Instruction Level Reverse Engineering (Disassembly) through EM Side Channel

Final Report

Team Number: 09
Client: Prof. Akhilesh Tyagi
Advisers: Varghese Vaidyan

Team Members:
Matthew Campbell
Noah Berthusen
Cristian George
Jesse Knight
Jacob Vaughn
Evan McKinney

sdmay21-09@iastate.edu
https://sdmay21-09.sd.ece.iastate.edu

Revised: 4/25/2021

Executive Summary

Development Standards & Practices Used

- Agile Scrum model used
- IEEE UART protocols
- Python PEP 8 Style
- Arduino Style Guidelines
- IEEE Code of Ethics

Summary of Requirements

- Program collects EM data and converts it to a usable format
- Model will predict opcodes with 90%+ accuracy and operands with 80%+ accuracy.
- Written in Python
- Well-documented code
- Predictions are formatted in a user-friendly format
- Large amount of data used to train the model

Applicable Courses from Iowa State University Curriculum

- COM S 311 (Introduction to the Design and Analysis of Algorithms)
- COM S 474 (Introduction to Machine Learning)
- CPRE 288 (Embedded System I)
- CPRE 381 (Computer Organization and Assembly Level Programming)
- CPRE 482x (HW Design for Machine Learning)
- EE 224 (Signals and Systems I)
- EE 321 (Communication Systems I)
- EE 201 (Electrical Circuits)
- EE 230 (Electronic Circuits and Systems)

New Skills/Knowledge acquired that was not taught in courses

- Convolutional neural networks (CNNs), Markov chains
- Side-channel observation of processors

Table of Contents

1 Introduction

1.1 Acknowledgement

We would like to thank Varghese Vaidyan for sharing his knowledge about working with the EM side channel for reverse engineering, and for sharing his equipment and experience with us in order to jump start the project. Additionally, we would like to thank ETG for letting us check out an oscilloscope for the entire Fall semester allowing us to begin long-term data capture.

1.2 Problem and Project Statement

General problem statement:

We want to be able to determine the assembly level code currently running on a processor by only reading the electromagnetic (EM) radiation that comes off of the processor. This kind of research has cyber security implications in that you could bypass systems if you could find out what code is running by observing the physical EM radiation from the processor.

General solution statement:

Our solution is to capture data using an electromagnetic probe and send that data to a machine learning algorithm. The machine learning algorithm will be able to look at the data and the surrounding data points to determine with a degree of certainty what opcode and operand is being executed.

1.3 Operational Environment

The resulting end product from this project will be used in a laboratory environment with minimal electromagnetic interference. At the moment, the operational environments have been 301 Durham, the TLA, and Jesse Knight's apartment as all environments have access to an oscilloscope with the necessary specifications.

1.4 Requirements

Requirements:

- Program collects EM data and classifies signals into executing instructions with opcodes and operands
- Model will predict opcodes with 90% accuracy and operands with 80% accuracy.

Non-functional requirements:

- Written in Python
- Well-documented code
- Predictions formatted to be user-friendly
- Large amount of data used to train the model

1.5 Standards and Constraints

694-1985 - IEEE Standard for Microprocessor Assembly Language

- Defines a common set of instructions for a microprocessor assembly language and establishes a naming convention.
- Instruction names must begin with a verb of action, and for the shortened version the mnemonic must begin with the same first letter of that verb.
- The standard covers arithmetic, logical/shift, data transfer, control transfer, and miscellaneous instructions. The standard also covers operand specification and addressing directives.

While working with our microprocessor, it's important to keep a consistent assembly language when training and testing the machine learning model. Different processors run on slightly different assembly languages and the different opcodes would cause conflicts with the model.

P26531 - Systems and Software Engineering -- Content Management for Product Life Cycle, User, and Service Management Documentation

- This standard is about efficient and effective development of a software product for software manuals, style guides, design documents, project management plans, testing plans, etc.
- The purpose is to define expectations for content management and the industry standard practices that define usage of it.

Using Agile software development practices allows the team to develop code based on sprints and allows the team to focus on whatever tasks are most important that week. In addition to GitLab, the Agile methodology is a good management practice to ensure that work is not being overwritten or forgotten for that sprint.

Constraints that we faced throughout this project:

- Budget:
  - We were limited to a budget of $100 which was intended to be used to purchase both a micro-controller and electromagnetic probe.
    - Micro-controller ~$20
    - Consumer electromagnetic probe ~$100+
      - Built our own probe instead.
- Minimum pipeline-size of 4 with an ARM M4 or M7 processor
  - Both ARM processors have a pipeline size of 6 meaning that the complexity of our data capture had increased due to more pipeline stages
- Oscilloscope bandwidth of 100 MHz:
  - Oscilloscopes available to us had operating frequencies at or lower than the nyquist frequency necessary to measure boards at stock configurations.
    - Downclocked processor to avoid aliasing.
- Covid-19 Pandemic:
  - Due to the global pandemic we were restricted on how we could meet.
  - Could not meet in-person while on-campus without reserving 6 stations in the TLA.

- Hardware testing and data collection placed a burden on the team member who had the board and oscilloscope available to them

1.6 Intended Uses and Users

Our single intended use is to measure EM radiation from a Cortex ARM M7 processor and output the corresponding opcodes and operands. The intended user for our project is our client Akhilesh Tyagi and other EM side-channel researchers.

1.7 Assumptions and Limitations

Assumptions:

- End users will have access to the necessary hardware and software
- Design is running on a Cortex ARM M7 processor with a 6 stage pipeline running at 20 MHz
- EM is measured using same tools used to train the machine learning model

Limitations:

- Program will need to run on a high-end GPU
- Input data to model must be in a specific format
- Processor must have at least a 4 stage pipeline

1.8 Security Concerns and Countermeasures

There are some security concerns to consider with this project. First off, since this is a part of a larger research project about processor side channels, information about our project should not be publicly shared yet.

Also there is security risk posed by the very existence of this project. Since the whole idea of the project is to collect data about what is happening on the processor without needing any digital access to the mechanics internals. The only thing standing between this project and the processors data is physical security.

1.9 Expected End Product and Deliverables

1. A machine learning algorithm capable of 90% opcode:

   The machine learning algorithm will be delivered at the end of the project, approximately May 2021. The algorithm will be created using python and will include the datasets used to test, train, and validate the algorithm. Additionally, the algorithm will include documentation describing how to redeploy the algorithm to a separate system.

