A Low-Latency and Resource-Efficient Scalable RSA Cryptoprocessor Architecture

Trio Adiono, Iput Heri Kurniawan, and Rachmad Vidya Wicaksana Putra

Abstract—RSA is one of the well-known cryptography method used in asymmetric cryptosystems. But, RSA challenges on architecture, performance, power and resource consumption still can be improved. In this research, we propose a low-latency and resource-efficient scalable RSA cryptoprocessor architecture to deal with power and resource consumption issues. It is obtained using two approaches. First, optimization of Radix-4 Montgomery multiplication that yields the reduction on resource utilization and latency. Second, designing a scalable architecture based on the optimized Radix-4 Montgomery multiplication. The proposed design is verified in FPGA through simulation and image encryption application. Synthesis results show that the proposed design achieves an optimal design in respect of low-latency, resource-efficient and scalability. It only requires 227k cycles latency and consumes 13k logic gate utilization for 512-bit RSA.

Index Terms—cryptography, low-latency, radix-4 Montgomery, resource-efficient, RSA

I. INTRODUCTION

NOWADAYS, we need to deal with the rapid increasing on the amount of data. Because, there are massive data transactions within various communication and computing applications, such as e-mail, cloud service, data storage, e-commerce, e-banking, etc [1]. In this condition, data security is playing a crucial role [2]. We acknowledge that one of the central issues is about the data encryption-decryption process. Researchers realized this problem and tried to overcome it using cryptography. This technology enables us to transmit or store sensitive and secret information across networks. Cryptography also provides the data security objectives such as data confidentiality, data integrity, authentication, and non-repudiation [1]. Currently, there are two main solutions namely symmetric (secret-key) and asymmetric (public-key) cryptography.

In the symmetric cryptography, the key must remain secret because it is used for both of data encryption and decryption. Only the authentic users (i.e. sender and receiver) know and share the same key. This scheme is kind

of inflexible, because once the key is revealed to other parties, the authentic users need to change the key shortly. The problem occurs if the secret-key is revealed to other parties without the authentic users notice about it. Despite of this issue, the secret-key is still actively researched and used in today's applications, such as image encryption [3]. Diffie-Hellman proposed the concept of the asymmetric cryptography in 1976 [4], to deal with this problem. In contrast to symmetric one, the asymmetric cryptography uses different keys for each encryption and decryption which are called public and private keys. One of the widely-used public-key cryptography technique is RSA that was firstly developed by Rivest, Shamir dan Adleman [5].

Main challenge on RSA algorithm is on the computational complexity for hardware implementation. Because, it requires large number of modular exponentiation that yields long computational time [2]. Hence, to perform RSA in fast and practical way, we need to design an efficient modular multiplication and implement it in physical hardware. There are several algorithms developed for this modular multiplication and the most popular one is the Montgomery modular multiplication. This algorithm enables the extensive use of RSA in many platforms, such as server, embedded systems, and even chip level. In the chip level, the existence of RSA hardware as crypto-accelerators [6] is favored because it can speed up the computational process. Furthermore, to obtain an optimal capability of the cryptosystem, RSA implementation on a configurable device such as Field Programmable Gate Array (FPGA) is considered, because it can retain the speed performance and physical security while providing the design flexibility [2]. Moreover, the use of FPGA is a preferred because it shortens the design time to market [7].

The emerging issues nowadays are on power and resource consumptions since many data transactions need an efficient power and resource. To overcome these problems, we propose a low-latency and resource-efficient scalable RSA architecture based on optimized Radix-4 Montgomery multiplication. Low-latency and resource-efficient lead the design to be low-power while consuming small resources. Meanwhile, scalability leads the design to accommodate various bit-width that correspond to the given parameters which later determine its configurability on FPGA. The importance of these scalability and configurability are for further use, such as accelerator in a System-on-Chip (SoC) [8], [9]. The optimization of Radix-4 Montgomery modular multiplication is obtained by simplifying the modular multiplication and exponentiation algorithms, utilizing pipeline and Carry-Save-Adder (CSA). The scalability and

Manuscript received July 05, 2017; revised July XX, 20XX.

T. Adiono is with the University Center of Excellence on Microelectronics, Institut Teknologi Bandung (ITB), Bandung, 40132, Indonesia (phone: +62-22-2506280; e-mail: tadiono@stei.itb.ac.id).

I. H. Kurniawan was with the University Center of Excellence on Microelectronics, Institut Teknologi Bandung (ITB), Bandung, 40132, Indonesia. He is now with Raon-Tech, Seongnam, South Korea (e-mail: iputkurniawan@gmail.com).

R. V. W. Putra was with the University Center of Excellence on Microelectronics, Institut Teknologi Bandung (ITB), Bandung, 40132, Indonesia. (corresponding author, phone: +62-22-2506280; e-mail: rachmad@pme.itb.ac.id).

Fig. 1. Illustration of the RSA encipher and decipher process

configurability features are obtained by designing a generic modular architecture.

This article is structured in several sections. In section 1, we summarize the research background. Afterwards, we briefly explain the RSA cryptosystems and elaborate the design methodology in section 2 and 3, respectively. Section 4 outlines the proposed architecture and it is followed by results and discussions on section 5. In sections 6, we draw some conclusions.

