OpenFSI Specification

Field Replaceable Unit Service Interface

Workgroup Specification

Revision 1.0.0 (December 12, 2016)

OpenPOWER™

www.openpowerfoundation.org

OpenFSI Specification: Field Replaceable Unit Service Interface

FSI Specication Work Group

Revision 1.0.0 (December 12, 2016)

Abstract

This document describes a Field replaceable unit Support Interface(FSI) suited to service all chips in a computer system via a common serial interface. Key features are ease of use, easy scalability, robustness, and support for virtualisation and the tunneling of interrupts and DMA control signals across the interface. FSI is superior to similar industry standard interfaces in these and many other features including speed, distance, data protection, and address range.

This document is a Standard Track, Work Group Specification work product owned by the FSI Specification Workgroup and handled in compliance with the requirements outlined in the *OpenPOWER Foundation Work Group (WG) Process* document. It was created using the *Master Template Guide* version 1.0.0. Comments, questions, etc. can be submitted to the public mailing list for this document at `<fru-fsi@mailinglist.openpowerfoundation.org>`.

Table of Contents

List of Figures

List of Tables

Preface

1. Conventions

The OpenPOWER Foundation documentation uses several typesetting conventions.

Notices

Notices take these forms:

Note

A handy tip or reminder.

Important

Something you must be aware of before proceeding.

Warning

Critical information about the risk of data loss or security issues.

Changes

At certain points in the document lifecycle, knowing what changed in a document is important. In these situations, the following conventions will used.

- *New text will appear like this.* Text marked in this way is completely new.
- ~~Deleted text will appear like this.~~ Text marked in this way was removed from the previous version and will not appear in the final, published document.
- Changed text will appear like this. Text marked in this way appeared in previous versions but has been modified.

Command prompts

In general, examples use commands from the Linux operating system. Many of these are also common with Mac OS, but may differ greatly from the Windows operating system equivalents.

For the Linux-based commands referenced, the following conventions will be followed:

$ prompt Any user, including the `root` user, can run commands that are prefixed with the $ prompt.

 # prompt The `root` user must run commands that are prefixed with the # prompt. You can also prefix these commands with the **sudo** command, if available, to run them.

Document links

Document links frequently appear throughout the documents. Generally, these links include a text for the link, followed by a page number in parenthesis. For example, this link, Preface [vi], references the Preface chapter on page vi.

2. Document change history

This version of the guide replaces and obsoletes all earlier versions.

The following table describes the most recent changes:

Revision Date	Summary of Changes
December 12, 2016	• Workgroup Specification version 1.0.0
August 16, 2016	• Chapter 1, added conformance definition Chapter 4, corrected typos in tables
June 5, 2016	• Public Review Draft - approved by WG
September 2, 2015	• Corrected comments per WG review
September 1, 2015	• Creation based on IBM FSI Spec Contribution

1. FSI Specification

1.1. Abstract

This document describes a *Field replaceable unit Support Interface*(FSI) suited to service all chips in a computer system via a common serial interface. Key features are *ease of use*, *easy scalability*, *robustness*, and support for *virtualisation*and the *tunneling of interrupts and DMA control signals*across the interface. FSI is superior to similar industry standard interfaces in these and many other features including speed, distance, data protection, and address range.

FSI is a point to point two wire interface operating in half duplex mode, which is capable of supporting distances of up to 4 meters at up to 166 MHz bus frequency. All operations are fully CRC checked to allow error recovery.

With its versatile architecture, the FSI is suited for a wide range of applications from service access to in-system test.

1.2. About this Document

This document assists device and system designers to understand how the FSI works and how to implement a working application. Various operating modes are described. It contains a comprehensive introduction into FSI data transfer schemes. Detailed sections cover the timing and electrical specifications for FSI in different operating modes.

Designers of FSI compatible devices should use this document as a reference and ensure that new devices meet all limits specified in this document. Designers of systems that include FSI devices should consider design rules provided in this document.

This specification aims to provide a description of FSI without referring to implementation details of existing FSI master and FSI slave versions.

To obtain the latest copy of this document, please contact the document owner.

1.3. Abbreviations

ABIST	Array Built-In Self Test
BIST	Built-In Self Test
BMC	Baseboard Management Controller
CFAM	Common FRU Access Macro; usually incorporates two FSI slaves and provides access to other chip pervasive / service functions via technology library support or logic methodology functions. resides on a standby voltage island
CPU	Central Processor Unit
CRC	Cyclic Redundancy Check
DC	Direct Current
DIO	Digital Input/Output
DMA	Direct Memory Access
FIFO	First In First Out
FRU	Field Replaceable Unit

FSI	Field Replaceable Unit(FRU) Support Interface
FSP	Flexible Support Processor
GND	Ground Power
I2C	Inter-Integrated Circuit
JTAG	Joint Test Action Group
LBIST	Logic Built-In Self Test
OPB	On-Chip Peripheral Bus
PCI-E	Peripheral Component Interconnect Express
PLL	Phase-Locked Loop
SCOM	Serial Communication
SCSN	System Control and Service Network
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus

1.4. Conformance Definition

Designs conforming to this specification must implement the entire FSI specification defined in this document. Features identified as optional are not required and may be implemented at the designers discretion.

The following list identifies the chapters, sub chapters and tables describing optional features.

- Section 2.2, “Key characteristics” [4]
- Table 4.1, “Protocol Command Messages” [9]
- Table 4.2, “Protocol Response Messages” [10]
- Section 4.2.11, “Overview: FSI Protocol Messages” [26]

2. Introduction

The servicing of a computer system is essential in order to manage it and to give the user the possibility to interact with the system in an automated or manual way. The system control structure and the firmware which are in charge of managing the components of a given system are getting increasingly complex the more chips from different vendors need to be serviced. One key contributor to the firmware complexity are the different hardware protocols to be supported, like industry standard protocols as I2C and JTAG or vendor specific solutions. In addition we do encounter major issues with the scalability from small systems to enterprise systems when using industry standard protocols.

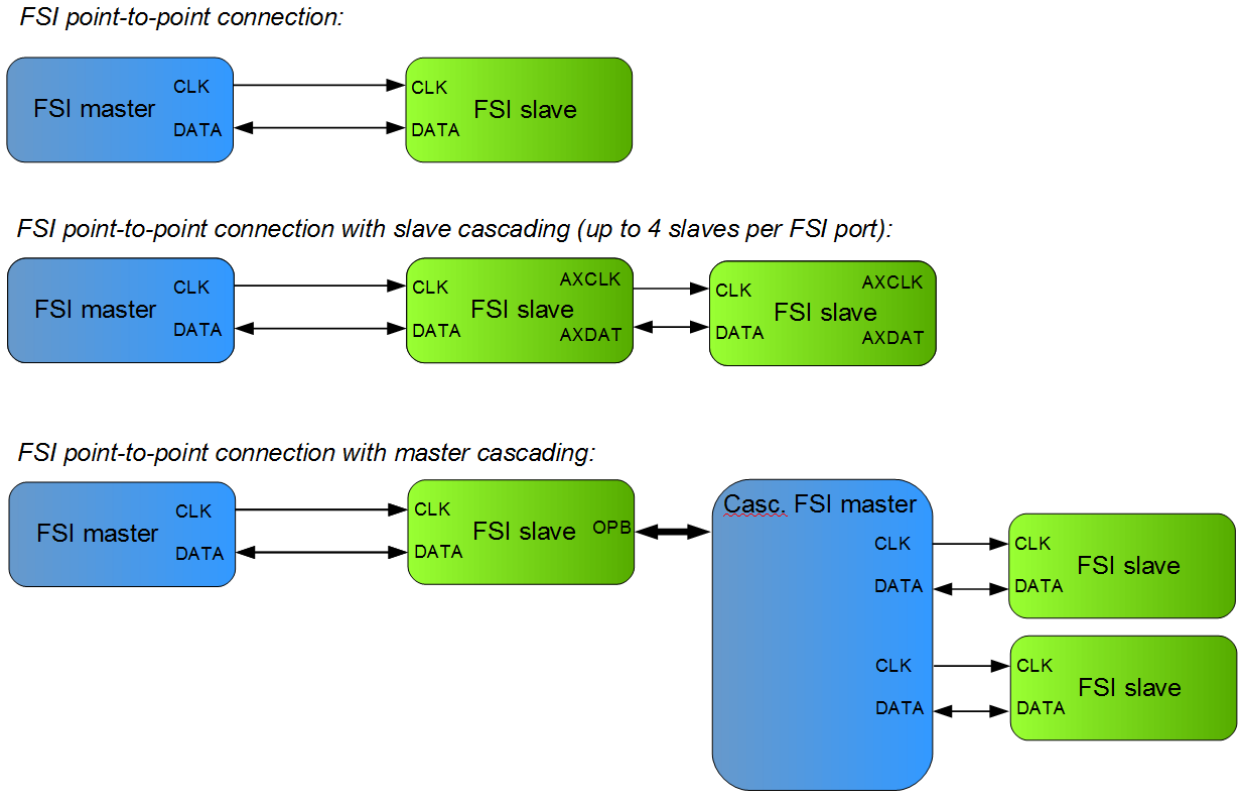
With FSI (Field replaceable unit Support Interface) we provide a seamless upgrade path from small systems up to enterprise systems due to the scalability and robustness of the FSI interface. The primary reason for FSI is the service access to all chips of a system. On the other hand FSI is also useful in the test area to run BIST or stimulate the chip with test patterns.

The FSI interface is a serial data link that operates in half duplex mode. Devices communicate in master/slave mode where the master device initiates a data transfer. FSI is a point-to-point two wire interface which is capable of supporting distances of up to 4 meters at up to 166 MHz bus frequency.

FSI interfaces have been used successfully for many years in IBM servers to attach IBM Flexible Support Processors to CPUs and IBM ASICs.

2.1. FSI Topologies

Figure 2.1. Different FSI Topologies



2.2. Key characteristics

FSI physical and electrical interface

- 2-wire interface: 1x unidirectional clock line1x bi-directional data line
- tristate push-pull / 1.2 V / 40 Ohm
- scalable bandwidth through programmable clock frequency (up to 166 MHz)
- slave presence detection via DC-level sensing (plug detect)

FSI protocol

- fully interlocked command/response scheme
- 23 bit addressing per FSI port (8MByte)

  default: absolute addressing
  optional: relative addressing
- 8bit / 16 bit / 32 bit data size
- flow control and slave error reporting via feedback
- CRC error reporting via feedback, optional re-transmission for error recovery

FSI data message transfer

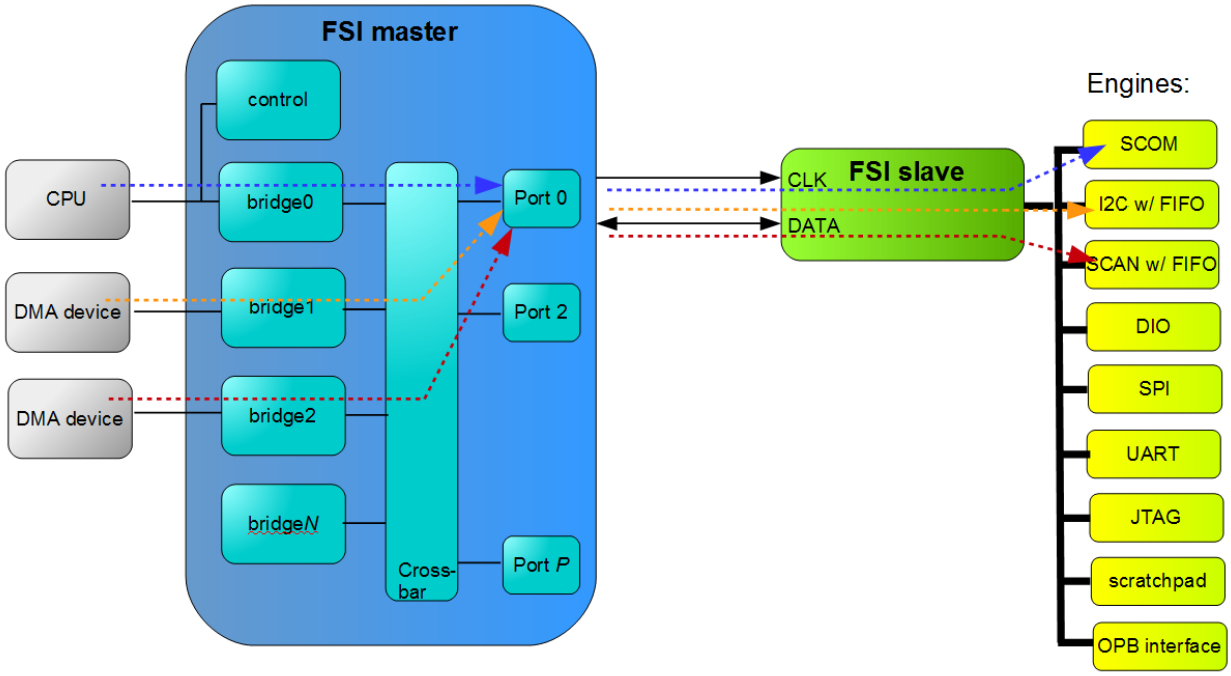
- half duplex with programmable turn-around time
- synchronous transmission from master to slave, asynchronous transmission from slave to master
- bit-oriented scheme (command/response message vary in length)
- 4 bit CRC field appended to each command/response
- optional error recovery performed by hardware
- optional slave triggered operation through hardware protocol based signaling (2 interrupt levels per slave, 2 sets of dma request/endOfTransfer for FIFO-device paced DMA transfers)

Master/Slave architecture

- port controller
- multiple bridge support for DMA channels via crossbar switches
- interface sharing:

  up to 4 cascaded slaves can be mapped into one FSI port,
  support of up to 8 DMA channels per FSI port (2 for each slave)

Figure 2.2. FSI master and slave architecture



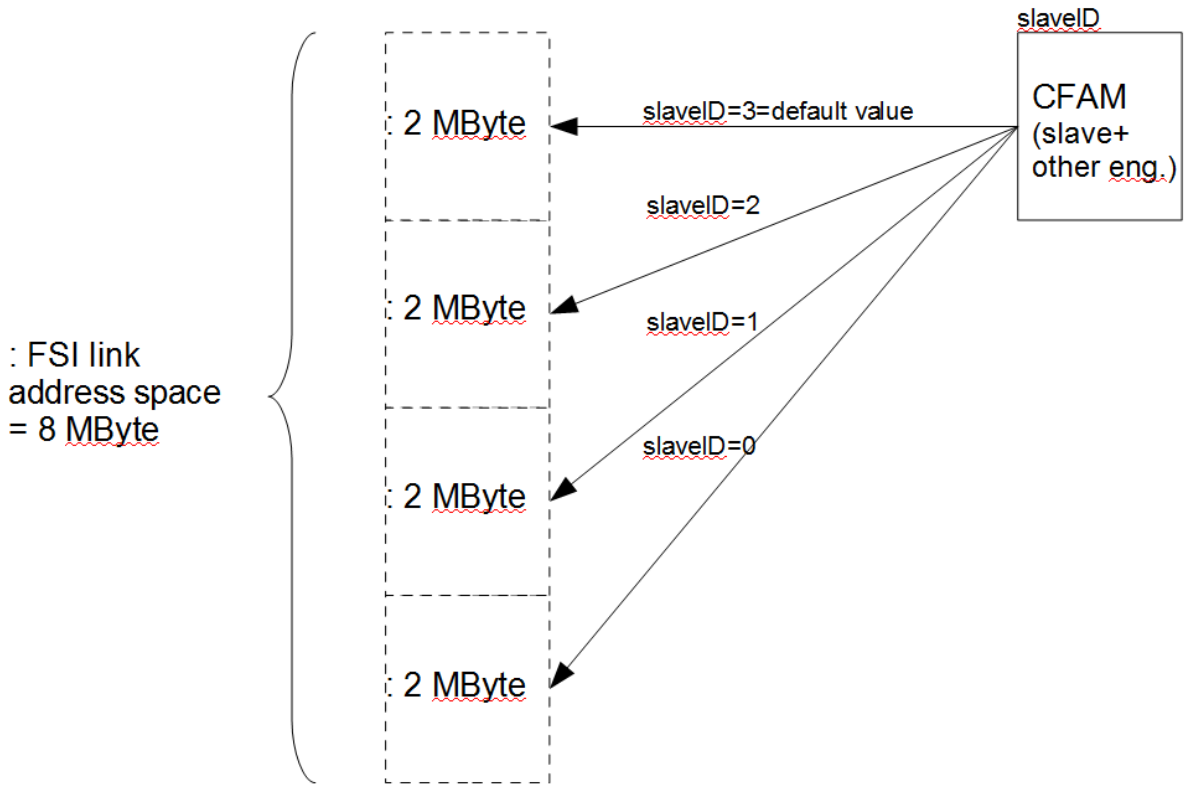
2.3. Link address space

Each FSI link supports a 8 Mbyte address space. This space is logically divided into 4 pages.

