Design and Implementation of a High-Speed Reconfigurable Modular Arithmetic Unit

Wei Li, Zibin Dai, Tao Chen, Tao Meng, and Xuan Yang

Institute of Electronic Technology, the PLA Information Engineering University, Zhengzhou 450004, China
try_1118@163.com

Abstract. A high-performance and dynamic reconfigurable modular arithmetic unit is presented, which provides full support to modulo $2^8/2^{16}/2^{32}$ addition and modulo$2^{32}/ 2^{16}+1/2^{32}$-1 multiplication operation. To save the hardware cost, we have adopted sharing technique to implement modular multiplication operation, and then optimized each critical block. The design has been realized using Altera's FPGA. Synthesis, placement and routing of reconfigurable design have accomplished on 0.18μm SMIC process. The result proves that the propagation time of the critical path is 6.04ns. Compared with other designs, the reconfigurable modular arithmetic unit not only supports for diverse modular arithmetic in the block ciphers, but also provides IP Core for reconfigurable cryptographic system.

1 Introduction

Block ciphers are the core mechanism of data encryption, digital signature and key management due to the characteristic of high-speed, simplified standardization and hardware realization. With the development of cipher chip design technology, the realization method of cipher algorithms is increasing gradually. The cipher processor is widely approved due to the characteristic of high-speed and flexible realization. The design scheme of reconfigurable cipher processing unit is: the hardware circuit can be reconfigured to adapt to more cipher algorithms and have a tradeoff between flexibility and efficiency.

Based on the analysis of block cipher architectures, most designs of block cipher algorithms have a few similar hardware units. In the block cipher algorithms, the frequency of modular arithmetic operation is very high. As for different block ciphers, there are large difference in the aspects of width and module. So the modular arithmetic unit not only takes up large area but also pays more delay time. Basing on the analysis of modular arithmetic operation, this paper presents a high-speed reconfigurable modular arithmetic unit(RMAU), which can be reconfigured to perform modulo $2^8/2^{16}/2^{32}$ addition and modulo$2^{32}/ 2^{16}+1/2^{32}$-1 multiplication operation.

This paper is organized as follows: section 2 presents a reconfigurable modular arithmetic unit. Section 3 optimizes the 16-bit multiplier with booth algorithm[1], leapfrog Wallace tree[2] and Ling adder[3]. Section 4 analyzes conventional adders in delay and area. Section 5 introduces a high-speed module modification circuit.

M. Xu et al. (Eds.): APPT 2007, LNCS 4847, pp. 50–59, 2007.

Section 6 shows the simulation result and compares the RMAU with reference[4],[5]. Finally, we conclude with section 7.

2 Design and Application of RMAU Unit

2.1 Analysis of Modular Operation Based on Block Ciphers

In the block cipher algorithms, the frequency of modular operation is very high. However, different algorithms have different operation mode and width. Based on the analysis of conventional block ciphers, if the width of two operands is n, module is usually 2^n or $2^n \pm 1$, and n is 8, 16 or 32. Table 1 shows the modular operation of published block cipher algorithms.

Table 1. Modular operation of block cipher

Algorithm	Addition/Subtraction	Multiplication
IDEA	Modulo 2^{16}	Modulo2^{16}+1
Blowfish	Modulo 2^{32}	
MARS	Modulo 2^{32}	Modulo2^{32}
FEAL	Modulo 2^8	
CAST-128	Modulo 2^{32}	
CAST-256	Modulo 2^{32}	
MMB		Modulo2^{32}-1
GOST	Modulo 2^{32}	
WAKE	Modulo 2^{32}	
WiderWake	Modulo 2^{32}	
RC2	Modulo 2^{16}	
RC5	Modulo 2^{32}	
RC6	Modulo 2^{32}	Modulo2^{32}
SAFER K	Modulo 2^8	
SAFER +	Modulo 2^8	
E2		Modulo2^{32}
Twofish	Modulo 2^{32}	

Based on the analysis above, conventional modular operation includes modulo 2^{16} addition or subtraction, modulo 2^{32} addition or subtraction, modulo 2^{16}+1 multiplication and modulo 2^{32}-1 multiplication. Therefore, the configurable modular operation unit should support to implement the operation above through dynamically reconfiguring the control signals.

