Riley Guidry, rguidry2@horizon.csueastbay.edu
Dang Tran, dtran134@horizon.csueastbay.edu
CMPE 323
7 December 2023

VGA Oscilloscope Using A/D Converters

Lab 5

Introduction

Our previous lab granted us the ability to display images via VGA cable. In this lab, we will use this power to display analog voltages that have been converted to digital signals through an Analog to Digital Converter (ADC or A/D Converter). We used our BASYS 3 FPGA board as always for the VGA conversions.

Theory

The AD7819 is an ADC that is known as a successive approximation register (SAR) ADC. Its duty is to take in a specific voltage through one of its pins, and convert it to an 8 bit output that is dependent on a reference voltage. This 8 bit number can be read by a computer as an integer, and that integer can then be used as a vertical coordinate.

An oscilloscope is a device that outputs a graphical image that represents the input voltage over time. Advanced oscilloscopes are expensive, but also have fast reading times, and can display complicated waveforms. With the SAR ADC and VGA cable, we can make our own basic oscilloscope by writing the 8 bit number outputted by the ADC to the VGA converter, and then writing from the FPGA board, through VGA cable, onto a monitor.

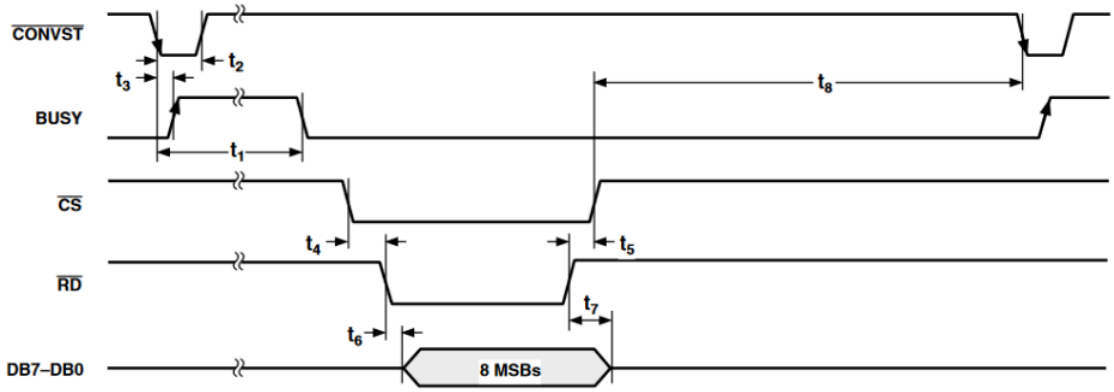
Procedure

The AD7819 takes time to convert the input voltage to its 8-bit output. There is a schematic pictured below of the digital timings of each of the signals in the ADC.

CONVST, to us, is simply a switch. It tells the ADC to start processing its input voltage. Once the ADC has started to convert, BUSY turns on for the entire duration of conversion. This is a flag, and we should mind when it is on, as to not disturb the ADC, or attempt to read our output before it has gone low again. CONVST will be ignored as long as BUSY is high, as well as CS and RD. CS and RD are activation switches, which control whether or not the window into viewing the 8 bit output is open. So, once BUSY is low, we have control over when we read our output, and for how long the window for reading the output is open. Finally, the ADC takes time to reset until it can read and convert its next voltage. The maximum speed of this whole process is 200 kilosamples per second, or one sample every 5 microseconds.

The timings of the ADC conversion mismatch with the VGA conversion timings. To store instantaneous values of the 8 bit output from the ADC, we use a buffer array instead of directly writing from the ADC to the VGA converter. This is purely to store the information and account for the difference in speeds in both of these operators.

The oscilloscope “lights up” at specific y-values depending on the 8-bit output. We pair this integer to VSYNC through a simple calculation to light up the desired coordinates.