A CFAM, which incorporates FSI slave and its associated engines to access the chip pervasive functions, can be mapped to one page through a 2 bit offset indicator, called slaveID. Thus, its possible to accommodate up to 4 CFAMs per FSI link. Having more than one CFAM connected to one FSI link is called “cascading”. This feature is used to increase the fan-out but requires a special CFAM with auxiliary FSI links.

Figure 2.3, “Link Address Space” [6] show the address allocation as a function of the 2 bit slaveID; each location can be directly addressed through an absolute address command (ABS_AR).

Figure 2.3. Link Address Space



Note

1. A CFAM usually incorporates two independent FSI slaves, connected to different FSI links, for redundancy reasons; so, there are two different slaveIDs and thus each FSI link might map the address space differently.

2. After applying a power on reset to the whole CFAM, the slaveIDs of its FSI slaves will be set to 3; a BREAK command has the same effect but affects only the connected slave.

3. In relative addressing mode, the FSI master decides if the next command must be issued as an absolute, relative or same address command; is the next command within a plus/minus 256 byte distance then a relative address is used; if it's exactly the same address as the previous one, then a same address command is used and in all other cases an absolute address command is used.

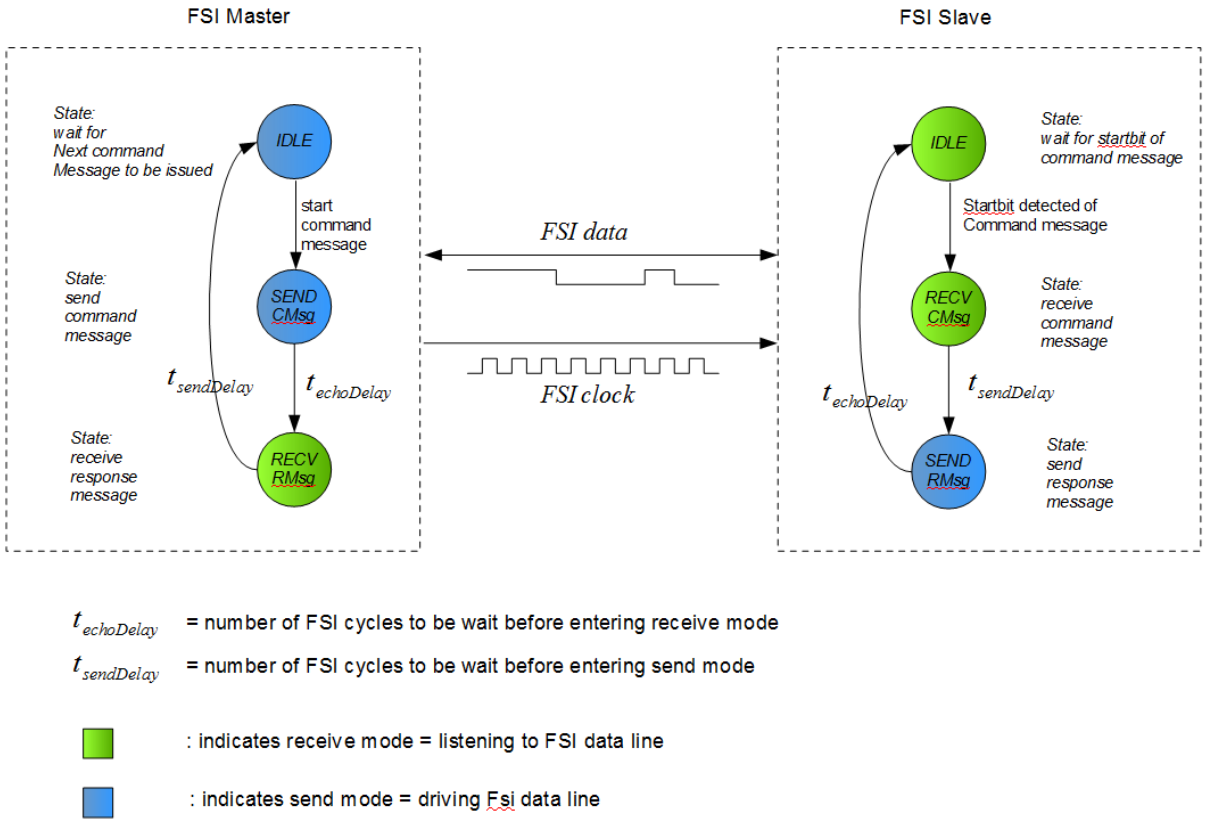
3. Mode of Operation

Mode of operation is half-duplex, i.e., master and slave drives the data line alternately. It is a synchronous, bit-serial transmission and the FSI master provides the clock to the slave and its attached functional units which is just called “Common Access Macro” (CFAM).

Before any transmission can be started, the FSI port needs to be enabled; by doing that the FSI port turns on the interface clock and holds the FSI data line in de-asserted state (logical 0)

Figure 3.1, “FSI mode of operation” [7] illustrates the main principle:

Figure 3.1. FSI mode of operation



$t_{echoDelay}$ = number of FSI cycles to be wait before entering receive mode

$t_{sendDelay}$ = number of FSI cycles to be wait before entering send mode

: indicates receive mode = listening to FSI data line

: indicates send mode = driving Fsi data line

It is assumed that master and slave are in IDLE state and the FSI clock is running So, master is ready to send any command to the slave and the slave itself is ready to receive any command from the master. As soon as the master is started, it moves on to the SEND state and applies the bit-serial FSI command to the data line.

Before entering the RECV state and listening to the data line, the master waits for a specific number of FSI cycles (echoDelay time). That prevents the master from listening to the data line too early since it just takes a while till the slave has captured the complete command and can react to it.

Each command/response is preceded by a start bit; the slave moves on to RECV state on receipt of that bit. When the complete bit stream of the applied command has been received then the slave also waits a specific number of FSI cycles (sendDelay time) before it moves on to the SEND state to

avoid any disturbances through possible line reflections. In SEND mode, the slave enables its line driver and sends its response to the master. The response is also preceded by a start bit; that gives the master the possibility to re-synchronize the incoming response.

Both delay timers, echoDelay $t_{echoDelay}$ and sendDelay $t_{sendDelay}$, are programmable in the range of 0..15 FSI cycles; they default to 15; hardware always adds 1 FSI cycle to the programmed values.

To re-synchronize the slave in protocol error cases, a special uni-directional command, called BREAK command, is defined to forcefully bring the slave into IDLE state.

4. FSI protocol

The following chapter introduces the messages an their exchange rules step by step.

Emphasis is placed on the term *“message”*. A message can either be a command that is sent from master to slave or a response of a slave that is sent back to the master.

Each message is described in a bit-oriented format and may vary in length to minimize overhead. The transmission of messages takes place serially on the FSI data line.

Command messages are transmitted in sync with the FSI clock line, the reception of the response message, however, might be out of sync.

The next two tables provides an overview of the two message types of the FSI protocol.

After that, all message exchange rules are treated in detail.

Command messages:

They represent all messages that can be sent out by a FSI master. The ones marked as “optional” belong to a protocol feature that can be enabled or disabled. Generally, all “optional” features are disabled per default.

Table 4.1. Protocol Command Messages

Command Message	**Description**
BREAK	Initializes all FSI slaves of a FSI interface
ABS_ADR	**Absolute Address command**used to initiate a data transfer number of transmitted address bits: **21 bit** supported data size: 32 bit, 16 bit, 8 bit
REL_ADR	**Optional: Relative Address command**used to initiate a data transfer, based on previous address. number of transmitted address bits: **8 bit**(plus sign bit) supported data size: 32 bit, 16 bit, 8 bit
SAME_ADR	**Optional: Same Address command**used to initiate a data transfer, based on previous address. number of transmitted address bits: **2 bit** supported data size: 32 bit, 16 bit, 8 bit
TERM	**Terminate command** Sent out to terminate/abort a data transfer.
D_POLL	**Data Polling command** Sent out in response of a busy message of a FSI slave.
E_POLL	**Optional:**used for hardware crc error recovery
I_POLL	**Optional: Interrupt Polling command** Sent out to get interrupt and DMA request/endOfTransfer status of a FSI slave.

Response messages:

They represent all messages that can be sent out by a FSI slave in response to a command message.

The one marked as “optional” belong to a protocol feature that can be enabled or disabled and is disabled per default.

Table 4.2. Protocol Response Messages

Response Message	**Description**
ACK	**Acknowlegde response** Indicates a successful write of data (32 bit, 16 bit or 8 bit); it completes the data transfer.
ACK_D	**Acknowlegde plus Data response** Indicates that FSI slave is providing the requested read data (32 bit, 16 bit or 8 bit); it completes the data transfer.
BUSY	**BUSY response** Indicates that FSI slave is currently not able to complete (acknowledge) the data transfer. Used to accommodate low speed devices on the slave side.
ERR_A	**Error Acknowledge response** Indicates that FSI slave has encountered an error in the course of executing the data transfer.
ERR_C	**Optional:**used for hardware CRC error recovery
I_POLL_RSP	**Interrupt polling Response** If FSI Master has Interrupt Poll (I_POLL) enabled then FSI Slave needs to support Interrupt Poll response (I_POLL_RSP). FSI slave provides the current interrupt status plus DMA control signal status of the DMA channels (request, endOfTime) for hardware paced DMA transfers

4.1. Command/Response Protocol

The message exchange principle follows a command/response method in which a FSI master sends out a command and waits for a response of the connected FSI slave.

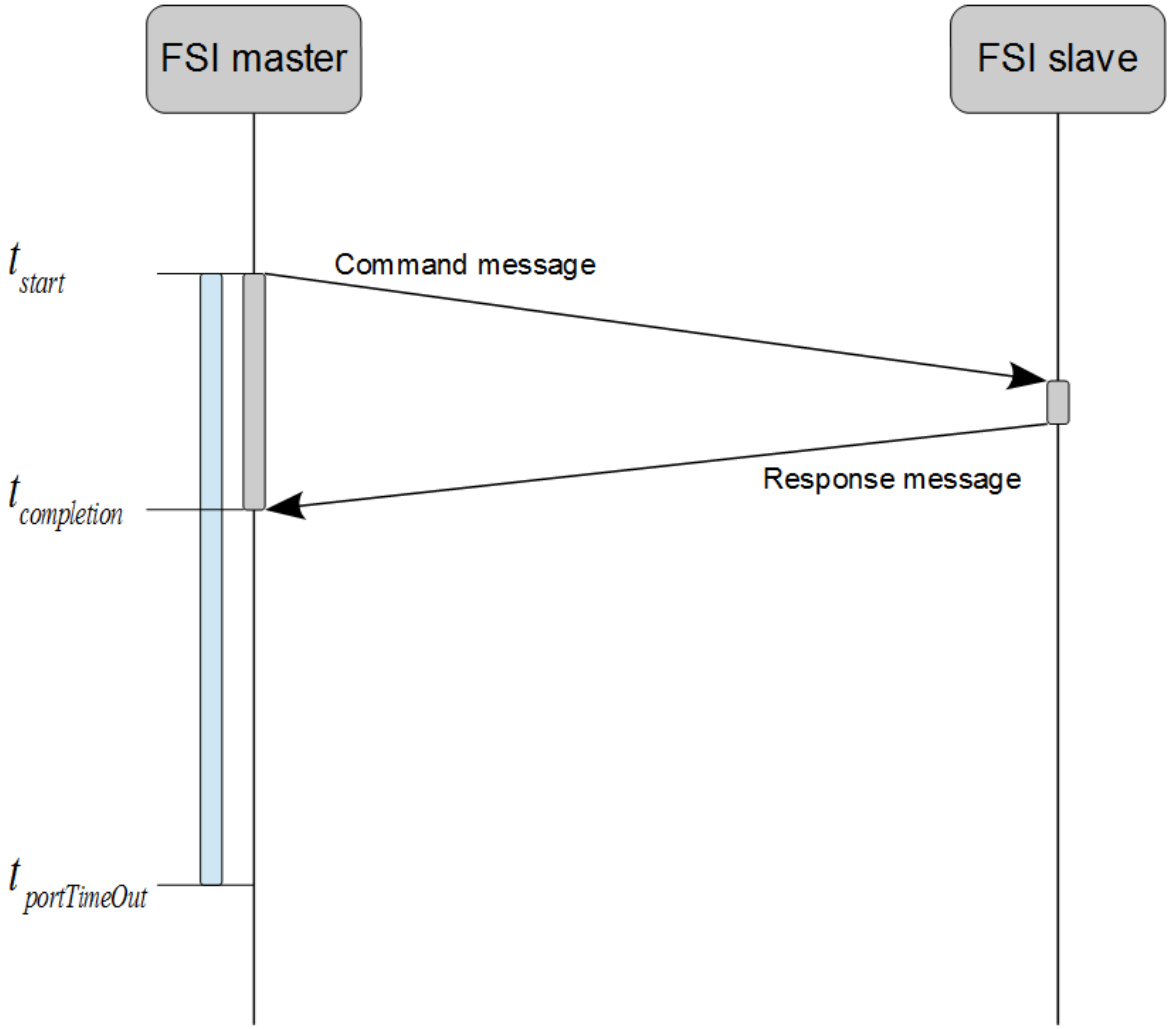
Notice: Once, a FSI port is enabled, the FSI clock is running continuously. There exists no framing of the command/response message. Instead, a leading “startbit “ is used to indicate the beginning of a message. That startbit pertains to each message. Furthermore, each message ends with a 4-bit CRC field that protects each message transmission.

The master issues a command message, waits for a programmable turn-around time, starts the FSI port timer, changes into receive mode and awaits a response from the slave before the port timer expires.

Should the slave not be able to respond within the defined time interval of 256 FSI clock cyles ($t_{portTimeOut}$) then the master will exhibit a so-called FSI port timeout.

In the interaction diagram below, the master issues the command at time t_0 and get a response back in time ($t_{completion}$)

Figure 4.1. Send command and Wait for response



4.2. FSI Data Transfer

For transferring data from master to slave (write case) or from slave to master (read case), FSI provides a so-called “Absolute Address Command Message”, short ABS_ADR.

By using that command message, the master transmits all FSI address bit to the slave,

i.e. 21 address bits, supporting a random access within a 2 Mbyte address space per FSI slave.

For the case of accomodating more than one FSI slave within a given FSI port (called “slave cascading”), the protocol defines in addition a 2 bit FSI slave identifier as part of each message, totally providing 8 Mbyte address range support per FSI port.

