# CSE 237D - Final Report
# SECURITY FOR MULTI-TENANT FPGA

| | |
|---|---|
| Elakkia Kanaka Parthipan | A53313335 |
| Aniket Tiwari | A53318400 |
| Dhananjay Jagtap | A53287122 |

## Table of Contents

## 1. Abstract

Field Programmable Gate Arrays (FPGAs) are excellent resources to target and optimise a specific algorithm. They can achieve massive parallelism and thus popular choices for today's data intensive computing. Hosting multiple users on an FPGA on the cloud is economical when one doesn't require the entire FPGA. However, this might lead to security issues, as only the programmable logic on an FPGA is separated, whereas the Power Delivery Network(PDN) is common. The power consumption traces obtained from a voltage sensor can be used to determine whether a user is taking advantage of the PDN and sabotaging the work of the other users on the board. A robust methodology for analysing and characterising the activity on the FPGA has been developed.

## 2. Introduction

Field Programmable Gate Arrays (FPGAs) can be configured to perform the functionality of any hardware that we require. The array of gates that make up an FPGA can be programmed to run a specific algorithm, using the combination of logic gates (usually implemented as lookup tables), arithmetic units, digital signal processors (DSPs) to do multiplication, static RAM for temporarily storing the results of those computation and switching blocks that let you control the connections between the programmable blocks. Some FPGAs are essentially systems-on-a-chip (SoC), with CPUs, PCI Express and DMA connections and Ethernet controllers, turning the programmable array into a custom accelerator for the code running on the CPU. The combination means that FPGAs can offer massive parallelism targeted only for a specific algorithm, and at much lower power compared to a GPU. And unlike an application-specific integrated circuit (ASIC), they can be reprogrammed when you want to change that algorithm (that's the field-programmable part).

All this makes an FPGA extremely attractive for today's data intensive computing and machine learning models. Companies such as Amazon Web Services, Microsoft etc offer FPGA as a service to anyone who wants to use it. Since the FPGAs are large, and a single user may not require the entire board, several users can be hosted on a single FPGA board. On the FPGA, the programmable logic is separated, thus the hardware built by the different users and the functionality of each of those modules will be independent of each other. However, the Power Delivery Network in the FPGA is common for all the programmable logic and it can be taken advantage of.

Simple circuits called power wasters can be built from the programmable logic on the

FPGA and they may seem harmless. However, they are capable of drawing all the power from the PDN, thereby not leaving power for the rest of the programmable logic on the board. This can disrupt the activity of the other users on the board leading to security issues. In-order to understand whether any circuit is acting with malicious intent, Professor Kastner's Research Group has designed a voltage sensor which consists of Carry Chain Adders. These voltage sensors will be integrated with every hardware core implemented on the FPGA. Changes in the delay of the Carry Chain Adders can be used to characterize the activity of the different users on the FPGA Board.

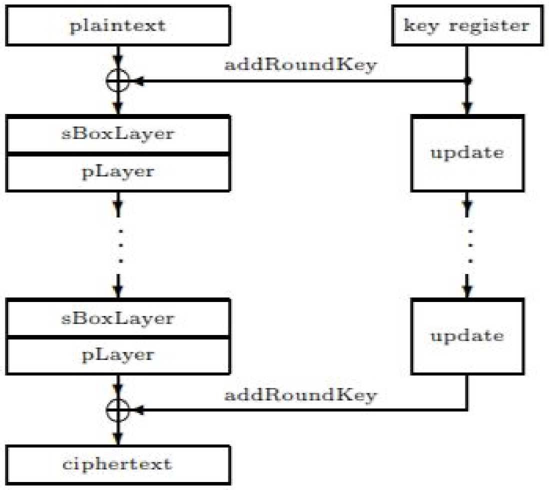
In-order to mimic the existence of several tenants on the FPGA Board, Professor Kastner's Research Group has implemented a Advanced Encryption Standard (AES) Soft core and a HLS(High Level Synthesis) core. PRESENT soft core and power waster on the FPGA. In our project, we will be adding one more tenant to the FPGA, namely the PRESENT HLS Core. This core was built, implemented on the Xilinx PYNQ Z2 Board and integrated with the Voltage Sensor containing the Carry Chain Adder.

The traces of the Carry Chain Adder from the already existing cores were collected. Then the traces were processed and provided as input to RESNET, which is a CNN Classifier in-order to get information about the activity on the FPGA board.

## 3. PRESENT CRYPTOGRAPHIC ALGORITHM

**PRESENT** is a lightweight block cipher, developed by the Orange Labs (France), Ruhr University Bochum (Germany) and the Technical University of Denmark in 2007. PRESENT is designed by Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. The algorithm is known for its extreme light-weightedness. It is 2.5 times smaller than AES.

```
generateRoundKeys()
for $i = 1$ to 31 do
    addRoundKey(STATE,$K_i$)
    sBoxLayer(STATE)
    pLayer(STATE)
end for
addRoundKey(STATE,$K_{32}$)
```



