Area efficient ASIC implementation of IDEA (International Data Encryption Standard)

Onur Tigli, *Member, IEEE, The George Washington University*

Abstract— An area-efficient ASIC implementation of the secret-key cipher IDEA (International Data Encryption Standard) is presented. The design is implemented using RTL design techniques by employing Verilog HDL for both design and verification. Logic synthesis is completed by applying several different ASIC synthesis techniques to achieve the lowest possible area and the highest possible throughput. The design is based on a bit parallel approach in the submodules and a pipelined approach in the main encryption module. Basic building blocks such as adders and multipliers were designed for the lowest area/highest throughput principle. With a system clock frequency of 10 MhZ the designed device permits a data conversion rate of 193.9 Mbits/sec.

Index Terms—IDEA, ASIC, RTL, Cryptology, Verilog HDL

I. Introduction

THE Data Encryption Standard (DES) algorithm has been a popular secret key encryption algorithm and is used in many commercial and financial applications. Although introduced in 1976, it has proved resistant to all forms of cryptanalysis [1]. However, its key size is too small by current standards and its entire 56-bit key space can be searched in an approximately 22 hours [2].

Lai and Massay introduced an iterated block cipher known as Proposed Encryption Standard (PES) [3]. Later an improved version of PES was introduced with the name IPES, which improved the security of the original algorithm against differential analysis. In 1992, IPES was commercialized and was renamed the International Data Encryption Algorithm (IDEA).

Hardware implementations of IDEA offer significant speed improvements over software implementations by exploiting parallelism among operators. In addition, they are likely to be cheaper, having lower power consumption and smaller footprint [1]. This paper summarizes a design that was initiated by this comparison in mind, and exploits the idea of a single-chip ASIC solution that provides high speed as well as the lowest possible area. Several different approaches were taken to designing such a dedicated chip. The first VLSI implementation of IDEA was developed and verified by Bonnenberg et. Al. In 1992 using a 1.5 um CMOS technology [4]. This implementation had an encryption rate of 44 Mb/sec.

In 1994, VINCI, a 177 Mb/sec VLSI implementation of the IDEA algorithm in 1.2 um CMOS technology, was reported by Curiger et.al. [5,6]. A 355 Mb/sec implementation in 0.8 um technology of IDEA was reported in 1995 by Wolter et. al. [7] followed by a 424 Mb/sec single chip implementation of 0.7 um technology by Salomoa et. al. was reported.

The work that is described in this paper was carried up to the logic synthesis stage as proposed in the specifications work. That means, the synthesis runs were carried out based on the libraries that were available at the time of the design. Namely, Lsi_10k synthesis libraries were utilized. The details of the library will be listed down in the relevant sections. Although a rough area calculation can be done based on the library attributes and a previously constructed standard cell library dimensions, the figures obtained for comparison to the previously reported work can be misleading as the numbers being used are based on pre-layout wire-load models and logic library cell descriptions. Therefore, more effort was placed on correct functionality of the design rather than physical implementation and this was achieved by applying exhaustive verification and synthesis techniques.

A complete working design was developed by using Verilog HDL. The design is a portable design and and verified fully for its functionality. Although the work was carried out for ASIC implementation, the entirely synthesizable code can easily be transferred into an FPGA environment.

II. IDEA Algorithm overview

IDEA is a secret-key cipher whose decryption and encryption processes are symmetric. The cipher IDEA is an iterated cipher consisting of 8 rounds followed by an output transformation. It takes 64-bit plaintext inputs and produces 64-bit ciphertext outputs by using a 128-bit key.

