Wave Visualizer

Final Report

Advisor:
Professor Stephen Edwards

CSEE 4840 Embedded Systems
Columbia University
Spring 2019

Tvisha Gangwani (trg2128@columbia.edu)
Jino Masaaki Haro (jmh2289@columbia.edu)
Ishraq Khandaker (ibk2110@columbia.edu)
Klarizsa Padilla (ksp2127@columbia.edu)
Zhongtai Ren (zr2208@columbia.edu)

1. Introduction

1.1. Wave Visualizer

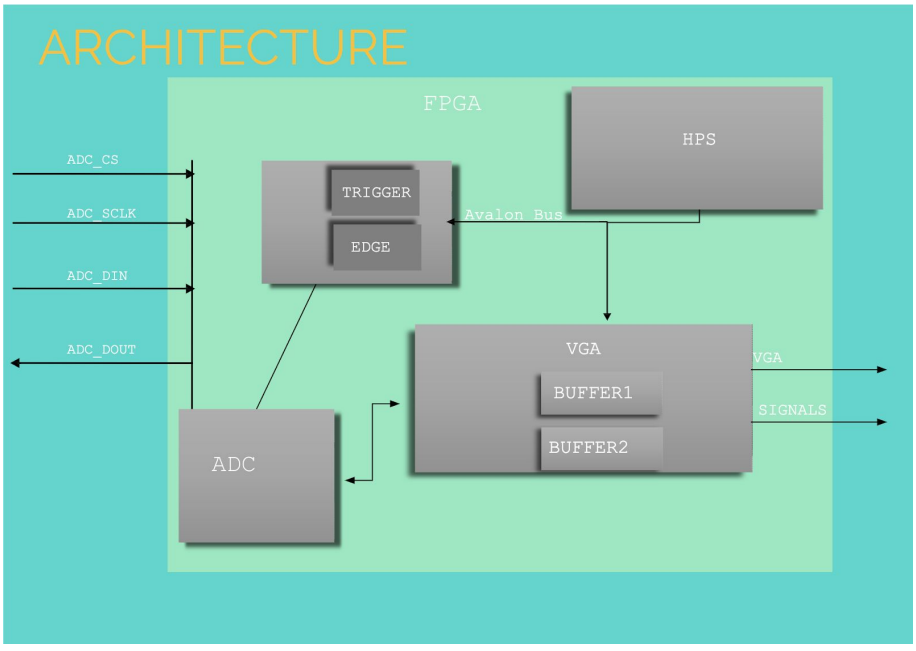
The group presents a “Wave Visualizer” with two control options. For this project, we are using the the ADC (LTC 2308) present on the FPGA to convert an analog input signal at channel 0 of the ADC. The signal is being sampled at 12.5 MHz to load a buffer of 640 values, which is refreshed every time the voltage is triggered at a user specified value. Furthermore, we are using the VGA display - similarly as in lab 3 - to display the waveform from the input signal. We are also using an USB mouse to allow the user to control the trigger value and edge by clicking on a “+” or “-” button on the screen. As a result, we obtain a waveform as supplied in the ADC input, triggered at the value and edge selected by the user.

1.2. Motivation

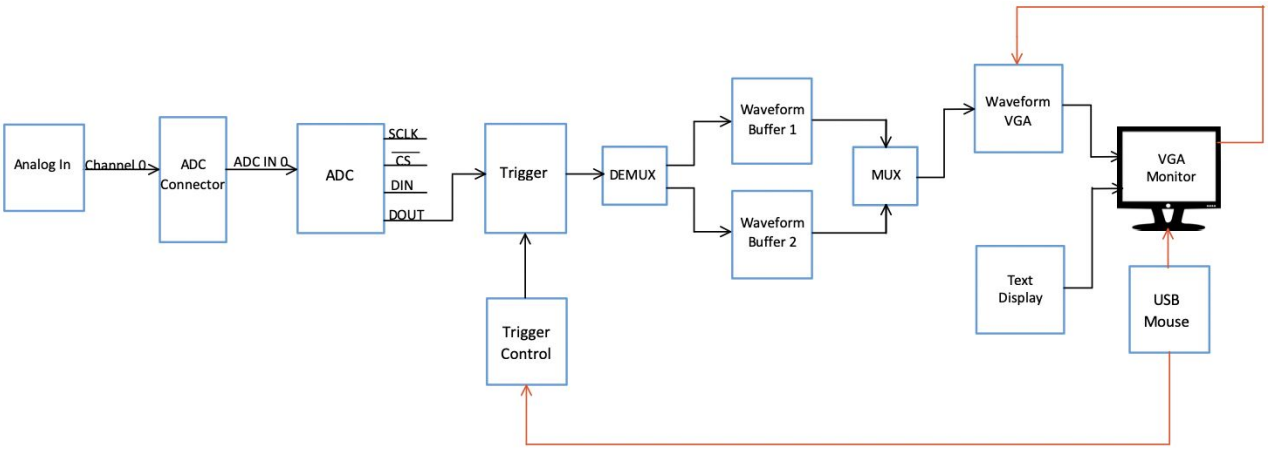
The first idea that came to mind, thinking of an FPGA, was to make a gaming project which uses sprites and movements similar to the lab 3 vga ball. But our group, consisting of two EE, two CE, and 1 CS student, wanted to combine our knowledge of software with that of hardware and work with signals. One of the most used devices in the EE/CE labs is the oscilloscope, hence we wanted to learn more about this highly utilized tool and how it works by constructing our own version using the FPGA and peripherals.

2. Architecture

2.1. Overview



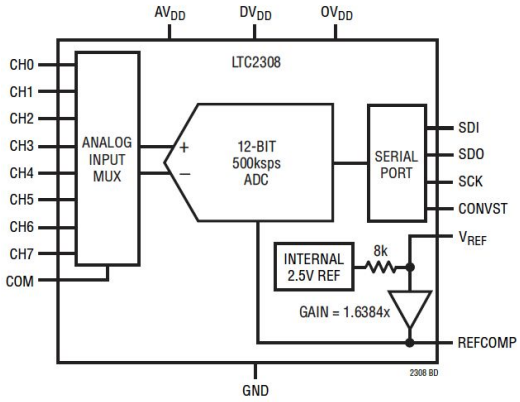
Architecture Image 1: System Architecture



Architecture Image 2: System Block Diagram

2.2. ADC

The ADC in this particular revision of the Altera DE1-SOC board is the LTC2308 chip. The LTC2308 is a low noise, 500 ksps, 12-bit ADC with a serial interface. The serial data output clock (ADC_SCLK) operates at any frequency up to 40 MHz, running from the 50 MHz internal conversion clock. The low power consumption and small size makes this ADC chip ideal for battery operated and portable applications. Furthermore, the 4-wire serial interface makes this ADC a good choice for isolated and remote data acquisition systems.



Architecture Image 3 : ADC Block Diagram

2.3. Trigger

Trigger is the most important and tricky part of our wave visualizer design. We use a trigger to detect the start point of the incoming digital signal value. Since the signal data transfer very fast, VGA screen will output all possible signals in a chaotic manner. In order to avoid this and always display the signal from a pinned point, we introduced trigger.

There are mainly two features of trigger: trigger_voltage and trigger_slope. We use them to track a specific point of the signal. The method of judging the slope is as followed: if the result of ADC data minus trigger_voltage change from negative to positive, the slope is positive, and vice versa. Then, we can find a specific point of the signal and display one single signal curve on the screen.

2.4. Memory Pipeline

The data come from ADC memory will firstly compare with trigger_voltage. If the ADC data meet the trigger_voltage and the slope is the same as what we set (either positive or negative), we will give ADC part an enable signal. Then FPGA will start transfer the data from ADC part to VGA part. Those data will be continuously stored in the frame buffer. Once the frame buffer is full, VGA part will send a stop signal. And FPGA will stop the transmission and wait for the next enable signal. The key was to use 2 buffers that, take turns storing input and displaying input on the screen. This allows us to have a steady wave on the screen that does not flicker.

2.5. Framebuffers

DE1-SoC linux frame buffer is used for the signal display. There are two framebuffers used to storage and display signal. Each time we use one of the them to update the ADC value. And we use another one to display the signal. We use a enable signal and an disable signal to start and stop the process of storagging data from ADC to framebuffers. To be more precise, we have a boolean value called “first” that when toggled with will decide which buffer is to be displayed and which one is taking in input at a given point in time.

2.6. Shift Registers

For this project, we decided to use two shift registers for communicating data to and from the on-board ADC. In addition to three other signals, ADC takes as serial input a 6-bit value, ADC_DIN and serially outputs a 12-bit value of the signal sample, ADC_DOUT. For the scope of this project, we only worked with the signal sent to ADC channel 0. As a result, the shift register for ADC_DIN is always loaded with the same 6-bit value to be shifted on the positive edge of the ADC clock, ADC_SCLK and when the ADC chip-select, ADC_CS_N is low. After all the bits have be shifted, and hence, serially sent to the ADC, this shift register is updated to have the same 6-bit value (6’b 100010) that was initially loaded. The function used to send ADC_DIN to the ADC and the usage of the shift register is shown below:

```
//controls the D_in signal
module toADC (input logic mclk, input logic ds, input logic cs, output logic din);
//make a shift register to send data to ADC

    logic [5:0] shiftreg = 6'b 100010; //initialize shift reg to 0s
    logic [5:0] counter = 6'd 0;

    always_ff @ ( posedge mclk )
    begin

        if (!cs && ds && counter < 6'd 6 )
        begin
            din <= shiftreg[5];
            shiftreg [5:1] <= shiftreg[4:0];
            shiftreg [0] <= 1'd 0;
            counter <= counter + 6'd 1;
        end

        else if(counter == 6'd 6 && !ds && cs)
        begin
            din <= din;
            shiftreg <= 6'b 100010;
            counter <= 6'd 0;
        end

        else
            din <= din;

    end

endmodule
```

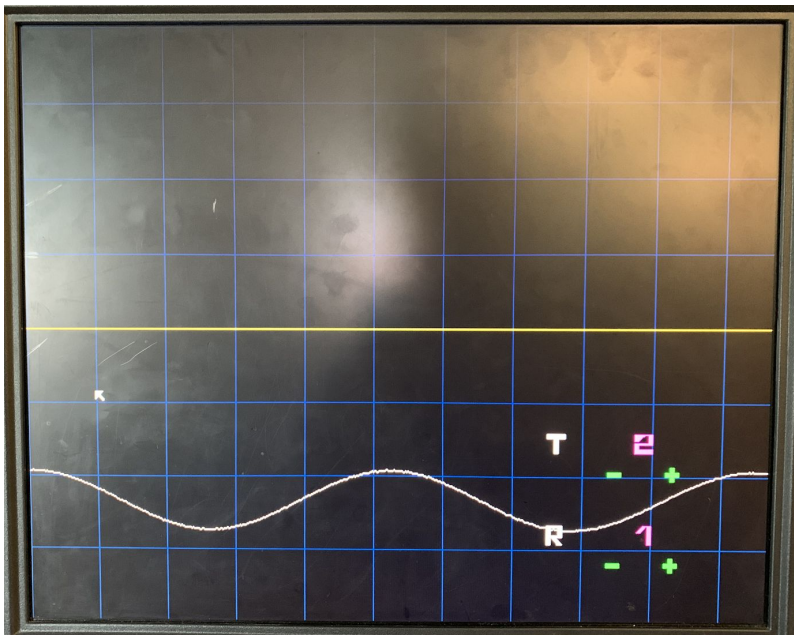
Architecture Image 4: Module for sending serial data to the ADC, using shift register

We used a similar idea for serially accepting data from the ADC into a shift register, which is then stored in a separate register every 12 cycles of the ADC_SCLK. This shift register is initialized to have a 12 bit value of all zeros. When the ADC_CS_N is low, a bit is shifted every ADC_SCLK cycle, starting with the most significant bit of ADC_DOUT. When the ADC_CS_N goes high after the 12 cycles, this shift register is copied into another register, ADC_REG, which is then passed on to the VGA buffers every time the value is updated, to then be displayed on the VGA monitor. The function used to receive ADC_DOUT serially from the ADC and the usage of the shift register is shown below:

```
//controls the D_out signal
module fromADC (mclk, ds, cs, dout, ready, out);

    input logic mclk, ds, cs, dout;
    output logic [11:0] out;
    //output logic [11:0] ADC_REG_PREV;
    output logic ready;
    logic [5:0] counter = 6'd 0;

    logic [11:0] shiftreg = 12'd 0;

    logic load_data = 1'd 1; //check this if we have issues displaying

    always_ff @ ( posedge mclk )
    begin

        if (!cs && ds && counter < 6'd 1)
        begin
            //ADC_REG_PREV[11:0] <= shiftreg[11:0];
            counter <= counter + 6'd 1;
        end

        else if (!cs && ds && counter >= 6'd 1 && counter <= 6'd 12)
        //if (!cs && ds && counter <= 6'd 11)
        begin
            shiftreg = {shiftreg[10:0], dout};
            counter <= counter + 6'd 1;
            load_data <= 1'd 1;
            ready <= 1'd 0;
            //ADC_REG_PREV[11:0] <= ADC_REG_PREV[11:0];
        end

        else if(cs && !ds && load_data)
        begin
            //counter = 6'd 0;
            //out <= shiftreg;
            out[11:0] <= shiftreg[11:0];
            counter = 6'd 0;
            load_data <= 1'd 0;
            ready <= 1'd 1;
        end

    end

endmodule
```

Architecture Image 5: Module for receiving serial data from the ADC, using shift register

2.7. VGA



Architecture Image 6: Final image of Display

The VGA display was divided into 3 sections- the display from the hardware, software and the waveform. The background was made a black color with the graph hard coded in with lines drawn every 120 pixels in dark blue. There is a middle line to indicate the center of the screen in yellow. The waveform is represented in white and displays the values coming in from the ADC. In order to display the waveform we had to create two memory buffers. This is so that we display one buffer, while the other one is loading from the ADC. The memory buffer can hold 640 values each 16 bits long. When the wave hits a trigger value the ADC starts sending "valid" data that has to be captured by one of the buffers. When the buffer is "full" its changes the boolean variable to full and holds it until it receives a new "valid" signal. At this point it will switch to the second buffer and start storing values in the second buffer. While the second buffer is loading we start to display the first buffer on the screen (aka the wave) in white. The code for the memory buffer can be seen in the screenshot below.

```
// 16 X 8 synchronous RAM with old data read-during-write behavior
module memory(input logic        clk,
          input logic [9:0]  a,
          input logic [15:0]  din,
          input logic    we,
          output logic [15:0] dout);

//we have 1280 pixels, so 1280 digits coming in each of 16 bits

   logic [15:0]              mem [639:0];
   integer j;
   integer flag;
   initial begin

   for(j = 0; j < 639; j = j+1)

        mem[j] = 16'd180;
    //end

end

   always_ff @(posedge clk) begin
      if (we) mem[a] <= din;
      dout <= mem[a];
   end

endmodule
```

Architecture Image 7: Code for Memory buffers.

The last part of the display consists of the mouse and the buttons. Both of which were supposed to initially come in from the software. We were able to control the mouse using ioctl commands that came in from the software and edited the "display of the mouse which we had hard coded onto the screen. We we hardcoded the structure of the mouse into the hardware as shown below.

```
logic [7:0]     mse[7:0];

initial begin
mse[0] = 8'b01111111;
mse[1] = 8'b00111111;
mse[2] = 8'b00011111;
mse[3] = 8'b00011111;
mse[4] = 8'b00111111;
mse[5] = 8'b01110011;
mse[6] = 8'b11100001;
mse[7] = 8'b11000000;
end

always_ff @(posedge clk) begin
shape <= mse[a_m];
end
endmodule
```

Architecture Image 8: Hardcoded structure of mouse in hardware.

