Optimization of Zuc Stream Cipher to Attain High Throughput (Gbps)

Laya Krishna Patibandla

Abstract*: Mobile communication plays a vital role in our day to day life. Huge amounts of personal data is stored and shared in the mobile communications. The evolution of mobile generations and the ever increasing threat to the user's information has pressed for the need of secure, safer and more efficient algorithms. For this, we go for EEA3/EIA3 based ciphering algorithm to ensure secure handling of data in mobile communications with LTE. In this regard the paper discusses about a Chinese cipher called ZUC. The paper lays emphasis on the optimization of ZUC algorithm to attain higher throughputs on FPGA.

***Key Terms*: cipher, LTE, optimization, throughput**

I. INTRODUCTION

Cipher in simple terms is a system of encryption and decryption. Communication systems rely on ciphers to maintain the privacy and security of the data being handled. Since a lot of user information is being transmitted to and fro in the communication systems, there is a need for appropriate method to conceal and transmit the data to the designated destination safely without any third party being able to tap this information[8]. Ciphers are such encryption-decryption systems designed for secure handling of data in the communication. Based on the type of data handling, ciphers mainly comprise of stream ciphers and block ciphers. Based on the key being used on the encryption and decryption sides, the ciphers are classified into symmetric and asymmetric ciphers. Stream cipher falls under symmetric encryption algorithms. The synchronous stream ciphers seem to be immune to error propagation. This is because each bit in a stream cipher is independently encrypted /decrypted. Stream ciphers tend to have significant software efficiency and speed when compared to block ciphers[1]. These advantages that stream ciphers have, prove that they have got an edge over block ciphers in several communication protocols in wired as well as wireless communication.

Block ciphers handle few bits of data together. The data being encrypted is called the plain text. P-bits of data is enciphered with a key to form p-bits of encrypted data . They usually include memory-less algorithms. Stream ciphers can be seen to have the concept of internal states and therefore include memory oriented algorithms for the same. They encrypt/decrypt in a serial fashion since each bit of plain text is XORed with the key to form the cipher text. The stream ciphers generate a string of random bits as key, called the key-stream. They are called key-stream generators for the same reason. The key-stream bits produced are XORed in a bit by bit fashion with the plain text/cipher text to encrypt/decrypt [6].

II. ZUC STREAM CIPHER

Academic and industrial research has given us many stream cipher algorithms. ZUC is one of them. It was proposed by a scientist Zu Chongzhi at "Data Assurance and Communication Security Research Center (DACAS) of the Chinese Academy of Sciences"[3]. ZUC algorithm accepts initial key and initial vector as inputs, and outputs a key-stream. The initial vector and initial key are of 128 bit length each. The output key produced is of 32 bit length. This 32 bit key produced is then XORed with the plain text to obtain the cipher text. The basic structure of a ZUC stream cipher is illustrated in the figure 2.1 shown below.

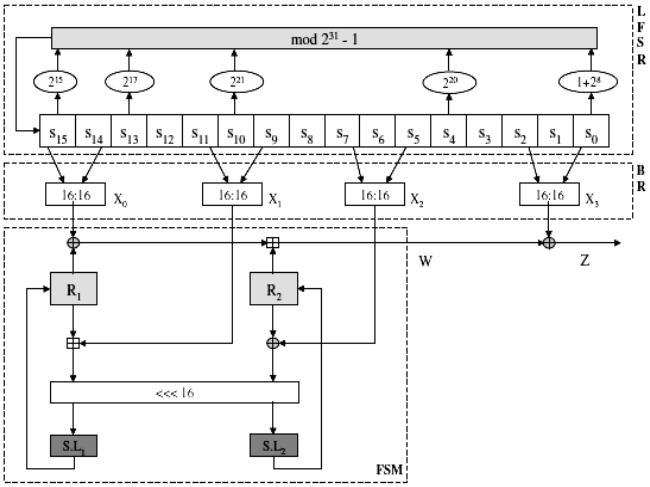

Fig. 2.1 the basic structure of ZUC stream cipher

The topic of utmost concern is whether these cipher algorithms could be efficiently executed on hardware-platform to achieve higher throughput[7]. However, after experimenting with the ZUC algorithm and taking its simulation and synthesis results into consideration, we learn that the linear feedback shift register block which is the crux of this cipher is pretty efficient in hardware[9].