Data

Experiment #1: Digital Signals of AD7819

AD7819 Timing Characteristics

Parameter	V_DD = 3 +/- 10%	V_DD = 5 +/- 10%	Unit	unit converted to sec	Clocks	Updated Clks	Conditions/Comments
~~t_power-up~~	~~1.5~~	~~1.5~~	~~us (max)~~	~~0.0000015~~	~~150~~		~~Power-Up time after CONVST rising edge~~
t_1	4.5	4.5	us (max)	0.0000045	450	450	Conversion Time (CONVST)
t_2	30	30	ns (min)	0.00000003	3	5	CONVST pulsewidth
t_3	30	30	ns (max)	0.00000003	3	1	CONVST falling edge to BUSY rising edge
t_4	0	0	ns (min)	0	0	0	CS to RD Setup Time
t_5	0	0	ns (min)	0	0	0	CS Hold Time after RD High
t_6	10	10	ns (max)	0.00000001	1	1	Data Access Time after RD Low
t_7	10	10	ns (max)	0.00000001	1	1	Bus Relinquish Time after RD High
t_8	100	100	ns (min)	0.0000001	10	80	Data Bus Relinquish to falling edge of CONVST delay
				Total Clocks:	468	538	
clock calculations based off of 100 MHz							

Source Code

```
module oscilloscope(
    input clk,
    input rst,
    output hsync,
    output vsync,
    output [3:0] red,
    output [3:0] green,
    output [3:0] blue,
    output convstb,        // convst = 0 or 1 (off / on)
    //input busy,          // may not be needed (works simultaneously w/ convst)
    output csb,
    output rdb,
    input [7:0] db);

    reg [1:0] pctr;
    reg [9:0] hctr;
    reg [9:0] vctr;
    reg [9:0] ADCctr;           // total clock cycles < 2^10 or 1024 clock cycles

    reg [7:0] buffer[639:0];    // declares 640 elem array of 8-bit ints

    reg [9:0] i;   // buffer index counter

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

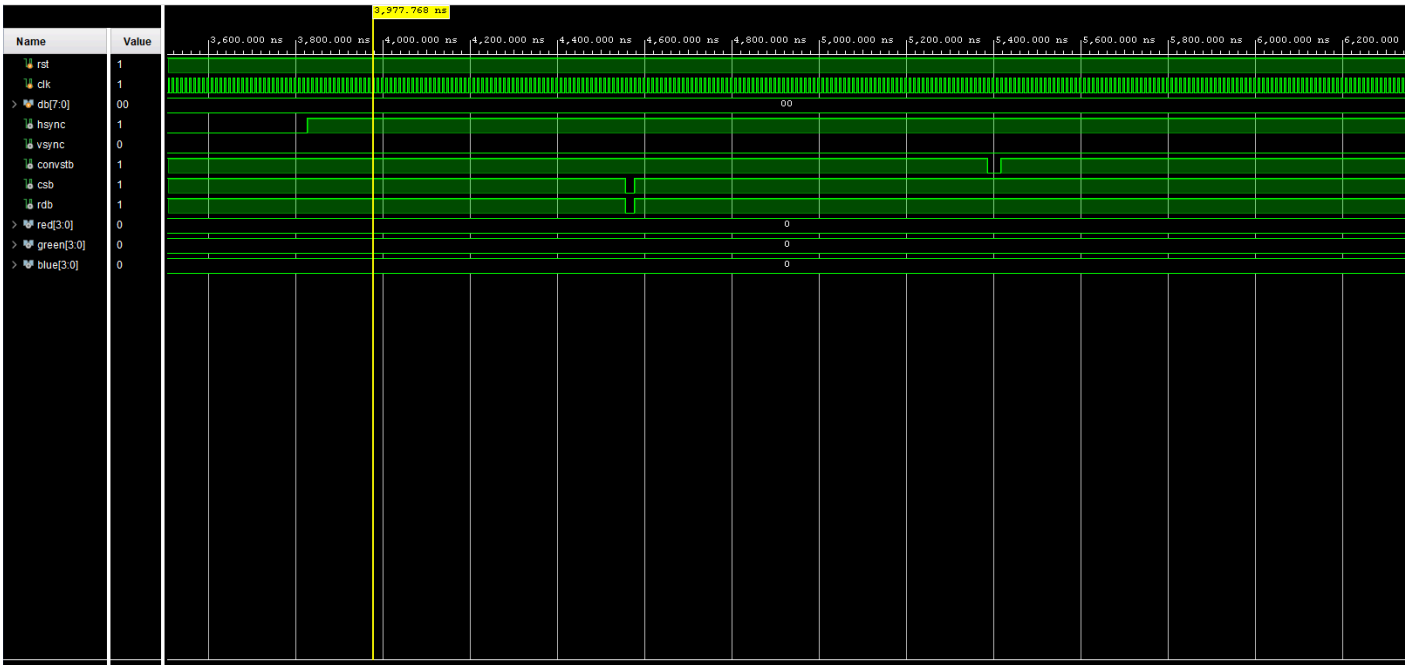
```
    reg [3:0] blue;
    wire pixel, hsync, vsync;

