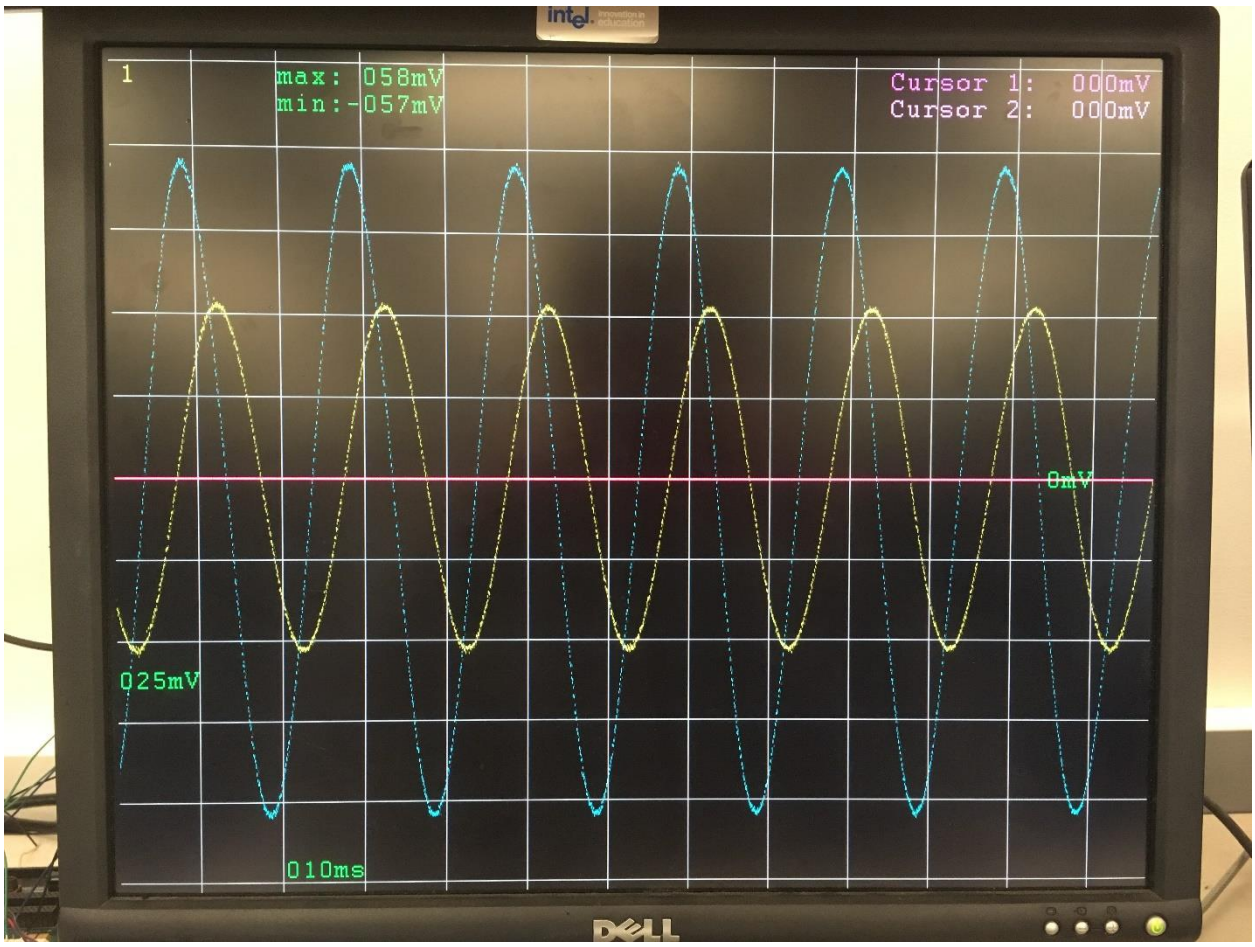
Digital Storage Oscilloscope

Daniel Richman and Jorge Troncoso

6.111 Final Project

16 December 2016



Project Abstract

Our project is a full-featured digital storage oscilloscope. The scope acquires data at 1 MSPS on up to two input channels using the analog to digital converter (ADC) on the Nexys 4 FPGA board. Signals are cleanly displayed on the monitor thanks to a user-controlled trigger level. The oscilloscope supports horizontal and vertical zoom, X-Y display mode, and signal analytics, including cursors, measurements, and a Fourier Transform to show a frequency domain representation of the input signal.

Acknowledgements

We would like to thank 6.111 lecturer Gim Hom for his excellent instruction. The labs and the problem sets helped us develop invaluable skills to complete this project. We would also like to thank Joe Steinmeyer and Alex Sloboda – our project advisor – for their guidance and helping us troubleshoot our project. Finally, we'd like to thank Mitchell Gu for providing an example project that computes the Fast Fourier Transform of a signal using the Nexys 4 FPGA board. We used his project extensively to display the FFT of the input signal.