Due to time restrictions, we were unable to bring in the various buttons from the software as previously decided. Thus, we had to make the last minute decision of hardcoding the buttons into the

hardware as well. Due to memory restrictions, and so as to not make the display too cluttered, we decided to just draw two letters on the screen. A “T” to represent the trigger value and an “R” to represent whether the trigger was on a rising or falling edge. We also hard coded in a “+” to increment the values and a “-” to decrement the values. The trigger value goes from 0 - 3 and the rising value goes from 0-1.We took in values from the software that represent the clicks of the mouse on the various buttons. In order for this to work we had to edit the ioctl functions we made in lab3. We had to add in two more values, one for the trigger and one for the rising or falling edge of the wave.

2.8. Mouse

There are three steps to set up the USB mouse to be available for the FPGA and use the information we get to realize the user interface.

The first step is to connect FPGA USB port with USB-mouse. We use HID protocol to communicate between FPGA and mouse. The details of using HID can be found here: http://libusb.org. In the file “device class definition for human interface devices (HID)” (hid1_11.pdf, https://usb.org/sites/default/files/documents/hid1_11.pdf), page 9, we will find all the protocol codes for USB devices. And the protocol code for mouse is 2. Set the protocol code to 2 and then FPGA will try to find a connection with USB-mouse device.

Protocol Codes

Protocol Code	Description
0	None
1	Keyboard
2	Mouse
3 - 255	Reserved

Architecture Image 9 : USB mouse protocol codes

The next step is to get data from USB-mouse. Reading the datasheet, we can know that there are 3 sections of data coming into the USB port. Byte 0 to 2 are Button status, X movement and Y movement, respectively. I built a structure to storage those three bytes.

```
struct usb_mouse_packet {
  uint8_t modifiers;
  uint8_t pos_x;
  uint8_t pos_y;
};

/* Find and open a USB keyboard device.  Argument should point to
   space to store an endpoint address.  Returns NULL if no keyboard
   device was found. */
extern struct libusb_device_handle *openmouse(uint8_t *);
```

Architecture Image 10 : USB mouse Data load

Offset	Size	Description
0	Byte	Button status.
1	Byte	X movement.
2	Byte	Y movement.

Meaning of incoming data

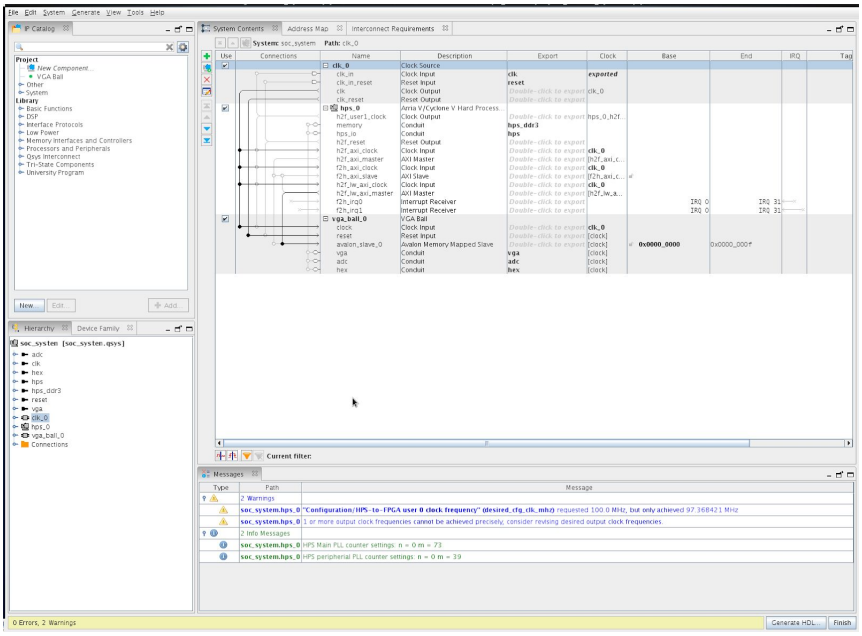
(Source: https://wiki.osdev.org/USB_Human_Interface_Devices)

For the third step, we implement the USB mouse data into two distinct purposes. (1) Continuously sending the mouse position to hardware so that we can display an arrow on the VGA screen. (2) Interface user's click with the parameter changing.

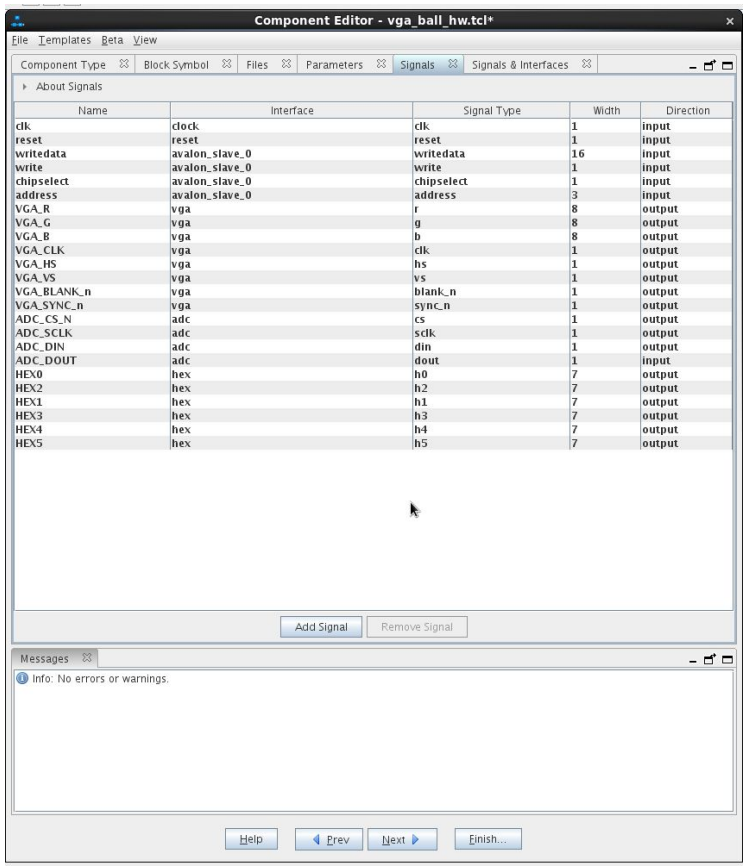
Since we need to set up the trigger's slope and voltage, we put different buttons for different operations on VGA screen. Each time we click the mouse, FPGA will decide which button has been clicked based on the x,y position of mouse. Then, FPGA can change the corresponding parameter value.

3. Hardware Design

For the hardware design of this project, we used the VGA Ball component from lab 3 and created the interfaces necessary for running the project. We created an adc interface, and a hex interface in addition to using the vga_ball interface from lab 3 with necessary edits. The QSYS connections as well as the signals are shown in the figures below.



Hardware Design Image 1: QSYS component and interfaces



Hardware Design Image 2: Component signals view

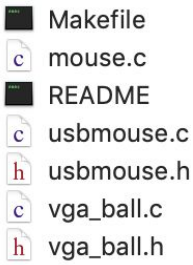
4. Software Overview

We have built a user interface (UI) for the wave visualizer that allows trigger control/interaction. The main idea is that we can control the mouse to click the different buttons on VGA screen in order to that change the value of two modifiable parameters.

We create the functions as follows:

(1) Open and connect to the USB mouse.
(2) Read the mouse position with the read_mouse.c and read_mouse.h files.
(3) Create the structure to hold and update the parameters' value (x,y position, trigger voltage, slope, etc.) then pass the value from software to hardware.
(4) Use mouse click status and mouse position to judge which button (displayed on the VGA screen) has been clicked so that the corresponding parameter value - such as trigger_value, trigger_slope, etc. - can be changed.

Function (1) and (2) are realized in the usbmouse.c and usbmouse.h files. Function (3) is realized in the vga_ball.c and vga_ball.h files. Function (4) is realized in the mouse.c file.



Software Image 1 : USB mouse c codes

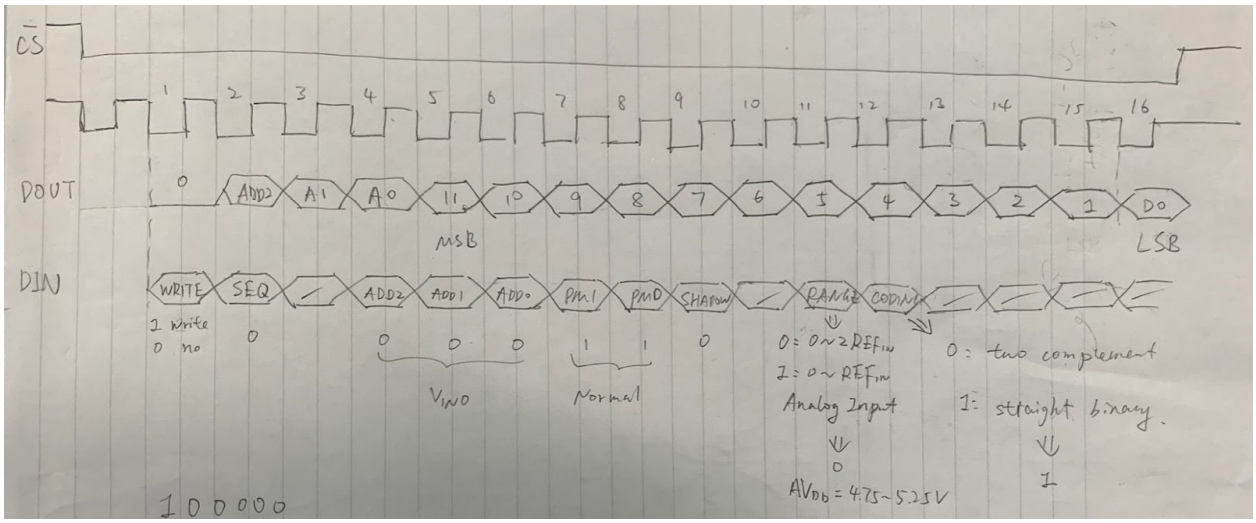
5. Memory Access and Timing

5.1. Model Sim and Test Bench

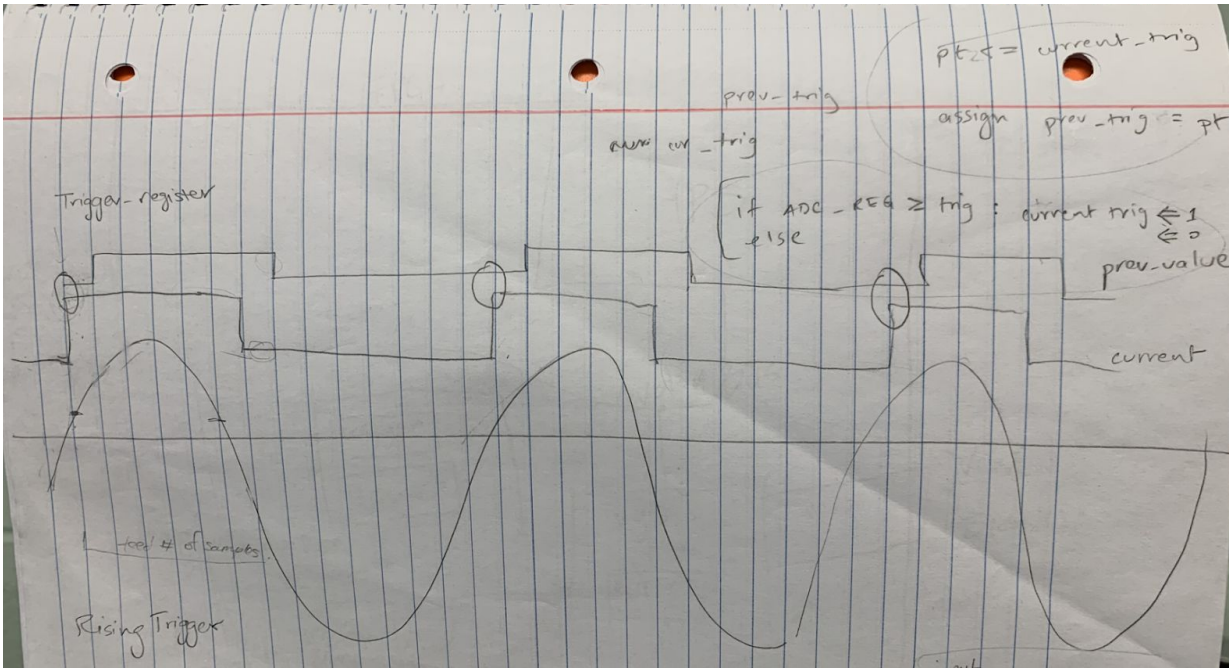
Model Sim and test benches were heavily used in the construction of this project as timing and the sequenization of data handoff are particularly crucial for live signal. In the test bench we created signals so that we could follow various variables that we had to make sure they were being synchronised as anticipated. This was quite helpful for debugging purposes.

5.2. “Magic” Timing Diagram Paper

The format of the paper in college ruled notebook is with multiple lines. We can rotate it 90 degrees and get a fixed interval paper. Then we can use it to draw time diagram and signal.



Memory Access and Timing Image 1: time diagram for ADC part

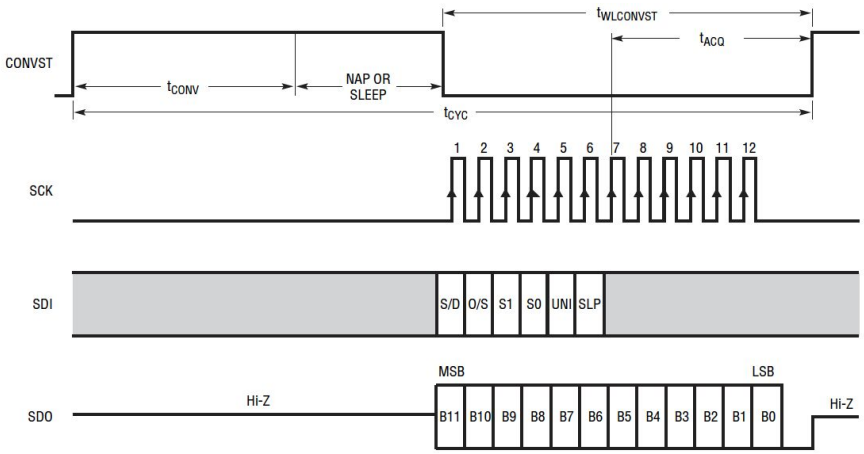


Memory Access and Timing Image 2: Trigger logic

5.3. ADC Timing Characteristics

SYMBOL	PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
$f_{SMPL(MAX)}$	Maximum Sampling Frequency		●			500	kHz
f_{SCK}	Shift Clock Frequency		●			40	MHz
t_{WHCONV}	CONVST High Time	(Note 9)	●	20			ns
t_{HD}	Hold Time SDI After SCK↑		●	2.5			ns
t_{SUDI}	Setup Time SDI Valid Before SCK↑		●	0			ns
t_{WHCLK}	SCK High Time	$f_{SCK} = f_{SCK(MAX)}$	●	10			ns
t_{WLCLK}	SCK Low Time	$f_{SCK} = f_{SCK(MAX)}$	●	10			ns
$t_{WLCONVST}$	CONVST Low Time During Data Transfer	(Note 9)	●	410			ns
$t_{HCONVST}$	Hold Time CONVST Low After Last SCK↓	(Note 9)	●	20			ns
t_{CONV}	Conversion Time		●		1.3	1.6	µs
t_{ACQ}	Acquisition Time	7th SCK↑ to CONVST↑ (Note 9)	●	240			ns
$t_{REFWAKE}$	REFCOMP Wakeup Time (Note 12)	$C_{REFCOMP}$ = 10µF, C_{REF} = 2.2µF			200		ms
t_{dDO}	SDO Data Valid After SCK↓	C_L = 25pF (Note 9)	●		10.8	12.5	ns
t_{hDO}	SDO Hold Time After SCK↓	C_L = 25pF	●	4			ns
t_{en}	SDO Valid After CONVST↓	C_L = 25pF	●		11	15	ns
t_{dis}	Bus Relinquish Time	C_L = 25pF	●		11	15	ns
t_r	SDO Rise Time	C_L = 25pF			4		ns
t_f	SDO Fall Time	C_L = 25pF			4		ns
t_{CYC}	Total Cycle Time				2		µs

Memory Access and Timing Image 2 : ADC Timing Characteristics



Memory Access and Timing Image 3 : ADC Timing Diagram

Please note that the FPGA user manual refers to the CONVST signal as the ADC_CS_N signal, the SCK signal as the ADC_SCLK, the SDI as the ADC_DIN, and the SDO signal as ADC_DOUT.

