Reverse Engineering of Integrated Circuits: Tools and Techniques

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

By

Abhijitt Dhavlle
Bachelor of Engineering
Mumbai University, 2014

Director: Dr. Sai Manoj PD, Professor
Department of Electrical and Computer Engineering

Spring Semester 2022
George Mason University
Fairfax, VA

Dedication

I sincerely dedicate this thesis to my beloved Jagannath, Baladev, Subhadra and Srila Prabhupada. I dedicate the thesis to my loving parents, Additi and Avinash Dhavale, my brother Aniruddha, and my loving and supportive wife Mrunali; and to my kind and loving in-laws, Akshay, Sitarani Devi and Sudhakar Bhosale. Without their motivation and nurture it would have been impossible to keep treading in testing times. This cannot be complete without crediting my grandfather, Ambadas Dhavale, and grandmother, Kamal Dhavale. I also would like to thank my friends, Sanket, Gaurav, Manideep and many others for their constant support and encouragement. I cannot forget to express my deep gratitude to my mentors, Sandhini, Mahamantra, Narsimhalila , and Kanhaigopal for their loving care and best wishes. I want to deeply appreciate Rajeev and Vidula Saawant, Priya and Sharad Baagwe, Sachin Solase, my cousins, and all the relatives for their care and blessings. I extend my gratitude to everyone who has been instrumental in the success so far and much more to come.

Acknowledgments

I would like to thank the following special people who made this seemingly long journey possible. Dr. Sai Manoj PD, my Masters and PhD advisor for his kind support always in trying to make me a better version of myself. I have learned from him many qualities which are conducive to being an enthusiastic and sincere student. I would like to thank Dr. Jim Jones, Dr. Brian Mark, Dr. Khaled Khasawneh, Dr. Amlan Ganguly, Dr. Avesta Sasan, Dr. Aydin, Dr. Liling Huang, Dr. Houman Homayoun, Dr. Setareh Rafatirad, Ms. Jammie Chang, Ms. Patricia Sahs; Mr. Martin Voogel, Dr. Pierre Maillard and Dr. Paula Chen from Xilinx Inc.; Dr. Andrew Schmidt from USCISI, Ms. Anagha Malkapurkar from ACPCOE; Dr. Umesh Mhapankar, Mrs. Sonali Sherigar, Mrs. Raji MP from Fr. Agnels Polytechnic; and Mrs. Radhakrishnan from MSAS school, for being instrumental in their capacity as my advisors, professors, collaborators, trainers, teachers and much more.

Table of Contents

List of Tables

List of Figures

Listings

Abstract

REVERSE ENGINEERING OF INTEGRATED CIRCUITS: TOOLS AND TECHNIQUES

Abhijitt Dhavlle

George Mason University, 2022

Thesis Director: Dr. Sai Manoj PD

Consumer and defense systems demanded design and manufacturing of electronics with increased performance, compared to their predecessors. As such systems became ubiquitous in a plethora of domains, their application surface increased, thus making them a target for adversaries. Hence, with improved performance the aspect of security demanded even more attention of the designers. The research community is rife with extensive details of attacks that target the confidential design details by exploiting vulnerabilities. The adversary could target the physical design of a semiconductor chip or break a cryptographic algorithm by extracting the secret keys, using attacks that will be discussed in this thesis. This thesis focuses on presenting a brief overview of IC reverse engineering attack and attacks targeting cryptographic systems. Further, the thesis presents my contributions to the defenses for the discussed attacks.

The globalization of the Integrated Circuit (IC) supply chain has rendered the advantage of low-cost and high performance ICs in the market for the end users. But this has also made the design vulnerable to over production, IP Piracy, reverse engineering attacks and hardware malware during the manufacturing and post manufacturing process. Logic

locking schemes have been proposed in the past to overcome the design trust issues but the new state-of-the-art attacks such as SAT has proven a larger threat. This work highlights the reverse engineering attack and a proposed hardened platform along with its framework.

On the other side, the side-channel attacks (SCAs) has been one of the emerging threats. These SCAs function by exploiting the side-channels which invariably leak important data during an application's execution. The information leaked through side-channels are inherent characteristics of the system and is often unintentional. This information can be microarchitectural or physical information such as power consumption, thermal maps, timing of the operation, acoustics, and cache-trace. Intercepting secret information based on the study of power signature is a subdivision of SCAs where power consumption information serves as a covert channel leaking crucial information about the executed operations. Such physical SCAs are known to be a significant threat to cryptosystems such as AES (Advanced Encryption Standard) and can reveal the encryption key efficiently. To overcome such concerns and protect the data integrity, I introduce *Power Swapper* in this work. The proposed Power Swapper thwarts the attack by randomly choosing one of the multiple modules that perform the intended activity, but have power signature different than a standard implementation and can lead to similar power consumption as one of the other modules that perform a different operation. To achieve this, I introduce carefully crafted swapping of the standby modules that are responsible for the AES operation thus deluding the attacker without hurting the crypto operation. This methodology has been validated for the AES power analysis attack and the key information observed by the attacker is seen to be incorrect, indicating the success of the proposed method.

Chapter 1: Introduction

1.1 Introduction to Hardware Security

Consumer and defense systems demanded design and manufacturing of electronics with increased performance, compared to their predecessors [3]. As such systems became ubiquitous in a plethora of domains, their application surface increased, thus making them a target for adversaries. Hence, with improved performance the aspect of security demanded even more attention of the designers. Confidentiality, Integrity and Availability are the major building blocks of the state-of-the-art systems. The confidentiality refers to the design confidentiality; and integrity refers to the design integrity where an adversary should not make malicious modifications to the design; whereas, availability refers to the system functioning as intended by the designer. Hence, to maintain these tripod principles, hardware security, a collection of various defense methodologies, emerged. The research community is rife with extensive details of attacks that target the confidentiality and integrity of the device. The adversary could target the physical design of a semiconductor chip or break a cryptographic algorithm by extracting the secret keys, using attacks that will be mentioned shortly. This thesis focuses on presenting a brief overview of IC counterfeiting attacks and attacks targeting cryptographic systems. Further, the thesis presents my contributions to the defenses for the aforementioned attacks. Next, I will be presenting an introduction to IC reverse engineering (RE) attacks and the physical side-channel attacks (SCA).

1.1.1 IC Reverse Engineering Attacks

IC manufacturing with millions and billions of transistors has become the need of the hour to support large consumer market that constantly demands increasing performance and

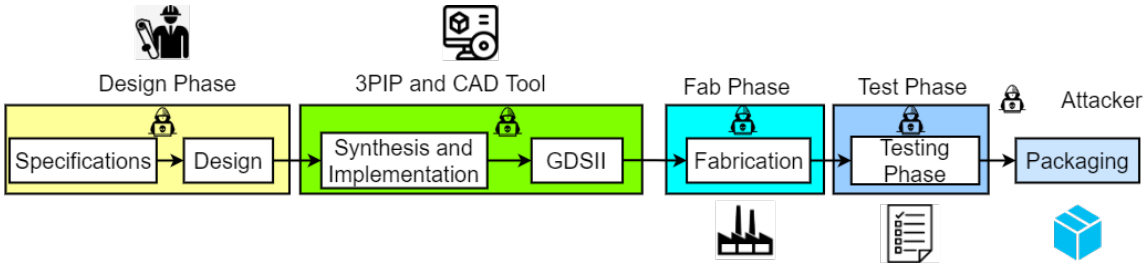

Figure 1.1: Integrated Chip (IC) supply chain process

speed. The semiconductor industry has progressed a lot in terms of speed and performance to satiate the needs but at the cost of manufacturing the IC offshore as a majority of companies have gone fabless due to various economic, cost and technology reasons thus exposing the IC supply chain to adversaries/attackers causing trust issues. The base reason for fabricating the design offshore is the smaller technology node, lower the number better the speed, it offers. The IC fabrication goes through a variety of design processes and each stage has become an access point where attackers can get their hands on to serve malicious purposes. The IC manufacturing chain is highly complex and is susceptible to various threats both from the outside and inside. Figure 1.1 presents the supply chain process, each process block is vulnerable to different type of attacks [4]. This has impacted the fabrication business economy due to the threats that are posed by untrusted fabs like reverse engineering, hardware trojans and IP piracy [5–7]. The IC design house imports the IP - intellectual property - from the IP vendors which are then integrated with the custom designs in the design house. The finalized design is thus sent to the Fabrication vendors for manufacturing. The hardware level attack could be executed at any of these stages. Logic locking [8–10], aka obfuscation, is a defense mechanism that is used to hide the true functionality of the circuit to prevent design leakage even if the attacker gets the design netlist. Additional gates are embedded in various parts of the design to hide the functionality and

these gates only act as transparent via media when correct logic is supplied. The correct logic is known as keys which are supplied to the circuit during the activation phase and are stored in the on-chip tamper proof memory. The activated IC works as designed only after the correct combination of the keys. The keys is a chain of '1's and '0's which is only known to the designer. With the state-of-the-art attacks [5–7] prevalent, protection using only obfuscation technique does not suffice to thwart the attack. A recent state-of-the-art attack known as SAT attack is based on the assumption that the attacker has access to the functional IC (IC activated by the designer using the correct key combinations) and the locked/obfuscated netlist has surged. The SAT attack [11] is based on the application of Distinguishing Input Patterns (DIPs) where the attack reduces key search space by iteratively applying DIPs to the obfuscated design and then applying the same DIPs to the functional IC thus eliminating incorrect key inputs. Along with logic locking type defense, split manufacturing [12] is a technique used to 'split' and manufacture different blocks of the design at different fabrication units. As a consequence, an IC responsible for activation of secondary IC could be manufactured at a trusted foundry, while other IC, at smaller technology, can be manufactured offshore (potential untrusted foundry). After manufacturing the designer stacks the two components together so the trusted platform can activate and control its untrusted counterpart. To achieve this methodology, an application to facilitate and evaluate different logic locking techniques against SAT attack was indispensable. The developed application will be described in the next chapter. Also, apart from the application, I was entrusted to develop and test hardware communication modules responsible for secure data transfer between the trusted and untrusted platforms. The same is presented in the next chapter.

1.1.2 Physical SCA

Data integrity and security became an essential part in the era of digital systems where privacy and confidentiality needs to be ensured. There have been a plethora of works addressing the attacks on systems, like those posed by malware [13–16], reverse engineering of hardware

[17, 18]; attacks on machine-learning assisted hardware-based malware detectors (HMDs) [19–21], adversarial attacks on machine learning [22], machine learning based attacks on hardware [23, 24], cache based side-channel attacks [25–27], etc. Of these, side-channel attack and cryptosystem has been discussed in this work. Cyrptographic mechanisms are employed to offer security to the data by encrypting the data streams with a secret key and transform the data into a human non-readable format. The attempt to exercise a brute force to decrypt the information is exhaustive and can even be unfeasible. To efficiently decode the secret key and decrypt the information, adversaries target utilizing the information obtained through side-channels, termed as side-channel attacks. Side-channels are inherent in any given design and side-channel attacks exploit the information from these rather than exploiting vulnerabilities in the software. There exist both physical and microarchitectural side-channels that can leak secure critical information through acoustics, electromagnetic (EM) radiations, power trace, thermal maps and cache-access information. Power signature based side-channel threats are a pivotal threat as power consumption is an inherent and preliminary characteristic of any digital system.

In this work I consider a power signature based side-channel attack on encryption algorithm executing on FPGAs as they are proliferating into data centers for compute-intensive operations such as encryption. For the power analysis based SCA to be successful, the attacker measures the power traces from the system while triggering crypto operations on the system. This trace is then studied statistically to deduce the secret key. The fundamental principle underlying this attack is that different modules (operations) of AES consume different power, and thereby studying the power trace reveals the operation, based on which the secret key can be deduced.

Pengyuan Yu et al. in [28] propose an intelligent place-and-route technique to facilitate symmetrical routing as a defense against power analysis SCA on FPGA. Work in [29] describes how a circuit can be transformed to a larger circuit to defend against probe-based physical SCAs, but, the technique proposed is very complex. Work in [30] and [31] describes

algorithmic countermeasures to thwart SCAs which attempts to minimize the correlation between the intermediate values and the secret key ;and by algorithmically adding noise respectively. Also, circuit-level countermeasures are presented in papers [32–36]. It is observed that the existing defenses require modifications in physical designs, leading to larger overheads and design complexity.

To overcome these challenges and defend against power analysis SCA, I propose Power Swapper. More details about the proposed Power Swapper will be discussed in the coming chapters.

Chapter 2: IC Reverse Engineering Defense

2.1 Trusted-Untrusted Design Integration

As discussed in the introduction section, a design could be subjected to IC reverse engineering (RE) attacks and design integrity attacks. Also, for security critical applications, the end design should be trustworthy. To address the IC manufacturing supply chain attacks [4–7], the idea of the project is to integrate two chips post manufacturing - one chip being a trusted chip and other could be manufactured at an offshore untrusted environment. The trusted environment could be a secure facility/foundry, and the untrusted could be a manufacturing unit outside the secure zone. To protect the untrusted design (UD) from attacks, the design would be obfuscated before the tapeout phase. The obfuscation [8–10, 37–39] adds additional gates to the design that do not contribute to the end functionality, yet inserted to increase robustness against SAT-Solver or SAT attacks [11]. The design is executed in different phases and evaluating robustness against attacks is one of the most important parts. Usually, obfuscation of the design is achieved by running various algorithms on a system, which is a laborious effort. Moreover, after the design framework is handed over to the end user, the user may not be familiar with the technicalities of the aforementioned obfuscation defenses and attacks. Automating the frontend tasks was proposed as a panacea to reduce the efforts in evaluating the attack and to render the system end-user friendly. Next, we will discuss more on the automation tool.

