Military Technical College
Kobry El-Kobbah
Cairo, Egypt

10th International Conference
On Aerospace Sciences&
Aviation Technology

DESIGN AND IMPLEMENTATION OF IDEA ALGORITHM KEY SCHEDULE ON FPGA

Dr. Khaled Shehata
Assoc. Prof., AAST
Communication Dept.

Dr. Nabil Hamdy
MOD
Signal Dept.

Dr. Salah Elagooz
Assoc. Prof., MTC
Communication Dept.

Eng. M. Helmy
MTC
Communication Dept.

Abstract:

In this paper the design and implementation of the International Data Encryption Algorithm (IDEA) key schedule is presented. The IDEA key schedule takes 128-bit input key and returns 52 subkeys each of 16 bits during the encryption or the decryption operation. The key schedule includes the design of the inverse modulo $(2^{16}+1)$ multiplier and the inverse modulo 2^{16} adder. The inverse modulo multiplier circuit is used to generate 18 inverse multiplicative keys and the inverse modulo adder circuit is used to generate 18 inverse additive keys. The inverse multiplicative key is calculated through multiplying the key to the power $(2^{16}-1)$ modulo $(2^{16}+1)$. A 16 bit counter controls the inverse modulo multiplier circuit during the modulo multiplication process. A zero state problem is denoted during the generation of the inverse multiplicative keys because 2^{16} is treated as zero during the modulo $(2^{16}+1)$ multiplication in the encryption process.

The IDEA key schdule is implemented on Xilinx FPGA Spartan II family and the target chip is XC2S100-5PQ208C.

Key Words:

FPGA, Modulo $(2^{16}+1)$ multiplier, IDEA, Inverse Modulo $(2^{16}+1)$multiplication, Key Schedule, Implementation.

1- Introduction:

IDEA cipher was developed to increase the security against differential cryptanalysis [1-3]. IDEA encrypts or decrypts 64-bit data blocks using symmetric 128-bit key. The 128-bit key is expanded further to 52-subkeys blocks each of 16 bits. The plaintext block is divided into four quarters, each of 16 bits long. The block cipher IDEA is believed to be very secure due to the proper interaction between three different group operations. This interaction adds confidence in IDEA's security. The reason beyond choosing the implementation of IDEA key schedule is that; IDEA is one of the most secure block algorithm available to the public at this time and no currently known attack against the full IDEA rounds performs better than exhaustive search [4-10]. To complete the IDEA algorithm implementation, IDEA key schedule must be done. IDEA is an iterated cipher consists of 8 rounds followed by an output transformation as shown in Figure 1.

The cipher is based on a design concept of mixing operations from different algebric groups. The three groups used are on pairs of 16-bit subblocks namely, bit-by-bit XOR, addition module 2^{16} and multiplication modulo $(2^{16}+1)$.

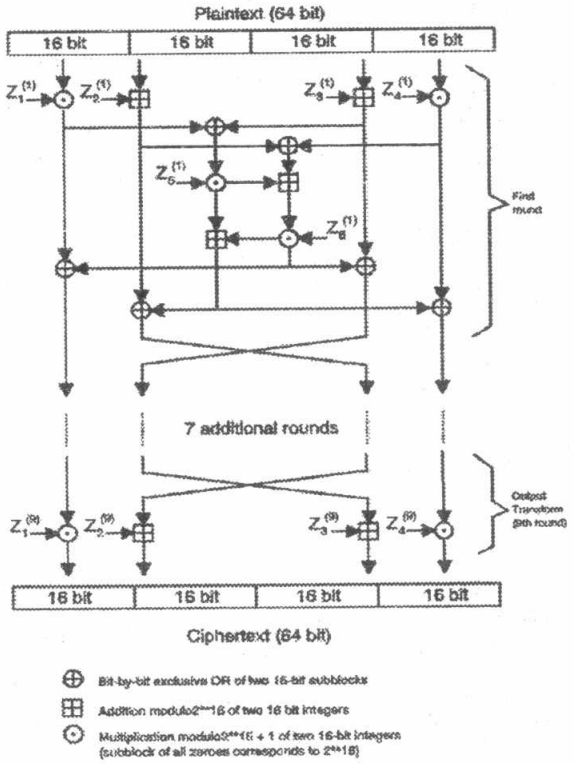

Fig. 1 IDEA Block Cipher Block Diagram [15]

As shown in Figure 1, each round consists of two stages the transformation stage and the multiplication addition (MA) stage. The two multiplications and the two additions at the beginning of Figure 1 are called the transformation stage and its process is reversable. The two multiplications and the two additions in the middle of Figure 1 are called the MA stage. The MA satage process is not reversable.