    wire convstb, csb, rdb;

    //450+5+1+1(read_time)+1+80 = 538 clks

    assign convstb = (ADCctr < 10'd3) ? 0 : 1;
    assign csb = ((ADCctr > 10'd454) && (ADCctr < 10'd457)) ? 0 : 1; // csb -> rdb
    assign rdb = ((ADCctr > 10'd454) && (ADCctr < 10'd457)) ? 0 : 1; // rdb -> csb

    always @(posedge clk or negedge rst) begin
        if (~rst) begin
            ADCctr <= 10'd0;
            i <= 10'd0;
        end else begin
            if (ADCctr < 10'd537) begin
                if (ADCctr == 10'd456) begin
                    if (i <= 10'd639) begin
                        buffer[i] <= db[7:0];
                        i <= i + 10'd1;
                    end else begin
                        i <= 10'd0;
                        buffer[i] <= db[7:0];
                    end
                end
                ADCctr <= ADCctr + 10'd1;
            end else begin
                ADCctr <= 10'd0;
            end
        end
    end

    always @(posedge clk) begin
        if (pctr < 2'd3) begin
            pctr <= pctr + 2'd1;
        end else begin
            pctr <= 2'd0;
        end
    end

    assign pixel = (pctr < 2'd2) ? 0 : 1;
    assign hsync = (hctr < 10'd96) ? 0 : 1;
    assign vsync = (vctr < 10'd2) ? 0 : 1;

    always @(posedge pixel or negedge rst) begin
        if (~rst) begin
            hctr <= 10'd0;
            vctr <= 10'd0;
        end else begin
            if (hctr < 10'd799) begin
                hctr <= hctr + 10'd1;
            end else begin
                hctr <= 10'd0;
                if (vctr < 10'd520) begin
                    vctr <= vctr + 10'd1;
                end else begin
                    vctr <= 10'd0;
                end
            end
        end
```