The major concept behind IDEA is to mix operations from three different algebraic groups. Three different group operations work on pairs of 16-bit subblocks. They are:

- Bitwise XOR of two 16-bit subblocks
- Addition of integers modulo 2^{16} where the 16-bit subblock is treated as the usual radix-two representation of an integer

- Multiplication of integers modulo $2^{16}+1$ where the 16-bit block is treated as the usual radix-two representation of an integer except that the all-zero subblock is treated as representing 2^{16}

Figure 1 shows a diagram of IDEA. The 64-bit data block is divided into four 16-bit subblocks, which are X1, X2, X3 and X4. These four subblocks are the input to the first round of the algorithm. As stated earlier, there is eight of the same round in total. In each round three of the operations are used and between each round the second and third subblocks are swapped. In the end, one more extra output transformation is applied which simply takes the outputs of he 8th round without any swapping and passes them through the operation of the first four blocks. The output of this last transform gives the ciphertext for the IDEA algorithm. The decryption process is essentially the same as the encryption process except that the subkeys are derived using a different algorithm.

The algorithm for computing the encryption subkeys is straightforward. The only operation that takes place during key scheduling is basic logical rotation. The algorithm uses 52 subkeys with the distribution of; 6 for each of eight rounds and 4 for the output transformation. First the 128-bit key is divided into eight 16-bit subkeys. These are the first eight subkeys for the algorithm (the six for the first round, and the first two for the second round.) Then, the key is rotated 25 bits to the left again divided into eight subkeys. The first four are used in round two; the last four are used in round three. The key is rotated another 25 bits to the left for the next eight subkeys, and so on until the end of the algorithm.

Decryption is the same, except that the key subblocks are reversed and slightly different. The decryption key subblocks are either the additive or multiplicative inverses of the encryption-key subblocks. (Note that this work only covers the encryption of the IDEA algorithm.)

IDEA can work within any block-cipher mode that works with DES algorithm: Electronic Code Book (ECB), Cipher Block Chaining (CBC), Output Feedback (OFB), and Cipher Feedback (CFB). IDEA's key length is 128 bits over twice as the DES key. Assuming that testing every possible key (brute-force attack) is the most efficient way to break the algorithm, it would require 2^{128} encryptions to recover the key. DES only requires 2^{56} encryptions to break. IDEA is much stronger to brute force attack.

III. Building Blocks

Before implementing the overall architecture the basic building blocks were implemented and tested for correct operation. The implementation of the XOR and addition operations were straightforward, whereas the structure of a modulo $2^{16}+1$ multiplier proved to be the major challenge both in terms of area and timing. The design approach will be described in a bottom-up hierarchical fashion. Figure 2 depicts

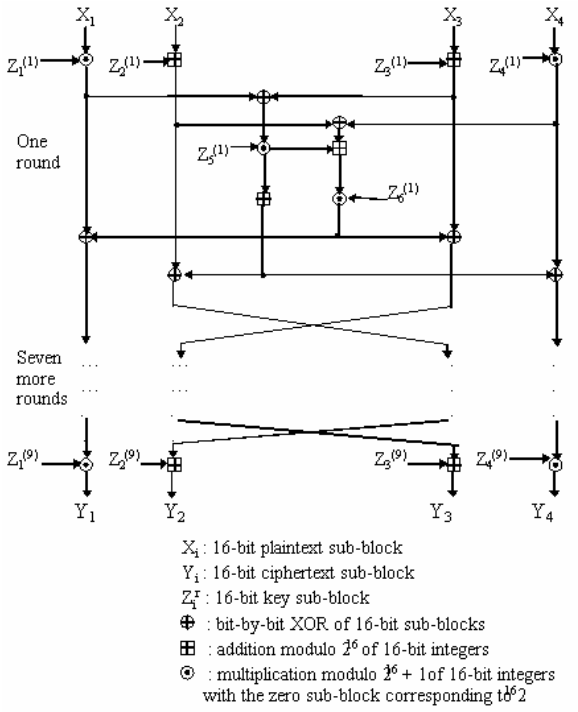

Fig. 1. The IDEA algorithm. Note that the output transformation is the same as the regular first step of each round.

this hierarchy, which was not only used for design development but also defined the synthesis approaches.

