DSP Platform Firmware and Software Redesign

Design Document

Team 47 - Fall 2022
Client: Matthew Post
Advisor: Dr. Phillip Jones

Yohan Bopearatchy - Firmware Engineer
Wyatt Duberstein - CLI Engineer
Blake Fisher - GUI Lead
Corbin Kems - CLI Engineer Lead
Cole Langner - Testing Lead
Jens Rasmussen - Project Lead
Long Zeng - Firmware Engineer Lead

Team Email: sdmay23-47@iastate.edu
Team Website: https://sdmay23-47.sd.ece.iastate.edu

Executive Summary

Development Standards & Practices Used

- IEEE 610.2-1987: Standard Glossary of Computer Applications Terminology
- IEEE 610.6-1991: Standard Glossary of Computer Graphics Terminology
- IEEE 610.10-1994: Standard Glossary of Computer Hardware Terminology
- IEEE 610.13-1993: Standard Glossary of Computer Languages
- IEEE/EIA 12207: Software Life Cycle Processes
- IEEE/IEC 29119-2: Test Processes
- Universal Asynchronous Receiver-Transmitter (UART)
- Universal Serial Bus (USB) 2.0

Summary of Requirements

- Easy-to-use graphical user interface (GUI).
- Appealing GUI.
- Command-line interface (CLI) to run all commands without the GUI.
- CyDAQ and USB driver on Windows capable of transferring data at 2Msps.
- Communication between the two CPU cores capable of transferring data at 10Msps.
- Support for 5 unique modes:
    - Balance Beam, Basic Operation, Config, DAQ, External ADC.
- Supports configuration of:
    - Muxes, Sampling Rates, Generation Rates, Filters, Balance Beam, and SDR.
- Exportation of data in CSV or MAT format.
- A live stream of data to a plotting window.
- Error-checking of communication with the CyDAQ within the GUI.

Applicable Courses from Iowa State University Curriculum

- CPR E 281: Digital Logic
- CPR E 288: Embedded Systems 1: Introduction
- CPR E 381: Computer Organization and Assembly Level Programming
- E E 224: Signals and Systems I
- E E 324: Signals and Systems II
- S E 309: Software Development Practices
- S E 319: Construction of User Interfaces
- S E 339: Software Architecture and Design

New Skills/Knowledge acquired that was not taught in courses

- Python
- PyQt 5
- Qt Designer
- Xilinx Vitis
- Xilinx Vivado
- Asynchronous programming

Table of Contents

List of Figures

1 Team

1.1 Team Members

Yohan Bopearatchy, Wyatt Duberstein, Blake Fisher, Corbin Kems, Cole Langner, Jens Rasmussen, Long Zeng

1.2 Required Skill Sets for Your Project

- C/C++
- Firmware Design
- Git/Gitlab
- Python
- Testing
- UI/UX Design
- Verilog
- VHDL

1.3 Skill Sets covered by the Team

- C/C++ (All)
- Firmware Design (Long, Yohan)
- Git/Gitlab (All)
- Python (Corbin, Wyatt, Blake)
- Testing (Cole)
- UI/UX Design (Blake, Jens)
- Verilog (Long, Yohan)
- VHDL (Long, Yohan)

1.4 Project Management Style Adopted by the team

We have adopted the hybrid, waterfall+agile, project management style. Our client gave a detailed list of requirements upfront so, there will be limited client involvement while working on the project. We will still have weekly meetings with the client for progress updates and feedback.

1.5 Initial Project Management Roles

Yohan Bopearatchy - Firmware Engineer
Wyatt Duberstein - CLI Engineer
Blake Fisher - GUI Lead
Corbin Kems - CLI Engineer Lead
Cole Langner - Testing Lead
Jens Rasmussen - Project Lead
Long Zeng - Firmware Engineer Lead

2 Introduction

2.1 Problem Statement

In the Signals and Systems I (EE 224) course in our Electrical Engineering curriculum, the students have been using the CyDAQ device to enhance their lab experience and deliver real-world results. Currently, the device allows students to measure and record electrical signals that are being input to the device, and outputs those measurements into graph files. Due to the hardware on the device not being properly utilized by inefficient firmware, the data transfer time is very high and requires a lot of waiting to be sent to the user's computer. There are also bugs within the user interface that have been causing issues for the students completing and TA's administering the labs; such as data not displaying correctly on the CyDAQ graphs or the user interface not being user-friendly.

Our proposed solution to these problems is to rework the firmware from the ground-up, utilizing the dual-core functionality of the CyDAQ for faster processing power and increasing the efficiency of the firmware code. The CyDAQ will also be implemented with USB communication to the host computer rather than UART, increasing the sampling speed to 10 MSps (internal) and 1 to 2 MSps (external). The User Interface will be completely redone in another language, using PyQt and focusing on intuitive and adaptable design. The design of the User Interface will primarily focus on making it easy for TAs or other senior design groups to add future functionality. This will be done by creating thorough documentation and following common coding conventions.

2.2 Intended Users and Uses

2.2.1 Electrical Engineering Students at ISU

Key Characteristics

- Participating in EE 224, EE 324 or any other signal processing related classes.
- Interested/passionate about electrical engineering and signal processing.

Needs Related to the Project

- Speed up CyDAQ for their class's labs.
- Increase reliability of signal processing equipment.
- Navigate easily through UI that is simple to understand and can safely handle errors and misconfigurations.

Benefits and Uses

- They will use the CyDAQ in labs and will benefit from a new, easy to use UI that enables faster learning

2.2.2 Electrical Engineering TAs and Professors at ISU

Key Characteristics

- Instructing classes in the ECpE department that use the CyDAQ.
- Knowledgeable about signal processing and knows the ins and outs of the CyDAQ to aid in problems that may arise

Needs Related to the Project

- Speed up CyDAQ for their class's labs.
- Increase reliability of signal processing equipment.
- Navigate easily through UI that is simple to understand and can safely handle errors and misconfigurations.

Benefits and Uses

- Better output and error reporting will make it easier to help students when they make mistakes
- An easier to use UI will make teaching the software take much less time

2.2.3 Instructors and Researchers at Other Universities

Key Characteristics

- Interested in streamlining signal processing learning activities.
- Looking for open-source signal processing technology.

Needs Related to the Project

- Access signal processing technology in an open-source environment.
- Utilize the platform in different ways not necessarily specific to ISU.

Benefits and Uses

- An open source project eliminates the need to design their own product from the ground up, costing time and money.