6. Project Plan

6.1. Lessons Learned

The labs are really useful for our project. We modified Lab 1 to display the digital value coming from ADC on the 7 segment hex display. We use lab 2 materials to interface the HID USB mouse to control the trigger values and edge. We use lab 3 lessons to link hardware and software and display the converted ADC signal data on the VGA screen.

6.2. Timeline

- Separate tasks into 3 parts: ADC data load; VGA display; mouse control.
- Block diagram and Simulation
  Samples into the buffers logic: experiment with Model sim that comes with quartus for the simulations.

- Interfaces and communication:
  Load ADC data stream: get the digital data stream from analog signal generator.
  Waveform buffer 640 samples, and 9 bit each sample.

- Storage and display on the VGA screen:
  Trigger value (rising edge) then start loading data
  Have it display the waveform
  Horizontal sweep (need a trigger event)
  Vertical (just math, keeping track)

- Figure out what the user interface will be:
  Mouse to control buttons virtually

Decide if information will be on vga's

What will the screen look like (text and other information on the screen?)

Make two waveform buffers and be displaying one while we are loading the other one

Grid stuff for text-- like pacman (use of tiles)

7. Debugging

7.1. Mouse

Although the values of the mouse came in from the software, the mouse was displayed using the hardware. This gave rise to a small issue where we were only able to display the mouse on one part of the screen. It could not move past the 320'th pixel horizontally. This, however, was an easy fix and was made to work by ignoring the least significant bit of hcount while drawing it up on the screen. Another interesting thing to note is that the initial tile for the arrow, as well as every tile in the second software iteration (described in detail in section 10), was coded in expected direct order. This caused the images to appear flipped when initially displayed on the screen. For the final software iteration, all tiles were coded in a "mirror" manner, which allowed a direct accurate display without the need of a reordering function.

7.2. Buffers

We had to create two buffers, as previously mentioned, to display the waveform. We had some issues while switching between the the two buffers and displaying the wave. This lead to an unwanted line blinking at the top of the screen. This was solved by matching up the values to the correct inputs and outputs to the two buffers to produce the correct form of the wave on the screen. We also had an issue with the index to which the pixel values were being saved. Thus, initially we had the wave moving down even though the voltage was increasing and moving up when it was increasing. But, we were able to find a solution for this by subtracting the value of the indexes from 480.

The output for the mouse and the rising and trigger values were coming in from the software. We initially had some problems with how these values were created. It took us sometime to find the bug in the code. But, we were successfully able to fix the issue.

We tried to rename all the components of the code to "vga" instead of "vga_ball" and also to break up the code into three files. A top file that governed the other two files - adc and vga. However, on attempting to do this not only did the ADC values no longer work but we also lost the vga driver that made the link between the hardware and the software. After several hours of trying to figure out where the issues were, we decided to just merge the files into one and this is how we were able to get all of the components working together.

7.3. Integration

Initially, we planned to design our system integration such that there would be three separate SystemVerilog files: one SV file for the ADC modules and data, one SV file for the VGA modules and display, and one top module that would combine these two files to run the entire oscilloscope project. However, after setting up per plan, while testing the functionality of this component, we encountered errors and the programs did not run. Furthermore, we were unable to combine the hardware and software pieces using the kernel. We anticipate that using a clock for each SV piece could have contributed to timing or synchronization errors. In addition, while setting up the project, we did not consider the kernel piece for hardware-software communication. As a result, when we tested to take the mouse input and reflect on the hardware, we were unable to accept the ioctl() calls and send the information. In order to

overcome the integration challenges, we decided to use the lab 3 hardware modules and files to implement the VGA, and then added the ADC pieces to integrate the two components.

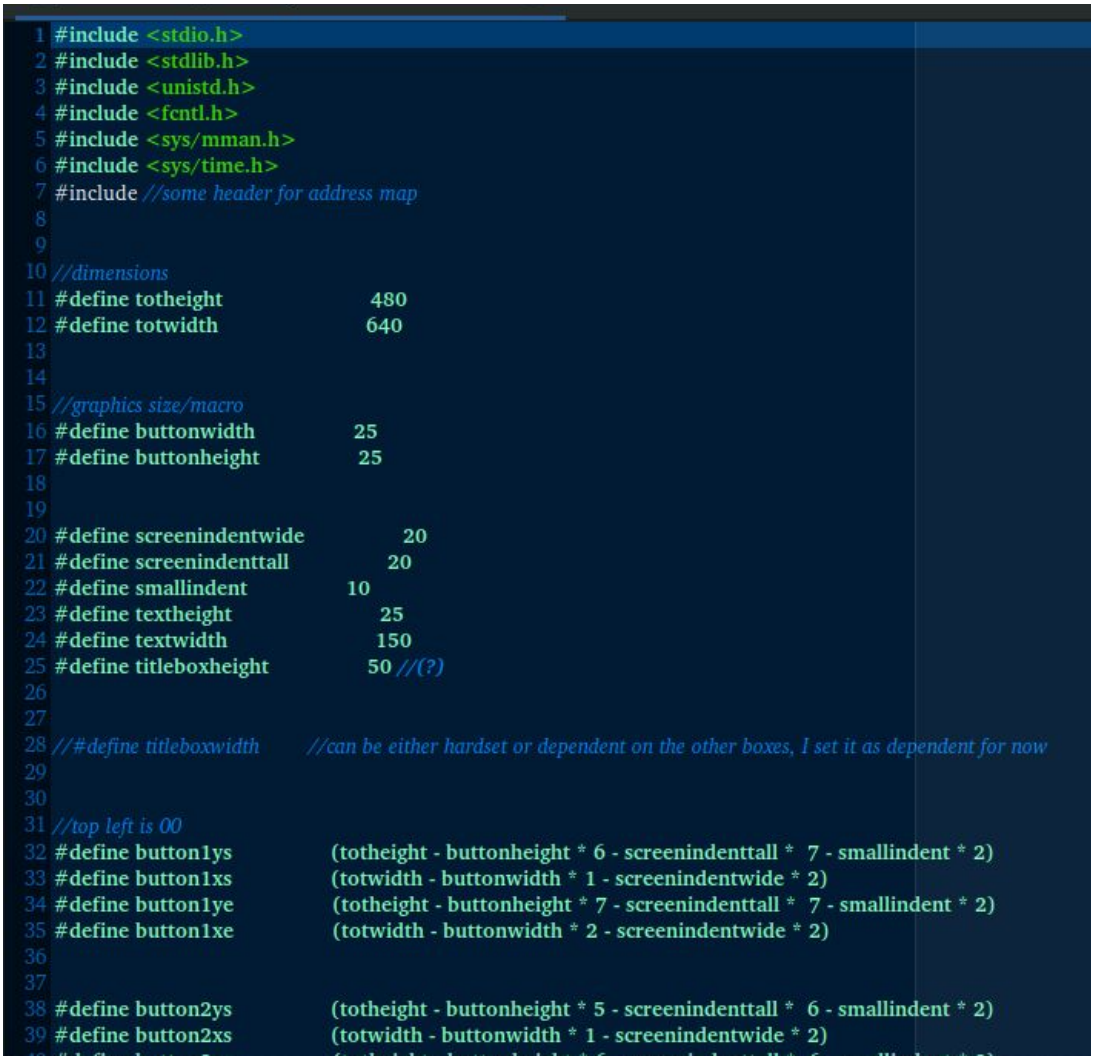
8. If Time Actually "Permitted"

Had we had more time, we would have had the ability to made the user interface a lot nicer, for example by fully integrating the second iteration of the software code (see section 10.1). We would have worked on using the software code to import the buffers that held the values for the various letters and numbers. We would also have been able to add a lot more functionality to our program. We had initially decided to add in operations on values like "horizontal sweep" and "vertical sweep" and display values like amplitude and labels that we were not able to accomplish by the deadline as the ADC took a long time for us to figure out and have control over.

9. Previous Iterations

9.1. Software

The user interface had a number of iterations, two of which were completely discarded. The first iteration included a static map of pixels to be displayed and was combined into a single file along with the mouse function.

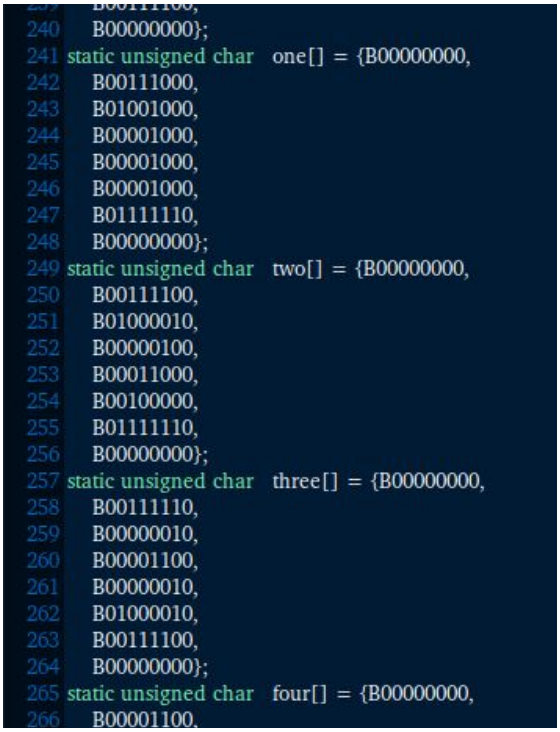


```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/time.h>
#include //some header for address map


//dimensions
#define totheight                480
#define totwidth                 640


//graphics size/macro
#define buttonwidth             25
#define buttonheight             25


#define screenindentwide            20
#define screenindenttall           20
#define smallindent             10
#define textheight                25
#define textwidth                 150
#define titleboxheight            50 //(?)


//#define titleboxwidth       //can be either hardset or dependent on the other boxes, I set it as dependent for now


//top left is 00
#define button1ys           (totheight - buttonheight * 6 - screenindenttall *  7 - smallindent * 2)
#define button1xs           (totwidth - buttonwidth * 1 - screenindentwide * 2)
#define button1ye           (totheight - buttonheight * 7 - screenindenttall *  7 - smallindent * 2)
#define button1xe           (totwidth - buttonwidth * 2 - screenindentwide * 2)


#define button2ys           (totheight - buttonheight * 5 - screenindenttall *  6 - smallindent * 2)
#define button2xs           (totwidth - buttonwidth * 1 - screenindentwide * 2)
```

Debugging Image A: Software Iteration 1
Intended to be a single c file holding UI and drivers

This code worked by reading in information from the standard "/dev/input/mice" used in most linux systems. The mouse display was created through pixel switches offset from read data. It read information from the mouse as three separate sections each corresponding to x position, y position, and button click status (any extraneous information was never allowed to be read, and as such discarded). It compared the mouse location and left click status to the hardcoded button positions, shown towards the bottom of Debugging Image A. If click was active and the position corresponded to a button, the corresponding function was called to change the displayed text. Dynamic character arrays (array memory allocations, not “arrays” in any sort of sense, simply a way to think of how numbers were displayed) were used to hold all changing text fields and then passed to a “VGA_BOX” module. Standard static character arrays were used for non-changing text displays and also sent to the “VGA_BOX” module. This software was discarded, as there was not enough memory on our SD card to display the static buttons.

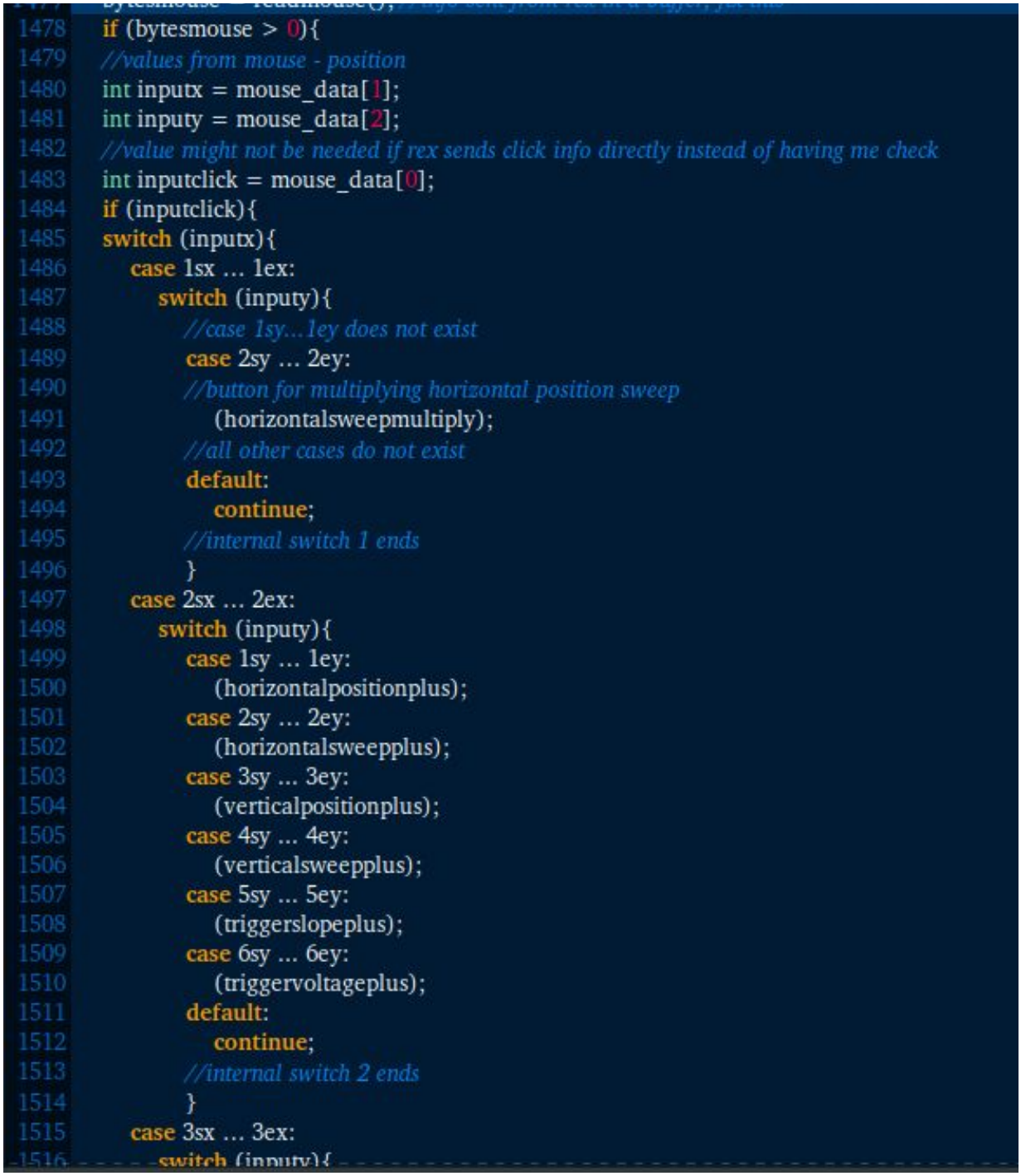
The second iteration of the software changed from this static pixel mapping to the use of tiles, as these would be passed in a temporary (ever-changing) buffer and did not have to be stored anywhere beyond the static pixels of the screen. In addition, this iteration differed from the first in the sense that the mouse was coded in a separate c file as noted in the Mouse Section (2.7). Being a less complex program, this version of the mouse remained in use beyond the failure of this software iteration as a whole. An excerpt of the tile encodings is shown in Debugging Image B to demonstrate formatting of a standard tile.



```
240    B00000000};
241 static unsigned char   one[] = {B00000000,
242    B00111000,
243    B01001000,
244    B00001000,
245    B00001000,
246    B00001000,
247    B01111110,
248    B00000000};
249 static unsigned char   two[] = {B00000000,
250    B00111100,
251    B01000010,
252    B00000100,
253    B00011000,
254    B00100000,
255    B01111110,
256    B00000000};
257 static unsigned char   three[] = {B00000000,
258    B00111110,
259    B00000010,
260    B00001100,
261    B00000010,
262    B01000010,
263    B00111100,
264    B00000000};
265 static unsigned char   four[] = {B00000000,
266    B00001100,
```

Debugging Image B : Tile Format
For use in software iteration 2

This iteration saw everything from static text to dynamic values interpreted in this tile format. Similar to the previous iteration, an (x,y,click) check was performed against predetermined button positions, calling the corresponding function if a button was clicked.

