ELEC 490/498 Project Final Report

Group 18
TeachEE: Accessible Electronics Instrumentation

John Giorshev (20103586, john.giorshev@queensu.ca)
Eric Yang (20120750, e.yang@queensu.ca)
Ethan Peterson (20105011, 17emp5@queensu.ca)
Timothy Morland (20096286, 17tm22@queensu.ca)

Submitted April 10, 2023

To:
Instructor Dr. Michael Korenberg (korenber@queensu.ca)
Instructor Dr. Alireza Bakhshai (alireza.bakhshai@queensu.ca)
Instructor Dr. Alex Tait (alex.tait@queensu.ca)
Instructor and Supervisor Dr. Sean Whitehall (sw109@queensu.ca)
Supervisor Dr. Thomas Dean (tom.dean@queensu.ca)

Executive Summary

The Covid-19 pandemic abruptly forced educational institutions towards online learning. As a result, many of these institutions were unequipped to deal with this new learning model. Much of the onsite equipment, such as oscilloscopes and signal generators, is ill-suited for online learning due to its high cost and poor portability. Additionally, the take-home equipment provided to students during the pandemic was typically of lower quality in comparison. In particular, the oscilloscopes provided by the Electronics II course taught at Queen's University were expensive and of poor quality. They had tiny 3-inch displays and a small maximum bandwidth of 100 kHz. Moreover, they were single channel, limiting the student's ability to see how signals in a circuit are related to one another.

TeachEE is a general-purpose electronics measurement instrument designed for remote engineering labs, functioning as both an oscilloscope and current monitor. It bundles together the functionality most required in Electrical Engineering labs and packages it with lightweight and portable software.

TeachEE's hardware consists of a four-layer custom designed Printed Circuit Board (PCB) powered by a USB connection to the computer. The PCB has a two-channel 1 MSPS ADC for reading low-speed signals and current from a hall-effect sensor. The PCB also has a two-channel 40 MSPS Analog to Digital Converted (ADC) for observing high speed signals. Both ADCs are connected to a Field Programmable Gate Array (FPGA), which is in turn connected to USB 2.0 PHY. The FPGA collects sample data from both ADCs and frames it into packets, which are sent over the USB PHY and transmitted to the computer to be decoded and displayed. This allows TeachEE to leverage the student's machine to create a streamlined user experience.

TeachEE's software is a full stack application written in Rust. Its workload is split between three modules, each of which runs in its own thread. The first module is the USB Manager, responsible for reading and decoding packets from the hardware driver into voltage and current samples. The second is the App, responsible for updating and displaying the user interface. The third module, the Controller, facilitates communication between the two, and performs signal triggering and frequency spectrum analysis. These three threads pass samples between each other using two sets of two buffers: the USB Buffers and App Buffers. By cycling between buffers within a set, much of the data processing can be parallelized, greatly improving performance.

Contents

List of Figures

List of Tables

1 Motivation & Background

During the Covid-19 pandemic, universities abruptly switched to remote learning as campuses were shut down. This sudden change exposed the lack of support for digital learning models [1]. Although the pandemic has since passed, forward-thinking institutions are starting to invest in online programs, for example, by leveraging AI chatbots and teaching assistants. Additionally, online course providers such as Udemy have recently risen in popularity as a cheap alternative to traditional postsecondary education. With this trend of remote higher education, engineering students will need their own lab equipment to complete their learning requirements.

TeachEE is a general-purpose electronics measurement instrument designed for remotely delivered engineering labs. It acts as both a USB oscilloscope and current monitor. Currently, there are few instrumentation options for students completing their labs remotely. The oscilloscopes used for onsite learning are bulky and difficult to use. TeachEE bundles together the functionality most commonly required for electrical engineering labs and packages it with portable software that can run on lower end computers with any operating system.

As TeachEE is targeted for electrical engineering students, it must be powerful enough for use in a variety of practices, but also simple and intuitive enough for novices. It must be compact, to be carried between classes, and operating-system agnostic, to ensure that students can use it with any machine. Finally, TeachEE must be cost-efficient, so that students can justify purchasing this oscilloscope as opposed to alternatives.

2 Design

The TeachEE design is derived from the system requirements set in the blueprint report. Given in Table 1 and 2 are the most important hardware and software requirements respectively. It should also be noted that the FPGA shares responsibility with the desktop application for fulfilling software requirements.

Table 1: Hardware Requirements

	Specification	Target Value	Tolerance
1	Voltage Input Bandwidth	100 kHz	±1 kHz
2	Current Input Bandwidth	100 kHz	±1 kHz
3	Number of Current Input Channels	1	0
4	Number of Voltage Input Channels	1	+2
5	Voltage Sample Rate	1 MSPS	0 MSPS
6	Current Sample Rate	1 MSPS	0 MSPS

Table 2: Software Requirements

1	**Functional Requirements**
1.1	The software shall be able to modify the horizontal and vertical scales of the plot.
1.2	The software shall be able to modify the trigger voltage.
1.3	The application shall be deployable to Windows, macOS, and Linux.
2	**Interface Requirements**
2.1	The software shall be able to capture voltage samples and export them to a CSV file.
2.2	The software shall receive samples via the FTDI 232 in synchronous mode.
3	**Performance Requirements**
3.1	The software shall be able to render waveforms at a rate of 30 Hz on screen.

Based on the system requirements, the project is divided into three subsystems; hardware, FPGA, and software. The following subsections correspond to each subsystem, each describing the subsystem's top level design and key design decisions made to comply with system requirements.

2.1 Hardware

The primary design driving requirements for the Printed Circuit Board (PCB) are the bandwidth and sample rate of the data acquisition channels. Bandwidth and sample rate constrain the selection of the input filters and Analog to Digital Converters (ADCs) on the PCB. After selecting these parts, they can be easily duplicated to add additional channels. A full requirement compliance check for the hardware and other subsystems is provided in Section 4.

Given the requirement for a single voltage and current channel with room for additional voltage channels, the team settled on a four-channel design. One ADC channel would be used for current and another for

voltage. The remaining two channels would be made available to be implemented as a stretch goal. The two spare channels would also have a higher sample rate to improve the utility of the instrument.

The data acquisition channels are connected to an FPGA rather than a microprocessor. This decision was made to prevent real time data processing limitations and ensure precise timing in the sampling of the signal waveforms. Moreover, an FPGA or ASIC is commonly used in existing commercial oscilloscopes to fulfill these real time requirements. A compromise that was made in the design was the use of an FPGA module, rather than installing the chip to the PCB directly. While such a module could not be used in TeachEE in a mass-production setting, it greatly shortened the PCB design time, and allows the FPGA to be easily replaced in the event of hardware failure. Additionally, the FPGA combines all the sample streams and provides a single point of contact to the USB interface.

Figure 1 is a complete hardware block diagram used as a guide for the PCB design implementation.

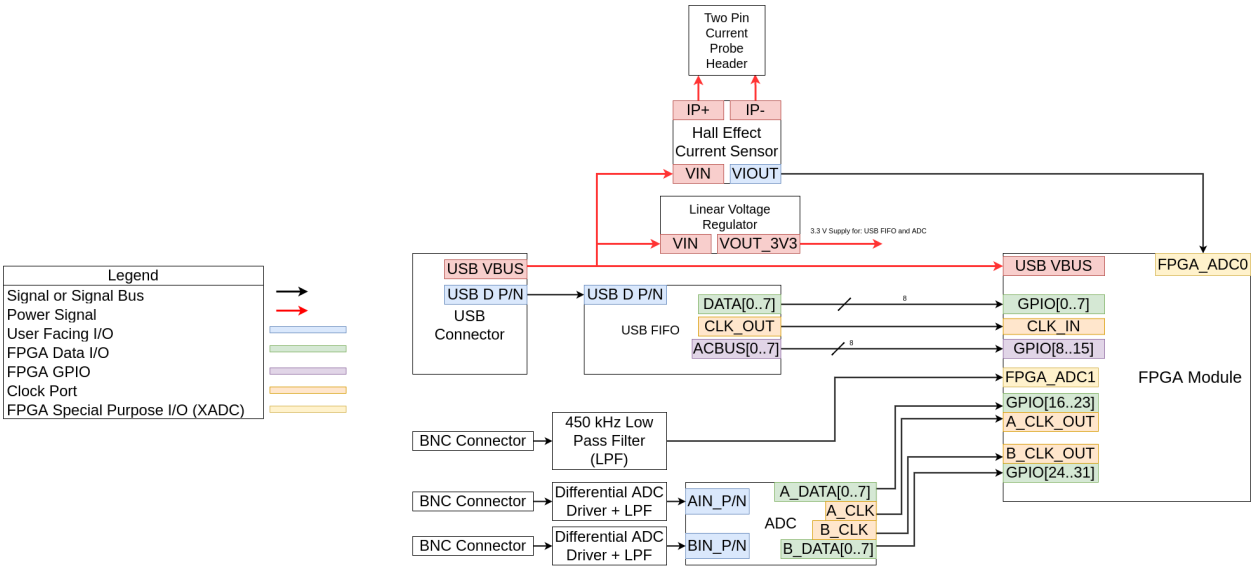


Figure 1: Hardware System Block Diagram

2.2 FPGA

The FPGA RTL is driven by the implicit requirement to send multiple streams of data with low latency. The sample rate and bandwidth requirements are not as concerning in the design of the FPGA subsystem. This is because sample rates can be easily tuned in the FPGA with clock generation PLLs and bandwidth depends on the analog bandwidth of the input circuitry on the PCB.

In order to transmit multiple streams, the FPGA needs to frame multichannel data in a standard packet format. A multiplexer is also used in the case that sending all data at once oversaturates USB 2.0. With these issues in mind, the diagram given in Figure 2 provides a system-level block diagram for the FPGA design.

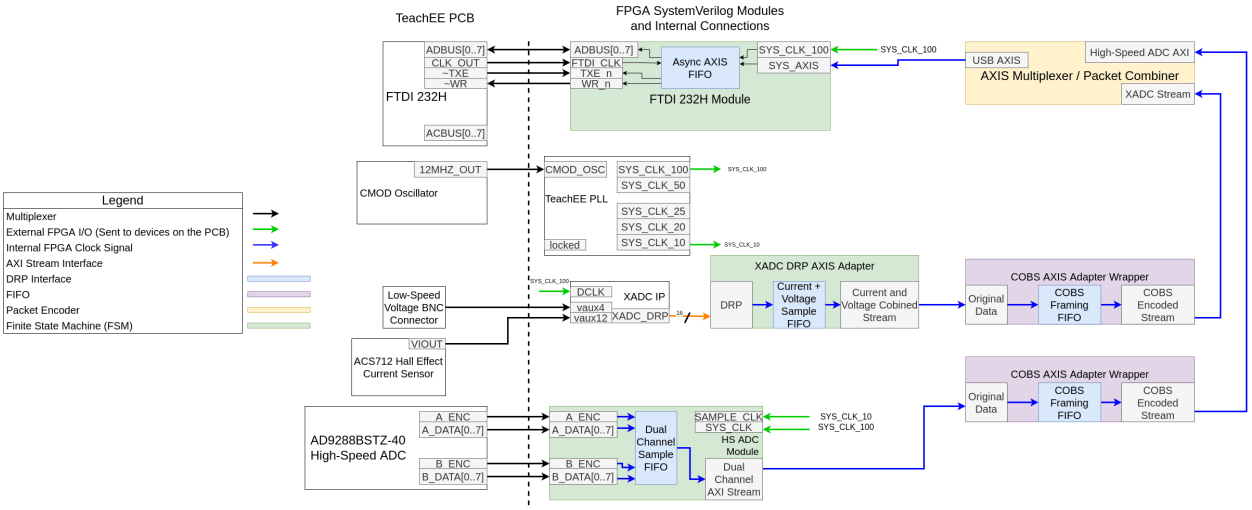


Figure 2: FPGA System Block Diagram

A full breakdown of Figure 2 is given in Section 3 focusing on implementation. Overall, the initial design is nearly identical to the final implementation. The team made use of Consistent Overhead Byte Stuffing (COBS) packet encoding as promised in the blueprint and added a standard Advanced Extensible Interface (AXI) Streaming protocol for the internal FPGA datapath.

2.3 Software

Several iterations were required to complete TeachEE's desktop application. Initially, the prototype was built using the Tauri GUI framework. Tauri was selected since it uses an instance of the Chrome Web Browser to render the user interface, allowing developers to take advantage of the rich library and tooling ecosystem for frontend web development. Despite these benefits, two major factors necessitated a change: 1) displaying a rapidly changing waveform using HTML and JavaScript at the required $60\,\mathrm{Hz}$ frame rate proved extremely difficult; 2) data transfer between Rust and JavaScript resulted in increased code complexity.

`egui` was selected as a replacement for Tauri because of its simple API and large built-in widget library. `egui` is an *immediate mode* GUI, meaning it re-renders all widgets on every frame, regardless of whether or not the application state had changed. This technique sacrifices efficiency for a simpler programming model in which a developer expresses their GUI as a single pure function $f : \text{State} \mapsto \text{UI}$ [2]. An *immediate mode* GUI is a good fit for TeachEE, since its UI changes frequently and independently of user-input, nullifying the aforementioned efficiency problem; even with a traditional *retained mode* GUI, widgets would still re-render whenever a new voltage/current is received.

Later in the software design process, performance issues were encountered related to processing the incoming data stream fast enough to update the GUI at $30\,\mathrm{Hz}$. The slowdown was caused by the original threading model, in which the USB Manager and App threads communicated via a single shared buffer protected by a mutex. When the App thread attempted to render a new frame, it would frequently be blocked by the USB Manager, which was simultaneously writing incoming samples. The frequency and duration of this blocking behaviour was prevalent enough to greatly reduce frame rate. To address this problem, the software was reimplemented using the three thread design described in Section 3.3.

3 Implementation

The implementation is broken up into three subsections corresponding to each subsystem.

3.1 Hardware

The hardware solution for TeachEE is implemented in the form of a four-layer PCB. The PCB is designed using the Electronic Design Automation (EDA) software Altium Designer, the industry standard tool for PCB design. Our design takes advantage of Altium's hierarchical schematics, allowing the schematic to be broken up into multiple pages with interconnects between the different sub-circuits. TeachEE's top level schematic is an interconnect between all the sub-circuits of the PCB, and is shown in Figure 3. All eight pages of the schematic are provided in Appendix A.

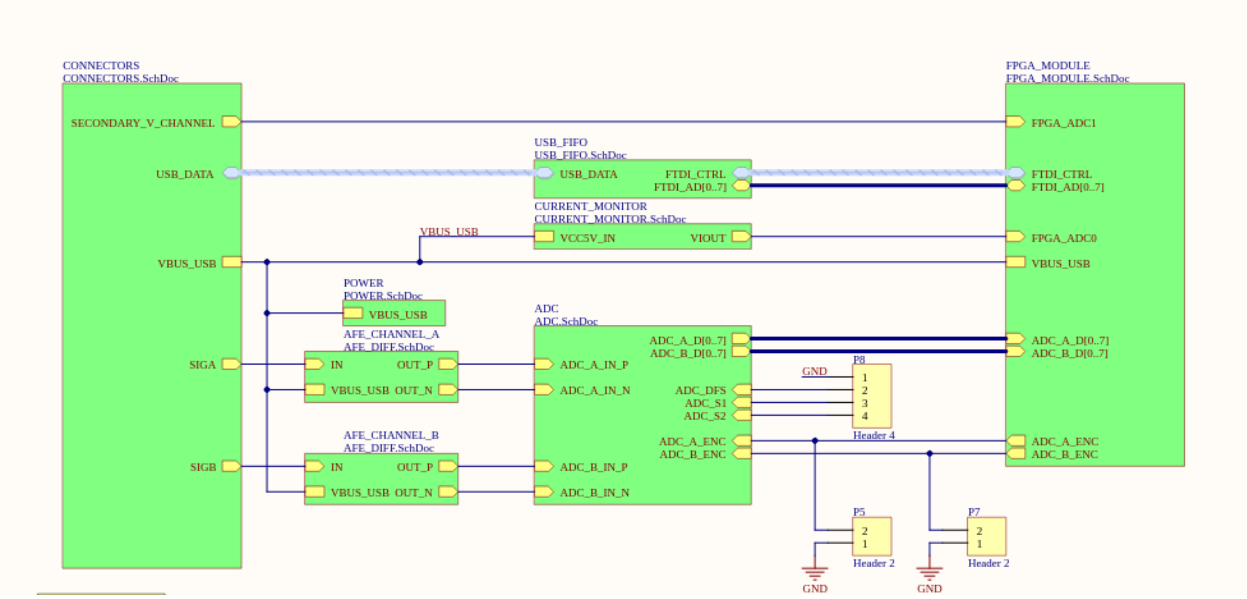


Figure 3: TeachEE PCB Top Level Schematic

The following subsections break down each sub-circuit sheet entry in Figure 3.

3.1.1 Power

The power schematic sheet contains a single LD1117V33C linear voltage regulator. The regulator takes the $5\,\mathrm{V}$ VBUS supply from the USB connection and provides a $3.3\,\mathrm{V}$ power output. A linear voltage regulator is employed over a switching buck converter for the following reasons:

1. The analog to digital converters require clean voltage references to produce accurate readings and function correctly. Buck converters typically have noisier outputs that could harm performance.
2. The linear voltage regulator requires fewer components to implement, easing chip shortage concerns.
3. The significantly lower efficiency of a linear regulator is not an issue as this regulator has a capacity of $1\,\mathrm{A}$, which is more than enough to run the devices on board.

3.1.2 USB FIFO

The USB FIFO is a queue that sends data out to the laptop over USB 2.0 Full Speed (FS), capable of up to 480 Mbps. This circuit takes a parallel byte stream from the FPGA and converts it to a USB 2.0 data stream. The FT232HQ USB FIFO made by FTDI is used to implement this functionality. This chip was selected over other USB Interface ICs for the following reasons:

1. Availability. USB Interfaces are one of many ICs in short supply in the ongoing chip shortage. In fact, this IC was ordered for the project after ELEC390 in July 2021 and only received in August 2022.
2. Extensive preexisting software tooling. FTDI provides a driver and configuration utility for extracting data from their USB interfaces. This significantly reduces the software development work required to build a sample streaming interface. This ease of use is critical in a project with such a short timeline.

For debugging and circuit visibility, all digital control signals on the FT232HQ are connected to test points. The schematic sheet also contains some surrounding circuitry in the form of power filtering, a clock oscillator,

and configuration EEPROM.

3.1.3 Current Monitor

The current monitor page contains a hall effect current sensor circuit. The circuit uses the ACS720KLATR current sensor. This sensor was chosen due to its supply chain availability and the fact that it reports its reading through a voltage output. The voltage output is linearly proportional to the current flowing through the sensor plus a DC offset. This is a preferable interface as it can be connected to an ADC channel without having to write a new driver for the specific sensor on the FPGA. As such, the output of the current sensor is connected to FPGA_ADC0, which is one of two ADC channels available on the FPGA module.

3.1.4 ADC

The TeachEE ADC schematic sheet contains an AD9288BSTZ dual channel $40\,\mathrm{MSPS}$ ADC. This chip provides two additional high-speed voltage channels in addition to the ADC channels provided by the FPGA module. The data outputs for channels A and B are sent to the FPGA along with the sample clocks. This ADC was selected for its FPGA friendly control interface. Since the control consists of only a data bus and sample clock, it is trivial to write an FPGA module to collect the data in comparison to other SPI and I2C based converters on the market.

Bypass capacitors and ferrite beads are used to power the ADC for the cleanest reference voltage possible. Additionally, the PCB contains non-populated footprints for low pass filters, and filter bypasses to simplify testing and debugging. For the digital side, all data busses and clocks have test points so multimeters or oscilloscopes can be connected to debug the FPGA control signals.

3.1.5 Analog Front End (AFE)

The Analog Front End (AFE) sits in front of the inputs of the high-speed ADC. As a result, this sub-circuit is duplicated twice in the top level sheet shown in Figure 3 to cover both channels. The AFE serves the following purposes in the data acquisition hardware.

1. Protect the ADC from over-voltage conditions. The front-end must clamp any voltages that would otherwise destroy the ADC.

2. Convert the signal from single ended to differential. The ADC takes in a differential input, so the AFE must convert the input voltage observed at the connector to a differential signal of equivalent magnitude.

This functionality is implemented in hardware using an AD8138 differential ADC driver. This all-in-one component takes care of voltage clamping and single-to-differential conversion. Moreover, it has sufficient analog bandwidth to avoid reducing the effective bandwidth of the ADC. For debugging, the sub-circuit contains 0-ohm bypass resistors that can be optionally soldered to send the unclamped signal directly to the ADC.

3.1.6 FPGA Module

The FPGA module is implemented using a CMOD A7-35T Xilinx Spartan 7 Breakout Board. It is connected to the PCB via two 28-pin headers. Two pins are used for power and ground while the rest are used for FPGA IO pins. The USB FIFO and ADC control pins are connected to the FPGA IO. There are also two special pins which can be used as ADCs. These two pins are used to implement the current and voltage channels at 1 MSPS. The current sensor output is wired to one of the channels and a BNC connector is connected to the other through a low pass filter for voltage sampling.

The FPGA module schematic had one mistake that was not discovered until the physical PCBs had already

been manufactured and delivered. To use the parallel-bus interface of the FT232HQ to send data, the FPGA must lock to a $60\,\mathrm{MHz}$ output clock from the FT232HQ. During layout, the FPGA pin wired to this clock was swapped to a normal GPIO pin rather than a Multi Region Clock Capable (MRCC) pin. As a result, the FPGA place and route was failing during compilation, as the FPGA could not route this clock from a standard GPIO pin. This issue was resolved by waiving the warnings and optimizing the SystemVerilog code to still pass timing without a proper clock route for the $60\,\mathrm{MHz}$ signal. Further information on this process is given in Section 3.2.

3.1.7 Connectors

The connectors page contains all the external IO connections for TeachEE. Specifically, the USB mini connector for data and three BNC connectors for scope probes. The current sensor is exposed on its own separate pin header. The BNC connectors are connected to the appropriate ADC input or AFE. Also, each connector has the appropriate low pass filters to satisfy Nyquist's theorem. Each connector also includes all the standard capacitors and resistors needed to connect an oscilloscope probe from the lab kit sold at the Queen's bookstore.

3.2 FPGA

The FPGA datapath and implementation can be summarized by examining the interconnections between modules in the top level SystemVerilog module. Below is each section of the top level TeachEE RTL module broken down and explained. The entire RTL codebase can be viewed in Appendix D.

3.2.1 Top Level Module Interface

Given below is the interface to the top level module of the TeachEE RTL code.

```
module teachee (
  input  var logic cmod_osc, // 12 MHz provided on the CMOD
  input  var logic ftdi_clk, // 60 MHz provided by the FTDI
```

```
  // FTDI Control Interface
  input  var logic ftdi_rxf_n,
  input  var logic ftdi_txe_n,

  output var logic ftdi_rd_n,
  output var logic ftdi_wr_n,
  output var logic ftdi_siwu_n,
  output var logic ftdi_oe_n,

  output var logic[7:0] ftdi_data,

  // High Speed ADC IO Interface
  output var logic hsadc_a_enc,
  input var logic[7:0] hsadc_a,

  output var logic hsadc_b_enc,
  input var logic[7:0] hsadc_b,

  input var logic[1:0] xa_n,
  input var logic[1:0] xa_p,
  ...
);
```

As shown above, the FPGA takes in a 12 MHz clock provided by an external oscillator and the 60 MHz clock from the FT232HQ. After the clocks, IO for the FT232HQ and AD9288BSTZ control signals are defined. Some additional pins spare pins and LED signals at the end of the interface declaration are not shown as they were used for debugging purposes.

3.2.2 PLL Clock Generation

After the module interface is declared, the PLL system clock generator is initialized. This is an IP core provided by the FPGA manufacturer Xilinx that takes the oscillator 12 MHz input and outputs a variety of useful clock signals for the rest of the design. Given below is the IP core module instantiation.

```
    teachee_pll teachee_pll_ip_inst (
        // Clock out ports
        .clk_100(clk_100),     // output clk_100
        .clk_50(clk_50),     // output clk_50
        .clk_25(clk_25),     // output clk_25
        .clk_20(clk_20),     // output clk_20
```

```
    .clk_10(clk_10),     // output clk_10

    // Status and control signals
    .reset(0), // input reset
    .locked(locked),       // output locked

    // Clock in ports
    .cmod_osc(cmod_osc)      // input cmod_osc
);
```

The module takes the external oscillator clock as an input and outputs 100, 50, 25, 20 and 10 MHz clocks. Different frequencies can be specified within the Xilinx Vivado IDE used to flash the FPGA. In the current implementation, only the 100 and 10 MHz clocks are used. 100 MHz is used throughout the top level system and 10 MHz is used as a sampling clock in the high speed ADC. One clock that is not covered by this PLL is the 60 MHz clock generated externally by the FT232HQ. The PLL also has a `locked` output. When driven high, this signal indicates that the PLL is locked to the input clock and the output clocks are functioning correctly. This signal is used to send a reset signal to all modules, holding them in reset until the PLL has locked and all clocks are up and running.

3.2.3 Interface Instantiations

Given below are the instantiations for the different interfaces used to make connections between the submodules. Interfaces in SystemVerilog can be thought of as groupings of wires that make managing inter-module connections easier as each wire does not need to be explicitly declared.

```
hsadc_interface hsadc_ctrl ();
...
axis_interface #(
    .DATA_WIDTH(2 * XADC_DRP_DATA_WIDTH)
) xadc_sample_channel (
    .clk(sys_clk),
    .rst(reset)
);

axis_interface #(
    .DATA_WIDTH(16)
) hsadc_sample_channel (
```

```
    .clk(sys_clk),
    .rst(reset)
);
```

The first interface declared is for the AD8138ARZ high-speed ADC. External IO signals are assigned to the different control wires in the interface. Next, the various AXIS interfaces required to connect the modules are instantiated. The interface width is specified using the `DATA_WIDTH` parameter. 8-bit interfaces are used to connect the packet stream output for a given ADC to the FT232HQ wrapper module that adapts the AXI Stream to FT232HQ control signals. The remaining two interfaces hold the sample data from the XADC and high-speed ADC respectively. The XADC is the dual channel 16-bit ADC built into the FPGA. The two sample data interfaces `hsadc_sample_channel` and `xadc_sample_channel` are each connected to the outputs of adapter modules that adapt the native interface of the device to an AXI Stream. The AXI Stream interface used is a custom wrapper for all the standard signals found in an AXI stream. The interface definition can be found in Appendix D.1.3.

The AXIS Protocol, also known as AXI Stream, is derived from ARM's AXI bus specification. Additional information on the AXI Stream protocol and the external libraries used to support it can be found in Section 3.2.7.

3.2.4 FT232HQ AXIS Wrapper

After instantiating the interfaces, they are used to make connections between the various submodules. Shown below is the `ft232h` module instance used to send data to the computer over USB.

```
ft232h usb_fifo (
    .ftdi_clk(ftdi_clk),

    .ftdi_rxf_n(ftdi_rxf_n),
    .ftdi_txe_n(ftdi_txe_n),

    .ftdi_rd_n(ftdi_rd_n),
    .ftdi_wr_n(ftdi_wr_n),
    .ftdi_siwu_n(ftdi_siwu_n),
```

```
        .ftdi_oe_n(ftdi_oe_n),

        .ftdi_adbus(ftdi_data),

        // Programmer AXIS Interface
        // CHANGE THIS BASED ON WHETHER YOU WANT TO USE XADC OR HSADC
        .sys_axis(xadc_usb_axis.Sink)
    );
```

This module consists of a state machine that adapts the AXI Stream of sample packets to control signals of the FT232HQ. The input is the `sys_axis` stream and the output is all the `ftdi_*` control signals used to produce data over USB. Wrapping the FT232HQ interface in a standard AXI Stream makes it trivial to interface with all the other modules in the TeachEE source code. The same design approach is used throughout the codebase to ensure compatibility between the various hardware interface modules. The state machine for the module is shown in Figure 4.

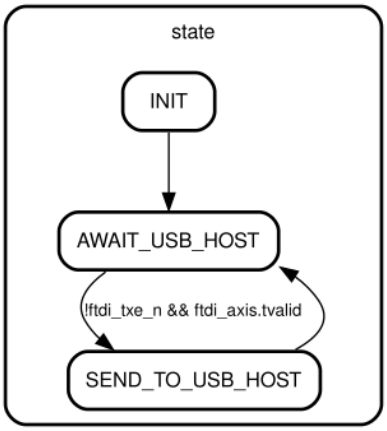


Figure 4: FT322H AXI Wrapper State Machine

The state machine waits for the AXI Stream to have data available and for the FT232HQ to be ready for new data. When this condition is satisfied, the byte is sent out and the state machine automatically transitions back to the await state.

Regarding the comment in the code above about changing the `sys_axis` stream input, in this case, the XADC is used.

3.2.5 XADC IP Core

Shown below is the XADC IP core used to extract data from the internal XADC on the FPGA. This IP core is made custom using the XADC Wizard tool in Xilinx Vivado. The core is configured to only output the two FPGA ADC channels in use on the TeachEE PCB via a Dynamic Reconfiguration Port (DRP) interface. The DRP signals are then fed into an AXIS adapter module for interface standardization.

```
xadc_teachee xadc_teachee_inst (
    // Clock and Reset
    .dclk_in(sys_clk),          // input wire dclk_in
    .reset_in(reset),         // input wire reset_in

    // DRP interface
    .di_in(0),              // input wire [15 : 0] di_in
    .daddr_in(xadc_daddr),        // input wire [6 : 0] daddr_in
    .den_in(xadc_den),            // input wire den_in
    .dwe_in(0),            // input wire dwe_in
    .drdy_out(xadc_drdy),        // output wire drdy_out
    .do_out(xadc_do),            // output wire [15 : 0] do_out

    // Dedicated analog input channel (we do not use this)
    .vp_in(0),              // input wire vp_in
    .vn_in(0),              // input wire vn_in

    // analog input channels, vaux4 is pin 15 = VIOUT of current sensor
    // vaux12 is pin 16 = low speed voltage channel.
    .vauxp4(xa_p[0]),            // input wire vauxp4
    .vauxn4(xa_n[0]),            // input wire vauxn4
    .vauxp12(xa_p[1]),          // input wire vauxp12
    .vauxn12(xa_n[1]),          // input wire vauxn12

    // conversion status signals
    .channel_out(),  // output wire [4 : 0] channel_out
    .eoc_out(),          // output wire eoc_out
    .alarm_out(),      // output wire alarm_out
    .eos_out(xadc_eos),          // output wire eos_out
    .busy_out()        // output wire busy_out
);
```

3.2.6 ADC AXIS Wrappers

Similarly to the `ft232h` module, both the XADC and high-speed ADC have AXIS wrapper modules. Both wrapper modules function in the same manner, they await new data from the ADC and place it in AXIS compliant FIFO queues. The `sample_stream` output is internally connected to this queue where samples can be consumed by the FT232HQ. Given below is the instantiation of both modules.

```
xadc_drp_addr_t xadc_daddr;
var logic xadc_den;

var logic xadc_drdy;
var logic[XADC_DRP_DATA_WIDTH-1:0] xadc_do;
var logic xadc_eos;

xadc_drp_axis_single_stream xadc_drp_axis_adapter_inst (
    .xadc_dclk(sys_clk),
    .xadc_reset(reset),

    // DRP and Conversion Signals
    .xadc_daddr(xadc_daddr),
    .xadc_den(xadc_den),
    .xadc_drdy(xadc_drdy),
    .xadc_do(xadc_do),

    .xadc_eos(xadc_eos),

    .sample_stream(xadc_sample_channel.Source)
);

hsadc_axis_wrapper hsadc (
    .sample_clk(clk_10),
    .stream_clk(sys_clk),
    .reset(reset),

    .hsadc_ctrl_signals(hsadc_ctrl.sink),
    .sample_stream(hsadc_sample_channel.source)
);
```

The XADC wrapper takes in the DRP control signals from the XADC IP core and the high-speed ADC wrapper gets all the control signals from a single interface declared earlier in the file.

3.2.7 AXIS and Verilog AXIS

AXIS, also referred to as AXI Stream, is the standard protocol used throughout the FPGA datapath. While there are many signals in an AXI interface, the primary signals are called `tdata`, `tready`, and `tvalid`. Only when `tready` and `tvalid` are asserted can data be consumed from the `tdata` bus. `tready` is controlled by the receiver so that it can regulate the rate at which it consumes data, while `tvalid` is controlled by the sender to notify the receiver when new data is ready on the `tdata` lines. The full specification contains many additional signals, but these three are the ones most often manipulated during data transmission [3]. Figures 5 and 6 show the AXIS read and write timing diagrams respectively.

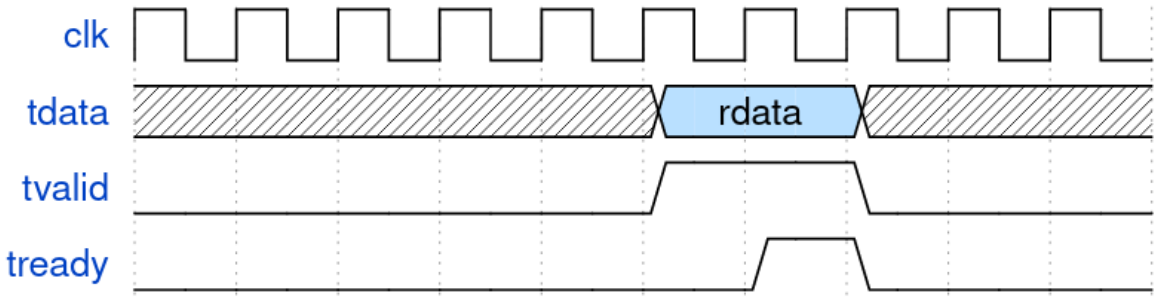


Figure 5: AXIS Read Transaction Timing Diagram

In AXIS read and write transactions, the producer is responsible for `tdata` and `tvalid` signals while the consumer toggles `tready`. This allows the consumer to know when new data becomes available and it is able to signal to the producer when it is ready to accept it via `tready`. All three signals should only be asserted or deasserted in synchronization with the system clock, as per the specification. As shown in Figure 5, `tvalid` is asserted on the same rising edge as the new data coming onto the `tdata` bus. The consumer sees this and asserts tready on the next rising edge consuming the data. In the case where multiple words of data are queued by the producer, the `tdata` value will advance to the next byte available after each cycle where both `tvalid` and `tready` are asserted. This allows a new read to take place every cycle if the producer has the data and the consumer is ready for it. This simple protocol yields strong performance and

flexibility with minimal FPGA Lookup Table (LUT) resources used. It should be noted that for any AXIS transaction to take place, `tready` and `tvalid` must both be true.

