TOTAL PHASE

Promira Serial Platform eSPI Analyzer User Manual

Total Phase : User Manuals : Hardware

Promira Serial Platform eSPI Analyzer User Manual

1 Revision History

Top

1.1 Changes in version 1.10

Added eSPI advanced/simple match triggers, hardware filters, and hardware statistics features.

Fixed Pin Description Table typo. IO1 signal is pin 5, and IO0 signal is pin 5.

Top

1.2 Changes in version 1.00

Initial revision.

2 General Overview

The Promira Serial Platform with eSPI Analysis Application non-intrusively monitors eSPI bus at up to 66 MHz bit rate in single, dual and quad IO modes. The Promira platform with eSPI analysis application supports two CS signals, two Reset signals, two Alert signals, and 11 Digital IO signals. The Promira platform connects to an analysis computer via Ethernet or Ethernet over USB. The application installed on the Promira platform is field-upgradeable and future-proof.

Top

2.1 eSPI Background

Super-I/O (SIO) and Port-80 debug card.

The eSPI has been specified by Intel as a replacement for the existing Intel Low Pin Count (LPC) interface on current server and client platforms. LPC bus is a legacy bus developed as the replacement for Industry Standard Architecture (ISA) bus. Some LPC bus limitations, which led to the development of eSPI, are:

- LPC requires up to 13 pins, of which 7 are required and 6 are optional.
- Current LPC implementations include a fabrication process cost burden as it is based on 3.3V IO signaling technology.
- The LPC bus clock frequency is fixed at 33 MHz that fixed the bandwidth at 133 Mbps.
- The LPC has a significant number of sideband signals.

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 a different protocol to meet a set of different requirements.

Comparing eSPI and SPI

Top

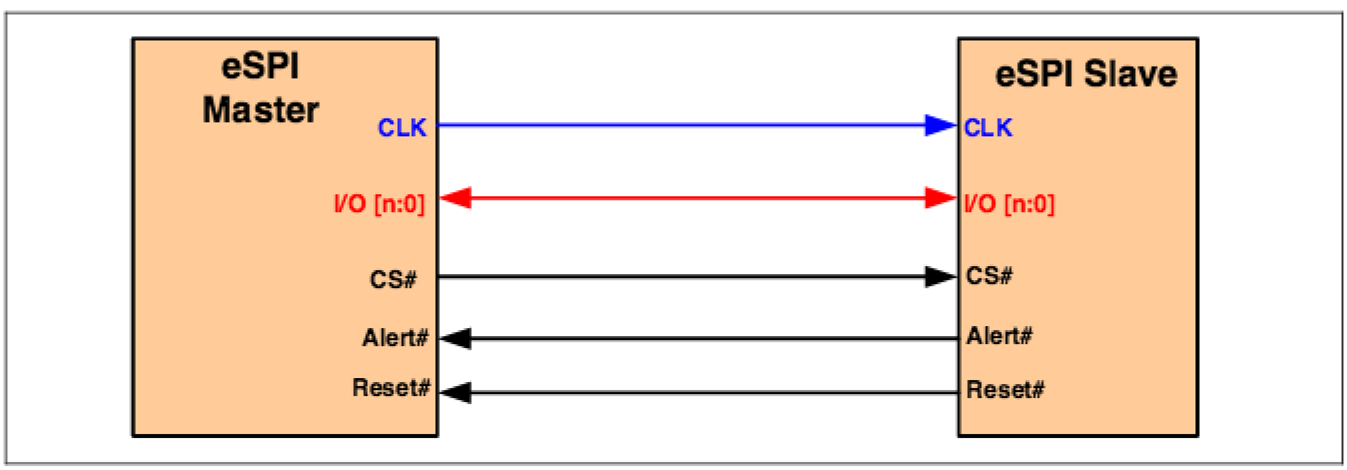

***Figure 1** : Intel eSPI*

Table 1 : eSPI/SPI Comparison

Pin Name	Intel eSPI	Motorola SPI
CS/SS (Master to Slave)	Yes	Yes
CLK/SCLK (Master to Slave)	Yes	Yes
IO [n:0] / MOSI/MISO (Bi-directional)	Yes	Yes
Reset (Master to Slave or Slave to Master)	Yes	No
Alert (Slave to Master)	Yes	No

2.1.2 eSPI Architecture

Top

eSPI 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.

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.

SPI slaves such as Flash and defined 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 run over the eSPI bus.

eSPI Architecture Description

Top

In a Single Master - Single Slave configuration, there could be multiple eSPI host bridges within a single eSPI master and there could be multiple eSPI endpoints within a single eSPI slave. 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.

In Single Master - Multiple Slaves configuration 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.

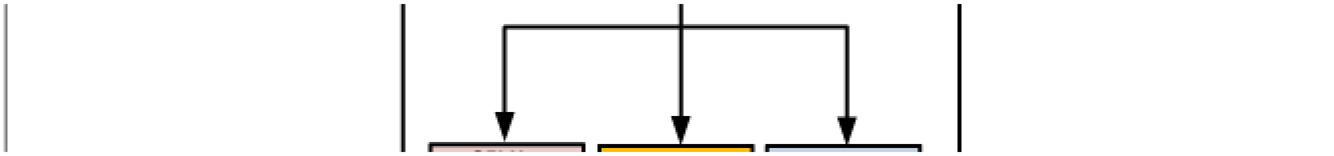

__Figure 3__ : Single Master - Single Slave

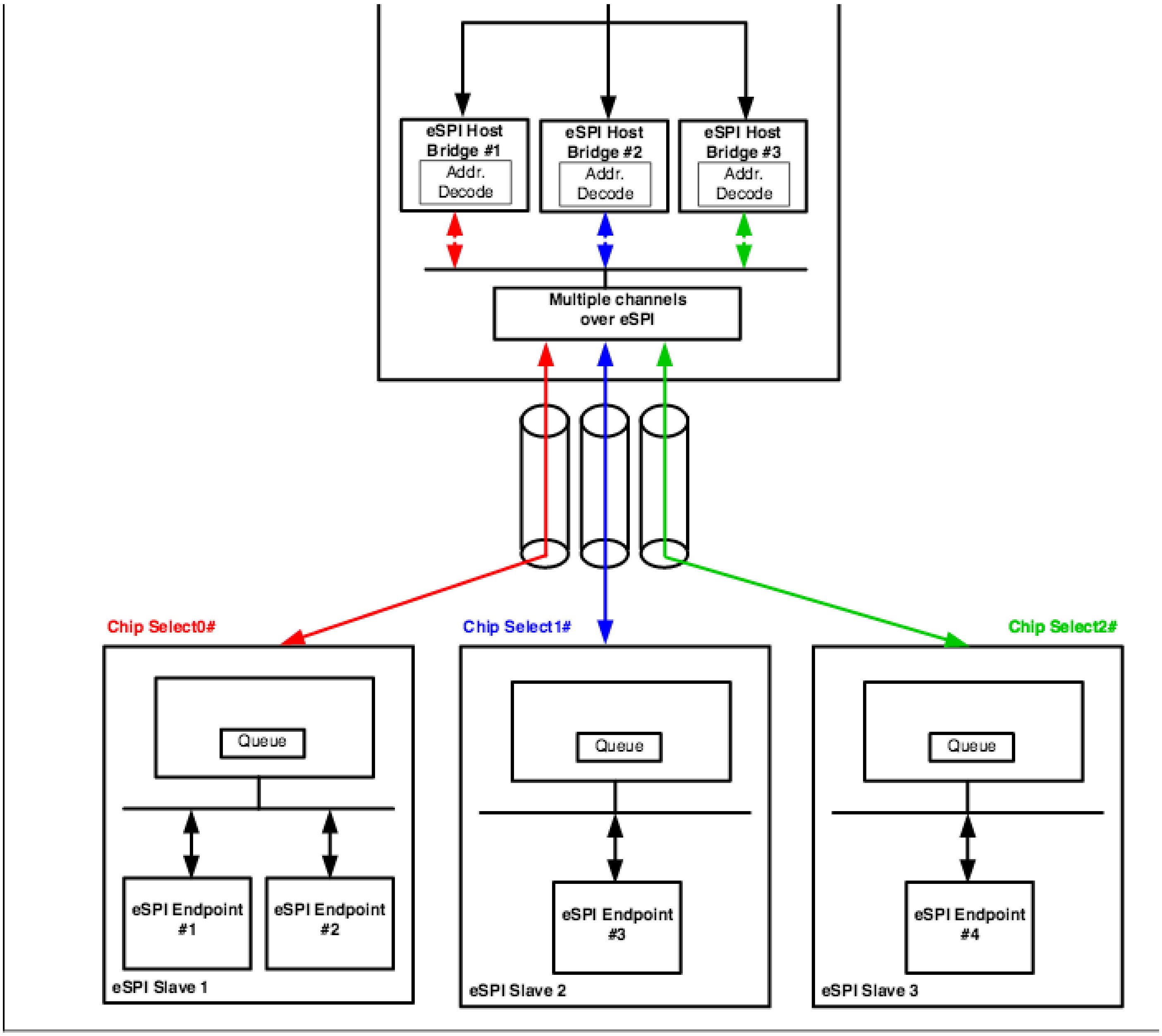

Figure 4 *: Single Master - Multi Slave*

2.1.3 eSPI Operations Theory

The Serial Clock must be low at the assertion edge of the Chip Select# (CS#) 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 eSPI master and eSPI slaves must tri-state the interface pins when their respective eSPI Reset# is asserted. The Chip Select#, I/O[n:0] and Alert# pins require weak pull-up to be enabled on these pins whereas the Serial Clock requires a weak pull-down. The weak pull-up/pull-down should be implemented either as an integral part of the eSPI master buffer or on the board.

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# pins continue to have the weak pull-up enabled for the proper operation of the eSPI bus.

