Reconfigurable Modular Arithmetic Logic Unit for High-Performance Public-Key Cryptosystems[⋆]

K. Sakiyama, N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede

Katholieke Universiteit Leuven, ESAT/COSIC, Kasteelpark Arenberg 10,
B-3001 Leuven-Heverlee, Belgium
{Kazuo.Sakiyama, Nele.Mentens, Lejla.Batina,
Bart.Preneel, Ingrid.Verbauwhede}@esat.kuleuven.be

Abstract. This paper presents a reconfigurable hardware architecture for Public-key cryptosystems. By changing the connections of coarse grain Carry-Save Adders (CSAs), the datapath provides a high performance for both RSA and Elliptic Curve Cryptography (ECC). In addition, we introduce another reconfigurability for the flip-flops in order to make the best of hardware resources. The results of FPGA implementation show that better performance is obtained for ECC on the same hardware platform.

Keywords: Public-Key Cryptography (PKC), RSA, Elliptic Curve Cryptography (ECC), FPGA implementation, Reconfigurable architecture.

1 Introduction

Diffie and Hellman introduced the idea of public-key cryptography [4] in the mid 70's. They showed that one can eliminate the need for prior agreement of a key in order to exchange some confidential data. Public-key cryptosystems also enable digital signatures. The best-known and most commonly used public-key cryptosystems are RSA and Elliptic Curve Cryptography (ECC). The RSA public-key cryptosystem is named after its inventors Rivest, Shamir and Adelman [11]. ECC, which was proposed in the mid 80's by Miller [9] and Koblitz [5], is based on a different algebraic structure. In the case of ECC, the group used is the group of points on an elliptic curve. It is important to point out that ECC offer equivalent security as RSA for much smaller key sizes. Other benefits include higher speed, lower power consumption and smaller certificates which is especially useful in constrained environments (smart cards, mobile phones, PDAs, *etc.*).

The security of the RSA cryptosystem is based on the difficulty of the RSA problem. It is still the most popular cryptosystem, especially for high-end devices that are typically used in e-commerce and Virtual Private Network (VPN)

[⋆] Kazuo Sakiyama, Nele Mentens and Lejla Batina are funded by FWO projects (G.0450.04, G.0141.03). This research has been also partially supported by the EU IST FP6 projects SCARD and ECRYPT.

servers. RSA, still the most popular public key cryptosystem, has at its root the modular exponentiation operation. Modular exponentiation consists of repeated modular multiplications, which is also the basic operation for ECC.

Our contribution deals with an FPGA implementation of RSA and ECC over a field of a prime characteristic. We used a reconfigurable datapath to achieve arbitrary precision in bits, hence easily bridging the gap between the bit-lengths for ECC from 160 bits to 2,048 bit long moduli for RSA. We use modular exponentiation based on Montgomery's method without any modular reduction which is also beneficial for side-channel attacks.

The results show that the proposed reconfigurable datapath is indeed a suitable solution for high-performance public-key cryptosystems such as RSA and ECC. Comparing the two with the same hardware resources and with corresponding bit-lengths that provide the similar security, we found that ECC-256p allows a better performance than RSA-2048. This research is of interest because due to a constant progress in cryptography and security applications an alternative solution for public-key services such as signatures, key-distribution *etc.* is needed.

This paper is organized as follows. Sect. 2 lists some relevant previous work. In Sect. 3 the details of our architecture are given. The main contribution of our work *i.e.* the reconfigurable datapath is explained in Sect. 4. The details of two cryptosystems that are implemented and the results are given in Sect. 5. Sect. 6 concludes the paper.

2 Related Work

This section reviews some of the most relevant previous work in hardware implementations for RSA and ECC. To consider both RSA and ECC on the same platform has only recently became more popular, since ECC have proven to be a mature technology. Some of the work is done on FPGAs and only very few implementations are presenting an ASIC implementation of ECC in the field of prime characteristic.

