A Robust $GF(p)$ Parallel Arithmetic Unit for Public Key Cryptography

Santosh Ghosh Monjur Alam Indranil Sen Gupta Dipanwita Roy Chowdhury
Department of Computer Science and Engineering
Indian Institute of Technology
Kharagpur - 721302
e-mail: {santosh, monjur, isg, drc}@cse.iitkgp.ernet.in

Abstract

This paper presents the architecture and FPGA implementation of a robust $GF(p)$ parallel arithmetic unit. The most efficient modular multiplication, inversion and division units greatly reduce the clock cycles requirement for point operations applicable to Elliptic Curve Cryptography. The parallel arithmetic unit helps to achieve a high speed up in cryptographic applications. The architecture can resist the cryptographic timing attack. Integrated input and output interface units provide lower bandwidth requirement to plug in the architecture with automated cryptographic systems. The design exhibits its elegance among competitive architecture with respect to throughput and robustness.

Keywords: GF(p) Arithmetic Operators, Elliptic Curve Cryptography, Timing Attack, FPGA Implementation

1 Introduction

In the present day of enormously increasing stabile and mobile communications data is normally transmitted through shared insecure channels. To protect the confidential data in the field of e-commerce transactions, scientific and military purposes cryptography become utmost important and ambitious issue. Several cryptosystems like DES, RSA, AES, ECC have been developed to protect the confidential data. Bulk data are generally encrypted using block cipher (DES, AES), where both parties share the same common secret key. However, the establishment and exchange of these secret keys is normally achieved via public key cryptosystems. RSA and ECC are two mostly secured such public key cryptosystems. Generally, public key cryptosystems are more computationally intensive and slower than their private key counterparts [1]. Field Programmable Gate Array (FPGA) is an ideal platform to provide hardware acceleration to the cryptographic applications. The most effective advantage of FPGA, compared to ASIC, in cryptography is that they can be re-programmed to perform the more computationally intensive operations of a range of ciphers depending on the security and application requirements.

Elliptic Curve Cryptography (ECC) was independently proposed in the mid-eighties by Victor Miller [2] and Neil Koblitz [3] as an alternative to the existing public key system, RSA. ECC has quickly established itself as the final choice in smart cards, credit cards, mobile phones and palm-top devices due to its significant power to provide equivalent security compared to existing public key cryptosystems at greatly reduced key size. ECC is mathematically secure as no subexponential time algorithm is known to date to solve the discrete logarithm problem on a suitably chosen elliptic curve. The advantages offered by ECC could be important in the environments where processing power, storage and bandwidth are constrained. It is estimated that the security level of 160 and 224 bits ECC cryptosystem is equivalent to the 1024 and 2048 bits RSA respectively [9, 10].

Two types of *Finite Fields* are generally used in ECC. Those are Extended Binary Field that is also known as *Galois Field* $GF(2^k)$ and Finite Field over large Prime that is called $GF(p)$. Unlike $GF(2^k)$, a very few hardware implementation of ECC on $GF(p)$ has been reported in the literature to date. The hardware complexity to implement ECC in $GF(p)$ is little bit higher than that of in $GF(2^k)$. However, the advantage of $GF(p)$ over $GF(2^k)$ is that, a k-bit $GF(p)$ arithmetic unit can operate on any inputs form 0 to $2^k - 1$ without any reconfiguration. Again the reconfiguration of $GF(2^k)$ hardware could be implemented for some selective values of k, say reconfigurable hardware for $k = 160, 192, 224$ which can process only $160, 192$ and 224-bit data. But in case of $GF(p)$ the k-bit arithmetic unit is capable to process any i-bit data where $1 \leq i \leq k$.

