by default upon eSPI Reset# deassertion.

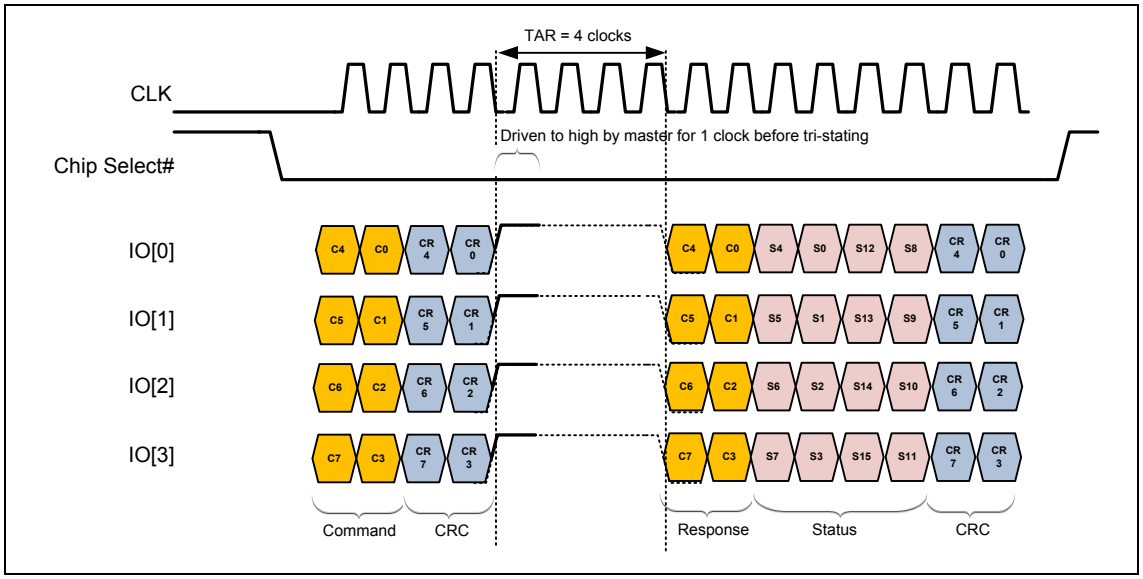
4.3 Turn-Around (TAR)

After the last bit of the Command Phase has been sent out on the data lines, the data lines enter the Turn-Around window. The eSPI master is required to drive all the data lines to logic '1' for the first clock of the Turn-Around window and tri-state the data lines thereafter. The number of clocks for the Turn-Around window is specified by the Operating Turn-Around Time Value field in the General Capabilities and Configurations register. This is to provide sufficient time for the slave to sample the command and prepare the response.

The eSPI slave must drive the Response phase on the bus immediately upon the expiry of the Turn-Around time as shown in the next diagram.

During the Turn-Around window, the data lines will be pulled high by the weak pull-up.

Figure 14: Turn-Around Time Example (TAR = 4 clocks)



The Turn-Around time can be queried from the individual eSPI slaves during initialization through the GET_CONFIGURATION command. The Turn-Around time could be different between different slaves. The eSPI master could choose to use a common Turn-Around value for all slaves or choose to optimize the performance by using different Turn-Around values for each of the slaves.

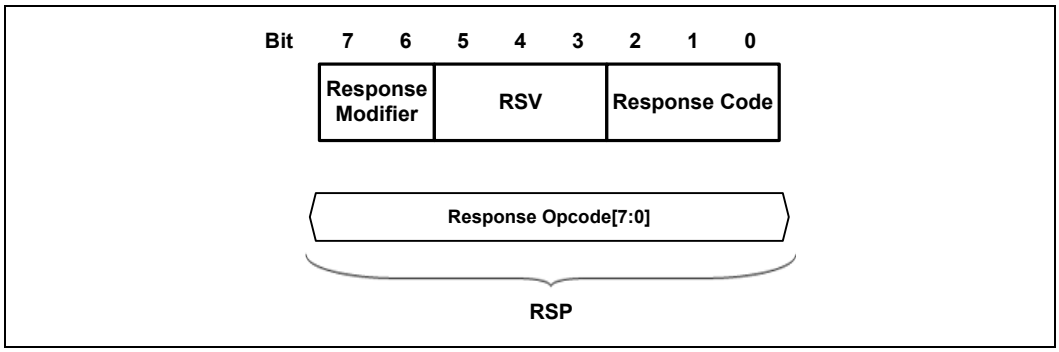
Prior to initialization, a fixed $t_{INIT-TAR}$ eSPI clocks is used as the initial Turn-Around time. All the eSPI master and slaves must support the initial Turn-Around time coming out of reset. During initialization, eSPI master could change the Turn-Around time to be used for the slave through the SET_CONFIGURATION command for each of the slaves. The Turn-Around time set by the eSPI master must be within the supported range advertised by the slaves. The new Turn-Around time will be effective at the deassertion edge of the Chip Select#.

4.4 Response Phase

The Response phase is driven by the eSPI slave in response to command initiated by an eSPI master. It consists of a RSP opcode, an optional header (HDR), optional data, STATUS and an optional CRC.

The RSP opcode is a 8-bit field consists of a Response Code and a Response Modifier.

Figure 15: Response Field



4.4.1 Response

The Response Code indicates whether the request is successful, deferred or contains errors.

The Response Modifier is a 2-bit field defined for the GET_STATUS with an ACCEPT response only. For all other responses, it must always have the value of "00".

The Response Modifier field indicates whether a peripheral (channel 0) completion, a virtual wire (channel 1) packet or a flash access (channel 3) completion is appended to the GET_STATUS response phase.

The Reserved (RSV) field of the RSP opcode must be driven to all 0's when the slave drives the response phase. It is reserved for future use by the specification. For the purpose of backward compatibility, the Reserved (RSV) field must be ignored by the master.

NO_RESPONSE is the default when the response phase is not driven by any slave. The eSPI master may terminate the transaction by deasserting Chip Select# at any point when this is detected.

Table 3: Response Field Encodings

RESPONSE	Encoding			Description
	[7:6]	[5:3]	[2:0]	
ACCEPT	R_1R_0[1]	RSV	000	Command was successfully received

RESPONSE	Encoding			Description
				If the command was a PUT_NP, a response of ACCEPT means that the non-posted transaction is being completed as a "connected" transaction.
DEFER	00	RSV	001	Only valid in response to a PUT_NP. A non-posted command was successfully received, and completing the non-posted transaction is deferred to a future split completion.
NON_FATAL_ERROR	00	RSV	010	The received command had an error with non-fatal severity. The error does not affect the ability to process the received command.
FATAL_ERROR	00	RSV	011	The received command had a fatal error that prevented the transaction layer packet from being successfully processed. Fatal errors include malformed transactions, Put without Free, Get without Avail and etc.
NO_RESPONSE	11	111	111	The response encoding of all 1's is defined as no response. It is the default response to the GET_CONFIGURATION when no slave is present as a result of the weak pull-up on the data lines. It is also the default response when fatal CRC error is detected on the command packet, or when command opcode is not supported and the slave must not drive the response phase.

Note:

1. The response encoding R_1R_0 is always "00" except for the GET_STATUS with an ACCEPT response which has the following definition:

Encoding[7:6] R_1R_0	Description
00	No append.
01	A Peripheral (channel 0) completion is appended.
10	A Virtual Wire (channel 1) packet is appended.
11	A Flash Access (channel 3) completion is appended.

4.4.2 Status

The Status field serves to provide information such as new pending requests from the slave and queue free information.

Refer to Section 5.3 - Slave Buffer Management for additional details about setting and clearing of the eSPI Status register bits.

Figure 16: Slave's Status Register Definition

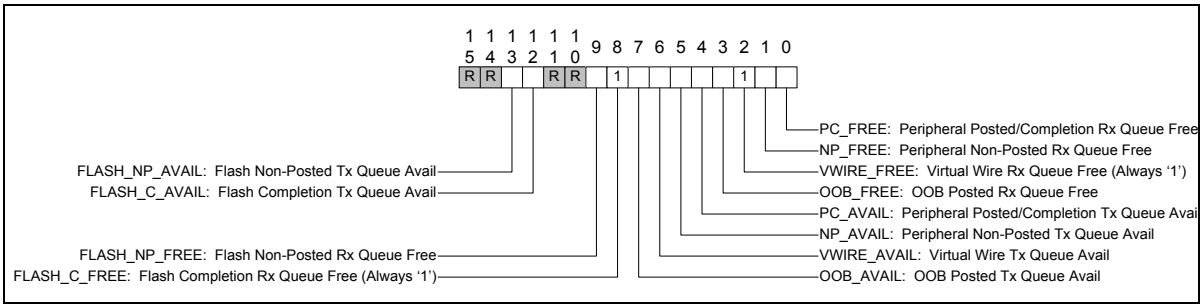


Table 4: Status Field Encodings

STATUS	Bits Position	Description
Slave's Rx queues Free		
PC_FREE	0	When '1', indicates the slave is free to accept at least one channel 0 peripheral posted or completion header and data up to maximum payload size.
NP_FREE	1	When '1', indicates the slave is free to accept at least one channel 0 peripheral non-posted header and 1 DW of Data (if applicable).
VWIRE_FREE	2	This bit must be always a '1'. Tunneling of channel 1 virtual wires is not flow controlled.
OOB_FREE	3	When '1', indicates the slave is free to accept at least one channel 2 OOB (tunneled SMBus) message with data up to maximum payload size.
Slave's Tx queues Available		
PC_AVAIL	4	When '1', indicates the slave has a channel 0 peripheral posted or completion header and optional data up to maximum payload size available to send.
NP_AVAIL	5	When '1', indicates the slave has a channel 0 peripheral non-posted header available to send.
VWIRE_AVAIL	6	When '1', indicates the slave has a channel 1 tunneled virtual wire available to send.
OOB_AVAIL	7	When '1', indicates the slave has a channel 2 OOB (tunneled SMBus) message with data up to maximum payload size available to send.
Slave's Rx queues Free		

STATUS	Bits Position	Description
FLASH_C_FREE	8	When '1', indicates the slave is free to accept at least one channel 3 Flash Access completion header and data up to maximum payload size. This bit must be always a '1'. The slave must be able to accept the completion for the non-posted request it sends.
FLASH_NP_FREE	9	When '1', indicates the slave is free to accept at least one channel 3 Flash Access non-posted header and data up to maximum payload size.
Reserved	11:10	Reserved.
Slave's Tx queues Available		
FLASH_C_AVAIL	12	When '1', indicates the slave has a channel 3 Flash Access completion header and data up to maximum payload size available to send.
FLASH_NP_AVAIL	13	When '1', indicates the slave has a channel 3 Flash Access non-posted header and data up to maximum payload size available to send.
Reserved	15:14	Reserved.

4.5 Alert Phase

Alert phase is signaled by the slave to request for service. In response to an Alert, the master can issue a GET_STATUS command to the corresponding slave to query for the cause of the Alert event.

The master then reacts accordingly to service the slave.

A slave could generate an Alert event due to any of the following reasons:

- There is a new request from slave. This could be a Posted, Non-Posted, deferred Completion, Virtual Wire messages, OOB messages or Flash Access requests.
- A slave buffer space has become free since the last status update was returned as not free.

Each of the cause that triggers the Alert event has the corresponding bit in the STATUS register. When the state of the STATUS register is different from the STATUS returned during the previous Response phase, the slave will generate a new Alert event. The difference in the STATUS register indicates a new event has occurred that requires the service from the master.

Following figures illustrate examples of the flow and the tracking of the slave's STATUS register on both sides of the bus.

Figure 17: Flow Diagram for a Slave to Master Peripheral Posted Write

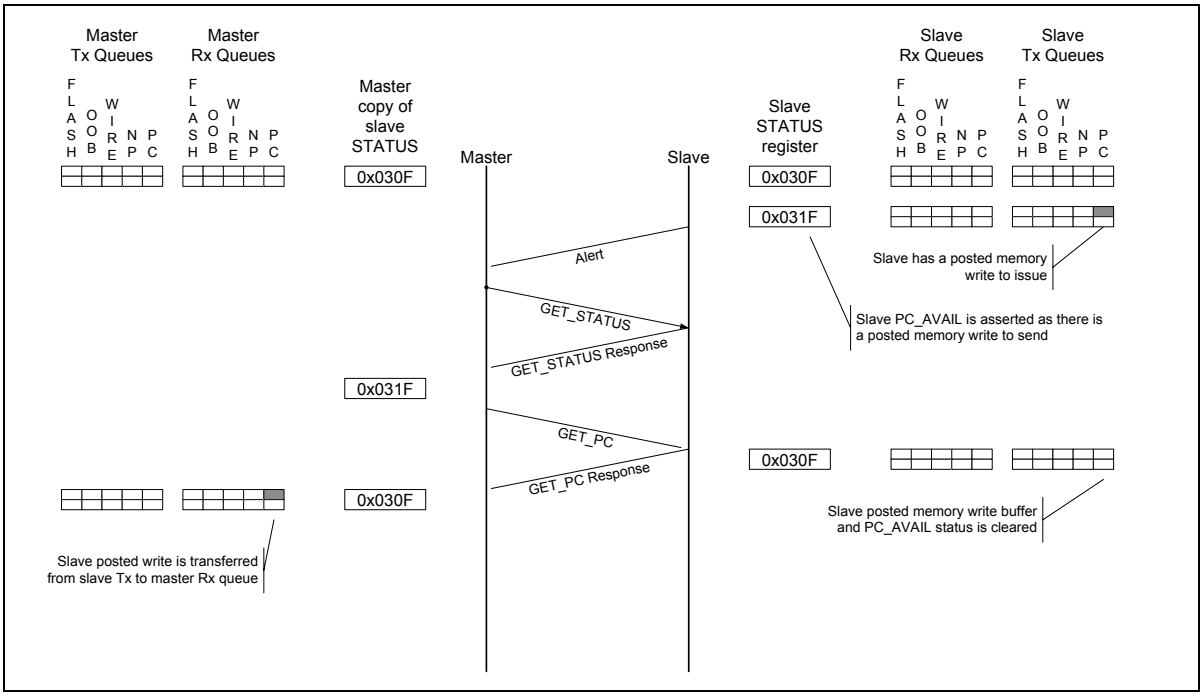


Figure 18: Flow Diagram for a Back-to-back Slave to Master Peripheral Posted Write

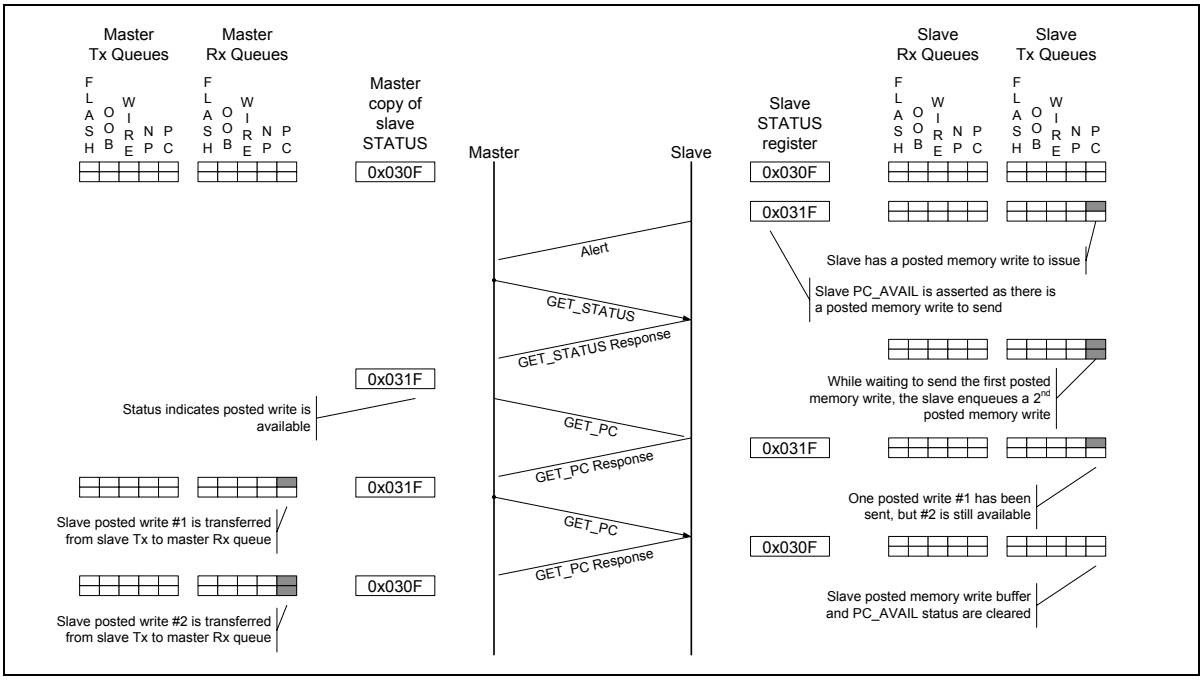
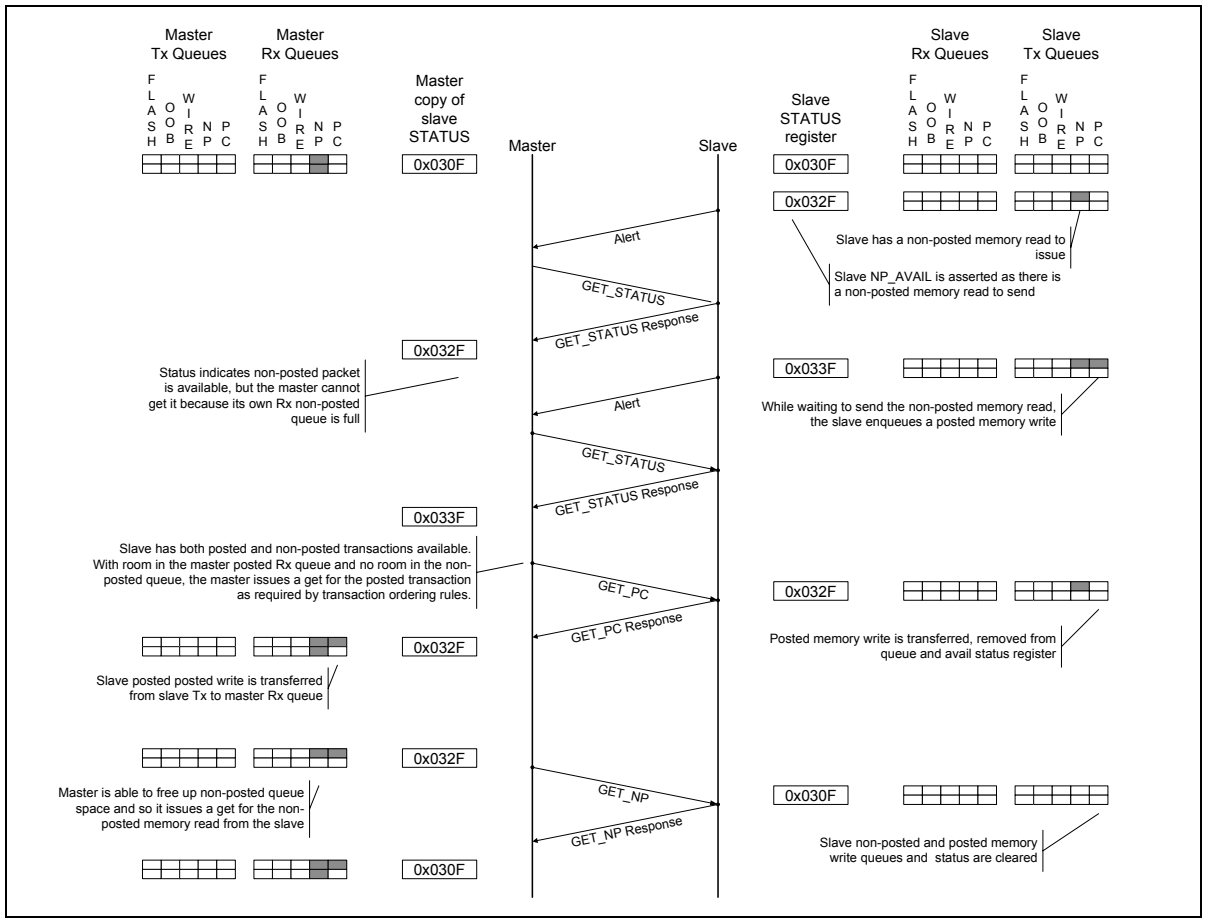
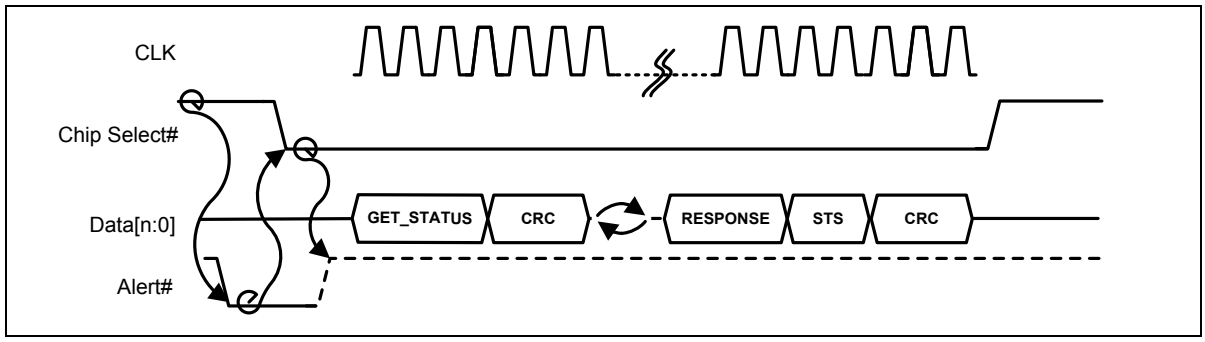


Figure 19: Flow Diagram for a Slave to Master Peripheral Posted Write passes Non-posted



4.6 Get Status Command

Figure 20: GET_STATUS Command



GET_STATUS is a channel independent command which is used to query the content of the Status register. The state of the Status register will be returned in the Response phase.

