DOI: 10.3745/KIPSTC.2010.17C.2.145

개선된 이진 확장 GCD 알고리듬 기반 GF(2^{163})상에서 Iterative 나눗셈기 설계

강 민 섭[†] · 전 병 찬[††]

요 약

본 논문에서는 표준기저(standard basis) 표기법을 이용하여 GF(2^{163}) 상에서개선된 나눗셈 알고리듬을 제안하고, 제안한 알고리듬을 기반으로 한 반복 하드웨어 구조(iterative hardware structure)를 갖는 고속 나눗셈기를 설계한다. 제안한알고리듬은 이진 확장 GCD 알고리듬을 기본으로 하고 있으며, 모듈러감소 (modular reduction)를 위한 모든 산술연산은 기존의 방법과 달리 하나의 while루프 내에서 수행된다. 제안된 알고리듬을 기본으로 하여 설계된 나눗셈기는 모듈러 연산을 위한 각 모듈이 하나의 클럭에 의해서제어되므로 계산 속도가 매우 빠르다. 여기에서 사용하는 감소 다항식(reduction polynomial)은SEC2 (Standards for Efficient Cryptography) 에서 권장하는 $f(x)=x^{163}+x^7+x^6+x^3+1$이며, 차수(degree) m은 163을 사용한다. 제안한 알고리듬은 Verilog HDL(Hardware Description Language)을 사용하여 FPGA로 구현되었으며, Xilinx-VirtexII XC2V8000 FPGA 상에서 85MHz로 동작함을 확인하였다. 또한, 구현 결과 및 성능 평가를 통하여 제안한 알고리듬의 종래의 두 알고리듬보다 성능이크게 개선됨을 보인다.

키워드 : 표준기저 표기법, 이진확장 GCD 알고리듬, 고속 나눗셈 알고리듬, 반복 구조, Verilog HDL, FPGA

Design of Iterative Divider in GF(2^{163}) Based on Improved Binary Extended GCD Algorithm

Min-Sup Kang[†] · Byong-Chan Jeon[††]

ABSTRACT

In this paper, we first propose a fast division algorithm in GF(2^{163}) using standard basis representation, and then it is mapped into divider for GF(2^{163}) with iterative hardware structure. The proposed algorithm is based on the binary ExtendedGCD algorithm, and the arithmetic operations for modular reduction are performed within only one "while-statement" unlike conventional approach which uses two "while-statement". In this paper, we use reduction polynomial $f(x)=x^{163}+x^7+x^6+x^3+1$ that is recommended in SEC2(Standards for Efficient Cryptography) using standard basis representation, where degree m = 163. We also have implemented the proposed iterative architecture in FPGA using Verilog HDL, and it operates at a clock frequency of 85 MHz on Xilinx-VirtexII XC2V8000 FPGA device. From implementation results, we will show that computation speed of the proposed scheme is significantly improved than the existing two approaches.

Keywords : Standard Basis Representation, Binary Extendedalgorithm, Fast Division Algorithm, Iterative Structure, Verilog HDL, FPGA

1. Introduction

ECC (Elliptic Curve Cryptography) among all known public key cryptography systems has been widely used in wireless application. In ECC algorithm, the most time consuming part is scalar multiplication that can be computed by point addition and doubling operations. In either case, major operations for time consuming are field multiplication and field inversion, while squaring and field addition have less computation time [1-3].

Several algorithms have been introduced for computing

※ This work was supported by the research program of 2009 year from Anyang University, Kyeonggi-Do SMBA, and IDEC, KAIST in Korea.
† 종신회원 : 안양대학교 컴퓨터공학과 교수
†† 준 회 원 : 안양대학교 컴퓨터공학과 석사과정
논문접수 : 2009년 7월 27일
수 정 일 : 1차 2009년 9월 16일, 2차 2009년 9월 29일
심사완료 : 2009년 9월 30일

field inversion/division operation based on the Extended Euclidean algorithm [2-6]. Although these algorithms can be easily implemented using software programs on a general-purpose computer, they would be slow and inefficient for public key cryptosystems which is used a very large field [3, 4]. In order to resolve these problems, the first sublinear time parallel algorithm that uses a polynomial number of processors has been introduced by Kannan Miller, and Rudolph [7], and a parallel extended GCD (Greatest Common Divisor) algorithm has been presented, which uses the concurrent-read concurrent-write (CRCW) parallel RAM (PRAM) model of computation [8].

