Efficient Hardware Design for Computing Pairings Using Few FPGA In-built DSPs

Riadh Brinci*, Walid Khmiri, Mefteh Mbarek, Abdellatif Ben Rabâa, Ammar Bouallègue

*(Labortoire Sys'Com, Ecole Nationale d'Ingénieurs de Tunis, 1000 Tunis
Email: br.riadh@gmail.com)

ABSTRACT

This paper is devoted to the design of a 258-bit multiplier for computing pairings over Barreto-Naehrig (BN) curves at 128-bit security level. The proposed design is optimized for Xilinx field programmable gate array (FPGA). Each 258-bit integer is represented as a polynomial with five, 65 bit signed integer, coefficients. Exploiting this splitting we designed a pipelined 65-bit multiplier based on new Karatsuba- Ofman variant using non-standard splitting to fit to the Xilinx embedded digital signal processor (DSP) blocks. We prototype the coprocessor in two architectures pipelined and serial on a Xilinx Virtex-6 FPGA using around 17000 slices and 11 DSPs in the pipelined design and 7 DSPs in the serial. The pipelined 128-bit pairing is computed in 1. 8 ms running at 225MHz and the serial is performed in 2.2 ms running at 185MHz. To the best of our knowledge, this implementation outperforms all reported hardware designs in term of DSP use.

***Keywords*-**Cryptography, Field Programmable Gate Array (FPGA), Modular Multiplication, Non-Standard Splitting, Pairing-Friendly Curves

1 INTRODUCTION

A bilinear pairing is a map $G_1 \times G_2 \rightarrow G_T$ where G_1 and G_2 are typically additive groups and G_T is a multiplicative group and the map is linear in each component. Many pairings used in cryptography such as the Tate pairing [1], R-ate pairing [2], ate pairing [3] and optimal pairings [4], choose G_1 and G_2 to be specific cyclic subgroups of $E(F_{p^k})$, and G_T to be a subgroup of $F^*_{p^k}$.

1.1 Ate pairing

Let F_p be a finite field and let E be an elliptic curve defined over F_p. Let r be a large prime dividing $\#E(F_p)$ and k the embedding degree of $E(F_p)$ with respect to r, namely, the smallest positive integer k such that $r|(p^k - 1)$. For any finite extension field K of F_p, denote with $E(F_p)[r]$ the K-rational r –torsion group of the curve. For $P \in E(K)$ and an integer s, let O be the infinity point of E and $f_{s,P}$ be a K –rational function or Miller function with divisor

$$(f_{s,P}) = \mathrm{s(P)} - (\mathrm{[s]P}) - (\mathrm{s} - 1)(O)$$

let $G_1 = E(F_p)[r], G_2 = E(F_{p^k}) \cap Ker(\pi_p - [p])$, where π_p is the p^{th} power of Frobenius endomorphism;

$$\pi_p : E \rightarrow E$$
$$(x, y) \rightarrow (x^p, y^p)$$

and $G_T = \mu_r \subset F^*_{p^k}$

Let $P \in G_1$, $P \in G_2$ and $t = p + 1 - \#E(Fp)$ be the trace of Frobenius, then,

$$\alpha(P, Q) = (f_{t-1,Q}(P))^{(p^k-1)/r}$$

is non-degenerate bilinear, and computable pairing, it is the ate pairing

1.2 Pairing-Friendly Curves

An elliptic curve E over F_p is called pairing-friendly whenever there exists a large prime $r|\#E(F_p)$ with $r > \sqrt{p}$ and the embedding degree k is small enough, i.e. $k < log_2(r)/8$. Many construction methods result in a parametrized family of elliptic curves, i.e. r and p are given by the evaluation of polynomials $r(u)$ and $p(u)$ at an integer value u. One of the most important examples of such families are the Barreto- Naehrig (BN) curves [5], ideally suitable for implementing pairings at the 128-bit security level. These curves have $k = 12$ are defined by

$$p(u) = 36u^4 + 36u^3 + 24u^2 + 6u + 1$$
$$r(u) = 36u^4 + 36u^3 + 18u^2 + 6u + 1$$

for some $u \in Z$ such that p is prime. We show that when we choose $u = 2^\tau + s$, where s is a reasonably small number, the modular multiplication in F_p can be substantially improved.

