Sherman Lam
E155
November 2, 2014

Lab 7 Report: USB, SPI, and VGA

1 Introduction

In this lab, I used both the PIC and FPGA on the μMudd32 to create a mouse-to-monitor interface. The PIC interfaced with the mouse through USB. The FPGA interfaced with the monitor through VGA. The PIC and FPGA communicated through SPI.

2 Design and Testing Methodology

1. Using spi2
2. sclk - P99, sdo - P75, sdi - 76
3. vga already has pll setup
4. Pin assignments based on μMudd32 pinout.

   `sync_b`

2.1 Mouse Tracking

All mouse tracking functionality was performed on the PIC. The provided code reports the change in position of the mouse. I modified it to track the absolute x,y coordinates of the cursor and placed limits on the position to keep the cursor in the visible screen area (640x480). Each time the mouse is moved and the cursor position is updated, the new position is sent over SPI to the FPGA.

To make data transfer easy, the x and y coordinates are sent at the same time. The max value each coordinate can have is 640. This can be represented in 10 or more bits. So, each coordinate is stored as a short (16-bits). A data packet consists of 32 bits. The bottom 16 bits are used to store the y coordinate while the top 16 bits are used to store the x coordinate. The data packet is then sent over SPI.

2.2 SPI

SPI is a serial data communication protocol and is commonly used to interface two digital systems. In each interface, there is a master - the device that performs the main job of managing the line of communication. The slave receives and send data when signaled by the the master.

In this lab, I used the PIC as the master since it has built in SPI peripherals and used the FPGA as the slave. The code I used was mostly based on SPI example code found in the class lectures. I used SPI2, which has the pinout from Table 1.

SPI Pin Mapping			
Signal	sclk	sdo	sdi
Pin	P99	P75	P76

Table 1: SPI 2 Pin Mapping

sclk is generated by the master and controls the timing of data transfer. **sdo** is the serial output pin for the master. That is, it transfers data from the master, to the slave. Finally, **sdi** is the serial input pin for the master. That is, it transfers data from the slave to the master.

On the PIC side, the baud rate was set to 1.25MHz. To send data through SPI, I just write the data (a 32-bit integer) to the SPI 2 buffer. Note that in order to send a 32-bit number, the MODE32 in the SPI 2 control register must be set to 1.

On the FPGA side, I used a simple shift register to receive data. Since the PIC doesn't need any information from the FPGA, the FPGA sends back 0s.

2.3 VGA and displays on the FPGA

All screen display functionality was handled by the FPGA. When a complete data packet is received through SPI, the data is parsed to obtain the x and y position of the cursor. Remember that the top 16 bits are used to store the x coordinate and the bottom 16 bits are used to store the y coordinate.

Two sets of RGB values are generated for the screen - one for the background and one for the cursor. The background image can be any pattern or color (see Section 2.4 to see how I defined a color gradient). The cursor image is black (RGB = 0,0,0) and white (RGB = 255,255,255). White corresponds to the image of the cursor while everything else is black. A final image with the cursor overlaid on top of the background can be obtained by performing a bitwise OR between the background and cursor images.

2.4 Color Gradient Generation

The background I used consists of two gradients. The x axis features a gradient with increasing red content. The y axis features a gradient with increasing green content. The blue content is left 0 on the whole image.

The RGB values can range from 0 to 255. So, the gradient that maximizes this range would be defined by:

$$r = \frac{x}{640}255$$
$$g = \frac{y}{480}255$$

However, it is difficult to implement division and multiplication in hardware. So, an alternative that provides nice results but doesn't fully utilize the range is:

$$r = x >> 2$$
$$g = y >> 2$$

Figure TODO shows the result of the gradient.

2.5 Cursor Generation

The cursor is just a 45-45-90 triangle with a short stem that in total forms an arrow. For a small 8x8 pixel area, the head is defined by x,y coordinates that follow the following constraint.

$$x + y <= 8$$

The stem of the arrow is defined by the following constraint.

$$x = y$$

However, when implemented on an area larger than the 8x8 pixel box, the constraints change slightly. Let *xpt* and *ypt* be the cursor position. For

$$dx = x - xpt$$
$$dy = y - ypt$$

The pixels in the arrow (head and stem) must follow these constraints:

$$0 <= dx <= 8$$
$$0 <= dy <= 8$$
$$dx + dy <= 8$$
$$dx = dy$$

3 Technical Documentation

3.0.1 FPGA and PIC Pin Layout

Figures 1 and 2 show parts of the μMudd32 schematic that provide the needed pinouts.

