A Dual-field Modular Division Algorithm and Architecture for Application Specific Hardware

Lo'ai A. Tawalbeh, Alexandre F. Tenca, Song Park and Ceţin. K. Koç
School of Electrical Engineering & Computer Science
Oregon State University
tawalbeh,tenca,parkso,koc@ece.orst.edu

Abstract— **This paper presents a dual-field modular division (inversion) algorithm and its hardware design. The algorithm is based on the Extended Euclidean and the Binary GCD algorithms. The use of counters to keep track of the difference between field elements in this algorithm eliminates the need for comparisons which are usually expensive and time-consuming. The algorithm has simple control flow and arithmetic operations making it suitable for application specific hardware implementation. The proposed architecture uses a scheduling method to reduce the number of hardware resources without significantly increasing the total execution time. Its datapath efficiently supports all the operations in the algorithm and uses carry-save unified adders for reduced critical path delay, making the proposed architecture faster than other previously proposed designs. Experimental results using synthesis for AMI $0.5\mu m$ CMOS technology are shown and compared with other dividers.**

Keywords—**Unified Algorithm, Finite Fields Arithmetic, Hardware Architecture, Modular Division, Modular Inverse, Dual-field Division, Carry-save adders**

I. Introduction

Computing the modular division or inverse in finite fields (most frequently used fields are prime fields – $GF(p)$ – and binary extension fields – $GF(2^n)$), is a very important arithmetic operation in cryptographic algorithms based on elliptic curves [1].

A previous study shows that the performance of Elliptic Curve Cryptography (ECC) can be improved when modular division or inverse are implemented in hardware [2], [3].

Based on the Extended Euclidean Algorithm (EEA) [4] and the Binary GCD algorithm [5], a Unified Modular Division (UMD) algorithm was developed and compared with other algorithms in [6]. The UMD is a dual-field algorithm able to compute the modular division in both $GF(p)$ and $GF(2^n)$ fields. It does not have complex operations and tests and it requires fewer clock cycles than other modular division algorithms. These characteristics make the algorithm suitable for hardware implementation. This work briefly presents the UMD algorithm developed in [6] and proposes a full-precision hardware architecture for it.

Most of the modular division (inversion) algorithms [3], [7], [8], [9] have integer and polynomial degree comparisons as part of their control flow. Differently from them, the UMD algorithm uses a counter variable to keep track of the difference between field elements and a single test of zero which basically eliminates the need for complex tests and reduces the complexity of each iteration [6]. Counters have also been used to some extend in some division algorithms in $GF(2^n)$ [9] and $GF(p)$ [10]. The counter can be implemented using fast up/down counters as the ones described in [11].

The public-key processor presented in [7] implements operations required for Elliptic Curve Cryptography (ECC) including modular inverse in $GF(2^n)$ only, while the algorithm in [5] considers only elements in $GF(p)$. The bit-serial systolic architecture presented in [9] computes the modular inverse in $GF(2^n)$ only. Another work in [8] presents a very simple dual-field arithmetic unit, however an unified algorithm for modular inverse/division was not shown, making the control section of the design somewhat unknown. All the modular division algorithms in these works are based on EEA.

This paper describes a hardware architecture for the UMD [6]. A scheduling technique is used to better utilize the adders in the datapath and reduce the hardware complexity of the final design. The computations in both fields ($GF(p)$ and $GF(2^n)$) are implemented with redundant Carry-Save (CS) adders to reduce the clock cycle time and make the design almost independent on the operand precision.

The following Section presents some mathematical concepts and the notation used in this paper. Section III introduces the unified modular division algorithm and discusses its properties. The overall organization of the hardware design that implements the UMD and its main functional blocks are shown and described in detail in Section IV. Experimental results are shown in Section V, followed by conclusions in Section VI.

II. Mathematical Concepts and Notation

The elements of the binary extension field $Y(x) \in GF(2^n)$ are non-zero polynomials of degree less than n when the polynomial basis is used to represent the field elements (which is the case in this paper). Each element has coefficients that are elements in $GF(2)$, which are represented by the values $\{0, 1\}$. On the other hand, the elements in the prime field $GF(p)$ are integers in the range $\{0, ..., p-1\}$ where p is a n-bit prime modulus in the range $2^{n-1} < p < 2^n$. Bit vectors are used to represent the elements in both fields as follows:

$$GF(2^n) : Y(x) = \sum_{i=0}^{n-1} y_i * x^i$$

$$GF(p) : Y = \sum_{i=0}^{n-1} y_i * 2^i$$

where $y_i \in \{0, 1\}$ in both cases. The polynomial $Y(x)$ is denoted as Y in the algorithm description for simplicity.

