International Journal of Engineering Research and Science & Technology

ISSN : 2319-5991
Vol. 3, No. 4
November 2014

www.ijerst.com

Email: editorijerst@gmail.com or editor@ijerst.com

International Journal of Engineering Research and Science & Technology

ISSN 2319-5991 www.ijerst.com
Vol. 3, No. 4, November 2014

Research Paper

REALIZATION OF EFFICIENT POWER DELAY PRODUCT MONTGOMERY MULTIPLIER BASED ON PENTANOMIALS

Satish Moddirala[1]* and S Suman[2]

Corresponding Author:* ***Satish Moddirala* ✉ *maddirala484@gmail.com*

In this paper, a low latency systolic Montgomery multiplier over GF(2m) based on irreducible pentanomial is presented. An efficient algorithm is presented to decompose the multiplication into a number of independent units to facilitate parallel processing. Besides, a novel so-called "pre-computed addition" technique is introduced to further reduce the latency. The proposed design involves significantly less area-delay and power-delay complexities compared with the best of the existing designs. It has the same or shorter critical-path and involves nearly one-fourth of the latency of the other in case of the National Institute of Standards and Technology recommended irreducible pentanomials. The functionality is verified using ISE simulator and the synthesis is carried out using XILINX ISE 12.3i with VERILOG HDL.

Keywords: Pentanomial, Montgomery Multiplier, Finite Field, Verilog

INTRODUCTION

A finite field is also often known as a Galois field, after the French mathematician Pierre Galois. A Galois field in which the elements can take q different values is referred to as GF(q). The formal properties of a finite field are there are two defined operations, namely addition and multiplication.The result of adding or multiplying two elements from the field is always an element in the field.One element of the field is the element zero, such that a + 0 = a for any element a in the field.One element of the field is unity, such that a • 1 = a for any element a in the field.For every element a in the field, there is an additive inverse element -a, such that a + (- a) = 0. This allows the operation of subtraction to be defined as addition of the inverse.For every non-zero element b in the field there is a multiplicative inverse element b-1 such that b b-1= 1. This allows the operation of division to be defined as multiplication by the inverse.The associative [a + (b + c) = (a + b) + c, a • (b • c) = [(a • b) • c], commutative [a + b = b + a, a • b = b • a], and distributive [a • (b + c) = a • b + a • c] laws apply.These properties cannot be satisfied for all possible field sizes. They can, however, be satisfied if the field size is any prime number or any integer power of a prime.

[1] M.Tech. Student, Department of ECE, Chirala Engineering College, Chirala 523155, Prakasam Dt., AP.
[2] Assistant Professor, Department of ECE, Chirala Engineering College, Chirala 523155, Prakasam Dt., AP.

The finite fields are classified as follows:

- The number of elements in a finite field is of the form p^n, where p is a prime number called the characteristic of the field, and n is a positive integer.
- For every prime number p and positive integer n, there exists a finite field with p^n elements.
- Any two finite fields with the same number of elements are isomorphic. That is, under some renaming of the elements of one of these, both its addition and multiplication tables become identical to the corresponding tables of the other one.

This classification justifies using a naming scheme for finite fields that specifies only the order of the field. One notation for a finite field is F_p^n or Fpn. Another notation is GF(p^n), where the letters GF stand for "Galois field".

A prime power field with p = 2 is also called a binary field.

First we consider fields where the size is prime, i.e., with n = 1. Such a field is called a prime field, and is canonically isomorphic to the ring Z/pZ, the set of integers modulo p. It is sometimes denoted Zp, but within some areas of mathematics, particularly number theory, this may be confused with the same notation for the ring of p-adic integers. The ring Z/pZ is a field since it contains a multiplicative inverse for each element other than zero (an integer that, multiplied by the element modulo p yields 1), and it has a finite number of elements (being p), making it a finite field.

Even though all fields of size p are isomorphic to Z/pZ, for $n \geq 2$ the ring of integers modulo pn, Z/pnZ, is not a field. The element p is nonzero and has no multiplicative inverse modulo pn. By comparison with the ring Z/4Z of size 4, the underlying additive group of the field (Z/2Z)[T]/(T2 + T + 1) of size 4 is not cyclic, but is isomorphic to the Klein four-group.

