
Faculty of Postgraduate Studies and Scientific Research

German University in Cairo

# Design and Implementation of Lightweight Hardware Security Platform For IoT Applications

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science in Electronics Engineering

By

**Mohammed Morhaf Abdulkarim Jaela**

Supervised by

**Mohamed Abd El-Ghany**

Assoc. Prof. Dr.

Electronics Department

German University in Cairo

**Amr T. Abdel-Hamid**

Assoc. Prof. Dr.

Networks Department

German University in Cairo

2018

# Examination Committee

**Supervisors:**

Name: Mohamed Abd El-Ghany

Position Title: Assistant Professor

Faculty: Information Engineering and Technology

University: German University in Cairo, Egypt

Name: Amr Talaat Abdel-Hamid

Position Title: Assistant Professor

Faculty: Information Engineering and Technology

University: German University in Cairo, Egypt

**National Examiners:**

Name: Hassan Mostafa

Position Title: Assistant Professor

Faculty: Electronics and Communications Engineering

University: Cairo University, Egypt

**International Examiners:**

Name: Diana Goehringer

Position Title: Professor

Faculty: Computer Science

University: Technical University of Dresden, Germany

# Declaration

I, ***Mohammed Morhaf Abdulkarim Jaela*** declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.

**Thesis Title:** Design and Implementation of Lightweight Hardware Security Platform For IoT Applications.

**Thesis type:** M.Sc.

I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at the German University in Cairo;
- Where anywhere I have consulted the published work of others, this is always clearly attributed;
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- Either none of this work has been published before submission, or parts of this work have been published as: (please list references below)

- Books, journals and other teaching materials made available to me by the German University in Cairo are for my own studies, and copying or using them for other purposes is an infringement of copyright;

Signature: Mohammed Morhaf Abdulkarim Jaela

# Acknowledgments

*"And say, "O My Lord! increase me in knowledge" "* – [Ta Ha: 114]

First and foremost, I am thankful to Almighty Allah for bestowing me health, persistence, and knowledge to complete this work. I implore Him to make my knowledge and skills useful to mankind.

I would like to thank the German Federal Foreign Office (AA), the German Academic Exchange Service (DAAD), Ulm University, and German University in Cairo(GUC), for giving me the opportunity to hold a scholarship to study a two-year Master Program in the Electronics Engineering.

I cannot thank my family enough for the unending affection, encouragement, respect and all the exciting and gloomy things I have shared with them. I express my deepest gratitude to my parents, brothers, sisters, and my fiancée for their emotional and moral support throughout my academic career and also for their tolerance, inspiration, and prayers.

I am deeply indebted and grateful to my supervisors Assoc. Prof. Dr. Mohamed Abd El-Ghany and Assoc. Prof. Dr. Amr T. Abdel-Hamid for providing me the much needed motivation and guidance in achieving this milestone.

I am very grateful to Assoc. Prof. Dr. Hassan Mostafa, head of the Opto-Nano-Electronics (ONE) lab, Cairo University, for helping me in all academic and non-academic matters.

Special thanks to my colleagues from the ONE lab: Khaled Essam, Shady Soliman, Ahmed Kamal, who I worked with them on many issues.

I would like to express my gratitude to all my Egyptian and Syrian friends who I met in Egypt, especially Haitham Senior, Zaher Rahhal, Mostafa El-Soda, Ahmed Alloush, Mohamed Belal, Ahmed Badr.

# Abstract

Internet of Things (IoT) is a promising technology that is continuously spreading around the world leading to many challenges facing cryptographic designers who are trying to fulfill the security standards of IoT constrained devices. In this thesis, a new design is proposed that adds a new dimension of security by using the concept of frequency hopping to generate a pseudo-random pattern for switching between 5 lightweight cryptographic ciphers: AEGIS, ASCON, COLM, Deoxys and OCB that are participating in the Competition for Authenticated Encryption, Security, Applicability, and Robustness (CAESAR). The proposed design exploits the advantages of Dynamic Partial Reconfiguration (DPR) technology in Field Programmable Gate Arrays (FPGAs) to switch between the 5 ciphers using Internal Configuration Access Port controller (AXI-HWICAP) providing a decrease of 58% and 80% in area utilization and power consumption respectively . The design is synthesized using Xilinx Vivado 2015.2 and mounted on Zynq evaluation board (XC7Z020LG484-1).

# Contents

# List of Tables

# List of Figures

# Acronyms

**AD** Associated Data. 34

**AE** Authenticated Encryption. 27–29

**AEAD** Authenticated Encryption with Associated Data. vi, 2, 28, 29, 31, 33–35, 49, 58, 59, 68

**AP SoC** All Programmable System on a Chip. 12

**API** Application Program Interface. 34

**APU** Application Processing Unit. 17, 22

**AXI** Advanced eXtensible Interface. vi, 14, 21, 22, 48, 58, 59, 64

**CAESAR** Competition for Authenticated Encryption: Security, Applicability, and Robustness. 31, 34, 58, 59, 68

**CLB** Configurable Logic Blocks. 19, 20

**DCP** Design Check Point. vii, 59

**DO** Data Outputs. 69

**DPR** Dynamic Partial Reconfiguration. 55, 59, 70, 73, 74

**FIFO** First-In, First-Out. 34, 35

**FPGA** Field Programmable Gate Array. 59–62, 64, 65, 68, 73

**HWICAP** Hardware ICAP. 47, 48

**ICAP** Internal Configuration Access Port. 46–48

**IEEE** Institute of Electrical and Electronics Engineers. 4

**IoT** Internet of Things. vi, 1–11, 23, 24, 48

**IP** Intellectual Property. 47

**ITU** International Telecommunicaation Union. vi, 5, 6

**LFSR** Linear Feedback Shift Register. 49, 51, 52, 54, 55, 58, 65

**MAC** Message Authentication Code. vi, 25, 26, 29, 30

**PDI** Puplic Data Inputs. 51, 68–70

**PL** Programming Logic. vi, 17, 19–22

**PS** Processing System. vi, 21, 22

**RFID** Radio Frequency Identification. 1

**RM** Reconfigurable Module. vii, 44, 59, 60, 62, 64, 67

**RP** Reconfigurable Partition. 44

**SDI** Secret Data Inputs. 51, 69, 70

**SDK** Software Development Kit. 69, 70

**SoC** System on Chip. 13–15

**ZC702** Xilinx evaluation and development board. 12, 13, 70

# Chapter 1

# Introduction

## 1.1 Motivation

Internet of Things (IoT) is a network of devices connected to each other in a wired or non-wired way where each device has a unique identity. These devices process data and send it to each other without human intervention [17] .The term IoT usually refers to resource-limited objects such as sensors, Radio Frequency Identification (RFID) tags or any other contactable device that has the ability to compute data while connected to the Internet [18]. Leading industry forecasts of IoT growth demonstrate a consensus that the number of connected devices is poised to grow rapidly, with a doubling or more between 2016 and 2020 as shown in figure 1.1

However, concerns about privacy and security are increasing, especially given that IoT takes considerable place in the contexts of governments and organizations, as well as infrastructure of public institutions. To address this issue, lightweight cryptography is becoming a considerable approach to make the connection and transfer of data between constrained devices more secure [19].

Lightweight cryptography is a cryptographic algorithm or protocol tailored for implementation in constrained environments such as RFID tags, sensors, contactless smart cards, and health-care devices. One of the most effective ways of exploiting lightweight cryptographic algorithms is by using authenticated encryption schemes. Authenticated encryption schemes are a class of symmetric key cryptographic algorithms that ensure that both of confidentiality and authenticity

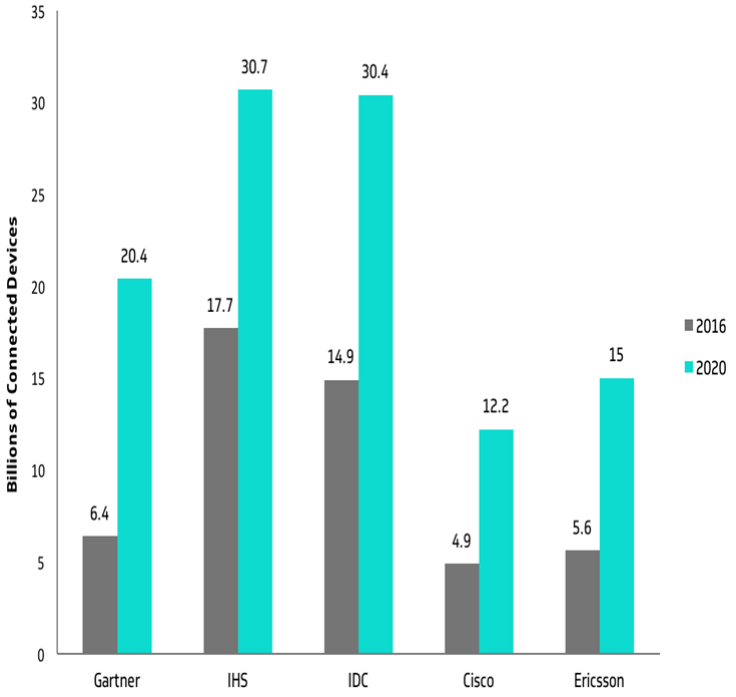



Figure 1.1: Leading Industry Forecasts Anticipate Significant IoT Growth [4].

of data are provided at the same time [20]. Confidentiality includes protecting data from being exposed without permission, while authenticity comprises ensuring both integrity of data and verification of its source. Sometimes data such as packet headers are required for handling by some applications, which requests authentication without the need for encryption. Such schemes are usually defined as Authenticated Encryption with Associated Data (AEAD) which is embraced in this article.

## 1.2 Thesis Aim

AEAD scheme provides one level of data security which is the symmetric key. However, nothing beyond this level has ever been added to tighten the security of data against attacks due to limited resources used by IoT devices. In this thesis, a new design is proposed to introduce a second dimension of security using the concept of frequency hopping by switching between 5 lightweight cryptographic algorithms that are currently under review in the ongoing CAESAR competition. The switching is performed using dynamic partial reconfiguration (DPR) technology to maintain a reasonable power consumption and area utilization, taking into consideration the limited resources of IoT devices while adding the second

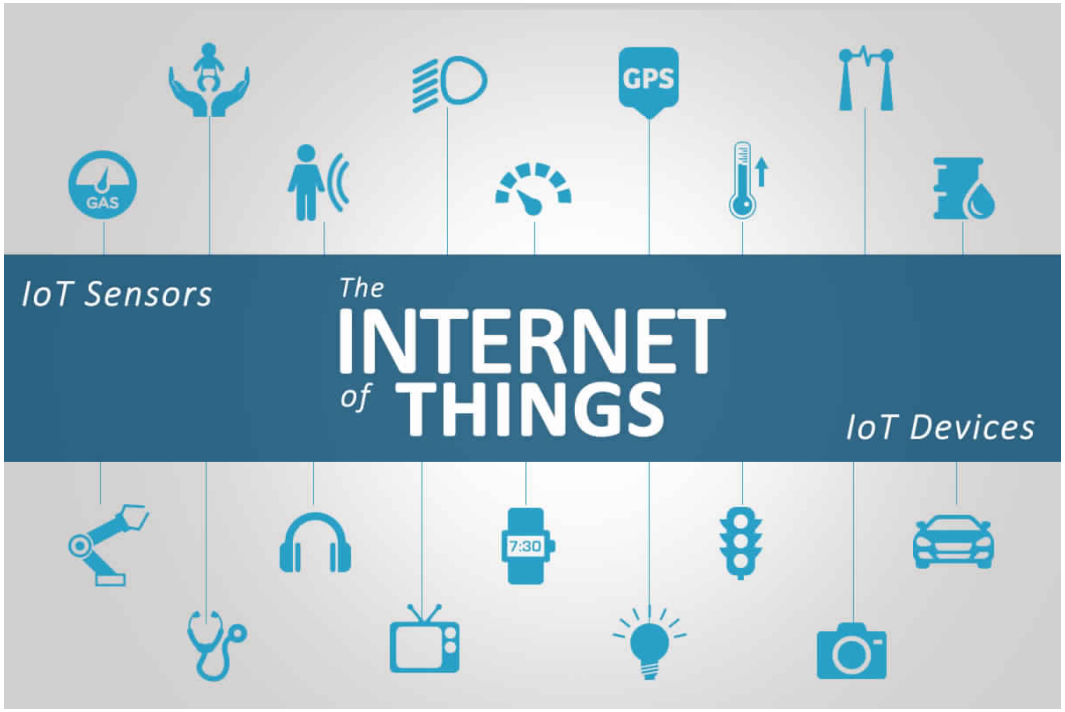



Figure 1.2: Internet of Things Devices and Sensors.

dimension of security based on the pseudo-random pattern for switching between different algorithms.

## 1.3 Thesis Organization

The rest of the thesis is organized as follows:

The second chapter discusses the fundamentals this project is based on which covers the board used and the important building blocks of the Zynq device. Also a general overview about the IoT in general as well as thes security challenges in IoT nodes and the application of IoT.

The third chapter introduces the goals of the cryptography. It also discuss the AEAD scheme and the types of the cryptography. Also a general overview about the CAESAR competition. In addition, some related work is discussed.

In Chapter 4, we propose a lightweight hardware security platform for IoT applications based on Algorithm hopping and DPR technique. The chapter starts with presenting of the algorithm hopping technique, then the DPR technology and the controller that is used. After that, the flow of the proposed design.

The fifth chapter discuss the requirements of this project, the design steps and the full implementation of the proposed design. It presents also the implementa-

tion results and how the system was being tested.

Finally, the whole work done in this project is concluded in the chapter 6 and after that some discussions about the results that I reached after finishing the project. In addition, suggests future work related to the project.

# Chapter 2

# Fundamentals

## 2.1 The Internet of Things: Context and Overview

The concept of the "Internet of Things" appeared for the first time by Kevin Ashton as the title of his presentation at Procter & Gamble (P&G) in 1999 [21].He described the IoT as: "The Internet Of Things IS About Empowering Computers, So They Can See, Hear, And Smell The World For Themselves".So far, there are many definitions of IoT. In this thesis, however, the definition of the IoT was considered from the IEEE and ITU.

### 2.1.1 IEEE definition

Institute of Electrical and Electronics Engineers (IEEE) is a global, professional engineering organization whose interest is to encourage technological modernization and greatness for the assistance of humanity. In its appropriate report on Internet of Things expressed in Mar. 2014 [22], IEEE express the concept of "Internet of Things" as: "A network of items - each embedded with sensors - which are connected to the Internet." [17]. This announcement is written as a depiction of the "Internet of Things," not as an official definition of the concept. But we can note that the description addresses just the physical aspect of IoT. One project that straight relates to IoT is IEEE P2413. IEEE P2413 is currently considering the architecture of IoT as three-tiered, with the layers explained in Fig. 2.1.

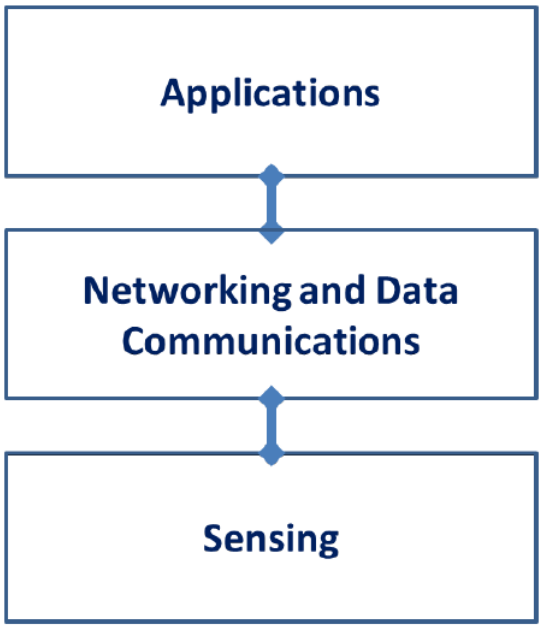



Figure 2.1: three-tier architecture of IoT

### 2.1.2 ITU definition

The ITU is the United Nations specialized agency for information and communication technologies (ICTs).In its 2005 IoT report, ITU terms the IoT as a network that is: "Available anywhere, anytime, by anything and anyone."

## 2.2 IoT Architecture

As shown in the simplified architecture in Fig. 2.3,the IoT is structured into three tiers of devices [5]. At the bottom, IoT nodes perform sensing and interact with the physical world. To assure scalability and ubiquitous network access, gateways and concentrators gather, protect (under users control) and route data from several and physically proximal IoT nodes, and route it to servers. The latter perform data aggregation and knowledge extraction, and deliver physically enhanced cloud services. Some additional intermediate levels of aggregation might be needed, according to the extent of data produced, the zone covered by a sub-network, and the density of IoT nodes, among the others.

An almost general architecture of IoT nodes is illustrate in Fig. 2.4, which specifics its main sub-systems, including processing and security assurance , power conversion and delivery, analog interfaces, radios, energy sources, system integration and

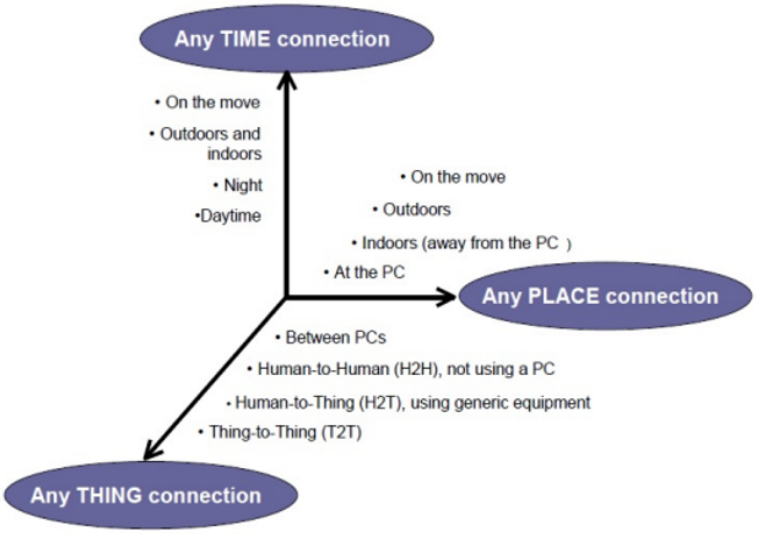



Figure 2.2: ITU definition of IoT

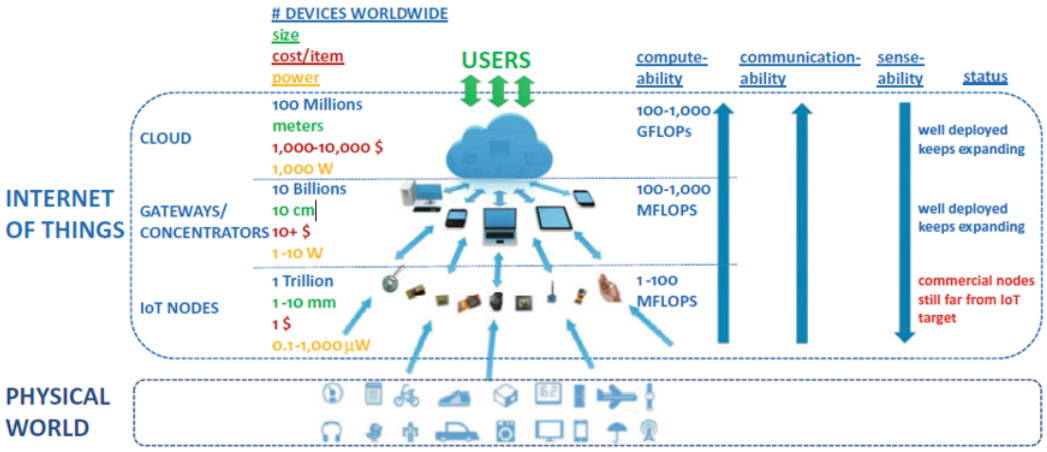



Figure 2.3: Simplified Architecture of IoT [5]

assembly [5]. As in Fig. 2.4, sensors are connected to analog interfaces that include an amplifier with programmable gain (and sometimes analog filters), and are multiplexed to share a single ADC for all analog channels. Analog voltages are generated by a DAC for actuation. Energy interfaces include an energy storage part (e.g., battery, super capacitor), and energy harvesters for energy refilling. Relatively general architecture of IoT nodes with detailed sub-systems

To adapt the strategy of power management to the real energy availability, there are circuits included to monitor the level of battery. In Fig. 2.4, memories are a significant element of IoT node processing. RAM is needed to store the data

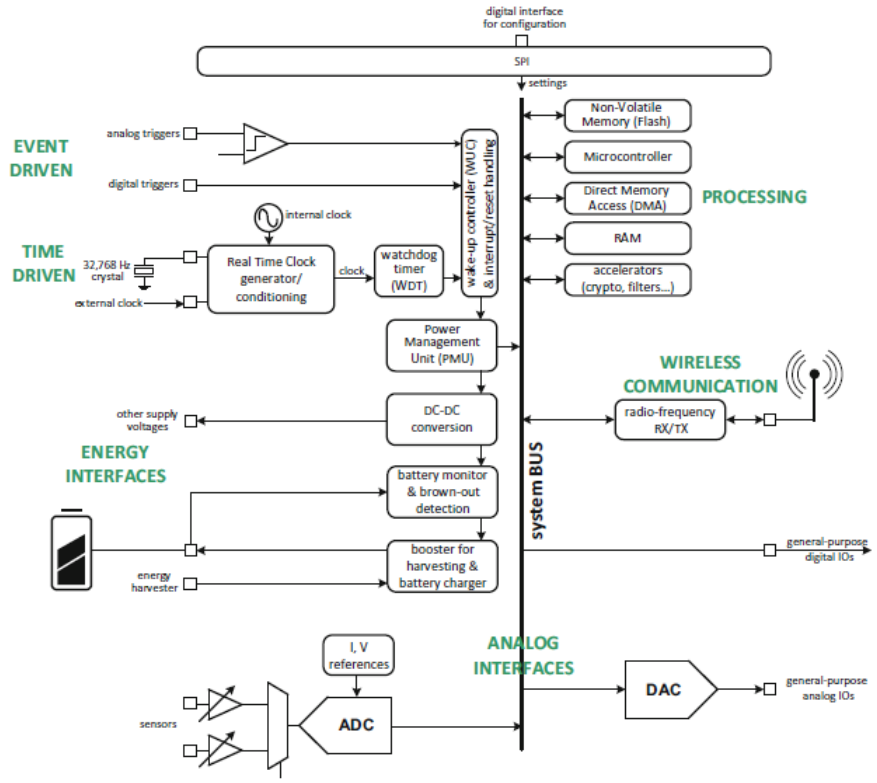



Figure 2.4: Relatively general architecture of IoT nodes with detailed sub-systems [5]

in sleep mode, as well as for the microcontroller/microprocessor execution. The Non-Volatile Memory (NVM) used to store the instructions, settings and also data that needs to be holden for a long period, thus reduce the power consumption associated with retention.

The function of sensors is to collect data from their surrounding environment. It can be classified in several ways [23], for example:

- **Location:** GPS, GLONASS, Galileo, Wi-Fi, Blue-tooth, Ultra-Wide-Band (UWB).
- **Biometric:** Fingerprint, Iris, Face.
- **Acoustic:** Microphone.
- **Environmental:** Temperature, Humidity, Pressure.
- **Motion:** Accelerometer, Gyroscope.

## 2.3 The applications of IoT

The IoT is a highly fragmented application scenario [24], and cover an extensive variety of applications , some of which are brief in the following:

### 2.3.1 Agriculture Sector

In the agriculture sector, the IoT infrastructure can monitor the quality, the actual usage and the availability of resources, for better and predictive management (e.g., irrigation) and storage (e.g., avoid waste of feed and fertilizing) [5]. Monitoring the environmental conditions permits to support the growth of animals and plants (e.g., aquaculture), optimally time the next course of action, and eventually ensure quality (e.g., wine) and boost the efficiency in the production process.

### 2.3.2 Automotive

In automotive, the IoT enables the monitoring of the state of a vehicle down to its critical components, from initial shipping to usage, to assess their correct utilization (e.g., detecting bumps, vibration) and repair (e.g., opening of containers, wearing parts) [5]. Depending on real usage, predictive repair can be executed to lengthen the vehicle lifetime, and lower the upkeep. Such capabilities enabled by the IoT are also very useful in fleet management and car sharing services. Also, distributed sensing and global sense making enables traffic control through distinguish and personalized way pricing to foster virtuous behavior and prioritize tasks for commercial (e.g., carpooling with various riders sharing cost) and personal vehicles (e.g., rapid transmission for critical goods), through virtual/dynamic city area borders.

### 2.3.3 Public Transportation

In public transportation, the occupancy and utilization can be monitored to assure an adequate quality of service, detect potential danger (e.g., potential collision between vehicles and pedestrians), and predict short term demand based on crowd monitoring in strategic locations [5]. On the road side, excessive congestion and pollution can be managed with real-time demand response schemes where the road pricing is dynamically adjusted through real-time observations and utilization

prediction, based on previous history and real-time data in strategic locations. Also, the transportation of serious merchandise and the circulation of slow (or frequently stationary) cars can be optimally coordinated with the normal traffic to reduce their negative effect.

### 2.3.4 Energy Management

Energy management at different scales can be made more effective by the IoT [24]. At the city scale, the smart grid offers sundry chances to leverage the sensing and sensemaking capabilities of the IoT to optimize the energy usage across many users, a better coordinated usage and planning of alternative energy sources, ultimately reducing the overall energy and the currently large gap between the peak and the average consumption.

### 2.3.5 Health Care

Health care is other critical application field in which the IoT pledges to mainly contribute to [24]. As few examples, the miniaturization and big lifetime of IoT nodes provides an unobtrusive mean to permanently monitor vital marks and other regarding parameters (e.g., behavioral) and improve deeper realization of the patients health evolution. In addition, the availability of big data from a big number of patients offers an unprecedented chance to explore linkages, build models and tools for predictive diagnosis, early medication and make drug discovery more efficient and effective. Similar considerations hold for the elderly and the handicapped as continual non-obtrusive observation allows for better and highly responsive/predictive care, while preserving individuals independency and offloading hospitals. Remote supervision also enhances the ability to share professionals across a larger number of individuals and patients, thus driving the care cost down.

