JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 23, 907-919 (2007)

Short Paper

Improved Modulo ($2^n + 1$) Multiplier for IDEA*

YI-JUNG CHEN[1], DYI-RONG DUH[2] AND YUNGHSIANG SAM HAN[3]
[1]*Department of Computer Science and Information Engineering*
National Taiwan University
Taipei, 106 Taiwan
[2]*Department of Computer Science and Information Engineering*
National Chi Nan University
Nantou, 545 Taiwan
[3]*Graduate Institute of Communication Engineering*
National Taipei University
Taipei, 237 Taiwan

International Data Encryption Algorithm (IDEA) is one of the most popular cryptography algorithms in date since the characteristic of IDEA is suitable for hardware implementation. This study presents an efficient hardware structure for the modulo ($2^n + 1$) multiplier, which is the most time and space consuming operation in IDEA. The proposed modulo multiplier saves more time and area cost than previous designs. With 16-bit input length, the proposed structure is 9.1% faster than that proposed by Zimmermann in 1999, and reduces the area about 35.22%. The proposed design enables IDEA to be implemented on hardware with high performance and low cost. Simulation results obtained from CPLD system developed by Altera indicate that the new design has 66Mb/sec encryption/decryption rate under 8.25MHz system clock rate with four pipeline stages for each round.

Keywords: modulo ($2^n + 1$) multiplier, logic design, IDEA, security, cryptography

1. INTRODUCTION

Data security is increasingly important given the popularity of the Internet. As a conventionally adopted symmetric cryptography algorithm, International Data Encryption Algorithm (IDEA) is wildly adopted in Internet security systems [10]. The popularity of IDEA makes the speed a significant issue for its hardware and software implementations.

The three major operations of IDEA are XOR, modulo addition, and modulo multiplication. Modulo addition sums up two inputs of n-bit length, and mods the result by 2^n. Modulo multiplication multiplies two inputs of n-bit length, and mods the result by ($2^n + 1$). Notably, an input value of zero is considered as 2^n. Therefore, the input length of modulo multiplication is $n + 1$ when the input value is zero. The encryption/decryption

Received May 31, 2005; revised September 16 & December 30, 2005; accepted February 22, 2006.
Communicated by Liang-Gee Chen.
* This paper was partially supported by the National Science Council of Taiwan, R.O.C., under grant No. NSC 90-2219-E-260-001.

process of IDEA comprises eight rounds with the same structure and a final output transformation. Encryption and decryption both adopt the same process, but different subkeys. A single round adopts four modulo multipliers, of which three are on the critical path. Reducing the circuit complexity for modulo multiplier significantly improves the performance of the entire IDEA chip when implementing IDEA in hardware.

Many studies have been proposed to improve the performance of modulo multipliers [1-3, 6, 7, 11-15]. These works can be roughly divided into three categories: look-up table method, $(n + 1) \times (n + 1)$-bit multiplier method, and modulo carry-save addition multiplier method. Look-up table is the fastest and easiest way to implement modulo $(2^n + 1)$ multiplier, but it is also the most space consuming one. Although a method is proposed to downsize the table significantly to 2 Mbits by a number theoretic transformation [6], the memory space is still too large to be implemented in an IDEA chip. Due to the space problem, Curiger *et al*. first proposed a modulo multiplier that adopts $(n + 1) \times (n + 1)$-bit multiplier and modulo $(2^n + 1)$ adders [3]. Bahrami and Sadeghiyan also presented a similar method in [1]. An $(n + 1) \times (n + 1)$-bit multiplier is adopted because the modulo multiplier takes input value of zero as 2^n. However, while the input length it has to deal with increases, the gate count of multiplier increases. Therefore, the methods proposed in [7, 11-14] all try to reduce the input length it has to take care from $n + 1$ to n by lowering the original partial product matrix of size $(2n + 1) \times (n + 1)$ to $n \times n$. Due to high-speed requirement, the carry-save adder (CSA) with Wallace tree [4] and Booth algorithm [8, 9] are the most commonly adopted adder structure for summing up the partial product matrix in these works. The basic idea of modulo CSA is to push down the carry propagation for each bit position of the current level to the next. Conversely, Booth algorithm accelerates the summation by lowering the partial products from the beginning.

This study proposes an efficient circuit structure for modulo multiplication based on [13] and [14]. The improvements of the proposed modulo $2^n + 1$ multiplier consider three factors: the handling of zero input values, the procedure for summing up the partial product matrix with the constant value for getting correct result, and the final modulo addition used to sum up the sum and carry vectors generated from the carry-save adder. The part for taking care of zero input value is extracted from the partial product matrix. This can downsize the matrix and shorten the time delay on the critical path. The CSA with the Wallace tree structure [5] is adopted to sum up the partial product matrix and the constant without any extra stage of adder for decreasing the time delay on the critical path. Finally, an efficient modulo CLA structure is proposed based on the carry-look-ahead adder (CLA) in [14]. This modulo CLA adds the carry-vector and sum-vector produced by the previous modulo CSA and modulates the results by $(2^n + 1)$. In other words, no extra modulo adder stage is required. These improvements give the proposed modulo $(2^n + 1)$ multiplier the best time delay performance and the best product cost of area and time delay among all methods compared. The rest of this paper is organized as follows. Section 2 describes the proposed structure of modulo multiplier and modulo CLA. Section 3 then compares the proposed method with existing implementations. Section 4 shows the simulation results. Conclusions are finally drawn in section 5.

