Indonesian Journal of Electrical Engineering and Computer Science
Vol. 21, No. 2, February 2021, pp. 735~743
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v21.i2.pp735-743

A compact FPGA-based montgomery modular multiplier

Ahmed A. H. Abd-Elkader[1], Mostafa Rashdan[2], El-Sayed A. M. Hasaneen[3], Hesham F. A. Hamed[4]
[1]Qusier Telecom, Telecom Egypt, Red Sea, Egypt
[2]Faculty of Energy Engineering, Aswan University, Aswan, Egypt
[3]Faculty of Engineering, Aswan University, Aswan, Egypt
[4]Faculty of Engineering, Minia University, Minia, Egypt, and Faculty of Engineering, Egyptian - Russian University, Cairo, Egypt

Article Info

Article history:

Received Jun 15, 2020
Revised Aug 11, 2020
Accepted Aug 21, 2020

Keywords:

FPGA
Lightweight cryptography
LUT
Modular multiplier
Virtex-6

ABSTRACT

This paper presents a compact design of Montgomery modular multiplier (MMM). MMM serves as a building block commonly required in security protocols relying on public key encryption. The proposed design is intended for hardware applications of lightweight cryptographic modules that is utilized for the system on chip (SoC) and internet of things (IoT) devices. The proposed design is an enhancement of the original MMM without any multiplication or subtraction processes. The main target of the new modification is enhancing the performance and reducing the area of the MMM hardware module. The operands and internal variables of the proposed hardware circuit is optimized to be bounded to the smallest efficient size to minimize the area and the critical path delay. The proposed design was coded in VHDL, implemented on the Virtex-6 FPGA, and its performance was analyzed utilizing XILINX ISE tools. Our design occupies the smallest area comparing with other implementations on the same FPGA type. The proposed design saves in a range between 60.0% and 99.0% of the resources compared with other relevant designs.

Corresponding Author:

Ahmed A. H. Abd-Elkader
Qusier Telecom
Telecom Egypt, TE Qusier, Red Sea, Egypt
Email: eng.ahmed_a_elkader@yahoo.com

1. INTRODUCTION

Cryptography is an important area of concern in the field of information security [1]. The most commonly used public-key cryptographic algorithms, such as RSA, Digital Signature Algorithm (DSA) and Elliptical Curve Cryptography (ECC), primarily rely on modular multiplication [2]. Therefore, a cryptographic system with high performance depends on the construction of modular multiplication. A Montgomery modular multiplier (MMM) is the widely used modular multiplier [3]. MMM is an effective technique for implementing the modular multiplication with large operands in high-performance hardware [4, 5]. For security schemes which are based on public key cryptography, it is important to use hardware modules to achieve a high performance. The security provided by means of public key cryptographic algorithms request a large number of arithmetic operations on abstract algebraic structures (finite fields and groups) [2]. Furthermore, these operations are conventionally executed over large numbers (160-3072 bits), which makes them considerably time-consuming operations. This issue has motivated the researchers to modify new hardware, specialized for accelerating the computation time as the focal design goal, which comes at the cost of high hardware resource consumption. However, a major problem with this kind of applications is that, the embedded systems require a fewer hardware resources. Therefore, this aspect is one of the greatest challenges of implementation of the

light-weight cryptographic systems. The preferred solution is applying cryptographic algorithms on the Field Programmable Gate Array (FPGA). The FPGA is low-cost, versatile and robust. For cryptographic systems in particular; FPGA is able to reconfigure for any new security requirements. Low-cost and low-power FPGAs are available on the market and are expected to become popular with applications like the Wireless Sensor Network (WSN) or the Internet of Things (IoT) [6-9].

This research presents a new approach to the MMM algorithm with higher efficiency and lower area costs, which is an important factor in the implementation of embedded devices systems. The proposed algorithm is a modification of the radix-2 MMM structure to improve the area and the efficiency, no subtraction operations are performed. The previous modification of MMM to discard the final subtraction presented a new set of parameters and more cycles [10, 11]. This method involved a calculation of the Montgomery product with longer operands and more iterations, which could reduce performance seriously. Our work imposes a tighter bound on previous assumptions, with no increase in operand size or number of iterations, which enables us to advance the hardware implementation efficiency. The compact structure of the proposed design over the other competitive designs is appropriate for embedded systems and hardware applications of the IoT devices. The proposed design has been coded in VHDL, and targeted Virtex-6 FPGA platform. The proposed design has been synthesized utilizing Xilinx ISE, and simulated utilizing ModelSim. The new approach various bit-length (160-1024) has been compared with the related works in terms of area in Slice LUTs, frequency in MHz, time in μs, throughput in Mbps, Area-Time and efficiency (Mbps per FPGA LUT). A comparison of the results reveals the improvement of the utilized hardware resources of the proposed design over the related works.