2. Automated data extraction tool to pull data from EMR probe.

The extraction tool will take data from either the oscilloscope or digitizer and convert it into a format usable by the machine learning algorithm. This tool will allow the client to easily extract additional data to run against the trained algorithm. We expect to have this data extraction tool to be completed by November 2020, but will be delivered to the client with the machine learning algorithm in May 202

2 Design

2.1 Previous Work and Literature

Several previous works have investigated electromagnetic side channel information leakage. Works such as *Electromagnetic Emission Measurement of Microprocessor Units* (Maćkowski) have investigated what signals can be measured from the power supply lines. This is slightly different from our project, where we are directly measuring the microprocessor for signals. More closely related to our project is the work *Electromagnetic Side Channel Information Leakage Created by Execution of Series of Instructions in a Computer Processor* (Yilmaz). However we make the distinction that in that work, while signals were being collected, there was no effort to recreate the actions of the processor. This is where our project expands on previous work; we are measuring the electromagnetic radiation from a microprocessor and attempting to recreate the executed assembly by using machine learning.

The task of identifying opcodes and operands as they move through a processor is no easy task. Since our chosen processor has a six-stage pipeline, this means that the EM signals can be a combination of six opcodes/operands. Including the temporal locality of the opcodes in our model will be essential in getting good results. The work *Convolutional neural network-based hidden Markov models for rolling element bearing fault identification* (Wang) is a fairly close example of what we must do, although the input data will be more complex and mixed up.

2.2 Proposed Design

We have proposed to collect data from an STM32H743 nucleo-144 board to observe during operation. The board runs at 480 MHz but can be downclocked below 20 MHz. The flexible clock rate of our board allows us to fit within the 100 MHz constraint set by our current project specifications. Additionally, we are using an EM probe built by the team. The oscilloscope we plan on using is an Agilent DSOX2024A which has an operating bandwidth of 200 MHz. The chosen oscilloscope and probe will allow us to observe the electromagnetic radiation without aliasing the signals. The combination of hardware being used will let us observe an ARM M4/M7 class processor at a frequency of at most 100 MHz and collect the observed data for future use in machine learning.

As for software, we plan on using Google's machine learning framework, Tensorflow, to design a neural network capable of identifying both the OPCode and Operand being executed by our processor. The design of the neural network will change as we refine it to achieve greater accuracy, but Tensorflow will allow us to validate, profile, and benchmark our neural network.

2.3 Technology Considerations

- Oscilloscope

    - There are many oscilloscopes to choose from, but we will have to work with what is in the lab that is available to us. The Tektronix DPO3012 is the model we have been given to perform data collection.
- EMC Probe
    - Purchasing an EMC probe will cost a couple hundred dollars to purchase. While much higher quality, the price makes this unrealistic.
    - Creating an EMC probe out of semi-rigid coax cable allows for a cheaper alternative, but is not quality controlled and provides no method for calibration.
- Development Board Selection
    - There are a variety of different development boards, but since we are not taking advantage of the different features that they have, any microcontroller such as the Nucleo STM743HZI2 that allows us to configure the clock frequency will work.
- IDE Selection
    - STMCubeIDE is the free IDE that is used to program the Nucleo boards. It is a C or C++ environment that requires creating functions on the simplest level. While it gives us direct access to the board and all set up functions, it is very in depth and requires significant embedded systems knowledge which isn't the goal of this project.
    - The Arduino IDE is another choice that has an open source library which allows it to control ARM based microcontrollers rather than AVR based ones with a variety of prewritten
    - Visual Studio Code and Jupyter Notebooks will be useful in prototyping the machine learning models. When running on the GPU cluster, individual scripts will be written
- Machine learning framework
    - Tensorflow, Keras, and Scikit-Learn are well suited for convolutional neural networks and have lots of flexibility when defining machine learning models.

2.4 Design Analysis

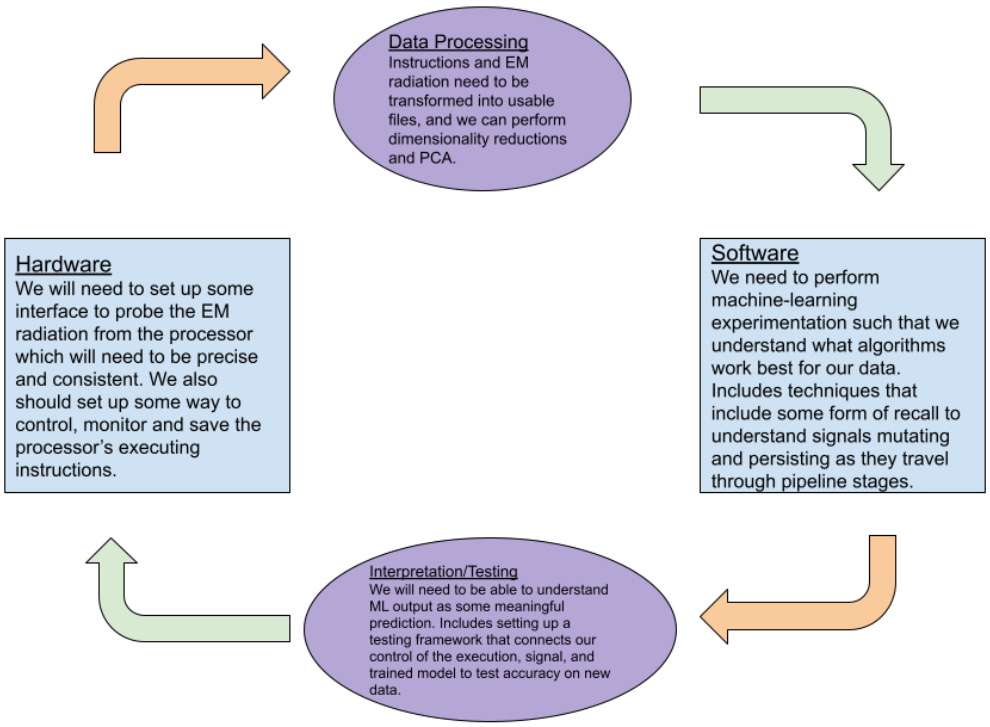
We ended up purchasing the Nucleo board for our testing, and controlled it using the Arduino IDE with the homemade EMC probe. The homemade EMC probe was able to capture data from the processor, however the magnitude of the probe was extremely low, and may require an amplifier in the future to capture more accurate data. Other things that were noticed were that other parts of the board were extremely noisy and created undesirable data that could ultimately affect the machine learning. The machine-learning should be able to pick out this data, but it may be important to measure from many different locations along the board to ensure that the various noise sources are considered when the machine learning runs.

2.5 Development Process

Our project's development process is a combination of Waterfall and Agile. The hardware side of our project mainly uses the Waterfall model as tasks need to be sequentially completed in order to work on the next task in line. On the software side of development we use Agile. Agile gives us a flexible and responsive approach to the task at hand as the tasks do not need to be sequentially completed and their importance varies as certain tasks are progressed. Using the Waterfall model

to ensure continuous hardware progression and the Agile model to flexibly and reactively respond to problems keeps productivity and progression at a maximum.