### 2.3.6 Smart Homes

Through the IoT, smart homes can manage utilities more efficiently by controlling individual appliances based on actual utilization and needs, and purchasing electricity when expenditure is lowest during the day in demand-response energy pricing schemes [5]. Unprecedented levels of security (e.g., environment access control) are achievable thanks to the pervasiveness of IoT nodes and sense making

capability. resident recognition permits to adjust lighting, sound, air conditioning/ heating based on individual preferences. This can be done in a predictive way, so that inhabitants do not need to push any button, leveraging the fine-grain knowing, of inhabitants traditions and the ability of the cloud to generalize and extract trends and predictions.

### 2.3.7 Smart Buildings

Smart buildings can use the IoT to be more adaptive to the real requirement and needs of the inhabitants, while guaranteeing the most astounding security and comfort standards [5]. Indeed, air quality and thermal/acoustic/visual comfort can be monitored and controlled for the first time with a granularity that goes down to the single room, with clear features in terms of comfort assurance and energy expenditure. Beyond normal building operation, the real-time capability of the IoT enables the ability to respond to critical events (e.g., fire) quickly, reducing the human and physical losses in case of emergencies.

### 2.3.8 Smart Cities

Through the IoT, in the smart cities, the resources management can be more efficient, be made much more flexible to temporary malfunctions and calamity, and promote virtuous behavior [6]. intelligent and weather-altering lighting, water/gas seep monitoring, intelligent parking with dynamic pricing and zone distribution, no physical borders and automated parking tips are simply a few ideas of how to handling the IoT to solve todays urban challenges. Ubiquitous vision can cement an unmatched level of safety and protection, detecting probable risk and provide critical information on crowd behavior and citizens needs (e.g., for adaptive and predictive carriage administration,, real-time digital signboard guidance to deny direct risk). Similarly, distributed audition permits to improve situational consciousness, build real-time noise civil maps to relieve noise pollution at crucial times, and localize noise events for security assurance. Intelligent irrigation of green areas and gardens is another sub-area where the IoT has possibility to make an effect,. Smart tourism promises to give tourists the ability to have an immediate understanding of the city, such as availability, crowdedness or quietness of different places to receive dynamic recommendations on tours that adapt to their

disposition, other than already available factual information on places. Waste management can be made more efficient and priced fairly as discussed before, while detecting potentially dangerous and inappropriate waste that would need to be disposed with different procedure (again, encouraging virtuous behavior).
Fig. 2.5 illustrates what people search for on Google, what people broadcast about on Twitter, and what people write around on LinkedIn [6].

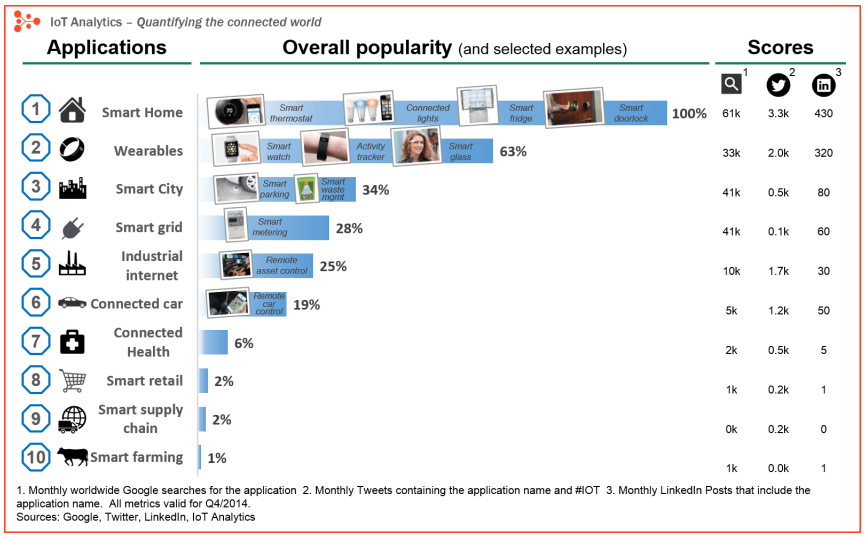



Figure 2.5: The Internet of Things application ranking [6]

## 2.4 Security Challenges in IoT Nodes

As shown in figure 2.6, global spending on IoT security is currently estimated at $703M for 2017 and forecasted to grow at a CAGR of 44% over the six-year period to become a $4.4B market opportunity by 2022 (Forecasts are based on the IoT security related revenue of leading technology companies in the field, across 12 industries and 21 technology areas).

However, Security represents a very broad challenge in the IoT, due to several factors [5] :

- Classic security algorithms such as publickey cryptography are not workable in most of IoT nodes, due to their stringent power and cost requirements.
- The vast number of IoT entities and the resulting scale of the IoT network make an unprecedented extremely large number of backdoors that can be exploited by enemies to perform physical and network attacks.

- The always-linked characteristic of IoT entities makes them rather vulnerable to eavesdropping and software attacks, data theft and device cloning [25]

- In our view, the personal information saved in the cloud will be likely shared across several service providers, which will deliver their service through cloud apps that will run on a data center-scale software platform. The entity will likely give access to the service providers as we do today when we install apps on a smartphone. This data sharing with various service providers creates new security challenges, which have to be solved mostly on the server side.

The above security challenges are clearly perceived to be critical by users, and more details about cryptography will be in Chapter 3.

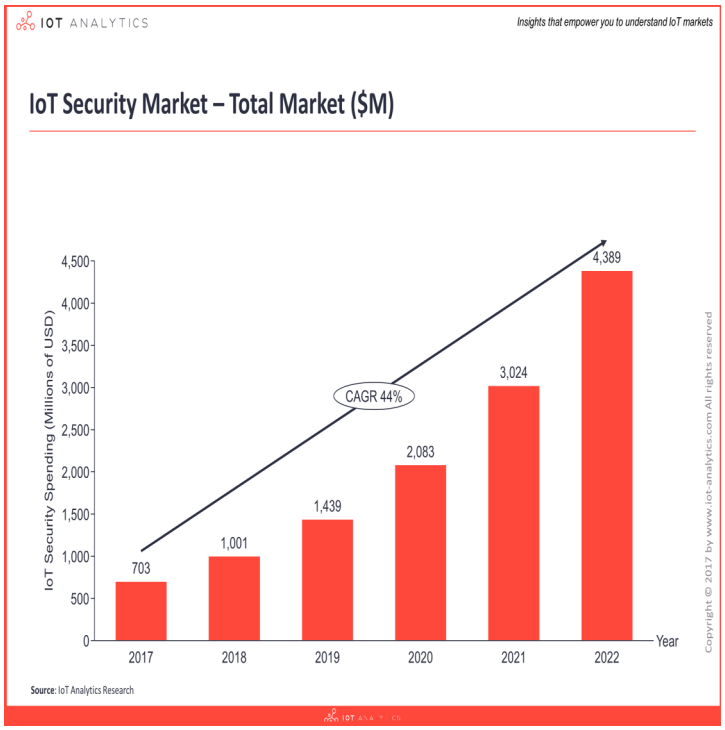



Figure 2.6: Global spending on IoT security [7]

## 2.5 Development Board

### 2.5.1 Overview

The board on which the proposed design was implemented is the Xilinx evaluation and development board (ZC702) board Fig. 2.7. The ZC702 is an evaluation

and development board depending on an XC7Z020-CLG484-1 Zynq-7000 All Programmable SoC (AP SoC) device. The Zynq-7000 All Programmable System on a Chip (AP SoC) made up of two parts, the Programming Logic (PL) which consists of 85,000 Series-7 cells, and the Processing System (PS) which is a dual Cortex-A9 Arm processor [8].


Figure 2.7: ZC702 Evaluation Kit [8].

### 2.5.2 ZC702 Board Features

The ZC702 board simplifies the design of embedded systems as it consist of numerous of the needed building elements of an embedded system [8]. Fig. 2.8 illustrates the important peripheral of the board. However, Some features of the board are listed here:

- Zynq XC7Z020-1CLG484C device.
- 1 GB DDR3 component memory (four 256 Mb x 8 devices).
- 128 Mb Quad SPI flash memory
- USB 2.0 ULPI (UTMI+ low pin interface) transceiver
- USB JTAG interface using a Digilent module.

- Clock sources
- Status LEDs
- User I/O

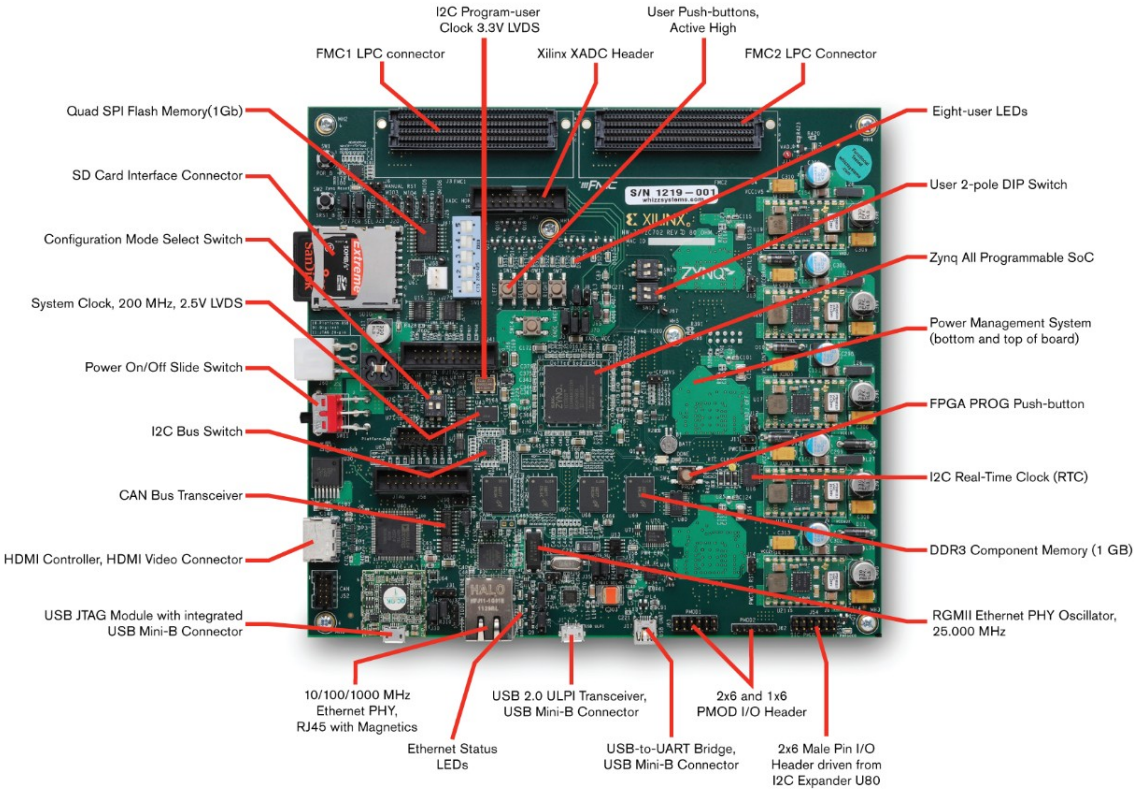



Figure 2.8: Feature Callout or the ZC702 Board [8]

## 2.6 System-on-Chip with Zynq

As is known, the term of "SoC" has existed for some time, and its implicit meaning is that one silicon chip can be used to perform all functions of the system, rather than using several different physical chips.In the past, the term "System on Chip (SoC)" usually refers to an Application Specific Integrated Circuit (ASIC), which can contain digital, analogue and radio frequency elements, together with mixed signal blocks for implementing analogue-to-digital and digital-to-analogue converters (ADCs and DACs).Let's focus a little on the digital side, the SoC can contain all aspects of the digital system: processing, high-speed logic, interfacing, memory, and so on. All these functions are usually achieved using separate devices

and then combined into a system at the Printed Circuit Board (PCB) level. The SoC solution is lower cost, enables faster and more secure data transfers between the various system elements, has higher overall system speed, lower power consumption, smaller physical size, and better reliability [9]. Fig. 2.9 illustrates a simple comparison of the system-on-a-board and the system-on-chip.

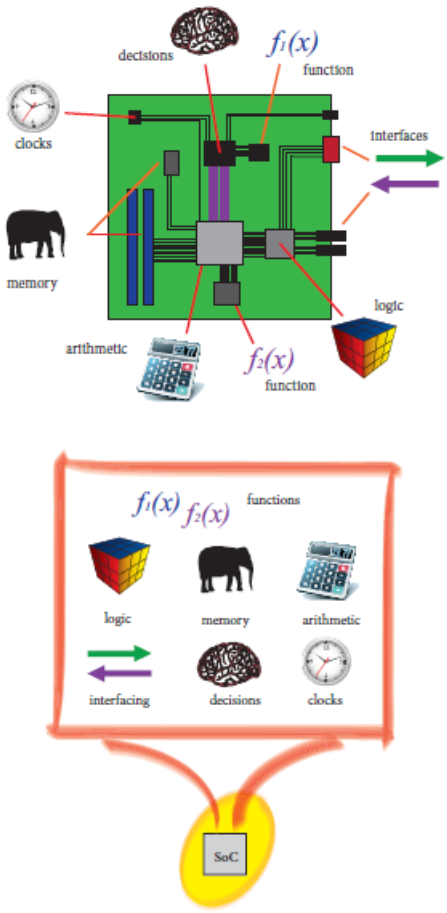



Figure 2.9: comparison of the system-on-a-board (top) and the system-on-chip (bottom) [9]

Because of the limited specification of the ASIC SoCs, it is not compatible with a large number of applications, especially when marketing, flexibility and upgrade-ability are required. Therefore, there was an urgent need for a more flexible solution, and therefore the SoC was encouraged to use FPGA.

Now, Zynq is the new generation of System-on-Chip (SoC) which was marketed by the Xilinx as an All-Programmable SoC (APSoC) device. Fig. 2.10 introduces a high level model of the architecture of Zynq. We also note that it consists of two parts, the first is a processing unit with fixed physical properties is not changeable

and is called a Processing System (PS) which consists of dual-core ARM Cortex-A9 processor, the second part is a Programmable Logic (PL) that is equivalent to an FPGA [9]. An industry standard Advanced Extensible Interface (AXI) is used to link between the two parts. Xilinx corporation produced many boards supporting the Zynq such as : ZYBO, MicroZed, ZedBoard, ZC702, ZC706, Zynq MMP, Zynq Mini-ITX. For more details visit Xilinx website https://www.xilinx.com/.

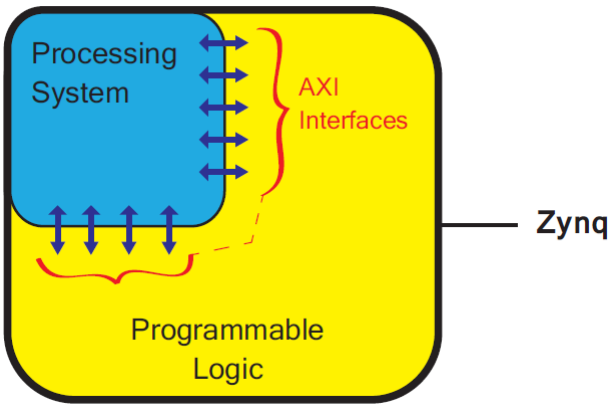



Figure 2.10: A simplified model of the Zynq architecture [9]

### 2.6.1 SoC Design Flow

Fig. 2.11 illustrates the simple design flow for SoC development. Normally, in any project, the first step is to determine the behavior of the system and the functions to be performed and this is the starting point of the design flow. And as we mentioned that the Zynq has two parts, ARM processor for software implementation and PL for hardware implementation, therefore, we partition the requirements to be implemented into two parts, a section to be executed using the PS and the other using the PL, where the best performance is obtained. After achieving the partition of the system, the process of developing hardware and software is done in parallel. Finally, the software and hardware parts are integrated and the system is fully tested [9].

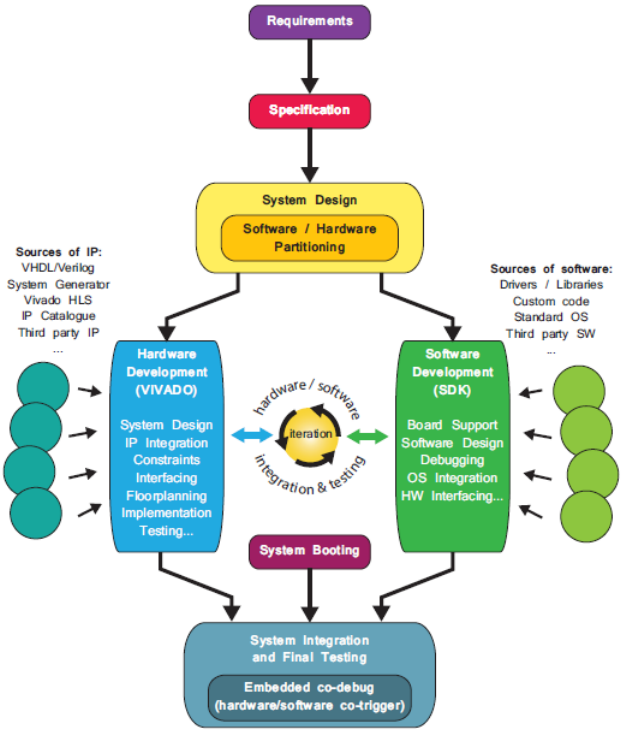



Figure 2.11: A basic model of the design flow for Zynq SoC [9]

### 2.6.2 Processing System

All Zynq devices have the same basic architecture, and all have a dual-core hard processor, which is positioned as a dedicated and optimized silicon component on the device.For comparison, the alternative to a hard processor is a soft processor like MicroBlaze from Xilinx , which is implemented by integrating elements of the PL [26]. In general, the advantage of soft processors is that the cloned number of it is controllable on the same PL. On the other hand, hard processors have higher performance as in Zynq. Fig. 2.12 shows the siting of the hard processor (ARM) and soft processor (MicroBlaze) on the Zynq device; the ARM as a dedicated resource, and the MicroBlaze located in the logic fabric.

Importantly, the Zynq processing system involve not solely the ARM processor, but a group of related processing resources forming an Application Processing Unit (APU), and additional peripheral interfaces, cache memory, memory interfaces, interconnect, and clock source circuitry [27].

Fig. 2.13 shows a block diagram of the architecture of the PS, where the Application Processing Unit (APU) is highlighted

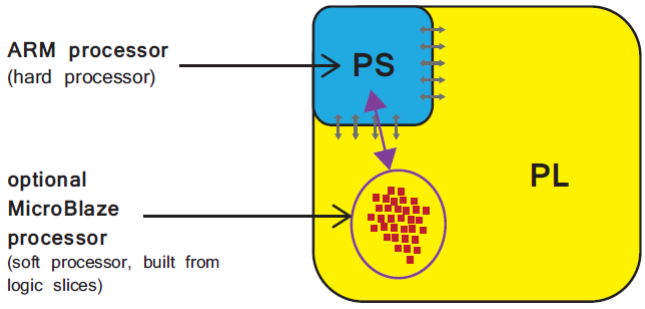



Figure 2.12: Locations of hard (ARM Cortex-A9) and soft (MicroBlaze) processors on a Zynq device

### 2.6.3 Application Processing Unit (APU)

The Fig. 2.14 below shows a simplified block diagram of the APU. APU is mainly consist of two ARM processing cores, each with related computational units: a NEON Media Processing Engine (MPE) and Floating Point Unit (FPU); a Memory Management Unit (MMU); and a Level 1 cache memory (in two sections for instructions and data). The APU also consist of a Level 2 cache memory, and a further On Chip Memory (OCM). The Snoop Control Unit (SCU) is considered as a bridge between the cores and the Level 2 cache and OCM memories [9].

From a programming perspective, backing for ARM instructions is achieved by the Xilinx Software Development Kit (SDK) which contains all needed elements to improve software for deployment on the ARM core.

### 2.6.4 Programmable Logic

The Fig. 2.15 shows the Programmable Logic which is the second essential part of the Zynq architecture. The PL basically consist of general purpose FPGA logic construction, which is constituted of slices and Configurable Logic Blocks (CLBs), and for interfacing there are also Input/Output Blocks (IOBs) [9]. Each of the labelled features in figure will be explained.

- **Configurable Logic Block (CLB):** Configurable Logic Block (CLB) CLBs are tiny, uniform sets of logic elements which are placed out in a two-dimensional array on the PL, and linked to other identical resources through programmable interconnects. Each Configurable Logic Blocks (CLB) is positioned next to a switch matrix and contains two logic slices, as shown in

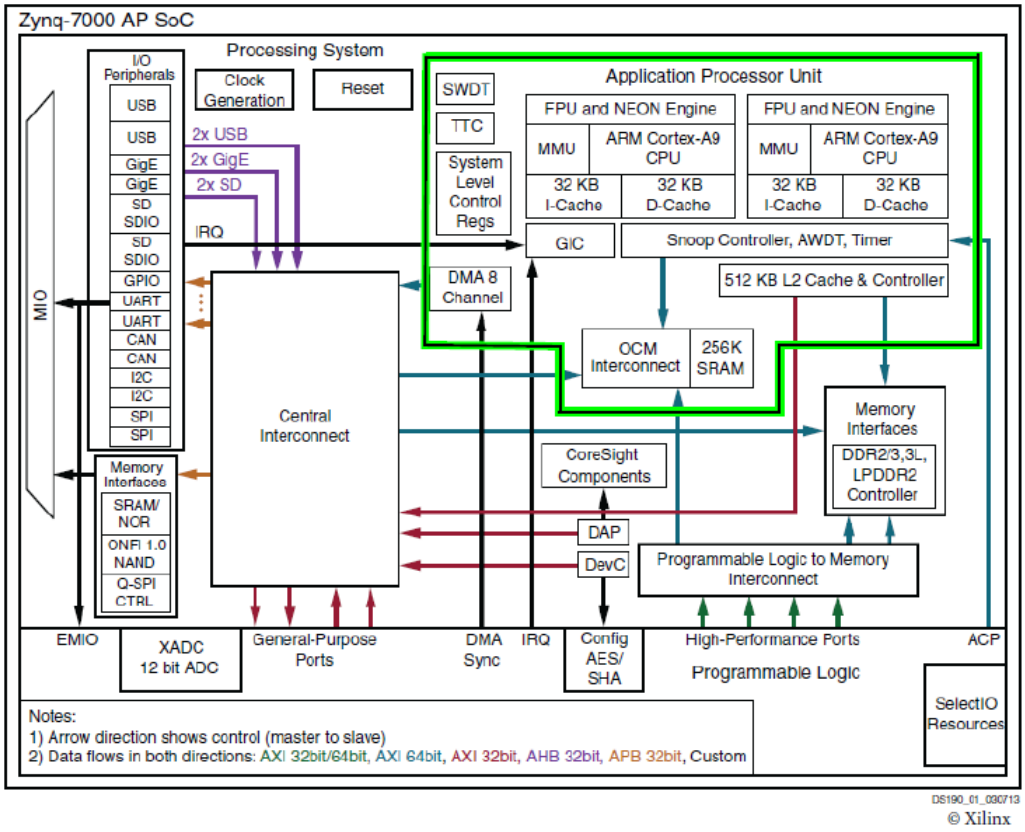



Figure 2.13: The Zynq Processing System [9]

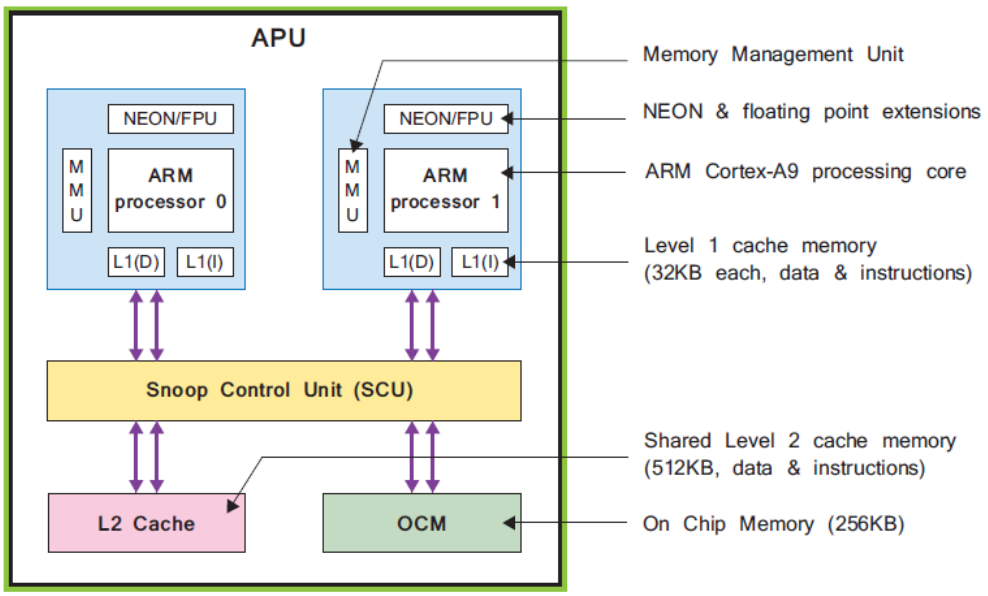



Figure 2.14: Block diagram of the application processing unit (simplified) [9]

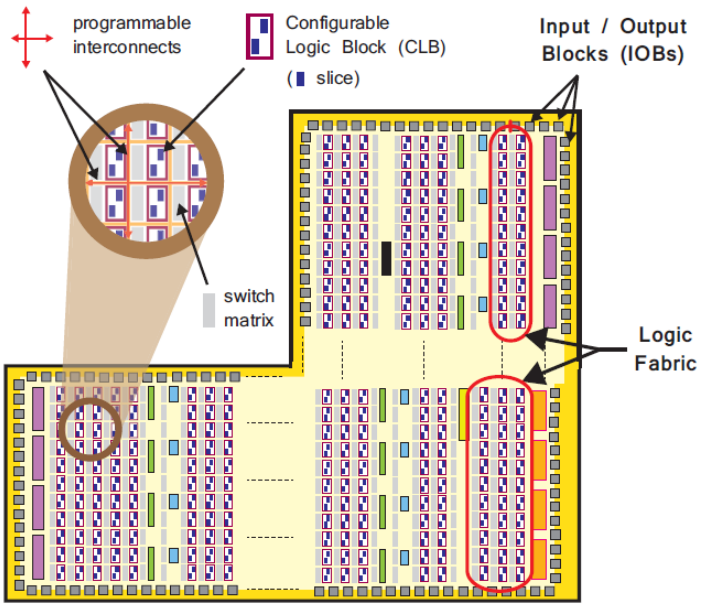



Figure 2.15: The logic fabric and its constituent elements [9]

Fig. 2.16.