The addition operation of the elements is different in each field. Addition of two polynomials in $GF(2^n)$ is done by a bitwise logic exclusive OR operation (a xor $b = a \oplus b = a'b + ab'$) between the two bit vectors being added. In other words the additions in $GF(2^n)$ are done modulo 2 [8], as shown in the following equation:

$$Y(x) + W(x) = \sum_{i=0}^{n-1} y_i * x^i + \sum_{i=0}^{n-1} w_i * x^i = \sum_{i=0}^{n-1} (y_i \ xor \ w_i) x^i$$

Subtraction and addition in $GF(2^n)$ are equivalent. Intermediate results of operations in $GF(2^n)$ that are represented

Function: Modular Division in $GF(p)$ and $GF(2^n)$ fields
Inputs: $0 \leq X < p$, $0 < Y < p$, $2^{n-1} < p < 2^n$, $Field$
Output: $Z = \frac{X}{Y} \mod p$ when $Field = GF(p)$, $Z(x) = \frac{X(x)}{Y(x)} \mod p(x)$ when $Field = GF(2^n)$
Algorithm:
$C = Y$, $U = X$, $D = p$, $W = 0$, $\delta = 0$

```
WHILE C ≠ 0
  IF c0 = 0 THEN
    C := C >> 1
    δ := δ − 1            /* Integer Operation */
  ELSE
    IF δ < 0 THEN C ⇔ D, U ⇔ W, δ := −δ
    END IF;
    k := 1
    IF((C + D) mod4 ≠ 0 AND Field = GF(p)) THEN k := −1
    ELSE   δ := δ − 1
    END IF;
    C := (C + k * D) >> 1, U := (U + k * W);
  END IF;
  U := (U + u0 * p) >> 1
END WHILE;
IF D = 1 THEN Z := W
ELSE Z := p − W
END IF;
```

Fig. 1. Unified Modular Division Algorithm (UMD)

by polynomials of degree greater or equal to n are reduced using a field polynomial $p(x)$ of degree n (irreducible polynomial).

Moreover, the addition of two elements Y and W in $GF(p)$ is done as a conventional integer addition. The propagation of carries in this case will depend on the use of redundant or non-redundant representation of elements. Carry-Save (CS) representation is used in this work. Modular reduction is required when the sum exceeds the value of p to keep the result in the set $\{0, ..., p-1\}$.

III. The Unified Modular Division Algorithm (UMD)

Figure 1 shows the dual-field division algorithm proposed in [6]. The UMD algorithm computes the modular division in $GF(2^n)$ when $Field = GF(2^n)$ ($Z(x) = \frac{X(x)}{Y(x)} \ mod\ p(x)$), and in $GF(p)$ when $Field = GF(p)$. In both cases, $Y \neq 0$. If X is set to one, the UMD algorithm computes the modular inverse. We must say that the operations on the control variable δ are always integer operations regardless of what is the specified field. On the other hand, specifying a field forces all the additions/subtractions to be done in this field. Swap of values between two variables is indicated by the symbol $\Leftrightarrow$. The notation for the least-significant bits of C and U is c_0 and u_0, respectively. Notice that inputs to the UMD algorithm $(X\ , Y)$ are bit vectors that represent elements in $GF(p)$ and $GF(2^n)$.

The UMD algorithm computes the division in $GF(p)$ based on some facts related to the greatest common divisor (GCD) of two numbers [4], [5], [6]. Basically, in $GF(p)$ it can be shown that when C is odd then either $C + D$ or $C - D$ is divisible by 4, and so it can be reduced (by right shift). In $GF(2^n)$, the divisibility by 4 cannot be enforced, but the result of $C + D$ is still divisible by 2.

The modular reduction operation $U := (U + k * W)/2 \bmod p$ in $Gf(p)$ or $U(x) := (U(x) + k * W(x))/x \bmod p(x)$ in $GF(2^n)$, is implemented by the combination of the two expressions: $U := (U + k * W)$ and $U := (U + u_0 * p) >> 1$. This way, the modular reduction is done by a simple conditional addition of the modulus.

