A

# DISSERTATION REPORT

*on*

# Implementation of Lightweight Cryptography Algorithm - PRESENT

*And*

# Register Configuration and Validation

*by*

**SHASHANK SINGH**

2015PEV5069

*under the supervision of*

**Dr. C. Periasamy**

Assistant Professor, ECE Department

MNIT, Jaipur

*Submitted in partial fulfillment of the requirements of the degree of*

**MASTER OF TECHNOLOGY**

*to the*


**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**MALAVIYA NATIONAL INSTITUTE OF TECHNOLOGY, JAIPUR**

**JULY - 2017**

**Department of Electronics and Communication Engineering**

**Malaviya National Institute of Technology, Jaipur**

# Certificate

This is to certify that this Dissertation report entitled " **Implementation of Lightweight Cryptography Algorithm - PRESENT**" and "**Register Configuration and Validation**" by SHASHANK SINGH (2015PEV5069) is the work completed under my supervision and guidance, hence approved for submission in partial fulfillment for the award of degree of Master Of Technology in VLSI DESIGN in the Department of Electronics and Communication Engineering, Malaviya National Institute of Technology, Jaipur in the academic session 2016-2017 for full time post graduation program of 2015-2017. The contents of this dissertation work, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

(Dr. C. Periasamy)

Assistant Professor, Dept. of ECE

MNIT, Jaipur

# Declaration

I, hereby declare that the work which is being presented in this project entitled "**Implementation of LightWeight Cryptography Algorithm - PRESENT** " and “**Register Configuration and Validation**” in partial fulfillment of degree of Master of Technology in VLSI Design, is an authentic record of my own work carried out under the supervision and guidance of Dr. C. Periasamy in Department of Electronics and Communication, Malaviya National Institute of Technology, Jaipur. I am fully responsible for the matter embodied in this project in case of any discrepancy found in the project and the project has not been submitted for the award of any other degree. I also confirm that I have consulted the published work of others, the source is clearly attributed and I have acknowledged all main sources of help.

SHASHANK SINGH

2015PEV5069

# Acknowledgement

I am grateful to my supervisor Dr. C. Periasamy for his constant guidance and encouragement and support to carry out this work. His excellent cooperation and suggestion provided me with an impetus to work and made the completion of work possible. He has been great source of inspiration to me, all through. I am very grateful to him for guiding me how to conduct research and how to clearly & effectively present the work done.

I would like to express my deepest sense of gratitude and humble regards to our Head of Department Prof. K. K. Sharma for giving encouragement in my endeavors and providing necessary facility in the Department. I am thankful to Intel Technology India Pvt. Ltd. for giving me internship opportunity and providing me the access for various EDA tools which helped me in my thesis work. I am very thankful to all faculty members of ECE, MNIT for their valuable assistance and advise. I would also like to thank my friends for their support in discussions which proved valuable for me. I am indebted to my parents and family for their constant support, love, encouragement and sacrifices made by them so that I could grow up in a learning environment. Finally, I express my sincere thanks to all those who helped me directly or indirectly to successfully complete this work.

# Abstract

This report is divided into two parts Part A and Part B. Part A begins with security aspects in IoT implementation, why security is important in IoT and what are the main challenges in its implementation. After this the idea of Lightweight Cryptography Algorithm is discussed, what are its advantages over other classical cryptography algorithms and other features. The main focus is kept on PRESENT cipher which is an ultra Lightweight algorithm.
It is implemented using Verilog and synthesized thereafter. The results obtained are compared with other Lightweight Cryptography Algorithms. Chapter 1 to chapter 5 cover Part A.

Second part of this report is a literature survey on a very important issue in semiconductor industry. The issue is Register Configuration and Validation. We all know that in a complex SoC we can have thousands of registers, so for specification matching we need to configure these registers using System RDL which is an industry specific language and than we need to validate it. For validation we need to generate various collaterals from the RDL file and validate the ovm collateral(IP specific register model) using CREST, which is an Intel's **C**onverged **Re**gister **S**pecification **T**est. After validation we can conclude whether the Register Specifications are matching with Design RTL or not. Chapter 6 and chapter 7 cover Part B.

# Contents

# List of Figures

# List of Tables

# PART A

# Chapter 1

# Introduction

## 1.1 Motivation

IoT is termed as Internet of Things. As it is estimated that by 2020, fifty billion gadgets should be associated with Internet, this phenomenon is called as Internet of Things. A device in the Internet of Things can be smart phones or PCs or any man-made devices that can have a unique identity (e.g. IP Address) and they must have the capacity to exchange information between them.
As of now a portion of such devices are accessible in business and research sector. If it continues like this then those days are not far when these gadgets will become an important part of life for a normal person.
IoT is going to take very important part in creating smart cities, smart homes and intelligent network between things. Especially in India, IoT is going to play key role in Digital India Campaign.

The biggest challenge of the IoT is that it is going to cover different hardware devices to communicate with each other e.g. communication between a washing machine and a smartphone, communication between a PC and a door's lock and this way we can have infinite combinations and we don't have a effective technology yet which can provide a common platform for such huge number of devices to communicate with each other. And most importantly the data interchange between these devices are not secure. If we talk about present time then data is considered as biggest asset and it forms the central system of IoT and if that only is not secured than it's a big issue, which needs to be resolved.

Security is itself a very broad topic. Protection of information has been an problem ever since the first two computers were connected to each other. With the commercialization of the Internet, security concerns expanded to cover personal privacy, financial transactions, and the threat of cybertheft. In

IoT, security is the least touched area. Whether accidental or malicious, intercepting the controls of a pacemaker, a car, or a nuclear reactor poses a threat to human life.

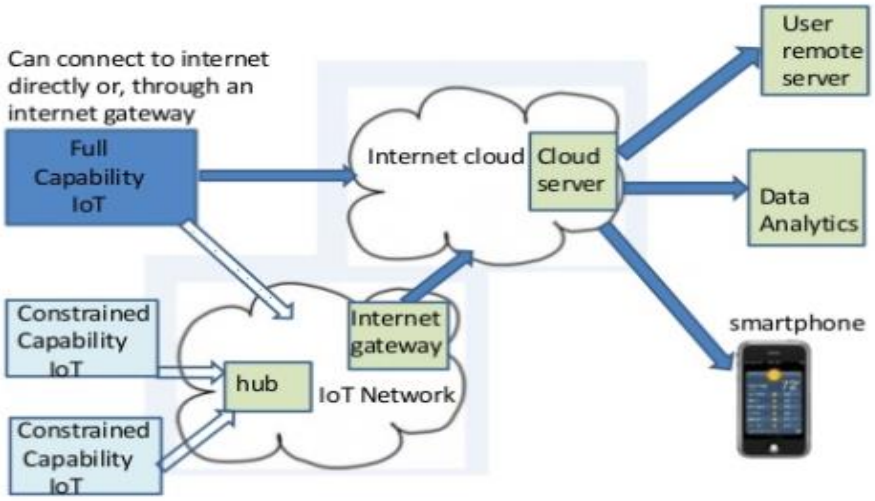


Figure 1.1: IoT system architecture

## 1.2 Challenge

The question arises why Security is such a big challenge in the world of IoT. As networking appliances have recently come in commercial market so this idea is relatively new and in such appliances security was never given importance. Most of the IoT products are available in market with outdated and obsolete embedded software and operating systems. So they don't receive the latest security updates.

The second and main problem is that in an embedded application the fully capable cryptographic environment is not possible because of the constraints like power dissipation, area and cost. The main criterion for the lightweight cipher is to have less memory space hence resulting a less Gate Equivalent (GEs) count for an efficient hardware implementation, without compromising the requirement of strong security properties. An ISO/IEC standard on lightweight cryptography requires that the design be made with 1000–2000 gate equivalents (GEs) [1]. RFID tags may have 1000–10000 GEs out of that only 300–2100 GEs would be available for security aspects [2]. Many algorithms have been designed in the past few years and implemented in the field of pervasive computing. For security applications, total GEs available would be approx 2000–3000. Block ciphers should be

limited to less GEs in order to fit in lightweight applications. Ciphers like AES [3], DES [4], [5] would result in high GEs that make them infeasible for small scale real time applications.

## 1.3 Objective

The first objective of this project is to implement a Lightweight cryptography algorithm PRESENT as a public key algorithm using Verilog and synthesize it for minimum area (Gate Count) and power. After it this algorithm is implemented using Perl. We all know that Perl is a scripting language and it is widely used in Linux platform. As most of the Operating Systems for IoT nodes are Real Time and they are mainly based on Linux, so a Perl implementation of this algorithm is very important. On top of that Perl is faster than C, C++ or any other compiler language, so speed of execution of this algorithm will be faster.

## 1.4 Overview of this Project

The intention of this work is to make it understandable for a concerned person and wanted to be explanatory to the observer or examiner. In the chapter 2 we will discuss about the theory and complexity of the algorithm. Chapter 3 deals with its implementation in Verilog and Perl, we will also discuss the synthesis of our design using Synopsys design Compiler. Chapter 4 is the conclusion chapter in which we will compare our obtained results with the existing results.

