FPGA IMPLEMENTATION OF WELCH-GONG STREAM CIPHER USING VLM3 ALGORITHM

Elizabeth Scaria[1], Shinoj K. Sukumaran[2]

[1]PG Scholar, Dept. of ECE, Government Engineering College, Idukki, Kerala, India
[2]Assistant Professor, Dept. of ECE, Government Engineering College, Idukki, Kerala, India

---***---

Abstract - *WG stream ciphers generate key bit-stream based on Welch-Gong (WG) transformation whose hardware cost is dominated by its transform's field multipliers. Montgomery modular multiplication is an efficient method for hardware implementation of the modular multiplication. This project presents an Field Programmable Gate Array (FPGA) implementation of WG stream cipher using VLM3 algorithm. The same WG transformation block is implemented using a field array multiplier also. Comparison of the Variable Length Montgomery modular multiplication(VLM3) multiplier with the field array multiplier shows a hardware efficiency of 45% and power reduction of 5.25%. To check the working of the system, a transmitter and receiver using WG generator is designed.The system is verified by Verilog in ISim simulator and synthesized in Virtex5 FPGA.*

Key Words: **Welch-Gong (WG) transform, Montgomery modular multiplication, VLM3 algorithm, Serialized Key Initialization Module (SKIM), Public Key Cryptosystem.**

1. INTRODUCTION

Synchronous stream ciphers are lightweight symmetric-key cryptosystems which are widely used in wireless communication. A synchronous stream cipher consists of a keystream generator which receives a key (K) and an Initial Value (IV) to produce a sequence of binary digits. This sequence is called the keystream or a pseudo-random sequence. Stream ciphers encrypt a plain-text by XORing it bit-by-bit with the key-stream bits. Since the secret key is shared between the sender and the receiver, an identical keystream can be generated at the receiving end. The XORing of this keystream with the cipher text recovers the original plaintext [1] .

The Welch–Gong (WG)(29, 11) is a stream cipher submitted to the hardware profile in phase 2 of the eSTREAM project [2]. Here 29 corresponds to GF (2^{29}) and 11 indicates the length of the Linear Feedback Shift Register, LFSR. It has been designed based on the WG transformations [3] to produce key bit-streams with mathematically proved randomness aspects. Despite of its attractive randomness and cryptographic properties, few designs have been proposed for the hardware implementations of the WG (29, 11). The hardware cost of the WG cipher is dominated by its transform's field multipliers [4].

Modular multiplication is widely used operations in many public-key cryptosystems (PKCs) [5-7]. The throughput rate and the required number of modular multiplication performed determine the performance of many PKCs. Montgomery modular multiplication is an efficient method for hardware implementation of the modular multiplication. It replaces the trial division with a series of addition operations and divisions by a power of two [8]. These divisions can be easily implemented using simple left shifters. The problem is that addition requires time consuming carry propagations of the large operands. To overcome this problem and to improve the performance of Montgomery modular multiplication algorithm and architecture, high-radix technique, systolic array architecture, carry-save addition architecture, and scalable architectures are used [9].

Rezai and Keshavarzi, 2016 proposed the Variable Length Montgomery modular multiplication (VLM3) algorithm[10]. They proposed a new integer expansion which relaxes the high radix partial multiplication into simple binary partial multiplication. The design strategy is using the multibit-scan multibit-shift technique in one clock cycle. It also executes several addition operations, required for zero chain, in one clock cycle instead of several clock cycles.

The objective of this project is to design and implement WG-stream cipher using VLM3 algorithm and to compare it with the implementation using field array multiplier.

2. INTEGER EXPANSION

The expansion of an n-bit integer X is defined as $X_{SD} = (Z_{n-1} Z_{n-2} \dots Z_1 Z_0)$. Each digit of the expansion , Z_i, includes a pair (k_i, $f^{(i)}$), where $f^{(i)}$ denotes the number of zero bits in sequence, and ki denotes the coefficient [10]. Fig. 1 shows the block diagram for converting an integer X from binary representation to the required integer expansion.

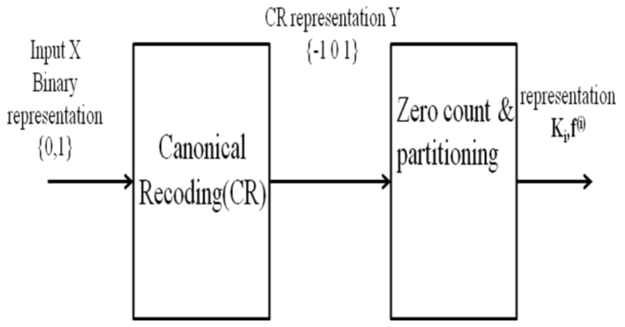


Fig -1: Block diagram for converting an integer X from binary representation to the integer expansion.

