IMLEMENTATION OF HIGH SPEED MONTGOMERY

NARASIMHA C

[1]Research Scholar, ECE Dept, St.Mary's Group of Institutions, Hyderabad, AP-India,

E-mail:- narasimha.chityala89@gmail.com

Abstract: *- Montgomery Algorithm for modular multiplication with a large modulus has been widely used in public key cryptosystems for secured data communication. Paper presents digit-serial systolic multiplication architecture for all-one polynomials (AOP) over $GF(2^m)$ for efficient implementation of Montgomery Multiplication (MM) Algorithm suitable for cryptosystem. Analysis shows the latency and circuit complexity of the proposed architecture are significantly less than those of earlier designs for same classes of polynomials. As the systolic multiplier has the features of unidirectional data flow, regularity, and modularity this structure is well suited to VLSI implementations. Proposed multipliers have clock cycle latency of (2N-1), where N=m/L, m is the word size and L is the digit size. Digit serial systolic architecture is not based on MM algorithm over $GF(2^m)$ is reported before. This architecture is also compared to two well known digit serial systolic architectures.*

Keywords: *Montgomery multiplier, Systoli multiplier, Xilinx, VHDL, Spartan-3e FPGA.*

1. INTRODUCTION TO RISC

Many Public Key Cryptographic (PKC) algorithms, such as RSA, Di_e-Hellman, and Elliptic Curve Cryptography, it requires modular multiplication of very large operands (sizes from 160 to 4096 bits) as their core arithmetic operation. For this operation to be fast, general purpose processors are not always the best choice. This is why specialized hardware, i.e., in the form of cryptographic co-processors, are more attractive.

Based upon the analysis of recent publications on hardware design for modular multiplication, this presents a new architecture that is scalable with respect to word size and pipelining depth. To our knowledge this is the reset time a word based algorithm for Montgomery's method is realized using high-radix bit-parallel multipliers that can perform two different types of arithmetic, (1) Integer arithmetic for operations in rings Zn or finite fields GF(p), and (2) Binary polynomial arithmetic for finite fields GF(2n) in a single unified architecture.Earlier have relied mostly on bit serial multiplication in combination with massive pipelining, or Radix-8 multiplication with a limitation to integer arithmetic. Our approach is centered around the notion that the optimal delay in bit-parallel multipliers grows with logarithmic complexity with respect to the operand size n, e.g. O(log3=2 n), while the delay of bit serial implementations grows with linear complexity O(n). Based on this observation we expect our design to be comparable in performance with other and ultimately outperform them for large values of w.

1.1 History

Since its conception in 1976 by Whitfield D and Martin Hellman [DH76] Public Key Cryptography has a long way. Many algorithms and standards have been proposed and implemented. The entire concept of e-Commerce" is based on the availability of reliable and secure methods for not only encryption, but also authentication, and integrity. With the ongoing digital revolution and advances in high performance computing, powerful desktop computer systems are available to almost everybody at low cost.

While there has always been a demand for hardware implementations of public key cryptography, it has risen dramatically in recent years, due to a paradigm shift in communications, from wire bound to wireless. New and small handheld devices with low power consumption and more and more features keep appearing. Those devices do not possess the computing power of desktop computers, but still require strong security mechanisms. The specialized cryptographic hardware comes into play. With the aforementioned multitude of different algorithms and standards, it is essential for any such hardware to support the necessary arithmetic primitives needed by those algorithms.

1.2 Modular Multiplication in Public Key Cryptosystems

The majority of the currently established Public-Key Cryptosystems (RSA, Di_e-Hellman, Digital Signature Algorithm (DSA), Elliptic Curves (ECC), etc.) require modular multiplication in finite fields as their core operation which accounts for up to 99% of the time spent for encryption and decryption. In order to improve the performance of the overall cryptosystem, therefore it is crucial to optimize modular multiplication.