The R-ate pairing [2] is a generalization of ate pairing and can be seen as an instantiation of optimal pairings [4]. Since the definition of the optimal ate pairing really depends on the particular elliptic curve one is using, we only provide the definition in the case of BN curves: using the same G_1 and G_2 as for ate

pairing, the optimal ate pairing on BN curves is defined as [6],

$$\varrho(P,Q) = (f.\left(f.l_{aQ,Q}(P)\right)^{p}.l_{\pi(aQ+Q),aQ}(P))^{(p^k-1)/r}$$

where $a = 6u + 2$, $f = f_{a,Q}(P)$ and $l_{A,B}$ denotes the line through points A and B

Algorithm 1 *Optimal Ate Pairing over BN Curves*

Input: $a = |6u + 2| = \sum_{i=0}^{s-1} a_i 2^i$, $P \in E(F_p)[r]$, $Q \in E(F_{p^{12}})[r] \cap ker(\pi_p - [p])$

Output: $\rho(P,Q) \in F_{p^{12}}$

1. $T \leftarrow Q, f \leftarrow 1$
2. ***for*** $i = s - 2$ ***downto*** 0
3. $T \leftarrow 2T, f \leftarrow f^2.l_{T,T}(P)$
4. ***if*** $a_i = 1$ ***then***
5. $T \leftarrow T + Q, f \leftarrow f.l_{T,Q}(P)$
6. ***end if***
7. ***end for***
8. $f \leftarrow (f.\left(f.l_{aQ,Q}(P)\right)^{p}.l_{\pi(aQ+Q),aQ}(P))^{(p^k-1)/r}$
9. ***return*** f

Algorithm 1 used arithmetic in $F_{p^{12}}$ based on irreducible binomials through a tower of extensions. In our paper we present the towering scheme as:

$F_{p^2} = F_p[i]/(i^2 - \beta)$, where $\beta = -1$

$F_{p^4} = F_{p^2}[s]/(s^2 - \xi)$, where $\xi = i + 1$

$F_{p^{12}} = F_{p^4}[t]/(t^3 - s) = F_{p^2}[\tau]/(\tau^6 - \xi)$,

The choice of this towering $F_{p^2} \rightarrow F_{p^4} \rightarrow F_{p^{12}}$ makes the final exponentiation much cheaper than other choices. Also we choose $p \equiv 3\ mod\ 4$ to accelerate arithmetic in F_{p^2} since multiplication by $\beta = -1$ is simple subtraction. For BN curve we choose $E: y^2 = x^3 + 2$ and $u = -(2^{63} + 857)$. This choice of curve parameters will simplify and speed up the reduction phases of the proposed Modular Integer Polynomial Montgomery Multiplier.

1.3 FPGA resources

FPGA manufacturers integrate more and more of dedicated function blocks into modern devices. For example, Xilinx Virtex-6 FPGAs include separate columns of additional function hard cores for memory (BRAM) and arithmetic DSP operations. The DSP blocks are grouped in pairs that span the height of four or five CLBs, respectively. The dual-ported BRAM matches the height of the pair of DSP blocks and supports a fast data path between memory and the DSP elements. Of particular interest is the use of these memory elements and DSP blocks for efficient Boolean and integer arithmetic operations with low signal propagation time. Large devices of Xilinx Virtex-6 class are equipped with up to thousand individual function blocks of these dedicated memory and arithmetic units. Originally, the integrated DSP blocks as indicated by their name were designed to accelerate DSP applications, e.g., Finite Impulse Response (FIR) filters, etc. However, these arithmetic units can be programmed to perform universal arithmetic functions not limited to the scope of DSP filter applications; they support generic multiplication, addition and subtraction of (un)signed integers [7].