Three data transfer sizes are supported, namely

- 8-bit data transfers

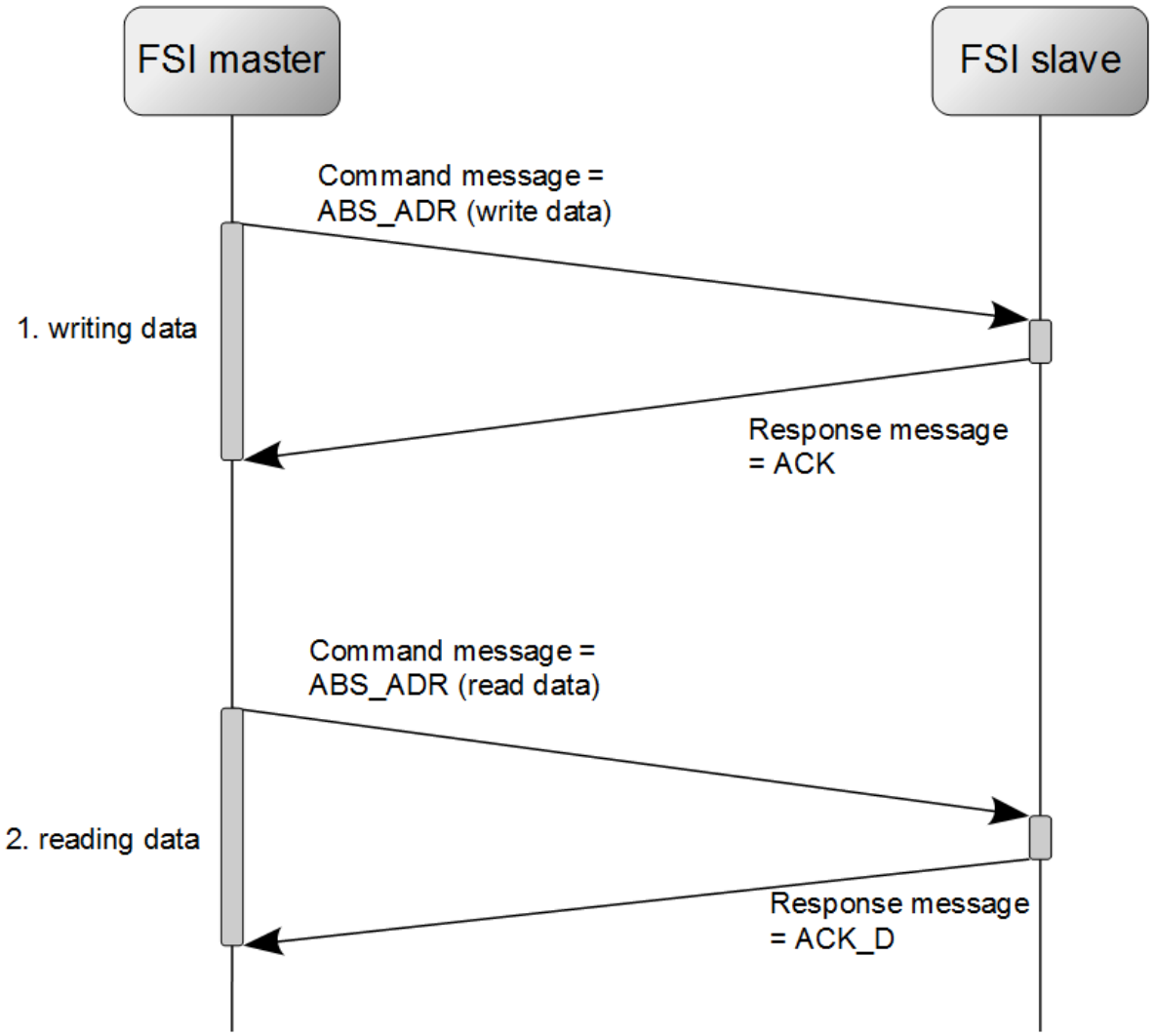
- 16-bit data transfers
- 32-bit data transfers.

The interaction diagram depicts a write data transfer, followed by a read data transfer and shows the basic exchange rules. It is assumed that the FSI slave is able to successfully complete the transfer in time without using flow control techniques for accommodating low-speed devices.

When writing data, successful completion is signalled by the FSI slave by responding with an acknowledge message (ACK).

In case of reading data, successful completion is indicated by an “acknowledge with appended data” message (ACK_D)

Figure 4.2. Example of write and read data transfer



4.2.1. Absolute Addressing

4.2.1.1. Format description: ABS_ADR

Table 4.3. Command Message Format: ABS_ADR (Absolute Address Commmand)

<table>
<tr><th>Bit [a]</th><th>Field [b]</th><th>Description</th></tr>
<tr><td>0</td><td>'1'</td><td>Start bit (very first bit of a message)</td></tr>
<tr><td>1:2</td><td>slaveID</td><td>Used to distinguish up to four cascaded FSI slaves per FSI port:
“00”: slaveID=0, “01”: slaveID=1, “10”:slaveID=2, “11: slaveID=3</td></tr>
<tr><td>3:5</td><td>"100"</td><td>Identifies ABS_ADR message</td></tr>
<tr><td>6</td><td>Read/notWrite</td><td>If at '1' then read data transfer is initiated.
If at '0' then write data transfer is initiated.</td></tr>
<tr><td>7:28</td><td>Address/Data Size</td><td>21-bit Address field for random access, 1-bit Data Size
<table>
<tr><th>Type</th><th>Address bit(7:25)</th><th>Address bit(26)</th><th>Address bit(27)</th><th>Data Size bit(28)</th></tr>
<tr><td>word</td><td>addr(0:18)</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>halfword</td><td>addr(0:18)</td><td>addr(19)</td><td>0</td><td>1</td></tr>
<tr><td>byte</td><td>addr(0:18)</td><td>addr(19)</td><td>addr(20)</td><td>0</td></tr>
</table></td></tr>
<tr><td>If bit6='0'
[29:36]
[29:44]
[29:60]</td><td>Write Data field:
8 bit Data
16 bit Data
32 bit Data</td><td>This data field only exists if bit6='0' (write data transfer):
8 bit data
16 bit data
32 bit data</td></tr>
<tr><td>Last 4 bits</td><td>CRC Value</td><td>4 bit CRC field</td></tr>
</table>

[a]Bit 0 is the most significant bit and is transmitted first on the FSI data line

[b]all described values represent values of a positive active logic if not otherwise stated; on the FSI data line they appear inverted, i.e. a logical '0' value is associated to a HIGH level and a logical '1' value to a LOW level.

4.2.1.2. Format description: ACK

Table 4.4. Response Message Format: ACK (Acknowledge)

Bit [a]	Field [b]	Description
0	'1'	Start bit (very first bit of a message)
1:2	slaveID	Used to distinguish up to four cascaded FSI slaves per FSI port: “00”: slaveID=0, “01”: slaveID=1, “10”:slaveID=2, “11: slaveID=3
3:4	"00"	Identifies ACK message
Last 4 bits	CRC Value	4 bit CRC field

[a]Bit 0 is the most significant bit and is transmitted first on the FSI data line

[b]all described values represent values of a positive active logic if not otherwise stated; on the FSI data line they appear inverted, i.e. a logical '0' value is associated to a HIGH level and a logical '1' value to a LOW level.

4.2.1.3. Format description: ACK_D

Table 4.5. Response Message Format: ACK_D (Acknowledge with appended data)

Bit [a]	Field [b]	Description
0	'1'	Start bit (very first bit of a message)
1:2	slaveID	Used to distinguish up to four cascaded FSI slaves per FSI port: “00”: slaveID=0, “01”: slaveID=1, “10”:slaveID=2, “11: slaveID=3
3:4	"00"	Identifies ACK message
	Read Data field:	
[5:12]	8 bit Data	8 bit data
[5:20]	16 bit Data	16 bit data
[5:36]	32 bit Data	32 bit data
Last 4 bits	CRC Value	4 bit CRC field

[a]Bit 0 is the most significant bit and is transmitted first on the FSI data line

[b]all described values represent values of a positive active logic if not otherwise stated; on the FSI data line they appear inverted, i.e. a logical '0' value is associated to a HIGH level and a logical '1' value to a LOW level.

4.2.2. Relative Addressing

To reduce protocol overhead, FSI defines a “relative addressing” mode that reduces the number of transmitted address bits significantly by relating to the address of the preceding data transfer command. If enabled then the FSI master choses the command to be issued based on the previous one, comparing the word-aligned (4 byte) address.

In cases in which that word-aligned address is identical, the same address command can be used (SAME_ADR) and then only the two low-order address bits are transmitted for distinguishing 8 bit, 16 bit or 32 bit data accesses. That greatly improves the access time when dealing with FIFO devices on the slave side.

If the word-aligned address is not identical but lies within a plus/minus 256 byte address range then the FSI master can use a relative address command (REL_ADR). That reduces the transmitted address bits from 21 to 8 bit.

In all other cases, an absolute address command (ABS_ADR) needs to be chosen.

4.2.2.1. Format description: REL_ADR

Table 4.6. Command Message Format: REL_ADR (Relative Address Commmand)

Bit [a]	Field [b]	Description
0	'1'	Start bit (very first bit of a message)
1:2	slaveID	Used to distinguish up to four cascaded FSI slaves per FSI port: “00”: slaveID=0, “01”: slaveID=1, “10”:slaveID=2, “11: slaveID=3
3:5	"101"	Identifies REL_ADR message
6	Read/notWrite	If at '1' then read data transfer is initiated. If at '0' then write data transfer is initiated.
7	Address sign	
8:16	Address/Data Size	8-bit Address field for random access, 1-bit Data Size

Bit [a]	Field [b]	Description
		Type / Address bit(8:13) / Address bit(14) / Address bit(15) / Data Size bit(16) word / addr(0:5) / 0 / 1 / 1 halfword / addr(0:5) / addr(6) / 0 / 1 byte / addr(0:5) / addr(6) / addr(7) / 0
If bit6='0'	Write Data field:	This data field only exists if bit6='0' (write data transfer):
[17:24]	8 bit Data	8 bit data
[17:32]	16 bit Data	16 bit data
[17:48]	32 bit Data	32 bit data
Last 4 bits	CRC Value	4 bit CRC field

[a]Bit 0 is the most significant bit and is transmitted first on the FSI data line

[b]all described values represent values of a positive active logic if not otherwise stated; on the FSI data line they appear inverted, i.e. a logical '0' value is associated to a HIGH level and a logical '1' value to a LOW level.

4.2.2.2. Format description: SAME_ADR

Table 4.7. Command Message Format: SAME_ADR (Same Address Commmand)

Bit [a]	Field [b]	Description
0	'1'	Start bit (very first bit of a message)
1:2	slaveID	Used to distinguish up to four cascaded FSI slaves per FSI port: “00”: slaveID=0, “01”: slaveID=1, “10”:slaveID=2, “11: slaveID=3
3:4	"11"	Identifies SAME_ADR message
5	Read/notWrite	If at '1' then read data transfer is initiated. If at '0' then write data transfer is initiated.
6:8	Address/Data Size	2-bit Address field for random access, 1-bit Data Size Type / Address bit(6) / Address bit(7) / Data Size bit(8) word / 0 / 1 / 1 halfword / addr(0) / 0 / 1 byte / addr(0) / addr(1) / 0
If bit5='0'	Write Data field:	This data field only exists if bit5='0' (write data transfer):
[9:16]	8 bit Data	8 bit data
[9:24]	16 bit Data	16 bit data
[9:40]	32 bit Data	32 bit data
Last 4 bits	CRC Value	4 bit CRC field

[a]Bit 0 is the most significant bit and is transmitted first on the FSI data line

[b]all described values represent values of a positive active logic if not otherwise stated; on the FSI data line they appear inverted, i.e. a logical '0' value is associated to a HIGH level and a logical '1' value to a LOW level.

4.2.3. Flow Control via Feedback

Flow control is a means for controlling the data transfer between engines that operate on different speeds. The completion in such a case may take longer, but as previously mentioned, the FSI slave needs to respond within a fixed portTimeOut interval.

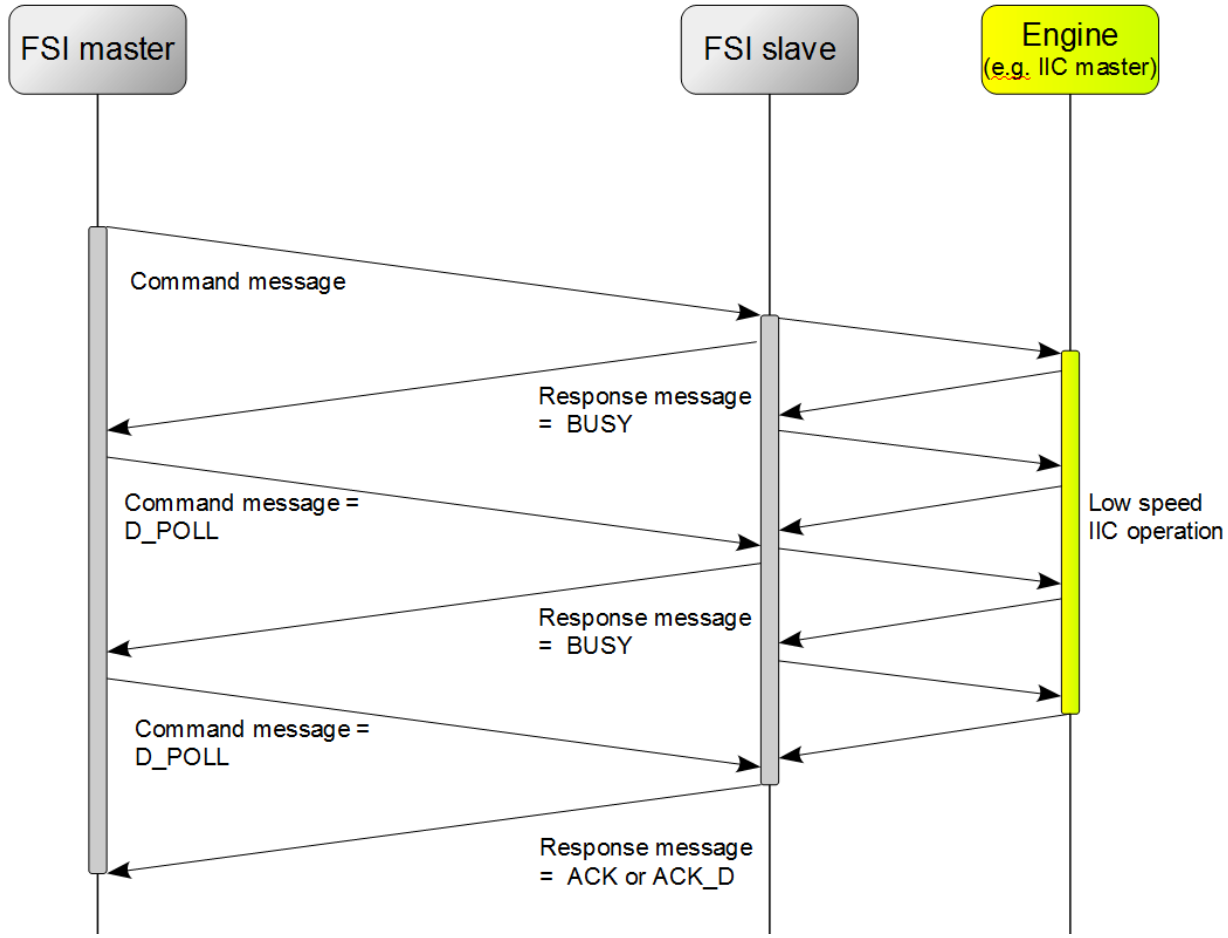
To solve that, an additional busy-indication (BUSY) as a short feedback message is introduced and the master needs to re-asserts its request upon receiving such a message by sending an explicit data poll command (D_POLL)

The following interaction diagram shows a case in which the FSI master is performing

a write or read data transfer to/from an engine behind the FSI slave that operates at a very low speed.

The FSI master issues the initial data transfer command to the FSI slave. The slave in turn forwards the command to the attached engine and helds the FSI interface busy till the requested target has completed the data transfer. As long as the FSI master gets a BUSY-response back from the FSI slave, it re-asserts its request for transferring data till the addressed target completes the transfer.

Figure 4.3. Flow Control via feedback



Should the addressed target be not able to complete the transfer within a defined time interval of about 1 ms (masterTimeOut interval) then the FSI master escapes from the “endless” BUSY_D_POLL loop by issuing a special terminate command message (TERM) instead of a D_POLL command message. That brings the FSI protocol state machine of the FSI slave back into idle state and stops further operation on the interface to attached engines.