One method of modular multiplication that is particularly suitable for those cryptosystems mentioned above is Montgomery Multiplication. It is a method that avoids the division that is usually necessary for finding the remainder, at the cost of an additional multiplication.As division is much more costly than multiplication, this method represents a significant improvement over regular modular multiplication.

1.3 Key sizes and Complexity of Public Key Schemes

Current Public Key schemes are computationally very expensive. On one side this has to do with the complexity of the operations involved, e.g. modular multiplication, modular inversion, etc. The length of the operands involved in such operations is much larger than the word size of traditional microprocessors. They range between 160 bits for Elliptic Curve-based cryptosystems and 2048 bits or more for RSA or Difie-Hellman. Operations on such large operands naturally need to be broken down into word based multi-precision operations. Speed complexity of operations like multiplication is given as O(n2), that means the time necessary for multiplication grows quadratic-ally with the operand length n in words.

To illustrate the complexity a little more, the following short example shows an estimate x of the number of integer multiplications necessary for the central operation of two popular Public Key methods. One of them is 160 bit Elliptic Curve scalar point multiplication and the other a 1024 bit RSA modular exponentiation. For more information on selecting key sizes for cryptographic applications. For simplicity we assume a word size w of 32 bits and only count simple integer multiplications. Even modular squaring can be implemented faster than multiplication; no distinction is made. The number of integer multiplications p necessary for one full precision modular multiplication can be approximated as 2n2, where n = [N/w].

2. FINITE GALOIS FIELD GF

2.1 Finite Galois Field GF(2^m)

Full Finite (Galois) fields GF(2^m)have widely applied to error control coding , in particular for the Reed-Solomon codes, and public-key cryptography. They have interesting properties that are absent in other conventional number systems, such as all operations are carry-free, the word length is constant and no round-off problem. Accordingly, the operation of GF(2^m)is simpler than one of GF(p) or GF(p^m), and also has wide application. Important operations in finite fields are addition, multiplication and inversion. Addition in GF(2^m)is simple and straightforward. This work focuses on the design of low-complexity and scalable hardware architectures for computing multiplications over GF(2^m) since inversion can be performed using repeated multiplication and addition. In this chapter, the back ground of multiplier architectures over GF(2^m)and the construction of this thesis are introduced.

2.2 Arithmetic over Binary Extension Fields GF(2^m)

Finite fields GF(2^m)with m > 1 are often represented in polynomial basis representation. The special case where p = 2 is usually referred to as binary extension fields. This class of finite fields is particularly suitable for arithmetic on digital computers because of the straightforward representation of coefficients as binary bit strings. Arithmetic in binary extension fields has different properties than prime fields, but is structurally similar. The role of the prime modulus is adopted by an Irreducible polynomial f(x) of degree m.

3. REPRESENTATION OF ELEMENTS IN GF(2^m)

GF(2m) is an extension field of GF(2), where m is a positive integer. The extensive field has 2m elements. Any element A of GF(2m) can be represented as a sum of m linearly independent elements in fields, is given as:

$$A=a0+a1x+a2x2+...+am-1xm-1,$$

where, GF(2), for $0 \le i \le m-1$, and x is a root of an irreducible polynomial over GF(2) of degree m. The set of {1, x, $x2$,…, $xm-1$} is called a polynomial basis. Moreover, suppose that is linearly independent, then, form basis called a normal basis. Polynomial and normal bases are widely used to represent field elements.

3.1 Addition

Addition of two binary polynomials is done as the addition of its coefficients modulo two without any carries

$$A(x) + B(x) = \sum_{i=0}^{m} (a_i + b_i)x^i \bmod 2$$

This in terms of logic circuits directly translates into XOR combinations of the coefficients

$$A(x) + B(x) = \sum_{i=0}^{m} (a_i \oplus b_i)x^i$$

It is obvious that addition of binary polynomials can be implemented in hardware more efficiently. Subtraction is the exact same operation, since each coefficient is its own additive inverse.