The validity of the UMD algorithm is shown in [6]. The inverse in the Montgomery domain can be also computed by the same algorithm. Considering the values r and Y as inputs to the UMD algorithm, where $r = 2^n$ or x^n (depends on the field), we get $Z = \frac{r}{Y} \pmod p = Y^{-1}r \pmod p$ which is the inverse of Y in the Montgomery domain. But if we use r and Yr as inputs, the algorithm computes $Z = \frac{r}{Yr} \pmod p = \frac{1}{Y} \pmod p$ which is the inverse of Y in the integer domain. The execution time is the same for all cases when we consider that the constant r is pre-computed.

IV. Overall Organization

Figure 2 shows the top level organization of the unified modular divider that implements the UMD algorithm. The main functional blocks are *Registers, Swapping Network (Multiplexers), Control and Datapath.*

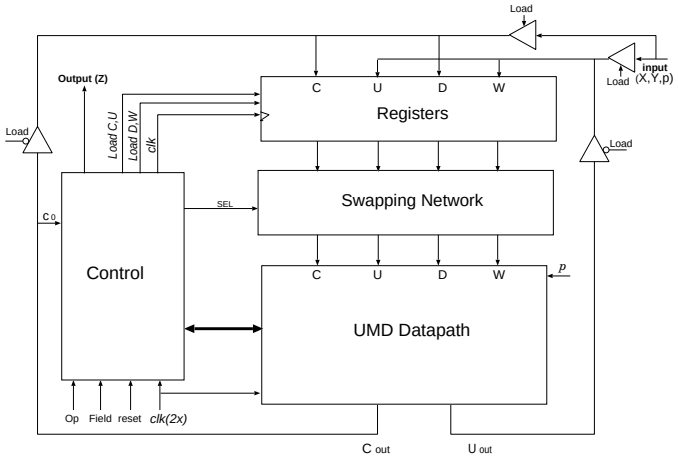

Fig. 2. Top Level Organization of The Modular Divider Which Implements The UMD Algorithm.

The registers C, U, D, and W are initialized with the inputs (X,Y,p) when $Load = 1$ through three-state buffers. When $Load = 0$, the registers receive their values from U_{out} and C_{out} coming from the datapath. U_{out} is fed back to either U or W registers depending on $Load\ U$ and $Load\ W$, respectively. Also, C_{out} is fed back to C or D registers depending on $Load\ C$ or $Load\ D$, respectively. All these signals are generated by the control block.

The swap operations ($C \Leftrightarrow D$, $U \Leftrightarrow W$) are realized by the Swapping Network which is a set of two-input muxes, controlled by the SEL signal provided by the control block and takes its value based on the value of δ (kept internally). It is described in more detail in Section IV-B.

A. Adders Scheduling for Efficiency

The UMD algorithm performs in the worst case (*else* part of the algorithm) 3 additions in each iteration which are shown in Figure 3. Using 3 adders will increase the area of the design significantly especially for large precision inputs. Another alternative is to use a single adder in more than one clock cycle to complete one iteration. Such a solution would increase the overall time to compute the division but would be a solution when the area is too restrictive. Therefore for this implementation of an isolated division unit the use of two adders is the best choice.

In this worst case scenario (figure 3), there is data dependency between additions $A1$ ($U + k * W$) and $A2$ ($U + u_0 * p$). Therefore, one iteration is complete only after two consecutive additions are performed. If we assign addition $A1$ to one adder (adder1) and addition $A2$ to another adder (adder2), each adder will be working for only half of the clock cycle time. Based on this observation, we propose a solution that uses adder1 to compute addition 3 ($A3$) in the second half of the iteration cycle, while adder2 is computing $A2$. This solution requires a register or latch between the two adders, clocked at twice the clock frequency at which iterations are executed.

As can be seen from Figure 3, Adder1 receives the operands to compute $A1$ during phase 1 (ϕ_1). At the end of ϕ_1, the adder output is latched and another set of input values is applied to compute $A3$ during ϕ_2. Note that during ϕ_1 the Adder2 is not used anyway because the signals are still propagating in the first half of the circuit. Another observation is that phase ϕ_2 can be shorter than ϕ_1 in order to keep the hardware units working most of the time.