II. RSA CRYPTOSYSTEM

A. Overview

The principles of RSA cryptosystems can be explained well using a famous Alice-Bob-Eve illustration. Suppose that there are two persons who want to communicate to each other, Alice (A) and Bob (B). Alice is the message sender and Bob is the message receiver. The plaintext message from Alice (P_A) is encrypted by RSA encipher using Bob's public key (E_B) and modulus number (M). Form of the plaintext message (P_A) is changed into a ciphertext message (C_A). The ciphertext message is then transmitted to Bob via reliable network. If the ciphertext message (C_A) arrives, Bob can decipher it by RSA decipher using his private key (D_B) and modulus number (M) and obtain the secret plaintext message from Alice (P_A). If there is Eve who tries to eavesdrop the ciphertext message (C_A) in the middle of transmission, the action is difficult to do in a reasonable amount of time, even if using the supercomputers. The illustration is depicted in Fig. 1.

B. RSA Keys Generator

To generate the RSA keys, we need to compute few formulations. Firstly, we need to define several variables formatted in integer, as follow:

In the sender side

C : ciphertext, encrypted data to send
 E : public-key for data encryption
 P : plaintext, message to send

In the receiver side

C : ciphertext, encrypted data to receive
 D : private-key for data decryption
 P : plaintext, message to obtain

In both sides, sender and receiver

M : modulus number

computation. P is the plaintext message to send and is encrypted using receiver's public-key E and modulus M to produce the ciphertext C . Hence, both sender and receiver need to know these numbers (i.e., E , M , and C). These numbers could also be eavesdropped by other parties, but it is likely difficult to do a reverse computation to obtain plaintext message P in a reasonable amount of time. If the ciphertext C arrives at the receiver side, decryption process can be initiated. The receiver decrypts the ciphertext C using receiver's private-key D and modulus M to obtain the plaintext P . Data decryption processing is performed using Eq. 2.

$$C = P^E \bmod M \quad (1)$$

$$P = C^D \bmod M \quad (2)$$

To get E , D , and M , we firstly need to select two prime number randomly which are big enough named p and q , in which $p \neq q$. The value of modulus M can be computed by Eq. 3. Afterwards, we need to compute the totient number $\phi(M) = L$ using Eq. 4 which utilizes Least Common Multiple (LCM) operation. This totient number is then utilized to get public-key number E for encryption. The public-key E is computed using Greatest Common Divisor (GCD) operation as presented in Eq. 5. The value of E must be smaller than totient L ($1 < E < L$) and co-prime of totient L . To speed up the calculation, we need to choose public-key E as relatively small number.

$$M = p \times q \quad (3)$$

$$\phi(M) = L = \text{LCM}(p-1, q-1) \quad (4)$$

$$\text{GCD}(L, E) = 1 \quad (5)$$

The private-key D for decryption is selected corresponding to the previously selected totient (L) and public-key (E) numbers. The relation used for determining private-key D is presented as Eq. 6. Variable H represents any integer number that complies with the Eq. 6. Thus, we can also state different expression of this equation as Eq. 7. The private-key D can be obtained easily if modulus M is small enough. On the other hand, if modulus M value is big enough (e.g., thousands), the computation needed to crack the decryption number D could spend more than 10 years using today's supercomputers. This selection of numbers is crucial in designing RSA cryptosystems.

$$E \times D = H \times L + 1 \quad (6)$$

$$(E \times D) \bmod L = 1 \quad (7)$$

III. PROPOSED DESIGN METHODOLOGY

RSA algorithm is designed to find two big prime numbers easily, but relatively impossible to find their factor. Montgomery modular multiplication is a popular algorithm used to compute power operation and big number modulus through modular multiplication. In this research, we optimize the Radix-4 Montgomery modular multiplication to achieve an efficient hardware implementation. A detailed explanation is given in the following subsections.

In the sender side, data encryption is performed using Eq. 1. The mod operator means the mathematical modulus

A. Montgomery Modular Multiplication

Montgomery modular multiplication or Montgomery multiplication for short, is used to break the power and modulus operation into modular multiplication. Breaking the power operation is essential, because such operation causes an increasing of sources significantly. For example, in RSA, we need to compute $C = P^E \bmod M$. If there is no simplification process, P^E needs to be done in a straightforward way. If P has b bit-width, then P^E needs $b \times E$ bit-width. It is inefficient for hardware implementation. Moreover, modular operation seems to increase the complexity if it is not handled appropriately.

To solve the challenges, we need to extract the equation $C = P^E \bmod M$ into Eq. 8 and Eq. 9. Benefit from Eq. 9 is that we avoid the bit-width increment from multiplication. Because, we obtain the result of the previous modulus and utilize it for the next multiplication and modulus processes. Hence, bit-width size of multiplication process is bounded. This is the first idea in Montgomery multiplication concept.

$$C = (P_1 \times P_2 \times \dots \times P_E) \bmod m \quad (8)$$

$$C = (((P_1 \bmod m) \times P_2 \bmod m) \times \dots) \times P_E \bmod m \quad (9)$$

The second idea is to transform the computational domain into Montgomery domain. Basic transformation is presented in Eq. 10, with $r = 2n$ and n represents the bit-width of modulus m . Variable r is the inverse of $r-1$ and their relation is characterized by Eq. 11.