2.2 Automation Tool for Design Security

The automation application as shown in Figure 2.1 provides a user-friendly interface to obfuscate a design using various algorithms, and to de-obfuscate the same. Please note that

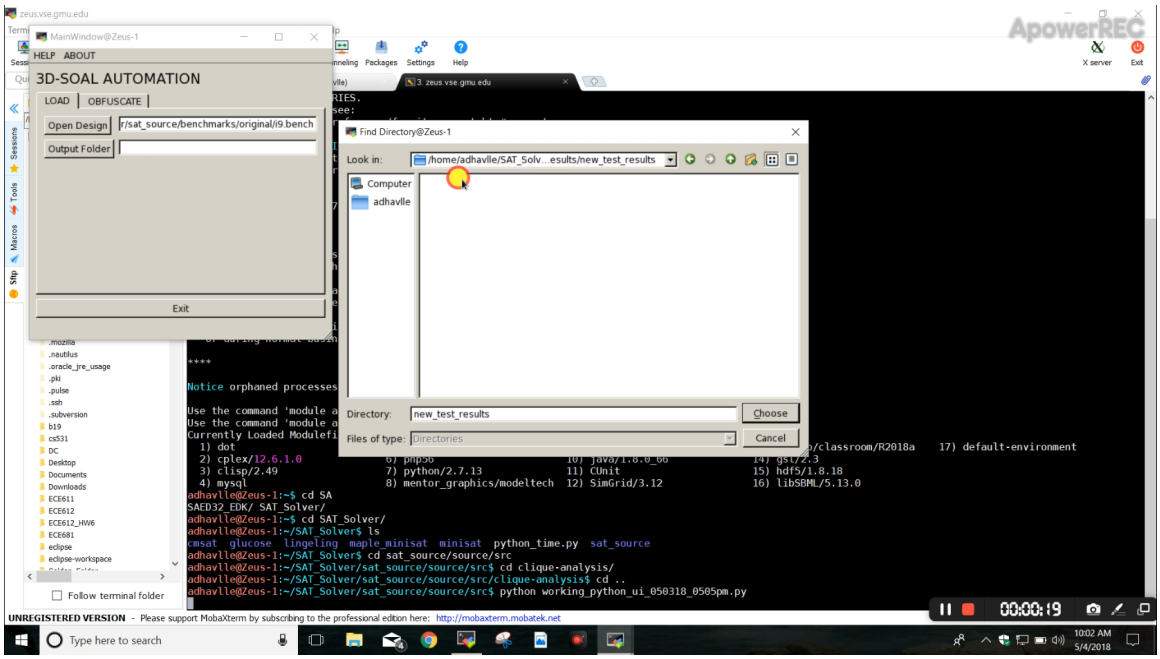

Figure 2.1: Choosing a design folder to select an input design

the obfuscation algorithms are standard and I have no contributions towards them. My task was to build the application that could utilize the standard algorithms in the background.

The parameters the application is capable of selecting are listed below:

1. The obfuscation algorithms the application can use are:
    - Iolts
    - Random
    - DAC12
    - TOC13MUX
    - TOC13XOR

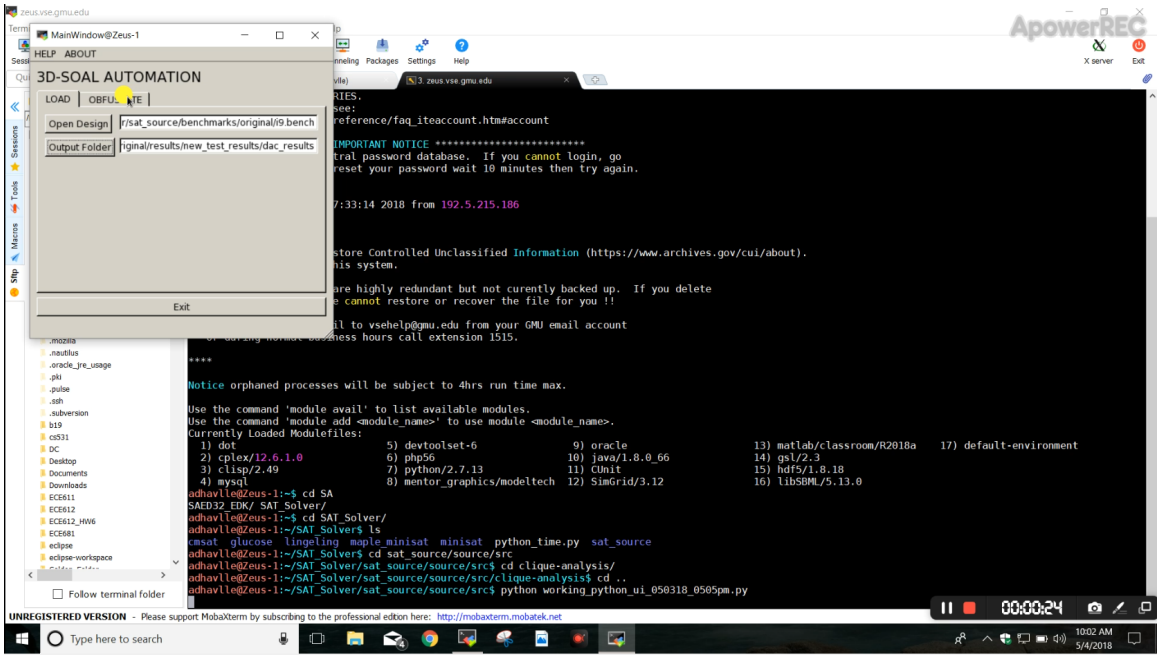

Figure 2.2: Choosing an output folder to save the resultant obfuscated design

2. The Attack tab has SAT and SMT type solvers. Within SAT, the application allows for five different solvers.
    - Lingeling
    - CMSAT
    - Glucose
    - Maple_minisat
    - Minisat
3. Number (in percent) of logic gates to obfuscate.
4. Selecting specific files required for DAC12 algorithm - .MUT and .CLIQUE files.

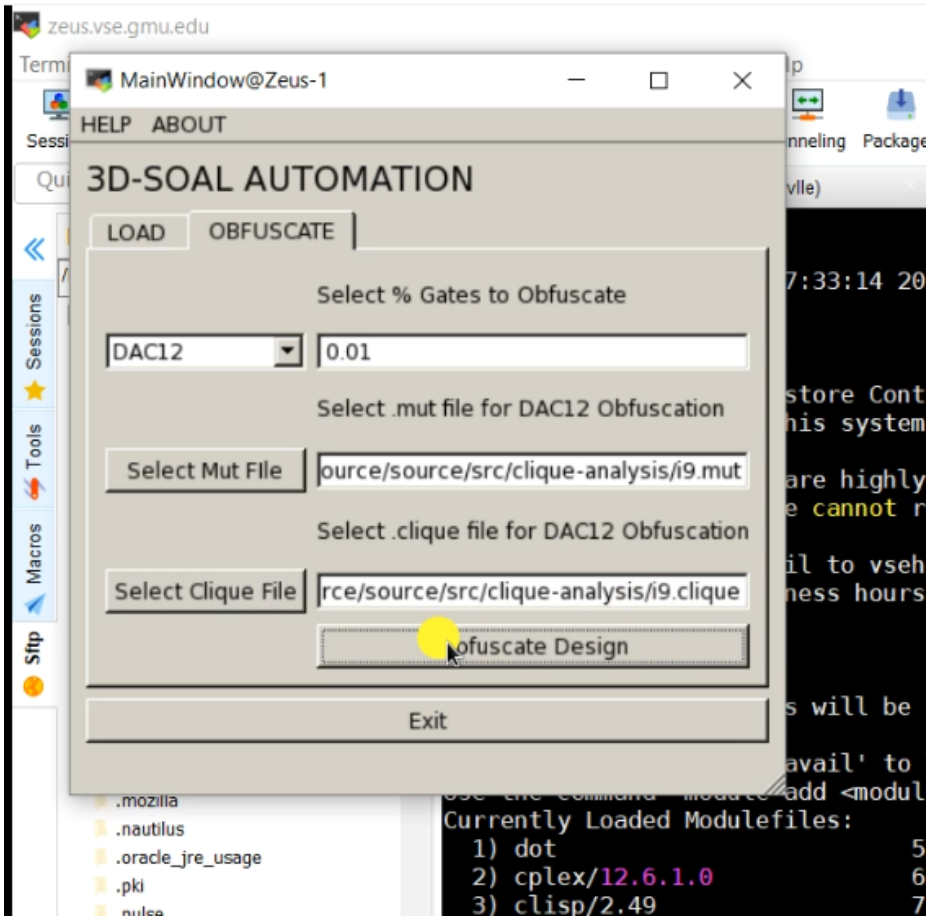

Figure 2.3: Selecting a obfuscation type, obfuscation percentage, and necessary files before obfuscating a design

5. Select the input and output folders to choose a base design from and to store the resultant outputs.

Prerequisites to executing the application are:

1. Ensure SAT solvers are installed on the system, which includes the dependencies also.
2. Ensure Python is installed.

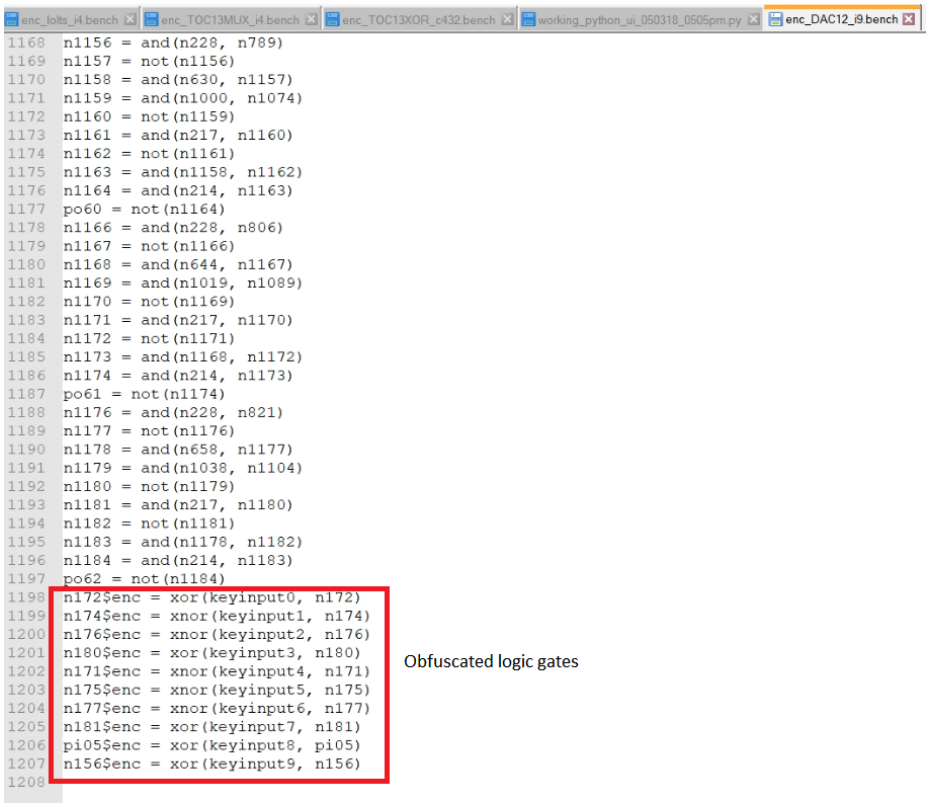

Figure 2.4: Output of an obfuscated design; the additional gates added at the end accommodate the secret key inputs

3. Install Qt designer - this is a GUI designing application that takes less efforts than directly writing the Python code for the GUI blocks.

4. Install PyQt-4/5 - this is an API to support the GUI designing.

5. The code for recreating the application is provided in Listing 2.1; the user interface's XML code is provided in Listing 2.2.

The steps to produce the obfuscated design is as follows:

1. Use the Load tab in the application and select the base design to obfuscate. Refer to

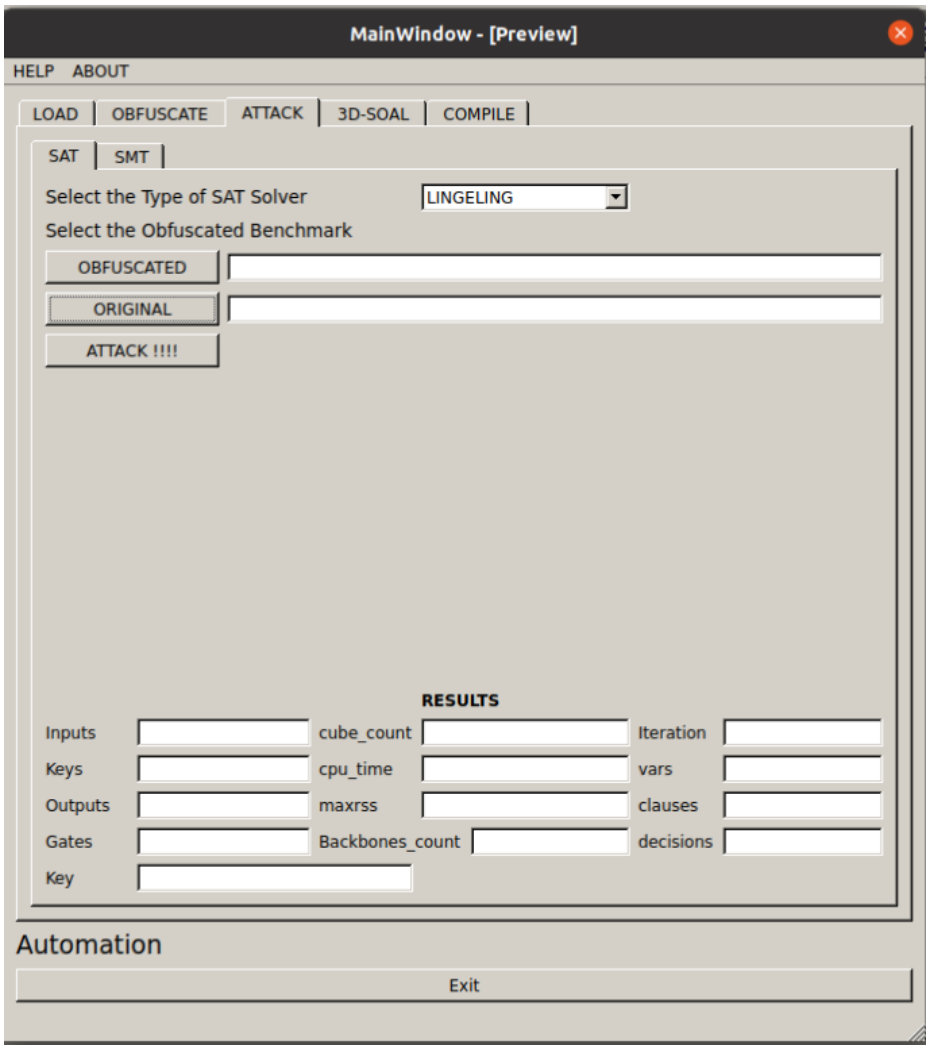

Figure 2.5: Automation for deploying different types of SAT attacks against obfuscated designs for robustness evaluation

Figure 2.1.