Contents

1 Introduction

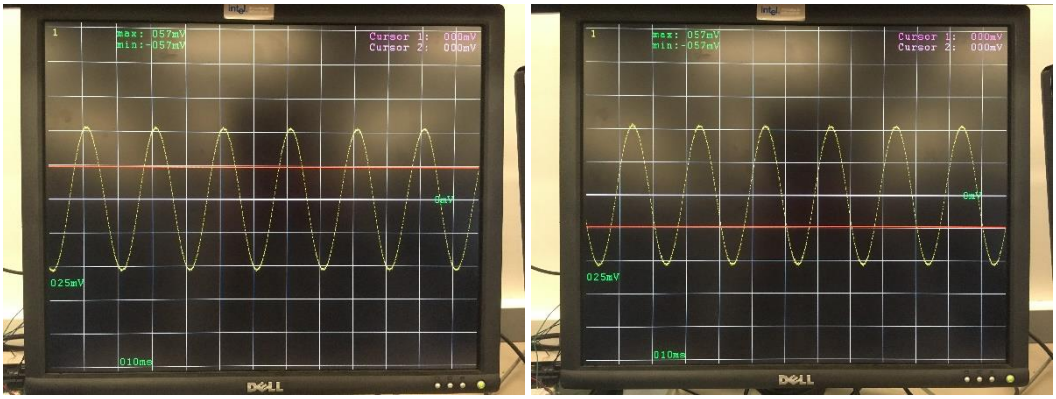
Many FPGA projects could also work on a microcontroller, but analog data acquisition and processing is a task well-suited to an FPGA. An essential tool of analog electronics is the oscilloscope, which acquires arbitrary external signals and displays them cleanly to the user in real time. Our final project implements a digital storage oscilloscope with customizable display and signal analysis features using the Artix 7 FPGA on a Nexys 4 DDR board. The ADC built into the board can acquire 12-bit samples on two input channels at 1 MSPS. All other components of the oscilloscope operate at a clock rate of 104 MHz.

The user controls the oscilloscope using the switches and buttons on the Nexys 4 FPGA board. The oscilloscope uses block RAM, ROM and flip-flops for storage. ROM is used to display the font for the *Text* display module, and block RAM is used to store samples of the analog input signals and implement the FFT system. Each of these systems is discussed further in their respective sections.

2 System Overview

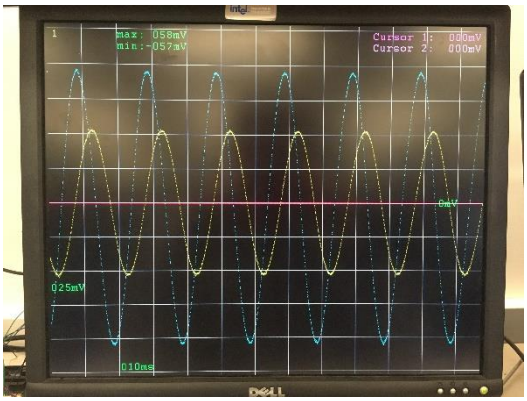
2.1 Features

Adjustable triggering system

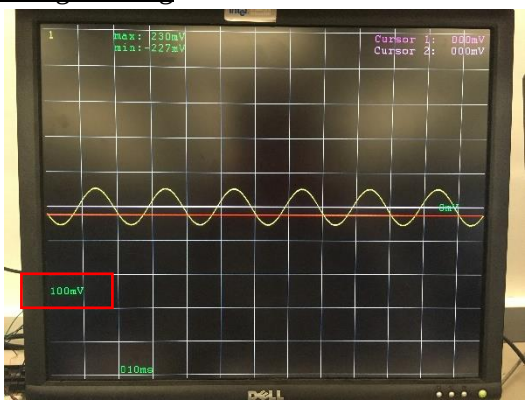


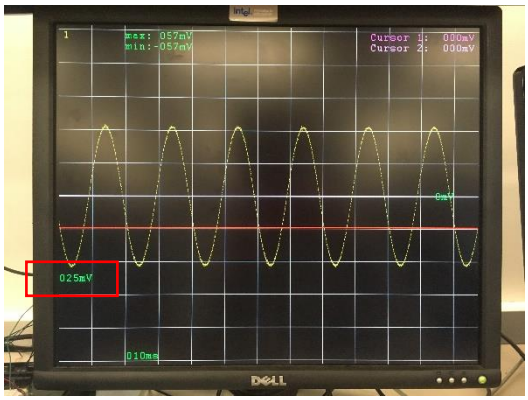
The trigger level is represented by the red line

2 Channels



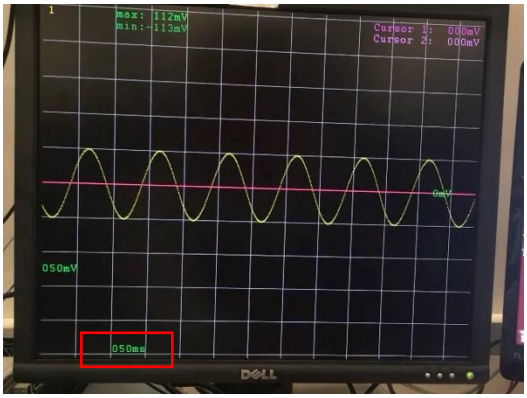
Voltage scaling





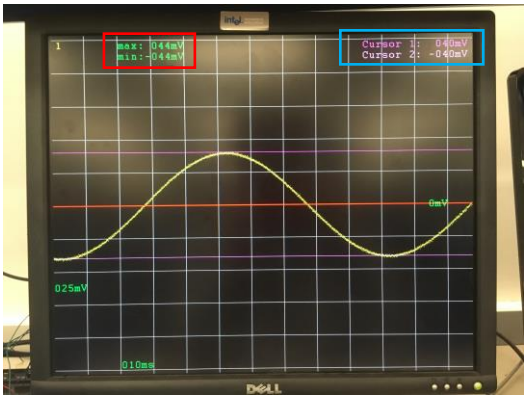
The volts per division is indicated in the red box