$$\tilde{a} = \text{MonMul}(a, r^2) = (a \times r) \bmod m \quad (10)$$

$$(r \times r^{-1}) \bmod m = 1 \quad (11)$$

As illustration, we are likely to face condition like Eq. 12 as generic form in RSA modular multiplication to represent a bracket computation from Eq. 9.

$$c = (a \times b) \bmod m \quad (12)$$

If we need to perform the computation in Montgomery domain, basically we perform these following steps:

- Convert the variables a and b into Montgomery domain using Eq. 13 and Eq. 14.

$$\tilde{a} = \text{MonMul}(a, r^2) = (a \times r) \bmod m \quad (13)$$

$$\tilde{b} = \text{MonMul}(b, r^2) = (b \times r) \bmod m \quad (14)$$

- Do modular multiplication between and in Montgomery domain using Eq. 15.

$$\tilde{c} = \text{MonMul}(\tilde{a}, \tilde{b}) = (\tilde{a} \times \tilde{b} \times r^{-1}) \bmod m \quad (15)$$

- Convert the variable $\tilde{c}$ into previous domain using Eq. 16.

$$c = \text{MonMul}(\tilde{c}, 1) = (\tilde{c} \times r^{-1}) \bmod m \quad (16)$$

B. Optimized Radix-4 Montgomery Multiplication

There are several types of Montgomery multiplication proposed in the past, such as Radix-2 and Radix-4 based calculation. In this work, we propose an optimized Radix-4

Montgomery multiplication that utilizes 2 CSA. This proposed algorithm can refine the speed performance while maintain small resources. Step-by-step optimization process is presented in next paragraphs.

The original Radix-2 Montgomery multiplication is presented as the algorithm-1. Comparison and subtraction in step-6 and step-7 of algorithm-1 cause inefficient hardware implementation. It can be proven that $S_{i+1} < 2M$ is unchanged if $A, B < M$ [10]. But, S_m can not be used as input of A or B for next modular multiplication. If we do 2 more iterations with $a_{m+1} = 0$ and input $A, B < 2M$, equation $S_{m+2} < 2M$ is fulfilled [10]. Therefore, S_{m+2} can be used again as input B for next modular multiplication. Thus, we allow S to has width 2-bit longer than before.

Algorithm-1: Radix-2 Montgomery Modular Multiplication

```

1.  S = 0;
2.  for i = 0 to m-1
3.    qi = (Si + aiB) mod 2
4.    Si+1 = (Si + qiM + aiB) / 2
5.  end for
6.  if Sm ≥ M
7.    return Sm - M
8.  else
9.    return Sm
10. End

```

Since the lower radix usually needs more computational latencies [1], it is better to use higher radix and overcome its consequences. Thus, utilizing Radix-4 is preferred than Radix-2. The original Radix-4 Montgomery multiplication is presented as algorithm-2 as advancement from algorithm-1. From algorithm-2, we eventually realize that the step-6 is inefficient to implement in hardware. Because, it has addition and multiplication within the loop. Moreover, there are 2 additions and 3 multiplications involved.

Algorithm-2: Radix-4 Montgomery Modular Multiplication

```

1.  n = count_bit(N);
2.  C = BoothEnc(A); % [c(n/2), c(n/2 - 1),
   c(n/2 - 2) .... c(1) c(0)]
3.  P = 0;
4.  for(i=1:(n/2)+1)
5.    q = mod(P + C(i)*B, 4);
6.    P = bitshift(P + C(i)*B + Sel(q,
   N(1))*N, - 2);
7.  end

```

To overcome issues on algorithm-2, we optimize few things there to obtain the algorithm-3 using substitution and normalization techniques. We substitute the 2 addition processes in step-6 of algorithm-2 with 2 CSAs respectively and 2 columns right-shifting operation. To corroborate its optimization, we also normalize the value of sum and $carry$.

The normalization is conducted at step-10 until step-27 in algorithm-3. Purpose of this normalization is to produce sum and $carry$ values that are the same with value P in step-6 of algorithm-2. Algorithm of normalization is presented in Eq. 17. The proposed algorithm-3 has several benefits, such as: 1) there are only 2 CSAs utilized; 2) there is only one addition outside the iteration module; 3) the process only needs $(n/2) + 1$ iteration latency compared to Radix-2 that

has n iteration latency; and 4) it has faster performance than interleave multiplication.

$$\text{bitshift}(sum + carry + C[1] \times B + (4 - q) \times N, -2) = \begin{cases} (sum, carry) = CSA(sum, carry, C(i) \times B) \\ (sum, carry) = CSA(sum, carry, (4 - q) \times N) \\ sum = \text{bitshift}(sum, -2) \\ carry = \text{bitshift}(carry, -2) \\ sum = \text{normalize}(sum, carry) + sum \end{cases} \quad (17)$$

