Crypto Engine for Secure communication in IoT chipsets.

Course No. : ESZG628T – Project Work

by:

Sarang Pramod Choudalwar

2020ht01012

Project Work carried out at:

Qualcomm India Pvt. Ltd. Bangalore
(Submitted in Partial fulfillment of Mtech. in Embedded System Program)

Under the supervision of

Sunil Pillai, Senior Staff Engineer,
Qualcomm India Pvt. Ltd. Bangalore.

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
VIDYA VIHAR, PILANI, RAJASTHAN - 333031.

(April-2022)

CERTIFICATE

This is to certify that the Dissertation entitled " **Crypto Engine for Secure communication in IoT chipsets** " is submitted by **SARANG PRAMOD CHOUDALWAR,** having ID number – 2020ht01012, for the partial fulfillment of the requirement of **M.tech. Embedded System** degree from BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE PILANI (RAJASTHAN) embodies his bonafide work under my supervision.

Place: Bangalore

Date : 25/04/2022

Signature of Supervisor

(Sunil Pillai)

Senior Staff Engineer,

Qualcomm India Pvt. Ltd. Bangalore

ACKNOWLEDGMENT

I express my deep gratitude to my Supervisor Sunil Pillai, Additional Examiner Kaustubh Sarwate, and Professor Sainath Bitragunta for all the technical guidance, constant encouragement, and enormous support provided for carrying out my project work.

I want to offer special thanks to Professor Vineet Garg, Professor Pawan Sharma, Professor Meetha Shenoy, Professor Indranil Sengupta, and Professor Onur Motlu, without their expertise and knowledge sharing session, this project work would not have been possible.

I also want to express my sincere gratitude to all my family members and friends for their extreme individual care and everlasting moral support.

Sarang Pramod Choudalwar
2020ht01012

ABSTRACT

With the increase in IoT devices worldwide, the security of IoT devices has become a matter of concern for many. While most of the focus has been on IoT network security, one must understand that IoT devices are often physically accessible. There is every possibility that the firmware of the IoT devices could be re-programmed to bring the entire network down. While it is not possible to put each IoT device in a secured location with 24x7 monitoring, the IoT chipset could be built with various security mechanisms built-in. The IoT devices are often small and are usually operated under stringent power constraints.

It has now become a necessity to support different security mechanisms like,

- Encipherment
- Digital signature
- Access control
- Data integrity
- Authentic Exchange
- Notarization

These mechanisms are required at the network level and at every chip level or even within the chip. The project aims at implementing some of these mechanisms as a part of a trust management entity using a hardware-software codesign approach. As a part of this project, a basic cryptographic algorithm for random number generation, hashing, stream cipher, and block cipher is implemented in both software and hardware. This project attempt to promote the hardware-software codesign approach by comparing hardware and software-based implementation of the algorithm on different parameters like Power, Performance, and Area.

Broad Academic Area of Work: Embedded Product Security

Keywords: Cryptography, RISC-V, FPGA, SOC-Design, Hardware-Software Codesign.

Signature of Student	Signature of Supervisor
Name: Sarang Pramod Choudalwar	Name: Sunil Pillai
Place: Bangalore	Place: Bangalore
Date: 25th April 2022	Date:25th April 2022

Contents

List of Figures:

1. Background and Introduction

Modern IoT SOC (system on chip) consists of multiple processing units like Bluetooth, WIFI, Power, DSP, and Application. Each of these processing units, also known as subsystems, is usually agnostic of each other's presence on the chip but would like to communicate with each other using a secured communication channel.

Each subsystem could be categorized into proprietary subsystem/core and customer subsystem/core. The proprietary subsystem runs core algorithms and technology, which companies like Qualcomm would like to protect from its customer. Even a passive attack like snooping poses a high-level security threat.

A snooping attack on proprietary core could be very well possible using a JTAG/USB port which would have been made open for the customer to debug their core. As customer cores are connected to the shared communication bus, malicious firmware on customer cores could also pose a security threat.

Attacks like Denial of Service are possible by running malicious software on the customer core or even jamming the wireless sensor network remotely. This gives a need for a robust encryption algorithm for data protection.

In use cases like over-the-air upgrades of proprietary technology, it is necessary to have data integrity checks and the digital signature. Not having these capabilities could pose a significant security threat.

Often, the individual subsystem is connected with external flash memory, and it is required that the subsystem execute the instructions only from the external authorized memory. This gives the need for a robust authentication mechanism.

If external RAM is connected to the individual subsystem, having access control of the region it has access to becomes critical.

Requirements like this, along with many other use-cases, gave the need for a trust management entity that could provide security services to other subsystems on the chip. The Crypto engine is the core part of the trust management entity. My project aims at building the same on hardware.

2. Objectives

The objectives of my project are as follows:

- Understand different types of commonly used encryptions and signing algorithms in detail.
- Design and implement DES/3-DES, SHA-512, and RSA hardware block on FPGA.
- Design and implement a communication interface between the crypto engine and host microprocessor/microcontroller.
- Firmware development for host micro-processor/micro-controller.
- Validation of host microprocessor/microcontroller firmware and communication interface.

3. Scope of Work

The scope of this project is as follows,

- Build a crypto engine for a trust management entity of a chipset. The crypto engine should be able to support symmetric encryption/decryption algorithms like DES/3-DES, hashing algorithms like SHA-512, and digital signature and key exchange algorithms, an algorithm like RSA should be supported. (Small FPGAs like Spartan-3/Artix-a7 could be used for implementing the hardware blocks.)
- For the SHA-512 hardware block, the host is responsible for indicating the length of data it wants to hash generated on (which is assumed to be multiple of 1024 block.)
- The host core is responsible for generating and sharing the DES-key(64bit) algorithm like RSA could be used as a key exchange algorithm.
- Some serial communication interface could expose the services from the crypto enginece. (USART/I2C/SPI could be a communication interface between the host and crypto engine.)
- The host controller/processor should be able to use this communication interface to exercise the crypto services. (Small embedded application core-based out of ARM/RISCV technology could be used.)

4. Wireless sensor network and IoT security.

The IoT network can be broadly classified into the following main categories,

- Wireless sensor networks.
- Vehicular networks.
- Industrial networks.

Each category IoT network has various constraints and has different performance capabilities. Each of the devices following in this category does have limitations in different capacities in terms of memory, power, performance, cost, flexibility, and upgradability. The focus area for the project is IOT chipsets which will be used in the wireless sensor networks. In wireless sensor networks, chip active current and time duration for which device is ON play a significant role in the complete lifetime of the device. Often, these devices are placed in remote places (many times places where human intervention is next to impossible.)

Traditionally wireless sensor networks were being made using motes. The diagram below shows the basic building block of a typical wireless mote.

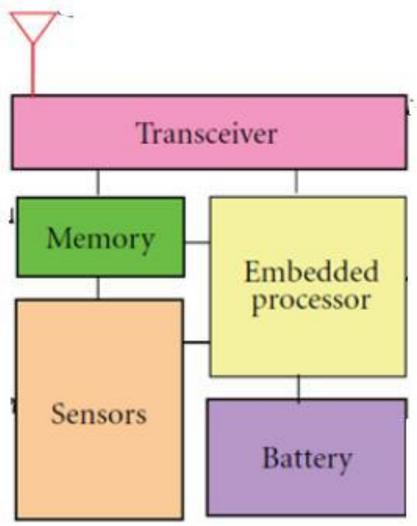


Figure 1. Embedded Node/ Mote

The building blocks of mote are sensor, Embedded processor, battery, memory, and a trans receiver radio unit. The embedded processor is capable of running all the security algorithms as required.

Figure 2. Telos Motes- Crossbow (Source:- Google images)

The wireless sensor motes/devices are usually operated on a battery, and the power consumption of a device becomes quite critical. There is typically a tradeoff between the performance and power consumption of the device. Most commercially available motes do not have security mechanisms inbuilt into the hardware, but they rely entirely on software implementation of the security algorithms.

Though the software security algorithm ensures flexibility and upgradeability of the security algorithms, they do come at the cost of running the embedded processor's extended duration of time. Software security mechanisms do not offer the best power optimization mechanisms.

One of the discouraging factors for implementing security at the hardware level is cost and area. Having a security mechanism in hardware requires a mote to be modified to accommodate additional integrated circuits.

Traditionally almost all the motes are built around COTS (Custom off-the-shelf CPU). They do offer significantly fewer unit costs. These standalone embedded processors usually do not provide application-specific security mechanisms.

One could consider developing own ASIC for a specific application, but traditionally it was challenged by,

1. High NRE cost
2. High licensing fees for CPU cores like ARM.
3. High licensing cost for tools used for ASIC development.
4. Lack of expertise.

Things are gradually changing around the world; with the rise in open-source RISC-V cores, open-source EDA tools, and open-source communities like OSFPGA, it has become relatively easy for even small-scale organizations to develop and manufacture ASICs keeping the NRE cost at a minimum.

SiFive, the pioneers of RISC-V core, have claimed that developing end-to-end chips with RISCV cores can be done within $100K.(https://www.sifive.com/blog/custom-chips-for-under-100k)

Rise of open-source CPU cores like RISCV along with open-source EDA development tools like sky wafer-pdk (https://www.skywatertechnology.com/blog/how-to-design-with-a-free-skywater-pdk/) from google along with significant initiatives like "tape out world" from the community like OSFPGA (https://osfpga.org/) – it has become possible now to develop a custom ASIC which could offer high security, better performance less power consumption in comparison with traditional COTS-based design.

Gradually the ASICs for wireless sensor network is resulting in renewed interest in the hardware-software co-design approach (https://www.sciencedirect.com/topics/engineering/hardware-software-codesign) where one is not bounded by capabilities offered by COTS-based system. The main objective of this project/dissertation exploring the possibilities offered by open-source platforms and develop and compare a set of security mechanisms using both software and hardware approach and then identify the best place to keep security mechanisms (hardware/software.)

5. The Crypto engine

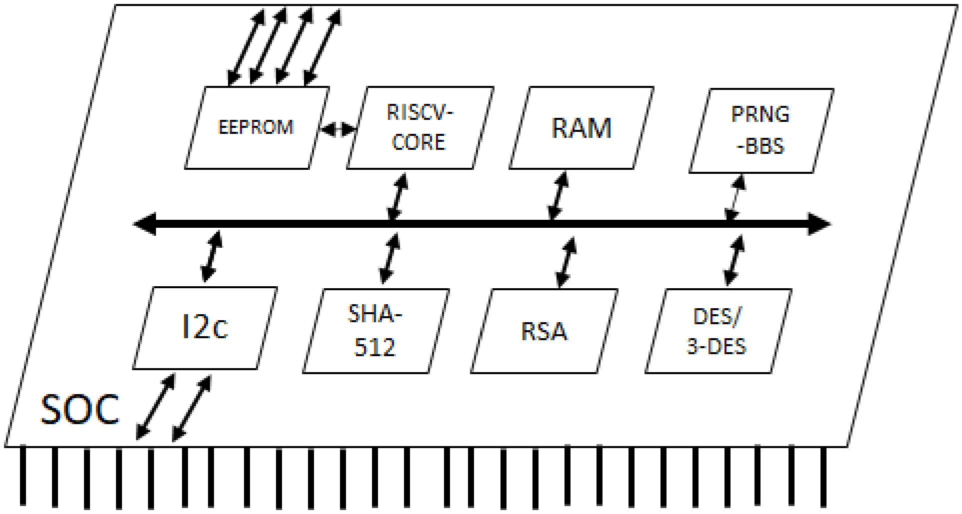


Figure 3. Block diagram of proposed SOC

The figure above shows the proposed building soc block diagram, offering security in the embedded processing unit itself.

5.1 RISC-V core :

The RISC-V core is the central processing unit for the SOC.RISC-V is open source CPU architecture – a different version of the RISC-V core can be selected depending upon the application the SOC is being made. In order to reduce development, cost many companies like SIFIVE, In-core, and C-DAC offer different RISC-V, which could be procured directly from them. The various cores support various capabilities.

Considering the time-limitation on the proposed project design and development of the RISC-V core is descoped.

5.2 Pseudo Random Number Generator (PRNG) :

PRNG is mainly used in many crypto engines for generating random numbers. These random number helps in the key encryption and decryption process. The random number generated from them can be used for symmetric as well as asymmetric key cryptography. Apart from helping in a key generation, they also help in identifying and avoiding reply attacks by appending themselves in the handshaking process. A random bit stream generated could also be used for symmetric key encryption – where both sender and receiver have a shared secret (which is a seed in this case).

A widely used BBS algorithm is decided to be used for random number generation.

5.2.1 Blum Blum Shub (BBS) algorithm:

- First, choose two large prime numbers, suppose p and q such that both have a remainder of 3 when divided by 4.
- Let n = p*q
- Now select the random seed value. One has to make sure s is relatively prime to n.
- BBS generator uses the following algorithm to generate random number Xi and random bits Bi according to the following algorithm

X0 = s^2 mod n
for i=1 to ∞
{
    Xi = $(Xi-1)^2$ mod n
    Bi = Xi mod 2
}

In order to check the performance improvement when using the software/hardware approach, it has been decided to implement the algorithm in both hardware and software.

The following snippet of images showcases the current progress made on the BBS algorithm.

