Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 51(65), Fascicola 2, 2006

An IP design of the idea cryptographic algorithm

M. A. Ajo[1], G. Fericean[2], M. Borda[2] and V. Rodellar[1]

Abstract – In this paper we introduce a library component implementation of the IDEA cryptographic algorithm that may be used embedded in security applications. The model allows scalability in the number of bits of the plaintext and ciphertext and in the number of keys. The hardware design has been modeled in VHDL portable code resulting in a technology independent soft-core.

Keywords: Reusability, IP core, IDEA algorithm, Cryptography.

I. IDEA ALGORITHM

The IDEA (International Data Encryption Algorithm) block cipher is a symmetric-key algorithm, which encrypts 64-bits plaintext blocks to 64-bit cipher text blocks using a 128-bit key K. The same algorithm is used for both encryption and decryption as it is a symmetric-key encryption system [1], [2], [3], [4]. IDEA has been patented in the U.S. and in several European countries, but the non-commercial use of IDEA is free everywhere. The cryptographic strength of IDEA is summarized by the following characteristics:

- Block length: The block length should be long enough to avoid preferences in the block appearance. The use of a block size of 64 bits is recognized as sufficiently strong.
- Key length: The key length should prevent exhaustive key searches. IDEA uses 128 bits.
- Confusion: The ciphertext should depend on the plaintext and key in a complicated and involved way. The objective is to complicate the determination of how the statistics of the ciphertext depend on the statistics of the plaintext. This goal is obtained by applying the operations of exclusive-OR, addition of integers modulus 2^{16} (65536) and multiplication of integers modulus $2^{16} + 1$ (65537) over two inputs of 16 bits. These three operations are incompatible in the sense that no pair of these three operations satisfies a distributive or an associative law.
- Diffusion: Each plaintext bit and each key bit should influence every ciphertext bit. The spreading out of a single plaintext bit over many ciphertext bits hides the statistical structure of the plaintext. The diffusion is provided by the basic building block of the algorithm denoted as MA (multiplication and addition). This block takes as inputs two 16-bit values derived from the plaintext and two 16-bit subkeys derived from the key and produces two 16-bit outputs. This particular structure is repeated eight times in the algorithm, providing very effective diffusion.

II. COMPUTATIONAL STRUCTURE

IDEA consists of 8 computationally identical rounds followed by a final transformation, as can be seen in Fig. 1. The 64-bit data block is divided into four 16-bit sub-blocks: X1, X2, X3 and X4. These four sub-blocks become the input to the first round of the algorithm. In each round the four sub-blocks are XOR-ed, added and multiplied with each other and with six 16-bit sub-keys ($K_1^{(1)}$… $K_6^{(8)}$). Between rounds, the second and third sub-blocks are swapped. Finally, the four sub-blocks are combined with four sub-keys ($K_1^{(9)}$… $K_4^{(9)}$) in a final transformation block.
In the next sub-sections the structure of a single round and final transformation stages will be described. And also sub-keys generation from the main key will be presented. The structure is described in terms of the basic operations involved in the algorithm and mentioned in the confusion characteristic, that is:

- $\oplus$ Exclusive OR
- $\boxplus$ Modulus addition
- $\odot$ Modulus multiplication

A. *Single round stage*

The basic structure for a single round is illustrated in Fig. 2. Specifically, it shows the structures for the first round. Next rounds have the same structure but with different sub-keys and ciphertext-derived inputs. The

[1] Departamento de Arquitectura y Tecnología de Sistemas Informáticos. Facultad de Informática. Universidad Politécnica de Madrid. Campus de Montegancedo s/n. Boadilla del Monte (28660 Madrid – SPAIN) email: ajo@adtech.es, victoria@pino.datsi.fi.upm.es
[2] Faculty of Electronics and Telecommunications. Technical University of Cluj-Napoca. C. Daicoviciu No. 15 (400020 Cluj Napoca – ROMANIA) email: Gabriel.Fericean@com.utcluj.ro, Monica.Borda@com.utcluj.ro

round, begins with an initial transformation that combines the four inputs sub-blocks (Y1, Y2, Y3, Y4) with four sub-keys (DK_1, DK_2, DK_3, DK_4), by using the addition and multiplication operations. The four outputs of this transformation (D1, D2, D3, D4) are then combined using the XOR operation to form two 16-bit blocks that are the input (D5 and D6) to the MA structure. The MA structure also takes two sub-keys (DK_5 and DK_6) as input and combines these inputs to produce two 16 bits-outputs. Finally, the four output blocks (D1, D2, D3, D4) from the upper transformation are XOR-operated with the obtained outputs (D9, D10) from the MA structure producing the four outputs blocks for this round. After this process, the output blocks Y1-2, Y1-3 are exchanged, so that Y1-1, Y1-3, Y1-2 and Y1-4 are used as input to the next round (in that order) along with the next 6 sub-keys. This procedure is followed for the eight rounds in total giving four output blocks: Y8-1, Y8-3, Y8-2 and Y8-4.