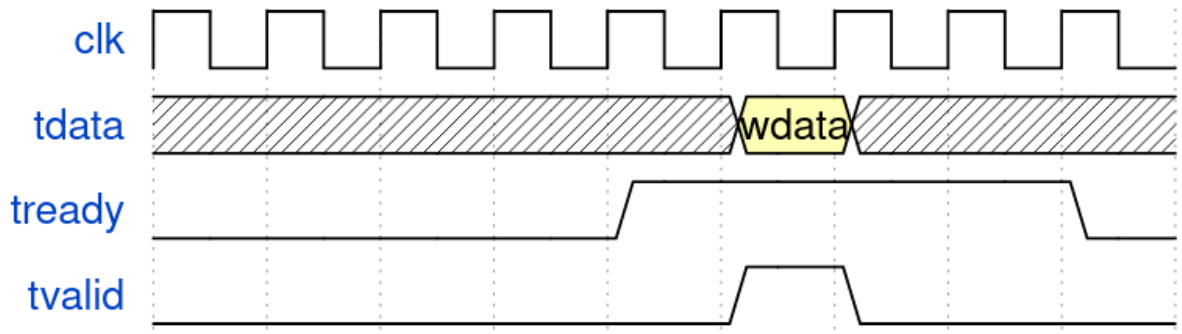


Figure 6: AXIS Write Transaction Timing Diagram

In the AXIS write case shown in Figure 6, the consumer first asserts `tready`, which tells the producer that it can accept new data. Once the producer has data, it asserts the value on `tdata` and drives `tvalid` high on the same rising edge. Since `tready && tvalid` is true, the data is written to the consumer successfully. This design pattern is replicated repeatedly throughout the FPGA codebase provided in Appendix D.

In addition to the simplicity of the protocol, there is significant library support for AXI Streams. In the case of TeachEE, the `verilog-axis` library is used [4]. This library contains a variety of useful Verilog modules, all of which expose an AXIS interface at their inputs and outputs. The standard interface across all modules trivializes connecting up the library modules with TeachEE modules. The only modification made to these library modules was to wrap them in another more ergonomic module. Since the library is written in Verilog, it does not have access to interfaces, which are a feature of SystemVerilog only. As a result, each AXIS signal is written out individually making instantiating and connecting the modules unwieldy. To bundle all these signals into one module port, the `axis_interface` is used. For each module in the `verilog-axis` library used by TeachEE, a wrapper module is written that places all the signals for the AXI Streams in AXIS interfaces, making module connectivity far simpler with SystemVerilog. This is possible due to the Xilinx Vivado project build configuration that cross compiles both Verilog

and SystemVerilog. The interface is also designed to seamlessly handle the producer and consumer cases discussed above using modports. The `Source` modport can be used on producer stream interfaces and the `Sink` modport can be used for consumers. The modports switch the direction of the signals from `input` to `output` depending on the use case.

`verilog-axis` contains many modules but there are three that were employed when writing the RTL code for TeachEE.

The first module is `axis_async_fifo`, which is wrapped by `axis_async_fifo_wrapper`. This module constructs a First-In-First-Out (FIFO) Queue and is used repeatedly throughout the code. The first and foremost purpose of the Queue is to provide sample memory to ensure samples are not dropped when the rate of production deviates from the rate of consumption. Secondly, the FIFO is asynchronous, meaning it can use different clock signals for its input and output. The module has internal synchronization logic to bring data across clock domains. This functionality was critical in the TeachEE RTL design as there were multiple clock domains that needed efficient data transfer between them. The top-level system, USB FIFO, HSADC, and XADC all had different clock signals and had their data inputs buffered by asynchronous FIFOs.

The second key module is the `axis_adapter`, which is wrapped by `axis_adapter_wrapper`. This module takes an AXI Stream and adapts it to a smaller width specified as a parameter in the module instantiation. This module was critical in getting the 32-bit sample stream from the XADC to fit the 8-bit stream accepted by the USB FIFO.

The final module used from the library was `cobs_encode`, which is wrapped by `cobs_encode_wrapper`. This module takes an 8-bit input stream and outputs the same data encoded using COBS. The COBS encoder is highly efficient and only creates two additional bytes of overhead, minimizing USB link bandwidth wastage [5]. To transmit samples to the computer, the 32-bit stream from the XADC (or 16-bit from the

HSADC) is adapted to 8 bits using `axis_adapter`. The resulting 8-bit stream is fed into the COBS encoder, whose output is connected to the USB FIFO. These two modules make up the packet encoding stack.

Perhaps the most important benefit of AXIS over other streaming interfaces is its direct compatibility with the more fully-featured AXI bus. AXI is a standard memory-mapped bus interface used in ARM processors [6]. Since all modules in TeachEE's FPGA codebase use AXI Streams, the platform is highly extensible and can be connected to ARM coprocessors with minimal effort. Such a design extension would make TeachEE architecturally similar to more expensive benchtop oscilloscopes that make use of both an ASIC and traditional processor through a high-speed interconnection protocol. An example of this architecture is Keysight's MegaZoom ASIC [7].

3.3 Software

TeachEE's software workload is split between three modules, each of which runs in its own thread. The first module is the USB Manager, responsible for reading and decoding packets from the hardware driver. The second is the App, responsible for updating and displaying the user interface. Finally, the Controller facilitates communication between the two and provides additional sample processing.

The software for TeachEE follows an improved threading model described in Figure 7 below. This design change was necessitated by software performance issues encountered in the initial design, described in Section 4.3.1. The USB manager and Controller share one set of two buffers, referred to as the USB Buffers, and the App and Controller share another set of two buffers, the App buffers. The threads coordinate their workflows such that each thread can exclusively operate on a buffer at a time. For example, while the USB Manager is writing samples to USB Buffer 0, the Controller can copy samples from USB Buffer 1 to App Buffer 1, and the App can read from App Buffer 0. After every work cycle, each thread swaps to the other buffer. This protocol parallelizes the workload as much as possible and minimizes thread block times.

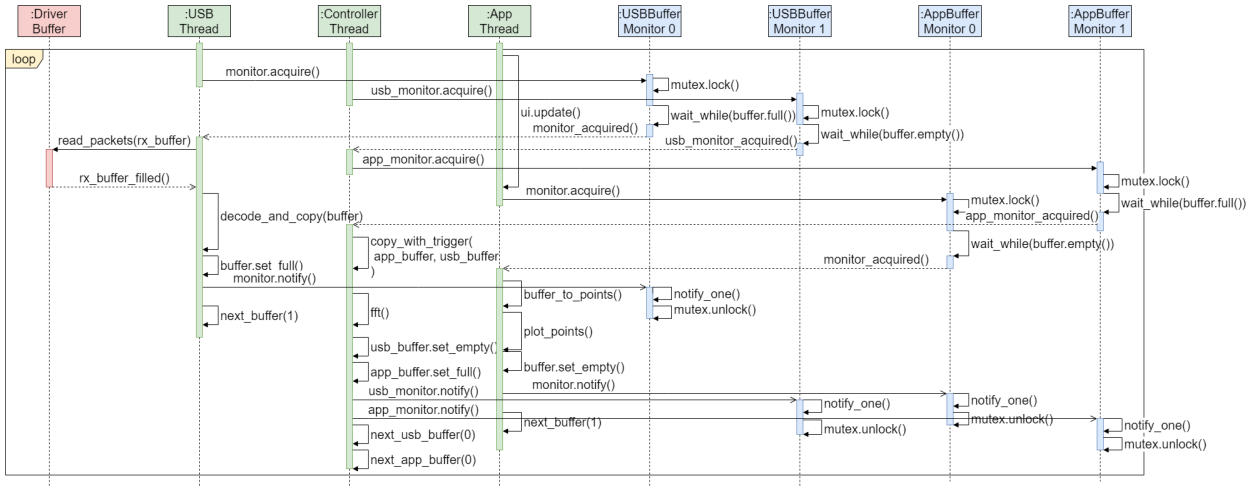


Figure 7: Threading Model Sequence Diagram

The sequence diagram above simplifies the buffer representation. Each buffer in the two sets actually represents several buffers/channels in the `Channels` struct. In the current implementation, two channels are supported and every buffer operation operates on both channels simultaneously. Furthermore, each `Channels` struct is wrapped in a `BufferState` enum, representing the emptiness/fullness of the channels, which in turn is guarded by a monitor. To allow these monitors to be shared between threads, they are wrapped by Rust's shared pointer, the `Arc`. Figure 8 below illustrates the shared data's configuration. Note that there is an additional field in the `Channels` struct used to store the results of frequency spectrum analysis performed in the Controller thread. This field is unused in the USB Buffer `Channels`.

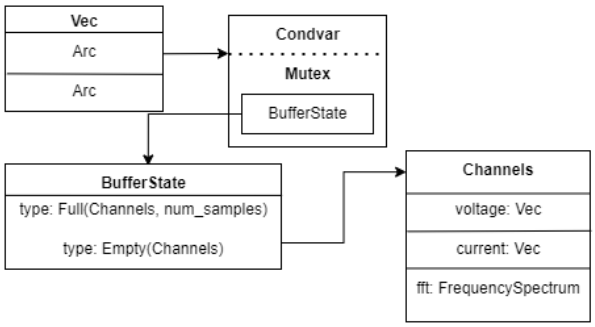


Figure 8: Shared data configuration representing one set of "buffers"

In Rust, a monitor is implemented as a condition variable and mutex tuple. The condition variable allows threads to wait while a condition is true. The mutex ensures that only one thread is allowed access to a given `Channels` at a time. A thread acquires a monitor by first locking the mutex. Once the mutex is locked, the thread checks the `BufferState`. If the thread intends to write to the buffers, but the buffers are `Full`, it unlocks the mutex and goes to sleep until it receives a notification that the buffers have been emptied. The same applies to threads attempting to read from `Empty` buffers. Once the thread is notified, it is woken from the wait function with the mutex reacquired. A thread releases a monitor by setting the `BufferState` to the appropriate type and notifies a thread waiting on the condition variable. The mutex is unlocked implicitly when it goes out of scope. An example of this procedure (run by the App thread) is given in the code block below.

```
// Wait until controller thread has filled the current buffer.
let mut buf_state = condvar
    .wait_while(mutex.lock().unwrap(), |buf_state| buf_state.is_empty())
    .unwrap();
// Get the Channels buffers.
let (channels, num_samples) = buf_state.unwrap();
...
// Release the monitor.
*buf_state = BufferState::Empty(channels);
condvar.notify_one();
```

3.3.1 The USB Manager

The USB Manager thread reads packets from the FTDI receiver queue into a buffer, decodes them into voltage and current samples, and populates the USB Buffers. The packets are read from the queue using an FTDI library. The thread always reads 120 000 bytes at a time so that after decoding, the App displays approximately 20 000 samples each cycle. This ensures that the plotted waveform does not expand or contract in the horizontal direction, which improves the user's experience.

The decoding function decodes the six-byte COBS packets into pairs of voltage and current samples. The main decoding loop is described in the code below.

```
...
'outer: while src_index + PACKET_SIZE < end
    && dst_index < channels.voltage1.len()
{
    // Packet error: offset byte not in [1, 5] or packet not delimited
    // with 0
    if self.rx_buf[src_index] == 0
        || self.rx_buf[src_index] >= PACKET_SIZE as u8
        || self.rx_buf[src_index + PACKET_SIZE - 1] != 0
    {
        // Find the next 0 and continue from there
    } else {
        let packet = &self.rx_buf[src_index..(src_index + PACKET_SIZE)];
        let mut block = packet[0] - 1;
        // Two bytes of packet overhead
        let mut decoded: [u8; PACKET_SIZE - 2] = [0; PACKET_SIZE - 2];
        let mut decoded_index = 0;

        // Note: the packet index is just the decoded_index + 1 (skip
        //       the first offset byte)
        // Example packet: 01 02 22 02 44 00
        // Decodes to:        00 22 00 44
        while decoded_index < decoded.len() {
            if block > 0 {
                decoded[decoded_index] = packet[decoded_index + 1];
            } else {
                decoded[decoded_index] = 0;
                block = packet[decoded_index + 1];
            }
            decoded_index += 1;
            block -= 1;
        }

        // Map digital values with 4096 discrete levels to samples
        channels.voltage1[dst_index] =
            (((decoded[3] as u16) << 4) | decoded[2] as u16) as f64
                * 3.3 / 4095.0;
        channels.current1[dst_index] =
            ((((decoded[1] as u16) << 4) | decoded[0] as u16) as f64
                * 3.3 / 4095.0 - 1.5) * (1.0 / 0.09);
        src_index += PACKET_SIZE;
        dst_index += 1;
    }
}
...
```

The full USB Manager code is provided in Appendix H.2.

3.3.2 The Controller

The Controller thread copies samples from the USB Buffers to the App Buffers. If enabled by the user, it also performs rising edge triggering and/or frequency spectrum analysis on the samples. The triggering algorithm uses hysteresis to find the first rising edge while ignoring rising edges from noise. It does this by setting the hysteresis to 25% of the signal peak-to-peak so that only larger rising edges are triggered on [8]. Triggering is done on both the voltage and current channels with separate thresholds provided by the user. The triggering function is provided below.

```rust
/// Returns position of first rising edge based on threshold
fn hysteresis_trigger(src: &[f64], trigger: f64) -> usize {
    // Hysteresis width as fraction of peak to peak
    const HYSTERESIS_WIDTH: f64 = 0.25;
    let mut max_val = f64::MIN;
    let mut min_val = f64::MAX;
    for &val in src.iter() {
        max_val = f64::max(max_val, val);
        min_val = f64::min(min_val, val);
    }
    let hysteresis = (max_val - min_val) * HYSTERESIS_WIDTH;

    // Trigger when signal first falls below 'trigger - hysteresis'
    // and then rises above trigger
    let first_lower = src.iter().position(|&val| val < trigger -
        hysteresis);
    if let Some(lower_idx) = first_lower {
        let first_higher = src[lower_idx..].iter().position(
            |&val| val >= trigger);
        if let Some(higher_idx) = first_higher {
            return lower_idx + higher_idx;
        }
    }
    // Don't trigger if it failed
    0
}
```

Spectrum analysis is performed using the `spectrum-analyzer` library. The provided function takes the sample data, sample rate, desired frequency range, and an optional scaling function and returns a `FrequencySpectrum` struct. This struct contains an array of frequency/magnitude tuples, which can

be passed to the App to be displayed.

The full Controller code is provided in Appendix H.3 and includes the trigger and spectrum analysis procedures.

3.3.3 The App

`egui` is a simple and fast GUI library written entirely in Rust [2]. It is an immediate mode GUI, which means that the UI is refreshed at approximately sixty frames per second and does not require any event handlers. This makes it ideal for writing simple and highly interactive UIs quickly as it eliminates the need for complex callbacks, UI objects, or state storage. `egui` was chosen to implement the front end for TeachEE instead of JavaScript to avoid the overhead of message passing.

The function below adds and manages a pair of offset/scale sliders, and is an example of the simplicity of the `egui` library. This function is called three times to add the three pairs of sliders shown in Figure 9.

```
fn offset_scale_sliders(
    ui: &mut Ui,
    offset: &mut f64,
    o_range: RangeInclusive<f64>,
    scale: &mut f64,
    s_range: RangeInclusive<f64>,
) {
    ui.columns(2, |uis| {
        uis[0].label("Offset");
        uis[0].add(Slider::new(offset, o_range).clamp_to_range(false));
        uis[1].label("Scale");
        uis[1].add(
            Slider::new(scale, s_range)
                .clamp_to_range(false)
                .logarithmic(true),
        );
    });
}
```

The App stores two structs: an `AppData` and a `UIControls`. `AppData` stores the App Buffers, trigger thresholds, and frequency spectrum enable, and is shared with the Controller thread. `UIControls` stores

the current UI state set by the user, e.g., offsets, scaling, trigger thresholds, and spectrum analysis enable/disable.

TeachEE implements several features reflected in the UI, including:

1. Individual channel scaling and offsetting (both horizontal and vertical)
2. Rising edge triggering with user-provided thresholds
3. Signal plotting with cursor and various navigation options
4. Plotting of signals in the frequency domain
5. Exporting plotted signals to a CSV file
6. Ability to turn individual channels on and off

Figure 9 shows all of TeachEE's features in use at the same time.

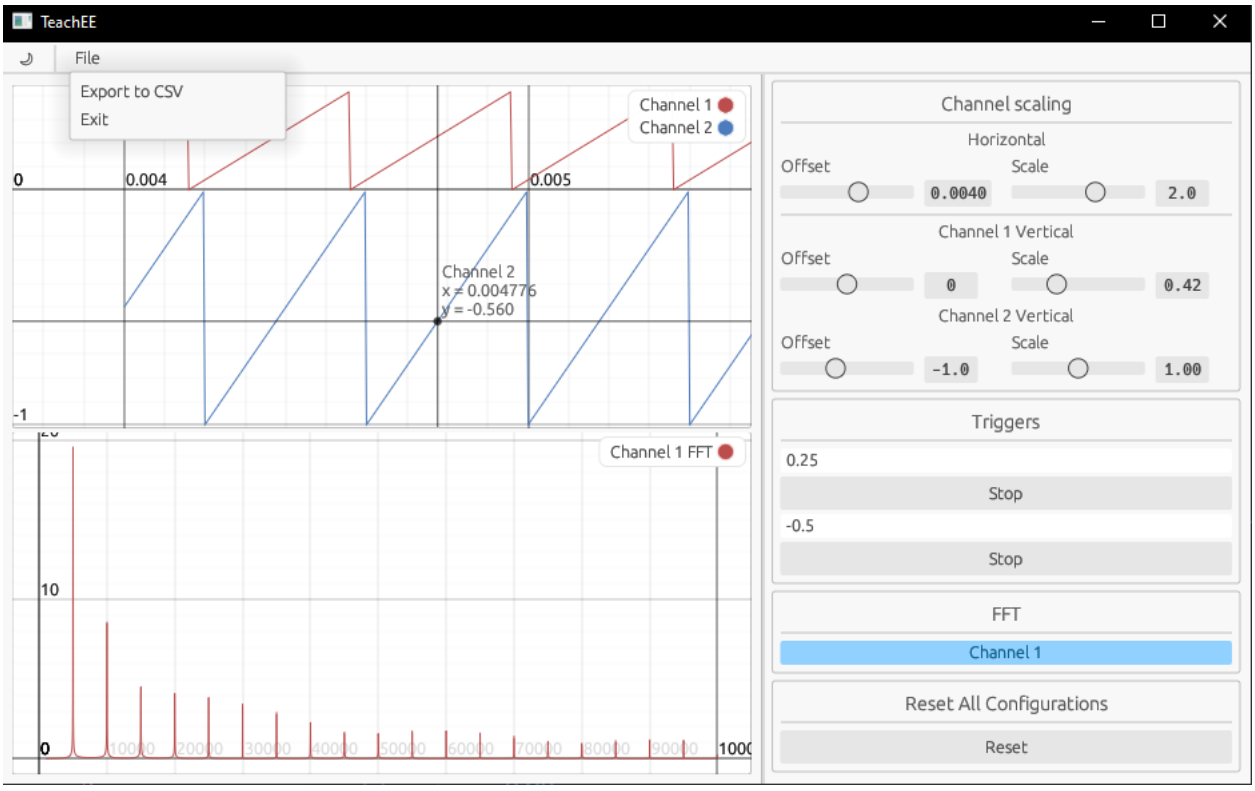


Figure 9: Example screenshot showcasing all UI features

The full App code is provided in Appendix H.4 and contains the UI implementation.

4 Testing, Evaluation & Verification

The testing and validation of TeachEE consisted of testing the hardware/PCB, FPGA, and software subsystems. The following subsections cover each subsystem of TeachEE in full detail. Testing was performed in an iterative manner for all subsystems: a design component was tested in both isolation and integration after being implemented to ensure that it was implemented and integrated correctly.

4.1 PCB Verification

The PCB verification process was broken up into physical inspection of the board and functional testing. The PCBs were both fabricated and assembled by an external vendor. Upon receiving the fully assembled TeachEE units, the following physical inspection steps were taken.

4.1.1 Physical Inspection and Acceptance Testing

1. Microscope examination of the PCB. Looking at each solder joint to identify any cold or weak links, tombstoning etc.

2. Assembly BOM check. Examine the components populated on the board. Ensure nothing was missed and make sure that components that were not supposed to be populated are left blank, as per the instructions to the vendor.

3. Solder the linear voltage regulator in for power on testing. This was not populated initially because it is through-hole. Through-hole components are expensive to assemble so they were done by hand for this project.

4. Basic electrical check. Use a DMM to check for any short circuits between the power rails and ground.

5. Power on TeachEE via the USB Mini port. Verify that all LEDs turn on and that the voltage regulator is outputting the correct voltage. Check for heat around the board indicating a possible short circuit.

Figures 10 and 11 are the front and back of the PCB respectively. These photos were provided by the vendor and provide a clear picture of the boards at the point of delivery and inspection. Figure 12 is a closeup of the area around the USB FIFO as this was a high density area in the PCB layout.

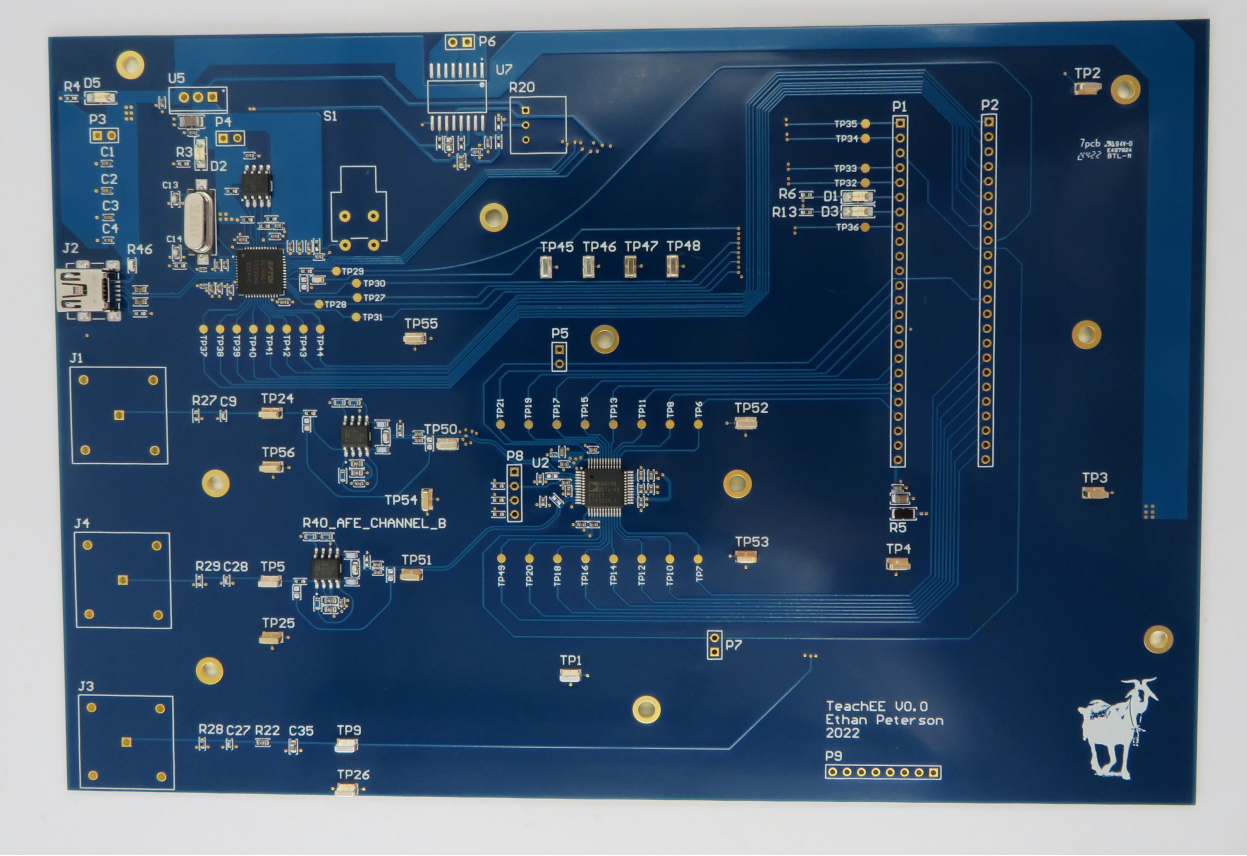


Figure 10: PCB Front Side After Delivery

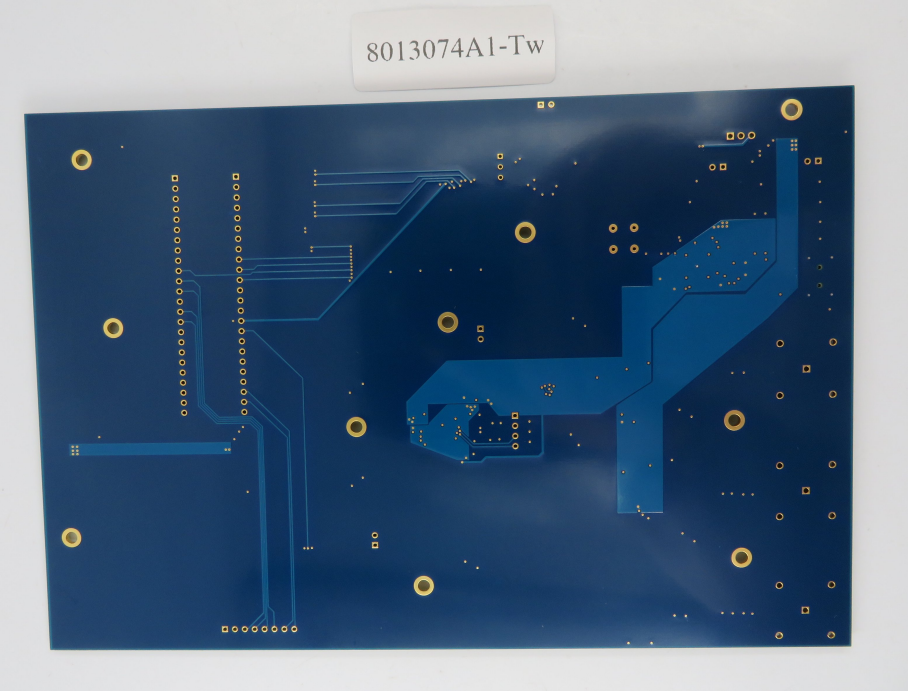


Figure 11: PCB Back Side After Delivery

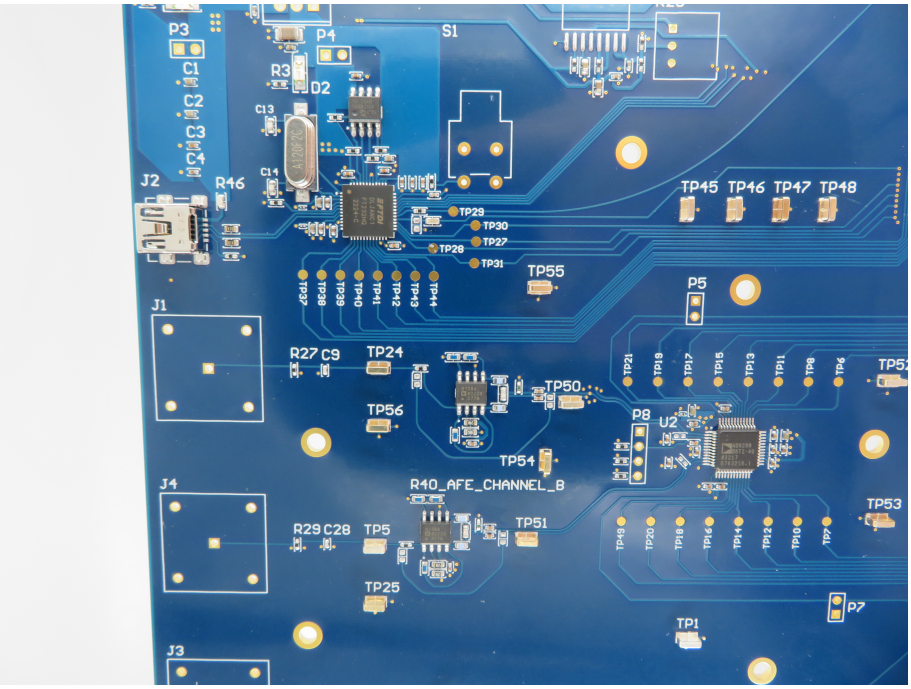


Figure 12: Close up of the FIFO on the PCB

4.1.2 Functional Testing

All three assembled PCBs passed these inspection checks. Next, functional testing was done on the three units. The functional tests are as follows.

1. Plug TeachEE to a computer via the USB Mini connector. Open the FTDI Utility to check if the USB FIFO is detected. Ensure that the EEPROM can be read and that the USB device descriptor can be changed to `TeachEE`. After making the EEPROM change, unplug the device and plug back in. Check that the USB device is detected with the right descriptor information. This descriptor is used by the desktop software to identify the USB device as the oscilloscope.

2. Use the `reader.py` script found in Appendix G to set the FT232HQ on TeachEE to synchronous data transmission mode 40.

3. After setting the device to synchronous mode, connect an oscilloscope to the FT232HQ clock test point on the PCB. Verify that a 60 MHz clock is supplied to the FPGA. The waveform should be a triangle wave. A screen capture of the clock waveform is given in Figure 13. The triangle wave is not ideal but the FPGA locks to it successfully nonetheless.

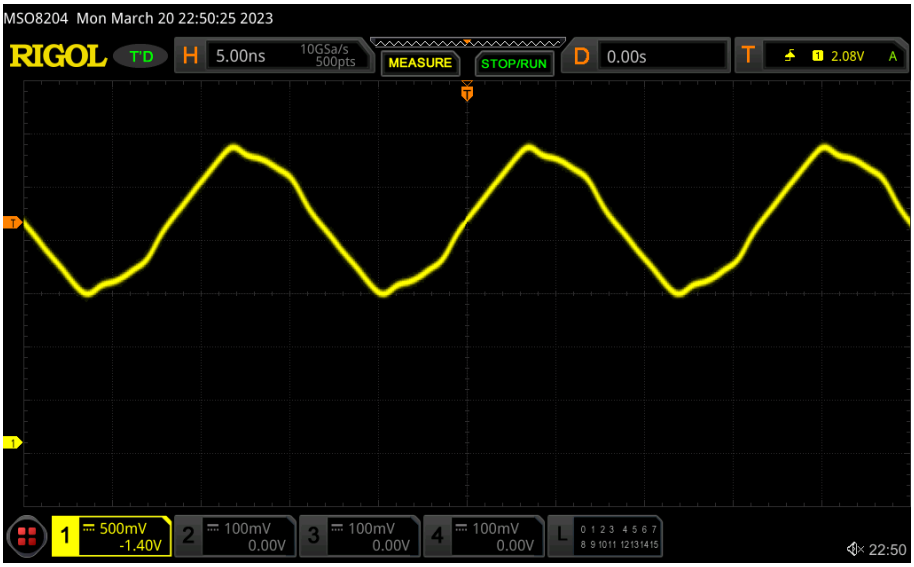


Figure 13: FT232HQ Clock Output Waveform

All three PCBs passed the functional tests enumerated above. At this point, all parts omitted from the assembly process were soldered onto the PCB. This included all through-hole components on the board except for the voltage regulator, which was soldered earlier for the power-up test. Through-hole parts were omitted from the assembly order to keep costs low. Moreover, due to the chip shortage, the hall effect current sensor had to be purchased separately and soldered manually. The through-hole parts that were soldered manually included all the pin headers for the FPGA, BNC connectors for the oscilloscope probes, and through-hole test points. To protect the PCB from any conductive surfaces or ESD, M3 standoff screws were installed into all the mounting holes. After this final assembly step, the PCB was ready for software and FPGA development. The FPGA module can be installed into the pin header and flashed. A fully assembled TeachEE is shown in Figure 14.

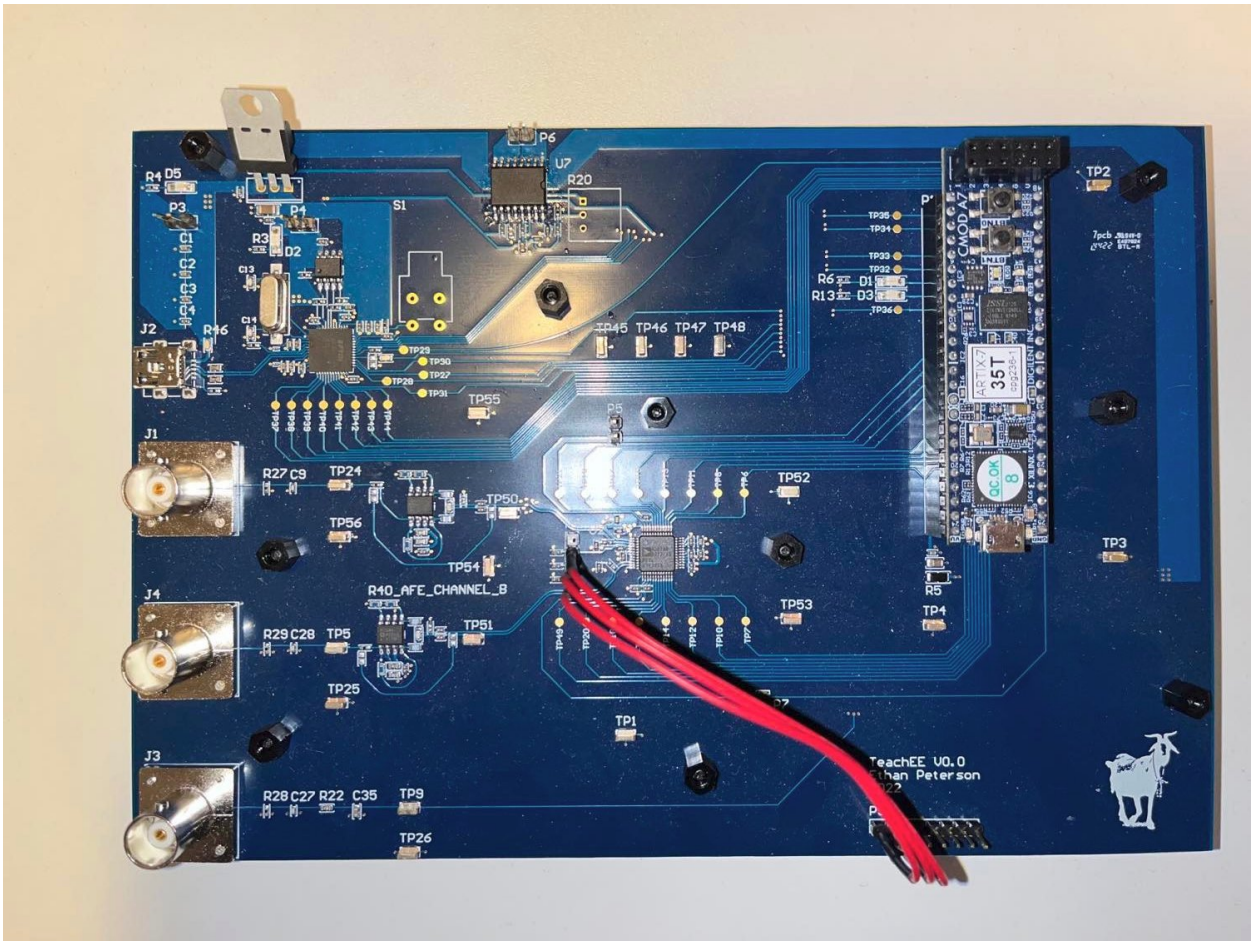


Figure 14: Final Assembled TeachEE

4.2 FPGA Verification

FPGA code testing was a continuous process taking place throughout the development cycle. SystemVerilog modules are tested and verified using “testbenches”. Testbenches are modules that wrap around the module to be tested and modulate the input signals through a variety of test cases designed by the programmer. Example testbenches can be found in Appendix D.6 and D.7 respectively.

