Iowa State University

CPR E 491

Instruction Level Reverse Engineering (Disassembly) through EM Side Channel

Sdmay21-09:

Noah Berthusen (Data Analysis Engr.)

Matthew Campbell (Test Engr.)

Cristian George (Meeting Scribe)

Jesse Knight (Signals Processing Engr.)

Evan McKinney (Integration Engr.)

Jacob Vaughn (Report Manager)

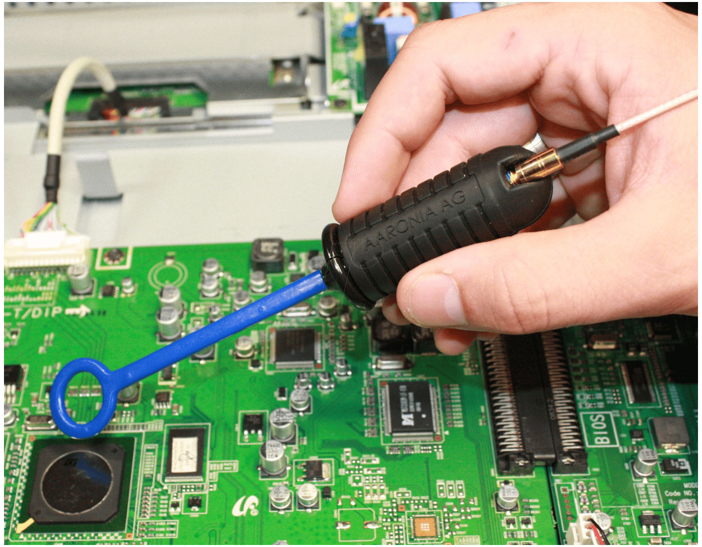
Advisors:

Dr. Akhilesh Tyagi

Varghese Vaidyan

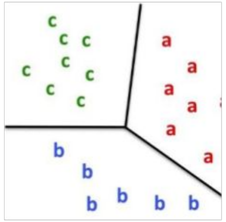
Project Vision

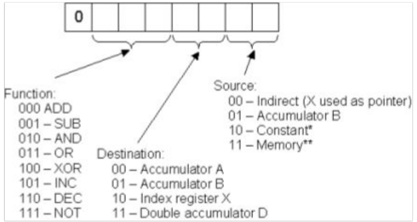
- Reverse engineer the executing program of a processor through measurements of the electromagnetic radiation that it emits.
- Code/data authentication
- Security Implications



```
0,0.0003216080384
1e-09,0.0006432160768
2e-09,0.0004824120576
3e-09,0.0001608040192
4e-09,0
5e-09,-0.000804020096
6e-09,-0.0011256281344
7e-09,-0.0003216080384
8e-09,-0.0006432160768
9e-09,-0.0006432160768
1e-08,-0.0003216080384
1.1e-08,-0.000804020096
1.2e-08,-0.0006432160768
1.3e-08,-0.0004824120576
1.4e-08,-0.0004824120576
1.5e-08,0.0003216080384
1.6e-08,0.0004824120576
1.7e-08,0
1.8e-08,0.0001608040192
1.9e-08,0.0003216080384
2e-08,0.0001608040192
2.1e-08,0.0003216080384
2.2e-08,0.0003216080384
2.3e-08,0.0004824120576
2.4e-08,0.0004824120576
2.5e-08,-0.0001608040192
2.6e-08,-0.0003216080384
2.7e-08,0.0001608040192
2.8e-08,0
```