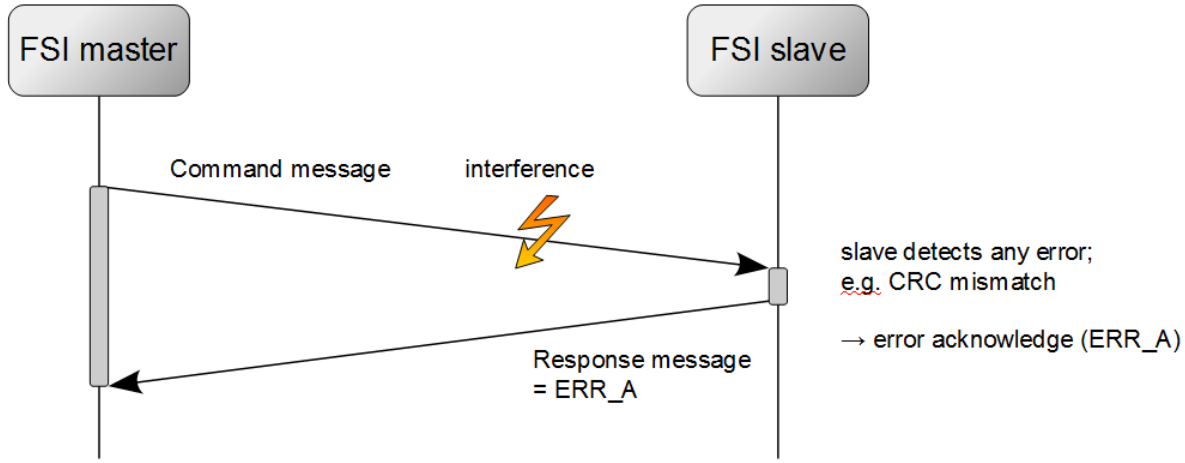
4.2.4. FSI Error Control via Feedback

To ensure error-free transmission, all message exchanges over FSI are strictly CRC protected.

For giving the FSI master the possibility to control such a case, the FSI slave needs to report any error that occurred either on the transmission line from master to slave or during the execution of the command itself on the slave side.

That is done by providing an error acknowledge (ERR_A) back to the FSI master, as depicted in the following figure.

Figure 4.4. FSI Error Control via Feedback



The ERR_A response message indicates that the FSI slave has detected a CRC error of the command message or has encountered an error in the course of executing the command.

4.2.4.1. Format description: ERR_A

Table 4.8. Response message format: ERR_A (error Achnowledge)

Bit [a]	Field [b]	Description
0	'1'	Start bit (very first bit of a message)
1:2	slaveID	Used to distinguish up to four cascaded FSI slaves per FSI port: “00”: slaveID=0, “01”: slaveID=1, “10”:slaveID=2, “11: slaveID=3
3:4	"10"	Identifies ERR_A message
5:8	CRC Value	4 bit CRC field

[a]Bit 0 is the most significant bit and is transmitted first on the FSI data line

[b]all described values represent values of a positive active logic if not otherwise stated; on the FSI data line they appear inverted, i.e. a logical '0' value is associated to a HIGH level and a logical '1' value to a LOW level.

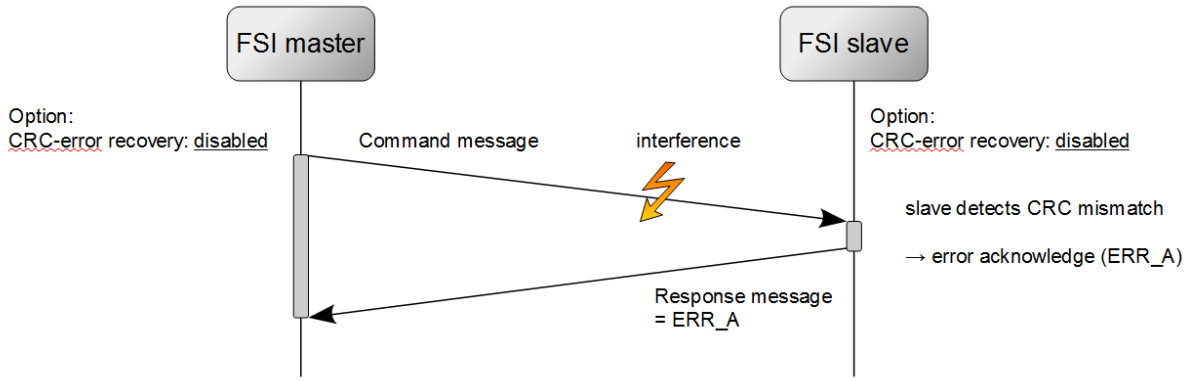
4.2.5. CRC Error Control via Feedback w/o CRC Error Recovery

The exchange of messages between master and slave is strictly CRC protected. Therefore, each side is able to detect any transmission error by calculating the expected CRC and comparing it with the received one.

The following two pictures illustrates what happens if both sides have their “hardware CRC error recovery” facility disabled:

Case 1: slave detects a transmission error

Figure 4.5. CRC error control via feedback w/o CRC error recovery: slave detection

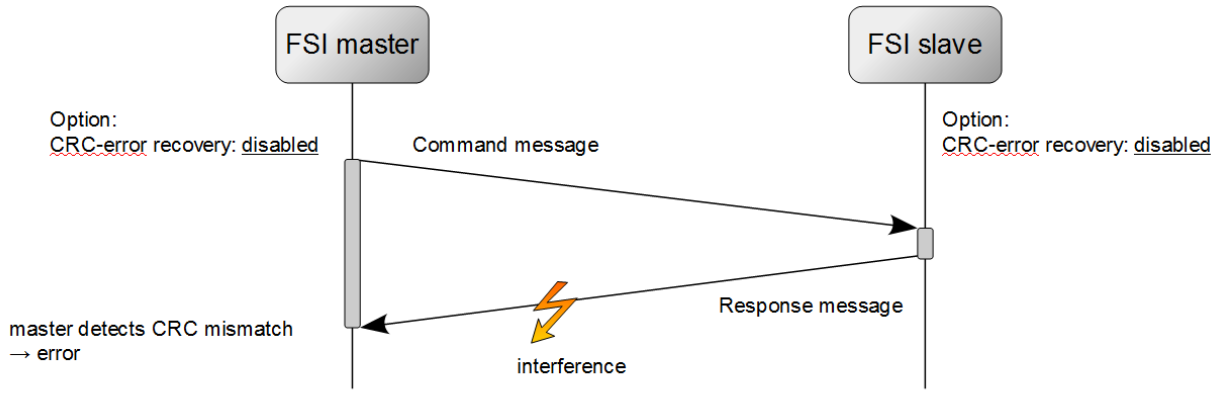


If the “CRC error recovery” option is disabled and the FSI slave detects a CRC mismatch then it needs to response with an error acknowledge, short ERR_A message.

Beside the “CRC mismatch”, the FSI slave can provide an ERR_A back to indicate other errors like protocol errors or errors that are related to the data transfer, e.g. on the interface between slave and engine).

Case 2: master detects a transmission error

Figure 4.6. CRC error control via feedback w/o CRC error recovery: master detection



In this example, the FSI slave receives an error-free command and provides an appropriate response message.

If the “CRC error recovery” option is disabled and the FSI master detects a CRC mismatch then it just raises an error, requesting recovery actions.

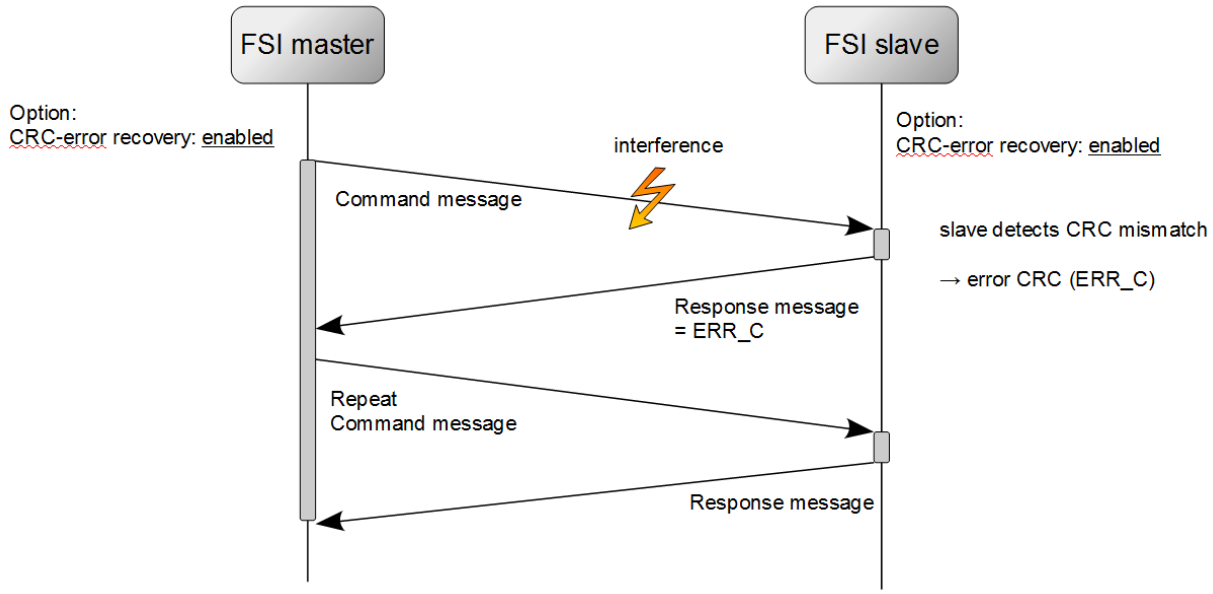
4.2.6. CRC Error Control via Feedback w/ CRC Error Recovery (Enhanced Feature)

The exchange of messages between FSI master and slave is always CRC error protected.

Each side is able to detect a transmission error by calculating the expected CRC and comparing it with the received one. Should there be a mismatch then a re-transmission (copy) is automatically requested. This is the case if the “hardware CRC error recovery” on both sides, i.e. FSI master and slave, is enabled.

Case 1: slave detects a transmission error

Figure 4.7. CRC error control via feedback w/ CRC error recovery: slave detection



If the “CRC error recovery” option is enabled and the FSI slave detects a CRC mismatch then it needs to response with an error CRC acknowledge, short ERR_C message.

It indicates that the FSI slave has received an erroneous command message.

The FSI master then re-transmits the original command message to the FSI slave.

Should this fail too, then the FSI master raises an error which ends the automatic recovery cycle.

4.2.6.1. Format description: ERR_C

Table 4.9. Response message format: ERR_C (Error CRC)

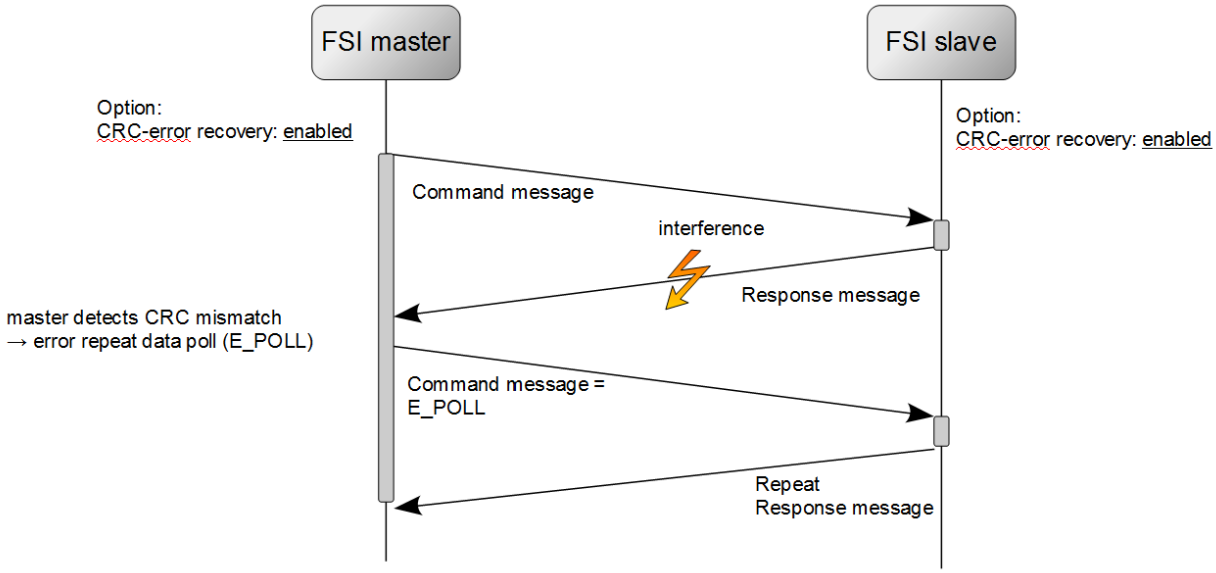
Bit [a]	Field [b]	Description
0	'1'	Start bit (very first bit of a message)
1:2	slaveID	Used to distinguish up to four cascaded FSI slaves per FSI port: “00”: slaveID=0, “01”: slaveID=1, “10”:slaveID=2, “11: slaveID=3
3:4	"11"	Identifies ERR_C message
5:8	CRC Value	4 bit CRC field

[a]Bit 0 is the most significant bit and is transmitted first on the FSI data line

[b]all described values represent values of a positive active logic if not otherwise stated; on the FSI data line they appear inverted, i.e. a logical '0' value is associated to a HIGH level and a logical '1' value to a LOW level.

Case 2: master detects a transmission error

Figure 4.8. CRC error control via feedback w/ CRC error recovery: master detection



If the “CRC error recovery” option is enabled and the FSI master detects a CRC mismatch then it request a re-transmission of the last response message by issuing an “error repeat data poll command message, short E_POLL.

Should this fail too, then the FSI master raises an error which ends the automatic recovery cycle.

- Slave receives an error-free command and acknowledges it
- Symbol: E_POLL: means that master has detected a CRC mismatch of the response and initiates a re-transmission of the slave response by sending a E_POLL command; should this fail, too, then master raises an error and error recovery cycle ends

4.2.6.2. Format description: E_POLL

Table 4.10. Command message format: E_POLL (Error Polling)

Bit [a]	Field [b]	Description
0	'1'	Start bit (very first bit of a message)
1:2	slaveID	Used to distinguish up to four cascaded FSI slaves per FSI port: “00”: slaveID=0, “01”: slaveID=1, “10”:slaveID=2, “11: slaveID=3
3:5	"011"	Identifies E_POLL message
6:9	CRC Value	4 bit CRC field

[a]Bit 0 is the most significant bit and is transmitted first on the FSI data line

[b]all described values represent values of a positive active logic if not otherwise stated; on the FSI data line they appear inverted, i.e. a logical '0' value is associated to a HIGH level and a logical '1' value to a LOW level.

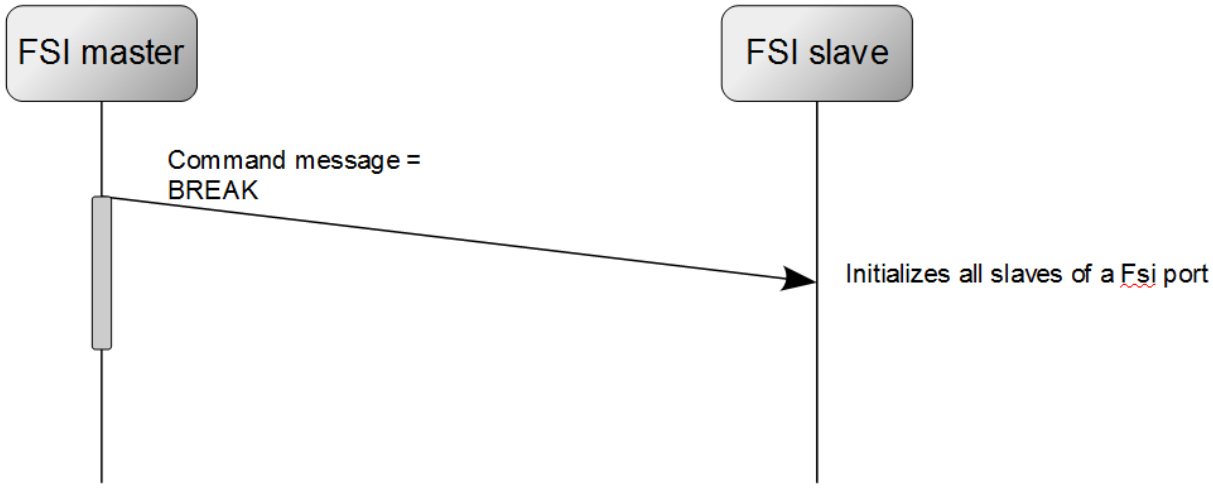
4.2.7. Special Command: BREAK

A BREAK command message is a uni-directional command sent from FSI master to slave; no response message is anticipated.