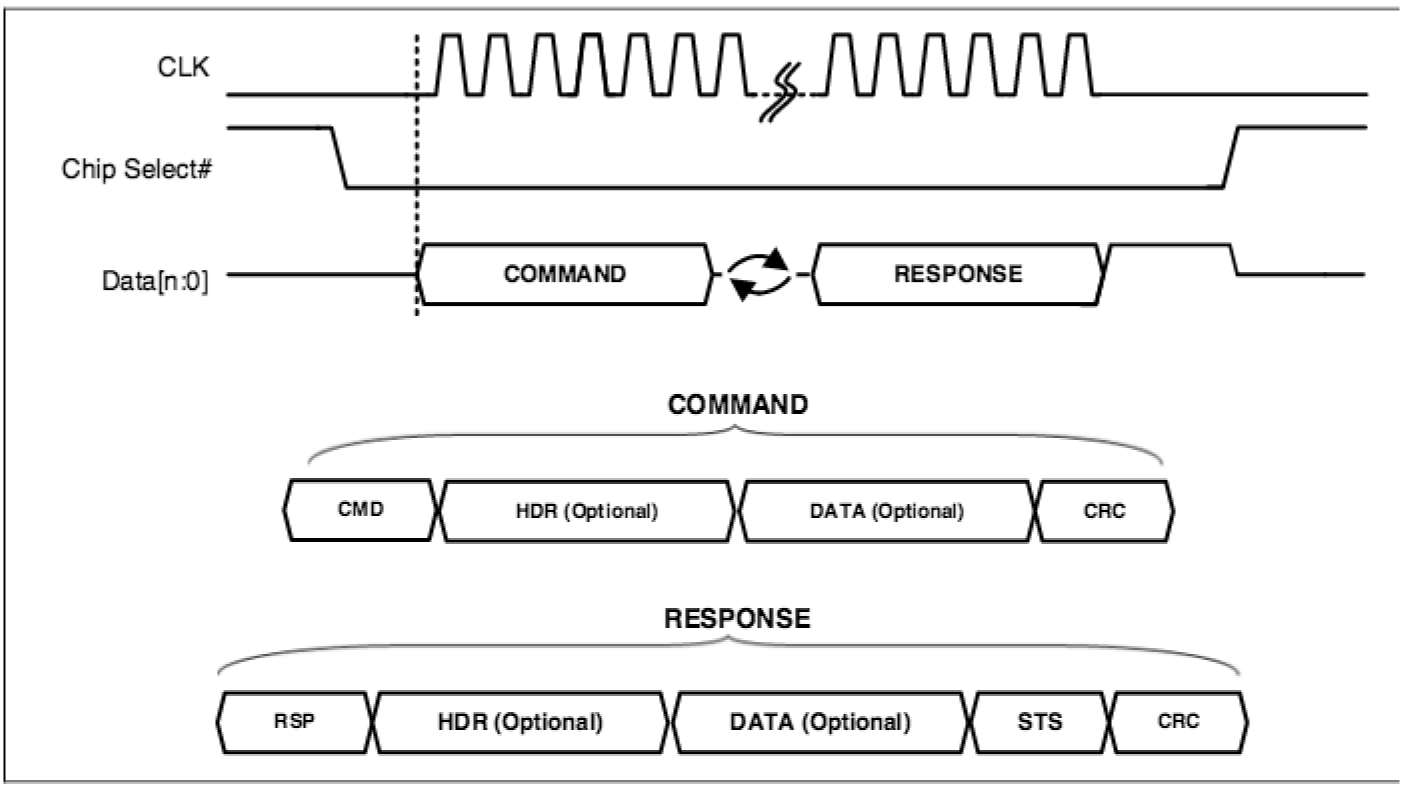

***Figure 5** : eSPI Operation*

An eSPI transaction consists of a Command phase driven by master, a Turn-Around (TAR) phase, and a Response phase driven by the slave. CRC generation is mandatory for all eSPI transactions where CRC byte is always transmitted on the bus. A transaction is initiated by the master by asserting the Chip Select#, starting the clock and driving the command onto the

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 Command Opcode is 8-bits wide.

Turn-Around (TAR) Phase

Top

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).

Response Phase

Top

The Response phase is driven by the eSPI slave in response to command initiated by an eSPI master. It consists of a RSP opcode, an optional header (HDR), optional data, STATUS (STS) and CRC. The RSP opcode is a 8-bit field consists of a Response Code and a Response Modifier.

Slave-initiated transactions

Top

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

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

At the last falling edge of the serial clock after CRC is sent, the eSPI slave must drive I/O[n:0] and Alert# pins to high until Chip Select# is deasserted. After Chip Select# deassertion, these pins are tri-stated by the slave, where the weak pull-ups maintain these pins at high.

Figure 6 *: Slave Triggered Transaction (Single Slave)*

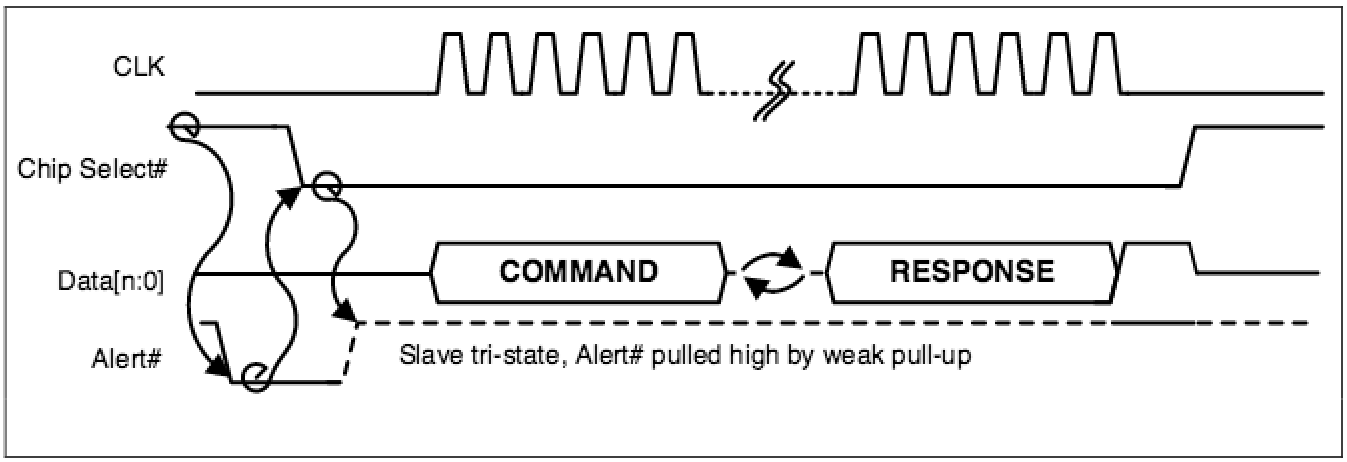

Figure 7 *: Slave Triggered Transaction (Multiple Slaves)*

Channels

Top

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 traffic 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.

There are four different channels types.

- **Virtual Wire 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.
- **OOB Channel:** The SMBus packets are tunneled through eSPI as Out-Of-Band (OOB) messages. The whole SMBus packet is embedded inside the eSPI OOB message as data.
- **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.

Link Layer

Top

All masters and slaves 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.

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#.

Each of the fields for an eSPI transaction is shifted out accordingly in a defined order. For fields that contain multiple bytes, the order of the bytes being shifted out on the eSPI bus is as follows (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.

Table 2 : Transaction Example

2	Tag[3:0]	Length[11:8]
3	Length[7:0]	
4	Address[31:24]	
5	Address[23:16]	
6	Address[15:8]	
7	Address[7 :0]	
8	Data[7:0]	
9 ... n	Data[15:8] . . .	
	Command Phase CRC[7:0]	
	(Turn Around)	
0	Response Opcode[7:0]	
1	Cycle Type[7:0]	
2	Tag[3:0]	Length[11:8]
3	Length[7:0]	
4	Address[31:24]	
5	Address[23:16]	
6	Address[15:8]	
7	Address[7 :0]	
8	Data[7:0]	
9 ... n	Data[15:8] . .	

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].

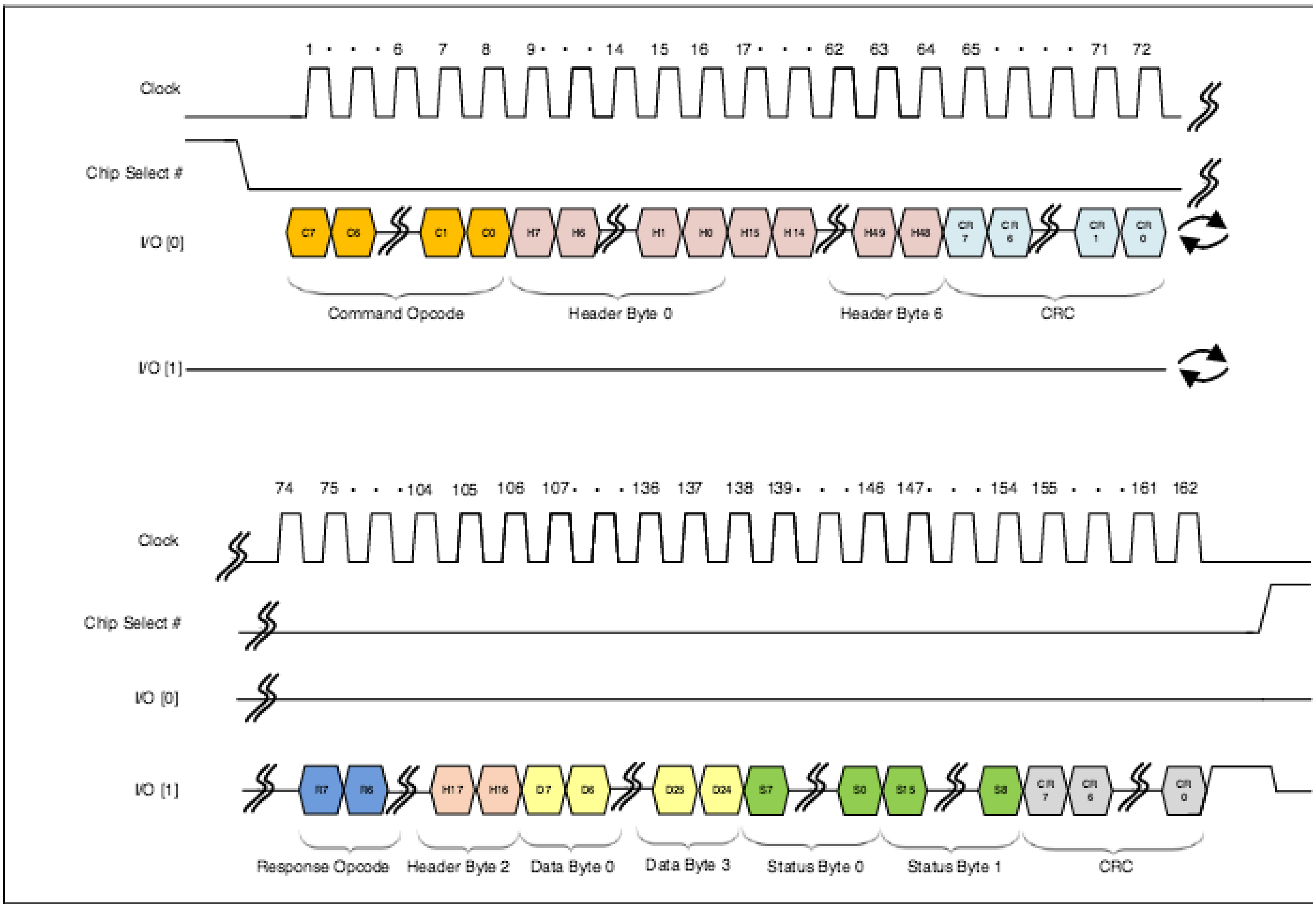

Figure 8 *: Single I/O Mode*

Dual IO mode

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.

