UNIVERSITY OF LEEDS

ELEC5566M
Mini Project

DE1-SoC Oscilloscope

Alexander Bolton - 200938078
Haider Shafiq - 201207577

May 2019

Submitted in accordance with the requirements for the degree of
Master of Science in Embedded Systems Engineering

The University of Leeds
School of Electronic and Electrical Engineering

Contents

1 Introduction

This report shall discuss the group project for the FGPA Design for System-on-Chip module. The group consisted of Alexander Bolton, and Haider Shafiq. The project chosen was to design and develop a multi-channel oscilloscope on the DE1-SoC. The aim of this project was to take multiple input signals from a signal generator, and display them accurately on a separate monitor, via the VGA port of the DE1-SoC.

The report shall cover the various parts of this project, and how they were then tested, as well as why they were needed. In chapter 2 the design of the VGA driver will be discussed, as well as the challenges presented with its development. The VGA port was used to output the waves to a separate monitor such that they can then be analysed. Chapter 3 shall discuss the development of the ADC module. The ADC used was the one on-board the DE1-SoC. This ADC has 10 pins, a ground pin a voltage pin, and 8 pins which can act as separate channels. The data from the ADC shall be displayed on the VGA, and they will be waves will be seen on the monitor. Multiples signals will be given to the ADC via a signal generator, and the waves will be displayed on a separate monitor, via the VGA.

In chapter 4 the design of the seven-segment display will be discussed. The seven-segment display will provide the user with the information of the waves on the screen, such as the voltage of the wave. This will be measured by the cursors on the screen which will be discussed in chapter 5. The controls for the system including two cursors in the x-axis and two cursors in the y-axis. These cursors will be controlled by the keys on the DE1-SoC board, and which cursor is being moved at which time shall be controlled by the switches on the board. The controls shall also acts as enable switches displaying, whether the cursors are being controlled, or whether the waves on screen are. The controls chapter will also discuss the design of the slower clock module, and the purpose of it.

Chapter 6 will discuss the design of the measurement's module. The measurements module was the module responsible for calculating the wave information such as the voltage of each wave. This information was then displayed on the seven-segment display. Chapter 7 will discuss any problems encountered while completing this project, such as problems with the LT24 LCD, or problems with the Terasic ADDA ADC board.

This report shall conclude in chapter 8 with an analysis of the overall work completed in the project, as well as any further work which could be undertaken as part of this project. This further work could be in the from of an expansion to the current work already completed, or any modifications to the current project which could improve it. All the code will be put in the appendices at the end of this report, including the code for any test benches.

2 VGA

For this project it was decided to make the wave as visible as possible it would be the most beneficial to use the Video Graphics Array (VGA) connector to connect the FPGA to a monitor. The VGA comprises of 3 buses of 8 bits for colour (Red, Green, Blue), a clock signal, a horizontal sync signal (HSync), and finally a vertical sync signal (VSync) as shown in Figure 1. The resolution chosen was 800x600 at a frequency of 75Hz. To carry out his resolution is required a clock of 49MHz (50MHz clock was used direct from the FPGA). All the timing specifications were found in the DE1-SoC manual (See Appendix A).

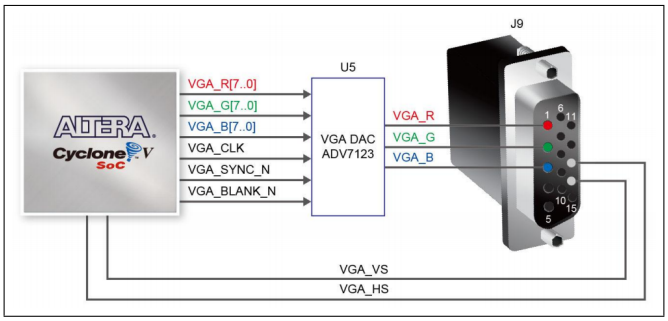

Figure 1: VGA Pins [1]

The VGA signal is made up of 4 parts which are the Sync Signal, Back porch, Display Interval, and Front porch as shown in Figure 2. Both HSync and VSync use the same layout however the VSync signal uses the HSync positive edge as a clock signal.

The horizontal part of the signal sets the sync to low for 1.6us. The sync signal is then set high and the back porch blanking period starts for 3.2us. At this point the colour signals (RGB) start. Each of the positive edge of the 49MHz clock is equivalent to 1 pixel. After 800 clock signals (around 16.2us) the RGB signal ends and the front porch of the signal begins. After the front porch the sync signal once again is low which indicates a horizontal line is complete.

Once the HSync signal has completed 600 iterations the vertical part of the signal begins. The vertical part of the signal is similar however there is no RGB signal included in this part of the signal and it uses the HSync as a clock signal for a number of "lines". Once this has been completed it indicates that a whole frame of image is complete.

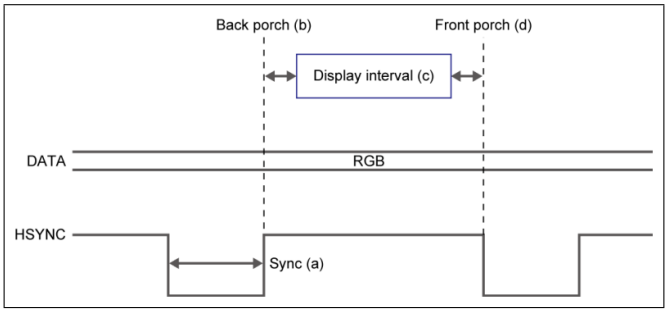

Figure 2: VGA Timings [1]

Initially for the IP block the library was created to carry out the VSync and HSync signals which was written by the project group. However, there was a 2ns delay which could not be

solved. A number of methods were attempted to correct this 2ns but created a problem with the HSync signal would be 2ns early or 2ns late which was enough to effect the VSync signal that it could not be displayed on a monitor. Due to timing constraints the decision was made to find and implement a pre-made IP and modify it to work on the DE1-SoC board.[2] The HSync and VSync parts of the IP was used to create the signals. A counter was created to count how many x (Clock signal within display interval) iterations and y (HSync signal) iterations have been completed. The x counter resets on the positive edge of the HSync and the y counter resets on the positive edge of the VSync signal. The IP was then modified to use a full 8 bit colour range available on the DE1-SoC.

```
1   always @(posedge clk)
2   begin
3    if (blank) begin //if blanking period
4      if(hsync) begin //if hsync then reset counter
5      x <= 0; //reset
6      end else if (vsync) begin //if vsync then reset counter
7      x <= 0; //reset
8      end
```

Figure 3: X Counter with reset

```
1   always @(posedge hsync) begin
2     if(vsync) begin
3       y <= 0;
4     end else begin
5       y <= y + 1;
6     end
7   end
8   //Assigns for red green and blue colours out. It also checks if it is in a blanking period.
9    assign red_out  = (blank) ? 0 : pixel_R;
10  assign green_out = (blank) ? 0 : pixel_G;
11  assign blue_out  = (blank) ? 0 : pixel_B;
```

Figure 4: Y Counter with reset

The grid was displayed by having an if statement in an always block which checked if x counter or y counter was equal to where a line wanted to be placed. When the counter equalled these values the RGB data signal was set to white. The wave was displayed by passing the x counter through the block to the top layer. This then went to the sampler which had a sample buffer. This data was passed back to the VGA IP which then the value is compared with the y counter. If it is equal to the y counter then the pixel is set to the desired colour. Offsets were added to the waves so it can be controlled

3 ADC

To allow signals to be captured so they can be sampled the onboard analogue-to-digital converter (ADC) was decided to be used. It was decided after having difficulty getting the TerasIC ADDA board to work, so decided to stay within the time constraints to use the onboard ADC. To initialise this; QSys was used to generate the appropriate code. The onboard ADC has 4 pins connected to the FPGA. The clock, data in, data out, and CS_n. QSys allows you to generate the code for these pins in a graphical user interface. It also allows you to select clock speed to update at and the channels you wish to use. The ADC pins allow up to 8 channels of ADC to be used so for expandability all channels where selected to be used. The code was generated, and then 2 .v files was copied into the main project and instantiated. The channels where labeled from CH0 to CH7. For this project only CH0 and CH1 were used and the outputs of these were inputted into the sampler. This then samples the values of 800 clock cycles (to match the screen width). This is then ready to be output to the VGA IP.

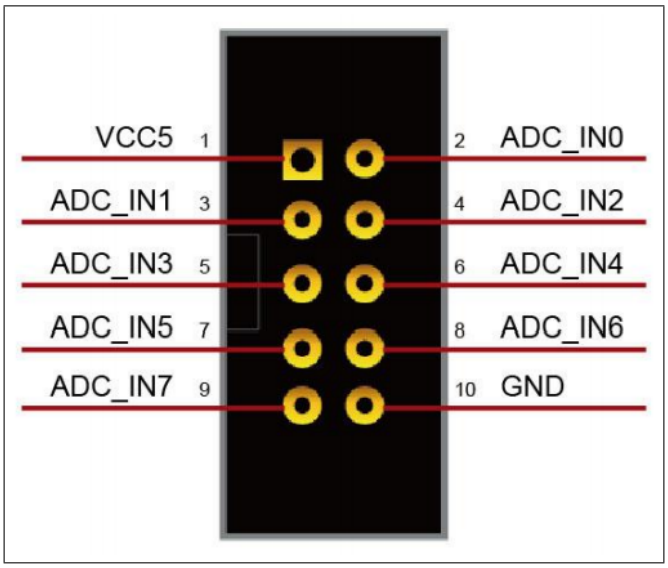

Figure 5: ADC Pins [1]

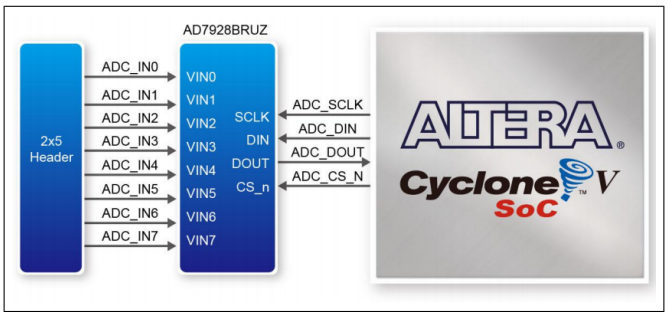

Figure 6: ADC Connections to FPGA [1]

4 Seven Segment Display

For measurements to be displayed it was decided for simplicity to use the seven segment display. Each seven segment display is made up of 6 pins which control each part of the seven segments. A limitation was found which did not allow a pin to control the decimal point which was required for our measurements.

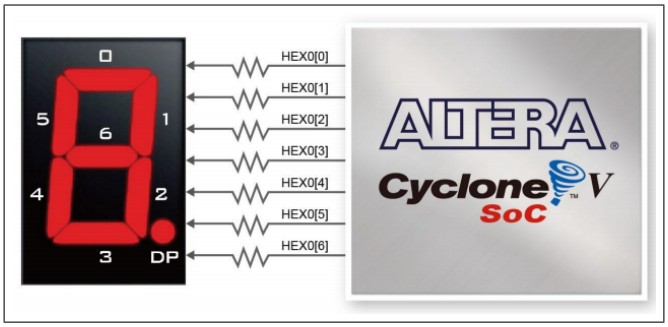

Figure 7: Seven Segment Display Pins [1]

It was very trivial to setup the seven segment display to output a signal number. To do this a register of 7 bits was initialised which could act for each bit of a single seven segment display. It had 2 inputs (clock and number) and then the output for the seven segment display. When a number was entered between 0 to 9, an if statement in an always block was run to give the appropriate output. It was noted that the output is inverted as the seven segment display is active low. This can be seen in Appendix C.