Algorithm-3: Optimized Radix-4 Montgomery Multiplication

```
% [c(n/2), c(n/2 - 1), c(n/2 - 2) . . . . , c(0)]
1. Abooth = BoothEnc(A);
2. P = 0;
3. S = 0;
4. C = 0;
5. for(i=1:(n/2)+1)
6.   q = mod(P + Abooth(i)*B, 4);
7.   [S,C] = CSA(S, C, Abooth(i)*B);
8.   [S,C] = CSA(S, C, Sel(q, N(1))*N);
9.
10.  % normalize
11.  if(S >= 0 && C >= 0)
12.    a = S(1:0) + Ci(1:0);
13.  elseif(S < 0 && C >= 0)
14.    a = C(1:0) - S(1:0);
15.  elseif(C < 0 && S >= 0)
16.    a = S(1:0) - C(1:0);
17.  elseif(S < 0 && C < 0)
18.    a = -S(1:0) - C(1:0);
19.  end
20.
21.  if(a >= 4)
22.    a = 1;
23.  elseif(a <= -4)
24.    a = -1;
25.  else
26.    a = 0;
27.  end
28.
29.  S = bitshift(S, -2);
30.  C = bitshift(C, -2);
31.
32.  S = S + a;
33.  P = S + C;
34. end
```

C. Arithmetic Units Selection

To support the resource and performance optimization, we also utilize the Radix-4 Booth multiplication and CSA units. Radix-4 Booth is selected for multiplication process because it is faster than any other existing techniques. In Radix-4 Booth, digits of multiplier are grouped into blocks, in which each block consists of 3-bit. Grouping is started from the Least-Significant-Bit (LSB) and the first block only contains 2-bit of multiplier [11].

The selection on adder is also important since we are dealing with big number for RSA. There are several types of adder, such as Carry-Propagation Adder (CPA), Carry-Look-Ahead Adder (CLA), and Carry-Save Adder (CSA). We consider to use Carry-Save Adder (CSA) because it has an efficient architecture without any carry propagation signal. Since we need to compute 2 addition, we use 2 CSAs structure coupled as a double-CSAs to compute data in a clock cycle.

D. Montgomery Modular Multiplication

Purpose of this modular exponential algorithm is to repeat the Montgomery modular multiplication until the process is finished. We utilize the square-and-multiply method. In mathematical expression, representation of binary number can be stated as Eq. 18, with l is number of bit e . Hence, the modular exponential equation can be stated as Eq. 19. This algorithm can also be written in a pseudocode algorithm-4.

$$e = \sum_{i=0}^{l-1} e_i \cdot 2^i \quad (18)$$

$$p = a^e \bmod m = a^{\sum_{i=0}^{l-1} e_i \cdot 2^i} \equiv \prod_{i=0}^{l-1} a^{2^i} \quad (19)$$

Algorithm-4: Montgomery Modular Exponentiation

```
1. A = R mod M;
2. G = R^2 mod M;
3. X̄ = Mont(X, G);
4. for i = t to 0
5.   A = Mont(A, A);
6.   if(e(i) = 1)
7.     A = Mont(A, X̄)
8.   end
9. end
10. A = Mont(A, 1);
11. return A;
```

IV. PROPOSED HARDWARE ARCHITECTURE

A. RSA Datapath Architecture

A low-latency and resource-efficient scalable RSA datapath architecture is obtained by designing a generic and efficient architecture that can be easily modified based on the predefined parameters. The design is mainly about Montgomery modular multiplication and exponentiation. The basic modular multiplication design for small or large value computations are similar. The difference is about on the full-adder operation. Since we are dealing with a large value of data, the full-adder operation is divided into several smaller adders to perform small value addition. For example, if we want to accommodate addition for RSA 512-bit, we can utilize a small value adder, such as 16-bit adder. Hence, we need 32 clock cycles to perform the full computation.

The proposed datapath architecture for Montgomery modular multiplication is depicted in Fig. 2. We can see that there are 5 main parts in this datapath architecture that are named as stage-1, stage-2, stage-3, stage-4 and stage-5. Five pipelines are inserted to separate between stages and increase the data processing throughput. Each stage is discussed thoroughly in the next passage.

Fig. 2. Proposed modular multiplication datapath architecture

Stage-1 Architecture

In the Montgomery modular multiplication, there are two inputs that change their value dynamically (i.e., A and B) and one input that has a fixed value (i.e., modulus number N). For input A , there are four possibilities of value, namely $Aconst$, $Gconst$, 1 and output of Montgomery modular multiplication. Meanwhile, input B has two possibilities of value, namely input x and output of modular multiplication. To determine which signal that is going through multiplexer and becoming input A and B , a state and counter based controller is utilized. Equations to determine the values of $Aconst$ and $Gconst$ are presented at Eq. 20 and Eq. 21 respectively.

$$Aconst = R \bmod N \quad (20)$$

$$Gconst = R^2 \bmod N \quad (21)$$

with

$$R = 2^{(n+2)} \text{ and } N = \sum_{i=0}^n 2^i n_i, n_i \in \{0,1\}$$

Stage-2 Architecture

In stage-2, Radix-4 Booth encoding is performed. The encoding processes transform the binary multiplication number (i.e., x signal) into Booth structured multiplication number (i.e., $Xbooth$ signal) in a single cycle. Each 3-bit of x , except the LSB that only needs 2-bit of x (i.e., $x[1:0]$), is transformed into Radix-4 Booth configuration ($Xbooth$) based on Table 1. After transformation, possibility of the 3-bit x values only consists of '0', '-1', '1', '-2' and '2'. The proposed architecture for this Radix-4 Booth encoding in stage-2 is depicted in Fig. 3. It is mainly a Look-Up Table (LUT) with encoding rules.