The canonical signed digit (CSD) representation is one of the existing signed digit (SD) representations with unique features. The CSD code is a ternary number system with the digit set {Ī 0 1}, where Ī stands for -1 [11]. The CSD representation for a given constant is unique and has two important characteristics: First, the representation has minimum number of nonzero digits. This is a minimum hamming weight representation which minimizes the required number of additions in arithmetic operations. Second, adjacent bits will never be nonzero. That is, the recoding is canonic. This property provides uniqueness characteristic.

For example for $X = (521022)_{10} = (1111111001100111110)_2$ The CR representation is $X_{CR} = (1000000\bar{1}010\bar{1}010000\bar{1}0)_{CR}$, which has only six nonzero digits. Canonical representation of the given integer X can be computed by applying Algorithm 6 in [10]. The schematic circuit for converting an integer from its binary representation to the corresponding CR representation is in [11].

Canonical recoded integer is split into unequal parts in this step. The count of zero bits in succession in each digit of the integer expansion is restricted to l, so that all digits in the expansion have an equal size in the hardware implementation. Here the optimal value of l is taken as two, which means that two successive zero bits followed by nonzero digit or three repeated zero bits [11]. For X in the above example, partitioning with $l = 2$ results in nine parts (1) (000) (000) (Ī0) (10) (Ī0) (10) (000) (Ī0).

In this step, we count successive zeros in each partition. Then each part is represented using this count and the coefficient. Thus each part is represented as a pair, $(k_i, f^{(i)})$. k_i indicates the sign of the nonzero digit. It is used for binary partial multiplication. At the same time, $f^{(i)}$ is the number of consecutive zero bits in each partition. Multibit shift is achieved using , $f^{(i)}$. For the same X in the above example, this step results in (1,0) (0,3) (0,3) (Ī,1) (1,1) (Ī,1) (1,1) (0,3) (Ī,1).

In hardware implementation, each pair is characterized by three bits. k_i is represented using a single bit and $f^{(i)}$ using two bits. k_i is as shown below:

$$K_i = \begin{cases} 0, \text{for positive integers} \\ 1, \text{for negative integers} \end{cases}$$

Therefore, hardware representation for the integer expansion of the above example is obtained as (000) (011) (011) (101) (001) (101) (001) (011) (101). Partitioning and zero counting can be carried out concurrently using Algorithm 7 in [10].

2. VLM3 ALGORITHM AND ARCHITECTURE

The design strategy is using the multibit-scan multibit-shift technique in one clock cycle. The reformulation of the Montgomery modular multiplication is done based on a new multiplier expansion. It relaxes the high-radix partial multiplication to binary multiplication. Algorithm shown below is used to perform Variable Length Montgomery modular multiplication [10].

VLM3 Algorithm

```
Input: $X_{SD}$ ($k_i$,$f^{(i)}$),Y,M;
Output: S=X.Y.$2^{-n}$ mod M
    1.  S(0) = 0;
    2.  **For** i = 0 to j-1
    3.  $a^{(i)} = f^{(i)}$
4. **If** $f^{(i)} = 3$ **Then** P(i) := S(i), $a^{(i)} = a^{(i)} - 1$;
5. **Else**
6.              **If** ki = 0 **Then** P := S(i) +$2^{a(i)}$ Y;
7.              **Else** P := S(i) - $2^{a(i)}$ Y;
8. qi =$P_{k...0}(2^{a(i)+1} - M^{-1}_{k...0})$ mod $2^{a(i)+1}$;
9. S(i+1)=(P(i) + qiM)/ $2^{a(i)+1}$;
10. **End for**
11. **If** S≥M **Then** S = S – M;
12. **Return** S;
```

X_{SD}, which is the expansion of the multiplier X, the n-bit multiplicand Y, and modulus M are the inputs to the Algorithm. Pc and Ps denote the carry and sum components of P. In this Algorithm $M_{k...0} = M \bmod 2^{k+1}$. In high-radix CSA Montgomery modular multiplication Algorithm [12],

$$(Pc(i), Ps(i)) = Sc(i) + Ss(i) + X^{(i)} \cdot Y \quad (1)$$

In VLM3 algorithm, because of the use of integer expansion of the multiplier this computation is relaxed to

$$P(i) = S(i) + 2^{a(i)} \cdot Y \quad \text{when } f^{(i)} \neq 3 \text{ and } k_i = 0 \quad (2)$$

$$P(i) = S(i) - 2^{a(i)} \cdot Y \quad \text{when } f^{(i)} \neq 3 \text{ and } k_i = 1 \quad (3)$$

or

$$P(i) = S(i) \quad \text{when } f^{(i)} = 3 \quad (4)$$

This is done in step 6, 7, and 4, respectively. These steps can be implemented using Multiplexer (MUX), and a Modified Barrel Shifter (MBS). A modified barrel shifter is designed so as to achieve only a limited number of shifts. Normally, several additions corresponding to consecutive zero bits are executed in several clock cycles. In VLM3 Algorithm, this can be executed in a single clock cycle. This action is shown in step 9. This can be implemented using MBS, adders and LUTs [10].

