Using HPI in Coprocessor Mode with OTG-Host - AN6010

Introduction

This application note is a guide for getting started using the HPI interface with EZ-Host or EZ-OTG. It illustrates how to interface an external master (CPU, DSP, ASIC, etc.) to the EZ-OTG/EZ-Host Programmable Embedded USB Host/Peripheral Controller utilizing the Host Port Interface (HPI). The EZ-OTG (CY7C67200) and EZ-Host (CY7C67300) will collectively be called OTG-Host throughout the remainder of this document due to their similarity in respect to the HPI. The HPI is a standard 16-bit parallel bus interface to an external master that has been designed to handle the high-bandwidth low-latency demands of today's embedded host applications. To help illustrate use of the HPI, this document provides detailed hardware connectivity, typical data transfer, and will explain the relationship between the two.

HPI Overview

OTG-Host has two main modes of operation: stand-alone mode and coprocessor mode. In coprocessor mode there are three physical interfaces available to the external master: Host Port Interface (HPI), High Speed Serial (HSS), and Serial Peripheral Interface (SPI). This application note addresses the HPI for coprocessor communications with the external master. The HSS and SPI interfaces will be described in a separate application note. Typically, the HPI port is not needed in stand-alone mode although it can be used in conjunction with stand-alone operation. This application note is intended to cover coprocessor mode of operation only. Downloading stand-alone firmware over the coprocessor interface is addressed in a separate application note.

The main difference between coprocessor mode and stand-alone mode is that in coprocessor mode the external microprocessor will build a Transfer Descriptor list (TD_list), send the list to the OTG-Host BIOS to operate on, and then check the status to determine what to do in the next frame. In stand-alone mode this is all handled by the OTG-Host's internal CY16 microprocessor using Cypress's provided frameworks.

The HPI interface is a slave-only interface and provides a hardware interface into the CY16 processor of OTG-Host for coprocessor communications. It does this through a bidirectional Mailbox and Direct Memory Access (DMA). The DMA channel is used to directly access the OTG-Host's internal memory space. While the coprocessor interfaces can do direct memory access it does not support DMA transfers controlled by typical DMA signaling (DREQ/DACK). The Mailbox channel is used for Link Control Protocol (LCP) commands and responses.

The OTG-Host BIOS implements the LCP for coprocessor communications with an external master. Applications can take full advantage of the features implemented by the BIOS or they can implement their own custom protocol, although the LCP protocol has proven to be very effective in most cases.

The key advantage of HPI over the serial interfaces (SPI or HSS) is the speed. The 16-bit parallel interface is capable of transfer rates of 16 MB/s.

Figure 1 shows a typical OTG-Host connection to an external master via the HPI port.

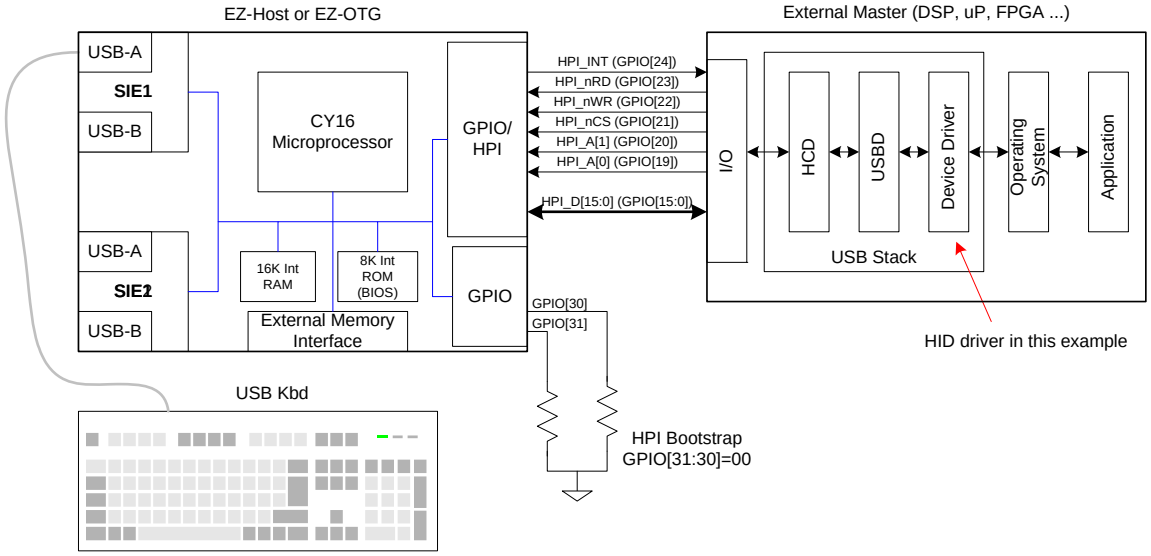

Figure 1. Host Port Interface (HPI)

Table 1 provides a collection of terms used throughout this document.

Table 1. Glossary of Terms

Term	Description
OTG-Host	Refers to either EZ-OTG or EZ-Host
CY16	OTG-Host 16-bit RISC based proprietary microcontroller
External Master	System CPU, DSP, ASIC, etc. that is external to the OTG-Host
USB Keyboard	Computer keyboard that has a USB Peripheral connection
PHY	OTG-Host USB Physical Layer (D+/D– signaling)
SIE 1	OTG-Host USB Serial Interface Engine 1
OTG	On-The-Go, supplement to the USB2.0 specification available at www.usb.org
Internal Memory	OTG-Host Registers, 16 kBytes of internal SRAM, and 8 kBytes internal ROM BIOS
HPI	Host Port Interface
HPI Bootstrap	External pins (GPIO30 and GPIO31) that tell the BIOS how to configure OTG-Host at boot up (power-on or pin reset).
DMA	Direct Memory Access (via HPI ADDRESS and HPI DATA port registers)
USB Stack	HCD, USBD, and Class Driver
HCD	Host Controller Driver
USBD	USB Driver
Class Driver	HID, Mass Storage, Printer, etc. Class drivers are specified by a collective group of contributors from various companies to guarantee interoperability between different manufacturers.
HID	Human Interface Device
OS	Operating System (Linux, WinCE, etc.)
RTOS	Real Time Operating system
App	User Application
BIOS	OTG-Host Basic Input Output System
LCP	BIOS to External Master Link Control Protocol—It supports HPI Transport, HSS Transport, SPI Transport
HPI Transport	ROM BIOS firmware to manage the HPI hardware interface for the LCP
TS	Transaction Structure, made up of a Tdesc and its associated Tdata (optional)
Tdesc	Transaction Descriptor
Tdata	Transaction Data
TB	Transaction Buffer, dynamically allocated internal memory that temporarily stores IN data
TD_list	Transaction Descriptor List, a chain of Transaction Descriptors per USB frame (SOF to EOT)
SOF	USB Start Of Frame
EOT	End Of Transfer, All transactions should be completed by the time EOT is reached.
POR	Power On Reset
GPIO	General Purpose Input Output

HPI Hardware Interface Pins Descriptions

The HPI shares General Purpose Input Output (GPIO) pins with the Integrated Drive Electronics (IDE) subsystems. Bootstrap pins (GPIO [31:30]) must be pulled low to configure OTG-Host to use the HPI port, see *Figure 1* on page 1. These should be pulled low with a resistor since the pins are GPIO pins and can be configured as outputs. During boot-up the BIOS will sense the bootstrap pins and configure the GPIO Control Register (0xC006) for coprocessor mode over the HPI port and then perform initialization for LCP communication over HPI. If a user would like a copy of the BIOS source code, it can be requested through the Cypress Support system.

The OTG-Host HPI port IO signals are 5-volt tolerant, but will only drive to 3.3 volts (see appropriate data sheet for exact details). As long the external master has TTL or 3.3-volt CMOS level inputs the OTG-Host HPI should not require voltage level translation. It must be noted that when using the EZ-OTG (CY7C67200) part, special caution must be taken in regards to GPIO[24:19] and GPIO[15:8]. This is described in the CY7C67200 data sheet in the Reset Pin section and is repeated here for convenience.

The Reset pin is active LOW and requires a minimum pulse duration of 16 12-MHz clock cycles (1.3 ms). A reset event will restore all registers to their default POR settings. Code execution will then begin 200 ms later at 0xFF00 with an immediate jump to 0xE000, which is the start of BIOS. It should be noted that for up to 3 ms after BIOS starts executing, GPIO[24:19] and GPIO[15:8] will be driven as outputs for a test mode. If these pins need to be used as inputs, a series resistor is required (10 ohm–48 ohm is recommended). Please refer to BIOS documentation for addition details.

HPI Port IO Signals

The following are the HPI port IO signals; they are subsequently described.

HPI_INT	HPI interrupt pin.
HPI_nCS	Low asserted chip select for the HPI.
HPI_A[1:0]	Address bits 1 and 0 for the HPI interface to determine the interface mode.
HPI_nWR	Low asserted write enable pin for the HPI interface. Data is latched on rising edge.
HPI_nRD	Low asserted read enable pin for the HPI interface.
HPI_D[15:0]	Data bits 15 through 0 for the HPI interface. HPI is a 16-bit bus only, it does not support an 8-bit mode.

HPI_INT

The interrupt output pin is asserted high by the OTG-Host during a programmable interrupt event. Some examples may include the completion of a transaction or the connection of a new peripheral device. The signal is asserted high until writing to the associated interrupt clearing register in the OTG-Host, which clears the interrupt event. The interrupt polarity is not programmable, so an external inverter may be required if a particular external master does not support active HIGH interrupt signaling.

Note that the OTG-Host is a level-triggered interrupt. In other words the HPI_INT pin will stay asserted until all of the events that caused the interrupt are attended to. If the external master uses an edge-triggered interrupt, caution must taken to guarantee interrupt events are not missed. It is therefore recommended that the external master implements a level-triggered interrupt, when possible. This scheme will assist the user's application with processing back-to-back interrupt events. To eliminate the possibility of missing an interrupt event (i.e., while servicing another), the external master should parse all pending interrupts before clearing the initial HPI_INT pin interrupt event.

Note: The external master should mask spurious HPI_INT pin activity during POR + 3 ms.

HPI_INT PIN PROGRAMMABLE INTERRUPT EVENTS

A "0" to "1" transition on any bit defined in the HPI STATUS port register will assert the HPI_INT pin. Note that various interrupt sources can be enabled or disabled via the Interrupt Routing Register (0x0142). The external master cannot cause the HPI_INT pin to assert.

There are three methods used by the external master to service the HPI_INT pin event.

1. Any USB related interrupt event is serviced by accessing either the Host1 Status register (0xC090) or the Host2 Status register (0xC0B0). Writing a "1" to the individual interrupt source bit will clear all related status bits (HPI STATUS and CPU Register) and deassert both the external HPI_INT pin event and the internal interrupt event.
2. Reading from either the SIE1msg register (0x0144) or the SIE2msg register (0x0148) will clear all related status bits (HPI STATUS) and deassert the HPI interrupt event.

   Note: The SIE1msg and SIE2msg registers make up a one-way mailbox from the internal CY16 CPU to the external master. For our example, the contents read from these registers can be HUSB_TDListDone (0x1000). Note that the message registers are only relevant when using the HPI interface. See Appendix B for a complete list of messages implemented by the BIOS.
3. Reading from the HPI MAILBOX port register will clear all related status bits (HPI STATUS) and deassert the external HPI_INT pin event.

A "1" to "0" transition on any bit defined in the HPI STATUS port register will deassert the HPI_INT pin. The external master can cause the HPI_INT pin to deassert.

Firmware running on OTG-Host can clear an interrupt event to deassert the external HPI_INT pin event and the internal interrupt event.

HPI_nCS

HPI_nCS is asserted low by the external master to select the HPI for read and write operations. HPI_nCS fundamentally signals that the transaction is intended for OTG-Host as opposed to another IC sharing the same bus. If the OTG-Host is the only IC on the bus, HPI_nCS can be continuously asserted.

HPI A [1:0]

The HPI A[1:0] signals are driven by the external master to access one of four port registers, as shown in *Table 2*.

Table 2. HPI Address Pins to Port Register Map

Port Registers	HPI A [1]	HPI A [0]	Access
HPI DATA	0	0	RW
HPI MAILBOX	0	1	RW
HPI ADDRESS	1	0	W
HPI STATUS	1	1	R

HPI DATA Port Register

WRITE: A data block write by the external master to the OTG-Host internal memory begins by writing the beginning memory address to the HPI ADDRESS port register (HPI A[1:0] = 10), followed by writing the data block contiguously to the HPI DATA port register (HPI A[1:0] = 00). The internal memory address will be automatically incremented with each sequential write of the HPI DATA port register.

READ: A data block read by the external master from the OTG-Host internal memory begins by writing the beginning memory address to the HPI ADDRESS port register (HPI A[1:0] = 10), followed by reading the data block contiguously from the HPI DATA port register (HPI A[1:0] = 00). The internal memory address will be automatically incremented with each sequential read of the HPI DATA port register.

Figure 7 on page 15 shows a typical access to the HPI DATA port register (shows a HPI DATA Write).