# Chapter 2

# Cryptography

## 2.1 Definition

The method of converting a plain text message i.e original information into an other form which cannot be read or understood and further transforming encrypted message back to it's original form is called Cryptography. Cryptography consists of two processes:

1. Encryption
2. Decryption

Now we will define some basic terms which are generally used with Cryptography.

- Plaintext – The original information message.
- Cipher text – The encrypted data.
- Cipher – Cipher is an algorithm which is used for transforming a plain text message into an encrypted message by using several techniques.
- Key – It is a critical information only known to sender and receiver.
- Cryptanalysis – Cryptanalysis is a process of finding out the original message or portion of original message from the encrypted message without knowing the key.
- Cryptology – It is a study of both cryptography and cryptanalysis.
- Substitution – Replacing a data by any other data.

- Transposition – Possible permutation of data i.e. mixing the data to increase randomness.

## 2.2 Attacks on cryptosystems

Categories of various attacks are based on action performed by the attacker. An attack can be of two types passive or active.

### 2.2.1 Passive Attack

In this attack, attacker tries to get the illegal access of the transmitted information by intercepting it in the transmission medium. The reason that this attack is called as passive because the information is not altered or communication channel is also not disturbed. It is actually an information theft in which owner does not have any idea about this, so it is very dangerous to have this kind of situation.

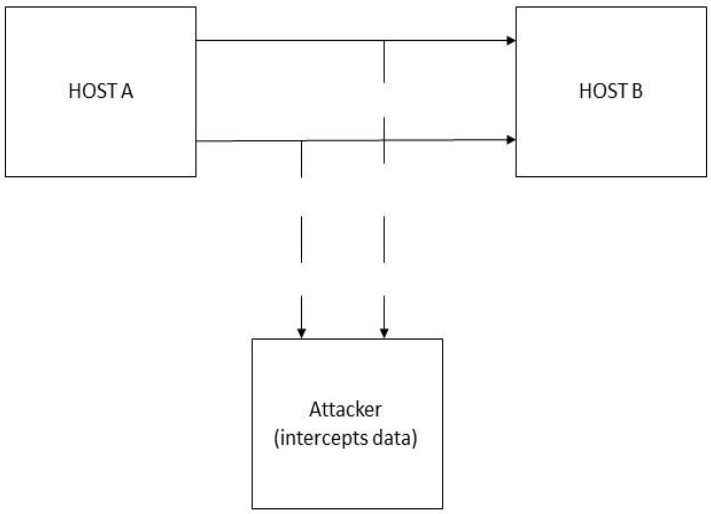


Figure 2.1: Passive Attack

### 2.2.2 Active Attack

In this type of attack, attacker tries to modify or alter the transmitted information by following ways:

- Intercepting the information channel in such a way to modify the data bits in message.
- Unintended initiation of transmission of information.
- Unauthorized deletion of data bits in information, so as to corrupt the data.
- Denial of access of information for legitimate users.

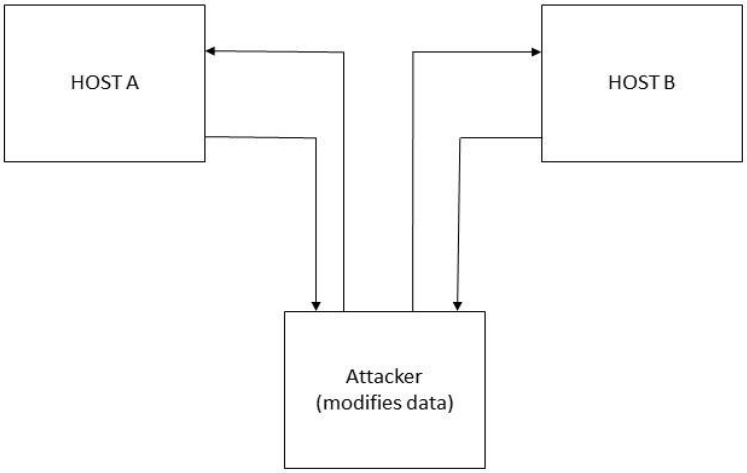


Figure 2.2: Active Attack

## 2.3 Other Types of Cryptography Attacks

- Cipher text only Attack (COA) – In this attack, the attacker has access to some portion of cipher text having no idea of the corresponding plaintext. This attack is said to be successful when the attacker is able to determine plaintext from any available set of cipher text.

- Known Plaintext Attack – In this method, the attacker knows the plaintext for some part of cipher text, using this information attacker

tries to determine the key. Once the attacker finds the key he decrypts the rest of cipher text. Linear cryptanalysis against block cipher is one of the example of this attack.

- Chosen Plaintext Attack – In this type of attack, attacker selects a plaintext and somehow manages to get its encrypted form. This results in plaintext-ciphertext pair available for attacker which makes the task of determining key very easy for attacker.

- Dictionary Attack – As per this method attacker maintains a dictionary having plaintext-ciphertext pairs which the attacker has collected for a long time. So when the attacker gets any ciphertext, he checks for its corresponding plaintext in that same dictionary.

- Brute Force Attack – This attack deals with using all possible combination of keys to decrypt a cipher text. For example suppose key length is 16 bits, then $2^{16}$ will be the total number of possible keys. So in this attack, attacker tries all these keys until he gets the proper plaintext. This attack will be of no use if the key length is too big.

- Birthday Attack – In this attack, attacker uses two different inputs that will result in a same hash value. If the attacker gets success than it is said to be collision and attacker will have a broken hash function.

- Man in Middle Attack – In this method attacker comes in between sender and receiver and pretend as a legal person tries to intercept the plain text. It is the main attack against public key cryptography, where exchange of key takes place before information exchange.

- Side channel attack – Physical Implementation of cryptosystem are exploited using this attack.

- Timing attack – As we know that computation time of different processes are different. In this attack, attacker checks the computation time in encryption process and based on those timing he tries to guess various processes carrying out in cryptosystem.

## 2.4 Types of cryptography

There are two types of cryptography, which are discussed below:

- Symmetric Key cryptography – The same key is used for both encryption and decryption. A sender and a receiver must have a common key. Key distribution between sender and receiver is a complicated problem. This method is generally very fast and ideal for encrypting large amount of data.

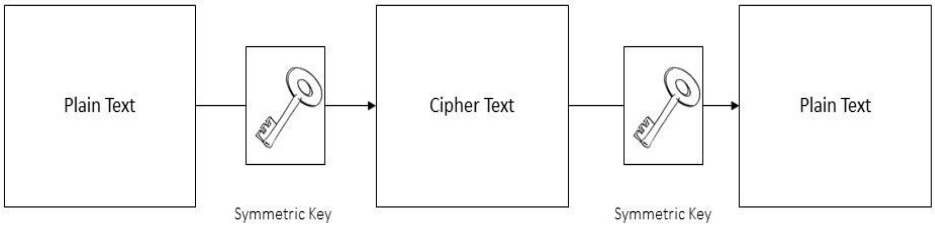


Figure 2.3: Symmetric Key cryptography

- Asymmetric key cryptography – In this method sender encrypts the plaintext with public key of receiver and sends the encrypted data which can be only decrypted using private key of receiver only. Thus there is no need of sharing the key. The public key is known to world but the private key is only known to receiver. This method is slower than symmetric key cryptography.

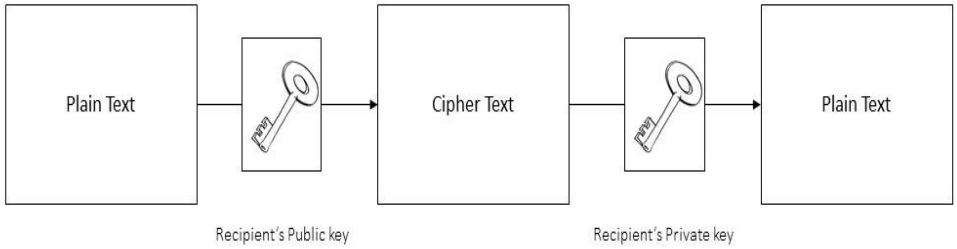


Figure 2.4: Asymmetric Key cryptography

## 2.5 Security Services of cryptography

The primary objective of using cryptography is to provide the following four fundamental information security services.

- Confidentiality – It is a type of security service in which the information has to be secured from unauthorized access and it can be implemented through various ways e.g physical securing or using some mathematical encryption algorithm.
- Data Integrity – It is a type of security service which helps in identifying an change in data. It checks whether the data is same or not since it has been created, stored or transmitted by a legal user. This service cannot prevent any alteration in data but it provides a way so that we will come to know whether the data has been modified in an unauthorized way.
- Authentication – This security service helps in identifying originator of message. With the help of this service the receiver ensures that data is sent by a verified sender.
- Non-repudiation – This helps in ensuring that an entity cannot refuse the ownership of any previous action or transaction means if a sender sends a message to a receiver and this security service is enabled in the transaction than later the original sender of message cannot deny his previous transaction.