The verification of the code takes place on a per-module basis using one testbench per module. This includes complex modules such as the packetizer, but also simple modules that wrap AXIS interfaces. This allows each module to be evaluated and verified independently of one another, ensuring easy integration into the broader TeachEE codebase.

Initially, testbenches were made up of a finite state machine that applied different inputs to the module, with the outputs being manually inspected by the developer in ModelSim. These initial testbenches are shown in Appendix D.6. As the FPGA development process continued, it quickly became difficult to manually inspect simulation waveforms due to the large number of outputs and multi-cycle operations. An example of one of the larger waveforms that needed to be manually inspected is shown in Figure 15.

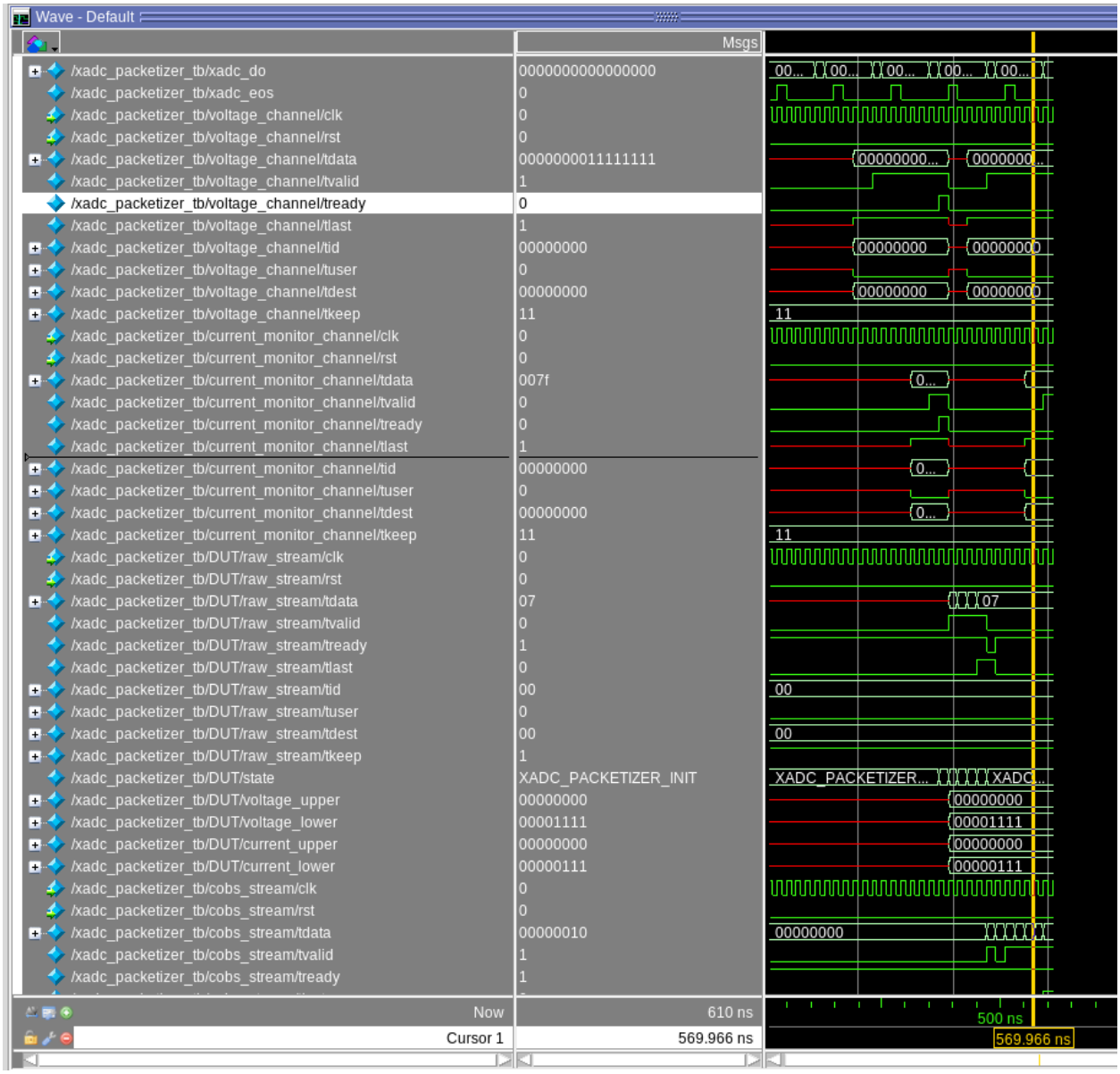


Figure 15: Example ModelSim Waveform from a Packetizer Module

4.2.1 Verification Automation With VUnit

To speed up testbench development and debugging, an automated testing framework, VUnit, was introduced. VUnit provides macros within SystemVerilog to make assertions on the values of registers and wires [9]. An example of a SystemVerilog testbench written for use with VUnit is given in Appendix D.7. This greatly improved development speed as checks were automated and manual inspection was no longer required,

except for specific debugging tasks. Moreover, VUnit also allows testbenches to be specified in a Python script. These Python scripts specify the dependencies and source files needed to run the test as well. To run a testbench, it is as simple as running the corresponding Python script so long as the simulation tool (ModelSim) and VUnit package is installed. Each VUnit testbench has an accompanying Python file. A Python testbench example is provided in Appendix F.

Another significant benefit of using VUnit is that it makes the testbenches easily callable from a command line interface (CLI). In order to further automate the testing, Continuous Integration (CI) was added to the TeachEE code repository. The TeachEE code is hosted in a GitHub repository [10]. GitHub provides a tool called Actions which can be used to run automated scripts on new code changes introduced to the repository in pull requests [11]. Since the VUnit testbenches are easily callable from a script, a GitHub Actions script was introduced to run all the testbenches on each pull request. The full source code is provided in Appendix E. The script installs ModelSim and Vunit, then runs the testbenches, reporting any errors automatically when new code is pushed. This ensured that new code did not break compatibility with older modules.

4.2.2 Hardware Simulation with BFMs

In many cases, additional supporting modules were needed to test a specific module. Examples of this are the `ft232h`, `xadc_drp_axis_adapter`, and `hsadc_axis_adapter`. Since all of these modules were designed to interact with the interface of a physical component on the PCB, mock interfaces were needed to test the module. These device mocks are called "Board Functional Models" or BFMs. Each BFM is designed to respond in the same way to input as the actual device itself. Of course, the BFM has no real data to return so the data coming back will often be a preset value or known pattern. The BFMs are written as finite state machines built to emulate the timing diagrams provided by the manufacturer for a specific part. Usually, they are more limited in their scope compared to the full device interface to limit the potential for bugs. For example, `ft232h_bfm` only implements the write interface of the FT232HQ as the USB

reads are not used in TeachEE. The main problem with BFMs is that they can produce bugs of their own. However, if the BFM is correct, then a module interacting with the hardware modeled by the BFM should work exactly as expected on the physical PCB. An example BFM that mocks the interface of the XADC is given in Appendix D.5.

4.2.3 FPGA Bugs and Solutions

During the FPGA and hardware verification process, the most difficult bugs were at the intersection of the PCB hardware and FPGA. Specifically, bringing up the FT232HQ and transmitting data proved difficult. The main reason for this is that without the USB FIFO up and running, there is little visibility into the FPGA firmware beyond the on-status LEDs and JTAG debugger. As a result, it was extraordinarily difficult to distinguish whether an issue was hardware or FPGA related. Eventually, it was discovered that an error in the state machine caused the `ft232h` module to start sending data before the chip was ready to accept it, leading to lost bytes or no bytes at all. After resolving this misunderstanding of the device interface, it was discovered that the data was being received but bytes were still missing. After probing the test points on the USB FIFO using an external oscilloscope, it was discovered that data is sent for one extra cycle after the USB FIFO stops accepting data, leading to dropped bytes. This was not a misunderstanding of the device function, but rather a misuse of asynchronous assignments in the code.

After resolving the dropped bytes and building a reliable channel, testing the remaining modules in hardware was far simpler as the output of such modules could be fed directly to the connected computer and evaluated.

Aside from Data Integrity, power-on race conditions proved an issue for the FPGA design. Since the design is dependent on an external clock generated by the USB FIFO, the data transmission often did not start up correctly if the clock was not received in a timely manner during the boot sequence. The clock was not available immediately since the device first had be set to synchronous mode by the software to produce the clock. This issue was resolved by probing the clock output pin of the FT232HQ during start-up and

triggering on the first clock edge. Once the source of the problem was established, a clever use of the `locked` output of the PLL was used to hold all modules in reset until the clock becomes available.

4.3 Software Verification

Overall software validation was done throughout the development life cycle using two tools. The first tool was a mock bytestream generator that populates the channel buffers with sawtooth waves. The generator simulates data arriving from the hardware and allowed developers to test Controller and App functionality without TeachEE hardware. This greatly increased development flexibility as the developers could write code and immediately test it remotely. For example, the trigger functionality correctness was verified with these mock bytestreams rather than the hardware. The mock bytestream code is provided in Appendix H.2.3. The second tool was an automated GitHub Action script run on every branch that builds the software, checks code formatting, and runs a linter to catch common mistakes. This tool ensured that code merged to the main branch was written with good style and did not introduce any build errors. This script is provided in Appendix H.5.

4.3.1 Design Iteration

Severe performance issues were encountered while implementing the initial software design described in Section 2.3. The frame rate of the UI was limited to approximately ten frames per second. This issue was caused by high levels of mutex contention: the USB Manager thread would repeatedly acquire the mutex, thereby starving the App thread. Because threads did not take turns, they could overwrite new data or read stale data. The new implementation addressed these concerns by using monitors to prevent redundant read/writes and limit contention, and runs at an average of fifty frames per second. These frame rates were measured using instrumentation code run in the UI update function. The code is provided below.

```
...
self.counter += 1;
if self.start.elapsed() > Duration::from_secs(1) {
```

```
        println!("{}", self.counter);
        self.counter = 0;
        self.start = Instant::now();
    }
    ...
```

The triggering functionality also underwent design iteration. The initial triggering algorithm used a moving average sliding window as the noise mitigation strategy. It required that the average of the last `WINDOW_SIZE` samples be less than the currently analyzed sample, in addition to requiring that the currently analyzed sample be greater than the trigger threshold. The sliding window iterator is shown below.

```
const WINDOW_SIZE: usize = 10;
let mut sum: f64 = src[lower_idx..(lower_idx + WINDOW_SIZE)].iter().sum();
// Iterate over all windows of size WINDOW_SIZE + 1. The first WINDOW_SIZE
// samples are used to calculate the average value of previous samples,
// which is then compared to the last sample. This is done to find a
// rising edge while attempting to filter out noise.
let first_higher = src[lower_idx..]
    .windows(WINDOW_SIZE + 1)
    .position(|window| {
        let val = *window.last().unwrap();
        val >= trigger && sum / (WINDOW_SIZE as f64) < val || {
            // Update sliding window.
            sum += val;
            sum -= *window.first().unwrap();
            false
        }
    });
```

Although this algorithm functioned well, it produced slightly unstable waveforms and did not consistently trigger at the exact same position on every iteration of a signal. Furthermore, concerns were made about the validity of the algorithm and the team wanted a more conventional implementation. These issues led to the research and implementation of the current hysteresis-based triggering algorithm described in Section 3.3.2.

4.3.2 Feature Testing

Every implemented feature was rigorously tested before proceeding with new features. For example, the packet decoding functionality was tested using bytestreams redirected from the hardware to a file. Because the voltage and current values from the file were known, they could be compared to the voltage and current values after being decoded by the software algorithm. This verified that the packets were being decoded correctly and that the bytes were mapped to the right voltage/current values. Additionally, assertions were used to alert the team of any unexpected packet errors. These assertions led to the discovery of the packet encoding problem described in Section 4.2.3. They also prompted additional decoder error handling as the bytestream channel was found to not be completely reliable.

Software bugs were also discovered during testing of the frequency spectrum analysis feature. The library provided spectrum analysis function produced frequencies twice that of signals generated using a lab signal generator. This factor-of-two error was ultimately determined to be caused by an incorrect sample rate assumption in software. Although the hardware was indeed sampling at $1\,\mathrm{MSPS}$, the sample rate per channel with two channels was actually $500\,\mathrm{kSPS}$. This correction fixed both the FFT functionality and the plots' time scales.

5 Project Planning and Budgeting

The following table lists the original milestones associated with this project.

Table 3: Milestones

No.	Milestone	Due date	Responsible Member(s)
1	Design and order PCB	Week 4	Ethan
2	Create skeleton of application	Week 11	Tim
3	Submit blueprint report	Week 11	All members
4	Verify PCB is functional	Week 12	Ethan
5	Read values from USB driver	Week 13	Tim and John
6	Create basic UI and waveform display	Week 14	Eric
7	FPGA programming	Week 16	Ethan
8	Implement oscilloscope triggering	Week 16	John
9	Implement CSV exporting and all UI functionality	Week 17	Eric
10	Testing of application	Week 18	Tim, Eric, and John
11	Testing of PCB	Week 18	Ethan
12	Testing of complete system	Week 20	All members
13	Open House	Week 22	
14	Final deliverable	Week 23	
15	Final project report	Week 24	All members

Many milestones were completed earlier than planned; in particular, the full FPGA and UI work were completed between weeks 12 and 13, and basic triggering was implemented in week 15. Due to the severe software performance issues described in Section 4.3.1, developing a new design became the top priority. This slightly delayed some less important UI work, but all required functionality was completed by week 19. Furthermore, many stretch goals were achieved, including reading from four concurrent channels, advanced triggering, and frequency spectrum analysis. Also, minimal testing was required near the end of the timeline as most issues were solved during development through iterative testing.

Table 4 breaks down the cost of the PCB fabrication and assembly as quoted by the supplier. The supplier contracted for TeachEE is Bittele (7PCB). Bittele takes care of both the fabrication and soldering of the PCBs. Moreover, Bittele takes the generated output from the EDA software (Altium) and procures all the parts directly to their facility. The procurement cost is expanded in the budget table with the major functional components on the PCB. An exhaustive list is not included as there are hundreds of miscellaneous resistors, capacitors, and inductors with marginal cost.

A full unit of the PCB cost $80 less than the target budget. This left additional budget to purchase replacement

components and cover unexpected expenses. Two additional backup boards were ordered in case the main board failed. Fortunately, the team did not encounter any hardware failures during the course of the project and therefore did not require any replacing of the board or its components. The full BOM for the PCB is provided in Appendix C.

Table 4: TeachEE Bill of Materials

Item / Part Number	**Supplier**	**Cost (CAD)**
PCB Fabrication	Bittele	$104.37
PCB Total BOM Procurement	Bittele	$126.67
Major Components (Included in cost above)		
AD9288BSTZ-40 ADC	Digi-Key	$14.70
AD8138ARZ-R7 ADC Driver	Digi-Key	$26.85
FT232HQ-REEL USB FIFO	Digi-Key	$6.41
ACS720KLATR-15AB-T Current Sensor	Digi-Key	$9.81
PCB Assembly	Bittele	$289.76
Total		$520.80
Budget Remaining		$79.20

6 Stakeholder Needs

6.0.1 Features and Quality

In the online electrical engineering labs during the pandemic, an electronics professor noticed issues with the kits provided to students. There was a general lack of quality and brittleness which caused students to require replacement kits. Additionally, the oscilloscopes and signal generators had few protections against extraneous inputs, resulting in students accidentally breaking their tools when incorrectly connected. At a functional level, the oscilloscopes were missing critical features: they were single channel and had no ability to measure current or perform frequency domain analysis.

Beyond the lab kits, the archaic design of a typical oscilloscope can be improved upon. Rather than rotating physical knobs to adjust the display, it is more intuitive to interact with the application like you would any other: through the interface of a computer application. This benefits novice engineers that are unfamiliar

with traditional oscilloscopes' interfaces.

TeachEE addresses these stakeholder needs. The application is intuitive to use, with the ability to click and drag to adjust the waveform display, as well zooming via mouse scroll. It can also intuitively display the spatial and time domain of signals at the same time.

6.0.2 Safety

As was mentioned in Section 4.3, the software has been thoroughly tested to ensure stability for users. In particular, the choice of Rust in the presence of complex threading models yields a high degree of confidence that undefined behaviour is avoided. Given that TeachEE can export CSV files onto the host computer, this opens a vector of damage. If these precautions were not taken then there is the possibility that TeachEE enters an invalid state and writes to the user's computer in an undesirable way.

A computer can be damaged through the USB port by devices such as "USB Killers" [12]. Since TeachEE works with high voltage and current, it is important to ensure protection of the user's computer during operation. The computer is protected from backflow by diodes in the voltage regulator. Additionally, on the device itself, the AFE clamps voltages that are too high.

6.0.3 Manufacturing Cost

The viability of manufacturing TeachEE at scale is promising due to the current design's significant drop in unit cost at increased volume. At scales greater than 100 units, the team estimates that the cost-per-unit could drop as low as $250. This reduction in cost provides an opportunity for competitive pricing compared to the PicoScope® 4000, a USB oscilloscope with similar specifications to TeachEE. The manufacturers' price for a PicoScope® 4000 starts at $930, indicating a potential profit of up to 272% if the TeachEE were sold at scale for $930 per unit [13]. These figures suggest that TeachEE could be a viable product with the potential for a good return on investment.

7 Compliance with System Requirements

The following tables outline the target hardware and software specifications of this project. The hardware and software specification evaluations are given in Tables 5 and 6 below, and show that the prototype meets or exceeds the specifications in all areas.

Table 5: Hardware Specification Evaluation

Specification	Target Value	Achieved?	Comments
Voltage Input Bandwidth	$100\,\mathrm{kHz}$	Yes	Ultimately $500\,\mathrm{kHz}$ of bandwidth was achieved through the low speed ADC and $20\,\mathrm{MHz}$ through the high speed ADC.
Current Input Bandwidth	$100\,\mathrm{kHz}$	Yes	This bandwidth matches the hall effect sensor.
Measurable Current Range	$-15\,\mathrm{A}$ to $15\,\mathrm{A}$	Yes	Matches the specification of the sensor.
Measurable Voltage Range	$0\,\mathrm{V}$ to $3.3\,\mathrm{V}$	Yes	This was achieved in both XADC and HSADC channels.
Number of Current Input Channels	1	Yes	
Number of Voltage Input Channels	1	Yes	Two additional voltage channels were added using the HSADC.
Power Input Voltage Rating	$5\,\mathrm{V}$	Yes	USB voltage is used as supply and the regulator is tolerant up to $12\,\mathrm{V}$.
Power Current Consumption Rating	$500\,\mathrm{mA}$	Yes	Current draw has not exceeded $500\,\mathrm{mA}$ in any operation setting.
Voltage Sample Rate	$1\,\mathrm{MSPS}$	Yes	The XADC can sample at $1\,\mathrm{MSPS}$ and HSADC can sample at $40\,\mathrm{MSPS}$.
Current Sample Rate	$1\,\mathrm{MSPS}$	Yes	
PCB Thickness	$1.6\,\mathrm{mm}$	Yes	Thickness was exactly 1.6 mm in a four-layer stackup.
PCB Dimensions	$\pm 400\,\mathrm{mm}^2$	Yes	The tolerance given is less than the final size of the PCB.
Voltage Sample Error	$\pm 20\,\%$	Yes	Values match almost exactly to the Rigol scope in the Sci '65 lab through empirical testing.
Current Sample Error	$\pm 20\,\%$	Yes	Current readings match a DMM within this tolerance.

Table 6: Software Specification Evaluation

Specification	Achieved?	Comments
The software shall be able to modify the horizontal and vertical scales of the plot.	Yes	Multiple plot navigation methods, including touch screen and trackpad compatibility.
The software shall be able to modify the trigger voltage.	Yes	Triggering algorithm was exceptionally stable.
The application shall be deployable to Windows, macOS, and Linux.	Yes	
The software shall be able to capture voltage samples and export them to a CSV file.	Yes	
The software shall receive samples via the FTDI 232 in synchronous mode.	Yes	
The software shall be able to render waveforms at a rate of 30 Hz on screen.	Yes	Achieved 50 Hz with final design.

8 Conclusions & Recommendations

Developing TeachEE provided the team with several valuable technical lessons, including the value of using Rust for applications requiring high performance and concurrency. Rust's focus on memory safety and zero-cost abstractions makes it an ideal choice for building high-performance systems with low-level control. The team found that Rust enabled them to write code that was both safe and efficient, allowing them to achieve the necessary performance requirements for the TeachEE prototype. Overall, the team believes that Rust was a key factor in the success of the TeachEE project, and they plan to continue using it for future projects that require high performance and concurrency.

Another key lesson learned is to avoid assuming ideal specifications for hardware components, for example, assuming ideal speeds for USB 2, which were not achievable in practice. Such assumptions led to unexpected issues in the prototype. As a result, adjustments needed to be made to accommodate the actual capabilities of the components. This experience taught the team the importance of conducting thorough research on the specifications of hardware components and testing them under realistic conditions.

In conclusion, the team considers the TeachEE prototype to be a success and are proud to have met or

exceeded all the product requirements. If TeachEE were to be launched as a product, it could potentially create lasting impact on the field of engineering by reducing the barrier of entry for aspiring electrical engineers. TeachEE's low cost makes it more accessible to students and hobbyists who may not have access to expensive laboratory equipment (see Section 6). By providing an affordable tool for learning and experimentation, TeachEE can help to democratize access to electrical engineering education and enable a more diverse range of individuals to engage with the discipline. For these reasons, the team believes further development in products like TeachEE is worthwhile.

9 Overall Team Effort

The following table quantifies the percentage effort each team member expended on all aspects of the project.

Table 7: Team Effort Table

Name	**Effort Expended %**
Ethan Peterson	TODO
Eric Yang	TODO
Timothy Morland	TODO
John Giorshev	TODO

References

[1] S. Gallagher and J. Palmer, "The pandemic pushed universities online. the change was long overdue." *Harvard Business Review*, Sep 2020. [Online]. Available: https://hbr.org/2020/09/the-pandemic-pushed-universities-online-the-change-was-long-overdue

[2] E. Ernerfeldt, "egui," 2023. [Online]. Available: https://docs.rs/egui/latest/egui/

[3] AMBA 4 AXI4-stream protocol specification. [Online]. Available: https://developer.arm.com/documentation/ihi0051/a/Introduction/About-the-AXI4-Stream-protocol

[4] A. Forencich, "Verilog AXI stream components readme," original-date: 2014-11-06T00:17:52Z. [Online]. Available: https://github.com/alexforencich/verilog-axis

[5] Consistent overhead byte stuffing (COBS). Section: programming. [Online]. Available: https://blog.mbedded.ninja/programming/serialization-formats/consistent-overhead-byte-stuffing-cobs/

[6] AMBA AXI and ACE protocol specification version e. [Online]. Available: https://developer.arm.com/documentation/ihi0022/e/AMBA-AXI3-and-AXI4-Protocol-Specification

[7] Keysight 54600 series: What is MegaZoom and what does it do for me? - technical support knowledge center open. [Online]. Available: https://edadocs.software.keysight.com/kkbopen/keysight-54600-series-what-is-megazoom-and-what-does-it-do-for-me-588261941.html

[8] N. Instruments, "Hysteresis," 2023. [Online]. Available: https://www.ni.com/docs/en-US/bundle/ni-elvis-iii-using-instruments/page/hysteresis.html

[9] VUnit: a test framework for HDL — VUnit documentation. [Online]. Available: https://vunit.github.io/

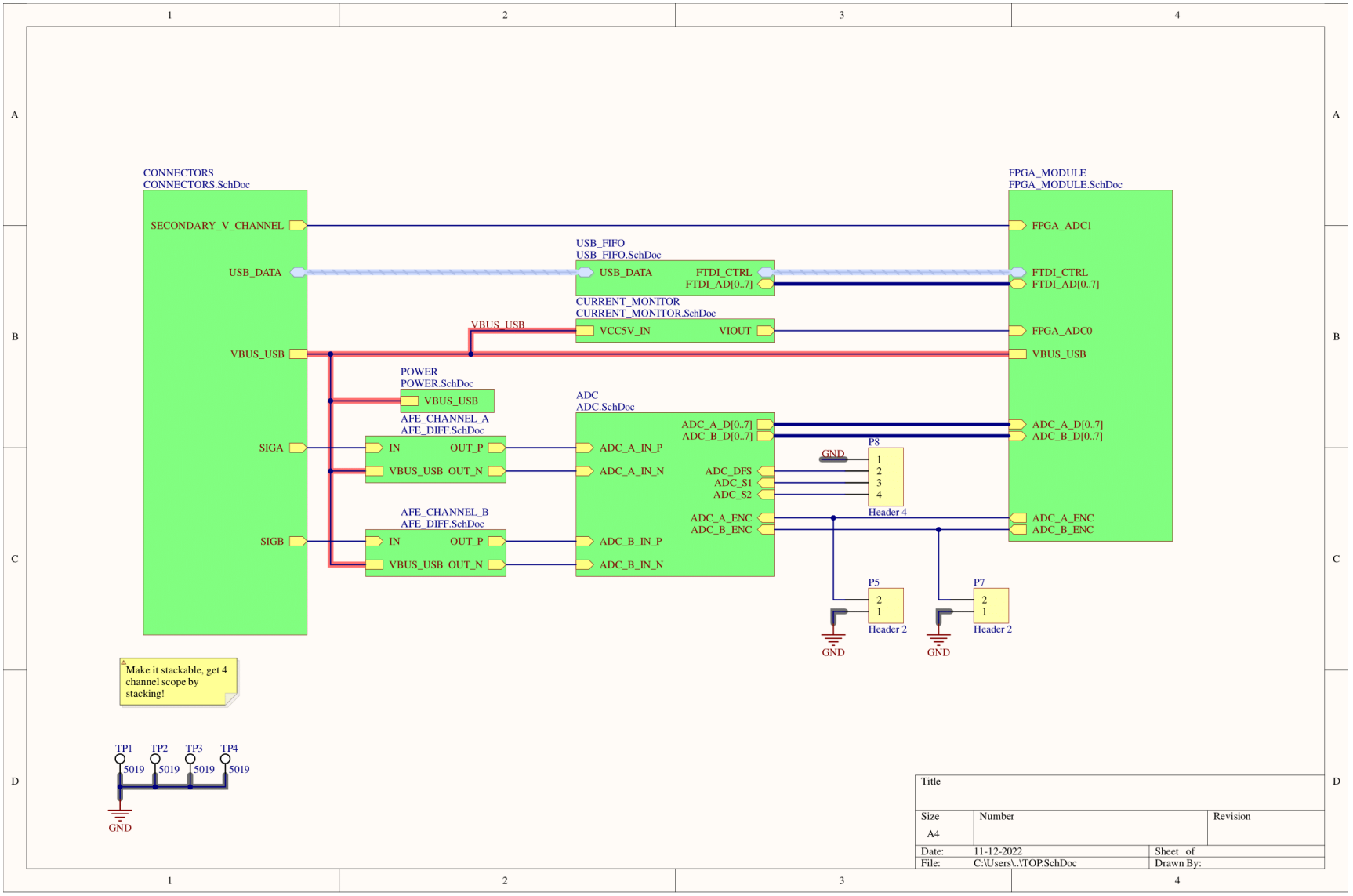
[10] E. Peterson, E. Yang, T. Morland, and J. Giorshev. teachee-capstone/teachee: Monolithic repository for our TeachEE capstone! [Online]. Available: https://github.com/teachee-capstone/teachee

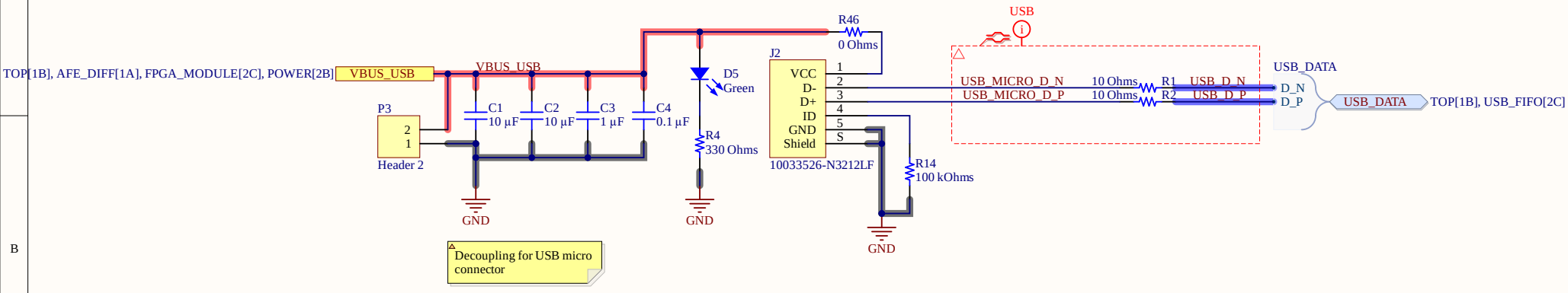
[11] Features • GitHub actions. [Online]. Available: https://github.com/features/actions

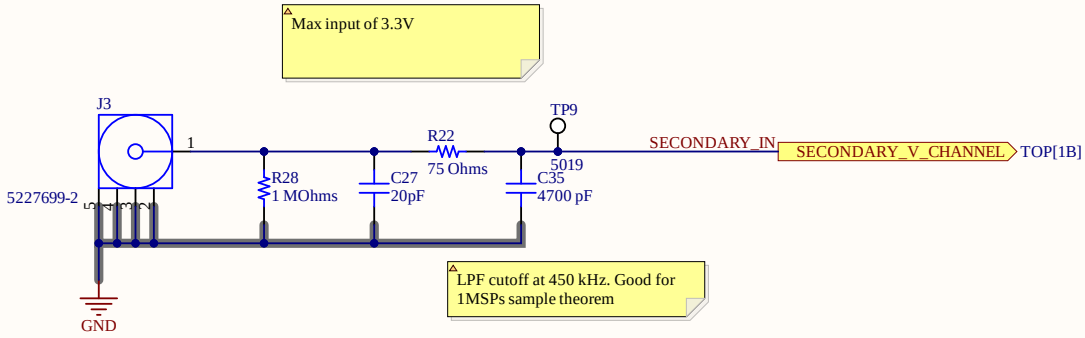
[12] L. Armasu, "Usb killer 2 shows that most usb-enabled devices are vulnerable to power surge attacks," 2023. [Online]. Available: https://www.tomshardware.com/news/usb-killer-2.0-power-surge-attack,32669.html

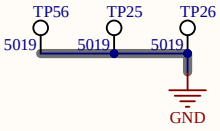
[13] "Oscilloscope | Pico Technology." [Online]. Available: https://www.picotech.com/products/oscilloscope

