USB Media

Philip Li (pcl2007)
Pamela Lee (pyl2001)
Yanjie Ma (ym2009)
Neeraj Bahati (nrb2001)
Alex Kuo (aek2001)
Michael Chuen (mpc2002)

Table of Contents

1. Introduction…….... 3
2. Project Description…….... 3
3. Responsibilities…….. 3
4. Project Details…….... 4
5. Problems Encountered……11
6. Lessons Learned…… 12
7. Advice……13
8. Appendix……13
9. Files……18

Introduction

Universal Serial Port (USB) is the new standard for connecting peripherals to computers. Its performance outshines that of the traditional serial and parallel ports on a computer. Serial and parallel ports are limited to certain devices configured a unique way. A common issue with parallel ports are its non-standards while serial ports have limited bandwidth and ports. However, the USB allows a means of standard connection between peripherals and computers while providing a fast connection. The basis for the project was the configuration of the USB so that it is recognized as a device that can transmit information to the host computer particularly as a remote control. Programs such as Winamp and Windows Media Player are a big force in the entertainment industry. Many people have replaced the traditional radio and CD player with a more advanced means of listening to music and have created a new market of products that will enhance a listeners experience, thus is the basis of our idea for the project.

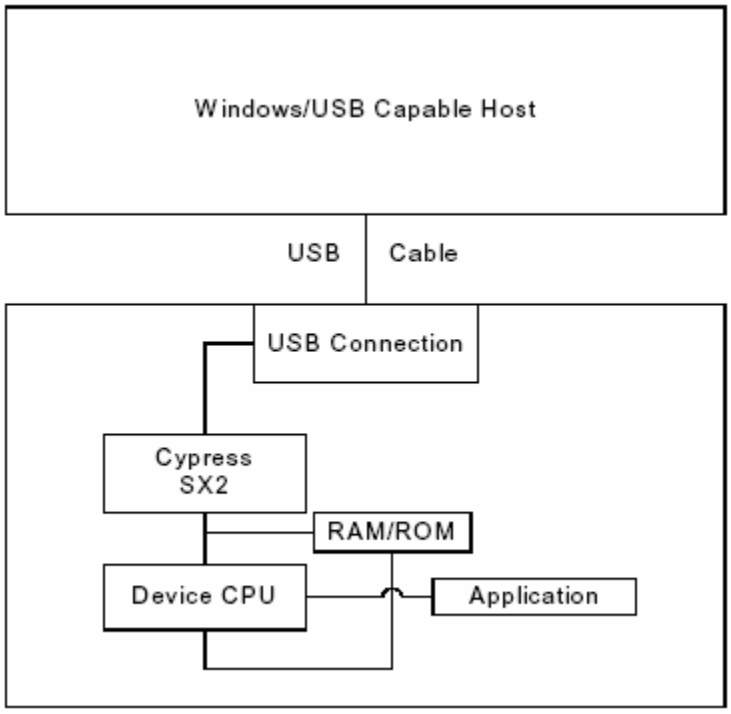
Project Description

The FPGA is used to receive information from a user to control several functions of a mp3 player: play, stop and skip song. When a button is pushed, a signal is sent to the OPB. The microblaze continually receives information from the USB indicating any command changes from the push buttons and sends the information to the USB using the Cypress CY&C68001 EZ-USB SX2 USB interface located on the board. Once data is transferred over the USB, the operating system on the host computer automatically installs drivers specific to the device specifications which will allow the computer to retrieve information from the USB. The information received is handled using a C program that will interface with a command line program that tells the music player which functions to perform.

Responsibilities

1. Reading Signal from Push Button –Alex, Mike, Neeraj
2. Sending information to OPB-Pam, Phil, Yanjie
3. Enumeration of USB device – Pam, Phil, Yanjie
4. Sending Information over USB – Phil, Yanjie
5. Reading Information from USB port – Alex, Neeraj
6. Command Interface with Winamp – Alex, Mike, Pam

Project Details

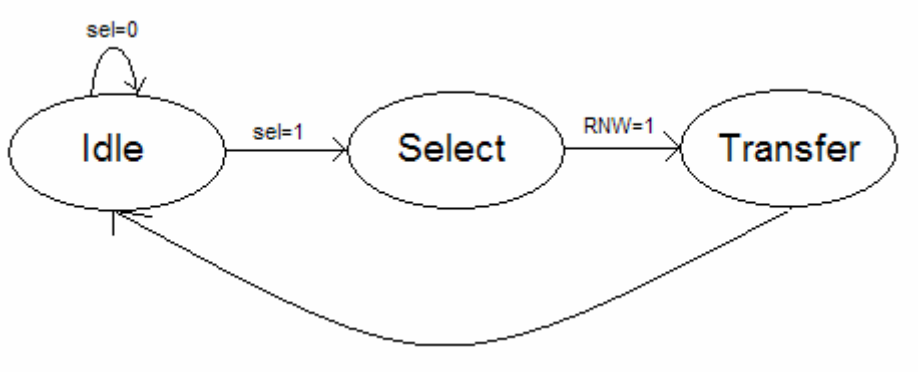


1. Push Buttons

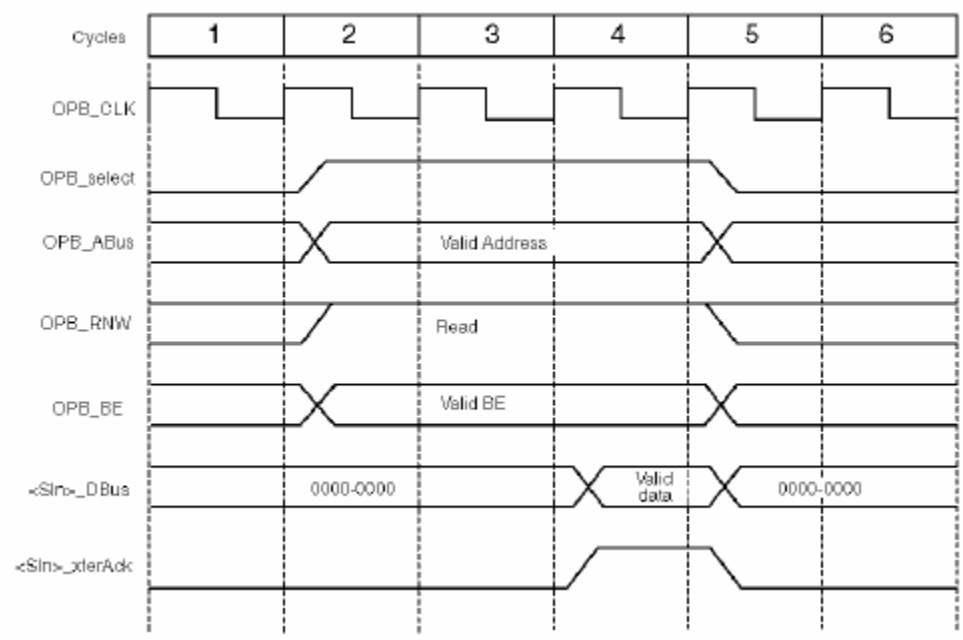
Peripheral Bus	Switch	FPGA Pin	Function
PB-A8	S1	100	Pushbutton 1
PB-A9	S2	101	Pushbutton 2
PB-A10	S3		Pushbutton 3
PB-A11	S4	109	Pushbutton 4
PB-A12	S5-1	110	DIP switch 1
PB-A13	S5-2	111	DIP switch 2
PB-A14	S5-3	112	DIP switch 3
PB-A15	S5-4	113	DIP switch 4
PB-A16	S5-5	114	DIP switch 5
PB-A17	S5-6	115	DIP switch 6
PB-A18	S5-7	121	DIP switch 7
PB-A19	S5-8	122	DIP switch 8

The push buttons assign a media player functionality to three of the four pushbuttons on the XESS XSB-300E. As you can see from the pushbutton assignments listed above, there is no FPGA pin for pushbutton three. We're unsure as to why the pin was either left unconnected or why they neglected to list the pin number. Since we could only work

with three functionalities, we decided to assign them as play, stop and next track. In essence, these buttons would be the same as pushing the play, stop and next track buttons within the player itself.



The actual implementation of the push buttons builds on the SRAM device that was created in Lab 6. To fully understand the individual signals and clock cycles, we examine the OPB clock diagram for a read operation. We can choose to ignore the write operation here because the transfer from the push buttons is in onlyone direction (from the pin to the OPB device). From the timing diagram, we choose to use a FSM in order to represent each stage of the transfer progress. The above diagram is a general diagram of the states that we implement in our FSM. When matched with the timing diagram, cycle three represents the first transition into the SELECT state. (So cycle two is IDLE and four is TRANSFER).



Clock Cycle of OPB During a Read Transaction

Because of the fact that the transfers from the push buttons are in a single direction, many of the additional signals and booleans that were needed in the BRAM/SRAM

implementation are no longer required. The general idea of mapping the ports and using an OBUF and address mapping still take place in our implementation.

2. USB Transmission:

Once information is received by the push buttons, the information is then passed into a C program that will then be passed into the USB. These signals in turn will then be used in the interrupt service routine.

Register values we are using to communicate between USB controller and Microblaze:

00: Command Interface
04: FIFO Read via End Point 2
08: FIFO Read via End Point 6
10: To_C State -> interrupt indication
14: Data_To_C State -> Data communication between vhdl and c
18: Empty Flag State

USB FIFO address line setting

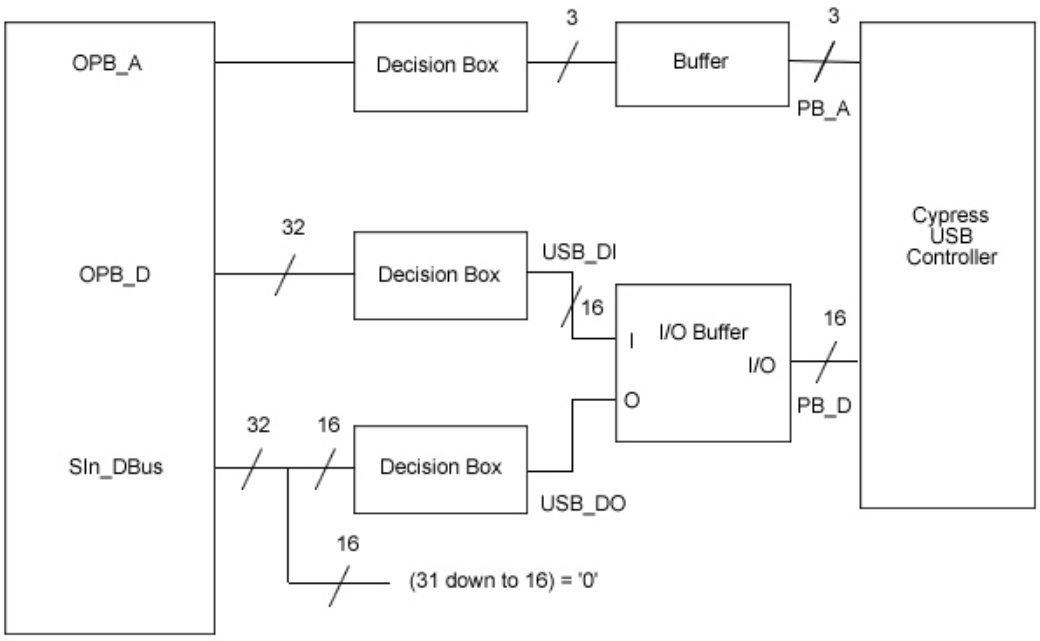
100: Command Interface
010: FIFO write
000: FIFO read from To_C state, Data_To_C state, and Empty Flag State

Command Protocol:

For the command address read or write byte, the first bit signifies address transfer or data transfer, the second bit signifies a read or write command and the next six bits represent the register address. For the command data write byte, the first bit signifies a data transfer and is split into two different bytes, where each of the four least significant bits contain half of the data that is to be transferred.