IDEA key schedule must be implemented along with the IDEA algorithm on the same FPGA because the generation of 52 subkey blocks requires a chip having 832 I/O, only for this purpose, to communicate with the IDEA FPGA. Another solution for the key schedule implementation is runtime reconfigurable such that the key schedule is done by directly modifying the bit stream download to the FPGA, thus enabling the implementation without any logic gates required for the key schedule. The first approach is the most suitable one, because the 52 subkey need to be generated inside the IDEA FPGA.[11]

The IDEA algorithm key schedule consists of input 128-bit register, control unit, inverse modulo (2^{16} + 1) multiplier unit, inverse modulo 2^{16} adder unit, swapping keys unit and 52 register each of 16 bits for the subkeys.

The latency to generate the decryption keys is much longer than the latency to generate the encryption keys. Most of the IDEA key schedule modules are easily implemented and the only overhead is introduced by the inverse modulo (2^{16}+ 1) multiplier. The inverse modulo multiplication operation has the maximum latency during the generation of the decryption keys.

This paper focuses on implementing the IDEA key schedule in both encryption and decryption operations on Xilinx FPGA spartan II family to produce the needed subkeys for the encryption or the decryption process.

Section 2 introduces the analysis of IDEA key schedule including the generation of the encryption and the decryption keys. The analysis of the key expansion for IDEA algorithm and the software verification for the design are introduced. Section 3 introduces the circuits design, sub modules and the implementation of the IDEA key schedule. Section 4 presents the simulation results. Section 5 gives concluding remarks of implementing the IDEA key schedule.

2- Analysis of the IDEA Key Schedule:

The 52 key subblocks of 16 bits used in encryption process are generated from the 128-bit user selected key as follows: The 128-bit key is partitioned into 8 subblocks that are directly used as the first eight key subblocks. The 128-bit key is then cyclic shifted to the left by 25 positions, after which the resulting 128-bit block is again partitioned into eight subblocks that are taken as the next eight key subblocks. The obtained 128-bit block is again cyclic shifted to the left by 25 positions to produce the next eight key subblocks. The cyclic shift left procedured is repeated 6 times until all 52 key subblocks are generated as shown in Figure 2.

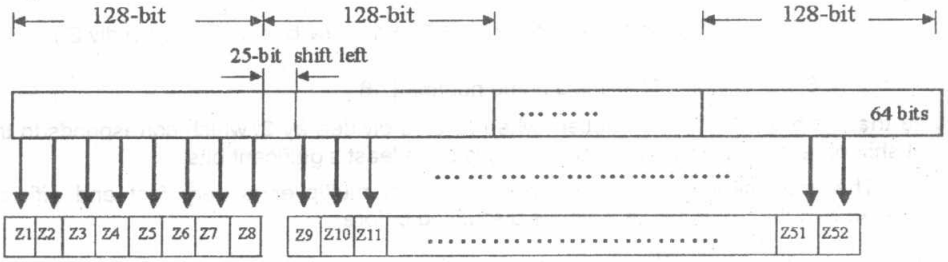

Fig. 2 IDEA Key Schedule Block Diagram

The 52 key subblocks of 16 bits used in the decryption process depends on the 52 key subblocks previously generated from the 128 bit user-selected key during the encryption process as shown in Table 1.

2.1 Mathematical Formula for Modulo ($2^n + 1$) Multiplier:

The modulo multiplication function is calculated as follows :

$$a * b \bmod n = a * b - q * n . \quad \text{where } q = \text{integer } (a * b) / n . \qquad (1)$$

There are two methods that can be used to implement the modulo ($2^{16}+1$) multiplier either Division by (subtract and shift) [12] or Systolic Array [13]. The division technique is based on subtracting the modulo number from the multiplication result until the subtraction result is less than the modulo number. The Systolic Array is based on Montogomry Algorithm [14]. The Systolic Modular multiplication throughput is one modular multiplication every clock cycle with a latency of (2n+2) cycles for multiplicands having n digits.

The analysis of the hardware implementation for the two techniques shows that, the first technique is very slow and the result comes after several times of subtraction controlled by a comparator while the second technique is very hard to implement beside it needs huge number of logic gates and needs some post-processing before the modulo multiplication result comes.

