International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Design of Compact and high Throughput Blowfish Encryption Method for IoT Networks using VHDL

Shreya Soni[a], Divyanshu Rao[b], Ravimohan[c]*

[a]*M. Tech. Schohlar,Department of ECE, SRIT,Jabalpur, India*
[b,c]*Assistant Professor,Department of ECE, SRIT,Jabalpur, India*

A B S T R A C T

Blowfish Encryption is best suitable for the dynamic networks like Internet of Things (IoT) because Blowfish Encryption is reconfigurable, hence this work is a design of implement the Area and Speed efficient Symmetric Key based modified Blowfish encryption algorithm. The design Register Transfer Level (RTL) entries done using VHDL and design is implemented on 7 series Vertex family Field programmable Gate Array (FPGA). This work aims to provide a simple, robust solution for the IoT networks where lightweight encryption is highly required for network security.A hardware implementation of Blowfish would be a powerful tool for any mobile device in IoT network or any technology requiring strong encryption. This final design uses the core shows library for worst-case scenario analysis and reaches an incredible encryption speed of 590 MBits/sec and a decryption speed of 559 MBits/sec. The area is 4996 standard cells and the power is a mere 63 mW. These results are very competitive and beat out the competition as far as speed is concerns. The overall design is an incredibly fast, efficient Blowfish implementation suitable for a plethora of applications. The speed can be drastically increased further at the expense of space and power by dataflow modelling uses while RTL entries.

Keywords:ASIC - Application Specific Integrated Circuit ,FPGA- Field programmable Gate Array, RC4- Rivest Cipher4, VHDL, Internet of Things (IoT)

1. Introduction

Encryption is a security control used primarily to provide confidentiality protection for data. It is a mathematical transformation to scramble data requiring protection (plaintext) into a form not easily understood by unauthorized people or machines (ciphertext). After being transformed into ciphertext, the plaintext appears random and does not reveal anything about the content of the original data. Once encrypted, no person (or machine) can discern anything about the content of the original data by reading the encrypted form of the data.

Encryption is a reversible transformation. It is useful only when encrypted data (ciphertext) can be reversed back to its original, unencrypted form (plaintext). If not reversible, the encrypted data are considered unreadable and unusable. This reversal process is referred to as decryption. An encryption process has a corresponding decryption process, which is used to reverse the encrypted data (ciphertext) back to its original content (plaintext).

Symmetric key Encryption: In this type to communication encryption and decryption key are same and sender and receiver have key before they start communication. The key that is used to encryption and decryption that may be identical or there may be easy transformation to go between two keys

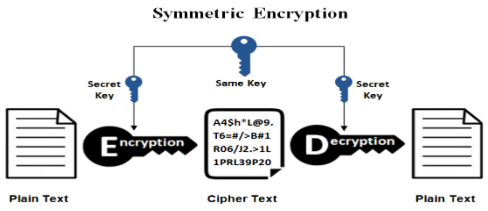


Figure 1 Symmetric Encryption

* *Corresponding author.* Tel.: +917389973999
E-mail address: shreyasoni889714@gmail.com

Table 1: Block and Stream Cipher Speed Comparison

Algorithm	Type	Clocks/ Round	No. of Round	Clock/Byte of output
RC4[2]	Stream cipher	N.A	N.A.	7
SEAL[7]	Stream cipher	NA	N.A.	4
Blowfish[1][2][3]	Block Cipher	9	16	18
RCS[5]	Block Cipher	12	16	23
DES[1,4]	Block Cipher	18	16	45
IDEA[2]	Block Cipher	50	8	50
Triples-DES[1]	Block Cipher	18	48	108

Table 2: Speed comparisons of block ciphers

Speed Comparison of Block Ciphers on A Pentium				
Algorithm	Clock cycles per round	Numbers of Round	No. of clock cycles per byte encrypted	Notes
Blowfish[1,2,3,4]	9	16	18	Free, unpatented
Khufu/Khafre[5]	5	32	20	Patented by Xerox
RC5[6]	12	16	23	Patented by RSA data security
DES[1,7]	18	16	45	56-bit key
IDEA[1,8]	50	8	50	Patented by Ascom-Sytec
Triple-DES[1,8,9]	18	48	108	