3.2 Multiplication

Multiplication in GF(2n) is slightly more complex:

$$A(x)B(x) = \sum_{i=0}^{m} \sum_{j=0}^{m} (a_i b_j)x^{i+j} \bmod f(x)$$

With p = 2 the partial products a_j, b_j, are the outputs of simple logical AND gates in hardware and summed up as before by XOR gates, according to their position in the resulting polynomial. Modular reduction takes place by adding (subtracting) f(x)x^{k-1} repetitively to the result, as long as the degree of the result k ≥ m. This approach is very simplistic, but achieves the desired effect. More efficient method for modular reduction is available in form of Montgomery's algorithm. Only a couple of minor adoptions are necessary to make this integer arithmetic algorithm work in conjunction with binary polynomials. In conclusion it can be said that except for the modular reduction, binary polynomial arithmetic is very similar in structure to integer arithmetic. The only big difference is the absence of any sort of carry propagation, which makes this type of arithmetic so attractive for high speed hardware implementations.

3.3 Montgomery Multiplication

In 1985 Peter L. Montgomery proposed a method [Mon85] for modular multiplication using Residue Number System (RNS) representation of integers. It replaces the costly division operation usually needed to perform modular reduction by simple shift operations, at the cost of having to transform the operands into the RNS before the operation and re-transforming the result thereafter. A radix R is selected to be two to the power of a multiple of the machine word size and greater than the modulus, i.e. R = 2kw > M. For the algorithm to work R and M need to be relatively prime, i.e. it must not have any common non-trivial divisors. With R a power of two, this requirement is easily satisfied by selecting an odd modulus. This also fits in nicely with the cryptographic algorithms that we are targeting, where the modulus is either a prime {always odd with the exception of 2} or the product of two primes and hence it is odd. RNS portrayal (representations) of integers are called M-residues and are usually denominated as the integer variable name with a bar above it.

An integer a is transformed into its corresponding M-residue $\bar{a}$ by multiplying it by R and reducing modulo M. The back-transformation is done in an equally straight-forward manner by dividing the residue by R modulo M. Hence we have the following equations as transformation rules between the integer and the RNS domain:

$$\bar{a} = aR \pmod{M}$$

$$a = \bar{a}R^{-1} \pmod{M}$$

Montgomery Multiplication can written simply as the product of two M-residues divided by the radix modulo M:

$$\bar{c} = \bar{a}\bar{b}R^{-1} \pmod{M}$$

Division by the Radix is necessary to make the result again an M-residue. This becomes more obvious as we expand the equation in the following way, in which we also introduce the function name MM (Op1, Op2) for the Montgomery Multiplication algorithm:

$$\begin{aligned} \bar{c} &= MM(\bar{a}, \bar{b}) \\ &= \bar{a}\bar{b}R^{-1} \pmod{M} \\ &= aRbRR^{-1} \pmod{M} \\ &= (ab)R \pmod{M} \\ &= cR \pmod{M} \end{aligned}$$

Assuming we have an implementation for of the MM algorithm at our disposal, it looks as if we still need a method to perform regular modular reduction if we want to transform integer variables into their respective M-residues. However, once the precision, and therefore the radi R, is fixed for the implementation, we can use the pre-computed constant R2 (mod M) in conjunction with the MM algorithm for transformation purposes:

$$\begin{aligned} \bar{a} &= MM(a, R^2) \\ &= aR^2R^{-1} \pmod{M} \\ &= aR \pmod{M} \end{aligned}$$

$$\begin{aligned} a &= MM(\bar{a}, 1) \\ &= aRR^{-1} \pmod{M} \end{aligned}$$

The benefits of Montgomery Multiplication over classical methods involving division are not overly evident for applications with only a few modular multiplications. However, for algorithms in which a lot of modular multiplications need to be per formed with respect to the same modulus, the performance gain is much more obvious, since the ratio between transformation overhead and actual modular arithmetic is much lower.