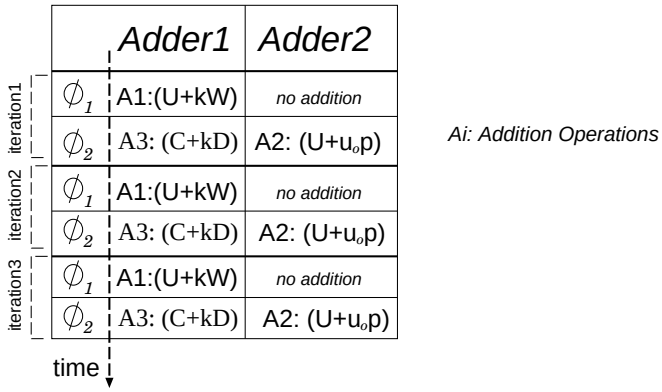

Fig. 3. Scheduling of Adders in UMD Algorithm Implementation

B. Datapath

An n-bit datapath was designed to support the computations described by the UMD algorithm and it is shown in Figure 4. Each iteration of the algorithm is implemented in one clock cycle. The critical path delay (the clock cycle time) is determined by the datapath and control block, and it will be addressed in more detail in Section IV-C.

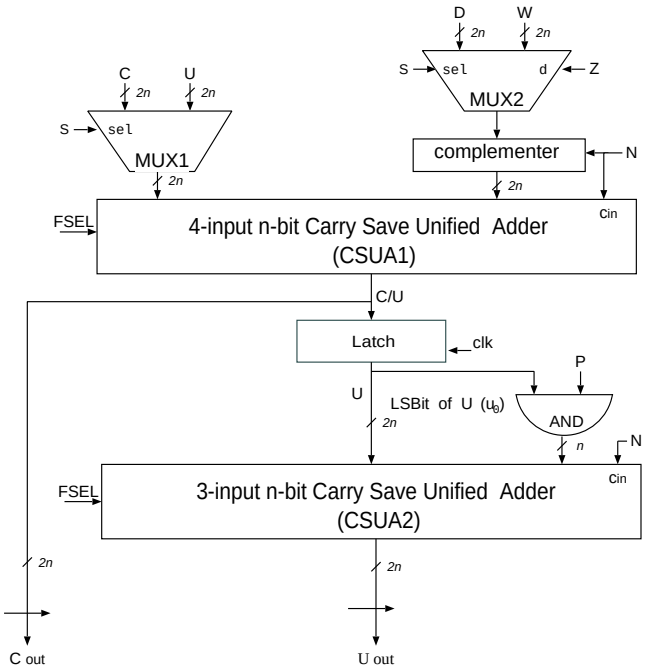

Fig. 4. The Unified Datapath of The Modular Divider

The proposed datapath uses two Carry-Save Unified Adders (CSUAs) to perform addition in both $GF(p)$ and $GF(2^n)$ fields. The CSUA is basically formed by dual-field adders which were described in [12] for a (3,2) design (3 inputs and 2 outputs) and in [13] for a (4,2) design. The (3,2) dual-field adder is similar in complexity to a full-adder and it performs bit addition with and without carry. This functionality is enabled by the input $FSEL$ (Field Select). When $FSEL = 0$, the carry out bits are forced to 0 and the dual-field adder performs bitwise modulo-2 addition of its inputs. When $FSEL = 1$, the dual-field adder performs the bitwise addition with carry (addition in $GF(p)$). Another implementations of unified adders can be used as the one proposed in [14].

CSUA1 was implemented using n (4,2) dual-field adders and CSUA2 was implemented using n (3,2) dual-field adders. The use of redundant form of the operands enables the circuit to have a critical path that is less sensitive to the operand precision. The addition time is less than the time for non-redundant adder, especially for large precision. A binary vector X is represented in CS form by two vectors XC and XS such that $X = XC + XS$. Therefore, the cost of CS representation comes from more registers and buses.

The three control lines: S, Z, and N in $MUX2$ corresponds to select, zero, and negate, respectively. When $Z = 1$ the output of the mux is forced to zero regardless of S. $N = 1$ produces a bit-complement of the input. Since we are dealing with numbers in two's complement represented in CS form, the change of sign is done by complementing each vector and adding 1. Thus, N is inserted as carry input into both CSUAs to get the change of sign operation in this system.

