Application Report
SPRAA13A − July 2005

# TMS320C6000™ EMIF to USB Interfacing Using Cypress EZ-USB SX2

Prateek Bansal, Todd Hiers
C6000 Applications

## ABSTRACT

This application report describes a glueless interface between a Texas Instruments TMS320C6416 digital signal processor (DSP) reference board and a Cypress CY7C68001 (EZ-USB SX2™) USB device. The two devices are interfaced through the external memory interface (EMIF) of the C6416 DSP.

The document provides a pin connection diagram demonstrating the interface between the two devices. In addition, it provides tables and timing diagrams indicating that timing requirements for such an interface were satisfied.

The application report describes interface with the TMS320C6416 DSP but the details in the application report will be applicable to other TMS320C6000™ devices (C671x, DM64x, C64x) as well.

This application report contains project code that can be downloaded from http://www.ti.com/lit/zip/SPRAA13.

## Contents

## List of Figures

## List of Tables

## 1 Introduction

The Cypress CY7C68001 (EZ-USB SX2™) device enables USB 2.0 support for an external master device, such as the TMS320C6000™ DSP. The TMS320C6416™ DSP reference design implementation using the SX2™ offers the following:

- An industry standard serial interface between a digital signal processor (TMS320C6416) and peripherals such as mass storage and imaging devices.
- Programmability to operate at high (480 Mbps) or full (12 Mbps) speed**.**
- 4K bytes First In First Out (FIFO) for maximum flexibility and throughput.
- 8- or 16-bit bidirectional data bus for command and data input/output.

The SX2 device is interfaced to the 16-bit external memory interface B (EMIF B) of TMS320C6416 device. The EMIF B is a secondary interface of the C6416 while the EMIF A is the primary EMIF.

## 2 Interface

The SX2 communicates with the EMIF B asynchronously. Figure 1 shows the pin connection between the two devices and Table 1 describes pin functionality of the two devices. The schematics for this reference design can be downloaded from this link:

http://www−s.ti.com/sc/psheets/sprc137/sprc137.zip

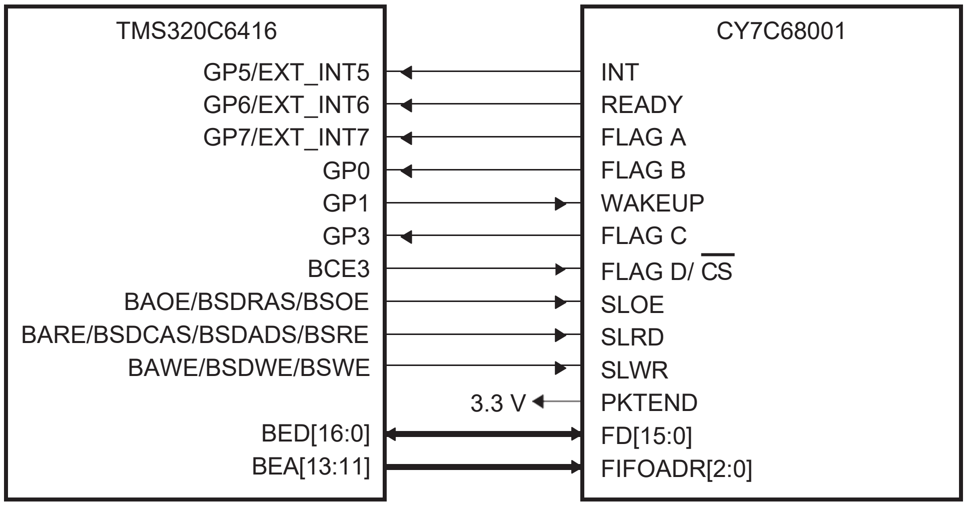

**Figure 1. TMS320C6416 to SX2 Interface Block Diagram**

**Table 1. SX2 toTMS320C6416 Pin Connections**

| SX2 Pin | TMS320C6416 Pin | Function |
|---|---|---|
| INT# | GP5/EXT_INT5 | INT# signal indicates that the SX2 has data to be read, or that an interrupt event has occurred. |
| READY | GP6/EXT_INT6 | READY gates DSP command reads and writes |
| FLAG A | GP7/EXT_INT7 | FLAGA pins report the status of the FIFO selected by the FIFOADR[2:0] pins. At reset, these pin report the status of the programmable flag. |
| FLAG B | GP0 | FLAGB pins report the status of the FIFO selected by the FIFOADR[2:0] pins. At reset, these pin report the status of the full flag. |
| WAKEUP | GP1 | This pin awakens the SX2 from a low-power mode |
| FLAG C | GP3 | FLAGC pins report the status of the FIFO selected by the FIFOADR[2:0] pins. At reset, these pin report the status of the empty flag. |
| FLAG D/CS# | BCE3 | FLAGD is connected to the chip select function. |
| SLOE | BAOE/BSDRAS/BSOE | When asserted, the data bus is driven by SX2 |
| SLRD | BARE/BSDCAS/BSDADS/BSRE | When asserted, the FIFO pointer is incremented on each rising edge of IFCLK. |
| SLWR | BAWE/BSDWE/BSWE | When asserted, data on the FD bus is written to the FIFO and the FIFO pointer is incremented on each rising edge of the IFCLK |
| PKTEND | | Always high. Commits the current buffer to USB. |
| FD[15:0] | BED[15:0] | Data Bus |
| FIFOADR[2:0] | BEA[13:11] | Address Bus Select for FIFO |

The SX2 presents two interfaces to the DSP:

1. FIFO Interface: This has eight 512 byte blocks (4K bytes total) in the endpoint RAM that directly serve as FIFO memories. There are four configurable endpoints that share this 4 KB memory space named EP2, EP4, EP6, and EP8
2. Command Interface: This is used to setup the SX2, read status, load descriptors, and access Endpoint 0.