```
    end

    always @(posedge pixel) begin
        if (vctr >= 10'd31 && vctr <= 10'd510 && hctr >= 10'd144 && hctr <= 10'd783) begin
            if (vctr == (400 - buffer[hctr-144]) ) begin
                red <= 4'h0;
                green <= 4'hf;
                blue <= 4'h0;
            end else begin
                red <= 4'h0;
                green <= 4'h0;
                blue <= 4'h0;
            end
        end else begin
            red <= 4'h0;
            green <= 4'h0;
            blue <= 4'h0;
        end
    end
endmodule
```

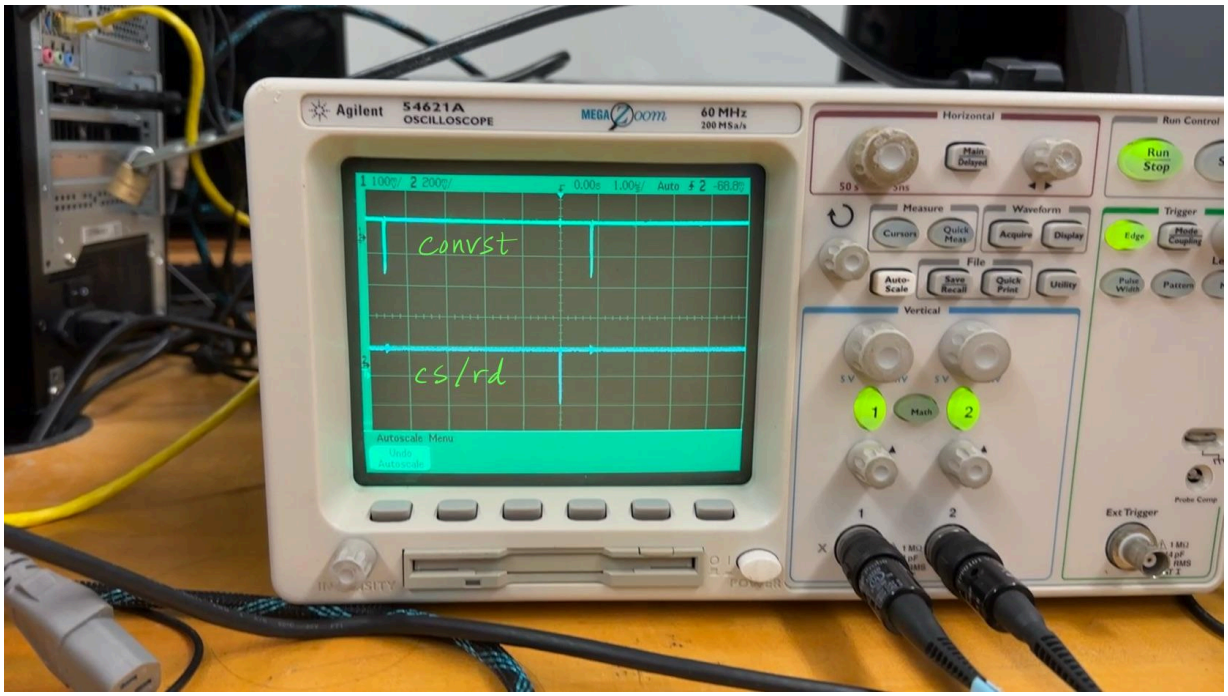
Testbench

```
module tb_exp1();
    reg rst,clk;
    reg [7:0] db;
    wire hsync,vsync;
    wire convstb,csb,rdb;
    wire [3:0] red;
    wire [3:0] green;
    wire [3:0] blue;

    oscilloscope a0(clk,rst,hsync,vsync,red,green,blue,convstb,csb,rdb,db);

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

    initial begin
        db = 8'b00000000;
        rst = 1'b0;            // reset the system
        clk = 1'b0;            // set the initial value of the clock
        #15
        rst = 1'b1;            // start the state machine
        #400000000;            // let it run for 400 million time ticks
    end