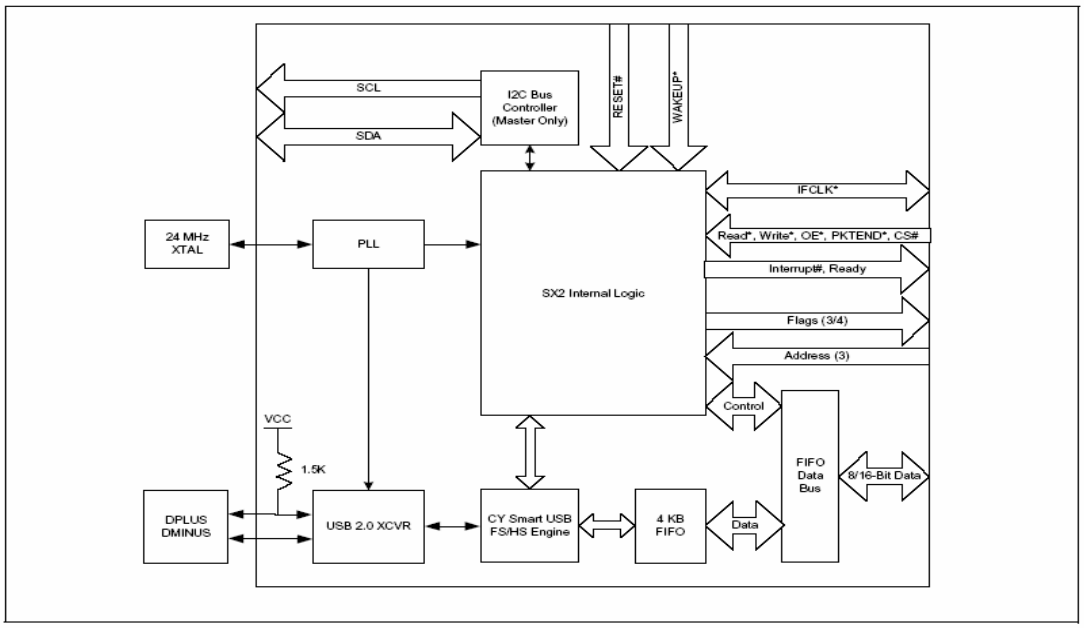
Enumeration:

A descriptor of the device, via a C program, was written and manually loaded in the USB interface rather than using the default descriptor given by the Cypress documentation.

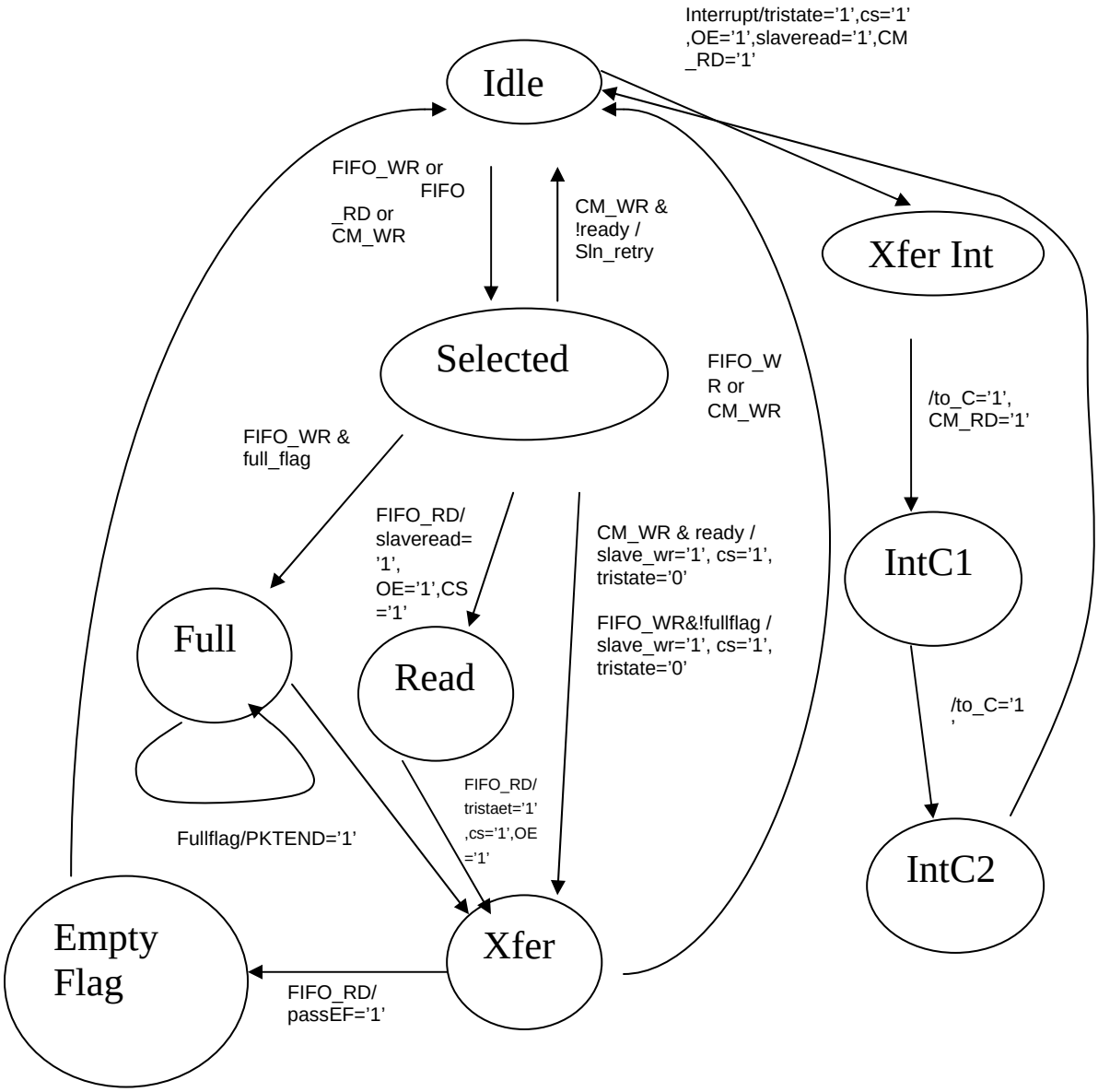


Block Diagram of OPB and USB Controller

Two modes of transfer over the USB were used: control and interrupt. Control transfer was used by the host to receive information about the device and configure it. It was used to initialize and report the status of the device. Interrupt transfer was used to actually send data over the USB. In this mode of transfer, data transfer is initialized in the opposite direction; data is retrieved from the device rather than having the information sent from the device.



Block Diagram of EZ-USB SX2



Finite State Machine for USB Controller

3. XMMS Controller

XMMS is a Linux based multimedia player based on Winamp. Originally, we planned on using Winamp. However, USB interfacing in Windows is significantly more complicated than in Linux. Therefore, we decided to use Linux and XMMS due to its similarity to Winamp.

Conveniently, XMMS can be operated using command line arguments, making it rather simple to send commands to the player once a signal is sent. The –p, –s, –f and –e switches can be used to perform the functionalities we desire. These correspond to play,

stop, forward and enqueue. Every time a command line argument is entered, XMMS automatically clears the playlist. The enqueue switch is used in order to prevent XMMS from clearing the playlist.

Our XMMS controller thus utilizes any keyboard like input, interprets the input, and then sends the respective command to XMMS.

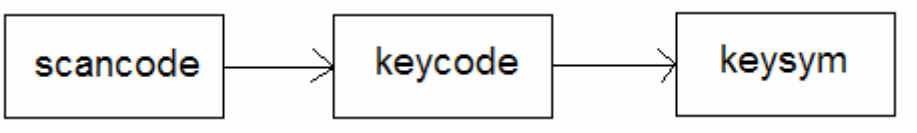
Once the USB transfer device is implemented, we decided that the best way for the Linux machine to receive the input would be to disguise the MicroBlaze as a Human Interface Device (HID). HIDs have their own device class definitions already installed on all major operating systems, so a separate driver class would not have to be written (which we actually started to do for a Windows environment, before we decided that Kernel programming was beyond what we were capable of). In particular, Linux comes with built in HID Usage Tables and convenient commands (ex. /sbin/lsusb) and files (ex. /proc/bus/usb/) that show very detailed reports and attributes whenever a USB device is inserted.

Since we had decided to write our XMMS controller to take in keyboard inputs, we decided to mimic a HID keyboard for our MicroBlaze. By obtaining an USB keyboard and running lsusb on the Linux machine, much of the required transfer variables were obtained. Combined with other information located throughout various Linux files, we "backwards engineered" the HID device and set the same variables into the MicroBlaze. That way, when the Linux machine queries the MicroBlaze for product information, the MicroBlaze now passes the information that a HID Keyboard would pass, and so Linux thinks a USB HID Keyboard has just been connected.

4. Translation of Information

Once the connection is made between the USB device and Linux machine, we must not only establish how the data is transferred (byte orders, memory size), but what data is transferred (if we receive 0011 as the data, filtering out header bytes, etc., what does that translate to?). For this, we examined how our keyboard interacted with Linux and once again mimic this behavior on the MicroBlaze.

Every time a key is pressed on the keyboard, a scancode is transmitted. When the key is released, another scancode is transmitted (released scancode = original scancode + 0x80). The Linux machine then takes the received scancode and, using a mapping of scancodes to keycodes viewable through *getkeycodes*, maps it into an equivalent keycode. The keycode is then finally mapped to a keysym, which is the output we traditionally see (the ASCI outputs, keyboard symbols, etc.)



Our XMMS controller takes a finite set of keysyms, and so we are interested in being able to mimic the generation of these specific keysyms from the MicroBlaze. Working backwards then, we derive the original scancodes and send them from the MicroBlaze to the Linux machine whenever a push button is pressed. Thankfully, Linux has a built in function that enables you to see the scancode whenever a button is pressed (must be under root). Using this feature, we establish the three buttons we want (inputs chosen in order to correspond to c_source_file for XMMS controller).