2. Proposed Work

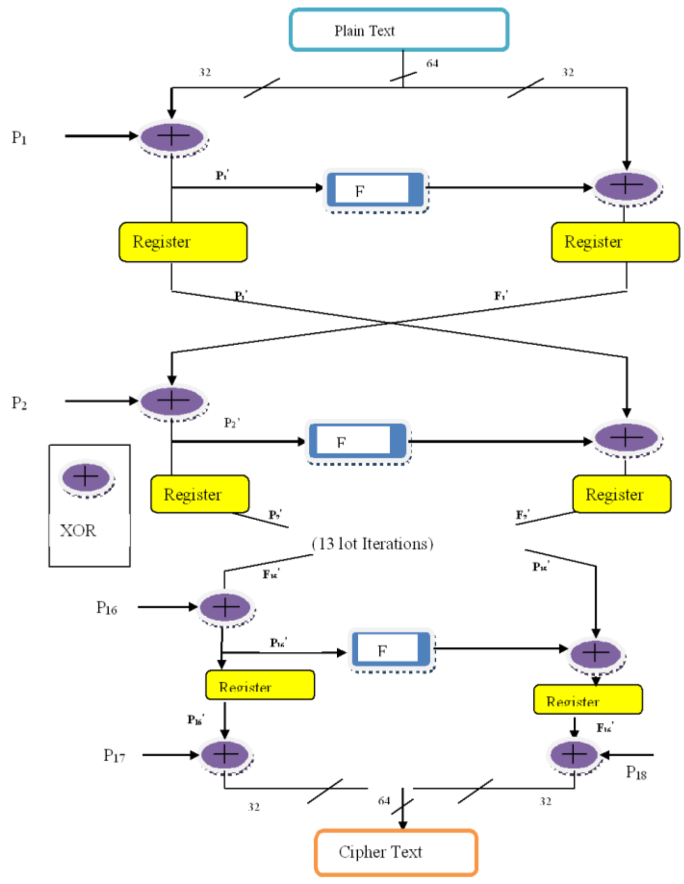


Figure 2 Proposed Blowfish Algorithm

Blowfish Encryption is best suitable for the dynamic networks like Internet of Things (IoT) because Blowfish Encryption is reconfigurable, hence this work is a design of implement the Area and Speed efficient Symmetric Key based modified Blowfish encryption algorithm. Figure 2 shows the proposed blowfish encryption process.

The design Register Transfer Level (RTL) entries done using VHDL and design is implemented on 7 series Vertex family Field programmable Gate Array (FPGA). This work aims to provide a simple, robust solution for the IoT networks where lightweight encryption is highly required for network security. Proposed design is blowfish algorithm with a new pipelined arrangement here proposed work has inserted register between all available rounds of blowfish procedure here pipeline is necessary because for increasing throughput of overall work. Here proposed work also uses tree addition structure for addition of numbers in S-box of blowfish.