TABLE I
RADIX-4 BOOTH ENCODING TO GENERATE PARTIAL PRODUCTS

Block	Partial Products
000	0
001	$1 \times \text{multiplicand}$
010	$1 \times \text{multiplicand}$
011	$2 \times \text{multiplicand}$
100	$-2 \times \text{multiplicand}$
101	$-1 \times \text{multiplicand}$
110	$-1 \times \text{multiplicand}$
111	0

Fig. 3. Radix-4 Booth encoding architecture

Stage-3 Architecture

In stage-3, the design produces $I1$ and $I2$. Generator q is responsible as controller for producing the valid value of $I2$ using Eq. 22 which can be simplified further to Eq. 23. The register q has 1-bit width. If $q = 1 \rightarrow I2 = N$ and otherwise ($q \neq 1$) $\rightarrow I2 = 0$. Meanwhile, register $I1$ is determined by $X(i)$. If $X(i) = 1 \rightarrow I1 = Y$ and otherwise ($X(i) \neq 1$) $\rightarrow I1 = 0$, with i is determined by the counter.

$$q = (\text{sum}[0] + \text{carry}[0] + (X[i] \times Y[0])) \bmod 2 \quad (22)$$

$$q = \text{sum}[0] \oplus \text{carry}[0] \oplus (X[i] \cdot Y[0]) \quad (23)$$

Stage-4 Architecture

Stage-4 is mainly about CSA and normalization processes. For CSA calculation, we need to perform 2 additions, namely $[S, C] = \text{CSA}(S, C, I1)$ and $[S, C] = \text{CSA}(S, C, I2)$. Normally, these operations are finished in 2 clock cycles by a single CSA. Since it can increase the system latency twice, we avoid it using a double-CSA structure to perform the calculation in a clock cycle. A double-CSA structure is obtained by integrating 2 CSA as depicted as Fig. 4. This structure fulfills the logical operations stated in Eq. 24 and Eq. 25.

$$S_3(i+1) = S_2(i+1) \oplus C_2(i+1) \oplus I_2(i+1) \quad (24)$$

$$C_3(i+2) = (S_2(i+1) \cdot C_2(i+1)) \vee (S_2(i+1) \cdot I_2(i+1)) \vee (C_2(i+1) \cdot I_2(i+1)) \quad (25)$$

Fig. 4. Architecture of the double-CSA – the coupled 2 CSAs

Fig. 5. Architecture of the normalization

Furthermore, *sum* and *carry* results from this addition process need to be normalized. For normalization process, we mainly utilize 2 multiplexers, 2 comparators and an adder. *Sum* and *carry* determine the first multiplexer control signals. Output of the first multiplexer is compared to a specific value for providing another control signals to the second multiplexer. Lastly, result of the second multiplexer and *sum* signals are summed using the adder. The normalization structure is depicted in Fig. 5.

Stage-5 Architecture

Fig. 6. Architecture of the 516-bit adder that is broken down into n number of 16-bit adders

In stage-5, we perform a big value addition. To efficiently implement the hardware, the big value addition is broken down into n number of small value addition. For example, in 516-bit addition case, we break down 516-bit adder into n -steps of 16-bit adders. In each step, addition is conducted using Boolean CSA and controlled by counter. In step-1, addition process is conducted as $sum[15:0] + carry[16] = X[15:0] + Y[15:0]$. The $carry[16]$ is delayed a clock cycle and becoming carry for stage-2. In step-2, $sum[31:16] + carry[31] = X[31:16] + Y[31:16] + carry[16]$. The process continues until step- n . The stage-5 architecture is depicted in Fig. 6. At last, the adder module produces value '1' on head-out signal if whole steps are finished.

B. RSA Controller Architecture

Montgomery algorithm is performed using modular multiplication and controlled as per modular exponentiation. For implementation, we utilize 3 registers to control RSA process, namely *state register*, *count register* and *count_key*. These following points elaborate each state condition of controller:

- State-0: It is reset condition, system is inactive, thus *state*, *count* and *count_key* = 0.
- State-1: It is start condition, system is activated. Modular multiplication calculates x_bar using equation $x_bar = mm1(X, G, N)$.
- State-2: It is activated after modular multiplication finished and indicated with *count* = 1. When *count* = 1, input A and B of modular multiplication are assigned by the output from previous modular multiplication. Here, modular multiplication is denoted with $A = Mont(A, A)$ with A is the output from previous modular multiplication.
- State-3: When head-out signal of modular multiplication is activated, state changes from 2 to 3. This change is indicated with *count* = 1. Here, value e is denoted as key and index i is denoted by *count_key*. At the start, *count_key* is assigned by bit number of key (encryption and decryption keys). When $e_i = 1$ and *count_key* $\neq 0$, then modular multiplication calculates $A = mm1(A, X_bar, N)$. If *head_out* = 1, then the process is finished and *count_key* is updated by *count_key* - 1. When $e_i = 1$ and *count_key* = 0 are fulfilled, then state transition from 3 to 4 is conducted and if otherwise, then state transition from 3 to 2 is conducted.

Fig. 7. Montgomery modular multiplication input controller architecture

- State-4: State-4 is reached if $count_key = 0$ or $e = 0$. Here, modular multiplication is denoted by $A = mm1(A, 1, N)$. If the computation is finished, $head_out$ signal is activated. Afterwards, $state$, $count$ and $count_key$ are reset to '0'.