Time scaling



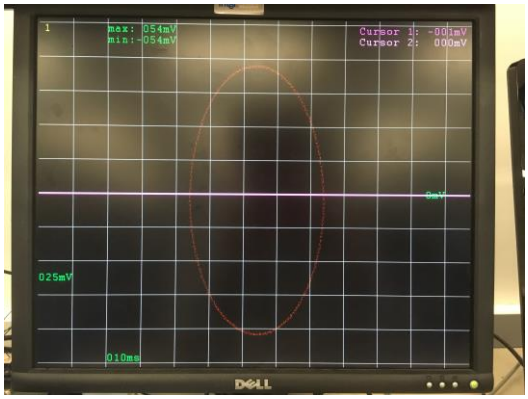
The time per division is indicated in the red box

Minima, Maxima, and Cursors

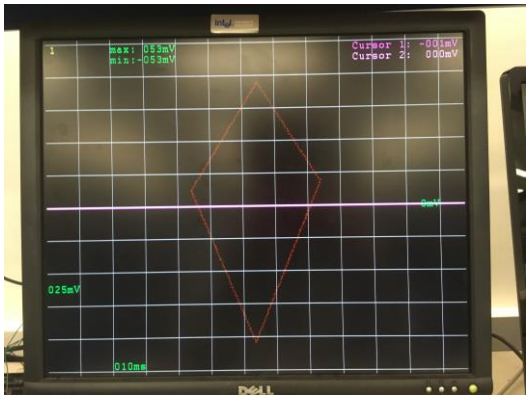


The maxima and minima of the signal are displayed in the red box. The cursor levels are represented by the purple lines. The values of the cursor levels are displayed in the blue box.

X-Y Display Mode

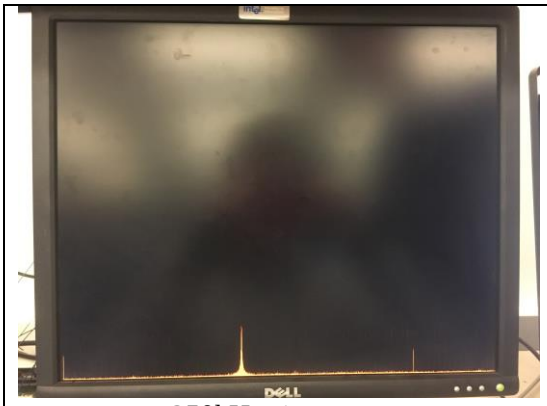


channel 1: 5kHz sine wave, 100mV peak-to-peak
channel 2: 5kHz sine wave, 200mV peak-to-peak



channel 1: 5kHz ramp, 100mV peak-to-peak
channel 2: 5kHz ramp, 200mV peak-to-peak