The selection of individual FIFOs or the command interface is done through FIFOADR[2:0] pins. These FIFOs/command interfaces are addressed by the following memory range in CE3 space of the EMIF B. The endpoints and their corresponding memory address range on the DSP are tabulated in Table 2. The address range is calculated knowing that the EMIF B uses the BEA [20:1] pins for addressing.

**Table 2. Endpoints and the Corresponding Memory Address Range**

| Endpoints | Start Address | End Address |
|---|---|---|
| FIFO2 | 0x6C00 0000 | 0x6C00 03FF |
| FIFO4 | 0x6C00 0800 | 0x6C00 0BFF |
| FIFO6 | 0x6C00 1000 | 0x6C00 13FF |
| FIFO8 | 0x6C00 1800 | 0x6C00 1BFF |
| Endpoint 0 (Command Interface) | 0x6C00 2000 | 0x6C00 203F |

## 2.1 Interrupts

The SX2 provides an output signal to the DSP for every interrupt condition. The SX2 has six interrupt sources, each of which can be enabled or disabled by setting or clearing the corresponding bit in the INTENABLE register. When an interrupt occurs, the INT# pin is asserted and the bit corresponding to the interrupt source is set in the INTENABLE register. On INT# assertion, the Interrupt Status byte (INTENABLE register) presents itself on the lower portion of the data bus (BED[7:0]). In order to read a SX2 register, the DSP is required to send a register read request. However, on interrupt occurrence, the DSP will not require a register read request for reading Interrupt Status byte. It can directly read the Interrupt status byte present on the data bus after an interrupt occurs. On reading the interrupt status byte, the DSP can identify the interrupt source and service it accordingly.

## 2.2 Flags

The FIFO Flag pins (FLAG A-FLAG C) report the status of the FIFO as selected by the FIFOADR[2:0] pin configurations. The FIFO flag pins can be independently configured, using the FLAGSAB/FLAGSCD registers to correspond to a programmable, full or empty flag from any of the four endpoint FIFOs.

On the reference board, FLAG D functions as a chip select pin.

As seen in Figure 1, the FLAG A output pin can be used either as a general purpose input or as an external interrupt to the DSP depending on the DSP's pin configuration. This gives an option of using software polling or hardware control via interrupt on the FLAG A status.

Figure 1 also indicates that the FLAG B and the FLAG C pins can only be used as general purpose inputs. This indicates that software polling is required to service the FLAG B and the FLAG C pins.

Some example FIFO status pin configurations are illustrated as follows.

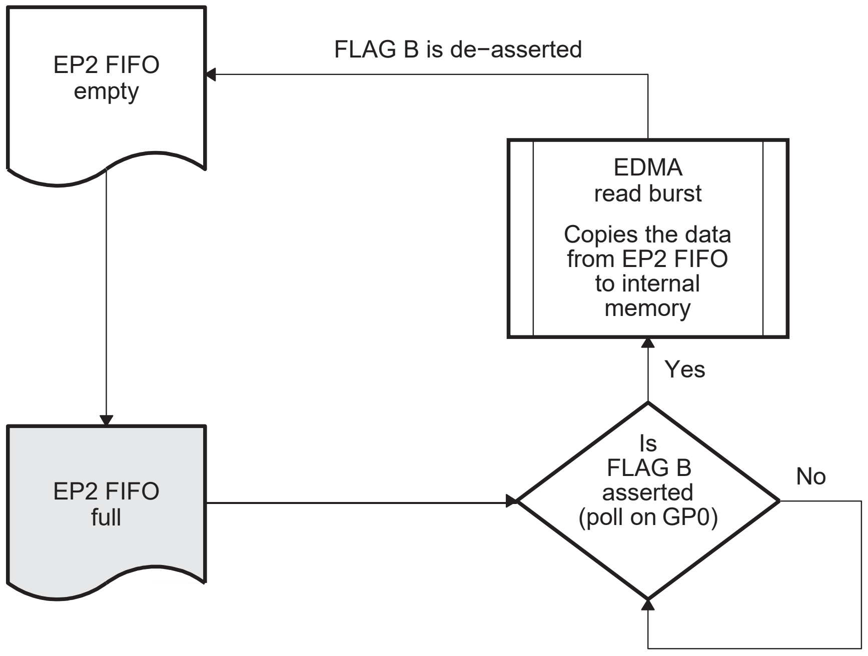
EP2 FIFO empty
FLAG B is de−asserted
EDMA read burst
Copies the data from EP2 FIFO to internal memory
Yes
EP2 FIFO full
Is FLAG B asserted (poll on GP0)
No

**Figure 2. Example Flag Configuration − Polling**

Consider the example shown in Figure 2. The FIFOADR[2:0] pins are configured to select the Endpoint 2 (EP2) FIFO. Assuming that the FLAG B pin is in its default state, this indicates a full flag status of the selected FIFO endpoint on FIFOADR pins i.e., EP2. The full flag status is monitored on the GP0 pin of the interface. This implies that software polling is required to service the GP0 or the FLAG B pin status. As a result, Figure 2 highlights an example application performing an EDMA read burst equal to the depth of the FIFO where the FLAG B pin reports that the EP2 FIFO is full.