TensorFlow





Requirements

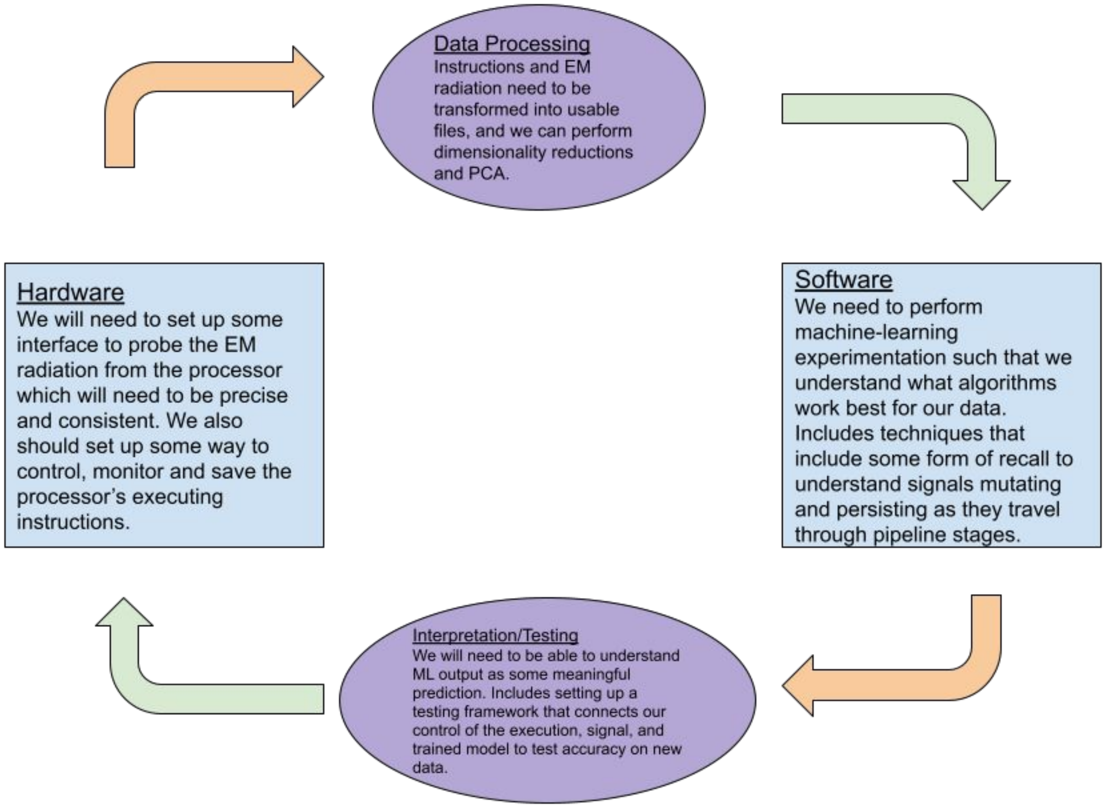
- Collecting electromagnetic radiation data from a ARM-M7 processor with a 20+ Mhz frequency and 6-stage pipeline.
- Building an interface between our EM antenna and code to organize and filter relevant data.
- Written in Python
- Large amount of data used to train the model

Requirements cont.

- Opcode detection with 90%+ accuracy
- Well-documented code
- Predictions formatted to be user-friendly

Constraints

- Budget: $100
- Oscilloscope availability
- Oscilloscope bandwidth
- Minimum pipeline size
- Computing resources for training
- Covid-19 Pandemic

