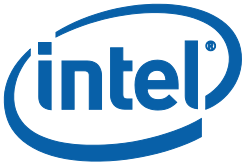


Enhanced Serial Peripheral Interface (eSPI)

Interface Base Specification (for Client and Server Platforms)

January 2016

Revision 1.0

327432-004

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
327432-002	0.7	• Updated with 0.6 spec review feedback.	October 2012
327432-003	0.75	• Updated with 0.7 spec review feedback.	June 2013
327432-004	1.0	• Included ECN– Change Alert# pin behavior (10-1-2014). • Included ECN- Clarify OOB packet payload (10-1-2014). • Updated with 0.75 spec review feedback.	January 2016

§ §

1 Glossary

Table 1: Table of Glossary

Term	Definition
TBD	**TBD**

§

2 Introduction

This base specification describes the architecture details of the Enhanced Serial Peripheral Interface (eSPI) bus interface for both client and server platforms.

The server platform specific support in addition to the base specification is described in a separate addendum document.

The devices that can be supported over the eSPI 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.

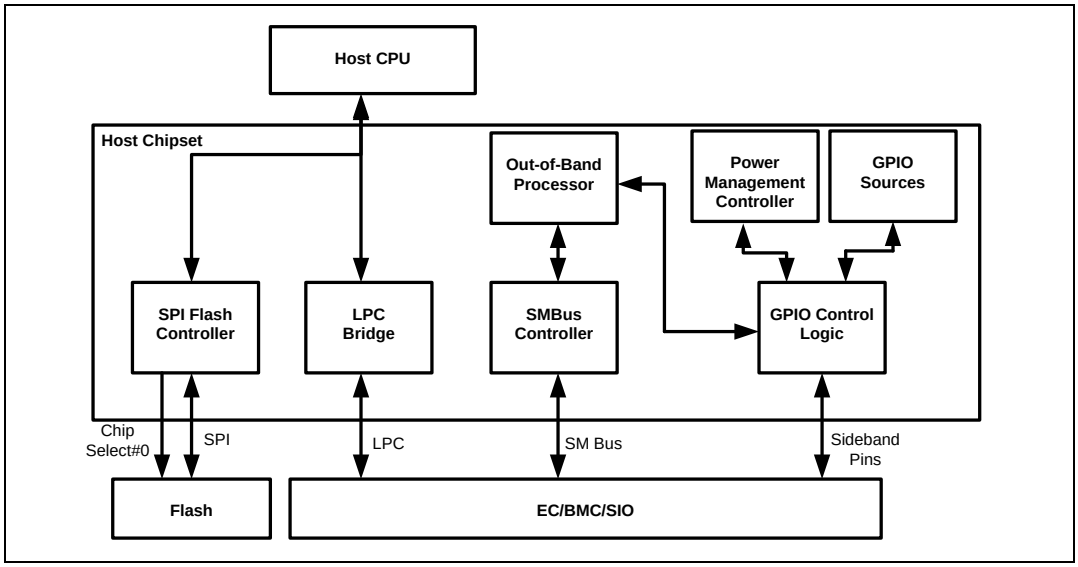
The specification generally refers to EC/BMC/SIO as the LPC device for the purpose of illustrating the eSPI bus capabilities and the comparison to LPC bus. However, EC/SIO is applicable for client platforms whereas BMC is generally associated with server platforms.

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 an EC/BMC/SIO is connected to the LPC bus.

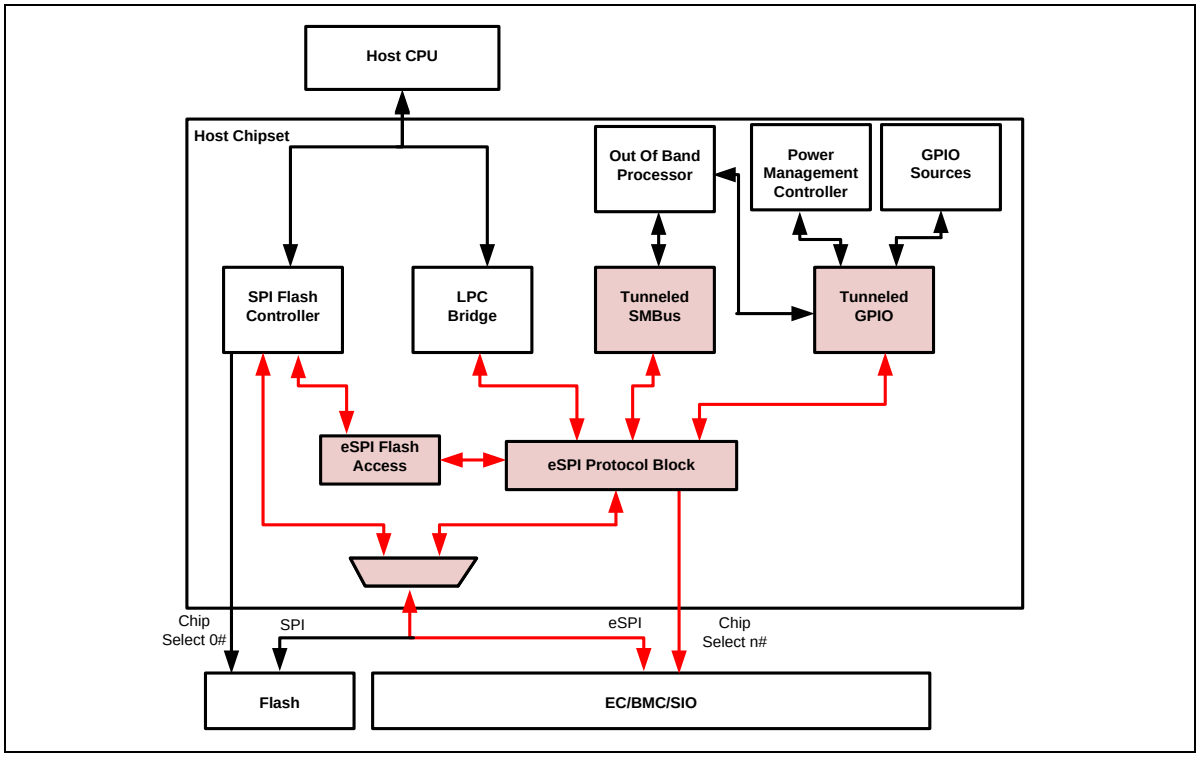
Figure 1: EC/BMC/SIO Communication over LPC



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 an 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.

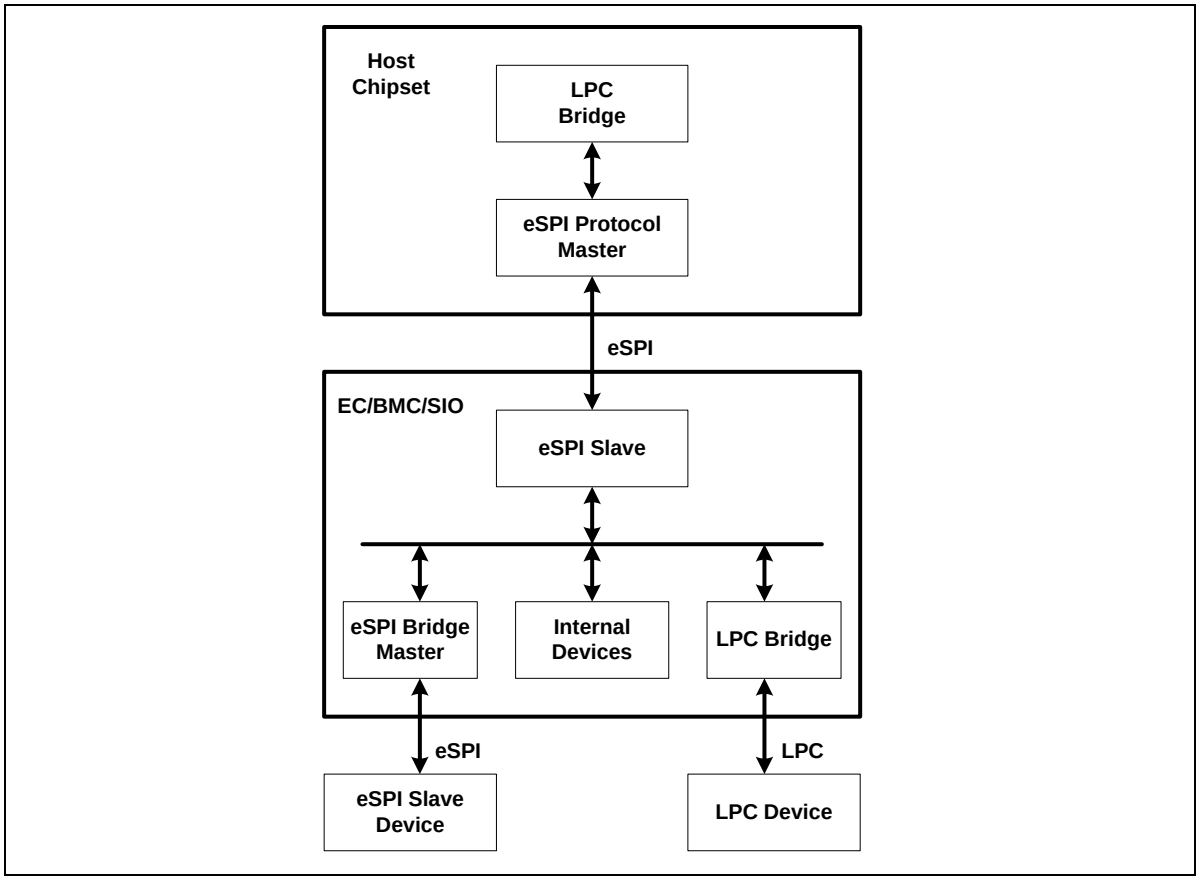
Out-Of-Band (OOB) messaging between Out-Of-Band Processor in the chipset and Embedded Controller (EC) or Baseboard Management Controller (BMC) is also tunneled through the new eSPI interface as in-band messages, thus replacing the SMBus interface for this purpose.

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.

The eSPI specification does not preclude the support of LPC bus behind eSPI and/or additional eSPI bus behind eSPI although the detail is outside the scope of the current specification. One of the possible system configurations is as shown below.

Figure 3: Example of LPC bus and Additional eSPI bus behind the eSPI



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, the eSPI master must ensure that only one of the Chip Select# pins is asserted based on source decode, thus 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#. 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 4: Single Master-Single Slave with eSPI Reset# from Slave to Master

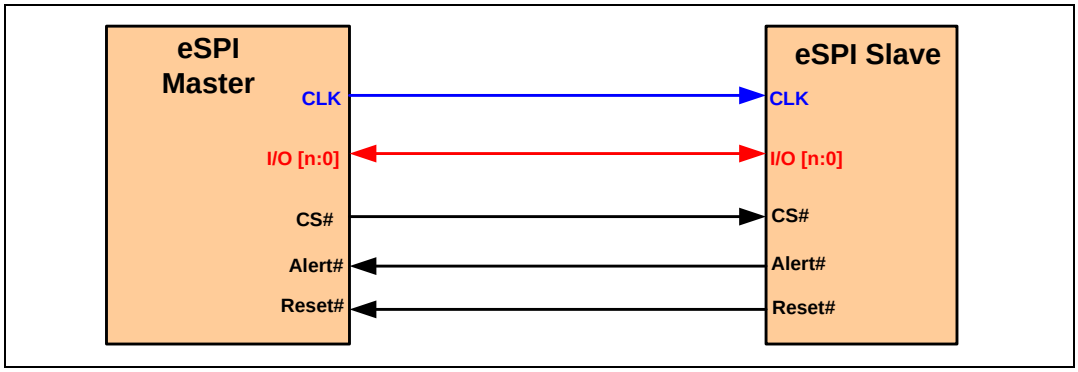


Figure 5: Single Master-Single Slave with eSPI Reset# from Master to Slave

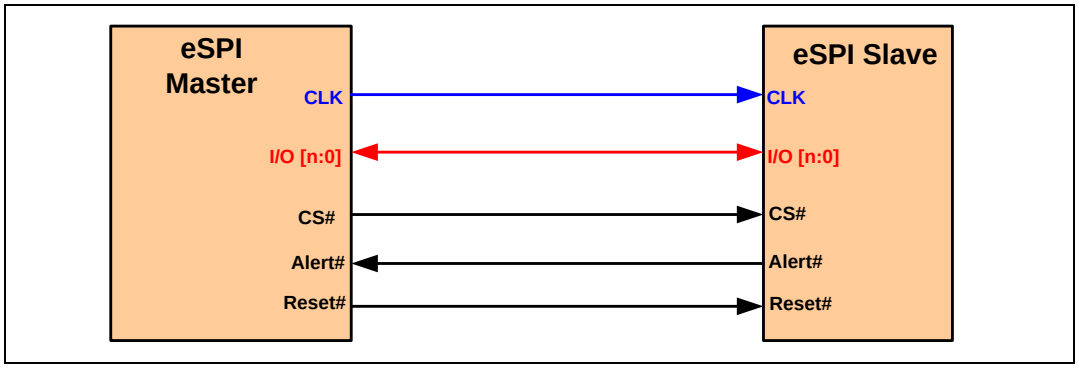
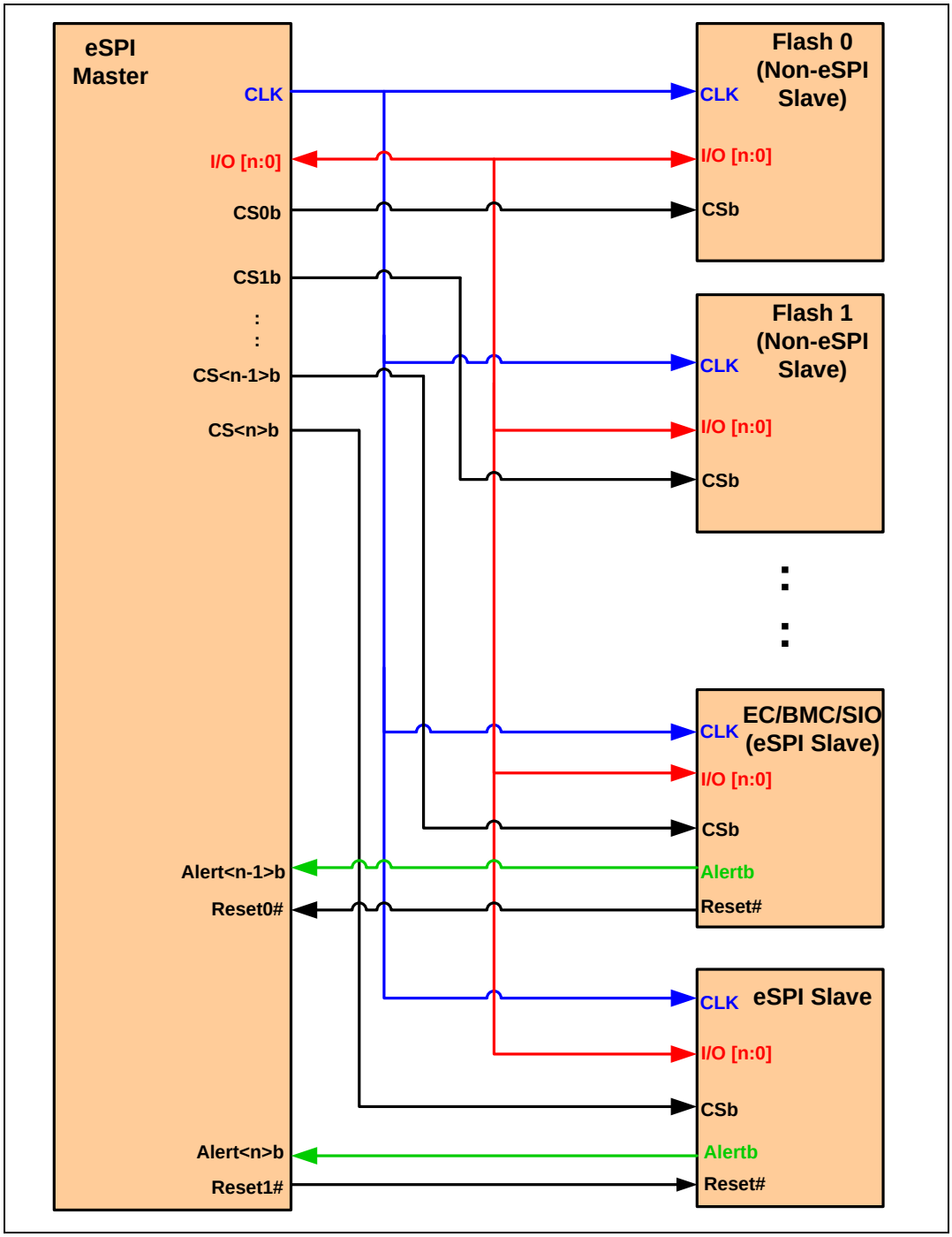


Figure 6: Single Master-Multiple Slaves with Two eSPI Reset#



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.

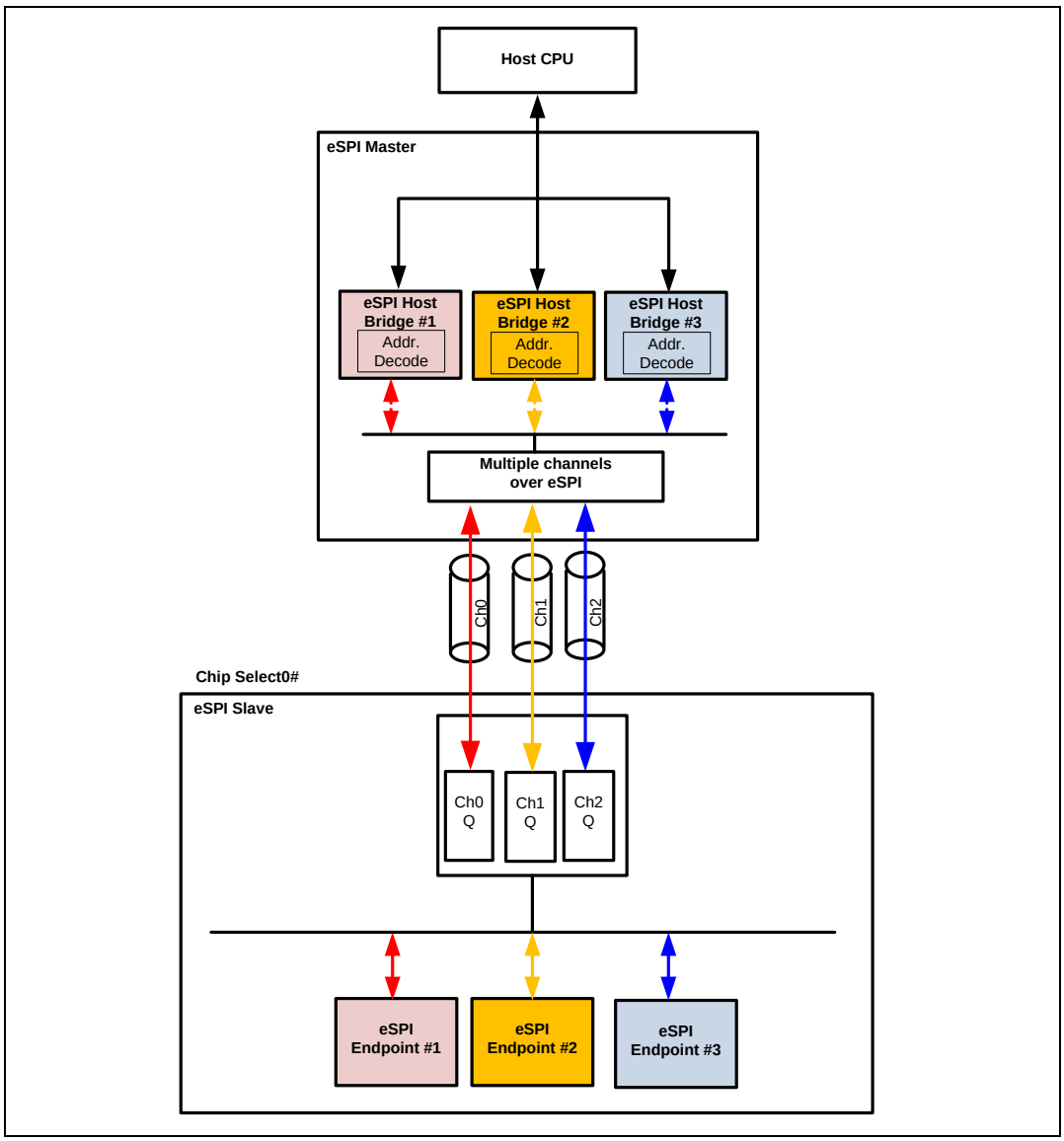
In an eSPI bus configuration with multiple slaves present, the eSPI master may support 2 eSPI Reset# pins, one from eSPI slave to eSPI master and another one from eSPI master to eSPI slaves. In this case, the master's eSPI interface will only be reset if all the slaves' eSPI interfaces are reset.

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 7: 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.

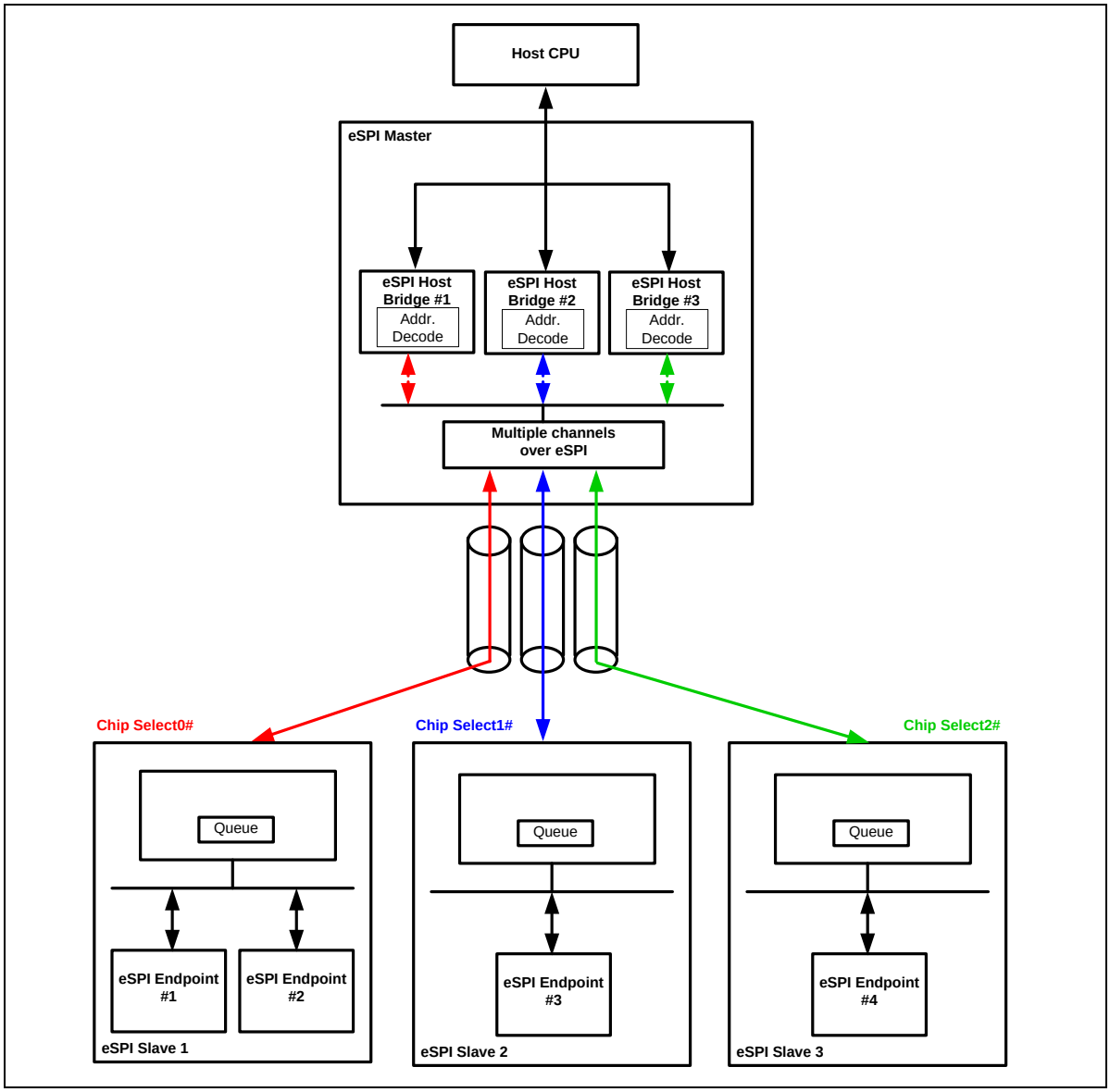
The use of channels allows multiple independent flows of command and data to be transferred over the same bus between the eSPI master and eSPI slave with no ordering requirement.

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.

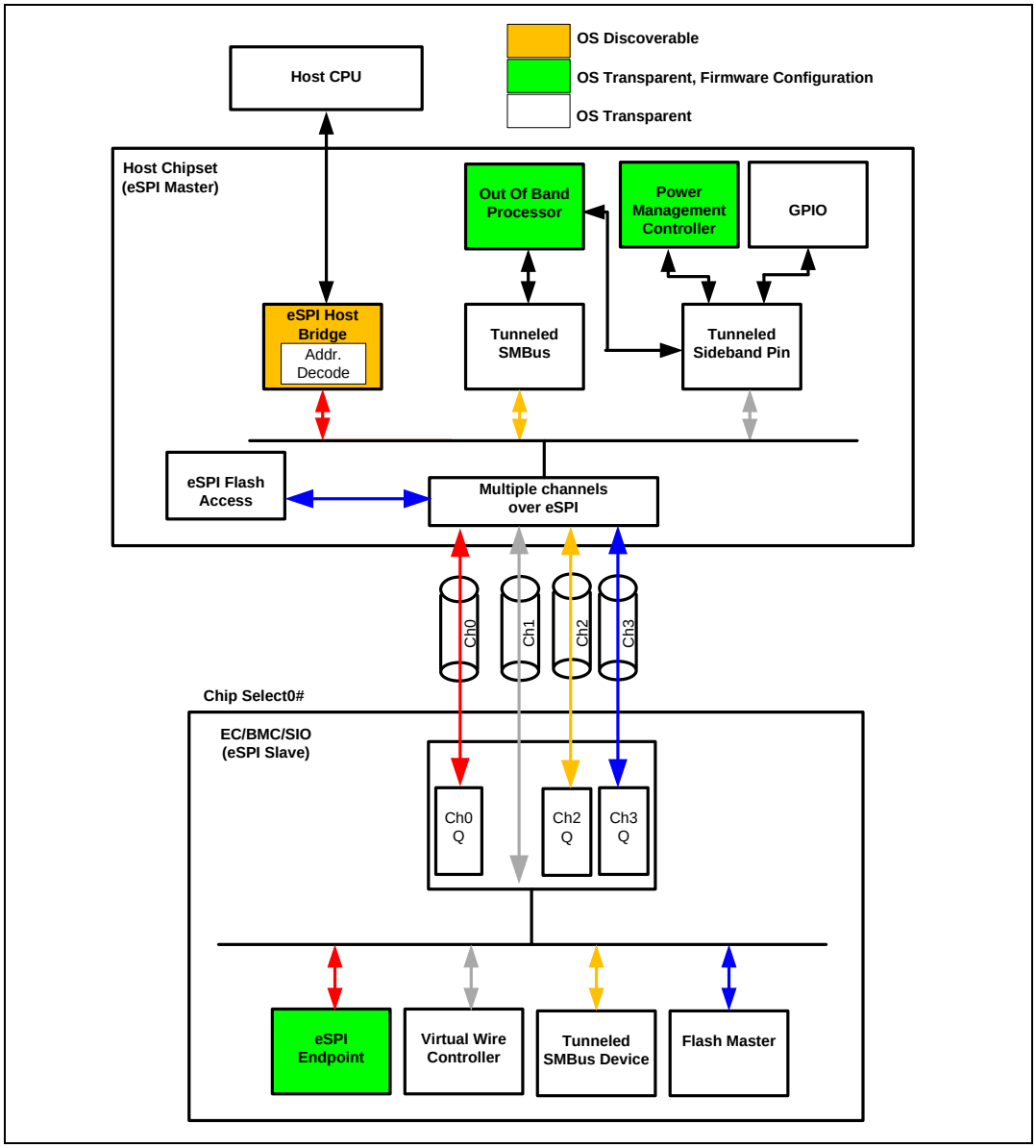
At any one time, only one of the Chip Select# pins can be asserted. Command and data transfer can then happen between the eSPI host bridge and the corresponding eSPI slave.