2. PROPOSED MODULO MULTIPLIER

Since the modulo multiplier takes zero as 2^n, the circuits can be divided into two

parts, those whose inputs have no zero value, and those with zero value inputs. The proposed improvements simply aim on these two parts. Some works propose designing a circuit that can handle both cases with the same circuit [12, 13]. However, this study recommends separating the zero-case handler from those handling non-zero values, since zero inputs cannot always occur, and extracting it outside the critical path improves the overall chip performance. For non-zero inputs, the circuit includes partial product matrix generation and summation circuit for partial product matrix. Figs. 1 and 2 show the major components of the proposed modulo (2^n + 1) multiplier for the case of non-zero and zero inputs respectively. The time delay and the area cost of each component are identified on the side. Each two-input monotonic cell counts as one gate area and delay, and a XOR cell counts as two gates area and delay for computing the time delay (T) and area cost (A). This is the same evaluation methods used in [1] and [11]. Any gate with more than two inputs is transformed to a multiple two-input monotonic gate in this paper. For example, a four-input AND is transformed into using three two-input AND gates.

2.1 Improvements on Non-zero Inputs Case

The first improvement we made was for the non-zero inputs case. This case is divided into two parts namely partial product matrix generation and summation circuit for partial product matrix. The design of partial product matrix generation is based on that proposed in [13]. It takes advantage of the concept of the redundancy in the binary representation of numbers in the finite integer ring, $R(2^n + 1)$. The partial product matrix we use is shown in Fig. 1 (a), where $P_{i,j}$ denotes the product of ith and jth positions of the two inputs, and equals to the result of a logical AND of the two bits. Comparing with the one proposed in [13], the part for taking care of input having zero value was extracted from the matrix. Because the zero case does not always occur, placing a zero-case handler on the critical path increases the time delay, and then degrades the performance of the entire IDEA chip. The constant that should be added on the result of CSAs to obtain correct result is summed in partial product matrix. However, no extra adder stage is included in the process. Therefore, the time delay on critical path is shortened.

The implementation on summing up the partial product matrix uses the carry-save adder (CSA) with Wallace tree structure [5], which performs the modulation while summing up the matrix by adding the inverted carry out to the least significant bit of the next stage as described in [13]. Since CSA pushes down the carry propagation to the next level, a final adder for summing up the carry-vector and sum-vector is generated from the partial product matrix summation.

Based on carry-look-ahead adder (CLA) [4], this study presents an efficient modulo CLA structure similar to that in [14]. This modulo CLA has two functions: adding carry-vector and sum-vector together and modulating the summation of them by (2^n + 1).

Let c_{i+1} (c_i) be the carry out (carry in) of a_i and b_i, which respectively denote the ith bit of the two inputs A and B. Let $g_i = a_i b_i$ and $p_i = a_i + b_i$, where g_i and p_i are traditionally called *generate* and *propagate*, and are the predictions of carry out generated and propagated at ith bit position, respectively. The following equation is obtained:

