ELEC5563: Digital Design of System on Chip

Report
Mini-Project: Oscilloscope

By Zubair Lutfullah (ID: 200720710) and Mohammed Bakir (ID: 200749333)

Abstract

This is a report on the implementation of an oscilloscope on the DE2-35 FPGA development kit for the Mini-Project as part of the ELEC5563 course. The basic functionality of an oscilloscope i.e. sample an analogue signal and display it were the minimum requirements. Additional features were suggested for additional marks. Apart from the minimum requirements, more than half of the additional functionalities have been successfully implemented along with other features which weren't directly indicated.

Table of Contents

Table of Figures

1 Introduction

This report is on the implementation of an oscilloscope on the DE2-35 board as part of the Mini-project for the ELEC5563 module. The minimum requirement of the projects were:

- Sample a signal using the ADC/DAC board.
- Display the signal sampled on to a screen via the VGA port.

After successful implementation of the minimum requirements, the additional features were implemented. The following is a list of additional features that were suggested in the project hand-out along with some notes on the implementation level.

- How many channels are you going to have? One or Two?
  - ✓ Two channels have been implemented. They are independent of each other.
- Will you have a trigger input? If so, how will this be implemented? Using a channel on the ADC card or using a digital input?
  - ✓ Yes. External 'edge' triggering has been implemented.
- If you decide to use both channels, will they be displayed using the same colour or different colours?
  - ✓ Yes. They are displayed in different colours.
- Is the amplitude/division going to be adjustable?
  - ✓ Yes. The amplitude Volts/div is adjustable using the switches on the board.
- Is the period/division going to be adjustable?
  - ✓ Yes the period/division is adjustable using the switches on the board.
- Is the sample frequency going to be adjustable?
  - ✓ No. The sampling frequency is fixed at 62.5MHz
- Is the voltage resolution going to be adjustable?
  - ✓ No. The voltage resolution is fixed at 14 bit. It is compressed to 7 bits to display on VGA.
- How would you make adjustments?
  - ✓ Switches on the DE2 are used to make adjustments to the configuration of the scope.
- Save current screen to memory? Recall the memory at a later time?
  - Not implemented
- Autoset?
  - Not implemented
- Cursors?
  - Not implemented
- Measurements?
  - Not implemented
- Maths?
  - Not implemented
- Averaging or programmable filter?
  - Not implemented

Additional features not mentioned in the project hand-out that were implemented are the background grid on the display and horizontal offset adjustment.

The rest of the report is structured as follows.

Chapter 2 gives a basic background on the main functions of an oscilloscope. Chapter 3 gives a detailed analysis of the implementation of the oscilloscope on the DE2. Chapter 4 gives a short note on additional work during the project. Chapter 5 concludes the report followed by the Appendix. The codes for the project are in the appendix along with explanations. References are given at the end.

2 Basics of oscilloscope

The basic oscilloscope takes an analogue input signal and displays it on a screen. Figure 1 shows an illustration of a standard oscilloscope.

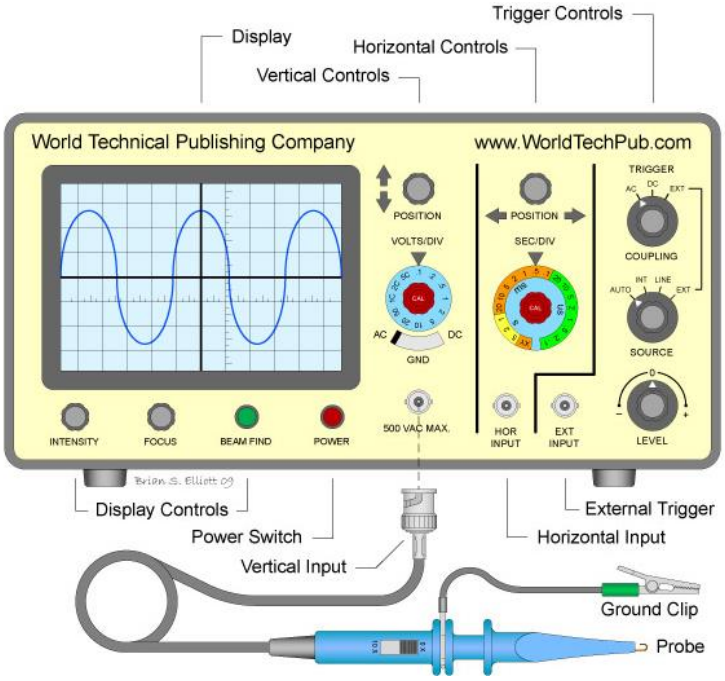

Figure 1 Illustration of basic oscilloscope [i]

The analogue signal is measured using a probe. Voltage controls are used to adjust the y-axis Volts/div. Time base controls are used to adjust the time scale i.e. the seconds/division on the horizontal axis. Trigger controls are used to start the sweep of the display. Triggers can be done automatically at periodic intervals, or external. If external triggering is used, it can be level or edge triggered. The voltage level for triggering is adjustable.

Chapter 3 explores the implementation of these main functionalities of an oscilloscope on the DE2-35 FPGA board.

3 Implementation

The basic task is to sample analogue signal and display it on the VGA display. The task is performed in several stages. Figure 2 shows a breakdown of the blocks forming an oscilloscope.

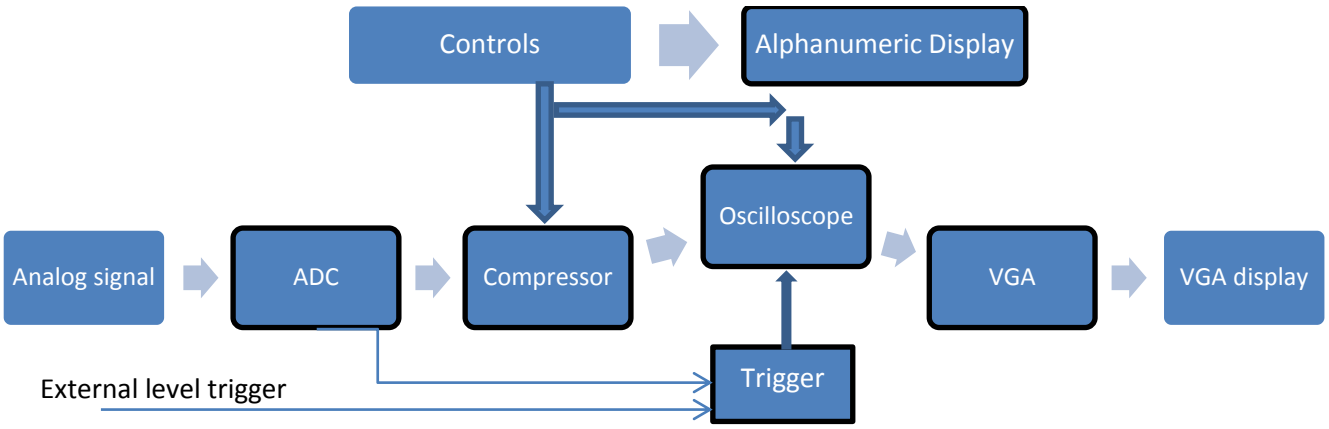

Figure 2 Illustration of signal flow and modules implemented in FPGA

In Figure 2, the blocks with a black border are implemented inside the FPGA. The analogue signal is sampled by the ADC block. The compressor block reduces the dynamic signal range of the signal from the ADC to within the range that the VGA display supports. This block also handles additional features of the oscilloscope. The oscilloscope block buffers the data and handles the output to VGA block. This block also handles additional features of the oscilloscope. The VGA block buffers the data again to be displayed on the VGA display.

Figure 3 illustrates the core part of the implementation that has been developed as part of this project. The rest of the schematic initializes existing driver IP cores such as the VGA/ADC block and the PLL blocks.

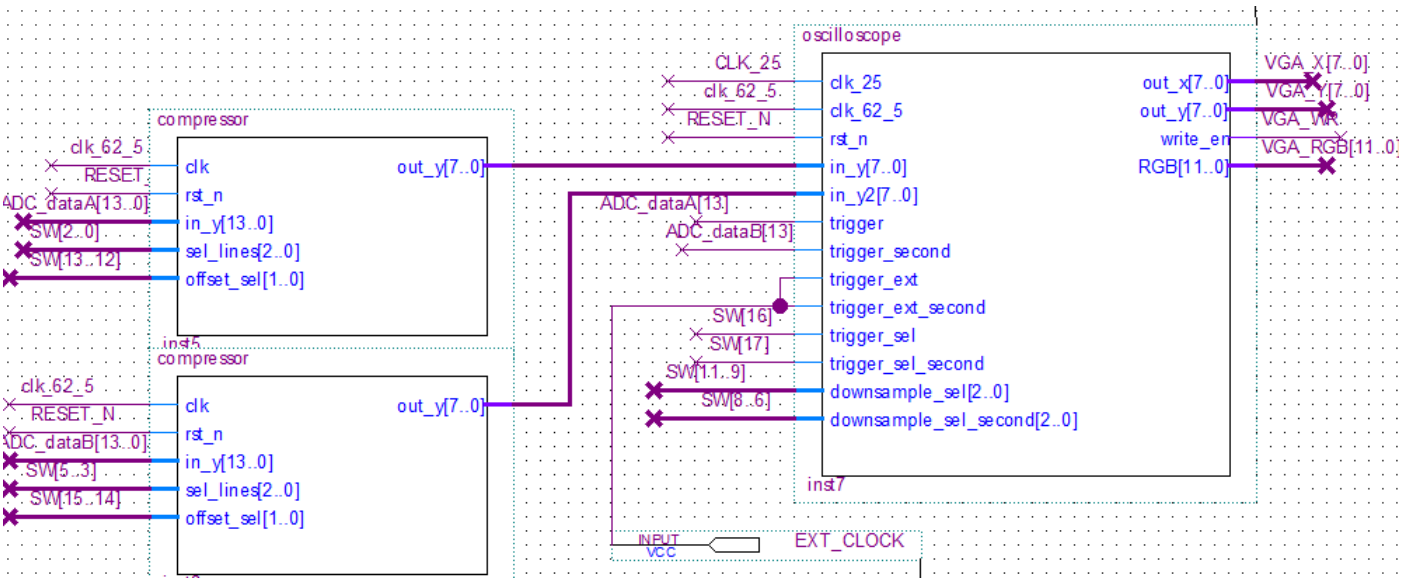

Figure 3 Portion of top-level Quartus schematic implementing core part of oscilloscope

Each block implemented in the FPGA is explained in the following sub-sections.

