A Hardware implementation of IDEA cryptosystem using a recursive multiplication approach.

Sourav Mukherjee
Department of Computer Science and Engineering
National Institute of Technology, Rourkela
Rourkela, Orissa, 769008
Email: - souravnitr@gmail.com

Bibhudatta Sahoo
Department of Computer Science and Engineering
National Institute of Technology, Rourkela
Rourkela, Orissa, 769008
Email: - bibhudatta.sahoo@gmail.com

***Abstract*—This paper covers the implementation of the International Data Encryption Algorithm (IDEA) using Very Large Scale Integrated Circuits Hardware Description Language (VHDL) with the help of Xilinx – ISE 10.1. In terms of security, this algorithm is very much superior and is already patented by Ascom. The whole algorithm is divided into modules and among all of them the most time consuming one is the modulo multiplication module. The multiplication algorithm that is used computes the product in a recursive fashion and it uses Divide and Conquer approach during multiplication, as mentioned in [2], which ultimately consumes less time and increases the throughput in the algorithm. Moreover the design is made pipelined for increasing the throughput. The block size considered here is same as of traditional IDEA encryption algorithm [1] which is of 64 bits with 16 bit sub-blocks [1].**

***Keywords*-CryptographicAlgorithm,IDEA,Hardware Implementations,Modulo Multiplier,VHDL,Partial Products.**

I. INTRODUCTION

Cryptography is the art of keeping data secure from unauthorized access so as to guarantee that only the intended users can access it. As computer technologies are getting advanced, more and more cryptographic applications are used. They are mainly used to support other applications which are very much sensitive to data security such as smart cards and commercial data exchange over a network. Not only for personal use but cryptographic algorithms are also very important in every aspect of professional activities.

A cryptographic algorithm generally consists of some specialized arithmetic computations which are complicated in terms of time complexity. It is because of the fact that these algorithms work with large amount of data either in blocks or simply in streams. Although a single traditional CPU is enough for performing these computations, but for a machine which works as a server in a huge network gets millions of client requests for performing cryptographic operations for them individually. This makes the workload huge. The computational resources may also be limited for example in smartcards, mobile phones, handheld computers, etc. Moreover if the associated network is of high speed, the speed of the necessary cryptographic computations also needs to be taken into account. For example in transmitting audio and video data for cable TV, pay TV, video conferencing and sensitive financial and commercial data, the speed of the cryptographic module to be embedded ,needs to be very high. Moreover for security related issues in wireless and sensor networks, there is a need for separate hardware device with very high processing rate because of limited battery of the nodes and for optimizing the bandwidth efficiency. So from the viewpoint of high speed and throughput, traditional software implementations of these complicated cryptographic algorithms are not efficient in real time applications like ATM, VPN, etc. This forces the system designers to go for hardware implementation of the cryptosystems [6].

Traditionally hardware implementations are based on ASIC technology, but they are not quite affordable every time especially in monetary terms. Moreover these ASICs are not adaptable to new changes once the hardware is built. The more efficient and convenient method is to use FPGA platforms which provides sufficient logics and storage elements on which any complex algorithm can be implemented [3]. They are adaptable to new changes and their granularity matches quite well with the cryptographic algorithms.

In this paper, the cipher used is a symmetric key block cipher named IDEA. It takes its input as 64 bit plain text and gives a 64 bit cipher text as output using a 128 bit key. While working on plain text, it divides the input data into 16 bit sub-blocks and operates on each block. It is described as one of the most secure block algorithm due to its high immunity to attacks. In spite of the fact that Data Encryption standard (DES) is another popular symmetric block cipher which is used in several financial and business applications, its drawback is the short key word length (56 bits), therefore highly prone to cryptanalysis attack. Thus IDEA is stronger compared to DES as it involves 128 bit key word length. Moreover unlike DES, IDEA doesn't need any S-Box or P-Box for permutation and substitution, so no memory module is required for implementing this cipher. The most crucial module of this algorithm is the design of the multiplier modulo a Fermat prime, which is one of the algebraic group operation used and the entire speed of IDEA depends on this module. So designing the multiplier is a major during the hardware or software implementation of IDEA because its speed is a big issue when hardware implemented IDEA is used in real time applications. The overall objective for hardware implementation of IDEA is to minimize the hardware requirements which results in

efficient use of silicon area and at the same time improve the processing speed and high throughput of data. As the performance of IDEA cipher depends entirely on the modulo (2^n + 1) multiplier design, the main objective is to design an efficient and fast modulo multiplier which is to be used in the entire IDEA algorithm.

