Efficient adaptation of the Karatsuba algorithm for implementing on FPGA very large scale multipliers for cryptographic algorithms

Walder Andre
Department of Electrical and Computer Engineering, Royal Military College of Canada, Canada

Article Info

Article history:

Received Jul 11, 2020
Revised Sep 28, 2020
Accepted Oct 8, 2020

Keywords:

128-bit × 128-bit multiplier
512-bit × 512-bit multiplier
Cryptographic key
FPGA
Karatsuba multiplier

ABSTRACT

Here, we present a modified version of the Karatsuba algorithm to facilitate the FPGA-based implementation of three signed multipliers: 32-bit × 32-bit, 128-bit x 128-bit, and 512-bit × 512-bit. We also implement the conventional 32-bit × 32-bit multiplier for comparative purposes. The Karatsuba algorithm is preferable for multiplications with very large operands such as 64-bit × 64-bit, 128-bit × 128-bit, 256-bit × 256-bit, 512-bit × 512-bit multipliers and up. Experimental results show that the Karatsuba multiplier uses less hardware in the FPGA compared to the conventional multiplier. The Xilinx xc7k325tfbg900 FPGA using the Genesis 2 development board is used to implement the proposed scheme. The results obtained are promising for applications that require rapid implementation and reconfiguration of cryptographic algorithms.

Corresponding Author:

Walder Andre,
Department of Electrical and Computer Engineering,
Royal Military College,
Box 17000, Station Forces,
Kingston, Ontario, K7K 7B4, Canada.
Email: walder.andre@polymtl.ca

1. INTRODUCTION

The need to protect data and information is crucial; it can make the difference between life and death. More particularly, in the military field, winning a war relies heavily on the protection of information [1]. The use of encryption keys is one of the means used to preserve the authenticity, confidentiality, non-denial, and integrity of the data. Encrypted messages use cryptographic keys, which are a binary number ranging from 0 to n. Figure 1 below shows an example of a block diagram to encrypt a message.

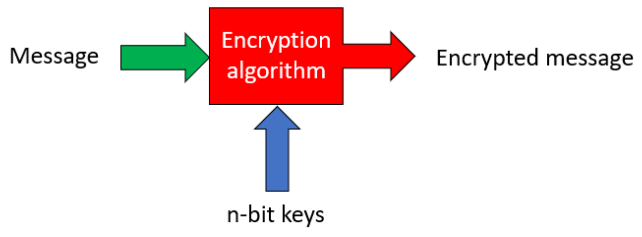


Figure 1. Message encryption process

The longer the cryptographic key, the more robust its decryption. The length of the key depends on the type of information protection desired to achieve. Therefore, the nature of the mission and the operation heavily influence the length of a key. And finally, it depends on the severity of the damage that could occur if the information is intercepted and decrypted. Most of the cryptographic algorithms are very difficult to decipher; the theoretical foundations are substantial [2]. As with any encryption algorithm, we perform a lot of arithmetical operations, and we need to find methods to accelerate these basic arithmetic operations. These methods are geared towards the multiplication of large numbers. Keys are ranging in length from 64 bits to 4096 bits depending on the security level we want to achieve and the type of the key generator used to generate them. As we said before, the longer the cryptographic keys, the stronger the cryptographic algorithm will be. For instance, we have the algorithms AES-128, DH, DSA, RSA-3072, SHA-256, and ECDH, ECDSA-256 present a security level of 128 bis. The algorithms AES-192, SHA-384, ECDH, ECDSA-384 provide a security level of 192 bits, and finally, the algorithms AES-256, SHA-512, ECDH, and ECDSA-521 exhibit a security level of 256 bits [3].

Harika et *al*. have presented a critical review of four multiplication algorithms, which are shift-And-Add Multiplier, Carry Save Adder, Booth Multiplier, and a modified version of the Booth multiplier. Based on this article, the Carry Save Adder was found to be more efficient in terms of execution time and less space in the FPGA than the other multiplication algorithms mentioned above. Different multiplication algorithms exist, such as Grid, Wallace-tree, Vedic, Lattice, Combinational, Sequential, Array and Montgomery, and Karatsuba [4, 5]. Several articles proposed implementation methods on FPGA of the Karatsuba algorithm. Yang has introduced a scheme for implementing a 256-bit x 256-bit multiplier, which exhibits 50% efficiency compared to traditional implementations [6].