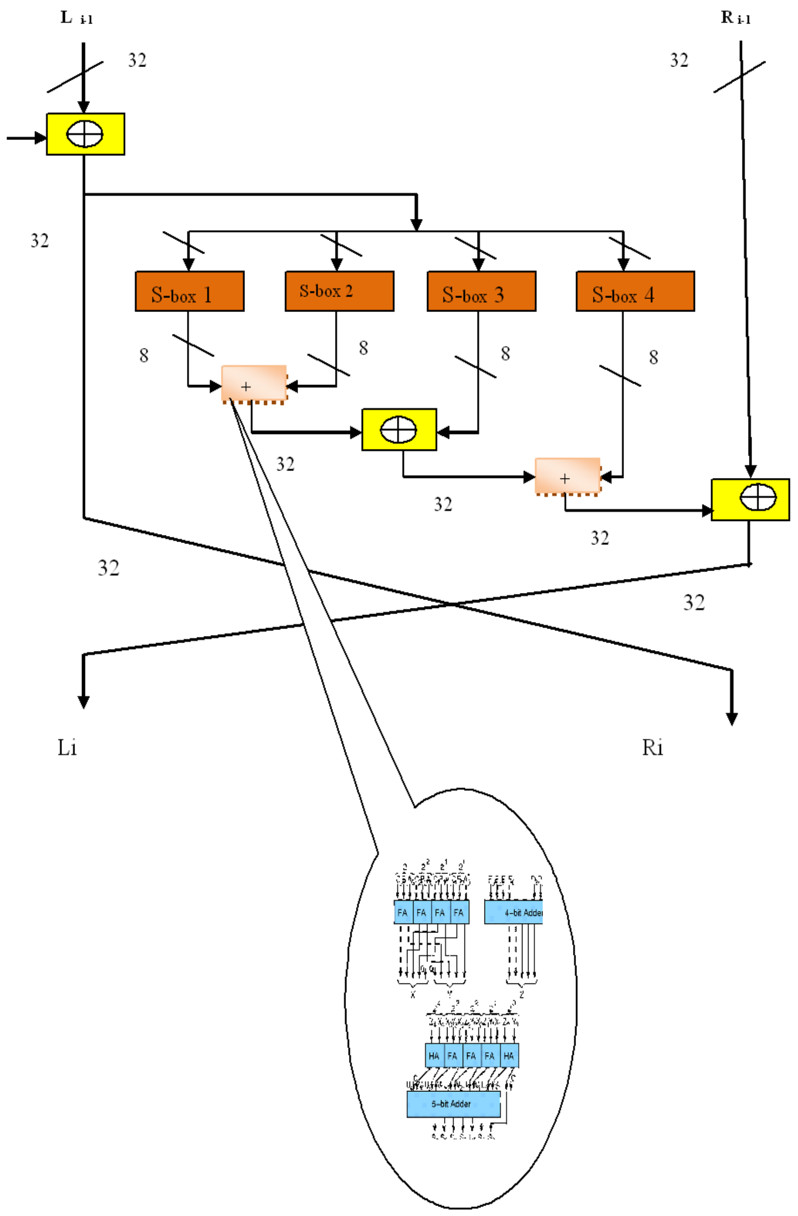


Figure 3 new design of Feistel network

Figure 3 above shows the proposed Wallace adder integration in blowfish encryption module in order reduce the chip area of IoT network nodes processors. As been discussed that every blowfish algorithm has requirement of Feistel work and every Feistel network necessary S-box and here addition is been done by new tree structure which will reduce area of complete design, proposed design also using combinational logic to design S-Box instead of using Memory element, memory is slower than any combinational circuit or IC and does not match up speed of processor. Hence use of combinational logic will enhance speed of design with significant amount.

Proposed design measure SNR and MSE between original and cipher data and also measure avalanche [2] and pence to achieve less area proposed work is using new addition structure and to achieve high speed procedure is been implemented with help of pipelining concept. Figure 4 below shows the proposed blowfish implementation flow of FPGA.

