

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE
ENGINEERING AND TECHNOLOGY

**DEVELOPMENT OF SIDE CHANNEL ANALYSIS ENVIRONMENT
USING SIMULATION DATA OF SYSTEM-ON-CHIP PROCESSORS**

M.Sc. THESIS

Yasin Firat KULA

Department of Electronics and Communication Engineering

Electronics Engineering Programme

JUNE 2019

**DEVELOPMENT OF SIDE CHANNEL ANALYSIS ENVIRONMENT
USING SIMULATION DATA OF SYSTEM-ON-CHIP PROCESSORS**

M.Sc. THESIS

**Yasin Fırat KULA
(504161226)**

Department of Electronics and Communication Engineering

Electronics Engineering Programme

Thesis Advisor: Assoc. Prof. Dr. Sıddıka Berna Örs Yalçın

JUNE 2019

**KIRMIK ÜSTÜ SİSTEM İŞLEMCİLERİNİN BENZETİM
VERİLERİ İLE YAN KANAL ANALİZİ ORTAMI GELİŞTİRİLMESİ**

YÜKSEK LİSANS TEZİ

**Yasin Fırat KULA
(504161226)**

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

Tez Danışmanı: Assoc. Prof. Dr. Sıddıka Berna Örs Yalçın

HAZİRAN 2019

Yasin Fırat KULA, a M.Sc. student of ITU Graduate School of Science Engineering and Technology 504161226 successfully defended the thesis entitled “DEVELOPMENT OF SIDE CHANNEL ANALYSIS ENVIRONMENT USING SIMULATION DATA OF SYSTEM-ON-CHIP PROCESSORS”, which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor : **Assoc. Prof. Dr. Sıddıka Berna Örs Yalçın**
Istanbul Technical University

Jury Members : **Prof. Dr. Güneş Karabulut Kurt**
Istanbul Technical University

Asst. Prof. Dr. Tuba Ayhan
MEF University

.....

Date of Submission : **30 April 2019**
Date of Defense : **21 June 2019**





To my dear family,



FOREWORD

First of all, I present great thanks to my project advisor Assoc. Prof. Dr. Sıddıka Berna Örs Yalçın for her invaluable information and infinite support in the course of the thesis, also for endearing research to me yet again.

Further thanks to Dr. Tuba Ayhan and Levent Aksoy, for thier support and provided insight during my postgraduate program.

Lastly, I present my eternal gratitude to my parents and my sister, who have always been with me from the beginning to the present, and bringing me where I am now.

June 2019

Yasin Fırat KULA



TABLE OF CONTENTS

	<u>Page</u>
FOREWORD	ix
TABLE OF CONTENTS	xi
ABBREVIATIONS	xiii
SYMBOLS	xv
LIST OF TABLES	xvii
LIST OF FIGURES	xix
SUMMARY	xxi
ÖZET	xxiii
1. INTRODUCTION	1
2. PULPINO PLATFORM	5
2.1 General Information	5
2.2 PULPino Working Environment.....	7
2.2.1 Prerequests for the environment	9
2.2.2 Setting up the environment.....	13
2.2.3 Behavioral simulation.....	15
2.2.4 Adding custom applications	18
2.2.5 Building FPGA related content	19
2.2.6 Potential problems and solutions.....	20
3. POST-IMPLEMENTATION SIMULATION AND DYNAMIC POWER ESTIMATION ON PULPINO	23
3.1 Preparing for Post-Implementation	23
3.1.1 First testbench.....	23
3.1.2 Editing the boot ROM	25
3.2 Post-Implementation Simulation with Official PULPino Testbench.....	26
3.3 Post-Implementation Average Dynamic Power Estimation	29
3.3.1 SAIF files and VCD files.....	30
3.3.2 Average dynamic power consumption estimation with SAIF	31
4. MOMENTARY POWER CONSUMPTION PROFILE GENERATION	33
4.1 Generating Momentary Power Consumption Profiles using VCD.....	33
4.2 Processing the Value Dump Files	35
4.3 Momentary Power Consumption Profile of an Advanced Encryption Standart Algorithm	39
4.3.1 Overview of Advanced Encryption Standart.....	39
4.3.2 Masking Advanced Encryption Standart.....	42
5. DIFFERENTIAL POWER ANALYSIS USING MOMENTARY POWER CONSUMPTION PROFILES	45
5.1 Differential Power Analysis	45

5.2 Test Attacks and Results.....	47
5.2.1 Attack on Initial XOR Stage.....	47
5.2.2 Attack on Initial SubBytes Stage.....	49
5.2.2.1 Results on standart AES	50
5.2.2.2 Results on byte-Masked AES	51
6. CONCLUSION	53
REFERENCES.....	55
APPENDICES	59
APPENDIX A.1	60
APPENDIX A.2	61
APPENDIX A.3	63
APPENDIX A.4	66
APPENDIX A.5	67
CURRICULUM VITAE.....	69



ABBREVIATIONS

NIST	: National Institute of Standards and Technology
AES	: Advanced Encryption Standard
DES	: Data Encryption Standard
TDES	: Triple Data Encryption Standard
DPA	: Differential Power Analysis
HDL	: Hardware Description Language
RISC	: Reduced Instruction Set Computer
PULP	: Parallel Ultra-Low Power
IoT	: Internet of Things
RAM	: Random Access Memory
ROM	: Read Only Memory
SPI	: Serial Peripheral Interface
FPGA	: Field Programmable Gate Array
GUI	: Graphical User Interface
RTL	: Register-Level Transfer
IP	: Intellectual Property
UART	: Universal Asynchronous Receiver-Transmitter
SSH	: Secure Shell
SAIF	: Switching Activity Interchange File
VCD	: Value Change Dump
UUT	: Unit Under Test
ALU	: Arithmetic Logic Unit
XOR	: Exclusive Or
GPIO	: General Purpose Input/Output



SYMBOLS

C_i	: Capacitance of line i
f_i	: Switching frequency of line i
V_i	: Voltage swing on line i
P	: Dynamic power consumption
k	: Number of clock pulses taken
n	: Number of tests performed
P_{ij}	: Switching number per clock cycle matrix
P_{avg}	: Average switching number matrix
S	: Switching weight matrix



LIST OF TABLES

	<u>Page</u>
Table 3.1 : Post-implementation resource usage of PULPino on Artix-7 Development Board.	23
Table 3.2 : Structural properties of used benchmarks.	31
Table 3.3 : Simulation times and run times for different benchmarks (25 MHz clock).....	32
Table 3.4 : Average power consumption results obtained by Vivado, using SAIF files.	32



LIST OF FIGURES

	<u>Page</u>
Figure 2.1 : The PULP Project family [1].	5
Figure 2.2 : The RI5CY core top level schematic [2].	6
Figure 2.3 : The zero-risc core top level schematic [2].	6
Figure 2.4 : The PULPino Platform top level schematic [2].	7
Figure 2.5 : The build flow of the PULPino environment [3].	8
Figure 2.6 : Example of a correctly edited .bashrc file.	10
Figure 2.7 : Terminal output for a succesful IP update.	14
Figure 2.8 : Expected terminal output after a succesful environment build.	15
Figure 2.9 : Terminal output when all the PULPino RTL is succesfully compiled.	16
Figure 2.10 : Output of the helloworld program in ModelSim.	17
Figure 2.11 : Example definitions in a CMake recognition file used in an encryption application.	19
Figure 2.12 : Top module of PULPino ZedBoard Implementation [3].	20
Figure 2.13 : Correctly edited “vsim.tcl“ to solve C compiler issue.	21
Figure 3.1 : Structure of first testbench.	24
Figure 3.2 : PULPino memory map [2].	25
Figure 3.3 : The modelsim.ini file with added Xilinx libraries.	26
Figure 3.4 : Contents of an example simulation folder.	28
Figure 4.1 : A cut view from a tabular value dump file belonging to PULPino simulation.	36
Figure 4.2 : Momentary switching profile for arithmetic logic unit.	37
Figure 4.3 : Momentary switching profile for core registers.	38
Figure 4.4 : Momentary switching profile of ALU (zoomed).	38
Figure 4.5 : Momentary switching profile of core registers (zoomed).	38
Figure 4.6 : Block schematic of a 128-bit AES algorithm [4].	40
Figure 4.7 : The SubByte operation [4].	41
Figure 4.8 : The Shift Rows operation [4].	41
Figure 4.9 : Operating of Mix Columns [4].	41
Figure 4.10 : Momentary power consumption profile of entire AES encryption stage for one message.	42
Figure 4.11 : Overview of basic masking [5].	43
Figure 5.1 : Switching activity profile of AES initial XOR operation for one message, with respect to clock pulses.	46
Figure 5.2 : Initial XOR attack results for plaintext numbers of 100,200 and 300.	48
Figure 5.3 : Initial XOR attack result for 4000 plaintext trials.	49
Figure 5.4 : Initial SubByte attack result for 2000 plaintext trials.	50

Figure 5.5 : Correlation profiles of possible key values for different number of plaintexts..... 51

Figure 5.6 : Initial SubByte attack result for 2000 plaintext trials..... 52

Figure 5.7 : Correlation profiles of possible key values for different number of plaintexts..... 52



DEVELOPMENT OF SIDE CHANNEL ANALYSIS ENVIRONMENT USING SIMULATION DATA OF SYSTEM-ON-CHIP PROCESSORS

SUMMARY

In the current era, with the drastic development in electronics, computer and communication technologies; the concept of digitalization became a part of daily life. This concept brought quite a lot improvements on data transferring and data communication, in terms of easiness and speed. This also brought forth the concern of data safety during the process. Cryptology, which is defined as the discipline of data integrity nowadays, steps in to cope with this problem. In this manner, cryptology offers ways to encrypt and protect the transferred and stored data.

Developed encryption algorithms are desired to be standardized worldwide. For example, a standard belonging The National Institute of Technology and Standards of United States of America (NIST), the Advanced Encryption Standard (AES) is a good living example of this, employing an algorithm originally named Rijndael. AES replaced their precursors, Data Encryption Standard and Triple Data Encryption Standard in 1999, offering better security and performance, and it is still in use.

Softcore processors are yet another topic that has a large popularity in embedded system applications in the latest years. Their aspect that stands out is their flexibility, coming mostly from being open source and editable designs. Black box microprocessor based products may not perform ideally for all application areas, and creating a custom design just for a specific application may not be cost effective. Hence, open source mentality is adapted to the hardware section to cope with the flexibility issues. Designers release their product openly and allow users to examine and edit them. This lets users to use whatever sections of design is needed or expand the design according to their application areas. Users can also contribute to the process by reporting bugs or offering valid extensions.

In the scope of this thesis, the two aforementioned concepts are planned to be used together. A softcore processor system is chosen and data for Differential Power Analysis (DPA) on AES is planned to be taken using this model. The claim is to show that real time power consumption measurements are not required necessarily, simulation level switching information can also be used instead. Verifying this would allow to test an algorithm against side channel attacks during simulation stage, without needing a realization. This would also remove the time consuming real time measurement stage. In this manner, this work showed that the claim is valid and it is possible to obtain acceptable results for DPA during simulation stage. Along with this, a softcore processor system was used to generate inputs; to obtain a working experience with open source systems and to be a further reference for our research.

During the course of work, open source softcore processor system named PULPino was used. First part of the work explains this design in detail, and shows how to use its working environment, to be a reference for further works.

It is firstly thought to make post-implementation simulations for PULPino to get more close to real test results for DPA. Implementation of PULPino is done for Artix-7 device in Xilinx Vivado. The resulting post-implementation model is used in ModelSim tool, along with PULPino testbench and simulations are performed for various benchmarks. In the next step, obtaining average dynamic power consumption and momentary switching profile generation is discussed. Some average dynamic power consumption estimations are done as an extra, for various bechmarks. Then, momentary switching activities are recorded for behavioral and post implementation simulations of different benchmarks. Post-implementation outputs being unefficiently large led to behavioral model activity dumps to be used. The expectation was to see that behavioral switching activity would be stil sufficient to explicit important areas of execution.

Since none of the freely available tools that is tried offered a momentary power consumption graphic output, MATLAB codes to performs this task are written from scratch. The power consumption profile technique is tried on several benchmarks and AES encryption and it is observed that they provide the critical section information on the profiles as expected. Then, using a large number of different plain text messages AES application is simulated and outputs are used for DPA and correlation analysis. Results of correlation analysis allows to find the exact value of the key section that is used, or provides a shrinked estimation groups for the key. Both of these means that the algorithm can be cracked, and proves the offered technique is valid.

KIRMIK ÜSTÜ SİSTEM İŞLEMCİLERİNİN BENZETİM VERİLERİ İLE YAN KANAL ANALİZİ ORTAMI GELİŞTİRİLMESİ

ÖZET

Günümüzde elektronik, bilgisayar ve haberleşme teknolojilerinin hızla gelişmesiyle insanların yaşamına girmiş olan "dijitalleşme" kavramı, verinin işlenmesi ve taşınması açısından birçok avantaj getirmiştir. Dijital ortamda veri çok hızlı ve kolay bir şekilde taşınabilmektedir. Fakat bu kolaylık, bilginin elektronik olarak taşınması ve saklanması sırasında verinin güvenliği ile ilgili endişe de doğurmuştur. Bu sorunun çözümü için, eski tarihlerden beri çeşitli biçimlerde varlığını sürdüren kriptoloji bilimi öne çıkmaktadır. Varlığının ilk dönemlerde kriptoloji tabiri, şifre üretme işini çağrıştırırken günümüzde kriptografi ve kriptanaliz alt disiplinleri ile veri bütünlüğünün korunması probleminde yoğunlaşan bir bilim dalı durumundadır. Kriptoloji, verinin bir uçtan diğerine taşınması sırasında güvenliği sağlamak için çeşitli şifreleme, saklama ve analiz yöntemleri ve algoritmaları önerir.

Geliştirilen şifreleme algoritmaları ülkeler tarafından standartlaştırılmaktadır. Amerika'nın Ulusal Standartlar ve Teknoloji Enstitüsü'ne (NIST) ait standartlardan olan Gelişmiş Şifreleme Standartı (AES) genel ismi altındaki Rijndael algoritması bunun güncel bir örneğidir. Bundan önce Veri Şifreleme Standartı (Data Encryption Standart - DES) algoritması NIST tarafından 1977 yılında standart olarak tanınmıştı. DES algoritmasının kırılmasının ardından yerini bu algoritmanın iteratif çalıştırılmasını içeren Üçlü Veri İşleme Standartı (Triple Data Encryption Standart - TDES) almıştır. Zaman içinde TDES'in de güvenilirliğini yitirmesiyle NIST yeni bir şifreleme standardı arayacağını duyurmuştur. Bu sebeple 1997 yılında AES ismi altında kullanılmak üzere seçilmek için algoritma arandığı çağrısında bulunmuştur. Bunun sonucunda kriptoloji ile uğraşan kişilerce toplamda 15 algoritma gönderildi ve bunlardan beş tanesi 1999 yılında finalist olarak belirlendi. Bu algoritmaların performans, hız, uygulama kolaylığı, güvenlik, yazılım ve donanıma uyarlanabilme kolaylığı gibi etkenler açısından incelenmesi sonucunda Rijndael algoritması standart olarak seçilmiştir. Bu algoritma yukarıda belirtilen parametreler açısından da TDES'i geride bırakmaktaydı. Böylece AES, kendisinden önce kullanılmakta olan TDES standartının yerine geçmiştir ve halen kullanılmaktadır.

Son yıllarda gömülü sistem gerçeklemlerinde popülerliği oldukça artan bir diğer kavram da gerçekleştirilebilir işlemcilerdir. Bu işlemciler yine açık olarak sunulan daraltılmış komut kümelerinin gerçeklemlerinden ortaya çıkmaktadır. Öne çıkan en büyük özellikleri, açık kaynak kodlu ve değiştirilebilir olmalarından kaynaklanan esneklikleridir. Mikroişlemci ve mikrodenetleyici gibi bileşenler çok amaçlı kullanıma göre tasarlanmaktadır, fakat bu tasarımlar her uygulama alanı için alan veya performans bakımından en ideal sonucu vermeyebilir. Üretici firmalara özel kullanım için tasarımlar üretirmek de maliyet açısından verimsiz olduğundan uygun bir yaklaşım olmamaktadır. Bu tür sorunları aşmak adına öncelikle yazılım dünyasında popülerleşmiş bir kavram olan açık kaynak kodlu tasarım düşüncesi de donanım

dünyasına uyarlanmaya başlanmıştır. Bununla birlikte, içeriği üretici firmalar tarafından gizli tutulan kapalı kutu tasarımlar yerine kaynak kodları açık olan gerçekleştirilebilir işlemciler kavramı öne çıkmıştır. Bu durum kullanıcılara kendi uygulama alanlarına göre tasarımın gereken bölümlerini kullanma, değiştirme, hatta yapacakları hata tespitleri veya eklentiler ile de geliştirme sürecine katkı sağlayabilme imkanı vermektedir.

Bu tez kapsamında, yukarıda anlatılan bu iki güncel kavramı birden içeren çalışmalar yapılması hedeflenmiştir. Seçilen bir gerçekleştirilebilir işlemcili sistem üzerinde AES algoritmasının gerçekleştirilip farksal güç analizi kullanılarak yan kanal saldırılarına karşı direncinin sınanması amaçlanmıştır. Farksal güç analizi için gerekenler arasında bir sistemin gerçekleştirilmiş hali üzerinde çok sayıda gerçek zamanlı güç ölçümlerinin alınması vardır. Bu süreç ise oldukça vakit almakta olup ayrıca da tasarımın bir gerçekleştirilmesinin yapılmış olmasını gerektirmektedir. Bu çalışmada bu işlemin henüz tasarım benzetim aşamasındayken dahi fonksiyonel benzetim kullanılarak ve anlık işaret değişimi çizgesi çıkarılarak yapılabileceği gösterilmiştir. Bu çalışma hem gerçek güç ölçümünde harcanan zaman, hem de tasarımın gerçekleştirilmesinden gelecek olan maliyet kaybının da önüne geçerek daha verimli bir biçimde farksal güç analizi girdileri üretilerek bir şifreleme algoritmasının kırılabilme durumunu gösterir niteliktedir. Bununla birlikte gerçekleştirilebilir işlemcili sistem test birimi olarak kullanılmış ve bu alanda bir çalışma tecrübesi elde edilmiş; ayrıca ileride bu konu üzerine yapılacak çalışmalar için bir yol açılmıştır.

Çalışmalar sürecinde ilk olarak bu gerçekleştirilebilir işlemci gerçekleştirmelerinden birini içeren açık kaynak kodlu bir mikrodenetleyici sistemi olan PULPino incelenmiştir. Bu sistem, ETH Zürich gömülü sistem ekibi tarafından geliştirilmiş paralel düşük güç tüketen platformlar projesi olan PULP girişiminin bir üyesidir. Sistemin diğer aile üyelerinden ayrılan özelliği küçük ve kompakt olmasıdır, bu sebeple de eğitsel çalışmalar için çok uygundur. Çalışmanın ilk bölümünde bu tasarım detaylıca tanıtılmıştır. Ayrıca tasarım ile birlikte sunulan otomatikleştirilmiş çalışma ortamının kullanımı da anlatılmıştır. Bu ortamın içinde tasarımın simülasyonunun yapılması, doğrudan FPGA içine yüklenerek çalıştırılması gibi süreçler bulunmaktadır. Çalışma ortamının bazı belirli araç sürümleri ve işletim sistemini gerektirmesi nedeniyle, gerçekleştirilebilir işlemcilerle çalışmaya yeni başlayacak, özellikle yeterli donanım ve yazılım alt yapısı olmayan kişilerce kullanımı zor ve alışması zaman alan bir süreç olabilmektedir. Bu nedenle bu çalışma ortamının kurulması, çalıştırılması, yeni otomatikleştirilmiş özellikler eklenmesi gibi konular da ilk bölümde detaylıca ele alınmıştır. Bununla birlikte, çalışma ortamının kurulumunda ve kullanımında karşılaşılabilecek sorunlar ve gerekebilecek araçlar da belirtilmiş ve bunlarla karşılaşılması durumunda nasıl bir çözüm uygulanabileceği de anlatılmıştır. Böylece bu bölümün çalışmalarını PULP ailesi ile yapacak kişiler için bir kılavuz görevi görmesi hedeflenmiştir.