No fields exist for which the size is not a prime power. For example, there is no field with 6 elements. Given a prime power q = pn, we may explicitly construct a finite field with q elements as follows. Select a monic irreducible polynomial f(T) of degree n in Fp[T]. (Such a polynomial is guaranteed to exist, once we know that a finite field of size q exists: just take the minimal polynomial of any primitive element for that field over the subfield Fp.) Then Fp[T]/(f(T)) is a field of size q. Here, Fp[T] denotes the ring of all polynomials in T with coefficients in Fp, (f(T)) denotes the ideal generated by f(T), and the quotient is meant in the sense of quotient rings — the set of polynomials in T with coefficients in Fp mod (f(T)).

MONTGOMERY MULTIPLICATION

Montgomery multiplication is a method for computing ab mod m for positive integers a, b, and m. It reduces execution time on a computer when there are a large number of multiplications to be done with the same modulus m, and with a small number of multipliers. In particular, it is useful for computing a^n mod m for a large value of n. The number of multiplications modulo m in such a computation can be reduced to a number substantially less than n by suc-cessively squaring and multiplying according to the pattern of the bits in the binary expression for n ("binary decomposition"). But it can still be a large enough number to be worthwhile speeding up if possible. The difficulty is in the reductions modulo m, which are, essentially, division operations, which are

costly in execution time. If one defers the modulus operation to the end, then the products will grow to very large numbers, which slows down the multiplications and also the final modulus operation.

To use Montgomery multiplication, we must have the multipliers a and b less than the modulus m. We introduce another integer r which must be greater than m, and we must have gcd(r, m) = 1. The method, essentially, changes the reduction modulo m to a reduction modulo r. Usually r is chosen to be an integral power of 2, so the reduction modulo r is simply a masking operation; that is, retaining the lg(r) low-order bits of an intermediate result, and discarding higher order bits. If r is a power of 2, we must have m odd, to satisfy the gcd requirement. (Any odd value from 3 to r " 1 is acceptable.)

In arithmetic computation, Montgomery reduction is an algorithm introduced in 1985 by Peter Montgomery that allows modular arithmetic to be performed efficiently when the modulus is large (typically several hundred bits).

A single application of the Montgomery algorithm (henceforth referred to as a "Montgomery step") is faster than a "naive" modular multiplication

$c \equiv a \times b \pmod{n}$

Because numbers have to be converted to and from a particular form suitable for performing the Montgomery step, a single modular multiplication performed using a Montgomery step is actually slightly less efficient than a "naive" one. However, modular exponentiation can be implemented as a sequence of Montgomery steps, with conversion only required once at the start and once at the end of the sequence. In this case the greater speed of the Montgomery steps far outweighs the need for the extra conversions.

Working with n-digit numbers to base d, a Montgomery step calculates $a \times b \div d^n \pmod{r}$. The base *d* is typically 2 for microelectronic applications, 2^8 for 8-bit firmware,[4] or 2^{32} or 2^{64} for software applications. For the purpose of exposition, we shall illustrate with $d = 10$ and $n = 4$.

To turn this into a modular operation with a modulus r, add, immediately before each shift, whatever multiple of r is needed to make the value in the accumulator a multiple of 10.The result will be that the final value in the accumulator will be an integer (since only multiples of 10 have ever been divided by 10) and equivalent (modulo r) to $472 \times a \div 10000$.Finding the appropriate multiple of r is a simple operation of single-digit arithmetic. When working to base 2, it is trivial to calculate: if the value in the accumulator is even, the multiple is 0 (nothing needs to be added); if the value in the accumulator is odd, the multiple is 1 (r needs to be added).The Montgomery step is faster than the methods of "naive" modular arithmetic because the decision as to what multiple of r to add is taken purely on the basis of the least significant digit of the accumulator. This allows the use of carry-save adders, which are much faster than the conventional kind but are not immediately able to give accurate values for the more significant digits of the result.