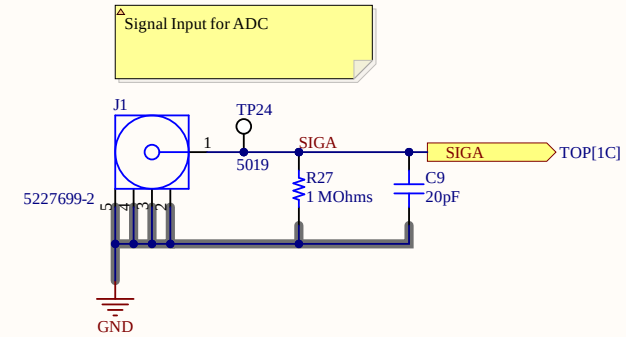
Appendix A Schematics

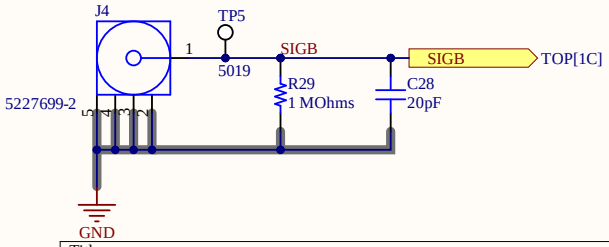




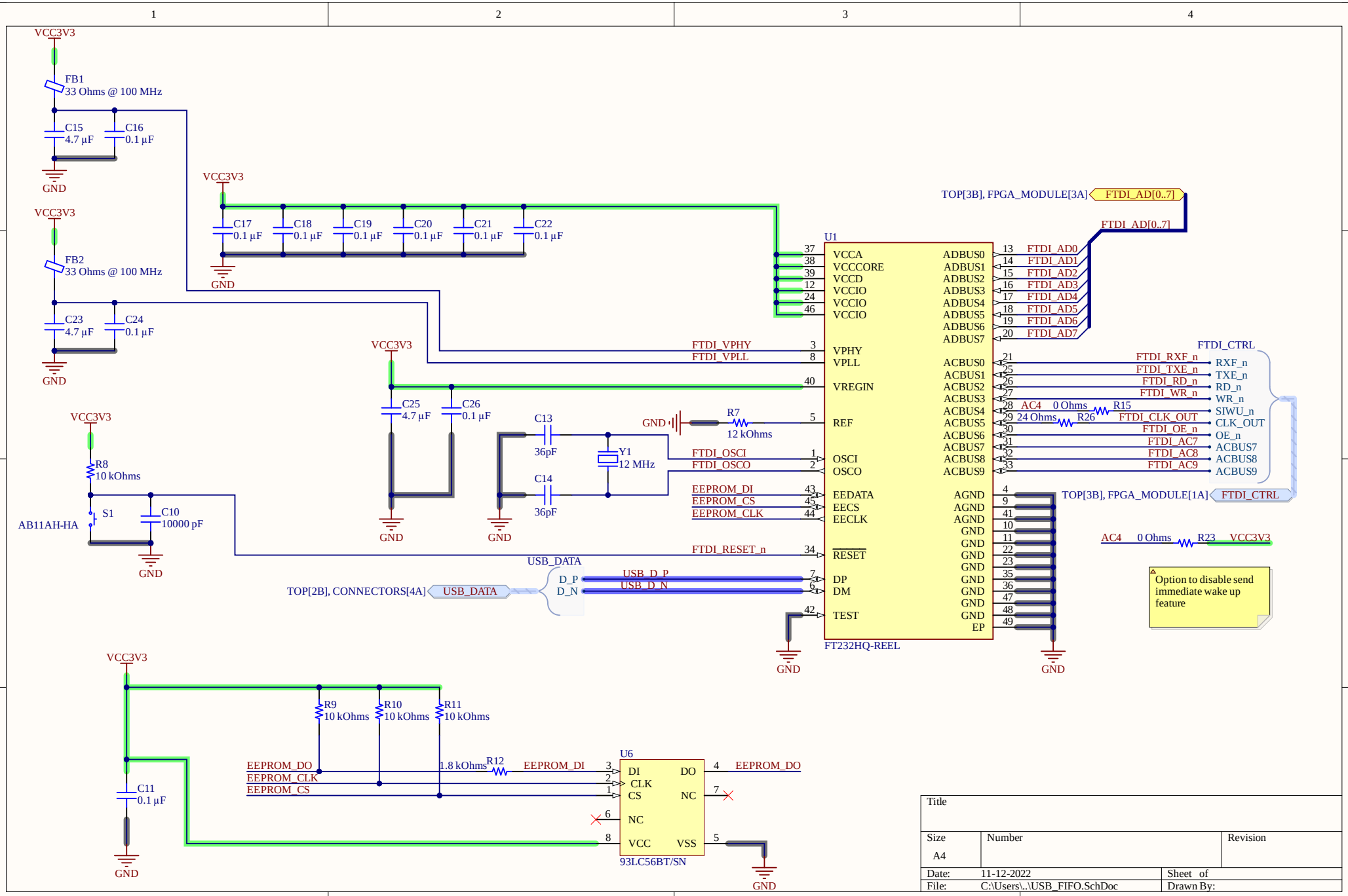


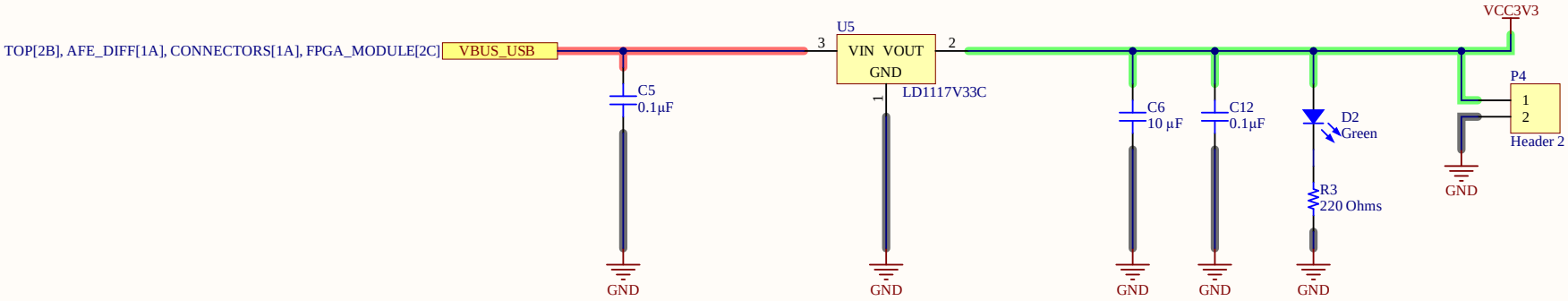




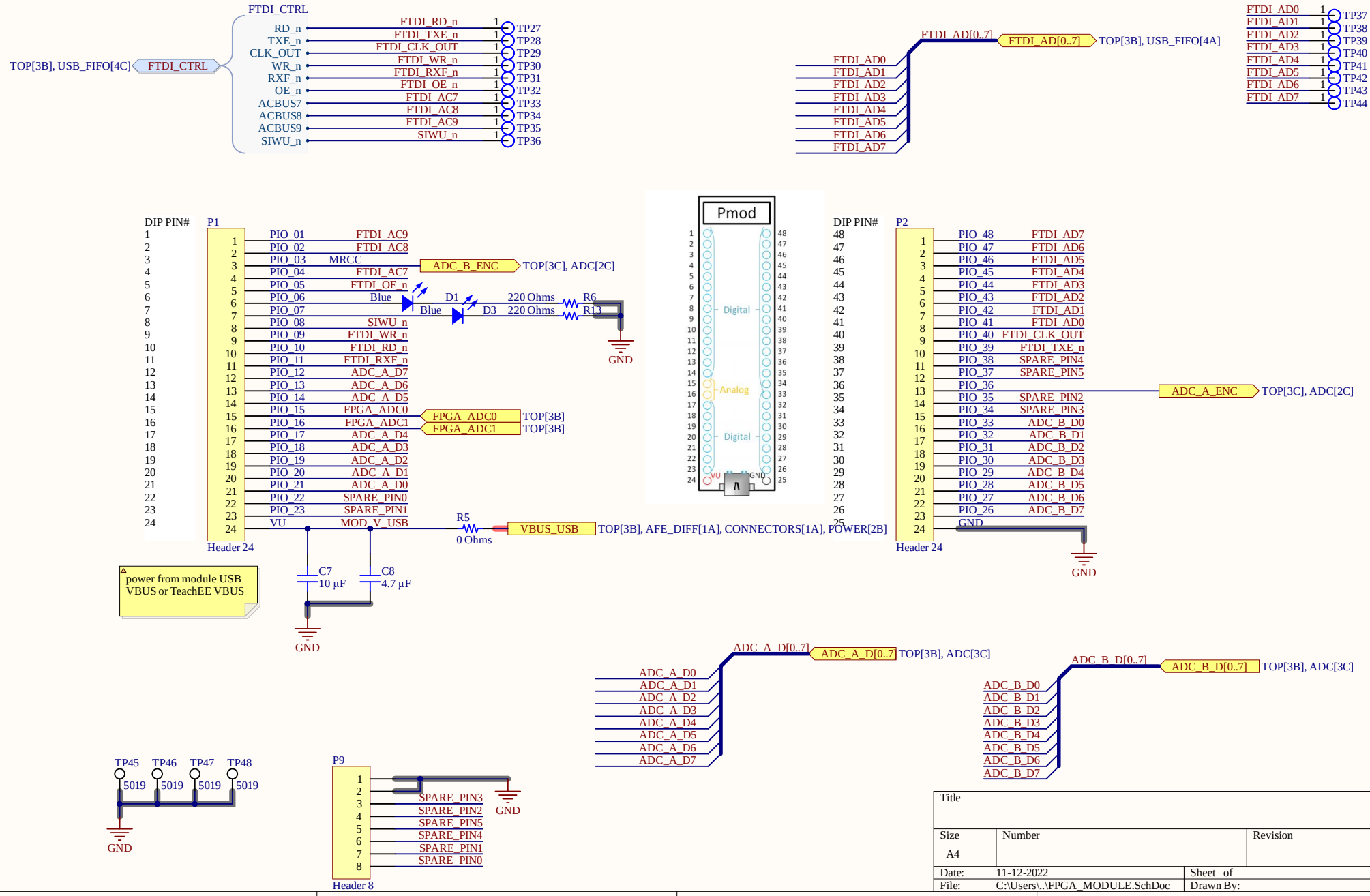


Title		
Size: A4	Number	Revision
Date: 11-12-2022		Sheet of
File: C:\Users\..\CONNECTORS.SchDoc		Drawn By:

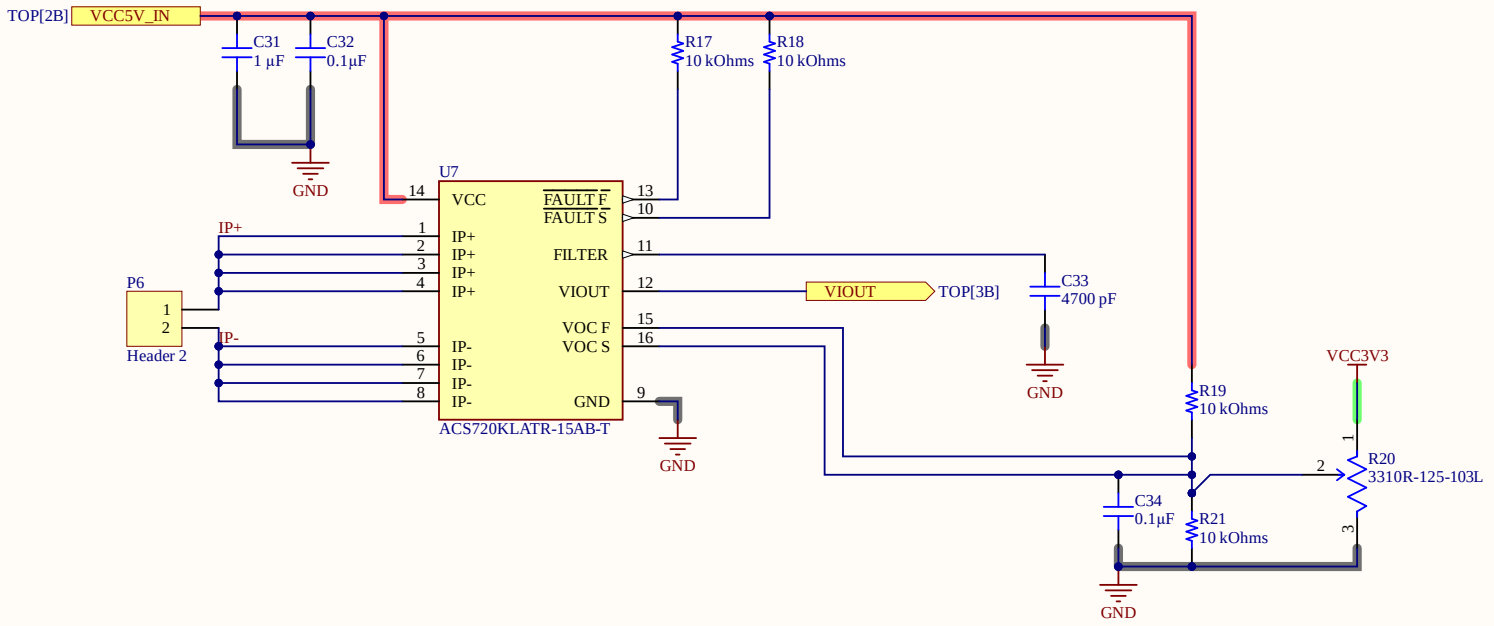

VCC3V3
FB1
33 Ohms @ 100 MHz
C15
4.7 µF
C16
0.1 µF
GND
C17
0.1 µF
C18
0.1 µF
C19
0.1 µF
C20
0.1 µF
C21
0.1 µF
C22
0.1 µF
FB2
33 Ohms @ 100 MHz
C23
4.7 µF
C24
0.1 µF
C25
4.7 µF
C26
0.1 µF
C13
36pF
C14
36pF
Y1
12 MHz
R7
12 kOhms
R8
10 kOhms
S1
AB11AH-HA
C10
10000 pF
C11
0.1 µF
R9
10 kOhms
R10
10 kOhms
R11
10 kOhms
R12
1.8 kOhms
U1
FT232HQ-REEL
VCCA
VCCCORE
VCCD
VCCIO
VCCIO
VCCIO
VPHY
VPLL
VREGIN
REF
OSCI
OSCO
EEDATA
EECS
EECLK
RESET
DP
DM
TEST
ADBUS0
ADBUS1
ADBUS2
ADBUS3
ADBUS4
ADBUS5
ADBUS6
ADBUS7
ACBUS0
ACBUS1
ACBUS2
ACBUS3
ACBUS4
ACBUS5
ACBUS6
ACBUS7
ACBUS8
ACBUS9
AGND
GND
FTDI_VPHY
FTDI_VPLL
FTDI_OSCI
FTDI_OSCO
EEPROM_DI
EEPROM_CS
EEPROM_CLK
FTDI_RESET_n
USB_DATA
USB_D_P
USB_D_N
TOP[2B], CONNECTORS[4A]
TOP[3B], FPGA_MODULE[3A]
FTDI_AD[0..7]
FTDI_AD0
FTDI_AD1
FTDI_AD2
FTDI_AD3
FTDI_AD4
FTDI_AD5
FTDI_AD6
FTDI_AD7
FTDI_CTRL
FTDI_RXF_n
FTDI_TXE_n
FTDI_RD_n
FTDI_WR_n
FTDI_CLK_OUT
FTDI_OE_n
FTDI_AC7
FTDI_AC8
FTDI_AC9
RXF_n
TXE_n
RD_n
WR_n
SIWU_n
CLK_OUT
OE_n
ACBUS7
ACBUS8
ACBUS9
AC4
0 Ohms
R15
24 Ohms
R26
TOP[3B], FPGA_MODULE[1A]
R23
VCC3V3
Option to disable send immediate wake up feature
U6
93LC56BT/SN
DI
CLK
CS
NC
VCC
DO
NC
VSS
EEPROM_DO
EEPROM_CLK
EEPROM_CS
EEPROM_DI
Title
Size
A4
Number
Revision
Date:
11-12-2022
Sheet of
File:
C:\Users\..\USB_FIFO.SchDoc
Drawn By:



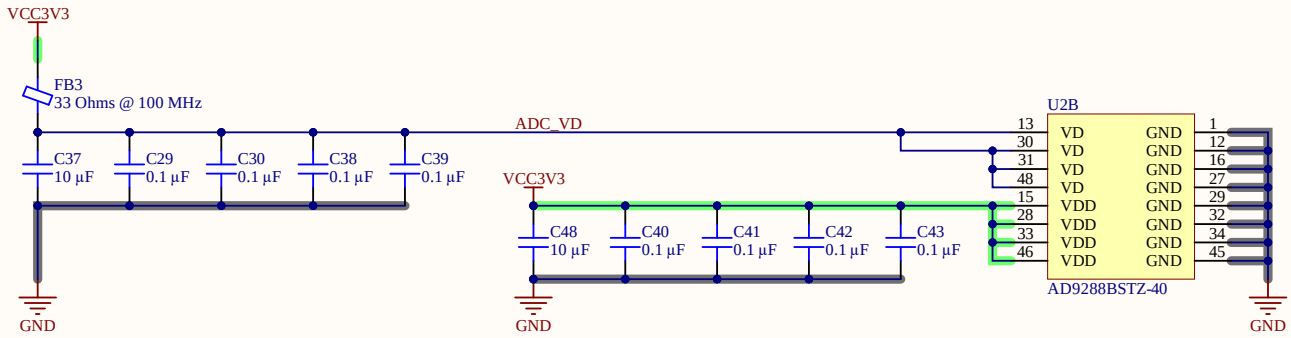
Title			
Size A4	Number		Revision
Date:	11-12-2022	Sheet of	
File:	C:\Users\..\POWER.SchDoc	Drawn By:	

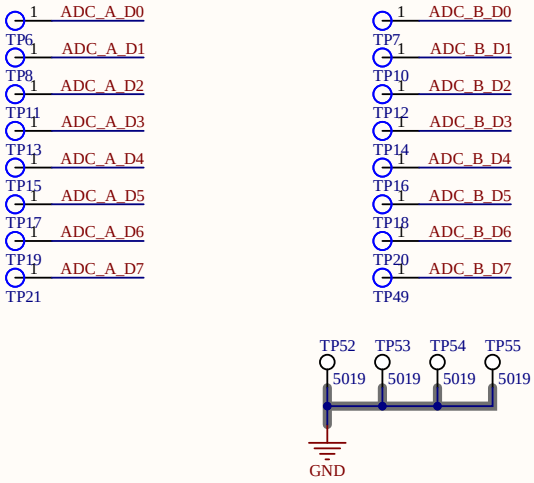

FTDI_CTRL
TOP[3B], USB_FIFO[4C]
RD_n
TXE_n
CLK_OUT
WR_n
RXF_n
OE_n
ACBUS7
ACBUS8
ACBUS9
SIWU_n
FTDI_RD_n
FTDI_TXE_n
FTDI_CLK_OUT
FTDI_WR_n
FTDI_RXF_n
FTDI_OE_n
FTDI_AC7
FTDI_AC8
FTDI_AC9
SIWU_n
TP27
TP28
TP29
TP30
TP31
TP32
TP33
TP34
TP35
TP36
FTDI_AD[0..7]
TOP[3B], USB_FIFO[4A]
FTDI_AD0
FTDI_AD1
FTDI_AD2
FTDI_AD3
FTDI_AD4
FTDI_AD5
FTDI_AD6
FTDI_AD7
TP37
TP38
TP39
TP40
TP41
TP42
TP43
TP44
DIP PIN#
P1
PIO_01
PIO_02
PIO_03
PIO_04
PIO_05
PIO_06
PIO_07
PIO_08
PIO_09
PIO_10
PIO_11
PIO_12
PIO_13
PIO_14
PIO_15
PIO_16
PIO_17
PIO_18
PIO_19
PIO_20
PIO_21
PIO_22
PIO_23
VU
MRCC
Blue
Blue
SIWU_n
FTDI_WR_n
FTDI_RD_n
FTDI_RXF_n
ADC_A_D7
ADC_A_D6
ADC_A_D5
FPGA_ADC0
FPGA_ADC1
ADC_A_D4
ADC_A_D3
ADC_A_D2
ADC_A_D1
ADC_A_D0
SPARE_PIN0
SPARE_PIN1
MOD_V_USB
ADC_B_ENC
TOP[3C], ADC[2C]
D1
D3
220 Ohms
220 Ohms
R6
R13
GND
FPGA_ADC0
TOP[3B]
FPGA_ADC1
TOP[3B]
R5
0 Ohms
VBUS_USB
TOP[3B], AFE_DIFF[1A], CONNECTORS[1A], POWER[2B]
Header 24
power from module USB
VBUS or TeachEE VBUS
C7
10 µF
C8
4.7 µF
GND
Pmod
Digital
Analog
Digital
VU
GND
DIP PIN#
P2
PIO_48
PIO_47
PIO_46
PIO_45
PIO_44
PIO_43
PIO_42
PIO_41
PIO_40
PIO_39
PIO_38
PIO_37
PIO_36
PIO_35
PIO_34
PIO_33
PIO_32
PIO_31
PIO_30
PIO_29
PIO_28
PIO_27
PIO_26
GND
FTDI_AD7
FTDI_AD6
FTDI_AD5
FTDI_AD4
FTDI_AD3
FTDI_AD2
FTDI_AD1
FTDI_AD0
FTDI_CLK_OUT
FTDI_TXE_n
SPARE_PIN4
SPARE_PIN5
SPARE_PIN2
SPARE_PIN3
ADC_B_D0
ADC_B_D1
ADC_B_D2
ADC_B_D3
ADC_B_D4
ADC_B_D5
ADC_B_D6
ADC_B_D7
ADC_A_ENC
TOP[3C], ADC[2C]
Header 24
GND
ADC_A_D[0..7]
ADC_A_D[0..7]
TOP[3B], ADC[3C]
ADC_A_D0
ADC_A_D1
ADC_A_D2
ADC_A_D3
ADC_A_D4
ADC_A_D5
ADC_A_D6
ADC_A_D7
ADC_B_D[0..7]
ADC_B_D[0..7]
TOP[3B], ADC[3C]
ADC_B_D0
ADC_B_D1
ADC_B_D2
ADC_B_D3
ADC_B_D4
ADC_B_D5
ADC_B_D6
ADC_B_D7
TP45
TP46
TP47
TP48
5019
5019
5019
5019
GND
P9
SPARE_PIN3
SPARE_PIN2
SPARE_PIN5
SPARE_PIN4
SPARE_PIN1
SPARE_PIN0
GND
Header 8
Title
Size
A4
Number
Revision
Date:
11-12-2022
Sheet of
File:
C:\Users\..\FPGA_MODULE.SchDoc
Drawn By:

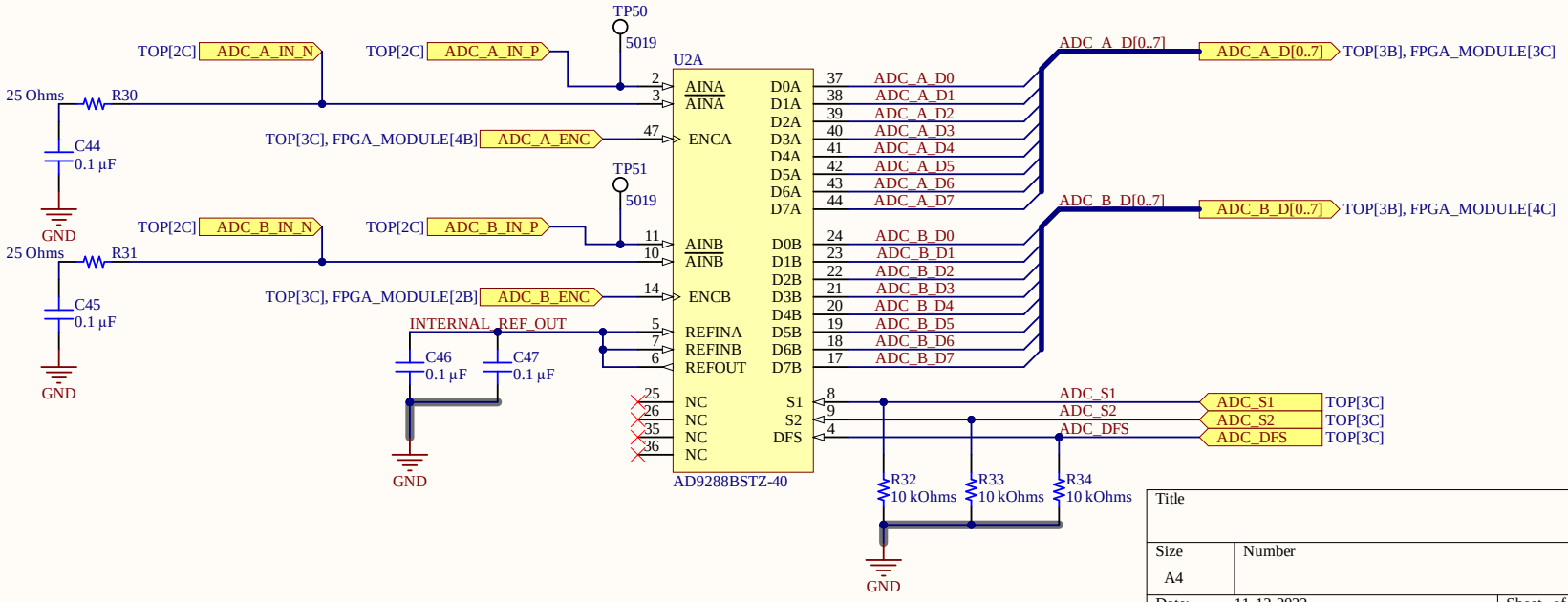
Sensor accepts 4.5 to 5.5V while USB_VBUS ranges between 4.75 and 5.25. Should be clear to power off USB bus voltage



Title		
Size: A4	Number	Revision
Date: 11-12-2022	Sheet of	
File: C:\Users\..\CURRENT_MONITOR.SchDoc	Drawn By:	

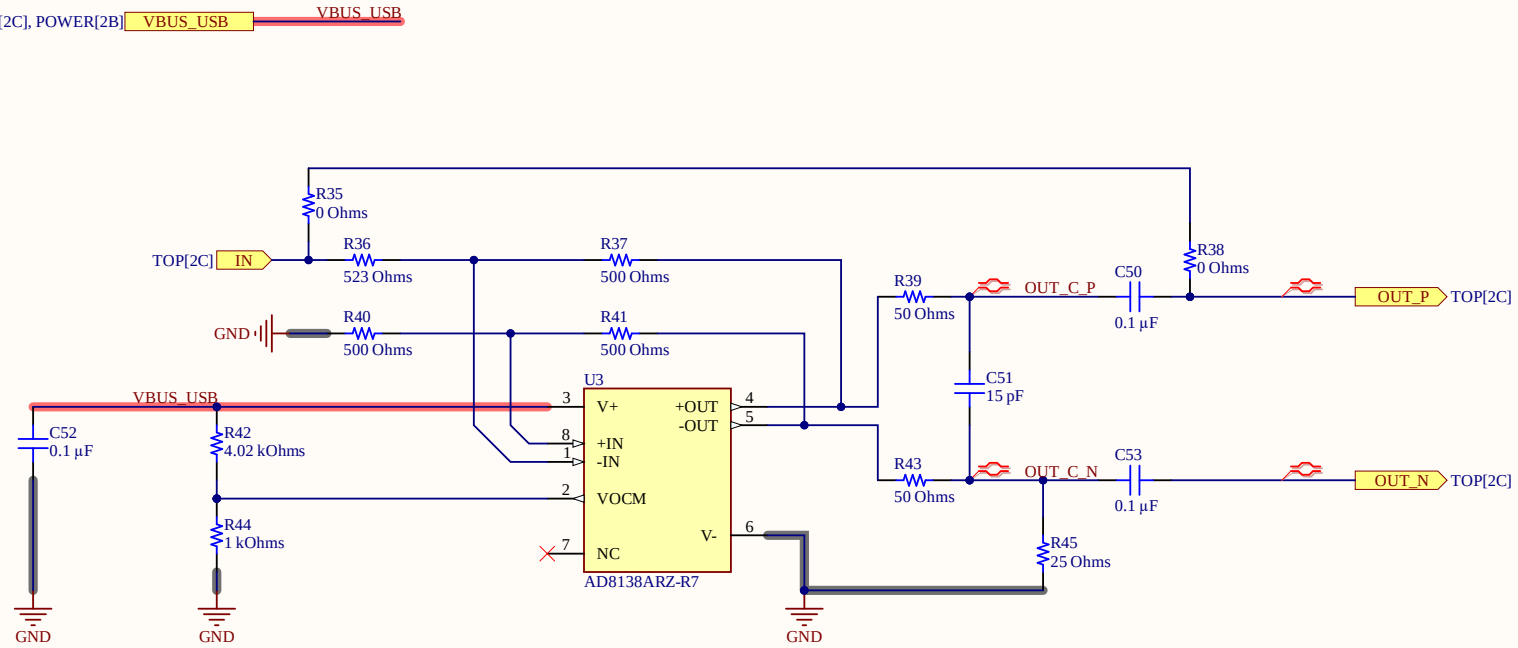






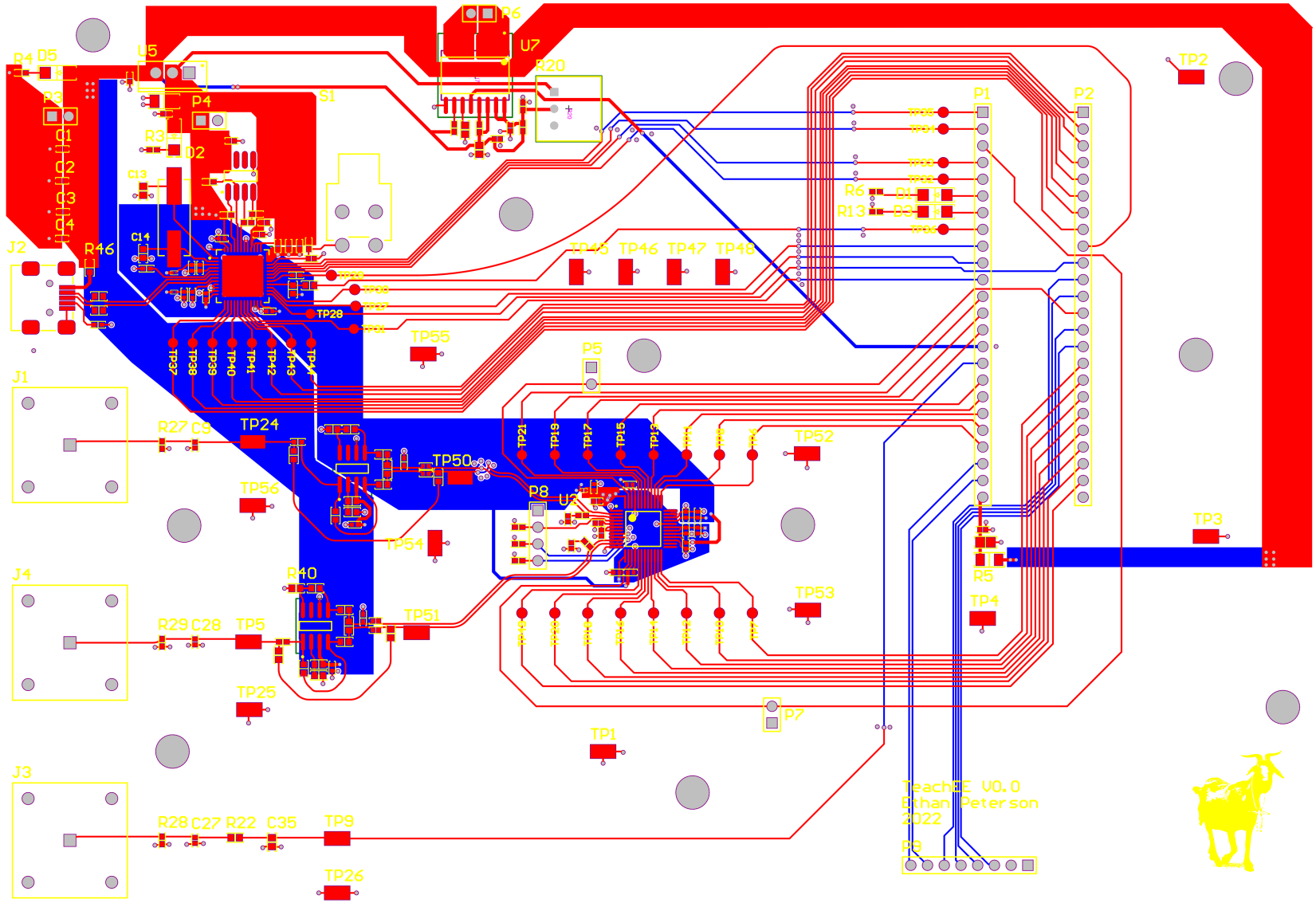
Title		
Size A4	Number	Revision
Date: 11-12-2022		Sheet of
File: C:\Users\..\ADC.SchDoc		Drawn By:

OP[2C], CONNECTORS[1A], FPGA_MODULE[2C], POWER[2B]



Title			
Size A4	Number		Revision
Date:	11-12-2022	Sheet of	
File:	C:\Users\..\AFE_DIFF.SchDoc	Drawn By:	

Appendix B PCB Layout



Appendix C PCB Bill of Materials

Designator	QTY	Description	Comment	Footprint	Part Number
C1, C2, C37, C48	4	CAP CER 10UF 10V X5R 0402	10 µF	CAP 0402_1005	CL05A106MP8NUB8
C3	1	CAP CER 1UF 10V X7S 0402	1 µF	CAP 0402_1005	GRM155C71A105KE11D
C4, C11, C16, C17, C18, C19, C20, C21, C22, C24, C26, C29, C30, C38, C39, C40, C41, C42, C43, C44, C45, C46, C47, C50_AFE_CHANNEL_A, C50_AFE_CHANNEL_B, C52_AFE_CHANNEL_A, C52_AFE_CHANNEL_B, C53_AFE_CHANNEL_A, C53_AFE_CHANNEL_B	29	CAP CER 0.1UF 50V X5R 0402	0.1 µF	CAP 0402_1005	CGA2B3X5R1H104M050BB
C5, C12, C32, C34	4	CAP CER 0.1UF 10V X7R 0402	0.1µF	CAP 0402_1005	0402ZC104KAT2A
C6	1	CAP CER 10UF 16V X5R 1206	10 µF	CAP 1206_3216 - 0.8MM	EMK316BJ106MD-T
C7	1	CAP CER 10UF 35V X6S 0805	10 µF	CAP 0805_2012	GRM21BC8YA106KE11L
C8, C15, C23, C25	4	CAP CER 4.7UF 16V X5R 0402	4.7 µF	CAP 0402_1005	CL05A475MO5NUNC
C9, C27, C28	3	CAP CER 20PF 25V NP0 0402	20pF	CAP 0402_1005	04023U200JAT2A

Designator	QTY	Description	Comment	Footprint	Part Number
C10	1	CAP MLCC 0.01UF 100V X7R 0402	10000 pF	CAP 0402_1005	HMK105B7103KVHFE
C13, C14	2	CAP CER 36PF 100V NP0 0603	36pF	CAP 0603_1608	06031A360JAT2A
C31	1	CAP CER 1UF 16V X7R 0603	1 µF	CAP 0603_1608	C0603C105K4RACAUTO
C33, C35	2	CAP CER 0603 4.7NF 16V X7R 10%	4700 pF	CAP 0603_1608	C0603C472K4RECAUTO
C51_AFE_CHANNEL_A, C51_AFE_CHANNEL_B	2	CAP CER 15PF 25V C0G/NP0 0603	15 pF	CAP 0603_1608	C0603C150J3GACAUTO
D1, D3	2	LED SMD	Blue	LED 1206_3216 BLUE	APTL3216QBC/D-01
D2, D5	2	LED SMD	Green	LED 1206_3216 GREEN	APTL3216ZGCK-01
FB1, FB2, FB3	3	FERRITE BEAD 33 OHM 0201 1LN	33 Ohms @ 100 MHz	FER 0201_0603	MMZ0603F330CT000
J1, J3, J4	3	Jack BNC Connector, 1 Position, Height 16.26 mm, Tail Length 6.35 mm, -55 to 85 degC, RoHS, Tube	5227699-2		5227699-2
J2	1	CONN RCPT MINI USB B 5POS SMD RA			10033526-N3212LF

Designator	QTY	Description	Comment	Footprint	Part Number
R1, R2	2	RES SMD 10 OHM 0.1% 1/10W 0603	10 Ohms	RES 0603_1608	CRT0603-BY-10R0ELF
R3, R6, R13	3	RES 220 OHM 1% 1/8W 0402	220 Ohms	RES 0402_1005	CRGP0402F220R
R4	1	RES SMD 330 OHM 5% 1/16W 0402	330 Ohms	RES 0402_1005	AC0402JR-07330RL
R5	1	1206 40 AMP JUMPER	0 Ohms	RES 1206_3216	JR1206X40E
R7	1	RES SMD 12K OHM 0.1% 1/16W 0402	12 kOhms	RES 0402_1005	CPF0402B12KE1
R8, R9, R10, R11, R17, R18, R19, R21, R32, R33, R34	11	RES 10K OHM 0.1% 1/10W 0402	10 kOhms	RES 0402_1005	RP73PF1E10KBTD
R12	1	RES 1.8K OHM 1% 1/16W 0402	1.8 kOhms	RES 0402_1005	RC0402FR-071K8L
R14	1	RES SMD 100K OHM 0.1% 1/16W 0402	100 kOhms	RES 0402_1005	CPF0402B100KE
R15, R46	2	RES SMD 0 OHM JUMPER 1/2W 0603	0 Ohms	RES 0603_1608	5110
R22	1	RES SMD 75 OHM 1% 1/10W 0603	75 Ohms	RES 0603_1608	AC0603FR-0775RL
R26	1	RES 24 OHM 1% 1/16W 0402	24 Ohms	RES 0402_1005	RC0402FR-0724RL

Designator	QTY	Description	Comment	Footprint	Part Number
R27, R28, R29	3	RES 1M OHM 1% 1/16W 0402	1 MOhms	RES 0402_1005	RMCF0402FT1M00
R36_AFE_CHANNEL_A, R36_AFE_CHANNEL_B	2	RES SMD 523 OHM 0.1% 1/16W 0402	523 Ohms	RES 0402_1005	ERA-2ARB5230X
R37_AFE_CHANNEL_A, R37_AFE_CHANNEL_B, R40_AFE_CHANNEL_A, R40_AFE_CHANNEL_B, R41_AFE_CHANNEL_A, R41_AFE_CHANNEL_B	6	RES SMD 500 OHM 0.05% 1/10W 0603	500 Ohms	RES 0603_1608	TNPU0603500RAZEN00
R39_AFE_CHANNEL_A, R39_AFE_CHANNEL_B, R43_AFE_CHANNEL_A, R43_AFE_CHANNEL_B	4	RES 50 OHM 5% 1/8W 0603	50 Ohms	RES 0603_1608	CH0603-50RJNTA
R42_AFE_CHANNEL_A, R42_AFE_CHANNEL_B	2	RES SMD 4.02K OHM 1% 1/10W 0603	4.02 kOhms	RES 0603_1608	AC0603FR-074K02L
R44_AFE_CHANNEL_A, R44_AFE_CHANNEL_B	2	RES SMD 1K OHM 1% 1/10W 0603	1 kOhms	RES 0603_1608	AA0603FR-071KL
R45_AFE_CHANNEL_A, R45_AFE_CHANNEL_B	2	RES 25 OHM 0.1% 1/20W 0402	25 Ohms	RES 0402_1005	FC0402E25R0BST0
S1	1	SWITCH PUSH SPST-NO 0.4VA 28V			AB11AH-HA
TP1, TP2, TP3, TP4, TP5, TP9, TP24, TP25, TP26, TP45, TP46, TP47, TP48, TP50, TP51, TP52, TP53, TP54, TP55, TP56	20	Test Point, 1 Position SMD, RoHS, Tape and Reel	5019	KSTN5019	5019
U1	1	IC HS USB TO UART/FIFO 48QFN	FT232	QFN-48	FT232HQ-REEL

Designator	QTY	Description	Comment	Footprint	Part Number
U2	1	Dual 8-Bit AD Converter with Parallel Interface, 40MSPS, -40 to +85 degC, ST-48, Pb-Free, Tray		ST-48M	AD9288BSTZ-40
U3_AFE_CHANNEL_A, U3_AFE_CHANNEL_B	2	IC ADC DRIVER 8SOIC	AD8138	R-8-IPC_A	AD8138ARZ-R7
U5	1	Fixed Low Drop Positive Voltage Regulator, 3.3V, 3-Pin TO-220	LD1117	TO220	LD1117V33C
U6	1	2K, 128x16-bit, 2.5V Microwire Serial EEPROM, 8-Pin SOIC 150mil, Commercial Temperature, Tape and Reel	93LC56	SOIC8	93LC56BT/SN
U7	1	CURRENT SENSOR	ACS720		ACS720KLATR-15AB-T
Y1	1	CRYSTAL 12.0000MHZ 20PF SMD	12 MHz	ABLS	ABLS-12.000MHZ-20-B-3-H-T

Appendix D TeachEE SystemVerilog RTL Code

This appendix contains all the SystemVerilog RTL files in the TeachEE project. Each subsection corresponds to the files in a particular folder of the GitHub repository.