Table 1. Encryption & Decryption keys subblocks:

Stage	Encryption	Decryption
Round 1	$Z_1, Z_2, Z_3, Z_4, Z_5, Z_6$	$Z_{49}^{-1}, -Z_{50}, -Z_{51}, Z_{52}^{-1}, Z_{47}, Z_{48}$
Round 2	$Z_7, Z_8, Z_9, Z_{10}, Z_{11}, Z_{12}$	$Z_{43}^{-1}, -Z_{45}, -Z_{44}, Z_{46}^{-1}, Z_{41}, Z_{42}$
Round 3	$Z_{13}, Z_{14}, Z_{15}, Z_{16}, Z_{17}, Z_{18}$	$Z_{37}^{-1}, -Z_{39}, -Z_{38}, Z_{40}^{-1}, Z_{35}, Z_{36}$
Round 4	$Z_{19}, Z_{20}, Z_{21}, Z_{22}, Z_{23}, Z_{24}$	$Z_{31}^{-1}, -Z_{33}, -Z_{32}, Z_{34}^{-1}, Z_{29}, Z_{30}$
Round 5	$Z_{25}, Z_{26}, Z_{27}, Z_{28}, Z_{29}, Z_{30}$	$Z_{25}^{-1}, -Z_{27}, -Z_{26}, Z_{28}^{-1}, Z_{23}, Z_{24}$
Round 6	$Z_{31}, Z_{32}, Z_{33}, Z_{34}, Z_{35}, Z_{36}$	$Z_{19}^{-1}, -Z_{21}, -Z_{20}, Z_{22}^{-1}, Z_{17}, Z_{18}$
Round 7	$Z_{37}, Z_{38}, Z_{39}, Z_{40}, Z_{41}, Z_{42}$	$Z_{13}^{-1}, -Z_{15}, -Z_{14}, Z_{16}^{-1}, Z_{11}, Z_{12}$
Round 8	$Z_{43}, Z_{44}, Z_{45}, Z_{46}, Z_{47}, Z_{48}$	$Z_7^{-1}, -Z_9, -Z_8, Z_{10}^{-1}, Z_5, Z_6$
Transformation	$Z_{49}, Z_{50}, Z_{51}, Z_{52}$	$Z_1^{-1}, -Z_2, -Z_3, Z_4^{-1}$

The used mathematical formula to implement modulo $(2^{16}+1)$ multiplier is as follows [15]:

Let a and b be two n-bits of non zero integers. Then:

$$a*b \bmod (2^n+1) = \begin{cases} (a*b \bmod 2^n) - (a*b \operatorname{div} 2^n) & \text{if } (a*b \bmod 2^n) \geq (a*b \operatorname{div} 2^n) \\ (a*b \bmod 2^n) - (a*b \operatorname{div} 2^n) + 2^n + 1 & \text{if } (a*b \bmod 2^n) < (a*b \operatorname{div} 2^n) \end{cases} \quad (2)$$

For n = 2, 4, 8, 16, in which $2^n + 1$ is a prime number [16].

Note that: $(a*b \operatorname{div} 2^n)$ is the quotient when (a*b) is divided by 2^n which corresponds to the right shift of (a*b) by n bits and $(a*b \bmod 2^n)$ is the n least significant bits.

The implementation of this modulo $(2^{16}+1)$ multiplier is very fast and efficient compared with the other two techniques discussed before.

2.2 Inverse Modulo ($2^{16}+1$) Multiplier:

The transformation stage and the output transformation each includes two modulo multipliers and their functions are reversable which means using secret key for encryption and its multiplicative inverse for decryption. The MA stage includes two modulo multipliers and their functions are not reversable which means using the same secret key for both operations encryption and decryption.

2.2.1 Computing Inverses:

Modular arithmetic sometimes permits the computation of multiplicative inverses. The analysis to implement the inverse modulo $(2^{16}+1)$ multiplier shows that, computing the multiplicative inverses can be done using one of the following three theorems [17]:

Euler Theorem, Extended Euclidean Theorem, and Chinease Remainder Theorem.

Euler Theorem is preferable for the hardware implementation over the other two theorems as it is very efficient and also applicaple for the hardware implementation.

2.2.2 Euler Theorem:

An important quantity in number theory which is reffered to as Euler's totient function and written as Ø(n). Euler's totient function is the number of positive integers less than and relatively prime to n.

It should be clear that for a prime number p: Ø (p) = p-1.