2.6 Design Plan



Our use cases laid the groundwork for the shape and direction of our design plan. Because of our design requirements we have chosen to use complex software such as ML in order to achieve the necessary accuracy to meet design requirements. Our requirement and use cases come from the software module connected to the testing block since that is where we define the accuracy benchmarks. The interaction of the hardware, data analysis, and software modules allow the user to interact with the application, satisfying the requirements and use cases.

2.7 Design Evolution

While the design has not changed significantly since the previous semester, the hardware that we are using has. We are still using the Nucleo-144 board as our electro-magnetic radiation generator, and the same antenna that was previously built. However, halfway through March, 2021 ETG requested we return the oscilloscope we were using for data acquisition and gave us another oscilloscope of a completely different make and model. The new oscilloscope has different

specifications from the previous model we were using and resulted in multiple revisions of our previously function data-capture framework.

Additionally, we have expanded data capture capabilities by expanding the data-length of our UART messages allowing for more permutations of data to be captured during a single run. Aside from quality-of-life changes for data capture allowing for more robust data sets the only major change in our design has been the change in oscilloscope and modifications that are required in our framework.

3 Testing

3.1 Unit Testing

Software: test a handful of different configurations and classification models.

Hardware: test accuracy of signals from nop collection, then for other opcodes and operands.

Acceptance testing for requirements: classification meets accuracy requirement.

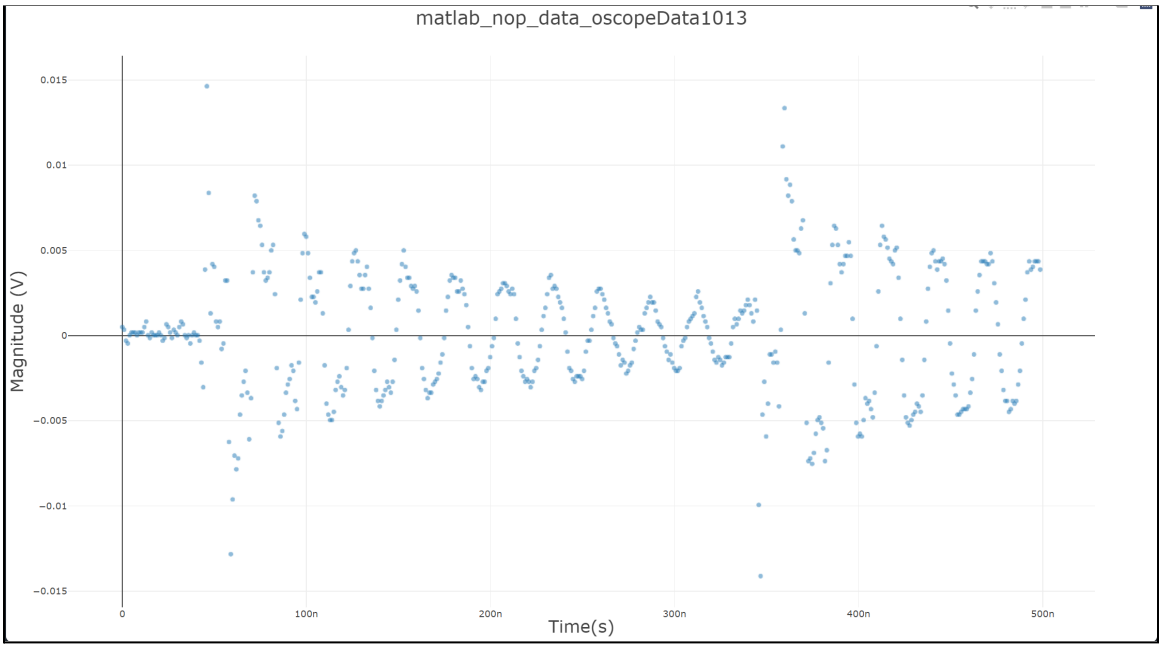
Unit testing: we will test our automated data collection schemes that make sure communication between Matlab, ARM board, and oscilloscope are working and data is collected correctly

3.2 Acceptance Testing

The client will be involved in live testing of the design in action. Design requirements are 90% accuracy for opcode so we will just use 80/20 split for training and verification. Demonstrating to the client that operations on the ARM processor match those observed and processed by our design.

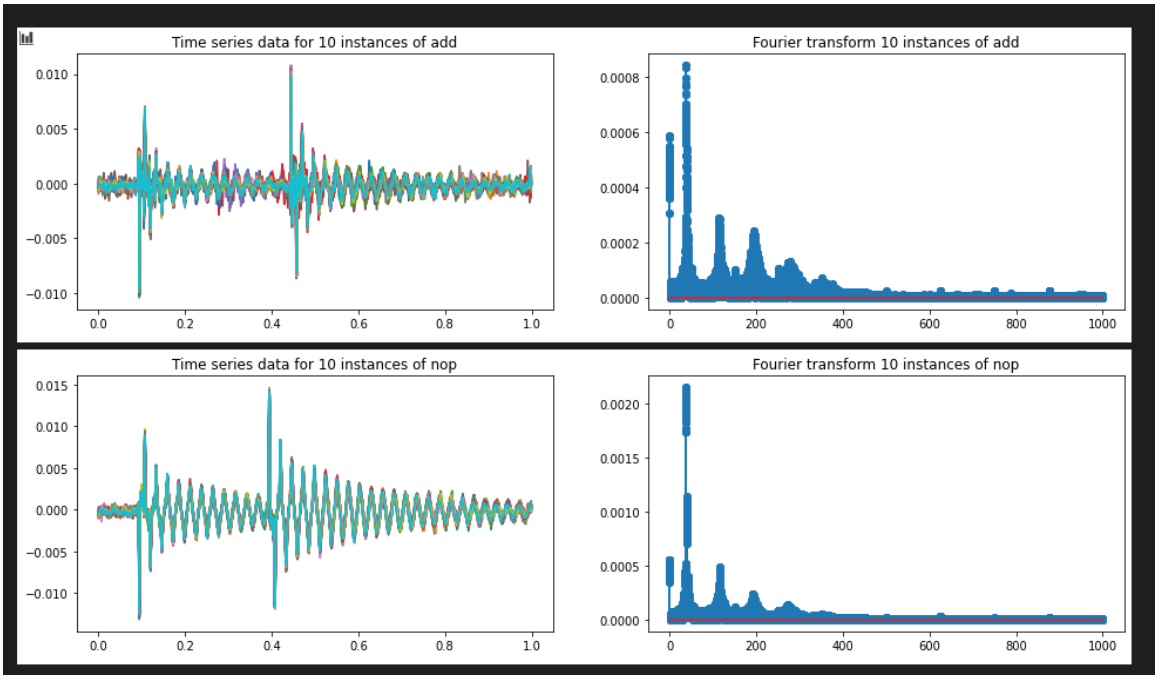
3.3 Results

We have been able to collect data from the microprocessor, view it in the oscilloscope, and save it in Matlab. Below is an example plot of oscilloscope data. The next steps are to collect data for several opcodes and begin to design a machine learning framework. The collected graph shows the EM radiation coming from a series of NOPs in the processor. This was done without a mount for the probe, so the results will be inconsistent, but provides a good example of what we might expect from data.