3.1 ADC

The signal conversion daughter board (THDB_ADA) by Terasic is used to sample the analogue signal. The daughter board interfaces to the GPIO header on the DE2 kit and provides dual ADC channels with 14-bit resolution and a data rate up to 65 MSPS.

The intellectual property (IP) core used to acquire samples from the daughter board was provided in a lab hand-out[ii]. However, the IP block only implements the firmware for acquiring data from one channel from the daughter card. The hardware daughter board supports a second channel and the connections exist. Firmware for the second channel was added to the IP using the GPIO header pin-out seen in Figure 4 and Figure 5 from the datasheet of the daughter card.

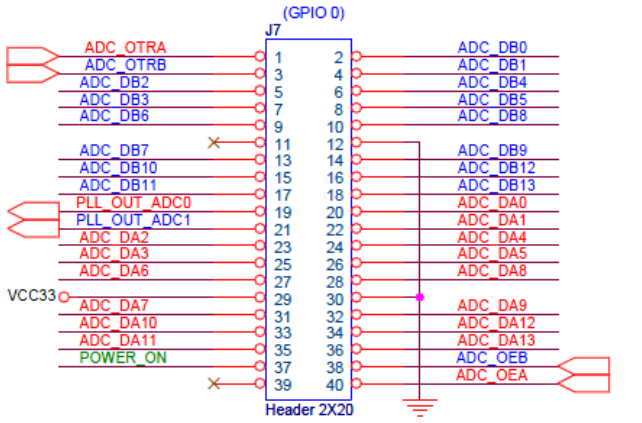

Figure 4 GPIO 0 header from ADC/DAC daughter card datasheet[iii]

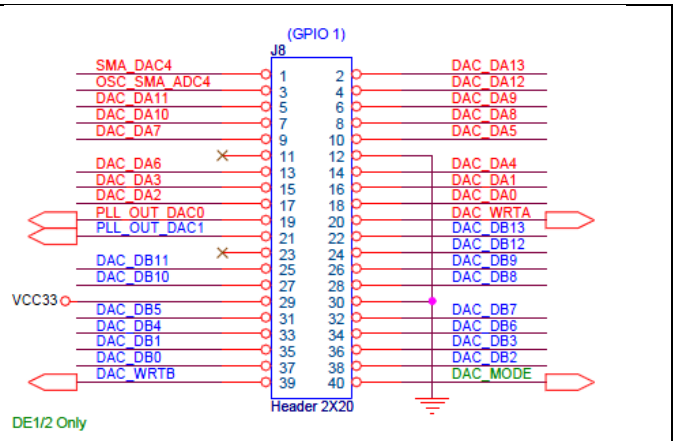

Figure 5 GPIO 1 header from ADC/DAC daughter card datasheet[iii]

Apart from adding input/outputs to the IP core, the following lines of code were added to the ADC IP block.

```
assign  adc_a = {GPIO_0_TEMP[17], GPIO_0_TEMP[19], GPIO_0_TEMP[20], GPIO_0_TEMP[21],
GPIO_0_TEMP[22], GPIO_0_TEMP[23], GPIO_0_TEMP[24],GPIO_0_TEMP[25], GPIO_0_TEMP[26],
GPIO_0_TEMP[27], GPIO_0_TEMP[28], GPIO_0_TEMP[29], GPIO_0_TEMP[30], GPIO_0_TEMP[31]};

// Above line assigns the output adc_a with the signals taken from the GPIO header in datasheet

assign  GPIO_0_TEMP[35] = 0;    // Channel A Enabled
```

The complete code is given in Appendix C.

The block diagram for the modified IP core used is given in Figure 6.

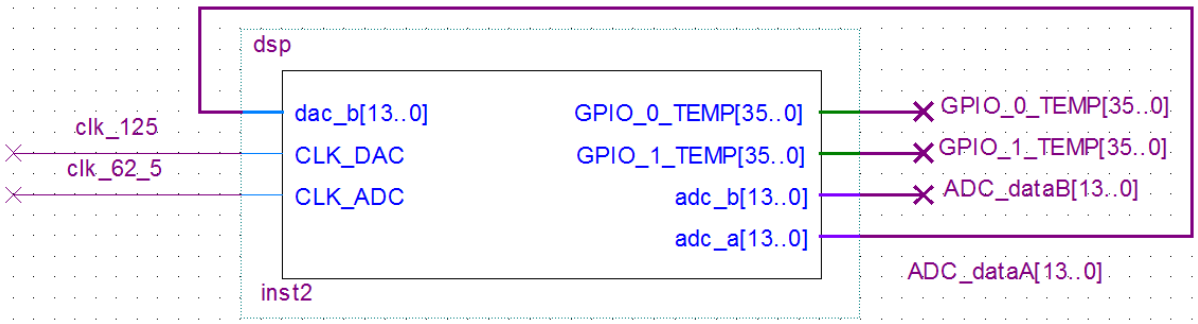

Figure 6 Schematic block for modified ADC/DAC IP core with second ADC channel

The loop back from ADC channel A is simply to check the working of the channel. The label ADC_dataA is used in other parts of the schematic to pass the data.

3.2 Compressor block

The compressor block performs multiple operations on the ADC data to enable it to be displayed on the VGA block.

It handles the following aspects of the oscilloscope.

- Reduce dynamic range of 14-bit ADC to 7-bit wide signal compatible with VGA block
- User-selectable Volts/div of signal being displayed
- User-selectable Horizontal offset of signal being displayed

A graphical illustration of the block can be seen in Table 3 and Table 4.

The ADC block gives a 14 bit wide signal representing the analogue signal as illustrated in Figure 8 in Table 3. However, the VGA display has a range of only 120 pixels which can be represented by 7 bits. Therefore, the dynamic range of the signal is compressed by bit shifting the signal by multiple values based on user input.

The VGA block takes an 8 bit input. Thus the bit width of the signal is set at 8 bit. The actual displayable range is 120 pixels.

The y-axis midpoint of the display is 60. Therefore the offset of the signal has to be set at 60 for the signal to be displayed in the centre of the screen. The original signal's offset is at 8192. The bit-shift for signal range compression shifts the offset by the same value. A constant is then added to correct the offset and bring it to 60.

The basic operation to reduce the dynamic signal range is illustrated in the following equations

$$ADC_{Data} >> X$$

Equation 1

Where X depends on user input and ranges from 4 to 10 to vary the Volts/Div on the display. For offset correction, the following equation is used.

$$Offset_{correction} = 60 - \frac{8192}{2^X}$$

Equation 2

For the range of X, offset correction values found using Equation 2 are given in Table 1.

Table 1 Offset correction values after solving equation 1 for multiple values of X

Switch[2..0] for Channel 1 Switch[5..3] for Channel 2	14 bit ADC signal shifted by X	Offset correction (constant)	Volts/div (V) on display
011	4	-452	0.003125
010	5	-196	0.0625
001	6	-68	0.125
000 and 100	7	-4	0.25
101	8	+28	0.5
110	9	+44	1
111	10	+52	2

After the offset correction, the user selectable manual offset is added to the signal which can take the values seen in Table 2.

Table 2 Switch input and manual offset selection for signal display

Switch [13..12] for Channel 1. SW[15..14] for Channel 2	Horizontal offset (in pixels)
00	0
01	+20
10	+40
11	-20

Figure 7 shows the compressor block from Quartus with its input and outputs shown.

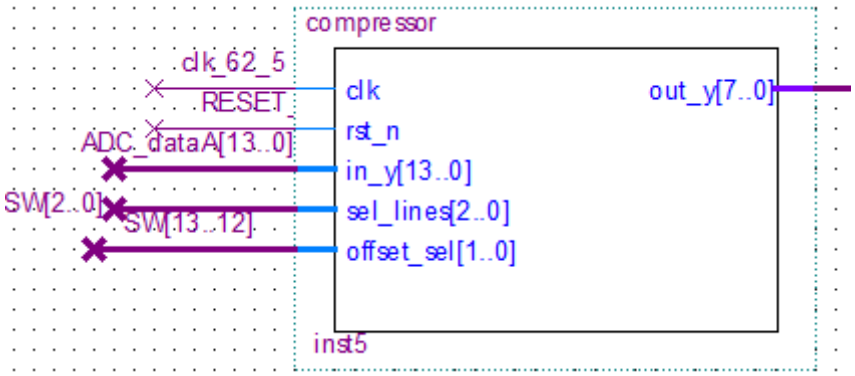

Figure 7 Schematic block for Compressor module

The block works in the 62.5MHz clock domain as the ADC is sampling at that frequency. The sel_lines connected to SW[2..0] are used to select the value of X. The offset_sel lines connected to SW[13..12] are used to select the offset of the signal.

The Verilog code for the compressor block along with details regarding the hardware design and implementation are given in Appendix A

Table 3 and Table 4 illustrate the operations on the signal.

