An improved and fast design of IDEA encryption on FPGA

Sapna tripathi[1], Mr. Ravimaohan[2]
[1]M. tech. Scholar. SRIT, Jabalpur,
[2]HOD & Associate Professor, SRIT, Jabalpur

Abstract: International Data Encryption Algorithm (IDEA) is a block cipher designed by James Massey of ETH Zurich and Xuejia Lai and was first described in 1991. As a block cipher, it is also symmetric. The algorithm was intended as a replacement for the Data Encryption Standard (DES). IDEA is a minor revision of an earlier cipher, Proposed Encryption Standard (PES); IDEA was originally called Improved PES (IPES).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 paper presents an efficient hardware structure for the modulo $(2n + 1)$ 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. The proposed design enables IDEA to be implemented on hardware with high performance and low cost. Simulation results obtained from Vertex FPGA system developed by Xillinx indicate that the new design has 37.57 MHz maximum speed for encryption and decryption with eight pipeline stages for each round.

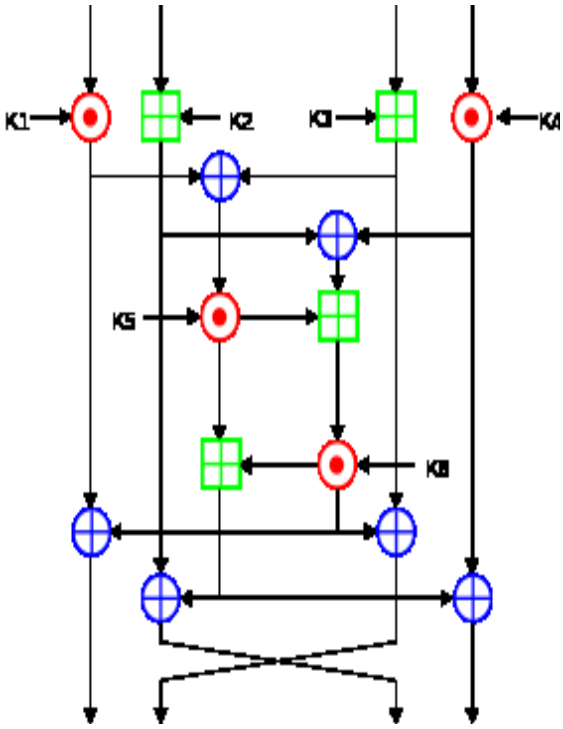
1. INTRODUCTION

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 $2n$. Modulo multiplication multiplies two inputs of *n*-bit length, and mods the result by $(2n + 1)$. Notably, an input value of zero is considered as $2n$. Therefore, the input length of modulo multiplication is $n + 1$ when the input value is zero. The encryption/decryption 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.



IDEA operates on 64-bit blocks using a 128-bit key, and consists of a series of eight identical transformations (a *round*, see the illustration) and an output transformation (the *half-round*). The processes for encryption and decryption are similar. IDEA derives much of its security by interleaving operations from different groups — modular addition and multiplication, and bitwise eXclusive OR (XOR) — which are algebraically "incompatible" in some sense. In more

ISSN: 2278 – 7798

detail, these operators, which all deal with 16-bit quantities, are:

- Bitwise eXclusive OR (denoted with a blue circled plus ⊕).
- Addition modulo 2^{16} (denoted with a green boxed plus ⊞).
- Multiplication modulo $2^{16}+1$, where the all-zero word (0x0000) is interpreted as 2^{16} (denoted by a red circled dot ⊙).

2. PROPOSED MODULO MULTIPLIER

Since the modulo multiplier takes zero as $2n$, 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. Figures Shown below explains 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.

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. below, where $P_{i,j}$ denotes the product of i^{th} and j^{th} 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 carryvector and sum-vector together and modulating the summation of them by $(2^n + 1)$.

ISSN: 2278 – 7798

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

$$c_{i+1} = g_i + p_i c_i. \qquad \text{Eq(1)}$$

2^{n-1}	2^{n-2}		2^2	2^1	2^0
$P_{n-1,0}$	$P_{n-2,0}$	...	$P_{2,0}$	$P_{1,0}$	$P_{0,0}$
$P_{n-2,1}$	$P_{n-3,1}$	...	$P_{1,1}$	$P_{0,1}$	$\overline{P_{n-1,1}}$
$P_{n-3,2}$	$P_{n-4,2}$	...	$P_{0,2}$	$\overline{P_{n-1,2}}$	$\overline{P_{n-2,2}}$
...	...	...	...	...	...
$P_{1,n-2}$	$P_{0,n-2}$	...	$\overline{P_{4,n-2}}$	$\overline{P_{3,n-2}}$	$\overline{P_{2,n-2}}$
$P_{0,n-1}$	$\overline{P_{n-1,n-1}}$	...	$\overline{P_{3,n-1}}$	$\overline{P_{2,n-1}}$	$\overline{P_{1,n-1}}$

$$\Rightarrow \begin{cases} T_{PPG} = 1 \\ A_{PPG} = n^2 \end{cases}$$

First Step:

$$g_{(i+1,i)} = g_{(i,i)} p_{(i+1,i+1)} + g_{(i+1,i+1)}$$

$$g_{(i+2,i)} = g_{(i,i)} p_{(i+1,i+1)} p_{(i+2,i+2)} + g_{(i+1,i+1)} p_{(i+2,i+2)} + g_{(i+2,i+2)}$$