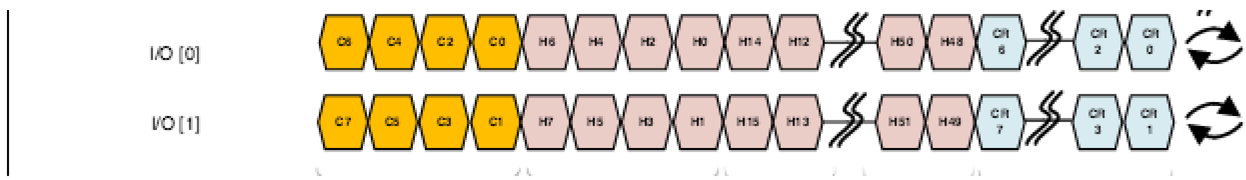

***Figure 9** : Dual I/O Mode*

Quad IO 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.

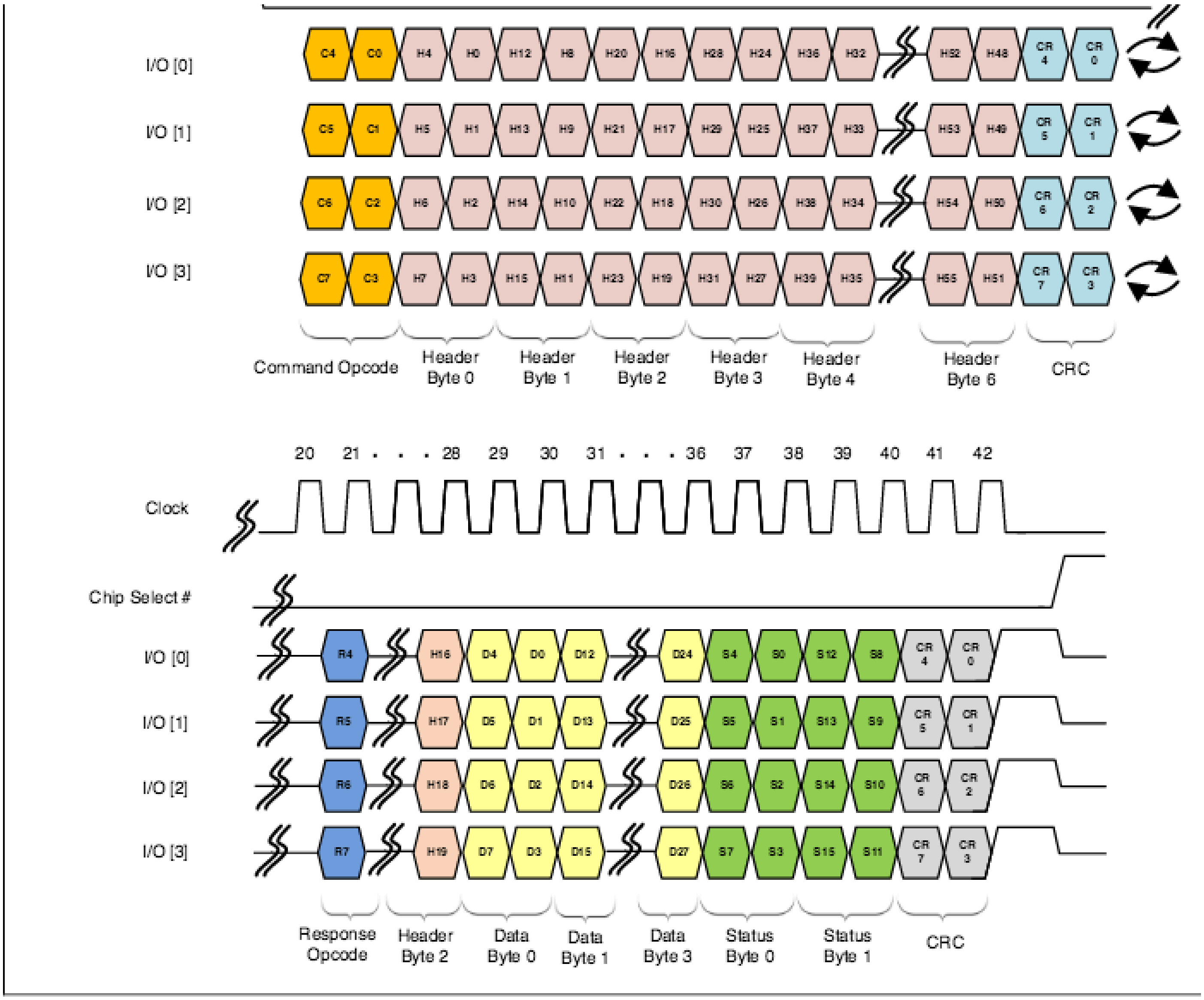

***Figure 10** : Quad I/O Mode*

2.1.4 eSPI References

- eSPI – Intel Enhanced Serial Peripheral Interface Specification
- LPC – Intel Low Pin Count Specification
- SPI – Wikipedia Serial Peripheral Interface Description

3.1.1 Connector Specification

The Promira Serial Platform with eSPI Analysis Application target connector is a standard 2x17 IDC male type connector 0.079x0.0792″ (2x2 mm). The Promira platform target connector allows for up to a 34-pin ribbon cable and connector.

One 34-34 cable is provided with the Promira platform: A standard ribbon cable 0.0392″ (1 mm) pitch that is 5.122″ (130mm) long with two 2x17 IDC female 2x2mm (0.079x0.079) connectors. This provided target ribbon cable will mate with a standard keyed boxed header.

3.1.2 Orientation

The pin order of the 2x17 IDC female connector in the provided target ribbon 34-34 cable is described in figure 11. When looking at the Promira platform front position with the 34-34 ribbon cable (figure 11), pin 1 is in the top left corner and pin 34 is in the bottom right corner.

Figure 11 *: Promira platform front position with 34-34 cable*

3.1.3 Pin Description

Table 3 (1) : Pin Description - Target Connector

Pin	Symbol	Description
1	Alert0	Alert pin for slave 0
3	Alert1	Alert pin for slave 1
4	V_{TGT}	Software configurable Vcc target power supply. NC/3.3V/5V

7	SCK	eSPI Clock
8	IO0	eSPI IO 0
9	CS0	eSPI Slave Select (Chip Select) 0
11	IO2	eSPI IO 2
13	IO3	eSPI IO 3
14	CS1	eSPI Slave Select (Chip Select) 1
15	Reset0	eSPI Reset Pin 0 (Master o Slaves)
17	DIO0	Software configurable digital input/output pin 0
19	DIO1	Software configurable digital input/output signal 1
20	DIO4	Software configurable digital input/output signal 4
21	DIO2	Software configurable digital input/output signal 2
22	V_{IO}	Software configurable Vcc IO level power supply. NC/1.8V
23	DIO3	Software configurable digital input/output signal 3
24	V_{IO}	Software configurable Vcc IO level power supply. NC/1.8V
25	DIO5	Software configurable digital input/output signal 5
26	DIO9	Software configurable digital input/output signal 9
27	DIO6	Software configurable digital input/output signal 6
29	DIO7	Software configurable digital input/output signal 7

33	DIO8	Software configurable digital input/output signal 8
2, 10, 12, 16, 18, 28, 30, 34	GND	Ground Connection

Note:

(1) When the Promira platform monitors a system that does not use any or all of the following signals: Pin 3 - Alert1, Pin 14 - CS1, and Pin 31 Reset1, it is strongly recommended that these signals are left as No Connect (NC). If that is not possible, any connected unused pins are required to have a logic level of 1 at all times.

Top

3.2 LEDs

LED2 (the middle LED in the Promira platform upper side) is blinking red, when the Promira platform data capture is active.

Top

3.3 Speeds

The Promira platform with eSPI analysis application is capable of monitoring the eSPI bus at bit rates 20, 25, 33, 50 and 66 MHz in single, dual, quad IO modes.

Top

3.4 Digital I/O

Promira platform digital inputs allow users to synchronize external logic with the analyzed eSPI data stream. When the state of an enabled digital input changes, an event is sent to the analysis PC. Digital input may not oscillate at a rate faster than 10 MHz. If digital input oscillates at a rate faster 10 MHz, then the events may not be passed to the PC. Digital inputs are rated for 1.8 V.

Promira platform digital outputs allow users to output events to external devices, such as an oscilloscope or logic analyzer, especially to trigger the oscilloscope to capture data. The digital

The Promira platform with eSPI analysis application contains a 64 MB on-board buffer. The memory provides a temporary FIFO storage buffer for capture data. This buffer serves two capture when the analysis computer can not stream the data off the analyzer fast enough. It is also used during a delayed-download capture to store all of the captured data.

4 Device Operation

Promira platform monitors the eSPI signals including: four IO signals, one SCK signal, two CS signals, two Alert signals, and two Reset signals.

eSPI specification requires that the master and slaves start communicating in single IO mode at 20 MHz on power-up and later on, the master configures the operating mode based on the slaves capabilities. This is done by sending a SET_CONFIGURATION command to the slaves 'General Capabilities Register' (offset 0x0008) with the desired IO mode/Alert pin mode/frequency setting. Promira platform captures and remembers the IO mode and Alert pin mode when it recognizes a SET_CONFIGURATION command in offset 0x0008 to a slave. This is automatic as long as the Promira platform captures all traffic from the beginning which is the recommended usage mode. Alternately, eSPI IO mode can also be configured by the user to Single/Dual/Quad. eSPI Alert mode can be configured by the user to IO1 signal or Alert0 signal.

Top

4.1 Capture Mode

Promira platform has two main capture modes: standard capture mode and delayed-download capture mode.

- In standard capture mode the capture data is streamed out from the Promira platform to the analysis computer immediately.
- In delayed-download capture mode, the capture data is not streamed out from the Promira platform to the analysis computer until after the analyzer has stopped monitoring the bus. When the captured is stopped, all the captured data is streamed from the analyzer to the analysis computer.

