International Journal of Current Engineering and Technology
E-ISSN 2277 – 4106, P-ISSN 2347 – 5161

Available at http://inpressco.com/category/ijcet

Research Article

High Performance WG Stream Cipher using Karatsuba Multiplier

Surya Rajan†* and Rincy Merin Varkey†

†Department of Electronics & Communication Engg., St. Joseph's College of Engineering & Technology, Palai, Kerala, India

Accepted 15 Nov 2015, Available online 20 Nov 2015, **Vol.5, No.6 (Dec 2015)**

Abstract

Large integer multiplication is one of the most resource and time consuming operation in cryptographic applications. So, efficiency in multiplication is very important. In this paper, architecture for high performance Welch-Gong(WG) cipher is explained. WG cipher is synchronous stream cipher based on WG transformations. These ciphers encrypt a plaintext or decrypt a cipher-text by adding the plain-text or cipher-text bit by bit with the generated key stream bits. The performance of the WG Cipher is increased by using high speed Karatsuba Multiplier. This multiplier works based on Karatsuba Algorithm which reduces the multiplication of two n-digit numbers to at most $n^{\log_2 3} \approx n^{1.585}$ single-digit multiplications in general. It is therefore faster than other algorithms. The WG cipher has been designed to produce key-stream with guaranteed randomness properties and is resistant to Time/Memory/Data trade-off attacks, algebraic attacks and correlation attacks. The architecture is synthesized using Xilinx ISE Design Suite14.2 and simulated using Xilinx ISim.

Keywords: *Welch-Gong transformation, Galois Field, Optimal Normal Basis (ONB), Karatsuba Algorithm, Stream Ciphers.*

1. Introduction

The expansions of computers and communication systems during the last decades has brought an increasing demand for methods to protect information in digital form. So, the cryptography took an important role in most of data exchange applications. Cryptographic algorithms are divided into Symmetric key algorithms and Asymmetric key algorithms. In Symmetric key algorithm, same secret key is shared by both the sender and receiver and in asymmetric key algorithms, both sender and receiver have different key. The symmetric key algorithms are further divided into Block Cipher and Stream cipher.

The cryptographic applications needs to compute large number operations. The numbers used in these applications may go up to hundreds or thousands of digits. Multiplication of such large numbers requires huge resource of time and space. A system's efficiency is generally determined by the performance of the multiplier because the multiplier is generally the slowest element in the system. It is vital to lower the requirements of that operation to provide efficient computation on those devices. The study of fast multiplication algorithms is always an important task done by mathematicians and computer scientists. In the 1960's, Russian mathematicians developed a faster way for multiplying two numbers. The technique became attributed to Karatsuba, and so the technique is called Karatsuba's multiplication.

The Welch-Gong(WG)(29, 11) [29 corresponds to Galois Field (2^{29}) and 11 is the length of the LFSR] is a stream cipher submitted in phase 2 of the eSTREAM project. It consists of consists of a WG key-stream generator which produces a long pseudo-random key-stream. The key stream is XORed with the plain-text to produce the cipher-text. It has been designed based on the WG transformations to produce key bit-streams with mathematically proved randomness aspects.

In WG cipher is also secure against algebraic attacks. Therefore, the WG(29, 11) is secure and has the randomness properties that cannot be offered by other ciphers. Hence, it has a potential to be adopted in practical applications. Despite of its attractive randomness and cryptographic properties, direct design using computation in the optimal normal basis (ONB), which reduces multiplications and inversion over Galois Field (2^{29}) is adopted.

In this paper, the implementation of a high performance WG Cipher is focussed. The performance of the WG Cipher is improved by using the high speed Karatsuba Multiplier based on Karatsuba algorithm. It was developed by Anatolii Alexeevitch Karatsuba in 1960. It reduces the multiplication of two n-digit numbers to at most $n^{\log_2 3} \approx n^{1.585}$ single-digit multiplications. It is therefore faster than the classical algorithm, which requires n^2 single-digit products

*Corresponding author **Surya Rajan is a PG Scholar and Rincy Merin Varkey** is working as Assistant Professor

faster, for sufficiently large n. Hence the multiplier of Galois field based on Karatsuba's divide and conquer algorithm allows for speedup of the top-level key algorithms.

