Alberto Rosas
Antonio Rodriguez
CMPE 323
7 December 2023

Lab 5: Simple Oscilloscope

Introduction:

The most important aspect of dealing with electronics is electricity. It is important to know the behavior of the electricity in any electrical system. Thus, a German physicist named Karl Ferdinand, went on to create the oscilloscope using a cathode ray tube. The oscilloscope measures an electrical voltage and converts it into a digital signal using an analog-to-digital converter. While the first commercial oscilloscope was not released until 1938, it has become a staple in the homes and labs of many engineers. In this lab, we are creating a simple version of the oscilloscope using an FPGA and a separate analog-to-digital converter.

Theory:

Experiment 1:

The A/D converter takes in 4 signals that we must generate. CONVST, CS, RD, and Busy. The CONVST signal is the longest so by using a counter to generate that signal we can develop the other 3 using the same counter. However, since BUSY is just notified when the A/D converter is in use, we can ignore it and code the time we need to wait into Verilog. To find the length of the CONVST signal to create a counter we converted the periods given into clock cycles by first finding the frequency, 1/Period, and then 100MHz frequency of the BASYS 3 clock divided by the frequency calculated previously. This gave us the number of clock cycles describing how each signal should act for 1 full conversation cycle. We ended up with a total clock cycle time of 555, 450 cycles of waiting,5 allowing for impedance before the RD and CS signals, 2 clock cycles for CS and RD to go low, and the buffer being written to in the second clock cycle. Then 98 clock cycles were arbitrarily chosen to wait after the read. We did not have to use the exact numbers given in the datasheet because they were minimums and did not account for the electrical impedance of these wires. Once the counter is implemented the 3 signals CONVST, RD, and CS can be created using an assign and if statement. Where the if statement declares the negative pulse width of each signal at their respective period. If done successfully, simulating and probing the signals will show us signals that fall under the specifications needed to be described in the A/D converter's datasheet.

Module

```
reg [8:0] CONVcount;
  assign convstb = (CONVcount < 3) ? 1'b0 : 1'b1;
  assign csb = (CONVcount > 454 && CONVcount < 457) ? 1'b0 : 1'b1;
  assign rdb = (CONVcount > 454 && CONVcount < 457) ? 1'b0 : 1'b1;

  always @(posedge clk or negedge rst) begin
    if (~rst) begin // if rst==0
      index <= 8'b0;
      CONVcount <= 9'd0;
    end
    else begin
        if(CONVcount < 555)begin     //450 +5 +2(read) +99 = 556
```

```
            CONVcount <= CONVcount + 9'b1;
          end else begin
            CONVcount <= 9'b0;
          end
        end
      end
    end
```

Test Module

```
oscilloscope gc0(clk,rst, capture, hsync,vsync, red, green, blue,convstb,csb,rdb,db);

    always begin
      #5 clk = ~clk; // trigger the clock every 10 ticks
    end

    initial begin
      rst = 1'b0; // reset the system
      clk = 1'b0; // set the initial value of the clock
      #5
      rst = 1'b1; // start the state clk
      #4000000000; // let it run for 400 million time ticks
    end

    always @(posedge clk) begin
      $display("clk=%b capture=%b convstb=%b csb=%b %d %x",clk,convict,csb,$time);
    end
endmodule
```

Experiment 2:

Once the signals needed for the A/D converter are in working order data must be read into the buffer and then outputted. To read we read once every CONVST signal when the counter is equal to 456. Every time a value is read the buffer increases filling up the buffer array. However, by adding a Capture input we can pause the reading into a full buffer by not resetting the index once the buffer is full and displaying only that data in a loop. Using the VGA module developed in LAb 4, to display the data we can think of the display as a graph where we must plot points. In this graph, the X-axis is the Hcounter(developed in Lab 4) and the Y-axis is the Vcounter. The buffer is size 620 which is the same as the X-axis. We can use this to declare the index of the buffer as the Hcounter minus the offset to display 1 point of data at every pixel across the screen. To plot the Y coordinate of each data point by creating an if statement where a pixel is colored when the Vsync equals the data value we should see the top of the screen have a line across it if the A/D converter is outputting 0 and a max of 255 pixels below that when a 333 voltage is applied to V.in. This makes it so that the graph is inverted so we need to invert it. The X-axis is correct so we need to only invert the Y-axis, this can be done by subtracting the buffer data from 330 then if the data is 0, 330-0=300 which draws a pixel near the middle of the screen. If the data is max 255 then 330-255 = 75 which is near the top of the screen. And all the other data points in between.

