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).

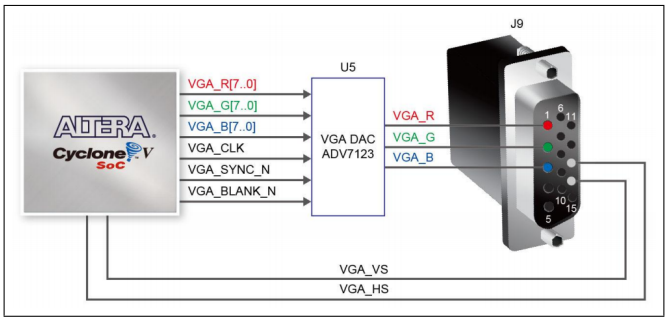

Figure 1: VGA Pins [1]

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.

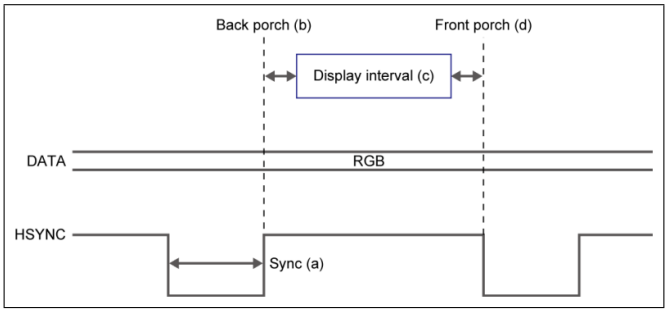

Figure 2: VGA Timings [1]

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
4      if(hsync) begin
5      x <= 0;
6      end else if (vsync) begin
7      x <= 0;
8      end
9    end else begin
10     x <= x+1;
```

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
```

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.

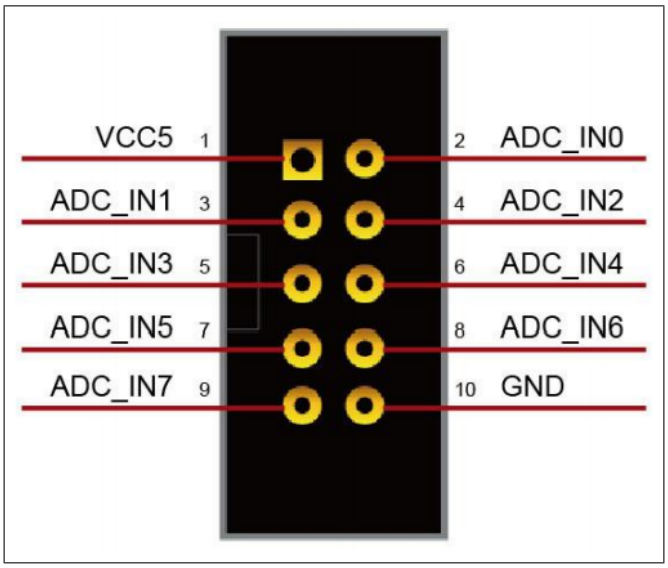

Figure 5: ADC Pins [1]

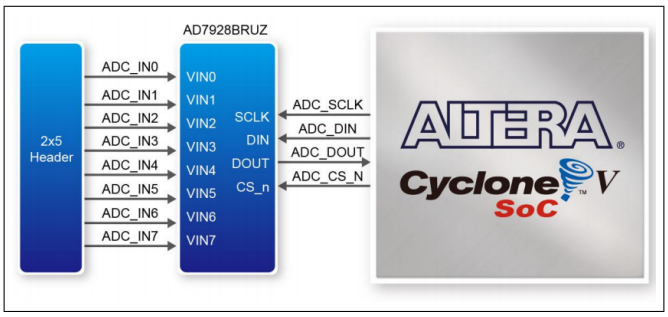

Figure 6: ADC Connections to FPGA [1]

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.

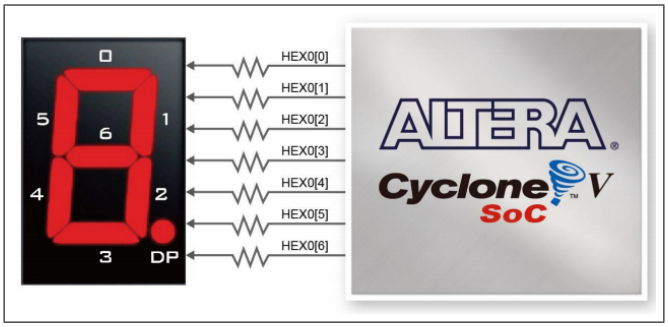

Figure 7: Seven Segment Display Pins [1]

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.

```
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
```

Figure 8: Generate Seven Segment Output Module

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 on 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	N/A
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 E. All the controls happen on the positive edge of buttonClock. In appendix A, it can be seen that the buttonClock attaches to the slClock [19]. The code for the slClock can be seen in appendix F. 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

7 Problems Encountered

8 Conclusion

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

```
module numbersplit(
  input clock,
  input [13:0] mynumber,
  input        start,
  output       doneOut,
  output [3:0] onesOut,
  output [3:0] tensOut,
  output [3:0] hundredsOut,
  output [3:0] thousandsOut
);
  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;
  assign doneOut = done;
  assign onesOut = ones;
  assign tensOut = tens;
  assign hundredsOut = hundreds;
  assign thousandsOut = thousands;
  always @(posedge clock) begin
     if(start) begin
        counter <= 0;
        done <= 0;
        ones <= 0;
        tens <= 0;
        hundreds <= 0;
        thousands<= 0;
     end else if(counter == mynumber) begin
        done <= 1;
     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
```

Figure 9: Number Split Module

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