# Chapter 3

# Understanding the Algorithm

## 3.1 Existing Work

As we know that AES [3] is the standard block cipher which is very popular in the present world of security. AES is the most favoured choice in almost all block cipher application. But there is a disadvantage too, as AES requires too much memory or GEs, it is not suitable for extremely constraint environment such as sensors and RFID tags. This limits its use in IoT devices.
AES is an SP(Substitution-Permutation) network block cipher.Resources required for AES are around 3600 GE [3]. Apart from AES we have one more block cipher known as DES [4] which is also very popular but in a constraint environment it fails to perform. DESL [6] and DESXL [7] are lightweight versions of DES and they are proposed by slight modification in original DES. This modification includes reducing the size of S-boxes and by applying key whitening.

Less memory space is the reason behind less Gate Equivalent (GEs) count , it is the main requirement of lightweight cipher, but it should not affect the security property of the algorithm, so achieving this is the main challenge.

We have many Lightweight ciphers too. Sony developed one compact cipher CLEFIA [8], [9]. Their main purpose was to achieve less GEs, it has two diffusion and two confusion properties that results in a hgher memory requirement. Few examples of lightweight cryptography algorithm for low power devices like RFID or sensors are HIGHT [10], mCrypton [11], SEA [12], TEA [13] and ICEBERG [14]; and these are summarized in Table 5.1 with respect to their GEs. Above mentioned algorithms have more than 2300 GEs which is the main disadvantage with them, so they can’t be used in constraint applications.

## 3.2 PRESENT – The Block Cipher

PRESENT is a Substitution Permutation network based on 80 bit or 128 bit key size and 64 bit block size [15]. PRESENT [15] is a block cipher with 32 rounds. PRESENT is one of the leanest lightweight algorithms designed and it has obtained the ISO/IEC standard for lightweight cryptography [15].

### 3.2.1 Encrypting Plain text

Block length of 64 bits and key length of 80 bits or 128 bits are supported by this algorithm. In this design we are using 80 bits of key. In low security transactions key length of 80 bits is more than enough.
This cipher has 32 rounds of encryption and in each round there are four operations a. addRoundKey, b. sBoxlayer, c. pLayer, d. The key schedule.

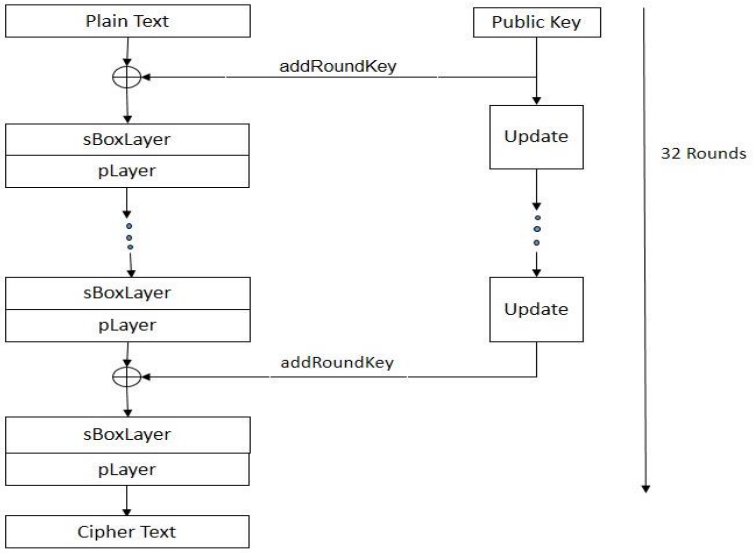


Figure 3.1: Encryption Block Diagram

a. addRoundKey - Let's assume we have a round key $K_i = k^i_{63}\ldots k^i_0$ for $1 \le i \le 31$ and plain text block as $b_{63}\ldots.b_0$, addRoundKey consists of the operation for $0 \le j \le 63$,

$$b_j \rightarrow b_j \oplus k^i_j$$

b. sBoxlayer – The S-Box used in PRESENT is a mapping of 4-bit to 4-bit values : $F^4 \rightarrow F^4$ . Its implementation in hexadecimal notation is shown below.

| x | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S[x] | c | 5 | 6 | b | 9 | 1 | 2 | d | e | 4 | f | 3 | 0 | 7 | 8 | A |

Table 3.1

For sboxLayer the current plain text block $b_{63}…b_0$ is considered as sixteen 4-bit words $w_{15}…w_0$.

c. pLayer – The bit permutation used in this block cipher is given in following table.

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P(i) | 0 | 16 | 32 | 48 | 12 | 17 | 33 | 49 | 20 | 18 | 34 | 50 | 4 | 19 | 35 | 51 |

| i | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P(i) | 1 | 5 | 9 | 13 | 8 | 21 | 37 | 53 | 36 | 56 | 38 | 54 | 44 | 60 | 39 | 55 |

| i | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P(i) | 2 | 6 | 10 | 14 | 24 | 22 | 26 | 30 | 52 | 57 | 42 | 58 | 28 | 61 | 62 | 59 |

| i | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P(i) | 3 | 7 | 11 | 15 | 40 | 23 | 27 | 31 | 25 | 41 | 43 | 47 | 29 | 45 | 46 | 63 |

Table 3.2

Bit position i of the block is moved to bit position P(i).

d. Key Scheduling – PRESENT can be implemented with either 80 or 128 bits keys. However here we are implementing this cipher with 80- bit keys. The user supplied key is stored in a key register K and represented as $k_{79}k_{78}...k_0$ . At round i the 64-bit round key consists of 64 leftmost bits of current content of register K. Hence at any round i, we have 64 – bit round key register as shown below

$K_i = \kappa_{63}\kappa_{62}\ldots\kappa_0 = k_{79}k_{78}\ldots k_{16}$

After this the key register K is updated as follows:

i) $[k_{79}k_{78}. . . k_1k_0] = [k_{18}k_{17}. . . k_{20}k_{19}]$
ii) $[k_{79}k_{78}k_{77}k_{76}] = sBoxLayer[k_{79}k_{78}k_{77}k_{76}]$
iii) $[k_{19}k_{18}k_{17}k_{16}k_{15}] = [k_{19}k_{18}k_{17}k_{16}k_{15}] \oplus round\_count$

Key register is rotated by 61 bit positions to the left. After that the left-most four bits are passed through S-Box and in third step round_count value is XORed with bits $k_{19}k_{18}k_{17}k_{16}k_{15}$ of K with least significant bits of round_count on the right.

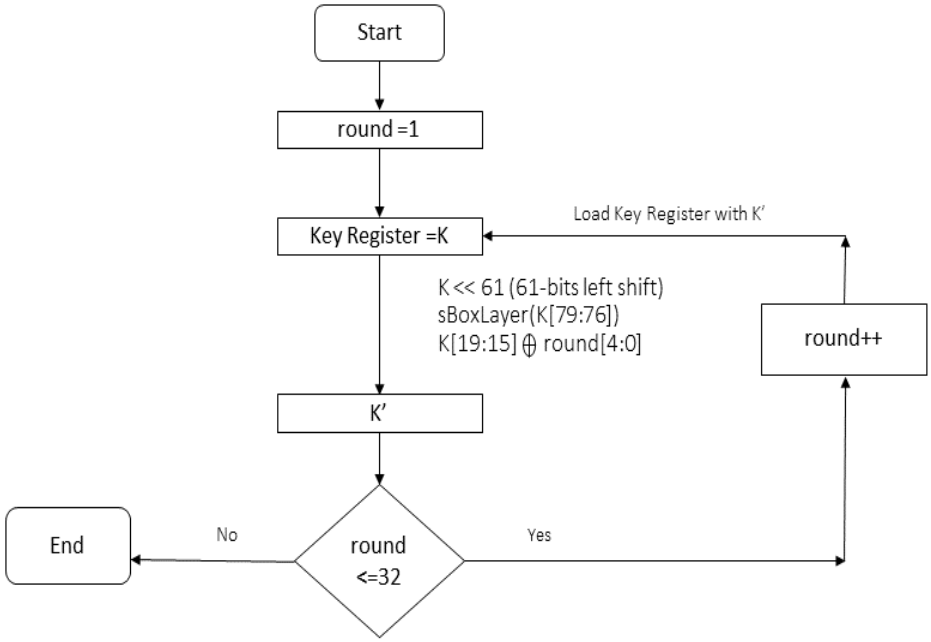


Figure 3.2: Key Updation

### 3.2.2 Decrypting Cipher text

Decryption is the process of getting plain text back from the cipher text. In this project PRESENT cipher is represented as Public Key Cipher. In this technique encryption is done using public key of the receiver which is known to outside world and decryption can be done only by using private key of receiver which is only known to the receiver, so here there is no problem of exchanging the key between sender and receiver and hence it is more secured. Decryption in PRESENT algorithm is the reverse process of encryption, it also runs for 32 rounds but the key used in first round is the private key of receiver.