2. WG Stream Cipher

WG cipher is a synchronous stream cipher which consists of a WG keystream generator. WG keystream generator produces a long pseudo-random key-stream. The key-stream is XORed with the plain-text to produce the cipher-text(encryption).The cipher-text is decrypted to give the original information. The WG cipher can be used with keys of different length like 80, 96, 112 and 128 bits. An initial vector (IV) of size 32 or 64 bits can be used with any of the above key lengths. Same length IVs and secret key can increase the security.

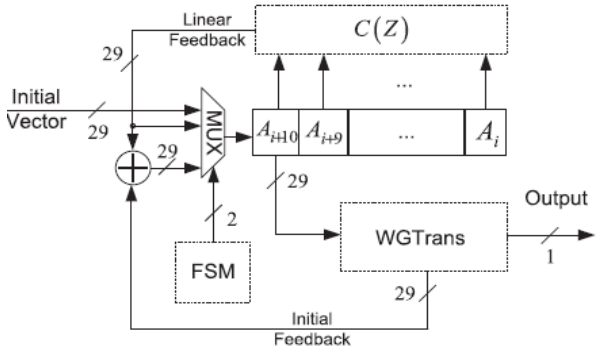

Fig.1 WG Stream Generator

The structure of the WG Generator is shown in fig 1. The WG keystream generator consists of an 11-stages LFSR over Galois field(2^{29}),29-bit WG transformation,LFSR feedback polynomial (C(Z)),Finite State Machine block (FSM) and Multiplexer.The LFSR feedback polynomial. The LFSR feedback polynomial:-

$$C(Z)= Z^{11} \oplus Z^{10} \oplus Z^{9} \oplus Z^{6} \oplus Z^{3} \oplus Z \oplus \beta \quad (1)$$

is a primitive polynomial of degree 11 over GF(2^{29}), where $\beta= \alpha^{464730077}$ is the generator of the ONB and α is a root of the defining polynomial of GF (2^{29}) given by

$$g(Z) = Z^{29} \oplus Z^{28} \oplus Z^{24} \oplus Z^{21} \oplus Z^{20} \oplus Z^{19} \oplus Z^{18} \oplus Z^{17} \oplus Z^{14} \oplus Z^{12} \oplus Z^{11} \oplus Z^{10} \oplus Z^{7} \oplus Z^{6} \oplus Z^{4} \oplus Z \oplus 1 \quad (2)$$

The output of the LFSR at A_{i+10} is filtered by an orthogonal 29-bit WG transformation given by

$$\text{WGTrans} = Tr(\text{WG Perm}(A_{i+10} \oplus 1)) \quad (3)$$

$$\text{WGPerm}(X) = 1 \oplus X \oplus X^{r1} \oplus X^{r2} \oplus X^{r3} \oplus X^{r4} \quad (4)$$

is the WG permutation, $r1 = 2^k + 1$, $r2 = 2^{2k} + 2^k + 1$, $r3 = 2^{2k} - 2^k + 1$, $r4 = 2^{2k} + 2^k - 1$, and k=10.

The WG ciphers consist of three phases of operations:- loading phase, key initialization phase, and running phase. The initial vector, linear feedback signal, initial feedback signal are the inputs provided to the LFSR along with the key that is provide internally. Initial vector is a random block of bits which is used to introduce randomness to the output of the cipher. The phases of the operations in the cipher is controlled by Finite State Machine (FSM).The loading phase takes 11 clock cycles in which the secret key and the initial vector are loaded to the LFSR, one stage at a time. The key initialization phase takes 22 clock cycles, during which the input to the LFSR is the bit-wise XOR of the Linear Feedback from the LFSR with the Initial Feedback signal WG Perm ($A_{i+10} \oplus 1$).The output of the key initialization phase serves as a session key for the running phase. By receiving the 34-th clock signal, the generator starts to generate its key-stream bits. While in the run phase, the only input to the LFSR is the Linear Feedback signal.