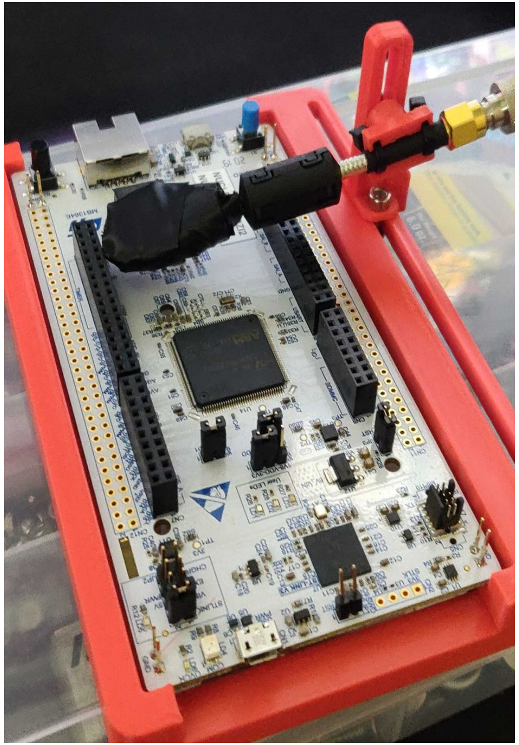

Data Processing
Instructions and EM radiation need to be transformed into usable files, and we can perform dimensionality reductions and PCA.
Software
We need to perform machine-learning experimentation such that we understand what algorithms work best for our data. Includes techniques that include some form of recall to understand signals mutating and persisting as they travel through pipeline stages.
Interpretation/Testing
We will need to be able to understand ML output as some meaningful prediction. Includes setting up a testing framework that connects our control of the execution, signal, and trained model to test accuracy on new data.
Hardware
We will need to set up some interface to probe the EM radiation from the processor which will need to be precise and consistent. We also should set up some way to control, monitor and save the processor's executing instructions.

System Design

- Development board connected to DSO
- EM Probe mounted on dev board using custom-built mount
- Dev boards triggers code execution using serial
- MatLab handles data capture from DSO
- Data is transformed and sent to TensorFlow for ML

Project Hardware

- Data Collection Interface
  - Nucleo-144
  - Tektronix DPO3012 DSO
  - EM Probe
  - DSO Probe for GPIO

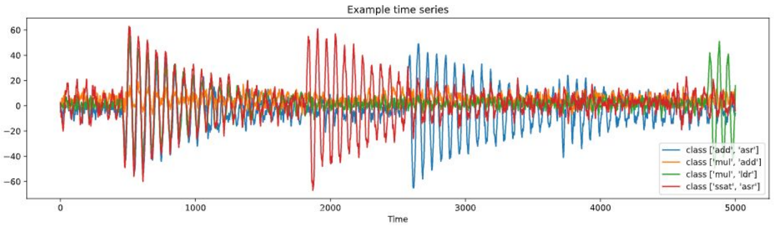


Project Software

- Embedded SW:
  - Nucleo-144 code
  - Data generation code
  - Matlab data capture script
- Machine learning experimentation:
  - Classification model for Opcodes
  - Various machine learning techniques tested

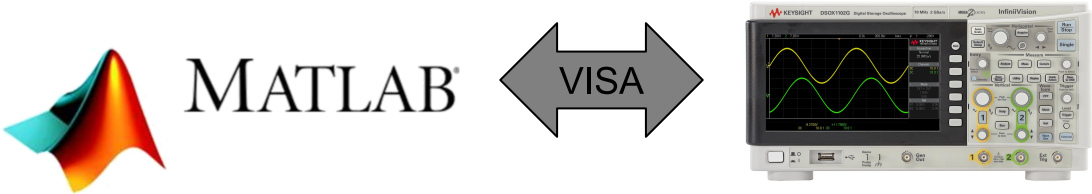
SW Platforms

- Python code will adhere to the PEP 8 standard
- NumPy and Pandas for data structures
- Scikit-Learn (sktime) used for machine learning
- Jupyter Notebooks



HW Platforms

- Visa communication between Matlab and Oscilloscope
- Serial communication between Matlab and Nucleo board



Automation

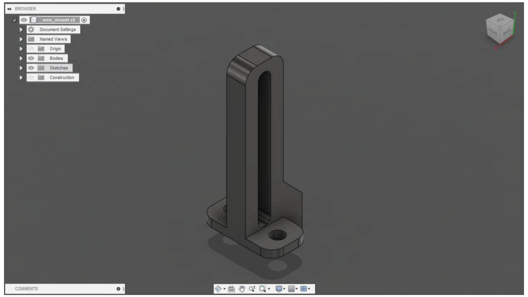
```
fprintf(visaObj,'HORIZONTAL:POSITION 800E-9');
fprintf(visaObj,'HORIZONTAL:SCALE 200E-9');
fprintf(visaObj,'CH1:SCALE 2');
fprintf(visaObj,'CH2:SCALE 0.001');
```

