Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.



THIS SPEC IS OBSOLETE

Spec No: 001-67541

Spec Title: AN1149 - IMPLEMENTING THE HARDWARE ASSISTED PARALLEL INTERFACE (HAPI)

Sunset Owner: Srikanth Neerella (SIRK)

Replaced By: NONE

Implementing the Hardware Assisted Parallel Interface (HAPI)

AN1149

Author: Sai Prashanth Chinnapalli
Associated Project: No
Associated Part Family: CY7C66113,CY7C66013,CY7C64113,CY7C64013
Software Version: N/A
Associated Application Notes: None

OBSOLETE

Application Note Abstract

Hardware Assisted Parallel Interface (HAPI) is a parallel interface that simplifies the parallel transfer of data at the back end. AN1149 describes how to use this interface

Introduction

The USB specification offers a highly versatile, inexpensive, and widely accepted standard for interfacing an external peripheral with a USB host (PC, Mac, embedded host, and so on). The communication between the USB host and the USB microcontroller is governed by the rules defined in the USB specification. However, the back-end transfer of data between the USB microcontroller and the peripheral device can be through any serial or parallel link.

HAPI (Hardware Assisted Parallel Interface) is a parallel interface that simplifies the parallel transfer of data at the back end. The HAPI engine interprets and generates the necessary control and handshake signals between a Cypress USB microcontroller and the external interface that sources or sinks data. The advantage of using the HAPI engine is that, because there is less firmware overhead for the USB microcontroller, the transfer throughput is improved.

HAPI functionality is offered in the CY7C64013, CY7C64113, CY7C66013, and CY7C66113 Cypress full-speed USB microcontrollers. These microcontrollers are RISC-based processors using the Cypress optimized CPU core. They are cost-effective solutions for USB applications with medium throughput requirements. The HAPI interface can sustain transfer rates of up to 1 MBps. The actual transfer rate in each implementation depends on what additional functions the Cypress USB microcontroller is to perform, in addition to reading from or writing to the HAPI interface.

It should be noted that if the HAPI interface is to be used at a data rate toward the high end of its specified bandwidth, the throughput bottleneck will be the USB FIFO size and not the HAPI interface. The Cypress USB microcontrollers that support HAPI offer a total aggregate FIFO size of 80 bytes for data endpoints. Therefore, interrupt and isochronous transfers are limited to a maximum of 80 KBps.

For bulk and control transfers, the actual throughput is unpredictable, and higher or lower throughputs may be achievable based on the available bus bandwidth. If a full-speed USB microcontroller with higher throughputs is required, consider using an EZ-USB® (AN21xx family) or an EZ-USB FX (CY7C646*xx*) part.

Table 1 offers a selector guide to help you determine which Cypress USB microcontroller with HAPI engine is most suitable for your application.

Table 1. USB Microcontrollers Supporting HAPI

Hub Requirement	I/O Pin Requirement (including HAPI)	Recommended USB Controller
Hub function required	39 or fewer pins	CY7C66113
	29 or fewer pins	CY7C66013
Hub function not required	36 or fewer pins	CY7C64113
	19 or fewer pins	CY7C64013

HAPI Functional Overview

Figure 1. HAPI/I²C Configuration Register 0x09 (Read/Write)

7	6	5	4	3	2	1	0
R/W		R/W	R/W	R	R	R/W	R/W
I²C Position	Reserved	LEMPTY Polarity	DRDY Polarity	Latch Empty	Data Ready	HAPI Port Width Bit 1	HAPI Port Width Bit 0

The HAPI function is enabled by initializing the least significant two bits of the HAPI/I²C Configuration register (0x09) to a nonzero value. These bits are cleared after a chip reset (a power-on reset or a watchdog reset), which means that the HAPI function is disabled and the I/O ports are not controlled by the HAPI engine. The HAPI engine can transfer one, two, or three bytes of data in a single HAPI Read/Write cycle. Table 2 shows the required settings for the HAPI/I²C Configuration register (0x09) to achieve single-byte or multi-byte data transfers.

Table 2. HAPI Port Configuration

Port Width Bits[1:0]	HAPI Port Width
11	24-bit transfer: P3[7:0], P1[7:0], P0[7:0]
10	16-bit transfer: P1[7:0], P0[7:0]
01	8-bit transfer: P0[7:0]
00	HAPI disabled

The following instructions show an example of how to initialize the HAPI interface for 16-bit transfers:

```
mov a, 2
iowr 09h
```

When a port is initialized to be under HAPI control, the output drivers for that port will be in CMOS output mode, regardless of the output configuration settings of the GPIO Configuration register (0x08).

Bits 2, 3, 4, and 5 of the HAPI/I²C Configuration register are also associated with the HAPI logic. Bits 4 and 5 select the active logic level of the Latch Empty and Data Ready bits. A setting of 0 for the LEMPTY Polarity bit causes the Latch Empty bit (Bit 3) to be low if the data latch is empty. A setting of 1 for the LEMPTY Polarity bit causes the Latch Empty bit to be high if the data latch is empty. As you will see later in this application note, the Latch Empty status bit is used in conjunction with a HAPI write operation.

Similarly, a setting of 0 for the DRDY Polarity bit causes the Data Ready bit to be low if new data is loaded by an external device to the data ports that are under HAPI control. The Data Ready status bit is associated with HAPI Read operations. A setting of 1 for the DRDY Polarity bit causes the Data Ready bit to be active low.

If the HAPI interface is configured to read or write multiple bytes in a single cycle (see Table 2), Port 0 must be the last port that the data is written to or read from by the Cypress microcontroller. A read or write operation on Port 0 causes an update to the Latch Empty bit or the Data Ready bit of the HAPI/I²C Configuration Register.

The major advantage of using the HAPI interface is that the HAPI engine automatically interprets the control signals and generates the handshake signals to the back-end interface. HAPI control and handshake signals are provided on Port 2. Bits [6:2] of Port 2 are used for the HAPI interface. Similar to the HAPI data ports, Port 2 HAPI pins are in CMOS output mode or open drain input mode when HAPI is active. The following sections describe control and handshake pins on Port 2.

P2[2] – Latch Empty Pin

This is an output handshake pin that is used in a HAPI write cycle. When active, this pin indicates to the external interface that the USB controller has processed the last data that was loaded in the HAPI buffers (HAPI port buffers are selected as shown in Table 2), and the HAPI engine is ready to receive more data from the back-end interface. This event tells the external interface to load more data into the HAPI FIFO. The active state of the Latch Empty pin is selected by the LEMPTY Polarity bit of the HAPI/I²C Configuration register. A setting of 0 selects an active-high state for the Latch Empty pin; a setting of 0 selects an active-low state.

Note The Latch Empty pin on Port 2 and the Latch Empty bit of the HAPI/I²C Configuration register have opposite active states.

P2[3] – Data Ready Pin

This is an output handshake pin that is used in a HAPI read cycle. When active, this pin signals to the external interface that the USB microcontroller has just loaded new data into HAPI buffers (HAPI port buffers are selected as shown in Table 2). This event tells the external device to perform a read of the HAPI FIFO. The active state of the Data Ready pin is selected by the DRDY Polarity bit of the HAPI/I²C Configuration register. A setting of 0 selects an active-high state for the Data Ready pin; a setting of 0 selects an active-low state.

