Tamper Analysis via Transient Electromagnetic Responses (TATER)
Final Report

Authors: Hannah Pearson, Matthew Covalt, Matthew Waltz, Roy Cochran, Lydia Engerbretson
Sponsor: Idaho Scientific
Date: 4-30-2018

Contact Information
Idaho Scientific: info@idahoscientific.com
Hannah Pearson: pear5988@vandals.uidaho.edu
Matthew Covalt: cova2624@vandals.uidaho.edu
Matthew Waltz: walt2909@vandals.uidaho.edu
Roy Cochran: coch1253@vandals.uidaho.edu
Lydia Engerbretson: ryan9529@vandals.uidaho.edu

TATER Final Report
4-30-2018

Abstract

In the following report it was found that there is a direct relationship between a machine level instruction and the electromagnetic emissions generated during the execution of a microprocessor. As instructions are executed, processor current draw fluctuations relative to the instruction being executed produce a unique profile in the electromagnetic fields surrounding the device under test. A capture setup and analysis algorithm were devised that could correlate between these profiles to determine if differences exist between the code which produced these electromagnetic signatures. This ability to detect differences between code sequences allowed for a proof of concept security system to be created which could detect if any modifications had been made to the baseline code that would normally be executing on a processor.

Contents

1 Introduction

1.1 Problem Statement

To design a system that can characterize and monitor the electromagnetic emissions of a microprocessor during its boot sequence to determine if modifications have been made to its boot code.

1.2 Summary

The first task of this project was to research a method by which we could profile the execution of a microprocessor based on electromagnetic emissions. This testing concluded that a H-Field loop antenna would be best to characterize the current variations produced from different instructions executing on the microprocessor.

We then had to decide on the specifics of the antenna that we wanted to use for our design and if we would create or purchase that instrument. Due to convenience we decided to purchase a H-Field Loop Antenna. The next task was to implement this antenna in a capture setup that allowed us to analyze an entire boot sequence in a consistent/reliable way. We were able to manage a system by which the device under test and the antenna were in a constant position. However, do to our setups susceptibility to noise and its capture size restrictions we determined that this was a major area of improvement for our project and ways to do so are later described.

The final step was to create an algorithm that would correlate the test capture against a set of baseline captures, ultimately with the goal of determining if any modifications had been made to the boot sequence that was gathered during that test capture. Completing this algorithm required a costume made system for alignment and correlation between capture sequences. A method of windowing was used that would divide each capture between a set amount of windows of configurable size and step through the capture correlating the signal segments within each window looking for any mismatches between the baseline and the test capture. We succeeded in the implementation of such an algorithm however, there are still some desired features. On balance we succeeded in our objective and have produced a system, further described in this document, that can detect modifications in machine level instruction based on electromagnetic emissions.

2 Antenna

2.1 H Field vs. E Field Probe Testing

Our initial testing began with the Beehive Electronics 100D EMC Electric Field (E-Field) Probe. Later we moved to the TekBox EMC probe set which contained 1 E-Field probe known as the E5, and 3 H-Field probes of different sizes known as H5, H10, H20 listed with increasing diameter of the loop. We ultimately purchased a RF Explorer H-Field Antenna, after seeing how perfectly it integrated into our hardware setup.

Figure 1: Beehive Electric E-Field Probe

Figure 2: TekBox EMC Probe Set - E5, H5, H10, H20

Figure 3: RF Explorer H-Field Probe

- E-Field (Electric Field) Probe: Responds to electric fields, electric fields created by voltage changes, stub antenna, not sensitive to direction
- H-Field (Magnetic Field) Probe: Responds to magnetic fields, magnetic fields created by current changes, loop antenna, sensitive to direction
- For clarity: Probe = Antenna

From our initial testing with the E-Field probes we were not visually able to notice the signal changing while code was executing/not-executing on the processor. However, once we added output to port instructions we could see large spikes which we determined to be related to these instructions. We noticed an alternation of the spikes, where the spike would primarily be positive on the first output to port and the spike primarily negative on the 2nd, and so on. We determined this to be the toggling ON (0 to 1) action as the positive spike and the toggling OFF (1 to 0) action as the negative spike. Again, we tried changing the resolution at which we were observing the signal and tried various types of instructions (load,store,add,...) and doing so we could not differentiate those instructions from the noise in between the output to port instructions.

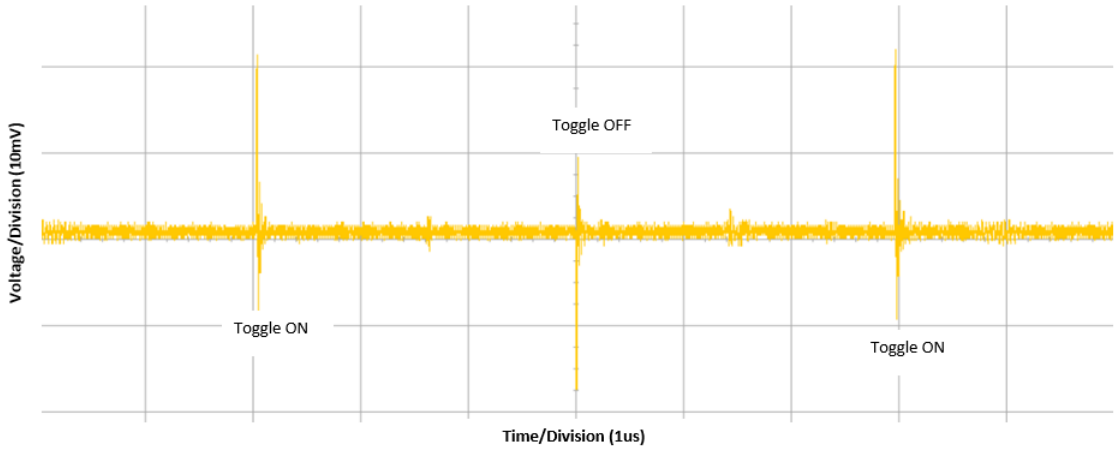

Figure 4: E-Field Probe Testing

However, once we received a TekBox H-Field probe we noticed a much more detailed profile. Not only could we see the large spikes that alternated for every output to port instruction, but we could also see a spike at every clock period. Similarly, we also noticed a smaller spike every half-clock period. In the current image, the output to port has closely similar positive and negative spikes for the toggle ON, then a smaller spike for the Toggle OFF. We noticed that if we changed the orientation of the H-field probe it changed the signal received, but only by amplitude of the signals, and sometimes if the spikes observed were more positively or negatively oriented. We determined through testing that we preferred having the probe lay parallel to the processor which gave a high magnitude signal. In later testing the H-Field probe we noticed if we changed the instruction (nop to lds/nop to add) we would see the spike at that clock period (instruction) increase in amplitude and change in the overall shape of the waveform that created that spike.

