FPGA Design for Algebraic Tori-Based Public-Key Cryptography

Junfeng Fan, Lejla Batina, Kazuo Sakiyama and Ingrid Verbauwhede
Katholieke Universiteit Leuven, ESAT/SCD-COSIC,
Kasteelpark Arenberg 10
B-3001 Leuven-Heverlee, Belgium
{Junfeng.Fan, Lejla.Batina, Kazuo.Sakiyama, Ingrid.Verbauwhede}@esat.kuleuven.be

Abstract

Algebraic torus-based cryptosystems are an alternative for Public-Key Cryptography (PKC). It maintains the security of a larger group while the actual computations are performed in a subgroup. Compared with RSA for the same security level, it allows faster exponentiation and much shorter bandwidth for the transmitted data. In this work we implement a torus-based cryptosystem, the so-called CEILIDH, on a multicore platform with an FPGA. This platform consists of a Xilinx MicroBlaze core and a multicore coprocessor. The platform supports CEILIDH, RSA and ECC over prime fields. The results show that one 170-bit torus T_6 exponentiation requires 20 ms, which is 5 times faster than 1024-bit RSA implementation on the same platform.

1 Introduction

Diffie and Hellman introduced the idea of Public-Key Cryptography (PKC) [3] in the mid 70's. Their breakthrough showed that one can eliminate the need for prior agreement of a key in order to exchange some confidential data. One important application of Public-Key services are digital signatures. The best-known and most commonly used public-key cryptosystems are based on factoring (RSA) and on the discrete logarithm problem in a large prime field (Diffie-Hellman, ElGamal, Schnorr, DSA) [7]. Elliptic Curve Cryptography (ECC), which was proposed in the mid 80's by Miller [8] and Koblitz [6], is based on a different algebraic structure. In the case of ECC, instead of integers modulo n another group is used *i.e.*, the group of points on an elliptic curve. It is important to point out that ECC offers 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.*).

Algebraic torus-based cryptosystems are another alternative for PKC. Torus-based cryptography assumes using algebraic torus to construct a group on which the discrete logarithm problem is defined. This idea was first introduced by Rubin and Silverberg in 2003 [10] and they proposed the name of CEILIDH. The idea behind it was to obtain the security of $\mathbb{F}_{p^6}$, while data to be transmitted are compressed with a factor 3 and the underlying arithmetic is performed in a subgroup *e.g.* $\mathbb{F}_p$. Their benefits are that they allow for shorter transmissions which is of interest for embedded applications.

So, torus-based cryptography gives a possibility to work in a subgroup, while maintaining the security of a bigger group. More precisely, Rubin and Silverberg showed that the factor $\frac{n}{\varphi(n)}$ can be achieved for compression. Here, $\varphi(n)$ is the Euler's totient function that is defined to be the number of positive integers less than or equal to n that are coprime to n. For example, $\varphi(6) = 2$ *i.e.* the numbers 1 and 5 are coprime to 6. They introduced a new public-key cryptosystem CEILIDH [10] that is based on torus T_6. Hence, in this case we get the compression of $\frac{6}{\varphi(6)} = 3$. The advantage of the torus when compared for RSA for example, lies in the compression factor, which allows one to use keys of length three times smaller than those for RSA. Another advantage is that the basic arithmetic behind is performed in a prime field, where the prime is 160-170 bits long which is a typical case of ECC. Thus, tori and ECC can be easily implemented using the same arithmetic unit.

In this paper we consider efficient implementations of CEILIDH. The work of Granger *et al.* [5] was first to introduce efficient arithmetic on T_6. They implemented T_6 on a PC and they concluded that CEILIDH is not much slower than XTR, which is another PK cryptosystem using the same idea of keeping the security of $\mathbb{F}_{p^6}$ while transmitting only two elements of $\mathbb{F}_p$. In this work we propose a flexible platform architecture which supports CEILIDH, RSA and ECC over prime fields. A hierarchical design method is used, and in this manner we show that all three cryptosystems can be efficiently supported by the same hardware platform. To our knowledge, this work presents the first architecture for efficient implementation of CEILIDH, ECC and RSA together.