Module

```
module oscilloscope(
  input clk,
  input rst,
  input capture,
  output hsync,
  output vsync,
  output [3:0] red,
  output [3:0] green,
  output [3:0] blue,
  output convstb,
  output csb,
  output rdb,
  input [7:0] db);

  wire convstb;
  wire csb;
  wire rdb;

  reg captureOld;
  reg [9:0] index;
  reg [7:0] buffer[639:0]; // declares 640 elem array of 8-bit ints
  reg [8:0] CONVcount;
  assign convstb = (CONVcount < 3) ? 1'b0 : 1'b1;
  assign csb = (CONVcount > 454 && CONVcount < 457) ? 1'b0 : 1'b1;
  assign rdb = (CONVcount > 454 && CONVcount < 457) ? 1'b0 : 1'b1;

  always @(posedge clk or negedge rst) begin
     if (~rst) begin // if rst==0
        index <= 8'b0;
        CONVcount <= 9'd0;
     end
     else begin
        if(capture && ~captureOld && (index > 639)) begin    //rising edge
          index <= 8'b0;
        end else
        if (CONVcount == 456 && (index <= 639) )begin
           buffer[index] = db[7:0];
           index <= index + 8'b1;
           CONVcount <= CONVcount + 9'b1;
        end else begin
           if(CONVcount < 555)begin     //450 +5 +2(read) +99 = 556
              CONVcount <= CONVcount + 9'b1;
```

```
            end else begin
                CONVcount <= 9'b0;
            end
        end
    end
end

reg [1:0] Pcount;     // declare 2-bit register
reg [9:0] Hcount;   // declare 10-bit register
reg [9:0] Vcount;   // declare 10-bit register

reg [9:0] element;
reg [9:0] prev;
reg [9:0] x;   // declare 10-bit register
reg [9:0] y;   // declare 10-bit register

wire pixelClk;
wire hsync;
wire vsync;

reg [3:0] red;
reg [3:0] blue;
reg [3:0] green;

// Pulse width
assign pixelClk = (Pcount < 2'd2) ? 1'b0 : 1'b1; // 50%
assign hsync = (Hcount < 10'd96) ? 1'b0 : 1'b1; // uses if/else
assign vsync = (Vcount < 10'd2) ? 1'b0 : 1'b1; // uses if/else

// Pixel clock
always @(posedge clk) begin
    if (Pcount < 2'd3) begin
        Pcount <= Pcount + 4'd1;
    end else begin
        Pcount <= 2'd0; //reset
    end
end

// Hsync and vsync
always @(posedge pixelClk or negedge rst) begin
    if (~rst) begin // if rst==0
        Hcount <=10'd0;   // set the flip flops all equal to zero
        Vcount <= 10'd0;  // set the flip flops all equal to zero
    end else begin
        if (Hcount < 10'd799) begin     //Sync Pulse
            Hcount <= Hcount + 10'd1;
            y <= Hcount - 144;
         end else begin
            Hcount <= 10'd0;
            Vcount <= Vcount + 10'd1;
            if(Vcount >= 10'd520) begin     //Sync Pulse (new row)
               Vcount <= 10'd0;
                                                 x <= x + 10'b1;
            end
         end
      end
   end

   // x = hsync counter (Hcount)
   // y = vsync counter (Vcount)
   always @(posedge pixelClk) begin
      if( (Vcount>=10'd31 && Vcount<=10'd510) && (Hcount>=10'd144 &&
Hcount<=10'd783))begin       // Only display time Vsync and Only display time Hsyn
         element = Hcount - 10'd144;
         if( (Vcount == (331-buffer[element]))) begin
            red = 4'b1111;
            green = 4'b0000;
            blue = 4'b0000;
         end
         else begin
            red = 4'b1111;
            green = 4'b1111;
            blue = 4'b1111;
         end
               end
               else begin
                        red = 4'b0000;
                        green = 4'b0000;
                        blue = 4'b0000;
               end
   end
endmodule
```