The data path of a basic cell that implements the VLM3 algorithm is shown in Fig. 2. It contains three MBS, two adders, four registers, a 3-bit shift register, a MUX, two XORs, a NAND gate, a $q^{(i)}$.M generator.

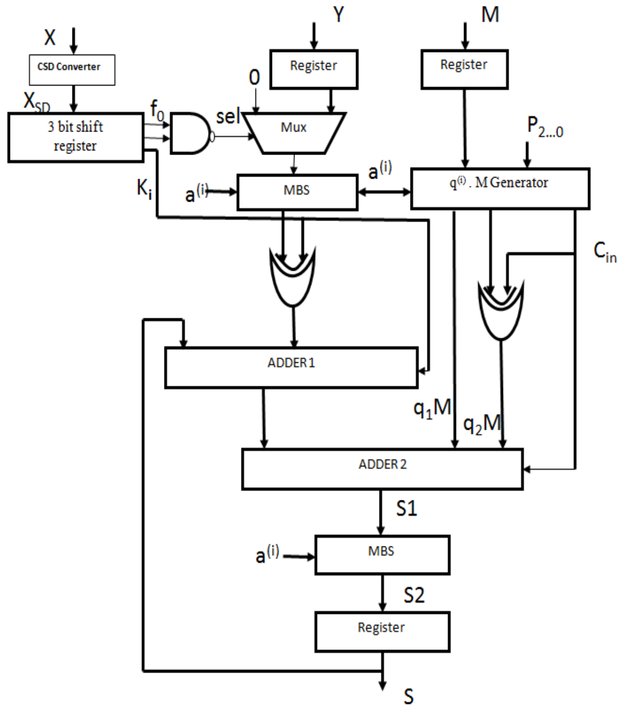


Fig -2: Data path of a basic cell of the VLM3 algorithm.

Each time the 3-bit sift register provides a single (k_i, $f^{(i)}$) pair of X_{SD}. From this $f^{(i)}$ is used to determine the select line of MUX1. Sel=0 when $f^{(i)}$=3, otherwise sel=1. k_i determines whether adder 1 acts as an adder or subtractor. Adder1 provides S(i) + $X^{(i)}$. Y in each clock cycle. Adder 2 provides P(i) + $q^{(i)}$. M. MBSs shift the inputs based on the value of $a^{(i)}$.

An LUT and two MBSs are used to compute $q^{(i)}$. M. For a(i)= 2, the coefficient q(i) depends on the least three bits of the partial results of adder1 P, and two bits of M, m2, and m1. Inorder to implement $q^{(i)}$. M generator, q(i) is split into qi and q2 which are powers of 2. Then shift M to get two components of $q^{(i)}$. M, q1 · M and q2 . M, based on q1 and q2. Finally, these components are added with P(i) in adder 2.

For example with $q^{(i)}$=4, we can split $q^{(i)}$ into q1 = 2 and q2 = 2 or q1 = 8 and q2 = - 4. Then, 4M can be replaced as 2M + 2M or 8M −4M. Note that the negative component, for example −4M in the previous example, is implemented by inverting the positive component, 4M, and introducing a $carry_{in}$ with the value of 1. Fig. 3 shows the architecture of the $q^{(i)}$. M generator.

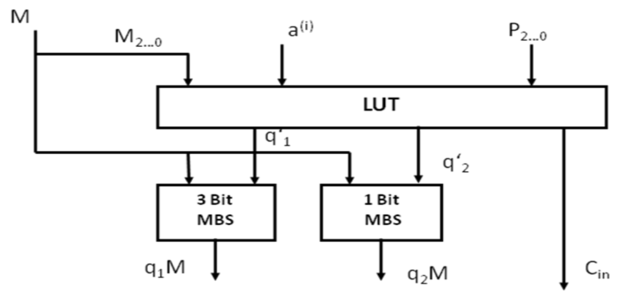


Fig -3: Architecture for the q(i) . M generator [10].