2. Select the output folder to store the resultant obfuscated design. Refer to Figure 2.2.

3. Move to the Obfuscate tab of the application and choose the obfuscation algorithm, percent of gates to obfuscate, and selecting .MUT and .CLIQUE files, if DAC12 was

chosen earlier. Refer to Figure 2.3.

4. Hit the 'obfuscate design' button to start the task.

5. Find the generated file in the output folder chosen earlier, and verify the change in the resultant design file. Refer to Figure 2.4.

6. For executing SAT-type attacks, click on the Attacks tab; select the type of solver, select an obfuscated design and original (unobfuscated) file; click the attack button to start the process. The results section shows different parameters of the outcome of the attack. These parameters are parsed by the script and shown in each of the boxes. Refer to Figure 2.5.

2.2.1 PyQt Code for Recreating a Basic Layout

```
from PyQt4 import QtCore, QtGui
try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s

try:
    _encoding = QtGui.QApplication.UnicodeUTF8
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig,
    _encoding)
except AttributeError:
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig)

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName(_fromUtf8("MainWindow"))
        MainWindow.resize(452, 724)
```

```
20          self.centralwidget = QtGui.QWidget(MainWindow)
21          self.centralwidget.setObjectName(_fromUtf8("centralwidget"))
22          self.gridLayout = QtGui.QGridLayout(self.centralwidget)
23          self.gridLayout.setObjectName(_fromUtf8("gridLayout"))
24          self.label = QtGui.QLabel(self.centralwidget)
25          font = QtGui.QFont()
26          font.setPointSize(14)
27          self.label.setFont(font)
28          self.label.setObjectName(_fromUtf8("label"))
29          self.gridLayout.addWidget(self.label, 0, 0, 1, 1)
30          self.tabWidget = QtGui.QTabWidget(self.centralwidget)
31          self.tabWidget.setObjectName(_fromUtf8("tabWidget"))
32          self.tab = QtGui.QWidget()
33          self.tab.setObjectName(_fromUtf8("tab"))
34          self.formLayout = QtGui.QFormLayout(self.tab)
35          self.formLayout.setObjectName(_fromUtf8("formLayout"))
36          self.pushButton_3 = QtGui.QPushButton(self.tab)
37          self.pushButton_3.setObjectName(_fromUtf8("pushButton_3"))
38          self.pushButton_3.clicked.connect(self.fetch_file)
39          self.formLayout.setWidget(0, QtGui.QFormLayout.LabelRole, self.
    pushButton_3)
40          self.lineEdit = QtGui.QLineEdit(self.tab)
41          self.lineEdit.setObjectName(_fromUtf8("lineEdit"))
42          self.formLayout.setWidget(0, QtGui.QFormLayout.FieldRole, self.
    lineEdit)
43          self.pushButton_4 = QtGui.QPushButton(self.tab)
44          self.pushButton_4.setObjectName(_fromUtf8("pushButton_4"))
45          self.pushButton_3.clicked.connect(self.fetch_output_folder)
46          self.formLayout.setWidget(1, QtGui.QFormLayout.LabelRole, self.
    pushButton_4)
47          self.lineEdit_3 = QtGui.QLineEdit(self.tab)
48          self.lineEdit_3.setObjectName(_fromUtf8("lineEdit_3"))
49          self.formLayout.setWidget(1, QtGui.QFormLayout.FieldRole, self.
    lineEdit_3)
50          self.tabWidget.addTab(self.tab, _fromUtf8(""))
```

```
51          self.tab_2 = QtGui.QWidget()
52          self.tab_2.setObjectName(_fromUtf8("tab_2"))
53          self.formLayout_2 = QtGui.QFormLayout(self.tab_2)
54          self.formLayout_2.setFieldGrowthPolicy(QtGui.QFormLayout.
    AllNonFixedFieldsGrow)
55          self.formLayout_2.setObjectName(_fromUtf8("formLayout_2"))
56          self.label_4 = QtGui.QLabel(self.tab_2)
57          self.label_4.setObjectName(_fromUtf8("label_4"))
58          self.formLayout_2.setWidget(0, QtGui.QFormLayout.FieldRole, self.
    label_4)
59          self.comboBox = QtGui.QComboBox(self.tab_2)
60          self.comboBox.setObjectName(_fromUtf8("comboBox"))
61          self.comboBox.addItem(_fromUtf8(""))
62          self.comboBox.addItem(_fromUtf8(""))
63          self.comboBox.addItem(_fromUtf8(""))
64          self.comboBox.addItem(_fromUtf8(""))
65          self.comboBox.addItem(_fromUtf8(""))
66          self.formLayout_2.setWidget(1, QtGui.QFormLayout.LabelRole, self.
    comboBox)
67          self.lineEdit_2 = QtGui.QLineEdit(self.tab_2)
68          self.lineEdit_2.setObjectName(_fromUtf8("lineEdit_2"))
69          self.formLayout_2.setWidget(1, QtGui.QFormLayout.FieldRole, self.
    lineEdit_2)
70          self.pushButton_2 = QtGui.QPushButton(self.tab_2)
71          self.pushButton_2.setObjectName(_fromUtf8("pushButton_2"))
72          self.formLayout_2.setWidget(2, QtGui.QFormLayout.FieldRole, self.
    pushButton_2)
73          self.tabWidget.addTab(self.tab_2, _fromUtf8(""))
74          self.gridLayout.addWidget(self.tabWidget, 1, 0, 1, 1)
75          self.pushButton = QtGui.QPushButton(self.centralwidget)
76          self.pushButton.setObjectName(_fromUtf8("pushButton"))
77          self.gridLayout.addWidget(self.pushButton, 2, 0, 1, 1)
78          self.label_2 = QtGui.QLabel(self.centralwidget)
79          self.label_2.setObjectName(_fromUtf8("label_2"))
80          self.gridLayout.addWidget(self.label_2, 3, 1, 1, 1)
```

```
81          MainWindow.setCentralWidget(self.centralwidget)
82          self.statusbar = QtGui.QStatusBar(MainWindow)
83          self.statusbar.setObjectName(_fromUtf8("statusbar"))
84          MainWindow.setStatusBar(self.statusbar)
85          self.menubar = QtGui.QMenuBar(MainWindow)
86          self.menubar.setGeometry(QtCore.QRect(0, 0, 452, 26))
87          self.menubar.setObjectName(_fromUtf8("menubar"))
88          self.menuHELP = QtGui.QMenu(self.menubar)
89          self.menuHELP.setObjectName(_fromUtf8("menuHELP"))
90          self.menuABOUT = QtGui.QMenu(self.menubar)
91          self.menuABOUT.setObjectName(_fromUtf8("menuABOUT"))
92          MainWindow.setMenuBar(self.menubar)
93          self.actionDont_ask_for_any_help = QtGui.QAction(MainWindow)
94          self.actionDont_ask_for_any_help.setObjectName(_fromUtf8("
    actionDont_ask_for_any_help"))
95          self.actionHkjhk = QtGui.QAction(MainWindow)
96          self.actionHkjhk.setObjectName(_fromUtf8("actionHkjhk"))
97          self.actionAbout_3D_SOAL_Automator = QtGui.QAction(MainWindow)
98          self.actionAbout_3D_SOAL_Automator.setObjectName(_fromUtf8("
    actionAbout_3D_SOAL_Automator"))
99          self.menuHELP.addSeparator()
100         self.menuHELP.addAction(self.actionDont_ask_for_any_help)
101         self.menuABOUT.addAction(self.actionAbout_3D_SOAL_Automator)
102         self.menubar.addAction(self.menuHELP.menuAction())
103         self.menubar.addAction(self.menuABOUT.menuAction())
104
105         self.retranslateUi(MainWindow)
106         self.tabWidget.setCurrentIndex(0)
107         QtCore.QMetaObject.connectSlotsByName(MainWindow)
108
109     def retranslateUi(self, MainWindow):
110         MainWindow.setWindowTitle(_translate("MainWindow", "3D-SOAL
    Automator", None))
111         self.label.setText(_translate("MainWindow", "3D-SOAL AUTOMATION",
    None))
```

```
112         self.pushButton_3.setText(_translate("MainWindow", "Open Design",
    None))
113         self.pushButton_4.setText(_translate("MainWindow", "Output Folder",
    None))
114         self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab),
    _translate("MainWindow", "LOAD", None))
115         self.label_4.setText(_translate("MainWindow", "Select % Gates to
    Obfuscate", None))
116         self.comboBox.setItemText(0, _translate("MainWindow", "Iolts", None)
    )
117         self.comboBox.setItemText(1, _translate("MainWindow", "Random", None
    ))
118         self.comboBox.setItemText(2, _translate("MainWindow", "DAC", None))
119         self.comboBox.setItemText(3, _translate("MainWindow", "MUX", None))
120         self.comboBox.setItemText(4, _translate("MainWindow", "TOC13", None)
    )
121         self.pushButton_2.setText(_translate("MainWindow", "Obfuscate Design
    ", None))
122         self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_2),
    _translate("MainWindow", "OBFUSCATE", None))
123         self.pushButton.setText(_translate("MainWindow", "Exit", None))
124         self.label_2.setText(_translate("MainWindow", "Created By :SSS, AAD"
    , None))
125         self.menuHELP.setTitle(_translate("MainWindow", "HELP", None))
126         self.menuABOUT.setTitle(_translate("MainWindow", "ABOUT", None))
127         self.actionDont_ask_for_any_help.setText(_translate("MainWindow", "
    Dont ask for any help!!", None))
128         self.actionHkjhk.setText(_translate("MainWindow", "hkjhk", None))
129         self.actionAbout_3D_SOAL_Automator.setText(_translate("MainWindow",
    "About 3D-SOAL Automator", None))
130
131
132     def fetch_file(self):
133         dlg = QFileDialog()
134         dlg.setFileMode(QFileDialog.AnyFile)
```

```
135         dlg.setFilter("Benchmark File (*.bench)")
136         filenames = QStringList()
137         self.lineEdit.setText(filenames)
138 
139     def fetch_output_folder(self):
140         dlg1 = QFileDialog()
141         dlg1.setFileMode(QFileDialog.AnyFile)
142         #dlg1.setFilter("Benchmark File (*.bench)")
143         folder_names = QStringList()
144         self.lineEdit3.setText(folder_names)
145 
146 if __name__ == "__main__":
147     import sys
148     app = QtGui.QApplication(sys.argv)
149     MainWindow = QtGui.QMainWindow()
150     ui = Ui_MainWindow()
151     ui.setupUi(MainWindow)
152     MainWindow.show()
153     sys.exit(app.exec_())
```

Listing 2.1: PyQt code for recreating a basic layout

2.2.2 User Interface (UI) designed in QT Designer

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ui version="4.0">
3  <class>MainWindow</class>
4  <widget class="QMainWindow" name="MainWindow">
5   <property name="geometry">
6    <rect>
7     <x>0</x>
8     <y>0</y>
```

```
    <width>452</width>
    <height>724</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <layout class="QGridLayout" name="gridLayout">
    <item row="0" column="0">
     <widget class="QLabel" name="label">
      <property name="font">
       <font>
        <pointsize>14</pointsize>
       </font>
      </property>
      <property name="text">
       <string>3D-SOAL AUTOMATION</string>
      </property>
     </widget>
    </item>
    <item row="1" column="0">
     <widget class="QTabWidget" name="tabWidget">
      <property name="currentIndex">
       <number>1</number>
      </property>
      <widget class="QWidget" name="tab">
       <attribute name="title">
        <string>LOAD</string>
       </attribute>
       <layout class="QFormLayout" name="formLayout">
        <item row="0" column="0">
         <widget class="QPushButton" name="pushButton_3">
          <property name="text">
           <string>Open Design</string>
```

```
          </property>
         </widget>
        </item>
        <item row="0" column="1">
         <widget class="QLineEdit" name="lineEdit"/>
        </item>
        <item row="1" column="0">
         <widget class="QPushButton" name="pushButton_4">
          <property name="text">
           <string>Output Folder</string>
          </property>
         </widget>
        </item>
        <item row="1" column="1">
         <widget class="QLineEdit" name="lineEdit_3"/>
        </item>
       </layout>
      </widget>
      <widget class="QWidget" name="tab_2">
       <attribute name="title">
        <string>OBFUSCATE</string>
       </attribute>
       <layout class="QFormLayout" name="formLayout_2">
        <property name="fieldGrowthPolicy">
         <enum>QFormLayout::AllNonFixedFieldsGrow</enum>
        </property>
        <item row="0" column="1">
         <widget class="QLabel" name="label_4">
          <property name="text">
           <string>Select % Gates to Obfuscate</string>
          </property>
         </widget>
        </item>
        <item row="1" column="0">
         <widget class="QComboBox" name="comboBox">
```

```
            <item>
             <property name="text">
              <string>Iolts</string>
             </property>
            </item>
            <item>
             <property name="text">
              <string>Random</string>
             </property>
            </item>
            <item>
             <property name="text">
              <string>DAC</string>
             </property>
            </item>
            <item>
             <property name="text">
              <string>MUX</string>
             </property>
            </item>
            <item>
             <property name="text">
              <string>TOC13</string>
             </property>
            </item>
           </widget>
          </item>
          <item row="1" column="1">
           <widget class="QLineEdit" name="lineEdit_2"/>
          </item>
          <item row="2" column="1">
           <widget class="QPushButton" name="pushButton_2">
            <property name="text">
             <string>Obfuscate Design</string>
            </property>
```