Test Module

```
module testBench1();
   reg rst,clk,capture;
   reg [7:0] db;
   wire hsync;
   wire vsync;
   wire [3:0] red;
   wire [3:0] blue;
   wire [3:0] green;
   wire convstb;
   wire csb;
   wire rdb;

   oscilloscope gc0(clk,rst,capture,hsync,vsync,red,green,blue,convstb,csb,rdb,db);

   always begin
      #5 clk = ~clk; // trigger the clock every 10 ticks
   end

   initial begin
      rst = 1'b0; // reset the system
      clk = 1'b0; // set the initial value of the clock
      #5
      rst = 1'b1; // start the state clk
      capture = 1;
      db = 8'hff;
      #4000000000; // let it run for 400 million time ticks
   end

   always @(posedge clk) begin
      $display("clk=%b capture=%b convstb=%b csb=%b %d
%x",clk,capture,convstb,csb,$time,gc0.buffer[0]);
   end
endmodule
```

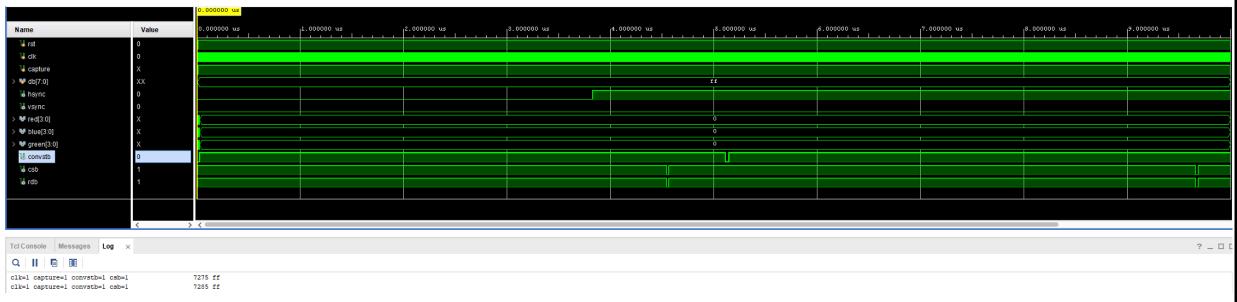
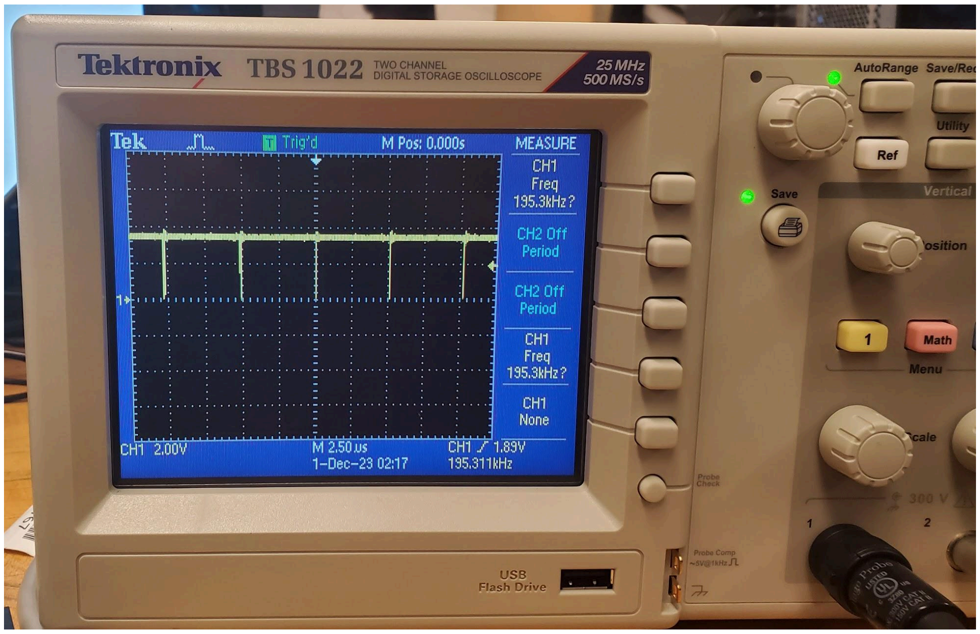
Experiment:

In all parts of the experimentation, we utilized the software Vivado 2021.1 and BASYS 3 FPGA by Xilinx. Other tools needed for this lab are the breadboard, a DC power generator, and a waveform generator. A VGA monitor and a VGA cable are used to display the measured electrical signal. We were provided a sample code for the first experiment that helped us declare the oscilloscope module. This module will read the digital values from the AD 7819 A/D converter, store the values in a buffer, and illuminate the pixels corresponding to each value in the buffer. A breadboard is used to connect the A/D converter to the BASYS3 FPGA. In the second experiment, we will utilize the VGA tools, DC power generator, and waveform generator to create and display an electrical signal.

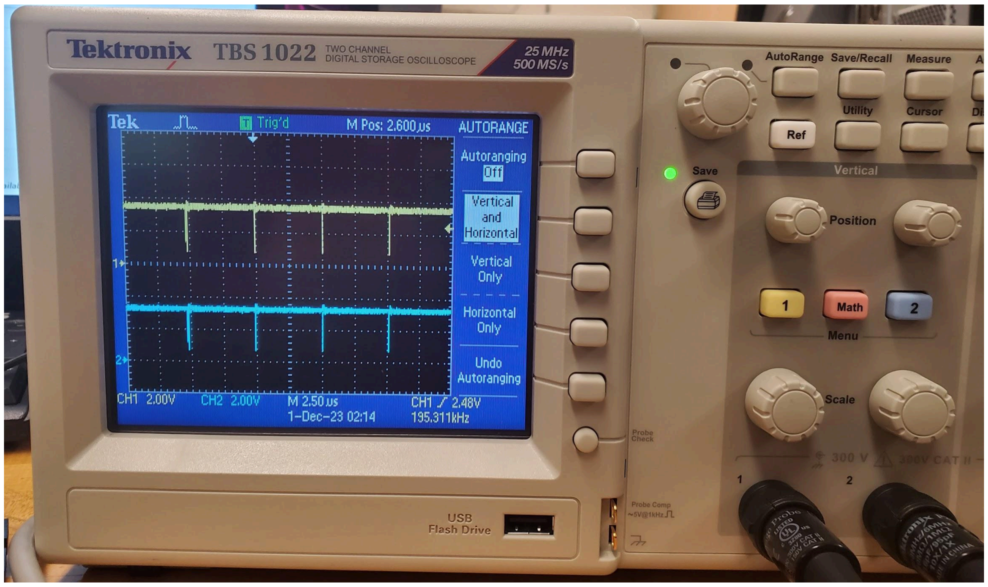
Hardware/Software	
Basys 3 (FPGA - Field Programmable Gate Array	https://www.mouser.com/datasheet/2/903/ds180_7Series_Overview-1591537.pdf https://digilent.com/reference/programmable-logic/basys-3/reference-manual
VGA Monitor	

VGA Cable	
Vivado 2021.1	https://www.xilinx.com/products/design-tools/vivado.html
AD7819 A/D Converter	https://www.analog.com/media/en/technical-documentation/data-sheets/AD7819.pdf
DC Power Supply	
Waveform Generator	
Breadboard	

Results:

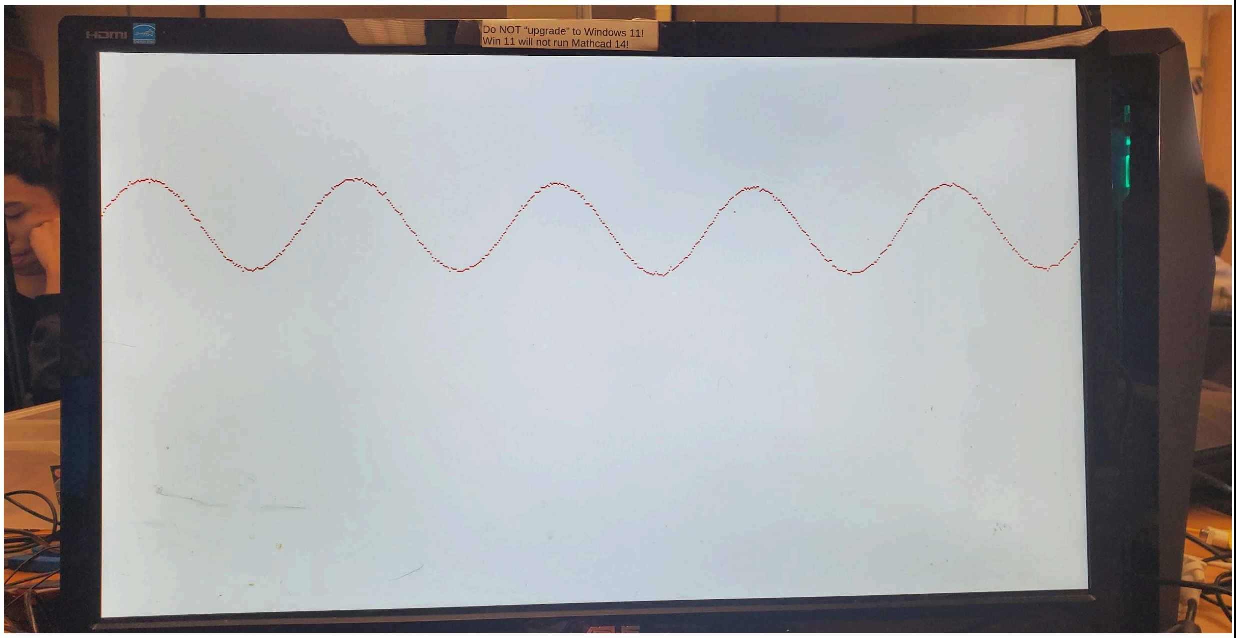
Experiment 1:
Simulation

Implementation


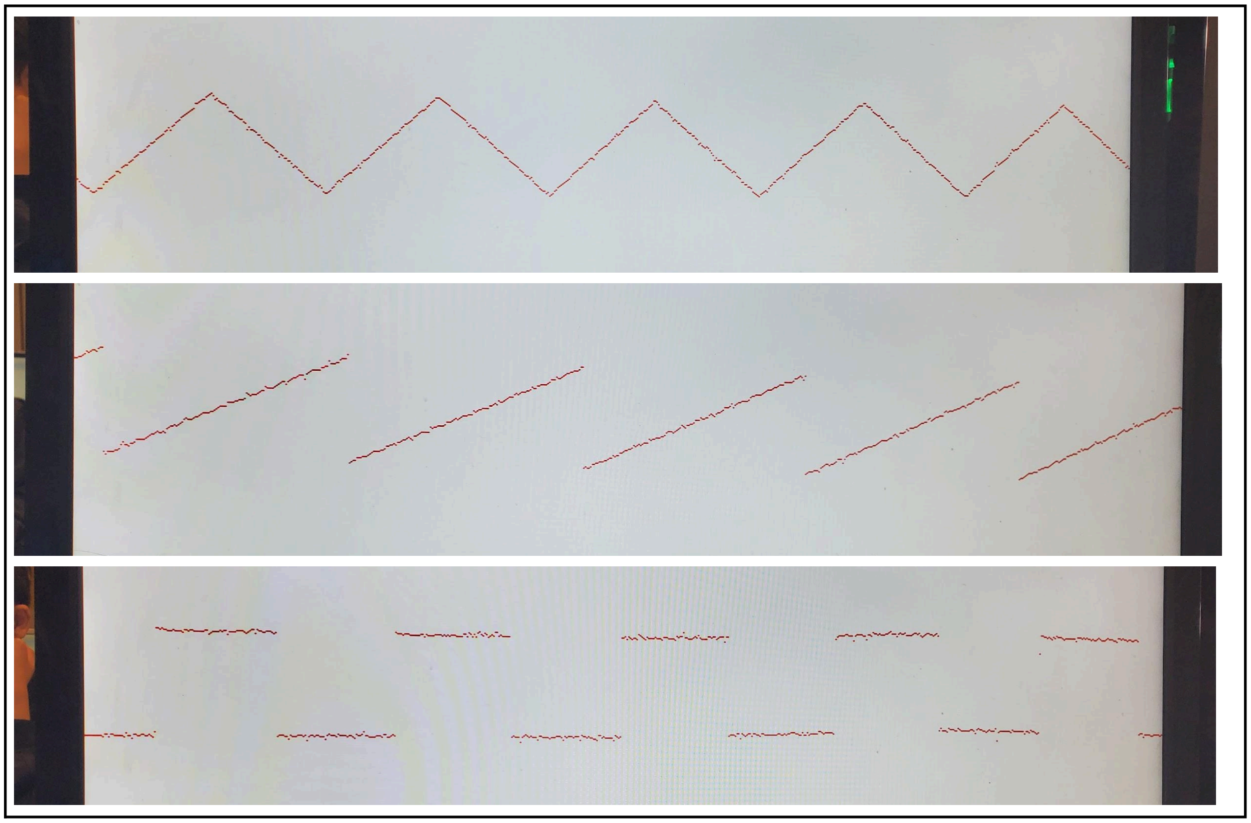
```
clk=1 capture=1 convstb=1 csb=1            7275 ff
clk=1 capture=1 convstb=1 csb=1            7285 ff
```


Tektronix TBS 1022 TWO CHANNEL DIGITAL STORAGE OSCILLOSCOPE
25 MHz 500 MS/s
Tek Trig'd M Pos: 2.600µs AUTORANGE
Autoranging Off
Vertical and Horizontal
Vertical Only
Horizontal Only
Undo Autoranging
CH1 2.00V CH2 2.00V M 2.50µs CH1 ⌐ 2.48V
1-Dec-23 02:14 195.311kHz
AutoRange Save/Recall Measure
Utility Cursor
Ref
Save
Vertical
Position
1 Math 2
Menu
Scale
USB Flash Drive

Experiment 2:

Implementation


Do NOT "upgrade" to Windows 11!
Win 11 will not run Mathcad 14!



Discussion:

Experiment 1: The biggest struggle we had when trying to conduct this experiment was analyzing the timings of the AD7819 A/D converter. The timings of this component were not as simple to understand as the timings for VGA. The documentation for VGA measured the delays for each signal, while the AD7819 only measured specific parts of each signal. This confused us because we could not figure out how long the delays were supposed to be. However, we discovered that exact timings were not needed, since the reading and writing times were variable. This allowed us to decide how long they were and adjust them whenever necessary.

Experiment 2: Something that we had trouble doing during the experiment was getting the right frequency. Because of this, we would get random pixels all over the screen. Once we found the right frequency, we implemented a capture system to freeze the waves at any point we wanted. Another issue we had was with inverted graphs, however, we fixed that by subtracting the buffer from 330. This meant that if the buffer was 0, then the points would be displayed in the middle of the screen (the bottom of the graph) and if it was 255, then it would be displayed by the top of the screen. Afterward, we had an issue where the points would fill the entire screen below them, so we decided to make everything white after the line
"if vctr=buffer[])".

Conclusion:

Through the development of a simple oscilloscope, we learned how to measure an electrical signal and properly utilize the information from an A/D converter, while simultaneously displaying the electrical signal to a screen using a buffer. The skills learned from this experiment, such as analyzing component timings and buffering information, are extremely useful for future projects. We can use these skills in similar projects where an analog signal needs to be measured and displayed. Some examples would be: measuring sound waves and radio waves.