- **Slice:** A sub-unit inside the CLB, which consists resources for implementing combinatorial and sequential logic circuits. As indicated in Fig. 2.16, Zynq slices are made up of 4 Lookup Tables, 8 Flip-Flops, and other logic.
- **Lookup Table (LUT):** A flexible resource capable of implementing (i) a logic function of up to six inputs; (ii) a small Read Only Memory (ROM); (iii) a small Random Access Memory (RAM); or (iv) a shift register. LUTs can be joined together to achieve bigger logic functions, memories, or shift registers, as wanted.
- **Flip-flop (FF):** A sequential circuit component fulfillment a 1-bit register, with reset functionality. Unit of the FFs can optionally be hired to perform a latch.
- **Switch Matrix:** A switch matrix take a seat after each CLB, and gives a flexible routing means for making links (i) between component within a CLB; and (ii) from one CLB to other resources on the PL.
- **Carry logic:** Arithmetic circuits require intermediate signals to be propagated between adjacent slices, and this is achieved via carry logic. The carry

logic comprises a chain of routes and multiplexers to link slices in a vertical column.

- **Input / Output Blocks (IOBs):** IOBs are resources which achieve connecting between the PL logic resources, and the physical device pads used to link to outer circuitry. Each IOB can handle a 1-bit input or output signal. IOBs are normally placed around the perimeter of the device.

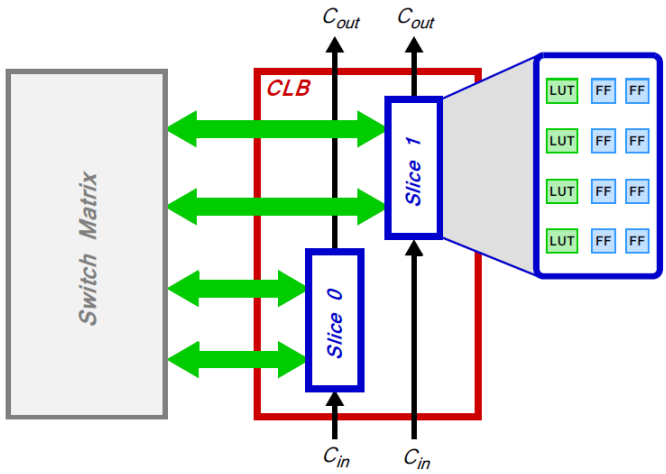



Figure 2.16: Composition of a Configurable Logic Block (CLB) [9]

### 2.6.5 Processing System - Programmable Logic Interfaces

The bridge between the two parts (PS,PL) is an Advanced eXtensible Interface (AXI) protocol which is part of the ARM Advanced Microcontroller Bus Architecture (AMBA) specification.The current version of AXI is AXI4. There are three types of AXI4 [28] and they will be listed in the following:

- **AXI4-full:** for requirements of high-performance memory-mapped.
- **AXI4-Lite:** simple, memory-mapped and used only for low throughput of data
- **AXI4-Stream:** for high-speed streaming data.

The term "memory mapped" is used in the above descriptions, and it is useful to briefly confirm its meaning. If a protocol is memory mapped, an address is specified

within the transaction issued by the master (read or write), which corresponds to an address in the system memory space.In the case of AXI4-Lite, which backs one data transfer per transaction, data is then written to, or read from, the specified address.

### AXI Interconnects and Interfaces

The primary interface between the PS and PL is via a set of nine AXI interfaces, each of which is composed of multiple channels. These make dedicated connections between the PL, and interconnects within the PS [9], as indicated in Fig. 2.17. It is useful to briefly define these two important terms:

- **Interconnect:** An interconnect is effectively a switch which manages and points traffic between attached AXI interfaces. There are many interconnects inside the PS, some which are imediately linked to the PL (as in Fig. 2.17.), and others which are for internal hire only. The connections between these interconnects are also created using AXI interfaces.

- **Interface:** A point-to-point link for crossing data, addresses, and handshaking signals among initiator and slave clients inside the system.

Note from the Fig. 2.17 that all of the interfaces are specifically linked to AXI interconnects existing inside the PS, with the exception of the ACP interface, that is linked directly to the Snoop Control Unit within the APU. Internally to the processing system, AXI interfaces are used within both the ARM APU (making connections between the processing cores and SCU, cache memory and OCM), and usually to link the different interconnects inside the PS. These connections are in addition to those at the PS -PL borders.Furthermore, the three interconnects illustrated in Fig. 2.17 (the Memory, Master and Slave Interconnects) are internally linked to the Central Interconnect, which is not shown here. Full specifics of PS internal connections, including a block diagram showing all AXI interconnects and interfaces, are available in [10].

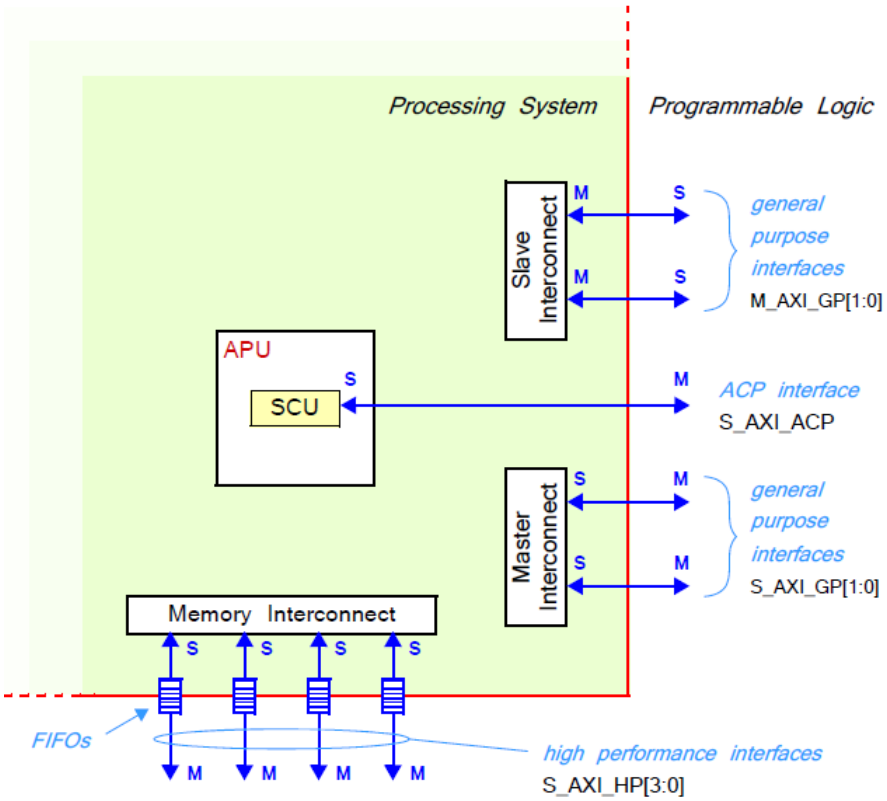



Figure 2.17: The structure of AXI interconnects and interfaces connecting the PS and PL [10]

# Chapter 3

# Cryptography and Previous Work

## 3.1 Introduction to Cryptography

Cryptography is the study and application of mechanisms that conceal the true meaning of data by converting it into formats which are non-readable for human and achieving the security goals [29].

### 3.1.1 Cryptographic Goals

Fig. 3.1 illustrates the security requirements of a simple IoT framework, in which the essential security requirements are classified from many aspects [29]:

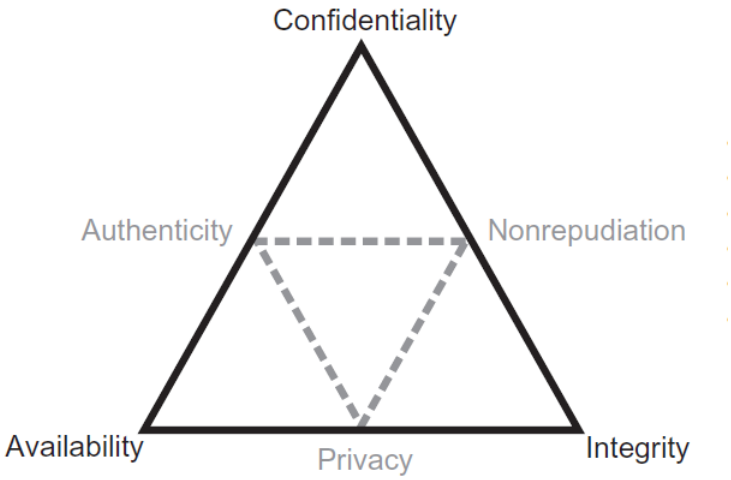



Figure 3.1: Security requirements in IoT .

- **Confidentiality:** ensures that, the exchanged data during a communication are kept confidential. Confidentiality is generally ensured through encryption [29].The data to be send is encrypted using a secret key and therefore encrypted text (cipher-text) is sent instead of the plain-text as shown in Fig. 3.2.

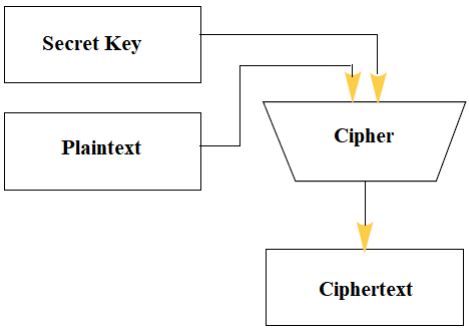



Figure 3.2: Block Diagram of a Confidentiality Process.

Ensuring data confidentiality is critical for IoT applications. Indeed, any failure would seriously threaten parties's privacy. Therefore, a vast deployment of IoT applications might be blocked. To achieve data confidentiality, cryptographic algorithms are commonly used to encrypt data. Doing so, even if the transferred data is attacked, the adversary will not be able to access its content.

- **Integrity:** integrity ensures that exchanged data between two entities during a communication process has not been altered by unauthorized entities [29]. Integrity is generally ensured by using cryptographic one-way hash functions.The input of these functions is an arbitrary length message but the output will be a fixed size message digest as shown in Fig. 3.3.

  The sender will compute the hash for the message, after that he will send the message and it's hash. Then the receiver will compute the hash for the message that he recieved, finally he will compare the computed hash with the hash value that he received as shown in Fig. 3.4. If the computed hash value matched with the hash value that received, that is mean the message has arrived correctly without alteration.

- **Authentication:** authenticity validates the origin of the data. Authentica-

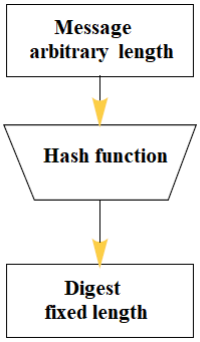



Figure 3.3: Block Diagram of a One-Way Hash Function.

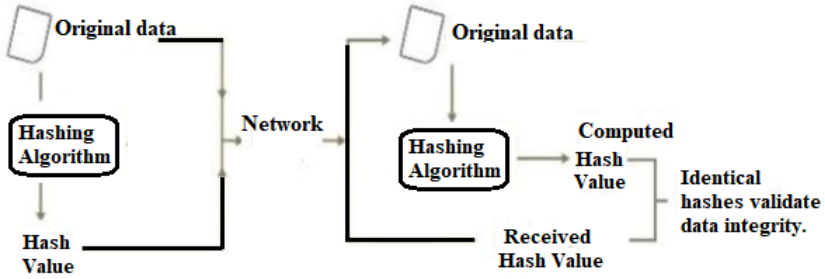



Figure 3.4: Data Integrity Check.

tion algorithms are often called Message Authentication Codes (MAC),and similar to hash functions generate a specific sized output called a message tag [29].The tag would be the data a verifier could utilize to validate a message. Unlike hash functions, the set of MAC functions needs a private key to block anyone from Falsification of tags as shown in Fig. 3.5.

- **Availability:** guarantees that data are available when needed by authorized entities.

- **Nonrepudiation:** guarantees the means to verify that an entity has really participated in an exchange of data, such as sending/receiving information or a digital signature [29].

However, cryptographic algorithms are classified into two essential groups [29].

- **Symmetric protocols**: in this class of algorithms, the same shared key between the involved parties is used to cipher and decipher data. Sometimes called as Secret-key cryptography.

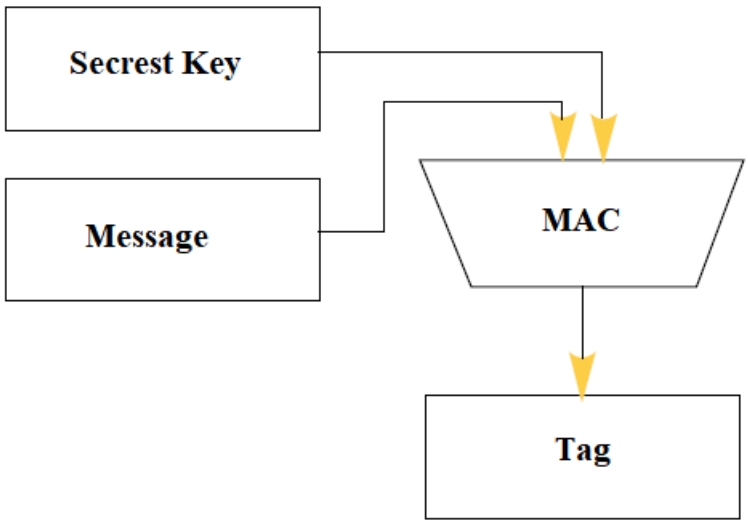



Figure 3.5: Block Diagram for a MAC Function.

- **Asymmetric protocols**: in this class of algorithms, a couple of public/ private keys is applied in the encryption/decryption process. The encrypting party uses the public key of the recipient to encrypt message. Public keys are not kept secret. To decrypt the encrypted data, the recipient uses its private key. Unlike public keys, private keys are kept confidential and only obtainable from their owner. Sometimes called as Public-key cryptography.

However, we will focus only on the Symmetric Cryptography due to the proposed design of this thesis based on Authenticated Encryption (AE).

### 3.1.2 Secret-key cryptography Types

- **Block cipher:** is a cipher algorithm that cipher a constant size of n-bits of data - known as a block - at one time. The normal sizes of each block are 64 bits, 128 bits, and 256 bits. So for instance, a 64-bit block cipher will gather 64 bits of plaintext and cipher it into 64 bits of ciphertext. In situations where bits of unencrypted-message (plaintext) is lower than the block volume, padding algorithms are called into play.For example, AES is a block cipher that encrypts a 128-bit (16-byte) block using a 128-bit, 192-bit, or 256-bit key [30].

- **Stream cipher:** is an encryption algorithm that cipher 1 bit or byte of unencrypted-message (plaintext) at a time. It utilizes an infinite stream of

pseudo random bits as the key. It encrypts a changeable-length message by use a public nonce (a message number used only one single time) and a privates key shared by the transmitter and receiver. For example RC4 [30].

- **Message-authentication code:** generates an authenticator of a changeable-length message using a secret key shared by the transmitter and receiver; some message-authentication codes also use nonces. Sending the authenticator simultaneously with the message protects the message against corruption. Message-authentication codes are usually structured from block ciphers or from cryptographic hash functions like SHA-3 [30].

- **Authenticated cipher:** An authenticated cipher, also known as an authenticated-encryption algorithm or Authenticated Encryption (AE) scheme, ciphers and authenticates messages, by use a public nonce and a private key shared by the transmitter and recipient. Authenticated ciphers are usually constructed as various combinations of block ciphers, stream ciphers, message-authentication codes, and hash functions [30].

However, the proposed design for this thesis deals only with authenticated cipher. Table 3.1 Simplifies the differences between the types of the Secret-key cryptography.

Table 3.1: Secret-key cryptography Types

| | **Message length** | **Encrypts** | **Authenticates** |
|---|---|---|---|
| Block cipher | Fixed | Yes | No |
| Stream cipher | Variable | Yes | No |
| Message-authentication code | Variable | No | Yes |
| Authenticated cipher | Variable | Yes | Yes |

## 3.2 Authenticated Encryption

Authenticated Encryption (AE) is a symmetric encryption algorithm which is mainly a combination of authentication and encryption that guarantees both confidentiality and authenticity of the data that is need to be send. Any scheme that guarantees authenticated encryption takes the input plaintext (m), and key (K) and gives ciphertext (C) and a tag (T) as output [31].The main purpose of the tag

is to check if the correct ciphertext is received, Considering that the tag considered as a checksum of the message. Another class of AE schemes is authenticated encryption with associated data (AEAD) which supports both data that needs encryption along with authentication and data that only needs authentication. Fig. 3.6 illustrates the basic block diagram of an AEAD.

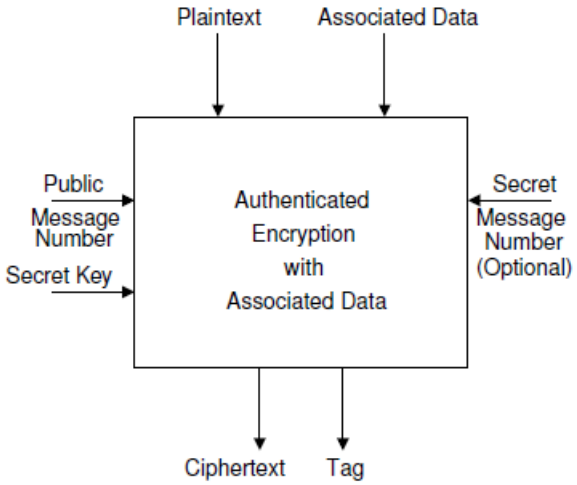



Figure 3.6: Block Diagram for Authenticated Encryption with Associated Data.

**Encryption Flow for AEAD Algorithm (Alice)**

To encrypt the plain-text, then the input for AEAD will be the plain-text,associated data,nonce and key and the output will be the ciphertext,tag and the associated data as shown in Fig. 3.7.

**Decryption Flow for AEAD Algorithm (Bob)**

To decrypt the cipher-text, then the input for AEAD will be the cipher-text,associated data,nonce,tag and key and the output will be the associated data and the plain-text as shown in Fig. 3.8.

### 3.2.1 Advantages of Authenticated Encryption

Combining authentication and encryption into a single hardware-level algorithm can lead to the advantages listed below [31].

- When using one algorithm to achieve authentication and encryption, then the required area will be less than if an encryption algorithm and another one for authentication were used

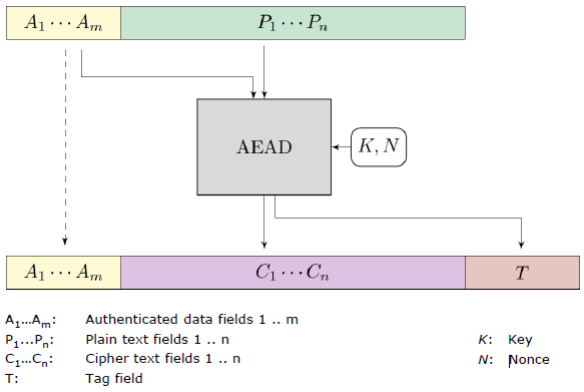



Figure 3.7: Encryption Flow for AEAD algorithm.

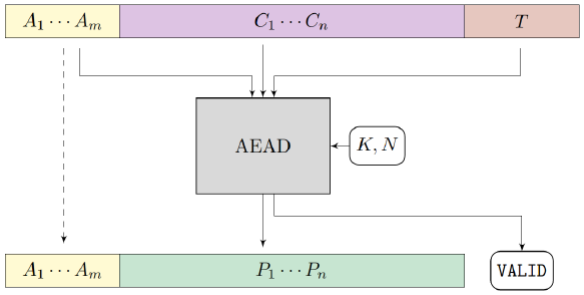



Figure 3.8: Decryption Flow for AEAD algorithm.

- designs with smaller area consume less power and this is a good solution for low-power applications.
- A combined algorithm needs only one key and so has a slight advantage in the issues of key management and key storage.

### 3.2.2 AE(AD) Constructions

Any AE algorithm is mainly a integration of an encryption algorithm and an authentication algorithm. There are three types of composition schemes for obtaining authenticated encryption and they vary in the way these two algorithms are combined [32].

1. **Encrypt-then-MAC(EtM):** In this schema, the message is encrypted first and the tag is calculated by taking the MAC over the acquired encrypted text. On the receiving side, the tag is first checked, and if it matches decryption will take place to obtain the plain-text. Fig. 3.9 shows the operation of

EtM composition scheme [32].

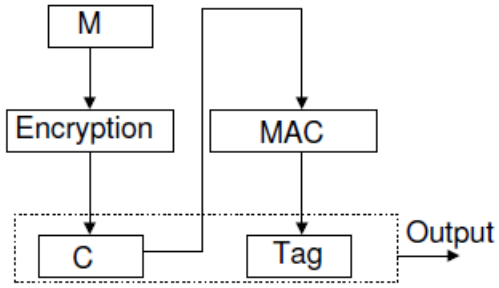



Figure 3.9: Encrypt-then-MAC(EtM).

2. **MAC-then-Encrypt(MtE):** A MAC is generated depending on the unencrypted-message (plaintext), then the unencrypted-message (plaintext) and MAC are together ciphered to generate an encrypted-message (ciphertext) based on both. After that, the sender will send the ciphertext (containing an encrypted MAC). And on the receivers side first decryption will takes place to get plaintext and tag pair and then verifies the tag. Fig. 3.10 shows the operation of MtE composition scheme [32].

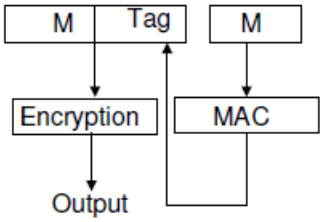



Figure 3.10: MAC-then-Encrypt(MtE).

3. **Encrypt-and-MAC (E&M):** A MAC is generated depending on the unencrypted-message (plaintext), and the plaintext is ciphered without the MAC. The plaintext's MAC and the encrypted-message are sent together. Fig. 3.11 shows the operation of E&M composition scheme [32].

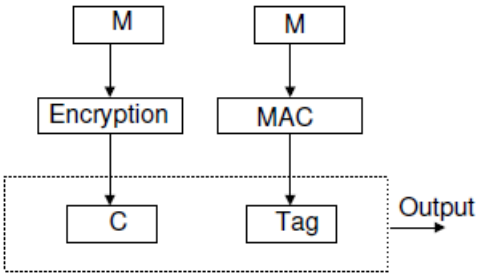



Figure 3.11: Encrypt-and-MAC (E&M).

## 3.3 CAESAR Competition

### 3.3.1 Introduction

CAESAR stands for " Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) ". The contest has called for submissions of authenticated ciphers on 2/1/2014.The ciphers in this contest ensure the entity that the data comes from the right person (authenticity), that only he can read it (confidentiality) and that it has not been altered (integrity) [33]. The main purpose of the contest is to find cryptography algorithm that has higher features than the current AEAD schemes and is appropriate for a very wide range of applications. The contest has certain requirements that each cipher must comply with.

### 3.3.2 Functional Requirements of the CAESAR Contest

The requirements of CAESAR competition are listed [33]:

- The cryptograpgy algorithm must gurantee both integrity and confidentiality to Plaintext and Secret message number and also integrity to Associated data and Public message number i.e., the cryptograpgy algorithm should be Authenticated Encryption with Associated Data(AEAD), which is a special situation of Authenticated Ciphers.

- The length of ciphertext should be specified by the length of plaintext ; i.e., it should not leak any data other than the plaintext length via the ciphertext length, for example by having the ciphertext length be the plaintext length

plus a specified constant.

- the submission should contains a list of recommended parameters. The number of recommendations should not exceed 10.

However, The contest is at the fourth and final round at the time of creating the design proposed in this thesis. 5 of the 7 finalists are chosen for the proposed design: AEGIS, ASCON, COLM, Deoxys and OCB. The selection of these 5 algorithms is a case study to show the effectiveness of using the proposed algorithm hopping technique and this technique is expendable to accommodate any other security algorithms.

### 3.3.3 AEGIS

AEGIS is a dedicated authenticated encryption algorithm where a message is used to update the state of the cipher, and message authentication is achieved almost for free [34]. AEGIS is constructed from the Advanced Encryption Standard (AES) round function. It has three variations: AEGIS-128L, AEGIS-128 and AEGIS-256. The first one uses eight AES round functions to process a 32-byte message block, the second one processes a 16-byte message block with five AES round functions and the third one uses six AES round functions. AEGIS is very fast and its computational cost is half that of AES [34]. AEGIS also offers high security as long as the initialization vector is not reused, which makes it impossible to recover the AEGIS state and key faster than exhaustive key search. AEGIS is suitable for network communication since it can protect a packet while leaving the packet header (associated data) unencrypted. The state update function of AEGIS-128 updates the 80-byte state $S_i$ with a 16-byte message block $m_i$. $S_{i+1}$= StateUpdate128($S_i$, $m_i$) is given as follows:

$$S_{i+1,0} = AESRouund(S_{i,4}, S_{i,0} \oplus m_i) \tag{3.1}$$

$$S_{i+1,1} = AESRouund(S_{i,0}, S_{i,1}) \tag{3.2}$$

$$S_{i+1,2} = AESRouund(S_{i,1}, S_{i,2}) \tag{3.3}$$

$$S_{i+1,3} = AESRouund(S_{i,2}, S_{i,3}) \tag{3.4}$$

$$S_{i+1,4} = AESRouund(S_{i,3}, S_{i,4}) \tag{3.5}$$

The state update function is shown in Fig. 3.12 :

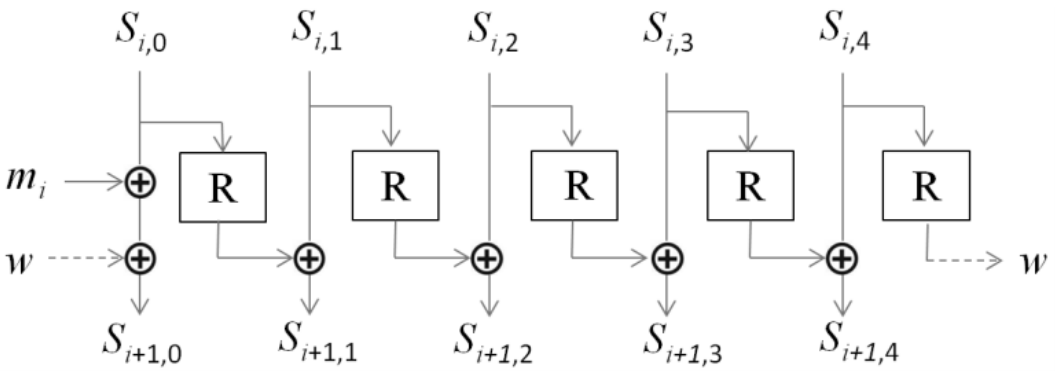



Figure 3.12: The state update function of AEGIS-128. R indicates the AES encryption round function without XORing with the round key and $w$ is a temporary 16-byte word [34]

### 3.3.4 ASCON

ASCON presents a low hardware scheme with extremely low memory requirements. While the scheme provides full security of 128 bits, it offers no nonce misuse resistance [35]. It is based on the Sponge wrap/Monkey Duplex mode of operation and eliminating inverse operations [36]. ASCON is optimized for minimal overhead (cipher text = plain text) and it is lightweight for constrained devices making it fast in hardware and software implementations. ASCON has several parameters used for encryption such as: secret key (K), associated data (A), public message number that is denoted by nonce (N)), and Initialization Vector (IV), in order to encrypt a plaintext (P), according to the formula:

$$E_{a,b,k,r}(K, N, A, P) = (C, T) \tag{3.6}$$

where a, b are the numbers of permutation rounds, r is the state size, and k is the secret key size. The output of this process is the ciphertext C, and the

authentication tag T [35]

Fig. 3.13 illustrates ASCON modes of operations which are the encryption mode and the decryption mode. P block is the main block in ASCON algorithm. P block has two flavors: one for carrying out the initialization/finalization process (Pa) and the other for performing the internal processes (Pb).