Fourier Transform

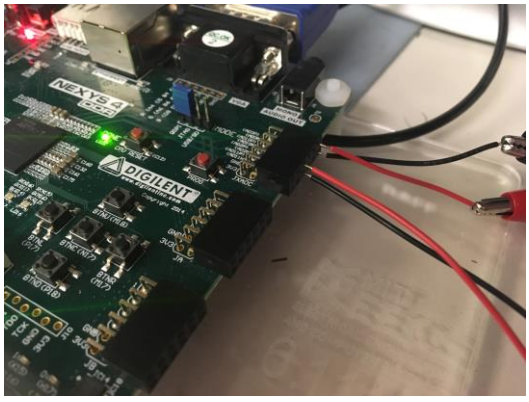


250kHz sine wave

500kHz sine wave

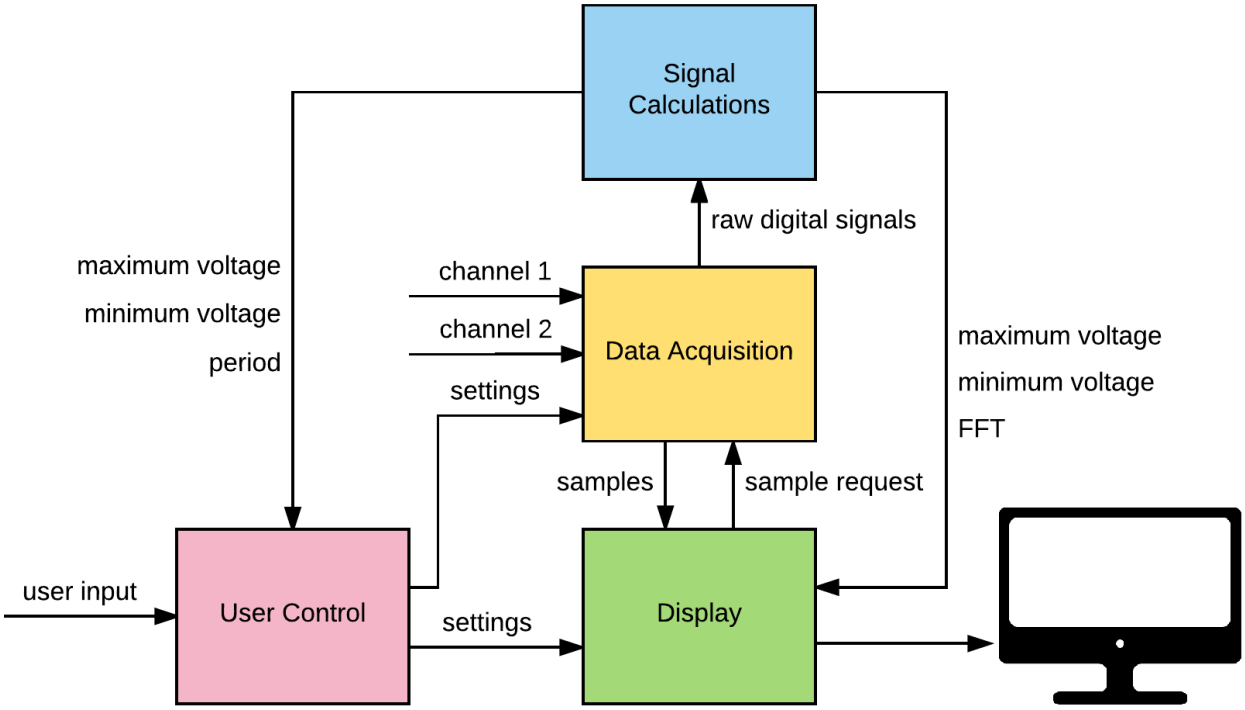
2.2 Physical Setup

Our oscilloscope uses the ADC on the Nexys 4 FPGA board to convert analog signals from two channels into digital signals. The positive and negative terminal of each channel is connected to an auxiliary analog input on the Nexys 4 board.



Analog signals going into auxiliary analog inputs of the Nexys 4 FPGA board.

2.3 Block Diagram



Our oscilloscope consists of four major blocks. The *Data Acquisition* block converts analog signals from two channels into digital signals and stores them in memory. The *Signal Calculations* block computes the maxima, minima, and Fast Fourier Transform of each signal. The *User Control* block adjusts the Oscilloscope's settings based on user input, and the *Display* block draws a waveform, an XY plot, or the fast Fourier Transform of a signal on a computer monitor.

A note about terminology: A block is a subsystem of the oscilloscope. Blocks may be composed of one or more Verilog modules