    always @(posedge clk) begin
        $display("hsync=%b vsync=%b convstb=%b csb=%b rdb=%b time=%d
buffer=%b",hsync,vsync,convstb,csb,rdb,$time,a0.buffer[0]);
    end
endmodule
```

Simulation



Oscilloscope Reading

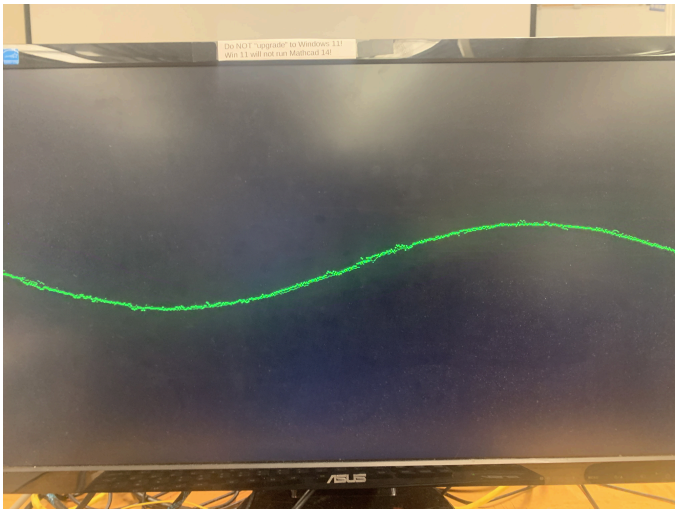
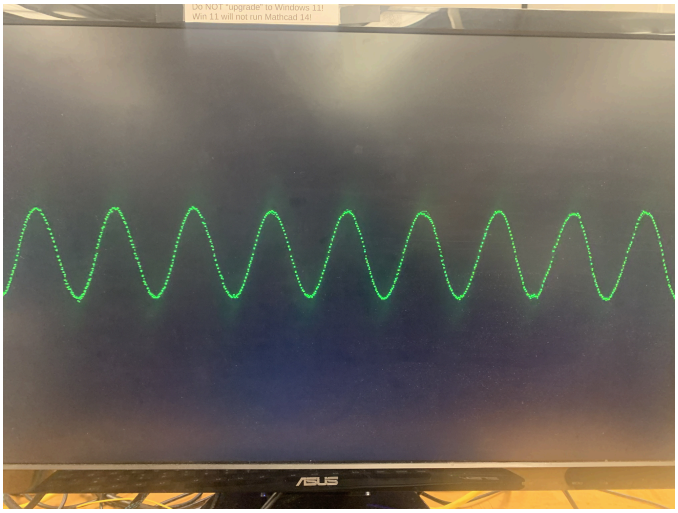


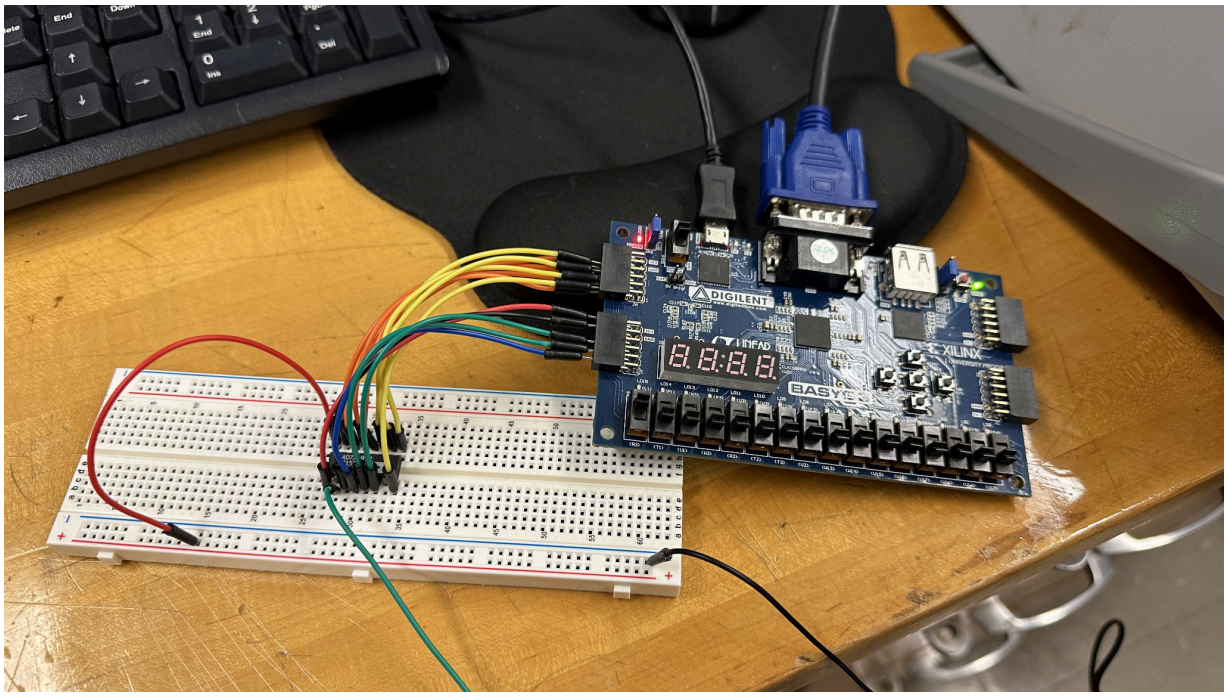
Experiment #2: Budget Oscilloscope

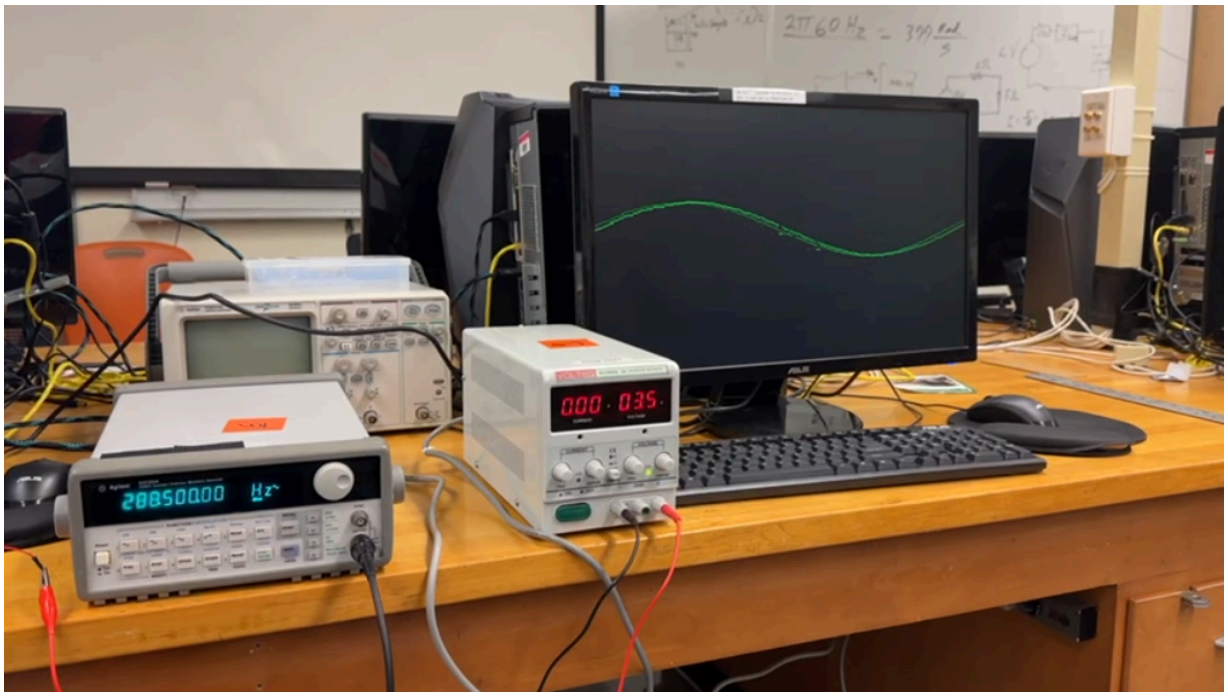
