EE / CprE / SE 491 – sdmay21-09

Instruction Level Reverse Engineering through EM Side Channel

Week 2 Report

2/08/2021 – 2/22/2021
Client: Akhilesh Tyagi
Faculty Advisor: Akhilesh Tyagi

Team Members:

Noah Berthusen — *Data Analysis Engineer*

Matthew Campbell — *Test Engineer*

Cristian George — *Meeting Scribe*

Jesse Knight — *Signals Processing Engineer*

Evan McKinney — *Integration Engineer*

Jacob Vaughn — *Report Manager*

Weekly Summary

This week we started using the data we collected to feed our machine learning algorithms. We are now able to run binary classification with 100% accuracy to classify nops and addi instructions. During this week we also found some problems with our execution times that we were not expecting. We plan to work out a way to use a debugger in the coming week to explore this problem further. The software team created environments in Frankenstein and Didymium-2 clusters for scalable machine learning. Included initiating an Anaconda environment that contains dependencies for ML packages. This will help us moving forward.

Past Week Accomplishments

- New data format
    - Determined that current assembly structure probably was not making ML any easier (one operation surrounded by nops is very similar to only nops)
    - Instead changed to repeating an operation a number of times not surrounded by nops. This suggestion was also given by Dr. Daniels in the PIRM meeting.
    - Changed the position of the antenna when collecting data. The previous position was potentially too close to the clock and was causing the data collected to be too noisy for ML.
- Machine-Learning
    - Were able to run binary classification with 100% accuracy
        - Able to tell difference between nop and addi
        - Using time series random forest classification, splits intervals of the time

series and gathers mean and standard deviation to create a new feature set for ML. Also tried working with KNN and dynamic time warping but this does not give as good results, or takes too long to train.

    - Created environments in Frankenstein and Didymium-2 clusters for scalable machine learning. Included initiating an Anaconda environment that contains dependencies for ML packages
- Python
    - Overlay time series graphs, Fourier transformations and CWT of python arrays (for visuals)

Pending Issues

- Execution time in between runs is inconsistent.
    - We have disabled compiler optimizations and are trying to generate an assembly file to observe how the compiler is interpreting our code.
    - After meeting with our advisor, we must also begin debugging using a separate software suite.

Individual Contributions

Team Member	Contribution	Weekly Hours	Total Hours
Noah Berthusen	ML training and environment setup	7	10
Matthew Campbell	Assembly debugging	3	7
Cristian George	Assembly debugging (1) + binary classifier experimentation	5	8
Evan McKinney	Debugging Assembly Timing + Time Series ML	10	13
Jacob Vaughn	Assembly debugging & Catching up on ML code	3	6
Jesse Knight	Data collection for binary classification Ordered new antenna parts	6	10