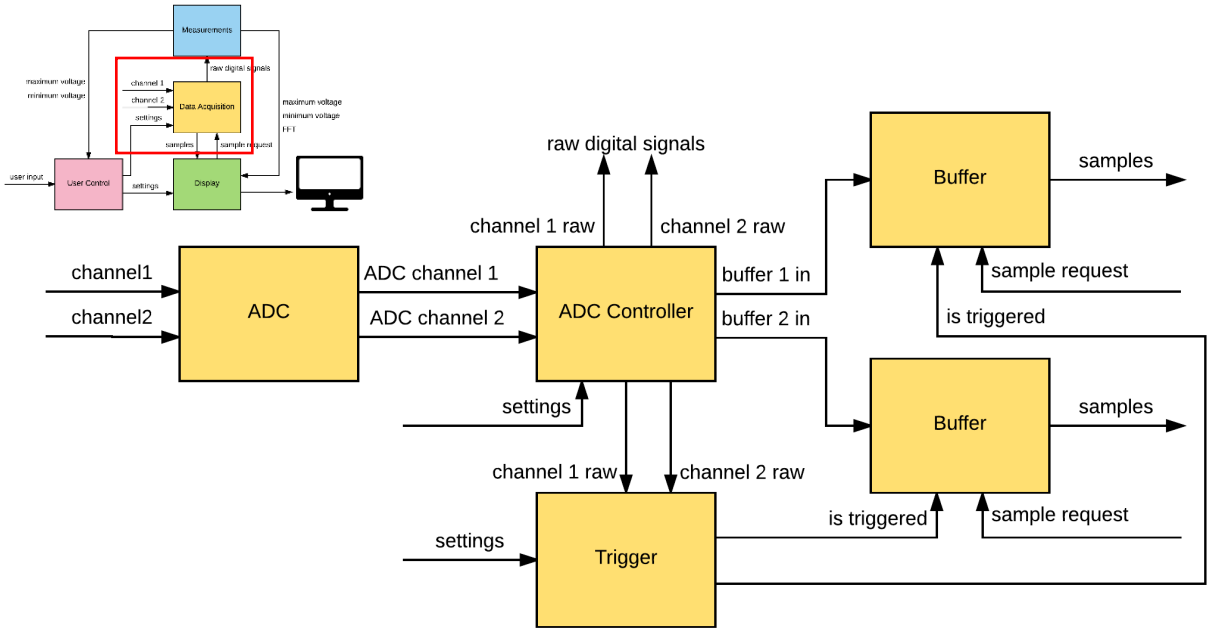
3 Block Design and Implementation

3.1 Data Acquisition

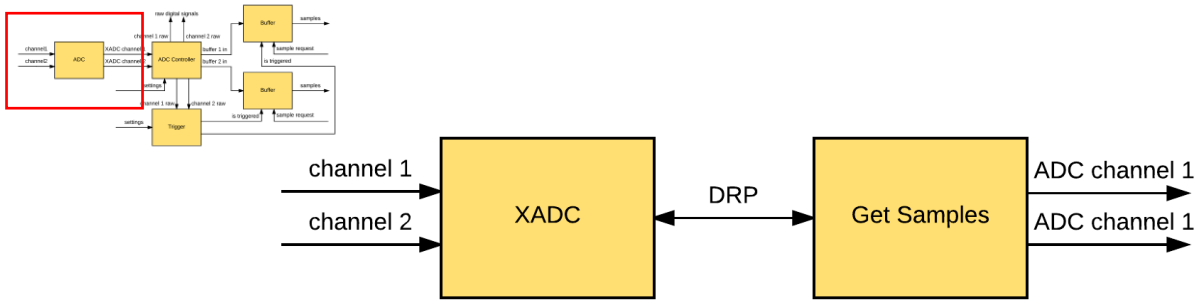
The *Data Acquisition* block receives analog inputs from two channels, settings sent from the *User Control* block, and sampleRequests sent from *Display* block. The *Data Acquisition* block converts the analog input signals into digital signals and stores them in two buffers, one for each channel. Other blocks can access the samples stored in these buffers by sending sampleRequests to the *Data Acquisition* block.

The *Data Acquisition* block also outputs the rawDigitalSignals so they can be used by other blocks.

3.1.1 Block Diagram



3.1.2 ADC



The ADC block instantiates the *XADC* core, reads the samples stored in the XADC's status registers, and outputs the samples so they can be used by other blocks.

The *XADC* core is a building block available for all Xilinx 7 Series FPGAs. It includes a dual 12-bit, 1 Mega sample per second (MSPS) ADC and an on-chip analog sensor. The samples obtained by the ADC are stored in the *XADC's* status registers and they can be accessed through the FPGA's dynamic reconfiguration port (DRP) (read pages 13-14 of the *XADC User Guide* for more information on status registers and the DRP)

To accommodate analog signals that are positive or negative with respect to GND, we set the *XADC* analog inputs to bipolar mode. When bipolar operation is enabled, the analog input can have a maximum range of ±0.5V, as opposed to unipolar mode, where the analog input must between 0V and 1V. Bipolar operation is selected by writing to configuration register 0 of the *XADC* core (read page 31 of the *XADC User Guide* for more information on how to write to the XADC's configuration registers).

Below is an example of how to instantiate the *XADC* core. The INIT_4D parameter writes to configuration register 0 and selects unipolar or bipolar mode for each of the 16 analog inputs. A logic 1 places the analog input in bipolar mode and a logic 0 places the analog input in unipolar mode. For example if you want to set analog inputs 2 and 3 to bipolar mode and all other analog inputs to unipolar mode, you would set the INIT_4D parameter to 16'b0000_0000_0000_0110, which is equivalent to 12'h0006 in hexadecimal.

```
XADC #(
    .INIT_40(16'h8000), // config reg 0
    .INIT_41(16'h410F), // config reg 1
    .INIT_42(16'h0400), // config reg 2
    .INIT_48(16'h0000), // Sequencer channel selection
    .INIT_49(16'h0008), // Sequencer channel selection
    .INIT_4A(16'h0000), // Sequencer Average selection
    .INIT_4B(16'h0000), // Sequencer Average selection
    .INIT_4C(16'h0000), // Sequencer Bipolar selection
    .INIT_4D(16'hFFFF), // Sequencer Bipolar selection, all analog inputs set to bipolar mode
    .INIT_4E(16'h0000), // Sequencer Acq time selection
    .INIT_4F(16'h0000), // Sequencer Acq time selection
    .INIT_50(16'hB5ED), // Temp alarm trigger
    .INIT_51(16'h57E4), // Vccint upper alarm limit
    .INIT_52(16'hA147), // Vccaux upper alarm limit
    .INIT_53(16'hCA33),  // Temp alarm OT upper
    .INIT_54(16'hA93A), // Temp alarm reset
    .INIT_55(16'h52C6), // Vccint lower alarm limit
    .INIT_56(16'h9555), // Vccaux lower alarm limit
    .INIT_57(16'hAE4E),  // Temp alarm OT reset
    .INIT_58(16'h5999), // VCCBRAM upper alarm limit
    .INIT_5C(16'h5111),  //  VCCBRAM lower alarm limit
    .SIM_DEVICE("7SERIES"),
    .SIM_MONITOR_FILE("design.txt")
)
```

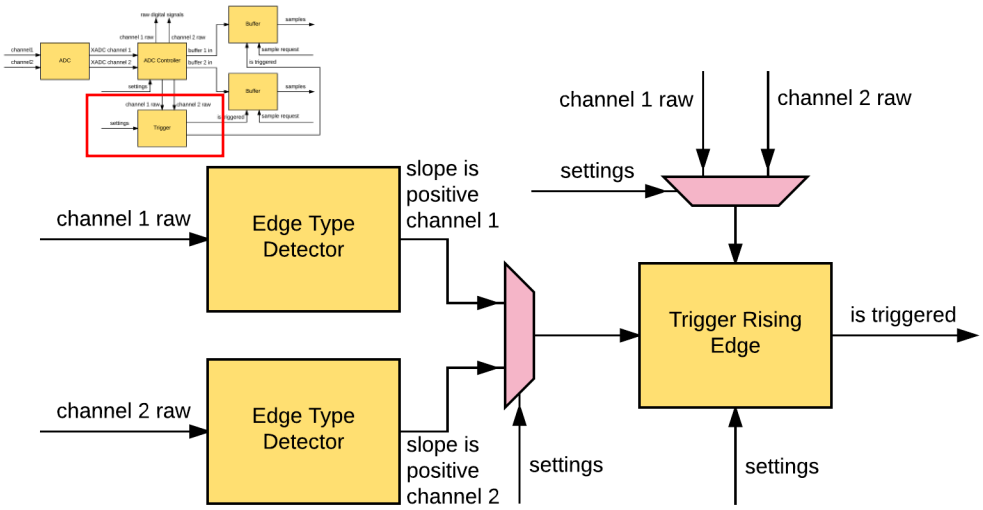
3.1.3 ADC Controller

The *ADC Controller* module accepts samples from the *ADC* block. On each clock cycle, each input sample is copied directly out on the raw outputs. The non-raw outputs are subsampled and sent to the *Buffer* modules; the subsampling rate is determined by the settings sent from the *User Control* block. Downsampling the data at this stage allows us to scale the time axis while avoiding the complexity of doing it once the data is already in the buffer.