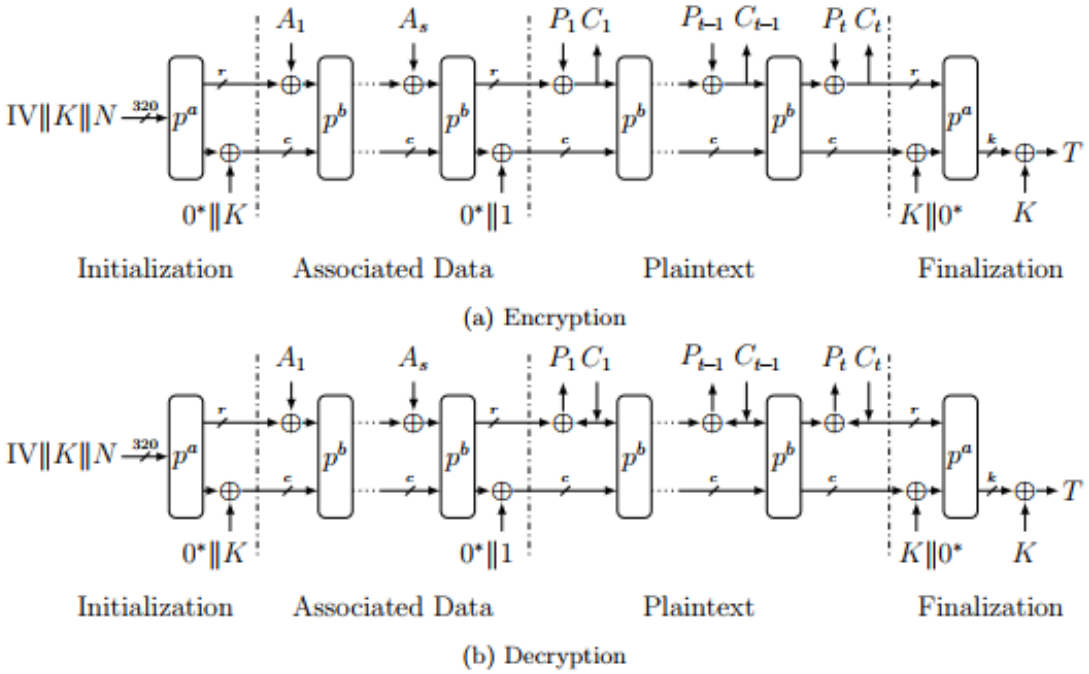



Figure 3.13: ASCON's mode of operation [35]

### 3.3.5 COLM

COLM [37] is a block cipher based on Encrypt-Linear mix-Encrypt mode, designed with the goal to achieve online misuse resistance, to be fully parallelizable, and to be secure against blockwise adaptive adversaries.

The authenticated encryption for complete message block is shown in Fig. 3.14 . COLM consists of two-layer parallelizable encryption. COLM mixes the output of the first encryption layer to generate the input to the second encryption layer, using linear mixing function. The high-speed COLM implementation instantiates two instances of AES to implement the two layers of encryption. In order to optimize COLM for low area, only one instance of AES is used to perform the two encryption layers. A Finite state machine and Multiplexers are added to control the data flow to the AES. The optimized encryption operation is processed in twice

the clock cycles of the non-optimized one and the same applies for the decryption operation.

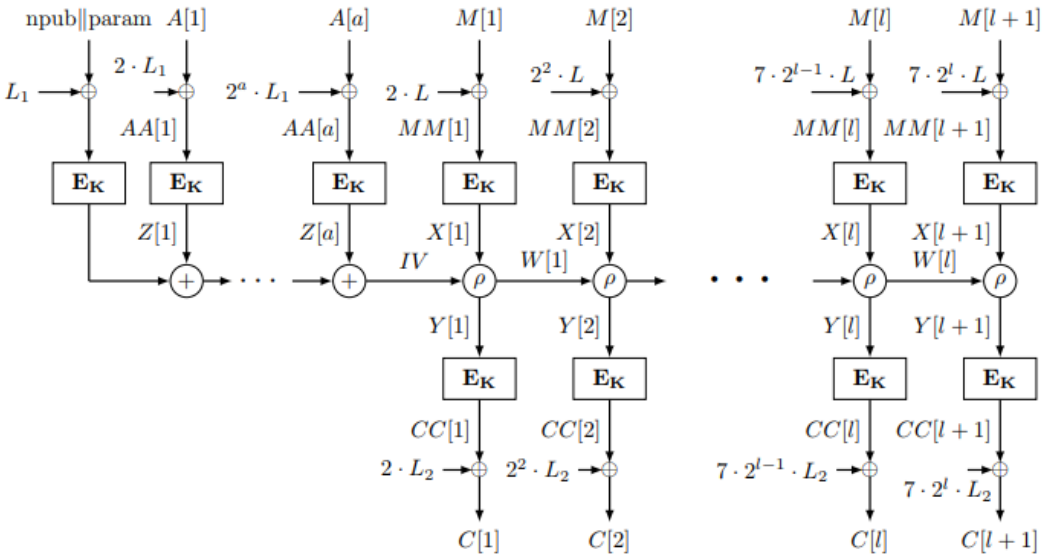



Figure 3.14: COLM authenticated encryption for complete message block. $E_K$. denotes the block cipher AES-128 [37]

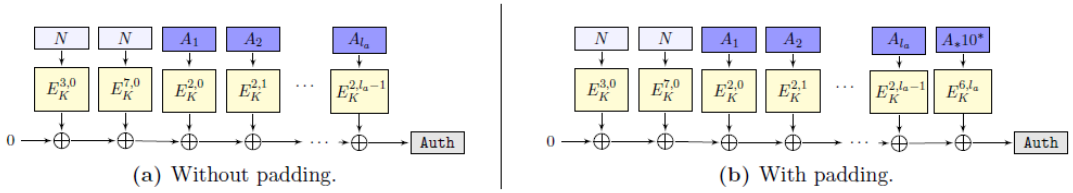
### 3.3.6 Deoxys

Deoxys presents a new authenticated encryption design based on custom made tweakable block ciphers using the AES round function as a building block [38]. Deoxys is an authenticated encryption scheme that provides full 128-bit security for both privacy and authenticity making it efficient in software. Moreover, Deoxys performs particularly well for small messages (only m + 1 block cipher calls are required for an m block message and no precomputation is required). In the nonce-misuse resistant versions of Deoxys, in addition to a full 128-bit security for unique nonces, birthday-bound security is obtained when the nonce is reused. Finally, Deoxys can be lightweight and the key can be hardcoded for further smaller area footprint. Deoxys uses a tweakable block cipher Deoxys-BC as internal primitive. Deoxys has two main mode variants:

- Nonce-Respecting Mode: this variant is for where adversaries are assumed to be nonce-respecting, meaning that the user must ensure that the value N will never be used for encryption twice with the same key.

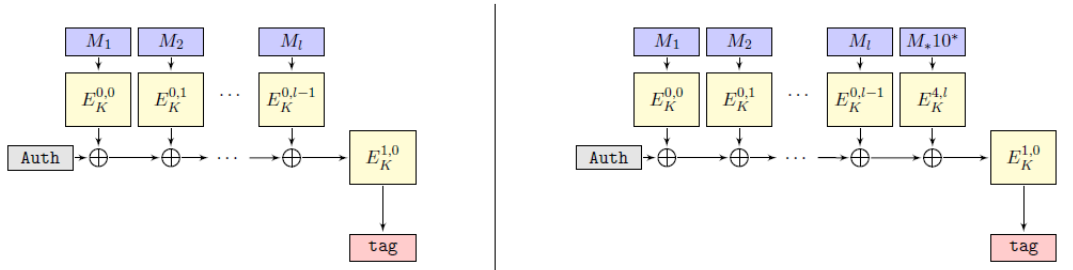
- Nonce-Misuse Resistant Mode: this variant is a new authenticated encryption mode named SCT , relaxes this constraint and allows the user to reuse the same N with the same key.

In this work, the focus is only on the second mode. The encryption algorithm is depicted in Fig. 3.15 and Fig. 3.16 for the authentication part and in Fig. 3.17 for the encryption part.



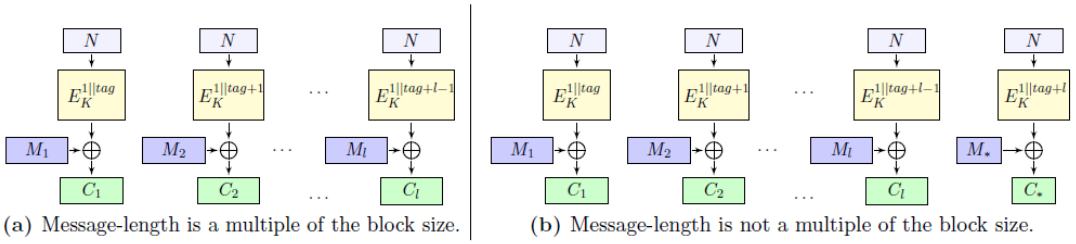

(a) Without padding. (b) With padding.

Figure 3.15: Handling of the associated data for the nonce-misuse resisting mode: in the case where the associated data is a multiple of the block size, no padding is needed [38].




(a) Without padding. (b) With padding.

Figure 3.16: Message processing in the authentication part of the nonce-misuse resisting mode: in the case where the message-length is a multiple of the block size, no padding is needed [38].




(a) Message-length is a multiple of the block size. (b) Message-length is not a multiple of the block size.

Figure 3.17: Message processing for the encryption part of the nonce-misuse resisting mode [38].

### 3.3.7 OCB

OCB (short for Offset Codebook) is an AEAD scheme that depends on a block cipher that must have a 128-bit block size [39]. OCB achieves both confidentiality and authenticity. Confidentiality is defined such that an adversary is unable to distinguish OCB-outputs from an equal number of random bits, while authenticity means that an adversary is unable to produce any valid nonce-cipher text pair that it has not already acquired. OCB is nearly as fast as Counter mode (CTR) because each block encryption requires just a few xors on top of an AES call. OCB is parallel as most of the computations are independent of one another which allows both hardware and software accelerations. It is also designed for minimal authentication overhead beyond what is required for provable security and simple encryption using a block cipher. OCB is not designed to resist nonce reuse or to enjoy beyond birthday bound security [39]. Fig. 3.18 is the illustration of OCB$[E, \tau]$. Here $E$: $\kappa \times \{0,1\}^{128} \rightarrow \{0,1\}^n$ is a blockcipher and $\tau \in [0 \,..\, 128]$ is the tag length. $M, C \in \{0, 1\}^*$. In the top section of the figure, Message $M$ has a full final block ($|M4 = n|$) (Checksum= $M_1 \oplus M_2 \oplus M_3 \oplus M_4$ ). In the middle section, Message $M$ has a short final block, $1 \leq |M_*| < n$ (Checksum= $M_1 \oplus M_2 \oplus M_3 \oplus M_*10^*$ ). In the bottom side, An AD of three full blocks (left) or two full blocks and one short one (right). Throughout: Offsets (the -values) are updated and used top-to-bottom, then left-to-right. Offset initialization and update functions (Init, $\text{Inc}_i$, $\text{Inc}_\$$, $\text{Inc}^*$) return n-bit strings. Each flavor of increment is an xor with some precomputed, $K$-dependent value [39].

### 3.3.8 The GMU Hardware API for the CAESAR

The GMU Hardware Application Program Interface (API), is a hardware API proposed to provide a common external interface for the hardware implementations of the ciphers participating in the CAESAR competition. It makes the comparison of different algorithms easier and fairer. This API named AEAD has the following features:

- Wide range of data port widths ranging from 8-bits to 256-bits.
- Independent data and key inputs.

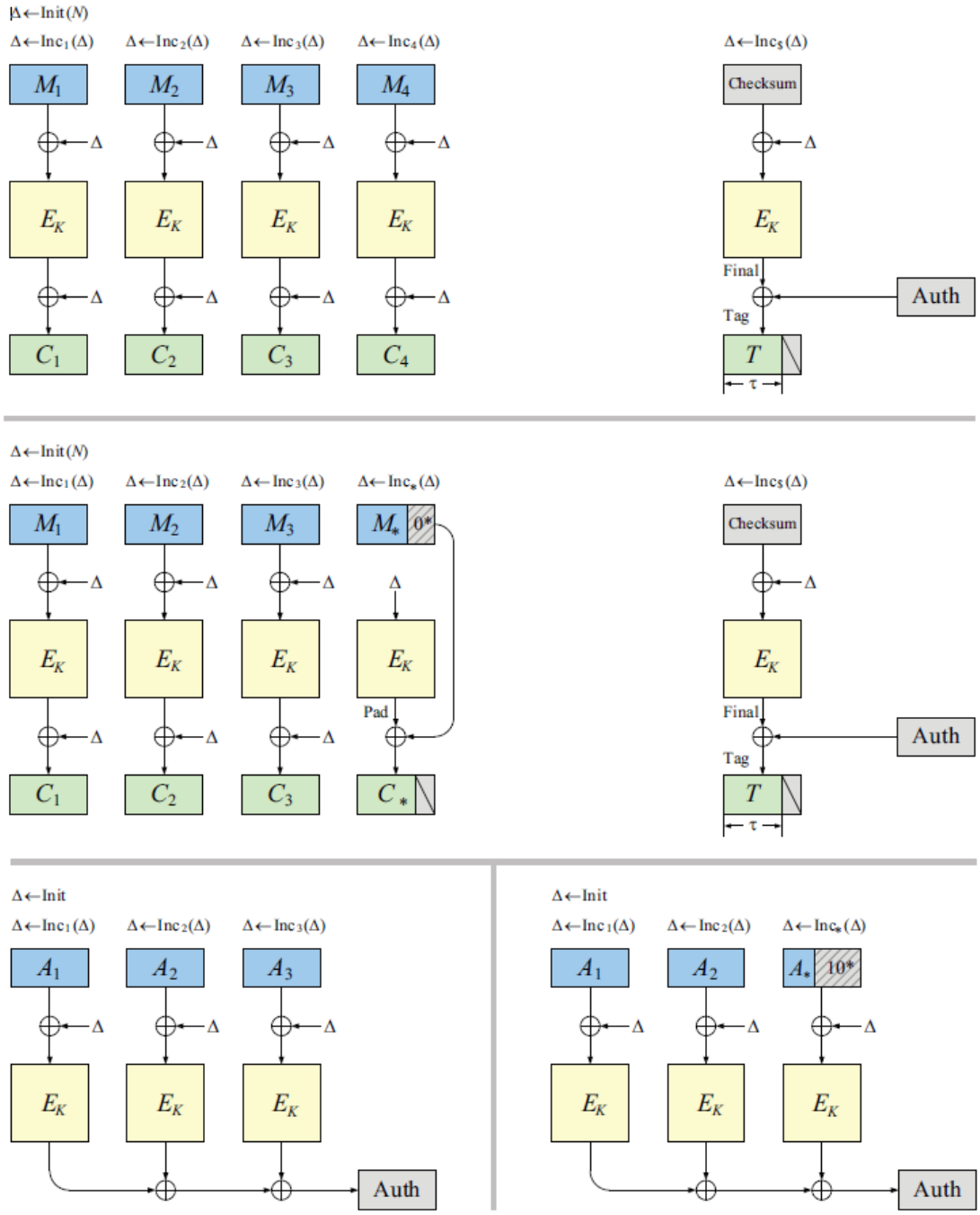



Figure 3.18: Illustration of OCB [39]

- Simple high-level communication protocol.
- Support for encryption and decryption within the same core.
- Ability to communicate with First-In, First-Out (FIFO)s.

The full specifications, features and the usage manual can be found in the API paper [11].

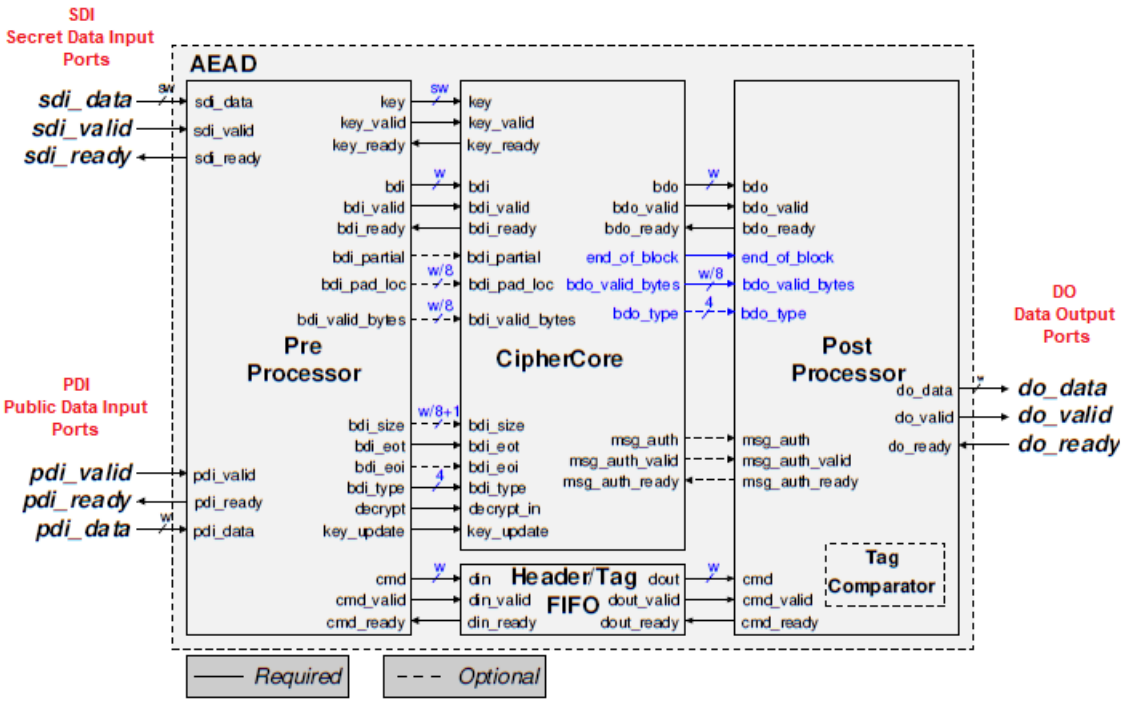



Figure 3.19: Top-level block diagram of a lightweight architecture of AEAD [11]

The GMU Hardware API separates the development of the Core (which is called CipherCore), containing the cipher specific part, and the external communication. One of the useful features is the support for a wide range of data port widths (ranging from 8 to 256 bytes), which are functionally completely separated from the CipherCore. Furthermore, it also supports an arbitrary length of the input stream. There is support for encryption and decryption with the same core. It is relative lightweight, and it can communicate with simple devices like FIFOs to use as memory [11].

The API uses a PreProcessor and a PostProcessor. The PreProcessor provides the key, public message number, secret message number, block data and tag to the CipherCore, with information about the expected operation. This indicates whether the data is Associated Data (AD) or plaintext, whether the core has to encrypt or decrypt, if the current is the last block, and more. The PostProcessor then takes care of the output, accepting the encrypted block from the CipherCore and delivering it to the output port [11].

Because of the availability of the open source code for the PreProcessor, Post-

Processor, and CMD FIFO, the designers of lightweight implementations of authenticated ciphers can focus exclusively on the development of the CipherCore unit.

The interface has a common public data input (pdi) port for associated data, nonce, plaintext, ciphertext and tag. An additional signal pdi_type is used to specify the type of data that is coming on the bus. The key and secret message number come through the secret data input port. The data output port do_data carries the cipher text and tag.

The input decrypt_in signal informs the core whether the current operation is encryption or decryption.

The CipherCore must provide msg_auth to indicate its result and set msg_auth_valid to high until the PostProcessor is ready (msg_auth_ready is active) [11].

## 3.4 Previous Work

Ibrahim et al. [40] proposed a low cost FPGA based cryptosystem named as Secure Cipher for high throughput to area ratio. A full loop unroll technique have used to implement the proposed design. The iterations of every loop in the algorithm are unrolled in such a way that the output of each iteration becomes the input of the successive loop iteration. The proposed Secure Cipher is low complexity encryption algorithm based on Feistal structure. It is a block cipher that consists of 5 encryption rounds only. Each encryption round consists of five logical and mathematical operations that operate on 8-bit data. The target device for the proposed cryptosystem implementation was a low cost Altera Cyclone II EP2C35F672C6N FPGA using Verilog HDL and the design was synthesized using Quartus II 12.1 sp1 edition. The proposed system has a throughput of 4600Mbps with 5.735Mbps/LE throughput to area ratio and the utilization was 802 LE.

HAFSA1 et al. [3] proposed an improved AES-ECC Cryptosystem using a co-design approach where AES runs on NIOS II softcore and ECC's scalar multiplication is implemented as a hardware accelerator. The proposed design provides advantages of both asymmetric-key and symmetric-key algorithms. The symmetric-key algorithm is used to encrypt a data transmitted in an insecure channel while the asymmetric-key is used to share the key with the other party so

that he can decrypt this data. The original AES-Key is generated by a Random Number Generator (RNG) in C and running on NIOS II. The FPGA-based DE2-115 development board featuring a Cyclone IV (Altera) was used for this work. The implementation results show that the design uses 11% of total logic elements, 9% of total combinational function and 7% of total memory. It runs at a frequency of 157.63 MHz and consumes 166.67 mW.

Soliman et al. [2] proposed 2 AES encryption designs based on the idea of integrating between iterative looping and pipelining to optimize between area, throughput and power to provide a competitive design ready to use in IoT low power enabling technologies . These designs were implemented using VHDL and synthesized using Xilinx ISE 14.2 on XC5VLX50-3 Virtex 5 FPGA device. The results showed a competitive throughput of 34 Gbps, and efficiency of 65.42 and 50.58 Mbps/slice respectively. Both designs were also synthesized using Vivado 2014.4 design suite and mounted on Zynq-7000 XC7Z010clq225-3 FPGA device to provide dynamic power consumption calculations. The results showed that the total dynamic power consumption reached 455 mW.

### 3.4.1 Limitations in the Previous work

All previous works support one dimension of security which is the algorithm itself. The cryptography scheme is constant all the work time of the design. Therefore, the attacker needs only to try to know the key, after that he able to attack the system.

However, In our proposed design, the algorithm itself is changeable as we will explain in the following chapters.

# Chapter 4

# Methodology and Proposed Design

## 4.1 Algorithm Hopping

Algorithm hopping is based on Frequency Hopping Spread Spectrum (FHSS) which is an approach to send radio signals by rapid switching of carriers between many frequency channels. The switching is based on a pseudo-random pattern only known to the transmitter and the receiver [41]. It is widely used as a multiple access method in the Code Division Multiple Access Scheme (CDMA) in communication systems.

The proposed design uses this idea by modeling the frequency channels as the authenticated encryption (AE) algorithms while sending the block data to be encrypted/decrypted as an equivalent to sending radio signals over different frequency channels. Normally, any cryptography systems uses one algorithm all the time and only the key is changeable during the run time. In our proposed algorithm hopping, the algorithm will be change per session. Also the key can be changed at any time. Therefore, if the attacker needs to hack our proposed design, he needs to know how many algorithms we use and which is the algorithm that used at each session. After that he needs to know the key of the used algorithm. That's mean our proposed design is more secure than the current designs. How the proposed design increased the level of security is discussed in chapter 6 , section 6.4

Figure 4.1 shows our proposed algorithm hopping technique, the design jumps between the 5 AEAD schemes that we chose from CAESAR competition. For example, during the session 2, the algorithm is COLM and for the session 4 , the AEAD scheme is OCB.