2.1 Finite State Machine

The architecture of Finite State Machine is shown in fig 2.

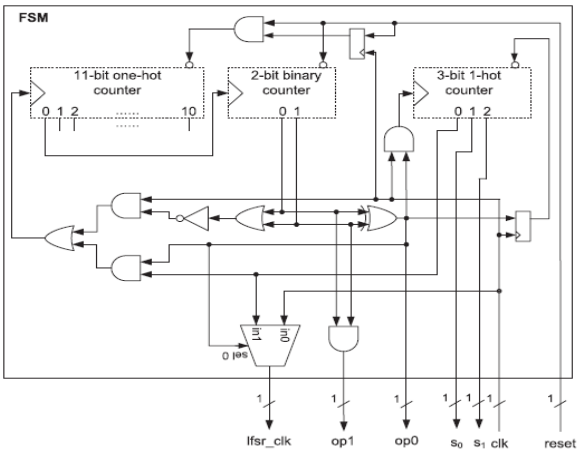

Fig.2 Finite State Machine

It has set of control signals like lfsrclk, s_0 and s_1. These are required for the serial computation of the initial feedback signal. Before each run of the cipher, the FSM resets its 11-bit one-hot counter to (1, 0, . . . , 0) and its 2-bit binary counter to (0, 0). When the reset signal is released, the 2-bit binary counter starts counting. At the same time, the 11-bit one-hot counter's reset input stays zero for an extra clock cycle. This assures that the (1, 0, . . . , 0) state of the 11-bit one-hot counter consumes a clock cycle at the beginning of the loading phase. So, it triggers the clock input of the 2-bit binary counter. The 2-bit binary counter changes its state to (1, 0), triggering the start of the key initialization phase. Then, the clk signal starts activating the clock input of the 3-bit one-hot counter. During this phase, the first output bit of the 3-bit one-hot counter drives the clock input of the 11-bit one-hot counter. Therefore, it takes 33 clock cycles for the 11-bit one-hot counter to complete 11 counts. Hence, it takes 33

clock cycles for the 2-bit binary counter to increment. When the running phase starts, with the 2-bit binary counter's state at (1, 1), the 11-bit and the 3-bit one-hot counters stop counting, as their clock inputs become idle.(Hayssam El-Razouk *et al* ,2014).

During the key initialization phase, the lfsr_clk is driven by the first output of the 3-bit one-hot counter. Hence, the LFSR shifts once every three clock cycles. The two signals s0 and s1 are derived from the 3-bit one-hot counter output according to table given below.

Table 1 Signal s_0 and s_1 as a function of the output of 3-bit one-hot counter

3-bit one-hot counter			S_1	S_0
bit 2	bit 1	bit 0		
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0

2.2 WG Transformation

The concept of optimal normal basis is introduced to reduce the hardware complexity of multiplying field elements. The multiplier uses an linear transformation to covert the normal basis representation of elements to suitable polynomials. These polynomials are multiplied according to the implementation platform. The normal basis is suitable for hardware implementation and squaring can be done by cyclic shift which is free of hardware.

Proposition 1: In a type-II ONB, the trace of the field multiplication of any two GF (2m) elements A = (a_0, a_1.., a_{m-1}) and B = (b_0, b_1, ..., b_{m-1}) is computed as the inner product of A and B as shown in equation below:

$$\text{Tr(AB)}=\sum_{i=0}^{m-1}(aibi) \quad (5)$$

The trace of the field multiplication of the two elements A and B can be computed using above equation. So the signals can be generated using the properties of trace function and thus WGTrans is computed as shown below with reduced number of field multiplications.

$$\text{WGTrans (X)} = Tr(1\oplus X \oplus X^{r1})+Tr\,(X^{2^{\wedge}\,2k}(X^{r1}\oplus X^{2\wedge k-1} \oplus X^{2\wedge 3k(2\wedge k-1)})) \quad (6)$$

The architecture of the WG transformation is designed from this equation and is shown in figure 3.

