ECE 493

Senior Design Final Document

Side Channel Analysis on Microcontrollers

Team Members:
Theresa Nguyen, Jorge Rodriguez, Vincent Guevara, Bradford Webb

Faculty Supervisor:
Dr. Jens-Peter Kaps

1. Executive Summary

Confidentiality of data as well as data authentication accomplished through cryptographic algorithms is a relevant topic with the growing usage of devices that are interconnected through wireless communication. However, running cryptographic algorithms on hardware and software can leak information, such as power consumption, that attackers can exploit in order to break the cryptosystem. This type of exploitation is known as a side-channel attack, and it is directed at the information gained as a result of the leakage, rather than the weakness of the particular algorithm used [1]. This occurrence is not ideal for security because the leaked heat and voltage emissions may allow an attacker to access the encrypted key code via reverse engineering.

The proposed project will provide a system that benchmarks hardware performance when running cryptographic algorithms on microcontrollers. The system will consist of a FOBOS (Flexible Open-source workBench fOr Side-channel analysis) shield, a PYNQ board, and a microcontroller. The PYNQ based side-channel analysis testbed will provide a platform to run Linux, and obtain power consumption measurements. The FOBOS shield board mainly performs side-channel analysis, as well as regulates power to the microcontroller via a voltage regulator chip. Lastly, the microcontroller will serve as the device-under-test (DUT) that runs cryptographic operations and returns the timing results of the system. Each of the boards will be interfaced using I2C serial protocol and configured to try and acquire measurements and information in the form of RAM usage, ROM usage, runtime, power consumption, and power leakage from the system side-channels. These measurements will be the outputs obtained by the system as a cryptographic algorithm is implemented. Analysis of this information will allow us to run tests to determine whether the secret key is protected or not.

This project is intended to serve as an educational tool for students interested in the topic of cryptography. Due to the lack of readily available, completely open-source software packages for side-channel analysis that include drivers for testing on the board, it is difficult to educate students about side-channel analysis with a hands-on approach [2]. Additionally, this project could potentially provide an easily accessible way for companies to analyze their devices for hardware vulnerabilities at a much more reasonable cost, since current solutions tend to be particularly expensive. Meanwhile, FOBOS at most is a platform that costs just under $500 for the logic analyzer and control board that the FOBOS shield would go on top of in assembly[3][4].

Table of contents

2. Approach

2.1 Project Origin and Solution to problem

Side-channel analysis attacks pose a grave threat to implementations of cryptographic algorithms in both software and hardware [5]. FOBOS was first developed in 2012 in order to evaluate hardware implementations on Field Programmable Gate Arrays (FPGAs) since there is a lack of readily available, complete open-source software pages for side-channel analysis that include drivers for testing on the board, or the system is available but it is very expensive. Meanwhile, FOBOS can obtain measurements and analyze them for just under $500, which makes it more suitable and affordable for educational as well as research purposes[3][4].

Currently, the FOBOS system is only able to obtain information leakage through power side channel analysis from FPGA devices, and not microcontrollers. FOBOS also uses an additional system called XXBX (eXtended eXternal Benchmarking eXtension) in order to obtain ROM usage, RAM usage, runtime and power consumption in order to have precise power measurement and computation of energy consumption.The purpose of this project is to integrate the XXBX functionality into the FOBOS system while also allowing FOBOS to obtain measurements not only from FPGA devices, but also expand to microcontrollers as well.

2.2 Alternative Designs

A different FPGA board can be used in the case that the PYNQ-Z1 is no longer in production, or is not compatible with the DUT or if the board is cheaper than the PYNQ. Because the components of FOBOS are built in a modular fashion, it can be easily replaced and adapted for new FPGA boards with few to little changes. Another alternative design is to fix the excessive heat produced by the voltage regulator chip on the FOBOS shield during processing. In order to regulate the excessive heat, we can add a heat sink, rewire components on the board, or/and change the size of copper area on the chip.

2.3 Individual Contributions

- **I2C Software**
  - Theresa: Confirmed I2C communication between two MSP430s, used existing AXI IIC functions to setup communication between PYNQ board and microcontrollers, and tested this with a logic analyzer.
  - Vince: Tested I2C on Arduino Uno and verified with Arduino serial monitor
- **I2C & PYNQ Hardware**
  - Vince: Incorporated AXI IIC IP and Timer Unit into existing PYNQ Controller block design
  - Jorge: Design and testing of Timer Unit in VHDL using Vivado Design Suite
- **XXBX**
  - Vince: Setting up the existing XXBX platform and confirming board communication using a logic analyzer
  - Jorge: Compiling and running a program on the existing system, documenting the protocols sent between the respective boards, and research into the functionality of the XXBX system
- **Power Measurement**
  - Brad: Power Consumption Configuration and Testing, research into Jupyter Notebook on PYNQ, and ADC on FOBOS Shield

3. Technical Section

3.1 Existing Models

3.1.1 Overview of the FOBOS System

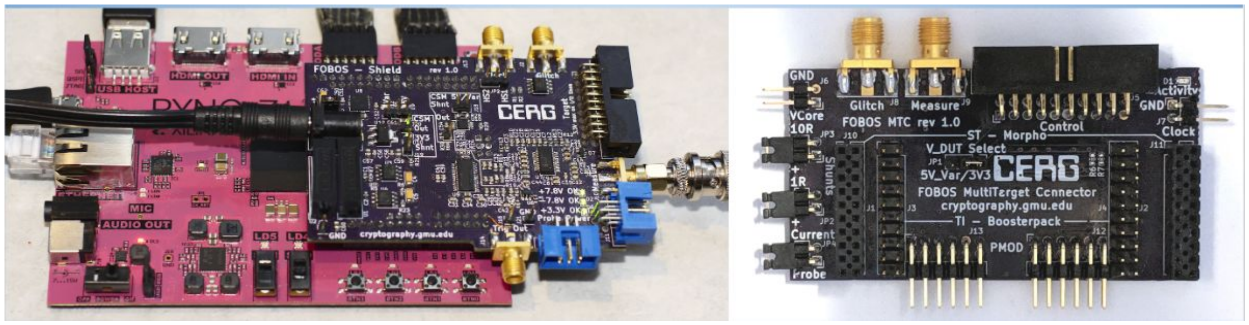


Figure 3.1: PYNQ Board and FOBOS Shield (Left), Multi-Target Connector (Right)

As discussed previously, the FOBOS system already existed before this project was proposed, however, it has gone through several iterations. The implementation of the proposed system in this project would mark the third version of FOBOS or "FOBOS 3." Above is an image of the existing components used in the FOBOS system. Beginning on the left is the PYNQ-Z1 python productivity board. This board consists of dual core ARM processors running Linux, Python, and Jupyter notebooks. It also has FPGA fabric for user customized programming. The board sitting on top of the PYNQ is the FOBOS Shield which provides power to the device under test and handles power consumption measurement through a high speed ADC. The board on the right is the Multi-Target Connector which serves as a connector between the DUT and the PYNQ board.

3.1.2 Overview of the XXBX System

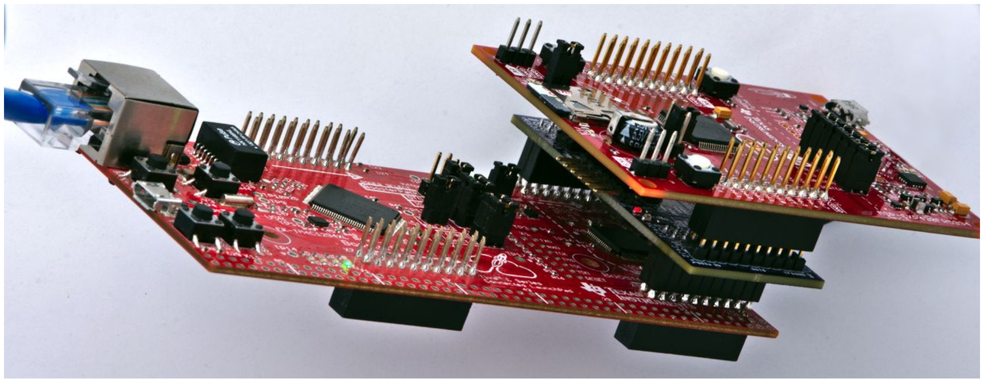


Figure 3.2

XXBX is a system that also already existed before this project was proposed. It can measure four of the desired metrics: ROM usage, RAM usage, runtime, and power consumption. This system consists of four components: the XXBX System (XBS), the XXBX Harness (XBH), the XXBS Power Shim (XBP), and the XXBX Device under Test (XBD). In the photo above, the ethernet is connected to the XBS (not shown), the bottom board is the XBH, the blue board (in the middle) is the XBP, and the board on top is the XBD. These will be discussed in detail in section 3.3 XXBX Integration.

3.2 Top-Down Design

3.2.1 Level 0 Diagram

The image below depicts the top level design or the Level 0 data flow diagram of the system. This diagram is meant to display the overall system inputs and outputs, and what the internal operations are. The main external inputs to the system are power supply and the user overseeing the system functionality. An intermediate input is the program for the cryptographic algorithm being implemented, which is included on the control board's System on Chip (SoC) and is passed to the target device. The system should be able to take in all the external inputs and through software, implements XXBX benchmarking, runs cryptographic algorithms, and computes metrics in the form of ROM usage, RAM usage, runtime, power consumption, and information side channel leakage.

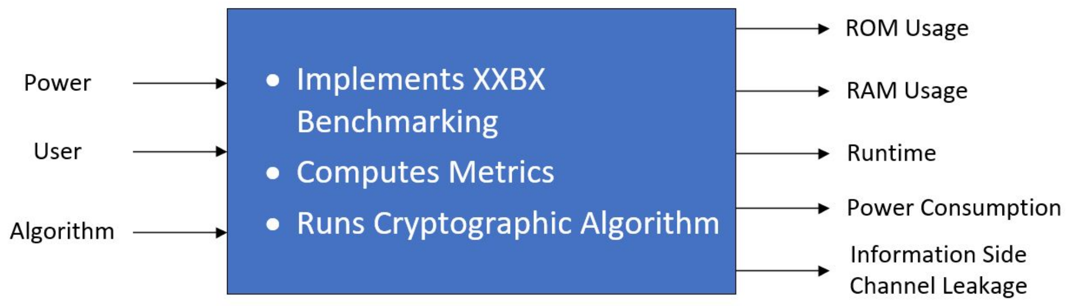

Power
User
Algorithm
Implements XXBX Benchmarking
Computes Metrics
Runs Cryptographic Algorithm
ROM Usage
RAM Usage
Runtime
Power Consumption
Information Side Channel Leakage

Figure 3.3

3.2.2 Level 1 Diagram

The level 1 data flow diagram (shown in Figure 3.4) breaks down level 0 diagram into each device that would form the whole system. The PYNQ, Shield, Multi-target connector, and the device under test will all be transmitting and receiving information via I2C serial communication protocol. As mentioned in the previous Level 0 section, a power supply is needed in order to power on the PYNQ, which will then supply the power to the rest of the boards. The PYNQ will cross-compile the algorithm code to be sent to the DUT. Based on a timing signal delivered from the DUT, the PYNQ will determine how long the DUT spent running cryptographic operations. The purpose of the shield is to facilitate power consumption measurement and regulate power going to the DUT. The Multi-target connector can vary how much voltage is needed in order to power up the DUT. It also allows different types of microcontrollers to be tested for side channel analysis.

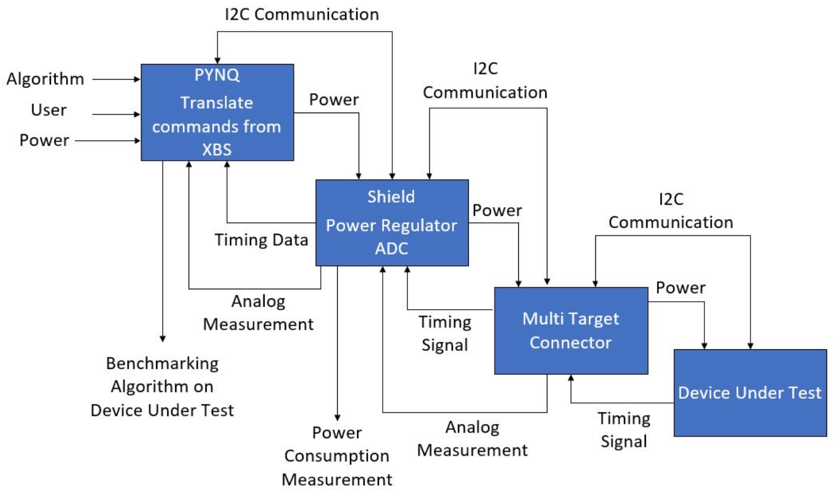


Figure 3.4

3.2.3 Level 2 Diagram for PYNQ

Now that the system has been described at the level one decomposition stage, it is possible to go further into the details of each individual component. The figure below shows the level two decomposition of the PYNQ board. The purpose of the ethernet input is to interface the System-on-Chip (SoC) on the PYNQ board, which contains an ARM processor that runs Linux OS and has Jupyter notebooks installed on it, with a user PC. This allows the user to view the files stored on the chip, and run the python code on it. The power signal that the PYNQ board is receiving comes from an AC adapter that is plugged into an electrical outlet. The adapter supplies 12V, 3A of power to the PYNQ board. This signal is then passed into a voltage/power regulator, so that the signal can be attenuated to 5V in order to power the FOBOS shield board.

In addition , the PYNQ board also contains an XADC that accepts analog power measurements that are returned from other parts of the system. These measurements are critical in assessing the power consumption of the device under test while it is running cryptographic operations. The clock wizard manages and delivers clock signals to several components such as the ADC interface, timer module, and IIC (inter-integrated circuit) module. A power control module sends power settings to the FOBOS shield regarding voltage and gain levels. Lastly, the IIC module handles the information being sent and received by the PYNQ board through I2C communication. The PYNQ board is the control board of this system, which is why it not only sends commands and configurations, but also receives all of the data and measurements for final analysis at the end of the runtime.

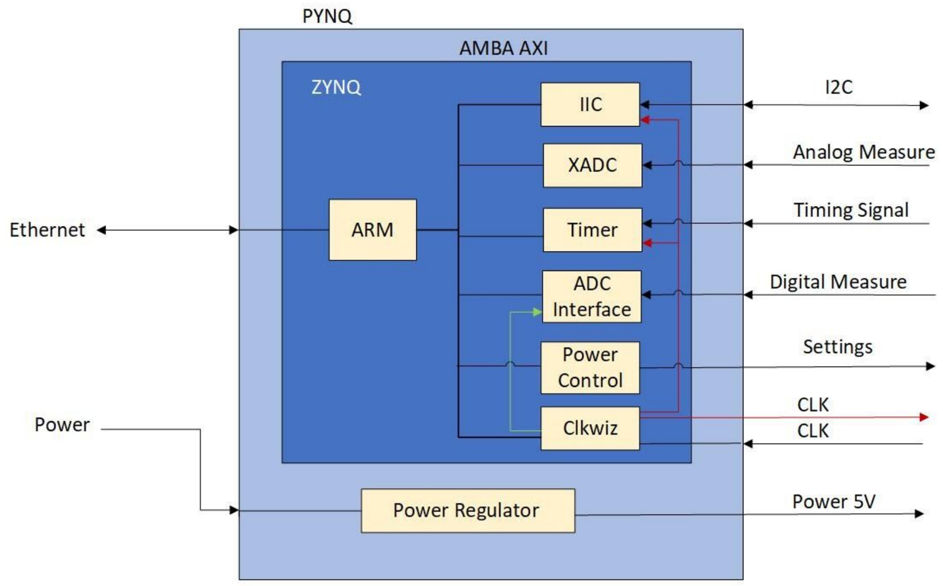


Figure 3.5

3.2.4 Level 2 Diagram for FOBOS SHIELD

The FOBOS shield is responsible for providing power to the DUT, as well as handling the power consumption measurement used for side-channel analysis through the high speed ADC. The shield provides several different values for powering the DUT through two voltage regulators and jumpers. This provides the ability for more than just one specific device to be used as the DUT, since different devices may have distinct power requirements. The current sensing modules amplify the current drawn from the DUT in order for the XADC on the PYNQ board to be able to obtain analog current measurements for power consumption analysis.

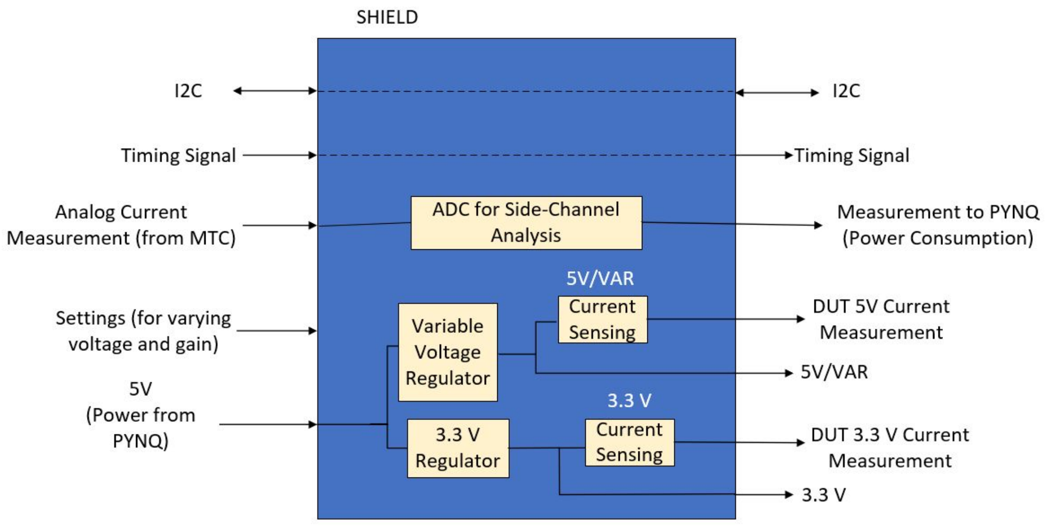

SHIELD
I2C
I2C
Timing Signal
Timing Signal
Analog Current Measurement (from MTC)
ADC for Side-Channel Analysis
Measurement to PYNQ (Power Consumption)
5V/VAR
Current Sensing
DUT 5V Current Measurement
Settings (for varying voltage and gain)
Variable Voltage Regulator
5V/VAR
5V (Power from PYNQ)
3.3 V
3.3 V Regulator
Current Sensing
DUT 3.3 V Current Measurement
3.3 V

Figure 3.6

3.2.5 Level 2 Diagram for Multi-Target Connector

The Multi-target connector (MTC) serves as a connector between the PYNQ board and the DUT. It passes the timing signal and I2C communication that it receives from the PYNQ board to the DUT. The MTC has 3.3V and 5V/ variable volts pins in order to supply the correct voltage required by the DUT. Inside the MTC, there is a shunt resistor which allows for current measurement for side-channel analysis to be obtained. These current measurements will then be sent to the high speed ADC on the shield.

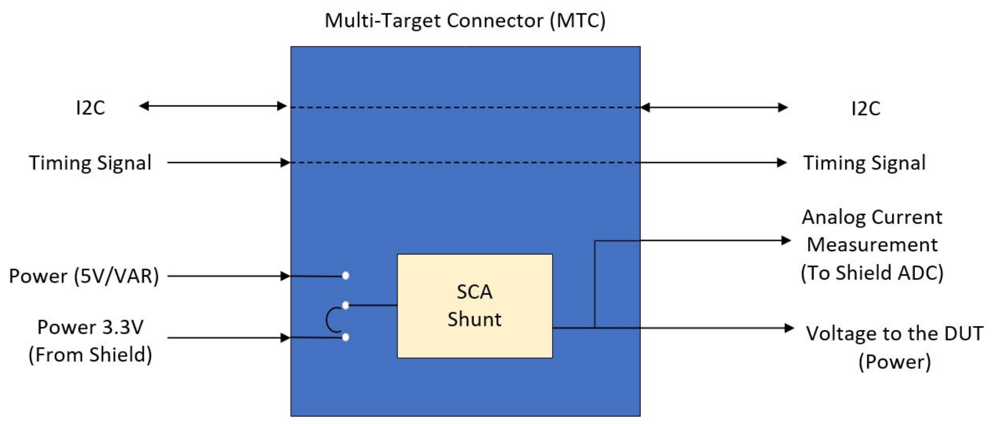

Multi-Target Connector (MTC)
I2C
Timing Signal
Power (5V/VAR)
Power 3.3V (From Shield)
SCA Shunt
I2C
Timing Signal
Analog Current Measurement (To Shield ADC)
Voltage to the DUT (Power)

Figure 3.7

3.2.6 Level 2 Diagram for Device Under Test

The DUT is the device that will perform the cryptographic operations it received in the program from the PYNQ board. The DUT has a bootloader which will compile programs without the help of an external compiler. After compiling the program, the bootloader will load it into the function core in order for the program to run. Once the program is finished running, the DUT will send the timing signal and stack usage measurements back to the PYNQ board for analysis.

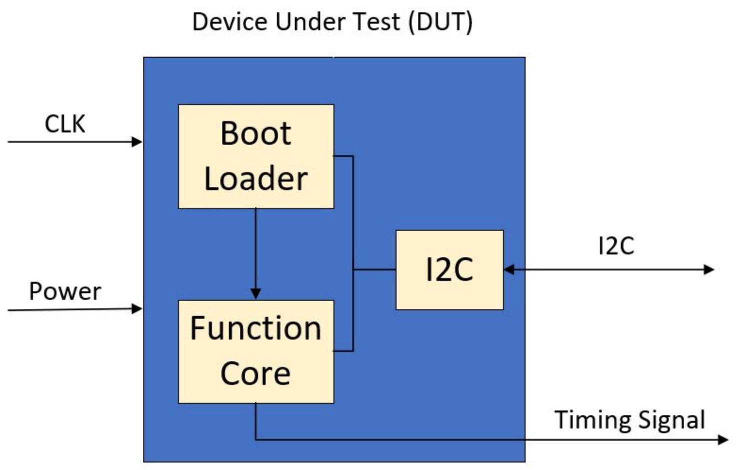


Figure 3.8

3.3 XXBX Integration

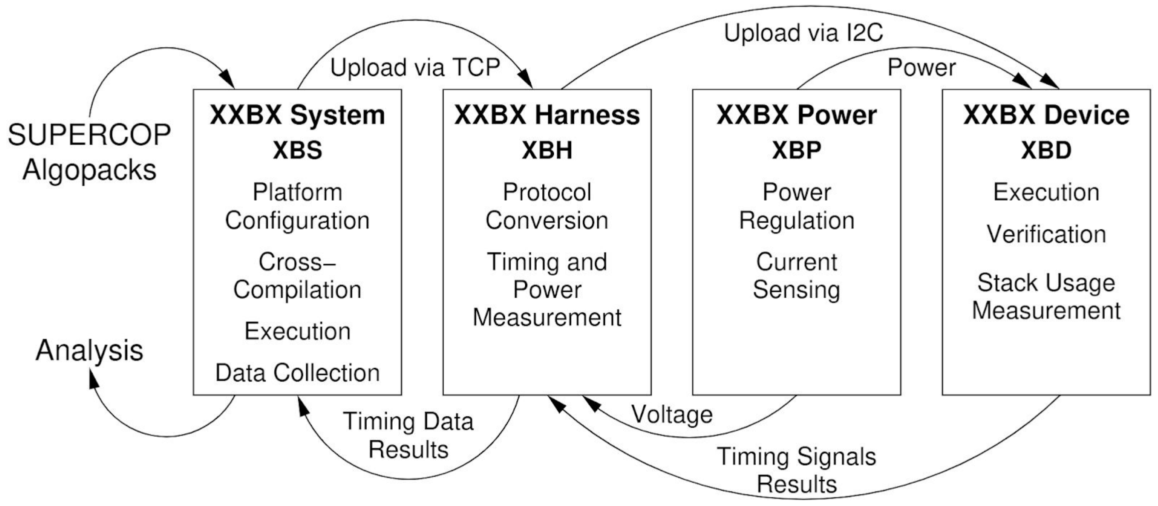


Figure 3.9

Since integrating the existing FOBOS and XXBX systems into one system, this section will identify what components of the FOBOS system should be extended to implement the XXBX protocol. The appropriate components and their responsibilities of the existing XXBX system is decribing in the diagram above. Ideally, the PYNQ board will work as the XXBX System (XBS) and XXBX Harness (XBH) after the XXBX protocol is integrated into the new FOBOS project. As the XBS, the PYNQ board should cross-compile cryptographic algorithm code and provide the ROM usage measurement. As the XBH, the PYNQ will handle protocol conversion and send data and instructions to the rest of the system via I2C, measure power consumption of the target device with the help of the FOBOS shield board, and measure the runtime of the cryptographic algorithm through a timing signal obtained from the DUT.

The FOBOS shield will serve as part of the XXBX Power shim (XBP) and will perform power regulation to the DUT, and facilitate analog measurements through current sensing modules. Additionally, the MTC board will serve as another part of the XBP by providing some more current sensing functionality in order to get the side-channel analysis current measurement that will be an input to the high speed ADC on the FOBOS shield board.

Lastly, the DUT or micrcontroller will serve as the XXBX Device (XBD). Its job is to run the cryptographic operations, and send results back to the XBH (PYNQ board). The DUT will also provide the stack usage measurements or the RAM usage metric that was discussed as one of the essential outputs of the system.

3.4 Physical Architecture

The overall system of the FOBOS 3 on Microcontroller (shown in Figure 3.10) is made up of the four main components that have been described in detail in the Top-Down design. This diagram shows the responsibilities of each board at a high level. The PYNQ which is the control of the system will be responsible for generating test vectors that will be sent to the microcontroller (shown as an MSP430 in the image). The PYNQ is also responsible for measuring the time it took for the DUT to finish running the algorithms. The shield has current sensing and signal conditioning that will modify the signal in a way that the XADC on the PYNQ would be able to measure. The XADC in the PYNQ is responsible for measuring the power consumption, which is the input we need in order to perform t-tests. The MTC has a current sensing module that will modify the signal so that the high speed ADC in the shield would be able to accept the measurement. The high speed ADC in the FOBOS will provide the power variation that occurred inside the MTC. While this is all repeated information, it is important to reiterate the jobs of each component, and how they are interconnected to formulate the entire system.