In this article, a new scheme for implementing the Karatsuba multiplier. The Karatsuba multiplier is very efficient in multiplying very large numbers, which constitutes an excellent asset in achieving complex cryptographic processors [7-9]. The conventional multiplication method has a complexity O(N^2), while Karatsuba has a complexity of O($N^{\log 3/\log 2}$). The following section will present the theoretical foundations for the Karatsuba algorithm and used the finding to implement a third-degree Karatsuba multiplier. Section 3 will introduce the proposed scheme; section 4 will show the experimental results.

2. THIRD-DEGREE KARATSUBA ANALYSIS

Here, we present the theoretical foundation for developing a third-degree Karatsuba multiplier formula. We will be using it to implement 32-bit × 32-bit Karatsuba multiplier, 128-bit × 128-bit Karatsuba multiplier, and 512-bit × 512-bit Karatsuba multiplier into FPGA. Weimerskirch laid out a more in-depth examination of the Karatsuba algorithm [10]. Let A(x) and B(x) the two operands of the third-degree Karatsuba multipliers.

$$A(x) = a_3x^3 + a_2x^2 + a_1x^1 + a_0x^0 \tag{1}$$

$$B(x) = b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0 \tag{2}$$

$$C(x) = \mathrm{A}(x)B(x)\ (2) \tag{3}$$

$$C_k = \sum_{i+j=k}^{\infty} a_ib_jx^{i+j} \tag{4}$$

$$C(x) = (a_0b_0)x^0 + (a_0b_1 + a_1b_0)x^1 + (a_0b_2 + a_2b_0 + a_1b_1)x^2 + (a_0b_3 + a_3b_0 + a_1b_2 + a_2b_1)x^3 + (a_1b_3 + a_3b_1 + a_2b_2)x^4 + (a_2b_3 + a_3b_2)x^5 + (a_3b_3)x^6 \tag{5}$$

Let

$$M_i = A_iB_i \tag{6}$$

$$M_{i,j} = \left(A_i + A_j\right)\left(B_i + B_j\right) \tag{7}$$

With *i*, *j* = 0, 1, 2, 3.

By applying some basic algebra to (5), it follows

$$C(x) = (a_0b_0)x^0 + \left(a_0b_1 + a_1b_0 + a_0b_0 + a_1b_1 - (a_0b_0 + a_1b_1)\right)x^1 + (a_0b_2 + a_2b_0 +$$

$$a_0b_0 + a_2b_2 - (a_0b_0 + a_2b_2) + a_1b_1)x^2 + \big(a_0b_3 + a_3b_0 + a_0b_0 + a_3b_3 - (a_0b_0 + a_3b_3) + a_1b_2 + a_2b_1 + a_1b_1 + a_2b_2 - (a_1b_1 + a_2b_2)\big)x^3 + (a_1b_3 + a_3b_1 + a_1b_1 + a_3b_3 - (a_1b_1 + a_3b_3) + a_2b_2)x^4 + \big(a_2b_3 + a_3b_2 + a_2b_2 + a_3b_3 - (a_2b_2 + a_3b_3)\big)x^5 + (a_3b_3)x^6 \quad (8)$$

After equaling (5) and (8), and by identification it follows:

$$a_0b_1 + a_1b_0 = a_0b_1 + a_1b_0 + a_0b_0 + a_1b_1 - (a_0b_0 + a_1b_1) \quad (9)$$

After applying (6) and (7) to the RHS of (9), it follows:

$$a_0b_1 + a_1b_0 = M_{0,1} - (M_0 + M_1) \quad (10)$$

For the coefficient for x^2, it follows:

$$a_0b_2 + a_2b_0 + a_1b_1 = a_0b_2 + a_2b_0 + a_0b_0 + a_2b_2 - (a_0b_0 + a_2b_2) + a_1b_1 \quad (11)$$