The keystream bits are obtained by XOR-ing the signals received from the inner product block and Galois field adder and the keystream bits are generated. The hardware cost of the WG cipher is dominated by its transform's field multipliers. Any decrease in the number of these multipliers would minimize the area of the overall cipher. This architecture uses five field multipliers.

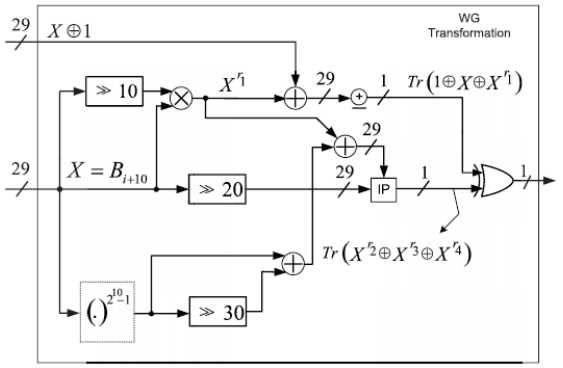

Fig.3 Design of WG transformation

2.3 Serialized Key Initialization Module

Serialized Key Initialization Module (SKIM) generates the initial feedback signal through serialized computation. The finite field multiplier used in the WG transformation block is used to achieve multiplication operation during the serialized computations. The function of the SKIM module is to compute a partial value of initial feedback signal and stores it in Register R2. The architecture of the SKIM module is shown in figure below.(Hayssam El-Razouk *et al* ,2014).

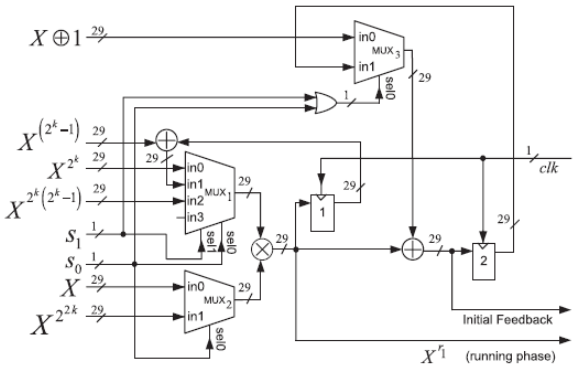

Fig.4 Architecture of SKIM

Initial feedback signal can be computed serially over 3 clock cycles, hence this complete round of serialized initial feedback computation is known as extended key initialization round. The control signals generated using FSM module is as inputs. During the extended key initialization round, the two signals $s0$ and $s1$ in Fig. 4 change values in each stage.. These two signals control the outputs of the three multiplexers MUX1, MUX2,and MUX3.In each stage of the extended key initialization round, the SKIM module computes a partial value of the initial feedback signal and stores it in Register 2.Hence, MUX1, MUX2, andMUX3 generate the signals X^{2k},X,and $X\oplus 1$ at their outputs, respectively.

2.4 Key Initialization and Operation

The recommended key sizes for WG cipher are 80, 96, 112 and 128 bits. An Initialization Vector (IV) of size 32 or 64 bits can be used with any of the above key

sizes. Initialization of the cipher is done by loading key bits and IV bits into the LFSR. To increase security, IVs of the same length as the secret key can also be used. In this work, both the Key and Initialization Vector have same bit size of 128 bit.(Y. Nawaz *et al*,2008).

Once the cipher has been initialized, the contents of the LFSR constitute the internal state of the cipher. To produce the running keystream, LFSR is clocked once and the contents of the stage 11 are fed into the WG transformation block which produces a single bit of the running keystream. The LFSR is clocked again and the updated contents of stage 11 are again fed to WG transformation block thus producing the next bit of the keystream and so on. This running keystream is added bitwise (XORed) to the plaintext to produce the ciphertext.The cipher text is added bitwise with the keystream to produce the given plaintext.

3. High Performance WG Stream Cipher

Multipliers plays a vital role in the performance of cryptosystems. Large integer multiplication is one of the most resource and time consuming operation in cryptography. It is vital to lower the requirements of that operation to provide efficient computation on those devices. Multipliers have large area, long latency and consume considerable power. Hence, optimizing the speed and area of the multiplier is a major design issue.