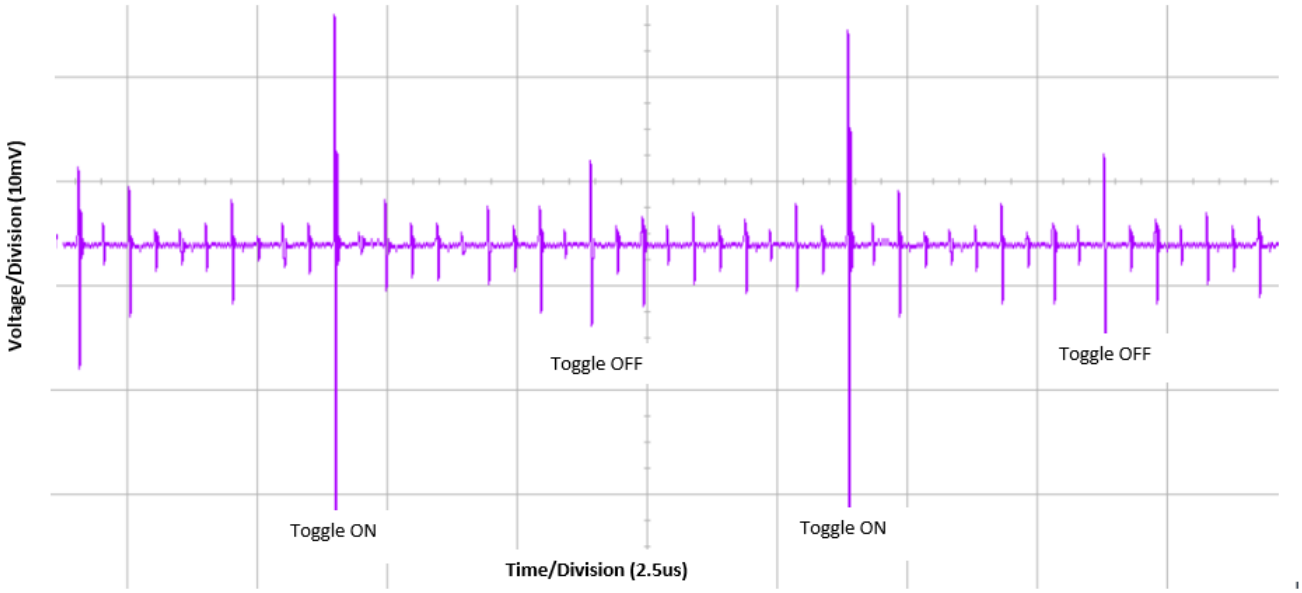

Figure 5: H-Field Probe Testing

The sequence of code that was executing on the processor for the previously described E-Field/H-Field Probe testing was a loop which inverted the 8-bits of a register and output that value to the pins of PORTB on our development board.

```
5  loop:
6          com     r16
7          out     PORTB,r16
8          rjmp    loop
```

Figure 6: Testing Code

As it was proving difficult to re-create test setups with the hand held TekBox EMC probe set we purchased the RF Explorer H-Field Probe which had similar characteristics to the H20 H-Field probet. The RF Explorer antenna, collected using an amplifier paired with an ADC acquisition platform, and processed collected data using a custom algorithm to create an EM profile of a processor's boot. The RF Explorer Probe helped alleviate this setup issue as it was shorter than the other probe and easier to mount onto a platform.

The difference that we noticed between the three TekBox EMC H-Field probes was that as we decreased the size of the H-Field Loop and used the H5 or H10 probes, the signal to noise ratio improved but the gain (amplitude) associated with that signal decreased. In further work on this project, using a smaller diameter H-field probe may be more useful if trying to improve the signal to noise ration and further amplification/filtering of the signal could be done to improve results and alleviate any sampling issues that resulted from changing the probe size.

The probes that we used were all Near Field Probes/Antennas, meaning that they were sensitive to either voltage variations (if E-Field) or current variations (if H-Field); in both cases, it only worked if they were within a few cm from the device under test. In the near field the wave impedance depends on whether there is primarily voltage or current variations, and is constant in the far field. If our device has two logic chips that are swinging 5V between each other with little current change, we could best detect this with an E-Field probe. If however we had a power supply trace that supplied a number of circuits where the voltage changes would be relatively small but, depending on the load, the current variations would be significant, we could best measure this with a H-Field probe. As we observed a more detailed representation of the processor execution using the H-field probe, we determined then that what we were measuring on our device under test was the current variations relative to the overall load (power consumption) of the various circuits on the board.

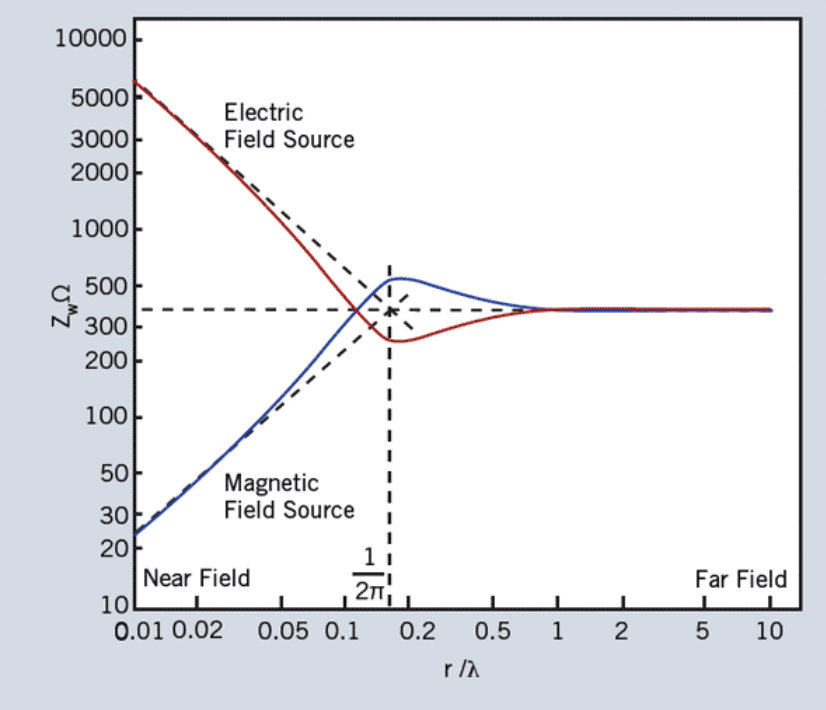

Figure 7: Near Field Probes

The above figure illustrates that in the near field the wave impedance which directly effects the signal observed is specific to whether the device is producing an electric or magnetic field.

2.2 What is Our Signal?

Using the TekBox H20 H-Field probe we determined that the frequencies that comprise our signal were between the range of 1MHz and 250 MHz. This was tested by keeping the probe constant and varying whether or not the processor was running. The spikes shown in the figure which lay outside of 250MHz were persistent between running and not running the processor and were assumed to be noise. Only the in area within the green box was there observed change.

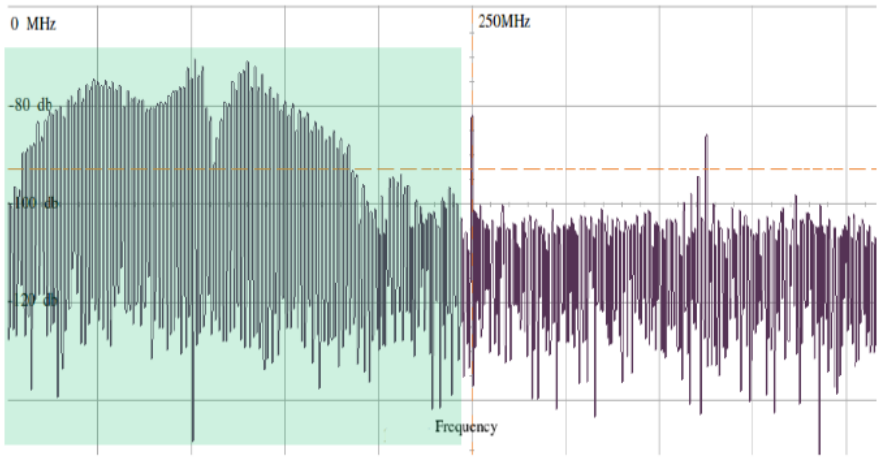

Figure 8: Frequencies of Interest

2.3 Antenna Design Considerations

Initially, much enthusiasm was shown in attempts to design an antenna, rather than simply purchasing one. Many designs were tried, most not even worthy of mention, as their consideration was minimal at best. The largest setback when attempting to design an antenna using the industry standard FEKO antenna design software, was that the student edition only allowed for a frequency sweep within one order of ten. This made frequency sweeps on our 1-250MHz range impossible, and discouraged a custom design.

One of the first antenna designs to be considered was a **Ferrite Rod antenna**. It was considered because it would have boasted a targeted radiated resistance, meaning there would have been no mismatch between the antenna and its attached coaxial cable, which would have in turn led to increased efficiency.

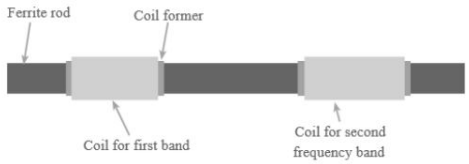

Figure 9: Example of a Ferrite Rod Antenna

The downside to this antenna was its immense bulk and its inability to cover the 1-250 MHz frequency range we needed to see, as it was able to cover the KHz range. Because of these reasons, as project learning continued throughout the first semester, this antenna design was scrapped in favor of different ones.