Applying (6) and (7) to the RHS of (11) to have:

$$a_0b_2 + a_2b_0 + a_1b_1 = M_{0,2} - (M_0 + M_2) + M_1 \quad (12)$$

For the coefficient for x^3, it follows:

$$a_0b_3 + a_3b_0 + a_1b_2 + a_2b_1 = a_0b_3 + a_3b_0 + a_0b_0 + a_3b_3 - (a_0b_0 + a_3b_3) + a_1b_2 + a_2b_1 + a_1b_1 + a_2b_2 - (a_1b_1 + a_2b_2) \quad (13)$$

Applying (6) and (7) to RHS of (13) to have:

$$a_0b_3 + a_3b_0 + a_1b_2 + a_2b_1 = M_{0,3} + M_{1,2} - (M_0 + M_3) - (M_1 + M_2) \quad (14)$$

For the coefficient for x^4, it follows:

$$a_1b_3 + a_3b_1 + a_2b_2 = a_1b_3 + a_3b_1 + a_1b_1 + a_3b_3 - (a_1b_1 + a_3b_3) + a_2b_2 \quad (15)$$

Applying (6) and (7) to (13) RHS to have:

$$a_1b_3 + a_3b_1 + a_2b_2 = M_{1,3} - (M_1 + M_3) + M_2 \quad (16)$$

And finally, for the coefficient for x^5, it follows:

$$a_2b_3 + a_3b_2 = a_2b_3 + a_3b_2 + a_2b_2 + a_3b_3 - (a_2b_2 + a_3b_3) \quad (17)$$

After applying (6) and (7) to (17) RHS, it follows:

$$a_2b_3 + a_3b_2 = M_{2,3} - (M_2 + M_3) \quad (18)$$

Replacing (10), (12), (14), (16), and (18) by their values in (8) yields into

$$C(x) = M_0 + \big(M_{0,1} - M_0 - M_1\big)x^1 + \big(M_{0,2} - M_0 - M_2 + M_1\big)x^2 + \big(M_{0,3} + M_{1,2} - M_0 - M_3 - M_1 - M_2\big)x^3 + \big(M_{1,3} - M_1 - M_3 + M_2\big)x^4 + \big(M_{2,3} - M_2 - M_3\big)x^5 + M_3x^6 \quad (19)$$

We have presented a new scheme to implement (19). Figures 2 and 3 present the first two steps in calculating the M_i and $M_{i,\,j}$ with i, j = 0, 1, 2, 3 to implement the Karatsuba algorithm. Once these two steps pass, what follows is the use of adders and shift registers to implement the rest of the equation. Figure 2 presents the separation of the operand A(x), which is of length N into four subgroups to give the a_0, a_1, a_2, and a_3. The same step is repeated on the B(x) operand to achieve b_0, b_1, b_2, and b_3. Figure 3 shows the generation of the variables M_n and $M_{n,\,m}$.