Farksal güç analizi için girdi üretmek adına AES'nin PULPino'da çalıştırılması için öncelikle gerçek dünyada güç ölçümü yapmaya yakınlık olması açısından PULPino tasarımının bir FPGA kartında gerçekleştirilmiş simülasyon modelinin kullanılması düşünülmüştür. Bunun için PULPino çalışma ortamındaki RTL kodları ile Xilinx Vivado aracında Artix-7 kartı için bu model üretilmiştir. Bu model için, PULPino çekirdeğini tek başına kullanarak, bazı test programlarının önyüklemeye hafızası içine atılan uygulama makine kodu ile çalıştırılması denenmiştir. Bu çalışma şekli, PULPino

işlemcisine eklenen bazı komut uzatmalarındaki bir tasarım hatasından dolayı başarılı olamadığı görülmüştür. Bahsi geçen hatanın başka çalışmalarda da raporlandığı yapılan incelemelerde görülmüştür. Bunun üzerine bu model, çalışma ortamı içindeki benzetim test modelinde kullanılmış olan ModelSim aracına aktarılmıştır. Böylece bu test ortamı kullanılarak gerçekleştirme sonrası benzetimler birkaç farklı test kodu için yapılmıştır.

Sonraki adımda, gerçekleştirme sonrası benzetimlerden işaret değişim aktivitesi elde ederek güç tahmini yapma yöntemleri incelenmiştir. Bunun için kullanılacak SAIF ve VCD uzantılı iki döküm dosyası türü bulunmaktadır. SAIF dosyaları tüm benzetim için toplamsal işaret değişimi bilgisini vermektedir. Bu dosyaların boyutları küçük olmakta fakat anlık işaret değişimi hakkında bilgi sunmamaktadır. Ek olarak bu dosyalarla Vivado aracında nasıl güç tüketimi tahmini yapılacağı gösterilmiş ve farklı test kodları için ortalama güç tüketimi tahminleri alınmıştır. Ardından, işaret değişimlerini anlık olarak tutan VCD dosyaları benzetim sırasında üretilmiştir. Gerçekleştirme sonrası benzetim modeli, verilen tasarımın FPGA kartına ait temel öğeleri içerecek şekilde modellenmiş halini içermektedir. Genel durumda bu temel birimlerden yüzlercesi tasarımdaki bir bloğu ifade etmektedir. Bu sebeple, tasarımdaki içsel işaretlerin ve giriş çıkışların sayısı katlanmaktadır. Bu durumun VCD dosyalarında çok sayıda işaret tutulduğundan dosya boyutlarında aşırı büyümeye sebep olduğu görülmüştür. Bu sebeple son olarak, gerçekleştirme sonrası benzetim yerine işlevsel model simülasyonu üzerinden işaret değişimi alma yoluna gidilmiştir. Buradaki beklenti, güç tüketimi çizgesinin yine de sistem üzerinde yapılacak bir işlemin önemli noktalarını gösterecek şekilde oluşacağı olmuştur.

Denenmiş olan ücretsiz güç tüketimi tahmini araçlarından hiçbiri anlık güç tüketimi bilgisi sunmadığından, bu işi yapacak bir kod MATLAB ortamında yazılmıştır. Yazılan kod, kendisine girdi olarak verilen işaret dökümü dosyasını işlemekte, daha sonra toplam işaret değişimlerini zamana bağlı olarak çizmekte ve bunlara ait sayısal değerleri tutmaktadır. Bu güç tüketimi tahmini çizgesi ile anlık güç tüketimi kestirme yöntemi çeşitli test uygulamaları ve AES ile denenmiş ve gerçekten de, uygulamalardan görülmesi beklendiği şekilde bir çizge olduğu gözlemlenmiştir. Ardından AES için çok sayıda farklı giriş mesajı verilerek yapılan benzetimler sonucunda elde edilen çizgeler farksal güç analizinde kullanılmıştır. Çizge üretimi, çözümlenmesi ve farksal güç analizi için gerekli tüm kodlar yazılmıştır. Ardından, üretilen girdiler ve tahmin edilen model kullanılarak korelasyon analizi yardımıyla AES anahtarı elde edilmeye çalışılmıştır. Korelasyon analizi sonucunda elde edilen korelasyon değerlerine göre anahtar bölümünün direkt olarak elde edilmesi veya anahtar bölümü tahmini kümesinin daraltılması amaçlanmıştır. Bu iki durumun görülmesi algoritmanın kırıldığını da gösteriyor olacaktır. Bunun için yapılan ölçümler ve sonuçlar sunulurken, önerilen yöntemin geçerliliği gösterilmiştir.



1. INTRODUCTION

Cryptology is a mathematical science discipline which changed its form within years, from being creating cyphers to studying information integrity and became even more important in modern era, due to digitalization. History of encryption spans to the middle of 14th century, though digital cryptography concept first officialised with an article presented in 1945 [6]. Afterwards, with the exceptional increase of digitalization in daily life, data safety also became a major concern for both designers and consumers. In the hardware realisation level of encryption algorithms, this concern is portrayed as the resistance of a design to the potential encryption cracking attacks; which gives away the very key used in an encryption algorithm.

Another very popular aspect of the modern years is to compose an open-source assemblage for both software and hardware products. Softcore processors [7] employs a identical type of approach to processor designing mentality; from openly introducing the instruction set architectures to providing actual Hardware Description Language (HDL) [8] level of the processor to users. The importance of this non black-box type approach is that it allows the users to examine, modify and extend an architecture and the corresponding hardware for specific needs. Befittingly, these custom instruction sets are designed in a manner that it easily allows further addition of new custom instructions. Henceforth, it could be stated that this type of architectures use the Reduced Instruction Set Computer (RISC) [9] concept. The RISC-V [10] architecture stands out as one of the highly common modern reduced instruction sets. In time, many processors have been developed that implement RISC-V instruction set architecture [11, 12]. Moreover, an ample amount of studies can be found in literature, which demonstrates the performances of some applications for different RISC-V processors [13–15].

As per the remarks made above, the works in this thesis combines softcore processor employing system-on-chip and encryption safety concepts together. First aim of the thesis is to employ an in demand open source system-on-chip desing and perform Field

Programmable Gate Array (FPGA) realisation based post-implementation dynamic power consumption analysis and automating the process for further use. Since the power consumption estimation for FPGA designs are highly tool reliant and possibly requiring to adapt the official designing and testing flows offered by the developers of the system-on-chip in question or requires users to come up with their own work flow from scratch; a time consuming adaptation period appears in the studies. This thesis offers a guide for performing post-implementation for the used RISC-V architecture employing system, to be utilized for further studies.

The second aim of the thesis is to offer a methodology for resistance tests for various encryption algorithms. It is shown that real time power consumption of an hardware can give away the vital information regarding encryption [16], even with the noise and the inclusion of power consumption of irrelevant hardware sections are taken into account [17]. However, works are done using on-chip real time power consumption measurements, hence it requires an already fabricated hardware to test its side-channel attack [18] resistance. This thesis aims to offer a simulation stage level side-channel attack resistance test methodology, so that a designer will be able to test his/her hardware or algorithm in behavioral simulation stages without being have to implement it first.

Second chapter of the thesis introduces the softcore processor employing system used during the process of works in thesis and explains the usage and build flow of the system in-depth, to be a reference for further studies. It also states various errors and solutions that could be encountered by future students.

Third chapter denotes the first leg of the work, which is creating and automizing the flow for performing a post-implementation simulation on the given softcore processor system. This part also shows how to create a custom task flow that is out of boundaries of the developer environment rules.

Chapter four explains the presented method to portray a digital circuit's momentary power consumption profile and demonstrates a number of graphical results, showing if the produced profiles are applicable on giving a viable idea on power consumption.

Fifth chapter explains the method for undergoing correlation analysis and includes the experiment results for regular and masked encryption, also discusses the results of key prediction when the resistance tests are done in simulation stage.



2. PULPINO PLATFORM

During the course of this work, simulation based tests are performed using a softcore processor employing system-on-chip platform. In this chapter, the overview of the platform that is used will be given along with the detailed information on the usage of it's working environment.

2.1 General Information

PULP (Parallel Ultra-Low Power) Platform project is one of the many popular RISC-V based innovatives that diversifies itself according to the number of used cores, and implicitly the System-on-Chip size. The project offers a group of open source low-power solutions for working areas like Internet of Things (IoT) [19], neural networks and microcontroller applications. It's main aspect is possessing low power consumption, and it does so employing near threshold computing; which achieves this by avoiding to make the device entering the strong inversion, applying performance compensation using parallel computing and managing a device's static power consumption in an excessive manner [20].

PULP family (it's illustration is given in Figure 2.1) also has single-core devices, PULPino Platform [3] is one of the members in this family. Here, "-ino" was used as

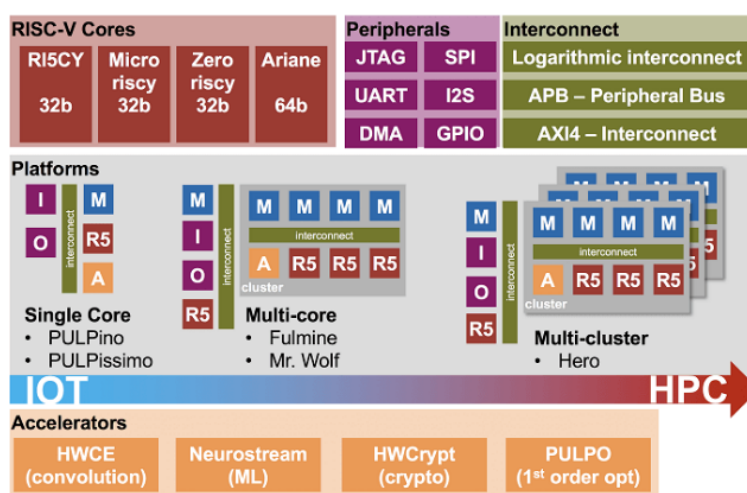


Figure 2.1 : The PULP Project family [1].

a diminutive suffix to imply smaller size of the device. The aim for this smaller scale design is to present a more compact and simple platform for some application areas, especially in IoT practices. This platform uses a modified version of PULP RISC-V instruction set architecture and is able to utilize various processor architectures that can support this instruction set. Currently, PULPino release offers readily available implementations for RI5CY [21] and zero-risc [22] cores. Top schematics of these cores are given in Figure 2.2 and Figure 2.3 respectively.

On the other hand, PULPino Platform offers a high number of peripherals that offers versatility during usage and testing stages. Below is a close-up list of the components included in the platform:

- A selectable single-core processor
- An optional Floating Point Unit
- Single-port data and instruction Random Access Memories (RAM)
- A boot Read-only Memory (ROM)
- Advanced Xtensible Interface interconnections for coordination between core and peripherals
- Timer and Event Unit, for power management

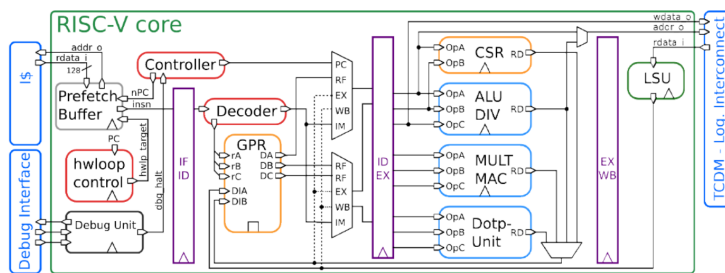


Figure 2.2 : The RI5CY core top level schematic [2].

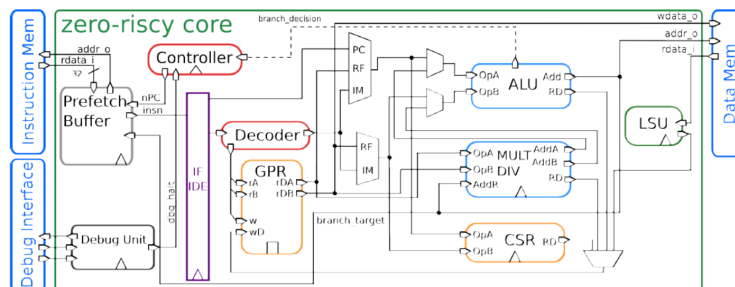


Figure 2.3 : The zero-risc core top level schematic [2].

- An Serial Peripheral Interface (SPI) [23] module
- Advanced debugger that allows access to memory map using JTAG

It is very important to state that all these components are open source, and allows others to change the hardware, add their own blocks and subtract defaultly available components. In fact, PULPino peripheral bus contains empty areas specifically ment for the users to add their custom peripherals [2]. However, it is very likely that some adjustments have to be made on the system behaviour providing the user doing radical changes on the architecture.

Figure 2.4 presents the top level schematic of PULPino Platform.

2.2 PULPino Working Environment

Being an open-source platform, related HDL codes of the PULPino Platform can be freely accessed by users. Along with this, all the first party content related to PULPino platform is also completely made available to users, in form of a buildable Linux-based [24] working environment [25]. The build flow belonging to this environment is given in 2.5 Using this working environment;

- All the HDL code related to PULPino system can be examined and modified.
- Using the simple command set offered by the environment, the design could be compiled, prepared and behaviorally simulated using ModelSim [26] and Verilator [27] habitats.

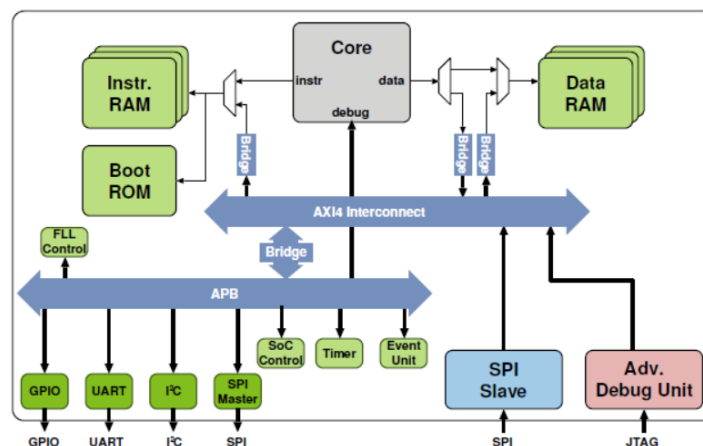


Figure 2.4 : The PULPino Platform top level schematic [2].

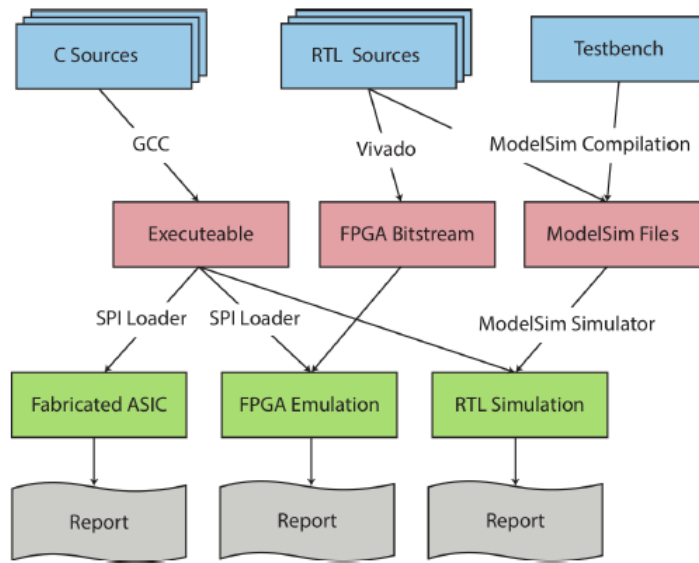


Figure 2.5 : The build flow of the PULPino environment [3].

- C/C++ programs could be prepared and compiled with the help of a custom instruction set toolchain. The environment also contains different C/C++ applications ready to be compiled and run on the platform.
- A Xilinx Vivado [28] project can be automatically generated to implement the platform on Field Programmable Gate Arrays (FPGA).
- PULPino system can be directly realized on a Xilinx ZedBoard [29] device, using automated working environment commands.

Apart from this, the working environment does not offer automatization for any other type of FPGA devices bar the aforementioned one. Moreover, simulation scripts are limited to perform behavioral simulation only, requiring the users to generate their own processes to perform post-implementation analysis and any sub-analyses related to implementation (e.g post-implementation power consumption estimations). Another important thing to note is that the PULPino environment’s building and scripting processes is only completely supported under Linux based operating systems.

The working environment is highly automatized and requires specific adjustments and tool versions to be fully utilized. Therefore, a general Linux and scripting knowledge is essential to fully benefit the environment. Works on this thesis are performed on a Linux Mint 17.3 [30] 64-bit operating system; using Xilinx Vivado 2015.1 and 2018.2 revisions and ModelSim SE 10.4 and PE 10.2 versions. It is also highly recommended

that to ensure the Linux based operating system that is planned to be used is up to date. It should be noted that all the codes, scripts and programs given in this thesis are written or executed within the specified operating system and tool versions, and are not guaranteed to work on different operating system distros and tool versions.

2.2.1 Prerequisites for the environment

PULPino working environment is strictly dependent on certain tools and software versions to properly executed. Most of these requirements are already presented in the depository readme files however, depending on the used operating system, users might still need to obtain various software that are not explicitly explained by the environment authors. Below, required tools and software for using the working environment are listed along with their reason for usage. Also, we included the missing software that was necessary to work with the environment within the operating system version used on this thesis, for further reference.

It is also strongly recommended to keep the operating system and its installer repositories up to date before starting to setup the PULPino working environment and its components. The terminal codes below performs these tasks. It should be noted that this process could take quite long time for newly-installed systems.

```
sudo apt update sudo apt upgrade
```

Requirements that are included in working environment documentation:

- CMake: The PULPino environment uses a CMake based file management system handle making and building processes. Source package of the program can be downloaded from its homepage [31]. Then, users should extract the archive wherever they want the installation should stay. Archive can be extracted using;

```
tar -xvzf <archivename>.tar.gz
```

Afterwards, users should go into the extracted directory and run the following commands;

```
./bootstrap make make install
```

Then the users should find the bin folder generated inside their installation directory (ex. <installationfolder>/cmake-3.13.0-rc2/bin). and add this bin directory to

the system's PATH variable. Manually adding to the PATH variable is not recommended since this requires the user to add the variable each time the user opens a new terminal. To make the location permanently stay at the PATH variable; .profile file or .bashrc file located on /home directory must be edited. Users could have need to enable show hidden files option in their system to see this file. If none of these files are readily available on the home directory, users could just create them on their own.

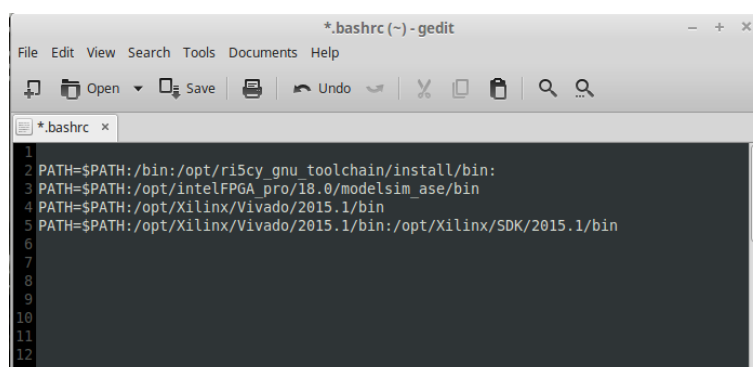
The general command for adding something to the PATH variable is given below, and this command should be copied into either ".bashrc" or ".profile" file.

```
$PATH = PATH:<pathtoadd>
```

It is advised to do this for each step requiring to add something to system's PATH variable. PATH variable contains the search locations of executable files that will be directly recognised by the system, without being need to specify the exact location of corresponding executable file. Figure 2.6 shows an example of an edited .bashrc file.

- **ModelSim:** All the scripts in the environment that handles compiling and simulation are written specifically for ModelSim tool. If a user aims to use readily available compile and simulation chain, using ModelSim is a necessity; otherwise, scripts must be written from scratch depending on the compiling and simulating habitat. ModelSim offers freely available versions for educational purposes.

The ModelSim folder that contains the executables should be added to the system's PATH variable, These files will be contained in "<path_to_modelsim>/modelsim_ase/linuxaloem" directory.



```
*.bashrc (-) - gedit
File Edit View Search Tools Documents Help
Open Save Undo Cut Copy Paste Find
*.bashrc x
1
2 PATH=$PATH:/bin:/opt/ri5cy_gnu_toolchain/install/bin:
3 PATH=$PATH:/opt/intelFPGA_pro/18.0/modelsim_ase/bin
4 PATH=$PATH:/opt/Xilinx/Vivado/2015.1/bin
5 PATH=$PATH:/opt/Xilinx/Vivado/2015.1/bin:/opt/Xilinx/SDK/2015.1/bin
6
7
8
9
10
11
12
13
```

Figure 2.6 : Example of a correctly edited .bashrc file.