The proposed design is a parallel $GF(p)$ arithmetic unit. The architecture is designed to perform the most efficient $GF(p)$ modular multiplication, inversion and division algorithms in binary number system. It has been observed

10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007)
0-7695-2978-X/07 $25.00 © 2007 IEEE

that such algorithms give the most efficient design with respect to both throughput and area. The proposed arithmetic unit performs all $GF(p)$ addition/subtractions, multiplications and inversion/divisions in $1,\ k\ and\ 2k$ clocks. Hence the design provides the robustness against any kind of data dependent timing attacks. The proposed design is mainly focussed to perform GF(p) Elliptic Curve (EC) operations directly in affine coordinates. Division is the most costly operation, which is essential to perform GF(p) EC group operations in affine coordinates. The current paper presents an efficient finite field invertor/divider unit, which can compute both the respective operations in twice clocks of the length of p. Intelligent loading and multiplexed display units have been integrated into the architecture to suit the target application.

The outline of this paper is as follows: In section 2 the design overview has been presented focussing on the design issues of the proposed work. Section 3 deals with the algorithms and architectures of different $GF(p)$ operations those are useful on ECC. The architecture of $GF(p)$ parallel arithmetic unit has been detailed in section 4. Section 5 deals with comparisons and analysis of the results. Finally section 6 concludes the work.

2 Design Overview

The implementation has been done by following conventional FPGA design abstraction. To achieve the final goal, maintained in the following subsection, first we set the input output specifications and sketched concurrent communicating blocks essentially be present at the highest architectural level of abstraction. Then for each of the individual blocks we have picked out the most efficient algorithms and logics, and mapped them into the most suitable architecture to get a optimum $GF(p)$ parallel arithmetic unit with respect to the conflicting performance parameters of robustness, throughput, area and power.

2.1 Motivation and Goal of the Design

Elliptic curve cryptosystems over $GF(p)$ have received very little attention to date due to the seemingly more attractive finite field $GF(2^k)$. However, our motivation goes to the finite field $GF(p)$ as unlike $GF(2^k)$, no reconfiguration logic is required for $GF(p)$ processor to process any input from 0 to k bits long. Örs et. al.[4] presented an elliptic curve processor effectively applicable to Montgomery domain numbers for elliptic curve point operations on projective coordinate. To perform point operations they focused mainly on modular multiplication and disregarded modular inversion and division operations. The $GF(p)$ ALU for encryption processor, reported by Daly et. al.[5], can perform point operations in affine as well as projective coordinates with a comparable performance where inputs and outputs are assumed to be represented in Montgomery domain. In this paper we propose a $GF(p)$ parallel arithmetic unit for natural binary number system that is effectively applicable for elliptic curve point operations regardless of point representing coordinates with an excellent performance.

2.2 Design Constraints

The present subsection gives an overview of the constraints under which the design is performed. The design strategies adopted to achieve the constraints are detailed in the sections 3 & 4. The architecture has been developed in a step by step method to achieve an efficient parallel $GF(p)$ arithmetic unit where p is assumed to be represented in natural binary format. The target application being elliptic curve schemes where the constraints of the design encompasses both throughput and power. The robust design architecture and novel implementation of $GF(p)$ multiplier and inversion/division units with proper freezing logic of internal registers helps to reduce the operational power without imposing any penalty of throughput. Thus the design could resist any kind of timing attacks. The input and output interface units and controller logics are designed carefully to make the architecture easily applicable to smart card like mobile devices with very less number of input/ouput connections. The lower pin count of the design also helps to reduce the power requirement to drive the device. The architecture of every arithmetic functional units are designed and optimized separately that effectively reduce the total area of the design.

3 Modular Arithmetic Operations and Respective Architectures

To perform public key cryptographic operations like point addition and point doubling operations on elliptic curve, mostly require addition, subtraction, multiplication, inversion and division in $GF(p)$. The following subsections carry the algorithm and efficient architecture for individual arithmetic operations.

3.1 GF(p) Addition

The modular addition operation adds two operands, A and B. (A, B) $\in$ [0, p - 1] and subtracts the modulus p from the sum if A + B $\geq p$. The architecture of k-bit $GF(p)$ adder is illustrated in Fig.1.