Thus to get the multipticative inverse (a^{-1})such that (a) is relatively prime to (n):

$$a^{-1} \bmod n = a^{\emptyset(n)-1} \bmod n. \quad (3)$$

For prime number $n=(2^{16}+1)=65537$, key K then $\emptyset(2^{16}+1) = 65536$ and

$$K^{-1} \bmod (2^{16}+1) = K^{\emptyset(2^{16}+1)-1} \bmod (2^{16}+1) = K^{65535} \bmod (2^{16}+1). \quad (4)$$

This means the inverse multiplicative key is calculated through multiplying the key to the power $(2^{16} - 1)$ modulo $(2^{16} + 1)$.

2.2.3 IDEA Key Schedule Zero Input Problem:

The problem with this inverse modulo $(2^{16}+1)$ multiplication operation appears when the output of a modulo multiplication operation is 2^{16} which is intercepted as zero and this zero will be multiplied by the key in the next step during the 65535 times of modulo multiplication. If the modulo multiplication result is zero, it is not true and will result in incorrect inverse key. Therefore, this zero output has to be detected.

This state is detected during the computation of the inverse modulo multiplication key in order to correct the inverse key result as shown by the following equation.

$$K * K^{-1} \bmod (2^{16} + 1) = 1. \quad (5)$$

2.2.4 The Solution for IDEA Key Schedule Zero Input Problem:

The solution for the IDEA key schedual problem, which appears during the generation of the inverse modulo multiplication keys, is concerned with the detection of the zero result during the process. The inverse modulo multiplier circuit treats the zero result as 2^{16} in the next operation of modulo multiplication. This means multiplying the key with 2^{16} instead of zero. The solution gives the correct inverse modulo $(2^{16} + 1)$ multiplication key that satisfy equation (5).

2.3 Inverse Modulo 2^{16} Adder:

The inverse modulo 2^{16} adder generates the inverse modulo addition keys. This operation is calculated through the subtraction process of the key from 2^{16}. The result is the inverse modulo addition key used during the decryption process.

2.4 Software Verification:

A software verification for the IDEA key schedule using the above technique for solving the zero input state problem results in correct inverse modulo multiplicative keys and consequently correct encryption/decryption process of the IDEA. Also, this verification provides a validity for the hardware implementation of the IDEA key schedule.

3- Design and Circuit Description:

The designed and implemented circuit is the IDEA key schedule, which includes the inverse modulo multiplication circuit and the inverse modulo addition circuit. The circuit consists of several sub-modules. All sub-modules design, simulation and verification are explained. Figure 3 shows the block diagram of the IDEA key schedule.

First the IDEA key schedule consists of four main blocks the control unit, the inverse modulo multiplication unit, the inverse modulo addition unit and the swapping keys unit. **Second** the input 128 bits key undergoes three operations at the same time as follows:

(1) At the Inverse modulo multiplication unit, the input 128 bits key are partitioned into 18 subblock keys each of 16 bits. Then these 18 subblock keys are taken to form the keys for the encryption process and generate the inverse modulo multiplication keys

for the decryption process. The Inverse modulo multiplication unit affects the Control unit in the sense that it has the maximum latency.

(2) At the Inverse modulo addition unit, the input 128 bits key are partitioned into 18 subblock keys each of 16 bits. Then these 18 subblock keys are taken to form the keys for the encryption process and generate the inverse modulo addition keys for the decryption process.

(3) At the Swapping keys unit, the input 128 bits key are partitioned into 16 subblock keys each of 16 bits. Then these 18 subblock keys are taken to form the keys for the encryption process and the keys for the decryption process.

(4) The Control unit is responsible for all control signals to the other three units. It has three output control signals the first is the encryption / decryption control signal, the second is the select subkey control signal in the decryption process and the third is the select output subkey control signal.