HPI MAILBOX Port Register

The external master's access to the BIOS implemented LCP is through the two-way HPI MAILBOX port register. Writing a LCP command to the HPI MAILBOX port register causes an internal interrupt that notifies the on-chip processor a new command is ready to be processed by the BIOS. The BIOS writes a response to the HPI MAILBOX port register, which causes an interrupt to the external processor via the HPI_INT pin. The HPI_INT pin deasserts automatically when the external master reads from the HPI MAILBOX port register. Complete LCP control flow diagrams are shown in the OTG-Host BIOS User Manual.

Figure 8 on page 16 shows a typical access to the HPI MAILBOX port register.

HPI ADDRESS Port Register

The HPI interface pre-fetches data from the on-chip memory system when the HPI ADDRESS port register is loaded, and after every read from the HPI DATA port register. Therefore, reading a block of n words from the HPI port results in n+1 read accesses to the on-chip memory system. The pre-fetch pipeline also delays the read data.

Loading the HPI ADDRESS port register must precede changing read/write direction. The memory addresses are auto-incremented after each access to the HPI DATA port register. Therefore any read or write to a new starting location must be preceded with a write to the HPI Address register to indicate the new starting address location.

Figure 9 on page 16 shows a typical access to the HPI ADDRESS port register.

It is important to note *Table 3*, which describes the memory access range of the HPI. The external master can write and read directly in this range only. LCP commands must be used to access areas outside of these ranges.

Table 3. Address Ranges Accessible Directly Over HPI

Address Range	Purpose
0x0000 – 0x3FFF	Internal RAM
0xC080 – 0xC09F	SIE1
0xC0A0 – 0xC0BB	SIE2
0xE000 – 0xFFFF	Internal ROM

HPI STATUS Port Register

This port register reports the status of OTG-Host. *Table 4* shows the grouping of interrupt sources available to the external master in the HPI STATUS port register.

Table 4. HPI STATUS Bit Fields

Bit16	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
VBUS Flag	ID Flag	Reserved	SOF/EOP2 Flag	Reserved	SOF/EOP1 Flag	Reset2 Flag	Mailbox IN Flag
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
Resume2 Flag	Resume1 Flag	SIE2msg	SIE1msg	Done2 Flag	Done1 Flag	Reset1 Flag	Mailbox OUT Flag

Figure 10 on page 17 shows a typical read from the HPI STATUS port register.

HPI_nWR

HPI_nWR is an active low signal driven by the external master during a write transaction. During a write transaction HPI_nCS must also be asserted in order for the OTG-Host to recognize the assertion of the HPI_nWR signal. HPI_nWR is asserted low for a minimum of t_{WP}. Data is written from the external master to the OTG-Host on the rising edge of HPI_nWR. The data must remain valid on the bus for t_{WDH} after HPI_nWR is deasserted in order for it to be properly latched by OTG-Host. The minimum spacing between HPI_nWR assertions is t_{CYC}.

HPI_nRD

HPI_nRD is an active low signal driven by the external master during a read transaction. During a read transaction HPI_nCS must also be asserted in order for the OTG-Host to recognize the assertion of the HPI_nRD signal. The minimum width of the HPI_nRD pulse is t_{RP}. T_{ACC} after the assertion of HPI_nRD the HPI D[15:0] signals switch from high impedance to driving mode and drive the data bus until t_{RDH} after HPI_nRD or HPI_nCS is deasserted. The minimum spacing in between HPI_nRD assertions is t_{CYC}.

Note: Please refer to the current EZ-Host or EZ-OTG data sheet for complete HPI timing parameter specifications.

HPI_D[15:0]

The OTG-Host bidirectional data bus is used to transfer data in and out of registers or memory. The data bus is normally held in a high-impedance state unless both HPI_nCS and HPI_nRD are asserted during a read transaction.

The maximum data transfer rate is one word every 6T, where T is 1/48 MHz, resulting in a rate of 16 Mbytes/second.

Basic OTG-Host HPI Schematic

Figure 2 illustrates the minimum hardware requirements for a typical application utilizing the HPI.

BIOS Boot Procedure

During boot-up the BIOS examines the status of GPIO[31:30] pins to determine which mode to configure itself. The four bootstrap modes are stand-alone, HPI, HSS and SPI. For HPI mode, these two pins are pulled low. These pins should be pulled low with a resistor since the pins are GPIO pins and thus can be configured as outputs. Connecting these pins directly to ground is not recommended.

Note: GPIO[31:30] are sampled by the BIOS during boot-up or after nRESET is deasserted. After boot-up these pins are available to the application as general purpose I/O.

A complete description of the boot up process can be found in the OTG-Host BIOS User Manual. While this application note mainly addresses co-processor mode, where the external processor interacts with the BIOS to initiate USB traffic, the OTG-Host products can also operate in stand-alone mode. Stand-alone code can be compiled separately and downloaded over the HPI interface. Downloading firmware over the HPI interface is commonly done when stand-alone firmware interacts with an external processor over the HPI interface. The method of accomplishing this is addressed in a separate application note.

Note: After a reset pin event occurs, the BIOS boot-up procedure executes for up to 3msec. Check the data sheet for the part you are using to understand interface interactions after an assertion of nRESET. For example, the EZ-OTG part will drive some of the HPI pins after reset. In some designs this can cause bus contentions and undesirable results.

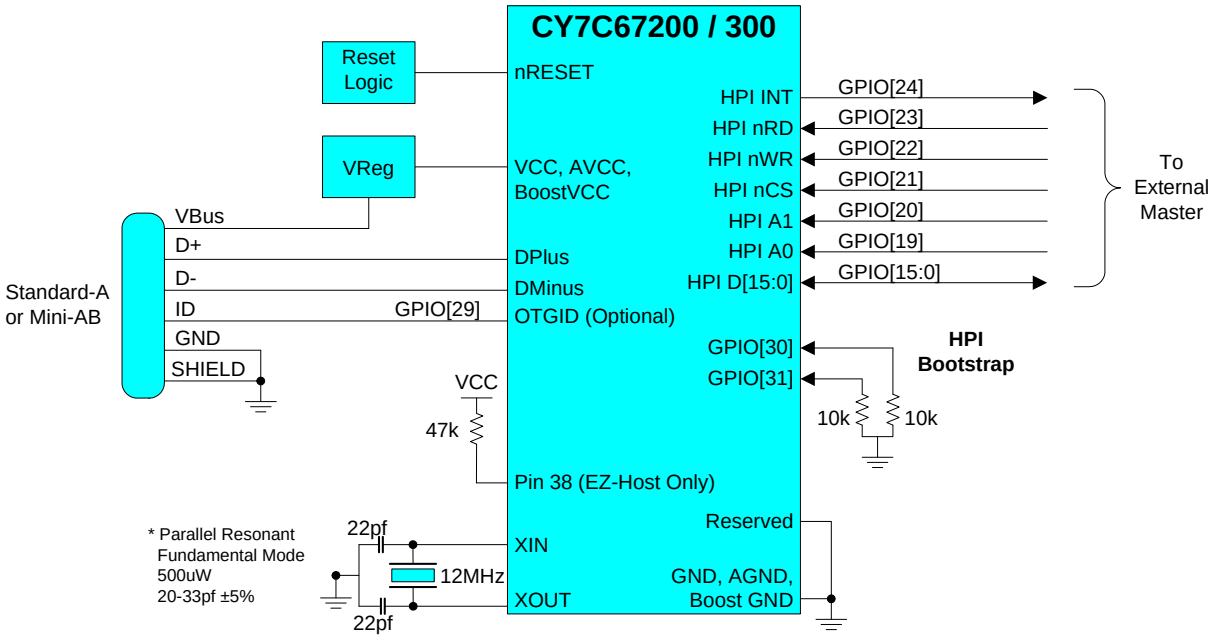

Figure 2. Example OTG-Host HPI Schematic

BIOS Interaction

OTG-Host has 8 KBytes of internal ROM containing the BIOS firmware. The BIOS firmware was written in assembly and is very compact code. The BIOS firmware source can be requested from Cypress through standard support procedures. The BIOS implements the HPI Transport, which exposes the LCP via the HPI hardware interface. The transport is capable of receiving LCP commands from an external master and sending back response via the HPI MAILBOX.

LCP adds the ability for the external master to access any CPU register and to access the external memory space of the CY16 processor.

The LCP is primarily used in co-processor mode embedded host or peripheral applications. Standalone applications will typically not use LCP, although they can. To understand how to enable the LCP processor when running in stand-alone mode please request a copy of the BIOS source code from Cypress support. This firmware example will show how to configure and initialize the LCP.

Figure 3 shows an example of issuing a command to the OTG-Host over the HPI interface using LCP. This is accomplished by writing a command (COMM_RESET in this example) to the HPI MAILBOX port register, which causes an internal interrupt that notifies the on-chip processor (BIOS) a new command is ready to be processed.

The on-chip processor writes a response (LCP) to the HPI MAILBOX port register to cause an external interrupt event (HPI_INT asserts). The HPI_INT pin de-asserts automatically when the external master reads from the HPI MAILBOX port register. Complete LCP control flow diagrams for all LCP commands are shown in the OTG-Host BIOS User Manual.

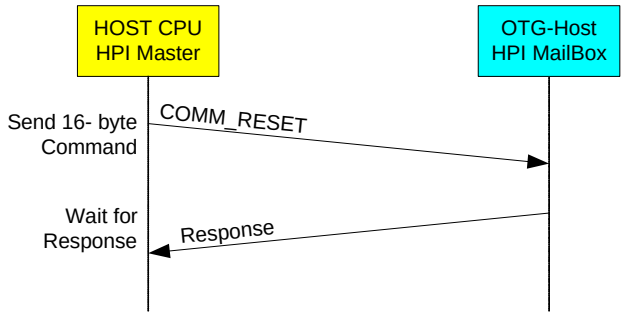

Figure 3. LCP Flow Control Diagram

The BIOS always enables the debug UART, GPIO[28:27] at boot up time. The UART is normally dedicated for serial debug, which is described in the BIOS User Manual.

It is important to note that the BIOS will configure the SIEs in the following manner at power up:

- In coprocessor mode: BIOS will not configure either PORT1A or PORT2A
- In standalone mode: BIOS configures:
  - PORT1A as a peripheral with the D+ pull-up resistor enabled if the OTG_ID pin = “1”
  - PORT1A as a host if the OTG_ID pin = “0”
  - PORT2A as a peripheral and connected (D+ pull-up enabled).
- The BIOS does not support PORT1B and PORT2B automatically at boot time.

USB Stack

HCD

The HCD builds a Transaction Descriptor List (TD_List) for each frame. The TD_List is first loaded into the OTG-Host’s on-chip memory. The OTG-Host device then transfers the data associated with this TD_List to or from USB.

The HCD is informed of the previous transaction completion via the SIE mailbox at which time it checks the TD_List status. The HCD then builds a new TD_List for the next frame and loads it into the OTG-Host device. While the TD_List transfer is executed, the HCD can copy the previous frame’s IN data from the OTG-Host part.

The LCP is the low-level hardware abstraction layer implemented within the HCD.

Note: Typically, this layer will need to be ported for each new hardware/software platform.

USBD

Typically, the Operating Systems (OS) ships with the USBD layer implemented. If the system being designed does not use an RTOS that includes the USBD, it will need to be done by the developer. The stand-alone frameworks provided with the OTG-Host Kits can be an essential resource for this effort.

Device Driver: This can include any type of USB peripheral device and may include Class devices such as HID, Mass Storage, Printer, etc.

Example EZ-Host/EZ-OTG HPI Registers

Table 5 gives a brief summary of typical registers accessed by both the external master and the on-chip CY16 microcontroller during HPI transactions. Other important registers and address locations are summarized in Appendix A of this document.

Table 5. Registers Associated with the HPI Port

Register Name	CY16 Access	External Access
Interrupt Routing Register	0x0142	via DMA
SIE1msg Register[1]	0x0144	via DMA
SIE2msg Register[1]	0x0148	via DMA
Host1 Status register	0xC090	via DMA
Host2 Status register	0xC0B0	via DMA
HPI MAILBOX Port Register	0xC0C6	HPI A[1:0]=01
HPI STATUS Port Register	NA	HPI A[1:0]=11

Table 5. Registers Associated with the HPI Port

Note
1. SIE1msg and SIE2msg values are BIOS dependent.

Example Data Transfer to Enumerate a USB Keyboard

Below are the steps required to enumerate a USB keyboard using OTG-Host. This section starts from a high level view of HPI bus transactions, describes the USB packet level transfers and then shows the step by step instructions of how to accomplish the tasks. Keep in mind that this particular example is basic and does not show all of the steps that might normally be done. For example getting the insert interrupt, determining the speed of the attached device and completing the full enumeration. There are systems where code space is a premium and the device to be used is well enough known that some steps can be eliminated.

Figure 4 is a diagram of the HPI high level transactions. The first step is an initialization step configuring the part. Step 2 sends the USB reset to the device and starts the USB traffic, enumerating the USB keyboard as seen in steps 3 and 4. These steps use a combination of direct writes to OTG-Host memory locations and LCP commands.

Note: The steps in the text are aligned with the steps in the diagrams.