More recent work on hardware implementation of RSA includes the work by McIvor *et al.* [7]. They use Carry Save Adders (CSAs) to perform the large word length additions required for MMM. The obtained performance for one 1024 bit RSA decryption on the Xilinx Virtex2 board was 2.63 *msec*. The work of Crowe *et al.* [3] also proposed a unique architecture for RSA and ECC. A hardware optimized version of MMM is used for modular multiplication. The so-called dual processor could operate in parallel for ECC or in a pipelined series for RSA.

The contribution presented in [1] is combining a systolic array architecture with a Montgomery based RSA implementation, achieving the notion of scalability as introduced originally in the work by Tenca and Koç [12]. The optimal bound for Montgomery's parameter R is achieved which, with some savings in hardware, omits completely all reduction steps that are presumed to be vulnerable to side-channel attacks. By using the optimal scheduling for the architecture the authors have obtained a substantial speed-up for ECC when compared with RSA implementation on the same platform.

3 Modular Arithmetic Logic Unit

3.1 Datapath of the MALU

The proposed architecture is a Montgomery modular multiplier with digit-serial multiplications (Algorithm 1). Four-to-two (4-2) CSAs (Fig.1-a) are used in the hardware implementation because they are considered as one of the most optimal solutions for a multi-operand addition including Algorithm 1.

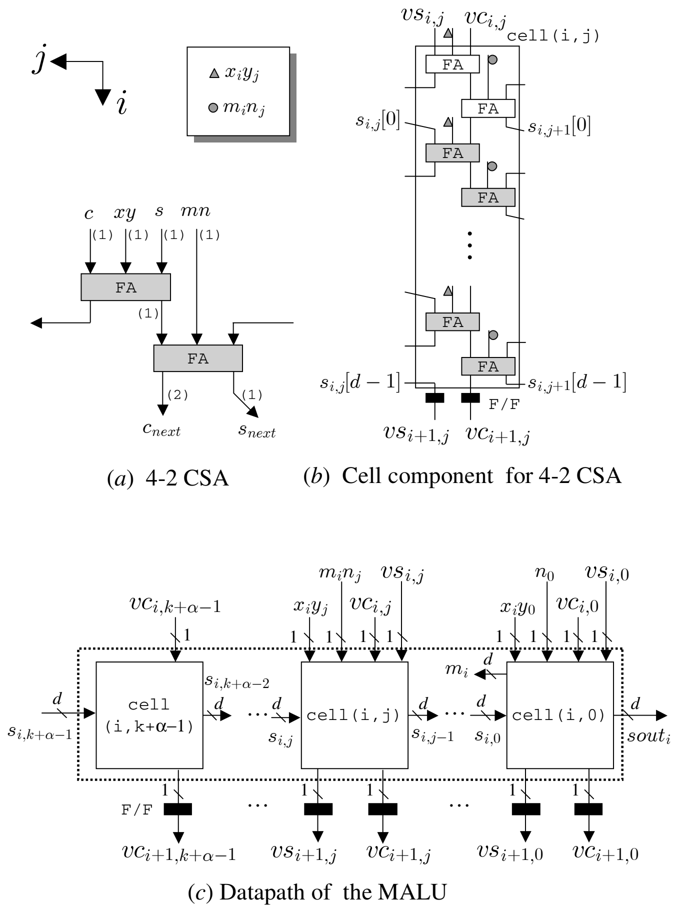

Fig. 1. Configuration of the MALU with 4-2 CSAs

The cell, a column of the datapath of the MALU uses d sets of 4-2 CSAs (Fig.1-b), *i.e.,* the inputs and outputs of the cell are presented in 2-bit CS-form during the operation. Therefore, the cell needs $2d$ sets of FAs. The critical path of the datapath is estimated with the critical path delay of the cell as follows.