$$c_{i+1} = g_i + p_i c_i. \tag{1}$$

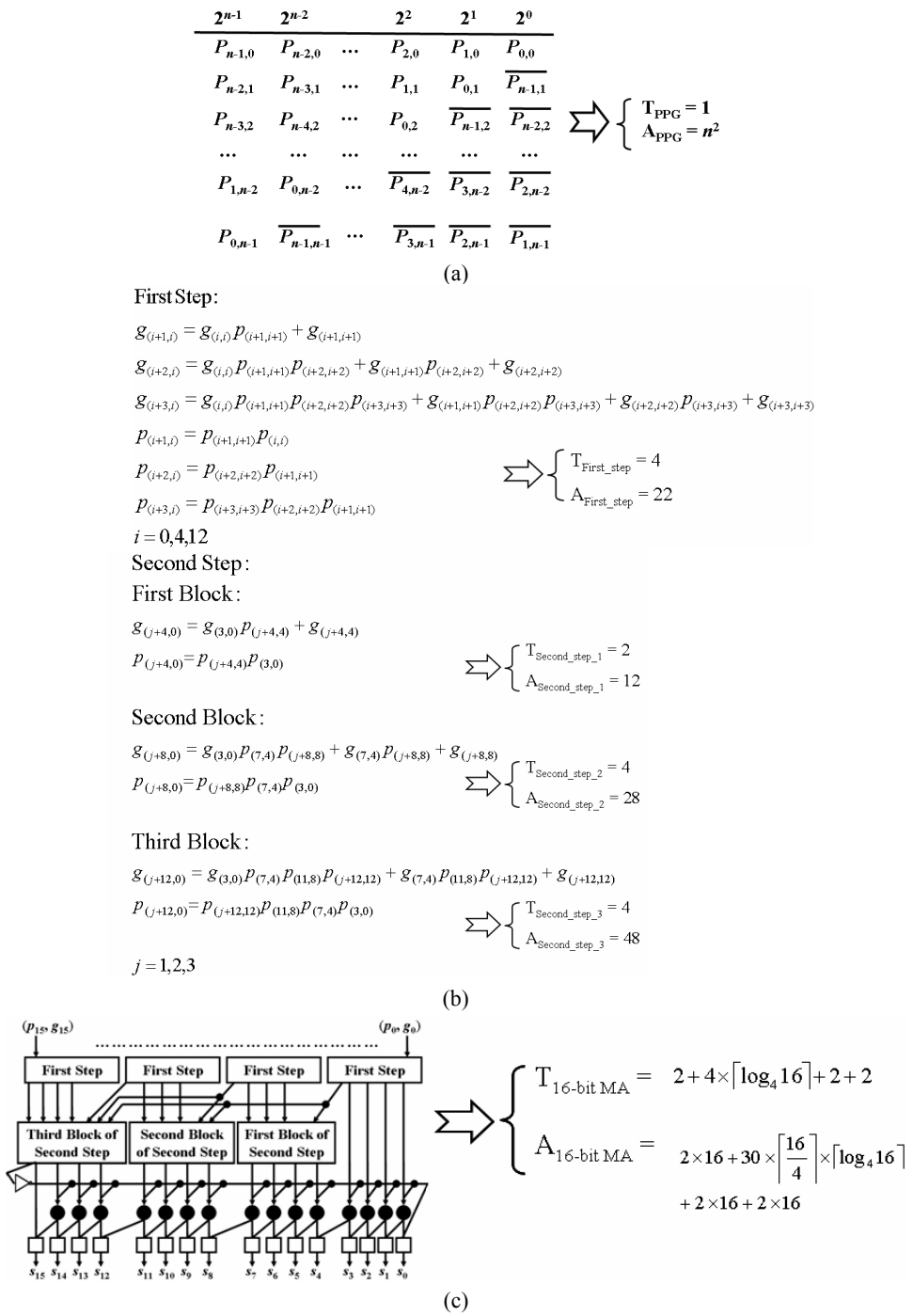

Fig. 1. Major components of the architecture of modulo $(2^n + 1)$ multiplier. (a) Partial product matrix. (b) Detail circuit of modulo $(2^n + 1)$ adder. (c) Modulo CLA structure for 16-bit input. In (c), the operation (•) in third level is for figuring out the carry in for each bit after modulating.

The complexity of calculation rises as the bit position increases when calculating the carry out of each bit position by the above equation. To reduce the complexity, an intermediate level of generate and propagate is usually incorporated, despite the resulting increase in time delay. Let k denote the base bit position and i denote the highest bit position of the intermediate group from kth bit to ith bit for calculating *group generate* and *group propagate*, the equations for calculating group generate $g_{(i,k)}$, group propagate $p_{(i,k)}$, and each bit's carry out c_{i+1} are given as follows:

$$g_{(i,k)} = g_{(i,j)}p_{(j,k)} + g_{(j,k)}, \text{ where } i \geq j \geq k \tag{2}$$

$$p_{(i,k)} = p_{(i,j)}p_{(j,k)}, \text{ where } i \geq j \geq k \tag{3}$$

$$c_{i+1} = g_{(i,k)} + p_{(j,k)}c_k, \text{ where } i \geq j \geq k. \tag{4}$$

Notably, $g_{(i,i)} = g_i$ and $p_{(i,i)} = p_i$. Eq. (2) (Eq. (3)) shows that a group generate (a group propagate) can be determined from two subgroup generates and one subgroup propagate (two subgroup propagates). Eq. (4) indicates that $c_{i+1} = g_{(i,0)} + p_{(j,0)}c_0$. That is, the carry out of any bit can be calculated by group generate, group propagate, and carry in of the CLA. Fig. 1 (b) shows an example of group generate and propagate generation of a 16-bit input length. Since the group generate and propagate are calculated for every 4-bit group, $\lceil \log_4 n \rceil$ levels of calculation would be required, where n is the input length. Here, the intermediate generate and propagation are calculated for every 4-bit, while the method proposed in [14] calculates group generate and propagate for every 2-bit.

The modulation can be performed by inverting the carry out of most significant bit and adding the resultant to the least significant bit as adopted in modulo CSA. Therefore, only a modification for the carry in on each bit position obtained from original CLA is needed. Each bit's carry in after modulation can be generated from the information of the inverted carry out of the most significant bit. The detail circuit of each modulo adder component and the structure of modulo adder for a 16-bit input are shown in Figs. 1 (b) and (c), respectively. Since the generate and propagate are calculated for every 4-bit group, two steps are required to calculate the final carry-in for each bit as shown in Fig. 1 (c). In Figs. 1 (b) and (c), g_i and p_i ($g_{(i,k)}$ and $p_{(i,k)}$) respectively denote the generate and the propagate (the group generate and the group propagate) of each bit position as defined before, and s_i is the summation result of ith position. The output after the second level is the carry in for each bit without modulation. To obtain the modulated result, the carry in after modulation for each bit position must be calculated. This can be done by inverting the carry out of most significant bit from the second level and by feeding it into ($\bullet$) operation, which is defined according to Eq. (4). The carry in after modulation for each bit can then be figured out. The summation after modulation can be obtained by XORing the actual carry in and propagate of each bit position ($\square$ operation).