Note The Data Ready pin on Port 2 and the Data Ready bit of the HAPI/I²C Configuration register have opposite active states.

P2[4] – STB (Strobe)

This is an active-low input control pin that is used in HAPI write cycles. A high-to-low transition on this pin — provided that chip select (CS) input is active — latches the data present on the HAPI data ports into the HAPI buffer. When

DRAFT

STB returns high, the HAPI engine signals the interrupt controller block and a HAPI interrupt is generated.

P2[5] – OE (Output Enable)

This is an active-low input control pin that is used in HAPI read cycles. A high-to-low transition on this pin — provided that CS input is active —loads the data in the HAPI FIFO out to the port pins. When OE returns high, the HAPI engine signals the interrupt controller block and a HAPI interrupt is generated.

P2[6] – CS (Chip Select)

This is an active-low input that gates both OE and STB inputs.

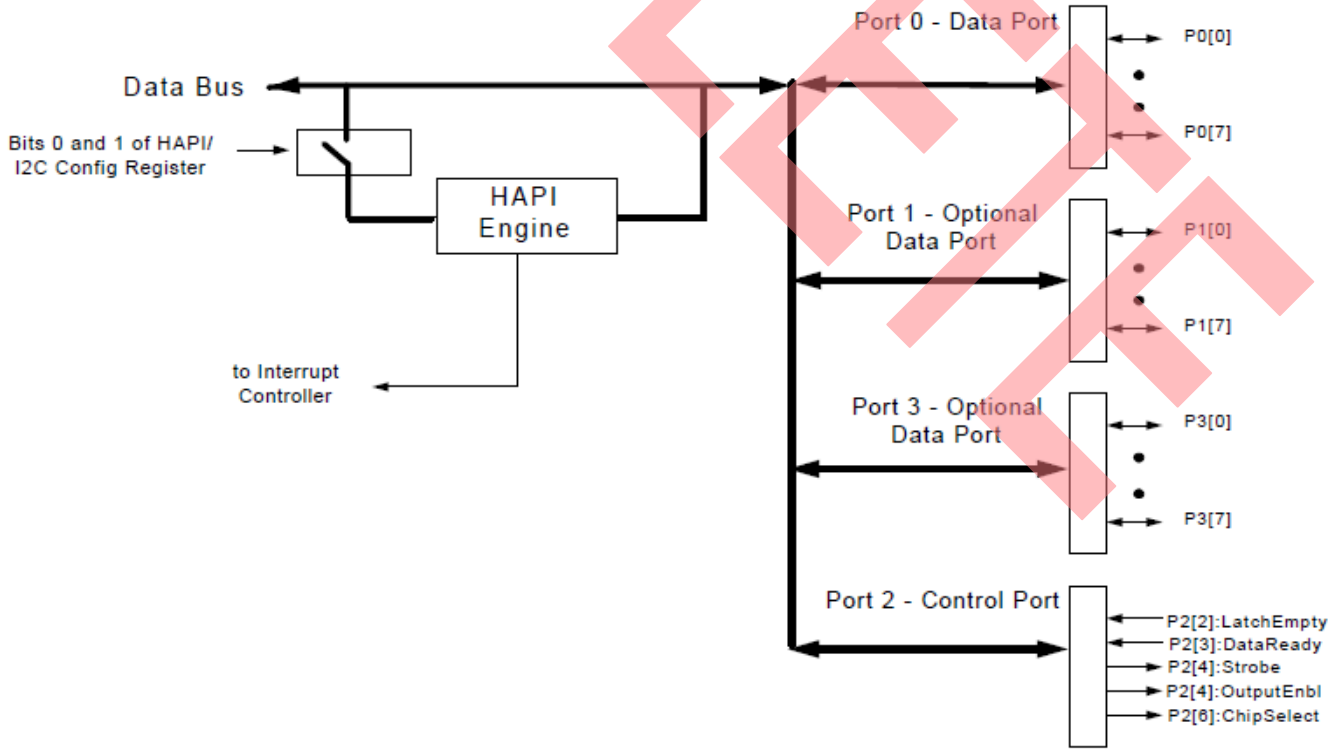
Table 3 provides a summary of HAPI port and bit definitions.

Figure 2 is a hardware block diagram of the HAPI interface. If HAPI is enabled, all of the port bits associated with HAPI operation are now accessed by the HAPI engine. Non-HAPI port bits of Port 2 are unaffected.

Table 3. Port 2 Pin and HAPI Configuration Bit Definitions

Pin	Name	Direction	Description (Port 2 Pin)
P2[2]	LatEmptyPin	Out	Ready for more input data from external interface
P2[3]	DReadyPin	Out	Output data ready for external interface
P2[4]	STB	In	Strobe signal for latching incoming data
P2[5]	OE	In	Output Enable; causes chip to output data
P2[6]	CS	In	Chip Select; gates STB and OE
Bit	**Name**	**R/W**	**Description (HAPI/I²C Configuration Register)**
2	Data Ready	R	Asserted after firmware writes data to Port 0, until OE is driven LOW.
3	Latch Empty	R	Asserted after firmware reads data from Port 0, until STB is driven LOW.
4	DRDY Polarity	R/W	Determines polarity of Data Ready bit and DReadyPin: If 0, Data Ready is active low, DReadyPin is active high. If 1, Data Ready is active high, DReadyPin is active low
5	LEMPTY Polarity	R/W	Determines polarity of Latch Empty bit and LatEmptyPin: If 0, Latch Empty is active low, LatEmptyPin is active high. If 1, Latch Empty is active high, LatEmptyPin is active low.

Figure 2. HAPI Block Diagram



OBSOLETE

HAPI Read by an External Interface

Firmware writes the data to the General Purpose I/O (GPIO) ports. If you need a two- or three-byte transfer, Port 0 must be written last, because a write to Port 0 causes the Data Ready pin to go active, which signals to the external device that data is available. The external device then drives the OE and CS pins active (low), which causes the HAPI data to be output to the port pins.

After the external device reads the HAPI data, it returns the OE pin inactive (high). This generates a HAPI interrupt in the USB microcontroller. If there is more data to be transferred, the firmware should load it into the HAPI ports. The act of writing the data to Port 0 causes the Data Ready pin to go active, and the process repeats.

Note The Data Ready pin (P2[3]) and Data Ready bit (Bit 2 of the HAPI/I^{2}C Configuration register) have opposite active states. Figure 3 shows the timing diagram for a HAPI read cycle.

HAPI Write to an External Interface

The external interface drives STB and CS pins active (low), and drives the data to the HAPI ports. This causes the Latch Empty pin to go inactive because the HAPI latch is now full. The external device should then return the STB pin high. This would generate a HAPI interrupt in the USB microcontroller. At this time the firmware should read the data latched into the HAPI ports and process it based on the application's requirements.

If you need a two- or three-byte transfer, Port 0 must be the last port that is read by the firmware, because a read from Port 0 causes the Latch Empty pin to go active. This would signal to the external device to load the new data into the HAPI latch if there is more data to transfer.

Note The Latch Empty pin (P2[2]) and Latch Empty bit (Bit 3 of the HAPI/I^{2}C Configuration register) have opposite active states. Figure 4 shows the timing diagram for a HAPI write cycle.