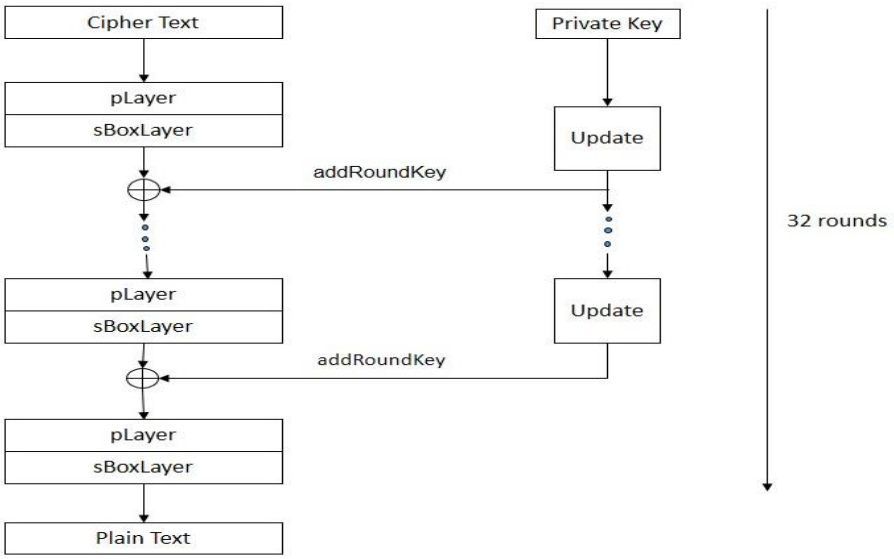


Figure 3.3: Decryption Block Diagram

## 3.3 Cryptanalysis of PRESENT Cipher

Cryptanalysis is the study of ciphers or cryptosystems with the aim to find weaknesses in them which will result in retrieval of the plaintext from the cipher text, without necessarily knowing the key or algorithm. This process is called as breaking the cipher. Breaking is sometimes used interchangeably with weakening. This means finding some way in design

that results in reducing the number of keys required in brute force attack. Brute force attack is simply trying all the possible combinations of key until the correct one is found. For example in the implementation of PRESENT cipher in this project we are using 80-bits of key, which means a brute force attack would need to try up to all $2^{80}$ combinations to find the correct key which is not possible given present and future computing abilities. However, a cryptanalysis of PRESENT cipher may reveal a technique using which we may get the correct key in less than $2^{80}$ combinations. While our cipher is not completely broken but now it is weaker.

| Cipher | Rounds | Key size | Attack type | Data complexity | Time complexity | Memory complexity | Reference |
|---|---|---|---|---|---|---|---|
| PRESENT | 24 | 80 | Stat. sat. | $2^{60}$ CP | $2^{20}$ PR | $2^{16}$ bytes | [16] |
| | 24 | 80 | Stat. sat. | $2^{57}$ CP | $2^{57}$ PR | $2^{32}$ bytes | [16] |
| | 7 | 128 | Bit-Pat. Int. | $2^{24.3}$ CP | $2^{100.1}$ MA | $2^{77}$ bytes | [17] |
| | 17 | 128 | Rel.-Key Rec | $2^{63}$ CP | $2^{104}$ MA | $2^{53}$ bytes | [17] |
| | 19 | 128 | Alg.-Dif | $6 \times 2^{62}$ | $2^{113}$ MA | Not specified | [18] |

Table 3.3 [22]

# Chapter 4

# Design Simulation and Synthesis

## 4.1 Simulation

Software used – Xilinx ISE 14.7
Simulator – ISim

## 4.2 Simulation Graphs

Design is implemented in Verilog.

### 4.2.1 Encryption - Waveforms

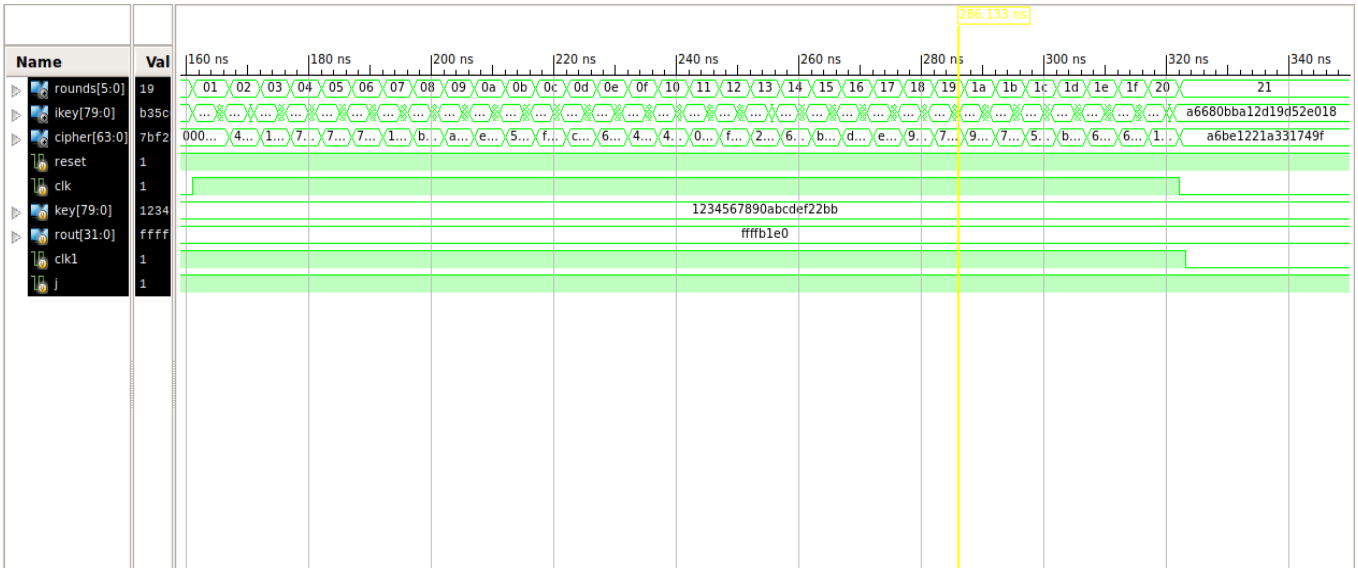


Figure 4.1: Waveform - Encryption

## 4.2.2 Decryption - Waveforms

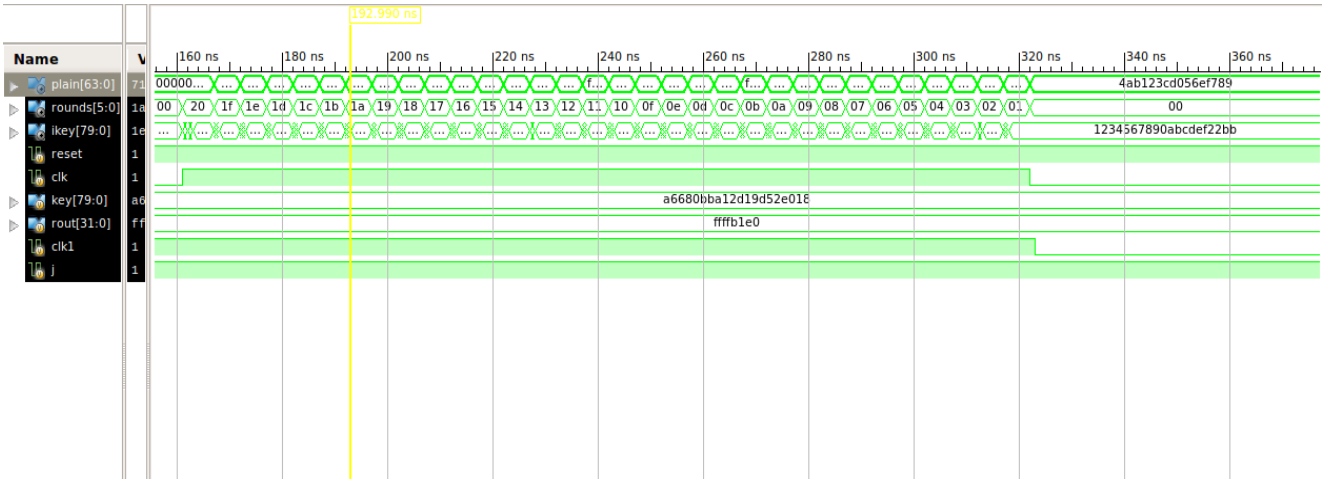


Figure 4.2: Waveform - Decryption

## 4.3 Simulation Results

As we can see from the simulation results of Encryption and Decryption that after 31 rounds of encryption the cipher text is formed.
This cipher text we are giving as input to decryption algorithm and after 31 rounds the plain text is successfully recovered.

a. Input/output of Encryption
Plain Text = 64'h4ab123cd056ef789
Key = 80'h1234567890abcdef22bb (public key of receiver)
Cipher Text = 64'ha6be1221a331749f

b. Input/output of Decryption
Cipher Text = 64'ha6be1221a331749f
Key = 80'ha6680bba12d19d52e018 (private key of receiver)
Plain Text = 64'h4ab123cd056ef789

## 4.4 Synthesis Results

Synthesis of any HDL is the process of converting high level language to gate level. Synthesis transforms high level Verilog/vhdl constructs, which don't have real physical hardware that can be wired up to perform a logic, into low level logical constructs which can be literally modeled in the form of transistor logic or look-up tables or other FPGA or ASIC hardware components.
In this project we are synthesizing our design using Synopsys Design Compiler version D-2010.03-SP5 and Xilinx ISE 14.7 synthesizer for Virtex 5, target device – XC5VLX20T.