- `ri5cy-toolchain` [32]: A RISC toolchain handles compiling, linking and building process of C/C++ codes for a RISC based processor architecture. With another way of saying, it carries over the entire process of converting a text based higher level code to RISC compatible machine code. Because of this, whenever a new instruction is customly added to the standart RISC-V instruction set, the corresponding toolchain must be edited to make it understand this new instruction as well. Since RI5CY core of PULPino uses some custom instructions, it uses a specially modified version of a standart RISC-V toolchain. This modified toolchain contains extensions that support modified RI5CY core instructions.

Firstly, the toolchain requires some packages to be present in an operating system.

These packages must be installed with the following command:

```
sudo apt-get install autoconf automake autotools-dev
curl libmpc-dev libmpfr-dev libgmp-dev gawk
build-essential bison flex texinfo gperf libtool
patchutils bc zlib1g-dev
```

Afterwards, toolchain repository must be downloaded with:

```
git clone https://github.com/pulp-platform/ri5cy_gnu_toolchain
```

Then, users should enter the `ri5cy-gnu-toolchain` folder, and while inside this folder, type “make“ in the terminal to start building process of the toolchain. It should be noted that this process may take a while. After the building is complete, a “`ri5cy_gnu_toolchain/install/bin`” must be present in the directory. This bin folder must be added to the `PATH` variable of the system.

Various tools that might be required and not stated in the working environment documentation are as follows:

- `tcsh`: This shell environment is used in some of the file generation processes. The terminal command below can be used to directly install this software.

```
sudo apt-get install tcsh
```

- `git`: This applicaton will be used to download various repositories within command lines and scripts. Following terminal command installs this application:

```
sudo apt-get git
```

- ia32-libs ve g++multilib libraries: If ModelSim would be run in 32-bit mode (the case in the working environment scripts) under a 64-bit operating system, these libraries must be installed. Otherwise, related working environment scripts have to be modified. For the aforementioned Linux version, the following commands could be used to install the libraries:

```
sudo dpkg add architecture i386      sudo apt-get update
sudo apt-get install ia32-libs      sudo apt-get install
g++multilib
```

- Python2.7 [33], pip and yaml packages: Pulpino main folder has some scripts depending Python2 and yaml packages.

Firstly, following commands installs python2.7, and pip for python2 and python3;

```
sudo apt install python2.7 python-pip
sudo apt install python3-pip
```

Afterwards, to install yaml packages system-wide;

```
sudo apt-get install python-yaml
```

- Vivado 2015.1: If the PULPino design is planning to be tested on an FPGA, or an automatically generated HDL project is desired to be used; precisely this version of Xilinx Vivado must be used, due to the scripts are only being compatible with this version. After generating all of the required outputs, a more recent version of Vivado could be used.

Vivado is freely available for users and can be downloaded on Xilinx servers (It should be searched inside archived downloads section, since this is an outdated version). After the downlad and installation is completed, commands below must be run on terminal before executing the Vivado user interface:

```
source <installation_path>/Xilinx/SDK/2015.1/settings64.sh
source <installation_path>/Xilinx/Vivado/2015.1/settings64.sh
```

Following these, typing the commands below in the terminal will execute Vivado in Graphical User Interface (GUI) mode:

```
cd /vivado
vivado &
```

- Vivado 2018: The project generated in 2015.1 version could be opened in the most recent Vivado versions. Since 2015.1 version being outdated, it does not support some SystemVerilog [34] structures used in the design, hence the official testbench for PULPino cannot be used in this habitat. Moreover, recent versions of Vivado, despite being able to support the aforementioned SystemVerilog structs, was still observed to be stuck during compilation process. To workaround this problem without doing in-depth search to entire HDL library of the design to pinpoint the problem, users can just generate the post-implementation model of design in Vivado and then import it to ModelSim for post-implementation simulation (PULPino environment only offers behavioral simulation automation on ModelSim or Verilator).

2.2.2 Setting up the environment

After all the required software installed, the system is ready to use the PULPino working environment. PULPino is presented in a GitHub repository [25] and can be downloaded using the terminal command;

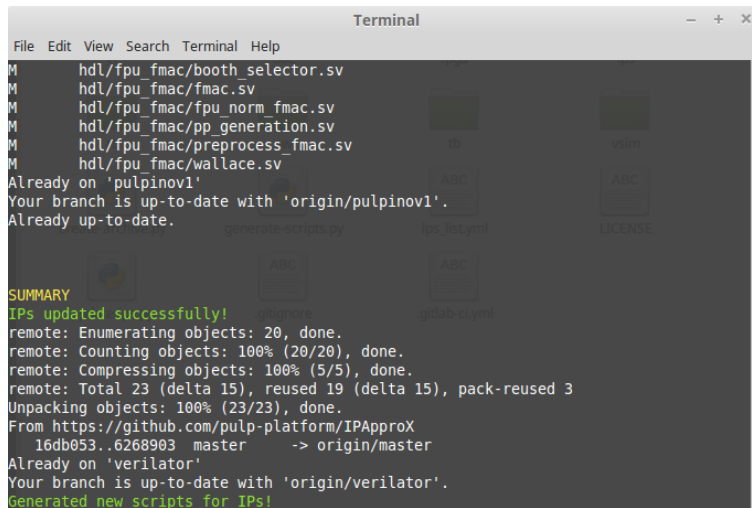
```
git clone https://github.com/pulp-platform/pulpino
```

This will download the “pulpino“ folder to the current directory. This folder is the main directory for any PULPino related work. However, this folder comes in an incomplete state. PULPino uses various different packages from other GitHub repositories. The files associated with this repositories will not come with PULPino installation itself, and must be downloaded using a Python script named “update-ips.py“ inside pulpino main folder. Following commands will execute this script to download extra files from the required repositories.

```
cd pulpino  
./update-ips.py
```

If the script finishes it’s run successfully, a success message similiar to the one in Figure 2.7 will be seen. Moreover, “pulpino/ips“ folder will now have the Register-Transfer Level (RTL) files of the Intellectual Properties (IP) used within PULPino.

In the next step, a build folder will be generated to hold a working session for users. This can be done inside the ”pulpino/sw” folder. This folder houses the C/C++



```
Terminal
File Edit View Search Terminal Help
M hdl/fpu_fmac/booth_selector.sv
M hdl/fpu_fmac/fmac.sv
M hdl/fpu_fmac/fpu_norm_fmac.sv
M hdl/fpu_fmac/pp_generation.sv
M hdl/fpu_fmac/preprocess_fmac.sv
M hdl/fpu_fmac/wallace.sv
Already on 'pulpinov1'
Your branch is up-to-date with 'origin/pulpinov1'.
Already up-to-date.

SUMMARY
IPs updated successfully!
remote: Enumerating objects: 20, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 23 (delta 15), reused 19 (delta 15), pack-reused 3
Unpacking objects: 100% (23/23), done.
From https://github.com/pulp-platform/IPApproX
 16db053..6268903 master -> origin/master
Already on 'verilator'
Your branch is up-to-date with 'origin/verilator'.
Generated new scripts for IPs!
```

Figure 2.7 : Terminal output for a succesful IP update.

applications and codes to be run on the PULPino, and contains building scripts for bringing together a working folder for simulations and application running. There are four readily available building scripts inside this folder:

- `cmake_configure.riscv gcc.sh`: Build a PULPino environment that uses RI5CY as it's core.
- `cmake_configure.riscvfloat gcc.sh`: Build a PULPino environment that uses RI5CY as it's core, which also contains an extra Floating Point Unit.
- `cmake_configure.zeroriscy gcc.sh`: Build a PULPino environment that uses zeroriscy as it's core.
- `cmake_configure.microriscy gcc.sh`: Build a PULPino environment that uses microrisc as it's core.

These scripts automatically assumes that the user will have a folder named “build“ inside the “sw“ folder (`pulpino/sw/build`) and it is being run within this folder. Therefore, in order to use the scripts without making any changes, one should create an empty folder named “build“ inside their “pulpino/sw“ folder and copy the building script that is desired to be used in this folder. During this thesis, PULPino with RI5CY core is used without a Floating Point Unit.

After copying into the empty build folder, the script should be run in terminal with;

```
./script_name.sh
```

If the build is successful, a terminal output similar to the one seen in Figure 2.8 will be printed. Now the PULPino is readily built to be used with ModelSim.

2.2.3 Behavioral simulation

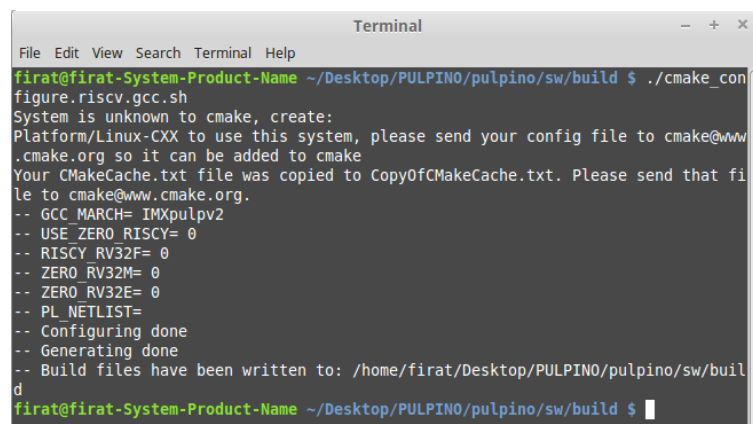
The official release of the PULPino offers behavioral simulation on ModelSim. Automated version of these simulations offers running various C/C++ programs directly on the system. The C/C++ program is first converted into machine code via the customized RISCY toolchain. Afterwards, various stimuli files are generated to be used inside the simulation testbench. There are two readily available methods for injecting the machine code inside the PULPino Platform;

- Preloading the instruction and data memories: Automatically generated machine code and program data are copied into PULPino instruction and data RAM's during the simulation runtime. The stimuli are read from text files.
- Loading with SPI: Instructions and program data are fetched in simulation runtime from the SPI module located within the PULPino. SPI stimuli is taken from a text file.

After the execution, all signal data can be observed by the users within ModelSim interface. This includes memories and peripherals inside PULPino.

Before beginning the program execution simulations, the PULPino design must be compiled for ModelSim. This process is automated and can be done by typing,

```
make vcompile
```



```
Terminal
File Edit View Search Terminal Help
firat@firat-System-Product-Name ~/Desktop/PULPINO/pulpino/sw/build $ ./cmake_con
figure.riscv.gcc.sh
System is unknown to cmake, create:
Platform/Linux-CXX to use this system, please send your config file to cmake@ww
.cmake.org so it can be added to cmake
Your CMakeCache.txt file was copied to CopyOfCMakeCache.txt. Please send that fi
le to cmake@www.cmake.org.
-- GCC_MARCH= IMXpulpv2
-- USE_ZERO_RISCY= 0
-- RISCY_RV32F= 0
-- ZERO_RV32M= 0
-- ZERO_RV32E= 0
-- PL_NETLIST=
-- Configuring done
-- Generating done
-- Build files have been written to: /home/firat/Desktop/PULPINO/pulpino/sw/buil
d
firat@firat-System-Product-Name ~/Desktop/PULPINO/pulpino/sw/build $
```

Figure 2.8 : Expected terminal output after a succesful environment build.

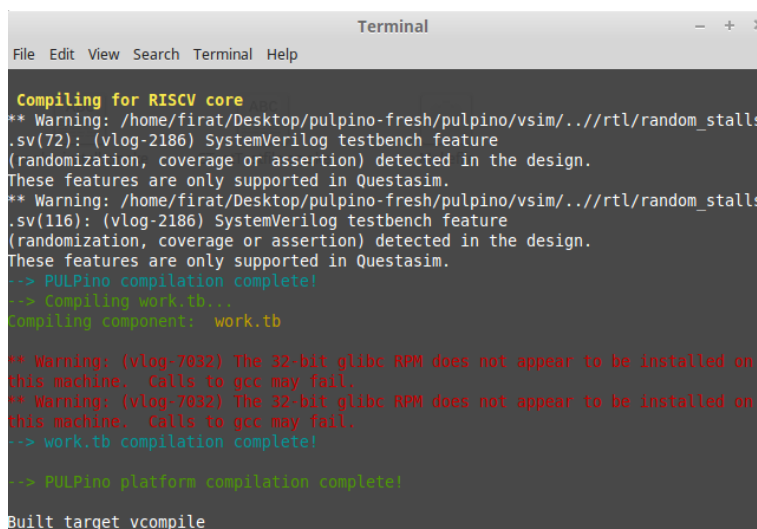
while inside the “pulpino/sw/build“ folder. This command compiles the RTL found in directories “pulpino/rtl“ , “pulpino/ips“ and “pulpino/tb“. These folders actually holds the source HDL code of the PULPino platform, it’s peripherals and the ModelSim testbench, and can be freely edited, examined or added into. It should be noted that the codes must be recompiled with the previously given terminal command, for the changes to be appear in ModelSim simulation.

When the compilation of the RTL codes are succesful, the terminal message given in Figure 2.9 must appear at the end. Warnings related to unsupported assertion statements and compiler calls can be ignored, as these will be simply not implemented during the compilation and won’t cause execution faults.

In order to execute the already available C programs that is contained in official PULPino working environment, the folder “pulpino/sw/apps“ must be observed firstly. Folders in this directory has related C/C++ codes to be compiled on custom RI5CY toolchain. To compile the codes and generate a ModelSim environment for the simulation, related make rules have to be invoked while inside “pulpino/sw/build“ folder. application names can be checked from CMake list files or double tapping tab button after writing “make“ to the terminal.

For example, a text output program is located at “pulpino/sw/apps/helloworld“ folder. To compile, generate stimuli files and create ModelSim simulation environment;

```
make helloworld
```



```
Terminal
File Edit View Search Terminal Help

Compiling for RISCV core
** Warning: /home/firat/Desktop/pulpino-fresh/pulpino/vsim/../rtl/random_stalls
.sv(72): (vlog-2186) SystemVerilog testbench feature
(randomization, coverage or assertion) detected in the design.
These features are only supported in Questasim.
** Warning: /home/firat/Desktop/pulpino-fresh/pulpino/vsim/../rtl/random_stalls
.sv(116): (vlog-2186) SystemVerilog testbench feature
(randomization, coverage or assertion) detected in the design.
These features are only supported in Questasim.
--> PULPino compilation complete!
--> Compiling work.tb...
Compiling component: work.tb

** Warning: (vlog-7032) The 32-bit glibc RPM does not appear to be installed on
this machine. Calls to gcc may fail.
** Warning: (vlog-7032) The 32-bit glibc RPM does not appear to be installed on
this machine. Calls to gcc may fail.
--> work.tb compilation complete!

--> PULPino platform compilation complete!

Built target vcompile
```

Figure 2.9 : Terminal output when all the PULPino RTL is succesfully compiled.

command must be executed in “pulpino/sw/build“ directory. When successful, this creates a special simulation folder inside “pulpino/sw/build/apps“ directory, named “helloworld“. This folder will have ModelSim configuration files to be used for simulation, stimuli files related to the compiled C code, tracer output and stdout folder for Universal Asynchronous Receiver-transmitter (UART) outputs. Afterwards, while still in the “pulpino/sw/build“ folder,

```
make helloworld.vsim
```

command can be used to automatically start the ModelSim behavioral simulation. This command automatically switches to the previously generated “pulpino/sw/build/apps/helloworld“ directory and invokes ModelSim simulation there, so that the stimuli files can be used which were generated from the C codes.

After the execution, this program will generate a “Hello World“ string and send it over UART module. Data sent from the UART will also appear in the command line of ModelSim. The executed behavioral simulation after calling helloworld.vsim command can be seen at Figure 2.10.

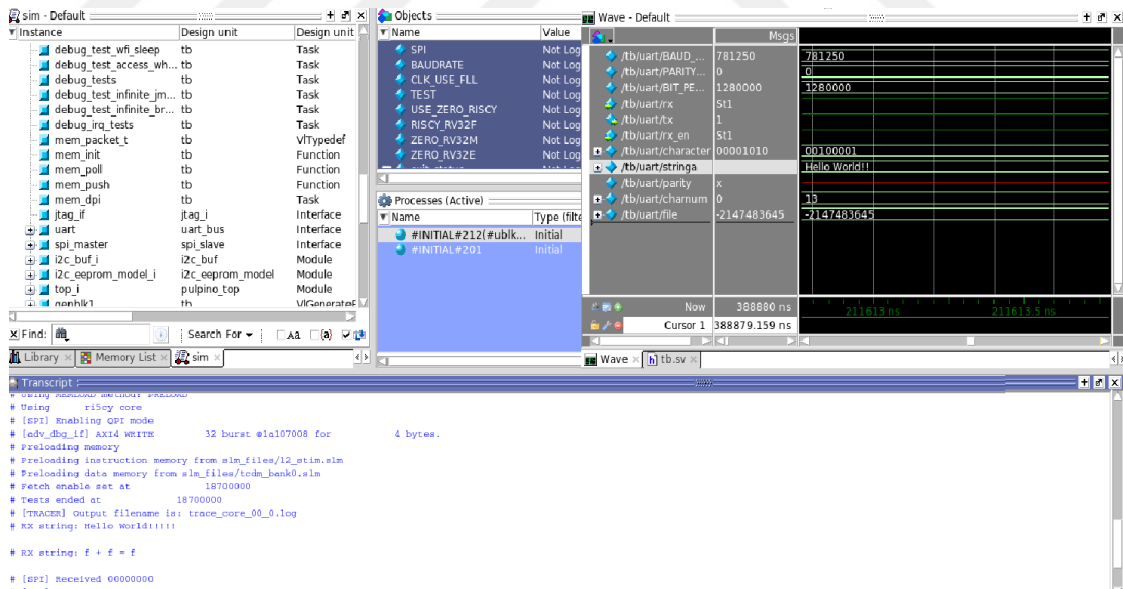


Figure 2.10 : Output of the helloworld program in ModelSim.

2.2.4 Adding custom applications

Inside the PULPino working environment, custom C programs can be included into the automation system to be ran on PULPino from ModelSim, similiarly to the helloworld example given at Section 2.2.3. Procedure for adding a custom application is given below, step by step:

1. Related codes must be copied into a user-created folder inside the directry “pulpino/sw/apps“. Name of the newly created folder is also user determined (ex. “pulpino/sw/apps/new_app“).
2. A file that is exactly named “CMakeLists.txt“ must be created in this newly created application folder. This file shall hold the names of C source files to be used and PULPino related header file includes, if any are used. This file lets the applciation to be recognized by automated CMake structures. One example of this file is presented at Figure 2.11. Here, CMAKE_SOURCE_DIR variable is automatically defined inside PULPino environment and it points to “pulpino/sw“ directory. Other directories inside the include definition are actually located within working environment and can be examined and edited by users. These files include PULPino platform related definitions and functions, like peripheral control.
3. Newly created application should be introduced to the build flow. This can be done by editing the file named “CMakeLists.txt“ inside “pulpino/sw/apps“ directory. At the end of this file, the following line should be added:

```
add_subdirectory(new_app)
```

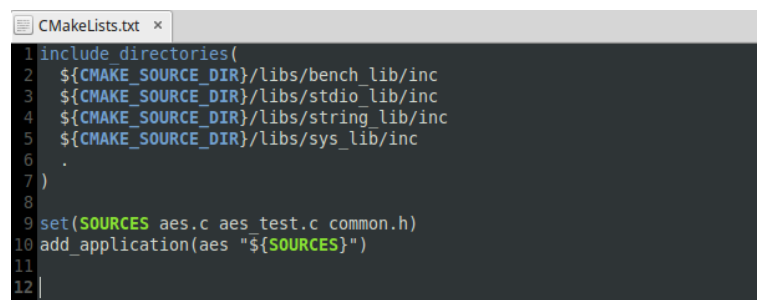
where “new_app“ denotes the name of the newly created application folder.
4. Finally, name of the newly created application folder can be used within build directory (ex. `make new_app`, `make new_app.vsim`). Compilation, simuation stimuli and ModelSim environment will then generated just like any other program within the app folder, for this newly created application; provided the correct terminal commands are typed.

2.2.5 Building FPGA related content

PULPino environment provides the process that uploads the design in ZedBoard platform. By using this process; users can generate a synthesisable HDL project of PULPino, generate a netlist based implementation project for ZedBoard, create stand alone Verilog HDL [35] netlists of the design and generate a customized embedded Linux environment to be loaded on a memory card, to be inserted on ZedBoard to run translated C applications.

Currently, the only two ways to run aforementioned C applications are either pre-loading their compiled versions on the memory card or sending them over to ZedBoard via Secure Shell (SSH) [36] protocol. The working environment provides automation to send executable code over SSH.