$$T_{4-2CSAs} = 2dT_{FA}\,. \tag{1}$$

Here, we assumed that the delay for the sum and carry calculations are the same. The propagation of $s_{i,j}$ goes through d sets of the cell and uses two FAs in every cell. The right-most cell, `cell(i,0)` provides m_i vector for the rest of cells. As expressed in Eq.(2), the path for generating a bit of m_i only consists of 3-input XOR in the right-most cell.

$$\begin{aligned} m_i[0] &= vs_{i,0} \oplus vc_{i,0} \oplus x_i[0]y_0 \\ m_i[1] &= s_{i,0}[1] \oplus c0_{next}[0] \oplus x_i[1]y_0 \\ &\vdots \\ m_i[d-1] &= s_{i,0}[d-1] \oplus c0_{next}[d-1] \oplus x_i[d-1]y_0 \,. \end{aligned} \tag{2}$$

In the worst combination of the paths through the logic generating m_i, it takes:

$$dT_{m_i} + dT_{FA} \,. \tag{3}$$

This path delay is assumed to be equivalent to or shorter than $T_{4-2CSAs}$.

As can been seen from the hardware configuration and the delay calculations, the datapath of the MALU has an area and delay that can be adapted with the size of d. In this way the propagation can be tuned to the speed of the system. The proposed array is flexible regarding the size of d: it can be decided by exploring the best combination of performance and cost.

3.2 Functionality of the MALU

Before explaining the general case, the main functionality of the MALU is explained with the case that $d = 1$. In this configuration, each cell is composed of one 4-2 CSA (Fig1-a). The 4-2 CSA sums up the four-bit inputs xy, mn, s and c and outputs two bits in the redundant CS-form whose value is $2(c_{next}) + s_{next}$ where s and c are the virtual sum and carries. The bit multiplications xy and mn are main inputs for computing the bit level of Montgomery multiplication in Algorithm 1, *i.e.* $(T + xy + mn)/2$.

Simply thinking, a multiplication can be computed with $(k+\alpha)^2$ times 4-2 CSA operations if the multiplicand and multiplier have $(k+\alpha)$ bit. However, considering that there are no carry propagations in the j-direction shown on Fig.1, it is natural to allocate $(k+\alpha)$ sets of cells in the j-direction to take the speed merit. This CSA array is defined as the minimal configuration of our proposed MALU. The connection of the CSA arrays in the i-direction are determined by the bit weights of the CSA's outputs (numbers in parenthesis in Fig.1-a,b) and the division of the bit-level Montgomery algorithm (1-bit right-shift). The connection is latched with $(2k + 2\alpha - 1)$ sets of F/Fs for virtual carries.

The explanation of the MALU for a general d is given as follows. As illustrated in Fig.1-c, the introduced MALU with 4-2 CSAs has four types of input vectors, $X = (x_g \cdots x_1 x_0)_{2^d}$, $Y = (y_{k+1} \cdots y_1 y_0)_2$, $N = (n_{k-1} \cdots n_1 n_0)_2$, and $S = (s_{h,k+\alpha-1} \cdots s_{1,k+\alpha-1} s_{0,k+\alpha-1})_{2^d}$ where $g = \lceil (k+1)/d \rceil$ and $h = \lceil k/d \rceil$. Here, X is the multiplier, Y is the multiplicands, and N is the modulus. The

Algorithm 1. Algorithm for d-digit serial Montgomery Modular Multiplication over GF(p) without final subtraction

```
Require: $N = (n_{k-1} \cdots n_1 n_0)_2$, $X = (x_k \cdots x_1 x_0)_2$, $Y = (y_k \cdots y_1 y_0)_2$
        with $x, y \in [0, 2N-1]$, $R = 2^{k+2}$, $gcd(N, 2) = 1$
Ensure: $T = XYR^{-1} \bmod 2N$
   1. $T \leftarrow 0$
   2. for $i$ from $0$ to $k+1$ do
   3.    $m_i \leftarrow t_0 \oplus x_i y_0$
   4.    $T \leftarrow (T + x_i y + m_i N)/2$      // addition stage
   5.    $m_{i+1} \leftarrow t_0 \oplus x_{i+1} y_0$
   6.    $T \leftarrow (T + x_{i+1} y + m_{i+1} N)/2$      // addition stage
                  ⋮
1+2d.    $m_{i+d-1} \leftarrow t_0 \oplus x_{i+d-1} y_0$
2+2d.    $T \leftarrow (T + x_{i+d-1} y + m_{i+d-1} N)/2$      // addition stage
3+2d. end for
4+2d. Return $T$
```