1.4 Outline

The remainder of this paper is organized as follows: Section II studies the most important existing works related to efficient hardware implementations of multiplication over $F_p$suitable for computing pairings over BN curves. Section III introduces our hardware design of 65 x 65 bit multiplier based on DSP macro for Virtex-6 and performance comparison. Section IV focuses on the hardware implementation and performance comparison of our 258 bit multiplier. Finally, section V provides conclusion and future works.

2 Related Works

Since 2009, many hardware implementations of multiplication over F_p suitable for computing pairings on BN curves was described. The first work was described by Fan et al. [8]. Their proposed architecture was based on Hybrid Montgomery Multiplier (HMM) where multiplication and reduction was interleaved. In same year, a new Application Specific Integrated Circuit (ASIC) implementation of pairings over BN curves was proposed. In 2010 Fan et al. [6] proposed a new pipelined and parallelized version of their HMM [8]. In 2011, Corona et al. [9] proposed a new hardware implementation of 258 bit multiplier suitable for computing pairings over BN curves. They used an asymmetric divide and conquer approach to efficiently implement their 65x65 bit multiplier. Their design used only 12 DSP slices on a Xilinx Virtex 6. In same year, Yao et al. [10] proposed a new hardware implementation of optimal ate pairing on Virtex 6. The design computed multiplication over F_p using 32 DSP slices. They combined Lazy reduction with RNS representation.

3 Finite field Modular Multiplier

Multiplication is one of the major elements of a pairing coprocessor. In this paper, we propose a Modular Integer Polynomial Montgomery Multiplier (MIPMM) based on 5-Term Karatsuba. It is hybrid multiplier that computation is achieved in four dependent phases. In open literature there is many works proposed to efficient implement serialization of Karatsuba in pairing computation. Few papers presented implementation in prime fields [6], [8][9]. This paper focuses on the Karatsuba serialization in prime field F_p

Algorithm 2 *Modular Integer Polynomial Montgomery Multiplier (MIPMM) for BN curves*

Input: $a(u) = \sum_{i=0}^{4} a_i u^i, b(u) = \sum_{i=0}^{4} b_i u^i$
$p(u) = 36u^4 + 36u^3 + 24u^2 + 6u + 1$
Output: $v(u) \equiv a(u).b(u).u^{-5} \bmod p(u)$
1. **Phase 1: Polynomial Multiplication**
2. $c(u) = \sum_{i=0}^{8} c_i u^i = a(u).b(u)$
*/*computed by algorithm 3*/*
3. **Phase 2: Partial coefficient reduction**
4. ***for*** $i = 0$ ***to*** 4
5. $q_i = c_i \; div \; 2^\tau; \;\; r_i = c_i \bmod 2^\tau$
6. $c_{i+1} = c_{i+1} + q_i; \;\; c_i = r_i - s.q_i$
7. ***end for***
8. **Phase 3: Polynomial reduction**
9. $t(u) = \left(-c_4 + 6\left(c_3 - 2c_2 - 6(c_1 - 9c_0)\right)\right)u^4$
10. $+\left(-c_3 + 6(c_2 - 2c_1 - 6c_0)\right)u^3$
11. $+(-c_2 + 6(c_1 - 2c_0))u^2$
12. $+(-c_1 + 6c_0)u$
13. $h(u) = 36t_4u^3$
14. $+36(t_4 + t_3)u^2$
15. $+12(2t_4 + 3(t_3 + t_2))u$
16. $+6(t_4 + 4t_3 + 6(t_2 + t_1))$
17. $v(u) = c(u)/u^5 + h(u)$
18. **Phase 4: Coefficient reduction**
19. ***for*** $i = 0$ ***to*** 3
20. $q_i = v_i \; div \; 2^\tau; \;\; r_i = v_i \bmod 2^\tau$
21. $v_{i+1} = v_{i+1} + q_i; \; v_i = r_i - s.q_i$
22. ***end for***
23. ***return*** $v(u)$

This multiplier consists of 258 bits five terms Karatsuba multiplication which is constructed by one 65x65 bits multiplier basic core. There is registers units to save intermediate results to be used later. The main contributions of this paper are the efficient use of in-built features offered by modern FPGA devices: DSP, adders, subtractors, shift registers… We also designed two variants of basic 65x65 bits multiplier using respectively 7 and 11 DSP slices. Our proposed serialization exploits the independence in each phase and between phases to reduce the cycle count. Figure 1 depicts the top level of the proposed design of Modular Integer Polynomial Montgomery Multiplier for BN curves.