PULPino working environment houses a folder named `fpga`; and this folder has all FPGA related building rules for PULPino, from generating Vivado projects to automatically loading the design on ZedBoard. In order to build all the contents of this folder, the terminal command “make all” must be run while in the folder “pulpino/fpga“. After the execution, all the content will be available within subfolders inside this directory. Namely, “fpga/sw“ will have FPGA realisation make rules and related programs, like the boot loader and custom embedded Linux operating system image to be run on a ZedBoard SD card. Folders “fpga/rtl“ and “fpga/ips“ contains some extra HDL blocks used specifically for FPGA implementation. The folder “fpga/pulpino“ has the pre-synthesised Vivado project for the PULPino, and “fpga/pulpem“ has the ready to be emulated project and it is directly used for generating bitstreams for ZedBoard emulation. Figure 2.12 shows the top level



```
CMakeLists.txt x
1 include_directories(
2   ${CMAKE_SOURCE_DIR}/libs/bench_lib/inc
3   ${CMAKE_SOURCE_DIR}/libs/stdio_lib/inc
4   ${CMAKE_SOURCE_DIR}/libs/string_lib/inc
5   ${CMAKE_SOURCE_DIR}/libs/sys_lib/inc
6
7 )
8
9 set(SOURCES aes.c aes_test.c common.h)
10 add_application(aes "${SOURCES}")
11
12
```

Figure 2.11 : Example definitions in a CMake recognition file used in an encryption application.

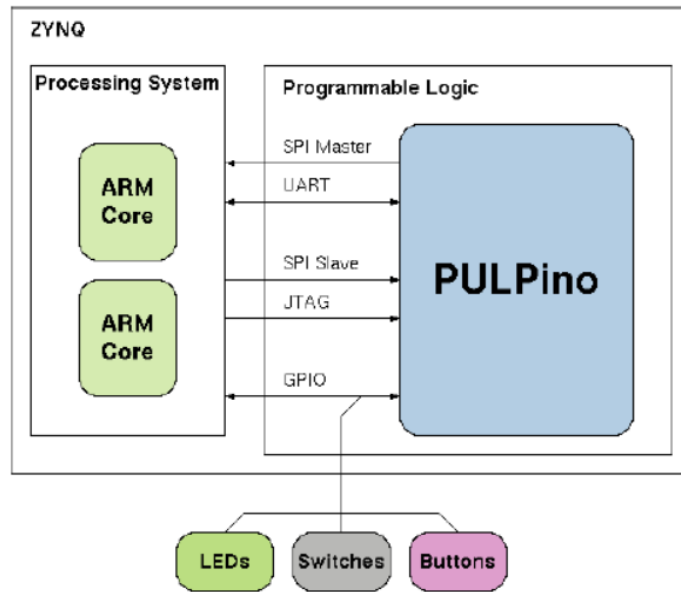


Figure 2.12 : Top module of PULPino ZedBoard Implementation [3].

schematic of this emulation project. The users can export and use the pre-synthesised Vivado project for their own purposes.

2.2.6 Potential problems and solutions

During the operations within the working environment it is quite possible to encounter various errors, since the environment is highly automated and depending on strict software versions. Below, possible errors that could be encountered are listed along with their solutions. Incompatibility errors encountered in the process of the thesis are also included in this list.

1. When trying to compile a C-Code within the environment, if users receive error messages regarding unrecognised objects and failed compiling, they must be sure to have `ia32libs` and `g++multilib` libraries in the system (Refer to Section 2.2.1).
2. Errors stating that some commands are not found indicates that either the program that contains the command is not installed, or executable files of this program is not in the `PATH` variable of the system (Refer to CMake requirement in Section 2.2.1).
3. During compilation or execution with ModelSim steps, if user receives an error regarding using a 64-bit system, all the “-64” flags inside the ModelSim execution and compilation scripts must be deleted. These are;
 - All the files under “/pulpino/vsim/tcl_files” directory

```

File Edit View Search Tools Documents Help
*vsim.tcl x
1 source tcl_files/config/vsim_ips.tcl
2
3 set cmd "vsim -quiet $TB \
4 -L pulpino_lib \
5 $VSIM_IP_LIBS \
6 +nowarnTRAN \
7 +nowarnTSCALE \
8 +nowarnTFMPC \
9 +MEMLOAD=$MEMLOAD \
10 -gUSE_ZERO_RISCY=$env(USE_ZERO_RISCY) \
11 -gRISCY_RV32F=$env(RISCY_RV32F) \
12 -gZERO_RV32M=$env(ZERO_RV32M) \
13 -gZERO_RV32E=$env(ZERO_RV32E) \
14 -t ps \
15 -voptargs="+acc -suppress 2103" \
16 $VSIM_FLAGS -dpicpppath /usr/bin/gcc"
17
18 # set cmd "scmd -sv_lib ./work/libri5cyv2sim"
19 eval $cmd
20
21 # check exit status in tb and quit the simulation accordingly
22 proc run_and_exit {} {
23 run -all
24 quit -code [examine -radix decimal sim:/tb/exit_status]
25 }

```

Figure 2.13 : Correctly edited “vsim.tcl“ to solve C compiler issue.

- “/pulpino/vsim/vcompile/rtl/vcompile_tb.sh”
 - “/pulpino/sw/apps/CMakeSim.txt”
4. During the execution of ModelSim, if an error regarding “/vsim_autocompile.so” object not being found, it means ModelSim cannot find the right C compiler. Inside the “pulpino/vsim/tcl_files/config/vsim.tcl” script, the “-dpicpppath” option must be added at the end of the line starting with “\$VSIM_FLAGS”, to point ModelSim to the correct compiler path. Generally in Linux based systems, the C compiler is located at “/usr/bin” folder. Correctly edited example of this script during the thesis can be seen at Figure 2.13.
 5. When building the FPGA related files in the environment, a build failure may be encountered during the processes that builds u-boot program; unless the Vivado SDK setting scripts are sourced before starting the build process.


```
source /opt/Xilinx/SDK/2015.1/settings64.sh && source /opt/Xilinx/Vivado/2015.1/settings64.sh
```
 6. During the build process of embedded Linux environment for the Zedboard implementation, an error related to gmake might be encountered. This problem can be bypassed by making a symbolic link from the make command executable:


```
sudo ln -s /usr/bin/make /usr/bin/gmake
```



3. POST-IMPLEMENTATION SIMULATION AND DYNAMIC POWER ESTIMATION ON PULPINO

This chapter examines the post-implementation simulation and dynamic power consumption estimation concepts done with PULPino platform. The post-implementation simulation was chosen at the first steps of the work, due to being the closest simulation to the actual FPGA implementation.

3.1 Preparing for Post-Implementation

The goal belonging to this part of the work, is to simulate PULPino platform on other FPGA boards that does not utilize build-in processor support (the official working environment only offers a realization on ZedBoard, which contains a built-in processor), then getting the post-implementation power consumption estimation results for various applications. After undergoing the working environment setting and HDL project generation steps stated in , synthesis and implementation processes are done for a different device. For the implementation process, Artix-7 Development Board [37] (xc7a200tfbg676-2 device) and Vivado 2018.2 was used. The resource usage of the implementation is given in Table 3.1.

3.1.1 First testbench

First attempt to perform a post-implementation simulation on PULPino was to try and import the official testbench came with PULPino working environment to Vivado

Table 3.1 : Post-implementation resource usage of PULPino on Artix-7 Development Board.

Resource Name	Used Amount	Total Amount	Percentage Used
FF	9883	267600	3.69
LUT	15647	133800	11.69
I/O	143	400	35.75
BRAM	16	365	4.38
DSP48	6	740	0.81
BUFG	3	32	9.38

2015 and execute the process there. It is done so by adding the testbench related RTL files into the project as simulation sources. However, during the compilation steps of the testbench, it was seen that Vivado 2015, being an outdated release of the software, did not support most of the vital SystemVerilog structures coded inside the testbench RTL. For this reason; as the second attempt, the project was imported to Vivado version 2018.2, which was the latest version during that leg of the study. This version recognized the required SystemVerilog structures, yet the simulation still could not be started due to the simulator getting stuck in the elaboration step, without any log report to contract the scope of the problem.

Following this problem, it was decided to just extract the RISC core from the system and simulate it as standalone. PULPino core region has a boot ROM to be firstly executed when the fetching signal is received, and there exist a register bank inside the RISCY core that can be examined during simulation. Herewith, the compiled application code was planned to be inserted in this boot ROM. Simulation was planned to be ended when the "isboot" signal inside the desing becomes low, which indicates the execution of the boot ROM code has been finished.

A simple testbench is written in the light of the given information above. The only parts used regarding the system in this testbench are the core region and the fundamental clock, reset and fetch signals. Signals related to peripherals that does not affect the running of the core region are not connected for simplicity. A visualization of this testbench can be seen at Figure 3.1.

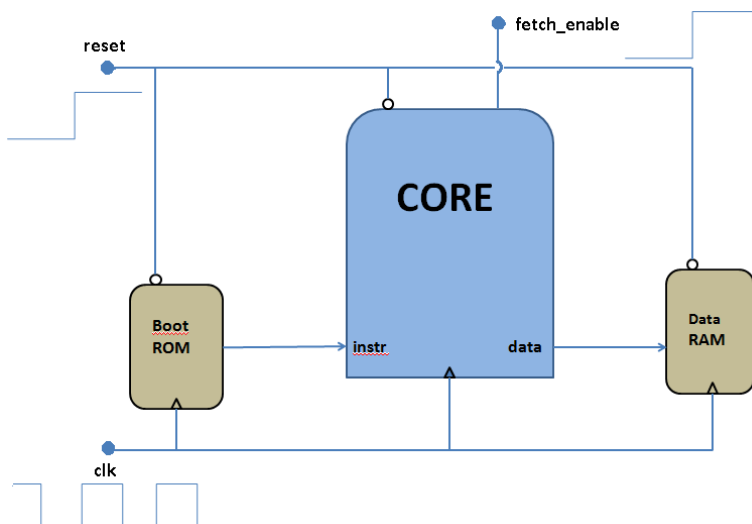


Figure 3.1 : Structure of first testbench.

3.1.2 Editing the boot ROM

Changing the contents of the boot ROM can be done with the commands proffered within PULPino build environment. This process uses the application folder, with the exact destination “pulpino/sw/apps/boot_code“. The C file “boot_code.c“ in this folder contains the code to be written in boot ROM. Users need to copy their own code inside this file and then compile it with the command:

```
make boot_code.install
```

from the terminal when accessed within “pulpino/sw/build“ folder. This turns the related C code to machine code, prepares a new boot ROM RTL file containing this machine code and switches it with the boot code RTL inside “pulpino/rtl“. It is advised to backup the original boot code, since this boot code is used to run the standard processes within PULPino environment. The new boot ROM RTL will also be within the folder “pulpino/sw/build/apps/boot_code/boot“. If necessary, more address space can be allocated for boot ROM from PULPino memory map, which can be changed from the top level PULPino SystemVerilog file [2]. Figure 3.2 shows the aforementioned memory map.

As benchmark applications for the system, some applications from The Worst-Case Execution Time Project [38] group was chosen to be executed on the processor, to perform runtime test and generate switching activity outputs. These benchmarks are discussed in Section 3.3.2.

This testbench was also failed to work properly. It is observed that during some conditions, newly added hardware loop extensions added in the RI5CY custom extensions [21] and causes the hardware to enter infinite loops and stalling the processor. It is later found that this problem was also detected by another study [15],

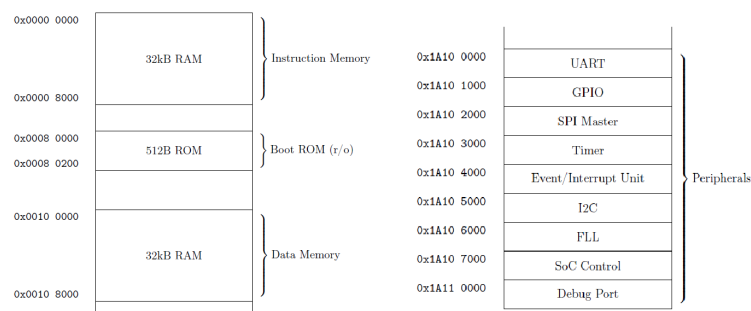


Figure 3.2 : PULPino memory map [2].

which aimed to verify PULPino. Since this was the case with benchmarks prepared for the boot ROM compiling options, it was finally decided that cancelling to work in Vivado environment and instead, exporting the post-implementation model to in order to be able to work with the official ModelSim PULPino testbench, this time compiling the benchmarks for the instruction RAM.

3.2 Post-Implementation Simulation with Official PULPino Testbench

Performing a post-implementation simulation in ModelSim requires a number of extra preparations. Being only a simulation tool, ModelSim does not offer FPGA device primitive based implementation models for written HDL codes. It depends on other tools to create the commercial FPGA device equivalents of a complete design, which being HDL netlists of circuits, in terms of FPGA primitives. These primitives are vendor dependant and are available from the corresponding design software to be compiled as exportable simulation libraries.

Aforementioned primitive simulation libraries of Xilinx for ModelSim can be generated from Vivado GUI, from Tools->Compile Simulation Libraries tab in toolbar menu. Libraries will be created in the user specified location and then need to be defined to ModelSim before using in a simulation. This is done by editing the "modelsim.ini" file inside it's installation directory. Xilinx library names and their specific paths must be added in this file accordingly. The edited .ini file that used during the studies in the thesis is given as an example in Figure 3.3, with the enframed entries being the Xilinx library definitions.

```

modelsim.ini
69 cyclone10gx_hssi = $MODEL_TECH/./altera/vhdl/cyclone10gx_hssi
70 cyclone10gx_hip = $MODEL_TECH/./altera/vhdl/cyclone10gx_hip
71
72 ; Verilog Section
73
74 altera_mf_ver = $MODEL_TECH/./altera/verilog/altera_mf
75 altera_ver = $MODEL_TECH/./altera/verilog/altera
76 altera_lnsim_ver = $MODEL_TECH/./altera/verilog/altera_lnsim
77 lpm_ver = $MODEL_TECH/./altera/verilog/220model
78 220model_ver = $MODEL_TECH/./altera/verilog/220model
79 sgate_ver = $MODEL_TECH/./altera/verilog/sgate
80 twentynm_ver = $MODEL_TECH/./altera/verilog/twentynm
81 twentynm_hssi_ver = $MODEL_TECH/./altera/verilog/twentynm_hssi
82 twentynm_hip_ver = $MODEL_TECH/./altera/verilog/twentynm_hip
83 fourteennm_ver = $MODEL_TECH/./altera/verilog/fourteennm
84 fourteennm_ctl_ver = $MODEL_TECH/./altera/verilog/fourteennm_ctl
85 cyclone10gx_ver = $MODEL_TECH/./altera/verilog/cyclone10gx
86 cyclone10gx_hssi_ver = $MODEL_TECH/./altera/verilog/cyclone10gx_hssi
87 cyclone10gx_hip_ver = $MODEL_TECH/./altera/verilog/cyclone10gx_hip
88
89 secureip = /opt/intelFPGA_pro/18.0/modelsim_ase/xlibs/secureip
90 unisim = /opt/intelFPGA_pro/18.0/modelsim_ase/xlibs/unisim
91 unimacro = /opt/intelFPGA_pro/18.0/modelsim_ase/xlibs/unimacro
92 unifast = /opt/intelFPGA_pro/18.0/modelsim_ase/xlibs/unifast
93 unisims_ver = /opt/intelFPGA_pro/18.0/modelsim_ase/xlibs/unisims_ver
94 unimacro_ver = /opt/intelFPGA_pro/18.0/modelsim_ase/xlibs/unimacro_ver
95 unifast_ver = /opt/intelFPGA_pro/18.0/modelsim_ase/xlibs/unifast_ver
96 simprims_ver = /opt/intelFPGA_pro/18.0/modelsim_ase/xlibs/simprims_ver
97
[vc]

```

Figure 3.3 : The modelsim.ini file with added Xilinx libraries.

For the next step, the post-implementation netlist model of the circuit is created within a single file in Verilog HDL format, using Vivado command line. For this work, the command below is used;

```
write_verilog -force -mode timesim pulpino_impl.v
```

where `pulpino_impl` denotes the user specified file name, option “-mode timesim” tells the program to generate post implementation timing simulation model, and -force option rewrites the file if it exists in the directory. A destination can also be specified for the generated file (It will be generated in the project root directory by default). Then, it is exported out of the project to be compiled in ModelSim for post-implementation simulation.

As the testbench of the simulation, the standart built-in PULPino testbench can be used. In the testbench file, memory loading technique is chosen to be SPI loading. Preloading method needs to access the individual memory blocks in the design. However, in order to increase performance, FPGA specific RTL of PULPino uses Xilinx pre-built RAM IPs, instead of the RAM RTL regularly used in PULPino working environment; which is not suitable for FPGA implementations. Because of the black-box design of Xilinx IPs, memory blocks of a Xilinx RAM cannot be referred directly in testbench code. While preload method tries to directly refer the RAM cells, SPI load refers them via addressing so it is suitable to be used with Xilinx RAMs. Downside of this method is that it undergoes within the simulation run, hence slows down the overall simulation runtime.

Since the post-implementation simulation flow does not exactly follow the build rules automatically implemented in PULPino working environment, meaning it is a custom flow, there exists some significant differences in the procedure. To initiate and automatize the post-implementation simulation, To begin with, folder is created that will hold post-implementation simulation related material. Contents of an example custom simulation folder can be seen at Figure 3.4. The simulation folder should essentially include:

- `slm_files`: This folder should hold stimulus files to be used in simulation. Since stimuli will be loaded with SPI, folder must include “`tcdm_bank0.slm`” and

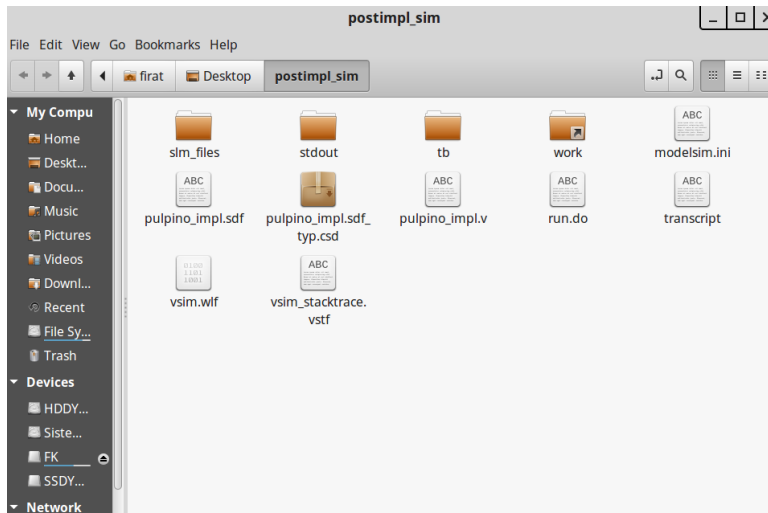


Figure 3.4 : Contents of an example simulation folder.

“spi_stim.txt“ files. These files will be automatically generated in related build/apps folder of the related application.

- Symbolic link to “work“ folder: A symbolic link of the folder “pulpino/vsim/work“ must be included in this file. This folder holds the compiled version of the PULPino testbench and it is automatically generated in the aforementioned directory.
- modelsim.ini: This file will specify which simulation libraries will potentially be included in the work flow. This folder can just be a copy of the modelsim.ini file located at the ModelSim installation directory. Otherwise, it has to include the Xilinx libraries to be used.
- Post-implementation netlist model: This file will be compiled before simulation starts, and will be the unit under test of the simulation testbench. In this example, it is a Verilog file named “pulpino_impl.v“.
- Automation script: This file contains the console commands that needed to be typed for start and complete the simulation flow. In ModelSim, these script can be written into text files with .do extensions, to be used with ModelSim’s “do“ command.

To start the simulation proces; firstly, ModelSim tool must be invoked within this custom folder. Secondly, the netlist model must be compiled using the command:

```
vlog name_of_file.v
```

This will compile the model into the working library folder. Then the simulation screen will be invoked using the command below:

```
vsim tb -L simprims_ver work.glbl -dpicpppath  
/usr/bin/gcc
```

Here, “tb“ is the name of the testbench, “-L simprims_ver“ indicates that simprims_ver Xilinx library will be used during simulation (contains simulation primitives for design written in Verilog), “work.glbl“ is a global definition file contained within netlist file and should be added into work library after netlist compilation. Lastly, -dpicpppath option specifies the correct location of system’s C compiler. Finally, the simulation is executed using:

```
run -all
```

This command runs the simulation until a finishing statement is encountered in the testbench.