The organization of the rest of the paper is as follows. The previous hardware and software implementations are covered in section II. Section III describes the IDEA cipher and its detailed operations as well as modules. Section IV describes the general architecture of the cryptosystem to be implemented and the proposed modulo multiplier architecture. Section V discusses the performance reviews and comparisons with previous schemes and section VI finally concludes the paper.

II. PREVIOUS WORK

In spite of the fact that IDEA works with 16 bit word blocks, software implemented IDEA cannot reach the speed that is required for online encryption in high speed networks. IDEA was implemented in software by Ascom, the patent holder of IDEA, and it achieved an encryption rate of 23.53 Mbps. Helger [11] proposed an approach using the Intel Pentium II 233MHz machine and achieved an encryption rate of 32.9 Mbps. Mencer [12] proposed a design of IDEA processor which achieved 528 Mbps on 4 XC4020XL devices. The first VLSI implementation of IDEA was developed and verified by Bonnenberg [13] using a CMOS technology with an encryption rate of 44 Mbps. With a system clock frequency of 25 MHz, Curiger et al. performed 177 Mbps VLSI implementation of IDEA [8]. Wolter reported a 355 Mbps VLSI implementation [14] in 1995. This is followed by Salomao's approach of single round implementation on chip with 424 Mbps data conversion rate. In another approach, the modulus multiplier is optimized using temporal parallelism and implemented with VHDL with a data conversion rate of 522 Mbps with comparatively less area requirements. Later Leong [3] proposed a 500 Mbps bit serial implementation of IDEA on Xilinx Virtex XCV300 -6 FPGA which is followed by Goldstein's approach with conversion rate of 1013 Mbps. Finally Ascom developed IDEACrypt Kernel with a speed of 720 Mbps. Recently Thaduri [5] implemented IDEA cipher having a throughput of 700 Mb/s.

III. THE IDEA BLOCK CIPHER

In this section, the entire algorithm for the IDEA block cipher is elaborated. It is a symmetric key cipher. The block size of data on which IDEA operates, is of 64 bit and the key size is of 128 bits. But all data operations in IDEA cipher are in 16 bit unsigned integers. The length of the incoming data should be either in normal in integer multiple of 64 bits or if not, is made by using padding bits. At the end of the algorithm, a 64 bit cipher text is created.

A. Basic structure of IDEA cipher.

IDEA is based on mixing operation of three different algebraic groups which are

- XOR (bitwise).
- Addition modulo 2^n.
- Multiplication modulo $(2^n + 1)$.

The security of IDEA depends on these three operations. The basic structure of IDEA cipher [9] is shown in Figure 1.