The rest of the paper is organized as follows. Section 2 gives a brief introduction on the mathematical background of torus. Section 3 describes the multicore platform and our implementation considerations. We show the implementation results in section 4 and conclude the paper and give some future work in section 5.

2 Mathematical Background

The field $\mathbb{F}_{p^6}$ can be viewed as an extension field of degree 6 over $\mathbb{F}_p$. More precisely, $\mathbb{F}_{p^6} = \mathbb{F}_p[x]/(f(x))$, where $deg(f) = 6$. So, the elements of $\mathbb{F}_{p^6}$ are six-tuples of elements from $\mathbb{F}_p$. There are other representations possible of the field $\mathbb{F}_{p^6}$ *e.g.* using subfields with exponents 2 and 3. In principle, it is possible to choose a representation in which field arithmetic can be performed more efficiently and then map an arbitrary element from torus T_6 to the corresponding one in any other representation. An algebraic torus $T_6(\mathbb{F}_p)$ is defined in such a way that over $\mathbb{F}_{p^6}$, this structure can be represented by a pair of elements from $\mathbb{F}_p$. This means that one can achieve the security of $\mathbb{F}_{p^6}$, while transmitting only two elements of $\mathbb{F}_p$. Those maps between various representations as well as the representations are given in [5]. The choice of representation is usually related to the implementation platform. This is where our considerations alter from those in [5].

The first representation denoted as F_1 is the basic one *i.e.* viewing $\mathbb{F}_{p^6}$ as an extension field of $\mathbb{F}_p$ of degree 6, so $F_1 = \mathbb{F}_{p^6} = \mathbb{F}_p[x]/(f(x))$, where $deg(f) = 6$. We consider only this representation for our implementation but for a complete cryptosystem also the mappings between different representations have to be implemented.

2.1 Overall Structure of Operations

In Fig. 1 we give the complete overview of all operations included in the torus arithmetic. The mappings τ and ρ are isomorphisms and so are the corresponding inverse mappings τ^{-1}, ψ. Those mappings are used to change from one representation to the other *e.g.* to map an element from T_6 to the one from F_2 where F_2 is a quadratic extension of $\mathbb{F}_{p^3}$, so $F_2 = \mathbb{F}_{p^3}[y]/g(y)$ and $deg(g) = 2$. In Fig. 1 we also see the relations among all those operations. Further on, it is denoted which subfields are used for representations F_1 and F_2.

2.2 The Representation F_1

Here we focus on the representation F_1 as the one in which we perform the required arithmetic that includes ex-

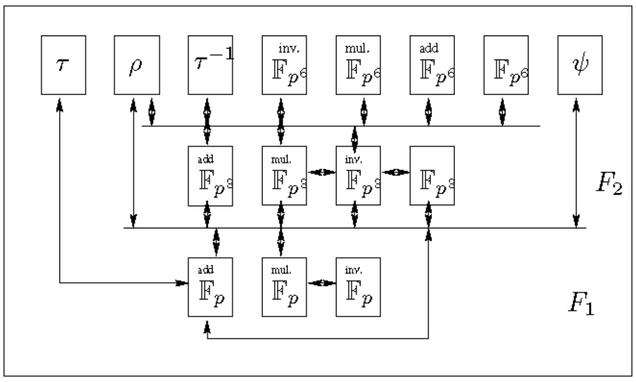


Figure 1. An overview of the $T_6(\mathbb{F}_p)$ operations.

ponentiation in this group. $F_1 = \mathbb{F}_{p^6} = \mathbb{F}_p[x]/(f(x))$ and let $p \equiv 2 \bmod 9$ (or $p \equiv 5 \bmod 9$) and $f(x) = \frac{x^9-1}{x^3-1} = x^6+x^3+1$ is an irreducible polynomial with a root z. Then, $z^6 = -z^3-1$ and each element from F_1 can be represented in the basis $\{1, z, z^2, z^3, z^4, z^5\}$. Hence, an arbitrary element from this group is denoted as $A(z) = \sum_{i=0}^{i=5} a_i z^i$. In order to perform exponentiation in this group we will first describe the basic operations in this field *i.e.* addition and multiplication. We denote multiplication/squarings and additions/subtractions in $\mathbb{F}_p$ with M and A respectively.