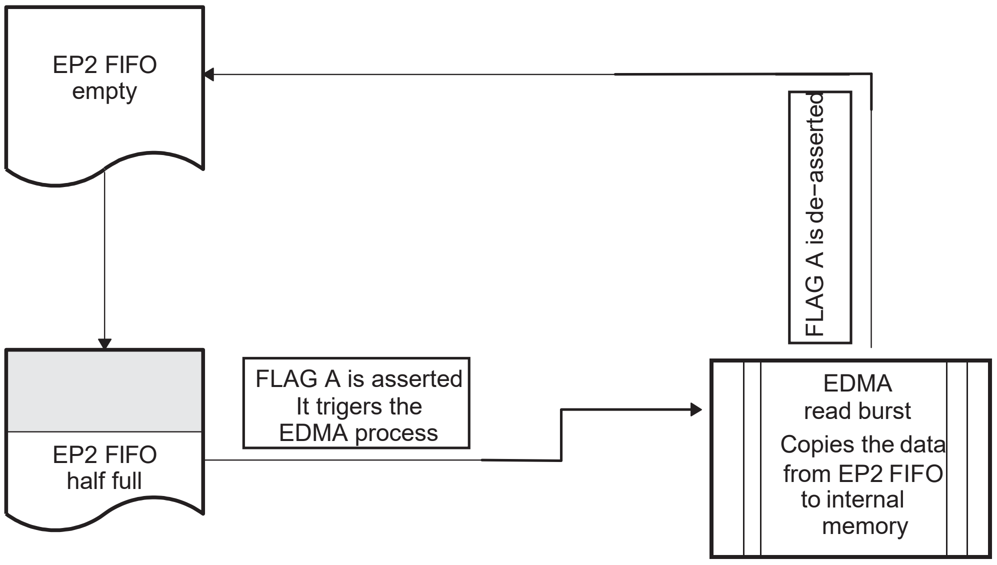

**Figure 3. Example Flag Configuration − Interrupt**

As shown in Figure 3, another application might require triggering an EDMA read burst event automatically when the FIFO is half full. An external interrupt will be needed for triggering. This requires the use of FLAG A pin as it can only be configured as an external interrupt. The FLAG A pin in its default state shows the programmable flag(PF) status of the selected FIFO endpoint on FIFOADR[2:0]. Assume that the EP2 FIFO is selected. The programmable flag can be setup, using the EPxPFH/KL registers, to indicate when the EP2 FIFO fills up to half of its capacity. Any interrupt occurrence on the FLAG A pin with this configuration triggers the EDMA read burst automatically.

There is an alternate way of checking the status of the endpoint flags. This requires reading the EPxxFLAGS register. The details of reading an SX2 register is described in Section 4.2

## 3 Configuration

### 3.1 C6416 DSP Configuration

The C6416 core can be programmed to run up to 720 MHz. This reference design is configured as follows:

- DSP boots in little-endian, PCI mode.
- DSP EMIF B runs at 120 MHz (CPU/6).
- DSP EMIF A runs at 150 MHz from an external clock.
- SX2 communicates with the EMIF B over the asynchronous memory interface.

Section 5 illustrates the timing requirements for the interface. The setup, strobe, and hold width were found to be 2, 6, and 3 clock cycles for reads and 2, 6, and 9 clock cycles for writes respectively with the clock synchronized to BECLKOUT1 (120 MHz). Since the CE3 space control register supports the maximum write hold width of 7 clock cycles, the CE3 space clock was synchronized to BECLKOUT2 running at 60 MHz (EMIF B input clock divided by 2 = 120/2 = 60 MHz). This is required to set the SNCCLK bit in CE Space Secondary Control Register (CESEC3) to 1 and the setup, strobe, and hold width in CE3 Space Control Register (CECTL3) to 1, 3, and 2 clock cycles for reads and 1, 3, and 5 clock cycles for writes respectively.

The following are the registers required to be configured on the DSP:

- Global Purpose Enable Register (GPEN).

  The GPEN register bits need to be set to indicate if the flag output pins of the SX2 are used as an external interrupt or a general purpose input (GPIO).
- Global Purpose Direction Register (GPDIR)

  It configures the GPIO pins to be inputs or outputs.
- CE3 Space Control Register (CECTL3)

  It specifies the setup, strobe and hold fields for both read and write cycles on the interface.
- CE3 Space Secondary Control Register (CESEC3)

  The SNCCLK bit in the register is set to 1 to synchronize the CE3 signals to BECLKOUT2.