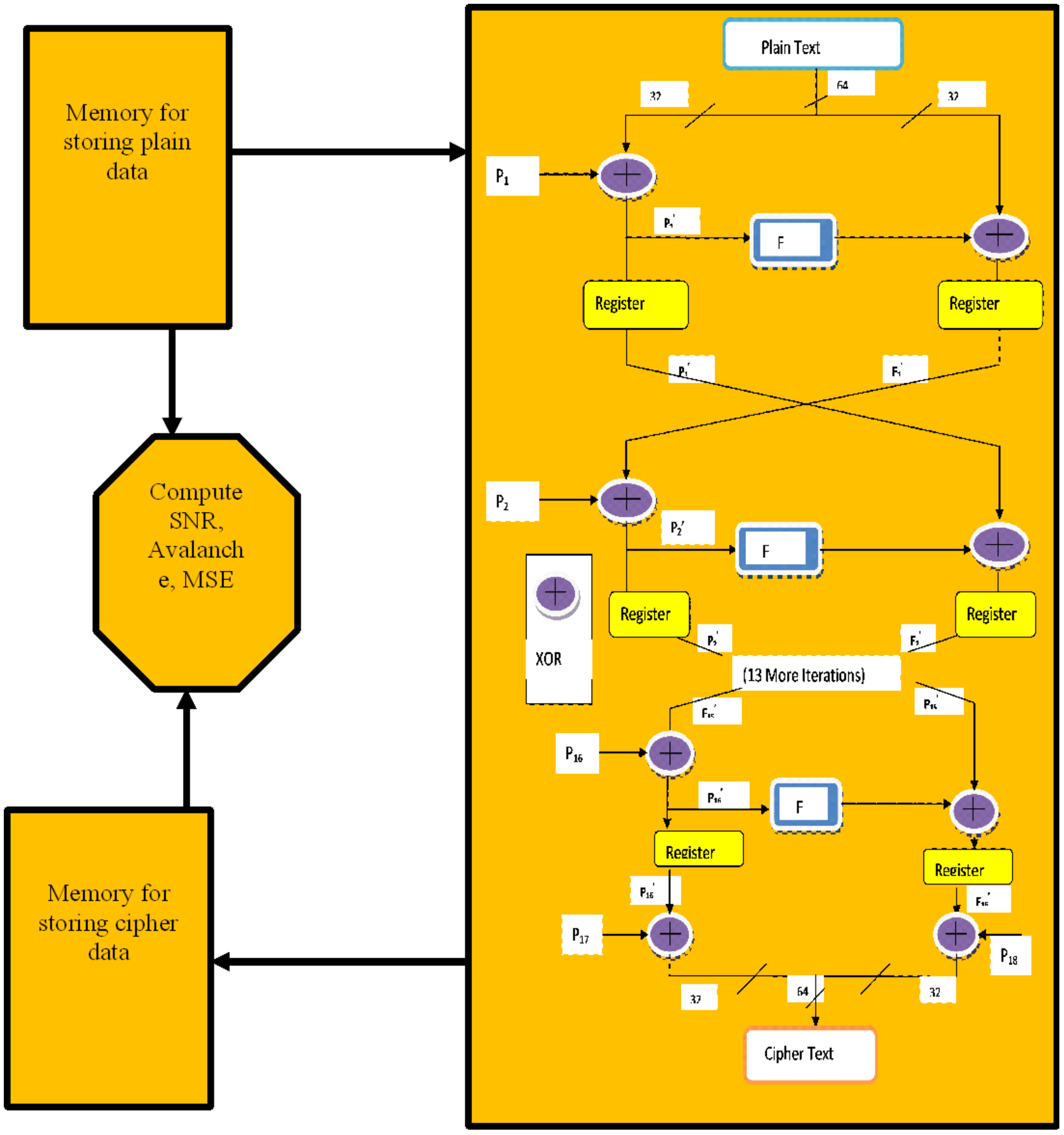


Figure 4proposed work SNR and MSE computation

3. Implementation and Results

For implementing blowfish algorithm select P arrays randomly as user wants. As par security concern all P arrays taken in a very random. Similarly, these keys Explored with input 32-bit data which will acts as an input to Feistel function Box. Then blowfish algorithm [8]is implemented using VHDL language and is verified functionally using Xilinx ISE 12.1i. simulated results are presented in this chapter. Xilinx-ISE simulation and Modalism SE PLUS 6.2C results shows that encryption scheme is a complex encryption scheme and is highly reliable.

The sub-keys are calculated using *Blowfish* algorithm:

Step1. Initialize first P-array and then four S-BOX, in order, with a fixed string. This string consists of hexadecimal digits of pi (less initial 3):

P1=0x03C3C00F,

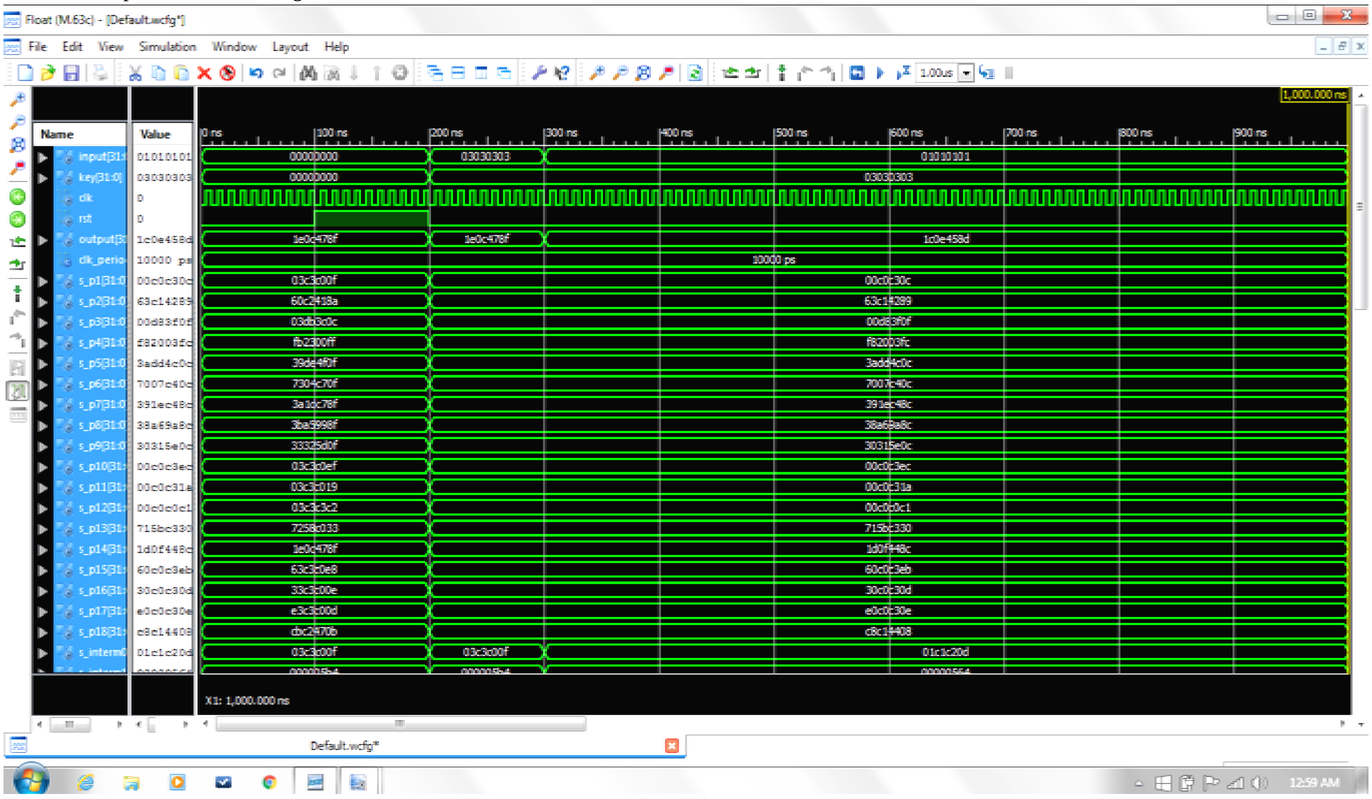
P2 = 0x60C2418A,

P3 = 0x03DB3C0C,

P4 = 0xFB2300FF, etc.