The binary Extended GCD algorithm was known that it is simple, but it has difficulty of hardware implementation [2-4]. In [3], an efficient algorithm is presented based on a modified version of the Euclid's GCD algorithm. Although this algorithm is suitable for implementing GF divider with systolic array structure, it is still time-consuming. Thus a fast algorithm that can perform arithmetic operation in fewer clock cycles [9, 10] is required, which is suitable for iterative hardware implementation.

In this paper, we propose the hardware implementation of iterative divider based on a fast division algorithm in $GF(2^{163})$ using standard (Polynomial) basis representation. The proposed algorithm is based on the binary Extended algorithm, and the arithmetic operations are performed for modular reduction in only one while-statement unlike conventional approach. Through implementation results, we have shown that the computation speed of our approach is significantly improved than that of the conventional approaches [2, 4] due to reduction of the number of clock cycles used.

This paper is organized as follows. Section 2 introduces problems of the conventional two algorithms for performing field division operation based on the Extendedalgorithm. Section 3 describes a proposed division algorithm and fast iterative divider design for speeding-up division operation in $GF(2^{163})$. In section 4, simulation results and performance analysis are given, which is based on the improved division algorithm. Finally, conclusion is given in section 5.

2. Related Works

Let $A(x)$ and $B(x)$ be the polynomial representations of two elements in $GF(2^m)$, $G(x)$ be the irreducible polynomial with degree m, where $B(x)$ 0, and $P(x)$ be the division result for $A(x)/B(x)$ mod $G(x)$. Then we have

$$A(x) = a_{m-1}x^{m-1}+a_{m-2}\,x^{m-2}+\ \cdots\ +a_1x+a_0$$
$$B(x) = b_{m-1}x^{m-1}+b_{m-2}x^{m-2}+\ \cdots\ +b_1x+b_0$$
$$G(x) = x^m+g_{m-1}x^{m-1}+g_{m-2}x^{m-2}+\ \cdots\ +g_1x+g_0$$
$$P(x) = p_{m-1}x^{m-1}+p_{m-2}x^{m-2}+\ \cdots\ +p_1x+p_0.$$

Each coefficient of these polynomials is binary digit 0 or 1, and all arithmetic operations are performed by taking coefficients of the results mod 2. When $A(x) = 1$, $P(x)$ is called the multiplicative inverse of $B(x)$. The binary Extended GCD algorithm is an efficient way of calculating modular division, $P(x)$. To compute the modular division, the algorithm is based on the following three facts: if R and S are both even, then GCD (S, R) = xGCD (S/x, R/x), if R is even and S is odd, then GCD (S, R) = GCD (S, R/x), if R and S are both odd, then GCD (S, R) = GCD ((S-R)/x, R) [2, 4].

A procedure for performing the inversion operation of "1/B(x) mod G(x)" over $GF(2^m)$ is shown in (Fig. 1), which is called binary Extended GCD algorithm [2].

This algorithm can be divided into three steps written in (1), (2) and (3). In step (1), to calculate "U/ x mod G", the algorithm examines the LSB (Least Significant Bit) of U to determine whether it is even (u_0 = 0) or odd ($u_0 \neq 0$). If it is even, the algorithm performs U/ x, otherwise it

```
Input : G(x), A(x), B(x)
Output : U has P(x) = A(x)/B(x) mod G(x)
Initialize : R = B(x), S=G= G(x), U = A(x), V = 0
while S ≠0 do
(1) while r == 0 do
        R = R/x
        if u == 0 then U = U/x
        else U = (U+G)/x end if
    end while
(2) while s0 == 0 do
        S = S/ x
        if v0 == 0 then V = V/ x
        else V = (V+G)/ x end if
    end while
(3) if S ≥ R then
        (S, R) = (S+R, R);
        (V, U) = (U+V, U);
    else
        (S, R) = (S, S+ R);
        (V, U) = (V, U+V);
    end if
end while
```

(Fig. 1) Binary Extended GCD algorithm over $GF(2^m)$

performs (U+G)/ x. In this algorithm, modular reduction is accomplished by a simple shift operation.