Therefore we consider it is practically viable to deploy and experiment ZUC on a suitable hardware-platform and achieve higher throughput.

Revised Manuscript Received on June 07, 2019.
Laya Krishna Patibandla, Electronics and Communication Engineering, JNTU/ SNIST/Hyderabad, INDIA.

III.METHODOLOGY

The ZUC algorithm accepts two inputs - initial key and initial vector, each of 128 bit length. The key-stream output is of 32-bit size and it is therefore a 32 bit stream cipher. The key-stream generated is XORed with the plain text bits to obtain the cipher text[7]. It is seen that if we XOR key-stream with plain text on the encryption side, we obtain our cipher text, similarly if we XOR key-stream with cipher text on the decryption side, we obtain our plain text.

The execution of this algorithm happens in two phases: the initialization phase and the working phase[9]. In the former, the input key and vector are loaded into the registers of linear feedback shift register. Then the cipher is triggered with clock. The initialization phase executes to fill the registers with random values after the numerous shift operations[11]. In the latter phase, a 32-bit word output (key-stream or key word) is produced for each iteration.

As stated in the specifications of ZUC [3],its general structure comprises of three logical layers. The figure 1 below shows the general structure of ZUC.

The three layers/blocks that form the structure of ZUC are

(i)Linear Feedback Shift Register (LFSR)

(ii) Bit-Reorganization (BR) and

(iii) Nonlinear Function / *FSM* procedure.

A. Different Blocks of ZUC

LFSR --- The linear feedback shift register block has 16 state variables/ registers each of 31-bit length. The LFSR block has two kind of operating modes: "the initialization mode and the working mode". Algorithm 1 shows the various steps involved in the initialization mode.

Here u is the 31 bit input fed from Right shifting 32 bit output *W* of the FSM block by 1 bit.

Algorithm 1. LFSRWithInitialisationMode

Input: u

begin
- $v = \{2^{15}S_{15} + 2^{17}S_{13} + 2^{21}S_{10} + 2^{20}S_4 + (1+2^8)S_0\} \bmod (2^{31}-1)$;
- $S_{16} = (v+u) \bmod (2^{31}-1)$;
- if $S_{16} = 0$ then
  - set $S_{16} = 2^{31}-1$
- $(S_1, S_2, \ldots, S_{15}, S_{16}) \rightarrow (S_0, S_1, \ldots, S_{14}, S_{15})$;

In the working mode, we feed no input to the LFSR, and it contains the steps as discussed in Algorithm 2.

Algorithm 2. LFSRWithWorkMode

begin
- $S_{16} = \{2^{15}S_{15} + 2^{17}S_{13} + 2^{21}S_{10} + 2^{20}S_4 + (1+2^8)S_0\} \bmod (2^{31}-1)$;
- if $S_{16} = 0$ then
  - set $S_{16} = 2^{31}-1$
- $(S_1, S_2, \ldots, S_{15}, S_{16}) \rightarrow (S_0, S_1, \ldots, S_{14}, S_{15})$;

BR --- The Bit Reorganization accepts half word inputs from few registers of LFSR and forms four 32-bit words as shown below in Algorithm 3.

Algorithm 3. Bitreorganization

begin
- $X_0 = S_{15H} \parallel S_{14L}$;
- $X_1 = S_{11L} \parallel S_{9H}$;
- $X_2 = S_{7L} \parallel S_{5H}$;
- $X_3 = S_{2L} \parallel S_{0H}$;

Here $\parallel$ stands for concatenation operation. The two half words are joined together and the result is a 32 bit value. The value obtained is stored in the registers X_0, X_1, X_2 and X_3.

FSM --- The R_1 and R_2 are two 32-bit memories/registers in the nonlinear function FSM. The inputs to this block come from the output registers of Bit Reorganization block. The output of this procedure is *W* of 32 bit length[10]. This output W is right shifted by one bit and the resultant 31 bit is fed as input to the register of LFSR during the initialization stage of LFSR.