Top

4.2 General Device Features

- Monitors SET_CONFIGURATION command to 'General Capabilities Register' and automatically remember the frequency of operation and IO mode configured by the master and reports it for every packet seen on the bus.
- Monitors SET_CONFIGURATION command to 'General Capabilities Register' and automatically remember Alert mode setting and appropriately monitors IO1 signal or Alert-0/Alert-1 signals (which is applicable only in a single master - single slave setting).
- Monitors SET_CONFIGURATION command to 'Channel Capabilities Registers' and automatically remember the master configurable fields (for example maximum payload request and maximum payload size for peripheral channel transactions).
- Monitors the status field returned in every response and remember the queues' availability status for all channels.
- Decodes commands based on channels and reports the information to the user for every packet.
- Reports correct/incorrect command phase and response phase CRC errors for every packet to the user.
- Reports Master side errors and Slave side errors.
- Resets the captured IO mode and frequency of operation on a reset toggle on the signal.

Top

4.3 Digital IO

The Promira platform has 11 digital IO signals that can be configured by the user to input or output. Digital inputs provide a means to users to insert events into the data stream. Digital outputs provide a means for users to match certain events and to send output to other devices, such as oscilloscopes. In this way, users can synchronize events on the bus with other signals they may be measuring. Digital input event (failing edge or rising edge) can trigger a capture. Digital output behavior can be configured to: set low, set high, toggle (initially low), and toggle (initially high).

Digital input event can trigger a capture, and capture event can toggle digital output on the following scenarios:

- Packet with a command field that matches an 8-bit value set by the user.
- Packet with a command field that does not match an 8-bit value set by the user.
- Peripheral channel transaction.

- Set Configuration command.
- Get Status command.
- Platform Reset command.
- Alert event on the eSPI bus.
- Reset event on the eSPI bus.

Top

4.4 Match/Action System

The Promira platform features a multi-tiered matching/action system that can perform one or more actions in response to a match/action.

The first level is simple matching which can match the occurrence of packet types by channel, user selected command value and events and trigger a capture and/or assert an external output pin in response.

The second level is advanced matching which provides three different options that are explained below. On a match the system can trigger a capture and/or assert an external output pin in response.

The third level is hardware filters which provides mechanism to filter out packets.

Simple match and advanced match are separates features. The user can select simple match feature or advanced match feature but not both features at the same time.

4.4.1 eSPI Simple Match

Simple match can trigger a capture, and capture event can toggle a digital output pin on the following scenarios:

- Packet with a command field that matches an 8-bit value set by the user.
- Packet with a command field that does not match an 8-bit value set by the user.
- Peripheral channel transaction.
- Virtual wire channel transaction.
- OOB channel transaction.
- Flash channel transaction.

TOTAL PHASE

It is possible to select multiple events to match the simple trigger. However, since a capture can only be triggered once, in the case of multiple selected events, the first of any of the selected events will trigger the capture.

When an ouput pin is selcted to be asserted on a match, the pin will be asserted only once, when the trigger occurs.

4.4.2 eSPI Advanced Match

The Promira eSPI advanced trigger is a complex pattern/sequence match engine that provides triggering on specific condition/sequence of events on the eSPI bus with multiple options and a high level of configurability specifically tailored around the eSPI protocol.

Using the advanced match feature, the user can specify and configure the analyzer to match and trigger on three different types of conditions/sequences based on eSPI packets.

Match/Trigger Option 1 - Multiple Packets

Configure the analyzer to match and trigger on a sequence of up to four eSPI packets (each packet is defined as a level). The match condition can be different for each level. An output trigger pin can be configured to be drive logic high or low (selectable) for each level when an eSPI packet satisfies that level's match condition.

Match/Trigger Option 2 - Non-Posted Transactions

Configure the analyzer to match and trigger on a specific non-posted transaction and corresponding completion(s). A non-posted transaction has two distinct phases, a request phase and one or more completion phases. Completions can be successful/unsuccessful, and connected or split across multiple completion packets. In the case of split completions the analyzer will track first, middle, last (only) completions as defined in the eSPI specification. An output trigger pin can be configured to be drive logic high or low (selectable) at different stages of the request/completion sequence as they occur on the wire. The four distinct stages are 1- Request, 2-First completion, 3-Middle completion (the first packet in case there are multiple middle completion packets), and 4-Last (Only) completion. Please note that the user only needs to configure the analyzer to match a specific non-posted request and the type of completion to look for (successful/unsuccessful); the analyzer tracks the completion(s) for the request automatically.

Match/Trigger Option 3 - Errors

The eSPI hardware filter feature provides mechanism to filter out packets (which are listed below) in order to discard unwanted data, and reduce the amount of captured data that is sent back to the analysis computer. The settings can be configured independently for each slave.

- Packet with a command field that matches an 8-bit value set by the user.
- Packet with a command field that does not match an 8-bit value set by the user.
- Peripheral channel transaction.
- Virtual wire channel transaction.
- OOB channel transaction.
- Flash channel transaction.
- Independent Channel Transaction

Top

4.5 Hardware Statistics

The Promira platform features hardware statistics which provides a count of packets / events for each slave. The following counters are available:

- Packets with command CRC error.
- Packets with response CRC error.
- Peripheral channel packets.
- Virtual wire channel packets.
- OOB channel packets.
- Flash channel packets.
- Get Configuration packets.
- Set Configuration packets.
- Get Status packets.
- Platform Resets.
- All packets that were filtered out
- All packets that were filtered out based on command

5 Software

5.1.1 Overview

The Promira Rosetta language bindings make integration of the Promira API into custom applications simple. Accessing Promira functionality simply requires function calls to the Promira API. This API is easy to understand, much like the ANSI C library functions, (e.g., there is no unnecessary entanglement with the Windows messaging subsystem like development kits for some other embedded tools).

First, choose the Rosetta bindings appropriate for the programming language. Different Rosetta bindings are included in the software download package available on the Total Phase website. Currently the following languages are supported: C/C++, C#, VB, Python. Next, follow the instructions for each language binding on how to integrate the bindings with your application build setup. As an example, the integration for the C language bindings is described below. (For information on how to integrate the bindings for other languages, please see the example code available for download on the Total Phase website.)

1. Include the `promira.h` and `promana.h` files in any C or C++ source module. The module may now use any API call listed in `promira.h` and `promana.h`.
2. Compile and link `promira.c` and `promana.c` with your application. Ensure that the include path for compilation also lists the directory in which `promira.h` and `promana.h` is located if the two files are not placed in the same directory.
3. Place the Promira DLL (promira.dll), included with the API software package, in the same directory as the application executable or in another directory such that it will be found by the previously described search rules.

5.1.2 Versioning

Since a new Promira DLL can be made available to an already compiled application, it is essential to ensure the compatibility of the Rosetta binding used by the application against the DLL loaded by the system. A system similar to the one employed for the DLL-Firmware cross-validation is used for the binding and DLL compatibility check.

Here is an example.

```
DLL v1.20: compatible with Binding >= v1.10 Binding v1.15: compatible
with DLL >= v1.15
```

While provided language bindings stubs are fully functional, it is possible to modify the code found within this file according to specific requirements imposed by the application designer.

For example, in the C bindings one can modify the DLL search and loading behavior to conform to a specific paradigm. See the comments in `promira.c` for more details.

6 API Documentation

Top

6.1 Introduction

The Promira API documentation that follows is oriented toward the Promira Rosetta C bindings. The set of Promira API functions and their functionality is identical regardless of which Rosetta language binding is utilized. The only differences will be found in the calling convention of the functions. For further information on such differences please refer to the documentation that accompanies each language bindings in the Promira API Software distribution

Top

6.2 General Data Types

The following definitions are provided for convenience. All Promira data types are unsigned.

```
typedef unsigned char u08; typedef unsigned short u16; typedef unsigned
int u32; typedef unsigned long long u64; typedef signed char s08;
typedef signed short s16; typedef signed int s32; typedef signed long
long s64; typedef float f32;
```

Top

6.3 Notes on Status Codes

Most of the Promira API functions can return a status or error code back to the caller. The complete list of status codes is provided at the end of this chapter. All of the error codes are assigned values less than 0, separating these responses from any numerical values returned by certain API functions.

Furthermore, all API calls can potentially return the error PA_API_UNABLE_TO_LOAD_FUNCTION. If this error is encountered, there is likely a serious version incompatibility that was not caught by the automatic version checking system. Where appropriate, compare the language binding versions (e.g., `PM_HEADER_VERSION` found in `promira.h` and `PM_CFILE_VERSION` found in `promira.c` or `PA_APP_HEADER_VERSION` found in `promana.h` and `PA_APP_CFILE_VERSION` found in `promana.c`) to verify that there are no mismatches. Next, ensure that the Rosetta language binding (e.g., `promira.c` and `promira.h` or `promana.c` and `promana.h`) are from the same release as the Promira DLL. If all of these versions are synchronized and there are still problems, please contact Total Phase support for assistance.

Any API function that accepts any type of handle can return the error `PA_APP_INVALID_HANDLE` if the handle does not correspond to a valid instance that has already been opened or created. If this error is received, check the application code to ensure that the open or create command returned a valid handle and that this handle is not corrupted before being passed to the offending API function.

Finally, any function call that communicates with an Promira device can return the error `PA_APP_COMMUNICATION_ERROR`. This means that while the handle is valid and the communication channel is open, there was an error receiving the acknowledgment response from the Promira application. The error signifies that it was not possible to guarantee that the connected Promira device has processed the host PC request, though it is likely that the requested action has been communicated to the Promira device and the response was lost.

These common status responses are not reiterated for each function. Only the error codes that are specific to each API function are described below.

All of the possible error codes, along with their values and status strings, are listed following the API documentation.

Top

6.4 Application Management Interface

All functions starting with `pm_` are for Application Management. Please refer to the Promira Serial general user manual for the details.

Top

Top

Overview

After opening the device with `pm_open` and starting an application with `pm_load`, a connection needs to be established with `pa_app_connect`. See the language specific sample programs for examples of this connection process.

Software Operational Overview

Top

There are a series of steps required for a successful capture. These steps are handled by the Data Center software automatically, but must be explicitly followed by an application programmer wishing to write custom software. The following is meant to provide a high-level overview of the operation of the Promira platform.

1. Determine the IP address of the Promira platform. The function `pm_find_devices` returns a list of IP addresses for all Promira platforms that are attached to the analysis computer.
2. Obtain a Promira connection handle by calling the function `pa_app_connect` on the appropriate IP addresses. All other software operations are based on this handle.
3. Configure the Promira platform as necessary. The API documentation provides complete details about the different configurations.
4. Start the capture by calling the function `pa_capture_start`.
5. Retrieve monitored data by using the read functions that are appropriate for the monitored bus type. There are different functions available for retrieving additional data such as byte- and bit-level timing.
6. End the capture by calling the function `pa_capture_stop`. At this point the capture is stopped, and no new data can be obtained. Captured data may still be read from the on-board buffer after calling this function.
7. Close the Promira platform handle with the function `pa_app_disconnect`.

If the Promira platform is disabled and then re-enabled it does not need to be re-configured. However, upon closing the handle, all configuration settings will be lost.