## 3.2 SX2 Configuration

There are a number of SX2 registers which need to be initialized to define the communication interface. The details of the SX2 register set can be found in Reference [2]. Some of the main registers are highlighted below:

- Endpoint Packet Length Registers (EPxPKTLENH)

  The WORDWIDE bit in the register needs to be set appropriately to indicate an 8 or 16 bit interface.
- Endpoint Configuration Registers (EPxCFG)

  The endpoints EP 2, 4, 6 and 8 share eight 512 byte buffers supporting bulk, interrupt and isochronous transfers. EP2 and EP6 can be double-, triple- or quad buffered. EP4 and EP8 can only be double-buffered. The endpoints can be configured by setting the EPxCFG registers with endpoint buffer size specified in EPxPKTLENH/L registers. The default endpoint memory configuration is:

  EP2: Bulk OUT, 512 bytes/packet, double buffered
  EP4: Bulk OUT, 512 bytes/packet, double buffered
  EP6: Bulk IN, 512 bytes/packet, double buffered
  EP8: Bulk IN, 512 bytes/packet, double buffered
- FIFO Flag x Assignments Registers (FLAGSAB/FLAGSCD)

  These registers need to be programmed to configure the various FIFO flags to reflect appropriate status of the endpoint buffers.
- EP × Programmable Flag Registers (EPxPFH/L)

  If a FIFO flag is selected to be a programmable flag, this register is configured to define the threshold condition for the flag assertion.

# 4 Communication

## 4.1 Writing to SX2

For data writes, the FIFOADR pins should be selected to write to appropriate IN endpoint FIFO buffer.

There are two types of command write sequences to SX2:

- The first is for downloading custom descriptor information into the SX2 which it uses for subsequent enumerations. There are two types of descriptor downloads.
  a. Downloading only Vendor ID (VID), Product ID (PID) and Device ID (DID) information. In this case, the SX2 enumerates with its default interface and endpoint configurations.
  b. Downloading complete descriptor information including interface and endpoint configuration information.
- The second command write sequence is for programming the SX2 control registers. This requires writing a command byte followed by two data bytes. Prior to writing the register, two conditions must be met:
  c. BED[13:11] pins must be [1 0 0].
  d. Ready line must be HIGH. The DSP should not initiate a command if the READY line or GP6 pin is not in a high state.

The command byte format is shown in Table 3.

**Table 3. Command Address Write Byte**

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| Address/Data = 1 | Read/Write = 0 | A5 | A4 | A3 | A2 | A1 | A0 |

- Each byte written to SX2 is either an address or data byte as determined by Bit7. Bit7 = 1 specifies an address byte and Bit7 = 0 indicates a data byte.
- For an address byte, Bit6 determines whether it is a read or write request. Bit6 = 0 implies it is a write request and Bit6 = 1 implies it is a read request.
- Bits[5:0] hold the register address of the request.
- For a data byte, the Bits[3:0] contain the data and Bits[6:5] are do not care.

To write a byte into SX2, the command address byte is followed by two data bytes. The first command data byte contains the upper nibble of data (Table 4) and the second data byte has the lower nibble of data (Table 5).

**Table 4. Command Data Write Byte One**

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| 0 | X | X | X | D7 | D6 | D5 | D4 |

**Table 5. Command Data Write Byte Two**

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| 0 | X | X | X | D3 | D2 | D1 | D0 |

For example:

To write the byte <00110110> into the IFCONFIG register (Address = 0x01), the sequence of commands is as follows:

- Write the command address byte 1000 0001
- Once the byte has been received by the SX2, the READY pin of SX2 (GP6 pin of C6416) goes low to inform that no more data should be sent. Once it is ready to receive data, the READY pin goes high. Hence GP6 pin (interfaced to READY pin) should be polled for high state before writing data bytes to SX2
- Write the first command data byte 0000 0011
- Write the second command data byte 0000 0110

## 4.2 Reading From SX2

For data reads, the FIFOADR pins should be selected to read from the appropriate OUT endpoint FIFO buffer directly. Register reads require first setting the register address that is to be read.

The INT# pin signals a USB event occurrence when in non-register read mode. When in a register read mode as explained in Section 2 [Interrupts], the $\overline{\text{INT}}$ pin signals the DSP that the requested register status byte is available for reading from the lower portion of data bus. Remember in order to read the control register content, the Ready signal should be asserted.

**Table 6. Command Address Read Byte**

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| Address/Data = 1 | Read/Write = 1 | A5 | A4 | A3 | A2 | A1 | A0 |

For example:

To read the contents of IFCONFIG register (0x01), the sequence of commands is as follows:

- Poll GP6 pin for high state to check if SX2 is ready to receive the information
- Send command address byte 1100 0001
- Poll GP5 pin (INT# pin of SX2) for low state. If in asserted state, read the data from the data bus