Now, we consider that this algorithm is implemented in iterative hardware structure. By using first clock cycle, the initial parameters stored in four registers of a size of 163 bits are transferred to their outputs of control block, and then in the module of step (1), the modular reduction of variable sets, (R, U) is performed depending on control bits of r_0 and u_0. Continuously, the updated values are fed to the same registers, and then the control bit r_0 is tested again in the module. The variable sets can be also updated if the bit r_0 is even, where one clock cycle is required. As a result, we can see that the number of clock cycle which will be used is the same as the iteration times in the module (see Table 1). In the module of step (2), modular reduction process is also performed for variable sets of (S, V) and a division result is at least obtained from the use of one clock in step (3). Note that U will have the division result P(x) = A(x)/B(x) mod G(x) if we replace U = 1 by U = A(x) [4].

<Table 1> shows an example for computing division in GF(2^4) based on the algorithm of (Fig. 1), G(x) = x^4+x+1, A(x) = x^3+x^2+x, and B(x) = x^3+x+1.

As shown in <Table 1>, we assume that each parameter is initialized as S = G(x), R = B(x), and U = A(x), where V = 0. Two items of Itr and #Clk represent iteration times that executed in outer while-statement and the number of clock cycle, respectively. For 1st iteration, 3 clocks are totally used since one cycle is used in each step of (1), (2), and (3). For 2nd iteration, step (2) takes three cycles since a while-statement repeats three times. Thus, 5 clocks are totally used after 2nd iteration. The algorithm terminates after 4 iterations, and then 16 clocks are needed for obtaining final division result, U = x+1.

〈Table 1〉 An example for computing division based on (Fig. 1)

Itr	Step	#Clk	S (=G(x))	R (=B(x))	V (=0)	U (=A(x))
1	(1)	1	x^4+x+1	x^3+x+1	0	x^3+x^2+x
	(2)	2	x^4+x+1	x^3+x+1	0	x^3+x^2+x
	(3)	3	x^4+x^3	x^3+x+1	x^3+x^2+1	x^3+x^2+x
2	(1)	4	x^4+x^3	x^3+x+1	x^3+x^2+1	x^3+x^2+x
	(2)-1	5	x^3+x^2	x^3+x+1	x^2+x+1	x^3+x^2+x
	(2)-2	6	x^2+x	x^3+x+1	x^3+x	x^3+x^2+x
	(2)-3	7	x^2+x	x^3+x+1	x^2+1	x^3+x^2+x
	(3)	8	x+1	x^3	x^2+1	x^3+x+1
3	(1)-1	9	x+1	x^2	x^2+1	x^3+x^2
	(1)-2	10	x+1	x	x^2+1	x^2+x
	(1)-3	11	x+1	1	x^2+1	x+1
	(2)	12	x+1	1	x^2+1	x+1
	(3)	13	x	1	x^2+1	x+1
4	(1)	14	x	1	x+1	x+1
	(2)	15	1	1	x+1	x+1
	(3)	16	0	1	0	x+1

3. Proposed Algorithm and Fast Divider Design

3.1 Division algorithm

To speeding-up division operation in GF(2^{163}), we present an advanced division algorithm without affecting the basic functionby modifying the binary Extended GCD algorithm described in (Fig. 1) [2]. (Fig. 2) shows the proposed algorithm for performing fast division operation in GF(2^{163}).