Example code is available for download from the Total Phase website. These examples demonstrate how to perform the steps outline above for each of the serial bus protocols supported.

Arguments

net_addr	The net address of the Promira Serial Platform. It could be an IPv4 address or a host name.

Return Value

This function returns a connection handle, which is guaranteed to be greater than zero if valid.

Specific Error Codes

PA_APP_UNABLE_TO_OPEN	Unable to connect to the application.

Details

Only one connection can be made to the application.

Disconnect to the Application (pa_app_disconnect)

Top

```
int pa_app_disconnect (PromiraConnectionHandle conn)
```

Disconnect to the application.

Arguments

conn	handle of the connection

Return Value

The number of the connections to applications disconnected is returned on success. This will usually be 1.

Specific Error Codes

None.

Details

If the conn argument is zero, the function will attempt to disconnect all possible handles, thereby disconnecting all connected handles. The total number of handle

```
version);
```

Return the version matrix for the application connected to the given handle.

Arguments

conn	handle of the connection
version	pointer to pre-allocated structure

Return Value

A status code is returned with `PA_APP_OK` on success.

Specific Error Codes

None.

Details

The `PromiraAppVersion` structure describes the various version dependencies of application components. It can be used to determine which component caused an incompatibility error.

```
struct PromiraAppVersion { /* Software, firmware, and hardware versions. */ u16 software; u16 firmware; u16 hardware; /* FW requires that SW must be >= this version. */ u16 sw_req_by_fw; /* SW requires that FW must be >= this version. */ u16 fw_req_by_sw; /* API requires that SW must be >= this version. */ u16 api_req_by_sw; };
```

If the handle is 0 or invalid, only `software`, `fw_req_by_sw`, and `api_req_by_sw` version are set.

Status String (pa_app_status_string)

Top

```
const char *pa_app_status_string (int status);
```

Return the status string for the given status code.

Arguments

not valid, it returns a NULL string.

Specific Error Codes

None.

Details

None.

Get Features (pa_app_features)

Top

```
int pa_app_features (PromiraConnectionHandle conn);
```

Return the device features as a bit-mask of values, or an error code if the handle is not valid.

Arguments

conn handle of the connection

Return Value

The features of the Promira platform are returned. These are a bit-mask of the following values.

```
#define PA_FEATURE_NONE (0) #define PA_FEATURE_ESPI (1<<0)
```

Specific Error Codes

None.

Details

None.

6.5.2 Configuration

Top

Target Power (pa_phy_target_power)

```
int pa_phy_target_power (PromiraConnectionHandle conn, u08 power_mask);
```

Activate/deactivate target power pins 4, 6 and/or 22, 24.

Table 4 : `power_mask` enumerated types

PA_PHY_TARGET_POWER_NONE	Disable target power pins 4, 6, 22, 24. Pins 4, 6, 22, 24 at GND level.
PA_PHY_TARGET_POWER_TGT1_5V	Enable 5V on target power pins 4 and 6.
PA_PHY_TARGET_POWER_TGT1_3V	Enable 3.3V on target power pins 4 and 6.
PA_PHY_TARGET_POWER_TGT2	Enable target power pins 22 and 24 with the same voltage as the all signals voltage level as programed by API function `pa_phy_level_shift`. The all logic level can be programed to 0.9V to 3.45V. The precision level of the level shifter is approximately 0.015V. For the Promira platform with eSPI Analysis application the only available voltage is 1.8V.
PA_PHY_TARGET_POWER_BOTH	Enable 5V on target power pins 4 and 6, and enable target power pins 22 and 24 with the same voltage as the all signals voltage level as programed by API function `pa_phy_level_shift`.
PA_PHY_TARGET_POWER_QUERY	Queries the target power pin state.

Return Value

The current state of the target power pins will be returned. The configuration will be described by the same values as in the table above.

Specific Error Codes

None.

Details

None.

Arguments

conn	handle of the connection
level	logic level from 0.9V to 3.45V

Return Value

The Actual logic level on the Promira host adapter will be returned.

Specific Error Codes

None.

Details

The call with `PA_PHY_LEVEL_SHIFT_QUERY` returns existing configuration and does not modify.

For Promira platform with eSPI analysis application, the logic level of all signal is fixed to 1.8V.

6.5.3 Monitoring API

Top

Start Capture (pa_capture_start)

```
int pa_capture_start (PromiraConnectionHandle conn, PromiraProtocol
protocol, PromiraTriggerMode trig_mode);
```

Start monitoring packets on the selected interface.

Arguments

conn	handle of the connection
protocol	enumerated values specifying the protocol to monitor (see Table 5)
trig_mode	enumerated values specifying the trigger mode (see Table 6)

Table 5 : `PromiraProtocol` enumerated values

PA_PROTOCOL_NONE	No Protocol

PA_TRIGGER_MODE_EVENT	Trigger on match event
PA_TRIGGER_MODE_IMMEDIATE	Trigger immediately

Return Value

A status code is returned with PA_APP_OK on success.

Specific Error Codes

PA_APP_FUNCTION_NOT_AVAILABLE	The connected Promira platform does not support capturing for the requested protocol.
PA_APP_UNKNOWN_PROTOCOL	A protocol was requested that does not appear in the enumeration detailed in Table 5.

Details

This function enables monitoring on the given Promira platform. See the section on the protocol-specific APIs. Functions for retrieving the capture data from the Promira platform are described therein.

Trigger Capture (pa_capture_trigger)

Top

```
int pa_capture_trigger (PromiraConnectionHandle conn);
```

Trigger the capture.

Arguments

conn handle of the connection

Return Value

A status code is returned with PA_APP_OK on success.

Specific Error Codes

be downloaded from the on-board buffer by calling the read function.

This triggers only when the capture started with `PA_TRIGGER_MODE_EVENT` as `trig_mode`.

Wait for Capture to Trigger (pa_capture_trigger_wait)

Top

```
int pa_capture_trigger_wait (PromiraConnectionHandle conn, int
timeout_ms, PromiraCaptureStatus * status);
```

Wait for the capture to trigger.

Arguments

conn	handle of the connection
timeout_ms	timeout value
status	filled with enumerated value described in Table 7

Table 7 : PromiraCaptureStatus Enums

PA_CAPTURE_STATUS_INACTIVE	Capture is not active
PA_CAPTURE_STATUS_PRE_TRIGGER	Filling pre-trigger
PA_CAPTURE_STATUS_PRE_TRIGGER_SYNC	Downloading pre-trigger
PA_CAPTURE_STATUS_POST_TRIGGER	Filling post-trigger
PA_CAPTURE_STATUS_TRANSFER	Capture stopped, downloading data
PA_CAPTURE_STATUS_COMPLETE	Capture stopped, all data downloaded

Return Value

A status code is returned with `PA_APP_OK` on success.

Specific Error Codes

None.

Stop Capture (pa_capture_stop)

Top

```
int pa_capture_stop (PromiraConnectionHandle conn);
```

Stop capturing data.

Arguments

conn	handle of the connection

Return Value

A status code is returned with `PA_APP_OK` on success.

Specific Error Codes

None.

Details

Captured data may still be read from the on-board buffer after calling this function, but new data will not be monitored.

Query Capture Status (pa_capture_status)

Top

```
int pa_capture_status (PromiraConnectionHandle conn,
PromiraCaptureStatus * status, PromiraCaptureBufferStatus *
buf_status);
```

Query the status of capture.

Arguments

conn	handle of the connection
status	filled with enumerated value described in Table 7
buf_status	filled with buffer status described in Table 8

Return Value

A status code is returned with `PA_APP_OK` on success.

Query the capture status and the states of the trigger and capture buffers.

When on-board capture buffer gets full, the analyzer will stop capturing new data while allowing all of the previously captured data to be downloaded. The `PA_CAPTURE_STATUS_TRANSFER` will indicate that the capture has stopped because the buffer became *full* or stopped by `pa_capture_stop`; previous data is still available for download. `capture_remaining` (to be downloaded) will return the amount currently in the on-board buffer.

The `PromiraCaptureBufferStatus` structure shows the current status of on-board capture buffer.

```
struct PromiraCaptureBufferStatus { u32 pretrig_remaining_kb; u32
pretrig_total_kb; u32 capture_remaining_kb; u32 capture_total_kb;
};
```

Table 8 : `PromiraCaptureBufferStatus` field descriptions

pretrig_remaining_kb	filled with amount of remaining pre-trigger data to capture (in KB)
pretrig_total_kb	filled with pre-trigger size set by user (in KB)
capture_remaining_kb	filled with amount of remaining total capture data to capture (in KB)
capture_total_kb	filled with total capture size set by user (in KB)

Configure Capture Buffer (pa_capture_buffer_config)

Top

```
int pa_capture_buffer_config (PromiraConnectionHandle conn, u32
pretrig_kb, u32 capture_kb);
```

Configure on-board capture buffer.

Arguments

conn	handle of the connection
pretrig_kb	amount (in KB) of pre-trigger data to capture

_ _

Specific Error Codes

PA_APP_CONFIG_ERROR	An attempt was made to set an invalid configuration.

Details

The hardware buffer may vary on the application and the license. It is specified in the license.

The size of `capture_kb` includes `pretrig_kb`. Attempting to set `pretrig_kb` greater than `capture_kb` will return an error.

The on-board buffer for pre-trigger data is circular queue which means it keeps last data before trigger happens. When it gets full but trigger doesn't happen yet, oldest data will be gone.

When the on-board capture buffer gets full, it will automatically stops monitoring. The data already monitored is still available for download.

For Promira platform with eSPI analysis application, each packet takes 8K fixed buffer size no matter how big actual data is.

Query Capture Buffer Config (pa_capture_buffer_config_query)

Top

```
int pa_capture_buffer_config_query ( PromiraConnectionHandle conn, u32
* pretrig_kb, u32 * capture_kb);
```

Query the current on-board capture buffer configuration.

Arguments

conn	handle of the connection
pretrig_kb	filled with pre-trigger size (in KB)
capture_kb	filled with total capture size (in KB)

Return Value

A status code is returned with `PA_APP_OK` on success.

Query the on-board capture buffer configuration set in `pa_capture_buffer_config`.

6.5.4 Digital I/O Functions

Top

Configure Digital I/O (pa_digital_io_config)

```
int pa_digital_io_config (PromiraConnectionHandle conn, u32 enable, u32
direction, u32 polarity);
```

Configure digital I/O.

Arguments

conn	handle of the connection
enable	a bitmask specifying which digital I/O should be enabled.
direction	a bitmask of the direction of the digital I/O. If a digital I/O's bit is `0` (`PA_DIGITAL_DIR_INPUT`), the digital I/O is configured as an input. Otherwise it will be an output.
polarity	a bitmask of the polarity of the digital I/O. If a digital I/O's bit is `0` (`PA_DIGITAL_ACTIVE_LOW`), the digital I/O is active low. Otherwise it will be active high.