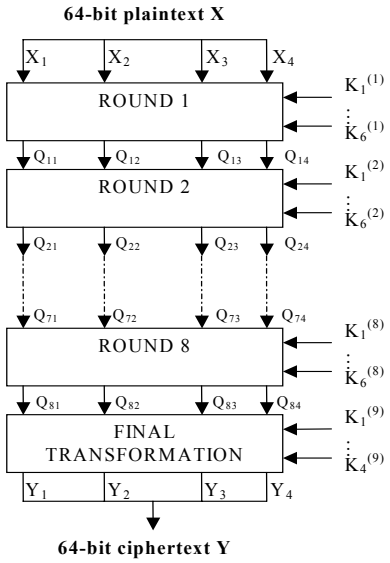

Figure 1. IDEA general structure

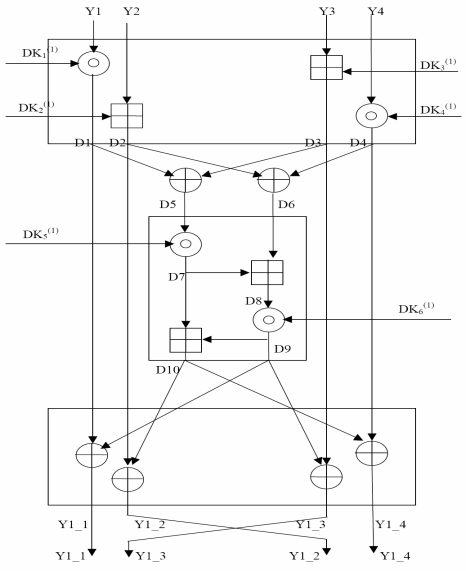

Figure 2. Structure for the first round

B. *Final transformation stage*

The final transformation stage has the same computational structure as the first transformation step of the structure for the first round, as can be seen in Figu. 3. The only difference is that the second and third inputs are interchanged before being applied to the operational units. This has the effect of undoing the interchange at the end of the eight rounds. The reason for this extra interchange is so that decryption has the same structure as encryption. This stage requires only four sub-key inputs, compared to six sub-key inputs for each of the first eight stages. The final four blocks, X1 to X4 are re-attached to form a 64-bit block of plain text. The whole process is repeated for each successive 64-block of ciphertext until all of the ciphertext has been decrypted.

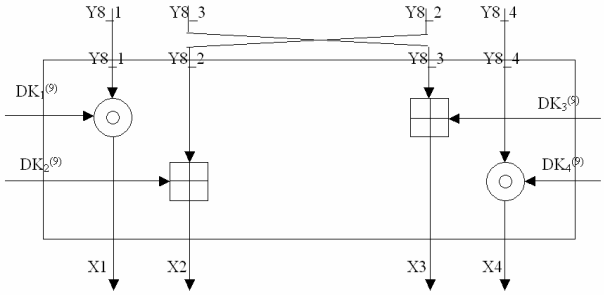

Figure 3. Final transformation structure

C. *Sub-key generation*

As mentioned earlier the algorithm works exactly the same in encryption and decryption modes, with the only difference being that sub-keys used for addition and multiplication are different. Encryption uses sub-keys derived directly from the main key. For the purpose of decryption, inverse sub-keys derived from the encrypted ones, are used.

D. *Encryption*

All 52 sub-keys used in encryption are obtained from the input key K. The scheme for generation is as follows. The first eight sub-keys are taken directly from the key. Then a circular left shift of 25 positions is applied to the key, and the next eight keys are extracted. This procedure is repeated until all 52 keys are generated.

E. *Decryption*

The derivation of the decryption sub-keys from the encryption is as follows: The first four sub-keys of decryption round r are derived from the first four sub-keys of encryption round (10 – r), where the final transformation is counted as round 9. The first and fourth decryption sub-keys are equal to the multiplicative inverse modulus (2^{16} + 1) of the corresponding first and fourth encryption sub-keys. For rounds 2 through 8, the second and third decryption sub-key share is equal to the additive inverse modulus (2^{16}) of the corresponding third and second encryption sub-keys. For rounds 1 and 9, the

second and third decryption sub-keys are equal to the additive inverse modulus (2^{16}) of the corresponding second and third encryption sub-keys. And finally for the first eight rounds, the last two sub-keys of decryption round r are equal to the last two sub-keys of the encryption round (9 - r).