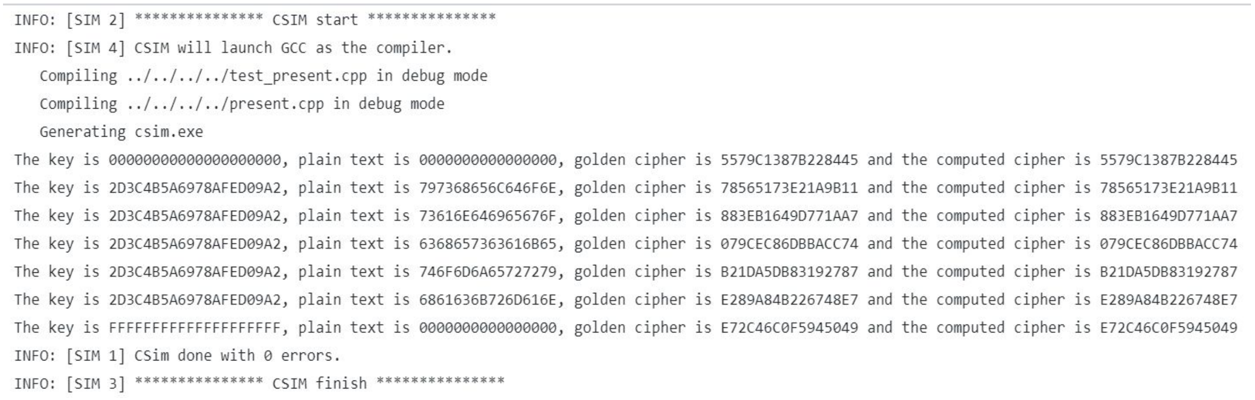
The PRESENT Cryptographic Algorithm has only substitution and permutation layers. These substitution and permutations are nothing but switching wires as and when necessary and thus, this algorithm is extremely light weight without any heavy computation.
The block size can be 64 bits and the key can be 80 or 120 bits. For the purpose of this project, a key of 80 bits was used. PRESENT is intended to be used in situations where lower power consumption and high chip efficiency is desired.

## 3.1 Python Implementation of the PRESENT Cryptographic Algorithm

The PRESENT Cryptographic algorithm was first implemented on python. This was done in-order to collect test input and output data for the HLS core to be built. The input and output generated from the python model will be used as the golden data to test whether the PRESENT HLS Core works as intended or not.
A few results from the paper [1] was used to test whether the present python implementation works as per the specification in the paper. The test vectors used for testing as shown below.