For the sake of simplicity we will drop the "bar" notation for distinguishing M-residues from integers throughout the remainder of this thesis, since the transformation to and from the RNS is not of significance here. When it becomes necessary to distinguish two domains, and extra indication will be provided.

3.4 Inherent Parallelism in Montgomery Multiplication

The common trade of when it comes to implementation of an algorithm in hardware versus one in software is that flexibility is sacrificed for speed. In many cases, however, the use of a particular algorithm is very specific to a certain application, so that a loss of flexibility is a low price to pay for the performance improvement. An additional benefit is

that the hardware solution can be optimized for reduced power consumption, since only a subset of

all the features available on general purpose processors will be necessary.

There are a number of different ways to improve on the performance of complex actions in hardware, where arithmetic and logic operations take at least one clock cycle each in software evaluation, where number of logic operations can be combined into a single clock cycle in custom built hardware. Intermediate results can be stored in fast local registers instead of in a standard register fille which may be placed far away. Loop-Unrolling may be used to perform multiple iterations of a task in a single clock cycle where this would help in balancing the critical paths of different tasks. Data independent shift operations or even permutations can be hardwired and therefore cost nothing, and they are slow in software.

Perhaps the most efficient way of speeding up complex operations in hardware, however, is through the utilization of inherent parallelisms that a particular algorithm offers. Identifying these parallelisms is only the first step, during which automated tools for algorithm analysis and transformation might be helpful. It should be noted; however, that the success rate of such tools is limited and often even thorough analysis by hand is difficult.

In the following sections three possible ways are identified, how to parallelize the Montgomery algorithm at different levels. The degree of parallelism that can be achieved varies with the data dependency of a particular level. Sometimes, when real simultaneity is impossible due to dependence on output from an earlier step, processes can still be coinciding in time, like use of pipelining. Depending on the definition this can still be viewed as a form of parallelism.

The first and innermost level of parallelism can be found inside the high radix digit multiplier, is a core component of this architecture. One level higher the parallel computation of the product A[i] B[j] and the product used for reduction UM[i] are

computed completely in parallel, once the initialization phase is over. Finally, pipelining of multiple MM Units constitutes yet another level of parallelism.

TOP LEVELDESIGN

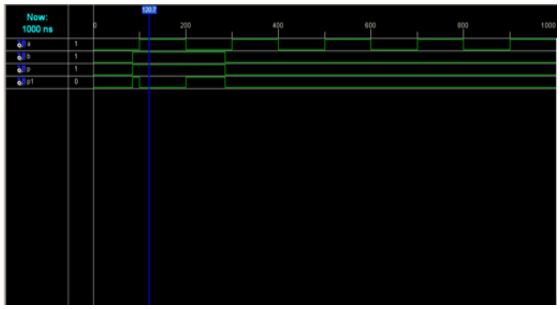


Fig. 3.1 simulation result of cell

4. CONCLUSION

This thesis proposed a systematic design of digit-serial systolic multipliers for AOPs for efficient implementation of MM algorithm suitable for the cryptosystem. The latency and circuit complexity of the proposed architecture are significantly less than the previously proposed digit-serial systolic multipliers for the same class of polynomials.

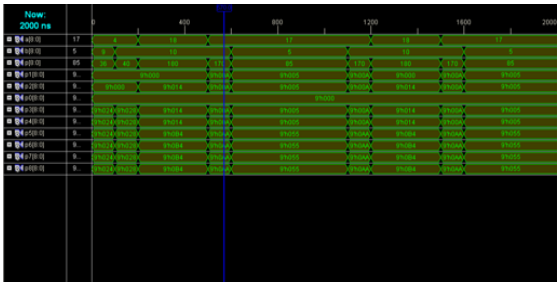
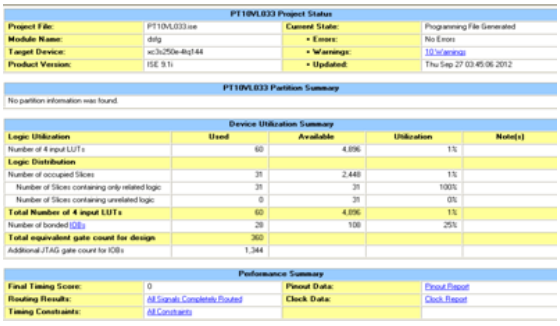