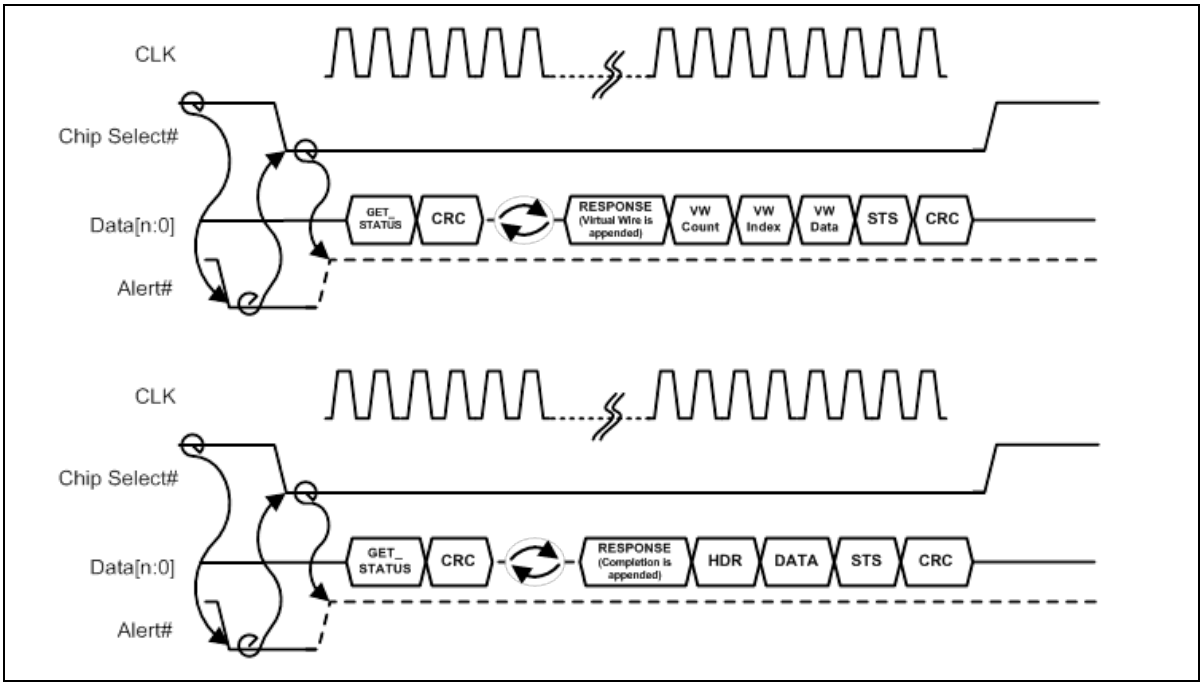
This command is typically used in response to the Alert event from the eSPI slave, to determine the cause of the Alert event and subsequently service the slave.

The response phase of the GET_STATUS allows a peripheral (channel 0) completion, a virtual wire (channel 1) packet or a flash access (channel 3) completion to be appended and sent together with the response. Only one is allowed to be appended to the GET_STATUS response as indicated by the Response Modifier field.

The eSPI master must always be ready to accept the peripheral (channel 0) completion, the virtual wire (channel 1) packet or the flash access (channel 3) completion. For the completion, it requires the eSPI master to pre-allocate the completion buffer appropriately when the non-posted transaction is initiated to the slave.

Refer to Section 4.4.2 for additional details of the Status register.

Figure 21: GET_STATUS Command (with Response Modifier)

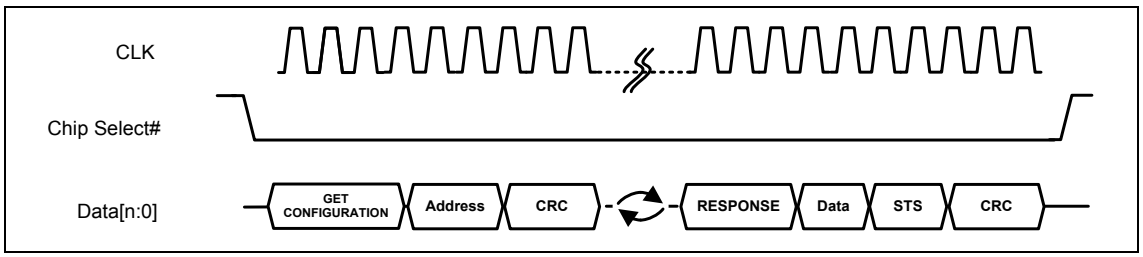


4.7 Get Configuration and Set Configuration Command

SET_CONFIGURATION and GET_CONFIGURATION commands are channel independent commands that are used to access the Channel Capability and Configuration registers on the eSPI slave side. Only DWord accesses are supported. Since there is no byte enables, software is required to perform a Read-Modify-Write access if modifying less than a full DWord.

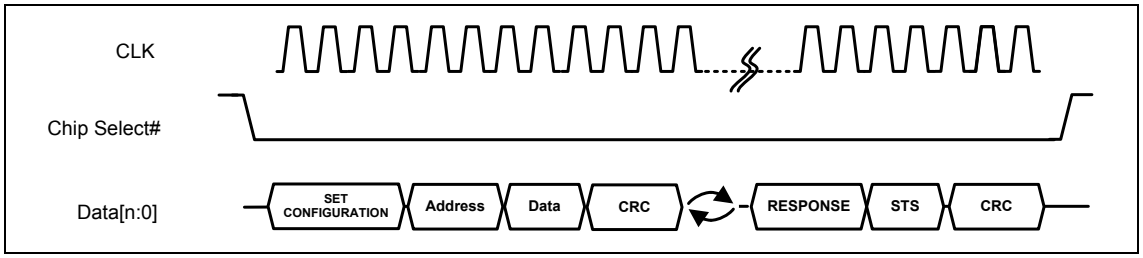
SET_CONFIGURATION and GET_CONFIGURATION commands can never be deferred and must be completed within the same cycle.

Figure 22: GET_CONFIGURATION Command



GET_CONFIGURATION command is used to read the Channel Capability and Configuration registers on the eSPI slaves. The GET_CONFIGURATION command phase consists of an 8-bit Command Opcode, 16-bit address and an optional 8-bit CRC. The response phase includes the 8-bit Response, 1 DW of Data, 16-bit Status and an optional 8-bit CRC.

Figure 23: SET_CONFIGURATION Command



SET_CONFIGURATION command is used to write the Channel Capability and Configuration registers on the eSPI slaves. The SET_CONFIGURATION command phase consists of an 8-bit Command Opcode, 16-bit address, 1 DW of Data and an optional 8-bit CRC. The response phase includes the 8-bit Response, 16-bit Status and an optional 8-bit CRC.

The eSPI slave contains addressable register space up to 4KB. The access is addressed at DWord boundary and only the lower 12-bits of the 16-bit address are used with address bit[1:0] hard-wired to always “00”. The 4 MSB address bits must be driven to all zeros by eSPI master. eSPI slaves should ignore the 4 MSB address bits.

Note: Implementation Note: Upon coming out of eSPI Reset#, eSPI master can initiate a GET_CONFIGURATION cycle to a particular eSPI slave to determine if the eSPI slave is present. If the eSPI slave is not present, the eSPI data lines remain pulled-up after the Turn-Around time. eSPI master can use this behavior to deduce that the eSPI slave is not present on the bus.

If the eSPI slave is present, the eSPI slave must drive the response phase upon the expiry of the Turn-Around time.

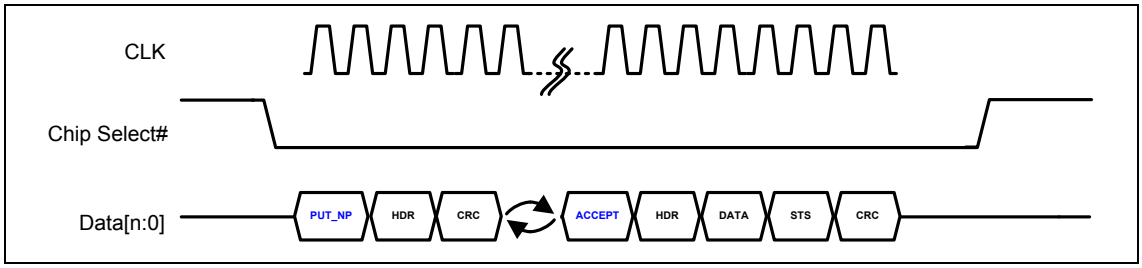
4.8 Non-Posted Transaction

eSPI master initiated non-posted transaction can be terminated as connected or deferred completion.

The eSPI master initiated non-posted transaction is terminated as a connected completion when the data and all the information needed to generate the response are immediately available.

The valid responses for non-posted transactions terminated as connected include ACCEPT, FATAL ERROR and NON-FATAL ERROR.

Figure 24: Connected Master Initiated Non-Posted Transaction

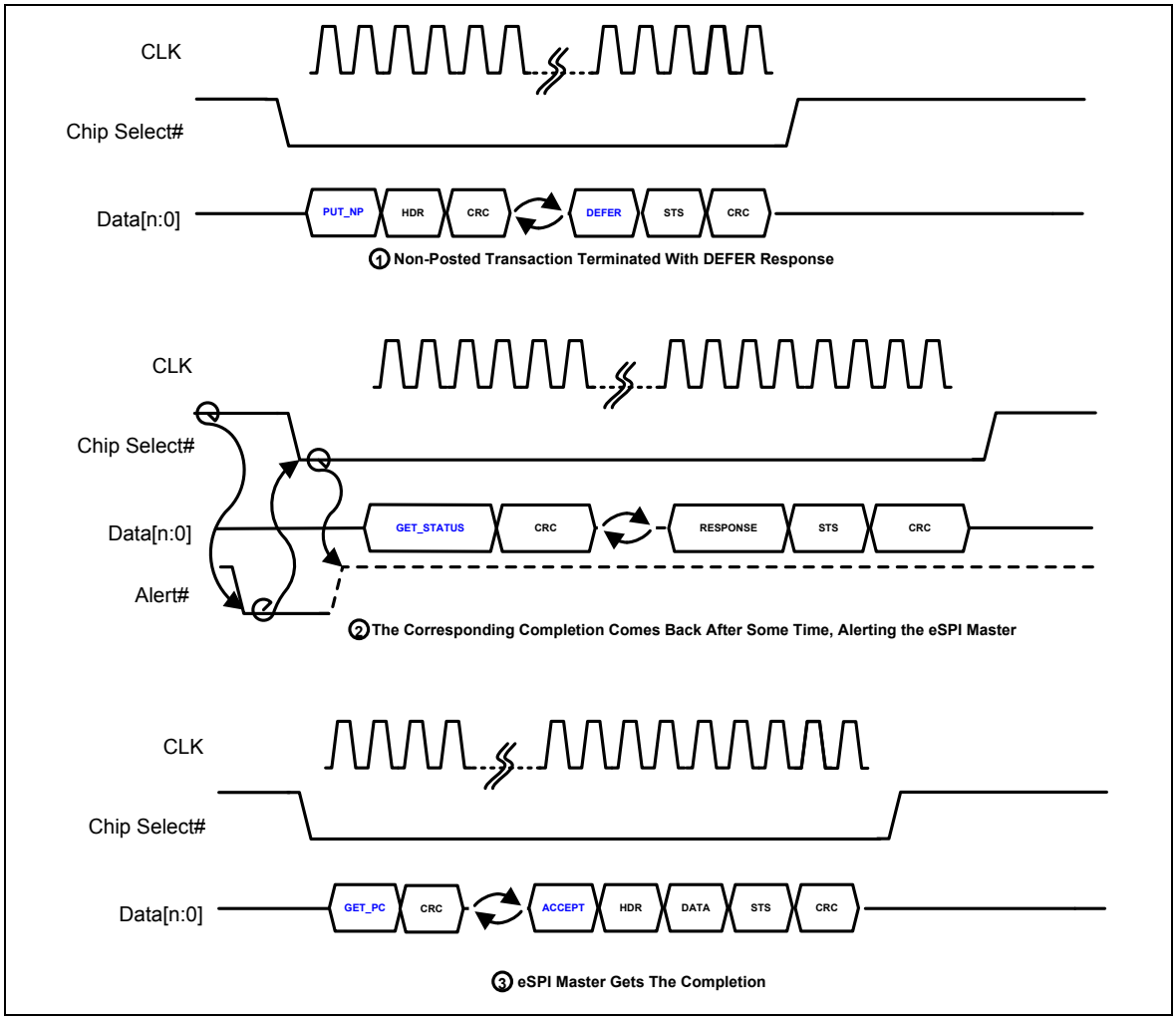


If the eSPI master initiated non-posted completion requires data or additional information which is not available immediately, the non-posted request is terminated with a "DEFER" response. The deferred completion can be returned some period of time in the future when the data or information is eventually available. The bus can be used for other transactions prior to the defer completion being returned, as long as the ordering rule is maintained.

When the deferred completion is returned, the valid responses include ACCEPT, FATAL ERROR and NON-FATAL ERROR.

The eSPI slave can complete the non-posted command with multiple split completions. If one of the split completions has an unsuccessful completion status, the remaining split completions will not be returned.

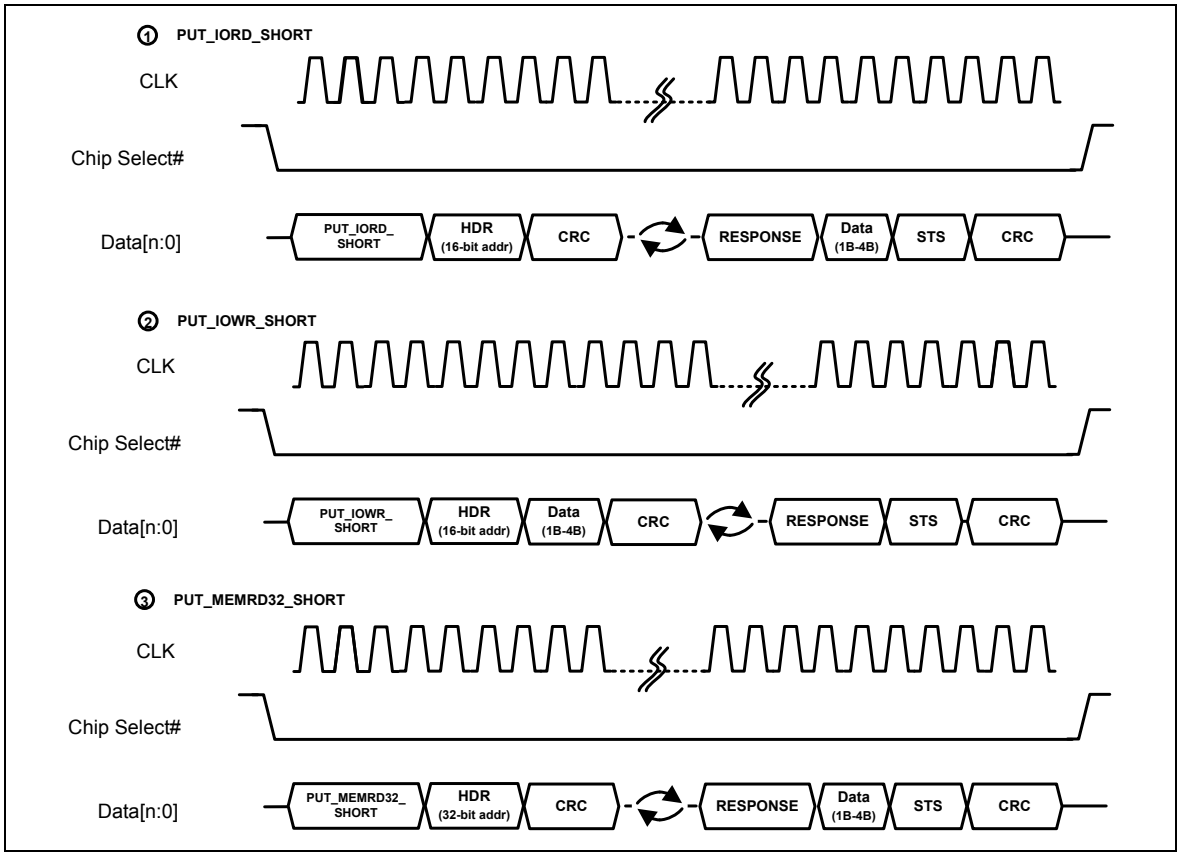
Figure 25: Deferred Master Initiated Non-Posted Transaction



eSPI supports short non-posted transactions from master to slave for requests of length 1-4 bytes that have less overhead and are thus more efficient. The unique opcode indicates the type of non-posted transaction and the request length. The header contains the address only and the number of address bytes for the transaction is implied by the opcode. The short non-posted transaction does not have the Tag field. The Tag field is implied as all 0's which will be returned by the slave in the completion header.

The short non-posted transactions can be terminated as connected or deferred completion. However, for optimized performance of short transaction, the slave should complete the transaction as connected where it can.