2.3 Requirements & Constraints

2.3.1 Functional Requirements

- The product should support 5 unique modes (Balance Beam, Basic Operation, Config, DAQ, External ADC) in the firmware, CLI, and UI.
- The product should have a CLI and UI that fully supports configuration of the following: Muxes, Sampling Rates, Generation Rates, Filters, Balance Beam, and SDR
- After reading in the signals, the UI and CLI should be able to export data in CSV format
- The CLI and UI should be completely decoupled
- Firmware on the CyDAQ and USB driver on Windows should be capable of transferring data at 2Msps
- Communication between the two CPU cores on the CyDAQ should be capable of 10Msps

2.3.2 Resource Requirements

- Previous implementation of CyDAQ hardware
- CyDAQ accessories used for labs (DAD, balance beam)
- Windows Lab Computers

2.3.3 Aesthetic Requirements

- The product's UI should be considerably more appealing to our client then the existing one
- The product's UI should support all 5 modes in one application
- The product should have a fully functional command line window to run all commands without the GUI
- The product's UI should live stream data to a plotting window

2.3.4 User Experience Requirements

- Students should be able to configure their CyDAQ in labs with minimum effort
- The UI should be able to handle and give a way to fix errors that arise from bad communication to the CyDAQ

2.3.5 Constraints

- The product's UI and CLI can't use administrator privileges when running on lab Windows computers

2.4 Engineering Standards

IEEE 610.2-1987: Standard Glossary of Computer Applications Terminology
- It is important to use common application terminology in our documentation so that our application is intuitive and easy to use.

IEEE 610.6-1991: Standard Glossary of Computer Graphics Terminology
- It is important to use common graphics terminology in our documentation so that our application is intuitive and easy to use.

IEEE 610.10-1994: Standard Glossary of Computer Hardware Terminology
- It is important to use a common hardware terminology so that our firmware is understandable and easily integrated into.

IEEE 610.13-1993: Standard Glossary of Computer Languages
- It is important to use a common programming language so that our code is understandable and easily integrated into.

IEEE/EIA 12207: Software Life Cycle Processes
- It is important to consider the different stages that software goes through during development and how they can help shape the project's design process.

IEEE/IEC 29119-2: Test Processes
- Testing is a critical part of software verification. Having test processes well defined with a standard behind them will ensure we are verifying our software properly.

Universal Asynchronous Receiver-Transmitter (UART)
- The existing communication uses UART, which will need to be understood and possibly added to throughout the project's lifecycle.

Universal Serial Bus (USB) 2.0
- The CyDAQ should be able to connect to any Windows PC over at least USB 2.0

3 Project Plan

3.1 Project Management/Tracking Procedures

We have adopted the hybrid, waterfall+agile, project management style. Our client gave a detailed list of requirements upfront so, there will be limited client involvement while working on the project. We will still have weekly meetings with the client for progress updates and feedback.

We will use GitLab to track progress, utilizing the issues / boards and milestones features. Since we are also using GitLab as version control, this will allow us to directly correlate progress with commits.

3.2 Task Decomposition

3.2.1 CLI

- Create prototype Python CLI tool that mimics existing CLI tool functionality
- Demo prototype and get approval
- Add new commands to support new GUI elements
- Add support for new mode configuration
- Data collection error checking

3.2.2 Firmware

- Firmware code walkthrough
- InterCore communication
- Function rewrite for ICC
- Live streaming supported

3.2.3 GUI

- Create updated wireframes using Figma
- Create mockup pages using code
- Integrate prototype pages with CLI commands

- Support new modes
- Add Live Streaming/Plotting support for basic and balance beam mode

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

3.2.1 Working CLI

- Have a prototype Python CLI tool that mimics existing CLI tool functionality
    - Demo the prototype to the client and advisor to get feedback
- Add new commands to support new GUI elements
- Add support for new mode configuration
- Data collection error checking

3.2.2 Firmware Improvement

- Firmware code walkthrough
- InterCore communication
    - Minimum 10msps required between CyDAQ cores
- Increased transfer speed between CyDAQ and lab computers
    - Minimum 2msps required
- Function rewrite for ICC
- Live streaming supported

3.2.3 Working GUI

- Create updated wireframes using Figma
- Create mockup pages using code
- Integrate prototype pages with CLI commands
- Support new modes
- Add Live Streaming/Plotting support for basic and balance beam mode