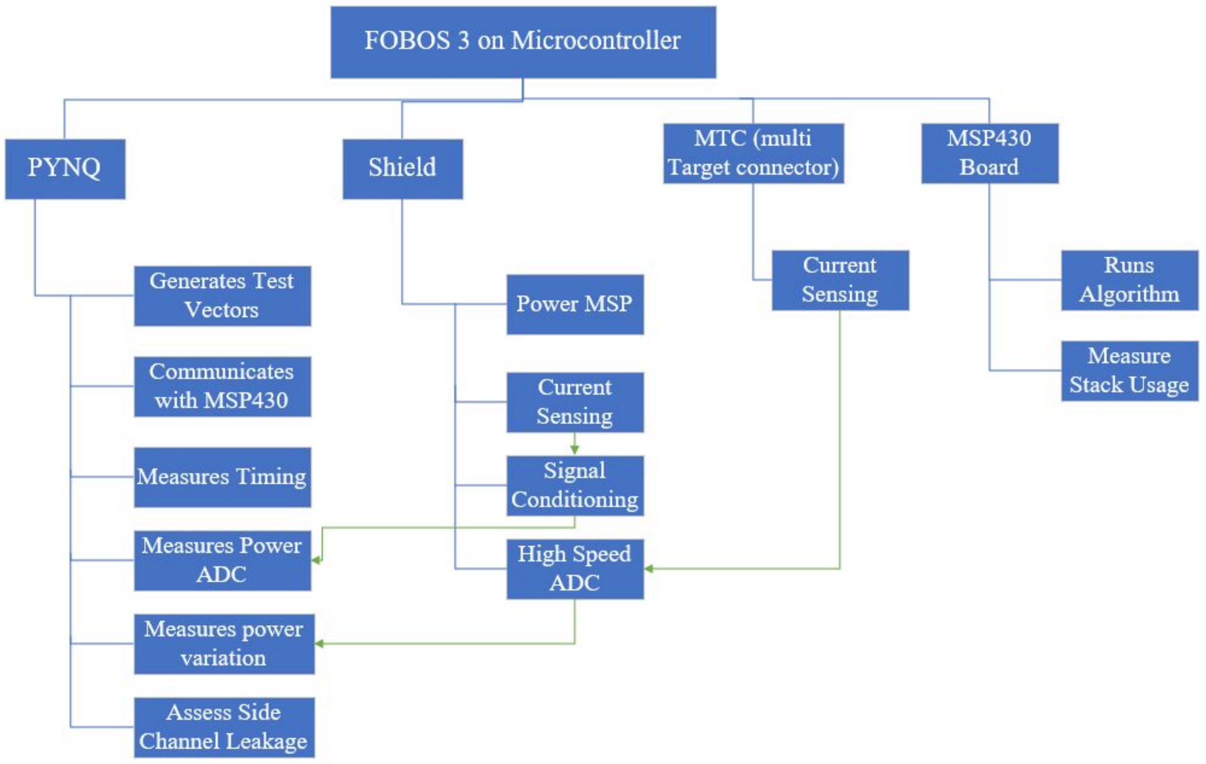


Figure 3.10

3.5 Overall System Architecture

Figure 3.11 shows how the system is connected and the end-to-end flow from the computer to the DUT and back. The system starts once the user starts the Jupyter notebook to run a test, the text in the same directory goes from the PYNQ to the shield, through the multi-target connector, and then to the DUT. Once the data is at the DUT, the algorithm contained inside of it is executed and the stack usage is measured. That stack usage measurement goes back from the MSP430 through the Multi-Target Connector and to the FOBOS shield. As the data goes from the DUT to the Shield , the current of the DUT is sent to the High Speed ADC on the FOBOS shield and will be later compared to the current from the FOBOS shield. Now the data the PYNQ did not have access to from the other components is now back to the PYNQ for final processing so it can determine if there is side channel leakage, the timing of the entire process, the RAM and ROM used, and the power consumption and variation. Resultant ciphertext is placed in the pre-determine directory based on the python code of the jupyter notebook on the host computer.

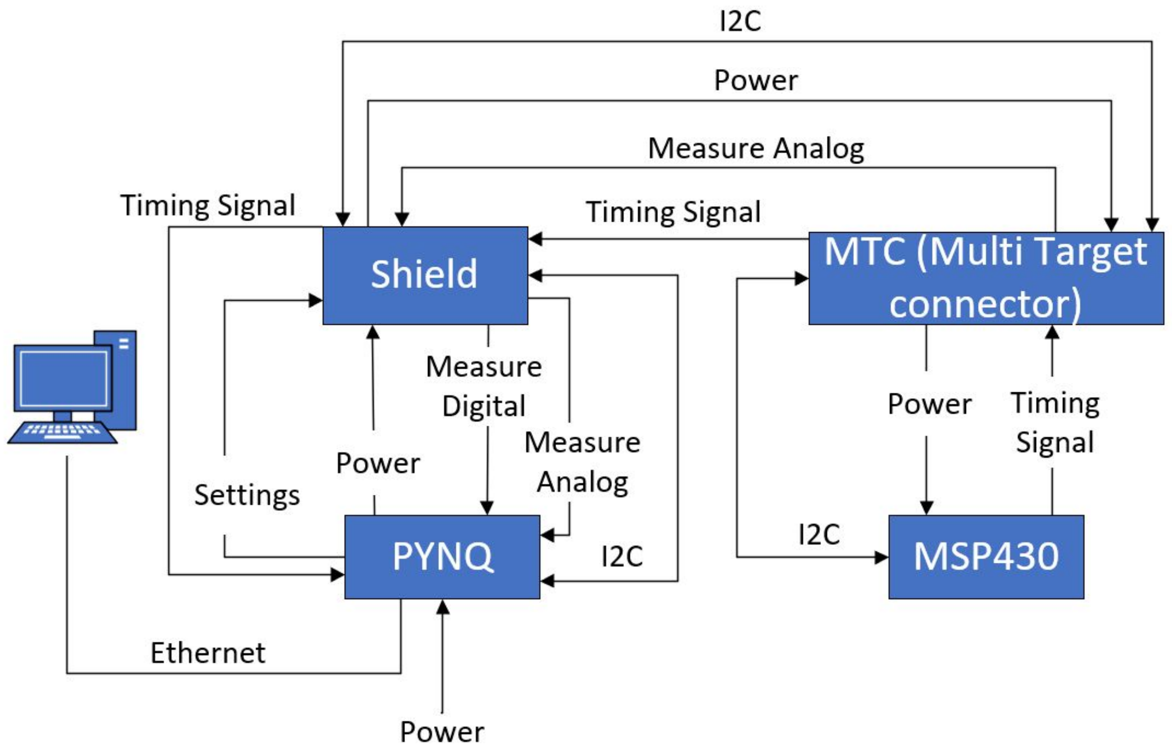


Figure 3.11

3.6 Detailed Schematics

3.6.1 PYNQ Controller Schematic

Figure 1 below is a screenshot of the PYNQ block diagram as shown in Vivado. These are all the components that make up the PYNQ controller of the FOBOS system. This FPGA design is then synthesized and implemented to generate a bitstream file used to program the board. Once the bitstream is in the PYNQs memory, we create an overlay in the Jupyter Notebook which allows us to use Python as an interface to access the components in our design.

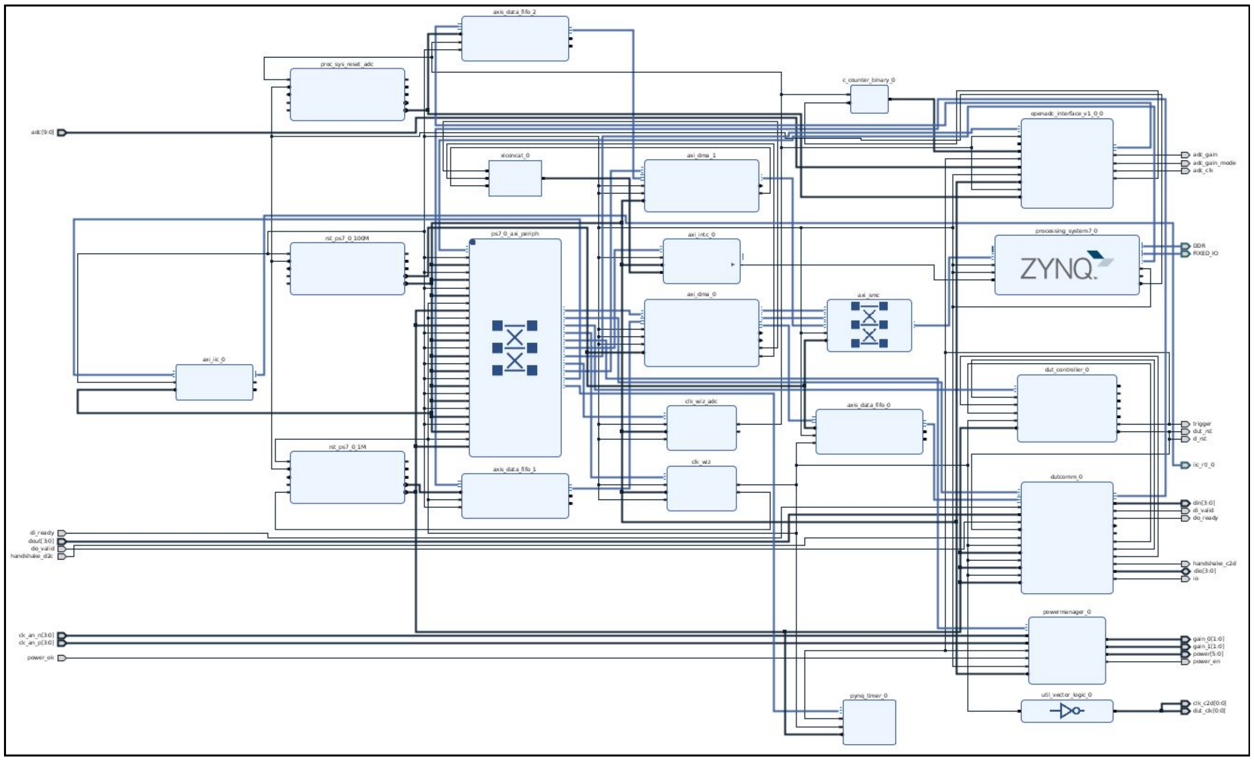


Figure 3.12: Pynq Controller Block Diagram

3.6.2 FOBOS Power Schematic

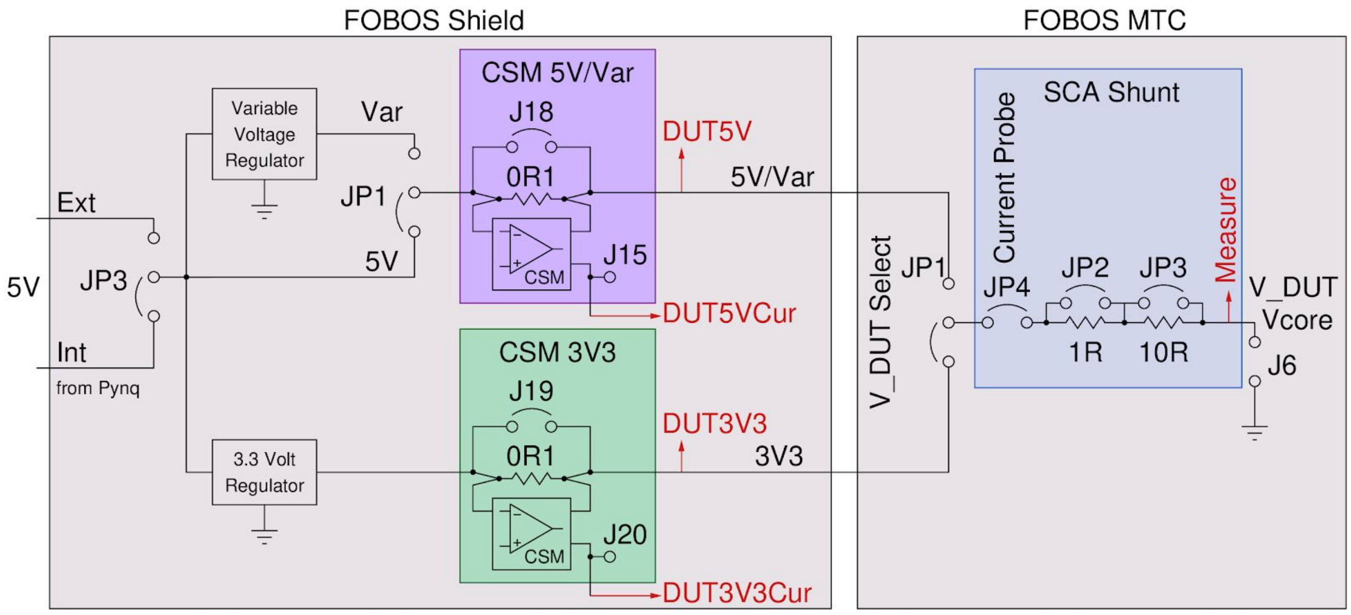


Figure 3.13: Power Schematic for FOBOS

Figure 3 shows the power schematic for the FOBOS Shield and Multi Target Connector. JP3 acts as a point of decision for if the user wants to regulate the input voltage (5V) to 3.3V or decide on a later decision of 5V/Var. JP1 is the point of decision for either 5V or Variable voltage regulation. After choosing the voltage regulation options, one is able to measure current or voltage depending on the jumper pin position of J19 and J18 for 3.3 and 5/Var Volts respectively. Measurement is done either through code run on the jupyter notebook activating the ADC existing on the PYNQ board or via multimeter and with an appropriate load resistor (if measuring current). After voltage passes through the Current Sense Modules (CSM), the voltage travels to the Multi-Target Connector where said voltage is further analyzed by the Side Channel Analysis(SCA) Shunt before it goes to the device under test (DUT).

3.7 Important Individual Modules

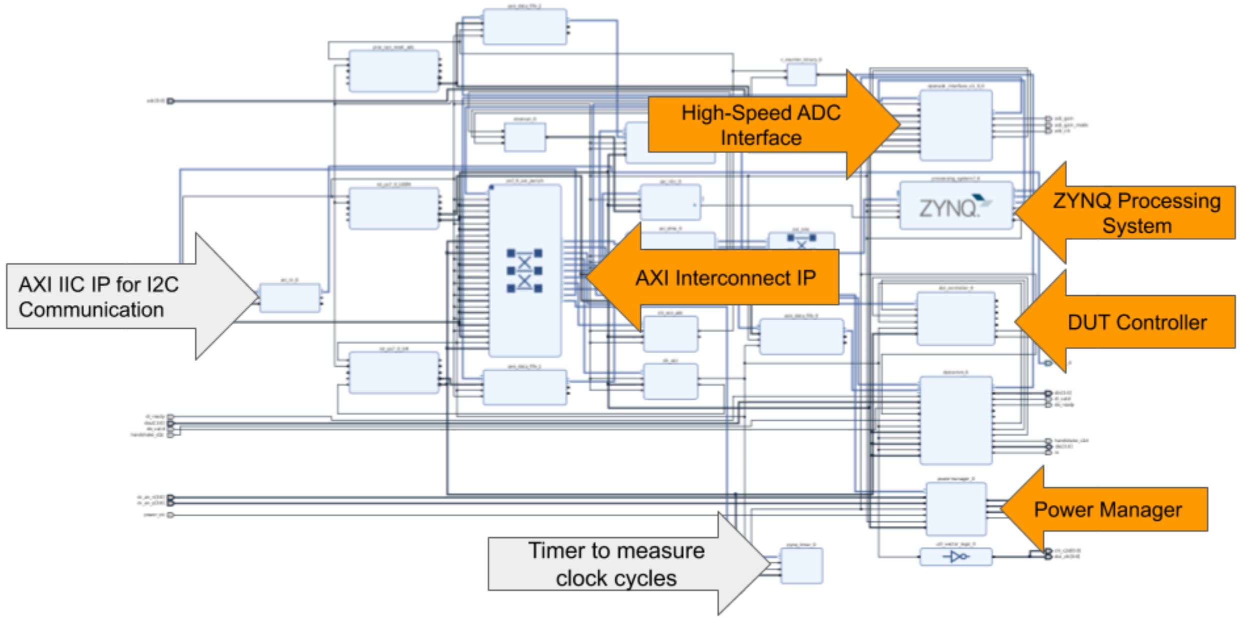


Figure 3.14: Main Components of PYNQ Controller

Figure 2 shows the essential components of the PYNQ controller. The components pointed to by the gray arrows are components we made or integrated ourselves. The components pointed to by the orange arrows were components already part of the existing design. These components will be explained in the following subsections.

3.7.1 AXI Interconnect IP

The AXI Interconnect IP is where all the IP we used are mapped to. It is responsible for coordinating the exchange between memory-mapped master devices and memory-mapped slave devices (the other modules in the block design).

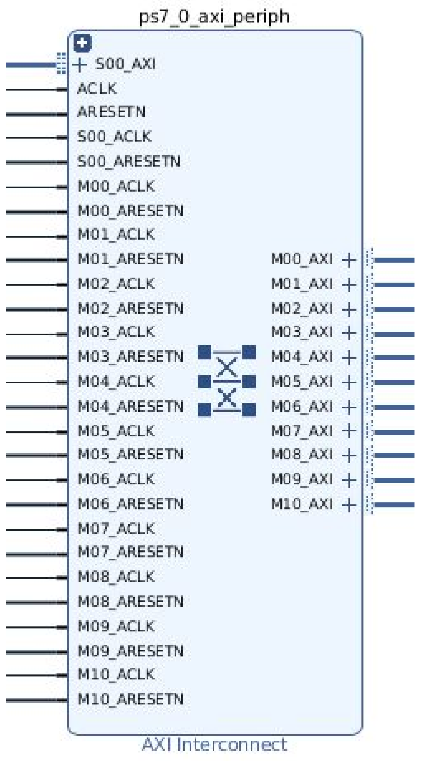


Figure 3.15: AXI Interconnect IP

3.7.2 AXI IIC IP

The AXI IIC IP is the vendor made IP that allows I2C communication to occur on the PYNQ board. Its wire connections were made using the autoconnect feature in Vivado. The important inputs to the module are the connections to the AXI Interconnect IP. The output of the AXI IIC IP is a signal bus going to the IIC IO buffers. The buffers map the internal signals to two output signals the SCL signal and the SDA signal. These two signals are then mapped to the PMOD in the constraint file.

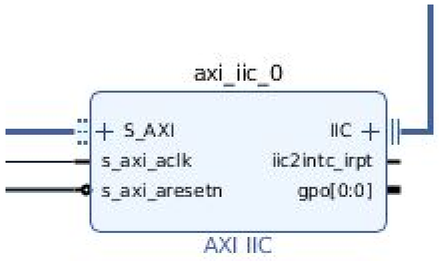


Figure 3.16: AXI IIC IP

3.7.3 DUT Controller

The DUT controller is responsible for coordinating the interaction between the PYNQ controller and the Device Under Test (DUT). This module outputs a trigger signal whenever the DUT is undergoing a cryptographic operation. The trigger signal is sent to the PYNQ timer and the Power Manager.

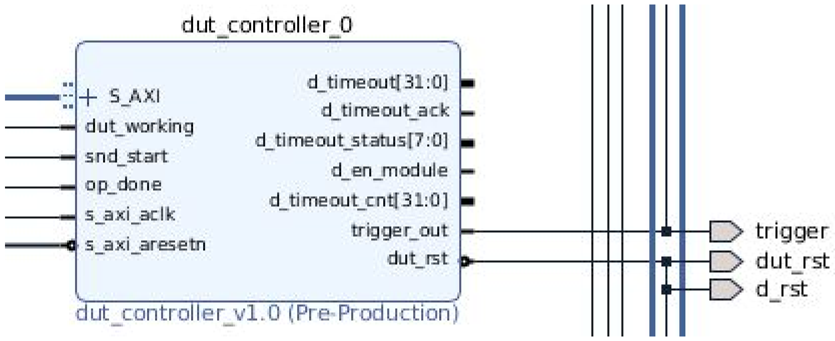


Figure 3.17: DUT Controller

3.7.4 Timer Module and Power Manager

The Pynq Timer module was created and implemented by our group. The timer counts the number of PYNQ clock cycles it takes for a DUT to finish conducting a cryptographic operation. It starts counting when the trigger signal coming from the DUT controller is high and continues to count until the trigger signal becomes low.

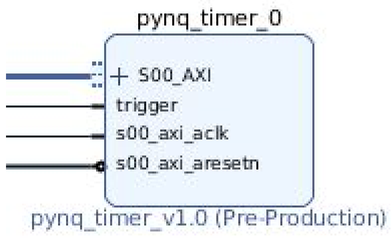


Figure 3.18: PYNQ Timer IP

The Power Manager was part of the already existing design. It uses the same trigger as the PYNQ timer. The Power Manager is responsible for setting the voltage as well as measuring the current and voltage using an internal XADC module.

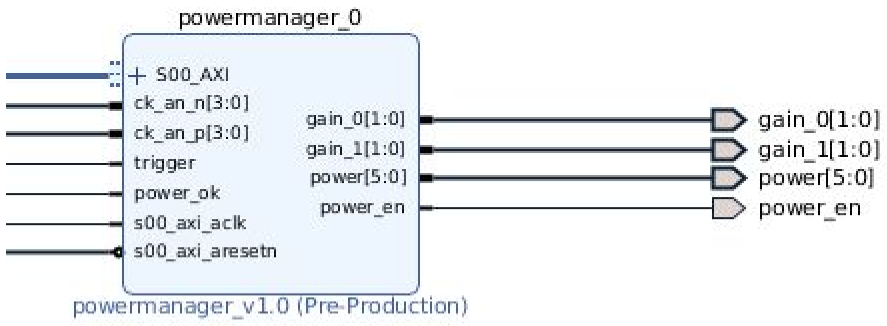


Figure 3.19: Power Manager

3.7.5 ZYNQ Processing System

The ZYNQ processing system (PS) contains dual core ARM processors as well as fixed peripherals and memory interfaces. The PS also hosts Linux, Python , and Jupyter notebooks to control the board from a software perspective.

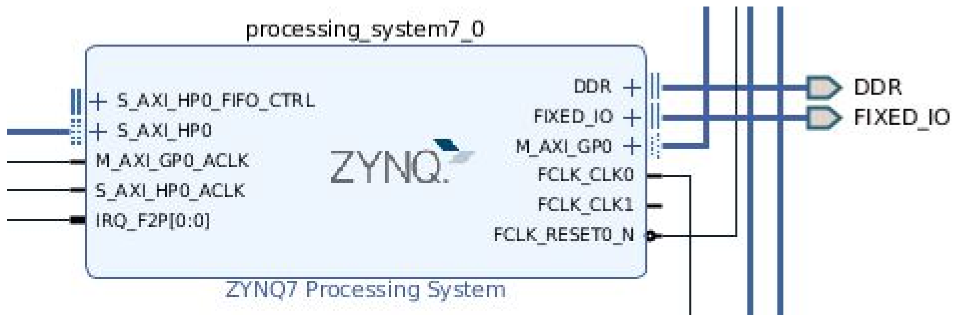


Figure 3.20: ZYNQ Processing System

3.7.6 FOBOS High Speed ADC Interface

This module on the PYNQ board is meant to serve as an interface for the digital power measurement that would be used for side-channel analysis. This measurement is originally gathered from the Multi-target connector board using the shunt resistor shown in the Level 2 diagram of the MTC. The high speed ADC on the FOBOS shield board collects this measurement, and transmits it to this module on the PYNQ.

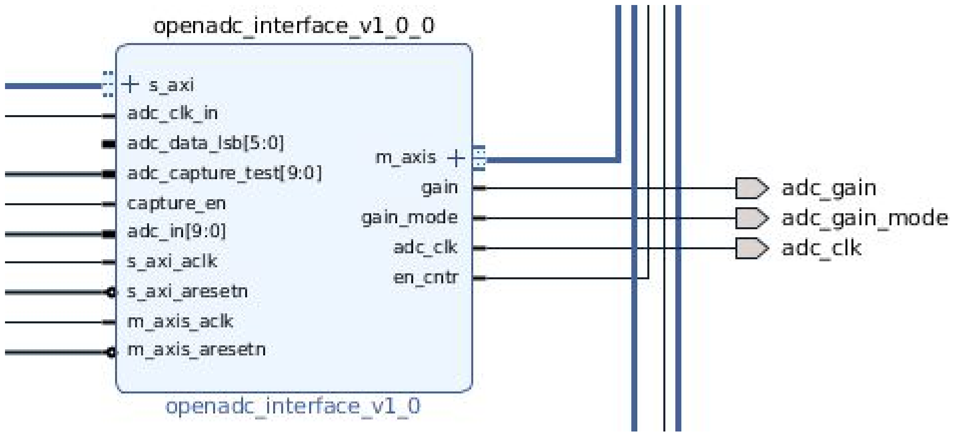


Figure 3.21: OpenADC Interface

4. Experimentation

4.1 I2C Hardware

The first step in I2C communication testing was to see whether or not the AXI IIC IP was working properly. To do this we set up a debug core in Vivado that captured the signals of the AXI IIC IP using the internal logic analyzer.