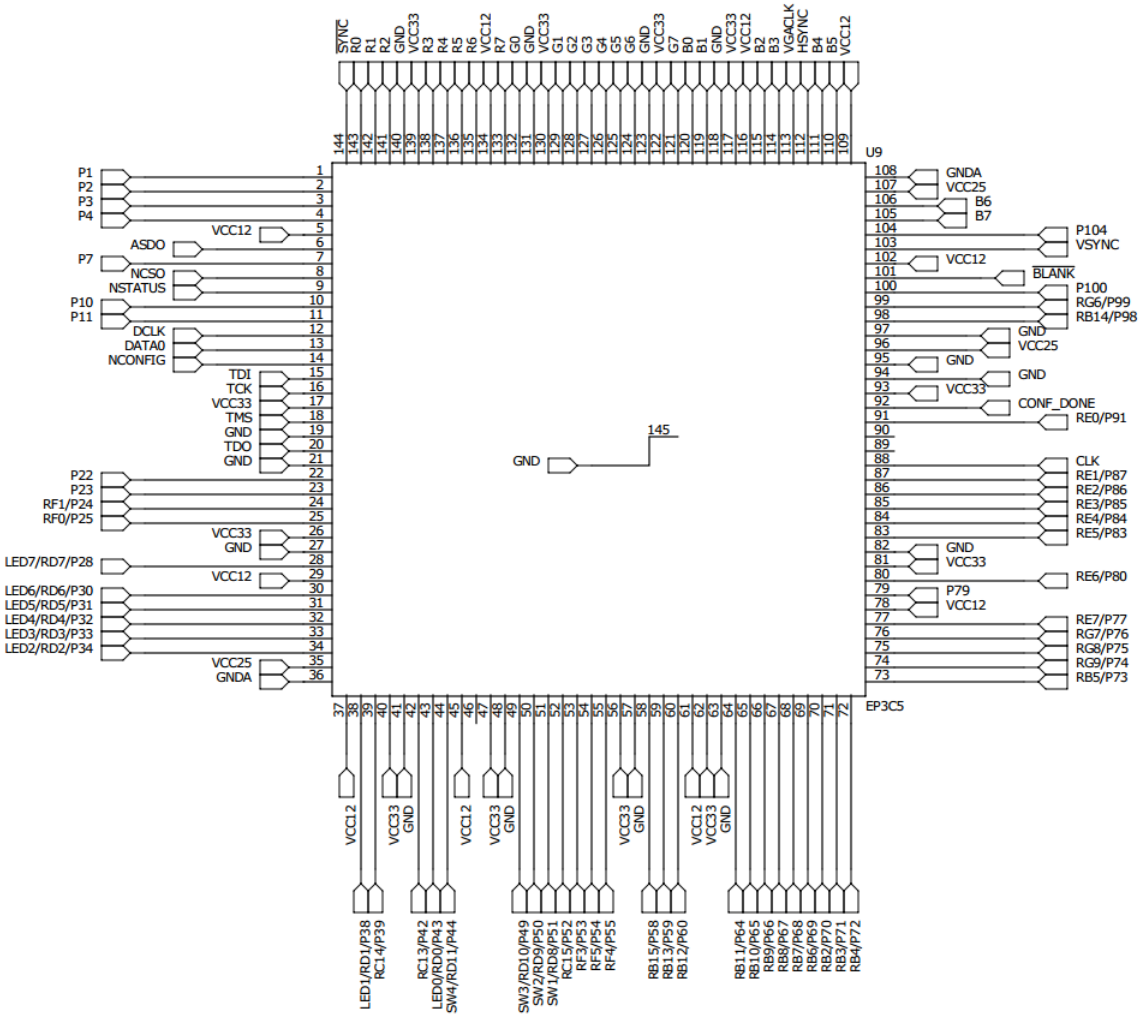

Figure 1: FPGA pin layout. Pins that start with R,B,or G are RGB pins that are connected to the DAC.