2.2 Hardware Architecture of RMAU

The whole architecture of RMAU is illustrated in Fig.1. It contains 16-bit multiplier, adder array block, modulo 2^{16}+1 and 2^{32}-1 modification circuit and some reconfigurable control signals (C1~C5). Beside that, the function of invers block is that it can acquire the negative logic of data input under the domination of signal C3.

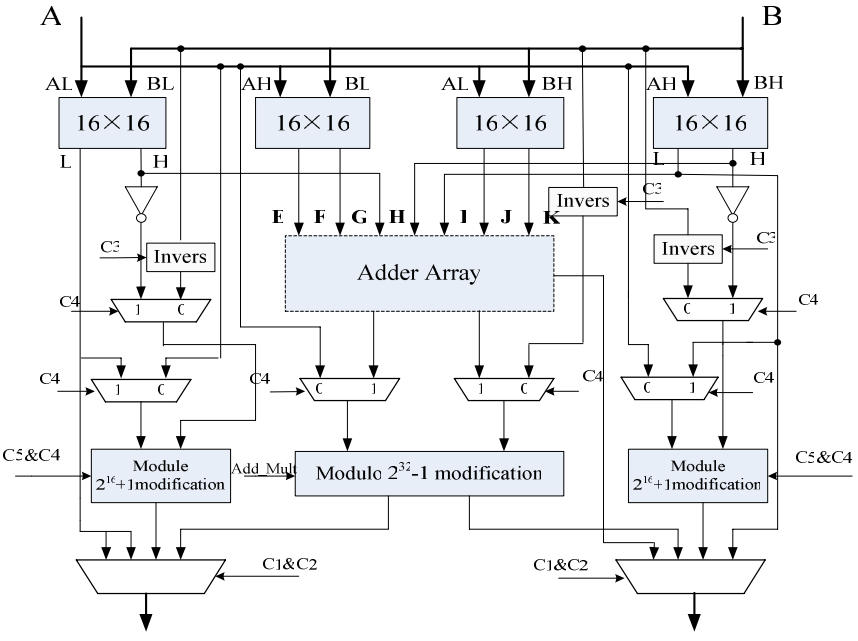

Fig. 1. The whole architecture of RMAU

Table 2. The details of C1~5's configurations

Mode	Control signal				Operation
	C1&C2	C3	C4	C5	
Config1	00	X	X	X	Modulo 2^{16} multiplication
Config2	01	X	X	X	Modulo 2^{32} multiplication
Config3	10	0	0	0	Modulo 2^{8} addition
Config4		1	0	0	Modulo 2^{8} subtraction
Config5		0	0	1	Modulo 2^{16} addition
Config6		1	0	1	Modulo 2^{16} subtraction
Config7		X	1	X	Modulo 2^{16}+1 multiplication
Config8	11	0	0	X	Modulo 2^{32} addition
Config9		1	0	X	Modulo 2^{32} subtraction
Config10		X	1	X	Modulo 2^{32}-1 multiplication

In the architecture of RMAU, it contains many reconfigurable elements whose function is determined through programmable configuration bits such as C1~C5. These elements are connected using a set of routing resources that are also programmable. In this way we can change the configurable routing to connect the blocks together to form necessary circuit. At the same time, we can save hardware sources. In this paper, through reconfiguring control signals (C1~C5), the RMAU can implement modular addition, modular subtraction and modular multiplication operation.

Moreover, it can also implement modulo $2^{16}+1$ and modulo $2^{32}-1$ multiplication operation. The details of C1~C5's configurations are presented in Table 2.