3.4 Project Timeline/Schedule

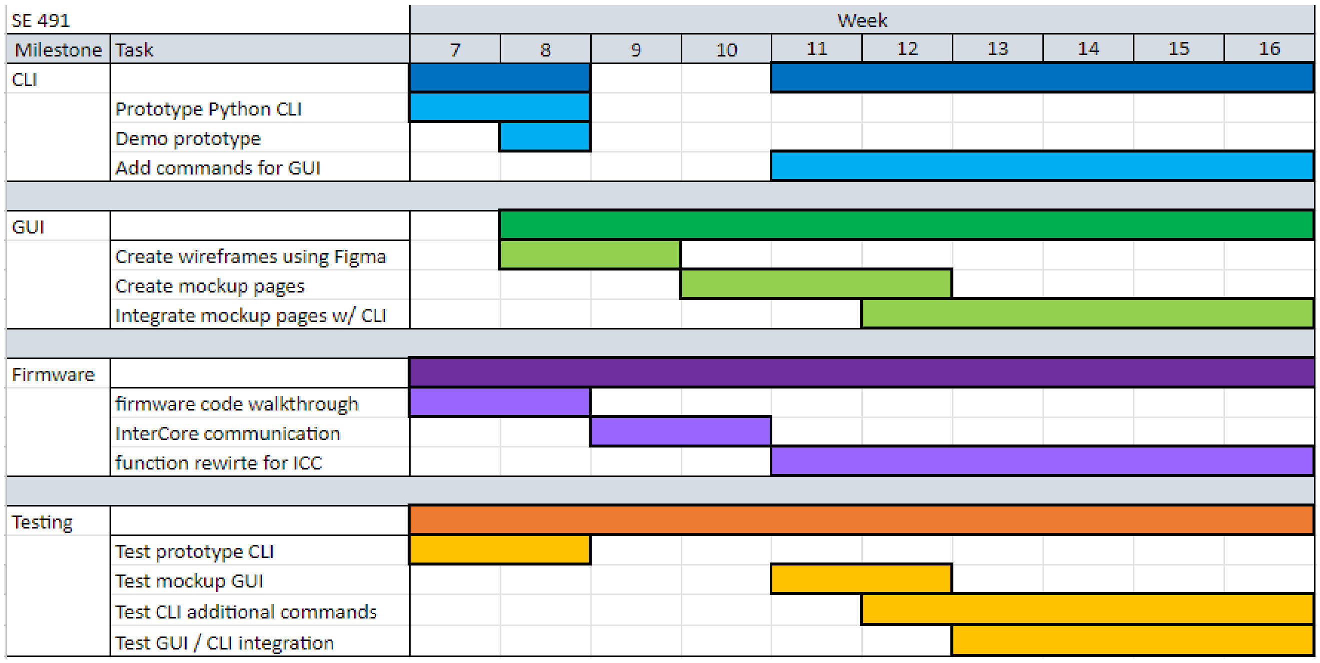

Figure 1: Semester 1 Gantt Chart

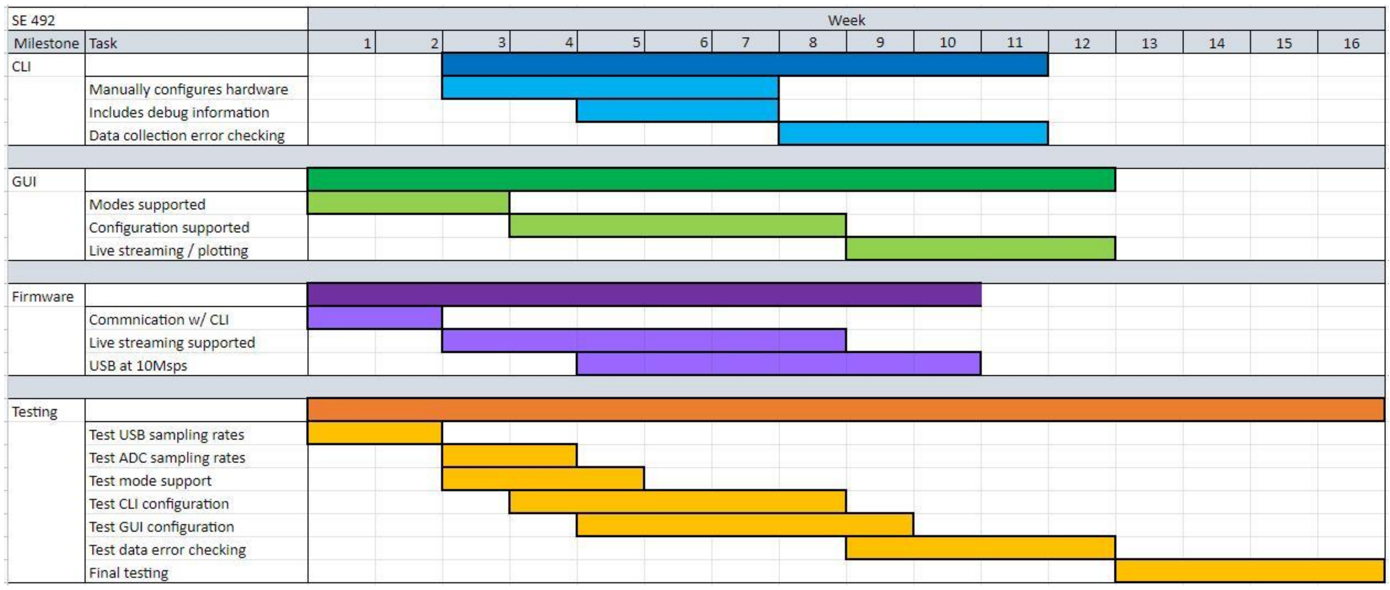

Figure 2: Semester 2 Gantt Chart

3.5 Risks And Risk Management/Mitigation

Time delays: other classwork
- Probability: 100%
- Consequences: Minor
- Mitigation: Spread out duties and communicate delays with advisor

USB can't support sample rate
- Probability: 1%
- Consequences: Major
- Mitigation: New protocol investigation

Code Issues
- Probability: 50%
- Consequences: Minor
- Mitigation: More time/debugging, code reviews with outside sources

Modes conflict with each other
- Probability: 10%
- Consequences: Major
- Mitigation: Testing and debugging

3.6 Personnel Effort Requirements

Milestones	**Hours / Task**	**Total Hours**
Working CLI - Prototype Python CLI	2 weeks x 4 hours x 2 people (16hrs)	136 hrs
Working CLI - Add commands for GUI	6 weeks x 4 hours x 2 people (48hrs)	
Working CLI - Manually configures hardware	5 weeks x 4 hours x 2 people (40hrs)	
Working CLI - Data collection error checking	4 weeks x 4 hours x 2 people (32hrs)	
Working GUI - Create wireframes using Figma	2 weeks x 4 hours x 2 people (16hrs)	256 hrs
Working GUI - Create mockup pages in QT Designer	3 weeks x 4 hours x 2 people (24hrs)	
Working GUI - Integrate mockup pages with CLI	5 weeks x 4 hours x 4 people (80hrs)	
Working GUI - Support for additional modes	3 weeks x 4 hours x 2 people (24hrs)	
Working GUI - Support all configuration	4 weeks x 4 hours x 2 people (32hrs)	
Working GUI - Include live plotting window	5 weeks x 4 hours x 4 people (80hrs)	
Firmware - Working ICC	4 Weeks x 8 hours x 2 people (64hrs)	248 hrs

Firmware - Communication with UI	2 weeks x 4 hours x 2 people (16hrs)	
Firmware - New modes	6 weeks x 6 hours x 2 people (72hrs)	
Firmware - USB driver update	6 weeks x 6 hours x 2 people (72hrs or more)	
Firmware - External ADC(analog to digital converter)	2 weeks x 6 hours x 2 people(24hrs or more)	
Total Hours		640 hrs

Figure 3: Personnel Effort Hours Breakdown

3.7 Other Resource Requirements

- ECE GitLab services
- Previous implementation of CyDAQ hardware
- CyDAQ accessories used for labs (DAD, balance beam)
- Windows lab computers

4 Design

4.1 Design Context

4.1.1 Broader Context

Public Health, Safety, and Welfare

- The CyDAQ board does not use any toxins or harmful chemicals that could harm the user or anyone who handles the board
- The CyDAQ board will come in a blue safety box that the user will not have to remove to ensure nothing on the board gets damaged or ruined
- The user will only need to plug in the needed cords from the CyDAQ board to the computer

Global, Cultural, and Social

- Changing the existing CyDAQ user interface will disrupt how users are used to using the CyDAQ , causing potential undesired change.
- The development of the firmware and software of the CyDAQ board will allow professors, students, and any other users to easily use CyDAQ without any complications and many set ups.

Environmental

- The CyDAQ board does not require much power to run, and can run for years resulting in high reusability.
- The production of the CyDAQ board is carbon-neutral and has a low ecological impact.

Economic

- Development of new software will be able to run on existing CyDAQ boards and lab computers, making new hardware will not need to be purchased. This will make the cost of the update very affordable to the EE department.

4.1.2 Prior Work/Solutions

The CyDAQ started out as a senior design project in 2018, and has since been the focus of multiple other senior design projects afterward. The initial creation of the board was the first focus of the first group, and subsequent projects have fixed bugs, implemented labs for the board, and attempted to re-implement the user interface.

The biggest advantage of having this product be in a completed state is that the end requirements are very well defined. New implementations of existing systems must, at the very least, provide the same functionality as the existing solution. Existing issues with the current implementation are also very clear to point out and improve upon, as there is a physical product that can be tested on throughout the project's development.