The rest of the paper is organized as follows. Section 2 explorers the classical algorithm of Modular Multiplier Reduction, the original MMM, and radix-2 MMM. Section 3 describes the algorithm, the hardware implementation, and the bounds on the Inputs/Outputs (I/O) of the proposed design. Section 4 provides the simulation and the synthesis results of the implementation of the proposed design. The bounds on the Inputs/Outputs (I/O) of the proposed design has been analyzed. The simulation of the proposed design and the accuracy of the design has been verified using ModelSim. The synthesis results of implementing the proposed design on XILINX Virtex-6 FPGA have been provided utilizing Xilinx ISE. Finally, Section 5 concludes our contribution work.

2. BACKGROUND

2.1. Classical modular multiplication

Algorithm 1 is a straightforward algorithm to compute the modular reduction of two multiplied integers $\mathcal{A}$, B. The first step is obtaining the product α of the integer numbers. The reduction modulo $\mathcal{m}$ step usually involves a division operation $\mathcal{D}$ of α by the modulus $\mathcal{m}$, q is the quotient. Computation of the quotient q and remainder $\mathcal{P}$ when α is divided by $\mathcal{m}$ is demonstrated in [12]. The third step is obtaining the residue $\mathcal{P}$ of the division operation as the result of modular multiplication reduction. It is a very time-consuming operation on both hardware and software platforms.

Algorithm-1 Classical Modular Multiplier
INPUT: Integers ($\mathcal{A}$, $\mathcal{B}$, $\mathcal{m}$)
OUTPUT: $\mathcal{P} = \mathcal{A} \times \mathcal{B} \, mod \, \mathcal{m}$.
1. $\alpha = \mathcal{A} \times \mathcal{B}$;
2. $\mathcal{D} = \alpha / \mathcal{m} = q\mathcal{m} + \mathcal{P}$;
3. Return ($\mathcal{P}$).

2.2. Montgomery modular multiplier algorithm

Montgomery introduced a technique to avoid the division process for the modular reduction of product of two integers [3]. For the three integers $(\mathcal{A}, \mathcal{B}, \mathcal{m})$, Montgomery substituted the division by modulus $\mathcal{m}$ with the division by modulus R. The Montgomery approach to calculate the reduction product of modulo $\mathcal{m}$ for two integers $\mathcal{A}$ and $\mathcal{B}$ is listed:

a) Choosing R > $\mathcal{m}$. R is an integral power of 2. The integer $\mathcal{m}$ must be odd to satisfy the condition $GCD(R, \mathcal{m}) = 1$.
b) Precompute the integers $R^{-1} and \, \grave{\mathcal{m}}$ such that:

$$RR^{-1} - \mathcal{m}\grave{\mathcal{m}} = 1 \quad (1)$$

$$R^{-1} \equiv 1 \, mod \, m \quad (2)$$

$$\grave{\mathcal{m}} = -m^{-1} \, mod \, r \quad (3)$$

c) Transform the operand to its Montgomery domain, which is known also with $REDC$ function [3].

$$REDC(x,y) = x \times y \times R^{-1} mod\ m \quad (4)$$

$$\bar{\mathcal{A}} = REDC(\mathcal{A}, R^2) = \mathcal{A} \times R\ mod\ m \quad (5)$$

$$\bar{\mathcal{B}} = REDC(\mathcal{B}, R^2) = \mathcal{B} \times R\ mod\ m \quad (6)$$

Giving the product in Montgomery domain such that:

$$\bar{\mathcal{P}} = REDC(\bar{\mathcal{A}}, \bar{\mathcal{B}}) = \mathcal{A} \times \mathcal{B} \times \mathrm{R}\ mod\ m \quad (7)$$

$$\text{If } \bar{\mathcal{P}} > m \text{ then } \bar{\mathcal{P}} = \bar{\mathcal{P}} - m \quad (8)$$

The previous step is expensive due to its reduction modulo m. Therefore, Montgomery applied a more efficient way to calculate it.

$$REDC(x,y) = \frac{(\mathcal{S} + \mathcal{S}\ \dot{m}\ mod\ R\ )m\ )}{R} \quad (9)$$

where $\mathcal{S} = x \times y$. This equation is applied for $\bar{\mathcal{A}}, \bar{\mathcal{B}}, \bar{\mathcal{P}}$. To reveal the final result, it is necessary to inverse transformation of the Montgomery domain to its original form:

$$REDC(\bar{\mathcal{P}}, 1) = \frac{(\bar{\mathcal{P}} + \bar{\mathcal{P}}\ \dot{m}\ mod\ R\ )m\ )}{R} \quad (10)$$

The preceding approach is slow because of a lot of addition and multiplication operations. More efficient approach is demonstrated in Algorithm-2, the algorithm Montgomery Modular Multiplier (denoted as MMM algorithm).

```
Algorithm-2 Montgomery Modular Multiplier (MMM)
Input: Integers (A, B, m)[ k bits] radix r representation
where 0 ≤ (A, B) < m, R = r^k, gcd(m, r) = 1,
ṁ = −m^−1 mod R
Output: P = A × B × R^−1 mod m
1. P = 0;
2. For (i from 0 to k − 1, i = i + 1) do
3. t_i = ((P_0 + A_i × B_0) × ṁ) Mod r;
4. P = (P + A_i × B + t_i × m)/r;
5. Loop;
6. If P > m then
   P = P − m;
   end If
7. Return(P).
```