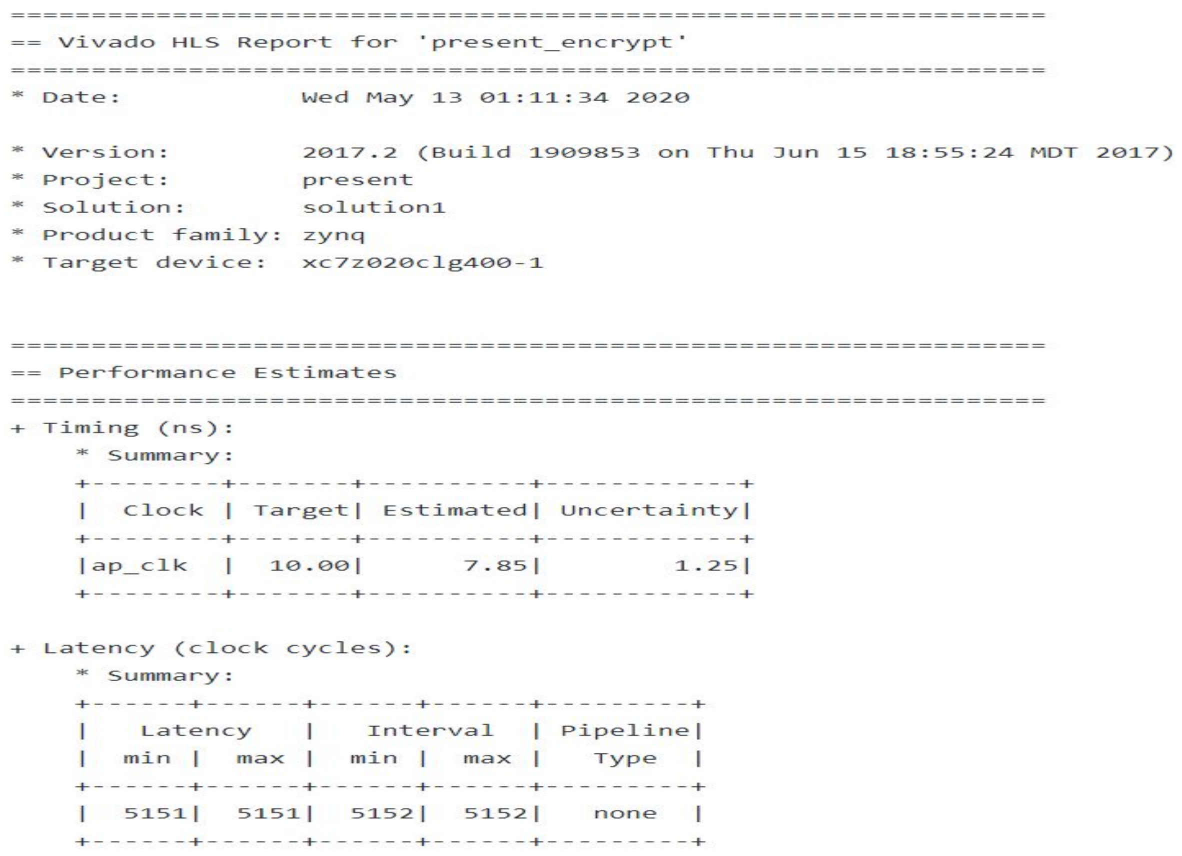
## 3.2 PRESENT HLS CORE

Once the golden input data was generated, the PRESENT Cryptographic core was built using C on Xilinx Vivado HLS. The Core takes in a 64 bit block message, 80 bit key and gives out a 64 bit cipher (encrypted message).
The simulation results depicting the correctness of the core are shown below:



```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../test_present.cpp in debug mode
   Compiling ../../../../present.cpp in debug mode
   Generating csim.exe
The key is 00000000000000000000, plain text is 0000000000000000, golden cipher is 5579C1387B228445 and the computed cipher is 5579C1387B228445
The key is 2D3C4B5A6978AFED09A2, plain text is 797368656C646F6E, golden cipher is 78565173E21A9B11 and the computed cipher is 78565173E21A9B11
The key is 2D3C4B5A6978AFED09A2, plain text is 73616E646965676F, golden cipher is 883EB1649D771AA7 and the computed cipher is 883EB1649D771AA7
The key is 2D3C4B5A6978AFED09A2, plain text is 6368657363616B65, golden cipher is 079CEC86DBBACC74 and the computed cipher is 079CEC86DBBACC74
The key is 2D3C4B5A6978AFED09A2, plain text is 746F6D6A65727279, golden cipher is B21DA5DB83192787 and the computed cipher is B21DA5DB83192787
The key is 2D3C4B5A6978AFED09A2, plain text is 6861636B726D616E, golden cipher is E289A84B226748E7 and the computed cipher is E289A84B226748E7
The key is FFFFFFFFFFFFFFFFFFFF, plain text is 0000000000000000, golden cipher is E72C46C0F5945049 and the computed cipher is E72C46C0F5945049
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

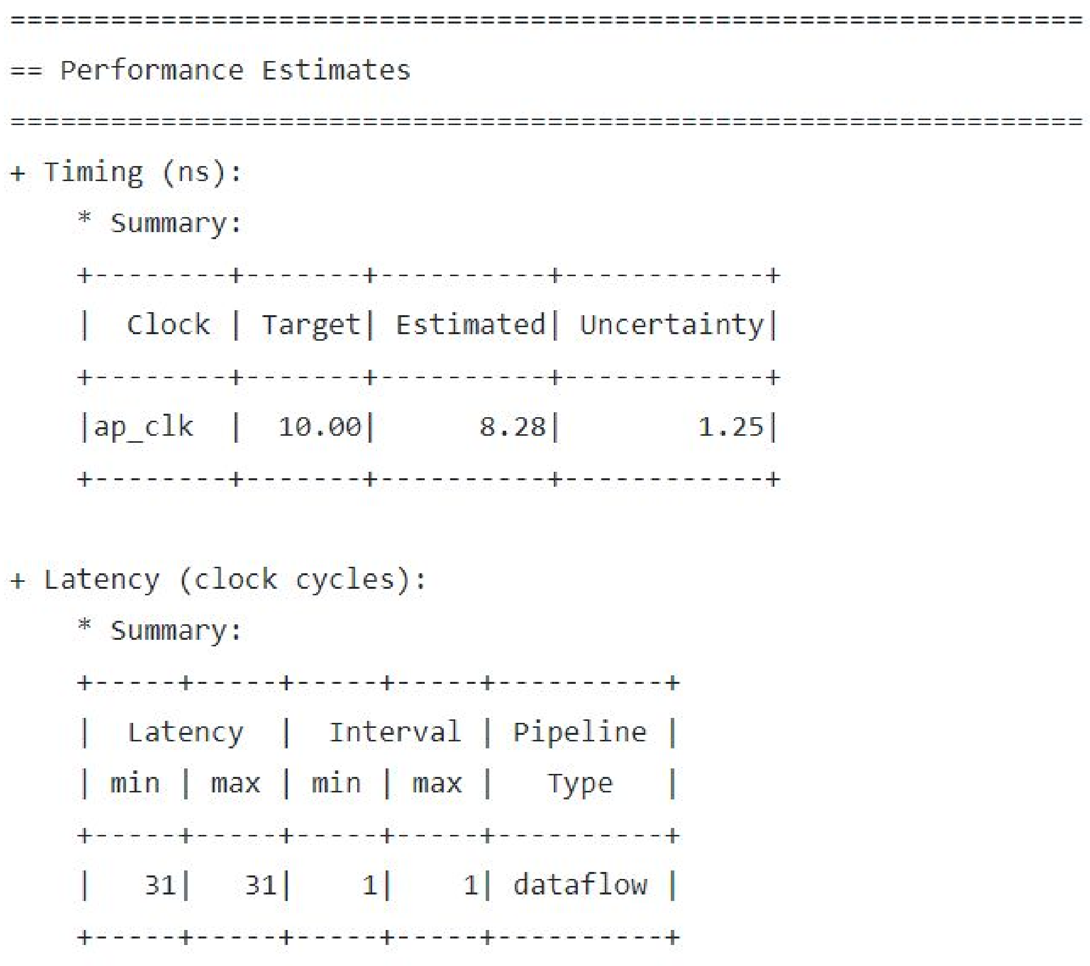
The base design was at a speed of 194 KHz. The throughput is calculated at clock period / latency (number of cycles)



```
================================================================
== Vivado HLS Report for 'present_encrypt'
================================================================
* Date:           Wed May 13 01:11:34 2020

* Version:        2017.2 (Build 1909853 on Thu Jun 15 18:55:24 MDT 2017)
* Project:        present
* Solution:       solution1
* Product family: zynq
* Target device:  xc7z020clg400-1