2.2 Improvement on Zero-case Handler

Another improvement is to deal with inputs with zero values outside the partial product matrix, *i.e.*, outside the critical path of the modulo multiplier. The modulo (2^n + 1) multiplication of inputs with zero values can be calculated as follows:

$$(2^n \times a) \bmod (2^n + 1) = ((-1) \times a) \bmod (2^n + 1) = 2^n + 1 - a$$
$$= \bar{a} + 2, \text{ where } a \in \{0, 1, \ldots, 2^n + 1\}. \quad (5)$$

The implementation of the zero-case handler is similar to that of the modulo adder for non-zero case. Fig. 2 shows the equations of the special adder for 16-bit input that realizes Eq. (5), and Fig. 3 shows the entire structure of zero-case handler. Since one of the inputs of Eq. (5) is the constant 2, the circuit of the adder for zero-case handler can be significantly reduced. Eqs. (2), (3), and (4) clearly indicate that $c_{i+1} = p_{(i,0)}$, for $i \geq 3$. Notably, the special adder does not require a modulation, and also requires no (•) operations that is required in Fig. 1 (c).

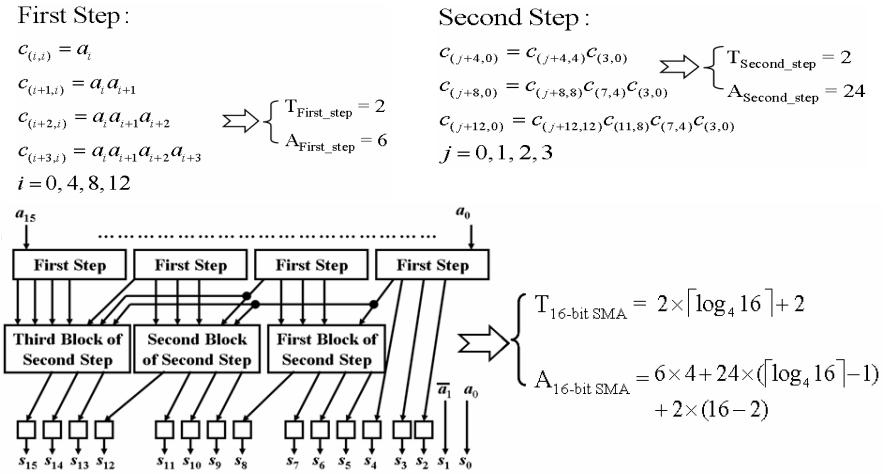

Fig. 2. The special adder for zero-case handler with a 16-bit input. This handler does not require modulation and CLA is designed to predict carry out of each bit position in advance according to inputs.

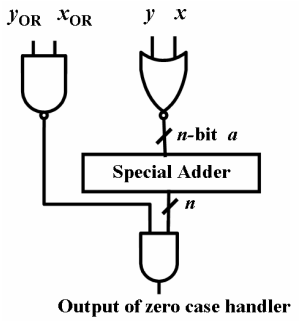

Fig. 3. The structure of the zero-case handler with n OR gates with NOT gates before the inputs of the special adder and n AND gates after the outputs of the special adder, where n denotes the input length.

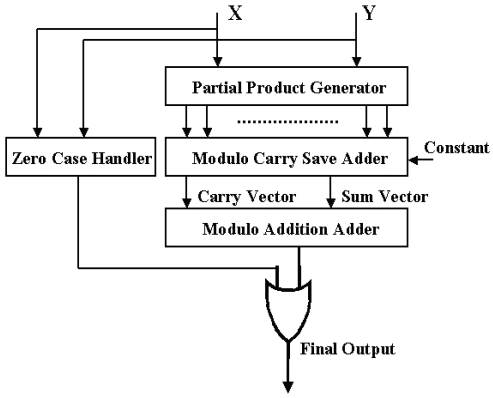

Fig. 4. The complete structure of modulo (2^n + 1) multiplier. The two inputs enter the zero-case handler and conventional multiplier for non-zero value concurrently, and n OR gates are used in the end to select the desired output.

Fig. 3 shows the structure of zero case handler. The two inputs are first bit-wise NORed such that any input other than zero is inverted as the input for special adder. X_{OR} and Y_{OR} in Fig. 3 denote the results of ORing all the bits of the two inputs. The output of NAND gate with X_{OR} and Y_{OR} as its inputs would be 1, and the output of special adder would be passed, when one of the inputs is zero. Consequently, the AND gates at the end acts as a multiplexer for selecting the output of the zero case handler, and the NAND gate on the left side is adopted to select the desired result for the output.

Fig. 4 shows the complete structure of the proposed modulo (2^n + 1) multiplier. The circuit on the right is the modulo multiplier for non-zero inputs, and the circuit on the left is the zero-case handler. The two inputs X and Y are executed concurrently by the two circuits. If none of the inputs is zero, then every output of the zero-case handler is zero. Additionally, the output of modulo multiplier is all zero if one or more inputs is zero. This property means that the multiplexer can be simplified to using n OR gates.