Figure 26: Master Initiated Short Non-Posted Transaction

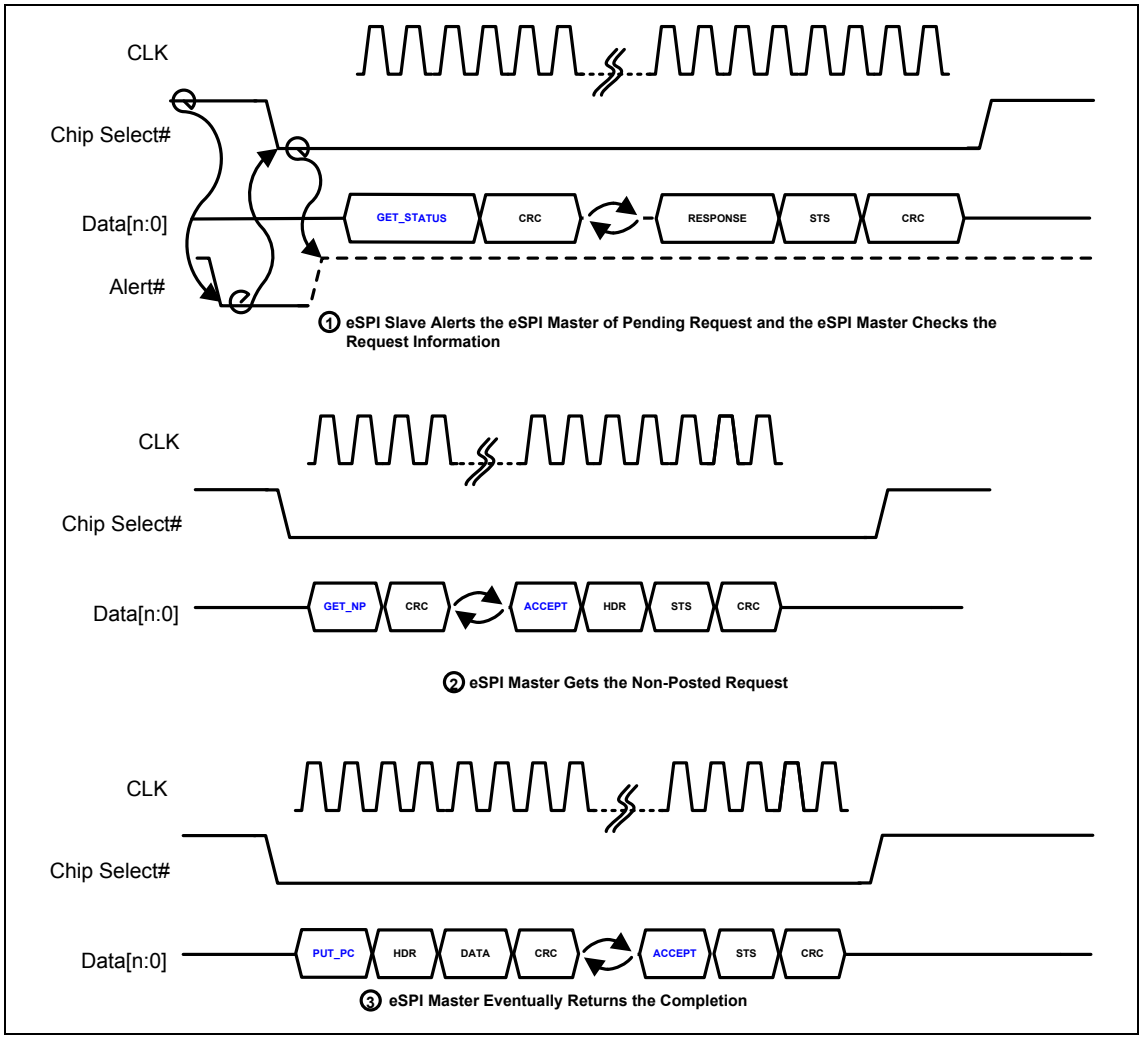


The eSPI slave can generate an Alert when there is a pending non-posted transaction. In response to that, the eSPI master would issue a GET_STATUS command to check for the pending request information.

The eSPI master would then generate a GET_NP command to fetch the non-posted transaction. Once the completion data and the information needed to return the response is available, the eSPI master would return the split completion back to the eSPI slave.

Completions to a non-posted request initiated by the eSPI slave are always split.

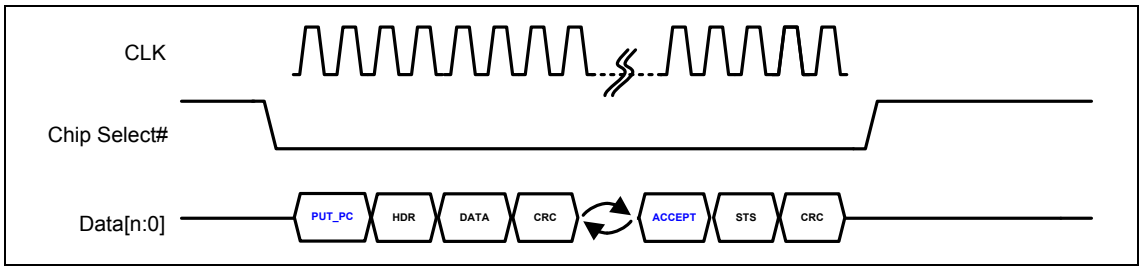
Figure 27: Slave Initiated Non-Posted Transaction



4.9 Posted Transaction

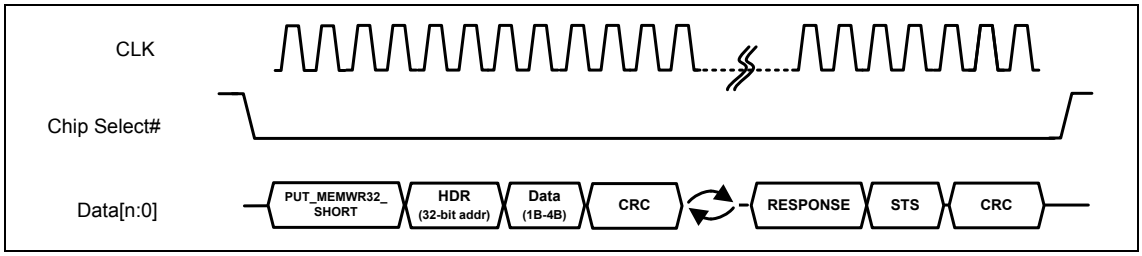
The valid responses for posted transactions initiated by eSPI master are ACCEPT, FATAL ERROR and NON-FATAL ERROR. DEFER response for posted transaction is invalid.

Figure 28: Master Initiated Posted Transaction



eSPI supports short posted transactions from master to slave for requests of length 1-4 bytes that have less overhead and are thus more efficient. The unique opcode indicates the short posted transaction and the request length. The header contains the address only and the number of address bytes for the transaction is implied by the opcode.

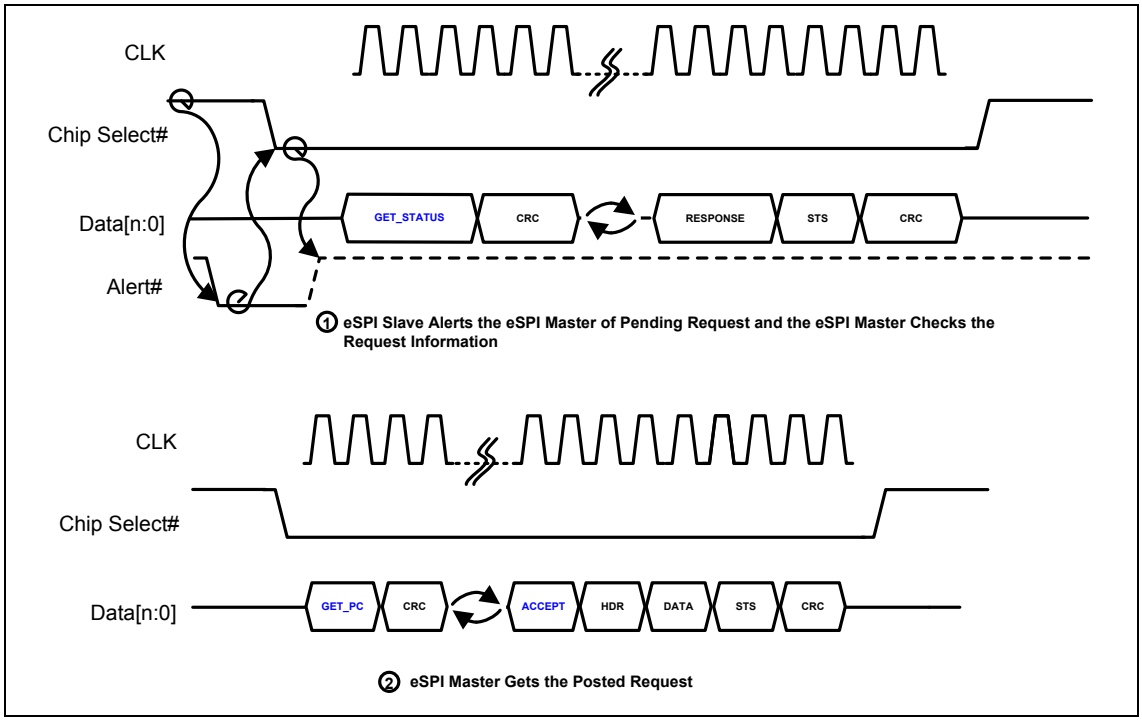
Figure 29: Master Initiated Short Posted Transaction



The eSPI slave can generate an Alert when there is a pending posted transaction. In response to that, the eSPI master would issue a GET_STATUS command to check for the pending request information.

The eSPI master would then generate a GET_PC command to get the posted transaction.

Figure 30: Slave Initiated Posted Transaction



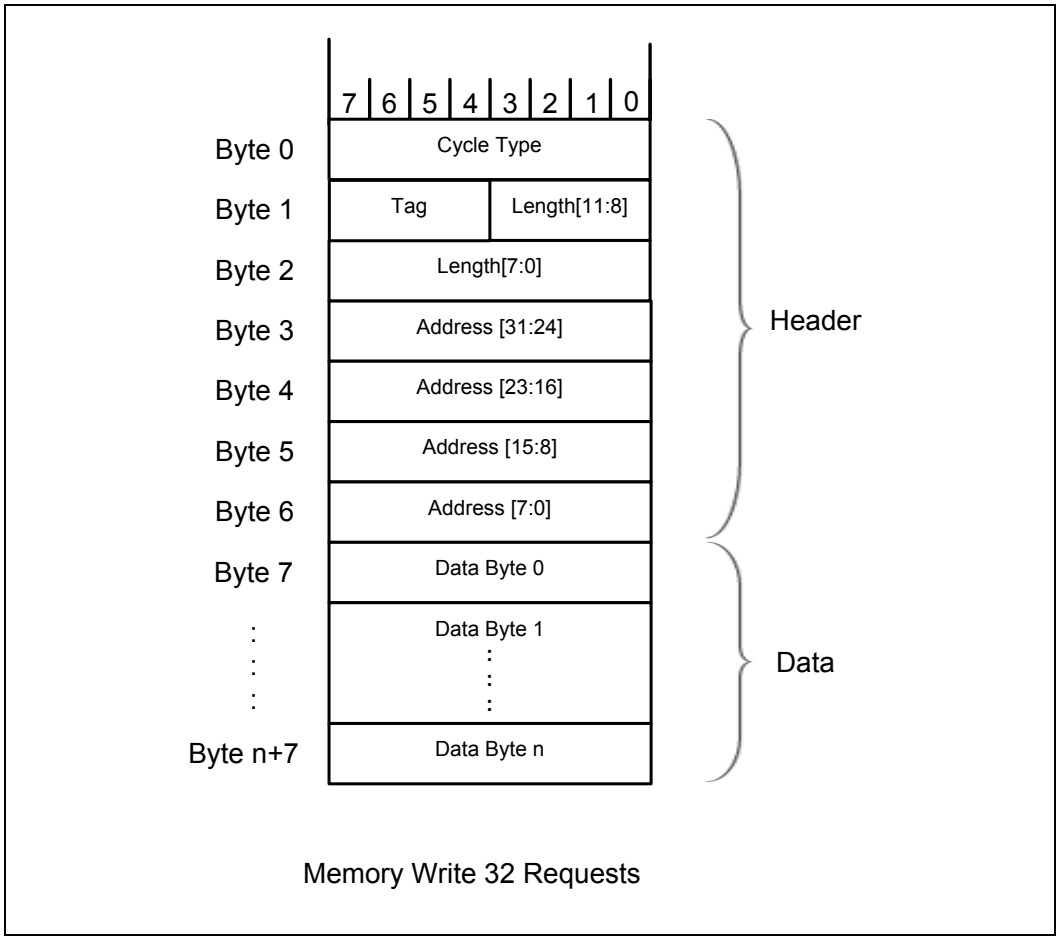
§ §

5 Transaction Layer

5.1 Cycle Types and Packet Format

The following diagram shows a general Enhanced Serial Peripheral Interface (eSPI) packet format. The description of the respective fields within the packet is described in the subsequent sections.

Figure 31: General eSPI Packet Format



5.1.1 Cycle Types

The summary of cycle types supported over the eSPI interface is shown in the table below. The Least-Significant-Bit (LSB) of the encodings distinguishes between a cycle with data and a cycle without data.

The direction of cycle type supported is specified in the table as "Up" or "Down". "Up" refers to the direction from eSPI slave to eSPI master and "Down" refers to the direction from eSPI master to eSPI slave.

Table 5: Cycle Types

Cycle Type	Encodings[3] [7:0]	Direction	Command Type	Channel Type	Description
eSPI Peripheral Channel					
Memory Read 32	00000000	Up/Down	Non-Posted	eSPI Peripheral Channel	32 bit addressing Memory Read Request. LPC Memory Read and LPC Bus Master Memory Read requests are mapped to this cycle type.
Memory Read 64	00000010	Up/Down	Non-Posted	eSPI Peripheral Channel	64 bit addressing Memory Read Request. LPC Memory Read and LPC Bus Master Memory Read requests are mapped to this cycle type.
Memory Write 32	00000001	Up/Down	Posted	eSPI Peripheral Channel	32 bit addressing Memory Write Request. LPC Memory Write and LPC Bus Master Memory Write requests are mapped to this cycle type.
Memory Write 64	00000011	Up/Down	Posted	eSPI Peripheral Channel	64 bit addressing Memory Write Request. LPC Memory Write and LPC Bus Master Memory Write requests are mapped to this cycle type.
I/O Read	00000100	Down	Non-Posted	eSPI Peripheral Channel	I/O Read Request. LPC I/O Read is mapped to this cycle type.
I/O Write	00000101	Down	Non-Posted	eSPI Peripheral Channel	I/O Write Request. LPC I/O Write is mapped to this cycle type.
Successful Completion Without Data	00000110	Up/Down	Completion	eSPI Peripheral Channel	Successful Completion Without Data. Corresponds to I/O Write, Flash Write or Flash Erase.
		Down		Flash Access Channel	

Cycle Type	Encodings[3] [7:0]	Direction	Command Type	Channel Type	Description
Successful Completion With Data	00001P_1P_0[1]1	Up/Down	Completion	eSPI Peripheral Channel	Successful Completion With Data. Corresponds to Memory Read, I/O Read or Flash Read.
		Down		Flash Access Channel	
Unsuccessful Completion Without Data	00001P_1P_0[1,2]0	Up/Down	Completion	eSPI Peripheral Channel	Unsuccessful Completion Without Data. Corresponds to Memory, I/O or Flash accesses.
		Down		Flash Access Channel	
OOB Message Channel					
OOB (Tunneled SMBus) Message	00100001	Up/Down	Posted	OOB Message Channel	SMBus Out-Of-Band Message. SMBus packet tunneling.
Flash Access Channel[4]					
Flash Read	00000000	Up/Down	Non-Posted	Flash Access Channel	Read from Flash.
Flash Write	00000001	Up/Down	Non-Posted	Flash Access Channel	Write to Flash.
Flash Erase	00000010	Up/Down	Non-Posted	Flash Access Channel	Flash Erase instruction. Erase part or the whole partition owned by the corresponding flash master.

Note:

1. The encoding P_1P_0 has the following definition:

Encoding P_1P_0	Description
00	Indicates the middle completion of a split completion sequence.
01	Indicates the first completion of a split completion sequence
10	Indicates the last completion of a split completion sequence.
11	Indicates the only completion for a split transaction.

2. For Unsuccessful Completion without Data, P_1 must be always a '1' as this is always the last or the only completion.

3. The combination of command opcode and cycle type encoding must be unique. There is no requirement that cycle type encodings must be unique across command opcodes.

4. Refer to Section 5.2.4 for detail operation of the Flash Access channel.

5.1.2 Tag

The Tag field is allowed to be non-unique for multiple outstanding non-posted requests on the same Channel that require completion.

Refer to Section 5.4 - Transaction Ordering Rule for more details about Tag and its association with the ordering of completions.

For posted requests which do not require completion, the usage of Tag field is implementation specific and beyond the scope of the specification.

5.1.3 Length

The length field indicates the request size or data payload specified in Bytes. The length field is 1-based. A value of all zeros indicates 4 KB of length.

For memory read, I/O read and Flash Read, the length field specifies the data payload size requested.

For memory write, I/O write, Flash write, OOB message with data and Completion with Data, the length field specifies the actual amount of data returned in the packet.

For Completion without Data or Un-Successful Completion, the length field must be driven to zeros by initiator. The receiver must ignore the length field.

I/O read and I/O write must not be more than 4 Bytes in length. eSPI slave must respond with Fatal Error if the request length exceeds 4 Bytes.

For Memory Write, Flash Write, OOB message with data and Completion, data payloads size should not exceed the Maximum Payload Size of the respective channels.

Read requests size initiated by eSPI slave on eSPI Peripheral Channel and Flash Access Channel should not exceed the Maximum Read Request Size set by the eSPI master in the Channel Capability and Configuration register for the respective channels.

Memory read and Flash read requests may be completed with one or multiple split completions.

A completion that does not cross 64B read completion boundary must be completed as a single transaction.

If the completion crosses 64B read completion boundary, the completion must be returned in multiple split completions, with each completion aligns to the 64B naturally

aligned address boundary except for the first completion which aligns to the starting address of the request.

For successful completion with data and unsuccessful completion without data, the additional cycle type encoding indicates whether the completion is the first, middle or the last completion for a split completion sequence, or whether it is the only completion that completes the split transaction.

5.1.4 Address

The eSPI memory transactions support both 32 bits and 64 bits addressing formats. For I/O cycles, only 16-bits address is used.

5.1.5 Data

The valid data field always starts at Byte 0, regardless of the address alignment. There is no byte enables associated with data. It is the responsibility of the requester to break the requests which are targeting non-contiguous locations into separate requests.

5.2 Channels

A channel provides a means to allow multiple independent flows of traffic to share the same physical bus.

Each set of the put_*/get_*/*_avail/*_free associates with the command and response of a corresponding channel.

Each of the channels has its dedicated resources such as queue and flow control. There is no ordering requirement between traffics from different channels.

The number and types of channels supported by a particular eSPI slave is discovered through the GET_CONFIGURATION command issued by the eSPI master to the eSPI slave during initialization.

The assignment of the channel type to the channel number is fixed. The eSPI slave can only advertise which of the channels are supported.

eSPI Peripheral transactions always use Channel 0. The PUT_PC/ PUT_NP/ GET_PC/ GET_NP/ PC_FREE/ NP_FREE/ PC_AVAIL/ NP_AVAIL commands and status fields are used for Channel 0 access.

Virtual Wires are communicated through Channel 1. The PUT_VWIRE/ GET_VWIRE/ VWIRE_AVAIL commands and status fields are used for Channel 1 access.

OOB Message and Flash Access use channel 2 and channel 3 respectively.

Commands such as GET_STATUS, SET_CONFIGURATION and GET_CONFIGURATION are not associated with any particular channel.

5.2.1 Peripheral Channel

eSPI Peripheral channel is used for communication between eSPI host bridge located on the master side and eSPI endpoints located on the slave side. LPC Host and LPC Peripherals are an example of eSPI host bridge and eSPI endpoints respectively. Other examples include ACPI devices connected to the eSPI bus which talk to a host controller residing on the eSPI master side. The eSPI Peripherals are not software discoverable.

The format of the eSPI Peripheral Memory and I/O request packet and completions are shown below.

The Tag and Channel information are used to match the completions with the corresponding requests.