D.0.1 TeachEE Main Project Top Level Module

```
`default_nettype none
`timescale 1ns / 1ps

import xadc_drp_package::*;

module teachee (
    input  var logic cmod_osc, // 12 MHz provided on the CMOD
    input  var logic ftdi_clk, // 60 MHz provided by the FTDI

    // FTDI Control Interface
    input  var logic ftdi_rxf_n,
    input  var logic ftdi_txe_n,

    output var logic ftdi_rd_n,
    output var logic ftdi_wr_n,
    output var logic ftdi_siwu_n,
    output var logic ftdi_oe_n,

    output var logic[7:0] ftdi_data,

    // High Speed ADC IO Interface
    output var logic hsadc_a_enc,
    input var logic[7:0] hsadc_a,

    output var logic hsadc_b_enc,
    input var logic[7:0] hsadc_b,

    // CMOD IO Declarations
    input var logic[1:0] btn,
    output var logic[1:0] led,

    input var logic[1:0] xa_n,
    input var logic[1:0] xa_p,

    // TeachEE IO Declarations
    output var logic[1:0] teachee_led,
    inout wire[5:0] spare_pin
);
    // Spare pin 0 = s1
    // spare pin 1 = s2
    // spare pin 4 = dfs

    var logic locked;
    var logic sys_clk;
    var logic reset;

```

```
47      assign reset = !locked;
48
49      assign sys_clk = clk_100;
50
51      var logic clk_100;
52      var logic clk_50;
53      var logic clk_25;
54      var logic clk_20;
55      var logic clk_10;
56
57      teachee_pll teachee_pll_ip_inst (
58          // Clock out ports
59          .clk_100(clk_100),     // output clk_100
60          .clk_50(clk_50),     // output clk_50
61          .clk_25(clk_25),     // output clk_25
62          .clk_20(clk_20),     // output clk_20
63          .clk_10(clk_10),     // output clk_10
64
65          // Status and control signals
66          .reset(0), // input reset
67          .locked(locked),       // output locked
68
69          // Clock in ports
70          .cmod_osc(cmod_osc)      // input cmod_osc
71      );
72
73      hsadc_interface hsadc_ctrl ();
74      assign hsadc_a_enc = hsadc_ctrl.channel_a_enc;
75      assign hsadc_ctrl.channel_a = hsadc_a;
76
77      assign hsadc_b_enc = hsadc_ctrl.channel_b_enc;
78      assign hsadc_ctrl.channel_b = hsadc_b;
79
80      assign hsadc_ctrl.s1 = spare_pin[0];
81      assign hsadc_ctrl.s2 = spare_pin[1];
82      assign hsadc_ctrl.dfs = spare_pin[4];
83
84      axis_interface #(
85          .DATA_WIDTH(2 * XADC_DRP_DATA_WIDTH)
86      ) xadc_sample_channel (
87          .clk(sys_clk),
88          .rst(reset)
89      );
90
91      axis_interface #(
92          .DATA_WIDTH(16)
93      ) hsadc_sample_channel (
94          .clk(sys_clk),
```

```
95          .rst(reset)
96      );
97
98      axis_interface #(
99          .DATA_WIDTH(8)
100     ) hsadc_usb_axis (
101         .clk(sys_clk),
102         .rst(reset)
103     );
104
105     axis_interface #(
106         .DATA_WIDTH(8)
107     ) xadc_usb_axis (
108         .clk(sys_clk),
109         .rst(reset)
110     );
111
112     ft232h usb_fifo (
113         .ftdi_clk(ftdi_clk),
114
115         .ftdi_rxf_n(ftdi_rxf_n),
116         .ftdi_txe_n(ftdi_txe_n),
117
118         .ftdi_rd_n(ftdi_rd_n),
119         .ftdi_wr_n(ftdi_wr_n),
120         .ftdi_siwu_n(ftdi_siwu_n),
121         .ftdi_oe_n(ftdi_oe_n),
122
123         .ftdi_adbus(ftdi_data),
124
125         // Programmer AXIS Interface
126         // CHANGE THIS BASED ON WHETHER YOU WANT TO USE XADC OR HSADC
127         .sys_axis(xadc_usb_axis.Sink)
128     );
129
130     xadc_drp_addr_t xadc_daddr;
131     var logic xadc_den;
132
133     var logic xadc_drdy;
134     var logic[XADC_DRP_DATA_WIDTH-1:0] xadc_do;
135     var logic xadc_eos;
136
137     xadc_drp_axis_single_stream xadc_drp_axis_adapter_inst (
138         .xadc_dclk(sys_clk),
139         .xadc_reset(reset),
140
141         // DRP and Conversion Signals
142         .xadc_daddr(xadc_daddr),
```

```
143         .xadc_den(xadc_den),
144         .xadc_drdy(xadc_drdy),
145         .xadc_do(xadc_do),
146 
147         .xadc_eos(xadc_eos),
148 
149         .sample_stream(xadc_sample_channel.Source)
150     );
151 
152     xadc_teachee xadc_teachee_inst (
153         // Clock and Reset
154         .dclk_in(sys_clk),          // input wire dclk_in
155         .reset_in(reset),        // input wire reset_in
156 
157         // DRP interface
158         .di_in(0),              // input wire [15 : 0] di_in
159         .daddr_in(xadc_daddr),        // input wire [6 : 0] daddr_in
160         .den_in(xadc_den),            // input wire den_in
161         .dwe_in(0),            // input wire dwe_in
162         .drdy_out(xadc_drdy),        // output wire drdy_out
163         .do_out(xadc_do),            // output wire [15 : 0] do_out
164 
165         // Dedicated analog input channel (we do not use this)
166         .vp_in(0),              // input wire vp_in
167         .vn_in(0),              // input wire vn_in
168 
169         // analog input channels, vaux4 is pin 15 = VIOUT of current
            ↪  sensor
170         // vaux12 is pin 16 = low speed voltage channel.
171         .vauxp4(xa_p[0]),            // input wire vauxp4
172         .vauxn4(xa_n[0]),            // input wire vauxn4
173         .vauxp12(xa_p[1]),          // input wire vauxp12
174         .vauxn12(xa_n[1]),          // input wire vauxn12
175 
176         // conversion status signals
177         .channel_out(),  // output wire [4 : 0] channel_out
178         .eoc_out(),          // output wire eoc_out
179         .alarm_out(),      // output wire alarm_out
180         .eos_out(xadc_eos),          // output wire eos_out
181         .busy_out()        // output wire busy_out
182     );
183 
184     // Configure HSADC AXI Streamer
185     hsadc_axis_wrapper hsadc (
186         .sample_clk(clk_10),
187         .stream_clk(sys_clk),
188         .reset(reset),
189 
```

```
        .hsadc_ctrl_signals(hsadc_ctrl.Sink),
        .sample_stream(hsadc_sample_channel.Source)
    );

    cobs_axis_adapter_wrapper #(
        .S_DATA_WIDTH(16),
        .M_DATA_WIDTH(8)
    ) hsadc_packetizer (
        .original_data(hsadc_sample_channel.Sink),
        .encoded_data(hsadc_usb_axis.Source)
    );

    cobs_axis_adapter_wrapper #(
        .S_DATA_WIDTH(2 * XADC_DRP_DATA_WIDTH),
        .M_DATA_WIDTH(8)
    ) xadc_packetizer (
        .original_data(xadc_sample_channel.Sink),
        .encoded_data(xadc_usb_axis.Source)
    );

    always_comb begin
        // Set one of these depending on which stream is being sent
        // xadc_sample_channel.tready = 1;
        // xadc_usb_axis.tready = 1;

        hsadc_sample_channel.tready = 1;
        hsadc_usb_axis.tready = 1;

    end

endmodule

`default_nettype wire
```

D.1 AXIS

D.1.1 axis_adapter_wrapper.sv

```
`timescale 1ns / 1ps
`default_nettype none

module axis_adapter_wrapper #(
    // Input AXIS Data width
    parameter S_DATA_WIDTH = 8,
    parameter M_DATA_WIDTH = 8,
    parameter ID_ENABLE = 1,
    parameter DEST_ENABLE = 1,
```

```
    parameter USER_ENABLE = 1
) (
    axis_interface.Sink original_data,
    axis_interface.Source modified_width_data
);

    // assumption that both streams are in the same clock domain
    axis_adapter #(
        .S_DATA_WIDTH(S_DATA_WIDTH),
        .M_DATA_WIDTH(M_DATA_WIDTH),
        .ID_ENABLE(ID_ENABLE),
        .DEST_ENABLE(DEST_ENABLE),
        .USER_ENABLE(USER_ENABLE)
        // .S_KEEP_ENABLE(0)
        // .M_KEEP_ENABLE(0)
    ) width_adapter (
        .clk(original_data.clk),
        .rst(original_data.rst || modified_width_data.rst),

        // AXI INPUT
        .s_axis_tdata(original_data.tdata),
        .s_axis_tkeep(original_data.tkeep),
        .s_axis_tvalid(original_data.tvalid),
        .s_axis_tready(original_data.tready),
        .s_axis_tlast(original_data.tlast),
        .s_axis_tid(original_data.tid),
        .s_axis_tdest(original_data.tdest),
        .s_axis_tuser(original_data.tuser),

        // AXI OUTPUT
        .m_axis_tdata(modified_width_data.tdata),
        .m_axis_tkeep(modified_width_data.tkeep),
        .m_axis_tvalid(modified_width_data.tvalid),
        .m_axis_tready(modified_width_data.tready),
        .m_axis_tlast(modified_width_data.tlast),
        .m_axis_tid(modified_width_data.tid),
        .m_axis_tdest(modified_width_data.tdest),
        .m_axis_tuser(modified_width_data.tuser)
    );
endmodule

`default_nettype wire

```

D.1.2 axis_async_fifo_wrapper.sv

```
`timescale 1ns / 1ps
`default_nettype none
```

```
3
4  module axis_async_fifo_wrapper #(
5      parameter DEPTH = 2,
6      parameter DATA_WIDTH = 8,
7      parameter USER_WIDTH = 1,
8      parameter ID_WIDTH = 8,
9      parameter DEST_WIDTH = 8,
10     parameter KEEP_WIDTH = (DATA_WIDTH+7)/8,
11     parameter KEEP_ENABLE = 0
12 ) (
13     axis_interface.Sink sink,
14     axis_interface.Source source
15 );
16
17     axis_async_fifo #(
18         .DEPTH(DEPTH),
19         .DATA_WIDTH(DATA_WIDTH),
20         .USER_WIDTH(USER_WIDTH),
21         .ID_WIDTH(ID_WIDTH),
22         .DEST_WIDTH(DEST_WIDTH),
23         .KEEP_WIDTH(KEEP_WIDTH),
24
25         // Propagate the unused signals
26         .ID_ENABLE(1),
27         .DEST_ENABLE(1),
28         .KEEP_ENABLE(KEEP_ENABLE)
29     ) tx_fifo (
30         // AXI Stream Input
31         .s_clk(sink.clk),
32         .s_rst(sink.rst),
33         .s_axis_tdata(sink.tdata),
34         .s_axis_tvalid(sink.tvalid),
35         .s_axis_tready(sink.tready),
36         .s_axis_tlast(sink.tlast),
37         .s_axis_tid(sink.tid),
38         .s_axis_tdest(sink.tdest),
39         .s_axis_tuser(sink.tuser),
40         .s_axis_tkeep(sink.tkeep),
41
42         // AXI Stream Output
43         .m_clk(source.clk),
44         .m_rst(source.rst),
45         .m_axis_tdata(source.tdata),
46         .m_axis_tvalid(source.tvalid),
47         .m_axis_tready(source.tready),
48         .m_axis_tlast(source.tlast),
49         .m_axis_tid(source.tid),
50         .m_axis_tdest(source.tdest),
```

```
        .m_axis_tuser(source.tuser),
        .m_axis_tkeep(source.tkeep)
    );



endmodule

`default_nettype wire
```

D.1.3 axis_interface.sv

```
`default_nettype none
`timescale 1ns / 1ps

interface axis_interface #(
    parameter DATA_WIDTH = 8,
    parameter USER_WIDTH = 1,
    parameter ID_WIDTH = 8,
    parameter DEST_WIDTH = 8,
    parameter KEEP_WIDTH = (DATA_WIDTH+7)/8
) (
    input var logic clk,
    input var logic rst
);

    logic[DATA_WIDTH-1:0] tdata;
    logic tvalid;
    logic tready;

    // Drive these to default values if unused.
    logic tlast;
    logic[ID_WIDTH-1:0] tid;
    logic[USER_WIDTH-1:0] tuser;
    logic[DEST_WIDTH-1:0] tdest;
    logic[KEEP_WIDTH-1:0] tkeep;

    // Default Values
    // tlast = 1
    // tkeep = 1
    // tid = 0
    // tuser = 0
    // tdest = 0


    modport Source (
        input clk, rst,
        output tdata, tkeep, tvalid, tlast, tid, tdest, tuser,
```

```
        input tready

    );

    modport Sink (
        input clk, rst,
        input tdata, tkeep, tvalid, tlast, tid, tdest, tuser,
        output tready
    );

endinterface //axis_interface

`default_nettype wire
```

D.2 COBS

D.2.1 cobs_axis_adapter_wrapper.sv

```
`timescale 1ns / 1ps
`default_nettype none

// This wrapper uses the axis width and cobs encoders wrappers to adapt
 ↪  a stream
// to a smaller size and then cobs encode it

module cobs_axis_adapter_wrapper #(
    parameter S_DATA_WIDTH = 8,
    parameter M_DATA_WIDTH = 8,
    parameter ID_ENABLE = 1,
    parameter DEST_ENABLE = 1,
    parameter USER_ENABLE = 1
) (
   axis_interface.Sink original_data,
   axis_interface.Source encoded_data
);

    // note that we will use the same clock as this is a synchronous
     ↪  module
    axis_interface #(
        .DATA_WIDTH(M_DATA_WIDTH)
    ) modified_width_data (
        .clk(original_data.clk),
        .rst(original_data.rst)
    );

    axis_adapter_wrapper #(
        .S_DATA_WIDTH(S_DATA_WIDTH),
        .M_DATA_WIDTH(M_DATA_WIDTH),
```

```
        .ID_ENABLE(ID_ENABLE),
        .DEST_ENABLE(DEST_ENABLE),
        .USER_ENABLE(USER_ENABLE)
    ) axis_width_adapter (
        .original_data(original_data),
        .modified_width_data(modified_width_data)
    );

    // Now feed the width adapter output directly into the COBS wrapper
    cobs_encode_wrapper cobs_encoder (
        .raw_stream(modified_width_data.Sink),
        .encoded_stream(encoded_data)
    );

    // ila_1 your_instance_name (
    //     .clk(original_data.clk), // input wire clk

    //     .probe0(original_data.tdata), // input wire [15:0]  probe0
    //     .probe1(original_data.tkeep), // input wire [15:0]  probe1
    //     .probe2(original_data.tdest), // input wire [15:0]  probe2
    //     .probe3(original_data.tuser), // input wire [15:0]  probe3
    //     .probe4(modified_width_data.tuser), // input wire [15:0]
     ↪  probe4
    //     .probe5(modified_width_data.tdata), // input wire [15:0]
     ↪  probe5
    //     .probe6(modified_width_data.tkeep), // input wire [15:0]
     ↪  probe6
    //     .probe7(modified_width_data.tdest), // input wire [15:0]
     ↪  probe7
    //     .probe8(original_data.tvalid), // input wire [0:0]  probe8
    //     .probe9(original_data.tready), // input wire [0:0]  probe9
    //     .probe10(original_data.tlast), // input wire [0:0]  probe10
    //     .probe11(modified_width_data.tvalid), // input wire [0:0]
     ↪  probe11
    //     .probe12(modified_width_data.tready), // input wire [0:0]
     ↪  probe12
    //     .probe13(modified_width_data.tlast), // input wire [0:0]
     ↪  probe13
    //     .probe14(0), // input wire [0:0]  probe14
    //     .probe15(0), // input wire [0:0]  probe15
    //     .probe16(0), // input wire [0:0]  probe16
    //     .probe17(0), // input wire [0:0]  probe17
    //     .probe18(0), // input wire [0:0]  probe18
    //     .probe19(0), // input wire [0:0]  probe19
    //     .probe20(0), // input wire [0:0]  probe20
    //     .probe21(0), // input wire [0:0]  probe21
    //     .probe22(0), // input wire [0:0]  probe22
    //     .probe23(0), // input wire [0:0]  probe23
```

```
    //      .probe24(0), // input wire [0:0]  probe24
    //      .probe25(0), // input wire [0:0]  probe25
    //      .probe26(0), // input wire [0:0]  probe26
    //      .probe27(0), // input wire [0:0]  probe27
    //      .probe28(0), // input wire [0:0]  probe28
    //      .probe29(0), // input wire [0:0]  probe29
    //      .probe30(0), // input wire [0:0]  probe30
    //      .probe31(0) // input wire [0:0]  probe31
    // );

endmodule

`default_nettype wire
```

D.2.2 cobs_encode_wrapper.sv

```
`default_nettype none
`timescale 1ns / 1ps
// Wrapper module to fit the cobs encoder library module to our
 ↪  interface

module cobs_encode_wrapper (
    axis_interface.Sink raw_stream, // Raw byte stream
    axis_interface.Source encoded_stream // Cobs encoded bytes
);
    axis_cobs_encode cobs_encoder (
        .clk(raw_stream.clk),
        .rst(raw_stream.rst),

        // AXI input
        .s_axis_tdata(raw_stream.tdata),
        .s_axis_tvalid(raw_stream.tvalid),
        .s_axis_tready(raw_stream.tready),
        .s_axis_tlast(raw_stream.tlast),
        .s_axis_tuser(raw_stream.tuser),

        // AXI Output
        .m_axis_tdata(encoded_stream.tdata),
        .m_axis_tvalid(encoded_stream.tvalid),
        .m_axis_tready(encoded_stream.tready),
        .m_axis_tlast(encoded_stream.tlast),
        .m_axis_tuser(encoded_stream.tuser)
    );
endmodule

`default_nettype none
```

D.3 FT232H

D.3.1 ft232h.sv

```
`default_nettype none
`timescale 1ns / 1ps

import ft232h_package::*;

module ft232h (
    input var logic ftdi_clk,

    // Control Inputs
    input var logic ftdi_rxf_n,
    input var logic ftdi_txe_n,

    // Control Outputs
    output var logic ftdi_rd_n,
    output var logic ftdi_wr_n,
    output var logic ftdi_siwu_n,
    output var logic ftdi_oe_n,

    // Data Bus
    output var logic[7:0] ftdi_adbus,

    // Programmer AXIS Interface
    axis_interface.Sink sys_axis
);

    // Define states for the device
    typedef enum int {
        INIT,
        AWAIT_USB_HOST,
        SEND_TO_USB_HOST
    } ftdi_state_t;

    ftdi_state_t state;

    // AXIS stream we will write to the FT232H
    axis_interface ftdi_axis (
        .clk(ftdi_clk),
        .rst(0)
    );
    assign ftdi_adbus = ftdi_axis.tdata;

    axis_async_fifo_wrapper #(
        .DEPTH(FT232H_AXIS_ASYNC_FIFO_DEPTH),
        .DATA_WIDTH(FT232H_AXIS_ASYNC_FIFO_DATA_WIDTH)
```

```
45      ) fpga_to_host_fifo (
46          .sink(sys_axis),
47          .source(ftdi_axis.Source)
48      );
49
50      // AXI Stream consumer state machine that sends data to FTDI
51      always_ff @(posedge ftdi_axis.clk) begin
52          // Run the control state machine here
53          case (state)
54              INIT: begin
55                  // Init signals into disabled state
56                  ftdi_rd_n <= 1;
57                  ftdi_wr_n <= 1;
58                  ftdi_siwu_n <= 1;
59                  ftdi_oe_n <= 1;
60
61                  state <= AWAIT_USB_HOST;
62              end
63              AWAIT_USB_HOST: begin
64                  if (!ftdi_txe_n && ftdi_axis.tvalid) begin
65                      ftdi_wr_n <= 0;
66                      ftdi_axis.tready <= 1;
67                      state <= SEND_TO_USB_HOST;
68                  end
69              end
70              SEND_TO_USB_HOST: begin
71                  ftdi_wr_n <= 1;
72                  ftdi_axis.tready <= 0;
73                  state <= AWAIT_USB_HOST;
74                  // If we fail the send condition. roll back to await
                    ↪  state
75                  // if (!(!ftdi_txe_n && ftdi_axis.tvalid &&
                    ↪  ftdi_axis.tready)) begin
76                  //     ftdi_wr_n <= 1;
77                  //     ftdi_axis.tready <= 0;
78                  //     state <= AWAIT_USB_HOST;
79                  // end
80              end
81              default: state <= INIT;
82          endcase
83      end
84
85  endmodule
86
87  `default_nettype wire
```

D.3.2 ft232h_bfm.sv

```
`default_nettype none
`timescale 1ns / 1ps

import ft232h_package::*;

// Tx only BFM for FT232H in FT245 Synchronous mode.
// Bit mode 0x40
module ft232h_bfm (
    output var logic clk,

    output var logic rxf_n,
    output var logic txe_n,

    input var logic rd_n,
    input var logic wr_n,
    input var logic siwu_n,
    input var logic oe_n,
    input var logic rst_n,

    // Set to input for simplicity
    // Since this is a write only BFM
    input var logic[7:0] data,

    // PC Side fake output
    output var logic[7:0] tdata,
    output var logic tvalid,
    input var logic tready
);

    initial begin
        clk = 0;
    end

    always begin
        #8.333
        clk = ~clk;
    end

    logic s_axis_tready;
    axis_fifo #(
        .DEPTH(FT232H_BFM_AXIS_FIFO_DEPTH)
    ) tx_fifo (
        // Clock and reset
        .clk(clk),
        .rst(~rst_n),

```

```
        // Input Side of FIFO
        .s_axis_tdata(data),
        .s_axis_tvalid(~wr_n),
        .s_axis_tready(s_axis_tready),

        // Output Side of FIFO
        .m_axis_tdata(tdata),
        .m_axis_tvalid(tvalid),
        .m_axis_tready(tready)
    );
    // handle inversion of ready signal to reflect ftdi chip
    assign txe_n = ~s_axis_tready;

endmodule

`default_nettype wire
```

D.3.3 ft232h_package.sv

```
`default_nettype none

package ft232h_package;
    parameter FT232H_AXIS_ASYNC_FIFO_DEPTH = 64;
    parameter FT232H_AXIS_ASYNC_FIFO_DATA_WIDTH = 8;

    parameter FT232H_BFM_AXIS_FIFO_DEPTH = 2;
endpackage

`default_nettype wire
```

D.4 HSADC

D.4.1 hsadc_axis_wrapper.sv

```
`default_nettype none
`timescale 1ns / 1ps

/*

This module will consume samples from both channels of the highspeed ADC and
output a single 16-bit AXI stream. This stream can then be compressed into a
byte stream using some of the other existing utilities in the repository for
stream manipulation.

 */

```

```
13 module hsadc_axis_wrapper (
14     input var logic sample_clk,
15     input var logic stream_clk,
16     input var logic reset,
17 
18     hsadc_interface hsadc_ctrl_signals,
19     axis_interface.Source sample_stream
20 );
21 
22     typedef enum int {
23         HSADC_INIT,
24         HSADC_COLLECT,
25         HSADC_AXIS_STORE,
26         HSADC_IDLE
27     } hsadc_axis_wrapper_state_t;
28 
29     hsadc_axis_wrapper_state_t state = HSADC_INIT;
30 
31     axis_interface #(
32         .DATA_WIDTH(16)
33     ) hsadc_axis (
34         .clk(sample_clk),
35         .rst(reset)
36     );
37 
38     axis_async_fifo_wrapper #(
39         .DEPTH(256),
40         .DATA_WIDTH(16),
41         .KEEP_ENABLE(0)
42     ) hsadc_sample_fifo (
43         .sink(hsadc_axis.Sink),
44         .source(sample_stream)
45     );
46 
47     // mirror channel A encode signal onto channel B so we can get data
     ↪  from
48     // each aligned at the same time.
49     assign hsadc_ctrl_signals.channel_b_enc =
     ↪  hsadc_ctrl_signals.channel_a_enc;
50 
51     var logic [31:0] cycle_counter = 0;
52     always_ff @(posedge sample_clk) begin
53         case (state)
54             HSADC_INIT: begin
55                 // Set the ADC into data alignment mode between channel
                   ↪  A and B
56                 hsadc_ctrl_signals.s1 <= 1;
57                 hsadc_ctrl_signals.s2 <= 0;
```

```
58
59                  // Two's comp output format
60                  hsadc_ctrl_signals.dfs <= 1;
61
62                  // Configure encode pins
63                  hsadc_ctrl_signals.channel_a_enc <= 0;
64
65                  // Start awaiting data
66                  state <= HSADC_COLLECT;
67              end
68              HSADC_COLLECT: begin
69                  // effective sampling at 1 MSPS
70                  // Only count to 8 instead of 10 since two cycles are
                        ↪  spent storing
71                  if (cycle_counter == 8) begin
72                      cycle_counter <= 0;
73                      hsadc_ctrl_signals.channel_a_enc <= 1;
74                      state <= HSADC_IDLE;
75                  end else begin
76                      cycle_counter <= cycle_counter + 1;
77                  end
78              end
79              HSADC_IDLE: begin
80                  // immediately drop the clock back low
81                  hsadc_ctrl_signals.channel_a_enc <= 0;
82
83                  // load up data to be sent into the FIFO
84                  hsadc_axis.tdata[15:8] <= hsadc_ctrl_signals.channel_a;
85                  hsadc_axis.tdata[7:0] <= hsadc_ctrl_signals.channel_b;
86                  hsadc_axis.tvalid <= 1;
87
88                  // transition to AXIS Store
89                  state <= HSADC_AXIS_STORE;
90              end
91              HSADC_AXIS_STORE: begin
92                  if (hsadc_axis.tvalid && hsadc_axis.tready) begin
93                      hsadc_axis.tvalid <= 0;
94
95                      // transition back into collection state
96                      state <= HSADC_COLLECT;
97                  end
98              end
99          endcase
100     end
101
102     // Set default values for the unused AXIS signals
103     always_comb begin
104         hsadc_axis.tlast = 1;
```

```
        hsadc_axis.tkeep = '1;
        hsadc_axis.tid = '0;
        hsadc_axis.tuser = '0;
        hsadc_axis.tdest = '0;
    end
endmodule

`default_nettype none

```

D.4.2 hsadc_bfm.sv

```
`default_nettype none
`timescale 1ns / 1ps

module hsadc_bfm (
    hsadc_interface hsadc_ctrl_signals
);
    var logic [7:0] counter = 0;
    always_ff @(posedge hsadc_ctrl_signals.channel_a_enc) begin
        hsadc_ctrl_signals.channel_a <= counter;
        hsadc_ctrl_signals.channel_b <= counter;

        counter <= counter + 1;
    end
endmodule

`default_nettype none

```

D.4.3 hsadc_interface.sv

```
`default_nettype none
`timescale 1ns / 1ps

interface hsadc_interface;
    var logic channel_a_enc;

    var logic[7:0] channel_a;

    var logic channel_b_enc;
    var logic[7:0] channel_b;

    var logic s1;
    var logic s2;
    var logic dfs;

    modport Source (
```

```
        output channel_a, channel_b,
        input channel_a_enc, channel_b_enc, s1, s2, dfs
    );

    modport Sink (
        input channel_a, channel_b,
        output channel_a_enc, channel_b_enc, s1, s2, dfs
    );

endinterface // hsadc_interface

`default_nettype wire

```

D.5 Xilinx XADC

D.5.1 xadc_bfm.sv

```
`default_nettype none
`timescale 1ns / 1ps

import xadc_drp_package::*;

// Read only BFM for the XADC in the CMOD A7 35T This module mocks the DRP and
// status signals provided by the xilinx IP wizard. It is assumed that the
// wizard was set to convert channels 4 and 12 continuously. NOTE: this BFM only
// triggers the EOS signal, it will need to be revised if single conversions are
// used.
module xadc_bfm (
    // Clock and reset
    input var logic dclk_in,
    input var logic reset_in,

    // DRP Interface
    input var logic[XADC_DRP_DATA_WIDTH-1:0] di_in, // DRP Data In
    input var logic[XADC_DRP_AXIS_ADDR_WIDTH-1:0] daddr_in, // DRP reg address in
    input var logic den_in, // DRP read enable
    input var logic dwe_in, // DRP write enable (not used in this bfm)
    output var logic drdy_out, // rising edge when data is on the output bus
    output var logic[XADC_DRP_DATA_WIDTH-1:0] do_out,

    // Dedicated Analog Input channel (not used)
```

```
25      input var logic vp_in,
26      input var logic vn_in,
27
28      // Analog input channels. In the real module we will connect these
        ↪  to the
29      // pins where the analog voltage is. However, in the BFM they can
        ↪  be left
30      // disconnected as we will generate fake conversions
31      input var logic vauxp4,
32      input var logic vauxn4,
33      input var logic vauxp12,
34      input var logic vauxn12,
35
36      // Conversion status signals
37      output var logic[4:0] channel_out,
38      output var logic eoc_out, // Indicates end of an ADC conversion
39
40      // Logical OR of all alarms in the xadc peripheral. I disable all
        ↪  of these
41      // so it should never go high.
42      output var logic alarm_out,
43      // Indicates end of a sequence of conversions. In this case, goes
        ↪  high after
44      // both channels 4 and 12 have been sampled.
45      output var logic eos_out,
46      output var logic busy_out // busy flag indicating a conversion in
        ↪  progress
47  );
48
49      typedef enum int {
50          INIT,
51          IDLE,
52          PULSE_EOS,
53          AWAIT_READ_REQUEST,
54          READ_WAIT_CYCLE, // DRP will not have data right away so we
            ↪  will waste a cycle here
55          PREP_READ_DATA, // Select the correct output data based on the
            ↪  address input
56          SHOW_READ_DATA
57      } xadc_bfm_state_t;
58
59      xadc_bfm_state_t state;
60
61      // Internal registers for fake data samples
62      var logic[XADC_DRP_DATA_WIDTH-1:0] vaux4_conv;
63      var logic[XADC_DRP_DATA_WIDTH-1:0] vaux12_conv;
64      var logic[XADC_DRP_DATA_WIDTH-1:0] conv_value; // Changes between
        ↪  vaux4 and vaux12 depending on address given.
```

```
65
66      always_ff @(posedge dclk_in) begin
67          if (reset_in) begin
68              state <= INIT;
69          end else begin
70              case (state)
71                  INIT: begin
72                      // Set initial states for outputs and then
                        ↪  transition into
73                      // operating mode
74                      drdy_out <= 0;
75                      do_out <= 16'h00_00;
76                      channel_out <= 5'b00000;
77                      eoc_out <= 0;
78                      eos_out <= 0;
79                      alarm_out <= 0;
80                      busy_out <= 0;
81
82                      // Set internal registers
83                      vaux12_conv <= 255;
84                      vaux4_conv <= 127;
85                      conv_value <= 0;
86
87                      // Proceed to next state
88                      state <= IDLE;
89                  end
90                  IDLE: begin
91                      // set eos pulse for the next cycle and transition
92                      eos_out <= 1;
93                      state <= PULSE_EOS;
94                  end
95                  PULSE_EOS: begin
96                      // Hold the EOS high for one cycle
97                      eos_out <= 0;
98                      state <= AWAIT_READ_REQUEST;
99                  end
100                 AWAIT_READ_REQUEST: begin
101                     if (den_in) begin
102                         // check the address provided and load the
                            ↪  conversion reg
103                         // TODO: These addresses should come from an
                            ↪  include
104                         if (daddr_in == XADC_DRP_ADDR_CURRENT_CHANNEL)
                            ↪  begin
105                             // if we are reading the current sensor
106                             conv_value <= vaux4_conv;
107                         end else if (daddr_in ==
                            ↪  XADC_DRP_ADDR_VOLTAGE_CHANNEL) begin
```

```
                            conv_value <= vaux12_conv;
                        end else begin
                            $display("Unknown Address provided");
                        end
                        state <= READ_WAIT_CYCLE;
                    end
                end
                READ_WAIT_CYCLE: state <= PREP_READ_DATA;
                PREP_READ_DATA: begin
                    // Set ready and data_output signals
                    drdy_out <= 1;
                    do_out <= conv_value;
                    state <= SHOW_READ_DATA;
                end
                SHOW_READ_DATA: begin
                    drdy_out <= 0;
                    do_out <= 0;
                    state <= IDLE;
                end
            endcase
        end
    end

endmodule

`default_nettype wire
```