The architecture is implemented using a 3-bit full adder (FA), a LUT, a 1-bit MBS, and a 3-bit MBS. Tables 1-3 defines the LUT output. For a(i) = 0, the q(i) = P_0(i).

Table -1: LUT for determining $q^{(i)}$ for $a^{(i)}$ = 2 [10]

$P_{2...0}$(i)	m_2m_1			
	00	01	10	11
000	0	0	0	0
001	7	5	3	1
010	6	2	6	2
011	5	7	1	3
100	4	4	4	4
101	3	1	7	5
110	2	6	2	6
111	1	3	5	7

Table -2: LUT for determining $q^{(i)}$ for $a^{(i)}$ = 1 [10]

$P_{1...0}$(i)	m_1	
	0	1
00	0	0
01	3	1
10	2	2
11	1	3

The q_i' in table 3 shows the required number of shifts, i.e., $q_i = 2^{q_i'}$. The LUT outputs, q_1'and q_2', are the control signals for the MBSs that implement q1 . M and q2 . M. The LUT also has an output Cin which is asserted 1 whenever q2 . M is negative. This signal is used as a carry-in for the adder 2 in Fig. 2.

Table -3: LUT for determining the components of q(i) [10]

$q^{(i)}$	q_1	q_1'	q_2	q_2'	c_{in}
0	1	0	-1	0	1
1	2	1	-1	0	1
2	1	0	1	0	0
3	2	1	1	0	0
4	2	1	1	1	0
5	4	2	2	0	0
6	4	2	1	1	0
7	8	3	-1	0	1

3. ARCHITECTURE OF THE WG CIPHER

WG cipher is a synchronous stream cipher which consists of a WG keystream generator. Fig. 4 shows the block diagram for a WG generator. It consists of an 11 stage linear feedback shift register (LFSR) over $F2^{29}$ (GF (2^{29}), extension field of GF(2) with 2^{29} elements. Each element in this field is represented as a 29 bit binary vector). Initial Vector, IV is the input during the loading phase. Input is given by XORing linear feedback and initial feedback. Linear Feedback is the input throughout the Pseudo Random Signal Generator (PRSG) phase.

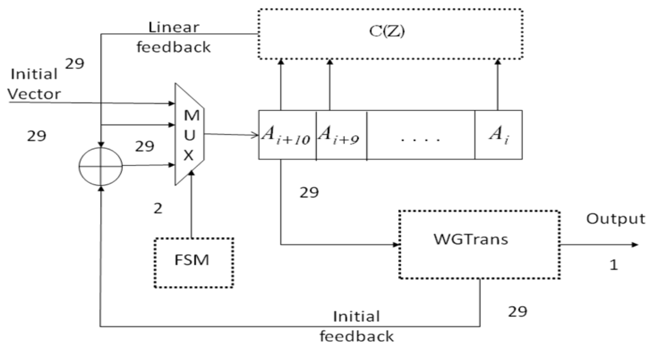


Fig -4: Data path of a basic cell of the VLM3 algorithm [4].

The LFSR feedback polynomial $C(Z) = Z^{11} \oplus Z^{10} \oplus Z^9 \oplus Z^6 \oplus Z^3 \oplus Z \oplus \beta$ is a primitive polynomial of degree 11 over GF(2^{29}). The output of the LFSR at A_{i+10} is filtered by an orthogonal 29-bit WG transformation (GF (2^{29}) $\longrightarrow$ GF (2)) given by:

$$\text{WGTrans} = \text{Tr}\,(\text{WGPerm}(A_{i+10} \oplus 1)) \qquad (5)$$

where

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

$$= (1 \oplus X \oplus X^{2^k+1} \oplus X^{2^{2k}+(2^k+1)} \oplus X^{2^k(2^k-1)+1} \oplus X^{2^{2k}+(2^k-1)}) \qquad (6)$$

is the WG permutation, $r1=2^k+1$, $r=2^{2k}+2^k+1$, $r3=2^{2k}-2^k+1$, $r4=2^{2k}+2k-1$. This results in a binary key-stream. The cipher operates in three phases. They are: loading phase, key initialization phase and running phase. The reader is referred to [13], [1], [14] for more details.