The architecture consist of two k-bit carry propagation adder. The first adder performs $A + B$ and the second adder is used to perform 2's complement subtraction by applying inverted p and 1 as its carry in input. A two channel to one channel multiplexer is used to select the final k-bit result

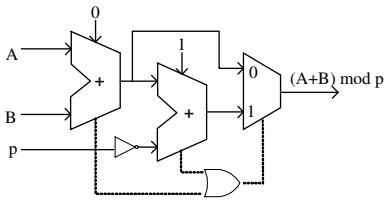

Figure 1. $GF(p)$ **Adder unit**

either from the first $(A + B)$ or second $(A + B - p)$ adder depending on their carry out values.

3.2 GF(p) Doubling

The modular doubling operation, 2A mod *p* of operand $A \in [0, p-1]$, left shifts the operand A by one bit and subtracts the modulus *p* from the temporary result if $2A \geq p$. Fig.2 depicts the architecture of k-bit modular doubling unit. The architecture works almost same as modular addition operation. The $2A$ operation is done by a one bit hard left shifter that produces k+1 bit output from k bit input. Finally the values $2A$ and $2A - p$ are multiplexed to give the correct result.

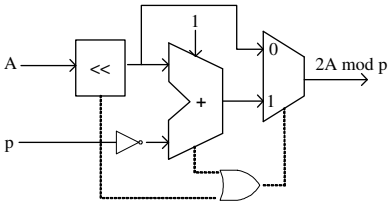

Figure 2. $GF(p)$ **Doubler unit**

3.3 GF(p) Subtraction

To perform modular subtraction, input B is bitwise inverted and added to input A with carry in 1. If the result is negative (i.e. the carry-out is low) then the modulus is added to produce an output in the range $[0, p-1]$. An architecture to perform $GF(p)$ subtraction is illustrated in Fig.3. In this architecture $A - B$ is computed by the first adder as 2's complement subtraction method. At the second adder the result of the first adder is simply added with the modulus p. The correct result is $A-B$ if $A \geq B$ or $A-B+p$ if $A < B$. The correct result is selected depending on the carry out bit of the first adder. The architecture performs modular subtraction without the cost of k-bit magnitude comparator to decide weather $A \geq B$ or not.

3.4 GF(p) Multiplication

Various types of $GF(p)$ multiplication algorithms have been reported in literature. Montgomery in 1985 [6] proposed one efficient modular multiplication without trial division. This algorithm works on the numbers represented in Montgomery domain and it produces the result in the same domain. The conversions from binary to Montgomery of the operands and Montgomery to binary of the result degrade the overall performance of $GF(p)$ multiplication in binary number system. In our design *Interleaved Multiplication Algorithm*[7, 8] has mapped into the architecture. The algorithm is given in $Algorithm1$ and respective hardware architecture is shown in Fig.4.

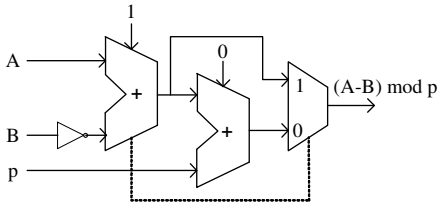

Figure 3. $GF(p)$ **Subtractor unit**

Algorithm 1: Interleaved Multiplication

Input: *p* and A, B ∈ [0, *p* - 1]
Output: AB mod *p*

1. T = 0
2. For i = 0 to k - 1 do
3. T = 2T + A.B_{k-1-i}
4. T = T mod *p*
5. End For
6. Return T

The time complexity of the algorithm is exactly k clock cycles, where k is the maximum bit length of applied *p*. The algorithm is carefully mapped into the architecture that eliminates the trial division by p in it's step 4. In the architecture, the step 3 and step 4 of the algorithm are implemented by instantiating one $GF(p)$ doubling unit followed by one $GF(p)$ adder unit. A mod-k down counter that generates the control signals of the multiplier unit, effectively implements the functionality of step 2 and step 6 of the algorithm. The counter output (i) is used to select B_i (i.e. B_{k-1-i}, according to the algorithm) from k-bit B at every clock and the *AND* array is used to perform A.B_i. The final result of the operation comes out through output buffers at $(k+1)^{th}$ clock.