The latch between the two carry-save unified adders lets the information at its input pass through during ϕ_1 and holds the information at its output when it is ϕ_2.

The UMD algorithm computes the modular division in $GF(p)$ when $Field = GF(p)$. The select signal S is synchronized with the latch. $MUX2$ is used to implement $k * D$ and $k * W$, where $k \in$ {-1,1}. In the case $k = -1$, the negative D and the negative W are obtained by setting $N = 1$. Both signals Z and N are synchronized with the main clock (clk).

If C is even, then it is shifted right one bit and the counter δ is decremented by one. If not, we test δ, if it is negative, the circuits swap the values of C and D, and U and W, and change the sign of δ. The swap operation is performed by the Swapping Network that precedes the datapath and takes its inputs from the C, U, D, and W registers.

The test $(C + D)\ mod\ 4 \neq 0$ can be implemented using a small two-level gate network.

The addition $U := (U+k*W)$ which corresponds to $A1$ in Figure 3, is performed in the first phase of the clock signal (ϕ_1) using the CSUA1. During ϕ_2, two separate additions happen: $C := (C+k*D)$ ($A3$ in Figure 3) using CSUA1, and $U := (U + u_0 * p)$ ($A2$) using CSUA2. Both outputs are shifted to the right by one bit to complete the algorithm operations.

An AND gate is used to select between the value 0 or the modulus p depending weather U is even or odd, respectively.

If the algorithm is computing the modular division in $GF(2^n)$, the same procedure described above is followed, except that the test $(C + D)\ mod\ 4 \neq 0$ is not applicable ($Field = GF(2^n)$). For both fields, the computation is done when $C = 0$, and the result is $Z = W$.

It can be shown that the UMD algorithm does not change the values of the operands once $C = 0$. Therefore, the test $C = 0$ can take several clock cycles. Another observation shows that the non-redundant representation of $C = 0$ takes only some particular values, which makes this test easier. So, using these two features we can make the test of zero for the CS representation simple and multi-cycle, allowing the design to be fast without a significant increase in area. Another possibility is to use counters to estimate when C reaches 0.

The Swapping Network shown in the datapath is composed of two-input muxes. The control signal (SEL) selects between the inputs. The two possible configurations of the Swapping Network are shown in Figure 5 (when SEL=0 or 1).

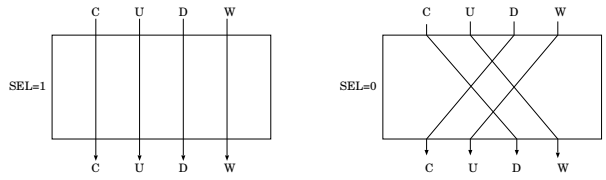

Fig. 5. The Two Possible Configurations of The Swapping Network.

C. Improving The System Performance

Figure 6 shows the critical path of the unified divider which will determine the clock period of the design.

$$clock\ period = 2 * max(delay\phi_1, delay\phi_2)$$

From the figure it is clear that ϕ_1 is longer than ϕ_2. There are two possibilities for the delays in ϕ_2 as shown in Figure 6, coming from the paths that include CSUA1 or CSUA2. Noticing that the delay of CSUA2 is smaller than the delay of CSUA1, the upper path is longer, so it is considered as the delay of ϕ_2.

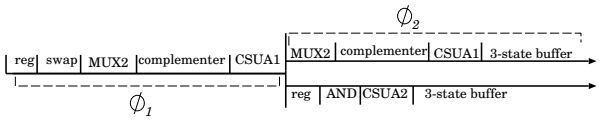

Fig. 6. The Delay Paths of The Modular Divider.

Since that $delay\phi_1 > delay\phi_2$, the delay of ϕ_1 determines the clock period of clk. In this case clk will have a 50% duty cycle. More performance could be extracted from the circuit if ϕ_2 could be made shorter. However, such a solution would involve critical implementation details for the design of the clock signal generator and clock distribution network.

V. Experimental Results and Comparisons

This section includes two categories of experimental results: (a) average number of iterations obtained from a *Maple* model and (b) the critical path delay results obtained by synthesis of the VHDL description of the algorithm.