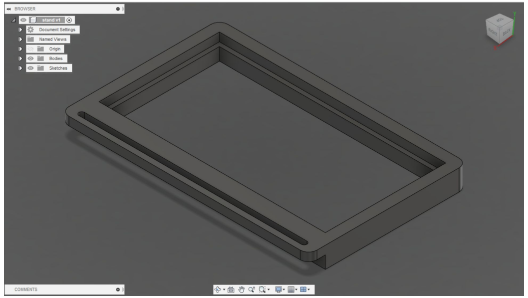
```
switch(data){
case(0):
asm volatile("movs r0, #0\n\t""add r3, r0, #88\n\t""lsl r3, r3, #8\n\t""add r3, r3, #2\n\t""lsl r3, r3, #16\n\t""add r4, r0, #65\n\t""lsl r4, r4, #4\n\t""add r4, r4, #4\
break;
case(1):
asm volatile("movs r0, #0\n\t""add r3, r0, #88\n\t""lsl r3, r3, #8\n\t""add r3, r3, #2\n\t""lsl r3, r3, #16\n\t""add r4, r0, #65\n\t""lsl r4, r4, #4\n\t""add r4, r4, #4\
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\
break;
case(3):
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\
break;
```

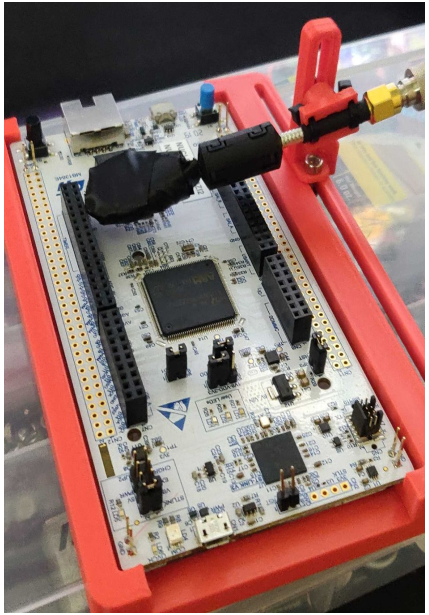
Homemade EM probe

https://www.eevblog.com/forum/blog/eevblog-1178-build-a-$10-diy-emc-probe/

EM probe mount



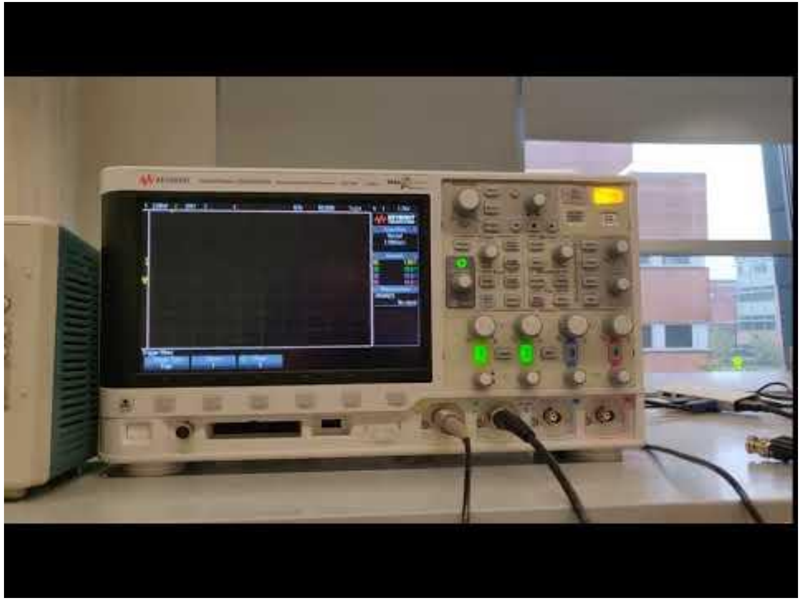


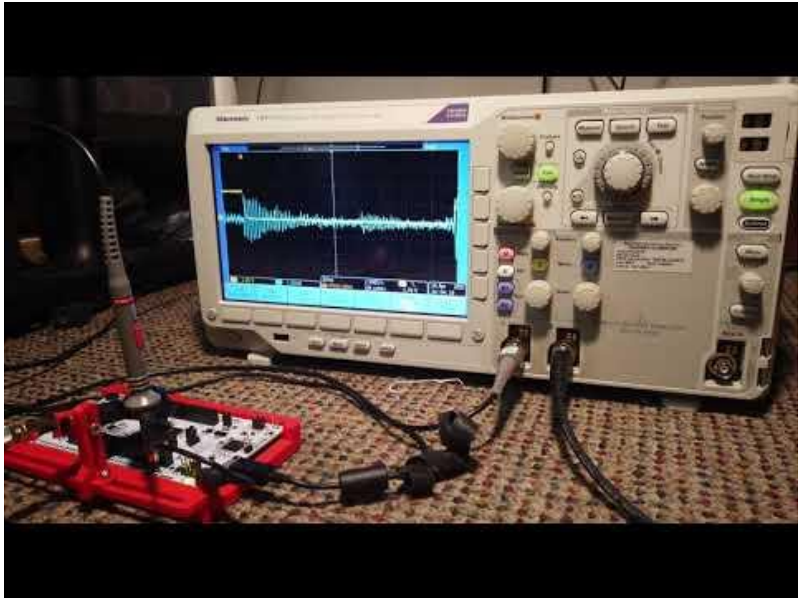


New Oscilloscope

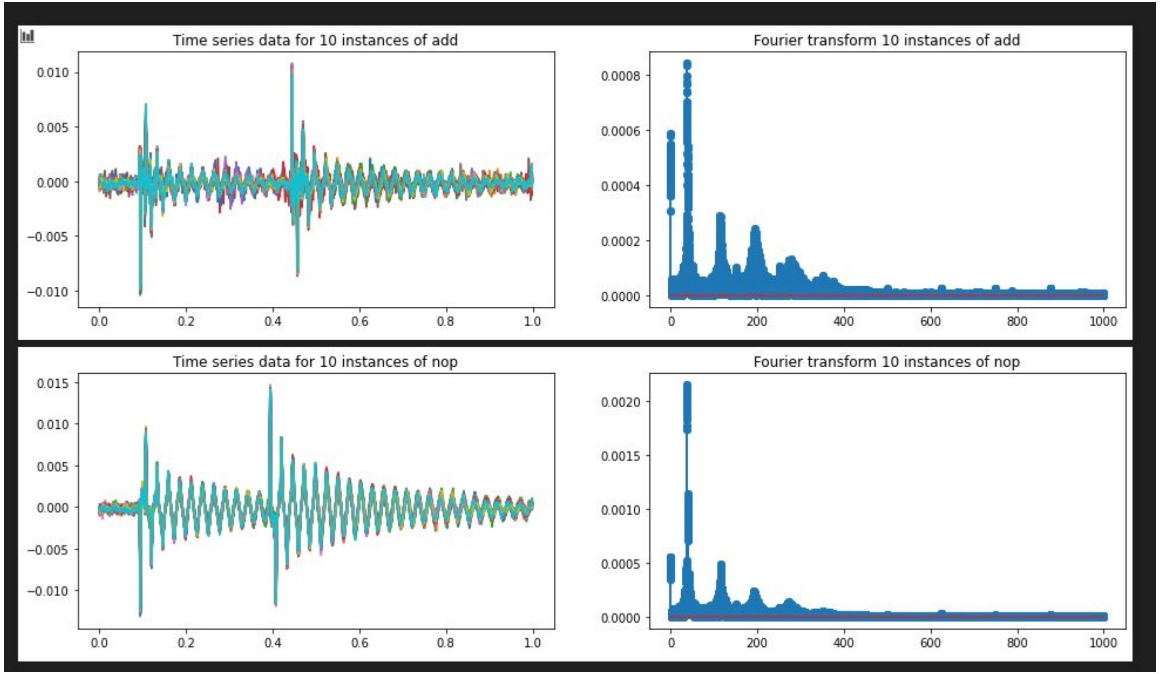
- DPO3012 vs DSOX2024A
- New VISA Commands
- New data format
- New bandwidth and sampling rate
- Significantly reduced capture rate