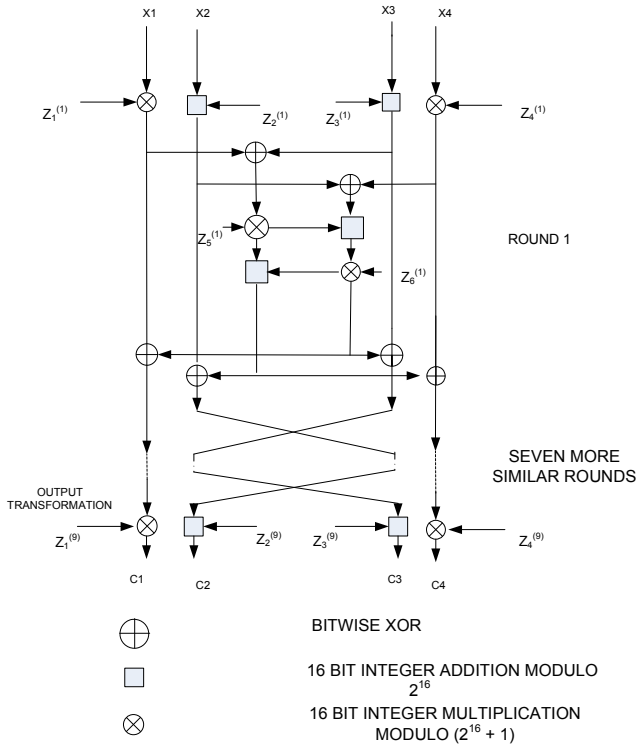

Figure 1. Basic structure of IDEA Cipher and its data flow

The IDEA cipher consists of 8 rounds which are identical in nature and a last output transformation round which is similar to upper half of any round. Before the starting of 1st round, the input 64 bit plain text is divided into four 16 bit sub-blocks, X1, X2, X3 and X4 respectively. At the end of encryption phase, four 16 bit sub-blocks of cipher text is created. Each round uses six 16 bit sub-key blocks $Z_1^{(n)}$, $Z_2^{(n)}$, … , $Z_6^{(n)}$ which are made from the input 128 bit key. The super-script n denotes the nth round. The output transformation phase, which is considered as 9th or the last round, uses 4 sub-keys, $Z_1^{(9)}$, $Z_1^{(9)}$, $Z_1^{(9)}$, $Z_1^{(9)}$. Every round except the 1st round, uses the output sub-blocks produced in the previous round. In between every round, the 2nd and the 3rd sub-blocks are swapped. The entire algorithm uses only three different algebraic group operations which are XOR, addition modulo 2^{16} and multiplication modulo (2^{16} + 1). The encryption phase of IDEA thus uses [(8*6) + 4] i.e. 52 sub-key blocks, which are made from the 128 bit input key. As IDEA involves only algebraic operations, no look-up tables or S-Boxes are used like DES or AES.

The decryption phase of IDEA is identical to that of the encryption phase. It uses the same sequence of operations as in the encryption phase. The only change is that the sub-keys are reversed and are slightly different. That means the sub-keys which are used in round 1 during encryption phase are

manipulated during last round of decryption phase. The sub-keys used in decryption are either additive or multiplicative inverse of the sub-keys used in the encryption phase.

B. Key Generation

The key generation phase of IDEA generates 52 sub-keys from the 128 bit input key. The basic steps of generating the encryption keys are:

- All the sub-keys are named as $Z_1^{(1)},\ldots Z_6^{(1)}$, $Z_1^{(2)},\ldots Z_6^{(2)}$, … , $Z_1^{(8)},\ldots Z_6^{(8)}$, $Z_1^{(9)},\ldots Z_4^{(9)}$.
- From the input 128 bit key, eight sub-blocks of 16 bits are partitioned and are assigned to $Z_1^{(1)},\ldots Z_2^{(2)}$ directly.
- Now the original 128 bit key block is rotated by 25 bits and a new 128 bit block is formed. Now another eight sub-blocks are generated from this new block.
- The rotation procedure is repeated until and unless sub-blocks used in previous rounds are found.

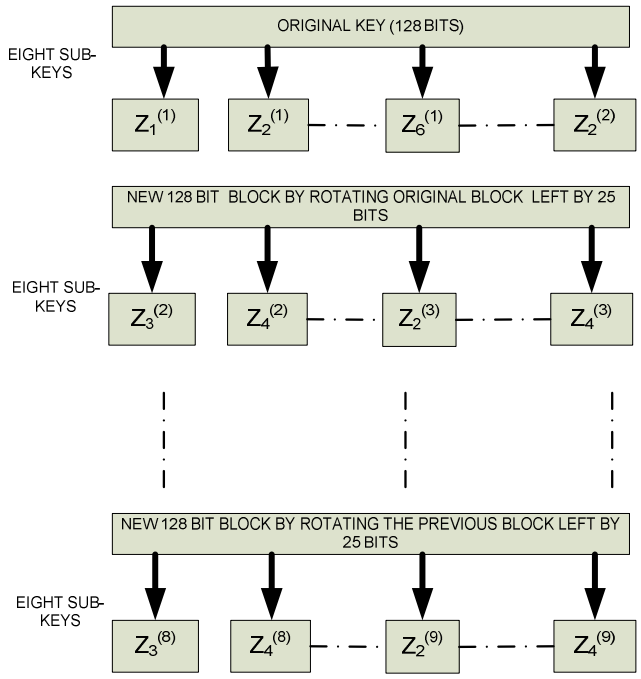

Figure 2. IDEA Encryption Key generation

Once the encryption keys are generated, the decryption keys can be generated directly by taking their additive inverse modulo 2^{16} and multiplicative inverse modulo $(2^{16} + 1)$ as required.

IV. ARCHITECTURE AND VHDL IMPLEMENTATION

The main objective of implementing a cryptosystem is to increase the data encryption and decryption rate so as to increase the throughput. FPGA implementations of cryptosystem offer adequate speed so that it can be easily embedded in real time applications. To implement an algorithm in hardware, we have to first realize the architecture of the entire system. The general architecture of hardware implementation of a cryptosystem is shown in Figure 3.