The FSM block involves picking values from the S boxes in a logical fashion. The S boxes are specific to the algorithm and they are constant values. The usage of combinational S boxes here helps induce non linearity into our algorithm, hence the name non linear block[12].

The steps of the FSM block or the nonlinear function *F* are illustrated in Algorithm 4.

Algorithm 4. The Nonlinear Function *F*

Input: X_0, X_1, X_2

1 begin
2. $W = (X_0 \oplus R_1) \boxplus R_2$;
3. $W_1 = R_1 \boxplus X_1$;
4. $W_2 = R_2 \oplus X_2$;
5. $R_1 = S(L_1(W_{1L} \parallel W_{2H}))$;
6. $R_2 = S(L_2(W_{2L} \parallel W_{1H}))$;

Where L1 and L2 are transformation functions and S is the S box specific to this algorithm from which values are picked.

The L1 and L2 are linear transformation functions defined as

L1(X) = X ^ (X $<<<_{32}$ 2) ^ (X $<<<_{32}$ 10) ^ (X $<<<_{32}$ 18) ^ (X $<<<_{32}$ 24)

L2(X) = X ^ (X $<<<_{32}$ 8) ^ (X $<<<_{32}$ 14) ^ (X $<<<_{32}$ 22) ^ (X $<<<_{32}$ 30)

The format "M $<<<_{32}$ N" depicts an N bit circular shift on M which is a 32 bit number.

.

IV. EXECUTION OF ZUC

As discussed earlier, execution of ZUC could be categorized into two steps:

(i) Initialization stage
(ii) working stage

During the initialization process, the following

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

functions are called for 32 iterations

1. BitReorg();
2. $W = F(X_0, X_1, X_2)$;
3. $LFSRInitialisationMode(W >> 1)$;

The control flows into the working stage after completing the initialization. In this, the algorithm performs the following function calls one time.

1. BitReorg();
2. $F(X_0, X_1, X_2)$;
3. $LFSRWorkMode()$;

After this the algorithm starts key-stream generation. For every iteration of the loop, the following functions are called once, and a key stream Z of 32 bit is generated:

1. BitReorg();
2. $Z = F(X_0, X_1, X_2) \wedge X_3$;
3. $LFSRWorkMode()$;

V. OPTIMIZATION TECHNIQUE

Initially the experimental results for ZUC algorithm were taken from the code compiled in ANSI C. After obtaining the results, the corresponding Handel-C code was written and modified to suit the syntax accordingly. The outputs from ANSI C and Handel C were verified taking ANSI C values as standard.

As a beginners code, the ZUC algorithm consumed about 90+ clock cycles for the generation of key-stream and 1000+ clock cycles for all the steps in the code. It was here the advantage of Handel-C was harnessed. The 'par' keyword available in Handel-C helped optimize the code through parallelism and usage of 'signal' keyword helped implement complex equations in a pipelined fashion. The S boxes that were initially stored in distributed RAM memory were later placed onto Block RAM memory for faster retrieval of S box values. The carry save adder scheme was chosen to implement our optimization. All these put together have made it easier to reduce the clock cycle consumption to one and hence increase the throughput.

In the first place, the number of modulo $(2^{31} - 1)$ additions required to update the registers/state variables of the LFSR block were five. In order to accomplish high efficiency and low overhead during implementation of the algorithm on FPGA, we split the calculation of v into two parts. By doing so the cost of whole calculation boils down to two clock cycles.

In the LFSR , consider the equation

$$v = \{ 2^{15} S_{15} + 2^{17} S_{13} + 2^{21} S_{10} + 2^{20} S_4 + (1+2^8) S_0 \} \bmod (2^{31}-1)$$

and

let $A = 2^{15}S_{15}$, $B = 2^{17}S_{13}$, $C = 2^{21}S_{10}$, $D = 2^{20}S_4$, $E = S_0$, $F = 2^8 S_0$.

We can now see how this scheme functions. It takes only two clock cycles to update state variables of the linear feedback shift register block as follows.