3. COMPARISON

The time delay and area cost of the circuits proposed in [7, 11-14], which all use modulo carry save adders with Wallace tree structure for their partial product matrix summations, were first formulated. Table 1 shows the stage number of Wallace tree under n-bit length input. Notably, $d(n)$ denotes the stage number of the Wallace tree structure with n-bit input length and end-around-carry, and $d'(n)$ denotes the stage number without end-around carry. Table 2 shows the time delays of the critical path of each method. The time delay T of the proposed architecture comprises the delay from partial product matrix (T_{PPM}), carry-save adders (T_{CSA}), modulo adder (T_{MA}), and final selection (T_{SEL}). Additionally, the time delay of the other four circuits all comprise the same components of the critical path of the proposed circuits in addition to zero case handler (T_{ZERO}). Table 3 shows the area cost of each method. The term n_i denotes the number of full adders in the Wallace tree of ith stage, and $n_0 = n$, where n denotes the input length,

Table 1. Stage number of Wallace tree $d(n)$ (under modulo CSA) with different input lengths.

Input Length	1-3	4	5-6	7-9	10-13	14-19	20-28	29-42	43-63	64
$d(n)$	1	2	3	4	5	6	7	8	9	10

Table 2. Formulated time delay equation for each method with n-bit input length.

	Components	Time Delay	$n = 16$
Our Method	$T_{PPM} + T_{CSA} + T_{MA} + T_{SEL}$	$\{1\} + \{4 \times d(n+1)\} + \{2 + 4 \times \lceil \log_4 n \rceil + 2 + 2\} + \{1\}$	40
Zimmermann	$T_{PPM} + T_{CSA} + T_{COR} + T_{MA}$	$\{5\} + \{4 \times d(\lfloor n/2 \rfloor)\} + \{4 + 4 + 1\} + \{2 \times \lceil \log_2 n \rceil + 6\}$	44
Wrzyszcz *et al.*	$T_{PPM} + T_{CSA} + T_{COR} + T_{MA} + T_{SEL}$	$\{4\} + \{4 \times d(n) + 4(n-1) + 2)\} + \{2 \times n\} + \{2 \times n\} + \{2\}$	156
Wang *et al.*	$T_{PPM} + T_{CSA} + T_{MA}$	$\{2 \times ((n-1) + 2) + 4 \times d'(n-1)\} + \{4 \times d(\lfloor n/2 \rfloor)\} + \{2 \times \lceil \log_2 n \rceil + 6\}$	86
Ma *et al.*	$T_{PPM} + T_{CSA} + T_{COR} + T_{MA}$	$\{6\} + \{4 \times d(\lfloor n/2 \rfloor)\} + \{4 + 4\} + \{2 \times \lceil \log_2 n \rceil + 6\}$	44
Bahrami *et al.*	$T_{PPM} + T_{CSA} + T_{SEL} + T_{MA}$	$\{2 \times ((n-1) + 2) + 4 \times d'(n-1)\} + \{4 \times d(n+1)\} + \{2\} + \{2 \times \lceil \log_2 n \rceil + 6\}$	92
Sousa	$T_{PPM} + T_{CSA} + T_{COR} + T_{MA}$	$\{6\} + \{4 \times d(\lfloor n/2 \rfloor)\} + \{4\} + \{2 \times \lceil \log_2 n \rceil + 6\}$	40

and $n_i = n_{i-1} - 3\lfloor n_{i-1}/3 \rfloor + 2\lfloor n_{i-1}/3 \rfloor$. $C'(n)$ denotes the number of full adders of a Wallace tree structure without end-around carry [1, 12]. The area costs of all circuits comprise partial product matrix (A_{PPM}), carry-save adders (A_{CSA}), modulo adders (A_{MA}) and circuits for zero case (A_{ZERO}). A final selection (A_{SEL}) unit is required for the proposed architecture, while correction (A_{COR}) units are required in the other schemes. Both Tables 2 and 3 have a column for the delay and cost with 16-bit long input.

The modules on the critical path in the proposed architecture are the partial product matrix, carry save adder, modulo adder, and a final selection to specify the final output from zero case handler and modulo adder. The time delay of the partial product matrix comprises a single gate. Compared with the partial product matrix in [13], the time delay of the proposed partial product matrix is reduced from 4 to 1 because the zero-case handling is extracted from the critical path. Since the constant is added, the CSA needs $d(n + 1)$ layers of full adder, where each full adder contributes a delay of four gates. The time delay of modulo adder is as described in Fig. 1 (c). Compared with the modulo adder proposed by Zimmermann [14], the number of stages for calculating group generate and propagate is reduced from $\lceil \log_2 n \rceil$ to $\lceil \log_4 n \rceil$ with a small increase in the time delay of a single stage. The final selection is an OR gate for each bit position and therefore contributes a delay of only one gate.

The time delay of the architecture proposed by Zimmermann, Sousa, and Ma [7, 11, 14] are formulated according to [11]. The time delay of the one proposed by Wrzyszcz *et al.* [13] is similar in the partial product matrix and CSA. The correction and modulo

Table 3. Formulated area cost equation for each method with *n*-bit input length.