Bidirectional HAPI Transfer to and from an External Interface

The sequence of events for a bidirectional HAPI interface is essentially a combination of a HAPI write and a HAPI read transfer sequence. The only additional issue is that when the HAPI interrupt is generated, the firmware must correctly identify which type of transfer caused the interrupt. The Data Ready signal will have returned to its inactive state by the time the HAPI interrupt is generated. The firmware should therefore examine the state of the Latch Empty signal; either from the HAPI/ I^{2}C Configuration register or off Port 2. If the Latch Empty signal is active, a HAPI write by an external device must have generated the interrupt. If the Latch Empty signal is active, a HAPI read by an external device must have generated the interrupt.

GPIO and HAPI Interrupt Sharing

GPIO and HAPI interrupts share the same interrupt and only one can be enabled at a time. Therefore, anytime the HAPI function is enabled, the GPIO interrupts will be disabled.

OBSOLETE

Figure 3. HAPI Read by External Interface from USB Microcontroller

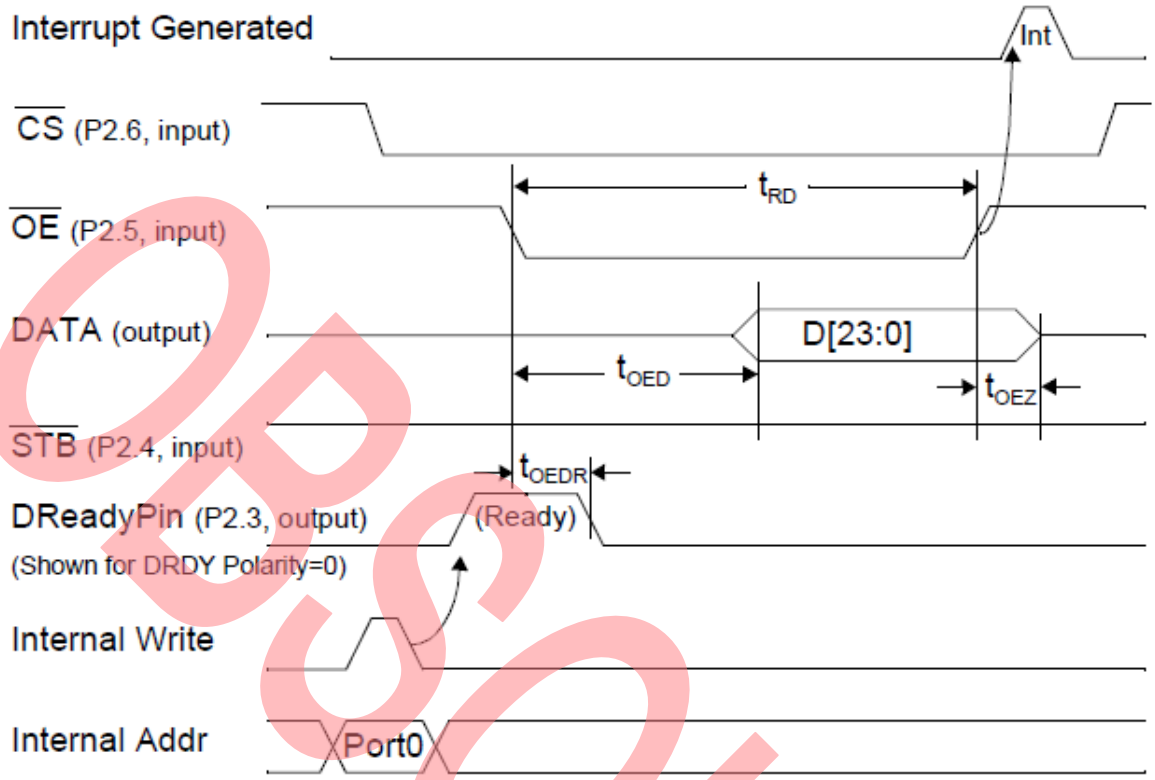
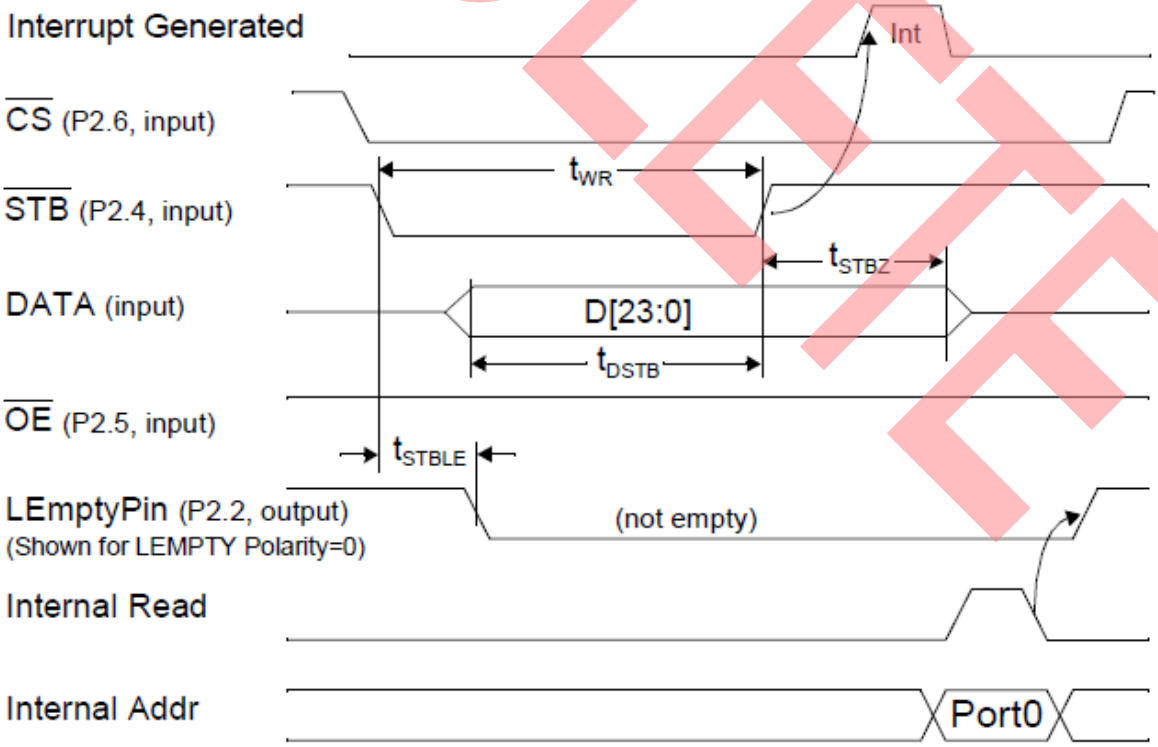