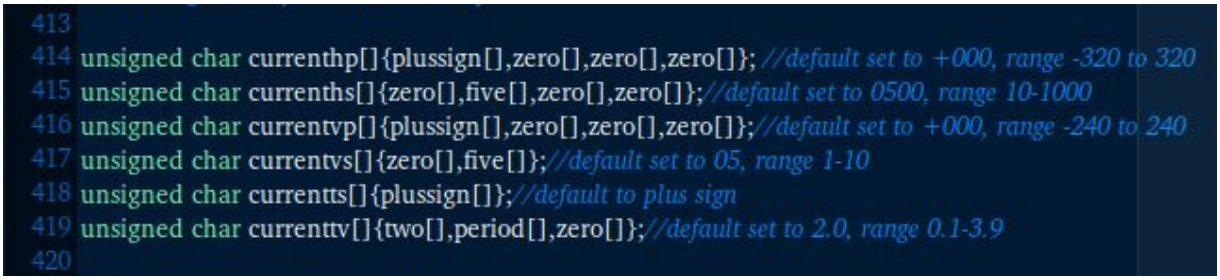


```
1478  if (bytesmouse > 0){
1479  //values from mouse - position
1480  int inputx = mouse_data[1];
1481  int inputy = mouse_data[2];
1482  //value might not be needed if rex sends click info directly instead of having me check
1483  int inputclick = mouse_data[0];
1484  if (inputclick){
1485  switch (inputx){
1486     case 1sx … 1ex:
1487        switch (inputy){
1488           //case 1sy…1ey does not exist
1489           case 2sy … 2ey:
1490           //button for multiplying horizontal position sweep
1491              (horizontalsweepmultiply);
1492           //all other cases do not exist
1493           default:
1494              continue;
1495           //internal switch 1 ends
1496           }
1497     case 2sx … 2ex:
1498        switch (inputy){
1499           case 1sy … 1ey:
1500              (horizontalpositionplus);
1501           case 2sy … 2ey:
1502              (horizontalsweepplus);
1503           case 3sy … 3ey:
1504              (verticalpositionplus);
1505           case 4sy … 4ey:
1506              (verticalsweepplus);
1507           case 5sy … 5ey:
1508              (triggerslopeplus);
1509           case 6sy … 6ey:
1510              (triggervoltageplus);
1511           default:
1512              continue;
1513           //internal switch 2 ends
1514           }
1515     case 3sx … 3ex:
1516        switch (inputy){
```

Debugging Image C: Case Switches for Button Correspondence

As seen in Debugging Image C, there were many functions in this iteration, all associated with distinct button variable change options. As the values presented on the screen were all tiles encoded in bit display data (8x8), there was no straightforward way to perform operations with any given variable on the screen. To make intended button operations work, a calculator had to be coded to work with “images.”

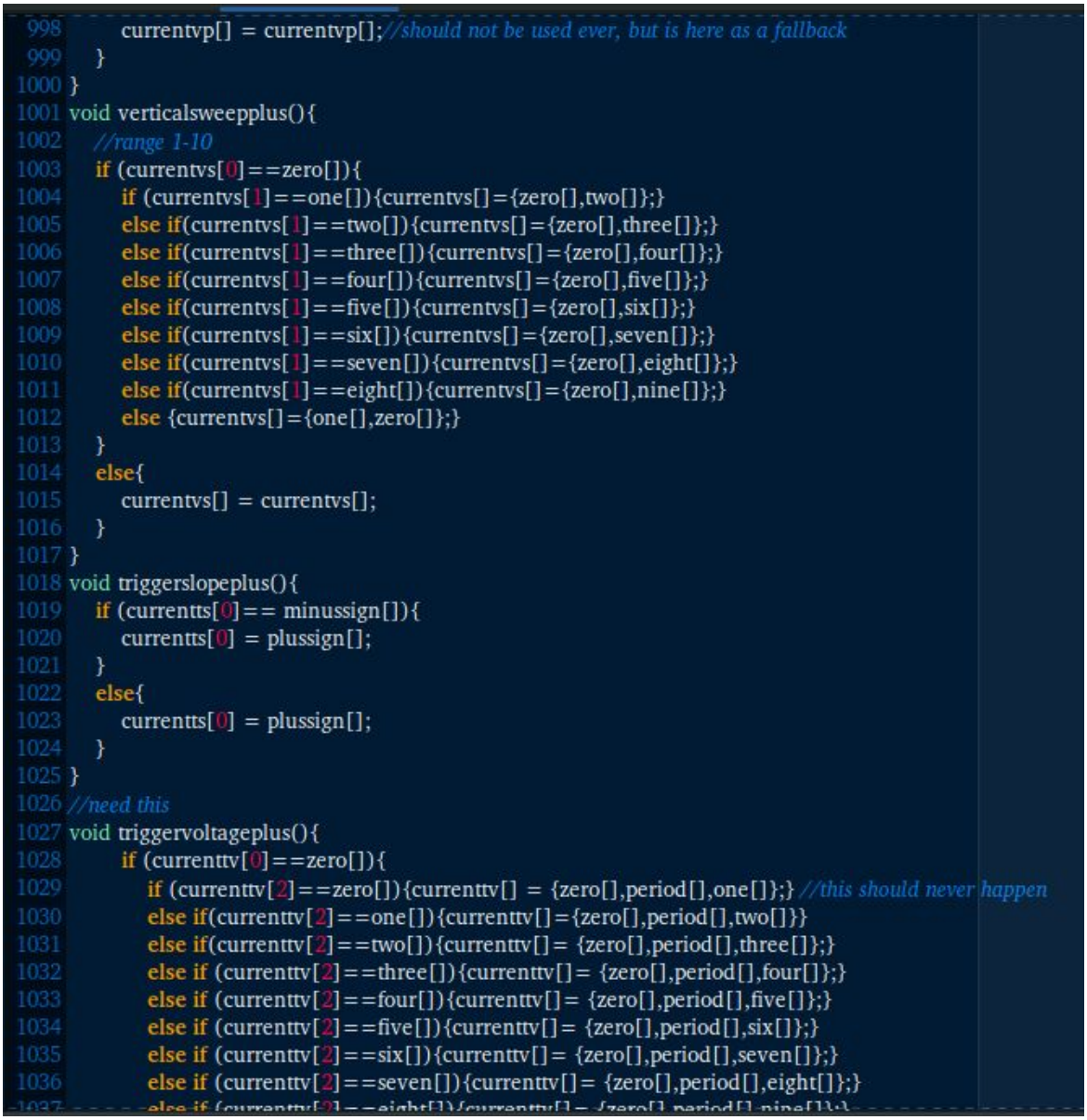
This calculator had different restrictions in each button function to correspond with the limits of the given operation (for example decimal operations in voltage, negative operations in position options, division in horizontal sweep, etc.).



```
413
414 unsigned char currenthp[]{plussign[],zero[],zero[],zero[]}; //default set to +000, range -320 to 320
415 unsigned char currenths[]{zero[],five[],zero[],zero[]};//default set to 0500, range 10-1000
416 unsigned char currentvp[]{plussign[],zero[],zero[],zero[]};//default set to +000, range -240 to 240
417 unsigned char currentvs[]{zero[],five[]};//default set to 05, range 1-10
418 unsigned char currentts[]{plussign[]};//default to plus sign
419 unsigned char currenttv[]{two[],period[],zero[]};//default set to 2.0, range 0.1-3.9
420
```

Debugging Image D : Set of volatile values for display where each internal array was passed as a continuous bit stream to the receiving buffer

An issue this iteration of the software would have had would be that of bit reversal. As the bits on the hardware code were being read in the opposite direction, the image was a flipped version of that which was encoded in the tile data. To fix this, a function was then written in the hardware code to flip incoming bit buffers. An alternate option would have been to do an equivalent function on the software side before forwarding the bits or recode the tiles with reversed bits.



```
998         currentvp[] = currentvp[];//should not be used ever, but is here as a fallback
999     }
1000 }
1001 void verticalsweepplus(){
1002     //range 1-10
1003     if (currentvs[0]==zero[]){
1004         if (currentvs[1]==one[]){currentvs[]={zero[],two[]};}
1005         else if(currentvs[1]==two[]){currentvs[]={zero[],three[]};}
1006         else if(currentvs[1]==three[]){currentvs[]={zero[],four[]};}
1007         else if(currentvs[1]==four[]){currentvs[]={zero[],five[]};}
1008         else if(currentvs[1]==five[]){currentvs[]={zero[],six[]};}
1009         else if(currentvs[1]==six[]){currentvs[]={zero[],seven[]};}
1010         else if(currentvs[1]==seven[]){currentvs[]={zero[],eight[]};}
1011         else if(currentvs[1]==eight[]){currentvs[]={zero[],nine[]};}
1012         else {currentvs[]={one[],zero[]};}
1013     }
1014     else{
1015         currentvs[] = currentvs[];
1016     }
1017 }
1018 void triggerslopeplus(){
1019     if (currentts[0]== minussign[]){
1020         currentts[0] = plussign[];
1021     }
1022     else{
1023         currentts[0] = plussign[];
1024     }
1025 }
1026 //need this
1027 void triggervoltageplus(){
1028         if (currenttv[0]==zero[]){
1029             if (currenttv[2]==zero[]){currenttv[] = {zero[],period[],one[]};} //this should never happen
1030             else if(currenttv[2]==one[]){currenttv[]={zero[],period[],two[]}}
1031             else if(currenttv[2]==two[]){currenttv[]= {zero[],period[],three[]};}
1032             else if (currenttv[2]==three[]){currenttv[]= {zero[],period[],four[]};}
1033             else if (currenttv[2]==four[]){currenttv[]= {zero[],period[],five[]};}
1034             else if (currenttv[2]==five[]){currenttv[]= {zero[],period[],six[]};}
1035             else if (currenttv[2]==six[]){currenttv[]= {zero[],period[],seven[]};}
1036             else if (currenttv[2]==seven[]){currenttv[]= {zero[],period[],eight[]};}
1037             else if (currenttv[2]==eight[]){currenttv[]= {zero[],period[],nine[]};}
```

Debugging Image E : Some Button Function Logic

Returns to buffer excluded, functions set as void, as this was never fully integrated

Shown functions (in order): Increase vertical sweep, set slope trigger to positive, increase trigger value in units

10. Conclusion

Learning how to program an ADC was extremely challenging but we learned a wealth of new skills including: research of product, usage of debugging tools, and the importance of timing.
We are all new to Verilog programing this semester, at the end of lab1 we would not have imagined we would come to enjoy the process of planning, designing and implementing a solution on an FPGA. Now, we enjoy the coding and don't want to give the FPGA back.

Communication is really important for a group project. We always need to talk with each other to make sure that we are on the same page. It is also a challenge to integrate everyone's job, but we did it.

We are very thankful for the invaluable support from the TAs and Professor Edwards. We met many problems and challenges throughout the project. Without the patient explanation and suggestions from Professor Edwards, we would not have been able to get the final wave visualizer.

11. References

https://wiki.osdev.org/USB_Human_Interface_Devices
https://usb.org/sites/default/files/documents/hid1_11.pdf
https://www.analog.com/media/en/technical-documentation/data-sheets/2308fc.pdf
https://people.ece.cornell.edu/land/courses/eceprojectsland/STUDENTPROJ/2015to2016/hj424/