Another design that was considered was the **Pulse Larsen Outdoor Multi Band Antenna**. This antenna was considered as it had a bandwidth comparable to the H20 EMC Probe, and had a superior gain.

Figure 10: Pulse Larsen Outdoor Multi Band Antenna

This antenna would have been difficult to implement into our design, as it had a volume much larger than could be easily integrated into the hardware setup. As such, this antenna was considered too large in spite of its gain and bandwidth.

The final design that was ultimately implemented was the **RF Explorer H-Loop Near Field Antenna (RFEAN2)**. This antenna boasted a fantastic 1MHz-7GHz bandwidth, and while this was much more than was required, it was a similar range to the probe set. Alongside its bandwidth, this antenna also had a roughly 35dB gain, which when added to the amplifier, allowed a comparable output to the probe set.

This antenna was successfully able to compete with the probe set and was much cheaper. In terms of gain and bandwidth, it operated slightly worse than the probes, but the antenna was able to be purchased for considerably cheaper, while at a very marginal loss in performance. In addition, the RF Explorer had a similar design to our probes, meaning that we would not have to change our hardware setup. All of these considerations meant that when we found this antenna, it was immediately purchased and subsequently implemented.

3 Hardware Testing Setup

3.1 Equipment for Initial Setup

Our initial test setup with the hand held TekBox EMC probe set did not vary much from later setups. We used a Pocket AVR programmer to program the ATMega328p which was on the Barebones AVR development board. We then used some H-Field probe/antenna that would be amplified by our 12V HiLetgo RF Wide Band 30dB Gain Amplifier. We did change the Barebones AVR development board from having a 1MHz Oscillator to a 16MHz oscillator. Our reasoning behind this was that it cut down on the overall length of the capture window that we needed to be gathered. Prior to this change, if we wanted to capture 8000 instructions at 1MHz (1us per instruction) that would require a capture window of at least 8ms. If we instead used a 16MHz oscillator (63ns per instruction), capturing 8000 instructions would take 500us. Initially, we had concerns that this would cause the waveforms of interest to bleed into one another as they may be too close to distinguish, but that was not observed with the 16MHz testing.

3.2 What is being Measured

As stated in the probe testing section we are using H-Field probes to measure current variations in the overall power consumption of the Barebones AVR development board. This means that as the various circuits on the board (primarily the ATMega328p Circuits) are loads to the overall power required by the board, and as the current has quick variations these loads consume power. What H-Field probes then produce as an output is a reading of the voltage running along the wire of the probe which is induced from the magnetic field passing through the loop of the probe. The area of the board that we observed to produced the highest magnitude signal was near the 0.1uF capacitor external to the board. However, the signal to noise ratio was undesirable at this position so we decided to place the probe roughly over the center of the ATMega328p processor.

3.3 Instruction Modifications Relative to NOP

The first figure shows the difference between varying the number of bits toggled in an output to port instruction. On the left 0 bits are being toggle and this increases until on the right of the figure it shows the effect of 8 bits being toggled. We noticed a distinct amplitude difference between the toggle ON (1 to 0) transition and the toggle OFF (0 to 1) transition. The main take away from this figure is that an output to port can be 1000 percent larger than the average amplitude observed from a NOP, making it easily detectable.

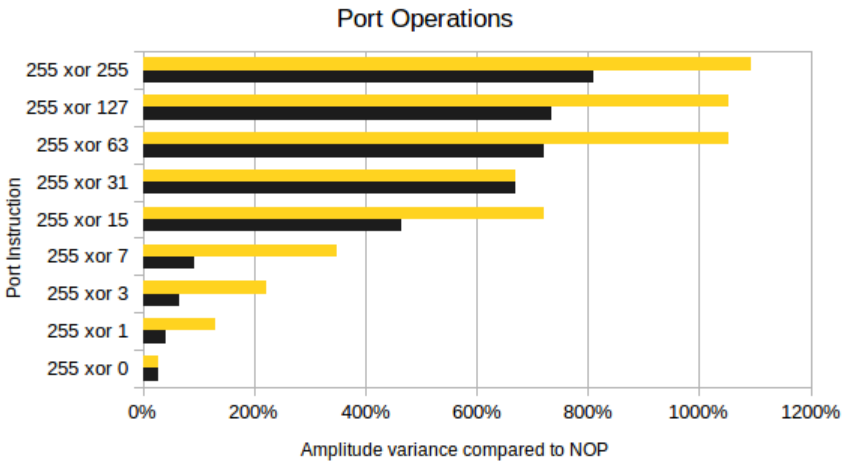

Figure 11: Output to Port Observations

This second figure shows the difference of a set of instructions based on the amplitude change it produces compared to the noise level. We observe a spike at every clock period, however the average amplitude of that spike varies depending on the instruction and that is what we are recording in this graph. Most instructions produce a similar average amplitude increase, however a few outliers were store and load instructions which produced on average a higher average amplitude increase. Most instructions lay within the 10 to 30 percent range which compared to the 1000 percent increase of an output to port requires a much higher resolution to analyze.7

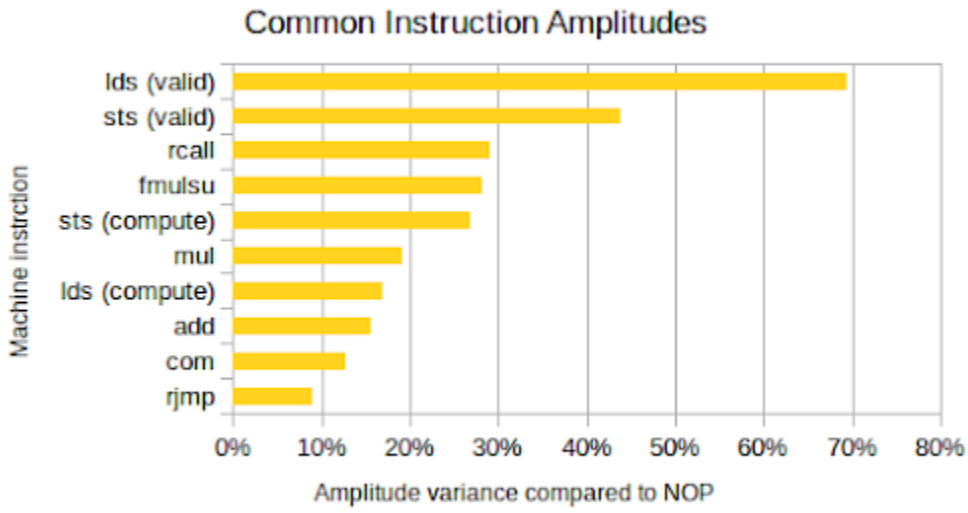

Figure 12: Common instruction Observations

3.4 Setup

The Barebones AVR development board was screwed into a sheet of acrylic. This sheet of acrylic was then glued to the bottom section of an aluminum case. This aluminum case was then screwed into an acrylic platform which formed an L shape. The RF Explorer H-Field probe was then permanently mounted in the vertical facing section and the aluminum case/development board was screwed to the bottom horizontal section to be easily unscrewed and slid out from underneath the probe, ideally aligned in the same position every time. The section covered by the black electrical tape is attached using epoxy; however it will break apart if handled roughly.

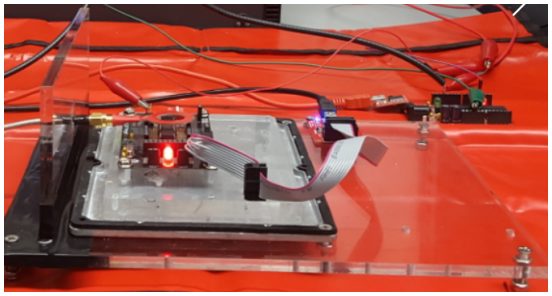

Figure 13: Test Platform Side View

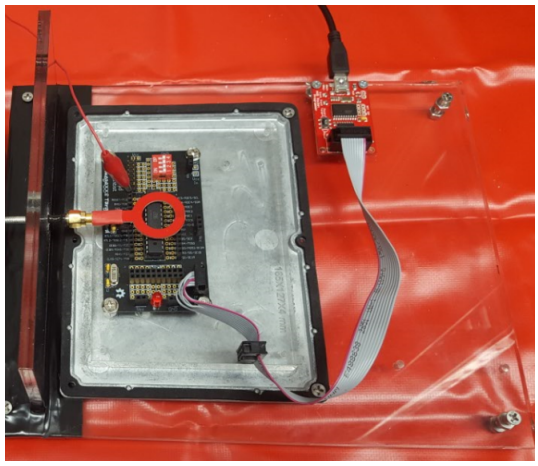

Figure 14: Test Platform Top View

The array of switches which we attached to the board is then soldered/wired up underneath the board, also under the board is a 220 Ohm resistor that enables usage of the board's red LED. We decided to move the switches and LED from a breadboard to the board itself as a connected breadboard produced an undesirable level of noise.