### 4.4.1 Synthesis using Design Compiler

Technology Library – nonlinear.db 0.18um

#### a. For Encryption

- Area Report (Area is expressed in Gate Counts)

| Reference | Library | Unit Area | Count | Total Area |
|---|---|---|---|---|
| AND2 | nonlinear | 2 | 3 | 6 |
| BUF | nonlinear | 1 | 12 | 12 |
| D_LAT | nonlinear | 2 | 212 | 424 |
| INBUF | nonlinear | 0 | 6 | 0 |
| INV | nonlinear | 1 | 24 | 24 |
| MUX2_1 | nonlinear | 3 | 4 | 12 |
| NAND2 | nonlinear | 1 | 13 | 13 |
| NOR2 | nonlinear | 1 | 73 | 73 |
| OR2 | nonlinear | 1 | 10 | 10 |
| OR_AND_INV_2_1 | nonlinear | 3 | 2 | 6 |
| **Total 10 References** | | | | **580 (Net Area)** |

Table 4.1

| | |
|---|---|
| Number of Ports | 232 |
| Number of Nets | 435 |
| Number of Cells | 359 |
| Number of references | 10 |
| Combinational Area(GEs) | 156 |
| Non Combinational Area(GEs) | 424 |
| Total Cell Area(GEs) | 580 |

Table 4.2

- Power Report

| | |
|---|---|
| Global operating voltage | 5V |
| Voltage Unit | 1 V |
| Capacitance Units | 0.100000 ff |
| Net switching Power | 84.9397 uW |
| Total Dynamic Power | 84.9397 uW |

Table 4.3

# b. For Decryption

- Area Report ( Area is expressed in Gate Counts)

| Reference | Library | Unit Area | Count | Total Area |
|---|---|---|---|---|
| AND2 | nonlinear | 2 | 4 | 8 |
| BUF | nonlinear | 1 | 12 | 12 |
| D_LAT | nonlinear | 2 | 212 | 424 |
| INBUF | nonlinear | 0 | 6 | 0 |
| INV | nonlinear | 1 | 25 | 25 |
| MUX2_1 | nonlinear | 3 | 2 | 6 |
| NAND2 | nonlinear | 1 | 4 | 4 |
| NOR2 | nonlinear | 1 | 65 | 65 |
| OR2 | nonlinear | 1 | 4 | 4 |
| OR_AND_INV_2_1 | nonlinear | 3 | 1 | 3 |
| EXOR2 | nonlinear | 2 | 64 | 128 |
| **Total 11 references** | | | | **679 (Net Area)** |

Table 4.4

| | |
|---|---|
| Number of Ports | 232 |
| Number of Nets | 471 |
| Number of Cells | 399 |
| Number of references | 11 |
| Combinational Area (GEs) | 255 |
| Non Combinational Area (GEs) | 424 |
| Total Cell Area (GEs) | 679 |

Table 4.5

- Power Report

| | |
|---|---|
| Global operating voltage | 5 V |
| Voltage Unit | 1 V |
| Capacitance Units | 0.100000 ff |
| Net switching Power | 70.7394 uW |
| Total Dynamic Power | 70.7394 uW |

Table 4.6

## 4.4.2 Synthesis using Xilinx ISE 14.7

### a. For Encryption

| Slice logic Utilization | Used | Available | Utilization |
|---|---|---|---|
| No. of Slice Registers | 185 | 12480 | 1% |
| No. used as Latches | 185 | | |
| No. of Slice LUTs | 70 | 12480 | 1% |
| No. used as logic | 70 | 12480 | 1% |
| No. using O6 output only | 70 | | |
| No. of occupied Slices | 51 | 3120 | 1% |
| No. of LUT Flip Flop pairs used | 188 | | |
| No. with an unused Flip Flop | 3 | 188 | 1% |
| No. with an unused LUT | 118 | 188 | 62% |
| No. of fully used LUT-FF pairs | 67 | 188 | 35% |
| No. of unique control sets | 3 | | |
| No. of slice register sites lost to control set restrictions | 3 | 12480 | 1% |
| No. of bonded IOBs | 152 | 172 | 88% |
| No. of BUFG/BUFGCTRLs | 2 | 32 | 6% |
| No. used as BUFGs | 2 | | |
| Average Fanout of Non-Clock Nets | 3.99 | | |

Table 4.7

- Timing Summary

| | |
|---|---|
| Speed Grade | -2 |
| Minimum Period | 1.692ns (Maximum Frequency: 591.102MHz) |
| Minimum input arrival time before clock | 1.277ns |
| Maximum output required time after clock | 3.063ns |

Table 4.8

- Clock Information

| Clock Signal | Clock buffer(FF name) | Load |
|---|---|---|
| Clk | IBUF+BUFG | 126 |
| k_3_1 | BUFG | 59 |

Table 4.9

## b. For Decryption

| Slice logic Utilization | Used | Available | Utilization |
|---|---|---|---|
| No. of Slice Registers | 213 | 12480 | 1% |
| No. used as Latches | 213 | | |
| No. of Slice LUTs | 70 | 12480 | 1% |
| No. used as logic | 70 | 12480 | 1% |
| No. using O6 output only | 70 | | |
| No. of occupied Slices | 57 | 3120 | 1% |
| No. of LUT Flip Flop pairs used | 216 | | |
| No. with an unused Flip Flop | 3 | 216 | 1% |
| No. with an unused LUT | 146 | 216 | 67% |
| No. of fully used LUT-FF pairs | 67 | 216 | 31% |
| No. of unique control sets | 3 | | |
| No. of slice register sites lost to control set restrictions | 3 | 12480 | 1% |
| No. of bonded IOBs | 152 | 172 | 88% |
| No. of BUFG/BUFGCTRLs | 2 | 32 | 6% |
| No. used as BUFGs | 2 | | |
| Average Fanout of Non-Clock Nets | 3.94 | | |

Table 4.10

- Timing Summary

| | |
|---|---|
| Speed Grade | -2 |
| Minimum Period | 1.692ns (Maximum Frequency: 590.937MHz) |
| Minimum input arrival time before clock | 1.280ns |
| Maximum output required time after clock | 3.049ns |

Table 4.11

- Clock Information

| Clock Signal | Clock buffer(FF name) | Load |
|---|---|---|
| Clk | IBUF+BUFG | 133 |
| k_3_1 | BUFG | 80 |

Table 4.12

# Chapter 5

# Results and Conclusion

## 5.1 Comparison of Light weighted Algorithms

| Lightweight Algorithm | Block Size | Key Length | GEs |
|---|---|---|---|
| HIGHT | 64 | 128 | 3048 |
| mCrypton | 64 | 64<br>96<br>128 | 2420<br>2681<br>3758 |
| SEA | 96 | 96 | 3758 |
| TEA | 64 | 128 | 2355 |
| ICEBERG | 64 | 128 | 7732 |
| CLEFIA | 128 | 128 | 2488 |
| PRESENT | 64 | 128 | 1884 |

Table 5.1 [19]

## 5.2 Comparison with this design

In this implementation of PRESENT cipher, total number of Gate Count for the entire algorithm (Encryption + Decryption) = 580 + 679 = 1259, which is less than the mentioned gate count of PRESENT in above table. So it is a more optimized implementation.

The power required by Encryption process = 84.9397 uW
The power required by Decryption process = 70.7394 uW

We can see that power required by this design is also very less, so it is suitable for low power devices used in IoT.

## 5.3 Future implementation

In this work we have seen classical cryptography approach towards IoT Security. Simultaneously if we develop processes to secure IoT devices at Network level itself then it will strengthen the security for these devices.

Following are some advantages of Network Level Security:

- Network-level security after implementation will cover most of IoT devices. Hence it won't be specific to single device.

- The implementation of Network level security will happen in cloud and the updates can be sent to all IoT devices connected to cloud.

- Network level security can be offered as a service by a third party having expertise in this specific area, so the device manufacturer who may not have required expertise need not to invest time in it.

- Network level security is an addition to device level security, without increasing device cost.

# PART B

# Chapter 6

# Register Configuration

## 6.1 System RDL

System RDL (Register Description Language) and its compiler are designed to automate and accelerate the process of both designing and documenting everything from a single register to a large number of registers and memories, across a variety of chips or boards. The language allows designers to abstractly define and describe everything from a single register to a number of boards, each containing any number of chips with their own associated registers and memories. The language and its compiler were created with design for re-use in mind, allowing designers to create a variety of output from a single RDL source file.

The chief advantage to this approach is the automatic synchronization of the hardware design with its documentation, verification source/code environment, driver development, and C/C++ software models. This automated approach to design has in practice radically reduced the design cycle for hardware designers, hardware verification engineers, and driver developers. It has also removed much of the documentation effort and greatly improved communication of register modifications and fixes throughout the design cycle from the designers to verification and software engineers.