5.2.1.2 Software implementation of BBS algorithm:

- Software used:- Windows subsystem for Linux
- The language used:- C
- The compiler used:- GCC
- Debugger used:- GDB

- Snapshot of the output of BBS algorithm software implementation: -

root@DESKTOP-MAQ414N: /mnt/c/Mtech/Sem_4/software/9_mar_2022

```
root@DESKTOP-MAQ414N:/mnt/c/Mtech/Sem_4/software/9_mar_2022# gcc -g BBS.c -o out
root@DESKTOP-MAQ414N:/mnt/c/Mtech/Sem_4/software/9_mar_2022# ./out
Enter number of random numbers needed:-
20
Output Random number is 143135
Output Random bit is 1
Output Random number is 177671
Output Random bit is 1
Output Random number is 97048
Output Random bit is 0
Output Random number is 89992
Output Random bit is 0
Output Random number is 174051
Output Random bit is 1
Output Random number is 80649
Output Random bit is 1
Output Random number is 45663
Output Random bit is 1
Output Random number is 69442
Output Random bit is 0
Output Random number is 186894
Output Random bit is 0
Output Random number is 177046
Output Random bit is 0
Output Random number is 137922
Output Random bit is 0
Output Random number is 123175
Output Random bit is 1
Output Random number is 8630
Output Random bit is 0
Output Random number is 114386
Output Random bit is 0
Output Random number is 14863
Output Random bit is 1
Output Random number is 133015
Output Random bit is 1
Output Random number is 106065
Output Random bit is 1
Output Random number is 45870
Output Random bit is 0
Output Random number is 137171
Output Random bit is 1
Output Random number is 48060
Output Random bit is 0
root@DESKTOP-MAQ414N:/mnt/c/Mtech/Sem_4/software/9_mar_2022#
```

Figure 4. BBS algorithm running in software

5.2.1.3 Hardware implementation of BBS algorithm

- EDA tool used:- EDA playground.
- Simulation tool :- Icarus Verilog 0.9.7
- Synthesis tool:- Yosys 0.9.0
- HDL Language used:- Verilog
- Snapshot of BBS implementation test done in Verilog

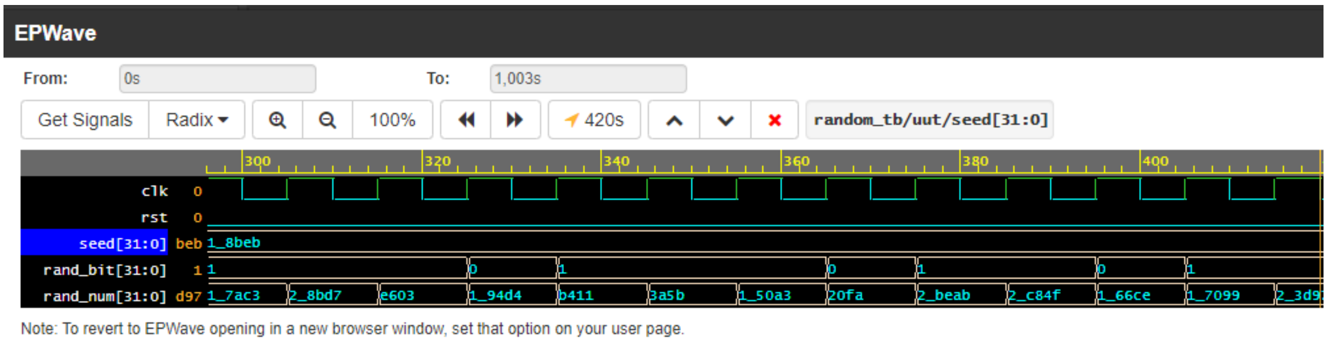


Figure 5. BBS implementation hardware simulation

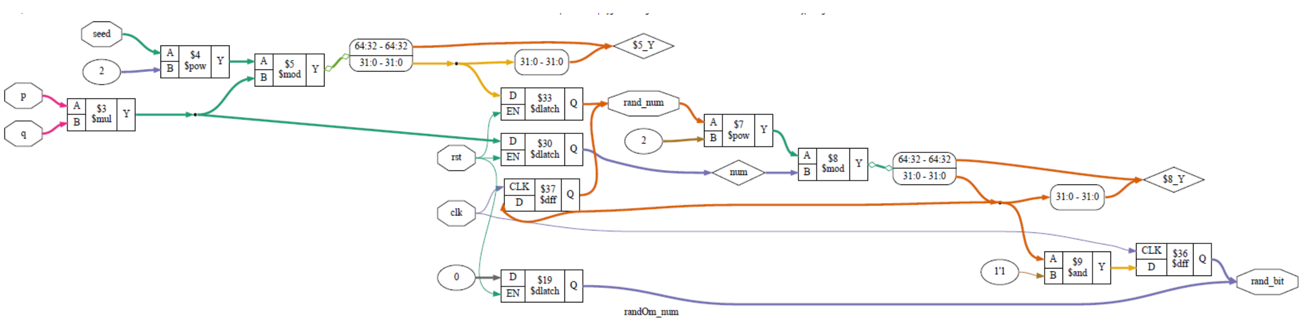


Figure 6.BBS synthesized hardware for FPGA

5.3 RSA Algorithm:

RSA is one of the most widely accepted and implemented general-purpose approaches for public-private key encryption. RSA is mainly used to protect small data chunks. The product secrets / random number seed or symmetric key is usually encrypted using the RSA algorithm. The RSA algorithm consists of 3 parts given below,

- Key Generation.
- Encryption
- Decryption

5.3.1 Key Generation in RSA :

- Select two numbers, p, and q, where p and q are coprime.
- Calculate n=p*q
- Calculate ø(n) =(p-1)*(q-1)
- Select integer e such that gcd(ø(n),e)=1; 1<e<ø(n)
- Calculate d such that d= e^{-1}mod ø(n)
- Public key (PU) = {e,n}
- Private key (PR) ={d,n}

5.3.2 Encryption in RSA:

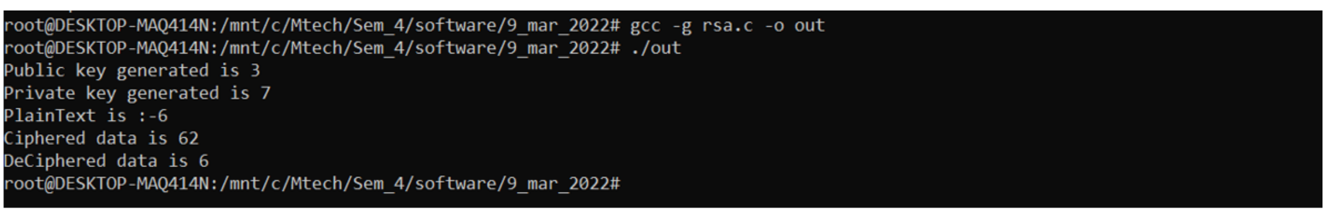
- Plaintext : P < n
- Ciphertext = C =$P^e \ mod \ n$

5.3.3 Decryption in RSA:-

- Ciphertext: C
- Plaintext= P = $C^e mod \ n$

5.3.4. Software implementation for RSA algorithm :

- Software used:- Windows subsystem for Linux
- The language used:- C
- The compiler used:- GCC
- Debugger used:- GDB
- Snapshot of software implementation RSA.



```
root@DESKTOP-MAQ414N:/mnt/c/Mtech/Sem_4/software/9_mar_2022# gcc -g rsa.c -o out
root@DESKTOP-MAQ414N:/mnt/c/Mtech/Sem_4/software/9_mar_2022# ./out
Public key generated is 3
Private key generated is 7
PlainText is :-6
Ciphered data is 62
DeCiphered data is 6
root@DESKTOP-MAQ414N:/mnt/c/Mtech/Sem_4/software/9_mar_2022#
```

Figure 7. Software implementation of RSA

5.3.5. Hardware implementation of RSA algorithm :

- EDA tool used:- EDA playground.
- Simulation tool :- Icarus Verilog 0.9.7
- Synthesis tool:- Yosys 0.9.0
- HDL language used:- Verilog
- Snapshot of RSA implementations done in Verilog

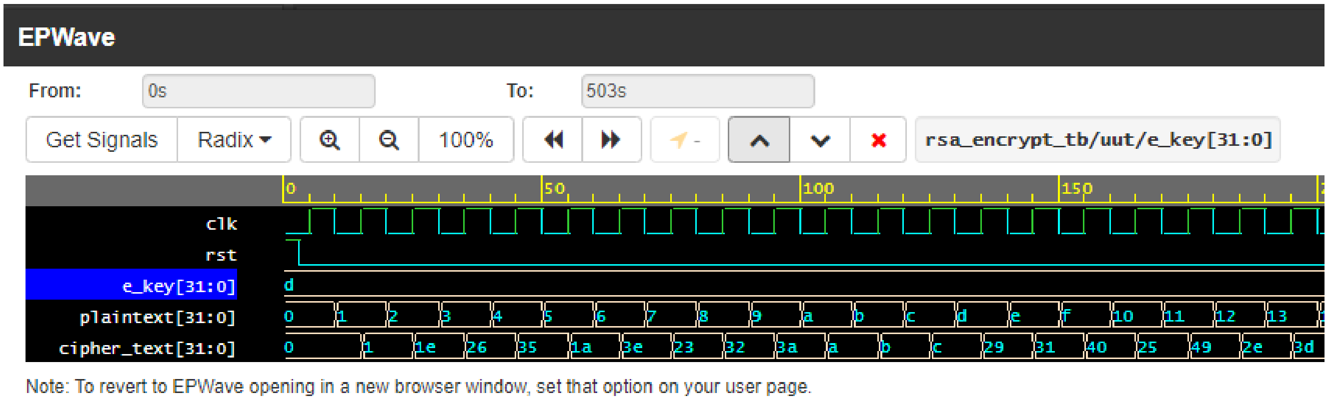


Figure 8: RSA Encrypt hardware simulation

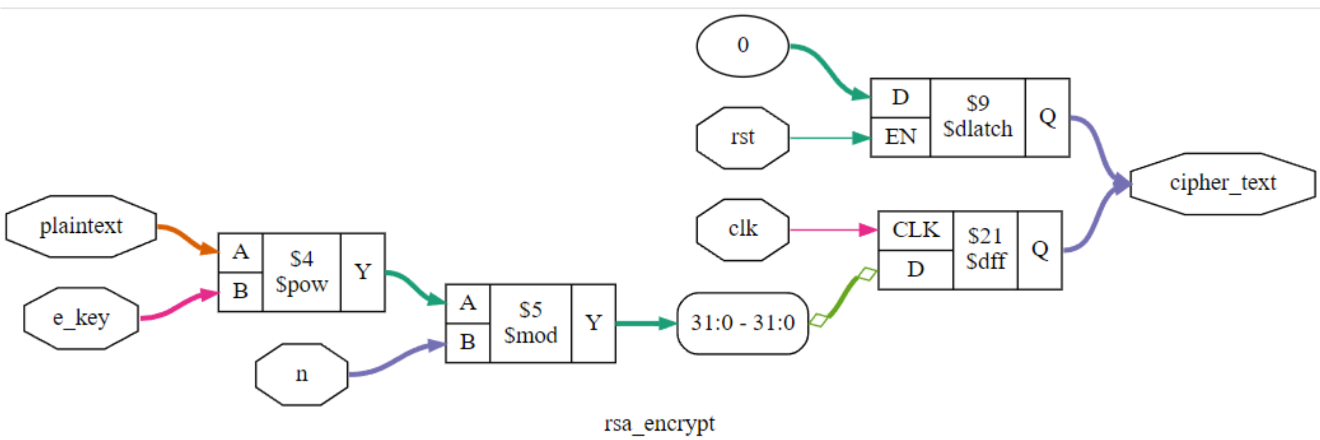


Figure 10.RSA encryption synthesized hardware

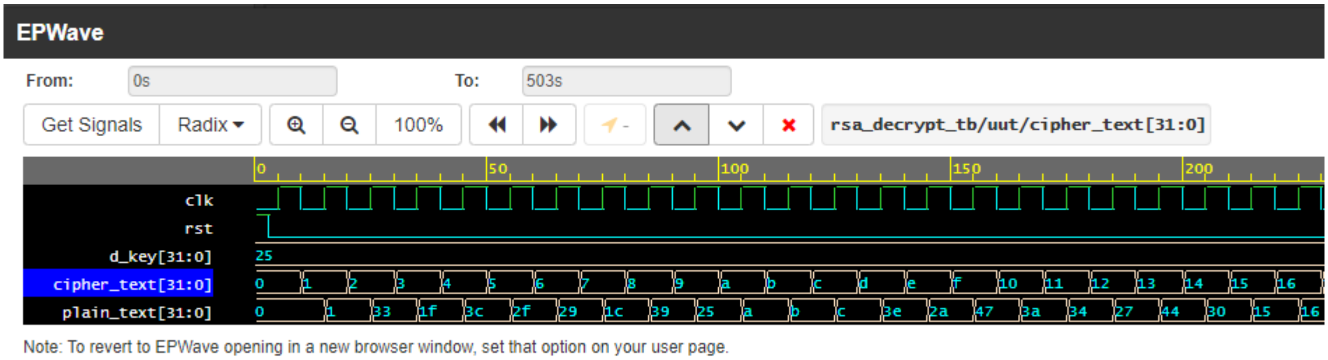


Figure 12: RSA Decrypt hardware simulation

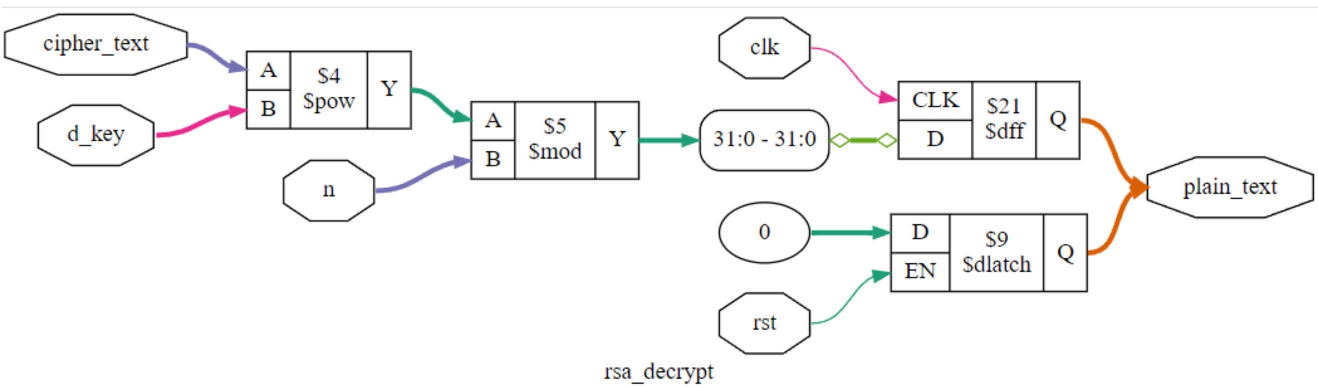


Figure 11: RSA Decrypt synthesized hardware

5.4 SHA-512:

There could be a different attack on the integrity of the data received over the IoT network. The integrity attack could be one of the following,

- Modification: - Some portion of the data is altered.
- Masquerading: - One entity pretends to be a different entity – for example, rouge wireless mote trying to get information by pretending to be a part of its network.