The master simply asserts the FSI data line for 256 FSI cycles (i.e. the FSI data line is pulled down on the electrical interface); this forces the slave into IDLE state and sets the default slaveID to the default value 3.

Typically, a BREAK is used as the very first command to initialize the FSI slave and is also used in some error scenarios in which the slave must get re-synchronized (in protocol error cases, for example).

Figure 4.9. Special command BREAK



4.2.7.1. Format description: BREAK

Table 4.11. Command message format: BREAK (Breaking)

Bit [a]	Field [b]	Description
0:255	"1111...11"	Identifies BREAK message

[a]Bit 0 is the most significant bit and is transmitted first on the FSI data line

[b]all described values represent values of a positive active logic if not otherwise stated; on the FSI data line they appear inverted, i.e. a logical '0' value is associated to a HIGH level and a logical '1' value to a LOW level.

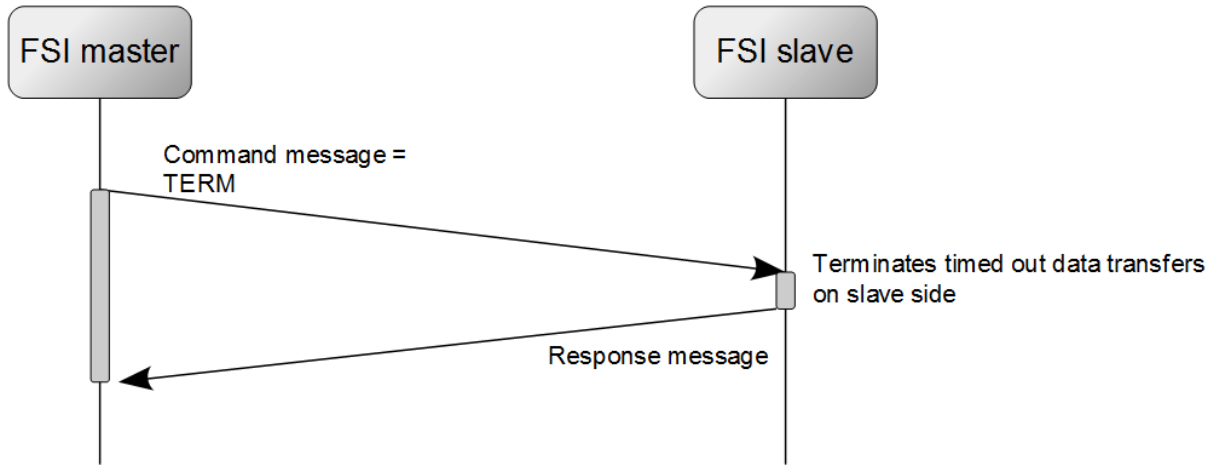
4.2.8. Special Command: TERM

TERM, short for terminate, is a normal command which anticipates a response from the slave.

The slave uses this command message to reset the state machine which controls the access to the attached engines. In addition, the slave freezes its internal error register for debugging purposes.

This command is also needed to abort any ongoing operation in case of an expired “MasterTimeOut” timer.

Figure 4.10. Special command TERM



4.2.8.1. Format description: TERM

Table 4.12. Command message format: E_POLL (Error Polling)

Bit [a]	Field [b]	Description
0	'1'	Start bit (very first bit of a message)
1:2	slaveID	Used to distinguish up to four cascaded FSI slaves per FSI port: “00”: slaveID=0, “01”: slaveID=1, “10”:slaveID=2, “11: slaveID=3
3:8	"111111"	Identifies TERM message
9:12	CRC Value	4 bit CRC field

[a]Bit 0 is the most significant bit and is transmitted first on the FSI data line

[b]all described values represent values of a positive active logic if not otherwise stated; on the FSI data line they appear inverted, i.e. a logical '0' value is associated to a HIGH level and a logical '1' value to a LOW level.

4.2.9. Interrupt Polling

The FSI protocol supports an interrupt and "alive" checking of the FSI slave through special Interrupt Polling (I_POLL) commands issued by a port controller of a FSI master; that is done by hardware; nofirmware intervention is required.

The collected interrupt pulses of all available FSI port controllers are passed on to a universal interrupt controller through an asynchronous signal-vector for further processing.

Once a FSI port is enabled, the associated port controller of the FSI master automatically issues I_POLL commands to the FSI slave and the slave provides its interrupt status back to the master through I_POLL_RSP response message.

The frequency of the issued I_POLL is determined by the port controller.

If there is no data transfer between master and slave then the port controller of the master issues I_POLL commands back-to-back (see first figure below).

In the case of a back-to-back data transfer between FSI master and slave, the port controller ensures that at least one I_POLL is issued between two successive data transfers (see second figure below).

Two interrupt levels/priorities are supported, namely INTR1 and INTR2 level.

Additional to the interrupt bits, the I_POLL_RSP contains also a 3-bit field for controlling hardware initiated (device paced) DMA transfers. Through this field, any FIFO device on the slave side is able to control any DMA device on the master side for memory-to-memory transfers.

Up to 2 DMA channels can be supported at a time per slave.

With a DMA request signaling, a FIFO device can initiate a DMA transfer and

End of DMA transfer (EOT) can be used to end a DMA transfer (equivalent to terminal count mechanism).

I_POLL issue rate for FSI CLK=166 MHz is typically about one I_POLL per 768 ns.

Figure 4.11. Interrupt polling

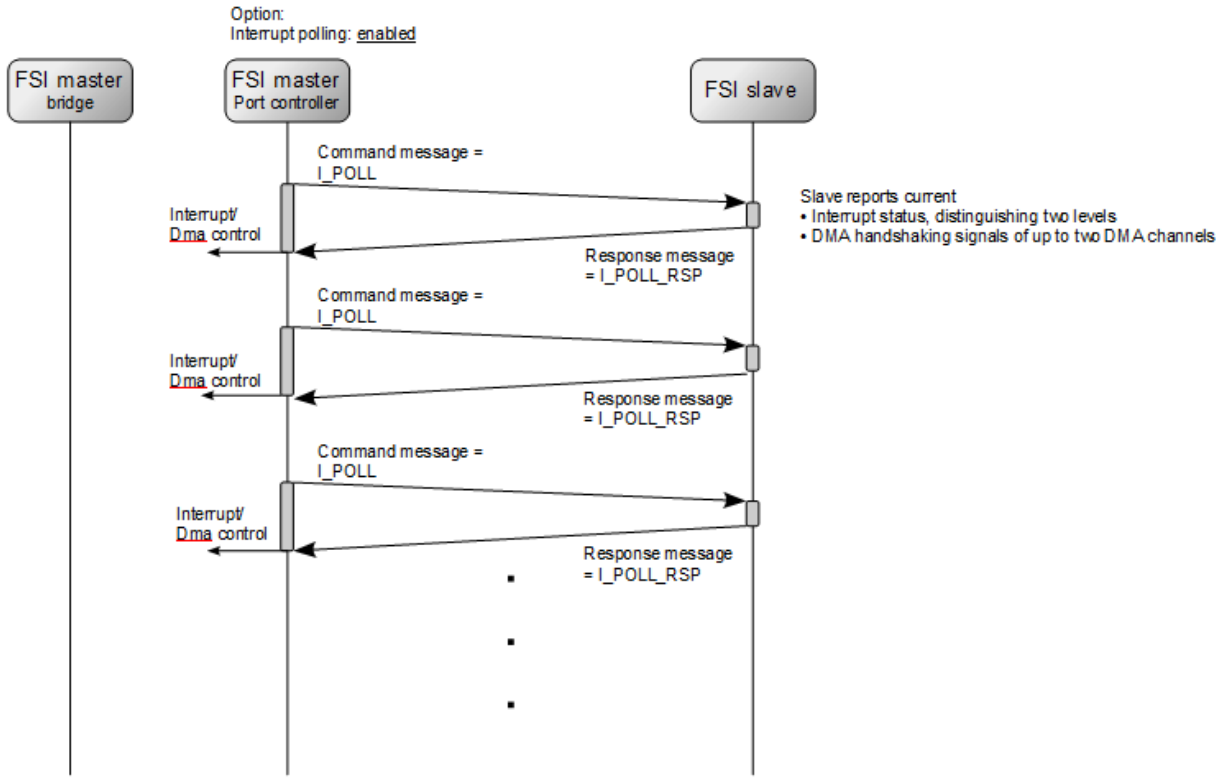
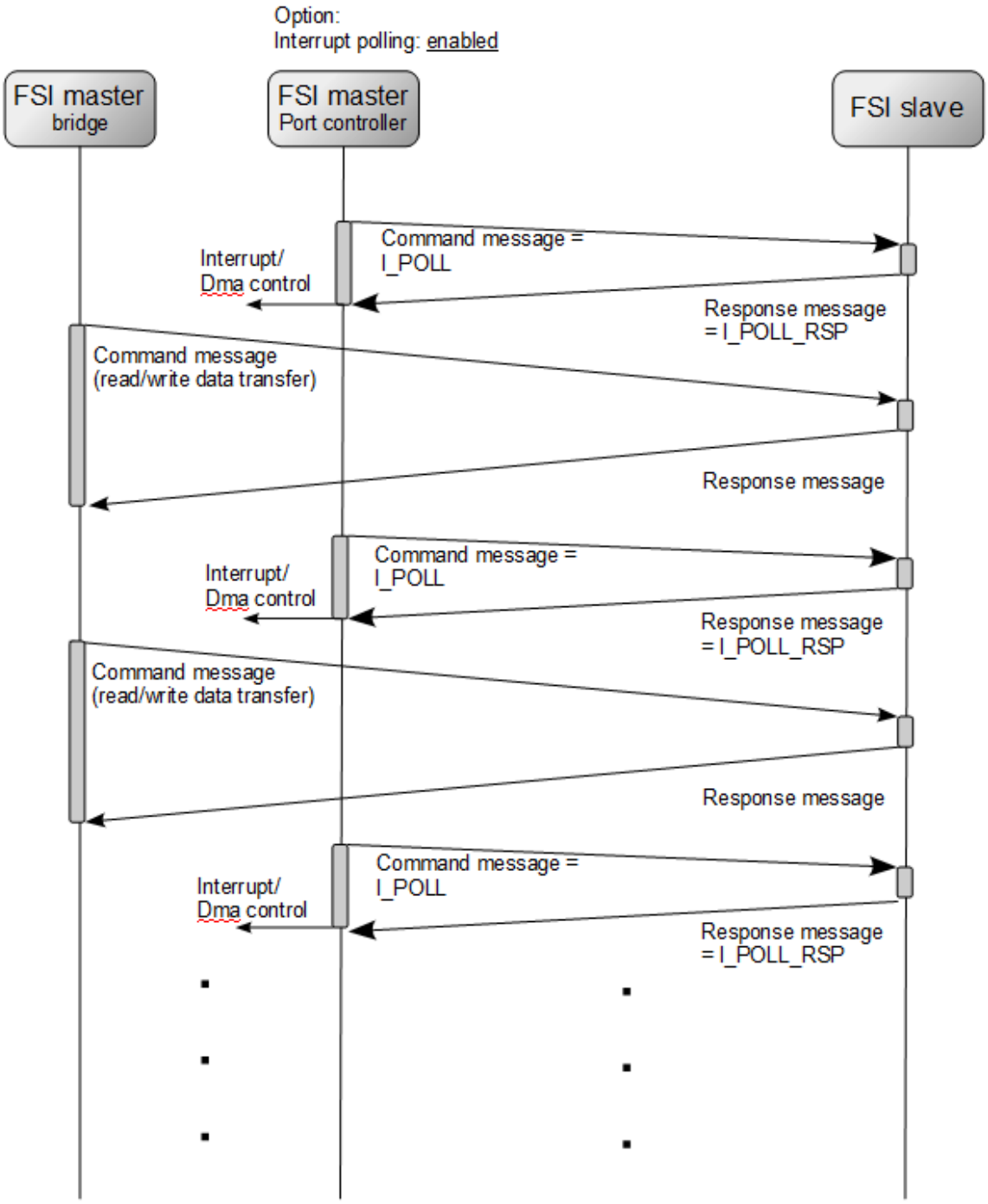


Figure 4.12. Interrupt polling



4.2.9.1. Format description: I_POLL_RSP (Interrupt Poll Response)

Table 4.13. Command message format: E_POLL_RSP (Error Polling Response)

Bit [a]	Field [b]	Description
0	'1'	Start bit (very first bit of a message)

Bit [a]	Field [b]	Description
1:2	slaveID	Used to distinguish up to four cascaded FSI slaves per FSI port: “00”: slaveID=0, “01”: slaveID=1, “10”:slaveID=2, “11: slaveID=3
3	"0"	Identifies I_POLL_RSP message
4;5	Interrupt field	Two different interrupt levels are supported: Bit4: interrupt 1 level, bit5: interrupt 2 level
6:8	DMA control field	3-bit field for controlling FIFO device paced DMA transfers. Encoding: “000”: “001”: dma_request1 “010”: dma_request2 “011”: dma_request1, dma_request2 “100”: dma_endOfTransfer1 “101”: dma_endOfTransfer2 “110”: dma_endOfTransfer1, dma_request2 “111”: dma_request1, dma_endOfTransfer2
9:12	CRC Value	4 bit CRC field

[a]Bit 0 is the most significant bit and is transmitted first on the FSI data line

[b]all described values represent values of a positive active logic if not otherwise stated; on the FSI data line they appear inverted, i.e. a logical '0' value is associated to a HIGH level and a logical '1' value to a LOW level.

4.2.10. CRC Generator / Checker

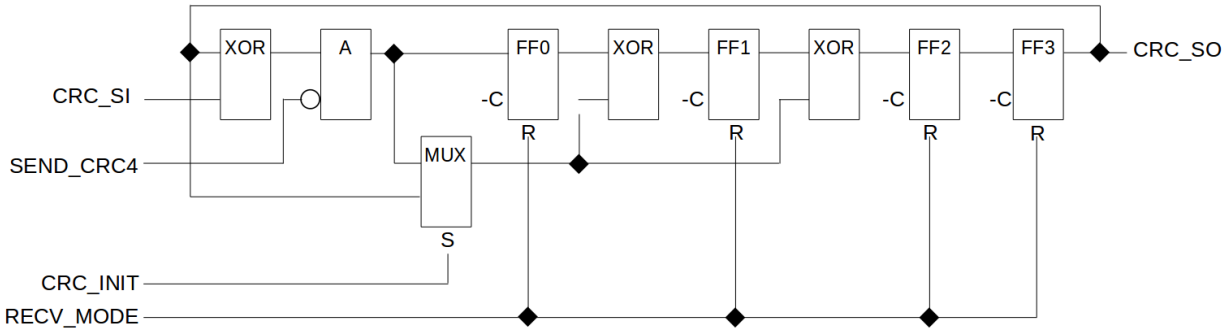
The 4 bit Cyclic Redundancy Code (CRC) field is part of each message, except for BREAK command message) and protects the transmission against interference.

For calculating the CRC value of a message, all bit are considered, including the start bit.

The polynomial is a fourth order type: x^4+x^2+x+1.

It protects all single, double and odd number of errors as well as burst errors with a length of up to four bits.

Figure 4.13. CRC Generator/Checker



4.2.11. Overview: FSI Protocol Messages

The following table summarized the messages with respect to protocol features.

The message sequence shall be in accordance with the message exchange rules as defined in the previous section.

Example: for transferring data between FSI master and slave, the FSI master needs to support an absolute address command for initiating a data transfer (ABS_ADR), plus a data poll command for continuing a data transfer request (D_POLL) in case of different operating speeds. The FSI slave needs to respond with an acknowledge (ACK) to indicate completion of a write data transfer or an acknowledge with appended data (ACK_D) to indicate completion of a read data transfer. Additionally, the FSI slave needs to provide a busy (BUSY) if data transfer is not yet completed or an error acknowledge (ERR_A) to indicate an encountered error condition during the transfer.

