University of Arkansas – CSCE Department
Capstone II – Final Report – Spring 2025

Electrostatic Discharge Latch-Up Detection

Gavin Edens, Kile Harvey, Luke Simmons, Carl von Bergen

Abstract

When designing integrated circuits (ICs) using Complementary Metal-Oxide-Semiconductor (CMOS) technology, an important issue that must be accounted for in the design process is latch-up, an unwanted occurrence in which power and ground become shorted due to voltage differentials across terminals on a CMOS gate. A common perpetrator of this phenomenon is electrostatic discharge (ESD), in which a charge is quickly released from an outside source into the IC, thereby causing latch-up. In industry, designs are tested for their resilience to this by using test benches that force conditions that can cause latch-up.

The Electrical Engineering (EE) team that we worked with on this project was tasked with creating a board that would function similarly to the test benches used in industry for causing latch-up in designs. There is information, however, that can be obtained by collecting data about the state of the design leading up to the occurrence of latch-up, which can lead to a deeper understanding of latch-up and why it occurs.

This data is tedious to manually parse through and interpret, so we created a desktop application that receives this data from the board and presents it in a way that is easy to interpret for users of the application, along with allowing for customization of the data visualization. We also allow control of the microprocessor on the board through interfacing with the desktop app.

1.0 Problem

Electrostatic discharge (ESD) can cause a phenomenon called latch-up in CMOS technology, which can permanently damage integrated circuits [1]. When electrical engineers design a circuit, this is something that they must be aware of and understand. This project focuses on helping electrical engineering students at the University of Arkansas to understand what ESD latch-up is and what leads to it by designing lab devices to measure the conditions of the circuit leading up to latch-up and using computer software to record and graphically display the data. Eventually, there are also hopes that this test bench can be extended for use in research applications for industry standard testing, too. As a joint project between an EE team and the Computer Science/Computer Engineering (CSCE) team, we improved upon last year's design by making the test bench communicate with a computer while also allowing for easy control and data processing.

In the electrical engineering industry, there are tests for resilience against latch-up caused by ESD. The standards organization for defining these is the Joint Electron Device Engineering Council (JEDEC). They define many electrical standards, one of which being JEDEC78F [2], which defines the testing standards for different levels of latch-up testing. Many electronics manufacturers use this standard for testing their circuits, so it is important to understand this testing standard as an engineer. This project is targeting JEDEC78F with levels A and B specifically in mind, but it is not an absolute requirement to precisely meet these standards for this project. Level A is more stringent, where a pin is tested at $\pm$ 100mA for current injection, and 1.5 times the maximum supply voltage for a voltage injection test. Level B is set by the manufacturer, who determines the current and voltage levels that the device can withstand if it does not pass level A testing and reports each value.

Last academic year, a device was made by an EE senior design team here at the University of Arkansas that can trigger and measure conditions leading up to one type of ESD latch-up. However, this initial product is missing many useful features. This year, the design is being improved upon by the EE and CSCE teams to add these missing features. On the electrical side, I^2C potentiometers were added to adjust testing criteria, and new types of ESD latch-up tests were available for testing, including: voltage tests, injections tests, and negative injection tests. The CSCE team were tasked with bridging the gap from the board to the desktop application, creating functionality for users to be able to select and start tests from the application, and displaying the data collected usefully. The computer software needs to be able to help lab students understand how the voltages and currents at various points of the board change as latch-up occurs.

This year, USB communication was integrated, so the test bench now has the ability to send and receive data from a controlling device. Using this, our application will display the data in a variety of ways so that students can explore the data collected – as opposed to needing to probe the device using an oscilloscope. It also will graph data in a way that is easily understandable, yet fully customizable so that users may interpret the data in different ways. It also will allow the users to trigger latch-up and load saved data from previous latch-up occurrences on request.

Dr. Chen, a professor at the University of Arkansas and our project sponsor, is also collaborating with industry partners, such as Huawei, making it ideal for the device and software to eventually be used for testing equipment under industry standards such as JEDEC IC Latch-Up Tests. Therefore, the project should be designed with this future-proofing in mind.

The contributions by the CSCE team will make it easier for students to understand and interpret latch-up in CMOS technology. Without our application, lab students would not be able to view latch-up data in easily digestible formats, quickly compare, save, and load data, or be able to select from a variety of latch-up tests; instead, data collection could only occur through an oscilloscope, which is not only a limited resource on campus, but also has a significant disadvantage in ease of use compared to a desktop application.

2.0 Objective

The objective of this project is to create an application that will interface with a latch-up test board designed by the EE Team. Upon request, the application should send a command to the test board, causing it to begin one of the latch-up tests. The application will then start receiving data from the board, where it will start graphing this data. The user will then be able to access different graphs of different data sets, along with allowing the data to be saved to a file for later access. Additionally, it allows for the graphing of previously saved data, with all the aforementioned functionality.