HPI Transactions (high-level view)

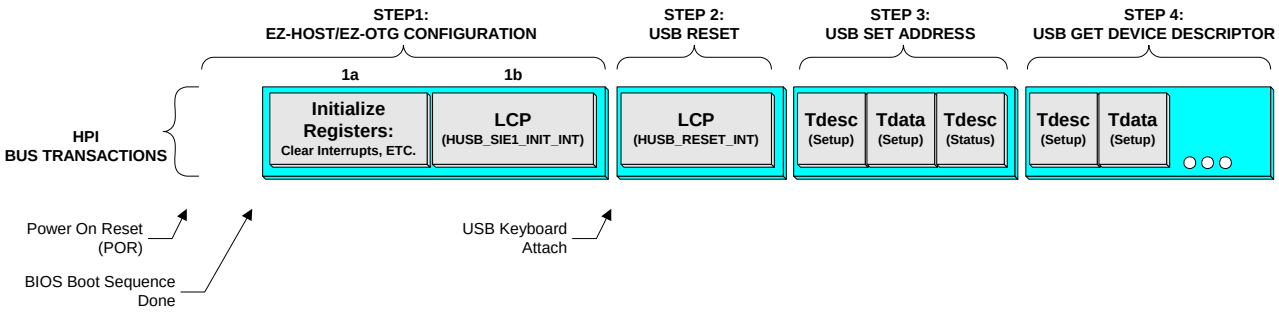

Figure 4. HPI Transactions Time-line for USB Keyboard Enumeration (details below)

Note: *Figure 5* hides the “Keep Alive” packets to reduce clutter.

USB Transfers (associated with HPI Transactions)

Figure 5 correlates the USB traffic with the HPI transactions shown in *Figure 4*. This figure was captured using a CATC™ USB analyzer, which captures the USB traffic. There are several USB protocol analyzers available on the market that provide similar information.

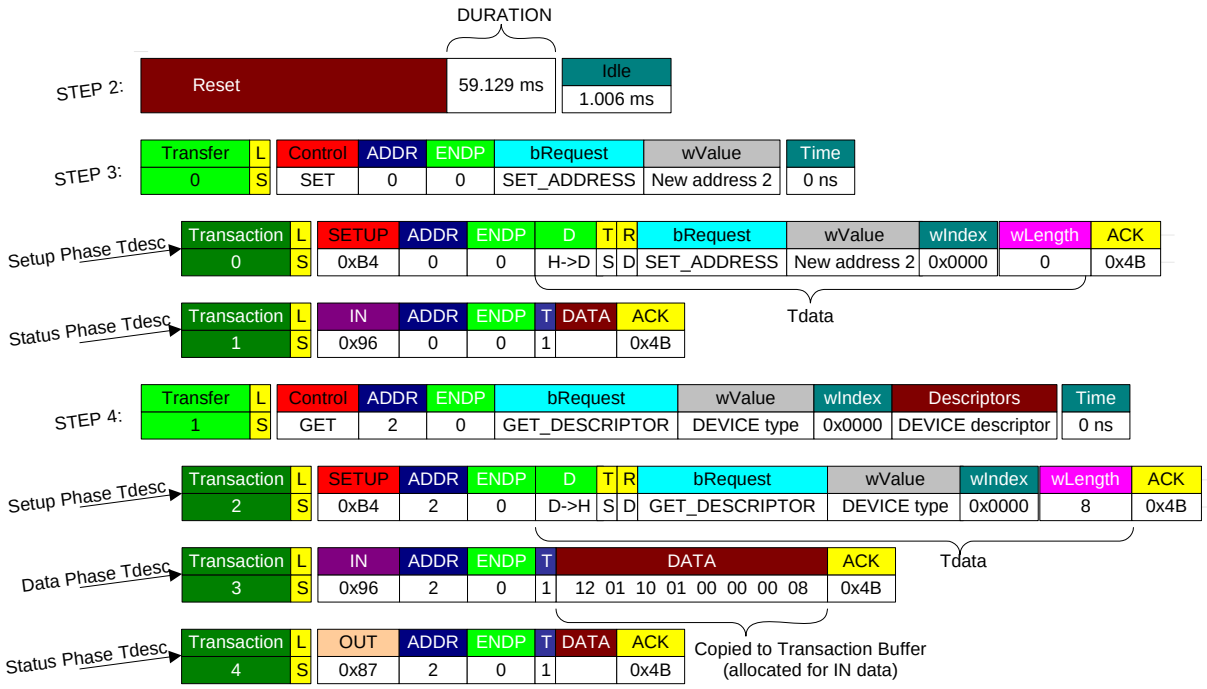

Figure 5. USB Transactions Time-line for USB Keyboard Enumeration (details below)

Step-by-Step HPI Bus Transactions Time-line for USB Keyboard Enumeration Details (low-level view)

STEP 1a:

Initialize EZ-Host/EZ-OTG registers—clear pending interrupts/status, initialize EOT, enable interrupts, etc.

a. Write SIE1msg Register (0x0144) to the HPI ADDRESS port register.

b. Write 0x0000 to the HPI DATA port register to clear SIE1msg register.

c. Write Host 1 Status Register (0xC090) to the HPI ADDRESS port register.

d. Write 0xFFFF to the HPI DATA port register to clear USB interrupts/status.

e. Write HUSB_pEOT (0x01B4) to the HPI ADDRESS port register.

f. Write 0x12C0 (4800 full-speed bit times) to the HPI DATA port register to initialize EOT. This value is user-specific and thus can be adjusted depending on the system.

g. Write Interrupt Routing Register (0x0142) to the HPI ADDRESS port register.

h. Write 0x0C40 (SOF/EOP1 to HPI enable, SOF/EOP1 to CPU enable, Resume1 to HPI enable) to the HPI DATA port register to enable USB interrupts.

i. Write Host 1 Interrupt Enable Register (0xC08C) to the HPI ADDRESS port register.

j. Write 0x0030 (Port A Connect Change Interrupt enable, Port B Connect Change Interrupt enable) to the HPI DATA port register to enable insert and remove interrupts.

Note: The sequence of events outlined in STEP 1a above should be considered as a starting point for your own initialization analysis. Keep in mind initialization is application- and OS-dependent. Another thing to keep in mind is you will likely have a limited number of peripherals you will be supporting and thus can increase or decrease the initialization and enumeration sequence depending on known conditions of your design.

STEP 1b:

To configure SIE1 (both ports) as a HOST the HUSB_SIE1_INIT_INT interrupt will be initiated using the COMM_EXEC_INT LCP command. *Figure 6* shows the LCP flow diagram for the COMM_EXEC_INT. To execute this interrupt, do the following:

k. Write COMM_INT_NUM (0x01C2) to the HPI ADDRESS port register.

l. Write HUSB_SIE1_INIT_INT (0x0072) to the HPI DATA port register.

m. Write COMM_EXEC_INT (0xCE01) to the HPI MAILBOX port register.

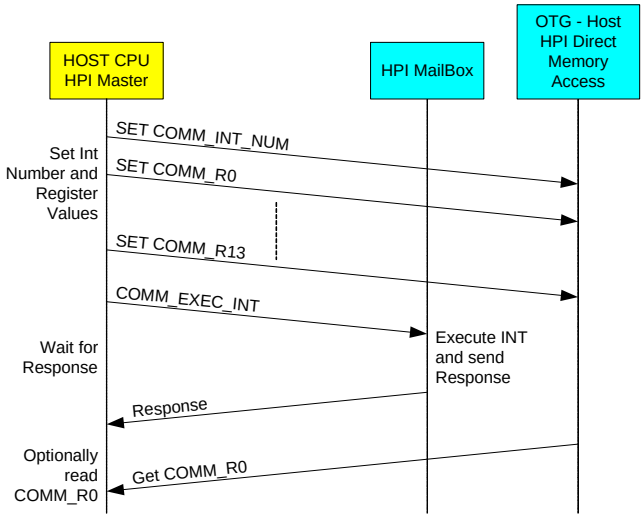

Figure 6. COMM_EXEC_INT HPI Flow Diagram

Background: HUSB_SIEx_INIT_INT is used to execute the Transaction Descriptor List (TD_List). It has the following functions:

- Set SIE as Host and perform initialize: The HUSB_SIEx_INIT_INT sets SIEx as a host and does initialization.
- Check for pending TD_List: At the beginning of every frame, it checks to see if there is a TD_List waiting for transfer. If true, it begins the TD_List transfer.
- Schedule and perform transfer: It transfers all TD data over USB.
- Update status and error handling: It updates the TD status after every transaction. It also does error handling for control and bulk transfers. For ISO and Interrupt transfer errors, it will let the HCD handle the error. The ActiveFlag is not changed to inactive for ISO and Interrupt transfers.
- After the TD_List is finished, the BIOS sends HUSB_TDListDone to the HCD via the SIE mailbox. It also sets a semaphore at HUSB_SIEx_pTDListDone_Sem for the HCD.

Before going on to STEP 2 do the following:

a. Wait for the HPI_INT pin to assert (previous command processed).

b. Read HPI STATUS port register to determine why HPI_INT pin asserted.

c. Read HPI MAILBOX port register to see COMM_ACK (0x0FED) or COMM_NACK (0xDEAD).

Note: Reading HPI MAILBOX port register clears status and deasserts the external interrupt: It is good practice to write R0–R13 even when they are not required for the interrupt. This keeps everything initialized. It is also good practice to read R0 after the interrupt is complete even if it has no relevance to the interrupt. This can be seen in *Figure 6*.

STEP 2:

To issue USB_RESET over SIE1 (Port 0) do the following:

a. Write COMM_INT_NUM (0x01C2) to the HPI ADDRESS port register.

b. Write HUSB_RESET_INT (0x0074) to the HPI DATA port register.

c. Write COMM_R0 (0x01C4) to the HPI ADDRESS port register.

d. Write DURATION (0x003C) to the HPI DATA port register.

e. Write COMM_R1 (0x01C6) to the HPI ADDRESS port register.

f. Write the port number (0x0000 - for this example) to the HPI DATA port register.

g. Write COMM_EXEC_INT (0xCE01) to the HPI MAILBOX port register.

Background: HUSB_RESET_INT performs three functions:

- USB Reset: Before accessing a USB device, the HUSB_RESET_INT will generate a USB reset, which forces the peripheral device to its default address of zero. After USB reset, configuration software can read the device's descriptor at the default address.
- Speed Detect: The HUSB_RESET_INT will detect the full/low speed of the attached device and then return the port status: FULL SPEED, LOW SPEED or NO DEVICE.
- SOF/EOP Generation: Based on the device speed HUSB_RESET_INT will generate SOF for full speed and EOP for low speed. If no device is attached on this port, there will be no SOF/EOP.

STEP 3:

To issue USB_SET_ADDRESS over SIE1 (Port 0) do the following:

a. External master creates the following Tdesc (0x050C, 0x0008, 0x00D0, 0x0001, 0x0013, 0x0514). The definition for each of the fields is described in the BIOS User Manual and the appropriate data sheet and is summarized here:

Table 6. SET ADDRESS Setup Phase Tdesc (12-bytes) Contents

Field Description		Value for USB Keyboard Example
0x00-01	Base Address of Data Buffer (BaseAddress)	0x050C
0x02-03	Port Number and Data Length	0x0008
0x04	PID (SETUP) and Endpoint Number (0)	0xD0
0x05	Device Address (DevAdd)	0x00
0x06	TD Control	0x01
0x07	Transaction Status	0x00

Table 6. SET ADDRESS Setup Phase Tdesc (12-bytes) Contents

0x08	Active Flag, Transfer Type, Retry Count	0x13
0x09	Residue	0x00
0x0A-0B	Pointer to Next TD (NextTDPointer)	0x0514

When writing the TD descriptor, caution must be taken to make sure the byte fields are written into memory correctly. The following table shows how this is done for the CY16, which is a little-endian processor.

Table 7. Little Endian Word values for Set ADDRESS Tdesc

Byte Address	Byte Value	Word Value	Byte Address	Byte Value	Word Value
0x0500	0x0C	0x050C	**0x0506**	0x01	0x0001
0x0501	0x05		0x0507	0x00	
0x0502	0x08	0x0008	**0x0508**	0x13	0x0013
0x0503	0x00		0x0509	0x00	
0x0504	0xD0	0x00D0	**0x050A**	0x14	0x0514
0x0505	0x00		0x050B	0x05	

b. External master creates the following Tdata starting at location 0x050C because this is where the Base Address of the Data Buffer was pointed to in the previous Tdesc: (0x0500, 0x0002, 0x0000, 0x0000).