Example oscilloscope data of EM radiation caused by NOP opcodes

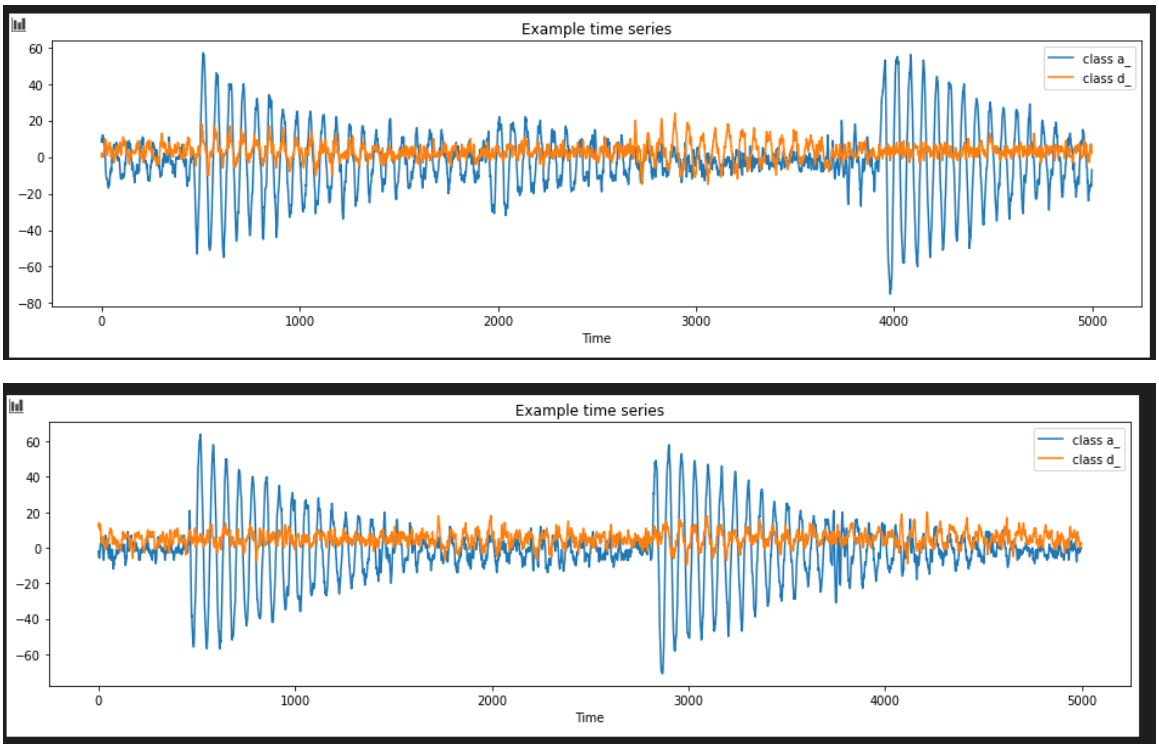
After we were able to get data from a single block of executed assembly, we began working with assembly code to build our database of samples for machine-learning. Below is a comparison of the signals add vs nop. When we were first collecting data, we were unable to get such a clear distinction between these two instructions. We needed to find a sampling resolution that didn't overburden our machine-learning with too large of feature vectors, but also clear enough to make distinctions between our labels. Furthermore, we needed to solve an issue we had with the timing not lining up consistently so similar looking waves were not nicely on top of each other like as shown below.



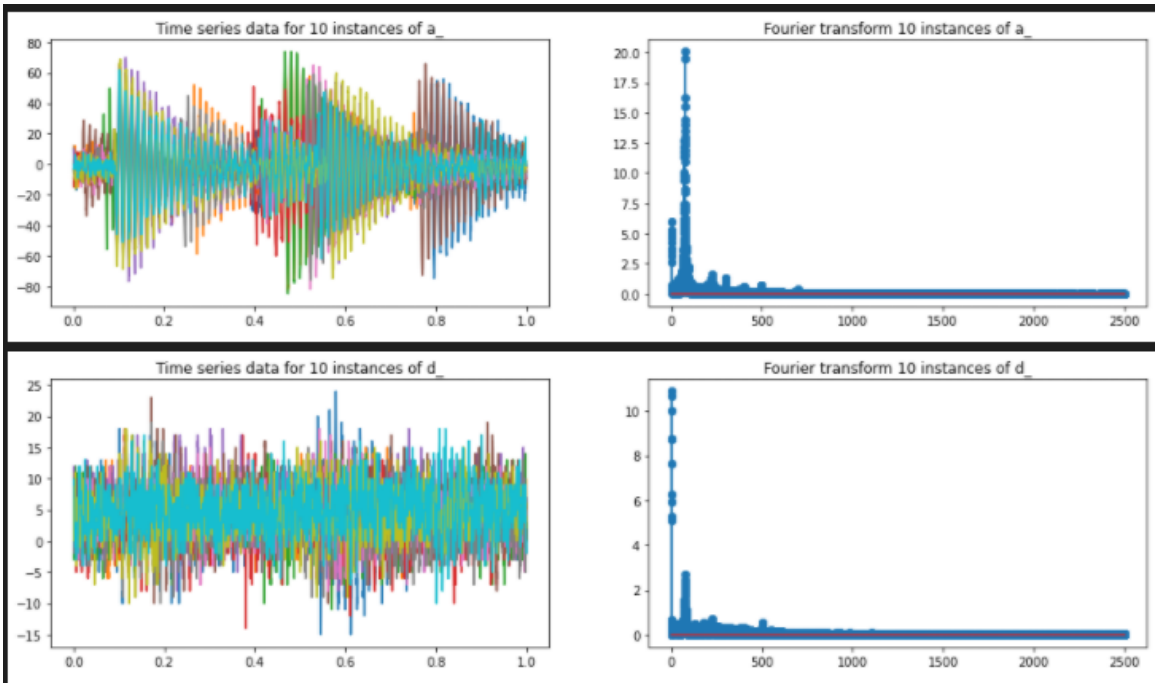
We used the Fourier transform to experiment with different representations of our data. For both single instruction and multiple instruction classification, we successfully train a random forest classifier and a time series forest classifier using mean and standard deviation features extracted from random intervals of the time series data. The Fourier transform ends up not being necessary as we achieve sufficient accuracy using the time series data itself.

For each possible label in the classification task, 10,000 time series samples are created by polling our EM probe 5000 times over 12 of the processor's clock cycles. To limit the scope of required data collection, we focused on classification between different functional groups from the STM32 instruction set, i.e. Memory access instructions and General data processing instructions. Using a selection of instructions that vary most from another allows us to achieve up to 98% classification accuracy for both single and multi class tasks.