Hashing function usually takes a block of m-size data and produces a fixed value hash code. The hash code generated by the hashing function is always unique, and data integrity can be easily checked by comparing hash code at both transmitter and receiver ends. A good hash function is expected to produce random and evenly distributed hash code of the same size.

SHA-512 is one of the most widely used hash function, and many commercial establishments still use SHA-512.

5.4.1. SHA 512 working:

SHA-2 variant specified SHA-512 protocol is being design and implemented below.SHA-512 specifies the data to be present in big-endian format, the Verilog code currently implemented is not converting big-endian format to little endian format.

5.4.1.1. Step 1: Appending padding bits:

- The input is processed in blocks of 1024 bits.
- The message is first padded so that its length is 896 (mod 1024 bits) i.e. the last block of data must contain 896 bits with padding.
- The padding consists of a single 1 bit followed by the necessary number of 0 bits.
- The length of the original message before padding in an unsigned 128-bit integer is appended, keeping the most significant byte first.
- The message is now a multiple of 1024 bits 896+128=1024, which is treated as blocks of 1024 bits each.

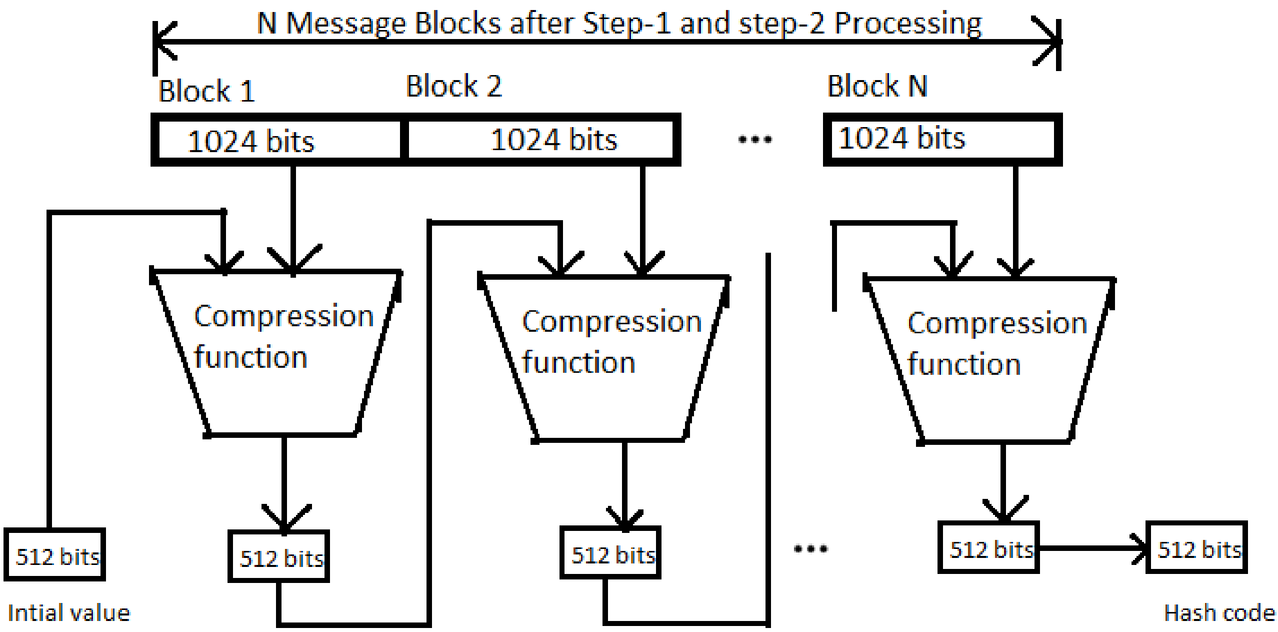
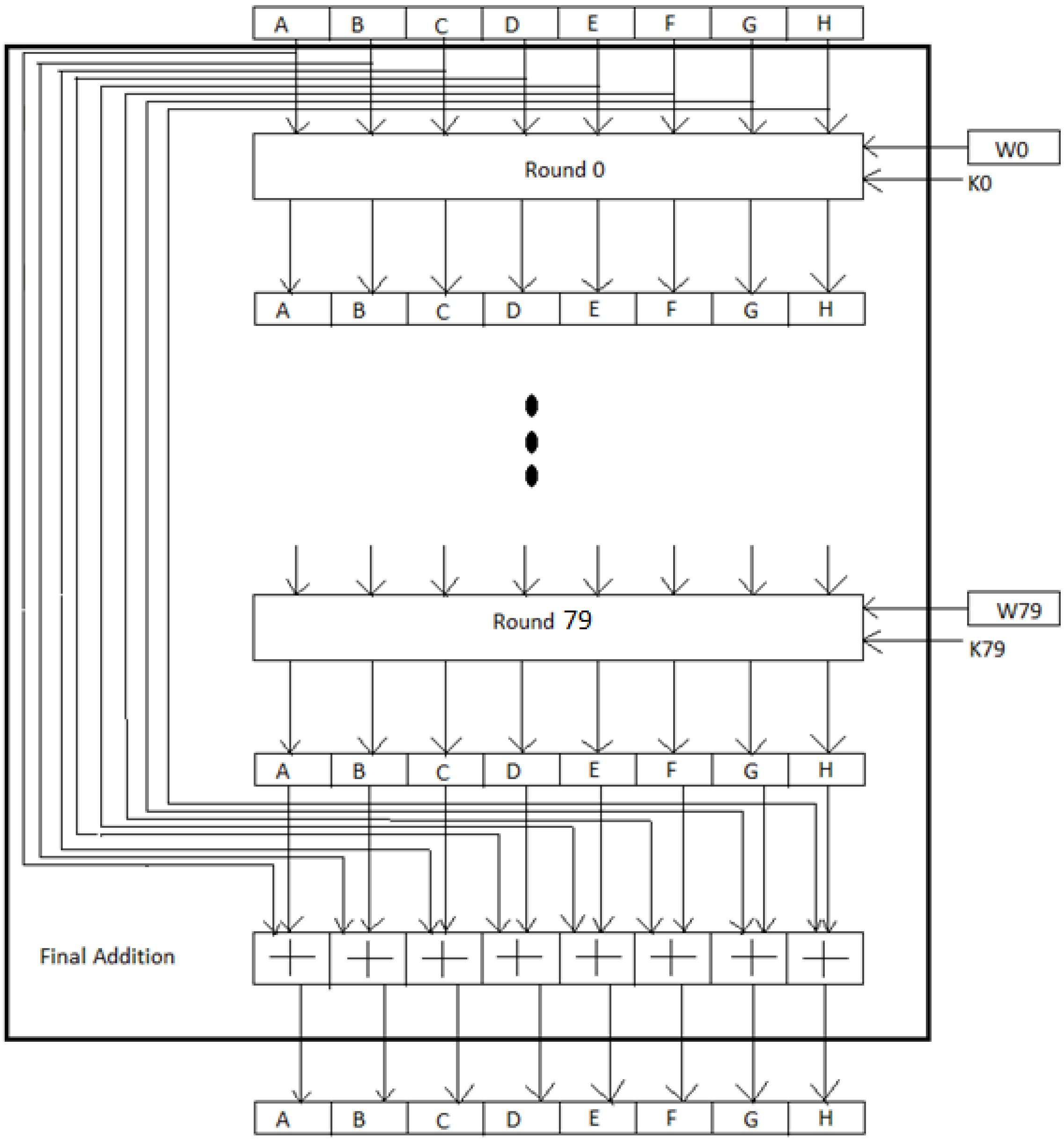


Figure 13 SHA-512 Hash calculation

5.4.1.2. Step 2:- Defining the initial value:-

- A 512-bit buffer is used to hold the intermediate and the final results of the hash function.
- A buffer can be represented as 8 64 bit registers (A,B,C,D,E,F,G,H)
- For SHA-512 initial value of these registers are fixed as given by-
- A = 0x6A09E667F3BCC908
- B = 0xBB67AE8584CAA73B
- C = 0x3C6EF372FE94F82B
- D = 0xA54FF53A5F1D36F1
- E = 0x510E527FADE682D1
- F = 0x9B05688C2B3E6C1F
- G = 0x1F83D9ABFB41BD6B
- H = 0x5BE0CD19137E2179

5.4.1.3. Step 3:-Compression function



- The compression function consists of 80 rounds.
- Input to the compression function is a 512-bit initial value or the output of another compression function.

- Each round also takes a predefined constant value, K.

428A2F98D728AE22	7137449123EF65CD	B5C0FBCFEC4D3B2F	E9B5DBA58189DBBC
3956C25BF348B538	59F111F1B605D019	923F82A4AF194F9B	AB1C5ED5DA6D8118
D807AA98A3030242	12835B0145706FBE	243185BE4EE4B28C	550C7DC3D5FFB4E2
72BE5D74F27B896F	80DEB1FE3B1696B1	9BDC06A725C71235	C19BF174CF692694
E49B69C19EF14AD2	EFBE4786384F25E3	0FC19DC68B8CD5B5	240CA1CC77AC9C65
2DE92C6F592B0275	4A7484AA6EA6E483	5CB0A9DCBD41FBD4	76F988DA831153B5
983E5152EE66DFAB	A831C66D2DB43210	B00327C898FB213F	BF597FC7BEEF0EE4
C6E00BF33DA88FC2	D5A79147930AA725	06CA6351E003826F	142929670A0E6E70
27B70A8546D22FFC	2E1B21385C26C926	4D2C6DFC5AC42AED	53380D139D95B3DF
650A73548BAF63DE	766A0ABB3C77B2A8	81C2C92E47EDAEE6	92722C851482353B
A2BFE8A14CF10364	A81A664BBC423001	C24B8B70D0F89791	C76C51A30654BE30
D192E819D6EF5218	D69906245565A910	F40E35855771202A	106AA07032BBD1B8
19A4C116B8D2D0C8	1E376C085141AB53	2748774CDF8EEB99	34B0BCB5E19B48A8
391C0CB3C5C95A63	4ED8AA4AE3418ACB	5B9CCA4F7763E373	682E6FF3D6B2B8A3
748F82EE5DEFB2FC	78A5636F43172F60	84C87814A1F0AB72	8CC702081A6439EC
90BEFFFA23631E28	A4506CEBDE82BDE9	BEF9A3F7B2C67915	C67178F2E372532B
CA273ECEEA26619C	D186B8C721C0C207	EADA7DD6CDE0EB1E	F57D4F7FEE6ED178
06F067AA72176FBA	0A637DC5A2C898A6	113F9804BEF90DAE	1B710B35131C471B
28DB77F523047D84	32CAAB7B40C72493	3C9EBE0A15C9BEBC	431D67C49C100D4C
4CC5D4BECB3E42B6	4597F299CFC657E2	5FCB6FAB3AD6FAEC	6C44198C4A475817

Figure 16. SHA-512 constants (K0 to K79 from left to right)

- The different values of K are obtained by taking the first 64 bits of the fractional part of the cubic root of the first 80 prime numbers.
- Each round also takes the value of W as input.
- The value of the last round is added to the initial value of the first round to produce 512-bit data, which acts as input to the next compression block.
- Inside each round of compression function following calculations take place,

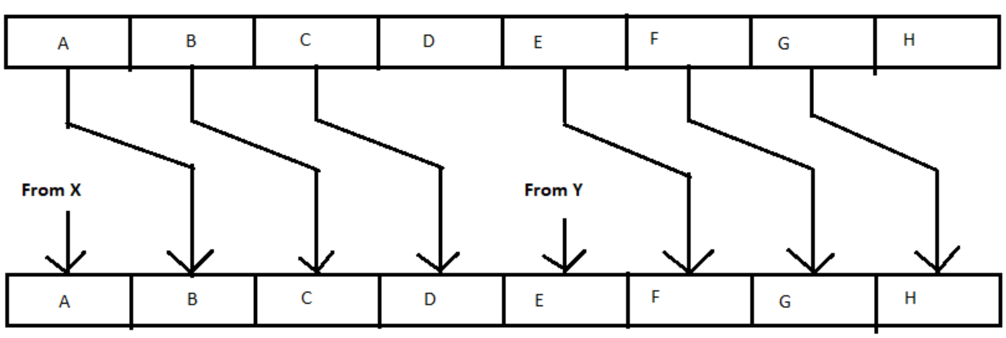


Figure 17. Internal of the round function

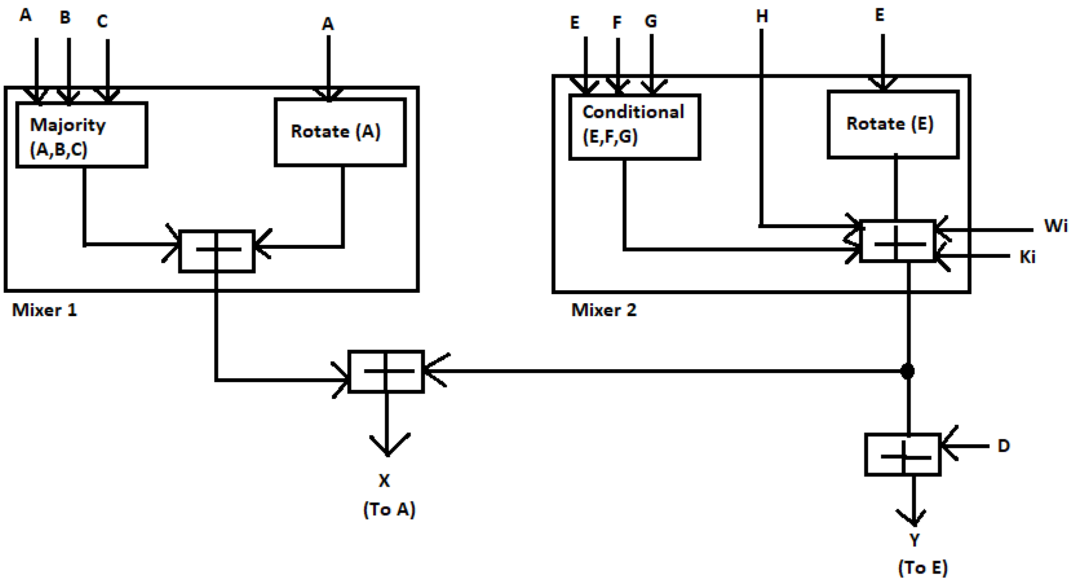


Figure 18. Internal of round function -2

- Majority(A,B,C) = (A AND B) ⊕ (B AND C) ⊕ (C AND A)
- Conditional(E,F,G) = (E AND F) ⊕ (NOT E AND G)
- Rotate(A) = $ROTR^{28}\ (A) \oplus ROTR^{34}\ (A) \oplus ROTR^{39}\ (A)$
- Rotate(E) = $ROTR^{14}\ (E) \oplus ROTR^{18}\ (E) \oplus ROTR^{41}\ (E)$
- $ROTR^{x}\ (Y)$ = circular right shift of Y by X bits.
- $Ki = Constant\ K\ value$
- Wi = A 64 bit word derived from current 1024 bit block.
- ⊕ = Ex-OR operation.
- + = Modulo addition in 2^{64} arithmetic.