Figure 8: 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 9: 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, 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. The Chip Select# and I/O[n:0] pins require weak pull-up to be enabled on these pins whereas the Serial Clock requires a weak pull-down. When functions as a driven output, Alert# pin does not require a weak pull-up to be enabled on the pin unless for the purpose of terminating the pin to inactive when it is not used. When functions as an open-drain output, Alert# pin requires a weak pull-up to be enabled on the pin.

The weak pull-up/pull-down 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/pull-down. For Alert# pin configured as open-drain, it is recommended that the weak pull-up be implemented on the board such that its impedance value could be adjusted accordingly when needed.

Refer to Section 8.1 - Electrical Specification for the value of the weak pull-up/pull-down 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 the weak pull-down on the Serial Clock are allowed to be disabled after the eSPI Reset# deassertion. However, I/O[n:0] and Alert# pin (open-drain) continue to have the weak pull-up enabled for the proper operation of the eSPI bus.

Table 2: 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, driven from eSPI slave to eSPI master.

Pin Name	Direction	Clock	Description
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 1 or 3 depending on the I/O mode. In Single I/O mode (n=1), 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 either a driven, or an open-drain output from the slave with default as a driven output. This pin is optional for Single Master-Single Slave configuration where I/O[1] can be used to signal the Alert event.

eSPI从设备使用此引脚从eSPI主设备请求服务。

§

4 Bus Protocol

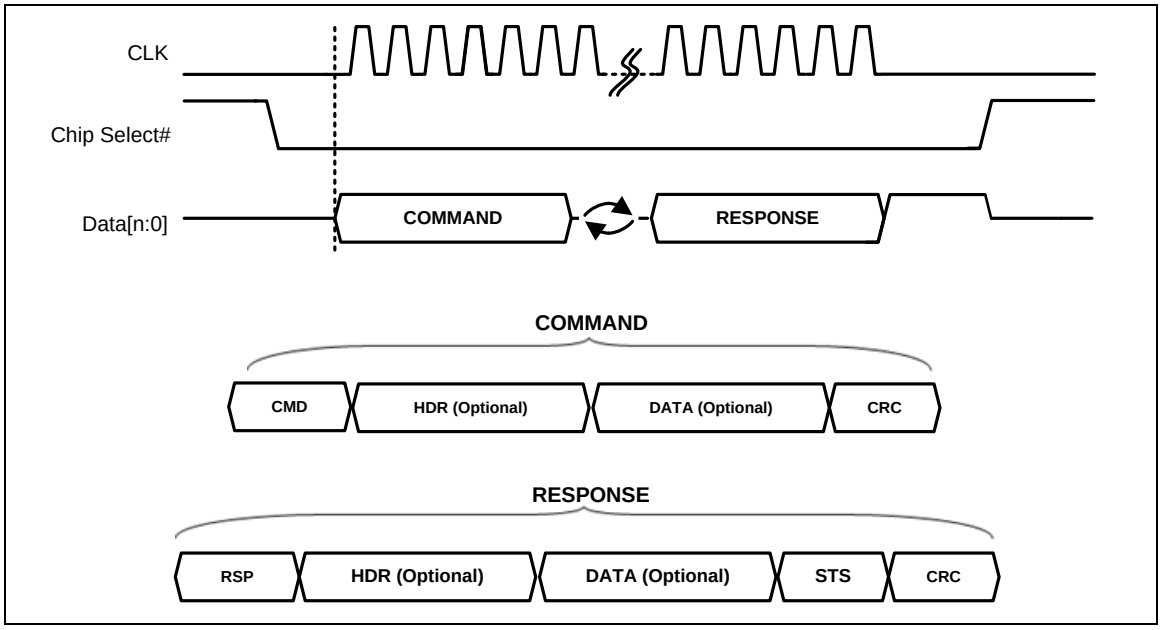
The details of the Enhanced Serial Peripheral Interface (eSPI) protocol are described in this section. The electrical of eSPI bus is similar to SPI bus with deviations specifically called out in this specification.

The Serial Clock must be low at the assertion edge of the Chip Select# while eSPI Reset# has been de-asserted. The first data is launched from master while the serial clock is still low and sampled on the first rising edge of the clock by slave. Subsequent 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 from slave on the falling edge of the clock. 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

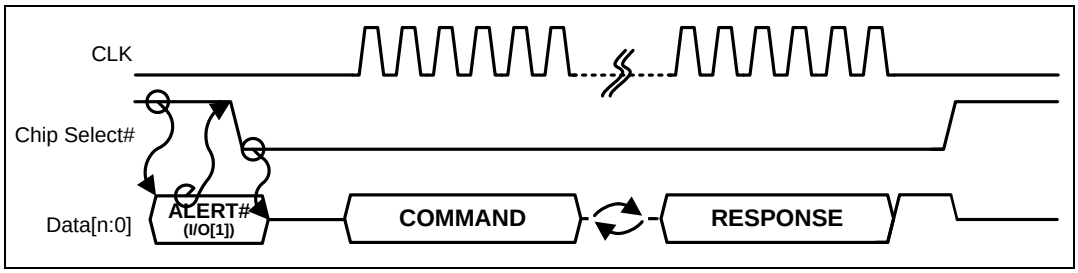
Figure 10: Basic eSPI Protocol



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 a CRC. The Response phase consists of a RSP, an optional header (HDR), optional data, a Status and a CRC. CRC generation is mandatory for all eSPI transactions where CRC byte is always transmitted on the bus. However, CRC checking is default disabled after reset and it is enabled by SET CONFIGURATION. When CRC checking is disabled, CRC byte is ignored by the receiver.

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.

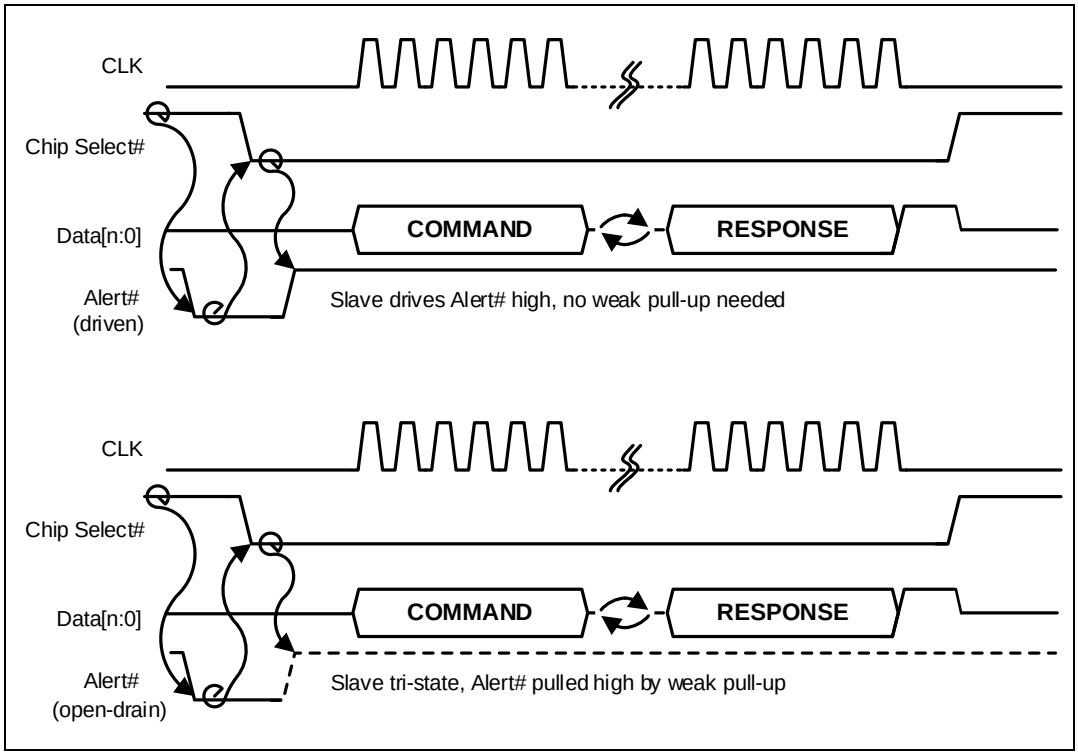
The Alert event can only be signaled by the slave when the slave's Chip Select# is high.

When I/O[1] is used to signal the Alert# event, it is toggled from tri-state to pulled low by the slave when the slave decides to request for service. The slave then holds the state of the I/O[1] 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 I/O[1] pin by tri-stating the pin within the t_{SLAZ} timing 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. At the last falling edge of the serial clock after CRC is sent, the eSPI slave must drive I/O[n:0] pins to high until Chip Select# is deasserted. Besides power friendly due to weak pull-up on these pins, the driving to high ensures no false Alert# event is generated by I/O[1] when Chip Select# is deasserted. At the deassertion edge of Chip Select#, these I/O[n:0] pins are tri-stated by the slave meeting the t_{SHQZ} Output Disable timing where the weak pull-ups maintain these pins at high, with the master continues to tri-state the I/O[n:0]. To signal an alert event after Chip Select# deassertion, the slave is only allowed to re-assert the I/O[1] pin after the t_{SHAA} timing.

When Alert# pin is used to signal the Alert# event, it is toggled by the slave from high to low (when the pin is a driven output) or tri-state to pulled low (when the pin is an open-drain output) when the slave decides to request for service. The I/O[n:0] pins remain tri-stated by the slave. The slave then holds the state of the Alert# pin until the Chip Select# is asserted by the master. Once the Chip Select# is asserted, the slave must drive the Alert# pin high (when the pin is a driven output), or release the ownership of the pin by tri-stating the pin (when the pin is an open-drain output). The t_{SLAZ} timing is not applicable to the Alert# pin. The master then continues to issue command to figure out the cause of the Alert event from the device and then service the request. At the last falling edge of the serial clock after CRC is sent, the eSPI slave must drive I/O[n:0] pins to high until Chip Select# is deasserted for power friendly reason due to weak pull-up on these pins. After Chip Select# deassertion, these I/O[n:0] pins are tri-stated by the slave after meeting the t_{SHQZ} Output Disable timing where the weak pull-ups maintain these pins at high, with the master continues to tri-state the I/O[n:0]. The t_{SHAA} timing is not applicable to Alert# pin. However, when Alert# pin is configured as open-drain and asserted, the weak pull-up on the pin must be such that the assertion of the CS# for the shortest possible transaction (which causes the slave to tri-state the Alert# pin), is able to pull the Alert# pin high fast enough to the deasserted value before or by the last failing edge of the serial clock at the end of the transaction.

In the case of error condition where Chip Select# is deasserted abruptly by the master, refer to Section 9.2.1.5 for the detail.

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 I/O[1] pin or drive the Alert# pin high (when the pin is a driven output) or tri-state the Alert# pin (when the pin is an open-drain output) 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.

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 a CRC byte.

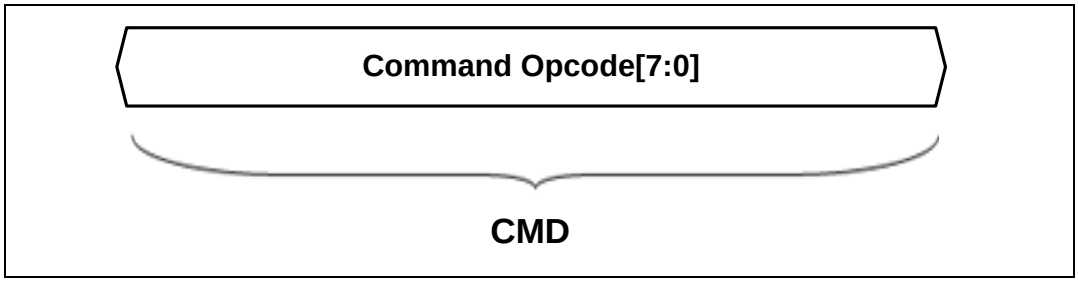
Similarly on the receive side, the CRC is checked at the receiving Link Layer when CRC checking is enabled. Once the packet passes the CRC check, the packet is sent to Transaction Layer where it is decoded and acted upon.

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 a CRC.

The CMD field consists of Command Opcode.

Figure 13: Command Opcode



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 packet with an invalid Command Opcode which is not defined by this specification, the slave must not respond to the transaction. The transaction will be terminated with the default response (NO_RESPONSE) on the bus.

Table 3: 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. Refer to Table 6 for the cycle types and the respective packet format.

CMD Opcode	Encoding[7:0]	Description
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. Refer to Table 6 for the cycle types and the respective packet format.
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 Refer to Table 6 for the cycle types and the respective packet format.
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. Refer to Table 6 for the cycle types and the respective packet format.
PUT_IORD_SHORT	$010000C_1C_0$[1]	Put a short (1, 2 or 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. Refer to Figure 37 for the packet format.
PUT_IOWR_SHORT	$010001C_1C_0$[1]	Put a short (1, 2 or 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. Refer to Figure 37 for the packet format.

CMD Opcode	Encoding[7:0]	Description
PUT_MEMRD32_SHORT	010010C_1C_0[1]	Put a short (1, 2 or 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. Refer to Figure 37 for the packet format.
PUT_MEMWR32_SHORT	010011C_1C_0[1]	Put a short (1, 2 or 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. Refer to Figure 37 for the packet format.
Virtual Wire Channel		
PUT_VWIRE	00000100	Put a Tunneled virtual wire packet. Refer to Figure 40 for the packet format.
GET_VWIRE	00000101	Get a Tunneled virtual wire packet. Refer to Figure 40 for the packet format.
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. Refer to Table 6 for the cycle types and the respective packet format.

CMD Opcode	Encoding[7:0]	Description
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. Refer to Table 6 for the cycle types and the respective packet format.
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. Refer to Table 6 for the cycle types and the respective packet format.
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. Refer to Table 6 for the cycle types and the respective packet format.
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.

CMD Opcode	Encoding[7:0]	Description
GET_CONFIGURATION	00100001	Command to discover the capabilities of the slave as part of the initialization.
RESET	11111111	In-band RESET command.

NOTES:

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	Reserved
11	4 bytes

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

4.3 Turn-Around (TAR)

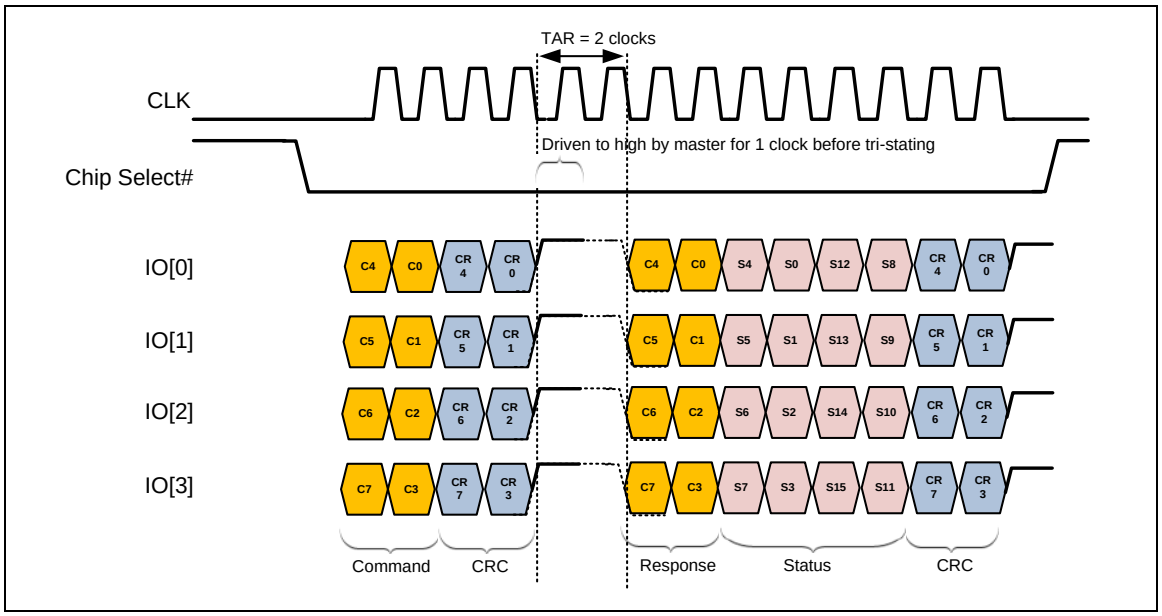
After the last bit of the Command Phase has been sent out on the data lines, the data lines enter the Turn-Around window. The eSPI master is required to drive all the data lines to logic '1' for the first clock of the Turn-Around window and tri-state the data lines thereafter. The number of clocks for the Turn-Around window is a fixed 2 serial clocks independent of the eSPI I/O Mode (single, dual or quad I/O). The slave may insert WAIT_STATE response code after the TAR window for any eSPI transactions if additional time is needed for the slave to sample the command and prepare the response.

The eSPI slave must not drive I/O[n:0] until the Response phase in all the eSPI I/O mode (single, dual, quad I/O). In single I/O mode particularly, the slave must not drive I/O[1] (MISO) until the Response phase. It must drive the Response phase on the bus immediately upon the expiry of the Turn-Around time as shown in the next diagram.

Optionally, the slave is allowed to start turning on its driver on the bus half a clock earlier at the end of the Turn-Around window (at the rising edge of the second clock) in preparation for driving the response code at the subsequent clock falling edge as required by the eSPI protocol.

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

Figure 14: Turn-Around Time (TAR = 2 clock)

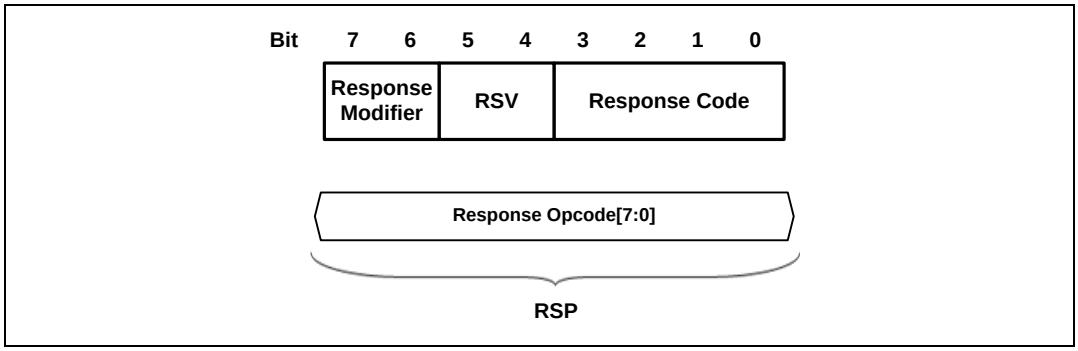


4.4 Response Phase

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, a STATUS and a CRC.

The RSP opcode is an 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, responded with error or wait state.

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” except for NO_RESPONSE where it is “11”.

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 flash access (channel 3) completion is only applicable when slave attached flash sharing is supported and in operation.

The Response Modifier is by default disabled. It is enabled through SET_CONFIGURATION by setting the Response Modifier Enable bit to ‘1’ in the General Capabilities and Configurations register. Refer to Section 7.2.1.3 for the register bit description.

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 4: Response Field Encodings

RESPONSE	Encoding			Description
	[7:6]	[5:4]	[3:0]	
ACCEPT	R_1R_0[1]	RSV	1000	Command was successfully received 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	0001	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	0010	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	0011	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 so forth.
WAIT_STATE	00	RSV	1111	Adds one byte-time of delay when responding to a transaction on the bus.
NO_RESPONSE	11	11	1111	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.

NOTES:

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. This is only applicable when slave attached flash sharing is supported and in operation.

4.4.2 Status

The 16-bit Status field serves to provide information such as new pending requests from the slave and queue free information. For status bits related to channels that are not enabled or if channels are enabled but not ready, or status bits for features that are not supported, these bits are don't care and must be ignored by the eSPI master. The reserved status bits must be driven to '0' by the slave.

The status field reflects the real time status of the slave at the point when the status field is transmitted on the bus. The AVAIL and FREE reflect the queue status after taken into account the command being received or sent in this transaction. However, as the command received is only decoded after the deassertion of CS#, the effect of the command (such as the SET_CONFIGURATION as an example) to the queue status if any, will not be reflected in the status field of the current transaction. The change of the queue status if any, will be signaled by the slave through ALERT# and reflected in the status field of the subsequent transaction.

The order of the Status bytes transmitted on the eSPI bus is described in Section 6.1.

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

Figure 16: Slave's Status Register Definition

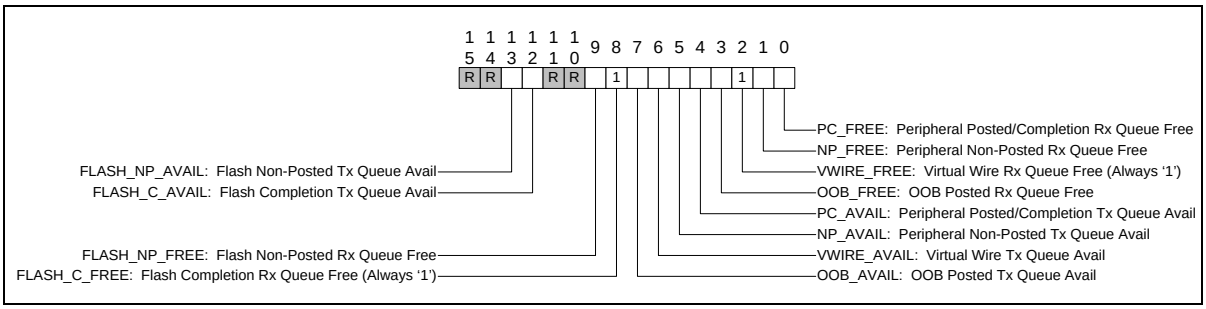


Table 5: Status Field Encodings

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. This bit is only applicable when master attached flash sharing is supported and in operation. Otherwise, the bit is a don't care.
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. This bit is only applicable when slave attached flash sharing is supported and in operation. Otherwise, the bit is a don't care.
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. This bit is only applicable when slave attached flash sharing is supported and in operation. Otherwise, the bit is a don't care.
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. This bit is only applicable when master attached flash sharing is supported and in operation. Otherwise, the bit is a don't care.

