Modular Multiplication Implementations

Dana Marie Zottola
Department of Electrical & Computer Engineering,
Oregon State University, Corvallis, Oregon 97331 -USA.
E-mail: zottola@engr.orst.edu
May 28, 2002

Abstract— **This paper addresses some key issues in modular multiplication implementations today. We first start by explaining the general need for modular multiplication and exponentiation in cryptosystems, as well as an implementation of a montgomery multiplier and squarer in $GF(2\hat{m})$ [1]. Then we discuss the desire for a unified architecture, being that since the finite fields GF(p) and $GF(2\hat{m})$ are the most commonly accepted in cryptosystems today, it is necessary for a crypto-coprocessor to operate in at least one of them. There are a few unified architectures which can operate in either of the finite fields, which allows interoperability between them. The two architectures are a bit-serial unified architecture [2], and a scalable and unified architecture [3]. Another aspect of montgomery exponentiation is that it can be made 'stronger' from a cryptographic point of view if the final subtractions are eliminated, as explained in [4] and improved upon in [5]. From the research that has already taken place in this field of study, our team proposes a new architecture for multiplication which is unified (bit-serial), scalable and wordbased, doesn't require final subtractions, and only requires approximately half the area of the formerly used architectures.**

I. Introduction

Different implementations have been introduced for multiplication in finite fields GF(p) and $GF(2\hat{m})$, and they range from scalable to unified architectures. The scalable architecures provide the flexibility that so many are looking for, and unified is another flexibility feature that addresses interoperability issues. Other architectures propose no final subtractions (correction step) to improve the speed of multiplication, and another is actually bit-serial. We will discuss each of these architectures in good detail before proposing our new architecture which emobodies the good of all of them, and outperforms them all(area and speed).

II. Montgomery Multiplication in $GF(2\hat{M})$

$GF(2\hat{m})$ is a widely accepted finite field for applications in combinatorial designs, sequences, error-control codes, and cryptography [1]. Montgomery multiplication in $GF(2\hat{m})$ is defined by $a(x)b(x)r\hat{(}-1)$ mod f(x). The field is generated by f(x), an irreducible polynomial, a(x) and b(x) are elements in $GF(2\hat{m})$, and r(x) is a fixed field element in $GF(2\hat{m})$. In a study done by H. Wu [1], a generalized algorithm for this operation, one which extends the range which the degree of r(x) can be chosen from. The original algorithm limits the degree of r(x) to be not less than m-1, while the one presented in this paper the degree of r(x) can be less than m-1. This is beneficial because if we consider the polynomial f(x) = xm + xk + 1, m/2 ¡= k ¡= m-1, it is efficient to choose r(x) = $x\hat{k}$. The results in the study show that this multiplier achieves the same complexity in terms of AND and OR gates as other multiplication implementations (Weakly Dual Basis, and a Polynomial Basis), and it is comparable in speed.

III. Scalable Architectures

As mentioned in the introduction, scalable architectures are desirable due to their ability to adjust to differing constrained areas. The scalable architectures are able to work on any given precision of an operand and it can adjust to any chip area. The paper by A.F. Tenca, G. Todorov, and C.K. Koc [6] discusses the scalable modular multiplier from a high-radix standpoint. It is an extension of previously proposed work on a radix-2, and this paper discusses a radix-8 implementation. The high radix word-based Montgomery algorithm is a generalization of the Montgomery Multiplication algorithm previously presented by the authors. It involves booth encoding, which is interesting. The hardware implementation of this algorithm includes two main functional blocks: Kernel and IO. The Kernel has a datapath (pipeline of Montgomery multiplication cells) where the algorithm computations take place. To supply control signals, there is also a control block to correspond to the datapath. A final reduction step is necessary to complete the algorithm implementation and provide the proper output

(word-serially and on the fly) from the Kernel. Details of the Kernel include booth encoding blocks, multiple generators, adders, and registers. The booth encoding is implemented using lookup tables. A radix-8 implementation is given (as an example) to show an actual processing element cell, and for experimental results. The results are compared to the radix-2 results that the authors already possess, and the radix-8 results that are included are those from a re-timed implementation (something they did to improve the critical path delay since it was unattractive in its original state). Figure 1 summarizes the critical path-delay for the radix-8 Kernel. This critical path delay is a