Multiplication is an important operation that is critically used in the integrated block of WG transformation and SKIM module to generate keystream bits.Hence, a fastest multiplier can improve the overall performance of the Wg Cipher. Karatsuba multiplier is one of the fastest multiplier.

3.1 Karatsuba Algorithm

The Karatsuba algorithm is a fast multiplication algorithm. It reduces the multiplication of two n-digit numbers to at most $n^{log_2 3} \approx n^{1.585}$ single-digit multiplications.It is therefore faster than the other algorithm, which requires n^2 single-digit products.

The basic step of Karatsuba's algorithm is a formula that allows us to compute the product of two large number using three multiplications of smaller numbers, each with about half as many digits, plus some additions and digit shifts.(Wajih E,Mohsen *et al*,2008).

This algorithm can also be applied for the multiplication of the binary numbers. In this approach, we take the two binary numbers X and Y and split them each into their most-significant half and their least-significant half, A and B respectively.

$$X = 2^{n/2}A + B \quad (7)$$

$$Y = 2^{n/2}C + D \quad (8)$$

The product of X and Y can be calculated as

$$X * Y = (2^{n/2}A + B)*(2^{n/2}C + D)$$

$$= 2^{n}AC + 2^{n/2} BC + 2^{n/2} AD + BD \quad (9)$$

Using the above equation as a recursive multiplication algorithm, we need to perform four n/2 multiplications, three shifts, and three O(n)-bit additions. If we use T(n) to denote the running time to multiply two n-bit numbers by this method, this gives us a recurrence of

$$T(n) = 4T(n/2) + cn \quad (10)$$

for some constant c.The cn term reflects the time to perform the additions.

Rewriting the above formula,

$$(2^{n} - 2^{n/2})AC + 2^{n/2}(A + B)(C + D) + (1 - 2^{n/2})BD \quad (11)$$

This equation results in only three multiplications of size n/2, plus a constant number of shifts and additions. So, the resulting recurrence is

$$T(n) = 3T(n/2) + c'n \quad (12)$$

for some constant c'. By applying master theorem, this recurrence to $n^{log_2 3} \approx n^{1.585}$.

This fastest Karatsuba Multipliers is used in Galois Field operation of the WG Transformation and SKIM Module.Hence the performance of the WG cipher will get increase.

4. Results and comparisons

4.1 Results

The architecture was modeled using Verilog in Xilinx ISE Design Suite 14.2 and the simulation of the design is performed using Xilinx ISim to verify the functionality of the design. The WG Cipher mainly consists of WG Generator which consists of a keystream generator which produces a sequence of binary digits. This sequence is called the running key or the keystream.

The RTL schematic and simulation result of WG Generator is shown below.

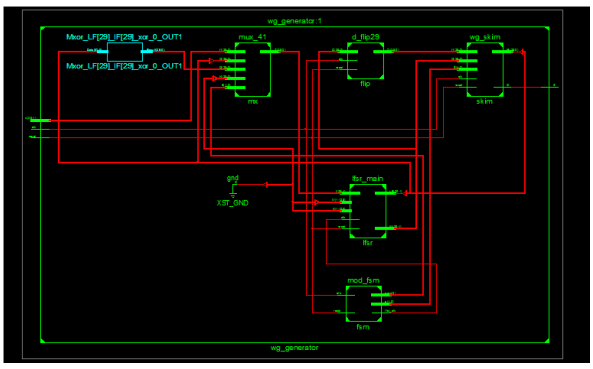

Fig.5 RTL Schematic of WG Generator

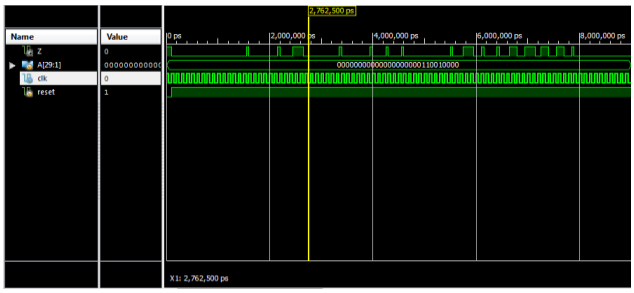

Fig.6 Output Waveform of WG Generator

The RTL schematic and simulation result of WG Cipher is shown below.