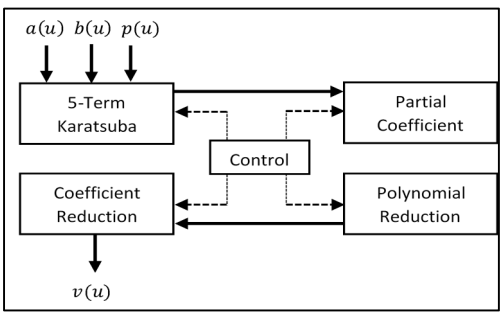


Figure 1 Top level of MIPMM

3.1 Five term Karatsuba Multiplier

The first work to compute Karatsuba using more than three terms was proposed by Peter Montgomery [11]. But it was not suitable for hardware implementation because the large number of addition and subtraction. Corona et al. [9] proposed new scheduling for addition and subtraction to fit hardware design. In this work we propose a more efficient scheduling to achieve one 258 bit multiplication in only 22 cycles. Algorithm 3 describes our Five term Karatsuba Multiplier. This multiplier is based on basic 65 bit multiplier core described in the next subsection.

Algorithm 3 *Proposed Five Term Karatsuba*

Input: $a(u) = \sum_{i=0}^{4} a_i u^i, b(u) = \sum_{i=0}^{4} b_i u^i$
Output: $c(u) \equiv \sum_{i=0}^{8} c_i u^i$
1. $p_0 = a_0 b_0$
2. $p_1 = a_1 b_1; \;\; n_0 = a_0 + a_1; \;\; n_1 = b_0 + b_1$
3. $p_2 = n_0 n_1$
4. $p_3 = a_2 b_2; \;\; n_2 = a_0 + a_2; \;\; n_3 = b_0 + b_2$
5. $p_4 = n_2 n_3$
6. $p_5 = a_3 b_3; \;\; n_4 = a_2 + a_3; \;\; n_5 = b_2 + b_3$
7. $p_6 = n_4 n_5; \;\; n_6 = a_3 + a_1; \;\; n_7 = b_3 + b_1$
8. $p_7 = n_6 n_7; \;\; n_8 = n_0 + n_4; \;\; n_9 = n_1 + n_5$
9. $p_8 = n_8 n_9$
10. $p_9 = a_4 b_4; \;\; n_{10} = a_0 + a_4; \;\; n_{11} = b_0 + b_4$
11. $p_{10} = n_{10} n_{11}; \;\; n_{12} = n_0 + a_4; \;\; n_{13} = n_1 + b_4$
12. $p_{11} = n_{12} n_{13}; \;\; n_{14} = a_2 + a_4; \;\; n_{15} = b_2 + b_4$
13. $p_{12} = n_{14} n_{15}; \;\; n_{16} = n_{14} + a_3; \;\; n_{17} = n_{15} + b_3$
14. $p_{12} = n_{16} n_{17}$
15. $c_0 = p_0$
16. $m_0 = p_1 + p_0; \;\; m_1 = p_1 - p_0;$
17. $c_1 = p_2 - m_0; \;\; m_9 = p_9 + m_0$
18. $m_{11} = p_1 + c_1; \;\; m_{14} = p_9 + p_3$
19. $m_2 = p_4 - p_3; \;\; m_5 = p_4 + c_1$
20. $c_2 = m_2 + m_1; \;\; m_3 = p_3 + p_5$
21. $s_1 = p_6 - m_3; \;\; m_8 = m_3 - m_9$
22. $m_7 = m_5 + s_1; \;\; m_{12} = s_1 - m_{11}$
23. $m_6 = p_8 - p_7$
24. $c_3 = m_6 - m_7; \;\; m_{10} = p_{10} + p_7$
25. $c_4 = m_{10} + m_8; \;\; m_{13} = p_{11} - p_{10}$
26. $c_5 = m_{13} - m_{11}; \;\; m_{15} = p_{12} + p_5$
27. $c_6 = m_{15} - m_{14}; \;\; m_{16} = p_{13} - s_1$
28. $c_7 = m_{16} - m_{15}$
29. ***return*** $c(u)$