================================================================
== Performance Estimates
================================================================
+ Timing (ns):
    * Summary:
    +--------+-------+----------+------------+
    |  Clock | Target| Estimated| Uncertainty|
    +--------+-------+----------+------------+
    |ap_clk  |  10.00|      7.85|        1.25|
    +--------+-------+----------+------------+

+ Latency (clock cycles):
    * Summary:
    +------+------+------+------+---------+
    |   Latency   |   Interval  | Pipeline|
    |  min |  max |  min |  max |   Type  |
    +------+------+------+------+---------+
    |  5151|  5151|  5152|  5152|   none  |
    +------+------+------+------+---------+
```

The PRESENT Core was then accelerated to a speed of 100 MHz. This was done using the Dataflow pragma of Vivado HLS. Pragmas such as array partitioning, loop unrolling and pipelining were also used to achieve this speed. Pragmas in Vivado HLS as compiler directives which automate the process of performing certain optimisation techniques for faster and compact hardware. For example, the loop unrolling pragma, unrolls all the statements inside a loop and tries to execute them all at once.

When the dataflow pragma is used, the throughput is calculated as clock period / interval (number of cycles)

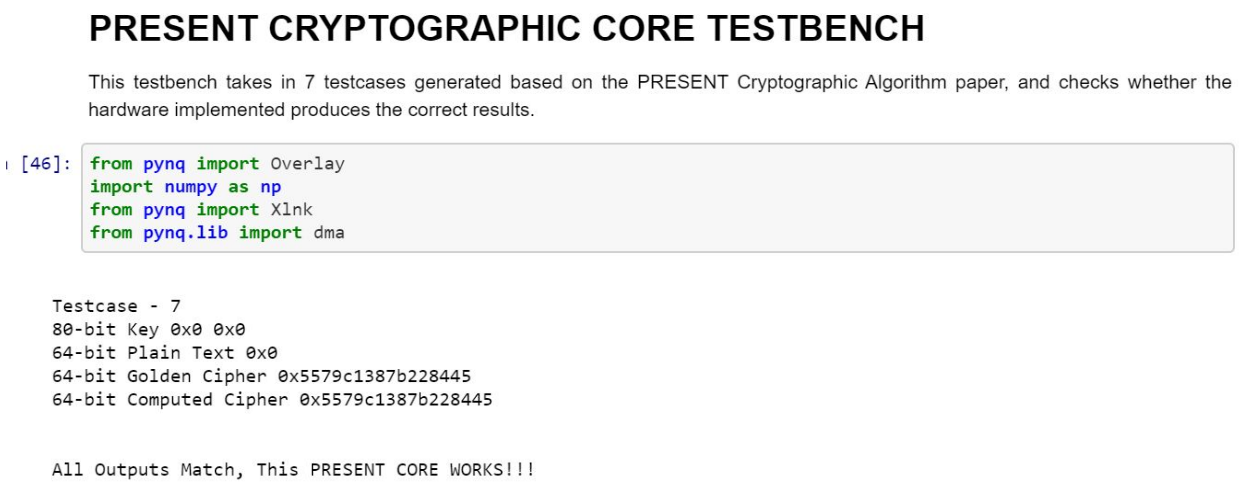


```
================================================================
== Performance Estimates
================================================================
+ Timing (ns):
    * Summary:
    +--------+-------+----------+------------+
    |  Clock | Target| Estimated| Uncertainty|
    +--------+-------+----------+------------+
    |ap_clk  |  10.00|      8.28|        1.25|
    +--------+-------+----------+------------+

+ Latency (clock cycles):
    * Summary:
    +-----+-----+-----+-----+----------+
    |  Latency  |  Interval | Pipeline |
    | min | max | min | max |   Type   |
    +-----+-----+-----+-----+----------+
    |   31|   31|    1|    1| dataflow |
    +-----+-----+-----+-----+----------+
```

## 3.3 PRESENT HLS CORE IMPLEMENTATION

The PRESENT Cryptographic HLS Core built was implemented on the Xilinx Pynq Z2 Board. PYNQ is an open-source project from Xilinx that makes it easier to use Xilinx platforms. Using the Python language and libraries, designers can exploit the benefits of programmable logic and microprocessors to build more capable and exciting electronic systems. PYNQ can be used with *Zynq*, *Zynq UltraScale+*, *Zynq RFSoC*, *Alveo* accelerator boards and AWS-F1 to create high performance applications. A PYNQ enabled board can be programmed using Jupyter Notebook in Python.
While implementing the PRESENT core, we make use of **AXI-4 Stream Interconnect** to pass the input to the core and get the output back. AXI4-Stream Interconnect core provides the capability to connect multiple master/slave AMBA AXI4-Stream protocol compliant endpoint IP.

The following figures depicts that the PRESENT Core works correctly when implemented on the FPGA.



PRESENT CRYPTOGRAPHIC CORE TESTBENCH

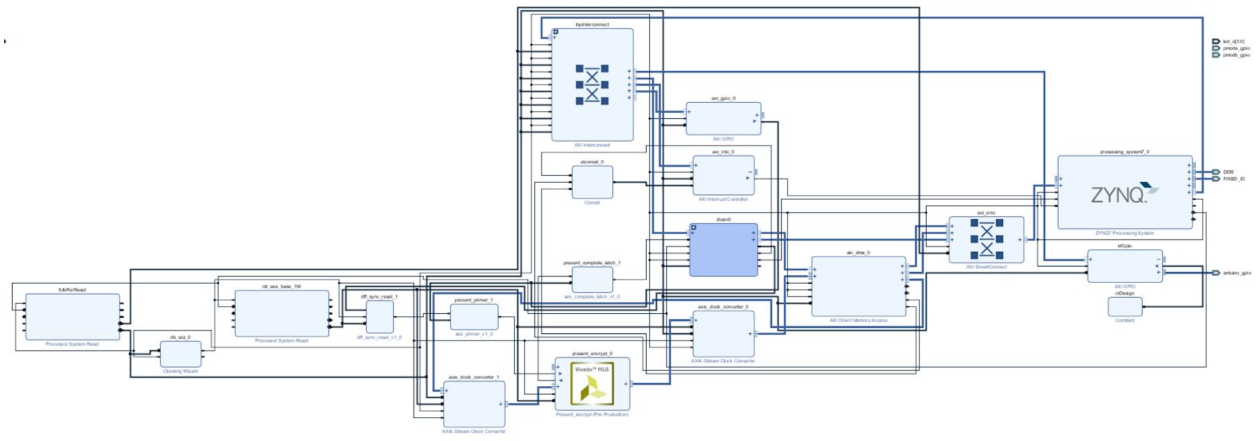
This testbench takes in 7 testcases generated based on the PRESENT Cryptographic Algorithm paper, and checks whether the hardware implemented produces the correct results.