This note explains the theory and practice of Montgomery multiplication. Montgomery multiplication is a method for computing *ab* mod *m* for positive integers *a*, *b*, and *m*.[1] It reduces execution time on a computer when there are a large number of multiplications to be done with the same modulus *m*, and with a small number of multipliers. In particular, it is useful for computing a^n mod *m* for a large value of *n*. The number of multiplications modulo *m* in such a

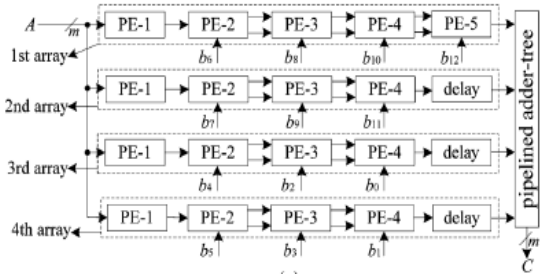
computation can be reduced to a number substantially less than *n* by successively squaring and multiplying according to the pattern of the bits in the binary expression for *n* ("binary decomposition"). But it can still be a large enough number to be worthwhile speeding up if possible. The difficulty is in the reductions modulo *m*, which are, essentially, division operations, which are costly in execution time. If one defers the modulus operation to the end, then the products will grow to very large numbers, which slows down the multiplications and also the final modulus operation.

To use Montgomery multiplication, we must have the multipliers a and b less than the modulus m. We introduce another integer r which must be greater than m, and we must have gcd(r, m) = 1. The method, essentially, changes the reduction modulo m to a reduction modulo r. Usually r is chosen to be an inte-gral power of 2, so the reduction modulo r is simply a masking operation; that is, retaining the lg(r) low-order bits of an intermediate result, and discarding higher order bits. If r is a power of 2, we must have m odd, to satisfy the gcd requirement. (Any odd value from 3 to r " 1 is acceptable.)

GALOIS FIELD IN CRYPTOGRAPHY

This paper introduces the basics of Galois Field as well as its implementation in storing data. This paper shows and helps visualizes that storing data in Galois Fields allows manageable and effective data manipulation, where it focuses mainly on application in computer cryptography. Details on the algorithm for Advanced Encryption Standard (AES), which is an example of computer cryptography that utilizes Galois Field, will also be included.

Figure 1: Low Latency Montgomery Multiplier Design



Galois Field, named after Evariste Galois, also known as finite field, refers to a field in which there exist finitely many elements. It is particularly useful in translating computer data as they are represented in binary forms. That is, computer data consist of combination of two numbers, 0 and 1, which are the components in Galois field whose number of elements is two. Representing data as a vector in a Galois Field allows mathematical operations to scramble data easily and effectively.

Binary System

In the binary numeral system or base-2 number system, we represent each value with 0 and 1. To convert a decimal numeral system or base-10 number system into binary system, we need to represent a decimal in terms of sums of $a_n 2^n$. That is, if x is the said decimal number then we wish to have

$$x = \sum_{n \in \mathbb{N}} a_n 2^n$$

The coefficients a_n is then written in descending order of *n* and all leading zeros are then omitted. The final result becomes the binary representation of the decimal *x*. Ultimately, binary system offers an alternative way of representing the elements of a Galois Field. Both the polynomial and binary representation of an

element have their own advantages and disadvantages.

Example:

$$19 = \ldots + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

So the binary representation of 19 is 10011 while the elements of $gf(2^3)$ in binary are

$gf\ (2^3)$ = (001, 010, 011, 100, 101, 110, 111)

Bit and Byte

Each 0 or 1 is called a bit, and since a bit is either 0 or 1, a bit is an element of $gf(2)$. There is also a byte which is equivalent to 8 bits thus is an element of $gf(2^8)$. Since we will be focusing on computer cryptography and as each datum is a series of bytes, we are only interested in Galois Field of order 2 and 2^8 in this paper.

Because computer stores data in bytes, each binary number must be 8 bits long. For number that is less than 8 bits long, leading zeros are added. It follows as well that the biggest number 1 byte can store is 11111111 = $2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$ = 255. Following from the preceding example, 19 is stored as 00010011 in byte.