3.5 GF(p) Inversion and division

The modular multiplicative inverse A^{-1}(mod *p*) of an integer A exists if and only if A and *p* are relatively prime, that is, gcd(A, *p*) = 1. One of the efficient modular inversion algorithms is *Binary Inversion Algorithm*, shown in Algorithm 2. The step 2 of the algorithm runs iteratively, and

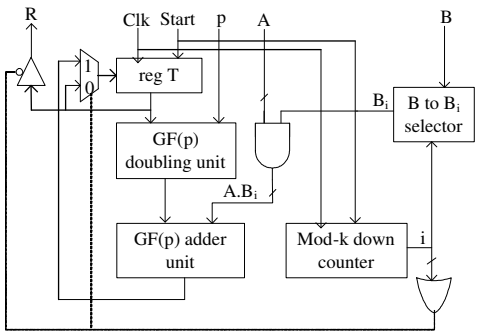

Figure 4. GF(p) interleaved multiplier unit

proceeds towards the goal. At every iteration either u or v is reduced by at least one bit length. It follows that the total number of iterations of step 2 is at most $2k$, where k is the maximum bit length of p and a. In [5] the authors proposed the outline of modular division operation using a modular inversion followed by a modular multiplication operation. The binary modular inversion algorithm (Algorithm 2) can easily be modified to perform modular division $b/a = ba^{-1}$. To obtain b/a (mod p) using this algorithm we have to initialize the x_1 variable in step 1 by b instead of 1.

Algorithm 2: Binary Inversion in $GF(p)$

Input: p and a $\in [0, p - 1]$
Output: $a^{-1} \bmod p$

---

1. $u = a, v = p, x_1 = 1, x_2 = 0$
2. **while u $\neq$ 1 and v $\neq$ 1 do**
2.1. **while u is even do**
2.2.1. u = u/2
2.2.2. if x_1 is even then $x_1 = x_1/2$ else $x_1 = (x_1+p)/2$
2.3. **end while**
2.4. **while v is even do**
2.5.1. v = v/2
2.5.2. if x_2 is even then $x_2 = x_2/2$ else $x_2 = (x_2+p)/2$
2.6. **end while**
2.7. if $u \geq v$ then $u = u - v, x_1 = x_1 - x_2$
2.8. else $v = v - u, x_2 = x_2 - x_1$
3. **end while**
4.1. if u = 1 then return $x_1 \bmod p$
4.2. else return $x_2 \bmod p$

The architecture works as follows. The d/$\bar{i}$ control line is used to select either of division and inversion operations. This signal helps to load the x_1 register by b or 1 at the beginning of the operation. The *Start* signal controls the loading operation of u, v, p, x_1 and x_2 registers through multiplexers either by their initial values or by the intermediate results at the beginning of every clock. There are two comparators to compare the present values of u and v