Figure 4. HAPI Write by External Device to USB Microcontroller



OBSOLETE

Sample Code – HAPI Read

Following is a firmware example of how to implement the HAPI interface to transfer data between any two Cypress USB controllers that support HAPI. This sample application contains two modules. The M_read.lst module is the code that should be running on the HAPI master. The HAPI master is the external device that generates the STB, OE, and CS control signals. The S_read.lst module is the code that should be running on the HAPI slave. This sample application uses a one-byte HAPI transfer mode.

```

0000            ;*****************************************************
0000            ;
0000            ; File: m_read.asm
0000            ; Description: HAPI Read code - master CPU
0000            ;
0000            ; Hardware setup:
0000            ;
0000            ;                   Master                   Slave
0000            ;            -------------                   --------------
0000            ;                         |                  |
0000            ;                    P0.7 |<----------------->| P0.7
0000            ;                    P0.6 |<----------------->| P0.6
0000            ;                    P0.5 |<----------------->| P0.5
0000            ;                    P0.4 |<----data bus---->| P0.4
0000            ;                    P0.3 |<----------------->| P0.3
0000            ;                    P0.2 |<----------------->| P0.2
0000            ;                    P0.1 |<----------------->| P0.1
0000            ;                    P0.0 |<----------------->| P0.0
0000            ;                         |                  |
0000            ;                         |                  |
0000            ;                    P3.1 |-/CS------------->| P2.6
0000            ;                    P3.0 |-/OE------------->| P2.5
0000            ;                    P1.1 |<-----latch empty-| P2.2
0000            ;                    P1.0 |<------data ready-| P2.3
0000            ;                    P3.2 |-/STB------------->| P2.4
0000            ;                         |                  |
0000            ;                         |                  |
0000            ;                         |                  |
0000            ;            -------------                   --------------
0000            ;
0000            ; HAPI Read Hints:
0000            ;
0000            ; The slave will write to data ports (e.g., port0 for 1 byte HAPI xfer).
The data will not appear on the common bus to which data ports are attached until both
0000            ; /cs and /oe go active (low). As soon as data is written to port0, DRDY
signal (either port or bit in reg 09h) will go active. When /oe is brought low by
0000            ; the master, DRDY pin goes inactive and data appears on the HAPI data
ports. Note that when HAPI is enabled in a device, the slave cannot write to Port 2
0000            ; bits under HAPI control. Only the master can write to port 2.
Also, if the Monitor program is used to check the state of the registers, the reading of
data ports
0000            ; on the slave is done at some internal stage of the port and NOT at the
external interface. For that reason, writes to HAPI data ports can only be verified
0000            ; by reading the HAPI data regs at the master side and not on the
slave 3652.
0000            ;
0000            ;*****************************************************
0000
0000            ;**************** assembler directives *****************
0000
0000             CPU   66013
0000
0000             XPAGEON
0000
0000            ;constant definitions
```

OBSOLETE

```
50  0000            count:                      equ    6
51  0000            data_ready_pin_ack:         equ    00000001B     ; P1[0]
52  0000            latch_empty_pin_ack:        equ    00000010B     ; P1[1]
53  0000          stb:                          equ    04h           ; P3[2]
54  0000          oe:                           equ    01h           ; P3[0]
55  0000          cs:                           equ    02h           ; P3[1]
56  0000
57  0000          PORT3_RESISTIVE:              equ    11B
58  0000          PORT2_RESISTIVE:              equ    11B
59  0000          PORT1_RESISTIVE:              equ    11B
60  0000          PORT0_RESISTIVE:              equ    11B
61  0000
62  0000          ;***** data memory variables
63  0000          counter:                 equ    23h
64  0000
65  0000          ; 30h to 35h RAM locations are used for the received (read) data
66  0000          hapi_read_buffer_base:  equ    30h
67  0000
68  0000          ;***** port definitions
69  0000          data_port0:                   equ    0
70  0000          data_port1:                   equ    1
71  0000          data_port2:                   equ    2
72  0000          data_port3:                   equ    3
73  0000             hapi_control:              equ    3
74  0000          hapi_status:                  equ    1
75  0000          port0_int:                    equ    4
76  0000          port1_int:                    equ    5
77  0000          port2_int:                    equ    6
78  0000          port3_int:                    equ    7
79  0000          i2c_config:                   equ    9
80  0000          gpioconfig:                   equ    8
81  0000          global_int:                   equ    20h
82  0000          endpoint_int:            equ    21h
83  0000             watchdog:                  equ    26h
84  0000
85  0000          ;*************** interrupt vector table ****************
86  0000
87  0000             ORG    00h
88  0000
89  0000 80 1B [05] jmp    reset                 ; reset vector
90  0002
91  0002 80 1A [05] jmp    error                 ; bus reset interrupt
92  0004
93  0004 80 1A [05] jmp    error                 ; 128us interrupt
94  0006
95  0006 80 1A [05] jmp    error                 ; 1024ms interrupt
96  0008
97  0008 80 1A [05] jmp    error                 ; endpoint 0 interrupt
98  000A
99  000A 80 1A [05] jmp    error                 ; endpoint 1 interrupt
100 000C
101 000C 80 1A [05] jmp    error                 ; endpoint 2 interrupt
102 000E
103 000E 80 1A [05] jmp    error                 ; endpoint 3 interrupt
104 0010
105 0010 80 1A [05] jmp    error                 ; endpoint 4 interrupt
106 0012
107 0012 80 1A [05] jmp    error                 ; hub interrupt vector
108 0014
109 0014 80 1A [05] jmp    error                 ; DAC interrupt vector
110 0016
111 0016 80 1A [05] jmp    error                 ; GPI0 interrupt
112 0018
113 0018 80 1A [05] jmp    error                 ; I2C interrupt vector
114 001A
```

OBSOLETE

```
115 001A            ;************** program listing ************************
116 001A
117 001A            ORG    1Ah
118 001A
119 001A  00   [08] error: halt
120 001B
121 001B            ;*******************************************************
122 001B            ;
123 001B            ;      Interrupt handler: reset
124 001B            ;      Purpose:  The program jumps to this routine when
125 001B            ;                the microcontroller has a power-on reset.
126 001B            ;
127 001B            ;*******************************************************
128 001B
129 001B            reset:
130 001B 19 20 [04]         mov A, 20h
131 001D 30    [05]         swap A, dsp         ; sets data memory stack pointer
132 001E
133 001E 2A 26 [05]         iowr   watchdog
134 0020 19 00 [04]         mov    A, 00h
135 0022 2A 04 [05]         iowr   port0_int
136 0024 2A 05 [05]         iowr   port1_int
137 0026 2A 06 [05]         iowr   port2_int
138 0028 2A 07 [05]         iowr   port3_int
139 002A
140 002A 19 FF [04]         mov    A, ffh
141 002C 2A 00 [05]         iowr   data_port0
142 002E 2A 01 [05]         iowr   data_port1
143 0030 2A 02 [05]         iowr   data_port2
144 0032 2A 03 [05]         iowr   data_port3
145 0034
146 0034 19 00 [04]         mov    a, 0         ; HAPI not enabled on master
147 0036 2A 09 [05]         iowr   i2c_config
148 0038
149 0038 19 03 [04]         mov    A, (PORT3_RESISTIVE | PORT2_RESISTIVE | PORT1_RESISTIVE |
PORT 0_RESISTIVE);
150 003A 2A 08 [05]         iowr   gpioconfig   ; set GPIOs to resistive mode. HAPI bits will
default to CMOS mode
151 003C
152 003C 19 00 [04]         mov    A, 0
153 003E 2A 20 [05]         iowr   global_int   ; no interrupts enabled on master
154 0040 19 00 [04]         mov    A, 00h
155 0042 2A 21 [05]         iowr   endpoint_int
156 0044 70 [08]            di
157 0045
158 0045            ; initialize read buffer
159 0045 19 00 [04]         mov    a, 0         ; clear the read buffer
160 0047 31 30 [05]         mov    [hapi_read_buffer_base], a
161 0049 31 31 [05]         mov    [hapi_read_buffer_base+1], a
162 004B 31 32 [05]         mov    [hapi_read_buffer_base+2], a
163 004D 31 33 [05]         mov    [hapi_read_buffer_base+3], a
164 004F 31 34 [05]         mov    [hapi_read_buffer_base+4], a
165 0051 31 35 [05]         mov    [hapi_read_buffer_base+5], a
166 0053
167 0053            read_loop:
168 0053 1C 30 [04]         mov    x, hapi_read_buffer_base   ; counter and write table
initialization
169 0055 19 06 [04]         mov    a, count
170 0057 31 23 [05]         mov    [counter], a
171 0059
172 0059           wait_for_ack_read:
173 0059           ; The master will wait until the slave drives the Data Ready signal. Once
data is ready to be read, master drives
174 0059           ; /cs and /oe active.
175 0059
```