C. Montgomery Modular Multiplication Controller

In Montgomery modular multiplication, there are 3 kinds of input, namely A , B , and modulus N . The modular multiplier inputs (i.e., A and B) controller architecture is depicted in Fig. 7. States of the controller are explained in the following points:

- If rst signal is inactive, then A and $B = 0$.
- If $start = 1$, then input $A = X$ and $B = Gconst$.
- In the $state = 2$, state transition is indicated as $count = 1$. In this condition, if $count_key = nKey$ (number of bit e), then A and $B = Aconst$. Meanwhile, if $count_key \neq nKey$, then A and $B =$ output of modular multiplication. If the state transition indicator $count \neq 1$, then A and B hold the static value.
- In $state = 3$, if state transition is indicated as $count = 1$, then values of $A = xbar$ and $B =$ output of modular multiplication. The $xbar$ register holds the output from computation $xbar = mont(X, Gconst, N)$. Result of this computation is saved in $xbar$ register until RSA system is stopped. Meanwhile, if $count \neq 1$, then A and B are unchanged.
- In $state = 4$, if state transition is indicated as $count = 1$,

then values of $A = 1$ and $B =$ output of modular multiplication. Meanwhile, if $count \neq 1$, then A and B are unchanged.

D. Interfacing Design

For implementation, FPGA is utilized because it is a configurable device that can retain the performance and physical security while providing the design flexibility. FPGA has a limited input and output pins. Thus, we need additional module to solve it. We choose to utilize Serial-to-Parallel (S/P) and Parallel-to-Serial (P/S) modules as interfaces in input and output respectively.

Serial-to-Parallel (S/P) Module

S/P interfacing module accumulates input data x that stream serially in a period of cycle. Within this period, S/P module delivers sum as output that has the same bit-width as RSA bit. The specific counter is utilized as controller of the S/P conversion. Since the architecture is designed for scalable purpose, predefined parameters are utilized and considered in architectural design. Each byte is passed through a delay component to provide $n+1$ bit of data. Fig. 8 shows the timing diagram needed for S/P conversion process. In input side, data are represented in a stream such as $CP1$, $CP2$, $CP3$, etc. After conversion, they are packed into single packet CP and passed to RSA core.

Fig. 8. Timing diagram for S/P interfacing datapath

Fig. 9. Timing diagram for P/S interfacing datapath

Parallel-to-Serial (P/S) Module

P/S interfacing module divides the RSA output into byte-by-byte each clock cycle serially. The specific counter is utilized as controller for this P/S conversion. Since the architecture is designed for scalable purpose, predefined parameters are utilized and considered in architectural design. The RSA output data is divided into byte-by-byte. Using a counter signal, each byte is selected each clock to be transmitted Fig. 9 shows the timing diagram needed for P/S conversion process. In input side, data is represented in a package *CC*. After conversion, they are divided into several data such as *CC1*, *CC2*, *CC3*, etc.

E. Top-Level Architecture

Core of the proposed RSA architecture design includes the datapath architecture (i.e., modular Montgomery datapath) and its modular exponential controller. As supportive modules, Serial-to-Parallel (S/P) and Parallel-to-Serial (P/S) modules are utilized. These modules are integrated into top-level design in which the architecture is depicted in Fig. 10.

V. RESULTS AND DISCUSSION

A. Verification

To verify the proposed RSA architecture, we design a test scenario. Firstly, the plaintext is encrypted to produce ciphertext that is propagated back to the system for decryption. Output of the decryption is compared with plaintext data to check the error rate.

The first test is a functional simulation using waveform. Result of this test is shown in Fig. 11. The waveform shows

that the proposed RSA architecture is working properly because the system can deliver and produce a correct result for each input accordingly. For the second test, we set-up an image encryption and decryption scenario and display the result from each process in VGA monitor. Results of this application are depicted in Fig. 12. Firstly, a landscape image data (Fig. 12a) is stored in a database. When the encryption process is activated, these data are fetched and processed in RSA core to produce the ciphertext that is displayed in Fig. 12b. Afterwards, the ciphertext is decrypted to obtain the plaintext again as displayed in Fig. 12c. By using a bare-eyes, we can roughly draw a conclusion that the decrypted image is the same with the original one. It proves that the proposed RSA architecture is working properly and able to reconstruct the original values from the ciphertext.

B. Benchmarks

Hardware vs Software Performance

We compare the computational time between software and hardware implementation. We compute the RSA using modeling software and FPGA. Comparisons are conducted in various size of RSA. Measurement results chart is presented in Fig. 13 and showing the proof that hardware implementation is far more superior than software which is predicted in the first place.

Optimized Radix-2 vs Optimized Radix-4 Montgomery

We design, compare and analyze the optimized Radix-2 Montgomery based RSA (OptR2RSA) and the optimized Radix-4 Montgomery based RSA (OptR4 RSA) to evaluate

Fig. 10. Top-level proposed RSA architecture

Fig. 11. Simulation results using waveform monitor

Fig. 12. Image encryption application – (a) original, (b) encrypted, (c) decrypted

their differences on FPGA implementation. The evaluation uses synthesis process in several sizes. The synthesis depends on the FPGA technology used as the target device, in which case, we only use FPGA Altera Cyclone II EP2C35F672C6[®]. The complete results are presented in Fig. 14 - Fig. 17. In Table 14 and Table 17, there are no data for 1024-bit design, because the target device can not accommodate the number of pins needed in the design.