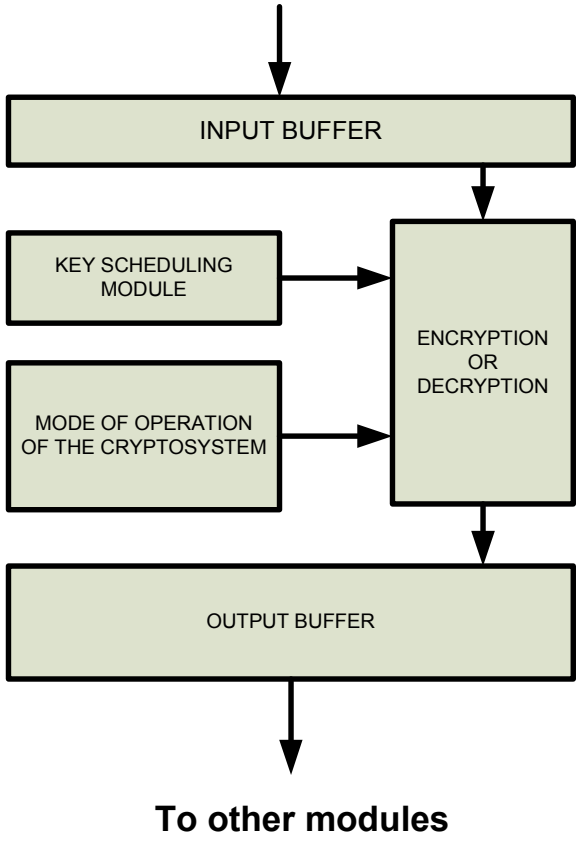

Figure 3. Hardware implementation of a Cryptosystem.

By implementing a cryptosystem in hardware, we can embed it as a separate encryption/Decryption module such that no extra overhead related to data security is required by the other modules.

A. Hardware Implementation of IDEA.

The IDEA cipher has three separate units other than key generation module. The performance and speed of hardware implemented IDEA depends on these three modules. These modules are:

- Multiplier Module.
- Addition unit.
- Inverse modulo Multiplier.

Among these modules, the main component which controls the speed and performance of IDEA is the modulo $(2^n + 1)$ multiplier module. It consumes the major portion of the clock cycles required by the entire algorithm. The modulus used in the multiplication is a Fermat prime

which is $(2^{16} + 1)$. One important thing here is that the operand 0 is treated as 2^{16}. The implementation of this multiplier in hardware is the most difficult task because the word length of the operands is comparatively large and implementing the multiplication sequentially is really time consuming.

B. *Multiplication Modulo* $(2^n + 1)$

Multiplication modulo a Fermat prime (p) is used as an important operation in many algorithms and it is crucial in various applications like pseudorandom number generation, Arithmetic processing and Cryptography [8]. In IDEA algorithm, this modulo multiplier plays a very important role in the throughput and speed. In general, a modulo multiplier consists of two stages, Multiplier module and modulo reduction. This paper mainly deals with the various multiplication schemes that have been implemented along with some newly proposed schemes.

The multiplier module is the stage where two binary n bit numbers are multiplied to form a product of 2n bits. The modulo reduction stage produce the product modulo the Fermat prime number and the final output becomes an n bit number. Various implementations have been done on this multiplier module so as to improve the efficiency of the cryptosystem. The most crucial part of multiplying two binary numbers is the generation of partial products. A lot of problems arise when two numbers are multiplied in a straightforward approach in hardware. As per human nature of calculation, when any expression is given as

$$xy \bmod (2^n + 1) = z' \bmod (2^n + 1) = z$$

At first the product is calculated by traditional method and then the modulo reduction is done by iterative subtraction method until the value falls under the range of 0 and 2^n.But the drawbacks of this implementation is inefficient use of silicon area which increases the hardware cost and it is time consuming.

Various techniques have been implemented before for multiplying two binary numbers of n bits where n is comparatively large.