3.0 Background

3.1 Key Concepts

3.1.1 CMOS

CMOS, which stands for complementary metal-oxide semiconductor, is a semiconductor technology that is characterized as having both N-type and P-type transistors. An N-type transistor conducts electricity when a voltage over a specific threshold is applied to its gate, whereas a P-type transistor conducts electricity when the voltage applied to its gate is under this threshold. They are used together to form logic gates and other components that have much lower power consumption than either type of transistor alone would be able to achieve. In the manufacturing process, both the P- and N-type transistors are able to be placed on a single silicon wafer by creating “wells” in the different substrate types [3].

3.1.2 ESD

Electrostatic discharge (ESD) occurs when charge builds on a surface and is nearly instantaneously dissipated onto another. This is the same phenomenon that happens when you get a shock from touching something metal. When it comes to integrated circuits, this can be damaging to the circuit if not properly protected against. Electrical components can become shorted, heated to the point of damage from the shock, among many other unwanted things [4]. This includes latch-up.

3.1.3 Latch-up

Latch-up is an occurrence in CMOS technology where, usually due to some outside charge, there is a feed-back loop created within a CMOS element that causes the supply voltage to be shorted to ground. This is due to how CMOS is implemented on silicon wafers. The inclusion of N- and P-type diffusion in the wafer can create what are known as “parasitic transistors,” where PNP or NPN junctions in the substrate which aren’t meant to be transistors can act as such. This leads to current flowing through the substrate in ways that were never intended. If these flow in just the right way, the parasitic transistors become stuck in an “on” state, allowing a low resistance path between ground and the supply voltage. Often this latch-up phenomenon can be caused by electrostatic discharge, as this is an easy way for charge to build up and release in a manner that can overpower transistors. Typically, this cannot be stopped by normal functionality and requires a full power-off of the design in which latch-up has occurred. This causes multiple problems, including loss of functionality, excess power consumption, and even damage to the IC [1].

3.2 Related Work

3.2.1 Other latch-up recording software

Some companies offer software solutions in order to detect latch-up, such as the one discussed in [5]. The issue with these solutions, however, is that they offer data in a rather unpleasant form-factor. We believe that we can make a more attractive and human-readable display of data collected from the board. In addition, our application will have better integration with the EE team's custom board, as we will be building the software specifically for use with said board. Overall, we expect our application to be superior to those already available on the market in terms of this specific use case.

3.2.2 Other serial data-plotting software

Likewise, there are other tools available which allow the plotting of data received through serial communication, such as [6] and [7]. While it is likely that we integrate something similar into our application, this plotting software by itself is not enough for us to be able to visualize all aspects of the data that we would like to, nor would it allow a user to customize what is displayed to the degree that we envision for the project.

4.0 Design

4.1 Use Cases

The primary use cases include the ability to cause latch-up on the test board from the desktop application, view graphical representations of the data received from the EE board, and write data to and read data from CSV files.

The user is able to pick from one of three latch-up tests, which include the positive current injection test, the negative current injection test, and the voltage pulse test. Note: The voltage pulse test is an option that is available in the application but the board does not currently support it. The user will then be able to choose to run the test, signaling to the board to start the selected test and begin sending data to the application. The graphable data includes the voltage or current of various points in the design, which is expanded upon in the requirements section.

For each of the tests, the application both graphs the data being received, and stores the data in a way that can easily be saved to a CSV file. Once the board signals that it is finished sending data, different graphs may be selected from, allowing the user to view all forms of recorded data against elapsed time. If desired, the user can then save the collected data to a formatted CSV file.

In addition to starting latch-up tests, the application is also able to read previously saved CSV files, allowing for access to previously run tests. This also includes the ability to view the same graphs available after latch-up tests.

Each of the tests involves different ways to cause latch-up. The positive and negative current injection tests alter the current through the design, whereas the voltage pulse test pulses the supply voltage of the design to values outside expected operational values.

4.2 Requirements

4.2.1 Functional Requirements

- Motherboard records data at multiple points on the unit under test leading to, during, and after ESD latch-up. The current unit under test is a CMOS inverter. Recorded data includes:
  - Voltage at inverter output
  - Voltage at inverter input
  - Current through Silicon Controlled Rectifier (SCR) anode
  - Current through SCR Gate
  - Current from positive current injection test
  - Current from negative current injection test
  - Total test bench current supply
- Motherboard sends data to our desktop application through serial communication
- Desktop application is used to:
  - Calculate and store the data collected by the motherboard
  - Graph the data collected by the motherboard
  - Trigger specific latch-up tests on the board:
    - Positive injection test
    - Negative injection test
    - Voltage pulse test
- CSCE team in charge of the development of the desktop application
- EE team in charge of designing the required boards along with writing the firmware of the microcontroller unit used for controlling the boards and communicating with the desktop application

4.2.2 Interface Requirements

- Motherboard interfaces with a swappable daughter board in order to be able to test multiple designs
- Serial communication between the computer and the testing device
  - Testing motherboard uses USB-C for power and data transmission