STATUS	Bits Position	Description
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

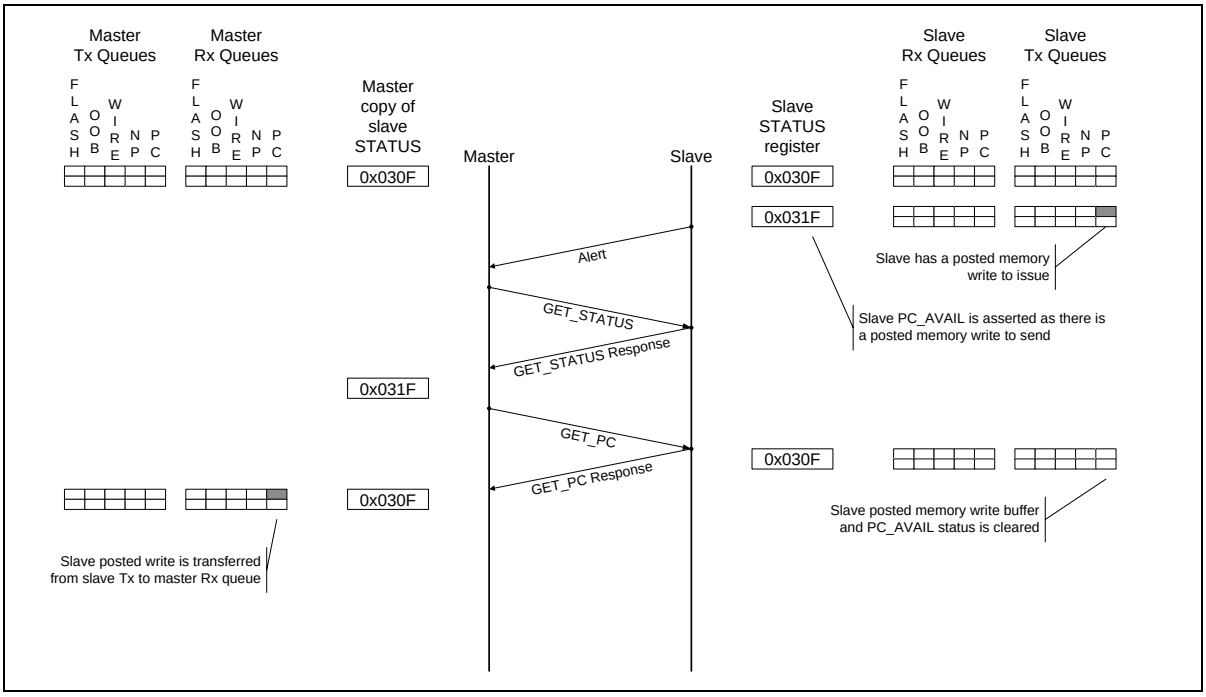


Figure 18: Flow Diagram for a Back-to-back Slave to Master Peripheral Posted Write

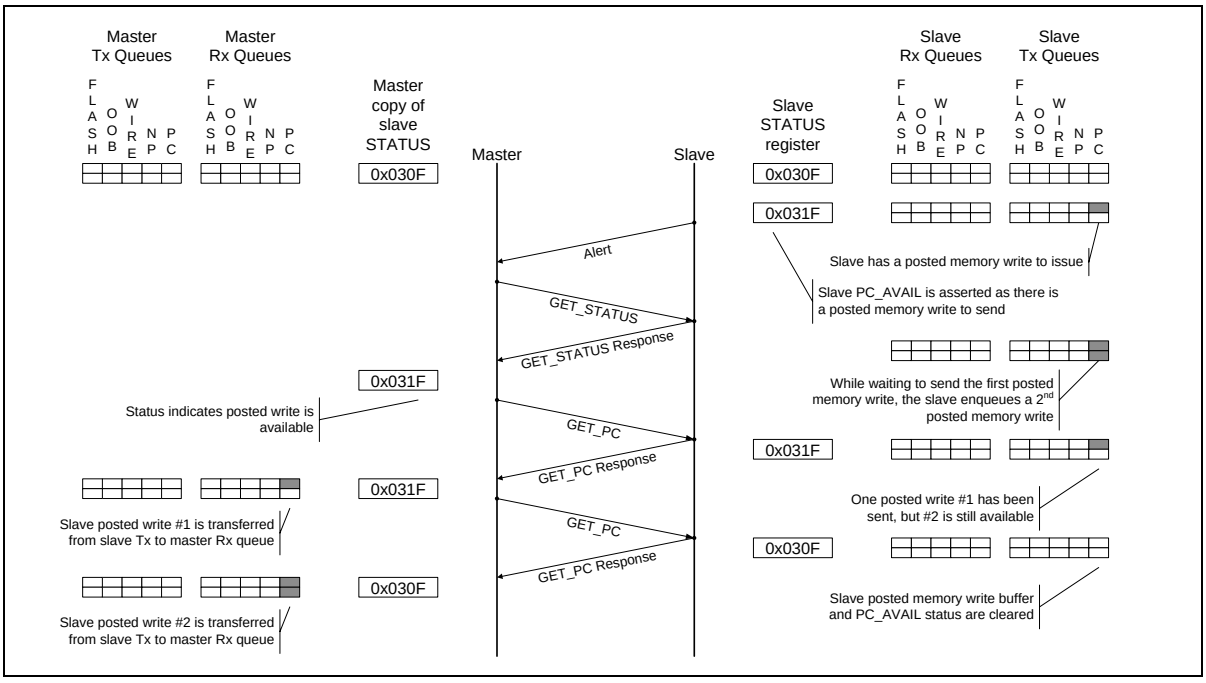
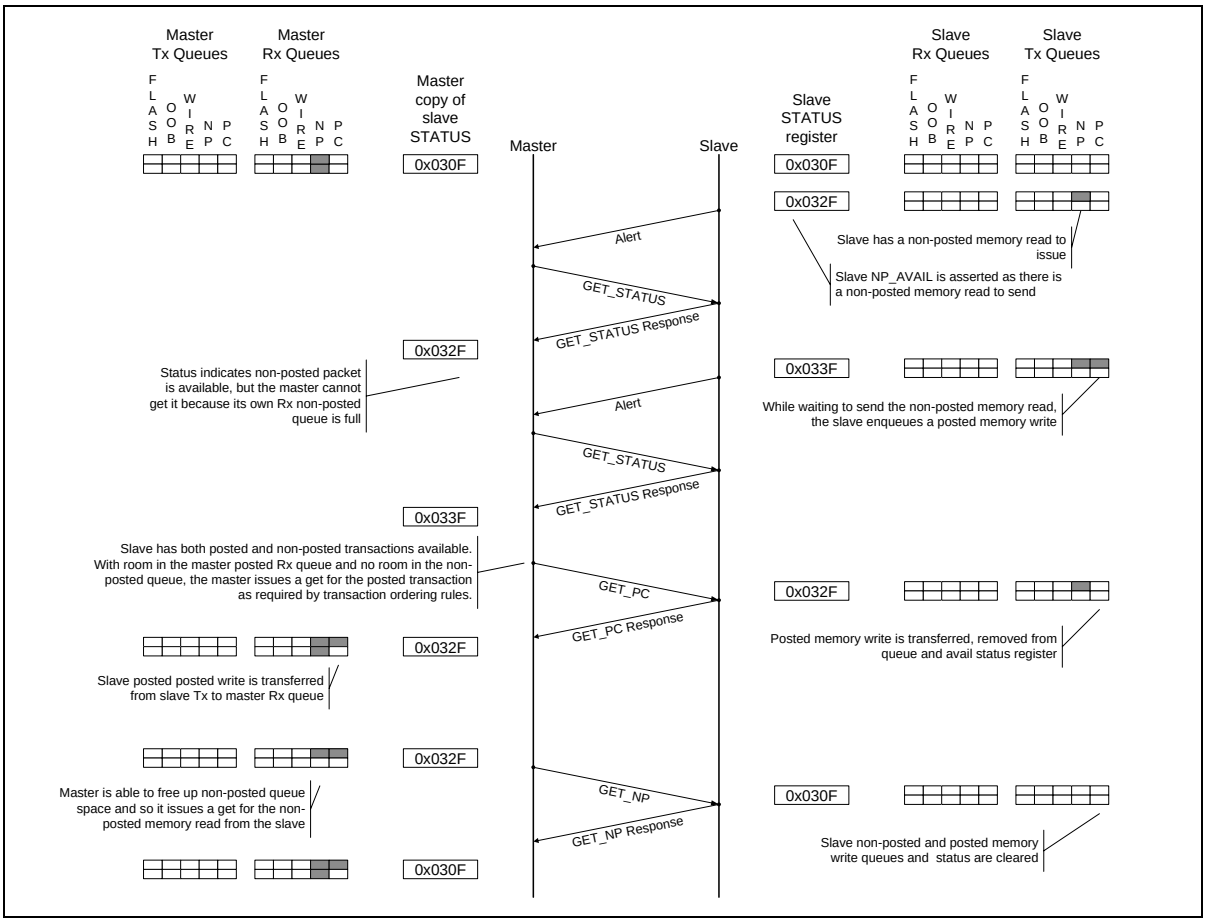
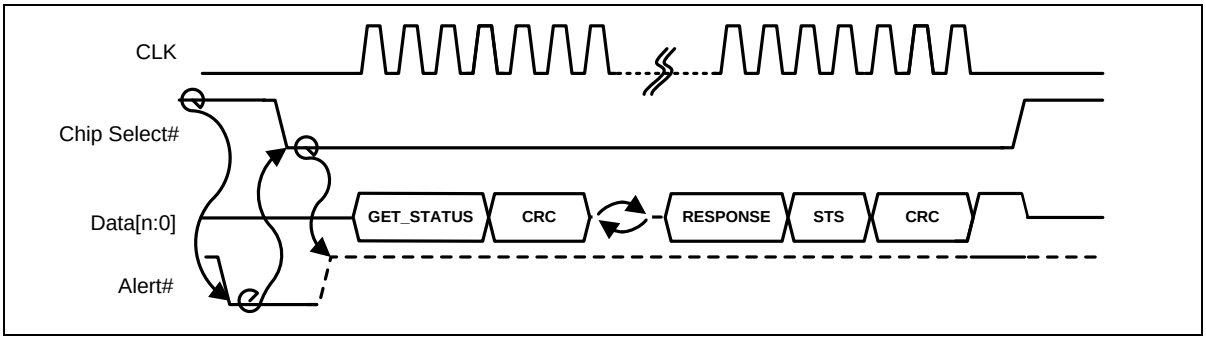


Figure 19: Flow Diagram for a Slave to Master Peripheral Posted Write passes Non-posted



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.

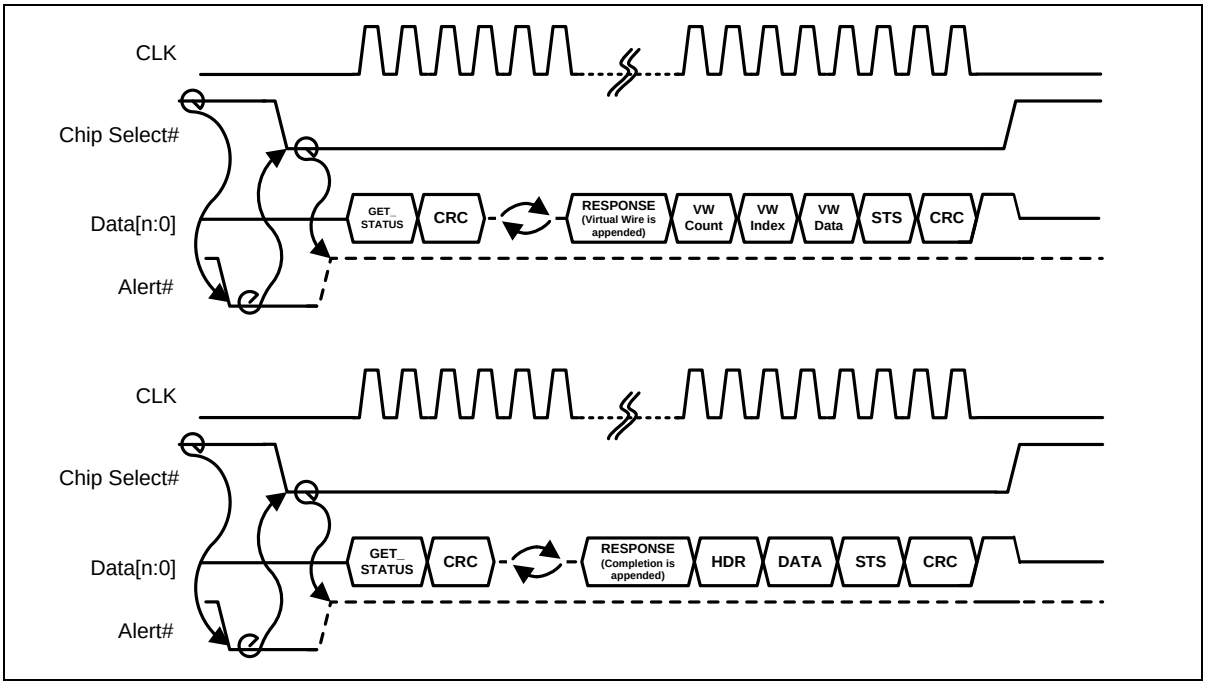
This command is typically used in response to the Alert event from the eSPI slave, to determine the cause of the Alert event and subsequently service the slave.

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 peripheral or flash access completion appended may be a partial or full completion corresponding to a prior non-posted transaction to the slave.

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)

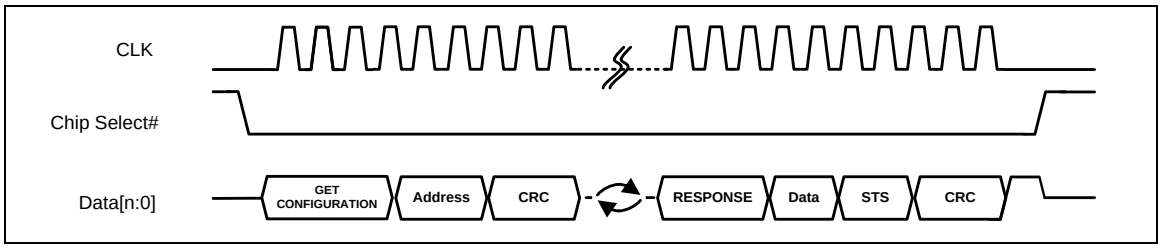


4.7 Get Configuration and Set Configuration Command

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.

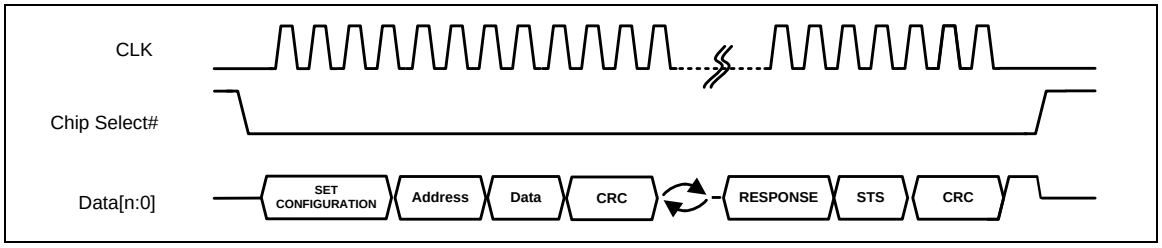
SET_CONFIGURATION and GET_CONFIGURATION commands can never be deferred and must be completed within the same cycle.

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, a 16-bit address and an 8-bit CRC. The response phase includes an 8-bit Response, 1 DW of Data, a 16-bit Status and an 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, a 16-bit address, 1 DW of Data and an 8-bit CRC. The response phase includes an 8-bit Response, a 16-bit Status and an 8-bit CRC.

eSPI slave must only be configured with capabilities that advertised as supported. Configuring eSPI slave through SET_CONFIGURATION with unsupported capabilities will result in undefined behavior which is implementation specific and beyond the scope of the specification.

The order of address bytes transmitted on the eSPI bus during GET_CONFIGURATION and SET_CONFIGURATION is described in Section 6.1.

The eSPI slave contains addressable register space up to 4 KB. 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.

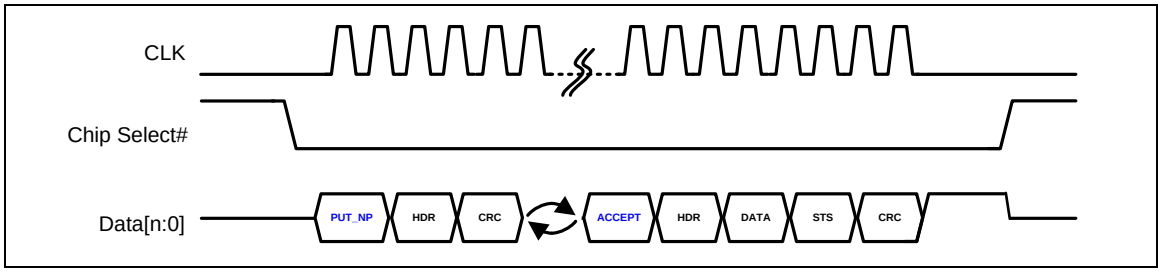
4.8 Non-Posted Transaction

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 with either a successful or unsuccessful completion, FATAL ERROR and NON-FATAL ERROR.

Figure 24: Connected Master Initiated Non-Posted Transaction

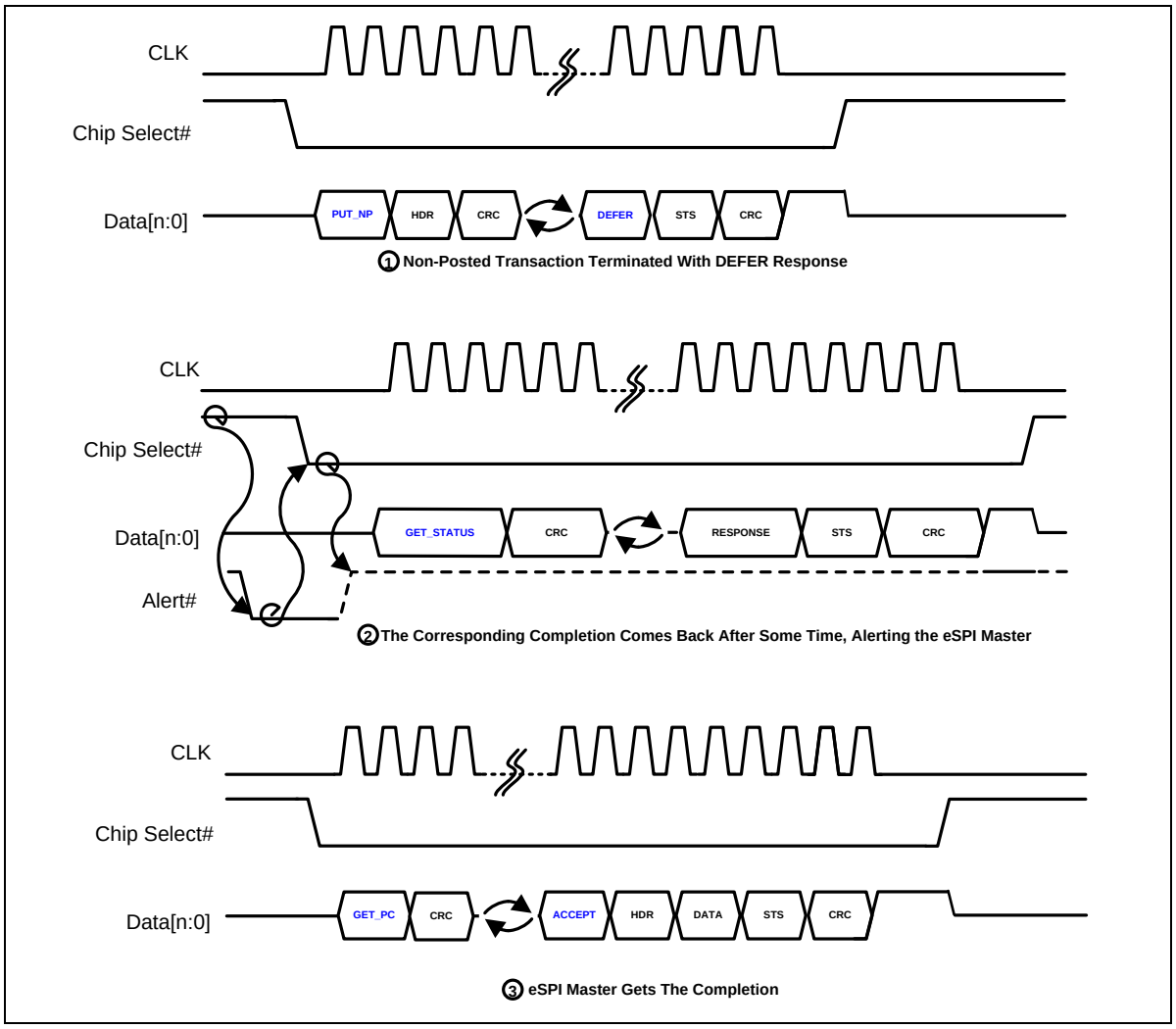


If the eSPI master initiated non-posted transaction requires data or additional information which is not available immediately, the non-posted request is terminated with a “DEFER” response. The deferred completion can be returned some period of time in the future when the data or information is eventually 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 only valid response is ACCEPT with either a successful or unsuccessful completion. For non-posted transaction that will be terminated with error, the slave is required to respond with FATAL ERROR or NON-FATAL ERROR as connected without deferring the transaction.

The eSPI slave can complete the non-posted command with multiple split completions either as connected or deferred. Refer to Section 5.1.3 for cases with 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

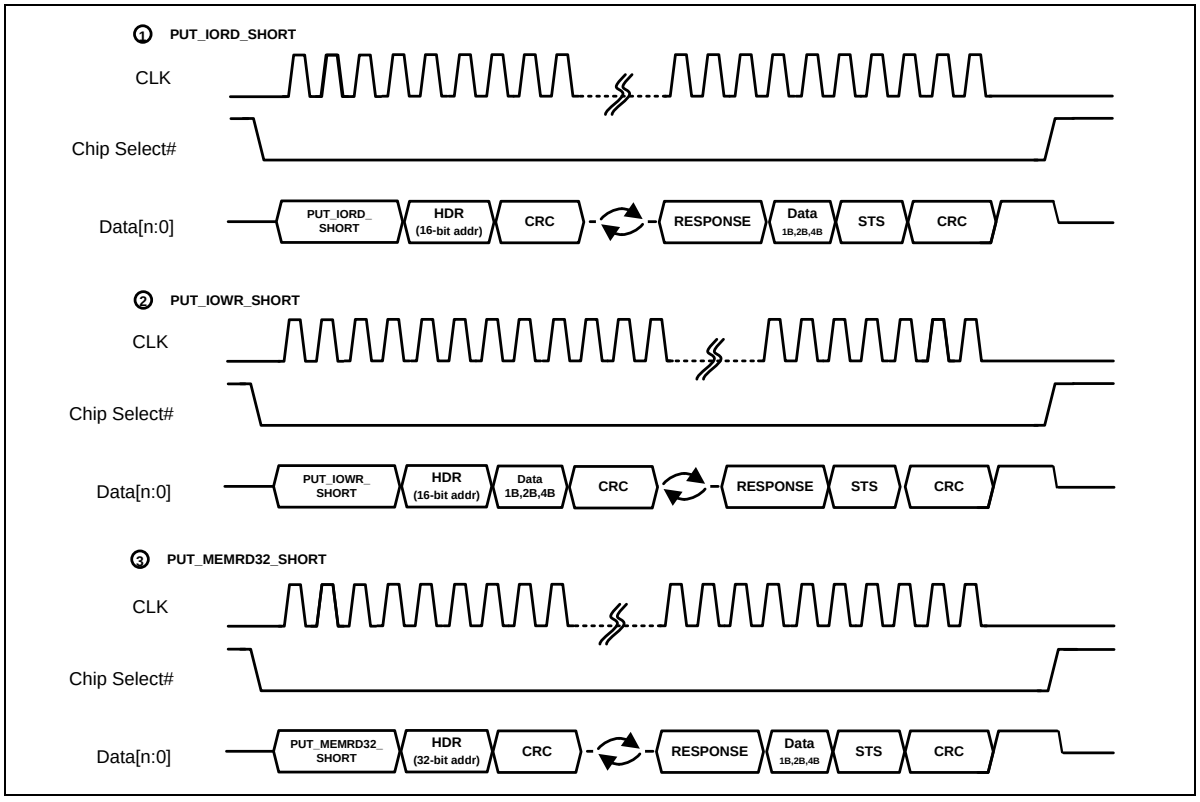


eSPI supports short non-posted transactions from master to slave for requests of length 1, 2 or 4 bytes that have less overhead and are thus more efficient. The unique opcode indicates the type of non-posted transaction and the request length. The header contains the address only and the number of address 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

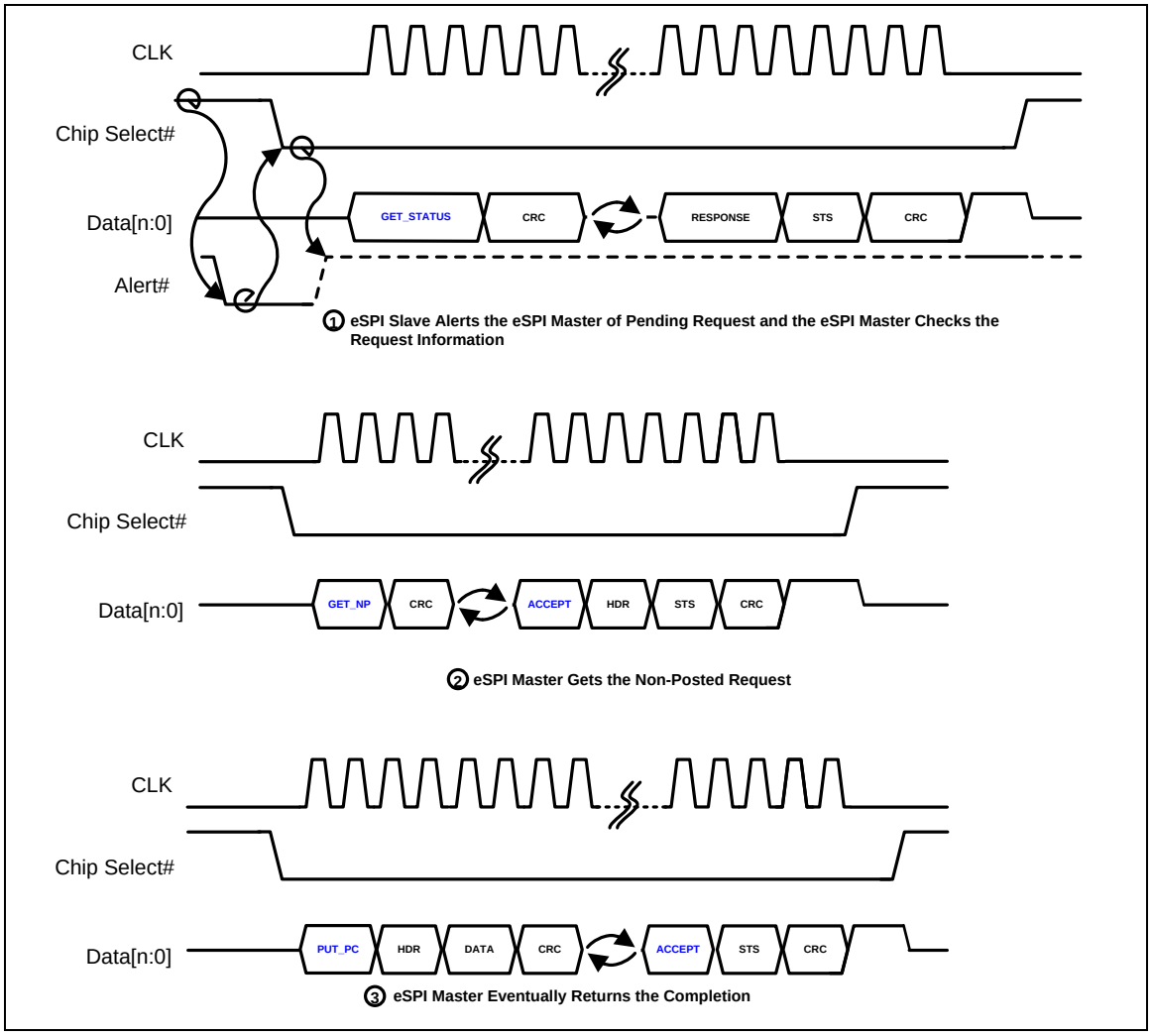


The eSPI slave can generate an Alert when there is a pending non-posted transaction. In response to that, the eSPI master would issue a GET_STATUS command to check for the pending request information.

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.

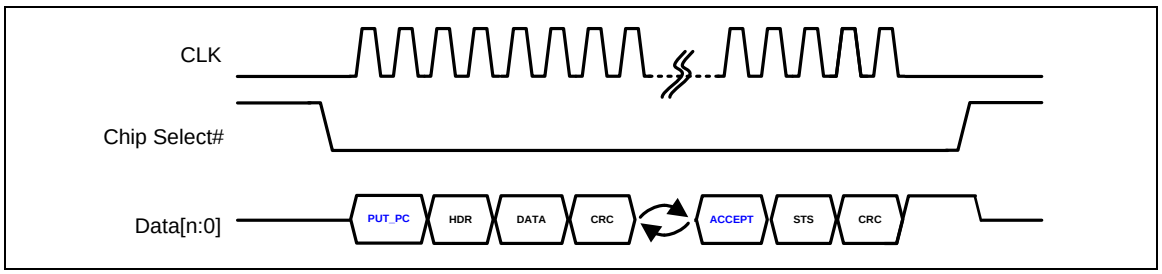
Figure 27: Slave Initiated Non-Posted Transaction



4.9 Posted Transaction

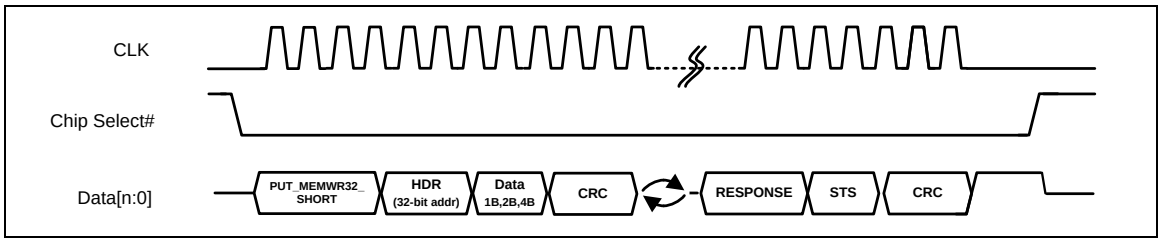
The valid responses for posted transactions initiated by eSPI master are ACCEPT, FATAL ERROR and NON-FATAL ERROR. DEFER response for posted transaction is invalid.