Grouped FSI Protocol Messages

Command message: BREAK *Response message: not defined*	used for slave initialization
Command message: ABS_ADR , D_POLL Response message: ACK, ACK_D , ERR_A , BUSY	used for data transfers (read, write)
Command message: TERM Response message: ACK , ERR_A, BUSY	used for terminate/abort case
Command message: SAME_ADR , REL_ADR Response message: ACK, ACK_D , ERR_A , BUSY	Optional: used for relative addressing
Command message: E_POLL Response message: ERR_C	Optional: used for hardware crc error recovery
Command message: I_POLL Response message: I_POLL_RSP	Optional: used for interrupt and DMA request/endOfTime polling

4.2.12. Bit Representation of the Protocol Messages

Figure 4.14, “FSI master command bit representation” [28]andFigure 4.15, “FSI slave response bit representation” [29] show the bit representation of the different FSI protocol messages. The defined bit sequences are positive logic. They have to appear inverted on the FSI data line (negative active).

Bit 0 represents the start bit; start bit is part of CRC.

Figure 4.14. FSI master command bit representation

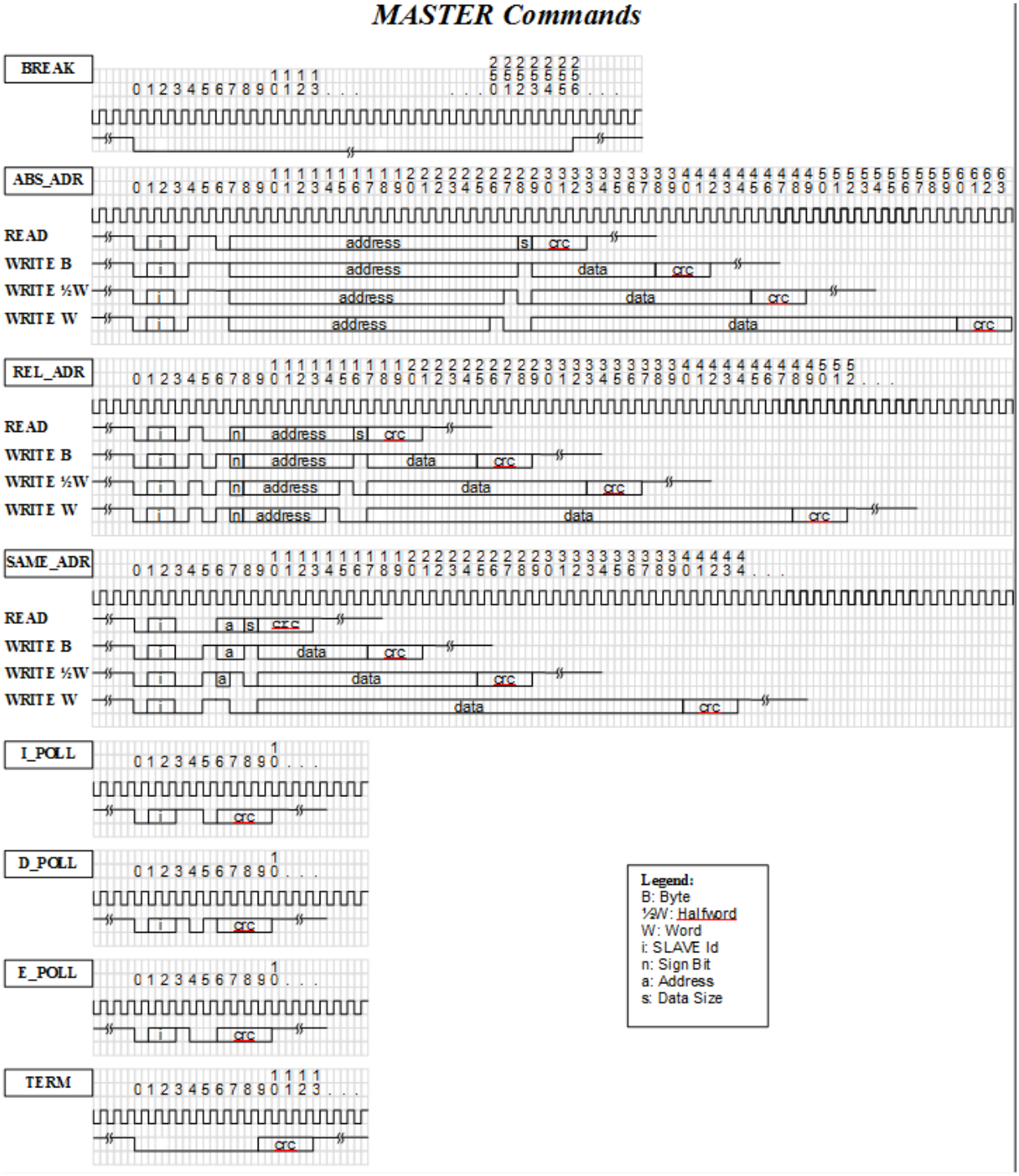
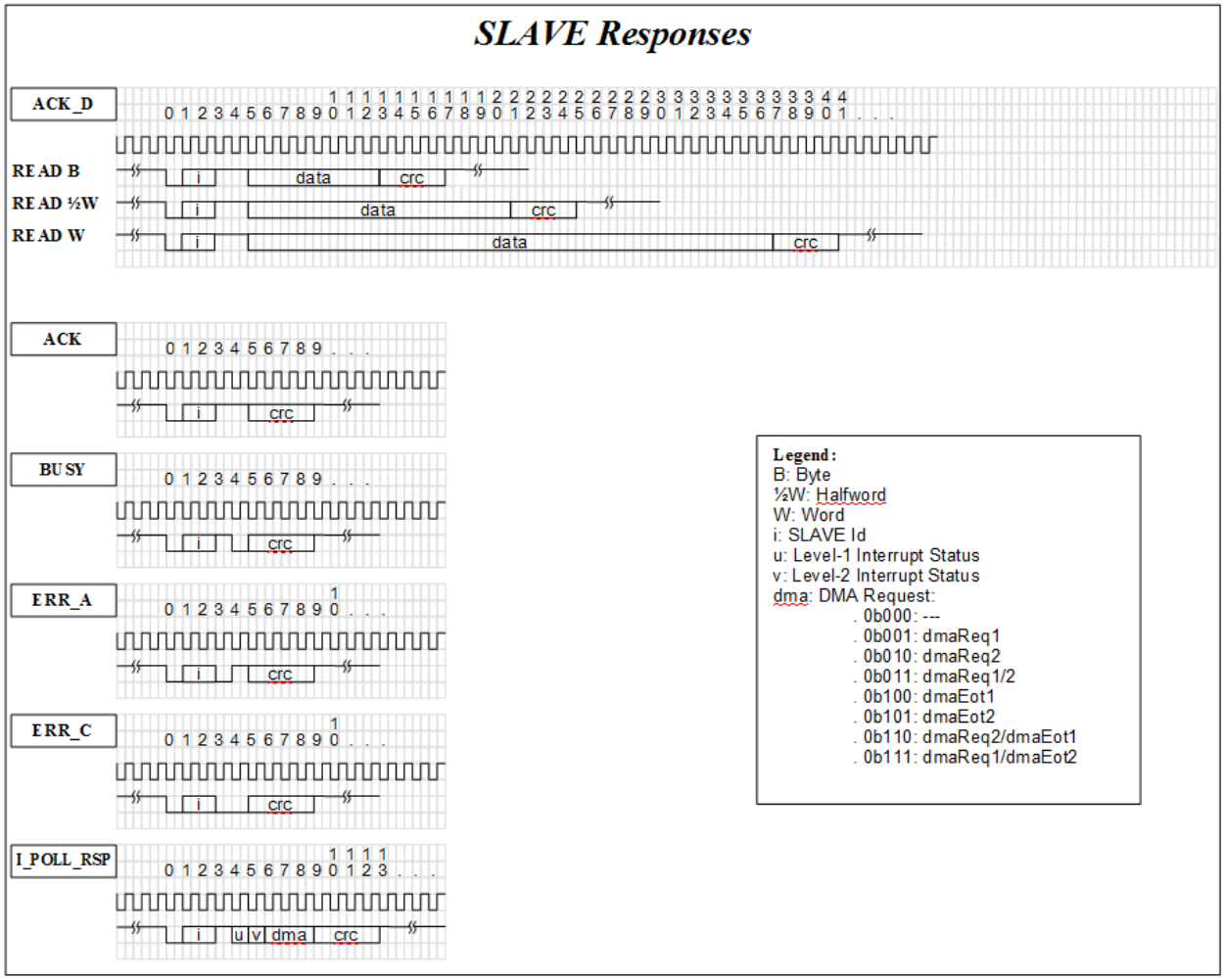


Figure 4.15. FSI slave response bit representation



5. FSI Electrical Characteristics

This section provides an overview of the electrical characteristics of FSI. The specification of the electrical characteristics like timing and voltage levels are following in the next section.

5.1. Principle of Operation

FSI is a point to point two wire interface which is capable of supporting distances of up to 4 meters at 166MHz bus frequency. It is a serial data link that operates in half duplex mode. Devices communicate in master/slave mode where the master device initiates a data transfer. Data is released on the master side with the falling clock edge and it is sampled on the slave side with the rising clock.

FSI supports static plug detection at the FSI master side through DC level sensing of the data line and permanent presence detection through idle poll requests initiated by the FSI master.

5.2. FSI signals

FSI is a 2-wire interface consisting of

fsi_clock:	uni-directional signal, master output, max. frequency 166MHz, continuously running
fsi_data:	bi-directional signal, bias resistors on master and slave side for static plug detection

5.3. Base electrical specifications

FSI IO supply voltage:	FSI signals are based on 1.2V IO supply signal levels. This allows the integration of FSI slave or master functions into processor or custom logic chips. Noise margin is reduced compared to interfaces with 3.3V signal levels, coupled noise on FSI signals shall be less than 100mV therefore. Careful design review is suggested if FSI slave and master are operating at different IO voltage levels (i.e. slave at 1.2V and master at 1.1V).
FSI bus speeds:	FSI as specified in this document is capable of supporting distances of up to 4 meters at 166MHz bus frequency. This can be achieved by following simple design/wiring rules provided in this document and is based on the point to point signal routing. If the characteristics of the signal path don't allow operation at 166MHz or if power consumption has to be reduced then the FSI master supports reduced clock frequencies by factors of 2. The FSI master is also capable to run individual FSI ports at 2 different FSI clock frequencies. This allows parallel usage of higher speed interfaces for critical usages and moderate speeds for general purpose. While higher and lower bus speeds are possible depending on environment and requirements, this document is defining parameters only for 166MHz fast and 41MHz moderate clock speed settings.
power consumption:	FSI is a low power interface using simple IO's without far-end termination. Interface power consumption is depending on bus speed and bias resistors for static plug detection. The FSI slave implementation is often operating from standby voltage supply levels allowing service access without system being fully operational. To allow simple designs for fault isolation and hot plug, power consumption for the FSI slave implementation has to be limited to 100mW max.
signal rise/fall times:	FSI can be operated at high clock speeds requiring fast signal rise/fall times. Max. FSI signal rise/fall times shall not exceed 2V/ns to minimize electromagnetic interference.

5.4. FSI Design/Wiring Recommendations

FSI is a simple interface which can be operated at 166MHz clock frequency. This can be achieved without problems by considering the design/wiring recommendations provided below.

wiring impedance:	wiring impedance of FSI clock and data signals on printed circuit boards and chip carriers shall be **50 Ω**to GND (= signal return path). Deviation from this rule has to be kept under close control.
driver impedance:	driver impedance for FSI clock and data signals shall be **40 Ω**to provide the required rise/fall times for 166MHz operation on longer nets across multiple packaging units. Expect signal over/ undershoot of 200mV with 40Ω driver and 50Ω wiring impedance.
wire length:	depending on net quality and capacitive loading, FSI is capable of supporting distances of **up to 4 meters**at 166MHz bus frequency.
wire stub length:	FSI consists of point to point connections, no stubs are present on FSI nets therefore, except for via stubs needed for signal routing across printed circuit boards. If bias resistors on printed circuit boards are used for static plug/presence detetection then these should be connected using wire stubs **shorter than 10mm**.
skew between clock and data:	for 166MHz operation, output skew between FSI clock and data of a FSI signal pair is allowed to be ±300ps at the FSI master output. Input skew at the FSI slave input has to be controlled to provide the required setup and hold times. The combined input skew between FSI clock and data of a FSI signal pair, including output and wiring skew, has to be limited to ±1500ps for 166MHz operation. To simplify card designs the following rule of thumb can be applied: wiring skew between FSI clock and data of a FSI signal pair has to be less than 10mm for each packaging unit (card, module).
signal reference, **return path:**	logic GND shall be used as reference for FSI signals. Return path continuity has to be controlled to minimize signal distortion and noise coupling/generation.
level shifter **for FSI signals:**	it's **not recommended**to use voltage level translators/shifters in the FSI signal path. Problems with signal integrity have to be expected because of the bi-directional FSI data signal characteristic.
EMI considerations:	FSI can be operated at high clock speeds requiring fast signal rise/fall times. Careful signal and return path routing has to be established to minimize electromagnetic interference (EMI). **signal rise/fall times** To limit electromagnetic interference through radiated and coupled noise caused by FSI, the max. FSI signal rise/fall times shall be below 2V/ns. Use this value for cross talk analysis and for EMI assessment. **driver impedance** FSI signals are near end terminated only. This allows the use of simple IO's and minimizes power consumption on FSI nets. Signals are reflected at the receiver and are traveling back to the driver. Driver impedance has to be close to wiring impedance to terminate the reflections at the driver (near end). **coupled noise** FSI is based on 1.2V signal levels. This allows the integration of FSI slave or master functions into processor or custom logic chips. But noise margin is reduced compared to interfaces with 3.3V signal levels. Coupled noise on FSI signals shall be less than 100mV therefore. Sufficient separation (=line to line spacing) between FSI signals (even between same clock/data pair) and other signals is required.

5.5. 4.5 Net Impedance and Terminations

FSI nets are near end (driver side) terminated only. Driver impedance and net impedance have to be controlled to provide the required characteristics for 166MHz operation and to terminate reflections travelling back from the far end (receiver side).

Bias networks on FSI data nets (weak pull down resistor at FSI master side, strong pull up resistor at FSI slave side) are required for presence detection and consume some of the low level noise margin.

The pictures below show typical implementation examples.