A. The Average Number of Iterations

Maple was used to describe the proposed algorithm ($Alg1 = UMD$ algorithm) and the unified Montgomery inverse algorithm presented in [3] ($Alg2$). At least 100 *random samples* were used to verify each algorithm operation and obtain statistics.

For consistency, no multiple-word calculation is considered here. For an $n-$bit input Y, $Alg2$ computes $Z = Y^{-1}2^k$, where $n \leq k \leq 2n$ is the number of algorithm iterations. A correction step is needed to get the inverse in the Montgomery domain ($Y^{-1}2^n$) or in the integer domain (Y^{-1}). Therefore, the total number of iterations required to compute the Montgomery inverse is $2k - n$. To compute the integer inverse it needs $2k$ iterations.

Also, $Alg2$ uses *number comparisons* to compare the size of the bit vectors that represent elements in the field (the same way it was done in [2], [7], [8]) instead of the counter (δ). These *comparisons* are expensive in both fields. On the other hand, the UMD algorithm has only additions which has a complexity of $O(1)$ since we are using redundant representation. The number of additions gives an idea of the overall work done by the algorithm. It was shown in [6] that $Alg1$ has up to 9% (5%) fewer additions than $Alg2$ when computing integer inverse in $GF(p)$ ($GF(2n)$). The comparisons were considered as an extra addition. There are k comparisons in $Alg2$. It is important to mention that this assumption is very conservative since the complexity of a comparison is much higher than the complexity of a redundant addition. From the above discussion we can say that the *comparison* limits a fast hardware implementation.

Figure 7 shows the number of iterations as a function of operand size required by $Alg1$ (UMD) and $Alg2$ (presented in [3]) to compute the integers modular inverse (not in Montgomery domain) in $GF(p)$ and $GF(2^n)$. The size of the operands is in the range (160 to 512) bits.

One can see from the figure that $Alg1$ executes on average 25% fewer iterations than $Alg2$ when computing the inverse in the integer domain for $GF(p)$. For $GF(2^n)$, $Alg1$ has about 40% fewer iterations than $Alg2$ when computing the inverse in the integer domain. As discussed before, $Alg2$ has a more complex iteration than the UMD algorithm and therefore, the UMD algorithm presented in this paper is much faster than the unified Montgomery algorithm presented in [3].

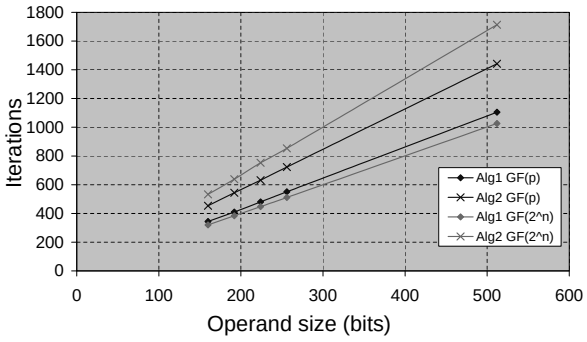

Fig. 7. The Number of Iterations as a Function of Operand Size Required by $Alg1$ (UMD) and $Alg2$ to Compute The Modular Inverse in $GF(p)$ and $GF(2^n)$

Notice that the number of iterations for the UMD algorithm in both fields increase linearly with the increase in the operand precision.

The public-key processor presented in [7] implements operations required for Elliptic Curve Cryptography (ECC) including modular inverse in $GF(2^n)$ only. It is mentioned in [7] that the $GF(2^n)$ inversion operation takes about 3.3 cycles for each bit. Figure 7 shows that the UMD algorithm needs only 2 cycles/bit to compute the modular inverse in $GF(2^n)$.

The dual-field arithmetic unit proposed in [8] performs one addition in each clock cycle and the adder is used to convert from carry-save form to non-redundant representation, significantly increasing the number of clock cycles. So, the only way to compare our algorithm with [8] is to compare the number of additions. It is shown in [6] that the number of additions reported in [8] is around 20 times greater than the number of additions in $Alg1$.

The comparison of the UMD algorithm with [2] was not considered because the work in [2] is a word-based algorithm that applies several strategies (variable shift, for example) to reduce the number of cycles.