The next challenge was to have numbers between 0 to 9999 to be separated into singular numbers to be output to the seven segment display. On FPGA division is a very expensive resource. It was decided to use a counter which the idea was taken from online. [3] The counter method uses 5 counters, ones, tens, hundreds, thousands, and counter. As the counters count up the ones gets to the number 9 it will reset the ones counter and increase the tens counter. This is the same for the tens, hundreds, and thousands. When the counter reaches the number it then stops and the output number is sent to the seven segment output generator module which turns the numbers into an output for the seven segment displays. This runs at full clock speed so the counting cannot be seen with the human eye. This code can be found in Appendix B.

5 Controls

This chapter of the report shall discuss the design of the control IP for the project. The purpose of this module was to enable and control both the cursors, and the waves, such that they appeared on the monitor. They could then be controlled using the keys and the switches, to move, increase/decrease the volts per division, as well as the time per division. This chapter will also discuss the button clock, also known as the slClock.

Due to the limited number of buttons and switches on the DE1-SoC board, various states were used to decide what was being controlled. These states were controlled by switches 9 and 8. State 1 was the default state when switches 9 and 8 were 0, this state controlled the cursors. State 2 happened when switch 8 was 1 and switch 9 was 0, this state controlled the waves. The final stage was stage 3, and this happened when both switch 9 and 8 were 1, and this displayed the test wave. The purpose of the test wave, was such that while the ADC module was being designed the controls module could still be worked, as well as the VGA module could be tested in hardware. There is a spare state left, which could be used for further expansion in the future. The states and functions shall be described in more detail below.

The following table below details state 1, when switch 9 and 8 are both 0. S7 – 0 represents switch 7 to 0, and B3 – 0 represents buttons 3 – 0.

Buttons or Switches	Purpose
S7	N/A
S6	N/A
S5	N/A
S4	Selects wave to be measured
S3	Allow the y-cursors to be moved
S2	Allow the x-cursors to be moved
S1	Enable y-cursors when 1, so that they display on the monitor.
S0	Enable x-cursors when 1, so that they display on the monitor.
B3	When S2 is 1 – Move cursor x1 right. When S3 is 1 – Move cursor y1 down. When both S2 and S3 is 1 – Move both x-cursors right.
B2	When S2 is 1 – Move cursor x1 left. When S3 is 1 – Move cursor y1 up. When both S2 and S3 is 1 – Move both x-cursors left.
B1	When S2 is 1 – Move cursor x2 right. When S3 is 1 – Move cursor y2 down. When both S2 and S3 is 1 – Move both y-cursors down.
B0	When S2 is 1 – Move cursor x2 left. When S3 is 1 – Move cursor y2 up. When both S2 and S3 is 1 – Move both y-cursors up.

Table 1: State 1 functions

During state 1 if switch 2 and switch 3 are 0, the buttons do nothing. While in state 1 switches 7 to 4 also do nothing. The table below shows state 2, when switch 9 is 0, and switch 8 is 1.

Buttons or Switches	Purpose
S7	N/A
S6	N/A
S5	Adjust the time/division for both waves.
S4	Allows a snapshot in time to be taking of both waves, similar to the run/stop feature on oscilloscopes.
S3	Adjust the volts/division for both waves.
S2	Allows both waves to be moved.
S1	Enables wave 2 when 1, so that it displays on the monitor.
S0	Enables wave 1 when 1, so that it displays on the monitor.
B3	When S2 is 1 – Move wave 1 down. When S3 is 1 – Increase volts/div for wave 1. When S4 is 1 – Freeze wave 1. When S5 is 1 – Increase time/div for wave 1.
B2	When S2 is 1 – Move wave 1 up. When S3 is 1 – Decrease volts/div for wave 1. When S4 is 1 – Run wave 1. When S5 is 1 – Decrease time/div for wave 1.
B1	When S2 is 1 – Move wave 2 down. When S3 is 1 – Increase volts/div for wave 2. When S4 is 1 – Freeze wave 2. When S5 is 1 – Increase time/div for wave 2.
B0	When S2 is 1 – Move wave 2 up. When S3 is 1 – Decrease volts/div for wave 2. When S4 is 1 – Run wave 2. When S5 is 1 – Decrease time/div for wave 2.

Table 2: State 2 functions

Similar to state 1, the buttons will do nothing if S2 – 5 are all 0. State 3, which occurs when both switch 8 and 9 are 1, enables the test wave to display on the monitor. All cursors and waves move by a size of 1, this is indicated by the localparam moveSize. The code for the controls IP can be seen in appendix P. All the controls happen on the positive edge of buttonClock. In appendix L, it can be seen that the buttonClock attaches to the slClock [19]. The code for the slClock can be seen in appendix Q. The slClock module was created so that the cursors and waves could be moved accurately. They could not be moved with any precision when running off of the main clock as the main clock ran at 50MHz. This meant that when the cursors or waves moved, they would move to fast to control. However, using the slClock [19] meant that the frequency was approximately 93Hz, and as such was much easier to control.

6 Measurement

This chapter of the report shall discuss the design of the measurement IP for the project. The purpose of this module was to be able to measure the voltage of the waveform on the monitor, using the y-cursors. This module would then display the voltage to the seven-segment display, in mV.

This module was designed to take the cursors from the control's module, and measure the difference between them, and display the value on the seven-segment display. This was designed so that the cursors can be moved to either of the waves on the monitor and measure the voltage between the peaks of the waveforms. Depending on what was being measured between the cursors the, value between them would display on the seven-segment display. Therefore, if the user wanted to measure wave 2, the cursors would be placed between the top peak, and bottom peak of the wave, and the voltage in mV, would then display on the seven-segment display. This is also how the user would measure the other wave on the screen.

The equation to calculate the voltage of the waves is:

$$VoltageX = ShiftDownX^2 * DiffX \tag{1}$$

Where X indicated which wave was being calculated i.e. shiftdownX was actually shiftdown1 or shiftdown2. Diffx was the difference between the cursors. The shiftdown function increases or decreases the volts/division for each wave. There are two functions as each wave may be shifted by different values.

The code for this module can be seen in appendix R. From the code it can be seen that the value of distance between the cursors is put into the variables vx1 or vx2. This output of these variables were then stored in the variable result, depending on the value of waveSel, which was then assigned to num. When the measurement IP module was instantiated into the top-level module, the local parameter num was attached to a local parameter in the top-level module also called num. This parameter controlled what values were displayed on the seven-segment display.

The waveSel register was designed such that when waveSel is equal to 0, wave 1 is being measured, and the result of the measurement is displayed on the seven-segment display. If waveSel is equal to 2, wave 2 is measured. The waveSel function is controlled in the control IP, when in state 1, if switch 4 is 0, waveSel is 0, and as such wave 1 is being measured. When switch 4 is 1, waveSel is 1, and as such wave 2 is measured.

7 Problems Encountered

This chapter will discuss the various problems encountered during this project. It also will discuss the decisions made in overcoming these problems.

When trying to work with the TerasIC ADDA, the IP was written in specification with the documentation. Unfortunately we could not get the ADDA to update and give an output to the sampler block, so it was abandoned. The replacement was using the onboard ADC which has a reduced resolution but will still meet our goals.

When first creating the VGA IP the VSync signals and HSync signals where coded by the project group, however it was quickly found that there was an error with the timing by a significant amount. This was corrected by replacing using always blocks to only use always blocks as counters, and using assigning if statements to set the outputs. This then fixed the timing issue which was off by only 2ns in the HSync. Since the VSync worked off of the HSync as its clock it created an issue as the slight timing added up per line (unit of positive clock edge of HSync) putting off the VSync clock by a significant amount. Values were adjusted to try to alleviate this issue, however the closest the clock could be to the actual value was 2ns. The decision was made to find a VGA IP which was precoded, and port it over to the DE1-SoC to save on time. The HSync and VSync signals were acquired from this IP and counters added.[2] From here it was simple to produce the grid and waves.

Initially the controls were to be done using the LT24 LCD. This module would utilise the touchscreen capabilities of the LT24, and would allow the user to use the display to decide what would display on the monitor, i.e. waves, cursors. The initial stage of this was to draw a user interface to the display, such that the user could interact with the display and choose which option was desired. However, when drawing to the screen difficulties were found in controlling what appeared on the screen. Instead random patterns would appear, which were unable to be manipulated into what was desired. After being unable to fix these, and taking into consideration time constraints, a decision was made to use the switches and keys control the system.

8 Conclusion

This chapter will conclude the report, as well as analyse the work completed in this project, against the original aims. It will also discuss the possibility of further work on the project, if more time were had to complete it. The original aims of the project were; to be able to take to signals from a signal generator, present them to the FPGA, and be able to view and measure them. This has been completed as multiple signals from the FPGA can be seen on the separate monitor using the VGA. Using the cursors, the waves can also be accurately measured, via use of the switches and keys. The value of the waves can be seen on the seven-segment display, in mV.

Given more time the project could be further extended. Currently, only 2 signals can be taken from the signal generator, and get measured by the DE1-SoC, and be displayed on the monitor. To expand this, all 8 channels of the ADC could be used. Also given more time, the LT24 touchscreen module would be fully developed and used to control various aspects of the project. These aspects would be the cursors, as well as enabling them, the waves, and also adjusting the volts/division as well as the time/division. Due to not having the touchscreen working, all the functions are on the switches. These switches can be set to toggles on the buttons, however, this was not done originally as it was not planned to have as many controls for the project. Further to these additions, the time/division which also be measured using the other cursors on the DE1-SoC scope, if more time were had, as well as a function to adjust the trigger threshold. Currently the trigger threshold has to be set manually within the code, every time it is to be changed the program has to be re-compiled and re-programmed onto the board.

In conclusion, while the original aims of the project were met, there is room to expand the project. This expansion could make the project more user friendly, and would also give the user the ability to measure more signals using the DE1-SoC scope.

9 Appendix

9.1 Appendix A - VGA Timing Specifications

VGA mode		Horizontal Timing Spec				
Configuration	Resolution(HxV)	a(us)	b(us)	c(us)	d(us)	Pixel clock(MHz)
VGA(60Hz)	640x480	3.8	1.9	25.4	0.6	25
VGA(85Hz)	640x480	1.6	2.2	17.8	1.6	36
SVGA(60Hz)	800x600	3.2	2.2	20	1	40
SVGA(75Hz)	800x600	1.6	3.2	16.2	0.3	49
SVGA(85Hz)	800x600	1.1	2.7	14.2	0.6	56
XGA(60Hz)	1024x768	2.1	2.5	15.8	0.4	65
XGA(70Hz)	1024x768	1.8	1.9	13.7	0.3	75
XGA(85Hz)	1024x768	1.0	2.2	10.8	0.5	95
1280x1024(60Hz)	1280x1024	1.0	2.3	11.9	0.4	108

Table 3: VGA Horizontal Timing Specification [1]

VGA mode		Vertical Timing Spec				
Configuration	Resolution(HxV)	a(lines)	b(lines)	c(lines)	d(lines)	Pixel clock(MHz)
VGA(60Hz)	640x480	2	33	480	10	25
VGA(85Hz)	640x480	3	25	480	1	36
SVGA(60Hz)	800x600	4	23	600	1	40
SVGA(75Hz)	800x600	3	21	600	1	49
SVGA(85Hz)	800x600	3	27	600	1	56
XGA(60Hz)	1024x768	6	29	768	3	65
XGA(70Hz)	1024x768	6	29	768	3	75
XGA(85Hz)	1024x768	3	36	768	1	95
1280x1024(60Hz)	1280x1024	3	38	1024	1	108

Table 4: VGA Vertical Timing Specification [1]