Some of the limitations of this project's context must also be considered. Because it is the product of multiple past senior design projects, the CyDAQ's software is very inconsistent. This is due to the fact that multiple independent teams have worked on it over the years with little to no interaction. Certain sections of the code have also been modified by electrical engineering staff between senior design projects to patch bugs and add functionality. Some of these fixes were very rushed, and require fixing before new functionality can be added on between them.

One existing product that the original senior design group in 2018 pointed out was National Instruments' myDAQ device. They concluded that using it for EE 224 labs instead of the CyDAQ wouldn't work very well because it required knowledge of Python scripting and creation of analog filtering circuits. Those two things didn't align very well with the teaching goals of EE 224, and would thus require too much overhead for students to use. This was one of the big reasons the CyDAQ had a Windows-based GUI built for it, as a CLIckable GUI is much easier to use than learning a scripting language.

4.1.3 Technical Complexity

The problem scope contains multiple challenging requirements that match the current industry standards. In this project, we will be using the most up-to-date forms of communication through USB, including our own firmware for the USB connection as well as converting from serial to USB 3.0. We will also be updating the language and function of the desktop application to a newer programming language, Python 3.10, from matlab. It also employs more modern programming techniques such as multithreading for more efficient performance for the students using it.

4.2 Design Exploration

4.2.1 Design Decisions

- Creating a CLI tool behind the GUI
    - This will allow us to abstract away communication with the CyDAQ, which makes it much easier to build other tools on top of it. We will build the GUI application on top of the CLI tool.
- Writing the CLI tool in a way that can be re-implemented by future projects very easily by creating a CLI Wrapper.
    - The wrapper library will allow for any other python program to easily interface with the CyDAQ, making it very easy for other groups to pick up on our progress in future semesters.
- Using PyQT to create the GUI
    - The overall decision and benefits/downsides are explained further in sections 4.2.2 and 4.2.3.
- Using the dual core functionality on the FPGA to improve the sampling speed
    - This will allow us to handle sending/receiving data as well as commands simultaneously with the CyDAQ, which will improve the responsiveness of the GUI application.
- Using USB connection with virtual COM port
    - This will enable us to greatly increase our data transfer speed, so students aren't waiting as long for their samples to get sent to their lab computers.

4.2.2 Ideation

Idea: Using PyQT to create the GUI
Ideation: We used the lotus blossom technique, placing "Windows GUI Framework" in the middle. After researching current frameworks, what they're specialized for, and what programming languages they used, we built out on the lotus blossom.

- Electron - develop native apps using web technologies such as JavaScript or HTML/CSS.
- Kivy - create desktop apps for Windows in Python
- PyQT - develop cross-platform widget-based GUIs in Python
- Tkinter - lightweight GUI library that comes with the official Python installation
- WinForms - develop application windows (forms) very similar to Windows OS windows in .NET

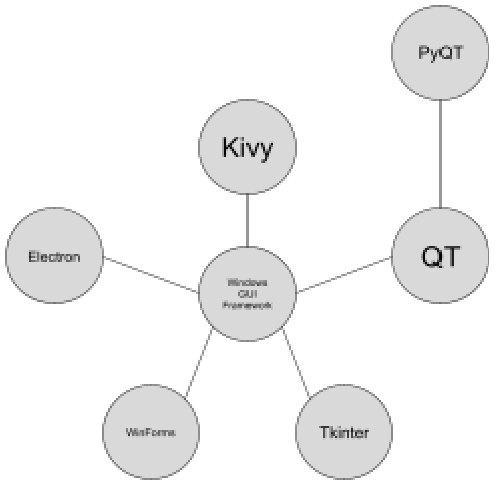

Figure 4: Ideation Lotus Blossom Diagram

4.2.3 Decision-Making and Trade-Off

One of the first design decisions we needed to make was how we were going to make our GUI. Because we were given complete control over what to use to make the GUI, we had a lot of options. Below is the process we used to narrow it down to one choice we think is the best.

We originally had 5 ideas for potential GUI solutions, but quickly realized that Tkinter and WinForms were going to produce a very bad user interface and shouldn't be considered. That left Electron, Kivy, and PyQT. All three would be good solutions, so we decided to use a decision matrix to help us decide.

OPTIONS							
Criteria	Weight	Electron		Kivy		PyQT	
		Score	Total	Score	Total	Score	Total
Learning Curve	0.1	3	.1	5	.5	5	.5
Look/Feel	0.4	4	1.6	3	1.2	4	1.6
Maintainability	0.3	4	1.2	3	.9	4	1.2
Reliability	0.2	2	.4	4	.8	4	.8
Total	1		3.3		3.4		4.1

Figure 5: Decision-making Matrix

PyQT was the obvious choice here with the higher score. Having it written in Python lowered the learning curve significantly, but it was able to beat out Kivy (another Python platform) by having a better look and feel that was more configurable. These are the main decision points that went into picking PyQT as our GUI framework.

4.3 Proposed Design

4.3.1 Overview

The CyDAQ software implementation can be split into three distinct components. Starting at the top is the user interface, which is where the end user will interact with the CyDAQ directly. This will handle all function calls to the CLI and hardware, as well as handle any input and output of files or data for sample collection or DAQ generation.

The middle component is the CLI wrapper and CLI tool, which acts as the middleman between the interface and hardware itself. It allows any external program to run easily defined functions that configure and control the CyDAQ device. It does this over USB and UART.

The bottom component is the CyDAQ firmware. It contains the code that runs on the CyDAQ. It connects the CLI with the actual hardware by converting CLI commands to assembly language using C++. All of the commands from the CLI are received and sent over USB/UART. It acts as the backend of the CyDAQ.

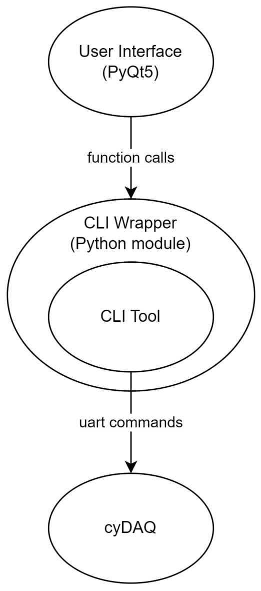

Figure 6: Basic Design Overview

4.3.2 Detailed Design and Visual(s)

The GUI is using PyQt5 to create a user interface with buttons and other various user input objects. On UI startup, an instance of the CLI Wrapper will be created for communication with the CyDAQ. The UI will use the wrapper to send config changes, start and stop sampling, and start and stop DAQ generation. The Wrapper will throw specific Python Exceptions when various errors occur, such as CyDAQ disconnected, improper configuration, and many others. This will allow for proper error handling on the GUI side, which can allow for users to better understand any issues they run into. The GUI will also have the ability to pick a file save location when a configuration requires it.