5.4.1.4. Step 4:- Derivation of Wi

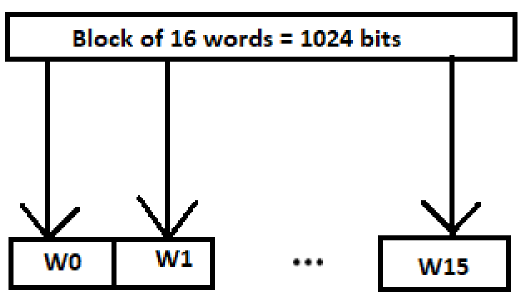


Figure 19. Calculation of values of W0 to W15

- W0 to W15 values are calculated as shown in the figure above.

- For values of word from W16 to W79 following formula is being used,
- Wi =W(i-16) + ROTSHIFT-1-8-7[W(i-15)] + W(i-7) + ROTSHIFT-19-61-6[W(i-2)]
- Here,
  - ROTSHIFT-X-Y-Z[P] = $ROTR^{X}(P) \oplus ROTR^{Y}(P) \oplus SHL^{Z}(P)$
  - $ROTR^{X}(Y)$ = Circular right shift of Y bits from X bits.
  - $SHL^{X}(Y)$ = Shift left Y by X bits.
  - $\oplus$ = Exclusive OR
  - $+$ = Module 2^{64} arithmetic.

5.4.2. Software implementation for SHA-512 algorithm**:**

- Software used:- Windows subsystem for Linux
- The language used:- C
- The compiler used:- GCC
- Debugger used:- GDB

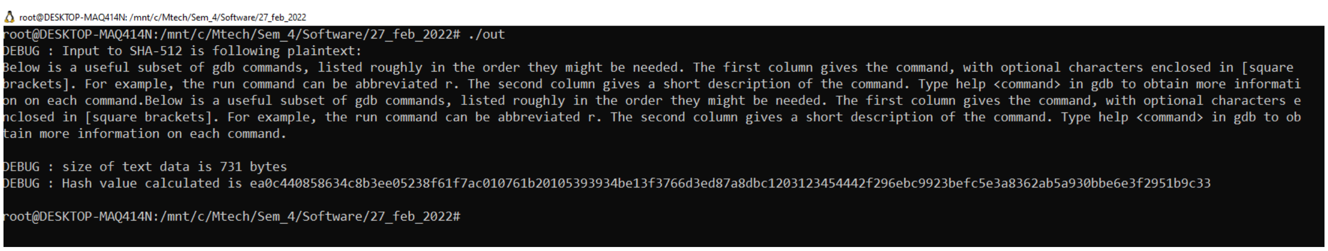


Figure 20. Snapshot of the output of SHA-512 algorithm implemented in software.

5.4.3. Hardware implementation for SHA-512 algorithm:

- EDA tool used:- EDA playground.
- Simulation tool :- Icarus Verilog 0.9.7
- Synthesis tool:- Yosys 0.9.0
- The language used:- Verilog
- The SHA-512 in hardware is implemented as different submodules bounded by the top module.
- The SHA 512 top module assumes that 1024-bit data is already available for it and it just needs to do the processing. Current implementation does not have addition of padding logic implemented.
- The different submodules are,
  - SHA-512 conditional module
  - SHA-512 constant K generator module
  - SHA-512 majority finder module
  - SHA-512 Mixer_1 module
  - SHA-512 Mixer_2 module
  - SHA-512 Rotate_A module
  - SHA-512 Rotate_E module
  - SHA-512 W generator module

- SHA-512 Round module
- SHA-512 Top module – this module ties up all the submodules.

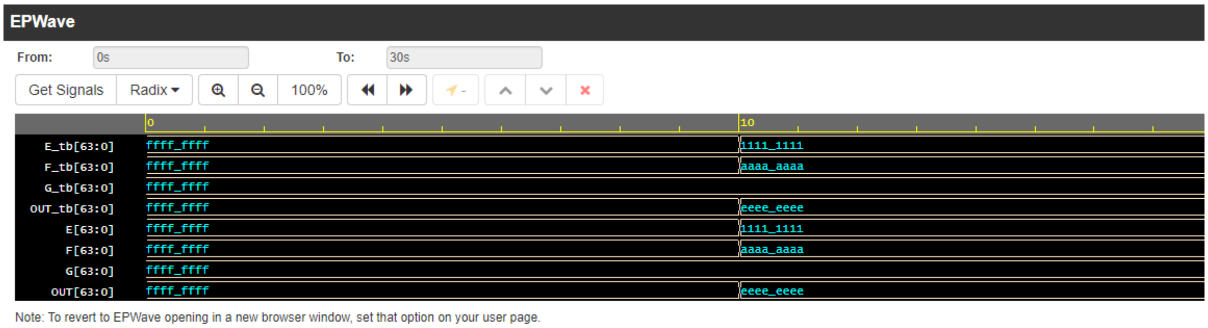


Figure 21. SHA-512 conditional module hardware simulation

5.4.3.1. SHA-512 conditional module implementation in hardware.

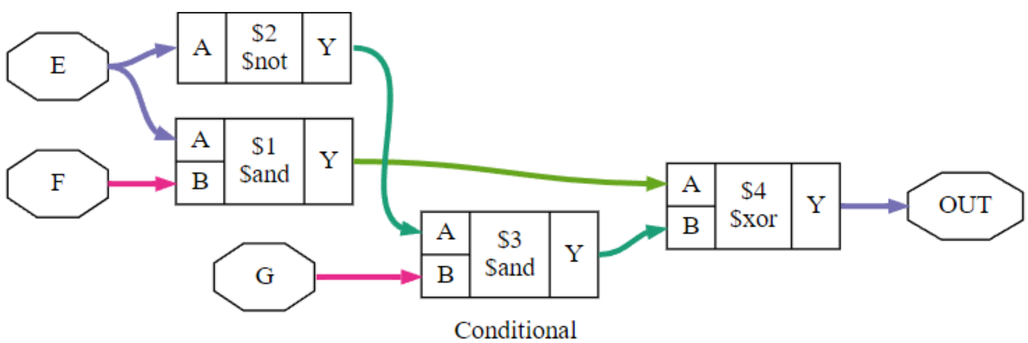


Figure 23. SHA-512- conditional module synthesized for hardware.

5.4.3.2. SHA-512 constant-K module implementation in hardware

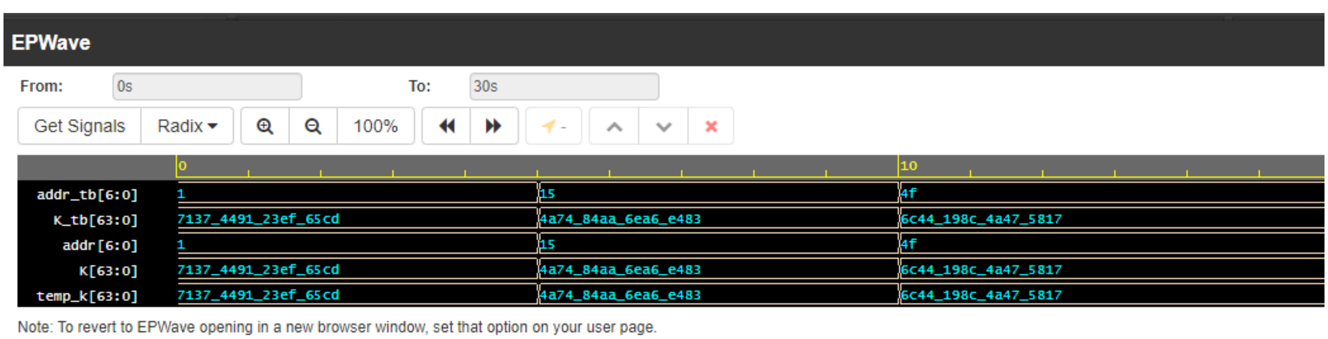


Figure 22. SHA-512 constant-K hardware simulation

- The synthesized design for K- the module generator is not shown due to the Microsoft word limitation on page structure.