Source Code

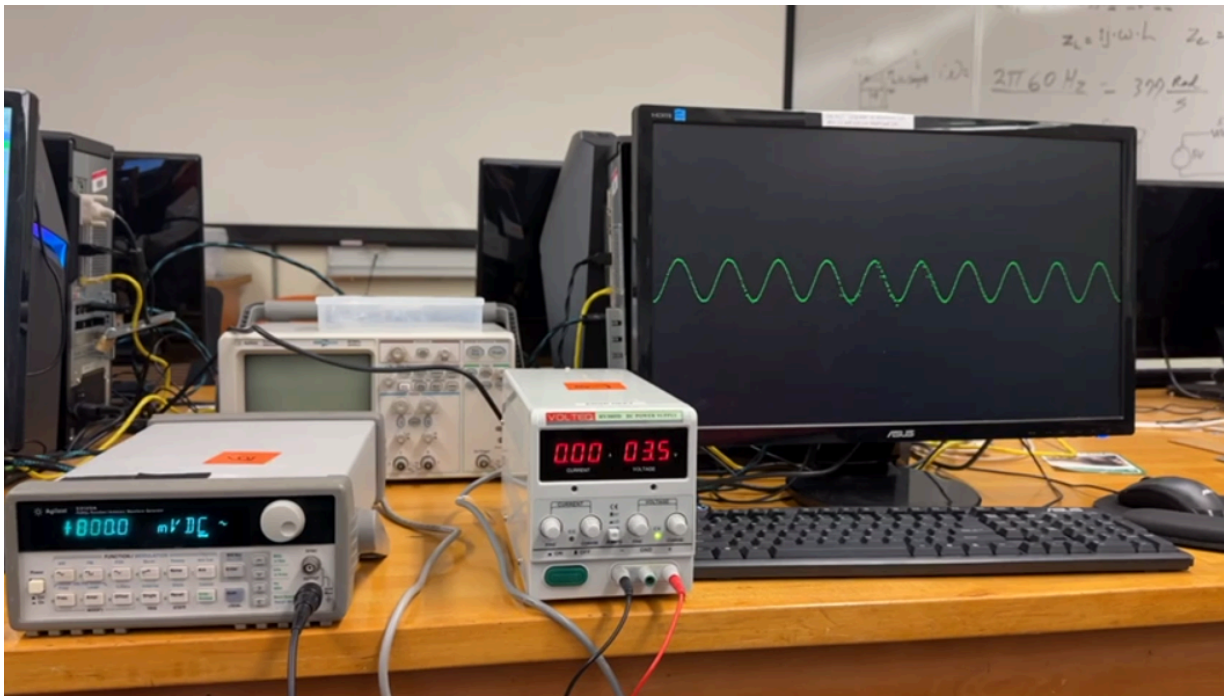
```
module oscilloscope(
    input clk,
    input rst,
    output hsync,
    output vsync,
    output [3:0] red,
    output [3:0] green,
    output [3:0] blue,
    output convstb,       // convst = 0 or 1 (off / on)
    //input busy,         // may not be needed (works simultaneously w/ convst)
    output csb,
    output rdb,
    input [7:0] db);