12. Repository: https://github.com/oscilloscope-prime/scope

13. Appendix: Code Listing

13.1. adc.sv (Functional ADC pre-integration, working with JTAG)

```
// CSEE 4840 Final Project: working with DE1-soc ADC
//
// Spring 2019
//
// By: oscilloscope group
// Uni: Ishraq (ibk2110) and Klarizsa (ksp2127)


module adc(         input logic         CLOCK_50,

                                        output logic                ADC_CS_N,
                                        output logic                ADC_SCLK,
                                        output logic                ADC_DIN,
                                        output logic [6:0]      HEX0, HEX1, HEX2, HEX3, HEX4, HEX5,
                                        output logic [11:0]     ADC_REG,
                                        input logic             ADC_DOUT,

                                        output logic                valid,
                                        input logic                 full,
                                        input logic [11:0]      trig,
                                        input logic                 rising
    );

//logic [11:0]          ADC_REG;

logic   ADC_DoSth;
```

```
		logic		validSample = 1'd 0;

		logic		[3:0] disp0, disp1, disp2, disp3, disp4, disp5;


		clockdiv cd(.clk(CLOCK_50), .en(ADC_SCLK));

		dosomething ds(.clk(CLOCK_50), .en(ADC_DoSth));

		chipselect cs(.mclk(CLOCK_50), .ds(ADC_DoSth), .csn(ADC_CS_N));

		toADC data2ADC(.mclk(CLOCK_50), .ds(ADC_DoSth), .cs(ADC_CS_N), .din(ADC_DIN));

		fromADC data4mADC(.mclk(CLOCK_50), .ds(ADC_DoSth), .cs(ADC_CS_N), .dout(ADC_DOUT),
.out(ADC_REG), .validSample(validSample));

		sendData send2vga(.validSample(validSample), .valid(valid));

		bin2dec b2d(.bin_data(ADC_REG), .dec0(disp0), .dec1(disp1), .dec2(disp2), .dec3(disp3), .dec4(disp4),
.dec5(disp5));

		hex7seg h0(.in(disp0), .out(HEX0));
		hex7seg h1(.in(disp1), .out(HEX1));
		hex7seg h2(.in(disp2), .out(HEX2));
		hex7seg h3(.in(disp3), .out(HEX3));
		hex7seg h4(.in(disp4), .out(HEX4));
		hex7seg h5(.in(disp5), .out(HEX5));

endmodule

module clockdiv(input logic clk, output logic en);

  parameter clockDivisor = 4'd 4 ;
  //register stores the value of clock cycles
  logic [3:0] i = 4'd 0;

  always_ff @( posedge clk )
  begin

    i <= i + 4'd 1;
    //resetting the clock
    if ( i >= (clockDivisor-1))
    begin
		i <= 4'd 0;
    end

  end

  assign en = (i<clockDivisor/2)?1'b0:1'b1;

endmodule


module dosomething(input logic clk, output logic en);

		logic [3:0] counter = 4'd 0;
```

```
	logic up_down = 1'd 0;

	always_ff @( posedge clk )
	begin

		if(counter == 4'd 0 && up_down == 1'd 0)
		begin
			up_down <= 1'd 1;
			counter <= counter + 4'd 1;
		end
		else if(counter == 4'd 1 && up_down == 1'd 1)
		begin
			up_down <= 1'd 0;
			counter <= counter + 4'd 1;
		end
		else if (counter == 4'd 2 && up_down == 1'd 0)
		begin
			counter <= counter + 4'd 1;
		end
		else
		begin
			counter <= 4'd 0;
		end

	end

	assign en = up_down;

endmodule


module chipselect(input logic mclk, input logic ds, output logic csn);

	logic [5:0] counter_down = 6'd 0; //counter(we need 12 cycles of low, 1 cycle of high)
	logic [5:0] counter_up = 6'd 0;
	logic chipselect = 1'd 1; //to control the value of chipselect
	logic hold1, hold2, hold3; //to introduce a cycle of delay on chipselect


	always_ff @ ( posedge mclk )
	begin

		if(ds && counter_up <= 6'd 20 && counter_down == 6'd 0)
		begin
			chipselect <= 1'd 1;
			counter_up <= counter_up + 6'd 1;
		end

		else if(ds && counter_up == 6'd 21 && counter_down == 6'd 0)
		begin
			chipselect <= 1'd 0;
			counter_down <= counter_down + 6'd 1;
			counter_up <= 6'd 0;
		end

		else if(ds && counter_up == 6'd 0 && counter_down <= 6'd 12)
		begin
```

```
                        chipselect <= 1'd 0;
                        counter_down <= counter_down + 6'd 1;
                end

                else if(chipselect == 1'd 0 && counter_down == 6'd 13)
                begin
                        counter_down <= 6'd 0;
                        counter_up <= 6'd 0;
                        chipselect <= 1'd 1;
                end
                hold1 <= chipselect;
                hold2 <= hold1;
                hold3 <= hold2;

        end

        assign csn = hold3;

endmodule


//controls the D_in signal
module toADC (input logic mclk, input logic ds, input logic cs, output logic din);
//make a shift register to send data to ADC

        logic [5:0] shiftreg = 6'b 110010; //send to channel 1. for channel 0 ==> 100010
        logic [5:0] counter = 6'd 0;

        always_ff @ ( posedge mclk )
        begin

                if (!cs && ds && counter < 6'd 6 )
                begin
                        din <= shiftreg[5];
                        shiftreg [5:1] <= shiftreg[4:0];
                        shiftreg [0] <= 1'd 0;
                        counter <= counter + 6'd 1;
                end

                else if(counter == 6'd 6 && !ds && cs)
                begin
                        din <= din;
                        shiftreg <= 6'b 100010;
                        counter <= 6'd 0;
                end

                else
                        din <= din;

        end

endmodule

//controls the D_out signal
module fromADC (input logic mclk, input logic ds, input logic cs, input logic dout, output logic [11:0] out, output logic
validSample);
/*
```

```
		input logic mclk, ds, cs, dout;
		output logic [11:0] out;
*/
		logic [5:0] counter = 6'd 0;

		logic [11:0] shiftreg;

		logic load_data = 1'd 1; //check this if we have issues displaying

		always_ff @ ( posedge mclk )
		begin

			if (!cs && ds && counter <= 6'd 11)
			begin
				shiftreg = {shiftreg[10:0], dout};
				counter <= counter + 6'd 1;
				load_data <= 1'd 1;
			end

			else if(cs && !ds && load_data)
			begin
				//counter = 6'd 0;
				//out <= shiftreg;
				out[11:0] <= shiftreg[11:0];
				counter = 6'd 0;
				load_data <= 1'd 0;
				validSample <= 1'd 1;
			end
			else
			begin
				validSample <= 1'd 0;
			end

		end

endmodule

module sendData(input logic validSample, output logic valid);

		always_comb
		begin
/*			if(full)
				valid <= 1d'0;
			else if(!full && validSample):
				valid <= 1d'1;
			else
				valid <= 1d'0;
*/
			valid <= validSample;
		end
endmodule

module hex7seg (input logic [3:0] in, output logic [0:7] out);

		logic [6:0] pre_seg_dis;
		always @ (*)
		begin
```

```
        case(in)

                4'h1: pre_seg_dis = 7'b1111001;
                4'h2: pre_seg_dis = 7'b0100100;
                4'h3: pre_seg_dis = 7'b0110000;
                4'h4: pre_seg_dis = 7'b0011001;
                4'h5: pre_seg_dis = 7'b0010010;
                4'h6: pre_seg_dis = 7'b0000010;
                4'h7: pre_seg_dis = 7'b1111000;
                4'h8: pre_seg_dis = 7'b0000000;
                4'h9: pre_seg_dis = 7'b0011000;
                4'ha: pre_seg_dis = 7'b0001000;
                4'hb: pre_seg_dis = 7'b0000011;
                4'hc: pre_seg_dis = 7'b1000110;
                4'hd: pre_seg_dis = 7'b0100001;
                4'he: pre_seg_dis = 7'b0000110;
                4'hf: pre_seg_dis = 7'b0001110;
                4'h0: pre_seg_dis = 7'b1000000;

        endcase

    end

    assign out = pre_seg_dis;

endmodule


module bin2dec (input logic [11:0] bin_data, output logic [3:0] dec0, output logic [3:0] dec1, output logic [3:0] dec2,
output logic [3:0] dec3, output logic [3:0] dec4, output logic [3:0] dec5);

    always @ (*)
    begin
    dec0 = (bin_data*409600/4096 ) %10;
    dec1 = (bin_data*409600/4096 /10) %10;
    dec2 = (bin_data*409600/4096 /100) %10;
    dec3 = (bin_data*409600/4096 /1000) %10;
    dec4 = (bin_data*409600/4096 /10000) %10;
    dec5 = (bin_data*409600/4096 /100000) %10;
    end

endmodule
```