OBSOLETE

```
176 0059 2A 26 [05]        iowr   watchdog
177 005B 29 01 [05]        iord   hapi_status
178 005D 10 01 [04]        and    a, data_ready_pin_ack
179 005F B0 59 [05]        jnz    wait_for_ack_read          ; DataReady Polarity is set in
slave, hence DRDY pin is active LOW.
180 0061
181 0061h           hapi_read:
182 0061 19 FC [04]        mov    a, ~(oe | cs )
183 0063 2A 03 [05]        iowr   hapi_control               ; drive /cs and /oe active
184 0065
185 0065 29 00 [05]        iord   data_port0
186 0067 32 00 [06]        mov    [x], a
187 0069
188 0069 19 03 [04]        mov    a, cs | oe
189 006B 2A 03 [05]        iowr   hapi_control               ; drive /cs and /oe inactive to
generate a HAPI interrupt at the slave 66013
190 006D
191 006D 22    [04]        inc    x
192 006E 27 23 [07]        dec    [counter]
193 0070 A0 74 [05]        jz     finished                   ; we are done with HAPI Read
194 0072 80 59 [05]        jmp    wait_for_ack_read          ; repeat until the entire buffer
is read back from the slave
195 0074
196 0074            ; we are done
197 0074            finished:
198 0074 00    [08]        halt
199
 1  0000            ;*****************************************************
 2  0000            ;
 3  0000            ;      File: s_read.asm
 4  0000            ;      Description: HAPI Read code - slave CPU.
 5  0000            ;
 6  0000            ;            Master                      Slave
 7  0000            ;       -------------                    --------------
 8  0000            ;                    |                   |
 9  0000            ;              P0.7 |<----------------->| P0.7
10  0000            ;              P0.6 |<----------------->| P0.6
11  0000            ;              P0.5 |<----------------->| P0.5
12  0000            ;              P0.4 |<----data bus----->| P0.4
13  0000            ;              P0.3 |<----------------->| P0.3
14  0000            ;              P0.2 |<----------------->| P0.2
15  0000            ;              P0.1 |<----------------->| P0.1
16  0000            ;              P0.0 |<----------------->| P0.0
17  0000            ;                   |                   |
18  0000            ;                   |                   |
19  0000            ;              P3.1 |-CS--------------->| P2.6
20  0000            ;              P3.0 |-OE--------------->| P2.5
21  0000            ;              P1.1 |<-----latch empty-| P2.2
22  0000            ;              P1.0 |<------data ready-| P2.3
23  0000            ;              P3.2 |-STB-------------->| P2.4
24  0000            ;                   |                   |
25  0000            ;                   |                   |
26  0000            ;                   |                   |
27  0000            ;       -------------                    --------------
28  0000            ;
29  0000            ;*****************************************************
30  0000
31  0000            ;**************** assembler directives *****************
32  0000
33  0000                  CPU    65013
34  0000
35  0000                  XPAGEON
36  0000
37  0000            ;constant definitions
38  0000            hapi_port_width:          equ    1
```

OBSOLETE

```
39  0000            count:                      equ    6
40  0000            drdy_polarity:              equ    00010000B    ; DRDY pin is active LOW
41  0000            lempty_polarity:            equ    00100000B    ; LEMPTY pin is active LOW
42  0000            drdy_lempty_bits:           equ    00010000B
43  0000            data_ready_pin_ack:         equ    00001000B    ; P2[3]
44  0000            latch_empty_pin_ack:        equ    00000100B    ; P2[2]
45  0000
46  0000            PORT3_RESISTIVE:            equ    11B
47  0000            PORT2_RESISTIVE:            equ    11B
48  0000            PORT1_RESISTIVE:            equ    11B
49  0000            PORT0_RESISTIVE:            equ    11B
50  0000            HAPI_GPIO_INT_BIT:          equ    20h
51  0000
52  0000            ;***** data memory variables
53  0000            counter:                    equ    21h
54  0000
55  0000            ; 30h to 35h RAM locations are used for xmit data
56  0000            hapi_read_buffer_base:      equ    30h
57  0000
58  0000            ;***** port bit definitions
59  0000            data_port0:                 equ    0
60  0000            data_port3:                 equ    3
61  0000            hapi_control:               equ    2            ; slave, silicon
62  0000            hapi_status:                equ    2            ; slave, silicon
63  0000            port0_int:                  equ    4
64  0000            port1_int:                  equ    5
65  0000            port2_int:                  equ    6
66  0000            port3_int:                  equ    7
67  0000            i2c_config:                 equ    9
68  0000            gpioconfig:                 equ    8
69  0000            global_int:                 equ    20h
70  0000            endpoint_int:               equ    21h
71  0000            watchdog:                   equ    26h
72  0000
73  0000            ;*************** interrupt vector table ****************
74  0000
75  0000            ORG    00h
76  0000
77  0000 80 1B [05] jmp    reset                                ; reset vector
78  0002
79  0002 80 1A [05] jmp    error                                ; bus reset interrupt
80  0004
81  0004 80 1A [05] jmp    error                                ; 128us interrupt
82  0006
83  0006 80 1A [05] jmp    error                                ; 1024ms interrupt
84  0008
85  0008 80 1A [05] jmp    error                                ; endpoint 0 interrupt
86  000A
87  000A 80 1A [05] jmp    error                                ; endpoint 1 interrupt
88  000C
89  000C 80 1A [05] jmp    error                                ; endpoint 2 interrupt
90  000E
91  000E 80 1A [05] jmp    error                                ; endpoint 3 interrupt
92  0010
93  0010 80 1A [05] jmp    error                                ; endpoint 4 interrupt
94  0012
95  0012 80 1A [05] jmp    error                                ; hub interrupt vector
96  0014
97  0014 80 1A [05] jmp    error                                ; DAC interrupt vector
98  0016
99  0016 80 66 [05] jmp    hapi_isr                             ; HAPI interrupt
100 0018
101 0018 80 1A [05] jmp    error                                ; I2C interrupt vector
102 001A
103 001A            ;************** program listing ************************
```