9.2 Appendix B - Number Split Module

```
  output [3:0] thousandsOut //thousands out
);
  //registers
  reg    [13:0] counter = 0;
  reg    [3:0]  ones = 0;
  reg    [3:0]  tens = 0;
  reg    [3:0]  hundreds = 0;
  reg    [3:0]  thousands = 0;
  reg done = 0;
  //assigns
  assign doneOut = done;
  assign onesOut = ones;
  assign tensOut = tens;
  assign hundredsOut = hundreds;
  assign thousandsOut = thousands;

  always @(posedge clock) begin
    //if start then set all values to 0
     if(start) begin
        counter <= 0;
        done <= 0;
        ones <= 0;
        tens <= 0;
        hundreds <= 0;
        thousands<= 0;
    //else then check if the counter is equal to number if so then set register done to signal completed
     end else if(counter == mynumber) begin
        done <= 1;
    //if not complete then count up to the number but keep a track of single digits
     end else if (!done) begin
        counter <= counter + 1;
        ones <= ones == 9 ? 0 : ones + 1;
        if(ones == 9) begin
          tens <= tens == 9 ? 0 : tens + 1;
          if(tens == 9) begin
             hundreds <= hundreds == 9 ? 0 : hundreds + 1;
             if(hundreds == 9) begin
                thousands <= thousands + 1;
             end
          end
        end
     end
  end
endmodule

module generateSevenSegOutput(
```

Figure 8: Number Split Module

9.3 Appendix C - Generate Seven Segment Output Module

```
input [3:0] number,
output [6:0] segOutput
);

reg [6:0] seg;

assign segOutput = seg;

always @(posedge clock) begin
  if(number == 0) begin
    seg <= ~(7'h3F);
  end else if(number == 1) begin
    seg <= ~(7'h06);
  end else if(number == 2) begin
    seg <= ~(7'h5B);
  end else if(number == 3) begin
    seg <= ~(7'h4F);
  end else if(number == 4) begin
    seg <= ~(7'h66);
  end else if(number == 5) begin
    seg <= ~(7'h6D);
  end else if(number == 6) begin
    seg <= ~(7'h7D);
  end else if(number == 7) begin
    seg <= ~(7'h07);
  end else if(number == 8) begin
    seg <= ~(7'h7F);
  end else if(number == 9) begin
    seg <= ~(7'h67);
  end
```