## 5 Timing Requirements

**Table 7. TMS320C6416 Timing Requirements**

| | | Min | Max |
|---|---|---|---|
| $t_{su(EDV-AREH)}$ | Setup time, EDx valid before ARE* high | 6.2 | |
| $t_{h(AREH-EDV)}$ | Hold time, EDx valid after ARE* high | 1 | |
| $t_{osu(SELV-AREL)}$ | Output setup time, select signals valid to ARE* low | RS*E − 1.6 | |
| $t_{oh(AREH-SELIV)}$ | Output hold time, ARE* high to select signals invalid | RH*E − 1.7 | |
| $t_{d(EK01H–AREV)}$ | Delay time, ECLKOUT high to ARE* valid | 0.8 | 6.6 |
| $t_{osu(SELV-AWEL)}$ | Output setup time, select signals valid to AWE* low | WS*E – 1.9 | |
| $t_{oh(AWEH-SELIV)}$ | Output hold time, AWE* high to select signals valid | WH*E – 1.7 | |
| $t_{d(EK01H-AWEV)}$ | Delay time, ECLKOUT high to AWE* valid | 0.9 | 6.7 |
| $t_{w(GPIH)}$ | Pulse duration for GPIO Inputs high | 16.66 | |
| $t_{w(GPIL)}$ | Pulse duration for GPIO Inputs low | 16.66 | |

NOTE: RS = Read Setup, RST = Read Strobe, RH = Read Hold, WS = Write Setup, WST = Write Strobe, WH = Write Hold, E = ECLKOUT Time Period

**Table 8. SX2Timing Requirements**

| | | Min | Max |
|---|---|---|---|
| $t_{RDpwl}$ | Pulse width low, SLRD | 50 | |
| $t_{RDpwh}$ | Pulse width high, SLRD | 50 | |
| $t_{IRD}$ | Interrupt to SLRD | 0 | |
| $t_{XINT}$ | SLRD to interrupt/FLAGS output prop delay | | 70 |
| $t_{XFD}$ | SLRD to command/FIFO data output propagation delay | | 15 |
| $t_{OEon}$ | SLOE turn on to FIFO data valid | | 10.5 |
| $t_{OEoff}$ | SLOE turn-off to FIFO data hold | | 10.5 |
| $t_{WR(pwl)}$ | Pulse low, SLWR | 50 | |
| $t_{WR(pwh)}$ | Pulse high, SLWR | 70 | |
| $t_{SFD}$ | Setup time, SLWR to command/FIFO DATA | 10 | |
| $t_{FDH}$ | Hold time, Command/FIFO DATA to SLWR | 10 | |
| $t_{RDYWR}$ | READY to SLWR time | 0 | |
| $t_{RDY}$ | SLWR to READY/FLAGS output propagation delay | | 70 |
| $t_{SFA}$ | Setup time, FIFOADR[2:0] to RD/WR/PKTEND | 10 | |
| $t_{FAH\_RD}$ | Hold time, SLRD/PKTEND to FIFOADR[2:0] | 20 | |
| $t_{FAH\_WR}$ | SLWR to FIFOADR[2:0] hold time [TBD] | 70 | |