D.5.2 xadc_drp_axis_adapter.sv

```
`default_nettype none
`timescale 1ns / 1ps

import xadc_drp_package::*;

module xadc_drp_axis_adapter (
    // Xilinx XADC IP Interface (Only putting through the required
    ↪  signals)
    input var logic xadc_dclk,
    input var logic xadc_reset,

    // DRP Interface and Conversion Signals
    output xadc_drp_addr_t xadc_daddr,
    output var logic xadc_den,
    input var logic xadc_drdy,
    input var logic[XADC_DRP_DATA_WIDTH-1:0] xadc_do,
    input var logic xadc_eos,

    // ADC Channel AXI Streams
```

```
    axis_interface.Source current_monitor_channel,
    axis_interface.Source voltage_channel
);

    typedef enum int {
        XADC_AXIS_INIT,
        XADC_AXIS_AWAIT_EOS,
        XADC_AXIS_START_DRP_VOLTAGE_READ,
        XADC_AXIS_AWAIT_VOLTAGE_DATA,
        XADC_AXIS_SEND_TO_VOLTAGE_FIFO,
        XADC_AXIS_START_DRP_CURRENT_READ,
        XADC_AXIS_AWAIT_CURRENT_DATA,
        XADC_AXIS_SEND_TO_CURRENT_FIFO
    } xadc_drp_axis_adapter_state_t;

    xadc_drp_axis_adapter_state_t state = XADC_AXIS_INIT;

    // Define AXI Stream Interfaces
    // These interfaces will be tagged as sinks into the FIFO
    axis_interface #(
        .DATA_WIDTH(XADC_DRP_DATA_WIDTH)
    ) xadc_current_axis (
        .clk(xadc_dclk),
        .rst(xadc_reset)
    );

    axis_interface #(
        .DATA_WIDTH(XADC_DRP_DATA_WIDTH)
    ) xadc_voltage_axis (
        .clk(xadc_dclk),
        .rst(xadc_reset)
    );

    // Create Async FIFOs
    axis_async_fifo_wrapper #(
        .DEPTH(XADC_DRP_AXIS_FIFO_DEPTH),
        .DATA_WIDTH(XADC_DRP_DATA_WIDTH),
        .KEEP_ENABLE(0)
    ) xadc_current_fifo (
        .sink(xadc_current_axis.Sink),
        .source(current_monitor_channel)
    );

    axis_async_fifo_wrapper #(
        .DEPTH(XADC_DRP_AXIS_FIFO_DEPTH),
        .DATA_WIDTH(XADC_DRP_DATA_WIDTH),
        .KEEP_ENABLE(0)
    ) xadc_voltage_fifo (
```

```
67          .sink(xadc_voltage_axis.Sink),
68          .source(voltage_channel)
69      );
70
71      always_ff @(posedge xadc_dclk) begin
72          case (state)
73              XADC_AXIS_INIT: begin
74                  // Init both AXIS Sinks
75                  xadc_current_axis.tdata <= 0;
76                  xadc_current_axis.tvalid <= 0;
77
78                  xadc_voltage_axis.tdata <= 0;
79                  xadc_voltage_axis.tvalid <= 0;
80
81                  xadc_daddr <= XADC_DRP_ADDR_VOLTAGE_CHANNEL;
82                  xadc_den <= 0;
83
84                  state <= XADC_AXIS_AWAIT_EOS;
85              end
86              XADC_AXIS_AWAIT_EOS: begin
87                  if (xadc_eos) begin
88                      xadc_daddr <= XADC_DRP_ADDR_VOLTAGE_CHANNEL;
89                      xadc_den <= 1;
90
91                      state <= XADC_AXIS_START_DRP_VOLTAGE_READ;
92                  end
93              end
94              XADC_AXIS_START_DRP_VOLTAGE_READ: begin
95                  xadc_den <= 0;
96
97                  state <= XADC_AXIS_AWAIT_VOLTAGE_DATA;
98              end
99              XADC_AXIS_AWAIT_VOLTAGE_DATA: begin
100                 if (xadc_drdy) begin
101                     xadc_voltage_axis.tdata <= xadc_do;
102                     xadc_voltage_axis.tvalid <= 1;
103
104                     state <= XADC_AXIS_SEND_TO_VOLTAGE_FIFO;
105                 end
106             end
107             XADC_AXIS_SEND_TO_VOLTAGE_FIFO: begin
108                 if (xadc_voltage_axis.tready &&
                    ↪  xadc_voltage_axis.tvalid) begin
109                     xadc_voltage_axis.tvalid <= 0;
110                     state <= XADC_AXIS_START_DRP_CURRENT_READ;
111                 end
112             end
113             XADC_AXIS_START_DRP_CURRENT_READ: begin
```

```
                xadc_daddr <= XADC_DRP_ADDR_CURRENT_CHANNEL;
                xadc_den <= 1;

                state <= XADC_AXIS_AWAIT_CURRENT_DATA;
            end
            XADC_AXIS_AWAIT_CURRENT_DATA: begin
                xadc_den <= 0;
                if (xadc_drdy) begin
                    xadc_current_axis.tdata <= xadc_do;
                    xadc_current_axis.tvalid <= 1;

                    state <= XADC_AXIS_SEND_TO_CURRENT_FIFO;
                end
            end
            XADC_AXIS_SEND_TO_CURRENT_FIFO: begin
                if (xadc_current_axis.tready &&
                ↪  xadc_current_axis.tvalid) begin
                    xadc_current_axis.tvalid <= 0;

                    state <= XADC_AXIS_AWAIT_EOS;
                end
            end
        endcase
    end

    // Set default values for the unused AXIS signals
    always_comb begin
        xadc_current_axis.tlast = 1;
        xadc_current_axis.tkeep = '1;
        xadc_current_axis.tid = '0;
        xadc_current_axis.tuser = '0;
        xadc_current_axis.tdest = '0;

        xadc_voltage_axis.tlast = 1;
        xadc_voltage_axis.tkeep = '1;
        xadc_voltage_axis.tid = '0;
        xadc_voltage_axis.tuser = '0;
        xadc_voltage_axis.tdest = '0;
    end
endmodule

`default_nettype wire
```

D.5.3 xadc_drp_axis_single_stream.sv

```
`default_nettype none
`timescale 1ns / 1ps

```

```
4  import xadc_drp_package::*;
5
6  module xadc_drp_axis_single_stream (
7      // Xilinx XADC IP Interface (Only putting through the required
       ↪  signals)
8      input var logic xadc_dclk,
9      input var logic xadc_reset,
10
11     // DRP Interface and Conversion Signals
12     output xadc_drp_addr_t xadc_daddr,
13     output var logic xadc_den,
14     input var logic xadc_drdy,
15     input var logic[XADC_DRP_DATA_WIDTH-1:0] xadc_do,
16     input var logic xadc_eos,
17
18     // ADC Channel AXI Streams
19     axis_interface.Source sample_stream
20 );
21
22     typedef enum int {
23         XADC_AXIS_INIT,
24         XADC_AXIS_AWAIT_EOS,
25         XADC_AXIS_START_DRP_VOLTAGE_READ,
26         XADC_AXIS_AWAIT_VOLTAGE_DATA,
27         XADC_AXIS_STORE_VOLTAGE_DATA,
28         XADC_AXIS_START_DRP_CURRENT_READ,
29         XADC_AXIS_AWAIT_CURRENT_DATA,
30         XADC_AXIS_SEND_TO_SAMPLE_FIFO
31     } xadc_drp_axis_adapter_state_t;
32
33     xadc_drp_axis_adapter_state_t state = XADC_AXIS_INIT;
34
35     // Define AXI Stream Interfaces
36     // These interfaces will be tagged as sinks into the FIFO
37     axis_interface #(
38         .DATA_WIDTH(2 * XADC_DRP_DATA_WIDTH)
39     ) xadc_sample_data_axis (
40         .clk(xadc_dclk),
41         .rst(xadc_reset)
42     );
43
44     // Create Async FIFOs
45     axis_async_fifo_wrapper #(
46         .DEPTH(XADC_DRP_AXIS_FIFO_DEPTH),
47         .DATA_WIDTH(2 * XADC_DRP_DATA_WIDTH),
48         .KEEP_ENABLE(0)
49     ) xadc_current_fifo (
50         .sink(xadc_sample_data_axis.Sink),
```

```
51          .source(sample_stream)
52      );
53
54      var logic [XADC_DRP_DATA_WIDTH-1:0] voltage_sample;
55      always_ff @(posedge xadc_dclk) begin
56          case (state)
57              XADC_AXIS_INIT: begin
58                  // Init both AXIS Sinks
59                  xadc_sample_data_axis.tdata <= 0;
60                  xadc_sample_data_axis.tvalid <= 0;
61
62                  xadc_daddr <= XADC_DRP_ADDR_VOLTAGE_CHANNEL;
63                  xadc_den <= 0;
64
65                  state <= XADC_AXIS_AWAIT_EOS;
66              end
67              XADC_AXIS_AWAIT_EOS: begin
68                  if (xadc_eos) begin
69                      xadc_daddr <= XADC_DRP_ADDR_VOLTAGE_CHANNEL;
70                      xadc_den <= 1;
71
72                      state <= XADC_AXIS_START_DRP_VOLTAGE_READ;
73                  end
74              end
75              XADC_AXIS_START_DRP_VOLTAGE_READ: begin
76                  xadc_den <= 0;
77
78                  state <= XADC_AXIS_AWAIT_VOLTAGE_DATA;
79              end
80              XADC_AXIS_AWAIT_VOLTAGE_DATA: begin
81                  if (xadc_drdy) begin
82                      voltage_sample <= xadc_do;
83
84                      state <= XADC_AXIS_START_DRP_CURRENT_READ;
85                  end
86              end
87              XADC_AXIS_START_DRP_CURRENT_READ: begin
88                  xadc_daddr <= XADC_DRP_ADDR_CURRENT_CHANNEL;
89                  xadc_den <= 1;
90
91                  state <= XADC_AXIS_AWAIT_CURRENT_DATA;
92              end
93              XADC_AXIS_AWAIT_CURRENT_DATA: begin
94                  xadc_den <= 0;
95                  if (xadc_drdy) begin
96
                        ↪  xadc_sample_data_axis.tdata[XADC_DRP_DATA_WIDTH-1:0]
                        ↪  <= xadc_do;
```

```
                    xadc_sample_data_axis.tdata[2*XADC_DRP_DATA_WIDTH -
                    ↪  1:XADC_DRP_DATA_WIDTH] <= voltage_sample;
                    xadc_sample_data_axis.tvalid <= 1;

                    state <= XADC_AXIS_SEND_TO_SAMPLE_FIFO;
                end
            end
            XADC_AXIS_SEND_TO_SAMPLE_FIFO: begin
                if (xadc_sample_data_axis.tready &&
                ↪  xadc_sample_data_axis.tvalid) begin
                    xadc_sample_data_axis.tvalid <= 0;

                    state <= XADC_AXIS_AWAIT_EOS;
                end
            end
        endcase
    end

    // Set default values for the unused AXIS signals
    always_comb begin
        xadc_sample_data_axis.tlast = 1;
        xadc_sample_data_axis.tkeep = '1;
        xadc_sample_data_axis.tid = '0;
        xadc_sample_data_axis.tuser = '0;
        xadc_sample_data_axis.tdest = '0;
    end
endmodule

`default_nettype wire
```

D.5.4 xadc_drp_package.sv

```
`default_nettype none

package xadc_drp_package;
    parameter XADC_DRP_DATA_WIDTH = 16;
    parameter XADC_DRP_AXIS_FIFO_DEPTH = 128;
    parameter XADC_DRP_AXIS_ADDR_WIDTH = 7;

    parameter XADC_DRP_SAMPLE_MSB_IDX = 15;
    parameter XADC_DRP_SAMPLE_LSB_IDX = 4;

    typedef enum logic[XADC_DRP_AXIS_ADDR_WIDTH-1:0] {
        XADC_DRP_ADDR_CURRENT_CHANNEL = 7'h14,
        XADC_DRP_ADDR_VOLTAGE_CHANNEL = 7'h1c
    } xadc_drp_addr_t;
endpackage

```

```
`default_nettype wire

```

D.5.5 xadc_packetizer.sv

```
`default_nettype none
`timescale 1ns / 1ps

import xadc_drp_package::*;

// This module will consume data from the XADC Channel FIFOs. The module will
// convert the two 16 bit streams into a single 8 bit stream. The 8 bit stream
// will contain header bytes to indicate what channel the sample is coming from.
// This stream will then be fed into a COBS encoder module which will feed the
// 8-bit USB fifo.

module xadc_packetizer (
    // NOTE: this module expects that both input streams share a clock since
    // they come from the same XADC
    axis_interface.Sink voltage_channel,
    axis_interface.Sink current_monitor_channel,

    // Packet stream that can be sent to the encoder (COBS in this case)
    axis_interface.Source packet_stream
);

    // shared clock abbreviated for simplicity
    var logic clk;
    assign clk = voltage_channel.clk;

    typedef enum int {
        XADC_PACKETIZER_INIT,
        XADC_PACKETIZER_LOAD_NEW_SAMPLES,
        XADC_PACKETIZER_SEND_VOLTAGE_UPPER,
        XADC_PACKETIZER_SEND_VOLTAGE_LOWER,
        XADC_PACKETIZER_SEND_CURRENT_UPPER,
        XADC_PACKETIZER_SEND_CURRENT_LOWER
    } xadc_packetizer_state_t;

    xadc_packetizer_state_t state = XADC_PACKETIZER_INIT;

    axis_interface #(
```

```
38          .DATA_WIDTH(8)
39      ) raw_stream (
40          .clk(clk),
41          .rst(voltage_channel.rst || current_monitor_channel.rst)
42      );
43
44      cobs_encode_wrapper cobs_encoder (
45          .raw_stream(raw_stream.Sink),
46          .encoded_stream(packet_stream)
47      );
48
49      // Declare upper and lower byte for voltage / current
50      var logic[7:0] voltage_upper;
51      var logic[7:0] voltage_lower;
52
53      var logic[7:0] current_upper;
54      var logic[7:0] current_lower;
55
56      always_ff @(posedge clk) begin
57          case (state)
58              XADC_PACKETIZER_INIT: begin
59                  // init raw_stream signals to default states
60                  raw_stream.tlast <= 0;
61                  raw_stream.tkeep <= 1;
62                  raw_stream.tid <= 0;
63                  raw_stream.tuser <= 0;
64                  raw_stream.tdest <= 0;
65
66                  voltage_channel.tready <= 0;
67                  current_monitor_channel.tready <= 0;
68
69                  raw_stream.tvalid <= 0;
70
71                  // Tell the input streams we are ready to intake bytes
72
73                  if (voltage_channel.tvalid &&
                    ↪  current_monitor_channel.tvalid) begin
74                      // Wait until we have valid data before starting a
                        ↪  load
75                      voltage_channel.tready <= 1;
76                      current_monitor_channel.tready <= 1;
77
78                      state <= XADC_PACKETIZER_LOAD_NEW_SAMPLES;
79                  end
80              end
81              XADC_PACKETIZER_LOAD_NEW_SAMPLES: begin
82                  // Wait until there is both voltage and current data
                    ↪  available
```

```
83                  // Then build a packet
84                  if (voltage_channel.tready && voltage_channel.tvalid &&
                     ↪  current_monitor_channel.tready &&
                     ↪  current_monitor_channel.tvalid) begin
85                      // load the registers with the sample data
86                      voltage_upper <=
                         ↪  voltage_channel.tdata[XADC_DRP_SAMPLE_MSB_IDX:8];
87                      voltage_lower <=
                         ↪  voltage_channel.tdata[7:XADC_DRP_SAMPLE_LSB_IDX];
88
89                      current_upper <=
                         ↪  current_monitor_channel.tdata[XADC_DRP_SAMPLE_MSB_IDX:8];
90                      current_lower <=
                         ↪  current_monitor_channel.tdata[7:XADC_DRP_SAMPLE_LSB_IDX];
91
92                      // The samples are only 12 bits so we are going to
                         ↪  abuse the
93                      // upper 4 bits to serve as packet header
94
95                      // load the packet header into voltager_upper
96                      // voltage_upper[15:12] <=
                         ↪  XADC_PACKET_HEADER_LOW_SPEED_SAMPLE;
97
98                      // Disable the tready signal and proceed to the
                         ↪  next state
99                      voltage_channel.tready <= 0;
100                     current_monitor_channel.tready <= 0;
101
102                     // enable tvalid to stream the data into raw_stream
103                     raw_stream.tvalid <= 1;
104
105                     // Note that we can't just use voltage_upper yet
                         ↪  due to async assign
106                     // Note this includes the header
107                     raw_stream.tdata <=
                         ↪  voltage_channel.tdata[XADC_DRP_SAMPLE_MSB_IDX:8];
108
109                     state <= XADC_PACKETIZER_SEND_VOLTAGE_UPPER;
110                 end
111             end
112             XADC_PACKETIZER_SEND_VOLTAGE_UPPER: begin
113                 if (raw_stream.tvalid && raw_stream.tready) begin
114                     // await tready to ensure the voltage upper is
                         ↪  written and
115                     // proceed to the next state
116
117                     raw_stream.tdata <= voltage_lower;
118                     state <= XADC_PACKETIZER_SEND_VOLTAGE_LOWER;
```

```
                end
            end
            XADC_PACKETIZER_SEND_VOLTAGE_LOWER: begin
                if (raw_stream.tvalid && raw_stream.tready) begin
                    // after sending lower voltage, transition and send
                     ↪  current data
                    raw_stream.tdata <= current_upper;
                    state <= XADC_PACKETIZER_SEND_CURRENT_UPPER;
                end
            end
            XADC_PACKETIZER_SEND_CURRENT_UPPER: begin
                // NEED TO HANDLE UPPER CURRENT CASE HERE
                if (raw_stream.tvalid && raw_stream.tready) begin
                    // load stream data with current lower byte and
                     ↪  transition out of the state
                    raw_stream.tdata <= current_lower;

                    // flag to cobs encoder that this is the last byte
                     ↪  of the packet
                    raw_stream.tlast <= 1;
                    state <= XADC_PACKETIZER_SEND_CURRENT_LOWER;
                end
            end
            XADC_PACKETIZER_SEND_CURRENT_LOWER: begin
                raw_stream.tvalid <= 0;

                // indicate that we are no longer ready for data from
                 ↪  the current and voltage stream
                voltage_channel.tready <= 0;
                current_monitor_channel.tready <= 0;
                // raw_stream.tlast <= 0;

                state <= XADC_PACKETIZER_INIT;
            end
        endcase
    end
endmodule

`default_nettype none
```

D.6 Legacy Non-VUnit Testbenches

D.6.1 ft232h_bfm_tb.sv

```
`default_nettype none
`timescale 1ns / 1ps

// Test bench for the TX only BFM
```

```
5  // timed_tb ftdi_sync ft232h_bfm_tb {state clk rxf_n txe_n rd_n wr_n
   ↪  siwu_n oe_n rst_n write_data tdata tvalid tready pc_data}
6  module ft232h_bfm_tb;
7
8      typedef enum int {
9          INIT,
10         RESET,
11         WRITE_AWAIT,
12         WRITING,
13         CHECK_ON_PC,
14         READ_OUT,
15         DONE
16     } ft232h_bfm_tb_state_t;
17
18     wire clk;
19
20     wire rxf_n;
21     wire txe_n;
22
23     var logic rd_n;
24     var logic wr_n;
25     var logic siwu_n;
26     var logic oe_n;
27     var logic rst_n;
28
29     var logic[7:0] write_data;
30
31     // PC Side wires (FIFO output)
32     wire[7:0] tdata;
33     wire tvalid;
34     var logic tready;
35
36     ft232h_bfm_tb_state_t state;
37     initial begin
38         rd_n = 1;
39         wr_n = 1;
40         siwu_n = 1;
41         oe_n = 1;
42         rst_n = 1;
43
44         tready = 0;
45         state = INIT;
46     end
47
48     ft232h_bfm ftdi (
49         .clk(clk),
50
51         .rxf_n(rxf_n),
```

```
52          .txe_n(txe_n),
53
54          .rd_n(rd_n),
55          .wr_n(wr_n),
56          .siwu_n(siwu_n),
57          .oe_n(oe_n),
58          .rst_n(rst_n),
59
60          .data(write_data),
61
62          // PC Side Signals (FIFO output)
63          .tdata(tdata),
64          .tvalid(tvalid),
65          .tready(tready)
66      );
67
68      var logic[7:0] pc_data;
69
70      always @(posedge clk) begin
71          case (state)
72              INIT: begin
73                  // Start reset procedure
74                  rst_n <= 0;
75                  state <= RESET;
76              end
77              RESET: begin
78                  rst_n <= 1;
79                  state <= WRITE_AWAIT;
80              end
81              WRITE_AWAIT: begin
82                  if (~txe_n) begin
83                      write_data <= 69;
84                      wr_n <= 0;
85                      state <= WRITING;
86                  end
87              end
88              WRITING: begin
89                  if (txe_n) begin
90                      wr_n <= 1;
91                      state <= CHECK_ON_PC;
92                  end else begin
93                      // If the FIFO still has space write more data
94                      write_data <= write_data + 1;
95                  end
96              end
97              CHECK_ON_PC: begin
98                  if (tvalid) begin
```

```
                    tready <= 1; // tell the FIFO that We want to start
                    ↪  clocking outputs
                    state <= READ_OUT;
                end
            end
            READ_OUT: begin
                if (tvalid) begin
                    pc_data <= tdata;
                end else begin
                    tready <= 0;
                    state <= DONE;
                end
            end
            DONE: begin
                $stop;
            end
        endcase
    end

endmodule

`default_nettype wire
```

D.6.2 ft232h_tb.sv

```
`default_nettype none
`timescale 1ns / 1ps
// timed_tb ftdi_sync ft232h_tb {sys_clk ftdi_clk state ftdi_txe_n
↪  ftdi_wr_n ftdi_adbus sys_axis.tdata sys_axis.tready sys_axis.tvalid
↪  pc_tvalid pc_tready pc_tdata}
module ft232h_tb;

    typedef enum int {
        AWAIT_FTDI,
        SEND_TO_FTDI,
        READ_OUT,
        DONE
    } ft232h_tb_state_t;

    ft232h_tb_state_t state;

    wire ftdi_clk;
    var logic sys_clk;

    wire ftdi_rxf_n;
    wire ftdi_txe_n;

    wire ftdi_rd_n;
```

```
22      wire ftdi_wr_n;
23      wire ftdi_siwu_n;
24      wire ftdi_oe_n;
25      wire ftdi_rst_n;
26
27      wire[7:0] ftdi_adbus;
28
29      // PC Side wires (FIFO output)
30      wire[7:0] pc_tdata;
31      wire pc_tvalid;
32      var logic pc_tready;
33
34      var logic second_write;
35
36      axis_interface sys_axis (
37          .clk(sys_clk),
38          .rst(0)
39      );
40
41      // get variable logic into initial vals
42      initial begin
43          sys_clk = 0;
44          pc_tready = 0;
45          second_write = 0;
46
47          sys_axis.tdata = 69;
48          sys_axis.tvalid = 0;
49
50          state = AWAIT_FTDI;
51      end
52
53      always begin
54          #100
55          sys_clk = ~sys_clk;
56      end
57
58      ft232h ftdi_dut (
59          .ftdi_clk(ftdi_clk),
60
61          .ftdi_rxf_n(ftdi_rxf_n),
62          .ftdi_txe_n(ftdi_txe_n),
63
64          .ftdi_rd_n(ftdi_rd_n),
65          .ftdi_wr_n(ftdi_wr_n),
66          .ftdi_siwu_n(ftdi_siwu_n),
67          .ftdi_oe_n(ftdi_oe_n),
68
69          .ftdi_adbus(ftdi_adbus),
```

```
70
71          // Programmer AXIS Interface
72          .sys_axis(sys_axis.Sink)
73      );
74
75      ft232h_bfm bfm (
76          .clk(ftdi_clk),
77
78          .rxf_n(ftdi_rxf_n),
79          .txe_n(ftdi_txe_n),
80
81          .rd_n(ftdi_rd_n),
82          .wr_n(ftdi_wr_n),
83          .siwu_n(ftdi_siwu_n),
84          .oe_n(ftdi_oe_n),
85          .rst_n(ftdi_rst_n),
86
87          .data(ftdi_adbus),
88
89          // PC Side Signals (FIFO output)
90          .tdata(pc_tdata),
91          .tvalid(pc_tvalid),
92          .tready(pc_tready)
93      );
94
95      always @(posedge sys_axis.clk) begin
96          case(state)
97              AWAIT_FTDI: begin
98                  if (sys_axis.tready) begin
99                      sys_axis.tvalid <= 1;
100                     state <= SEND_TO_FTDI;
101                 end
102             end
103             SEND_TO_FTDI: begin
104                 if (sys_axis.tready && sys_axis.tvalid) begin
105                     sys_axis.tdata <= sys_axis.tdata + 1;
106                 end else begin
107                     sys_axis.tvalid <= 0;
108                     pc_tready <= 1;
109                     state <= READ_OUT;
110                 end
111             end
112             DONE: begin
113                 $stop;
114             end
115         endcase
116     end
117
```

```
    always @(posedge ftdi_clk) begin
        // consume readout from FTDI clock domain
        case (state)
            READ_OUT: begin
                // Read out the pc_tdata signal in sim. Proceed to done
                ↪  when there is no more data
                if (!(pc_tvalid && pc_tready)) begin
                    state <= DONE;
                end
            end
        endcase
    end

endmodule

`default_nettype wire
```

D.6.3 xadc_bfm_tb.sv

```
`default_nettype none
`timescale 1ns / 1ps

// timed_tb xadc xadc_bfm_tb {dclk_in reset_in state daddr_in den_in
↪  drdy_out do_out eos_out current_addr}

// Runs a read cycle against the BFM to verify its function. In this
↪  tb, I will
// read from both the current monitor and voltage channel addresses for
↪  teachee
module xadc_bfm_tb;

    typedef enum int {
        INIT,
        AWAIT_CONVERSION_SEQ,
        START_DRP_READ,
        WAIT_CYCLE0,
        WAIT_CYCLE1,
        VIEW_RESULTS,
        DONE
    } xadc_bfm_tb_state_t;

    xadc_bfm_tb_state_t state;

    var logic dclk_in;
    var logic reset_in;
    var logic[6:0] daddr_in;
    var logic den_in;

```

```
27      wire drdy_out;
28      wire[15:0] do_out;
29      wire eos_out;
30
31      // Set initial state and signal values
32      initial begin
33          dclk_in = 0;
34          reset_in = 0;
35          daddr_in = 0;
36          den_in = 0;
37          state = INIT;
38      end
39
40      always begin
41          #10 // not true clock freq, for test purposes. Varies based off
                ↪  sample rate
42          dclk_in = ~dclk_in;
43      end
44
45      xadc_bfm DUT (
46          // Clock and reset
47          .dclk_in(dclk_in),
48          .reset_in(reset_in),
49
50          // DRP interface (Reduced to read only in this tb)
51          .di_in(0), // not used since we are not writing
52          .daddr_in(daddr_in),
53          .den_in(den_in),
54          .dwe_in(0), // write enable is also not used
55          .drdy_out(drdy_out),
56          .do_out(do_out),
57
58          // Must be connected in the real xadc IP core
59          .vp_in(0),
60          .vn_in(0),
61
62          // Analog input channels (also need to be connected in real IP
                ↪  core)
63          .vauxp4(0),
64          .vauxn4(0),
65          .vauxp12(0),
66          .vauxn12(0),
67
68          // conversion status signals
69          .channel_out(),
70          .eoc_out(),
71          .alarm_out(), // not used
72          .eos_out(eos_out),
```

```
73          .busy_out()
74      );
75
76      var logic[6:0] current_addr;
77      // DRP Read FSM
78      always @(posedge dclk_in) begin
79          case (state)
80              INIT: begin
81                  // Set signals to their initial values
82                  dclk_in <= 0;
83                  reset_in <= 0;
84                  daddr_in <= 0;
85                  den_in <= 0;
86
87                  current_addr <= 7'h14; // start with current channel
88
89                  // AWAIT EOS SIGNAL
90                  state <= AWAIT_CONVERSION_SEQ;
91              end
92              AWAIT_CONVERSION_SEQ: begin
93                  if (eos_out) begin
94                      // If eos goes high, new samples have been written
                        ↪  to regs.
95                      // Read out the current and voltage sample
96
97                      // load up the address and pull the read enable
98                      daddr_in <= current_addr;
99                      den_in <= 1;
100                     state <= START_DRP_READ;
101                 end
102             end
103             START_DRP_READ: begin
104                 // move current address to the voltage channel
105
106                 daddr_in <= 0;
107                 den_in <= 0;
108
109                 state <= WAIT_CYCLE0;
110             end
111             WAIT_CYCLE0: state <= WAIT_CYCLE1;
112             WAIT_CYCLE1: state <= VIEW_RESULTS;
113             VIEW_RESULTS: begin
114                 if (current_addr == 7'h14) begin
115                     current_addr <= 7'h1c;
116                     state <= AWAIT_CONVERSION_SEQ;
117                 end else state <= DONE;
118             end
119             DONE: $stop;
```

```
        endcase
    end

endmodule

`default_nettype wire
```

D.6.4 xadc_drp_axis_adapter_tb.sv

```
`default_nettype none
`timescale 1ns / 1ps

import teachee_defs::*;

module xadc_drp_axis_adapter_tb;
    typedef enum int {
        READ_VOLTAGE_SAMPLE,
        VIEW_VOLTAGE_SAMPLE,
        READ_CURRENT_SAMPLE,
        VIEW_CURRENT_SAMPLE,
        DONE
    } xadc_drp_axis_adapter_tb_state_t;

    xadc_drp_axis_adapter_tb_state_t state;

    var logic xadc_dclk;
    var logic xadc_reset;
    xadc_drp_addr_t xadc_daddr;
    var logic xadc_den;

    var logic xadc_drdy;
    var logic[15:0] xadc_do;
    var logic xadc_eos;

    initial begin
        xadc_dclk = 0;
        xadc_reset = 0;

        voltage_channel.tready = 1;
        current_monitor_channel.tready = 1;
    end

    always begin
        #10
        xadc_dclk = ~xadc_dclk;
    end

    // Create AXI interfaces to connect up to adapter outputs
```

```
40      axis_interface #(
41          .DATA_WIDTH(16)
42      ) voltage_channel (
43          .clk(xadc_dclk),
44          .rst(1'b0)
45      );
46
47      axis_interface #(
48          .DATA_WIDTH(16)
49      ) current_monitor_channel (
50          .clk(xadc_dclk),
51          .rst(1'b0)
52      );
53
54      xadc_drp_axis_adapter adapter_dut (
55          .xadc_dclk(xadc_dclk),
56          .xadc_reset(0),
57
58          // DRP and Conversion Signals
59          .xadc_daddr(xadc_daddr),
60          .xadc_den(xadc_den),
61          .xadc_drdy(xadc_drdy),
62          .xadc_do(xadc_do),
63
64          .xadc_eos(xadc_eos),
65
66          .current_monitor_channel(current_monitor_channel.Source),
67          .voltage_channel(voltage_channel.Source)
68      );
69
70      xadc_bfm xadc_bfm_inst (
71          // Clock and reset
72          .dclk_in(xadc_dclk),
73          .reset_in(xadc_reset),
74
75          // DRP interface (Reduced to read only in this tb)
76          .di_in(0), // not used since we are not writing
77          .daddr_in(xadc_daddr),
78          .den_in(xadc_den),
79          .dwe_in(0), // write enable is also not used
80          .drdy_out(xadc_drdy),
81          .do_out(xadc_do),
82
83          // Must be connected in the real xadc IP core
84          .vp_in(0),
85          .vn_in(0),
86
```

```
87          // Analog input channels (also need to be connected in real IP
            ↪  core)
88          .vauxp4(0),
89          .vauxn4(0),
90          .vauxp12(0),
91          .vauxn12(0),
92
93          // conversion status signals
94          .channel_out(),
95          .eoc_out(),
96          .alarm_out(), // not used
97          .eos_out(xadc_eos),
98          .busy_out()
99      );
100
101     always @(posedge xadc_dclk) begin
102         case (state)
103             READ_VOLTAGE_SAMPLE: begin
104                 if (voltage_channel.tready && voltage_channel.tvalid)
                    ↪  begin
105                     state <= VIEW_VOLTAGE_SAMPLE;
106                 end
107             end
108             VIEW_VOLTAGE_SAMPLE: begin
109                 state <= READ_CURRENT_SAMPLE;
110             end
111             READ_CURRENT_SAMPLE: begin
112                 if (current_monitor_channel.tready &&
                    ↪  current_monitor_channel.tvalid) begin
113                     state <= VIEW_CURRENT_SAMPLE;
114                 end
115             end
116             VIEW_CURRENT_SAMPLE: begin
117                 state <= DONE;
118             end
119             DONE: begin
120                 $stop;
121             end
122         endcase
123     end
124 endmodule
125
126 `default_nettype wire
```

D.7 VUnit Testbenches

D.7.1 axis_adapter_wrapper_tb.sv

```
`default_nettype none
`timescale 1ns / 1ps

`include "vunit_defines.svh"

module axis_adapter_wrapper_tb;

    var logic reset = 0;
    var logic clk;

    typedef enum int {
        SEND_ORIGINAL_DATA,
        READ_OUT_FIRST_BYTE,
        READ_OUT_SECOND_BYTE,
        TESTS_FINISHED
    } axis_adapter_wrapper_tb_state_t;

    axis_adapter_wrapper_tb_state_t state = SEND_ORIGINAL_DATA;

    always begin
        #10
        clk <= !clk;
    end

    axis_interface #(
        .DATA_WIDTH(16)
    ) original_data (
        .clk(clk),
        .rst(reset)
    );

    axis_interface #(
        .DATA_WIDTH(8)
    ) modified_width_data (
        .clk(clk),
        .rst(reset)
    );

    axis_adapter_wrapper #(
        .S_DATA_WIDTH(16),
        .M_DATA_WIDTH(8)
    ) DUT (
        .original_data(original_data.Sink),
        .modified_width_data(modified_width_data.Source)
```

```
45      );
46
47      `TEST_SUITE begin
48          `TEST_SUITE_SETUP begin
49              // what would normally go in an initial block we can put
                    ↪  here
50              clk = 0;
51              reset = 0;
52
53              original_data.tdata = 16'h6971;
54              original_data.tvalid = 0;
55              original_data.tlast = 1;
56              original_data.tuser = 0;
57              original_data.tkeep <= '1;
58              original_data.tid <= '0;
59              original_data.tdest <= '0;
60
61              modified_width_data.tready = 0;
62          end
63
64          `TEST_CASE("VIEW_REDUCED_WIDTH STREAM") begin
65              while (state != TESTS_FINISHED) begin
66                  @(posedge clk) begin
67                      case (state)
68                          SEND_ORIGINAL_DATA: begin
69                              original_data.tvalid <= 1;
70                              if (original_data.tready &&
                                    ↪  original_data.tvalid) begin
71                                  modified_width_data.tready <= 1;
72                                  original_data.tvalid <= 0;
73
74                                  state <= READ_OUT_FIRST_BYTE;
75                              end
76                          end
77                          READ_OUT_FIRST_BYTE: begin
78                              if (modified_width_data.tready &&
                                    ↪  modified_width_data.tvalid) begin
79                                  `CHECK_EQUAL(modified_width_data.tdata,
                                        ↪  8'h71);
80
81                                  state <= READ_OUT_SECOND_BYTE;
82                              end
83                          end
84                          READ_OUT_SECOND_BYTE: begin
85                              if (modified_width_data.tready &&
                                    ↪  modified_width_data.tvalid) begin
86                                  `CHECK_EQUAL(modified_width_data.tdata,
                                        ↪  8'h69);
```

```
                                    modified_width_data.tready <= 0;

                                    state <= TESTS_FINISHED;
                                end
                            end
                        endcase
                    end
                end
            end
        end
        `WATCHDOG(0.1ms);
endmodule

`default_nettype wire
```