Computational speed using OptR2RSA is faster than the OptR4RSA but it requires more latency especially for big sized design (i.e., ≥ 512 -bit). For the resource aspect, the OptR4RSA is larger than the OptR2RSA, because it reduces the iteration number by utilizing additional logic as trade-off. Meanwhile, there is swap position in throughput aspect around 128-bit size. In the smaller size (< 128 -bit), the

Fig. 13. Comparison chart on computational time vs size of RSA in respect of implementation

Fig. 14. Comparison chart on speed vs size of RSA

Fig. 16. Comparison chart on logic utilization vs size of RSA

OptR4RSA is faster than the OptR2RSA. But in the larger size, the OptR2RSA start to outperform the OptR4RSA slightly. These benchmarks deliver information that can be considered in choosing the optimal combination of latency and resource consumption for applications.

Benchmarks with Other Designs

We also perform benchmarks on the proposed design to the existing results out there for 512-bit RSA and they are presented in Table 2. We need to consider that the target device is not homogeneous. These benchmarks deliver information that can be considered further in observing the optimal design on latency, resource consumption and scalability.

Regarding the latency, the proposed OptR4 is the second-most efficient in the benchmarks with 227k clock cycles. Meanwhile, in term of resource consumption, the proposed OptR4 is the third-most efficient with 13k logic gate utilization. The proposed OptR4 can offer the scalability feature as well. From the benchmarks, we conclude that the proposed design OptR4 is competitive enough compared to other designs and it achieves an optimal design in respect of low-latency, resource-efficient and scalability for dealing with power and resource consumption issues.

Fig. 15. Comparison chart on latency vs size of RSA

Fig. 17. Comparison chart on throughput vs size of RSA

TABLE II
BENCHMARKS WITH OTHER DESIGNS FOR 512-BIT RSA

Design	Speed (MHz)	Latency (Cycles)	Logic Gate Utilization	Throughput (Kbps)	Target Device	Scalability
Yang '98 [12]	125	540k	74k	118	0.6 μ m	No
Su '99 [13]	100	510k	76k	100	0.6 μ m	No
Hsieh '99 [14]	125	6500k	4.5k	10.5	0.6 μ m	Yes
Leu '00 [15]	105	270k	78k	200	0.6 μ m	No
Yingli '01 [16]	40	180k	96k	113	0.6 μ m	Yes
Hong '03 [17]	300	530k	77k	289	0.6 μ m	No
Sun '03 [18]	220	405k	40k	276	0.35 μ m	Yes
Hong '06 [19]	333	260k	174k	656	0.35 μ m	No
OptR2RSA	89.61	424k	8k	108.14	FPGA	Yes
OptR4RSA	20	227k	13k	44.36	FPGA	Yes

VI. CONCLUSION

A low-latency and resource-efficient scalable RSA cryptoprocessor architecture is proposed. It is achieved successfully using two approaches, namely the optimization of Radix-4 Montgomery multiplication and scalable architecture. Overall, the proposed design is quite competitive compared to other existing designs and able to achieve an optimal design in respect of low-latency, resource-efficient and scalability for dealing with power and resource consumption issues. It only requires 227k cycles latency and consumes 13k logic gate utilization for 512-bit RSA. It can be utilized as solution for numerous kind of embedded systems which require high scalability and configurability within a short development time.