3.2 Basic 65 bit multiplier core

The five term Karatsuba multiplier turned around this module. We propose two different architectures to perform asymmetric multiplication using common non-standard splitting technique. For instance, the asymmetric operands can be computed by the following equation with 25 and 18 bits to fit DSP core of the FPGA. We reserve the most significant bit as operand's sign. Let

$$Z = AB = (\sum_{i=0}^{64} a_i 2^i)(\sum_{i=0}^{64} b_i 2^i)$$

We decide to not perform full DSP computation for the last core $B_{51:64}A_{0:23}$ to avoid non-useful operations. So, we can write $Z = Z_0 + 2^{24}Z_1 + 2^{48}Z_2$ as described in Figure 2 For example we can split the operands of Z_0 as

$$Z_0 = B_{0:16}A_{0:23} + 2^{17}(B_{17:33}A_{0:23} + 2^{17}(B_{34:50}A_{0:23} + 2^{17}(B_{51:64}A_{0:23})))$$

In this work we propose two designs called respectively full pipelined and serial architectures depicted respectively. In the first design, we propose three sets of DSP cores arrangement (Z_0, Z_1 and Z_2): the two first sets have the same design and each set is performed by four DSP slices.

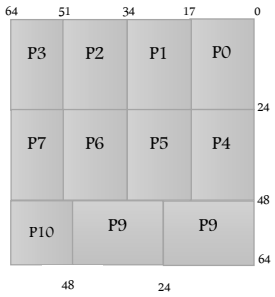


Figure 3 Proposed tilling

The last set is computed by only three slices. We used in the basic core three DSP parameterization detailed as eight 25x18 bit, two 25x15 and one 18x18 DSP configuration. This idea reduces the frequency of the multiplier but let us reduce power consumption by saving extra registers and non-useful operations. This first design achieves one 65x65 bit multiplication in seven cycles using eleven DSP cores.

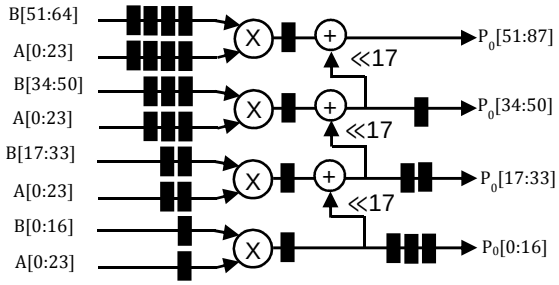


Figure 4 Proposed hardware design of the first DSP set

The diagram in describes the delay constrained of the full pipelined architecture that takes seven cycles to achieve 65x65 bit multiplication.

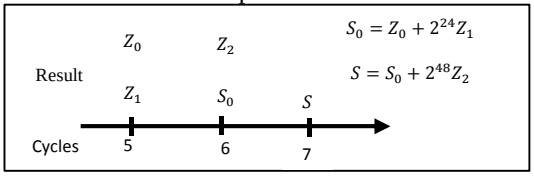


Figure 5 delay constrained of the full pipelined architecture

In the second design, we rearrange operations to make Z_0 and Z_1 share the same hardware. So, we compute Z_0 and Z_2 in parallel. To get results at same time we added a pipeline stage in the DSP set of Z_2. At the 5^{th} we have our outputs. In the second cycle, we entered the operands of Z_1 to get it at the 6^{th} cycle. At this time we have also the result of the addition $S_0 = Z_0 + 2^{48}Z_2$. The adder is an in-built feature of the FPGA configured to give result after one cycle. Finally, we performed the last addition, configured also with latency one, giving full result after seven cycles using only seven DSP slices. In this second architecture we have added extra hardware finite state machine, multiplexers and demultiplexers.