The CLI Wrapper is a class that exposes various methods other programs can use to easily configure and control the CyDAQ. It does this by using the Pexpect library to create a terminal session in another thread to asynchronously run the CLI tool. This allows for commands to be sent to the CLI tool in the exact same way that a user running the CLI tool would, over stdin. Output works the same way, as the Pexpect library listens on stdout until an expected response is received. This allows for a simple but robust communication interface for external programs to use.

The CLI tool is a Python script that listens for user input and processes input commands synchronously to communicate directly with the CyDAQ device over UART. It behaves much like a terminal application, but only with commands related to configuring and controlling the CyDAQ device. A help menu can be printed with a command, which lists all available commands that can be run. It processes input over stdin and outputs to stdout (Python input() and print()).

The firmware is a two standalone vitis program written in C++. one program for each core. The two programs are linked by using flags and shared registers. The program for the communication core handles receiving commands from the CLI, and returns messages such as sampling data or error code back to the CLI. Next it will wake up the sampling core by setting the flag to active, and share all of the needed data with those shared registers. Then the sampling core will take over and start sampling based on the configurations sent from the other core, and send the data back to the communication core. Finally the communication core will send the data to the CLI over USB/UART.

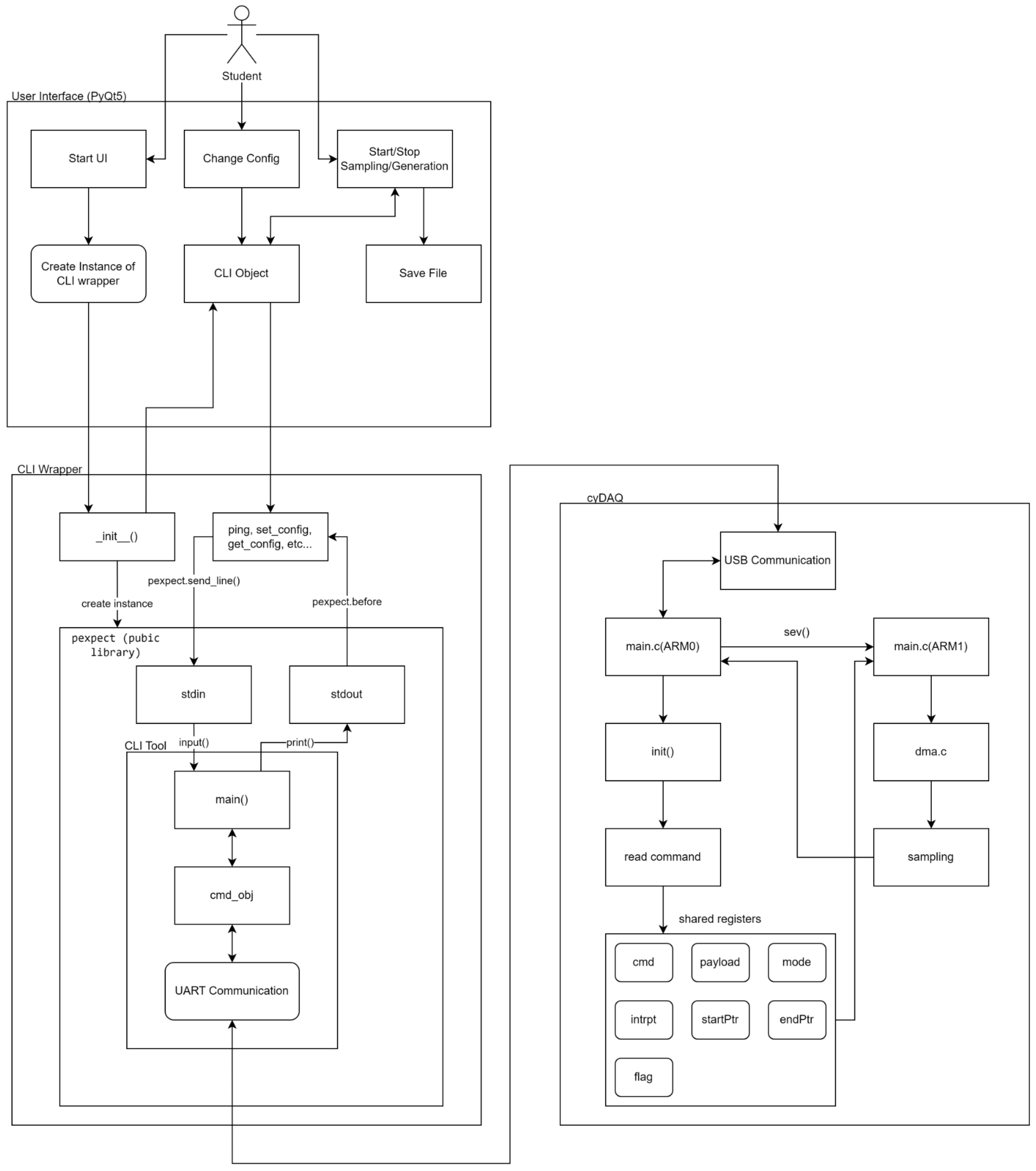

Figure 7: Detailed Design Overview

4.3.3 Functionality

Users will plug the CyDAQ to the computer using the new USB connection and turn on the circuit board. The user will also need to open the CyDAQ GUI on the computer. Once the CyDAQ is plugged in, turned on, and the user has opened the GUI, the user will be able to choose which mode they want to run on the CyDAQ as well as configure the settings for the test they are running. When the mode and settings are set, the user will click the Start button and the CyDAQ will sample data until the user clicks the Stop button. Once the sampling is stopped, the CyDAQ will send the user the full data compilation as a .csv file. The user then analyzes the data and repeats the same process of choosing the mode, configuring the settings, and sampling until they are satisfied with the results.

4.3.4 Areas of Concern and Development

In order to meet user needs the new user interface design will be more intuitive and user friendly, as well as less glitchy and more functional for the current and new modes that will be implemented using the CyDAQ.

In order to meet user needs the hardware will also be utilized more as we will be able to implement dual-core functionality, allowing for faster data processing and more synchronization of tasks on the CyDAQ.

A concern that we have is that, as the developers of the new user interface, we might find it easy to use and understand, but the end user, who doesn't know anything about the CyDAQ when beginning to use the user interface, might struggle to understand what they are doing or why. This could cause students to run into error messages or behavior in the GUI that can have a complicated message but a very simple solution.

There are a few approaches that can be taken to solve the above issue. One would be to create thorough documentation for the GUI, including troubleshooting steps when strange behavior occurs. The documentation would also include screenshots of expected behavior to lessen confusion. Another solution would be to implement more robust error handling on the CLI and GUI applications. Having clear feedback when very common errors occur would prevent unnecessary confusion and allow for TAs to help students much easier.