Figure 9: Generate Seven Segment Output Module

9.4 Appendix D - Top Level RTL View

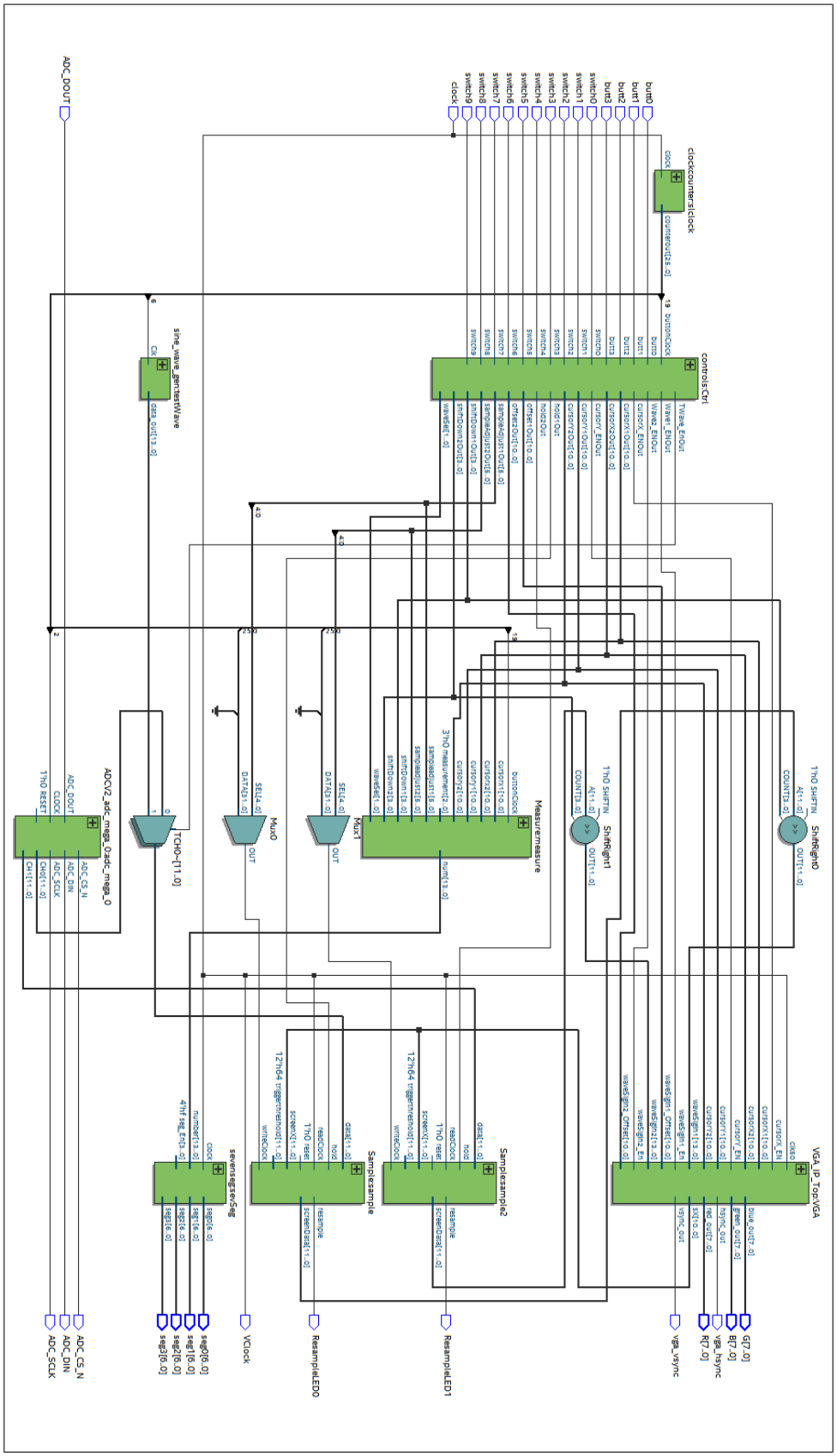

Figure 10: Top Level RTL View

9.5 Appendix E - Slower Clock RTL View

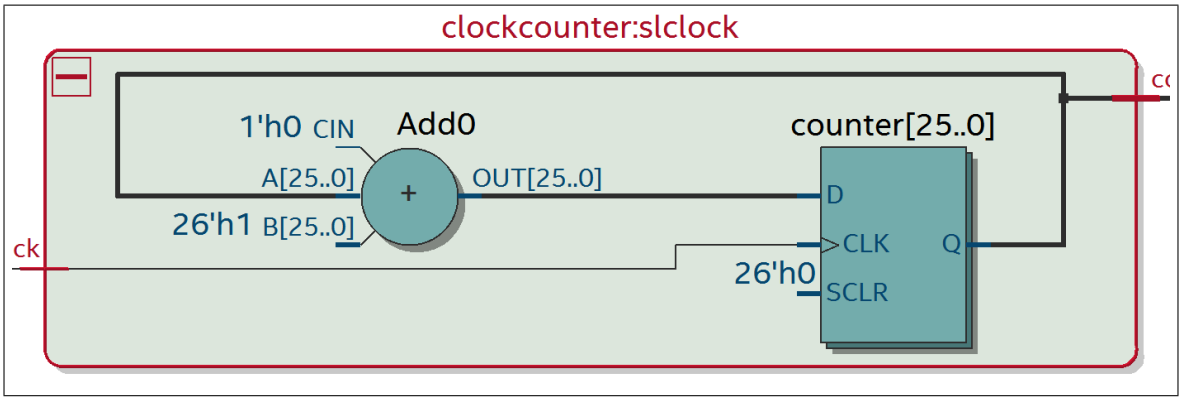

Figure 11: Slower Clock RTL View

9.6 Appendix F - VGA Top Level RTL View

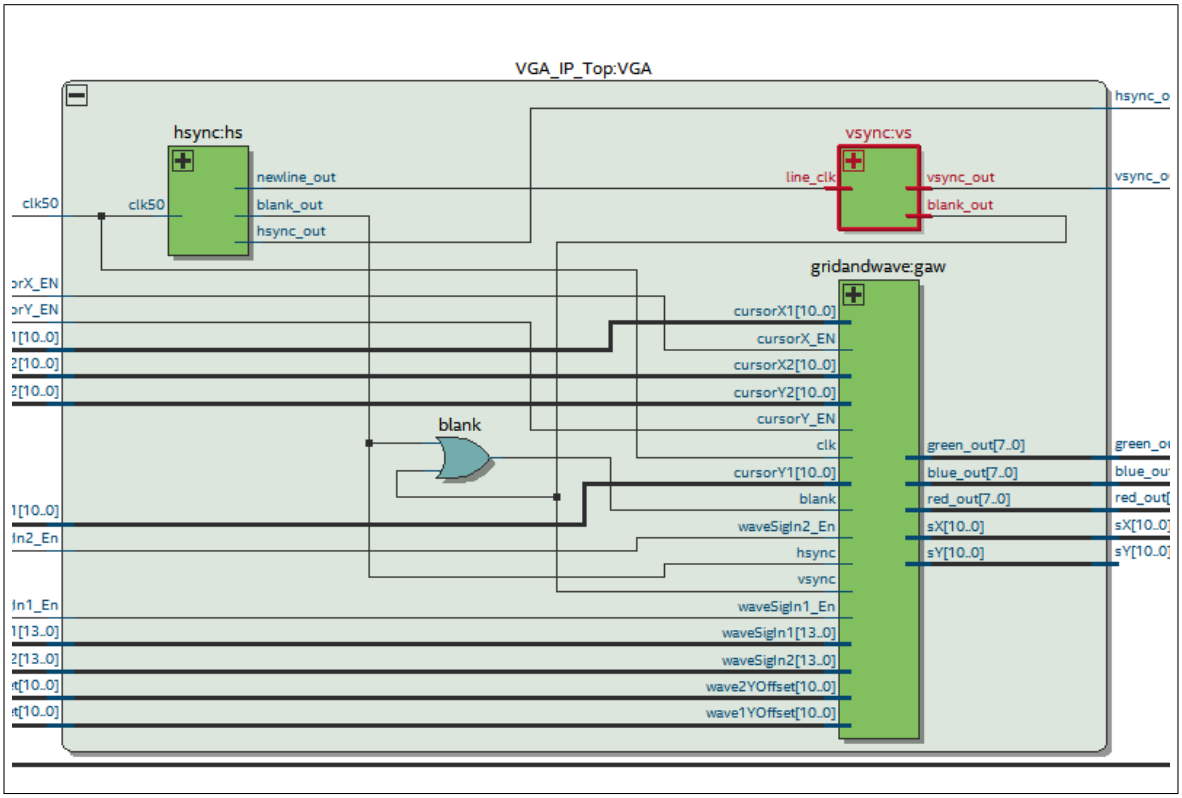

Figure 12: VGA IPRTL View

9.7 Appendix G - VGA HSync RTL View

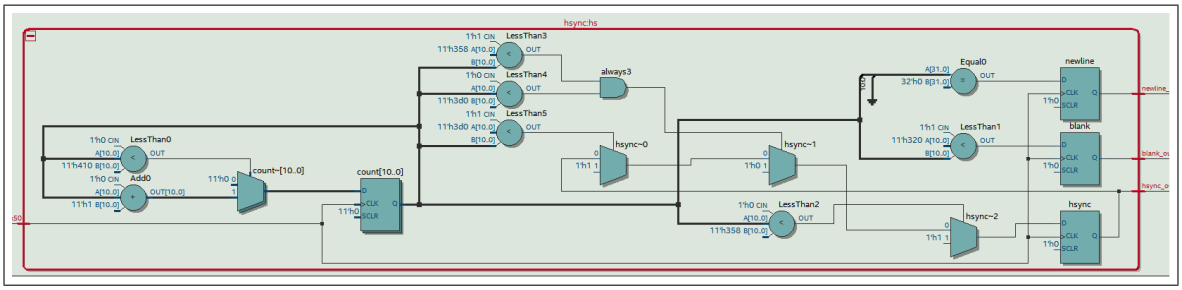

Figure 13: VGA HSync RTL View

9.8 Appendix H - VGA VSync RTL View

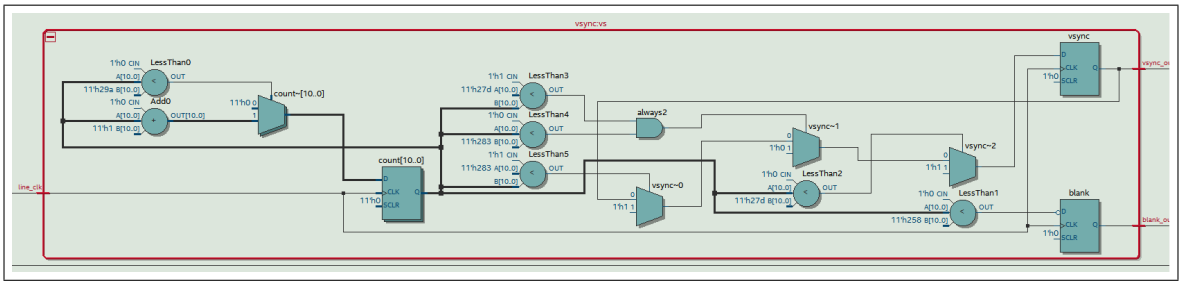

Figure 14: VGA VSync RTL View

9.9 Appendix I - Measurement RTL View

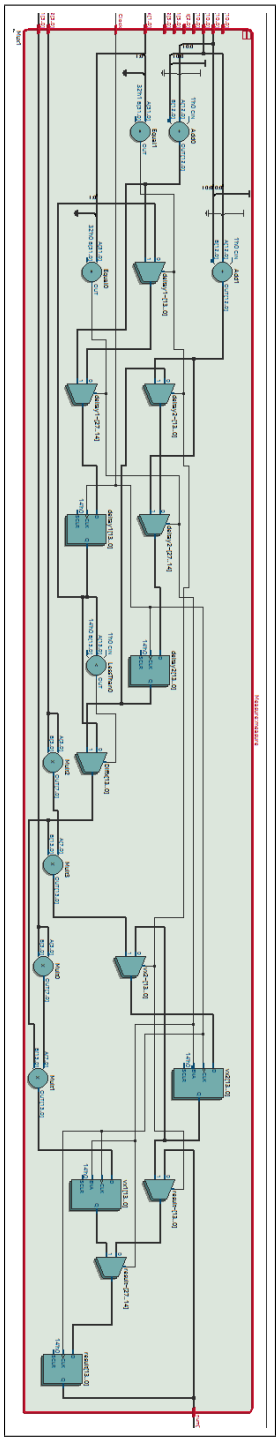

Figure 15: Measurement RTL View

9.10 Appendix J - Seven Segment Display RTL View

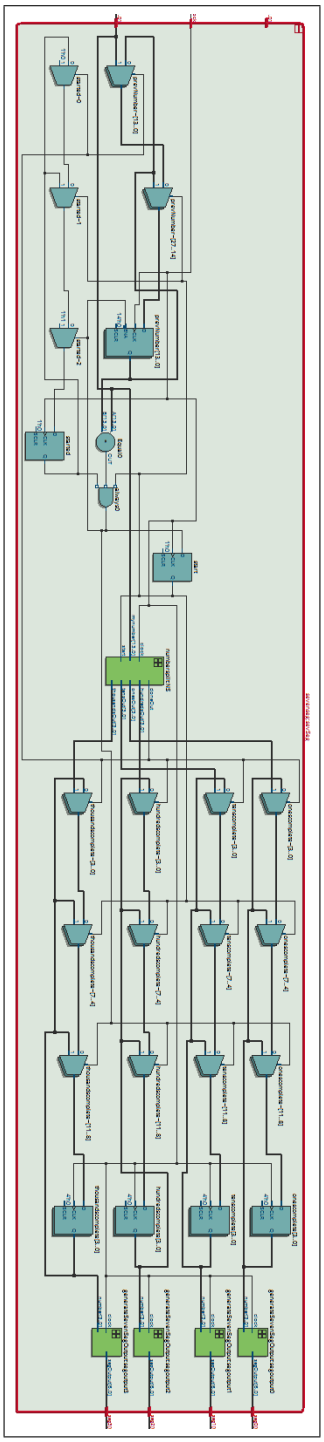

Figure 16: Seven Segment Display RTL View

9.11 Appendix K - Sampler RTL View

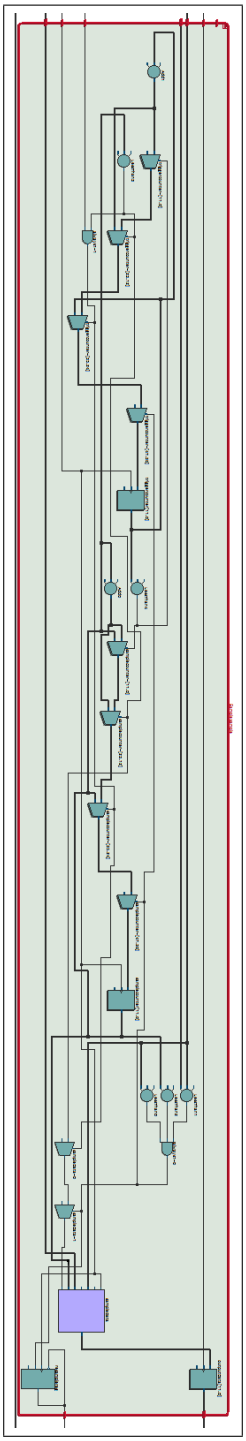

Figure 17: Sampler RTL View

9.12 Appendix L - Top Level Module Code

```
/* Top level module for FPGA_MiniProject
Alexander Bolton - 200938078
Haider Shafiq - 201207577
N-Channel Oscilloscope */

// Top Level Module
module FPGA_MiniProject(
input          clock, //50MHz Clock
//Controls
input          switch0, //Cursor X En
input          switch1,  //Cursor Y En
input          switch2, //Signal 1 En
input          switch3,  //Signal 2 En
input          switch4, //Cursor Y set
input          switch5, //Cursor X Set
input          switch6, //Wave 1 Shift /Squish
input          switch7, //Wave 2 Shift/Squish
input          switch8, //Wave 1 Clock
input          switch9, //Wave 2 Clock
input          butt0,  //x1/y1 left/up
input          butt1,    //x1/y1 right/down
input          butt2,  //x2/y2 left/up
input          butt3,  //x2/y2 right/down
//ADC Onboard
output         ADC_CS_N,
output         ADC_SCLK, //clock
output         ADC_DIN, //Data in
input          ADC_DOUT, //Data out
//VGA
output         vga_hsync, //VGA HSync signal out
output         vga_vsync, //VGA VSync signal out
output  [7:0]   R, //VGA Red Colour bits out
output  [7:0]   G, //VGA Red Colour bits out
output  [7:0]   B, //VGA Red Colour bits out
output         VClock, //VGA Clock out
output         ResampleLED0, //Resample LED
output         ResampleLED1, //Resample LED
//seven seg
output  [6:0]   seg0, //Seven Seg Display 0 Output
output  [6:0]   seg1, //Seven Seg Display 1 Output
output  [6:0]   seg2, //Seven Seg Display 2 Output
output  [6:0]   seg3 //Seven Seg Display 3 Output
);
// Various wires to enable waves/cursors/clocks
wire [11:0] testwave; //test wave output
wire Wave1_EN;
wire Wave2_EN;
wire cursorX_EN;
wire cursorY_EN;
wire TWave_EN;
wire [25:0] slClock;
wire [10:0] offset1;
wire [10:0] offset2;
wire [13:0] num;
wire hold1;
wire hold2;
//Wires to hold the position of cursors
wire [10:0] cursorY1;
wire [10:0] cursorY2;
wire [10:0] cursorX1;
wire [10:0] cursorX2;
//Wired for increasing/decreasing volts/div
wire [3:0] shiftDown1;
wire [3:0] shiftDown2;
//Wires for getting more/less samples on the screen
wire [5:0] sampleAdjust1;
wire [5:0] sampleAdjust2;
wire [11:0] TCH0;
//Following is to have the sample adjust changing on the slower clock (Counter)
wire sampleWriteClock1;
wire sampleWriteClock2;
assign sampleWriteClock1 = slClock[sampleAdjust1];
assign sampleWriteClock2 = slClock[sampleAdjust2];
//Assigning the VGA Clock to the normal Clock (50MHz)
assign VClock = clock;
//Assigning which wave should occupy channel 0, either the actual wave or the test wave
assign TCH0 = (TWave_EN == 1) ? testwave : CH0;
//Wave wires
wire [11:0] waveSigIn1;
wire [11:0] waveSigIn2;
wire [11:0] sampledwave1;
wire [11:0] sampledwave2;
//VGA IP Wires
wire [10:0] sX;
wire [10:0] sY;
```

Figure 18: FPGA_MiniProject.v Part 1

```
//To programatically change down shifts
assign waveSigIn1 = (sampledwave1 >> shiftDown1); // Squish
assign waveSigIn2 = (sampledwave2 >> shiftDown2); //needs to change to wave sample 2 sampledwave1
//Wires for each of the channels, can potertially have an 8 channel scope (2 wires for 5v and ground)
//For the purposes of the demo we have 2 channels
wire [11:0] CH0;
wire [11:0] CH1;
wire [11:0] CH2;
wire [11:0] CH3;
wire [11:0] CH4;
wire [11:0] CH5;
wire [11:0] CH6;
wire [11:0] CH7;
wire [1:0] waveSel;
//Instatiating the Test Sine Wave module
//Used a test sine wave so we could work on controls, while we were working on the ADC
sine_wave_gen testWave(
  .Clk (slClock[6]),
  .data_out (testwave)
  );
//Instatiating the VGA module
VGA_IP_Top VGA(
  .clk50    (clock), //50 MHz Clock
  .cursorX_EN (cursorX_EN), //Cursor X Enable
  .cursorY_EN (cursorY_EN), //Cursor Y Enable
  .cursorY1   (cursorY1), //Cursor Y1 Position
  .cursorY2   (cursorY2), //Cursor Y2 Position
  .cursorX1   (cursorX1), //Cursor X1 Position
  .cursorX2   (cursorX2), //Cursor X2 Position
  .waveSigIn1 (waveSigIn1), //Wave signal in for channel 0
  .waveSigIn2 (waveSigIn2), //Wave signal in for channel 1
  .waveSigIn1_En (Wave1_EN), //Channel 0 Enable
  .waveSigIn2_En (Wave2_EN), //Channel 1 Enable
  .hsync_out  (vga_hsync), //HSync out for VGA
  .vsync_out  (vga_vsync), //VSync out for VGA
  .red_out  (R), //VGA Colour Out
  .blue_out   (G), //VGA Colour Out
  .green_out  (B), //VGA Colour Out
  .waveSigIn1_Offset (offset1), //Channel 0 Offset
  .waveSigIn2_Offset (offset2), //Channel 1 Offset
  .sX (sX), //Counter X value
  .sY (sY) //Counter Y value
);
//Instatiating the 1st sample module for channel0
Sample sample(
  .readClock (clock), //50MHz Clock to read as fast as possible
  .writeClock (sampleWriteClock1), //Sample write clock to vary sampling rate
  .hold (hold1), //Hold
  .data (TCH0), //Date in
  .screenX (sX), //Counter X in from VGA IP
  .reset (0), //reset
  .screenData (sampledwave1), //screenData out
  .resample(ResampleLED0), //Resample LED indicator
  .triggerthreshold(100) //trigger threshold
);
//Instatiating the 1st sample module for channel0
Sample sample2(
  .readClock (clock), //50MHz Clock to read as fast as possible
  .writeClock (sampleWriteClock2), //Sample write clock to vary sampling rate
  .hold (hold2), //Hold
  .data (CH1), //Date in
  .screenX (sX), //Counter X in from VGA IP
  .reset (0), //reset
  .screenData (sampledwave2), //screenData out
  .resample(ResampleLED1), //Resample LED indicator
  .triggerthreshold(100) //trigger threshold
);
//Instatiating the 7seg module
sevenseg sevSeg(
  .clock (clock), //50MHz clock
  .seg_En (4'b1111), //Seven Segment Enable
  .number (num), //4 digit number in
  .seg0 (seg0), //Seven Segment Display out 0
  .seg1 (seg1), //Seven Segment Display out 1
  .seg2 (seg2), //Seven Segment Display out 2
  .seg3 (seg3) //Seven Segment Display out 3
);
//Instatiating the slower clock module
clockcounter slclock(
  .clock (clock), //50MHz Clock in
  .counterout (slClock) //Counter reg out
);
//Instatiating the Scope controls module
controls Ctrl(
  .switch0 (switch0), //Cursor X En
```