2.3 Design Scheme of RMAU

In the architecture of RMAU, 16-bit multiplier, 16-bit adder, 32-bit adder and module modification circuit are the basic blocks of RMAU. So RMAU can implement modular addition and modular subtraction operation by setting the signals C1~C5. Moreover, because it has 16-bit multiplier, RMAU can also implement modulo 2^8 and modulo 2^{16} multiplication operation. The illustration about modulo $2^{16}+1$, modulo 2^{32} and modulo $2^{32}-1$ multiplication scheme will be given as follows.

Modulo 2^{32} multiplication operation can be expressed as $A\times B \bmod 2^{32}$, where A and B are 32-bit data. AH and BH denote 16 most significant bits of A and B. AL and BL denote16 least significant bits of A and B.

(i) $A\times B=C_1 2^{32}+C_2$, where $0\le C_1, C_2<2^{32}$
(ii) $AL\times BL=D_1 2^{16}+D_2$, $AL\times BH=E_1 2^{16}+E_2$
(iii) $AH\times BL=F_1 2^{16}+F_2$, where $0\le D_1, D_2, E_1, E_2, F_1, F_2<2^{16}$

Therefore, we may get

$C_2[31:16]=(D_1+E_2+F_2)\bmod 2^{16}$, and $C_2[15:0]=D_2$

From the equation above, C_2 is the result of $A\times B \bmod 2^{32}$

As to modulo $2^{16}+1$ multiplication operation, we design the specific module modification circuit. There are two main realization methods: One is Ma algorithm[6], which can modify directly in the process of multiplication operation. Another method is Low—High algorithm scheme, which can modify the result after multiplication operation. In this paper, we adopt Low—High algorithm to realize module modification circuit. The modified scheme of modulo $2^{16}+1$ multiplication operation is:

$A\times B=(C_1 2^{16}+C_2)\bmod (2^{16}+1)$, where $0\le C_1, C_2<2^{16}$
Since $C_1 2^{16}+C_2=C_1(2^{16}+1)+(C_2-C_1)$
Therefore, $(C_1 2^{16}+C_2)\bmod (2^{16}+1)=(C_2-C_1)(2^{16}+1)$

From the equation above, C_2-C_1 is the result of $A\times B \bmod 2^{32}$

As to modulo $2^{32}-1$ multiplication operation, it can be easily seen that the modified scheme is the same as modulo $2^{16}+1$ multiplication. We adopt Low+High algorithm to modified 32-bit multiplication result. The scheme is described as follows.

$A\times B=C_1 2^{32}+C_2$, where $0\le C_1, C_2<2^{32}$
Where,$(AL\times BL=D_1 2^{16}+D_2,\ AL\times BH=E_1 2^{16}+E_2,\ AH\times BL=F_1 2^{16}+F_2,$
$AH\times BH=G_1 2^{16}+G_2)$

Here $((0\le D_1, D_2, E_1, E_2, F_1, F_2, G_1, G_2)<2^{16})$
We may get $C_1 2^{32}+C_2=C_1(2^{32}-1)+(C_2+C_1)$
Since $C_2[15:0]=D_2$, $C_2[31:16]=(D_1+E_2+F_2)\bmod 2^{16}$

$C_1[15:0]=\{((D_1+E_2+F_2)\bmod 2^{16})\text{cout}+(E_1+F_1+G_2)\bmod 2^{16}\}\bmod 2^{16}$
$C_1[31:16]=\{(((D_1+E_2+F_2)\bmod 2^{16})\text{cout}+(E_1+F_1+G_2)\bmod 2^{16})\bmod 2^{16})\text{cout}+G_1\}\bmod 2^{16}$
(Here, $(X\bmod 2^{16})$cout is the carry of $X \bmod 2^{16}$)

Based on the analysis above, it can be seen that the series carry increase the delay time of whole path. In order to minimize the delay time of critical path, we incorporate the carry together. Therefore, the equation can be described as follows.