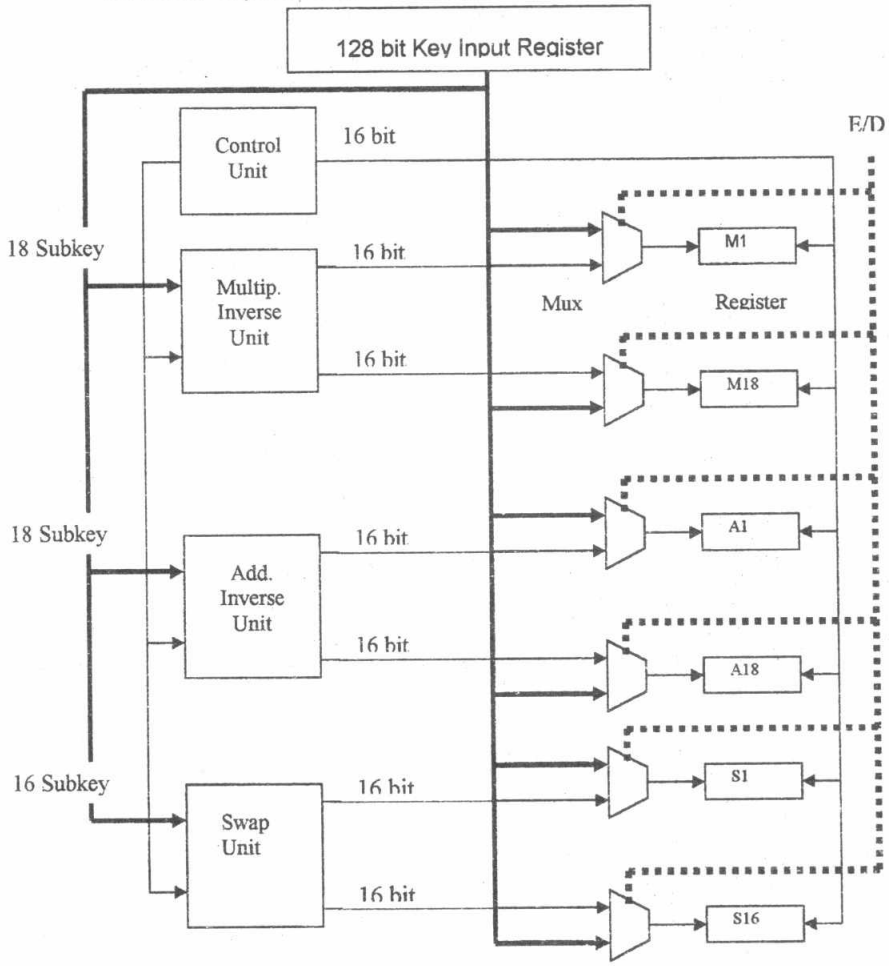

Fig. 3 IDEA Key Schedule Block Diagram

3.1 Inverse Modulo (2^{16} + 1) Multiplier Block Diagram:

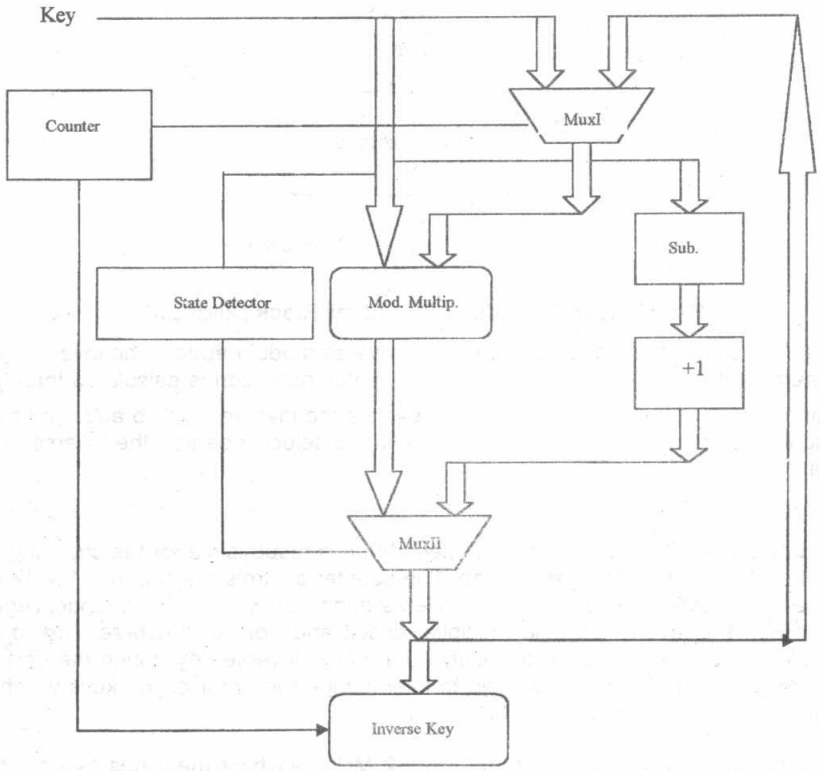

Fig. 4 Inverse Modulo (2^{16} + 1) Mu ltiplier Block Diagram

Figure 4 shows the block diagram of the inverse modulo multiplier. The inverse modulo multiplier has one input and one output each of 16 bits.