13.2. vga_ball.sv (All components integrated)

```
/*
 * Avalon memory-mapped peripheral that generates VGA & ADC interface
 *
 * Oscilloscope Project
 */


module vga_ball(    input logic             clk,
    input logic             reset,
    input logic [15:0]  writedata,
    input logic             write,
```

```
        input                                   chipselect,
        input logic [2:0]       address,

                                //VGA logic
        output logic [7:0]      VGA_R, VGA_G, VGA_B,
        output logic            VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n,
        output logic            VGA_SYNC_n,

                                //ADC logic
                                output logic            ADC_CS_N,
                                output logic            ADC_SCLK,
                                output logic            ADC_DIN,
                                input logic             ADC_DOUT,

                                output logic [6:0]      HEX0, HEX1, HEX2, HEX3, HEX4, HEX5
                        );

        logic [11:0] ADC_REG;

        logic [3:0] testHex;
        initial begin
                testHex = 4'd0;
        end

        logic prev_trig;
        logic cur_trig;
/*
        assign prev_trig = 0;
        assign cur_trig = 0;
*/
        logic   ADC_DoSth;

        logic       [3:0] disp0, disp1, disp2, disp3, disp4, disp5;

        logic [8:0] sample;
        assign sample [8:0] = ADC_REG[11:3];

        logic ready;
        logic valid;
        //assign valid = 1;

        logic full;

        logic [11:0] trig =12'd1;
        //assign trig = 12'd 2000; //trigger looks for 2V
        logic [11:0] trig_thousand;
        assign trig_thousand = trig * 12'd 1000;

        logic rising = 1'b 1;
        //assign rising = 1'b 1; //for rising edge. toggle to 0 for falling


/* ======================================================================vga
stuff=============================================================================
======= */

 logic [10:0]           hcount;
```

```
  logic [9:0]     vcount;

  logic [7:0]                 background_r, background_g, background_b;
  logic [10:0]     posX;
  logic [9:0]      posY; //trigger
  logic [10:0]     dummy;

  logic [1:0] flag;
  logic [1:0] flag2;



//some variables for memory

                /*input logic       clk,*/
                logic [9:0]  a1;
                logic [8:0]  din1;
                logic we1;
                logic [15:0] dout1;

                logic [9:0]  a2;
                logic [8:0]  din2;
                logic we2;
                logic [15:0] dout2;

/* ================================================================================ ADDING
MOUSE ================================================================================ */

                        logic [7:0] shape;
                        logic [2:0] a_m;

                        logic [15:0] shape_p;
                        logic [3:0] a_p;

                        logic [15:0] shape_mi;
                        logic [3:0] a_mi;

                        logic [15:0] shape_t;
                        logic [3:0] a_t;

                        logic [15:0] shape_r;
                        logic [3:0] a_r;

                        logic [15:0] shape_p1;
                        logic [3:0] a_p1;

                        logic [15:0] shape_mi1;
                        logic [3:0] a_mi1;

//FOR ZERO RISING
                        logic [15:0] shape_o;
                        logic [3:0] a_o;
//FOR ONE RISING
                        logic [15:0] shape_one;
                        logic [3:0] a_one;

mouse mouse(.*);
```

```
plus p(.*), p1(clk, shape_p1,a_p1);
minus m(.*), mi1(clk, shape_mi1,a_mi1);
T t(.*);
R r(.*);

//for rising
zero z(.*);
one o(.*);

//FOR ZERO Trigger
                        logic [15:0] shape_o_trg;
                        logic [3:0] a_o_trg;
//FOR ONE Trigger
                        logic [15:0] shape_one_trg;
                        logic [3:0] a_one_trg;

                        logic [15:0] shape_two;
                        logic [3:0] a_two;

                        logic [15:0] shape_th;
                        logic [3:0] a_th;


//for trigger

zero z1(clk,shape_o_trg,a_o_trg);
one o1(clk,shape_one_trg,a_one_trg);
th th(.*);
two two(.*);

/* ========================================================================== ADDING
MOUSE ========================================================================== */

logic first = 1'b1;

logic [9:0] a_display;
logic [9:0] a_input = 10'b0;
logic [15:0] dout_display;
logic [8:0] din_input;
logic  we_input;
assign we_input = valid;
assign din_input = sample;
assign din1 = din_input;
assign din2 = din_input;

//FOR RISING
assign a_one = vcount - 400;
assign a_o = vcount -400;

//FOR TRIG
assign a_one_trg = vcount - 325;
assign a_o_trg = vcount - 325;
assign a_two = vcount - 325;
assign a_th = vcount - 325;
                        logic [15:0] shape_trg;
                        logic [3:0] a_trg;
```

```
                        logic [15:0] shape_rising;
                        logic [3:0] a_rising;

memory m1(clk, a1, din1, we1, dout1),
     m2(clk, a2, din2, we2, dout2);

vga_counters counters(.clk50(clk), .*);

always_comb begin
//logic for RISING
            if(rising)
                        begin
                        shape_rising = shape_one;
                        a_rising = a_one;
                        end
            else
                        begin
                        shape_rising = shape_o;
                        a_rising = a_o;

                        end


end

always_comb begin
//logic for trigger
            if(trig == 12'd1)
                        begin
                        shape_trg = shape_one_trg;
                        a_trg = a_one_trg;
                        end
            else if (trig == 12'd0)
                        begin
                        shape_trg = shape_o_trg;
                        a_trg = a_o_trg;

                        end

            else if (trig == 12'd2)
                        begin
                        shape_trg = shape_two;
                        a_trg = a_two;

                        end
            else
                        begin
                        shape_trg = shape_th;
                        a_trg = a_th;

                        end


end
```

```
always_comb begin


 if (first) begin
   a1 = a_display;
   a2 = a_input;
   //din2 = din_input;
   dout_display = dout1;
   we1 = 1'b0;
   we2 = we_input;
 end else begin
   a1 = a_input;
   a2 = a_display;
   //din1 =  din_input;
   dout_display =dout2;
   we1 = we_input;
   we2 =  1'b0;
 end
 end

              logic start;
              initial begin
                        start = 1'd1;
              end

 always_ff @(posedge clk)
              begin
              if (start) begin
              full  <= 1'b1; //changed to 0
              a_input = 10'b0;
              first = 1'b1;
              start = 1'd 0;
              end

              else
              begin

              if (valid)begin

                        if(full)

                        begin
                        full <= 1'b0;
                        end

                        else if(a_input == 10'd639) begin //added here
                                  a_input <= 10'd0;
                                  full <= 1'b1;
                                  first<= ~ first;
                        end
                        else begin
                                  a_input <=  a_input + 10'd1;
                                  full <= full;
                        end
              end
              else if (!valid) begin
                        full <= full;
```

```
            end
            end
end


 always_ff @(posedge clk)
  if (reset) begin
            background_r <= 8'h0;
            background_g <= 8'h0;
            background_b <= 8'h80;

            posX <= 8'd50;
            posY <= 8'd50;
            flag <= 1'd0;
            flag2 <= 1'd0;
            dummy <= hcount;
            rising<= 1'd1;
            trig <=12'd1;

   end else if (chipselect && write) begin

    case (address)


      3'h0 : posX <= writedata;
      3'h1 : posY <= writedata;
             3'h2 : rising <= writedata;
      3'h3 : trig <= writedata;



             //3'h0 : din_input<= writedata[10:0];

             //3'h1 : posY <= writedata[10:0];



    endcase
            end

assign a_m  = vcount - posY;
assign a_p  = vcount - 350;
assign a_mi  = vcount - 350;
assign a_t  = vcount - 325;
assign a_r  = vcount - 400;

assign a_p1  = vcount - 425;
assign a_mi1  = vcount - 425;


  always_comb begin
            a_display = hcount[10:1];
    {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
    if (VGA_BLANK_n )begin
            //if( (hcount[10:0]< posX+30) && (hcount[10:0] > posX-30) && (vcount[9:0]>posY-15) && (vcount[9:0] <
posY + 15) )
```

```
            //if (((hcount[10:0] - posX)*(hcount[10:0]- posX)) + (4*(vcount[9:0]- posY)*(vcount[9:0]- posY)) < 900)
            //if ( (vcount[9:0] == posY) &&( (hcount[10:0]< posX+30) && (hcount[10:0] > posX-30) ) )
            if( ((hcount[10:1]< posX+8) && (hcount[10:1] >= posX) && (vcount[9:0]>=posY) && (vcount[9:0] < posY + 8) ) && (shape[hcount[10:1] - posX]))
                        //if (shape[hcount[10:1] - posX])
                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
            else if( ((hcount[10:1]< 550+16) && (hcount[10:1] >= 550) && (vcount[9:0]>=350) && (vcount[9:0] < 350 + 16) ) && (shape_p[hcount[10:1] - 550]))
                        //if (shape[hcount[10:1] - posX])
                        {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hff, 8'h00};

            else if( ((hcount[10:1]< 500+16) && (hcount[10:1] >= 500) && (vcount[9:0]>=350) && (vcount[9:0] < 350 + 16) ) && (shape_mi[hcount[10:1] - 500]))
                        {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hff, 8'h00};

            else if( ((hcount[10:1]< 450+16) && (hcount[10:1] >= 450) && (vcount[9:0]>=325) && (vcount[9:0] < 325 + 16) ) && (shape_t[hcount[10:1] - 450]))
                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};


            else if( ((hcount[10:1]< 450+16) && (hcount[10:1] >= 450) && (vcount[9:0]>=400) && (vcount[9:0] < 400 + 16) ) && (shape_r[hcount[10:1] - 450]))
                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};

            else if( ((hcount[10:1]< 550+16) && (hcount[10:1] >= 550) && (vcount[9:0]>=425) && (vcount[9:0] < 425 + 16) ) && (shape_p1[hcount[10:1] - 550]))
                        //if (shape[hcount[10:1] - posX])
                        {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hff, 8'h00};

            else if( ((hcount[10:1]< 500+16) && (hcount[10:1] >= 500) && (vcount[9:0]>=425) && (vcount[9:0] < 425 + 16) ) && (shape_mi1[hcount[10:1] - 500]))
                        {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hff, 8'h00};
            //else if ( (hcount[10:1]< 550+15) && (hcount[10:1] > 550-15) && (vcount[9:0]>350-15) && (vcount[9:0] < 350 + 15) )
                        //{VGA_R, VGA_G, VGA_B} = {8'h00, 8'hff, 8'h00};
            //else if ( (hcount[10:1]< 500+15) && (hcount[10:1] > 500-15) && (vcount[9:0]>350-15) && (vcount[9:0] < 350 + 15) )
                        //{VGA_R, VGA_G, VGA_B} = {8'h00, 8'hff, 8'h00};
            //RISING
            else if( ((hcount[10:1]< 525+16) && (hcount[10:1] >= 525) && (vcount[9:0]>=400) && (vcount[9:0] < 400 + 16) ) && (shape_rising[hcount[10:1] - 525]))
                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'hff};

            else if( ((hcount[10:1]< 525+16) && (hcount[10:1] >= 525) && (vcount[9:0]>=325) && (vcount[9:0] < 325 + 16) ) && (shape_trg[hcount[10:1] - 525]))
                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'hff};


            else if (dout_display[9:0] == (480 - vcount[9:0]))
            //else if (sample[8:0] == (480 - vcount[9:0]))
                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};

            else if(vcount[9:0] == 240 )
                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'h00};

            else if ( (hcount[10:1]%60 == 0 || vcount[9:0]%60 == 0 ))
                        {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h33, 8'h66};
```

```
            //else if (dout[9:0] == vcount[9:0])
                        //{VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};

/*
            if (hcount[10:6] == 5'd3   &&
               vcount[9:0]== 10'd1023)
              {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};*/


            else
              {VGA_R, VGA_G, VGA_B} =

          //{background_r, background_g, background_b};
                        //{ 8'hff, 8'h00, 8'hff };
                         { 8'h00, 8'h00, 8'h00 };
            end

  end

/* ================================================================================adc
stuff=============================================================================================
======= */

            clockdiv cd(.clk(clk), .en(ADC_SCLK));

            dosomething ds(.clk(clk), .en(ADC_DoSth));

            chipselect cs(.mclk(clk), .ds(ADC_DoSth), .csn(ADC_CS_N));

            toADC data2ADC(.mclk(clk), .ds(ADC_DoSth), .cs(ADC_CS_N), .din(ADC_DIN));

            fromADC data4mADC(.mclk(clk), .ds(ADC_DoSth), .cs(ADC_CS_N), .dout(ADC_DOUT), .out(ADC_REG),
.ready(ready));

            trackTrig trigSig(.mclk(clk), .ADC_REG(ADC_REG), .trig(trig_thousand), .cur_trig(cur_trig),
.prev_trig(prev_trig), .rising(rising), .full(full));

            sendData send2vga(.mclk(clk), .testHex(testHex), .ready(ready), .cur_trig(cur_trig), .full(full), .valid(valid),
.prev_trig(prev_trig));

            bin2dec b2d(.bin_data(ADC_REG), .dec0(disp0), .dec1(disp1), .dec2(disp2), .dec3(disp3), .dec4(disp4),
.dec5(disp5));

            hex7seg h0(.in(testHex), .out(HEX0));
            hex7edge h1 (.in(rising), .out(HEX1));
            hex7seg h2(.in(disp2), .out(HEX2));
            hex7seg h3(.in(disp3), .out(HEX3));
            hex7seg h4(.in(disp4), .out(HEX4));
            hex7seg h5(.in(disp5), .out(HEX5));


/* ================================================================================vga
stuff=============================================================================================
======= */

endmodule
```

```
module clockdiv(input logic clk, output logic en);

  logic [1:0] i = 2'b 00;

  always_ff @( posedge clk )
  begin

   i <= i + 2'd 1;

  end

  assign en = i[1];

endmodule


module dosomething(input logic clk, output logic en);

        logic [3:0] counter = 4'd 0;
        logic up_down = 1'd 0;

        always_ff @( posedge clk )
        begin

                if(counter == 4'd 0 && up_down == 1'd 0)
                begin
                        up_down <= 1'd 1;
                        counter <= counter + 4'd 1;
                end
                else if(counter == 4'd 1 && up_down == 1'd 1)
                begin
                        up_down <= 1'd 0;
                        counter <= counter + 4'd 1;
                end
                else if (counter == 4'd 2 && up_down == 1'd 0)
                begin
                        counter <= counter + 4'd 1;
                end
                else
                begin
                        counter <= 4'd 0;
                end

        end

        assign en = up_down;

endmodule


module chipselect(input logic mclk, input logic ds, output logic csn);

        logic [5:0] counter_down = 6'd 0; //counter(we need 12 cycles of low, 1 cycle of high)
        logic [5:0] counter_up = 6'd 0;
        logic chipselect = 1'd 1; //to control the value of chipselect
        logic hold1, hold2, hold3; //to introduce a cycle of delay on chipselect
```

```
	always_ff @ ( posedge mclk )
	begin

		if(ds && counter_up <= 6'd 20 && counter_down == 6'd 0)
		begin
			chipselect <= 1'd 1;
			counter_up <= counter_up + 6'd 1;
		end

		else if(ds && counter_up == 6'd 21 && counter_down == 6'd 0)
		begin
			chipselect <= 1'd 0;
			counter_down <= counter_down + 6'd 1;
			counter_up <= 6'd 0;
		end

		else if(ds && counter_up == 6'd 0 && counter_down <= 6'd 12)
		begin
			chipselect <= 1'd 0;
			counter_down <= counter_down + 6'd 1;
		end

		else if(chipselect == 1'd 0 && counter_down == 6'd 13)
		begin
			counter_down <= 6'd 0;
			counter_up <= 6'd 0;
			chipselect <= 1'd 1;
		end
		hold1 <= chipselect;
		hold2 <= hold1;
		hold3 <= hold2;

	end

	assign csn = hold3;

endmodule


//controls the D_in signal
module toADC (input logic mclk, input logic ds, input logic cs, output logic din);
//make a shift register to send data to ADC

	logic [5:0] shiftreg = 6'b 100010; //initialize shift reg to 0s
	logic [5:0] counter = 6'd 0;

	always_ff @ ( posedge mclk )
	begin

		if (!cs && ds && counter < 6'd 6 )
		begin
			din <= shiftreg[5];
			shiftreg [5:1] <= shiftreg[4:0];
			shiftreg [0] <= 1'd 0;
			counter <= counter + 6'd 1;
```

```
                end

                else if(counter == 6'd 6 && !ds && cs)
                begin
                        din <= din;
                        shiftreg <= 6'b 100010;
                        counter <= 6'd 0;
                end

                else
                        din <= din;

        end

endmodule

//controls the D_out signal
module fromADC (mclk, ds, cs, dout, ready, out);

        input logic mclk, ds, cs, dout;
        output logic [11:0] out;
        //output logic [11:0] ADC_REG_PREV;
        output logic ready;
        logic [5:0] counter = 6'd 0;

        logic [11:0] shiftreg = 12'd 0;

        logic load_data = 1'd 1; //check this if we have issues displaying

        always_ff @ ( posedge mclk )
        begin

                if (!cs && ds && counter < 6'd 1)
                begin
                        //ADC_REG_PREV[11:0] <= shiftreg[11:0];
                        counter <= counter + 6'd 1;
                        ready <= 1'd 0;
                end

                else if (!cs && ds && counter >= 6'd 1 && counter <= 6'd 12)
                //if (!cs && ds && counter <= 6'd 11)
                begin
                        shiftreg = {shiftreg[10:0], dout};
                        counter <= counter + 6'd 1;
                        load_data <= 1'd 1;
                        ready <= 1'd 0;
                        //ADC_REG_PREV[11:0] <= ADC_REG_PREV[11:0];
                end

                else if(cs && !ds && load_data)
                begin
                        //counter = 6'd 0;
                        //out <= shiftreg;
                        out[11:0] <= shiftreg[11:0];
                        //counter = 6'd 0;
                        load_data <= 1'd 0;
                        ready <= 1'd 0;
```

```
                end

                if(counter == 6'd 13)
                begin
                        ready <= 1'd 1;
                        counter <= 6'd 0;
                end
                else
                        ready <= 1'd 0;

        end

endmodule
/*
module trackTrig (input logic mclk, input logic[11:0] ADC_REG, input logic[11:0] trig, input logic full, output logic
cur_trig, output logic prev_trig, input logic rising);

        logic hold_prev = 1'b0;
        logic hold1 = 1'b0;

        always_ff @ ( posedge mclk )
        begin
        if(rising && full)
        begin

                if(ADC_REG >= trig)
                begin
                        cur_trig <= 1'b1;
                        hold_prev <= 1'b1;
                end
                else if (ADC_REG < trig)
                begin
                        cur_trig <= 1'b0;
                        hold_prev <= 1'b0;
                end
                else
                begin
                        cur_trig <= 1'b0;
                        hold_prev <= 1'b0;
                end

                hold1 <= hold_prev;

        end

        end

        assign prev_trig = hold1;


endmodule*/

module trackTrig (input logic mclk, input logic[11:0] ADC_REG, input logic[11:0] trig, input logic full, output logic
cur_trig, output logic prev_trig, input logic rising);

        logic hold_prev = 1'b0;
        logic hold1 = 1'b0;
```

```
always_ff @ ( posedge mclk )
begin
if(rising && full)
begin
        if(ADC_REG >= trig)
        begin
                cur_trig <= 1'b1;
                hold_prev <= 1'b1;
        end
        else if( ADC_REG<trig)/* added these conditions*/
        begin
                cur_trig <= 1'b0;
                hold_prev <= 1'b0;
        end/* added these conditions*/
        else
        begin
                cur_trig <= 1'b0;
                hold_prev <= 1'b0;
        end

        hold1 <= hold_prev;
end
else if(!rising && full)
begin
        if(ADC_REG <= trig)
        begin
                cur_trig <= 1'b1;
                hold_prev <= 1'b1;
        end
        else
        begin
                cur_trig <= 1'b0;
                hold_prev <= 1'b0;
        end

        hold1 <= hold_prev;
end

end

assign prev_trig = hold1;


endmodule

module sendData (input logic mclk, input logic ready, input logic cur_trig, input logic full, output logic valid, input logic
prev_trig, output logic [3:0] testHex);


        initial
        begin
        testHex=4'd0;
        end

        always_ff @ ( posedge mclk )
        begin
```

```
                //if previous trig was 0, curr trig was 1, vga_full was true and ready is true--> set valid to true
                if(!prev_trig && cur_trig && full)
                        valid <=1'd1;
                else if(prev_trig && !cur_trig && full) //added here
                begin   valid <=1'd1;
                                testHex <= 4'd5;
                end
                else if(!full && ready && valid)
                        valid<= 1'd0;
                else if(!full && ready && !valid)
                        valid<= 1'd1;
                else
                        valid<=1'd0;


        end
endmodule

module hex7seg (input logic [3:0] in, output logic [0:7] out);

        logic [6:0] pre_seg_dis;
        always @ (*)
        begin

                case(in)

                        4'h1: pre_seg_dis = 7'b1111001;
                        4'h2: pre_seg_dis = 7'b0100100;
                        4'h3: pre_seg_dis = 7'b0110000;
                        4'h4: pre_seg_dis = 7'b0011001;
                        4'h5: pre_seg_dis = 7'b0010010;
                        4'h6: pre_seg_dis = 7'b0000010;
                        4'h7: pre_seg_dis = 7'b1111000;
                        4'h8: pre_seg_dis = 7'b0000000;
                        4'h9: pre_seg_dis = 7'b0011000;
                        4'ha: pre_seg_dis = 7'b0001000;
                        4'hb: pre_seg_dis = 7'b0000011;
                        4'hc: pre_seg_dis = 7'b1000110;
                        4'hd: pre_seg_dis = 7'b0100001;
                        4'he: pre_seg_dis = 7'b0000110;
                        4'hf: pre_seg_dis = 7'b0001110;
                        4'h0: pre_seg_dis = 7'b1000000;

                endcase

        end

        assign out = pre_seg_dis;

endmodule

module hex7edge (input logic in, output logic [0:7] out);

        logic [6:0] pre_seg_dis;
        always @ (*)
        begin
```

```
                case(in)

                        4'h1: pre_seg_dis = 7'b1001110;

                        4'h0: pre_seg_dis = 7'b0001110;

                endcase

        end

        assign out = pre_seg_dis;

endmodule


module bin2dec (input logic [11:0] bin_data, output logic [3:0] dec0, output logic [3:0] dec1, output logic [3:0] dec2,
output logic [3:0] dec3, output logic [3:0] dec4, output logic [3:0] dec5);

        always @ (*)
        begin
        dec0 = (bin_data*409600/4096 ) %10;
        dec1 = (bin_data*409600/4096 /10) %10;
        dec2 = (bin_data*409600/4096 /100) %10;
        dec3 = (bin_data*409600/4096 /1000) %10;
        dec4 = (bin_data*409600/4096 /10000) %10;
        dec5 = (bin_data*409600/4096 /100000) %10;
        end

endmodule




/* ======================================================================================VGA
Modules=========================================================================
======== */


module vga_counters(
 input logic                 clk50, reset,
 output logic [10:0] hcount,  // hcount[10:1] is pixel column
 output logic [9:0]  vcount,  // vcount[9:0] is pixel row
 output logic                VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);


  // Parameters for hcount
  parameter HACTIVE      = 11'd 1280,
            HFRONT_PORCH = 11'd 32,
            HSYNC        = 11'd 192,
            HBACK_PORCH  = 11'd 96,
            HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
                           HBACK_PORCH; // 1600

  // Parameters for vcount
```

```
  parameter VACTIVE      = 10'd 480,
            VFRONT_PORCH = 10'd 10,
            VSYNC        = 10'd 2,
            VBACK_PORCH  = 10'd 33,
            VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC +
                           VBACK_PORCH; // 525

  logic endOfLine;

  always_ff @(posedge clk50 or posedge reset)
    if (reset)          hcount <= 0;
    else if (endOfLine) hcount <= 0;
    else                hcount <= hcount + 11'd 1;

  assign endOfLine = hcount == HTOTAL - 1;

  logic endOfField;

  always_ff @(posedge clk50 or posedge reset)
    if (reset)          vcount <= 0;
    else if (endOfLine)
      if (endOfField)   vcount <= 0;
      else              vcount <= vcount + 10'd 1;

  assign endOfField = vcount == VTOTAL - 1;

  // Horizontal sync: from 0x520 to 0x5DF (0x57F)
  // 101 0010 0000 to 101 1101 1111
  assign VGA_HS = !( (hcount[10:8] == 3'b101) &
                     !(hcount[7:5] == 3'b111));
  assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

  assign VGA_SYNC_n = 1'b0; // For putting sync on the green signal; unused

  assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
                       !( vcount[9] | (vcount[8:5] == 4'b1111) );

  /* VGA_CLK is 25 MHz
   *             __    __    __
   * clk50    __|  |__|  |__|
   *
   *           _____       __
   * hcount[0]__|     |_____|
   */
  assign VGA_CLK = hcount[0]; // 25 MHz clock: rising edge sensitive

endmodule



//MOUSE
module mouse(input logic clk, output logic [7:0] shape, input logic [2:0] a_m);

logic [7:0]     mse[7:0];

initial begin
mse[0] = 8'b01111111;
```

```
mse[1] = 8'b00111111;
mse[2] = 8'b00011111;
mse[3] = 8'b00011111;
mse[4] = 8'b00111111;
mse[5] = 8'b01110011;
mse[6] = 8'b11100001;
mse[7] = 8'b11000000;
end

always_ff @(posedge clk) begin
shape <= mse[a_m];
end
endmodule



// +
module plus(input logic clk, output logic [15:0] shape_p, input logic [3:0] a_p);

logic [15:0]     mse[15:0];

initial begin
mse[0] = 16'b0000000000000000;
mse[1] = 16'b0000000000000000;
mse[2] = 16'b0000001111000000;
mse[3] = 16'b0000001111000000;
mse[4] = 16'b0000001111000000;
mse[5] = 16'b0000001111000000;
mse[6] = 16'b0011111111111100;
mse[7] = 16'b0011111111111100;
mse[8] = 16'b0011111111111100;
mse[9] = 16'b0011111111111100;
mse[10] =16'b0000001111000000;
mse[11] =16'b0000001111000000;
mse[12] =16'b0000001111000000;
mse[13] =16'b0000001111000000;
mse[14] =16'b0000000000000000;
mse[15] =16'b0000000000000000;
end

always_ff @(posedge clk) begin
shape_p <= mse[a_p];
end
endmodule

// -
module minus(input logic clk, output logic [15:0] shape_mi, input logic [3:0] a_mi);

logic [15:0]     mse[15:0];

initial begin
mse[0] = 16'b0000000000000000;
mse[1] = 16'b0000000000000000;
mse[2] = 16'b0000000000000000;
mse[3] = 16'b0000000000000000;
mse[4] = 16'b0000000000000000;
mse[5] = 16'b0000000000000000;
```

```
mse[6] = 16'b0011111111111100;
mse[7] = 16'b0011111111111100;
mse[8] = 16'b0011111111111100;
mse[9] = 16'b0011111111111100;
mse[10] =16'b0000000000000000;
mse[11] =16'b0000000000000000;
mse[12] =16'b0000000000000000;
mse[13] =16'b0000000000000000;
mse[14] =16'b0000000000000000;
mse[15] =16'b0000000000000000;
end

always_ff @(posedge clk) begin
shape_mi <= mse[a_mi];
end
endmodule

// T
module T(input logic clk, output logic [15:0] shape_t, input logic [3:0] a_t);

logic [15:0]     mse[15:0];

initial begin
mse[0] = 16'b1111111111111111;
mse[1] = 16'b1111111111111111;
mse[2] = 16'b1111111111111111;
mse[3] = 16'b1111111111111111;
mse[4] = 16'b0000001111000000;
mse[5] = 16'b0000001111000000;

mse[6] = 16'b0000001111000000;
mse[7] = 16'b0000001111000000;
mse[8] = 16'b0000001111000000;
mse[9] = 16'b0000001111000000;
mse[10] =16'b0000001111000000;
mse[11] =16'b0000001111000000;
mse[12] =16'b0000001111000000;
mse[13] =16'b0000001111000000;
mse[14] =16'b0000001111000000;
mse[15] =16'b0000001111000000;
end

always_ff @(posedge clk) begin
shape_t <= mse[a_t];
end
endmodule

// R
module R(input logic clk, output logic [15:0] shape_r, input logic [3:0] a_r);
//assign a_m  = vcount - posY;
logic [15:0]     mse[15:0];

initial begin
mse[0] = 16'b0011111111111111;
mse[1] = 16'b0011111111111111;
mse[2] = 16'b0011110000001111;
mse[3] = 16'b0011110000001111;
```

```
mse[4] = 16'b0011110000001111;
mse[5] = 16'b0011110000001111;
mse[6] = 16'b0011110000001111;
mse[7] = 16'b0011110000001111;
mse[8] = 16'b0011111111111111;
mse[9] = 16'b0011111111111111;
mse[10] =16'b0000000111101111;
mse[11] =16'b0000001111001111;
mse[12] =16'b0000011110001111;
mse[13] =16'b0000111100001111;
mse[14] =16'b0001111000001111;
mse[15] =16'b0011111000001111;
end

always_ff @(posedge clk) begin
shape_r <= mse[a_r];
end
endmodule

// 0
module zero(input logic clk, output logic [15:0] shape_o, input logic [3:0] a_o);

logic [15:0]     mse[15:0];

initial begin
mse[0] = 16'b1111111111111111;
mse[1] = 16'b1111111111111111;
mse[2] = 16'b1111000000001111;
mse[3] = 16'b1111000000001111;
mse[4] = 16'b1111000000001111;
mse[5] = 16'b1111000000001111;
mse[6] = 16'b1111000000001111;
mse[7] = 16'b1111000000001111;
mse[8] = 16'b1111000000001111;
mse[9] = 16'b1111000000001111;
mse[10] =16'b1111000000001111;
mse[11] =16'b1111000000001111;
mse[12] =16'b1111000000001111;
mse[13] =16'b1111000000001111;
mse[14] =16'b1111111111111111;
mse[15] =16'b1111111111111111;
end

always_ff @(posedge clk) begin
shape_o <= mse[a_o];
end
endmodule


// 1
module one(input logic clk, output logic [15:0] shape_one, input logic [3:0] a_one);

logic [15:0]     mse[15:0];

initial begin
mse[0] = 16'b1111111100000000;
mse[1] = 16'b1111011110000000;
```

```
mse[2] = 16'b1111001111000000;
mse[3] = 16'b1111000111100000;
mse[4] = 16'b1111000011110000;
mse[5] = 16'b1111000001111000;
mse[6] = 16'b1111000000000000;
mse[7] = 16'b1111000000000000;
mse[8] = 16'b1111000000000000;
mse[9] = 16'b1111000000000000;
mse[10] =16'b1111000000000000;
mse[11] =16'b1111000000000000;
mse[12] =16'b1111000000000000;
mse[13] =16'b1111000000000000;
mse[14] =16'b1111000000000000;
mse[15] =16'b1111000000000000;
end

always_ff @(posedge clk) begin
shape_one <= mse[a_one];
end
endmodule


// 2
module two(input logic clk, output logic [15:0] shape_two, input logic [3:0] a_two);

logic [15:0]     mse[15:0];

initial begin
mse[0] = 16'b1111111111111111;
mse[1] = 16'b1111000111111111;
mse[2] = 16'b1111000001111111;
mse[3] = 16'b1111000000111111;
mse[4] = 16'b1111000000011111;
mse[5] = 16'b1111000000001111;
mse[6] = 16'b1111000000000000;
mse[7] = 16'b1111111111111111;
mse[8] = 16'b1111111111111111;
mse[9] = 16'b0000000000001111;
mse[10] =16'b0000000000001111;
mse[11] =16'b0000000000001111;
mse[12] =16'b1111111111111111;
mse[13] =16'b1111111111111111;
mse[14] =16'b1111111111111111;
mse[15] =16'b1111111111111111;
end

always_ff @(posedge clk) begin
shape_two <= mse[a_two];
end
endmodule


// 3
module th(input logic clk, output logic [15:0] shape_th, input logic [3:0] a_th);

logic [15:0]     mse[15:0];
```

```
initial begin
mse[0] = 16'b1111111111111111;
mse[1] = 16'b1111111111111111;
mse[2] = 16'b1111111111111111;
mse[3] = 16'b1111000000000000;
mse[4] = 16'b1111000000000000;
mse[5] = 16'b1111000000000000;
mse[6] = 16'b1111000000000000;
mse[7] = 16'b1111111111111111;
mse[8] = 16'b1111111111111111;
mse[9] = 16'b1111111111111111;
mse[10] =16'b1111000000000000;
mse[11] =16'b1111000000000000;
mse[12] =16'b1111000000000000;
mse[13] =16'b1111111111111111;
mse[14] =16'b1111111111111111;
mse[15] =16'b1111111111111111;
end

always_ff @(posedge clk) begin
shape_th <= mse[a_th];
end
endmodule




// 16 X 8 synchronous RAM with old data read-during-write behavior
module memory(input logic        clk,
                    input logic [9:0]  a,
                    input logic [8:0]  din,
                    input logic              we,
                    output logic [15:0] dout);

//we have 1280 pixels, so 1280 digits coming in each of 16 bits

  logic [15:0]                                    mem [639:0];

  //integer j;
  //integer flag;
  //initial begin

  //for(j = 0; j < 639; j = j+1)

                          //mem[j] = 16'd180;
              //end

//end

  always_ff @(posedge clk) begin
    if (we) mem[a] <= din;
    dout <= mem[a];
  end

endmodule
```