ASCII

ASCII stands for American Standard Code for Information Interchange. Since there are exactly 255 characters in ASCII, we can uniquely assign each character to an element in $gf(2^8)$ or represent it as a byte. The most frequently used ASCII codes and their values are given in the following table.

where Dec indicates the decimal representation (which can be converted to binary and stored as a byte) and Chr stands for character.

Dec	Chr	Dec	Chr	Dec	Chr	Dec	Chr	Dec	Chr	Dec	Chr
32	Space	48	0	64	@	80	P	96	‘	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	”	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	’	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o		

RESULTS

Figure 2: Waveform of Low Latency Montgomery Multiplier

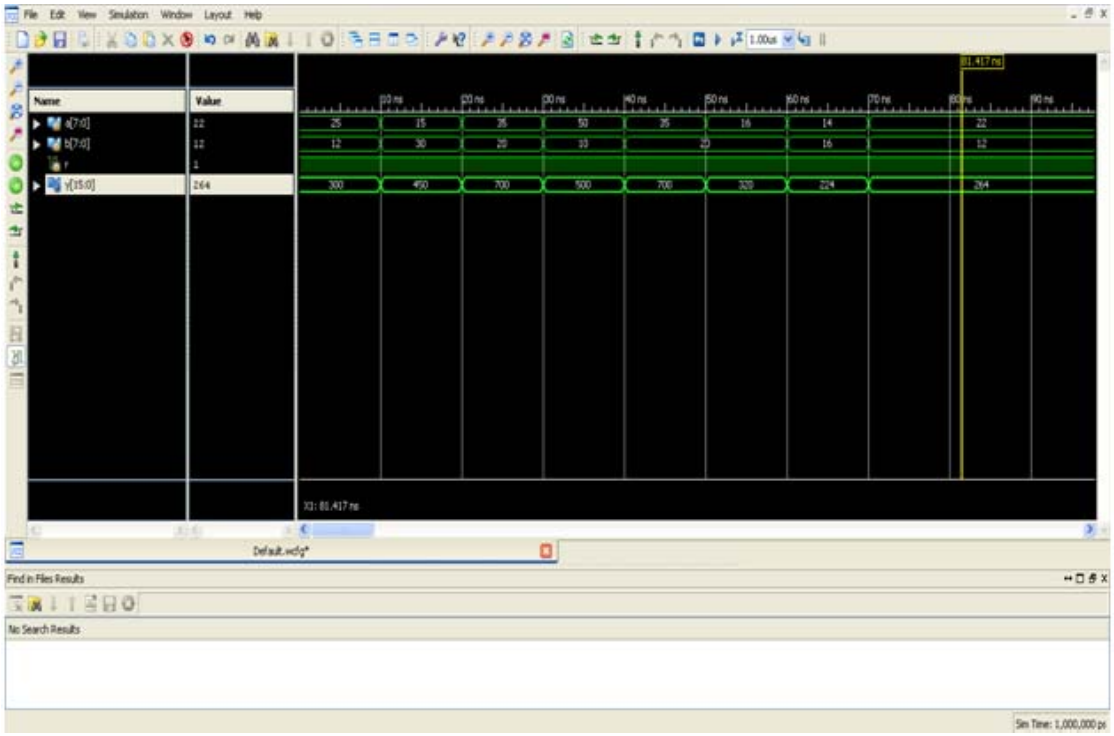
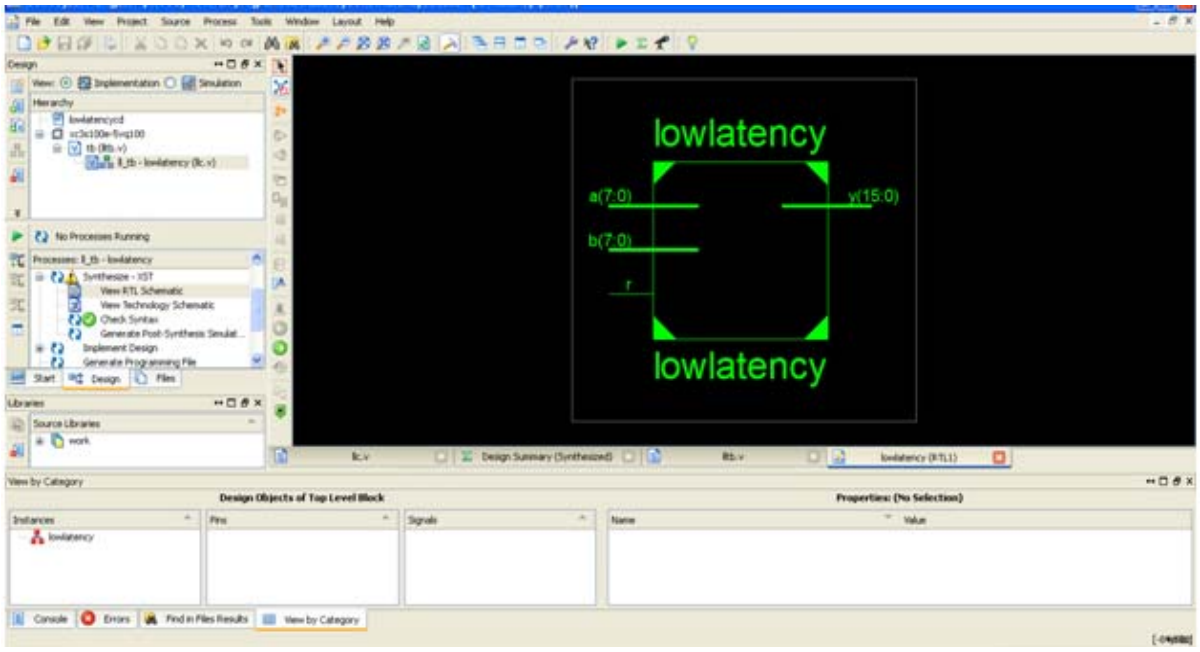