D.7.2 cobs_axis_adapter_wrapper_tb.sv

```
`default_nettype none
`timescale 1ns / 1ps

`include "vunit_defines.svh"

module cobs_axis_adapter_wrapper_tb;

    var logic reset = 0;
    var logic clk;

    typedef enum int {
        SEND_ORIGINAL_DATA,
        READ_OUT_FIRST_BYTE,
        READ_OUT_SECOND_BYTE,
        READ_OUT_THIRD_BYTE,
        READ_OUT_FOURTH_BYTE,
        TESTS_FINISHED
    } cobs_axis_adapter_wrapper_tb_state_t;

    cobs_axis_adapter_wrapper_tb_state_t state = SEND_ORIGINAL_DATA;

    always begin
        #10
        clk <= !clk;
    end

    axis_interface #(
        .DATA_WIDTH(16)
    ) original_data (
        .clk(clk),
        .rst(reset)
```

```
32      );
33
34      axis_interface #(
35          .DATA_WIDTH(8)
36      ) encoded_data (
37          .clk(clk),
38          .rst(reset)
39      );
40
41      axis_interface #(
42          .DATA_WIDTH(16)
43      ) fifo_data (
44          .clk(clk),
45          .rst(reset)
46      );
47
48      // want to see how the module behaves against a FIFO output since
        ↪  that is
49      // what it will get in use
50      axis_async_fifo_wrapper #(
51          .DATA_WIDTH(16),
52          .DEPTH(5),
53          .KEEP_ENABLE(0)
54      ) test_fifo (
55          .sink(original_data.Sink),
56          .source(fifo_data.Source)
57      );
58
59      cobs_axis_adapter_wrapper #(
60          .S_DATA_WIDTH(16),
61          .M_DATA_WIDTH(8)
62      ) DUT (
63          .original_data(fifo_data.Sink),
64          .encoded_data(encoded_data.Source)
65      );
66
67      `TEST_SUITE begin
68          `TEST_SUITE_SETUP begin
69              // what would normally go in an initial block we can put
                ↪  here
70              clk = 0;
71              reset = 0;
72
73              original_data.tdata = 16'h6971;
74              original_data.tvalid = 1;
75              original_data.tlast = 1;
76              original_data.tuser = 0;
77              original_data.tkeep = '1;
```

```
78              original_data.tid = '0;
79              original_data.tdest = '0;
80
81              encoded_data.tready = 0;
82
83          end
84
85          `TEST_CASE("VIEW_REDUCED_WIDTH_COBS_STREAM") begin
86              // Load the FIFO UP
87              for (int i = 0; i < 5; i = i + 1) begin
88                  @(posedge clk) begin
89                      if (original_data.tready && original_data.tvalid)
                        ↪  begin
90                          original_data.tdata = original_data.tdata + 1;
91                      end
92                  end
93              end
94
95              while (state != TESTS_FINISHED) begin
96                  @(posedge clk) begin
97                      case (state)
98                          SEND_ORIGINAL_DATA: begin
99                              original_data.tvalid <= 1;
100                             if (original_data.tready &&
                                ↪  original_data.tvalid) begin
101                                 encoded_data.tready <= 1;
102                                 original_data.tvalid <= 0;
103
104                                 state <= READ_OUT_FIRST_BYTE;
105                             end
106                         end
107                         READ_OUT_FIRST_BYTE: begin
108                             if (encoded_data.tready &&
                                ↪  encoded_data.tvalid) begin
109                                 `CHECK_EQUAL(encoded_data.tdata,
                                    ↪  8'h03);
110
111                                 state <= READ_OUT_SECOND_BYTE;
112                             end
113                         end
114                         READ_OUT_SECOND_BYTE: begin
115                             if (encoded_data.tready &&
                                ↪  encoded_data.tvalid) begin
116                                 // `CHECK_EQUAL(encoded_data.tdata,
                                    ↪  8'h71);
117
118                                 state <= READ_OUT_THIRD_BYTE;
119                             end
```

```
                        end
                        READ_OUT_THIRD_BYTE: begin
                            if (encoded_data.tready &&
                             ↪  encoded_data.tvalid) begin
                                // `CHECK_EQUAL(encoded_data.tdata,
                                 ↪  8'h69);

                                state <= READ_OUT_FOURTH_BYTE;
                            end

                        end
                        READ_OUT_FOURTH_BYTE: begin
                            if (encoded_data.tready &&
                             ↪  encoded_data.tvalid) begin
                                // `CHECK_EQUAL(encoded_data.tdata,
                                 ↪  8'h00);

                                encoded_data.tready <= 0;

                                state <= TESTS_FINISHED;
                            end
                        end
                    endcase
                end
            end
            `CHECK_EQUAL(0, 0);
        end
    end
    `WATCHDOG(0.1ms);
endmodule

`default_nettype wire
```

D.7.3 cobs_encode_wrapper_tb.sv

```
`default_nettype none
`timescale 1ns / 1ps

`include "vunit_defines.svh"

module cobs_encode_wrapper_tb;

    var logic reset;
    var logic clk;

    // Will be 16 bits of sample data plus 0 byte overhead
    var logic[31:0] encoded_packet;

```

```
14      typedef enum int {
15          LOAD_FIRST_BYTE,
16          LOAD_SECOND_BYTE,
17          CONSUME_FIRST_BYTE,
18          CONSUME_SECOND_BYTE,
19          CONSUME_THIRD_BYTE,
20          CONSUME_FOURTH_BYTE,
21          RUN_CHECK
22      } cobs_encode_wrapper_tb_state_t;
23
24      cobs_encode_wrapper_tb_state_t state;
25
26      always begin
27          #10
28          clk <= !clk;
29      end
30
31      axis_interface #(
32          .DATA_WIDTH(8)
33      ) raw_stream (
34          .clk(clk),
35          .rst(reset)
36      );
37
38      axis_interface #(
39          .DATA_WIDTH(8)
40      ) encoded_stream (
41          .clk(clk),
42          .rst(reset)
43      );
44
45      cobs_encode_wrapper DUT (
46          .raw_stream(raw_stream),
47          .encoded_stream(encoded_stream)
48      );
49
50      `TEST_SUITE begin
51          `TEST_SUITE_SETUP begin
52              // what would normally go in an initial block we can put
                ↪  here
53              clk = 0;
54              reset = 0;
55
56              // Configure the raw_stream
57              raw_stream.tdata = 'h69;
58              raw_stream.tvalid = 1;
59              raw_stream.tlast = 0;
60              raw_stream.tuser = 0;
```

```
61
62              // configure the encoded output
63              encoded_stream.tready = 0;
64
65              // initialize state
66              encoded_packet = 0;
67          end
68
69          `TEST_CASE("CHECK_COBS_OUTPUT") begin
70              automatic int bytes_loaded = 0;
71              automatic int bytes_consumed = 0;
72              // load in the bytes into the input FIFO
73              while (bytes_loaded < 2) begin
74                  @(posedge clk) begin
75                      if (raw_stream.tvalid && raw_stream.tready) begin
76                          raw_stream.tdata = raw_stream.tdata + 1;
77                          bytes_loaded = bytes_loaded + 1;
78                          if (bytes_loaded == 2) begin
79                              raw_stream.tlast = 1;
80                          end
81                      end
82                  end
83              end
84              @(posedge clk) begin
85                  raw_stream.tlast = 0;
86              end
87
88              // now consume the output
89              encoded_stream.tready = 1;
90              while (bytes_consumed < 5) begin
91                  @(posedge clk) begin
92                      if (encoded_stream.tvalid && encoded_stream.tready)
                        ↪  begin
93                          bytes_consumed = bytes_consumed + 1;
94                          // now check each byte for equality with the
                            ↪  expected packet data
95                          case (bytes_consumed)
96                              1: begin
97                                  `CHECK_EQUAL(encoded_stream.tdata,
                                    ↪  'h04);
98                              end
99                              2: begin
100                                 `CHECK_EQUAL(encoded_stream.tdata,
                                    ↪  'h69);
101                             end
102                             3: begin
103                                 `CHECK_EQUAL(encoded_stream.tdata,
                                    ↪  'h6a);
```

```
                                end
                                4: begin
                                    `CHECK_EQUAL(encoded_stream.tdata,
                                    ↪  'h6b);
                                end
                                5: begin
                                    `CHECK_EQUAL(encoded_stream.tdata,
                                    ↪  'h00);
                                end
                            endcase
                        end
                    end
                end
            end
        end

        `WATCHDOG(0.1ms);
endmodule

`default_nettype wire
```

D.7.4 hsadc_axis_wrapper_tb.sv

```
`default_nettype none
`timescale 1ns / 1ps

`include "vunit_defines.svh"

module hsadc_axis_wrapper_tb;
    var logic reset;

    var logic sample_clk;
    var logic stream_clk;

    always begin
        #10
        stream_clk <= !stream_clk;
    end

    always begin
        #100
        sample_clk <= !sample_clk;
    end

    axis_interface #(
        .DATA_WIDTH(16)
    ) hsadc_stream_axis (
        .clk(stream_clk),
```

```
26          .rst(reset)
27      );
28
29      hsadc_interface hsadc_ctrl_signals ();
30
31      hsadc_axis_wrapper hsadc_wrapper (
32          .sample_clk(sample_clk),
33          .stream_clk(stream_clk),
34          .reset(reset),
35
36          .hsadc_ctrl_signals(hsadc_ctrl_signals.Sink),
37          .sample_stream(hsadc_stream_axis)
38      );
39
40      hsadc_bfm bfm (
41          .hsadc_ctrl_signals(hsadc_ctrl_signals.Source)
42      );
43
44      var logic [7:0] counter = 0;
45      `TEST_SUITE begin
46          `TEST_SUITE_SETUP begin
47              sample_clk = 0;
48              stream_clk = 0;
49              reset = 0;
50
51          end
52
53          `TEST_CASE("CHECK_FIRST_SAMPLES") begin
54              hsadc_stream_axis.tready = 1;
55              while (1) begin
56                  @(posedge stream_clk) begin
57                      // Run checks here
58                      if (hsadc_stream_axis.tready &&
                            ↪  hsadc_stream_axis.tvalid) begin
59                          `CHECK_EQUAL(hsadc_stream_axis.tdata[7:0],
                                ↪  counter);
60                          `CHECK_EQUAL(hsadc_stream_axis.tdata[15:8],
                                ↪  counter);
61                          counter <= counter + 1;
62                          $display("passed counter = %d, assertion",
                                ↪  counter);
63                      end
64
65                      if (counter == 10) begin
66                          break;
67                      end
68
69                  end
```

```
            end
        end
    end

endmodule

`default_nettype none
```

D.7.5 xadc_packetizer_tb.sv

```
`default_nettype none
`timescale 1ns / 1ps

`include "vunit_defines.svh"

import xadc_drp_package::*;

module xadc_packetizer_tb;

    var logic reset;
    var logic clk;

    always begin
        #10
        clk <= !clk;
    end

    // Define AXIS interfaces for the packetizer
    axis_interface #(
        .DATA_WIDTH(XADC_DRP_DATA_WIDTH)
    ) voltage_channel (
        .clk(clk),
        .rst(reset)
    );

    axis_interface #(
        .DATA_WIDTH(XADC_DRP_DATA_WIDTH)
    ) current_monitor_channel (
        .clk(clk),
        .rst(reset)
    );

    axis_interface #(
        .DATA_WIDTH(8)
    ) cobs_stream (
        .clk(clk),
        .rst(reset)
    );
```

```
39
40      // Declare XADC BFM and AXIS Adapter
41      xadc_drp_addr_t xadc_daddr;
42      var logic xadc_den;
43      var logic xadc_drdy;
44      var logic[XADC_DRP_DATA_WIDTH-1:0] xadc_do;
45      var logic xadc_eos;
46
47      xadc_bfm xadc (
48          .dclk_in(clk),
49          .reset_in(reset),
50
51          // DRP
52          .di_in(0),
53          .daddr_in(xadc_daddr),
54          .den_in(xadc_den),
55          .dwe_in(0),
56          .drdy_out(xadc_drdy),
57          .do_out(xadc_do),
58
59          .vp_in(0),
60          .vn_in(0),
61
62          .vauxp4(0),
63          .vauxn4(0),
64          .vauxp12(0),
65          .vauxn12(0),
66
67          .channel_out(),
68          .eoc_out(),
69
70          .alarm_out(),
71          .eos_out(xadc_eos),
72          .busy_out()
73      );
74
75      xadc_drp_axis_adapter xadc_axis_adapter (
76          .xadc_dclk(clk),
77          .xadc_reset(reset),
78
79          // DRP
80          .xadc_daddr(xadc_daddr),
81          .xadc_den(xadc_den),
82          .xadc_drdy(xadc_drdy),
83          .xadc_do(xadc_do),
84          .xadc_eos(xadc_eos),
85
86          // AXIS OUTPUTS
```

```
87          .current_monitor_channel(current_monitor_channel.Source),
88          .voltage_channel(voltage_channel.Source)
89
90      );
91
92      xadc_packetizer DUT (
93          .voltage_channel(voltage_channel.Sink),
94          .current_monitor_channel(current_monitor_channel.Sink),
95          .packet_stream(cobs_stream)
96      );
97
98      `TEST_SUITE begin
99          `TEST_SUITE_SETUP begin
100             // what would normally go in an initial block we can put
                ↪  here
101             clk = 0;
102             reset = 0;
103             cobs_stream.tready = 1;
104         end
105
106         `TEST_CASE("VERIFY_COBS_VOLTAGE_CURRENT_PACKETS") begin
107             // if we stick these two values into a cobs encoder we get
                ↪  the
108             // following byte sequence we can check
109
110             // input sequence
111             // 0x00 0x0F 0x00 0x07
112
113             // output sequence
114             // 0x01 0x02 0x0F 0x02 0x07 0x00
115
116
117             automatic int bytes_consumed = 0;
118             while (bytes_consumed < 6) begin
119                 @(posedge clk) begin
120                     if (cobs_stream.tready && cobs_stream.tvalid) begin
121                         bytes_consumed = bytes_consumed + 1;
122                         case (bytes_consumed)
123                             1: begin
124                                 `CHECK_EQUAL(cobs_stream.tdata, 'h01);
125                             end
126                             2: begin
127                                 `CHECK_EQUAL(cobs_stream.tdata, 'h02);
128                             end
129                             3: begin
130                                 `CHECK_EQUAL(cobs_stream.tdata, 'h0F);
131                             end
132                             4: begin
```

```
                                    `CHECK_EQUAL(cobs_stream.tdata, 'h02);
                                end
                                5: begin
                                    `CHECK_EQUAL(cobs_stream.tdata, 'h07);
                                end
                                6: begin
                                    `CHECK_EQUAL(cobs_stream.tdata, 'h00);
                                end
                            endcase
                        end
                    end
                end
                // Placeholder until full TB is implemented
            end
        end

        `WATCHDOG(1ms);
    endmodule

    `default_nettype wire
```

D.8 Example Projects

D.8.1 blink

```
module blink(
    output led,
    output TEACHEE_LED0,
    output TEACHEE_LED1,
    input btn,
    input sysclk
);

    logic led_state = 0;
    logic[31:0] counter = 0;
    assign led = btn;

    always @(posedge sysclk) begin
        counter <= counter + 1;
        if (counter == 6000000) begin
            led_state <= ~led_state;
            counter <= 0;
        end
    end
    assign TEACHEE_LED0 = led_state;
    assign TEACHEE_LED1 = ~led_state;
endmodule
```

D.8.2 ftdi_sync

```
`default_nettype none
`timescale 1ns / 1ps

// TODO: Write about ftdi set bit mode prerequisites here.
module ftdi_sync (
    input var logic sys_clk, // 12 MHz provided on the CMOD
    input var logic ftdi_clk, // 60 MHz provided by the FTDI

    // FTDI Control Interface
    input var logic ftdi_rxf_n,
    input var logic ftdi_txe_n,

    output var logic ftdi_rd_n,
    output var logic ftdi_wr_n,
    output var logic ftdi_siwu_n,
    output var logic ftdi_oe_n,

    output var logic[7:0] ftdi_data,

    // CMOD IO Declarations
    input var logic[1:0] btn,
    output var logic[1:0] led,

    // TeachEE IO Declarations
    output var logic[1:0] teachee_led
);

    typedef enum int {
        INIT,
        IDLE,
        SEND_TO_HOST
    } state_t;

    state_t state = IDLE;

    // Programmer data input side wires
    axis_interface sys_axis (
        .clk(sys_clk),
        .rst(0)
    );


    ft232h usb_fifo (
        .ftdi_clk(ftdi_clk),

        .ftdi_rxf_n(ftdi_rxf_n),
```

```
47          .ftdi_txe_n(ftdi_txe_n),
48
49          .ftdi_rd_n(ftdi_rd_n),
50          .ftdi_wr_n(ftdi_wr_n),
51          .ftdi_siwu_n(ftdi_siwu_n),
52          .ftdi_oe_n(ftdi_oe_n),
53
54          .ftdi_adbus(ftdi_data),
55
56          // Programmer AXIS Interface
57          .sys_axis(sys_axis.Sink)
58      );
59
60      always_ff @(posedge sys_clk) begin
61          // Do write state machine here
62          case (state)
63              INIT: begin
64                  sys_axis.tdata <= 69;
65                  sys_axis.tvalid <= 0;
66                  state <= IDLE;
67              end
68              IDLE: begin
69                  if (sys_axis.tready) begin
70                      sys_axis.tvalid <= 1;
71                      state <= SEND_TO_HOST;
72                  end
73              end
74              SEND_TO_HOST: begin
75                  if (sys_axis.tready && sys_axis.tvalid) begin
76                      sys_axis.tdata <= sys_axis.tdata + 1;
77                  end
78                  sys_axis.tvalid <= 0;
79                  state <= IDLE;
80              end
81          endcase
82      end
83
84      // Set defaults for unused signals
85      always begin
86          sys_axis.tlast <= 1;
87          sys_axis.tkeep <= '1;
88          sys_axis.tid <= '0;
89          sys_axis.tuser <= '0;
90          sys_axis.tdest <= '0;
91      end
92  endmodule
93
94  `default_nettype wire
```

D.8.3 pll_blink

```
`default_nettype none
`timescale 1ns / 1ps

module pll_blink (
    output var logic TEACHEE_LED0,
    output var logic TEACHEE_LED1,
    input var logic cmod_osc
);

    var logic sys_clk;
    var logic locked;

    pll_100 pll_example_100 (
        // Clock out ports
        .sys_clk(sys_clk),     // output sys_clk
        // Status and control signals
        .reset(0), // input reset
        .locked(locked),       // output locked
        // Clock in ports
        .cmod_osc(cmod_osc)      // input osc_in
    );

    var logic[31:0] counter = 0;
    always_ff @(posedge sys_clk) begin
        if (locked) begin
            counter <= counter + 1;
            if (counter == 100_000_000) begin
                counter <= 0;
                TEACHEE_LED0 <= ~TEACHEE_LED0;
                TEACHEE_LED1 <= ~TEACHEE_LED1;
            end
        end else begin
            TEACHEE_LED0 <= 0;
            TEACHEE_LED1 <= 0;
        end
    end

endmodule

`default_nettype wire
```

D.8.4 xadc_axis

```
`default_nettype none
`timescale 1ns / 1ps

```

```
4  import xadc_drp_package::*;
5  module xadc_axis (
6      input var logic sys_clk, // 12 MHz provided on the CMOD
7      input var logic ftdi_clk, // 60 MHz provided by the FTDI
8
9      // FTDI Control Interface
10     input var logic ftdi_rxf_n,
11     input var logic ftdi_txe_n,
12
13     output var logic ftdi_rd_n,
14     output var logic ftdi_wr_n,
15     output var logic ftdi_siwu_n,
16     output var logic ftdi_oe_n,
17
18     output var logic[7:0] ftdi_data,
19
20     // CMOD IO Declarations
21     input var logic[1:0] btn,
22     output var logic[1:0] led,
23
24     input var logic[1:0] xa_n,
25     input var logic[1:0] xa_p,
26
27     // TeachEE IO Declarations
28     output var logic[1:0] teachee_led
29 );
30
31     axis_interface #(
32         .DATA_WIDTH(32)
33     ) voltage_channel (
34         .clk(sys_clk),
35         .rst(0)
36     );
37
38     axis_interface #(
39         .DATA_WIDTH(16)
40     ) current_monitor_channel (
41         .clk(sys_clk),
42         .rst(0)
43     );
44
45     axis_interface #(
46         .DATA_WIDTH(8)
47     ) sys_axis (
48         .clk(sys_clk),
49         .rst(0)
50     );
51
```

```
52      ft232h usb_fifo (
53          .ftdi_clk(ftdi_clk),
54
55          .ftdi_rxf_n(ftdi_rxf_n),
56          .ftdi_txe_n(ftdi_txe_n),
57
58          .ftdi_rd_n(ftdi_rd_n),
59          .ftdi_wr_n(ftdi_wr_n),
60          .ftdi_siwu_n(ftdi_siwu_n),
61          .ftdi_oe_n(ftdi_oe_n),
62
63          .ftdi_adbus(ftdi_data),
64
65          // Programmer AXIS Interface
66          .sys_axis(sys_axis.Sink)
67      );
68
69      var logic xadc_reset;
70      xadc_drp_addr_t xadc_daddr;
71      var logic xadc_den;
72
73      var logic xadc_drdy;
74      var logic[15:0] xadc_do;
75      var logic xadc_eos;
76
77
78      xadc_drp_axis_single_stream xadc_drp_axis_adapter_inst (
79          .xadc_dclk(sys_clk),
80          .xadc_reset(0),
81
82          // DRP and Conversion Signals
83          .xadc_daddr(xadc_daddr),
84          .xadc_den(xadc_den),
85          .xadc_drdy(xadc_drdy),
86          .xadc_do(xadc_do),
87
88          .xadc_eos(xadc_eos),
89
90          // .vauxp4(xa_p[0]),
91          // .vauxn4(xa_n[0]),
92
93          // .vauxp12(xa_p[1]),
94          // .vauxn12(xa_n[1]),
95
96          .sample_stream(voltage_channel.Source)
97      );
98      // xadc_drp_axis_adapter xadc_drp_axis_adapter_inst (
99      //     .xadc_dclk(sys_clk),
```

```
100     //      .xadc_reset(0),
101
102     //      // DRP and Conversion Signals
103     //      .xadc_daddr(xadc_daddr),
104     //      .xadc_den(xadc_den),
105     //      .xadc_drdy(xadc_drdy),
106     //      .xadc_do(xadc_do),
107
108     //      .xadc_eos(xadc_eos),
109
110     //      // .vauxp4(xa_p[0]),
111     //      // .vauxn4(xa_n[0]),
112
113     //      // .vauxp12(xa_p[1]),
114     //      // .vauxn12(xa_n[1]),
115
116     //      .current_monitor_channel(current_monitor_channel.Source),
117     //      .voltage_channel(voltage_channel.Source)
118     // );
119
120
121     xadc_axis_ip xadc_teachee_inst (
122         // Clock and Reset
123         .dclk_in(sys_clk),          // input wire dclk_in
124         .reset_in(0),        // input wire reset_in
125
126         // DRP interface
127         .di_in(0),              // input wire [15 : 0] di_in
128         .daddr_in(xadc_daddr),        // input wire [6 : 0] daddr_in
129         .den_in(xadc_den),            // input wire den_in
130         .dwe_in(0),            // input wire dwe_in
131         .drdy_out(xadc_drdy),        // output wire drdy_out
132         .do_out(xadc_do),            // output wire [15 : 0] do_out
133
134         // Dedicated analog input channel (we do not use this)
135         .vp_in(0),              // input wire vp_in
136         .vn_in(0),              // input wire vn_in
137
138         // analog input channels, vaux4 is pin 15 = VIOUT of current
            ↪  sensor
139         // vaux12 is pin 16 = low speed voltage channel.
140         .vauxp4(xa_p[0]),            // input wire vauxp4
141         .vauxn4(xa_n[0]),            // input wire vauxn4
142         .vauxp12(xa_p[1]),          // input wire vauxp12
143         .vauxn12(xa_n[1]),          // input wire vauxn12
144
145         // conversion status signals
146         .channel_out(),  // output wire [4 : 0] channel_out
```

```
        .eoc_out(),           // output wire eoc_out
        .alarm_out(),       // output wire alarm_out
        .eos_out(xadc_eos),           // output wire eos_out
        .busy_out()         // output wire busy_out
    );

    cobs_axis_adapter_wrapper #(
                                .S_DATA_WIDTH(32),
                                .M_DATA_WIDTH(8)
    ) packetizer (
                    .original_data(voltage_channel.Sink),
                    .encoded_data(sys_axis.Source)
    );

    always_comb begin
        // prevent stalling of the current monitor fifo
        current_monitor_channel.tready = 1;
    end

endmodule

`default_nettype wire
```

Appendix E FPGA GitHub Actions Script

```
on:
  pull_request:
    branches: [ main ]

name: RTL

jobs:
  lint:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2

      - uses: teachee-capstone/svlint-action@v1.03
        with:
          files: |
            rtl/axis/*.sv
            rtl/ft232h/*.sv
            rtl/xadc/*.sv
            rtl/examples/*/*.sv
        env:
          SVLINT_CONFIG: rtl/.svlint.toml
```

```yaml

  # This job is a modified version of what is shown on this blog post:
  ↪ https://purisa.me/blog/testing-hdl-on-github/
  vunit:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
        with:
          fetch-depth: 0
          submodules: recursive
      # Grab a python env to use vunit
      - uses: actions/setup-python@v2
        with:
          python-version: '3.x'

      - name: Install vunit
        run: pip install vunit_hdl
        # ModelSim requires these 32-bit libraries to be installed:
        ↪ https://www.intel.com/content/www/us/en/programmable/support/support-re
        # Some of these are technically only required for the GUI, but
        ↪ it won't load on a headless server without them.
      - name: Install ModelSim dependencies
        run: |
          sudo dpkg --add-architecture i386
          sudo apt-get update
          sudo apt-get install lib32z1 lib32stdc++6 libexpat1:i386
          ↪ libc6:i386 libsm6:i386 libncurses5:i386 libx11-6:i386
          ↪ zlib1g:i386 libxext6:i386 libxft2:i386

      - name: Run Modelsim Installer
        run: |
          wget -O ModelSimSetup.run
          ↪ 'https://download.altera.com/akdlm/software/acdsinst/20.1std.1/720/ib
          chmod +x ModelSimSetup.run
          ./ModelSimSetup.run --mode unattended --accept_eula 1
          ls $HOME/intelFPGA/20.1/modelsim_ase
          sed -i 's/linux_rh60/linux/g'
          ↪ $HOME/intelFPGA/20.1/modelsim_ase/vco

      # Run all python files in the folder that end in _tb. As we add
      # testbenches, they will automatically run in this job
      - name: Run VUnit Test cases
        run: |
          export
          ↪ VUNIT_MODELSIM_PATH=$HOME/intelFPGA/20.1/modelsim_ase/bin
          for f in rtl/python/*_tb.py; do echo $f && python $f; done

```

Appendix F Python VUnit Testbenches

F.1 axis_adapter_wrapper_tb.py

```
"""
VUnit testbench for the axis bus width adapter.
"""

from pathlib import Path

import vunit_util

WORKSPACE = Path(__file__).parent / "workspace"
RTL_ROOT = Path(__file__).parent / ".."

# Create source list
sources = [
    RTL_ROOT / "testbenches" / "axis_adapter_wrapper_tb.sv",
    RTL_ROOT / "verilog-axis" / "rtl" / "axis_adapter.v",
    RTL_ROOT / "verilog-axis" / "rtl" / "axis_fifo.v",
    RTL_ROOT / "axis" / "*.sv"
]

# Create VUnit instance and add sources to library
vu, lib = vunit_util.init(WORKSPACE)
lib.add_source_files(sources)

# Create testbench
tb = lib.test_bench("axis_adapter_wrapper_tb")

# Suppress vopt deprecation error
vu.add_compile_option('modelsim.vlog_flags', ['-suppress', '12110'])

# Run
vu.main()
```

F.2 cobs_axis_adapter_wrapper_tb.py

```
"""
VUnit testbench for the cobs axis width adapter. This module, reduces
 ↪  the bus
width and then outputs the data cobs encoded per byte
"""

from pathlib import Path

import vunit_util

```

```
WORKSPACE = Path(__file__).parent / "workspace"
RTL_ROOT = Path(__file__).parent / ".."

# Create source list
sources = [
    RTL_ROOT / "testbenches" / "cobs_axis_adapter_wrapper_tb.sv",
    RTL_ROOT / "verilog-axis" / "rtl" / "axis_adapter.v",
    RTL_ROOT / "verilog-axis" / "rtl" / "axis_fifo.v",
    RTL_ROOT / "verilog-axis" / "rtl" / "axis_async_fifo.v",
    RTL_ROOT / "verilog-axis" / "rtl" / "axis_cobs_encode.v",
    RTL_ROOT / "axis" / "*.sv",
    RTL_ROOT / "cobs" / "*.sv"
]

# Create VUnit instance and add sources to library
vu, lib = vunit_util.init(WORKSPACE)
lib.add_source_files(sources)

# Create testbench
tb = lib.test_bench("cobs_axis_adapter_wrapper_tb")

# Suppress vopt deprecation error
vu.add_compile_option('modelsim.vlog_flags', ['-suppress', '12110'])

# Run
vu.main()
```

F.3 cobs_encode_wrapper_tb.py

```
"""
VUnit testbench for the cobs encoder wrapper.
"""

from pathlib import Path

import vunit_util

WORKSPACE = Path(__file__).parent / "workspace"
RTL_ROOT = Path(__file__).parent / ".."

# Create source list
sources = [
    RTL_ROOT / "testbenches" / "cobs_encode_wrapper_tb.sv",
    RTL_ROOT / "cobs" / "cobs_encode_wrapper.sv",
    RTL_ROOT / "verilog-axis" / "rtl" / "axis_cobs_encode.v",
    RTL_ROOT / "verilog-axis" / "rtl" / "axis_fifo.v",
    RTL_ROOT / "axis" / "axis_interface.sv"
]
```

```
20
21  # Create VUnit instance and add sources to library
22  vu, lib = vunit_util.init(WORKSPACE)
23  lib.add_source_files(sources)
24
25  # Create testbench
26  tb = lib.test_bench("cobs_encode_wrapper_tb")
27
28  # Suppress vopt deprecation error
29  vu.add_compile_option('modelsim.vlog_flags', ['-suppress', '12110'])
30
31  # Run
32  vu.main()
```

F.4 hsadc_axis_wrapper_tb.py

```
1   """
2   VUnit testbench for the High Speed ADC AXIS Adapter
3   """
4
5   from pathlib import Path
6
7   import vunit_util
8
9   WORKSPACE = Path(__file__).parent / "workspace"
10  RTL_ROOT = Path(__file__).parent / ".."
11
12  # Create source list
13  sources = [
14      RTL_ROOT / "testbenches" / "hsadc_axis_wrapper_tb.sv",
15      RTL_ROOT / "hsadc" / "*.sv",
16      RTL_ROOT / "axis" / "*.sv"
17  ]
18
19  # Create VUnit instance and add sources to library
20  vu, lib = vunit_util.init(WORKSPACE)
21  lib.add_source_files(sources)
22
23  # Create testbench
24  tb = lib.test_bench("hsadc_axis_wrapper_tb")
25
26  # Suppress vopt deprecation error
27  vu.add_compile_option('modelsim.vlog_flags', ['-suppress', '12110'])
28
29  # Run
30  vu.main()
31
```

F.5 xadc_packetizer_tb.py

```python
"""
VUnit testbench for the XADC COBS Packetizer.
TB uses the XADC BFM and AXIS Adapter to test the COBS packetizer
"""

from pathlib import Path

import vunit_util

WORKSPACE = Path(__file__).parent / "workspace"
RTL_ROOT = Path(__file__).parent / ".."

# Create source list
sources = [
    RTL_ROOT / "testbenches" / "xadc_packetizer_tb.sv",
    RTL_ROOT / "xadc" / "*.sv",
    RTL_ROOT / "cobs" / "cobs_encode_wrapper.sv",
    RTL_ROOT / "verilog-axis" / "rtl" / "axis_cobs_encode.v",
    RTL_ROOT / "verilog-axis" / "rtl" / "axis_fifo.v",
    RTL_ROOT / "verilog-axis" / "rtl" / "axis_async_fifo.v",
    RTL_ROOT / "axis" / "*.sv"
]

# Create VUnit instance and add sources to library
vu, lib = vunit_util.init(WORKSPACE)
lib.add_source_files(sources)

# Create testbench
tb = lib.test_bench("xadc_packetizer_tb")

# Suppress vopt deprecation error
vu.add_compile_option('modelsim.vlog_flags', ['-suppress', '12110'])

# Run
vu.main()
```

Appendix G Python Packet Integrity Test Script

```python
from pylibftdi import Device, Driver

print(Driver().list_devices())
user_in = input("exit now or start polling? y/n")
if user_in == 'y':
    exit()

with Device('FT84Y22T') as dev:
```

```
    # takes us from async to sync mode within ft245.
    # if you don't run this the clock does not come up
    dev.ftdi_fn.ftdi_set_bitmode(1, 0x40)
    data = dev.read(20_000)
    anomaly_counter = 0
    for i in range(1, len(data)):
        print(data[i])
        if data[i - 1] == 0 and data[i] > 5:
            anomaly_counter += 1

    print("found %d anomalies" % anomaly_counter)
```

Appendix H TeachEE Rust Code

H.1 main.rs

```
// hide console window on Windows in release
#![cfg_attr(not(debug_assertions), windows_subsystem = "windows")]

use std::thread;

use eframe::{self, NativeOptions, Theme};

use structopt::StructOpt;
use teachee_desktop::{
    app::App,
    controller::{AppData, Controller, USBData},
    sample_source::{FtSampleSource, Manager, SineSampleSource},
};

#[derive(Debug, StructOpt)]
struct Opt {
    /// Read fake sine data instead of a real TeachEE device
    #[structopt(short, long)]
    sine: bool,
}

fn main() {
    let options = NativeOptions {
        default_theme: Theme::Light,
        ..NativeOptions::default()
    };

    let opt = Opt::from_args();

    eframe::run_native(
        "TeachEE",
        options,
```

```
        Box::new(move |_cc| {
            let app_data = AppData::default();
            let controller_app_data = app_data.clone();

            let manager_data = USBData::default();
            let controller_manager_data = manager_data.clone();

            thread::Builder::new()
                .name("USB Manager".into())
                .spawn(move || {
                    if opt.sine {
                        ↪ Manager::<SineSampleSource>::manager_loop(manager_data)
                    } else {
                        ↪ Manager::<FtSampleSource>::manager_loop(manager_data)
                    }
                })
                .unwrap();

            thread::Builder::new()
                .name("Sample Controller".into())
                .spawn(move || {
                    Controller::new(controller_manager_data,
                     ↪ controller_app_data).controller_loop()
                })
                .unwrap();

            Box::new(App::new(app_data))
        }),
    );
}
```

H.2 /sample_source

H.2.1 mod.rs

```
use std::{marker::PhantomData, thread, time::Duration};

mod ft;
mod sine;

// flatten module structure
pub use ft::FtSampleSource;
pub use sine::SineSampleSource;

use crate::controller::{BufferState, Channels, USBData};