    reg [1:0] pctr;
    reg [9:0] hctr;
    reg [9:0] vctr;
    reg [9:0] ADCctr;           // total clock cycles < 2^10 or 1024 clock cycles

    reg [7:0] buffer[639:0];    // declares 640 elem array of 8-bit ints

    reg [9:0] i;   // buffer index counter

    reg [3:0] red;
    reg [3:0] green;
    reg [3:0] blue;
    wire pixel, hsync, vsync;

    wire convstb, csb, rdb;

    //450+5+1+1(read_time)+1+80 = 538 clks

    assign convstb = (ADCctr < 10'd3) ? 0 : 1;
    assign csb = ((ADCctr > 10'd454) && (ADCctr < 10'd457)) ? 0 : 1; // csb -> rdb
    assign rdb = ((ADCctr > 10'd454) && (ADCctr < 10'd457)) ? 0 : 1; // rdb -> csb

    always @(posedge clk or negedge rst) begin
        if (~rst) begin
            ADCctr <= 10'd0;
            i <= 10'd0;
        end else begin
            if (ADCctr < 10'd537) begin
                if (ADCctr == 10'd456) begin
                    if (i <= 10'd639) begin
                        buffer[i] <= db[7:0];
                        i <= i + 10'd1;
                    end else begin
                        i <= 10'd0;
                        buffer[i] <= db[7:0];
                    end
                end
                ADCctr <= ADCctr + 10'd1;
            end else begin
                ADCctr <= 10'd0;
            end
        end
    end
```