Total 18 sub-keys are taken depends on user.

Step 2. XOR P1 with first 32 bits of data, XOR P2 with second 32-bits of data, and so on for all bits of key (possibly up to P18). Repeatedly cycle through key bits until entire P-array has been XORed with data bits. (For every short key, there is at least one equivalent longer key; for example, if A is a 64-bit key, then AA, AAA, etc., are equivalent keys.)
Step 3. Encrypt all-zero string with *Blowfish* algorithm, using sub-keys described in steps (1) and (2).
Step 4. Replace P1 and P2 with output of step (3).
Step 5. Encrypt output of step (3) using blowfish algorithm with modified
Sub-keys.
Step 6. Replace P3 and P4 with output of step (5).
Step 7. Continue process, replacing all entries of P array, and then all four S-BOX in order, with output of continuously changing Blowfish algorithm. In total, 521 iterations are required to generate all required sub-keys. Applications may store sub-keys rather than execute this derivation method multiple times.
Each S-BOX having 8-bit input and these 8 bits is converted into 32-bit output.
There is total 4 S-BOX are used.
These total 4 S-BOX forms a Feistel function which has 32-bit input and these 32-bit data is converted into four 8 bit and each 8-bit given as an input to every S-BOX.
Output of S-BOX are EX-OR and added which is a 32-bit output of Feistel function.
Total 16 F functions are generated.
For simulation of these S-BOX we have to use few equations and every S-BOX have various equation. For demonstration example input of one S-BOX is taken 8 bits as 01010101 (85H) and output generated by this S-BOX is 00000000000000000000001101100000 (352H) (32-bit number). Figure 5.2 shows RTL view of S-BOX.
Generation of Feistel Function: In similar manner we may generate output of S-BOX and after E-XOR, addition of all output of all S-BOX output of Feistel function is generated. A Feistel Network is a general procedure of transforming any function (usually called an F function) into a permutation. It was invented by Horst Feistel and has been used in many blocks cipher designs. working of a Feistel Network is given below:
Split each block into halves
Right half becomes new left half
New right half is final result when left half is XOR'd with result of applying Fto right half and key. (Note that previous rounds may be derived even if function Fis not invertible.)
For generation of output of Feistel function following method adopted.
The input to F function is 32 bits applied which is divided in four 8 bits.
Every 8 bit is input of S-BOX. (Total 4 S-BOX are there). These S-BOX converts 8 bits into 32 bits data.
The output of S-BOX1 and S-BOX2 are added and mod 2^{32} result is EXOR with output of 3rd S-BOX.
After EXOR output result is then again added (mod2^{32}) with output of 4th S-BOX.
Then total output of F function is generated.


Float (M.63c) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
1.00us
1,000.000 ns
Name
Value
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
input[31:0] 01010101 00000000 03030303 01010101
key[31:0] 03030303 00000000 03030303
clk 0
rst 0
output[31:0] 1c0e458d 1e0c478f 1e0c478f 1c0e458d
clk_period 10000 ps 10000 ps
s_p1[31:0] 00c0c30c 03c3c00f 00c0c30c
s_p2[31:0] 63c14289 60c2418a 63c14289
s_p3[31:0] 00d83f0f 03db3c0c 00d83f0f
s_p4[31:0] f82003fc fb2300ff f82003fc
s_p5[31:0] 3add4c0c 39de4f0f 3add4c0c
s_p6[31:0] 7007c40c 7304c70f 7007c40c
s_p7[31:0] 391ec48c 3a1dc78f 391ec48c
s_p8[31:0] 38a69a8c 3ba5998f 38a69a8c
s_p9[31:0] 30315e0c 33325d0f 30315e0c
s_p10[31:0] 00c0c3ec 03c3c0ef 00c0c3ec
s_p11[31:0] 00c0c31a 03c3c019 00c0c31a
s_p12[31:0] 00c0c0c1 03c3c3c2 00c0c0c1
s_p13[31:0] 715bc330 7258c033 715bc330
s_p14[31:0] 1d0f448c 1e0c478f 1d0f448c
s_p15[31:0] 60c0c3eb 63c3c0e8 60c0c3eb
s_p16[31:0] 30c0c30d 33c3c00e 30c0c30d
s_p17[31:0] e0c0c30e e3c3c00d e0c0c30e
s_p18[31:0] c8c14408 cbc2470b c8c14408
s_interm 01c1c20d 03c3c00f 03c3c00f 01c1c20d
X1: 1,000.000 ns
Default.wcfg*
12:59 AM
Figure.5 Simulation Results of Blowfish Algorithm. (Encryption)

Data encryption occurs via a 16-round Feistel network. Each round consists of a key dependent permutation, and a key- and data-dependent substitution. All operations are XORs and additions on 32-bit words. only additional operations are four indexed array data lookups per round. The encrypted output is shown in Figure 5. One input data as “00000001000000010000000100000001”(01010101) is taken and encrypted output found “00011100000011100100010110001101”(1C0E458D). for key value is obtained“00000011000000110000001100000011”(03030303). In this example 32 bit input data is given and after simulation output is generated as shown.The total time taken for these simulation of 32 bit data is 4.398 ns.Similarly decrypted data is generated. Decryption works in reverse order of encryption beginning with ciphertext as input. sub-keys are used in reverse order.