- Desktop application stores data received from the device for:
    - Graphing
    - Saving to a CSV file if requested
- Customizable with multiple graphs and views that can be selected by the user:
    - Multiple tabs can be created with a button for graphing
    - Zoom with scroll wheel
    - Dependent variables selectable with dropdown
    - User can select device from menu bar if multiple devices are detected
    - User can set the potentiometer value, ranging from 0 to 9999 Ω.
- Desktop application will be able to send commands to the device
    - Starting selected latch up test
        - Selection is done through a drop-down menu on the main page or from the menu bar
        - Test is started through a button on the main page or from the menu bar
- Computer must have a usable USB port

4.2.3 Performance Requirements

- Software runs on desktop, so not limited by embedded chip processing power
    - Calculations are not performed on the embedded device, which allows for a higher effective sampling rate
- Board records data at a rate of once every 10 ms, or 100 Hz
- Communication must happen in near-real-time
- Computer must have enough storage/memory to store the data collected

4.3 High-Level Architecture

Our program is written entirely in Python. The EE team's board utilizes an STM32 microprocessor, which communicates with our program through USB serial communication, specifically using the UART protocol.

The application has four main components: the user interface, the data manager, the communication controller, and the graph manager.

The communication controller acts as a bridge between the rest of the application and the EE board, allowing two-way communication. The controller initiates the serial port of the connected board, saving it into memory, which then enables the sending and receiving methods to function off the defined port.

The data manager handles receiving data from the communication controller, calculating the actual value of data to be graphed, translating control bytes into commands and vice versa, writing to and reading from CSV files, and sending data to the GUI for plotting. Control bytes from the board are used to let the data manager know when to start and stop recording data, along with letting the data manager know what type of data is being sent.

The user interface can be broken down further into both the graphical user interface (GUI) and the backend of user-interactable elements of the GUI, the graphing manager. The graphing manager is the bridge between the GUI and the data manager, it sends commands that cause one of two things to occur: the data manager will either read data from or write data to a specified CSV file, or the data manager will translate the command into a control byte and then forward this control byte to the communication controller so that it can be sent to the board. Additionally, the GUI is where all visual representations of data can be accessed.