Consider the equation of v and the above assumptions to interpret v as $\{A + B + C + D + E + F\} \bmod (2^{31} - 1)$. Now let us split the six terms into three pairs and compute them individually.

$$T1 = \{A + B\} \bmod (2^{31} - 1)$$

$$T2 = \{C + D\} \bmod (2^{31} - 1)$$

$$T3 = \{E + F\} \bmod (2^{31} - 1)$$

In the first clock cycle, T1,T2 and T3 are calculated in a parallel fashion. Then the equation v is computed in the second clock signal, as shown in the below figure 5.1.

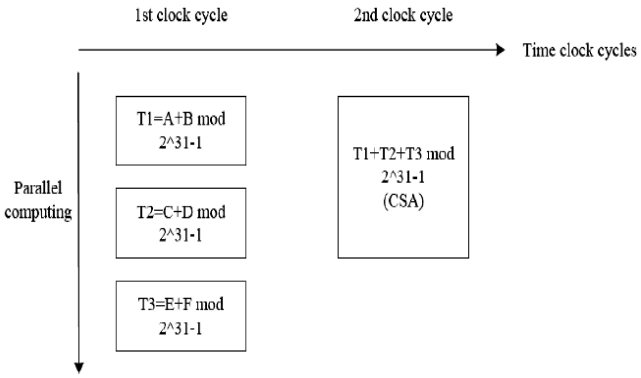

Fig. 5.1 parallel compuatation of T1,T2,T3

By splitting the calculations and computing them in a parallel manner, the calculation of equation v is reduced to two modulo $(2^{31}-1)$ additions. Since this calculation is complex of all the steps and consumes more time, it becomes the critical path. By minimizing the time required for this calculation of *equation v*, we can accomplish an improved throughput.

Imbibing the usage of Carry-Save Adder (CSA) to implement the addition, we reduce the time taken for computation of v. The benefits of this adder were harnessed to calculate v and we were able to decrease the number of modulo-$(2^{31}-1)$ additions from three to two.

→Par
{
$T_1 = \{A+B\} (\text{mod } 2^{31}-1)$
$T_2 = \{C+D\} (\text{mod } 2^{31}-1)$
$T_3 = \{E+F\} (\text{mod } 2^{31}-1)$
}

→ $v = \{T_1+T_2+T_3\} (\text{mod } 2^{31}-1)$

The calculation of ' v ' is said to require more time than any other steps in the ZUC algorithm. Therefore this equation is said to be the critical path in this algorithm. For tackling this, the emphasis has been laid on minimizing the clocks consumed by the LFSR initialization and LFSR working modes' equations.

VI. RESULTS AND CONCLUSIONS

The ZUC algorithm has been simulated in DK Design Suite using the HandelC given by Mentor Graphics. HandelC rendered the flexibility to optimize the code and obtain a 32 bit key in a single clock cycle. Optimization of the code was possible because of simplified implementation of pipelining and parallelism.

The synthesis reports of the corresponding Verilog HDL code were generated in the ISE taking Virtex 7 Xilinx VC709 as the target device. The maximum frequency was obtained from the synthesis report as 153.229 MHz. From this the maximum throughput of the algorithm was calculated and found to be 4.9 GHz.

The ZUC algorithm has been simulated in DK Design Suite using the HandelC given by Mentor Graphics. HandelC rendered the flexibility to optimize the code and obtain a 32 bit key in a single clock cycle. Optimization of the code was possible because of simplified implementation of pipelining and parallelism.

The synthesis reports of the corresponding Verilog HDL code were generated in the ISE taking Virtex 7 Xilinx VC709 as the target device. The maximum frequency was obtained from the synthesis report as 153.229 MHz. From this the maximum throughput of the algorithm was calculated and found to be 4.9 GHz. Figure 6.1 shows the synthesis results of the ZUC stream cipher.