Figure 19: FPGA_MiniProject.v Part 2

```
  .switch1 (switch1), //Cursor Y En
  .switch2 (switch2), //Signal 1 En
  .switch3 (switch3), //Signal 2 En
  .switch4 (switch4), //Cursor Y set
  .switch5 (switch5), //Cursor X Set
  .switch6 (switch6), //Wave 1 Shift /Squish
  .switch7 (switch7), //Wave 2 Shift/Squish
  .switch8 (switch8), //Wave 1 Clock
  .switch9 (switch9), //Wave 2 Clock
  .butt0 (butt0), //x1/y1 left/up
  .butt1 (butt1),   //x1/y1 right/down
  .butt2 (butt2), //x2/y2 left/up
  .butt3 (butt3), //x2/y2 right/down
  .buttonClock (slClock[19]), //Slower clock to move cursors/wave
  .hold1Out (hold1), //Freeze wave 1
  .hold2Out (hold2), //Freeze wave 2
  .cursorY1Out (cursorY1), //Cursor y1
  .cursorY2Out (cursorY2), //Cursor y2
  .cursorX1Out (cursorX1), //Cursor x1
  .cursorX2Out (cursorX2), //Cursor x2
  .shiftDown1Out (shiftDown1), //Squish wave 1 down
  .shiftDown2Out (shiftDown2), //Squish wave 2 down
  .sampleAdjust1Out (sampleAdjust1), //Adjust wave 1 to have more samples
  .sampleAdjust2Out (sampleAdjust2), //Adjust wave 2 to have more samples
  .cursorX_ENOut (cursorX_EN), //Enable x-cursors
  .cursorY_ENOut (cursorY_EN), //Enable y-cursors
  .Wave1_ENOut (Wave1_EN), //Enable wave 1
  .Wave2_ENOut (Wave2_EN), //Enable wave 1
  .offset1Out (offset1), //Offset for wave 1
  .offset2Out (offset2), //Offset for wave 2
  .TWave_EnOut (TWave_EN), //Enable Test wave
  .waveSel (waveSel) //Select wave to be measured
);
//Instatiating the Scope measurements module
Measure measure(
  .buttonClock (slClock[19]), //Slower clock to measure distance between cursors
  .cursory1 (cursorY1), //Cursor y1
  .cursory2 (cursorY2), //Cursor y2
  .cursorx1 (cursorX1), //Cursor x1
  .cursorx2 (cursorX2), //Cursor x2
  .sampleadjust1 (sampleAdjust1), //Adjust wave 1 to have more samples
  .sampleadjust2 (sampleAdjust2), //Adjust wave 2 to have more samples
  .shiftDown1 (shiftDown1), //Squish wave 1 down
  .shiftDown2 (shiftDown2), //Squish wave 2 down
  .waveSel (waveSel), //Select wave to be measured
  .measurement (0), //Selelct Cursors - on x permenantly
  .num (num) //Num to be displayed on the seven-seg
);
//Generated ADC module using QSys
ADCV2_adc_mega_0 #(
  .board          ("DE1-SoC"),
  .board_rev      ("Autodetect"),
  .tsclk          (13),
  .numch          (7),
  .max10pllmultby (1),
  .max10plldivby  (1)
) adc_mega_0 (
  .CLOCK    (slClock[2]),    //               clk.clk
  .RESET    (0),    //             reset.reset
  .CH0      (CH0),      //           readings.export
  .CH1      (CH1),      //                   .export
  .CH2      (CH2),      //                   .export
  .CH3      (CH3),      //                   .export
  .CH4      (CH4),      //                   .export
  .CH5      (CH5),      //                   .export
  .CH6      (CH6),      //                   .export
  .CH7      (CH7),      //                   .export
  .ADC_SCLK (ADC_SCLK), // external_interface.export
  .ADC_CS_N (ADC_CS_N), //                   .export
  .ADC_DOUT (ADC_DOUT), //                   .export
  .ADC_DIN  (ADC_DIN)   //                   .export
);

endmodule
```

Figure 20: FPGA_MiniProject.v Part 3

9.13 Appendix M - VGA Top Level Module Code

```
//800x600 VGA IP - Alexander Bolton
module VGA_IP_Top(
input clk50, //60 MHz Clock
input cursorX_EN, //Cursor X Enable
input cursorY_EN, //Cursor Y Enable
input [10:0] cursorY1, //Cursor Y1 Position
input [10:0] cursorY2, //Cursor Y2 Position
input [10:0] cursorX1, //Cursor X1 Position
input [10:0] cursorX2, //Cursor X2 Position
input [10:0] waveSigIn1_Offset, //Wave signal 0 offset
input [10:0] waveSigIn2_Offset, //Wave signal 1 offset
input [13:0] waveSigIn1, //Wave signal in for channel 0
input [13:0] waveSigIn2, //Wave signal in for channel 1
input waveSigIn1_En, //Wave signal enable for channel 0
input waveSigIn2_En, //Wave signal enable for channel 1
output hsync_out, //HSync out to monitor
output vsync_out, //VSync out to monitor
output [7:0] red_out, //RGB Out (Red)
output [7:0] blue_out, //RGB Out (Blue)
output [7:0] green_out, //RGB Out (Green)
output [10:0] sX, //Counter X Out
output [10:0] sY //Counter Y Out
);
  wire line_clk, blank, hblank, vblank;
  assign blank = hblank || vblank;
  //Instantiate HSync
  hsync hs(
  .clk50 (clk50), //50MHz clock
  .hsync_out (hsync_out), //HSync output
  .blank_out (hblank), //Blanking period out
  .newline_out (line_clk) //new line out
  );
  //Instantiate HSync
  vsync vs(
  .line_clk (line_clk), //Line clock in (goes to new line out)
  .vsync_out (vsync_out), //vsync output
  .blank_out (vblank) //Blanking period out
  );

  gridandwave gaw(
  .clk (clk50), //50 MHz Clock
  .blank (blank), //Blanking Period
  .red_out (red_out), //Red Out
  .green_out (green_out), //Green Out
  .blue_out (blue_out), //Blue Out
  .hsync (hblank), //HSync in
  .vsync (vblank), //VSync in
  .waveSigIn1 (waveSigIn1), //Wave signal in for channel 0
  .waveSigIn2 (waveSigIn2), //Wave signal in for channel 1
  .waveSigIn1_En (waveSigIn1_En), //Wave signal enable for channel 0
  .waveSigIn2_En (waveSigIn2_En), //Wave signal enable for channel 1
  .cursorX_EN (cursorX_EN), //Cursor X Enable
  .cursorY_EN (cursorY_EN), //Cursor Y Enable
  .cursorY1 (cursorY1), //Cursor Y1 Position
  .cursorY2 (cursorY2), //Cursor Y2 Position
  .cursorX1 (cursorX1), //Cursor X1 Position
  .cursorX2 (cursorX2), //Cursor X2 Position
  .wave1YOffset (waveSigIn1_Offset), //Wave signal 1 offset
  .wave2YOffset (waveSigIn2_Offset), //Wave Signal 2 offset
  .sX (sX), //Output of x counter
  .sY (sY) //output of y counter
  );


endmodule
```

Figure 21: VGA_IP_Top.v

```
//VSync Clock written by https://github.com/mstump/verilog-vga-controller/  - Mike Stump
//This module controls the VSync signal
module vsync(
  input line_clk, //Line Clock in
  output vsync_out, //VSync out to monitor
  output blank_out //Blank period out
  );

   reg [10:0] count = 10'b0000000000; //counter set to 0
   reg vsync  = 0; //vsync reg to 0
   reg blank  = 0; //blanking period reg to 0

  //at posedge line clock count up
   always @(posedge line_clk)
   if (count < 666)
     count <= count + 1;
   else
     count <= 0;
   //at posedge line clock if count is above 600 blanking period on
   always @(posedge line_clk)
   if (count < 600)
     blank    <= 0;
   else
     blank    <= 1;
   //at posedge line clock if count is between values then vsync is on
   always @(posedge line_clk)
   begin
    if (count < 637)
      vsync   <= 1;
    else if (count >= 637 && count < 643)
      vsync   <= 0;
    else if (count >= 643)
      vsync   <= 1;
   end
  //assign to outputs
   assign vsync_out  = vsync;
   assign blank_out  = blank;

endmodule // hsync
//HSync Clock written by https://github.com/mstump/verilog-vga-controller/  - Mike Stump
//This module controls the HSync signal
module hsync(
  input clk50, //50MHz clock in
  output hsync_out, //HSync out to monitor
  output blank_out, //blanking period out
  output newline_out //newline clock out
  );
   //counter set to 0
   reg [10:0] count = 10'b0000000000;
  //hSync, blank, and new line regs to 0
   reg hsync  = 0;
   reg blank  = 0;
   reg newline  = 0;
  //at posedge 50MHz clock count up when below 1040 else reset.
   always @(posedge clk50)
   begin
    if (count < 1040)
      count  <= count + 1;
    else
      count  <= 0;
   end
   //at posedge 50MHz clock if count is at 0 then newline clock to 1
   always @(posedge clk50)
   begin
    if (count == 0)
      newline <= 1;
    else
      newline <= 0;
   end
  //at posedge 50MHz clock if count is above or equal to 800 then blanking period on
   always @(posedge clk50)
   begin
    if (count >= 800)
      blank  <= 1;
    else
      blank  <= 0;
   end
  //Control hsync clock
   always @(posedge clk50)
   begin
    if (count < 856) // pixel data plus front porch
      hsync <= 1;
    else if (count >= 856 && count < 976)
      hsync <= 0;
```