III. IMPLEMENTATION ISSUES AND RESULTS

The general structure described above was modeled in VHDL, according to the restrictions and recommendations for high level behavioral synthesis [5]. The data format size was defined as a generic parameter taking value 16, as default. The design strategy adopted was bottom up, developing in the first instance the basic blocks: XOR, modulus adder and multiplier. The complex blocks of the structure were constructed by using the basic blocks, and they were designed using the available techniques for circuit technology-independent power reduction at the RTL level. These techniques mainly focus on better management of switching activity of the dynamic power consumption. Thus, pipelining and path balancing techniques have been applied to avoid glitch propagation and to balance the delay among basic blocks [6]. The basic blocks, the arithmetic operators, one round structure and the complete system with 8 rounds were simulated, synthesized and tested using the EDA tool Quartus II from Altera. The designs have been implemented on the NIOS development board EP2S60F672C5ES. This provides a hardware platform for developing embedded systems based on Altera Stratix II devices. The results are measured in terms of the resource utilization and delays. Concerning the resource demand, the number of adaptive look-up tables (ALUT) and DSP blocks and registers are given. The measured delays are the delays between an input and an output (I/O t_{pd}) and the key changes to output (K/O t_{pd}).

A. *Arithmetic operators*

The addition operations, both modulus 2 and modulus 2^{16} are implemented by using the XOR and '+' addition defined in the IEEE.STD_LOGIC_1164 and IEEE.NUMERIC_STD libraries. The multiplication modulus $2^{16}+1$ is a slightly more complex operation than the others. It has been implemented following an algorithmic approach that includes unsigned multiplication, addition, subtraction and comparison operations:

1. if operand1 = 0 then operand1 := 2^{16}
2. if operand2 = 0 then operand2 := 2^{16}
3. multiplication := operand1*operand2
4. div := multiplication/2^{16}
5. rem := multiplication mod 2^{16}
6. if (rem>div) then result := rem − div
   else result := rem − div + 2^{16} + 1

The implementation of the multiplication modulus $2^{16}+1$ uses the modulus 2^{16} multiplication synthesized by the development tool. The division and modulus is calculated by taking the upper 16 bits and lower 16 bits of the multiplication result. The addition and subtraction are the operations defined in the IEEE.NUMERIC_STD library for the UNSIGNED data type. The comparisons used in the implementation are UNSIGNED '>' and '=0'.

The key distribution block is implemented using a simple structural design, producing every key for the different stages of the algorithm from the 128 bit encryption key. The following table shows the synthesis results for the basic operators involved in the algorithm. The adders demand a similar amount of physical resources but the adder modulus 2^{16} is around a 1.3 times slower that the adder modulus 2. The multiplier is 1.9 times slower than the adder modulus 2^{16} and 2.5 times slower than the adder modulus 2. These delay factor relations will be of interest when balancing the delays among blocks, as we will see later on.

Table 1. Synthesis results for the basic arithmetic operators

Operation	**ALUs**	**DSP**	**I/O tpd**
Multiplication mod $2^{16}+1$	73	2	24.9ns
Addition mod 2^{16}	16	0	12.9ns
Addition mod 2	16	0	9.8ns

B. *Rounds*

We have implemented a direct version of the IDEA algorithm as it is shown in the dependencies graph in Fig. 2, by using the operators described in the last section. The synthesis results obtained for the INITIAL, MA and XOR blocks, a single round, the output transformation, and the complete IDEA algorithm composed of eight rounds are shown in Table 2. The INITIAL, the MA and final transformation blocks involve the same type and number of operations. They all demand 4 DSP units as they have two multipliers embedded. The number of ALUT's is similar for both the MA and Final transformation blocks and it is higher in the INITIAL because the two adders modulo 2 have been enclosed on this block. Concerning the delay, the MA block is the slowest of all three. This is due to the way in which the adders and multipliers are connected. In this case, the realization of a multiplication is followed by realization of an addition and vice versa. While in the other two blocks the four operations are independent and can be done in parallel.

Table 2. Synthesis results for the direct implementation.

Block	**ALUTs**	**DSP**	**K/O t_{pd}**	**I/O t_{pd}**
INITIAL	210	4	< I/O	27.3ns
MA	164	4	< I/O	42.5ns
XOR	64	0	<I/O	10ns
Single round	407	8	59.2ns	58.3ns
Final transformation	178	4	25.9ns	25.5ns
IDEA	3135	68	418ns	414ns

The delay results obtained for the complete IDEA algorithm implies a processing capacity of:

2.41 Mwords/s * 64bits/word = 154 Mbps
= 19.28 MBytes/s

A global view of the RTL synthesis results for the complete system is shown in Fig. 4.