B. Synthesis Results

The experimental data presented in this section were generated using Mentor Graphics CAD tools. The target technology was set to $AMI05\_fast\ auto$ (0.5 μm CMOS with hierarchy preserved) provided in the ASIC Design Kit (ADK) from the same company [15].

The unified modular divider design presented in this paper was described in VHDL and simulated in ModelSim for functional correctness. It was synthesized using Leonardo synthesis tool for the mentioned technology. ADK provides a consistent environment for comparison between the designs, and a reasonable approximation of the system performance when using commercial ASIC technology.

Figure 8 shows the critical path delays (in *nano*-seconds) of the UMD design for the precision range from 128-bit to 512-bit. The delay at 128-bit is 10.01 nsec, at 256-bit is 10.85 nsec, and at 512 is 10.86 nsec. From the figure we can notice that the delay increases as the number of bits increase and it seems to became constant at higher precision. This indicates that the critical path delay (clock period) of the UMD algorithm become independent of the operands size at high precisions.

The public-key processor presented by Goodman and et.al in [7] runs at clock rate of 50 MHz (clock period = 20 nsec), and it is considered a good representative of this class of hardware designs. The divider proposed in this work has a worst case clock period of less than 11 nsec at 512-bit operand size, which is twice faster than the processor presented in [7]. Also, as we mentioned in the previous subsection, Goodman's processor, from now on called $G\mu P$ takes 3.3 cycles/bit to perform a modular inverse in $GF(2^n)$. Let the total computation time of a given design be T_{design} which is given by:

$$T_{design} = cycles/bit * n * clock\ period.$$

where n is the operand size in bits. At n = 512-bits, the total

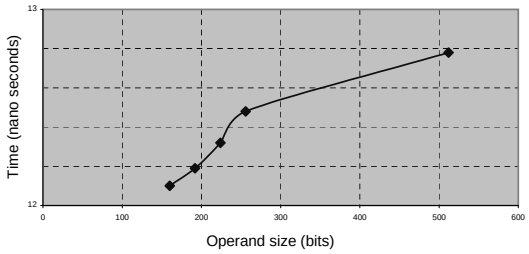

Fig. 8. The Critical Path Delay of The UMD Design in *nano*-seconds.

Operand size (bits)	Area (gates)
128-bit	22853
160-bit	28560
192-bit	34227
224-bit	39913
256-bit	45600
512-bit	91090

TABLE I
THE AREA OF THE UMD DESIGN IN GATES FOR DIFFERENT OPERAND SIZES

computation time of $G\mu P$ ($T_{G\mu P}$) will be:

$$T_{G\mu P} = 3.3 * 512 * 20 \times 10^{-9} = 33.79\mu sec$$

and the total computational time for our design (T_{UMD}) will be:

$$T_{UMD} = 2 * 512 * 10.86 \times 10^{-9} = 11.12\mu sec$$

Comparing $T_{G\mu P}$ and T_{UMD}, we find that the UMD design is 3 times faster. Also, even when we run the proposed divider circuit at 50 MHz we get $T_{UMD} = 2 * 512 * 20 \times 10^{-9} = 20.48$ μsec which is still 1.65 times faster than the design in [7].

Table I shows the total number of gates for the UMD design as a function of operand size. The area for the UMD design can be extracted from the experimental data presented in Table I as $A_{UMD} = 177.7 * n + 108$. From this equation we can say that the proposed divider design has area complexity of $O(n)$.

The architecture presented in [9] is dedicated to $GF(2^n)$ only. The authors mentioned that the design has an area complexity of $O(n)$ without specifying any constants.

The proposed design in [16]is also dedicated to $GF(2^n)$ only and has an area complexity of $O(nlog(n))$.

VI. Conclusion

The proposed Unified Modular Divider architecture that implements the UMD algorithm proposed in [6] can compute the inverse in both $GF(p)$ and $GF(2^n)$ fields in an efficient way. To the best of our knowledge, this is the first unified division/inversion architecture to use a counter to keep track of the difference between the values of elements in the field. The number of clock cycles required by the UMD divider is $2n$ in the worst case while another published design executes division in $3.3n$ cycles. The datapath uses pipelining based on scheduling analysis which makes the best use of the hardware resources in the divider, therefore, the datapath efficiently implements the iterations of the UMD algorithm with two adders only. The area complexity of the proposed design is $O(n)$.