augend vector S is provided to the MALU by d bits in every cycle and eventually added to the result of the modular multiplication of X and Y (modulo N). The intermediate results are stored in $VS = (vs_{i,k+\alpha-1} \cdots vs_{i,1} vs_{i,0})_2$ and $VC = (vc_{i,k+\alpha-1} \cdots vc_{i,1} vc_{i,0})_2$. They are reset to zero when a modular multiplication starts to execute ($i = 0$). After finishing a Montgomery multiplication, the result is output from the right-most cell by d bits in every cycle as $Sout = (sout_g \cdots sout_1 sout_0)_{2^d}$.

The MALU has two independent stages for GF(p) operation. One is the Carry-Save(CS)-stage that implements the Montgomery algorithm in a CS-form. Another converts the CS-form integer into a normal integer by propagating carries, namely the Carry-Propagate(CP)-stage. Moreover the CP-stage is capable of adding/subtracting S to/from the result of the CS-stage. When subtracting S from XY, we use the 2's complement of S. More precisely, each bit of S is inverted in setting a register for S and $2N + 1$ is provided from inputs of mn at the first cycle of the CP stage. For reducing the hardware cost and the critical path delay, the CP calculations are executed in the same datapath of the MALU as the CS-calculations. The operation of the MALU is explained in Eq.(4).

$$\mathrm{MALU}_N(XR, YR, SR) = (XY \pm S)R \bmod N . \qquad (4)$$

Here R is selected as $R = 2^{k+\alpha}$ where k is the bit-length of the secret key and α is a value determined so that the final reductions can be avoided. In our case, we chose $\alpha = 4$. The details are explained by the following Lemma [2].

Lemma 1. *If the Montgomery parameter R satisfies the following inequality $R > 16N$, then for inputs $X, Y < 4N$ and $S < 2N$ the result T will satisfy: $T < 4N$ (as required).*

Proof: The Montgomery multiplication as implemented in the MALU calculates the following:

$$\begin{aligned} T = \mathrm{MALU}_N(X, Y, S) &= \frac{XY + MN}{R} + S \\ &< \frac{4N \cdot 4N}{16N} + N + 2N \leq 4N\,. \end{aligned} \tag{5}$$

While the reduction step was needed in the original notation of Montgomery's algorithm, we use a method which does not require the reduction. For convenience of repeating usage of Eq.(4), the so-called Montgomery form is applied because the output is in the Montgomery form as well. The latency to calculate a MALU_N needs $2 \cdot \lceil (k + \alpha)/d \rceil$ cycles in total.

4 Reconfigurable Datapath

In order to obtain high-performance modular operations, we should allocate $k + \alpha - 1$ cells for the datapath of the MALU. For instance, the case of ECC-256p and RSA-2048 need 260 and 2,052 cells, respectively. Since we target a platform which supports both ECC-256p and RSA-2048, we introduce a coarse grain datapath of the $\mathrm{MALU}_{260 \times 1}$ ($k = 256$ and $d = 1$) and allocate its clones. For the general case ($K \times D$ sets of $\mathrm{MALU}_{k \times d}$) the block diagram of the reconfigurable datapath is illustrated in Fig.2.