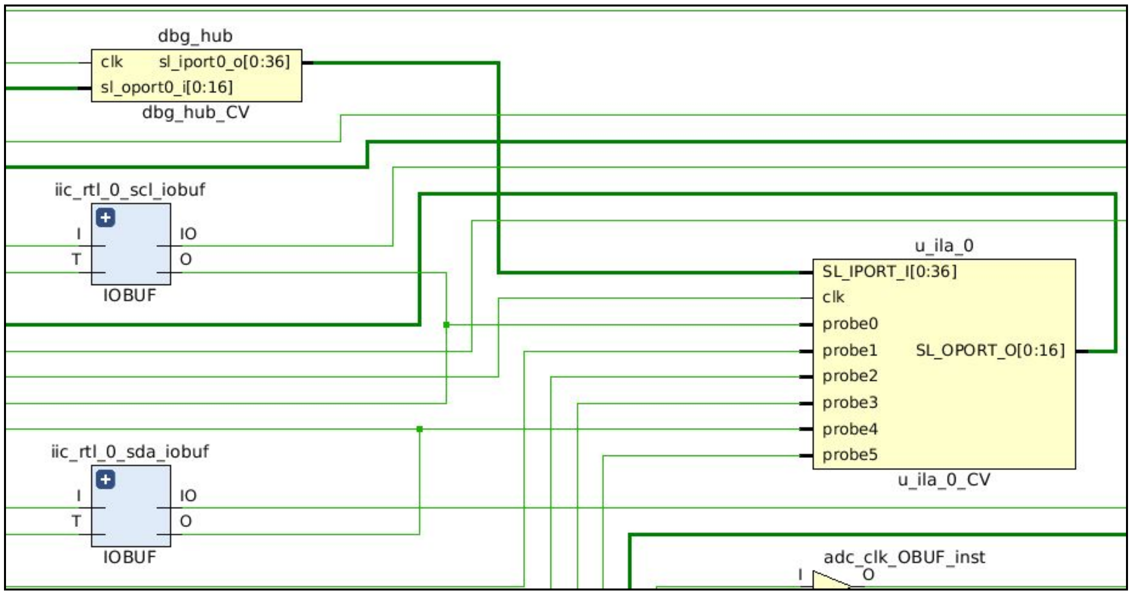
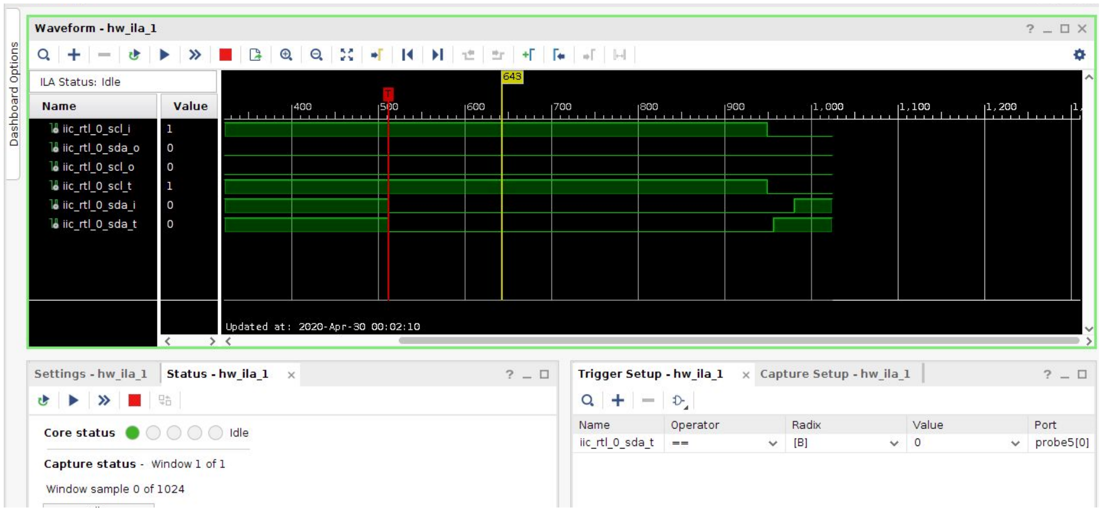


Figure 4.1: Synthesized design showing the debug core and probes

We set the trigger so that whenever the sda_t signal became 0, the output would be captured. So after running the IIC send function on the Jupyter notebook, we got this waveform.



Waveform - hw_ila_1

ILA Status: Idle

Name	Value
iic_rtl_0_scl_i	1
iic_rtl_0_sda_o	0
iic_rtl_0_scl_o	0
iic_rtl_0_scl_t	1
iic_rtl_0_sda_i	0
iic_rtl_0_sda_t	0

Updated at: 2020-Apr-30 00:02:10

Settings - hw_ila_1 Status - hw_ila_1

Core status Idle

Capture status - Window 1 of 1

Window sample 0 of 1024

Trigger Setup - hw_ila_1 Capture Setup - hw_ila_1

Name	Operator	Radix	Value	Port
iic_rtl_0_sda_t	==	[B]	0	probe5[0]

Figure 4.2: Waveform of Internal Logic Analyzer

4.2 I2C Software

In order to test for successful I2C communication on the PYNQ and microcontrollers, we must first verify that the microcontroller board itself can send and receive data. This was tested using two MSP430s, one as a master, and the other as a slave, and observed the I2C data using a logic analyzer. We made the master maintain a counter which transmitted data to the slave at every 0.5 second. The slave would then display the value of the counter it receives on its LCD display. In Figure 4.3, the logic analyzer showed that data A3 was transmitted to address 45.

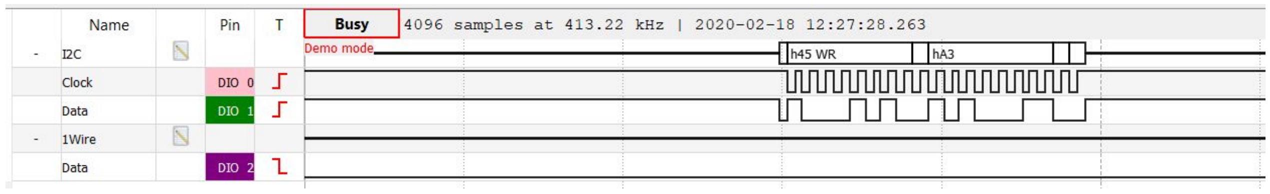
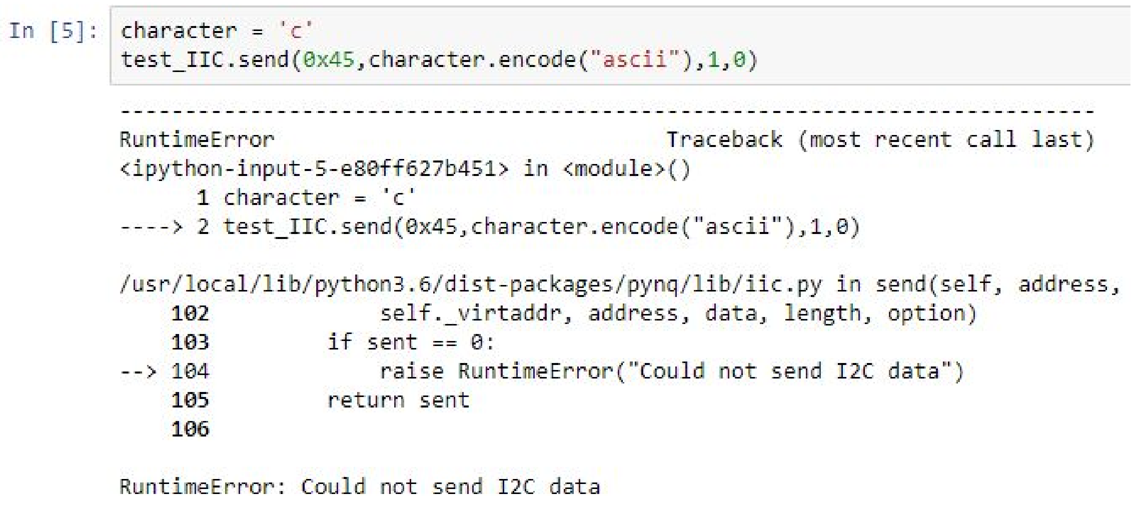


Figure 4.3 : Waveforms of I2C communication between two MSP430

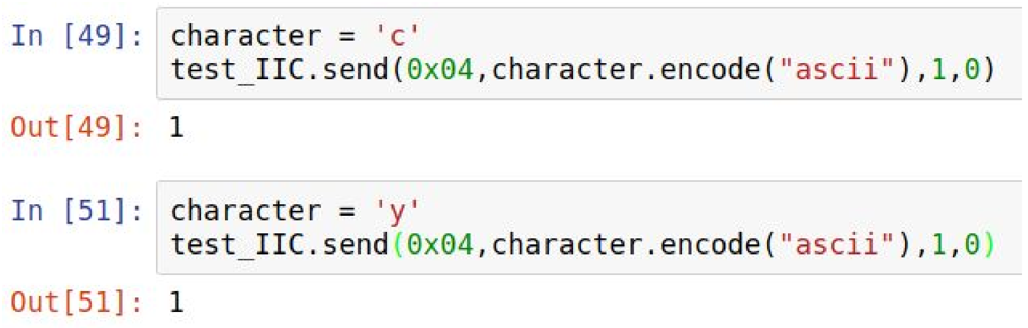
Next, we would need to verify the communication between the PYNQ board and a microcontroller. We used the same MSP430 connected in I2C with the PYNQ where the PYNQ is the master and the MSP430 is the slave. At first, we used the built AXI IIC libraries send function from the PYNQ overlay to send data. The function kept giving us a RunTimeError: I2C could not send data in Jupyter Notebook(Figure 4.4). We then tried to use the Standard controller logic flow, and the dynamic controller logic flow but we still received the same error as the send function.



```
In [5]: character = 'c'
        test_IIC.send(0x45,character.encode("ascii"),1,0)

        ---------------------------------------------------------------------------
        RuntimeError                              Traceback (most recent call last)
        <ipython-input-5-e80ff627b451> in <module>()
              1 character = 'c'
        ----> 2 test_IIC.send(0x45,character.encode("ascii"),1,0)

        /usr/local/lib/python3.6/dist-packages/pynq/lib/iic.py in send(self, address,
            102                 self._virtaddr, address, data, length, option)
            103             if sent == 0:
        --> 104                 raise RuntimeError("Could not send I2C data")
            105             return sent
            106

        RuntimeError: Could not send I2C data
```

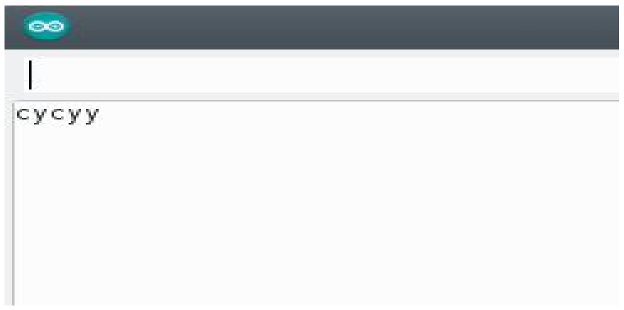
Figure 4.4 : Unsuccessful I2C between MSP430 and PYNQin Jupyter Notebook

With no progress, we decided to test I2C communication between arduino and a new PYNQ board. This was because Arduino has a built-in interface for sending and receiving data so it is easier to use and test if the PYNQ board can transmit data via I2C. We first tested I2C communication on Arduino Uno using PYNQ overlay's predefined functions from AXI IIC libraries. This was proven successful since there was no error present in Jupyter Notebook showed Figure 4.5 when we ran the send function and the received data appeared on the Arduino serial monitor which is shown in Figure 4.6.



```
In [49]: character = 'c'
         test_IIC.send(0x04,character.encode("ascii"),1,0)

Out[49]: 1

In [51]: character = 'y'
         test_IIC.send(0x04,character.encode("ascii"),1,0)

Out[51]: 1
```

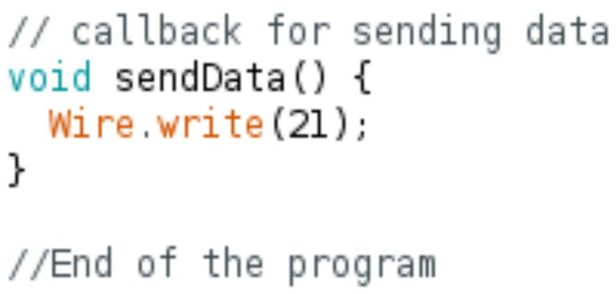
Figure 4.5 : Successful I2C between MSP430 and Arduino Uno in Jupyter Notebook



```
cycyy
```

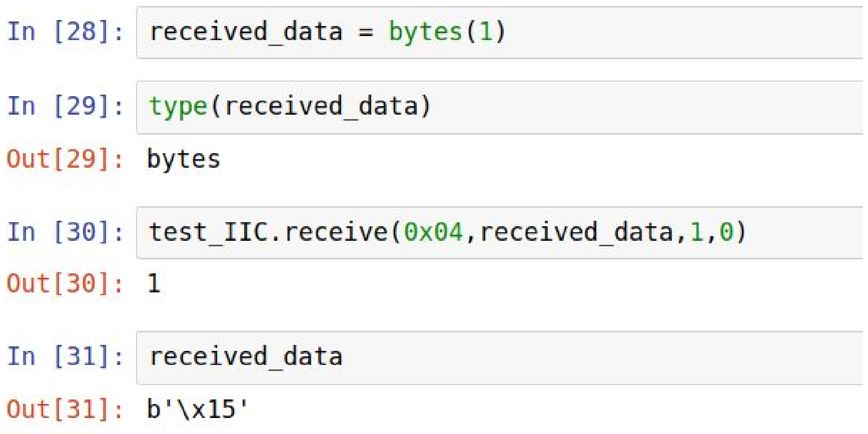
Figure 4.6 : Arduino Serial Monitor

We also tested if the PYNQ board was able to receive data from the Arduino and this was proven successful as well. In Figure 4.7 showed the Arduino Sketch writing 21 in decimal value and Figure 4.8 showed Jupyter Notebook on PYNQ receiving 15 in hex.



```
// callback for sending data
void sendData() {
  Wire.write(21);
}

//End of the program
```

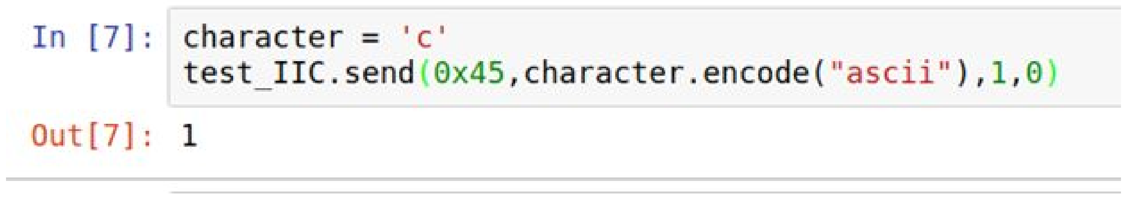
Figure 4.7 : Arduino Sketch



```
In [28]: received_data = bytes(1)

In [29]: type(received_data)
Out[29]: bytes

In [30]: test_IIC.receive(0x04,received_data,1,0)
Out[30]: 1

In [31]: received_data
Out[31]: b'\x15'
```

Figure 4.8 : Jupyter Notebook on Pynq

By testing on the Arduino Uno, we now know that the PYNQ board is capable of transmitting and receiving data. The PYNQ board we used previously with the MSP430 was defective and therefore, the I2C communication could not function properly. We switched the old PYNQ board into a new one and the I2C communication between PYNQ and MSP430 was successful via AXI IIC send function. Figure 4.9 showed that in Jupyter Notebook where there was no error present. This could further be verified using a logic analyzer where in Figure 4.10 showed the address is in h45, and the data being transmitted is h63. Figure 4.11 shows the MSP430 LCD displaying “99” in decimal.



```
In [7]: character = 'c'
        test_IIC.send(0x45,character.encode("ascii"),1,0)

Out[7]: 1
```

Figure 4.9 :Successful I2C between MSP430 and PYNQ in Jupyter Notebook

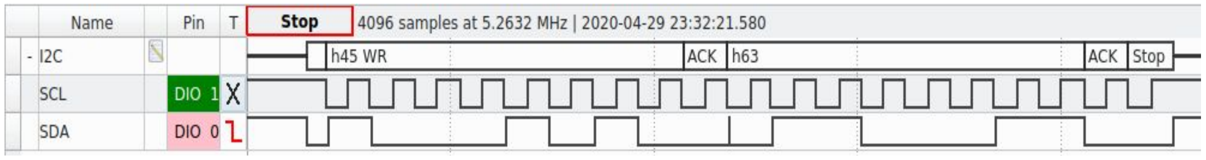


Figure 4.10 :Successful I2C between MSP430 and PYNQ in Logic Analyzer

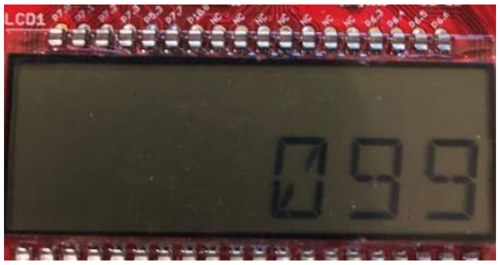


Figure 4.11 :MSP430 screen

4.3 Observation of the XXBX Protocol

The existing XXBX system uses a specific protocol over I2C to communicate to the DUT. Knowledge of this protocol is important in understanding the "command-level" communication between the boards in the system. In other words, how one component is telling another component what to do. This protocol was observed by installing the XXBX system on a Linux machine and running it with a cryptographic algorithm supplied from the SUPERCOP algorithm package. While the system was running, we used GTKterm, a serial line terminal emulator, to capture shell logs of the communication between XBS to XBH to XBD into a text file. This log file was then used to document the XXBX protocol in a communications diagram. A portion of this diagram, as well as the corresponding section in the log file that was used to draw it.

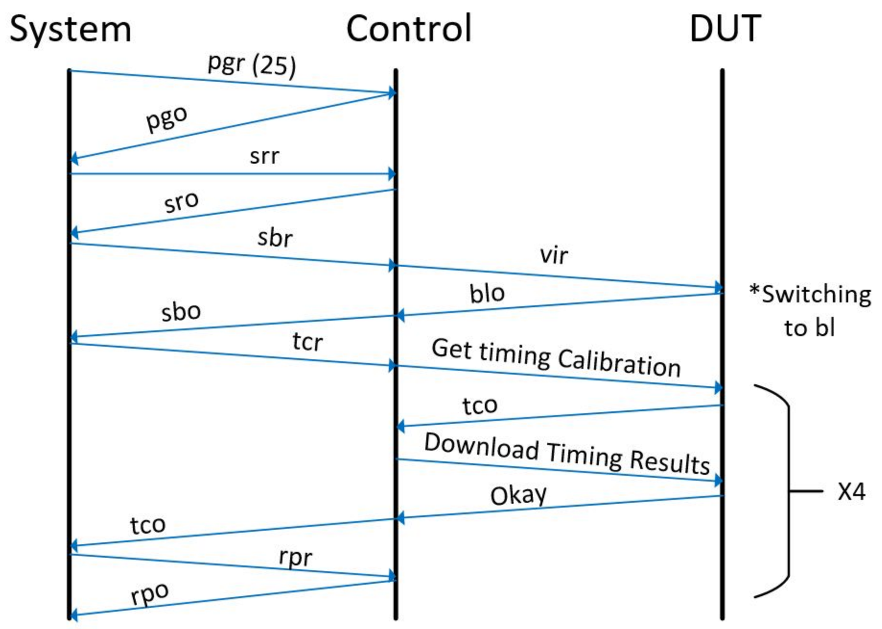
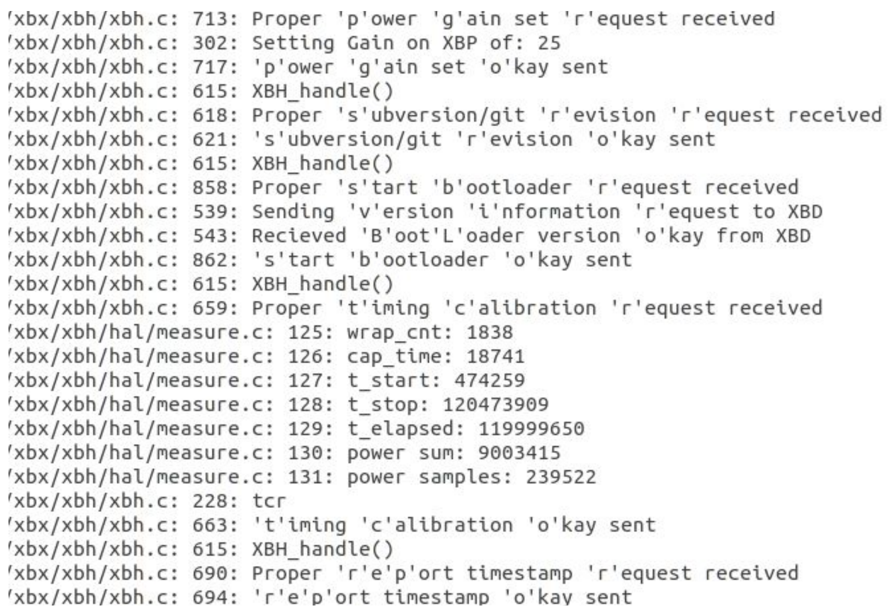


Figure 4.12



```
'xbx/xbh/xbh.c: 713: Proper 'p'ower 'g'ain set 'r'equest received
'xbx/xbh/xbh.c: 302: Setting Gain on XBP of: 25
'xbx/xbh/xbh.c: 717: 'p'ower 'g'ain set 'o'kay sent
'xbx/xbh/xbh.c: 615: XBH_handle()
'xbx/xbh/xbh.c: 618: Proper 's'ubversion/git 'r'evision 'r'equest received
'xbx/xbh/xbh.c: 621: 's'ubversion/git 'r'evision 'o'kay sent
'xbx/xbh/xbh.c: 615: XBH_handle()
'xbx/xbh/xbh.c: 858: Proper 's'tart 'b'ootloader 'r'equest received
'xbx/xbh/xbh.c: 539: Sending 'v'ersion 'i'nformation 'r'equest to XBD
'xbx/xbh/xbh.c: 543: Recieved 'B'oot'L'oader version 'o'kay from XBD
'xbx/xbh/xbh.c: 862: 's'tart 'b'ootloader 'o'kay sent
'xbx/xbh/xbh.c: 615: XBH_handle()
'xbx/xbh/xbh.c: 659: Proper 't'iming 'c'alibration 'r'equest received
'xbx/xbh/hal/measure.c: 125: wrap_cnt: 1838
'xbx/xbh/hal/measure.c: 126: cap_time: 18741
'xbx/xbh/hal/measure.c: 127: t_start: 474259
'xbx/xbh/hal/measure.c: 128: t_stop: 120473909
'xbx/xbh/hal/measure.c: 129: t_elapsed: 119999650
'xbx/xbh/hal/measure.c: 130: power sum: 9003415
'xbx/xbh/hal/measure.c: 131: power samples: 239522
'xbx/xbh/xbh.c: 228: tcr
'xbx/xbh/xbh.c: 663: 't'iming 'c'alibration 'o'kay sent
'xbx/xbh/xbh.c: 615: XBH_handle()
'xbx/xbh/xbh.c: 690: Proper 'r'e'p'ort timestamp 'r'equest received
'xbx/xbh/xbh.c: 694: 'r'e'p'ort timestamp 'o'kay sent
```

Figure 4.13

4.4 Designed Timer Unit

The need for this unit arises from the runtime measurement requirement for this system. Runtime, measured in clock cycles per byte encrypted, refers to the amount of time the DUT is performing cryptographic operations. This measurement cannot be accurately obtained or represented by directly using the clock on the DUT, therefore, it was necessary to create an IP module on the PYNQ board to perform the task counting clock cycles.

This module was designed using VHDL code in the Xilinx Vivado design suite since that environment is compatible with the PYNQ board. The functionality of the timer is as follows: an external trigger signal acts as an enable signal of the timer. When this signal is in active mode (implying that the DUT is running crypto) the counter process in the timer should increment at each clock cycle of the PYNQ board (every 10 ns because the PYNQ runs at 100 MHz frequency). If the trigger goes to its inactive value, the timer will stall until it activates again. A clear signal is also present to be able to reset the timer back to zero when necessary. Additionally, an overflow flag was included to indicate if the timing calculations from the timer are no longer reliable/have gone over the specified limit.

In order to test this unit, we designed a testbench for the timer also using VHDL code. This allowed us to generate a behavioral simulation to verify if the timer was designed properly. This simulation is shown below.

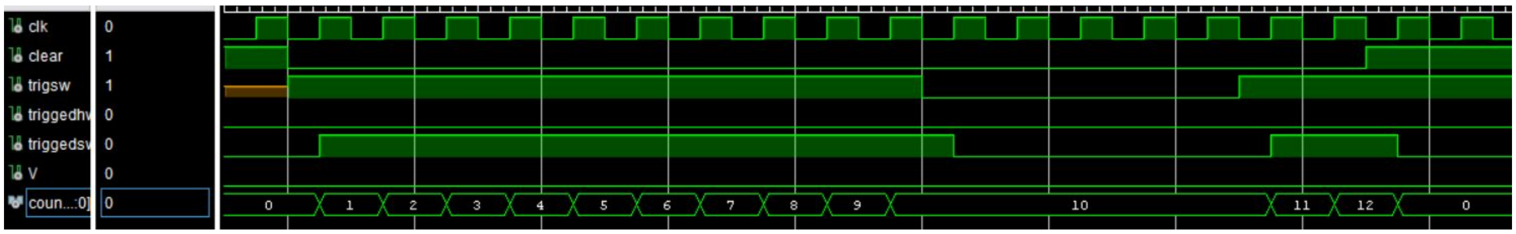
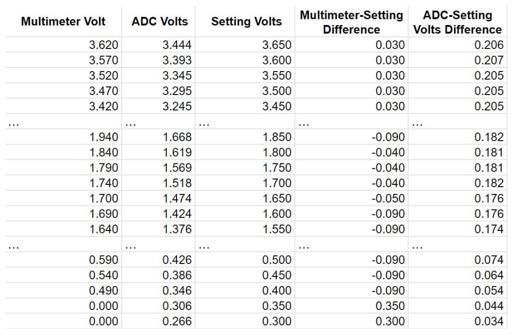


Figure 4.14

4.5 Power Consumption

First we analyzed variable voltage sweep of values set in a Jupyter notebook cell of ADC measurements. Then, in an open source spreadsheet application, found trendlines of best fit and calibrated ADC data curve closer to set variable data. Next, one creates the cells necessary in Jupyter notebook to properly store the ADC data in a way that can be analyzed for linear or polynomial regression. Then, one adds cells that help to visualize and graph the error margin and comparisons between calibrated and non calibrated data.