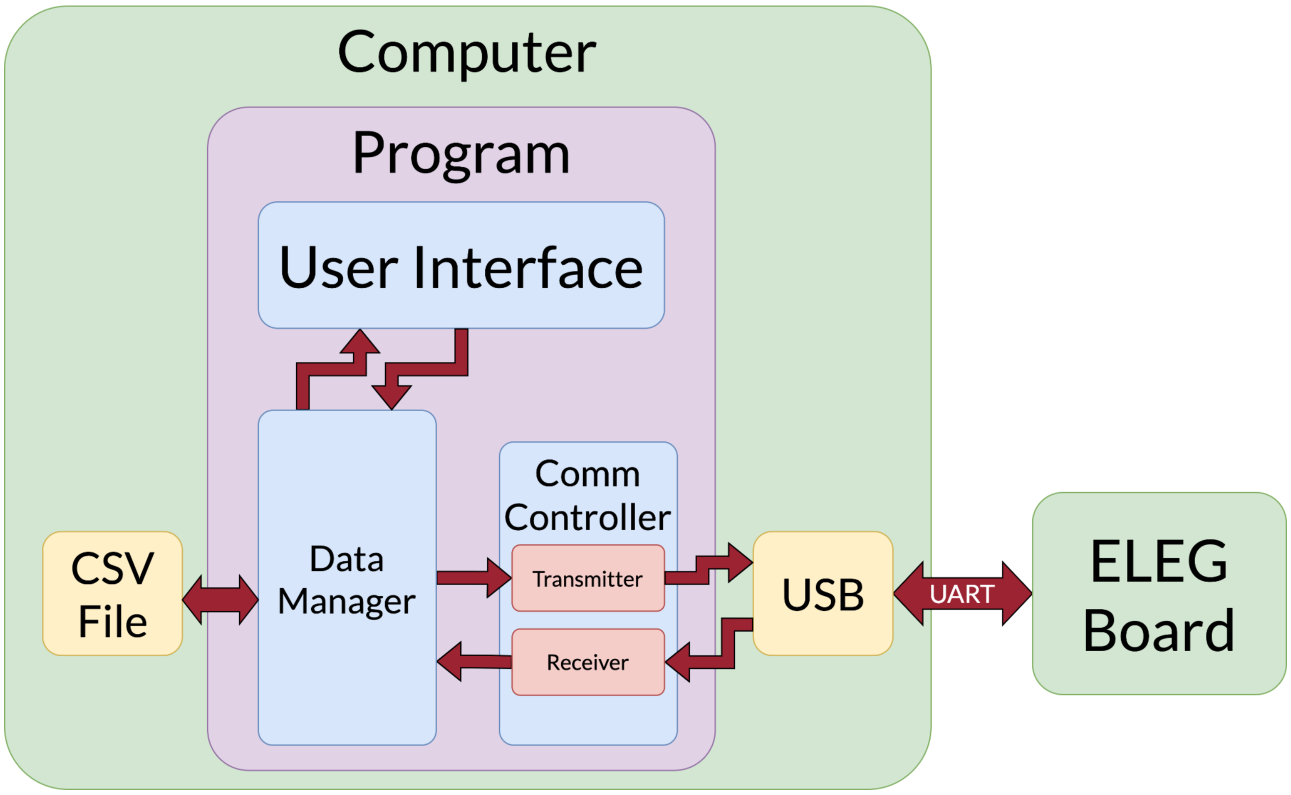


Figure 1: High-level architecture diagram

4.4 Detailed Architecture

Our project was developed using Python 3.12; we also utilized setuptools in order to package the project into a Python module. Qt6 was used as the GUI toolkit and PyQtGraph was also used to implement the graphs, both of which were implemented with the help of PySide6. When it comes to managing data, both Python queues and Pandas [8] dataframes are being utilized in order to allow for multithreading and CSV storage (more information in section 4.5). Finally, serial communication over USB is managed using PySerial. This stack allowed for easy development while meeting technical and performance requirements; it also allowed the application to be cross-platform for Windows, macOS, and Linux. As of now, the application is being developed as a proprietary program and will not be released as open source. def allowed

We used Git to handle our version control and used GitHub to host the main repository privately. Our development flow involves creating a different branch for each new feature and merging these into the main branch once fully tested. We also have GitHub Actions running tests, linters, and builds to make sure there are no regressions in our software.

The GUI contains ways to control the device and visual representations of the data. In the file menu in the menu bar, there are options for starting a new data collection session ("New Project"), and loading test data from a CSV.

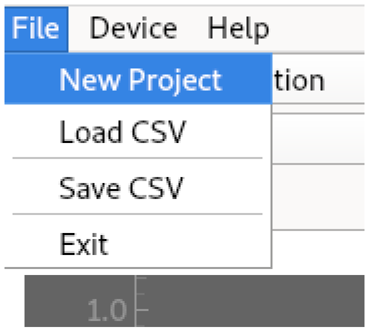


Figure 2: File Dropdown

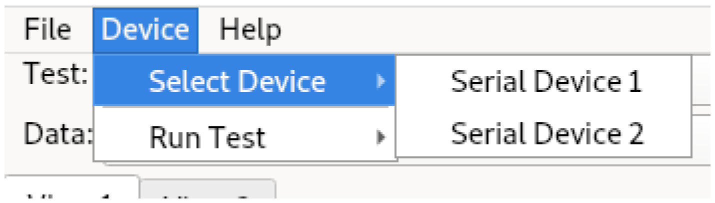


Figure 3: Device Dropdown

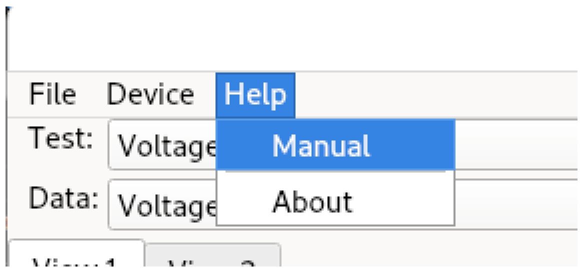
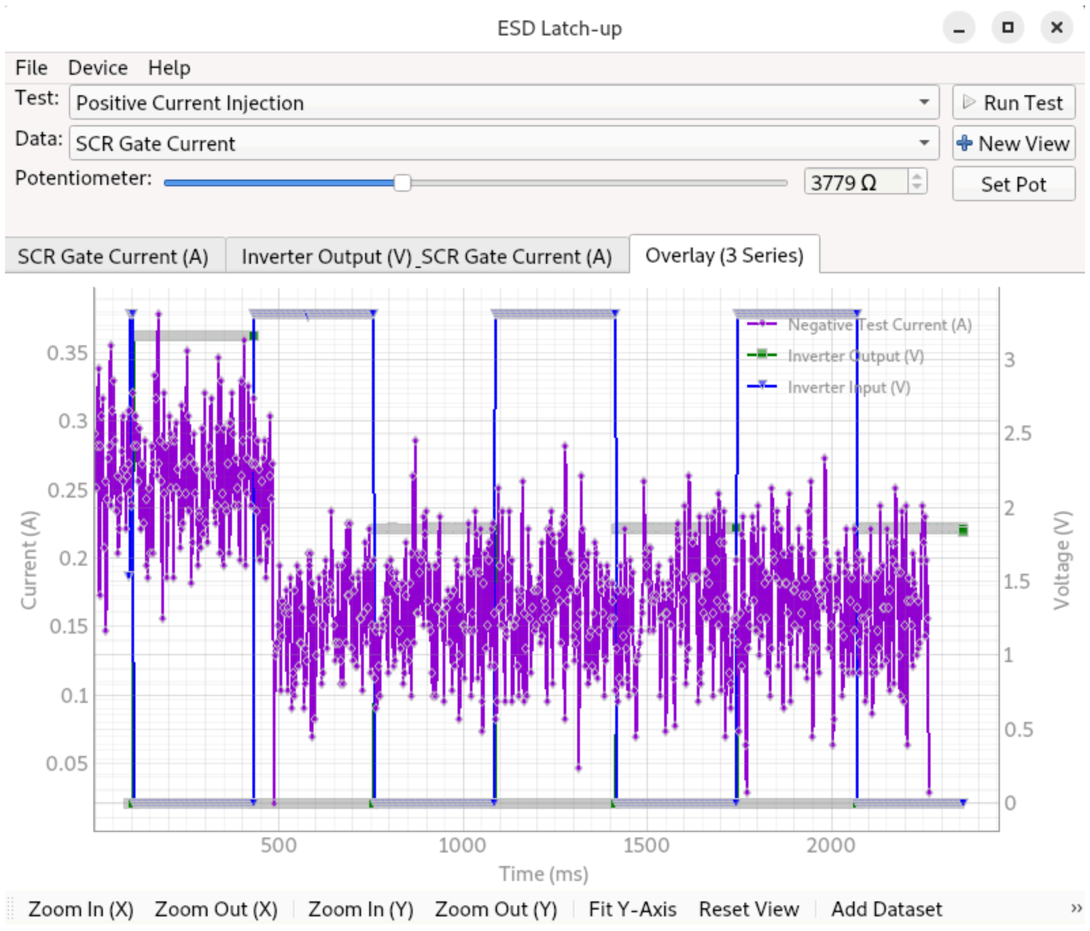