$C_1=(D_1+E_2+F_2)\bmod 2^{16})\text{cout}+((E_1+F_1+G_2)\bmod 2^{16})\text{cout}2^{16}+(E_1+F_1+G_2)\bmod 2^{16}+G_1 2^{16}$
we can obtain: $(C_2+C_1)\bmod(2^{32}-1)$ is the result of $A\times B \bmod(2^{32}-1)$

3 Analysis and Realization of 16-Bit Multiplier

As the whole circuit architecture, 16-bit multiplier is the core block, but it has a long delay time. So we make the optimization to improve its performances in speed, area and power. The improved 16-bit multiplier we designed is shown in Fig.2. We consider the optimization of multiplier mainly from three parts: Firstly, through compared with other booth algorithms, we adopt modified booth 2 algorithm[7], Secondly, the traditional Wallace tree is replaced by leapfrog Wallace tree which minimizes the delay time of partial products compression. Finally, we adopt the Ling adder[3] to get the final result. The detailed illustration about Ling adder is depicted in 4.1.

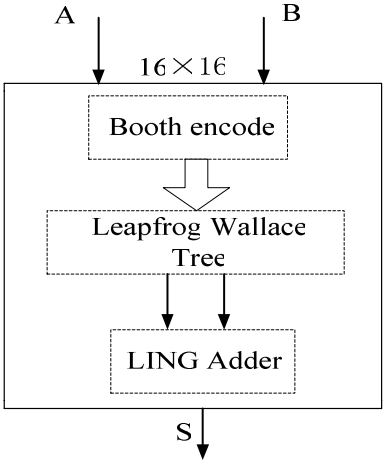

Fig. 2. Structure of 16-bit multiplier

3.1 Analysis of Booth Algorithm

Recently, a recoding scheme introduced by Booth [8] reduces the number of partial products by about a factor of two. But there is different characteristic in various booth algorithms. The analysis result is shown in Table 3.

Table 3. Analysis of booth algorithms

Algorithm type	Partial products number	Required extra circuit
None	16	None
Booth3	6	Shifting, inverting, 3M.
Booth4	5	Shifting, inverting, 3M, 5M, 6M, 7M.
Redundant Booth3	7 constant+2 carry =9	Shifting, inverting, 3M.
Modified Booth2	9	Shifting, inverting.

Based on the analysis above, booth algorithm has an obvious advantage in partial products compression. As shown in Table 3, booth 4 algorithm produces 4 partial products. But it requires more multiple generation circuit including 3M, 5M, 6M and 7M; Redundant booth 3 algorithm produces 6 partial products, and it requires 1 bias constant and two carry combining numbers. Moreover, it also requires multiple generation circuit; booth 3 algorithm products 6 partial products, and requires 3M generation circuit.

Modified booth 2 algorithm produces 9 partial products. But all of the multiples can be produced using simple shifting and complementing. So the modified booth 2 algorithm shows advantage in speed and area compared to other booth algorithms.

3.2 Analysis of Leapfrog Wallace Tree

As we know, Wallace tree architecture can enhance the compressing speed of partial products. The traditional Wallace tree architecture is shown in Fig.3.(a). Detailed illustration is shown in references [9]. In this paper, we have adopted the leapfrog Wallace tree (as shown in Fig.3.(b))which has a high performance in speed. Furthermore, it reduces the spurious switching in the circuit to save power dissipation.

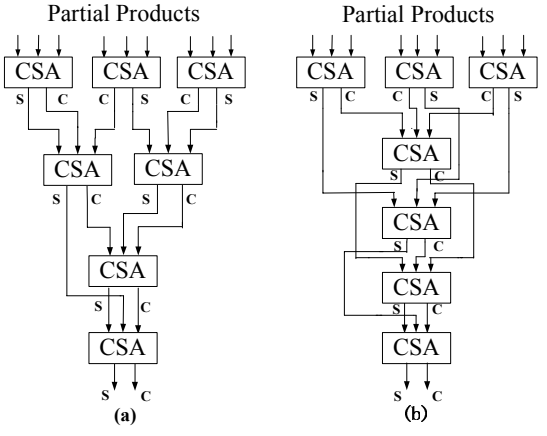

Fig. 3. Structure of Wallace tree

An improved method is proposed through adopting leapfrog Wallace tree architecture. The main idea of it is to make C and S signals enter next level CSA circuit separately. They can arrive at input port of every level CSA circuit at the same time. So the invalid inversion can be decreased. Compared with traditional Wallace tree architecture, there is not data waiting time in the leapfrog Wallace tree architecture, which accelerate compression of partial products.