Table 8. SET ADDRESS Setup Phase Tdata (8-bytes)

Field Description		Value for USB Keyboard Example
0x00	bmRequestType	0x00
0x01	bRequest	0x05
0x02-03	wValue	0x0002
0x04-05	wIndex	0x0000
0x06-07	wLength	0x0000

Table 9. Little-Endian Word Values for SET ADDRESS Tdata

Byte Address	Byte Value	Word Value	Byte Address	Byte Value	Word Value
0x050C	0x00	0x0500	**0x0510**	0x00	0x0000
0x050D	0x05		0x0511	0x00	
0x050E	0x02	0x0002	**0x0512**	0x00	0x0000
0x050F	0x00		0x0513	0x00	

c. External master creates the following Tdesc for the Status Phase, which is a zero length packet with an IN PID (0x0000, 0x0000, 0x0090, 0x0041, 0x0013, 0x0000):

Table 10.SET ADDRESS Status Phase Tdesc Contents (12 Bytes)

Field Description		Value for our USB Keyboard Example
0x00-01	Base Address of Data Buffer (BaseAddress)	0x0000 (don't care)
0x02-03	Port Number and Data Length	0x0000
0x04	PID (IN) and Endpoint Number (0)	0x90
0x05	Device Address (DevAdd)	0x00
0x06	TD Control	0x41
0x07	Transaction Status	0x00
0x08	Active Flag, Transfer Type, Retry Count	0x13
0x09	Residue	0x00
0x0A-0B	Pointer to Next TD (NextTDPointer)	0x0000 (NULL)

Table 11.Little-Endian Word Values for Status Phase Tdesc

Byte Address	Byte Value	Word Value	Byte Address	Byte Value	Word Value
0x0514	0x00	0x0000	**0x051A**	0x41	0x0041
0x0515	0x00		0x051B	0x00	
0x0516	0x00	0x0000	**0x051C**	0x13	0x0013
0x0517	0x00		0x051D	0x00	
0x0518	0x90	0x0090	**0x051E**	0x00	0x0000
0x0519	0x00		0x051F	0x00	

Note: Contiguously arrange the TL above to take advantage of the HPI ADDRESS auto-increment feature.

d. Write base address of TD_List (0x0500) to the HPI ADDRESS port register. This is the starting address location where the TD_List will be placed in OTG-Host's internal memory.

e. Write the entire contents of TD_List (0x050C, 0x0008, 0x00D0, 0x0001, 0x0013, 0x0514, 0x0500, 0x0002, 0x0000, 0x0000, 0x0000, 0x0000, 0x0090, 0x0041, 0x0013, 0x0000) to the HPI DATA port register to copy it into the OTG-Host's internal memory at offset 0x0500.

f. Write HUSB_SIE1_pCurrentTDPtr (0x01B0) to the HPI ADDRESS port register so the address of our TD_List can be written to HUSB_SIE1_pCurrentTDPtr, which will submit the TD_List.

g. Write base address of TD_List (0x0500) to the HPI DATA port register to submit the TD_List.

h. Wait for the HPI_INT pin to assert (previous TD_List completed).

i. Read HPI STATUS port to determine why HPI_INT pin asserted.

j. Assuming that the SIE1msg bit is set in the HPI STATUS register (see *Table 4* on page 4), write SIE1msg register (0x0144) to the HPI ADDRESS port register so we can read the message that the BIOS wrote.

k. Again, assuming the TD_List completed without errors, you should read HUSB_TDListDone (0x1000) from the HPI DATA port register. Reading this will clear status and interrupts.

l. Now we want to read back the Status Byte (byte 0x07) and Active flag in RetryCnt (byte 0x08). In some implementations the entire TD_List is read back and parsed for convenience because it can be one simple function and you may need information from other bytes in the TD_List anyway. For this simple example we will just read back the information we need at the moment. To read back the Status Byte write the address of 0x0506 to the HPI ADDRESS port register.

m. Read from the HPI DATA port register to check the status of the first TS. Since we read back words and the CY16 is little endian, the word read back will contain the Status byte and the Control byte.

n. Again read from the HPI DATA port. Since reads are auto-incremented it is not required to write the HPI ADDRESS again but you can if it is more code efficient in your application. This time you will read back the word from address 0x0508, which will contain the Residue Byte and the RetryCnt bytes. The Active Flag in the RetryCnt byte should be cleared and ideally the Retry Bits will still be equal to 3 indicating that no retries were required.

o. Steps l through n should be repeated for each Tdesc in the list and appropriate action taken depending on the results.

This completes the SET_ADDRESS for step 3. Refer to *Figure 5* on page 8 to correlate the steps that have been taken thus far. The device should now have a USB device address of 2 so all further communications to this device will be at address 2.

STEP 4:

The next step in our example is to send a GET DEVICE DESCRIPTOR to the keyboard. The main difference, as compared to the previous TD_List, is that we have three phases. Referring again to *Figure 5* on page 8, in step 4 we see that there is a setup phase, a data phase, and a status phase. The setup phase will require data to be sent as part of the transfer. This data is pointed to in BaseAddress of the Tdesc. The next Tdesc, will initiate the IN Transfer to actually get the data. This Tdesc, will point to buffer space (BaseAddress) where the returned data from the IN Token is placed. The third Tdesc will send the status phase, which is a zero length OUT Token. To issue USB_GET_DEVICE_DESCRIPTOR over SIE1 (Port 0) do the following:

a. Create a SETUP Tdesc as shown in *Table 12* (0x050C, 0x0008, 0x02D0, 0x0001, 0x0013, 0x0514):

Table 12.GET DEVICE DESCRIPTOR Setup Phase Tdesc Contents (12 Bytes)

Field Description		Value for USB Keyboard Example
0x00-01	Base Address of Data Buffer (BaseAddress)	0x050C
0x02-03	Port Number and Data Length	0x0008
0x04	PID (SETUP) and Endpoint Number (0)	0xD0
0x05	Device Address (DevAdd)	0x02
0x06	TD Control	0x01
0x07	Transaction Status	0x00
0x08	Active Flag, Transfer Type, Retry Count	0x13
0x09	Residue	0x00
0x0A-0B	Pointer to Next TD (NextTDPointer)	0x0514

Table 13.Little endian word values for GET DEVICE DESCRIPTOR

Byte Address	Byte Value	Word Value	Byte Address	Byte Value	Word Value
0x0500	0x0C	0x050C	**0x0506**	0x01	0x0001
0x0501	0x05		0x0507	0x00	
0x0502	0x08	0x0008	**0x0508**	0x13	0x0013
0x0503	0x00		0x0509	0x00	
0x0504	0xD0	0x02D0	**0x050A**	0x14	0x0514
0x0505	0x02		0x050B	0x05	

b. The data that will be sent in the SETUP packet is defined in the USB 2.0 specification. Create the following Tdata (0x0680, 0x0100, 0x0000, 0x0008):

Table 14.GET DEVICE DESCRIPTOR Setup Phase Tdata (8 Bytes)

Field Description		Value for USB Keyboard Example
0x00	bmRequestType	0x80
0x01	bRequest	0x06
0x02-03	wValue	0x0100
0x04-05	wIndex	0x0000
0x06-07	wLength	0x0008

Table 15.GET DESCRIPTOR Data Buffer in Little-Endian Format

Byte Address	Byte Value	Word Value	Byte Address	Byte Value	Word Value
0x050C	0x80	0x0680	**0x0510**	0x00	0x0000
0x050D	0x06		0x0511	0x00	
0x050E	0x00	0x0100	**0x0512**	0x08	0x0008
0x050F	0x01		0x0513	0x00	

c. Next, the external master creates the following Tdesc for the IN Token and points to address 0x052C as the address location to place the data returned from the peripheral (0x052C, 0x0008, 0x0290, 0x0041, 0x0013, 0x0x0528):

Table 16.GET DEVICE DESCRIPTOR Data Phase Tdesc (12 Bytes) Contents

Field Description		Value for our USB Keyboard Example
0x00-01	Base Address of Data Buffer (BaseAddress)	0x052C
0x02-03	Port Number and Data Length	0x0008
0x04	PID (IN) and Endpoint Number (0)	0x90
0x05	Device Address (DevAdd)	0x02
0x06	TD Control	0x41
0x07	Transaction Status	0x00
0x08	Active Flag, Transfer Type, Retry Count	0x13
0x09	Residue	0x00
0x0A-0B	Pointer to Next TD (NextTD-Pointer)	0x0528

Table 17.GET DESCRIPTOR Data Phase Tdesc in Little-Endian Format

Byte Address	Byte Value	Word Value	Byte Address	Byte Value	Word Value
0x0514	0x2C	0x052C	**0x051A**	0x41	0x0041
0x0515	0x05		0x051B	0x00	
0x0516	0x08	0x0008	**0x051C**	0x13	0x0013
0x0517	0x00		0x051D	0x00	
0x0518	0x90	0x0290	**0x051E**	0x28	0x0528
0x0519	0x02		0x051F	0x05	

Note: EZ-Host/EZ-OTG will fill the Transaction Buffer (TB) with the 8 bytes of IN data from the USB Keyboard starting at offset 0x052C, which is after the last Tdesc.

d. The last Tdesc for this TD_List is to send a zero length OUT packet to the device. To do this the external master

creates the following Tdesc (0x0000, 0x0000, 0x0210, 0x0041, 0x0013, 0x0000):

Table 18.GET DEVICE DESCRIPTOR Status Phase Tdesc (12 Bytes) Contents

Field Description		Value for our USB Keyboard Example
0x00-01	Base Address of Data Buffer (BaseAddress)	0x0000 (don't care)
0x02-03	Port Number and Data Length	0x0000
0x04	PID (OUT) and Endpoint Number (0)	0x10
0x05	Device Address (DevAdd)	0x02
0x06	TD Control	0x41
0x07	Transaction Status	0x00
0x08	Active Flag, Transfer Type, Retry Count	0x13
0x09	Residue	0x00
0x0A-0B	Pointer to Next TD (NextTD-Pointer)	0x0000 (NULL)

Table 19.GET DEVICE DESCRIPTOR Status Phase Tdesc in Little-Endian Format

Byte Address	Byte Value	Word Value	Byte Address	Byte Value	Word Value
0x0520	0x00	0x0000	**0x0526**	0x41	0x0041
0x0521	0x00		0x0527	0x00	
0x0522	0x00	0x0000	**0x0528**	0x13	0x0013
0x0523	0x00		0x0529	0x00	
0x0524	0x10	0x0210	**0x052A**	0x00	0x0000
0x0525	0x02		0x052B	0x00	

Note: Contiguously arrange the TL above to take advantage of the HPI ADDRESS auto-increment feature.

e. Write base address of TL (0x0500) to the HPI ADDRESS port, which will be our starting location for the TD_list.

f. Write the entire contents of TL (0x050C, 0x0008, 0x02D0, 0x0001, 0x0013, 0x0514, 0x0680, 0x0100, 0x0000, 0x0008, 0x0520, 0x0008, 0x0290, 0x0041, 0x0013, 0x0x052C, 0x0000, 0x0000, 0x0210, 0x0041, 0x0013, 0x0000) to the HPI DATA port register to copy it into the OTG-Host internal memory at offset 0x0500.

g. Write HUSB_SIE1_pCurrentTDPtr (0x01B0) to the HPI ADDRESS port register.

h. Write base address of TD_List (0x0500) to the HPI DATA port register to submit the TD_List

i. Wait for the HPI_INT pin to assert (previous TD_List completed)

j. Read HPI STATUS port register to determine why HPI_INT pin asserted.

k. Assuming that the status was that there is an SIE1msg, write SIE1msg register (0x0144) to the HPI ADDRESS port register.

l. Read the message value, which should be HUSB_TDListDone (0x1000), from the HPI DATA port register to clear status and interrupts.

m. Now we want to read back the Status Byte (byte 0x07) and Active flag in RetryCnt (byte 0x08). In some implementations the entire TD_List is read back and parsed for convenience because it can be one simple function and you may need information from other bytes in the TD_List anyway. For this simple example we will just read back the information we need at the moment. To read back the Status Byte write the address of 0x0506 to the HPI ADDRESS port register.

n. Read from the HPI DATA port register to check the status of the first TS. Since we read back words and the CY16 is little endian, the word read back will contain the Status byte and the Control byte.

o. Again read from the HPI DATA port. Since reads are auto-incremented it is not required to write the HPI ADDRESS again but you can if it is more code efficient in your application. This time you will read back the word from address 0x0508, which will contain the Residue Byte and the RetryCnt bytes. The Active Flag in the RetryCnt byte should be cleared and ideally the Retry Bits will still be equal to 3 indicating that no retries were required.

p. Steps m through o should be repeated for each Tdesc in the list and appropriate action taken depending on the results.

Note: At this point, if it has been determined that all TS successfully completed then *copy* the contents of the Transaction Buffer (the 8 bytes of IN data starting at 0x052C) to the external master to process.

To complete enumeration we still need to issue the following sequence: GET CONFIGURATION DESCRIPTOR, SET CONFIGURATION, HID Boot Protocol, and HID Set Idle. For device specific information it is recommended that you acquire the current specification found on the USB-IF web site (http://www.usb.org). It is always useful to have a USB analyzer and compare how you are communicating with the peripheral as compared to how a full operating system like Windows XP is communicating with the peripheral.

HPI Breakpoint Register (for debug)

The HPI has a breakpoint register that can be used for debug although usually the developer will use the tools provided for the external system for this. This register is found at address 0x0140 and can be accessed directly by the HPI port. When the program counter matches the address found in the HPI Breakpoint Register, INT 127 will trigger. For more information on this interrupt refer to the BIOS User Manual. For a copy of the debugger stub used by gdb contact Cypress support.

Note: Typically, debug is performed via UART or USB.

Using Asynchronous Ports on an External CPU/DSP

It is important to note that some processors have async ports that will be used for HPI communications. In this case it is important to pay attention to how the master behaves on interrupts and when communicating with other async ports on the same part. In a configuration such as this it is possible that interrupts can happen in the middle of a HPI request and cause undesirable results.