3.1.4 Trigger

The trigger system processes the incoming data stream from either channel and asserts the isTriggered signal for one clock cycle when the input data is on a rising edge and crosses the user-defined trigger threshold (the trigger threshold is one of the settings outputted by the *User Control* block). Trigger glitching primarily occurs because of input noise, which can cause a falling edge to be misdetected as a rising edge. To mitigate trigger glitching, the trigger system includes an edge type detector that smooths the input signal, effectively low-pass filtering it and reducing fuzz. The trigger system consists of two parts: the *Edge Type Detector* module, which senses whether the signal is on a rising edge, and the *Trigger Rising Edge* module, which checks the signal level and performs the triggering. Note that the trigger is not used in X-Y display mode or in FFT display mode.

3.1.4.1 Block Diagram



3.1.4.2 Edge Type Detector

This module receives the raw data input stream from the *ADC Controller* and computes a moving average of the signal by convolving the past five samples with the kernel (1, 2, 3, 2, 1). This is effectively a low-pass filter, so it reduces high-frequency noise. Then, a slope is computed from the signal by subtracting the smoothed samples four timesteps apart. If the slope is positive, the slopeIsPositive signal goes high, which activates the *Trigger Rising Edge* module.

3.1.4.3 Trigger Rising Edge

This module asserts the isTriggered output when three conditions are met: (a) the input signal has crossed the threshold value, (b) the last trigger occurred longer ago than the TRIGGER_HOLDOFF parameter (this prevents noise from causing re-triggering on the same edge), and (c) the module is not disabled by the *Edge Type Detector* module.

3.1.5 Buffer

The Buffer module stores ADC samples for further processing. Its primary input is the *ADC Controller* module's subsampled output, although its input could also come from a math module or other preprocessing unit. The Buffer instantiates two BRAMs, an active BRAM and a locked BRAM, that it uses as ring buffers. Incoming samples are stored in the active BRAM. When the isTriggered signal from the Trigger system goes high, the address of the most recent sample in the active BRAM is stored in the triggerSample register.

We instantiate one *Buffer* module for each data channel. A helper module, the *Buffer Selector* module, interfaces between the display system and the *Buffer* modules and produces an activeBRAMSelect signal that controls which BRAM is active and which is locked. At the start of drawing each display frame, the activeBRAMSelect signal toggles. This ensures that the data in the locked BRAM is not altered while the frame is being drawn.

The *Buffer* module only supports read access to the locked BRAM. Upon sampleRequests from the *Display* block, the *Buffer* module outputs the requested sample three clock cycles after the sampleRequest is received.

3.2 Signal Calculations

The *Signal Calculations* block consists of two module, the *Measurements* module, and the *Fast Fourier Transform* module.

3.2.1 Measurements

The *Measurements* module accepts the raw data streams from the *ADC Controller* and computes several rolling pieces of information about each channel. Primarily, it computes the minima and maxima of the signals, which are later displayed on the screen and used to compute optimal Autoset settings. There are several approaches to storing a rolling minimum and maximum. To minimize memory usage, we opted to store only the smallest (and largest) voltage value observed and the number of samples ago that each value was seen. If a smaller (larger) sample arrives, the minima and maxima are immediately updated. If the recorded voltage extremes become too stale, they are discarded and updated to the current value of the sample.

One drawback to this approach is that the displayed minima and maxima may not agree with the signal shown on screen. However, one standard approach (measuring only the portion of the signal shown on screen) might be misleading in our case because of the undersampling performed for display.