Figure 22: VGA_IP.v Part 1

```
    else if (count >= 976)
      hsync <= 1;
  end // always @ (posedge clk50)

  //Assign outputs
   assign hsync_out    = hsync;
   assign blank_out    = blank;
   assign newline_out  = newline;

endmodule
//This module controls the pixels of the measurement grid and waves
module gridandwave(
  input clk, //50MHz clock
  input blank, //Blanking period in
  input hsync, //HSync in
  input vsync, //VSync in
  input cursorX_EN, //Cursor Enables
  input cursorY_EN, //Cursor Enables
  input [10:0] cursorY1, //CursorY1 Position
  input [10:0] cursorY2, //CursorY2 Position
  input [10:0] cursorX1, //CursorX1 Position
  input [10:0] cursorX2, //CursorX2 Position
  input [13:0] waveSigIn1, //Channel 0 in
  input [13:0] waveSigIn2, //Channel 1 in
  input [10:0] wave1YOffset, //Channel 0 offset in
  input [10:0] wave2YOffset,  //Channel 1 offset in
  input waveSigIn1_En, //Channel 0 enable
  input waveSigIn2_En, //Channel 1 enable
  output [7:0] red_out,  //red colour output to monitor
  output [7:0] green_out, //green colour output to monitor
  output [7:0] blue_out, //blue colour output to monito
  output [10:0] sX, //Counter x Out
  output [10:0] sY //Counter y Out
  );
  localparam gridoffset = 20; //Offset for grid on x axis
  reg [19:0] x; //Register for x counter
  reg [19:0] y; //Register for y counter
   reg [7:0] pixel_R = 0; //Register for red pixel colour
  reg [7:0] pixel_G = 0; //Register for green pixel colour
  reg [7:0] pixel_B = 0; //Register for blue pixel colour

  //Assigns for counter outputs
  assign sX = x;
  assign sY = y;
   always @(posedge clk)
   begin
    if (blank) begin //if blanking period
      if(hsync) begin //if hsync then reset counter
      x <= 0; //reset
      end else if (vsync) begin //if vsync then reset counter
      x <= 0; //reset
      end
    end else begin
      x <= x+1; //count up
      //wave code - if wave enabled and y is equal to the wave signal plus offset
      if (waveSigIn1_En && y == (waveSigIn1 + wave1YOffset)) begin
      pixel_R <= 8'b00000000; //set colours
      pixel_G <= 8'b11111111; //set colours
      pixel_B <= 8'b11111111; //set colours
      end else if (waveSigIn2_En && (y == waveSigIn2 + wave2YOffset)) begin
      pixel_R <= 8'b11111111; //set colours
      pixel_G <= 8'b00000000; //set colours
      pixel_B <= 8'b11111111; //set colours
      //cursor code - If cursor enabled and the x or y value is equal to the specific cursor value
      end else if(cursorX_EN && x == cursorX1) begin
      pixel_R <= 8'b11111111; //set colours
      pixel_G <= 8'b11111111; //set colours
      pixel_B <= 8'b00000000; //set colours
      end else if (cursorX_EN && x == cursorX2) begin
      pixel_R <= 8'b11111111; //set colours
      pixel_G <= 8'b11111111; //set colours
      pixel_B <= 8'b00000000; //set colours
      end else if (cursorY_EN && y == cursorY1) begin
      pixel_R <= 8'b00000000; //set colours
      pixel_G <= 8'b11111111; //set colours
      pixel_B <= 8'b00000000; //set colours
      end else if (cursorY_EN && y == cursorY2) begin
      pixel_R <= 8'b00000000; //set colours
      pixel_G <= 8'b11111111; //set colours
      pixel_B <= 8'b00000000; //set colours
      //Y grid code - This is a very inefficient peice of code which draws the grid.
      end else if(y == 60 * 1) begin
      pixel_R <= 8'b11111111; //set colours
      pixel_G <= 8'b11111111; //set colours
      pixel_B <= 8'b11111111; //set colours
      end else if(y == 60 * 2) begin
```

Figure 23: VGA_IP.v Part 2

```
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(y == 60 * 3) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(y == 60 * 4) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(y == 60 * 5) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(y == 60 * 6) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(y == 60 * 7) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(y == 60 * 8) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(y == 60 * 9) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        //X grid code - This is a very inefficient peice of code which draws the grid.
        end else if(x == (60 * 1) - gridoffset) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(x == (60 * 2) - gridoffset) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(x == (60 * 3) - gridoffset) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(x == (60 * 4) - gridoffset) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(x == (60 * 5) - gridoffset) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(x == (60 * 6) - gridoffset) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(x == (60 * 7) - gridoffset) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(x == (60 * 8) - gridoffset) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(x == (60 * 9) - gridoffset) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(x == (60 * 10) - gridoffset) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(x == (60 * 11) - gridoffset) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(x == (60 * 12) - gridoffset) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(x == (60 * 13) - gridoffset) begin
        pixel_R <= 8'b11111111; //set colours
        pixel_G <= 8'b11111111; //set colours
        pixel_B <= 8'b11111111; //set colours
        end else if(x == (60 * 14) - gridoffset) begin
```

Figure 24: VGA_IP.v Part 3

```
      pixel_R <= 8'b11111111; //set colours
      pixel_G <= 8'b11111111; //set colours
      pixel_B <= 8'b11111111; //set colours
      end else begin //else set the background to black
      pixel_R <= 8'b0; //set colours to black
      pixel_G <= 8'b0; //set colours to black
      pixel_B <= 8'b0; //set colours to black
      end
    end
   end
  //HSync counter for y counter
  always @(posedge hsync) begin
    if(vsync) begin
      y <= 0;
    end else begin
      y <= y + 1;
    end
  end
  //Assigns for red green and blue colours out. It also checks if it is in a blanking period.
   assign red_out  = (blank) ? 0 : pixel_R;
  assign green_out = (blank) ? 0 : pixel_G;
  assign blue_out  = (blank) ? 0 : pixel_B;

endmodule // color
```

Figure 25: VGA_IP.v Part 4

9.14 Appendix N - Seven Segment Display Code

```
module sevenseg(
input clock, //50MHz Clock
input [3:0] seg_En, //7 Seg Enables
input [13:0] number, //Number of 4 digits
output [6:0] seg0, //Seven Segment Display 0 output
output [6:0] seg1, //Seven Segment Display 1 output
output [6:0] seg2, //Seven Segment Display 2 output
output [6:0] seg3 //Seven Segment Display 3 output
);
//Wires
wire done;
wire [3:0] ones; //Wires value in ones
wire [3:0] tens; //Wires value in tens
wire [3:0] hundreds; //Wires value in hundreds
wire [3:0] thousands; //Wire value in thousands
//Registers
reg start = 0; //if numbersplit starts
reg started = 0; //if numbersplit is currently ongoing
reg [13:0] prevNumber = 0; //previous number
reg numberCountto = 0; //number
reg [3:0] onescomplete; //ones register
reg [3:0] tenscomplete; //tens register
reg [3:0] hundredscomplete; //hundreds register
reg [3:0] thousandscomplete; //thousands register

//Number split module instantiation
numbersplit NS(
.clock (clock),
.mynumber (number),
.start (start),
.doneOut(done),
.onesOut (ones),
.tensOut (tens),
.hundredsOut (hundreds),
.thousandsOut (thousands)
);
//Seven Seg Display outputs
generateSevenSegOutput segoutput0(
.clock (clock),
.number (onescomplete),
.segOutput (seg0)
);

generateSevenSegOutput segoutput1(
.clock (clock),
.number (tenscomplete),
.segOutput (seg1)
);

generateSevenSegOutput segoutput2(
.clock (clock),
.number (hundredscomplete),
.segOutput (seg2)
);

generateSevenSegOutput segoutput3(
.clock (clock),
.number (thousandscomplete),
.segOutput (seg3)
);
//This always block keeps the number in the numbersplit module changing mid count.
always @(posedge clock) begin
  if(prevNumber != number && !start && !started) begin
    start <= 1;
    started <= 1;
  end else if(start) begin
    start <= 0;
  end else if(done) begin
    started <= 0;
    onescomplete <= ones;
    tenscomplete <= tens;
    hundredscomplete <= hundreds;
    thousandscomplete <= thousands;
    prevNumber <= number;
  end
end

endmodule


//Based on idea from https://stackoverflow.com/questions/22882882/split-up-a-four-digit-number-in-verilog
//This module splits numbers into ones, tens, hundreds, thousands by counting. This is less expensive than division.
module numbersplit(
  input clock, //50MHz Clock
  input [13:0] mynumber, //number to count to
```

Figure 26: SevenSeg_IP.v Part 1

```
  input         start, //start out
  output        doneOut, //complete out
  output [3:0] onesOut, //ones out
  output [3:0] tensOut, //tens out
  output [3:0] hundredsOut, //hundreds out
  output [3:0] thousandsOut //thousands out
);
  //registers
  reg    [13:0] counter = 0;
  reg    [3:0]  ones = 0;
  reg    [3:0]  tens = 0;
  reg    [3:0]  hundreds = 0;
  reg    [3:0]  thousands = 0;
  reg done = 0;
  //assigns
  assign doneOut = done;
  assign onesOut = ones;
  assign tensOut = tens;
  assign hundredsOut = hundreds;
  assign thousandsOut = thousands;

  always @(posedge clock) begin
    //if start then set all values to 0
     if(start) begin
        counter <= 0;
        done <= 0;
        ones <= 0;
        tens <= 0;
        hundreds <= 0;
        thousands<= 0;
    //else then check if the counter is equal to number if so then set register done to signal completed
     end else if(counter == mynumber) begin
        done <= 1;
    //if not complete then count up to the number but keep a track of single digits
     end else if (!done) begin
        counter <= counter + 1;
        ones <= ones == 9 ? 0 : ones + 1;
        if(ones == 9) begin
          tens <= tens == 9 ? 0 : tens + 1;
          if(tens == 9) begin
             hundreds <= hundreds == 9 ? 0 : hundreds + 1;
             if(hundreds == 9) begin
                thousands <= thousands + 1;
             end
          end
        end
     end
  end
endmodule

module generateSevenSegOutput(
input clock,
input [3:0] number,
output [6:0] segOutput
);

reg [6:0] seg;

assign segOutput = seg;

always @(posedge clock) begin
  if(number == 0) begin
    seg <= ~(7'h3F);
  end else if(number == 1) begin
    seg <= ~(7'h06);
  end else if(number == 2) begin
    seg <= ~(7'h5B);
  end else if(number == 3) begin
    seg <= ~(7'h4F);
  end else if(number == 4) begin
    seg <= ~(7'h66);
  end else if(number == 5) begin
    seg <= ~(7'h6D);
  end else if(number == 6) begin
    seg <= ~(7'h7D);
  end else if(number == 7) begin
    seg <= ~(7'h07);
  end else if(number == 8) begin
    seg <= ~(7'h7F);
  end else if(number == 9) begin
    seg <= ~(7'h67);
  end
end

endmodule
```

Figure 27: SevenSeg_IP.v Part 2

9.15 Appendix O - Sampler Code

```
//SampleandTrigger

module Sample(
  input readClock,  //Clock to read values to VGA
  input writeClock, //Clock to write
  input [11:0] data,//Data in
  input [11:0] screenX, //Screen counter in
  input reset, //reset
  input hold, //hold
  input [11:0] triggerthreshold, //trigger threshold in
  output [11:0] screenData, //Screendata out to VGA IP
  output resample //Resample LED
);

reg [11:0] sampleData[800:0]; //BRAM block of sampled data
reg [11:0] triggerHighPoint = 0; //Trigger threshold highpoint
reg [11:0] samplecounter = 0; //sample counter
reg [11:0] triggercounter = 0; //trigger counter
reg [11:0] outputcounter = 0; //output counter
reg [11:0] outputData = 0; //output data
reg resamplereg = 0; //for LED
assign resample = resamplereg; //for LED
assign screenData = outputData; //screen data out to VGA IP
//Read data to VGA IP out of BRAM
always @(posedge readClock) begin
  outputData <= sampleData[screenX];
end

//Trigger Sample Data
always @(posedge writeClock) begin
//find highest point after
  if(data >= 0 && data <= triggerthreshold && samplecounter > 800) begin
    samplecounter <= 0;
    triggercounter <= 0;
    resamplereg = ~resamplereg;
  //If hold then do nothing
  end else if(samplecounter < 800 && hold) begin

  //if samplecounter is below 480 then sample
  end else if(samplecounter < 800) begin
    sampleData[samplecounter] <= data;
    samplecounter <= samplecounter + 1;
    triggercounter <= triggercounter+1;
  //if trigger counter is over 2000 then set to 0
  end else if (triggercounter > 2000) begin
    triggercounter <= 0;
  //else count up
  end else begin
    samplecounter <= samplecounter + 1;
    triggercounter <= triggercounter+1;
  end
end

endmodule
```