The entire platform is set on a red shielding pad. The acrylic platform is also shown. For this setup we used the 30dB HiLetgo RF Amplifier. This required an external power supply of 12V. We chose to use this amplifier instead of the 20dB TekBox amplifier because it required its own external power supply, as compared to the 20dB amplifier to which power was supplied via a USB. The Arduino used for capture reset is triggered via the UART connection, and once triggered sends a reset signal to the Barebones AVR development board and also drives a pin high which triggers the oscilloscope to begin a capture after a specified delay time.

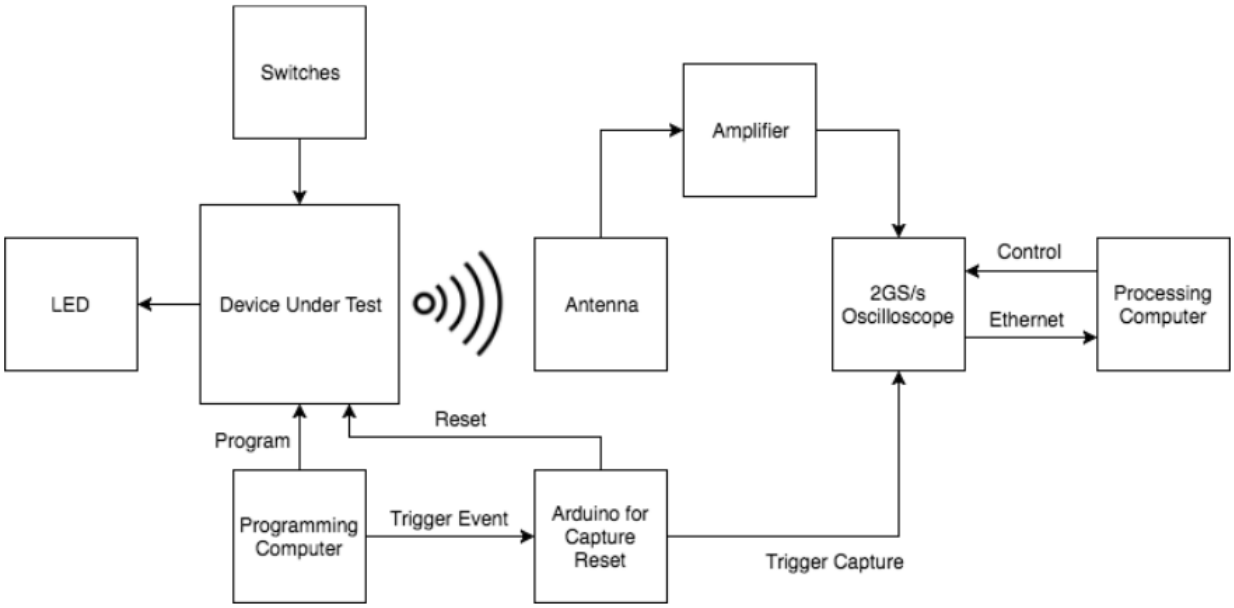

Figure 15: Main Setup Diagram

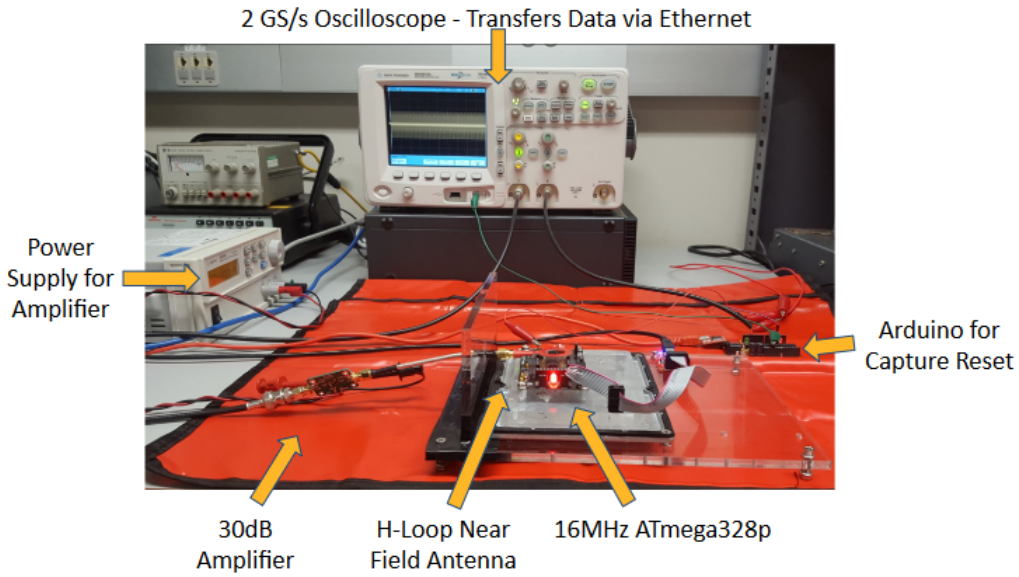

Figure 16: Main Setup

The oscilloscope that we used was an Agilent MSO6012A 2GS/s Oscilloscope. It is programmed and initialized via its Ethernet connection. This connection is also used to transfer data in a .csv file format for processing in the algorithm. With our current setup this captures 40000 data samples at a 12 bit resolution 2GS/s rate. This file is then roughly 5MBytes in size and takes 90-100sec for a complete transfer. The connection between the probe and the amplifier is made at analog input 1 and the trigger for capture reset is at analog input 2. A grounding connection is also made with the Arduino for capture reset. One of the primary settings required by this setup is that the input to analog input 1 be delayed by roughly 127ms, as that was the time found between the trigger being received by the oscilloscope and code actually beginning to execute on the Barebones development board after it received a reset.

In the planned setup that would use the KC705 and FMCADC4 Daughter Board, the system would work in a similar way with noted differences. To start, the FPGA-ADC system would replace the oscilloscope. This is a desirable change because the maximum capture window available at a 2GS/s (or even a 1GS/s) rate on the oscilloscope was relatively small and required us to make a custom bootloader that was short enough and would run quickly enough to perform tests on. With the FPGA-ADC setup, we planned to instead stream data at roughly 1GS/s which would allow us to execute longer sequences of code with the trade off of decreasing our sampling rate, which could have led to issues when analyzing the data, as that sampling rate was only 4X the Nyquist rate required by a max 250MHz signal. In order to determine whether or not this alternate setup would cause these issues, it would be necessary to set up a system as described and perform more testing.