The new approach utilizes the $REDC$ function for two integer operands $\mathcal{A}, \mathcal{B}$ directly. where $\mathcal{S} = \mathcal{A} \times \mathcal{B}$.

$$\mathcal{P} = REDC(\mathcal{A}, \mathcal{B}) = \frac{(\mathcal{S} + \mathcal{S}\ \dot{m}\ mod\ R\ )m\ )}{R} \quad (11)$$

For MMM algorithm there are k iterations, where k is the bit length of the modulus m. The multiplier $\mathcal{A}$ bits are scanned from LSB to MSB, $\mathcal{P}_0\ and\ \mathcal{B}_0$ are the LSBs of $\mathcal{P}\ and\ \mathcal{B}$ respectively. The steps 3 and 4 is repeated for every iteration for its corresponding $\mathcal{A}_i$ and for its accumulated $\mathcal{P}$ according to the step 3 result 1 or 0; modulus m added or not to step 4. To get the validation result the MMM algorithm is used once more.

$$Redc(\mathcal{P}, 1) = \frac{(\mathcal{P} + \mathcal{P}\ \dot{m}\ mod\ R\ )m\ )}{R} \quad (12)$$

```
Algorithm-3 Radix-2 (MMM)
Input: Integers (𝒜, ℬ, 𝓂)[ k bits] radix-2 representation.
where 0 ≤ (𝒜, ℬ) < 𝓂, R = 2^k, gcd(𝓂, 2) = 1,
Output: 𝒫 = 𝒜 × ℬ × 2^−k mod 𝓂
1. 𝒫 = 0;
2. For (i from 0 to k − 1, i = i + 1) do
3. t_i = (𝒫_0 + 𝒜_i × ℬ_0) mod 2;
4. 𝒫 = (𝒫 + 𝒜_i × ℬ + t_i × 𝓂)/2;
5. Loop;
6. If 𝒫 > 𝓂 then
   𝒫 = 𝒫 − 𝓂;
   end If
7. Return (𝒫).
```

Algorithm-3 demonstrates the Radix-2 version of MMM, where $r = 2$, [13, 14]. Therefor, $\acute{m} = 1$. The division by 2 in step-4 is a simple one-bit right shift operation. Therefore, for each iteration loop two additions are required and the final subtraction (step 8).

3. RESEARCH METHOD

3.1. Proposed algorithm of the modular multiplier

The modified MMM Radix-2 modular multiplication algorithm is presented in Algorithm-4. This algorithm is the modification of Algorithm-3 without the final subtraction step, and with a modification of the construction to reduce the area and enhance the efficiency. In Algorithm-3, the multiplier corresponding bit $\mathcal{A}_i$ is multiplied by the multiplicand Lest Significant Bit (LSB) $\mathcal{B}_0$ in step-3, and by the multiplicand $\mathcal{B}$ in step-4. In Algorithm-3, the multiplication in step-3 is AND operation, and in step-4 is a Multiplexer.

```
Algorithm-4 Proposed Modular Multiplier
Input: Integers (𝒜, ℬ, 𝓂)[ k bits] radix 2 representation.
where 0 ≤ (𝒜, ℬ) < 𝓂, gcd(𝓂, 2) = 1.
Output: 𝒫 = 𝒜 × ℬ × 2^−k mod 𝓂
1. α = 0;
2. For (i from 0 to k − 1, i = i + 1) do
3. If 𝒜_i= '1' then
 γ = ℬ;
 else
 γ = 0;
 End If;
4. If α_1 xor γ_0 = '1' then
 ζ = γ + 𝓂;
 else
 ζ = γ;
 End If;
5. α = ζ + v;
6. v = α >>1;
7. Loop;
8. 𝒫 = v
9. Return 𝒫
```

Nevertheless, in step-3 of Algorithm-4 the value of $\mathcal{A}_i$ utilized only once and select between two values; the multiplicand value $\mathcal{B}$ or the zero value. The final subtraction in Algorithm-3 is a source of leakage and hardware consumption. In order to remove the extra subtraction and have a uniform input and output range, Walter [10] proposed altering the range of $\mathcal{A}, \mathcal{B}, \mathcal{m}$ to be within [0, $2k$), increasing the number of iterations from k to $k + 2$, and setting the value of R to $2^{k+2} mod\ \mathcal{m}$ [15, 16]. In Algorithm-4 to decrease the leakage and increase the efficiency there is no subtraction operation also, and the output is $k + 1\ bits$. The precondition in [10] of input operands $\mathcal{A}$ and $\mathcal{B}$ is $\mathcal{A} < 2\mathcal{m}$ and $\mathcal{B} < 2\mathcal{m}$, which can strictly degrade performance [17, 18]. By contrast, the new modification in this work presented by Algorithm-4 retaining the range of operands and modulus within $[0, k)$, that has been enhanced the performance of the proposed design than the other related works. Furthermore, the value of R and iterations in the proposed Algorithm-4 is still 2^k and k,respectively. Therefore, these modifications improve the performance of the proposed algorithm. Furthermore, the hardware resources and interconnects required will be less, and thus the area consumed is also has been reduced.