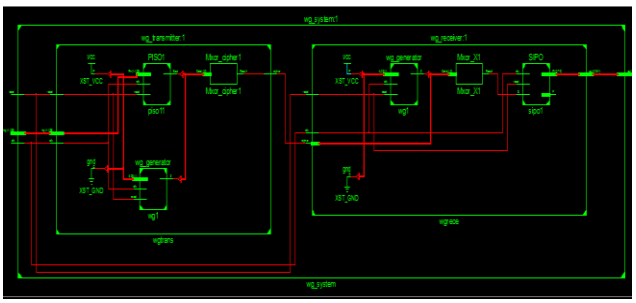

Fig.7 RTL Schematic of WG Cipher

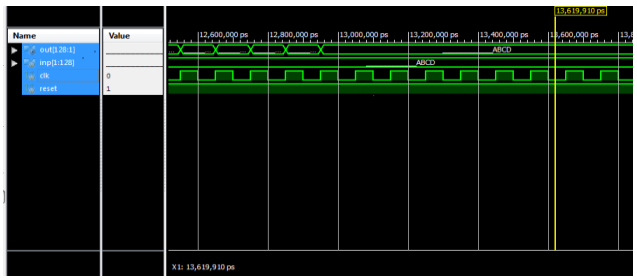

Fig.8 Output Waveform of WG Cipher

4.2 Comparison

A 32-bit Array multiplier and Karatsuba Multiplier was modelled using Verilog in Xilinx ISE Design Suite 14.2 and their delay was calculated. The table below shows the delay of the these multipliers.

Table 1 Delay Comparison of Karatsuba Multiplier and Array Multiplier

Type of Multiplier	Karatsuba Multiplier	Array Multiplier
Delay(Ns)	47.245	95.887

From the table, it is clear that Karatsuba Multiplier is fastest.

Multiplication is an important operation in the cryptographic applications.So, the effect of these multipliers on performance of WG Cipher was studied and the delay is shown below in the table.

Table 2 Delay Comparison of Wg Cipher

Wg Cipher	With Array Multiplier	With Karatsuba Multiplier
Delay(Ns)	40.125	24.512

So, from the table it is clear that the WG Cipher with Karatsuba has the lesser delay.