Figure 32: Peripheral Memory or I/O Write Packet Format

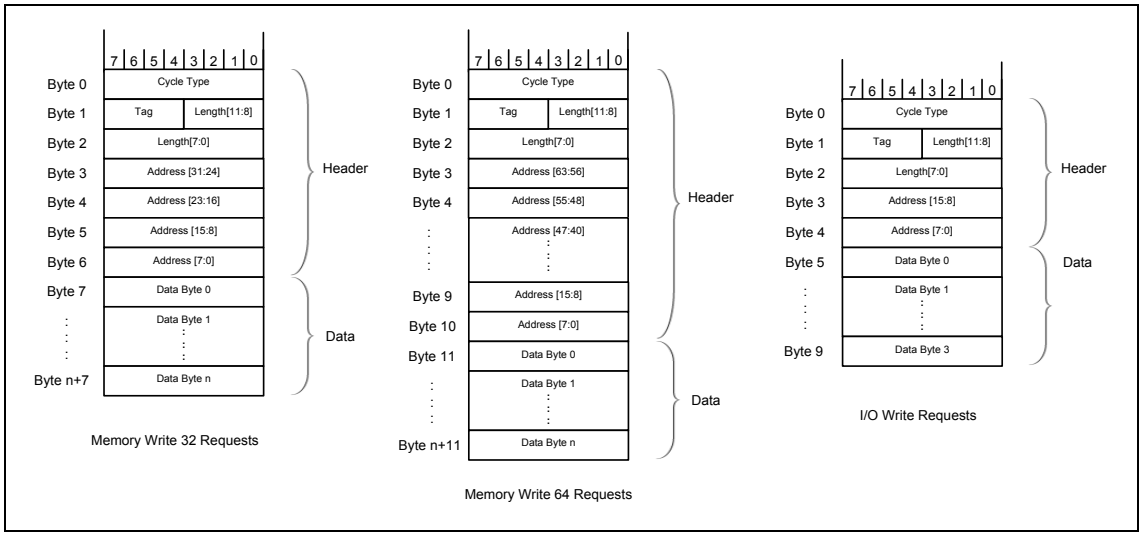


Figure 33: Short Peripheral Memory or Short I/O Write Packet Format (Master Initiated only)

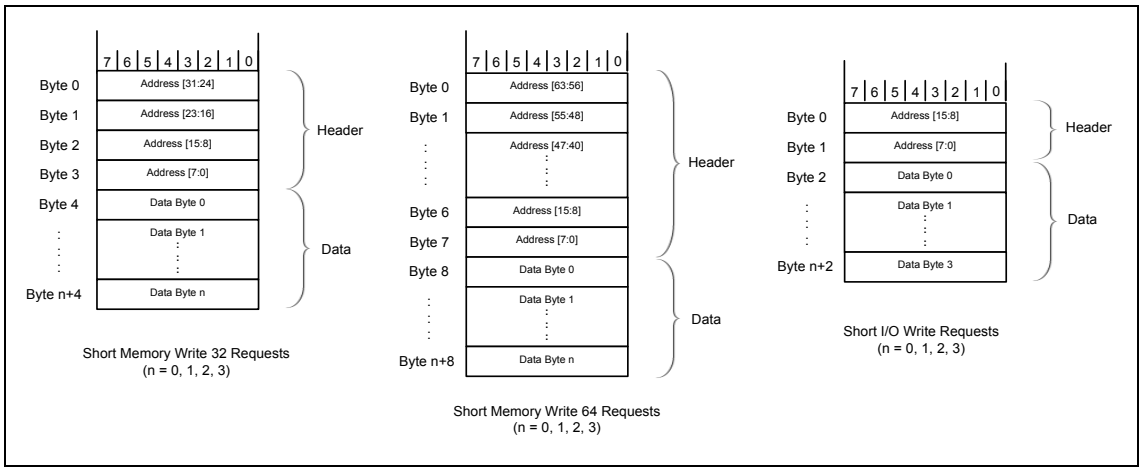


Figure 34: Peripheral Memory or I/O Read Packet Format

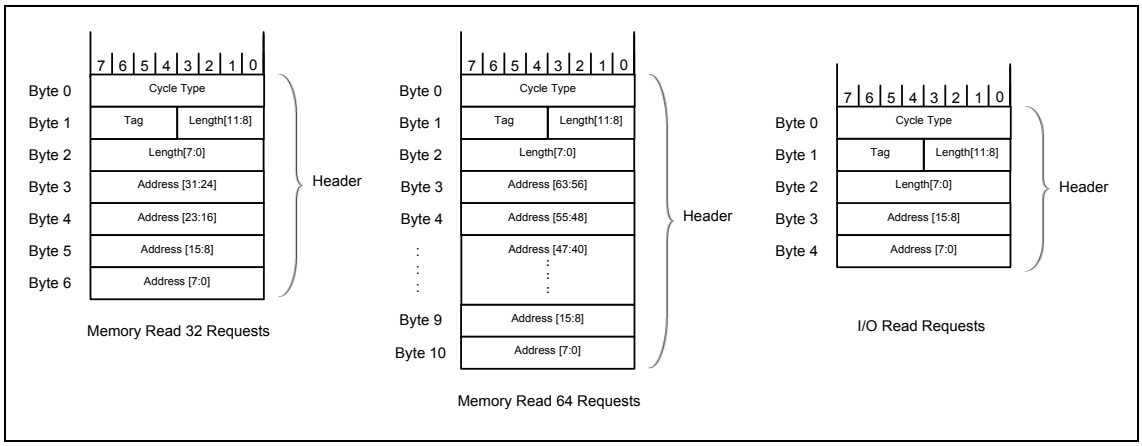


Figure 35: Short Peripheral Memory or Short I/O Read Packet Format (Master Initiated only)

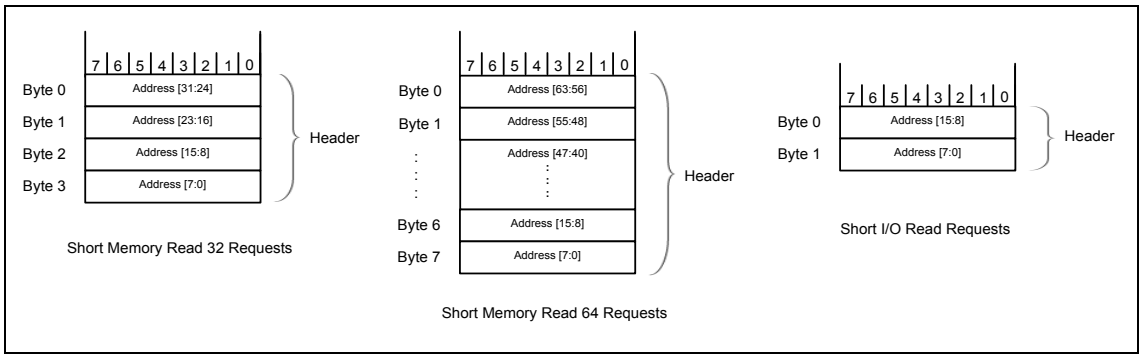
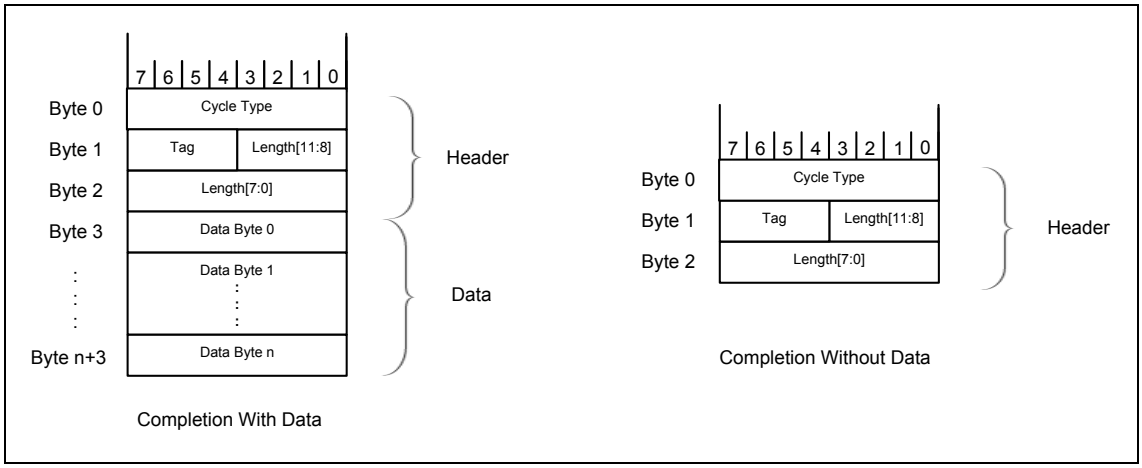


Figure 36: Peripheral Memory or I/O Completion With and Without Data Packet Format



Memory Read and Write requests can be initiated by both eSPI master and slave. I/O Read and Write requests, and Short Memory Read and Write requests can only be initiated by eSPI master.

The eSPI packet format does not contain Byte Enables. In the case where the data to be written are non-contiguous, the requester is responsible to break the request into several contiguous sub-requests.

5.2.2 Virtual Wires Channel

The Virtual Wire channel is used to communicate the state of Sideband pins or GPIO tunneled through eSPI as in-band messages. Serial IRQ interrupts are communicated through this channel as in-band messages.

The Command phase consists of Command Opcode, Virtual Wire Packet and an optional CRC.

The Virtual Wire Packet begins with the Virtual Wire Count as the header byte where the count indicates the number of Virtual Wire groups communicated by the packet. It is then followed by one or more Virtual Wire groups. Each of the Virtual Wire group consists of 2 bytes, namely a Virtual Wire Index and a Virtual Wire Data. Multiple Virtual Wire groups up to 64 groups are allowed to be sent in the same packet.

The definition of the Virtual Wire Count is as follow:

Bits	Description
7:6	Reserved
5:0	**Count:** The 6-bit count field allows up to 64 Virtual Wire groups to be communicated in the same packet. This is a 0-based count.

The number of Virtual Wire groups in a single Virtual Wire packet must not exceed the Operating Maximum Virtual Wire Count configured in the Channel 1 Capability and Configuration register. This applies to any Virtual Wire packet initiated by eSPI master or eSPI slaves.

The Virtual Wire Index points to one of the many pre-defined groups of Virtual Wires. The format of the Virtual Wire Data is specific to the corresponding Virtual Wire Index. It contains information specific to the Virtual Wire group that it is communicating.

The unused virtual wire slots within a particular Virtual Wire Index would be made Reserved. The initiator must drive the Reserved field to '0' and the receiver must ignore the Reserved field.

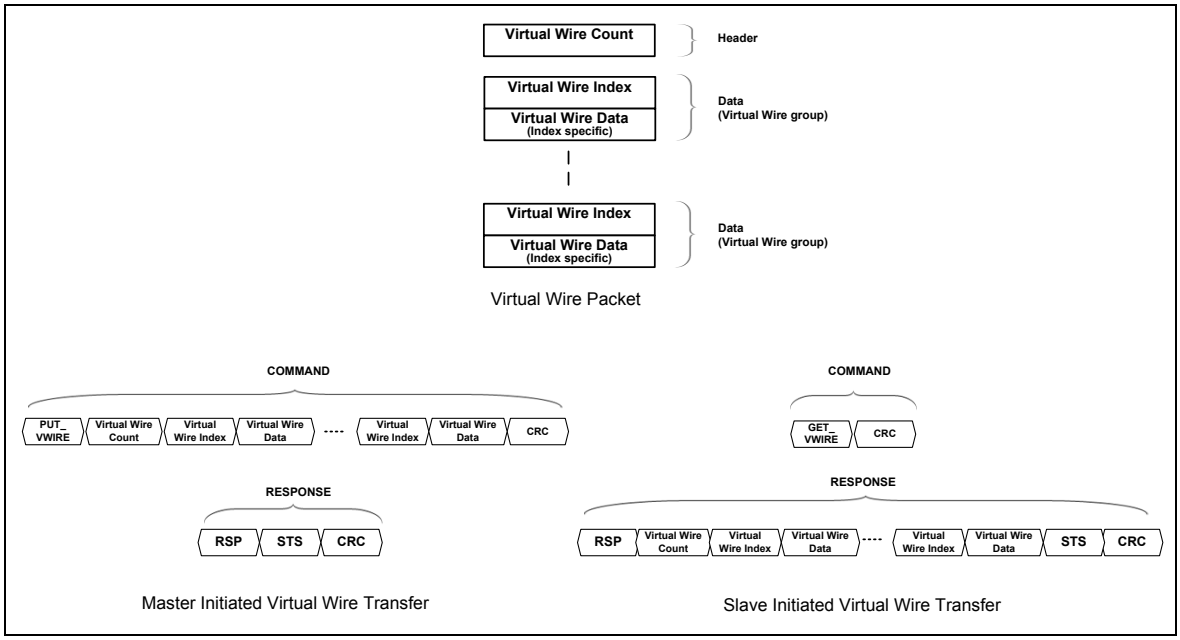
Virtual Wire packets are not subjected to flow control. The receiver must always be ready to receive the Virtual Wire packets at any time, as long as the channel is enabled.

The length of the Virtual Wire Packet is (1 + 2*n) bytes where n is the number of Virtual Wire groups in the packet.

Virtual Wire should be given the highest priority over traffic from other channels to ensure that the latency is kept to a minimum.

The diagram below shows the Virtual Wire packet format.

Figure 37: Virtual Wire Packet Format



The components on both sides of the bus must track the logical state of the individual Virtual Wires. When the logical state of the Virtual Wire changes, the new state must be communicated to the other component connected to the same bus using the appropriate Virtual Wire messages.

Some of the Virtual Wires such as PME and Interrupt can be shared by multiple eSPI slaves. The eSPI master must track the logical state of the individual Virtual Wires independently for each of the eSPI slaves and is responsible to aggregate the different sets of Virtual Wires.

The messages from eSPI master to slave are unicast only, meaning they can only target one particular eSPI slave.

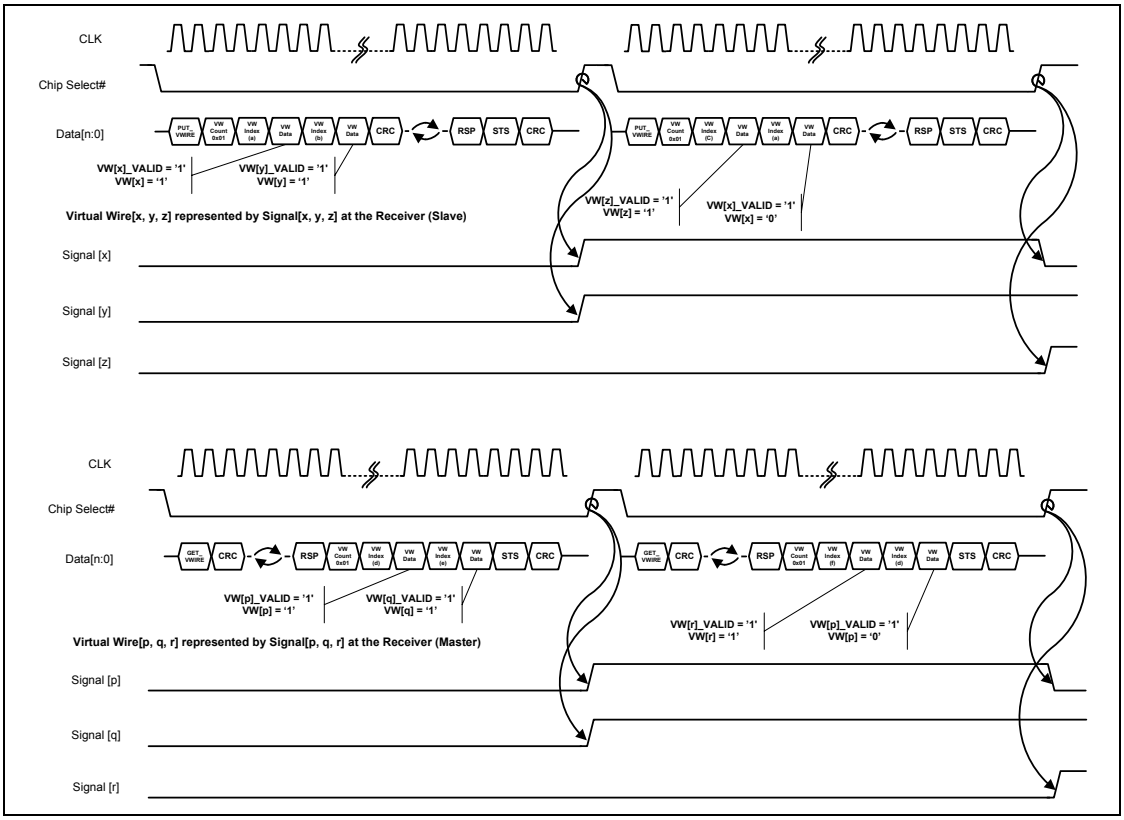
When PLTRST message is received over the link, indicating platform reset, the individual Virtual Wires reset by Platform Reset should be initialized to the default value.

When virtual wires change state due to reset (either going into reset or exiting reset), a new message will not be sent to notify the other component of the state change as both sides would be looking at the same reset.

If an un-supported Virtual Wire message is received, the receiver should drop the message silently.

The Virtual Wires tunneled through eSPI will only take effect at the receiver when the Chip Select# is deasserted at the end of the transaction. The initiator can only assume the Virtual Wires are sent at the deassertion edge of the Chip Select#.

Figure 38: Virtual Wires at the Receiver



5.2.2.1 Virtual Wire Index

The following table defines the Virtual Wire Index, the corresponding pre-defined Virtual Wire groups and the associated Virtual Wire Data formats.

Table 6: Virtual Wire Index Definition

<table>
<tr><th colspan="2">Virtual Wire Index</th><th rowspan="2">Virtual Wire Group</th><th rowspan="2">Virtual Wire Data Format</th></tr>
<tr><th>Start</th><th>End</th></tr>
<tr><td>0</td><td>1</td><td>Interrupt event</td><td>
<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>L</td><td colspan="7">IRQ Line</td></tr>
</table>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7</td><td>Interrupt Level:
0b: Low
1b: High</td></tr>
<tr><td>6:0</td><td>Interrupt Line: Specify the interrupt (IRQ) line to be sent to the interrupt controller.
Index=0h: IRQ 0 – 127
Index=1h: IRQ 128 – 255</td></tr>
</table>
Interrupt event virtual wires are defined from slave to master only.
</td></tr>
<tr><td>2</td><td>7</td><td>System Event</td><td>
<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="4">Valid</td><td colspan="4">Level</td></tr>
</table>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7:4</td><td>Valid: This field indicates the validity of the 1-to-1 corresponding Level bits.
0b: Not valid
1b: Valid</td></tr>
<tr><td>3:0</td><td>Level: Each of the bits in this field indicates the state of a virtual wire signal to be communicated.
0b: Low
1b: High</td></tr>
</table>
Note: The Valid field is to handle the case where the Virtual Wire may be located in a shallower power well compared to another Virtual Wire in the same group and may not be valid at the time the Virtual Wire message is sent.
</td></tr>
</table>

<table>
<tr><th colspan="2">Virtual Wire Index</th><th>Virtual Wire Group</th><th>Virtual Wire Data Format</th></tr>
<tr><td>8</td><td>63</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>64</td><td>127</td><td>Platform specific</td><td>Platform specific</td></tr>
<tr><td>128</td><td>255</td><td>General Purpose I/O Expander</td><td>
<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="4">Valid</td><td colspan="4">Level</td></tr>
</table>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7:4</td><td>Valid: This field indicates the validity of the 1-to-1 corresponding Level bits.
0b: Not valid
1b: Valid</td></tr>
<tr><td>3:0</td><td>Level: Each of the bits in this field indicates the state of a virtual GPIO to be communicated.
0b: Low
1b: High</td></tr>
</table>
Note: The Valid field is to handle the case where the Virtual Wire may be located in a shallower power well compared to another Virtual Wire in the same group and may not be valid at the time the Virtual Wire message is sent.
</td></tr>
</table>

5.2.2.2 System Event Virtual Wires

The eSPI specification defines the following system event Virtual Wires covering the platform independent standard sideband signals.

If supported, these standard virtual wires must be implemented in accordance to the specification to enable inter-operability and compatibility. However, the support of these Virtual Wires is platform specific.

Platform specific Virtual Wires are allowed by the specification with Virtual Wire Index 64 to 127. They are defined in the respective platform specific documents and outside the scope of the specification.