13.3. soc_system_top.sv (Setting up pins for ADC, VGA, and HEX Displays)

```
// ====================================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
// ====================================================================
//
// Modified 2019 by Stephen A. Edwards
//
// Permission:
//
//   Terasic grants permission to use and modify this code for use in
//   synthesis for all Terasic Development Boards and Altera
//   Development Kits made by Terasic.  Other use of this code,
//   including the selling ,duplication, or modification of any
//   portion is strictly prohibited.
//
// Disclaimer:
//
//   This VHDL/Verilog or C/C++ source code is intended as a design
//   reference which illustrates how these types of functions can be
//   implemented.  It is the user's responsibility to verify their
//   design for consistency and functionality through the use of
//   formal verification methods.  Terasic provides no warranty
//   regarding the use or functionality of this code.
//
// ====================================================================
//
//  Terasic Technologies Inc

//  9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070. Taiwan
//
//
//                     web: http://www.terasic.com/
//                     email: support@terasic.com
module soc_system_top(

 ///////// ADC /////////
 inout              ADC_CS_N,
 output       ADC_DIN,
 input              ADC_DOUT,
 output       ADC_SCLK,

 ///////// AUD /////////
 input              AUD_ADCDAT,
 inout              AUD_ADCLRCK,
 inout              AUD_BCLK,
 output       AUD_DACDAT,
 inout              AUD_DACLRCK,
 output       AUD_XCK,

 ///////// CLOCK2 /////////
 input              CLOCK2_50,

 ///////// CLOCK3 /////////
 input              CLOCK3_50,
```

```
//////// CLOCK4 ////////
input              CLOCK4_50,

//////// CLOCK ////////
input              CLOCK_50,

//////// DRAM ////////
output [12:0] DRAM_ADDR,
output [1:0]  DRAM_BA,
output        DRAM_CAS_N,
output        DRAM_CKE,
output        DRAM_CLK,
output        DRAM_CS_N,
inout [15:0]  DRAM_DQ,
output        DRAM_LDQM,
output        DRAM_RAS_N,
output        DRAM_UDQM,
output        DRAM_WE_N,

//////// FAN ////////
output        FAN_CTRL,

//////// FPGA ////////
output        FPGA_I2C_SCLK,
inout             FPGA_I2C_SDAT,

//////// GPIO ////////
inout [35:0]  GPIO_0,
inout [35:0]  GPIO_1,

//////// HEX0 ////////
output [6:0]  HEX0,

//////// HEX1 ////////
output [6:0]  HEX1,

//////// HEX2 ////////
output [6:0]  HEX2,

//////// HEX3 ////////
output [6:0]  HEX3,

//////// HEX4 ////////
output [6:0]  HEX4,

//////// HEX5 ////////
output [6:0]  HEX5,

//////// HPS ////////
inout              HPS_CONV_USB_N,
output [14:0] HPS_DDR3_ADDR,
output [2:0]  HPS_DDR3_BA,
output        HPS_DDR3_CAS_N,
output        HPS_DDR3_CKE,
output        HPS_DDR3_CK_N,
output        HPS_DDR3_CK_P,
```

```
output        HPS_DDR3_CS_N,
output [3:0]  HPS_DDR3_DM,
inout [31:0]  HPS_DDR3_DQ,
inout [3:0]   HPS_DDR3_DQS_N,
inout [3:0]   HPS_DDR3_DQS_P,
output        HPS_DDR3_ODT,
output        HPS_DDR3_RAS_N,
output        HPS_DDR3_RESET_N,
input         HPS_DDR3_RZQ,
output        HPS_DDR3_WE_N,
output        HPS_ENET_GTX_CLK,
inout         HPS_ENET_INT_N,
output        HPS_ENET_MDC,
inout         HPS_ENET_MDIO,
input         HPS_ENET_RX_CLK,
input [3:0]   HPS_ENET_RX_DATA,
input         HPS_ENET_RX_DV,
output [3:0]  HPS_ENET_TX_DATA,
output        HPS_ENET_TX_EN,
inout         HPS_GSENSOR_INT,
inout         HPS_I2C1_SCLK,
inout         HPS_I2C1_SDAT,
inout         HPS_I2C2_SCLK,
inout         HPS_I2C2_SDAT,
inout         HPS_I2C_CONTROL,
inout         HPS_KEY,
inout         HPS_LED,
inout         HPS_LTC_GPIO,
output        HPS_SD_CLK,
inout         HPS_SD_CMD,
inout [3:0]   HPS_SD_DATA,
output        HPS_SPIM_CLK,
input         HPS_SPIM_MISO,
output        HPS_SPIM_MOSI,
inout         HPS_SPIM_SS,
input         HPS_UART_RX,
output        HPS_UART_TX,
input         HPS_USB_CLKOUT,
inout [7:0]   HPS_USB_DATA,
input         HPS_USB_DIR,
input         HPS_USB_NXT,
output        HPS_USB_STP,

///////// IRDA /////////
input         IRDA_RXD,
output        IRDA_TXD,

///////// KEY /////////
input [3:0]   KEY,

///////// LEDR /////////
output [9:0]  LEDR,

///////// PS2 /////////
inout         PS2_CLK,
inout         PS2_CLK2,
inout         PS2_DAT,
```

```
inout              PS2_DAT2,

//////// SW ////////
input [9:0]   SW,

//////// TD ////////
input              TD_CLK27,
input [7:0]   TD_DATA,
input              TD_HS,
output        TD_RESET_N,
input              TD_VS,


//////// VGA ////////
output [7:0]  VGA_B,
output        VGA_BLANK_N,
output        VGA_CLK,
output [7:0]  VGA_G,
output        VGA_HS,
output [7:0]  VGA_R,
output        VGA_SYNC_N,
output        VGA_VS
);

  soc_system soc_system0(
    .clk_clk                    ( CLOCK_50 ),
    .reset_reset_n              ( 1'b1 ),

    .hps_ddr3_mem_a             ( HPS_DDR3_ADDR ),
    .hps_ddr3_mem_ba            ( HPS_DDR3_BA ),
    .hps_ddr3_mem_ck            ( HPS_DDR3_CK_P ),
    .hps_ddr3_mem_ck_n          ( HPS_DDR3_CK_N ),
    .hps_ddr3_mem_cke           ( HPS_DDR3_CKE ),
    .hps_ddr3_mem_cs_n          ( HPS_DDR3_CS_N ),
    .hps_ddr3_mem_ras_n         ( HPS_DDR3_RAS_N ),
    .hps_ddr3_mem_cas_n         ( HPS_DDR3_CAS_N ),
    .hps_ddr3_mem_we_n          ( HPS_DDR3_WE_N ),
    .hps_ddr3_mem_reset_n       ( HPS_DDR3_RESET_N ),
    .hps_ddr3_mem_dq            ( HPS_DDR3_DQ ),
    .hps_ddr3_mem_dqs           ( HPS_DDR3_DQS_P ),
    .hps_ddr3_mem_dqs_n         ( HPS_DDR3_DQS_N ),
    .hps_ddr3_mem_odt           ( HPS_DDR3_ODT ),
    .hps_ddr3_mem_dm            ( HPS_DDR3_DM ),
    .hps_ddr3_oct_rzqin         ( HPS_DDR3_RZQ ),

    .hps_hps_io_emac1_inst_TX_CLK ( HPS_ENET_GTX_CLK ),
    .hps_hps_io_emac1_inst_TXD0   ( HPS_ENET_TX_DATA[0] ),
    .hps_hps_io_emac1_inst_TXD1   ( HPS_ENET_TX_DATA[1] ),
    .hps_hps_io_emac1_inst_TXD2   ( HPS_ENET_TX_DATA[2] ),
    .hps_hps_io_emac1_inst_TXD3   ( HPS_ENET_TX_DATA[3] ),
    .hps_hps_io_emac1_inst_RXD0   ( HPS_ENET_RX_DATA[0] ),
    .hps_hps_io_emac1_inst_MDIO   ( HPS_ENET_MDIO  ),
    .hps_hps_io_emac1_inst_MDC    ( HPS_ENET_MDC   ),
    .hps_hps_io_emac1_inst_RX_CTL ( HPS_ENET_RX_DV ),
    .hps_hps_io_emac1_inst_TX_CTL ( HPS_ENET_TX_EN ),
    .hps_hps_io_emac1_inst_RX_CLK ( HPS_ENET_RX_CLK ),
    .hps_hps_io_emac1_inst_RXD1   ( HPS_ENET_RX_DATA[1]  ),
```

```
.hps_hps_io_emac1_inst_RXD2   ( HPS_ENET_RX_DATA[2]  ),
.hps_hps_io_emac1_inst_RXD3   ( HPS_ENET_RX_DATA[3]  ),

.hps_hps_io_sdio_inst_CMD     ( HPS_SD_CMD          ),
.hps_hps_io_sdio_inst_D0      ( HPS_SD_DATA[0]      ),
.hps_hps_io_sdio_inst_D1      ( HPS_SD_DATA[1]      ),
.hps_hps_io_sdio_inst_CLK     ( HPS_SD_CLK          ),
.hps_hps_io_sdio_inst_D2      ( HPS_SD_DATA[2]      ),
.hps_hps_io_sdio_inst_D3      ( HPS_SD_DATA[3]      ),

.hps_hps_io_usb1_inst_D0      ( HPS_USB_DATA[0]     ),
.hps_hps_io_usb1_inst_D1      ( HPS_USB_DATA[1]     ),
.hps_hps_io_usb1_inst_D2      ( HPS_USB_DATA[2]     ),
.hps_hps_io_usb1_inst_D3      ( HPS_USB_DATA[3]     ),
.hps_hps_io_usb1_inst_D4      ( HPS_USB_DATA[4]     ),
.hps_hps_io_usb1_inst_D5      ( HPS_USB_DATA[5]     ),
.hps_hps_io_usb1_inst_D6      ( HPS_USB_DATA[6]     ),
.hps_hps_io_usb1_inst_D7      ( HPS_USB_DATA[7]     ),
.hps_hps_io_usb1_inst_CLK     ( HPS_USB_CLKOUT      ),
.hps_hps_io_usb1_inst_STP     ( HPS_USB_STP         ),
.hps_hps_io_usb1_inst_DIR     ( HPS_USB_DIR         ),
.hps_hps_io_usb1_inst_NXT     ( HPS_USB_NXT         ),

.hps_hps_io_spim1_inst_CLK    ( HPS_SPIM_CLK  ),
.hps_hps_io_spim1_inst_MOSI   ( HPS_SPIM_MOSI ),
.hps_hps_io_spim1_inst_MISO   ( HPS_SPIM_MISO ),
.hps_hps_io_spim1_inst_SS0    ( HPS_SPIM_SS   ),

.hps_hps_io_uart0_inst_RX     ( HPS_UART_RX     ),
.hps_hps_io_uart0_inst_TX     ( HPS_UART_TX     ),

.hps_hps_io_i2c0_inst_SDA     ( HPS_I2C1_SDAT     ),
.hps_hps_io_i2c0_inst_SCL     ( HPS_I2C1_SCLK     ),

.hps_hps_io_i2c1_inst_SDA     ( HPS_I2C2_SDAT     ),
.hps_hps_io_i2c1_inst_SCL     ( HPS_I2C2_SCLK     ),

.hps_hps_io_gpio_inst_GPIO09  ( HPS_CONV_USB_N ),
.hps_hps_io_gpio_inst_GPIO35  ( HPS_ENET_INT_N ),
.hps_hps_io_gpio_inst_GPIO40  ( HPS_LTC_GPIO ),

.hps_hps_io_gpio_inst_GPIO48  ( HPS_I2C_CONTROL ),
.hps_hps_io_gpio_inst_GPIO53  ( HPS_LED ),
.hps_hps_io_gpio_inst_GPIO54  ( HPS_KEY ),
.hps_hps_io_gpio_inst_GPIO61  ( HPS_GSENSOR_INT ),

.vga_r (VGA_R),
.vga_g (VGA_G),
.vga_b (VGA_B),
.vga_clk (VGA_CLK),
.vga_hs (VGA_HS),
.vga_vs (VGA_VS),
.vga_blank_n (VGA_BLANK_N),
.vga_sync_n (VGA_SYNC_N),

.adc_cs                 (ADC_CS_N),
.adc_sclk               (ADC_SCLK),
```

```
    .adc_din                (ADC_DIN),
    .adc_dout               (ADC_DOUT),

                .hex_h0                                                 (HEX0),
                .hex_h1                                                 (HEX1),
                .hex_h2                                                 (HEX2),
                .hex_h3                                                 (HEX3),
                .hex_h4                                                 (HEX4),
                .hex_h5                                                 (HEX5)

  );

  // The following quiet the "no driver" warnings for output
  // pins and should be removed if you use any of these peripherals
/*
  assign ADC_CS_N = SW[1] ? SW[0] : 1'bZ;
  assign ADC_DIN = SW[0];
  assign ADC_SCLK = SW[0];
*/
  assign AUD_ADCLRCK = SW[1] ? SW[0] : 1'bZ;
  assign AUD_BCLK = SW[1] ? SW[0] : 1'bZ;
  assign AUD_DACDAT = SW[0];
  assign AUD_DACLRCK = SW[1] ? SW[0] : 1'bZ;
  assign AUD_XCK = SW[0];

  assign DRAM_ADDR = { 13{ SW[0] } };
  assign DRAM_BA = { 2{ SW[0] } };
  assign DRAM_DQ = SW[1] ? { 16{ SW[0] } } : 16'bZ;
  assign {DRAM_CAS_N, DRAM_CKE, DRAM_CLK, DRAM_CS_N,
          DRAM_LDQM, DRAM_RAS_N, DRAM_UDQM, DRAM_WE_N} = { 8{SW[0]} };

  assign FAN_CTRL = SW[0];

  assign FPGA_I2C_SCLK = SW[0];
  assign FPGA_I2C_SDAT = SW[1] ? SW[0] : 1'bZ;

  assign GPIO_0 = SW[1] ? { 36{ SW[0] } } : 36'bZ;
  assign GPIO_1 = SW[1] ? { 36{ SW[0] } } : 36'bZ;
/*
  assign HEX0 = { 7{ SW[1] } };
  assign HEX1 = { 7{ SW[2] } };
  assign HEX2 = { 7{ SW[3] } };
  assign HEX3 = { 7{ SW[4] } };
  assign HEX4 = { 7{ SW[5] } };
  assign HEX5 = { 7{ SW[6] } };
*/
  assign IRDA_TXD = SW[0];

  assign LEDR = { 10{SW[7]} };

  assign PS2_CLK = SW[1] ? SW[0] : 1'bZ;
  assign PS2_CLK2 = SW[1] ? SW[0] : 1'bZ;
  assign PS2_DAT = SW[1] ? SW[0] : 1'bZ;
  assign PS2_DAT2 = SW[1] ? SW[0] : 1'bZ;

  assign TD_RESET_N = SW[0];
```

```
endmodule
```