Input	Initial_scancode	End_scancode
1	0x29	0xA9
2	0x02	0x82
3	0x03	0x83

Problems Encountered

Since there was no precedent in the USB device on the FPGA, we decided that we needed to read up on all the background information available. Thus, right after the completion of lab 6, we all met up to divide up the work. Out of our group team members, we divided into two smaller groups - one to work on the Microblaze side and one to work on the operating system side. Both groups were unfamiliar with USB protocols and found reading material to familiarize themselves with the device. By the 75% demo, we had just finished reading books, manuals, and documentation and started to actually code for the Microblaze. We realized we had spent too much time reading more material than we needed to actually complete the project.

When we hit an impasse in the VHDL side of our project, we recognized that we needed help. We were in constant contact with Marcio, at both his office hours each week, to enlist his aid with any and all problems we were confronted with. However, even with Marcio's help and countless hours spent in the lab working, we were still unable to get the Microblaze portion of our project to work.

VHDL:

The failure of the VHDL code has been the obstacle for the project to move forward. The USB peripheral code is not able to communicate with the USB controller. We attempted different approaches during the process, including different operating mechanisms in the finite state machines but we were not able to write to and read from the registers in the USB controller at the end. We suspected that the OPB bus was not working properly but we tried to fetch values back to the Sln_DBus for the different OPB address input and we successfully received the values in the Microblaze. So our speculation is that the main problem lies in the finite state machines mechanism that

communicates with the USB controller. Another problem that we encountered is that to perform a read in the command interface at the USB controller, the VHDL code has to enter the finite state machine cycle in response to an interrupt signal from the USB controller. What we decided to do is that after the VHDL code stores in a register the input data from the USB controller, the Microblaze performed a check to see if they contain values, and read from it if it does. This mechanism generated another problem for us in coding the Microblaze in terms of deciding how often and where we should check for an interrupt signal. But we did not discover this problem until later in the process. Using an ISR could have been a better solution to this problem.

Operating System Side:

Since we had divided our project into two portions, the Microblaze side and the operating system side, and since the operating system side was largely contingent upon completion of the Microblaze side, this bottleneck made us rethink our approach on the operating system side. Instead of waiting for the Microblaze side to be finished, we tried to get as many pieces on the operating system side finished as possible. We decided to write a program which would control the XMMS player and to simulate the pushbuttons with a USB device. Our belief was that if we could simulate the pushbuttons with another USB device, when the Microblaze side was written, we could send the same recognition information that device would send to effectively trick the operating system into believing it was the USB device. This simulated device would only have three functionalities to simulate the three pushbuttons available on the Microblaze board. These functionalities would mimic three codes that the USB device would send. Therefore, by writing a program which could interpret the codes sent, we could finish a portion of the project which would easily be inserted when the USB functionality was finished.

We considered using several devices that the Microblaze could mimic. Initially, we considered using a USB keyboard. However, we were unable to find anyone who was in possession of a USB keyboard. We next turned to USB mice which are much more readily available than their keyboard counterparts. Due to our limited knowledge of how USB mice events were handled in C programs, we decided against using a USB mouse. Next, we chose to use a PS/2 keyboard because we were able to find information on how keycodes were sent and rationalized that the process would be similar to a USB keyboard. We were able to write a program which read keyboard strokes from a PS/2 keyboard and subsequently controlled the XMMS player.

Lessons Learned

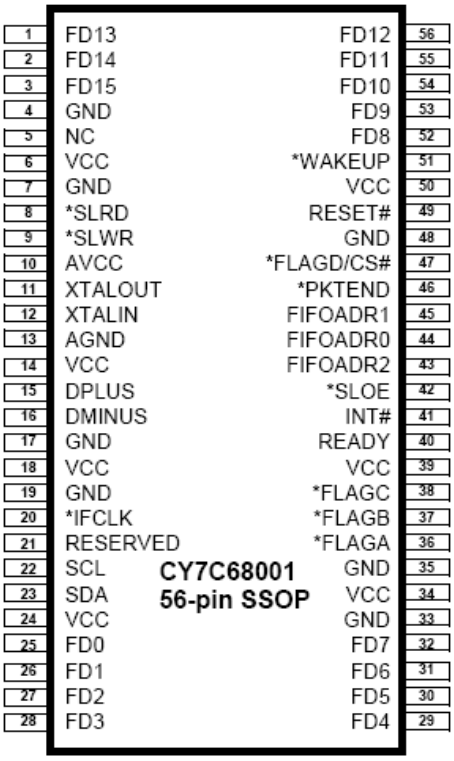
With a group as large as six members, it is difficult to keep in constant communication. At the beginning of our project, we did not set up weekly meetings or even determine a constant mode of communication, which was one of the largest obstacles in our project. We learned too late that it was important to get everyone together in one room to discuss any difficulties encountered and any progress made.

Advice

As with any large project, it is better to start as early as possible. Our project was divided onto two fronts, the Microblaze side and the operating system side, where the latter was dependent on the implementation and thus the completion of the former. The implementation of reading from the USB was based on the how the USB device was enumerated as well as how the information was to be sent.

Since we were the sole group working with USB this semester and there were no previous projects done on it, there was a lot of research done prior and during the project. It is imperative that you familiarize yourself with the various manuals and data sheets on the devices/chips that are associated with the project. We spent a large majority of the time looking for information that was already in the manuals.

Appendix



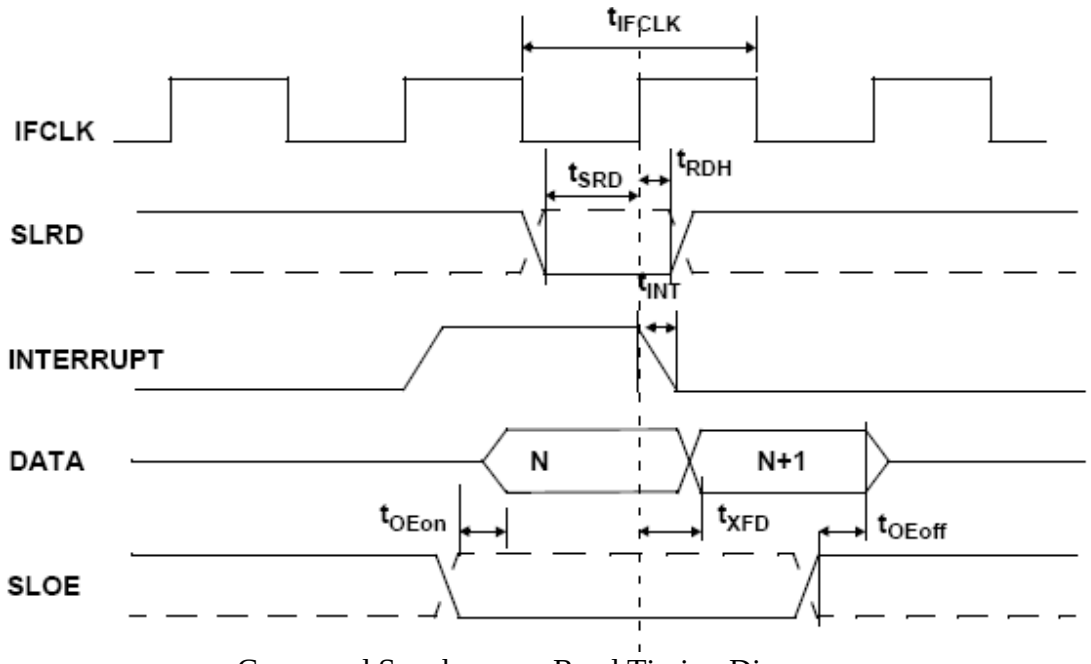
CY7C68001 56-pin SSOP Pin Assignment

The connections of the USB chip to the FPGA are listed below:

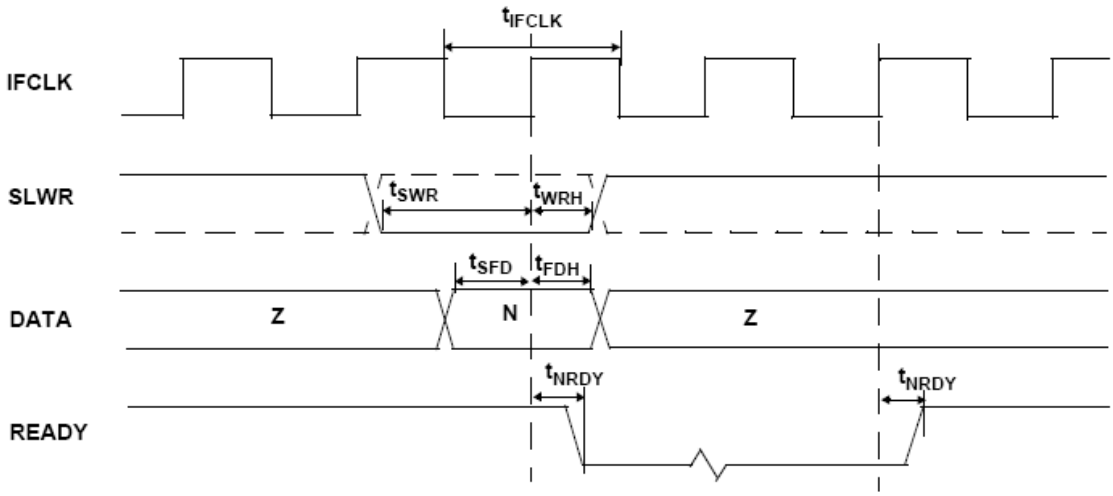
Peripheral Bus	USB Bus	USB Chip Pin	FPGA Pin	Function
	USB-IFCLK	IFCLK	183	Input/output clock for the USB interface
	USB-CS#	CS#	148	USB interface chip-select
PB-OE#		SLOE	125	FIFO/command data bus output-enable
PB-A18		SLRD	121	FIFO/command read input
PB-A19		SLWR	122	FIFO/command write input
PB-A17		PKTEND	115	End-of-packet input
	USB-FLAGA	FLAGA	152	Programmable flag status output
	USB-FLAGB	FLAGB	152	FIFO full status output
	USB-FLAGC	FLAGC	151	FIFO empty status output
	USB-READY	READY	150	Insert a wait state during R/W operations
	USB-INT#	INT#	149	Interrupt request output
PB-A0		SA0	83	FIFO/command address line 0
PB-A1		SA1	84	FIFO/command address line 1
PB-A2		SA2	86	FIFO/command address line 2
PB-D0		SD0	153	FIFO/command data line 0
PB-D1		SD1	145	FIFO/command data line 1
PB-D2		SD2	141	FIFO/command data line 2
PB-D3		SD3	135	FIFO/command data line 3
PB-D4		SD4	126	FIFO/command data line 4
PB-D5		SD5	120	FIFO/command data line 5
PB-D6		SD6	116	FIFO/command data line 6
PB-D7		SD7	108	FIFO/command data line 7
PB-D8		SD8	127	FIFO/command data line 8
PB-D9		SD9	129	FIFO/command data line 9
PB-D10		SD10	132	FIFO/command data line 10
PB-D11		SD11	133	FIFO/command data line 11
PB-D12		SD12	134	FIFO/command data line 12
PB-D13		SD13	136	FIFO/command data line 13
PB-D14		SD14	138	FIFO/command data line 14
PB-D15		SD15	139	FIFO/command data line 15
	USB-VBUS	NC	184	USB power-sense