Figure 28: Master Initiated Posted Transaction



eSPI supports short posted transactions from master to slave for requests of length 1, 2 or 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.

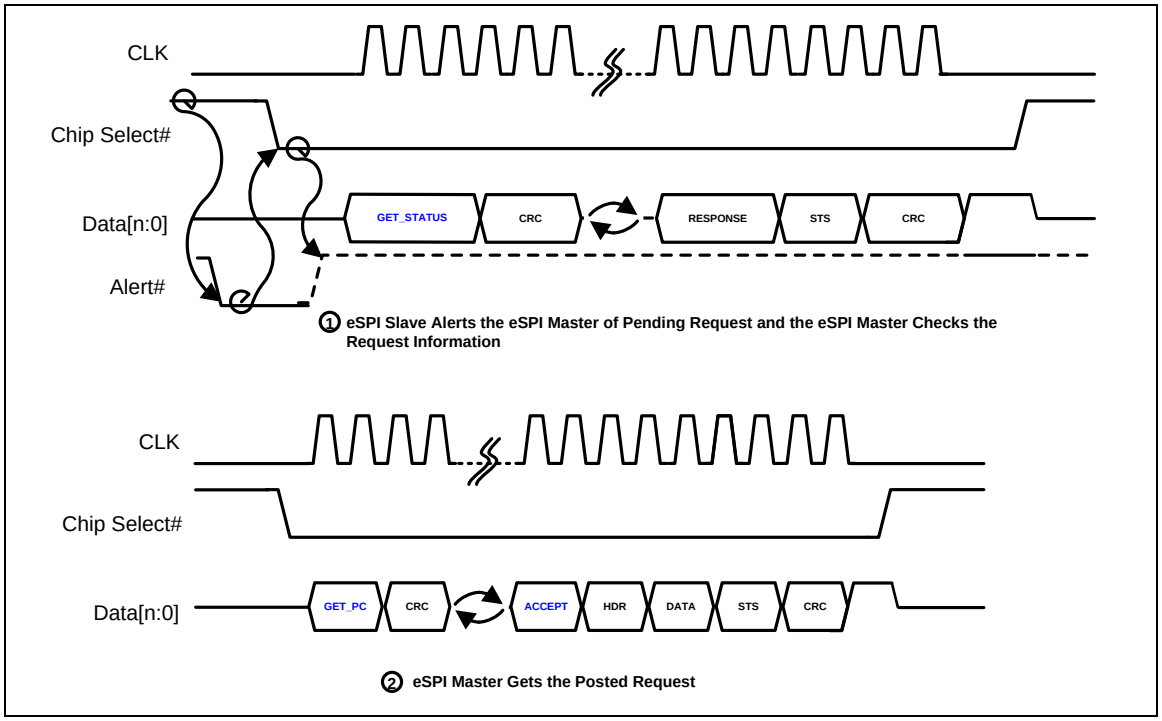
Figure 29: Master Initiated Short Posted Transaction



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.

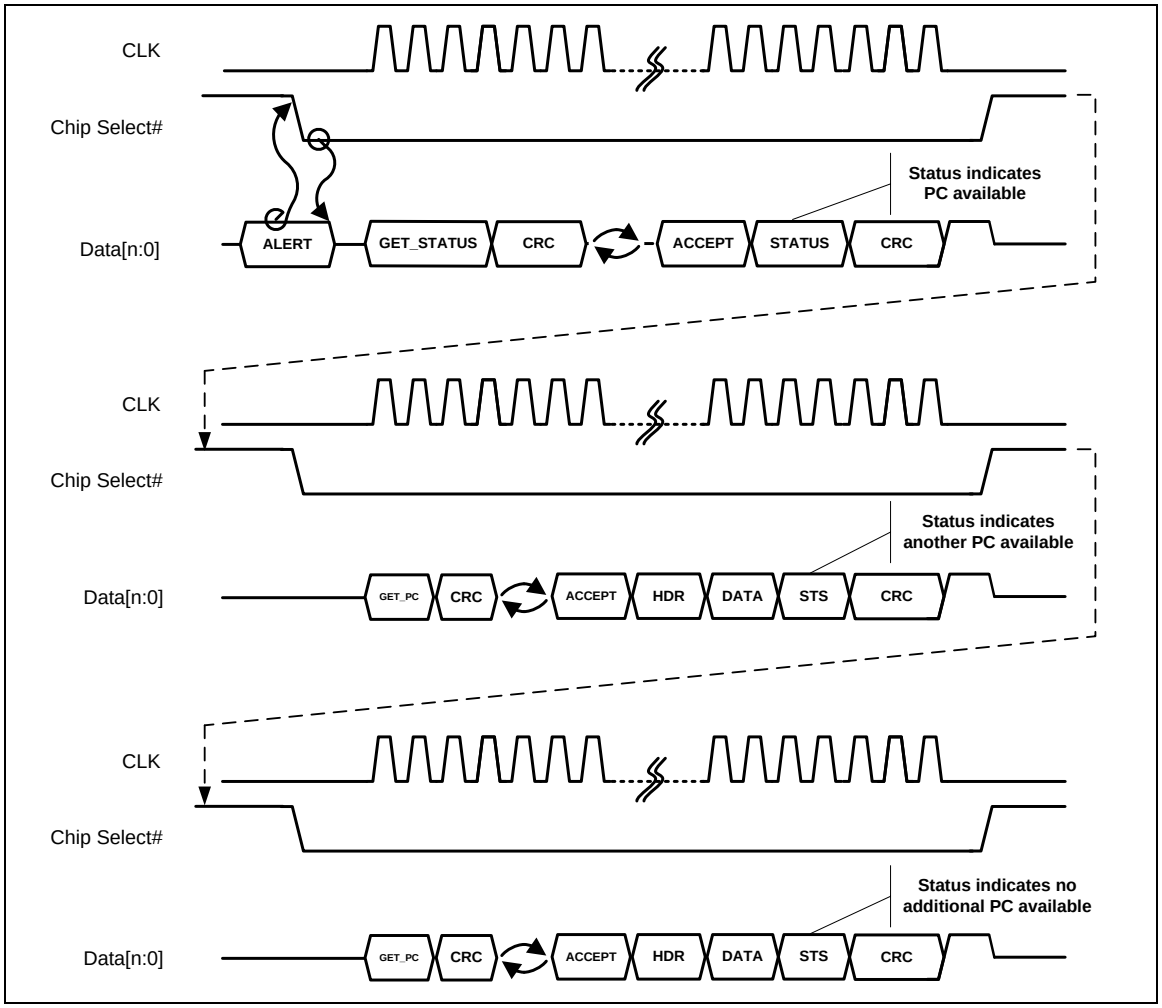
The eSPI master would then generate a GET_PC command to get the posted transaction.

Figure 30: Slave Initiated Posted Transaction



For illustration, a pipelined back-to-back bus mastering posted write transactions from eSPI slave with corresponding status indication is as shown below.

Figure 31: Pipelined Back-to-Back Bus Mastering Posted Write Transactions



4.10 WAIT STATE

All eSPI transactions support WAIT_STATEs by the eSPI slave during the response phase. After the 2 clocks Turn-Around (TAR) window, the eSPI slave is allowed to respond with WAIT_STATE Response Code before it terminates the transaction with ACCEPT, DEFER, NON-FATAL ERROR or FATAL ERROR.

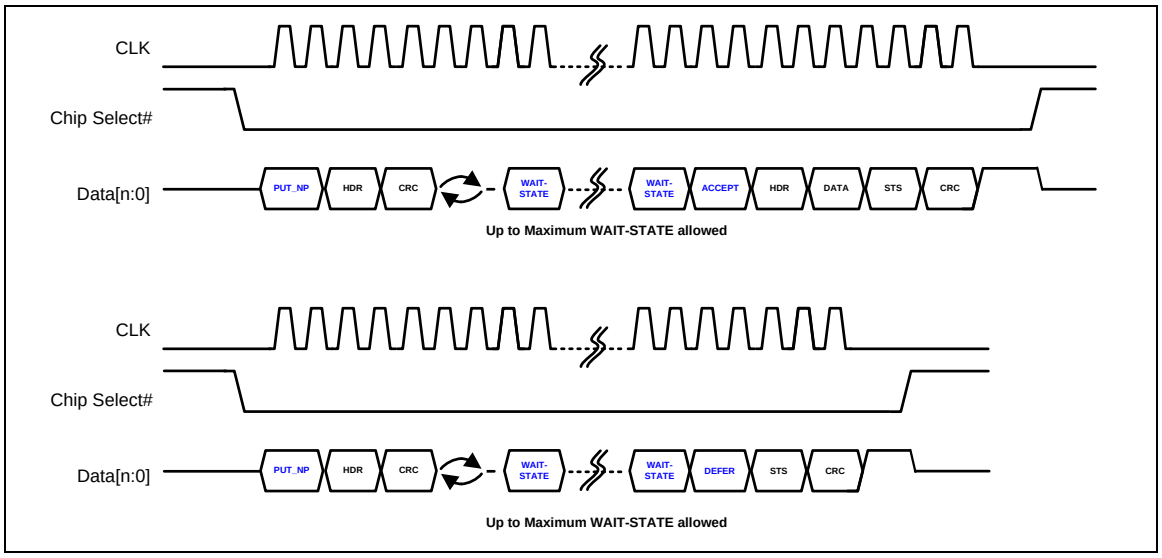
One or more WAIT_STATE Response Code may be inserted by the eSPI slave at the start of the Response Phase, up to the Maximum WAIT_STATE value configured by the eSPI master. The eSPI master is not required to check for Maximum WAIT_STATE violation. It is the eSPI slave's responsibility to ensure the number of WAIT_STATE Response Code inserted does not exceed the Maximum WAIT_STATE allowed.

The WAIT_STATE capability provides additional time beyond the Turn-Around (TAR) window if needed for the slave to sample the command and prepare the response. It

allows the master initiated non-posted transaction which would otherwise be responded immediately with DEFER, to be completed in the same transaction when some additional delay is needed by the eSPI slave beyond the Turn-Around (TAR) window. The additional delay provided by a WAIT_STATE Response Code is one byte time, which corresponds to 8 serial clocks in the Single I/O mode, 4 serial clocks in the Dual I/O mode or 2 serial clocks in the Quad I/O mode respectively.

WAIT_STATE Response Code is not included in the CRC calculation. It is defined with the encoding that has at least 2-bit differences compares to all other response code encodings. eSPI master is required to handle the wait state at the point WAIT_STATE Response Code is received.

Figure 32: Master Initiated Non-Posted Transaction Responded with WAIT STATE



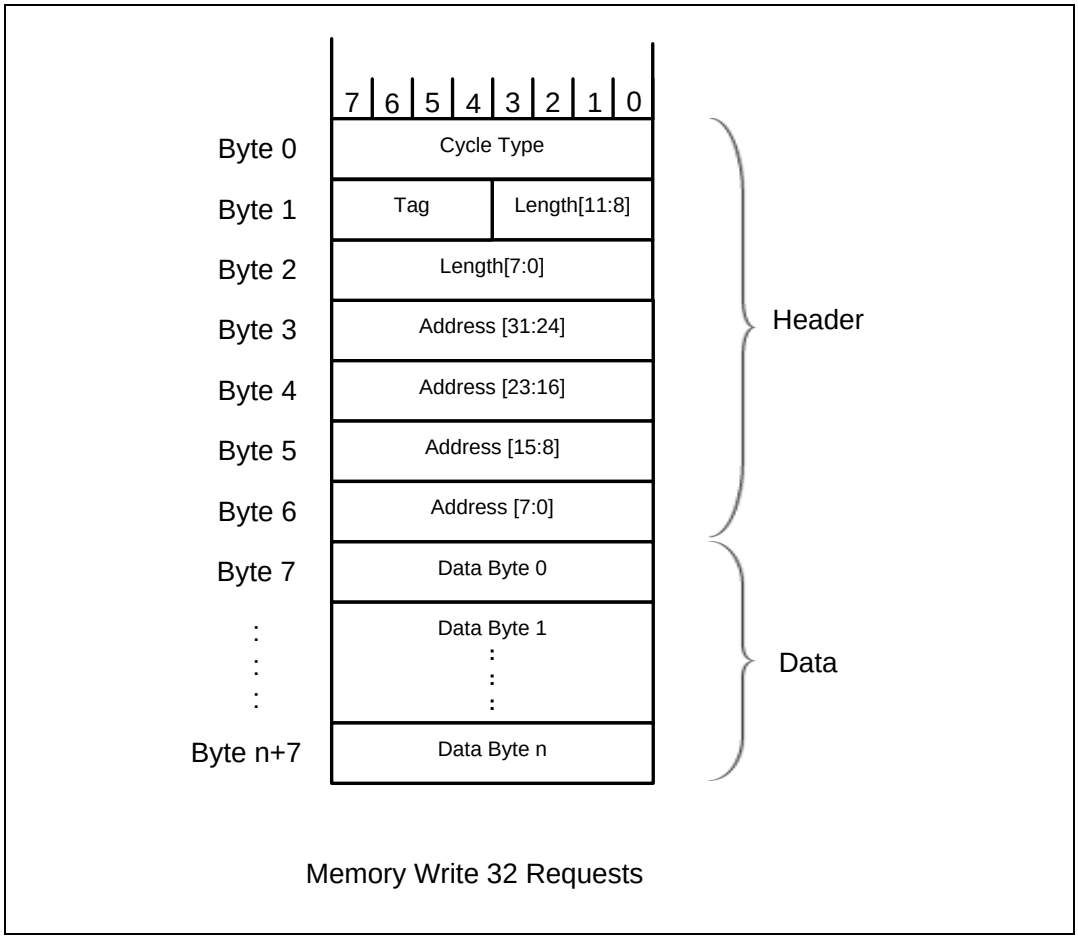
§

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 33: 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 6: 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. Refer to Figure 36 for the packet format.
Memory Read 64	00000010	Up/Down	Non-Posted	eSPI Peripheral Channel	64-bit addressing Memory Read Request. Support of upstream Memory Read 64 is mandatory for eSPI slaves that are bus mastering capable. Refer to Figure 36 for the packet format.
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. Refer to Figure 34 for the packet format.
Memory Write 64	00000011	Up/Down	Posted	eSPI Peripheral Channel	64-bit addressing Memory Write Request. Support of upstream Memory Write 64 is mandatory for eSPI slaves that are bus mastering capable. Refer to Figure 34 for the packet format.

Cycle Type	Encodings[3] [7:0]	Direction	Command Type	Channel Type	Description
Message	0001$r_2r_1r_0$0	Up/Down	Posted	eSPI Peripheral Channel	Message Request. Refer to Figure 38 for the packet format.
Message with Data	0001$r_2r_1r_0$1	Up/Down	Posted	eSPI Peripheral Channel	Message Request with data payload. Refer to Figure 38 for the packet format.
Successful Completion Without Data	00000110	Up	Completion	eSPI Peripheral Channel	Successful Completion Without Data. Corresponds to I/O Write. Refer to Figure 39 for the packet format.
Successful Completion With Data	00001P_1P_0[1]1	Up/Down	Completion	eSPI Peripheral Channel	Successful Completion With Data. Corresponds to Memory Read or I/O Read. Refer to Figure 39 for the packet format.
Unsuccessful Completion Without Data	00001P_1P_0[1,2]0	Up/Down	Completion	eSPI Peripheral Channel	Unsuccessful Completion Without Data. Corresponds to Memory or I/O. Refer to Figure 39 for the packet format.
OOB Message Channel					
OOB (Tunneled SMBus) Message	00100001	Up/Down	Posted	OOB Message Channel	SMBus Out-Of-Band Message. SMBus packet tunneling. Refer to Figure 45: OOB (Tunneled SMBus) Message Packet Format for the packet format.
Flash Access Channel[4]					
Flash Read	00000000	Up	Non-Posted	Flash Access Channel	Read from Flash. Refer to Figure 48 for the packet format.
Flash Write	00000001	Up	Non-Posted	Flash Access Channel	Write to Flash. Refer to Figure 48 for the packet format.
Flash Erase	00000010	Up	Non-Posted	Flash Access Channel	Flash Erase instruction. Erase part or the whole partition owned by the corresponding flash master. Refer to Figure 48 for the packet format.

Cycle Type	Encodings[3] [7:0]	Direction	Command Type	Channel Type	Description
Successful Completion Without Data	00000110	Down	Completion	Flash Access Channel	Successful Completion Without Data. Corresponds to Flash Write or Flash Erase. Refer to Figure 39 for the packet format.
Successful Completion With Data	00001P_1P_0[1]1	Down	Completion	Flash Access Channel	Successful Completion With Data. Corresponds to Flash Read. Refer to Figure 39 for the packet format.
Unsuccessful Completion Without Data	00001P_1P_0[1,2]0	Down	Completion	Flash Access Channel	Unsuccessful Completion Without Data. Corresponds to Flash accesses. Refer to Figure 39 for the packet format.

NOTES:

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. The message routing field $r_2r_1r_0$ has the following definition:

Encoding $r_2r_1r_0$	Description
000	Local – Terminated at receiver.
001 - 111	Reserved.

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.

The 4-bit Tag field allows up to 16 unique non-posted requests to be outstanding at any one time which have no ordering requirement among each other. However, the number of outstanding non-posted requests required to be supported by eSPI master or slave is implementation specific as it is a function of performance target outside the scope of the specification.

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 and Flash Read, the length field specifies the data payload size requested.

For memory 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.

For Short I/O and Short Memory, there is no length field defined as the length of the transaction is embedded in the command opcode itself which supports 1, 2 or 4 bytes access. Short command does not support 3 bytes access.

~~For Flash Erase, the least significant 3 bits of the length field specifies the size of the block to be erased with the encoding matches the value of the Flash Block Erase Size field of the Channel Capabilities and Configuration register~~.

For Memory Write, data payload size must not exceed the naturally aligned address boundary of the corresponding Maximum Payload Size.

For Flash Write and OOB message with data, data payload size must not exceed the Maximum Payload Size of the respective channel with no address alignment requirement. The data payload of the OOB message affected by the Maximum Payload Size is the actual payload of the protocol embedded in the message itself. Refer to Section 5.2.3 for the OOB message payload.

For Flash Erase length field definition, refer to the Master Attached Flash Sharing (MAFS), and Slave Attached Flash Sharing (SAFS) section for detail.

Read requests size initiated on the eSPI Peripheral Channel must not exceed the naturally aligned address boundary of the corresponding Maximum Read Request Size in its Channel Capability and Configuration register. For Flash Access Channel, read requests size must not exceed the corresponding Maximum Read Request Size with no address alignment requirement.

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

A Memory read or Flash read that does not exceed the Maximum Payload Size of the respective channel must be completed with a single completion.

If the read request size exceeds the Maximum Payload Size of the respective channel, the completion must be returned in multiple split completions, with each completion contains up to the Maximum Payload Size. For eSPI Peripheral Channel, each completion is aligned to the naturally aligned address boundary of the Maximum Payload Size except for the first completion which aligns to the starting address of the request. For Flash Access Channel, each completion contains up to Maximum Payload Size with no address alignment requirement.

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

For addresses below 4 GB, the memory transactions must use the 32 bits addressing format. When 64 bits addressing format is used, the upper 32 bits address [63:32] must not be all 0.

Support of upstream 64-bit addressing memory transaction is mandatory for eSPI slaves that are bus mastering capable. eSPI master must support both upstream 32-bit and 64-bit addressing memory transactions.

For Short Memory and Short I/O transactions, the 1, 2 or 4 bytes accesses must not cross the Double Word (DW) address boundary.

For other memory transactions, the address may start or end at any byte boundary. However, the address and payload length combination must not cross the naturally aligned address boundary of the corresponding Maximum Payload Size. It must not cross a 4 KB address boundary.

For Flash Access Channel and OOB messages, there is no address alignment requirement as long as the payload length does not exceed the corresponding Maximum Payload Size.

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.

Peripheral channel is reset when eSPI host bridge is reset by Platform Reset (PLTRST#). Prior to PLTRST# assertion, eSPI master and slave complete the HOST_RST_WARN and HOST_RST_ACK Virtual Wires handshake. After sending the HOST_RST_ACK, eSPI slave must not send any Peripheral channel transaction, nor any host domain Virtual Wires (i.e. Virtual Wires reset by PLTRST#) such as SMI#, SCI#, RCIN# or IRQ. Until PLTRST# is deasserted, no transaction shall occur on eSPI peripheral channel and no host domain Virtual Wires shall be sent from eSPI master or slave. eSPI peripheral channel is enabled by default after PLTRST# deassertion.

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

I/O transaction is only supported through the Short I/O command opcodes for request length of 1, 2 or 4 bytes. 3 bytes I/O transaction is not supported. I/O cycle type is not defined for the eSPI peripheral channel packet.

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

Figure 34: Peripheral Memory Write Packet Format

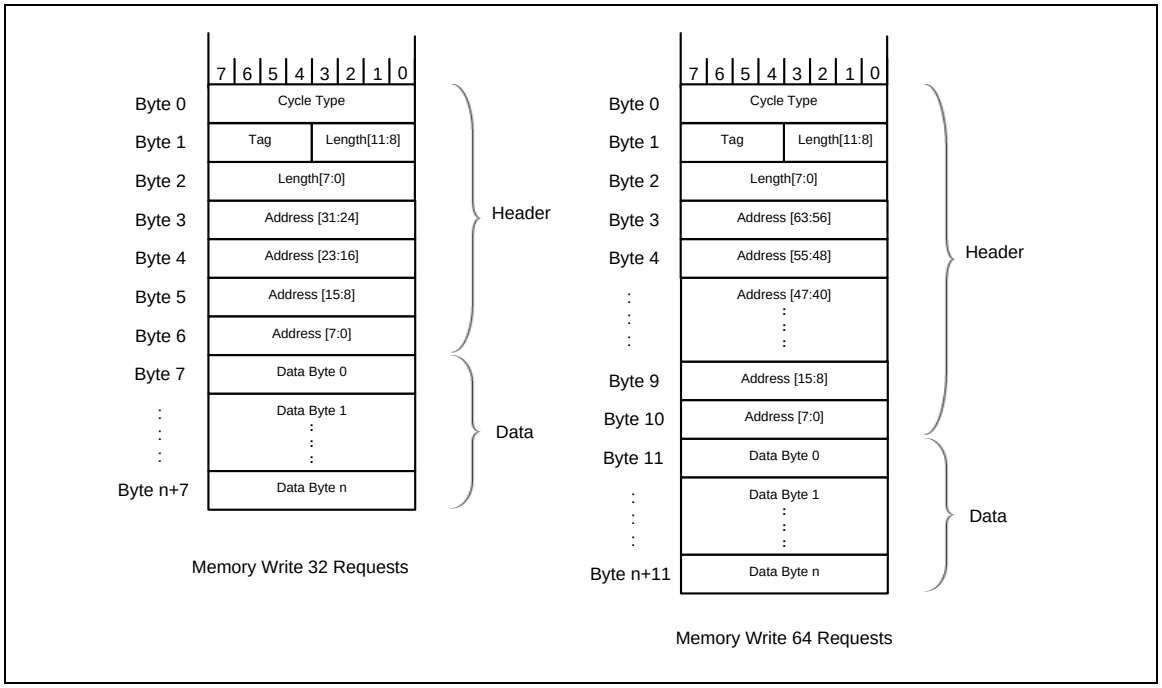


Figure 35: Short Peripheral Memory or Short I/O Write Packet Format (Master Initiated only)

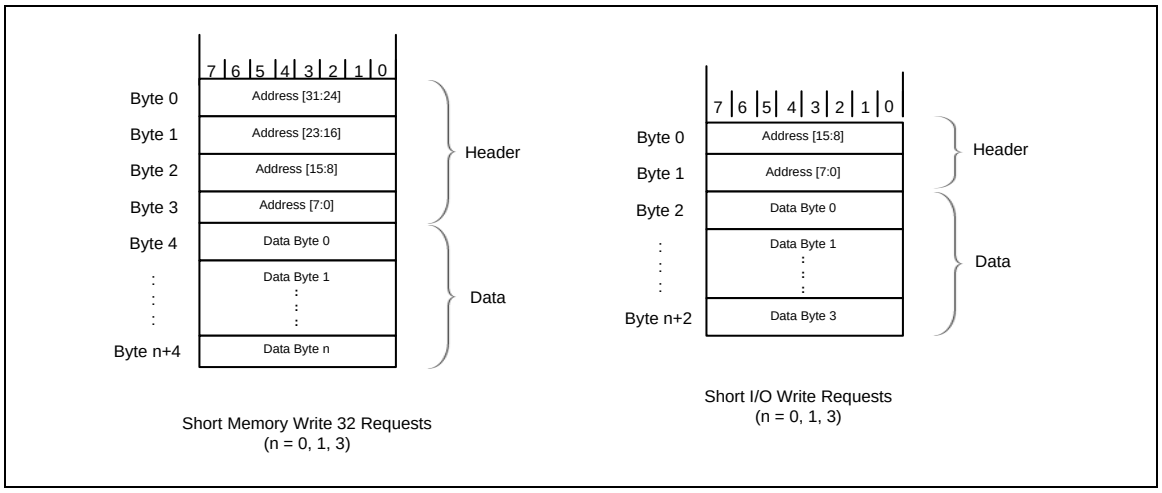


Figure 36: Peripheral Memory Read Packet Format

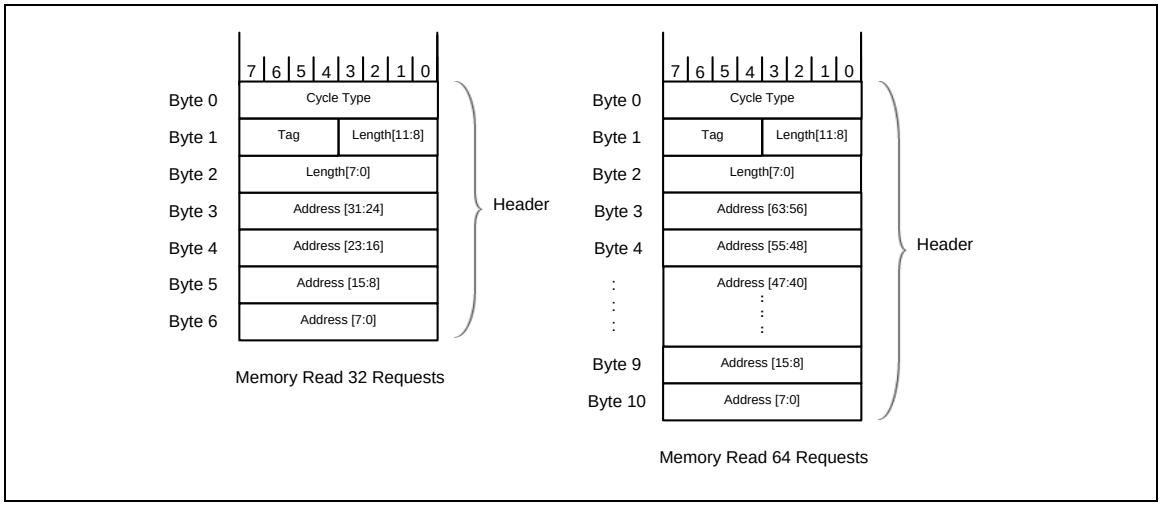


Figure 37: Short Peripheral Memory or Short I/O Read Packet Format (Master Initiated only)