Table 7: System Event Virtual Wires for Index=2

Virtual Wire Index	**2**
Virtual Wire Group	**System Event**
Reset	**eSPI Reset#**
Direction	**Master to Slave**

Bit	Virtual Wire	Description
7		Reserved
6		**SLP_S5# Valid:** This bit indicates the validity of SLP_S5# virtual wire on bit[2]. '0' – Not valid '1' – Valid
5		**SLP_S4# Valid:** This bit indicates the validity of SLP_S4# virtual wire on bit[1]. '0' – Not valid '1' – Valid
4		**SLP_S3# Valid:** This bit indicates the validity of SLP_S3# virtual wire on bit[0]. '0' – Not valid '1' – Valid
3	RSV	Reserved
2	SLP_S5#	**S5 Sleep Control:** Sent when the power to non-critical systems should be shut off in S5 (Soft Off). Polarity: Active low. Reset: Active.
1	SLP_S4#	**S4 Sleep Control:** Sent when the power to non-critical systems should be shut off in S4 (Suspend to Disk). Polarity: Active low. Reset: Active.
0	SLP_S3#	**S3 Sleep Control:** Sent when the power to non-critical systems should be shut off in S3 (Suspend to RAM). Polarity: Active low. Reset: Active.

Table 8: System Event Virtual Wires for Index=3

Virtual Wire Index	**3**
Virtual Wire Group	**System Event**
Reset	**eSPI Reset#**
Direction	**Master to Slave**

Bit	Virtual Wire	Description
7		Reserved
6		Reserved
5		**PLTRST# Valid:** This bit indicates the validity of PLTRST# virtual wire on bit[1]. '0' – Not valid '1' – Valid
4		**SUS_STAT# Valid:** This bit indicates the validity of SUS_STAT# virtual wire on bit[0]. '0' – Not valid '1' – Valid
3	RSV	Reserved
2	RSV	Reserved
1	PLTRST#	**Platform Reset:** Command to indicate Platform Reset assertion and de-assertion. Polarity: Active low. Reset: Active.
0	SUS_STAT#	**Suspend Status:** Sent when the system will be entering a low power state soon. Polarity: Active low. Reset: Inactive.

Table 9: System Event Virtual Wires for Index=4

Virtual Wire Index	**4**
Virtual Wire Group	**System Event**
Reset	**eSPI Reset#**
Direction	**Slave to Master**

Bit	Virtual Wire	Description
7		**PME# Valid:** This bit indicates the validity of PME# virtual wire on bit[3]. '0' – Not valid '1' – Valid
6		**WAKE# Valid:** This bit indicates the validity of WAKE# virtual wire on bit[2]. '0' – Not valid '1' – Valid
5		**BATLOW# Valid:** This bit indicates the validity of BATLOW# virtual wire on bit[1]. '0' – Not valid '1' – Valid
4		**PWRBTN# Valid:** This bit indicates the validity of PWRBTN# virtual wire on bit[0]. '0' – Not valid '1' – Valid
3	PME#	**PCI Power Management Event:** eSPI slaves generated PCI PME# event. Shared by multiple eSPI endpoints. Polarity: Active low. Reset: Inactive.
2	WAKE#	**Wake#:** Used by the slave to wake the Host from Sx on any event; also general purpose event to wake on LID switch or AC insertion, etc. If the event occurs while system is in S0, a SCI is generated instead. Polarity: Active low. Reset: Inactive.
1	BATLOW#	**Battery Low:** Sent when the battery power is too low to support wake from S1-S5. Polarity: Active low. Reset: Inactive.

Bit	Virtual Wire	Description
0	PWRBTN#	**Power Button:** Sent to indicate that the platform power button has been pressed by the user. Causes the system to go to a sleep state. If the system is already in a sleep state, will cause a wake event. Polarity: Active low. Reset: Inactive.

Table 10: System Event Virtual Wires for Index=5

Virtual Wire Index	**5**
Virtual Wire Group	**System Event**
Reset	**eSPI Reset#**
Direction	**Slave to Master**

Bit	Virtual Wire	Description
7		Reserved
6		**ERROR_NONFATAL Valid:** This bit indicates the validity of ERROR_NONFATAL virtual wire on bit[2]. '0' – Not valid '1' – Valid
5		**ERROR_FATAL Valid:** This bit indicates the validity of ERROR_FATAL virtual wire on bit[1]. '0' – Not valid '1' – Valid
4		**BOOT_DONE Valid:** This bit indicates the validity of BOOT_DONE virtual wire on bit[0]. '0' – Not valid '1' – Valid
3	RSV	Reserved
2	ERROR_NONFATAL	**Error Non-Fatal:** Sent when a non-fatal error is detected which is not due to eSPI transaction on the bus. Note: Non-fatal error due to transaction on the bus will be signaled through the response phase. Polarity: Active high. Reset: Inactive.

Bit	Virtual Wire	Description
1	ERROR_FATAL	**Error Fatal:** Sent when a fatal error is detected which is not due to eSPI transaction on the bus. Note: Fatal error due to transaction on the bus will be signaled through the response phase. Polarity: Active high. Reset: Inactive.
0	BOOT_DONE	**Boot Done:** Sent when EC or BMC has completed its boot process as indication to eSPI master to continue with the G3 to S0 exit. eSPI master waits for the assertion of this virtual wire before proceeding with the SLP_S5# deassertion. Polarity: Active high. Reset: Inactive.

Table 11: System Event Virtual Wires for Index=6

Virtual Wire Index	**6**
Virtual Wire Group	**System Event**
Reset	**PLTRST#**
Direction	**Slave to Master**

Bit	Virtual Wire	Description
7		Reserved
6		**RCIN# Valid:** This bit indicates the validity of RCIN# virtual wire on bit[2]. '0' – Not valid '1' – Valid
5		**SMI#:** This bit indicates the validity of SMI# virtual wire on bit[1]. '0' – Not valid '1' – Valid
4		**SCI# Valid:** This bit indicates the validity of SCI# virtual wire on bit[0]. '0' – Not valid '1' – Valid
3	RSV	Reserved

Bit	Virtual Wire	Description
2	RCIN#	**Keyboard Controller CPU Reset:** Sent to request CPU reset on behalf of the keyboard controller. Polarity: Active low. Reset: Inactive.
1	SMI#	**System Management Interrupt (SMI):** Sent as general purpose alert resulting in SMI code being invoked by the BIOS. Polarity: Active low. Reset: Inactive.
0	SCI#	**System Controller Interrupt (SCI):** Sent as general purpose alert resulting in ACPI method being invoked by the OS. Polarity: Active low. Reset: Inactive.

5.2.2.3 Communicating Timing Event on Virtual Wires

Some of the events communicated through the Virtual Wire Channel could be timing events.

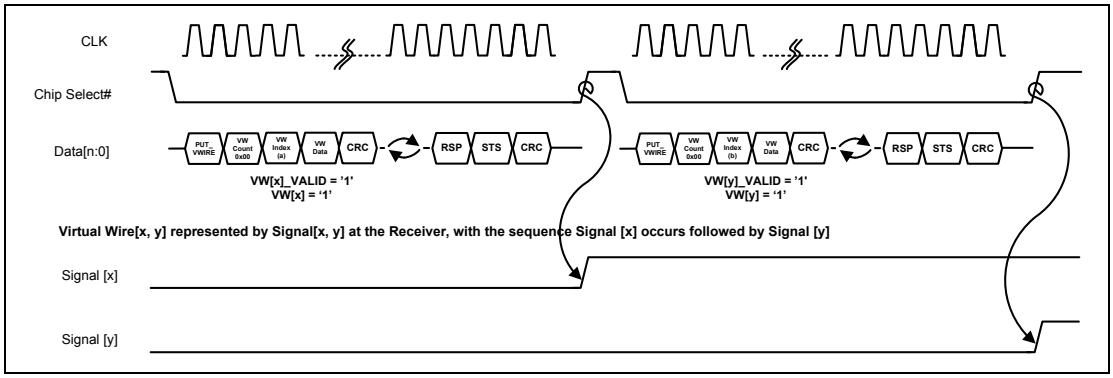
For example, the assertion of a particular pin could indicate one event. The prolonged assertion of the same pin for a certain period of time could indicate a different event.

One solution is to send two different messages, for each of the events; one message is sent when the pin asserts and another message is sent when the pin has been asserted for a certain period of time.

This method requires the source of the message to implement a timer that times the duration of the pin assertion, which upon time-out, results in a message being sent.

Multiple Virtual Wires communicated in the same packet will change state at the receiver the same time at the deassertion edge of the Chip Select#. If the sequence of the Virtual Wires is required to be communicated, the Virtual Wire that happens later must be communicated in the next Chip Select# assertion to signify the sequence.

Figure 39: Virtual Wires with Sequence Communicated



5.2.2.4 Interrupt Event

Interrupt event is supported from eSPI slave to master through the virtual wire channel. Virtual Wire Index 0 and 1 are defined for communicating interrupt events and up to 256 IRQ lines can be communicated in-band over the eSPI bus.

Both level-triggered and edge-triggered interrupt are supported.

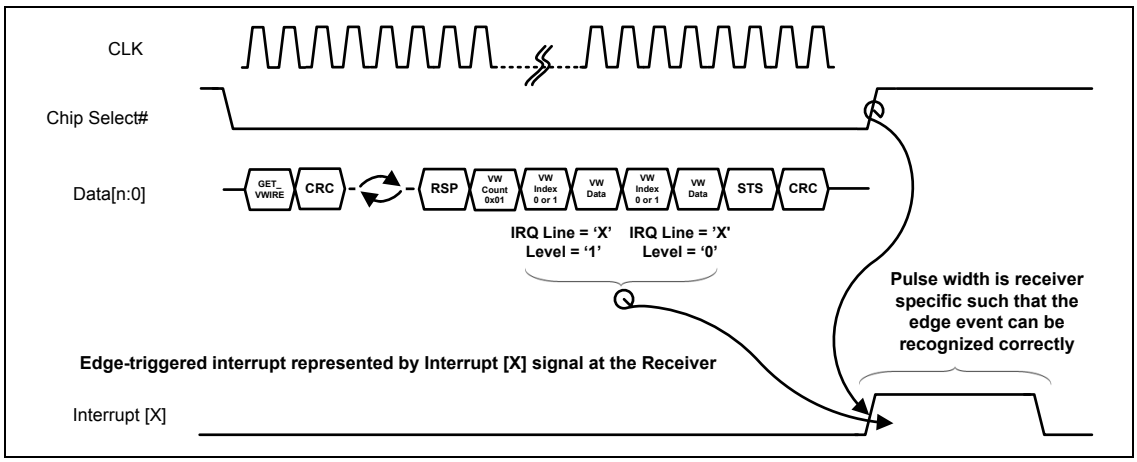
For level-triggered interrupt, the level of the interrupt line is communicated thru the virtual wire whenever there is a change to the state of the interrupt line from '0' to '1' or vice versa. The interrupt line allocated must be configured to the slave during initialization. Multiple interrupt sources on the slave are allowed to be shared on a single level-triggered interrupt line. The shared interrupt line will not change state until all the pending interrupts are serviced which will then trigger an interrupt event virtual wire being sent by the slave. The exact method of interrupt line allocation and interrupt sharing is platform specific and outside the scope of the specification.

To avoid spurious interrupts when using level-triggered interrupts, it is recommended that the software driver and the eSPI slave implement the following behavior: When the driver has completed the interrupt service routine, it should issue a posted memory write to the eSPI slave device to clear the interrupt. Then, the driver should issue a memory read to the same interrupt clear register. At the eSPI slave, the interrupt event virtual wire should be sent as cleared, between the posted memory write that cleared the interrupt, and returning the subsequent memory read completion.

The interrupt event virtual wire defines a mechanism of sending edge-triggered interrupt. An edge event interrupt is communicated in the same Virtual Wire packet by first indicating the new value of the interrupt line, and subsequently the next value that the interrupt line toggles. This may be interleaved by any other Virtual Wire groups within the same packet. The mechanism consumes two of the Virtual Wire count. The receiver must track the state of the interrupt event virtual wire while it is being received and translate it to an interrupt event pulse taking effect at the deassertion edge of the Chip Select#. The pulse width required is receiver specific such that the edge event interrupt can be recognized correctly by its internal logic. Edge-triggered interrupts must not be shared on an interrupt line.

As interrupt event virtual wire is communicated through an independent channel from the peripheral accesses, data may not be guaranteed to be in the main memory before interrupt is delivered to the CPU. This can lead to data consistency problem with the Producer-Consumer model. Software must perform a read to any register in the slave such that the slave's posted write buffers are flushed before accessing data written by the slave to the main memory.

Figure 40: Edge-triggered Interrupt through Virtual Wire



5.2.2.5 General-Purpose I/O Expander

The specification allows the eSPI master to claim the General-Purpose I/O (GPIO) pins physically resided on the eSPI slave side as part of its own virtual I/O pins.

If the Virtual GPIO is configured as an output pin, the eSPI master tunnels the state of the Virtual GPIO pin through in-band messaging and the eSPI slave, upon receiving the message, reflects the state on the GPIO pin physically located on the slave side.

If the Virtual GPIO is configured as an input pin, the eSPI slave samples the state of the physical GPIO pin and then tunnels the state of the GPIO pin through in-band messaging on any pin state transition.

All the GPIO pins sharing the same index number must be configured to the same direction. They can either be configured as all inputs or all outputs, but not a combination of inputs and outputs.

Similarly, a group of Virtual GPIOs sharing the same index will share the same reset. The reset is programmable to be reset by either eSPI Reset# or Platform Reset.

The GPIO software interface on the eSPI master and eSPI slave is implementation specific. The software is responsible to set up the configuration for the GPIO pins on both sides appropriately and in the consistent manner. The detail of the GPIO Configuration Registers is implementation specific.

The mapping of the Virtual GPIO pin to the physical GPIO pin on the eSPI slave side is implementation specific and outside the scope of the specification.

5.2.3 OOB (Tunneled SMBus) Message Channel

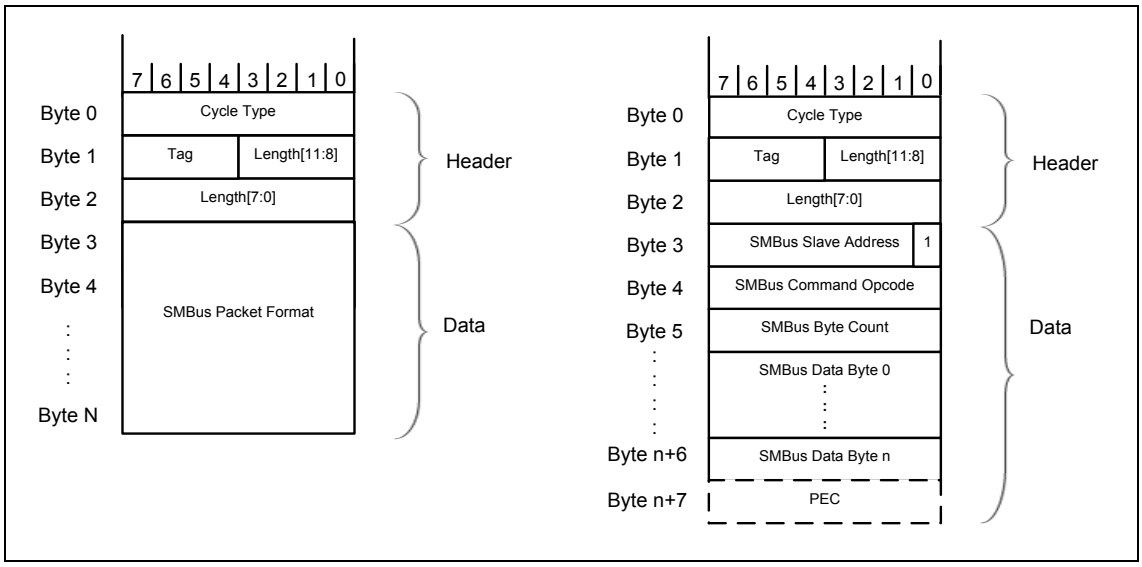
The SMBus packets can be tunneled through eSPI as Out-Of-Band(OOB) messages. The whole SMBus packet is embedded inside the eSPI OOB message as data.

Only SMBus block writes are tunneled through the eSPI OOB message.

The SMBus Slave Address, SMBus Command Opcode, SMBus Byte Count and SMBus Data fields are sent as data within the eSPI OOB message packet.

The presence of SMBus PEC is determined through a simple arithmetic operation between the eSPI header length field and the SMBus Byte Count.

Figure 41: OOB (Tunneled SMBus) Message Packet Format



5.2.4 Run-time Flash Access Channel

The Flash Access channel provides a path allowing the flash components to be shared run-time between chipset and the eSPI slaves that require flash accesses such as EC and BMC.

Once the Flash Controller in the chipset has completed the flash initialization, the Flash Access channel is enabled on the eSPI slave side.

The Flash Access channel uses the same packet format as the eSPI Peripheral Channel transactions.

The Tag field is used to match the completion with the request. Flash access requests with the same tag must be completed in order.

Figure 42: Flash Access Request Packet Format

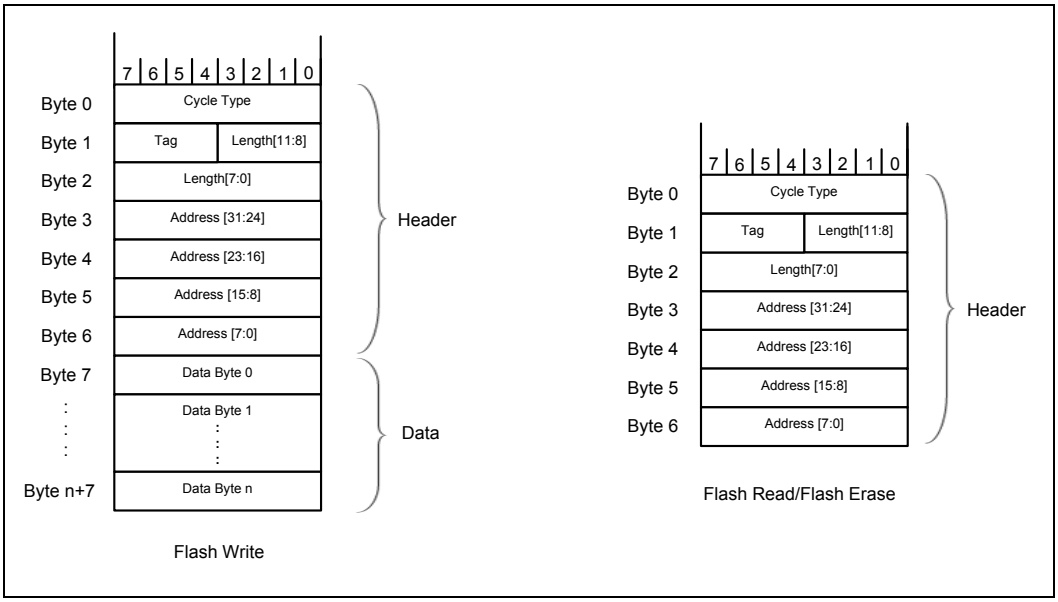
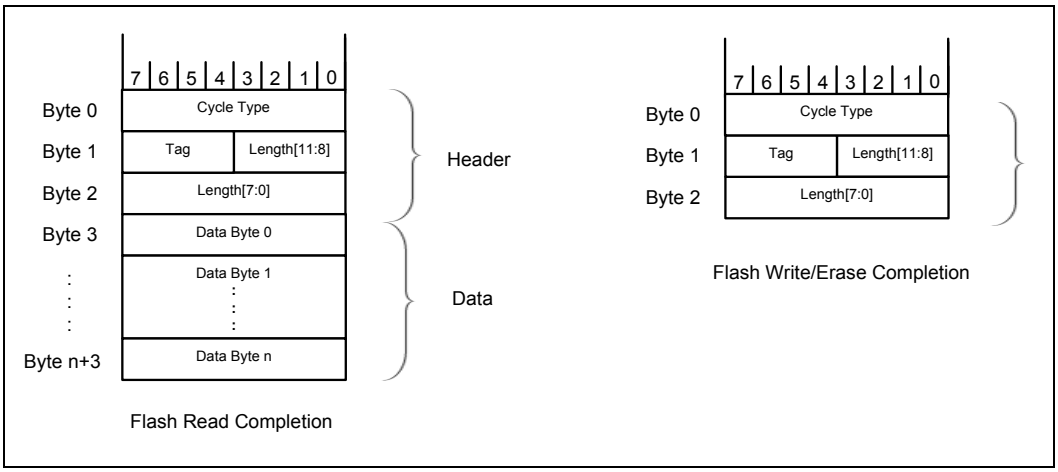


Figure 43: Flash Access Completion Packet Format



5.2.4.1 Master Attached Flash Sharing

Master Attached Flash Sharing refers to the scheme where flash components are attached to the eSPI master such as the chipset. eSPI slaves are allowed to access to the shared flash component through the Flash Access channel. The flash components may be on an independent SPI interface, or on a shared SPI/eSPI interface depending on the system configuration.