3.2. Proposed design of radix-2 MMM

The hardware circuit for Algorithm-4 is revealed in Figure 1. The proposed design utilizes a counter instead of using loop to obtain a reduced area, it needs one Clock for each iteration, the clock increases the counter by one. The counter counts k iterations. Due to the registers used to synchronize implemented values; there is a delay of one more clock, therefore, the counter counts from 0 to k. The counter is a part of control unit responsible for scanning the multiplier $\mathcal{A}$ from LSB to MSB and get out one bit for iteration ($\mathcal{A}_i$). The proposed design is composed of one $k+1$ bit Adder, one $k+2$ bit Adder, and two Multiplexers. Initially register $\boldsymbol{v}$ is loaded with zero. The corresponding bit $\mathcal{A}_i$ selects between two values, the multiplicand $\mathcal{B}$ if it is '1' or zero $k - bits$ otherwise. The output of the first Multiplexer is denoted by γ.

The first Adder executes addition of modulus $\mathcal{m}$ and γ, the output is loaded to an input of the second Multiplexer, the other input is γ. In step-4 of Algorithm-4, the inputs of the XOR gate are the bit of the order one of the register α (α_1) and the LSB of the register γ (γ_0). A_1 is utilized instead of v_0 to reduce the critical path delay. The output of the XOR gate is the selector of the second Multiplexer to define even output (Z) of the Multiplexer. Step-4 defines the purpose of XOR gate and the second Multiplexer. According to the condition that modulus $\mathcal{m}$ must be odd; (odd integer + $\mathcal{m}$) is an even integer. If $\boldsymbol{\gamma}$ is odd it converted to even by adding modulus $\mathcal{m}$ to it, step-4. The second Adder executes addition of accumulator $\boldsymbol{v}$ and Z, the output is loaded to variable α. The value of α is shifted right one bit, divided by 2, and loaded to the register of the accumulated value $\boldsymbol{v}$. Those procedures are reiterated for each clock. Thus, the given architecture takes k+1 clock cycle to make out the calculation process, subsequently, the value of register $\boldsymbol{v}$ is loaded to the output $\mathcal{P}$. The output $\mathcal{P}$, $k+1$ $bits$,is the reduction product of the two input integers multiplied by 2^{-k}. The size of the hardware components depends on the size of the operands and the modulus for the multiplication process. This design can be easily scaled according to requirement for any number of bits.

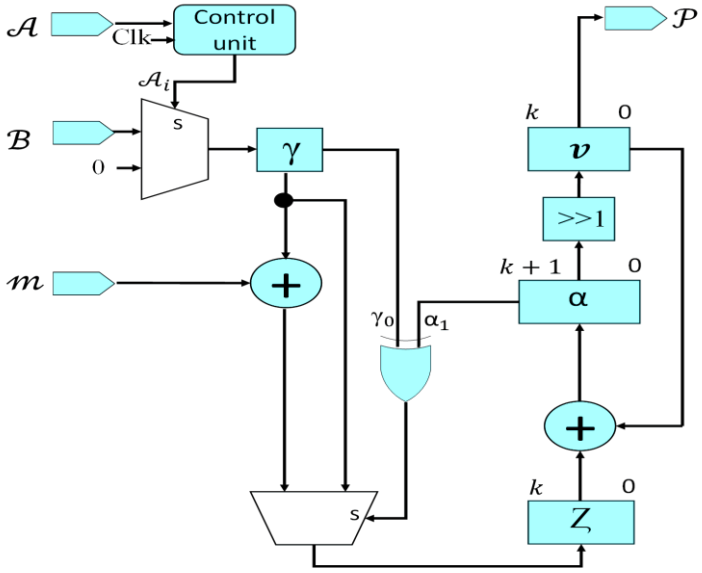

Figure 1. Proposed modular multiplier circuit

3.3. Bounds on the I/O

Outputs from multiplications are re-used as inputs throughout the cryptographic systems. So, it's important to keep those numbers bound. In particular, for all outputs $\mathcal{P}$ we will show that $\mathcal{P} < 2^k$ is maintainable. The four variables in the Algorithm-4 are γ, Z, α, and $\boldsymbol{v}$, the bounded bit-length of it are $k, k+1, k+2$, and $k+1$, respectively, Figure 1. The maximum value of modulus $\mathcal{m}$ is:

$$\mathcal{m} = 2^k - 1 \tag{13}$$

The precondition of MMM is $\mathcal{B} < \mathcal{m}$, thus, the maximum value of variable γ is the maximum value of the multiplicand $\mathcal{B}$:

$$\gamma = \mathcal{B} = 2^k - 2 \tag{14}$$

The value of variable Ƶ is:

$$Ƶ = \gamma + m \tag{15}$$

The maximum value of variable α is:

$$\alpha = Ƶ + v_{h-1} \tag{16}$$

where, h is the iteration number. The value of the accumulated variable v_h is:

$$v = \frac{\alpha}{2} \tag{17}$$

After the last iteration $h = k$, and the value of the output $\mathcal{P}$ is:

$$\mathcal{P} = v_k \tag{18}$$

The verification that the output and other variables for $k = 8$ are below the size of their implemented hardware variables shown in Table 1. The ceiling function maps $\alpha/2$ to the least integer greater than or equal to $\alpha/2$. In general, the maximum values of the Algorithm-4 variables are as follows for any $k - bit$ $(\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}, \boldsymbol{m})$:

$$\gamma = 2^k - 2 \tag{19}$$

$$Ƶ = \gamma + m = 2^{k+1} - 3 \tag{20}$$

$$\alpha = Ƶ + \boldsymbol{v}_{h-1} = 2^{k+2} - 2^{k-(h-2)} - 5 \tag{21}$$

$$v_h = \left\lceil \frac{\alpha}{2} \right\rceil = 2^{k+1} - 2^{k-(h-1)} - 2 \tag{22}$$

The maximum value of the output $\mathcal{P}$ is:

$\mathcal{P} = v_k = 2^{k+1} - 4$Therefore, the bit-length of the output $\mathcal{P}$ is bounded on $k + 1$ bits.

Table 1. Bounds on the output of the proposed design

h	v_{h-1} $< 2^{k+1}$	γ $< 2^k$	$Ƶ = \gamma + m$ $< 2^{k+1}$	$\alpha = Ƶ + v_{h-1}$ $< 2^{k+2}$	$v_h = \lceil \alpha/2 \rceil$ $< 2^{k+1}$
1	0	$2^k - 2$	$2^{k+1} - 3$	$2^{k+1} - 3$	$2^k - 1$
2	$2^k - 1$	$2^k - 2$	$2^{k+1} - 3$	$2^{k+2} - 2^k - 4$	$2^{k+1} - 2^{k-1} - 2$
3	$2^{k+1} - 2^{k-1} - 2$	$2^k - 2$	$2^{k+1} - 3$	$2^{k+2} - 2^{k-1} - 5$	$2^{k+1} - 2^{k-2} - 2$
4	$2^{k+1} - 2^{k-2} - 2$	$2^k - 2$	$2^{k+1} - 3$	$2^{k+2} - 2^{k-2} - 5$	$2^{k+1} - 2^{k-3} - 2$
5	$2^{k+1} - 2^{k-3} - 2$	$2^k - 2$	$2^{k+1} - 3$	$2^{k+2} - 2^{k-3} - 5$	$2^{k+1} - 2^{k-4} - 2$
6	$2^{k+1} - 2^{k-4} - 2$	$2^k - 2$	$2^{k+1} - 3$	$2^{k+2} - 2^{k-4} - 5$	$2^{k+1} - 2^{k-5} - 2$
7	$2^{k+1} - 2^{k-5} - 2$	$2^k - 2$	$2^{k+1} - 3$	$2^{k+2} - 2^{k-5} - 5$	$2^{k+1} - 2^{k-6} - 2$
8	$2^{k+1} - 2^{k-6} - 2$	$2^k - 2$	$2^{k+1} - 3$	$2^{k+2} - 2^{k-6} - 5$	$2^{k+1} - 2^{k-7} - 2$

4. RESULTS AND DISCUSSION

The hardware architectures for MMM, existing and proposed method, has been presented in Sections 2 and 3, respectively. This architecture of the proposed design has been implemented in VHDL. This work has been simulated and synthesized using ModelSim and Xilinx ISE tools respectively, where the target device is used Virtex-6 device XC6VLX760-2FF1760 FPGA.

4.1. Simulation of the proposed algorithm

The simulation and verification of the design is carried out using ModelSim, Figure 2. As an example, $k = 8$, $\mathcal{A} = 165, \mathcal{B} = 231$, and the modulus $m = 245$. As expected, the final result is $\mathcal{P} = 280$,where, $\mathcal{P} = \mathcal{A} \times \mathcal{B} \times 2^{-k} \, mod \, m = 165 * 231 * 2^{-8} \, mod \, 245 = 280$. The result is $k + 1$ bits as a consequence of eradicating the final subtraction step.