Things to consider when a HPI read/write is in progress:

- No interrupts can start
- If in an interrupt, it cannot end until HPI traffic is completed
- Do not communicate with other async ports.

Appendix A: Logic Analyzer Screens for HPI Communication

Example Transactions

This appendix contains typical signalling (communication) on the HPI for OTG-Host. These examples were captured on a logic analyzer in timing mode when connected to the CY3663 SBC, which is part of the OTG-Host DVK.

HPI DATA

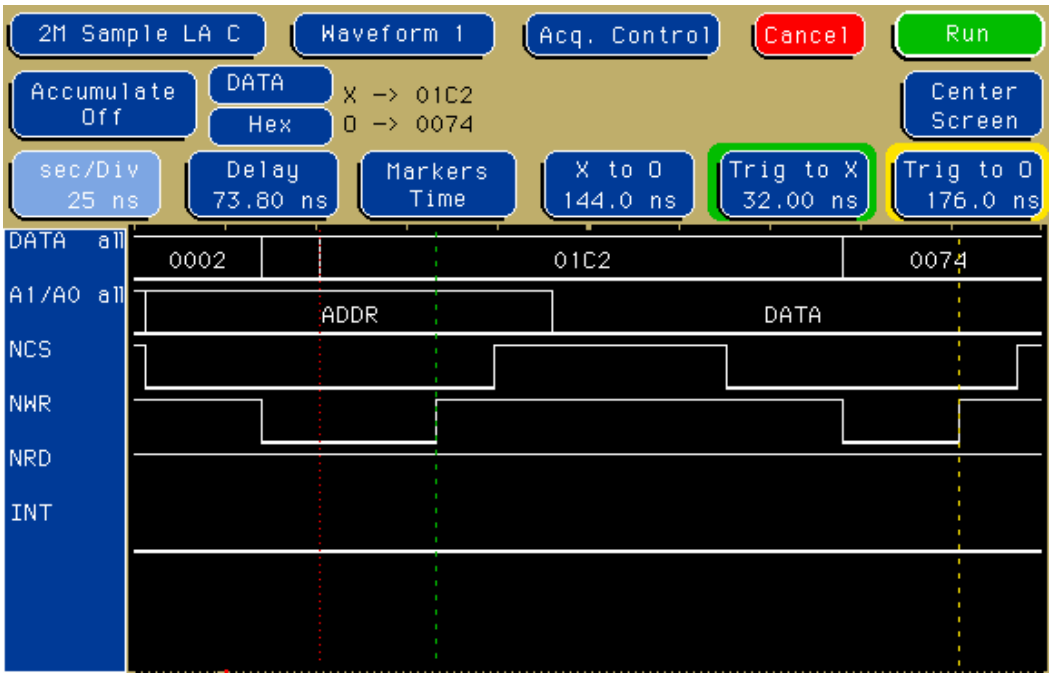

Figure 7. HPI DATA Bus Transaction (Write HUSB_RESET_INT (0x74) to COMM_INT_NUM (0x01C2))

HPI MAILBOX

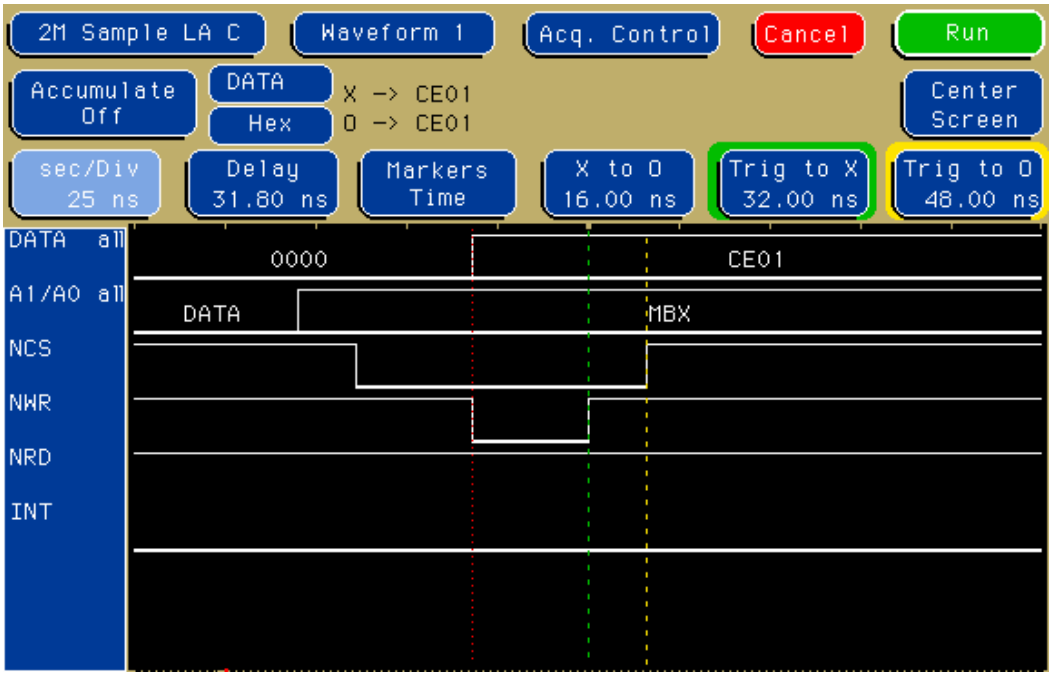

Figure 8. Write to HPI MAILBOX Port Register (Write COMM_EXEC_INT to mailbox)

HPI ADDRESS

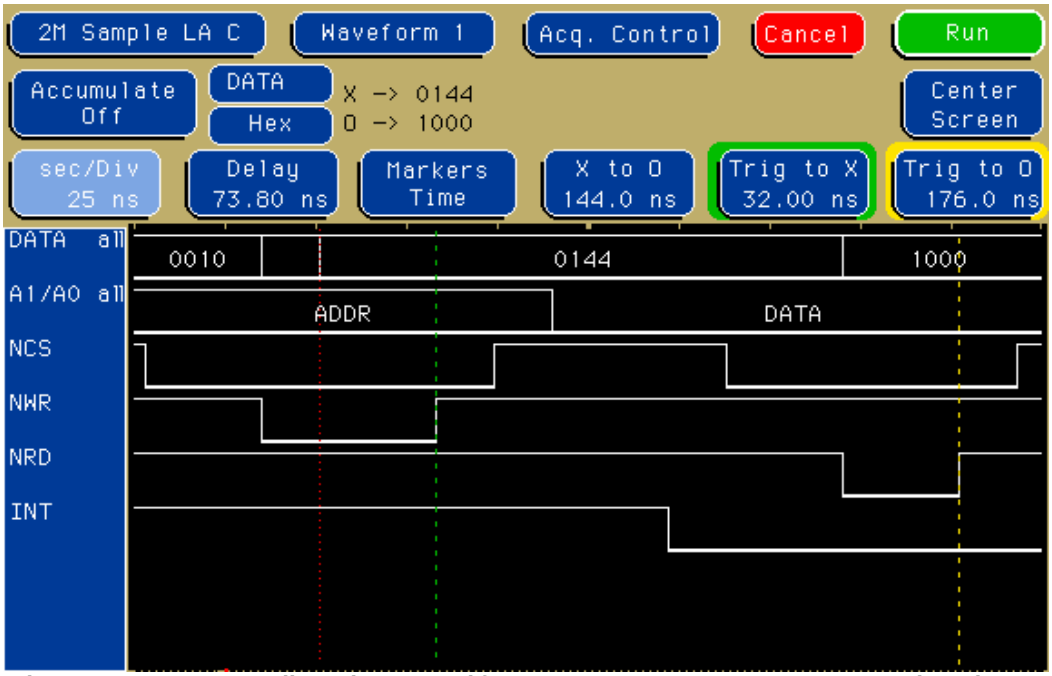

Figure 9. Write to HPI ADDRESS Followed By a Read from HPI DATA to Access SIE1msg Register (0x1000 = DONE)

HPI STATUS

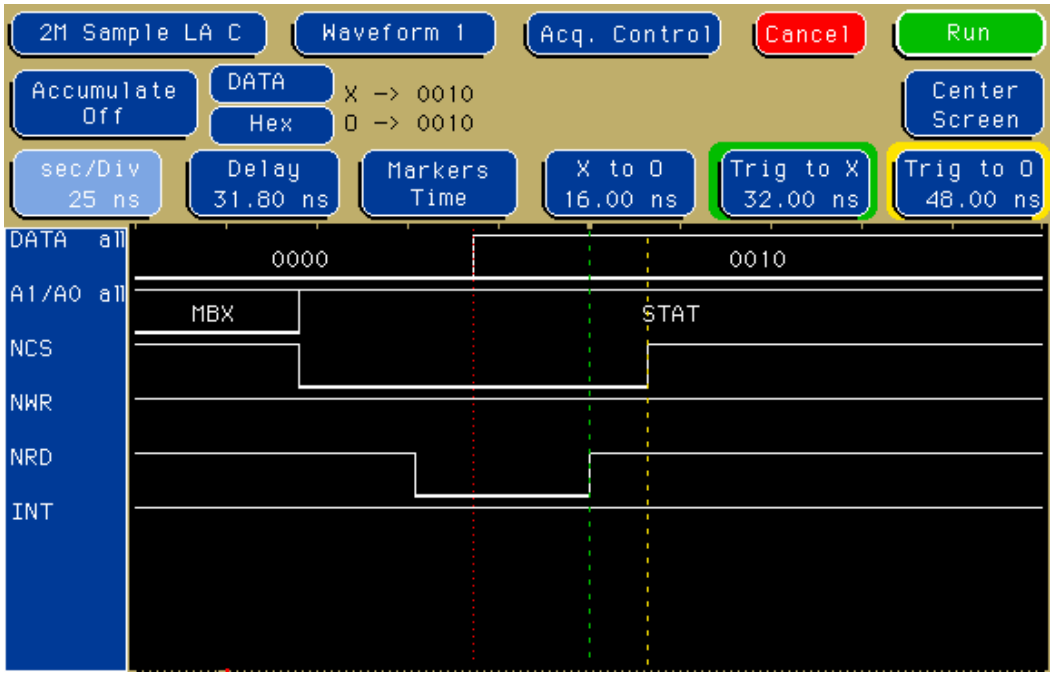

Figure 10. Read from HPI STATUS (0x0010 = SIE1msg)

Appendix B: Important Values used in HPI Communications

OTG-Host LCP equate values from lcp_cmd.h	
OTG-Host Port Commands	**Value**
COMM_RESET	0xFA50
COMM_JUMP2CODE	0xCE00
COMM_EXEC_INT	0xCE01
COMM_READ_CTRL_REG	0XCE02
COMM_WRITE_CTRL_REG	0XCE03
COMM_CALL_CODE	0xCE04
COMM_READ_XMEM	0xCE05
COMM_WRITE_XMEM	0xCE06
C0MM_CONFIG	0xCE07
COMM_READ_MEM	0xCE08
COMM_WRITE_MEM	0xCE09
Responses from LCP	
COMM_ACK	0x0FED
COMM_NAK	0xDEAD
COMM_ASYNC	0xFOOD

Message for SIE1 and SIE2 in register 0x0144 and 0x0148	
Message	**Value**
HUSB_TDListDone	0x1000
HUSB_SOF	0x2000
HUSB_ARMV	0x0001
HUSB_AINS_FS	0x0002
HUSB_AINS_LS	0x0004
HUSB_AWakeUp	0x0008
HUSB_BRMV	0x0010
HUSB_BINS_FS	0x0020
HUSB_BINS_LS	0x0040
HUSB_BWakeUp	0x0080

Message for SIE1 and SIE2 in register 0x0144 and 0x0148	
Message	**Value**
SUSB_EP0_MSG	0x0001
SUSB_EP1_MSG	0x0002
SUSB_EP2_MSG	0x0004
SUSB_EP3_MSG	0x0008
SUSB_EP4_MSG	0x0010
SUSB_EP5_MSG	0x0020
SUSB_EP6_MSG	0x0040
SUSB_EP7_MSG	0x0080
SUSB_RST_MSG	0x0100
SUSB_SOF_MSG	0x0200
SUSB_CFG_MSG	0x0400
SUSB_SUS_MSG	0x0800
SUSB_ID_MSG	0x4000
SUSB_VBUS_MSG	0x8000

Important Memory addresses for host mode		
Name	**Address**	**Comment**
HUSB_SIE1_pCurrentTDPtr	0x01B0	Address for SIE1 current TD pointer
HUSB_SIE2_pCurrentTDPtr	0x01B2	Address for SIE2 current TD pointer
HUSB_pEOT	0x01B4	Address for End of Transfer value
HUSB_SIE1_pTDListDone_Sem	0x01B6	Address for SIE1 TD List Done Semaphore
HUSB_SIE2_pTDListDone_Sem	0x01B8	Address for SIE2 TD List Done Semaphore

Important Memory addresses for LCP commands		
Name	**Address**	**Comment**
COMM_MEM_ADDR	0x01BC	Address for COMM_RD/WR_MEM
COMM_MEM_LEN	0x01BE	Address for COMM_RD/WR_MEM
COMM_LAST_DATA	0x01C0	
COMM_CTRL_REG_ADDR	0x01BC	Address for COMM_RD/WR_CTRL_REG
COMM_CTRL_REG_DATA	0x01BE	Address for COMM_RD/WR_CTRL_REG
COMM_CTRL_REG_LOGIC	0x01C0	Address used for AND/OR
REG_WRITE_FLG	0x0000	Value for COMM_CTRL_REG_LOGIC
REG_AND_FLG	0x0001	Value for COMM_CTRL_REG_LOGIC
REG_OR_FLG	0x0002	Value for COMM_CTRL_REG_LOGIC
COMM_TIMEOUT	0x01BE	Address setting Timeout for sending response to host
COMM_CODE_ADDR	0x01BC	Address for COMM_CALL_CODE and COMM_JUMP2CODE
COMM_INT_NUM	0x01C2	Address used for COMM_EXEC_INT
COMM_R0	0x01C4	
COMM_R1	0x01C6	
COMM_R2	0x01C8	
COMM_R3	0x01CA	
COMM_R4	0x01CC	
COMM_R5	0x01CE	
COMM_R6	0x01D0	
COMM_R7	0x01D2	
COMM_R8	0x01D4	
COMM_R9	0x01D6	
COMM_R10	0x01D8	
COMM_R11	0x01DA	
COMM_R12	0x01DC	
COMM_R13	0x01DE	

Appendix C: Example HPI Communication from Cypress Linux Driver

The following information was captured in state mode on a logic analyzer. The states are captured on the rising and falling edges of either HPI_nRD or HPI_nWR on the Cypress CY3663 EZ-Host Mezzanine board while in HPI co-processor mode. The data is then parsed to remove any activity that is not specific to the HPI interface and then annotated in the right column.