Now, we reconsider classical binary Extended algorithm described in (Fig. 1). In order to perform GCD operation, in step (1), A(x) and B(x) are computed depend on the control bits of $u_0$and r_0, respectively. Continuously G(x) and V are computed after completing the check of two conditions, s_0 and v_0, respectively. Finally, the computation of both GCD (S, R) and GCD (V, U) is performed by comparing S to R in step (3).

```
Input : G(x), A(x), B(x)
Output : U has P(x) = A(x) / B(x) mod G(x)
Initialize : R = B(x), S =G= G(x), U = A(x), V = 0
while S ≠0 do
(1) while r0 == 0 or s0 == 0 do
      if r0 == 0 then
          R = R/ x
          U = (U+u0・G)/ x
      end if
      if s0 == 0 then
          S = S/ x
          V = (V+v0・G)/ x
      end if
    end while
(2) if S ≥ R then
      (S, R) = (S+R, R);
      (V, U) = (U+V, U);
    else
      (S, R) = (S, S+R);
      (V, U) = (V, U+V);
    end if
end while
```

(Fig. 2) Proposed algorithm for fast division in GF(2^{163})

For hardware implementation of this algorithm, a number of processing time will be needed because final results are obtained in step (3) after completing the check of each condition in two while-statements of (1) and (2) every iteration routine.

In the proposed algorithm, only one while-statement (see step (1)) is first executed, which is controlled by two bits of r_0 and s_0,and then two if-statements perform modular reduction within the while-statement. Thus, modular reduction for (R, U) is performed in statement “if r_0 == 0 then” and (S, V) is also performed in statement “if s_0 == 0 then” depend on the conditions of r_0 and s_0, respectively.

If the proposed division algorithm is implemented in an iterative hardware structure, these two if-statements in while-statement can be constructed to each independent module which is controlled by same clock signal. As a result, processing time is very fast since each input variable which is need to obtain both GCD (S, R) and GCD (V, U) is calculated by using the same clock signal.

〈Table 2〉 An example of computing division in GF(2^4) based on (Fig 2)

Itr	Step	#Clk	S	R	V	U
1	(1)	1	x^4+x+1	x^3+x+1	0	x^3+x^2+x
	(2)	2	x^4+x^3	x^3+x+1	x^3+x^2+x	x^3+x^2+x
2	(1)-1	3	x^3+x^2	x^3+x+1	x^2+x+1	x^3+x^2+x
	(1)-2	4	x^2+x	x^3+x+1	x^3+x	x^3+x^2+x
	(1)-3	5	$x+1$	x^3+x+1	x^2+1	x^3+x^2+x
	(2)	6	$x+1$	x^3	x^2+1	x^3+x+1
3	(1)-1	7	$x+1$	x^2	x^2+1	x^3+x^2+x
	(1)-2	8	$x+1$	x	x^2+1	x^3+x
	(1)-3	9	$x+1$	1	x^2+1	$x+1$
	(2)	10	x	1	x^2+x	$x+1$
4	(1)	11	1	1	$x+1$	$x+1$
	(2)	12	0	1	0	$x+1$

<Table 2> demonstrates the proposed algorithm of (Fig. 2) for computing divisions in GF(2^4), where G(x) = x^4+x+1, A(x) = x^3+x^2+x and B(x) = x^3+x+1, which are the same parameters used in <Table 1>.

As described in <Table 2>, U represents A(x) = $x+1$ as the final division result, and 12 clocks are used after completing 4 iterations while in the conventional two different algorithms, 16 clocks [2] and 15 clocks [4] are used to complete the division operation, respectively. <Table 3> shows the main distinctive feature of the conventional and proposed algorithms.

3.2 Design of fast divider with iterative structure

(Fig. 3) shows the block diagram of fast divider for GF(2^{163}) with iterative architecture on the basis of the proposed division algorithm.