The use of redundant unified adders significantly reduces the critical path delay (and as a result the total computation time) making the proposed architecture faster than many other previously proposed designs. The low time and area complexity of the proposed architecture and its efficient datapath makes it suitable for cryptographic hardware applications.

Acknowledgements This work was supported by the NSF CAREER grant CCR-0093434.

References

[1] G. B. Agnew, R. C. Mullin, and S. A. Vanstone, "An implementation of elliptic curve cryptosystems over $GF(2^{155})$," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 5, pp. 804–813, 1993.

[2] A. A. Gutub, A. F. Tenca, E. Savas and C. K. Koc , "Scalable and unified hardware to compute Montgomery inverse in $GF(P)$ and $GF(2^n)$," in *Cryptographic Hardware and Embedded Systems — CHES 2002*, B.S. Kaliski Jr. et al., Ed. 2003, Lecture Notes in Computer Science, No. 2523, pp. 484–499, Springer, Verlag Berlin Heidelberg 2003.

[3] E. Savas and C. K. Koc, "Architectures for unified field inversion with applications in elliptic curve cryptography," in *The 9th IEEE international conference on Electronic, Circuits and systems-ICECS 2002*.

[4] D. E. Knuth, *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*, Third edition, 1998.

[5] N. Takagi, "A vlsi algorithm for modular division based on the binary gcd algorithm," *IEICE Trans. fundamentals*, vol. E81-A, no. 5, pp. 724–728, May 1998.

[6] Alexandre F. Tenca and Lo'ai A. Tawalbeh, " An Algorithm for Unified Modular Division in $GF(p)$ and $GF(2^n)$ Suitable for Cryptographic Hardware," *IEE Electronics Letters*, vol. 40, no. 5, pp. 304–306, March 2004.

[7] J. Goodman and A. P. Chandrakasan, "An energy-efficient reconfigurable public-key cryptography processor," *IEEE Journal of solid-state circuits*, vol. 36, no. 11, pp. 1808–1820, November 2001.

[8] J. Wolkerstorfer, "Dual-field arithmetic unit for $gf(p)$ and $gf(2^n)$," in *Cryptographic Hardware and Embedded Systems — CHES 2002*, B.S. Kaliski Jr. et al., Ed. 2003, Lecture Notes in Computer Science, No. 2523, pp. 484–499, Springer, Verlag Berlin Heidelberg 2003.

[9] A. D. Daneshbeh and M. A. Hasan, "A unidirectional bit serial architecture for double-bases division over $GF(2^m)$," in *IEEE 16th Symposium on Computer Arithmetic*. 2003, IEEE Computer Society Press, Los Alamitos, CA.

[10] M. E. Kaihara and N. Takagi, "A vlsi algorithm for modular multiplication/division," in *IEEE 16th Symposium on Computer Arithmetic*. 2003, IEEE Computer Society Press, Los Alamitos, CA.

[11] M. Stan, A. Tenca, and M. Ercegovac, "Long and fast up/down counters," *IEEE Transactions on Computers*, vol. 47, no. 7, pp. 722–734, July 1998.

[12] E. Savas, A. F. Tenca, and Ç. K. Koç, "A scalable and unified multiplier architecture for finite fields $GF(p)$ and $GF(2^m)$," in *Cryptographic Hardware and Embedded Systems — CHES 2000*, Ç. K. Koç and C. Paar, Eds. 2000, Lecture Notes in Computer Science, No. 1717, pp. 281–296, Springer, Berlin, Germany.

[13] E. Savas A. F. Tenca and C. K. Koc, "A design framework for scalable and unified multipliers in $GF(p)$ and $GF(2^n)$," *International Journal of Computer Research*, To appear 2004.

[14] L.-S. Au and N. Burgess, "Unified radix-4 multiplier for $GF(p)$ and $GF(2^n)$," in *The IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP'03)*, The Hague, The Netherlands, June 24-26 2003, pp. 226–232.

[15] ASIC Design Kit. Mentor Graphics Co, "http://www.mentor.com /partners/hep/AsicDesignKit/dsheet/ami05databook.html," .

[16] J-H. Guo and C-L. Wang, "Systolic array implementation of Euclid's algorithm for inversion and division in $GF(2^m)$," *IEEE Transactions on Computers*, vol. 47, no. 10, pp. 1161–1167, October 1998.