Table 3 Illustration of stages of operations in compressor block. Operation 1 and 2

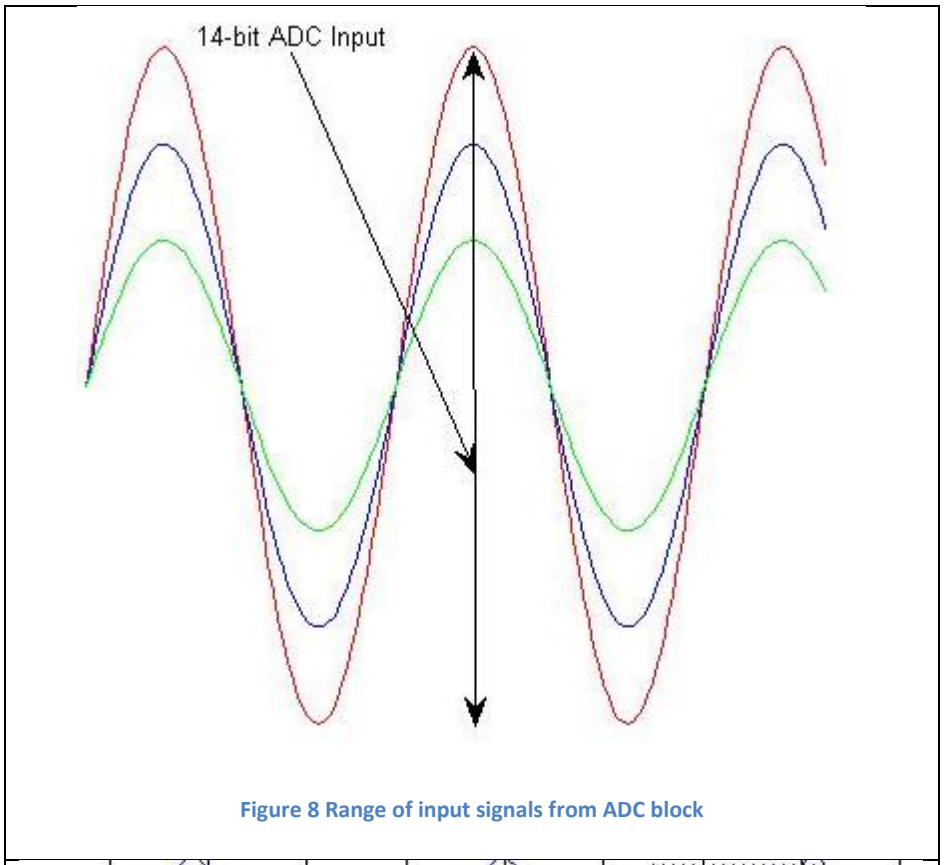

Figure 8 Range of input signals from ADC block

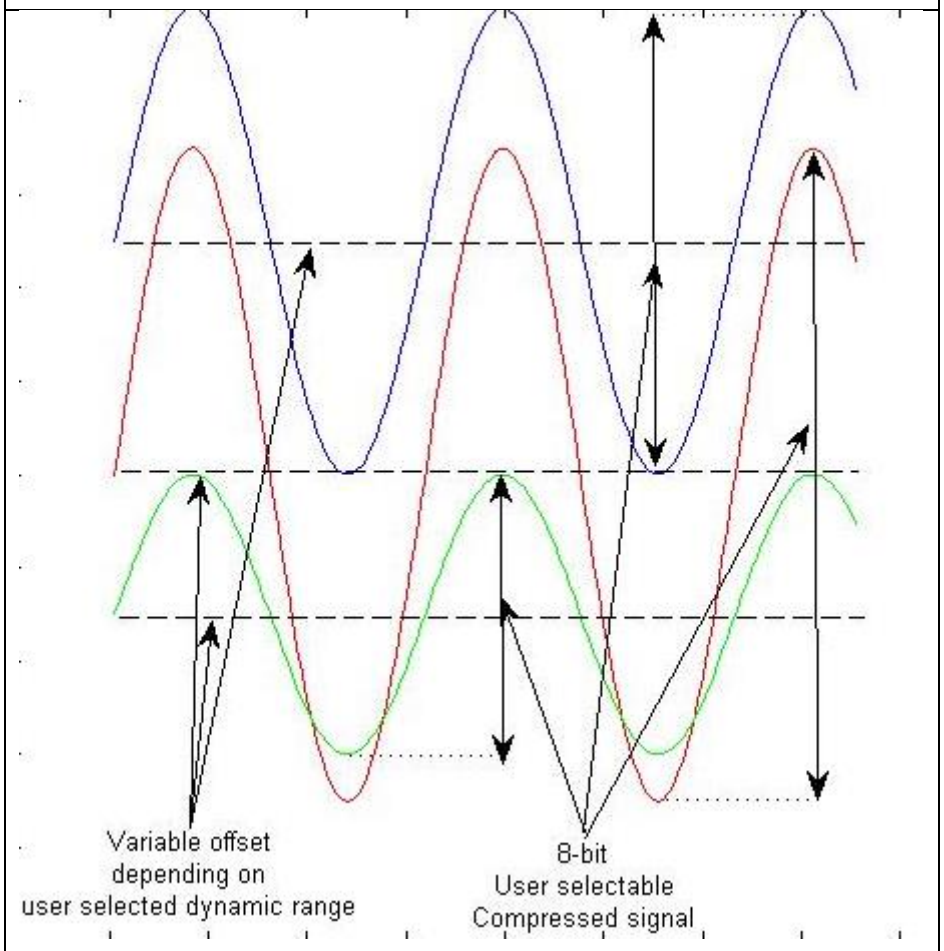

Figure 9 Dynamic range compression of signal after bit shifting by a user selectable number.

Table 4 Illustration of stages of operations in compressor block. Operation 3 and 4

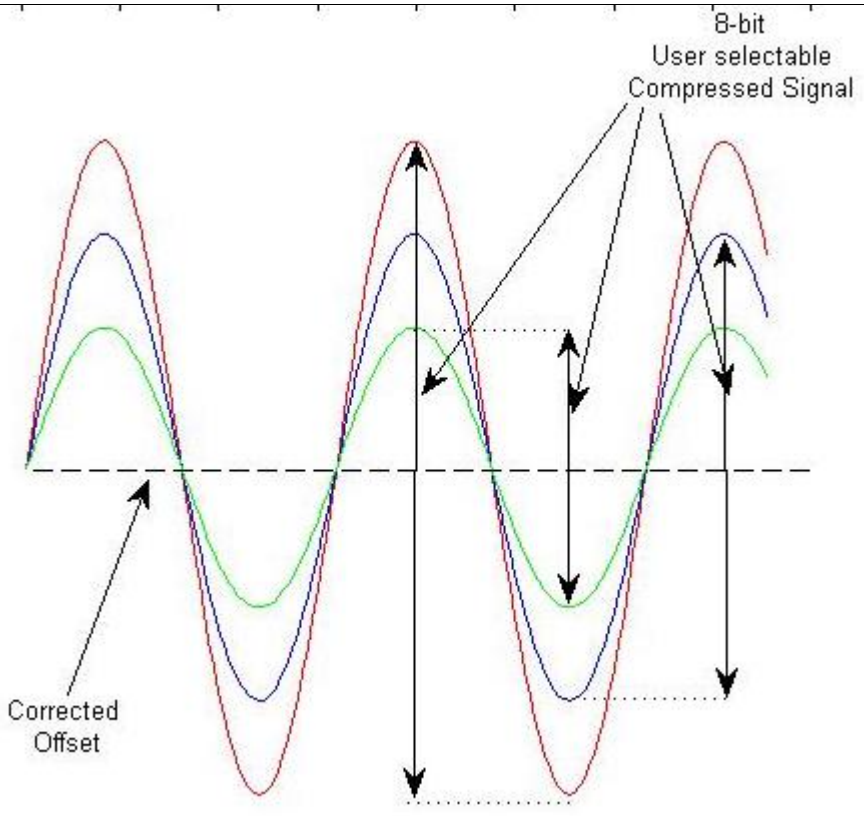

Figure 10 Offset of signal corrected by adding/subtracting calculated constants

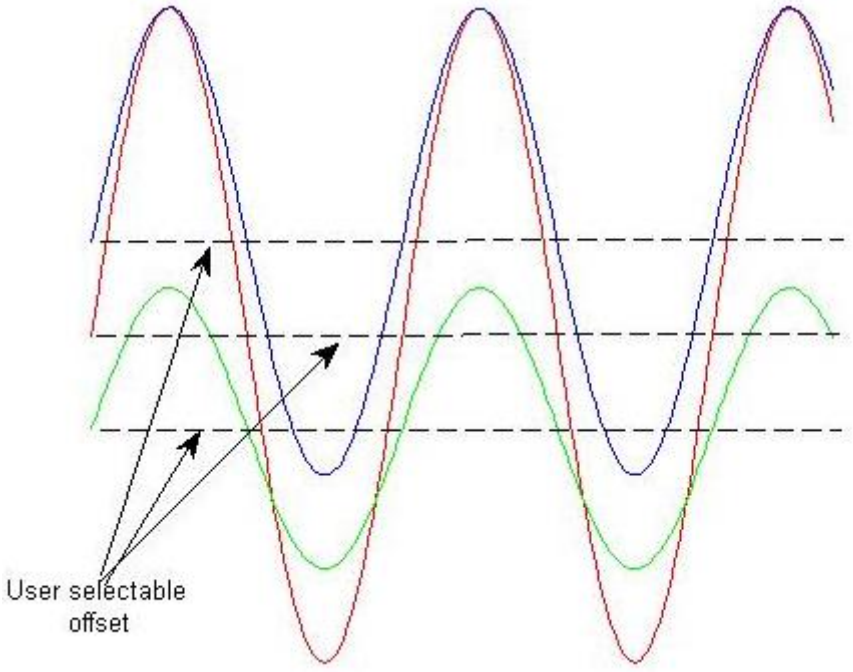

Figure 11 User selectable offsets added to signal

3.3 Oscilloscope block

The oscilloscope block performs multiple operations on the data from the compressor block before passing it to the VGA block.

It handles the following aspects of the oscilloscope.

- Buffers the ADC data to be displayed on the VGA
- Clears the display so that the next buffer can be displayed.
- Provides clock-domain crossing. ADC data is coming at 62.5MHz. VGA driver IP requests data at 25MHz.
- User-selectable time base i.e. seconds/div of signal being displayed on horizontal axis.
- User-selectable Auto/External trigger.

The display process of an oscilloscope is repetitive and is implemented using a state machine. The following steps need to be done in order

1) Wait for trigger signal
2) Clear the VGA display after receiving a trigger signal.
3) Store ADC samples in a buffer
4) Display the buffer on the screen
5) Wait for next trigger signal.

The ADC samples come at a 62.5MHz clock. The VGA block requests data at a 25MHz clock. Therefore, the state-machine also works in two different clock domains. The state machine is handled by the 25MHz clock only. Flags are used for signalling between the clock domains when needed.

Figure 12 illustrates the state machine.