Figure 3: RTL Schematic of Low Latency Montgomery Multiplier



CONCLUSION

In this paper, we have presented a novel PCA technique and modular reduction scheme for Montgomery multiplication over GF(2^m)based on irreducible pentanomials. To illustrate the efficiency of the proposed approach, it has designed the multiplier for the irreducible pentanomial $f(x) = x^{13} + x^4 + x^3 + x^2 + 1$, for simplicity of procedding. We have decomposed the Montgomery multiplication into two concurrent blocks and we have derived a lower-latency multiplier using the proposed modular reduction scheme using PCA. The proposed design involves significantly less area-delay and power-delay complexities than the reported multiplier for irreducible pentanomial, with nearly one-fourth of the latency of the other, for the NIST recommended pentanomials.

REFERENCES

1. Digital Signature Standard (DSS), FIPS 186–2, National Institute of Standards and Technology, 2000.
2. Lee C Y, Horng J S, Jou I C and Lu E H (2005), "A Digit-Serial Multiplier for Finite Field", *IEEE Trans. Very Large Scale Integr.(VLSI) Syst*., Vol. 13, No. 4, pp. 476–483.
3. Lee C Y, Horng J S, Jou I C and Lu E H (2005), "Low-Complexitybit-Parallel Systolic Montgomery Multipliers for Special Classes of", *IEEE Trans. Comput*., Vol. 54, No. 9, pp. 1061–1070.
4. Meher P K (2009), "On Efficient Implementation of Accumulation in Finite Field Over and its Applications", *IEEE Trans. Very Large ScaleIntegr. (VLSI) Syst*., Vol. 17, No. 4, pp. 541–550.
5. Murthy N R (2006), "Cryptographic Applications os Brahmagupta-Bhaskara Equation", *IEEE Circuits Syst. I,* Vol. 53, No. 7, pp. 1565-1571.
6. Wu H (2008), "Bit-Parallel Polynomial Basis Multiplier for New Classes Offinite Fields", *IEEE Trans. Comput*., Vol. 57, No. 8, pp. 1023–1031.
7. Zhang T and Parhi K K (2001), "Systematic Design of Original and Modified Mastrovito Multipliers for General Irreducible Polynomials", *IEEETrans. Comput*., Vol. 50, No. 7, pp. 734–749.

International Journal of Engineering Research and Science & Technology
Hyderabad, INDIA. Ph: +91-09441351700, 09059645577
E-mail: editorijlerst@gmail.com or editor@ijerst.com
Website: www.ijerst.com