The datapath can be configured by changing the interconnection of the $\mathrm{MALU}_{d \times k}$ that is determined by the three multiplexors (Fig.3) and two- or three-bit registers for selecting them. The *sel1* and *sel2* are used for configuring the datapath, and the *sel3* is used for configuring the flip-flops. Those multiplexors and flip-flops for the configuration are considered as the area overhead (denoted as AO) introduced by the reconfigurable feature. It is approximately estimated as follows:

$$\begin{aligned} AO_{base} &= DK(A_{MUX(sel1)} + A_{MUX(sel2)} + A_{MUX(sel3)} + A_{FF(config)}) \\ &= \{(6d + k)DK - 2dK\} \cdot A_{2-1MUX} + \{3DK - K\} \cdot A_{FF}\,. \end{aligned} \tag{6}$$

Here, we ignored the area increase caused by the complexity of the wiring. In addition to the AO_{base}, some more flip-flops are not used depending on the configuration. As an example, we consider the case of using eight clones of $\mathrm{MALU}_{260 \times 1}$ ($K = 2$ and $D = 4$). When supporting RSA-2048, the datapath is configured as $\mathrm{MALU}_{2080 \times 1}$. In this configuration, the horizontal connections of the MALUs are not re-timed by flip-flops for S and R (REG_S and REG_R). Therefore, the total area overhead becomes as follows:

$$\begin{aligned} AO_{RSA-2048} &= AO_{base} + 14A_{FF} \\ &= 11,448A_{2-1MUX} + 36A_{FF}\,. \end{aligned} \tag{7}$$

Likewise, for RSA-2048 with CRT (Chinese Remainder Theorem) [10], the datapath is configured to have two sets of $\mathrm{MALU}_{1040 \times 1}$. Therefore, the area overhead for RSA-2048 with CRT is estimated as follows:

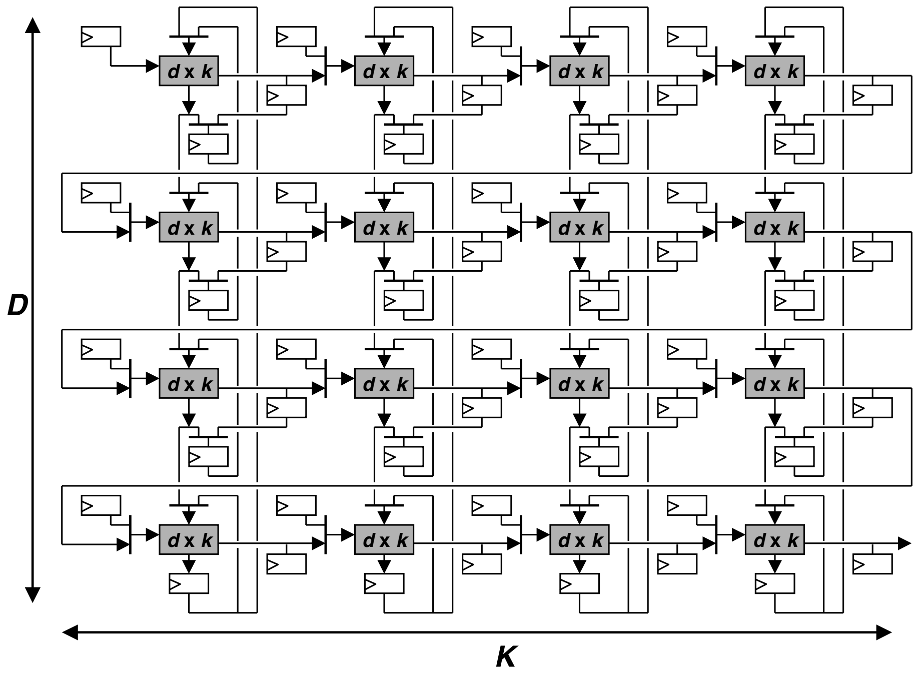

Fig. 2. Reconfigurable Datapath using $D \times K$ sets of MALU$_{d\times k}$

$$\begin{aligned} AO_{RSA-2048(CRT)} &= AO_{base} + 12A_{FF} \\ &= 11,448A_{2-1MUX} + 34A_{FF} . \end{aligned} \tag{8}$$