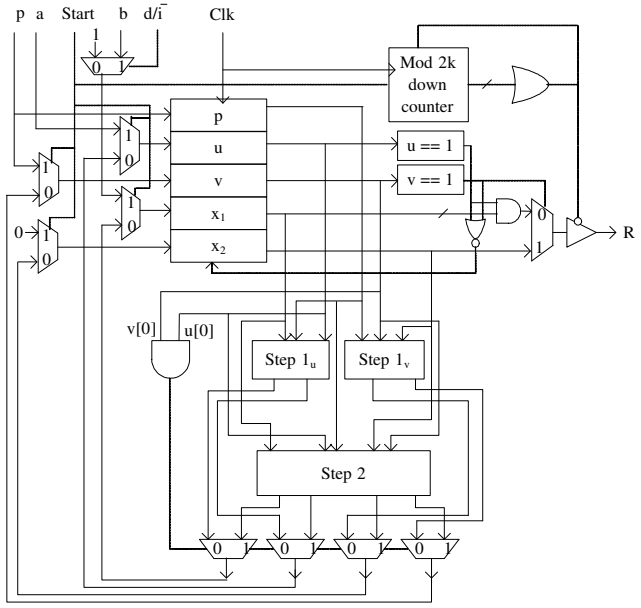

Figure 5. $GF(p)$ inverter and divider unit

registers with 1. If any one of the comparator outputs become true then at the next clock all internal registers are freezed as the final result is available in the x_1 or x_2. The blocks Step-1$_u$ and Step-1$_v$ of the architecture perform the operations within two inner while loop of the algorithm. The operations defined in step 2.7 and 2.8 of the algorithm is performed by the Step-2 block of the architecture. Design details of these two blocks are shown in Fig.6. Step-

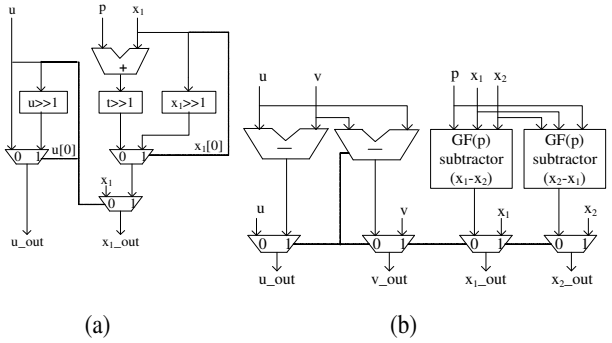

Figure 6. (a) Step-1 and (b) Step-2 block of Fig.5

1$_u$ (Fig.6a) performs divide-by-2 operation by one bit hard right shift. It uses the odd/even detector signals (u[0] and x_1[0]) to select either the modified or old values of corresponding variables at every clock. For the Step-1$_v$ we instantiate the same module by replacing the u, x_1, u_out and x_1_out by v, x_2, v_out and x_2_out respectively.

0-7695-2978-X/07 $25.00

Step-2 module (Fig.6b) consists of two k-bit magnitude subtractors, two k-bit $GF(p)$ subtractors and four k-bit 2 $\times$ 1 multiplexers. Two magnitude subtractors concurrently perform $u - v$ and $v - u$ and at the same time two $GF(p)$ subtractor perform $(x_1 - x_2)$ mod *p* and $(x_2 - x_1)$ mod *p*. The borrow out signal of $v - u$ operation, that says whether the current value of $u \geq v$, used to select its final output.

The binary inversion/division algorithm may not take exactly 2k number of clock cycles for all input values of a and *p*. In order to resist cryptanalysis timing attack each operation should take the same (maximum) number of clock cycles. Therefore at $(2k+1)^{th}$ clock the final result is available at the inverter/divider output. In the inverter/divider architecture this is achieved by the cost of one mod-2k down counter, one $\lceil \log_2 2k \rceil$-input *OR* gate and an active low enabled buffer. The output of the *OR* gate enables the active low output buffer to pass the result from x_1 or x_2 register to its output port. To resist the power analysis attack the architecture could be modified by following ways. Eliminate the freezing logic of u, v, x_1 and x_2 registers, so that they are updated at every clock. Integrate one more temporary register to hold the result from its actual time of generation to the 2k clocks. Thus the power distribution of this modified architecture will not leak any information regarding a or b.

4 The GF(p) parallel arithmetic unit

The top level architecture of proposed $GF(p)$ parallel arithmetic unit, illustrated in Fig.7, combines modulo adder, subtractor, multiplier and invertor/divider units listed in previous section. The input and output interface blocks, the result multiplexing block and the control logics have been made in-built to the architecture. The internal processing word size of the architecture is k bits long. The 160-bit FPGA implementation on Sparatan-3 board is efficiently interfaced with the parallel port of the computer.