OBSOLETE

```
104 001A
105 001A            ORG    1Ah
106 001A
107 001A 00    [08] error: halt
108 001B
109 001B            ;*******************************************************
110 001B            ;
111 001B            ;      Interrupt handler: reset
112 001B            ;      Purpose: The program jumps to this routine when
113 001B            ;               the microcontroller has a power-on reset.
114 001B            ;
115 001B            ;*******************************************************
116 001B
117 001B            reset:
118 001B 19 20 [04]         mov A, 20h
119 001D 30 [05]            swap A, dsp                     ; sets data memory stack pointer
120 001E
121 001E 2A 26 [05]         iowr    watchdog
122 0020 19 00 [04]         mov     A, 00h
123 0022 2A 04 [05]         iowr    port0_int
124 0024 2A 05 [05]         iowr    port1_int
125 0026 2A 06 [05]         iowr    port2_int
126 0028 2A 07 [05]         iowr    port3_int
127 002A
128 002A 19 FF [04]         mov     A, FFh
129 002C 2A 00 [05]         iowr    data_port0
130 002E 19 31 [04]         mov     a, drdy_polarity | lempty_polarity | hapi_port_width
131 0030 2A 09 [05]         iowr    i2c_config              ; hapi enabled on slave
132 0032 19 06 [04]         mov     a, count
133 0034 31 21 [05]         mov     [counter], a            ; initialize counter
134 0036
135 0036 19 00 [04]         mov     a, 00h                  ; initialize the read buffer to
0, 1, 2, 3, 4, and 5.
136 0038 31 30 [05]         mov     [hapi_read_buffer_base], a
137 003A 01 01 [04]         add     a, 1h
138 003C 31 31 [05]         mov     [hapi_read_buffer_base+1], a
139 003E 01 01 [04]         add     a, 1h
140 0040 31 32 [05]         mov     [hapi_read_buffer_base+2], a
141 0042 01 01 [04]         add     a, 1h
142 0044 31 33 [05]         mov     [hapi_read_buffer_base+3], a
143 0046 01 01 [04]         add     a, 1h
144 0048 31 34 [05]         mov     [hapi_read_buffer_base+4], a
145 004A 01 01 [04]         add     a, 1h
146 004C 31 35 [05]         mov     [hapi_read_buffer_base+5], a
147 004E
148 004E 1C 30 [04]         mov     x, hapi_read_buffer_base
149 0050 19 03 [04]         mov     A, (PORT3_RESISTIVE | PORT2_RESISTIVE | PORT1_RESISTIVE |
PORT0_RESISTIVE);
150 0052 2A 08 [05]         iowr    gpioconfig              ; set GPIOs to resistive mode.
HAPI bits will default to CMOS mode.
151 0054
152 0054 19 20 [04]         mov     A, HAPI_GPIO_INT_BIT
153 0056 2A 20 [05]         iowr    global_int              ; Only enable the HAPI interrupt
154 0058 19 00 [04]         mov     A, 00h
155 005A 2A 21 [05]         iowr    endpoint_int
156 005C 72    [08]         ei
157 005D
158 005D            ; initiate HAPI Read. In this sample app, the slave initiates the
transfer. This drives the Data Ready pin active and the master begins the HAPI Read cycle.
159 005D            ; In a more realistic situation, the slave would look for a flag from the
master before initiating the HAPI Read sequence.
160 005D
161 005D 1A 30 [06]         mov     a, [hapi_read_buffer_base]
162 005F 2A 00 [05]         iowr    data_port0
163 0061 22 [04]            inc     x
```

OBSOLETE

```
0062
0062            main:
0062 2A 26 [05]       iowr    watchdog
0064 80 62 [05]       jmp     main                      ; Main loop just waits for the
HAPI interrupt
0066
0066            hapi_isr:
0066 29 02 [05]       iord    hapi_control
0068 10 08 [04]       and     a, data_ready_pin_ack     ; because polarity bit is '1',
the DRDY pin must be LOW before the next slave write to Port 0 buffer.
006A A0 71 [05]       jz      end_hapi_isr
006C
006C 1B 00 [07]       mov     a, [x+0]
006E 2A 00 [05]       iowr    data_port0                ; load data to Port 0
0070 22    [04]       inc     x                         ; increment the table pointer
0071
0071            end_hapi_isr:
0071 73    [08]       reti
```

Sample Code - HAPI Write

Following is a firmware example of how to implement the HAPI interface to transfer data between any two Cypress USB controllers that support HAPI. This sample application contains two modules. The M_write.lst module is the code that should be running on the HAPI master. The HAPI master is the external device that generates the STB, OE, and CS control signals. The S_write.lst module is the code that should be running on the HAPI slave.

```

0000            ;*****************************************************
0000            ;
0000            ; File: m_write.asm
0000            ; Description: HAPI Write code - master CPU.
0000            ;
0000            ; Hardware setup:
0000            ;
0000            ;                   Master                    Slave
0000            ;            -------------                    --------------
0000            ;                          |                  |
0000            ;                    P0.7 |<----------------->| P0.7
0000            ;                    P0.6 |<----------------->| P0.6
0000            ;                    P0.5 |<----------------->| P0.5
0000            ;                    P0.4 |<----data bus---->| P0.4
0000            ;                    P0.3 |<----------------->| P0.3
0000            ;                    P0.2 |<----------------->| P0.2
0000            ;                    P0.1 |<----------------->| P0.1
0000            ;                    P0.0 |<----------------->| P0.0
0000            ;                          |                  |
0000            ;                          |                  |
0000            ;                    P3.1 |-/CS------------->| P2.6
0000            ;                    P3.0 |-/OE------------->| P2.5
0000            ;                    P1.1 |<-----latch empty-| P2.2
0000            ;                    P1.0 |<------data ready-| P2.3
0000            ;                    P3.2 |-/STB------------>| P2.4
0000            ;                          |                  |
0000            ;                          |                  |
0000            ;                          |                  |
0000            ;            -------------                    --------------
0000            ;
0000            ;*****************************************************
0000
0000            ;**************** assembler directives *****************
0000
0000                  CPU    66013
0000
```