Figure 44: Independent Flash SPI and eSPI Interface

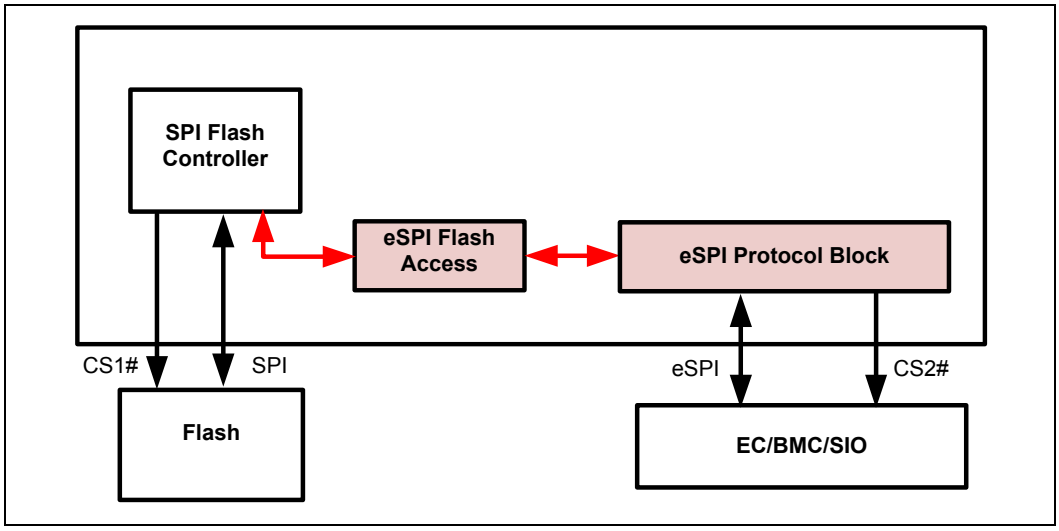
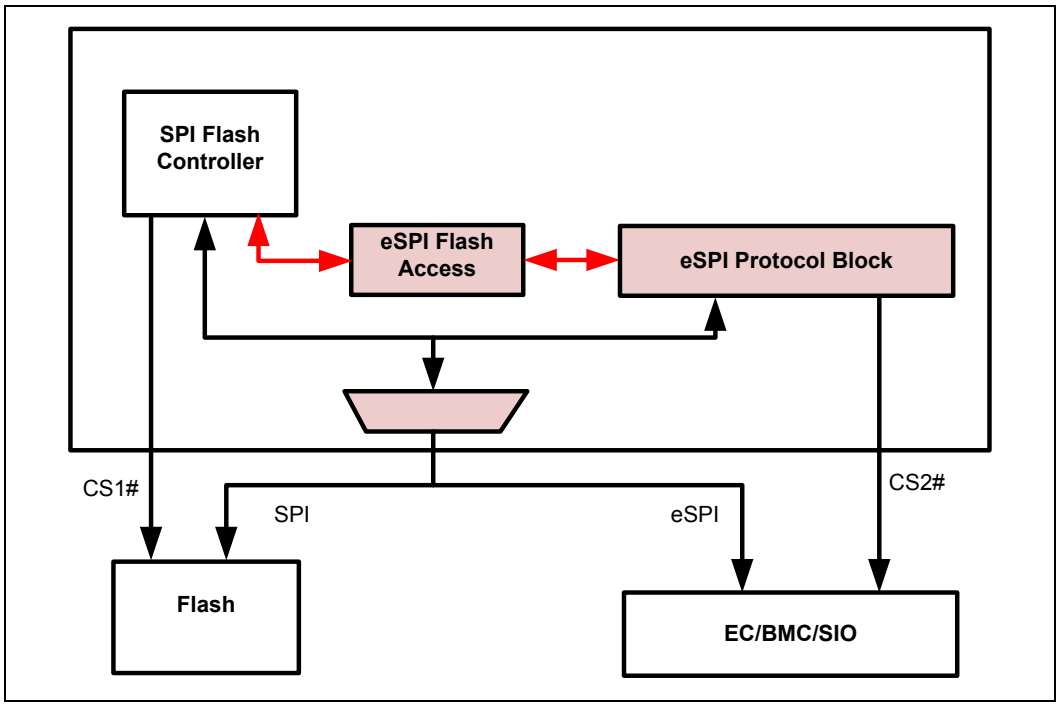


Figure 45: Shared SPI and eSPI Interface



eSPI slaves such as EC and BMC must be able to function appropriately with codes executed from other storage devices such as its own ROM before Flash Access channel is enabled for the run-time flash sharing.

The Master Attached Flash Sharing scheme uses 2 dedicated eSPI command opcodes: GET_FLASH_NP and PUT_FLASH_C (Table 2).

Any run-time access to the flash component through the eSPI interface will go through the eSPI master, which then routes the cycle to the Flash Access block, before the cycle is forwarded to the SPI Flash Controller. SPI Flash controller will perform the access to the flash on behalf of the eSPI slave.

SPI Flash controller as the flash device owner is responsible to handle the differences between the different flash vendors, making them transparent to the flash access channel on the eSPI bus.

The flash access addresses used by the eSPI slaves in the Flash Access transactions are logical addresses. The logical addresses are contiguous and start from 00000000h and rolls over based on the upper bound specified by ”Run-time Flash Access Region Size”. The translation from logical to physical flash address will be done within the Flash Access block. The translated physical address will then be used by the SPI Flash controller to access the flash.

Any attempt to access a flash region outside of the logical address space is considered an error. The SPI Flash controller is required check this and would synthesize an unsuccessful completion back to the eSPI slave.

The action taken by the eSPI slave in response to unsuccessful completion is implementation specific.

In the Master Attached Flash Sharing scheme, Flash Read, Flash Write and Flash Erase commands are supported over eSPI bus. These commands will be forwarded by the eSPI master to the flash controller where they will be mapped to the corresponding flash instructions by the flash controller.

Flash Read and Flash Write transactions are non-posted transactions. Each of the transactions will have a corresponding completion which indicates the status of the requested operation, together with data if the cycle is a Flash Read. The status of the completion will be conveyed back to the eSPI slave.

Flash Access Channel Maximum Read Request Size parameter in the Channel Capability and Configuration register is defined to allow the eSPI master to limit the Flash Read request size to the size supported by the eSPI master.

Similarly, Flash Access Channel Maximum Payload Size parameter in the Channel Capability and Configuration register is defined to allow the eSPI master to limit the Flash Write data payload size.

Flash Erase is a non-posted request with no data. This command instructs the SPI Flash controller to erase a part of the region allocated to the eSPI slave. The Address field specifies the beginning of the erase block and the least significant 2 bits of the length field specifies the size of the block to be erased. The encoding of the least significant 2 bits of the length field matches the value of the Flash Block Erase Size field of the Channel Capabilities and Configuration register. The specified address must

be aligned to the block erase size. The supported erase block size is programmable and is communicated by the eSPI master to the slave through the Channel Capabilities and Configuration register.

eSPI master will forward the transaction as it is to the flash controller. The flash controller will then perform the necessary check to ensure that the cycle is supported, prior to sending it out to the flash. If the cycle is not supported due to invalid addressing mode (32-bit versus 24/26-bit addressing), unsupported command, unsupported block erase size or any other reasons, the flash controller will synthesize an unsuccessful completion to the eSPI master which will then forward the completion to the eSPI slave over the eSPI bus, without sending the instruction to flash.

5.2.4.2 Slave Attached Flash Sharing

Slave Attached Flash Sharing scheme is defined only for server platforms and it is not included in this client version of the specification.

5.3 Slave Buffer Management

eSPI protocol defines a simplified buffer management mechanism. The eSPI slave communicates the availability of new transactions for transmission and availability of receive buffers to store the incoming transactions through the Status field of the Response phase.

The eSPI master does not need to communicate the queue information to the eSPI slave. eSPI master will wait until its relevant internal queue is free before servicing the requests from slave. If ordering rule permits, the eSPI master can choose to service another request which has queue resource available, while waiting for the relevant queue resource to free up.

The eSPI slave is responsible to ensure ordering prior to presenting the request to the eSPI master. A request should not be seen by the eSPI master through the Status field until it has met the ordering requirement with respect to other pending requests. For example, if a non-posted read at the top of the non-posted transmit queue is ordered behind a posted write, the non-posted read should not set the NP_AVAIL bit in the status register until all the posted writes in front of the non-posted read have been evicted.

The respective free indications can only be set if the eSPI slave receiver buffers can accept at least one maximum payload size packet. The free indication in the Status field returned as part of the Response phase must comprehend the buffer size consumed by the current transaction. For example, if the eSPI master initiates a posted write that exhausts the posted/completion queue of the eSPI slave receiver, the PC_FREE indication must be cleared in the Status field during the Response Phase of the corresponding command.

When the eSPI master issues a GET_* command, the Status field in the Response Phase must reflect the next state of the buffer associated with the GET_* command. For example, if the eSPI master issues a GET_PC and the PC_AVAIL Status bit is set during the Response Phase of this command, it indicates that there is yet another posted or completion transaction available after this command. If the PC_AVAIL Status bit is cleared, it indicates that there is no additional posted or completion transactions available after this command at the time of reporting.

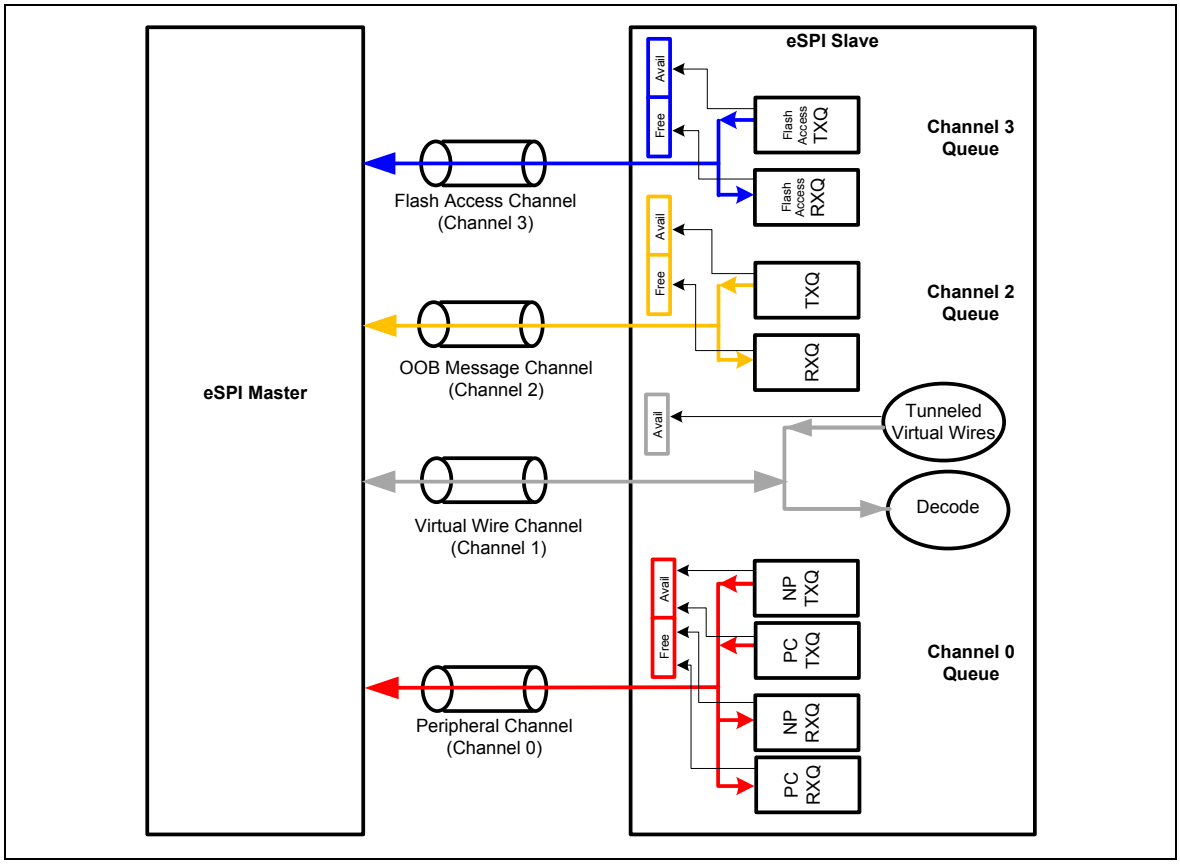


Figure 46: eSPI Slave Buffer Design (Conceptual)

5.4 Transaction Ordering Rule

The ordering rules specified here apply to the transactions within the same channel and share the same Chip Select# pin. There is no ordering requirement between transactions on different channels. There is no ordering requirement between transactions with the same channel number but involving eSPI slaves using different Chip Select# pin.

Row pass Column?	Posted Request	Non-Posted Request	Completion
Posted Request	No	Yes	No
Non-Posted Request	No	No	No
Completion	No	Yes	No

All the completions corresponding to the same channel with the same tag must be returned in request order. There is no requirement for completions from the same channel but different tag to be returned in request order. There is no requirement for completions from different channel to be returned in request order.

5.5 Zero Length Read and Write

Zero length memory, I/O and Flash reads and writes are not supported.

§ §

6 Link Layer

6.1 Single I/O, Dual I/O and Quad I/O Modes

All masters and slaves must support Single I/O mode of operation. Support for Dual I/O and Quad I/O mode of operation is advertised by the slave through the General Capabilities and Configurations register. Dual I/O and Quad I/O mode can be independently support by a particular Enhanced Serial Peripheral Interface (eSPI) slave.

By default coming out of eSPI Reset#, both master and slave operate in Single I/O mode. The mode of operation can be changed by the master using the SET_CONFIGURATION command.

The SET_CONFIGURATION is completed with the current mode of operation. The new mode of operation will only take effect at the deassertion edge of the Chip Select#.

In Dual I/O mode, the I/O[1:0] pins become bi-directional to form the bi-directional data bus and all the command and response phases are transferred over the two bi-directional pins at the same time, effectively doubling the transfer rate.

Each of the fields are shifted out from least significant byte to most significant byte where each bytes is shifted from the most significant bit to the least significant bit.

Figure 47: Command Packet Transmission Timing Waveform (Single I/O)

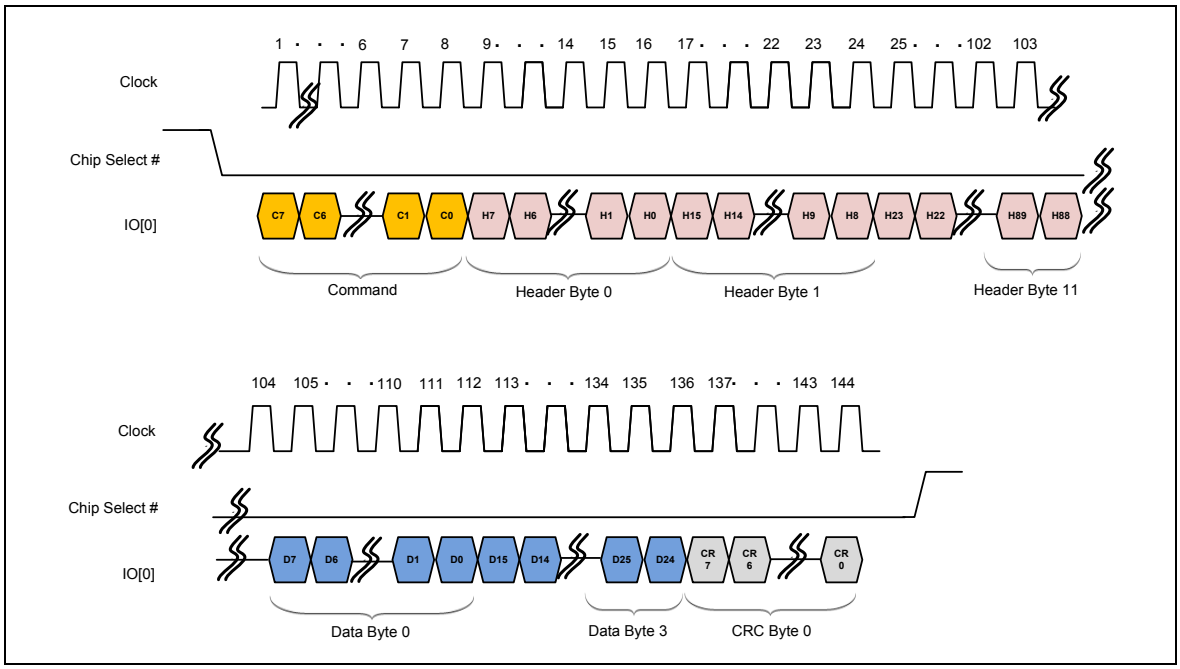


Figure 48: Command Packet Transmission Timing Waveform (Dual I/O)