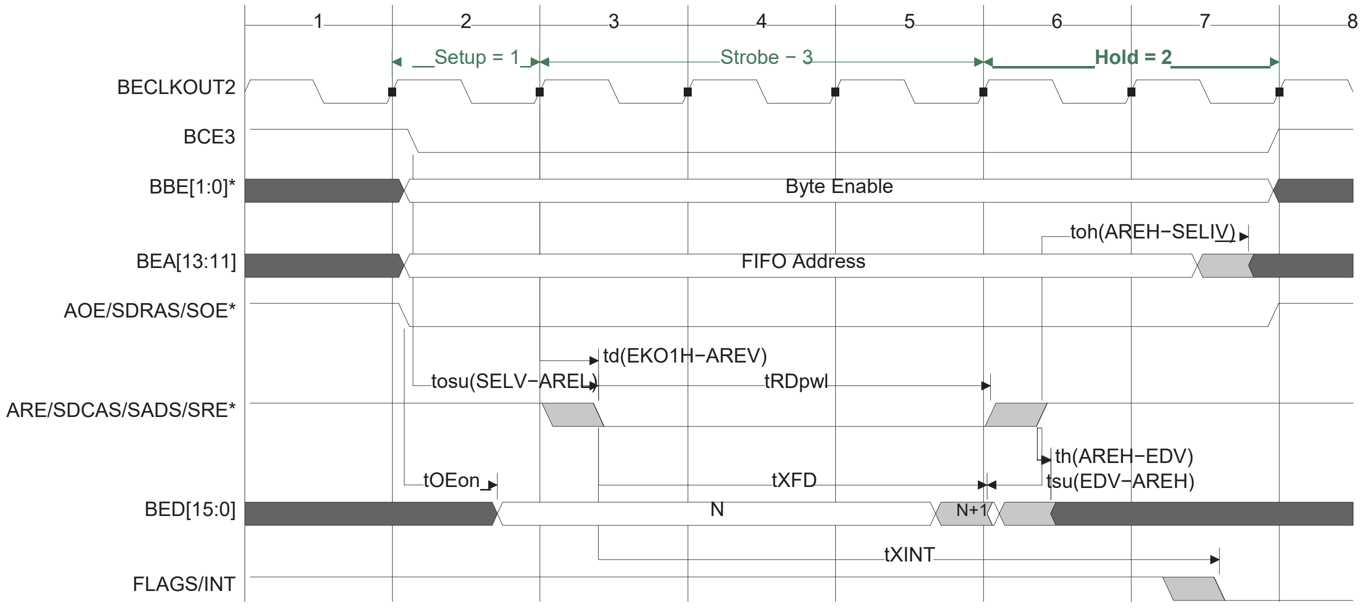

**Figure 4. Timing Diagram for Asynchronous Read**

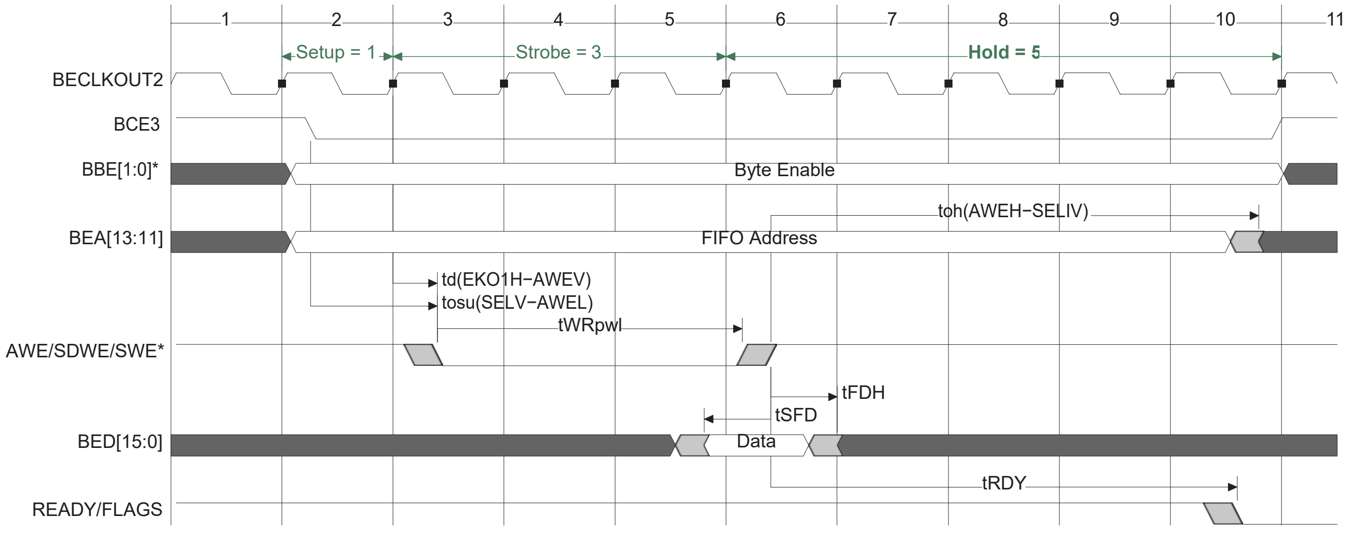

**Figure 5. Timing Diagram for Asynchronous Write**

## 6 Firmware

The firmware driver code shows low−level communication between the SX2 device and the TMS320C6416 DSP. The code implements the initialization and data handling that is required for a typical USB application. The code can be used as a starting framework for any customized application. It is by no means intended to be an extensive implementation of the USB protocol.

Figure 6 shows the block diagram of a typical USB application. The USB host is typically a PC and the USB device is the TMS320C6416 DSP interfaced to the SX2 device. The PC application used for host−side communication is the Cypress EZ−USB Control Panel [Reference 7]. The PC application (Cypress EZ−USB Control Panel) sends out the USB requests to perform a certain operation (specified in Chapter 9 of USB 2.0 Specification) and the SX2 device responds to these requests. If these requests can not be handled by the SX2 device (a slave device) alone, it interrupts the external master (TMS320C6416 DSP) to service those requests.

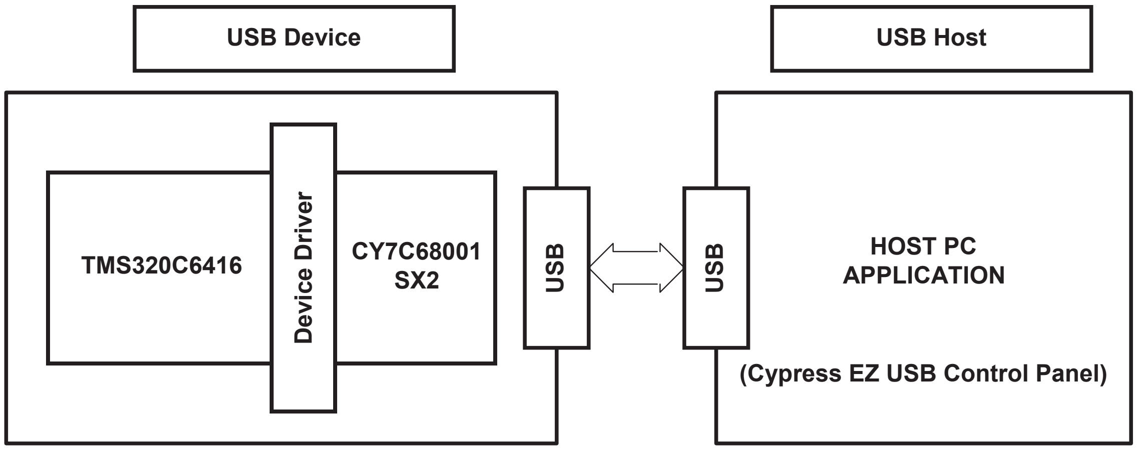

**Figure 6. System Setup**

The firmware code is written in the C language and structured for reuse in any application. Table 9 describes the function of each files in the firmware:

| File Name | Function |
|---|---|
| Vectors.asm | Defines the interrupt vector table |
| C6416_sx2_descriptors.c | Defines the default descriptor that is loaded to the SX2 for enumeration |
| C6416_sx2_low_level_io.c | Implements low level I/O functions for communication between the TMS320C6416 DSP and the SX2 device |
| C6416_sx2_regs_init.c | Defines initialization values for SX2 registers |
| C6416_sx2_setup.c | Implements the data handling for endpoint 0 |
| C6416_sx2_process.c | Implements data loopback between the USB device and PC. |
| C6416_sx2_main.c | Implements the main function that enumerates SX2 and calls data handling functions. |

Figure 7 shows the firmware code flow:

The different components in the flowchart (Figure 7) are explained in further detail:

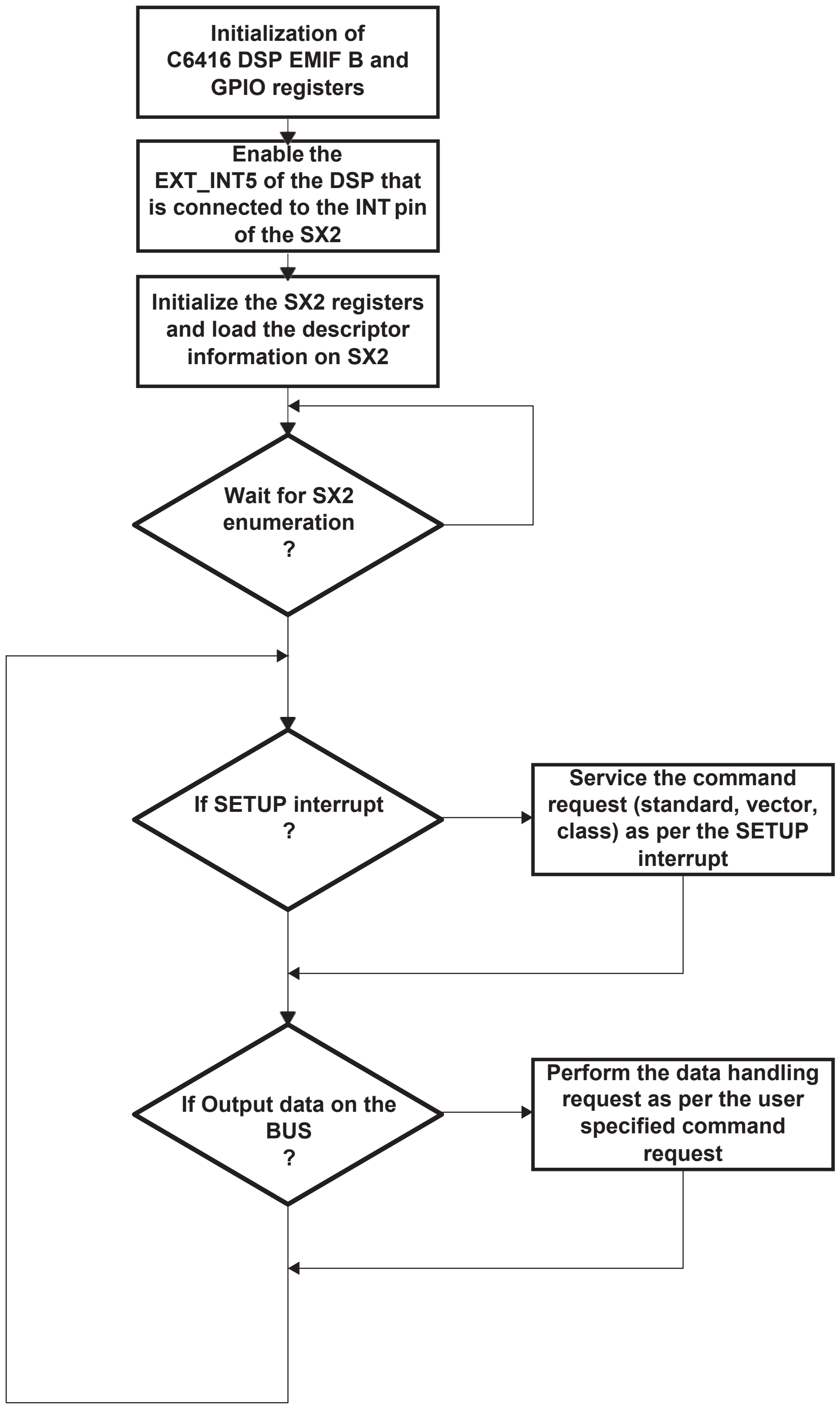
Initialization of C6416 DSP EMIF B and GPIO registers
Enable the EXT_INT5 of the DSP that is connected to the INT pin of the SX2
Initialize the SX2 registers and load the descriptor information on SX2
Wait for SX2 enumeration ?
If SETUP interrupt ?
Service the command request (standard, vector, class) as per the SETUP interrupt
If Output data on the BUS ?
Perform the data handling request as per the user specified command request

**Figure 7. Firmware Code Flowchart**

## 6.1 C6416 Initalization

The EMIF B and GPIO registers are initialized on the TMS320C6416 DSP as per Section 3.1.