The key bits and IV bits are loaded into the LFSR, to initialize the cipher. The state of the LFSR is represented as S(1), S(2), S(3), ..., S(11) $\in F_{29}$. Each stage S(i) $\in F_{29}$, as $S_{1,..,29}$ (i) where 1 ≤ 11 Similarly key bits are represented as $K_{1,..,j}$, $1 \le j \le 128$ and IV bits as $IV_{1,..,m}$, $1 \le 128 \le m$.128 bits key and IV are loaded as shown below [1]:

$S_{1,..16}(1) = k_{1,..,16}$	$S_{17,..24}(1) = IV_{1,..,8}$	$S_{1,..8}(2) = k_{17,..,24}$
$S_{9,..24}(2) = IV_{9,..,24}$	$S_{1,..16}(3) = k_{25,..,40}$	$S_{17,..24}(3) = IV_{25,..,32}$
$S_{1,..8}(4) = k_{41,..,48}$	$S_{9,..24}(4) = IV_{33,..,48}$	$S_{1,..16}(5) = k_{49,..,64}$
$S_{17,..24}(5) = IV_{49,..,56}$	$S_{1,..8}(6) = k_{65,..,72}$	$S_{9,..24}(6) = IV_{57,..,72}$
$S_{1,..16}(7) = k_{73,..,88}$	$S_{17,..24}(7) = IV_{73,..,80}$	$S_{1,..8}(8) = k_{89,..,96}$
$S_{9,..24}(8) = IV_{81,..,96}$	$S_{1,..16}(9) = k_{97,..,112}$	$S_{17,..24}(9) = IV_{97,..,104}$
$S_{1,..8}(10) = k_{113,..,120}$	$S_{9,..24}(10) = IV_{105,..,120}$	$S_{1,..8}(11) = k_{121,..,128}$
$S_{17,..24}(11) = IV_{121,..,128}$		

The proposed WG transformation circuit using VLM3 multiplier is shown in Fig. 5. It includes a Serialized Key Initialization Module, SKIM [4], which schedules the appropriate inputs to the field multiplier in each stage of the sequential computation through some multiplexers.

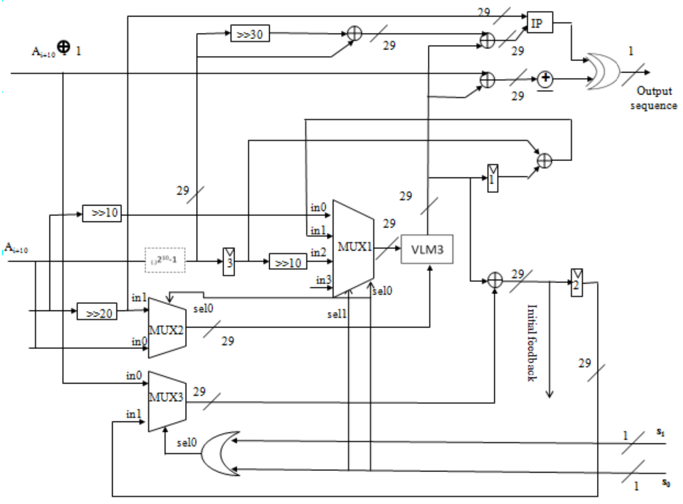


Fig -5: Proposed WG transformation using VLM3 multiplier

Block named IP utilizes two 29-bit inputs to generate the inner product. The inner product of two n-bit vectors, A = (a_0,..., a_{n-1}) and B = (b_0,..., b_{n-1}), is computed as

$$A\,.\,B = \sum_{i=0}^{n-1} a_i b_i \in \{0,1\} \qquad (7)$$

$\oplus$ adds the 29-bits at its input over GF (2).

When two 29 bit numbers are multiplied, the actual result will be 58 bits. Here a field operation is used and the output is 28 bits itself. First, a field array multiplier is placed which takes the reminder, when the product is divided by 2^{29}. It can be obtained by taking the 29 LSB bits of the 58 bit product. The same operation can be achieved using VLM3 multiplier, whose area is lesser than that of a field array multiplier. The modification mainly focuses on the design of the WG transformation using VLM3 multiplier as shown in Fig. 5. In VLM3 Algorithm, the modulus must be a prime number. So the prime number next to 2^{29} is taken as the modulus. The result will be in the Montgomery domain and it is also limited to 29 bit.

4. VERIFICATION

In order to verify the system, a transmitter and a receiver is designed and simulated. In the transmitter section there is a PISO which converts the plain text into sequence of bits. WG generator takes IV and KEY as inputs and generates keystream. These bits are XORed with the PISO output to get the cipher text. The block diagram for the transmitter is shown in Fig. 6.

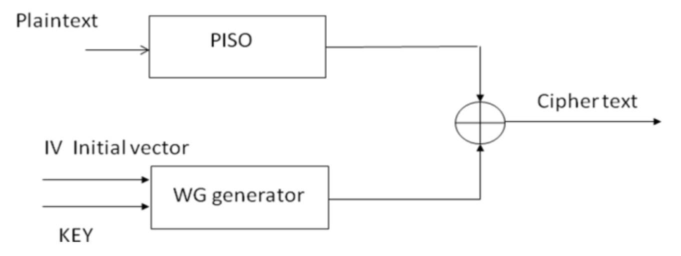


Fig -6: Block diagram for the transmitter

At the receiver side, the same WG generator used in the transmitter is used to generate the keystream. The bit-by-bit XORing of this keystream bit with the cipher text results in a series of data. This is passed into a SIPO to recover the plain text. The block diagram for the Receiver is shown in Fig. 7.