3.2.2 Fast Fourier Transform

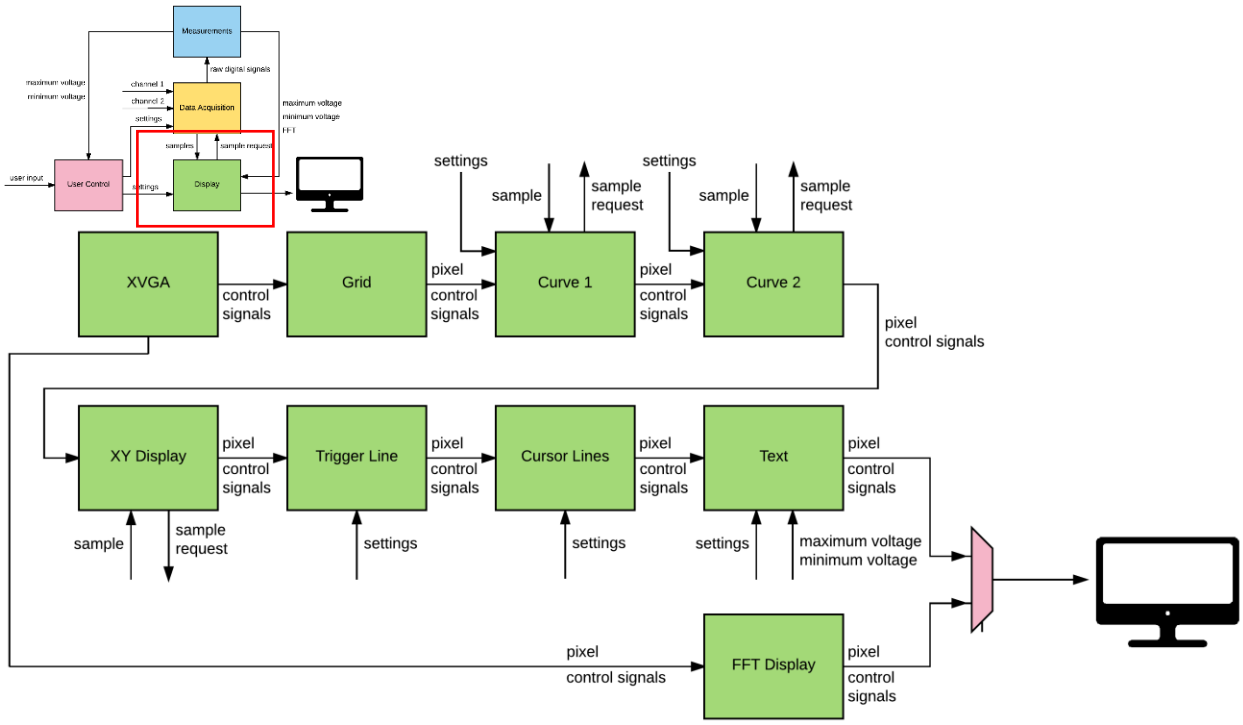
The *Fast Fourier Transform* module is responsible for computing a 2048-point Fast Fourier Transform of the Channel 1 input data. The module instantiates an IP block design lightly modified from Mitchell Gu's Magnitude FFT design, which consists of the FFT IP core followed by a series of math IP cores that square the real and imaginary parts of the output, sum them, then take the square root to compute the magnitude. Interestingly, the FFT system takes up approximately a quarter of the circuit area of the FPGA and triples the place-and-route implementation time.

3.3 Display

Our display architecture consists of a sprite pipeline which is bypassed only if the oscilloscope is in FFT display mode. All XVGA control signals (*displayX*, *displayY*, *hsync*, *vsync*, *blank*) pass in series through all sprites. (We refer to the signals *hcount* and *vcount* as *displayX* and *displayY*.) Each sprite is responsible for appropriately delaying the control signals so that they reach the monitor at the same time as the appropriate pixel.

Each sprite also outputs a *pixel* value, which it either passes through from the previous sprite or overwrites with its own data.

3.3.1 Block Diagram



3.3.2 XVGA

The XVGA module generates the XVGA control signals (*displayX*, *displayY*, *hsync*, *vsync*, *blank*) required to display images on a screen. We decided to use a screen resolution of 1280x1024 and a clock frequency of 104 MHz. To update the screen at a rate of 60 frames per second, we would've needed a

108MHz clock, however, the ADC can't sample at 1MSPS at this clock frequency, so we reduced it to 104MHz. Due to the reduced clock rate, our screen updates at a rate of 58 frames per second.

3.3.3 Grid

The Grid module draws grid lines over a black background on the screen.

3.3.4 Curve 1/Curve 2

Each Curve module requests samples from the *Data Acquisition* block and displays a waveform representing the string of samples on the screen. The *Curve 1* module displays the samples from channel 1, and the *Curve 2* module displays the samples from channel 2.

To display a waveform, each Curve module reads 1,279 samples that came before the latest trigger sample. The 1,279th sample is displayed on the left side of the screen (pixel 1) and the trigger sample is displayed on the right side of the screen (pixel 1,280)

Each curve module also receives setting from the *User Control* block which control the scaling of the voltage and time axes.

3.3.5 XY Display

The *XY Display* module plots Channel 2 samples as a function of Channel 1 samples. For instance, when the two channels are tied together, a 1:1 diagonal line is displayed; independent noise on both channels results in a fuzzy dot at the center of the screen; and two sine waves can produce an ellipse or, if they differ in frequency, a more complex pattern (a Lissajous curve). The X-Y display mode can help to visualize very fast frequency signals.

To minimize memory consumption, the *XY Display* module uses two pixel line buffers. While one row of samples is being displayed from Buffer A, Buffer B is being populated with X-Y samples; the buffers switch roles at the end of a row. Each XVGA line has 1,280 displayed pixels and 1,688 total pixels, including porches, so there are 1,688 clock cycles available to populate Buffer B with samples. Therefore, the most recent 1,688 pairs (x, y) of data samples from the two channels are read, and a pixel is drawn at location x in the next row if the vertical position of the next row (displayY + 1) equals y.

3.3.6 Trigger Line

The *Trigger Line* module receives a trigger level (the trigger level is one of the setting outputted by the *User Control* block) and draws a red, horizontal line representing the trigger level on the screen.

3.3.7 Cursor Lines

The oscilloscope has two cursors that can be used to measure the amplitude of a signal. These cursors can be moved up and down to take measurements from different signals.