Address/Selection	FIFOADR2	FIFOADR1	FIFOADR0
FIFO2	0	0	0
FIFO4	0	0	1
FIFO6	0	1	0
FIFO8	0	1	1
COMMAND	1	0	0
RESERVED	1	0	1
RESERVED	1	1	0
RESERVED	1	1	1

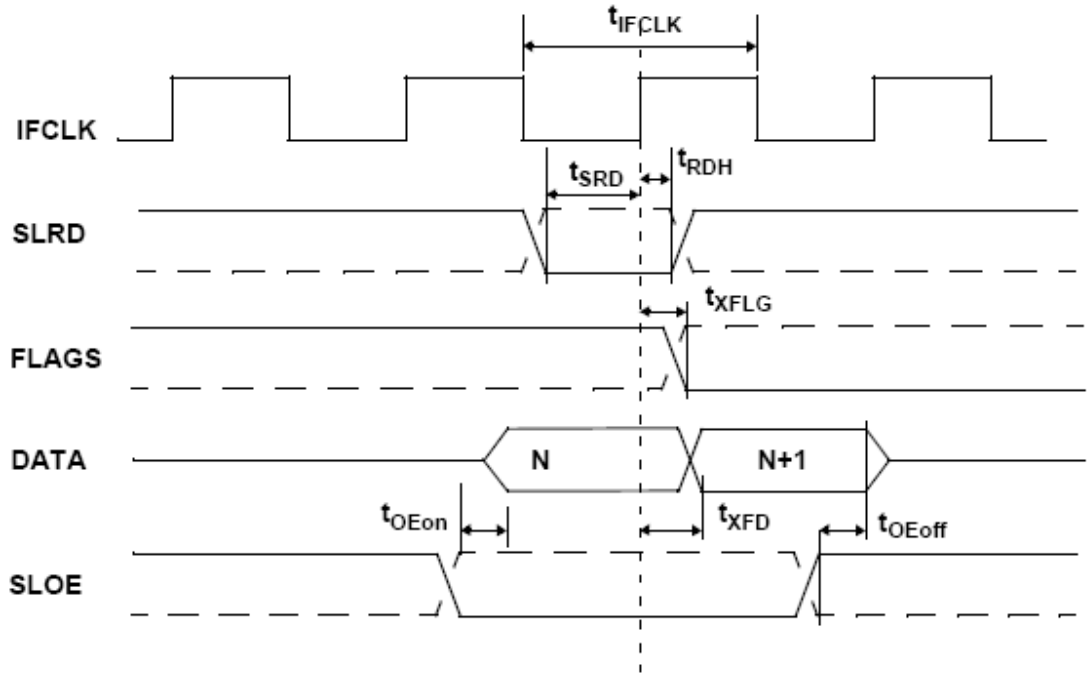
FIFO Address Lines Setting



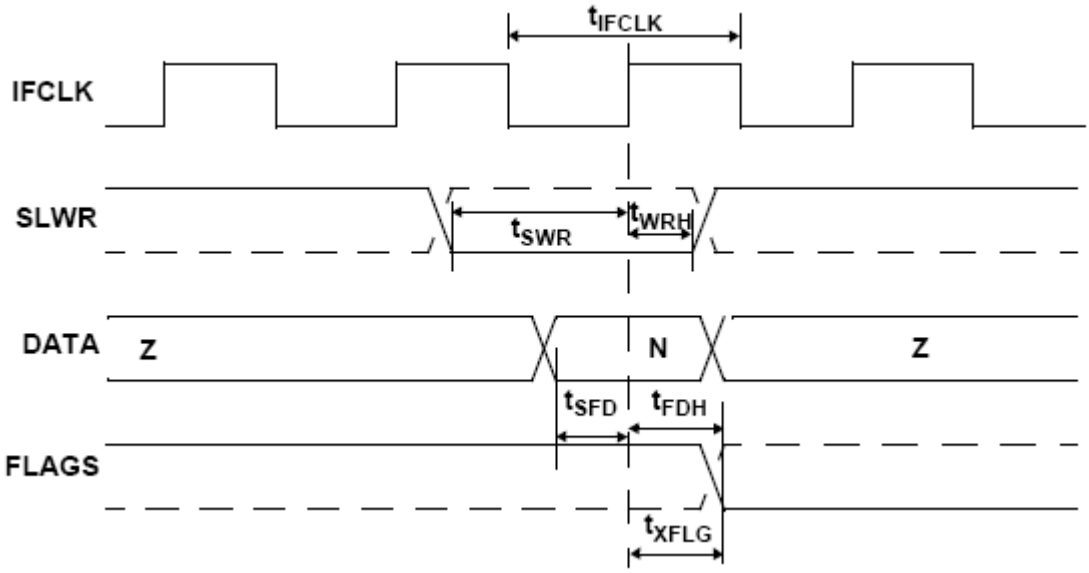
Command Synchronous Read Timing Diagram



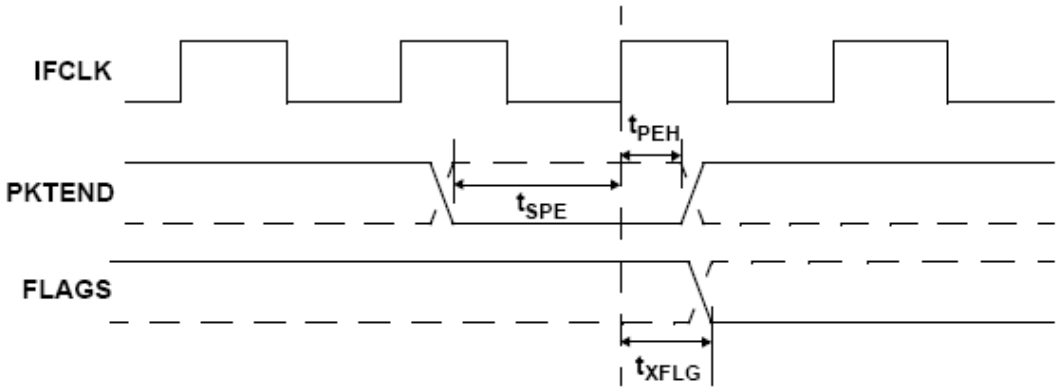
Command Synchronous Write Timing Diagram



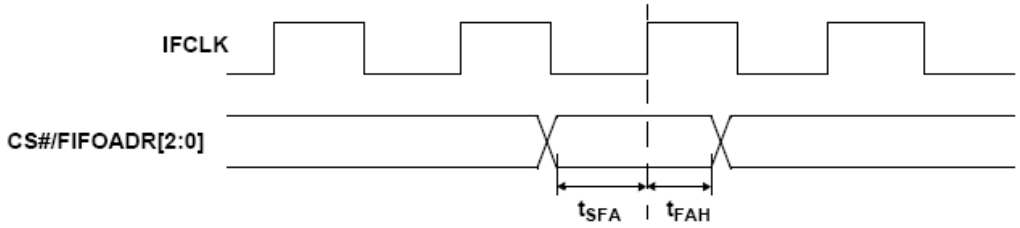
Slave FIFO Synchronous Read Timing Diagram



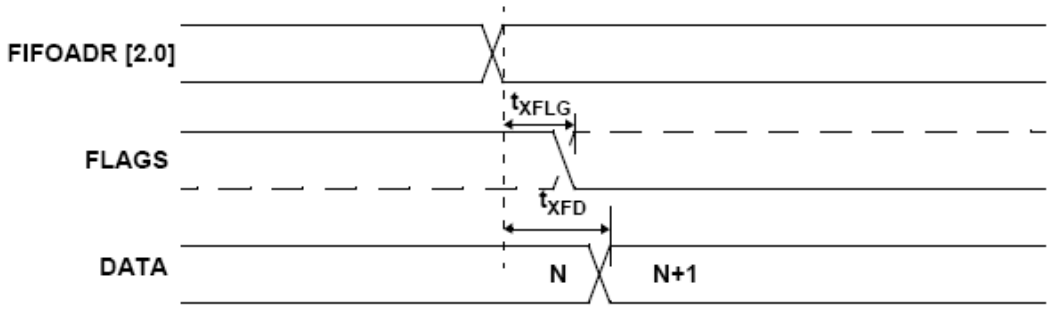
Slave FIFO Synchronous Write Timing Diagram



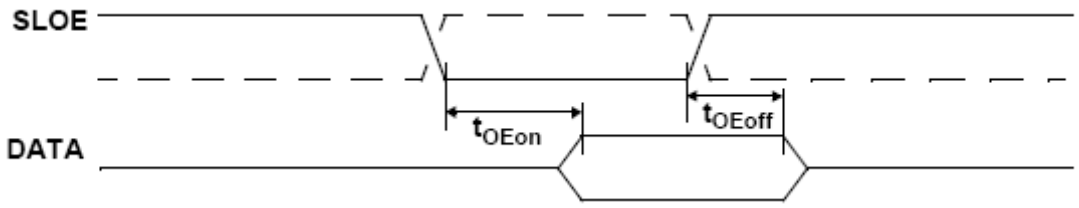
Slave FIFO Synchronous Packet End Strobe Timing Diagram