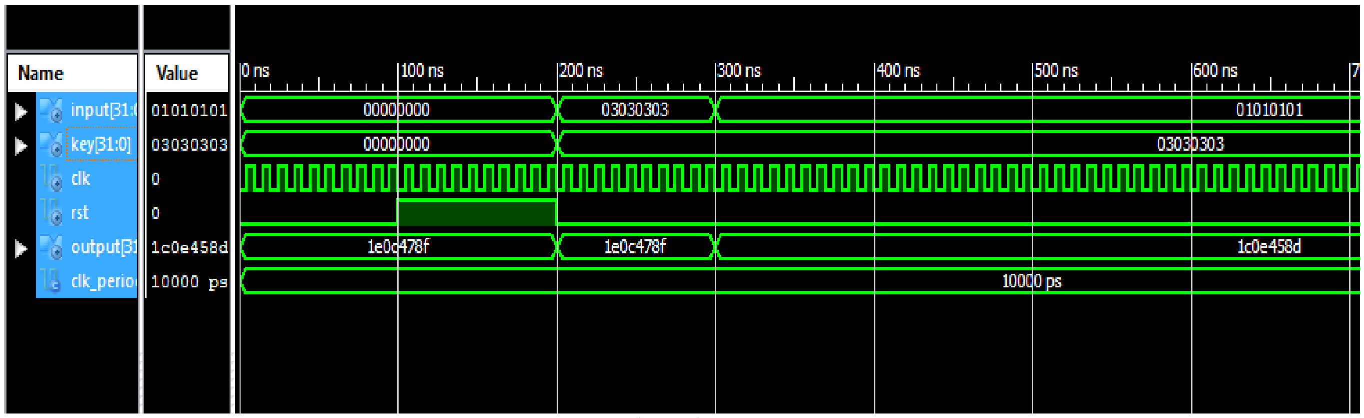
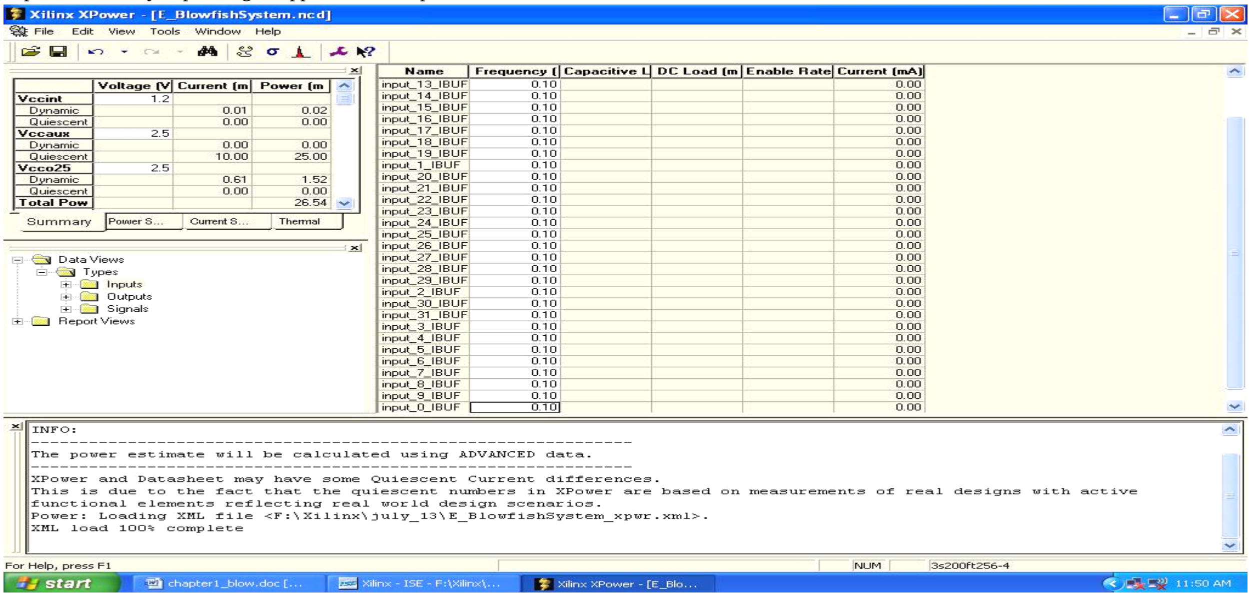


Figure 6 Simulation Results of Blowfish Algorithm. (Decryption)

Decryption (Figure 6) works in reverse order of encryption, beginning with cipher text as input and output is our plaintext. sub-keys are used in reverse order. 32-bit data is “00000001000000010000000100000001” (01010101) and 32-bit key is “00000011000000110000001100000011” (03030303) and final output after decryption obtain is “00011100000011100100010110001101” (1C0E458D).