Multimeter Volt	ADC Volts	Setting Volts	Multimeter-Setting Difference	ADC-Setting Volts Difference
3.620	3.444	3.650	0.030	0.206
3.570	3.393	3.600	0.030	0.207
3.520	3.345	3.550	0.030	0.205
3.470	3.295	3.500	0.030	0.205
3.420	3.245	3.450	0.030	0.205
...	...	...	...	...
1.940	1.668	1.850	-0.090	0.182
1.840	1.619	1.800	-0.040	0.181
1.790	1.569	1.750	-0.040	0.181
1.740	1.518	1.700	-0.040	0.182
1.700	1.474	1.650	-0.050	0.176
1.690	1.424	1.600	-0.090	0.176
1.640	1.376	1.550	-0.090	0.174
...	...	...	...	...
0.590	0.426	0.500	-0.090	0.074
0.540	0.386	0.450	-0.090	0.064
0.490	0.346	0.400	-0.090	0.054
0.000	0.306	0.350	0.350	0.044
0.000	0.266	0.300	0.300	0.034

Figure 4.15: Spreadsheet Raw Data

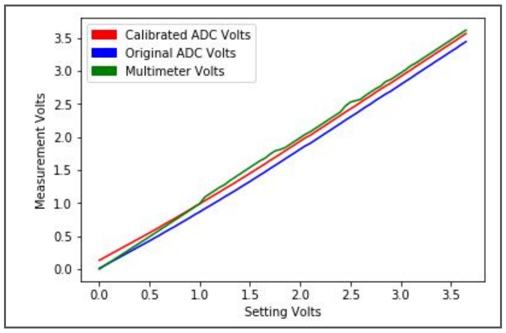


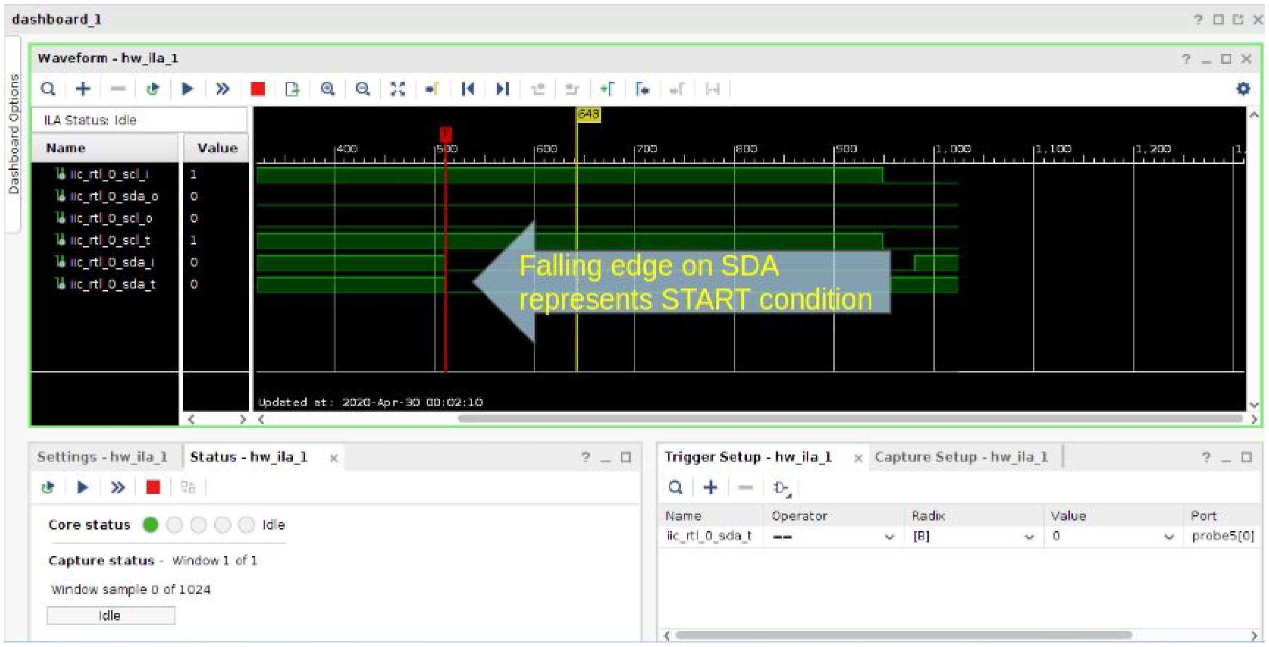
Figure 4.16: Pre and post Calibration Chart

For the multimeter data, one measured from the output pins of voltage and ground for the voltage. Afterwards, one recorded the results in a spreadsheet to be later transcribed in the proper data structure in a cell of a Jupyter notebook. Then, additional further analysis is performed in the Jupyter notebook for linear or polynomial regression with the multimeter data incorporated. For current measurements, an appropriate load resistor is created. This required soldering of smaller resistors in order for the load that can handle the current output from the FOBOS shield. Once completed, measurements are calculated, tabulated again in a spreadsheet and then transcribed in a cell in a jupyter notebook for calibration and visualization.

5. Experiment Validation

5.1 I2C Hardware

The test of the AXI IIC IP was successful because changes were observed in the waveform of the ILA. The falling edge transition on SDA, which represents a start condition, is the first step in the I2C protocol so this indicates a working IP.



ILA Status: Idle

Name	Value
iic_rtl_0_scl_i	1
iic_rtl_0_sda_o	0
iic_rtl_0_scl_o	0
iic_rtl_0_scl_t	1
iic_rtl_0_sda_i	0
iic_rtl_0_sda_t	0

Updated at: 2020-Apr-30 00:02:10

Settings - hw_ila_1 | Status - hw_ila_1

Core status Idle

Capture status - Window 1 of 1

Window sample 0 of 1024

Idle

Trigger Setup - hw_ila_1 | Capture Setup - hw_ila_1

Name	Operator	Radix	Value	Port
iic_rtl_0_sda_t	==	[B]	0	probe5[0]

Figure 5.1: Working operation of AXI IIC IP

5.2 I2C software

The purpose of I2C software was to test send and receive data using the PYNQ as a master, and the microcontroller as a slave. This test was partially successful since we were able to send data to the MSP430 and verified using the logic analyzer and the MSP430 on board LCD display . The data that we were sending was the character ‘c’ in ascii which is 63 in hex and 99 in decimal. This matched the results we received from the logic analyzer which showed the data being transmitted and received is h63 in Figure 4.10 while the MSP430 on board display showed 99 in decimal in Figure 4.11. However, we were not able to test if the PYNQ can receive information from the MSP430 due to the setback of using a defective PYNQ board. Although we were not able to test sending data from MSP430, we were able to send data from Arduino Uno and the PYNQ board received it. In Figure 4.7 showed the Arduino Sketch writing 21 in decimal value and Figure 4.8 showed Jupyter Notebook on PYNQ receiving 15 in hex which converts to 21 in decimal. So in theory, the MSP430 would also work when it is programmed correctly in Code Composer to send data after receiving data from the PYNQ.

5.3 Designed Timer Unit

After verifying the correct functionality of the timer unit individually, the next step was to integrate it into the block design of the PYNQ board. At first this involved learning how to import a packaged IP into Vivado. Upon successfully importing the package we could then select the timer from the IP library in the block design editor. After adding the module to the design, we noticed that the trigger port was missing. To fix this we needed to go back into the IP packager tool to review the vhd file and correct the syntax error. After repackaging the IP, the trigger port appeared and we were able to successfully synthesize and implement the new design.

5.4 Power Consumption

Calibration techniques were a success as margin of error between calibrated adc data and multimeter data is insignificant. This is contrasted with an error of up to .2 Volts with the original adc data. The slope used in the linear regression is 1.0239. The load resistor that was appropriate for this measurement experiment was approximately 51 ohms with a current tolerance of 150.6 mA. This was done by soldering three 160 ohms in parallel. This meets the requirement of properly ascertaining the voltage and by way of the resistance, the current as well which provides a power measurement of 112mW at most. Power consumption will decrease as voltage decreases throughout variable voltage sweeps.

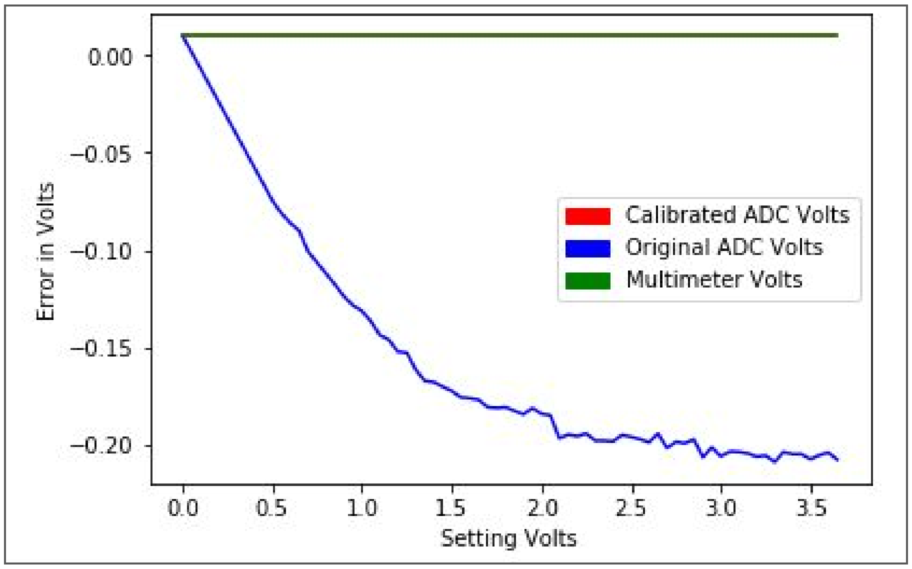


Figure 5.2: Margin of error between measurement methods.

6. Other Important Considerations (See below Points)

6.1 Reason for the Project and Benefits

The motivation surrounding this project is a contest currently being held by the National Institute for Standards and Technology (NIST). This contest is intended to standardize lightweight cryptographic algorithms for use in IoT devices, since many of these devices are not able to support current cryptographic measures in their design. Benchmarking the hardware performance on these algorithms is crucial in determining the security of systems. A system like the FOBOS (and XXBX) system is quite useful for performing this benchmarking and could benefit the cryptographic community in terms of research and supporting the development of secure systems.

6.2 Potential Use of the Project

As an educational tool, the FOBOS system could benefit future students who are interested in cryptography and side-channel analysis. The system could be used as a demonstration tool in a class or used as part of a side-channel analysis assignment. The FOBOS system is intended to be fully open source in order to provide a more suitable and affordable alternative to other benchmarking systems. As mentioned, this would be beneficial for universities, and potentially for smaller businesses engaging in lightweight cryptographic research.

6.3 Cost figures for the project

There was fortunately no money spent on this project, as all the necessary components were provided to us by our faculty supervisor. Xilinx also donated to our faculty two additional new PYNQ boards, which we were able to use and test on. The combined time spent on this project was roughly 200 hours. If our faculty supervisor did not provide us with the necessary components, it would cost around $500-$600. This includes the PYNQ-Z1 board, FOBOS shield and Multi-target connector boards, and microcontroller target device.

6.4 Alternatives to the Implemented Design

The FOBOS is an on-going research project which is always adapting and making the system run more efficiently. Currently, the voltage regulator on the FOBOS shield gets excessively hot when it is processing. In order to improve and regulate the heating issue, one could rewire the circuitry for better wires management or change the size of the area of copper on the chip to help dissipate the extra heat. Another improvement on the board would be to have better pins placement for easier access when measuring power consumption. However, in order to make these pins more accessible, the FOBOS shield would need to be physically bigger and that could be more costly during fabrication.

6.5 Maintainability/maintenance of the final design solution

Electronics will be more prone to breaking after years of usage. FOBOS are built in a modular fashion, and therefore, if the board itself is defective or no longer works, it can be easily replaced for a new FPGA board. FOBOS is also open-source so anyone can use the software and they will also be able to update the software if we have new software that enables FOBOS to run more efficiently and effectively. .

6.6 Retirement/Disposal at End of Project Lifetime

Once FOBOS hardware reaches the end of its lifetime, it can be safely discarded at any municipal electronic waste center.

7. Administrative Part

7.1 Project Progress and Completion

A significant portion of this project was setting up the I2C software and hardware. After many hours of debugging and testing, we were able to confirm that the PYNQ board could successfully send data to a microcontroller via I2C. This, along with completing the other tasks mentioned in the Experimentation section, served as a major point of progress for our project. However, unfortunate circumstances in the form of internal team issues between 492/493 semesters, defective hardware (one PYNQ board was not working correctly), and the overall pandemic situation this Spring did not allow for the full implementation and integration of the system for final analysis. For this reason, the Experimentation outlines the testing and results of the individual modules that were completed for this project. While the project could not be finished as planned, it was still possible to obtain important intermediate results that will serve as a foundation for successful implementation of the FOBOS system in the future.

7.2 Changes to the design, task, or schedule

Due to the current situation, our project can no longer proceed as originally planned. We have consulted with our faculty supervisor regarding the status of the project, as well as scale down modification plans moving forward. We have decided to drop the goal of finding measurements for ROM, RAM, runtime, power consumption, and side channel information leakage on the designed system. Finding these measurements and confirming their validity/correctness would require the system to be fully integrated, as well as extensive testing that would not be possible within the current time constraints. By the end of this semester, the goal is to have the individual components of the system working properly. This includes the Power measurement unit, I2C communication, and timing unit required for measurements.

Originally, we wanted the PYNQ board to be able to communicate with the Tiva LaunchPad and use the pre-existing XXBX system to evaluate the behavior of the XXBX protocol on the new FOBOS system. However, due to not knowing the PYNQ board was defective, the I2C communication took longer than needed even though we had the correct codes. Therefore, we scaled down our goal to ensure that I2C communication was successful between the PYNQ and MSP430.

Our project does require people to work together on one system and it was quite difficult during Covid-19 since we could not meet each other and have the system set up in one location. In order to comply with not meeting each other face to face, we had to make additional trips back and forth from each others' house to drop off required components needed for testing into the mailbox. This was something that was hard since our tasks are intertwined with each other. If one person cannot finish his/ her tasks, then the person whose tasks can only be performed after will be affected and put on hold until the other person finishes his/ her task.

7.3 Funds spent

The fund we spent on this project was $0. This was due to the nature of the project being built upon two existing platforms. Our faculty supervisor was able to lend us the required components such as the analog discovery, FOBOS shield, MTC, and the XXBX boards. Xilinx also provided us with additional PYNQ-Z1 and its accessories kit for research purposes. The microcontrollers that were used were a requirement from a previous class and the Arduino Uno was something one of our team members already had for personal use.

7.4 Man Hours

- **I2C Software: 60 hours**
  - The majority of the hours were spent on debugging and testing whether or not the I2C was successful between the PYNQ and the DUT. Even though our codes were right, we were working with a defective PYNQ board which made debugging and testing more extraneous than it was supposed to be. If we were working with a functional PYNQ board from the beginning, the hours would have been cut down to 40 hours in software.
- **I2C & PYNQ Hardware: 20 hours**
  - Half of the time was spent integrating the AXI IIC IP into the block design and figuring out how to map the IIC SCL and SDA lines to the PMOD ports of the PYNQ. The other half of the time was spent creating the PYNQ Timer IP and

integrating it into the existing block diagram.

- ❖ **XXBX: 30 hours**
  - ➢ Most of the hours were spent setting up the XXBX platform on a Linux system. The rest of the time went to documenting and observing how the timing protocols worked between boards.
- ❖ **Power Measurement: 55 hours**
  - ➢ Early portion of hours were based on the setup of equipment. Middle portion of hours were spent on calculations of raw ADC measurements and load resistor, and learning how to use Jupyter notebook. Last portion of hours was learning how to do and apply linear and polynomial regression with the data measured.
- ❖ **Non-technical work: 30 hours**
  - ➢ Non-technical work was spent on tasks such as design documentations, presentations, weekly meetings as well as PM responsibilities.

8. Lessons Learned and Team Experience

8.1 Lessons Learned

- ❖ Allocate more time for debugging and testing
- ❖ Having back-up plans for testing (Simulating I2C with Arduino, Testing Timer with button, etc.)
- ❖ The PYNQ board we were using was defective which affected our results when trying to communicate via I2C
- ❖ More consideration for potential setbacks (whether due to personal reasons, defective equipment, or other reasons) during time allocation
- ❖ Learned to work as a team and adapt to each other's schedule
- ❖ Be open to taking on a project that you might not know much about. It can be a good learning experience in terms of developing knowledge and skills.
- ❖ Designing IP cores in Vivado
- ❖ Experience with Jupyter Notebook environment

- Utilizing a logic analyzer
- Working with Linux OS on a Xilinx System on Chip

8.2 Team Experience

- Listening to each other, especially the project manager is a key component in successful communication.
- Know when to be a business professional even if we are friends outside of the project.
- Determining individual strengths and weaknesses and working according to that is crucial for efficacy and productivity.
- If team members are not on the same page in terms of expectations and requirements it can be problematic.
- It is important to be flexible when working with others in terms of scheduling meetings and partitioning of tasks
- Having practice time ahead of key presentations is beneficial for presentation cohesion
- Dressing professionally for presentations gives a good impression to the audience
- Overall, even though working in teams can be difficult, it provides new sets of skills that are essential for engineers heading into the industry.

9. References

[1] S. Bhunia and M. Tehranipoor, “Side Channel Attack,” *Side Channel Attack - an overview | ScienceDirectTopics.*[Online].Available: www.sciencedirect.com/topics/computer-science/side-channel-attack. [Accessed 11-Oct-2019].

[2] **https://cryptography.gmu.edu/fobos/**
https://github.com/GMUCERG/xbx/blob/master/docs/XXBX-UserGuide.pdf

[3]https://store.digilentinc.com/analog-discovery-2-100msps-usb-oscilloscope-logic-analyzer-and-variable-power-supply/

[4] https://store.digilentinc.com/pynq-z1-python-productivity-for-zynq-7000-arm-fpga-soc/

[5] Kaps, Jens Peter. “Flexible, Opensource WorkBench fOr Side-Channel Analysis ...” [Online]. Available: http://www.hostsymposium.org/host2018/hwdemo/HOST_2017_hwdemo_5.pdf. [Accessed 04-May-2020].

Appendix A: Proposal

ECE 492

Senior Design Project Proposal

Side Channel Analysis on Microcontrollers

Team Members:
Theresa Nguyen, Jorge Rodriguez, Vincent Guevara, Bradford Webb

Faculty Supervisor:
Dr. Jens-Peter Kaps

2. Executive Summary

Confidentiality of data as well as data authentication accomplished through cryptographic algorithms is a relevant topic with the growing usage of devices that are interconnected through wireless communication. However, running cryptographic algorithms on hardware and software can leak information, such as power consumption, that attackers can exploit in order to break the cryptosystem. This type of exploitation is known as a side-channel attack, and it is directed at the information gained as a result of the leakage, rather than the weakness of the particular algorithm

used [1]. This occurrence is not ideal for security because the leaked heat and voltage emissions may allow an attacker to access the encrypted key code via reverse engineering.

The proposed project will implement cryptographic algorithms on a system using two MSP430 microcontrollers, a FOBOS shield, and a PYNQ board. The PYNQ based side-channel analysis testbed will provide a platform to run Linux, and obtain power consumption measurements. The FOBOS shield board mainly performs side-channel analysis, as well as regulate power to the MPS430s via a voltage regulator chip. Lastly, the MPS430s will serve as the device-under-test (DUT) that determines the timing results of the system. Each of the boards will be interfaced using I2C serial protocol and configured to speak the XXBX protocol to try and benchmark RAM, ROM, runtime, power consumption, and power leakage from side-channels. These measurements will be the outputs obtained by the system as a cryptographic algorithm is implemented. Analysis of this information will allow us to run tests to determine whether the secret key is protected or not.

This project is intended to serve as an educational tool for students interested in the topic of cryptography. Due to the lack of readily available, complete open-source software packages for side-channel analysis that include drivers for testing on the board, it is difficult to educate students about side-channel analysis with a hands-on approach [2]. Additionally, this project may provide an easily accessible way for companies to analyze their devices for hardware vulnerabilities at a much more reasonable cost. Current solutions tend to be particularly expensive. So much so, that most retailers are not even posting their prices on their websites[3]. Meanwhile, FOBOS at most is a platform that costs just under $500 for the logic analyzer and control board that the FOBOS shield would go on top of in assembly[4][5].

3. Problem Statement

2.1 Motivation

The increase in embedded systems and the Internet of Things has led to a rethinking of how smaller systems utilize cryptographic algorithms. Since most cryptographic algorithms were initially invented to be implemented on large desktop and server systems, the algorithms' performance on smaller devices can be quite expensive in regards to cost, time, and CPU usage. In light of this, the National Institute of Standards and Technology (NIST) has initiated a request for lightweight cryptographic algorithms to be solicited, evaluated, and standardized for use in constrained environments. This project seeks to add to the research effort promoted by NIST.

2.2 Identification of Need

Side channel attacks are an increasing method in compromising valuable assets in today's war in cybersecurity. Additionally, solutions to this problem tend to be behind massive paywalls and/or have a steep learning curve to get past in order for IT teams with several companies to comprehend and implement successfully. Furthermore, companies are searching for new talent that have these cryptography skills, however, very few educational systems exist for students to learn proper side channel attack analysis. Our system uses FOBOS, (Flexible Open-source workBench fOr Side-channel analysis), to help achieve an educational and affordable solution through differential power analysis (DPA). Our team is augmenting this system by expanding the range of hardware devices that can be evaluated against the ciphers generated by the host FPGA(Field Programmable Gate Array) by evaluating two microcontrollers. This expansion is important because the more scenarios that can be evaluated, the more scenarios that can be verified against side channel attacks. The specific microcontroller used is the Texas Instruments' MSP430FR6989.

To further confirm the processing results, the FPGA used is a Xilinx PYNQ-Z1 evaluation board that possesses a Linux system on a chip that can serve a Jupyter Notebook on an IP that visualizes the DPA that was performed on the MSP 430. Incidentally, because the user just has to know Python to implement the notebook and any other instructions on the chip. This allows for the user to not only analyze the power differentials faster, but more clearly because it is less overstimulating than looking at massive numerical sets that do not fit on a standard monitor. FOBOS will solve the problem of having an affordable open source educational and commercial device that can evaluate side-channel attacks.

4. Approach

3.1 Intro

Data confidentiality is created through cryptographic algorithms where keys and other different cryptic information is formed. Another way of looking at cryptographic algorithms are just sophisticated mathematical formulas or encryption methods[6]. A specific type of encryption is a cipher[6]. When running cryptographic algorithms on hardware and software, the device leaks heat and voltages through its side-channels. These leaks can be measured by power consumption and electromagnetic radiation. This problem is most dangerous when a secret key is created and used. A key is essentially a solution to these encryption methods and formulas that are valuable as long as they are generated but not known[5]. The only exception is the public encryption key, but only because the decryption key is unknown.

In order to fight the leakage, we will be implementing cryptographic algorithms based on reference implementations in C on an MSP430 microcontroller. In order for this to be done, we are going to have to adjust a PYNQ based side-channel analysis testbed connected to the MSP430 controller by way of I2C as well as the XXBX protocol. By the end of the project, as many ciphers as possible will be run. In addition to running the ciphers , one will also be able to visually confirm the power consumption results through the graphs generated by the Jupyter notebook.

3.2 FOBOS

Before tackling the problem, one must first understand the component with which we use to evaluate how the attacker obtains the key from the data leakage. The aforementioned component is the Flexible Open-source workBench fOr Side-channel analysis (FOBOS). FOBOS is an open-source framework used to conduct side-channel analysis attacks on Field Programmable Gate Arrays (FPGA) which supports multiple FPGA boards. FOBOS is made in a way that it is easily adapted for newer FPGA boards, oscilloscopes, and a wide variety of techniques to attack. In addition, FOBOS includes all the software needed to run the most prominent of all side-channel analysis attacks, Differential Power Analysis attacks. FOBOS supports power analysis of FPGAs and includes the software to control the device when attacked. In the midst of attack, FOBOS is able to trigger an oscilloscope, as well as obtain and analyze the readings using a plethora of power analysis techniques.

In order to obtain the desired results, the group must be able to analyze the side-channel using Python, evaluate power consumption measurements using ADC in FPGA and OpenADC, obtain experience with cryptographic algorithms, and work with Linux on a Xilinx Zynq System on Chip. The reason why power consumption is being measured is because side-channel analysis is done using differential power analysis (DPA). Additionally, FOBOS relies on the power traces leaked from the channel and analyzed by a triggered oscilloscope to make a guess on the key given to it from the user.

3.3 Communication and Data Visualizations

To make sure that the microcontrollers are communicating properly with the Xilinx PYNQ and vice versa, the group must inspect and integrate the I2C libraries from Xilinx to the PYNQ board. After the PYNQ board is properly integrated with the I2C libraries, the I2C data needs to be confirmed to be operational from the MSP 430 microcontrollers. To ensure the software power analysis works properly, a spectrum analyzer will be used to confirm the power leaks are or are not happening from the side channel in a similar way that is represented in software. To be able to access the converted power information by the ADC, Python code is needed along with additional code that is used to produce a graph in a Jupyter notebook. The team is able to use the Juypter notebook because the Xilinx Pynq has a Linux OS on the Chip.

3.4 XXBX

What brings the algorithm input, communication, data visualization, and power analysis together is the XXBX protocol. First the algorithms are sent to be cross-compiled on software. Then, the power consumption data determined by the user is collected and sent to the targeted device. Meanwhile, a power shim unit takes measurements of the victim device for power consumption data which is then compared to what was submitted by the user. The victim device also is being measured on how much ROM, RAM, and runtime usage is occurring from the control device commands. Once the comparison of power values have been made , the data is sent back to the control device to determine if a key value can be estimated from the power values leaked from the side channel and measured from the oscilloscope and power shim. These values after being compared are also then visualized on software via FOBOS on the graph previously stated.

5. System Design

4.1 Functional Architecture/Decomposition

Level- 0 Diagram

Level-0 of functional decomposition illustrates a black box design of the high-level system inputs and outputs. In addition, it highlights the top-level functions performed by the system. Level-0 diagram can be seen below.

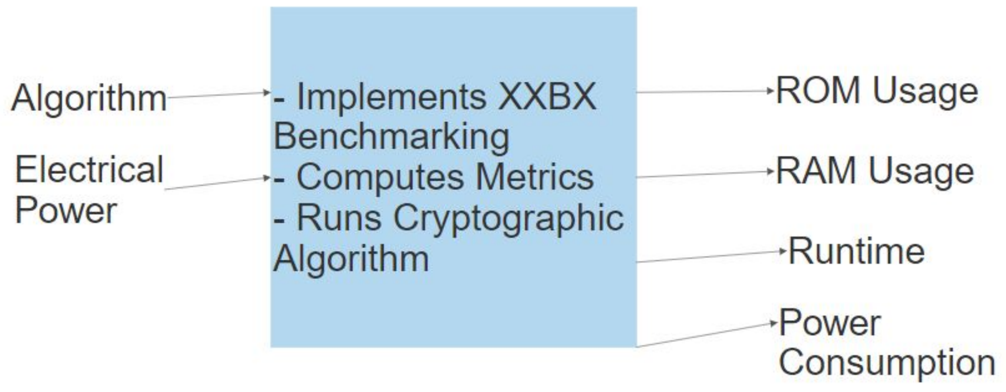


Figure 1: Black Box Design (Level 0)

Level- 1 Diagram

Level-1 of functional decomposition introduces the relationships between the top-level functions described in Level-0. While it is still not incredibly detailed, Level-1 provides an illustration of each individual component necessary to build the system, and how they are connected and communicate with each other.