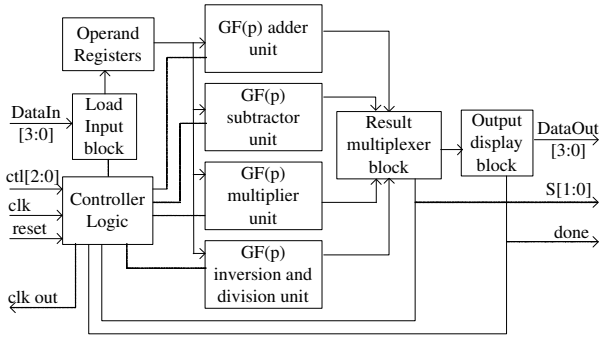

Figure 7. Top Level Architecture of $GF(p)$ parallel Arithmetic Unit

4.1 Input Output interface

The present design has 4-bit I/O to aid in the interface with the conventional parallel port of computer. The important blocks are Load Input Block and Output Display Block which performs the I/O interface between the external part of the design and actual processing blocks of the chip.

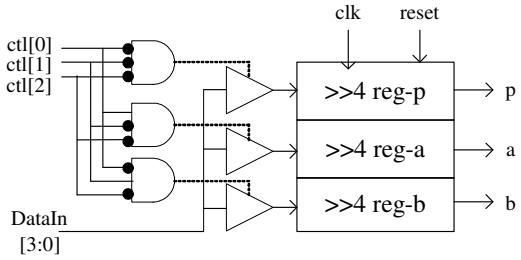

Figure 8. Load Input Block

The *Load Input Block* (Fig.8) receives the data from the 4-bit $DataIn$ port at each clock and loads the operands and modulus to the respective registers. The registers are designed as 4 bits right shift per system clock fashion. The flow of data from $DataIn$ port to these registers are controlled by the 3-bit control input lines. If maximum size of operands and modulus is k bits long then it takes $k/4$ system clocks to load one operand.

The *Output Display Block* (Fig.9) despatches the result of operations from the k-bit long output register to the 4-bit $DataOut$ port. This register works in two different modes, *load* and *shift*. At every clock if *done* signal is high then it is loaded by a new result, available to the output channel of Result Multiplexer Block (Fig.10), otherwise it shifts its present content by 4 bits toward right.

4.2 Result Multiplexer Block

Result Multiplexer Block, shown in Fig. 10, consist of four k-bit temporary registers and a four channel to one channel sequential multiplexer, where each channel is k bits long. Every register is dedicated to hold the result of a particular functional arithmetic unit. There are two bit control signals, $s[1]$ and $s[0]$, generated by the control logic, are used for controlling the sequential selection of the individual registers output channels to its output port.

4.3 Controller Logic

The state transition diagram of *Controller Logic* is shown in Fig. 11. It generates the *Start* signal for individual oper-

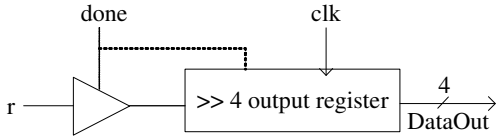

Figure 9. Output Display Block

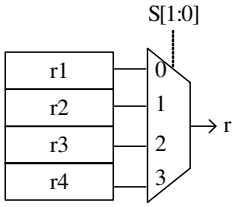

Figure 10. Result Multiplexer Block

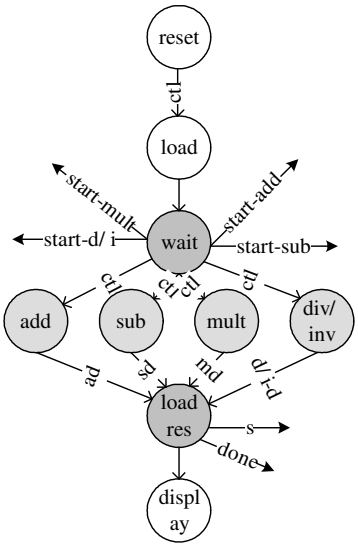

Figure 11. State diagram of controller logic

ator units and controls data flow among other active blocks of the design. This controller logic works as the master controller of the chip that generates the control signals for block level data communication. The data flow through internal data paths inside the individual active functional blocks are controlled by its own controller logic.