Slave FIFO Synchronous Address Timing Diagram



Slave FIFO Address to Flags/Data Timing Diagram



Slave FIFO Output Enable Timing Diagram

Files

USB Controller Source Code

main.c

```
#include "xparameters.h"
#include "xbasic_types.h"
#include "xio.h"

#define ADDR_CM 0x01800000  // XIO_Out
#define ADDR_FIFO_RD 0x01800004 // XIO_In
#define ADDR_FIFO_WR 0x01800008 // X_IO_Out
#define ADDR_SIG_INTERRUPT 0x01800010
#define ADDR_READ_INTERRUPT 0x01800014
#define ADDR_EMPTY_FLAG 0x01800018
#define DESC_LENGTH 0x42
#define BASE_ADDR 0x01800000

#define TEMP1 0x01800004
#define TEMP2 0x01800000
#define TEMP3 0x01800008

Xuint8 short_descriptor[6]= {
180,
4,
2,
16,
1,
0
};

Xuint8 descriptor[66] = {
// Descriptor for USB Remote Control - values written in integer format
// Philip Li
//  Device Descriptor  (ALL in INTEGER FORMAT)
18,      //descriptor length
01,      //descriptor type
00,02,         //specification version BCD
00,      //device class
00,      //device subclass
00,      //protocol
64,      //max packet size for EP0
64, 64,  //VID Vendor ID
64, 64,  //PID Product ID
64, 64,  //Device ID
00,      //Manucfacturer String Index
00,      //Product String Index
01,      //Number of configurations
// DeviceQualDscr
10,         //descriptor length
06,         //descriptor type
00,02,            //BCD
00,         //device class
00,         //device subclass
```

```
00,         //device sub-sub-class
64,         //max packet size
01,         //number of configuration
00,         //reserved
//HighSpeedConfigDscr
9,    //Dscr length
02,   //Dscr type
46,00,      //total data length ( 46 bytes)
1,    //number of interfaces
1,    //configuration number
0,    //configuration string
160,  //attibutes self powered, no remote wakeup 10100000
50,   //power requirement (50mA)
//Interface Descriptor
9,        //Dscr length
04,       //Dscr type
00,       //#index of interface
00,       //alternate setting
02,       //num of endpoints
03,       //interface class (HID)
00,       //interface subclass
01,       //interface sub-subclass (ProtocolCcode,0-None,1-Keyboard,2-
Mouse)
00,       //interface dscr string index
//Class Descriptor (HID interface)
9,    //Dscr length
33,   //Dscr type - HID 0x21
00,01,      //HID spec release number
00,   //hardware target country
01,   //number of HID class dscr to follow
34,   //Descriptor type 0x22  ???
//total length of report descriptor
// Endpoint Descriptor EP2
07,       // Dscr length
05,       // Dscr type
02,       // Endpoint number and direction
03,       // Endpoint type  0000 0011 - OUT interrupt transfer (Read
p.113 USBComplete)
00,       // Max packet size LSB 512bytes
02,       // Max packet size MSB (0x0200)
01,       // polling interval maximum latency p.114 USBComplete
// Endpoint Descriptor EP6
07,       // Dscr length
05,       // Dscr type
134,      // Endpoint number and direction
03,       // Endpoint type  1000 0110 -IN interrupt transfer (Read
p.113 USBComplete)
00,       // Max packet size LSB 512bytes
02,       // Max packet size MSB (0x0200)
01        // polling interval maximum latency p.114 USBComplete
};

// USB address is from 0000 0001 1000 0000 to 0000 0001 1000 0111
int main()
{
  // variables for reading DSCR file
```

```
  //variables

  int ready=0;
  int i;
  Xuint32 incoming_byte;
  Xuint32 incoming_byte1;
  Xuint32 incoming_byte2;
  Xuint32 incoming_byte3;
  Xuint32 incoming_byte4;
  Xuint32 incoming_byte5;
  Xuint32 incoming_byte6;
  Xuint8 incoming_byte8;
  Xuint8 incoming_byte7;
  Xuint8 outgoing_byte;


  // write register 0x1
  outgoing_byte = 0x5;
  write_register(0x01, outgoing_byte);

  print("write 0xf2 to register0x07\r\n");
  outgoing_byte = 0xf2;   // 11110010
  write_register(0x06, outgoing_byte);

  print("finish writing\r\n");

  for(i=0; i<1000; i++){}
  incoming_byte7 = read_register(0x07);
  print("\r\n");
  print("read from the register and the byte is \r\n");
  putnum(incoming_byte6);

  print("write the descriptor now\r\n");
  write_descriptor2();

}



Xuint8 read_register ( Xuint8 regaddress ){
  int ready = 0;
  Xuint32 incoming;
  low_level_command_write(0xC0 + regaddress);
  while( ready == 0){
    XIo_In32(ADDR_CM);
    incoming =  XIo_In32(ADDR_SIG_INTERRUPT);
    print("\r\n");
    print("addr_sig_interrupt is \r\n");
    putnum(incoming);
    if (incoming != 0x0)ready =1;
  };

  incoming =  XIo_In32(ADDR_READ_INTERRUPT);
  print("incoming\r\n");
  putnum(incoming);
  return incoming;
}
```

```
void write_register (Xuint8 regaddress, Xuint8 data){
  low_level_command_write(0x80 + regaddress); // command address byte
  low_level_command_write(0xF & (data>>4)); // command data byte one
  low_level_command_write(0xF & data); // command data byte two
}

void low_level_command_write(Xuint8 data){
  Xuint32 outgoing_byte;
  outgoing_byte = (0xFF & data);
  putnum(outgoing_byte);
  print("\r\n");
  //  XIo_In32(ADDR_CM);
  XIo_Out32(ADDR_CM, outgoing_byte);
}

void write_descriptor2(){
  Xuint8 lengthOfRegister = 0x06; // Descriptor length = 6
  int i;
  low_level_command_write(0x80 + 0x30); // command address byte 0x30 is
the Desc RAM
  low_level_command_write(lengthOfRegister);
  low_level_command_write(0);

  for ( i = 0; i <6 ; i ++){
    low_level_command_write(short_descriptor[i]);
  }

}

void write_descriptor(){
  Xuint8 lengthOfRegister = 0x42; // Descriptor length = 66
  int i;
  low_level_command_write(0x80 + 0x30); // command address byte 0x30 is
the Desc RAM
  low_level_command_write(lengthOfRegister);
  low_level_command_write(0);

  for ( i = 0; i <66 ; i ++){
    low_level_command_write(descriptor[i]);
  }

}
```