OBSOLETE

```
38  0000                    XPAGEON
39  0000
40  0000              ;constant definitions
41  0000              count:                      equ    6
42  0000              hapi_port_width:            equ    01
43  0000              data_ready_pin_ack:         equ    00000001B     ; P1[0]
44  0000              latch_empty_pin_ack:        equ    00000010B     ; P1[1]
45  0000              stb:                        equ    04h           ; P3[2]
46  0000              oe:                         equ    01h           ; P3[0]
47  0000              cs:                         equ    02h           ; P3[1]
48  0000
49  0000              PORT3_RESISTIVE:            equ    11B
50  0000              PORT2_RESISTIVE:            equ    11B
51  0000              PORT1_RESISTIVE:            equ    11B
52  0000              PORT0_RESISTIVE:            equ    11B
53  0000
54  0000              ;***** data memory variables
55  0000              counter:                    equ    23h
56  0000
57  0000              ; 30h to 35h RAM locations are used for xmit data
58  0000              hapi_write_buffer_base:     equ    30h
59  0000
60  0000              ;***** port bit definitions
61  0000              data_port0:                 equ    0
62  0000              data_port1:                 equ    1
63  0000              data_port2:                 equ    2
64  0000              data_port3:                 equ    3
65  0000              hapi_control:               equ    3
66  0000              hapi_status:                equ    1
67  0000              port0_int:                  equ    4
68  0000              port1_int:                  equ    5
69  0000              port2_int:                  equ    6
70  0000              port3_int:                  equ    7
71  0000              i2c_config:                 equ    9
72  0000              gpioconfig:                 equ    8
73  0000              global_int:                 equ    20h
74  0000              endpoint_int:               equ    21h
75  0000              watchdog:                   equ    26h
76  0000
77  0000              ;*************** interrupt vector table ***************
78  0000
79  0000              ORG    00h
80  0000
81  0000 80 1B [05]   jmp    reset                                  ; reset vector
82  0002
83  0002 80 1A [05]   jmp    error                                  ; bus reset interrupt
84  0004
85  0004 80 1A [05]   jmp    error                                  ; 128us interrupt
86  0006
87  0006 80 1A [05]   jmp    error                                  ; 1024ms interrupt
88  0008
89  0008 80 1A [05]   jmp    error                                  ; endpoint 0 interrupt
90  000A
91  000A 80 1A [05]   jmp    error                                  ; endpoint 1 interrupt
92  000C
93  000C 80 1A [05]   jmp    error                                  ; endpoint 2 interrupt
94  000E
95  000E 80 1A [05]   jmp    error                                  ; endpoint 3 interrupt
96  0010
97  0010 80 1A [05]   jmp    error                                  ; endpoint 4 interrupt
98  0012
99  0012 80 1A [05]   jmp    error                                  ; hub interrupt vector
100 0014
101 0014 80 1A [05]   jmp    error                                  ; DAC interrupt vector
102 0016
```

OBSOLETE

```
103 0016 80 1A [05] jmp    error                                    ; GPI0 interrupt
104 0018
105 0018 80 1A [05] jmp    error                                    ; I2C interrupt vector
106 001A
107 001A            ;************** program listing ************************
108 001A
109 001A            ORG  1Ah
110 001A
111 001A 00    [08] error: halt
112 001B
113 001B            ;*****************************************************
114 001B            ;
115 001B            ;     Interrupt handler: reset
116 001B            ;     Purpose: The program jumps to this routine when
117 001B            ;              the microcontroller has a power-on reset.
118 001B            ;
119 001B            ;*****************************************************
120 001B
121 001B            reset:
122 001B 19 20 [04]       mov A, 20h
123 001D 30    [05]       swap A, dsp                    ; sets data memory stack pointer
124 001E
125 001E 2A 26 [05]       iowr   watchdog
126 0020 19 00 [04]       mov    A, 00h
127 0022 2A 04 [05]       iowr   port0_int
128 0024 2A 05 [05]       iowr   port1_int
129 0026 2A 06 [05]       iowr   port2_int
130 0028 2A 07 [05]       iowr   port3_int
131 002A
132 002A 19 00 [04]       mov    a, 0                    ; hapi not enabled on master
133 002C 2A 09 [05]       iowr   i2c_config
134 002E
135 002E 19 03 [04]       mov    A, (PORT3_RESISTIVE | PORT2_RESISTIVE | PORT1_RESISTIVE |
PORT0_RESISTIVE);
136 0030 2A 08 [05]       iowr   gpioconfig              ; initialize GPIO config
register. The HAPI data and control pins will be in CMOS OUT mode
137 0032
138 0032 19 00 [04]       mov    A, 0                    ; disable all interrupts on
master
139 0034 2A 20 [05]       iowr   global_int
140 0036 19 00 [04]       mov    A, 00h
141 0038 2A 21 [05]       iowr   endpoint_int
142 003A 70    [08]       di
143 003B
144 003B            ; initialize write buffer with 0, 1, 2, 3, 4, and 5.
145 003B 19 00 [04]       mov    a, 00h
146 003D 31 30 [05]       mov    [hapi_write_buffer_base], a
147 003F 01 01 [04]       add    a, 1h
148 0041 31 31 [05]       mov    [hapi_write_buffer_base+1], a
149 0043 01 01 [04]       add    a, 1h
150 0045 31 32 [05]       mov    [hapi_write_buffer_base+2], a
151 0047 01 01 [04]       add    a, 1h
152 0049 31 33 [05]       mov    [hapi_write_buffer_base+3], a
153 004B 01 01 [04]       add    a, 1h
154 004D 31 34 [05]       mov    [hapi_write_buffer_base+4], a
155 004F 01 01 [04]       add    a, 1h
156 0051 31 35 [05]       mov    [hapi_write_buffer_base+5], a
157 0053
158 0053            write_loop:
159 0053 1C 30 [04]       mov x  , hapi_write_buffer_base   ; counter and table pointer
initialization
160 0055 19 06 [04]       mov    a, count
161 0057 31 23 [05]       mov    [counter], a
162 0059
163 0059                  wait_for_ack_write:
```

OBSOLETE

```
0059 2A 26 [05]        iowr   watchdog
005B 29 01 [05]        iord   hapi_status
005D 10 02 [04]        and    a, latch_empty_pin_ack     ; on the slave side (swrite.asm)
lempty_polarity bit is set. Therefore Latch Empty pin is active LOW.
005F B0 59 [05]        jnz    wait_for_ack_write         ; wait for latch to be empty
before a write
0061
0061             hapi_write:
0061 2A 26 [05]        iowr   watchdog
0063 1B 00 [07]        mov a, [x+0]
0065 2A 00 [05]        iowr   data_port0                 ; write data to port 0
0067 22    [04]        inc    x                          ; increment table index
0068 27 23 [07]        dec [counter]                     ; decrement counter
006A
006A             prepare_control_bits_to_write:
006A
006A 29 03 [05]        iord   hapi_control
006C 10 F9 [04]        and    a, ~(cs | stb)
006E 2A 03 [05]        iowr   hapi_control               ; drive /cs and /stb active
0070 29 03 [05]        iord   hapi_control
0072 0D 06 [04]        or     a, cs | stb
0074 2A 03 [05]        iowr   hapi_control               ; now drive /cs and /stb inactive
to generate an interrupt at the slave 66013
0076
0076 1A 23 [06]        mov    a, [counter]               ; check to see if transfer is
done
0078 16 00 [04]        cmp    a, 0
007A A0 7E [05]        jz     finished                   ; if done, exit
007C 80 59 [05]        jmp    wait_for_ack_write         ; repeat until the entire buffer
is xmitted
007E
007E             ; we are done
007E             finished:
007E 00    [08]        halt
007F


0000             ;*****************************************************
0000             ;
0000             ; File: s_write.asm
0000             ; Description: HAPI Write code - slave CPU.
0000             ;
0000             ; Hardware setup:
0000             ;
0000             ;            Master                    Slave
0000             ;      -------------                   --------------
0000             ;                   |                  |
0000             ;              P0.7 |<---------------->| P0.7
0000             ;              P0.6 |<---------------->| P0.6
0000             ;              P0.5 |<---------------->| P0.5
0000             ;              P0.4 |<----data bus---->| P0.4
0000             ;              P0.3 |<---------------->| P0.3
0000             ;              P0.2 |<---------------->| P0.2
0000             ;              P0.1 |<---------------->| P0.1
0000             ;              P0.0 |<---------------->| P0.0
0000             ;                   |                  |
0000             ;                   |                  |
0000             ;              P3.1 |-/CS------------->| P2.6
0000             ;              P3.0 |-/OE------------->| P2.5
0000             ;              P1.1 |<-----latch empty-| P2.2
0000             ;              P1.0 |<------data ready-| P2.3
0000             ;              P3.2 |-/STB------------>| P2.4
0000             ;                   |                  |
0000             ;                   |                  |
0000             ;                   |                  |
```