Figure 4: Help Dropdown

In the main screen, there is a drop-down box for selecting which test is going to be run, and the test will eventually be able to be started by pressing “Run Test.” Any of the tests will also be able to be started immediately from the “Device” menu. The graph will update in real time until it gets a signal from the device that the test is done. The user can switch between different graph views by using the tab switcher toward the top of the screen. New tabs can be created with the dependent variable being selectable through the drop-down box, and hitting the “New View” button to make the new tab.


ESD Latch-up

File Device Help

Test: Positive Current Injection

Data: SCR Gate Current

Potentiometer: 3779 Ω

Figure 5: Main window view after a test has been run and user added multiple datasets

When saving data to the disk, a window will open up asking the user for a file path to save the test data to, where it will then be saved as a CSV file. For opening a saved CSV to view previously recorded data, a window will open asking the user to select the CSV file. These menus are accessible from the "File" menu in the menu bar.

In regard to installing the application, we used PyInstaller to package the program into an executable. We wrote documentation which can be opened from the "Help" menu for users to access, in addition to writing descriptive tool-tips.

Communication with the test board will eventually be done through UART over USB, and further collaboration between the electrical team and computer science team is still required to develop the structure of commands and data sent between the computer and testing device.

4.5 Data Management and Storage

All internal data management and storage is handled by the data manager, whereas all external data management and storage is done through CSV files.

Internally, data is stored in two ways: through Python queues and Pandas dataframes. Queues are used to store any set of data that needs to be read from and/or written to by multiple sections of the application at once. This is because Python queues are thread safe, which allows for easy implementation of multithreading without worrying about data corruption. Dataframes are used whenever data is being written to or read from CSV files, as they allow for simple formatting and maintaining of structure without unnecessary overhead.

The application can either read in data from the EE board or simply open a CSV file in order to store data in the memory of the application for graphing. Whenever graphing live, queues are used to send data one at a time to the user interface, while a full dataframe is sent to be graphed whenever opening a CSV.

The user may save data collected from a test or read from previous tests to a CSV file whenever a test is not being run.

5.0 Development Plan

5.1 Tasks

5.1.1 Create project

This task included the creation of the GitHub repository and setup of the project file structure that was used during the development of the application. While completing this task, multiple systems were set up to ensure that each member of the team could operate as efficiently as possible, including concise version control and main branch protection (with code review being required before allowing merges).

5.1.2 GUI opening

Luke Simmons and Gavin Edens worked on opening the main UI window using the PySide6 python module. This window includes the basic elements of the program, such as the buttons, dropdown menus, and views for graphing. The functionality of each element was not fully implemented yet, but the structure of the UI was.

5.1.3 Generate sample data

Kile Harvey collaborated with the EE team in order to accurately generate random data in a manner similar to how the EE board will send data to the desktop application. During this task, the communication controller was created, which acts as a communication bridge between the board and the rest of the application. The storage of data from the board was also solidified, with Python's built-in queue being utilized for live read and write, and Pandas dataframes being utilized for saving to and reading from CSV files.