2.2.1 Addition in $\mathbb{F}_{p^6}$

Take two elements from F_1, $A(z) = \sum_{i=0}^{i=5} a_i z^i$ and $B(z) = \sum_{i=0}^{i=5} b_i z^i$. Then the sum is defined as: $C(z) = \sum_{i=0}^{i=5} c_i z^i = \sum_{i=0}^{i=5}(a_i + b_i)z^i$ and it requires 6 additions in $\mathbb{F}_p$.

2.2.2 Multiplication in $\mathbb{F}_{p^6}$

Multiplication of two polynomials of degree five can be performed in $18M$ plus many additions [1]. We explain that in more detail as it was also elaborated in [11]. Let $A(z) = \sum_{i=0}^{i=5} a_i z^i$ and $B(z) = \sum_{i=0}^{i=5} b_i z^i$ be two fifth degree polynomials to be multiplied. Write $A = A_0 + A_1 z^3$ and $B = B_0 + B_1 z^3$ where A_i, B_i for $i = 0, 1$ are second degree polynomials. Then $A \cdot B = A_0 B_0 + (A_0 B_1 + A_1 B_0)z^3 + A_1 B_1 z^6$ where one can precompute the values $C_0 = A_0 B_0$, $C_1 = A_1 B_1$ and $C_2 = (A_0 - A_1)(B_0 - B_1)$. This results in $A \cdot B = C_0 + (C_0 + C_1 - C_2)z^3 + C_1 z^6$. Let $A_0 = a_0 + a_1 x + a_2 x^2$ and $B_0 = b_0 + b_1 x + b_2 x^2$, where $a_i, b_i \in \mathbb{F}_p$ for $i = 0, 1, 2$; then for C_0 we get

$C_0 = a_0b_0 + (a_0b_1 + a_1b_0)x + (a_2b_0 + a_1b_1 + a_0b_2)x^2 + (a_2b_1 + a_1b_2)x^3 + a_2b_2x^4$. If the following values $c_0 = a_0b_0$, $c_1 = a_1b_1$, $c_2 = a_2b_2$, $c_3 = (a_0 - a_1)(b_0 - b_1)$, $c_4 = (a_0 - a_2)(b_0 - b_2)$ and $c_5 = (a_1 - a_2)(b_1 - b_2)$ are precalculated we finally get: $C_0 = c_0 + (c_0 + c_1 - c_3)x + (c_0 + c_1 + c_2 - c_4)x^2 + (c_1 + c_2 - c_5)x^3 + c_2x^4$. It follows from above that each C_i requires $6M + 11A$ so the total number of multiplications adds to $18M$. The result $A \cdot B$ still has to be reduced modulo an irreducible polynomial in $\mathbb{F}_{p^6}$, which adds a few more additions to the total number of multiplications. According to [5] this all adds to $18M + 60A$ as the cost for one multiplication in F_1 in basis $\{z, z^2, z^3, z^4, z^5, z^6\}$.

2.3 Operations in $\mathbb{F}_p$

To summarize, we need the following arithmetic operations in $\mathbb{F}_p$: addition, subtraction, multiplication and inversion. Exponentiation is performed via repeated multiplications. We use Montgomery's modular multiplication algorithm to perform modular multiplications. The algorithm of Montgomery is the best manner to avoid the time-consuming trial division in modular multiplications [9]. Alg. 1 shows a high radix Montgomery algorithm called FIOS (Finely Integrated Operand Scanning), which is suitable for a software implementation on a w-bit datapath.