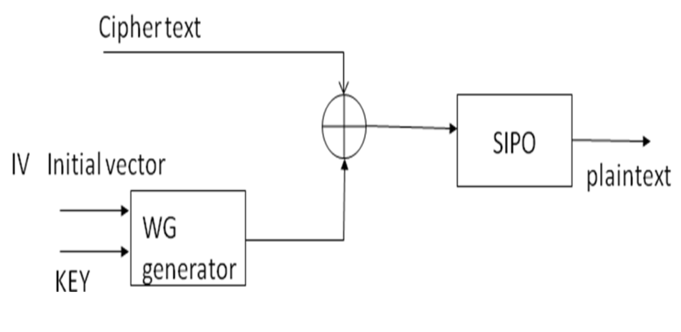


Fig -7: Block diagram for the receiver

5.RESULTS AND DISCUSSION

RTL schematic and simulation results of a 29 bit VLM3 multiplier is shown below. WG generator and transceiver with both VLM3 and field array multiplier are also given. Coding is done using Verilog Hardware Description Language (HDL). The simulation was done in Xilinx ISE Vivado Design Suite 14.2.

5.1 VLM3 MULTIPLIER

The functional simulation result of 29 bit VLM3 architecture is shown in Fig. 8. X and Y are the multiplier and multiplicand respectively. M is the modulus. Res_rdy is a signal which indicates the output is ready and S is the output, whose value is X . Y. R mod M. Where, R = 2^{-n} mod M. In order to get the actual result, an extra Montgomery multiplication by the constant 2^n mod M is done.

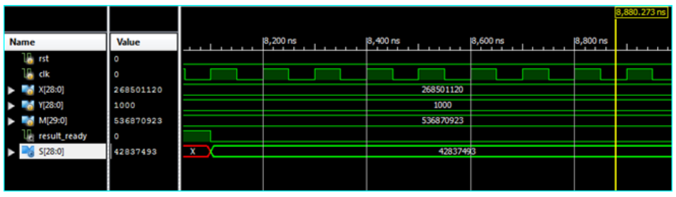


Fig -7: Simulation result of 29 bit VLM3 architecture

5.2 WG GENERATOR

Functional simulation result of WG generator with field array multiplier is shown in Fig. 8. IV and KEY are the 128 bit inputs. OUT is the one bit output sequence with randomness properties.

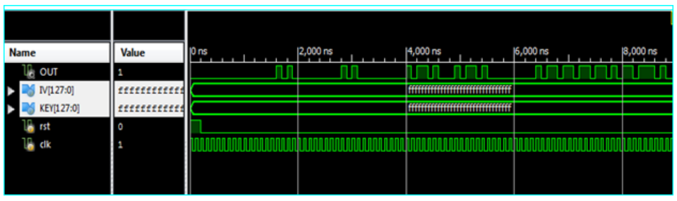


Fig -8: Simulation simulation result of WG generator with field array multiplier

Simulation result of WG generator with VLM3 multiplier is shown in Fig. 9. Area is increased when WG generator is implemented with VLM3 multiplier. Power consumption also shows a slight increase.

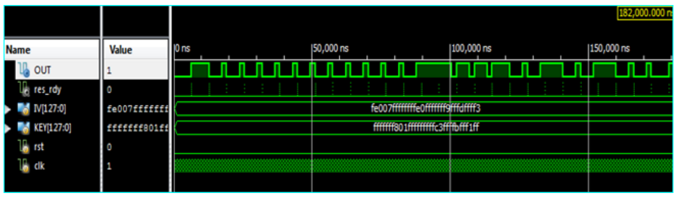
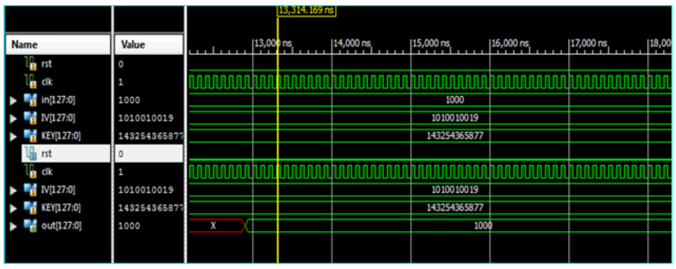