The language has a rich set of features to describe and implement a wide variety of registers and memories. The RDL and it's compiler have evolved from the past. The language has gone through significant change since its inception, adopting what worked best in previous designs, reworking what did not, and adding what was missing. The result of this was RDL v1.0, which was written to be a solid and highly customizable platform for current and future register development.

## 6.2 Background

In May 2009, The SPIRIT Consortium announced the release of the System RDL specification. System RDL is a language for the design and delivery of registers to be used in IP blocks within electronic designs. The System RDL 1.0 Standard was transferred to Accellera upon the merger of The SPIRIT Consortium with Accellera Organization in 2010 [20].

## 6.3 Outputs obtained by RDL

Following are the outputs obtained after compiling and executing RDL code.

| | |
|---|---|
| **CSPEC** | C-Spec formatted Word document containing a register address summary table for all spaces (MEM, MMIO, IO, CFG) as well a detailed table of each register and its fields |
| **Firmware C Header** | Generates three C-code firmware header files for each unit address map. It will contain C typedef declarations of the address maps, #define constants for each register hexadecimal offset address, and a unique "union" typedef declaration for each register containing the field name and size within the register. |
| **XML** | Generates a SPIRIT XML tree representing the addrmap hierarchy and all associated properties in the data model extracted from the RDL. |
| **OVM (for RAL)** | Generates multiple System Verilog output files which can be compiled into a validation environment containing the OVM and Saola packages. The generated output files contain class object definitions but will not compile on their own. Rather, they must be included within a testbench module, program, or package. |
| **HTML** | Generates an HTML table for documentation representing the addrmap hierarchy and all associated properties. Each level of RDL hierarchy is represented in a navigation bar on the left side with hyperlinks to the addrmap or reg declaration and related properties. |

| | |
|---|---|
| **Fuse** | Generates multiple System Verilog output files which can be compiled into a validation environment compiled with the OVM and Saola packages. The generated output files contain class object definitions but will not compile on their own. Rather, they must be included within a test bench module, program, or package. |
| **System Verilog RTL** | Generate synthesizable RTL modules in System Verilog format. The generator only parses the lowest/unit-level level addrmap and it will generate two files per unit addrmap or defined Module Name. |
| **DFx / TAP** | Generates multiple TAP-related output files for DFx pre/post-silicon verification and documentation. |
| **CRIF** | Generates a Control Register Interchange Format (CRIF) style XML tree representing the addrmap hierarchy and all associated properties in the data model extracted from the RDL. The XML structure is based on IP-XACT but includes additional attributes for content, register files, collections, MSR, and registers/fields. CRIF is an internal format to allow interchangeable register formats to import into legacy solutions CRGen and CRWebViewer |

Table 6.1

## 6.4 Hierarchies in RDL file

**Field:** A primitive hardware design element (e.g. wire, flip-flop, memory, etc...).
**Register:** A set of fields atomically accessed by software.
**Register file:** A set of registers and other register files.
**Address map:** A mapping of registers to user specified addresses.

## 6.5 Field – Purpose

Memory accessed by software may contain a single entity or a number of bit-fields each with its own meaning and purpose. In RDL each entity in a software read or write is termed as a field. In the example below, the register “simpleReg” contains a single 32-bit field, whereas the register “complexReg” contains three distinct fields (cntr, command, and flag).

```
// Field Definitions
field simpleField {sw = rw; hw = rw; }; // read+write field
field counterField {sw = r; counter;}; // read-only counter
```

```
field writeonlyField {sw = w; }; // software write-only

// Register Definitions
reg simpleReg {
simpleField data[31:0]; // single 32-bit field
};

reg complexReg {
counterField cntr[31:16]; // 16-bit counter
writeonlyField command[2]; // 2-bit software write-only field
writeonlyField flag; // 1-bit software write-only field
};
```

## 6.5.1 Field Properties – Access

hw = (rw|r|w|na); Design's ability to sample/update a field
sw = (rw|r|w|na); Programmer's ability to read/write a field

## 6.6 Registers – Purpose

In RDL a register is defined as a set of one or more RDL field instances that are atomically accessible by software at a given address. A register definition specifies its width and the types and sizes of the fields that fit within that width (address allocation is handled by the register file and address maps).

```
field myField { sw=rw; hw=rw; };
reg myReg {
myField a; // single bit, assigned bit position 0
myField b[3]; // 3-bit array, assigned bits [3:1]
myField c[5:5]; // single bit, designer placed at bit 5
myField d[9:6]; // 4-bit array, designer placed at bits 6-9
};
```

## 6.6.1 Registers – Internal Instantiation

Register components instantiated using standard component instantiation syntax shown above are internal registers. Unless a register is meant to

represent a RAM, ROM, TCAM, or some other complicated memory system, the register should be instantiated as an internal register.

### 6.6.2 Registers – External Instantiation

Complex memories (SDRAM, ROM, RAMBUS, TCAM, etc...) are defined in terms of fields and registers, but should be instantiated as external registers. Syntactically registers are instantiated as external registers if the keyword **external** is placed before the register type name. For example:

```
reg myReg { field {} data[31:0]; };
external myReg extReg; // single external register
external myReg extArray[31:2]; // external register array
```

### 6.6.3 Register Properties

name = “full name”; Replaces instance name in HTML or SGML/FM.
desc = “description”; Appears below name in HTML or SGML/FM.
regwidth = 32; Specifies register width, must be power of two.
accesswidth = 16; Specifies software access width (power of two).
errextbus = Generate error input for external instances.

The **name** and **desc** properties are for documentation purposes. These properties are identical to the field properties by the same name.
The **regwidth** (was **width** before, soon to be deprecated) property determines the maximum bit-width of the register. The width must be a power of two, and must be at least 8-bits wide.
The **accesswidth** property determines the minimum width software operation that may be performed on a register. By default the **accesswidth** is identical to the width of the register that is by default no sub-word access is allowed. The **accesswidth** must be a power of two, and must be at least 8-bits wide. The **accesswidth** may not exceed the width of the register.

The **errextbus** property specifies that when instantiated as an external register an additional error input pin will be generated. If the error is asserted at the posedge of any clock cycle during an outstanding software access the transaction is cancelled and an invalid access error is asserted.

All instances of any register defined with the errextbus property must be instantiated as an external register.

## 6.7 Register Files – Purpose

A register file in RDL is defined as a logical grouping of one or more register and register file instances (similar to the ANSI-C "struct" construct). The register file provides some address allocation support, which is useful when there is a need to introduce an address gap between registers. Designers are encouraged to leave the majority of address allocation to the address map, as it makes the register file code more portable, reusable, and easier to read (refer to the address map documentation for details on address allocation/specification).Register files are typically used when a group of registers should be treated as a unit by software.

For example a set of registers that control a fifo:

```
regfile fifoRfile {
reg pointerReg { field {} data[31:0]; };
reg fifoStatusReg {
field {} full;
field {} empty;
};
pointerReg head;
pointerReg tail;
fifoStatusReg status;
};
```

## 6.8 Address Maps – Purpose

An address map in RDL maps registers, register files and address maps to either virtual or final addresses. During HDL generation, it is the address map that is converted into a design module in HDL. All registers and fields instantiated within a register file are generated within this module. Any address maps instantiated within an address map are interfaced with forming a hierarchical tree of design modules.

```
reg myReg { field {} data[31:0]; };
addrmap {
myReg std; // standard RDL component instantiation syntax
myReg vaddr @0x300; // virtual address 0x300 applied
myReg arrayIncr[100] += 8;
myReg arrayVaddr[50] @0x200 += 0x10;
};
```

## 6.8.1 Address Map Properties

name = “full name”; Replaces instance name in HTML or SGML/FM
desc = “description”; Appears below name in HTML or SGML/FM
alignment; Specifies alignment of all instantiated components
sharedextbus; Forces all external registers to share buses

The **name** and **desc** properties are for documentation purposes.
If a register, register file, or address map instance is not explicitly assigned an address, the compiler will assign an address automatically. By default the address will be aligned to the width of the component being instantiated (i.e. the address of a 64 bit register will be aligned to the next 8-byte boundary). The **alignment** property allows designers to override the default address alignment. All alignments must be a power of two (1, 2, 4, 8, 16, etc…) and are in units of bytes.
The **sharedextbus** property allows designers to force the compiler to combine the address, read data, and write data buses for all external registers instantiated within the address map.

