A NEW RECONFIGURABLE BIT-SERIAL SYSTOLIC DIVIDER FOR $GF(2^M)$ AND $GF(P)$.

Aaron E. Cohen and Keshab K. Parhi

University of Minnesota Twin Cities
Department of Electrical and Computer Engineering
{aecohen, parhi}@ece.umn.edu

ABSTRACT

This paper focuses on the design of a new dual field divider that can achieve performance of $1/m$ throughput. This dual field division unit can operate at 118 MHz with a latency of $7m - 2$ cycles and has an area requirement 15 $XOR2$, 40 $AND2$, 29 $MUX2$, and 7 INV gates per processing element with a total of $2m$ processing elements. It is intended to be used in an Elliptic Curve Crypto-Accelerator for $GF(2^m)$ and $GF(p)$. The actual performance for scalar point multiplication in $GF(2^{571})$ running at 100 MHz would be 20.4 kP/s. The actual performance for scalar point multiplication in $GF(p)$ with $|p| = 521$ running at 100 MHz would be 24.4 kP/s.

1. INTRODUCTION

In the mid 1980's the elliptic curve cryptography (ECC) was developed independently by Miller and Koblitz. This new invention allows public key cryptosystems to be developed for smaller key sizes but with comparable security strength to the RSA cryptosystem. This advantage, smaller key size with similar security strength, does not come free. Elliptic curve cryptosystems have higher hardware complexity than systems built for the RSA Cryptosystem. One of the additional mathematical requirements for elliptic curve cryptosystems is modular division, also known as finding the inverse in a field. This can be accomplished with one of the Extended GCD algorithms. These algorithms are extensions to regular GCD algorithms.

The binary GCD algorithm [1] is well known to lend itself to fast hardware implementations. Therefore it was only natural that an extension of the binary GCD algorithm be implemented to perform division in Galois fields such as $GF(2^m)$ or $GF(p)$. To the best of our knowledge reconfigurable dual field, $GF(2^m)$ and $GF(p)$, modular dividers have not been previously designed.

The importance of Modular Division (i.e. division in $GF(p)$) is driven by cryptographic signature algorithms. Similarly the new public key algorithms based on elliptic curve cryptography are driving the importance of large scale division units in both $GF(2^m)$ and $GF(p)$. The difficulty with routing signals for large scale implementations have led designers to look for various methods to avoid this problem such as systolic array architectures. Naturally systolic architectures [2] are one of the more efficient ways to implement the extended binary GCD algorithm in hardware.

This paper is organized as follows. Section II presents previous research in the area of GCD hardware implementations. Section III describes a general background on how the binary GCD algorithm works. Section IV presents the proposed dual field divider. Then Section V provides a conclusion and a brief discussion of potential future research ideas.

2. RELATED WORK

The pioneering work in systolic arrays and the binary GCD hardware implementations was accomplished by Brent and Kung [3]. Their design lacked the ability to compute the inverse and it lacked the ability to switch its computation from either $GF(2^m)$ or $GF(p)$ to the other field.

Recently, an implementation of the MSB first divider for $GF(2^m)$ was first presented in [4]. This systolic array is based on the Extended Euclidean GCD Algorithm [1]. This architecture achieves a throughput of $1/m$, with a critical path of $T_{AND2} + 3T_{XOR2} + T_{MUX2}$, latency of $5m - 4$, and area requirements of $2m$ processing elements with each containing 11 $MUX2$, 7 $AND2$, and 5 $XOR2$.

More recently, the LSB first divider unit was designed in [5] for $GF(2^m)$. The LSB first divider is a systolic array based on the Extended Binary GCD Algorithm [1]. This architecture achieves a throughput of $1/m$, with a critical path of $T_{AND2} + 2T_{XOR2}$, latency of $5m - 2$, and area requirements of $2m$ processing elements with each containing 7 $MUX2$, 12 $AND2$, 1 $OR2$, 4 $XOR2$, and 7 INV.

There are alternative inverse algorithms designed to work with the N-Residues used primarily with Montgomery Multipliers [6]. These are referred to as the *Almost Montgomery Inverse* architectures in [7–9].

3. BACKGROUND

The extended binary GCD algorithm dates back to antiquity. It is only recently that it has become more important due to its relative simplicity for conversion to hardware implementations. A simple explanation of the binary GCD algorithm is listed below.