Return Value

A status code is returned with `PA_APP_OK` on success.

Specific Error Codes

None.

Details

The number of digital IOs may vary on the application and the license. It is specified in the license.

Promira platform with eSPI analysis application supports up to 11 digital I/Os.

Top

Query Digital I/O Config (pa_digital_io_config_query)

Arguments

conn	handle of the connection
enable	a bitmask specifying which digital IOs are enabled.
direction	a bitmask of the direction of the digital I/O.
polarity	a bitmask of the polarity of the digital I/O.

Return Value

A status code is returned with `PA_APP_OK` on success.

Specific Error Codes

None.

Details

None

6.5.5 Notes on Protocol-Specific Read Functions

All read functions return a timestamp, a duration, a status, and an event value through the `PromiraReadInfo` parameter.

```
struct PromiraReadInfo { u64 timestamp; u64 duration; u32 status; u32 events; };
```

Table 9 : `PromiraReadInfo` structure

timestamp	filled with the timestamp when the packet or the events begins. This is the number of nanoseconds from where capture started and will be reset to 0 when `pa_capture_start` gets called.
duration	filled with the number of nanoseconds that the packet or the events actually took.
status	filled with the status bitmask as detailed in Tables 10. See also Table 13 for eSPI.

PA_READ_OK	0x00000000	Read successful
PA_READ_ERR_CODE_MASK	0x000000ff	Mask for the protocol specific error code

Table 11 : Read Events definitions

PA_EVENT_DIGITAL_INPUT_MASK	0x00000fff	Mask for the bitmask of digital inputs
PA_EVENT_DIGITAL_INPUT	0x00001000	Digital input event
PA_EVENT_SLAVE_ID_MASK	0xf0000000	Mask for the index of slave
PA_EVENT_SLAVE0	0x00000000	Event from slave 0
PA_EVENT_SLAVE1	0x10000000	Event from slave 1
PA_EVENT_SLAVE2	0x20000000	Event from slave 2
PA_EVENT_SLAVE3	0x30000000	Event from slave 3

Each match parameter is represented by two separate fields: type and value. The `PromiraMatchType` enumerated type is used to determine whether a connection value field should be disabled, match on equal, or match on not equal. The different enumerated values are listed below. Restrictions on usage are indicated by footnotes.

Table 12 : `PromiraMatchType` enumerated values

PA_MATCH_TYPE_DISABLED	0	Disable
PA_MATCH_TYPE_EQUAL	1	Matched when field is equal to the given value
PA_MATCH_TYPE_NOT_EQUAL	2	Matched when field is not equal to the given value

Top

6.6 eSPI API

6.6.1 Notes

The Source and Fatalness of Error		
PA_READ_ESPI_ERR_TYPE_MASK	0x000000c0	Mask for the source of the error
PA_READ_ESPI_ERR_TYPE_MASTER	0x00000080	Error reported from master
PA_READ_ESPI_ERR_TYPE_SLAVE	0x00000040	Error reported from slave
PA_READ_ESPI_ERR_TYPE_MISC	0x000000c0	Miscellaneous error
PA_READ_ESPI_ERR_FATAL_MASK	0x00000020	Mask for fatal error and non-fatal error
Master Error Code		
PA_READ_ESPI_MST_INVALID_RSP_CODE	0x000000a0	Invalid Response Code (FATAL) Response code is not part of the specification
PA_READ_ESPI_MST_INVALID_CYCLE_TYPE	0x000000a1	Invalid Cycle Type (FATAL) Cycle type in the response phase is not part of the specification
PA_READ_ESPI_MST_NO_RSP	0x000000a3	Response Code: No Response (FATAL) Response code is all bitwise 1
PA_READ_ESPI_MST_RSP_FATAL	0x000000a4	Response Code: Fatal Error (FATAL) Response code indicates fatal error

PA_READ_ESPI_MST_PERIF_REQ_SIZE	0x000000a6	Peripheral Channel (Read request size > Max read request size): Fatal Error (FATAL)
PA_READ_ESPI_MST_PERIF_4K_XING	0x000000a7	Peripheral Channel (Address + Length crossing 4K boundary): Fatal Error (FATAL)
PA_READ_ESPI_MST_VW_MAX_COUNT	0x000000a8	Virtual Wire Channel (Count > Max virtual wire count): Fatal Error (FATAL)
PA_READ_ESPI_MST_OOB_PAYLOAD	0x000000a9	OOB Channel (SMBus Byte Count > Max Payload Size): Fatal Error (FATAL)
PA_READ_ESPI_MST_FLASH_PAYLOAD	0x000000aa	Flash Access Channel (Payload length > Max Payload Size): Fatal Error (FATAL)
PA_READ_ESPI_MST_FLASH_REQ_SIZE	0x000000ab	Flash Access Channel (Read request size > Max read request size): Fatal Error (FATAL)
PA_READ_ESPI_MST_PERIF_PAYLOAD_SIZE	0x000000ad	Peripheral Channel (Payload length > Max Payload Size) and (Read request size > Max read request

		Payload Size) and (Read request size > Max read request size): Fatal Error (FATAL)
`PA_READ_ESPI_MST_RSP_NON_FATAL`	0x00000080	Response Code: Non Fatal Error (NON_FATAL) Response code indicates non-fatal error
Slave Error Code		
`PA_READ_ESPI_SLV_PUT_WO_FREE`	0x00000060	PUT without FREE (FATAL) Slave receives the PUT command through channel which doesn't have free queue for posted commands
`PA_READ_ESPI_SLV_GET_WHEN_UNAVAIL`	0x00000061	GET without AVAIL (FATAL) Slave receives the GET command without any indication by the slave that it has data available
`PA_READ_ESPI_SLV_PERIF_PAYLOAD`	0x0000006a	Peripheral Channel (Payload length > Max Payload Size): Fatal Error (FATAL)
`PA_READ_ESPI_SLV_PERIF_REQ_SIZE`	0x00000062	Peripheral Channel (Read request size > Max read request

		crossing 4K boundary): Fatal Error (FATAL)
PA_READ_ESPI_SLV_VW_MAX_COUNT	0x00000064	Virtual Wire Channel (Count > Max virtual wire count): Fatal Error (FATAL)
PA_READ_ESPI_SLV_OOB_PAYLOAD	0x00000065	OOB Channel (SMBus Byte Count > Max Payload Size): Fatal Error (FATAL)
PA_READ_ESPI_SLV_FLASH_PAYLOAD	0x00000066	Flash Access Channel (Payload length > Max Payload Size): Fatal Error (FATAL)
PA_READ_ESPI_SLV_FLASH_REQ_SIZE	0x00000067	Flash Access Channel (Read request size > Max read request size): Fatal Error (FATAL)
PA_READ_ESPI_SLV_PERIF_PAYLOAD_SIZE	0x00000068	Peripheral Channel (Payload length > Max Payload Size) and (Read request size > Max read request size): Fatal Error (FATAL)
PA_READ_ESPI_SLV_FLASH_PAYLOAD_SIZE	0x00000069	Flash Access Channel (Payload length > Max Payload Size) and (Read request size > Max read request

		Command value is not defined in the specification
PA_READ_ESPI_SLV_INVALID_CYCLE_TYPE	0x00000041	Invalid Cycle Type (NON FATAL) Cycle type in the command phase is not part of the specification
Miscellaneous Error Code		
PA_READ_ESPI_PARTIAL_BYTE	0x000000e0	Number of data bits is not a multiple of 8
PA_READ_ESPI_RESET_WHILE_CS	0x000000e1	Reset# asserted while CS# is asserted
PA_READ_ESPI_ALERT_WHILE_CS	0x000000e2	Alert#(IO1) asserted while CS# is asserted
PA_READ_ESPI_INVALID_LENGTH	0x000000e3	Invalid eSPI packet length
PA_READ_ESPI_MORE_THAN_ONE_CS	0x000000e4	More than one chip select active at the same time
Error Bitmasks		
PA_READ_ESPI_ERR_BAD_CMD_CRC	0x00000100	Bad CRC for command
PA_READ_ESPI_ERR_BAD_RSP_CRC	0x00000200	Bad CRC for response

Table 14 : Read Events for eSPI definitions

PA_EVENT_ESPI_ALERT_RISING	0x00010000	Alert on rising edge
PA_EVENT_ESPI_ALERT_FALLING	0x00020000	Alert on falling edge
PA_EVENT_ESPI_RESET_RISING	0x00040000	Reset on rising edge

6.6.2 eSPI Monitor Interface

Top

Set Operating Configuration (pa_espi_operating_config)

```
int pa_espi_operating_config (PromiraConnectionHandle conn, u08
slave_id, const PromiraEspiOperatingCfg * cfg);
```

Set the operating configuration of eSPI system.

Arguments

conn	handle of the connection
slave_id	id of slave
cfg	configuration described in Table 15

Table 15 : `PromiraEspiOperatingCfg` field descriptions

io_mode	SPI IO mode to monitor described in Table 16
alert_pin	Alert pin to monitor described in Table 17
perif_max_req_size	Max read request size for peripheral channel described in Table 18
perif_max_payload	Max payload size for peripheral channel described in Table 18
vw_max_count	Max virtual wire count
oob_max_payload	Max payload size for OOB channel described in Table 18
flash_max_req_size	Max read request size for flash channel described in Table 18
flash_max_payload	Max payload size for flash channeldescribed in Table 18

Table 16 : `PromiraSpiIOMode` enumerated values

`PA_SPI_IO_UNKNOWN`	-1	Unknown

Table 17 : `PromiraEspiAlertPin` enumerated values

PA_ESPI_ALERT_UNKNOWN	0	Enumerated alert pin to monitor
PA_ESPI_ALERT_PIN	1	The designated alert pin
PA_ESPI_ALERT_IO1	2	IO1 pin for alert to monitor

Table 18 : `PromiraEspiAlign` enumerated values

PA_ESPI_ALIGN_UNKNOWN	0	Undefined or enumerated value
PA_ESPI_ALIGN_64_BYTES	1	64 bytes
PA_ESPI_ALIGN_128_BYTES	2	128 bytes
PA_ESPI_ALIGN_256_BYTES	3	256 bytes
PA_ESPI_ALIGN_512_BYTES	4	512 bytes
PA_ESPI_ALIGN_1024_BYTES	5	1024 bytes
PA_ESPI_ALIGN_2048_BYTES	6	2048 bytes
PA_ESPI_ALIGN_4096_BYTES	7	4096 bytes

Return Value

A status code is returned with `PA_APP_OK` on success.