Power Consumption:Figure 7 shows the power consumption reports. While executing data and total power consumed after encryption and decryption is shown in figure. both methods are carried out simultaneously hence, total consumption is 26.54Mw at 0.01Mhz frequency of input data. power consumption is linearly depending of application of input data.



	Voltage (V	Current (m	Power (m
Vccint	1.2		
Dynamic		0.01	0.02
Quiescent		0.00	0.00
Vccaux	2.5		
Dynamic		0.00	0.00
Quiescent		10.00	25.00
Vcco25	2.5		
Dynamic		0.61	1.52
Quiescent		0.00	0.00
Total Pow			26.54

Name	Frequency (	Capacitive L	DC Load (m	Enable Rate	Current (mA)
input_13_IBUF	0.10				0.00
input_14_IBUF	0.10				0.00
input_15_IBUF	0.10				0.00
input_16_IBUF	0.10				0.00
input_17_IBUF	0.10				0.00
input_18_IBUF	0.10				0.00
input_19_IBUF	0.10				0.00
input_1_IBUF	0.10				0.00
input_20_IBUF	0.10				0.00
input_21_IBUF	0.10				0.00
input_22_IBUF	0.10				0.00
input_23_IBUF	0.10				0.00
input_24_IBUF	0.10				0.00
input_25_IBUF	0.10				0.00
input_26_IBUF	0.10				0.00
input_27_IBUF	0.10				0.00
input_28_IBUF	0.10				0.00
input_29_IBUF	0.10				0.00
input_2_IBUF	0.10				0.00
input_30_IBUF	0.10				0.00
input_31_IBUF	0.10				0.00
input_3_IBUF	0.10				0.00
input_4_IBUF	0.10				0.00
input_5_IBUF	0.10				0.00
input_6_IBUF	0.10				0.00
input_7_IBUF	0.10				0.00
input_8_IBUF	0.10				0.00
input_9_IBUF	0.10				0.00
input_0_IBUF	0.10				0.00

```
INFO:
------------------------------------------------------------
The power estimate will be calculated using ADVANCED data.
------------------------------------------------------------
XPower and Datasheet may have some Quiescent Current differences.
This is due to the fact that the quiescent numbers in XPower are based on measurements of real designs with active
functional elements reflecting real world design scenarios.
Power: Loading XML file <F:\Xilinx\july_13\E_BlowfishSystem_xpwr.xml>.
XML load 100% complete
```

Figure 7 Power consumption in encryption

4. Comparative Results

By observing synthesis report it is clear that after executing Blowfish system method total delay is 4.398ns which is very less as compared to other results. Which indicates speed of this Blowfish system. Table 3 shows sythesis results.

Table 3 Synthesis report of Blowfish system

1.	Number of Slices	570
2.	Number of 4 Input LUTs	1007
3.	Number of IOs	98
4.	Number of bonded IOBs	98

5.	Time	4.398 ns
6.	Maximum Frequency	227.37 MHz

Table 4 Comparative results

	Parameters	Proposed work	[1]
1.	Number of Slices	570	581
2.	Time	4.398 ns	6.32 ns

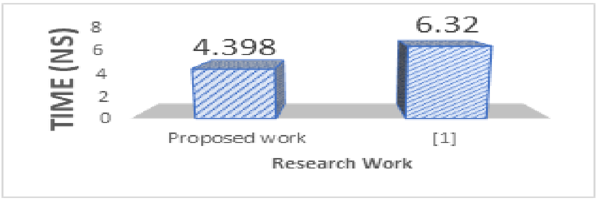


Figure 8 Comparative results: Time(ns)

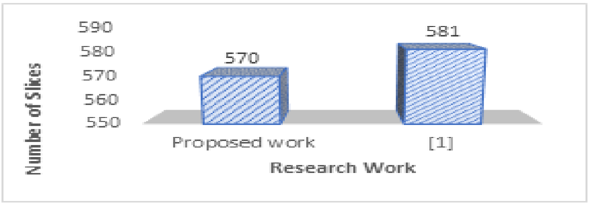


Figure 9 Comparative results: Slices

Form table 4 and figure 8 and figure 9 above it may be observed that total number of slices required for proposed work are less than base work and required propagation time is also less hence we may conclude that proposed design is best suitable for blowfish encryption in IoT networks.