Hardware Implementations



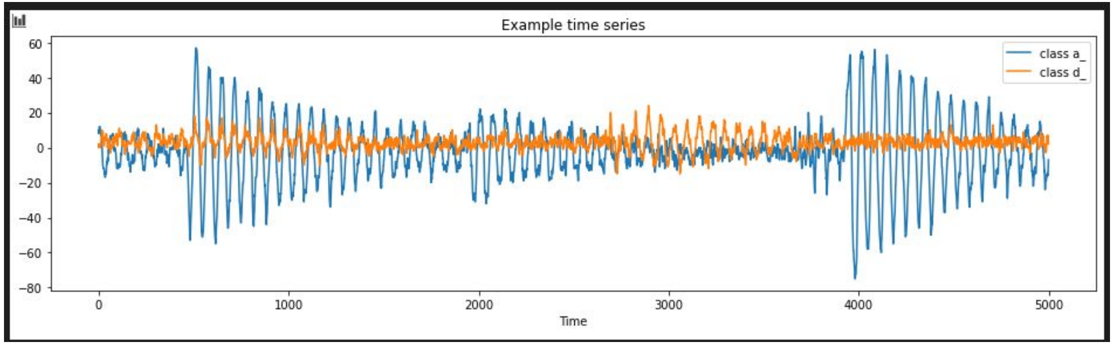


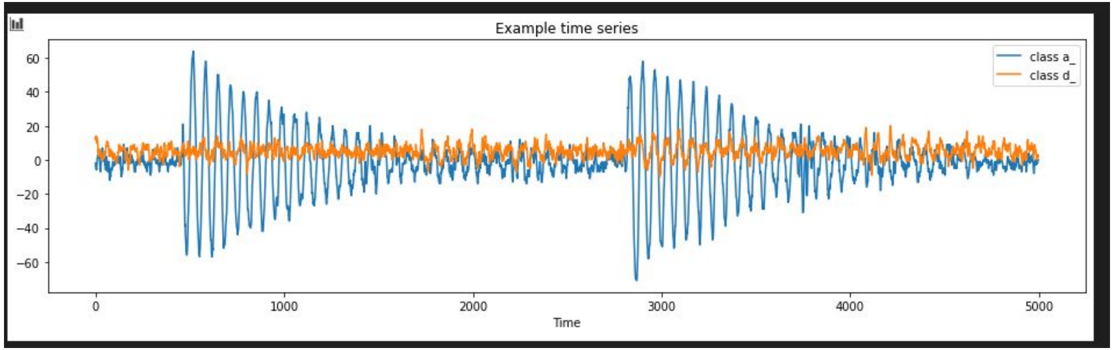
Machine Learning process



- Single instructions padded with nops
- Coherence among samples is an easier ML task
- Binary classification reaches 98+% accuracy