3.3 Coefficient and polynomial reductions

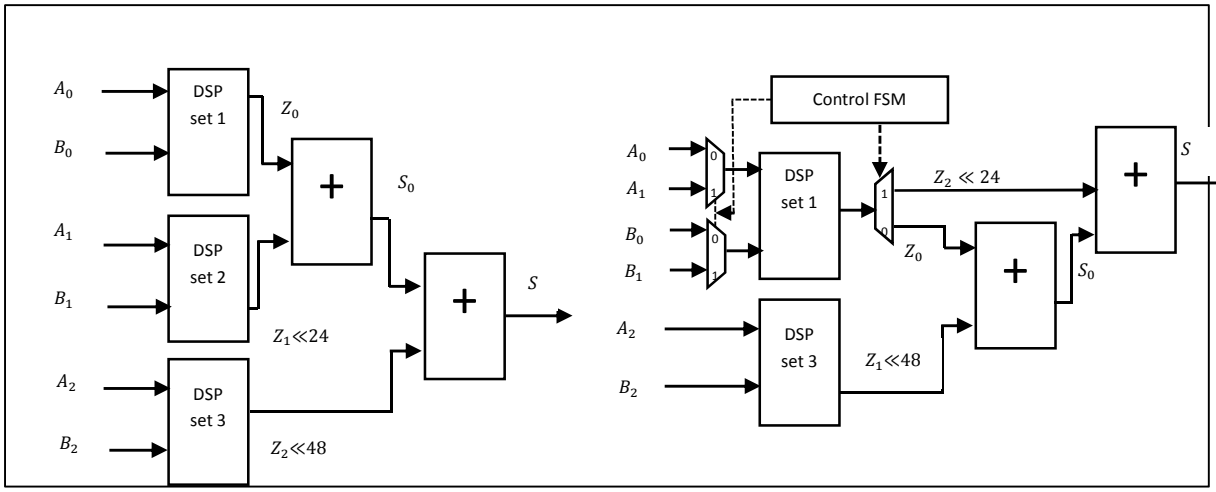
The architecture described in Pipelined architecture

(b) Serial architecture

Figure 6 depicts the top level of each architecture shows the polynomial reduction phase. It can reduce the coefficients one by one taking twelve cycles to achieve the entire reduction. We performed multiplication by s using shifts and addition. In this phases, the complexity can be reduced by exploiting the characteristics of the different constants. Since $s = 2^5(2^4 + 2^3) + 2^6 + (2^4 + 2^3) + 1$, multiplication by s is performed by three additions in three cycles. There is also multiplication by the following constants $6, 9, 12$ and36 computed by shifts and additions, e.g. $6a = 2^2a + 2a$; $9a = 2^3a + a$; $12a = 2^3a + 2^2a$; $36a = 2^5a + 2^2a$;

3.4 Delay constrained of MIPMM

As mentioned before, the 65x65 bit multiplication takes seven cycles to achieve one multiplication. We get all partial products (PP) $p_{i\in\{0,13\}}$ shown in algorithm 3 after 13 cycles. However the delay of datapath for the post partial products combined with PP is two clock cycles. As result, five term Karatsuba gives the first output after 22 clock cycles. As soon as each $c_{i\in\{0,7\}}$ gets out from the pipelined PP core, it is scheduled on the fly to be partially reduced. This phase ends at the 22 second clock cycle. Other reduction phases also combined with phase one take 13 cycles Therefore, to sum up, the cost of the entire multiplier is 35 clock cycles.