Specific Error Codes

None.

Details

The maximum payload size for any channel can be up to 256 bytes

The maximum virtual wire count is a 0-based count which 0 means 1.

`PA_ESPI_ALERT_IO1` for the alert pin can be assigned for slave 0. In eSPI specification, IO1 can be used as the alert pin when there is only on slave.

Read the operating configuration that eSPI system is using.

Arguments

conn	handle of the connection
slave_id	id of slave.
cfg	filled with values described in Table 15

Return Value

A status code is returned with `PA_APP_OK` on success.

Specific Error Codes

None.

Details

None

Read eSPI (pa_espi_read)

Top

```
int pa_espi_read (PromiraConnectionHandle conn, PromiraReadInfo * info,
PromiraEspiPacketInfo * pkt_info, u32 max_bytes, u08 * packet);
```

Read eSPI data from the analyzer.

Arguments

conn	handle of the connection
info	filled with values described in Table 9
pkt_info	filled with values described in Table 19
max_bytes	maximum number of data bytes to read
packet	an allocated array of `u08` which is filled with the received data

Table 19 : `PromiraEspiPacketInfo` structure

channel	logical channel the packet sent over, described in Table 20

cmd_length	the length of command phase of packet including CRC

Table 20 : `PromiraEspiChannel` enumerated values

`PA_ESPI_CHANNEL_UNKNOWN`	-1	Unknown channel
`PA_ESPI_CHANNEL_PERIF`	0	Peripheral
`PA_ESPI_CHANNEL_VW`	1	Virtual Wire
`PA_ESPI_CHANNEL_OOB`	2	OOB
`PA_ESPI_CHANNEL_FLASH`	3	Flash
`PA_ESPI_CHANNEL_INDEP`	4	Channel-Independent

Table 21 : `PromiraEspiEnumFreq` enumerated values

`PA_ESPI_ENUM_FREQ_UNKNOWN`	-1	Unknown
`PA_ESPI_ENUM_FREQ_20M`	0	20 MHz
`PA_ESPI_ENUM_FREQ_25M`	1	25 MHz
`PA_ESPI_ENUM_FREQ_33M`	2	33 MHz
`PA_ESPI_ENUM_FREQ_50M`	3	50 MHz
`PA_ESPI_ENUM_FREQ_66M`	4	66 MHz

Return Value

A status code is returned with `PA_APP_OK` on success.

Specific Error Codes

PA_APP_ESPI_READ_EMPTY	No data was seen.

Details

Only when `events` of `info` is `PA_EVENT_ESPI_PACKET`, `pkt_info` will be filled with information.

Configure the eSPI simple matching system for triggering.

Arguments

conn	handle of the connection
act_on	bitmask for trigger conditions described in Table 22
pkt_match	trigger conditions for packet described in Table 23
actions	bitmask for actions when conditions meet, described in Table 25

Table 22 : bitmask for `act_on`

`PA_ESPI_ACT_ON_DIG_IN_MASK`	0x00000fff	Do actions on any given digital inputs
`PA_ESPI_ACT_ON_ALERT`	0x00010000	Do actions on alert
`PA_ESPI_ACT_ON_RESET`	0x00040000	Do actions on reset
`PA_ESPI_ACT_ON_SLAVE_BITMASK`	0xf0000000	Do actions on any events from specified slaves.

Table 23 : `PromiraEspiPacketMatch` structure

`ch_match_bitmask`	Bitmask for channels type described in Table 24
`cmd_match_type`	Command match type described in Table 12
`cmd_match_val`	Command match value

Table 24 : bitmask for `ch_mask`

`PA_ESPI_CHANNEL_MATCH_PERIF`	0x00000001	Act on any packet from peripheral channel

PA_ESPI_CHANNEL_MATCH_FLASH	0x00000008	Act on any packet from flash channel
PA_ESPI_CHANNEL_MATCH_PERIF	0x00000010	Act on any packet from independent channel

Table 25 : bitmask for `actions`

PA_ESPI_ACTION_DIG_OUT_MASK	0x00000fff	Activate digital output
PA_ESPI_ACTION_TRIGGER	0x80000000	Trigger

Return Value

A status code is returned with `PA_APP_OK` on success.

Specific Error Codes

None.

Details

This function is to trigger or activate digital outputs on the specific conditions

Digital output will not activate on digital input.

Configure Hardware Filter (pa_espi_hw_filter_config)

Top

```
int pa_espi_hw_filter_config (PromiraConnectionHandle conn, u08
slave_id, const PromiraEspiPacketMatch * pkt_match);
```

Configure the eSPI hardware filter.

Arguments

conn handle of the connection

A status code is returned with PA_APP_OK on success.

Specific Error Codes

None.

Details

This function is to filter out packets that match specific conditions

Read eSPI statistics (pa_espi_stats_read)

Top

```
int pa_espi_stats_read (PromiraConnectionHandle conn, u08 slave_id,
PromiraEspiStats * stats);
```

Read eSPI hardware statistics from the analyzer.

Arguments

conn	handle of the connection
slave_id	select slave number for statistics
stats	statistical counters based on espi events as described in Table 26

Return Value

A status code is returned with PA_APP_OK on success.

Specific Error Codes

None.

Details

Read ESPI hardware statistics

```
struct PromiraEspiStats { u32 ch_perif u32 ch_vw u32 ch_oob u32
ch_flash u32 get_cfg u32 set_cfg u32 get_sts u32 pltf_reset u32
alert u32 reset u32 cmd_crc u32 resp_crc u32 fltr_out_pkts u32
fltr_out_cmds };
```

`ch_oob`	A count of all OOB channel packets seen on the bus
`ch_flash`	A count of all flash channel packets seen on the bus
`get_cfg`	A count of all GET_CONFIGURATION packets seen on the bus
`set_cfg`	A count of all SET_CONFIGURATION packets seen on the bus
`get_sts`	A count of all GET_STATUS packets seen on the bus
`pltf_reset`	A count of platform reset commands (inband reset) seen on the bus
`alert`	A count of alert events seen on bus
`reset`	A count of reset events seen on bus
`cmd_crc`	A count of all packets with command crc error seen on the bus
`resp_crc`	A count of all packets with response crc error seen on the bus
`fltr_out_pkts`	A count of all packets that were filtered out (based on HW filter configuration)
`fltr_out_cmds`	A count of all packets with a specific command that were filtered out (based on HW filter configuration)

Top

Configure Advanced Trigger (pa_espi_adv_trig_config)

```
int pa_espi_adv_trig_config (PromiraConnectionHandle conn, u08
salve_id, const PromiraEspiAdvancedTrig1 * pkt_adv_trig1_level1, const
PromiraEspiAdvancedTrig1 * pkt_adv_trig1_level2, const
PromiraEspiAdvancedTrig1 * pkt_adv_trig1_level3, const
PromiraEspiAdvancedTrig1 * pkt_adv_trig1_level4, const
PromiraEspiAdvancedTrig2 * pkt_adv_trig2, const
PromiraEspiAdvancedTrigError * pkt_adv_trig_error);
```

Configure the eSPI advanced triggering system.

pkt_adv_trig1_level1	Advanced trigger match conditions for option 1, level1
pkt_adv_trig1_level2	Advanced trigger match conditions for option 1, level2
pkt_adv_trig1_level3	Advanced trigger match conditions for option 1, level3
pkt_adv_trig1_level4	Advanced trigger match conditions for option 1, level4
pkt_adv_trig2	Advanced trigger conditions for option 2
pkt_adv_trig_error	Advanced trigger conditions for trigger on error

Return Value

A status code is returned with `PA_APP_OK` on success.

Specific Error Codes

None.

Details

This function is to trigger and/or activate digital outputs on complex packet matching

Digital output will not activate on digital input.

```
struct PromiraEspiAdvancedTrigBytes16{ u08 byte0 u08 byte1 u08
byte2 u08 byte3 u08 byte4 u08 byte5 u08 byte6 u08 byte7 u08 byte8
u08 byte9 u08 byteA u08 byteB u08 byteC u08 byteD u08 byteE u08
byteF };
```

```
struct PromiraEspiAdvancedTrigBytes2{ u08 byte0 u08 byte1 };
```

```
struct PromiraEspiAdvancedTrig1 { u08 cmd_byte u08 cmd_cyc u08
cmd_tag u16 cmd_len u64 cmd_addr PromiraEspiAdvancedTrigBytes16
cmd_data PromiraEspiAdvancedTrigBytes16 cmd_data_mask u08 rsp_byte
u08 rsp_cyc u08 rsp_tag u16 rsp_len u64 rsp_addr
PromiraEspiAdvancedTrigBytes16 rsp_data
PromiraEspiAdvancedTrigBytes16 rsp_data_mask
PromiraEspiAdvancedTrigBytes2 sts_byte
PromiraEspiAdvancedTrigBytes2 sts_mask u08 trg_pin u08
trg_pin_polarity u08 trg_pin_direction u08 cmd_byte_enable u08
cmd_cyc_enable u08 cmd_tag_enable u08 cmd_len_enable u08
```

Table 27 : `PromiraEspiAdvancedTrig1` field descriptions

`cmd_byte`	Command value
`cmd_cyc`	Command header cycle type value
`cmd_tag`	Command header tag value
`cmd_len`	Command header espi length value
`cmd_addr`	Command header address value
`cmd_data`	Command phase data
`cmd_data_mask`	Command phase bitwise enables for data match
`rsp_byte`	Response value
`rsp_cyc`	Response header cycle type value
`rsp_tag`	Response header tag value
`rsp_len`	Response header espi length value
`rsp_addr`	Response header address value
`rsp_data`	Response phase data
`rsp_data_mask`	Response phase bitwise enables for data match
`sts_byte`	Status field
`sts_mask`	Bitwise enables for status match
`trg_pin`	Select a digital pin number (0 - 10) to be driven on a condition match
`trg_pin_polarity`	Polarity of the digital pin
`trg_pin_direction`	Direction of the digital pin
`cmd_byte_enable`	Enable match on a command
`cmd_cyc_enable`	Enable match on cycle type in command header
`cmd_tag_enable`	Enable match on tag in command header

rsp_byte_enable	Enable match on a response
rsp_cyc_enable	Enable match on cycle type in response header
rsp_tag_enable	Enable match on tag in response header
rsp_len_enable	Enable match on espi length in response header
rsp_addr_enable	Enable match on address in response header
rsp_data_enable	Enable match on data in response phase
sts_byte_enable	Enable match on status in response phase
trg_pin_enable	Enable a digital pin for trigger
lvl_select_enable	Enable this level for matching
lvl_select_immediate	Set in cases where the packet that matches this conditions should immediately follow the packet that matched the previous level's conditions

```
struct PromiraEspiAdvancedTrig2 { u08 cmd_byte2 u08 cmd_cyc2 u08
cmd_tag2 u16 cmd_len2 u64 cmd_addr2 PromiraEspiAdvancedTrigBytes16
cmd_data2 PromiraEspiAdvancedTrigBytes16 cmd_data_mask2 u08
trg_pin_req u08 trg_pin_req_polarity u08 trg_pin_req_direction u08
trg_pin_cmpl0 u08 trg_pin_cmpl0_polarity u08
trg_pin_cmpl0_direction u08 trg_pin_cmpl1 u08
trg_pin_cmpl1_polarity u08 trg_pin_cmpl1_direction u08
trg_pin_cmpl2 u08 trg_pin_cmpl2_polarity u08
trg_pin_cmpl2_direction u08 cmd_byte2_enable u08 cmd_cyc2_enable
u08 cmd_tag2_enable u08 cmd_len2_enable u08 cmd_addr2_enable u08
cmd_data2_enable u08 succ_cmpl_enable u08 unsucc_cmpl_enable u08
trg_pin_req_enable u08 trg_pin_cmpl0_enable u08
trg_pin_cmpl1_enable u08 trg_pin_cmpl2_enable };
```