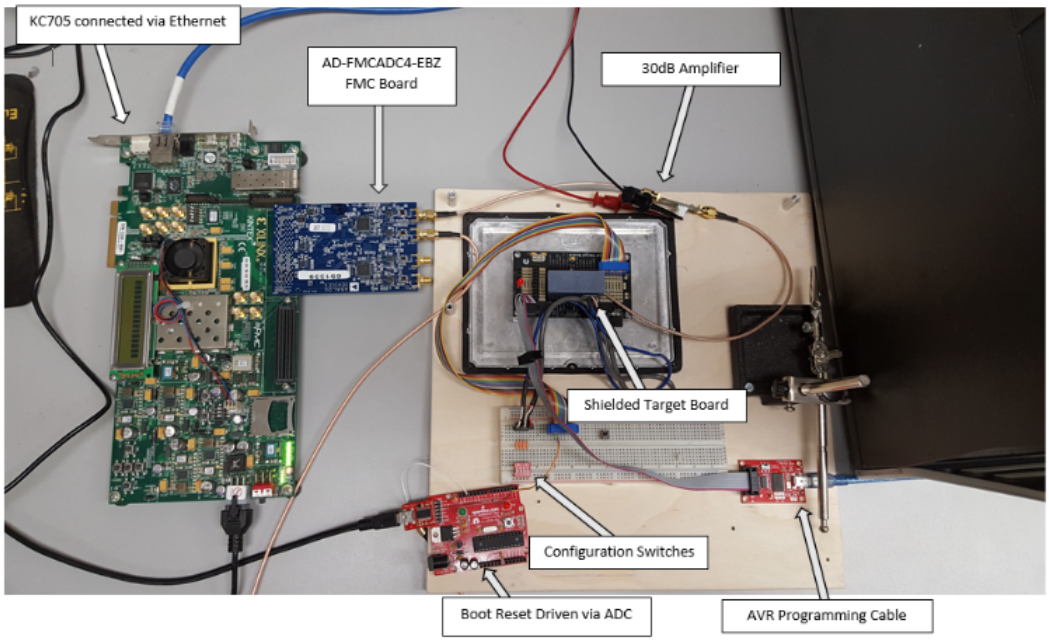

Figure 17: Alternative Setup

3.5 FPGA Design and Issues

The main goal of the FPGA subsystem is to achieve a higher data capture rate in addition to better accuracy with each collected sample. This would also allow for a more refined setup and triggering interface directly with the FPGA.

We began by downloading the Analog Devices HDL reference designs which include an implementation of a ZC706 FGPA interfacing with the FMCADC4, linked below:

https://github.com/analogdevicesinc/hdl/releases/tag/2017_r1

In order to adapt the provided hardware reference design to suit the KC705, we:

- Replaced the Zynq processing system with a MicroBlaze and adapted connections to other peripherals.
- Replaced the Zynq SPI interface with a Xilinx Quad SPI driver IP.
- Removed the DMA controller from the hardware design, and replaced it with a singular GPIO sample request line.
- Updated system constraints and ported to the KC705.
- Replaced the Zynq memory interface with a standard MIG controller.

The DMA core was replaced with a simple GPIO request line for samples in order meet both the timing requirements of the design and to account for the single memory interface on the KC705 as opposed to the dual memory interface on the ZC706.

The next step was to implement the software driver capable of properly configuring the ADC daughter board and collecting samples into memory for streaming over the Ethernet interface. We cloned the no-OS software repository from Analog Devices which contained the base code for configuring the ADC available here:

https://github.com/analogdevicesinc/no-OS

Adding the Ethernet interface was accomplished by adding a LwIP server in order to drive TCP packets through the interface. We began by testing the throughput of the connection; using an EthernetLite core limited the theoretical bandwidth to 100Mbits/sec. Qualitative testing and fine-tuning of the LwIP core resulted in approximately 18.5 seconds in streaming time to transfer 100MB (50 million samples @ 14 bits). Fine tuning changes included increasing packet pool size and pending packet allocation size.

We then integrated the FMCADC4 code for gathering samples, and adapted it to send a GPIO rather than a DMA request in order to begin sample streaming. Using a UART interface for output, we configure and query the status of the ADC core in order to ensure the clocks are properly locked and items such as the sampling rate and JESD204B receiving link are properly synchronized with the daughter board. The following image displays the result of this test, indicating a successful configuration:

```
waltz@Waltz ~/Documents/git/tater $ ./script.sh
ad9528 clock configured correctly!
ad9680 configured via spi correctly!
QPLL ENABLE
Rx link is enabled
Measured Link Clock: 250MHz
Link status: DATA
SYSREF captured: Yes
adc_setup adc core initialized (1000 MHz)
ad9680 - PN9 sequence mismatch
ad9680 - PN23A sequence mismatch
memory transfer request started.
transfering 100MB via ethernet...
complete.
```

Figure 18: Configuration and Ethernet Transfer Success

3.6 FPGA Issues

We then encountered the issue of samples not being correct when read from the ADC. Using an integrated ILA core we were able to determine that the samples, while they are being written to memory and are readable via the MicroBlaze debug module and can be streamed out via Ethernet, appear to be random values. Using a function generator to force a known signal into the FMCADC4 also demonstrated this to be the case.

This may stem directly from the reported self-test failure from the ADC itself, which has an on-board testing system in order to test both the sample collection and interaction with the JESD204B

link layer. From the previous image testing of both the Polynomial Sequence (PN) 9 and 23A, which are available ADC tests used to ensure correct ADC function, were not successful in passing. These tests generate a sequence of values and transmit them over the JESD204B link and then are read back to ensure that the values correctly match.

3.7 Building the FPGA Design

Currently Vivado 2017.1 was used for building each section. Code for both the hardware reference design and the software driver code has been packaged into this project's (TATER) GitLab repository. The hardware design can be inserted into the `projects` directory from the first download, and the software portion can be placed into the `projects` directory of the no-OS download. Both portions can then be built by running the 'make' command in the respective project directory.

3.8 Testing the Bootcode

Both figures show a run of our bootcode where the voltage level observed is relatively low until a sudden point at which the bootcode has started executing which is represented by a sharp voltage increase. It then indicates that it is finished with a loop of output to port instructions which produce a voltage signal that goes outside of the scale of the graph.

The difference between the two figures is that an output to port instruction has been inserted in the middle of the right figure. The main take away from this is how you can easily visualize some changes made within the bootcode.

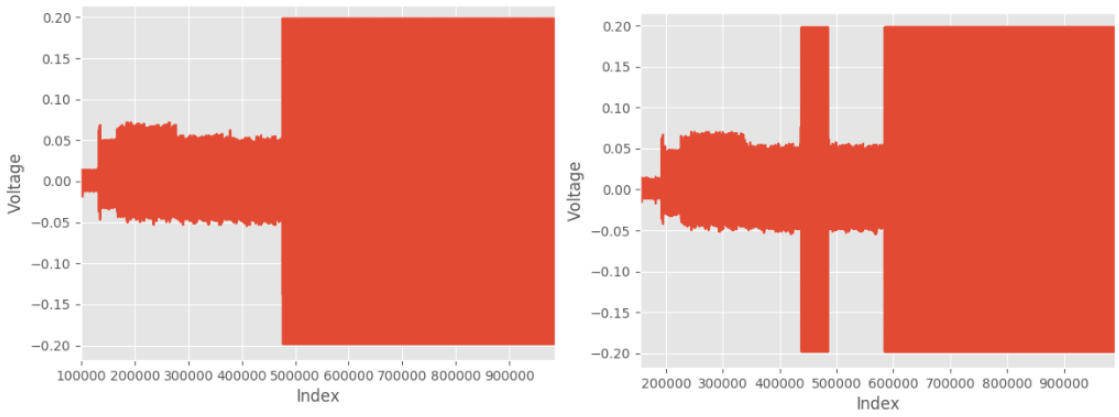

Figure 19: Bootcode Comparison

3.9 Testing the Bootcode - A Closer Look

The following figure represents a zoomed in version of different runs of an output to port instruction. Visually there are some differences but overall the shape of the waveform can be observed as relatively consistent between runs.

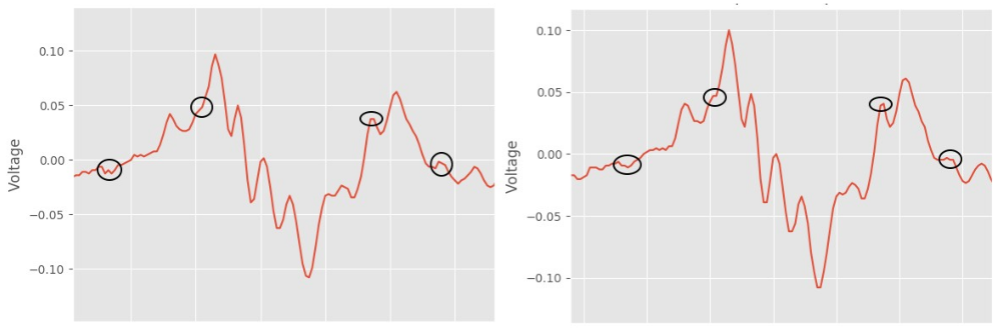

Figure 20: Different run same instruction

The following figure represents a slightly zoomed out version of a single run of the previous figure. Which not only shows the signal characteristics of an output to port instruction but also of an ones compliment and relative jump instruction.