NS	Bits Per Word 8	16	32	64	128	NS	Bits Per Word 8	16	32	64	128
1	**10.7**	**10.3**	**13.1**	**18.9**	**20.2**	10	11.2	15.2			
2	10.8	**12.1**	14.4	**20.5**	**30.4**	11	11.2	15.3			
3	10.9	12.5	15.7	23.0		12	11.2	**15.4**			
4	11.0	**12.9**	**17.0**	**25.4**		13	**11.3**	15.4			
5	**11.1**	**12.7**	17.6			14	11.3	15.4			
6	11.1	13.5	18.2			15	11.3	**15.5**			
7	**11.2**	**14.3**	18.7			20	**11.4**				
8	11.2	**14.9**	**19.2**			26	**13.0**				
9	11.2	15.1									

Critical path delay for radix-8 Kernel (nsec)

Figure 1 [6]

function of the number of stages for the kernel, as well as the number of bits per word in the operands. [6] The authors conclude that from the data obtained in the experiments, the fastest designs are achieved with a word size of 8 bits. The area is 14964 NOR gates (256-bit precision) with 15 stages. Additionally, they conclude that the radix-8 scalable multiplier is able to perform as well as the radix-2 design for small areas, and better than the radix-2 design for large areas. [6] Another paper by Koc and Tenca [7] presents an architecture that is simply word-based and scalable. The benefits of this architecture are the same as the architecture presented above, and it shares the same flexibilities. The authors propose an algorithm for Radix-2 Montgomery multiplication, and claim that it is adequate for hardware implementation because it is composed of simple operations. Those simple operations are word-by-bit multiplication, bit-shift (division by 2), and addition. The division by 2, or bit-shift is simple in hardware because operands are expressed as bit-strings [7]. The dependencies of the multiplication algorithm need to be taken into consideration for the pipeline implementation of it. These dependencies, as mentioned before, help determine the number of processing elements necessary for the hardware implementation as a scalable architecture and to achieve desired performance. The scalable architecture for this multiplication includes a kernel and IO. The kernel consists of stages of processing elements, arranged as a pipeline. The processing element block diagram is shown in Figure 2. To skip to the results (our

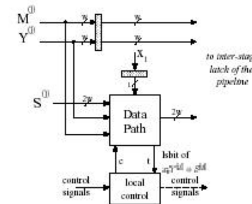


Fig. 10. The block diagram of the processing element (PE).

Figure 2 [7]

architecture is similar to this, so more details will be provided in the New Architecture section), Figure 3 summarizes the execution time for the kernel configurations and operand precisions.

p	Operand's precision - $n = 256$, Word size - w: 8	16	32	64	128
1	56.6	32.2	20.5	13.7	8.8
2	31.3	18.3	12.0	8.0	**7.8**
3	21.6	13.1	8.7	**7.3**	9.5
4	16.3	10.2	7.0	8.1	11.0
5	13.6	8.7	**6.8**	8.7	
6	11.4	7.3	7.1	9.1	
7	10.1	6.3	7.5	9.7	
8	8.9	5.5	7.8		
9	8.3	**5.4**	8.1		
10	7.5	5.5	8.2		
11	7.0	5.7	8.5		
12	6.4	5.7	8.6		
13	5.9	5.7	8.5		
14	5.6	5.9	8.8		
15	5.3	6.1			
16	4.9	5.9			
20	4.9	6.6			
26	**4.7**	8.3			
35	5.0				
50	6.1				

p	Operand's precision - $n = 1024$, Word size - w: 8	16	32	64	128
1	885.0	492.5	300.8	186.3	105.1
2	488.8	279.6	175.7	108.8	70.1
3	335.3	197.9	126.5	82.0	56.4
4	254.3	154.8	101.5	68.4	**49.7**
5	209.0	130.7	86.7	58.3	
6	176.6	109.0	76.3	51.6	
7	155.6	93.8	69.1	**47.0**	
8	137.2	81.7	63.6		
9	125.1	74.3	57.8		
10	144.4	68.5	53.3		
11	104.5	63.8	49.6		
12	95.6	58.9	45.8		
13	87.9	55.2	42.6		
14	82.3	52.2	**40.2**		
15	76.8	49.6			
16	72.1	46.9			
20	60.0	42.4			
26	44.8	**41.1**			
35	33.5				
50	**27.0**				

TABLE III

TOTAL EXECUTION TIME (IN μs) FOR DIFFERENT KERNEL CONFIGURATIONS AND DIFFERENT OPERAND'S PRECISION

Figure 3 [7]