The *Start* signal of individual arithmetic units is generated by controller logic at a specific clock cycle depending on the user choice of current operation specified by the three control input (*ctl[2:0]*) lines. The controller generates *s[1]* and *s[0]* signals for Result Multiplexer Block and *done* signal for Output Display Unit. These three control lines also go out from the chip as output control lines. The *done* signal helps the user to detect valid output data and *s[1]* and *s[0]* guide to inform the operation for which present result is coming out.

5 Experimental Results

The architecture is implemented by Verilog. The synthesis of the top module has been done using Xilinx ISE 7.1i tool where the target device was Spartan-3 xc3s5000 FPGA with speed grade -4 and package fg900. The post-synthesis, post-map and post-place and route results for the top architecture are presented in Table 1. The behavioral, post-translate, post-map and post-place & route verilog models of the design has been verified using Modelsim XE III 6.0a simulator. And finally the design is implemented on Spartan-3 xc3s5000-fg900 board and tested by using the Logic Analyzer.

Table 1. Results for $GF(p)$ parallel arithmetic unit on Xilinx Sparatan-3 xc3s5000-5-fg900 FPGA

word length	F(MHz)	#gate	#slice	%device
32-bit	98.88	17.8K	1069	3%
64-bit	80.82	35.3K	2098	6%
96-bit	68.54	54.7K	3178	9%
128-bit	52.40	71.7K	4184	12%
160-bit	45.16	90.3K	5289	15%
192-bit	43.53	107.6K	6307	18%
224-bit	39.70	126.8K	7328	21%
256-bit	37.33	149.2K	8830	26%
384-bit	30.00	222.9K	13253	39%
524-bit	24.40	303.3K	18060	54%

Table 2. Execution time for different operations in various bit length $GF(p)$ parallel arithmetic unit

Word length	Execution time (μs)			
	Interleaved multiplication (k clk)	binary division (2k clk)	point addition (4k+5 clk)	point doubling (5k+7 clk)
32-bit	1.60	3.20	6.65	8.35
64-bit	3.20	6.40	13.05	16.35
96-bit	4.80	9.60	19.45	24.35
128-bit	6.40	12.80	25.85	32.35
160-bit	8.00	16.00	32.25	40.35
192-bit	9.60	19.20	38.65	48.35
224-bit	11.20	22.40	45.05	56.35
256-bit	12.80	25.60	51.45	64.35
384-bit	19.20	38.40	77.05	96.35
524-bit	26.20	52.40	105.05	131.35

Using this parallel arithmetic unit point addition in affine coordinate can be performed by only one modular division, two modular multiplications and 6 addition/subtraction operations and that takes exactly $4k + 5$ clock cycles, as one subtraction can be performed concurrently with some other operations. Similarly, point doubling can be performed by only one division, three multiplications and 8 addition/subtraction operations and it requires exactly $5k + 7$ clock cycles. Required execution time for different essential operations in ECC are listed in Table 2. The results are acquired from the Sparatan-3 xc3s5000-5-fg900 device at a

frequency 20MHz. It is easily be evidenced that at the maximum clock frequency listed in Table 1. the time required for these modular operations of different bit length will be much less than that of in Table 2.

A performance comparison is given in Table 3 with two other $GF(p)$ ALU those are already reported in literature. In [4] the systolic multiplier reported by Örs et al. performs modular operations in Montgomery domain, applicable to projective coordinate ECC point operations, is operated at clock frequency 91.3 MHz. It takes 74 and 70 μs per point addition and point doubling respectively for a 160-bit design.