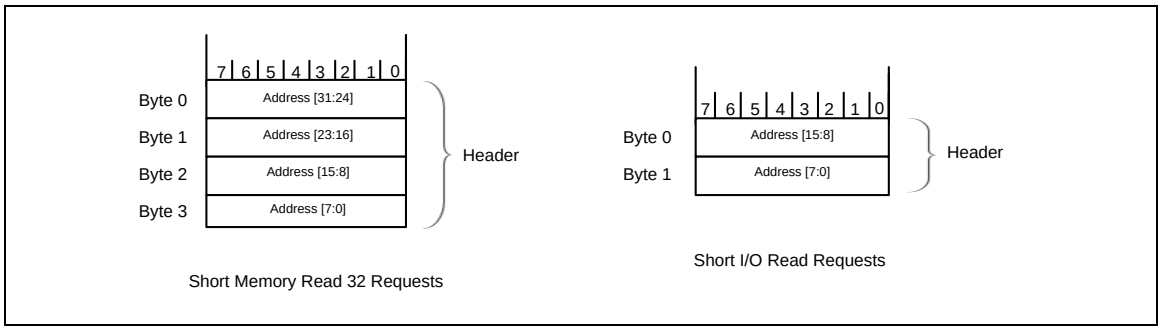


Figure 38: Peripheral Message Packet Format

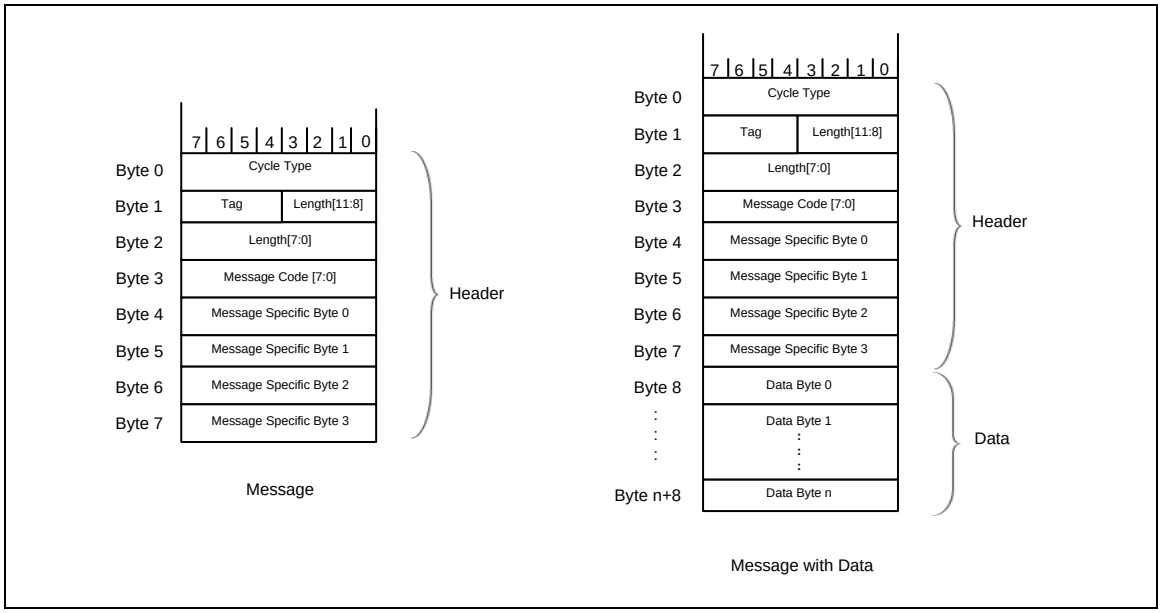
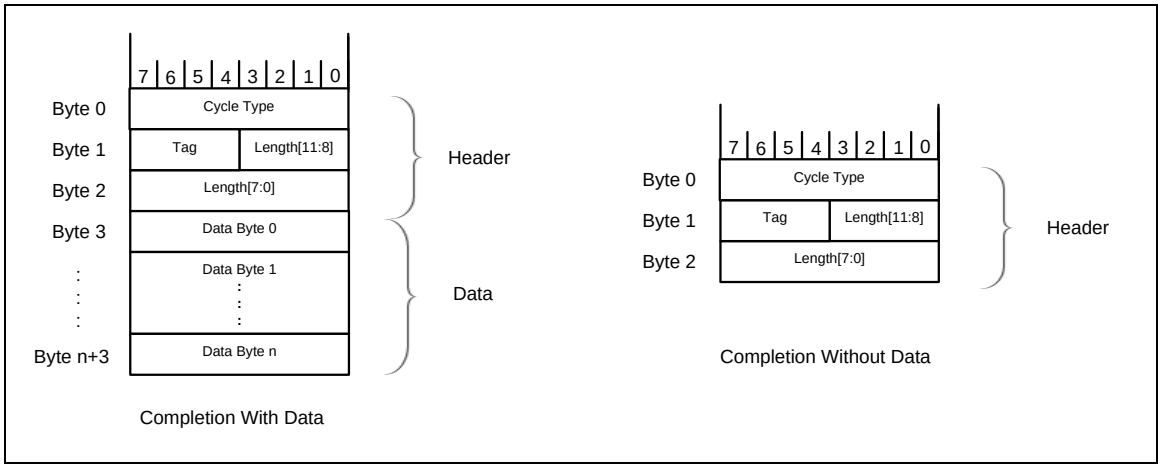


Figure 39: Peripheral Memory or I/O Completion With and Without Data Packet Format



Memory Read and Write requests can be initiated by both eSPI master and slave. Short 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.

The Message and Message with Data cycle types are posted transactions using PUT_PC and GET_PC command opcodes. Both Message and Message with Data packets use the same header format. The Length field specifies the size of the payload in the Message with Data. The 4 message specific bytes in the message header are not included in the message Length. For Message cycle type, the Length field is Reserved and it must be sent with all 0s.

The Message Code in the packet header defines the functionality and usage of the message.

Table 7: Message Codes

Name	Cycle Type	Message Code [7:0]	Routing $r_2r_1r_0$	Direction	Description
LTR	Message	0000_0001	000	Up	Latency Tolerance Reporting.

5.2.1.1 Latency Tolerance Reporting (LTR) Message

The Latency Tolerance Reporting (LTR) enables eSPI slaves to report their service latency requirements for peripheral channel upstream Memory Reads and Writes, so that power management can be implemented with consideration of eSPI slaves' service requirements.

The eSPI master is not required to honor the requested service latencies, but is strongly encouraged to provide a worst case service latency that does not exceed the latencies indicated by the LTR mechanism.

All eSPI masters must support LTR message. LTR is mandatory for eSPI slaves that support bus mastering using peripheral channel upstream memory requests.

Setting the Latency Value field to all 0's indicates that the eSPI slave will be impacted by any delay and that the best possible service is requested.

If an eSPI slave has no service requirement, it must have the Requirement bit clear. When bus mastering is disabled on an eSPI slave, if the slave had previously reported latency with the Requirement bit set, it must send a new LTR Message with Requirement bit clear. If an eSPI slave is put into an offline or equivalent of a PCI non-D0 state, the slave is required to send an LTR message with the Requirement bit clear.

An eSPI slave that supports LTR must transmit an initial LTR Message before issuing any upstream memory requests. It is recommended that eSPI slave transmits an LTR Message shortly after the peripheral channel is enabled.

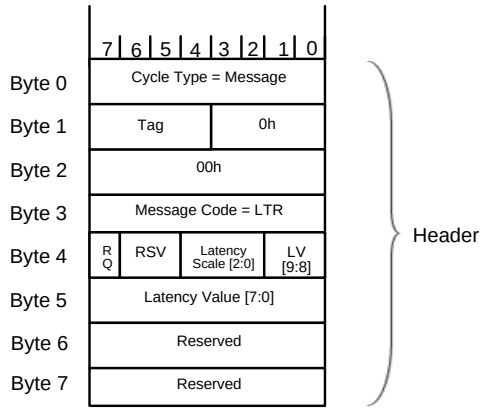
Whenever the service requirement changes, eSPI slave should transmit an updated LTR Message.

If the latency tolerance is being reduced, it is recommended to transmit the updated LTR Message ahead of first anticipated Request with the new requirement, allowing the amount of time indicated in the previously issued LTR Message. If the tolerance is being increased, then the update should immediately follow the final Request with the preceding latency tolerance value.

It is strongly recommended that eSPI slave sends no more than two LTR Messages within any 500µs time period. eSPI master must not generate an error if more than two LTR Messages are received within a 500µs time period.

LTR uses Message cycle Type with no data payload.

Figure 40: LTR Message Format



LTR Message

Table 8: LTR Message Field Description

Message Field	Description
RQ	**Requirement (RQ):** A '0' indicates that eSPI slave has no service requirement. When this bit is a '1', the remaining fields are valid to indicate latency tolerance requirement for the eSPI slave.
LS[2:0]	**Latency Scale (LS[2:0]):** This is the multiplier to the Latency Value (LV[9:0]) field to yield an absolute time value for the latency tolerance.
LV[9:0]	**Latency Value (LV[9:0]):** Along with the Latency Scale (LS[2:0]) field, this specifies the service latency that tolerable by the eSPI slave.

LS[2:0]	Description
000	Value times 1 ns
001	Value times 32 ns
010	Value times 1,024 ns
011	Value times 32,768 ns
100	Value times 1,048,576 ns
101	Value times 33,554,432 ns
110 - 111	Reserved

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 a Command Opcode, Virtual Wire Packet and a 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 slave. eSPI slave must advertise the support of 8 or more Virtual Wire groups being communicated in a single Virtual Wire packet.

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.

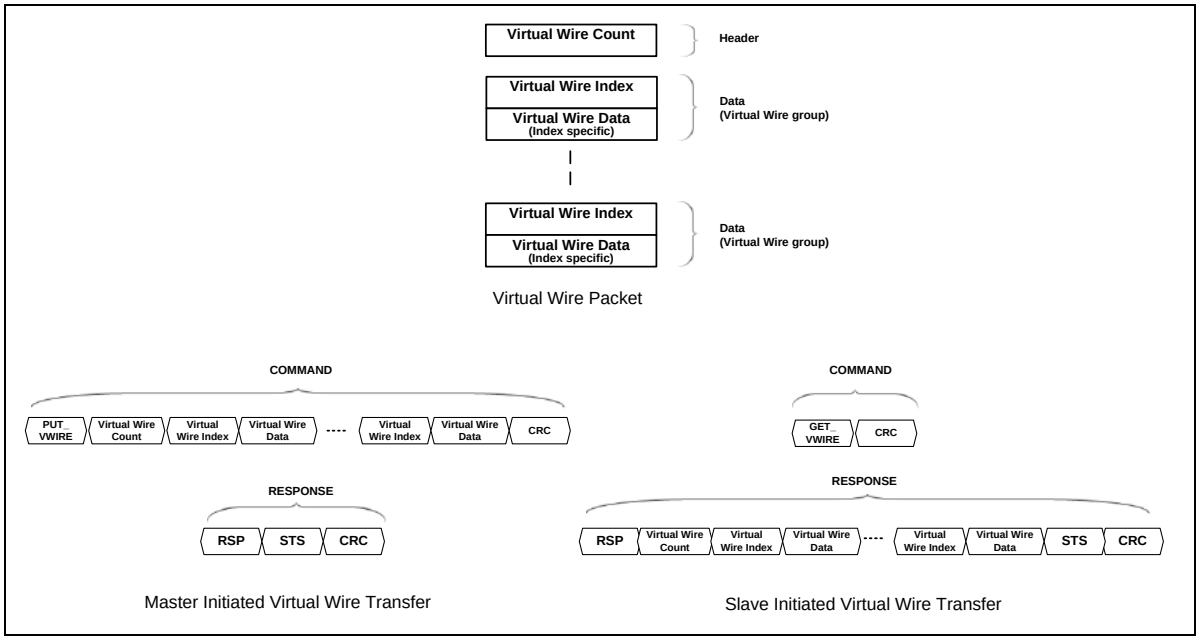
Virtual Wire packets are not subjected to flow control. The receiver must always be ready to receive the Virtual Wire packets at any time, as long as the channel is enabled.

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 41: 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.

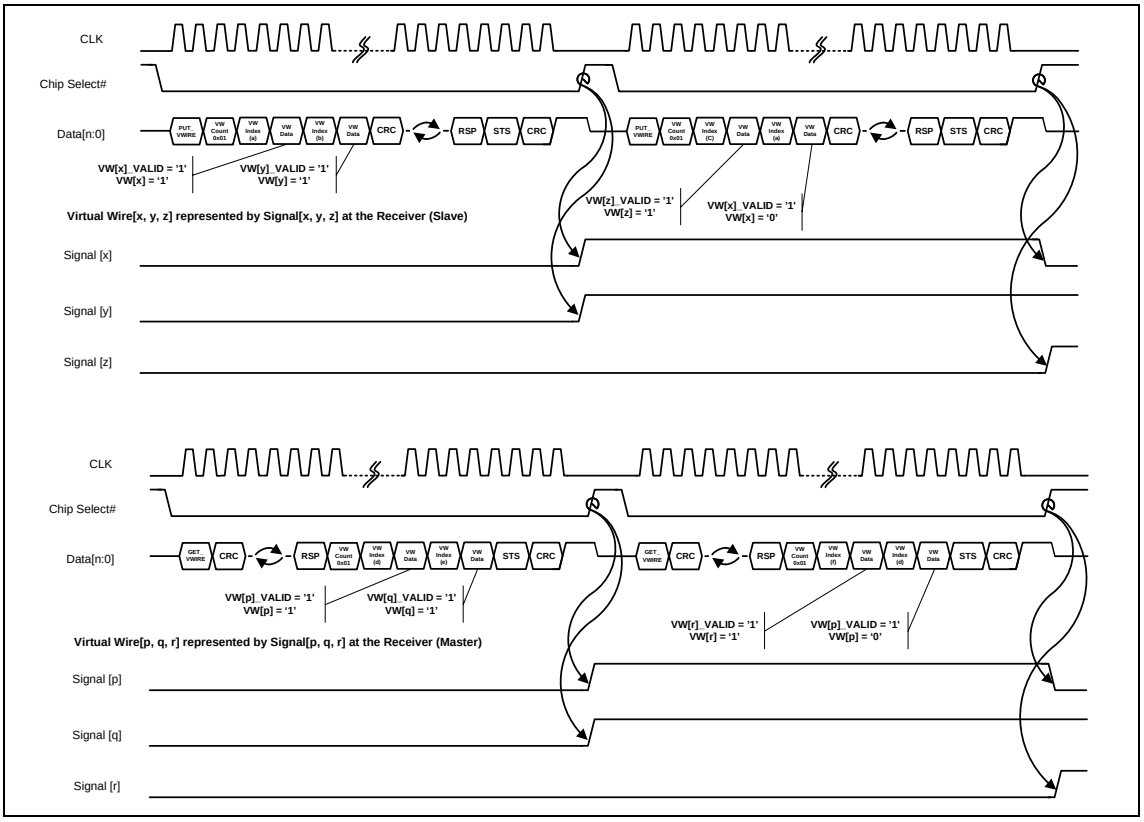
When PLTRST# message is received over the link, indicating platform reset, the individual Virtual Wires reset by Platform Reset should be initialized to the default value. To avoid behavior ambiguity, it is recommended that Virtual Wires reset by Platform Reset should not be sent together with PLTRST# Virtual Wire in a single Virtual Wire packet. In any case, PLTRST# takes priority and these Virtual Wires are reset regardless of their level sent in the packet.

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# for the purpose of synchronizing the physical pin state change with the Virtual Wires communicated.

Figure 42: 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 9: 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.
Interrupt level high (‘1’) indicates interrupt assertion. Interrupt events virtual wires are active high.
</td></tr>
</table>

<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>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
This bit functions as a “Mask” bit. When ‘0’, the corresponding virtual wire must retain its previous value and it must not be updated for this virtual wire packet.</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>
<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>
</table>

<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>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
This bit functions as a “Mask” bit. When ‘0’, the corresponding virtual wire must retain its previous value and it must not be updated for this virtual wire packet.</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.

Unless being specifically called out, all system events (Virtual Wire Index 2 to 7) are level by default. These include ERROR FATAL and ERROR NON-FATAL virtual wires which are level-triggered. Any state change on a system event results in the new state being communicated with the corresponding Level (‘0’ or ‘1’). The default reset state for the respective virtual wires is as described in the virtual wire table below.

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 10: System Event Virtual Wires for Index=2

Virtual Wire Index	**2**
Virtual Wire Group	**System Event**
Reset	**eSPI Reset# [1]**
Direction	**Master to Slave**

Notes:

1. Depending on the usage, the state of these virtual wires may need to be retained in deeper power well such that they are not reset by eSPI Reset#.

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.

Bit	Virtual Wire	Description
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 11: 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		**OOB_RST_WARN Valid:** This bit indicates the validity of OOB_RST_WARN virtual wire on bit[2]. '0' – Not valid '1' – Valid
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	OOB_RST_WARN	**OOB Reset Warn:** Sent by master just before the OOB processor is about to enter reset. Upon receiving, the EC or BMC must flush and quiesce its OOB Channel upstream request queues and assert OOB_RST_ACK VWire upon completing all the outstanding transactions. The master subsequently completes any outstanding posted transactions or completions and then disables the OOB Channel via a write to the slave's Configuration Register. Polarity: Active high. Reset: Inactive.

Bit	Virtual Wire	Description
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: Active.

Table 12: 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		Reserved
4		**OOB_RST_ACK Valid:** This bit indicates the validity of OOB_RST_ACK virtual wire on bit[0]. '0' – Not valid '1' – Valid
3	PME#	**PCI Power Management Event:** eSPI slaves generated PCI PME# event. Used by the slave to wake the host from Sx through PCI defined PME#. If the event occurs while system is in S0, a SCI is generated instead. Shared by multiple PCI devices on the platform. Polarity: Active low. Reset: Inactive.

Bit	Virtual Wire	Description
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. It is used to generate an eSPI device specific non-PME# wake. If the event occurs while system is in S0, a SCI is generated instead. Shared by multiple eSPI endpoints. Note: The eSPI WAKE# virtual wire is not equivalent to the PCIe WAKE# pin and it does not function as the PCIe WAKE#. Polarity: Active low. Reset: Inactive.
1	RSV	Reserved
0	OOB_RST_ACK	**OOB Reset Acknowledge:** Sent by slave in response to OOB_RST_WARN virtual wire. Refer to the description of OOB_RST_WARN for details. Polarity: Active high. Reset: Inactive.

Table 13: 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		**SLAVE_BOOT_LOAD_STATUS Valid:** This bit indicates the validity of SLAVE_BOOT_LOAD_STATUS virtual wire on bit[3]. '0' – Not valid '1' – Valid
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

Bit	Virtual Wire	Description
4		**SLAVE_BOOT_LOAD_DONE Valid:** This bit indicates the validity of SLAVE_BOOT_LOAD_DONE virtual wire on bit[0]. '0' – Not valid '1' – Valid
3	SLAVE_BOOT_LOAD_STATUS	**Slave Boot Load Status:** Sent by EC or BMC upon completion of Slave Boot Load from the master attached flash. '0' – The boot image is corrupted, incomplete or otherwise unusable. '1' – The boot code load was successful and that the integrity of the image is intact, or the boot code load from master attached flash is not required. Note: The Slave_Boot_Load_Status must be sent in either the same or a previous virtual wire message as the Slave_Boot_Load_Done. Polarity: As defined above. Reset: '0'.
2	ERROR_NONFATAL	**Error Non-Fatal:** Sent by slave when a non-fatal error is detected. Note: Refer to Section 9.2 for the error conditions that Non-Fatal Error virtual wire is signaled. Polarity: Active high. Reset: Inactive.
1	ERROR_FATAL	**Error Fatal:** Sent by slave when a fatal error is detected. Note: Refer to Section 9.2 for the error conditions that Fatal Error virtual wire is signaled. Polarity: Active high. Reset: Inactive.
0	SLAVE_BOOT_LOAD_DONE	**Slave Boot Load 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 14: 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		**HOST_RST_ACK Valid:** This bit indicates the validity of HOST_RST_ACK virtual wire on bit[3]. '0' – Not valid '1' – Valid
6		**RCIN# Valid:** This bit indicates the validity of RCIN# virtual wire on bit[2]. '0' – Not valid '1' – Valid
5		**SMI# Valid:** 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	HOST_RST_ACK	**Host Reset Acknowledge:** Sent by slave in response to HOST_RST_WARN virtual wire. Refer to the description of HOST_RST_WARN for details. Polarity: Active high. Reset: Inactive.
2	RCIN#	**Reset CPU INIT#:** 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.

Table 15: System Event Virtual Wires for Index=7

Virtual Wire Index	**7**
Virtual Wire Group	**System Event**
Reset	**PLTRST#**
Direction	**Master to Slave**

Bit	Virtual Wire	Description
7		Reserved
6		**NMIOUT# Valid:** This bit indicates the validity of NMIOUT# virtual wire on bit[2]. '0' – Not valid '1' – Valid
5		**SMIOUT# Valid:** This bit indicates the validity of SMIOUT# virtual wire on bit[1]. '0' – Not valid '1' – Valid
4		**HOST_RST_WARN Valid:** This bit indicates the validity of HOST_RST_WARN virtual wire on bit[0]. '0' – Not valid '1' – Valid
3	RSV	Reserved
2	NMIOUT#	**NMI Output:** Sent by master as indication that NMI# event occurs. The '0' and '1' on this virtual wire correspond to the assertion and deassertion of the NMI# to CPU respectively. Note: This virtual wire is typically only used in server platforms. Polarity: Active low. Reset: Inactive.
1	SMIOUT#	**SMI Output:** Sent by master as indication that SMI# event occurs. The '0' and '1' on this virtual wire correspond to the assertion and deassertion of the SMI# to CPU respectively. Note: This virtual wire is typically only used in server platforms. Polarity: Active low. Reset: Inactive.

Bit	Virtual Wire	Description
0	HOST_RST_WARN	**Host Reset Warn:** Sent by master just before the Host is about to enter reset. Upon receiving, the EC or BMC must flush and quiesce its upstream Peripheral Channel request queues and assert HOST_RST_ACK VWire upon completing all the outstanding transactions. The master subsequently completes any outstanding posted transactions or completions and then disables the Peripheral Channel via a write to the slave's Configuration Register. Polarity: Active high. 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.

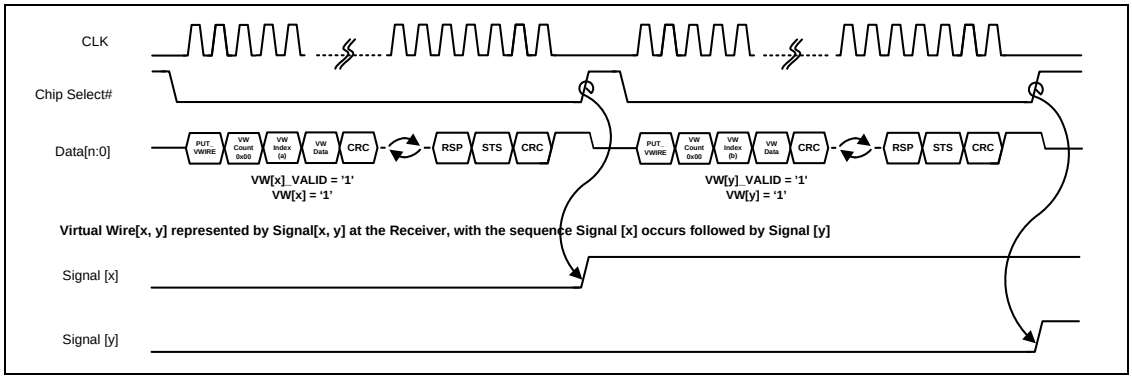
For example, the assertion of a particular pin could indicate one event. The prolonged assertion of the same pin for a certain period of time could indicate a different event.

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 different 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 43: 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.

Interrupt level high (‘1’) indicates interrupt assertion whereas interrupt level low (‘0’) indicates interrupt deassertion. Interrupt events virtual wires are active high.

For simplicity, eSPI interrupt event virtual wires are default deasserted (level low) thus eliminating the legacy SERIRQ behavior where SERIRQ line is default pulled high (‘1’) even though there is no interrupt expected to be generated.

Table 16: Interrupt Event (IRQ) Virtual Wire Generation

IRQ Source (In Slave)	Source Level	IRQ Enable (0=Disable, 1=Enable)	Slave to Master IRQ Virtual Wire (Active High)
Active High	0	0	Default. No IRQ VW sent
	0	0 -> 1 1 -> 0	No IRQ VW sent
	0 -> 1	1	Assertion. IRQ VW (Level=’1’) sent
	1 -> 0	1	Deassertion. IRQ VW (Level=’0’) sent
	1	0 -> 1	Assertion. IRQ VW (Level=’1’) sent
	1	1 -> 0	Deassertion. IRQ VW (Level=’0’) sent
	0 -> 1 1 -> 0	0	No IRQ VW sent
Active Low	1	0	Default. No IRQ VW sent
	1	0 -> 1 1 -> 0	No IRQ VW sent
	0 -> 1	1	Deassertion. IRQ VW (Level=’0’) sent
	1 -> 0	1	Assertion. IRQ VW (Level=’1’) sent
	0	0 -> 1	Assertion. IRQ VW (Level=’1’) sent
	0	1 -> 0	Deassertion. IRQ VW (Level=’0’) sent
	0 -> 1 1 -> 0	0	No IRQ VW sent

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.