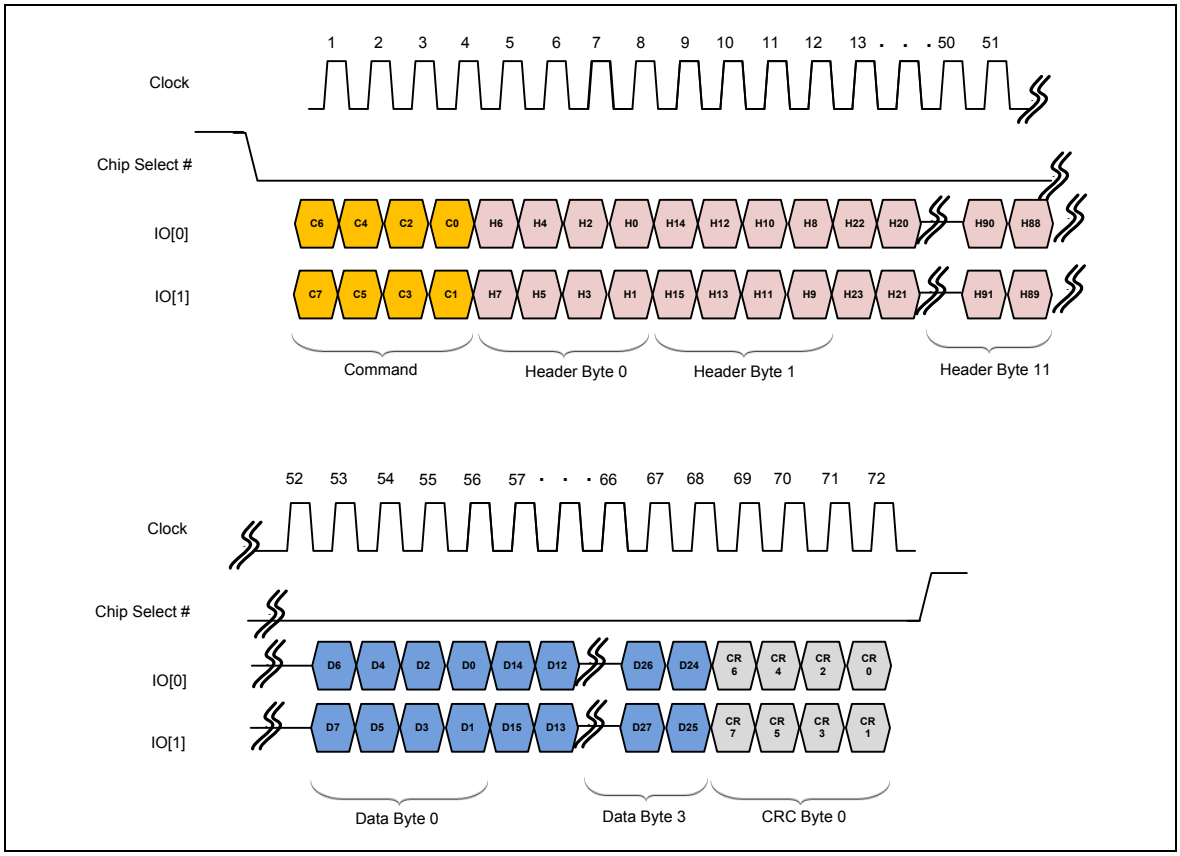
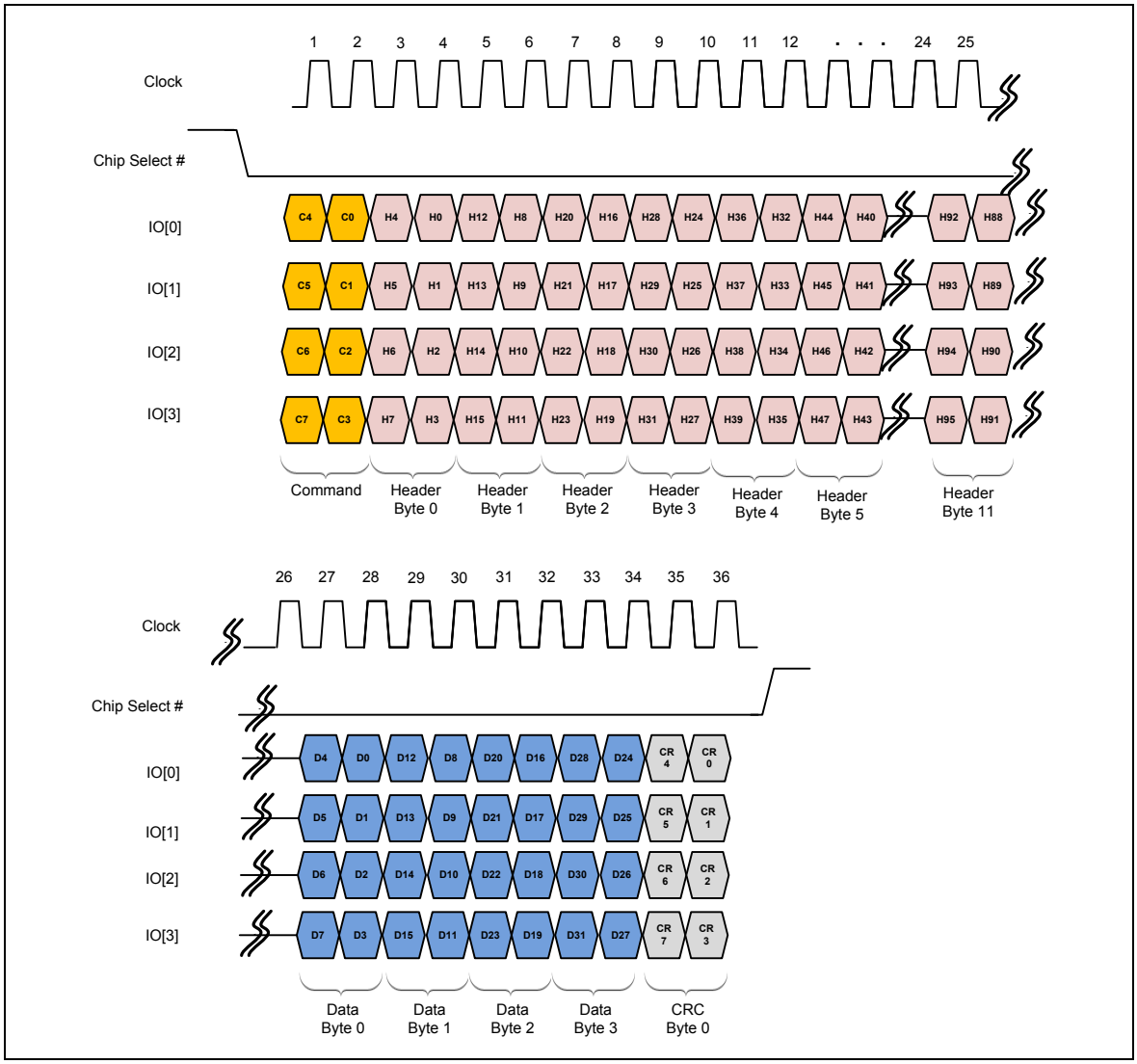


Figure 49: Command Packet Transmission Timing Waveform (Quad I/O)



6.2 Cyclic Redundancy Check (CRC)

CRC-8 is used to protect the eSPI transaction packets. The Command Phase and Response Phase contain respective CRC byte. For Command Phase, the CRC calculation includes all the bytes during the Command Phase such as the Command Opcode, Header (if present) and Data (if present). For Response Phase, the CRC calculation includes all the bytes during the Response Phase such as the Response code, Header (if present), Data (if present) and Status.

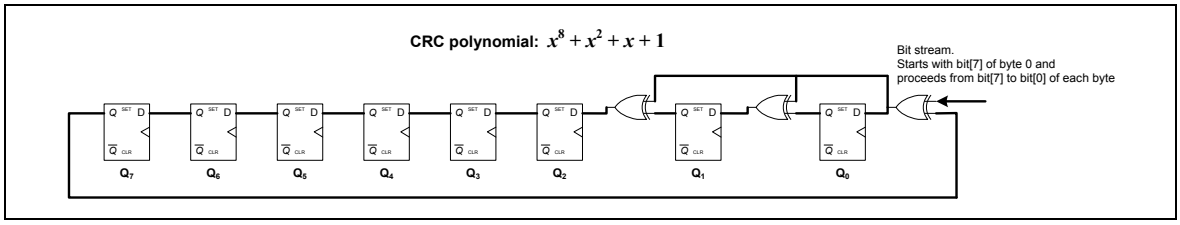
The CRC value is calculated using the following rules:

- The polynomial is expressed as: $x^8 + x^2 + x + 1$.
- The polynomial used for the CRC calculation has a coefficient expressed as "07h".
- The seed value is "00h". The CRC storage registers are reset to the initial value of 00h prior to any CRC byte calculation.
- The CRC calculation starts with bit[7] of byte 0 and proceeds from bit[7] to bit[0] of each byte.

CRC support is optional, with the default as disabled after the eSPI reset#. CRC support can be discovered through the GET_CONFIGURATION register and can be enabled through the SET_CONFIGURATION register.

When CRC support is disabled, the CRC bytes are not transmitted on the bus during the eSPI transactions. The CRC checking is disabled in this case.

Figure 50: CRC Polynomial Representation



§ §

7 Slave Registers

The Enhanced Serial Peripheral Interface (eSPI) defines a set of slave registers. These registers are required for enumeration, configuration and for proper operation of the respective independent channels defined for the eSPI bus.

The following tables describe the register attribute and register default value encodings used in this specification.

Table 12: Register Attribute Description

Register Attribute	Description
RO	**Read-Only.** Register bits are read-only and cannot be altered by software.

Table 13: Register Default Values Encoding Description

Register Default Value	Description
Platform Specific	**Platform Specific.** The default value of the register is platform specific.
Table	**Table.** The default value advertised in this field is described by a table. See the description of the register to associate the register with the corresponding table.
HwInit	**Hardware-Init.** Register with the default value marked as "HwInit" indicates that the default value is determined by the hardware capability and the default value should reflect the supported hardware capability.

7.1 Status Register

The Status register bits are reset by eSPI Reset#.

The content of the Status register is returned in the corresponding Status field of the Response Phase.

Refer to Section 4.4.2 for the description of the Status register field.

7.2 Capabilities and Configuration Registers

The capabilities and configuration register bits are reset by eSPI Reset#.

Register fields that are marked as Reserved must always return zero when read. Writing to the Reserved fields has no effect.

The GET_CONFIGURATION and SET_CONFIGURATION commands are used to access these registers. The registers are only accessible on DWord granularity. When configuring the registers using the SET_CONFIGURATION command, the new register

value to the slave will only take effect at the deassertion edge of the Chip Select#. Thus, the SET_CONFIGURATION command is ran on the eSPI bus based on the current pre-configured settings.

Registers from offset 800h to FFFh are reserved as platform specific. This provides a 2KB register space for platform specific application.

Table 14: Slave Registers

Start (Hex)	End (Hex)	Register Name
000	003	Reserved
004	007	Device Identification
008	00B	General Capabilities and Configurations
00C	00F	Reserved
010	013	Channel 0 Capabilities and Configurations
014	01F	Reserved
020	023	Channel 1 Capabilities and Configurations
024	02F	Reserved
030	033	Channel 2 Capabilities and Configurations
034	03F	Reserved
040	043	Channel 3 Capabilities and Configurations
044	7FF	Reserved
800	FFF	Platform Specific registers

7.2.1.1 Offset 00h: Reserved

7.2.1.2 Offset 04h: Device Identification

Bit	Type	Default	Description
31:8	RO	0	**Reserved.**
7:0	RO	01h	**Version ID:** Indicates compliance to specific eSPI specification revision. Slaves compliant to this revision of the specification must advertise a value of "01h" in this field.

7.2.1.3 Offset 08h: General Capabilities and Configurations

Bit	Type	Default	Description
31	RW	0b	**CRC Enable:** This bit is set to '1' by eSPI master to enable the CRC on the eSPI bus. By default, CRC is disabled and CRC bytes are not transmitted on the bus. 0b: CRC is disabled. 1b: CRC is enabled.
30	RO	HwInit	**CRC Support:** This bit specifies whether CRC is supported by the slave. 0b: CRC is not supported. 1b: CRC is supported.
29	RO	0	**Reserved.**
28	RW	0b	**Alert Mode:** This bit serves to configure the Alert mechanism used by the slave to initiate a transaction on the eSPI interface. 0b: I/O[1] pin is used to signal the Alert event. 1b: A dedicated Alert# pin is used to signal the Alert event. Note: This bit can only be '0' in a single master-single slave topology. For single master-multiple slave topology, this bit must be programmed to '1'.

<table>
<tr>
<td>27:26</td>
<td>RW</td>
<td>00b</td>
<td>I/O Mode Select: eSPI master programs this field to enable the appropriate mode of operation, which will take effect at the deassertion edge of the Chip Select#.

The I/O Mode configured in this field must be supported by both the master and the slave. Single I/O mode is supported by default.

<table>
<tr><th>Encoding</th><th>Operating Mode</th></tr>
<tr><td>00</td><td>Single I/O</td></tr>
<tr><td>01</td><td>Dual I/O</td></tr>
<tr><td>10</td><td>Quad I/O</td></tr>
<tr><td>11</td><td>Reserved.</td></tr>
</table>
</td>
</tr>
<tr>
<td>25:24</td>
<td>RO</td>
<td>HwInit</td>
<td>I/O Mode Support: This field indicates the I/O modes supported by the slave.

<table>
<tr><th>Encoding</th><th>Supported I/O Mode</th></tr>
<tr><td>00</td><td>Single I/O</td></tr>
<tr><td>01</td><td>Single and Dual I/O</td></tr>
<tr><td>10</td><td>Single and Quad I/O</td></tr>
<tr><td>11</td><td>Single, Dual and Quad I/O</td></tr>
</table>
</td>
</tr>
<tr>
<td>23</td>
<td>RO</td>
<td>0</td>
<td>Reserved.</td>
</tr>
<tr>
<td>22:20</td>
<td>RW</td>
<td>000b</td>
<td>Operating Frequency: This field identifies the frequency of operation.

<u>Bits</u> <u>Frequency</u>
000 20 MHz
001 25 MHz
010 33 MHz
011 50 MHz
100 66 MHz
Others Reserved.

Note: This field has a default value of “000” to reflect $t_{INIT-FREQ}$ (Table 16) of 20MHz max.</td>
</tr>
<tr>
<td>19</td>
<td>RO</td>
<td>0</td>
<td>Reserved.</td>
</tr>
</table>

<table>
<tr>
<td>18:16</td>
<td>RO</td>
<td>HwInit</td>
<td>Maximum Frequency Supported: This field identifies the maximum frequency of operation supported by the slave.

<u>Bits</u> <u>Frequency</u>
000b 20 MHz
001b 25 MHz
010b 33 MHz
011b 50 MHz
100b 66 MHz
Others Reserved.

The slave that indicates support for the maximum frequency of operation through this field will also support all the lower frequencies on the list.

Note: Support for $t_{INIT-FREQ}$ (Table 16) is mandatory.</td>
</tr>
<tr>
<td>15:12</td>
<td>RW</td>
<td>8h</td>
<td>Operating Turn-Around (TAR) Time Value: The number of eSPI clocks that the master is required to wait for the slave’s response after the last clock of the command phase.

This is a 1-based field. When “0”, it indicates a value of 16.

The Turn-Around value used is the number programmed in this field multiplied by 2 eSPI clocks.
The value configured in this field must never be less than the value advertised in the Minimum Turn-Around Time Supported field.

This field is set by the eSPI master to define the TAR values to be used in

The default value of this field is 8h to reflect the $t_{INIT-TAR}$ (Table 16) of 16 eSPI clocks.</td>
</tr>
<tr>
<td>11:8</td>
<td>RO</td>
<td>HwInit</td>
<td>Minimum Turn-Around (TAR) Time Supported: This field specifies the minimum TAR time required by the eSPI slave to respond to a command.

The eSPI master would read this field and communicate the actual TAR time used to the slave through the Operating Turn-Around Time Value field.

This is a 1-based field. When “0”, it indicates a value of 16.

The Turn-Around Value supported is the value specified in this field multiplied by 2 eSPI clocks.</td>
</tr>
<tr>
<td>7:0</td>
<td>RO</td>
<td>HwInit</td>
<td>Channel Supported: Each of the bits when set indicates that the corresponding channel is supported by the slave.

<table>
<tr><th>Bits</th><th>Channel</th></tr>
<tr><td>0</td><td>Peripheral Channel</td></tr>
<tr><td>1</td><td>Virtual Wire Channel</td></tr>
<tr><td>2</td><td>OOB Message Channel</td></tr>
<tr><td>3</td><td>Flash Access Channel</td></tr>
<tr><td>4:7</td><td>Reserved for platform specific channels</td></tr>
</table>
</td>
</tr>
</table>

7.2.1.4 Offset 10h: Channel 0 Capabilities and Configurations

Bit	Type	Default	Description
31:15	RO	0	**Reserved.**
14:12	RW	001b	**Peripheral Channel Maximum Read Request Size:** eSPI master sets the maximum read request size for the Peripheral channel. The maximum read request size must not cross the naturally aligned address boundary for the corresponding size. 000b: Reserved. 001b: 64 bytes address aligned max read request size. 010b: 128 bytes address aligned max read request size. 011b: 256 bytes address aligned max read request size. 100b: 512 bytes address aligned max read request size. 101b: 1024 bytes address aligned max read request size. 110b: 2048 bytes address aligned max read request size. 111b: 4096 bytes address aligned max read request size.
11	RO	0	**Reserved.**
10:8	RW	001b	**Peripheral Channel Maximum Payload Size Selected:** eSPI master sets the maximum payload size for the Peripheral channel. The value set by the eSPI master must never be more than the value advertised in the Max Payload Size Supported field. 000b: Reserved. 001b: 64 bytes max payload size. 010b: 128 bytes max payload size. 011b: 256 bytes max payload size. 100b – 111b: Reserved.
7	RO	0	**Reserved.**
6:4	RO	HwInit	**Peripheral Channel Maximum Payload Size Supported:** This field advertises the Maximum Payload Size supported by the slave. 000b: Reserved. 001b: 64 bytes max payload size. 010b: 128 bytes max payload size. 011b: 256 bytes max payload size. 100b – 111b: Reserved.

3:2	RO	0	**Reserved.**
1	RO	0b	**Peripheral Channel Ready:** When this bit is a '1', it indicates that the slave is ready to accept transactions on the Peripheral channel. eSPI master should poll this bit after the channel is enabled before running any transaction on this channel to the slave. 0b: Channel is not ready. 1b: Channel is ready.
0	RW	0b	**Peripheral Channel Enable:** This bit is set to '1' by eSPI master to enable the Peripheral channel. The channel is by default disabled after the eSPI Reset#.

7.2.1.5 Offset 20h: Channel 1 Capabilities and Configurations

Bit	Type	Default	Description
31:22	RO	0	**Reserved.**
21:16	RW	0	**Operating Maximum Virtual Wire Count:** The maximum number of Virtual Wire groups that can be sent in a single Virtual Wire packet. This is a 0-based count. The default value of 0 indicates count of 1. The value configured in this field must never be more than the value advertised in the Maximum Virtual Wire Count Supported field.
15:14	RO	0	**Reserved.**
13:8	RO	HwInit	**Maximum Virtual Wire Count Supported:** This field advertises the Maximum Virtual Wire Count supported by the slave. If the slave supports different count value as initiator and as receiver of the Virtual Wires, this field indicates the lower of the two. The Virtual Wire Count specifies the number of Virtual Wire groups communicated in a single Virtual Wire packet. This is a 0-based count.
7:2	RO	0	**Reserved.**
1	RO	0b	**Virtual Wire Channel Ready:** When this bit is a '1', it indicates that the slave is ready to accept transactions on the Virtual Wire channel. eSPI master should poll this bit after the channel is enabled before running any transaction on this channel to the slave. 0b: Channel is not ready. 1b: Channel is ready.
0	RW	0b	**Virtual Wire Channel Enable:** This bit is set to '1' by eSPI master to enable the Virtual Wire channel. The channel is by default disabled after the eSPI Reset#.

7.2.1.6 Offset 30h: Channel 2 Capabilities and Configurations

Bit	Type	Default	Description
31:11	RO	0	**Reserved.**
10:8	RW	001b	**OOB Message Channel Maximum Payload Size Selected:** eSPI master sets the maximum payload size for the OOB Message channel. The value set by the eSPI master must never be more than the value advertised in the Max Payload Size Supported field. 000b: Reserved. 001b: 64 bytes max payload size. 010b: 128 bytes max payload size. 011b: 256 bytes max payload size. 100b – 111b: Reserved.
7	RO	0b	**Reserved.**
6:4	RO	HwInit	**OOB Message Channel Maximum Payload Size Supported:** This field advertises the Maximum Payload Size supported by the slave. 000b: Reserved. 001b: 64 bytes max payload size. 010b: 128 bytes max payload size. 011b: 256 bytes max payload size. 100b – 111b: Reserved.
3:2	RO	0	**Reserved.**
1	RO	0b	**OOB Message Channel Ready:** When this bit is a '1', it indicates that the slave is ready to accept transactions on the OOB Message channel. eSPI master should poll this bit after the channel is enabled before running any transaction on this channel to the slave. 0b: Channel is not ready. 1b: Channel is ready.
0	RW	0b	**OOB Message Channel Enable:** This bit is set to '1' by eSPI master to enable the OOB Message channel. The channel is by default disabled after the eSPI Reset#.

7.2.1.7 Offset 40h: Channel 3 Capabilities and Configurations

Bit	Type		Description
31:16	RW	0000h	**Run-time Flash Access Region Size:** eSPI master specifies the Flash region size allocated to the slave in 4KB granularity by writing to this field. This field is applicable only to master attached flash sharing scheme. "0000h": Flash Access region is not allocated to the slave. "0001h": Allocate 4KB of Flash Access region to the slave. "0002h": Allocate 8KB of Flash Access region to the slave. : : "FFFFh": Allocate 255 MB of Flash Access region to the slave.
15	RO	0b	**Reserved.**
14:12	RW	001b	**Flash Access Channel Maximum Read Request Size:** eSPI master sets the maximum read request size for the Flash Access channel. The maximum read request size must not cross the naturally aligned address boundary for the corresponding size. 000b: Reserved. 001b: 64 bytes address aligned max read request size. 010b: 128 bytes address aligned max read request size. 011b: 256 bytes address aligned max read request size. 100b: 512 bytes address aligned max read request size. 101b: 1024 bytes address aligned max read request size. 110b: 2048 bytes address aligned max read request size. 111b: 4096 bytes address aligned max read request size.
11	RO	0b	**Reserved.**
10:8	RW	001b	**Flash Access Channel Maximum Payload Size Selected:** eSPI master sets the maximum payload size for the Flash Access channel. The value set by the eSPI master must never be more than the value advertised in the Max Payload Size Supported field. 000b: Reserved. 001b: 64 bytes max payload size. 010b: 128 bytes max payload size. 011b: 256 bytes max payload size. 100b – 111b: Reserved.