5.4.3.3. SHA-512 Majority finder module implementation in hardware.

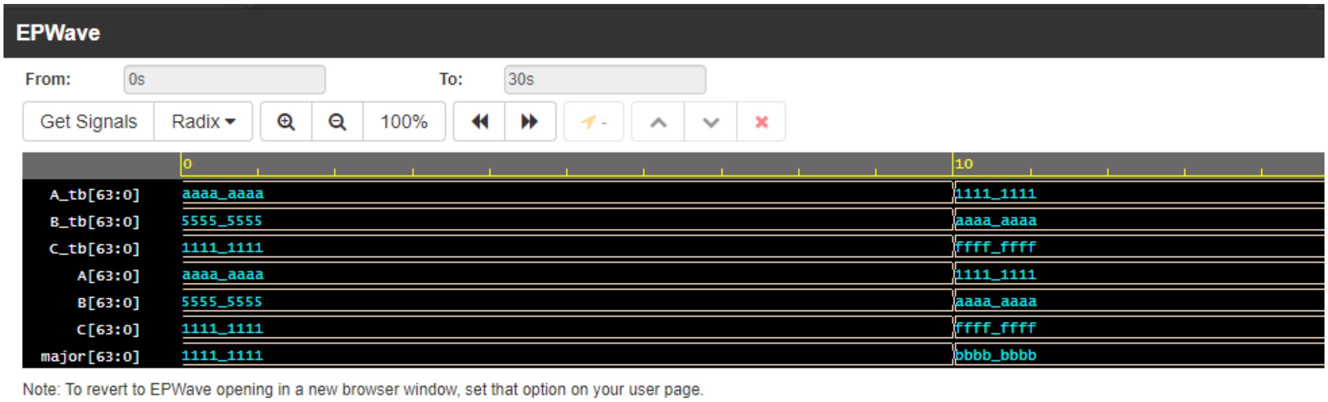


Figure 24. SHA-512 Majority finder hardware simulation

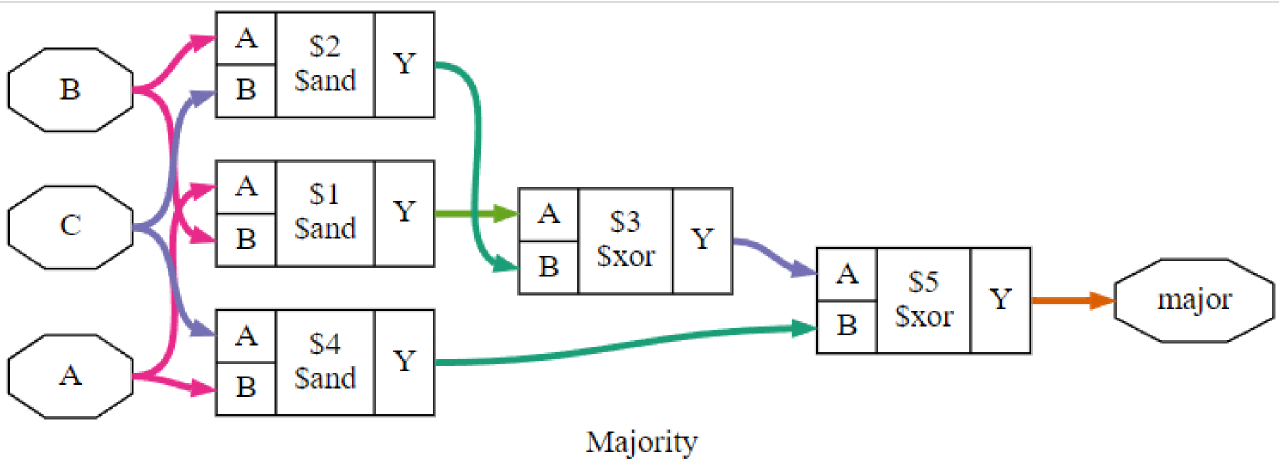


Figure 25. SHA-512 Majority finder synthesized hardware

5.4.3.4. SHA-512 ROTATE_A module Hardware implementation.

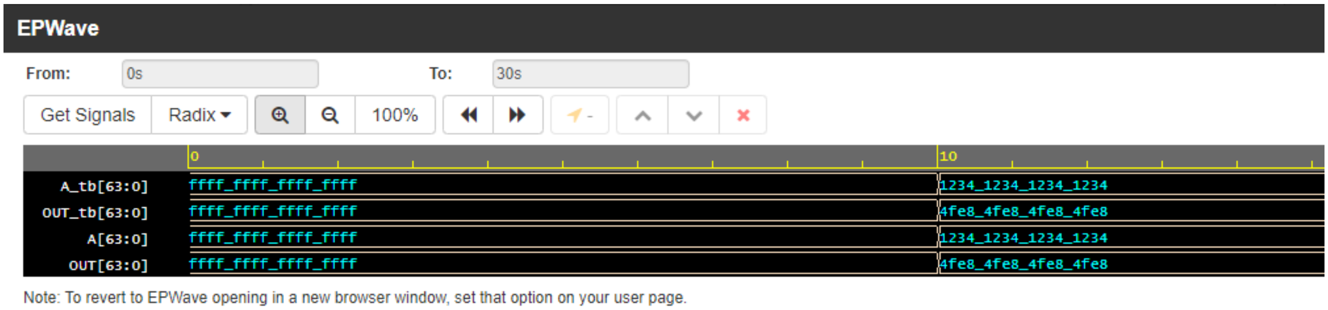


Figure 27. SHA-512 ROTATE_A module hardware implementation

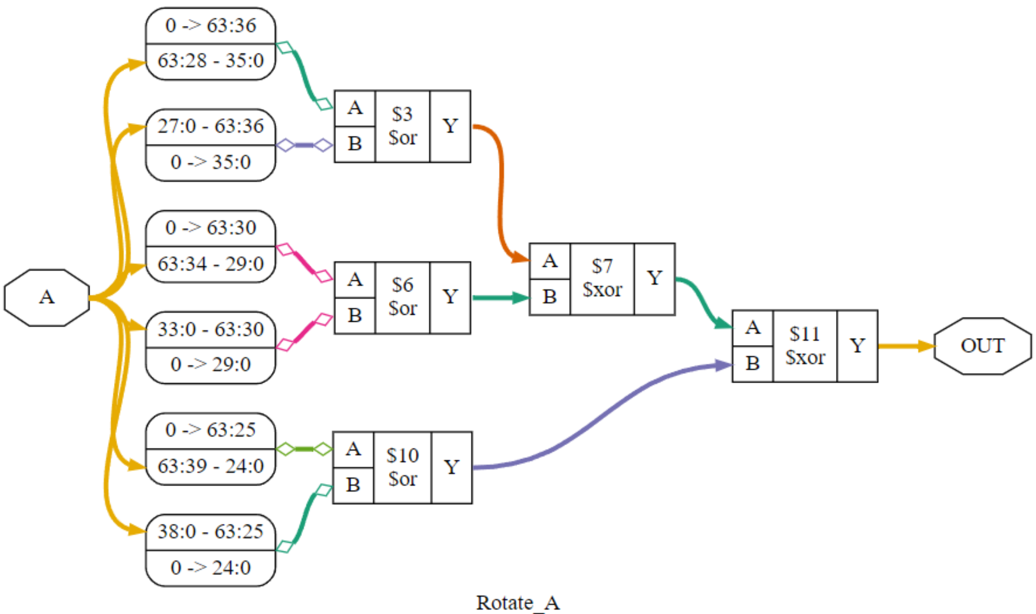


Figure 26. SHA-512 ROTATE_A module synthesized hardware

5.4.3.5. SHA-512 ROTATE_E module Hardware implementation.

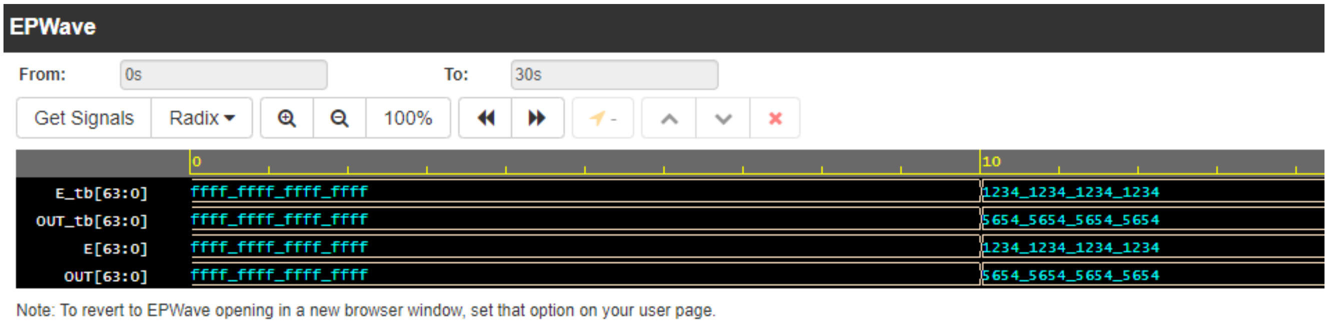


Figure 29. SHA-512 ROTATE_E module Hardware simulation

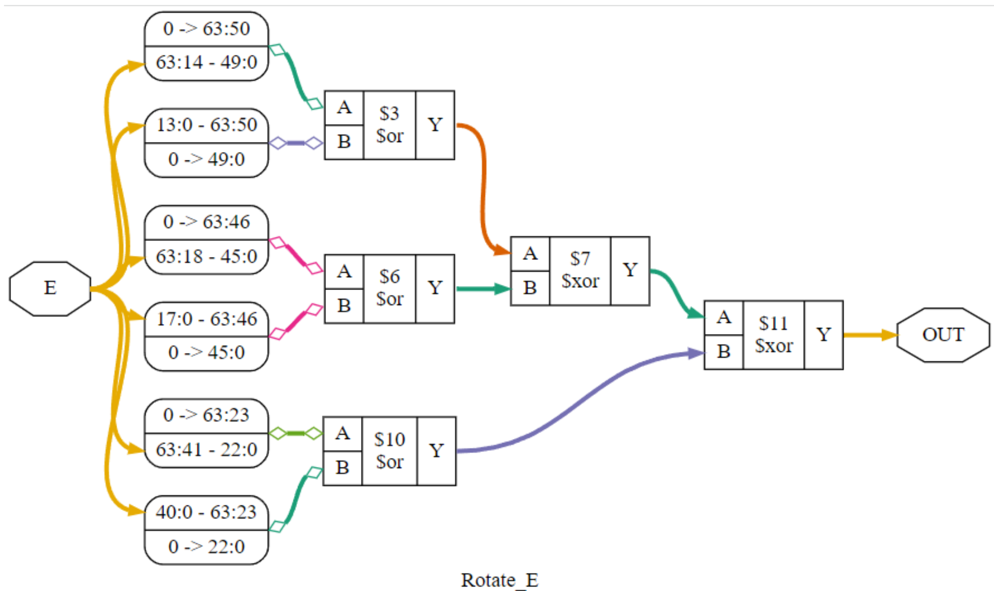


Figure 28. SHA-512 ROTATE_E module synthesized hardware

5.4.3.6. SHA-512 W value generator module hardware implementation :

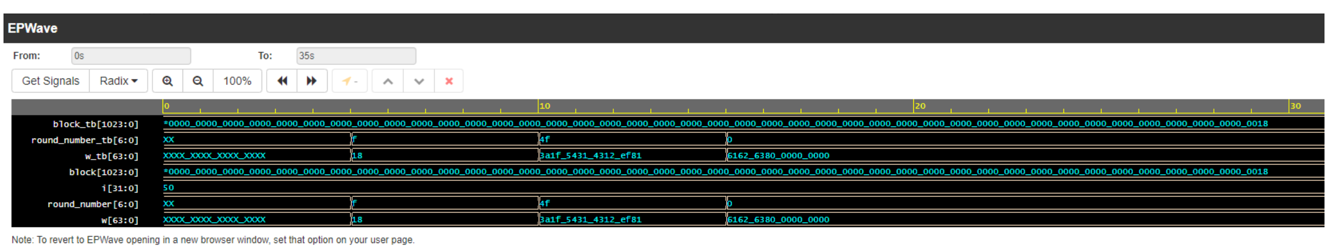


Figure 30. SHA-512 W value generator hardware simulation

The synthesized module for the W value generator is not shown due to the limitation of Microsoft word.

5.4.3.6. SHA-512 Mixer_1 module hardware implementation :

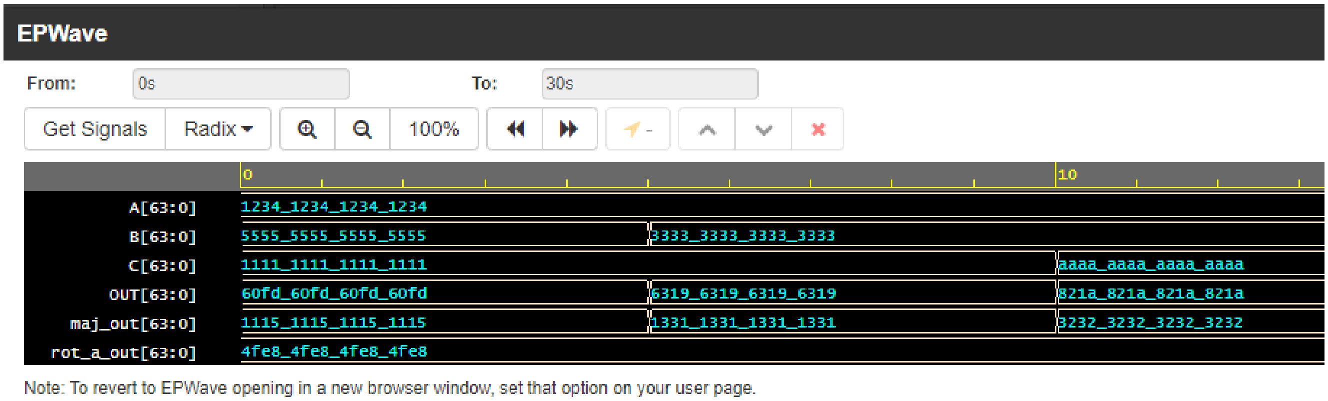


Figure 31. SHA-512 mixer-1 hardware simulation

5.4.3.7. SHA-512 Mixer_2 Module hardware implementation :

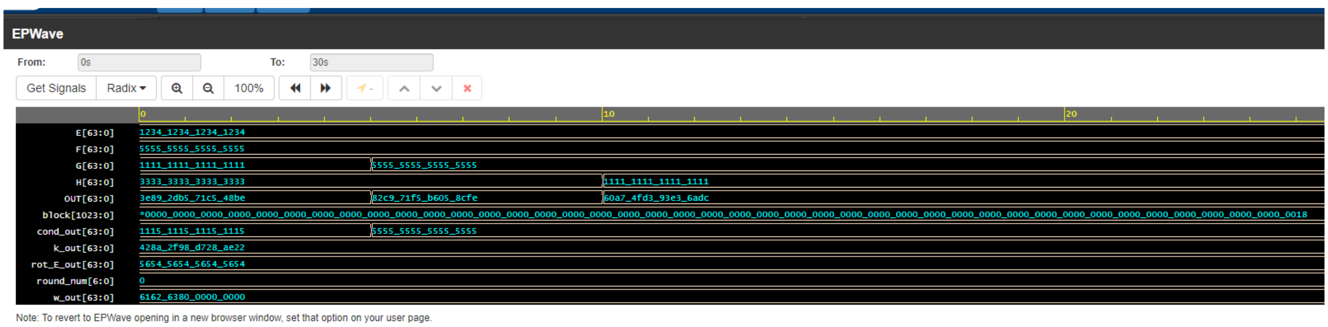


Figure 32. SHA-512 mixer_2 hardware simulation

5.4.3.8. SHA-512 Round module Hardware implementation :

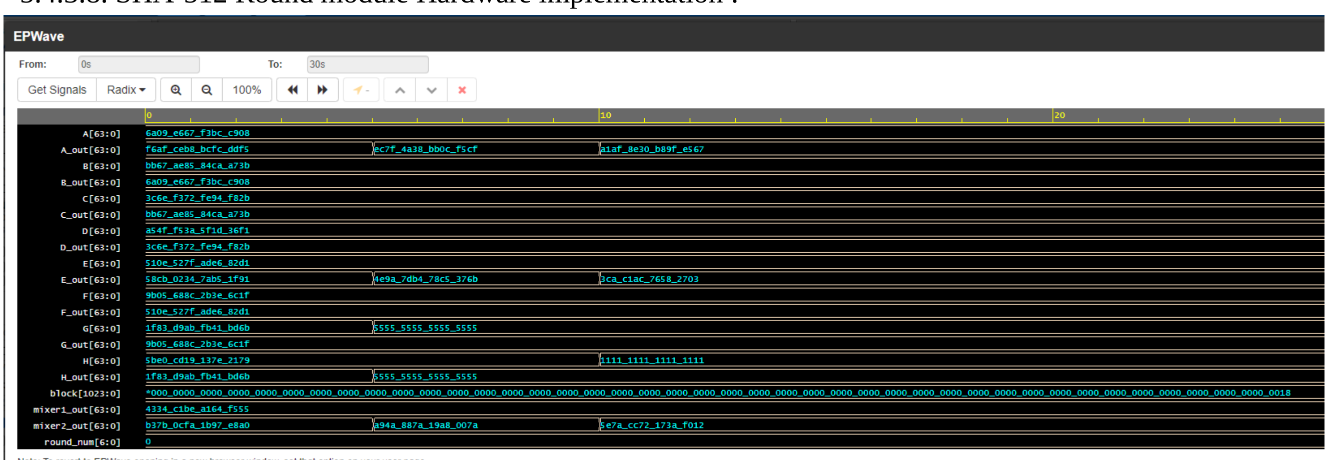


Figure 33. SHA-512 Round module hardware simulation

5.4.3.9. SHA-512 Top module Hardware implementation

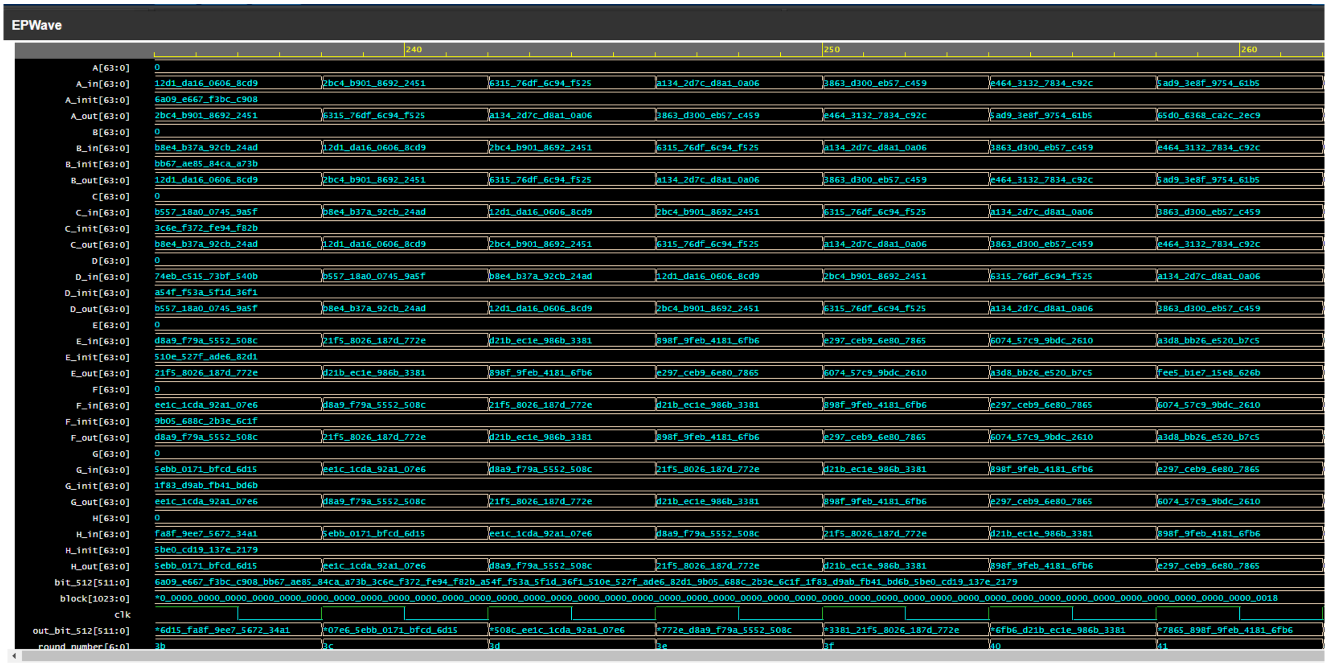


Figure 34. SHA-512 TOP module Hardware simulation

5.5 Data Encryption standard DES :

Data encryption standard is one of the most widely used block cipher techniques. The use-cases like remote firmware upgrades or information about other motes in an extensive wireless sensor network make it necessary to encrypt a large block of data in the shortest duration of time. Encryption standard like DES becomes really useful under such situation.