The figure below shows the synthesis report of the WG Cipher with the Array Multiplier and Karatsuba Multiplier especially showing the delay of the system.

```
LUT6:I3->O           11   0.205   0.987  skim/mult/mult1/mult1/Madd_P<53:49>_Madd_lut<2>1
LUT6:I4->O            1   0.203   0.580  skim/mult/mult1/mult1/Madd_P<58:54>_Madd_lut<2>1_
LUT6:I5->O            9   0.205   0.830  skim/mult/mult1/mult1/Madd_P<58:54>_Madd_lut<2>1
LUT6:I5->O            9   0.205   0.934  skim/mult/mult1/mult1/Madd_P<58:54>_Madd_xor<2>11
LUT6:I4->O            8   0.203   0.803  skim/mult/mult1/mult1/Madd_P<68:64>_Madd_lut<1>1_
LUT6:I5->O           10   0.205   0.961  skim/mult/mult1/mult1/Madd_P<68:64>_Madd_lut<1>1
LUT5:I3->O            7   0.203   1.002  skim/mult/mult1/mult1/Madd_P<68:64>_Madd_cy<2>1 (
LUT6:I3->O            3   0.205   0.898  skim/mult/mult1/mult1/Madd_P<78:74>_Madd_cy<2>1_SW
LUT6:I2->O           14   0.203   0.958  skim/mult/mult1/mult1/Madd_P<78:74>_Madd_cy<2>1 (
LUT4:I3->O            6   0.205   0.745  skim/mult/mult1/mult1/Madd_P<83:79>_Madd_lut<2>1
LUT6:I5->O           10   0.205   0.857  skim/mult/mult1/mult1/Madd_P<83:79>_Madd_xor<2>11
LUT6:I5->O            7   0.205   0.774  skim/mult/mult1/mult1/Madd_P<93:89>_Madd_lut<2>1_
LUT4:I3->O            7   0.205   1.002  skim/mult/mult1/mult1/Madd_P<93:89>_Madd_lut<2>1
LUT6:I3->O            7   0.205   0.774  skim/mult/mult1/mult1/Madd_P<98:94>_Madd_lut<2>1
LUT6:I5->O            5   0.205   0.819  skim/mult/mult1/mult1/Madd_P<98:94>_Madd_xor<2>11
LUT6:I4->O           16   0.203   1.005  skim/mult/mult1/mult1/Madd_P<108:104>_Madd_lut<1>1
LUT5:I4->O            1   0.205   0.808  skim/mult/mult1/mult1/Madd_P<108:104>_Madd_xor<2>1
LUT6:I3->O            7   0.205   0.878  skim/mult/mult1/mult1/Madd_P<113:109>_Madd_xor<1>1
LUT6:I4->O            1   0.203   0.580  skim/mult/mult1/adder2/fulladder12/XOR_0000021_xo<
LUT6:I5->O            2   0.205   0.617  skim/mult/mult1/adder2/fulladder12/XOR_0000021_xo<
LUT6:I5->O            1   0.205   0.684  Mxor_O_xo<0>47_SW1 (N281)
LUT6:I4->O            1   0.203   0.580  Mxor_O_xo<0>47 (O)
end scope: 'wgtrans/wg1/skim:O'
LUT3:I2->O            1   0.205   0.000  wgrece/X1 (wgrece/X)
FDRE:D                    0.102          wgrece/sipo11/temp_0
----------------------------------------
Total                    40.215ns (8.254ns logic, 31.961ns route)
                                  (20.5% logic, 79.5% route)
```

Fig.9 Delay of the WG Cipher with Array Multiplier

```
LUT6:I5->O            3   0.205   0.651  mult4/mult4/mult4/add2/fulladder4/XOR_0000
LUT6:I5->O            4   0.205   0.684  mult4/mult4/mult4/add3/fulladder4/Y<1>1 (m
LUT5:I4->O            4   0.205   0.684  mult4/mult4/mult4/add3/fulladder6/XOR_0000
LUT6:I5->O            4   0.205   0.684  mult4/mult4/add2/fulladder6/Y<1>1 (mult4/m
LUT5:I4->O            3   0.205   0.651  mult4/mult4/add2/fulladder8/XOR_0000021_xo<
LUT6:I5->O            6   0.205   0.745  mult4/mult4/add3/fulladder8/Y<1>1 (mult4/m
LUT6:I5->O            9   0.205   0.830  mult4/mult4/add3/fulladder9/Y<1>1 (mult4/m
LUT6:I5->O            2   0.205   0.617  mult4/mult4/add3/fulladder11/XOR_0000021_x
LUT6:I5->O            5   0.205   0.715  mult4/add2/fulladder11/Y<1>1 (mult4/add2/X
LUT5:I4->O           12   0.205   0.909  mult4/add2/fulladder13/Y<1>1 (mult4/add2/X
LUT6:I5->O            6   0.205   0.745  mult4/add2/fulladder19/Y<1>1 (mult4/add2/X
LUT6:I5->O            9   0.205   0.934  mult4/add2/fulladder21/XOR_0000021_xo<0>1
LUT6:I4->O            1   0.203   0.684  mult4/add3/fulladder19/Y<1>1_SW2 (N230)
LUT6:I4->O            9   0.203   0.830  mult4/add3/fulladder21/Y<1>1 (mult4/add3/X
LUT6:I5->O            6   0.205   0.849  mult4/add3/fulladder26/XOR_0000021_xo<0>1
LUT6:I4->O            2   0.203   0.721  add3/fulladder27/Y<1>1_SW4 (N138)
LUT6:I4->O            1   0.203   0.684  add3/fulladder25/Y<1>1_SW4 (N255)
LUT6:I4->O            7   0.203   0.774  add3/fulladder28/XOR_0000021_xo<0>1 (Z<27>
end scope: 'wgrece/wg1/skim/skim/mult/mult1:Z<27>'
LUT5:I4->O            1   0.205   0.684  add2/adder1/fulladder27/Y<1>1_SW1 (N108)
LUT6:I4->O            2   0.203   0.617  add2/adder1/fulladder29/XOR_0000021_xo<0>1
end scope: 'wgrece/wg1/skim/skim:IF<28>'
end scope: 'wgrece/wg1/skim:IF<28>'
LUT4:I3->O            1   0.205   0.000  wgrece/wg1/mx/Mmux_Z211 (wgrece/wg1/M1<29>
FDR:D                     0.102          wgrece/wg1/lfsr/W_29
----------------------------------------
Total                    24.152ns (5.252ns logic, 18.900ns route)
```