```
          </widget>
         </item>
        </layout>
       </widget>
      </widget>
     </item>
     <item row="2" column="0">
      <widget class="QPushButton" name="pushButton">
       <property name="text">
        <string>Exit</string>
       </property>
      </widget>
     </item>
     <item row="3" column="1">
      <widget class="QLabel" name="label_2">
       <property name="text">
        <string>Created By :SSS, AAD</string>
       </property>
      </widget>
     </item>
    </layout>
   </widget>
   <widget class="QStatusBar" name="statusbar"/>
   <widget class="QMenuBar" name="menubar">
    <property name="geometry">
     <rect>
      <x>0</x>
      <y>0</y>
      <width>452</width>
      <height>26</height>
     </rect>
    </property>
    <widget class="QMenu" name="menuHELP">
     <property name="title">
      <string>HELP</string>
```

```
149      </property>
150      <addaction name="separator"/>
151      <addaction name="actionDont_ask_for_any_help"/>
152     </widget>
153     <widget class="QMenu" name="menuABOUT">
154      <property name="title">
155       <string>ABOUT</string>
156      </property>
157      <addaction name="actionAbout_3D_SOAL_Automator"/>
158     </widget>
159     <addaction name="menuHELP"/>
160     <addaction name="menuABOUT"/>
161    </widget>
162    <action name="actionDont_ask_for_any_help">
163     <property name="text">
164      <string>Dont ask for any help!!</string>
165     </property>
166    </action>
167    <action name="actionHkjhk">
168     <property name="text">
169      <string>hkjhk</string>
170     </property>
171    </action>
172    <action name="actionAbout_3D_SOAL_Automator">
173     <property name="text">
174      <string>About 3D-SOAL Automator</string>
175     </property>
176    </action>
177   </widget>
178   <resources/>
179   <connections/>
180  </ui>
```

Listing 2.2: UI code designed in QT designer for the application

2.3 Reconfigurable Modules for Trusted-Untrusted Platform Communication

The trusted and untrusted platforms are stacked/connected together so the master platform can control the other untrusted platform. Depending on the scenario and other design constraints there may arise a need to either establish a serial (less wires and slower data) communication versus a parallel (more wires and faster data) communication, or vice versa. To address this, I was asked to design and test a parallel-to-parallel hardware communication modules. The interface of both the platforms is a serial connection, the final ends being a parallel data-in and data-out approach. The two platforms may have different operating clock speeds. This could also be referred as a clock domain crossing scenario. Hence, it is necessary to realize the design using finite state machine (FSM) approach. The block diagram of the communication blocks is shown in Figure 2.6; the block diagrams of the state machine is shown in Figure 2.7. Referring to Figure 2.6, the modules use a AXI interface to communicate with each other and other blocks of the higher level design. The AXI interface has standardized signal names to denote their functionality. The data bus could be an input or a output that takes data into the block or out from the block. The valid signal denotes that the content on the data bus is 'valid' and ready to be consumed. The valid could be an input or an output signal depending on whether it is placed on the consumer side or the generator side. A consumer block accepts the data from the block previous to it; a generator block generates some data and feeds it to the consumer block after it. A ready signal indicates the block is ready to process next piece of data.

The two state machines with different clocks is shown in Figure 2.7. The state machine begins in Idle state; part (a) waits for the valid signal and content on the data bus. The Check Busy state checks if the done signal goes high, after which the ready-out signal is set to '1'. The done signal is active when the data is processed by the block and ready to be transferred to the next block. In part (b), the Check Busy state expects a busy wait signal based on which it moves to the next state, Send Data, which converts the parallel data into

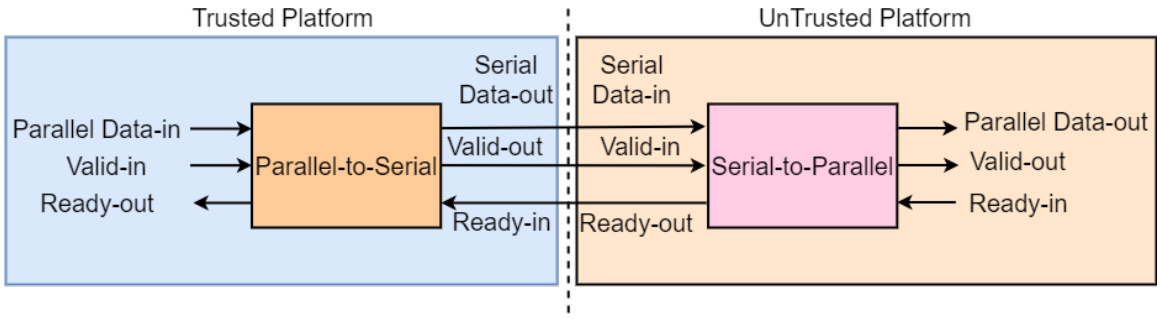

Figure 2.6: Hardware communication modules that establish communication between the trusted and untrusted platforms

serial data. While this conversion occurs, the state machine checks for the last piece of data to be processed in the Check Last state. After the conversion of data, the Send Data block will transfer the data out on the bus if the Ready-in signal is active.

The modules were realized in VHDL language and Xilinx Vivado was used to simulate, synthesize and generate waveforms. The simulation waveforms for 64, 128, and 512 bit data input are shown in Figure 2.8, 2.9, and 2.10. It can be seen in the diagrams that the ready-out signal is set to '1' when the data processing is complete. The code for the parallel to serial hardware module is presented in Listing 2.3. The resource utilization for 64, 128 and 512-bits configuration implemented on a Zedboard FPGA is shown in Figure 2.1, 2.2, and 2.3.

2.3.1 VHDL code for Parallel to Serial Module

```
1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_SIGNED.ALL;
5 use IEEE.numeric_std.all;
6 use work.my_package.all;
```

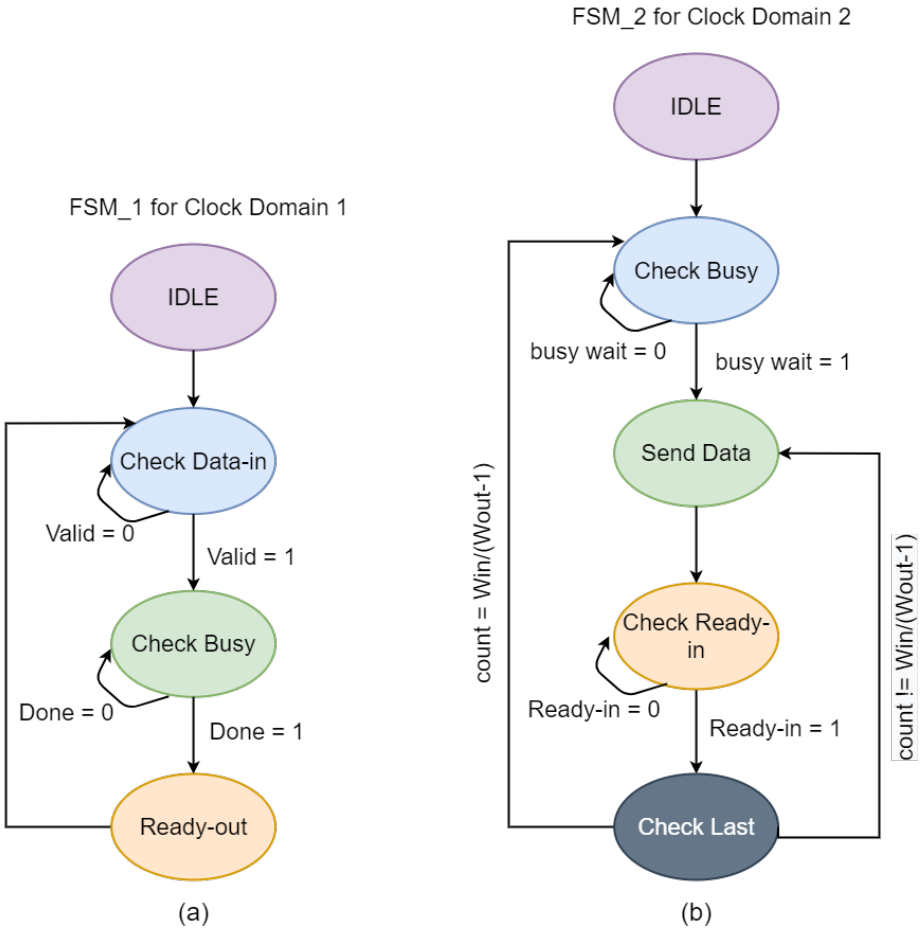

Figure 2.7: Finite State Machine (FSM) diagrams for input and output side, which utilize two different clocks

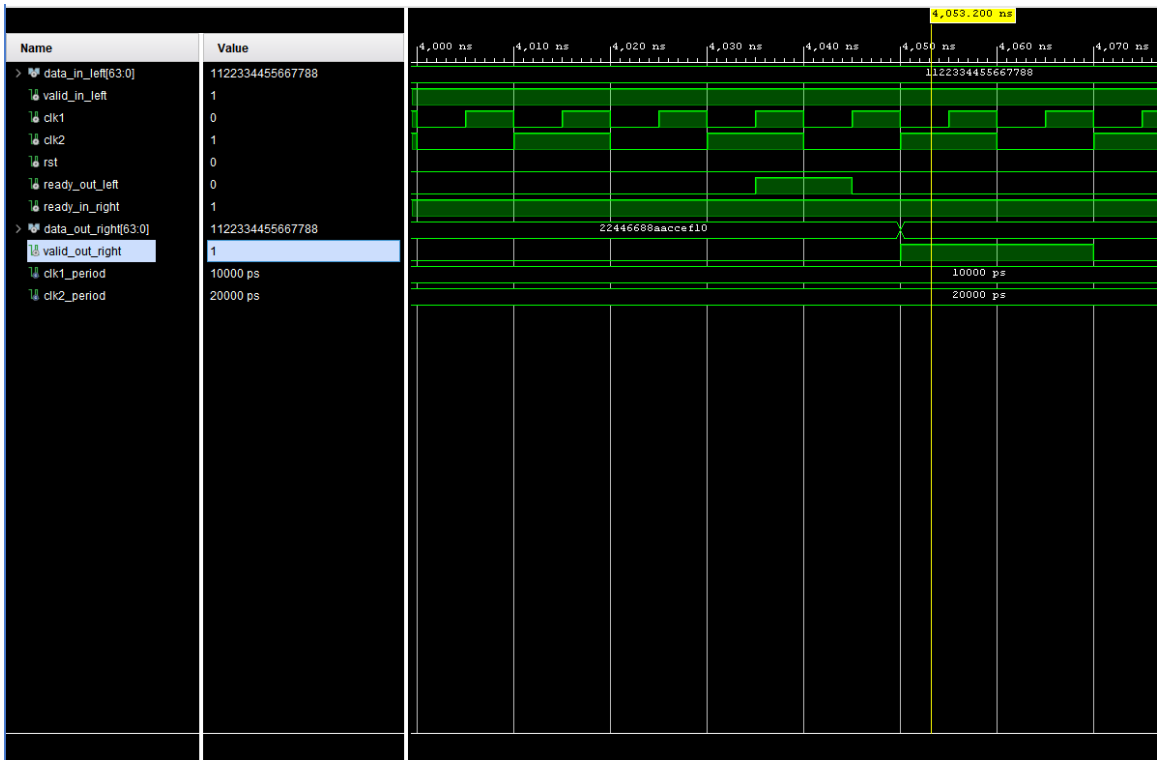

Figure 2.8: Simulation waveform for 64-bit parallel input

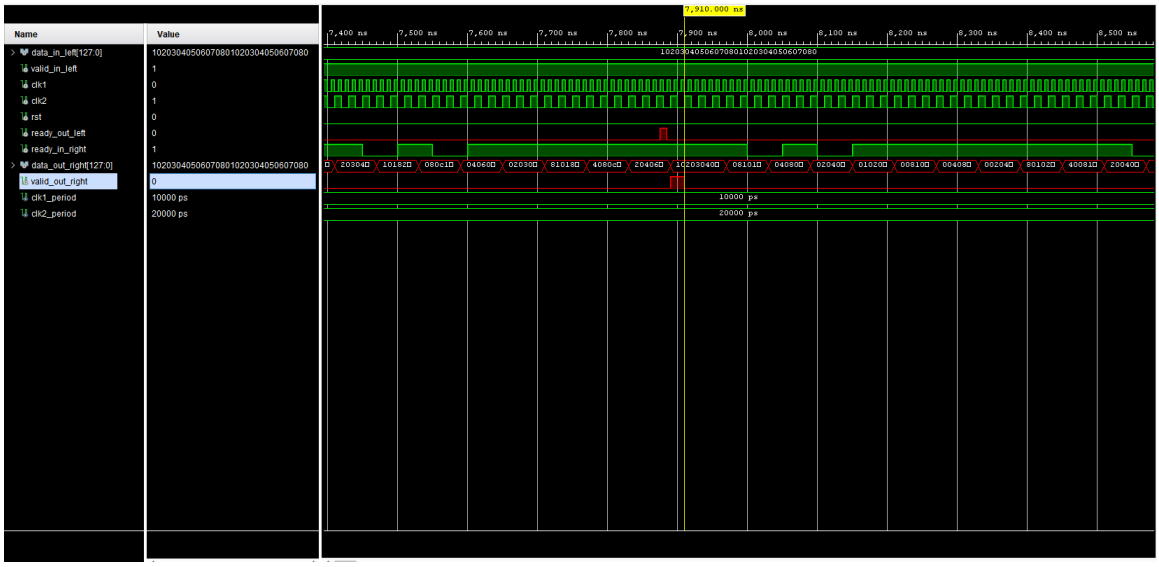

Figure 2.9: Simulation waveform for 128-bit parallel input

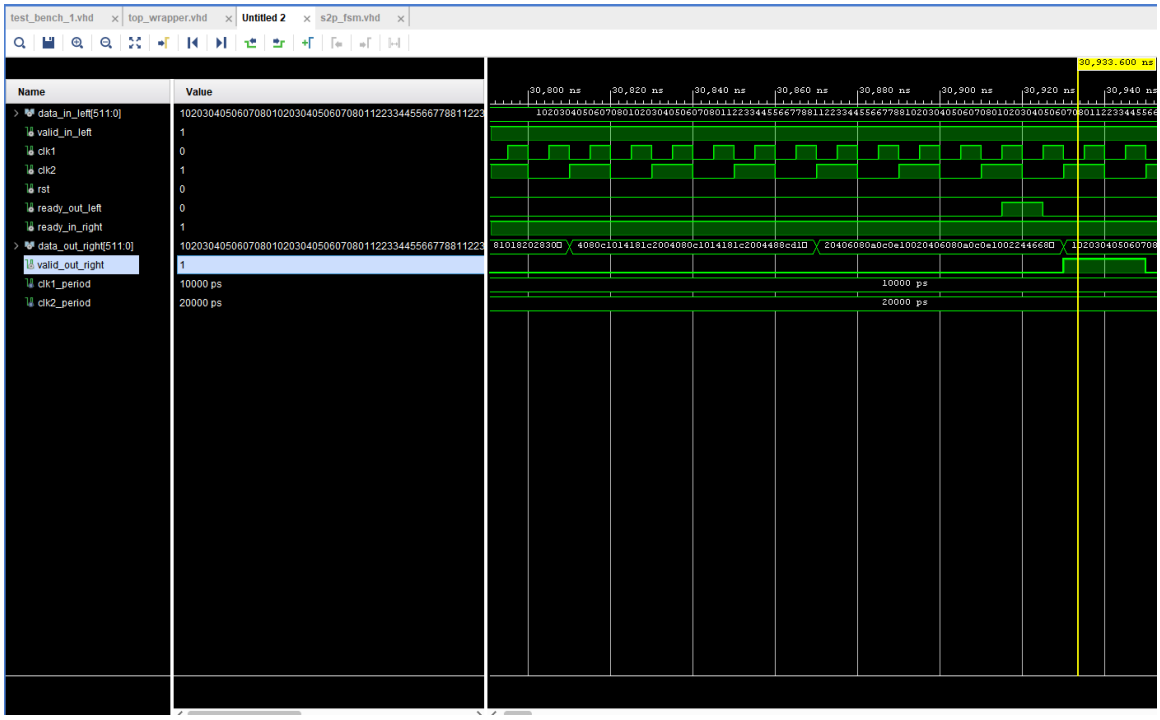

Figure 2.10: Simulation waveform for 512-bit parallel input