IV. UNIFIED ARCHITECTURES

Since the finite fields GF(p) and GF(2m̂) are the most widely accepted in cryptography, it is essential for a crypto-coprocessor to operate in at least one of them. A few papers propose that a coprocessor should operate in both fields to promote interoperability between the two. One such paper by J. Grossschadl [2], proposes a bit-serial unified architecture which performs the modulo reduction during the multiplication (concurrently) by reducing each intermediate result. The bit-serial notion is used in order for elements in either field, GF(p) and GF(2m̂), to be represented

using a bit-string. This study uses the MSB-first iterative modulo multiplication algorithm shown in figure 4, and carry-save adders. The carry save adders were chosen to eliminate the carry propagation in long integer addition. The claim is that they are widely used in arithmetic circuits due to their performance in terms of speed and area [2]. The paper presents an optimized shift-and-add modulo multiplication from other bit-serial implementations. This modified version includes redundant representation of the intermediate result for use by carry-save adders and continuous modulus reduction instead of doing it all at once. This architecture includes an n-bit Modulus/IP register for storing the bit-string representation of either the modulus or irreducible polynomial. Also included are 4 n-bit registers, a pipelined w-bit carry-lookahead adder, I/O register, and multiplicand/multiplier register. The performance estimation is in Figure 5. For an (n+1)-bit arithmetic unit and a w-bit CLA, the number of clock cycles for a modulo multiplication is shown. This design is scalable in size, and can operate over a wide range of finite fields.

```
INPUT: An n-bit modulus M (i.e., 2^(n-1) ≤ M < 2^n), a
       Multiplicand A < M, and a multiplier B < M.
OUTPUT: Result R = A*B mod M
       1: R ← 0
       2: for i from n-1 downto 0 do
       3:    R ← 2*R + A*B[i]
       4:    q ← |R/M|
       5:    R ← R-q*M
       6: endfor
```

MSB-first shift-and-add multiplication with interleaved modulo reduction

Figure 4: MSB-first shift-and-add multiplication with interleaved modulo reduction. [2]

$$c = 1.5*n + 1.5*([n/w] + \log_2(w)) = 1.5*n$$

Estimated number of clock cycles for a modulo multiplication (w-bit CLA)

Figure 5 [2]

The second proposed architecture was explained in a paper by E. Savas, A.F. Tenca, and C.K. Koc [3]. This architecure is a unified and scalable implementation of multiplication. This architecture 'can handle operands of any size, and the wordsize can be selected based on the area and performance requirements' [3]. The algorithm multiple-word Mongtomery multiplication are shown for both GF(p) and GF(2m̂), and the concurrency of those algorithms are used to implement the pipelined architecture. Parallelism among the isntructions across the different iterations of the i-loop of the algorithms can be used to accomplish the concurrent computation of Montgomery multplication [3]. The pipelined architecture consists of registers and processing units. The processing unit itself consists of two layers of adder blocks. Those adder blocks are called dual-field adders because they are capable of performing addition both with and without carry. The addition with carry corresponds to addition in GF(p), while the addition without carry corresponds to addition in GF(2m̂). Figure 6 shows the execution times of hardware and software implementations of the GF(p) multiplication. Here we can see the speedup after optimized synthesis. 'The fundamental contribution of this research is to show that it is possible to design a dual-field arithmetic unit without compromising scalability, the time performance, and area efficiency" [1].

precision	Hardware (us) (80 MHz, w=32, k=7)	Software (us) (on ARM with Assembly)	speedup
160	4.1	18.3	4.46
192	5.0	251.0	5.02
224	5.9	33.2	5.63
256	6.6	42.3	6.41
1024	61.0	570.0	9.34

Execution Times of HW and SW Implementations of GF(p) Multiplication

Figure 6 [1]

V. Montgomery Exponentiation with No Final Subtractions

As pointed out in a paper by C.D. Walter [4], if the method of Mongtomery exponentiation is properly setup, no final subtractions are necessary for cleanup. This is particularly valuable because it saves computation time to eliminate the subtractions, as well as aiding in avoiding certain comparisons which make timing attacks more successful on the cryptosystem. The basic details are that a final subtraction is only necessary when A=0, but A=0 'clearly leads to all numbers being identically 0 throughout the exponentiation' [4], and in particular the final output is 0 and does not require a final subtraction. In a study by G. Hachez and J. Quisquater [5], this particular point is improved upon by showing that if the pre-multiplication phase is reduced mod N, and then a normal mutliplication algorithm is used for that, then we are guaranteed that there will not be any required final subtractions, and we gain speed in the process. "Because the result is returned by value and not address, if the result must be kept, it must be copied. To avoid timing

attacks in the other case (no copy), an empty loop is executed to simulate the time taken by the copy. This method can easily be detected in a power attack" [6]. This new method helps to strengthen the multiplication because no conditional instructions are executed anymore.

VI. New Architecture

After reviewing the current research, our team developed a new architecture that is scalable, unified and wordbased with no final subtractions, and is smaller and faster than all previous implementations.