## 6.2 Interrupt

The interrupt EXT_INT5 of the TMS320C6416 DSP (connected to the INT pin of SX2 device) is enabled to identify the SX2 interrupt requests. The interrupt service routine int_isr() services the SX2 interrupts.

The Interrupt Service Routine distinguishes between the two types of interrupts (requested data ready or asynchronous interrupt) by examining the read_interrupt flag. If the ISR finds read_interrupt to be true, it clears the interrupt and does no further processing. This indicates to the c6416_sx2_read_reg () function that the data is available to read. If read_interrupt is FALSE, then this interrupt is an asynchronous interrupt that needs to be parsed. Depending on the source of the interrupt, some flags are set or toggled to tell the other code sections what is happening. Every time the interrupt occurs, data is ready to be read, and the SX2 calls the c6416_sx2_low_level_read() function. This is true whether it is requested data or an interrupt source.

## 6.3 Enumeration

The SX2 registers are initialized and the descriptor information is loaded to the SX2 for enumeration. The descriptor used for this application is defined in the SX2 datasheet. The user should change the descriptor information as per their application. The SX2 asserts the ENUMOK interrupt to signal the completion of the enumeration process.

## 6.4 Low Level I/O Functions

The low−level I/O functions like reading/writing from SX2 are defined in the file c6416_sx2_usb_io.c. They are implemented as per Section 4.

## 6.5 Endpoint 0 Transfer

The SX2 automatically responds to all USB standard requests covered in chapter 9 of the USB 2.0 specification except the Set/Clear Feature Endpoint requests. Some of the standard requests that are handled automatically by the SX2 device are GET_CONFIGURATION, GET_DESCRIPTOR, GET_INTERFACE, SET_CONFIGURATION, SET_DESCRIPTOR and SET_INTERFACE.

When the host issues a Set Feature or a Clear feature request, the SX2 will trigger a SETUP interrupt to the external master. The USB spec requires that the device respond to the Set/Clear endpoint feature request by setting/clearing the STALL condition on that endpoint.

The c6416_sx2_setup.c file implements the above functionality. In addition, it also demonstrates how to handle the Endpoint 0 USB data transfer for standard, class or vendor specific requests.

This example acknowledges the vendor request 0xAA and 0xAB. This vendor request could be used by the host application to indicate application specific status. The example also uses vendor requests 0xB6 and 0xB8 to signify a short packet.

All other requests are currently stalled. To stall a request, the external master initiates a write request for the SETUP register, 0x32, and writes any non−zero value to the register.

To complete endpoint zero data transfers, the ep0buf_ready flag is used. If the SX2 receives a setup request with a non−zero length, it asserts the EP0BUF interrupt. For an IN request, this interrupt indicates that the EP0 buffer is available to be written to. For an OUT request, this interrupt indicates that a packet was transferred from the host to the SX2.

## 6.6 Data Loopback

The file c6416_sx2_process.c performs bulk data transfers between different endpoints. If there is bus activity, then the DSP checks to see if the SX2 asserted the FLAGS interrupt. This indicates that the host sent data to one of the OUT endpoints, EP2 or EP4. If there is OUT data available, the DSP reads the EP24FLAGS register. This register checks the empty status of EP2 and EP4. If one of the endpoints contains data, the DSP reads data out of the endpoint, one byte at a time, and subsequently writes the bytes into one of the endpoints used for USB IN data. EP2 data is looped into EP6, and EP4 data is looped into EP8. The DSP continues to read and write data until the endpoint becomes empty.

The example can also be used to send an IN data packet of size less than the configured IN packet size. To do this, use the vendor request 0xB6 or 0xB8 which will write to the INPKTEND register, thereby committing any data already in EP6 or EP8 to the USB, regardless of the configured IN packet size.

## 7 Conclusion

The application note described the interface between the TMS320C6416 DSP and the Cypress CY7C68001. It provided details about the device configuration and communication protocol for this interface. The timing requirements were also provided for this interface. Finally, it delved into the firmware code associated with this interface.

## 8 References

1. *TMS320C6414,TMS320C6415,TMS320C6416 Fixed-Point Digital Signal Processors Data Sheet* (literature number SPRS146H)
2. CY7C68001 EZ-USB SX2™ High-Speed USB Interface Design, Cypress Semiconductor Corporation http://www.cypress.com/products/datasheet.cfm?partnum=SX2-56PVC
3. *TMS320C6000 EMIF to External FIFO Interface Application Report* (literature number SPRA543)
4. Bulk Transfers with the Cypress EZ-USB SX2™ Connected to a Hitachi SH3™ DMA Interface, Cypress Semiconductor Corporation http://www.cypress.com/products/datasheet.cfm?partnum=SX2-56PVC
5. *TMS320C6000 EMIF to External Asynchronous SRAM Interface Application Report* (literature number SPRA542A)