After finding success with single instruction classification, we began working on ways to classify instructions where the training data contains multiple instructions in it. The challenge of a pipelined processor is that for an instruction being executed in a program, the EM radiation will always have unpredictable contributions since we don't know the other instructions and where they are in the pipeline stages. One of the ways we worked on moving forward was to create signals that contain random permutations of instructions and test our ability to identify if a given instruction is present.



Here are two different time series samples collected from a permutation of add and mul and add and asr instructions. The two different series are clearly different from each other, likely because the main contribution to EM radiation comes from the hamming weight of signals in the processor, and the hamming weight will be very different for instructions in different functional groups. However, by permuting the instructions, we need to be able to recognize features of waves independent of time, e.g. the two large peaks in class A begin at 3000 and 4000 points in each. This is one of the benefits of using the random interval decision tree.



Multiclass classification for our set of instructions from the functional group reaches over 98% accuracy. However, we were unable to train a model that contained a complete dataset from instructions from all functional groups due to limitations in computational resources.

4 Implementation

As we conclude this semester our implementation is as follows:

Hardware capture:

1. Nucleo-144 is fitted into a custom probe enclosure to ensure consistency.
2. Electromagnetic probe is connected to Tektronix DPO3012
3. GPIO_B Pin 4 of Nucleo-144 is connected to oscope
4. Python generation script is ran and then contents of the resulting text file are loaded into the case statements of embedded-c code.
5. Nucleo-144 is flashed with updated embedded code.
6. MatLab script is launched triggering data capture of multiple permutations of data.
    a. Results captured are processed into a format usable for machine learning.
7. Newly capture .csv files are pushed to GitHub for the machine learning team to process.

Machine learning:

1. Data is processed to combined from files, labeled and placed into a single pandas dataframe
2. Fully-processed data is ran through our most recent model
    a. As of the writing of this paper the latest model is an ensemble learning method consisting of random forests. The next stages of our project would be to combine

layers of models to classify between instruction functional groups and then more specifically inside of that group. We would have needed to automate the data collection more to achieve this much collection, and our models would likely take too much training to finish in our time scope.

We are using the same Nucleo-144 board we purchased last semester to run assembly sequences through it. The electromagnetic-radiation emitted from the development board is captured using the self-built probe and an oscilloscope. The data is processed using a MatLab program which also handles the oscilloscope/development board triggering using UART and outputs the collected data into a suitable format for machine learning.

The newly collected data is then used to run through the current machine learning model that we are evaluating. Throughout the project we have evaluated different machine learning approaches including multi-class, multi-label, and single classification. We evaluated different machine learning methods and have most recently had success with a random-forest ensemble learning approach.

We used a development branch of the Python package sktime in combination with scikitlearn to write our machine learning algorithms. We experimented with modifying the default time-series forest classifier by introducing a pipeline which combines feature importances for the training data as a function of time to make more well-informed choices in the decision tree but was unable to complete this due to limitations in computational power.

5 Closing Material

5.1 Conclusion

At this moment we have configured our microcontroller to work with the STMCube IDE and have downclocked the processor to meet our update project requirement of data capture at a frequency of at least 20 MHz. This was done as we could not find access to a oscilloscope with a high enough bandwidth to avoid aliasing. We are currently capturing EM data from the board to use as part of our data set for machine learning.

We were tasked with capturing data using an electromagnetic probe with an ARM processor operating with a minimum of a four-stage pipeline and running at a frequency of at least 200 MHz. Data captured from our device is to then be sent to a machine learning algorithm that will reach 90%+ accuracy when detecting opcodes.

We had to modify our requirements as we did not have access to a high bandwidth oscilloscope but otherwise have shown that our current design is capable of high-accuracy classification. Using the automated data-capture scripts for the oscilloscope we have been able to show multi-class classification of opcodes using only the electromagnetic data captured with an accuracy of 100%. We are still experimenting with other machine learning methods that would provide for more robust assembly strings as we expand our classification algorithm to support longer strings that contain a wider range of instructions.

There are other data hazards to consider which we have not sufficiently analyzed, the primary one being the effect our processor's pipeline has on the data we collect. As the ARM Cortex M7 has a 6 stage, dual-issue pipeline we have disabled compiler optimizations to reduce the likeli-hood of the second execute pipeline being used in runtime. However, as we begin to analyze more complex instruction sequences the data captured at any given point will be electromagnetic radiation emitted by a unique instruction in every separate state of the pipeline. We must take this into account as we expand the sequence complexity of instructions we analyze.

While we were unable to expand our classification algorithm to support the entire ARM instruction set, which consists of over 200 unique instructions, we believe that we could continue expanding on supported instructions given more time.

5.2 References

B. B. Yilmaz, M. Prvulovic and A. Zajić, "Electromagnetic Side Channel Information Leakage Created by Execution of Series of Instructions in a Computer Processor," in IEEE Transactions on Information Forensics and Security, vol. 15, pp. 776-789, 2020, doi: 10.1109/TIFS.2019.2929018.

Andrzej Kwiecień, Michał Maćkowski and Krzysztof Skoroniak Series: Communications in Computer and Information Science, Year: 2012, Volume 291, Page 191

Maćkowski, Michał & Skoroniak, Krzysztof. (2009). Electromagnetic Emission Measurement of Microprocessor Units. 39. 103-110. 10.1007/978-3-642-02671-3_12.

Shuhui Wang, Jiawei Xiang, Yongteng Zhong, Yuqing Zhou, Convolutional neural network-based hidden Markov models for rolling element bearing fault identification, Knowledge-Based Systems, Volume 144, 2018, Pages 65-76, ISSN 0950-7051, https://doi.org/10.1016/j.knosys.2017.12.027.

https://www.keysight.com/upload/cmc_upload/All/2000_series_prog_guide.pdf
https://www.st.com/resource/en/reference_manual/dm00314099-stm32h742-stm32h743-753-and-stm32h750-value-line-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf
https://www.st.com/resource/en/datasheet/stm32h743zi.pdf
https://www.st.com/resource/en/programming_manual/dm00237416-stm32f7-series-and-stm32h7-series-cortexm7-processor-programming-manual-stmicroelectronics.pdf
https://www.st.com/resource/en/user_manual/dm00629856-description-of-the-integrated-development-environment-for-stm32-products-stmicroelectronics.pdf
https://github.com/stm32duino
https://www.eevblog.com/forum/blog/eevblog-1178-build-a-$10-diy-emc-probe/

Appendix I: Operations Manual

- Generate switch cases statements
    - In the root directory of the git repository navigate to the python folder
    - Edit the generation.py script:
        - If a different opcode needs to be in the series then change the assignment of 'opcode1' or 'opcode2' to a string containing valid and syntactically correct ARM Assembly Code.