Figure 28: Sample_IP.v

9.16 Appendix P - Controls Module Code

```
/* Controls for FPGA_MiniProject
This module contros which waves/cursor display on screen
the position of the waves/cursors. the volts/div aswell as the time/div
N-Channel Oscilloscope */

//All the inputs & outputs
module controls(
input          switch0, //Cursor X En
input          switch1,  //Cursor Y En
input          switch2, //Signal 1 En
input          switch3,  //Signal 2 En
input          switch4, //Cursor Y set
input          switch5, //Cursor X Set
input          switch6, //Wave 1 Shift /Squish
input          switch7, //Wave 2 Shift/Squish
input          switch8, //Wave 1 Clock
input          switch9, //Wave 2 Clock
//IMPORTANT - BUTTONS ARE ACTIVE LOW
input          butt0,  //x1/y1 left/up
input          butt1,    //x1/y1 right/down
input          butt2,  //x2/y2 left/up
input          butt3,  //x2/y2 right/down
input            buttonClock, //Clock for button refresh
output         hold1Out,
output         hold2Out,
output  [10:0]  cursorY1Out,
output  [10:0]  cursorY2Out,
output  [10:0]  cursorX1Out,
output  [10:0]  cursorX2Out,
output  [3:0]   shiftDown1Out,
output  [3:0]   shiftDown2Out,
output  [5:0]   sampleAdjust1Out,
output  [5:0]   sampleAdjust2Out,
output         cursorX_ENOut,
output         cursorY_ENOut,
output         Wave1_ENOut,
output         Wave2_ENOut,
output  [10:0]  offset1Out,
output  [10:0]  offset2Out,
output         TWave_EnOut,
output reg  [1:0]   waveSel
);
//Default posiions of the cursors
localparam defaultY1 = 60; // 60 pixels = ~500mV
localparam defaultY2 = 120;
localparam defaultX1 = 32;
localparam defaultX2 = 90;
//Parameter for Cursor move Cursor by 1 every clock cycle
localparam moveSize = 1;
//Set each cursor to the default position
reg [10:0] cursorY1 = defaultY1;
reg [10:0] cursorY2 = defaultY2;
reg [10:0] cursorX1 = defaultX1;
reg [10:0] cursorX2 = defaultX2;
//Offset sets the intial postion of the waves on the screen
reg [10:0] offset1 = 30;
reg [10:0] offset2 = 200;
//Initially showing the defualt samples on the screen
reg [5:0] sampleAdjust1 = 0;
reg [5:0] sampleAdjust2 = 0;
//The hold and buttpush reg's are to make sure the buttons work on push and not hold
//Useful for control the size of the time or volts/div
reg hold1 = 0;
reg hold2 = 0;
reg buttPush = 0;
reg buttPush1 = 0;
//Following is to get the test wave up and running, instead of wave 1
reg TWave_En = 0;
assign TWave_EnOut = TWave_En;
//Intial value of shiftdown, allows user to see decent size wave
reg [3:0] shiftDown1 = 3;
reg [3:0] shiftDown2 = 3;
//Enable cursors and Waves to 0
reg cursorX_EN = 0;
reg cursorY_EN = 0;
reg Wave1_EN = 0;
reg Wave2_EN = 0;
//assignments
assign hold1Out = hold1;
assign hold2Out = hold2;
assign cursorY1Out = cursorY1;
assign cursorY2Out = cursorY2;
assign cursorX1Out = cursorX1;
assign cursorX2Out = cursorX2;
assign shiftDown1Out = shiftDown1;
```

Figure 29: Controls_IP.v Part 1

```
assign shiftDown2Out = shiftDown2;
assign sampleAdjust1Out = sampleAdjust1;
assign sampleAdjust2Out = sampleAdjust2;
assign cursorX_ENOut = cursorX_EN;
assign cursorY_ENOut = cursorY_EN;
assign Wave1_ENOut = Wave1_EN;
assign Wave2_ENOut = Wave2_EN;
assign offset1Out = offset1;
assign offset2Out = offset2;
//Following code is for state 1, when switch 8 & 9 are 0
//Contros whether the cursors are on the screen, and the position of them
//buttonClock is atached to the slower clock, so every 93Hz
always @ (posedge buttonClock)
begin
  //State 1
  if (!switch9 && !switch8)
  begin
  //Switch on Cursors when switches 0 & 1 are on
  cursorX_EN <= switch0;
  cursorY_EN <= switch1;
    //Code for yCursors when switch 3 is 1, and key 3 is being pushed move cursor y1 down
    if (switch3 && !butt3)
    begin
        cursorY1 <= cursorY1 + moveSize;
    end
    //Code for yCursors, when switch 3 is 1, and key 2 is being pushed move cursor y1 up
    else if (switch3 && !butt2)
    begin
        cursorY1 <= cursorY1 - moveSize;
    end
    //Code for yCursors, when switch 3 is 1, and key 1 is being pushed move cursor y2 down
    else if (switch3 && !butt1)
    begin
        cursorY2 <= cursorY2 + moveSize;
    end
    //Code for yCursors, when switch 3 is 1, and key 0 is being pushed move cursor y2 up
    else if (switch3 && !butt0)
    begin
        cursorY2 <= cursorY2 - moveSize;
    end
    //Code for xCursors
    //When switch 2 is 1, and key 3 is being pushed move cursor x1 right
    if (switch2 && !butt3)
    begin
        cursorX1 <= cursorX1 + moveSize;
    end
    //When switch 2 is 1, and key 2 is being pushed move cursor x1 left
    else if (switch2 && !butt2)
    begin
        cursorX1 <= cursorX1 - moveSize;
    end
    //When switch 2 is 1, and key 1 is being pushed move cursor x2 right
    else if (switch2 && !butt1)
    begin
        cursorX2 <= cursorX2 + moveSize;
    end
       //When switch 2 is 1, and key 0 is being pushed move cursor x2 left
    else if (switch2 && !butt0)
    begin
        cursorX2 <= cursorX2 - moveSize;
    end
    //Code to move both Y Cursors @ same time
    if (switch3 && switch2 && !butt3)
    begin
        cursorY1 <= cursorY1 + moveSize;
        cursorY2 <= cursorY2 + moveSize;
        cursorX1 <= defaultX1;
    end
    if (switch3 && switch2 && !butt2)
    begin
        cursorY1 <= cursorY1 - moveSize;
        cursorY2 <= cursorY2 - moveSize;
        cursorX1 <= defaultX1;
    end
    //Code to move both X Cursors @ same time
    if (switch3 && switch2 && !butt1)
    begin
        cursorX1 <= cursorX1 + moveSize;
        cursorX2 <= cursorX2 + moveSize;
        cursorY2 <= defaultY2;
    end
    if (switch3 && switch2  && !butt0)
    begin
        cursorX1 <= cursorX1 - moveSize;
        cursorX2 <= cursorX2 - moveSize;
```

Figure 30: Controls_IP.v Part 2

```
        cursorX2 <= cursorX2 - moveSize;
        cursorY2 <= defaultY2;
    end
    //Menu 1 switch 4 decides what wave we gonna measure
    if (!switch4)
    begin
        waveSel <= 0;
    end
    else if (switch4)
    begin
        waveSel <= 1;
    end
  end
end
//Following code is for state 2, when switch 8 is 1 & switch 9 is 0
//Contros whether the waves are on the screen, and the position of them
always @ (posedge buttonClock)
begin
  //State 2
  if (!switch9 && switch8)
  begin
  //Switch on waves when switches 0 & 1 are on
  Wave1_EN <= switch0;
  Wave2_EN <= switch1;
      //Code to move wave 1 down the screen, when switch 2 is 1, switch 5 is 0, and key 3 is being pushed
      if (switch2 && !butt3 && !switch5)
    begin
        offset1 <= offset1 + moveSize;
    end
    //Code to move wave 1 up the screen, when switch 2 is 1, switch 5 is 0, and key 2 is being pushed
    else if (switch2 && !butt2 && !switch5)
    begin
        offset1 <= offset1 - moveSize;
    end
      //Code to move wave 2 down the screen, when switch 2 is 1, switch 5 is 0, and key 1 is being pushed
    else if (switch2 && !butt1 && !switch5)
    begin
        offset2 <= offset2 + moveSize;
    end
    //Code to move wave 2 up the screen, when switch 2 is 1, switch 5 is 0, and key 0 is being pushed
    else if (switch2 && !butt0 && !switch5)
    begin
        offset2 <= offset2 - moveSize;
    end
  end
end
//Code for changing the volts/div
always @ (posedge buttonClock)
begin
  //Again on state 2
  if (!switch9 && switch8)
  begin
     //When switch 3 is 1, buttpush is 0, and key 3 is pressed
    if (switch3 && !butt3 && !buttPush)
    begin
       //Make buttpush 1, Increase volts/div for wave 1
      buttPush <= 1;
      shiftDown1 = shiftDown1 + 1;
    end
     //When switch 3 is 1, buttpush is 0, and key 2 is pressed
    else if (switch3 && !butt2 && !buttPush)
    begin
       //Make buttpush 1, Decrease volts/div for wave 1
      buttPush <= 1;
      shiftDown1 = shiftDown1 - 1;
    end
     //When switch 3 is 1, buttpush is 0, and key 1 is pressed
    else if (switch3 && !butt1 && !buttPush)
    begin
       //Make buttpush 1, Increase volts/div for wave 2
      buttPush <= 1;
      shiftDown2 = shiftDown2 + 1;
    end
     //When switch 3 is 1, buttpush is 0, and key 0 is pressed
    else if (switch3 && !butt0 && !buttPush)
    begin
       //Make buttpush 1, Decrease volts/div for wave 2
      buttPush <= 1;
      shiftDown2 = shiftDown2 - 1;
    end
    //Immediately after button is pushed, reest buttpush so that button only works on press not push
    else if ((butt0 && butt1 && butt2 && butt3) && buttPush) begin
      buttPush <= 0;
    end
  end
end
```

Figure 31: Controls_IP.v Part 3

```
// Code for holding postion of waves on screen i.e. freezing the wave
always @ (posedge buttonClock)
begin
  //Again on state 2
  if (!switch9 && switch8)
  begin
    if (switch4 && !butt3 && !hold1)
    begin
      hold1 <= 1;
    end
    else if (switch4 && !butt2 && hold1)
    begin
      hold1 <= 0;
    end
    else if (switch4 && !butt1 && !hold2)
    begin
      hold2 <= 1;
    end
    else if (switch4 && !butt0 && hold2)
    begin
      hold2 <= 0;
    end
  end
end
// Code for changing time/div
always @ (posedge buttonClock)
begin
  //Again on state 2
  if (!switch9 && switch8)
  begin
     //When switch 5 is 1, buttpush is 0, and key 3 is pressed
    if (switch5 && !butt3 && !buttPush1)
    begin
       //Make buttpush 1, Increase time/div for wave 1
      buttPush1 <= 1;
      sampleAdjust1 <= sampleAdjust1 + 1;
    end
     //When switch 5 is 1, buttpush is 0, and key 2 is pressed
    else if (switch5 &&  !butt2 && !buttPush1)
    begin
       //Make buttpush 1, Decrease time/div for wave 1
       buttPush1 <= 1;
      sampleAdjust1 <= sampleAdjust1 - 1;
    end
     //When switch 5 is 1, buttpush is 0, and key 1 is pressed
    else if (switch5 &&  !butt1 && !buttPush1)
    begin
       //Make buttpush 1, Increase time/div for wave 2
      buttPush1 <= 1;
      sampleAdjust2 <= sampleAdjust2 + 1;
     //When switch 5 is 1, buttpush is 0, and key 0 is pressed
    end
    else if (switch5 &&  !butt0 && !buttPush1)
    begin
       //Make buttpush 1, Decrease time/div for wave 2
      buttPush1 <= 1;
      sampleAdjust2 <= sampleAdjust2 - 1;
    end
    //Immediately after button is pushed, reest buttpush so that button only works on press not push
    else if ((butt0 && butt1 && butt2 && butt3) && buttPush1) begin
      buttPush1 <= 0;
    end
  end
end
// Code for testWave on screen
always @ (posedge buttonClock)
begin
  //State 3
  if (switch9 && switch8)
  begin
    TWave_En <= switch0;
  end
end
endmodule
```