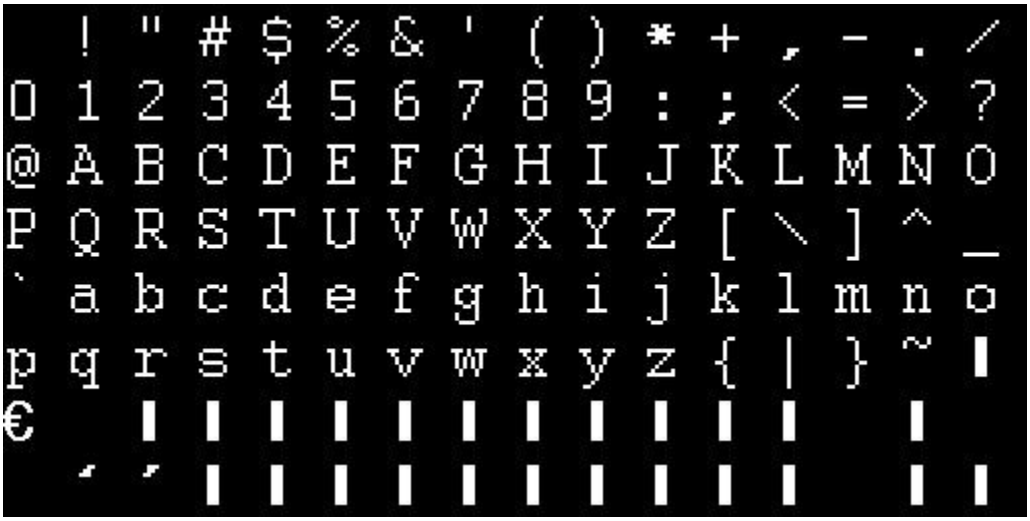
The *Cursor Lines* block receives settings from the *User Control* block indicating the voltage level of each cursor and draws two purple, horizontal lines representing cursor 1 and cursor 2.

3.3.8 Text

The *Text* module displays the channel that is currently selected, the volts per division, the time per division, the maxima of the selected channel, the minima of the selected channel, and the voltage levels of cursor 1 and cursor 2.

The volts per division, time per division, and voltage levels of cursor 1 and cursor 2 are computed from the settings sent from the *User Control* block. Information about the channel that is currently selected also comes with the settings sent from the *User Control* block. The maxima and minima of the selected channel are sent from the *Measurements* module.

To display text, first we created a monochrome bitmap of all the characters we wanted to display (see image below). Then we converted the bitmap into a COE file using the 6.111 provided MATLAB script and loaded it into BROM as an init file. The *Text* module, requests pixel data from the BROM to draw characters on the screen.



3.3.10 FFT Display

The *FFT Display* module displays the Fast Fourier Transform representation of the signal. This display mode is activated by flipping a switch on the Nexys 4 FPGA board.

The *FFT Display* module is configured for 2,048 points, meaning that it reads 2,048 input samples per frame and produces 2,048 output points, of which 1,024 are in the right half-plane of the frequency space. We show only these right 1,024 points, which are stored in a BRAM by the *Fast Fourier Transform* module. Unlike the regular display system, the *FFT Display* system does not have two buffers, so it is possible for the values in the BRAM to change over the course of a single display frame. This may result in some display artifacts. However, in many cases this is not a problem because the frequency representation of most inputs does not typically change too rapidly. (If our scope were configured for single-shot triggering, a more sophisticated FFT display system might be useful.)

3.4 User Control

The *User Control* block processes commands from the user and adjusts the settings of the oscilloscope. Users can flip the switches and the buttons on the Nexys 4 FPGA board to change the trigger level, the scaling of the voltage axis, the scaling of the time axis, the cursor levels, the selected channel,

and the display mode (normal, X-Y, or FFT). After processing commands from the user, the *User Control* block sends the settings to the *Data Acquisition* and the *Display* blocks.

Users can also choose to adjust the settings automatically using our autoset feature. Autoset, computes an optimal trigger level, scaling of the voltage axis, and scaling of the time axis based on measurements sent from the *Measurements* block. Autoset uses the maxima and minima of the input signal to obtain a trigger level that stabilizes the signal and to obtain a voltage scaling setting that ensures the entire signal is in view. Thanks to the Trigger system, Autoset knows the interval between triggers and selects a horizontal scaling settings that shows three full period of the signal on the screen.

4 Conclusion

We are very pleased with the final result. We were able to implement several features in our oscilloscope and we learned a lot about digital design through the process.

We believe there were a number of factors that allowed us to succeed in this project. After finishing the last lab, we set ourselves a team goal that consisted in building a minimum viable product (MVP) in two weeks. The MVP didn't have any of the special features described in this report, it just displayed a waveform on the screen. This ambitious goal pushed us to refine our design quickly and get to a working system. Once we were able to display a waveform on the screen, we implemented additional features on top of the original MVP project. Therefore, in the remaining four weeks of the project, we were always building on top of a working system. We also uploaded our project to a GitHub repository which allowed us to work on different features in parallel. Finally, we set ourselves clear goals every week and wrote them in a Google Doc. This document served as a Go-To list to find out what we had to achieve each week.

5 Appendix

GitHub Repository: https://github.com/Daniel-And-Jorge/6.111-Final-Project/tree/master/Oscilloscope_v1

Project Overview: https://www.youtube.com/watch?v=rRyLpQIlsDQ