13.4. mouse.c (Software to run the mouse and communicate with hardware)

```
/*
 * Userspace program that communicates with the vga_ball device driver
 * through ioctls
 *
 * Stephen A. Edwards
 * Columbia University
 */

#include <stdio.h>
#include "vga_ball.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>

#include <stdlib.h>
#include <arpa/inet.h>
#include "usbmouse.h"

/* References on libusb 1.0 and the USB HID/mouse protocol
 * https://nxmnpg.lemoda.net/3/libusb_interrupt_transfer
 * http://libusb.org
 * http://www.dreamincode.net/forums/topic/148707-introduction-to-using-libusb-10/
 * http://www.usb.org/developers/devclass_docs/HID1_11.pdf
 */

int vga_ball_fd;

// for mouse
struct libusb_device_handle *mouse;
uint8_t endpoint_address;

/* Read and print the background color
void print_background_color() {
  vga_ball_arg_t vla;

  if (ioctl(vga_ball_fd, VGA_BALL_READ_BACKGROUND, &vla)) {
      perror("ioctl(VGA_BALL_READ_BACKGROUND) failed");
      return;
  }
  printf("%02x %02x %02x\n",
                vla.background.red, vla.background.green, vla.background.blue);
}

/* Set the background color
void set_background_color(const vga_ball_color_t *c,unsigned short xcoord, unsigned short ycoord)
{
  vga_ball_arg_t vla;
```

```
 vla.x = xcoord;
 vla.y = ycoord;
 vla.background = *c;
 if (ioctl(vga_ball_fd, VGA_BALL_WRITE_BACKGROUND, &vla)) {
    perror("ioctl(VGA_BALL_SET_BACKGROUND) failed");
    return;
 }
}*/
void print_coordinate_info() {
 vga_ball_arg_t vla;

            if (ioctl(vga_ball_fd, VGA_BALL_READ_COORD, &vla)) {
                    perror("ioctl(VGA_BALl_READ_COORD) failed");
                    return;
            }
            //printf("(%d, %d)", vla.x, vla.y);
 printf("\n");
}

//Write the coordinates to thc: In function 'main':
//mouse.c:166:11: error: 'mouse display
void write_coordinates(vga_ball_arg_t* c)
{
            vga_ball_arg_t vla;
            vla = *c;
            //printf("HERE(%d, %d)", vla.x, vla.y);
            printf("HERE(%d, %d, %d,%d)", vla.x, vla.y,vla.r,vla.t);
            if (ioctl(vga_ball_fd, VGA_BALL_WRITE_COORD, &vla)) {
                    perror("ioctl(VGA_BALL_WRITE_COORD) failed");
                    return;
            }
}

int main()
{

 vga_ball_arg_t vla;
 //----------------------MOUSE_START-------------------------
  // struct sockaddr_in serv_addr;
 int px = 320;
 int py = 240;
 int numx, numy;
 int modifierss = 0;
 struct usb_mouse_packet packet;
 int transferred;

  //button_1 is horizontal_sweep
 int pos_button_1_x = 500;
 int pos_button_1_y = 350;

 //button_2 is trigger_voltage
 int pos_button_2_x = 500;
 int pos_button_2_y = 425;

 int inputx = 320;
 int inputy = 240;
 int inputclick = 0;
```

```
 int x_distance = 50;
 int y_distance = 75;
 int x_width = 16;
 int y_width = 16;

   //save trigger_voltage and horizontal sweep value
 int trigger_voltage = 2; // default 2, range(1.0 to 3.0)
 int sweep_value = 2;   // default us 1, range is (1~100)
 int trigger_slope = 1;
 //the logic is drop all data except every sweep_value sample.
 char str[50] = "without mouse click";

static const char filename[] = "/dev/vga_ball";
 // char keystate[12];

 /* Open the mouse */
 if ( (mouse = openmouse(&endpoint_address)) == NULL ) {
   fprintf(stderr, "Did not find a mouse\n");
   exit(1);
 }

 if ( (vga_ball_fd = open(filename, O_RDWR)) == -1) {
   fprintf(stderr, "could not open %s\n", filename);
   return -1;
 }
 for (;;)
 {
   libusb_interrupt_transfer(mouse, endpoint_address,
         (unsigned char *) &packet, sizeof(packet),
         &transferred, 0);
  //c: In function 'main':
//mouse.c:166:11: error: 'mous // printf("%d\n", flg1);

   if (transferred == sizeof(packet)) {
     if (packet.pos_x > 0x88) {
       numx = -(0xFF - packet.pos_x + 1);
     }
     else { numx = packet.pos_x;}

     if (packet.pos_y > 0x88) {
       numy = -(0xFF - packet.pos_y + 1);
     }
     else { numy = packet.pos_y;}

     if (px < 1) { px = 1;}
     else if (px > 0 && px < 640) { px = px + numx; }
     else if (px > 639) { px = 639;}
     else {px = 320;}

     if (py < 1) { py = 1;}
     else if (py > 0 && py < 480) { py = py + numy; }
     else if (py > 479) { py = 479;}
     else {py = 240;}

     inputx = px;
     inputy = py;
```

```
    inputclick = packet.modifiers;
    modifierss = packet.modifiers;

    if (packet.modifiers == 1){
//              printf("flag1");
     if (pos_button_1_y<inputy && inputy<(pos_button_1_y + x_width)){
//               printf("flag2");
      if ( pos_button_1_x<inputx && inputx<(pos_button_1_x + x_width) && trigger_voltage > 0){
//                printf("flag3");
        trigger_voltage = trigger_voltage - 1;
        str[50] = "click add trigger_voltage";}
      else if ( (pos_button_1_x + x_distance)<inputx && inputx<(pos_button_1_x + (x_distance+x_width)) &&
trigger_voltage < 3){
        trigger_voltage = trigger_voltage + 1;
        str[50] = "click add trigger_voltage";}
       // else {continue;}
     }
     else if ((pos_button_1_y+y_distance)<inputy && inputy<(pos_button_1_y+(y_distance+y_width))){
      if ( pos_button_1_x<inputx && inputx<(pos_button_1_x + x_width)){
        // sweep_value = sweep_value *2; str = "click button sweep_value x2";}
        trigger_slope = 0;
        str[50] = "click button trigger_slope minus";}
      else if ( (pos_button_1_x + x_distance)<inputx &&
        inputx<(pos_button_1_x + (x_distance+x_width))){
        // sweep_value = sweep_value /2; str = "click button sweep_value /2";}
        trigger_slope = 1;
        str[50] = "click button trigger_slope pos";}
       // else {continue;}
      }
      printf("trigger_voltage: %d, trigger_slope: %d, the button state: %s", trigger_voltage,trigger_slope,str);
    }
            //ADDING R and t
            vla.r =trigger_slope;
            vla.t = trigger_voltage;
    vla.x = px;
    vla.y = py;
    printf("  position of x, y are: %d %d; left click is %d\n",px,py,modifierss);
    write_coordinates(&vla);
    //usleep(400000);
    }
   }
}
 //----------------------mouse_END----------------------

/*
 int flag = 0;
 int flag2 =0;
 int a =0;
// int i;
 static const char filename[] = "/dev/vga_ball";*/

 //static const vga_ball_color_t colors[] = {
  // { 0xff, 0x00, 0x00 }, /* Red */
  // { 0x00, 0xff, 0x00 }, /* Green */
 //  { 0x00, 0x00, 0xff }, /* Blue */
  // { 0xff, 0xff, 0x00 }, /* Yellow */
  // { 0x00, 0xff, 0xff }, /* Cyan */
```

```
 //  { 0xff, 0x00, 0xff }, /* Magenta */
 //  { 0x80, 0x80, 0x80 }, /* Gray */
  // { 0x00, 0x00, 0x00 }, /* Black */
   //{ 0xff, 0xff, 0xff }  /* White */
 //};
/*
vla.x = px;
 vla.y = py;
# define COLORS 9

 printf("VGA ball Userspace program started\n");

 if ( (vga_ball_fd = open(filename, O_RDWR)) == -1) {
   fprintf(stderr, "could not open %s\n", filename);
   return -1;
 }

 printf("initial state: ");
// print_background_color();
 print_coordinate_info();
 write_coordinates(&vla);
   printf("initial state: ");
// print_background_color();
 print_coordinate_info();

 while(1) {
  // set_background_color(&colors[i % COLORS ],600,200);
   //print_background_color();

                        if (flag ==0){
                        vla.x = vla.x + 60;
                        }


                        /*if (flag2==0){
                        vla.y = vla.y+ 20;
                        }
                        else
                        {
                        vla.y = vla.y -20;
                        }


                        if(vla.x > 1250)
                        {
                        vla.x =30;
                        }

                        /*
                        if(vla.y > 465)
                        {
                        flag2 = 1;
                        }
                        if(vla.y <16)
                        {
                        flag2 = 0;
                        }
```

```
                        //vla.x= 180;
                        vla.y= 180;

                        //printf("XandY(%d, %d)", vla.x, vla.y);
                        print_coordinate_info();
                        write_coordinates(&vla);
                        a =a+1;
                        printf("a:%d",a);

                        usleep(400000);

                        //vla.x= 120;
                        vla.y= 120;

                        //printf("XandY(%d, %d)", vla.x, vla.y);
                        print_coordinate_info();
                        write_coordinates(&vla);
                        usleep(400000);
                        a=a+1;
                        printf("a:%d",a);

 }

 printf("VGA BALL Userspace program terminating\n");
 return 0;
}*/
```

13.5. vga_ball.h (Header file for the mouse)

```
#ifndef _VGA_BALL_H
#define _VGA_BALL_H

#include <linux/ioctl.h>

/*typedef struct {
        unsigned char red, green, blue;
} vga_ball_color_t;
 */

/*typedef struct {

 vga_ball_color_t background;
} vga_ball_arg_t;

typedef struct {
unsigned short x, y;

} vga_ball_arg_t;*/

typedef struct {
 unsigned short x, y, r,t;
 //vga_ball_color_t background;
} vga_ball_arg_t;

#define VGA_BALL_MAGIC 'q'
```

```
/* ioctls and their arguments */
//#define VGA_BALL_WRITE_BACKGROUND _IOW(VGA_BALL_MAGIC, 1, vga_ball_arg_t *)
//#define VGA_BALL_READ_BACKGROUND  _IOR(VGA_BALL_MAGIC, 2, vga_ball_arg_t *)
#define VGA_BALL_WRITE_COORD _IOW('q', 1, vga_ball_arg_t *)
#define VGA_BALL_READ_COORD  _IOR('q', 2, vga_ball_arg_t *)

#endif
```