First, the input key enters 2:1 MUX I which is controlled via 16 bit counter and this output is the first input of the modulo multiplier. The counter starts the inverse modulo multiplication operation and control the output from the inverse modulo multiplier. The same input enters the modulo multiplier circuit.

Second, the two inputs to the modulo multiplier are modulo multiplied in parallel.

Third, the output from the modulo multiplier is fed back again to the MUX I and this operation is done 65534 until the inverse multiplicative key is produced.

If the output from the MUX I is all zero then this output is encoded to become the key shift left 16 bits plus one which enable the operation to continue to get the correct multiplicative inverse key.

As shown in Figure 4 the multiplicative inverse key is produced after 65535 clocks using Euler Theorem. Therefore, before the decryption process starts and to get the inverse multiplicative keys for 18 keys, the key schedule needs 18 X 65535 clocks which is very large number of clocks.

The Inverse multiplicative key unit consists of 18 : 1 MUX with 5 select lines and the Inverse multiplicative key circuit.

3.2 Inverse Modulo 2^{16} Adder Block Diagram:

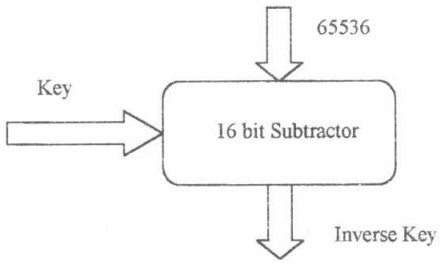

Fig. 5 Inverse Modulo 2^{16} Adder Block Diagram

Figure 5 shows the block diagram of the inverse modulo adder. The inverse modulo adder generates the inverse modulo addition keys, this operation is calculated through the subtraction process of the key from 2^{16}. The result is the inverse modulo addition key. The Inverse additive key unit consists of 18 : 1 MUX with 5 select lines and the Inverse additive key circuit.

3.3 Design of the Control Unit:

It consists of 5 bit counter and 5 bit decoder. It is responsible for the choosing of the process whether encryption / or decryption. The counter controls the two 18 : 1 MUX inside the Inverse multiplicative key unit and the Inverse additive key unit. The decoder regulates the output from the Inverse modulo multiplier circuit and from the Inverse modulo adder circuit. It enables the right register to receive the proper inverse key during the decryption process. The swapping keys unit is used to interchange the order of the keys which were used during the encryption process in MA stage.

Finally, the key schedule module has 52 MUX, each of them has two inputs, the encryption key and the decryption key. These MUXs are controlled via the control signal encryption / decryption.

3.4 Design of the Sub-Modules:

Figure 2 shows the overall sub-modules of the IDEA key schedule circuit. The different sub- modules design is explained as follows:

(a) Modulo (2^{16}+1) multiplier:

Figure 6 shows the block diagram of the modulo multiplier which has two inputs and one output each of 16 bits. The operation of this modulo multiplier is explained as follows:

First the two inputs are multiplied with the array multiplier. The output of the multiplier is divided into two sections, the 16 most significant bits and the 16 least significant bits.
Second the subtractor subtracts the 16 most significant bits from the 16 least significant bits and at the same time the two sections of the multiplication result are compared.
Third the subtraction result is incremented by one. The comparator output is low if the 16 least significant bits is greater than or equal the 16 most significant bits. The comparator output controls 2 : 1 MUX which has two inputs, the subtraction result and the subtraction result plus one. If the comparator output is high, the MUX output is the subtraction result plus one otherwise, the MUX output is the subtraction result.

The modulo multiplier consists of several sub-modules which are 16 bit array multiplier, 16 bit subtractor, 16 bit comparator, one bit incrementer and 2 : 1 MUX.[18]

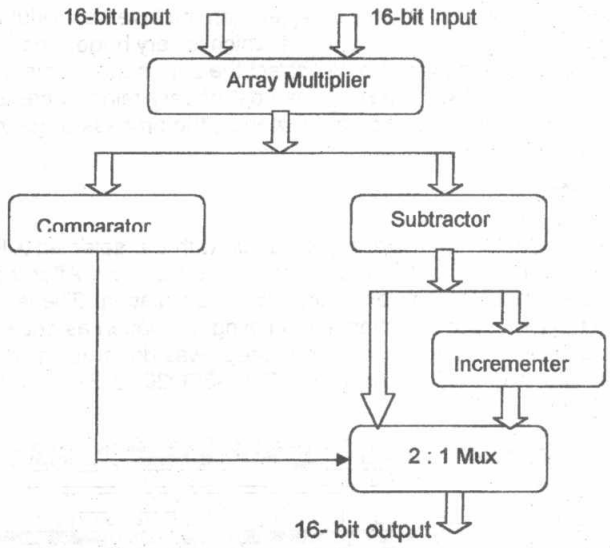

Fig. 6 Modulo (2^{16} + 1) Multiplier Block Diagram

(b) Array multiplier:

The multiplication is done in parallel using the array multiplier. It is two 16-bit parallel input circuit and 32-bit parallel output. The array multiplier is based on the idea that partial products in the multiplication process may be independently computed in parallel. An n*n multiplier requires n * (n-2) full adder, n-half adders, and n^2 AND gates [19]. The worst case delay of the multiplier is (2n +1)t, where t is the worst case delay of the adder having the longest path. The Array multiplier result in latency of one clock. The array multiplier has 224 full adder. The full adder has 3AND2 gates and 2XOR3 gates.

3.5 Design Integration:

The implemented sub-sub modules are simulated and then integrated to constitute the sub modules. The design features simplicity, speed and high rate of throughput. Table 2 shows the latency and the number of gates of the inverse modulo multiplier and the inverse modulo adder implemented modules.

Table 2 Implementations of Inverse modulo multiplier and Inverse modulo adder.

	Inverse modulo multiplier	Inverse modulo adder
Latency	$65535[(2n + 1)t_1 + nt_2 + t_3 + 3t_4]$	nt_2
No. of gates	1803	32

Where t_1 is the full adder latency inside the array multiplier, t_2 is the full subtractor latency, t_3 is the incrementer latency, and t_4 is the 2:1 MUX latency. Calculating nt_2 and comparing it with $(2n + 1)t_1 + nt_2 + t_3 + t_4$ multiplied by 65536. The result showed that nt_2 can be neglected.

The Inverse modulo multiplier latency is larger than the Inverse modulo adder latency at the order of 65535[(2n + 1) t_1 + nt_2 + t_3 +t_4] value which is very huge. The hardware of the Inverse modulo multiplier increased with 1% to correct the zero state problem in order to get correct inverse key. Results showed that, at the cost of very minor increase of the used number of gates about 1% with almost the same latency, the process of generating inverse multiplicative keys executed correctly.

4 – Simulation Results:

The simulation results for the IDEA key schedule with the selected Xilinix family and targeted fitting chip is presented. The schematic design entry is used. After the sub modules are individually simulated in both functional and timing simulation. These integrated sub modules are simulated again in both functional and timing simulation as shown in Figures 7. The design entry, simulation, and implementation strategy was done using Xilinx Foundation Series Ver 2.1 [20]. The target chip is Xilinx XC2S100-5PQ208C Spartan II family having 100.000 gates count [21].

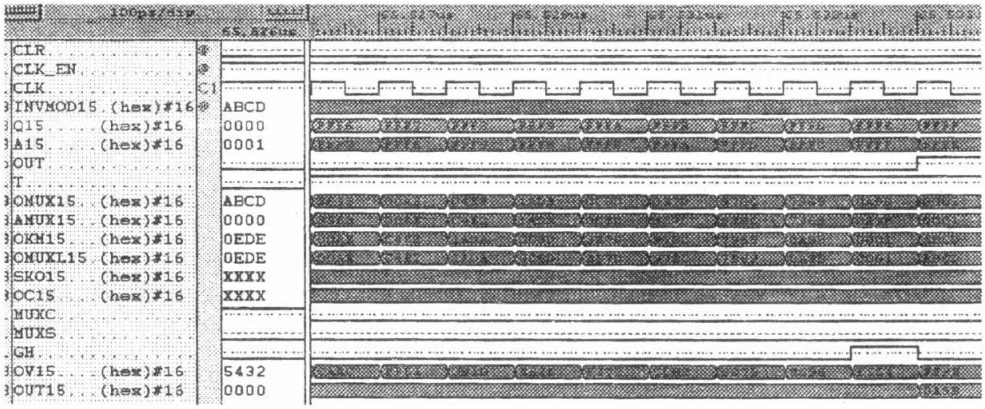

Fig. 7 Inverse Modulo (2^{16}+1) Multiplier Simulation Results

In Figure 7 the 16 bits input key applied to the Inverse modulo (2^{16} + 1) multiplier is INVMOD15. The counter state is Q15. The first Mux output is OMUX15. The second Mux output is OMUXL15. The subtractor output is SKO15. The incrementer output is OC15. The modulo multiplier output is OKMT15, and the output is OUT15. The inverse modulo multiplication result was exactely same as resulted from the software implementation. The input key is (0X ABCD) hexadecimal value and the output inverse multiplicative key is (0X 0A9B) generated after 65535 clocks.