opb_usb.vhd

```
------------------------------------------------------------------------
--------
--
-- OPB Peripheral:  USB controller
--
-- Philip Li (pcl2007), Yanjie Ma (ym2009), Pamela Lee (pyl2001)
--
------------------------------------------------------------------------
--------
library ieee;
use ieee.std_logic_1164.all;

entity opb_usb is

  generic (
    C_OPB_AWIDTH : integer                   := 32;
    C_OPB_DWIDTH : integer                   := 32;
    C_BASEADDR   : std_logic_vector(0 to 31) := X"00000000";
    C_HIGHADDR   : std_logic_vector(0 to 31) := X"FFFFFFFF");

  port (
    OPB_Clk     : in  std_logic;
    OPB_Rst     : in  std_logic;
    OPB_ABus    : in  std_logic_vector(31 downto 0);
    OPB_BE      : in  std_logic_vector(3 downto 0);
    OPB_DBus    : in  std_logic_vector(31 downto 0);
    OPB_RNW     : in  std_logic;
    OPB_select  : in  std_logic;
    OPB_seqAddr : in  std_logic;        -- Sequential Address
    Sln_DBus    : out std_logic_vector(31 downto 0);
    Sln_errAck  : out std_logic;        -- (unused)
    Sln_retry   : out std_logic;        -- (unused)
    Sln_toutSup : out std_logic;        -- Timeout suppress
    Sln_xferAck : out std_logic;        -- Transfer acknowledge

    PB_D        : inout std_logic_vector(15 downto 0);
    PB_A        : out std_logic_vector(2 downto 0);
    PB_OE       : out std_logic;
    PB_SLRD     : out std_logic;
    PB_SLWR     : out std_logic;
    PB_PKTEND   : out std_logic;
    USB_IFCLK   : out std_logic;
    USB_CS      : out std_logic;
    USB_FLAGA   : in std_logic;
    USB_FLAGB   : in std_logic;
    USB_FLAGC   : in std_logic;
    USB_READY   : in std_logic;
    USB_INT     : in std_logic;

    BUTTON_S1   : in std_logic;
    BUTTON_S2   : in std_logic;
    BUTTON_S4   : in std_logic

    );
```

```
end opb_usb;

-------------------------------------------------------------------------
--------
-------------------------------------------------------------------------
--------

architecture Behavioral of opb_usb is

  constant USB_AWIDTH : integer := 3;  -- Number of address lines on
the USB
  constant USB_DWIDTH : integer := 16;  -- Number of data lines on the
USB

  component OBUF_F_24
    port (
      O : out STD_ULOGIC;               -- the pin
      I : in STD_ULOGIC);               -- signal to pin
  end component;
-------------------------------------------------------------------------
--------
  component BUF
    port (
      O : out STD_ULOGIC;               -- the pin
      I : in STD_ULOGIC);               -- signal to pin
  end component;
-------------------------------------------------------------------------
--------
  component IOBUF_F_24
    port (
      O : out STD_ULOGIC;               -- signal from pin
      IO : inout STD_ULOGIC;               -- the pin
      I : in STD_ULOGIC;               -- signal to pin
      T : in STD_ULOGIC                 -- 1 = drive IO with I
      );
  end component;

  signal USB_DI, USB_DO : std_logic_vector(15 downto 0);
  signal ABus : std_logic_vector(2 downto 0);
  signal interrupt_byte : std_logic_vector(15 downto 0);
  signal Addr8bit  : std_logic_vector(7 downto 0);

  signal tristate, to_C : std_logic;
  signal FIFO_RD, FIFO_WR, CM_RD, CM_WR, DATA_TO_C_STATE, TO_C_STATE,
EMPTY_FLAG_STATE : std_logic;
  signal LAST_PKT_STATE : std_logic;
  signal pass_flagEF, empty_flag, full_flag, empty_flag_temp :
std_logic;
  signal chipselect, rnw : std_logic;
  signal output_enable, slave_write, slave_read, cs : std_logic;
  signal interrupt, ready,  pktend, last_pkt, j, k : std_logic;
  signal to_C_out, to_C_1 : std_logic;
  signal interrupt_byte_out  : std_logic_vector(15 downto 0);

  -- Sln_xferAck is generated directly from state bit 1 in some cases
  constant STATE_BITS : integer := 4;
```

```
  constant Idle      : std_logic_vector(0 to STATE_BITS-1) := "0000";
  constant Read      : std_logic_vector(0 to STATE_BITS-1) := "0010";
  constant Xfer      : std_logic_vector(0 to STATE_BITS-1) := "0110";
  constant IntC1     : std_logic_vector(0 to STATE_BITS-1) := "1000";
  constant IntC2     : std_logic_vector(0 to STATE_BITS-1) := "1010";
  constant Emptf     : std_logic_vector(0 to STATE_BITS-1) := "1001";
  constant Selected  : std_logic_vector(0 to STATE_BITS-1) := "1011";
  constant XferInt   : std_logic_vector(0 to STATE_BITS-1) := "0011";
  constant Full      : std_logic_vector(0 to STATE_BITS-1) := "0001";
  signal present_state, next_state : std_logic_vector(0 to 3);

  -- Critical: Sln_xferAck is generated directly from state bit 0!
  constant Idle2     : std_logic_vector(0 to 2) := "000";
  constant Selected2 : std_logic_vector(0 to 2) := "001";
  constant Xfer2     : std_logic_vector(0 to 2) := "111";

  signal present_state2, next_state2 : std_logic_vector(0 to 2);

begin

  databus : for i in 0 to 15 generate

    data_buffer : IOBUF_F_24 port map (
      O => USB_DO(i),
      IO => PB_D(i),
      I => USB_DI(i),
      T => tristate);
  end generate;

  addrbus : for j in 0 to 2 generate

    addr_buffer : OBUF_F_24 port map (
      O => PB_A(j),
      I => ABus(j));
  end generate;

  to_C_buffer : BUF port map (
    O => to_C_out,
    I => to_C_1);

  chipselect <= OPB_select when OPB_ABus(31 downto 16) =
"0000000110000000" else '0';
  rnw <= OPB_RNW;

-- Selectively passing OPB_ABUS to USB_A
  Addr8bit <= OPB_ABUS(7 downto 0) when chipselect='1' else X"00";
  -- pass address ABus and select types of transfer
  -- List of Register Values to communicate with Microblaze
  -- Addr3bit   = X"00" => Command Interface    XOut  CM_WR
  --            = X"04" => FIFO_RD EndPoint 2   XIn   FIFO_RD
  --            = X"08" => FIFO_WR EndPoint 6   XOut  FIFO_WR
  --            = X"10" => to-C                 XIn   TO_C_STATE
  --            = X"18" => empty_flag           XIn   EMPTY_FLAG_STATE
  --            = NOT USED "110" => full_flag   XIn   FULL_FLAG_STATE
  --            = X"14" => data_to_C            XIn   DATA_TO_C_STATE
  ------------------------------------------------------------------------
--------
```

```
  addr_reg_and_select_transfer : process (OPB_Clk, OPB_Rst)
  begin  -- process
    if OPB_Rst = '1' then
      ABus <= (others => '0');
      FIFO_RD <= '0';
      FIFO_WR <= '0';
      CM_WR <= '0';
      EMPTY_FLAG_STATE <= '0';
      DATA_TO_C_STATE <= '0';
      TO_C_STATE <='0';

    elsif OPB_Clk'event and OPB_Clk = '1' then
        if CM_RD = '1' then             -- pass the Address for CM_RD
transfer
          -- mode, triggered by CYPRESS
          ABus <= "100";
        elsif chipselect ='1' then
          if Addr8bit = X"00" and rnw = '0' then      -- addr for
command interface
            ABus <= "100";
          elsif Addr8bit = X"04" then   -- addr for FIFO Read
            ABus <= "000";
          elsif Addr8bit = X"08" then   -- addr for FIFO write
            ABus <= "010";
          else ABus <= "000";           -- otherwise just 0
          end if;

        end if;
        -- determine the operating mode here

        if chipselect = '1' then
          if Addr8bit = X"04" and rnw = '1'  then  -- Set FIFO_RD
            FIFO_RD <= '1';
          elsif Addr8bit = X"10" and rnw = '1' then  --Set TO_C_STATE
            TO_C_STATE <= '1';
          elsif Addr8bit = X"14" and rnw = '1' then  -- Set DATA_TO_C
            DATA_TO_C_STATE <= '1';
          elsif Addr8bit = X"08" and rnw = '0' then  --Set FIFO_WR
            FIFO_WR <= '1';
          elsif Addr8bit = X"00" and rnw = '0' then  --Set CM_WR
            CM_WR <= '1';
          elsif Addr8bit = X"18" and rnw = '1' then  --Set
EMPTY_FLAG_STATE
            EMPTY_FLAG_STATE <= '1';
          else
            FIFO_RD <= '0';
            FIFO_WR <= '0';
            CM_WR <= '0';
            DATA_TO_C_STATE <= '0';
            TO_C_STATE <= '0';
            EMPTY_FLAG_STATE <= '0';
          end if;
        end if;
    end if;
  end process addr_reg_and_select_transfer;

  data_register_opb_inputs: process (OPB_Clk, OPB_Rst)
```

```
  begin
    if OPB_Rst = '1' then
      USB_DI <= (others => '0');
    elsif OPB_Clk'event and OPB_Clk = '1' then
      if FIFO_WR = '1' and CM_WR = '1' then
        USB_DI <= OPB_DBus(15 downto 0);
      else
        USB_DI <= "0000000000000000";
      end if;
    end if;
  end process data_register_opb_inputs;

  data_register_opb_outputs: process (OPB_Clk, OPB_Rst)
  begin
   if OPB_Rst = '1' then
      Sln_DBus(15 downto 0) <= (others => '0');
   elsif OPB_Clk'event and OPB_Clk = '1' then
     if chipselect ='1' then
       if FIFO_RD = '1' then
         Sln_DBus(15 downto 0) <= USB_DO; -- X"A101"
       elsif TO_C_STATE = '1' then
        Sln_DBus(15 downto 0)  <= "000000000000000" & to_C_out; --
X"A102"; --
       elsif DATA_TO_C_STATE = '1' then
        Sln_DBus(15 downto 0) <=  interrupt_byte_out;          --
X"A103";
       elsif EMPTY_FLAG_STATE = '1' then
        Sln_DBus(15 downto 0) <= "000000000000000" & empty_flag; --
X"A104"; --
       else
         Sln_DBus(15 downto 0) <= (others => '0');
       end if;
     else
       Sln_DBus(15 downto 0) <= (others => '0');
     end if;
   end if;
  end process data_register_opb_outputs;

  Sln_DBus(31 downto 16) <= (others => '0');

  interrupt_signals_and_output_to_OPB : process (OPB_Clk, OPB_Rst)
  begin  -- process
    if OPB_Rst = '1' then
      interrupt_byte <= (others => '0');
    elsif OPB_Clk'event and OPB_Clk = '1' then
      if present_state(1) = '1' and CM_RD = '1' then
        interrupt_byte_out <=  "00000000" & USB_DO(7 downto 0);
      end if;
    end if;
  end process interrupt_signals_and_output_to_OPB;

  interrupt_to_C_output_to_OPB : process (OPB_Clk, OPB_Rst)
  begin  -- process
    if OPB_Rst = '1' then               -- asynchronous reset (active
low)
      to_C_1 <= '0';
    elsif OPB_Clk'event and OPB_Clk = '1' then
```

```
      if to_C = '1' then
        to_C_1 <= '1';
      end if;
    end if;
  end process;

  empty_flag_process : process (OPB_Clk, OPB_Rst)
  begin  -- process
    if OPB_Rst = '1' then
      empty_flag <= '0';
    elsif OPB_Clk'event and OPB_Clk = '1' then
      if pass_flagEF = '1' then
      empty_flag <= empty_flag_temp;  -- empty_flag=1 => means FIFO
empty
      else
      empty_flag <= '0';
      end if;
    end if;
  end process empty_flag_process;
  empty_flag_temp <=
    '0' when USB_FLAGC = '1' else
    '1';

  -- Connecting OPB_Clk to CYPRESS CLK
  USB_IFCLK <= OPB_Clk;

  -- Unused outputs
  Sln_errAck  <= '0';
  Sln_toutSup <= '0';


PB_OE <=
  '0' when output_enable = '1'else
  '1';
PB_SLRD <=
  '0' when slave_read = '1' else
  '1';
PB_SLWR <=
  '0' when slave_write = '1' else
  '1';
PB_PKTEND <=
  '0' when pktend = '1' else
  '1';
USB_CS <=
  '0' when cs = '1' else
  '1';
interrupt <=
  '0' when USB_INT = '1'else
  '1';
full_flag <=
  '0' when USB_FLAGB ='1' else
  '0';
ready <= USB_READY;

  -- Sequential part of the FSM
  fsm_seq : process(OPB_Clk, OPB_Rst)
  begin
```

```
    if OPB_Rst = '1' then
      present_state <= Idle;
    elsif OPB_Clk'event and OPB_Clk = '1' then
      present_state <= next_state;
    end if;
  end process fsm_seq;

  -- Combinational part of the FSM
  fsm_comb : process(OPB_Select, OPB_Rst, present_state, interrupt,
FIFO_RD, FIFO_WR, CM_WR, CM_RD, ready, full_flag, last_pkt)
  begin

    output_enable <= '0';
    slave_read <= '0';
    slave_write <= '0';
    pktend <= '0';
    cs <= '0';
    tristate <= '0';
    to_C <= '0';
    pass_flagEF <= '0';                 -- Default Values
    CM_RD <= '0';
    Sln_retry <= '0';

    if OPB_RST = '1' then
      next_state <= Idle;
    else
      case present_state is

        when Idle =>
           if interrupt = '1' then
             tristate <= '1';
             output_enable <= '1';
             slave_read <= '1';
             CM_RD <= '1';
             cs <= '1';
             next_state <= XferInt;
           elsif ( FIFO_RD = '1'or FIFO_WR = '1' or CM_WR = '1') then
             next_state <= Selected;
           elsif (TO_C_STATE = '1' or  DATA_TO_C_STATE = '1' or
EMPTY_FLAG_STATE = '1') then
             next_state <= Selected;
           else
             next_state <= Idle;
           end if;

        when XferInt =>
             to_C <= '1';
             CM_RD <= '1';
             next_state <= IntC1;

        when IntC1 =>
             to_C <= '1';
             next_state <= IntC2;

        when IntC2 =>
             next_state <= Idle;
```

```
        when Selected =>
            if FIFO_RD = '1' then
              slave_read <= '1';
              output_enable <= '1';
              cs <= '1';
              next_state <= Read;

            elsif FIFO_WR = '1' then
              if full_flag = '0' then
                slave_write <= '1';
                cs <= '1';
                tristate <= '0';
                next_state <= Xfer;
              elsif full_flag = '1' then  -- full_flag, flush signals
to HOST
                pktend <= '1';
                next_state <= Full;
              end if;
            elsif CM_WR = '1' then
              if ready = '1' then
                slave_write  <= '1';
                cs <= '1';
                tristate <= '0';
                next_state <= Xfer;
              elsif ready = '0' then
                Sln_retry <= '1';     -- not ready, OPB_retry
                next_state <= Idle;
              end if;
            elsif (TO_C_STATE = '1' or  DATA_TO_C_STATE = '1' or
EMPTY_FLAG_STATE = '1') then
              next_state <= Xfer;
            else
              next_state <= Idle;
            end if;

        when Full =>
             if full_flag = '0' and FIFO_WR = '1' then
               slave_write <= '1';
               cs <= '1';
               tristate <= '0';
               next_state <= Xfer;
             elsif full_flag = '1' and FIFO_WR = '1' then
               pktend <= '1';
               next_state <= Full;
             else
               next_state <= Idle;
             end if;

        when Read =>
            if FIFO_RD = '1'  then
              tristate <= '1';
              cs <= '1';
              output_enable <= '1';
              next_state <= Xfer;
            else
              next_state <= Xfer;
            end if;
```

```
        when Xfer =>
            if CM_WR = '1' or FIFO_WR = '1' then
              next_state <= Idle;
            elsif FIFO_RD = '1' then
              pass_flagEF <= '1';
              next_state <= Emptf;
            elsif (TO_C_STATE = '1' or  DATA_TO_C_STATE = '1' or
EMPTY_FLAG_STATE = '1') then
              next_state <= Idle;
            else
              next_state <= Idle;
            end if;

        when Emptf =>                   -- EmptF lets C reads
empty_flag_signal
          next_state <= Idle;

        when others=>
          next_state <= Idle;

      end case;
    end if;
  end process fsm_comb;
  Sln_xferAck <= present_state(1);  -- present_state = Xfer

end Behavioral;

-- compile-command: "ghdl -a opb_usb.vhd"
-- End:
```