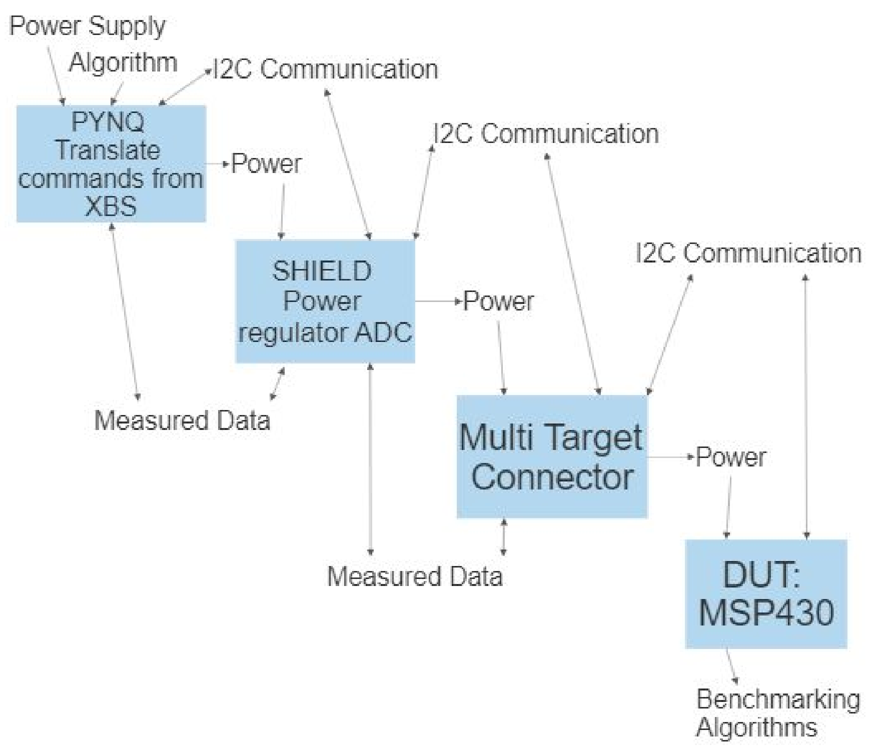


Figure 2: White Box Design (Level 1)

<u>Level-2 Diagrams</u>

Similar to Level-1, Level-2 functional decomposition shows the relationships/connections between the "second-level" functions. As a result, these diagrams tend to be more detailed in their descriptions. Below are the level-2 specific diagrams for each of the components displayed in the Level-1 decomposition.

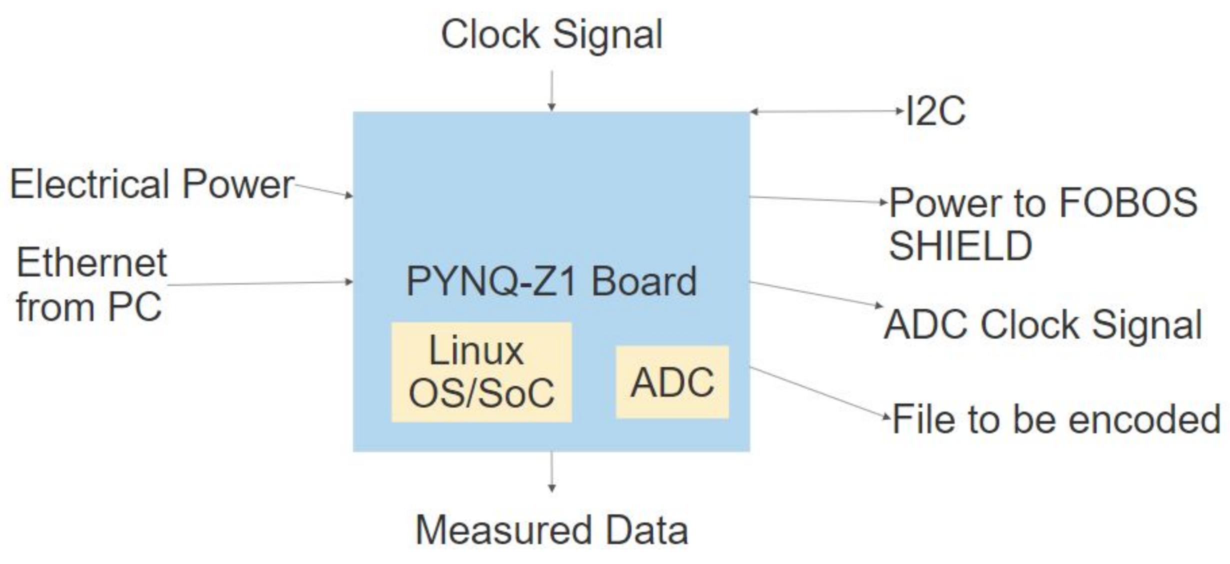


Figure 3: Level-2 Diagram for Pynq

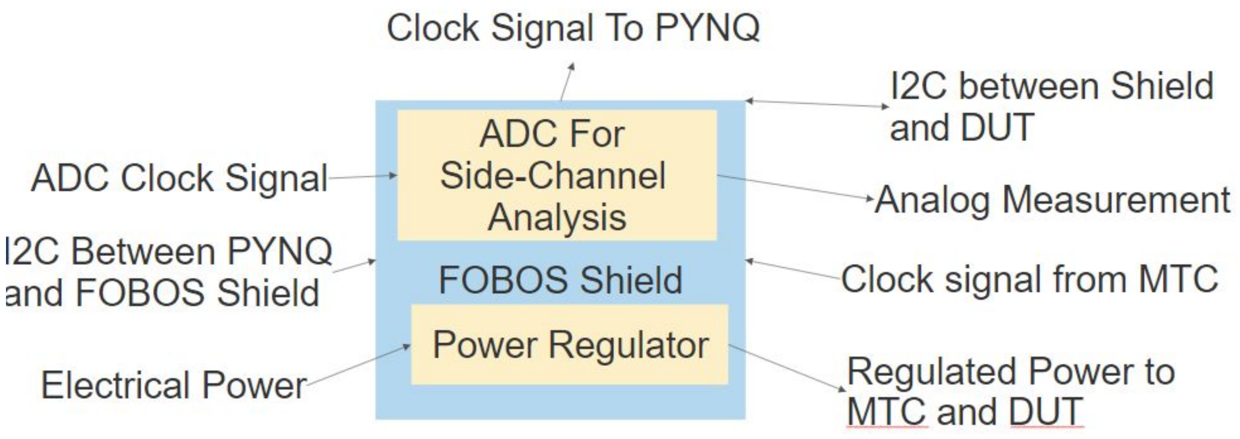

Clock Signal To PYNQ
I2C between Shield and DUT
ADC For Side-Channel Analysis
ADC Clock Signal
Analog Measurement
I2C Between PYNQ and FOBOS Shield
FOBOS Shield
Clock signal from MTC
Power Regulator
Electrical Power
Regulated Power to MTC and DUT

Figure 4: Level 2 Diagram for FOBOS Shield

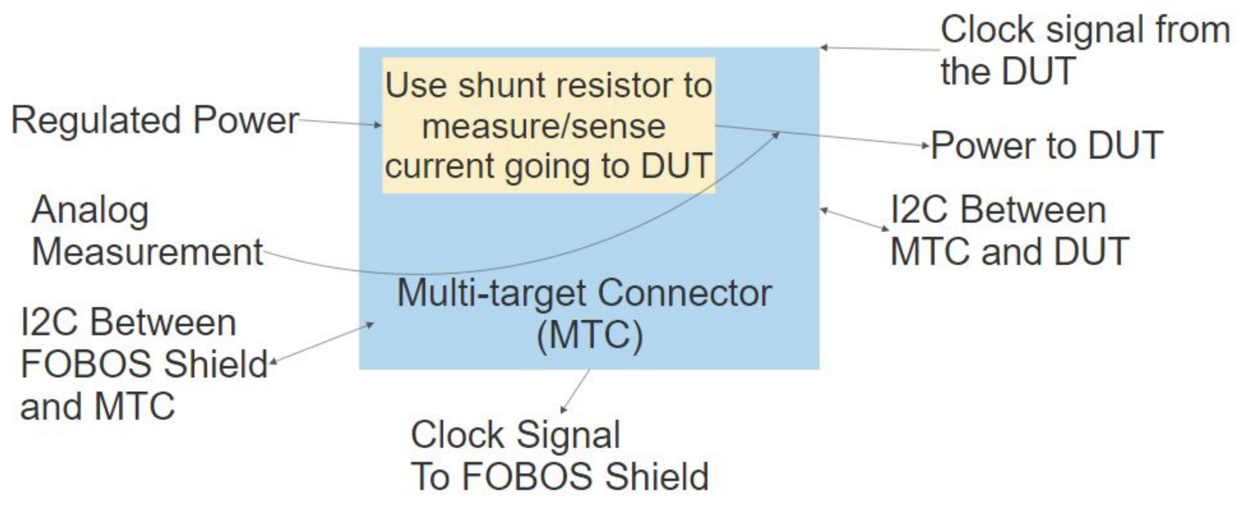

Clock signal from the DUT
Use shunt resistor to measure/sense current going to DUT
Regulated Power
Power to DUT
Analog Measurement
I2C Between MTC and DUT
Multi-target Connector (MTC)
I2C Between FOBOS Shield and MTC
Clock Signal To FOBOS Shield

Figure 5: Level 2 Diagram for Multi-target connector (MTC)

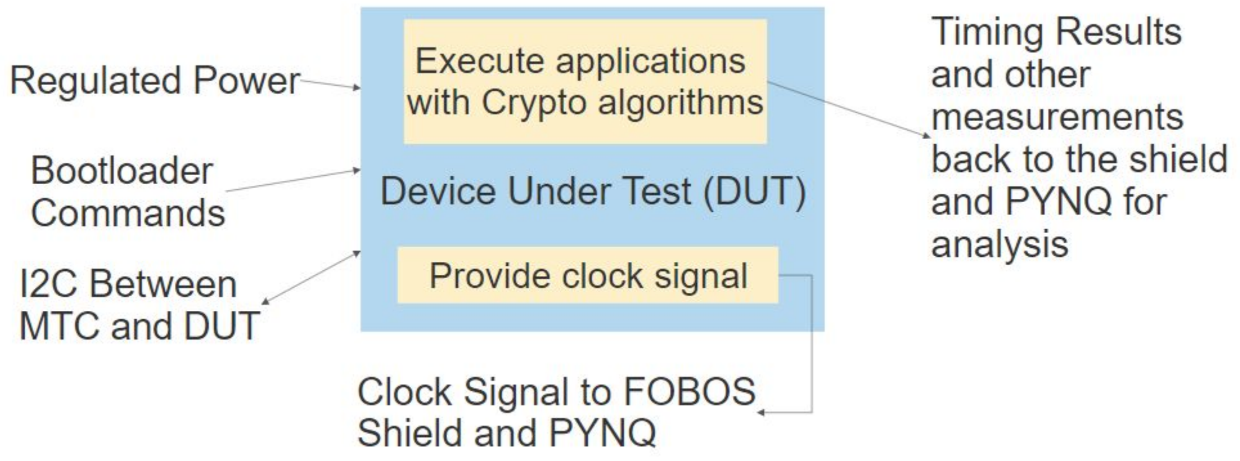

Execute applications with Crypto algorithms
Timing Results and other measurements back to the shield and PYNQ for analysis
Regulated Power
Bootloader Commands
Device Under Test (DUT)
I2C Between MTC and DUT
Provide clock signal
Clock Signal to FOBOS Shield and PYNQ

Figure 6: Level 2 Diagram for Device under test (DUT)

4.2 System Architecture

The system architecture comprises of the major hardware and software components in a hierarchical order.

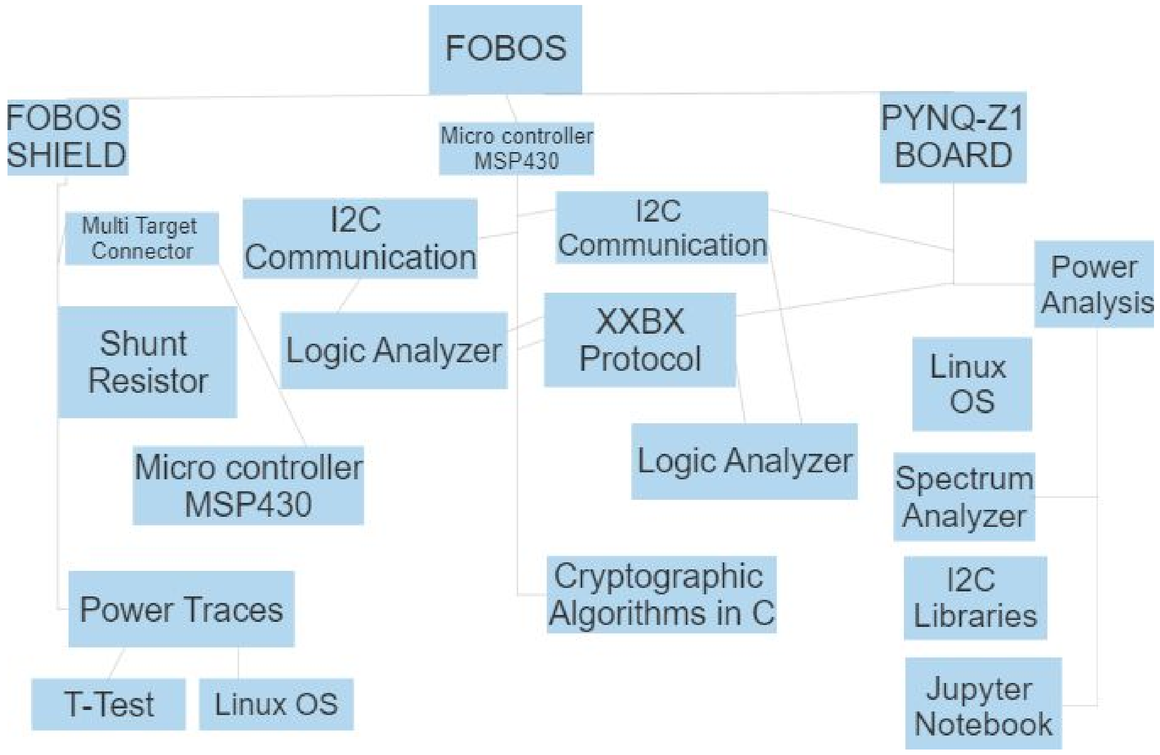


Figure 7: System Architecture

4.3 Preliminary Design

<u>XXBX Protocol</u>

The design of our project is based upon the XXBX system. XXBX stands for eXtended external Benchmarking eXtension, and it is a tool for benchmarking the performance, memory usage, and power/energy consumption of cryptographic software on microcontrollers. XXBX is an extension of the SUPERCOP that allows for benchmarking on embedded systems with additional metrics for measuring power consumption on top of ROM, and RAM usage.

The system (shown below) comprises of four main components: the XXBX Software (XBS), the XXBX Harness (XBH), the XXBX Power shim (XBP) , and the XXBX Device Under Test (XBD). The XBS is software that runs on a Linux PC. The XBS is responsible for compiling algorithm packages, uploading firmware to the XBH, and performing data collection and analysis. Communication between the XBS and XBH is via TCP/UDP. The XBH translates commands from the XBS to a form the XBD can understand. The XBH communicates to the XBD via I^{2}C. The XBP amplifies the power consumed by the XBD so that the XBH can measure it using its ADC. The XBH also needs to measure the execution time of XBD through a timer capture pin.

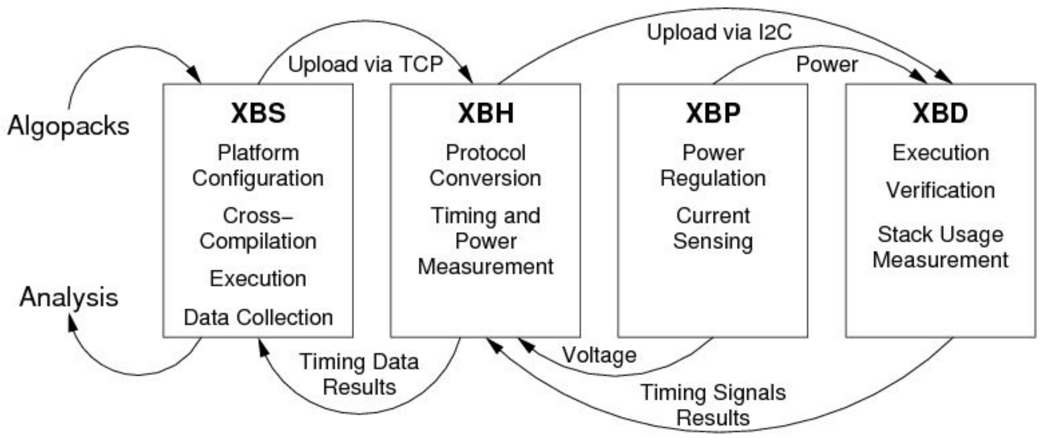


Figure 8: XXBX Framework

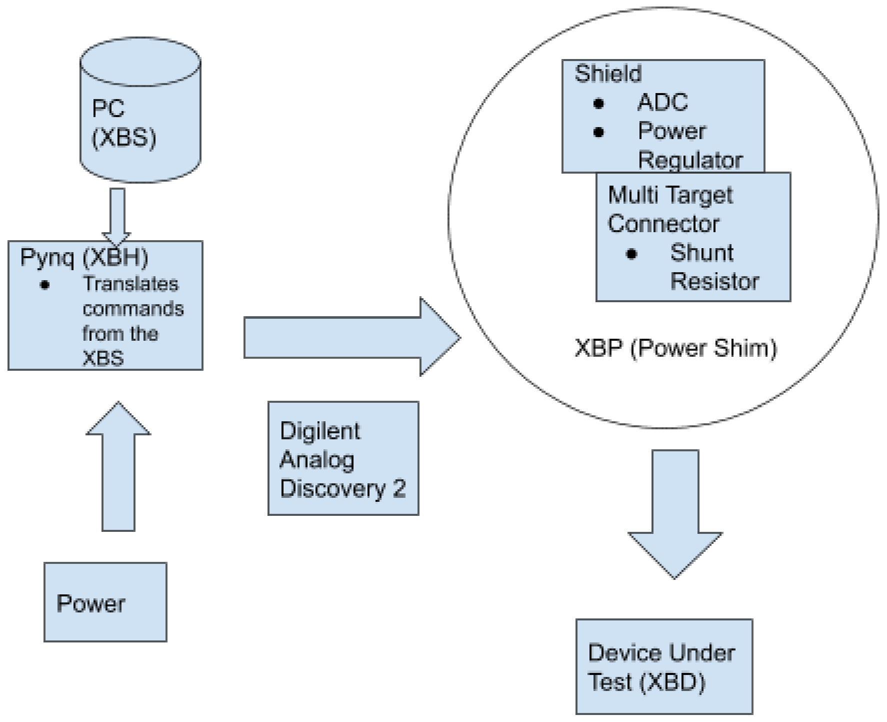


Figure 9: XXBX Implementation with our System

The figure above depicts how the XXBX protocol will be implemented, with the particular circuit boards that will be used in our project. The diagram specifies which circuit boards correspond to each stage of the XXBX benchmark.

In the first stage, the PYNQ board is connected to the PC (which is the XBS) as the XBH of the system. Here, the job of the PYNQ board is to translate the commands from the XBS, and measure the ROM usage, RAM usage and power consumption with the help of an ADC. From there, the user can use Analog Discovery 2 to analyze the signals created by the PYNQ circuit board by way of connecting it to a PC with a USB connection. The CERG FOBOS shield then communicates with the PYNQ board through I^2C. The shield's ADC is used to measure the analog power signal and the it's power regulator is used to regulate the power that goes to the XXBX Device Under Test (DUT), also known as the XBD. The leaked power signal of the XBD through a multitarget connector (MTC) board, which is used as a current sensing shunt resistor for the power signal going to the XBD. The shield, MTC and DUT are also connected using the I^2C protocol.

The execution time on DUT is important for measurement and analyzing, therefore, the DUT relays this information, along with additional analog measurements, back to the XBH (PYNQ board in our case). After analysis on the PYNQ, we can use the resulting power consumption graph to attempt to break the cryptographic algorithm in order to obtain the encrypted key.

Note: The FOBOS shield is actually a previous design. There are a few potential issues, such as the noise in ADC when the input signal has high frequency, and the voltage regulator chip overheating possibly due to insufficient cooling conditions. It is our intention to identify the cause of each problem and develop adequate solutions to improve the functionality of the FOBOS shield.

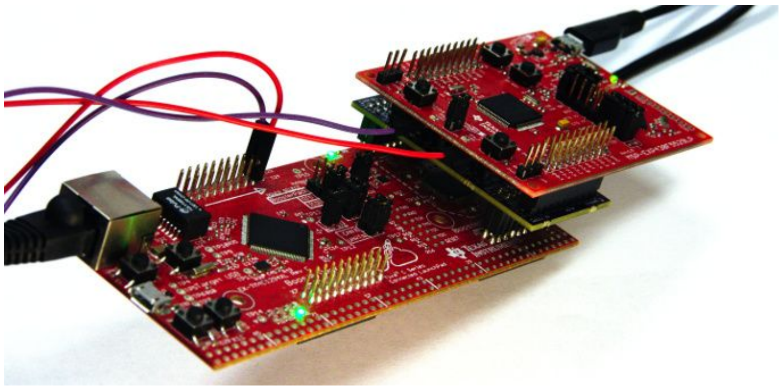


Figure 10: Example of an XBX Harness

Above is a picture of the XXBX Harness (XBH) on the bottom. The XBH is a microcontroller board connected to a computer running the XBS and to the XBD via I2C. The XBH also passes programs filled with data relevant such as power consumption and execution time (Note: The XBH in our project will be the PYNQ board). In the middle, is an XBP board specializing in power regulation and current sensing. At the very top, an XBD board performs execution, verification, and stack usage measurement. The data obtained in the XBD will then be sent back to the XBH as shown in.

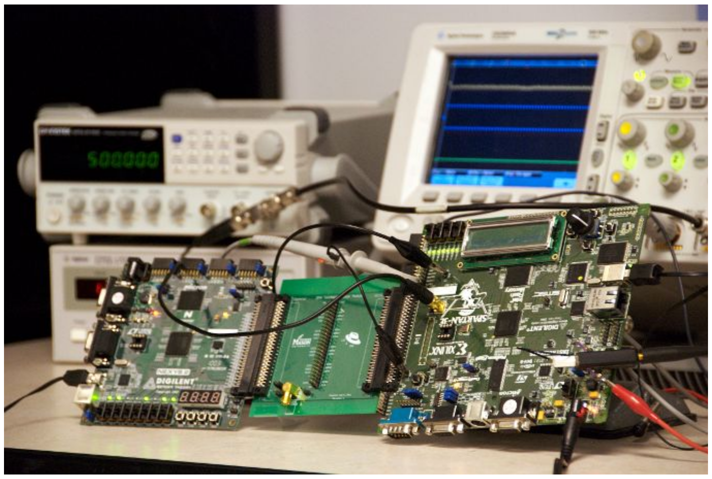


Figure 11

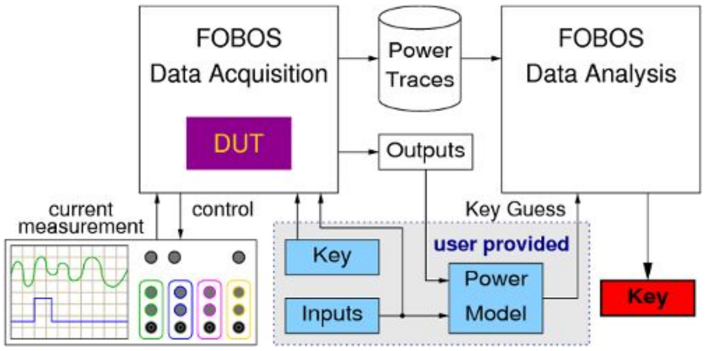


Figure 12: FOBOS Overview[2]

Above is a photo of FOBOS, as well as the technical sketch of how FOBOS works. FOBOS is an open-source framework used to conduct side-channel analysis attacks on Field Programmable Gate Arrays (FPGA) which supports multiple FPGA boards. FOBOS is made in a way that it is easily adapted for newer FPGA boards, oscilloscopes, and a wide variety of techniques to attack. In addition, FOBOS includes all the software needed to run the most prominent of all side-channel analysis attacks, Differential Power Analysis attacks. FOBOS supports power analysis of FPGAs and includes the software to control the device when attacked. In the midst of attack, FOBOS is able to trigger an oscilloscope, as well as obtain and analyze the readings using a plethora of power analysis techniques.

Unit Definitions of Target Metrics

XXBX

- ROM
  - Measured in Kilobytes
- RAM
  - Retrieved from XBD
- Runtime
  - Cycles per bytes
  - Measured one block of data at a time
- Power Consumption
  - Measured in Watts
- Power Leakage from Side Channel
  - Measured in Watts

6. Preliminary Experiment Plan

By next semester, the entire system design has to be complete and ready to begin running cryptographic algorithms. This includes, calibrating the power supply and connections between the boards, setting up software and hardware communication using I2C, designing the copper area on FOBOS shield circuit board, and establishing XXBX protocol to communicate with the device under test. The goal for next semester is run and crack as many cryptographic algorithms as possible.

List of Experiments:

1) Attack on Advanced Encryption Standard (AES): The file containing the plaint-text and key located on the PYNQ board will send the information through the interfacing board and FOBOS shield to the MSP430s for encryption using AES. After the encryption is done the cipher will be sent back to the PYNQ board. The ADC will be used to measure power consumption, and we will use that data to try to find the hidden key code. This experiment procedure will be similar for several cryptographic algorithms.

2) T-test on encryption algorithm: This test is used to determine the amount of leakage of the system without imposing an attack. If there is minimal leakage, then the T-test is said to be passed. If there is a significant amount of leakage, the T-test fails. It is important to note that failure of the T-test does not guarantee that the code will be cracked, and the key will be obtained. It simply tests whether or not the data is protected or unprotected.

6. Preliminary Project Plan

What follows is a listing of the tasks we plan to accomplish by the end of this semester.

1) Power Supply

FOBOS 3 can supply power to the microcontroller that we want to test.

People: Lead: Bradford

Tasks:

- Check connections from FOBOS Shield to microcontroller

- Calibrate voltage and current measurement
    - Establish relationship between multimeter, adc, and manually set data
    - Confirm feasibility of linear regression and/or polynomial regression in jupyter notebook
    - Learn and code linear regression and curve fitting methods in jupyter notebook
- Take measurements using multimeter
- Solder load resistor bundle

Resources:

- PYNQ board with FOBOS Shield (picked up by Brad 09/30/19)
- 3 160 Ohm Resistors
- 1 Breadboard
- 1 Multimeter
- 5-8 pin wires
- Access to FOBOS git repository
- PYNQ-Z1 page: https://store.digilentinc.com/pynq-z1-python-productivity-for-zynq-7000-arm-fpga-soc/
    - contains links to datasheet, schematics, and resource center

2) I2C Communication Software

Python software to send and receive data as I2C master via an I2C peripheral, and C software for the MSP 430 to talk I2C as a slave.