An automation script for the process is also written for automatization. The script in question can be observed in Appendix A.1 and includes the explained commands above collectively, with some optional commands.

3.3 Post-Implementation Average Dynamic Power Estimation

Following the successful execution of post-implementation simulation for given benchmarks, the dynamic power consumption estimation phase of the work was initiated. The post implementation model here will provide internal signal level change information that produced from a close approximate model of on-FPGA implementation execution of the device. On a digital circuit, one of the determining factors of dynamic power consumption is the load capacitance charges and discharges. Assuming that C_i , f_i , V_i respectively representing load capacitance value for a line i , switching frequency of the line i and the value of the voltage swing in the design; a dynamic power consumption relation is given as;

$$P = \sum_i C_i f_i V_i^2 \quad (3.1)$$

As stated above, a post-implementation simulation will provide a close to real signal level change activity, and this shall further provide the switching frequency information for equation 3.1. V_i is a device dependant parameter and it is constant when the same

FPGA device is used. The line capacitance values will be provided by the libraries used within implementation tool that produced the post-implementation model.

3.3.1 SAIF files and VCD files

Accurate dynamic power consumption estimation of a design essentially needs circuit switching activity information as its input, and there are two general file formats for containing these activity profiles. One of them being Switching Activity Interchange Files (SAIF) [39]; which holds the information like total switching amounts of design signals, total time a signal spent at high/low value and total glitches. Second technique is generating Value Change Dump (VCD) files [35]. These files hold values of each specified circuit signal for each clock cycle of simulation; but does not outright provide information about total switching values. These activity files can act as inputs to readily available power consumption estimation tools of current digital design softwares.

If a comparison between these two activity file types should be made, some vital differences could be seen. SAIF files, being containing switching information for the total simulation time scale; can only be used in average power consumption estimation. On the other hand, their file size is considerably small (around couple hundred megabytes for all signals was observed during this work). VCD files present time by time signal values; so it requires more computing time to calculate average power consumption, but it is also possible to make some observations on momentary power consumption. In terms of file size however, since its size expected to be linearly increased with time, they might become extremely large for long simulations (files as large as tens of gigabytes were produced during this work), making processing these files unproductive and problematic. Because of this, most of the modern tools discarded VCD for using in average power consumption estimations. For example, Vivado power estimator no longer supports VCD files to be used as inputs, only accepting SAIF files. Albeit, it is still necessary to work with dump files if momentary power consumption behaviour should desired to be examined.

During this work, both file types were used for different power analyses. SAIF files were used to get average power consumption estimations for different benchmarks,

VCD files were used for drawing a momentary power consumption profile for processor during the execution of machine codes (Refer to Chapter 4).

3.3.2 Average dynamic power consumption estimation with SAIF

As mentioned in Section 3.1.2, due to being unable to successfully compile the testing environment on Vivado, design files were exported to the ModelSim to be used together with official PULPino testbench. In ModelSim SE versions, there exists a command group named “power” for generating SAIF files of a given design. To create a SAIF output from a simulation, firstly the command below must be added before the simulation run statement:

```
power add -in -inout -internal -out -r /testbench/uut/*
```

This command starts the switching recording input signals (-in), output signals (-out), inout signals (-inout) and internal signals (-internal) for all submodules of the Unit Under Test (UUT) of the testbench. After the -r option the hierarchical name of the design should be entered, with “*” meaning to include all submodules below UUT’s hierarchy. Then the run statement should be followed by:

```
power report -all -bsaif test.saif
```

This will result in printing out the specified switching activity in a SAIF file named “test.saif”. The file then can be used as input for a desired power estimation tool that support it.

Below, the summary of steps for preparing and running a program in PULPino for a SAIF-using power consumption estimation within simulation environment are given as follows:

1. Firstly, three benchmarks that are written in C language were compiled for RI5CY processor via RI5CY toolchain; namely compress (data compression) [40], crc (cyclic redundancy check) [41] and edn (finite impulse response filter)

Table 3.2 : Structural properties of used benchmarks.

Benchmark	Size (Bytes)	Nested Loops	Arrays	Bit Operations
compress	13411	Yes	Yes	No
edn	10563	Yes	Yes	Yes
crc	5168	No	Yes	Yes

Table 3.3 : Simulation times and run times for different benchmarks (25 MHz clock).

Benchmark	Run Time	Simulation Time
compress	≈ 40 min	4.39 ms
edn	≈ 85 min	4.93 ms
crc	≈ 65 min	4.06 ms

Table 3.4 : Average power consumption results obtained by Vivado, using SAIF files.

Benchmark	Average Dynamic Power
compress	0.010 W
edn	0.006 W
crc	0.007 W

calculations) [42]. Structural properties of these benchmarks are given at Table 3.2.

2. SPI stimuli of these compiled codes were generated in PULPino working environment, using file building scripts.
3. During the ModelSim post-implementation simulation, each of these stimuli was sent via an SPI simulation model block to the instruction RAM of PULPino and the simulation was run until the code execution was completed.
4. ModelSim-generated SAIF files were moved back to Vivado again; to act as input files for Xilinx power estimation tool.

For each benchmark run, different SAIF files are given as simulation activity input, along with PULPino post implementation model. Then the Xilinx power estimation tool produces an output, that can be a text report or a graphical report, which includes both average static power consumption and average dynamic power consumption estimations. Since the static power depends on the FPGA device itself, thus being same for all benchmarks, it is deducted from the results and only the dynamic power is taken in consideration. Obtained average dynamic power consumption results are presented in Table 3.4. The timing results are also given in Table 3.3. It should be noted that the timing results also includes the part when the stimulus being transferred over SPI module, causing run times to be longer than expected. On the other hand, sending via SPI part was not included in SAIF files.

4. MOMENTARY POWER CONSUMPTION PROFILE GENERATION

The load capacitance value in digital circuit depends on line lengths and fan-out. When trying to generate momentary power consumption profile from VCD files, the design's mapping and routing information could be used in conjunction with switching activities. However, this will only affect the resolution of actual power consumption values and it requires more computing time during generating momentary power consumption profile. Since the concern to observe sudden changes in power consumption becomes not entirely necessary, and post-implementation VCD files become ineffectively large; it was decided to just use the information of behavioral simulation switching activity dump. Post-implementation activity files being far larger than behavioral files is the naturally expected result of actual FPGA primitive models replacing the written HDL (which what actually happens during mapping stage of implementation), which greatly increases the total number of internal signals. Due to the same reason, post-implementation simulation run times became longer as well.

Main reason for getting a momentary power consumption profile is to use it for differential power analysis [16] on the implementations of cryptographic algorithms during early design stages. With the acquired information, it is aimed to test the behaviour of a cryptography algorithm implementation against side channel attacks during early simulation stages of hardware designing process.

4.1 Generating Momentary Power Consumption Profiles using VCD

Since SAIF files only provide information for summative switching activity, this file type is not suitable for sourcing information on instantaneous switching activity. Thus; VCD files can be used instead, which gives a signal's value for each clock cycle for the duration of a simulation.

At first, the process of generating the VCD files were performed with the same steps stated in 3.3.2, using the Verilog HDL system functions "dumpvars" and "dumpfile". These statements must be added to the corresponding sections of the design testbench.

However, this method is discarded since the VCD file sizes were unpractically large; even so when selective dumping had been performed by only dumping the values of specific parts of the core. Due to these problems, it was decided to use behavioral simulation for generating momentary power consumption profile. This will result in discarding the actual path length and fan-out effect on the power consumption estimation.

By using the VCD file produced from behavioral simulation, it is assumed that all the load capacitances of all the lines given in Equation 3.1 are constant and roughly the same. It is obvious that the power supply voltage V_i in Equation 3.1 is the same for all lines in the same FPGA device. Hence, in order to calculate dynamic power consumption in Eq. 1, we only use switching activity. We claim that momentarily power consumption information obtained in this way will be sufficient for DPA on implementation of cryptographic algorithms, since the main concern is not the actual value of power consumption, but the behaviour of the temporal switching activity.

Behavioral simulation is performed on ModelSim again as explained in Chapter 2, while recording switching activity in the desired parts of the simulation. Using behavioral analysis indeed shrunked the output file size; but still only some portions of the core, Arithmetic Logic Unit (ALU) and core register blocks, dumped to achieve an acceptable VCD file size.

VCD files are generated using the tabular list format of ModelSim (.lst files), for easier parsing in MATLAB [43]. This type of value dumping does not require the usage of Verilog dumpfile system functions. In order to initiate list dumping, the command below can be included in the related .do file:

```
add list -r /tb/top_i/core_region_i/CORE/RISCV_CORE/* run
-all
```

In this statement “add list“ opens up a new list, “-r hierarchical_element_name“ indicates that the specified element will be added to the list along with all it’s internal signals. In the example, the element name “/tb/top_i/core_region_i/CORE/RISCV_CORE/*“ points to the entire core region of PULPino, while the “*“ symbol denotes that all submodules under RISCV_CORE module will be included in the scope. To sum up, this statement adds all the specified

signals, along with their submodules and all of their internal signals, to a newly created list.

After completing the simulation the list can be extracted to a text based .lst file with the command below:

```
write list -window .main_pane.list.interior.cs.body
destination_dir/list1.lst
```

The “write list command“ saves the value change dump stored in the list file to a tabular file with the name specified in “destination_dir/list1.lst“. The file name is user determined. The “-window object_name“ option shows that the list to be written is a windowed list named “object_name“ ; “.main‘_pane.list.interior.cs.body“ for this example where “list“ word in the object indicator holds the name of the windowed list.

A screen view of how these saved tabular list files look like can be seen at Figure 4.1. Leftmost column holds the meaningful time values of simulation, while the other columns show the levels of specified signals in that moment. In this figure, only a cut set of the list is given due to list being too large, since there are too many signals added to the list to be recorded.

4.2 Processing the Value Dump Files

To extract data from VCD files, we wrote a MATLAB script for reading VCD files which first parses the switching information in the files, then compares the value changes of each recorded signal whenever a value-level change happens in the circuit. This script is given in Appendix A.2.

The script first imports the text based dump files and takes the parts that states current times and actual list of signal values for that given time. Time values and the signal value list are separately stored in two matrices. The script then compares the value differences of the adjacent rows of signal value list matrix, gets the sum of the value changes found in comparison between two rows and stores them to their related time index.

1	ps	/tb/top_i/core_region_i/CORE/RISCV_CORE/clk_i	/tb/top_i/core_region_i/CORE/RISCV_CORE/core_id_i	/tb/top_i/core_region_
2	delta	/tb/top_i/core_region_i/CORE/RISCV_CORE/rst_ni	/tb/top_i/core_region_i/CORE/RISCV_CORE/cluster_id_i	
3		/tb/top_i/core_region_i/CORE/RISCV_CORE/clock_en_i	/tb/top_i/core_region_i/CORE/RISCV_CORE/instr_req_o	
4		/tb/top_i/core_region_i/CORE/RISCV_CORE/test_en_i	/tb/top_i/core_region_i/CORE/RISCV_CORE/instr_gnt_i	
5		/tb/top_i/core_region_i/CORE/RISCV_CORE/boot_addr_i	/tb/top_i/core_region_i/CORE/RISCV_CORE/instr_rvalid_i	
6	118120000	+5	St0 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
7	118140000	+2	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
8	118140000	+4	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
9	118140000	+5	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
10	118140000	+6	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
11	118140000	+7	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
12	118140000	+8	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
13	118140000	+9	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
14	118140000	+10	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
15	118140000	+11	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
16	118160000	+2	St0 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
17	118160000	+3	St0 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
18	118180000	+2	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
19	118180000	+3	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
20	118180000	+4	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
21	118180000	+5	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
22	118180000	+6	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
23	118180000	+7	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
24	118180000	+8	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
25	118180000	+9	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
26	118180000	+10	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
27	118200000	+2	St0 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
28	118200000	+3	St0 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
29	118220000	+2	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
30	118220000	+4	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
31	118220000	+5	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
32	118220000	+6	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
33	118220000	+7	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
34	118220000	+8	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
35	118220000	+9	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
36	118240000	+2	St0 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
37	118260000	+2	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
38	118260000	+4	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
39	118260000	+5	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
40	118260000	+6	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
41	118260000	+7	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
42	118260000	+8	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00
43	118260000	+9	St1 St1 St1 St0 00000000000000000000000000000000	0000 000000 1 St1 St1 00

Figure 4.1 : A cut view from a tabular value dump file belonging to PULPino simulation.

Using the script, total switching for each moment of the simulation is calculated and plotted. Finally obtained momentary power consumption profiles of PULPino ALU and core registers for each benchmark can be seen at Figures 4.2 and 4.3, with their zoomed versions given at Figures 4.4 and 4.5.

When the results of two different power analyses are compared, it could be seen that according to Figure 4.2, compress operation caused more hectic and higher amount of switching compared to the other benchmarks, while crc showing mostly consistent switching behaviour with averagely lesser in amount than it was in compress. Sudden rises in switching amounts are much more sparse compared to other benchmarks, and the maximum switching amount observed in unit time is also lower. Although it may not be completely accurate to compare a momentary approximate graphical result with an average power consumption estimation of just some vital sections of the system, in a sense this interpretation is apparently overlapping with the results in Table 3.4, where $P_{compress} > P_{crc} > P_{edn}$ was obtained. Similar observations can also be made for core register switching profiles given in Figure 4.5.

After seeing that the VCD processing script produces outputs that are expected in terms of shape, activities regarding the main encryption algorithm that will be the main test unit are taken and examined in the next section.

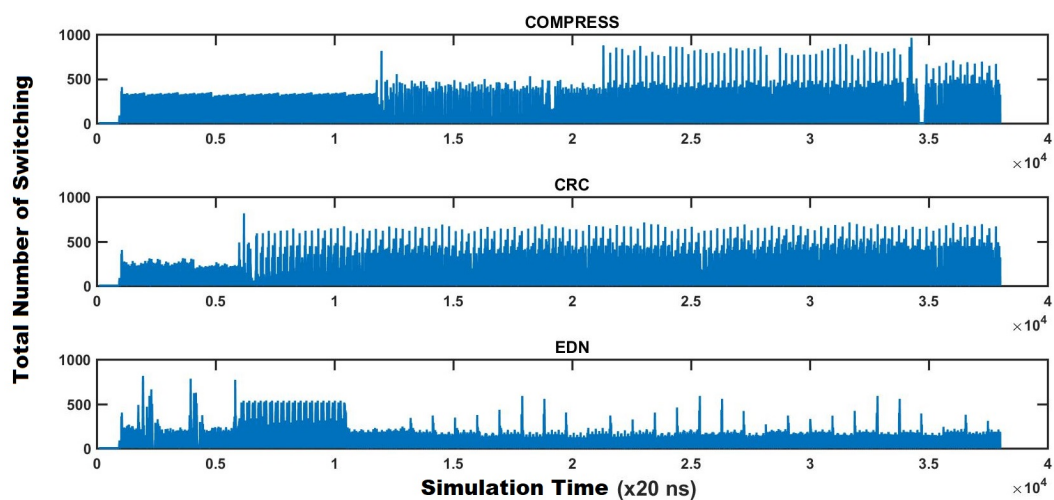


Figure 4.2 : Momentary switching profile for arithmetic logic unit.

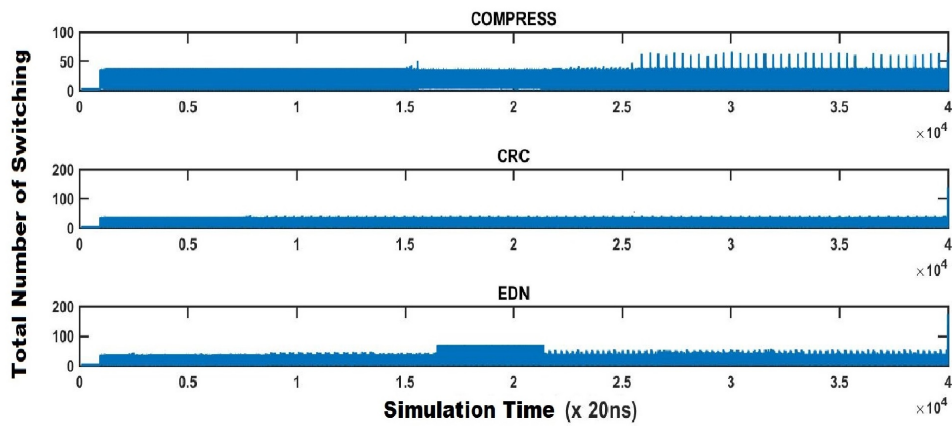


Figure 4.3 : Momentary switching profile for core registers.

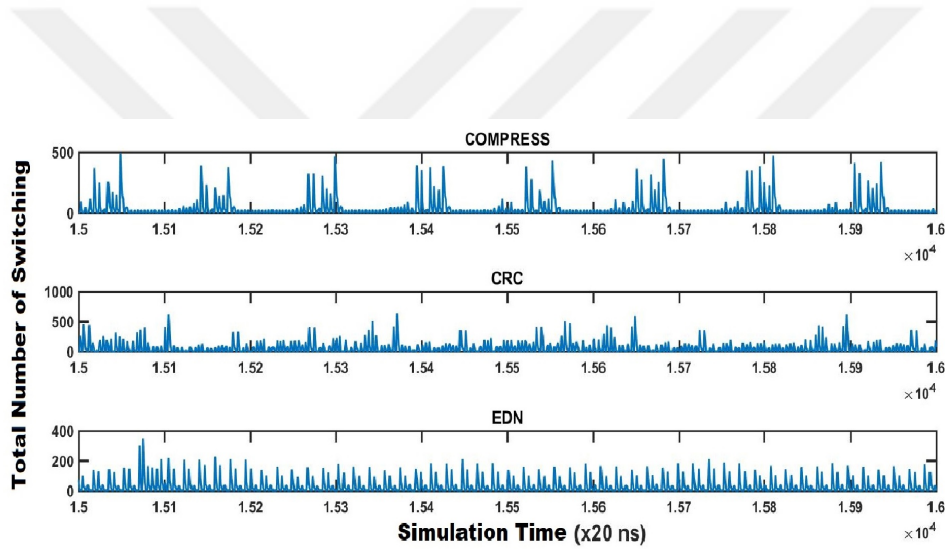


Figure 4.4 : Momentary switching profile of ALU (zoomed).

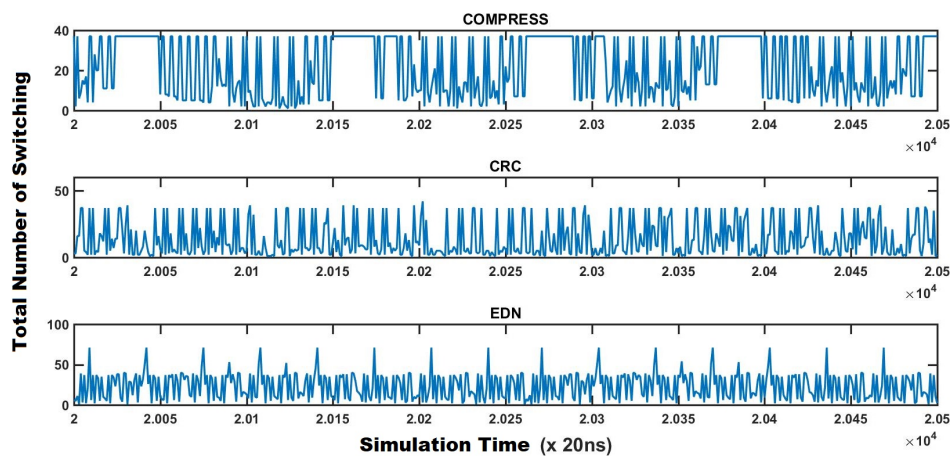


Figure 4.5 : Momentary switching profile of core registers (zoomed).

4.3 Momentary Power Consumption Profile of an Advanced Encryption Standart Algorithm

4.3.1 Overview of Advanced Encryption Standart

As the purpose of creating power consumption profile is to use them in DPA analysis of implementations of cryptographic algorithms, an AES [44] algorithm was also executed on PULPino in ModelSim environment.

The AES algorithm is an encryption standart specified by the United States' NIST. The original name of the algorithm was called Rijndael, a combination of the names of it's founders [45], and it's the chosen algorithm for AES among the fifteen algorithm purposals.

The Rijndael algorithm selected for AES has three adoptions depending it's key size; being either 128, 192 or 256 bits. A 128 bit one is used during this work.

The AES algorithm contains four fundamental operations in it's rounds; named Subbytes, shift rows, mix columns and adding round key. In one round these operations are performed once, in the given order. A 128 bit AES operation consists of ten rounds. A general schematic of 128 bit AES is given in Figure 4.6.