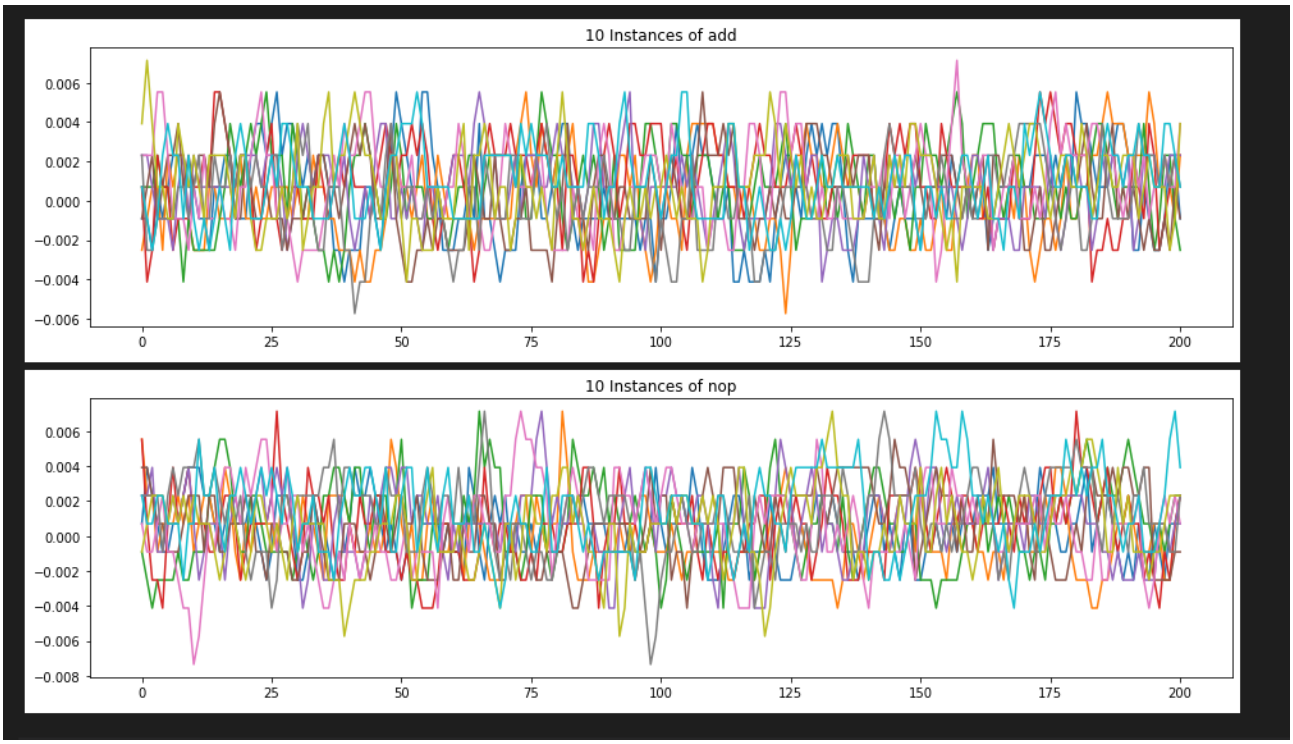
Plans for Coming Week

- Cristian:
    - Use VSCode Arduino extension to debug/step-through code on the board.

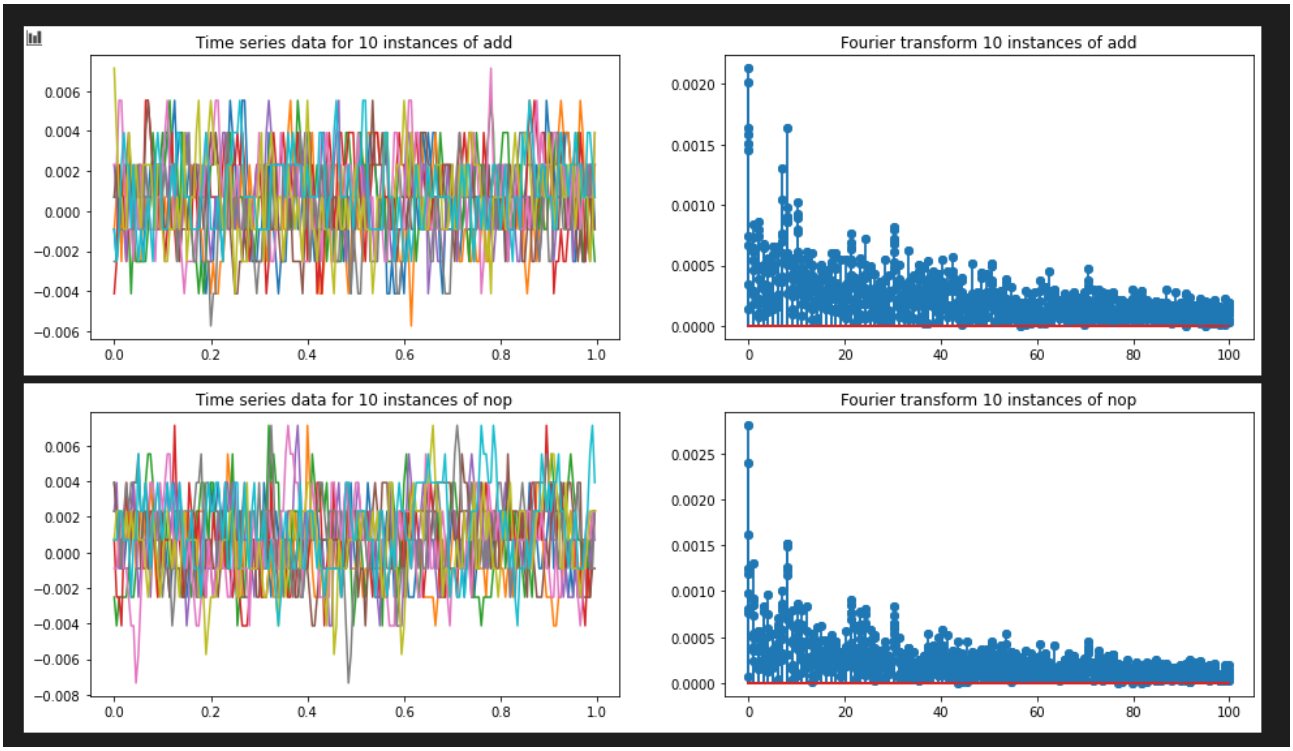
    - Install appropriate third--party packages to support our board with Arduino.
- Jesse, Matt:
    - Convert current Arduino implementation into a STM32CubeIDE implementation.
    - Update serial/uart calls to the appropriate HAL calls for our board.
- ML Team (Evan, Noah, Jacob):
    - Begin classification at a larger scale
    - Instead of binary classification move on to a larger class set of around four classes.
    - Create an environment on the HPC cluster to take advantage of GPUs when training on large amounts of data.
    - Look into matched filter classification

Summary of weekly advisor meeting (If applicable/optional)