A. Overview

All of the benefits of the previously discussed work can be put into one architecture. We have explored such an option, and come up with an architecture that is optimal in many ways. We utilize a bit-serial architecture in order to implement the unified feature, we eliminate final subtractions for speedup, and of course we have a scalability feature for flexibility.

B. Algorithm

The Montgomery multiplication algorithm is generalized in figure 7. How we use it for our implementation is we pre-compute every exponent we'll need and use normal multiplication (modular) to reduce. This insures that we'll have an operand in the proper range and we don't need the final subtractions.

{Pre-condition: N prime to 2^t}
S = 0
for i=0 to p-1
  $q_i = (s_0 + a_i b_0) n_0' \bmod 2^t$
  $S = (s + a_i * B + q_i * N) \text{ div } 2^t$
  {Invariant: $0 \leq S < N+B$}
endfor
{Post-condition: $S2^{pt} = A*B + Q*N$}

Montgomery Multiplication

Figure 7 [5]

C. Implementation and Results

Figure 8, borrowed from [7] is similar to the pipeline we have used to implement our architecture. The only difference from this figure is that we have added the n-bit Modulus or IP (irreducible polynomial) register for the unified feature (to use either GF(p) or GF(2m̂)), and to allow bit-serialization. In terms of speed

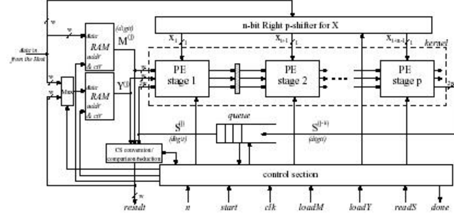
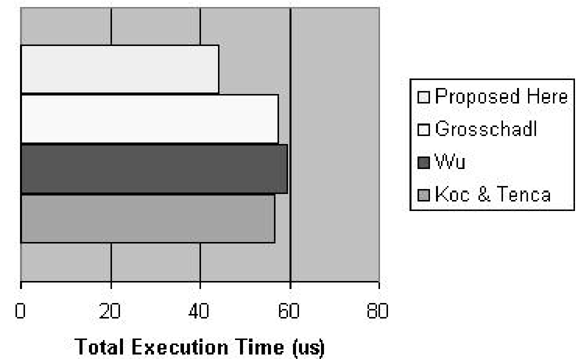


Fig. 9. Pipelined organization for the multiplier.

Figure 8 [7]

and area, we have outperformed all previously mentioned and proposed architectures. This implementation provides a 45smallest. Graph 1 shows a graph of the execution times of 3 architectures compared to the one we have proposed. With the precision at 256 bits, and a word size of 8 bits, our architecture outperforms them all. (Numbers are all false).

Execution Time Comparison



Graph1

VII. Conclusions

The purpose of this study was to explore what has been done in multiplication architectures, and then provide a new solution with all of the benefits and an increase in speed and a reduction in area. Montgomery multiplication is key in cryptographic applications today, and a unified architecture is becoming almost required. Here we have shown that an architecture can be unified, scalable, word-based, bit-serial,

and require no final subtractions, in addition to being extremely fast and small.

References

[1] H. Wu, "Montgomery multplier and squarer in gf(2m̂)," *Cryptographic Hardware and Embedded Systems (CHES)*, pp. 264–276, 2000.

[2] J. Grossschadl, "A bit-serial unified multiplier architecture for finite fields gf(p) and gf(2m̂)," *Cryptographic Hardware and Embedded Systems (CHES)*, pp. 206–223, 2001.

[3] E. Savas A.F. Tenca and C.K. Koc, "A scalable and unified multiplier architecture for finite fields gf(p) and gf(2m̂)," *Cryptographic Hardware and Embedded Systems (CHES)*, pp. 277–292, 2000.

[4] C.D. Walter, "Montgomery exponentiation needs no final subtractions," *Electronic Letters (ELL)*, vol. 35, no. 21, pp. 1831–1832, October 1999.

[5] G. Hachez and J.-J. Quisquater, "Montgomery exponentiation with no final subtractions: Improved results," *Cryptographic Hardware and Embedded Systems (CHES)*, pp. 293–301, 2000.

[6] A.F. Tenca G. Todoroc and C.K. Koc, "High-radix design of a scalable modular multiplier," *Cryptographic Hardware and Embedded Systems (CHES)*, pp. 189–205, 2001.

[7] A.F. Tenca and C.K. Koc, "A word-based algorithm and scalable architecture for montgomery multiplication," *Cryptographic Hardware and Embedded Systems (CHES)*, 1999.