        - If you are adding more instructions, create a new opcode variable containing and ARM assembly string and concatenate into the 'opcodes' string
        - Edit the first parameter of the 'with open' statement at the end of the file to more accurately describe the new instruction sequence you are generating.
        - Save all changes
    - Run the generation script
        - Using Python 3
            - If you do not have python3 it can be download from https://www.python.org/downloads/
            - Run the script in the terminal or using your preferred python IDE
                - Terminal: "python3 generation.py"
                - IDE: Hit the run button on the IDE window
            - Wait for the program to exit
    - Locate newly outputted file and move-on to the next step
- Load new switch case statement into STMCube IDE
    - Copy and pasting raw text works fine
    - Generally fold switch statement
- Upload code to Nucleo Board
    - Using run or debug will put code onto processor
    - Can also generate compiled binary and place into the board's storage device
- Connect probes to oscilloscope
    - Standard 1x probe connected to channel 1, make sure attenuation is set properly on scope
    - ECM probe connected to channel 2
    - Matlab code will automatically set all measurement parameters, so there is no need to adjust them manually prior to collection
- Install board and probes into custom mount
    - Nucleo board will simply drop into place
    - Adjust probe mount placement
        - Vertical adjustment installs via two M3 screws and nuts into horizontal adjustment slot
        - EMC probe can be mounted via two zip ties
        - Rotational adjustment can be then attached via one M3 screw in vertical adjustment slot
        - CAD available for reference
    - Insert probe into CN12 pin3 and connect to ground pins
- Connect board and oscilloscope to computer running Matlab
    - Nucleo board connects via microUSB in CN1
    - Oscilloscope connects via a standard USB type B printer cable
- Load Matlab code and prepare for data collection
    - The Matlab program requires information about board and oscilloscope in order to run properly
        - Line 13 under 'Setup' requires path to save data
        - Line 20 under 'Serial Communication Setup' requires the serial COM port that the nucleo is attached to

        - Line 39 under 'Interface Configuration' requires the VISA address of the oscilloscope which can be found following the oscilloscope manual
- Run Matlab code
    - Pressing run in the Matlab environment will take a few moments and then open a dialog box for run options
        - Number of runs
            - The number of times that the oscilloscope will
        - Name of file
            - Usually the types of opcodes that are currently being tested
    - Data collection for a standard ten thousand runs will take approximately half an hour
    - Once finished, the Matlab program will automatically trim any failed data collection runs and save the data file in a CSV file in the specified path
    - The program will also save a CSV file that contains the order of the case statements that were run
- Upload CSV to Gitlab
    - Once the data and order files have been saved, they can be pushed to the repository so that the software team has access
- Train machine learning model
    - Download CSV from Gitlab if not yet acquired
    - Write the correct file path to all CSV data. This will work through all files in the directory and turn them into a single pandas dataframe
        - Can specify whether or not to use Fourier transform to augment the data
        - Libraries such as Matplotlib, Numpy, Pandas, and Scikit-Learn can be installed using pip or Anaconda in a virtual environment.
        - https://pypi.org/project/pip/ contains all necessary libraries and instructions
    - Run the jupyter notebook to train a classification model
        - Jupyter notebook explorer is needed to edit these types of files. IDEs such as Visual Studio Code also have built-in support for .ipynb filetypes
    - This model can be saved to be used later during testing
- Using trained model to predict opcodes
    - Use same process as training model to load in oscilloscope CSV data into panda dataframe
        - Use same format as when training model, i.e. be sure to Fourier transform the data if the model expects it
    - Instead of training, have the model predict opcodes by calling model.predict and passing the pandas dataframe
    - Accuracy can be tested using Scikit-Learn or by hand