Fig -9: Simulation simulation result of WG generator with VLM3 multiplier

5.3 VERIFICATION

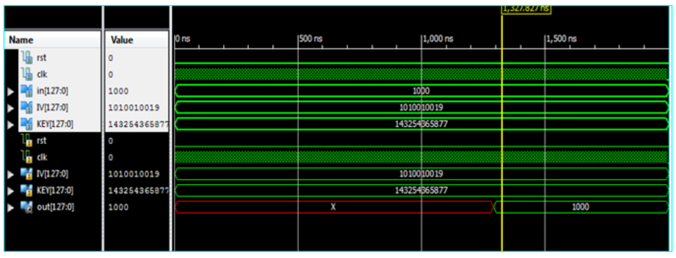
To verify the working, a transmitter and a receiver is designed and the simulation results are given below. Fig. 10 shows the simulation result of the transceiver whose WG generator uses a field array multiplier.



Name	Value
rst	0
clk	1
in[127:0]	1000
IV[127:0]	1010010019
KEY[127:0]	143254365877
rst	0
clk	1
IV[127:0]	1010010019
KEY[127:0]	143254365877
out[127:0]	1000

Fig -10: Simulation result of transceiver with field array multiplier

A message signal, "in" is given as input to a PISO of the transmitter. Same KEY and IV inputs are given to the WG transformation block of both the transmitter and receiver. "out" signal is the output of the SIPO. The same message signal is obtained at the output. Fig. 11 shows the simulation result of the transceiver whose WG generator uses a VLM3 multiplier.



Name	Value
rst	0
clk	0
in[127:0]	1000
IV[127:0]	1010010019
KEY[127:0]	143254365877
rst	0
clk	0
IV[127:0]	1010010019
KEY[127:0]	143254365877
out[127:0]	1000

Fig -11: Simulation result of transceiver with VLM3 multiplier

6. PERFORMANCE ANALYSIS

Performance analysis focuses on the multiplier being used in the WG transformation. The design summary of both field array multiplier and VLM3 multiplier when synthesized in Virtex5 family, device XC5VLX20T, package FF323 is given in table 4.

Table -4: Device, Power and Delay Utilization Summary of Field array multiplier and VLM3 multiplier

Logic utilization of multipliers (synthesized using Virtex5)	Field array multiplier (29 BIT)	VLM3 multiplier (29 BIT)
No. of slice registers	88	164
No. of slice LUTs	1400	544
No. of fully used LUT FF	73	51
No. of bonded IOB	89	145
Total	1650	904
Power(mW)	343	325
Delay(ns)	2.826	2.872

Total device utilization is obtained from the number of slice registers, slice LUTs, LUT FFs, and number of bonded IOBs. Power and delay is also compared. Even though total device utilization and power is reduced, delay is slightly increased in VLM3 multiplier.

Graphical representation of the design parameters of both VLM3 and field array multipliers are shown in chart 1 and chart 2.

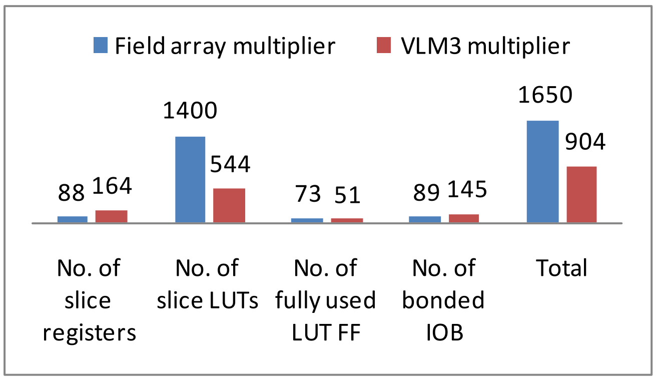


Chart -1: Device Utilization Graph of the multipliers

When device utilization of both multipliers are compared it can be seen that total device utilization is less for VLM3 multiplier and it shows a hardware efficiency of 45%. power reduction of 5.25% is also obtained.