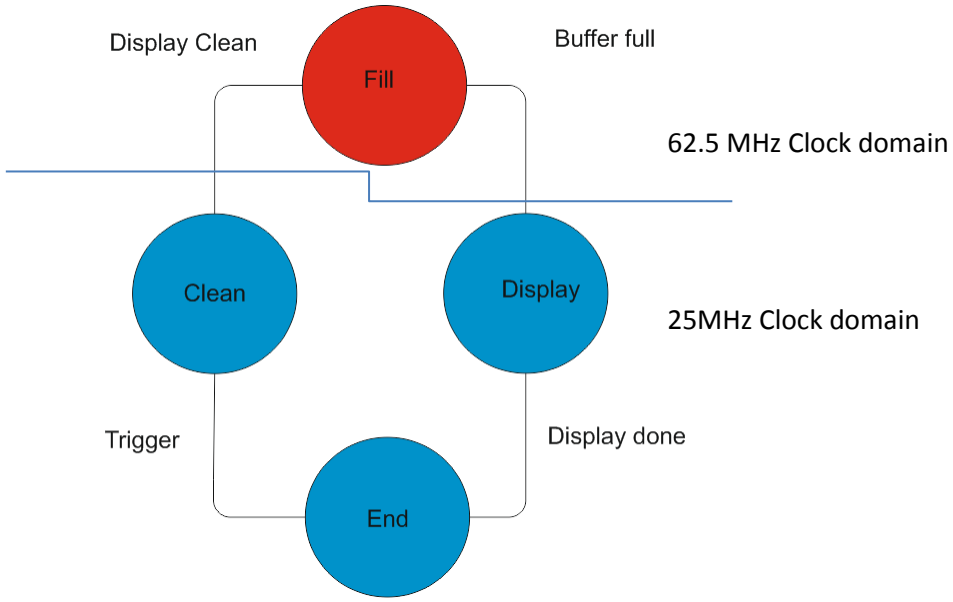

Figure 12 Illustration of State machine in Oscilloscope block

Data is handled between the clock domains using the dual port ram.

Table 5 shows the operations happening on the compressed data in each state and clock domain.

Table 5 Explanation of State Machine in oscilloscope block and operations in each state

25 MHz clock domain	62.5 MHz clock domain
State : End	State : End
- When the trigger signal is detected, the state is changed to Clean. - Otherwise the state stays in the End state. - Temporary flags and variables are cleared	- Temporary flags and variables are cleared
State : Clean	State : Clean
- In this state, the ram is read again. This time those pixels are replaced with black colour. - Or white if those locations lie on the grid locations. - When the ram is read and the display cleaned, the state is changed to Fill.	- Nothing is done in this state in this clock domain.
State : Fill	State : Fill
- If buffer flag from other clock domain is detected high, the state is changed to Display. - Temporary flags and variables are cleared	- In this state, the RAM is filled with ADC samples that have been compressed in the compressor block. - Depending on the user-selectable horizontal time base setting, samples are down-sampled before being stored in the buffer. - When the buffer is full, the buffer flag is pulled high.
State : Display	State : Display
- In this state, samples are read from the RAM and output to the VGA block. - When the buffer has been completely displayed, the state is changed to End	- Nothing is done in this state in this clock domain.

Figure 13 shows the oscilloscope block with its inputs and outputs. It takes in compressed ADC samples on the in_y and in_y2 (channel 2) lines. It takes trigger, external trigger and select lines which are explained in section 3.4. The down sample lines configure the time base of the display. The outputs configure the VGA display IP block.

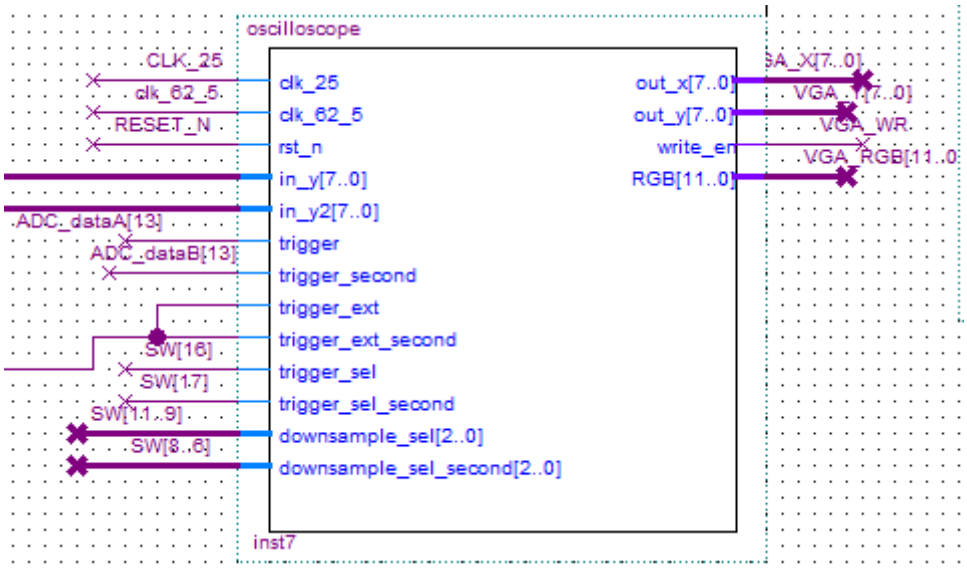

Figure 13 Schematic block for Oscilloscope module

The Verilog code for the oscilloscope block along with details regarding the implementation are given in Appendix B.

3.3.1 Grid

The oscilloscope block indirectly handles the grid display as well. While cleaning samples from the display, it checks the pixel location and replaces the colour with white if the location lies on a grid. Or black if it lies otherwise. The grid, created in Microsoft Paint is given in the Figure 14. Each white square is 20x20 pixels.

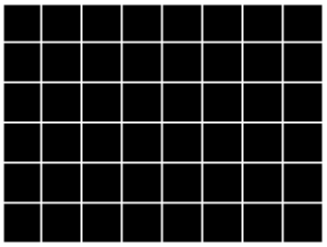

Figure 14 Grid of the oscilloscope. Each 20x20 block is 20 pixels.

Whenever a location in multiples of 20 is checked and cleared, a white pixel is written there by the oscilloscope.

3.3.2 Calibration

The grid is calibrated manually at the very end. The operations (Y-axis scale/timebase) are performed in powers of two in Verilog. Therefore, changes in the configurations do not result in a linear scaling factor. The configurations are checked manually with the signal generator and the display is calibrated. The configured time-base and y-axis settings are displayed on the alphanumeric display as explained in section 3.6.

3.4 Trigger

The trigger block handles when the Display will be refreshed with a new buffer of ADC samples. In the implementation, it controls the state machine execution.

There are two types of triggers handled by the trigger block

- Automatic triggers based on ADC data
- External edge trigger

If an oscilloscope triggers periodically, the display will appear to move horizontally. Therefore, automatic triggering has to be based on the period of the input signal. Every rising trend of the analogue signal is detected. At that point, the trigger signal is sent to the oscilloscope blog.

Instead of complex math operations to find the slope of the input data, automatic triggering is based on the MSB of the ADC data. The ADC data is 14 bit and the 0 value is 8191. i.e. the DC value is 14'b01111111111111. For an analogue signal above 0, the MSB of the ADC signal is high. i.e. 14'b1xxxxxxxxxxxxx. Therefore, every pos-edge of the MSB indicates the point when the analogue signal goes from negative values to positive values. These positive edges are downsampled and every 32nd posedge is used as a trigger signal to the state machine in the oscilloscope block.

For the external edge trigger, a simple edge detection using an XOR of the previous and current value is used to send the trigger signal to the state machine. The input is via the EXT_CLK SMA connector on the DE2 board.

The always block handling the trigger functionality is inside the oscilloscope module seen in Appendix B.

3.5 VGA

The VGA-LCD display driver IP was provided as part of a lab hand-out[iv]. The LCD part of the driver IP is not used and is disconnected from any I/O pins as the same GPIO pins are used by the ADC module.

The major limitation faced was due to super-pixelated output on the VGA display due to limitations in memory. Every 4 pixels on the display were represented by the same pixel in memory. This resulted in a highly pixelated display.

In the case of a 640x480 display, the actual resolution is 160x120 as seen in Figure 15.

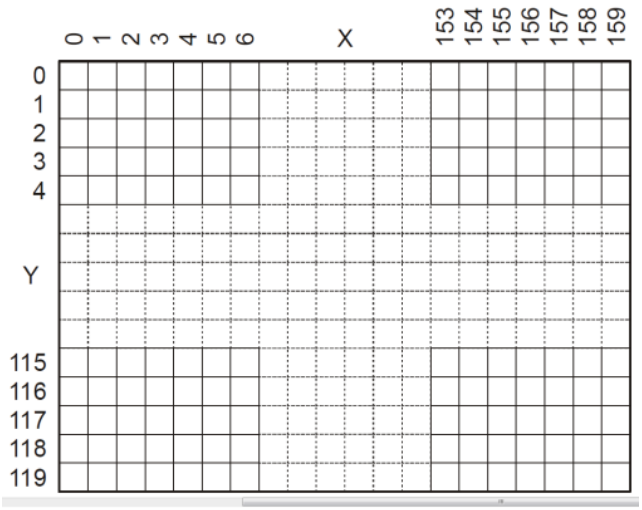

Figure 15 VGA display pixel locations[iv]

The other modules are designed with the locations of the top left corner (0,0) and bottom right corner(159,119) in mind.

3.6 Alphanumeric Display

The alphanumeric display is used to display the time base (secs/div) and vertical (Volts/div) settings selected based on the external 17 switches on the DE2 board. The switches are seen in Figure 16Figure 16 labelled with their functions.

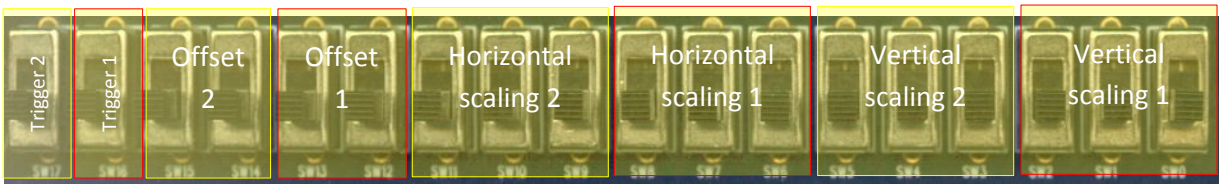

Figure 16 DE2 switches numbered 17 to 0 from the left. Functions in the oscilloscope labelled

Table 1 in section 3.2 shows the switch settings for vertical scaling i.e. voltage (V/div). Table 2 in section 3.2 shows the switch settings for offset. Table 6 below shows the switch settings for the trigger.

Table 6 Trigger switch settings

Switch[17] for Channel 2 Switch[16] for Channel 1	Value	Trigger mode
16	High	External
17	Low	Auto

Error! Not a valid bookmark self-reference. shows the settings for the horizontal scaling i.e. time base (uS/div).