Appendix IV: Code

main.c

```
//void USART3_IRQHandler(void)

//{

//  HAL_UART_IRQHandler(&huart3);

//}

UART_HandleTypeDef huart3;

uint8_t data_r[2];

int data = 0;

uint8_t *point = data_r;


void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)

{

    if (huart->Instance == USART3)

    {

       data = data_r[0] + (data_r[1]<<8);

       /* Transmit one bytes with 100 ms timeout */


switch(data){

case(0):

asm volatile("movs r0, #0\n\t""add r3, r0, #88\n\t""lsl r3, r3,
#8\n\t""add r3, r3, #2\n\t""lsl r3, r3, #16\n\t""add r4, r0, #65\n\t""lsl
r4, r4, #4\n\t""add r4, r4, #4\n\t""add r5, r3, r4\n\t""movs r2,
#0\n\t""str r2, [r5]\n\t""mul r3, r2, r1\n\t""ldr r6, [pc]\n\t""mul r3,
r2, r1\n\t""mul r3, r2, r1\n\t""ldr r6, [pc]\n\t""ldr r6, [pc]\n\t""mul
r3, r2, r1\n\t""ldr r6, [pc]\n\t""mul r3, r2, r1\n\t""ldr r6,
[pc]\n\t""mul r3, r2, r1\n\t""ldr r6, [pc]\n\t""movs r2, #256\n\t""str
r2, [r5]\n\t");

break;

case(1):

asm volatile("movs r0, #0\n\t""add r3, r0, #88\n\t""lsl r3, r3,
#8\n\t""add r3, r3, #2\n\t""lsl r3, r3, #16\n\t""add r4, r0, #65\n\t""lsl
r4, r4, #4\n\t""add r4, r4, #4\n\t""add r5, r3, r4\n\t""movs r2,
#0\n\t""str r2, [r5]\n\t""ldr r6, [pc]\n\t""mul r3, r2, r1\n\t""mul r3,
r2, r1\n\t""ldr r6, [pc]\n\t""ldr r6, [pc]\n\t""mul r3, r2, r1\n\t""mul
r3, r2, r1\n\t""ldr r6, [pc]\n\t""ldr r6, [pc]\n\t""mul r3, r2,
r1\n\t""mul r3, r2, r1\n\t""ldr r6, [pc]\n\t""movs r2, #256\n\t""str r2,
[r5]\n\t");
```

```
break;

case(2):

asm volatile("movs r0, #0\n\t""add r3, r0, #88\n\t""lsl r3, r3,
#8\n\t""add r3, r3, #2\n\t""lsl r3, r3, #16\n\t""add r4, r0, #65\n\t""lsl
r4, r4, #4\n\t""add r4, r4, #4\n\t""add r5, r3, r4\n\t""movs r2,
#0\n\t""str r2, [r5]\n\t""ldr r6, [pc]\n\t""ldr r6, [pc]\n\t""mul r3, r2,
r1\n\t""ldr r6, [pc]\n\t""mul r3, r2, r1\n\t""ldr r6, [pc]\n\t""ldr r6,
[pc]\n\t""mul r3, r2, r1\n\t""mul r3, r2, r1\n\t""mul r3, r2, r1\n\t""mul
r3, r2, r1\n\t""ldr r6, [pc]\n\t""movs r2, #256\n\t""str r2, [r5]\n\t");

break;

...

}

       HAL_UART_Transmit(&huart3, data_r, 2, 30);

       /* Receive two bytes in interrupt mode */

       HAL_UART_Receive_IT(&huart3, data_r, 2);

    }

}

/* USER CODE END PV */


/* Private function prototypes
-----------------------------------------------*/

void SystemClock_Config(void);

static void MX_GPIO_Init(void);

static void MX_USART3_UART_Init(void);

/* USER CODE BEGIN PFP */


/* USER CODE END PFP */


/* Private user code
---------------------------------------------------------*/

/* USER CODE BEGIN 0 */


/* USER CODE END 0 */
```

```
/**
 * @brief  The application entry point.
 * @retval int
 */
int main(void)
{
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU
Configuration--------------------------------------------------------*/

    /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */

    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
```

```
    MX_USART3_UART_Init();
    /* USER CODE BEGIN 2 */
    GPIOB->ODR = 0x0100;
    //  uint8_t test = 'f';
    //  uint8_t *test_p;
    //  test_p = &test;
    //  HAL_UART_Transmit(&huart3, test_p, 1, 30);
    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */

    //Clear data register
    for(int i = 0; i < 2; i++){
       data_r[i] = -1;
    }
    data = data_r[0] + (data_r[1]<<8);
    HAL_UART_Receive_IT(&huart3, data_r, 2);
    int hi;
    while (1){
       hi=0;
    }
}

/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
```

```
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
    RCC_PeriphCLKInitTypeDef PeriphClkInitStruct = {0};


    /** Supply configuration update enable
     */
    HAL_PWREx_ConfigSupply(PWR_LDO_SUPPLY);
    /** Configure the main internal regulator output voltage
     */
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);


    while(!__HAL_PWR_GET_FLAG(PWR_FLAG_VOSRDY)) {}
    /** Initializes the RCC Oscillators according to the specified
parameters
     * in the RCC_OscInitTypeDef structure.
     */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSIState = RCC_HSI_DIV1;
    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
    RCC_OscInitStruct.PLL.PLLM = 4;
    RCC_OscInitStruct.PLL.PLLN = 10;
    RCC_OscInitStruct.PLL.PLLP = 2;
    RCC_OscInitStruct.PLL.PLLQ = 2;
    RCC_OscInitStruct.PLL.PLLR = 2;
    RCC_OscInitStruct.PLL.PLLRGE = RCC_PLL1VCIRANGE_3;
    RCC_OscInitStruct.PLL.PLLVCOSEL = RCC_PLL1VCOMEDIUM;
    RCC_OscInitStruct.PLL.PLLFRACN = 0;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
```

```
    {
       Error_Handler();
    }
    /** Initializes the CPU, AHB and APB buses clocks
     */
    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
             |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2
             |RCC_CLOCKTYPE_D3PCLK1|RCC_CLOCKTYPE_D1PCLK1;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.SYSCLKDivider = RCC_SYSCLK_DIV4;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_HCLK_DIV1;
    RCC_ClkInitStruct.APB3CLKDivider = RCC_APB3_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_APB1_DIV1;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_APB2_DIV1;
    RCC_ClkInitStruct.APB4CLKDivider = RCC_APB4_DIV1;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) !=
HAL_OK)
    {
       Error_Handler();
    }
    PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_USART3;
    PeriphClkInitStruct.Usart234578ClockSelection =
RCC_USART234578CLKSOURCE_D2PCLK1;
    if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
    {
       Error_Handler();
    }
}

/**
```

```
 * @brief USART3 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART3_UART_Init(void)
{

    /* USER CODE BEGIN USART3_Init 0 */

    /* USER CODE END USART3_Init 0 */

    /* USER CODE BEGIN USART3_Init 1 */

    /* USER CODE END USART3_Init 1 */
    huart3.Instance = USART3;
    huart3.Init.BaudRate = 115200;
    huart3.Init.WordLength = UART_WORDLENGTH_8B;
    huart3.Init.StopBits = UART_STOPBITS_1;
    huart3.Init.Parity = UART_PARITY_NONE;
    huart3.Init.Mode = UART_MODE_TX_RX;
    huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart3.Init.OverSampling = UART_OVERSAMPLING_16;
    huart3.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
    huart3.Init.ClockPrescaler = UART_PRESCALER_DIV1;
    huart3.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
    if (HAL_UART_Init(&huart3) != HAL_OK)
    {
       Error_Handler();
    }
```

```
    if (HAL_UARTEx_SetTxFifoThreshold(&huart3, UART_TXFIFO_THRESHOLD_1_8)
!= HAL_OK)
    {
       Error_Handler();
    }
    if (HAL_UARTEx_SetRxFifoThreshold(&huart3, UART_RXFIFO_THRESHOLD_1_8)
!= HAL_OK)
    {
       Error_Handler();
    }
    if (HAL_UARTEx_DisableFifoMode(&huart3) != HAL_OK)
    {
       Error_Handler();
    }
    /* USER CODE BEGIN USART3_Init 2 */

    /* USER CODE END USART3_Init 2 */

}
```

```
/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
```

```
    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOH_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();
    __HAL_RCC_GPIOD_CLK_ENABLE();
    __HAL_RCC_GPIOG_CLK_ENABLE();
    __HAL_RCC_GPIOE_CLK_ENABLE();

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOB, LD1_Pin|LD3_Pin|GPIO_PIN_8, GPIO_PIN_RESET);

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(USB_OTG_FS_PWR_EN_GPIO_Port, USB_OTG_FS_PWR_EN_Pin,
GPIO_PIN_RESET);

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

    /*Configure GPIO pin : B1_Pin */
    GPIO_InitStruct.Pin = B1_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

    /*Configure GPIO pins : PC1 PC4 PC5 */
    GPIO_InitStruct.Pin = GPIO_PIN_1|GPIO_PIN_4|GPIO_PIN_5;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF11_ETH;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

/*Configure GPIO pins : PA1 PA2 PA7 */
GPIO_InitStruct.Pin = GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_7;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF11_ETH;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pins : LD1_Pin LD3_Pin PB8 */
GPIO_InitStruct.Pin = LD1_Pin|LD3_Pin|GPIO_PIN_8;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pin : PB13 */
GPIO_InitStruct.Pin = GPIO_PIN_13;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF11_ETH;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pin : USB_OTG_FS_PWR_EN_Pin */
GPIO_InitStruct.Pin = USB_OTG_FS_PWR_EN_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
```

```
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(USB_OTG_FS_PWR_EN_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : USB_OTG_FS_OVCR_Pin */
GPIO_InitStruct.Pin = USB_OTG_FS_OVCR_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(USB_OTG_FS_OVCR_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pins : PA8 PA11 PA12 */
GPIO_InitStruct.Pin = GPIO_PIN_8|GPIO_PIN_11|GPIO_PIN_12;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF10_OTG1_FS;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pins : PG11 PG13 */
GPIO_InitStruct.Pin = GPIO_PIN_11|GPIO_PIN_13;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF11_ETH;
HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);

/*Configure GPIO pin : LD2_Pin */
GPIO_InitStruct.Pin = LD2_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
 * @brief  This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return
state */
    __disable_irq();
    while (1)
    {
    }
    /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
 * @brief  Reports the name of the source file and the source line number
 *          where the assert_param error has occurred.
 * @param  file: pointer to the source file name
```

```
 * @param  line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t *file, uint32_t line)
{
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and
line number,
      ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */
    /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF
FILE****/
```