Algorithm 1 Radix-2^w n-bit Montgomery modular multiplication (FIOS). [2]

Input: integers $P = (p_{s-1}, ..., p_0)_r$, $X = (x_{s-1}, ..., x_0)_r$, $Y = (y_{s-1}, ..., y_0)_r$, where $0 \leq X, Y < P$, $r = 2^w$, $s = \lceil \frac{n}{w} \rceil$, $R = r^s$ with $gcd(p, r) = 1$ and $p' = -P^{-1} \text{mod } r$.
Output: $X \cdot Y \cdot R^{-1} \bmod p$

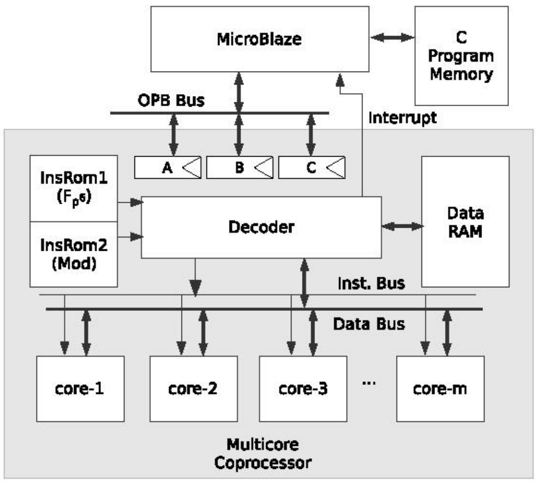
```
1: $z = (z_{s-1}, ..., z_0)_r \leftarrow 0$
2: for $i = 0$ to $s - 1$ do
3:     $T \leftarrow (z_0 + x_0 \cdot y_i) \cdot p' \bmod r$
4:     $Z \leftarrow (Z + X \cdot y_i + P \cdot T)/r$
5: end for
6: if $Z > p$ then
7:     $Z \leftarrow Z - P$
8: end if
9: return $Z$
```

3 Implementation

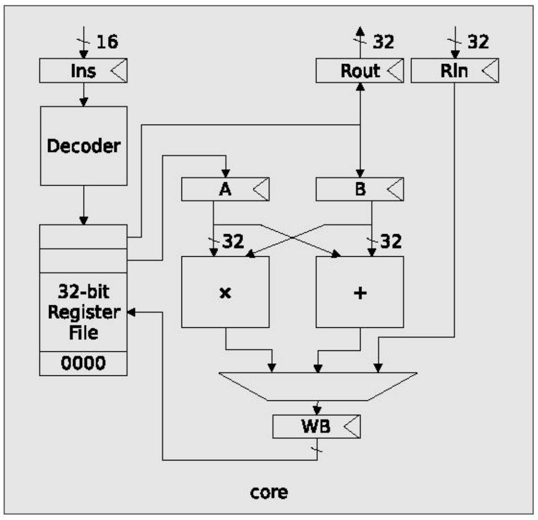
3.1 Platform Architecture

We implement the torus-based cryptosystem on a multi-core platform. This platform has multiple data-paths and is completely programmable, thus different algorithms can be efficiently implemented on it. In Fig. 2 the block diagram of the platform is shown. It consists of a MicroBlaze processor and a multicore coprocessor. MicroBlaze is a synthesizable core offered by Xilinx, and is used here as a controller. The coprocessor is the workhorse of the implementation. Multiple cores of the coprocessor can be programmed to perform different computation, such as modular multiplications and additions with arbitrary operand length. Therefore, different Public-Key cryptosystems such as ECC over $\mathbb{F}_p$ and RSA can also be easily implemented on this platform.

As shown in Fig. 2(a), the MicroBlaze processor communicates with the coprocessor via memory-mapped registers, *i.e.*, instruction register (A) and two data sharing register (B and C), and an interrupt signal. The coprocessor consists of a decoder, data memory (DataRAM), microinstruction memory (InsRom) and multiple embedded cores. Fig. 2(b) shows the block diagram of a core. Each core here



(a) Schematic block diagram for the platform.



(b) Schematic block diagram for the core.

Figure 2. Overview of the multi-core system.