4 Design of Adder Array Hardware Architecture

4.1 Analysis of Adder Unit

In the reconfigurable multiplier, 16-bit multiplier, module modification circuit and adder array all include the adder circuit. So the fast adder is important to high

performance multiplier design. Compared with conventional adder, we adopt Ling adder. In the Ling scheme, the group carry generate and propagate (G and P) are replaced by similar functions (called H and I respectively) which can be produced in fewer stages than the group G and P. we make the comparison between Ling adder and other conventional adders in area and speed performance. Synthesize adders with Synopsys Design Compiler, the result is shown in Table 4.

Table 4. Analysis of conventional adder

Adder type	Delay ns	Area μm^2
CPA	1.21	4896
BK	1.27	5121
KS	1.13	6433
CLA	1.19	5255
Synopsys Adder	1.19	3998
LING	1.07	5264

Based on the analysis above, Ling adder shows an obvious advantage in speed. The area of Ling adder is similar with BK[10], CPA and Synopsys adder. What is more, the size of Ling adder is smaller than KS[11] and CLA adder. So Ling adder is best choice in the consideration of speed and area.

4.2 Hardware Architecture of Adder Array

In the architecture of RMAU, adder array is the basic block which is shown in Fig.4. We optimize the block with CSA and LING adder. Beside, there are two combination elements in adder array. The function of it is to incorporate two input ports into one output ports. The adder array is designed to accomplish a series of addition operation. We put the result of adder array as inputs into the module modification circuit, and then accomplish modulo $2^{16}+1$ or $2^{32}-1$ multiplication.

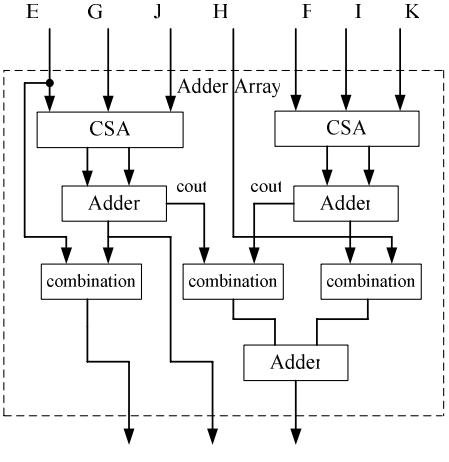

Fig. 4. Structure of adder array

5 Analysis and Implementation of Module Modification Circuit

The scheme about modulo 2^n+1 multiplication is illustrated in 2.3. We adopt Low-High algorithm. The architecture of module modification circuit is shown in Fig.5. The architecture not only provides support for modulo 2^n+1 multiplication implementing, but also accomplishes 8 or 16-bit modular addition and subtraction operation. In the process of modular multiplication operation, we segment the sum of 16-bit multiplier into n least significant bits (A) and n most significant bits (B), and then make them as inputs into the module modification circuit, which accomplishes A-B operation. As to modular addition and subtraction operation, we use signal C5 to distinguish between those two operations.

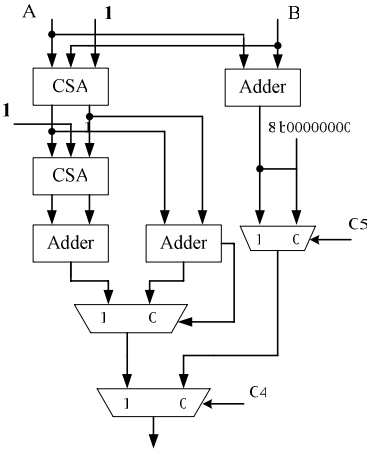

Fig. 5. Architecture of module modification circuit

6 Analysis and Comparison of Performance

6.1 Performance Evaluation of RMAU

Synthesize the reconfigurable modular arithmetic unit with Synopsys Design Compiler, based on SMIC 0.18μm CMOS technology, without regard to wire load, at worst case, the result in area and speed of RMAU are listed in Table 5

Slack indicates whether synthesis result meets the constraint. A positive Slack value denotes the design meets the required setup-time. And a negative Slack implies that the hold time of the endpoint flop is violated. As to our design, when the constraint is 8ns, the value of Slack is 0.00ns. Therefore, the maximum frequency is 125MHz.