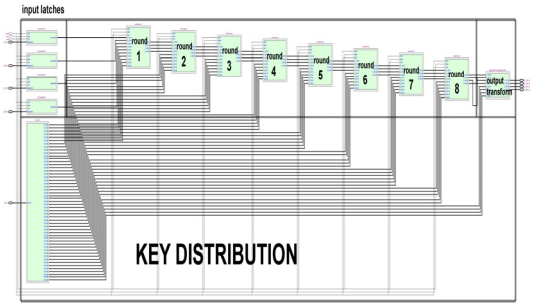

Figure 4. RTL synthesis results

In order to balance the delay among the blocks the action of each round inside the algorithm is decomposed into three pipeline stages, as shown in Fig. 5. The first stage holds the result of the first block and the first two addition operations: D1, D2, D5, D6, D3 and D4, from Fig. 2. The second stage holds the result of the MA block: D10 and D9, and again D1, D2, D3 and D4. The third stage holds the result of the output block: Y1_1, Y1_3, Y1_2, Y1_4. Each pipeline stage is controlled by a clock signal, enable signal, and reset signal, allowing for the control by an external asynchronous state machine.

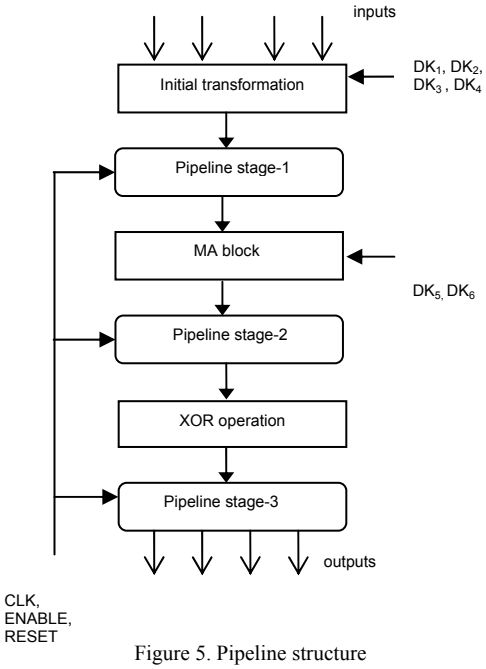

Figure 5. Pipeline structure

Finally, the eight rounds and the final transformation stage are connected together with the key distribution block to build up a complete IDEA encryption or decryption system. That makes a total of 25 pipeline stages (8*3 + 1 for the input latch), which means that 25 cycles are needed to get 64 bits of data from the input to the output. Table 3 shows the synthesis results for the pipelined structure shown in Fig. 5.

Table 3. Synthesis results for the pipeline implementation

Block	ALUT	Registers	DSP	F.max (MHz.)
INITIAL	226	96	4	N/A
MA	180	96	4	N/A
XOR	64	64	0	N/A
Single round	470	256	8	39.37
Final transfor	178	0	4	N/A
IDEA	3.945	2.112	68	37.13

The results in terms of ALUTs and DSP resources are similar to the ones obtained for the direct implementation case in Table 2. But now, additionally, a 2.112 bit register is needed. The max frequency for the complete IDEA algorithm is 37.13 MHz, which means:

37.13 Mwords/s * 64bits/word = 2.376.32 Mbps
= 297.04 MBytes/s.

IV. SUMMARY

In this paper, the design of the IDEA encryption and decryption algorithm oriented toward a high level synthesis for FPGA's implementation has been described. Two different structural approximations have been proposed. The first is based on the direct mapping to natural operations sequence and the second is an improved version introducing three pipelining stages. Both versions demand the same number of ALUT's and DSP blocks. The pipeline version demands the additional amount of 2.112 registers but this allows for a speed improvement of 15.37 times. The pipelined version of this design is suitable for high speed communication, video or audio encryption. The non pipelined version could be useful in secure controllers, and accessing encrypted memory program or data.

REFERENCES

[1] Stallings W. "Cryptography and Network Security: Principles and Practice Second Edition", Prentice Hall, New Jersey, 1999.

[2] Borko Furth, Darko Kirovski, "Multimedia Security Handbook", February 17, 2004.

[3] W. Mao, *Modern Cryptography*, Prentice-Hal, 2004.

[4] A. J. Menezes, V. Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, New York 1997.

[5] Michel Keating and Pierre Bricand, *Reuse Methodology Manual: For System-on-a-Chip Designs*. Third Edition. Kluwer Academic Publishers, 2002.

[6] Christian Piguet, Low-Power CMOS Circuits. Technology, Logic Design and CAD Tools. CRC Press 2006.