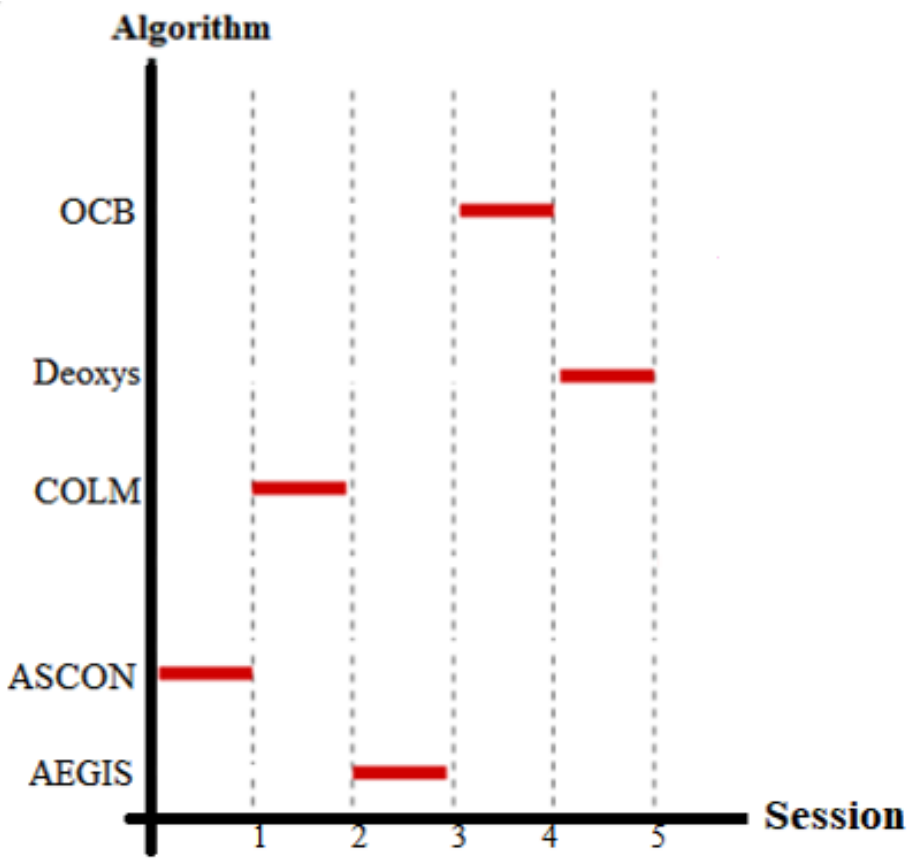



Figure 4.1: Proposed Algorithm Hopping technique

However, in the testing phase of the proposed design, we chose the session as one message. Consequently, we sent 5 messages and each message with different AEAD scheme. Alternatively, the session in the real life will be for example 1 [K] messages or 10 [K] messages, and who determines the number of messages is the application itself. If the application needs to read the output of the sensor each one hour, then the number of messages during a day will be less than the application that needs to make acquisition for data each 1 [ms]. Then the number of messages per session for the first application will be different from the second application. The sequence of switching is random according to the output of the pseudo random number generator which is Linear Feedback Shift Register (LFSR) circuit in our proposed design. The output of the LFSR is based on seed value

which is able to be changed during the run time through the PS.

LFSR is performed as a chain of Flip- Flops, connected together as a shift register [42]. Some taps of the shift register chain are used as inputs to either an XOR or XNOR gate.The output of this gate is then used as a feedback to the beginning of the shift register chain. There are some special properties of the LSFR that can be listed as follows:

- LFSR patterns are pseudo-random.
- Output patterns are deterministic. The next state can be figured out by knowing the position of the XOR gates as well as the current pattern.
- A pattern of all 0's cannot appear when the taps use XOR gates.
- A pattern of all 1's cannot appear when the taps use XNOR gates.
- The maximum possible number of iterations of any LFSR:

$$= 2^{bits} - 1 \tag{4.1}$$

An example design of a 5-bit LSFR using XNOR gate is shown in Fig. 4.2.

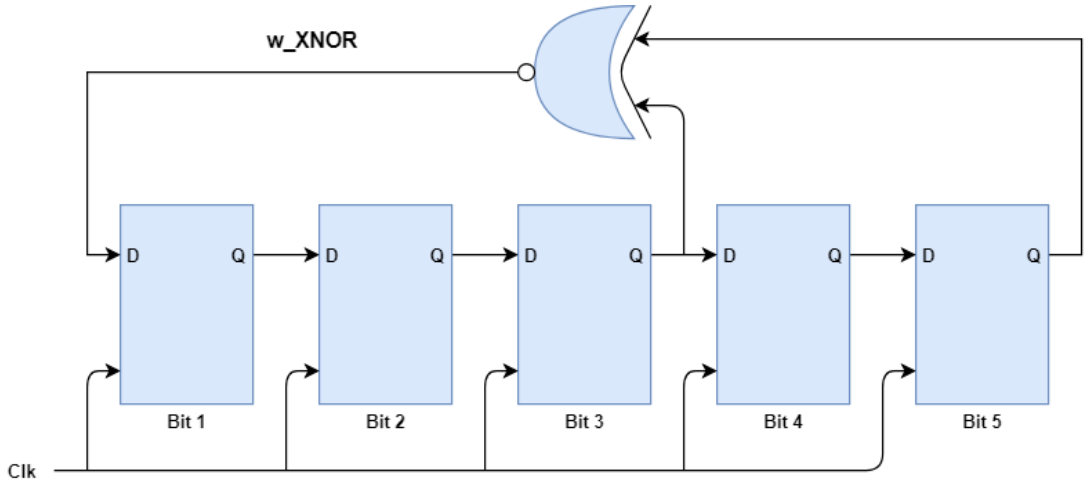



Figure 4.2: 5-bit LSFR using XNOR gate

## 4.2 Dynamic Partial Reconfiguration

### 4.2.1 FPGA Configuration

Just to recap, FPGAs contains a large amount of programmable logic and registers, which can be connected together in different ways to realize different functions [12]. it is sometimes useful to imagine that the FPGA is consist of two distinct layers: the logic gates/registers and the programmable SRAM configuration cells ( configuration Memory (CM) ) as shown in Fig. 4.3.

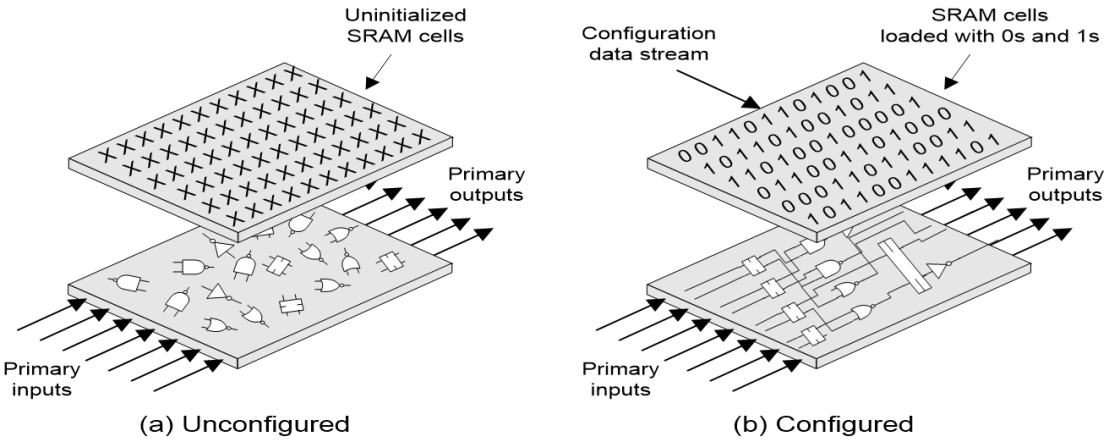



Figure 4.3: Two Distinct Layers of FPGA [12]

An FPGA is reconfigured by writing the bit-stream into the Configuration Memory (CM) that controls function computed on logic layer. FPGA architectures can be categorized according to the capability of the configuration, as illustrated in Fig.4.4. At the top level, FPGAs can be divided into one-time configurable devices that can only be applied as an ASIC substitute and configurable FPGAs. Configurable FPGA devices can in turn be distinguished in partially and globally reconfigurable devices [12].

When use of globally reconfiguration to configure the FPGA, the whole device configuration is swapped. Consequently, all the states inside the FPGA get lost and the device will have to restart its operation. However, focus will be put on partially reconfigurable systems, which allow to update only a portion of the resources of an FPGA. Partial reconfiguration can be achieved either passive by reconfiguring a portion of the device (changing the functionality) when the device is inactive without affecting other areas of the device or active where the operation

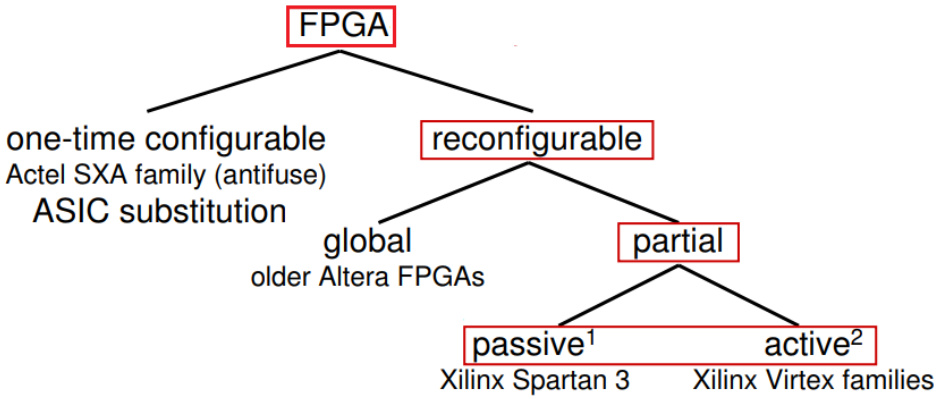



Figure 4.4: Classification of FPGAs by their configuration capabilities [12]

can seamlessly continue during the reconfiguration process.

### 4.2.2 DPR Technology

Because of the recent evolution of FPGA technology, during the system operation time, the designer can update/reconfigure a certain part of the internal structure of the FPGA without affecting the rest of the parts by using a mechanism known as Dynamic Partial Reconfiguration (DPR). This requires the designer to divide his design into 2 parts: Static part and dynamic part [1]. The dynamic part consists of a set of Reconfigurable Modules (RMs) which are swapped during runtime after being floor-planned onto a Reconfigurable Partition (RP). When it is desired to switch to a certain RM, its corresponding partial bit file is loaded at runtime without affecting the static part. A block diagram illustrating different parts of a DPR dependent design is shown in Fig. 4.5.

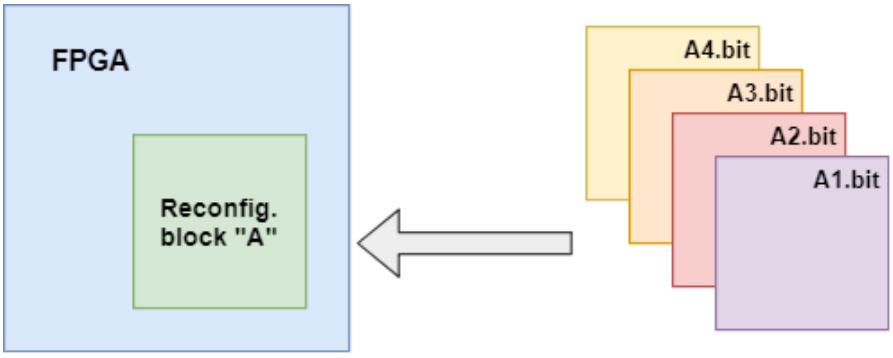



Figure 4.5: Basic structure of partial reconfiguration design

As shown in Fig.4.5, the function performed in reconfig Block A is changed by downloading one of many partial BIT files, A1.bit, A2.bit, A3.bit, or A4.bit.

The programmable logic(PL) in the FPGA design is splited into two independent types, reconfigurable logic and static logic. The gray region of the FPGA chunk represents static logic and the block part labeled Reconfig Block "A" represents reconfigurable logic.The static part remains functioning and is unchanged by the loading of a partial BIT file.The reconfigurable part is changed by the contents of the partial BIT file.

### 4.2.3 DPR Benefits

As shown in Fig.4.6 DPR has many advantages over traditional full configuration including [13]:

- **Reduce cost and size:** Partial reconfiguration allow the designers to reduce the area of their designs by dynamically time-multiplexing portions of the available hardware resources. The ability to load functions on an as-needed basis also lessens the amount of idle logic, thereby saving additional space.

- **System Flexibility:** Traditionally, to change the function of the device, the engineer have to make new placement and routing of the design, then download full bitstream into the device. In contrast, when using DPR, the designer needs only to place and route the modified function in context with the already-verified remainder of the design, after that download the new partial image to a device in the field. Furthermore, the designer can dynamically adds new functions while the system is running, improving system up-time. The new function can be plugged into the same area without need to redesign the system or move to a bigger device. Also there will be flexibility in choosing the protocols or algorithms available for an application.

- **Reduce power consumption:** One of the methods used to reduce static power is to simply utilize a device that is less in size. With DPR,
  basically time slice the FPGA and run portions of their design individually. consequently,the system requires a much smaller device or fewer devices because not every portion of the design is needed 100% of the time. DPR also has the potential to decrease operating power as well as static power. For instance, many functions must be able to run at a very rapid speed, but that

maximum performance might only be needed a little percentage of the time. To save power, engineers can utilize DPR to swap out a high performance design with a low power version of the same designinstead of designing exclusively for maximum performance. The engineer can then turn back to the high-performance design when the system requires it.

- **Provide Adaptive systems**
- **Improving fault tolerant/self-repairing systems**
- **Enabling new techniques in design security**

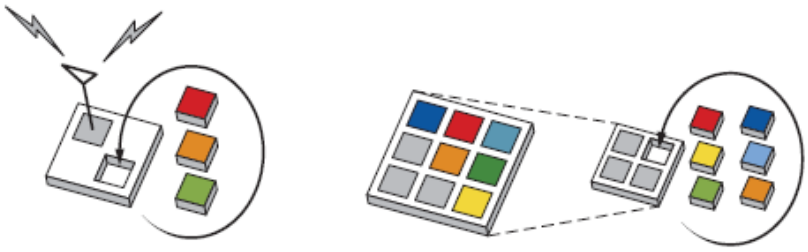


Figure 4.6: Modifying Functionality and Reducing Size using Partial Reconfiguration [13]

### 4.2.4 DPR Terminology

The following terminology is specific to the DPR technique [1] and is used throughout this thesis.

**Bottom-Up Synthesis**

Bottom-Up Synthesis is synthesis of the design by modules, whether in one project or multiple projects. Bottom-Up Synthesis requires that a separate netlist is written for each Partition, and no optimizations are done across these boundaries, ensuring that each portion of the design is synthesized independently. Top-level logic must be synthesized with black boxes for Partitions.

**Configuration**

A Configuration is a complete design that has one Reconfigurable Module for each Reconfigurable Partition. There might be many Configurations in a Partial Reconfiguration FPGA project. Each Configuration generates one full BIT file as well as one partial BIT file for each Reconfigurable Module.

**Configuration Frame**

Configuration frames are the smallest addressable segments of the FPGA configuration memory space. Reconfigurable frames are built from discrete numbers of these lowest-level elements. In a 7 series device, the base reconfigurable frames are one element (CLB, BRAM, DSP) wide by one clock region high.

**Partial Reconfiguration (PR)**

Partial Reconfiguration is modifying a subset of logic in an operating FPGA design by downloading a partial bitstream.

**Partition**

A Partition is a logical section of the design, user-defined at a hierarchical boundary, to be considered for design reuse. A Partition is either implemented as new or preserved from a previous implementation. A Partition that is preserved maintains not only identical functionality but also identical implementation.

**Partition Pin**

Partition pins are the logical and physical connection between static logic and reconfigurable logic. Partition pins are automatically created for all Reconfigurable Partition ports.

**Reconfigurable Frame**

Reconfigurable frames represent the smallest reconfigurable region within an FPGA. Bitstream sizes of reconfigurable frames vary depending on the types of logic contained within the frame.

**Reconfigurable Logic**

Reconfigurable Logic is any logical element that is part of a Reconfigurable Module. These logical elements are modified when a partial BIT file is loaded. Many types of logical components can be reconfigured such as LUTs, flip-flops, BRAM, and DSP blocks.

**Reconfigurable Module (RM)**

A Reconfigurable Module (RM) is the netlist or HDL description that is implemented within a Reconfigurable Partition. Multiple Reconfigurable Modules will exist for a Reconfigurable Partition.

**Reconfigurable Partition (RP)**

Reconfigurable Partition (Reconfigurable Partition (RP)) is an attribute set on an instantiation that defines the instance as reconfigurable. The Reconfigurable Partition is the level of hierarchy within which different Reconfigurable Modules are implemented. Tcl commands such as opt_design, place_design and route_design detect the HD.RECONFIGURABLE property on the instance and process it correctly.

**Static Logic**

Static logic is any logical element that is not part of a Reconfigurable Partition. The logical element is never partially reconfigured and is always active when Reconfigurable Partitions are being reconfigured. Static logic is also known as Top-level logic.

**Static Design**

The Static design is the part of the design that does not change during partial reconfiguration. The static design includes the top level and all modules not defined as reconfigurable. The static design is built with static logic and static routing.

### 4.2.5 Reconfigurable Elements

Some component types can be reconfigured and some cannot [1]. For 7 series devices, the component rules are as follows.

**Reconfigurable resources include:**

- Slice logic (LUTs, flip-flops, and carry logic, for example)
- Memories (block RAM, distributed RAM, shift register LUTs)
- and DSP component types as well as routing resources.

**Logic that must remain in static logic includes:**

- Clock-modifying blocks (MMCM, DCM, PLL, PMCD)
- Global clock buffers (BUFG)
- Device feature blocks (BSCAN, ICAP, STARTUP, or -PCIE, for example)

### 4.2.6 Managing Dynamic Device Reconfiguration

To configure the FPGA, engineers begin as normal by loading a full design bitstream upon power-up. After the chip is fully configured and operational, engineers can utilize partial bit files at any time to adapt the pre-defined regions while the rest of the FPGA remains completely active and uninterrupted [1]. The engineers can select from the following configuration ports to load the partial bit file:

**Externally:**

- Slave SelectMAP
- Slave Serial
- JTAG

**Internally:**

- Internal Configuration Access Port (ICAP) (ICAP, an internal representation of the SelectMAP interface)

Table 4.1 lists the recommended default reconfiguration speed of the different configuration interfaces for Xilinx Virtex-4 and later FPGAs.

Table 4.1: Configuration Speed for the Different Interfaces on Xilinx FPGAs [1].

| Configuration Mode | Max Clock Rate | Data Width | Max Bandwidth |
|---|---|---|---|
| SelectMap / ICAP | 100 MHz | 32-bit | 3.2 Gbps |
| Serial Mode | 100 MHz | 1-bit | 100 Mbps |
| JTAG | 66 MHz | 1-bit | 66 Mbps |

The designers of system can control the initiation of reconfiguration and download of the partial configuration image using a wide set of techniques, two of which are shown in Fig.4.7 system of the self-reconfigurable FPGA on the left in Fig.4.7 hires a common application in which a small microprocessor is utilized to read the partial bitfile out of the flash and forward the data to the internal configuration port (ICAP). Design on the right in figure uses the external processor to read the partial bitfile from the flash and send the data to the standard configuration port

of the FPGA and this method known as as an externally-reconfigurable FPGA [13].

Partial bitstreams holds all the configuration commands and information needed for partial reconfiguration. The task of download a partial bitstream into an FPGA does not need knowledge of the physical position of the reconfigurable module. Since configuration frame addressing information is involved in the partial bitstream, it cannot be sent to the wrong portion of the FPGA. However, the proposed design use the concept of the the left side of the Fig.4.7, which provide a self-reconfigurable FPGA.

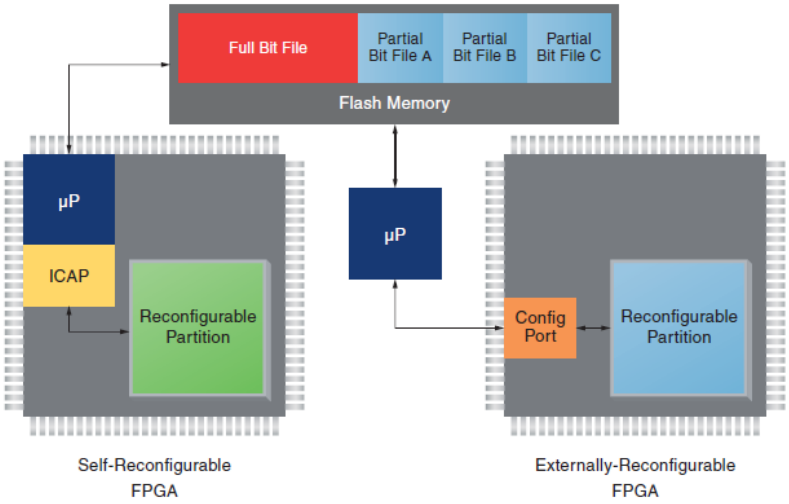



Figure 4.7: Two Methods of Delivering a Partial Bit File [13]

### 4.2.7 DPR controllers

Currently, there are several DPR controllers implemented as Intellectual Properties (IPs) and offered by Xilinx such as: HWICAP, PRC and PCAP [1]. However, it is possible to implement custom IP controllers such as ZYCAP [43]. A comparative study was done by [14] showing the performance of these controllers based on area utilization, power consumption and maximum throughput for a software defined radio encoder. As shown in Fig.4.8, it is clear that PRC consumed most resources while providing the highest throughput in comparison with Hardware ICAP (HWICAP ) which consumed much less resources while providing the least throughput.

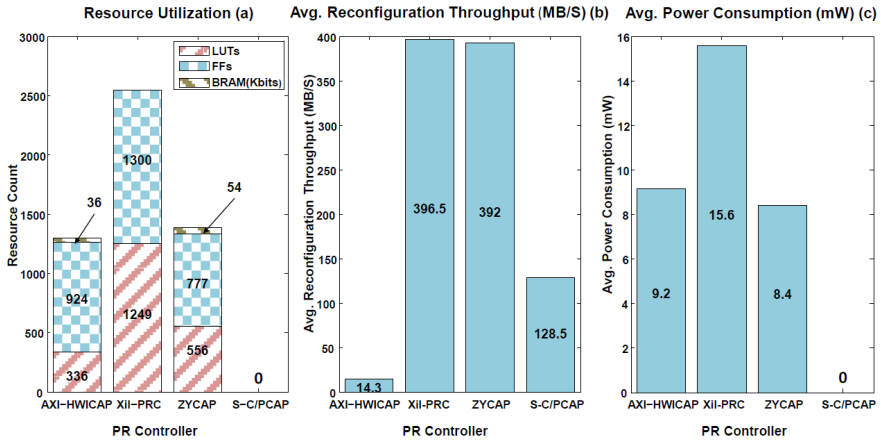



Figure 4.8: (a) Resource Utilization, (b) Avg. Reconfiguration Throughput and (c) Power Consumption Comparisons between Different PR Controllers [14].

### 4.2.8 Xilinx AXI-HWICAP controller

Xilinx provides many Intellectual Property (IP) cores to connect the ICAP module with the user design. These IP cores corresponds to partial reconfiguration controller that enables an embedded microprocessor such as ARM processors or Microblaze to access the configuration memory. ICAP is a xilinx predefined macro that has direct access to the configuration memory for both write and read modes [1] as depicted in Fig.4.9. CSB is the active low interface select signal, RDWRB is the read/write select signal. BUSY is valid only for read operations and remains low for write operations. In Xilinx 7-series FPGAs, the ratio of the ICAP interface data width to the configuration memory is 32 bit wide. The max theoretical reconfiguration throughput of ICAP is equal to 400 MB/S at a frequency of 100 MHz [15] . In practice, it was found that the measured throughput is much less than the theoretical one due to the addition of the reconfiguration overhead to the DPR at the system level.

AXI-HWICAP [15] is an ICAP controller designed for AXI bus interfaces where it is connected as a slave peripheral. During DPR, the partial bitstream data are buffered from an external memory to a write/read FIFOs inside the core where there is a finite state machine that monitors the status of the FIFOs and supplies partial bitstream data to the ICAP and then to the configuration memory Fig.4.10. Based on the study in [14], AXI-HWICAP is the optimum choice as a controller in the proposed design despite the limitations on the maximum throughput which

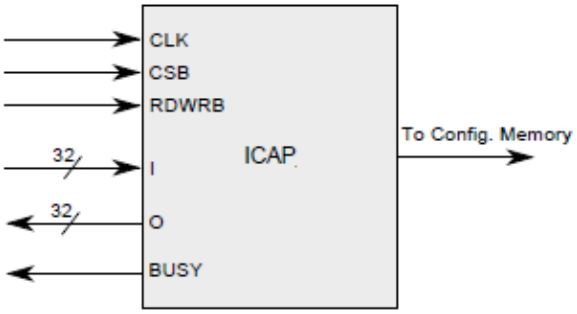



Figure 4.9: Xilinx ICAP Primitive.

is not an issue for IoT constrained devices that requires low area and power requirements. Another reason to choose the AXI-HWICAP controller is to make the proposed design more generic, since the AXI-HWICAP is allocated in the PL side whereas the PCAP is located in the PS side, then to use PCAP controller that is means that we need a chip that has hard core ( which is an ARM core in our proposed design), in contrast using AXI-HWICAP provides the ability of implement the proposed design in any FPGA without need of the hard-core.

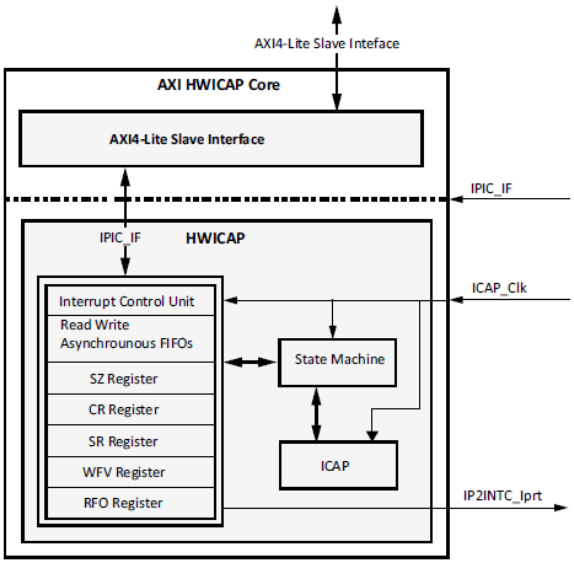



Figure 4.10: Top Level Block Diagram for the AXI HWICAP Core [15]

## 4.3 Proposed design

### 4.3.1 Design modules

As shown in Fig.4.11, the design is divided into 2 parts: Encryption module and decryption module. Each module is loaded onto an FPGA and accordingly, each module has 2 parts: Static part and dynamic part. It is important to note that both parts include the same hardware modules except for Linear Feedback Shift Register (LFSR) which is only needed for the static part of the encryption module. The static part for both modules consists of the following components:

- First In First Out (FIFOs) for inputs and outputs.
- AEAD top.

The dynamic part for both modules consists of one reconfigurable region for the cipher modules and each cipher module consist of the following components:

- Pre processor.
- Cipher core.
- Post processor.