```
7
8  entity P2S is
9      generic(
10         w_out : integer := 1;
11         w_in : integer := 512
12     );
13     Port ( data_in : in  STD_LOGIC_VECTOR (w_in-1 downto 0);
14            valid_in : in  STD_LOGIC;
15            ready_out : out  STD_LOGIC;
16            clk1 : in  STD_LOGIC;
17            clk2 : in  STD_LOGIC;
18            rst  : in  STD_LOGIC;
19            data_out : out  STD_LOGIC_VECTOR (w_out-1 downto 0);
20            valid_out : out  STD_LOGIC;
21            ready_in : in  STD_LOGIC);
22 end P2S;
23
24 architecture Behavioral of P2S is
25
26 type state_type1 is (FSM1_IDLE, FSM1_CHKDIN, FSM1_CHKBUSY, FSM1_READYOUT);
27 type state_type2 is (FSM2_IDLE, FSM2_CHKBUSY, FSM2_SEND_DATA,
       FSM2_CHK_READYIN, FSM2_CHKLAST, FSM2_DONE);
28
29 type T_SLVV is array (NATURAL range <>)of std_logic_vector(w_out-1 downto 0)
       ;
30 signal state1, next_state1 : state_type1;
31 signal state2, next_state2 : state_type2;
32
33 --Declare internal signals for all outputs of the state-machine
34
35 signal busy_wait  : std_logic;  -- example output signal
36 signal done      : std_logic;  -- example output signal
37 signal count: std_logic_vector(log2(w_in/w_out)-1 downto 0);
38 signal mux_in : T_SLVV (w_in/w_out-1 downto 0);
39
```

```
signal ready_out_buff : std_logic;  -- example output signal
signal valid_out_buff : std_logic;  -- example output signal

--other outputs
begin
valid_out <= valid_out_buff;
ready_out <= ready_out_buff;

--FSM1 defined
FSM1_state_update: process (clk1)
   begin
      if (clk1'event and clk1 = '1') then
         if (rst = '1') then
            state1 <= FSM1_IDLE;
         else
            state1 <= next_state1;
         end if;
      end if;
   end process;

FSM1_ready_out: process (state1, rst, clk1, done, valid_in, count)
   begin
      if (state1 = FSM1_IDLE) then
         ready_out_buff <= '1';
      elsif(state1 = FSM1_READYOUT) then
        ready_out_buff <= '1';

    elsif(state1 = FSM1_CHKDIN and valid_in = '1') then
      ready_out_buff <= '0';
      else
      ready_out_buff <= ready_out_buff;
    end if;

   end process;

```

```
FSM1_busy_wait: process (state1, rst, clk1, done, valid_in)
   begin
      if (state1 = FSM1_CHKDIN and valid_in = '1') then
         busy_wait <= '1';
      elsif(state1 = FSM1_READYOUT) then
         busy_wait <= '0';
      elsif(state1 = FSM1_IDLE) then
      busy_wait <= '0';
    else
      busy_wait <= busy_wait;
    end if;
   end process;

FSM1_digram: process (state1, rst, clk1, done, valid_in)
   begin
      next_state1 <= state1;
      case (state1) is
         when FSM1_IDLE =>
        next_state1 <= FSM1_CHKDIN;
         when FSM1_CHKDIN =>
            if valid_in = '1' then
               next_state1 <= FSM1_CHKBUSY;
        else
          next_state1 <= FSM1_CHKDIN;
            end if;
         when FSM1_CHKBUSY =>
        if done = '1' then
          next_state1 <= FSM1_READYOUT;
        else
          next_state1 <= FSM1_CHKBUSY;
        end if;
         when FSM1_READYOUT =>
        next_state1 <= FSM1_CHKDIN;
         when others =>
            next_state1 <= FSM1_IDLE;
```

```
110       end case;
111    end process;
112
113 --FSM2 defined
114 FSM2_state_update: process (clk2)
115    begin
116       if (clk2'event and clk2 = '1') then
117          if (rst = '1') then
118             state2 <= FSM2_IDLE;
119          else
120             state2 <= next_state2;
121          end if;
122       end if;
123    end process;
124
125 FSM2_done_proc: process (state2, rst, clk2, count)
126    begin
127       if (state2 = FSM2_CHKLAST and count = w_in/w_out -1) then
128          done <= '1';
129       elsif (state2 = FSM2_CHKBUSY or state2 = FSM2_IDLE) then
130       done <= '0';
131     else
132       done <= done;
133     end if;
134    end process;
135
136 FSM2_valid_out: process (state2, rst, clk2, count, busy_wait)
137    begin
138       if (state2 = FSM2_SEND_DATA and busy_wait = '1') then
139          valid_out_buff <= '1';
140     elsif (state2 = FSM2_CHK_READYIN) then
141       valid_out_buff <= '0';
142     elsif (state2 = FSM2_CHKLAST and count /= w_in/w_out -1) then
143       valid_out_buff <= '0';
144       elsif (state2 = FSM2_CHKBUSY or state2 = FSM2_IDLE) then
```

```
145       valid_out_buff <= '0';
146     else
147       valid_out_buff <= valid_out_buff;
148     end if;
149    end process;
150
151 FSM2_digram: process (state2, rst, clk2, busy_wait, ready_in, count)
152    begin
153       next_state2 <= state2;
154       case (state2) is
155          when FSM2_IDLE =>
156         next_state2 <= FSM2_CHKBUSY;
157          when FSM2_CHKBUSY =>
158             if busy_wait = '0' then
159                next_state2 <= FSM2_CHKBUSY;
160         else
161           next_state2 <= FSM2_SEND_DATA;
162             end if;
163          when FSM2_SEND_DATA =>
164           next_state2 <= FSM2_CHK_READYIN;
165          when FSM2_CHK_READYIN =>
166         if ready_in = '0' then
167           next_state2 <= FSM2_CHK_READYIN;
168         else
169           next_state2 <= FSM2_CHKLAST;
170         end if;
171          when FSM2_CHKLAST =>
172         if count = w_in/w_out-1 then
173           next_state2 <= FSM2_CHKBUSY;
174         else
175           next_state2 <= FSM2_SEND_DATA;
176         end if;
177       when others =>
178             next_state2 <= FSM2_IDLE;
179       end case;
```