Table 3. Clock cycles and execution time comparison for ECC point operations in $GF(p)$

Operation	word length = 160					
	#clock Cycles			Execution Time (μs)		
	Örs	Daly	Present	Örs	Daly	Present
Multiplication	3k+4	k+1	k	5.62	8.05	8.00
Inversion	9/2k^2+6k	2k	2k	1350	16.00	16.05
Division	9/2k^2+9k+4	3k+1	2k	1356	20.05	16.05
Point Addition	42k+56	5k+9	4k+5	74	40.45	32.25
Point Doubling	40k+38	6k+12	5k+7	70	48.60	40.35

The $GF(p)$ ALU, listed in [5], performs Montgomery domain arithmetics effectively useful for affine (or projective) coordinates, is operated at 20MHz clock frequency. It takes 40.45 and 48.60 μs per point addition and point doubling respectively in affine coordinate for 160-bit design.

The 160-bit design presented here, operating at 20 MHz on the Sparatan-3 xc3s5000-5-fg900 FPGA will perform the same operations in 32.25 and 40.35 μs respectively and at its highest operating frequency (45.16 MHz) it takes 14.28 and 17.87 μs respectively. In this design inputs and outputs are represented in natural binary numbers, so that it saves the input output domain conversion overhead.

6 Conclusion

The proposed $GF(p)$ parallel arithmetic unit is effectively applicable to perform all necessary modular operations for an ECC in affine (or projective) coordinates. The present design takes $8k + 14$ and $14k + 7$ clock cycles for respective point doubling and point addition in projective coordinate, where point operations are mainly based on the modular multiplication. The design is absolutely fulfilled authors initial goals that ware targeted in section 2.1. The post synthesis and post place and rout results have indicated that its implementation various bit lengths are trustfully applicable to ECC schemes those are competitive with much higher bit length RSA schemes. It is recommended to use latest FPGA (Virtex 4 or Virtex 5) to achieve better throughput using the present architecture. The design performs directly on natural binary numbers that will be helpful to exhibit its elegance in cryptographic applications.

References

[1] B. Schneier, *Applied Cryptography*, second ed., Wiley, New York, 1996.

[2] V. S. Miller, *Use of elliptic curves in cryptography*, Adv. Cryptogr. Crypto'85, 1985, pages 417–426.

[3] N. Koblitz, *Elliptic curve cryptosystems*, Math. Comp. Vol. 48, 1987 pages 203–209.

[4] S. B. Örs, L. Batina, B. Preneel, J. Vandewalle, *Hardware implementation of elliptic curve processor over GF(p)*, Proceedings of the Application-Specific Systems, Architectures, and Processors ASAP, 2003, pages 433-443.

[5] A. Daly, W. Marnane, T. Kerins, E. Popovici, *An FPGA implementation of a $GF(p)$ ALU for encryption processors*, Microprocessors and Microsystems, Vol. 28, 2004, pages 253–260

[6] P. L. Montgomery, *Modular multiplication without trial division*, Math. Comput. Vol. 44, 1985, pages 519–521.

[7] G. R. Blakley, *A computer algorithm for the product AB modulo M*, IEEE Transactions on Computers, Vol. 32, No. 5, pages 497–500, May 1983.

[8] K. R. Sloan, Jr. Comments on "*A computer algorithm for the product AB modulo M*". IEEE Transactions on Computers, Vol. 34, No. 3, pages 290–292, March 1985.

[9] I. Blake, G. Seroussi, N. Smart, *Elliptic Curves in Cryptography*, London Mathematical Society Lecture Note Series 265, Cambridge University Press, 2000.

[10] D. Hankerson, A. Menezes, S. Vanstone, *Guide to Elliptic Curve Cryptography*, Spinger, United States, 2003.

[11] A. Daly, W. Marnane, T. Kerins, E. Popovici, *Fast modular division for application in ECC on reconfigurable logic* Field-Programmable Logic and ApplicationsFPL 2003, (LNCS 2778), 2003, pages 786-795.

[12] G. Orlando, C. Paar, *A scalable GF(p) elliptic curve processor architecture for programmable hardware* CHES 2001, LNCS 2162, pages 348–363, 2001.

[13] P. C. Kocher, *Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems*, *CRYPTO'96*, pages 104–113.

0-7695-2978-X/07 $25.00