5.1.4 Loading data from CSV

Harvey utilized Pandas dataframes in order to load data stored in a CSV into a Python queue, eventually allowing the GUI to plot previously saved data. This was done using a built-in function of Pandas that reads a CSV and creates a dataframe from it. During this task, the data manager was created, which acts as the source of any data for the GUI. It also serves the purpose of handling reading/writing of/to CSV files.

5.1.5 Graphing with sample data

Simmons and Edens worked on graphing the data, using the NumPy/SciPy modules to graph the sample data. During this task, the graph manager class was created in order to receive data from the data manager and plot the user-specified data points onto the application window.

5.1.6 UART communication

Carl von Bergen worked with the dummy board provided by the EE team to open a UART (Universal Asynchronous Receive Transmit) connection via USB to interface with the application.

5.1.7 Read in data from UART

After opening a UART connection, Von Bergen worked to read live data from the dummy board and tested its functionality of sending and receiving data.

5.1.8 Save data to CSV

Harvey once again utilized Pandas in order to add file saving functionality to the data manager. This addition involves storing data from the communication controller in a Pandas dataframe for the sole purpose of writing the data to a CSV. Pandas has a built-in function for doing this, similar to the function used for creating a dataframe from a CSV.

5.1.9 Live graph data

Edens tweaked the graph manager class to allow plotting to the graph as data comes in, rather than needing the full data set before the plot is updated. This required an update to how the plot object is initialized.

5.1.10 Customize graphs

Simmons and Edens worked to create a menu that allows the user to overlay/select any desired data sets. They also implemented functionality for creating multiple views to further isolate the data being shown.

5.1.11 Start test from computer

Once live viewing of all relevant data was achieved, every member worked to initiate the several tests, outlined earlier in this proposal, from the application.

5.1.12 Board Communication Protocol

Harvey worked with the ELEG team to establish a communication protocol between the board and the desktop application. Eventually, it was decided that data would be sent over UART using ASCII encoding. Commands are sent from the application with a single ASCII character followed by a carriage return. The board sends back confirmation signals to let the application know that the command was received correctly. When a test is started, raw data from analogue to digital converters on the board is sent to the application, with roughly 14 milliseconds between each data point.

5.1.13 Create installable executable

After all of the above tasks were completed and the minimum viable product was achieved, Simmons created an installer to permanently install the program to the system. This installs the application to a permanent directory for each OS and generates a launcher entry (Start Menu, Launchpad, etc.) for our app. An icon was also designed for the program.

5.1.14 Finalize Documentation

Once the application had been finalized and no further changes were planned, all members collaborated to create a comprehensive and easily understood documentation to ensure any user will be able to fully operate the application in a lab environment.

5.2 Schedule

Task Description	Assigned Members	Est Start	Est End	Actual Start	Actual End
Create Project	Luke Simmons	11/12/24	11/17/24	11/12/24	11/17/24
GUI Opening	Luke Simmons and Gavin Edens	1/20/25	1/24/25	1/21/25	1/25/25
Generate Sample Data	Kile Harvey	1/20/25	1/24/25	1/21/25	2/3/25
Loading Data from CSV	Kile Harvey	1/27/25	1/31/25	1/27/25	2/6/25

Graphing with Sample Data	Luke Simmons and Gavin Edens	2/3/25	2/14/25	2/4/25	2/21/25
UART Communication	Carl von Bergen	2/3/25	2/7/25	2/6/25	2/21/25
Read in data from UART	Carl von Bergen	2/10/25	2/14/25	2/22/25	3/21/25
Save Data to CSV	Kile Harvey	2/17/25	2/21/25	2/7/25	2/10/25
Live Graph Data	Luke Simmons and Gavin Edens	2/24/25	3/7/25	2/21/25	3/13/25
Customize Graphs	Luke Simmons and Gavin Edens	3/10/25	3/14/25	3/17/25	4/14/25
Start Test from Computer	All Members	3/17/25	3/28/25	4/14/25	4/23/25
Board Communication Protocol	Kile Harvey	N/A	4/4/25	3/31/25	4/14/25
Create Installable Executable	Luke Simmons	3/31/25	4/4/25	1/15/25	4/23/25
Finalize Documentation	All Members	4/7/25	4/28/25	4/14/25	4/23/25

5.3 Deliverables

- Architecture overview: A diagram of how different parts of the application connect together. Includes detailed descriptions of each part.
- Python code: Our entire project will be programmed in Python. This is split into multiple modules with major sections including communication control, data management and user interface.
- Installation and usage documentation
- Sample data: This will be a CSV file with test data.

6.0 Key Personnel

Gavin Edens – Edens is a senior Computer Science and Computer Engineering major in the Electrical Engineering and Computer Science Department at the University of Arkansas. He has experience working at a local MSP handling IT services.

Kile Harvey – Harvey is a senior Computer Engineering major in the Electrical Engineering and Computer Science Department at the University of Arkansas. He is currently an hourly research assistant in Dr. Jia Di's research lab, "TruLogic." He has experience in FPGA development and microcontroller implementation, along with desktop application development.