The first example is for a HUSB_INIT_INT (Interrupt 0x72). This is followed by miscellaneous reads and writes including a write to 0xC008 (CPU Speed Register).

MACHINE 1 - State Listing (see step 1b in example)

Label->	nCS	nWR	nRD	A1/A0	D[15:0]	Interrupt		Comment
Base->	Hex	Hex	Hex	Symbol	Hex	Hex		
Write 0x0072 to COMM_INT_NUM								
-62	0	0	1	ADDR	01C2	0	Wrt	Add 0x01C2
-60	0	0	1	DATA	0072	0	Wrt	0x0072 to Add 0x01C2 + 0
Write COMM_R0 through COMM_R13								
-58	0	0	1	ADDR	01C4	0	Wrt	Add 0x01C4
-56	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x01C4 + 0
-54	0	0	1	ADDR	01C6	0	Wrt	Add 0x01C6
-52	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x01C6 + 0
-50	0	0	1	ADDR	01C8	0	Wrt	Add 0x01C8
-48	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x01C8 + 0
-46	0	0	1	ADDR	01CA	0	Wrt	Add 0x01CA
-44	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x01CA + 0
-42	0	0	1	ADDR	01CC	0	Wrt	Add 0x01CC
-40	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x01CC + 0
-38	0	0	1	ADDR	01CE	0	Wrt	Add 0x01CE
-36	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x01CE + 0
-34	0	0	1	ADDR	01D0	0	Wrt	Add 0x01D0
-32	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x01D0 + 0
-30	0	0	1	ADDR	01D2	0	Wrt	Add 0x01D2
-28	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x01D2 + 0
-26	0	0	1	ADDR	01D4	0	Wrt	Add 0x01D4
-24	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x01D4 + 0
-22	0	0	1	ADDR	01D6	0	Wrt	Add 0x01D6
-20	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x01D6 + 0
-16	0	0	1	ADDR	01D8	0	Wrt	Add 0x01D8
-14	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x01D8 + 0
-12	0	0	1	ADDR	01DA	0	Wrt	Add 0x01DA
-10	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x01DA + 0
-8	0	0	1	ADDR	01DC	0	Wrt	Add 0x01DC
-6	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x01DC + 0
-4	0	0	1	ADDR	01DE	0	Wrt	Add 0x01DE
-2	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x01DE + 0
Write COMM_EXEC_INT (0xCE01) to mailbox to initiate interrupt.								
0	0	0	1	MBX	CE01	0	Wrt	HPI Mailbox with 0xCE01
6	0	0	1	ADDR	C08C	0	Wrt	Add 0xC08C

MACHINE 1 - State Listing (continued) (see step 1b in example)								
Label->	nCS	nWR	nRD	A1/A0	D[15:0]	Interrupt		Comment
Base->	Hex	Hex	Hex	Symbol	Hex	Hex		
8	0	1	0	DATA	C800	0	Rd	0xC800 from Add 0xC08C + 0
10	0	0	1	ADDR	C08C	0	Wrt	Add 0xC08C
12	0	0	1	DATA	C800	0	Wrt	0xC800 to Add 0xC08C + 0
68	0	1	0	STAT	0001	1	Rd	HPI Status Reg with 0x0001
Read back HPI mailbox, which equals COMM_ACK (0x0FED)								
70	0	1	0	MBX	0FED	1	Rd	HPI Mailbox = 0x0FED
Write COMM_R0 (0x01C4) with 0x0000								
74	0	0	1	ADDR	01C4	0	Wrt	Add 0x01C4
76	0	1	0	DATA	0000	0	Rd	0x0000 from Add 0x01C4 + 0
78	0	0	1	ADDR	C08C	0	Wrt	Add 0xC08C
80	0	1	0	DATA	C201	0	Rd	0xC201 from Add 0xC08C + 0
82	0	0	1	ADDR	C08C	0	Wrt	Add 0xC08C
84	0	0	1	DATA	C2F1	0	Wrt	0xC2F1 to Add 0xC08C + 0
86	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
Write COMM_CTRL_REG_ADDR (0x01BC) to 0xC008 (CPU Speed Register)								
98	0	0	1	ADDR	01BC	0	Wrt	Add 0x01BC
100	0	0	1	DATA	C008	0	Wrt	0xC008 to Add 0x01BC + 0
Write COMM_CTRL_REG_DATA (0x01BE) to 0x0001								
102	0	0	1	ADDR	01BE	0	Wrt	Add 0x01BE
104	0	0	1	DATA	0001	0	Wrt	0x0001 to Add 0x01BE + 0
Write COMM_CTRL_REG_LOGIC (0x01C0) to 0x0000 (REG_WRITE_FLG)								
106	0	0	1	ADDR	01C0	0	Wrt	Add 0x01C0
108	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x01C0 + 0
Write COMM_WRITE_CTRL_REG (0xCE03) to HPI mailbox								
110	0	0	1	MBX	CE03	0	Wrt	HPI Mailbox with 0xCE03
116	0	0	1	ADDR	C08C	0	Wrt	Add 0xC08C
118	0	1	0	DATA	C2F1	0	Rd	0xC2F1 from Add 0xC08C + 0
120	0	0	1	ADDR	C08C	0	Wrt	Add 0xC08C
122	0	0	1	DATA	C2D1	0	Wrt	0xC2D1 to Add 0xC08C + 0
124	0	1	0	STAT	0001	1	Rd	HPI Status Reg with 0x0001
Read back HPI mailbox, which equals COMM_ACK (0x0FED)								
126	0	1	0	MBX	0FED	1	Rd	HPI Mailbox = 0x0FED
Write COMM_R0 (0x01C4) with 0x0000								
130	0	0	1	ADDR	01C4	0	Wrt	Add 0x01C4
132	0	1	0	DATA	0000	0	Rd	0x0000 from Add 0x01C4 + 0
Write COMM_CTRL_REG_ADDR (0x01BC) to 0xC008 (CPU Speed Register)								
136	0	0	1	ADDR	01BC	0	Wrt	Add 0x01BC
138	0	0	1	DATA	C008	0	Wrt	0xC008 to Add 0x01BC + 0
Write COMM_CTRL_REG_DATA (0x01BE) to 0x0001								
140	0	0	1	ADDR	01BE	0	Wrt	Add 0x01BE

MACHINE 1 - State Listing (continued) (see step 1b in example)								
Label->	nCS	nWR	nRD	A1/A0	D[15:0]	Interrupt		Comment
Base->	Hex	Hex	Hex	Symbol	Hex	Hex		
142	0	0	1	DATA	0001	0	Wrt	0x0001 to Add 0x01BE + 0
Write COMM_CTRL_REG_LOGIC (0x01C0) to 0x0000 (REG_WRITE_FLG)								
144	0	0	1	ADDR	01C0	0	Wrt	Add 0x01C0
146	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x01C0 + 0
Write COMM_WRITE_CTRL_REG (0xCE03) to HPI mailbox								
148	0	0	1	MBX	CE03	0	Wrt	HPI Mailbox with 0xCE03
150	0	1	0	STAT	0100	0	Rd	HPI Status Reg with 0x0100
152	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
162	0	1	0	STAT	0001	1	Rd	HPI Status Reg with 0x0001
Read back HPI mailbox, which equals COMM_ACK (0x0FED)								
164	0	1	0	MBX	0FED	1	Rd	HPI Mailbox = 0x0FED
166	0	0	1	ADDR	01C4	0	Wrt	Add 0x01C4
168	0	1	0	DATA	0000	0	Rd	0x0000 from Add 0x01C4 + 0
170	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000

The following table shows the state listing when doing a Set Address (address = 0x02) to a USB keyboard. Note that in this example the starting location of the buffer is located at address 0x1010. The starting location for the TD_list is at 0x0E10.

MACHINE 1 - State Listing (see step 3 in example)								
Label->	nCS	nWR	nRD	A1/A0	D[15:0]	Interrupt		Comment
Base->	Hex	Hex	Hex	Symbol	Hex	Hex		
-568	0	0	1	ADDR	C090	1	Wrt	Add 0xC090
-566	0	0	1	DATA	0200	1	Wrt	0x0200 to Add 0xC090 + 0
-564	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
-72	0	1	0	STAT	0400	1	Rd	HPI Status Reg with 0x0400
-64	0	0	1	ADDR	C096	1	Wrt	Add 0xC096
-62	0	1	0	DATA	01CB	1	Rd	0x01CB from Add 0xC096 + 0
Write information for Set Address starting at 0x1010								
-2	0	0	1	ADDR	1010	1	Wrt	Add 0x1010
0	0	0	1	DATA	0500	1	Wrt	0x0500 to Add 0x1010 + 0
2	0	0	1	DATA	0002	1	Wrt	0x0002 to Add 0x1010 + 1
4	0	0	1	DATA	0000	1	Wrt	0x0000 to Add 0x1010 + 2
6	0	0	1	DATA	0000	1	Wrt	0x0000 to Add 0x1010 + 3
Write TD_list at address 0x0E10 pointing to the buffer at 0x1010. Setup (D) to EP0 at device address 0.								
8	0	0	1	ADDR	0E10	1	Wrt	Add 0x0E10
10	0	0	1	DATA	1010	1	Wrt	0x1010 to Add 0x0E10 + 0
12	0	0	1	DATA	0008	1	Wrt	0x0008 to Add 0x0E10 + 1
14	0	0	1	DATA	00D0	1	Wrt	0x00D0 to Add 0x0E10 + 2
16	0	0	1	DATA	0001	1	Wrt	0x0001 to Add 0x0E10 + 3
18	0	0	1	DATA	0013	1	Wrt	0x0013 to Add 0x0E10 + 4
20	0	0	1	DATA	0000	1	Wrt	0x0000 to Add 0x0E10 + 5
Write the address of the TD_list (0x0E10) to the SIE1_CurrentTDPtr (0x01B0)								
22	0	0	1	ADDR	01B0	1	Wrt	Add 0x01B0
24	0	0	1	DATA	0E10	1	Wrt	0x0E10 to Add 0x01B0 + 0
28	0	0	1	ADDR	C090	1	Wrt	Add 0xC090
30	0	0	1	DATA	0200	1	Wrt	0x0200 to Add 0xC090 + 0
32	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
1456	0	1	0	STAT	0400	1	Rd	HPI Status Reg with 0x0400
1464	0	0	1	ADDR	C090	1	Wrt	Add 0xC090
1466	0	0	1	DATA	0200	1	Wrt	0x0200 to Add 0xC090 + 0
1468	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
1630	0	1	0	STAT	0010	1	Rd	HPI Status Reg with 0x0010
SIE1msg (0x0144) = 0x1000 or DONE								
1632	0	0	1	ADDR	0144	1	Wrt	Add 0x0144
1634	0	1	0	DATA	1000	0	Rd	0x1000 from Add 0x0144 + 0
1636	0	0	1	ADDR	0144	0	Wrt	Add 0x0144
1638	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x0144 + 0
Read back last TD_list to check status								
1640	0	0	1	ADDR	0E10	0	Wrt	Add 0x0E10