```
[46]: from pynq import Overlay
      import numpy as np
      from pynq import Xlnk
      from pynq.lib import dma
```

```
Testcase - 7
80-bit Key 0x0 0x0
64-bit Plain Text 0x0
64-bit Golden Cipher 0x5579c1387b228445
64-bit Computed Cipher 0x5579c1387b228445


All Outputs Match, This PRESENT CORE WORKS!!!
```

## 3.4 INTEGRATION WITH VOLTAGE SENSOR

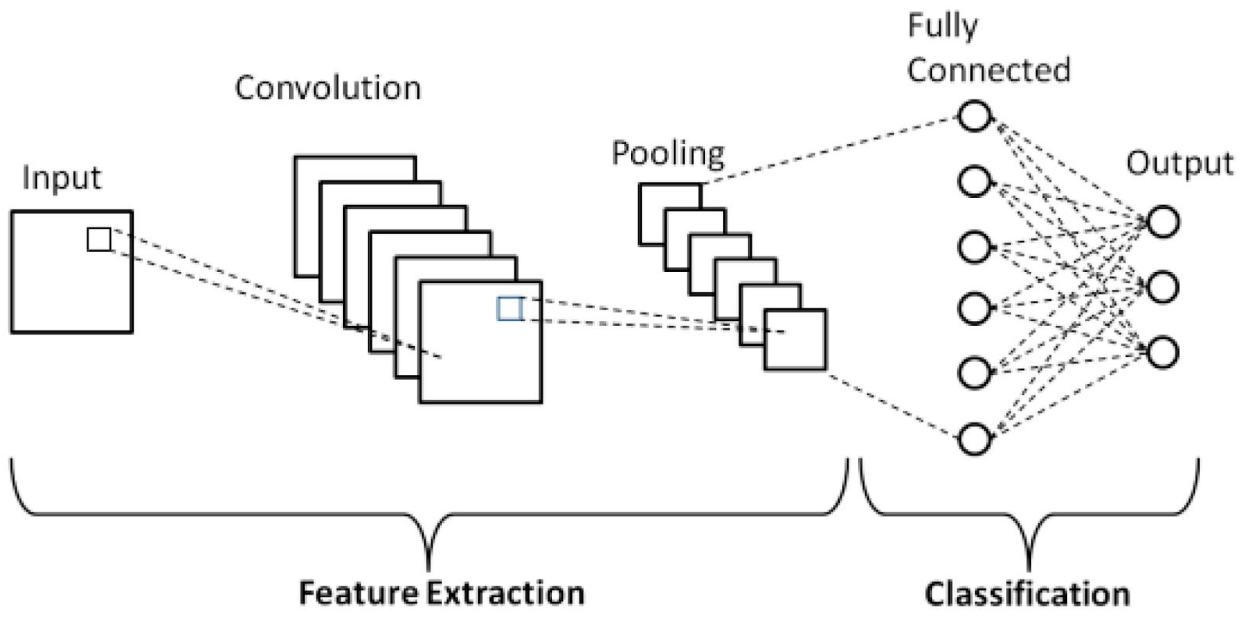
The PRESENT Cryptographic Core built has been integrated with the Voltage Sensor built by Professor Kastner’s Research Group. Integrating the core with the sensor ensures that the sensor is stabilised before starting the core and collecting the traces. The clock given to the sensor is the speed at which the core will also be run at, in-order to ensure that we can collect the traces for power consumption without any discrepancies. This implementation also uses the AXI-4 Stream Interconnect for Direct Memory Access.



We are in the process of collecting the traces and running into some issues regarding that. We also need to collect sufficient data in-order to feed it to the classifier.

## 4. CNN CLASSIFIER

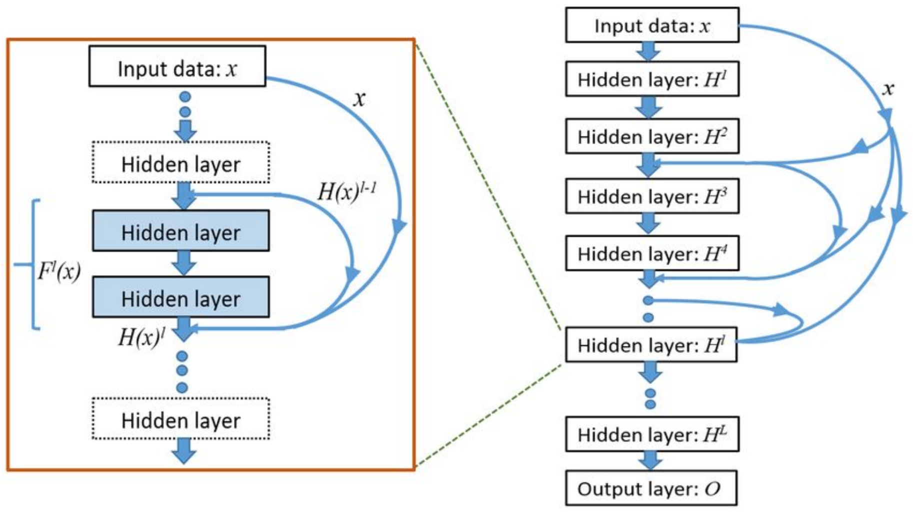
Convolutional Neural Networks are commonly used Machine learning Techniques that help in classifying the labeled data into given classes. The following diagram shows an example of the same.



We shall be using Resnet for our CNN here. The salient features of Resnet are:

1. The Resnet has Batch normalisation at its core thereby making it resistant to Overfitting
2. The ResNet makes use of the Identity Connection, which helps to protect the network from vanishing gradient problem.
3. Deep residual Network increases the accuracy of model.

Here is the diagram for a Resnet.



We have used Resnet with 18 layers for our purpose here. The output of Resnet is then given to a linear layer to distribute into various classes.