	Components	Area Cost	n = 16
Our Method	$A_{PPM} + A_{CSA} + A_{MA} + A_{SEL} + A_{ZERO}$	$\{n^2\} + \{7n \times \sum_{i=1}^{d(n+1)} \lfloor \frac{n_{i-1}}{3} \rfloor\} + \{2n + 30 \times \lceil \frac{n}{4} \rceil \times \lceil \log_4 n \rceil + 2n + 2n\} + \{n\} + \{8 \times \lceil \frac{n}{4} \rceil \times \lceil \log_4 n \rceil + 2 \times (n-2) + 2 \times \sum_{i=1}^{\lceil \log_2 n \rceil} \frac{n}{2^i} + n + 1\}$	2467
Zimmermann	$A_{PPM} + A_{CSA} + A_{COR} + A_{MA}$	$\{9 \times (\frac{n}{2} + 1) \times n\} + \{7n \times \sum_{i=1}^{d(n/2)} \lfloor \frac{n_{i-1}}{3} \rfloor\} + \{45 \times n\} + \{\frac{3}{2} \times n \times \lceil \log_2 n \rceil + 8 \times n\}$	3808
Wrzyszcz *et al.*	$A_{PPM} + A_{CSA} + A_{COR} + A_{MAR} + A_{SEL}$	$\{n^2 + 3n\} + \{7n \times \sum_{i=1}^{d(n)} \lfloor \frac{n_{i-1}}{3} \rfloor + (7 \times (n-1) + 3)\} + \{3n\} + \{3 \times n\} + \{3 \times n\}$	2124
Wang *et al.*	$A_{PPM} + A_{CSA} + A_{MA}$	$\{2 \times (3 \times (n-1) + 7) + 7 \times C'(n-1)\} + \{7n \times \sum_{i=1}^{d'(n+1)} \lfloor \frac{n_{i-1}}{3} \rfloor\} + \{\frac{3}{2} n \lceil \log_2 n \rceil + 8n\}$	1987
Ma *et al.*	$A_{PPM} + A_{CSA} + A_{COR} + A_{MA}$	$\{9 \times \frac{n^2}{2}\} + \{7n \times \sum_{i=1}^{d(n/2)} \lfloor \frac{n_{i-1}}{3} \rfloor\} + \{45n\} + \{\frac{3}{2} n \lceil \log_2 n \rceil + 2 \times 7n\}$	3760
Bahrami *et al.*	$A_{PPM} + A_{CSA} + A_{MA} + A_{SEL} + A_{ZERO}$	$\{2 \times (3 \times (n-1) + 7) + 7 \times C'(n-1)\} + \{7n \times \sum_{i=1}^{d'(n+1)} \lfloor \frac{n_{i-1}}{3} \rfloor\} + \{\frac{3}{2} n \lceil \log_2 n \rceil + 8n\} + \{2 \times 3 \times n\} + \{2 \times 3 \times n\}$	2179
Sousa	$A_{PPM} + A_{CSA} + A_{COR} + A_{MA}$	$\{9 \times (\frac{n}{2} + 1) \times n\} + \{7n \times \sum_{i=1}^{d(/21)} \lfloor \frac{n_{i-1}}{3} \rfloor\} + \{(21 + 7) \times n\} + \{\frac{3}{2} n \lceil \log_2 n \rceil + 8n\}$	3536

adder circuit proposed in [13] comprise a special ripple adder, where each adder comprises the structure of half adder. The selection unit in [13] is a 2-to-1 multiplexer for each bit position on the critical path. The architectures proposed in [1] and [12] are similar. The CSA and modulo adder of [1] and [12] are the same as those in [7, 11, 14]. Significantly, the partial product matrices in [1] and [12] need to convert the input into diminished-1 representation and thus need a subtractor for the matrix. Finally, the selection unit of [1] comprises a 2-to-1 multiplexer.

The area cost of the proposed multiplier is contributed by five major components: the partial product matrix, CSA, modulo adder, zero case handler, and final selection unit. The area cost of partial product matrix and modulo adder is described in Figs. 1 (b) and (c). The modulo adder needs $\lceil \log_4 n \rceil$ layers of operation, each needing $\lceil n/4 \rceil$ operations. Thus, the modulo adder needs $\lceil n/4 \rceil \times \lceil \log_4 n \rceil$ operations, each costing (12 + 28 + 48)/3 ≈ 30 in area cost. The CSA is constructed from a Wallace tree structure and each bit position would need $\sum_{i=1}^{d(n+1)} \lfloor n_{i-1}/3 \rfloor$ full adders, which contributes 7 in area cost for each adder. The zero case handler comprises the special adder, n NOR gates for input selection, two n-bit OR gates to ORed all the bits of the two inputs, a NAND for selection line production, and n AND gates for the final output selection. Fig. 2 shows the area cost of

the special adder, which needs $\lceil n/4 \rceil \times \lceil \log_4 n \rceil$ operations, each costing 24/3 = 8 in area cost.

As mentioned earlier, the area cost of the architecture proposed by Ma, Sousa, and Zimmermann [7, 11, 14] is formulated according to [11]. The circuit proposed by Wrzyszcz [13] has the same partial product matrix and CSA architecture as the circuits presented herein. The modulo adder and correction comprise two n-bit ripple half adders, where each half adder contributes an area cost of 3. The selection unit of [13] comprises n 2-to-1 multiplexers, each contributing 3 to the area cost. The area cost of the architecture proposed by Bahrami [1] and Wang [12] are almost the same as the architecture just described, but the one in [1] has zero case handler. The partial product generation in [1] and [12] is constructed by an n-bit substractor as described in time delay. The CSA and modulo adder are the same as those in other proposed architectures. The zero case handler and selection unit in [1] are both constructed from n 2-to-1 multiplexers.