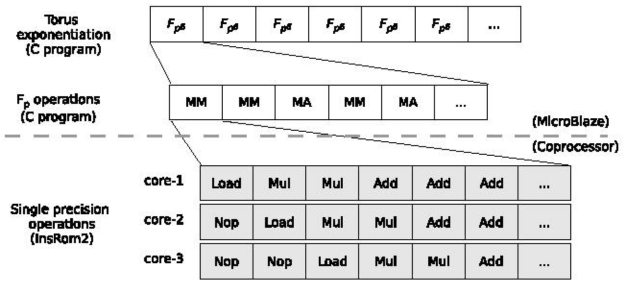
is a highly simplified Load/store CPU, and supports only 7 instructions. It does not support branch jumps. We also utilize the dedicated multipliers on the FPGA to construct the ALU of each core. In order to reduce the area, both InsRom and DataRAM are single port memory and implemented in the Block RAM of the FPGA.

The decoder fetches instructions from the instruction register (register A), and performs correspondding microinstructions stored in InsRom. The microinstructions are dispatched to the cores in parallel via the instruction bus. The data memory has only one read/write port, therefore, a single data memory access is allowed in each cycle. The decoder manages the data memory so that conflicts are avoided.

3.2 Implementation Hierarchy

As shown in Fig. 1, torus arithmetic can be represented in various ways and on different levels. On the platform shown in Fig. 2, the torus exponentiation is performed in three levels. One torus exponentiation consists of a sequence of $\mathbb{F}_{p^6}$ operation, which consists of a sequence of Modular Multiplications (MM) and Modular Additions (MA) in $\mathbb{F}_p$. Obviously, the sequence of modular operations can either be generated in software *e.g.* as a C code or can be put in the coprocessor. We investigate both of these two types of implementation.

3.2.1 Type-A Implementation



Torus exponentiation (C program)
F_{p^6}
...
F_p operations (C program)
MM
MA
(MicroBlaze)
(Coprocessor)
Single precision operations (InsRom2)
core-1
core-2
core-3
Load
Mul
Add
Nop

Figure 3. Torus exponentiation in hierarchy: Type-A Implementation.

Fig. 3 shows the Type-A implementation. Here the MicroBlaze generates the sequence of MM and MA, and sends them to the coprocessor one by one. For example, the MicroBlaze puts a "MM" instruction to register A to perform a modular multiplication.

```
MM  AddrC, AddrA, AddrB
```

The coprocessor decodes this instruction, and executes the corresponding microinstructions that are stored in the InsRom. After finishing this multiplication, the coprocessor generates an interrupt signal, which will be monitored and handled by the MicroBlaze. Afterwards the MicroBlaze can send next instruction.

As one $\mathbb{F}_{p^6}$ operation consists of $18M + 60A$, the total of 78 register A accesses and 78 interrupts handling are required. One register A access together with one interrupt handling requires 184 clock cycles, while one 170-bit modular multiplication requires 193 clock cycles. Therefore, the communication between the MicroBlaze processor and the coprocessor becomes the bottleneck of the whole system.

3.2.2 Type-B Implementation

One possible way to improve the performance is to reduce the communication overhead. Without losing any flexibility, we add another instruction ROM (InsRom1) to the coprocessor and we denote this architecture as Type-B. In the InsRom1 we store the sequence on level 2. Fig. 4 shows this implementation.

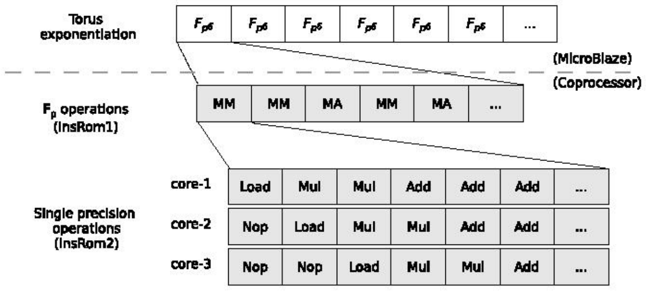


Figure 4. Torus exponentiation in hierarchy: Type-B Implementation.

Now MicroBlaze sends instruction on level 1.

```
T6M  AddrC, AddrA, AddrB
```

The coprocessor decodes this instruction, and fetches the corresponding sequence of MM and MA in InsRom1. For each MM or MA, the coprocessor performs the corresponding microinstructions stored in InsRom2. The Type-B implementation requires only one register A access and one interrupt for each $\mathbb{F}_{p^6}$ operation, therefore the performance is improved.