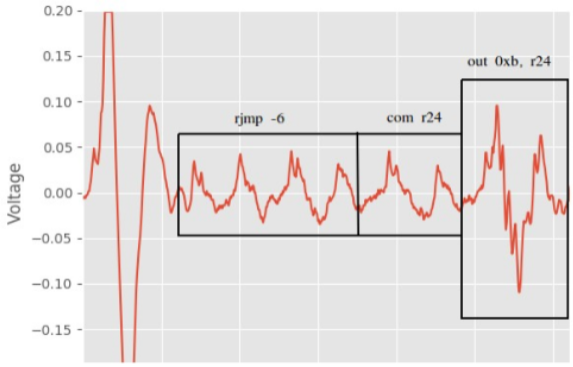

Figure 21: Visualizing whats running

3.10 What Changes the Current System

There were many external things that would change our capture results, specifically the level of noise that we observed. If we plugged anything else into the board it would increase the overall noise level. If we had the breadboard connected via two cables that greatly increased our noise level. Even having the Arduino used for capture reset slightly increased our noise level, but as it was necessary we did not remove it.

If there were any overhead florescent lights that would increase our noise level. If any computers were running nearby or any devices plugged into the same wall outlet that would increase the noise level. Changing between amplifiers also changed the level of noise. The USB connected 20dB amplifier had a visibly higher noise level compared to the externally supplied 30dB amplifier. However, even with the 30dB amplifier changing the external power supply changed the level of noise, after tying a couple of different power supplies the Agilent U8002A supply was found to show the least amount of noise.

We also noticed that touching and holding any of the wires would drastically change our signal,

and we were cautious as to repositioning any of the wires, but did not not see much change if the wires were arranged differently.

So the ideal setup that we used for most of our testing was in a dark room, with no devices plugged in/running nearby. Whith only a single laptop that was connected via USB to the barebones target device, and the Arduino for capture reset. This setup produced always above a 95 correlation percentage between different captures of the same code sequence (with our algorithm).

During our expo demonstration we faced two main issues. The first being that there were overhead lights, and the second being that everyone in the room was using a shared power supply. This level of noise changed our algorithms correlation percentage of the same code (different runs) to 70 percent. We were able to alleviate this by taking multiple (5) baseline captures instead of 1 and averaging them together into a single model which new captures would be ran against. This also caused the delay time required for the oscilloscope to capture our boot sequence to continuously vary roughly 1ms from the set 127ms in a way that we did not previously notice in lab. We hypothesized this to be due to the shared power supply and people adding/removing devices.

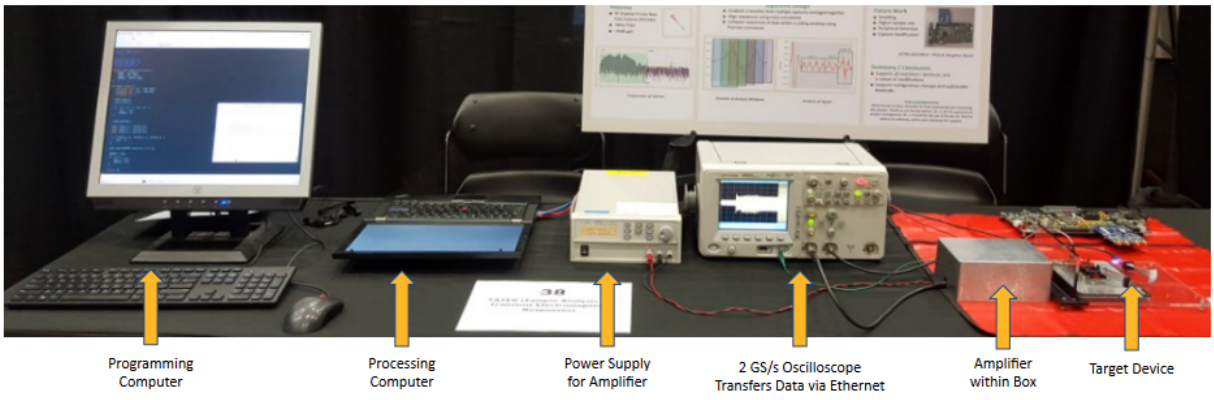

Figure 22: Expo Setup

4 Software Algorithm

4.1 Summary

The software algorithm relies of the creation of a baseline in order to have a directly comparable model used for ensuring the validity of an arbitrary capture.

In signal processing, cross-correlation is a measure of similarity of two series as a function of the displacement of one relative to the other. We use this displacement similarity in order to align the start of each capture with the exact boot code startup. This is performed in steady increments over two different signals in order to determine the boot code starting location of each. If a capture is not able to be properly aligned, then it may be a strong indicator of a modified capture.

We then also use the Pearson correlation coefficient, which is a measure of the linear similarity between data points. This is used to compare the baseline model to a testable capture with potential foreign modifications.

4.2 Baseline Creation

The baseline works by averaging multiple hardware captures together to produce a single output file that is consistent with repeated captures. This accounts for and minimizes minor fluctuations that may occur in the signals due to incidental noise and other anomalies. Here is an outline of the baseline function:

- Align the baseline capture(s) sequences using cross-correlation to the start of the boot code, removing excess front and back parts from the signal.
- Create an array of aggregated data points from the baseline captures, averaged for each different configuration change.
- This array of samples holds the base case that will be used to differentiate against the modified capture(s).
- Write array of samples to a model file.

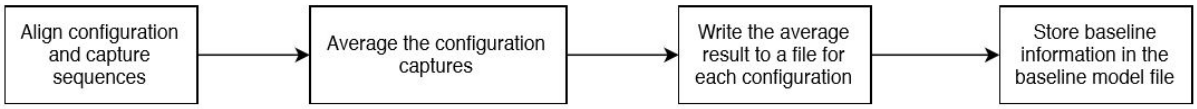

Figure 23: This diagram is the basic outline of baseline creation

The model file can then be used for testing of actual modified boot code.

4.3 Using the Baseline for Analysis

The core of the algorithm relies on the Pearson correlation coefficient in order to determine the similarity between the two signals: The baseline and a capture. Below is the algorithm with the correlation value in `r`, and the sum is computed over the available window (described below). `x` and `y` are the single iterable samples in the window, while `xÌĎ` and `ÈŞ` represent the mean across the full window.

$$r = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sqrt{\sum (x - \bar{x})^2 \sum (y - \bar{y})^2}}$$

Figure 24: Equation used for obtaining the Pearson correlation coefficient

The following lists the steps taken by the algorithm in order to determine a passing or failing output:

- Use the length of the capture to determine which potential configuration this may be, and use the appropriate data from the model file.
- Align both sequences (baseline and capture) to the start of the boot code. This is based on the displacement found using cross correlation. This allows us to compare how similar the sequences are in regards to location of peaks and time in between. Initially aligning the sequences ensures that correlation values will be maximized.
- Use sliding windows and apply Pearson correlation to the windowed sections of the capture data, advancing by a set fraction of a window. These windows are compared against the baseline data to determine where the modifications are occurring. Sliding windows accomplish two purposes. For one, the sliding windows allow the location of a signal mismatch to be determined. Additionally, the windowed comparison produces a more accurate result than simply correlating the entire sequence at once.

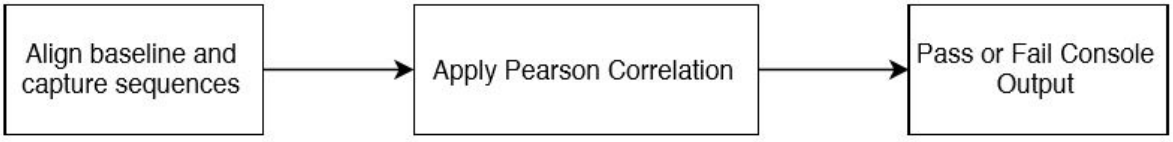

Figure 25: This diagram is the basic outline of the analysis process

Rather than directly comparing the baseline to an alternate capture, windowing is used in order to ensure a brief amplitude spike caused by external noise or other factors does not influence the result.