The DES was specified in NIST-1977, and it is based on the Feistel cipher. DES is capable of ciphering 64 bits of data at a time, and it also needs a 64-bit long key. DES works in three phases,

1. Initial permutation: - In this phase, initial rearrangement of the input bits happens.
2. 16 rounds of permutation and substitution: - after the initial permutation, 16 rounds of permutation and substitution are involved.
3. The final ciphertext is produced after swapping the 32-bit value of the 16^{th} round and passing through the final inverse permutation.

5.5.1. The initial permutation (IP) **:**

The initial permutation transposes the input plaintext into scrambled data using following matrix.

```
initial_perm[64] = { 58, 50, 42, 34, 26, 18, 10, 2,
                     60, 52, 44, 36, 28, 20, 12, 4,
                     62, 54, 46, 38, 30, 22, 14, 6,
                     64, 56, 48, 40, 32, 24, 16, 8,
                     57, 49, 41, 33, 25, 17, 9, 1,
                     59, 51, 43, 35, 27, 19, 11, 3,
                     61, 53, 45, 37, 29, 21, 13, 5,
                     63, 55, 47, 39, 31, 23, 15, 7 }
```

5.5.2 Key Transformation :

Des requires a 64-bit key to be used. The 64-bit keys are reduced to 56 bit by ignoring every 8th bit. The following array is being used to scramble the data accordingly.

```
key[56] = { 57, 49, 41, 33, 25, 17, 9,
            1, 58, 50, 42, 34, 26, 18,
            10, 2, 59, 51, 43, 35, 27,
            19, 11, 3, 60, 52, 44, 36,
            63, 55, 47, 39, 31, 23, 15,
            7, 62, 54, 46, 38, 30, 22,
            14, 6, 61, 53, 45, 37, 29,
            21, 13, 5, 28, 20, 12, 4 }
```

The 56-bit key is then divided into two halves of 28-bit each. The 28-bit data is rotated in a circular manner, either by 1 bit or 2 bits depending upon the round number. Following shift_table array is used to do the round iteration.

```
shift_table[16] = { 1, 1, 2, 2,
                    2, 2, 2, 2,
                    1, 2, 2, 2,
                    2, 2, 2, 1 }
```

Once the individual 28-bit data set is rotated, they are combined together to form 48-bit data using the following table.

```
key_comp[48] = { 14, 17, 11, 24, 1, 5,
                 3, 28, 15, 6, 21, 10,
                 23, 19, 12, 4, 26, 8,
                 16, 7, 27, 20, 13, 2,
                 41, 52, 31, 37, 47, 55,
                 30, 40, 51, 45, 33, 48,
                 44, 49, 39, 56, 34, 53,
                 46, 42, 50, 36, 29, 32 }
```

5.5.2. The Expansion permutation IP :

The 64-bit data is divided into two sets, the right and the left half, each consisting of 32 bits. For a particular round, 32-bit right half is only used. The 32 bit right half is expanded into 48-bit using the expansion mechanism shown below,

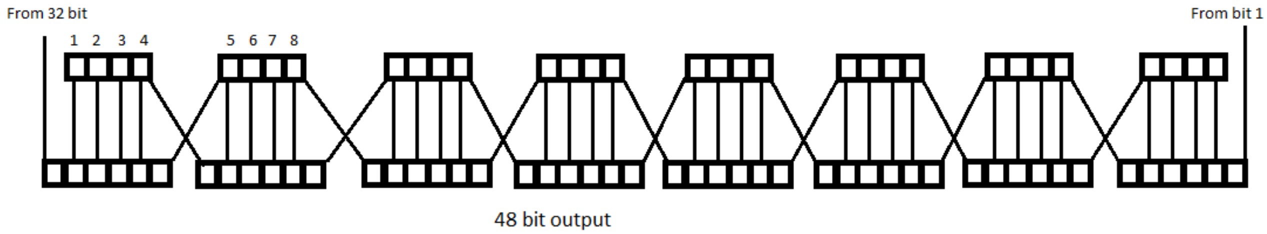


Figure 35. Expansion permutation

The output produced after expansion permutation is XORed with expanded right half data. The Output of Xor is then fed into the sbox substitution box to produce 32-bit data.

5.5.3. S-Box substitution :

The 48bit of data is divided into 8 groups where each group consists of 6 bits. The figure below shows the Xbox substitution method, which produces 48-bit data from the 32-bit input data.

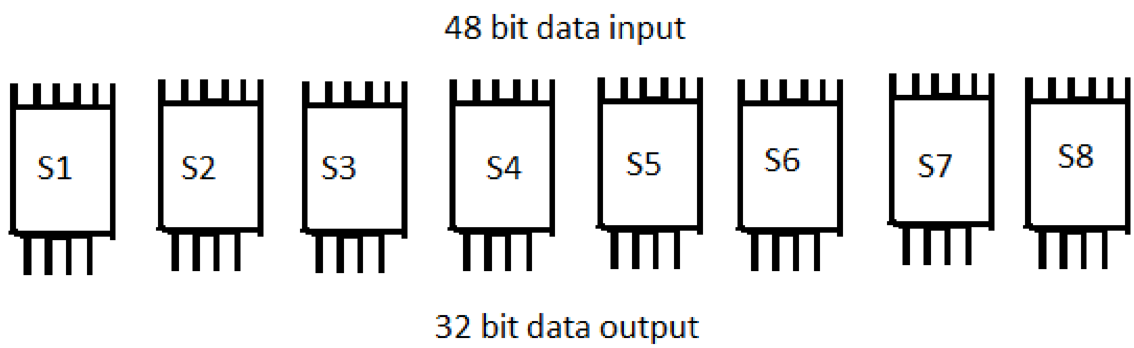


Figure 36. Xbox Substitution

The individual S-boxes computes the data by considering the 1st and 5th bit as row number, whereas the 2nd to 4th bit corresponds to column number in the predefined array. The predefined arrays are given below.

```
int s1[4][16] = {14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,
                  0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,
                  4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
                  15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 };

int s2[4][16] = { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,
                  3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
                  0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
                  13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9};
```

```
int s3[4][16]={10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,
               13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
               13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
                1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 };

int s4[4][16]={7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,
              13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
              10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
               3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 };

int s5[4][16]={2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
              14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
               4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
              11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 };

int s6[4][16]={12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
               10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
                9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
                4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 };

int s7[4][16]={4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
              13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
               1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
               6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 };

int s8[4][16]={13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
                1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
                7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
                2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 };
```

5.5.4. P-Box Permutation :

The 32-bit out of the sbox permutation is permutated using the following P-box permutation.

```
int per[32] = { 16, 7, 20, 21,
                29, 12, 28, 17,
                1, 15, 23, 26,
                5, 18, 31, 10,
                2, 8, 24, 14,
                32, 27, 3, 9,
                19, 13, 30, 6,
                22, 11, 4, 25 };
```

The entire round of DES algorithm is showcased in the below diagram,

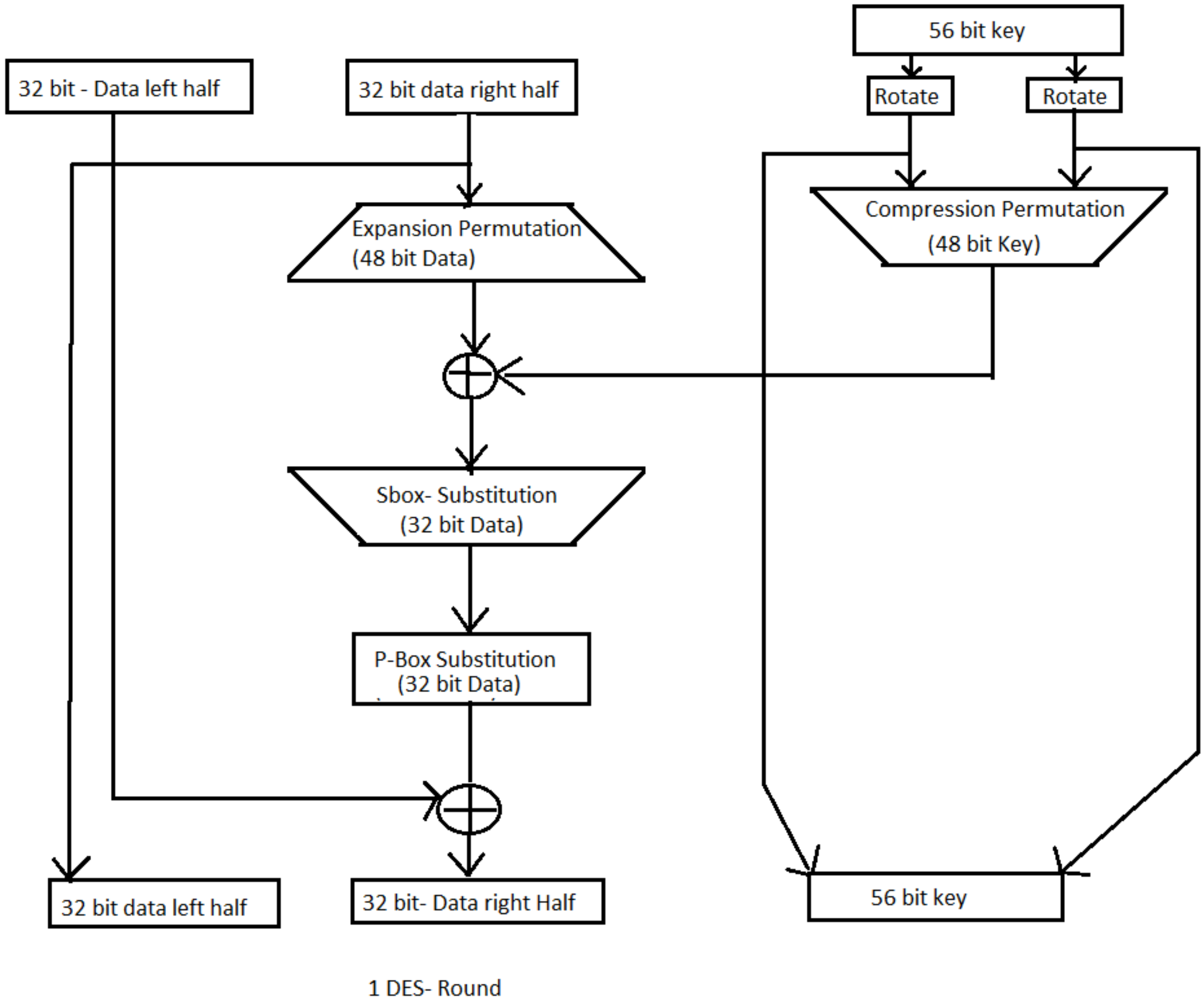


Figure 37. 1 Round of DES

5.5.5. Final Permutation :

The two halves at the output of the 16th round are swapped and passed through inverse permutation, using the below array.

```
int final_perm[64] = { 40, 8, 48, 16, 56, 24, 64, 32,
            39, 7, 47, 15, 55, 23, 63, 31,
            38, 6, 46, 14, 54, 22, 62, 30,
            37, 5, 45, 13, 53, 21, 61, 29,
            36, 4, 44, 12, 52, 20, 60, 28,
            35, 3, 43, 11, 51, 19, 59, 27,
            34, 2, 42, 10, 50, 18, 58, 26,
            33, 1, 41, 9, 49, 17, 57, 25 };
```

5.6. DES Decryption **:**

The Des decryption uses the same structure as des encryption. The difference being the keys used will be in reverse order, as well as the initial permutation and inverse permutation will replace their places.

5.7. 3-DES :

In 3-DES, the data to be encrypted is passed through 1-DES thrice with different 64-bit keys for each iteration. For decryption, similar keys are used in reverse order.