A. Multiplication Modulo $2^{16}+1$

The time required for completing this operation and the area that this unit will have a significant influence on the overall

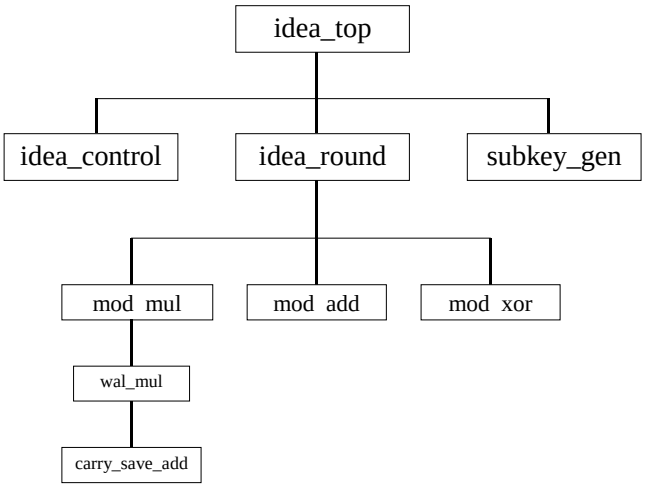

Fig. 2. The hierarchy of the modules that was used to construct the design and carry out the synthesis.

performance of the design. Various methods of implementing such a multiplication unit have been investigated and compared. As further discussion of each method is out of the scope of this report only the one that had been implemented is discussed. Further details can be found in [8].

As a result the algorithm that was proposed in Meier and Zimmerman [9] which uses modulo 2^n adders with bit-pair

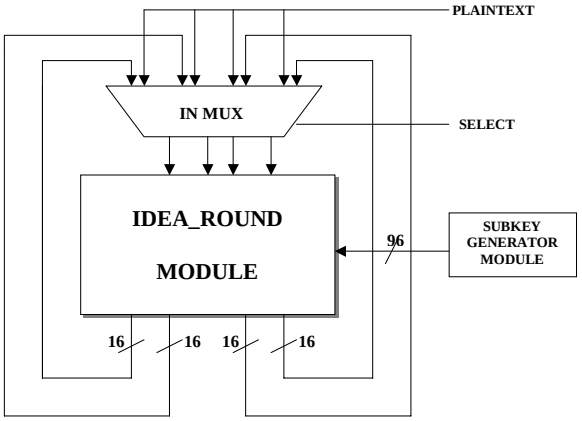

Fig. 3. The datapath of the entire design that contains the idea_round module the subkey generator module and the input multiplexer

```
assign AD=A-1;
assign BD=B-1;

//module instantiation
//wallace multiplier

walmul_16 w1(.A(AD),.B(BD),.C(CD));

assign result=CD+AD+BD+1;

always @ (result)

begin
if(result_low<=result_high)
C<=(result_low-result_high)+1;
else
C<=(result_low-result_high);
end

endmodule
```

This algorithm requires a total of six additions and subtractions, one 16-bit multiplication and one comparison. Therefore, the major component that defines the performance of this module is the multiplier. Among different multiplier structures, Wallace multiplier architecture was picked and implemented. When compared to other multipliers such as Booth, Wallace is superior due to its ability to perform multiplication in one pass/cycle. It's purely combinational and therefore doesn't introduce extra cycles to complete the operation, as is the case in Booth. The Wallace multiplier utilizes adders for calculating stage-by-stage multiplication results. In the Wallace multiplier design several adder units are instantiated, which have various input/output widths. For this reason, a parametrizable carry save adder was designed and this module was instantiated several times with different widths in the Wallace multiplier.