opb_usb_ v2_1_0.mpd

```
 ##############################################################################
##
## Microprocessor Peripheral Definition
##
##############################################################################

BEGIN opb_usb, IPTYPE = PERIPHERAL, EDIF=TRUE

BUS_INTERFACE BUS = SOPB, BUS_STD = OPB, BUS_TYPE = SLAVE

## Generics for VHDL
PARAMETER c_baseaddr     = 0xFFFFFFFF, DT = std_logic_vector, MIN_SIZE
= 0xFF
PARAMETER c_highaddr     = 0x00000000, DT = std_logic_vector
PARAMETER c_opb_awidth   = 32,         DT = integer
PARAMETER c_opb_dwidth   = 32,         DT = integer

## Ports
PORT opb_abus    = OPB_ABus,    DIR = IN, VEC = [0:(c_opb_awidth-1)],
BUS = SOPB
PORT opb_be      = OPB_BE,      DIR = IN, VEC = [0:((c_opb_dwidth/8)-
1)], BUS = SOPB
PORT opb_clk     = "",          DIR = IN,                      BUS =
SOPB
PORT opb_dbus    = OPB_DBus,    DIR = IN, VEC = [0:(c_opb_dwidth-1)],
BUS = SOPB
PORT opb_rnw     = OPB_RNW,     DIR = IN,
BUS = SOPB
PORT opb_rst     = OPB_Rst,     DIR = IN,
BUS = SOPB
PORT opb_select  = OPB_select,  DIR = IN,
BUS = SOPB
PORT opb_seqaddr = OPB_seqAddr, DIR = IN,
BUS = SOPB
PORT sln_dbus    = Sl_DBus,     DIR = OUT, VEC = [0:(c_opb_dwidth-1)],
BUS = SOPB
PORT sln_errack  = Sl_errAck,   DIR = OUT,
BUS = SOPB
PORT sln_retry   = Sl_retry,    DIR = OUT,
BUS = SOPB
PORT sln_toutsup = Sl_toutSup,  DIR = OUT,
BUS = SOPB
PORT sln_xferack = Sl_xferAck,  DIR = OUT,
BUS = SOPB


# User Ports
PORT PB_D   = "", DIR = INOUT, VEC = [0:15], 3STATE=FALSE,
IOB_STATE=BUF
PORT PB_A   = "",       DIR = OUT, VEC = [0:2], 3STATE=FALSE,
IOB_STATE=BUF
PORT PB_OE  = "",       DIR = OUT
PORT PB_SLRD      = "",       DIR = OUT
PORT PB_SLWR      = "",       DIR = OUT
```

```
PORT PB_PKTEND  = "",   DIR = OUT
PORT USB_IFCLK    = "",       DIR = OUT
PORT USB_CS       = "",       DIR = OUT
PORT USB_FLAGA    = "",       DIR = IN
PORT USB_FLAGB    = "",       DIR = IN
PORT USB_FLAGC    = "",       DIR = IN
PORT USB_READY    = "",       DIR = IN
PORT USB_INT      = "",       DIR = IN


PORT BUTTON_S1    = "",       DIR = IN
PORT BUTTON_S2  = "",   DIR = IN
PORT BUTTON_S4    = "",       DIR = IN

END
```

system.mhs

```
# Parameters
PARAMETER VERSION = 2.1.0

# Global Ports

PORT FPGA_CLK1 = FPGA_CLK1, DIR = IN
PORT RS232_TD = RS232_TD, DIR=OUT
PORT RS232_RD = RS232_RD, DIR=IN
PORT PB_D = PB_D, DIR=INOUT, VEC = [0:15]
PORT PB_A = PB_A, DIR=OUT, VEC = [0:2]
PORT PB_OE = PB_OE, DIR=OUT
PORT PB_SLRD = PB_SLRD, DIR=OUT
PORT PB_SLWR = PB_SLWR, DIR=OUT
PORT PB_PKTEND = PB_PKTEND, DIR=OUT
PORT USB_IFCLK = USB_IFCLK, DIR=OUT
PORT USB_CS = USB_CS, DIR=OUT
PORT USB_FLAGA = USB_FLAGA, DIR=IN
PORT USB_FLAGB = USB_FLAGB, DIR=IN
PORT USB_FLAGC = USB_FLAGC, DIR=IN
PORT USB_READY = USB_READY, DIR=IN
PORT USB_INT = USB_INT, DIR=IN
PORT BUTTON_S1 = BUTTON_S1, DIR=IN
PORT BUTTON_S2 = BUTTON_S2, DIR=IN
PORT BUTTON_S4 = BUTTON_S4, DIR=IN

# USB peripheral

BEGIN opb_usb

 PARAMETER INSTANCE = usb_peripheral
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x01800000
 PARAMETER C_HIGHADDR = 0x01800fff
 PORT OPB_Clk = sys_clk
 BUS_INTERFACE SOPB = myopb_bus
 PORT PB_D = PB_D
 PORT PB_A = PB_A
 PORT PB_OE = PB_OE
 PORT PB_SLRD = PB_SLRD
 PORT PB_SLWR = PB_SLWR
 PORT PB_PKTEND = PB_PKTEND
 PORT USB_IFCLK = USB_IFCLK
 PORT USB_CS = USB_CS
 PORT USB_FLAGA = USB_FLAGA
 PORT USB_FLAGB = USB_FLAGB
 PORT USB_FLAGC = USB_FLAGC
 PORT USB_READY = USB_READY
 PORT USB_INT = USB_INT
 PORT BUTTON_S1 = BUTTON_S1
 PORT BUTTON_S2 = BUTTON_S2
 PORT BUTTON_S4 = BUTTON_S4

END
```

```
# The main processor core

BEGIN microblaze
 PARAMETER INSTANCE = mymicroblaze
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_ICACHE = 0
 PORT Clk = sys_clk
 PORT Reset = fpga_reset
 BUS_INTERFACE DLMB = d_lmb
 BUS_INTERFACE ILMB = i_lmb
 BUS_INTERFACE DOPB = myopb_bus
 BUS_INTERFACE IOPB = myopb_bus
END

# Block RAM for code and data is connected through two LMB busses
# to the Microblaze, which has two ports on it for just this reason.

# Data LMB bus

BEGIN lmb_v10
 PARAMETER INSTANCE = d_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_data_controller
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00000FFF
 BUS_INTERFACE SLMB = d_lmb
 BUS_INTERFACE BRAM_PORT = conn_0
END

# Instruction LMB bus

BEGIN lmb_v10
 PARAMETER INSTANCE = i_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_instruction_controller
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00000FFF
 BUS_INTERFACE SLMB = i_lmb
 BUS_INTERFACE BRAM_PORT = conn_1
END

# The actual block memory

BEGIN bram_block
```

```
 PARAMETER INSTANCE = bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = conn_0
 BUS_INTERFACE PORTB = conn_1
END

# Clock divider to make the whole thing run

BEGIN clkgen
 PARAMETER INSTANCE = clkgen_0
 PARAMETER HW_VER = 1.00.a
 PORT FPGA_CLK1 = FPGA_CLK1
 PORT sys_clk = sys_clk
 PORT pixel_clock = pixel_clock
 PORT fpga_reset = fpga_reset
END

# The OPB bus controller connected to the Microblaze
# All peripherals are connected to this

BEGIN opb_v20
 PARAMETER INSTANCE = myopb_bus
 PARAMETER HW_VER = 1.10.a
 PARAMETER C_DYNAM_PRIORITY = 0
 PARAMETER C_REG_GRANTS = 0
 PARAMETER C_PARK = 0
 PARAMETER C_PROC_INTRFCE = 0
 PARAMETER C_DEV_BLK_ID = 0
 PARAMETER C_DEV_MIR_ENABLE = 0
 PARAMETER C_BASEADDR = 0x0fff1000
 PARAMETER C_HIGHADDR = 0x0fff10ff
 PORT SYS_Rst = fpga_reset
 PORT OPB_Clk = sys_clk
END

# UART: Serial port hardware

BEGIN opb_uartlite
 PARAMETER INSTANCE = myuart
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_CLK_FREQ = 50_000_000
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_BASEADDR = 0xFEFF0100
 PARAMETER C_HIGHADDR = 0xFEFF01FF
 PORT OPB_Clk = sys_clk
 BUS_INTERFACE SOPB = myopb_bus
 PORT RX=RS232_RD
 PORT TX=RS232_TD
END
```

system.mhs

```
PARAMETER VERSION = 2.2.0
PARAMETER HW_SPEC_FILE = system.mhs

BEGIN PROCESSOR
 PARAMETER HW_INSTANCE = mymicroblaze
 PARAMETER DRIVER_NAME = cpu
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN OS
  PARAMETER PROC_INSTANCE = mymicroblaze
  PARAMETER OS_NAME = standalone
  PARAMETER OS_VER = 1.00.a
  PARAMETER STDIN = myuart
  PARAMETER STDOUT = myuart
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = myuart
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.00.b
END

# Use null drivers for peripherals that don't need them
# This supresses warnings

BEGIN DRIVER
 PARAMETER HW_INSTANCE = usb_peripheral
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = lmb_data_controller
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = lmb_instruction_controller
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END
```

system.ucf

```
net sys_clk period = 25.000;

net FPGA_CLK1 loc="p77";

net RS232_TD loc="p71";
net RS232_RD loc="p73";


net PB_OE   loc="p125";
net PB_SLRD       loc="p121";
net PB_SLWR       loc="p122";
net PB_PKTEND     loc="p115";


net PB_A<0> loc = "p83";
net PB_A<1> loc = "p84";
net PB_A<2> loc = "p86";

net PB_D<0>       loc = "p153";
net PB_D<1>       loc = "p145";
net PB_D<2>       loc = "p141";
net PB_D<3>       loc = "p135";
net PB_D<4>       loc = "p126";
net PB_D<5>       loc = "p120";
net PB_D<6>       loc = "p116";
net PB_D<7>       loc = "p108";
net PB_D<8>       loc = "p127";
net PB_D<9>       loc = "p129";
net PB_D<10>      loc = "p132";
net PB_D<11>      loc = "p133";
net PB_D<12>      loc = "p134";
net PB_D<13>      loc = "p136";
net PB_D<14>      loc = "p138";
net PB_D<15>      loc = "p139";

net USB_IFCLK     loc="p163";
net USB_CS  loc="p148";

net USB_FLAGA     loc="p162";
net USB_FLAGB     loc="p152";
net USB_FLAGC     loc="p151";
net USB_READY     loc="p150";
net USB_INT loc="p149";

net BUTTON_S1     loc="p100";
net BUTTON_S2     loc="p101";
net BUTTON_S4     loc="p109";
```