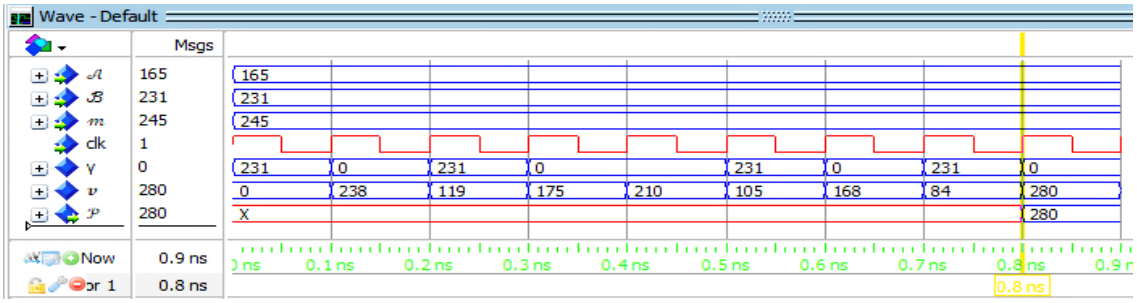

Figure 2. Simulation of the proposed MMM

4.2. Implementation and comparison

In this section, a comparison of state of the art $GF(p)$ hardware Montgomery Modular Multiplication is presented. Table 2 shows that, the performance analysis of our work with existing designs. The metrics used to evaluate the proposed hardware designs for various bit-length are area in Slice LUTs, frequency in MHz, time in μs, throughput in Mbps, Area-Time per bit-length (AT/b), and efficiency (Mbps per FPGA LUT). Efficiency metric has been used in previous works to evaluate the area resources used and performance achieved in cryptographic hardware architectures [19].

K. Javeed [14] presented a radix-2 implementation of the MMM and IMM algorithms on FPGA. The outcomes indicate that the radix-2 MMM design is more proficient in terms of calculation time, FPGA slice area and throughput as compared to the radix-2 IMM design. The synthesis results of implementation of MMM are shown in Table 2. [14] offers low computation time and reasonable throughput, but at the cost of higher area. Our proposed design has an advance on all aspects over the comparative one. Our work has achieved an improvement in efficiency by 61%, while consuming only about 40 % of the total slice-LUTs when compared with [14].

S. Ghosh [20] presented a modification of an interleaved multiplier using Montgomery ladder and the high-speed adder circuits. The design of [20] offers higher frequency and throughput for 256 and 224 bit-length. In contrast, this work consumes much higher area, a higher AT/b requirement, and lower efficiency. A comparison of the two works reveals that our proposed algorithm has achieved an advance in efficiency by an average 5.25 times the efficiency of [20], whereas also consuming only around 19% of the total slice-LUTs.

Based on interleaved multiplication algorithm; K. Javeed [21] presented a modified version of radix-4 and radix-8 Booth encoded modular multipliers over general $GF(p)$. Because of the utilization of higher radix approach; the design has a lower clock- cycles, and higher throughput for 224 and 256 bit-length. On the other hand, this approach utilized much hardware resources, a higher AT/b requirement, and lower efficiency. Our proposed algorithm has performed an improvement in efficiency by around seven-times the efficiency of [21], furthermore, consuming less than 13% of the total slice-LUTs.

The previous works introduced the term “Normalized-LUT” to stand for DSP cost in the measure of LUT [11, 22, 23]. Yan [24] presented an Implementation of hybrid 256-bit Montgomery modular multiplier over GF (p) on FPGAs, which utilizes Karatsuba and Knuth multiplication algorithms in various stages. This work provided higher frequency and higher throughput over our proposed work. By contrast, the cost of utilizing the hardware resource utilization was very high, and the efficiency is very low. Our work has attained an enhancement in efficiency by 10.5 times the efficiency of [24], while consuming only 1.0% of the total slice-LUTs.

Based on an interleaved multiplication algorithm and Montgomery power laddering, Javeed [25] presented an implementation of radix-4 modular multipliers. This work has a development in frequency, throughput over the previous work in [21] by twice. Nevertheless, our proposed algorithm has an advanced efficiency over algorithm in [25] by four- times, while consuming only 12% of the total slice-LUTs.

Liu [11] proposed a design of 258-bit multiplier based on KO-3 algorithm construed from Karatsuba algorithm. This study has a perfection in time, and throughput over the proposed design, but at the cost of the much higher area, thus having a higher AT/b requirement and lower efficiency. Our work has attained an advance in efficiency by 2.7 times the efficiency of [11], while consuming only 0.33% of the total slice-LUTs.

J. Ding [22] introduced Broken-Karatsuba multiplication with the non-least-positive form. Based on the modified modular multiplication algorithm, a 256-bit two-stage modular multiplier was built. This work accomplishes the computation of 256-bit MMM in just 9 cycles; thus, it has a perfect delay time, and a high throughput. On the other hand, the cost of the area is much higher, thus having a higher AT/b requirement and lower efficiency. Our work has attained an improvement in efficiency by 1.8 times the efficiency of [22], while consuming only 1.8% of the total slice-LUTs.

J. Ding [26] implemented Montgomery modular multiplication and utilized a modular multiplication approach based on non-least positive form (NLP) combining Karatsuba and schoolbook multiplication, which saves 2 base multiplications compared to Karatsuba-only designs. This work performs 256-bit MMM computation in just 16 cycles; hence, it has a reasonable delay time and a high throughput. By contrast, this work consumes much higher area, a higher requirement for AT / b and less efficiency. A comparison of the two works shows that our proposed algorithm has achieved an average efficiency advance of 1.1 times the efficiency of 3-way technique in [26], whereas only about 3.4% of the total slice-LUTs are consumed.