At the beginning of the algorithm, a plain text is stored as a matrix in the form given in Figure 4.6, where each element represents a byte of the plain text. A subbyte operation seen at the start of a round, replaces the bytes given as input using a matrix called S-Box [44]. This is also a reversible operation. An illustration of this stage can be seen in Figure 4.7.

Outputs produced from subbytes operation then processed in shift rows block. This block circularly shifts each row by a specific amount. First row stays as it is, second row is shifted by one; and for all other rows, the elements are shifted by one deficient of their row index number, such as two shifts by third row and three shifts for fourth row. Figure 4.8 illustrates this operation.

Thirdly, the mix columns operation is applied. This operation takes four byte inputs column by column and applies a specific reversible linear transformation to produce it's output [44]. The way of its operation is portrayed at Figure 4.9.

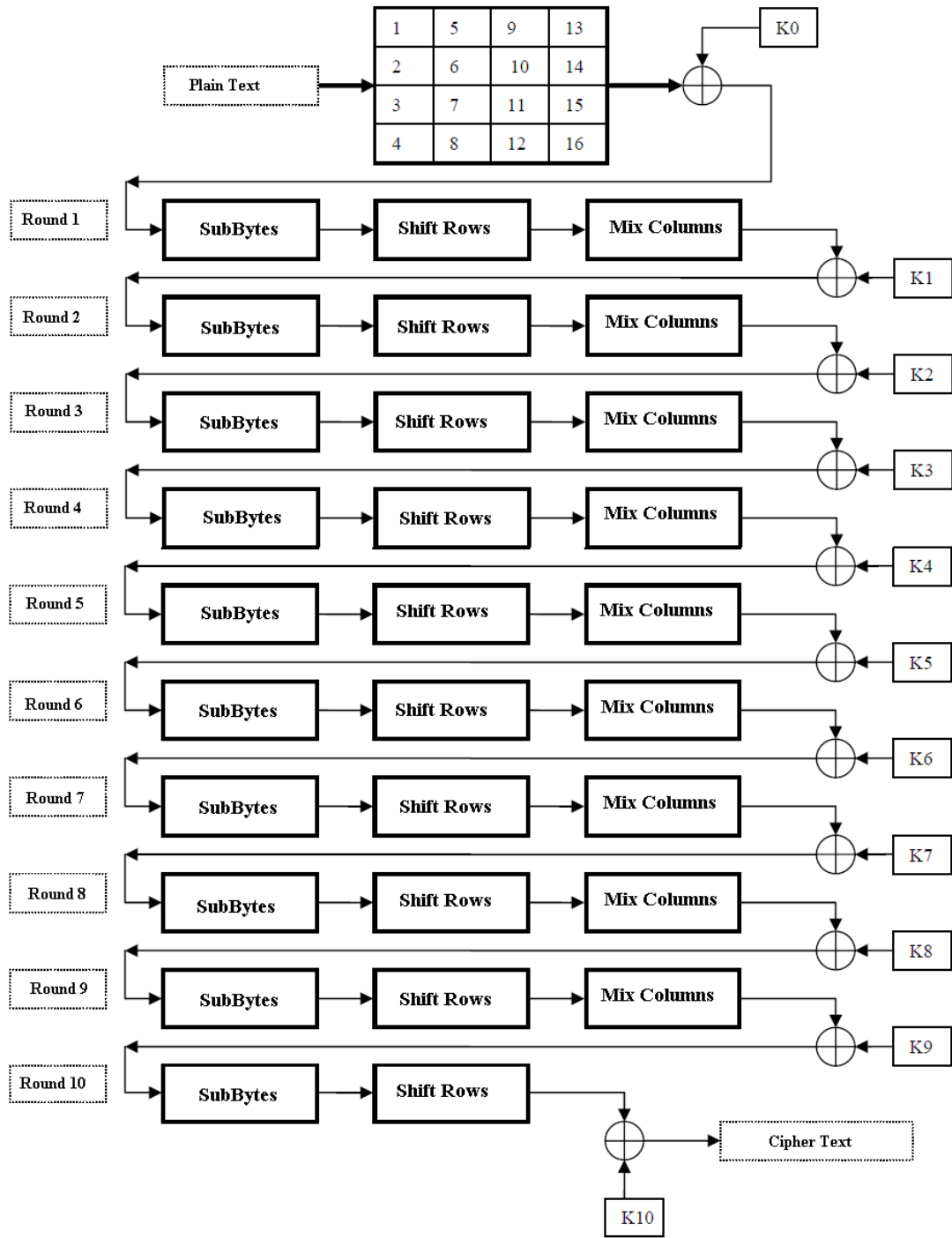


Figure 4.6 : Block schematic of a 128-bit AES algorithm [4].

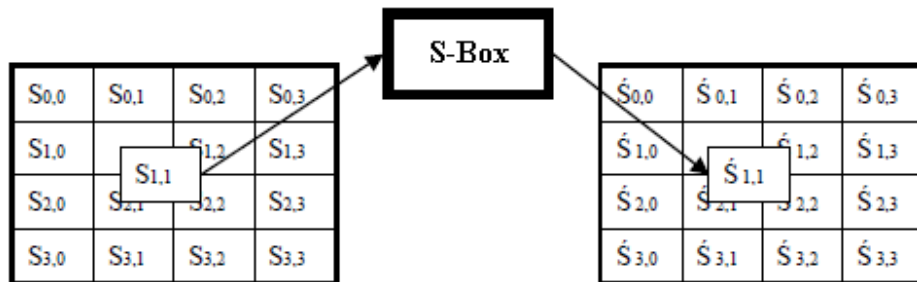


Figure 4.7 : The SubByte operation [4].

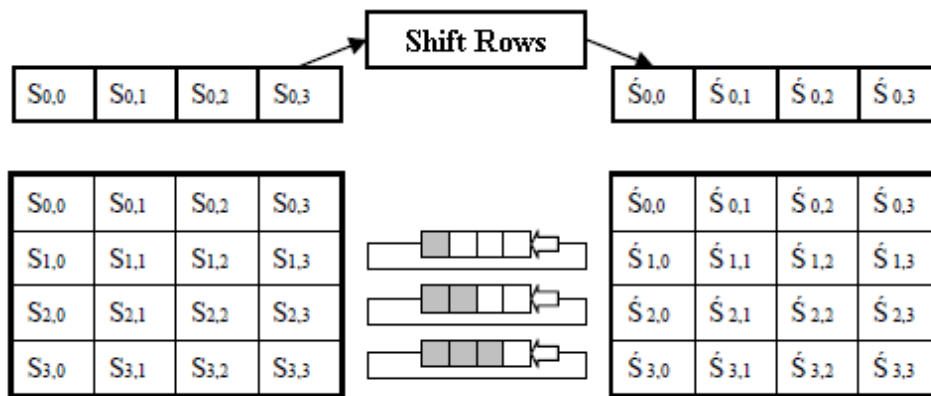


Figure 4.8 : The Shift Rows operation [4].

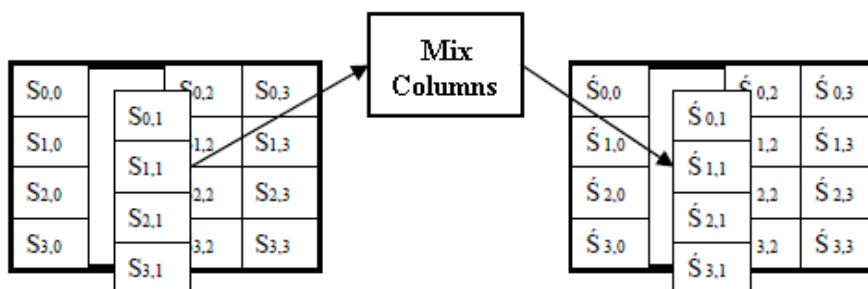


Figure 4.9 : Operating of Mix Columns [4].

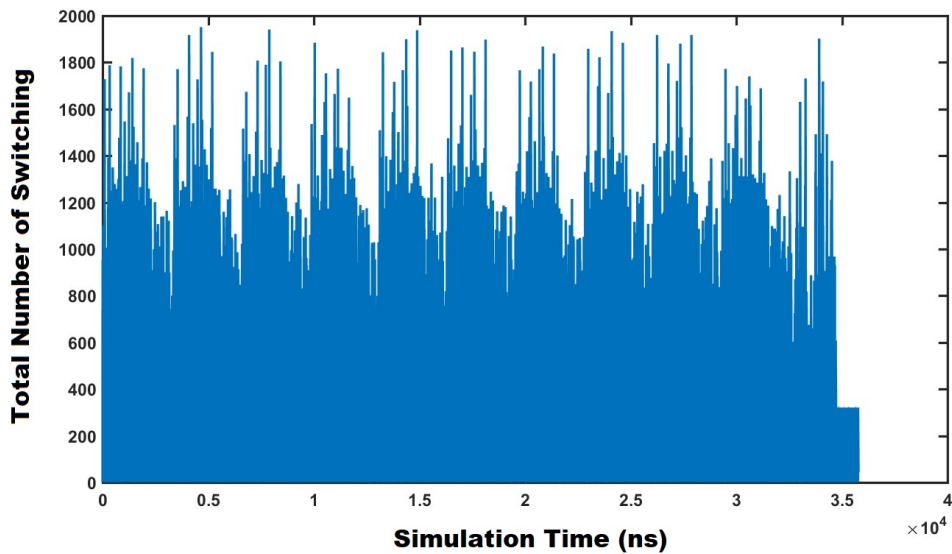


Figure 4.10 : Momentary power consumption profile of entire AES encryption stage for one message.

As the last operation of a round, processed 4x4 matrix is added with a stage sub key, which is generated by a scheduling process in the Rijndael algorithm [44]. These sub keys are shown by K 's in Figure 4.6. The addition is performed by bitwise exclusive or (XOR) operations.

As for the AES implementation in C, the specially arranged AES encryption algorithm within PULPino working environment was modified and used. This algorithm can be seen in Appendix A.3. Steps explained in Section 3.3.2 are repeated and VCD files are generated and momentary power consumption profiles are plotted as a simple demonstration. As stated, the AES encryption is consisted of ten rounds. In Figure 4.10, the ten rounds of the AES encryption are clearly visible by the sudden peak groups, further solidifying our claim that still being able to distinguish the abrupt changes in the power consumption activity, even for a behavioral simulation with capacitance values taken as constant.

4.3.2 Masking Advanced Encryption Standart

Even though AES appears to be resistant to common cryptanalysis techniques, some hardware realisations of it can be open against DPA attacks [5]. Thought behind the attacks is predicting the sections of the key used within algorithm operations via various side effects. The protection mechanism comes from the idea of altering this

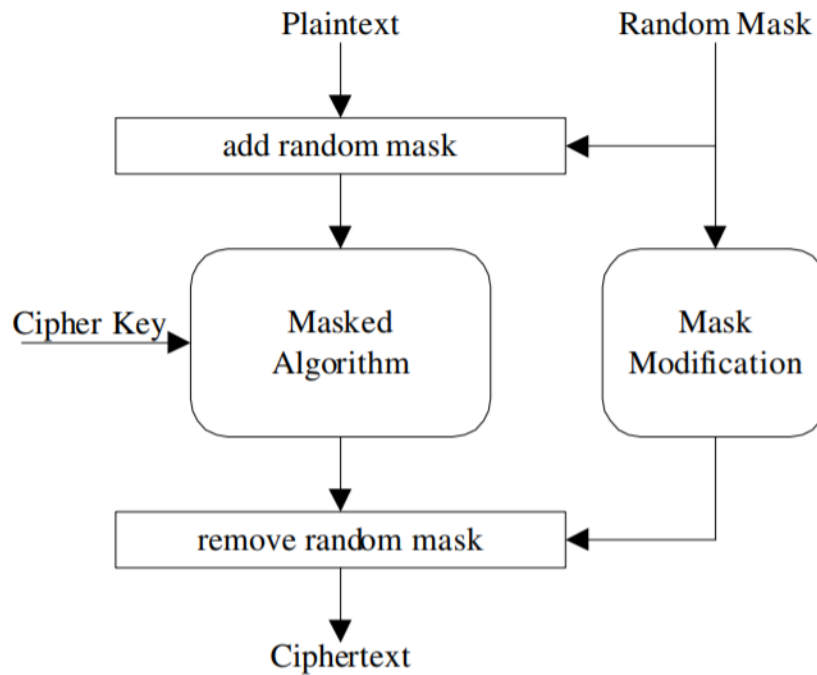


Figure 4.11 : Overview of basic masking [5].

vital information used inside algorithm functions. Performing this altering procedure is named as masking.

The basic principle is to add a randomly generated mask value to the plaintext before the first stage, then remove this mask to get the actual encrypted result. In order to negate the mask value effect, mask modification and unmasking blocks are employed to keep track the required operations. A portrayal of this basic mentality is given in Figure 4.11.

The AES consists of linear and nonlinear functions. Since the result of linear functions taking input of masked plaintexts will be the sum of the same functions when mask value and plaintext value given as input, unmasking these functions are straightforward. These linear functions include the round key addition, MixColumns and ShiftRows operations. The nonlinear function is the SubBytes operation, and more advanced approaches are required to unmask this process. One approach given in [46] is used to demonstrate the protection provided against the side-channel attack we perform in this work.



5. DIFFERENTIAL POWER ANALYSIS USING MOMENTARY POWER CONSUMPTION PROFILES

The penultimate chapter of the thesis examines the simulation based side channel analysis environment we offer. The main concepts are explained, details regarding the environment are given and the analysis results are presented.

5.1 Differential Power Analysis

Differential Power Analysis is an attack that can extract vital information from a device's real-time power consumption measurements [16]. First, a large number of runtime measurements are taken from a working device. Secondly, some models are predicted and a power consumption estimation is made from these, to be used in conjunction with the real time results. Then the results are compared and their correlation is checked. At the end, according to the analysis, a guess on the vital algorithm information is tried to be extracted [17]. In this section, a method to do a similar attack during the course of simulation stage of a design is explained.

A total switching number per clock cycle matrix P_{ij} given in Equation 5.1 is constructed for n test messages, where each operation takes up k clock cycles of time in total, also i and j denoting the i 'th clock cycle for j 'th test input. Afterwards, an average switching number matrix P_{avg} is obtained by taking average of the elements of each individual row. This provides an average switching count for each test case. P_{avg} is a vital element in correlation analysis since this will be reference vector that contains information about device's switch activity.

$$P_{ij} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & \cdots & \cdots & p_{1,k} \\ p_{2,1} & \cdot & \cdot & \cdot & \cdot & \cdot \\ p_{3,1} & \cdot & \cdot & \cdot & \cdot & \cdot \\ \vdots & \cdot & \cdot & \cdot & \cdot & \cdot \\ \vdots & \cdot & \cdot & \cdot & \cdot & \cdot \\ p_{n,1} & \cdot & \cdot & \cdot & \cdot & p_{n,k} \end{bmatrix}, P_{avg} = \begin{bmatrix} P_{avg1} \\ \vdots \\ \vdots \\ P_{avgn} \end{bmatrix} \quad (5.1)$$

Script extension to create these matrices are given in Appendix A.4. Figure 5.1 shows the initial XOR operation of AES for one message, with respect to clock pulses.

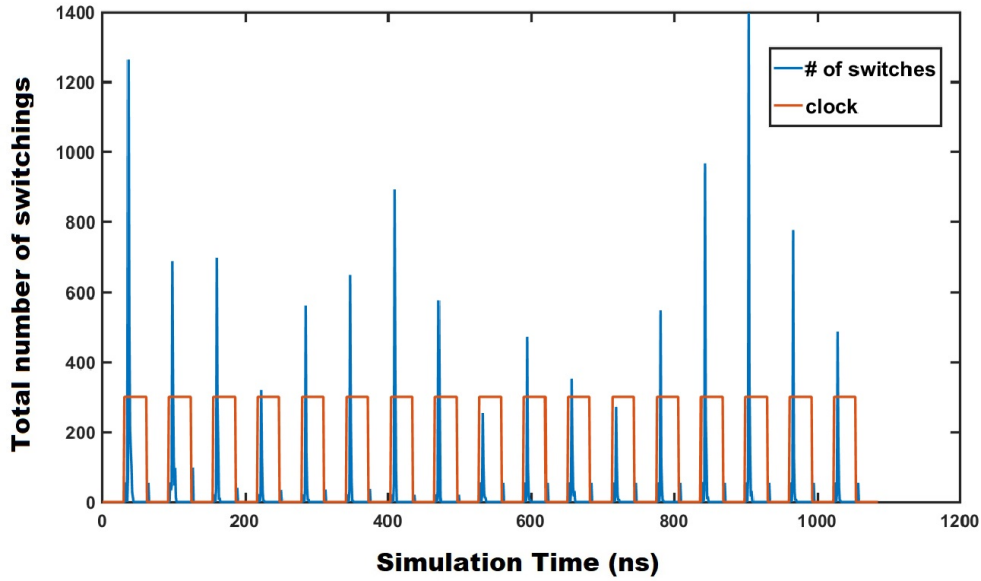


Figure 5.1 : Switching activity profile of AES initial XOR operation for one message, with respect to clock pulses.

The section of the AES algorithm that the attack is performed (the section with the recorded switching activity) then emulated in MATLAB. Using this emulation, all possible values that one byte of the key can take are consecutently given here as input, along with the messages to be used in AES simulation. The bit differences of the input message and the processed output message then compared. With this comparison, total number of 0 to 1 transitions are counted for each message and the results are recorded in a switching weight matrix named S (Equation 5.2). A byte consists of eight bits, hence it has 2^8 possible values. Thus, the size of this newly created matrix will be n -by-256, where n represents the total number of test messages, just as before.

$$S = \begin{bmatrix} s_{1,1} & s_{1,2} & s_{1,3} & \dots & \dots & s_{1,256} \\ s_{2,1} & \cdot & \cdot & \cdot & \cdot & \cdot \\ s_{3,1} & \cdot & \cdot & \cdot & \cdot & \cdot \\ \vdots & \cdot & \cdot & \cdot & \cdot & \cdot \\ \vdots & \cdot & \cdot & \cdot & \cdot & \cdot \\ s_{n,1} & \cdot & \cdot & \cdot & \cdot & s_{n,256} \end{bmatrix} \quad (5.2)$$

In the final step, correlation analysis is performed among the matrix P_{avg} and each column of matrix S one by one. Resulting correlation coefficients are stored and the

one with the highest value should correspond to the correct value of the wanted key byte. Repetitions for other key bytes then could be applied for obtaining remaining AES key bytes.

5.2 Test Attacks and Results

Testing schemes for this work involves applying side channel attacks on the standart and masked AES designs, for a large number of individual plain text inputs. First type of attack will include the main input section of the algorithm, where the raw form of the plain text is added with the first sub key of the process. In another perspective, this also means attacking on the initial XOR operation in the encryption process. A second type of attack is performed on the S-box output of the first round, which is the first S-box operation that is performed in the algorithm. This operation remotely uses the result of the first XOR operation output of algorithm block. In fact, this attack is performed initial sub-byte block of the algorithm, which contains S-box transformation.

Since the dump file sizes for entire simulation takes up quite large space even for one plain text input, final outputs become massive and grow unpractical when trying to hold activity records for thousands of test messages. For this reason, certain parts of the AES operation is better to be taken for size reduction. Though, it is very difficult to predict at which time the operation we want to record power of will be executed. Although, this problem is solved by using a triggering signal during the simulation when writing the value changes to a list. One of the idle output pins on PULPino is set in C code just before the section that is being attack starts executing. A script was written so when this signal is set, the recording to VCD file starts. After the execution of the related part is finished, the same pin is set to zero, indicating the simulator to halt dumping switching activity.

5.2.1 Attack on Initial XOR Stage

For our first testing attack case, an attack to the first byte of the first XOR operation is planned to be performed, as per the side channel attack usage. In the AES C code, just before the operations in question is performed, we set a specific unused pin on the PULPino General Purpose Input and Output (GPIO) pin set. This pin being on high value will trigger the value change dump within the simulation. Right after the

code statement that performs the operation that is desired to be power measured, the same GPIO pin will be reset to stop recording. This will allow to exactly record the one byte XOR operation section of the simulation for each individual plain text input. Since we are only recording certain parts of the simulation, the switching dump list will have sudden jumps in the time column, with each jump indicating the start of the power recording of a new message. Our parsing script contains a section that detects these jumps, then separates and stores each message's power recording. This part of the script can be seen at Appendix A.5.