- One of the implemented methods is the look-up table method where the two numbers to be multiplied forms an address in a table and in that address the product of the two numbers is to be stored. This method is very inefficient in terms of storage requirements as it takes a memory space of $2^{2n} \times n$ bits for n bit numbers.
- Another implemented approach for binary multiplication is the sequential method of shift and add technique. A binary 1 in multiplier just adds the multiplicand with the partial product and a value 0 does nothing except shifting the partial product. This technique requires n iterations for an n bit number and thus the performance depends on the number of bits in the multiplier. This approach is efficient when there is a long sequence of 0's in the multiplier.
- A slightly more efficient algorithm is the Booth's multiplication and its modified scheme. This approach is also used for signed 2's complement multiplication. Here instead of checking every individual bits of multiplier, the checking is done on bit pair for judging whether the multiplier has sequences of 0's or 1's.
- For fast implementations, Wallace trees are used. Wallace tree is an interconnection of carry save adders where the main objective is to reduce the number of partial products. In the last stage carry save adders are used for fast addition.
- For further increase of throughput, parallelism is exploited both in temporal and spatial aspects. In the pipelined approach, the advantage is that it takes sufficiently less time for giving the results but increases area consumption in slices. In the parallel approach, the multiplicand and the multiplier bits are processed parallel and there is no iterative addition of partial products. So in this case also, the time taken is much less but the design complexity is more

C. *Proposed multiplier scheme and architecture*

In this paper, the multiplier proposed is different from the previous implemented approaches. It uses a divide and conquer strategy where it divides the entire problem and each sub-problems are solved recursively. The basic strategy of this design is shown in Figure 4. This is new approach [2] for multiplication which is incorporated in IDEA hardware implementation. This algorithm is special in the sense that instead of having n additions for n bit multiplications; it is having $\log(n)$ additions. The basic idea is discussed below.

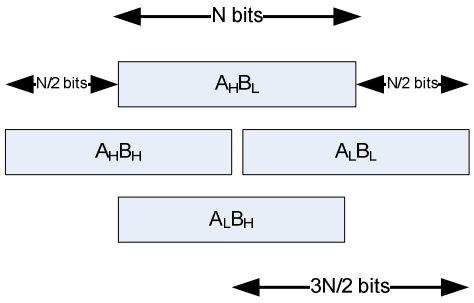

Figure 4: Divide and Conquer strategy for multiplying two N bit numbers.

If A and B are two N bit binary numbers and if they are split into two halves as A_HA_L and B_HB_L respectively, four N/2 bit multipliers can be used to calculate four partial products which are A_HB_H , A_LB_L , A_HB_L , A_LB_H . Then the partial products can be arranged as shown in Figure 4. This is one step of breaking N X N bit multiplication into $N/2 \times N/2$ bit multiplication. These $N/2 \times N/2$ bit multipliers can be further degraded into four $N/4$ bit multipliers in the same way. So in general way we are breaking a problem

into a number of sub problems and each such sub problems are solved recursively. This is Divide and Conquer strategy. The basic idea of this recursive approach [2] is to break the larger number into smaller numbers and multiply them in parallel and then combine their result to get the final product.

If two unsigned binary numbers are represented as

$$A = \sum_{i=0}^{n-1} a_i 2^i \text{ and } B = \sum_{i=0}^{n-1} b_i 2^i$$

And if

$$A_1 = \sum_{i=n/2}^{n} a_i 2^i \text{ and } B_1 = \sum_{i=n/2}^{n} b_i 2^i$$

$$A_0 = \sum_{i=0}^{n/2-1} a_i 2^i \text{ and } B_0 = \sum_{i=0}^{n/2-1} b_i 2^i$$

…… (1)

Such that

$$A = A_1 2^{n/2} + A_0$$

$$B = B_1 2^{n/2} + B_0 \text{ ……... (2)}$$

Then

$$AB = (A_1B_1)2^n + (A_0B_1)2^{n/2} + (A_1B_0)2^{n/2} + A_0B_0$$

…….. (3)

So from (3), we can conclude that multiplication of two n bit numbers involves four n/2 bit multiplications and addition of four results.

However the addition $(A_1 * B_1)* 2^n + (A_0 * B_0)$ takes no time as there is no overlapping of digits between them. So it requires only concatenation of two n bit sub-products. The block diagram of N bit recursive fast multiplier [2] is shown in Figure 5.

Base condition [2] can be anything in this context. The original numbers can be divided into small bit numbers until they are small enough to be multiplier directly. Here the approach is to break them into 2 bit multiplications using logic gates. Suppose the two 2 bit numbers are a_1a_0 and b_1b_0.