Below, we present our modified version of the Karatsuba algorithm.

```
/* Let A and B two binary numbers of size nwidth */
min_stdvec = 8

procedure karatsuba(A, B)
   if (A < limit) or (B < limit)
      return A × B

       $a_0$ = A((nwidth/min_stdvec)-1 downto 0)
       $a_1$ = A((nwidth/min_stdvec)*2-1 downto (nwidth/min_stdvec))
       $a_2$ = A((nwidth/min_stdvec)*3-1 downto (nwidth/min_stdvec)*2)
       $a_3$ = A((nwidth/min_stdvec)*4-1 downto (nwidth/min_stdvec)*3)

       $b_0$ = B((nwidth/min_stdvec)-1 downto 0)
       $b_1$ = B((nwidth/min_stdvec)*2-1 downto (nwidth/min_stdvec))
       $b_2$ = B((nwidth/min_stdvec)*3-1 downto (nwidth/min_stdvec)*2)
       $b_3$ = B((nwidth/min_stdvec)*4-1 downto (nwidth/min_stdvec)*3)

       /* 4 KA calls to compute $M_0$, $M_1$, $M_2$, $M_3$; $M_{1,2}$, $M_{1,3}$ and $M_{2,3}$ */
       karatsuba($a_0$ , $b_0$, $M_0$)
       karatsuba($a_1$ , $b_1$, $M_1$)
       karatsuba($a_2$ , $b_2$, $M_2$)
       karatsuba($a_3$ , $b_3$, $M_3$)

       $a'_{0,1} = a_0 + a1$
       $a'_{0,2} = a_0 + a_2$
       $a'_{0,3} = a_0 + a_3$
       $a'_{1,2} = a_1 + a_2$
       $a'_{1,3} = a_1 + a_3$
       $a'_{2,3} = a_2 + a_3$

       $b'_{0,1} = b_0 + b_1$
       $b'_{0,2} = b_0 + b_2$
       $b'_{0,3} = b_0 + b_3$
       $b'_{1,2} = b_1 + b_2$
       $b'_{1,3} = b_1 + b_3$
       $b'_{2,3} = b_2 + b_3$

       /* 6 KA calls to compute */
       karatsuba($a'_{0,1}$, $b'_{0,1}$, $M_{0,1}$)
       karatsuba($a'_{0,2}$, $b'_{0,2}$, $M_{0,2}$)
       karatsuba($a'_{0,3}$, $b'_{0,3}$, $M_{0,3}$)
       karatsuba($a'_{1,2}$, $b'_{1,2}$, $M_{1,2}$)
       karatsuba($a'_{1,3}$, $b'_{1,3}$, $M_{1,3}$)
       karatsuba($a'_{2,3}$, $b'_{2,3}$, $M_{2,3}$)

       $term_7 = M_3$                                              $x_6$
       $term_6 = M_{2,3} – M_2 – M_3$                              $x_5$
       $term_5 = M_{1,3} – M_1 – M_3 + M_2$                        $x_4$
       $term_4 = M_{0,3} – M_0 – M_3 + M_{1,2} – M_1 – M_2$        $x_3$
       $term_3 = M_{0,2} – M_0 – M_2 + M_1$                        $x_2$
       $term_2 = M_{0,1} – M_0 – M_1$                              $x_1$
       $term_1 = M_0$                                              $x_0$

      result = term7_slr + term6_slr + term5_slr + term3_slr + term4_slr + term2_slr + term1_slr

end
```