For ECC-256p, we configure the datapath so that two sets of MALU$_{260\times4}$ can be used in parallel. This configuration uses vertical series of MALU$_{260\times1}$. The intermediate values, the virtual carry and sum, are stored in the flip-flops for VS and VC only at the bottom of Fig.2. In this configuration, only one-fourth of of REG$_X$, REG$_Y$ and REG$_N$ are used in each MALU$_{260\times1}$. Therefore, the total area of the overhead becomes as follows:

$$\begin{aligned} AO_{ECC-256p} &= AO_{base} + (260 \times 12 + 260 \times 24 \times 3/4)A_{FF} \\ &= 11,448A_{2-1MUX} + 7,822A_{FF} . \end{aligned} \tag{9}$$

In the case of the RSA configuration, we can utilize the flip-flops with almost no waste, while the configuration of ECC can not use them effectively. In order to exploit the unused flip-flops, we introduce another reconfigurability. Different from RSA, ECC needs to store the intermediate variables during point operations. For the purpose, two sets of 14 words of 260-bit RAM (28×260-bit RAM) can be configured with the unused flip-flops. In this case, the area overhead becomes as follows:

$$AO_{ECC-256p} = 11,448A_{2-1MUX} + 542A_{FF} . \tag{10}$$

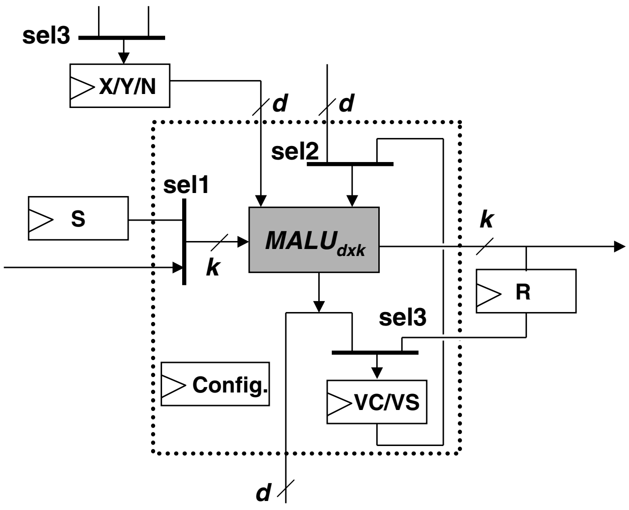

Fig. 3. Flip-Flops and and Multiplexors of MALU$_{d\times k}$

Thus, we can make the best use of the hardware resources also for the ECC configuration. For the critical path delay for each configuration, we have different delays as follows:

$$\begin{aligned} T_{RSA-2048} = T_{RSA-2048(CRT)} &= 2T_{FA} + 2T_{2-1MUX} + T_{FF} + T_{wiring} \\ T_{ECC-256p} &= 8T_{FA} + 4T_{2-1MUX} + T_{FF} + 4T_{wiring} \, . \end{aligned} \tag{11}$$

We assumed that the wiring delay in the critical path of ECC is four times longer than that of RSA. As seen from the Eq.(11), the critical path delay of ECC is about four times longer than RSA. Therefore, we need to assume a circumstance where we can use two different clock frequency, *e.g.*, providing a divided clock in the ECC configuration, in order to facilitate a high performance for both configurations.

5 Performance Comparison

5.1 RSA

The main operation in the RSA algorithm is a modular exponentiation [11]. The two most straightforward algorithms to implement this are given in Algorithm 2, where G is a finite abelian group and e is a positive integer. The basic operations in both algorithms are multiplications and squarings. To be able to use the same datapath for both operations and also for side-channel issues [6] the squarings are not performed on a dedicated squarer, but on the multiplier. Taking into account an expected value of $\frac{n}{2}$ ones in e, the total number of multiplications in both algorithms is $\frac{3n}{2}$. In the left-to-right algorithm the multiplications have to be performed consecutively requiring one memory location for intermediate values.