```

```rust
pub type Result<T> = std::result::Result<T, Box<dyn std::error::Error>>;

/// A trait which represents a source of samples. Could be a real TeachEE or a mock data.
pub trait SampleSource {
    /// Attempt to init sample source.
    fn try_init() -> Result<Self>
    where
        Self: Sized;

    /// Read samples into a pre-allocated buffer. Return the number of samples written and their
    /// channel.
    fn read_samples(&mut self, samples: &mut Channels) -> Result<usize>;
}

/// A generic manager which reads from a given `SampleSource` into shared buffers.
pub struct Manager<T>
where
    T: SampleSource,
{
    data: USBData,
    phantom: PhantomData<T>,
}

impl<T> Manager<T>
where
    T: SampleSource,
{
    /// Infinite loop which continually attempts to initialize the `SampleSource`.
    pub fn manager_loop(data: USBData) {
        let mut manager = Self {
            data,
            phantom: PhantomData,
        };

        loop {
            // sleep so we don't waste CPU cycles reinitializing a device which might not even be
            // plugged in.
            thread::sleep(Duration::from_secs(1));

            match T::try_init() {
                Ok(reader) => manager.read_samples_loop(reader),
```

```
                Err(error) => eprintln!("{error:?}"),
            }
        }
    }

    /// Inner loop which continually reads the `SampleSource` until an error occurs.
    fn read_samples_loop(&mut self, mut reader: T) {
        // TODO: set connection status flag on self.storage = true

        loop {
            for buf in self.data.bufs.iter() {
                let (condvar, mutex) = &**buf;

                let mut buf_state = condvar
                    .wait_while(mutex.lock().unwrap(), |buf_state| buf_state.is_full())
                    .unwrap();

                let (mut channels, _) = buf_state.unwrap();
                match reader.read_samples(&mut channels) {
                    Ok(num_samples) => {
                        // Tell Controller that this buffer is full, and wake them up if waiting.
                        *buf_state = BufferState::Full(channels, num_samples);
                        condvar.notify_one();
                    }
                    Err(error) => {
                        eprintln!("{error:?}");
                        // TODO: set connection status flag on self.storage = false
                        return;
                    }
                }
            }
        }
    }
}
```

H.2.2 ft.rs

```
use std::{thread, time::Duration};

use libftd2xx::{BitMode, Ft232h, FtdiCommon};

use crate::controller::{Channels, BUF_SIZE};

```

```rust
use super::{Result, SampleSource};

const MASK: u8 = 0xFF;
const LATENCY_TIMER: Duration = Duration::from_millis(16);
const IN_TRANSFER_SIZE: u32 = 0x10000;
const PACKET_SIZE: usize = 6;
const RX_BUF_SIZE: usize = BUF_SIZE * PACKET_SIZE;

/// A sample source that reads from a real TeachEE device.
pub struct FtSampleSource {
    ft: Ft232h,
    rx_buf: Vec<u8>,
}

impl SampleSource for FtSampleSource {
    fn try_init() -> Result<Self> {
        // See:
        ↪ https://ftdichip.com/wp-content/uploads/2020/08/AN_130_FT2232H_Used_In_
        let mut ft = Ft232h::with_description("TeachEE")?;
        ft.set_bit_mode(MASK, BitMode::Reset)?;
        thread::sleep(Duration::from_millis(10));

        ft.set_bit_mode(MASK, BitMode::SyncFifo)?;
        ft.set_latency_timer(LATENCY_TIMER)?;
        ft.set_usb_parameters(IN_TRANSFER_SIZE)?;
        ft.set_flow_control_rts_cts()?;
        ft.purge_rx()?;

        thread::sleep(Duration::from_millis(1000));

        Ok(Self {
            ft,
            rx_buf: vec![0; RX_BUF_SIZE],
        })
    }

    fn read_samples(&mut self, channels: &mut Channels) ->
    ↪ Result<usize> {
        let num_bytes = self.read_bytes()?;
        let num_samples = self.decode_and_copy(channels, num_bytes);
        Ok(num_samples)
    }
}

impl FtSampleSource {
    fn read_bytes(&mut self) -> Result<usize> {
        self.ft.read_all(&mut self.rx_buf)?;
        Ok(RX_BUF_SIZE)
```

```
53      }
54      fn decode_and_copy(&mut self, channels: &mut Channels, num_bytes:
        ↪  usize) -> usize {
55          // Position after first 0
56          let start = self.rx_buf[..num_bytes]
57              .iter()
58              .position(|&x| x == 0)
59              .unwrap()
60              + 1;
61          // Position after last 0
62          let end = num_bytes
63              - self.rx_buf[..num_bytes]
64                  .iter()
65                  .rev()
66                  .position(|&x| x == 0)
67                  .unwrap();
68          let mut src_index = start;
69          let mut dst_index = 0;
70
71          'outer: while src_index + PACKET_SIZE < end && dst_index <
            ↪  channels.voltage1.len() {
72              // Packet error: offset byte not in [1, 5] or packet not
                ↪  delimited with 0
73              if self.rx_buf[src_index] == 0
74                  || self.rx_buf[src_index] >= PACKET_SIZE as u8
75                  || self.rx_buf[src_index + PACKET_SIZE - 1] != 0
76              {
77                  src_index += 1;
78                  // Find the next 0
79                  match self.rx_buf[src_index..(end - 1)]
80                      .iter()
81                      .position(|&x| x == 0)
82                  {
83                      Some(i) => {
84                          // Advance to the next packet (which follows
                            ↪  the 0)
85                          src_index += i + 1;
86                      }
87                      None => {
88                          // No more valid packets or the erroneous
                            ↪  packet was the last
89                          break;
90                      }
91                  }
92              } else {
93                  let packet = &self.rx_buf[src_index..(src_index +
                    ↪  PACKET_SIZE)];
94                  let mut block = packet[0] - 1;
```

```
                    // Two bytes of packet overhead
                    let mut decoded: [u8; PACKET_SIZE - 2] = [0;
                    ↪  PACKET_SIZE - 2];
                    let mut decoded_index = 0;

                    // Note: the packet index is just the decoded_index + 1
                    ↪  (skip the first offset byte)
                    // Example packet: 01 02 22 02 44 00
                    // Decodes to:        00 22 00 44
                    while decoded_index < decoded.len() {
                        if block > 0 {
                            decoded[decoded_index] = packet[decoded_index +
                            ↪  1];
                        } else {
                            decoded[decoded_index] = 0;
                            block = packet[decoded_index + 1];

                            if block == 0 {
                                // The 4 sample bytes cannot be zero; skip
                                ↪  this packet
                                src_index += PACKET_SIZE;
                                continue 'outer;
                            }
                        }
                        decoded_index += 1;
                        block -= 1;
                    }

                    channels.voltage1[dst_index] =
                        (((decoded[3] as u16) << 4) | decoded[2] as u16) as
                        ↪  f64 * 3.3 / 4095.0;
                    channels.current1[dst_index] =
                        ((((decoded[1] as u16) << 4) | decoded[0] as u16)
                        ↪  as f64 * 3.3 / 4095.0 - 1.5)
                            * (1.0 / 0.09);
                    src_index += PACKET_SIZE;
                    dst_index += 1;
                }
            }

            dst_index
        }
    }
```

H.2.3 sine.rs

```
use std::{
    cmp::min,
```

```rust
    thread,
    time::{Duration, Instant},
};

use crate::controller::Channels;

use super::{Result, SampleSource};

const SAMPLE_DELAY_SEC: f64 = 1.0 / 1_000.0;
const CHUNK_DELAY_SEC: f64 = 1.0 / 100.0;

const SINE_PERIOD_SEC: f64 = 0.1;
const SINE_AMPLITUDE: f64 = 100.0;

/// A sample source that outputs a sine wave on one channel
pub struct SineSampleSource {
    start: Instant,
    last_read: Instant,
}

impl SampleSource for SineSampleSource {
    fn try_init() -> Result<Self> {
        let now = Instant::now();
        Ok(Self {
            start: now,
            last_read: now,
        })
    }

    fn read_samples(&mut self, channels: &mut Channels) ->
    ↪  Result<usize> {
        // sleep to emulate blocking read
        thread::sleep(Duration::from_secs_f64(CHUNK_DELAY_SEC));

        let now = Instant::now();
        let num_samples = min(
            ((now - self.last_read).as_secs_f64() / SAMPLE_DELAY_SEC)
            ↪  as usize,
            channels.voltage1.len(),
        );
        let mut t = (now - self.start).as_secs_f64() % SINE_PERIOD_SEC;

        for (v_sample, c_sample) in channels
            .voltage1
            .iter_mut()
            .zip(channels.current1.iter_mut())
            .take(num_samples)
        {
```

```
            t = (t + SAMPLE_DELAY_SEC) % SINE_PERIOD_SEC;
            *v_sample = SINE_AMPLITUDE * (t * SINE_PERIOD_SEC).sin();
            *c_sample = *v_sample;
        }

        Ok(num_samples)
    }
}
```

H.3 controller.rs

```
use std::{
    cmp::min,
    mem::take,
    sync::{Arc, Condvar, Mutex},
};

use spectrum_analyzer::{
    samples_fft_to_spectrum, scaling::divide_by_N_sqrt, FrequencyLimit,
    ↪ FrequencySpectrum,
};

pub const SAMPLE_RATE_PER_CHANNEL: usize = 500_000;
const MAX_FFT_SAMPLES_INPUT: usize = 0x4000;

// Number of samples in each channel's buffer
pub const BUF_SIZE: usize = 20000;
const NUM_BUFS: usize = 2;

#[derive(Debug)]
pub struct Channels {
    pub voltage1: Vec<f64>,
    pub current1: Vec<f64>,
    pub fft1: FrequencySpectrum,
}

impl Default for Channels {
    fn default() -> Self {
        Self {
            voltage1: vec![0.0; BUF_SIZE],
            current1: vec![0.0; BUF_SIZE],
            fft1: FrequencySpectrum::default(),
        }
    }
}

#[derive(Debug)]
pub enum BufferState {
```

```
37      Full(Channels, usize),
38      Empty(Channels),
39  }
40
41  impl BufferState {
42      pub fn is_empty(&self) -> bool {
43          match *self {
44              BufferState::Empty(_) => true,
45              BufferState::Full(..) => false,
46          }
47      }
48
49      pub fn is_full(&self) -> bool {
50          match *self {
51              BufferState::Empty(_) => false,
52              BufferState::Full(..) => true,
53          }
54      }
55
56      /// Fetch the Channels from the enum. take() is used to take
        ↪  ownership
57      /// by swapping with an empty Channels.
58      pub fn unwrap(&mut self) -> (Channels, usize) {
59          match self {
60              BufferState::Empty(c) => (take(&mut *c), 0),
61              BufferState::Full(c, num_samples) => (take(&mut *c),
                ↪  *num_samples),
62          }
63      }
64  }
65
66  /// Represents two buffers that you can swap accesses between.
67  #[derive(Debug, Clone)]
68  pub struct USBData {
69      pub bufs: Vec<Arc<(Condvar, Mutex<BufferState>)>>,
70  }
71
72  #[derive(Debug, Clone)]
73  pub struct AppData {
74      pub bufs: Vec<Arc<(Condvar, Mutex<BufferState>)>>,
75      pub voltage_trigger_threshold: Arc<Mutex<f64>>,
76      pub current_trigger_threshold: Arc<Mutex<f64>>,
77      pub fft: Arc<Mutex<bool>>,
78  }
79
80  fn generate_buffers() -> Vec<Arc<(Condvar, Mutex<BufferState>)>> {
81      let mut v = Vec::with_capacity(NUM_BUFS);
82      (0..NUM_BUFS).for_each(|_| {
```

```
            v.push(Arc::new((
                Condvar::new(),
                Mutex::new(BufferState::Empty(Channels::default())),
            )))
        });
        v
    }

    impl Default for USBData {
        fn default() -> Self {
            Self {
                bufs: generate_buffers(),
            }
        }
    }

    impl Default for AppData {
        fn default() -> Self {
            Self {
                bufs: generate_buffers(),
                voltage_trigger_threshold: Arc::new(Mutex::new(0.0)),
                current_trigger_threshold: Arc::new(Mutex::new(0.0)),
                fft: Arc::new(Mutex::new(false)),
            }
        }
    }

    /// Controller class copies from USB data buffer 1 into App data buffer
     ↪  1,
    /// then USB data buffer 2 into App data buffer 2.
    pub struct Controller {
        usb_data: USBData,
        app_data: AppData,
    }

    impl Controller {
        pub fn new(usb_data: USBData, app_data: AppData) -> Self {
            Self { usb_data, app_data }
        }
        pub fn controller_loop(&mut self) {
            loop {
                // Cycle between the two buffers in each pair.
                for (dbuf, abuf) in
                 ↪  self.usb_data.bufs.iter().zip(self.app_data.bufs.iter())
                 ↪  {
                    let (data_condvar, data_mutex) = &**dbuf;
                    let (app_condvar, app_mutex) = &**abuf;

```

```
                // If the current USB buffer is empty, release lock, wait until USB thread has filled it.
                // When this function returns the lock will have been reacquired.
                let mut data_state = data_condvar
                    .wait_while(data_mutex.lock().unwrap(), |data_state| {
                        data_state.is_empty()
                    })
                    .unwrap();

                // Wait if app buffer is full.
                let mut app_state = app_condvar
                    .wait_while(app_mutex.lock().unwrap(), |app_state| app_state.is_full())
                    .unwrap();

                let (mut dst, _) = app_state.unwrap();
                let (src, num_samples) = data_state.unwrap();
                let num_remaining = Self::copy_channels(
                    &mut dst,
                    &src,
                    num_samples,

                    *self.app_data.voltage_trigger_threshold.lock().unwrap(),

                    *self.app_data.current_trigger_threshold.lock().unwrap(),
                );

                if *self.app_data.fft.lock().unwrap() {
                    let mut temp: [f32; MAX_FFT_SAMPLES_INPUT] = [0.0; MAX_FFT_SAMPLES_INPUT];
                    for i in 0..min(temp.len(), num_remaining) {
                        temp[i] = dst.voltage1[i] as f32;
                    }

                    dst.fft1 = samples_fft_to_spectrum(
                        &temp[..min(temp.len(), num_remaining.next_power_of_two())],
                        SAMPLE_RATE_PER_CHANNEL as u32,
                        FrequencyLimit::Range(65.0, 100_000.0),
                        Some(&divide_by_N_sqrt),
                    )
                    .unwrap();
                }

                *data_state = BufferState::Empty(src);
                *app_state = BufferState::Full(dst, num_remaining);
```

```
168
169                 // Tell USB thread that the current data buffer is now
                    ↪  empty.
170                 data_condvar.notify_one();
171                 // Tell App thread that the current app buffer is now
                    ↪  full.
172                 app_condvar.notify_one();
173             }
174         }
175     }
176
177     fn copy_channels(
178         dst: &mut Channels,
179         src: &Channels,
180         num_samples: usize,
181         v_trigger: f64,
182         c_trigger: f64,
183     ) -> usize {
184         // Return the least number of samples between the two channels
            ↪  (discard some from the channel with more)
185         min(
186             Self::copy_with_trigger(&mut dst.voltage1, &src.voltage1,
                ↪  num_samples, v_trigger),
187             Self::copy_with_trigger(&mut dst.current1, &src.current1,
                ↪  num_samples, c_trigger),
188         )
189     }
190
191     fn copy_with_trigger(dst: &mut [f64], src: &[f64], num_samples:
    ↪  usize, trigger: f64) -> usize {
192         if trigger > -15.0 && trigger < 15.0 {
193             let first_sample: usize;
194             #[cfg(feature = "window_trigger")]
195             {
196                 first_sample =
                    ↪  Self::window_trigger(&src[..num_samples], trigger);
197             }
198             #[cfg(not(feature = "window_trigger"))]
199             {
200                 first_sample =
                    ↪  Self::hysteresis_trigger(&src[..num_samples],
                    ↪  trigger);
201             }
202
203             dst[..(num_samples -
                ↪  first_sample)].copy_from_slice(&src[first_sample..num_samples]);
204
205             return num_samples - first_sample;
```

```rust
        }

        // If no trigger, just copy the available samples without triggering.
        dst[..num_samples].copy_from_slice(&src[..num_samples]);
        num_samples
    }

    #[cfg(feature = "window_trigger")]
    fn window_trigger(src: &[f64], trigger: f64) -> usize {
        let first_lower = src.iter().position(|&val| val < trigger);
        if let Some(lower_idx) = first_lower {
            const WINDOW_SIZE: usize = 10;
            let mut sum: f64 = src[lower_idx..(lower_idx + WINDOW_SIZE)].iter().sum();

            // Iterate over all windows of size WINDOW_SIZE + 1. The first WINDOW_SIZE samples
            // are used to calculate the average value of previous samples, which is then
            // compared to the last sample. This is done to find a rising trigger
            // while attempting to filter out noise.
            let first_higher = src[lower_idx..]
                .windows(WINDOW_SIZE + 1)
                .position(|window| {
                    let val = *window.last().unwrap();
                    val >= trigger && sum / (WINDOW_SIZE as f64) < val || {
                        // Update sliding window.
                        sum += val;
                        sum -= *window.first().unwrap();
                        false
                    }
                });
            if let Some(higher_idx) = first_higher {
                // higher_idx is relative to lower_idx. Also add WINDOW_SIZE to get the
                // last element in the window.
                return lower_idx + higher_idx + WINDOW_SIZE;
            }
        }

        // If trigger failed, copy without trigger
        0
    }

    #[cfg(not(feature = "window_trigger"))]
```

```rust
    fn hysteresis_trigger(src: &[f64], trigger: f64) -> usize {
        // Hysteresis width as fraction of peak to peak
        const HYSTERESIS_WIDTH: f64 = 0.25;
        let mut max_val = f64::MIN;
        let mut min_val = f64::MAX;
        for &val in src.iter() {
            max_val = f64::max(max_val, val);
            min_val = f64::min(min_val, val);
        }
        let hysteresis = (max_val - min_val) * HYSTERESIS_WIDTH;

        // Trigger when signal first falls below trigger - hysteresis
        ↪ and then rises above trigger
        let first_lower = src.iter().position(|&val| val < trigger -
        ↪ hysteresis);
        if let Some(lower_idx) = first_lower {
            let first_higher = src[lower_idx..].iter().position(|&val|
            ↪ val >= trigger);
            if let Some(higher_idx) = first_higher {
                return lower_idx + higher_idx;
            }
        }

        0
    }
}
```

H.4 app.rs

```rust
use std::{
    error::Error,
    fmt,
    fs::{remove_file, File},
    ops::RangeInclusive,
    sync::{Arc, Mutex},
    time::{SystemTime, UNIX_EPOCH},
};

use csv::Writer;
use eframe::egui::*;

use crate::controller::{AppData, BufferState, SAMPLE_RATE_PER_CHANNEL};

#[derive(Debug, Default)]
enum TriggerControl {
    #[default]
    Start,
    Stop,
```

```rust
}

// Allows enum to be formatted to string
impl fmt::Display for TriggerControl {
    fn fmt(&self, f: &mut fmt::Formatter<'_>) -> fmt::Result {
        fmt::Debug::fmt(self, f)
    }
}

const GROUP_SPACING: f32 = 3.0;
const BUTTON_HEIGHT: f32 = 25.0;
const ERROR_COLOUR: Color32 = Color32::LIGHT_RED;

// 0.5 MSPS
const SAMPLE_PERIOD: f64 = 1.0 / (SAMPLE_RATE_PER_CHANNEL as f64);

const H_SCALE_RANGE: RangeInclusive<f64> = RangeInclusive::new(0.1,
↪  10.0);
const H_OFFSET_RANGE: RangeInclusive<f64> =
    RangeInclusive::new(-10000.0 * SAMPLE_PERIOD, 10000.0 *
    ↪  SAMPLE_PERIOD);
const V_SCALE_RANGE: RangeInclusive<f64> = RangeInclusive::new(0.1,
↪  10.0);
const V_OFFSET_RANGE: RangeInclusive<f64> = RangeInclusive::new(-5.0,
↪  5.0);

#[derive(Debug)]
struct UIControls {
    h_offset: f64,
    h_scale: f64,
    channel1_v_offset: f64,
    channel1_v_scale: f64,
    channel2_v_offset: f64,
    channel2_v_scale: f64,
    v_trigger_button_text: TriggerControl,
    v_trigger_threshold_text: String,
    v_trigger_format_wrong: bool,
    c_trigger_button_text: TriggerControl,
    c_trigger_threshold_text: String,
    c_trigger_format_wrong: bool,
    export_error_string: String,
    fft: bool,
    fft_dom_freq: f32,
}

impl Default for UIControls {
    fn default() -> Self {
        Self {
```

```
            h_offset: 0.0,
            h_scale: 1.0,
            channel1_v_offset: 0.0,
            channel1_v_scale: 1.0,
            channel2_v_offset: 0.0,
            channel2_v_scale: 1.0,
            v_trigger_button_text: TriggerControl::default(),
            v_trigger_threshold_text: String::new(),
            v_trigger_format_wrong: false,
            c_trigger_button_text: TriggerControl::default(),
            c_trigger_threshold_text: String::new(),
            c_trigger_format_wrong: false,
            export_error_string: String::new(),
            fft: false,
            fft_dom_freq: 0.0,
        }
    }
}

#[derive(Debug)]
pub struct App {
    data: AppData,
    buf_idx: usize,
    ui_controls: UIControls,
}

impl App {
    pub fn new(data: AppData) -> Self {
        Self {
            data,
            buf_idx: 0,
            ui_controls: UIControls::default(),
        }
    }
}

fn update_trigger(
    ui: &mut Ui,
    trigger_val: &mut Arc<Mutex<f64>>,
    button_text: &mut TriggerControl,
    textedit_text: &mut String,
    format_wrong: &mut bool,
    (offset, scale): (f64, f64),
    hint_text: &str,
) {
    ui.with_layout(
        Layout::top_down(Align::Center).with_cross_align(Align::Min),
        |ui| {
```

```
            if *format_wrong {
                ui.style_mut().visuals.extreme_bg_color = ERROR_COLOUR;
            }
            let re = ui.add(
                TextEdit::singleline(textedit_text)
                    .hint_text(hint_text)
                    .desired_width(f32::INFINITY),
            );
            if re.lost_focus() &&
             ↪  re.ctx.input().key_pressed(Key::Enter) {
                let parsed = textedit_text.parse::<f64>();
                match parsed {
                    Ok(new_value) => {
                        *trigger_val.lock().unwrap() = new_value;
                        *button_text = TriggerControl::Stop;
                        *format_wrong = false;
                    }
                    Err(_) => {
                        *format_wrong = true;
                    }
                }
            }
        },
    );
    if ui
        .add(Button::new(button_text.to_string()).min_size((0.0,
        ↪  BUTTON_HEIGHT).into()))
        .clicked()
    {
        use TriggerControl::*;
        match button_text {
            Start => {
                let parsed = textedit_text.parse::<f64>();
                match parsed {
                    Ok(new_value) => {
                        *trigger_val.lock().unwrap() = (new_value -
                         ↪  offset) / scale;
                        *button_text = Stop;
                        *format_wrong = false;
                    }
                    Err(_) => {
                        *format_wrong = true;
                    }
                }
            }
            Stop => {
                *trigger_val.lock().unwrap() = f64::INFINITY;
                *button_text = Start;
```

```
157                 }
158             };
159         }
160 }
161
162 fn offset_scale_sliders(
163     ui: &mut Ui,
164     offset: &mut f64,
165     o_range: RangeInclusive<f64>,
166     scale: &mut f64,
167     s_range: RangeInclusive<f64>,
168 ) {
169     ui.columns(2, |uis| {
170         uis[0].label("Offset");
171         uis[0].add(Slider::new(offset, o_range).clamp_to_range(false));
172         uis[1].label("Scale");
173         uis[1].add(
174             Slider::new(scale, s_range)
175                 .clamp_to_range(false)
176                 .logarithmic(true),
177         );
178     });
179 }
180
181 impl eframe::App for App {
182     fn update(&mut self, ctx: &Context, frame: &mut eframe::Frame) {
183         // Always redraw on the next frame. Ensures that state changes
            ↪  from
184         // other threads are immediately reflected in the UI.
185         ctx.request_repaint();
186
187         let Self {
188             data,
189             buf_idx,
190             ui_controls,
191             ..
192         } = self;
193
194         TopBottomPanel::top("top").show(ctx, |ui| {
195             ui.horizontal_wrapped(|ui| {
196                 ui.visuals_mut().button_frame = false;
197                 widgets::global_dark_light_mode_switch(ui);
198                 ui.separator();
199                 ui.menu_button("File", |ui| {
200                     let mut export_button = Button::new("Export to
                        ↪  CSV");
201                     if !ui_controls.export_error_string.is_empty() {
```

```
202                         export_button =
                             ↪ export_button.fill(ERROR_COLOUR);
203                     }
204                     let mut re = ui.add(export_button);
205                     if !ui_controls.export_error_string.is_empty() {
206                         re =
                             ↪ re.on_hover_text(&ui_controls.export_error_string);
207                     }
208
209                     if re.clicked() {
210                         let epoch_time = SystemTime::now()
211                             .duration_since(UNIX_EPOCH)
212                             .unwrap()
213                             .as_secs();
214                         let pid = std::process::id();
215                         let file_name = format!("{}-{}-{}", epoch_time,
                             ↪ pid, "teachee.csv");
216                         let file = match
                             ↪ File::create(file_name.clone()) {
217                             Ok(a) => a,
218                             Err(_) => {
219                                 ui_controls.export_error_string =
220                                     format!("Unable to open file: {}",
                                         ↪ file_name);
221                                 return;
222                             }
223                         };
224                         let mut writer = Writer::from_writer(file);
225                         let (condvar, mutex) = &*data.bufs[*buf_idx];
226                         let mut buf_state = condvar
227                             .wait_while(mutex.lock().unwrap(),
                                 ↪ |buf_state| buf_state.is_empty())
228                             .unwrap();
229                         let (channels, num_samples) =
                             ↪ buf_state.unwrap();
230
231                         let v_strs = channels.voltage1[0..num_samples]
232                             .iter()
233                             .map(|e| e.to_string());
234
235                         let c_strs = channels.current1[0..num_samples]
236                             .iter()
237                             .map(|e| e.to_string());
238                         let do_writing = move || -> Result<(), Box<dyn
                             ↪ Error>> {
239                             writer.write_field("Channel1")?;
240                             writer.write_record(v_strs)?;
241                             writer.write_field("Channel2")?;
```

```
                            writer.write_record(c_strs)?;
                            Ok(())
                        };

                        if let Err(e) = do_writing() {
                            ui_controls.export_error_string =
                                format!("Error writing csv output:
                                 ↪  {e}");
                            _ = remove_file(file_name);
                            return;
                        }

                        ui_controls.export_error_string.clear();
                    }

                    if ui.button("Exit").clicked() {
                        frame.close();
                    }
                });
            })
        });

        SidePanel::right("controls")
            .resizable(false)
            .show(ctx, |ui| {
                ui.vertical_centered(|ui| {
                    ui.add_space(GROUP_SPACING);

                    ui.group(|ui| {
                        ui.add_space(GROUP_SPACING);
                        ui.label(RichText::new("Channel
                         ↪  scaling").size(14.0));


                         ↪  ui.add(Separator::default().spacing(GROUP_SPACING
                         ↪  * 2.0));

                        ui.label("Horizontal");
                        ui.add_space(GROUP_SPACING);
                        offset_scale_sliders(
                            ui,
                            &mut ui_controls.h_offset,
                            H_OFFSET_RANGE,
                            &mut ui_controls.h_scale,
                            H_SCALE_RANGE,
                        );
```

```
285
                    ↪ ui.add(Separator::default().spacing(GROUP_SPACING
                    ↪ * 2.0));
286
287                 ui.label("Channel 1 Vertical");
288                 ui.add_space(GROUP_SPACING);
289                 offset_scale_sliders(
290                     ui,
291                     &mut ui_controls.channel1_v_offset,
292                     V_OFFSET_RANGE,
293                     &mut ui_controls.channel1_v_scale,
294                     V_SCALE_RANGE,
295                 );
296
297                 ui.add_space(GROUP_SPACING);
298
299                 ui.label("Channel 2 Vertical");
300                 ui.add_space(GROUP_SPACING);
301                 offset_scale_sliders(
302                     ui,
303                     &mut ui_controls.channel2_v_offset,
304                     V_OFFSET_RANGE,
305                     &mut ui_controls.channel2_v_scale,
306                     V_SCALE_RANGE,
307                 );
308             });
309
310             ui.add_space(GROUP_SPACING);
311
312             ui.group(|ui| {
313                 ui.add_space(GROUP_SPACING);
314                 ui.label(RichText::new("Triggers").size(14.0));
315
316
                    ↪ ui.add(Separator::default().spacing(GROUP_SPACING
                    ↪ * 2.0));
317
318                 ui.vertical_centered_justified(|ui| {
319                     update_trigger(
320                         ui,
321                         &mut data.voltage_trigger_threshold,
322                         &mut ui_controls.v_trigger_button_text,
323                         &mut
                            ↪ ui_controls.v_trigger_threshold_text,
324                         &mut
                            ↪ ui_controls.v_trigger_format_wrong,
325                         (ui_controls.channel1_v_offset,
                            ↪ ui_controls.channel1_v_scale),
```

```
                                "Channel 1",
                            );
                            update_trigger(
                                ui,
                                &mut data.current_trigger_threshold,
                                &mut ui_controls.c_trigger_button_text,
                                &mut ui_controls.c_trigger_threshold_text,
                                &mut ui_controls.c_trigger_format_wrong,
                                (ui_controls.channel2_v_offset, ui_controls.channel2_v_scale),
                                "Channel 2",
                            );
                        });
                    });

                    ui.add_space(GROUP_SPACING);

                    ui.group(|ui| {
                        ui.add_space(GROUP_SPACING);
                        ui.label(RichText::new("FFT").size(14.0));

                        ui.add(Separator::default().spacing(GROUP_SPACING * 2.0));

                        ui.vertical_centered_justified(|ui| {
                            if ui.toggle_value(&mut ui_controls.fft, "Channel 1").clicked() {
                                *data.fft.lock().unwrap() = ui_controls.fft;
                            }
                        });

                        ui.add_space(GROUP_SPACING);

                        let mut dominant_freq_str = "Dominant Frequency: ".to_string();
                        if ui_controls.fft {

                            dominant_freq_str.push_str(&ui_controls.fft_dom_fre
                        }
                        ui.label(dominant_freq_str);
                    });

                    ui.add_space(GROUP_SPACING);

```

```
365                     ui.group(|ui| {
366                         ui.add_space(GROUP_SPACING);
367                         ui.label(RichText::new("Reset All
                            ↪  Configurations").size(14.0));
368 
369 
                            ↪  ui.add(Separator::default().spacing(GROUP_SPACING
                            ↪  * 2.0));
370 
371                         ui.vertical_centered_justified(|ui| {
372                             if ui
373 
                                    ↪  .add(Button::new("Reset").min_size((0.0,
                                    ↪  BUTTON_HEIGHT).into()))
374                                 .clicked()
375                             {
376                                 *ui_controls = UIControls::default();
377 
                                    ↪  *data.voltage_trigger_threshold.lock().unwrap()
                                    ↪  = f64::INFINITY;
378 
                                    ↪  *data.current_trigger_threshold.lock().unwrap()
                                    ↪  = f64::INFINITY;
379                                 *data.fft.lock().unwrap() = false;
380                             }
381                         });
382                     })
383                 });
384             });
385 
386         CentralPanel::default().show(ctx, |ui| {
387             // Get the current buffer. Alternating between buffers is
                ↪  done by toggling
388             // the buf_idx.
389             let (condvar, mutex) = &*data.bufs[*buf_idx];
390 
391             // Wait until controller thread has filled the current
                ↪  buffer.
392             let mut buf_state = condvar
393                 .wait_while(mutex.lock().unwrap(), |buf_state|
                    ↪  buf_state.is_empty())
394                 .unwrap();
395 
396             let (channels, num_samples) = buf_state.unwrap();
397 
398             // Mapping i -> t using the fixed sample rate to get point
                ↪  (i * period, samples[i]).
```

```
399             let voltage =
                ↪  plot::Line::new(plot::PlotPoints::from_parametric_callback(
400                 |i| {
401                     (
402                         i * SAMPLE_PERIOD * ui_controls.h_scale +
                            ↪  ui_controls.h_offset,
403                         channels.voltage1[i as usize] *
                            ↪  ui_controls.channel1_v_scale
404                             + ui_controls.channel1_v_offset,
405                     )
406                 },
407                 0.0..(num_samples as f64),
408                 num_samples,
409             ))
410             .name("Channel 1");
411
412             let current =
                ↪  plot::Line::new(plot::PlotPoints::from_parametric_callback(
413                 |i| {
414                     (
415                         i * SAMPLE_PERIOD * ui_controls.h_scale +
                            ↪  ui_controls.h_offset,
416                         channels.current1[i as usize] *
                            ↪  ui_controls.channel2_v_scale
417                             + ui_controls.channel2_v_offset,
418                     )
419                 },
420                 0.0..(num_samples as f64),
421                 num_samples,
422             ))
423             .name("Channel 2");
424
425             let fft_line = if ui_controls.fft {
426                 // Update dominant frequency while we hold the monitor
427                 ui_controls.fft_dom_freq = channels.fft1.max().0.val();
428                 // Convert array of (Freq, FreqVal) tuples to (f64,
                    ↪  f64) points
429
                    ↪  plot::Line::new(plot::PlotPoints::from_parametric_callback(
430                     |i| {
431                         (
432                             channels.fft1.data()[i as usize].0.val() as
                                ↪  f64,
433                             channels.fft1.data()[i as usize].1.val() as
                                ↪  f64,
434                         )
435                     },
436                     0.0..(channels.fft1.data().len() as f64),
```

```
                        channels.fft1.data().len(),
                    ))
                    .name("Channel 1 FFT")
            } else {
                plot::Line::new(plot::PlotPoints::default())
            };

            // Next update, use the other buffer.
            *buf_idx ^= 0x1;
            *buf_state = BufferState::Empty(channels);
            condvar.notify_one();
            drop(buf_state);

            let plot_height = (frame.info().window_info.size.y - 42.0) / 2.0;
            let mut time_plot = plot::Plot::new("plot").legend(plot::Legend::default());
            if ui_controls.fft {
                time_plot = time_plot.height(plot_height);
            }

            time_plot.show(ui, |ui| {
                ui.line(voltage);
                ui.line(current);
            });

            if ui_controls.fft {
                plot::Plot::new("fft")
                    .height(plot_height)
                    .legend(plot::Legend::default())
                    .show(ui, |ui| {
                        ui.line(fft_line);
                    });
            }
        });
    }
}
```

H.5 xtask: main.rs

```
use std::process;

use structopt::StructOpt;
use xshell::{cmd, Shell};

type Result<T> = std::result::Result<T, Box<dyn std::error::Error>>;

#[derive(Debug, StructOpt)]
```

```rust
enum Opt {
    #[structopt(about = "Runs CI checks")]
    Ci,
}

fn main() {
    if let Err(err) = try_main() {
        eprintln!("Error: {err:?}");
        process::exit(1);
    }
}

fn try_main() -> Result<()> {
    match Opt::from_args() {
        Opt::Ci => ci(),
    }
}

fn ci() -> Result<()> {
    let sh = Shell::new()?;
    cmd!(sh, "cargo clippy --workspace -- -D warnings").run()?;
    cmd!(sh, "cargo fmt -- --check").run()?;
    cmd!(sh, "cargo test --workspace").run()?;
    Ok(())
}
```