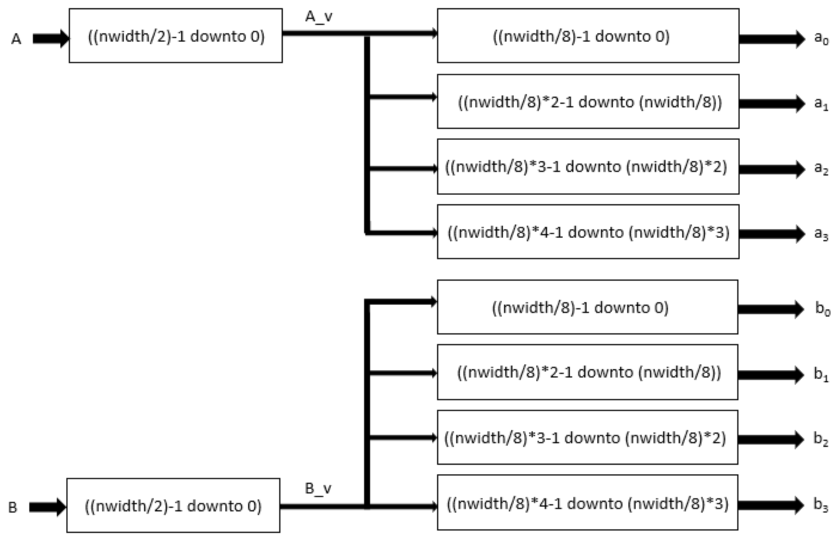


Figure 2. Generic n-bit × n-bit third-degree Karatsuba input preprocessing

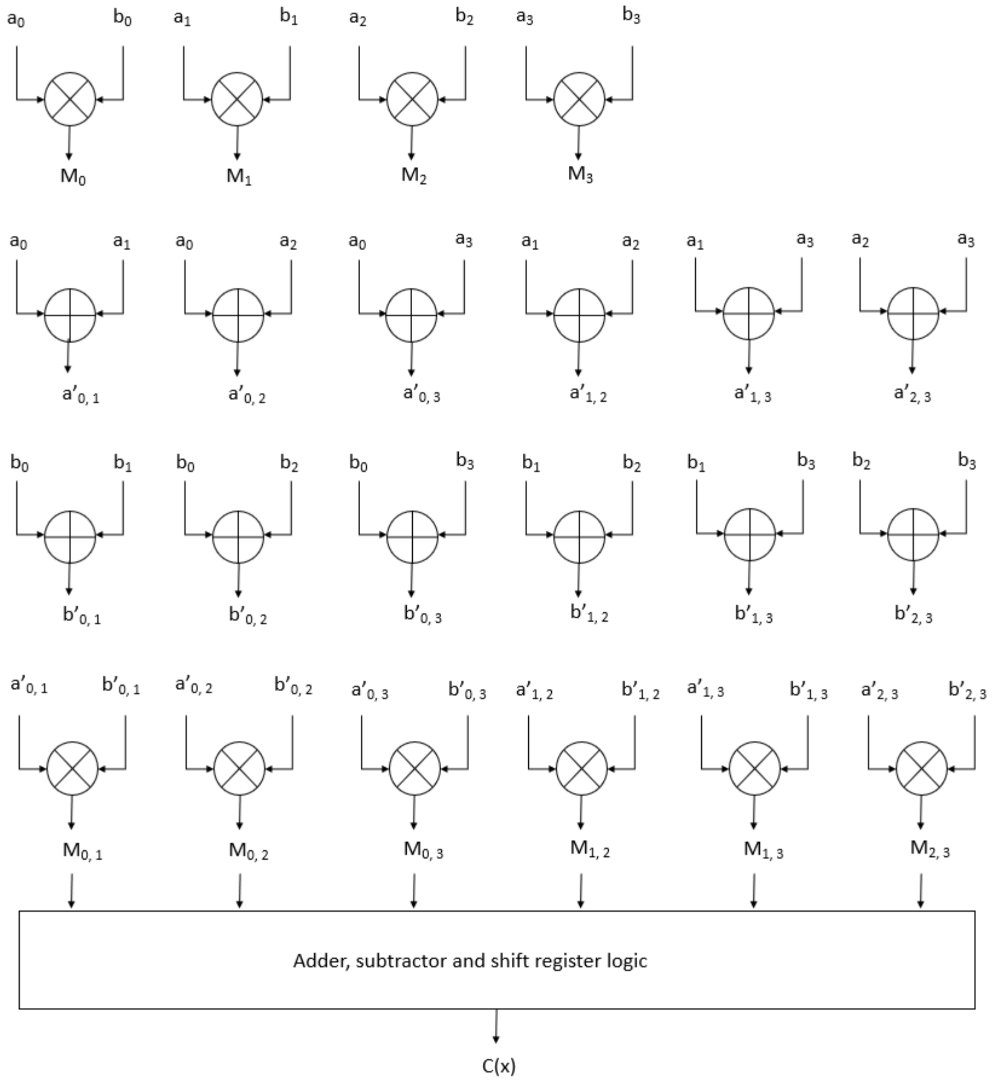

a_0 b_0 M_0
a_1 b_1 M_1
a_2 b_2 M_2
a_3 b_3 M_3
$a'_{0,1}$ $a'_{0,2}$ $a'_{0,3}$ $a'_{1,2}$ $a'_{1,3}$ $a'_{2,3}$
$b'_{0,1}$ $b'_{0,2}$ $b'_{0,3}$ $b'_{1,2}$ $b'_{1,3}$ $b'_{2,3}$
$M_{0,1}$ $M_{0,2}$ $M_{0,3}$ $M_{1,2}$ $M_{1,3}$ $M_{2,3}$
Adder, subtractor and shift register logic
C(x)

Figure 3. C(X) generation for a 32-bit × 32-bit Karatsuba

3. RESULTS AND ANALYSIS

Section 3.1 and 3.2 show the simulation and implementation results for 32-bit x 32-bit, 128-bit x 128-bit, and 512-bit x 512-bit multipliers.

3.1. Simulation results

For the sake of visibility, we present a shortened part of the simulations in Figure 4, Figure 5, and Figure 6. The results for the three implemented multipliers are consistent and give the expected values.