Using logic gates, the final product $P_3P_2P_1P_0$ can be written as

$$P_3 = (a_1b_1)(a_0b_0)$$

$$P_2 = (a_1b_1)(a_0' + b_0')$$

$$P_1 = ((a_1b_0)(a_0' + b_1')) + ((a_0b_1)(a_1' + b_0'))$$

$$P_0 = (a_0b_0)$$

This above multiplier contains 12 two input logic gates.

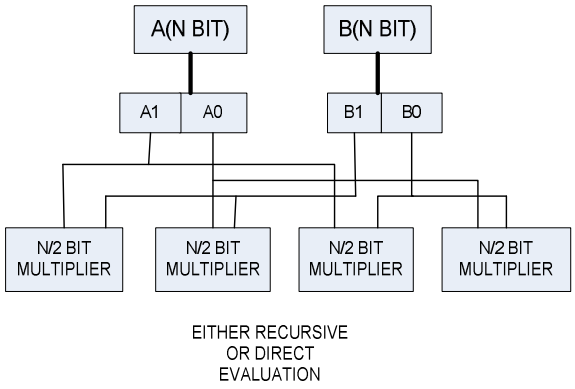

Figure 5. Block diagram of N bit recursive fast multiplier

After calculating the final product of the two numbers, the modulo reduction is performed. This is done using Low High Lemma introduced by Lai and Massey [1]

$$(xy) \bmod (2^n + 1) = (xy) \bmod 2^n - (xy) div 2^n$$

If $(xy) \bmod 2^n \geq (xy) div 2^n$

Else if $(xy) \bmod 2^n < (xy) div 2^n$

$$(xy) \bmod (2^n + 1) = (xy) \bmod 2^n - (xy) div 2^n + 2^n + 1$$

Here $(xy) \bmod 2^n$ corresponds to the least significant bits of xy and $(xy) div 2^n$ corresponds to right-shift of xy by n bits. One thing to note is that $(xy) \bmod 2^n = (xy) div 2^n$ implies that the value of $(xy) \bmod (2^n + 1) = 0$ but it is not possible as $2^{16} + 1$ is a prime number.

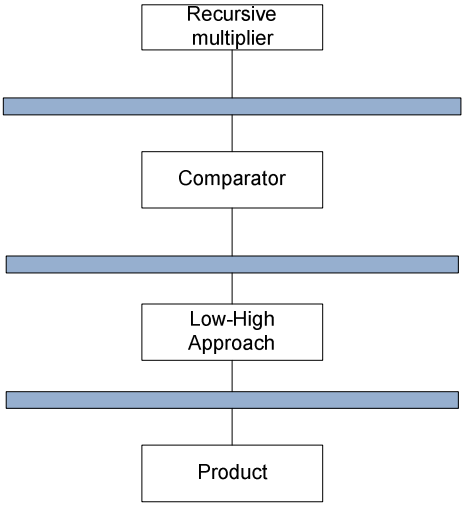

Figure 6. Pipelined architecture of the proposed multiplier.

For increasing the throughput of the design, a three stage pipeline is used in the original modulo multiplier. To do this, pipelined registers are incorporated inside the design which is driven by clocks. The pipelined architecture is shown in Figure 6

V. RESULTS AND OBSERVATIONS

The performance parameters [10] which are to be accounted for implementing IDEA in hardware are:

- Throughput or data conversion rate: It is taken as an important tool for measuring the timing performance of IDEA i.e. the amount of data processed and encrypted per unit of time.
- Area Requirements: It can be reported either in terms of number of Look-Up Tables (LUTs) or number of slices.
- Maximum Clock frequency: It is the maximum clock frequency that can be achieved by the design.
- Latency: It denotes the delay i.e. the time taken for the input data to move to the output port. For a pipelined design, the latency is measured by product of delay of a single pipelined stage and the number of pipelined stages.

The proposed multiplier is synthesized using VHDL and the synthesis report is given in Table I.

TABLE I

Preferred FPGA Device	Virtex2P-XC2VP40-FG676-7
Number of slices used	264
Number of internal multiplier used	4
Maximum clock Frequency	1099.505 MHz
Minimum input arrival time before clock	15.153 ns
Maximum output required after clock	8.135 ns
Size of bits processed	16

Table I shows the data corresponding to the implemented multiplier and the corresponding test bench waveform is shown in Figure 7. From this Table I, the throughput can be calculated as

$$Throughput = \frac{Allowed\_frequency(f) \times number\_of\_bits}{Number\_of\_clock\_cycles}$$

So based on the operational frequency (f), the throughput will be different.