$$g_{(i+3,i)} = g_{(i,i)} p_{(i+1,i+1)} p_{(i+2,i+2)} p_{(i+3,i+3)} + g_{(i+1,i+1)} p_{(i+2,i+2)} p_{(i+3,i+3)} + g_{(i+2,i+2)} p_{(i+3,i+3)} + g_{(i+3,i+3)}$$

$$p_{(i+1,i)} = p_{(i+1,i+1)} p_{(i,i)}$$

$$p_{(i+2,i)} = p_{(i+2,i+2)} p_{(i+1,i+1)}$$

$$p_{(i+3,i)} = p_{(i+3,i+3)} p_{(i+2,i+2)} p_{(i+1,i+1)}$$

$$\Rightarrow \begin{cases} T_{First\_step} = 4 \\ A_{First\_step} = 22 \end{cases}$$

$$i = 0,4,12$$

Second Step:

First Block:

$$g_{(j+4,0)} = g_{(3,0)} p_{(j+4,4)} + g_{(j+4,4)}$$

$$p_{(j+4,0)} = p_{(j+4,4)} p_{(3,0)}$$

$$\Rightarrow \begin{cases} T_{Second\_step\_1} = 2 \\ A_{Second\_step\_1} = 12 \end{cases}$$

Second Block:

$$g_{(j+8,0)} = g_{(3,0)} p_{(7,4)} p_{(j+8,8)} + g_{(7,4)} p_{(j+8,8)} + g_{(j+8,8)}$$

$$p_{(j+8,0)} = p_{(j+8,8)} p_{(7,4)} p_{(3,0)}$$

$$\Rightarrow \begin{cases} T_{Second\_step\_2} = 4 \\ A_{Second\_step\_2} = 28 \end{cases}$$

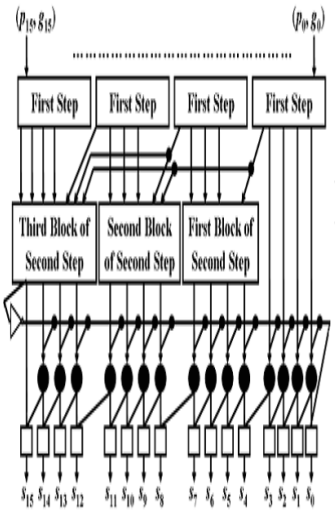
Third Block:

$$g_{(j+12,0)} = g_{(3,0)} p_{(7,4)} p_{(11,8)} p_{(j+12,12)} + g_{(7,4)} p_{(11,8)} p_{(j+12,12)} + g_{(j+12,12)}$$

$$p_{(j+12,0)} = p_{(j+12,12)} p_{(11,8)} p_{(7,4)} p_{(3,0)}$$

$$\Rightarrow \begin{cases} T_{Second\_step\_3} = 4 \\ A_{Second\_step\_3} = 48 \end{cases}$$

$$j = 1,2,3$$



$$\Rightarrow \begin{cases} T_{16\text{-bit MA}} = 2+4\times\lceil\log_4 16\rceil+2+2 \\ A_{16\text{-bit MA}} = 2\times16+30\times\left\lceil\frac{16}{4}\right\rceil\times\lceil\log_4 16\rceil \\ \qquad\qquad +2\times16+2\times16 \end{cases}$$

J

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 k^{th} bit to i^{th} 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 $ci+1$ are given as follows:

ISSN: 2278 – 7798

$g_{(i,k)} = g_{(i,j)}p_{(j,k)} + g_{(j,k)}$, where $i \geq j \geq k$ *Eq*. (2)

$p_{(i,k)} = p_{(i,j)}p_{(j,k)}$, where $i \geq j \geq k$ *Eq*(3)

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

Notably, $g_{(i,i)} = g_i$ and $p_{(i,i)} = pi$. 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. above 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, $[\log_4 n]$ 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. above, 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. above. In Figs. above, *gi* 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 *si* is the summation result of *i*th 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 (•) 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 (□ operation).

3. Results

Platform Used: **Vertex4 XC4VLX80-12FF1148**			
Tool Used: **Xilinx 12.2**			
	Used	**Available**	**Utilization**
Number of Slices	**405**	**35840**	**1%**
Number of 4 input LUTs	**707**	**71680**	**0%**
Number of bonded IOBs	**48**	**368**	**6%**

The table shown above is result of proposed design. the propagation delay time observed is 26.611 ns and hence we could say our max speed for the design is 37.57 Mhz. Our Area utilisation is 405 slices. Our results are 16% more area efficient and 200% more faster as compare to previous Modulo multiplier design of ref[1].

4. 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

ISSN: 2278 – 7798

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 Xilinx EDA. Simulation results indicate that the IDEA chip with one round implemented and cut into eight pipeline stages can achieve frequency of 37.57 MHz, and the no, of slice in Vertex4 FPGA are 405. 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 $2n + 1$ 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 $(2n + 1)$," *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 $2n + 1$ 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 $2n + 1$ 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 $(2n + 1)$," in *Proceedings of IEEE ASILMAR-29*, 1995, pp. 956-960.
13. A. Wrzyszcz and D. Milford, "A new modulo $2a + 1$ 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 $(2n \pm 1)$ 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.

ISSN: 2278 – 7798