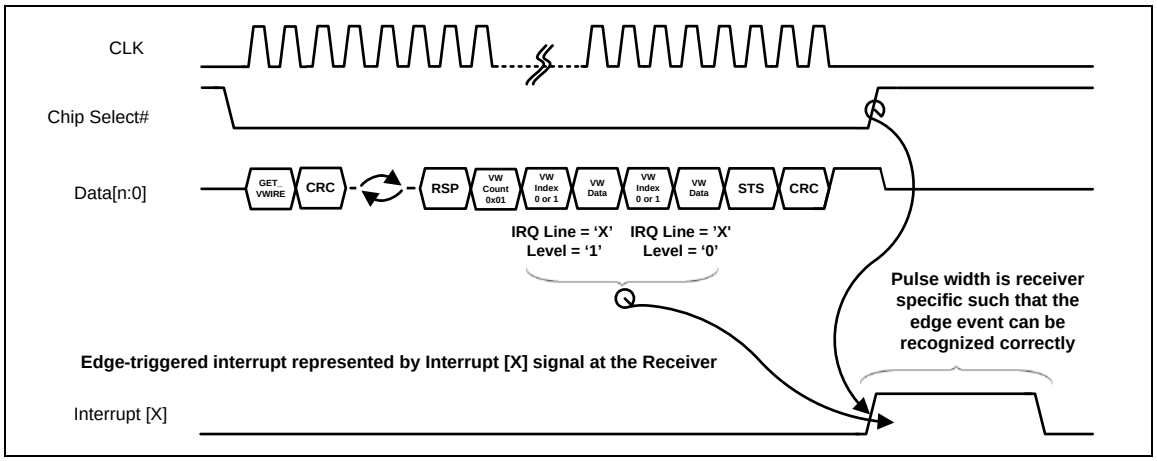
Each of the virtual wire including interrupt event virtual wire must not have more than 2 transitions being communicated within a Virtual Wire packet, otherwise the corresponding virtual wire behavior is undefined. The initiator must make sure this is not violated. It is optional for the receiver to check for this violation. The 2 transitions may be a ‘0’-to-‘1’-to-‘0’ or ‘1’-to-‘0’-to-‘1’ transition. 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.

It is possible for eSPI slave to communicate edge-triggered interrupt using two separate Virtual Wire packets, i.e. interrupt assertion in one Virtual Wire packet followed by interrupt deassertion in a subsequent Virtual Wire packet. However, it is the responsibility of the slave to ensure that interrupt assertion edge is always communicated without being lost, such as in the boundary where the next interrupt event happens before the prior interrupt deassertion virtual wire is able to be sent out.

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 44: 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. These include the SMBus Management Component Transport Protocol (MCTP) packets which are based on the SMBus block write protocol.

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

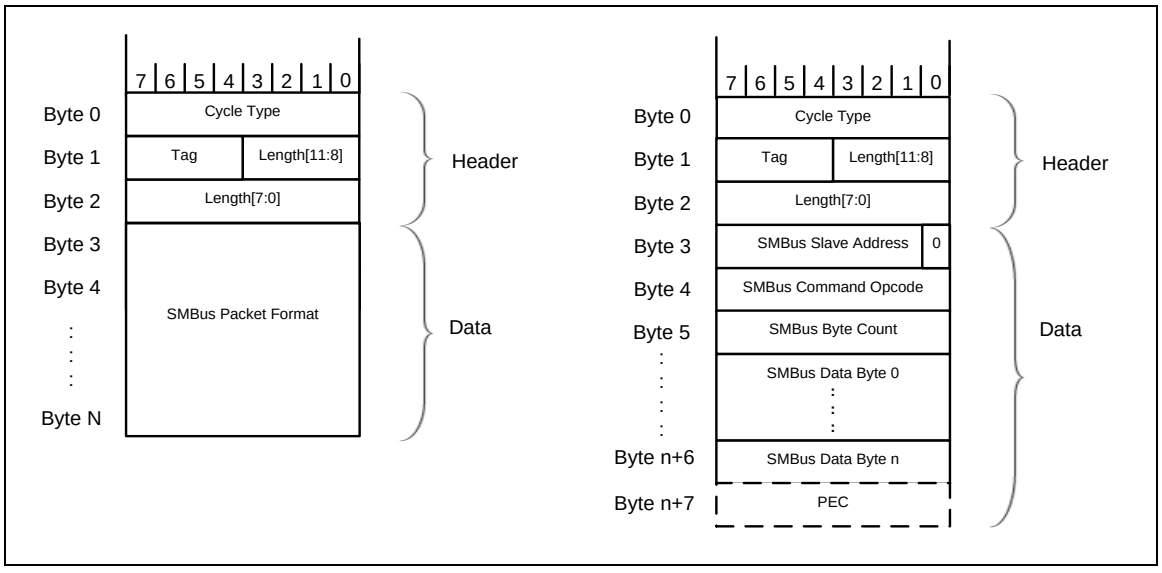
The SMBus Byte Count field does not include the PEC byte. It comprehends the actual payload of the SMBus block write packet itself excluding the 3 SMBus header bytes.

The Length field of the OOB message comprehends the count by the SMBus Byte Count field, in addition to the 3 header bytes (i.e. SMBus Slave Address, SMBus Command Opcode and SMBus Byte Count) and an optional PEC byte.

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

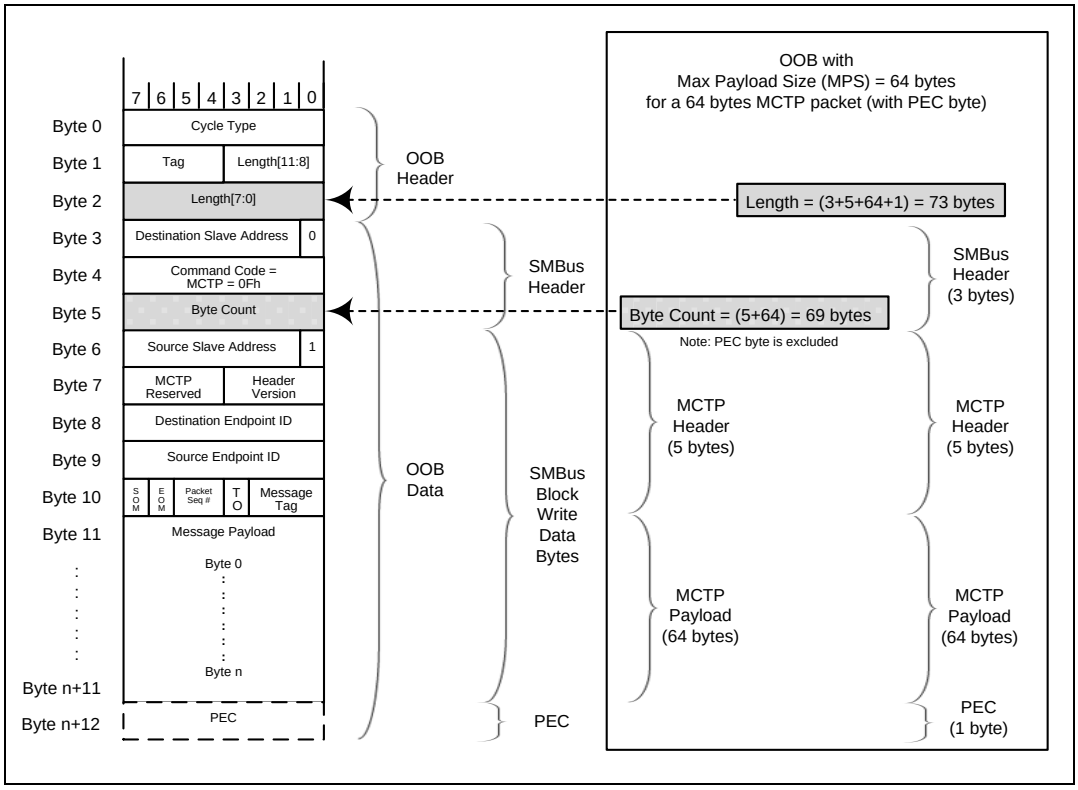
The Maximum Payload Size (MPS) for OOB Message channel applies to the actual payload of the protocol embedded in the packet that tunneled through the channel, such as but not limited to the MCTP and the generic SMBus block writes.

Figure 45: OOB (Tunneled SMBus) Message Packet Format



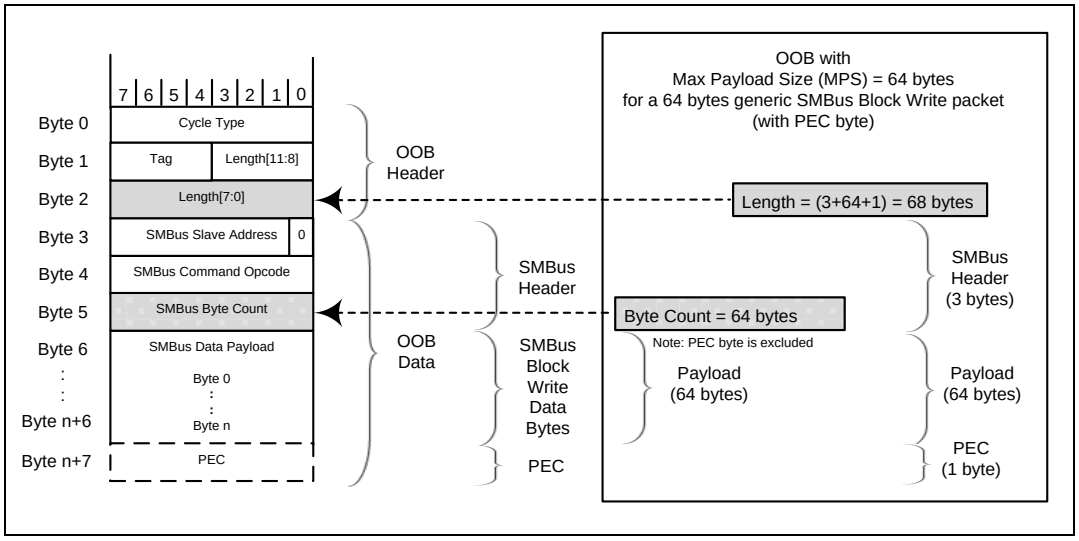
MCTP over SMBus is a specific form of the SMBus block write packet with the SMBus Command Opcode of 0Fh (i.e. MCTP). The MCTP header and MCTP payload are embedded as the SMBus block write data bytes. For eSPI OOB MCTP packet, the Maximum Payload Size (MPS) applies to the MCTP payload itself excluding the MCTP header and the optional PEC byte. For example, MPS of 64 bytes allows the transfer of a MCTP packet with up to 64 bytes MCTP payload over the OOB Message channel. In the case of 64 bytes MCTP payload with the optional PEC byte, the SMBus byte count field and the OOB header length field are 69 bytes and 73 bytes respectively.

Figure 46: OOB MCTP Packet



For eSPI OOB generic SMBus block write packet, the Maximum Payload Size (MPS) applies to the number of SMBus block write data bytes allowed in a packet excluding the optional PEC byte. For example, MPS of 64 bytes allows the transfer of a generic SMBus block write packet with up to 64 bytes data payload over the OOB Message channel. In the case of 64 bytes data payload with the optional PEC byte, the SMBus byte count field and the OOB header length field are 64 bytes and 68 bytes respectively.

Figure 47: OOB Generic SMBus Block Write 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 48: Flash Access Request Packet Format

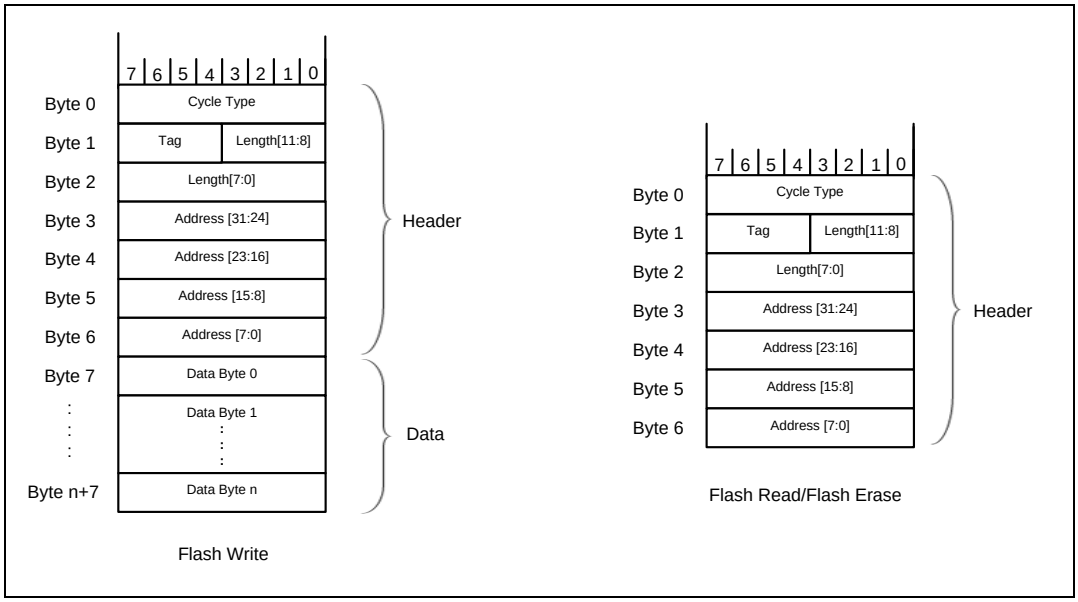
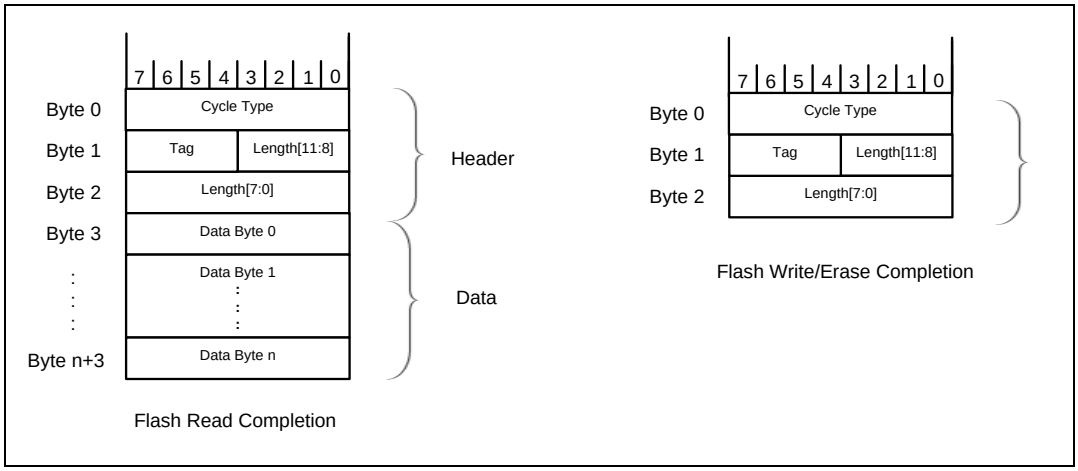


Figure 49: Flash Access Completion Packet Format



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. However, length field encoding of "011" is not applicable for Flash Erase in Master Attached Flash Sharing (MAFS).

Figure 50: Independent Flash SPI and eSPI Interface

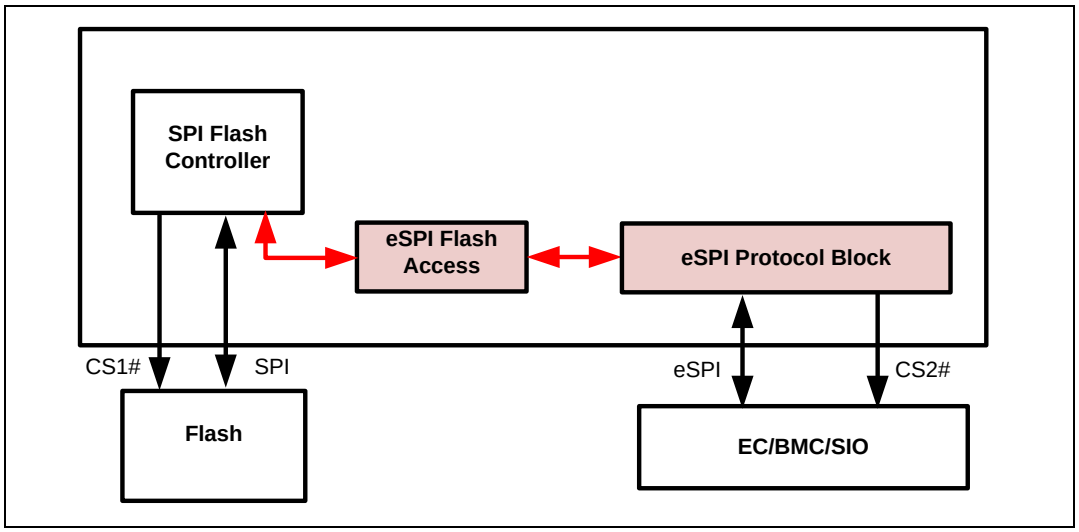
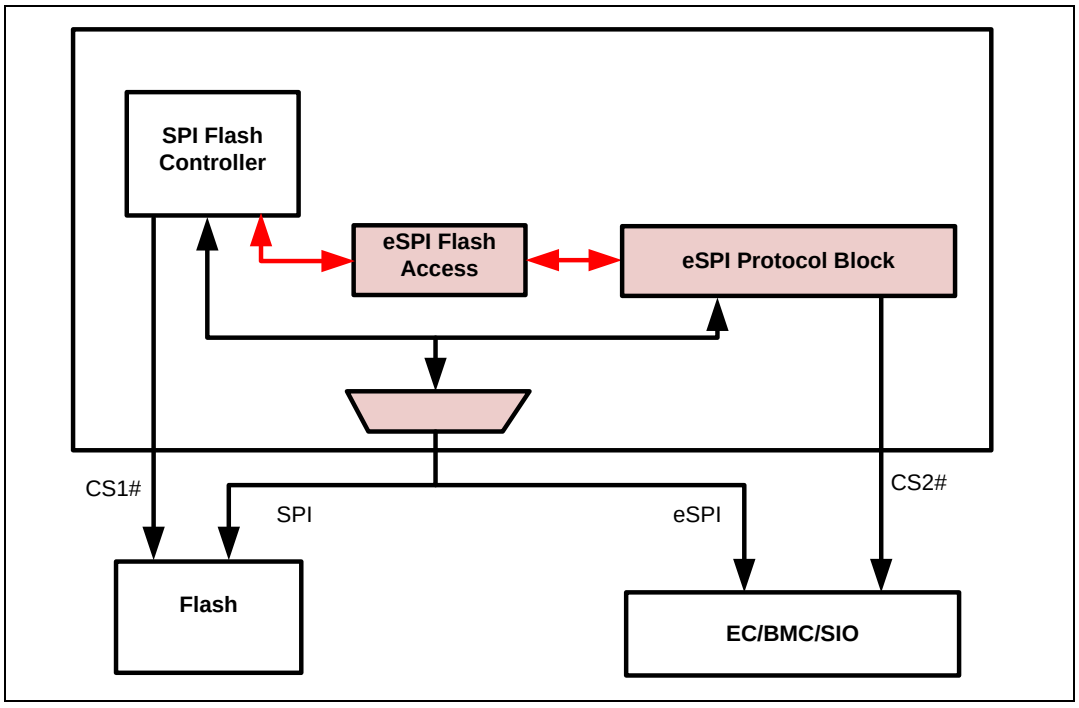


Figure 51: Shared SPI and eSPI Interface



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 3).

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 physical Flash Linear Addresses. The physical addresses cover the entire flash addressing space. However, the SPI Flash controller may impose access permission control for Flash Access transactions initiated by the eSPI slaves. The detail of the access permission control is outside the scope of the specification.

Any attempt to access a flash region without the access permission is considered an error. The SPI Flash controller is required to 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 3 bits of the length field specifies the size of the block to be erased. The encoding of the least significant 3 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 the eSPI base specification.

The detail of the Slave Attached Flash Sharing is described in the eSPI addendum for server platforms.

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.

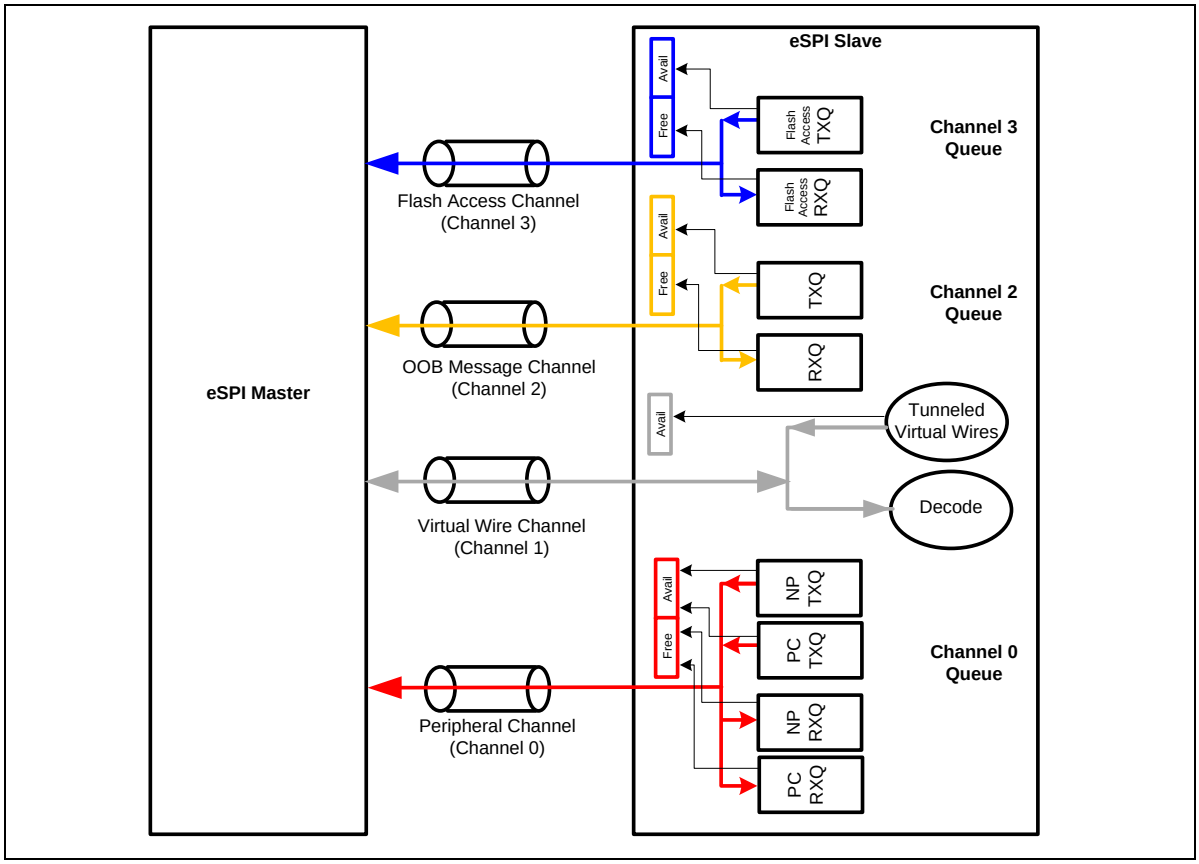
The eSPI slave is responsible to ensure ordering prior to presenting the request to the eSPI master. A request should not be seen by the eSPI master through the Status field until it has met the ordering requirement with respect to other pending requests. For example, if a non-posted read at the top 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.

The _AVAIL or _FREE, once asserted, must continue to be committed by eSPI slave until the corresponding action is taken by master to the associated slave's buffer. For example, PC_AVAIL once asserted by slave must only be affected by GET_PC command from master, PC_FREE once asserted by slave must only be affected by PUT_PC command from master and so on. Once asserted, the slave is not allowed to change the _AVAIL or _FREE indication due to other unrelated events.

Figure 52: 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 or Completion	Non-Posted Request
Posted Request or Completion	No[1]	Yes[3]
Non-Posted Request	No[2]	No[4]

No - indicates that the subsequent transaction is not allowed to complete before the previous transaction to preserve ordering in the system.

Yes - indicates that the subsequent transaction must be allowed to complete before the previous one or a deadlock can occur.

NOTES:

1. Posted request must not pass posted request to ensure the most updated data is written last. Completion must pull posted write data back to the originating bus to avoid stale data.
2. Non-posted request must push posted write data to avoid reading stale data.
3. Posted request or completion must be allowed to pass non-posted request to avoid deadlocks.
4. Non-posted requests are not required to pass each other.

The transaction ordering rule requires eSPI master and eSPI slave to pre-allocate completion buffer for every non-posted transactions they initiated. This ensures that completion returned will not block the forward progress of any posted transactions behind.

To avoid possible deadlock, there must be no in-out dependency between Transmit (Tx) and Receive (Rx), specifically an eSPI agent cannot make freeing up of the Rx queue for a channel dependent on the forward progress of the corresponding Tx queue. This applies to all the corresponding Tx/Rx pairs for the peripheral channel, virtual wire channel, OOB channel and flash access channel.

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 Single I/O mode, I/O[1:0] pins are uni-directional. eSPI master drives the I/O[0] during command phase, and response from slave is driven on the I/O[1]. eSPI slave is required to tri-state I/O[1] during command phase as I/O[1] can be driven by eSPI master such as when initiating an In-Band Reset command.

In Dual I/O mode, 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 of the Single I/O mode.

In Quad I/O mode, I/O[3:0] pins are bi-directional data bus and all the command and response phases are transferred over the four bi-directional pins at the same time, effectively doubling the transfer rate of the Dual I/O mode.

Each of the fields for an eSPI transaction is shifted out accordingly in the defined order. For fields that contain multiple bytes, the order of the bytes being shifted out on the eSPI bus is as follow: (LSB = Least Significant Byte, MSB = Most Significant Byte)

- Header:
    - Length: From MSB (with Tag field) to LSB
    - Address: From MSB to LSB. This applies to eSPI transactions with address including GET_CONFIGURATION and SET_CONFIGURATION.
- Data: From LSB to MSB
- Status: From LSB to MSB

Each of the bytes is shifted from the most significant bit (bit[7]) to the least significant bit (bit[0]).

An example of a master initiated peripheral channel memory read is as shown below.