Fig. 5 compares the time delay and area cost of different modulo multiplier architecture with input lengths ranging from 1 bit to 32 bits. The two charts in Fig. 5 are estimated according to the equations shown in Tables 2 and 3. Fig. 5 indicates that the proposed scheme and Sausa's scheme have the best time delays. Although Sausa's scheme has a small time delay, it has a much larger area cost because it has a large circuit for partial product reduction. Fig. 6 shows the relationship between the area-delay product and the input length and indicate that the time delay of the proposed modulo multiplier is smallest in all input lengths, while the area cost is slightly higher than those of Wrizyszcz's and Wang's. Fig. 6 also indicates that the proposed multiplier has the lowest area-delay product for all input lengths.

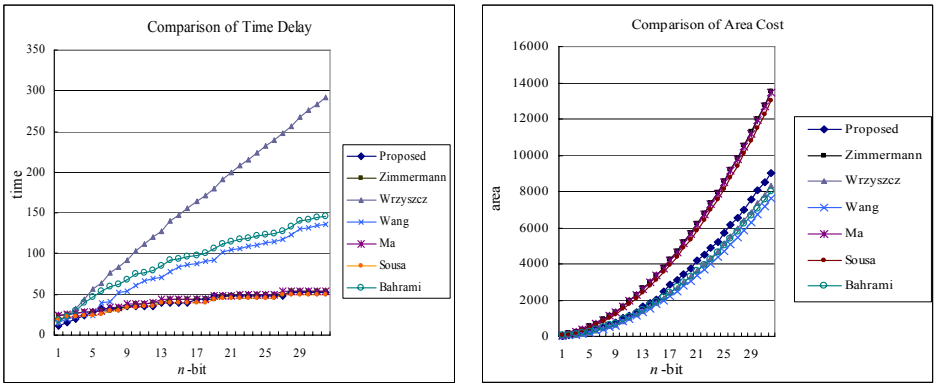

Fig. 5. The comparison of time delay and area cost within input lengths of 1 bit to 32 bits.

4. SIMULATION RESULT

Validity of the proposed design was verified to use CPLD to design an IDEA chip with the proposed modulo multiplier structure. The CPLD chip used for simulation was an Altera EPF10K200SFC672-1 of FLEX10K series with Max+PlusII as the design software. This chip was used because the authors intend to design an IDEA chip with a PCI

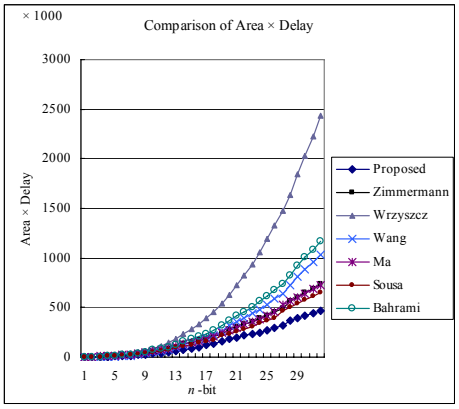

Fig. 6. The comparison of area-delay product of the proposed circuits with the others.

interface in the near future, and only the PCI development kit produced by Altera has this CPLD in its package.

A simulation of our IDEA chip with the proposed new modulo multiplier design was performed. The timing simulation reveals that the proposed structure has a time delay of 59.5ns. An IDEA chip with single round of circuit was implemented, and each round was partitioned into four pipeline stages. To complete encryption/decryption process, the circuit was reused for 8.5 times. Although taking more time to process data, this implementation has a lower area cost than fully implementing 8.5 rounds of circuits. The clock rate of the proposed modulo multiplier can be up to 8.25 MHz, and the encryption/decryption rate is approximately 66Mb/sec.

5. CONCLUSION

This study presents an efficient modulo multiplier structure with improvements in the partial product matrix, modulo adder for modulo CSA, and zero-case handler. Comparing this structure with existing methods demonstrates that this structure not only has the best time delay performance but also the best product cost of area and time delay among the seven methods compared. A simulation on implementing IDEA chip with the proposed modulo multiplier structure was also developed on CPLD. Simulation results indicate that the IDEA chip with one round implemented and cut into four pipeline stages can achieve frequency of 8.25 MHz, and the encryption (decryption) rate is approximately 66Mb/sec. The clock rate of the simulation structure can be further improved in the near future. Additionally, the practical implementation of the proposed method will be performed in the near future.

REFERENCES

1. M. Bahrami and B. Sadeghiyan, "Efficient modulo 2^n + 1 multiplication schemes for IDEA," in *Proceedings of IEEE International Symposiums on Circuits and Systems*, 2000, pp. 653-656.