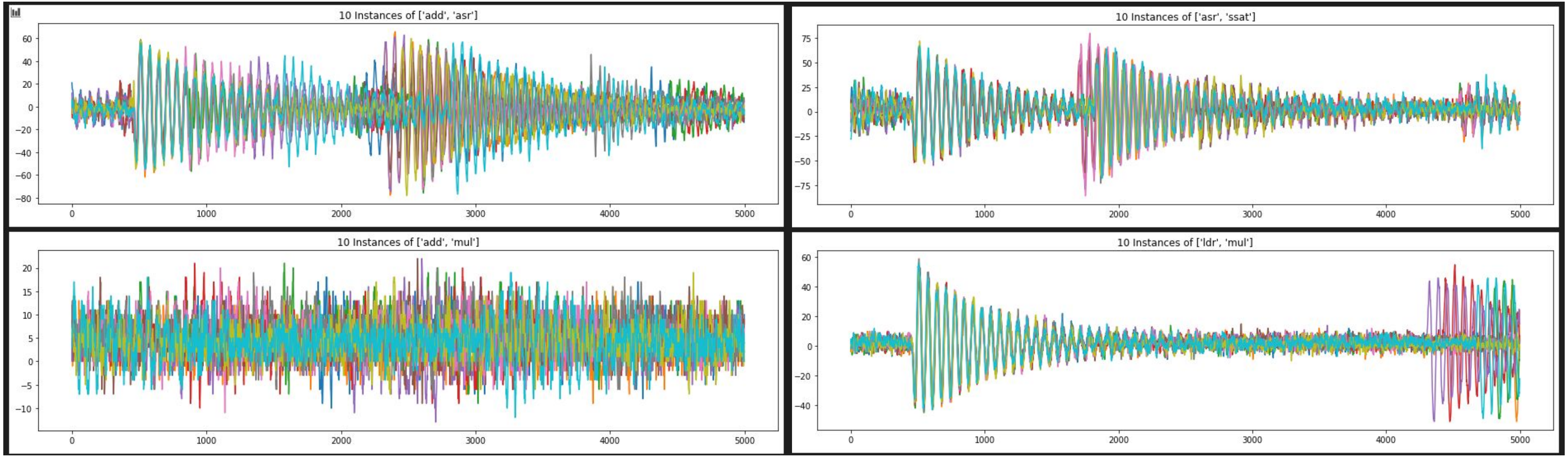
Machine Learning process cont.





- Multiple instructions in pipeline - our approach is to train on random permutations of order instructions
- Pros: Irrespective of precise timing to break up time series into clock cycles
- Cons: Doesn't scale well since requires too much training data
- Need to use multilabel ML

Machine Learning process cont.



Multilabel ML Attempts

	add	asr	mul	ssat	ldr
Random	.500	.499	.500	.250	.249
SVC	.501	.494	.626	.494	.629
Random Forest [n=100]	.502	.494	.626	.494	.630
Time Series Random Interval Tree	.502				
Random Interval Spectral Forest	.502				

```
[7]
y0_train = np.array([y_train_multilabel[i][0] for i in range(len(y_train_multilabel))])
y0_test = np.array([y_train_multilabel[i][0] for i in range(len(y_test_multilabel))])
start = time.time()
# # now we can apply any scikit-learn classifier
# classifier = RandomForestClassifier(n_estimators=100)
classifier = svm.SVC()
# multi_target_forest = MultiOutputClassifier(classifier, n_jobs=-1)
classifier.fit(X_train_tab, y0_train)
print(time.time() - start)
y_pred = classifier.predict(X_test_tab)
accuracy_score(y0_test, y_pred)

64.5960762500763

0.5013763763763763
```

Conclusions: Done

- Hardware acquired and set up
- Processor, oscilloscope, Matlab communication established
- Preliminary machine learning model prototyping

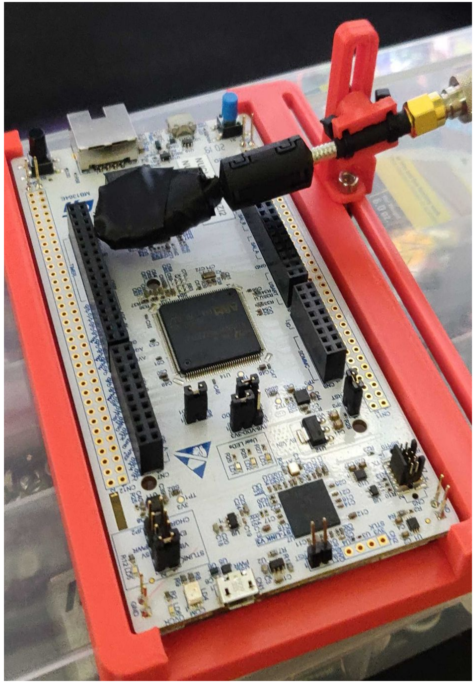
Lessons learned

- Design process
- Tool automation
- Compiler optimizations
- Time series machine learning

Future Work

- Collection of more data
- Train new machine learning model
- Use to detect operands in running code
- Expand model to support even more instructions

Questions?



sdmay21-09.sd.ece.iastate.edu