## 4.1 Characteristics of voltage signal measurements

The first step in designing the classifier is to decide what inputs should be given to the classifier. These inputs will determine the feasibility of the classification in the first place. In order to do so the inputs should be such that they can accurately characterise the core which has generated them. Generally the characteristic features of a signal are embedded in the frequency component of the signal. Thus, Fourier analysis is a preferred method to draw out these inherent frequency characteristics. But a global Fourier transform is only accurate as long as the frequency components do not change with time which is the case with Stationary signals. However, this global frequency transform gives us the presence of all frequency components within a signal or a data set without giving us any temporal/time information. This becomes a problem especially in our case when the signal is non-stationary and varying in length. This is because the cores are running at different time instances and the run times for each of them is different as well. Thus, the frequency response would be different for different frames of observation.
This can be solved by using a windowed fourier transform/ analysis which will be able to give us good localization in both frequency as well as time domain.

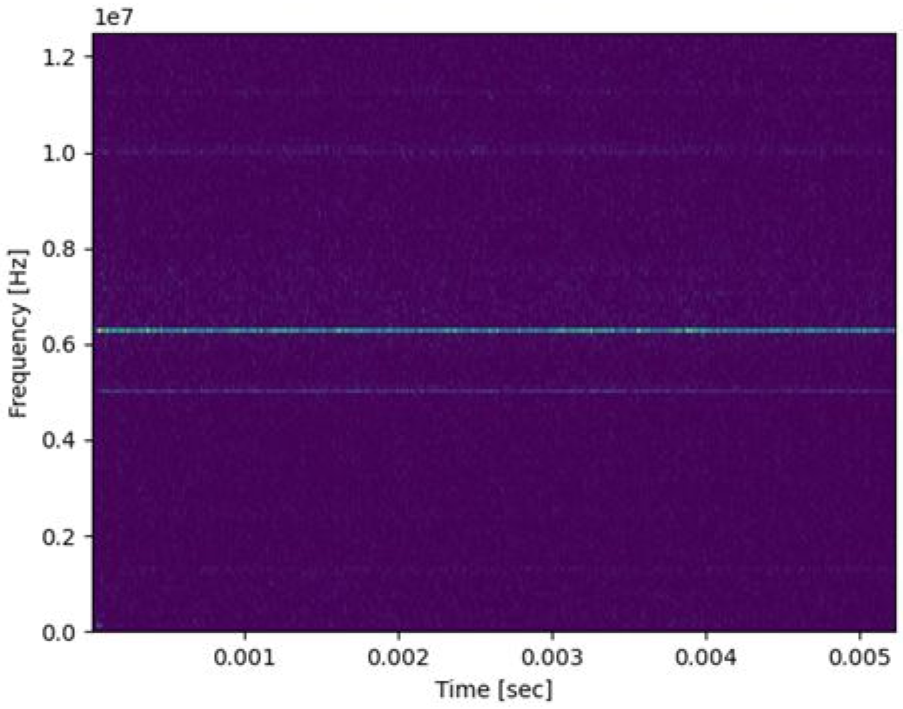
## 4.2 Techniques for analysing the voltage traces

Based on our discussion above we came to a conclusion that  windowed fourier transform techniques should be analysed based on that there are two approaches that we looked at they were

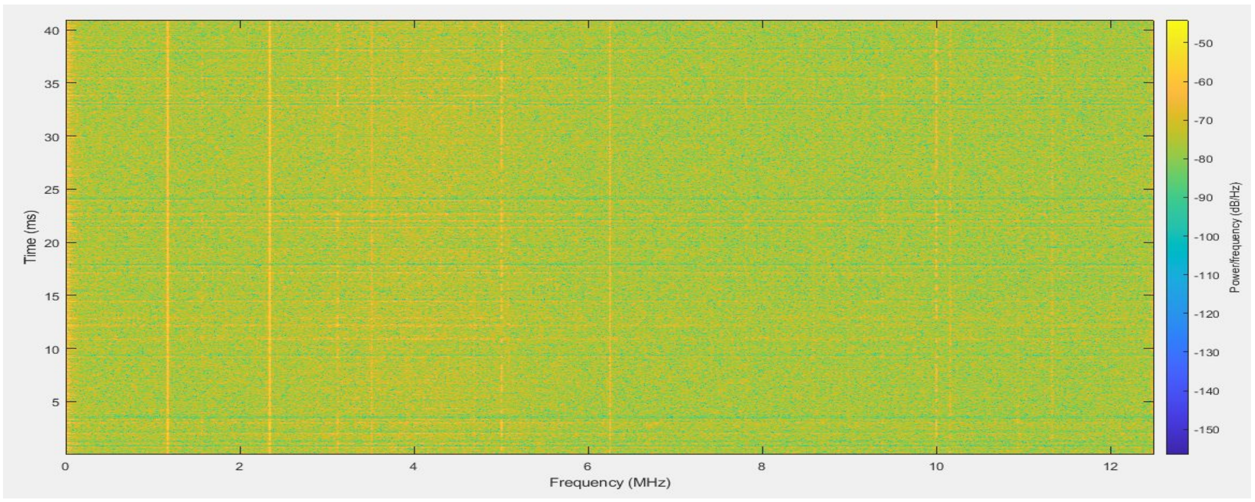
1) Spectrograms (Short Time Fourier Transforms)
2) Continuous Wavelet Transforms

### 4.2.1 Spectrograms

Short Time Fourier Transforms are basically fourier transforms carried out over short periods of time of a signal. This means that if the frequency component changes during the time period it can be observed using a STFT. When this STFT is represented in a visual format with each colour representing the intensity of a frequency component as it varies with time it is called a Spectrogram. The Spectrogram/STFT has a fixed window length over the entire signal. This in turn limits the resolution with which a frequency signal can be identified to the size of the window which is the time period selected. The Spectrogram only gives us the magnitude information about the frequency components. Also it is proportional to the square of the magnitude of the signal present.