Figure 53: Byte Ordering on the eSPI Bus

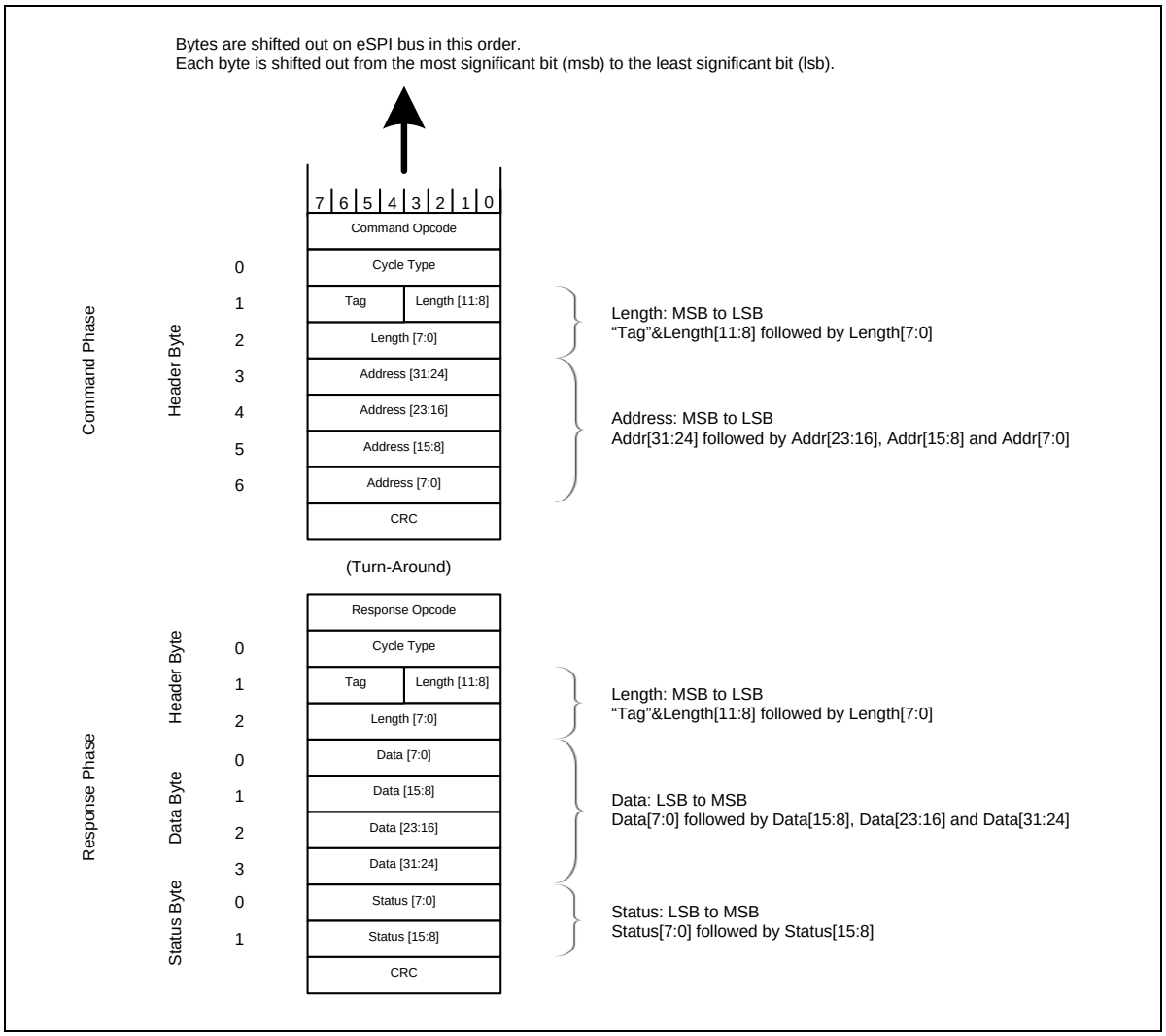


Figure 54: Single I/O Mode

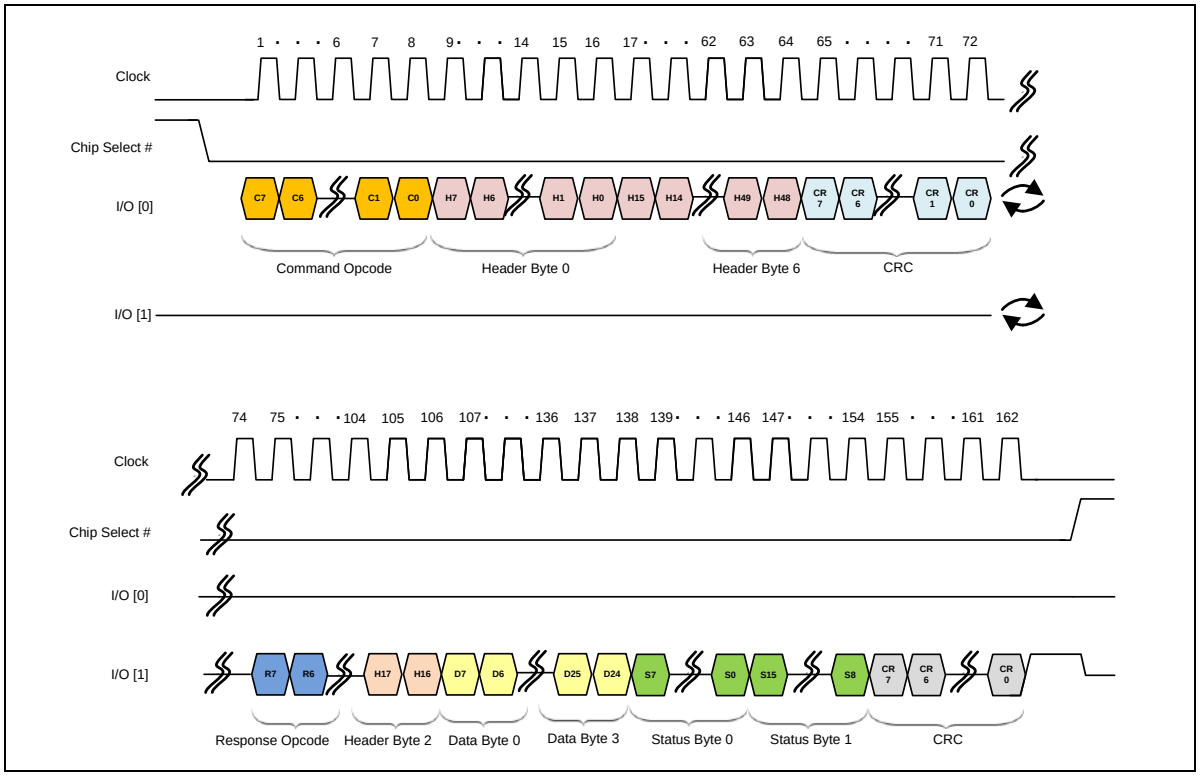


Figure 55: Dual I/O Mode

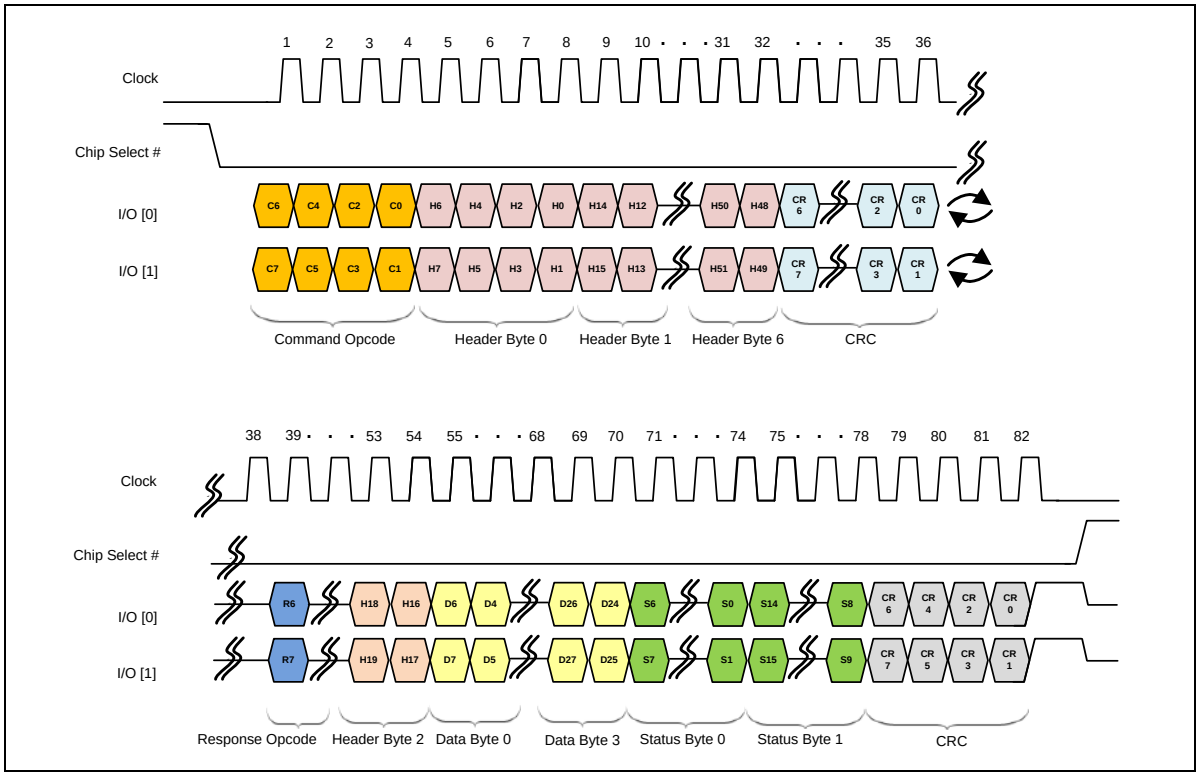
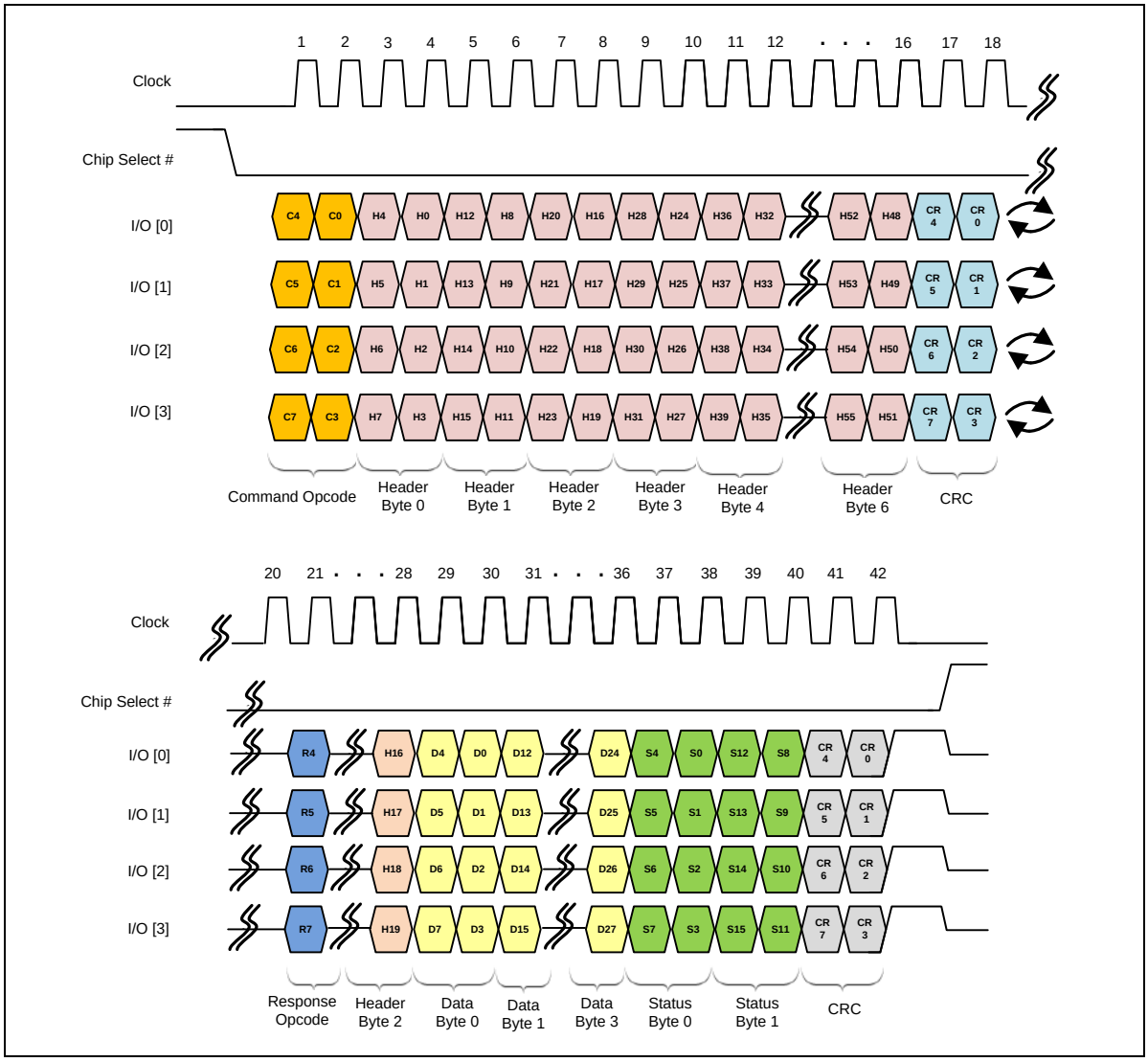


Figure 56: Quad I/O Mode



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 (except WAIT_STATE Response Code which is not included in the CRC calculation), 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 generation is mandatory for eSPI. However, CRC checking is default disabled after eSPI reset# and it is enabled through SET_CONFIGURATION by setting the CRC Checking Enable bit whereby upon the successful SET_CONFIGURATION, CRC checking is enabled on both eSPI master and eSPI slave at the deassertion edge of CS#. Both eSPI master and eSPI slave must always be capable of supporting CRC checking as platform requirements determine if CRC checking will be enabled for an eSPI interface.

When CRC checking is disabled, the CRC byte is ignored by the receiver.

Figure 57: CRC Polynomial Representation

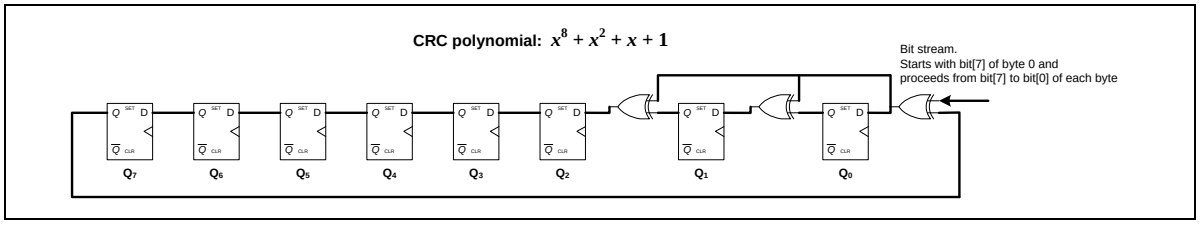


Table 17: CRC Byte with Input Data D7:D0 (⊕ denotes logical XOR)

	1st Clock	2nd Clock	3rd Clock	4th Clock
Q_0	D7	D6	D5	D4
Q_1	D7	D7⊕D6	D6⊕D5	D5⊕D4
Q_2	D7	D7⊕D6	D7⊕D6⊕D5	D6⊕D5⊕D4
Q_3	0	D7	D7⊕D6	D7⊕D6⊕D5
Q_4	0	0	D7	D7⊕D6
Q_5	0	0	0	D7
Q_6	0	0	0	0
Q_7	0	0	0	0

	5th Clock	6th Clock	7th Clock	8th Clock. CRC = $Q_{7:0}$
Q_0	D3	D2	D7⊕D1	D7⊕D6⊕D0
Q_1	D4⊕D3	D3⊕D2	D7⊕D2⊕D1	D6⊕D1⊕D0
Q_2	D5⊕D4⊕D3	D4⊕D3⊕D2	D7⊕D3⊕D2⊕D1	D6⊕D2⊕D1⊕D0
Q_3	D6⊕D5⊕D4	D5⊕D4⊕D3	D4⊕D3⊕D2	D7⊕D3⊕D2⊕D1
Q_4	D7⊕D6⊕D5	D6⊕D5⊕D4	D5⊕D4⊕D3	D4⊕D3⊕D2
Q_5	D7⊕D6	D7⊕D6⊕D5	D6⊕D5⊕D4	D5⊕D4⊕D3
Q_6	D7	D7⊕D6	D7⊕D6⊕D5	D6⊕D5⊕D4
Q_7	0	D7	D7⊕D6	D7⊕D6⊕D5

§

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 18: Register Attribute Description

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

Table 19: 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#. In addition, the register may be reset by additional reset as described in the respective register section.

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 20: 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

This register is also reset by the In-band RESET command.

Bit	Type	Default	Description
31	RW	0b	**CRC Checking Enable:** This bit is set to ‘1’ by eSPI master to enable the CRC checking on the eSPI bus. By default, CRC checking is disabled. 0b: CRC checking is disabled. 1b: CRC checking is enabled.
30	RW	0b	**Response Modifier Enable:** This bit is set to ‘1’ to enable the use of Response Modifier by eSPI slave to append either a peripheral (channel 0) completion, a virtual wire (channel 1) packet or a flash access (channel 3) completion to the GET_STATUS response phase. When this bit is a ‘0’, eSPI slave must only use the Response Modifier of “00”, i.e. no append. By default, the Response Modifier is disabled.
29	RO	0	**Reserved.**

<table>
<tr><th>Bit</th><th>Type</th><th>Default</th><th>Description</th></tr>
<tr><td>28</td><td>RW</td><td>0b</td><td>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: Alert# pin is used to signal the Alert event.

Note: This bit can only be ‘0’ or ‘1’ in a single master-single slave topology. For single master-multiple slave topology, this bit must be programmed to ‘1’.

Alert Mode is allowed to change from default ‘0’ to ‘1’ during runtime in both single or multiple slaves topologies provided when this happens, only a single slave is enabled for generating Alert# event.</td></tr>
<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>
</table>

<table>
<tr><th>Bit</th><th>Type</th><th>Default</th><th>Description</th></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>RW</td><td>0</td><td>Open Drain Alert# Select: This bit is set to ‘1’ by eSPI master to configure the Alert# pin as an open-drain output.
By default, Alert# pin operates as a driven output. This bit must only be programmed to ‘1’ if open-drain Alert# pin is supported by the slave.
The bit must be valid when Alert Mode bit is a ‘1’ indicating Alert# pin is used for signaling the Alert event.
0b: Alert# pin is a driven output.
1b: Alert# pin is an open-drain output.</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 22) of 20MHz max.</td></tr>
<tr><td>19</td><td>RO</td><td>HwInit</td><td>Open Drain Alert# Supported: This bit indicates the support of the Alert# pin as an open-drain output by the slave.

0b: Open-drain Alert# pin is not supported.
1b: Open-drain Alert# pin is supported.</td></tr>
</table>

<table>
<tr><th>Bit</th><th>Type</th><th>Default</th><th>Description</th></tr>
<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 22) is mandatory.</td></tr>
<tr><td>15:12</td><td>RW</td><td>0</td><td>Maximum WAIT STATE Allowed: eSPI master sets the maximum WAIT STATE allowed to be responded by slave before the slave must respond with an ACCEPT, DEFER, NON-FATAL ERROR or FATAL ERROR response code.
This is a 1-based field in the granularity of byte time. When “0”, it indicates a value of 16 byte time.
A byte time corresponds to 8 serial clocks in the Single I/O mode, 4 serial clocks in the Dual I/O mode or 2 serial clocks in the Quad I/O mode.</td></tr>
<tr><td>11:8</td><td>RO</td><td>0</td><td>Reserved.</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

This register is also reset by the Platform Reset (PLTRST#).

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 length of the read request must not cross the naturally aligned address boundary of the corresponding Maximum Read Request 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.**

Bit	Type	Default	Description
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. The payload of the transaction must not cross the naturally aligned address boundary of the corresponding Maximum Payload Size. 000b: Reserved. 001b: 64 bytes address aligned max payload size. 010b: 128 bytes address aligned max payload size. 011b: 256 bytes address aligned 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 address aligned max payload size. 010b: 128 bytes address aligned max payload size. 011b: 256 bytes address aligned max payload size. 100b – 111b: Reserved.
3	RO	0	**Reserved.**
2	RW	0b	**Bus Master Enable:** When this bit is a '0', it disables the slave from generating bus mastering cycles on the Peripheral channel. When this bit is a '1', it allows the slave to generate bus mastering cycles on the Peripheral channel. Prior to clearing the Bus Master Enable bit from '1' to '0', there must be no outstanding non-posted cycle pending completion from the slave.

Bit	Type	Default	Description
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	1b	**Peripheral Channel Enable:** The channel is by default enabled after the eSPI Reset#. This bit is cleared to '0' by eSPI master to disable the Peripheral channel. Besides, clearing this bit from '1' to '0' triggers a reset to the Peripheral channel. The channel remains disabled until this bit is set to '1' again. Prior to disabling the Peripheral channel, the Bus Master Enable bit should be cleared to '0' to disable the bus mastering cycles.

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 maximum number of Virtual Wire groups being communicated in a single Virtual Wire packet. eSPI slave must advertise a value of “000111b” or more in this field to indicate the support of at least 8 Virtual Wire groups being 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.

Bit	Type	Default	Description
0	RW	0b	**Virtual Wire Channel Enable:** This bit is set to '1' by eSPI master to enable the Virtual Wire channel. Clearing this bit from '1' to '0' will not reset the Virtual Wire channel whereby the state of all the Virtual Wires must continue to be maintained internally. When this bit is '0', no transaction shall occur on 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. The Maximum Payload Size applies to the actual payload of the protocol embedded in the OOB packet. Refer to Section 5.2.3 for the detail of the OOB message payload. 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.**

Bit	Type	Default	Description
6:4	RO	HwInit	**OOB Message Channel Maximum Payload Size Supported:** This field advertises the Maximum Payload Size supported by the slave. The Maximum Payload Size applies to the actual payload of the protocol embedded in the OOB packet. Refer to Section 5.2.3 for the detail of the OOB message payload. 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. Clearing this bit from '1' to '0' triggers a reset to the OOB Message channel such as during error handling. The channel remains disabled until this bit is set to '1' again. The channel is by default disabled after the eSPI Reset#.

7.2.1.7 Offset 40h: Channel 3 Capabilities and Configurations

Bit	Type	Default	Description
31:15	RO	0b	**Reserved.**

Bit	Type	Default	Description
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 length of the read request must not exceed the corresponding Maximum Read Request Size with no address alignment requirement. 000b: Reserved. 001b: 64 bytes max read request size. 010b: 128 bytes max read request size. 011b: 256 bytes max read request size. 100b: 512 bytes max read request size. 101b: 1024 bytes max read request size. 110b: 2048 bytes max read request size. 111b: 4096 bytes max read request size.
11	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: Slave attached flash sharing. This bit is a Read-Only '0' in the base specification as only master attached flash sharing is defined.
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.

Bit	Type	Default	Description
7:5	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.
4:2	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. 000b: Reserved 001b: 4 Kbytes 010b: 64 Kbytes 011b: Both 4 Kbytes and 64 Kbytes are supported 100b: 128 Kbytes 101b: 256 Kbytes 110b – 111b: Reserved
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. Clearing this bit from '1' to '0' triggers a reset to the Flash Access channel such as during error handling. The channel remains disabled until this bit is set to '1' again. 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 21: Electrical Specification

Symbol	Parameter	Condition	Min	Typ	Max	Unit
Vcc	eSPI I/O voltage		1.71	1.8	1.89	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
$R^{1}_{reset\text{-}PU}$	Weak pull-up impedance	V_{out} = 0.7*Vcc	10k		30k	Ohm
$R^{1}_{reset\text{-}PD}$	Weak pull-down impedance	V_{out} = 0.3*Vcc	10k		30k	Ohm
$R_{alert\text{-}PU}$	Weak pull-up impedance for Alert# pin	V_{out} = 0.7*Vcc		4.7k[3]		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

NOTES:

1. Weak pull-up on eSPI data and Chip Select# pins (except Alert# pin) and weak pull-down on eSPI clock 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.

3. The weak pull-up impedance value is defined for a typical eSPI bus loading when Alert# pin is configured as open-drain. Platform is required to adjust this value accordingly such that when Alert# pin is asserted, the assertion of the CS# for the shortest possible transaction (which causes the slave to tri-state the Alert# pin), is able to pull the Alert# pin high fast enough to the deasserted value before or by the last failing edge of the serial clock at the end of the transaction.