```
    always @(posedge clk) begin
        if (pctr < 2'd3) begin
            pctr <= pctr + 2'd1;
        end else begin
            pctr <= 2'd0;
        end
    end

    assign pixel = (pctr < 2'd2) ? 0 : 1;
    assign hsync = (hctr < 10'd96) ? 0 : 1;
    assign vsync = (vctr < 10'd2) ? 0 : 1;

    always @(posedge pixel or negedge rst) begin
        if (~rst) begin
            hctr <= 10'd0;
            vctr <= 10'd0;
        end else begin
            if (hctr < 10'd799) begin
                hctr <= hctr + 10'd1;
            end else begin
                hctr <= 10'd0;
                if (vctr < 10'd520) begin
                    vctr <= vctr + 10'd1;
                end else begin
                    vctr <= 10'd0;
                end
            end
        end
    end

    always @(posedge pixel) begin
        if (vctr >= 10'd31 && vctr <= 10'd510 && hctr >= 10'd144 && hctr <= 10'd783) begin
            if (vctr == (400 - buffer[hctr-144]) ) begin
                red <= 4'h0;
                green <= 4'hf;
                blue <= 4'h0;
            end else begin
                red <= 4'h0;
                green <= 4'h0;
                blue <= 4'h0;
            end
        end else begin
            red <= 4'h0;
            green <= 4'h0;
            blue <= 4'h0;
        end
    end
endmodule
```

Oscilloscope on VGA Display	
	

FPGA Setup w/ A/D Converter


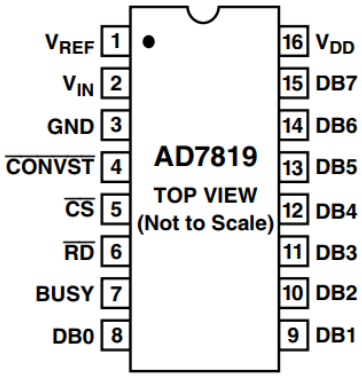

2085.0000 Hz
000.035


+0.000 mV DC
000.035

Discussion

1. We learned that we could set our total clock cycle to either lower or larger than what the AD7819 documentation specifies for the timing characteristics.
2. If there is a max clock cycle for the timing of a specific condition, then we can only go as large as the max, and it would not be ideal to go faster as the AD7819 would need time to read the data bits coming in.
3. If there is a min clock cycle for the timing of a specific condition, then we can increase the conversion time as it would give AD7819 more time to process the data bits.
4. We found that most of the issues lie in the hardware aspect. The wiring of the breadboard to PMOD ports is critical. We had to be mindful of which wire goes where, so we suggested assigning color codes to specific wires to avoid mixing up and crossing the wrong inputs into the wrong outputs.

PIN CONFIGURATION
DIP/SOIC



Conclusion

Our final lab assignment in CMPE 323 tasked us with creating a VGA Oscilloscope using A/D Converters, making it the most challenging one yet. To complete it, we had to construct a budget oscilloscope using the tools provided. The difficulty lay in applying the skills and knowledge gained throughout the course, with documentation serving as our guide for the experiments. This teaching method is vital, reflecting the real-world scenario where independent problem-solving is essential. Instead of a predefined "right" solution, success hinges on acquired skills and persistent effort. Cheating robs us of valuable learning experiences as proper understanding emerges from tackling challenges head-on. As the saying goes, "Nothing beats actual experience." This lab not only had us build an oscilloscope that we can actually make for ourselves at home but also offered a fresh perspective on problem-solving in the field of computer engineering, representing just the initial step in our journey.