Table 2. Comparison of modular multiplication implementations on FPGA

Design	Bit-length	Frequency (MHz)	Slice LUTs	DSP	Normalized LUT	Time (μs)	Throughput (Mbps)	AT/b	Efficiency (Mbps/LUT)
[14]	160	207	958	0	958	0.78	205.13	4.67	0.214
	192	186	1150	0	1150	1.04	184.62	6.23	0.161
	224	169	1342	0	1342	1.34	167.16	8.03	0.125
	256	154	1534	0	1534	1.68	152.38	10.07	0.099
	384	115	2302	0	2302	3.36	114.29	20.14	0.050
	512	92.5	3070	0	3070	5.56	92.09	33.34	0.030
	1024	51	6142	0	6142	20.10	50.95	120.56	0.008
[20]	160	191	2002	0	2002	0.84	190.48	10.51	0.095
	192	184.6	2401	0	2401	1.04	184.62	13.01	0.077
	224	179.1	2787	0	2787	1.25	179.20	15.55	0.064
	256	174	3207	0	3207	1.48	172.97	18.54	0.054
[21]	160	91.6	2911	0	2911	0.88	181.82	16.01	0.062
	192	89	3511	0	3511	1.08	177.78	19.75	0.051
	224	87.3	4053	0	4053	1.29	173.64	23.34	0.043
	256	85.5	4606	0	4606	1.50	170.67	26.99	0.037
[24]	256	206	16900	108	81376	0.13	1954.20	41.65	0.024
[25]	256	166	5300	0	5300	0.77	332.47	15.94	0.063
[11]	256	68	187900	0	187900	0.01	17414.97	10.79	0.093
[22]	256	161	4700	48	33356	0.06	4579.56	7.28	0.137
[26]	256	256	3500	24	17828.00	0.06	4096.00	4.35	0.230
Proposed design	160	210.82	379	0	379	0.76	209.51	1.81	0.553
	192	189.13	451	0	451	1.02	188.15	2.40	0.417
	224	171.49	525	0	525	1.31	170.73	3.08	0.325
	256	156.86	614	0	614	1.64	156.24	3.93	0.254
	384	116.94	914	0	914	3.29	116.64	7.84	0.128
	512	93.22	1211	0	1211	5.50	93.04	13.02	0.077
	1024	51.46	2408	0	2408	19.92	51.41	46.84	0.021

5. CONCLUSION

This paper introduces a modular multiplier with a novel algorithm and compact architecture based on the MMM algorithm without any subtraction or multiplication procedures. The motivation behind this work, is to present a design of a lower area and higher efficiency than the relevant designs of MMM. This work has been implemented on Xilinx Virtex-6 FPGA platform, and verified by Modelsim simulator. The implementations of various bit lengths have been evaluated in terms of area, frequency, throughput, AT/b, and efficiency. The proposed multiplier performs a 256-bit and 1024-bit modular multiplication in 1.64 μs and 19.92 μs, occupies 614 LUTs and 2408 LUTs, and runs at 156 MHz and 51 MHz, respectively. We can infer from the synthesized results that the proposed multiplier consumes the lowest area as compared with other multipliers. Consequently, the developed architecture has a competent area-time and enhanced efficiency. Thus, the developed architecture is a very adequate to be applied for efficient lightweight asymmetric cryptosystems.

REFERENCES

[1] H. Delfs and H. Knebl, "Introduction to Cryptography Principles and Applications", *Third. Berlin*, Heidelberg: Springer Berlin Heidelberg, 2015.
[2] W. Stallings, "Cryptography and Network Security Principles and Practice, Six", *New Jersey: Pearson Education*, 2014.
[3] P. L. Montgomery, "Modular Multiplication Without Trial Division," *Math. Comput.*, vol. 44, no. 170, p. 519, 1985, doi: 10.2307/2007970.
[4] A. Parihar and S. Nakhate, "Fast Montgomery modular multiplier for Rivest-Shamir-Adleman cryptosystem," *IET Inf. Secur.*, vol. 13, no. 3, pp. 231-238, 2019, doi: 10.1049/iet-ifs.2018.5191.