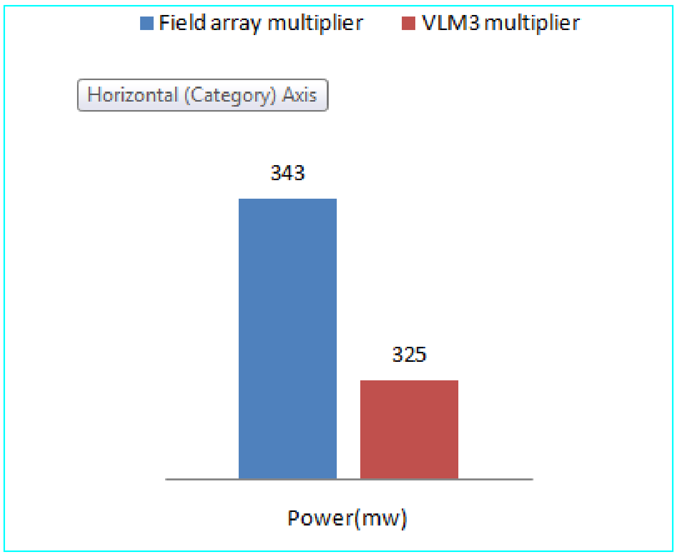


Chart -3: Power analysis of the multipliers

7. CONCLUSION

Variable Length Montgomery Modular Multiplication Algorithm for modular multiplication is studied in this project. 8 bit VLM3 architecture is realized in Spartan 3E FPGA. Device utilization, power (mW) and delay (ms) of 29 bit field array multiplier and VLM3 multiplier is compared.

Among these, VLM3 multiplier shows a hardware efficiency of 45% and total power reduction of 5.25% with a slight increase in delay.

WG transformation block in the WG generator is designed using both field array multiplier and VLM3 multiplier. To verify the working of the WG stream cipher, a transmitter and receiver is simulated and synthesized in Virtex 5.

REFERENCES

[1] G. Gong and Y. Nawaz. (2005). "The WG Stream Cipher" [Online]. Available:http://www.ecrypt.eu.org/stream/wgp2.html

[2] (2005). eSTREAM "The ECRYPT Stream Cipher Project " [Online]. Available: http://www.ecrypt.eu.org/stream/

[3] S. Golomb, G. Gong, H.-K. Lee, and P. Gaal, "New binary pseudo-random sequences of period 2n − 1 with ideal autocorrelation," IEEE Trans. Inf. Theory, vol. 44, no. 2, pp. 814–817, Mar. 1998.

[4] H. E. Razouk, A. R. Masoleh, "New Implementations of the WG Stream Cipher," IEEE Trans. Very Large Scale Integr, Vol. 22, No.9, 1865-1868, Sep. 2014.

[5] F. Gandino, F. Lamberti, G. Paravati, J. Bajard, and P. Montuschi, "An algorithmic and architectural study on Montgomery exponentiation in RNS," IEEE Trans. Comput, Vol. 61, No.8, 1071-1083, Aug. 2012.

[6] M. D. Shieh, J. H. Chen, H. H. Wu, and W. C. Lin, "A new modular exponentiation architecture for efficient design of RSA cryptosystem," IEEE Trans. Very Large Scale Integr, Vol. 16, No.9, 1151-1161, Sep. 2008.

[7] N. Nedjah, L. M. Mourelle, M. Santana, and S. Raposo, "Massively parallel modular exponentiation method and its implementation in software and hardware for high-performance cryptographic systems," IET Comput. Digit. Techn., vol. 6, no. 5, pp. 290-301, Sep. 2012.

[8] P. L. Montgomery, "Modular Multiplication without Trial Division," Math. Comput., vol. 44, no. 170, pp. 519-521, 1985.

[9] M. Moayedi and A. Rezai "Design and Evaluation of Novel Effective Montgomery Modular Multiplication Architecture," International Journal of Security and its Applications, 10, 261-270, 2016.

[10] A.Rezai, and P. Keshavarzi , " High-Throughput Modular Multiplication and Exponentiation Algorithms Using Multibit-Scan–Multibit-Shift Technique," IEEE Trans. Very Large Scale Integr, Vol. 23, No.9, 1710-1719, Sep. 2012.

[11] G. A. Ruiz and M. Granda," Efficient Canonic Signed Digit Recoding,"Microelectronics Journal, 9, 1090-1097, 2011.

[12] T.Kowsalya and Dr. S. Palaniswami,"Mathematical Analysis of Modular Multiplication for Key Exchange in Wireless Network, International Journal of Advanced Engineering Technology", 2, 1145-1148, 2016.

[13] Y. Nawaz and G. Gong, "WG: A family of stream ciphers with designed randomness properties," *Inf. Sci.*, vol. 178, no. 7, pp. 1903–1916, 2008.

[14] C. Lam, M. Aaagaard and G. Gong "Hardware Implementations of Multi-output Welch-Gong Ciphers," Dept. Department of Electr. Comput. Eng., Univ. Waterloo, Waterloo, ON, Canada, Tech. Rep. CACR 2011-01, 2009 [Online]. Available: http://cacr.uwaterloo.ca/techreports/2011/cacr2011-01.pdf, 2009.