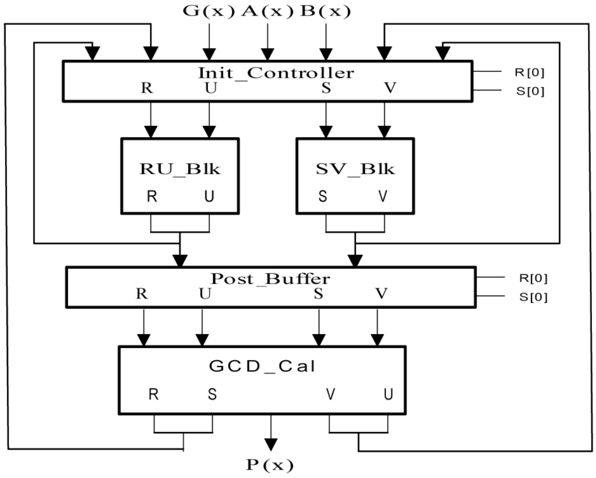
This divider has three inputs of G(x), A(x), and B(x), and it obtains P(x) as the final division results after performing operation of “A(x)/B(x) mod G(x)”. Init_Controller block consists of four registers of a size of 163 bits for storing input vectors and control signals for controlling the system. Both RU_Blk and SV_Blk are reduction modules to play an important role for performing modular reduction of (R, U) and (S, V), respectively, and they correspond to 1st if-statements and 2nd if-statements in step (1) of (Fig. 2), respectively. These two modules are operated by one clock to execute modular operation while two clock cycles are required for this operation in conventional approach [2].

Thus, the proposed architecture requires no more than 3m clock cycles (after m iterations) to yield the final division result. (Fig. 4) and (Fig. 5) illustrate detailed block diagrams of RU_Blk and SV_Blk shown in (Fig. 3), respectively.

These two reduction modules can be directly designed from step (1) of our algorithm using three operators of

〈Table 3〉 Comparison of conventional and proposed algorithms

Items \ Algorithms		Ref. [2]	Proposed
# while-statement		3	2
Processing sequence for finding GCD		1. In 1st while-statement, R and U are first calculated 2. In 2nd while-statement, S and V are calculated	1. In 1st if-statement, R and U are calculated 2. In 2nd if-statement, S and V are calculated
Total iteration times	Condition of (R=0) or (S=0)	2m+2	3m
	Condition of (R≠0) and (S≠0)	m(m+2)	3m
# Clock cycle in GF(2^4)		16	12



(Fig. 3) Architecture of fast divider for GF(2^{163})

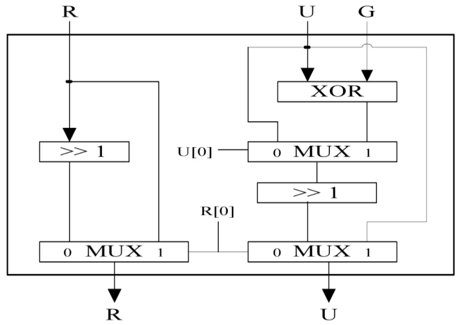
XOR, MUX, and Shifter (>>). Note that computation results of RU_Blk and SV_Blk are simultaneously transferred into Post_Buffer block using one clock.

(Fig. 6) shows a detailed block diagram of GCD_Cal shown in (Fig. 3). It also can be directly derived from step (2) of our algorithm using four operators of XOR, MUX, INV and CMP (Comparator), where INV (Inverter) is used for handling else-statement described in step (2).

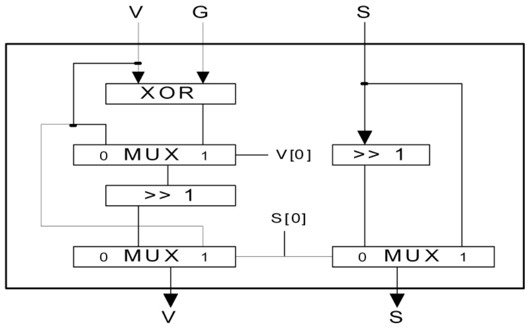
This module performs mainly arithmetic operations for comparison (CMP) and addition (XOR) using four parameters calculated in previous modules, and it outputs operation results after a given clock cycles.