Algorithm 2. Algorithms for left-to-right and right-to-left binary exponentiation

```
Require: $g \in G$, $e = (e_{n-1}e_{n-2} \cdots e_1e_0)_2$
Ensure: $g^e$
 1: $A \leftarrow 1$
 2: for $i$ from $n-1$ downto 0 do
 3:     $A \leftarrow A \cdot A$
 4:     if $e_i = 1$ then
 5:         $A \leftarrow A \cdot g$
 6:     end if
 7: end for
 8: Return $A$
```

```
Require: $g \in G$, $e = (1\ e_{n-2} \cdots e_1e_0)_2$
Ensure: $g^e$
    $A \leftarrow 1$, $S \leftarrow g$
    for $i$ from 0 to $n-1$ do
        if $e_i = 1$ then
            $A \leftarrow A \cdot S$
        end if
        $S \leftarrow S \cdot S$
    end for
    Return $A$
```

In the right-to-left algorithm the multiplications can be parallelized, which doubles the speed. However, the right-to-left algorithm uses two memory locations for intermediate values.

5.2 ECC

In ECC, the equivalent operation of the modular exponentiation in RSA is a point multiplication, which multiplies a point on the elliptic curve with a scalar, resulting in another point on the curve. Similar to the left-to-right and right-to-left binary algorithms for modular exponentiation, a point multiplication can be performed using Algorithm 3 [8], where P is a point on the elliptic curve and k is a positive integer. The point at infinity $\mathcal{O}$ is the identity element for elliptic curve operations. Similar to modular the modular exponentiation algorithms, the left-to-right algorithm will be used when the storage of intermediate values is the bottleneck, while the right-to-left algorithm will be used for higher speed when the datapath allows parallelism.

The point operations in Algorithm 3 are point additions and point doublings. In our case a point addition and a point doubling respectively consist of 14 and 21 multiply/add operations by the MALU in the underlying finite field. Therefore the total number of multiplications for point multiplication is estimated as $\frac{49l}{2}$.

Algorithm 3. Algorithm for left-to-right and right-to-left binary point multiplication

```
Require: $P = (x, y)$, $k = (k_{l-1}k_{l-2} \cdots k_0)_2$
Ensure: $Q = (x', y') = kP$
 1: $Q \leftarrow \mathcal{O}$
 2: for $i$ from $l-1$ downto 0 do
 3:     $Q \leftarrow 2Q$
 4:     if $k_i = 1$ then
 5:         $Q \leftarrow Q + P$
 6:     end if
 7: end for
 8: Return $Q$
```

```
Require: $P = (x, y)$, $k = (1\ k_{l-2} \cdots k_0)_2$
Ensure: $Q = (x', y') = kP$
    $Q \leftarrow \mathcal{O}, S \leftarrow P$
    for $i$ from 0 to $l-1$ do
        if $k_i = 1$ then
            $Q \leftarrow Q + S$
        end if
        $S \leftarrow 2S$
    end for
    Return $Q$
```

However, as we allocate two MALUs for the ECC case, the number becomes $21l$ by processing point additions and doublings in parallel.

5.3 Performance Estimation for RSA and ECC from FPGA Implementation

We implemented the proposed datapath on a Xilinx FPGA (Spartan 3). The place-and-route result is shown in Table 1. The design is set as the ECC configuration and the critical path delay for the RSA configuration is estimated by the result of STA (Static Timing Analysis).