This also has the advantage of being a controllable sensitivity knob; the larger the window the less chance of being affected by random noise, and the larger the step size the less granularity of the detected differences. Conversely, having a smaller window size is able to provide better sensitivity, and a smaller step size provides better feedback on the location of the modification.

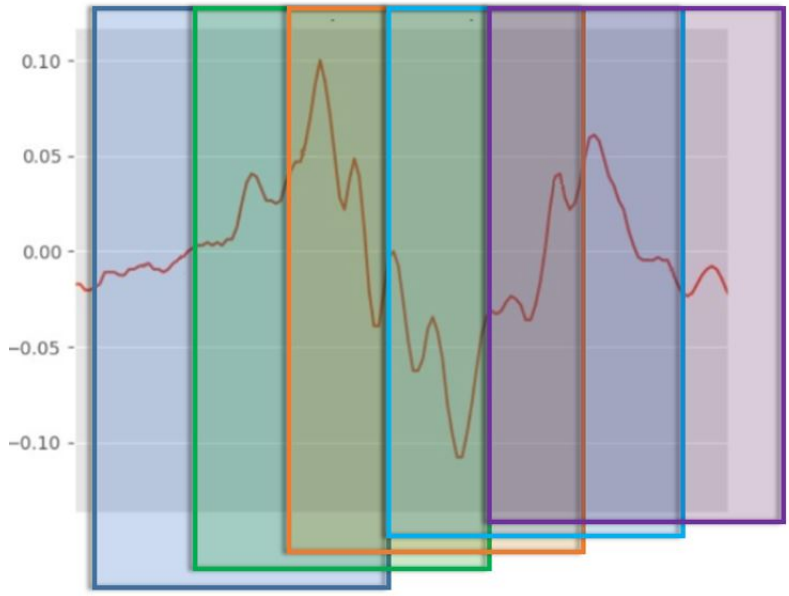

Figure 26: This diagram shows a visual representation of the sliding windows over the capture data.

5 Software Program

5.1 Summary

The software algorithm is combined into one program. The program has two options: baseline and analyze. The baseline option of the algorithm requires any number of baseline captures to create a base model file. The base model file is then used in the analyze option of the program, which in addition takes any number of modified captures determining whether or not the capture has been modified, returning a pass if they are the same and a fail if they are different. If the correlation coefficient is below a specified value, then the program will inform the user.

5.2 Software Program Input/Output and Outline

Basic input for program.py baseline:

- length variation: acceptable difference in capture length from other captures in the sequence.
- debug: write filtered baseline models each in order to visually inspect results.

- baseline file: name of file to write the base model output to (i.e. base.model or test.model). This file is created from gathering all the data from the baseline capture files to create a baseline model that will be fed to the analyze portion of this program.
- baseline capture(s): baseline capture files which are used to create a baseline model
- threshold front: front threshold value for start of sequence in order to aid with the alignment of the first peak.
- threshold back: back threshold value for back of sequence; the algorithm currently uses the variation of port toggles in order to signal the end of testable bootcode.

Below is the help menu for the baseline option of the program. We have a script that runs this program with default values for quick results.

```
lydienge@DESKTOP-C904LOB:/mnt/c/Users/Lydia Engerbretson/Documents/CS 480/tater_2/tater/sw$ python3 program.py baseline -h
usage: program.py baseline [-h] [--length_variation LENGTH_VARIATION]
                           [--debug DEBUG] [--baseline_file BASELINE_FILE]
                           [--baseline_captures BASELINE_CAPTURES]
                           [--threshold_front THRESHOLD_FRONT]
                           [--threshold_back THRESHOLD_BACK]

optional arguments:
  -h, --help            show this help message and exit
  --length_variation LENGTH_VARIATION, -lv LENGTH_VARIATION
                        acceptable difference in capture length from the
                        length of the baseline
  --debug DEBUG, -d DEBUG
                        write filtered baseline models each in order to
                        compare results
  --baseline_file BASELINE_FILE, -b BASELINE_FILE
                        name of file to write base model output to, required
  --baseline_captures BASELINE_CAPTURES, -f BASELINE_CAPTURES
                        file with list of capture files to add to the baseline
  --threshold_front THRESHOLD_FRONT, -tf THRESHOLD_FRONT
                        front threshold value for start of sequence
  --threshold_back THRESHOLD_BACK, -tb THRESHOLD_BACK
                        back threshold value for Back of sequence, this is
                        port toggles
```

Figure 27: This output will appear if you enter: python program.py baseline -h or –help

Basic input for program.py analyze:

- window size: number of data points analyzed within each window
- window divisions: what fraction of the window size it is moved forward each iteration
- accepted correlation: passing correlation value
- length variation: acceptable difference in capture length from the length of the baseline
- baseline file: the base model file (i.e. base.model or test.model) to compare modified captures against
- capture file (s): capture files for analysis

Below is the help menu for the analyze option of the program. We have a script that runs this program with default values for quick results.

```
lydienge@DESKTOP-C904LOB:/mnt/c/Users/Lydia Engerbretson/Documents/CS 480/tater_2/tater/sw$ python3 program.py analyze -h
usage: program.py analyze [-h] [--length_variation LENGTH_VARIATION]
                          [--debug DEBUG] [--baseline_file BASELINE_FILE]
                          [--capture_file CAPTURE_FILE]
                          [--window_size WINDOW_SIZE]
                          [--window_step WINDOW_STEP]
                          [--window_correlation WINDOW_CORRELATION]
                          [--average_correlation AVERAGE_CORRELATION]
                          [--show_animation SHOW_ANIMATION]

optional arguments:
  -h, --help            show this help message and exit
  --length_variation LENGTH_VARIATION, -lv LENGTH_VARIATION
                        acceptable difference in capture length from the
                        length of the baseline
  --debug DEBUG, -d DEBUG
                        write filtered baseline models each in order to
                        compare results
  --baseline_file BASELINE_FILE, -b BASELINE_FILE
                        baseline to compare against
  --capture_file CAPTURE_FILE, -c CAPTURE_FILE
                        capture to analyze
  --window_size WINDOW_SIZE, -wz WINDOW_SIZE
                        number of data points analyzed within each window
  --window_step WINDOW_STEP, -ws WINDOW_STEP
                        what fraction of the window size it is moved forward
                        each iteration
  --window_correlation WINDOW_CORRELATION, -wc WINDOW_CORRELATION
                        number of data points analyzed within each window
  --average_correlation AVERAGE_CORRELATION, -ac AVERAGE_CORRELATION
                        total number of bad correlations detected averaged in
                        each window
  --show_animation SHOW_ANIMATION, -gif SHOW_ANIMATION
                        show the pass/fail gif at the end of verification
```

Figure 28: This output will appear if you enter: python program.py analyze -h or –help

5.3 Software Program Configuration Changes

- Configurations vary in length due to variable time loading peripherals
- Certain configurations may occur in multiple locations
- The system only detects a limited subset of configuration changes, because as more configurations are added 2$\hat{}$(number of configurations) of baseline captures are required.

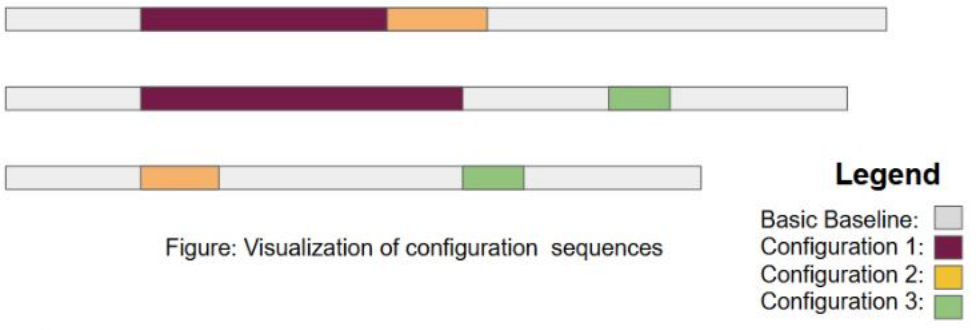

Figure 29: This diagram is a visual representation of configuration changes.