Algorithm: Binary GCD Algorithm [1]

K1. [Initialize] Set $c \leftarrow 0$.
K2. [Done?] If $v = 0$, terminate with $|u|$ as the answer.
K3. [Make v odd] Set $v \leftarrow v/2$ and $c \leftarrow c + 1$ zero or more times, until v is odd.
K4. [Make $c \leq 0$] If $c > 0$, interchange $u \leftrightarrow v$ and set $c \leftarrow -c$.
K5. [Reduce.] Set $w \leftarrow (u + v)/2$. If w is even, set $v \leftarrow w$; otherwise set $v \leftarrow w - v$. Return to step K2.

The binary GCD algorithm as well as the extended binary GCD algorithm require the following three rules for calculating the GCD.

Table 1: GCD Simplification Rules

Rule	u	v	$\frac{(u+v)}{2}$	$GCD(u,v)$ =
1	odd	even		$GCD(u, v/2)$
2a	odd	odd	even	$GCD(u, (u+v)/4)$
2b			odd	$GCD(u, (u-v)/4)$
3	even	even		$2 * GCD(u/2, v/2)$

The main idea is when adding or subtracting in $GF(p)$ it is not guaranteed that the number of bits to represent the result will be less than the number stored in v. However if the result of the division after the addition or subtraction is even then the number of bits to represent the result will be reduced and hence our new modified Binary GCD algorithm will terminate. It can be proved that the algorithm will finish in $2m = 2\log_2(max(u, v)) + 2$ iterations because it is necessary to swap u and v such that the smaller odd number is stored in u until v is reduced.

4. DISCUSSION

4.1. New Modified Extended Binary GCD Algorithm

The key requirements of the New Modified Extended Binary GCD Algorithm is first U must be odd at all times in the algorithm. Otherwise, rule 3 must be applied which can be implemented easily by shifting.

Algorithm: New Modified Extended Binary GCD Algorithm

Inputs: A, B, G
Outputs: $R = A/B \bmod G$
Initialize: $U = G, V = B, S = A, R = 0$, state = 0, count = 0

```
for i = 1 to 2m do
    UpV = (U + V); UmV = (U − V);
    RpS = (R + S); RmS = (R − S);
    if (mode == 1) then
        UpV = UpV/2; UmV = UmV/2;
        if (RpS[0] == 1) then RpS = (RpS − G);
        RpS = RpS/2;
        if (RmS[0] == 1) then RmS = (RmS + G);
        RmS = RmS/2;
    if (state == 0) then
        count = count + 1;
        if (V[0] == 1) then
            state = 1; U = V; R = S;
    else
        count = count − 1;
    if (V[0] == 1) then
        if (UpV[0] == 1) then
            V = UpV; S = RpS;
        else
            V = UmV; S = RmS;
    if (count == 0) then state = 1;
    if (S[0] == 1) then S = S + G;
    V = V/2; S = S/2;
```

It is important to note that the result may not be in the correct range $(0 \leq R < p)$ because of the lack of intermediate sign testing. Therefore for other functional units it may be necessary to bring the result into the correct range $(0 \leq R < p)$. This can be accomplished by a correction phase.

4.2. Systolic Array Implementation

Systolic arrays [2] contain two important features which make designing systolic architectures very powerful. Firstly, they achieve nearest neighbor communication. Therefore they do not contain long interconnect wires. Secondly they are easy to pipeline therefore they do not contain long critical paths.

Our modified Extended Binary GCD algorithm can be descibed by a data flow graph as in Fig. 1.

The square boxes correspond to control logic whereas the circular boxes correspond to the data processing elements.

The additional mode control signal specifies which method to use $GF(2^m)$ or $GF(p)$.

$$mode = \begin{cases} 1, & GF(p) \\ 0, & GF(2^m) \end{cases}$$

One could essentially run $GF(2^m)$ divisions immediately after $GF(p)$ divisions and vice versa simply by changing the value of the mode signal.