REFERENCES

- [1] S. S. Ghoreishi, M. A. Pourmina, H. Bozorgi and M. Dousti, "High speed RSA implementation based on modified Booth's technique and Montgomery's multiplication for FPGA platform," in *Proc. of 2009 2nd Inter. Conf. on Advances in Circuits, Electronics and Microelectronics*, pp. 86-93, October 2009.
- [2] T. Blum and C. Paar, "High-radix Montgomery modular exponentiation on reconfigurable hardware," in *IEEE Trans. on Computers*, vol. 50, no. 7, pp. 759-764, July 2001.
- [3] S. Zhou, Z. Wei, B. Wang, X. Zheng, C. Zhou and Q. Zhang, "Encryption method based on a new secret key algorithm for color images," in *AEÜ - Int. Journal of Electronics and Communications*, vol. 70, no. 1, pp. 1-7, January 2016.
- [4] W. Diffie and M. Hellman, "New directions in cryptography," in *IEEE Trans. on Information Theory*, vol. 22, no. 6, pp. 644-654, November 1976.
- [5] R. L. Rivest, A. Shamir, L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," in *Communications of the ACM*, vol. 21, no. 2, pp. 120-126, February 1978.
- [6] A. E. Cohen and K. K. Parhi, "Architecture optimizations for the RSA public key cryptosystem: a tutorial," in *IEEE Circuits and Systems Magazine*, vol. 11, no. 4, pp. 24-34, November 2011.
- [7] M. Chen, Y. Zhang and C. Lu, "Efficient architecture of variable size HEVC 2D-DCT for FPGA platforms," in *AEÜ - Int. Journal of Electronics and Communications*, vol. 73, pp. 1-8, March 2017. DOI: 10.1016/j.aeue.2016.12.024
- [8] R. V. W. Putra and T. Adiono, "Hybrid multi System-on-Chip architecture: a rapid development design for high-flexibility system," in *Proc. of 2016 Int. Conf. on Electronics, Information, and Communications*, pp. 1-4, January 2016.
- [9] R. V. W. Putra and T. Adiono, "VLSI architecture for configurable and low-complexity design of hard-decision Viterbi decoding algorithm," in *Journal of ICT Research and Applications*, vol. 10, no. 1, pp. 57-75, June 2016.
- [10] T. Blum, "Modular exponentiation reconfigurable hardware," Master Thesis, Worcester Polytechnic Institute, 8 April 1999.
- [11] ASIC Design for Signal Processing, Booth recoding. URL: <http://www.geoffknapge.com/fyp/>
- [12] C.-C. Yang, T.-S. Chang and C.-W. Jen, "A new RSA cryptosystem hardware design based on Montgomery's algorithm," in *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, vol. 45, no. 7, pp. 908-913, July 1998.
- [13] C.-Y. Su, S.-A. Hwang, P.-S. Chen and C.-W. Wu, "An improved Montgomery's algorithm for high-speed RSA public-key cryptosystem," in *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 2, pp. 280-284, June 1999.
- [14] Y.-H. Hsieh, "Design and implementation of an RSA encryption/decryption processor on IC smart card", Master Thesis, National Taiwan University, Taiwan, June 1999.
- [15] J. J. Leu and A. Y. Wu, "Design methodology for Booth-encoded Montgomery module design for RSA cryptosystem," in *Proc. of 2000 IEEE Int. Symp. on Circuits and Systems*, pp. 357-360, May 2000. DOI: 10.1109/ISCAS.2000.857438
- [16] F. Yingli and G. Zhiqiang, "A new RSA crypto-system hardware implementation based on high-radix Montgomery's algorithm," in *Proc. of 2001 4th Int. Conf. on ASIC*, pp. 348 -351, 2001.
- [17] J.-H. Hong and C.-W. Wu, "Cellular-array modular multiplier for fast RSA public-key cryptosystem based on modified Booth's algorithm," in *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 3, pp. 474-484, June 2003.
- [18] M.-C. Sun, C.-P. Su, C.-T. Huang and C.-W. Wu, "Design of a scalable RSA and ECC crypto-processor," in *Proc. of the Asia and South Pacific Design Automation Conf.*, pp. 495-498, January 2003.
- [19] J. H. Hong and B. Y. Tsai, "A fast bit-interleaving RSA cryptosystem based on Radix-4 cellular-array modular multiplier," in *Proc. of 2006 IEEE Asia Pacific Conf. on Circuits and Systems*, pp. 1446-1449, December 2006.

Trio Adiono received the B.Eng. on Electrical Engineering and M.Eng. in Microelectronics from Bandung Institute of Technology (ITB), Indonesia, in 1994 and 1996, respectively. He obtained his Ph.D. in VLSI Design from Tokyo Institute of Technology, Japan, in 2002. From 2002 to 2004, he was a research fellow of the Japan Society for the Promotion of Science (JSPS) in Tokyo Institute of Technology. In 2005, he was a visiting scholar at MESA+, Twente University, Netherlands. In 2006, he was a guest lecturer at Malaysian Institute of Aviation Technology, University of Malaysia. He received the "Second Japan Intellectual Property (IP) Award" in 2000 from Nikkei BP for his research on "Low Bit-rate Video Communication LSI Design". He also holds a Japanese Patent on "High Quality Video Compression System". Currently, he is a lecturer at the School of Electrical Engineering and Informatics ITB, and serves as a Head of the Microelectronics Center ITB and IC Design Laboratory ITB. He has co-founded several start-up companies such as BeyondMPEG Inc. (Japan) and Versatile Silicon Technology, Xirka Silicon Technology, Fusi Global Technology (Indonesia). He had several key committee positions and keynote speaker in national and international conferences, workshops, and training in microelectronics area. He currently serves as a chair of the IEEE SSCS Indonesia Chapter. His research interests include VLSI design, signal and image processing, smart card, electronics solution design and integration.

Iput Heri Kurniawan received B.Sc. in Electrical Engineering from School of Electrical Engineering and Informatics, Bandung Institute of Technology (ITB), Indonesia, in 2008 and M.Eng. in Electrical Engineering from Korea University, Seoul, South Korea, in 2013. From 2008 to 2011, he was with PT. Xirka Silicon Technology, Indonesia, as a research engineer in digital circuit design for OFDMA baseband. Since 2013 until now, he has been with Raon-Tech, Seongnam, South Korea, as a research engineer in digital circuit designs for digital front-ends, FM basebands, and haptic drivers. His research interests include low-power MIMO detectors.

Rachmad Vidya Wicaksana Putra received B.Sc. on Electrical Engineering in 2012 and M.Sc. on Microelectronics with a distinction Cum Laude in 2015, both from Bandung Institute of Technology (ITB), Indonesia. He received an Indonesia Endowment Fund for Education (IEFE/LPDP) Scholarship for his master and doctoral studies. He was an intern at Versatile Silicon Technology in 2010 for 2 months and an engineer at Fusi Global Technology for almost 2 years (2012-2014). He has been involved as teaching assistant at Electrical Engineering Department, School of Electrical Engineering and Informatics ITB (2012-2017). Currently, he is pursuing his PhD. His interests are on VLSI, SoC, computer architecture, IoT and visible light communications.