Table 28 : PromiraEspiAdvancedTrig2 field descriptions

cmd_byte2	Non-posted request command value

`cmd_addr2`	Non-posted request address
`cmd_data2`	Non-posted request completion data
`cmd_data_mask2`	Non-posted request completion bitwise enables for data match
`trg_pin_req`	Select a digital pin number (0 - 10) to be driven on a request packet condition match
`trg_pin_req_polarity`	Polarity of the digital pin
`trg_pin_req_direction`	Direction of the digital pin
`trg_pin_cmpl0`	Select a digital pin number (0 - 10) to be driven on a 'first' completion packet match (if the request has a completion type 'FIRST')
`trg_pin_cmpl0_polarity`	Polarity of the digital pin
`trg_pin_cmpl0_direction`	Direction of the digital pin
`trg_pin_cmpl1`	Select a digital pin number (0 - 10) to be driven on the first 'middle' completion packet match (if the request has a completion type 'MIDDLE')
`trg_pin_cmpl1_polarity`	Polarity of the digital pin
`trg_pin_cmpl1_direction`	Direction of the digital pin
`trg_pin_cmpl2`	Select a digital pin number (0 - 10) to be driven on a 'last' or 'only' completion packet condition match (the the request has a completion type 'LAST/ONLY')
`trg_pin_cmpl2_polarity`	Polarity of the digital pin
`trg_pin_cmpl2_direction`	Direction of the digital pin
`cmd_byte2_enable`	Enable match on request command
`cmd_cyc2_enable`	Enable match on request cycle type
`cmd_tag2_enable`	Enable match on request tag value

succ_cmpl_enable	Enable match on successful completion
unsucc_cmpl_enable	Enable match n ran unsuccessful condition
trg_pin_req_enable	Enable a digital pin for trigger on request
trg_pin_cmpl0_enable	Enable a digital pin for trigger on 'first' completion
trg_pin_cmpl1_enable	Enable a digital pin for trigger on the first 'middle' completion
trg_pin_cmpl2_enable	Enable a digital pin for trigger on 'last' or 'only' completion

```
struct PromiraEspiAdvancedTrigError { u08 err_code u08
err_code_enable };
```

Table 29 : `PromiraEspiAdvancedTrigError` field descriptions

err_code	Error code value for match
err_code_enable	Enable match on packet error status

Top

6.7 Error Codes

Table 30 : eSPI Analysis Application Error Codes

Literal Name	Value	pa_app_status_string() return value
PA_APP_OK	0	ok
PA_APP_UNABLE_TO_LOAD_LIBRARY	-1	unable to load library

PA_APP_INCOMPATIBLE_DEVICE	-5	incompatible device version
PA_APP_COMMUNICATION_ERROR	-6	communication error
PA_APP_UNABLE_TO_OPEN	-7	unable to open device
PA_APP_UNABLE_TO_CLOSE	-8	unable to close device
PA_APP_INVALID_HANDLE	-9	invalid device handle
PA_APP_CONFIG_ERROR	-10	configuration error
PA_APP_MEMORY_ALLOC_ERROR	-11	unable to allocate memory
PA_APP_UNABLE_TO_INIT_SUBSYSTEM	-12	unable to initialize subsystem
PA_APP_INVALID_LICENSE	-13	invalid license
PA_APP_UNKNOWN_PROTOCOL	-14	unknown promira protocol
PA_APP_STILL_ACTIVE	-15	promira still active
PA_APP_INACTIVE	-16	promira inactive
PA_APP_FUNCTION_NOT_AVAILABLE	-17	promira function not available
PA_APP_READ_EMPTY	-18	nothing to read
PA_APP_TIMEOUT	-31	timeout to collect a response
PA_APP_CONNECTION_LOST	-32	connection lost
PA_APP_QUEUE_FULL	-50	queue is full
PA_APP_UNKNOWN_CMD	-83	unknown command sent

7 Electrical Specifications

Top

7.1 DC Characteristics

Table 31 : Absolute Maximum Rating

V_{IO}	Input voltage (5, 7, 8, 11, 13, 21, 23, 25, 27, 29 pins)		-0.5	4.6	V

Table 32 : Operating Conditions

Symbol	Parameter	Conditions and Notes	Min	Max	Units
T_a	Ambient Operating Temperature		10 (50)	35 (95)	C (F)
I_{Core}	Core Current Consumption	(1)		500	mA

Note:

(1) The core current consumption includes the current consumption for the entire internal Promira platform. Typical current consumption example at 5 V with 66 MHz single eSPI master read operation is 340 mA using USB connection. Add 70 mA for operation with gigabit Ethernet connection.

Table 33 : DC Characteristics (1)

Symbol	Parameter	Conditions and Notes	Min	Max	Units
V_{VTGT}	Target power voltage (4 and 6 pins)		3.3	5.0	V
V_{VIO}	IO power voltage (22 and 24 pins)		1.8	1.8	V
V_{IL}	Input low voltage (1, 3 pins)		-0.5	0.45	V
V_{IH}	Input high voltage (1, 3 pins)		1.26	5.5	V
V_{OL}	Output low voltage (1, 3 pins)			0.2	V
V_{OH}	Output high voltage (1, 3 pins)	(1)			V

V_{OL}	Output low voltage (5, 7, 8, 11, 13, 21, 23, 25, 27, 29 pins)			0.5	V
V_{OH}	Output high voltage (5, 7, 8, 11, 13, 21, 23, 25, 27, 29 pins)		1.25		V
V_{IL}	Input low voltage (9, 14, 15, 17, 19, 20, 26, 31, 32, 33 pins)		0.18		V
V_{IH}	Input high voltage (9, 14, 15, 17, 19, 20, 26, 31, 32, 33 pins)		1.62		V
V_{OL}	Output low voltage (9, 14, 15, 17, 19, 20, 26, 31, 32, 33 pins)			0.18	V
V_{OH}	Output high voltage (9, 14, 15, 17, 19, 20, 26, 31, 32, 33 pins)		1.62		V
I_{VTGT}	Target power current (4 and 6 pins)	(2)		50	mA
I_{VIO}	IO power current (22 and 24 pins)	(2)		50	mA
I_I	Input/output current (eSPI pins)			10	mA
I_L	Input/output leakage current (eSPI pins)			100	uA
C_{IN}	Input/output capacitance (eSPI pins)	1 MHz	2	12	pF

Notes:

(1) Outputs are open collector, and therefor they are set by their pull-ups values and pull-ups voltage rail.

(2) Option 1: Two pins have 50 mA each. Option 2: One pin has 100 mA, and one pin has 0 mA, etc. Total current consumption on both pins should not exceed 100 mA.

Table 34 : Current Consumption Calculation Example

		Consumption			
4, 6	V_{TGT}	Target Power	(2)	100	mA
22, 24	V_{IO}	IOPower	(2)	100	mA
15, 31, 17, 19, 21, 23, 20, 25, 27, 29, 33, 26, 32	Reset[0:1], DIO[0:10]	eSPI Reset and Digital IO Signals	(3)	60	mA
	Total Current Consumption For Promira Core and Outputs		(4)	760	mA

Notes:

(1) The core current consumption includes the current consumption for the entire internal Promira platform, but does not include the output signals current consumption.

(2) Option 1: Two pins have 50 mA each. Option 2: One pin has 100 mA, and one pin has 0 mA. Etc. Total current consumption on both pins should not exceed 100 mA.

(3) Option 1: Six pins have 10 mA each. Option 2: One pin has 60 mA, and the other pins have 0 mA. Etc. Total current consumption on all pins should not exceed 60 mA.

(4) If the total current consumption for the Promira platform core and outputs is over 500 mA, then a USB 3.0 port and USB 2.0 cable or Total Phase external AC adapter should be used. A USB 3.0 port supplies up to 900 mA. A USB 2.0 port supplies up to 500 mA. Total Phase external AC adapter supplies up to 1.2 A. In this example the total current consumption for the Promira platform core and outputs is 760 mA, therefor USB 3.0 port and USB 2.0 cable or Total Phase external AC adapter should be used.

Top

7.2 Signal Ratings

The Promira Serial Platform has built-in electrostatic discharge protection to prevent damage to the unit from high voltage static electricity.

8 Legal / Contact

Top

8.1 Disclaimer

Total Phase shall in no event be liable to any party for direct, indirect, special, general, incidental, or consequential damages arising from the use of its site, the software or documentation downloaded from its site, or any derivative works thereof, even if Total Phase or distributors have been advised of the possibility of such damage. The software, its documentation, and any derivative works is provided on an "as-is" basis, and thus comes with absolutely no warranty, either express or implied. This disclaimer includes, but is not limited to, implied warranties of merchantability, fitness for any particular purpose, and non-infringement. Total Phase and distributors have no obligation to provide maintenance, support, or updates.

Information in this document is subject to change without notice and should not be construed as a commitment by Total Phase. While the information contained herein is believed to be accurate, Total Phase assumes no responsibility for any errors and/or omissions that may appear in this document.

Top

8.2 Life Support Equipment Policy

manner, Buyer agrees to indemnify and hold harmless Total Phase, its officers, employees, affiliates, and distributors from any and all claims arising from such use, even if such claim alleges that Total Phase was negligent in the design or manufacture of its product.

Top

8.3 Contact Information

Total Phase can be found on the Internet at http://www.totalphase.com/. If you have support-related questions, please go to the Total Phase support page at http://www.totalphase.com/support/. For sales inquiries, please contact .

promana-v1.10.000.pdf (800 KB)

Related articles

Promira Serial Platform System User Manual