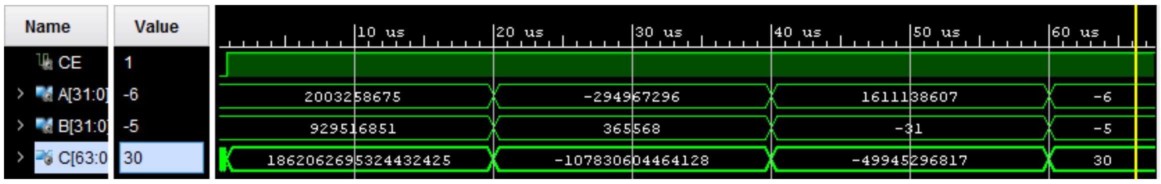


Figure 4. Karatsuba multiplier 32-bit × 32-bit simulation results

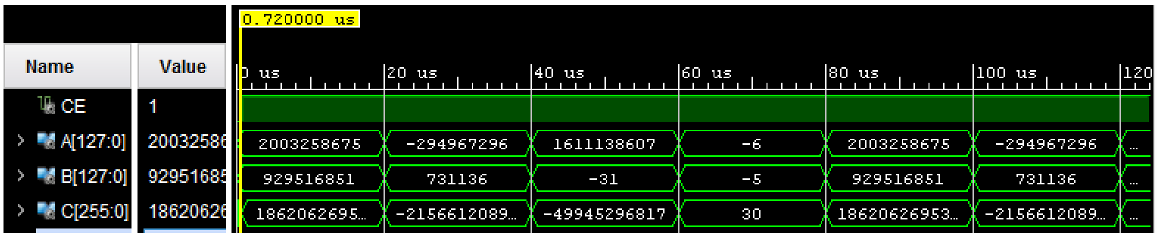


Figure 5. Karatsuba multiplier 128-bit × 128-bit simulation results

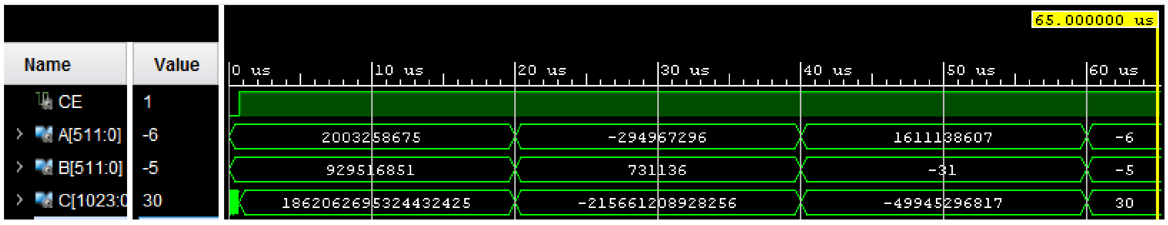
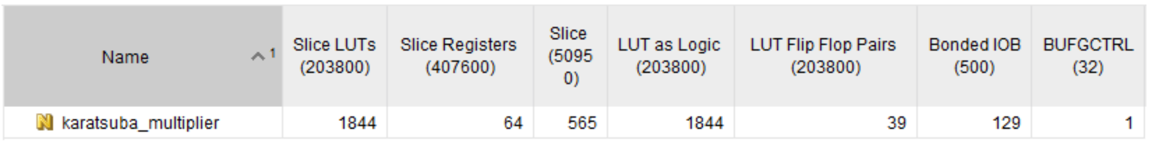


Figure 6. Karatsuba multiplier 512-bit × 512-bit simulation results

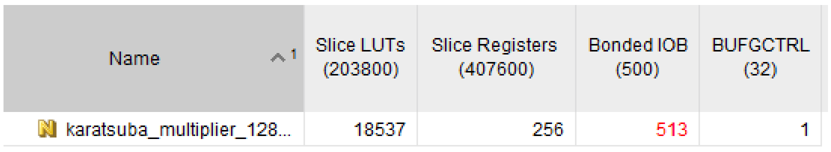
3.2. Implementation results

As shown in Figure 7, the implementation on FPGA of the 32-bit × 32-bit does not display any error. Of the 500 limited IO capability, only 64 are bounded, and 1844/203800 slice LUTs are used. Regarding 128-bit x 128-bit. By quadrupling the multiplier size, we multiply by a factor of 10 the size of the slice LUTs used, as depicted in Figure 8. The implementation of the 128-bit x 128-bit multiplier exceeded the IO capabilities of the FPGA, as shown in Figure 8. And it got worse with the implementation of the 512-bit x 512-bit multiplier, as shown in Figure 9. This result is not a surprise and does not depends on the scheme but rather the capacity of the FPGA used.