Table 7 Time base switch settings

Switch [8..6] for Channel 1 SW[11..9] for Channel 2	Time base (usecs/div)
000	0.33
001	0.66
010	1.00
011	1.33
100	1.66
101	2.00
110	2.33
111	2.66

The current switch settings are displayed on the alphanumeric display on the DE2 board. The driver IP and display code for the alphanumeric display are adapted from the DE2 test examples in the CD provided with the kit[v].

Figure 17 shows the LCD block schematic. The module code is given in Appendix F.

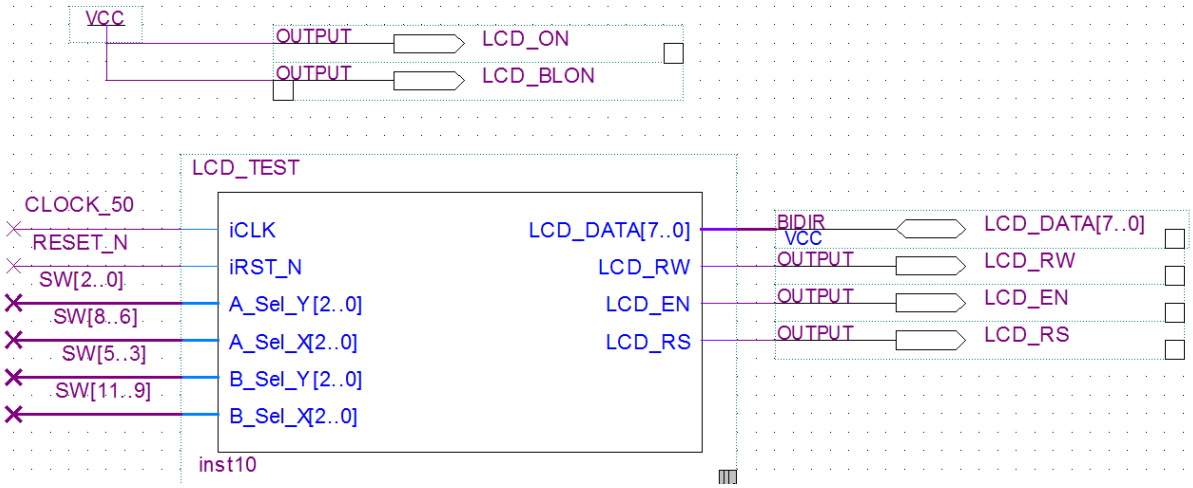

Figure 17 Alphanumeric LCD block diagram

Every time a change in the input is detected, the LCD is refreshed to reflect the new display. The LCD display format is the same for both channels. Line 1 displays settings for channel 1. Line 2 displays settings for channel 2.

Table 8 16x2 Alphanumeric display values based on switch inputs. Line 1 for Channel 1. Line 2 for channel 2.

Alphanumeric display character location	0	1	2	3	4	5	6	7	8	9		10	11	12	13	14	15
Switch[5..3] channel 1 Switch[8..6] channel 2											**Switch[5..3] channel 1 Switch[8..6] channel 2**						
000	A/B	:	0	.	2	5	0	0	V		**000**	0	.	3	3	u	S
001	A/B	:	0	.	1	2	5	0	V		**001**	0	.	6	6	u	S
010	A/B	:	0	.	0	6	2	5	V		**010**	1	.	0	0	u	S
011	A/B	:	0	.	0	0	3	1	V		**011**	1	.	3	3	u	S
100	A/B	:	0	.	2	5	0	0	V		**100**	1	.	6	6	u	S
101	A/B	:	0	.	5	0	0	0	V		**101**	2	.	0	0	u	S
110	A/B	:	1	.	0	0	0	0	V		**110**	2	.	3	3	u	S
111	A/B	:	2	.	0	0	0	0	V		**111**	2	.	6	6	u	S

4 Additional work

Apart from work that formed the core part of the implementation, other activities during the course of the project that have not been thoroughly documented in this report are listed below.

- A module was created using a simple counter that generated triangular waveforms to facilitate testing without the signal generator and the ADC module.
- The triangle generator and compressor block were initially tested using ModelSim test benches.
- Signal Tap II was used to analyse the ADC signals
    - Initially, the ADC MSB based trigger functionality was checked using Signal Tap II
    - This also allows testing without the VGA block
- Timing analyser was used to analyse timing for the code.
    - Pipelining resulted in better timing for the compressor module
    - Timing issues remain unresolved in FILL state of state machine.

5 Conclusion

The main functionality of the oscilloscope has been implemented on the DE2-35 board. More than half of the additional features specified in the project hand-out have been implemented successfully in Verilog.

Figure 18 and Figure 19 show the oscilloscope display while testing.

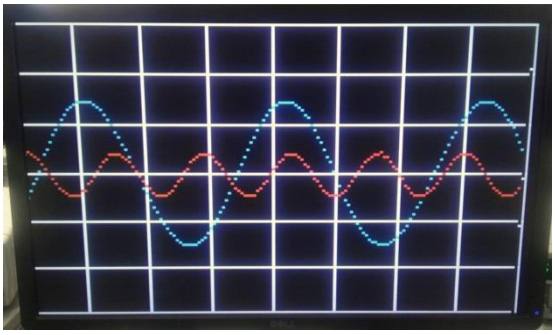

Figure 18 Oscilloscope display during testing

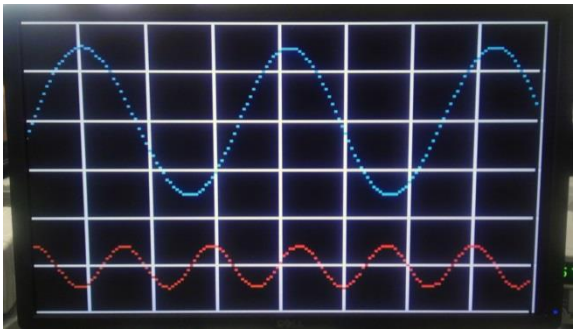

Figure 19 Oscilloscope display during testing with offset changed

The rest of the additional features can be implemented. However, adding further complexity to the state machine in Verilog would make the module unwieldy. The use of the NIOS processor to handle the oscilloscope would have simplified design and addition of further functionality. However, that was not used because of technical issues in the lab installation of Quartus 12.

Appendix A Compressor module code

The compressor block takes an ADC input and reduces its dynamic range by dividing by a user selectable factor. Factor can only change in powers of 2 therefore; the implementation is a simple bit-shift operation.

Offset adjustments are carried out via simple addition/subtraction. The synthesis tool handles the implementation details of the adder/subtractor.

Timing analysis failed therefore, the operations were pipelined. Bit shifting was carried out in one cycle. And addition/subtraction for the offset was carried out in the other.

The implementation is via a case statement. Depending on user input a shift is done on the ADC data and an offset added. The offset varies depending on a case statement in a separate always block.

The following code implements the compressor block explained in section 3.2.

```
/*
This module takes a 14 bit ADC input and outputs a compressed 8 bit data to fit the VGA pixels
The dynamic range of the signal is reduced depending on the user select lines.
Amplitude per division and offset depends on the select lines.
Authors: Zubair Lutfullah and M. Bakir
*/
module compressor
(
                              // Input Ports
       input clk,                     //Clk
       input rst_n,                   //Reset Neg
       input [13:0] in_y,             //ADC Input
       input [2:0]sel_lines,          //Select lines for signal dynamic range compression
       input [1:0]offset_sel,         //Select lines for offset in channel
                              // Output Ports
       output  [7:0]out_y             //Compressed output for Oscilloscope
);

reg [13:0]temp_y;    //Temporary register to pipeline compressor to meet timing.
reg [13:0]temp_y2;   //Temporary register to pipeline compressor to meet timing.
integer offset;//NOTE: Integer. Used for manual offset. Synthesis tool optimizes. Not good practice.

always @ (posedge clk)
begin
       if(!rst_n)
       begin
              temp_y <=0;
             temp_y2 <=0;
       end
       else
       begin
              case (sel_lines)
              4'b000: begin
              //offset subtraction is (2^14/2) = 8192 -> 8192 >> 7/6/5/4 + whatever to bring it to 60
                                    temp_y <= (in_y >> 7); //Process done in multiple stages
                                    temp_y2 <= temp_y - 4 + offset; //to improve timing
                                    end
              4'b001: begin
                                    temp_y <= (in_y >> 6); //Process done in multiple stages
                                    temp_y2 <= temp_y - 68 + offset; //to improve timing
                                    end
              4'b010: begin
                                    temp_y <= (in_y >> 5); //Process done in multiple stages
                                    temp_y2 <= temp_y - 196 + offset; //to improve timing
                             end
              4'b011: begin
                                    temp_y <= (in_y >> 4); //Process done in multiple stages
                                    temp_y2 <= temp_y - 452 + offset; //to improve timing
                             end
              4'b100: begin
                                    temp_y <= (in_y >> 7); //Process done in multiple stages
```

```
                                temp_y2 <= temp_y - 4 + offset; //to improve timing
                        end
                4'b101: begin
                                temp_y <= (in_y >> 8); //Process done in multiple stages
                                temp_y2 <= temp_y + 28 + offset; //to improve timing
                        end
                4'b110: begin
                                temp_y <= (in_y >> 9); //Process done in multiple stages
                                temp_y2 <= temp_y + 44 + offset; //to improve timing
                        end
                4'b111: begin
                                temp_y <= (in_y >> 10); //Process done in multiple stages
                                temp_y2 <= temp_y +52 + offset; //to improve timing
                        end
                default: begin
                                temp_y <= in_y >> 7;
                        end
                endcase
        end
end

//Manual offset calculation
always @ (*)
begin
        case(offset_sel)
        2'b00: offset <= 0;
        2'b01: offset <= 20;
        2'b10: offset <= 40;
        2'b11: offset <= -20;
        endcase
end

//Output the compressed and corrected signal
assign out_y  = temp_y2[7:0];

endmodule
```

Appendix B Oscilloscope module code

The oscilloscope module is the main block that takes in the 8-bit compressed signal and outputs it to the VGA block depending on trigger and user configurations.

The oscilloscope module implements a state machine overlapping two clock domains which handles the process. The always block for the 62.5MHz clock takes the ADC data and stores it in the buffer after downsampling based on user time base settings. Once the buffer is full, a flag is raised which is being checked by the 25MHz clock always block. When the buffer full flag is detected, the state is changed to Display.

In the display state, the 25MHz always block is working. It outputs the buffer of stored samples on to the VGA block for display. Once all samples are displayed, the state is changed to End.