For hardware implementation, the previous division algorithm[2] needs several processing time because reduction operation for variable sets, (U, R) is first performed in step (1) of 1st while-statement and then variable sets of (S, V) are caculated in step (2) of 2nd while-statement.

It should be noted here that two modules for perorming modular reduction in designed divider are controlled by same common clock while conventional approaches [2] requires different two clcok signals for this modular reduction.



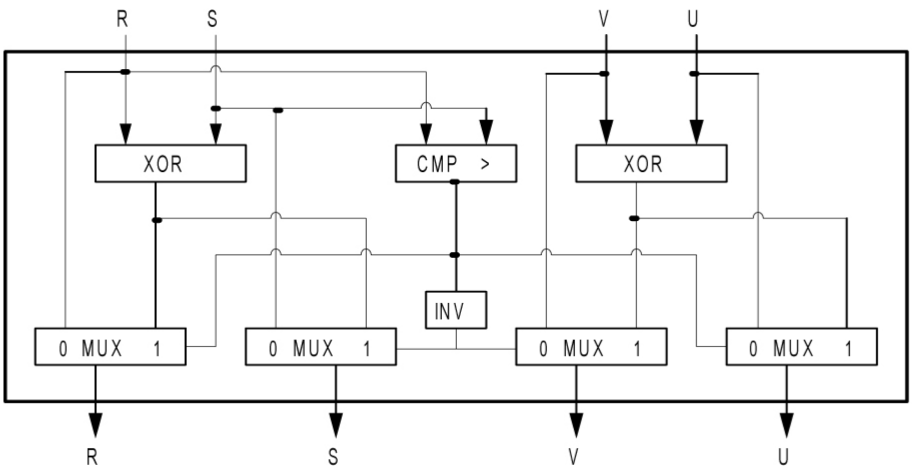
(Fig. 4) Block diagram of RU__Blk



(Fig. 5) Block diagram of SV__Blk

4. Implementation Results

In this paper, we use reduction polynomial f(x), f(x) $=x^{163}+x^7+x^6+x^3+1$, that is recommended in SEC2 (Standards



(Fig. 6) Block diagram of GCD__Cal

for Efficient Cryptography) [11] using standard basis representation, where an irreducible binary polynomial of degree m=163 is used.

The proposed division algorithm was described using Verilog HDL at the Behavioral level, and it has been successfully implemented with Xilinx FPGA using the ISE 6.x. tool. To verify functionality of the designed divider, timing simulation is performed using Xilinx simulator and Mentor Graphics ModelSim^{Tm}. (Fig. 7) shows the timing simulation result of the designed divider for $GF(2^{163})$ using Xilinx simulator.

To compare performance of conventional algorithm to the proposed algorithm, the set of input data used for this simulation[4] is as follows: $A(x) = x^5+x^3+x+1$, $B(x) = x^6+x^3+x^2+x$, $G(x) = x^8+x^4+x^3+x^2+1$.

In (Fig. 7), a_x, b_x, g_x, and p_x represent $A(x)$, $B(x)$, $G(x)$, and $P(x)$, respectively, which have each 163 bits. Through simulation result, we can see that $P(x)=x^7+x^4+x^2+1$ (95 in Hexa-decimal) is obtained as the output result after 23ns delays while the result of the conventional approach[2] is obtained after 33ns delays. Implementation result is summarized in <Table 4>, which is obtained from logic synthesis using Xilinx ISE 6.x. tool, where FPGA target device used is Xilinx-VirtexII XC2V8000ff1152-5.

<Table 5> shows comparison of synthesis results between three diffenrent algorithms which are implemented with Xilinx-VirtexII FPGA device using the ISE 6.x. tool.