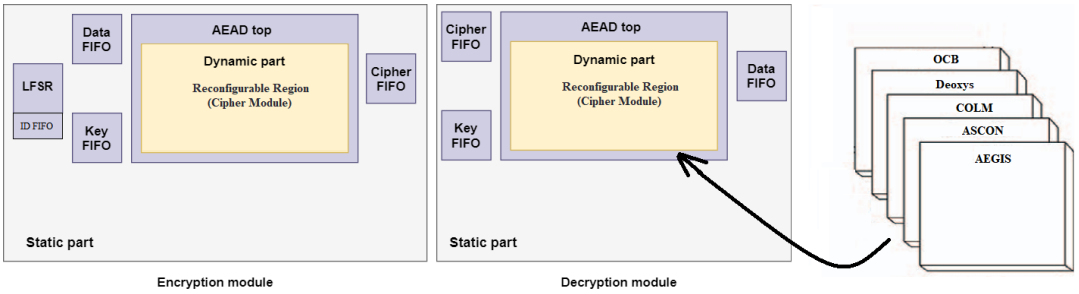



Figure 4.11: Block Diagram of the Proposed Design (PL Side)

The functionality of each component is explained as follows:

#### 4.3.1.1 AEAD top

As shown in Fig.4.12, the AEAD interface consists of 3 main data buses:

- Public Data Inputs (PDI)

- Secret Data Inputs (SDI)
- Data Outputs (DO)

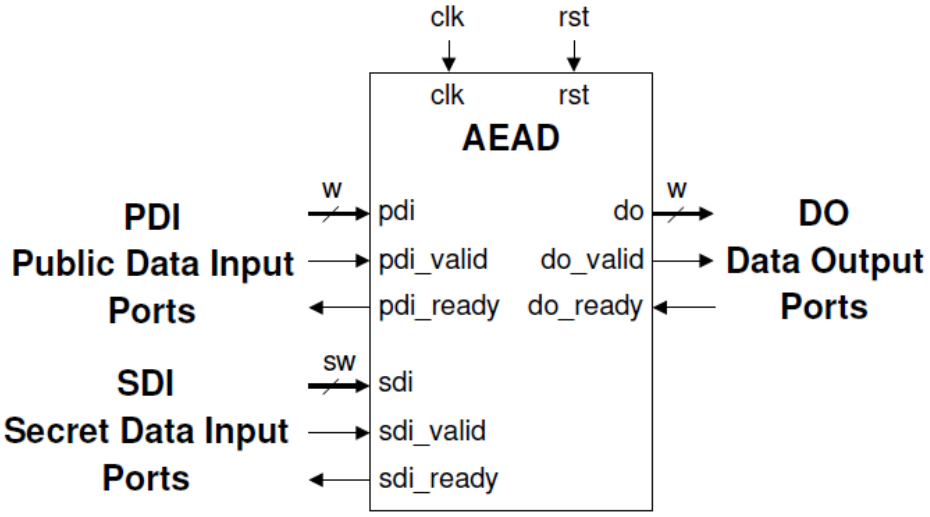



Figure 4.12: AEAD interface [11]

In addition, there are the corresponding control signals such as: valid and ready. The valid signal indicates that the source is ready to send data while the ready signal indicates that the destination is ready to receive them. The Generic parameters for AEAD chosen as : Public data input width (W) = 256, Secret data input width (SW) = 32, Output data width W = 256.

All possible inputs and outputs of the AEAD are illustrated in Fig.4.13. AD denotes Associated Data, Npub denotes Public Message Number, such as Number used once (Nonce) or Initialization Vector. Nsec denotes Secret Message Number, which was recently introduced in some authenticated ciphers. Both Npub and Nsec are expected to be unique for each message encrypted using a given key. The difference is that Npub is directly buffered to the other side, while Nsec is sent in the encrypted form. Secret Data Inputs (SDI) port is responsible for processing the secret key while Puplic Data Inputs (PDI) port handles the rest of the inputs.

The API is also composed of several generic parameters such as: Key length, data block size, SDI port width, PDI port width and type of data padding which are modified based on the requirements of each cipher. The main idea of the AEAD top is to modify the API by choosing the largest of the generic parameters of the 5 ciphers and grouping them in 1 static port from which each dynamic module's parameters are modified.

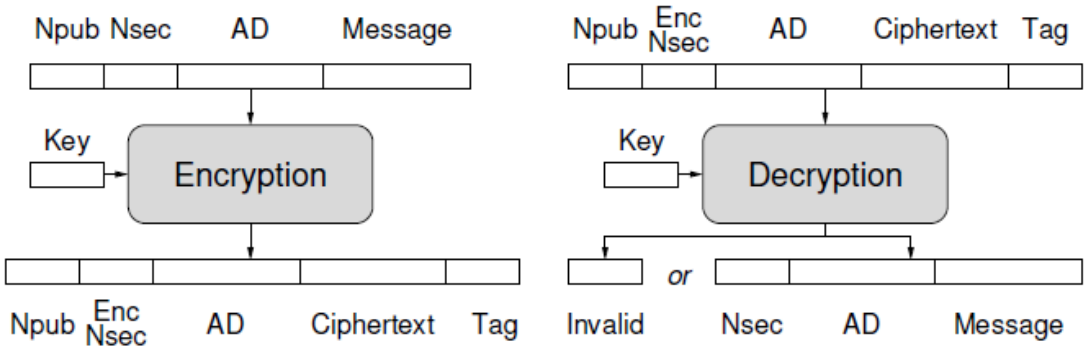



Figure 4.13: Inputs and outputs of AEAD [11]

#### 4.3.1.2 LFSR

The total number of flip flops used depends on the number of bits needed for the design. Since there are 5 ciphers to switch between, only 3 bits are needed and consequently, 3 flip flops are used in the LFSR. As shown in Fig.4.14 , the taps are at bit 0 and bit 2. All of the register elements share a common clock input, which is omitted from the symbol for reasons of clarity. The data input to the LFSR is generated by XOR-ing the tap bits, while the remaining bits function as a standard shift register. The pseudo-random pattern is generated based on a seed input to the LFSR that is only known to the sender on the encryption side to start the hopping sequence. The main reason for not using the LFSR in the decryption side is that the next step in the hopping sequence is sent prior to decryption from the encryption side.

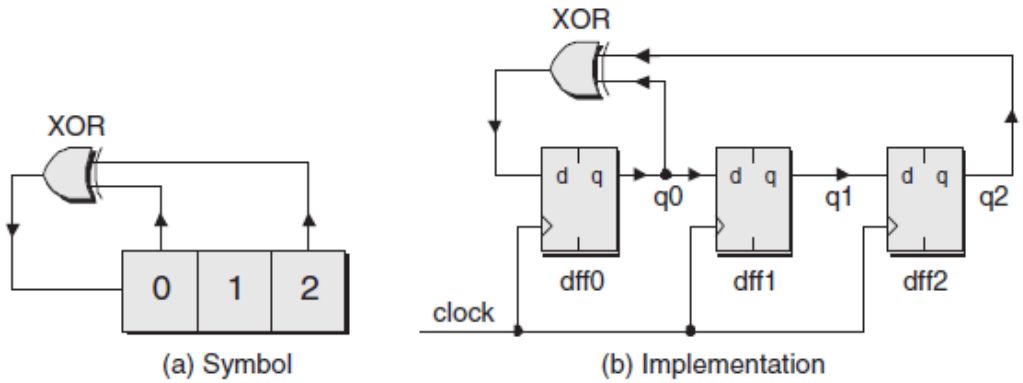



Figure 4.14: 3 bits LFSR with XOR feedback path

The table 4.2 shows the generated values from the LFSR when the seed value is ”001” and the corresponding algorithm for each.

Table 4.2: Patterns Which generated from LFSR.

| Corresponding Algorithm | Clock | q0 | q1 | q2 |
|---|---|---|---|---|
| Initial Value (seed) | .... | 0 | 0 | 1 |
| AEGIS | 1 | 1 | 0 | 0 |
| ASCON | 2 | 0 | 1 | 0 |
| COLM | 3 | 1 | 0 | 1 |
| Deoxys | 4 | 1 | 1 | 0 |
| OCB | 5 | 1 | 1 | 1 |
| Skip | 6 | 0 | 1 | 1 |
| Skip | 7 | 0 | 0 | 1 |
| AEGIS | 8 | 1 | 0 | 0 |
| : | : | : | : | : |

**Seeding the LFSR:**

One quirk with LFSRs that depending on XOR is that, if one happens to discover itself in the all-0s values, it will happily continue to shift all 0s indefinitely ( similarity for XNOR-based LFSRs and the all-1s value ). This is of special concern when power is first applied to the circuit. each register bit can randomly launch containing either a logic 0 or a logic 1, and the LFSR can therefore "wake up" containing its " forbidden" value . Therefore, it is necessary to launch an LFSR with a seed value.

However, the proposed design includes a multiplexer at the input of the LFSR for loading a seed value as shown in Fig.4.15 When the control signal is putted in its active state, the LFSR will load with a hard-wired seed value. After loading the seed value, the feedback path is chosen and the device returns to its LFSR mode of operation. In addition, we can at any time to load a new seed by select the seed-data path to change the sequence of patterns that generated from the LFSR.

#### 4.3.1.3 Crossing Clock Domains Using FIFOs

A clock domain crossing exists whenever information is transmitted from a component driven by one clock to a component driven by different clock [16] as shown in Fig. 4.16

In the proposed design there are two parts, PS and PL. The operating frequency of PL is 10 MHz while the PS operates at a frequency of 667.66 MHz. The test vectors (Secret input data/Public input data ) which are initially stored in fixed

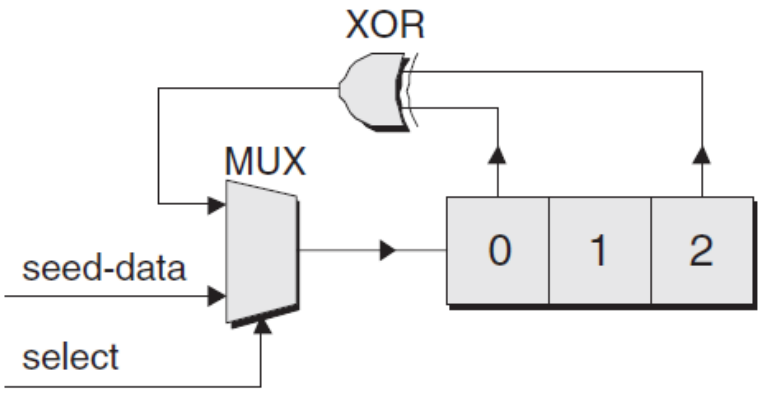



Figure 4.15: Circuit for loading seed values

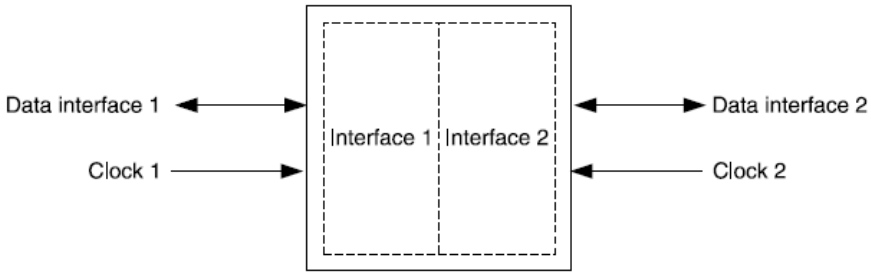



Figure 4.16: Clock Domain Crossing [16]

arrays in the C code should transfer to PL side to process (encryption/decryption) them and then the outcome (decrypted message/cipher-text) will be re-sent to the PS to transmit it to the destination (another IoT node ). This means that the data will transfer from one clock domain to another one. consequently, some problems will appear related to the different clocks between PS and PL. Therefore, 3 FIFOs used to solve this issue (Data FIFO, Key FIFO, Cipher FIFO ). The functionality of the 3 FIFOs used in the encryption and decryption modules is to allow for reading and storing inputs/outputs from external text files that contain the test vectors for each cipher. These FIFOs have constant input/output port size based on the generic parameters initialized in the AEAD top module. The width of Data/Cipher FIFO is 256 bits, while the width of key FIFO is 32 bits as shown in Fig.4.17.

However, the LFSR circuit controlled by the PS using "Enable signal" and the output of the LFSR "cipher ID" will be sent to the PS side to make the transformation of the bit file of the corresponding cipher. In addition, from the ARM core we can change the value of the seed to change the sequence of patterns

at any time. Therefore, we added a FIFO "ID-FIFO" to store the output of the LFSR, then send it to the PS and when we want to change the sequence of the patterns, we use the same FIFO to write in.

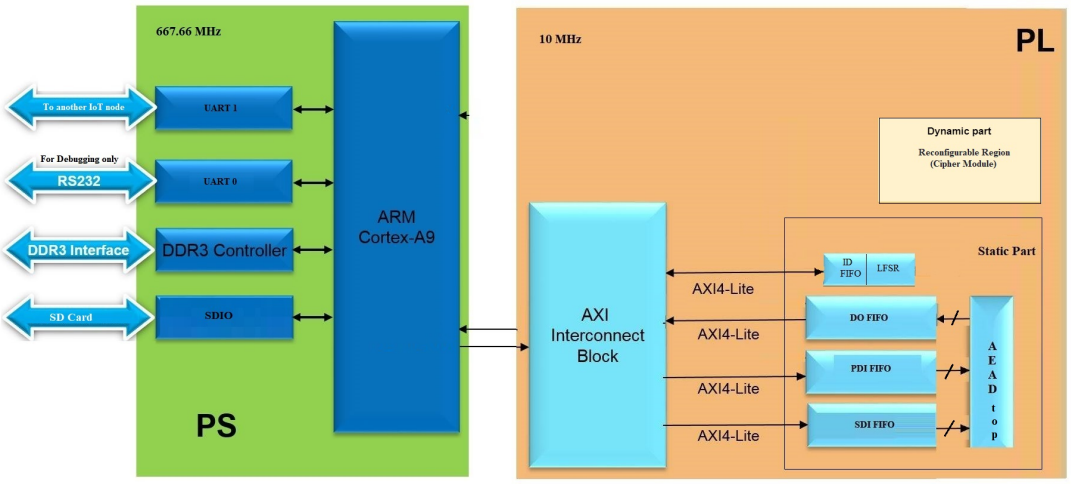



Figure 4.17: Full Block Diagram of the Proposed Design ( PL + PS)

#### 4.3.1.4 Pre processor

The first component in the dynamic part is the pre-processor which is responsible for the following tasks common to the 5 ciphers:

- Loading and activating keys
- Serial-In-Parallel-Out loading of input blocks
- Padding input blocks
- Keeping track of the number of data bytes left to process

The preprocessor is implemented in the API as an interface for processing the input data after modifying its input ports and parameters specific to each cipher in order to provide the cipher core with the appropriate data.

#### 4.3.1.5 Post processor

The second component in the dynamic part is the post processor which is responsible for the following tasks common to the 5 ciphers:

- Clearing any portions of output blocks not belonging to cipher or plaintext

- Parallel-In-Serial-Out conversion of output blocks into words
- Formatting output words into segments
- Storing decrypted messages until the result of authentication is known
- Generating the status block with the result of authentication

The post processor included in the API is modified by adjusting the output ports and parameters specific to each cipher to ensure correct data is stored in the FIFOs.

### 4.3.2 Design flow

The Dynamic Partial Reconfiguration (DPR) flow is carried out using Xilinx Vivado tool. A complete design for the encryption module is shown in Fig.4.18. A similar design is tailored for the decryption module with the same components. The system is controlled by the ARM Cortex A9 microprocessor on the Processing System (PS) side. The processor communicates with the Programmable Logic (PL) side through the AXI bus interface. The static part consists of: the AXI HWICAP, the AXI connections, AEAD top, FIFOs in addition to the LFSR in the encryption side. The dynamic part contains Reconfigurable Partition (RP) that holds the 5 Reconfigurable Modules (RMs) corresponding to the 5 ciphers. The partial bitstreams for these RMs are stored in the SD card to be loaded through a software code that interfaces the PS with the Personal Computer (PC) via UART connection. The operating frequency of PL is 10 MHz while the PS operates at a frequency of 667.66 MHz.

A complete flow of a full encryption/decryption operation is described as follows:

i. The sender starts the encryption module by entering the seed into the LFSR from the PS side.

ii. The LFSR generates the first ID that corresponds to a specific cipher.

iii. The processing system transfers the bitstream of the corresponding cipher to the AXI-HWICAP which writes it into the configuration memory.

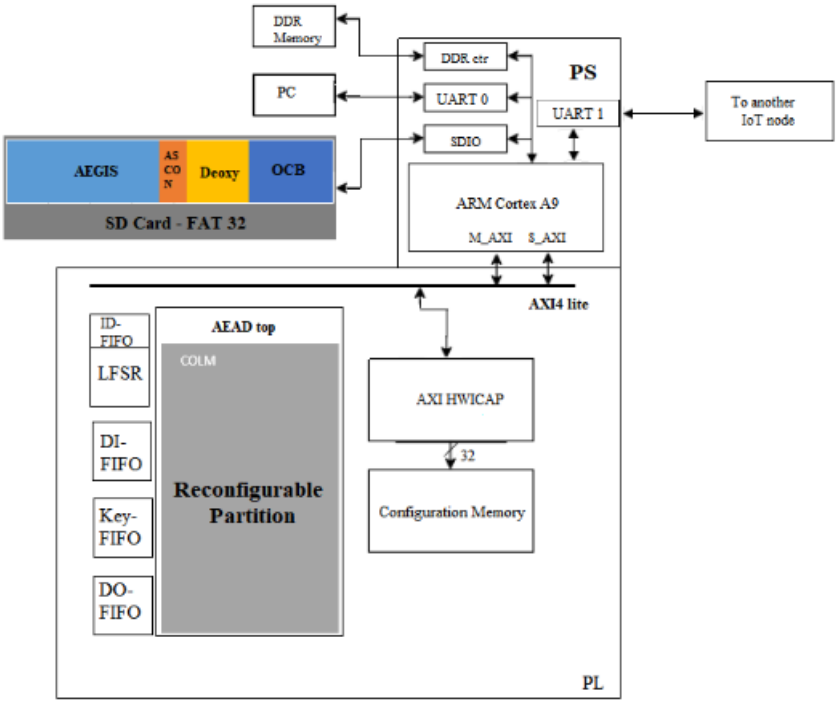



Figure 4.18: Hardware design of the encryption module

iv. The PS starts to send the PDI,SDI to the corresponding FIFOs, after that the AEAD top starts to read the data from these FIFOs (PDI FIFO, SDI FIFO).

v. The output data is stored into the output FIFO until the encryption is done.

vi. When done, the PS read the output FIFO.

vii. The PS sends the enable signal along with the ID of the cipher to the decryption side.

viii. The decryption module switches to the corresponding cipher and starts receiving the data encrypted through the (UART 1) inside the PS.

ix. When the decryption is done, a status message is sent back to the encryption module to mark the success of the full operation and allow for the next hop.

# Chapter 5

# System Implementation and Results

## 5.1 Requirements

**Software Tools:**

- Vivado Design Suite 2015.2

  The DPR feature needs a license, and since the available license for us is only 2015.2, then we select the Vivado Design Suite 2015.2 to implement our proposed design.

- Xilinx Software Development Kit 2015.2

- Terminal program (Teraterm)

**Hardware Tools:**

- ZC02 board (Zynq Evaluation and Development).

- SD Card (FAT32)

**Licensing:**

- Xilinx Partial Reconfiguration

## 5.2 DPR design flow

**Step 1: Partition the system into modules**

The system has many modules: the AXI HWICAP, the AXI connections, AEAD top, LFSR, FIFOs, CAESAR AEAD algorithms ( AEGIS, ASCON, COLM, Deoxys, OCB ).

**Step 2: Define static modules and reconfigurable modules**

- **Static:** AXI HWICAP, the AXI connections, AEAD top, LFSR, FIFOs.
- **Dynamic:** CAESAR AEAD algorithms ( AEGIS, ASCON, COLM, Deoxys, OCB ).

**Step 3: Decide the number of PR regions (PRRs)**

The system needs only one PRR.

**Step 4: Generate Design Check Point (DCP) for Static and RM modules**

Vivado tool is used to synthesize the static and all the RMs modes in all given DPR design configurations. The RMs modes are given to the synthesis tool as a set of HDL files to determine the numbers and types of resource requirements on the Field Programmable Gate Array (FPGA) such as (LUTs, BRAM, and DSP blocks). The output from the synthesis step is: DCP ( Design Check Point file contains the netlist of the design supported by Xilinx Vivado Design suit ) file. Fig.5.1 shows the DCP files for the RMs modules, whereas Fig.5.2 shows the DCP file for the static module.

**Step 5: Load Static and one RM for the RP** The target of this step is floorplanning, this step is done manually by the designer by arranging the set of RRs with the static partition into rectangular shapes on the FPGA floorplan.

- Load the static design using the open_checkpoint command (In the Tcl Shell window of Vivado).

  " open_checkpoint Synth/Static/system_wrapper.dcp ". As shown in Fig.5.3

- Load one RM for the RP by using the read_checkpoint command:

  " read_checkpoint -cell system_i/CAESAR_0/U0/CAESAR_v1_S_AXI_inst/

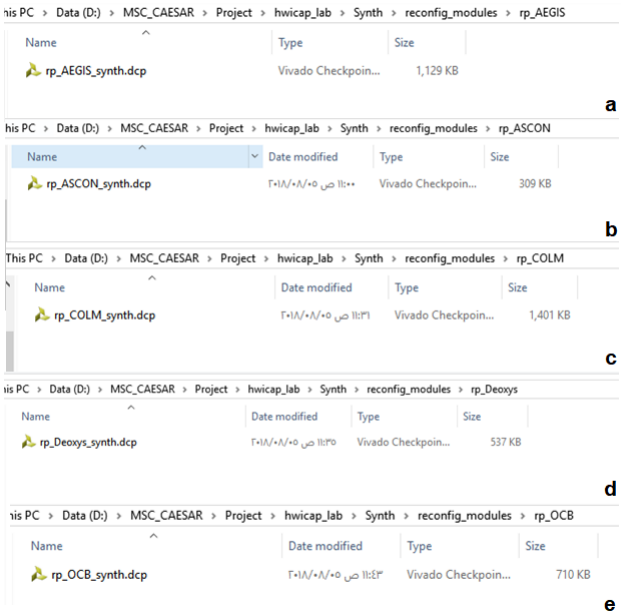



Figure 5.1: DCP files for all RMs

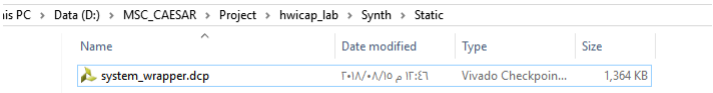



Figure 5.2: DCP file for the static module

rp_instance Synth/reconfig_modules/rp_AEGIS/rp_AEGIS_synth.dcp ". As shown in Fig. 5.4

- Define each of the loaded RMs (submodules) as partially reconfigurable by setting the HD.RECONFIGURABLE property using the following commands.
  " set_property HD.RECONFIGURABLE 1 [get_cells system_i/CAESAR_0/U0/CAESAR_v1_0_S00_AXI_inst/rp_instance] "

- Save the assembled design state for this initial configuration using the following command.
  " write_checkpoint Checkpoint/initial_config.dcp "

- Create floor-plan which defines the RP region and sets the Pblock property RESET_AFTER_RECONFIG=TRUE on the Pblock, and turns ON the SNAPPING_ON property. As shown in Fig.5.5, 5.6

**Step 6: Create and Implement First Configuration**

The target of this step is the implementation or (Place and Route), and this

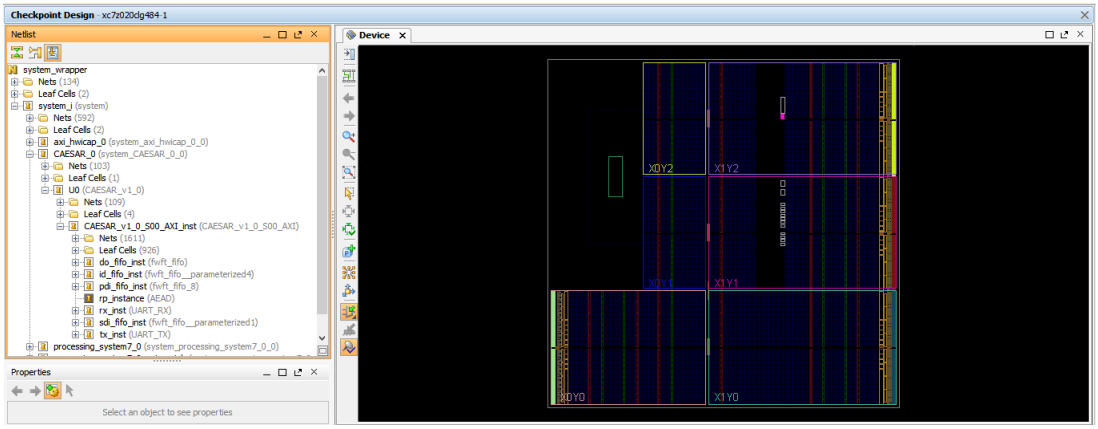



Figure 5.3: Black box in the static design

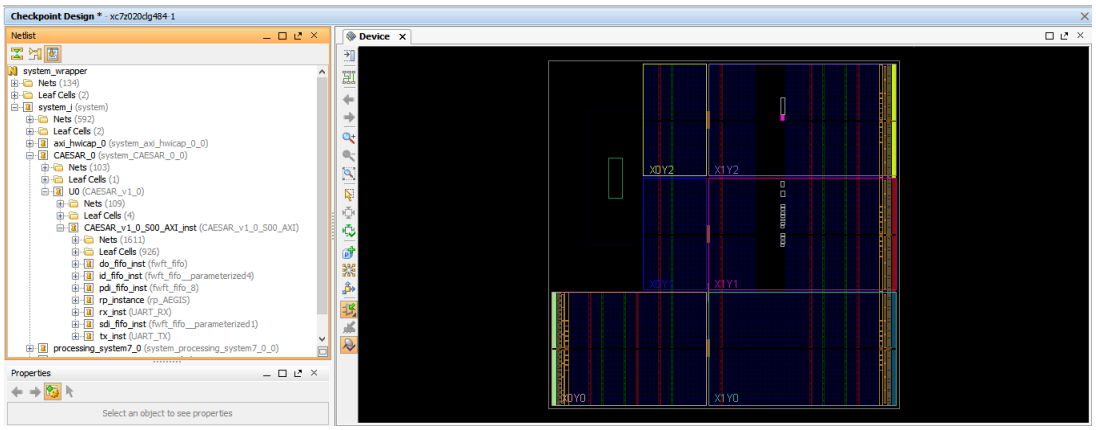



Figure 5.4: Load one RM for the RP

step is done automatically by the tool using the design netlists and the (area, time and floorplanning) constraints generated from the above steps to map the design on the FPGA device. This step includes optimize, place and route the design by executing the following commands:

opt_design, place_design, route_design ” in the TCL window ”

Then, Save the full design checkpoint, and finally, Save checkpoints for the reconfigurable module (ASCON) .