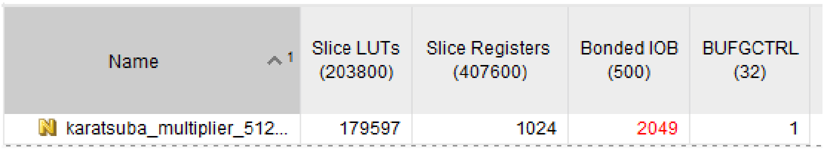
Name	Slice LUTs (203800)	Slice Registers (407600)	Slice (50950)	LUT as Logic (203800)	LUT Flip Flop Pairs (203800)	Bonded IOB (500)	BUFGCTRL (32)
karatsuba_multiplier	1844	64	565	1844	39	129	1

Figure 7. Karatsuba multiplier 32-bit × 32-bit implementation results



Name	Slice LUTs (203800)	Slice Registers (407600)	Bonded IOB (500)	BUFGCTRL (32)
karatsuba_multiplier_128...	18537	256	513	1

Figure 8. Karatsuba multiplier 128-bit × 128-bit implementation results



Name	Slice LUTs (203800)	Slice Registers (407600)	Bonded IOB (500)	BUFGCTRL (32)
karatsuba_multiplier_512...	179597	1024	2049	1

Figure 9. Karatsuba multiplier 512-bit × 512-bit implementation results

4. CONCLUSION

In this paper, we have proposed a modified version of the Karatsuba algorithm as well as a new scheme to facilitate FPGA implementation. Results obtained from 32-bit x 32-bit Karatsuba multiplier, 128-bit x 128-bit Karatsuba multiplier, and 512-bit x 512-bit Karatsuba multiplier have met the expectation. They are promising for applications that require the rapid implementation and reconfiguration of cryptographic algorithms. The next step is to use these multipliers to implement a complete cryptographic algorithm on FPGA.

ACKNOWLEDGEMENTS

The author would like to thank CMC Microsystems for providing design tools, materials, and support.

REFERENCES

[1] W. Andre and O. Couillard, "Design and implementation of a new architecture of a real-time reconfigurable digital modulator (DM) into QPSK,8-PSK, and 16-PSK on FPGA," *International Journal of Reconfigurable and Embedded Systems*, vol. 7, no. 3, pp. 173-185, 2018.

[2] K. Kwangjo, "Cryptography: A new open access journal," *Cryptography*, vol. 1, no. 1, pp. 1-4, 2016. [Online]. Available: https://pdfs.semanticscholar.org/d78d/59fc08de0b1b2f43249cc0a586a8febf5e6d.pdf.

[3] Cisco Security, "Next Generation Cryptography," Tools Cisco, 2015. [Online]. Available: https://tools.cisco.com/security/center/resources/next_generation_cryptography.

[4] K. Harika,. *et al*, "Analysis of different multiplication algorithms & FPGA implementation," *IOSR Journal of VLSI and Signal Processing* (IOSR-JVSP) vol. 4, no. 2, pp. 29-35, 2014.

[5] P. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[6] D. Yang et *al*., "Efficient FPGA implementation of modular multiplication based on Montgomery algorithm," *Microprocessors and Microsystems*, vol. 47, pp. 209-215, 2016.

[7] Joachim von zur Gathen and Jamshid Shokrollahi, "Efficient FPGA-based Karatsuba multipliers for polynomials over F_2," *International Workshop on Selected Areas in Cryptography* SAC 2005: *Selected Areas in Cryptography* pp 359-369, 2005.

[8] D. Suzuki, "How to maximize the potential of FPGA resources for modular exponentiation," *Cryptographic Hardware and Embedded Systems-CHES* 2007, pp. 272–288, 2007.

[9] J. Samanta,. *et al*, "Modified Karatsuba multiplier for key equation solver in RS Code," *in Radioelectronics and Communications Systems*, vol. 58, no. 10, pp. 452-461, 2015.

[10] A. Weimerskirch and C. Paar, "Generalizations of the Karatsuba algorithm for efficient implementations," *IACR Cryptol. ePrint Arch*, pp. 1-17, 2006. [Online]. Available: http://eprint.iacr.org/2006/224.pdf.