Push Button Source Code

opb_bram.vhd

```
------------------------------------------------------------------------
--------
--
-- Simple OPB peripheral: a BRAM controller
--
-- Stephen A. Edwards
-- sedwards@cs.columbia.edu
--
------------------------------------------------------------------------
--------
library ieee;
use ieee.std_logic_1164.all;

entity opb_bram is

  generic (
    C_OPB_AWIDTH : integer                   := 32;
    C_OPB_DWIDTH : integer                   := 32;
    C_BASEADDR   : std_logic_vector(0 to 31) := X"01800000";
    C_HIGHADDR   : std_logic_vector(0 to 31) := X"01800FFF");

  port (
    OPB_Clk     : in  std_logic;
    OPB_Rst     : in  std_logic;
    OPB_ABus    : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
    OPB_BE      : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    OPB_DBus    : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
    OPB_RNW     : in  std_logic;
    OPB_select  : in  std_logic;
    OPB_seqAddr : in  std_logic;        -- Sequential Address
    Sln_DBus    : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Sln_errAck  : out std_logic;        -- (unused)
    Sln_retry   : out std_logic;        -- (unused)
    Sln_toutSup : out std_logic;        -- Timeout suppress
    Sln_xferAck : out std_logic;        -- Transfer acknowledge

    PB_A        : out std_logic_vector(8 to 9));

end opb_bram;

architecture Behavioral of opb_bram is

  constant RAM_AWIDTH : integer := 2;  -- Number of address lines on
the RAM
  constant RAM_DWIDTH : integer := 1;  -- Number of data lines on the
RAM

  component OBUF_F_24
    port (
      O : out std_ulogic;
      I : in  std_ulogic);
  end component;
```

```
  signal RNW : std_logic;
  signal RAM_DI, RAM_DO : std_logic_vector(0 to RAM_DWIDTH-1);
  signal ABus : std_logic_vector(0 to RAM_AWIDTH-1);
  signal chip_select : std_logic;
  signal output_enable : std_logic;
  signal WE, RST, PI : std_logic;

  -- Critical: Sln_xferAck is generated directly from state bit 0!
  constant STATE_BITS : integer := 3;
  constant Idle     : std_logic_vector(0 to STATE_BITS-1) := "000";
  constant Selected : std_logic_vector(0 to STATE_BITS-1) := "001";
  constant Read     : std_logic_vector(0 to STATE_BITS-1) := "011";
  constant Xfer     : std_logic_vector(0 to STATE_BITS-1) := "111";

  signal present_state, next_state : std_logic_vector(0 to STATE_BITS-
1);

begin

  pinpus: for i in 8 to 9 generate
    pinpad : OBUF_F_24 port map (
      O => PB_A(i),
      I => ABus(i-8));
  end generate;

  register_opb_inputs: process (OPB_Clk, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      RAM_DI <= (others => '0');
      ABus <= (others => '0');
      RNW <= '0';
    elsif OPB_Clk'event and OPB_Clk = '1' then
      RAM_DI <= OPB_DBus(0 to RAM_DWIDTH-1);
      ABus <= OPB_ABus(C_OPB_AWIDTH-3-(RAM_AWIDTH-1) to C_OPB_AWIDTH-
3);
      RNW <= OPB_RNW;
    end if;
  end process register_opb_inputs;

  register_opb_outputs: process (OPB_Clk, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      Sln_DBus(0 to RAM_DWIDTH-1) <= (others => '0');
    elsif OPB_Clk'event and OPB_Clk = '1' then
     if output_enable = '1' then
        Sln_DBus(0 to RAM_DWIDTH-1) <= RAM_DO;
     else
       Sln_DBus(0 to RAM_DWIDTH-1) <= (others => '0');
     end if;
    end if;
  end process register_opb_outputs;

  -- Unused outputs
  Sln_errAck  <= '0';
  Sln_retry   <= '0';
  Sln_toutSup <= '0';
```

```
  Sln_DBus(RAM_DWIDTH to C_OPB_DWIDTH-1) <= (others => '0');

  chip_select <=
    '1' when OPB_select = '1' and
        OPB_ABus(0 to C_OPB_AWIDTH-3-RAM_AWIDTH) =
        C_BASEADDR(0 to C_OPB_AWIDTH-3-RAM_AWIDTH) else
    '0';

  -- Sequential part of the FSM
  fsm_seq : process(OPB_Clk, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      present_state <= Idle;
    elsif OPB_Clk'event and OPB_Clk = '1' then
      present_state <= next_state;
    end if;
  end process fsm_seq;

  -- Combinational part of the FSM
  fsm_comb : process(OPB_Rst, present_state, chip_select, OPB_Select,
RNW)
  begin
    RST <= '1';                          -- Default values
    -- WE <= '0';
    output_enable <= '0';
    if OPB_RST = '1' then
      next_state <= Idle;
    else
      case present_state is
        when Idle =>
          if chip_select = '1' then
            next_state <= Selected;
          else
            next_state <= Idle;
          end if;

        when Selected =>
          if OPB_Select = '1' then
            if RNW='1' then
              next_state <= Xfer;
            end if
          else
            next_state <= Idle;
          endd if;

        -- State encoding is critical here: xfer must only be true here
        when Xfer =>
          next_state <= Idle;

        when others =>
          next_state <= Idle;
      end case;
    end if;
  end process fsm_comb;

  Sln_xferAck <= present_state(0);
```

```
end Behavioral;

-- Local Variables:
-- compile-command: "ghdl -a opb_bram.vhd"
-- End:
```

system.mhs

```
# Parameters
PARAMETER VERSION = 2.1.0

# Global Ports

PORT FPGA_CLK1 = FPGA_CLK1, DIR = IN
PORT RS232_TD = RS232_TD, DIR=OUT
PORT RS232_RD = RS232_RD, DIR=IN
PORT PB_A = PB_A, DIR=OUT, VEC = [8:9]

# Hint: Put your peripheral first in this file so it will be analyzed
# first and will generate errors faster.

# BRAM example peripheral

BEGIN opb_bram
 PARAMETER INSTANCE = bram_peripheral
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x01800000
 PARAMETER C_HIGHADDR = 0x01800FFF
 PORT OPB_Clk = sys_clk
 BUS_INTERFACE SOPB = myopb_bus
 PORT PB_A = PB_A
END

# The main processor core

BEGIN microblaze
 PARAMETER INSTANCE = mymicroblaze
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_ICACHE = 0
 PORT Clk = sys_clk
 PORT Reset = fpga_reset
 BUS_INTERFACE DLMB = d_lmb
 BUS_INTERFACE ILMB = i_lmb
 BUS_INTERFACE DOPB = myopb_bus
 BUS_INTERFACE IOPB = myopb_bus
END

# Block RAM for code and data is connected through two LMB busses
# to the Microblaze, which has two ports on it for just this reason.

# Data LMB bus

BEGIN lmb_v10
 PARAMETER INSTANCE = d_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_data_controller
```

```
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00000FFF
 BUS_INTERFACE SLMB = d_lmb
 BUS_INTERFACE BRAM_PORT = conn_0
END

# Instruction LMB bus

BEGIN lmb_v10
 PARAMETER INSTANCE = i_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_instruction_controller
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00000FFF
 BUS_INTERFACE SLMB = i_lmb
 BUS_INTERFACE BRAM_PORT = conn_1
END

# The actual block memory

BEGIN bram_block
 PARAMETER INSTANCE = bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = conn_0
 BUS_INTERFACE PORTB = conn_1
END

# Clock divider to make the whole thing run

BEGIN clkgen
 PARAMETER INSTANCE = clkgen_0
 PARAMETER HW_VER = 1.00.a
 PORT FPGA_CLK1 = FPGA_CLK1
 PORT sys_clk = sys_clk
 PORT pixel_clock = pixel_clock
 PORT fpga_reset = fpga_reset
END

# The OPB bus controller connected to the Microblaze
# All peripherals are connected to this

BEGIN opb_v20
 PARAMETER INSTANCE = myopb_bus
 PARAMETER HW_VER = 1.10.a
 PARAMETER C_DYNAM_PRIORITY = 0
 PARAMETER C_REG_GRANTS = 0
 PARAMETER C_PARK = 0
 PARAMETER C_PROC_INTRFCE = 0
 PARAMETER C_DEV_BLK_ID = 0
 PARAMETER C_DEV_MIR_ENABLE = 0
```

```
 PARAMETER C_BASEADDR = 0x0fff1000
 PARAMETER C_HIGHADDR = 0x0fff10ff
 PORT SYS_Rst = fpga_reset
 PORT OPB_Clk = sys_clk
END

# UART: Serial port hardware

BEGIN opb_uartlite
 PARAMETER INSTANCE = myuart
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_CLK_FREQ = 50_000_000
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_BASEADDR = 0xFEFF0100
 PARAMETER C_HIGHADDR = 0xFEFF01FF
 PORT OPB_Clk = sys_clk
 BUS_INTERFACE SOPB = myopb_bus
 PORT RX=RS232_RD
 PORT TX=RS232_TD
END
```

system.ucf

```
net sys_clk period = 18.000;

net FPGA_CLK1 loc="p77";

net RS232_TD loc="p71";
net RS232_RD loc="p73";

net PB_A<8> loc="p100";
net PB_A<9> loc="p101";
```

xparameters.h

```
/******************************************************************
*
* CAUTION: This file is automatically generated by libgen.
* Version: Xilinx EDK 6.2 EDK_Gm.11
* DO NOT EDIT.
*
* Copyright (c) 2003 Xilinx, Inc.  All rights reserved.
*
* Description: Driver parameters
*
******************************************************************/

#define STDIN_BASEADDRESS 0xFEFF0100
#define STDOUT_BASEADDRESS 0xFEFF0100

/******************************************************************/

#define XPAR_LMB_DATA_CONTROLLER_BASEADDR 0x00000000
#define XPAR_LMB_DATA_CONTROLLER_HIGHADDR 0x00000FFF
#define XPAR_LMB_INSTRUCTION_CONTROLLER_BASEADDR 0x00000000
#define XPAR_LMB_INSTRUCTION_CONTROLLER_HIGHADDR 0x00000FFF
#define XPAR_BRAM_PERIPHERAL_BASEADDR 0x01800000
#define XPAR_BRAM_PERIPHERAL_HIGHADDR 0x01800FFF

/******************************************************************/

#define XPAR_XUARTLITE_NUM_INSTANCES 1
#define XPAR_MYUART_BASEADDR 0xFEFF0100
#define XPAR_MYUART_HIGHADDR 0xFEFF01FF
#define XPAR_MYUART_DEVICE_ID 0
#define XPAR_MYUART_BAUDRATE 9600
#define XPAR_MYUART_USE_PARITY 0
#define XPAR_MYUART_ODD_PARITY 1
#define XPAR_MYUART_DATA_BITS 8

/******************************************************************/
```

XMMS Controller

test.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

int main (void) {
  int input;
  int runBefore = 0;
  char fileName[] = "/root/mp3/*.mp3";

  // Temporarily interface window
  do {
    char command[50] = "xmms "; // initialize
    printf("Select: (1) Play (2) Next (3) Stop (4) Quit\n");
    scanf("%d", &input);
    if (input == 1)
      strcat(command, "-p ");
    if (input == 2)
      strcat(command, "-f ");
    if (input == 3)
      strcat(command, "-s ");
    if (input == 4);
    else {
      if (runBefore == 0) {
        strcat(command, fileName);
        strcat(command, " &");
        runBefore = 1;
      }
      system(command);
    }
  }
  while (input != 4);
}
```