## 6.9 RDL Example

```
`include "lib_udp.rdl"
addrmap system_demo_1 {
 addrmap component1 {
   reg reg_demo_1 {
        name = "This is my first demo register";
        shared;
        regwidth = 64;
        RTLSeqType = "LATCH";

     field {
```

```
        AccessType = "RW";
        RTLSeqType = "FF";
        } f1[16:1] = 16'h0;

      field {
        AccessType = "RW";
        RTLSeqType = "FF";
        } f2[17:17] = 1'b0;

      field {
        AccessType = "RW";
        RTLSeqType = "FF";
        } f3[21:20] = 2'b00;

       };

     bridge = true;
      addrmap  {
           Space = "MEM";
           BaseAddress = "0x8000";
           reg_demo_1 reg1;
           reg1->AliasAddress = "0x100";
           reg_demo_1 reg2;
           reg2->AliasAddress = "0x110";
           reg_demo_1 reg1;
           reg1->AliasAddress = "0x200";
           reg_demo_1 reg2;
           reg2->AliasAddress = "0x210";
       } intF1;


    addrmap  {
           Space = "CFG";
           BaseAddress = "1/20/0";
           AliasAddress = "0x700";
           reg_demo_1 reg1;
           reg_demo_1 reg2;
           reg2->AliasAddress = "0x10";
      } intF2;

    };
   component1 comp1;
};
```

## 6.10 Actual Register Definition

The following example discusses in detail the actual register definition. The example shows a 16-bit register with some additional bit field features regarding side effects and reset value exceptions. The example also demonstrates attribute overriding of fields vs. the enclosing Struct.
The following figure shows this register definition:

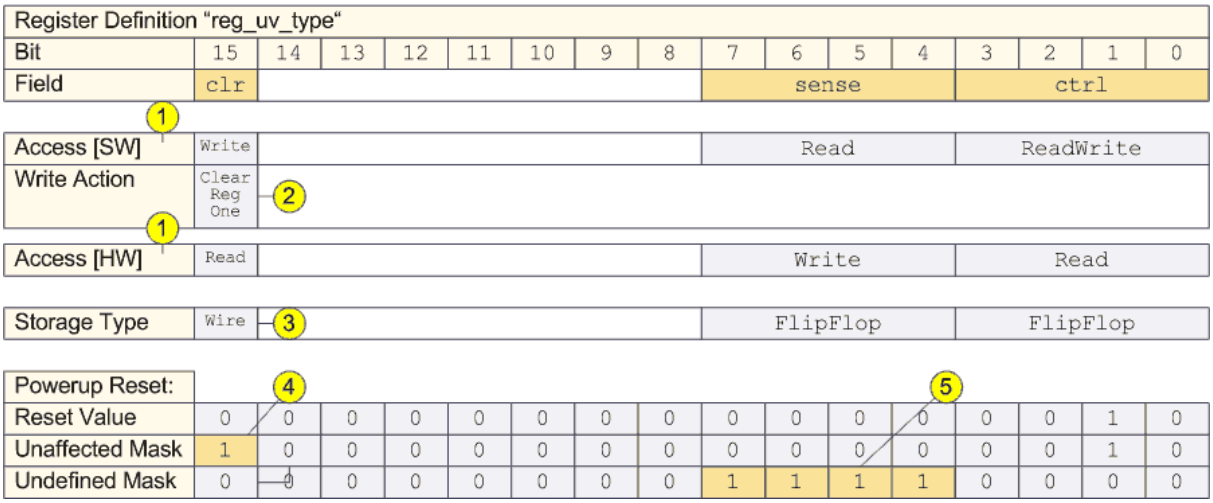


Figure 6.1: Register Field Configuration [21]

We assume that there are two valid access agents for this register type - The Software side (i.e., over the bus interface) and the Hardware side (from component kernel). The register type features the following fields:

- A "ctrl" field which is implemented with Flip-flops and is a standard SW ReadWrite field (ReadOnly from HW side) and is used to control the module.
- A "sense" field which would contain values directly from some sensor and is SW read-only. As this field reflects the sensor readout, the reset value on power-up cannot be predicted - it is undefined.
- A "clr" field which will clear the whole register when a "1" is written to it. It is WriteOnly from SW and will not be affected by a reset

## 6.11 Annotations

(1) In this example there are only two: SW and HW. Both can have independent properties.

(2) For the "clr" field, there is no value physically stored. The field is implemented by a signal going to the peripheral kernel. Thus, the "Storage Type" (i.e., physical implementation) for this field is set to "Wire".

(3) As the "clr" field is not affected by a reset (i.e., its value does not change on reset), the "Unaffected Mask" is set to one for its position.

(4) As the "sense" field has an undefined reset value, its bit positions are set in the "Undefined Mask".

# Chapter 7

# Register Validation

## 7.1 CREST

CREST is Intel's **C**onverged **Re**gister **S**pecification **T**est [21]. It is a System Verilog/OVM-based test sequence that leverages Saola RAL (Register Abstraction Layer) to perform basic control register validation.

## 7.2 How does it work?

The diagram below shows CREST within a Saola/OVM Test environment.

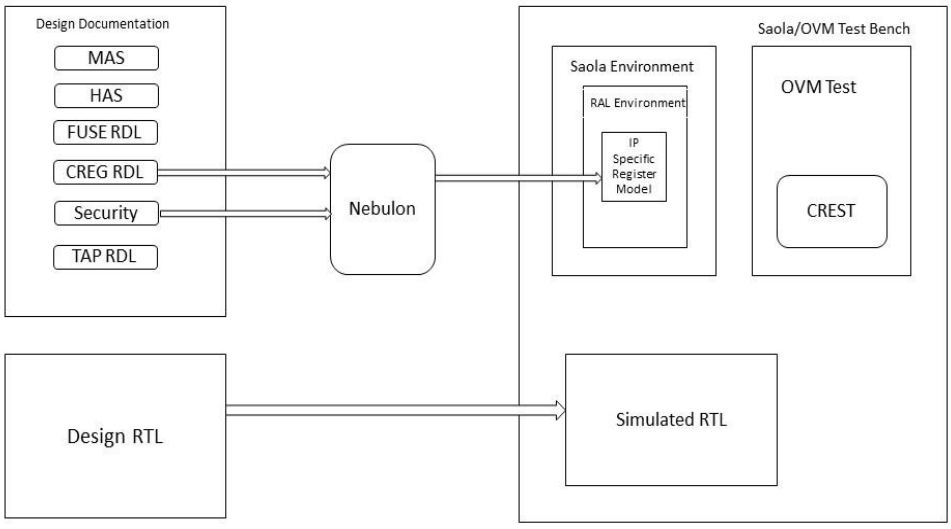


Figure 7.1: Block Diagram - Register Validation

The CREST sequence is called from a wrapper OVM Test residing in the Saola/OVM environment. CREST first queries the RAL Environment to discover the set of registers to be tested. It then commands RAL to perform a sequence of reads and writes, which RAL implements by sending transactions to the simulated RTL. The responses received from the RTL are checked against the register models in the RAL Environment.

By thoroughly testing the registers documented in the RAL Environment, CREST builds confidence in the equivalence between the IP-specific Register Model in RAL and the simulated RTL. Because the IP-specific Register Model is auto-generated from the CREG RDL and security.pm file, CREST also raises confidence that the Design RTL matches the register specifications (i.e., CREG RDL and security.pm).

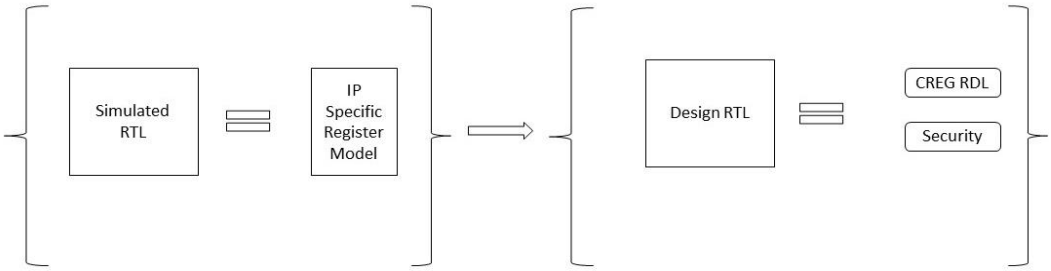


Figure 7.2: Design Specification Equivalency

## 7.3 CrestParser

CREST provides a acerun log parser which can provide simplified transaction list from the original log.
The goal of the parser is to produce a simplified transcript of all the operations performed during the test along with other test config information to aid in debug activities. Example snapshot of acerun & crest transcript file below.

## 7.3.1 Modes of operation

**Single file:** Parses a single file and produces the transcript file along with the summary information for a single test.
**Multiple files:** Parses multiple crest log files and produces a transcript file for each input log, at the end it provides a combined summary for all the inputs.