People: Lead: Theresa, Second: Jorge

Tasks:

- Get I2C working between two MSP430s
- Observe I2C data using a logic analyzer
- Get I2C working on FOBOS 3 in Python (depends on I2C HW task)

Resources:

- ECE 447 slides on I2C and code example
- Logic Analyzer (Analog Discovery 1)
- PYNQ board with FOBOS Shield and MTC

3) I2C Communication Hardware

We need I2C hardware on PYNQ

People: Lead: Vincent, Second: Bradford

Tasks:

- PYNQ has I2C in the PL, do we have access to the pins?
- Xilinx has I2C IP called IIC, is this available for these boards: PYNQ-Z1, Ultra96-V2, Basys3?
- Use FOBOS2 and FOBOS3 Vivado projects from git and try to add IIC
- Create a new Vivado project for the Ultra96-V2 and try to add IIC
- Add I2C to FOBOS2 and FOBOS3

Resources:

- Access to FOBOS git repository
- **PYNQ-Z1** page: https://store.digilentinc.com/pynq-z1-python-productivity-for-zynq-7000-arm-fpga-soc/
  - contains links to datasheet, schematics, and resource center
- **BASYS3** page: https://store.digilentinc.com/basys-3-artix-7-fpga-trainer-board-recommended-for-introductory-users/
  - contains links to datasheet, schematics, and resource center
- **Ultra96 V2** page: https://www.96boards.org/product/ultra96/
  - contains links to datasheet, schematics, and much more

4) FOBOS Shield Circuit Board

The FOBOS Shield works, mostly but still has a few bugs that have to be resolved.

People: Lead: Dr. Kaps, second Theresa

Tasks:

- Use KiCAD, open FOBOS shield and FOBOS MTC
- Determine the size of copper area needed to provide adequate cooling of the TPS7a7100 chip for the variable power supply or if a heat sink is needed
- I2C might require change of traces

- Can the V_in pin on the PYNQ connector be used to power the FOBOS shield, if so, design a 5V 1.5A power supply that fits on the shield.

Resources:

- Access to FOBOS git repository
- Datasheet of power chip TPS7A7100: http://www.ti.com/lit/ds/symlink/tps7a7100.pdf

5) XXBX Protocol

XXBX uses a specific protocol over I2C to talk to the device under test: XBD

People: Lead: Jorge, Second: Vincent

Tasks:

- Read XXBX documentation
- Install XXBX system on a Linux machine
- Run XXBX for an algorithm
- Observe XBH-XBD communication with a logic analyzer

Resources:

- XXBX git: https://github.com/GMUCERG/xbx
- XXBX Setup
- Logic Analyzer

6) Side-channel Analysis

Determine side channel leakage of cryptographic algorithms on microcontrollers

People: Lead: Jorge

Tasks:

- Read FOBOS Documentation
- Arrange meeting with Abubakr for T-Test demo
- Install FOBOS on a Linux Machine
- Get power traces from Abubakr and run analysis yourself
- Work with Abubakr on the FOBOS Tutorials

Resources:

- Abubakr Abdulgadir <aabdulga@gmu.edu>
- Access to FOBOS git repository

7. Potential Problems

Side channel analysis is a topic not taught in most undergraduate ECE courses. We will have to learn about cryptographic algorithms, their mechanism of encryption, and their strengths and weaknesses. The most important however, is knowledge of how the CPU of a microcontroller implements these algorithms. This will help us to identify when the CPU is performing computationally expensive operations such as modular exponentiation and multiplication, as well as how many clock cycles it will take to encrypt one byte.

This project utilizes already existing tools and software. Expanding upon the project using the MSP430 can introduce errors not accounted for by previous developers. Some of these errors could be communication related such as interfacing the XBX harness with the device under test via I2C, or power related such as over-heating of the CPU.

8. References

[1] S. Bhunia and M. Tehranipoor, "Side Channel Attack," *Side Channel Attack - an overview | ScienceDirectTopics.*[Online].Available: www.sciencedirect.com/topics/computer-science/side-channel-attack. [Accessed 11-Oct-2019].

[2] **https://cryptography.gmu.edu/fobos/**
https://github.com/GMUCERG/xbx/blob/master/docs/XXBX-UserGuide.pdf

[3] https://www.rambus.com/security/dpa-countermeasures/training/

[4]https://store.digilentinc.com/analog-discovery-2-100msps-usb-oscilloscope-logic-analyzer-and-variable-power-supply/

[5] https://store.digilentinc.com/pynq-z1-python-productivity-for-zynq-7000-arm-fpga-soc/

[6]http://www.laits.utexas.edu/~anorman/BUS.FOR/course.mat/SSim/history.html

[7] https://cryptography.gmu.edu/xxbx/

[8] https://csrc.nist.gov/Projects/Lightweight-Cryptography

Appendix B: Design Document

ECE 492

Senior Design Document

Side Channel Analysis on Microcontrollers

Team Members:
Theresa Nguyen, Jorge Rodriguez, Vincent Guevara, Bradford Webb

Faculty Supervisor:
Dr. Jens-Peter Kaps

Table of contents

1. Introduction and Problem Statement

The emergence of Internet of Things (IoT) devices, as well as other areas such as sensor networks and cyber-physical systems, have greatly benefited their respective areas of influence.

However, these situations where highly-constrained devices are wirelessly interconnected through a network have raised significant concern regarding their overall security. A majority of the current cryptographic standards are not designed for such devices, which prompted NIST (National Institute for Standards and Technology) to launch a competition to standardize lightweight cryptographic algorithms. These algorithms are designed for use in IoT devices and other similar constrained devices to ensure proper encryption of data passing through the networks.

Running cryptographic algorithms in hardware and software often allows leakage of information through system side-channels. This leakage is often in the form of power consumption and electromagnetic radiation, and could result in acquisition of the secret key. This design project involves modifying an existing PYNQ (Xilinx Zynq SoC running Linux) based side-channel analysis testbed to connect to an MSP430 Microcontroller using the I2C serial communication protocol. The main goal is to integrate the eXtended eXternal Benchmarking eXtension (XXBX) protocol, and obtain key metrics through side-channel analysis. Additionally, due to many side-channel analysis systems being quite costly, this project is intended to be an affordable, open-source alternative for educational and research usage.

2. Requirements Specification

NIST is currently in the process of selecting one or more authentication encryption that would be more suitable for highly-constrained devices. The submissions for the standardization process must fulfill NIST's evaluation criterias. The submissions will be evaluated on security, cost, performance, and side channel resistance. These criterions will be taken into account when implementing this design project. FOBOS (Flexible Open-source workBench fOr Side-channel analysis) will be able to control the DUT(device under test) in order to obtain all the required measurements and analyze them.

2.1 Mission Requirements:

- This project should implement a system that performs side-channel analysis (SCA), while also gathering various other important metrics such as ROM and RAM usage, runtime, and power consumption.
- Provide support for competitions of cryptographic algorithms through benchmarking and evaluation.
- Should be a lower cost, open-source alternative to existing SCA systems.

2.2 Operational Requirements:

- **Input/Output Requirements**
    - C code for a cryptographic algorithm will be an input to the system.
    - Selected DUT will be an input to the system.
    - The boards making up the system must contain general-purpose input output (GPIO) pins for easy interconnectivity between them.
    - This system shall receive commands to configure settings for varying the voltage and gain.

    - System must provide a port(s) to allow the connection of current probes for power measurements.

- **External Interface Requirements**
    - This system shall take a 12V, 3A of power via an external power supply.
    - The control board (PYNQ) will connect to a user PC via a Gigabit ethernet cable.
    - A ribbon cable will be used to connect the Multi-target Connector board to the FOBOS Shield board.

- **Functional Requirements**
    - Successfully obtain the desired metrics at proper stages of the system.
    - Calibrating the power generation and measurement from the FOBOS Shield board.
    - All boards should be communicating using the I2C serial language protocol.

- **Technological and System-Wide Requirements**
    - Base new system on existing FOBOS and XXBX projects
    - PYNQ (Xilinx Zynq SoC running Linux) board (control board)
    - A PC with Linux OS installed.
    - Python 3 programming language must be installed on the machine for software development.
    - Xilinx Vivado Design Suite must be installed on the machine for compiling control software and generating the bitstream.

3. System Design

3.1 Level 0 Diagram

The level 0 data flow diagram (shown in Figure 3.1) helps with the understanding of what our overall system inputs and outputs are. The system should be able to take in all the external inputs and through software, implements XXBX benchmarking, runs cryptographic algorithms, and computes metrics in order produce ROM usage, RAM usage, runtime, power consumption, and information side channel leakage independently without the help of any other measuring tools (e.g. oscilloscope).

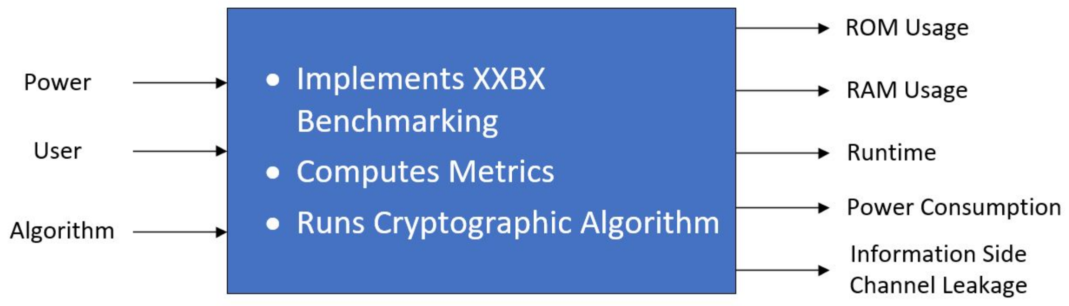

Power
User
Algorithm
Implements XXBX Benchmarking
Computes Metrics
Runs Cryptographic Algorithm
ROM Usage
RAM Usage
Runtime
Power Consumption
Information Side Channel Leakage

Figure 3.1

3.2 Level 1 Diagram

The level 1 data flow diagram (shown in Figure 3.2) breaks down level 0 diagram into each device that would form the whole system. The PYNQ, Shield, Multi-target connector, and the device under test are all communicating with each other via I2C. A power supply is needed in order to power on the PYNQ which the PYNQ then will supply power to all the other devices. The PYNQ will cross-compiled the algorithm to be sent to the DUT. The PYNQ then waits to see how long it takes for the DUT to run the algorithms. The PYNQ will then analyze the result of the timing signal. The purpose of the shield is to help with power consumption measurement by utilizing its internal ADC. The Multi-target connector can vary how much voltage is needed in order to power up the DUT. It also allows different types of microcontrollers to be tested for side channel analysis.

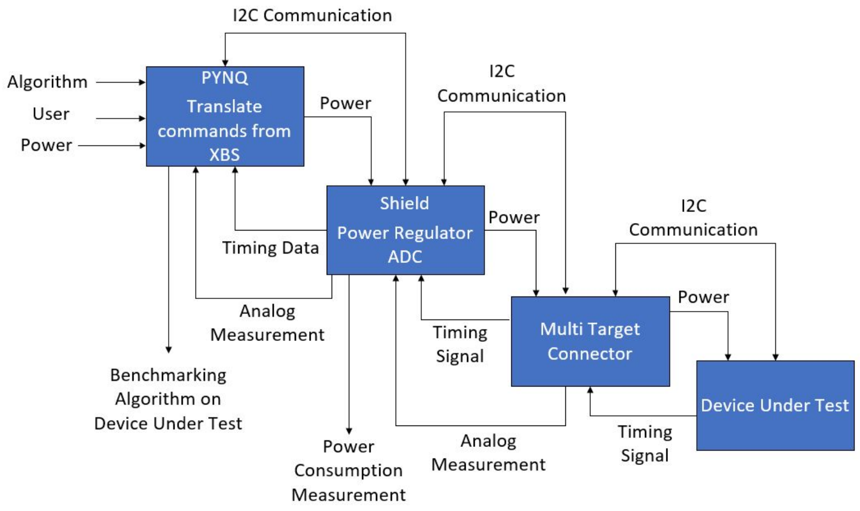


Figure 3.2

3.3 Level 2 Diagram for PYNQ

Now that the system has been described at the level one decomposition stage, it is possible to go further into the details of each individual component. Figure 3.3 shows the level two decomposition of the PYNQ board. The purpose of the ethernet input is to interface the System-on-Chip (SoC) on the PYNQ board, which contains an ARM processor that runs Linux OS and has Jupyter notebooks installed on it, with a user PC. This allows the user to view the files stored on the chip, and run the python code on it. The power signal that the PYNQ board is receiving comes from an AC adapter that is plugged into an electrical outlet. The adapter supplies 12V, 3A of power to the PYNQ board. This signal is then passed into a voltage/power regulator, so that the signal can be attenuated to 5V in order to power the FOBOS shield board.

In addition to the ARM processor, the SoC also contains an XADC that accepts analog measurements that are returned from other parts of the system. The clock wizard manages and delivers clock signals to several components such as the ADC interface, timer module, and IIC (inter-integrated circuit) module. A power control module sends power settings to the rest of the system regarding voltage and gain levels. Lastly, the IIC module handles the information being sent and received by the PYNQ board through I2C communication. The PYNQ board is the control board in our system, which is why it not only sends commands and configurations, but also receives all of the data and measurements for final analysis at the end of the runtime. Ideally, the PYNQ board will work as the XXBX System (XBS) and XXBX Harness (XBH) after the XXBX protocol is integrated into the new FOBOS project. As the XBS, it will Cross compile cryptographic algorithm code. As the XBH, the PYNQ will handle protocol conversion and send data and instructions to the rest of the system via I2C, measure power consumption of the target device, and measure the runtime of the cryptographic algorithm through a timing signal obtained from the DUT.

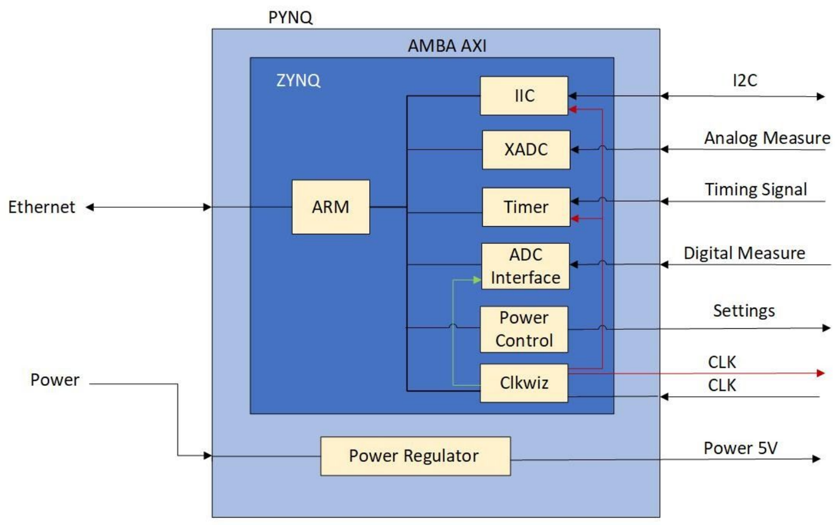


Figure 3.3

3.4 Level 2 Diagram for FOBOS SHIELD

The FOBOS shield is responsible for providing power to the DUT, as well as handling the power consumption measurement used for side-channel analysis through the high speed ADC. The shield provides several different values for powering the DUT through two voltage regulators and jumpers. This allows for more than just one device to be used as the DUT since different devices may have distinct power requirements. The current sensing modules on each regulator are to obtain analog current measurements that will be sent back to the PYNQ board for recording and analysis. Similar to the other boards, the shield has an IIC module that transmits and receives data through I2C. The shield will serve as part of the XXBX Power (XBP) and will perform power regulation to the DUT, and obtain analog measurements through current sensing modules.

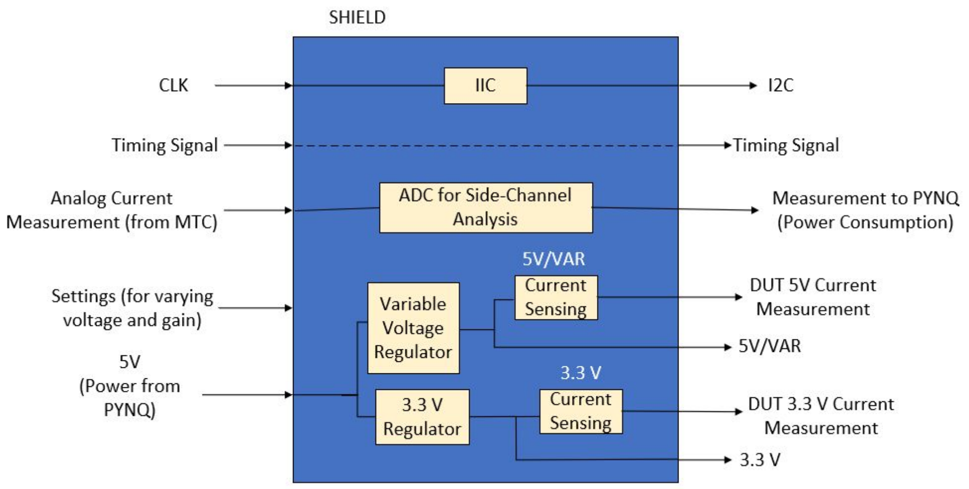

SHIELD
CLK
IIC
I2C
Timing Signal
Timing Signal
Analog Current Measurement (from MTC)
ADC for Side-Channel Analysis
Measurement to PYNQ (Power Consumption)
5V/VAR
Current Sensing
Settings (for varying voltage and gain)
Variable Voltage Regulator
DUT 5V Current Measurement
5V/VAR
5V (Power from PYNQ)
3.3 V
3.3 V Regulator
Current Sensing
DUT 3.3 V Current Measurement
3.3 V

Figure 3.4

3.5 Level 2 Diagram for Multi-Target Connector

The Multi-target connector (MTC) will use I2C in order to communicate with the shield and the DUT. It also passes the timing signal that it receives from the DUT to the shield. The importance of having a multi-target connector is that it can supply the required voltage to the DUT via the shunt regulator pins for 3.3 Volts and 5V/Variable Volts respectively. The power setting is important as to safely power on the DUT and it also will affect the power consumption analysis as the active pin setting will have more voltage than the inactive pin setting. The MTC will serve as another part of the XBP by providing some more current sensing functionality in order to get the current measurement that will be an input to the high speed ADC on the FOBOS shield board.

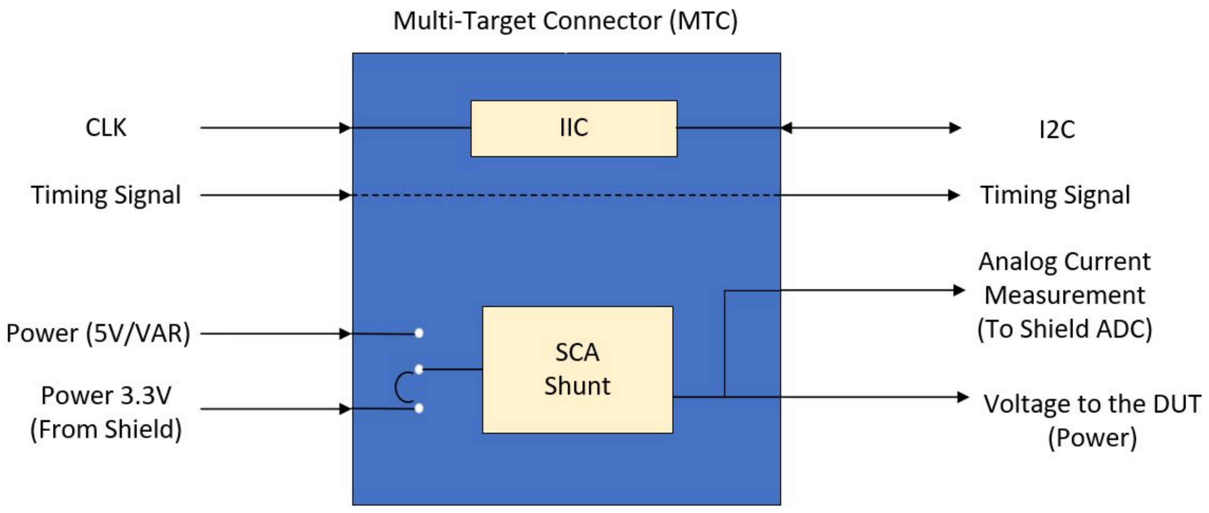

Multi-Target Connector (MTC)
CLK
IIC
I2C
Timing Signal
Timing Signal
Analog Current Measurement (To Shield ADC)
Power (5V/VAR)
SCA Shunt
Power 3.3V (From Shield)
Voltage to the DUT (Power)

Figure 3.5

3.6 Level 2 Diagram for Device Under Test

The DUT is the microcontroller that the system wants to perform side channel analysis on. The Shield will supply the required power for the microcontroller to function properly. A bootloader is needed in order to compile programs without the help of an external compiler(e.g., Code Composer Studio). The bootloader will compile the program and loads it into the function core for the program to run. Using I2C and a clock, the PYNQ board is able to keep track of how long it would take for the DUT to finish running the algorithms.

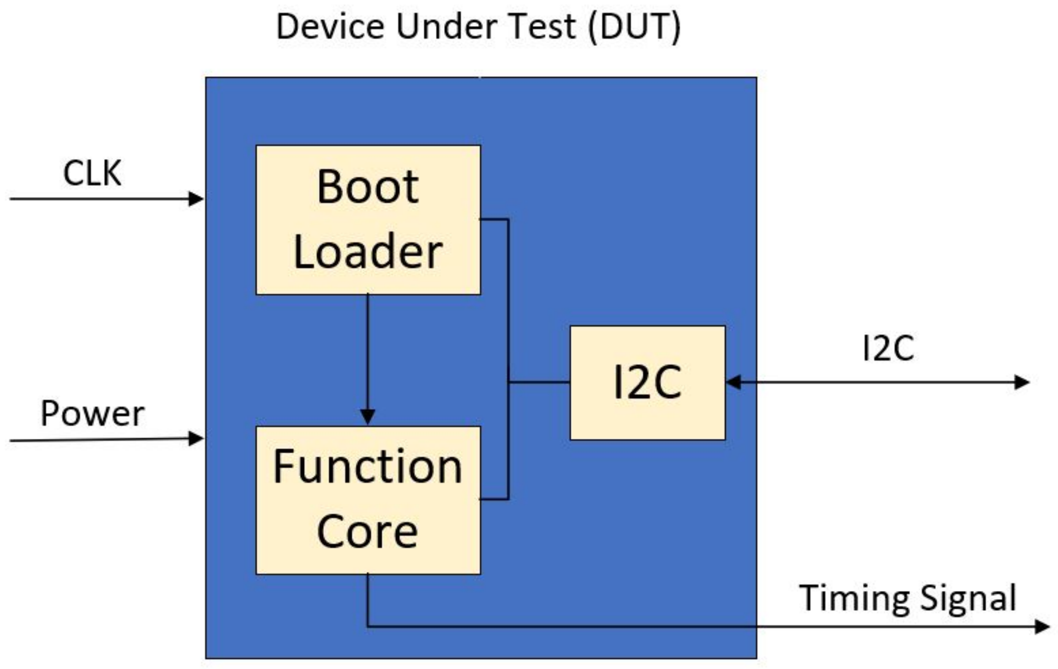


Figure 3.6

3.7 Physical Architecture

The overall system of the FOBOS 3 on Microcontroller (shown in Figure 3.7) comprises of four components. The PYNQ which is the control of the system will be responsible for generating test vectors that will be sent to the MSP430. The PYNQ is also responsible for measuring the time it took for the DUT to finish running the algorithms. The shield has current sensing and signal conditioning that will modify the signal in a way that the ADC on the PYNQ would able to accept. The ADC in the PYNQ is responsible for measuring the power consumption, which is the input we need in order to perform t-tests. The MTC has a current sensing will be modify the signal so that the ADC in the shield would be able to accept. The high speed ADC in the FOBOS will provide the power variation that occurred inside the MTC.

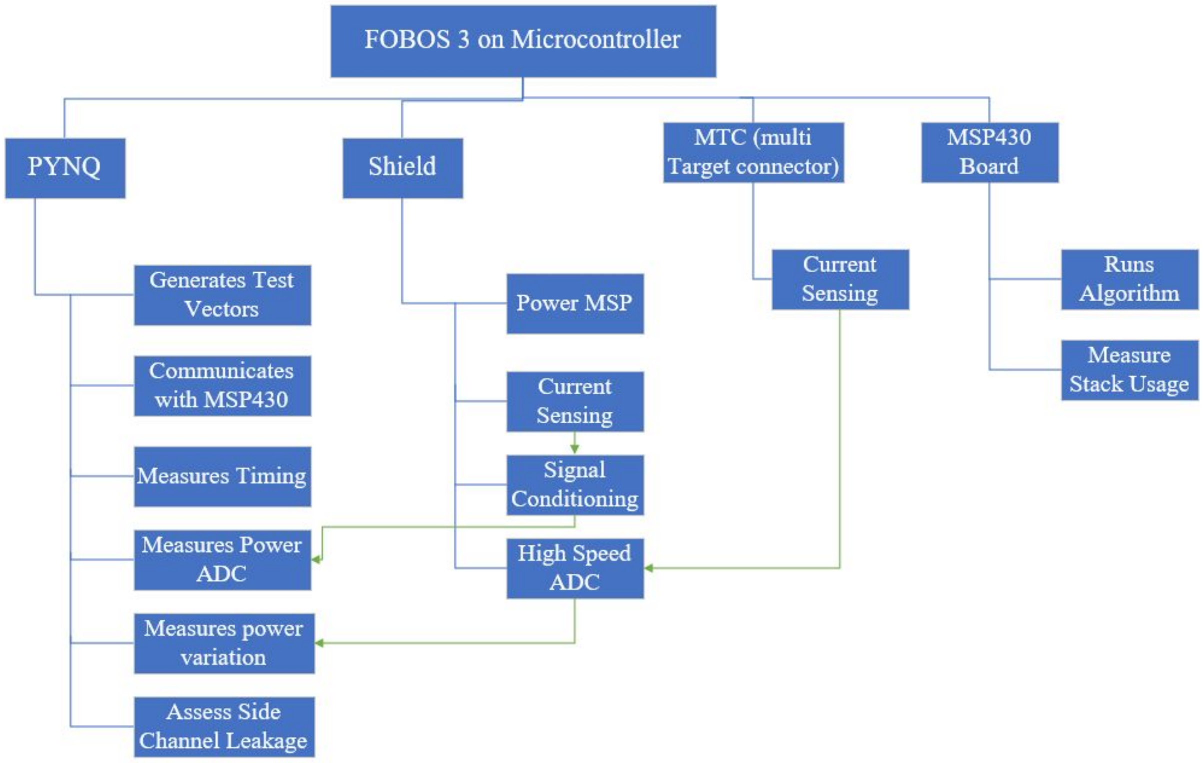


Figure 3.7

3.8 Overall System Architecture

Figure 3.8 shows how the system is connected and flows from the computer to the DUT and then back to the computer with the results of the side channel analysis. The system starts once the user starts the Jupyter notebook to run a test, the text in the same directory goes from the Pynq to the shield, through the multi-target connector, and then to the DUT. Once the data is at the DUT, the algorithm contained inside of it is executed and the stack usage is measured. That stack usage measurement goes back from the MSP430 through the Multi-Target Connector and to the FOBOS shield. As the data goes from the DUT to the Shield , the current of the DUT is sent to the High

Speed ADC on the FOBOS shield and will be later compared to the current from the FOBOS shield. Now the data the PYNQ did not have access to from the other components is now back to the PYNQ for final processing so it can determine if there is side channel leakage, the timing of the entire process, the RAM and ROM used, and the power consumption and variation. Resultant ciphertext is placed in the pre-determine directory based on the python code of the jupyter notebook on the host computer.