4.4 Technology Considerations

Our GUI is using PyQt5. The biggest strength of the library is that it is written in Python, a very well known and widely used programming language. It will be very easy for others to fix bugs or add features in the future because the library and programming language are very well documented online. One drawback of using PyQt5 is that its graphical designer is outdated and difficult to use. This makes it difficult in the initial stages of building the application, as the added overhead makes GUI development take longer.

Our CLI wrapper has the major advantage of allowing other tools (like the GUI) to be completely decoupled from the UART communication with the CyDAQ. This is beneficial because it allows future teams to use the CLI wrapper to communicate with the CyDAQ without having to re-implement the UART communication. One disadvantage of the CLI wrapper is that it is entirely dependent on the CLI tool, which means they will most likely come bundled together. This could potentially be a disadvantage for a future team if they wanted to rewrite the CLI tool in a different language. However, this is unlikely to occur.

4.5 Design Analysis

By the end of the first semester, we were able to get a working prototype for the GUI that uses a new CLI tool wrapped in a new CLI wrapper. This prototype only supports one mode, basic operation, but was able to verify that our proposed design is feasible nonetheless. We were able to take a basic sample on the CyDAQ, send it over the CLI to the GUI, and save it as a .csv file exactly as a student would be expected to do in a lab from our newly built GUI application. We even demonstrated this functionality to our sponsor and advisor in our last meeting of the semester, and they were able to give us some good feedback.

Our prototype and feedback we were able to receive at the end of this semester has enabled us to make a much better plan for next semester from a GUI and CLI standpoint. We were asked to add some more features and images to the basic operation mode to help out students, as well as some live information about the sampling buffer size. We were also given more detailed information about the other modes our UI will need to support. We know that we are on the same page with our advisor and sponsor because of our live demonstrations, which will also set us up for success next semester.

Lastly, for the firmware side of things, we have been able to establish dual-core functionality on the CyDAQ, but have been running into a lot of issues with switching it over from serial to USB communication. So the design is mostly there, but the implementation is still a work in progress. The design is good but we have a lot of bugs to work out from the previous version that weren't addressed. We were also working on rewriting all of the supported functions on the firmware to adapt them with our new dual-core implementation. Also with minimal interference with the CLI. In order to achieve this, we kept all of the command format the same as before, but split the functions to two different parts for each cores to handle.

5 Testing

5.1 Unit Testing

5.1.1 CLI

Lab Computer Usage Test - Manual running of the CLI tool on a lab computer to make sure it is capable of running as a non-administrator. CyDAQ connection not required.
Tool - We are using the 'unittest' library in Python to test the CLI. This allows us to write class objects that test the inputs and outputs that might come from using the CLI.

5.1.2 Firmware

PuTTY ASCII Binary Test - Manual command input using PuTTY. Instead of using the CLI, We also use PuTTY to put in custom or debug commands to communicate with the FPGA.
Data Transfer Efficiency Test - Testing how much data sample we get from the CyDAQ board using the updated firmware code we have been working on this semester. We will be running the code and retrieving the data sample on Vitis 2020.1.

5.1.3 GUI

Lab Computer Usage Test - Manual running of the GUI on a lab computer to make sure it is capable of running as a non-administrator. CyDAQ connection not required.

5.2 Interface Testing

5.2.1 CLI

Run the CLI tool manually while the CyDAQ is connected by USB to a lab computer. The CLI tool should be able to ping the CyDAQ using the ping command. This confirms that the UART connection over USB is working between the computer and the CyDAQ.

5.2.2 Firmware

The firmware on the CyDAQ, the CLI tool, and the GUI are the three major units that need to interface with each other over two separate connections. The following can be done to test each interface effectively.

5.2.3 GUI

Run the GUI tool on a lab computer and edit some configurations in basic operation mode. The configuration changes should update to the CLI tool and display in console output. This will verify that the GUI is able to create a CLI instance and send commands to it.

5.3 Integration Testing

5.3.1 CLI -> Firmware

The integration between the firmware and the CLI tool is critical to the overall functionality of the CyDAQ because it determines how the samples from the hardware can be shown and used from a computer. This can be tested by manually running the CLI tool and verifying that specific configurations made in the CLI tool are successfully uploaded to the CyDAQ when the send command is sent. These configurations will be created from existing lab material, as testing what students are configuring the CyDAQ to do is a good start for testing. Once the configuration is sent to the CyDAQ, sampling can be started/stopped, then the data sampled will be analyzed to confirm that it matches from the corresponding lab scenario.

5.3.2 GUI -> CLI

Communication between the GUI and CLI will be tested both manually and with automated tests. The manual portion will include configuring the GUI to match scenarios from student labs and having an unbiased third-party follow the lab instructions to test our work. The automated portion of the testing can check the functionality of the CLI commands run by interfacing with the GUI. An example of this could be pinging the CyDAQ or ensuring the correct configuration information is transmitted. More complex CyDAQ configurations can also be automated to make the testing process much faster, ensuring it is run more often throughout the development cycle.

5.4 System Testing

5.4.1 Hardware -> Firmware -> CLI -> GUI

This overall system testing will be a combination of the aforementioned unit, interface, and integration tests. It will be a very manual process due to the nature of our project and the need to be in the lab with a physical CyDAQ box in front of us.

5.5 Regression Testing

At the end of every major feature update to any component of the CyDAQ system, a full system test will be completed before that update is marked as completed. This will involve getting a third party to complete lab portions and report any issues they had with configuring the CyDAQ in the process. For smaller updates, integration tests only need to be run on the corresponding sections affected.

5.6 Acceptance Testing

5.6.1 Non-Functional

client Acceptance

We will give live demonstrations of the software for the client on a regular basis to ensure the non-functional requirements of the GUI are being met. The demonstrations will primarily focus on the aesthetic and user experience requirements of the GUI application.

Student Acceptance

It is important that Students are able to configure their CyDAQ in labs with minimum effort. We will ensure that this is true by having an outside party complete a lab with the updated CyDAQ software just as a regular student in EE 224 would. They will work on the lab without any other help to ensure ease of use of the software. Positive or negative feedback will directly be related to a pass or fail for this test.

5.6.2 Functional

We will ensure the CyDAQ supports the 5 modes requested by the client in the firmware, CLI, and GUI during the demonstration. We will do sampling tests that quantify the rates our product can handle and compare these to the 2 Msps through USB and 10 Msps between cores requirements given by the client.

5.7 Security Testing

We've determined that security testing isn't applicable to our current project as this is simply a device that reads signal inputs and outputs them in a closed lab setting. Everything that runs in the labs is confined to each individual lab computer, and the devices themselves don't hold any critical data or could cause harm to students in any physical way.