[5] E. Ozcan and S. S. Erdem, "A High Performance Full-Word Barrett Multiplier Designed for FPGAs with DSP Resources," in *Digital Circuits and Sub-Systems II*, vol. 2, no. 3, pp. 73-76, 2019. doi: 10.1109/prime.2019.8787740.
[6] F. G. Abdulkadhim, Z. Yi, C. Tang, M. Khalid, and S. A. Waheeb, "A survey on the applications of IoT: An investigation into existing environments, present challenges and future opportunities," *TELKOMNIKA (Telecommunication, Computing, Electronics and Control)*, vol. 18, no. 3, pp. 1447-1458, 2020, doi: 10.12928/TELKOMNIKA.v18i3.15604.
[7] R. Jyothi and N. G. Cholli, "An efficient approach for secured communication in wireless sensor networks," *Int. J. Electr. Comput. Eng.*, vol. 10, no. 2, pp. 1641-1647, 2020, doi: 10.11591/ijece.v10i2.
[8] M. Gunturi, H. D. Kotha, and M. Srinivasa Reddy, "An overview of internet of things," *J. Adv. Res. Dyn. Control Syst.*, vol. 10, no. 9, pp. 659-665, 2018, doi: 10.12928/telkomnika.v18i5.15911.
[9] A. Karim Mohamed Ibrahim, R. A. Rashid, A. H. F. A. Hamid, M. Adib Sarijari, and M. A. Baharudin, "Lightweight IoT middleware for rapid application development," *TELKOMNIKA (Telecommunication, Computing, Electronics and Control)*, vol. 17, no. 3, pp. 1385-1392, 2019, doi: 10.12928/TELKOMNIKA.V17I3.11793.
[10] C. D. Walter, "Montgomery exponentiation needs no final subtractions," *Electron. Lett.*, vol. 35, no. 21, pp. 1831-1832, 1999, doi: 10.1049/el:19991230.
[11] R. Liu and S. Li, "A design and implementation of montgomery modular multiplier," in *Proceedings - IEEE International Symposium on Circuits and Systems*, vol. 2019-May, no. 1, pp. 1-4, 2019, doi: 10.1109/ISCAS.2019.8702684.
[12] B. Schneier, "Applied Cryptography," *Electr. Eng.*, vol. 1, no. 32, pp. 429-455, 1996, doi: 10.1.1.99.2838.
[13] K. Pratibha and R. Muthaiah, "Survey on Hardware Implementation of Montgomery Modular," *Int. J. Pure Appl. Math.*, vol. 119, no. 12, pp. 13437-13452, 2018.
[14] K. Javeed, D. Irwin, and X. Wang, "Design and performance comparison of modular multipliers implemented on FPGA platform," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10039 LNCS, pp. 251-260, 2016, doi: 10.1007/978-3-319-48671-0_23.
[15] G. S. Y. Baek, "Uniform Montgomery multiplier," *J. Cryptogr. Eng.*, 2019, doi: 10.1007/s13389-019-00213-7.
[16] M. Der Shieh, J. H. Chen, W. C. Lin, and H. H. Wu, "A new algorithm for high-speed modular multiplication design," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 56, no. 9, pp. 2009-2019, 2009, doi: 10.1109/TCSI.2008.2011585.
[17] Z. Liu and J. Großschädl, "New speed records for montgomery modular multiplication on 8-bit AVR microcontrollers," *Prog. Cryptology--AFRICACRYPT 2014*, vol. 8469 LNCS, pp. 215-234, 2014, doi: 10.1007/978-3-319-06734-6_14.
[18] G. Hachez and J. J. Quisquater, "Montgomery exponentiation with no final subtractions: Improved results," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 1965 LNCS, pp. 293-301, 2000, doi: 10.1007/3-540-44499-8-23.
[19] L. Rodríguez-Flores, M. Morales-Sandoval, R. Cumplido, C. Feregrino-Uribe, and I. Algredo-Badillo, "Compact FPGA hardware architecture for public key encryption in embedded devices," *PLoS One*, vol. 13, no. 1, pp. 1-21, 2018, doi: 10.1371/journal.pone.0190939.
[20] S. Ghosh, D. Mukhopadhyay, and D. R. Chowdhury, "High Speed F p Multipliers and Adders on FPGA Platform," in *Design and Architectures for Signal and Image Processing (DASIP)*, pp. 21-26, 2010.
[21] K. Javeed and X. Wang, "Radix-4 and radix-8 booth encoded interleaved modular multipliers over general Fp," *Conf. Dig. - 24th Int. Conf. F. Program. Log. Appl. FPL 2014*, 2014, doi: 10.1109/FPL.2014.6927452.
[22] J. Ding and S. Li, "Broken-Karatsuba multiplication and its application to Montgomery modular multiplication," in *2017 27th International Conference on Field Programmable Logic and Applications, FPL 2017*, pp. 5-8, 2017, doi: 10.23919/FPL.2017.8056769.
[23] G. C. T. Chow, K. Egurot, W. Luk, and P. Leong, "A karatsuba-based montgomery multiplier," *Proc. - 2010 Int. Conf. F. Program. Log. Appl. FPL 2010*, pp. 434-437, 2010, doi: 10.1109/FPL.2010.89.
[24] X. Yan, G. Wu, D. Wu, F. Zheng, and X. Xie, "An implementation of montgomery modular multiplication on FPGAs," in *Proceedings - 2013 International Conference on Information Science and Cloud Computing, ISCC 2013*, pp. 32-38, 2014, doi: 10.1109/ISCC.2013.19.
[25] K. Javeed, X. Wang, and M. Scott, "Serial and parallel interleaved modular multipliers on FPGA platform," *25th Int. Conf. F. Program. Log. Appl. FPL 2015*, pp. 2-5, 2015, doi: 10.1109/FPL.2015.7293986.
[26] J. Ding and S. Li, "A low-latency and low-cost Montgomery modular multiplier based on NLP multiplication," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 67, no. 7, pp. 1319-1323, 2020, doi: 10.1109/TCSII.2019.2932328.