**Step 7: Create Other Configurations**

In this step we had read the next set of RM DCPs, crate and implement the rest of the configurations which are ” ASCON, COLM, Deoxys, OCB ” since we were implemented the first RM ” ASCON ” in previous step.

**Step 8: Run PR Verify**

PR Verify is run to make sure that the design doesn’t have any errors.

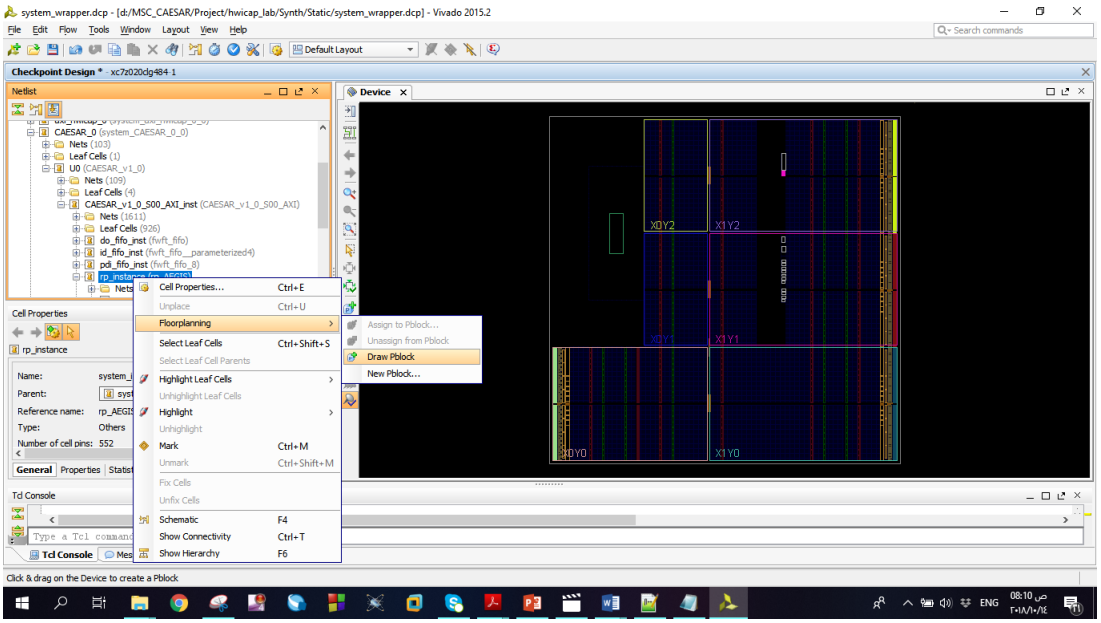



Figure 5.5: Create floor-plan which defines the RP region (a)

**Step 9: Generate Bit Files**

Finally, a full and partial bitstream sets are generated for different configurations and partial blocks (RMs) in the design. The FPGA target is initially configured by a full bitstream (A complete configuration image) and for runtime reconfiguration, partial bitstreams of the required RMs are loaded from an external memory. The full and partial bitstream is generated as shown in Fig.5.7.

**Step 10: Generate Software Application**

Run the SDK and write the code that controls the arm to test the design.

**Step 11: Test the Design**

Run the code and make sure that the design is correct.

**Step 12: Generate Bit Files**

In case of errors, correct the design and regenerate bitstreams.

**Step 13: Generate Software Application**

Edit the code and make sure that the design is correct.

**Step 14: Test the Design**

Run the code and make sure that the design is correct.

Fig.5.8 shows all steps of the DPR design flow.

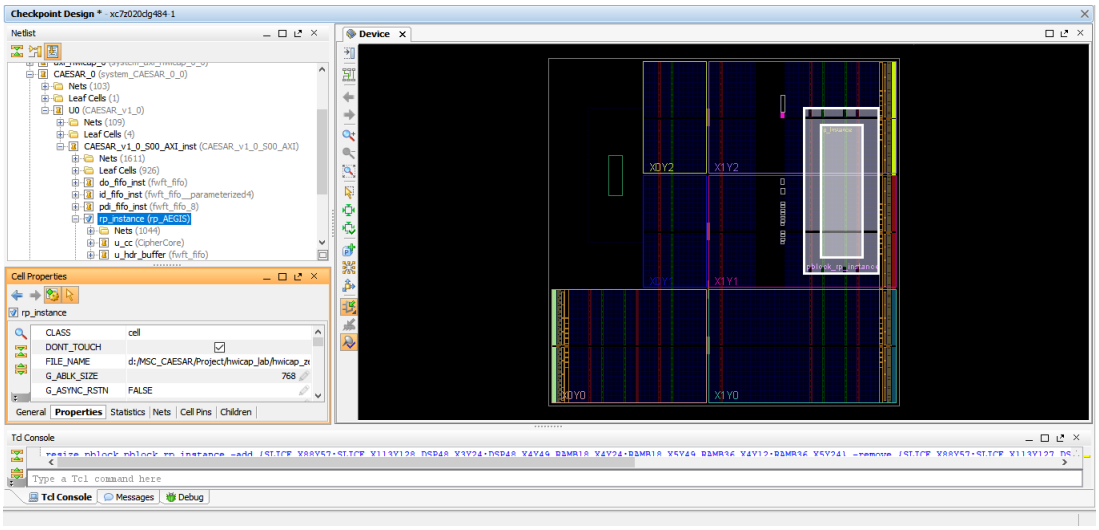



Figure 5.6: Create floor-plan which defines the RP region (b)

## 5.3 System Block Design

The full block design of the system is shown in Fig.5.9. Based on the study in [14], AXI-HWICAP is the optimum choice as a controller in the proposed design despite the limitations on the maximum throughput which is of second priority for IoT constrained devices that require low area and power requirements. The PDR triggers (signals that trigger the configuration of certain RMs) came from a program that I built running on the ARM processor. CAESAR_0 is the basic design instance and axi_hwicap_0 is the HWICAP instance. They are both connected to processing system7_0 which is the ARM processor through the AXI interconnect.

In Fig.5.10, a close up of CAESAR_0 (my design) within the block design is shown.

## 5.4 Implementation Results

### 5.4.1 Resource Utilization

As shown in Table 5.1, the resource utilization of each configuration is calculated for a Xilinx XC7Z020LG484-1 Zynq FPGA [44] after synthesis using Xilinx Vivado 2015.2. The 5 configurations correspond to the 5 ciphers: AEGIS, ASCON, COLM, Deoxys and OCB respectively. It is important to note that these results are based on the original implementations by the designers of the ciphers. There are no modifications applied to optimize for lower resources utilization. The main

| Name | Date modified | Type | Size |
|---|---|---|---|
| AEGIS.bin | ٢٠١٨/١٠/٢٥ م ٠٤:٤٧ | BIN File | 654 KB |
| AEGIS.prm | ٢٠١٨/١٠/٢٥ م ٠٤:٤٧ | PRM File | 1 KB |
| ASCON.bin | ٢٠١٨/١٠/٢٥ م ٠٤:٤٩ | BIN File | 654 KB |
| ASCON.prm | ٢٠١٨/١٠/٢٥ م ٠٤:٤٩ | PRM File | 1 KB |
| COLM.bin | ٢٠١٨/١٠/٢٥ م ٠٤:٥١ | BIN File | 654 KB |
| COLM.prm | ٢٠١٨/١٠/٢٥ م ٠٤:٥١ | PRM File | 1 KB |
| Config_AEGIS.bit | ٢٠١٨/١٠/٢٥ م ٠٤:٤٦ | BIT File | 3,951 KB |
| Config_AEGIS_pblock_rp_instance_partial... | ٢٠١٨/١٠/٢٥ م ٠٤:٤٧ | BIT File | 654 KB |
| Config_ASCON.bit | ٢٠١٨/١٠/٢٥ م ٠٤:٤٨ | BIT File | 3,951 KB |
| Config_ASCON_pblock_rp_instance_parti... | ٢٠١٨/١٠/٢٥ م ٠٤:٤٩ | BIT File | 654 KB |
| Config_COLM.bit | ٢٠١٨/١٠/٢٥ م ٠٤:٥٠ | BIT File | 3,951 KB |
| Config_COLM_pblock_rp_instance_partia... | ٢٠١٨/١٠/٢٥ م ٠٤:٥١ | BIT File | 654 KB |
| Config_Deoxys.bit | ٢٠١٨/١٠/٢٥ م ٠٤:٥٢ | BIT File | 3,951 KB |
| Config_Deoxys_pblock_rp_instance_parti... | ٢٠١٨/١٠/٢٥ م ٠٤:٥٣ | BIT File | 654 KB |
| Config_OCB.bit | ٢٠١٨/١٠/٢٥ م ٠٤:٥٤ | BIT File | 3,951 KB |
| Config_OCB_pblock_rp_instance_partial.bit | ٢٠١٨/١٠/٢٥ م ٠٤:٥٥ | BIT File | 654 KB |
| Deoxys.bin | ٢٠١٨/١٠/٢٥ م ٠٤:٥٣ | BIN File | 654 KB |
| Deoxys.prm | ٢٠١٨/١٠/٢٥ م ٠٤:٥٣ | PRM File | 1 KB |
| OCB.bin | ٢٠١٨/١٠/٢٥ م ٠٤:٥٥ | BIN File | 654 KB |
| OCB.prm | ٢٠١٨/١٠/٢٥ م ٠٤:٥٥ | PRM File | 1 KB |

Figure 5.7: Generated Bit Files

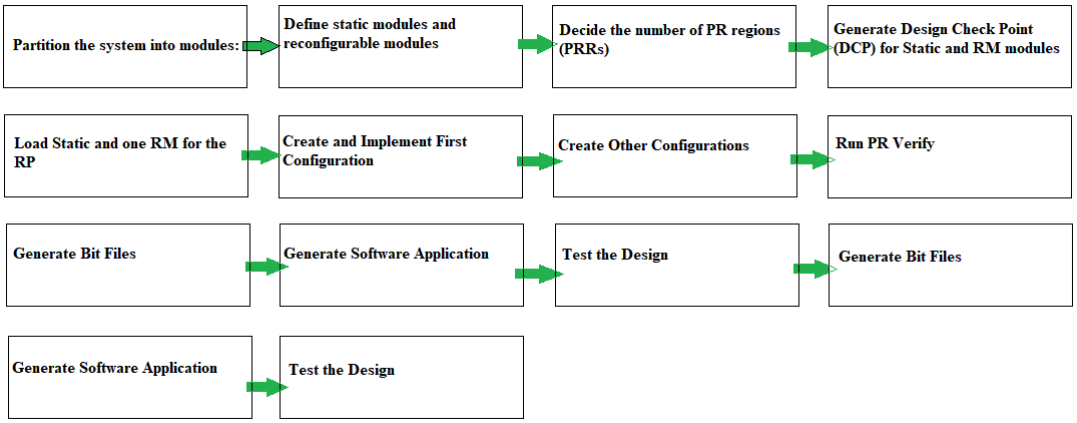



Figure 5.8: DPR Design Flow

purpose is to prove the flexibility of the proposed design to embrace any configuration.

Table 5.2 presents the resources utilized by the static components of the design including the inputs/outputs FIFOs and the LFSR. PDI FIFOs exhibits the largest area with 801 slice LUTs, 277 slice registers. This is because these FIFOs have largest input/output port width of 256 bits and a depth of 32 registers.

By integrating the utilization results for the dynamic and static parts, it is found that DPR helps in decreasing the utilization by 58% when compared to using a traditional design with the 5 configurations physically mounted onto the

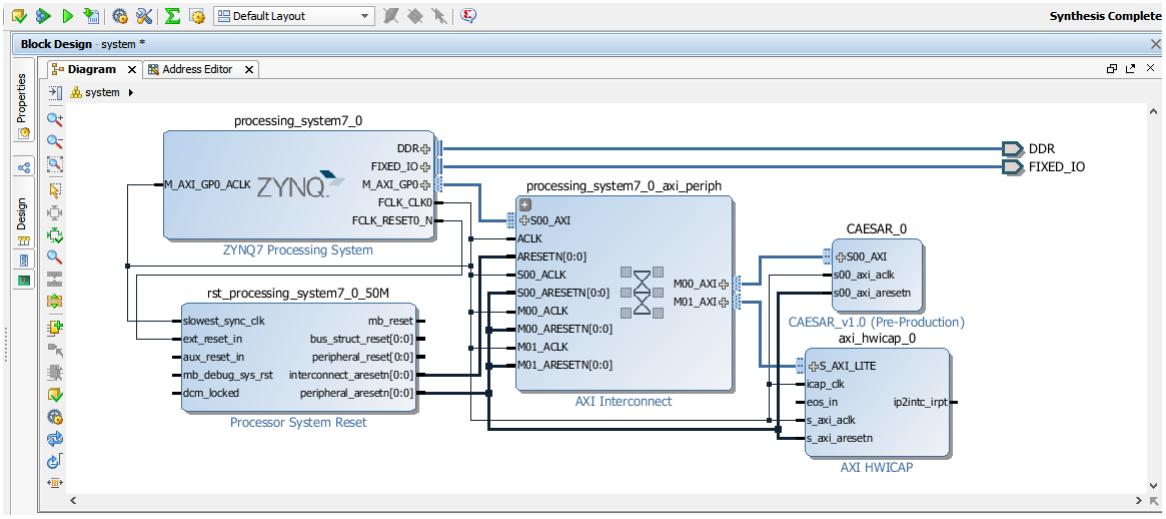



Figure 5.9: System Block Design

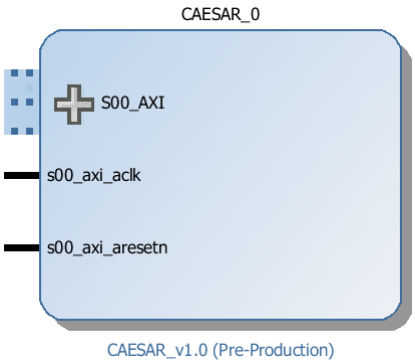



Figure 5.10: Encryption Module

FPGA at the same time without switching. These results apply to both encryption and decryption modules as they are using the same components except for the LFSR which is amortized over the whole design area.

### 5.4.2 Power consumption

In Table 5.3, dynamic power consumption is calculated for each configuration for both encryption and decryption modules at an operating frequency of 10 MHz using Xilinx Vivado 2015.2.

Table 5.1: Resources utilization of the dynamic configurations

| Configuration | Slice LUTs | Slice registers | F7 MUXs | F8 MUXs | BRAM tiles |
|---|---|---|---|---|---|
| AEGIS | 7250 | 2117 | 2052 | 1024 | 0 |
| ASCON | 1321 | 890 | 2 | 0 | 0 |
| COLM | 7547 | 2686 | 1137 | 376 | 4 |
| Deoxys | 2971 | 1593 | 641 | 256 | 0 |
| OCB | 4009 | 1612 | 593 | 200 | 4 |

Table 5.2: Resource utilization of the static modules

| Component | Slice LUTs | Slice registers | F7 MUXs | F8 MUXs | BRAM tiles |
|---|---|---|---|---|---|
| LFSR | 4 | 3 | 0 | 0 | 0 |
| DO/PDI FIFOs | 801 | 277 | 0 | 0 | 0 |
| SDI FIFO | 126 | 45 | 0 | 0 | 0 |

Table 5.3: Dynamic power consumption of the 5 configurations

| Configuration | Dynamic Power Consumption (mW) |
|---|---|
| AEGIS | 9 |
| ASCON | 2 |
| COLM | 11 |
| Deoxys | 6 |
| OCB | 4 |

With an average consumption of 6.4 mW, the proposed design allows for 80% reduction in power when compared with a conventional design using the 5 configurations in parallel without exploiting the capabilities of DPR. Figure 5.11 shows the Dynamic power consumption of each RM.

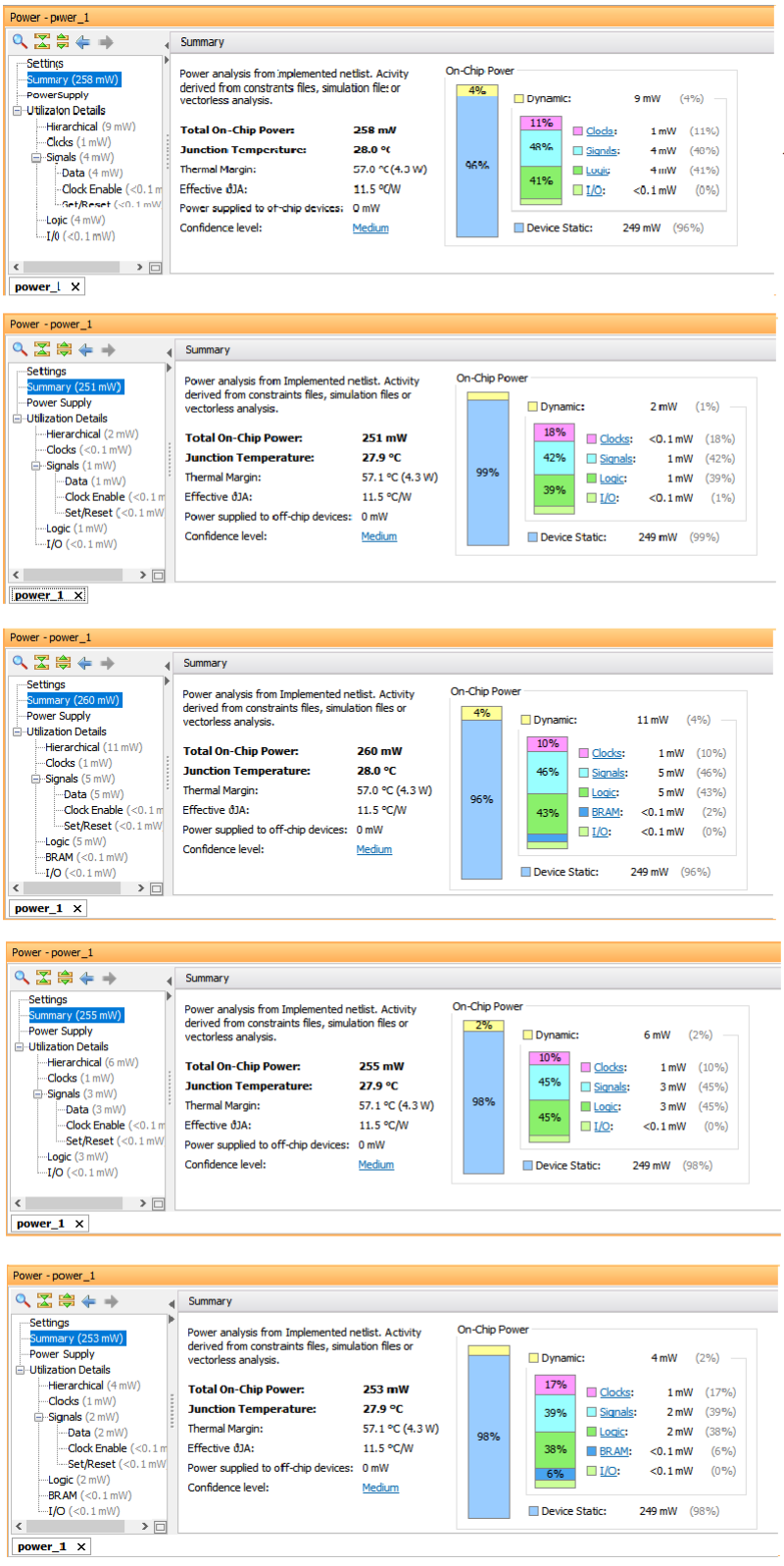



Figure 5.11: The power consumption of the RMs

### 5.4.3 Reconfiguration time

The speed of configuration is directly related to the size of the partial bit file and the bandwidth of the configuration port. In general, the reconfiguration time can be calculated by dividing the size of the bitstream (in bits) by the throughput of the ICAP [45]. The calculated configuration time is 1.67 ms for the 5 configurations as they are using the same RP with fixed area on the FPGA. Although the reconfiguration time is the main drawback of DPR, it is compensated by allowing the ARM processor to do other tasks during this time which helps in increasing the speed of any reconfigurable design.

## 5.5 Testing and Verification

The RTL codes and their test vectors folder for all algorithms have been downloaded from [33]. The AEAD top,LFSR, and FIFOs are written in VHDL .The test vectors folder consist of 3 text files: PDI (Public data Input), SDI (Secret Data Input ) and DO ( data output ). PDI contains many messages, and each message has its own ID. And each PDI message has some instructions and information to help the pre-processor to know if the process is encryption or decryption, the length of the plain text and associated data, the plain text and associated data. SDI contains many messages, and each message related to the unique message in the PDI file, and each message has instructions and information to help the preprocessor.The PDI message and the related SDI message (I mean the two messages have the same ID ) will be the input for the AEAD, and the output will be the message that has the same ID inside the DO file.

However, The first challenge was to convert each CAESAR AEAD algorithm from the VHDL code to the Block IP and add the required elements to achieve the communication between the Block IP and the ARM processor.This is done by ”create and package custom IP” feature inside Vivado, and for more details about how to create a custom block IP see [46].

We choose one of the 5 algorithms which is ”AEGIS” and converted it to block IP. Consequently, In this step we have a block design system which mainly consist of 2 components: ZYNQ7 processing System (PS) and AEGIS_ZED_REPO_IP IP (the AEGIS algorithm after converted to block level) as shown in Fig.5.12.

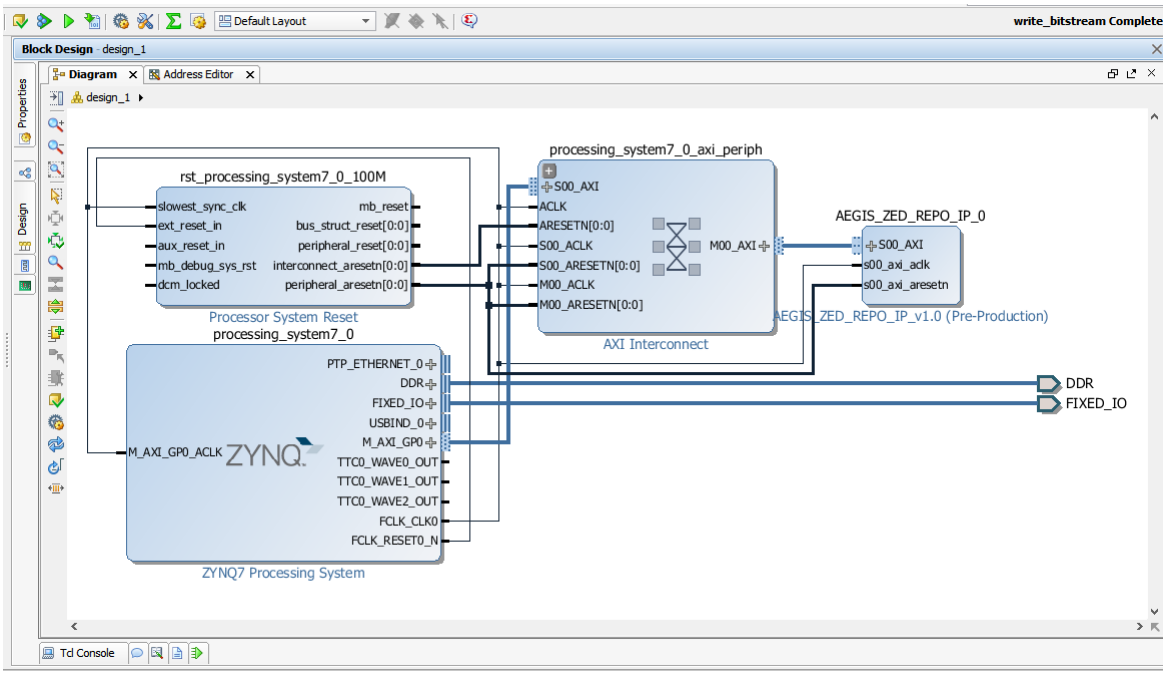



Figure 5.12: AEGIS IP in Block Design level

The communication between these blocks is AXI4-Lite protocol. AXI4-Lite is memory-mapped, therefore, we send the address of the register that we want to write into, then the value that we need to write. We wrote a C code using Software Development Kit (SDK) (Xilinx Eclipse) to write and read from the registers that related to the block IP.

However, we chose one message from the PDI file, and the corresponding message ( has the same ID ) from the SDI file, after that we write them into the corresponding registers of the Block IP (AEGIS_IP). Subsequently, we read the registers that related to the Data Outputs (DO) of the AEAD, and check if the output is the same of the corresponding message (message that has the same ID of the input message ) inside the DO file. Unfortunately, the data that we read was not matched with the correct value inside the DO file.

We have spent a lot of time in this phase to read a correct value, after that we found the problem was related to the timing issue. And this is what called "clock domain crossing". After we made search for it and read some documents such as [16] related to the clock domain issue, we found that the solution will be using the FIFOs in the input and output as we mentioned it in the section 4.3.1.

Now, we added PDI FIFO to store the PDI data before process them from the AEAD, SDI FIFO to store the SDI data before process them from the AEAD,

DO_FIFO to store the output of AEAD. And also we added a signal inside the block IP which named as "input_Enable", this signal is active high and we control it from the C code, when we send all input data " PDI and SDI " to the FIFOs registers, we make the "input_Enable" signal high level to till the AEAD to read the input data from the FIFOs and start the processing. When the AEAD finish the processing, it will send the "output_valid" to the DO_FIFO and starts to send the output data to the DO_FIFO, and when the DO_FIFO is filled, it will make the " FIFO_output_Valid" high.

In the C code, after we send all the input data "PDI and SDI" we always check the bit "FIFO_output_Valid", if it is high, then we will read the registers of the DO_FIFO. We have compared between the data of DO file (corresponding message) and the data that we got from the DO_FIFO after the processing, and we found it is correct.