5.8 Results

We want to test the CyDAQ to make sure our final product is usable by the students and meets all of our clients requirements. We will ensure this compliance by customizing tests for each requirement to have concrete evidence to show the client. We will also be conducting manual testing to verify lab processes work as expected everytime a major milestone has been achieved. This combination of manual and automated testing will allow for difficult to find bugs as well as common usability requirements to be found before the final product ends up in real labs.

6 Implementation

Now that we have a baseline GUI, CLI, and CLI Wrapper implemented, next semester we plan to add more features and fix existing bugs until all required features in 3.3 have been successfully implemented. This will require finishing the dual-core mode in the firmware, then updating the CLI tool to support dual-core communication. While these are being worked on, the layout for the new modes can be added to the GUI as well as creating placeholder methods in the CLI Wrapper if necessary. Then, the firmware team will create the new modes and optimize the code to speed up the sampling speed on the CyDAQ, and the CLI/GUI teams will enable support for those modes as they are implemented. Part of one of these new modes will be the livestreaming method, which will also need to be added to the basic operation mode that has already been created.

We will still have weekly meetings with our advisor next semester, so we will be able to modify this plan as needed. This will also make sure we are on the right track to finishing development in the expected time frame, and make any necessary changes to our expectations if certain aspects of the project are taking too long to implement.

7 Professional Responsibility

7.1 Areas of Responsibility

Area of Responsibility	**Definition**	**ACM Code of Ethics**
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence.	2.1 Strive to achieve high quality in both the processes and products of professional work, 2.2 Maintain high standards of professional competence, conduct, and ethical practice, 2.3 Know and respect existing rules pertaining to professional work,2.6 Perform work only in areas of competence
Financial Responsibility	Deliver products and services of realizable value and at reasonable costs	N/A
Communication Honesty	Report work truthfully, without deception, and understandable to stakeholders.	1.3 Be honest and trustworthy, 1.5 Respect the work required to produce new ideas, inventions, creative works, and computing artifacts, 2.4 Accept and provide appropriate professional review, 2.5 Give comprehensive and thorough evaluations of computer systems and their impacts, including analysis of possible risks, 3.2 Articulate, encourage acceptance of, and evaluate fulfillment of social responsibilities by members of the organization or group, 4.1 Uphold, promote, and respect the principles of the Code,
Health, Safety, Well-Being	Minimize risks to safety, health, and well-being of stakeholders	1.1 Contribute to society and to human well-being, acknowledging that all people are stakeholders in computing, 1.2 Avoid harm, 3.3 Manage personnel and resources to enhance the quality of working life.
Property Ownership	Respect property, ideas, and information of clients and others.	1.5 Respect the work required to produce new ideas, inventions, creative works, and computing artifacts., 1.6 Respect privacy, 1.7 Honor confidentiality, 2.8 Access computing and communication resources only when authorized or when compelled by the public good, 2.9 Design and implement systems that are robustly and usably secure,
Sustainability	Protect environment and natural resources locally and globally.	N/A
Social Responsibility	Produce products and services that benefit society and communities.	2.7 Foster public awareness and understanding of computing, related technologies, and their consequences, 3.1 Ensure that the public good is the central concern during all professional computing work, 3.4 Articulate, apply, and support policies and processes that reflect the principles of the Code, 3.5 Create opportunities for members of the organization or group to grow

		as professionals, 3.6 Use care when modifying or retiring systems, 3.7 Recognize and take special care of systems that become integrated into the infrastructure of society, 4.2 Treat violations of the Code as inconsistent with membership in the ACM

Figure 8: Professional Responsibility Areas

The ACM Code of Ethics focuses mainly on societal, social, and communication ethics. It doesn't directly focus on sustainability or financial responsibility, because it seems to be focused more on the human impact of engineering compared to the NSPE code of ethics. It also divides each section into many specific parts. For example, social responsibility has 7 areas under the ACM code of ethics while the NSPE only has one. This makes it so the ACM code of ethics has more concrete ethical standards, but it also means certain things don't necessarily apply directly (sustainability and financial responsibility for example).

7.2 Project Specific Professional Responsibility Areas

Work Competence: This is very applicable to our project because we are expected to produce high quality work with timeliness, integrity, and professional competence. We are performing at a high level in this area by working consistently throughout the semester, meeting expectations, and meeting deadlines.
Financial Responsibility: This is not very applicable to our project because we are not creating a product with realizable value that will be sold.
Communication Honesty: This is applicable to our project because it is important to communicate changes to and receive feedback from our client frequently. We are performing at a high level in this area by having weekly meetings where updates are shared.
Health, Safety, Well-being: This is not very applicable to our project because there aren't any real risks to health, safety, or well-being.
Property Ownership: This is applicable to our project because we are utilizing CyDAQ hardware provided and paid for by the ECpE department and we need to act responsibly with the equipment. We are performing at a high level in this area by being respectful of the equipment we are given and exercising caution when using it.
Sustainability: This is not very applicable to our project because there aren't any natural resources being used, just computing resources.
Social Responsibility: This is applicable to our project because we are creating a product that will enhance Electrical Engineering student's experience in EE224 and EE324. We are performing at a high level in this area by constantly thinking about how design choices will affect the end-user.

7.3 Most Applicable Professional Responsibility Area

The most applicable professional responsibility area to our project would be Work Competence. We strongly believe that, for our project to be successful, we will need to work competently and professionally with each other as a team and our mentor/advisor. If we are not responsible in this regard to each other as teammates, we will be unable to successfully work as a team to complete our required tasks. Many elements of our project require separate teams to develop systems separately, then put them together. This would not happen very well without successful inter-team Work Competence. If we are not responsible in Work Competence with our sponsor/advisor, we will never be on the same page, and any of the work we do produce might not be what was expected. These reasons are why Work Competence is the most applicable processional responsibility area.

8 Closing Material

8.1 Discussion

The end goal of this project is to create a much better piece of software than currently exists for Electrical Engineering students to control the CyDAQ in their labs. This requires recreating most major components including completely redesigning the user interface, rewriting the CLI tool, and overhauling the firmware running on the CyDAQ. Once existing functionality has been replicated with our version, new modes will be created; as requested by the Electrical Engineering Department. The functional requirements in section 2.3 explain each required and new feature in more detail.

8.2 Conclusion

8.2.1 CLI

We have recreated the CLI tool, which involved re-writing large chunks of the existing code as well as re-using certain parts of code that were still usable. Because this was one of the first tasks we completed for this project, it was also a chance for us to learn more about the CyDAQ through the existing CLI implementation. This was also a major constraint, as it required spending extra time to learn about the project as a whole. Once complete, the CLI tool was demonstrated to the client and sponsor for approval, then work on the CLI wrapper was started.