Figure 5.1. FSI net with chip terminations

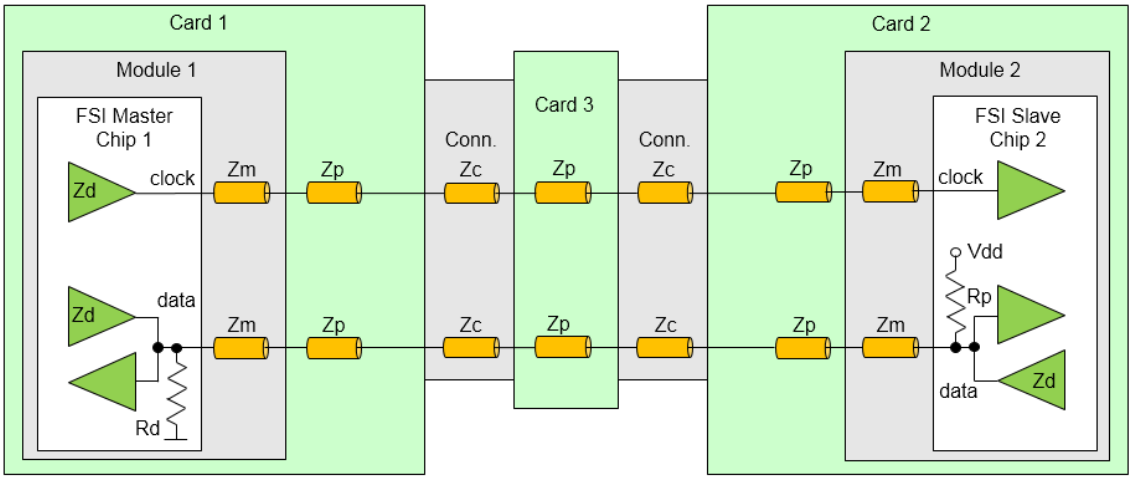


Figure 5.2. FSI net with card terminations

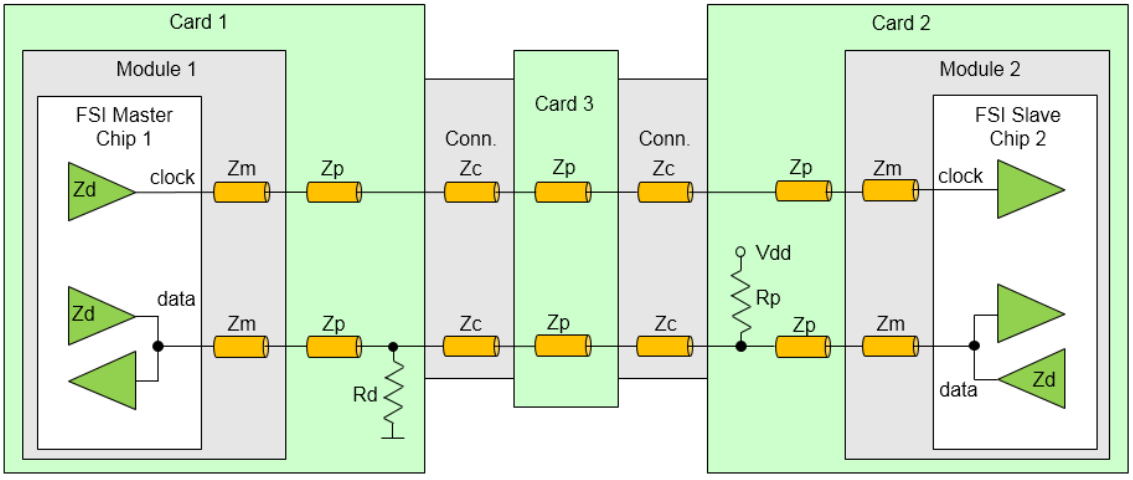


Table 5.1. Typical impedance and bias resistor values

Zd	Zm	Zp	Rd	Rp
40Ω	50Ω	50Ω	10kΩ	750Ω

6. Electrical Specifications

Table 6.1. FSI Electrical Specifications

Symbol	Parameter	Conditions	min	nom	max	Unit
V_{IO}	IO supply voltage		1.0	1.2	1.3	V
V_{IH}	high-level input voltage		0.8		1.5	V
V_{IL}	low-level input voltage		-0.3		0.4	V
V_{OH}	high-level output voltage	I_O= 2mA	V_{IO}- 0.1		V_{IO}	V
V_{OL}	low-level output voltage	I_O= -2mA	0		V_{IO}+ 0.1	V
V_{hys}	input hysteresis		30	100	150	mV
Z_D	driver output impedance		32	40	48	Ω
R_D	FSI data master side pull down bias resistor		7k	10k	15k	Ω
R_P	FSI data slave side pull up bias resistor		600	750	1500	Ω
f_{clk}	FSI clock frequency		1		166	MHz
T_{clk}	FSI clock cycle time	f_{clk}= 166MHz		6.0		ns
t_h	FSI clock high time	f_{clk}= 166MHz	1.5		2.7	ns
t_r, t_f	signal rise/fall time of FSI clk, data	f_{clk}= 166MHz	0.3		1.5	ns
t_{os}	FSI clk/data output skew	f_{clk}= 166MHz	-300		300	ps
t_{is}	FSI clk/data input skew	f_{clk}= 166MHz	-1.5		1.5	ns
t_{su}	set up time	f_{clk}= 166MHz	0.6			ns
t_{hd}	hold time	f_{clk}= 166MHz	0.6			ns
T_{clk}	FSI clock cycle time	f_{clk}= 41MHz		24.0		ns
t_h	FSI clock high time	f_{clk}= 41MHz	7.0		11.7	ns
t_r, t_f	signal rise/fall time of FSI clk, data	f_{clk}= 41MHz	0.3		5.0	ns
t_{os}	FSI clk/data output skew	f_{clk}= 41MHz	-500		500	ps
t_{is}	FSI clk/data input skew	f_{clk}= 41MHz	-5.0		5.0	ns
t_{su}	set up time	f_{clk}= 41MHz	2.0			ns
t_{hd}	hold time	f_{clk}= 41MHz	2.0			ns

Figure 6.1. FSI Master output timing

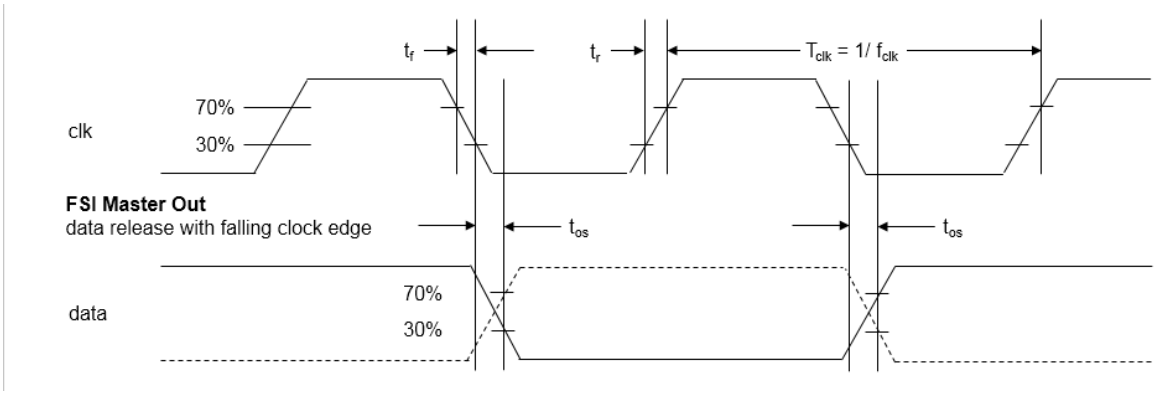
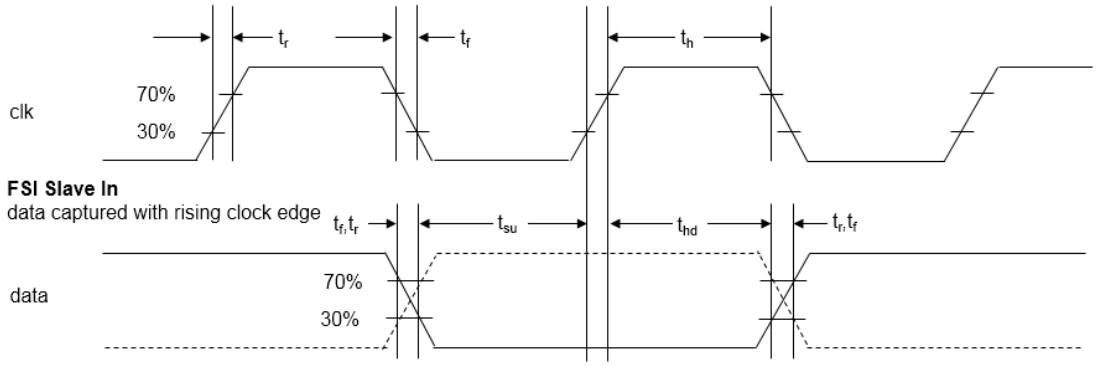


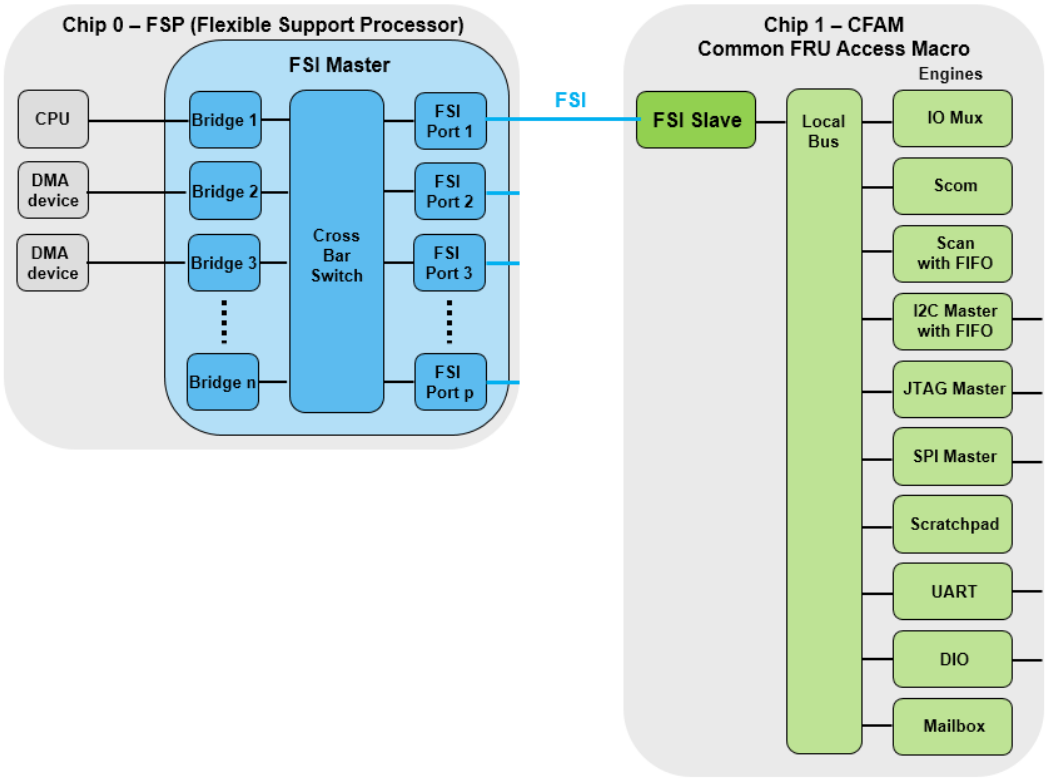
Figure 6.2. FSI Slave input timing



7. Application Examples

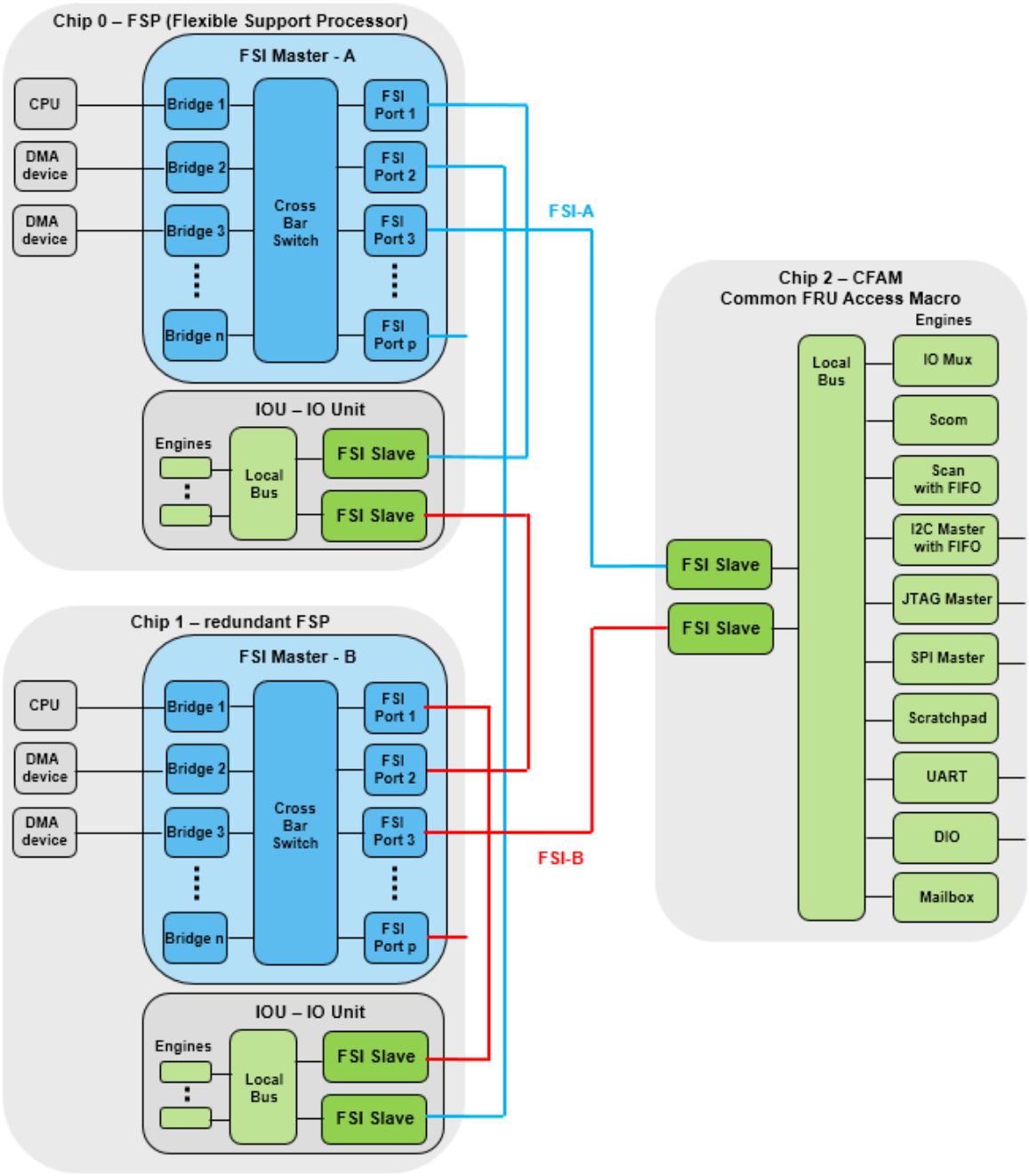
7.1. FSI Master/Slave Implementation Example

Figure 7.1. FSI Slave input timing



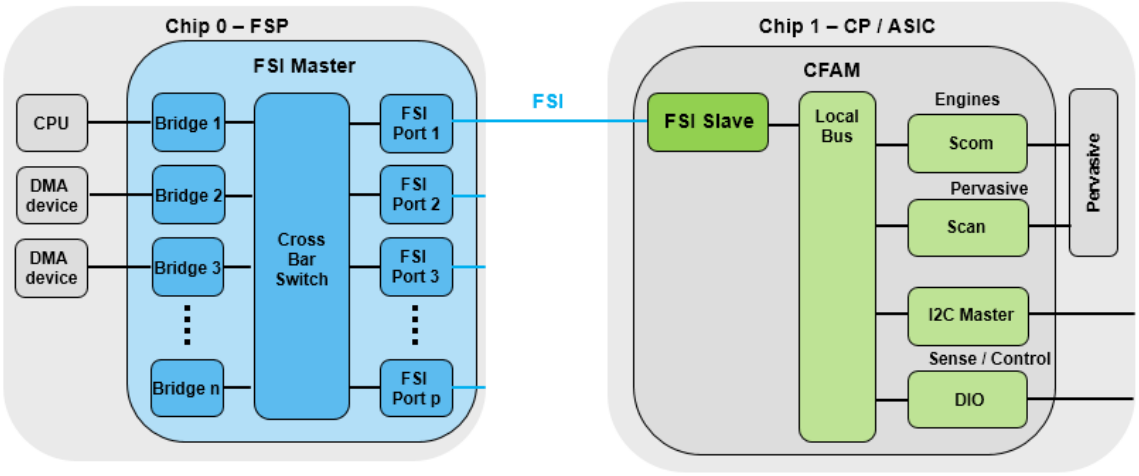
7.2. Redundant FSI Master/Slave Implementation Example