MACHINE 1 - State Listing (continued) (see step 3 in example)								
Label->	nCS	nWR	nRD	A1/A0	D[15:0]	Interrupt		Comment
Base->	Hex	Hex	Hex	Symbol	Hex	Hex		
1642	0	1	0	DATA	1010	0	Rd	0x1010 from Add 0x0E10 + 0
1644	0	1	0	DATA	0008	0	Rd	0x0008 from Add 0x0E10 + 1
1646	0	1	0	DATA	00D0	0	Rd	0x00D0 from Add 0x0E10 + 2
1648	0	1	0	DATA	0001	0	Rd	0x0001 from Add 0x0E10 + 3
1650	0	1	0	DATA	0003	0	Rd	0x0003 from Add 0x0E10 + 4
1652	0	1	0	DATA	0000	0	Rd	0x0000 from Add 0x0E10 + 5
1658	0	0	1	ADDR	C096	0	Wrt	Add 0xC096
1660	0	1	0	DATA	01CC	0	Rd	0x01CC from Add 0xC096 + 0
Write next TD_list at 0x01E0. This time is IN (9) on EP0 at Device Address 0 (Status Stage)								
1662	0	0	1	ADDR	0E10	0	Wrt	Add 0x0E10
1664	0	0	1	DATA	1010	0	Wrt	0x1010 to Add 0x0E10 + 0
1666	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x0E10 + 1
1668	0	0	1	DATA	0090	0	Wrt	0x0090 to Add 0x0E10 + 2
1670	0	0	1	DATA	0041	0	Wrt	0x0041 to Add 0x0E10 + 3
1672	0	0	1	DATA	0013	0	Wrt	0x0013 to Add 0x0E10 + 4
1674	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x0E10 + 5
Submit the TD_list by writing address of TD_list (0x0E10) to SIE1_CurrentTDPtr (0x01B0)								
1676	0	0	1	ADDR	01B0	0	Wrt	Add 0x01B0
1678	0	0	1	DATA	0E10	0	Wrt	0x0E10 to Add 0x01B0 + 0
1682	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
2928	0	1	0	STAT	0400	1	Rd	HPI Status Reg with 0x0400
2930	0	0	1	ADDR	C090	1	Wrt	Add 0xC090
2932	0	0	1	DATA	0200	1	Wrt	0x0200 to Add 0xC090 + 0
2934	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
3032	0	1	0	STAT	0010	1	Rd	HPI Status Reg with 0x0010
SIE1msg (0x0144) = 0x1000 or DONE								
3034	0	0	1	ADDR	0144	1	Wrt	Add 0x0144
3036	0	1	0	DATA	1000	0	Rd	0x1000 from Add 0x0144 + 0
3038	0	0	1	ADDR	0144	0	Wrt	Add 0x0144
3040	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x0144 + 0
Read back last TD_list to check status								
3042	0	0	1	ADDR	0E10	0	Wrt	Add 0x0E10
3044	0	1	0	DATA	1010	0	Rd	0x1010 from Add 0x0E10 + 0
3046	0	1	0	DATA	0000	0	Rd	0x0000 from Add 0x0E10 + 1
3048	0	1	0	DATA	0090	0	Rd	0x0090 from Add 0x0E10 + 2
3050	0	1	0	DATA	0041	0	Rd	0x0041 from Add 0x0E10 + 3
3052	0	1	0	DATA	0003	0	Rd	0x0003 from Add 0x0E10 + 4
3054	0	1	0	DATA	0000	0	Rd	0x0000 from Add 0x0E10 + 5
3064	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
4380	0	1	0	STAT	0400	1	Rd	HPI Status Reg with 0x0400
4382	0	0	1	ADDR	C090	1	Wrt	Add 0xC090
4384	0	0	1	DATA	0200	1	Wrt	0x0200 to Add 0xC090 + 0

MACHINE 1 - State Listing (continued) (see step 3 in example)								
Label->	**nCS**	**nWR**	**nRD**	**A1/A0**	**D[15:0]**	**Interrupt**		**Comment**
Base->	**Hex**	**Hex**	**Hex**	**Symbol**	**Hex**	**Hex**		
4386	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
5818	0	1	0	STAT	0400	1	Rd	HPI Status Reg with 0x0400
5820	0	0	1	ADDR	C090	1	Wrt	Add 0xC090
5822	0	0	1	DATA	0200	1	Wrt	0x0200 to Add 0xC090 + 0
5824	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
7284	0	1	0	STAT	0400	1	Rd	HPI Status Reg with 0x0400
7286	0	0	1	ADDR	C090	1	Wrt	Add 0xC090
7288	0	0	1	DATA	0200	1	Wrt	0x0200 to Add 0xC090 + 0
7290	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
8756	0	1	0	STAT	0400	1	Rd	HPI Status Reg with 0x0400
8758	0	0	1	ADDR	C090	1	Wrt	Add 0xC090
8760	0	0	1	DATA	0200	1	Wrt	0x0200 to Add 0xC090 + 0
8762	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000

The following table shows the state listing when doing a Get Device Descriptor from the USB keyboard at device address 0x02. Note that in this example the starting location of the buffer is located at address 0x1010. The starting location for the TD_List is at 0x0E10.

MACHINE 1 - State Listing (see step 4 in example)								
Label->	**nCS**	**nWR**	**nRD**	**A1/A0**	**D[15:0]**	**Interrupt**		**Comment**
Base->	**Hex**	**Hex**	**Hex**	**Symbol**	**Hex**	**Hex**		
-24	0	1	0	STAT	0400	1	Rd	HPI Status Reg with 0x0400
-22	0	0	1	ADDR	C096	1	Wrt	Add 0xC096
-20	0	1	0	DATA	0170	1	Rd	0x0170 from Add 0xC096 + 0
Start writing of buffer at 0x1010 for a GET_DESCRIPTOR (6), Type = DEVICE (1), wLength = 8								
-2	0	0	1	ADDR	1010	1	Wrt	Add 0x1010
0	0	0	1	DATA	0680	1	Wrt	0x0680 to Add 0x1010 + 0
2	0	0	1	DATA	0100	1	Wrt	0x0100 to Add 0x1010 + 1
4	0	0	1	DATA	0000	1	Wrt	0x0000 to Add 0x1010 + 2
6	0	0	1	DATA	0008	1	Wrt	0x0008 to Add 0x1010 + 3
Write the TD_list to address 0x0E10. This TD_list will do a SETUP (D) to EP0, Device Address 2 and the data for the SETUP packet is located at address 0x1010								
8	0	0	1	ADDR	0E10	1	Wrt	Add 0x0E10
10	0	0	1	DATA	1010	1	Wrt	0x1010 to Add 0x0E10 + 0
12	0	0	1	DATA	0008	1	Wrt	0x0008 to Add 0x0E10 + 1
14	0	0	1	DATA	02D0	1	Wrt	0x02D0 to Add 0x0E10 + 2
16	0	0	1	DATA	0001	1	Wrt	0x0001 to Add 0x0E10 + 3
18	0	0	1	DATA	0013	1	Wrt	0x0013 to Add 0x0E10 + 4
20	0	0	1	DATA	0000	1	Wrt	0x0000 to Add 0x0E10 + 5
Write the TD_list address (0x0E10) to the HUSB_SIE1_pCurrentTDPtr (0x01B0) to submit (Setup Stage)								
22	0	0	1	ADDR	01B0	1	Wrt	Add 0x01B0
24	0	0	1	DATA	0E10	1	Wrt	0x0E10 to Add 0x01B0 + 0
26	0	0	1	ADDR	C090	1	Wrt	Add 0xC090
28	0	0	1	DATA	0200	1	Wrt	0x0200 to Add 0xC090 + 0
30	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
1468	0	1	0	STAT	0400	1	Rd	HPI Status Reg with 0x0400
1470	0	0	1	ADDR	C090	1	Wrt	Add 0xC090
1472	0	0	1	DATA	0200	1	Wrt	0x0200 to Add 0xC090 + 0
1474	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
1636	0	1	0	STAT	0010	1	Rd	HPI Status Reg with 0x0010
Read SIE1msg Register (0x1000 = DONE)								
1638	0	0	1	ADDR	0144	1	Wrt	Add 0x0144
1640	0	1	0	DATA	1000	0	Rd	0x1000 from Add 0x0144 + 0
1642	0	0	1	ADDR	0144	0	Wrt	Add 0x0144
1644	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x0144 + 0
Now read back TD_list to check status								
1646	0	0	1	ADDR	0E10	0	Wrt	Add 0x0E10
1648	0	1	0	DATA	1010	0	Rd	0x1010 from Add 0x0E10 + 0
1650	0	1	0	DATA	0008	0	Rd	0x0008 from Add 0x0E10 + 1
1652	0	1	0	DATA	02D0	0	Rd	0x02D0 from Add 0x0E10 + 2
1654	0	1	0	DATA	0001	0	Rd	0x0001 from Add 0x0E10 + 3

MACHINE 1 - State Listing (continued) (see step 4 in example)								
Label->	nCS	nWR	nRD	A1/A0	D[15:0]	Interrupt		Comment
Base->	Hex	Hex	Hex	Symbol	Hex	Hex		
1656	0	1	0	DATA	0003	0	Rd	0x0003 from Add 0x0E10 + 4
1658	0	1	0	DATA	0000	0	Rd	0x0000 from Add 0x0E10 + 5
1664	0	0	1	ADDR	C096	0	Wrt	Add 0xC096
1666	0	1	0	DATA	0171	0	Rd	0x0171 from Add 0xC096 + 0
Write next TD_list at 0x01E0. This time is IN (9) on EP0 at Device Address 02 (Data Stage)								
1674	0	0	1	ADDR	0E10	0	Wrt	Add 0x0E10
1676	0	0	1	DATA	1010	0	Wrt	0x1010 to Add 0x0E10 + 0
1678	0	0	1	DATA	0008	0	Wrt	0x0008 to Add 0x0E10 + 1
1680	0	0	1	DATA	0290	0	Wrt	0x0290 to Add 0x0E10 + 2
1682	0	0	1	DATA	0041	0	Wrt	0x0041 to Add 0x0E10 + 3
1684	0	0	1	DATA	0013	0	Wrt	0x0013 to Add 0x0E10 + 4
1686	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x0E10 + 5
Write the TD_list address (0x0E10) to the HUSB_SIE1_pCurrentTDPtr (0x01B0) to submit								
1688	0	0	1	ADDR	01B0	0	Wrt	Add 0x01B0
1690	0	0	1	DATA	0E10	0	Wrt	0x0E10 to Add 0x01B0 + 0
1694	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
2952	0	1	0	STAT	0400	1	Rd	HPI Status Reg with 0x0400
2954	0	0	1	ADDR	C090	1	Wrt	Add 0xC090
2956	0	0	1	DATA	0200	1	Wrt	0x0200 to Add 0xC090 + 0
2958	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
3116	0	1	0	STAT	0010	1	Rd	HPI Status Reg with 0x0010
Read SIE1msg Register (0x1000 = DONE)								
3118	0	0	1	ADDR	0144	1	Wrt	Add 0x0144
3120	0	1	0	DATA	1000	0	Rd	0x1000 from Add 0x0144 + 0
3122	0	0	1	ADDR	0144	0	Wrt	Add 0x0144
3124	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x0144 + 0
Now read back TD_list to check status								
3126	0	0	1	ADDR	0E10	0	Wrt	Add 0x0E10
3128	0	1	0	DATA	1010	0	Rd	0x1010 from Add 0x0E10 + 0
3130	0	1	0	DATA	0008	0	Rd	0x0008 from Add 0x0E10 + 1
3132	0	1	0	DATA	0290	0	Rd	0x0290 from Add 0x0E10 + 2
3134	0	1	0	DATA	0041	0	Rd	0x0041 from Add 0x0E10 + 3
3136	0	1	0	DATA	0003	0	Rd	0x0003 from Add 0x0E10 + 4
3138	0	1	0	DATA	0000	0	Rd	0x0000 from Add 0x0E10 + 5
Read Device Descriptor data from buffer (0x1010)								
3140	0	0	1	ADDR	1010	0	Wrt	Add 0x1010
3142	0	1	0	DATA	0112	0	Rd	0x0112 from Add 0x1010 + 0
3144	0	1	0	DATA	0200	0	Rd	0x0200 from Add 0x1010 + 1
3146	0	1	0	DATA	0000	0	Rd	0x0000 from Add 0x1010 + 2
3150	0	1	0	DATA	0800	0	Rd	0x0800 from Add 0x1010 + 3
3154	0	0	1	ADDR	C096	0	Wrt	Add 0xC096
3156	0	1	0	DATA	0172	0	Rd	0x0172 from Add 0xC096 + 0