B. Addition modulo 2^{16}

Addition in modulo 2^{16} requires no special approach as in the multiplication. The same idea that was employed in the design of the carry save adder subblock was used to develop the modulo adder unit The only major modification was made in recognizing the output. The last stage carry is discarded and the first 16 bits constitute the sum. Following is the Verilog code for the mod_adder unit:

```
module mod_add(sum,a,b);

parameter WIDTH=16;

output [WIDTH-1:0] sum;
input  [WIDTH-1:0] a;
input  [WIDTH-1:0] b;

integer i;
reg   [WIDTH-1:0] sum;
reg   [1:0] result;
reg   [WIDTH:0]c ;

always@(a or b)
begin
  c[0]=0;
        for(i=0;i<WIDTH;i=i+1)
        begin
           result=a[i]+b[i]+c[i];
           sum[i]=result[0];
           c[i+1]=result[1];
        end
end
endmodule
```

C. Architecture

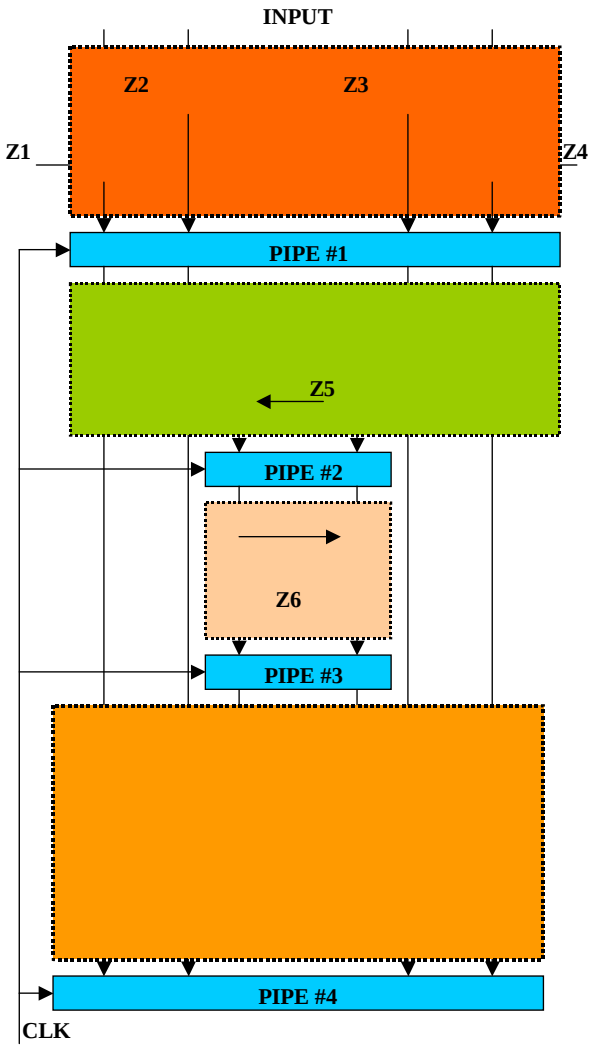

Fig. 4. The 4-stage pipelined implementation of the idea_round module

Based on this pipelined design, the critical path, which essentially determines the speed of the entire device, turns out to be the third stage block. This is due to the fact that, an addition is followed by a multiplication which results in

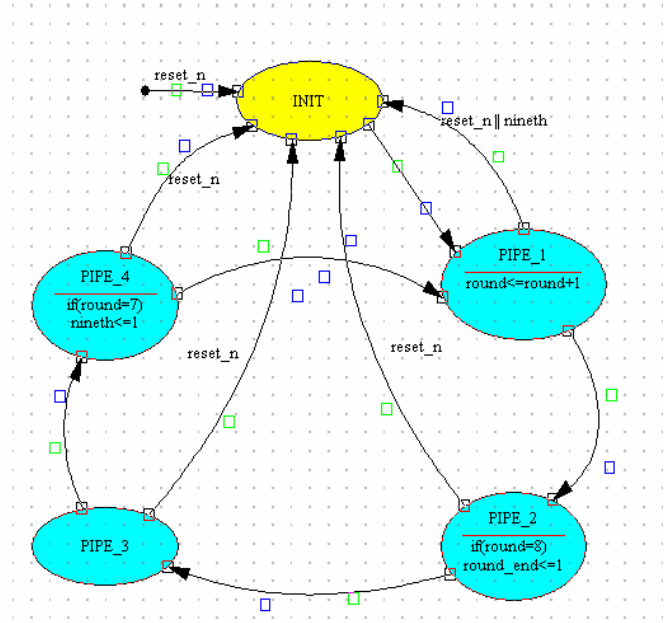

Fig. 5. The ASM diagram of the control module. Note that only some of the most pertinent signals are listed for convenience.

addition of both of the combinational delays. In other stages the largest delay is either the multiplication or the addition delay depending on the operations that take place.