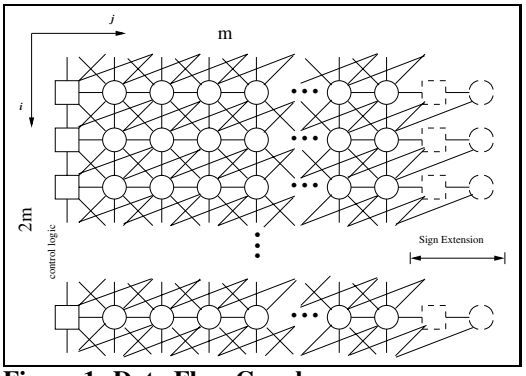

Figure 1: Data Flow Graph

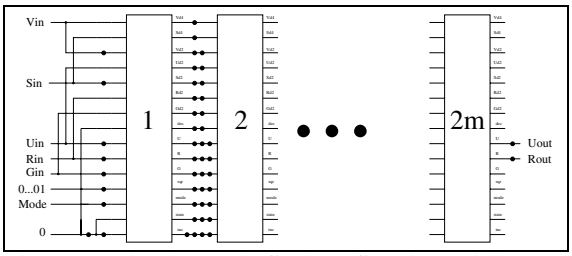

Figure 2: Pipelined Bit Serial LSB First Divider

It is possible to generate an LSB first bit-serial dual field divider by using the following projection vector ($d^T = (0, 1)$), scheduling vector ($s^T = (3, 1)$), and processor space vector ($p^T = (1, 0)$) as in Fig. 2. These values are extremely efficient because they lead to a hardware utilization efficiency of 100 percent (i.e. $|d^T s| = 1$).

Table 2: Signals and Delays

	$\mathbf{e}^T$	$\mathbf{p}^T\mathbf{e}$	$\mathbf{s}^T\mathbf{e}$
v, s	(1,-2)	1	1
dec,v, u, r, s, g	(1,-1)	1	2
ctrl,u, r, g	(1,0)	1	3
ctrl,carries	(0, 1)	0	1
inc	(1, 1)	1	4

4.3. Performance

The performance of the dual field division unit is determined by the critical path in the processing elements and the latency through the division unit. A careful analysis shows that the critical path (CP) is $T_{CP} = 5T_{MUX} + 3T_{XOR}$. The latency through the division unit is determined by the number of delay elements along the control signal path. The bit-serial divider requires m bits input, then each pipeline adds a three cycle delay for a total of $6m - 3$ cycles through the unit, plus 1 for the pipelined input. Therefore the total latency is $7m - 2$ cycles.

The area is determined by the number of logic gates in each processing element times the number of processing elements. There are $2m$ processing elements and each one consists of 15 $XOR2$, 40 $AND2$, 29 $MUX2$, and 7 INV gates.

For this project a VHDL description was designed and synthesized with Xilinx's ISE 6. Analysis in Xilinx Timing Analyzer revealed a critical path latency of 8.470 ns which effectively indicates an operational frequency of 118 MHz. Therefore a target frequency of 100 MHz can easily be achieved on a field programmable gate array (FPGA).

4.4. Elliptic Curve Cryptography

The complexity of scalar point multiplication for elliptic curve cryptography in affine coordinates is

$$\begin{aligned} T_{Cycles} &= 1T_{ADD} + 3 + O(m) \times \\ &\quad (1T_{DIV} + 3T_{MLT} + 2T_{SQR} + 4T_{ADD} + 2) + \\ &\quad O\left(\frac{m}{3}\right) \times (1T_{DIV} + 1T_{MLT} + 1T_{SQR} + 8T_{ADD} \\ &\quad +1) + 1 \approx \frac{45}{3}m^2 + 9m + 5 \end{aligned}$$

Substituting into the previous equation with $T_{MLT} = T_{SQR} = m$, $T_{DIV} = 7m$, and $T_{ADD} = 1$ then the total cycles for one scalar point multiplication becomes $T_{Cycles} = \frac{45}{3}m^2 + 9m + 5$.

The same scalar point multiplication operation can be accomplished in what is known as projective coordinates with a cycle complexity of

$$\begin{aligned} T_{Cycles} &= 1T_{ADD} + 3 + O(m) \times \\ &\quad (7T_{MLT} + 5T_{SQR} + 4T_{ADD} + 2) \\ &\quad +O\left(\frac{m}{3}\right) \times (13T_{MLT} + 1T_{SQR} + 7T_{ADD} + 1) \\ &\quad +T_{CTA} + 1 \approx \frac{56}{3}m^2 + \frac{26}{3}m + 5 \end{aligned}$$

The total cycles can be computed with the values $T_{MLT} = T_{SQR} = m$, $T_{ADD} = 1$, $T_{CTA} = 2m^2$ then the total cycles for one scalar point multiplication becomes $T_{Cycles} = \frac{56}{3}m^2 + \frac{26}{3}m + 5$.

Therefore the division unit provides a performance speedup of 1.24 times the performance without a division unit.