7	RO	0b	**Flash Sharing Mode:** When Flash Access channel is supported, this bit advertises the flash sharing scheme intended by the slave. 0b: Master attached flash sharing. 1b: Reserved.
6:4	RO	HwInit	**Flash Access Channel Maximum Payload Size Supported:** This field advertises the Maximum Payload Size supported by the slave. 000b: Reserved. 001b: 64 bytes max payload size. 010b: 128 bytes max payload size. 011b: 256 bytes max payload size. 100b – 111b: Reserved.
3:2	RW	01b	**Flash Block Erase Size:** eSPI master sets this field to communicate the block erase size to the slave. This field is applicable only to master attached flash sharing scheme. 00b: Reserved 01b: 4 KBytes 10b: Reserved 11b: 64 KBytes
1	RO	0b	**Flash Access Channel Ready:** When this bit is a '1', it indicates that the slave is ready to accept transactions on the Flash Access channel. eSPI master should poll this bit after the channel is enabled before running any transaction on this channel to the slave. 0b: Channel is not ready. 1b: Channel is ready.
0	RW	0b	**Flash Access Channel Enable:** This bit is set to '1' by eSPI master to enable the Flash Access channel. The channel is by default disabled after the eSPI Reset#.

§ §

8 Operating Specification

8.1 Electrical Specification

Note: The electrical specification defined in this section is preliminary and it is subjected to change.

Table 15: Electrical Specification

Symbol	Parameter	Condition	Min	Typ	Max	Unit
Vcc	eSPI I/O voltage		1.62	1.8	1.98	V
R_{ON}	Output driver impedance	V_{out} = Vcc/2	15	25	35	Ohm
V_{IL}	Input low voltage				0.3*Vcc	V
V_{IH}	Input high voltage		0.7*Vcc			V
V_{HYS}	Input hysteresis voltage		0.1*Vcc			V
$R^{1}_{reset-PU}$	Weak pull-up impedance	V_{out} = 0.7*Vcc	10k		30k	Ohm
C_{in}	Input capacitance				5	pF
C_L[2]	Load capacitance			10		pF
I_{IL}	Input leakage current	0 < V_{in} < Vcc			±10	uA

Note:

1. Weak pull-up on eSPI data, clock, Chip Select# and Alert# pins must be implemented as an integral part of the eSPI master buffer or on the board.

2. C_L is the test load defined for AC timing measurement.

8.2 Timing Parameters

All timing parameters for the Enhanced Serial Peripheral Interface (eSPI) are specified from a device (slave) perspective. The host is required to account for channel effects in meeting the specified timings with the device.

Note: The timing parameters defined in this section are preliminary and they are subjected to change.

Table 16: AC Timing Specification

Symbol	Parameter Description
t_{CKH}	Clock High Time
t_{CKL}	Clock Low Time
t_{SLCH}	Chip Select# Setup Time
t_{CHSH}	Chip Select# Hold Time
t_{SHSL}	Chip Select# Deassertion Time
t_{DVCH}	Data In Setup Time
t_{CHDX}	Data In Hold Time
t_{CLQV}	Output Data Valid Time
t_{CLQX}	Output Data Hold Time
t_{SHQZ}	Output Disable Time
t_{SLAZ}	Chip Select# Assertion to I/O[1] or ALERT# Tri-stated
t_{INIT}	eSPI Reset# Deassertion to First Transaction (GET_CONFIGURATION)
$t_{INIT-FREQ}$	Initial Bus Frequency upon eSPI Reset# Deassertion
$t_{INIT-TAR}$	Initial Turn-Around Time upon eSPI Reset# Deassertion

Symbol	20MHz		25MHz		33MHz		50MHz		66MHz		
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Unit
t_{CK}	50		40		30		20		15		ns
t_{CKH}	0.4		0.4		0.4		0.4		0.4		t_{CK}
t_{CKL}	0.4		0.4		0.4		0.4		0.4		t_{CK}
t_{SLCH}	25		20		15		10		7		ns
t_{CHSH}	50		40		30		20		15		ns
t_{SHSL}	50		40		30		20		15		ns

Symbol	20MHz		25MHz		33MHz		50MHz		66MHz		
t_{DVCH}	12		10		7		5		3		ns
t_{CHDX}	12		10		7		5		3		ns
t_{CLQV}		20		15		10		6		4	ns
t_{CLQX}		0		0		0		0		0	ns
t_{SHQZ}		15		12		9		6		4	ns
t_{SLAZ}		15		12		9		6		4	ns
t_{INIT}	1		1		1		1		1		us
$t_{INIT-FREQ}$		20		20		20		20		20	MHz
$t_{INIT-TAR}$	16		16		16		16		16		t_{CK}

Figure 51: Input Timing Diagram

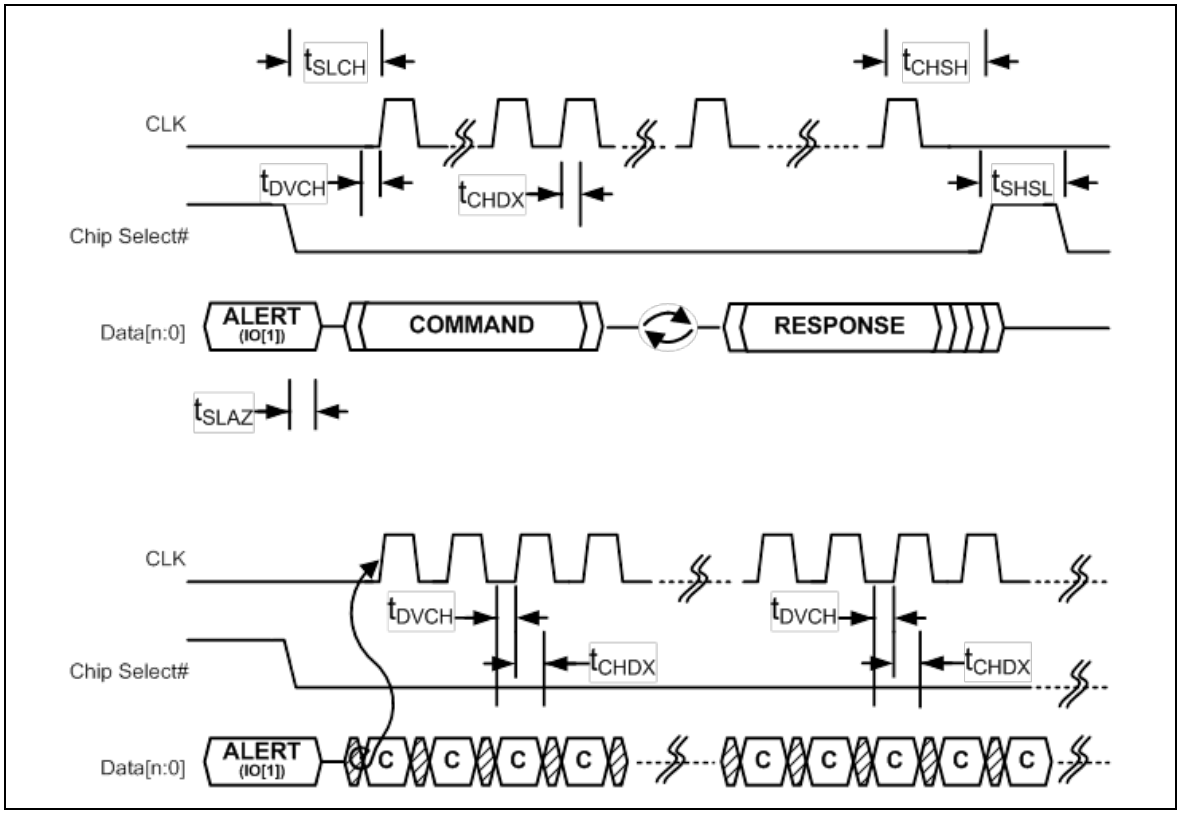
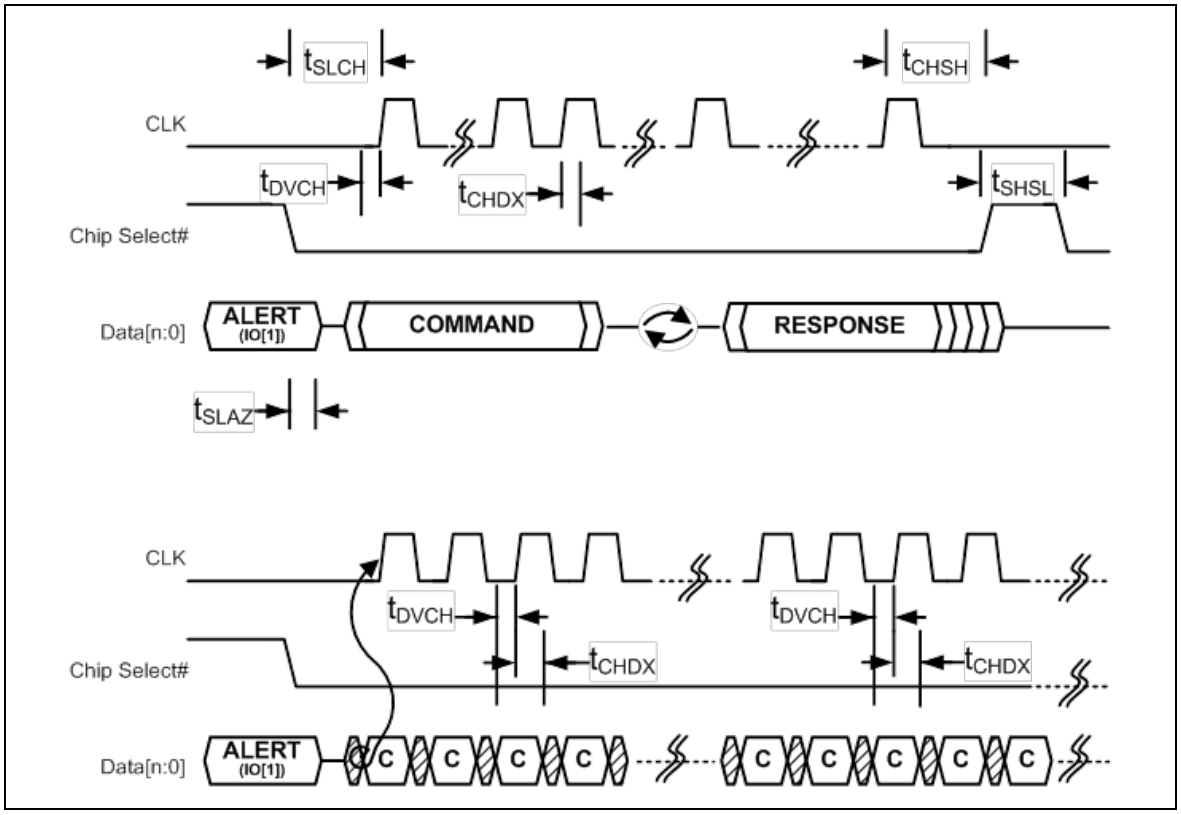


Figure 52: Output Timing Diagram



§ §

9 System Architecture

9.1 Interrupts

The Enhanced Serial Peripheral Interface (eSPI) provides a mechanism for eSPI endpoints that are transparent to software to communicate their interrupts through the Interrupt Event Virtual Wires.

eSPI endpoints from different channels share the same set of interrupt lines routed over the dedicated Virtual Wire channel.

Interrupts sent as the Interrupt Event Virtual Wires will be mapped to the respective IRQ lines. The detail of interrupt mapping is platform specific and outside the scope of the specification.

The ACPI method is used to communicate the IRQ number used by the eSPI endpoints.

The specification does not preclude the endpoints that are transparent to PCI software from using Message Signaled Interrupt (MSI). However, the method to enable MSI support is beyond the scope of the specification.

9.2 Error Detection and Handling

eSPI bus supports error detection capability through CRC protection when CRC support is enabled. The errors detected can be logged and reported through the respective eSPI master configuration space, which is outside the scope of this specification.

There is no error correction capability or hardware recovery mechanism defined for the eSPI bus.

The categories of errors that can be detected over the eSPI bus include:

- Errors detected by the eSPI slave on the command packet from eSPI master that passes the CRC check. This could be due to errors at the higher layers such as invalid command, invalid cycle type, invalid address range, unexpected completion, unsuccessful completion, malformed transaction, etc.
- Errors detected by the eSPI master on the response packet from eSPI slave that passes the CRC check. This could be caused by errors at the higher layers such as unsuccessful completion, invalid response, invalid cycle type, invalid address range, unexpected completion, malformed transaction, etc.
- CRC error detected on packets from eSPI master to eSPI slave and vice versa, caused by lower layer transmission and reception errors.

Due to lack of hardware recovery mechanism, all the errors detected on the eSPI bus fall into one of the Fatal or non-Fatal category.

Segregating the errors into Fatal and non-Fatal categories is optional. It provides a path for the software to handle the non-Fatal error in a more robust manner instead of treating the non-Fatal error as System Error.

eSPI master that does not support the segregation of errors into Fatal and non-Fatal categories may choose to handle these errors in the same manner.

eSPI slaves that do not support the segregation of errors into Fatal and non-Fatal errors may choose to report all errors as Fatal Error response.

Note: Implementation Note: If error segregation into Fatal and non-Fatal errors is supported, the eSPI master can choose to generate a System Error in response to Fatal Error and generate an interrupt or SMI# in response to Non-Fatal Error. Handling the error through interrupt or SMI# requires the corresponding device driver or BIOS support.

9.2.1 Slave Fatal Error Response

eSPI slaves may communicate to the eSPI master that the current transaction has a serious error by returning a Fatal Error response in the Response Phase.

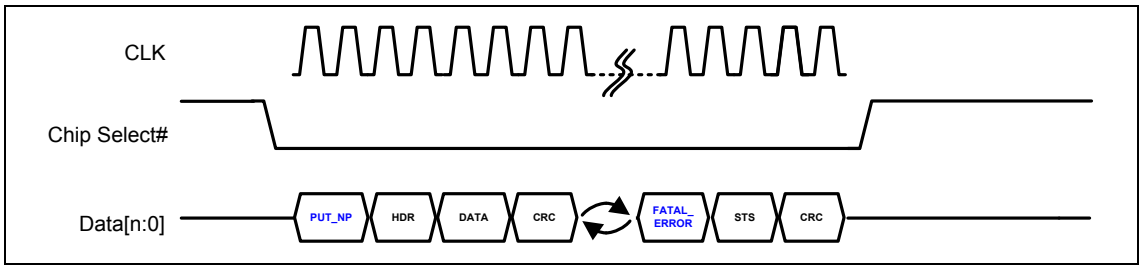
This could be due to the corresponding command could not be processed or that a severe error has been detected by the eSPI slaves that resulted in its inability to make forward progress. Examples of errors that fall into this category include receiver buffer overflow, unsupported command and unsupported cycle type.

Based on the response, the eSPI master may choose to generate a System Error (SERR) if it is a PCI device or route the error as an interrupt or SMI#. Alternatively, the eSPI master may choose to take other necessary actions or no action. The decision taken by the eSPI master in response to Fatal Error is implementation specific and beyond the scope of the specification.

The Response with Fatal Error comprises the Response, Status and an optional CRC. There is neither Header nor Data field during the Response phase.

A deferred completion may be completed with a Fatal Error response at a later point in time.

Figure 53: Non-Posted Transaction with FATAL Error Response



9.2.2 Slave Non-Fatal Error Response

The eSPI slaves can return a Non-Fatal error in response to a command which is erroneous but does not impede the processing of the command and the forward progress of the bus. Examples for this type of errors include unsuccessful completions where software can attempt to recover from the error.

The intent of this mechanism is to provide a method for slaves to communicate Non-Fatal errors to higher layer protocol stacks for more robust error recovery.

The behavior of the eSPI master in response to receiving a Non-Fatal Error is implementation specific and beyond the scope of the specification.

eSPI slaves may choose not to support Non-Fatal error reporting. In this case, the Non-Fatal Error response will not be used by these slaves.

9.2.3 Slave No Response

In the case of CRC error, invalid command or invalid cycle type, the boundary of the command packet is indeterminate.

The eSPI slave must not drive the Response Phase when the boundary of the command packet cannot be determined.

After the command phase and the Turn-Around time, upon receiving the Response Code of all 1's (NO_RESPONSE), the eSPI master can deduce that there is either no slave present, or the slave has encountered a fatal error. The slave does not drive the response phase in this case and the Response Code of all 1's is a result of the weak pull-up on the I/O[n:0] pins.

9.2.4 Master Error Handling

For eSPI slave initiated posted writes which are unsuccessful, the error is not communicated back to the initiating slave. The error handling, logging and reporting in the eSPI master is implementation specific.

For eSPI slave initiated non-posted transactions which are unsuccessful, an unsuccessful completion will be returned to the eSPI slave. The corresponding error handling, logging and reporting in the eSPI master is implementation specific.

When a CRC error is detected on the Response Phase received from eSPI slave, the Response packet boundary is indeterminate.

The eSPI master is allowed to stop the clock and de-assert the Chip Select# at any point to terminate the transaction. The eSPI slave is required to be ready to accept new command on the next Chip Select# assertion.

9.3 Reset

eSPI Reset# is an out-of-band pin used to communicate the interface reset event between eSPI master and eSPI slave. Unless otherwise specified, the entire eSPI interface related hardware logic and circuit will be reset by eSPI Reset#. Although the eSPI interface is reset by the eSPI Reset#, the eSPI controller may or may not be reset. It is hardware implementation specific and outside the scope of the specification.

Platform Reset event is communicated through the PLTRST virtual wire. Platform Reset can be used to reset the GPIOs which are used to control the other board components that share the same reset.

In typical implementation, the eSPI Reset# is the same as Platform Reset. For Embedded Controller or Baseboard Management Controller, the eSPI Reset# is connected to the Reset signal of deeper power well compared to Platform Reset.

9.4 Power Management Event (PME)

Power Management Event (PME) is used by eSPI slaves to request for wake up from low power states.

This event is communicated as in-band message through the Virtual Wire channel. The assertion of this event is not qualified with PCI Power Management Configuration registers.

9.5 Power Sequencing & Initialization

This section describes the entry and exit flows for various platform power management states. The actual flow may differ slightly from one implementation to another. Implementers should refer to the respective component specification for the exact flows.

9.5.1 Exit from G3

The following sequence of steps will be performed by the Enhanced Serial Peripheral Interface (eSPI) controller upon deassertion of eSPI Reset# on exit from G3:

1. By default, eSPI master and slaves operate in low speed mode with a clock frequency of $t_{INIT-FREQ}$.
2. By default, eSPI master and slaves operate in single input (MISO) and single output mode (MOSI).
3. eSPI master will wait for t_{INIT} from eSPI Reset# de-assertion before starting the first operation on the bus.

4. eSPI master initiates GET_CONFIGURATION instruction to discover specific capabilities of the eSPI slaves.
    a. The mechanism by which the eSPI master knows which Chip Select# and Alert# pin correspond to specific eSPI slave is implementation specific.
5. eSPI master evaluates the discovered capabilities and performs a SET_CONFIGURATION command to the eSPI slaves to configure the capabilities based on supported configurations.
    a. To reduce the initialization time, eSPI master could configure the eSPI slaves to run at a higher supported bandwidth.
6. The Virtual Wire channel is then enabled, if supported
7. Once the Flash controller is ready, the Flash Access Channel is enabled if supported. The shared flash can then be accessed by both eSPI master and the corresponding eSPI slave.
8. Depending on the configuration, chipset may wait for the BOOT_DONE Virtual Wire message from eSPI slave before continuing the exit sequence.
9. Chipset sends in-band Virtual Wire messages to communicate the SLP_S5, SLP_S4 and SLP_S3 de-assertion as part of the power up sequence.
10. Chipset sends SUS_STAT Virtual Wire message to eSPI slave to communicate SUS_STAT de-assertion.
11. Once the core well is up and out of reset, the corresponding PLTRST deassertion message is sent from Chipset to eSPI slave.
12. The eSPI Peripheral Channel is then enabled if supported and cycles can then be initiated by both the master and slaves through this channel.

§ §