OBSOLETE

```
29  0000            ; ------------ --------------
30  0000            ;
31  0000            ;*******************************************************
32  0000
33  0000            ;**************** assembler directives *****************
34  0000
35  0000                  CPU    66013
36  0000
37  0000                  XPAGEON
38  0000
39  0000            ;constant definitions
40  0000            count:                    equ    6
41  0000            hapi_port_width:          equ    01
42  0000            drdy_polarity:            equ    00010000B
43  0000            lempty_polarity:          equ    00100000B
44  0000            drdy_lempty_bits:         equ    00000000B
45  0000            data_ready_pin_ack:       equ    00001000B          ; P2[3]
46  0000            latch_empty_pin_ack:      equ    00000100B          ; P2[2]
47  0000            latch_empty:              equ    04h
48  0000            data_ready:               equ    08h
49  0000
50  0000            read:                     equ    01h
51  0000            write:                    equ    02h
52  0000            ack:                      equ    04h
53  0000            HAPI_GPIO_INT_BIT:        equ    20h
54  0000            PORT3_RESISTIVE:          equ    11B
55  0000            PORT2_RESISTIVE:          equ    11B
56  0000            PORT1_RESISTIVE:          equ    11B
57  0000            PORT0_RESISTIVE:          equ    11B
58  0000
59  0000            ;***** data memory variables
60  0000            counter:                  equ    23h
61  0000
62  0000            ; 30h to 3Fh RAM locations are used for xmit data
63  0000            hapi_buffer_base:         equ    30h
64  0000
65  0000            ;***** port bit definitions
66  0000            data_port0:               equ    0
67  0000            data_port1:               equ    1
68  0000            data_port2:               equ    2
69  0000            data_port3:               equ    3
70  0000            hapi_control:             equ    2
71  0000            hapi_status:              equ    2
72  0000            port0_int:                equ    4
73  0000            port1_int:                equ    5
74  0000            port2_int:                equ    6
75  0000            port3_int:                equ    7
76  0000            i2c_config:               equ    9
77  0000            gpioconfig:               equ    8
78  0000            global_int:               equ    20h
79  0000            endpoint_int:             equ    21h
80  0000            watchdog:                 equ    26h
81  0000
82  0000            ;*************** interrupt vector table ***************
83  0000
84  0000            ORG 00h
85  0000
86  0000 80 1B [05] jmp    reset                          ; reset vector
87  0002
88  0002 80 1A [05] jmp    error                          ; bus reset interrupt
89  0004
90  0004 80 1A [05] jmp    error                          ; 128us interrupt
91  0006
92  0006 80 1A [05] jmp    error                          ; 1024ms interrupt
93  0008
```

OBSOLETE

```
94  0008 80 1A [05] jmp    error                           ; endpoint 0 interrupt
95  000A
96  000A 80 1A [05] jmp    error                           ; endpoint 1 interrupt
97  000C
98  000C 80 1A [05] jmp    error                           ; endpoint 2 interrupt
99  000E
100 000E 80 1A [05] jmp    error                           ; endpoint 3 interrupt
101 0010
102 0010 80 1A [05] jmp    error                           ; endpoint 4 interrupt
103 0012
104 0012 80 1A [05] jmp    error                           ; hub interrupt vector
105 0014
106 0014 80 1A [05] jmp    error                           ; DAC interrupt vector
107 0016
108 0016 80 41 [05] jmp    hapi_isr                        ; HAPI interrupt
109 0018
110 0018 80 1A [05] jmp    error                           ; I2C interrupt vector
111 001A
112 001A            ;************** program listing ************************
113 001A
114 001A            ORG  1Ah
115 001A
116 001A 00    [08] error: halt
117 001B
118 001B            ;*****************************************************
119 001B            ;
120 001B            ;     Interrupt handler: reset
121 001B            ;     Purpose: The program jumps to this routine when
122 001B            ;              the microcontroller has a power on reset.
123 001B            ;
124 001B            ;*****************************************************
125 001B
126 001B            reset:
127 001B 19 20 [04]       mov A, 20h
128 001D 30 [05]          swap A, dsp                     ; sets data memory stack pointer
129 001E
130 001E 2A 26 [05]       iowr   watchdog
131 0020 19 FF [04]       mov    A, FFh
132 0022 2A 04 [05]       iowr   port0_int
133 0024 2A 05 [05]       iowr   port1_int
134 0026 2A 06 [05]       iowr   port2_int
135 0028 2A 07 [05]       iowr   port3_int
136 002A
137 002A 19 31 [04]       mov    a,drdy_polarity | lempty_polarity | hapi_port_width
138 002C 2A 09 [05]       iowr   i2c_config              ; hapi enabled on slave
139 002E
140 002E 19 03 [04]       mov    A, (PORT3_RESISTIVE | PORT2_RESISTIVE | PORT1_RESISTIVE |
PORT0_RESISTIVE);
141 0030 2A 08 [05]       iowr   gpioconfig
142 0032
143 0032 1C 30 [04]       mov    x, hapi_buffer_base
144 0034 19 20 [04]       mov    A, HAPI_GPIO_INT_BIT
145 0036 2A 20 [05]       iowr   global_int              ; only enable the HAPI interrupt
146 0038 19 00 [04]       mov    A, 00h
147 003A 2A 21 [05]       iowr   endpoint_int
148 003C 72    [08]       ei
149 003D
150 003D            main:
151 003D 2A 26 [05]       iowr   watchdog                ; main loop does nothing except
wait for a HAPI interrupt
152 003F 80 3D [05]       jmp    main
153 0041
154 0041            hapi_isr:
155 0041
156 0041 29 02 [05]       iord   hapi_control
```

OBSOLETE

```
157 0043 10 04 [04]        and     a, latch_empty_pin_ack    ; Latch Polarity bit is set. A
HIGH indicates latch is not empty. That means master must have done a write.
158 0045 A0 4C [05]        jz      end_hapi_isr
159 0047
160 0047 29 00 [05]        iord    data_port0
161 0049 32 00 [06]        mov     [x+0], a
162 004B 22    [04]        inc     x
163 004C
164 004C              end_hapi_isr:
165 004C 73    [08]        reti
166 004D
167
168
```

OBSOLETE

Document History

Document Title: AN1149 - Implementing the Hardware Assisted Parallel Interface (HAPI)

Document Number: 001-67541

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3179839	CSAI	2/24/2011	OLD APP. NOTE: Obtained spec. # for note to be added to spec. system. Updated copyright. Added source disclaimer, revision disclaimer. Applied new template. Copy edit. **This note had no technical updates.**
*A	3198844	DSG	3/17/2011	Minor Change: Fixed footer
*B	4184440	RSKV	11/06/2013	Obsolete document.

OBSOLETE

In March of 2007, Cypress recatalogued all of its Application Notes using a new documentation number and revision code. This new documentation number and revision code (001-xxxxx, beginning with rev. **), located in the footer of the document, will be used in all subsequent revisions.

EZ-USB is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

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

© Cypress Semiconductor Corporation, 2000–2013. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.