Both Type-A and Type-B offer high flexibility. Instead of 170-bit MM/MA, one can compose program with microinstructions to perform 1024-bit MM/MA, thus 1024-bit RSA is supported. In order to support ECC, on level 2 we can also put a sequence of MM/MA to construct a Point Addition (PA) or Point Doubling (PD) operations instead of $\mathbb{F}_{p^6}$. We also implemented 160-bit ECC and 1024-bit RSA on this platform to compare their performance with the performance of the torus.

3.3 Implementation of Montgomery Modular Multiplication

The performance of one Montgomery modular multiplication is bounded by the system architecture and the instruction scheduling method in use. Efficient instruction scheduling method for Montgomery modular multiplication on multicore system was discussed in [4]. The main challenge here is to reduce the number of data transfers between different cores. The data dependency of Alg. 1 is mainly caused by the carry generated by additions, *i.e.,* by the computation of $Z \leftarrow (Z + X \cdot y_i + P \cdot T)/r$. To utilize all the cores efficiently, carry should be used only in the core where it was generated. In [4], we observed that only Z_0 has to be generated in the end of each iteration, while $Z_{s-1}, .., Z_1$ can be generated in the end of the loop. Based on this observation, an instruction scheduling method which avoids carry transfers and efficently utilizes all the cores is proposed. The result in [4] shows that a 256-bit MM on a 4-core system is 2.96 times faster than the single core based implementation. We use this instruction scheduling method here in the CEILIDH implementation.

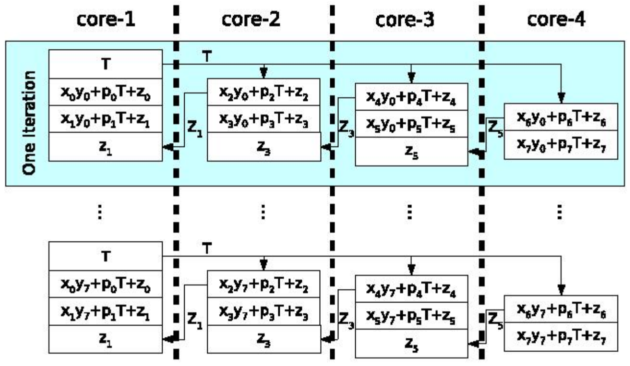


Figure 5. Parallelized 256-bit Montgomery modular multiplication on a 4-core system.

Fig.5 shows how this method works. During the whole loop (z_1, z_0) is generated and stored in core-1, (z_3, z_2) in core-2, (z_5, z_4) in core-3 and (z_7, z_6) in core-4. Carry is only used in the local core. At the end of each iteration, z_2 in core-2 becomes new z_1 and is sent to core-1. Also, z_4 is sent to core-2 and z_6 is sent to core-3. After eight iterations and a conditional substraction, $Z = X \cdot Y \cdot R^{-1} \bmod P$ is obtained and stored separately in the register file of each core.

4. Results

We implemented this platform on a Xilinx FPGA Virtex-II Pro. Table 1 shows the number of clock cycles for different modular operations. The result shows that one 170-bit Montgomery modular multiplication requires 193 clock cycles, while one addition needs 47 clock cycles. The reason that modular additions are relatively slow is that we only use one core to perform modular additions and subtractions. This is because carry needs to be transferred if multiple cores are used to perform modular additions.

While 160-bit modular operations are a little bit faster than 170-bit operations, 1024-bit Montgomery modular multiplication is about 23 times slower than 170-bit multiplication.

Table 1. Number of clock cycles for different operations.

Bitlength	Operations	Number of clock cycles
	Interrupt Handling	184
170-bit (torus)	Modular Mult.	193
	Modular Add.	47
	Modular Sub.	61
160-bit (ECC)	Modular Mult.	163
	Modular Add.	40
	Modular Sub.	53
1024-bit (RSA)	Modular Mult.	4447

Table 2 shows the results of $\mathbb{F}_{p^6}$ multiplication and ECC PA/PD of Type-A and Type-B implementations. One 170-bit $\mathbb{F}_{p^6}$ multiplication takes 22348 clock cycles for the Type-A implementation. However, only 5908 clock cycles are requried for the Type-B implementation, which is 3.78 times faster than the Type-A implementation. For ECC, PA on Type-B implementation is about 2.49 times faster compared with that on Type-A, and PD is about 2.17 times faster.