To actually know whether Spectrograms truly give us the characteristic information of the core we created a spectrogram of AES_Baseline data in MALTAB by using a window size of 2048 and an overlap of 128. This gave us the following plot. The plot gave us peaks at the expected frequencies based on the frequency of operation of the power supply, the core and the sampling technique. This reconfirmed the fact that Spectrograms can be a valid approach for analysis.



Spectrogram of AES_Baseline data

We noticed that our signal had the following properties(As mentioned above):

- Varying data length
- Non stationary signal

To account for these properties, we take the spectrogram for a fixed time period of all applications(irrespective of the application being run)
These spectrograms inherently tackle the non-stationary problem and taking the time period for a fixed time period will help us in getting the uniformity in order to ensure that the test data can be replicated in the real life setting.

### 4.2.2 Continuous Wavelet Transform

Spectrogram has a fixed value for time and frequency resolution. The resolution is fixed with both time and frequency. To bring more control for the same, we used Continuous Wavelet transform(CWT). This is also called travelling wavelet transform. The properties and configuration for the wavelet transform was as follows:

- Wavelet: Gauss Wavelet
- Frequency scaling values: 5
- Library used : pywt

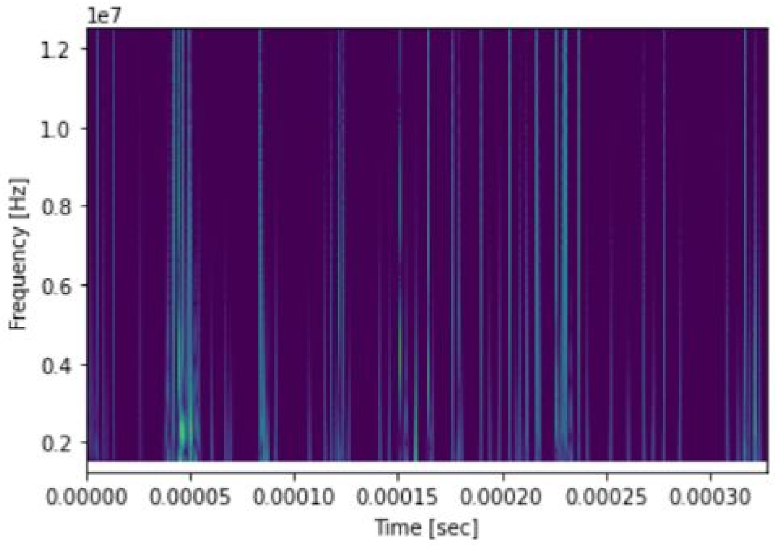
The approach had the following Pros:

- The frequency resolution was high for most of the values.

The Cons were:

- It introduced a lot of noise in data thereby making the classification difficult.
- The Data generation step itself was taking significantly more time.

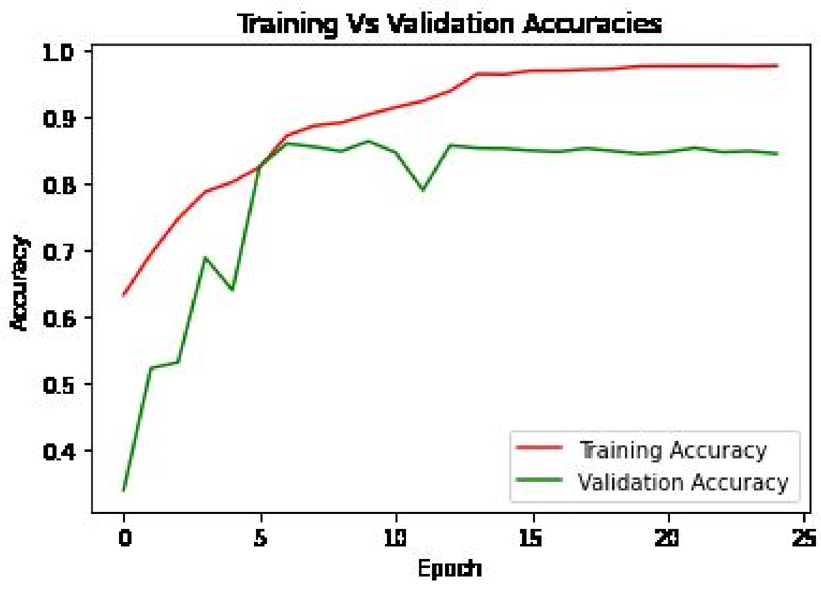
The following is an example of CWT transform:



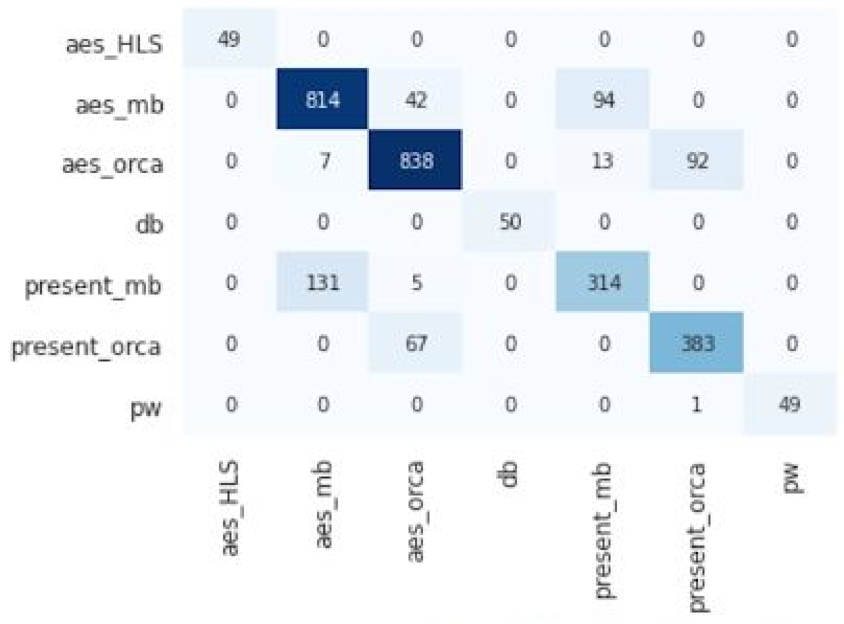
### 4.3 Classification Results

We ran our classifier using Google Colab owing to the high computational capabilities of the same. The following are the training results for the model.
We ran our classifier for the following programs:

- AES implementation for HLS (aes_hls)
- A baseline value when nothing is running for acting as baseline(db)