The end state also works in the 25Mhz always block. Its function is to wait for the next trigger before proceeding to the clean state.

The clean state works in the 25Mhz always block. It is the same as the display state. However, instead of colouring the pixel to draw the analogue signal on the display, the same pixels are read from the buffer and replaced with the background pixel colour; black for background and white if the location lies on the grid. The next state is fill.

The following code implements the oscilloscope block explained in section 3.3.

```
/*
This module takes a 8 bit compressor input and outputs a signal to the VGA block
depending on the trigger from the trigger block.
Horizontal time base is adjusted by downsampling the input
based on user selected settings.
Authors: Zubair Lutfullah and M. Bakir
*/
module oscilloscope(
        //INPUTS
        input clk_25,                       //25MHz clock
        input clk_62_5,                     //62.5 MHz clock
        input rst_n,                        //Reset Negative
        input [7:0]in_y,                    //Channel 1    from compressor
        input [7:0]in_y2,                   //Channel 2 from compressor
        input trigger,                      //Channel 1 trigger
        input trigger_second, //Channel 2 trigger
        input trigger_ext,
        input trigger_ext_second,    //Second_channel Ext Trigger.
        input trigger_sel,
        input trigger_sel_second,    //Second_channel trigger sel

        input [2:0]downsample_sel,   //Channel 1 Time-base downsample sel
        input [2:0]downsample_sel_second, //Channel 2 Time-base downsample sel
        //OUTPUTS
        output reg [7:0]out_x,       //Horizontal pixel location output for VGA block
        output reg [7:0]out_y,  //Vertical pixel location output for VGA block
        output reg write_en,         //Write Enable for VGA block.
        output reg [11:0]RGB);       //Color of pixel

//State machine variables
parameter CLEAN = 2'b00;
parameter FILL = 2'b01;
parameter DISPLAY = 2'b10;
parameter END = 2'b11;
reg [1:0]state;

//Temporary Registers
reg [7:0]read_addr;   //Read Address for reading from RAM
reg [7:0]write_addr;  //Write Address for writing to RAM
reg buffer_full;      //flag if buffer has been filled by ADC.
reg display_clean; //flag if vga screen has been cleared.

// Declare the RAM variable
reg [7:0] ram_y[159:0];              //Channel 1 buffer
```

```
reg [7:0] ram_y2[159:0];     //Channel 2 buffer
reg second_channel;                 //flag for which channel is being processed

//Horizontal time base downsample registers
reg [2:0]downsample_counter; //Counter for downsampling channel 1
reg [2:0]downsample_counter_second;   //Counter for downsampling channel 2

//Trigger registers
reg [4:0] trigger_counter;   //Counter for downsampling trigger
reg trigger_bit;                    //Flag for trigger to state machine
reg trigger_ext_prev;        //Temp reg to store previous trigger ext

reg [4:0] trigger_counter_second; //Counter for downsampling trigger
reg trigger_bit_second;                     //Flag for trigger to state machine
reg trigger_ext_prev_second; //Temp reg to store previous trigger ext

//Clock domain 62.5 MHz
always @ (posedge clk_62_5)
        begin
        if(!rst_n)     //Clear registers on reset
                begin
                write_addr <= 0;
                buffer_full <= 0;
                downsample_counter <=0;
                downsample_counter_second <=0;
                end //end if rst
        else
        begin

        case(state)    //State machine

                CLEAN :begin
                                        //Nothing in this clock domain
                                end

                FILL :
                begin
                //Fill buffer from ADC samples based on downsample counter.
                //If buffer full. Raise flag for other clock domain.
                        if(write_addr == 8'b10011111)
                        begin
                                buffer_full <= 1;
                                //state <= DISPLAY;
                        end//end if writeadd == 8'b111111
                        else
                        begin
                                if(second_channel)

                                        begin
                                                if(downsample_counter_second == 0)
                                                        begin

                                                        if(in_y[7])
                                                                ram_y[write_addr] <= 8'b10000000;
                                                        else
                                                                ram_y[write_addr] <= in_y;

                                                        write_addr <= write_addr + 1;

                                                        end

                                        end
                                else
                                        begin
                                                if(downsample_counter == 0)
                                                        begin

                                                                if(in_y2[7])
                                                                        ram_y2[write_addr] <= 8'b10000000;
                                                                else

                                                                        ram_y2[write_addr] <= in_y2;
```

```
                                                write_addr <= write_addr + 1;
                                        end
                                end

                        end//end else
                end     //end fill case

                DISPLAY:        begin
                                        //Nothing in this clock domain

                                end

                END:            begin
                                        //Clear temporary registers for next cycle.
                                        write_addr <= 0;
                                        buffer_full <= 0;
                                        downsample_counter <= 0;
                                        downsample_counter_second <= 0;
                                end

                default: begin
                                        write_addr <= 0;
                                        buffer_full <= 0;
                                        downsample_counter <= 0;
                                        downsample_counter_second <= 0;
                                end //end default case

                endcase //endcase state

                //Downsample for horizontal time base adjustment
                if(downsample_counter == downsample_sel)
                        downsample_counter <= 0;     //Reset counter based on user input
                else
                        downsample_counter <= downsample_counter + 1;

                if(downsample_counter_second == downsample_sel_second)
                        downsample_counter_second <= 0; //Reset counter based on user input
                else
                        downsample_counter_second <= downsample_counter_second + 1;

        end//end begin of else of if(rst_n)
end //end always @ posedge clk 65MHz


//Second clock domain. 25MHz.
always @ (posedge clk_25)
        begin
        if(!rst_n)
        begin
        //Clear registers on reset
                display_clean <=0;
                read_addr <= 0;
                write_en <= 0;
                read_addr <= 0;
                write_en <= 0;
                out_y <= 0;
                out_x <= 0;

        end//end if rst_n

        else
        begin
                case (state)   //State machine

                CLEAN :
                begin
                //Read buffer again. Clear those pixels by
                //replacing with black for background
                //or white if grid location.
                //Once cleaned. Next state is FILL to refill buffer.
                                if(read_addr == 8'b10011111)
```

```
                                begin
                                      display_clean <= 1;
                                      state <= FILL;
                                 end  //end (if(read_addr == 8'b111111))
                                else
                                      begin
                                            if(second_channel)
                                                  begin
                                                  out_y <= ram_y[read_addr];

                                                  end
                                            else
                                                  begin
                                                  out_y <= ram_y2[read_addr];


                                                  end

                                            read_addr <= read_addr + 1;
                                            out_x <= read_addr;

                                            write_en <= 1;
                                            //Check if GRID location
                                            if(read_addr == 0 || read_addr == 20  || read_addr == 40  ||
read_addr == 60  || read_addr == 80  || read_addr == 100 || read_addr == 120
                                             || read_addr == 140  || read_addr == 160 || out_y == 0 ||
out_y == 19 || out_y == 39 || out_y == 59 || out_y == 79
                                             || out_y == 99 || out_y == 119)
                                            RGB <= 12'b111111111111;

                                            else
                                            RGB <= 12'b000000000000;

                                      end

                  end    //end CLEAN case

                  FILL : begin
                              //Clear temporary registers.
                              //Wait for buffer flag to go high
                              //by the other clock domain.
                                          read_addr <= 0;
                                          write_en <= 0;
                                          read_addr <= 0;
                                          write_en <= 0;
                                          out_y <= 0;
                                          out_x <= 0;

                                          if(buffer_full)
                                                state <= DISPLAY;
                                          else
                                                state <= FILL;

                              end    //end FILL case

                  DISPLAY :
                  begin

                  //Read buffer. Output to VGA block to display_clean
                         if(read_addr == 8'b10011110)
                           begin
                                state <= END;
                           end  //end (if(read_addr == 8'b111111))
                          else
                          begin

                                if(second_channel)
                                begin
                                      out_y <= ram_y[read_addr];

                                      RGB <= 12'b000000111111;
                                 end
```

```
                                else
                                begin
                                        out_y <= ram_y2[read_addr];

                                        RGB <= 12'b111111000000;
                                end     //end if second_channel

                                read_addr <= read_addr + 1;
                                out_x <= read_addr;
                                write_en <= 1;


                        end   //end else

                end     //end DISPLAY case

                END:            begin
                                        //Clear temporary registers
                                        //Wait for trigger flag to go high
                                        //signaling another loop for the state machine.
                                                read_addr <= 0;
                                                write_en <= 0;
                                                out_y <= 0;
                                                out_x <= 0;

                                                if(trigger_bit == 1)
                                                begin
                                                        state <= CLEAN;
                                                        second_channel <= 1;
                                                end

                                                else if(trigger_bit_second == 1)
                                                        begin
                                                                state <= CLEAN;
                                                                second_channel <= 0;
                                                        end
                                                else
                                                                state <= END;

                                        end     //end END case

                default: begin
                                                read_addr <= 0;
                                                write_en <= 0;
                                                out_y <= 0;
                                                out_x <= 0;
                                        end
                endcase

        end //end else of if rst
end     //end always @ posedge clk 65MHz

//Trigger Channel 1
always @ (posedge trigger)
begin

if(trigger_sel == 1)
        begin//If external trigger is selected
        if(state == END && trigger_ext^trigger_ext_prev)
                begin //Raise flag if edge is detected
                        trigger_bit <= 1;
                end
        else
                begin
                        trigger_bit <= 0;
                        trigger_ext_prev <= trigger_ext;
                end
        end
else
begin//If auto trigger is selected
        if(state == END && trigger_counter == 0)
                trigger_bit <= 1;            //Raise flag if counter is zero. downsample trigger.
```

```
        else
                trigger_bit <= 0;
end

//Keep trigger counter running always
trigger_counter <= trigger_counter + 1;
end


//Trigger Channel 2
always @ (posedge trigger_second)
begin

if(trigger_sel_second == 1)
        begin  //If external trigger is selected
        if(state == END && trigger_ext_second^trigger_ext_prev_second)
                begin  //Raise flag if edge is detected
                        trigger_bit_second <= 1;
                end
        else
                begin
                        trigger_bit_second <= 0;
                        trigger_ext_prev_second <= trigger_ext_second;
                end
        end
else
begin  //If auto trigger is selected

        if(state == END && trigger_counter_second == 0)
                trigger_bit_second <= 1;     //Raise flag if counter is zero. downsample trigger.
        else
                trigger_bit_second <= 0;
end

//Keep trigger counter running always
trigger_counter_second <= trigger_counter_second + 1;
end

endmodule
```