5 – Conclusion:

The design of the IDEA algorithm key schedule including the design of Inverse modulo (2^{16} + 1) multiplier is presented, designed, simulated, and implemented on FPGA. The motivation of the design is the generation of the needed 52 subkeys blocks for the IDEA algorithm that enables IDEA to operate in both processes encryption and decryption. The advantage of the design is the pipelined architecture of the inverse modulo multiplier to accelerate the IDEA key schedule design. The maximum achieved speed is 50 MHZ which is due to the complexity of the inverse modulo multiplier design. The number of CLB is 810 for the IDEA algorithm key schedule. Hardware simulation results showed that, at the cost of a minor increase of the used number of gates in the Inverse modulo multiplier design about (1%) and with almost the same latency, the process of the generation of the inverse keys is executed correctly to correct the IDEA key schedule zero state problem.

6 – References:

[1] Bruce Schneier, Handbook of Applied Cryptography Protocols, Algorithm Source Code in C. Press 1993.

[2] Bruce Schneier, Journal (USA), Vol.18, no.13, p.50, The IDEA encryption algorithm, 1993.

[3] Borst J., Knudsen L. R, Rijmen V., Two attacks on reduced IDEA EUROCRYPT 97 International Conference in Germany,1997.

[4] John Kelsey, Bruce Schneier and David Wagner,"Key Schedule Cryptanalysis of IDEA, G-DES, Gost, Safer and Triple DES", Advances in Cryptology –Eurocrypt'97, Springer Verilag, 1997.

[5] Eli Biham, Alex Biryukov and Adi Shamir, "Miss in the Middle Attacks on IDEA, Khufu and Khafre", 1998.

[6] J. Borst, "Differential-Linear Cryptanalysis of IDEA", Technical Report ESAT-COSICReport, Departement of Electrical Engineering, Katholieke Universiteit Leuven, February 1997.

[7] L.R. Knudsen and V. Rijmen, "Truncated Differentials of IDEA", Technical Report ESAT-COSIC Report, Departement of Electrical Engineering, Katholieke Universiteit Leuven, Feb. 1997.

[8] J. Daemon, R. Govaerts, and J. Vandewalle, "Cryptanalysis of 2,5 rounds of IDEA", Technical Report ESAT- COSIC Report, Departement of Electrical Engineering, Katholieke Universiteit Leuven, March 1994.

[9] J. Daemon , R. Govaerts, and J. Vandewalle, "Weak Keys for IDEA", In T. Helleseth, Advances in Cryptology –Eurocrypt'94, pp 224-231.Springer Verlag, 1994.

[10] P. Hawkes, "Differential- Linear Weak Key Classes of IDEA", Lecture Notes on Computer Science 1403, Advances in Cryptology, Proceedings of Eurocrypt'98, pp112-126, Verilag 1998.

[11] O. Y. H. Cheung, K. H. Tosi, P. H. W. Leong, "Tradeoffs in Parallel and Serial Implementations of the IDEA ", Chinese University of Hong Kong, Computer Science and Engineering Department, 2001.

[12] Juris Blukis, Mark baker. Practical Digital Electronics. Second edition 1976.

[13] S. E. Eldridge, and C. D. Walter, Hardware implementation of Montgomery's modular multiplication algorithm, IEEE Transaction on computers. 1991.

[14] Colin D. Walter, Systolic Modular Multiplication IEEE Transaction on Computers, Vol.42, NO.3, March 1993.

[15] Nabil H., Khaled S., Atalla H., M. Helmy, Design and Implementation of High Performance modulo (2^{16} + 1) multiplier on FPGA, 3rd International Conference on Electrical Engineering, ICEENG, May 2002.

[16] William Stalling, Handbook of Network and Internetwork Security. Press 1993.

[17] Elizabeth Dorothy, "Cryptography and Data Security" 2nd Edition, Addison Weseely, California,1982.

[18] Thomas L. Floyd, Handbook of Digital Fundamentals. Press 1997.

[19] Neil H. E.Weste, Kamran Eshraghian. Principals of CMOS VLSI Design. 1994.

[20] S. Kelem, Virtex Configuration Architecture Advanced Users' Guide, Xilinx Inc., September 1999. Application Note XAPP151, Version 1.2.

[21] Ashock K. Sharma, Handbook of programmable Logic PLDs, CPLDs, and FPGAs. Press 1998.