Figure 32: Controls_IP.v Part 4

9.17 Appendix Q - Counter Module Code

```
/* Slower clock for FPGA_MiniProject
This module takes the defaullt 50MHz clock & slows it to ~93Hz
N-Channel Oscilloscope */

module clockcounter(
input clock,
output [25:0] counterout
);
//Store output of the register
reg [25:0] counter = 0;
assign counterout = counter;
//Every clock cycle increase the counter by 1
always @(posedge clock) begin
  counter <= counter + 1;
end
endmodule
```

Figure 33: Counter_IP.v

9.18 Appendix R - Measure Module Code

```
/* Measurements for FPGA_MiniProject
This module measures the voltage of the waves and dispalys it on the seven seg display
N-Channel Oscilloscope */

//All the inputs & outputs
module Measure(
input       buttonClock,
input  [10:0] cursory1,
input  [10:0] cursory2,
input  [10:0] cursorx1,
input  [10:0] cursorx2,
input  [5:0]  sampleadjust1, //sample rate
input  [5:0]  sampleadjust2,
input  [3:0]  shiftDown1, //shrink
input  [3:0]  shiftDown2,
input  [1:0]  waveSel, //wave select to measure
input  [2:0]  measurement, //0 - no measurement, 1 is cursor x, 2 is cursor y
//Number to be dispalyed on the 7 seg
output [13:0] num
);
//Differance in cursor values, initialy 0
reg [13:0] deltay1 = 0;
reg [13:0] deltay2 = 0;
reg [13:0] deltax1 = 0;
reg [13:0] deltax2 = 0;
//Result 6, if nothing happens should see 6 on 7 seg display
reg [13:0] result = 6;
//Reg's to store result of wave in
reg [13:0] vx1 = 0;
reg [13:0] vx2 = 0;
reg [13:0] fy1 = 0;
//Num = Result
assign num = result;
//Delta x
wire [13:0] Diffx;
//Find Delta x - makes sure we get no -ve values
assign Diffx = (deltay1 < 0) ? deltay2 : deltay1;
//Delta y
wire [13:0] Diffy;
//Find Delta y
assign Diffy = (deltax1 < 0) ? deltax2 : deltax1;

always @(posedge buttonClock)
begin
  if (waveSel == 0)
  begin
  //Get values of Delta's
  deltay1 <= cursory1 - cursory2;
  deltay2 <= cursory2 - cursory1;
  deltax1 <= cursorx1 - cursorx2;
  deltax2 <= cursorx2 - cursorx1;
  //Caluclate voltage
  vx1 <= (((shiftDown1)* (shiftDown1)) * Diffx);
  //Store value in result
  result <= vx1;
  end
  else if(waveSel == 1)
  begin
  deltay1 <= cursory1 - cursory2;
  deltay2 <= cursory2 - cursory1;
  deltax1 <= cursorx1 - cursorx2;
  deltax2 <= cursorx2 - cursorx1;
  //Caluclate voltage
  vx2 <= (((shiftDown2)* (shiftDown2)) * Diffx);
  //Store value in result
  result <= vx2;
  end
end
endmodule
```

Figure 34: Measure.v

9.19 Appendix S - VGA Module Original Team Code

This is the original code for the VGA IP which didn't work due to a 2ns time delay.

```
module VGA_drawPixel(
input          clock,
input          x_pos,
input          y_pos,
input      [7:0] colour_R,
input      [7:0] colour_G,
input      [7:0] colour_B,
output         vga_hsync,
output         vga_vsync,
output  [7:0]   R,
output  [7:0]   G,
output  [7:0]   B

);
//parameters for 640x480
localparam clockspeed = 25000000;
localparam h_a=3800;  //nanoseconds - Sync //3800
localparam h_b=1900;  //nanoseconds - Backporch //1900
localparam h_c=25400;   //nanoseconds - Display Interval //25400
localparam h_d=600;     //nanoseconds - Front Porch //600
localparam v_a=2;     //lines - Sync
localparam v_b=33;    //lines - Backporch
localparam v_c=480;   //lines - Display Interval
localparam v_d=10;    //lines - Front Porch
localparam Horizontal_Size = 640;
localparam Vertical_Size = 480;


reg [9:0] screenPosition = 0;
reg [9:0] linePosition = 0;

//What the counters must count up to to switch at the correct time.
localparam integer h_a_endcount = clockspeed * (h_a * 0.000000001);
localparam integer h_b_endcount = clockspeed * (h_b * 0.000000001);
localparam integer h_c_endcount = clockspeed * (h_c * 0.000000001);
localparam integer h_d_endcount = clockspeed * (h_d * 0.000000001);
//localparam integer v_a_endcount = clockspeed * (v_a * 0.000000001);
//localparam integer v_b_endcount = clockspeed * (v_b * 0.000000001);
//localparam integer v_c_endcount = clockspeed * (v_c * 0.000000001);
//localparam integer v_d_endcount = clockspeed * (v_d * 0.000000001);

//calculate highest reg size required
localparam Hregsize_a = $clog2(h_a_endcount);
localparam Hregsize_b = $clog2(h_b_endcount);
localparam Hregsize_c = $clog2(h_c_endcount);
localparam Hregsize_d = $clog2(h_d_endcount);
localparam Vregsize_a = $clog2(v_a);
localparam Vregsize_b = $clog2(v_b);
localparam Vregsize_c = $clog2(v_c);
localparam Vregsize_d = $clog2(v_d);
localparam Hozregsize = $clog2(Horizontal_Size);
localparam Verregsize = $clog2(Vertical_Size);

//counter initilisations
reg [Hregsize_a:0] h_a_counter  = 0;
reg [Hregsize_b:0] h_b_counter  = 0;
reg [Hregsize_c:0] h_c_counter  = 0;
reg [Hregsize_d:0] h_d_counter  = 0;
reg [Vregsize_a:0] v_a_counter  = 0;
reg [Vregsize_b:0] v_b_counter  = 0;
reg [Vregsize_c:0] v_c_counter  = 0;
reg [Vregsize_d:0] v_d_counter  = 0;

//Positions counter initilisations
reg [Hozregsize:0] HozPixel = 0;
reg [Verregsize:0] VerPixel = 0;
//Indicator for section of signal
reg [2:0] HozsigIndicator = 0;
/*
0 - sync
1 - backporch
2 - data
3 - frontporch
*/
reg [2:0] VerSigIndicator = 0; //Keep track of each counter.
reg VerSigOn = 0; //When 1 it will turn off colour signals
reg rstV = 0;
/*
0 - sync
1 - backporch
2 - data
3 - frontporch
*/

//assign hsync. frame
```

Figure 35: Original VGA IP Part 1

```
assign vga_hsync = (HozsigIndicator == 0) ? 0 : 1;
assign R = (HozsigIndicator == 2 && VerSigOn == 0)? colour_R : 0;
assign G = (HozsigIndicator == 2 && VerSigOn == 0)? colour_G : 0;
assign B = (HozsigIndicator == 2 && VerSigOn == 0)? colour_B : 0;
//assign vsync
assign vga_vsync = (VerSigIndicator == 0 && VerSigOn == 1 && rstV == 0) ? 0 : 1;
//counters
always @(posedge clock) begin
  //If sync
  if(HozsigIndicator == 0) begin
    if(h_a_counter == h_a_endcount) begin
      h_a_counter <= 0;
      HozsigIndicator <= 1;
    end else begin
    h_a_counter <= h_a_counter + 1;
    end
  end

  //if backporch
  if(HozsigIndicator == 1) begin
    if(h_b_counter == h_b_endcount) begin
      h_b_counter <= 0;
      HozsigIndicator <= 2;
    end else begin
    h_b_counter <= h_b_counter + 1;
    end
  end

  //if data
  if(HozsigIndicator == 2) begin
    if(h_c_counter == Horizontal_Size) begin //changed to work with counting pixels
      h_c_counter <= 0;
      HozsigIndicator <= 3;
      if(VerSigOn == 0) begin
        VerPixel <= VerPixel + 1;
      end else begin
        VerPixel <= 0;
      end
    end else begin
    h_c_counter <= h_c_counter + 1;
    end
  end

  //if frontporch
  if(HozsigIndicator == 3) begin
    if(h_d_counter == h_d_endcount) begin
      h_d_counter <= 0;
      HozsigIndicator <= 0;
    end else begin
    h_d_counter <= h_d_counter + 1;
    end
  end

end

//Pixel Counter & V Sync
always @(posedge clock) begin
  //##Counter
  if(VerSigOn == 0) begin
  //always one behind
    if(h_c_counter == 640 || vga_hsync == 0 || VerSigOn == 1) begin
      HozPixel <= 0;
    end
    else begin
      HozPixel <= h_c_counter + 1;
    end
  end
  if(VerSigOn == 0 && vga_hsync == 1 && HozsigIndicator == 1) begin
      if(VerPixel >= Vertical_Size) begin
        VerSigOn <= 1;
      end
  end

  if(rstV == 1) begin
    VerSigOn <= 0;
  end
  //end

  //###### V SYNC ######


end

//VerPixel == 0;
reg rstcounter = 1;
```

Figure 36: Original VGA IP Part 2

```
//Vsync
always @(posedge vga_hsync) begin

  if(VerSigOn == 1 && rstV == 0) begin
    //if sync
    if(VerSigIndicator == 0) begin
        if(v_a_counter == v_a) begin
          v_a_counter <= 0;
          VerSigIndicator <= 1;
        end else begin
        v_a_counter <= v_a_counter + 1;
        end
      end

    //if backporch
    else if(VerSigIndicator == 1) begin
      if(v_b_counter == v_b) begin
        v_b_counter <= 0;
        VerSigIndicator <= 2;
      end else begin
      v_b_counter <= v_b_counter + 1;
      end
    end

    //if data
    else if(VerSigIndicator == 2) begin
      if(v_c_counter == v_c) begin
        v_c_counter <= 0;
        VerSigIndicator <= 3;
      end else begin
      v_c_counter <= v_c_counter + 1;
      end
    end

    //if frontporch
    else if(VerSigIndicator == 3) begin
      if(v_d_counter == v_d) begin
        v_d_counter <= 0;
        VerSigIndicator <= 0;
        rstV <= 1;
      end else begin
      v_d_counter <= v_d_counter + 1;
      end
    end
  end

  if(rstV == 1) begin
    rstcounter <= rstcounter + 1;
    if(rstcounter == 1) begin
    rstcounter <= 0;
    rstV <= 0;
    end
  end

end
endmodule
```

Figure 37: Original VGA IP Part 3

References

[1] TerasIC, *TerasIC DE1-Soc User Manual*. TerasIC, 2014.

[2] M. Stump, “Verilog vga clock controller,” Jun 2012, accessed April 2019. [Online]. Available: https://github.com/mstump/verilog-vga-controller/blob/master/src/clock.v

[3] “Split up a four-digit number in verilog.” [Online]. Available: https://stackoverflow.com/questions/22882882/split-up-a-four-digit-number-in-verilog