Appendix C ADC block

The following code it the ADC block with firmware for the second channel added to it.

```
// CODE TO INTERFACE WITH THE DSP BOARD
module dsp(
                GPIO_0_TEMP,                    //      GPIO Connection 0
                GPIO_1_TEMP,                    //      GPIO Connection 1
                adc_b,                  //      Output Data from ADC B
                  adc_a,                        //      Output Data from ADC A   //ZLK
               dac_b,                           //      Input Data towards DAC B
         CLK_DAC,                       //      DAC Sampling clock
                CLK_ADC);                       //      ADC Sampling clock

//////////////////////// GPIO    ////////////////////////////////
inout   [35:0]  GPIO_0_TEMP;                                    //      GPIO Connection 0
inout   [35:0]  GPIO_1_TEMP;                                    //      GPIO Connection 1

//////////////////////  ADC DAC I/O   ////////////////////////////
output  [13:0]adc_b;
output  [13:0]adc_a;    //ZLK
input   [13:0]dac_b;

////////////////////    SAMPLING FREQUENCIES ///////////////////////
input CLK_DAC, CLK_ADC;


//////////////////            GPIO MAPPING            ////////////////////////
assign  adc_b =
{GPIO_0_TEMP[15],GPIO_0_TEMP[13],GPIO_0_TEMP[14],GPIO_0_TEMP[12],GPIO_0_TEMP[11],GPIO_0_TEMP[9],GPIO_0_TEMP[10],GPIO_0
_TEMP[8],
                GPIO_0_TEMP[7],GPIO_0_TEMP[5],GPIO_0_TEMP[6],GPIO_0_TEMP[4],GPIO_0_TEMP[3],GPIO_0_TEMP[1]};
```

```
assign  adc_a =
{GPIO_0_TEMP[31],GPIO_0_TEMP[29],GPIO_0_TEMP[30],GPIO_0_TEMP[28],GPIO_0_TEMP[27],GPIO_0_TEMP[25],GPIO_0_TEMP[26],GPIO_
0_TEMP[24],
                GPIO_0_TEMP[23],GPIO_0_TEMP[21],GPIO_0_TEMP[22],GPIO_0_TEMP[20],GPIO_0_TEMP[19],GPIO_0_TEMP[17]};
        //ZLK

assign
{GPIO_1_TEMP[19],GPIO_1_TEMP[21],GPIO_1_TEMP[22],GPIO_1_TEMP[24],GPIO_1_TEMP[23],GPIO_1_TEMP[25],GPIO_1_TEMP[27],GPIO_
1_TEMP[29],
         GPIO_1_TEMP[26],GPIO_1_TEMP[28],GPIO_1_TEMP[31],GPIO_1_TEMP[33],GPIO_1_TEMP[30],GPIO_1_TEMP[32]} = dac_b; //B

assign  GPIO_1_TEMP[34] = CLK_DAC;              //Input write signal for PORT B
assign  GPIO_1_TEMP[17] = CLK_DAC;              //Input write signal for PORT A

assign  GPIO_1_TEMP[35] = 1;                    //Mode Select. 1 = dual port, 0 = interleaved.

assign  GPIO_0_TEMP[32] = 1;                    //POWER ON

assign  GPIO_0_TEMP[33] = 0;                    //Enable B
assign  GPIO_0_TEMP[35] = 0;                    //ZLK. Channel A ENABLED!!!

assign  GPIO_1_TEMP[18] = CLK_DAC;              //PLL Clock to DAC_B
assign  GPIO_1_TEMP[16] = CLK_DAC;              //PLL Clock to DAC_A

assign  GPIO_0_TEMP[18] = CLK_ADC;              //PLL Clock to ADC_B
assign  GPIO_0_TEMP[16] = CLK_ADC;              //PLL Clock to ADC_A

endmodule
```

Appendix D Alphanumeric Display block

The following code implements the Alphanumeric display module using the driver IP and test code that came with the DE2 disk[v].