Fig.3.2 Simulation result of Existed sfg



PT10VL033 Project Status			
Project File:	PT10VL033.xise	Current State:	Programming File Generated
Module Name:	dsfg	• Errors:	No Errors
Target Device:	xc3s250e-4tq144	• Warnings:	10 Warnings
Product Version:	ISE 9.1i	• Updated:	Thu Sep 27 03:45:06 2012

PT10VL033 Partition Summary

No partition information was found.

Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Number of 4 input LUTs	60	4,896	1%	
Logic Distribution				
Number of occupied Slices	31	2,448	1%	
Number of Slices containing only related logic	31	31	100%	
Number of Slices containing unrelated logic	0	31	0%	
Total Number of 4 input LUTs	60	4,896	1%	
Number of bonded IOBs	28	108	25%	
Total equivalent gate count for design	360			
Additional JTAG gate count for IOBs	1,344			

Performance Summary			
Final Timing Score:	0	Pinout Data:	Pinout Report
Routing Results:	All Signals Completely Routed	Clock Data:	Clock Report
Timing Constraints:	All Constraints		

Fig 3.3 Design summary report of proposed DSFG

This is the first design for digit-serial systolic multiplier based on MM algorithm over $GF(2^m)$. This thesis proposed a systematic design of digit-serial systolic multipliers proposed architecture is high performance than the Existed serial systolic multipliers for the same class of polynomials. Total design is coded in VHDL language and it is functionally verified using ISE simulator. The synthesis done using Xilinx9.1 synthesis tool. We generate a synthesis reports for both SFG and DSFG based on Spartan 3e FPGA. The Experimental results compared in above table.

Future scope

Both measured and simulation results have shown that systematic design of digit-serial systolic multipliers are not as effective as the serial systolic multipliers for the same class of polynomials at low to moderate bit widths. Increases the bit width we get better results.

5. REFERENCES

[1]Low Complexity Digit Serial Systolic MontgomeryMultipliers for Special Class of Somsubhra Talapatra, Hafizur Rahaman, and Jimson Mathew , IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 18, NO. 5, MAY 2010

[2] C. Paar, P. Fleischmann, and P. Soria-Rodriguez, "Fast arithmetic for public-key algorithms in Galois fields with composite exponents,"IEEE Trans. Computers, vol. 48, no. 10, pp. 1025–1034, Oct. 1999.

[3]R. Lidl and H. Niederreiter, Introduction to Finite Fields and Their Applications. Cambridge, U.K.: Cambridge Univ. Press, 1994.

[4] A. Reyhani-Masoleh and M. A. Hasan, "Low complexity bit parallel architectures for polynomial basis multiplication over GF(2m)" IEEE Trans. Computers, vol. 53, no. 8, pp. 945–959, Aug. 2004.

[5] D. Hankerson, A. Menezes, and S. Vanstone, Guide to Elliptic Curve Cryptography. New York: Springer-Verlag, 2004.

[6] W. Diffie and M. E. Hellman, "New directions in cryptography," IEEE Trans. Inf. Theory, vol. 22, no. 6, pp. 644–654, Nov. 1976.

[7] P. L. Montgomery, "Modular multiplication without trial division," Math. Computation, vol. 44, pp. 519–521, 1985. C. K. Koc and T. Acar, "Montgomery multiplication in GF (2M)," Designs, Codes, Cryptography, vol. 14, no. 1, pp. 57–69, Apr. 1998.

[8] H.Wu, "Montgomery multiplier and squarer for a class of finite fields,"IEEE Trans. Computers, vol. 51, no. 5, pp. 521– 529, 2002.