MACHINE 1 - State Listing (continued) (see step 4 in example)								
Label->	nCS	nWR	nRD	A1/A0	D[15:0]	Interrupt		Comment
Base->	Hex	Hex	Hex	Symbol	Hex	Hex		
Write next TD_list at 0x01E0. This time is OUT (1) on EP0 at Device Address 02 (Status Stage)								
3158	0	0	1	ADDR	0E10	0	Wrt	Add 0x0E10
3160	0	0	1	DATA	1010	0	Wrt	0x1010 to Add 0x0E10 + 0
3162	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x0E10 + 1
3164	0	0	1	DATA	0210	0	Wrt	0x0210 to Add 0x0E10 + 2
3166	0	0	1	DATA	0041	0	Wrt	0x0041 to Add 0x0E10 + 3
3168	0	0	1	DATA	0013	0	Wrt	0x0013 to Add 0x0E10 + 4
3170	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x0E10 + 5
Write the TD_list address (0x0E10) to the HUSB_SIE1_pCurrentTDPtr (0x01B0) to submit								
3172	0	0	1	ADDR	01B0	0	Wrt	Add 0x01B0
3174	0	0	1	DATA	0E10	0	Wrt	0x0E10 to Add 0x01B0 + 0
3176	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
4450	0	1	0	STAT	0400	1	Rd	HPI Status Reg with 0x0400
4452	0	0	1	ADDR	C090	1	Wrt	Add 0xC090
4454	0	0	1	DATA	0200	1	Wrt	0x0200 to Add 0xC090 + 0
4456	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
4554	0	1	0	STAT	0010	1	Rd	HPI Status Reg with 0x0010
Read SIE1msg Register (0x1000 = DONE)								
4556	0	0	1	ADDR	0144	1	Wrt	Add 0x0144
4558	0	1	0	DATA	1000	0	Rd	0x1000 from Add 0x0144 + 0
4560	0	0	1	ADDR	0144	0	Wrt	Add 0x0144
4562	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x0144 + 0
Now read back TD_list to check status								
4564	0	0	1	ADDR	0E10	0	Wrt	Add 0x0E10
4566	0	1	0	DATA	1010	0	Rd	0x1010 from Add 0x0E10 + 0
4568	0	1	0	DATA	0000	0	Rd	0x0000 from Add 0x0E10 + 1
4570	0	1	0	DATA	0210	0	Rd	0x0210 from Add 0x0E10 + 2
4572	0	1	0	DATA	0041	0	Rd	0x0041 from Add 0x0E10 + 3
4574	0	1	0	DATA	0003	0	Rd	0x0003 from Add 0x0E10 + 4
4576	0	1	0	DATA	0000	0	Rd	0x0000 from Add 0x0E10 + 5
4584	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
5904	0	1	0	STAT	0400	1	Rd	HPI Status Reg with 0x0400
5912	0	0	1	ADDR	C096	1	Wrt	Add 0xC096
5914	0	1	0	DATA	0174	1	Rd	0x0174 from Add 0xC096 + 0
Start writing of buffer at 0x1010 for a GET_DESCRIPTOR (6), Type = DEVICE (1), wLength = 18								
5916	0	0	1	ADDR	1010	1	Wrt	Add 0x1010
5918	0	0	1	DATA	0680	1	Wrt	0x0680 to Add 0x1010 + 0
5920	0	0	1	DATA	0100	1	Wrt	0x0100 to Add 0x1010 + 1
5922	0	0	1	DATA	0000	1	Wrt	0x0000 to Add 0x1010 + 2
5924	0	0	1	DATA	0012	1	Wrt	0x0012 to Add 0x1010 + 3
Write the TD_list to address 0x0E10. This TD_list will do a SETUP (D) to EP0, Device Address 2 and the data for the SETUP packet is located at address 0x1010								

MACHINE 1 - State Listing (continued) (see step 4 in example)								
Label->	nCS	nWR	nRD	A1/A0	D[15:0]	Interrupt		Comment
Base->	Hex	Hex	Hex	Symbol	Hex	Hex		
5926	0	0	1	ADDR	0E10	1	Wrt	Add 0x0E10
5928	0	0	1	DATA	1010	1	Wrt	0x1010 to Add 0x0E10 + 0
5930	0	0	1	DATA	0008	1	Wrt	0x0008 to Add 0x0E10 + 1
5932	0	0	1	DATA	02D0	1	Wrt	0x02D0 to Add 0x0E10 + 2
5934	0	0	1	DATA	0001	1	Wrt	0x0001 to Add 0x0E10 + 3
5936	0	0	1	DATA	0013	1	Wrt	0x0013 to Add 0x0E10 + 4
5938	0	0	1	DATA	0000	1	Wrt	0x0000 to Add 0x0E10 + 5
Write the TD_list address (0x0E10) to the HUSB_SIE1_pCurrentTDPtr (0x01B0) to submit (Setup Stage)								
5940	0	0	1	ADDR	01B0	1	Wrt	Add 0x01B0
5942	0	0	1	DATA	0E10	1	Wrt	0x0E10 to Add 0x01B0 + 0
5944	0	0	1	ADDR	C090	1	Wrt	Add 0xC090
5946	0	0	1	DATA	0200	1	Wrt	0x0200 to Add 0xC090 + 0
5948	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
7400	0	1	0	STAT	0400	1	Rd	HPI Status Reg with 0x0400
7402	0	0	1	ADDR	C090	1	Wrt	Add 0xC090
7404	0	0	1	DATA	0200	1	Wrt	0x0200 to Add 0xC090 + 0
7406	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
7570	0	1	0	STAT	0010	1	Rd	HPI Status Reg with 0x0010
Read SIE1msg Register (0x1000 = DONE)								
7572	0	0	1	ADDR	0144	1	Wrt	Add 0x0144
7574	0	1	0	DATA	1000	0	Rd	0x1000 from Add 0x0144 + 0
7576	0	0	1	ADDR	0144	0	Wrt	Add 0x0144
7578	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x0144 + 0
Now read back TD_list to check status								
7580	0	0	1	ADDR	0E10	0	Wrt	Add 0x0E10
7582	0	1	0	DATA	1010	0	Rd	0x1010 from Add 0x0E10 + 0
7584	0	1	0	DATA	0008	0	Rd	0x0008 from Add 0x0E10 + 1
7586	0	1	0	DATA	02D0	0	Rd	0x02D0 from Add 0x0E10 + 2
7588	0	1	0	DATA	0001	0	Rd	0x0001 from Add 0x0E10 + 3
7590	0	1	0	DATA	0003	0	Rd	0x0003 from Add 0x0E10 + 4
7592	0	1	0	DATA	0000	0	Rd	0x0000 from Add 0x0E10 + 5
7594	0	0	1	ADDR	C096	0	Wrt	Add 0xC096
7596	0	1	0	DATA	0175	0	Rd	0x0175 from Add 0xC096 + 0
Write next TD_list at 0x0E10. This time is IN (9) on EP0 at Device Address 02 (Data Stage)								
7664	0	0	1	ADDR	0E10	0	Wrt	Add 0x0E10
7666	0	0	1	DATA	1010	0	Wrt	0x1010 to Add 0x0E10 + 0
7668	0	0	1	DATA	0008	0	Wrt	0x0008 to Add 0x0E10 + 1
7670	0	0	1	DATA	0290	0	Wrt	0x0290 to Add 0x0E10 + 2
7672	0	0	1	DATA	0041	0	Wrt	0x0041 to Add 0x0E10 + 3
7674	0	0	1	DATA	0013	0	Wrt	0x0013 to Add 0x0E10 + 4
7676	0	0	1	DATA	0E1C	0	Wrt	0x0E1C to Add 0x0E10 + 5
Write next TD_list at 0x01EC. This time is IN (9) on EP0 at Device Address 02 (Data Stage)								

MACHINE 1 - State Listing (continued) (see step 4 in example)								
Label->	nCS	nWR	nRD	A1/A0	D[15:0]	Interrupt		Comment
Base->	Hex	Hex	Hex	Symbol	Hex	Hex		
7678	0	0	1	ADDR	0E1C	0	Wrt	Add 0x0E1C
7680	0	0	1	DATA	1018	0	Wrt	0x1018 to Add 0x0E1C + 0
7682	0	0	1	DATA	0008	0	Wrt	0x0008 to Add 0x0E1C + 1
7684	0	0	1	DATA	0290	0	Wrt	0x0290 to Add 0x0E1C + 2
7686	0	0	1	DATA	0001	0	Wrt	0x0001 to Add 0x0E1C + 3
7688	0	0	1	DATA	0013	0	Wrt	0x0013 to Add 0x0E1C + 4
7690	0	0	1	DATA	0E28	0	Wrt	0x0E28 to Add 0x0E1C + 5
Write next TD_list at 0x0E28. This time is IN (9) on EP0 at Device Address 02 (Data Stage)								
7692	0	0	1	ADDR	0E28	0	Wrt	Add 0x0E28
7694	0	0	1	DATA	1020	0	Wrt	0x1020 to Add 0x0E28 + 0
7696	0	0	1	DATA	0002	0	Wrt	0x0002 to Add 0x0E28 + 1
7698	0	0	1	DATA	0290	0	Wrt	0x0290 to Add 0x0E28 + 2
7700	0	0	1	DATA	0041	0	Wrt	0x0041 to Add 0x0E28 + 3
7702	0	0	1	DATA	0013	0	Wrt	0x0013 to Add 0x0E28 + 4
7704	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x0E28 + 5
Write the TD_list address (0x0E10) to the HUSB_SIE1_pCurrentTDPtr (0x01B0) to submit (Data Stage)								
7706	0	0	1	ADDR	01B0	0	Wrt	Add 0x01B0
7708	0	0	1	DATA	0E10	0	Wrt	0x0E10 to Add 0x01B0 + 0
7710	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
8972	0	1	0	STAT	0400	1	Rd	HPI Status Reg with 0x0400
8974	0	0	1	ADDR	C090	1	Wrt	Add 0xC090
8976	0	0	1	DATA	0200	1	Wrt	0x0200 to Add 0xC090 + 0
8978	0	1	0	STAT	0000	0	Rd	HPI Status Reg with 0x0000
9706	0	1	0	STAT	0010	1	Rd	HPI Status Reg with 0x0010
Read SIE1msg Register (0x1000 = DONE)								
9708	0	0	1	ADDR	0144	1	Wrt	Add 0x0144
9710	0	1	0	DATA	1000	0	Rd	0x1000 from Add 0x0144 + 0
9712	0	0	1	ADDR	0144	0	Wrt	Add 0x0144
9714	0	0	1	DATA	0000	0	Wrt	0x0000 to Add 0x0144 + 0
Now read back TD_list to check status								
9716	0	0	1	ADDR	0E10	0	Wrt	Add 0x0E10
9718	0	1	0	DATA	1010	0	Rd	0x1010 from Add 0x0E10 + 0
9720	0	1	0	DATA	0008	0	Rd	0x0008 from Add 0x0E10 + 1
9722	0	1	0	DATA	0290	0	Rd	0x0290 from Add 0x0E10 + 2
9724	0	1	0	DATA	0041	0	Rd	0x0041 from Add 0x0E10 + 3
9726	0	1	0	DATA	0003	0	Rd	0x0003 from Add 0x0E10 + 4
9728	0	1	0	DATA	0E1C	0	Rd	0x0E1C from Add 0x0E10 + 5
Read back the Data from Get Device Descriptor								
9730	0	0	1	ADDR	1010	0	Wrt	Add 0x1010
9732	0	1	0	DATA	0112	0	Rd	0x0112 from Add 0x1010 + 0
9734	0	1	0	DATA	0200	0	Rd	0x0200 from Add 0x1010 + 1
9736	0	1	0	DATA	0000	0	Rd	0x0000 from Add 0x1010 + 2

MACHINE 1 - State Listing (continued) (see step 4 in example)								
Label->	**nCS**	**nWR**	**nRD**	**A1/A0**	**D[15:0]**	**Interrupt**		**Comment**
Base->	**Hex**	**Hex**	**Hex**	**Symbol**	**Hex**	**Hex**		
9738	0	1	0	DATA	0800	0	Rd	0x0800 from Add 0x1010 + 3
Now read back TD_list to check status								
9740	0	0	1	ADDR	0E1C	0	Wrt	Add 0x0E1C
9742	0	1	0	DATA	1018	0	Rd	0x1018 from Add 0x0E1C + 0
9744	0	1	0	DATA	0008	0	Rd	0x0008 from Add 0x0E1C + 1
9746	0	1	0	DATA	0290	0	Rd	0x0290 from Add 0x0E1C + 2
9748	0	1	0	DATA	0001	0	Rd	0x0001 from Add 0x0E1C + 3
9750	0	1	0	DATA	0003	0	Rd	0x0003 from Add 0x0E1C + 4
9752	0	1	0	DATA	0E28	0	Rd	0x0E28 from Add 0x0E1C + 5
Read back the Data from Get Device Descriptor (next packet)								
9754	0	0	1	ADDR	1018	0	Wrt	Add 0x1018
9756	0	1	0	DATA	04B4	0	Rd	0x04B4 from Add 0x1018 + 0
9758	0	1	0	DATA	120D	0	Rd	0x120D from Add 0x1018 + 1
9760	0	1	0	DATA	0100	0	Rd	0x0100 from Add 0x1018 + 2
9762	0	1	0	DATA	0201	0	Rd	0x0201 from Add 0x1018 + 3
Now read back TD_list to check status								
9764	0	0	1	ADDR	0E28	0	Wrt	Add 0x0E28
9766	0	1	0	DATA	1020	0	Rd	0x1020 from Add 0x0E28 + 0
9768	0	1	0	DATA	0002	0	Rd	0x0002 from Add 0x0E28 + 1
9770	0	1	0	DATA	0290	0	Rd	0x0290 from Add 0x0E28 + 2
9772	0	1	0	DATA	0041	0	Rd	0x0041 from Add 0x0E28 + 3
9774	0	1	0	DATA	0003	0	Rd	0x0003 from Add 0x0E28 + 4
9776	0	1	0	DATA	0000	0	Rd	0x0000 from Add 0x0E28 + 5
Read back the Data from Get Device Descriptor (next packet)								
9778	0	0	1	ADDR	1020	0	Wrt	Add 0x1020
9780	0	1	0	DATA	0100	0	Rd	0x0100 from Add 0x1020 + 0

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.com