Fig.5.13 shows the output of the AEGIS_IP after we send the PDI and SDI using the ARM core, and we used the SDK terminal to display the output.

We repeated this step for all algorithms, and we ensured that the output data is as expected.

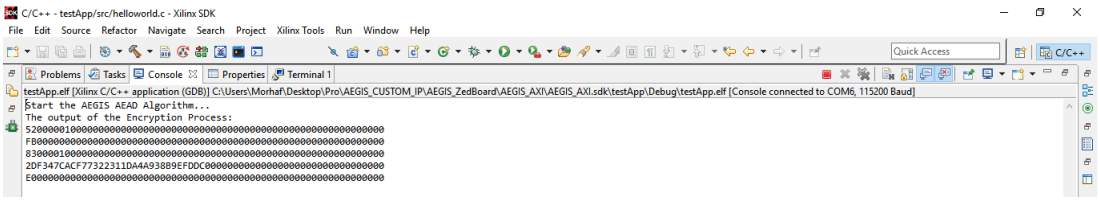



Figure 5.13: The output of the Encryption Process using AEGIS Algorithm

In the Encryption side, the system consist of two parts, static part and dynamic part. Static part consist of 3 IP blocks as shown in Fig. 5.9, AXI-HWICAP IP block, PS block and our IP block "CAESAR". AXI-HWICAP IP block and PS block are produced by Vivado tool, but the CAESAR block is our work and it consist of: AEAD top, 3 FIFOs and LFSR. The dynamic part consist of one reconfigurable region which is where the cipher module will be located. The DPR design flow is as it illustrated in section 5.2. After we connected the ZC702 board to the PC and define the port that it connected with, we program the FPGA and we run the C code in "Launch on Hardware (GDB)" mode in SDK tool, to show the pattern that produced from the LFSR and the data output using the terminal

inside the SDK. The output data of the AEAD will be send to another IoT node using ( UART 1 ) as shown in Fig. 4.17 .

Figures (5.14,5.15) are showing the output of the encryption module, there are 5 messages and each message encrypted using one of the CAESAR algorithms. Generating cipher IDs is the task of the LFSR as we mentioned before.

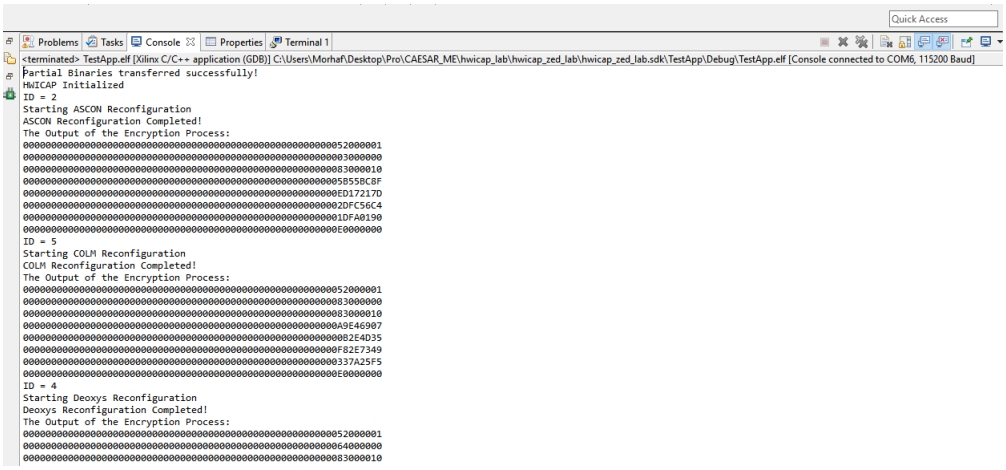



Figure 5.14: The output of the Encryption Module (1)


Figure 5.15: The output of the Encryption Module (2)

The output message is divided into 5 lines: first line is a header to inform if the output data is plain text or cipher text, the length of this data and if this line is last line in the output message or not, as we shown that the first line is: (0x52000001)and this means that the data is cipher text and will be one byte in our case and this line is not the last line for the output message. The second line will be the cipher text. The third line is a header related to the tag which define

the length of the tag and then if this line is the last one, as we have shown that the third line is (0x8300001) which means that the tag in our case is 16 byte. And the fourth line is the tag value. And the last line of the output value corresponding to the status of this message, if the encryption done correctly, then the value will be "0xE0000000" else the value will be "0xF0000000".

However, In the decryption side, the CAESAR IP block consist of: AEAD and FIFOs. First, the decryption module will receive the cipher ID and the PS will till the HWICAP to start transforming the bit file of this algorithm to the configuration memory. After that, the decryption module will receive the cipher text and decrypts it. Fig.5.16 shows the output of the decryption module.

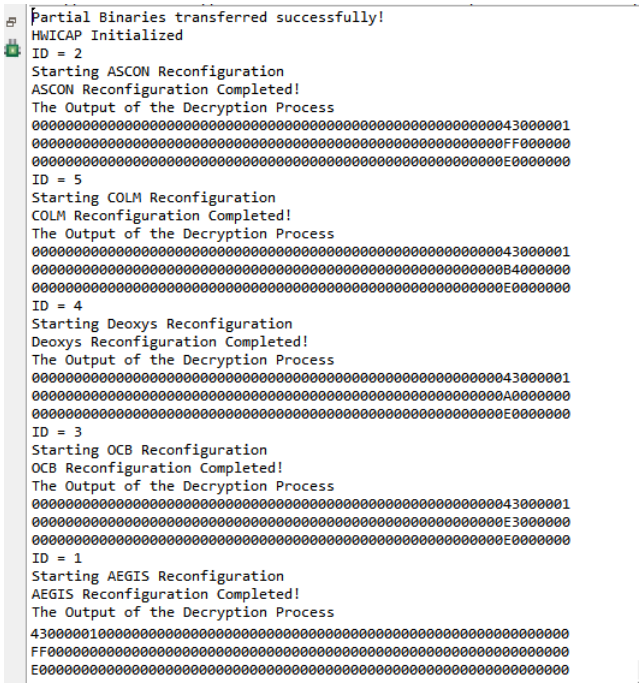



Figure 5.16: The output of the Decryption Module

As we shown, the output message consist of 3 lines: the first one is to inform if the output data is cipher text or plain text, and the length of this data. The second one is the data and the last line is status of this message, "0xE0000000" means the status is success and "0xF0000000"is Fail.

# Chapter 6

# Conclusion and Discussion

## 6.1 Conclusion

In this thesis, a new design that adds a new dimension of security for constrained IoT devices using the concept of algorithm hopping , in analogy to the well known frequency hopping technique, is introduced. DPR technology is used for switching between 5 lightweight ciphers that are currently under review in the CAESAR contest and provides a reduction of 57% and 80% in area utilization and power consumption respectively. The design is mounted on a Xilinx XC7Z020LG484-1 Zynq FPGA and synthesized using Xilinx Vivado 2015.2.

## 6.2 Discussion from Utilization Perspective

Figure 6.1 shows a comparison between the static implementation and DPR-based implementation of the proposed design. It shows that if the proposed design implemented without DPR technique, then we have to reserve area for all the algorithms. in contrast, in our case we have reserved space for the larger algorithm and then during the run time we reconfigure the algorithm required.

However, figure 6.1 shows scale drawings of the 5 algorithms utilization (LUT only), and to illustrate the comparison from perspective of all resources that used, we make the figure 6.2

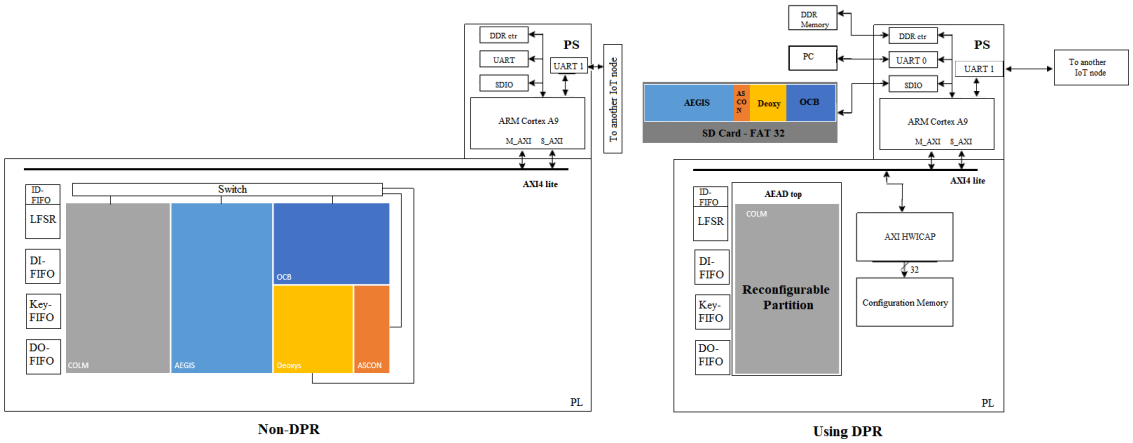



Figure 6.1: Comparison between DPR and non-DPR implementation.
The figure shows scale drawings of the 5 algorithms utilization (LUT only)

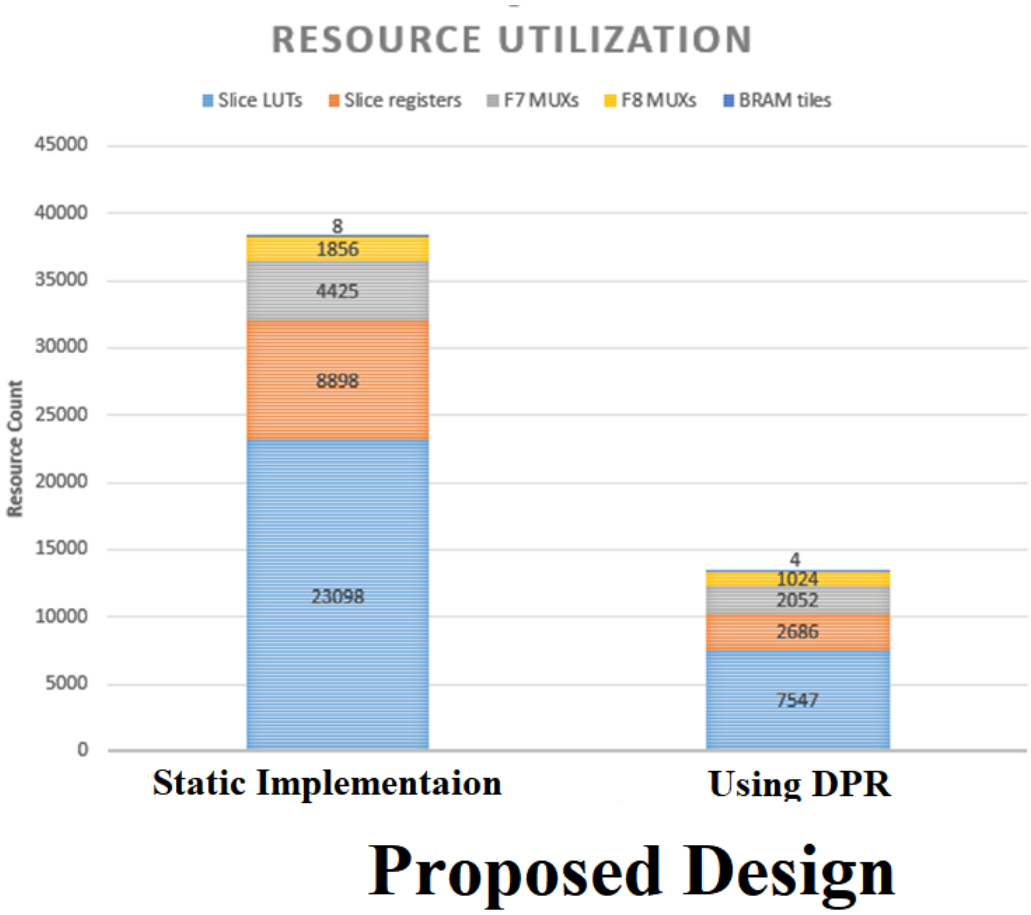



Figure 6.2: Comparison between DPR and non-DPR implementation for the proposed design from the point of view of all resources used.

## 6.3 Discussion from Power Perspective

Figure 6.3 shows that the using of DPR provides a reduction of 80% in power consumption assuming all algorithms are used at the same time. However, on the

assumption that a simple controller is used to switch between the algorithms, and therefore one algorithm only works and the rest does not at the run time, the power consumption will be the sum of : the average power of the 5 algorithms plus the leakage power of each algorithm plus the consumed power of the simple controller.

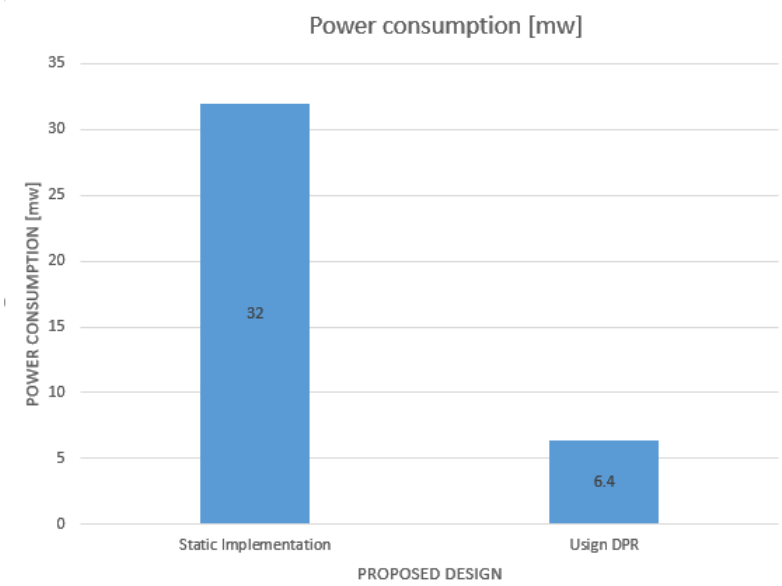



Figure 6.3: Comparison between DPR and non-DPR implementation of the proposed design from the point of view of power consumption .

However, the table 6.1 shows the comparison between our work and [2] when the two design work at 100 Mhz. Whereas, the table 6.2 the comparison between our work and [3] when the two design work at 157.63 MHz.

Table 6.1: Comparison between the proposed design and [2]

| Parameter | Proposed Design | [2] |
|---|---|---|
| device | XC7Z020LG484-1 | XC7Z010clq225-3 |
| power [mw] | 63.4 | 455 |
| confidentiality | yes | yes |
| Integrity | yes | No |
| Authentication | yes | No |
| second dimension of security | yes | No |

**Energy per bit:**

Energy per bit is the consumed energy for encrypting a single bit while energy per block is the consumed energy to encrypt a complete block.
Table 6.3 shows the energy per bit values for the most commonly used low-power wireless technologies.

Table 6.2: Comparison between the proposed design and [3]

| Parameter | Proposed Design | [3] |
|---|---|---|
| device | XC7Z020LG484-1 | Cyclone IV (Altera) |
| power [mw] | 100.2 | 166.67 |
| confidentiality | yes | yes |
| Integrity | yes | yes |
| Authentication | yes | yes |
| second dimension of security | yes | No |

Table 6.3: Energy per bit for low-power wireless technologies

| Technology | Energy per Bit(µJ/bit) |
|---|---|
| ANT | 0.71 |
| BLE | 0.153 |
| Zigbee | 185.9 |
| Wi-Fi | 0.00525 |

Since there are 5 algorithms in the proposed design, the Energy per bit is calculated for each one, after that the average Energy per bit is calculated.

**Energy per bit for COLM algortihm:**

Power = 11 mW.

Time for complete encryption = 39 * 100 n = 3.9 µs as the frequency is 10MHZ.

Energy = 0.0429 µJ.

Energy per bit = 335 pJ/bit (as the PDI = 128 bits).

**Energy per bit for AEGIS algortihm:**

Power = 9 mW.

Time for complete encryption = 23 * 100 n = 2.3 µs as the frequency is 10MHZ.

Energy = 20.7 nJ.

Energy per bit = 161.7 pJ/bit (as the PDI = 128 bits).

**Energy per bit for Deoxys algortihm:**

Power = 6 mW.

Time for complete encryption = 19 * 100 n =1.9 µs as the frequency is 10MHZ.

Energy = 11.4 nJ.

Energy per bit = 89 pJ/bit (as the PDI = 128 bits).

**Energy per bit for OCB algortihm:**

Power = 4 mW.

Time for complete encryption = 17 * 100 n =1.7 µs as the frequency is 10MHZ.

Energy = 6.4 nJ.

Energy per bit = 50 pJ/bit (as the PDI = 128 bits).

**Energy per bit for ASCON algortihm:**

Power = 2 mW.

Time for complete encryption = 16 * 100 n =1.6 µs as the frequency is 10MHZ.

Energy = 3.2 nJ.

Energy per bit = 25 pJ/bit (as the PDI = 128 bits).

**The average Energy per bit**

The average energy per bit for the proposed design is 130.34 pJ/bit. After the comparison between the proposed system and the table 6.3, it can be seen that the energy per bit for the proposed design is a lot lower than the energy per bit for the communication modules that are being used today.

## 6.4 Brute-Force Attack

The attack which uses the method "Trial and error" by guessing passwords is called Brute force attack. An attacker first gathers the fundamental information about the user. For example, user's full name, room number, vehicle number, children names etc.

The attacker continuously tries random passwords on the basis of the user's personal information. The attacker tries this until he/she gets success. This may take hours, days, months and years also.

The table 6.4 shows the number of alteration for different key sizes.

Table 6.4: NO. OF ALTERNATION for Different Key Sizes

| KEY SIZE | NO. OF ALTERNATION |
|---|---|
| 32-bit | $2^{32}$ |
| 56-bit | $2^{56}$ |
| 128-bit | $2^{128}$ |
| 168-bit | $2^{168}$ |

In the proposed design, there are 5 ciphers, the key size for each cipher is 128 bits. Therefore, the number of alternation for each cipher is: $2^{128}$ which equal to : $3.4028237e+38$.However,the attacker needs ($2^{numberOfBits}$ multiplied by duration of a single attack.) to attack the system. If one attack takes a millisecond, then it'll take ($TimeToBreak128Bits = 1.076080142e+28$ ) years to go through $2^{128}$ values. The key is different for each cipher algorithm.

Since we make hopping between the 5 algorithms, then the time that is need to break the system is: ($TimeToBreak5Algorithms = TimeToBreak128Bits^5$ ). In addition, the sequence of patterns is random and it is changeable during the run time and since we have 3-bits LFSR, then the time that is need to break the system will be ($TimeToBreak5Algorithms * 7$).

## 6.5 Future Work

The reconfiguration time is the main drawback of DPR and we suggest the following recommendations for future research investigations to reduce the time of reconfiguration :

- Reduce the reconfiguration time overhead by reduce the bitstream size of the RMs by applying compression /decompression techniques.
- Optimize the way the bitstreams are transferred by improve the ICAP interface speed.
- Another approach to reduce the reconfiguration overheads is to apply scheduling techniques that attempt to hide the reconfiguration latency by fetching the configurations in advance and storing them in idle reconfigurable regions.

# Bibliography


[1] Xilinx Inc., "Vivado design suite partial reconfiguration user guide UG909 ," Apr. 2017.

[2] S. M. Soliman, B. Magdy, and M. A. A. El-Ghany, "Efficient Implementation of the AES Algorithm for Security Applications," *29th IEEE International System-on-Chip Conference (SOCC), Seattle, WA, 2016.*

[3] A. HAFSA, N. ALIMI, A. SGHAIER, M. ZEGHID, and M. MACHHOUT, "A Hardware-Software Co-designed AES-ECC Cryptosystem," *International Conference on Advanced Systems and Electric Technologies (IC_ASET)*, 2017.

[4] C. Lane, "Book: The Long View: Mobile 2025 - Part II - What is the outlook for machine-based demand?," *Bernstein Research*, 2017.

[5] A. Massimo, "Enabling the internet of things from integrated circuits to integrated systems," *Springer*, 2017.

[6] C. Thangavel and P. Sudhaman, "Book: Connected Environments for the Internet of Things," *Springer*, 2017.

[7] " IoT security market Report." Available:: https://iot-analytics.com/new-iot-security-report/, 2017.

[8] "ZC702 Evaluation Kit Getting Started Guide", UG926 (v6.0), December 17, 2013, found under http://www.xilinx.com/.

[9] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, "The Zynq Book," July 2014.

[10] Xilinx Inc., "Zynq-7000 Technical Reference Manua UG585 v1.7," 2014.

[11] E. Homsirikamol, W. Diehl, A. Ferozpuri, F. Farahmand, M. U. Sharif, and K. Gaj, "GMU Hardware API for Authenticated Ciphers," *International Conference on Reconfigurable Computing and FPGAs*, pp. 1–8, Dec. 2015.

[12] C. Maxfield, "Book: The design warrior's guide to fpgas," *Newnes*, 2004.

[13] D. Dye, "Partial Reconfiguration of Xilinx FPGAs Using ISE Design Suite," *Xilinx*, 2012.

[14] A. Kamaleldin, A. Mohamed, A. Nagy, Y. Gamal, A. Shalash, and H. Mostafa, "Design Guidelines for the High-Speed Dynamic Partial Reconfiguration Based Software Defined Radio Implementations on Xilinx Zynq FPGA," *IEEE International Symposium on Circuits and Systems*, May 2017.

[15] Xilinx Inc., "AXI HWICAP PG134," Oct. 2016.

[16] S. Kilts, "Book: Advanced FPGA Design Architecture, Implementation, and Optimization," *Wiley-Interscience*, 2007.

[17] R. Minerva, A. Biru, and D. Rotondi, "Towards a definition of the Internet of Things (IoT)," *IEEE*, May 2015.

[18] C. Bekara, "Security Issues and Challenges for the IoT-based Smart Grid," *Procedia Computer Science Journal*, vol. 34, pp. 532–537, 2014.

[19] M. Katagi and S. Moriai, "Lightweight Cryptography for the Internet of Things," *Sony Corporation*, pp. 7–10, 2008.

[20] S. Koteshwara and A. Das, "Comparative study of Authenticated Encryption targeting lightweight IoT applications," *IEEE Design and Test*, vol. 34, pp. 26–33, 2017.

[21] K. Ashton, "That " internet of things"thing," *RFiD Journal*, vol. 22, pp. 97 – 114, 2009.

[22] IEEE, " Special Report: The Internet of Things," 2014.

[23] S. C. Mukhopadhyay and N. K. Suryadevara, "Book: Internet of Things: Challenges and Opportunities," *Springer*, 2014.

[24] O. Vermesan and P. Friess, " Book: Internet of Things-From Research and Innovation to Market Deployment," *River*, 2014.

[25] N. Mahalle and P. Railkar, "Book: Identity Management for Internet of Things," *River*, 2015.

[26] Xilinx Inc., "LogiCORE IP MicroBlaze Micro Controller System, Product Specification." Available:: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1ds865_microblaze_mcs.pdf, 2012.

[27] ARM, "Cortex-A9 MPCore Technical Reference Manual." Available:: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0407g/DDI0407G_cortex_a9_mpcore_r3p0_trm.pdf, 2011.

[28] "AXI Reference Guide", UG761 (v13.1), March 7, 2011, found under http://www.xilinx.com/.

[29] S. Li and L. D. Xu, "Book: Securing the internet of things," *Syngress*, 2017.

[30] C. Paarr and J. Pelzl, " Book: Understanding Cryptography," *Springer*, 2010.

[31] P. Rogaway, "Authenticated-encryption with associated-data," *ACM Conference on Computer and Communications Security, ACM Press*, 2002.

[32] H. Krawczyk, " The order of encryption and authentication for protecting communications (or: How secure is SSL?," *Springer*, 2001.

[33] C. Competition, "https://competitions.cr.yp.to/caesar-call.html," 2014.

[34] H. Wu and B. Preneel, "AEGIS: A Fast Authenticated Encryption Algorithm," in *Selected Areas in Cryptography – SAC 2013. Lecture Notes in Computer Science* (T. Lange, K. Lauter, and P. Lisonek, eds.), vol. 8282, Berlin,Heidelberg, (2014): Springer.

[35] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1.2." Submission to the CAESAR competition: http://competitions.cr.yp.to/round3/asconv12.pdf, 2016.

[36] J. Daemen, "Permutation-based Encryption, Authentication and Authenticated Encryption," *DIAC- Directions in Authenticated Ciphers*, July 2012.

[37] E. Andreeva, A. Bogdanov, N. Datta, A. Luykx, B. Mennink, M. Nandi, E. Tischhauser, and K. Yasuda, "COLM v1." Submission to CAESAR competition., 2015.

[38] C. Cid, T. Huang, T. Peyrin, Y. Sasaki, and L. Song, "A Security Analysis of Deoxys and its Internal Tweakable Block Ciphers," *IACR Transactions on Symmetric Cryptology*, vol. 3, pp. 73–107, 2017.

[39] T. Krovetz and P. Rogaway, "The OCB Authenticated-Encryption Algorithm," *RFC 7253, DOI 10.17487/RFC7253, (May 2014)*, May 2014.

[40] M. S. Ibrahim, I. Ahmed, M. I. Aslam, M. Ghazaal, M. Usman, K. Raza, and S. Khan, "A Low Cost FPGA based Cryptosystem Design for High Throughput Area Ratio," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 8, 2017.

[41] N. H. Motlagh, "Frequency Hopping Spread Spectrum : An Effective Way to Improve Wireless Communication Performance," *Advanced Trends in Wireless Communications*, 2011.

[42] Xilinx Inc., " Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators," Jul. 1996.

[43] K. Vipin and S. A. Fahmy, "ZyCAP : Efficient Partial Reconfiguration Management on the Xilinx Zynq," *IEEE Embedded Systems Letters*, vol. 6, no. 3, pp. 41–44, 2014.

[44] Xilinx Inc., "ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC UG850 ," Sep. 2015.

[45] R. L. Roux, G. V. Schoory, and P. V. Vuuren, "Block RAM-based architecture for real-time reconfiguration using Xilinx FPGAs," *SACJ, Research Article*, 2015.

[46] Xilinx Inc., "Creating and Packaging Custom IP UG1118 ." Available:: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_2/ug1118-vivado-creating-packaging-custom-ip.pdf, 2017.