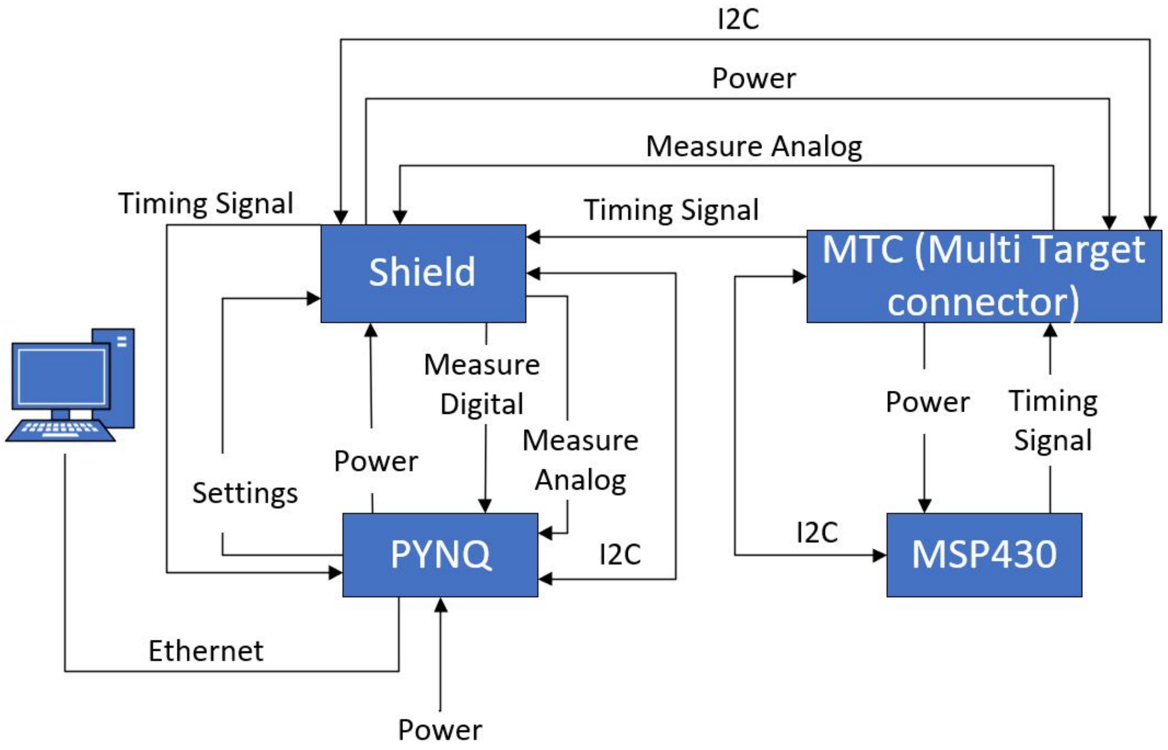


Figure 3.8

4. Background Knowledge/Phenomenology

4.1 Cryptographic Text

Input is a plaintext file which then goes through an explicitly complicated algorithm (i.e. a cipher) and then the output is an encrypted ciphertext file. The encryption process happens through a Jupyter notebook file. The process happens one block at a time until the entire file is fully encrypted. An example of encryption is Advanced Encryption Standard AES-128. AES is firstly a part of Correlational Power Analysis. If the encryption happens properly then the ciphertext will be able to be de-encrypted by the key and the de-encrypted text will match the plaintext. If the text does not match then, an attack has occured that changed the nature of the text. Below in Figure 4.1 is a basic, high-level diagram of encryption of plaintext data.

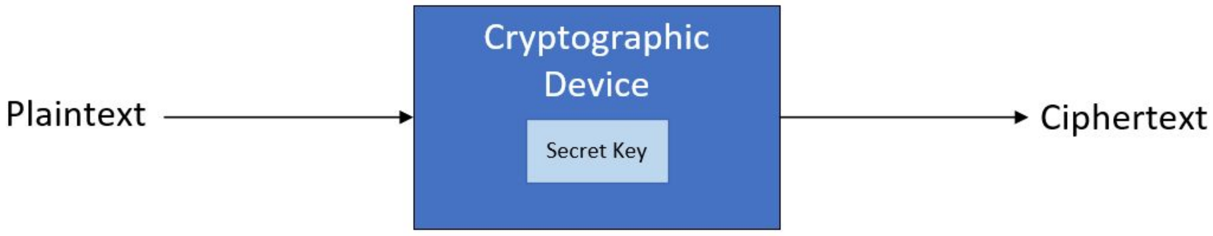


Figure 4.1

4.2 Side-channel Analysis

A Side-channel attack is an attack that analyzes information gathered from a system when performing a cryptographic algorithm, rather than looking for weaknesses in the algorithm itself according to Bochert [1]. Data such as timing characteristics, power consumption, and electromagnetic radiation leaks provide information about the cryptographic algorithm used and can lead to reverse engineering of the secret key. Because these attacks are almost impossible to prevent remotely, it is critically important to test cryptographic algorithms on benchmarking hardware to determine the amount of information leakage occurs. If the leakage is substantial, the algorithm may be considered an unprotected implementation.

$$t = \frac{\overline{X}_1 - \overline{X}_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}}$$

Figure 4.2a

The tool used for leakage assessment in FOBOS is known as the Welch's t-test. This statistical test is an adaptation of the more popular Student's t-test, and works better when two samples have unequal variances and/or unequal sample sizes [2]. As a result, it works very well with the FOBOS system in analyzing fixed versus random power traces. Fixed power traces are obtained through the repeated encryption of the same data, while random power traces are obtained through the encryption of different data (the data in this case is plaintext). The t-test takes these random and fixed power traces as inputs and allows for comparison of the leakage of the same cryptographic implementation, but with both fixed and random plaintexts. If these values are not distinguishable, meaning they look relatively the same, then the implementation may be considered protected. However, if the random and fixed leakages can be distinguished, the implementation is considered unprotected and fails the test. In general, a t-value of ±4.5 or higher results in failure of the t-test. Figure 4.2a above is an image depicting the formula for Welch's t-test, where the X's are sample means, s_0 and s_1 are sample variances, and N_0 and N_1 are the sample sizes. Figure 4.2b shows an example of its use with an AES-GCM implementation.

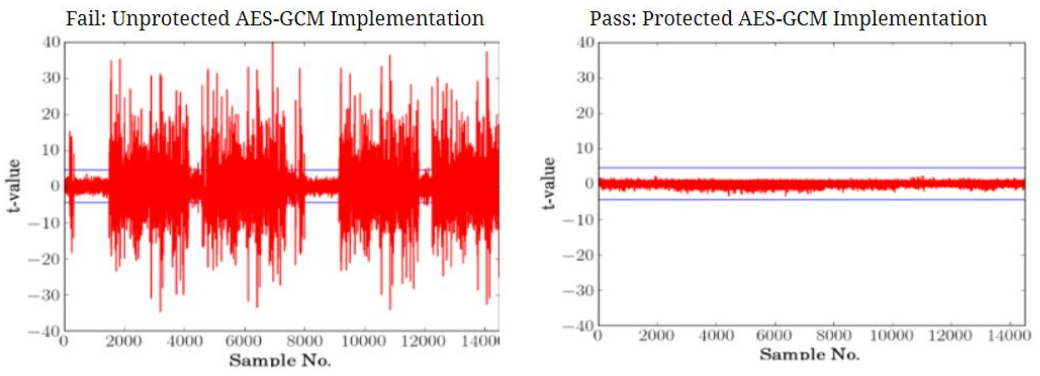


Figure 4.2b

4.3 Differential Power Analysis

Differential Power Analysis (DPA) is a type of side channel attack. It consists of taking a collection of power traces over time and uses statistical methods to ascertain the key value. A popular type of DPA is correlation power analysis (CPA). CPA uses an intermediate value that is a function of part of the key and known data. The power consumption of the devices when the intermediate value is processed is estimated for each key guess. A statistical method is then used to find out which key was most likely used by correlating the hypothetical power and the real power consumption according to Mangard, et. al. [3]

4.4 Voltage Chip Regulator (TPS7A7100)

The voltage chip regulator that is currently on the FOBOS shield is extremely hot when it is used. There is a thermal protection feature that will enable if the junction temperature is roughly 160C or higher. To ensure reliable operation of the system, thermal shutdown must be avoided. The datasheet that came with the TPS747100 chip provided a power dissipation equation as well as junction-to-ambient thermal resistance equation in order to help with the heat distribution.

Power dissipation equation: $P_D = (V_{IN} – V_{OUT}) * I_{OUT}$

Junction-to-ambient thermal resistance Equation: $R_{\theta JA} = (+125°C – T_A) / P_D$

Power dissipation can be minimized by using the lowest possible input voltage necessary to achieve the required output voltage regulation. Junction-to-ambient thermal resistance depends on the maximum ambient temperature, maximum device junction temperature, and power dissipation of the device. By calculating the $R_{\theta JA}$, Figure 4.4 show the amount of copper area can be estimated using the chart [5].

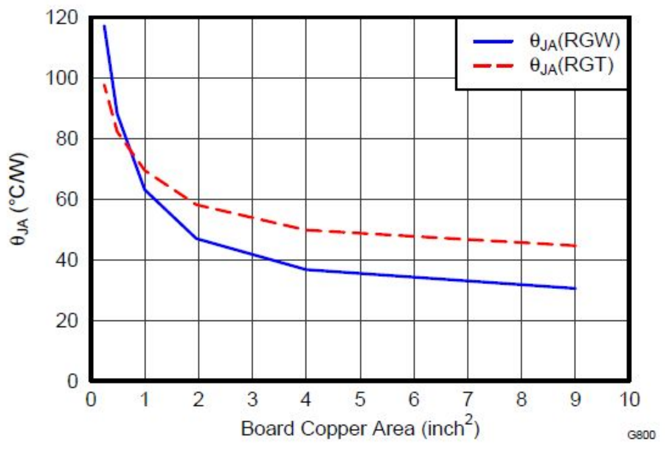


Figure 42. θ_{JA} vs Board Size

Figure 4.4

5. Detailed Design

Currently two Jupyter notebooks are being used to perform evaluation. One takes a plain text file block by block into ciphertext and the other is used for power consumption analysis. The power consumption metric is important so that one knows the values to expect when a device is under attack. The way the FOBOS system is able to calculate the power consumption is through the Xilinx ADC components that are internal to the PYNQ-Z1 board and ADC internal to the FOBOS shield as well. To further regulate the power for the DUT, the FOBOS shield has two pin settings for the respective shunt regulators of 3.3Volts and 5 or Variable Volts. The control board then has the ability to ascertain the instantaneous, average, and maximum values for the voltage and the current. Samples used and the variable voltage setting are also tabulated. Additionally, the shunt regulators have 4 gain settings each for even greater flexibility and to emphasize variances in power if necessary for better analysis. All of these factors help the FOBOS system be as flexible as possible when connecting to various types of microcontrollers.

5.1 Multi-Target Connector Schematic Breakdown

<u>ChipWhisperer Schematic</u>

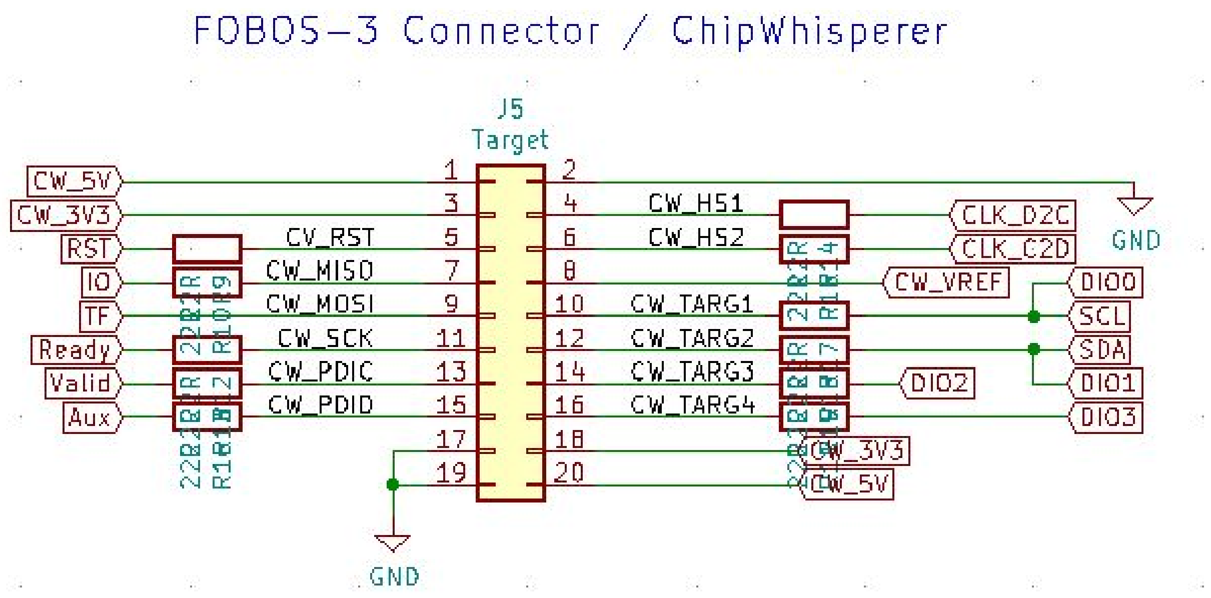


Figure 5.1a

This component serves as the communications interface between the PYNQ board and the underlying subsystems. On the left side of the schematic, there are MISO, MOSI, and SCK pins which allow for SPI functionality. On the right side of the schematic, there are SCL and SDA pins which allow for I2C functionality.

Power Filter and Power Measurement Schematic

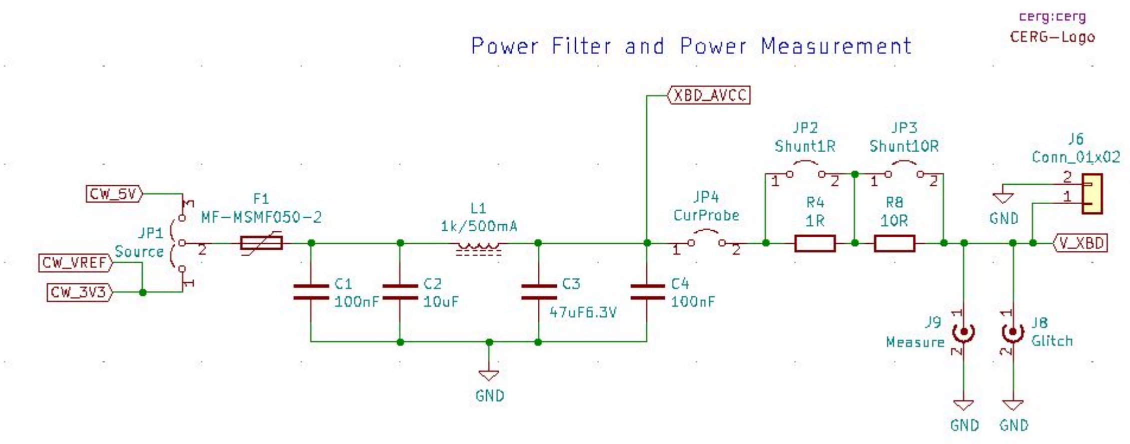


Figure 5.1b

Power measurement is conducted via the shunt resistors on the right hand side of the schematic. The filter portion of this circuit removes unwanted noise from the power signal to allow clean and accurate readings that will be sent back to the FOBOS shield board.

5.2 FOBOS Shield Schematic Breakdown

Power Supplies Schematic

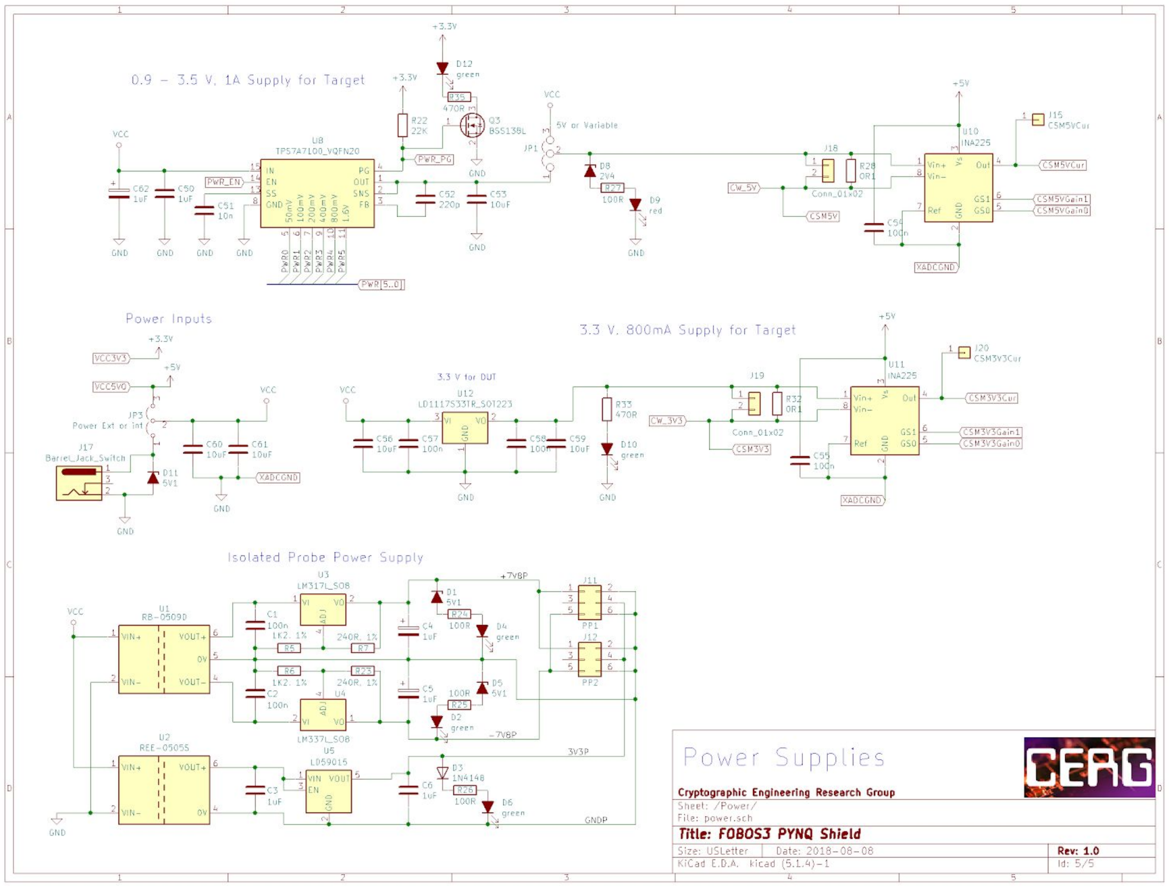

0.9 – 3.5 V, 1A Supply for Target
Power Inputs
3.3 V, 800mA Supply for Target
Isolated Probe Power Supply
Power Supplies
Cryptographic Engineering Research Group
Title: FOBOS3 PYNQ Shield
Size: USLetter Date: 2018-08-08 Rev: 1.0
KiCad E.D.A. kicad (5.1.4)-1 Id: 5/5

Figure 5.2a

The FOBOS Shield supplies power to the device under test. There are two power options available for use, a 3.3 V fixed supply and a 5V/Variable supply. The selection is chosen by connecting wires to the appropriate jumper on the board.

OpenADC Schematic

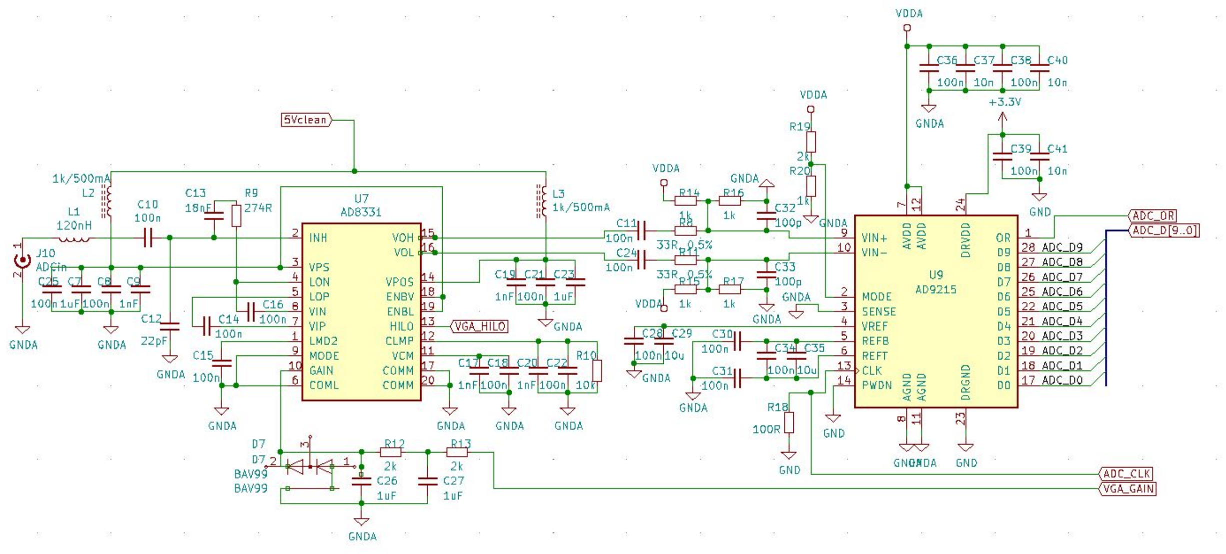


Figure 5.2b

The analog to digital converter on the FOBOS shield measures the current sent to it by the multi-target connector. The analog to digital converter is the AD9215, which has a 10 bit resolution and can measure up to 105 MS/s [4].

6. Prototyping progress report

6.1 Power Consumption Prototype

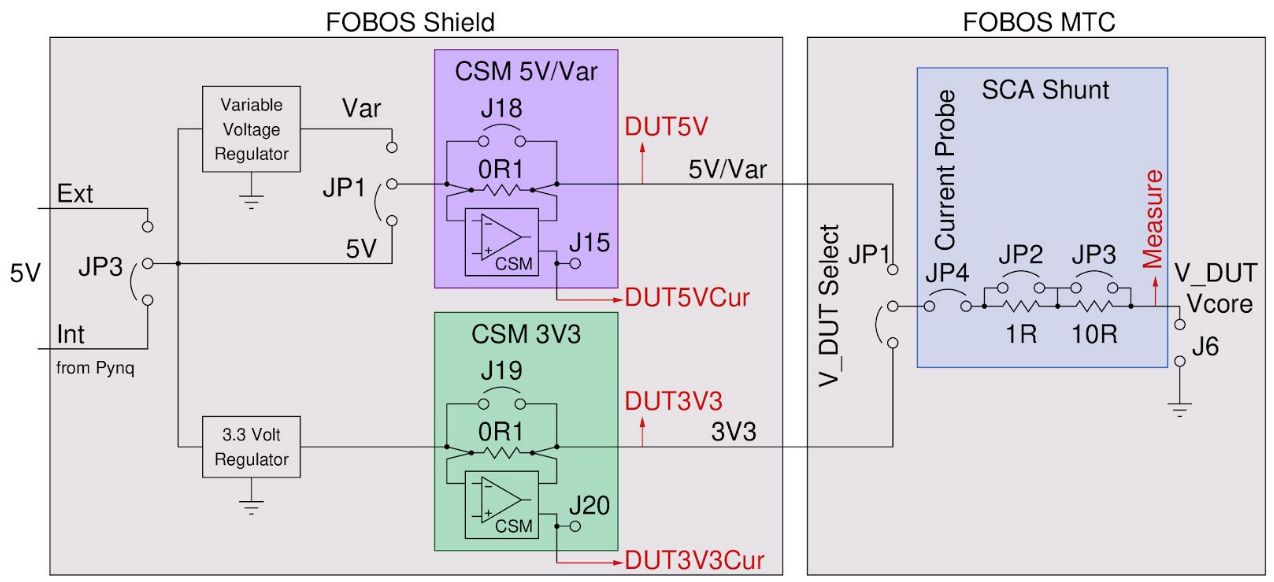


Figure 6.1

Power Consumption Verification is complete, but potential changes are under evaluation. More blocks that calculate current have been coded as well as multimeter confirmation of voltage values. Using Jupyter notebook to perform power analysis on control board and FOBOS shield via shunt regulators. The team discovered that PYNQ-Z1 evaluation board can handle both power supply input and USB input from computer. Previously it was assumed that the evaluation board could only handle the USB data/power input and the AC power supply was input to the FOBOS shield. After power consumption evaluation is finished, gain level outcomes will be recorded.

A majority of the issues encountered when working on this part of the project came when configuring the dedicated machine. The machine needed data cleared for a Linux dual boot partition to run the FOBOS project as intended.. After getting the linux partition, the repository had to have the permissions changed in order to push changes to build files related to the PYNQ bitstream file necessary to run tests via jupyter notebooks. After this issue was resolved, the PYNQ board required reformatting to set the password in the jupyter portal to its expected value. When the board was correctly formatted, a test was conducted with an FPGA device as the DUT to see if the FOBOS system successfully outputs ciphertext when given plaintext.

6.2 I2C Communication Hardware

In order for the PYNQ board to communicate with the DUT, the I2C protocol needs to be implemented. Prior to this, the already existing project needed to be created on a Linux machine. The project was downloaded from the FOBOS repository. When first building the project on Vivado, the implementation failed during the place design. This was due to a misnaming of one of the ports in the XDC file. After fixing the port name, the project was able to complete implementation and a bitstream file was successfully generated. The next step for the I2C hardware would be to integrate the I2C IP into the block design.

6.3 XXBX Prototype

The first few meetings regarding the XXBX and learning about its functionality started with making sure a computer was present with a Linux environment set. Once the Linux environment was setup on the computer, software packages from an Ubuntu repository were saved onto the computer to be able to communicate with the XXBX. However, in order for communication between the computer and the XXBX system to be fully established, a new ethernet profile needed to be created that contained the IP address of the XBXX control board. Once this was implemented, the computer and XXBX were able to communicate with one another by way of using the ping function on the command terminal of the Linux environment.