2. A. Curiger, H. Bonnenberg, R. Zimmermann, N. Felber, H. Kaeslin, and W. Fichtner, "VINCI: VLSI implementation of the new secret-key block cipher IDEA," in *Proceedings of IEEE Custom Integrated Circuits Conference*, 1993, pp. 15.5.1-15.5.4.
3. A. V. Curiger, H. Bonnenberg, and H. Kaeslin, "Regular VLSI architecture for multiplication modulo (2^n + 1)," *IEEE Journal of Solid-State Circuits*, Vol. 26, 1991, pp. 990-994.
4. D. D. Gajski, *Principles of Digital Design*, Prentice Hall, 1997.
5. J. L. Hennessy and D. A. Patterson, *Computer Architecture – A Quantitative Approach*, 2nd ed., Morgan Kaufmann, 1996.
6. F. A. Jullien, "Implementation of multiplication, modulo a prime number, with applications to number theoretic transforms," *IEEE Transactions on Computers*, Vol. C-29, 1980, pp. 899-905.
7. Y. Ma, "A simplified architecture for modulo 2^n + 1 multiplication," *IEEE Transactions on Computers*, Vol. 47, 1998, pp. 333-337.
8. P. E. Madrid, B. Millar, and E. E. Swartzlander Jr., "Modified booth algorithm for high radix multiplication," in *Proceedings of IEEE Computer Design: VLSI in Computer and Processors*, 1992, pp. 118-121.
9. D. A. Patterson and J. L. Hennessy, *Computer Organization & Design*, *The Hardware/Software Interface*, 2nd ed., Morgan Kaufmann, 1998.
10. W. Stallings, *Cryptography and Network Security – Principles and Practice*, 2nd ed., Prentice Hall, 1999.
11. L. Sousa, "A universal architecture for designing efficient modulo 2^n + 1 multipliers," *IEEE Transactions on Circuits and System*s, Vol. 52, 2005, pp. 1166-1178.
12. Z. Wang, G. A. Jullien, and W. C. Miller, "An algorithm for multiplication modulo (2^n + 1)," in *Proceedings of IEEE ASILMAR-29*, 1995, pp. 956-960.
13. A. Wrzyszcz and D. Milford, "A new modulo 2^a + 1 multiplier," in *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 1993, pp. 614-617.
14. R. Zimmermann, "Efficient VLSI implementation of modulo ($2^n \pm 1$) addition and multiplication," in *Proceedings of the 14th IEEE Symposium on Computer Architecture*, 1999, pp. 158-167.
15. R. Zimmermann, A. Curiger, H. Bonnenberg, H. Kaeslin, N. Felber, and W. Fichtner, "A 177mb/s VLSI implementation of the international data encryption algorithm," *IEEE Journal of Solid-State Circuits*, Vol. 29, 1994, pp. 303-307.

Yi-Jung Chen (陳依蓉) received her B.S. and M.S. degrees in Computer Science and Information Engineering from National Chi Nan University, Nantou, Taiwan, in 2000 and 2002, respectively. She is currently working towards the Ph.D. at the Department of Computer Science and Information Engineering at National Taiwan University. Her research interests are in the area of computer architecture and high-level synthesis in memory hierarchy.

Dyi-Rong Duh (杜迪榕) received the B.S. degree in Electronics Engineering from the National Taiwan Institute of Technology, the M.S. degree in Computer Engineering

from the Tamkang University, and the Ph.D. degree in Computer Science and Information Engineering from the National Taiwan University, in 1983, 1988, and 1994, respectively. He worked for the Sysgration Ltd Taiwan as the R&D manager of the computer peripherals division from June 1988 to July 1989. He was a teaching assistant from August 1989 to July 1990 in the Department of Computer Science and Information Engineering at the National Taiwan University. From August 1990 to July 1994, he was a lecturer in the Department of Electronics Engineering at Hwa Hsia Institute of Technology, Taipei Hsien, Taiwan. He is currently an associate professor in the Department of Computer Science and Information Engineering at the National Chi Nan University, Nantou Hsien, Taiwan. His research interests include graph-theoretic interconnection networks, graph theory, embedded systems, and computer architectures.

Yunghsiang S. Han (韓永祥) was born in Taipei, Taiwan, on April 24, 1962. He received the B.S. and M.S. degrees in Electrical Engineering from the National Tsing Hua University, Hsinchu, Taiwan, in 1984 and 1986, respectively, and the Ph.D. degree from the School of Computer and Information Science, Syracuse University, Syracuse, NY, in 1993. From 1986 to 1988 he was a lecturer at Ming-Hsin Engineering College, Hsinchu, Taiwan. He was a teaching assistant from 1989 to 1992 and from 1992 to 1993 a research assistant in the School of Computer and Information Science, Syracuse University. He is a winner of 1994 Syracuse University Doctoral Prize. From 1993 to 1997 he was an Associate Professor in the Department of Electronic Engineering at Hua Fan College of Humanities and Technology, Taipei Hsien, Taiwan. From 1997 to 2004 he was with the Department of Computer Science and Information Engineering at National Chi Nan University, Nantou, Taiwan. He was promoted to Full Professor in 1998. From June to October 2001 he was a visiting scholar in the Department of Electrical Engineering at University of Hawaii at Manoa, HI, and from September 2002 to January 2004 he was the SUPRIA visiting research scholar in the Department of Electrical Engineering & Computer Science and CASE center at Syracuse University, NY. He is now with the Graduate Institute of Communication Engineering at National Taipei University, Taipei, Taiwan. His research interests are in wireless networks, security, and error-control coding.