E. Subkey_generator module

Subkey generator is one of the most important modules as it is responsible of providing the correct subkeys with correct

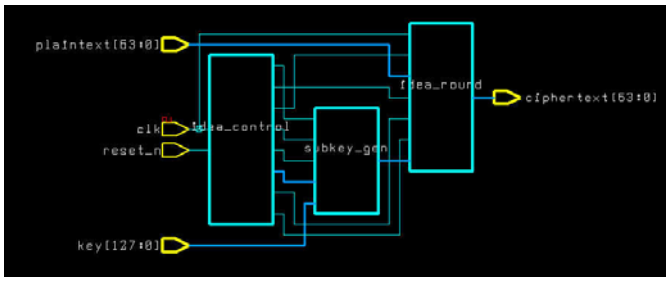

Fig. 6. The top module that instantiates and connects the major blocks: idea_control, idea_round, subkey_gen

timing. In order to achieve this a two register design was established. One register holding the current round keys and the second one holding the shifted version of the keys. As it was explained in the algorithm section, for certain rounds the idea_round module will require keys both from the current snapshot of the key register and from the shifted version of it. The subkey generator simply accepts three clocking signals from the control logic in order to achieve this timing. Once all of the keys are used for the current round – which is not until the end of the third stage in the pipeline- the shift operation takes place and the following round subkeys are prepared within the last clock cycle of the current round.

F. Control logic

The control logic is a simple ASM with five discrete states. One of them is the initialization step at which entire device is reset and prepared for encryption cycles to start. Once this state is passed the encryption cycles start. As described earlier, there are four stages in the pipeline, hence four states – excluding the initialization- in the control logic. The control logic visits each of these states once and generates the relevant control signals, which triggers the idea_round and the subkey_gen modules to complete their corresponding operations. Each round of idea takes 4 clock cycles (4 states). There are 8 rounds in the entire encryption. This requires 4x8=32 clock cycles. However, after the last cycle of the eight round, one more clock cycle is required to complete the output transform. Therefore, the ciphertext is ready 32+1=33 clock cycles after the initialization is entered. The output ciphertext is displayed on the output bus for one lock cycle and the control logic returns automatically back to the initialization state and sits to wait for a new plaintext input to be encrypted. Throughout the calculation period (=33 clock cycles) the output is set to be high impedance. The output bus is driven only when the calculations are complete. Figure 5 shows a simplified ASM diagram of the control logic.

After all of the major blocks, namely idea_round, subkey_gen and the control, were designed and tested for correct operation a top module that instantiates all of these as well as the other subblocks were designed. Figure 6 shows the top module snapshot that was obtained in the end of the synthesis process.

IV. VERIFICATION

The verification strategy was completed as proposed in the specifications. Each block that are described in the previous section was tested for correct operation beginning with the lowest level in the hierarchy. Individual testbenches were developed in Verilog that traverses through all possible input/output combinations; simulated using Verilog XL simulator and the results were examined by using DAI SignalScan waveform viewer for correctness.

After the verification of every block that had been designed, the major test was to be carried out on the top module, which defines the entire design. In order to complete this test, a publicly available C code that implements IDEA was found. This code was then modified in order to accommodate user entry for keys and plaintexts. Also, a facility to display the outputs was added to the code. Then this modified code was used to obtain several sets of key, plaintext, and ciphertext arrays. These values were compared with the results obtained from the Verilog simulations with the same inputs. The design passed all of these tests and functionality was completely verified.