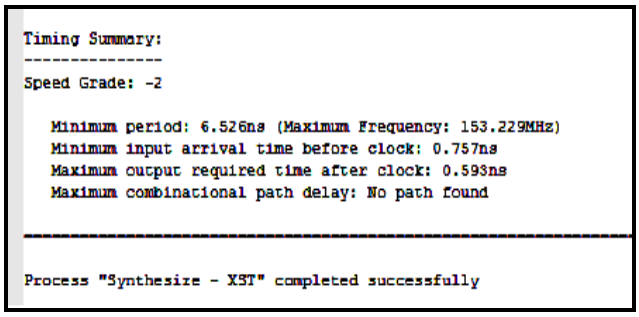

Fig. 6.1 the timing summary from the synthesis reports

The ZUC algorithm is seen to have the flexibility to be scaled and used more efficiently. The provision to imbibe a combination of S-boxes in the algorithm paves way for a larger non linearity and makes the algorithm robust to the cryptanalysis attacks.

Extensions to the existing generic ZUC algorithm - the X2 and X3 architectures have been proposed earlier. These architectures were designed to prove the scalability of the generic ZUC algorithm. Though their results on FPGA have not been witnessed in practical, when implemented on ASIC they were seen to produce larger throughputs than the generic ZUC algorithm[6].Therefore the ZUC stream cipher implementation on FPGA in the form of X2 and X3 architectures in future may also yield higher throughputs i.e almost ten folds greater than what we have seen in the results of generic algorithm.

VII. ACKNOWLEDGEMENTS

I thank Dr.Sudhir Reddy, Scientist, Department of Space, INDIA, for the constant support and encouragement he has rendered throughout my project work. His patience and constructive criticism has helped me improve my knowledge in composing my thesis and refining my paper.

REFERENCES

1. S. S. Gupta, A. Chattopadhyay, and A. Khalid, “Designing integrated accelerator for stream ciphers with structural similarities,” Cryptography and Communications, vol. 5, no. 1, pp. 19–47, 2013.
2. Z. Liu, L. Zhang, J. Jing, and W. Pan, “Efficient Pipelined Stream Cipher ZUC Algorithm in FPGA,” in The First International Workshop on ZUC Algorithm, December, pp. 2–3.
3. P. Kitsos, N. Sklavos, G. Provelengios, and A. N. Skodras, “FPGA-based Performance Analysis of Stream Ciphers ZUC, Snow3g, Grain v1, Mickey v2, Trivium and E0,” Microprocessors and Microsystems, 2012.
4. International Organization for Standardization, “ISO/IEC 18033- 4:2005: Information Technology – Security Techniques – Encryption Algorithms – Part 4: Stream ciphers”, 2005.
5. S. Sen Gupta, A. Chattopadhyay, and A. Khalid, “HiPAcc-LTE: An Integrated High Performance Accelerator for 3GPP LTE Stream Ciphers,” Progress in Cryptology{INDOCRYPT 2011}, pp. 196–215, 2011.
6. L. Zhang, L. Xia, Z. Liu, J. Jing, and Y. Ma, “Evaluating the optimized implementations of snow3g and zuc on FPGA,” in Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on, pp. 436–442, IEEE, 2012.
7. P. Kitsos, N. Sklavos, and A. Skodras, “An FPGA Implementation of the ZUC Stream Cipher,” in Digital System Design (DSD), 2011 14th Euromicro Conference on, pp. 814–817, IEEE, 2011.
8. C. Koc, “RSA hardware implementation,” RSA Laboratories, August, 1995.
9. 3GPP TS 33.401 v11.0.1 3rd Generation Partnership Project, Technical Specification Group Services and Systems Aspects. 3GPP System Architecture Evolution (SAE): Security Architecture. Release 11, June 2011
10. 3rd Generation Partnership Project: Long Term Evaluation Release 10 and beyond (LTEAdvanced) Proposed to ITU at 3GPP TSG RAN Meeting, Spain ,2009.
11. Debraize, B., Corbella, I.M.: Fault analysis of the stream cipher Snow 3G. In: Fault Diagnosis and Tolerance in Cryptography (FDTC’09), September ,2009.
12. Ekdahl, P., Johansson, T.: A new version of the stream cipher SNOW. In: Selected Areas in Cryptography (SAC’02), LNCS, vol. 2595, pp. 47–61. Springer, Heidelberg,2003.

AUTHOR’s PROFILE

P. Laya Krishna - M-Tech scholar in Digital Systems and Computer Electronics at Sreenidhi Institute of Science and Technology, an autonoumous institution affiliated to JNTU, Hyderabad.