```
180     end process;
181
182 --process for counter
183 Counter: process (clk2)
184   begin
185     if clk2='1' and clk2'event then
186       if (rst = '1') then
187         count <= (others => '0');
188       elsif state2 = FSM2_CHKLAST then
189         count <= count + 1;
190       elsif(state2 = FSM2_CHKBUSY)then
191         count <= (others => '0');
192       elsif (state2 = FSM2_IDLE) then
193         count <= (others => '0');
194       else
195         count <= count;
196       end if;
197     end if;
198 end process;
199
200  --original one commented below
201   generate_data_out : for i in 0 to (w_in/w_out)-1 generate
202         mux_in(i) <= data_in((I*w_out)+w_out-1 downto I*w_out);
203    end generate;
204    data_out <= mux_in(to_integer(unsigned(count)));
205
206 end Behavioral;
```

Listing 2.3: VHDL code for realizing the parallel to serial converter hardware module

Table 2.1: Resource utilization for P2S and S2P module for 64-bit parallel input data

Total Resource	Slice LUTs	Slice Registers	F7 Muxes	F8 Muxes	Bonded IOB	BUFCTRL
	53200	106400	26600	13300	200	32
Top Wrapper Utilization (%)	0.10	0.09	0.03	0.03	67.50	6.25
P2S Utilization (%)	0.07	0.02	0.03	0.03	0.0	0.0
S2P Utilization (%)	0.03	0.08	0.0	0.0	0.0	0.0

Table 2.2: Resource utilization for P2S and S2P module for 128-bit parallel input data

Total Resource	Slice LUTs	Slice Registers	F7 Muxes	F8 Muxes
	53200	106400	26600	13300
Top Wrapper Utilization (%)	0.15	0.16	0.06	0.06
P2S Utilization (%)	0.11	0.02	0.06	0.06
S2P Utilization (%)	0.04	0.14	0.0	0.0

Table 2.3: Resource utilization for P2S and S2P module for 512-bit parallel input data

Total Resource	Slice LUTs	Slice Registers	F7 Muxes	F8 Muxes	Bonded IOB	BUFCTRL
	53200	106400	26600	13300	200	32
Top Wrapper Utilization (%)	0.35	0.52	0.26	0.24	515.50	6.25
P2S Utilization (%)	0.30	0.02	0.26	0.24	0.0	0.0
S2P Utilization (%)	0.04	0.50	0.0	0.0	0.0	0.0

Chapter 3: Defense Against CPA-based Physical Side-Channel Attack

3.1 Physical Side-Channel Attack

In this section, an introduction to physical side-channel attacks is provided, followed by the setup used for power measurement and a background of correlation power analysis (CPA). Data integrity and security became an essential part in the era of digital systems where privacy and confidentiality needs to be ensured. There have been a plethora of works addressing the attacks on systems, like those posed by malware [13–16], reverse engineering of hardware [17, 18], attacks on machine-learning assisted hardware-based malware detectors (HMDs) [19,20], adversarial attacks on machine learning [22], cache based side-channel attacks [25, 27, 40], etc. Of these, side-channel attack on cryptosystem is discussed in this work. To prevent such attacks, cyrptographic mechanisms are employed to offer security to the data by encrypting the data streams with a secret key and transform the data into a human non-readable format. The attempt to exercise a brute force to decrypt the information is exhaustive and can even be unfeasible. To efficiently decode the secret key and decrypt the information, adversaries target utilizing the information obtained through side-channels, termed as side-channel attacks. Side-channels are inherent in any given design and physical side-channel attacks exploit the information from these rather than exploiting vulnerabilities in the software. There exist both physical and microarchitectural side-channels that can leak secure critical information through acoustics, electromagnetic (EM) radiations, power trace, thermal maps and cache-access information [41–45]. Power signature based side-channel threats are a pivotal threat as power consumption is an inherent and preliminary characteristic of any digital system.

This work considers a power signature based side-channel attack on encryption algorithm executing on FPGAs as they are proliferating into data centers for compute-intensive operations such as encryption. For the power analysis based SCA to be successful, the attacker measures the power traces from the system while triggering crypto operations on the system. This trace is then studied statistically to deduce the secret key. The fundamental principle underlying this attack is that different modules (operations) of AES consume different power, and thereby studying the power trace reveals the operation, based on which the secret key can be deduced.

Pengyuan Yu et al. in [28] propose an intelligent place-and-route technique to facilitate symmetrical routing as a defense against power analysis SCA on FPGA. Work in [29] describes how a circuit can be transformed to a larger circuit to defend against probe-based physical SCAs, but, the technique proposed is very complex. Work in [30] and [31] describes algorithmic countermeasures to thwart SCAs which attempts to minimize the correlation between the intermediate values and the secret key ;and by algorithmically adding noise respectively. Also, circuit-level countermeasures are presented in papers [32–36]. It is observed that the existing defenses require modifications in physical designs, leading to larger overheads and design complexity.

To overcome these challenges and defend against power analysis SCA, I propose Power Swapper. More details on the proposed Power Swapper are presented below.

3.1.1 Introduction to Correlation Power Analysis

The setup harnessed for measuring the power is described in this section followed by a brief introduction to the process of CPA (Correlation Power Analysis) analysis for key extraction.

3.1.1.1 FOBOS Setup

The setup is built specially for measuring FPGA core power for physical side-channel attack analysis. The setup is termed as FOBOS (Flexible Open-source workBench fOr Side-channel

analysis) [1, 2]. Figure 3.1 shows the block diagram of the FOBOS setup, Figure 3.2 is a photo of the components connected together for power measurement. The Controller is an Artix-7 based FPGA that receives the test vectors from the PC and communicates with the DUT FPGA (Device Under Test) which is the target FPGA platform running the AES implemenatation. The target initiates instances of the AES cryptosystem and delivers the results to the controller. Meanwhile, the controller also triggers Picoscope measurement cycle at the same time as the AES and delivers the measured power and the cipher text to the PC. The Picoscope captures the entire trace of the AES cycle and keeps iterating for every new AES cycle. The PC runs the Python scripts that are responsible for sending the test vectors and key (secret key) to the controller and accumulating all the results in a numpy array. The FOBOS is completely reconfigurable to suit the specific needs of the measurements and application. The setup is described in more detail in [1, 2].

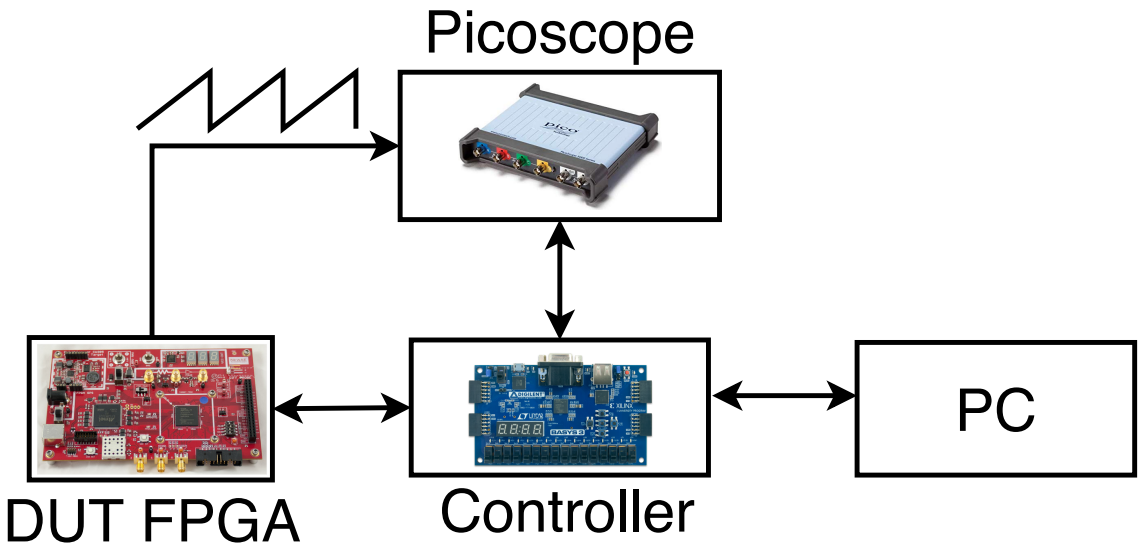

Figure 3.1: In-house setup for measuring the power of the DUT running AES implementation [1, 2]

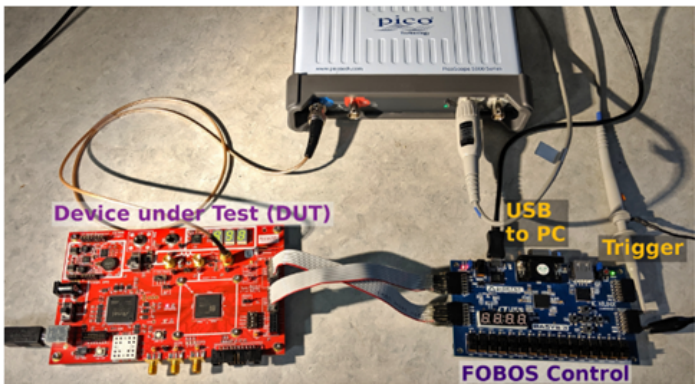

Figure 3.2: Actual photo of the FOBOS setup for power measurement [1, 2]

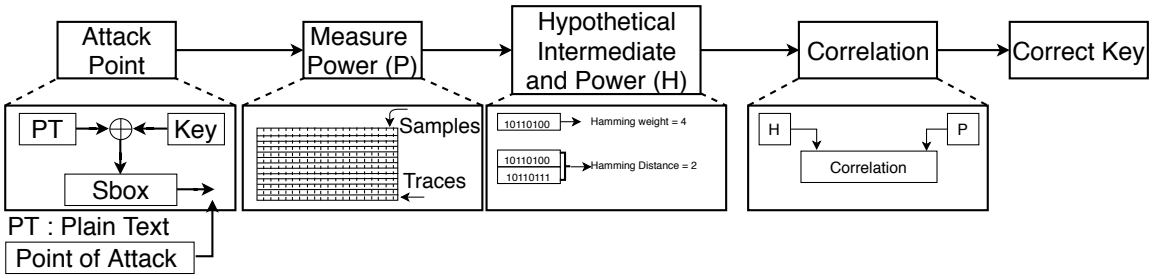

Figure 3.3: The process of Correlation Power Analysis (CPA) to extract the correct sequence of key used by the AES

3.1.1.2 Attack Model

The attack model and the assumptions for a successful attack have been described below:

1. Physical access: The adversary needs to have physical access to the cryptosystem for obvious reasons - the CPA analysis needs power traces as one of its inputs to calculate key bytes. The physical access taps the power input to the FPGA so the traces can

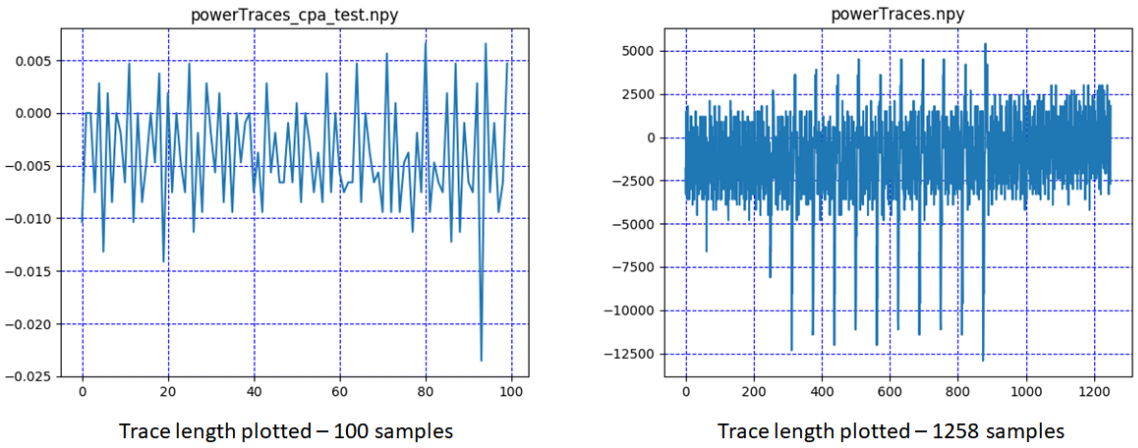

Figure 3.4: Measured power traces with two different sample sizes

be captured.

2. Access to PT: The attacker also has access to the plaintext(PT) which is used by the system.

3. Access to the AES implementation: The adversary needs to have an idea of how the AES has been implemented internally. This is needed to choose the appropriate attack point and decide whether only PT and power traces are sufficient to succeed the attack phase.

4. AES timing: It is helpful for the attacker to have access to the time it takes for the intermediate values to be processed and be available at the output of the point of attack. This has been discussed in the next section.

3.1.1.3 CPA Analysis

Figure 3.3 illustrates the process of how CPA analysis is performed on the system to derive the correct combination of key input. In design, the length of the plain text, cipher text

PT	Key Guesses						
0	0	1	2	3	4	.....	FF
11	0	1	2	3	4	.....	FF
22	0	1	2	3	4	.....	FF
33	0	1	2	3	4	.....	FF
44	0	1	2	3	4	.....	FF
55	0	1	2	3	4	.....	FF
66	0	1	2	3	4	.....	FF
77	0	1	2	3	4	.....	FF
88	0	1	2	3	4	.....	FF
99	0	1	2	3	4	.....	FF

(a)

XORed Values						
0	1	2	3	4	.....	FF
11	10	9	8	15	.....	EE
22	23	20	21	18	.....	DD
33	32	35	34	37	.....	CC
44	45	46	47	40	.....	BB
55	54	53	52	51	.....	AA
66	67	64	65	70	.....	99
77	76	79	78	73	.....	88
88	89	90	91	92	.....	77
99	98	97	96	103	.....	66

(b)

Binary Values						
0	1	10	11	100	.....	11111111
10001	10000	1001	1000	10101	.....	11101110
100010	100011	100000	100001	11000	.....	11011101
110011	110010	110101	110100	110111	.....	11001100
1000100	1000101	1000110	1000111	1000000	.....	10111011
1010101	1010100	1010011	1010010	1010001	.....	10101010
1100110	1100111	1100100	1100101	1110000	.....	10011001
1110111	1110110	1111001	1111000	1110011	.....	10001000
10001000	10001001	10010000	10010001	10010010	.....	1110111
10011001	10011000	10010111	10010110	100000011	.....	1100110

(c)

Hamming Weights						
0	1	1	.....	.....	.....	8
2	1	2	.....	.....	.....	6
2	3	1	.....	.....	.....	6
4	3	4	.....	.....	.....	4
2	3	3	.....	.....	.....	6
4	3	4	.....	.....	.....	4
4	5	3	.....	.....	.....	4
6	5	5	.....	.....	.....	2
2	3	2	.....	.....	.....	6
4	3	5	.....	.....	.....	4

(d)

Figure 3.5: Correlation Power Analysis (CPA) steps

Hamming Weights						
0	1	1	.....	.....	.....	8
2	1	2	.....	.....	.....	6
2	3	1	.....	.....	.....	6
4	3	4	.....	.....	.....	4
2	3	3	.....	.....	.....	6
4	3	4	.....	.....	.....	4
4	5	3	.....	.....	.....	4
6	5	5	.....	.....	.....	2
2	3	2	.....	.....	.....	6
4	3	5	.....	.....	.....	4
0x00	0x01	0x02	0x03	.....	.....	0xFF

Measured Power Traces						
.....	.....	1.2	.....	.....	.....	.....
.....	.....	2.3	.....	.....	.....	.....
.....	.....	1.5	.....	.....	.....	.....
.....	.....	4.8	.....	.....	.....	.....
.....	.....	3.7	.....	.....	.....	.....
.....	.....	4.6	.....	.....	.....	.....
.....	.....	3	.....	.....	.....	.....
.....	.....	5.5	.....	.....	.....	.....
.....	.....	2.2	.....	.....	.....	.....
.....	.....	5.6	.....	.....	.....	.....
0x00	0x01	0x02	0x03	.....	.....	0xFF

Figure 3.6: Example of correlation between measured and hypothetical power

and the key is 128 bits wide. The adversary must try to derive as much correct key bits as possible to break the cryptosystem security. Once the correct key has been derived, the

adversary gets access to the systems where the same key was used for providing security. There are some assumptions that are precursory to the tampering of the system which have been discussed previously. The CPA attack is one of the ways in which an adversary can gain access to the AES key. Referring to Figure 3.3, the attacker begins by deciding the point of attack. The point of attack is selected such that the value (output) available relates to the combination of the plaintext and the key (the attacker does not have access to the key, partial or whole). The block in conventional AES implementation that is chosen is the 'sbox' aka substitution box. The contents of this lookup table is open sourced and hence even the attacker has access to it. As discussed previously, the attacker has access to the time it takes for the data to reach the point of attack or the sbox in this case. Proceeding further, the power to the DUT FPGA is measured and stored. Power values corresponding to one full AES cycle are known as samples, whereas, the individual runs of the AES (with different test vector, with the same key) are known as traces; sample traces with different sample sizes is shown in Figure 3.4. The CPA then involves calculating the hypothetical intermediate and the hypothetical power, shown in Figure 3.5. The output of a sbox block is tried to mimic here to calculate the hypothetical power. Thing to note here is the attacker has no access to the actual key used and hence, it tries to generate all possible values (typically it is done byte wise, so a total of 256 possible values). After the output value of the sbox is known, by a combination of guessed key and plaintext, hamming weight or distance is calculated to represent power. This hypothetical power and the actual measured power are then correlated to see which power output value (hypothetical) corresponds strongly with the actual measured value and the correct key is the one that corresponds to that hypothetical value calculated previously; the process of correlation is shown in Figure 3.6. By iterating through this process a number of times the full key is derived.

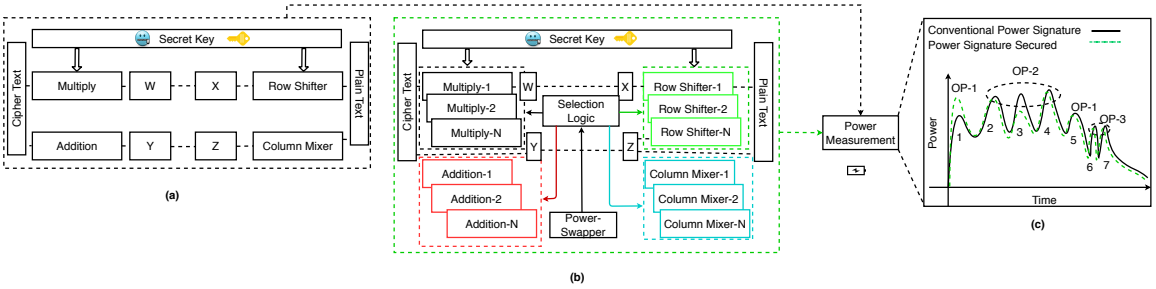

Figure 3.7: (a)Conventional cryptosystem where the attacker can deduce secret information from the observed power traces that correspond to the operations;(b) Cryptosystem protected by Power Swapper where the power trace does leak information but it leads to wrong information deduction by the attacker as standby modules aka approximate-modules add misleading information by leaking power traces, similar to other modules in magnitude, that are not known previously to the attacker; (c) Power signatures of the outputs of a conventional unprotected cryptosystem and a system protected by the proposed Power Swapper. *The figure is shown for visualization purposes and should not be used as an accurate depiction of a AES cryptosystem's internal structure*