Table 5. Performance of RMAU in area and speed

Constraint(ns)	Area(μm^2)	Slack(ns)
5	157480	-5.34
8	99697	+0.00
10	95127	+0.04

6.2 Contrast with Other Designs

In this paper, we compare RMAU with the reconfigurable elements of COBRA [4] and RELOG_DIGG[5] cipher processor. COBRA cipher processor designed special block to implement modulo 2^{32} multiplication and modulo 2^{32} addition operation. RELOG_DIGG cipher processor also designed special block to accomplish modulo $2^{16}+1$, 2^{32} multiplication and modulo 2^{32} addition operation. But in our design, RMAU can implement modular addition, modular subtraction and modular multiplication operation. Moreover, it can also implement modulo $2^{16}+1$ and modulo $2^{32}-1$ multiplication operation. RMAU has an obvious advantage in flexibility. The comparison in delay time between RMAU and special modular arithmetic element of two cipher processor are listed in Table 6. From these we can see: RMAU also has a high performance with other designs.

Table 6. The comparison with other designs

Design	Modular multiplication			Modular addition
	Modulo 2^{32}	Modulo $2^{16}+1$	Modulo $2^{32}-1$	Modulo 2^{32}
RELOG_DIGG	13.685ns	14.701ns	--	6.910ns
COBRA	5.5ns	--	--	5.0ns
RMAU	3.6ns	3.7ns	6.04ns	2.82ns

7 Conclusion

In this paper, we present a high-speed RMAU, which can achieve both short critical path and low power dissipation. Furthermore, we optimize 16-bit multiplier, module modification block and analyze conventional adder. Synthesize design with Synopsys Design Compiler, Simulation results show achievements on both high speed and small size.

References

1. Booth, A.D.: A Signed Binary Multiplication Technique. Quarterly Journal of Mechanics and Applied Mathematics 4, 236–240 (1951)
2. Nan, Y.S., Chen, O.T.: Low-power multipliers by minimizing switching activities of partial products. In: IEEE International Symposium on Circuits and Systems[C]. IEEE Circuits and Systems Society, Arizona USA, pp. 93–96 (2002)
3. Ling, H.: High-Speed Binary Adder. IBM Journal of Research and Development 25, 156–166 (1981)
4. Ying jie Qu.: The Research and Design of the Reconfigurable Logic for Cryptography[D]. In: Beijing University of Technology, Beijing China (2002)
5. Elbirt, A.J.: Reconfigurable Computing For Symmetric-Key Algorithms[D] Massachusetts: Electrical and Computer Engineering Department University of Massachusetts Lowell (2002)
6. Yu tai Ma: A Simplified Architecture for Modulo (2^n + 1) Multiplication. IEEE Transactions on Computers 47 (1998)

7. MacSorley, O.L.: High-Speed Arithmetic in Binary Computers. Proceedings of the IRE 49, 67–91 (1961)
8. Weinberger, A., Smith, J.L.: A One-Microsecond. Adder Using One-Megacycle Circuitry. IRE Transactions on Electronic Computers, 65–73 (1956)
9. Wallace, C.S.: A Suggestion for a Fast Multiplier. IEEE Transactions on Electronic Computers, 14–17 (1964)
10. Brent, P.R., Kung, T.H.: A regular layout for parallel adders. IEEE Trans. Comput. 32, 260–264 (1982)
11. Kogge, P., Stone, H.: A parallel algorithm for the efficient solution. IEEE Trans. Comput. 22, 783–787 (1973)