5.8. Software implementation for DES algorithm :

- Software used:- Windows subsystem for Linux
- The language used:- C
- The compiler used:- GCC
- Debugger used:- GDB
- The images below show the debug logs for the DES algorithm implemented in the software.

```
root@DESKTOP-MAQ414N:/mnt/c/Mtech/Sem_4/Software/DES# ./out_c
Key before parity drop is 133457799bbcdff1
Key after parity drop is f0ccaaf556678f


Encryption start

PlainText before intial Permutation is 123456789abcdef
PlainText after intial Permutation is cc00ccfff0aaf0aa
left data before expansion is cc00ccff
right data before expansion is f0aaf0aa
expansion_permutaion_op is 7a15557a1555
32 bit value computed from sbox is 5c82b597
32 bit value computed from pbox pboxSubsitution is 234aa9bb
result after exor of pbox and left data half is ef4a6544
data at the end of round 0 is left_data =f0aaf0aa right_data =ef4a6544 combined data is =f0aaf0aaef4a6544


left data before expansion is f0aaf0aa
right data before expansion is ef4a6544
expansion_permutaion_op is 75ea5430aa09
32 bit value computed from sbox is f8d03aae
32 bit value computed from pbox pboxSubsitution is 3cab87a3
result after exor of pbox and left data half is cc017709
data at the end of round 1 is left_data =ef4a6544 right_data =cc017709 combined data is =ef4a6544cc017709


left data before expansion is ef4a6544
right data before expansion is cc017709
expansion_permutaion_op is e58002bae853
32 bit value computed from sbox is 2710e16f
32 bit value computed from pbox pboxSubsitution is 4d166eb0
result after exor of pbox and left data half is a25c0bf4
data at the end of round 2 is left_data =cc017709 right_data =a25c0bf4 combined data is =cc017709a25c0bf4


left data before expansion is cc017709
right data before expansion is a25c0bf4
expansion_permutaion_op is 5042f8057fa9
32 bit value computed from sbox is 21ed9f3a
32 bit value computed from pbox pboxSubsitution is bb23774c
result after exor of pbox and left data half is 77220045
data at the end of round 3 is left_data =a25c0bf4 right_data =77220045 combined data is =a25c0bf477220045


left data before expansion is a25c0bf4
right data before expansion is 77220045
expansion_permutaion_op is bae90400020a
32 bit value computed from sbox is 50c831eb
32 bit value computed from pbox pboxSubsitution is 2813adc3
result after exor of pbox and left data half is 8a4fa637
data at the end of round 4 is left_data =77220045 right_data =8a4fa637 combined data is =772200458a4fa637
```

Figure 38. DES software implementation Encryption logs -part1

```
left data before expansion is b7d5d7b2
right data before expansion is c5783c78
expansion_permutaion_op is 60abf01f83f1
32 bit value computed from sbox is 7b8b2635
32 bit value computed from pbox pboxSubsitution is c268cfea
result after exor of pbox and left data half is 75bd1858
data at the end of round 11 is left_data =c5783c78 right_data =75bd1858 combined data is =c5783c7875bd1858


left data before expansion is c5783c78
right data before expansion is 75bd1858
expansion_permutaion_op is 3abdfa8f02f0
32 bit value computed from sbox is 9ad18b4f
32 bit value computed from pbox pboxSubsitution is ddbb2922
result after exor of pbox and left data half is 18c3155a
data at the end of round 12 is left_data =75bd1858 right_data =18c3155a combined data is =75bd185818c3155a


left data before expansion is 75bd1858
right data before expansion is 18c3155a
expansion_permutaion_op is f16068aaaf4
32 bit value computed from sbox is 64799af1
32 bit value computed from pbox pboxSubsitution is b7318e55
result after exor of pbox and left data half is c28c960d
data at the end of round 13 is left_data =18c3155a right_data =c28c960d combined data is =18c3155ac28c960d


left data before expansion is 18c3155a
right data before expansion is c28c960d
expansion_permutaion_op is e054594ac05b
32 bit value computed from sbox is b2e88d3c
32 bit value computed from pbox pboxSubsitution is 5b81276e
result after exor of pbox and left data half is 43423234
data at the end of round 14 is left_data =c28c960d right_data =43423234 combined data is =c28c960d43423234


left data before expansion is c28c960d
right data before expansion is 43423234
expansion_permutaion_op is 206a041a41a8
32 bit value computed from sbox is a7832429
32 bit value computed from pbox pboxSubsitution is c8c04f98
result after exor of pbox and left data half is a4cd995
data at the end of round 15 is left_data =43423234 right_data =a4cd995 combined data is =434232340a4cd995


right data is 43423234 left data is a4cd995
data for inveerse permutation is a4cd99543423234
final result after des computation is 85e813540f0ab405
```

Figure 39. Des software implementation -Encryption logs part-2

```
Decryption start

PlainText before intial Permutation is 85e813540f0ab405
PlainText after intial Permutation is a4cd99543423234
left data before expansion is a4cd995
right data before expansion is 43423234
expansion_permutaion_op is 206a041a41a8
32 bit value computed from sbox is a7832429
32 bit value computed from pbox pboxSubsitution is c8c04f98
result after exor of pbox and left data half is c28c960d
data at the end of round 15 is left_data =43423234 right_data =c28c960d combined data is =43423234c28c960d


left data before expansion is 43423234
right data before expansion is c28c960d
expansion_permutaion_op is e054594ac05b
32 bit value computed from sbox is b2e88d3c
32 bit value computed from pbox pboxSubsitution is 5b81276e
result after exor of pbox and left data half is 18c3155a
data at the end of round 14 is left_data =c28c960d right_data =18c3155a combined data is =c28c960d18c3155a


left data before expansion is c28c960d
right data before expansion is 18c3155a
expansion_permutaion_op is f16068aaaf4
32 bit value computed from sbox is 64799af1
32 bit value computed from pbox pboxSubsitution is b7318e55
result after exor of pbox and left data half is 75bd1858
data at the end of round 13 is left_data =18c3155a right_data =75bd1858 combined data is =18c3155a75bd1858


left data before expansion is 18c3155a
right data before expansion is 75bd1858
expansion_permutaion_op is 3abdfa8f02f0
32 bit value computed from sbox is 9ad18b4f
32 bit value computed from pbox pboxSubsitution is ddbb2922
result after exor of pbox and left data half is c5783c78
data at the end of round 12 is left_data =75bd1858 right_data =c5783c78 combined data is =75bd1858c5783c78


left data before expansion is 75bd1858
right data before expansion is c5783c78
expansion_permutaion_op is 60abf01f83f1
32 bit value computed from sbox is 7b8b2635
32 bit value computed from pbox pboxSubsitution is c268cfea
result after exor of pbox and left data half is b7d5d7b2
data at the end of round 11 is left_data =c5783c78 right_data =b7d5d7b2 combined data is =c5783c78b7d5d7b2
```
Figure 40. DES software implementation Decryption logs-part1

```
left data before expansion is 77220045
right data before expansion is a25c0bf4
expansion_permutaion_op is 5042f8057fa9
32 bit value computed from sbox is 21ed9f3a
32 bit value computed from pbox pboxSubsitution is bb23774c
result after exor of pbox and left data half is cc017709
data at the end of round 3 is left_data =a25c0bf4 right_data =cc017709 combined data is =a25c0bf4cc017709


left data before expansion is a25c0bf4
right data before expansion is cc017709
expansion_permutaion_op is e58002bae853
32 bit value computed from sbox is 2710e16f
32 bit value computed from pbox pboxSubsitution is 4d166eb0
result after exor of pbox and left data half is ef4a6544
data at the end of round 2 is left_data =cc017709 right_data =ef4a6544 combined data is =cc017709ef4a6544


left data before expansion is cc017709
right data before expansion is ef4a6544
expansion_permutaion_op is 75ea5430aa09
32 bit value computed from sbox is f8d03aae
32 bit value computed from pbox pboxSubsitution is 3cab87a3
result after exor of pbox and left data half is f0aaf0aa
data at the end of round 1 is left_data =ef4a6544 right_data =f0aaf0aa combined data is =ef4a6544f0aaf0aa


left data before expansion is ef4a6544
right data before expansion is f0aaf0aa
expansion_permutaion_op is 7a15557a1555
32 bit value computed from sbox is 5c82b597
32 bit value computed from pbox pboxSubsitution is 234aa9bb
result after exor of pbox and left data half is cc00ccff
data at the end of round 0 is left_data =f0aaf0aa right_data =cc00ccff combined data is =f0aaf0aacc00ccff


right data is f0aaf0aa left data is cc00ccff
data for inveerse permutation is cc00ccfff0aaf0aa
final result after des computation is 123456789abcdef
root@DESKTOP-MAQ414N:/mnt/c/Mtech/Sem_4/Software/DES#
```
Figure 41. DES software implementation Decryption logs- part-2

5.9 Hardware implementation for DES algorithm

- EDA tool used:- EDA playground.
- Simulation tool :- Icarus Verilog 0.9.7
- Synthesis tool:- Yosys 0.9.0
- The language used:- Verilog
- The image below shows the software implementation logs of the DES algorithm.

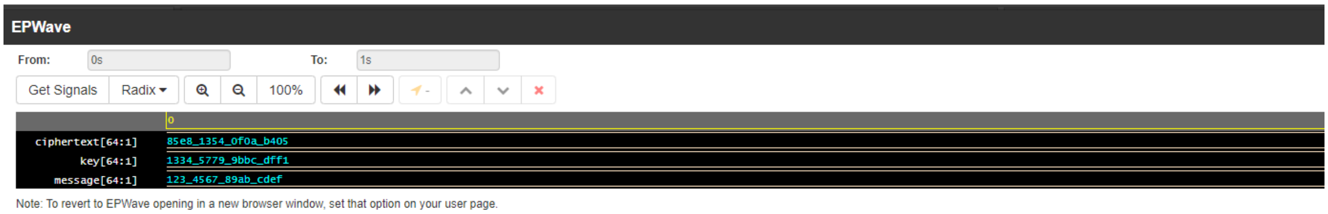


Figure 42. DES encryption in hardware using hardware simulation.

Note:- The current design of the DES block is not getting synthesized with the Yosys synthesis tool. Bug fixing is required for the same.

5.10. I2c Communication Protocol :

The I2c protocol is the most widely used protocol in wireless sensor networks/IoT networks to communicate with external sensors and devices. The I2C communication protocol works on 2 line communication. The diagram below shows the communication happening between two peripherals/devices using the I2C bus.

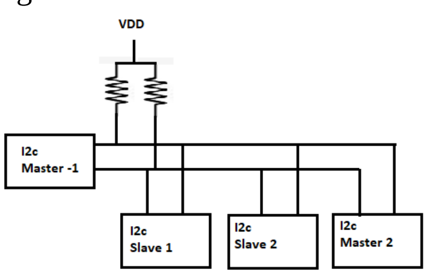


Figure 44. Typical I2c Communication

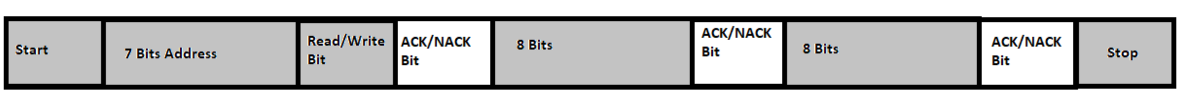


Figure 43. I2c communication datagram

After discussing the need for an I2c protocol for peripheral, it has been found that all the SOC peripheral would be working on the APB bus, so there is a need to build an I2c to APB bridge in hardware/software design. The effort needed to build the I2C-to APB bridge comes higher than the available time limit for the project. Hence the work is descoped for final desertion.

5.11. $\boldsymbol{E^2PROM}$ and RAM Memory :

The RISC-V CPU core uses internal ROM memory for storing instructions and data required for initial booting and complete operation of the SOC. The RAM memory is being used as temporary storage for CPU operations. The design and development of both memories do not fall within the scope of this Dissertation/Project.

6. Results and Conclusion

In order to identify performance differences, one could get using dedicated hardware for the crypto engine. I have used Si-five-based RISC-V microcontroller board for testing software implementation, whereas Artix-A7-35T FPGA for testing hardware implementations.

6.1. Software Implementation results :

6.1.1 Important Specifications of SiFive RISCV hi-five-revb development board :

Microcontroller - FE310-G002
Operating Voltage - 3.3 V and 1.8 V
I/O Voltage - 3.3 V
Default operating frequency – 16MHz
VDD supply current at 16MHz - 8mA

The table below has captured the approximate time it takes as well the approximate battery it will consume for running a particular algorithm to run on the FE310-G002 microcontroller.

Algorithm	Approximate time taken	Approximate battery consumption
BBS generating 10 random numbers and 10 random bits	6.5ms	14.448x10^{-6} mAH
RSA Encryption	8.2ms	18.2224 x10^{-6}mAH
RSA Decryption	8.3ms	18.4448 x10^{-6}mAH
RSA Key exchange	-	-
SHA512 for 731 bytes of data	85.5ms	188.8888 x10^{-6}mAH
SHA512 for 316 bytes of data	54.65ms	121.4448 x10^{-6}mAH
SHA512 for 8 bytes of data	44.04ms	97.8664 x10^{-6}mAH
DES Encryption	62.92ms	139.8224 x10^{-6}mAH
DES Decryption	63.65ms	139.8224 x10^{-6}mAH
3-DES Encryption	186.23ms	413.8448 x10^{-6}mAH
3-DES Decryption	188.98ms	419.9552 x10^{-6}mAH

6.2. Hardware Implementation results **:**

6.2.1 Important Specifications of Artix-A7 35T FPGA development board :

- FPGA - Artix-7 XC7A35T-L1CSG324
- On-chip analog-to-digital converter
- Programmable over JTAG and Quad-SPI Flash
- 16 MB Quad-SPI Flash
- Logic Cells – 33,280
- DSP Slices- 90
- Memory cells- 1800

Artix A7-35T is capable of simulating the entire RISCV core family of the processor. In order to measure performance parameters, the digital blocks of the algorithm are clocked at 16MHz, and it's being assumed approximate 8mA (ideally, it would be dependent upon the power mode of the RISC-V core.) of current would be consumed when the RISC-V core has dedicated hardware for the algorithm.