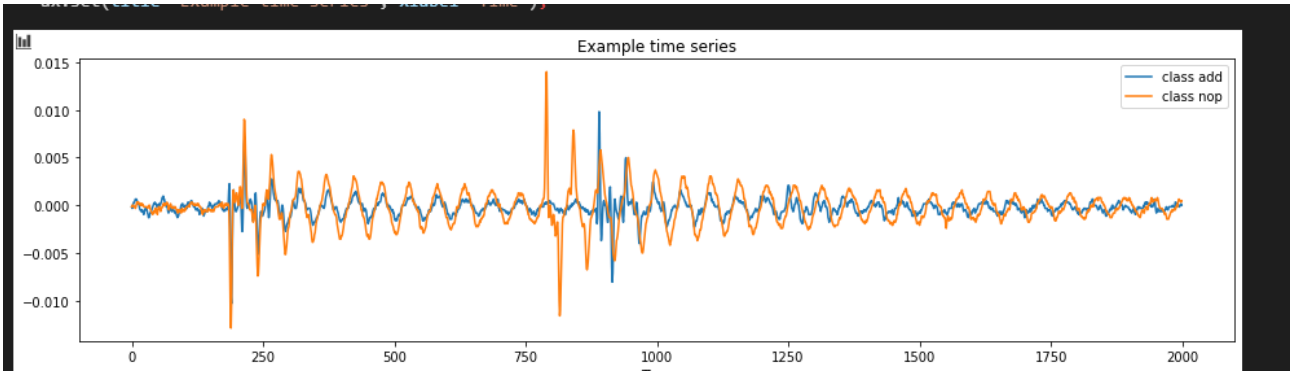
Meeting with Varghese at 12 pm on Monday the 22nd. The topic of the meeting was attempting to determine why the code was still appearing optimized even though the compiler optimization flag was set to off. Varghese proposed running the program in debug mode to make sure each of the operations was happening. I.e. stepping through the code should result in 12 consecutive ADD instructions instead of 1 ADD instruction and 11 noOps (since an optimizer will condense the ADD instructions into one).



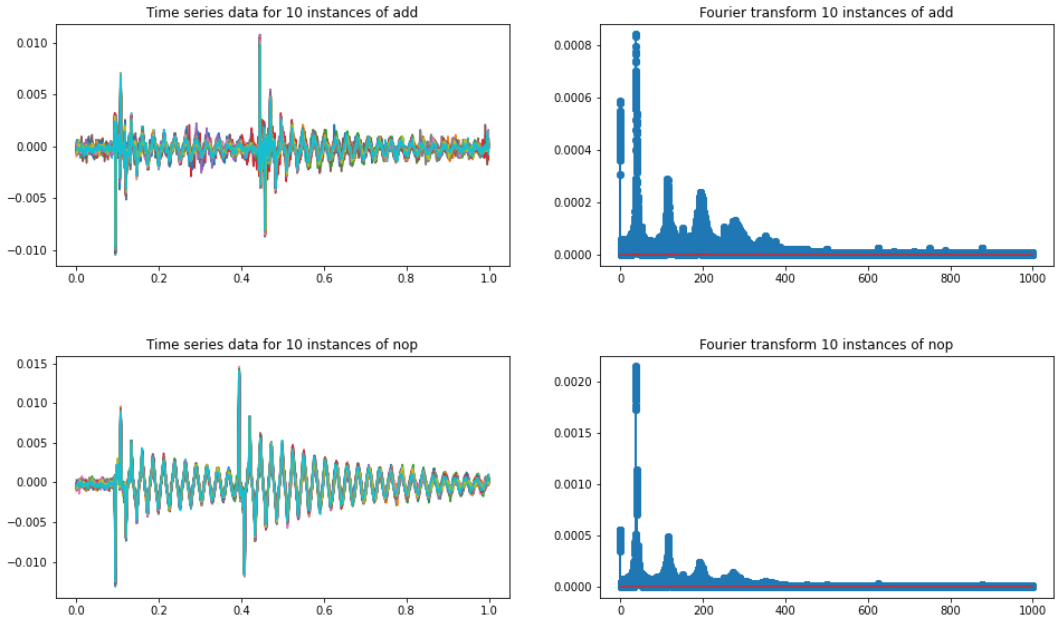
This was the original set of data, we could not get machine learning to work over 66% accuracy, which was the same as random guessing because our dataset was ⅔ add instructions and ⅓ nop instructions.



Clearly this dataset is mostly noise.



This is for a new dataset. Before we were doing an instruction once and surrounded it by nops in order to isolate it. Since we were having troubles with the timing, we found it was mostly noise because we were not capturing the signal at the appropriate time. For this dataset we filled the entire execution span with the same instruction on repeat, guaranteeing that our capture window would see the instruction happening. This means our dataset more accurately reflects the instruction but is much less precise because it is not feasible to repeat an instruction many times in a row for classification during a normal program's execution.



This image shows clearly a difference between the addis and the nops and that means our ML can perform classification easily, reaching 100% accuracy on an 80/20 train-test dataset split.

```
[33]
#let's get a baseline for comparison
from sklearn.dummy import DummyClassifier

classifier = DummyClassifier(strategy="prior")
classifier.fit(X_train_tab, y_train)
classifier.score(X_test_tab, y_test)

0.494
```

```
[35]
# now we can apply any scikit-learn classifier
classifier = RandomForestClassifier(n_estimators=100)
classifier.fit(X_train_tab, y_train)
y_pred = classifier.predict(X_test_tab)
accuracy_score(y_test, y_pred)

1.0
```