Once the code from GitHub and the XXBX were able to communicate, the full set up of the XXBX hardware was needed. This was done by connecting the pins from the XBH to the XBP, and then allocating the pins from the XBP to the XBD. Voltage then was wired from the XBP to the XBD using two female wires. The ground was then wired from the XBP to the XBD using a female wire, similar to the voltage connection.

Now that everything was connected and the XXBX was able to communicate with the computer (XBS), waveforms were then needed to be evaluated. The Waveform software made by

Texas Instruments was then downloaded onto the computer (XBS). After this, the computer's ethernet port was connected directly to the XBD to obtain results, with the computer's USB port connected directly to the XBH to provide voltage to the entire XXBX.

A walkthrough of the lengthy code attached to the XXBX interface came in handy in one meeting, and many more hours have been spent evaluating the Python code that makes the XXBX run properly. Now that everything is set up and the code is understood, for the most part, evaluations will take place futuristically. This will be done in order to evaluate more diverse waveforms and create the combination of group members' tasks into one task.

6.4 Overall Schematic

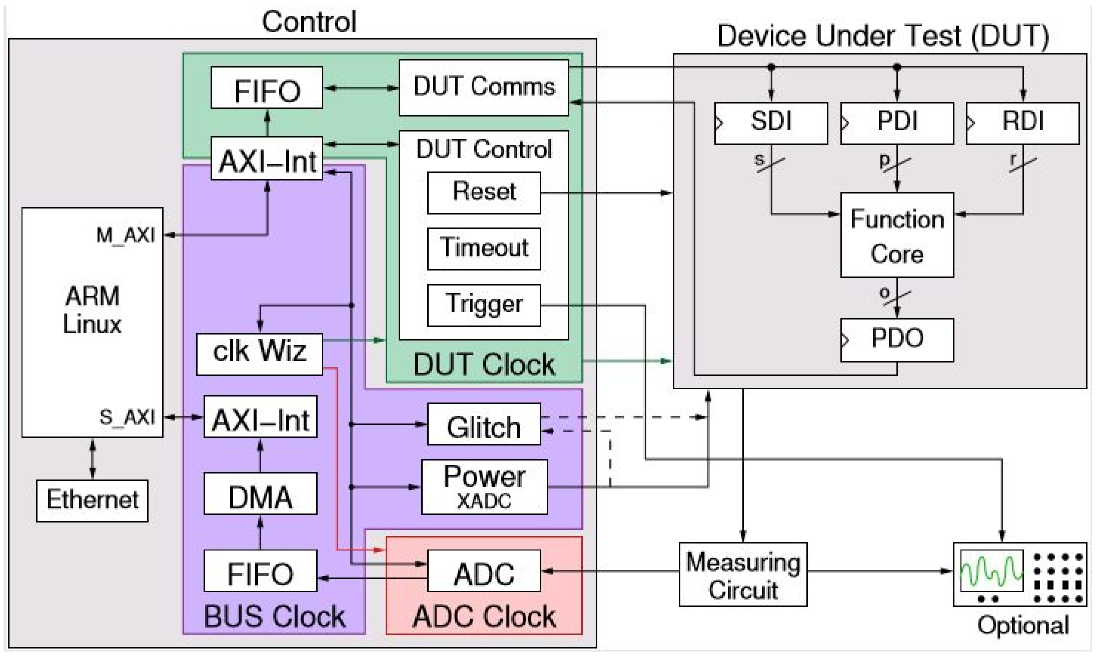


Figure 6.4

Figure 6.4 shows a detailed overview of the connection and communication between the control and DUT boards. While it is similar to the level 2 decompositions of the PYNQ board and DUT, this provides some information on the FOBOS 3 system in a generalized manner.

6.5 Necessary Parts/Hardware

- PYNQ-Z1 FPGA Evaluation Board
- FOBOS Shield
- Multi-target connector
- AC Power Adapter
- Ethernet cable
- Data USB to USB Micro cable
- Personal Laptop w/ Linux OS, Xilinx Vivado 2017.2, and Python2.7.15+
- MSP430 Microcontroller

7. ECE493 Testing Plan

To first test the XXBX component of the project, communication between the XXBX harness and XXBX device must be examined using a logic analyzer. After that, has been determined, the entire system will be run to see if the desired metrics are obtained and recorded in the XXBX harness (the PYNQ board in this case).

Since the voltage chip regulator temperature is still an ongoing issue with the FOBOS shield, the copper area required to dissipate heat on voltage chip regulator must be determined to help regulate the temperature and prevent the thermal protection feature from enabling. If this itself is not enough to provide adequate cooling, then rewiring the chip to another area on the board could be another option.

Power consumption is tested by using multimeter leads for voltage and/or current. The multimeter values are then compared to the values produced by the jupyter notebook. Different voltages can be tested by the shunt regulators to verify proper function of each power metric given that the different settings do not damage the DUT.

The final testing involves running a cryptographic algorithm throughout the entire system to verify if the appropriate measurements are accounted for and are accurate. The Welch's t-test should then determine whether the implemented cryptographic algorithm is protected or not based on the power traces obtained throughout the runtime of the system.

8. Schedule and Tasks

Side Channel Analysis on Microcontroller

Team Members:

Theresa Nguyen, Jorge Rodriguez, Vincent Guevara, Bradford Webb, John Lowry

Faculty Supervisor: Dr. Jens-Peter Kaps

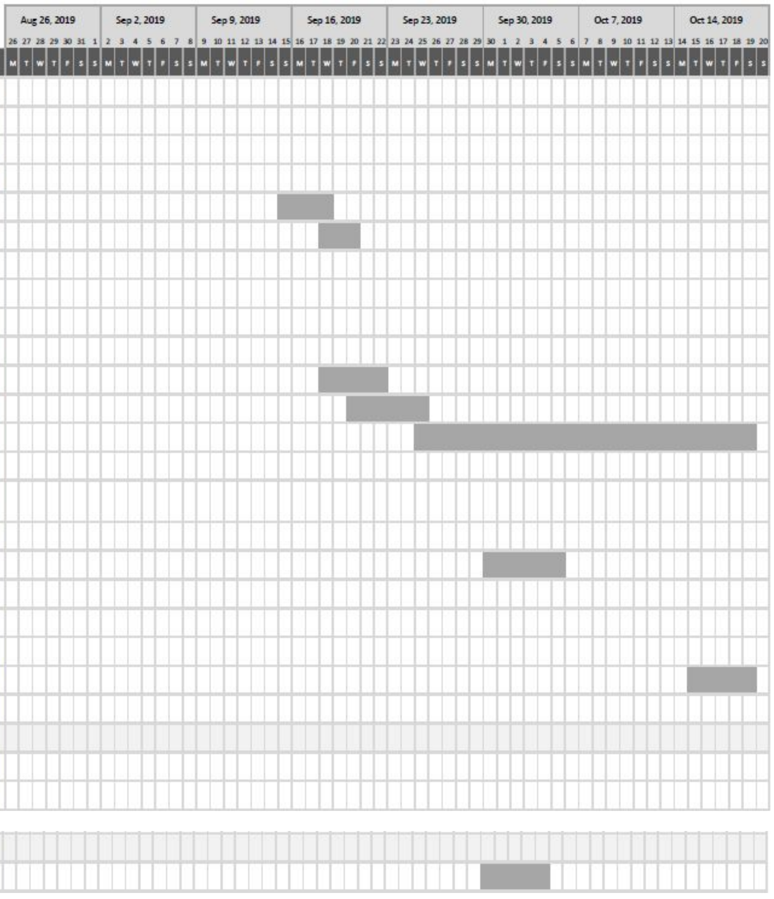
Project Start: Sun, 9/1/2019

Display Week: 0

SIMPLE GANTT CHART by Vertex42.com

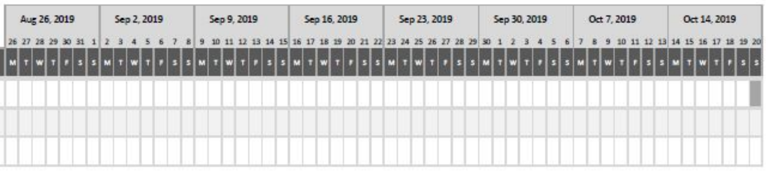
https://www.vertex42.com/ExcelTemplates/simple-gantt-chart.html

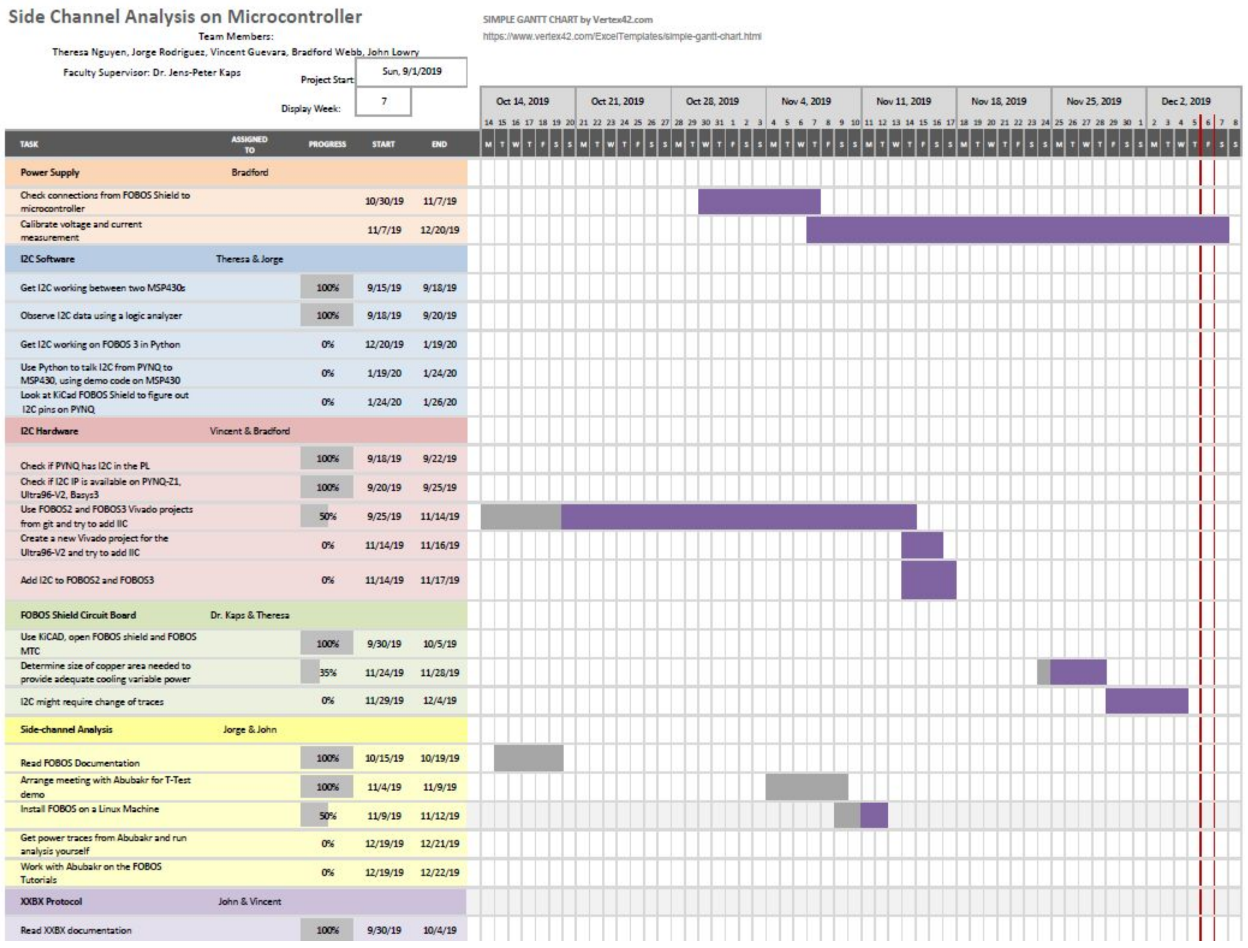
TASK	ASSIGNED TO	PROGRESS	START	END
Power Supply	Bradford			
Check connections from FOBOS Shield to microcontroller			10/30/19	11/7/19
Calibrate voltage and current measurement			11/7/19	12/20/19
I2C Software	Theresa & Jorge			
Get I2C working between two MSP430s		100%	9/15/19	9/18/19
Observe I2C data using a logic analyzer		100%	9/18/19	9/20/19
Get I2C working on FOBOS 3 in Python		0%	12/20/19	1/19/20
Use Python to talk I2C from PYNQ to MSP430, using demo code on MSP430		0%	1/19/20	1/24/20
Look at KiCad FOBOS Shield to figure out I2C pins on PYNQ		0%	1/24/20	1/26/20
I2C Hardware	Vincent & Bradford			
Check if PYNQ has I2C in the PL		100%	9/18/19	9/22/19
Check if I2C IP is available on PYNQ-Z1, Ultra96-V2, Basys3		100%	9/20/19	9/25/19
Use FOBOS2 and FOBOS3 Vivado projects from git and try to add IIC		50%	9/25/19	11/14/19
Create a new Vivado project for the Ultra96-V2 and try to add IIC		0%	11/14/19	11/16/19
Add I2C to FOBOS2 and FOBOS3		0%	11/14/19	11/17/19
FOBOS Shield Circuit Board	Dr. Kaps & Theresa			
Use KiCAD, open FOBOS shield and FOBOS MTC		100%	9/30/19	10/5/19
Determine size of copper area needed to provide adequate cooling variable power		35%	11/24/19	11/28/19
I2C might require change of traces		0%	11/29/19	12/4/19
Side-channel Analysis	Jorge & John			
Read FOBOS Documentation		100%	10/15/19	10/19/19
Arrange meeting with Abubakr for T-Test demo		100%	11/4/19	11/9/19
Install FOBOS on a Linux Machine		50%	11/9/19	11/12/19
Get power traces from Abubakr and run analysis yourself		0%	12/19/19	12/21/19
Work with Abubakr on the FOBOS Tutorials		0%	12/19/19	12/22/19
XXBX Protocol	John & Vincent			
Read XXBX documentation		100%	9/30/19	10/4/19



Display Week: 0

TASK	ASSIGNED TO	PROGRESS	START	END
Install XXBX system on a Linux machine		50%	10/20/19	10/25/19
Run XXBX for an algorithm		0%	10/25/19	10/28/19
Observe XBH-XBD communication with a logic analyzer		0%	12/4/19	12/6/19





Side Channel Analysis on Microcontroller

Team Members:

Theresa Nguyen, Jorge Rodriguez, Vincent Guevara, Bradford Webb, John Lowry

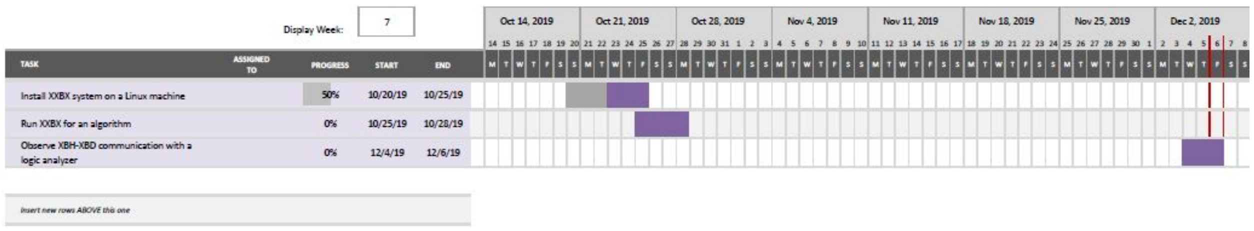
Faculty Supervisor: Dr. Jens-Peter Kaps

Project Start: Sun, 9/1/2019

Display Week: 7

SIMPLE GANTT CHART by Vertex42.com
https://www.vertex42.com/ExcelTemplates/simple-gantt-chart.html

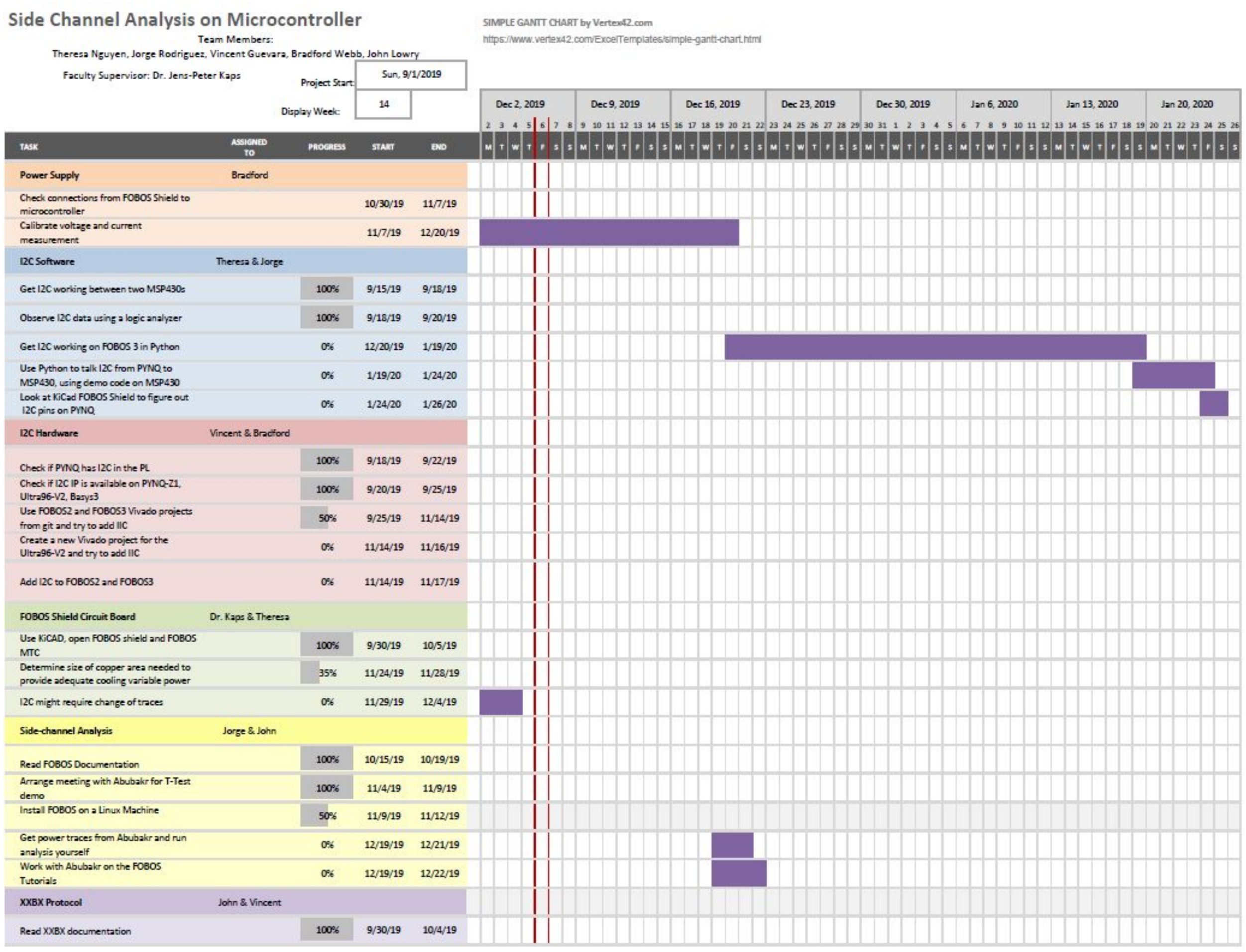
TASK	ASSIGNED TO	PROGRESS	START	END
Power Supply	Bradford			
Check connections from FOBOS Shield to microcontroller			10/30/19	11/7/19
Calibrate voltage and current measurement			11/7/19	12/20/19
I2C Software	Theresa & Jorge			
Get I2C working between two MSP430s		100%	9/15/19	9/18/19
Observe I2C data using a logic analyzer		100%	9/18/19	9/20/19
Get I2C working on FOBOS 3 in Python		0%	12/20/19	1/19/20
Use Python to talk I2C from PYNQ to MSP430, using demo code on MSP430		0%	1/19/20	1/24/20
Look at KiCad FOBOS Shield to figure out I2C pins on PYNQ		0%	1/24/20	1/26/20
I2C Hardware	Vincent & Bradford			
Check if PYNQ has I2C in the PL		100%	9/18/19	9/22/19
Check if I2C IP is available on PYNQ-Z1, Ultra96-V2, Basys3		100%	9/20/19	9/25/19
Use FOBOS2 and FOBOS3 Vivado projects from git and try to add IIC		50%	9/25/19	11/14/19
Create a new Vivado project for the Ultra96-V2 and try to add IIC		0%	11/14/19	11/16/19
Add I2C to FOBOS2 and FOBOS3		0%	11/14/19	11/17/19
FOBOS Shield Circuit Board	Dr. Kaps & Theresa			
Use KiCAD, open FOBOS shield and FOBOS MTC		100%	9/30/19	10/5/19
Determine size of copper area needed to provide adequate cooling variable power		35%	11/24/19	11/28/19
I2C might require change of traces		0%	11/29/19	12/4/19
Side-channel Analysis	Jorge & John			
Read FOBOS Documentation		100%	10/15/19	10/19/19
Arrange meeting with Abubakr for T-Test demo		100%	11/4/19	11/9/19
Install FOBOS on a Linux Machine		50%	11/9/19	11/12/19
Get power traces from Abubakr and run analysis yourself		0%	12/19/19	12/21/19
Work with Abubakr on the FOBOS Tutorials		0%	12/19/19	12/22/19
XXBX Protocol	John & Vincent			
Read XXBX documentation		100%	9/30/19	10/4/19



Display Week: 7

TASK	ASSIGNED TO	PROGRESS	START	END
Install XXBX system on a Linux machine		50%	10/20/19	10/25/19
Run XXBX for an algorithm		0%	10/25/19	10/28/19
Observe XBH-XBD communication with a logic analyzer		0%	12/4/19	12/6/19

Insert new rows ABOVE this one


Side Channel Analysis on Microcontroller

Team Members:

Theresa Nguyen, Jorge Rodriguez, Vincent Guevara, Bradford Webb, John Lowry

Faculty Supervisor: Dr. Jens-Peter Kaps

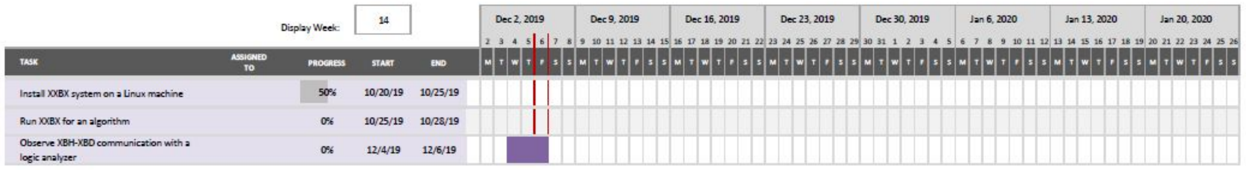
Project Start: Sun, 9/1/2019

Display Week: 14

SIMPLE GANTT CHART by Vertex42.com

https://www.vertex42.com/ExcelTemplates/simple-gantt-chart.html

TASK	ASSIGNED TO	PROGRESS	START	END
Power Supply	Bradford			
Check connections from FOBOS Shield to microcontroller			10/30/19	11/7/19
Calibrate voltage and current measurement			11/7/19	12/20/19
I2C Software	Theresa & Jorge			
Get I2C working between two MSP430s		100%	9/15/19	9/18/19
Observe I2C data using a logic analyzer		100%	9/18/19	9/20/19
Get I2C working on FOBOS 3 in Python		0%	12/20/19	1/19/20
Use Python to talk I2C from PYNQ to MSP430, using demo code on MSP430		0%	1/19/20	1/24/20
Look at KiCad FOBOS Shield to figure out I2C pins on PYNQ		0%	1/24/20	1/26/20
I2C Hardware	Vincent & Bradford			
Check if PYNQ has I2C in the PL		100%	9/18/19	9/22/19
Check if I2C IP is available on PYNQ-Z1, Ultra96-V2, Basys3		100%	9/20/19	9/25/19
Use FOBOS2 and FOBOS3 Vivado projects from git and try to add IIC		50%	9/25/19	11/14/19
Create a new Vivado project for the Ultra96-V2 and try to add IIC		0%	11/14/19	11/16/19
Add I2C to FOBOS2 and FOBOS3		0%	11/14/19	11/17/19
FOBOS Shield Circuit Board	Dr. Kaps & Theresa			
Use KiCAD, open FOBOS shield and FOBOS MTC		100%	9/30/19	10/5/19
Determine size of copper area needed to provide adequate cooling variable power		35%	11/24/19	11/28/19
I2C might require change of traces		0%	11/29/19	12/4/19
Side-channel Analysis	Jorge & John			
Read FOBOS Documentation		100%	10/15/19	10/19/19
Arrange meeting with Abubakr for T-Test demo		100%	11/4/19	11/9/19
Install FOBOS on a Linux Machine		50%	11/9/19	11/12/19
Get power traces from Abubakr and run analysis yourself		0%	12/19/19	12/21/19
Work with Abubakr on the FOBOS Tutorials		0%	12/19/19	12/22/19
XXBX Protocol	John & Vincent			
Read XXBX documentation		100%	9/30/19	10/4/19


Display Week: 14

TASK	ASSIGNED TO	PROGRESS	START	END
Install XXBX system on a Linux machine		50%	10/20/19	10/25/19
Run XXBX for an algorithm		0%	10/25/19	10/28/19
Observe XBH-XBD communication with a logic analyzer		0%	12/4/19	12/6/19

References

[1] J. R. Bochert. “Software Protection Against Side Channel Analysis Through a Hardware Level Power Difference Eliminating Mask,” in Journal of Cyber Security and Information Systems. February 2, 2016. [Online] Accessed on December 5, 2019. https://www.csiac.org/journal-article/software-protection-against-side-channel-analysis-through-a-hardware-level-power-difference-eliminating-mask/

[2] Ruxton, G. D. (2006). "The unequal variance t-test is an underused alternative to Student's t-test and the Mann–Whitney U test". *Behavioral Ecology*. 17: 688–690. doi:10.1093/beheco/ark016.

[3] Mangard, Oswald, Popp. “Power Analysis Attacks - Revealing the secrets of Smart Cards” Accessed on December 5, 2019.

[4] Analog Devices, 10-Bit, 65/80/105 MSPS, 3 V A/D Converter , AD9215 datasheet, 2013. Accessed on December 6, 2019.

[5] “TPS7A7100 datasheet,” TPS7A7100 datasheet | TI.com. [Online]. Available: http://www.ti.com/document-viewer/TPS7A7100/datasheet. [Accessed: 07-Dec-2019].