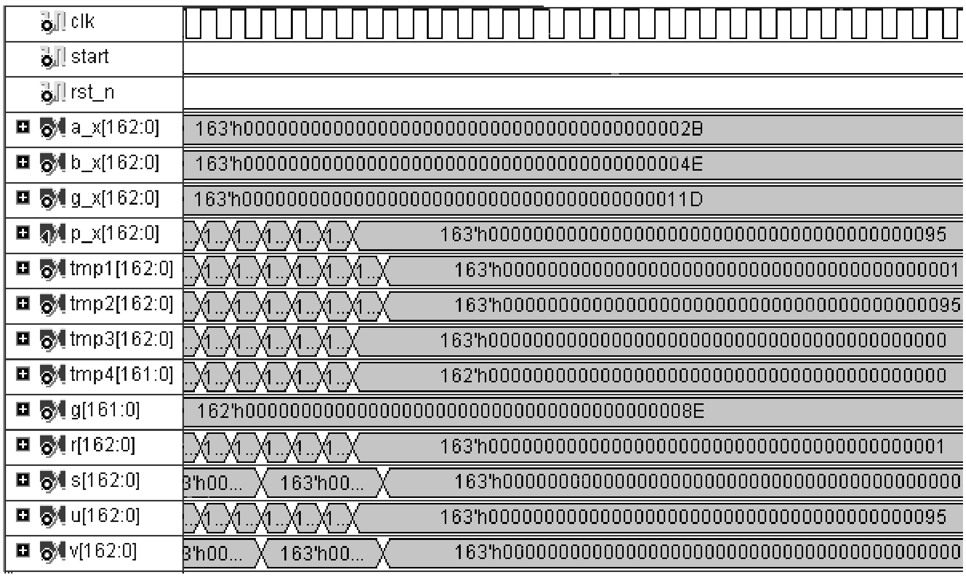
In order to provide a fair comparison, we have implemented directly conventional two algorithms [2 ,4] in hardware with Xilinx-VirtexIIFPGA device using Verilog HDL. As can be seen from <Table 5>, we can see that the proposed method is not only shorter critical path delay(Clock period) than conventional approaches, but also hardware overhead is smaller than two approaches [2, 4].

<Table 6> shows the comparison of the number of clock for three different algorithms which are obtained from implementation results..

In comparison of "#Clocks" of <Table 6>, we showed the proposed method is approximately reduced by 26 % and 47 % for $GF(2^{32})$, and by 26 % and 40 % for $GF(2^{64})$, respectively, compared to two conventional methods [2, 4]. Also, <Table 7> shows the comparison oftotal delays for three different algorithms.

〈Table 4〉 Implementation result

HDL Synthesis	# Registers	5 (163-bit)
	# Comparator	2 (163-bit)
	# Muxs	3 (163-bit 4-to-1)
	# Xors	2 (163-bit xor2)
Cell Usage	# BELS	2,297
	# FFs/Lats	817
	No. of Slices	967
	No. of LUTs	1,644
Timing Summary	Min. period	11.7ns (85MHz)
	Min. input arrival time	6.7ns
Total equivalent gates		18,988



(Fig. 7) Timing simulation result

〈Table 5〉 Comparison of synthesis results

Items \ Algorithms	Ref. [2]	Ref. [4]	Proposed
Clock period(ns)	12.6	13.4	11.7
#Slices	1,092	1,034	967

〈Table 6〉 Comparison of the number of clock

Items \ Algorithms		Ref. [2]	Ref. [4]	Proposed
#Clocks	GF(2^8)	33	33	23
	GF(2^{16})	55	64	42
	GF(2^{32})	92	128	68
	GF(2^{64})	207	256	154

〈Table 7〉 Comparison of total delays (ns)

Items \ Algorithms		Ref. [2]	Ref. [4]	Proposed
Total delay(ns)	GF(2^8)	346	386	260
	GF(2^{16})	693	858	525
	GF(2^{32})	1,159	1,715	850
	GF(2^{64})	2,608	3,430	1,925

In <Table 7>, “Total delay” means overall delay time required to obtain final division result. In comparison of total delay time, compared to two conventional methods [2, 4], the proposed method is approximately improved by 26% and 50% for GF(2^{32}), and by 24% and 44% for GF(2^{64}), respectively. This is because that the number of the used clocks is dramatically reduced compared to conventional two approaches. The designed divider operates at a clock frequency of 85 MHz on Xilinx-VirtexII XC2V8000 FPGA device.