(a) Pipelined architecture (b) Serial architecture

Figure 6 Top level design of proposed pipelined and serial architectures

4 PAIRING DESIGN

Most operations in optimal pairing algorithm steps of are performed in $F_{p^{12}}$. Many techniques give efficient computation in extended fields with low complexity. We choose methods with minimum squaring and multiplication. The underlying operations are computed in base field. Therefore we design our coprocessor as scheduling of F_p operations.

As shown in Algorithm 1, Miller loop phase consists of the following major operations:

- Doubling step, is the elliptic curve point doubling combined with the computation of the line l.
- Addition step, is the elliptic curve point addition combined with the computation of the line l.
- Squaring of the Miller function f.
- Spare multiplication of f by l having only half of non-zero coefficients.

We adopt homogenous coordinates proposed in [12] to efficient compute the different curve operations in the Miller loop. The listed above steps need arithmetic in F_{p^2} such as multiplication and squaring. We propose Karatsuba method described by the following equation to compute multiplication.

$v_0 = a_0 b_0; v_1 = a_1 b_1$

$c_0 = v_0 - v_1$

$c_1 = (a_0 + a_1)(b_0 + b_1) - v_0 - v_1$

We also refer to complex method to perform squaring in F_{p^2}. First, we precompute $v_0 = a_0 a_1$. Then, the square $c = a^2$ is computed as

$c_0 = (a_0 + a_1)(a_0 - a_1)$

$c_1 = 2v_0$

Multiplication and squaring operations need respectively 36 and 39 cycles to get out their results. To efficient compute Miller loop we made rearrangements and scheduling in each step to fit our design. In the doubling step we have to compute three squaring and two multiplications which are equivalent to 12 F_p multiplications in the first part. They takes 48 cycles. In the second part, we have 17 F_p multiplications giving results out in the 94th cycle. The f^2 and $f.l$ need 111 F_p multiplications computed in 3.885 cycles. To sum up, each Miller loop iteration takes 3.979 clock cycles. Using the same strategy in curve rearrangement addition step is achieved in 3,385 clock cycles. As result, Miller loop takes 277.000 cycles.

The final exponentiation consists of final addition and final exponentiation. Table 1 gives the different operation in this step and the cycle count.

step	F_p multiplications
Final Addition	204
f^{p^6-1}	579
f^{p^6+1}	768
$f^{p^6-p^2+1/n}$	1813
Others	356
Cycle count	130.200

Table 1 Cycle count of the final exponentiation

5 RESULTS AND COMPARISON

The hole design has been done in VHDL using Xilinx ISE design suite on a Virtex-6 xc6vlx240t-3ff784 FPGA. It used in total 17560 slices, 7 and 11 DSP cores in our serial and pipelined architectures respectively. It runs at 185Mhz and finishes pairing computation on BN curve at 128 bit security level in 2.2 ms.

Table 2 lists the performance hardware implementations reported in recent literature. Compared with the other hardware implementation [6] our design saves DSP cores

Designs	**Curve**	**Architecture**	**Target**	**Area**	**Frequency** MHz	**Cycles** $x10^3$	**Delay** ms
This work	BN_{128}	Pipelined	xc6vlx240t	17560 slices, 11 DSP	225	407	1.8
	BN_{128}	Serial	xc6vlx240t	14890 slices, 7 DSP	185	407	2.2
[8]	BN_{126}	HMM digit-serial	ASIC 130mm	183k Gates	204	861	4.2
[6]	BN_{128}	HMM parallel	xc6vlx240t	4014 slices, 42 DSP	210	245	1.17
[13]	BN_{128}	Blakley	Xc4vlx200	52000 slices	50	821	16.4
[14]	BN_{126}	Montgomery	xc6vlx240t	3813 slices, 144 DSP	166	70	0.43
[15]	BN_{126}	RNS (Parallel)	xc6vlx240t	5237 slices, 64 DSP	210	78	0.338

Table 2 Performance comparison of hardware implementations of pairings at around 128-bit security

until 90%. Our goal is to keep the design of the pairing coprocessor full used by efficient resource sharing at high frequency. It is a serial implementation with minimum area use. The current design not only gains in in-built DSP slices with comparable pairing computation time but also shows that modern FPGA can be able to perform pairing with high complexity at higher security level on different friendly curves with large algebraic closure.