First byte of the key that is used in the algorithm is arbitrarily set to hexadecimal A0, which is 160 in decimal. As per the procedure presented at Section 5.1, predicted switching activity numbers for each possible key value is obtained, using the XOR operation among the targeted key byte and corresponding plaintext byte as expected model. For various numbers of different plaintexts, correlation value by guessed key value graphics are obtained. It is observed that for this attack, the correlation profile settles even for relatively small number of plaintexts, only difference being the diminishing correlation value as the number of plaintexts increases. The order of correlation magnitude is also observed to be consistent for dominant peak values, where key value decimal 96 (hexadecimal 60) being the highest correlated prediction. Figure 5.2 shows the correlation value by key guess value for 100,200 and 300 plaintexts, and portrays the described situation. Figure 5.3 presents the correlation profile for 4000 plaintexts, and still does not provide a drastic difference from Figure

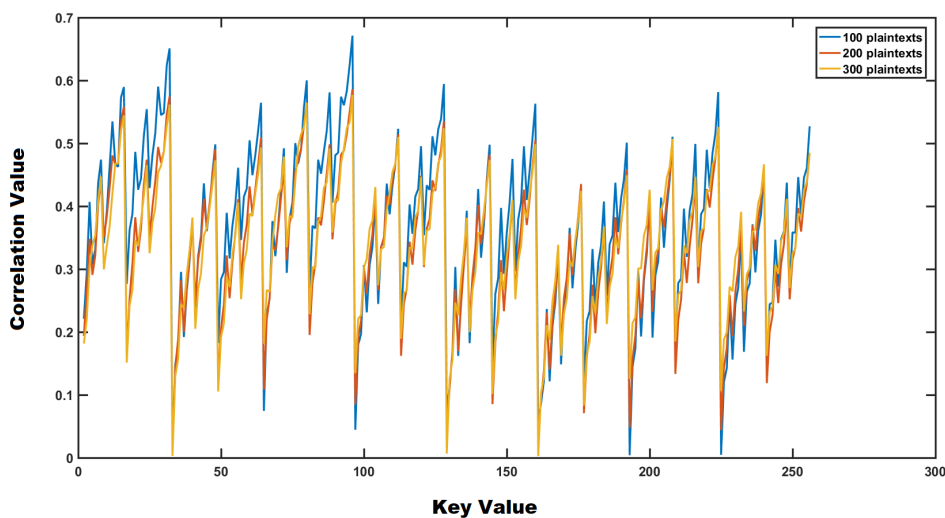


Figure 5.2 : Initial XOR attack results for plaintext numbers of 100,200 and 300.

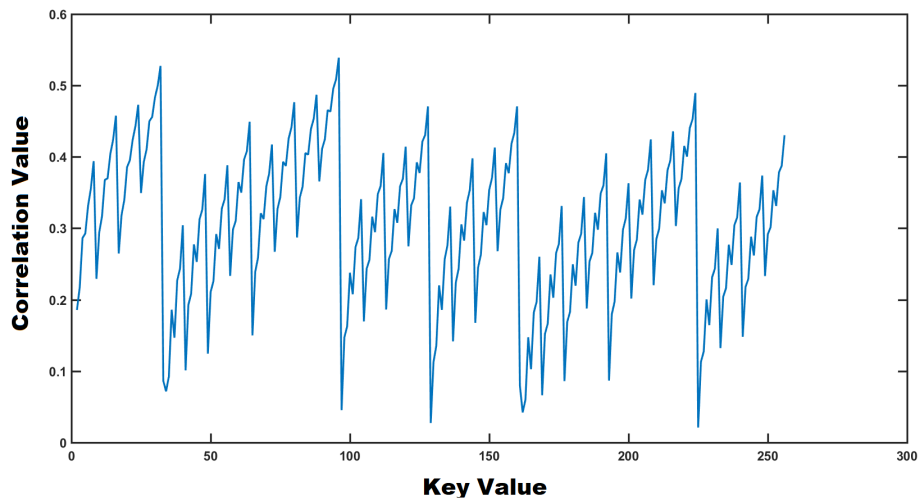


Figure 5.3 : Initial XOR attack result for 4000 plaintext trials.

5.2's profiles or a distinctive peak value. For 4000 plaintexts, the key guess with the largest correlation value is still hexadecimal 60; while the correlation value for the real key, hexadecimal A0, is the guess with tenth largest correlation value.

Results show that using this attack, correct key could not be guessed; however, having a consistent correlation profile with a periodic profile shape suggests that the setup works correct, but enough randomness cannot be provided from just an XOR operation to distinguish a correct guess. In the C implementation, result of the input plaintext is overwritten by the XOR output, therefore the total number of non-zero bits in the other operand, key guesses in this case, directly determines the number of changes that will occur at the XOR's output. This means that the number of switches will be directly determined by the Hamming weight of the key guess number that is used. This supports the distinct correlation profile and unsuccessfulness of guessing the correct key. This result necessitated to find an attack on an operation block with a much more distinct domain and range matching relation.

5.2.2 Attack on Initial SubBytes Stage

Following the failure of correct key prediction with the initial XOR attack, a more comprehensive attack is defined, which covers a larger portion of the encryption and uses two chained blocks as its model. The attack is performed on the first round's SubByte block, specifically the output of the non-linear S-box operation. This operation is realised in C code as a look-up table, which will not be directly related

to the Hamming weight of the operand. Until coming up to this output, the XOR operation described in Section 5.2.1 is performed first, then the result of the S-box input that is generated from the XOR result of the first key byte is used to get the S-box output. Hence, the predicted model will be using this two operations in the correlation calculations. Also, the simulation scripts and C codes will be extended to include the value change activity of this portion of the executions in VCD files.

5.2.2.1 Results on standart AES

To perform the attack on the S-box output of the encryption, a total of 2000 executions are performed on the processor, and the correlation analysis is performed with a model overlapping with the initial XOR and S-box operations. The most significant byte of the key is set to decimal 160 again, similiarly the previous attack. Figure 5.4 shows that after 2000 plaintext trials, the correct key is predicted by the program. Furthermore, a correlation profile by the number of plaintext graph is obtained in Figure 5.5.

This figure contains 256 different plots, each representing a different key value between 0 and 255. The plot associated with the correct key 160 is given in red, and it can be seen that after around 200 plaintexts, the correct key value is clearly distinguishable.

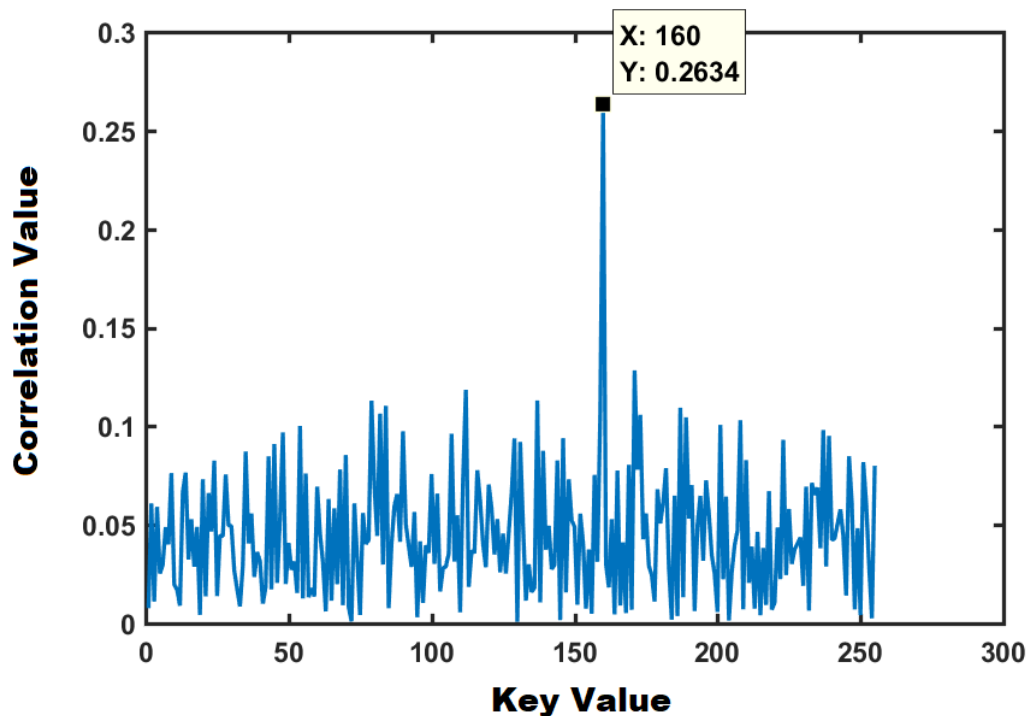


Figure 5.4 : Initial SubByte attack result for 2000 plaintext trials.

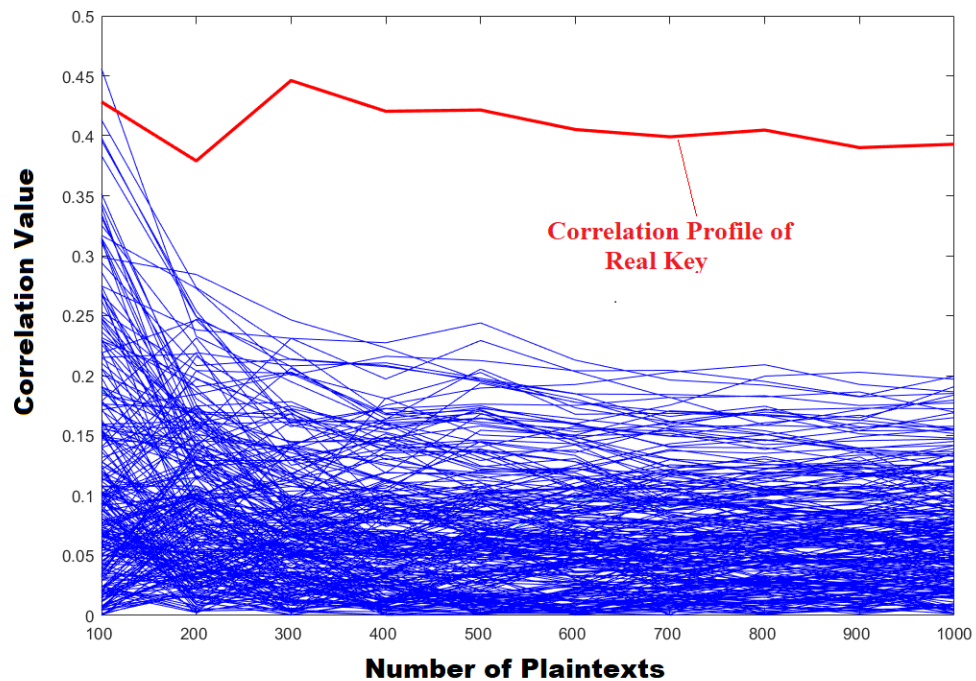


Figure 5.5 : Correlation profiles of possible key values for different number of plaintexts.

This satisfies the expectations of using a higher complexity operation to lessen patterns between domain-range match-ups and making the correct key correlation distinguishable.

5.2.2.2 Results on byte-Masked AES

After successfully obtaining the most significant key byte, same attack procedure is applied on a masked AES implementation to demonstrate the side channel attack protection provided by the masked realisation. As the masked algorithm, the byte-masked implementation described in [46] is used in C language. As for the testing arrangements, a setup similar to the standard AES test is used. For the predicted model, first XOR and first SubByte blocks of standard AES are used. Maximum 2000 plaintexts are given to the program and correlation values are plotted. Figure 5.6 shows the correlation by key guess value, and it can be observed that there are no uniquely distinguishable peak or a pattern in the profile. Moreover, Figure 5.7 shows the key guess correlation profiles by the number of plaintexts shows diminishing correlation values for all key guess correlation plots. The correct key byte which is shown in red is expressing a continuous non-distinguishable profile for the entire figure, satisfying the expectations.

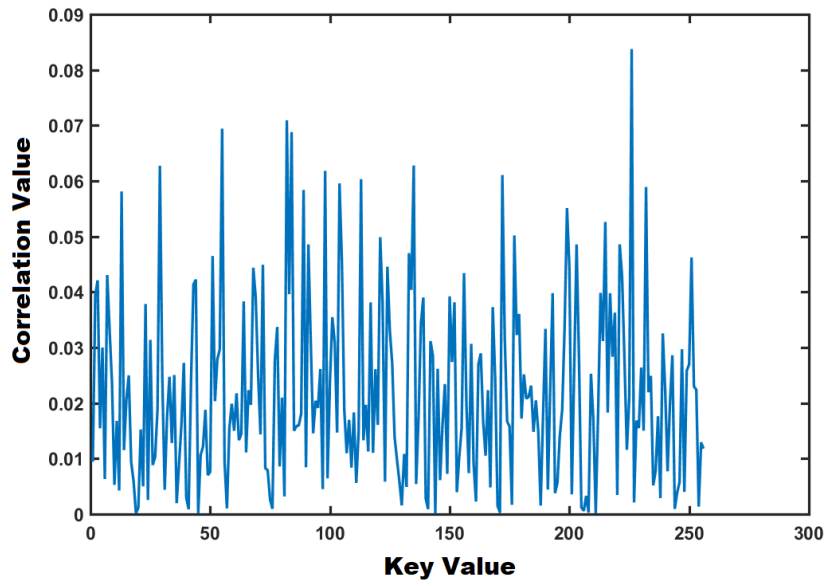


Figure 5.6 : Initial SubByte attack result for 2000 plaintext trials.

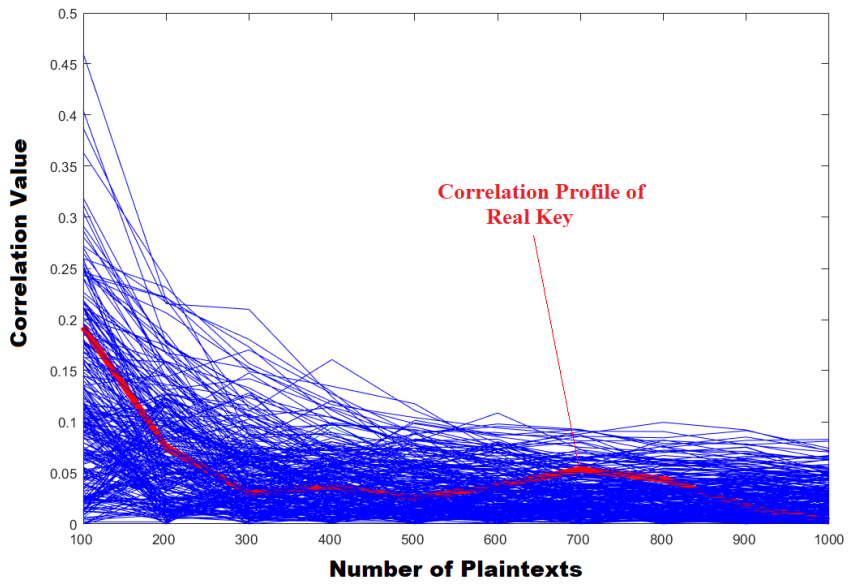


Figure 5.7 : Correlation profiles of possible key values for different number of plaintexts.

6. CONCLUSION

During the course of this work, a simpler simulation stage based side channel attack resistance test was offered that is analogous to DPA, using just the simulation switching activity of a digital design, and it is demonstrated on a softcore processor system. To this end, at first, a RISC-V implementation named PULPino was introduced and its software and working environment usage is explained. Then, post-implementation simulation and power consumption estimation flows are added to PULPino's working environment. On the way, hardships of adding and generalizing out of the frame custom flows for a softcore processor's standard work flow are observed. Afterwards, power estimation methods are discussed and several of these approaches are tested on our simulation based resistance test. It is observed that taking entire post-simulation switching activity to consideration becomes unefficient. Next, the offered resistance test methodology is explained with some examples. Then it is shown that even with just keeping track of number of switchings in a system during behavioral simulation was sufficient to successfully obtaining the key or narrowing the guesses required to crack.

In the end, way to work with an open source softcore processor system was experienced, and steps to these are explained in a user manual like manner to be a reference for further studies. These steps included tasks expanding from setting up the processor to showing . Also the main hypothesis was showed on one of these softcore processor systems, which is being able to successfully guess the key values of an AES algorithm during simulation stage of the design; without even being need to undergo a complete power estimation step. Codes to perform this resistance test are also does not depend on any third party power estimation tool. Furthermore, since the simulation environment is noise-free, it certainly indicates if a design is resistant to side channel attacks or not, since performing this with real time measurements will include noise and parasitics and will be harder to crack in actual implementation.

Future work includes generation of a common implementation flow and a parametric testbench environment standart for various open source processor employing systems. This will lead to being able to test and compare various softcore processor implementations on the same setting, while also reducing the adaptation and specific realisation environment learning steps of the related work. Test models for various peripherals or blocks can also be made for inter-compatibility, like a generalized RAM block to replace black box IP's used for FPGA implementation stage. On the cryptography side, several blocksare though to be offered for efficient side-channel attack and DPA protection, and comparisons for these blocks can be made. Perfomance comparisons of cryptographic designs may also be made on various open source processor systems. Finally, current open source instruction set architectures can be extended to be efficient on cryptographic applciations, and their realizations could be demonstrated either on FPGA or as an application-specific integrated circuit.

REFERENCES

- [1] **PULP Platform**, <https://pulp-platform.org/>, online accessed: 20.04.2019.
- [2] **Traber, A. and Gautschi, M.**, (2017). PULPino Datasheet, Integrated Systems Lab, ETH Zurich, Switzerland.
- [3] **Zaruba, F., Stucki, S., Pullini, A., Haugou, G., Flamand, E., Gürkaynak, F.K. and Benini, L.** (2016). PULPino: A small single-core RISC-V SoC, *3rd RISC Workshop*, Oracle Conference Center, Redwood Shores.
- [4] **Kayış, H.**, (2006), AES Uygulamasının FPGA Gerçeklemelerine Karşı Güç Analizi Saldırısı.
- [5] **Pramstaller, N., Oswald, E. and Mangard, S.** (2019). A Masked AES ASIC Implementation.
- [6] **Shannon, C.E.** (1945). A mathematical theory of cryptography, *Bell Telephone Labs report*.
- [7] **Nade1, J.B. and Sarwadnya, R.V.** (2015). The Soft Core Processors: A Review, *International Journal of Innovative Research In Electrical, Electronics, Instrumentation and Control Engineering*, pp.197–203.
- [8] **Chinedu, O.K., Genevera, E.C. and Akinyele, O.O.** (2011). Hardware description language (HDL): An efficient approach to device independent designs for VLSI market segments, *3rd IEEE International Conference on Adaptive Science and Technology (ICAST 2011)*, pp.262–267.
- [9] **Stallings, W.** (1988). Reduced instruction set computer architecture, *Proceedings of the IEEE*, 76(1), 38–55.
- [10] **Waterman, A. and Arsanovic, K.**, (2017). RISC-V Instruction Set Manual: Volume 1, EECS Department, University of California, Berkeley.
- [11] **Asanović, K., Avizienis, R., Bachrach, J., Beamer, S., Biancolin, D., Celio, C., Cook, H., Dabbelt, D., Hauser, J., Izraelevitz, A., Karandikar, S., Keller, B., Kim, D., Koenig, J., Lee, Y., Love, E., Maas, M., Magyar, A., Mao, H., Moreto, M., Ou, A., Patterson, D.A., Richards, B., Schmidt, C., Twigg, S., Vo, H. and Waterman, A.** (2016). The Rocket Chip Generator, **Technical ReportUCB/EECS-2016-17**, EECS Department, University of California, Berkeley, <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html>.