5. Conclusion

A fast division algorithm in GF(2^{163}) using standard basis representation has been presented based on the binary Extended GCD algorithm, where reduction polynomial $f(x)=x^{163}+x^7+x^6+x^3+1$ is used[11]. The proposed algorithm has been implemented in divider for GF(2^{163}) of the iterative hardware structure with less latency on a FPGA. Through implementation results, we have shown that the computation speed of our approach is significantly improved than that of two conventional approaches [2, 4] due to reducing the number of the used clocks.

The designed divider for GF(2^{163}) operates at a clock frequency of 85 MHz on Xilinx-VirtexII XC2V8000 FPGA device. The proposed hardware structure is suitable for high-speed cryptographic applications such as elliptic curve cryptosystem.

References

[1] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 2nd Edition, New Jersey, Prentice Hall Inc., 1999.

[2] D. E. Knuth, *The Art of Computer Programming: Semi-numerical Algorithms*, Addison-Wesley, 3rd ed. Reading, MA, 1998.

[3] J. Guo, and C. Wang, “Systolic Array Implementation of Euclidian's Algorithm for Inversion and Division in GF,” *IEEE Trans. Computers*, Vol.47, No.10, Oct., pp.1161-1167, 1998.

[4] C.-H. Kim, S.-H. Kwon, J.-J. Kim, and C.-P. Hong, “A Compact and Fast Division Architecture for a Finite Field,” *Proc. ICCSA2003, LNCS*, Vol.2667, pp.855-864, Aug., 2003.

[5] N. Sklavos, K. Papadomanolakis, P. Kitsos and O. Koufopavlou, “Euclidean Algorithm VLSI Implementations,” *Proc.IEEE-ICECS'02*, Vol. II, pp. 557-560, Sep., 2002.

[6] H. Brunner, A. Curiger, and M. Hofstetter, “On Computing Multiplicative Inverses in GF(2^m),” *IEEE Trans. on Computers*, Vol.42, No.8, pp.1010-1015, Aug., 1993.

[7] R. Kannan, G. Miller, and L. Rudolph, “Sublinear Parallel Algorithm for Computing the Greatest Common Divisor of Two Integers,” SIAM Journal on Computing, Vol.16, No.1, pp.7-16, 1987.

[8] Sidi Mohamed Sedjelmaci, “A Parallel Extended GCD Algorithm,” J. of Discrete Algorithms, Vol.6, No.3, pp.526-538, 2008.

[9] A. Daly, W. P. Marnane, T. Kerins, and E. Popovici, “Fast Modular Division for Application in ECC on Reconfigurable Logic,” *13th International Conference FPL 2003,* pp.786-795, Sep., 2003.

[10] G. M. de Dormale, P. Bulens, and J.-J. Quisquater, “Efficient Modular Division Implementation (ECC over GF(p) Affine Coordinates Application),” 14th International Conference FPL 2004, 23-240, Aug., 2004.

[11] *Certicom Research*, “SEC2: Recommended Elliptic Curve Cryptography Domain Parameters,” 1999.

강 민 섭

e-mail : mskang@anyang.ac.kr

1979년 광운대학교 전자통신공학과(학사)

1984년 한양대학교 전자공학과(공학석사)

1992년 일본 오사카대학교 전자공학과(공학박사)

1984년~1992년 한국전자통신연구원 선임연구원

2001년 University of California, Irvine 전기전자공학과 객원연구원

1993년~현 재 안양대학교 컴퓨터공학과 교수

관심분야 : VLSI 테스트, 암호프로세서 설계, 신호처리, 네트워크 보안, RFID/USN

전 병 찬

e-mail : cad_jbc@naver.com

2008년 안양대학교 컴퓨터공학과(학사)

2008년~현 재 안양대학교 컴퓨터공학과 석사과정

관심분야 : 암호 프로세서 설계, 네트워크 보안, RFID/USN