Figure 7.2. Redundant FSI Master/Slave Implementation Example



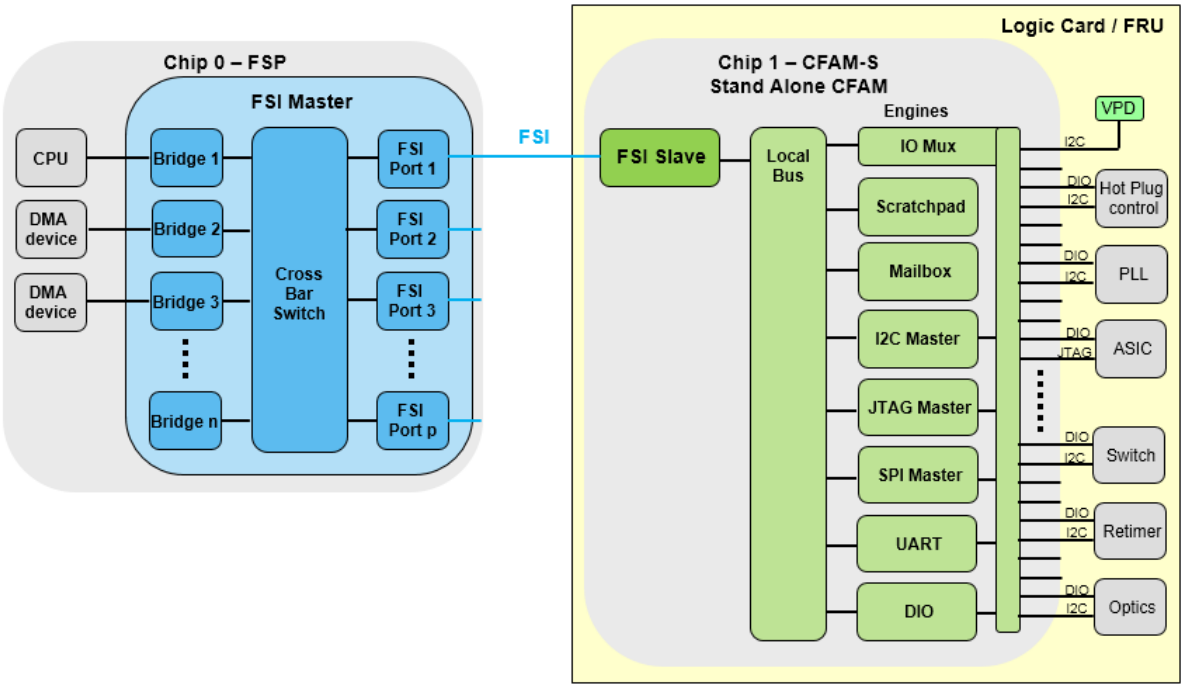
7.3. FSI Slave implemented in ASIC

Figure 7.3. FSI Slave implemented in ASIC

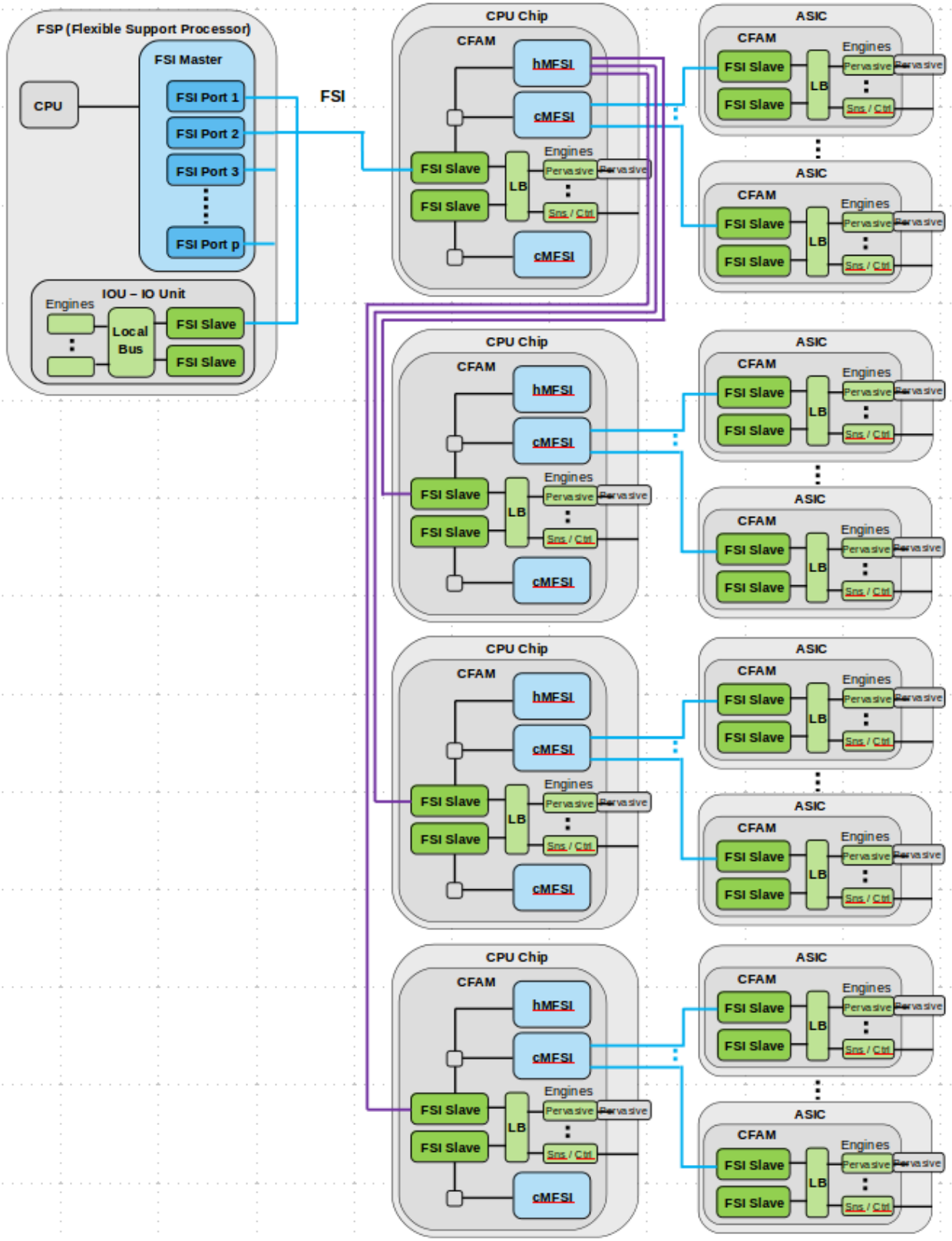


7.4. FSI Slave implemented as Stand Alone Version

Figure 7.4. FSI Slave implemented as Stand Alone Version

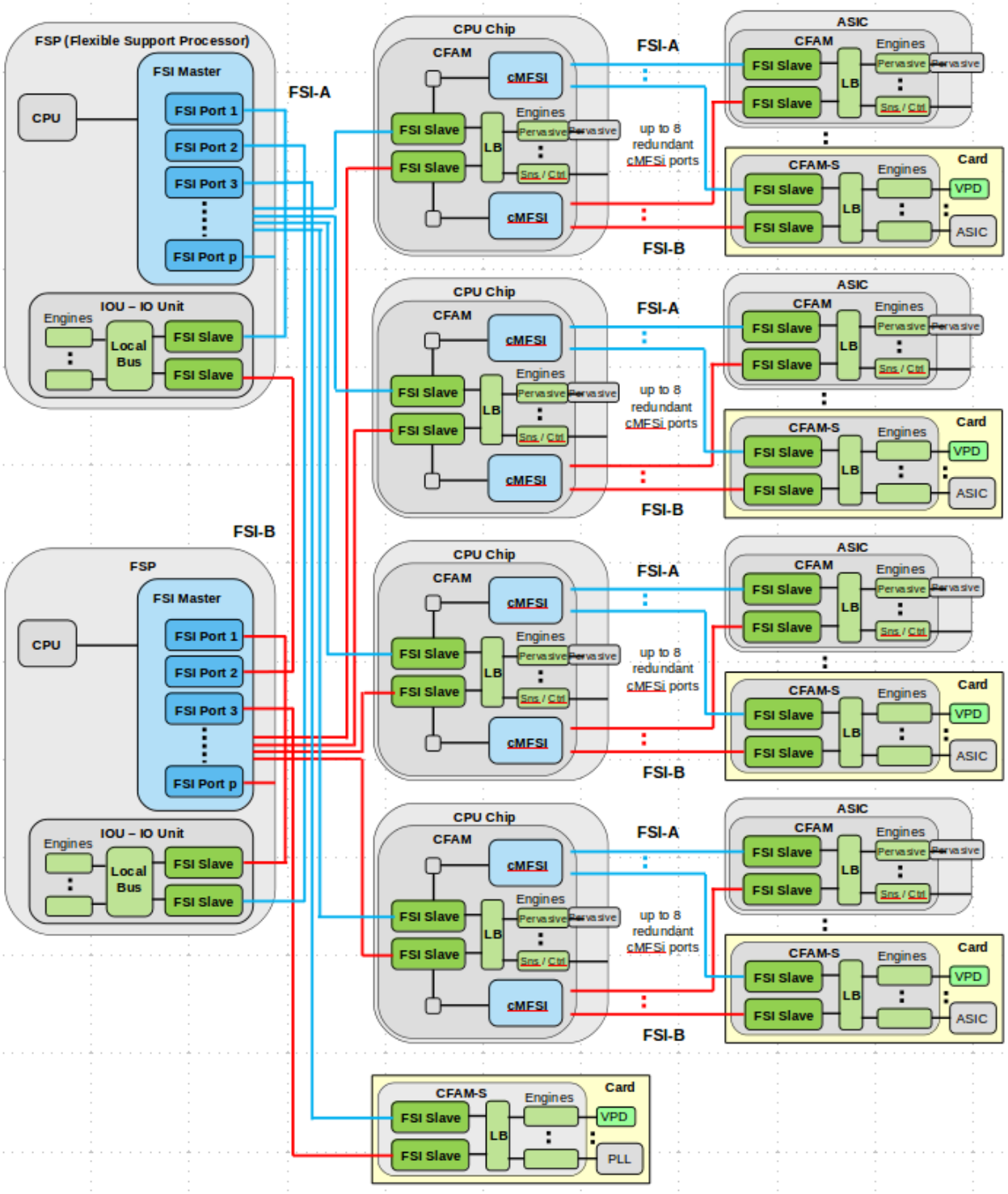


7.5. Low End Server Example with FSI for Drawer Control



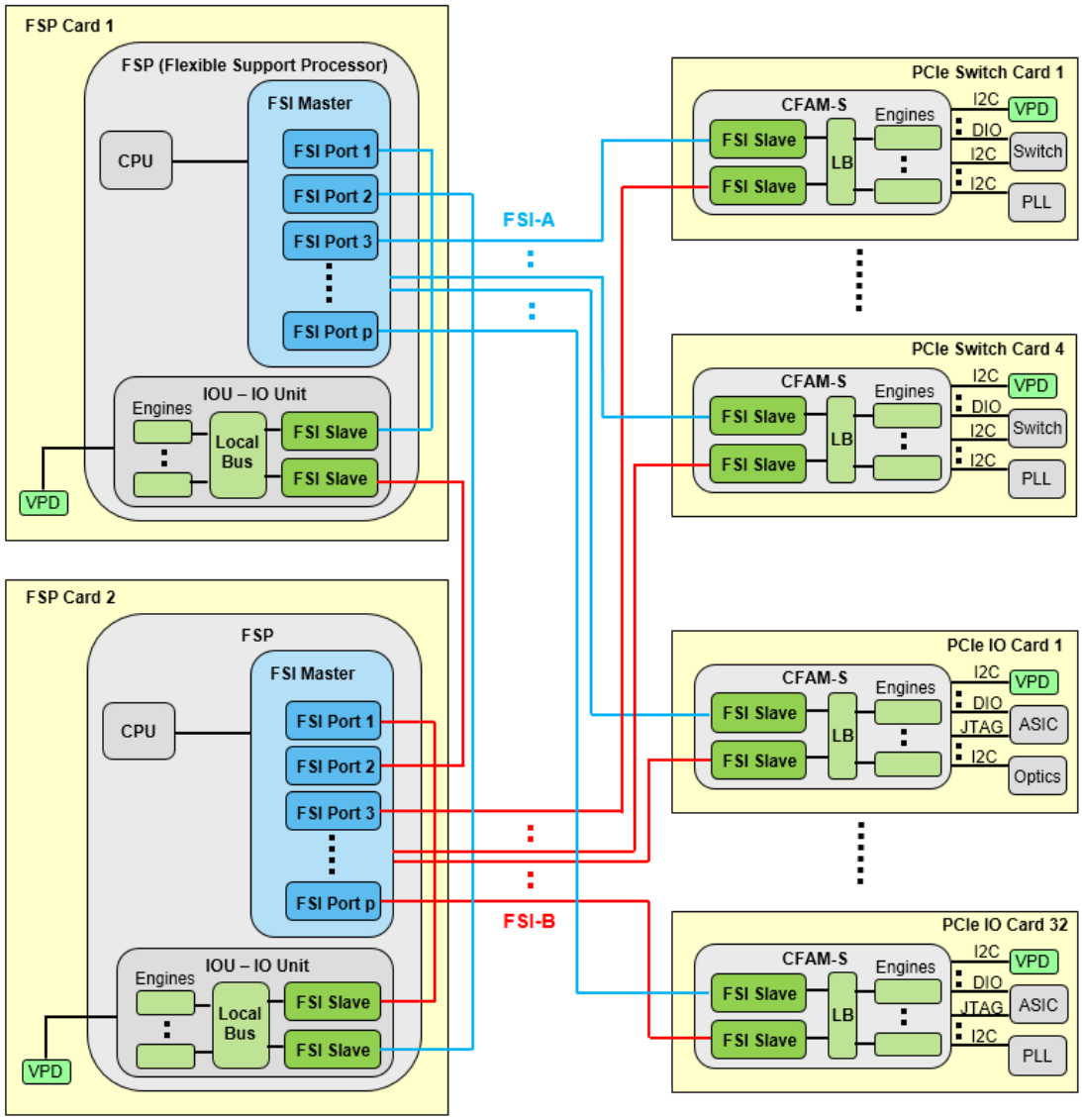
7.6. High End Server Example with Redundant FSI for Drawer Control

Figure 7.6. High End Server example with redundant FSI for drawer control



7.7. IO Drawer Example with Redundant FSI for Drawer Control

Figure 7.7. IO Drawer Example with Redundant FSI for Drawer Control



Appendix A. OpenPOWER Foundation overview

The OpenPOWER Foundation was founded in 2013 as an open technical membership organization that will enable data centers to rethink their approach to technology. Member companies are enabled to customize POWER CPU processors and system platforms for optimization and innovation for their business needs. These innovations include custom systems for large or warehouse scale data centers, workload acceleration through GPU, FPGA or advanced I/O, platform optimization for SW appliances, or advanced hardware technology exploitation. OpenPOWER members are actively pursing all of these innovations and more and welcome all parties to join in moving the state of the art of OpenPOWER systems design forward.

To learn more about the OpenPOWER Foundation, visit the organization website at openpowerfoundation.org.

A.1. Foundation documentation

Key foundation documents include:

- *Bylaws of OpenPOWER Foundation*
- *OpenPOWER Foundation Intellectual Property Rights (IPR) Policy*
- *OpenPOWER Foundation Membership Agreement*
- *OpenPOWER Anti-Trust Guidelines*

More information about the foundation governance can be found at openpowerfoundation.org/about-us/governance.

A.2. Technical resources

Development resouces fall into the following general categories:

- Foundation work groups
- Remote development environments (VMs)
- Development systems
- Technical specifications
- Software
- Developer tools

The complete list of technical resources are maintained on the foundation Technical Resources web page.

A.3. Contact the foundation

To learn more about the OpenPOWER Foundation, please use the following contact points:

- General information -- <info@openpowerfoundation.org>
- Membership -- <membership@openpowerfoundation.org>
- Technical Work Groups and projects -- <tsc-chair@openpowerfoundation.org>
- Events and other activities -- <admin@openpowerfoundation.org>
- Press/Analysts -- <press@openpowerfoundation.org>

More contact information can be found at openpowerfoundation.org/get-involved/contact-us.