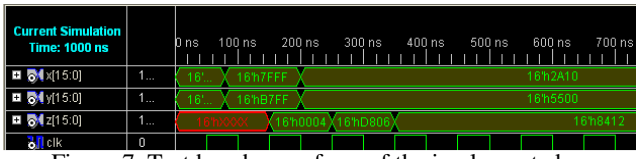

Figure 7. Test bench waveform of the implemented multiplier

VI. CONCLUSION

This paper gives a thorough study of the previous implemented schemes of the IDEA block cipher in hardware. Finally, a new multiplication scheme has been proposed which follows a divide and conquers strategy to get the final result. The proposed design is implemented and synthesized using VHDL and it is found that it works faster than traditional multiplication algorithm. For 16 bit numbers, the decomposition can be made as 16 => 8 => 4 => 2. So instead of taking N additions, this algorithm now takes log (N), N bit additions as a whole. Although the IDEA algorithm is not implemented as a whole using this multiplication algorithm, it is expected to give a comparatively better result that the sequential approach. The implementation is partial in the sense that only the modulo $(2^n + 1)$ multiplier, which is one of the main module of this IDEA block cipher, is tested. The other modules are yet to be tested as a whole for testing the entire encryption-decryption process. The concept of modified Booth's recoding is also planned to be incorporated in future work. The performance of the cipher using this scheme is yet to achieve.

REFERENCES

[1] X.Lai and J.L Massey "A Proposal for a New Block Encryption Standard," in advances in Cryptology – EUROCRYPT 90,Berlia,Germany: Springer Verlag pp. 389-404, 1990.

[2] Albert N.Danysh, A Recursive Fast Multiplier, Signals, Systems & Computers, 1998, pp.197-201

[3] Tsoi Kuen Hung,Leong," Cryptographic Primitives on Reconfigurable Platforms", PhD thesis,The Chinese University of Hong Kong,2002.

[4] R.Zimmermann,A.Curiger,H.Bonnenberg,H.Kaeslin,N.Felher,W.Fitchner,"A 177 Mb/s VLSI Implementation of the International Data Encryption Algorithm", IEEE Journal of Solid State Circuit,vol.29,110.3,pp.303-307,March 1994.

[5] Thaduri,M.,Yoo,S. and Gaede,R, " An Efficient Implementation of IDEA encryption algorithm using VHDL", ©2004 Elsevier.

[6] P. Kitsos *, N. Sklavos, M.D. Galanis, O. Koufopavlou , "64 Bit Block ciphers: Hardware Implementations and Comparison analysis",593-604,3rd November,2004,Elsevier.

[7] Stefan Wolter,Hogler Matz,Andreas Schubert and Ruiner Laur, " On the VLSI Implementation of International Data Encryption Algorithm.", © IEEE 1995.

[8] Curiger,Bonnenberg and Kaeslin,H., "Regular VLSI Architecture for Multiplication Modulo ($2^n + 1$) .",IEEE Journal of Solid State Circuits,vol.27,NO. 7,July 1991,pp 990-994.

[9] Bruce Schneier,"Applied Cryptography", 2nd Edition,Wiley publications.

[10] Francisco Rodríguez-Henríquez, N. A. Saqib, A. Díaz-Pèrez, and Cetin Kaya Koc. 2006.*Cryptographic Algorithms on Reconfigurable Hardware (Signals and Communication Technology)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[11] Helger Lipmaa. Idea: A cipher for multimedia architectures, In Selected Areas in Cryptography 98, pages 248{263. Springer-Verlag, 1998.

[12] O. Mencer, M. Morf, and M.J. Flynn. Hardware software tri-design of encryption for mobile communication units. In Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on, volume 5, pages 3045 –3048 vol.5, May 1998.

[13] H. Bonnenberg, Andreas Curiger, Norbert Felber, Hubert Kaeslin, and Xuejia Lai. Vlsi implementation of a new block cipher. In Proceedings of the 1991 IEEE International Conference on Computer Design on VLSI in Computer & Processors, ICCD '91, pages 510–513, Washington, DC, USA, 1991. IEEE Computer Society.

[14] Stefan Wolter,Hogler Matz,Andreas Schubert and Ruiner Laur, " On the VLSI Implementation of International Data Encryption Algorithm.", © IEEE 1995.