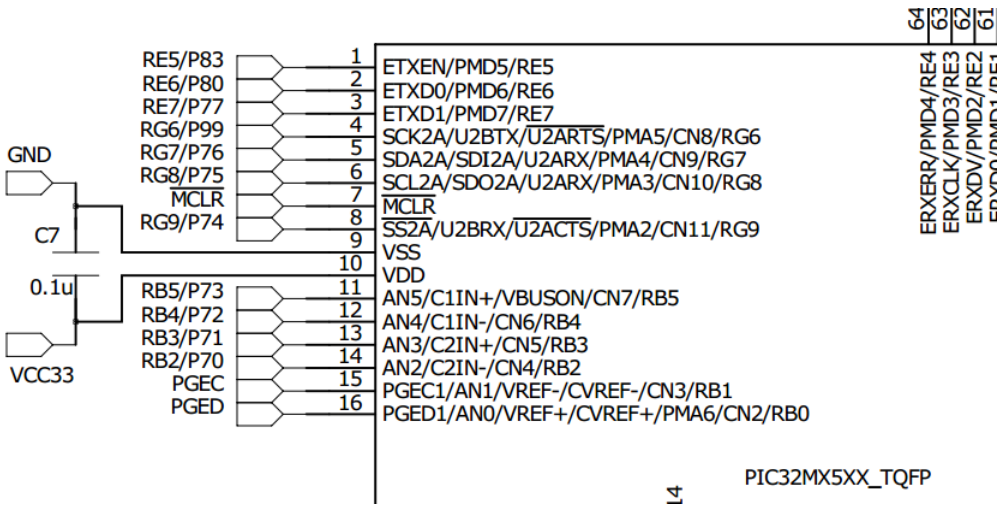

Figure 2: PIC pin layout. Relevant SPI 2 pins are SCK2A, SDI2A, and SDO2A.

3.1 FPGA Code

This section provides pieces of the code that I added to / modified from the provided FPGA VGA code.

3.1.1 lab7_SL.sv

```
1  module lab7_SL( input  logic        clk, reset,
2                  input  logic        sclk, sdo, sdi,         // spi communication lines
3                  output logic        vgaclk,                 // 25 MHz VGA clock
4                  output logic        hsync, vsync, sync_b,   // to monitor & DAC
5                  output logic [7:0]  r, g, b);               // to video DAC
6
7      logic [9:0] xpt;
8      logic [9:0] ypt;
9      logic ready;
10     logic [31:0] data;
11     //for displaying vga
12     vga vga1(.clk(clk),.xpt(xpt),.ypt(ypt),
13                 .vgaclk(vgaclk),.hsync(hsync),.vsync(vsync),
14                 .sync_b(sync_b),.r(r),.g(g),.b(b));
15     //for receiving spi data
16     spiRx spi(.sclk(sclk),.sdo(sdo),.reset(reset),.ready(ready),
17                       .sdi(sdi),.data(data));
18     //for parsing data
19     parse parser(.data(data),.reset(reset),.ready(ready),.xpt(xpt),.ypt(ypt));
20 endmodule
21
22
23 /*
24 Module for receiving 32 bit spi data
25
26 Author: Sherman Lam
27 Email: slam@g.hmc.edu
28 Date: 11-2-14
29 */
30 module spiRx(   input logic         sclk, sdo, reset,   // SPI data lines
31                 output logic        sdi,                // SPI data
32                 output logic        ready,              // whether the data packet is ready
33                 output logic [31:0] data);              // data packet
34     //counter for 32 bits
35     parameter bits = 5'd31;
36     logic [4:0] counter;
37
38     //read on rising edge of spi clock
39     always_ff@(posedge sclk) begin
40         if (reset)  counter = '0;
41         else            counter = counter + 1'b1;
42     end
43
44     //shift register
45     always_ff@(posedge sclk) begin
46         data = {data[30:0],sdo};
47     end
48
49     assign ready = (counter == 0);
50
51     //send back bogus data
```

```
52      always_ff@(negedge sclk)
53          sdi = 0;
54
55  endmodule
56
57
58  /*
59  Module for parsing spi data
60
61  Author: Sherman Lam
62  Email: slam@g.hmc.edu
63  Date: 11-2-14
64  */
65  module parse(   input logic [31:0] data,        // data packet
66                  input logic ready, reset,       // whether data is ready to be stored, reset
67                  output logic [9:0] xpt, ypt);   // x and y position of cursor
68
69      logic [31:0] dataCp;
70
71      //store 32 bit data
72      always_ff@(posedge ready, posedge reset) begin
73          if (reset)      dataCp = '0;
74          else            dataCp = data;
75      end
76
77      //mask to get x and y
78      always_comb begin
79          xpt = dataCp[25:16];
80          ypt = dataCp[9:0];
81      end
82  endmodule
```

3.1.2 vga.sv

4 Results and Discussion

The cursor works as expected. The point of the cursor is design to not leave the 640x480 screen. However, if the cursor is on the 640 and/or 480 boundary, the body and stem of the arrow are allowed to exceed the bounded space. Only the very tip cannot leave.

The pin SYNC_b in the provided VGA code corresponds with the pin $\overline{SYNC}$.

5 Conclusion

5.1 Time Spent

Programming 6 hrs

Writing Report 2hrs

Total Time Spent 8hrs

5.2 Suggestions for lab

Remove PLL section since it is in the provided VGA file. Or, put note that says "This is for your interest only". Provide pins for spi communications.