3.2 Power-Swapper: Proposed Defense Against CPA-based Side-Channel Attack

The proposed Power Swapper has been outlined in Figure 3.7 where part (a) shows the internal structure of a conventional FPGA cryptosystem where each module has its own power consumption rating. The attacker then performs the power analysis on the system and then through statistical methods the adversary tries to deduce the secret key information. This is possible due to the fact that the instantaneous power consumption value would correspond to the operation of module. Based on these sequence of operations, the attacker can deduce the secret information. As shown in Figure 3.7(c), the power consumption waveform has seven peaks in total each corresponding to some operation. For instance, peak 1 and 5 have the same magnitude and inferred to belong to the same operation 'OP-1'; peaks 2, 3 and 4 belong to 'OP-2' and so on. The information leakage in this case is

maximum and it is highly correlating the power traces which can lead to leakage of secret information.

On the contrary, Figure 3.7(b) shows the cyrptosystem being secured by Power Swapper where the internally implemented functional modules still perform the same tasks as in a conventional FPGA shown previously except for the fact that there are other approximate-blocks that perform the same function but are designed in such a way that they have different power consumption ratings. These approximate blocks are chosen randomly during runtime by the selection logic which is controlled by the Power Swapper. Since each block still does the same task, there will not be any deviations in the functionality of the application.

Power-Swapper uses physically unclonable function (PUF) block within the Power Swapper to make the selection process random and unpredictable to the attacker. Figure 3.7(c) shows the waveform of the power traces corresponding to the proposed Power Swapper where some of the peaks show different values compared to the conventional FPGA power trace. Peak 1 which previously would give information of operation OP-1 now corresponds to OP-2 as per the attacker based on the power analysis.

Peak 3, which belonged to OP-2 now corresponds to the power consumption similar to OP-1. As can be seen, the victim is not altered yet the power traces are completely different and they are not known to and which mislead the attacker. Even if the attacker tries to study a large number of patterns to find the power trace modifications injected by the approximate modules, the efforts would become futile as the trace will keep sweeping between different power magnitudes due to the randomness derived from the PUF block. The power consumption magnitudes of the alternate implementations of basic blocks range from *p1* to *p5* where $p1 < p2 < p3 < p4 < p5$ and the range of the alternate block-1 performing the operations is in the range *p1 to p3*, while that of the block-2 would be in range of *p2 to p4* while yet another block would have it in *p3 to p5* range. As there is overlap in the power consumption range, one approximate block's power corresponds to some other block's range when they both are swapped during runtime. Hence, the attacker would be forced to deduce

the sequence of operations as two different ones whereas internally the same row shifting operation was performed with two different power signatures.

Refer to Figure 3.8 which shows the power trace of AES implementation without the proposed method. The trace was observed with the following parameters: DUT clock as 1 MHz, sampling frequency of 50 MHz and ADC sampling rate of 50MSps. If we observe the magnitude of the waveform and compare that with the magnitude shown in Figure 3.9, the change can be vividly seen owing to the approximate modules that perform essentially the same task but with a different power consumption. The way this disrupts the CPA power analysis is: the hypothetical intermediate that will be calculated by the adversary will remain the same (as discussed previously); key and the plaintext remain the same for a particular AES cycle. On the contrary, the actual power consumption would come out to be different and hence, if not for all the parts of the key but for some, parts of the key the adversary derives is incorrect disrupting the CPA analysis. Figure 3.9 illustrates an increase in the magnitude but it can also be the opposite depending how the approximate modules are designed. Hence, theoretically, if one harnesses the approximate modules for enhancing security in FPGA based crypto implementations, power analysis based side channel attacks could be thwarted.

3.2.1 Experimental Results

The experimental setup used for capturing AES traces (without Power Swapper) was: 1. Artix-7 based FPGA controller, CW305 Artix FPGA Target DUT board, PicoScope 5000 series for capturing the power traces, system clock frequency used was 1 MHz. The cyrpto application implemented on the DUT board was AES [1] with 128 bits of plaintext, key and output cipher text. Pearson correlation was used to calculate hypothetical power. Automation scripts were used to provide test vectors to the DUT and 1 million traces were collected for analysis.

Refer to Table 3.1 for the power traces observed by the attacker with Power Swapper. As can be seen from table, the information deduced by the attacker based on power consumption

values are completely different compared to the actual operation executed on the core. The modified power signatures observed by the attacker are highlighted in red. The modified signatures are a result of the Power Swapper choosing one of the approximate blocks. Similarly, referring to Table 3.2, the key derived using CPA without and with Power Swapper has been shown. The bytes/nibble that were wrongly correlated to the key guesses - as described previously - have been highlighted. These wrong portions of the keys are observed as the effect of the approximate modules introduced by Power Swapper . It is to be noted that the length of the plaintext and key does not in anyway affect the efficacy of the proposed method. The results in the Tables 3.1, 3.2 provide sufficient proof that by employing approximate modules, cryptosystems can be rendered resilient against power analysis based side-channel attacks.

Ovehead Analysis: As with every system, the proposed methodology will also have overheads. The Power Swapper requires that approximate modules be added to the original implementation of AES and these modules will be selected during the application execution. Needless to say, the modules will require additional space on the FPGA fabric along with some increase in power consumption. Switching between these modules will also lead to small overheads. The small, if not insignificant, overhead would be the trade off between security and power/area. The resource utilization for two variants of AES is shown in Table 3.3 and 3.4. One variant is implemented as a base version, without any pragmas in Vivado HLS 2019; Table 3.3 presents the resource utilization results. Another variant is implemented using pipeline and loop unroll pragmas; the results are presented in Table 3.4. Both variants essentially function the same except that the power signatures are different. The total hardware resource utilization will be decided based on how many variants of AES, or blocks within AES, is utilized to include a range of power signatures for each block. The results presented in Table 3.3 and 3.4 are for reference only; they will scale based on the security to area/power tradeoff chosen as per the required resiliency against attack.

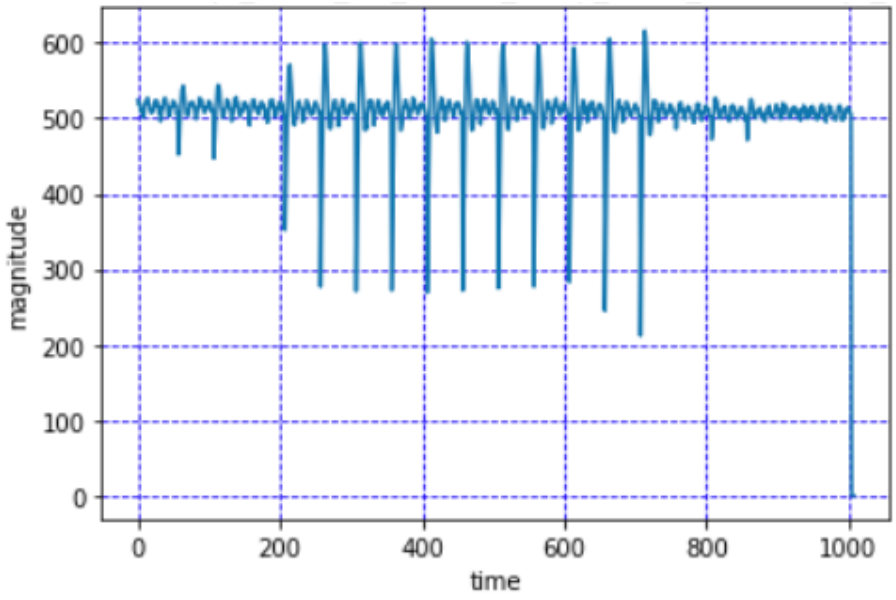

Figure 3.8: Power trace of AES engine implemented on FPGA

Table 3.1: Impact of Power Swapper on power trace extraction

Scenario	**Victim Power Trace**	**Power Trace with Power Swapper**	
		Instance-1	**Instance-2**
Scenario-1	OP-1/OP-2/OP-3/OP-4	OP-3/OP-2/OP-1/OP-4	OP-2/OP-3/OP-4/OP-1
Scenario-2	OP-1/OP-1/OP-4/OP-3	OP-2/OP-3/OP-2/OP-1	OP-3/OP-1/OP-1/OP-3

3.2.2 Conclusion and Future Work

In this work, I discussed the physical power SCAs, discussed the severity of the threats posed and delineated the works in the past. In contrast to the existing works, proposed Power Swapper will preserve the victim's secret information without any modifications to the victim algorithm in itself. I hope the community will be intrigued by the preliminary

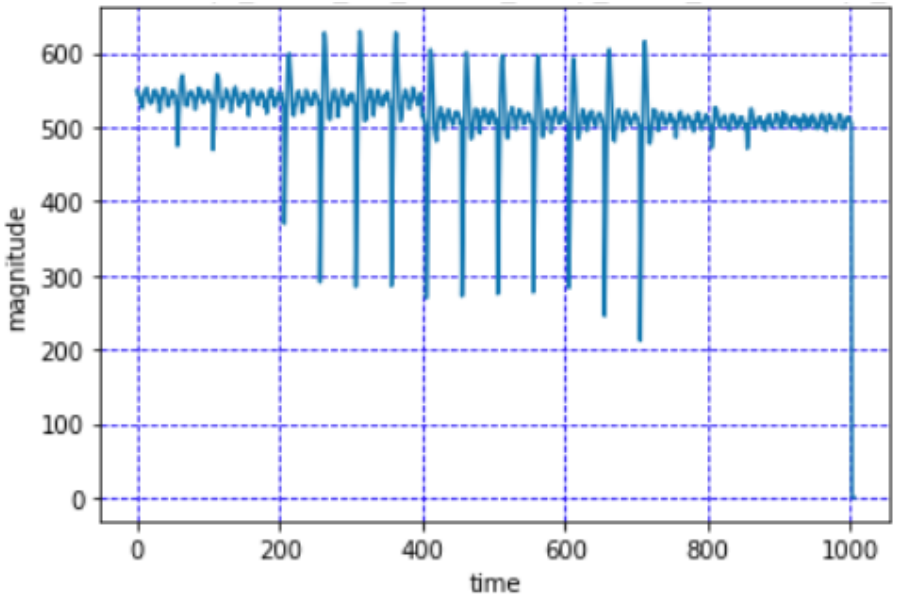

Figure 3.9: Power trace of AES controlled by Power Swapper . The magnitude is seen to increase(in this case) given the presence of approximate functional blocks

Table 3.2: Impact of Power Swapper on keys

Trace #	**Correct key**	**Incorrect key with Power Swapper**
Trace-1	51720187c36e0c8523acb8535a870703	51522187ca6ea28523acb8e35a870793
Trace-2	d14a900c7391d64101fe33a85b0793cb	a14a90dc7391d63201fe33a85b1693cb

results discussed in this work and I plan to develop this work in future to deliver more details of the mechanism that would benefit the security critical processes.

Table 3.3: Resource utilization of AES without pragmas implemented on a Zedboard FPGA

Name	BRAM 18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	64	-
FIFO	-	-	-	-	-
Instance	0	-	1245	6648	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	183	-
Register	0	-	1311	32	-
Total	0	0	2556	6895	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	2	12	0

Table 3.4: Resource utilization of AES with loop unroll and pipeline pragmas implemented on a Zedboard FPGA

Name	BRAM 18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1478	-
FIFO	-	-	-	-	-
Instance	0	-	4356	16698	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	162	-
Register	0	-	3717	32	-
Total	0	0	8073	18370	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	7	34	0

Acknowledgement

I would like to thank Dr.Jens-Peter Kaps and Dr. Abubakr Abdulgadir from Cryptographic Engineering Research Group - George Mason University for providing access to the FOBOS

[1] setup for power-side channel analysis, and offering all the kind help in configuring FOBOS for the experiments.

Chapter 4: Conclusion

This thesis discussed the importance of security to computing systems. We discussed the attacks, like reverse engineering, threatening the security community and their capabilities to disrupt design confidentiality. Further, defense against IC reverse engineering attacks was discussed which includes the automation application for logic locking defense and robustness evaluation against attacks, followed by the design of hardware communication modules. Further, physical side-channel attack on crypto systems on FPGA was discussed in detail along with a proposed defense against such attack. The thesis also includes full working code for the automation application and hardware module.

Bibliography

Bibliography

[1] A. Abdulgadir, W. Diehl, and J.-P. Kaps, "An open-source platform for evaluating side-channel countermeasures in hardware implementations of lightweight authenticated ciphers," in *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, Cancun, Mexico, Dec 2019.

[2] R. Velegalati and J.-P. Kaps, "Towards a Flexible, Opensource BOard for Side-channel analysis (FOBOS)," Cryptographic architectures embedded in reconfigurable devices, CRYPTARCHI 2013, June 2013.

[3] S. Bavikadi, A. Dhavlle, A. Ganguly, A. Haridass, H. Hendy, C. Merkel, V. J. Reddi, P. R. Sutradhar, A. Joseph, and S. M. P. Dinakarrao, "A Survey on Machine Learning Accelerators and Evolutionary Hardware Platforms," *IEEE Design & Test*, pp. 1–1, 2022.

[4] A. Dhavlle, R. Hassan, M. Mittapalli, and S. M. P. Dinakarrao, "Design of hardware trojans and its impact on cps systems: A comprehensive survey," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[5] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, Aug 2014.

[6] R. Torrance and D. James, "The state-of-the-art in semiconductor reverse engineering," in *Proceedings of the 48th Design Automation Conference*, ser. DAC '11. New York, NY, USA: ACM, 2011, pp. 333–338. [Online]. Available: http://doi.acm.org/10.1145/2024724.2024805

[7] M. M. Tehranipoor, U. Guin, and S. Bhunia, "Invasion of the hardware snatchers," *IEEE Spectr.*, vol. 54, no. 5, pp. 36–41, May 2017. [Online]. Available: https://doi.org/10.1109/MSPEC.2017.7906898

[8] J. A. Roy, F. Koushanfar, and I. L. Markov, "Epic: Ending piracy of integrated circuits," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '08. New York, NY, USA: ACM, 2008, pp. 1069–1074. [Online]. Available: http://doi.acm.org/10.1145/1403375.1403631

[9] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri, "Fault analysis-based logic encryption," *IEEE Transactions on Computers*, vol. 64, no. 2, pp. 410–424, Feb 2015.

[10] Y. Xie and A. Srivastava, "Anti-sat: Mitigating sat attack on logic locking," *IEEE*

Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 2, pp. 199–207, Feb 2019.

[11] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, May 2015, pp. 137–143.

[12] F. Imeson, A. Emtenan, S. Garg, and M. Tripunitara, "Securing computer hardware using 3d integrated circuit (IC) technology and split manufacturing for obfuscation," in *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*. Washington, D.C.: USENIX, 2013, pp. 495–510. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/imeson

[13] S. Shukla, G. Kolhe, S. M. P D, and S. Rafatirad, "Stealthy malware detection using rnn-based automated localized feature extraction and classifier," in *International Conference on Tools with Artificial Intelligence (ICTAI)*, 2019.