The code implements a simple edge detection to detect changes in the switch inputs. Every time a switch input is detected, the display is refreshed. Case statements are used to pass fixed ASCII codes to the display.

```
module  LCD_TEST (      //      Host Side
                        iCLK,iRST_N,
                        //      LCD Side
                        LCD_DATA,LCD_RW,LCD_EN,LCD_RS,
                        A_Sel_Y,
                        A_Sel_X,
                        B_Sel_Y,
                        B_Sel_X);
//      Host Side
input                   iCLK,iRST_N;

input [2:0]A_Sel_X;
input [2:0]A_Sel_Y;
input [2:0]B_Sel_X;
input [2:0]B_Sel_Y;

//      LCD Side
output [7:0]   LCD_DATA;
output                  LCD_RW,LCD_EN,LCD_RS;
//      Internal Wires/Registers
reg     [5:0]   LUT_INDEX;
reg     [8:0]   LUT_DATA;
reg     [5:0]   mLCD_ST;
reg     [17:0]  mDLY;
reg                     mLCD_Start;
reg     [7:0]   mLCD_DATA;
reg                     mLCD_RS;
wire            mLCD_Done;
reg refresh;
reg [2:0]A_Sel_X_temp;
reg [2:0]A_Sel_Y_temp;
reg [2:0]B_Sel_X_temp;
reg [2:0]B_Sel_Y_temp;

parameter       LCD_INTIAL      =       0;
parameter       LCD_LINE1       =       5;
parameter       LCD_CH_LINE     =       LCD_LINE1+16;
parameter       LCD_LINE2       =       LCD_LINE1+16+1;
```

```
parameter      LUT_SIZE       =      LCD_LINE1+32+1;

always@(posedge iCLK or negedge iRST_N)
begin
if(!iRST_N)
begin
       LUT_INDEX      <=     0;
       mLCD_ST        <=     0;
       mDLY           <=     0;
       mLCD_Start     <=     0;
       mLCD_DATA      <=     0;
       mLCD_RS        <=     0;
end
else
begin
       if(LUT_INDEX<LUT_SIZE)
       begin
              case(mLCD_ST)
              0:     begin
                            mLCD_DATA      <=     LUT_DATA[7:0];
                            mLCD_RS        <=     LUT_DATA[8];
                            mLCD_Start     <=     1;
                            mLCD_ST        <=     1;
                     end
              1:     begin
                            if(mLCD_Done)
                            begin
                                   mLCD_Start     <=     0;
                                   mLCD_ST        <=     2;
                            end
                     end
              2:     begin
                            if(mDLY<18'h3FFFE)
                            mDLY    <=     mDLY+1;
                            else
                            begin
                                   mDLY    <=     0;
                                   mLCD_ST<=     3;
                            end
                     end
              3:     begin
                            LUT_INDEX      <=     LUT_INDEX+1;
                            mLCD_ST<=     0;
                     end
              endcase
       end
       else
              begin
                     if(refresh == 1)
                            LUT_INDEX<= 0;
              end

              //EDGE DETECTION FOR REFRESH
              A_Sel_Y_temp <= A_Sel_Y;
              A_Sel_X_temp <= A_Sel_X;
              B_Sel_Y_temp <= B_Sel_Y;
              B_Sel_X_temp <= B_Sel_X;

              if(A_Sel_X_temp != A_Sel_X || A_Sel_Y_temp != A_Sel_Y || B_Sel_X_temp != B_Sel_X ||
B_Sel_Y_temp != B_Sel_Y)
                     refresh <= 1;
              else
                     refresh <= 0;
end
end

always @ (*)
begin
case(LUT_INDEX)
//     Initial
LCD_INTIAL+0:  LUT_DATA       <=     9'h038;
LCD_INTIAL+1:  LUT_DATA       <=     9'h00C;
```

```
LCD_INTIAL+2:  LUT_DATA       <=     9'h001;
LCD_INTIAL+3:  LUT_DATA       <=     9'h006;
LCD_INTIAL+4:  LUT_DATA       <=     9'h080;
//     Line 1
LCD_LINE1+0:   LUT_DATA       <=     9'h141;//A     //     0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
                                                    // A : 0 . 0 0 0  0  V   0  .  0  0   u S
LCD_LINE1+1:   LUT_DATA       <=     9'h13A;//:
LCD_LINE1+2:   case (A_Sel_Y)
                                     3'b000:LUT_DATA       <=     9'h130;// depends on input
                                     3'b001:LUT_DATA       <=     9'h130;// depends on input
                                     3'b010:LUT_DATA       <=     9'h130;// depends on input
                                     3'b011:LUT_DATA       <=     9'h130;// depends on input
                                     3'b100:LUT_DATA       <=     9'h130;// depends on input
                                     3'b101:LUT_DATA       <=     9'h130;// depends on input
                                     3'b110:LUT_DATA       <=     9'h131;// depends on input
                                     3'b111:LUT_DATA       <=     9'h132;// depends on input
                                     endcase
LCD_LINE1+3:   LUT_DATA       <=     9'h12E;//.
LCD_LINE1+4:   case (A_Sel_Y)
                                     3'b000:LUT_DATA       <=     9'h132;// depends on input
                                     3'b001:LUT_DATA       <=     9'h131;// depends on input
                                     3'b010:LUT_DATA       <=     9'h130;// depends on input
                                     3'b011:LUT_DATA       <=     9'h130;// depends on input
                                     3'b100:LUT_DATA       <=     9'h132;// depends on input
                                     3'b101:LUT_DATA       <=     9'h135;// depends on input
                                     3'b110:LUT_DATA       <=     9'h130;// depends on input
                                     3'b111:LUT_DATA       <=     9'h130;// depends on input
                                     endcase
LCD_LINE1+5:   case (A_Sel_Y)
                                     3'b000:LUT_DATA       <=     9'h135;// depends on input
                                     3'b001:LUT_DATA       <=     9'h132;// depends on input
                                     3'b010:LUT_DATA       <=     9'h136;// depends on input
                                     3'b011:LUT_DATA       <=     9'h133;// depends on input
                                     3'b100:LUT_DATA       <=     9'h135;// depends on input
                                     3'b101:LUT_DATA       <=     9'h130;// depends on input
                                     3'b110:LUT_DATA       <=     9'h130;// depends on input
                                     3'b111:LUT_DATA       <=     9'h130;// depends on input
                                     endcase
LCD_LINE1+6:   case (A_Sel_Y)
                                     3'b000:LUT_DATA       <=     9'h130;// depends on input
                                     3'b001:LUT_DATA       <=     9'h135;// depends on input
                                     3'b010:LUT_DATA       <=     9'h132;// depends on input
                                     3'b011:LUT_DATA       <=     9'h131;// depends on input
                                     3'b100:LUT_DATA       <=     9'h130;// depends on input
                                     3'b101:LUT_DATA       <=     9'h130;// depends on input
                                     3'b110:LUT_DATA       <=     9'h130;// depends on input
                                     3'b111:LUT_DATA       <=     9'h130;// depends on input
                                     endcase
LCD_LINE1+7:   case (A_Sel_Y)
                                     3'b000:LUT_DATA       <=     9'h130;// depends on input
                                     3'b001:LUT_DATA       <=     9'h130;// depends on input
                                     3'b010:LUT_DATA       <=     9'h135;// depends on input
                                     3'b011:LUT_DATA       <=     9'h132;// depends on input
                                     3'b100:LUT_DATA       <=     9'h130;// depends on input
                                     3'b101:LUT_DATA       <=     9'h130;// depends on input
                                     3'b110:LUT_DATA       <=     9'h130;// depends on input
                                     3'b111:LUT_DATA       <=     9'h130;// depends on input
                                     endcase
LCD_LINE1+8:   LUT_DATA       <=     9'h156;// V
LCD_LINE1+9:   LUT_DATA       <=     9'h120;// blank space
LCD_LINE1+10:  case (A_Sel_X)
                                     3'b000:LUT_DATA       <=     9'h130;// depends on input
                                     3'b001:LUT_DATA       <=     9'h130;// depends on input
                                     3'b010:LUT_DATA       <=     9'h131;// depends on input
                                     3'b011:LUT_DATA       <=     9'h131;// depends on input
                                     3'b100:LUT_DATA       <=     9'h131;// depends on input
                                     3'b101:LUT_DATA       <=     9'h132;// depends on input
                                     3'b110:LUT_DATA       <=     9'h132;// depends on input
                                     3'b111:LUT_DATA       <=     9'h132;// depends on input
                                     endcase
LCD_LINE1+11:  LUT_DATA       <=     9'h12E;// .
LCD_LINE1+12:  case (A_Sel_X)
```

```
                                3'b000: LUT_DATA       <=     9'h133;// depends on input
                                3'b001: LUT_DATA       <=     9'h136;// depends on input
                                3'b010: LUT_DATA       <=     9'h130;// depends on input
                                3'b011: LUT_DATA       <=     9'h133;// depends on input
                                3'b100: LUT_DATA       <=     9'h136;// depends on input
                                3'b101: LUT_DATA       <=     9'h130;// depends on input
                                3'b110: LUT_DATA       <=     9'h133;// depends on input
                                3'b111: LUT_DATA       <=     9'h136;// depends on input
                                endcase
LCD_LINE1+13:  case (A_Sel_X)
                                3'b000: LUT_DATA       <=     9'h133;// depends on input
                                3'b001: LUT_DATA       <=     9'h136;// depends on input
                                3'b010: LUT_DATA       <=     9'h130;// depends on input
                                3'b011: LUT_DATA       <=     9'h133;// depends on input
                                3'b100: LUT_DATA       <=     9'h136;// depends on input
                                3'b101: LUT_DATA       <=     9'h130;// depends on input
                                3'b110: LUT_DATA       <=     9'h133;// depends on input
                                3'b111: LUT_DATA       <=     9'h136;// depends on input
                                endcase
LCD_LINE1+14:  LUT_DATA       <=     9'h175;// u
LCD_LINE1+15:  LUT_DATA       <=     9'h173;// s
//     Change Line
LCD_CH_LINE:   LUT_DATA       <=     9'h0C0;
//     Line 2
LCD_LINE2+0:   LUT_DATA       <=     9'h142;//B     //     0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
                                                          // A : 0 . 0 0 0  0  V   0  .  0  0   u S
LCD_LINE2+1:   LUT_DATA       <=     9'h13A;//:
LCD_LINE2+2:   case (B_Sel_Y)
       3'b000: LUT_DATA       <=     9'h130;// depends on input
       3'b001: LUT_DATA       <=     9'h130;// depends on input
       3'b010: LUT_DATA       <=     9'h130;// depends on input
       3'b011: LUT_DATA       <=     9'h130;// depends on input
       3'b100: LUT_DATA       <=     9'h130;// depends on input
       3'b101: LUT_DATA       <=     9'h130;// depends on input
       3'b110: LUT_DATA       <=     9'h131;// depends on input
       3'b111: LUT_DATA       <=     9'h132;// depends on input
       endcase
LCD_LINE2+3:   LUT_DATA       <=     9'h12E;//.65;
LCD_LINE2+4:   case (B_Sel_Y)
       3'b000: LUT_DATA       <=     9'h132;// depends on input
       3'b001: LUT_DATA       <=     9'h131;// depends on input
       3'b010: LUT_DATA       <=     9'h130;// depends on input
       3'b011: LUT_DATA       <=     9'h130;// depends on input
       3'b100: LUT_DATA       <=     9'h132;// depends on input
       3'b101: LUT_DATA       <=     9'h135;// depends on input
       3'b110: LUT_DATA       <=     9'h130;// depends on input
       3'b111: LUT_DATA       <=     9'h130;// depends on input
       endcase
LCD_LINE2+5:   case (B_Sel_Y)
       3'b000: LUT_DATA       <=     9'h135;// depends on input
       3'b001: LUT_DATA       <=     9'h132;// depends on input
       3'b010: LUT_DATA       <=     9'h136;// depends on input
       3'b011: LUT_DATA       <=     9'h133;// depends on input
       3'b100: LUT_DATA       <=     9'h135;// depends on input
       3'b101: LUT_DATA       <=     9'h130;// depends on input
       3'b110: LUT_DATA       <=     9'h130;// depends on input
       3'b111: LUT_DATA       <=     9'h130;// depends on input
       endcase
LCD_LINE2+6:   case (B_Sel_Y)
       3'b000: LUT_DATA       <=     9'h130;// depends on input
       3'b001: LUT_DATA       <=     9'h135;// depends on input
       3'b010: LUT_DATA       <=     9'h132;// depends on input
       3'b011: LUT_DATA       <=     9'h131;// depends on input
       3'b100: LUT_DATA       <=     9'h130;// depends on input
       3'b101: LUT_DATA       <=     9'h130;// depends on input
       3'b110: LUT_DATA       <=     9'h130;// depends on input
       3'b111: LUT_DATA       <=     9'h130;// depends on input
       endcase
LCD_LINE2+7:   case (B_Sel_Y)
       3'b000: LUT_DATA       <=     9'h130;// depends on input
       3'b001: LUT_DATA       <=     9'h130;// depends on input
       3'b010: LUT_DATA       <=     9'h135;// depends on input
```

```
        3'b011: LUT_DATA        <=      9'h132;// depends on input
        3'b100: LUT_DATA        <=      9'h130;// depends on input
        3'b101: LUT_DATA        <=      9'h130;// depends on input
        3'b110: LUT_DATA        <=      9'h130;// depends on input
        3'b111: LUT_DATA        <=      9'h130;// depends on input
        endcase
LCD_LINE2+8:    LUT_DATA        <=      9'h156;// V
LCD_LINE2+9:    LUT_DATA        <=      9'h120;// blank space32;
LCD_LINE2+10:   case (B_Sel_X)
        3'b000: LUT_DATA        <=      9'h130;// depends on input
        3'b001: LUT_DATA        <=      9'h130;// depends on input
        3'b010: LUT_DATA        <=      9'h131;// depends on input
        3'b011: LUT_DATA        <=      9'h131;// depends on input
        3'b100: LUT_DATA        <=      9'h131;// depends on input
        3'b101: LUT_DATA        <=      9'h132;// depends on input
        3'b110: LUT_DATA        <=      9'h132;// depends on input
        3'b111: LUT_DATA        <=      9'h132;// depends on input
        endcase
LCD_LINE2+11:   LUT_DATA        <=      9'h12E;// .
LCD_LINE2+12:   case (B_Sel_X)
        3'b000: LUT_DATA        <=      9'h133;// depends on input
        3'b001: LUT_DATA        <=      9'h136;// depends on input
        3'b010: LUT_DATA        <=      9'h130;// depends on input
        3'b011: LUT_DATA        <=      9'h133;// depends on input
        3'b100: LUT_DATA        <=      9'h136;// depends on input
        3'b101: LUT_DATA        <=      9'h130;// depends on input
        3'b110: LUT_DATA        <=      9'h133;// depends on input
        3'b111: LUT_DATA        <=      9'h136;// depends on input
        endcase
LCD_LINE2+13:   case (B_Sel_X)
        3'b000: LUT_DATA        <=      9'h133;// depends on input
        3'b001: LUT_DATA        <=      9'h136;// depends on input
        3'b010: LUT_DATA        <=      9'h130;// depends on input
        3'b011: LUT_DATA        <=      9'h133;// depends on input
        3'b100: LUT_DATA        <=      9'h136;// depends on input
        3'b101: LUT_DATA        <=      9'h130;// depends on input
        3'b110: LUT_DATA        <=      9'h133;// depends on input
        3'b111: LUT_DATA        <=      9'h136;// depends on input
        endcase
LCD_LINE2+14:   LUT_DATA        <=      9'h175;// u
LCD_LINE2+15:   LUT_DATA        <=      9'h173;// s
default:                LUT_DATA        <=      9'h000;
endcase
end

LCD_Controller                  u0      (       //      Host Side
                                                .iDATA(mLCD_DATA),
                                                .iRS(mLCD_RS),
                                                .iStart(mLCD_Start),
                                                .oDone(mLCD_Done),
                                                .iCLK(iCLK),
                                                .iRST_N(iRST_N),
                                                //      LCD Interface
                                                .LCD_DATA(LCD_DATA),
                                                .LCD_RW(LCD_RW),
                                                .LCD_EN(LCD_EN),
                                                .LCD_RS(LCD_RS)         );

endmodule
```

References

[i] http://en.wikipedia.org/wiki/Oscilloscope

[ii] Lab 6 Handout, ELEC5563 Digital Design of System on Chip, School of Electronic and Electrical Engineering, University of Leeds

[iii] AD/DA Daughter Card – GPIO configuration, Terasic

[iv] Lab 5 Handout, ELEC5563 Digital Design of System on Chip, School of Electronic and Electrical Engineering, University of Leeds

[v] DE2 System CD by Terasic