- [12] **VectorBlox**, (2013), ORCA-FPGA optimized RISC-V, <https://riscv.org/wp-content/uploads/2016/01/Wed1200-2016-01-05-VectorBlox-ORCA-RISC-V-DEMO.pdf>, online; accessed 13.01.2019.
- [13] **Li, L. and Gautschi, M.** (2017). Approximate DIV and SQRT Instructions for the RISC-V ISA: An Efficiency vs Accuracy Analysis, *27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pp.1–8.
- [14] **Zimmer, B., Lee, Y., Puggelli, A., Kwak, J., Jevtic, R., Keller, B., Bailey, S., Blagojevic, M., Chiu, P., Le, H., Chen, P., Sutardja, N., Avizienis, R., Waterman, A., Richards, B., Flatresse, P., Alon, E., Asanovic, K. and Nikolic, B.** (2015). A RISC-V vector processor with tightly-integrated switched-capacitor DC-DC converters in 28nm FDSOI, *Symposium on VLSI Circuits (VLSI Circuits)*, Kyoto, Japan, pp.C316–C317, doi: 10.1109/VLSIC.2015.7231305.
- [15] **Choudhury, S.R., Thiruvathodi, S., Seetharaman, V., Cockrell, M., Michelson, J. and Redgrave, J.**, (2017), Verifying PULPino RISC-V Core for a Google Accelerator with STING.
- [16] **Kocher, P., Jaffe, J. and Jun, B.** (1999). *Differential Power Analysis*, Springer.
- [17] **Ors, S.B., Gurkaynak, F., Oswald, E. and Preneel, B.** (2004). Power-analysis attack on an ASIC AES implementation, *International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.*, volume 2, pp.546–552.
- [18] **Mangard, S., Oswald, E. and Popp, T.** (2007). *Power Analysis Attacks: Revealing the Secrets of Smart Cards*, Advances in Information Security, Springer.
- [19] **Höller, J., Tsiatsis, V., Mulligan, C., Karnouskos, S., Avesand, S. and Boyle, D.**, (2014). IoT Reference Architecture, pp.199–223.
- [20] **Rossi, D., Conti, F., Marongiu, A., Pullini, A., Loi, I., Gautschi, M., Tagliavini, G., Capotondi, A., Flatresse, P. and Benini, L.** (2015). PULP: A parallel ultra low power platform for next generation IoT applications, *2015 IEEE Hot Chips 27 Symposium (HCS)*, pp.1–39.
- [21] **Traber, A., Gautschi, M. and Schiavone, S.D.**, (2017). The RISC-V User Manual, Micrel Lab and Multitherman Lab, University of Bologna, Italy & Integrated Systems Lab, ETHZ Zurich, Switzerland.
- [22] **ZeroRisc**, <https://github.com/pulp-platform/zero-riscy>, online accessed: 20.04.2019.
- [23] **Russell, J. and Cohn, R.** (2012). *Serial Peripheral Interface Bus*, Bookvika.
- [24] **Raffo, D.**, (2018). Linux Quick Reference Guide, 6th edition.
- [25] **PULPino**, <https://github.com/pulp-platform/pulpino>, online; accessed 06.02.2019.

- [26] **Mentor Graphics**, ModelSim, https://www.mentor.com/company/higher_ed/modelsim-student-edition, online; accessed 06.02.2019.
- [27] **Veripool**, Verilator, <https://www.veripool.org/projects/verilator/wiki>, online; accessed 06.02.2019.
- [28] **Xilinx**, Vivado, <https://www.xilinx.com/products/design-tools/vivado.html>, online; accessed 06.02.2019.
- [29] **AVNET**, (2017), ZedBoard Getting Started Guide, <http://zedboard.org/sites/default/files/documentations/GS-AES-Z7EV-7Z020-G-V7-1.pdf>, Online; accessed 29.03.2019.
- [30] **Linux Mint**, https://www.mathworks.com/products/matlab.html?s_tid=hp_products_matlab, online; accessed 22.02.2019.
- [31] **CMake**, <https://cmake.org/>, online accessed: 16.04.2019.
- [32] **RI5CY Toolchain**, https://github.com/pulp-platform/ri5cy_gnu_toolchain, online; accessed 06.02.2019.
- [33] **Python2.7**, <https://www.python.org/download/releases/2.7/>, online accessed: 20.04.2019.
- [34] **Accellera**, (2004). SystemVerilog 3.1a Language Reference Manual.
- [35] **Std-1364-2001** (2001). IEEE Standard Verilog Hardware Description Language, *IEEE*.
- [36] **VanDyke Software Inc.**, (2008). An Overview of the Secure Shell (SSH).
- [37] **Xilinx**, (2015). AC701 Evaluation Board for the Artix-7 FPGA.
- [38] **WCET**, <http://www.mrtc.mdh.se/projects/wcet/benchmark.html>, online; accessed 06.02.2019.
- [39] **Chadha, R. and Bhasker, J.** (2013). *An ASIC Low Power Primer: Analysis, Techniques and Specification*, Springer.
- [40] **Sarwar, S.M. and Koretsky, R.M.** (2017). *UNIX: The Textbook, Third Edition*, Chapman and Hall.
- [41] **Matloff, N.** (2001). Cyclic Redundancy Checking.
- [42] **Lami, H.** (1979). *Analog and Digital Filters - Design and Realization*, Prentice Hall.
- [43] **MathWorks**, MATLAB, online; accessed 28-February-2019.
- [44] **Information Technology Laboratory (National Institute of Standards and Technology)** (2001). *Announcing the Advanced Encryption Standard (AES)*.
- [45] **Daor, J., Daemen, J. and Rijmen, V.** (1999). AES Proposal: Rijndael.

- [46] **Yao, Y., Yang, M., Patrick, C., Yuce, B. and Schaumont, P.** (2018). Fault-assisted side-channel analysis of masked implementations, *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp.57–64.



APPENDICES

APPENDIX A.1 : Post Implementation Run Script

APPENDIX A.2 : Switch Activity Parser and Counter

APPENDIX A.3 : AES Algorithm Used

APPENDIX A.4 : Switch Matrix Generation Script

APPENDIX A.5 : Script to Detect Message Separations



APPENDIX A.1

```
1
2 # Compile PULPino Post-Implementation Netlist
3 vlog pulpino_impl.v
4
5 # Launch simulator for testbench "tb", using simprims library
6 # and wok libraries, use c compiler at /usr/bin/gcc
7 vsim tb -L simprims_ver work.glbl -dpicppath /usr/bin/gcc
8
9 # Record signals under core region, including internals
10 add list -r {sim:/tb/top_i/pulpino_i/core_region_i/*}
11
12 # Run simulation until break statement
13 run -all
14
```

APPENDIX A.2

```
1
2 MULTIPLE_FILE = 0; % Read from individual dump files
3 ONE_FILE = 1; % Read from one collective dump file
4 max_msg = 1; % Used in multiple file mode
5 start_row = 6; % The row in lst files which timing information begins
6 total_msgs = 300;
7
8 %% Using multiple message files
9 if(MULTIPLE_FILE)
10     % Path to folder containing list files
11     LST_FOLDER = 'C:\Users\adm\Dropbox\Firat\matlab\lists';
12 end
13
14 %% Using just one file to read
15 if(ONE_FILE)
16     path = 'C:\Users\adm\Dropbox\Firat\matlab\1000msgs.lst'; %
17     max_msg = 1;
18 end
19
20 %% LIST PARSING
21 for msg_no = 1:1:max_msg
22
23     if(MULTIPLE_FILE)
24         path = [ LST_FOLDER '\list' int2str(msg_no) '.lst'];
25     end
26
27     f = fopen(path);
28     g = textscan(f,'%s','delimiter','\n');
29     fclose(f);
30     for j= start_row:1:size(g{1,1},1)
31         index = 1;
32         for k = 1:1:100
33             if(~isequal(g{1,1}{j,1}(k),' '))
34                 time_temp(j-(start_row-1),index) = g{1,1}{j,1}(k);
35                 index = index+1;
36             else
37                 if(g{1,1}{j,1}(k+1) == '+')
38                     delta(j-(start_row-1),1:2) = g{1,1}{j,1}(k+2:k+3);
39                 else
40                     delta(j-(start_row-1),1:2) = ['0' g{1,1}{j,1}(k+3)];
41                 end
42             end
43         end
44     end
45
46     for i=1:1:size(time_temp,1)
47         while (isempty( str2num( time_temp(i,size(time_temp,2)) )))
48             if( isempty(str2num(time_temp(i,end-1)) ))
49                 time_temp(i,1:size(time_temp,2)) = ...
50                     [ '00' time_temp(i,1:size(time_temp,2)-2) ];
51             end
52             if( isempty(str2num(time_temp(i,end)) ))
53                 time_temp(i,1:size(time_temp,2)) = ...
54                     [ '0' time_temp(i,1:size(time_temp,2)-1) ];
55             end
56         end
57     end
58
59     [delta_ivrs,clocks]=...
60     hist(str2num(time_temp),unique(str2num(time_temp)));
61     num_delta = str2num(delta);
62     time = str2num(time_temp) + num_delta;
63
64     data_start_col=1;
```

```
65 while(~isequal(g{1,1}{start_row,1}(data_start_col),'S'))
66     data_start_col=data_start_col+1;
67 end
68 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
69
70 for k = start_row:1:size(g{1,1},1)-1
71     if(g{1,1}{k,1}(data_start_col) ~= 'S') ...
72         data_start_col = data_start_col+1; end
73     if(g{1,1}{k+1,1}(data_start_col) == ' ') ...
74         X=data_start_col+1; else X=data_start_col; end
75     diffs(k-start_row+1) = ...
76         sum(g{1,1}{k,1}(data_start_col:end) ~= g{1,1}{k+1,1}(X:end));
77 end
78 diffs = [0 diffs];
79
80 end
81
```

APPENDIX A.3

```

1  #include "common.h"
2  #include "gpio.h"
3  #include <stdio.h>
4
5  #define AES_MAXROUNDS      14
6  #define AES_BLOCKSIZE     16
7  #define AES_IV_SIZE       16
8
9  #if __BYTE_ORDER == __BIG_ENDIAN
10 # define ntohl(x)         (x)
11 # define ntohs(x)        (x)
12 # define htonl(x)        (x)
13 # define htons(x)        (x)
14 #elif __BYTE_ORDER == __LITTLE_ENDIAN
15 # define ntohl(x)        __bswap_32 (x)
16 # define ntohs(x)        __bswap_16 (x)
17 # define htonl(x)        __bswap_32 (x)
18 # define htons(x)        __bswap_16 (x)
19 #endif
20
21 typedef struct aes_key_st
22 {
23     uint16_t rounds;
24     uint16_t key_size;
25     uint32_t ks[(AES_MAXROUNDS+1)*8];
26     uint8_t iv[AES_IV_SIZE];
27 } AES_CTX;
28
29 typedef enum
30 {
31     AES_MODE_128,
32     AES_MODE_256
33 } AES_MODE;
34
35 #define rot1(x) (((x) << 24) | ((x) >> 8))
36 #define rot2(x) (((x) << 16) | ((x) >> 16))
37 #define rot3(x) (((x) << 8) | ((x) >> 24))
38 #define mt  0x80808080
39 #define ml  0x7f7f7f7f
40 #define mh  0xfefefefe
41 #define mm  0x1b1b1b1b
42 #define mul2(x,t)  (((t)=((x)&mt), \
43                    (((x)+(x)&mh)^(((t)-((t)>>7))&mm)))
44
45 #define inv_mix_col(x,f2,f4,f8,f9) (\
46     (f2)=mul2(x,f2), \
47     (f4)=mul2(f2,f4), \
48     (f8)=mul2(f4,f8), \
49     (f9)=(x)^(f8), \
50     (f0)=((f2)^(f4)^(f0)), \
51     (f2)^(f9), \
52     (f4)^(f9), \
53     (f8)^(rot3(f2)), \
54     (f8)^(rot2(f4)), \
55     (f8)^(rot1(f9))
56
57 /*
58  * AES S box
59  */
60 static const uint8_t aes_sbox[256] =
61 {
62     0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,
63     0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76,
64     0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,
65     0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0,
66     0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,
67     0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15,

```

```

68     0x04,0xC7,0x23,0xC3,0x18,0x96,0x05,0x9A,
69     0x07,0x12,0x80,0xE2,0xEB,0x27,0xB2,0x75,
70     0x09,0x83,0x2C,0x1A,0x1B,0x6E,0x5A,0xA0,
71     0x52,0x3B,0xD6,0xB3,0x29,0xB3,0x2F,0x84,
72     0x53,0xD1,0x00,0xED,0x20,0xFC,0xB1,0x5B,
73     0x6A,0xCB,0xBE,0x39,0x4A,0x4C,0x58,0xCF,
74     0xD0,0xEF,0xAA,0xFB,0x43,0x4D,0x33,0x85,
75     0x45,0xF9,0x02,0x7F,0x50,0x3C,0x9F,0xA8,
76     0x51,0xA3,0x40,0x8F,0x92,0x9D,0x38,0xF5,
77     0xBC,0xB6,0xDA,0x21,0x10,0xFF,0xF3,0xD2,
78     0xCD,0x0C,0x13,0xEC,0x5F,0x97,0x44,0x17,
79     0xC4,0xA7,0x7E,0x3D,0x64,0x5D,0x19,0x73,
80     0x60,0x81,0x4F,0xDC,0x22,0x2A,0x90,0x88,
81     0x46,0xEE,0xB8,0x14,0xDE,0x5E,0x0B,0xDB,
82     0xE0,0x32,0x3A,0x0A,0x49,0x06,0x24,0x5C,
83     0xC2,0xD3,0xAC,0x62,0x91,0x95,0xE4,0x79,
84     0xE7,0xC8,0x37,0x6D,0x8D,0xD5,0x4E,0xA9,
85     0x6C,0x56,0xF4,0xEA,0x65,0x7A,0xAE,0x08,
86     0xBA,0x78,0x25,0x2E,0x1C,0xA6,0xB4,0xC6,
87     0xE8,0xDD,0x74,0x1F,0x4B,0xBD,0x8B,0x8A,
88     0x70,0x3E,0xB5,0x66,0x48,0x03,0xF6,0x0E,
89     0x61,0x35,0x57,0xB9,0x86,0xC1,0x1D,0x9E,
90     0xE1,0xF8,0x98,0x11,0x69,0xD9,0x8E,0x94,
91     0x9B,0x1E,0x87,0xB9,0xCE,0x55,0x28,0xDF,
92     0x8C,0xA1,0x89,0x0D,0xBF,0xE6,0x42,0x68,
93     0x41,0x99,0x2D,0x0F,0xB0,0x54,0xBB,0x16,
94 };
95
96 static const unsigned char Rcon[30]=
97 {
98     0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,
99     0x1b,0x36,0x6c,0xd8,0xab,0x4d,0x9a,0x2f,
100    0x5e,0xbc,0x63,0xc6,0x97,0x35,0x6a,0xd4,
101    0xb3,0x7d,0xfa,0xef,0xc5,0x91,
102 };
103
104 void AES_encrypt_main(AES_CTX *ctx, const uint8_t
105 *msg, uint8_t *out, int length)
106 {
107     int i;
108     uint32_t tin[4], tout[4], iv[4];
109
110     uint32_t msg_32[4];
111     uint32_t out_32[4];
112     memcpy2(msg_32, msg, AES_BLOCKSIZE,0);
113     msg += AES_BLOCKSIZE;
114
115     for (i = 0; i < 4; i++)
116         tin[i] = ntohl(msg_32[i]);
117
118     AES_encrypt(ctx, tin);
119
120     for (i = 0; i < 4; i++)
121     {
122         tout[i] = tin[i];
123         out_32[i] = htonl(tout[i]);
124     }
125
126     memcpy2(out, out_32, AES_BLOCKSIZE,0);
127     out += AES_BLOCKSIZE;
128
129 }
130
131

```

```

132 static void AES_encrypt(const AES_CTX *ctx,
133     uint32_t *data)
134 {
135
136     uint32_t tmp[4];
137     uint32_t tmp1, old_a0, a0, a1, a2, a3, row;
138     int curr_rnd;
139     int rounds = ctx->rounds;
140
141     const uint32_t *k = ctx->ks;
142
143     /* Pre-round key addition */
144     set_gpio_pin_value(30, 1); // setting gpio_out[30] to enable
145                                 // core dumping in testbench
146     data[0] ^= *(k++);
147     set_gpio_pin_value(30, 0); // resetting gpio_out[30] to pause
148                                 // core dumping in testbench
149     for (row = 1; row < 4; row++)
150         data[row] ^= *(k++);
151
152
153     /* Encrypt one block. */
154     //for (curr_rnd = 0; curr_rnd < rounds; curr_rnd++)
155     {
156         /* Perform ByteSub and ShiftRow operations together */
157         for (row = 0; row < 4; row++)
158         {
159
160             a0 = (uint32_t) aes_sbox[(data[row%4]>>24)&0xFF];
161             a1 = (uint32_t) aes_sbox[(data[(row+1)%4]>>16)&0xFF];
162             a2 = (uint32_t) aes_sbox[(data[(row+2)%4]>>8)&0xFF];
163             a3 = (uint32_t) aes_sbox[(data[(row+3)%4]&0xFF)];
164             /* Perform MixColumn if not last round */
165             if (curr_rnd < (rounds - 1))
166             {
167                 tmp1 = a0 ^ a1 ^ a2 ^ a3;
168                 old_a0 = a0;
169                 a0 ^= tmp1 ^ AES_xtime(a0 ^ a1);
170                 a1 ^= tmp1 ^ AES_xtime(a1 ^ a2);
171                 a2 ^= tmp1 ^ AES_xtime(a2 ^ a3);
172                 a3 ^= tmp1 ^ AES_xtime(a3 ^ old_a0);
173             }
174
175             tmp[row] = ((a0 << 24) | (a1 << 16) | (a2 << 8) | a3);
176         }
177
178         for (row = 0; row < 4; row++)
179             data[row] = tmp[row] ^ *(k++);
180     }
181 }
182

```

APPENDIX A.4

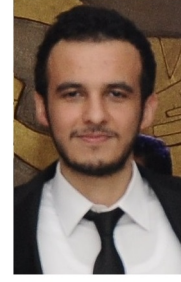
```
1
2 r_diffs = [];
3 r_times = [];
4 x=0;
5 for k = 1:length(clocks)
6     r_times = [ r_times clocks(k):1:clocks(k)+30 ];
7     temp = zeros(1,31);
8     for j = 1:1:delta_ivrs(k)
9         temp(num_delta(x+j)+1) = diffs(x+j);
10    end
11    x = x+delta_ivrs(k);
12    r_diffs = [ r_diffs temp ];
13 end
14 r_diffs = transpose(r_diffs);
15 r_times = transpose(r_times);
16
17 i=1;k=1;
18 while(1)
19     msgs{k} = r_diffs(i:k*length(r_diffs)/total_msgs);
20     i=(k*length(r_diffs)/total_msgs)+1;
21     if(i>length(r_diffs))
22         break;
23     end
24     k=k+1;
25 end
26
27 k=1;
28 while(1)
29     i=1;j=1;
30     while(1)
31         if(i+61<length(msgs{k}))
32             clks_avg(k,j) = sum(msgs{k}(i:i+61))/62;
33         else
34             break;
35         end
36         j=j+1;
37         i=i+62;
38     end
39     k=k+1;
40     if(k>size(msgs,2))
41         break;
42     end
43 end
44
45 for i=1:size(clks_avg,1)
46     avg(i,:) = sum(clks_avg(i,:))/size(clks_avg,2);
47 end
48
```


APPENDIX A.5

```
1 %% DETECT MESSAGE SEPERATION POINTS && Average switching per clock cycle
2
3 i=1;j=0;i_init=i;
4 while(i<size(r_times,1))
5     i=i+1;
6     i_init=i;
7     while(r_times(i)-r_times(i-1)<100000 && i<=size(r_times,1) )
8         i=i+1;
9         if( i>size(r_times,1) ) break; end
10    end
11    j=j+1;
12    msgs{j} = r_times(i_init-1:i-1);
13    if( i>=size(r_times,1) ) break; end
14 end
15
16 i=1;k=1;
17 while(1)
18     msgs{k} = r_diffs(i:k*length(r_diffs)/total_msgs);
19     i=(k*length(r_diffs)/total_msgs)+1;
20     if(i>length(r_diffs))
21         break;
22     end
23     k=k+1;
24 end
```



CURRICULUM VITAE



Name Surname: Yasin Fırat Kula

Place and Date of Birth: Çorlu/Tekirdağ - 28.10.1993

E-Mail: kulay@itu.edu.tr

EDUCATION:

- **B.Sc.:** 2016, Istanbul Technical University, Electrical and Electronics Faculty, Electronics and Communications Engineering Department
- **M.Sc.:** 2019, Istanbul Technical University, Graduate School of Science Engineering and Technology , Electronics Department

PROFESSIONAL EXPERIENCE AND REWARDS:

- Working as Research Assistant in Istanbul Technical University Electronics and Communications Department since 2017

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- Kula F., Ors B., 2019. Average Power Consumption Estimation and Momentary Power Consumption Profile Generation of a Softcore Processor. *Seventh International Conference on Digital Information Processing and Communications (ICDIPC)*, p. 41-46. May 2-4, 2019 Trabzon, Turkey.

OTHER PUBLICATIONS, PRESENTATIONS AND PATENTS:

- Ayhan T., Kula F., Altun M., 2017. A Power Efficient System Design Methodology Employing Approximate Arithmetic Units. *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2017 Bochum, Germany.
- Kula F., Ayhan T., Altun M., 2018. FPGA Üzerinde Yaklaşık FIR Süzgeç Tasarımı. *Sinyal İşleme Uygulamaları Kurultayı (SIU)*, 2018 İzmir, Turkey.