The design is synthesized and implemented on a Xilinx Virtex-II Pro (XC2VP30) FPGA. A maximum frequency of 74 MHz can be achieved. The data memory and instruction memory are implemented in block RAM of the FPGA board. In total, 5419 slices are used for this design, where the coprocessor requires 3285 slices. Table 3 shows the performance of torus, ECC and RSA on this platform. One 170-bit T_6 exponentiation requires 20 ms, while one 1024-bit RSA exponentiation requires 96 ms. In this case, CEILIDH is about 5 times faster than RSA on the same platform. Note that one $\mathbb{F}_{p^6}$ multiplication requires 18 MM and 60 MA. Further performance improvement is acquirable by

Table 2. Number of clock cycles for different operations in Type-A and Type-B implementation.

Architecture Type	Operations	Number of clock cycles
Type-A	torus T_6 Mult.	22348
	ECC PA	7185
	ECC PD	5793
Type-B	torus T_6 Mult.	5908
	ECC PA	2888
	ECC PD	2665

Table 3. Performance comparison between torus, ECC and RSA on the same platform.

PKC	Area $[slices]$	Freq. $[MHz]$	Time $[ms]$
170-bit torus	5419	74	20
1024-bit RSA	5419	74	96
160-bit ECC	5419	74	9.4

performing parallel computation between these modular operations. On the same platform, one 160-bit ECC scalar multiplication requires 9.4 ms, which is about two times faster than CEILIDH.

5 Conclusions and Future work

We describe a design approach of CEILIDH on a multicore platform. A MicroBlaze is used as a controller together with a multicore coprocessor. The result shows that 170-bit T_6 exponentiation requires 20 ms, which is about 5 times faster than 1024-bit RSA on the same platform. Compared to ECC, CEILIDH has the same advantage of small key size and small cypher length. However, it is about two times slower than ECC with equivalent security.

For future work, we believe that by deploying fast modular adders, the performance can be improved. Also, by explorering parallelism between modular operations, further improvement is obtainable.

6. Acknowledgements

Junfeng Fan, Lejla Batina and Kazuo Sakiyama are funded by research grants of Katholieke Universiteit Leuven (OT/06/40) and FWO projects (G.0300.07 and G.0450.04). This work was supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), by the EU IST FP6 projects (ECRYPT) and by the IBBT-QoE project of the IBBT.

References

[1] T. Blum and C. Paar. High-radix Montgomery modular exponentiation on reconfigurable hardware. *IEEE Transactions on Computers*, 50(7):759–764, July 2001.

[2] Ç.K. Koç, T. Acar, and B. Kaliski Jr. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, June 1996.

[3] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[4] J. Fan, K. Sakiyama, and I. Verbauwhede. Montgomery modular multiplication algorithm on multi-core systems. In *Proceedings of the IEEE Workshop on Signal Processing Systems: Design and Implementation (SIPS 2007)*, pages 261–266, 2007.

[5] R. Granger, D. Page, and M. Stam. A comparison of CEILIDH and XTR. In D. Buell, editor, *Proceedings of Algorithmic Number Theory - ANTS-VI*, number 3076 in Lecture Notes in Computer Science, pages 235–249, 2004.

[6] N. Koblitz. Elliptic curve cryptosystem. *Math. Comp.*, 48:203–209, 1987.

[7] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[8] 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.

[9] P. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[10] K. Rubin and A. Silverberg. Torus-based cryptography. In D. Boneh, editor, *Advances in Cryptology: Proceedings of CRYPTO'03*, number 2729 in Lecture Notes in Computer Science, pages 349–365. Springer-Verlag, 2003.

[11] M. Stam and A. Lenstra. Efficient Subgroup Exponentiation. In B. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Proceedings of 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, number 2535 in Lecture Notes in Computer Science, pages 318–332. Springer-Verlag, 2002.