Algorithm	Approximate time taken	Approximate battery consumption
BBS generating 10 random number and 10 random bits	<0.625usec	$<0.00139\text{x}10^{-6}$ mAH
RSA Encryption	<0.0625usec	$<0.00139\text{x}10^{-6}$mAH
RSA Decryption	<0.0625usec	$<0.00139\text{x}10^{-6}$mAH
RSA Key exchange	-	-
SHA512 for 731 bytes of data	<5usec	$<0.0112\text{x}10^{-6}$mAH
SHA512 for 316 bytes of data	-	-
SHA512 for 8 bytes of data	-	-
DES Encryption	<2usec	$<0.00445\text{x}10^{-6}$mAH
DES Decryption	<2usec	$<0.00445\text{x}10^{-6}$mAH
3-DES Encryption	<6usec	$<0.01334\text{x}10^{-6}$mAH
3-DES Decryption	<6usec	$<0.01334\text{x}10^{-6}$mAH

Note:- The table assumes the data is available to the crypto block at 0-time intervals. When the crypto block is integrated with the CPU core, there would be a significant overhead of introducing the input data to the engine and taking out the data from the crypto engine.

Conclusion :

The hardware implementation of the same crypto algorithm can give a significant boost in terms of power and performance. The area of the chip is bound to increase. Making ASIC with a dedicated hardware engine as compared with a custom off-the-shelf CPU is bound to increase NRE cost multiple folds but can offer unmatched performance.

7. Future Work

- Testing of all the algorithm to be compliant with NIST test vectors.
- Design and development of RISC-V core.
- Bug-fixing and making all the RTL designs synthesizable on FPGA platform
- Code optimizations and bug-fixing in software.
- Integration of different algorithm on a common platform.
- Hardware-software partitioning.
- SHA512 to be made compatible with SHA-3.
- Design and Development of AES and its variants.

8. Project Task Tracker

Task	Sub-Task	Present Status	Test Results	Comment
Implement BBS for random byte and bits generation.	1. Implement BBS algorithm in software for random bit/Byte generation.	Completed	Pass	
	2. Implement BBS algorithm in hardware for random bit/Byte generation	Completed	Pass	
Implement RSA Algorithm in Software	1. Implement Key Generation and distribution mechanism in software	Completed	Pass	
	2. Implement RSA Enncryption algorithm in software	Completed	Pass	
	3. Implement RSA Decryption algorithm in software	Completed	Pass	
Implement RSA Algorithm in Hardware	1. Implement Key Generation and distribution mechanism in Hardware	In Progress	FAIL	Bug fixing is needed here
	2. Implement RSA Enncryption algorithm in Hardware	Completed	Pass	
	3. Implement RSA Decryption algorithm in Hardware	Completed	Pass	
Implement SHA-512 Algorithm in Software	1. Implement SHA-512 in software	Completed	Pass	
Implement SHA-512 Algorithm in Hardware	1. Implement SHA-512 conditional module	Completed	Pass	
	2. SHA-512 constant K generator module	Completed	Pass	
	3. SHA-512 majority finder module	Completed	Pass	
	4. SHA-512 Mixer_1 module	Completed	Pass	
	5. SHA-512 Mixer_2 module	Completed	Pass	
	6. SHA-512 Rotate_A module	Completed	Pass	
	7. SHA-512 Rotate_E module	Completed	Pass	
	8. SHA-512 W generator module	Completed	Conditioonal Pass	Design in not synthesizable- Bug fixing is needed here
	9. SHA-512 Round module	Completed	Conditioonal Pass	Design in not synthesizable- Bug fixing is needed here
	10. SHA-512 Top module	Completed	Conditioonal Pass	Design in not synthesizable- Bug fixing is needed here
Implement DES Algorithm in Software	Task not Started	Pending	-	
Implement DES Algorithm in Hardware	Task not Started	Pending	-	
Implement I2C Algorithm in software	Task not Started	Pending	-	
Implement I2C Algorithm in Hardware	Task not Started	Pending	-	
Measure Performance parameters of different Algorithm in Hardware-Software approach		Completed	-	

9. Plan of Work

Phases	Start Date-End Date	Work to be done
Dissertation Outline	7 Jan 2022–15 Jan 2022	Literature Review and prepare Dissertation Outline -Completed.
Design & Development	15 Jan 2022 – 15 Apr 2022	Design & Development Activity-In progress
Testing and bugfixes	16 Feb 2022 – 15 Apr 2022	Software Testing, User Evaluation & Conclusion -In progress in parallel
Dissertation Review	14 Mar 2022-25 Mar 2022	Submit Dissertation to Supervisor & Additional Examiner for review and feedback – In progress.
Submission	15 Apr 2022-25 Apr 2022	Final Review and submission of Dissertation-Pending

10.Literature References

[1] R. Zimmermann, "A 177 Mb/s VLSI Implementation of the International Data Encryption Algorithm", IEEE Journal of Solid-State Circuits. Vol. 29, No. 3, 1994, pp 303-307.

[2] Bonnenbergt, "VLSI Implementation of a New Block Cipher," IEEE Journal on Information Theory 1991, pp 510-513.

[3] K. Wong, "A Single-Chip FPGA Implementation of the Data Encryption Standard (DES) Algorithm" Global Telecommunications Conference, 1998. GLOBECOM 98, IEEE, Vol. 2, pp. 827-832

[4] Teo Pock Chueng, "Implementation of Pipelined Data Encryption Standard (DES) Using Altera CPLD," TENCON 2000 Proceedings, Vol. 3, IEEE 2000, pp 17-21

[5] FPGA-based implementation of the SHA-256 hash algorithm – Manel Kammoun, Manel Elleuchi, Mohamed Abid Mohammed S. BenSaleh

[6] Gaurav Sharma and Ajay Kakkar, "Cryptography Algorithms and Approaches used for Data Security," International Journal of Scientific & Engineering Research, 2012, Vol. 3, No. 6, pp. 1-6.

[7] Gaurav Sharma and Ajay Kakkar, "Cryptography Algorithms and Approaches used for DES algorithms," International Conference on Advancements in Computing and Communication, Vol. 2, 2012, pp. 427-432

[8] cryptography and network security principles and practice 7th edition

[9] Implementation of RSA Cryptosystem Using Verilog - Chiranth E, Chakravarthy H.V.A, Nagamohanareddy P, Umesh T.H, Chethan Kumar M. International Journal of Scientific & Engineering Research Volume 2, Issue 5, May-2011

[10] https://www.sifive.com/blog/custom-chips-for-under-100k

[11] https://www.skywatertechnology.com/blog/how-to-design-with-a-free-skywater-pdk/

[12] https://osfpga.org/

[13] https://www.sciencedirect.com/topics/engineering/hardware-software-codesign

11. Abbreviations

IoT	Internet of things
SOC	System on Chip
ASIA	Application-specific integrated circuit
DSP	Digital Signal Processing
JTAG	Joint Test Action Group
USB	Universal Serial Bus
DES	Data Encryption Standard
3-DES	Triple Data Encryption Standard
SHA	Secure Hash Algorithm
RSA	Rivest–Shamir–Adleman Cryptography
FPGA	Field Programmable Gate Array
USART	Universal Asynchronous Receive Transmit
I2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface
COTS	Custom off the shelf CPU
ARM	Advanced RISC Machine
EDA	Electronic Design Automation
BBS	Blum Blum Shub Algorithm
NIST	National Institute of Standard and Technology
NRE	Non-Recurring Engineering Cost
BBS	Blum Blum Shub

12. Particulars of the Supervisor and Examiner

	Supervisor	Additional Examiner
Name	Sunil Pillai	Kaustubh Sarwate
Qualification	MS, Embedded Systems Design	MS, computer science
Designation	Engineer, Senior Staff	Engineer, Principal/Manager,
Employing Organization and Location	Qualcomm India Pvt. Ltd. Bangalore.	Qualcomm India Pvt. Ltd. Bangalore.
Phone No. (with STD Code)	(+91) 9986660573	(+91) 9886653329
Email Address	psunil@qti.qualcomm.com	ksarwate@qti.qualcomm.com

13.Remarks of the Supervisor

This project focuses on the hardware realization of a few fundamental cryptographic implementations related to data confidentiality, source authentication, and message integrity services in IoT devices. The choice of block cipher DES and its improvised version, Triple DES, as the encryption algorithm is expected to help in the analysis and evaluation of some fundamental cryptographic operations. The chosen data integrity and authentication algorithms are expected to further reinforce the understanding of message schedulers and iterative block operations, involving complex computations like 64-bit modulo operations, etc.,

Further, exposing the crypto services to programmable controllers on the SoC is critical and is expected to align with the standard definition of the cryptographic operations in the scope of this project. However, the references require a more detailed investigation along with citations of any challenges or technical limitations to implement the same on FPGA, if any.

Information about the Supervisor:

I'm a security architect and engineer in Qualcomm with experience in working with diversified product ranges from low power to high-performance devices, focusing mainly on security of the overall System-On-Chip. I've worked on the usage of various cryptographic primitives required in the design and implementation of Root-of-Trust for wide range of Qualcomm products.

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
WORK-INTEGRATED LEARNING PROGRAMMES DIVISION
Second Semester 2021-2022

ES ZG628T: Dissertation EC-1: Dissertation Outline Evaluation Sheet

Upload Softcopy of Outline Report and Evaluation Sheet on BITS LMS Taxila 15 Jan, 2022

ID No. : 2020ht01012
NAME OF THE STUDENT : Sarang Pramod Choudalwar
EMAIL ADDRESS : 2020ht01012@wilp.bits-pilani.ac.in
STUDENT'S EMPLOYING ORGANIZATION & LOCATION : Qualcomm India Pvt. Ltd. Bangalore
SUPERVISOR'S NAME : Sunil Pillai
SUPERVISOR'S EMPLOYING ORGANIZATION & LOCATION : Qualcomm India Pvt. Ltd. Bangalore
PROPOSED DISSERTATION TITLE: Crypto Engine for Secure communication in IoT chipsets.

Student must enclose the Dissertation Outline document in proper format (as given on the next page) containing details of the proposed topic of Dissertation Work, background, scope of work, objectives, plan of work, literature references and particulars of the Supervisor as well as one additional examiner in terms of Name, Designation, Qualification and contact information. The student's Mentor should either be the Supervisor or the Additional Examiner. The Supervisor and Additional Examiner should verify and sign at the end of the Dissertation Outline document.

DISSERTATION OUTLINE EVALUATION
(Please put a tick (✔) mark in the appropriate box)

EC No.	Component	Excellent	Good	Fair	Poor
1.	Dissertation Outline		✔		

	Supervisor	Additional Examiner
Name	Sunil Pillai	Kaustubh Sarwate
Qualification	MS, Embedded Systems Design	MS, computer science
Designation	Engineer, Senior Staff	Engineer, Principal/Manager
Employing Organization and Location	Qualcomm India Pvt. Ltd. Bangalore.	Qualcomm India Pvt. Ltd. Bangalore
Phone No. (With STD Code)	(+91) 9986660573	(+91) 9886653329
Email Address	psunil@qti.qualcomm.com	ksarwate@qti.qualcomm.com
Signature		
Date	15th-Jan -2022	15th-Jan-2022

ES ZG628T: Dissertation Mid-Semester Progress Evaluation Sheet

Upload Softcopy of Mid-Semester Progress Report and Mid-Semester Evaluation Sheet by
15 Mar 2022

ID No. : 2020ht01012
NAME OF THE STUDENT : Sarang Pramod Choudalwar
EMAIL ADDRESS : 2020ht01012@wilp.bits-pilani.ac.in
SUPERVISOR'S NAME : Sunil Pillai
DISSERTATION TITLE: Crypto Engine for Secure communication in IoT chipsets.

EVALUATION

DISSERTATION PROGRESS EVALUATION *(Please put a tick (✔) mark in the appropriate box)*
(Note:-The evaluation from the supervisor and external examiner is still in progress.)

EC No.	Component	Excellent	Good	Fair	Poor
1.	Dissertation Outline		✔		
2.	Work Progress & Achievements		✔		
3.	Initiative and Originality		✔		
4.	Documentation & Expression		✔		
5.	Research & Innovation			✔	
6.	Relevance to the Work Environment		✔		

	Supervisor	Additional Examiner
Name	Sunil Pillai	Kaustubh Sarwate
Qualification	MS, Embedded Systems Design	MS, computer science
Designation	Engineer, Senior Staff	Engineer, Principal/Manager
Employing Organ and Location	Qualcomm India Pvt. Ltd. Bangalore.	Qualcomm India Pvt. Ltd. Bangalore
Phone No. (with STD Code)	(+91) 9986660573	(+91) 9886653329
Email Address	psunil@qti.qualcomm.com	ksarwate@qti.qualcomm.com
Signature		
Date	15^{th}-Mar -2022	15^{th}-Mar-2022

14. Checklist of Items for the Final Dissertation / Project / Project Work Report

1.	Is the final report neatly formatted with all the elements required for a technical Report?	Yes
2.	Is the Cover page in proper format as given in Annexure-A?	Yes
3.	Is the Title page (Inner cover page) in proper format?	Yes
4.	(a) Is the Certificate from the Supervisor in proper format? (b) Has it been signed by the Supervisor?	Yes Yes
5.	Is the Abstract included in the report properly written within one page? Have the technical keywords been specified properly?	Yes
6.	Is the title of your report appropriate? The title should be adequately descriptive, and precise and must reflect the scope of the actual work done. Uncommon abbreviations / Acronyms should not be used in the title	Yes
7.	Have you included the List of abbreviations / Acronyms?	Yes
8.	Does the report contain a summary of the literature survey?	Yes
9.	Does the Table of Contents include page numbers? (i). Are the Pages numbered properly? (Ch. 1 should start on Page # 1) (ii). Are the Figures numbered properly? (Figure Numbers and Figure Titles should be at the bottom of the figures) (iii). Are the Tables numbered properly? (Table Numbers and Table Titles should be at the top of the tables) (iv). Are the Captions for the Figures and Tables proper? (v). Are the Appendices numbered properly? Are their titles appropriate	 Yes Yes Yes Yes Yes
10.	Is the conclusion of the report based on a discussion of the work?	-
11.	Are References or Bibliography given at the end of the report? Have the References been cited properly inside the text of the report? Are all the references cited in the body of the report	Yes Yes Yes
12.	Is the report format and content according to the guidelines? The report should not be a mere printout of a Power Point Presentation, or a user manual. Source code of software need not be included in the report.	Yes

Declaration by Student:
I certify that I have properly verified all the items in this checklist and ensure that the report is in proper format as specified in the course handout.

Place: Bangalore

Date: 25th April 2022

Signature of the Student

Name: Sarang Pramod Choudalwar

ID No.: 2020ht01012