5. Conclusion

Input signal selection procedure is presented and blowfish algorithm [8]is implemented using VHDL language and is verified functionally using Xilinx-ISE 12.2i. simulated results are presented in this chapter. Xilinx-ISE simulation results are been explained. Generation of keys explained using S-BOX and Feistel function are also generated and explained. encryption and decrypted outputs are generated and simulation results with RTL view schematic are generated and presented. Comparison from various architecture (algorithms) with speed, Clocks/Round, No. of Round and other parameters are presented from various literature. Blowfish Encryption scheme and Decryption is implemented using VHDL language on FPGA. This is a high secured algorithm. Software cryptanalysis of such a complex algorithm is highly impossible. Hardware implementation of such a high secured algorithm is highly necessary for high-speed applications. due to hardware implementation of algorithm, there is no need of any external platform to run algorithm, and also plain text is encrypted in negligible amount of time. Simulation results shows that this algorithm it may encrypt a plain text in less than 98ns of time.

References

[1] R. Anusha, S. N, V. S. Shetty and S. B. K, "Analysis and Comparison of Symmetric Key Cryptographic Algorithms on FPGA," 2022 4th International Conference on Smart Systems and Inventive Technology (ICSSIT), 2022, pp. 293-300, doi: 10.1109/ICSSIT53264.2022.9716416.

[2] P. Parvathy and A. S. RemyaAjai, "VLSI Implementation of Blowfish Algorithm for Secure Image Data Transmission," 2020 International Conference on Communication and Signal Processing (ICCSP), 2020, pp. 0770-0774, doi: 10.1109/ICCSP48568.2020.9182088.

[3] S. Divya, K. V. Prema and B. Muniyal, "Privacy Preservation Mechanism For The Data Used In Image Authentication," 2019 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER), 2019, pp. 1-6, doi: 10.1109/DISCOVER47552.2019.9007939.

[4] S. B. Nalawade and D. H. Gawali, "Design and implementation of blowfish algorithm using reconfigurable platform," 2017 International Conference on Recent Innovations in Signal processing and Embedded Systems (RISE), 2017, pp. 479-484, doi: 10.1109/RISE.2017.8378204.

[5] M. Goyal and A. Sharma, "Implementation and Analysis of various Encryption Techniques with Blowfish on various Data Files," 2021 International Conference on Technological Advancements and Innovations (ICTAI), 2021, pp. 541-546, doi: 10.1109/ICTAI53825.2021.9673454.

[6] Kurniawan Nur Prasetyo,YudhaPurwanto, Denny Darlis Department of Computer System, Telkom Univercity, Bandung, An implementation of data encryption for Internet of Things using blowfish algorithm[8]on FPGA, 2nd International Conference onInformation and Communication Technology (ICoICT), 2014, 0464-9719/$26.00 © IEEE 2014 DOI:10.1109/ICoICT.2014.6914043

[7] Dr. J. Abdul Jaleel, Jisha Mary Thomas, Guarding Images using a Symmetrickey[6] Cryptographic Technique: Blowfish Algorithm, ISSN: 2277-3754 ISO 9001:2008 Certified International Journal of Engineering and Innovative Technology (IJEIT) Volume 3, problem 2, August 2013

[8] Saikumar Manku1 and K. Vasanth, BLOWFISH ENCRYPTION ALGORITHM FOR INFORMATION SECURITY, VOL. 10, NO. 10, JUNE 2015 ISSN 1819-6608, ARPN Journal of Engineering and Applied Sciences

[9] Pia Singh, Prof. Karamjeet Singh, IMAGE ENCRYPTION and DECRYPTION USING blowfish algorithm[8]IN MATLAB, International Journal of Scientific and Engineering Research, Volume 4, problem 7, July-2013 150 ISSN 2229-5518

[10] Tanjyot Aurora1, Parul Arora, Blowfish Algorithm, ISSN 2319-7080 International Journal of Computer Science and Communication Engineering IJCSCE Special problem on "Recent Advances in Engineering and Technology" NCRAET-2013

[11] AshwakALabaichi, Faudziah Ahmad, RamlanMahmod, Security Analysis of Blowfish algorithm, ISBN: 978-1-4673-5256-7/13,IEEE

[12] TingyuanNie, Chuanwang Song, XulongZhi, Performance Evaluation of DES[3] and Blowfish Algorithms, Supported by Shandong Province Natural Science Foundation (ZR2009GL007) and A Work of Shandong Province Higher Educational Science and Technology Program (J09LG10) 978-1-4244-5316-0/10/$26.00 ©2010 IEEE

[13] MsNehaKhatri – Valmik1, Prof. V. K Kshirsagar, Blowfish Algorithm, IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN: 2278-0661, p-ISSN: 2278-8727Volume 16, problem 2, Ver. X (Mar-Apr. 2014), PP 80-83

[14] Ankita Deshpande, P.S.Choudhary, fpga[13] Implementation of Blowfish Cryptographic Algorithm, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 4, problem 4, April 2014

[15] Bruce Sctmeier, Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish), Counterpane Systems, 730 Fair Oaks Ave, Oak Park, IL 60302 sclmeier@chinet.com