[14] S. Shukla, G. Kolhe, S. M. PD, and S. Rafatirad, "Rnn-based classifier to detect stealthy malware using localized features and complex symbolic sequence," in *International Conference On Machine Learning And Applications (ICMLA)*, 2019.

[15] S. Shukla, G. Kolhe, S. M. P. Dinakarrao, and S. Rafatirad, "On-device Malware Detection using Performance-aware and Robust Collaborative Learning," *Design Automation Conference (DAC)*, 2021.

[16] A. Dhavlle, S. Shukla, S. Rafatirad, H. Homayoun, and S. M. Pudukotai Dinakarrao, "Hmd-hardener: Adversarially robust and efficient hardware-assisted runtime malware detection," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 1769–1774.

[17] G. Kolhe and et.al., "Security and complexity analysis of lut-based obfuscation: From blueprint to reality," in *Int. Conference On Computer Aided Design*, 2019.

[18] Z. Chen, G. Kolhe, and et.al, "Estimating the circuit deobfuscating runtime based on graph deep learning," in *Design, Automation and Test in Europe Conference (DATE)*, 2020.

[19] S. M. P. Dinakarrao, S. Amberkar, S. Bhat, A. Dhavlle, H. Sayadi, A. Sasan, H. Homayoun, and S. Rafatirad, "Adversarial attack on microarchitectural events based malware detectors," in *Design Automation Conference*, 2019.

[20] S. Shukla, G. Kolhe, S. M. P. D, and S. Rafatirad, "Microarchitectural events and image processing-based hybrid approach for robust malware detection: Work-in-progress," in *Proceedings of the International Conference on Compliers, Architectures and Synthesis for Embedded Systems Companion*, 2019.

[21] A. Dhavlle and S. Shukla, "A novel malware detection mechanism based on features extracted from converted malware binary images," 2021.

[22] S. Barve, S. Shukla, S. M. P. Dinakarrao, and R. Jha, "Adversarial Attack Mitigation Approaches using RRAM Neuromorphic Architectures," *GLSVLSI*, 2021.

[23] M. Meraj Ahmed, A. Dhavlle, N. Mansoor, P. Sutradhar, S. M. Pudukotai Dinakarrao, K. Basu, and A. Ganguly, "Defense against on-chip trojans enabling traffic analysis attacks," in *2020 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, 2020, pp. 1–6.

[24] M. M. Ahmed, A. Dhavlle, N. Mansoor, S. M. P. Dinakarrao, K. Basu, and A. Ganguly, "What can a remote access hardware trojan do to a network-on-chip?" in *International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[25] F. Brasser, L. Davi, A. Dhavlle, and et al., "Advances and throwbacks in hardware-assisted security: Special session," in *Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2018.

[26] A. Dhavlle, S. Rafatirad, K. Khasawneh, H. Homayoun, and S. M. P. Dinakarrao, "Imitating functional operations for mitigating side-channel leakage," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 868–881, 2022.

[27] A. Dhavlle, R. Mehta, S. Rafatirad, H. Homayoun, and S. M. P. D, "Entropyshield:side-channel entropy maximization for timing-based side-channel attacks," in *21 st International Symposium on Quality Electronic Design (ISQED)*, 2020.

[28] P. Yu and P. Schaumont, "Secure fpga circuits using controlled placement and routing," in *IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*, 2007.

[29] Y. Ishai, A. Sahai, and D. Wagner, "Private circuits: Securing hardware against probing attacks," in *Advances in Cryptology*, 2003.

[30] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '99, 1999.

[31] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *Advances in Cryptology — CRYPTO' 99*, M. Wiener, Ed., 1999.

[32] D. Suzuki, M. Saeki, and T. Ichikawa, "Random switching logic: A new countermeasure against dpa and second-order dpa at the logic level," *IEICE Transactions*, vol. 90-A, pp. 160–168, 01 2007.

[33] E. Trichina, "Combinational logic design for aes subbyte transformation on masked data," IACR report, Tech. Rep., 2003.

[34] Shengqi Yang, W. Wolf, N. Vijaykrishnan, D. N. Serpanos, and Yuan Xie, "Power attack resistant cryptosystem design: a dynamic voltage and frequency switching approach," in *Design, Automation and Test in Europe*, 2005.

[35] A. Dubey, R. Cammarota, and A. Aysu, "Maskednet: The first hardware inference engine aiming power side-channel protection," in *International Symposium on Hardware Oriented Security and Trust (HOST)*, 2020, pp. 197–208.

[36] R. Matovu, A. Serwadda, A. V. Bilbao, and I. Griswold-Steiner, "Defensive charging: Mitigating power side-channel attacks on charging smartphones," in *Conference on Data and Application Security and Privacy*. Association for Computing Machinery, 2020, p. 179–190.

[37] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 709–720. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516656

[38] A. Vijayakumar, V. C. Patil, D. E. Holcomb, C. Paar, and S. Kundu, "Physical design obfuscation of hardware: A comprehensive investigation of device and logic-level techniques," *Trans. Info. For. Sec.*, vol. 12, no. 1, pp. 64–77, Jan. 2017. [Online]. Available: https://doi.org/10.1109/TIFS.2016.2601067

[39] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Camoperturb: secure IC camouflaging for minterm protection," in *Proceedings of the 35th International Conference on Computer-Aided Design, ICCAD 2016, Austin, TX, USA, November 7-10, 2016*, 2016, p. 29.

[40] A. Dhavlle, S. Bhat, S. Rafatirad, H. Homayoun, and S. M. P. D, "Work-in-progress: Sequence-crafter: Side-channel entropy minimization to thwart timing-based side-channel attacks," in *Conference on Compliers, Architectures and Synthesis for Embedded Systems (CASES)*, 2019.

[41] F.-X. Standaert, *Introduction to Side-Channel Attacks*, 2010.

[42] F. Koeune and F.-X. Standaert, *A Tutorial on Physical Security and Side-Channel Attacks*, 2005.

[43] C. Shepherd, K. Markantonakis, N. van Heijningen, D. Aboulkassimi, C. Gaine, T. Heckmann, and D. Naccache, "Physical fault injection and side-channel attacks on mobile devices: A comprehensive analysis," *Computers & Security*, vol. 111, p. 102471, 2021.

[44] M. A. Al Faruque, S. R. Chhetri, A. Canedo, and J. Wan, "Acoustic side-channel attacks on additive manufacturing systems," in *International Conference on Cyber-Physical Systems (ICCPS)*, 2016, pp. 1–10.

[45] S. S. Mirzargar and M. Stojilović, "Physical side-channel attacks and covert communication on fpgas: A survey," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2019, pp. 202–210.

Curriculum Vitae

Abhijitt Dhavlle pursued his diploma in industrial electronics from Fr.Agnel polytechnic, India, followed by Bachelor of Engineering in Electronics and Telecommunication from Mumbai University, India; Later, Abhijitt pursued his Master of Science in Computer Engineering from George Mason University, Virginia. Abhijitt interned at University of Southern California Information Sciences Institute, Arlington, and later interned at Xilinx, Inc., San Jose. Abhijitt's research interests are side-channel attacks, physical side-channel attacks, machine learning and deep learning, and hardware accelerators for machine learning.

Abhijitt Dhavlle

Electrical and Computer Engineering, George Mason University, Fairfax, VA, USA 22031
571-320-7417 | adhavlle@gmu.edu | LinkedIn Profile

Computer engineering graduate with over 4 years of professional and internship experience. Strong background in Applied Deep Learning and Machine Learning, Python, EDA analysis, Embedded Systems, Digital System Design, Hardware Security, and Computer Architecture. Experience in GUI designing, sensor interfacing and monitoring, MATLAB scripting, and Deep Learning Toolbox.

Education

George Mason University, Fairfax, Virginia **Expected August 2022**
Ph.D. in Computer Engineering - Ph.D. Candidate

George Mason University, Fairfax, Virginia **May 2022**
Master of Science in Computer Engineering

University of Mumbai, India **May 2014**
Bachelor of Engineering in Electronics and Telecommunication Engineering

Experience

Application Test Development Intern, Radiation Effects Team, Xilinx Inc. **May 2021 – August 2021**

Design and Integration of Radiation Tolerant Testbench for New Gen. (Versal) FPGAs

- Developed a code for continuous logging of Deep Learning Processing Unit (DPU) Inference on ImageNet dataset for testing.
- Integrated Error Correction Code IP for mitigating radiation induced errors.
- Testing of Fault Aware Training (FAT) model for radiation tolerant testbench.

Technologies: Xilinx Vivado, Xilinx Vitis-AI, Python, Deep Learning

Graduate Research Assistant, George Mason University, Fairfax **January 2019 – Present**

Machine Learning for Side-Channel Attack Evaluation

- Evaluated Neural Network, MLP, Logistic Regression, SVM, and Random Forest ML models on a novel side-channel attack.
- Developed a Python code for automating the attack and Hardware Intrusion Detection system -based defense interaction.
- Automated the generation of perturbed instances of the attack for defense evasion and evaluation of attack performance.

Technologies: Machine Learning, Deep Learning, Python, Pandas, Numpy, Scipy, Sklearn

ML-assisted Off-Chip Hardware Trojan Payload Analysis

- Developed a Artificial Neural Network (ANN) based model to detect applications running on the processor based on the communication packets observed by a hardware Trojan in the Network-on-Chip (NoC).
- Developed a Python code for utilizing custom one-hot encoded inputs and outputs for ANN model.
- Developed a Artificial Neural Network (ANN) model to detect NOC architecture and applications running on a processor from the perspective of an inserted hardware Trojan.

Technologies: Deep Learning, Machine Learning, Python, Pandas, Scikit-learn, Numpy, Seaborn

Machine Learning for Malware Detection using Performance Counters and Network Data

- Utilized Machine Learning models for developing a hardware-based intrusion detector for malware classification.
- Developed the process flow on a IoT device running malware and benign application suite. The ML-based intrusion detector utilized network traffic data and hardware attributes for classification.

Technologies: Machine Learning, Regression, Ensemble Learning, Python, Pandas, Scikit-learn, Numpy, Seaborn

Machine learning assisted Hardware-based Malware Detection

- Extracted hardware performance counters corresponding to various hardware events on Intel-based system running plethora of malware and benign applications.
- Performed exploratory data analysis and component selection to determine predominant events for different classes of malware family.
- Implemented decision tree, multi-layer perceptron, random forest, support vector machine, K-nearest neighbor, and neural network to classify malware and benign applications.
- Implemented classifiers in RTL for FPGA inference. Reported power and area analysis of the implemented classifiers.

Technologies: Deep Learning, Machine Learning, Python, Pandas, Scikit-learn, Numpy, Xilinx Vivado, MATLAB

Hardware Research Intern at University of Southern California - Information Sciences Institute **May 2020 – August 2020**

Power Analysis Side-Channel Attack for Breaking FPGA Bitstream Encryption

- Development and analysis of side-channel attack techniques for Virtex Series FPGAs to decipher secret key to allow bitstream readback for reprogrammability and portability.
- Contributed as a beta tester for verifying WBSTAR attack on FPGA to retrieve encrypted bitstream.

Technologies: Jupyter Notebook, Python, Microsoft Excel, VHDL, Xilinx Vivado, *ChipWhisperer* for Side-Channel Analysis

Technical Skills

Languages: Python, C/C++, MATLAB, Verilog, VHDL
Hardware platforms: Digilent Zedboard, Zybo, Baysy3; Zynq UltraScale MPSoC ZCU102, Raspberry Pi 3/4
ML libraries and frameworks: TensorFlow, Keras, WEKA, scikit-learn, numpy, pandas, matplotlib, scipy
IDE and Tools: Docker, Mobaxterm, PyQt, PyCharm, Jupyter notebook, MS-Office suite, MS-Excel, Simulink, Deep Learning Toolbox
Design, Synthesis and Simulation Tools: VHDL, Verilog, Synopsys Design Compiler, Prime Time, IC Compiler, Xilinx ISE, Vivado HLS
Miscellaneous Tools: Digital oscilloscopes, Logic analyzers, Protocol Analyzers, Power supplies, PCB design tools, Autodesk Fusion 360 CAD tool; USB, I2C, SPI, UART protocols; Soldering; Linux

Relevant Coursework

Digital System Design with VHDL, Computer Architecture, Operating Systems, Advanced Microprocessor & Microcontrollers, Applied Machine Learning, Object-Oriented Programming; Instructed VHDL, IoT Design with Raspberry Pi, H/w Accelerators for Machine Learning.

Selected Publications

1. **Abhijitt Dhavlle**, S. Shukla, S. Rafatirad, H. Homayoun, and S. M. PD. - "HMD-hardener: Adversarially robust and efficient hardware-assisted runtime malware detection" - Design, Automation Test in Europe Conference Exhibition (DATE), 2021
2. **Abhijitt Dhavlle** , Rakibul Hassan, Manideep Mittapalli, and Sai Manoj PD - "Design of Hardware Trojans and its Impact on CPS Systems: A Comprehensive Survey" - International Symposium on Circuits and Systems (ISCAS), 2021
3. **Abhijitt Dhavlle** and Sai Manoj Pudukotai Dinakarrao - "A Comprehensive Review of ML-based Time-Series and Signal Processing Techniques and their Hardware Implementations" - International Green and sustainable computing Conference (IGSC), 2020
4. M Meraj Ahmed, **Abhijitt Dhavlle**, Naseef Mansoor, Purab Sutradhar, Sai Manoj Pudukotai Dinakarrao, Kanad Basu and Amlan Ganguly -"Defense against On-Chip Trojans Enabling Traffic Analysis Attacks" - AsianHOST, 2020
5. Sai Manoj PD, Sairaj Amberkar, Sahil Bhat, **Abhijitt Dhavlle** et al. - "Adversarial Attack on Microarchitectural Events based Malware Detectors" - Design Automation Conference (DAC), 2019