Luke Simmons – Simmons is a senior Computer Science and Computer Engineering major in the Electrical Engineering and Computer Science Department at the University of Arkansas. Currently, he is an Undergraduate Researcher in Dr. Chris Farnell's RIOT Lab, focusing on cybersecurity. He has also interned at Walmart doing software development and at a local MSP doing IT services.

Carl von Bergen – Von Bergen is a senior Computer Science major in the Electrical Engineering and Computer Science Department at the University of Arkansas. He is currently taking Computer Networks with Dr. Dale Thompson, Computer Graphics with Dr. John Gauch, and Information Security with Dr. Chris Farnell.

Zhong Chen – Dr. Zhong Chen is an associate professor of the Electrical Engineering and Computer Science department here at the University of Arkansas, along with the sponsor of our project.

Robert Saunders – Robert Saunders is a faculty member of the EE department. We have collaborated with him to better understand the scope of this project and ESD latch-up itself. He also assisted us in acquiring a dummy board from the EE team for testing purposes.

Electrical Engineering Team: Logan Gentry, Peter Brinkman

Dallas Blank – Blank is a graduate student of Dr. Chen, and is currently standing in for Dr. Chen in sponsor duties. He was a member of the senior design team that this year's project is based off of, and has been providing guidance on the project, along with signing off on proposals and reports in order to keep both teams on schedule.

Harshdeep Singh – Singh was also a member of the previous academic year's senior design team and has provided guidance on the project.

Industry Professionals: Scott Ward (Texas Instruments), Robert Gauthier (IBM), Nate Peachey (Qorvo), Kathy Muhonen (Qorvo), Hang Li (Huawei), Shih-Hung Chen (AMD)

7.0 Facilities and Equipment

Pictured in figure 6 (below), is the "dummy" board that the Computer Science team was supplied with. This board was used to simulate actual communication between our application and the Electrical Engineering team's finished board. In short, the board simply retransmitted any message that it received back to the sender. This allowed our team to determine if messages that were sent over the UART connection were what we expected to be transmitting. This also allowed our team to make significant progress toward a finished product, independent of the Electrical Engineering team's progress.

Pictured in figure 7 (below) is the finalized version of the mainboard and daughterboard created by the Electrical Engineering team.