6 CONCLUSION

In this paper we introduce a new hardware design to efficiently serialize polynomial integer multiplication on BN curves over large prime field. Due to deep arrangement and careful scheduling of different steps of the coprocessor our design saves 90% of DSP slices and achieves one pairing computation in 1.8 ms. Our future work will be the multi-pairing computation to respond faster to many client requests. We plan also to implement other curves and different types of pairings on this architecture. Furthermore, we will provide an optimal parameter set and pairing implementations for higher security level including 192-bit or 256-bit security.

REFERENCES

[1] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient Algorithms for Pairing Based Cryptosystems. *CRYPTO 2002, volume 2442 of Lecture Notes in Computer Science*, pages 354-368. Springer, 2002.

[2] E. Lee, H. S. Lee, and C. M. Park. Efficient and Generalized Pairing Computation on Abelian Varieties. *Cryptology ePrint Archive, Available from http://eprint.iacr.org/. Report 2009/040.*

[3] F. Hess, N. P. Smart, and F. Vercauteren. The Eta Pairing Revisited. *IEEE Transactions on Information Theory*, 52(10), pages 459-462, Oct.2006.

[4] F. Hess. Pairing Lattices. *Pairing 2008, volume 5209 of Lecture Notes in Computer Science*, pages 18-38. Springer, 2008.

[5] P. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. *Selected Areas in Cryptography, SAC 2005*, LNCS 3897, pages 319-331, 2006.

[6] J. Fan, F. Vercauteren, and I. Verbauwhede. Efficient hardware implementation of Fp-arithmetic for pairing-friendly curves. *Computers, IEEE Transactions on*, *61*(5), 2012, 676-685.

[7] T.Güneysu. Utilizing hard cores of modern FPGA devices for high-performance cryptography, *Journal of Cryptographic Engineering*, *1*(1), 2011, 37-55.

[8] J. Fan, F. Vercauteren, and I. Verbauwhede. Faster Fp Arithmetic for Cryptographic Pairings on Barreto-Naehrig Curves. *CHES 2009, volume 5747 of Lecture Notes in Computer Science*, pages 240-253. Springer, 2009.

[9] C. Corona, C., Moreno, E. F., & Henriquez, F. R. Hardware design of a 256-bit prime field multiplier suitable for computing bilinear pairings. *International Conference on Reconfigurable Computing and FPGAs (ReConFig 2011)*, 2011, 229-234.

[10] G. X. Yao, J. Fan, R. C. Cheung, and I. Verbauwhede. A high speed pairing coprocessor using RNS and lazy reduction. *Cryptology ePrint Archive, Available from http://eprint.iacr.org/. Report 2011/258*, 2011

[11] P. L. Montgomery. Five, six, and seven-term Karatsuba-like formulae, *IEEE Transactions on Computers*, vol. 54(3), 362-369

[12] Costello, C., Lange, T., & Naehrig, M. (2010). Faster pairing computations on curves with high-degree twists. In *Public Key Cryptography–PKC 2010* (pp. 224-242). Springer Berlin Heidelberg.

[13] Ghosh, S., Mukhopadhyay, D., & Roychowdhury, D. (2010). High speed flexible pairing cryptoprocessor on FPGA platform. *Pairing-Based Cryptography-Pairing 2010* (pp. 450-466). Springer Berlin Heidelberg.

[14] Ghosh, S., Verbauwhede, I., & Roychowdhury, D. (2013). Core based architecture to speed up optimal ate pairing on FPGA platform. In *Pairing-Based Cryptography–Pairing 2012* (pp. 141-159). Springer Berlin Heidelberg.

[15] Yao, G. X., Fan, J., Cheung, R. C., & Verbauwhede, I. (2013). Faster pairing coprocessor architecture. In *Pairing-Based Cryptography–Pairing 2012* (pp. 160-176). Springer Berlin Heidelberg.