Given a 100 MHz clock speed using affine coordinate scalar point multiplication the total scalar point multiplication per second (kP/s) for $GF(2^{571})$ is

$$\frac{(100 \times 10^6 \text{ cycles per second})}{(\frac{45}{3}(572^2) + 9(572) + 5 \text{ cycles per } kP)} \approx 20.4 \text{ kP/s}$$

Similarly, for a 100 MHz clock speed using affine coordinate scalar point multiplication the total scalar point multiplication per second (kP/s) for $GF(p)$ where $|p| = 521$ is

$$\frac{(100 \times 10^6 \text{ cycles per second})}{(\frac{45}{3}(522^2) + 9(522) + 5 \text{ cycles per } kP)} \approx 24.4 \text{ kP/s}$$

4.5. Comparison

In [4,5] m is defined to be the highest order in the reduction polynomial but our definition of m is it includes all the bits

and the highest order bit.

Table 3: Field Divider Comparison

	[4]	[5]	Fig. 2
$GF(2^m)$	Yes	Yes	Yes
$GF(p)$	–	–	Yes
Throughput	$\frac{1}{m-1}$	$\frac{1}{m-1}$	$\frac{1}{m}$
Latency			
$GF(2^m)$	5m-9	5m-7	7m-2
$GF(p)$	–	–	7m-2
Critical Path	$T_{AND2}+$ $3T_{XOR2}+$ T_{MUX2}	$T_{AND2}+$ $2T_{XOR2}$	$5T_{MUX2}+$ $3T_{XOR2}$
Gates / PE			
$MUX2$	11	7	29
$AND2$	7	12	40
$OR2$	0	1	21
$XOR2$	5	4	15
INV	0	7	7

There are a few things one can observe in Table 3. First, there are significantly more multiplexors in the dual field divider design. These additional multiplexors are due to the sign extension and the dual field option. Similarly an increased number of gates are required to minimize the critical path. This architecture was designed without the option to change the number of pipelining cutsets based on which processing mode the system is in. It would not be difficult to add this feature and a substantial boost would be added to the $GF(2^m)$ mode because it does not require the division by 4 path. Also, the assumption of the lowest order bit being one was not made which accounts for an additional cycle and limits the throughput to $\frac{1}{m}$ instead of $\frac{1}{m-1}$ as in [4, 5].

5. CONCLUSION

Our contributions are the LSB first bit-serial dual field divider for $GF(2^m)$ and $GF(p)$ and its performance analysis for elliptic curve scalar point multiplication.

Our research has confirmed that dual field divider units are feasible for hardware implementations. With dual field divider units it is possible to increase scalar point multiplication in elliptic curve cryptography 1.24 times.

Future research includes minimizing the critical path, overall latency, and power because these are important architectural optimizations. Secondly, it may be possible to extend this technique to an implementation of the Extended Euclidean GCD Algorithm. Thirdly, it may be possible to obtain an increased speedup with an early exit path. Finally, designing a dual field divider with a form of redundant arithmetic would be useful in minimizing the overall latency however this cannot be done without increasing wire delays.

6. REFERENCES

[1] D. E. Knuth, *The Art of Computer Programming*, Addison-Wesley, 1997.

[2] K. K. Parhi, *VLSI Digital Signal Processing*, Wiley & Sons, 1999.

[3] R. P. Brent and H. T. Kung, "Systolic VLSI arrays for polynomial GCD computation," *IEEE Transactions on Computers*, vol. C-33, pp. 731–736, 1984.

[4] J. Guo and C. Wang, "Bit-Serial Systolic Array Implementation of Euclid's Algorithm for Inversion and Division in $GF(2^m)$," *IEEE International Symposium on VLSI Technology, Systems, and Applications*, pp. 113 – 117, June 1997.

[5] C. Kim, S. Kwon, C. P. Hong, and G. I. Nam, "Efficient Bit-Serial Systolic Array for Division over $GF(2^m)$," *International Symposium on Circuits and Systems, ISCAS*, vol. 2, pp. 252–255, 2003.

[6] P. Montgomery, "Modular Multiplication Without Trial Division," *Mathematics of Computation*, vol. 44 no. 170, pp. 519–521, April 1985.

[7] B. S. Kaliski Jr., "The Montgomery Inverse and Its Applications," *IEEE Transactions on Computers*, vol. 44 no. 8, pp. 1064–1065, 1995.

[8] E. Savas and C. K. Koc, "The Montgomery Modular Inverse - Revisited," *IEEE Transactions on Computers*, vol. 49 no. 7, pp. 763–766, 2000.

[9] A. A.-A. Gutub, A. F. Tenca, and C. K. Koc, "Scalable VLSI Architecture for GF(p) Montgomery Modular Inverse Computation," *IEEE Computer Society Annual Symposium on VLSI*, pp. 53–58, 2002.