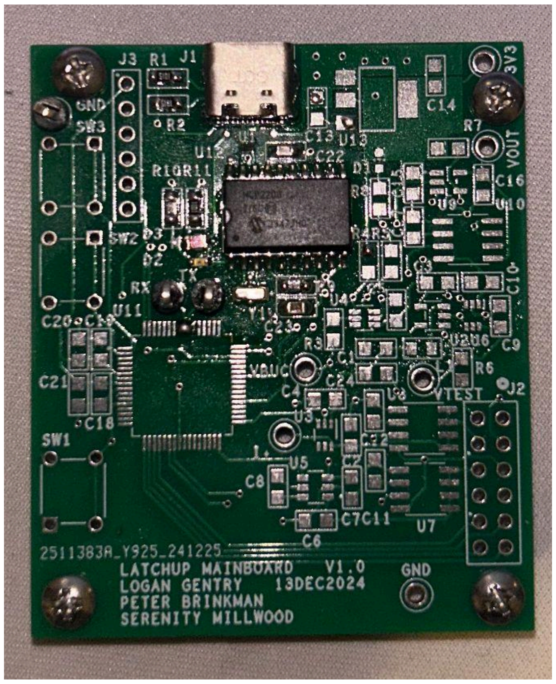


Figure 6: Dummy board for communication testing

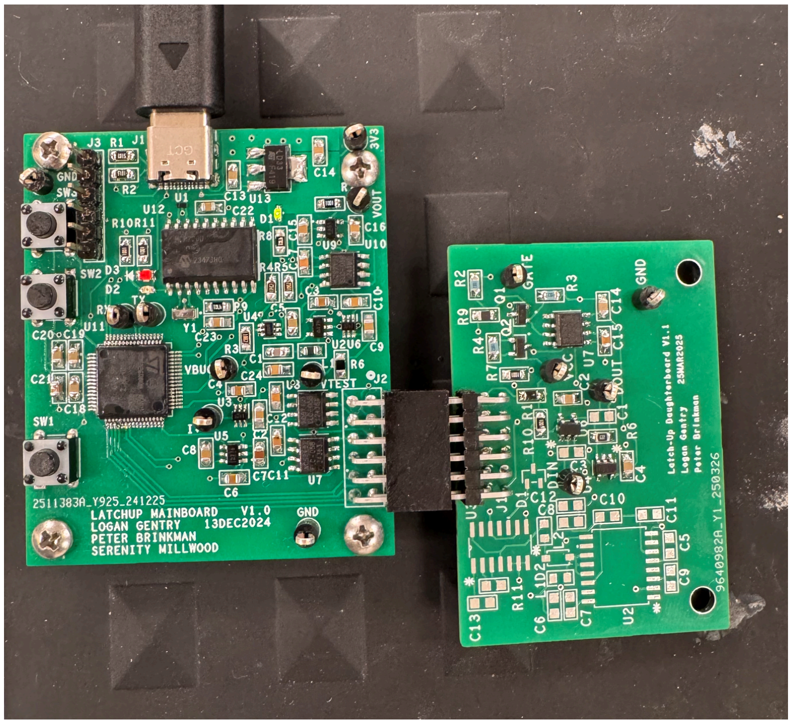


Figure 7: Mainboard (left) and daughterboard (right) plugged into computer

8.0 References

[1] E. Haseloff, “Latch-up, ESD, and other phenomena,” Texas Instruments Inc., SLYA014A, 2000. Available: https://www.ti.com/lit/an/slya014a/slya014a.pdf

[2] JEDEC, “JEDEC STANDARD IC Latch-Up Test,” JEDEC Solid State Technology Association, JESD78F.02, November 2023. Available: https://www.jedec.org/standards-documents/docs/jesd-78b

[3] R. Sheldon, “Complementary metal-oxide semiconductor (CMOS),” TechTarget, August 2022. Available: https://www.techtarget.com/whatis/definition/CMOS-complementary-metal-oxide-semiconductor

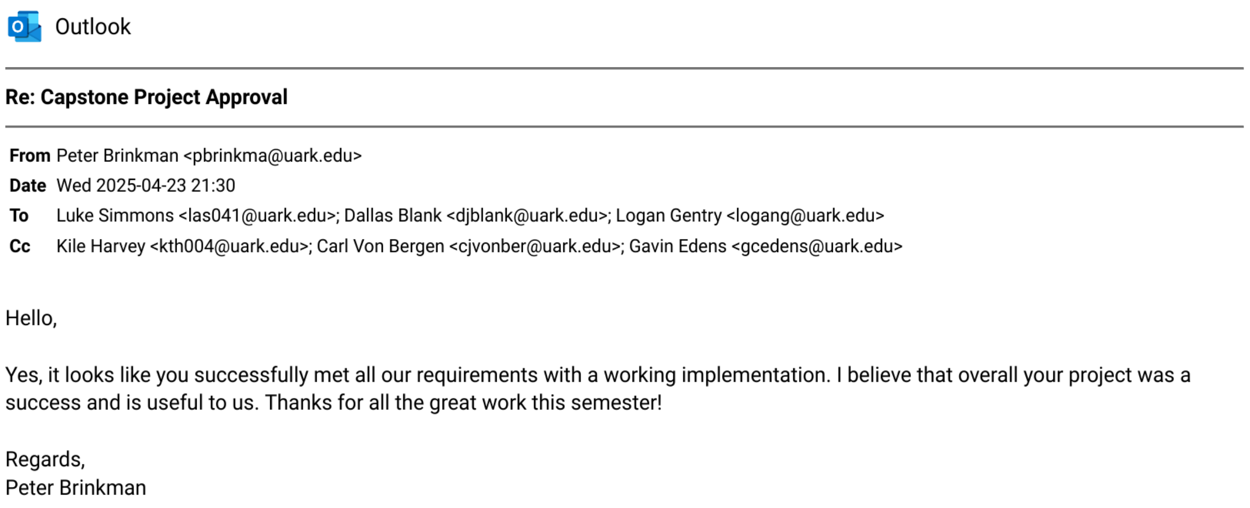
[4] P. Kirvan, “Electrostatic discharge (ESD),” TechTarget, March 2023. Available: https://www.techtarget.com/whatis/definition/electrostatic-discharge-ESD

[5] Robson Technologies Inc., “Are you overlooking latch-up?” February 25, 2021. Available: https://www.testfixtures.com/are-you-overlooking-latch-up/

[6] K. Söderby, “Using the serial plotter tool,” Arduino, September 12, 2024. Available: https://docs.arduino.cc/software/ide-v2/tutorials/ide-v2-serial-plotter/

[7] Electric UI, Available: https://electricui.com/features

[8] Pandas Software, Available: https://pandas.pydata.org/



Outlook

Re: Capstone Project Approval

From Peter Brinkman <pbrinkma@uark.edu>
Date Wed 2025-04-23 21:30
To Luke Simmons <las041@uark.edu>; Dallas Blank <djblank@uark.edu>; Logan Gentry <logang@uark.edu>
Cc Kile Harvey <kth004@uark.edu>; Carl Von Bergen <cjvonber@uark.edu>; Gavin Edens <gcedens@uark.edu>

Hello,

Yes, it looks like you successfully met all our requirements with a working implementation. I believe that overall your project was a success and is useful to us. Thanks for all the great work this semester!

Regards,
Peter Brinkman

Figure 8: Approval email from sponsor team