The CLI wrapper was created as a library that implements the CLI tool using the pexpect Python library. Pexpect allows for mocking a terminal application and sending and receiving commands exactly like a user would. This ensured that the CLI and CLI wrapper were completely decoupled, which is an important aspect of this project. Once completed, the CLI wrapper was used by the GUI to send commands to the CyDAQ easily and safely.

Since the CLI and CLI Wrapper are in a demonstrated working state, this will make it very easy to add new features as they are needed next semester. This also proves that our proposed implementation works in the real world, which is a very good spot to be in right now.

8.2.2 Firmware

We have started implementing the CLI to firmware communication for the new CyDAQ firmware. We learned a lot about the CyDAQ and how it operates. Since the CyDAQ has a dual core ARM processor, we implemented two standalone programs that run separately on each core, and communicate to each other with shared registers.

The first core(ARM0) handles taking all of the commands from the CLI, and sending data back to the CLI. The other core(ARM1) handles just sampling and processing data. When the CLI sends a command to the CyDAQ, ARM0 will receive it, and identify the process it needs to do. All of the setups and configurations are stored and shared with ARM1. And when ARM0 receives a command to start sampling, it uses a flag to call ARM1 to start sampling and send data to ARM0, and ARM0 will send the data to CLI when needed. Similarly, ARM0 uses the flag to notify ARM1 to stop sampling when needed.

We have recreated the firmware based on the stable one that is being used in the labs. Majority of the functions are split into two parts: command part, and sampling part. The command part is handled by the communication core, and the sampling part is handled by the sampling core.

We also changed the UART communication method to USB over UART for data transmitting speed. The firmware is using a USB driver that mocks as a UART COM port. Although it is still using serial communication, the USB data is being transferred using a separate micro-usb port. This way we are able to maximize the sampling speed, and eventually achieve live data streaming(10 Msps through ICC, 2 Msps through USB).

8.2.3 GUI

For the GUI so far, we have designed and built a home screen and a screen for one of the CyDAQ modes. We went through multiple iterations of the UI and ultimately settled on what we have right now. Next semester our goals are to build out the screen for the rest of the modes, add images and indicators, and do more touch-ups. One of the constraints of achieving these goals this semester was PyQT and QtDesigner. This framework has a steep learning curve and is not very intuitive to use. We have a decent grasp of it now to where we can make the new screens with minimal constraints. What could've been done differently is research more about different UI frameworks for Python and read more documentation before trying to build something.

8.3 References

[1] "sdmay18-31 • Develop lab interface for EE224 data collection," sdmay18-31.sd.ece.iastate.edu. [Online]. Available: https://sdmay18-31.sd.ece.iastate.edu/. [Accessed: Dec. 02, 2022]

[2] "sddec20-14 • Introduction of Real-World Signals and Systems into ECpE DSP Laboratory Curriculum," sddec20-14.sd.ece.iastate.edu. [Online]. Available: http://sddec20-14.sd.ece.iastate.edu/. [Accessed: Dec. 02, 2022]

[3] "Dual ARM Hello World on Zynq Using Vitis," Hackster.io. [Online]. Available: https://www.hackster.io/whitney-knitter/dual-arm-hello-world-on-zynq-using-vitis-9fc8b7. [Accessed: Dec. 02, 2022]

8.4 Appendices

8.4.1 Team Contract

CyDAQ DSP Platform Firmware and Software Redesign

Team Members:

1) Blake Fisher
2) Corbin Kems
3) Long Zeng
4) Yohan Bopearatchy
5) Cole Langner
6) Jens Rasmussen
7) Wyatt Duberstein

Team Procedures

1. **Meetings**: Monday or Tuesday, 8:30pm, Discord (or SIC as necessary).
2. **Communication**: Discord, in-person.
3. **Decision-making**: Majority vote.
4. **Record-keeping**: Wyatt (or a substitute as necessary).

Participation Expectations

1. **Attendance**: Use Discord to communicate any tardiness or missing a meeting.
No-call no-show = round of shots.
2. **Responsibility**: Be on time or communicate if you are having issues.
3. **Communication**: Provide thoughtful input. Ask questions. Ask for help when needed.
4. **Commitment**: Do your work or communicate if you are having issues.

Leadership

1. **Leadership roles for each team member:**

1) Blake Fisher - UI	Title: GUI Lead
2) Corbin Kems - CLI Tool	Title: CLI Engineer Lead
3) Long Zeng - Firmware (USB driver)	Title: Firmware Engineer Lead
4) Yohan Bopearatchy - Firmware	Title: Firmware Engineer
5) Cole Langner - Tests and CI/CD	Title: Test Lead
6) Jens Rasmussen - CLI / Documentation	Title: Project Lead
7) Wyatt Duberstein - CLI / Issues	Title: CLI Engineer

2. **Supporting and guiding the work of all team members:**
Stuck for a day? Ask for help.

3. **Recognizing the contributions of all team members:**
GitLab analytics.

Collaboration and Inclusion

1. **Describe the skills, expertise, and unique perspectives each team member brings.**
Blake: UI/UX Design
Cole: Web Dev frontend and backend
Corbin: Web (react, angular)/Android (native, flutter) Frontend, Spring/DotNet backend, Python IOT dev, self-hosting, cybersecurity, some embedded

Jens: Android Frontend, JavaScript (React, Angular), Python (basics)
Long: Java frontend(android) and backend(spring), C, VHDL, verilog, circuit design(logic and analog)
Wyatt: Java backend(spring), cybersecurity, self-hosting
Yohan: Java frontend(android), Python(basics), C, VHDL, verilog

2. **Encouraging and supporting contributions and ideas from all team members:** Be open-minded. No bad ideas. Weekly meetings to share ideas and collaborate.
3. **Procedures for identifying and resolving collaboration or inclusion issues:** Vote on issues. Communicate any issues in a team setting as soon as they come up.

Goal-Setting, Planning, and Execution

1. **Team goals:** Complete a working project. Consistent meetings with good attendance. Take ownership of your faults. Communicate.
2. **Planning and assigning individual and team work:** GitLab issue boards.
3. **Keeping on task:** Gitlab analytics.

Consequences for Not Adhering to Team Contract

1. **How will you handle infractions of any of the obligations of this team contract?** As a group, communicate with the team member(s) as soon as the issue comes up.
2. **What will your team do if the infractions continue?** Take the issue to the professor.

****

a) I participated in formulating the standards, roles, and procedures as stated in this contract.
b) I understand that I am obligated to abide by these terms and conditions.
c) I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.

1) Jens Rasmussen	DATE: 9/6/2022
2) Wyatt Duberstein	DATE: 9/6/2022
3) Long Zeng	DATE: 9/6/2022
4) Yohan Bopearatchy	DATE: 9/7/2022
5) Cole Langner	DATE: 9/9/2022
6) Blake Fisher	DATE: 9/9/2022
7) Corbin Kems	DATE: 9/9/2022