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 22: AC Timing Specification

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

Symbol	20 MHz		25 MHz		33 MHz		50 MHz		66 MHz		
	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}	75		60		45		30		22		ns
t_{CLSH}	50		40		30		20		15		ns
t_{SHSL}	50		40		30		20		15		ns
t_{DVCH}	12		10		7		5		3		ns

Symbol	20 MHz		25 MHz		33 MHz		50 MHz		66 MHz		
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Unit
t_{CHDX}	12		10		7		5		3		ns
t_{CLQZ}		15		12		9		8		6	ns
t_{CLQV}		20		15		10		8		6	ns
t_{CLQX}	0		0		0		0		0		ns
t_{SHQZ}		15		12		9		8		6	ns
t_{SLAZ}		15		12		9		8		6	ns
t_{SHAA}	15		12		9		8		6		ns
t_{INIT}	1		1		1		1		1		us
$t_{INIT-FREQ}$		20		20		20		20		20	MHz

Figure 58: Input Timing Diagram

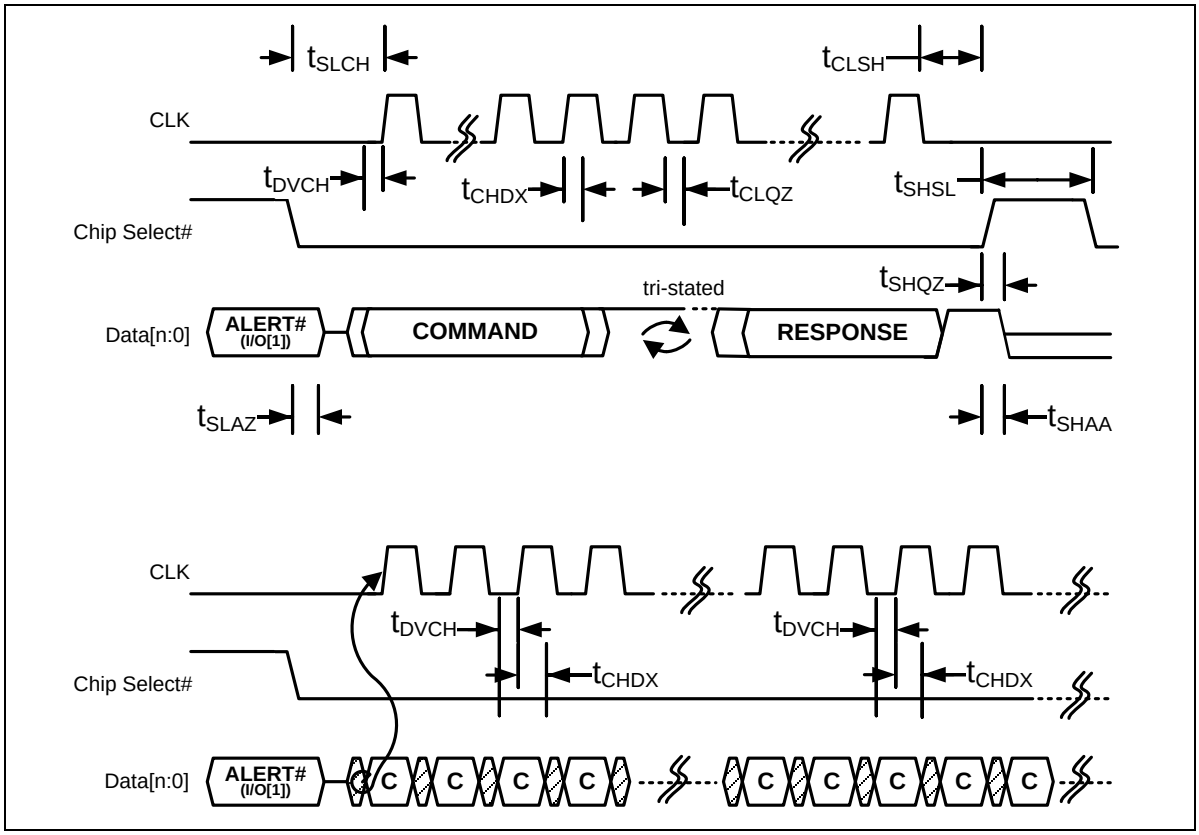
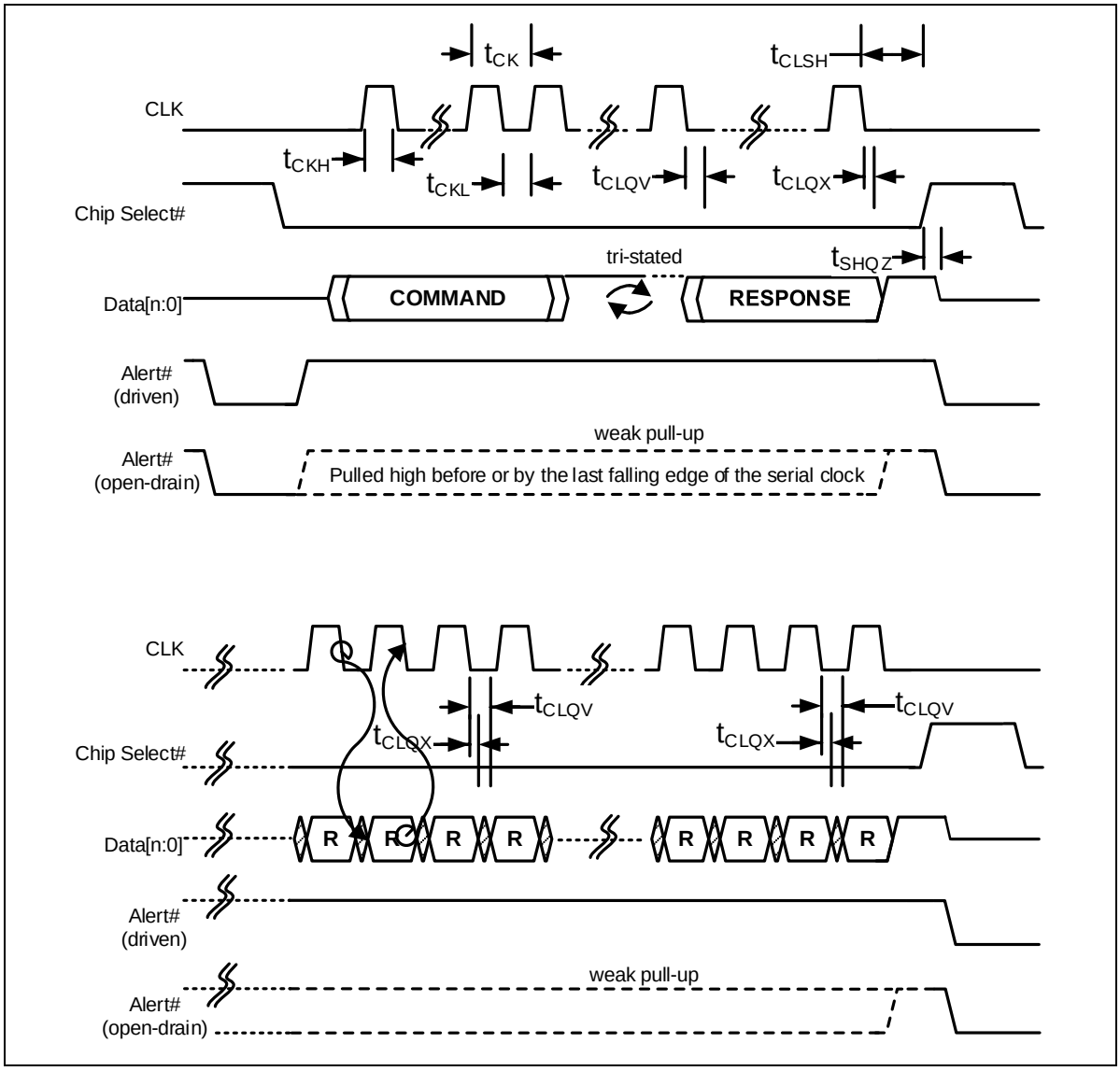


Figure 59: Output Timing Diagram



§

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 checking 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 are detectable over the eSPI bus by the eSPI slave and the eSPI master are described in Section 9.2.1 and 9.2.2.

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.

When eSPI Fatal Errors cannot be recovered by software, an eSPI_Reset# is required to be asserted to reset both eSPI master and slave. This may lead to a platform level reset for the recovery.

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's Detected Errors

This section describes the error detection and handling requirements for eSPI slave.

During the detection, the error may fall under one of the following Detection Phase (DP):

1. Error results in uncertainty on the command phase boundary. In this case, CRC checking is not applicable as CRC byte location is not known.

2. Command phase CRC error. Command packet is successfully decoded and its boundary is known. However, CRC error is detected on the command packet.

3. Correct CRC but other error detected. The error results in eSPI slave not being able to complete the execution of the command packet received.

4. Error detected outside of the command phase, such as unexpected deaasertion of Chip Select#, or any internal error detected by eSPI slave. The details of the internal errors are beyond the scope of the specification.

Table 23: Slave's Detected Errors

Error Condition[1]	DP[2]	R/O[3]	Slave's Response and Handling Response Code (RC), Completion (C), Virtual Wire (VW)			
			RC	C	VW	Description
Invalid Command Opcode	1	R	X			NO_RESPONSE Response Code. Command is discarded
Invalid Cycle Type (with respect to command)	1	R	X			NO_RESPONSE Response Code. Command is discarded

Error Condition[1]	DP[2]	R/O[3]	Slave's Response and Handling Response Code (RC), Completion (C), Virtual Wire (VW)			
			RC	C	VW	Description
Command phase CRC Error	2	R	X			NO_RESPONSE Response Code. Command is discarded
Unexpected deassertion of Chip Select#	1, 4	R				Slave tri-state the bus t_{SHQZ} after Chip Select# is deasserted. Note: Master is expected to detect CRC error during the response phase if CRC checking is enabled
Protocol Error • PUT without FREE • GET without AVAIL	3	R	X			FATAL_ERROR Response Code. Command is discarded
Malformed Packet during Command Phase <u>Peripheral Channel:</u> • Payload length > Max Payload Size (aligned) • Read request size > Max Read Request Size (aligned) • (Address + Length) crosses 4KB (aligned) boundary <u>Virtual Wire Channel:</u> • Count > Max Virtual Wire Count <u>OOB Channel:</u> • SMBus Byte Count > Max Payload Size <u>Flash Access Channel:</u> • Payload length > Max Payload Size • Read request size > Max Read Request Size	3	R	X		X	FATAL_ERROR Response Code. Command is discarded. or FATAL ERROR Virtual Wire. Before signaling the FATAL ERROR Virtual Wire, the transaction is completed on the eSPI bus (ACCEPT) with the following: Posted: Command is discarded Completion: Command is discarded Non-posted: Unsuccessful Completion without Data is returned and command discarded Virtual Wire: Command is discarded

Error Condition[1]	DP[2]	R / O[3]	Slave's Response and Handling Response Code (RC), Completion (C), Virtual Wire (VW)			
			RC	C	VW	Description
Unsupported Command (excluding Short Command)	3	O	X	X	X	NON_FATAL_ERROR Response Code. Command is discarded. or NON-FATAL ERROR Virtual Wire (when command is posted). Before signaling the NON-FATAL ERROR Virtual Wire, the transaction is completed on the eSPI bus with command discarded. or Unsuccessful Completion without Data (when command is non-posted). The transaction is completed with unsuccessful completion returned and command discarded
Unsupported Cycle Type (with respect to command)	3	O	X	X	X	NON_FATAL_ERROR Response Code. Command is discarded. or NON-FATAL ERROR Virtual Wire (when command is posted). Before signaling the NON-FATAL ERROR Virtual Wire, the transaction is completed on the eSPI bus with command discarded. or Unsuccessful Completion without Data (when command is non-posted). The transaction is completed with unsuccessful completion returned and command discarded
Unsupported Message Code	3	O	X		X	NON_FATAL_ERROR Response Code. Command is discarded. or NON-FATAL ERROR Virtual Wire. Before signaling the NON-FATAL ERROR Virtual Wire, the Message transaction is completed on the eSPI bus with command discarded

Error Condition[1]	DP[2]	R / O[3]	Slave's Response and Handling Response Code (RC), Completion (C), Virtual Wire (VW)			
			RC	C	VW	Description
Unsupported Length, Unsupported Address/Length alignment, Out of Range Address/Length combination (excluding Short Command)	3	O	X	X	X	NON_FATAL_ERROR Response Code. Command is discarded. or NON-FATAL ERROR Virtual Wire (when command is posted). Before signaling the NON-FATAL ERROR Virtual Wire, the transaction is completed on the eSPI bus with command discarded. or Unsuccessful Completion without Data (when command is non-posted). The transaction is completed with unsuccessful completion returned and command discarded
Short Command (terminated as connected, non-DEFER) that fails to be completed successfully • PUT_IORD_SHORT • PUT_IOWR_SHORT • PUT_MEMRD32_SHORT • PUT_MEMWR32_SHORT	3	O	X		X	NON_FATAL_ERROR Response Code. Command is discarded. or NON-FATAL ERROR Virtual Wire. Before signaling the NON-FATAL ERROR Virtual Wire, the transaction is completed on the eSPI bus with the following: Posted: Command is discarded Non-posted: Data of all 1's is returned for non-posted requires data. Command is discarded
All other posted Command that fails to be completed successfully	3	O	X		X	NON_FATAL_ERROR Response Code. Command is discarded. or NON-FATAL ERROR Virtual Wire. Before signaling the NON-FATAL ERROR Virtual Wire, the transaction is completed on the eSPI bus with command discarded

Error Condition[1]	DP[2]	R / O[3]	Slave's Response and Handling Response Code (RC), Completion (C), Virtual Wire (VW)			
			RC	C	VW	Description
All other non-posted Command that fails to be completed successfully, including Short Command after DEFER	3	O	X	X		NON_FATAL_ERROR Response Code. Command is discarded. or Unsuccessful Completion without Data. The transaction is completed with unsuccessful completion returned and command discarded
Unexpected completion received (i.e. completion without non-posted request, or completion with invalid tag)	3	O	X		X	NON_FATAL_ERROR Response Code. Command is discarded. or NON-FATAL ERROR Virtual Wire. Before signaling the NON-FATAL ERROR Virtual Wire, the transaction is completed on the eSPI bus with command discarded
All other Non-Fatal Error conditions detected by the Slave (including errors detected outside of the bus transaction)	3, 4	O			X	NON-FATAL ERROR Virtual Wire
All other Fatal Error conditions detected by the Slave (including errors detected outside of the bus transaction)	3, 4	O			X	FATAL ERROR Virtual Wire
Unsupported or Reserved Virtual Wire (VW) with Valid bit set • Within supported VW Indices, or • Unsupported or reserved VW Indices	3	O				No error is reported. The transaction is completed on the eSPI bus with Virtual Wire received being silently discarded without any effect

NOTES:

1. Invalid command opcode or cycle type refers to unknown command opcode or cycle type which is not defined by the eSPI Base Specification. This includes command opcode or cycle type that may be added later to any eSPI Addendum but not supported by the eSPI agents. Unsupported command opcode or cycle type refers to command opcode or cycle type which is defined by the eSPI Base Specification but it is not supported based on specific product requirement.
2. Detection Phase (DP). The error detected falls under one of the Detection Phase.

3. Required or Optional (R/O). Error conditions marked with Required (R) must be supported by eSPI slave.

9.2.1.1 No Response

In the case of invalid command, invalid cycle type or CRC error, 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, the eSPI master can deduce that there is either no slave present, or the slave has encountered error and responded with NO_RESPONSE. 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.

9.2.1.2 Fatal Error Response

The eSPI slave communicates to the eSPI master that the current transaction has a serious error by returning a Fatal Error response in the Response Phase, or by signaling the Fatal Error through the Virtual Wire message.

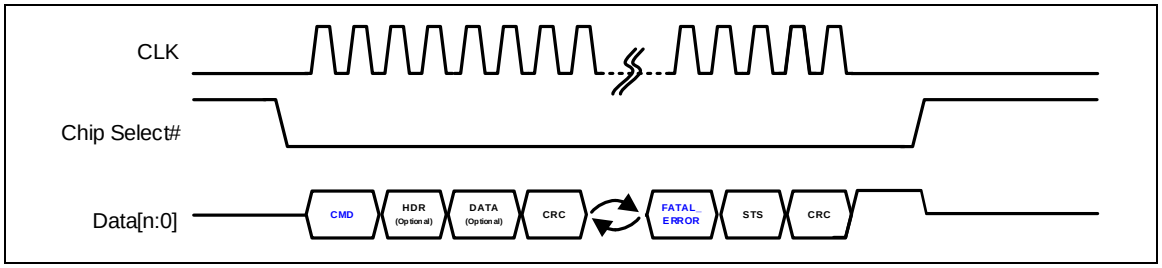
This could be due to the corresponding command could not be processed or that a severe error has been detected by the eSPI slaves that resulted in its inability to make forward progress.

The error conditions with Fatal Error response from slave are as described in Table 23.

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 a Response, a Status and a CRC. There is neither Header nor Data field during the Response phase.

Figure 60: Transaction with FATAL Error Response



9.2.1.3 Non-Fatal Error Response

The eSPI slave returns 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.

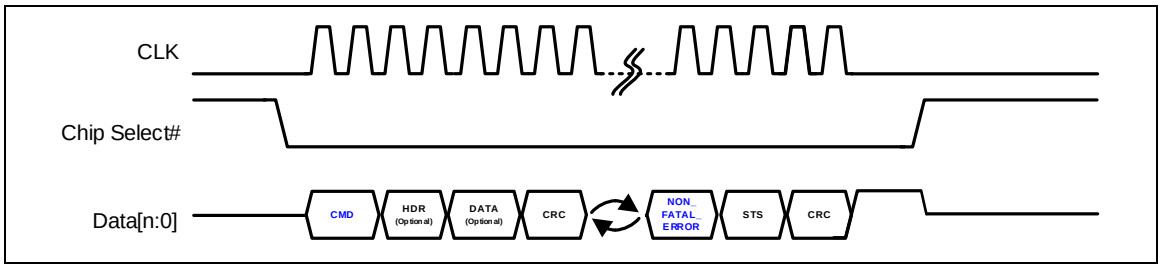
The error conditions with Non-Fatal Error response from slave are as described in Table 23.

The intent is 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.

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

Figure 61: Transaction with Non-FATAL Error Response



9.2.1.4 Unsuccessful Completion

For non-posted transaction that cannot be completed due to error, the eSPI slave returns an unsuccessful completion without data.

In the case of multiple split completions, the unsuccessful completion may be returned in any of the split completion. However, when one of the split completions has an unsuccessful completion status, the remaining split completions are not returned. The unsuccessful completion is the last split completion.

The error conditions with unsuccessful completion from slave are as described in Table 23.

9.2.1.5 Unexpected Chip Select# Deassertion

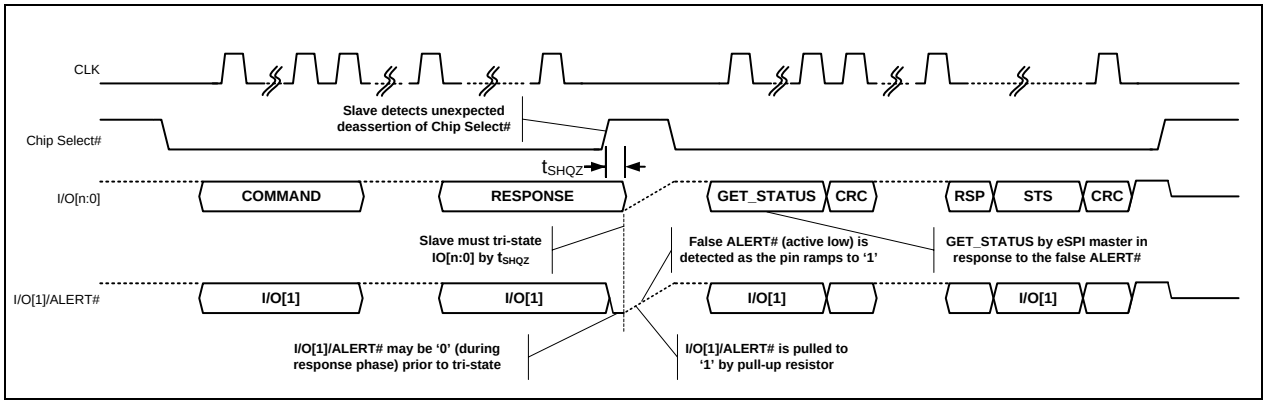
The deassertion of Chip Select# by eSPI master may be unexpected to eSPI slave. As an example, such error condition happens when the transaction length intended by the slave is corrupted on the bus and as a result, an incorrect length is being received by the master:

1. eSPI master deasserts Chip Select# sooner than slave expects. The slave expects to send more data but the transaction is ended with Chip Select# deassertion.
2. eSPI master deasserts Chip Select# later than slave expects. The slave detects more eSPI serial clocks after it has completed the response phase on the bus.

The eSPI slave is required to tri-state the bus t_{SHQZ} after Chip Select# is deasserted. The eSPI I/O[n:0] pins are expected to be pulled high by the pullup resistors on the bus. However, for scenario 1, due to time taken to ramp the pin high by the pull-up resistor when the prior state driven by the slave is '0' before the tri-stating, a false ALERT# may be detected on I/O[1] if the pin is also functioning as the ALERT# input in a single master-single slave configuration. Besides causing an unnecessary GET_STATUS from eSPI master, the spurious ALERT# will not affect the eSPI bus functionality.

The error condition is detectable by the eSPI master as it will result in a CRC error detected on the response phase when CRC checking is enabled.

Figure 62: Unexpected Chip Select# Deassertion



9.2.2 Master's Detected Errors

Table 24: Master's Detected Errors

Error Condition[1]	R / O[2]	Error Type	Master's Handling
Invalid Response Code (with respect to command)	R	Fatal Error	Master terminates the transaction abruptly by deasserting Chip Select#. For the corresponding (peripheral, flash access) channel with error detected: • Master to return Unsuccessful Completion to the initiator (such as host CPU) for ANY outstanding Master-to-Slave non-posted. • Discard any subsequent completion pulled from the Slave. (Note 1) • Discard any subsequent Master-to-Slave completions. (Note 2) • A reset to the Slave is required for this channel part of the software error handling. Notes: 1. Master-to-Slave non-posted has been completed with Unsuccessful Completion. 2. To avoid delivering subsequent data to the Slave as the erroneous response phase may be associated with a Master-to-Slave completion (thus it is "lost" from Slave perspective). Delivering subsequent completion to the Slave results in data going out of context.
Invalid Cycle Type (with respect to command)	R	Fatal Error	
Response phase CRC Error	R	Fatal Error	
Response Code: NO_RESPONSE (After initialization phase)[3]	R	Fatal Error	

Error Condition[1]	R / O[2]	Error Type	Master's Handling
Response Code: FATAL_ERROR	R	Fatal Error	In the case of NO_RESPONSE, Master terminates the transaction abruptly by deasserting Chip Select#. If Response Code is FATAL_ERROR or NON_FATAL_ERROR, Chip Select# is deasserted normally at the end of the response phase. For the corresponding channel with error detected, if command is a • Master-to-Slave non-posted. Master to return Unsuccessful Completion to the requestor • Master-to-Slave completion. Master to discard any subsequent completion to the Slave on this channel. (Note 3) A reset to the Slave is required for this channel part of the software error handling. Notes: 3. To avoid delivering subsequent data to the Slave as the erroneous response phase is associated with a Master-to-Slave completion (thus it is "lost" from Slave perspective). Delivering subsequent completion to the slave results in data going out of context.
Response Code: NON_FATAL_ERROR	O	Non-Fatal Error	Return Unsuccessful Completion to the Slave for non-posted that requires completion. Discard posted transaction from the Slave

Error Condition[1]	R / O[2]	Error Type	Master's Handling
Malformed Packet during Response Phase Peripheral Channel: • Payload length > Max Payload Size (aligned) • Read request size > Max Read Request Size (aligned) • (Address + Length) crosses 4KB (aligned) boundary Virtual Wire Channel: • Count > Max Virtual Wire Count OOB Channel: • SMBus Byte Count > Max Payload Size Flash Access Channel: • Payload length > Max Payload Size • Read request size > Max Read Request Size	R	Fatal Error	Return Unsuccessful Completion to the Slave for non-posted that requires completion. Discard posted transaction from the Slave
Unsupported Cycle Type (with respect to command)	O	Non-fatal Error	Return Unsuccessful Completion to the Slave for non-posted that requires completion. Discard posted transaction from the Slave
Unsupported Message Code	O	Non-fatal Error	Discard Message transaction from the Slave
Unsupported Length, Unsupported Address/Length alignment, Out of Range Address/Length combination	O	Non-fatal Error	Return Unsuccessful Completion to the Slave for non-posted that requires completion. Discard posted transaction from the Slave
Unsuccessful completion received	O	Non-fatal Error	Forward unsuccessful completion to the requester
Receive ERROR FATAL Virtual Wire	R	Fatal Error	No additional handling besides error logging and reporting
Receive ERROR NON FATAL Virtual Wire	O	Non-fatal Error	No additional handling besides error logging and reporting
Unexpected completion received (i.e. completion without non-posted request, or completion with invalid tag)	O	Non-Fatal Error	The transaction is completed on the eSPI bus with Completion received silently discarded.

Error Condition[1]	R/O[2]	Error Type	Master's Handling
Unsupported or Reserved Virtual Wire (VW) with Valid bit set • Within supported VW Indices, or • Unsupported or reserved VW Indices	O	No error reported	The transaction is completed on the eSPI bus with Virtual Wire received being silently discarded without any effect.

NOTES:

1. Invalid command opcode or cycle type refers to unknown command opcode or cycle type which is not defined by the eSPI Base Specification. This includes command opcode or cycle type that may be added later to any eSPI Addendum but not supported by the eSPI agents. Unsupported command opcode or cycle type refers to command opcode or cycle type which is defined by the eSPI Base Specification but it is not supported based on specific product requirement.
2. Required or Optional (R/O). Error conditions marked with Required (R) must be supported by eSPI master.
3. During initialization phase, the NO_RESPONSE for a GET_CONFIGURATION cycle indicates that slave is not present on the corresponding Chip Select#. It is not an error condition.

9.2.2.1 Master's 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 an invalid Response Code, an invalid Cycle Type, or a CRC error is detected on the Response Phase received from eSPI slave, the Response packet boundary is indeterminate. When this happens, 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 detect and handle the unexpected deassertion of Chip Select#.

9.3 Reset

9.3.1 eSPI 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 for all the channels will be reset by eSPI Reset#, including all the internal queues. 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.

9.3.2 In-band RESET Command

In-band RESET command intends to recover the eSPI master and slaves such that both sides are reset to a known set of interface settings to allow communication to re-establish. The eSPI interface must be still functional in order to transmit and receive the RESET command.

One example where eSPI master and slaves may go out of synchronization is when SET_CONFIGURATION from eSPI master to eSPI slave results in an error. As the transaction does not complete successfully, it is uncertain on the state of the interface settings after the error.

The in-band RESET command has the following behavior. It is defined such that the slave is able to detect the In-band RESET command opcode regardless of the I/O mode, i.e. either in Single, Dual or Quad I/O configuration.

- RESET command opcode is FFh (i.e. all 1's).
- It is sent with the 20MHz speed or lower.
- No CRC byte and thus CRC checking must be ignored.
- The transaction has no response phase from eSPI slave.
- All I/O lines are driven to high ('1') for 16 eSPI clocks and tri-stated at the deassertion edge of CS#, meeting the t_{SHQZ} Output Disable timing.

When eSPI slave detects the RESET command opcode, it behaves in the following:

- Ignore all the subsequent bits received.

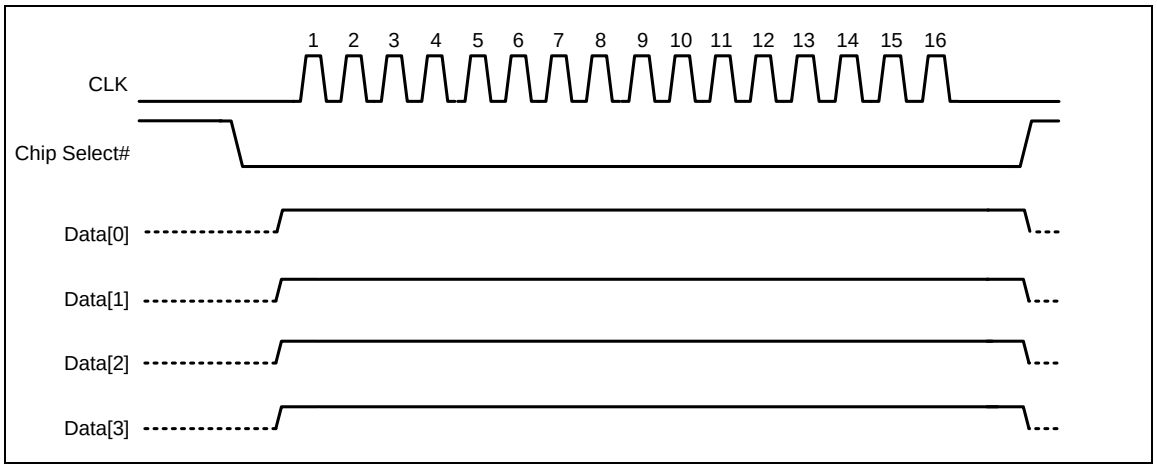
- Bypass or ignore the CRC checking.
- Wait until CS# deassertion, and assert the in-band reset internally at the CS# deassertion edge.

The In-band RESET will reset the following slave register settings to the default reset value:

- Offset 008h-00Bh: General Capabilities and Configurations

All other slave registers are not reset by the In-band RESET, and they must retain their values across the In-band RESET.

Figure 63: In-band RESET Command



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 a GET_CONFIGURATION 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.
    b. GET_CONFIGURATION is initiated.
5. eSPI master evaluates the discovered capabilities and performs 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.
8. Chipset waits for the SLAVE_BOOT_LOAD_DONE Virtual Wire message from eSPI slave before continuing the exit sequence. eSPI slave must send the SLAVE_BOOT_LOAD_DONE message regardless of whether flash access channel is supported. SLAVE_BOOT_LOAD_STATUS must be valid at the same time or prior to the SLAVE_BOOT_LOAD_DONE Virtual Wire.
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.

§