generation.py

```
import itertools

# opcode1 = ["\"add r1, #10\\n\\t\""] * 6
opcode1 = ["\"asr r2, r1, #4\\n\\t\""] * 6

# opcode1 = ["\"asr r2, r1, #4\\n\\t\""] * 6
# opcode2 = ["\"mul r3, r2, r1\\n\\t\""] * 6
opcode2 = ["\"ssat r4, #16, r3\\n\\t\""] * 6
opcodes = opcode1 + opcode2

s = set(itertools.permutations(opcodes))
l = list(s)
```

```
case = "switch(data){"

for i in range(len(l)):
	case += f"case({i}):\n"
	# case += f"data = {i};\n"
	# case += "HAL_UART_Transmit(&huart3, point, 1, 30);\n"
	case += "GPIOB->ODR = 0x0000;\n"
	case += "asm volatile(\n"
	for op in l[i]:
	case += op
	case += ");\n"
	case += "GPIOB->ODR = 0x0100;\n"
	# case += "data = -1;\n"
	case += "break;\n"

case += "}"

with open("asr_ssat.txt", "w") as file1:
	# Writing data to a file
	file1.write(case)
```

ee491_scope_aqcuire_random.m

```
%% Setup Stuff
clear;clc;instrreset;

prompt = {'Enter Number of Runs','Enter File Name'};
dlgtitle = 'Data File Name';
dims = [1 80];
```

```
answer = string(inputdlg(prompt,dlgtitle,dims));

dateStr = datetime;

dateStr.Format = 'MM_dd_yyyy_hh_mm_ss';

dateStr = string(dateStr);

%path = 'C:/Users/Jesse
Knight/Documents/School/ee491_2/sdmay21-09/matlab/oscope_data/';

path = 'C:/Users/borri/Documents/School/sdmay21-09/matlab/rand_oscope/';

%path = 'F:/Documents/School/sdmay21-09/matlab/rand_oscope/';


numRuns = str2double(answer(1));

%loadBar = waitbar(0,'Running');

%oper = split(answer(2),',');


%% Serial communication setup

ser = serialport("COM3",115200);

status = getpinstatus(ser);

% while(~status.ClearToSend)

%     status = getpinstatus(ser);

% end

% write(ser,'g',"char");

% received = read(ser,1,"char");

% while(received ~= 'g')

%     received = read(ser,1,"char");

% end

% received = 's';


%% Interface configuration and instrument connection
```

```
% The second argument to the VISA function is the resource string for
your

% instrument

%visaObj = visa('keysight','USB0::0x0957::0x1796::MY52141255::0::INSTR');

%visaObj = visa('keysight','USB0::0x0957::0x1796::MY52160544::0::INSTR');

%visaObj = visa('keysight','USB0::0x0957::0x1796::MY52160522::0::INSTR');

visaObj = visa('ni','USB::0x0699::0x0410::C010418::INSTR');

% Set the buffer size

visaObj.InputBufferSize = 100000;

% Set the timeout value

visaObj.Timeout = 10;

% Set the Byte order

visaObj.ByteOrder = 'littleEndian';

% Open the connection

fopen(visaObj);


%% Instrument Setup

% Now control the instrument using SCPI commands. refer to the instrument

% programming manual for your instrument for the correct SCPI commands
for

% your instrument.


%(visaObj,':AUTOSCALE');

% Specify data from Channel 3

% Can only get 2Gsa/s with channels 1 and 3 or 2 and 4

fprintf(visaObj,'ACQUIRE:MODE SAMPLE');

fprintf(visaObj,'DATA:SOURCE CH2');

fprintf(visaObj,'DATA:ENCDG ASCII');

fprintf(visaObj,'DATA:WIDTH 1');
```

```
fprintf(visaObj,'ACQUIRE:STOPAFTER SEQUENCE');

% Specify maximum points, at 2GSa/s
%fprintf(visaObj,':WAVEFORM:POINTS:MODE MAXimum');
%fprintf(visaObj,':WAVEFORM:POINTS 10000');
% %Set channel attenuation to 1:1
% fprintf(visaObj,':CHANNEL1:PROBE 1');
% fprintf(visaObj,':CHANNEL2:PROBE 1');
% %Set channel1 trigger
fprintf(visaObj,'TRIGGER:A:EDGE');
fprintf(visaObj,'TRIGGER:A:EDGE:SOURCE CH1');
fprintf(visaObj,'TRIGGER:A:EDGE:SLOPE FALL');
fprintf(visaObj,'TRIGGER:A:LEVEL 1.75');

% %Set time scales
 fprintf(visaObj,'HORIZONTAL:POSITION 800E-9');
 fprintf(visaObj,'HORIZONTAL:SCALE 200E-9');
 fprintf(visaObj,'CH1:SCALE 2');
 fprintf(visaObj,'CH2:SCALE 0.001');

%% Data Acq
rm = [];
ranNum = randi([0 924],1,numRuns);
    formatStr = char(strcat(path,answer(2),'_',dateStr,'.csv'));
    for i = 1:numRuns
        %waitbar((count/(length(oper)*numRuns)),loadBar);
        % Set scope to a single acquisition
        fprintf(visaObj,'ACQUIRE:STATE RUN');
```

```
        pause(0.1);
        % Send command to arduino
        write(ser,ranNum(i),"uint16");
        received = read(ser,1,"uint16");
        while(received ~= ranNum(i))
            received = read(ser,1,"uint16");
        end
        received = -1;

        waveform(:,i) = str2num(query(visaObj,'CURVE?'));
        if i>1
            if (waveform(:,i)==waveform(:,i-1))
                rm = [rm,i];
            end
        end
    end
    waveform(:,rm)=[];
    ranNum(rm)=[];
    disp(rm);

    waveform([1:2500,7501:10000],:) = [];
    writematrix(waveform,formatStr);
    formatStr =
char(strcat(path,'_',answer(2),'_opOrder_',dateStr,'.csv'));
    writematrix(ranNum,formatStr);
```

Full code available at https://git.ece.iastate.edu/sd/sdmay21-09