- PRESENT encryption algorithm for microblaze soft processor(present_mb).
- AES encryption implemented on microblaze soft processors(aes_mb).
- PRESENT encryption algorithm for Orca soft processor(present_orca).
- AES encryption implemented on Orca soft processors(aes_orca).



The following is the diagram for the confusion matrix formed for the



| | aes_HLS | aes_mb | aes_orca | db | present_mb | present_orca | pw |
|---|---|---|---|---|---|---|---|
| aes_HLS | 49 | 0 | 0 | 0 | 0 | 0 | 0 |
| aes_mb | 0 | 814 | 42 | 0 | 94 | 0 | 0 |
| aes_orca | 0 | 7 | 838 | 0 | 13 | 92 | 0 |
| db | 0 | 0 | 0 | 50 | 0 | 0 | 0 |
| present_mb | 0 | 131 | 5 | 0 | 314 | 0 | 0 |
| present_orca | 0 | 0 | 67 | 0 | 0 | 383 | 0 |
| pw | 0 | 0 | 0 | 0 | 0 | 1 | 49 |

We noticed that the present_mb accuracy is very less because:

- The data has not been cleaned for present_mb(i.e. A lot of redundant data is encountered in it)
- The microblaze implementation for aes and present show similar characteristics.

We hope to expand the scope of the model in order to include more codes to differentiate from.However, the above work acts as a good proof of concept for the achievability of a classifier that shall be able to do the classification.

## 5. MILESTONES

Elakkia Kanaka Parthipan A53313335

1. Present HLS Cryptographic Core - A-
2. Implement Present HLS Cryptographic Core on Pynq - A
3. Add the PL Sensor to the Core and get traces out for analysis - A+

Milestones 1 and 2 have been completed and Milestone 3 is 80% complete. It will be completed in the near future. A possible reason for the little delay in completion of milestone 3 could be the underestimation of the difficulty and time taken to collect the traces.

## 6. CONCLUSION

In conclusion, we were able to build the PRESENT Cryptographic HLS core, implement it on the Xilinx Pynq Z2 board and integrate it with the Voltage Sensor. We plan to collect the traces and classify the PRESENT HLS core traces as well in the near future.

## 7.REFERENCES

[1] PRESENT: An Ultra-Lightweight Block Cipher by A. Bogdanov1 , L.R. Knudsen2 , G. Leander1 , C. Paar1 , A. Poschmann1 , M.J.B. Robshaw3 , Y. Seurin3 , and C. Vikkelsoe2