5.4 Software Program Challenges

- As a team with client approval, we made the decision that we will not implement the detection of boot code peripheral responses within the software algorithm. Some peripheral initialization times may vary significantly and this creates "dead space" within the capture data where detection of foreign modification would not be possible. Some potential solutions are:
  - Detect a sequence of instructions used in the machine code that poll for a peripheral. If detected in code, then ensure a corresponding match in the baseline signal.
  - Build a list of features that occur at particular regions of the boot code, and check for the appearance of these features.
  - Use machine learning in order to "learn" how a peripheral behaves.
- Currently the system is only able to detect a limited subset of boot code configurations. This is limited due to time of collecting baseline captures.
- We are not able to achieve complete perfection with detections of modifications within the capture data because we are not sampling the data at a fast enough rate. The current hardware setup is physically not able to sample fast enough. Since we do not have a lot of data points, it is hard to match and correlate the data, especially places in the capture data where there are not any peaks. Also, modifications of small instructions such as additions, subtractions, and multiplications are difficult to detect because changes of a similar magnitude occur due to environmental noise and other variations, whereas larger instructions such as loads and stores can be recognized consistently.

6 Algorithm Verification and Testing

To test and verify that the software algorithm worked, we ran many different modification files against a baseline. Listed below are the different tests that we ran against the software algorithm.

- Default boot code hardware captures, including configuration changes
- Insertion and deletion modifications
- Modified looping instructions for maximum detection
- Detection of single instruction modifications
- Large boot code modifications

If the correlation value was 95 percent or better, than the program would output a pass value, if it was less than 95 percent, then the program would output a fail value.

7 Results

Any insertion and/or deletion of assembly-level instructions are able to be detected. This was tested by programming the target board with a custom boot sequence consisting of approximately 20 instructions, and proceeding to remove or add a single instruction.

7.1 Instruction Modifications within Boot Code

The following tests were done by gathering captures of boot sequences with a single instruction inserted/modified in that boot sequence. If something fails then the algorithm was not able to detect that an insertion/modification was made. If something succeeds then the algorithm was able to determine that an insertion/modification was made. Some tests were not done and marked as N/A.

The iterations loop was used to test for the presence of modifications based on repeated iterations. This was done in order to alleviate the potential miss of a single instruction by the algorithm. The code listing for the `exclusive-or` loop is shown below, with register 27 set to 100.

```
decrements:
    eor r20,r21
    dec r27
    cpi r27,0
    brne decrements
```

The figure below illustrates the results of these tests:

Original Instruction	Modified Instruction	Single Change	Loop (100 iterations)
Eor 255	Eor 0	N/A	Success
And 255	And 0	Fail	Success
Add 0	Sub 0	Success	N/A
Add 0	Add 3	Fail	Fail
Add 255	Add 0	Success	Success
Mul 0	Mul 3	Fail	Success
Mul 0	Mul 255	Fail	N/A
Sts 0 - Addr 0	Sts 255 - Addr 0	Success	Success
Sts 0 - Addr FF	Sts 0 - Addr FF	Success	Success
Sts 255 - Addr 0	Sts 255 - Addr FF	Success	Success
Lds 0 - Addr 0	Lds 255 - Addr 0	Success	Success
Lds 0 - Addr FF	Lds 0 - Addr FF	Success	Success
Lds 255 - Addr 0	Lds 255 - Addr FF	Success	Success
Out 0	Out 255	Success	Success

Key	
Fail	Less than 95% Convergence
Success	More than 95% Convergence
N/A	Not Tested

Figure 30: Table containing the results of comparing modified instructions

7.2 Conclusion

We found that there is a direct relationship between a machine level instruction and the electromagnetic emissions generated during the execution of a microprocessor. That as instructions are executed a unique profile is created in the electromagnetic fields surrounding the device under test. We devised a bench-top setup that if isolated from sources of noise can gather accurate captures of the electromagnetic waveforms characteristic of the processors execution. We also devised an algorithm that could correlate between captures searching for dissimilarities between the signals.

7.3 Project Improvements

- Shielding - Some basic EM shielding ideas would be to encase the whole acrylic L-Shaped platform in a box that would cancel any external noise form lights, or nearby devices. However, the noise would still be present from devices/peripherals attached to the device under test. This latter issue requires further work to provide possible solutions.
- Higher Sampling Rate - As our maximum frequency of interest is 250MHz by Nyquist that

requires at least 2X sampling rate or 500MS/s. However, in order to ensure that the signal being captured is as smooth as possible a higher sampling rate would be desired. The reason for this is that the algorithm works the best if whatever variance between different signals is removed.

- Data Streaming - We would have like to use the KC705-FMCADC4 setup that would allow us to stream data at 1GS/s, instead of only having a fixed capture window at 1GS/s which was the case for the oscilloscope setup. Any future setup doesn't specifically need the KC705-FMCADC4, just something that provides high sample rates with data streaming.
- Peripheral Detection - We would like to be able to detect operations that access peripherals; However, as peripherals take a variable amount of time to be accessed this means our algorithm would need to support areas of possible dead space where there may or may not be anything depending on the time it takes to access the peripheral.
- Modification of Similar Instructions - We would like to improve the accuracy of our test setup as to notice the relatively small changes between similar instructions. This could be done by a combination of improving the signal to noise ratio and optimizing the algorithm.
- Hardware Testing - With the current test setup it is quite difficult to adjust or remove anything from the setup without noticeably changing and probably breaking the capture setup. However, if everything was encased in a shielding box different triggers could be used to ensure that no foreign actor had access to the inside of the box.

7.4 Practical Setup

If the idea behind this project was to be implemented at an industry level we see two possible models. The first model would be a portable device that could be taken into the field to test processors. However, this would be susceptible to noise and would require the probe to be placed in exactly the same location before every test. The second model would be a system where the function of our project would be implemented in the design phase and totally integrated with the board. This would ultimately be more costly, but less susceptible to noise and variability between tests.

8 Code Listings

All code is currently available on the GitLab repository and will be delivered however the client deems suitable.

9 Acknowledgements

Many thanks to Idaho Scientific for both sponsoring and mentoring this project. Thanks to our faculty advisor Dr. Li for his assistance in project management, Dr. J. Frenzel for the use of his lab, Dr. Shih for advice on antennas, and to John Jackshaw for supplies.

10 References

1. Fred B, Marc W, Bartek G, Yijun S: *Practical Electro-Magnetic Analysis*.
https://csrc.nist.gov/csrc/media/events/non-invasive-attack-testing-workshop/documents/03_debeer.pdf

2. Nader S, Alireza N, Alenka Z, Milos P: *Spectral Profiling: Observer-Effect-Free Profiling by Monitoring EM Emanations*.
http://alenka.ece.gatech.edu/wp-content/uploads/sites/463/2016/08/MICRO16.pdf

3. Nader S, Alireza N, Alenka Z, Milos P: *EDDIE: EM-Based Detection of Deviations in Program Execution*.
http://alenka.ece.gatech.edu/wp-content/uploads/sites/463/2017/06/IS-CA17.pdf

4. Robert C, Alenka Z, Milos P: *A Practical Methodology for Measuring the Side-Channel Signal Available to the Attacker for Instruction-Level Events*.
http://ieeexplore.ieee.org/document/7011392/?reload=true

5. Leonid Rozhkov: *Design Efficient FFT Digital Processor for Electromagnetic Fields*.
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=565747

6. Angelos Keromytis: *Leveraging the Analog Domain for Security*.
https://www.darpa.mil/program/leveraging-the-analog-domain-for-security

7. Paul K, Joshua J, and Benjamin J: *Differential Power Analysis*.
http://gauss.ececs.uc.edu/Courses/C653/lectures/SideC/lecture11-dpa.pdf

8. Strazdins, K. Homma, K. Nagase, V. Nguyen, M. Noro, T. Yamagajo: *Efficient Parallel Electromagnetic Field Analysis Exploiting Symmetry*.
http://users.cecs.anu.edu.au/~peter/papers/AF.html

9. J.F. Frenzel: *Power-supply current diagnosis of VLSI circuits*.
https://ieeexplore.ieee.org/document/285105/

10. J.F. Frenzel: *A comparison of methods for supply current analysis*.
https://ieeexplore.ieee.org/document/164086/