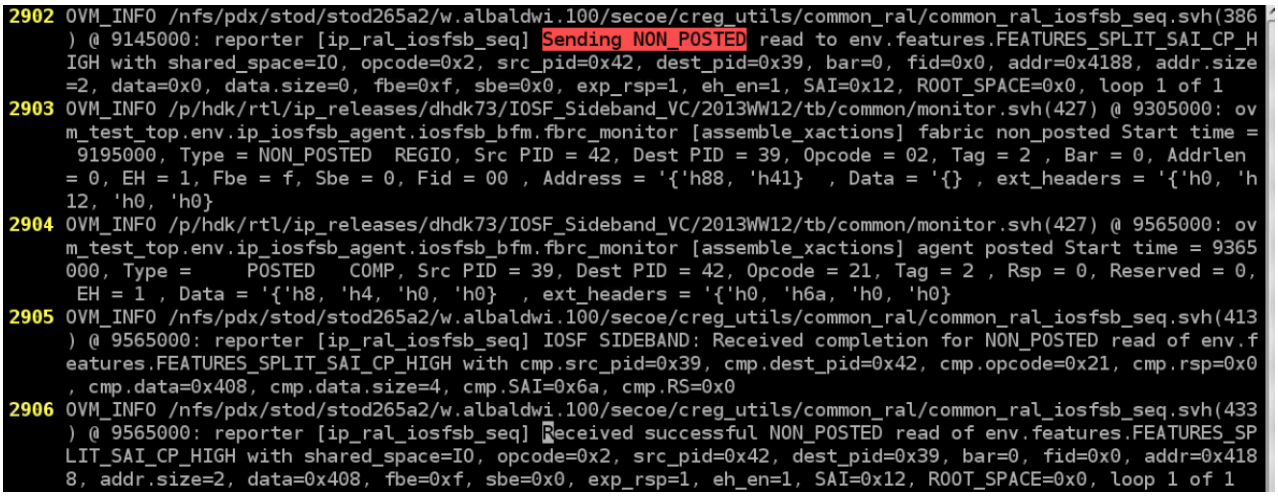


Figure 7.3: Log file Output [21]

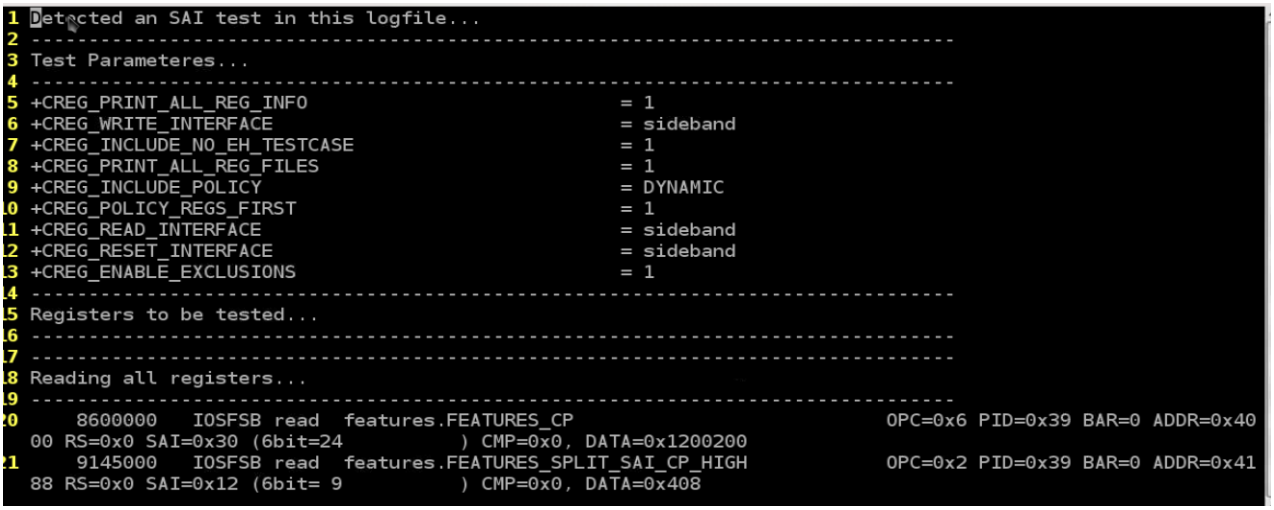

Figure 7.4: CREST Parser Script Output [21]

### 7.3.2 Output files produced by the parser

***.transcript file** : For each input file a transcript file is produced it contains test configuration & transcript logs for the test.
***.exclude :** For each input file a excluded file is produced if there are any fields/ registers / regfiles excluded.
**crest.log :** For each parser run this is the stdout of the parser.
**crest_logs/<test_name>.failing_regs.log :** the parser produces a *.failing_regs.log for each test. This file contains the failing registers for that test.
**crest_logs/<test_name>.passing_regs.log :** the parser produces a *.passing_regs.log for each test. This file contains the passing registers for that test.
**crest_logs/<test_name>.not_tested.log :** the parser produces a *.not_tested.log for each test. This file contains the registers that were not tested in this test.
**crest_logs/<test_name>.coverage.log :** the parser produces a *.coverage.log for SAI test only. This file captures the functional coverage information of parser for SAI test.

# Bibliography

[1] K. Finkenzeller, *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*. Hoboken, NJ, USA: Wiley, 2003.

[2] A. Juels and S. A. Weis, “Authenticating pervasive devices with human protocols,” in *Advances in Cryptology*. Berlin Germany: Springer-Verlag, 2005, pp. 293–308.

[3] National Institute of Standards and Technology (NIST). (Nov. 26, 2001). *Advanced Encryption Standard (AES)*, Federal Information Processing Standards Publication 197. [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[4] National Institute of Standards and Technology (NIST). (Dec. 30, 1993). *Data Encryption Standard (DES)*, Federal Information Processing Standards Publication 46-2. [Online]. Available: http://www.umich.edu/~x509/ssleay/fip46/fip46-2.htm

[5] National Institute of Standards and Technology (NIST). (October 25, 1999). *Data Encryption Standard (DES)*, Federal Information Processing Standards Publication 46-3. [Online]. Available: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[6] A. Poschmann, G. Leander, K. Schramm, and C. Paar, “New light-weight crypto algorithms for RFID,” in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2007, pp. 1843–1846.

[7] T. Eisenbarth and S. Kumar, “A survey of lightweight-cryptography implementations,” *IEEE Des. Test. Comput.*, vol. 24, no. 6, pp. 522–533, Nov./Dec. 2007.

[8] T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata, “The 128 bit blockcipher CLEFIA,” in *Fast Software Encryption* (Lecture Notes in

Computer Science), vol. 4593, A. Biryukov, Ed. Berlin, Germany: Springer-Verlag, 2007, pp. 181–195.
[9] *The 128 Bit Blockcipher CLEFIA: Algorithm Specification*, Sony Corporation, Tokyo, Japan, 2007.

[10] D. Hong *et al.*, "HIGHT: A new block cipher suitable for low-resource device," in *Cryptographic Hardware and Embedded Systems* (Lecture Notes in Computer Science), vol. 4249, L. Goubin and M. Matsui, Eds. Berlin, Germany: Springer-Verlag, 2006, pp. 46–59.

[11] L. Brown, J. Pieprzyk, and J. Seberry, "LOKI—A cryptographic primitive for authentication and secrecy applications," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 453, J. Pieprzyk and J. Seberry, Eds. Berlin, Germany: Springer-Verlag, 1990, pp. 229–236.

[12] F.-X. Standaert, G. Piret, N. Gershenfeld, and J.-J. Quisquater, "SEA: A scalable encryption algorithm for small embedded applications," in *Smart Card Research and Applications* (Lecture Notes in Computer Science), vol. 3928, J. Domingo-Ferrer, J. Posegga, and D. Schreckling, Eds. Berlin, Germany: Springer-Verlag, 2006, pp. 222–236.

[13] D. J. Wheeler and R. M. Needham, "TEA, a tiny encryption algorithm," in *Fast Software Encryption* (Lecture Notes in Computer Science), vol. 1008, B. Preneel, Ed. Berlin, Germany: Springer-Verlag, 1994, pp. 363–366.

[14] F.-X. Standaert, G. Piret, G. Rouvroy, J.-J. Quisquater, and J.-D. Legat, "ICEBERG : An involutional cipher efficient for block encryption in reconfigurable hardware," in *Fast Software Encryption*, B. Roy and W. Meier, Eds. Berlin, Germany: Springer-Verlag, 2004, pp. 279–298.

[15] A. Bogdanov *et al.*, "PRESENT—An ultra-lightweight block cipher," in *Cryptographic Hardware and Embedded Systems* (Lecture Notes in Computer Science), vol. 4727, P. Paillier and I. Verbauwhede, Eds. Berlin, Germany: Springer-Verlag, 2007, pp. 450–466.

[16] B. Collard and F.-X. Standaert, "A Statistical Saturation Attack against the Block Cipher PRESENT," in *proceedings of CT-RSA*, 2009.

[17] M. R. Z'aba, H. Raddum, M. Henricksen, and E. Dawson, "Bit-Pattern Based Integral Attack," in *Kaisa Nyberg, editor, FSE*, vol. 5086 of Lecture Notes in Computer Science, pages 363–381. Springer, 2008.

[18] M. Albrecht, and C. Cid, "Algebraic Techniques in Differential Cryptanalysis," in *proceedings of FSE*, 2009.

[19] G. Bansod, N. Raval, and N. Pisharoty, "Implementation of a New Lightweight Encryption Design for Embedded Security," in *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 1, Jan 2015, pp. 142-151.

[20] http://www.eda.org/activities/working-groups/systemrdl

[21] Intra Intel Network.

[22] O. Özen, K. Varıcı, C. Tezcan, and Ç. Kocair, "Lightweight Block Ciphers Revisited: Cryptanalysis of Reduced Round PRESENT and HIGHT" in Boyd C., González Nieto J. (eds) *Information Security and Privacy* (Lecture Notes in Computer Science) vol 5594. Springer, Berlin, Heidelberg, 2009.