Table 1. Implementation result of the proposed datapath

Target Platform	Number of Slices	Critical Path [$nsec$]	
		RSA config.	ECC config.
xc3s5000	27,597	10.6	25.3

Based on the required number of multiplications for RSA and ECC, we estimate the performance and compare them with each other. The result is summarized in Table 2. For the ECC case, the latency of point multiplication is used for the performance, and the latency of modular exponentiation is estimated for the RSA case. As seen from the result, the performance of modular exponentiation for RSA-2048 is slower than ECC by a factor of 7 approximately. Even when applying CRT for RSA-2048, the performance is almost half of that of ECC-256p. Moreover, ECC-256p offers stronger security than RSA-2048.

Table 2. Performance comparison of RSA-2048, RSA-2048 with CRT, and ECC-256p

Type of PKC	MALU Config.	Max. Clock Freq. [MHz]	Performance [$msec$]
RSA-2048	$\text{MALU}_{2048\times1}$	95	133.1
RSA-2048(CRT)	$2\times\ \text{MALU}_{1024\times1}$	95	33.3
ECC-256p	$2\times\ \text{MALU}_{260\times4}$	40	17.7

6 Conclusions

We presented a new reconfigurable datapath that enables modular operations for different bit-widths. In addition, the flip-flops are also reconfigured depending on the configuration in order to use hardware resources effectively. The estimated performance based on an FPGA implementation is shown as a case study of RSA-2048 (with and without CRT) and ECC-256p. The results prove that our proposed datapath is suitable for a high-performance cryptosystem supporting both RSA and ECC over GF(p). Especially in our case, ECC-256p shows a better performance than RSA-2048 on the same amount of hardware resources.

References

1. L. Batina, G. Bruin-Muurling, and S.B. Örs. Flexible hardware design for RSA and elliptic curve cryptosystems. In T. Okamoto, editor, *In Topics in Cryptology - CT-RSA - The Cryptographers' Track at the RSA Conference*, volume 2964 of *Lecture Notes in Computer Science*, pages 250–263. Springer-Verlag, 2004.
2. L. Batina and G. Muurling. Montgomery in Practice: How to Do It More Efficiently in Hardware. In B. Preneel, editor, *In Topics in Cryptology - CT-RSA - The Cryptographers' Track at the RSA Conference*, number 2271 in Lecture Notes in Computer Science, pages 40–52, San Jose, USA, February 18-22 2002. Springer-Verlag.
3. F. Crowe, A. Daly, and W. Marnane. A Scalable Dual Mode Arithmetic Unit for Public Key Cryptosystems. In *Proc. IEEE International Conference Conference on Information Technology - ITCC'05*, pages 568–573, Las Vegas, 2005.
4. W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.
5. N. Koblitz. Elliptic curve cryptosystem. *Math. Comp.*, 48:203–209, 1987.
6. P. Kocher, J. Jaffe, and B. Jun. Introduction to differential power analysis and related attacks. `http://www.cryptography.com/dpa/technical`, 1998.
7. C. McIvor, M. McLoone, J. McCanny, A. Daly, and W. Marnane. Fast Montgomery Modular Multiplication and RSA Cryptographic Processor Architectures. In *Proceedings of 37th Annual Asilomar Conference on Signals, Systems and Computers*, pages 379–384, November 2003.
8. A.J. Menezes. *Elliptic Curve Public Key Cryptosystems.* Kluwer Academic Publishers, 1993.
9. V. Miller. Uses of elliptic curves in cryptography. In H. C. Williams, editor, *Advances in Cryptology: Proceedings of CRYPTO'85*, number 218 in Lecture Notes in Computer Science, pages 417–426. Springer-Verlag, 1985.
10. J.-J. Quisquater and C. Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronics Letters*, 18:905–907, October 1982.
11. R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
12. A.F. Tenca and Ç.K. Koç. A scalable architecture for Montgomery multiplication. In Ç.K. Koç and C. Paar, editors, *Proceedings of 1st International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, number 1717 in Lecture Notes in Computer Science, pages 94–108, Worcester, Massachusetts, USA, August 12-13 1999. Springer-Verlag.