Fig.10 Delay of the WG Cipher with Karatsuba Multiplier

Conclusions

The arrival of 4G mobile technology led to the design of WG stream cipher. The WG cipher has been designed to produce key-stream with guaranteed randomness properties. It is secure against Time /Memory/Data trade-off attacks, algebraic attacks and correlation attacks. Multiplication is an important operation that is critically used in the integrated block of WG transformation and SKIM module to generate keystream bits.From the results ,it is clear that the delay of WG Cipher has reduced greatly when Karatsuba's divide and conquer algorithm based multiplier is used in Galois field operation. This improves the performance of the WG Stream Cipher

and hence can be used in high speed communication applications.

References

Hayssam El-Razouk and Arash Reyhani Masoleh, Guang Gong (2014),New Implementation of the WG stream cipher, *IEEE Transactions on VLSI Systems*,Vol 22, No.9, pp. 1865 – 1878.

Y. Nawaz and G. Gong (2008) ,WG: A family of stream ciphers with designed randomness properties, *Inf. Sci.*, vol. 178, no. 7, pp. 1903-1916.

G. Gong and Y. Nawaz. (2005), The WG Stream Cipher [Online].

G. Gong and A. Youssef (2002),Cryptographic properties of the Welch- Gong transformation sequence generators, *IEEE Trans. Inf. Theory*, vol. 48, no. 11, pp. 2837-2846.

C. Lam, M. Aagaard, and G. Gong,(2011) Hardware Implementations of Multi-output Welch-Gong Ciphers, *Technical Report, CACR* 2011-01, University of Waterloo.

S. Gupta, A. Chattopadhyay, K. Sinha, S. Maitra, and B. Sinha (2013),High performance hardware implementation for RC4 stream cipher, *IEEE Trans. Comput.,* vol. 62, no. 4, pp. 730-743.

Majid Bakhtiari, Mohd Aizaini Maarof,(2011)An Efficient Stream Cipher Algorithm for Data Encryption, *International Journal of Computer Science Issues,* Vol. 8, Issue 3, No. 1.

Galanis, M.D. Kitsos, P.Kostopoulos, G. Sklavos, N. Koufopavlou, O. Goutis,(2004) Comparison of the hardware architectures and FPGA implementations of stream ciphers, *IEEE International Conference on Electronics, Circuits and Systems,* pp.571-574.

Wajih E,Mohsen M, Medien Z, Belgacem B,(2008) Efficient hardware architecture of recursive Karatsuba-Ofman multiplier, *Design and Technology of Integrated Systems International Conference,IEEE,* page 1-6.

Gary C.T. Chow, Ken Eguroy, Wayne Luk and Philip Leong,(2010) A Karatsuba-based Montgomery Multiplier,*International Conference on Field Programmable Logic and Application,IEEE*.

Soniya, Suresh Kumar (2013) A Review of Different Type of Multipliers and Multiplier-Accumulator Unit,*International Journal of Emerging Trends Technology in Computer Science,* Vol.2, No.4, p.p.(364-368).