V. SYNTHESIS

Once the functionality of the design was verified, the design

TABLE 1

THE RESULTS OBTAINED FROM THE BOTTOM-UP HIERARCHICAL LOGIC SYNTHESIS BY USING LSI_10K LIBRARY **WORST** CASE OPERATING CONDITIONS ATTRIBUTES.

Module	*Worst path (largest delay)*	*Best path (smallest delay)*	*Non Combin. Area*	*Combin. Area*
Carry_save_add	10.97 ns	4.88 ns	-	231
Wallace_mul	47.07 ns	26.56 ns	-	5,651
Mod_mul	82.01 ns	23.24 ns	-	7,973
Mod_add	36.28 ns	8.35 ns	-	420
Mod_xor	8.37 ns	8.07 ns	-	64
Idea_round	97.86 ns	2.79 ns	3,244	31,178
Subkey_gen	9.22 ns	3.88 ns	1,111	4,064
Idea_control	8.51 ns	5.57 ns	81	47
Idea_top	**105.24 ns**	**2.38 ns**	**7,390**	**32,340**

TABLE 2

THE RESULTS OBTAINED FROM THE BOTTOM-UP HIERARCHICAL LOGIC SYNTHESIS BY USING LSI_10K LIBRARY **BEST** CASE OPERATING CONDITIONS ATTRIBUTES.

Module	*Worst path (largest delay)*	*Best path (smallest delay)*	*Non Combin. Area*	*Combin. Area*
Carry_save_add	6.81 ns	6.38 ns	-	306
Wallace_mul	16.54 ns	10.82 ns		5,682
Mod_mul	26.31ns	8.48 ns	-	7,941
Mod_add	13.38 ns	6.59 ns	-	475
Mod_xor	5.92 ns	5.84 ns	-	64
Idea_round	28.24 ns	0.76 ns	3,240	32,636
Subkey_gen	1.06 ns	0.88 ns	4,064	1,111
Idea_control	6.28 ns	1.48 ns	78	48
Idea_top	**29.46 ns**	**0.77 ns**	**7,382**	**33,795**

is taken into Synopsys design analyzer suite for logic synthesis. At this stage, once again separate synthesis scripts were developed for each of the 9 major modules. These modules were optimized based on mainly minimum area constraints and secondly timing constraints. Bottom-up hierarchical synthesis technique was employed based on the hierarchy diagram depicted in Figure 2. In this technique, every module is synthesized individually with their pre-determined constraints, optimized and saved with the constraints intact. Then these optimized blocks are read in and linked on the higher hierarchy level. The synthesis engine is set to optimize the top level and doesn't try to reoptimize the lower level blocks. This is achieved by setting the don't_touch attribute on the lower level blocks once they are optimized and saved.

Table1 and Table2 lists down all the pertinent information that was obtained from the synthesis process. Note that the only library that was available was LSI_10k and it was used for all the static timing analysis and logical synthesis. All of the synthesis work was carried out by using two extreme cases namely, the worst-case library attributes and the best-case library attributes. Therefore, the obtained results reflect the span of the possible results that can be obtained if a physical implementation is sought.

VI. CONCLUSION

In this paper an area-efficient hardware implementation of the IDEA algorithm has been presented. 728 lines of synthesizable Verilog code was developed which can be used in any platform to synthesize the IDEA hardware. This design utilizes a bit-pair recoding algorithm for the multiplication operation and a four-stage linear pipeline architecture for the idea encryption rounds. The entire design was developed by using Verilog HDL and the verification process proved the design to be working correctly. The logic synthesis was also carried out and the results obtained based on the static timing analysis has been listed.

The device is capable of a data throughput of 19.39 Mbit/s (at 10 MHz) for the worst case operating condition attributes and a data throughput of 77.5 Mbits/s (at 40 MHz) for the best case operating conditions. The total combinational area is 7,390 unit area and the total non-combinational area is 32,340 unit area. A unit area is equivalent to the physical area covered by a 2-input NAND gate. Depending on the technology that is chosen for fabrication of the device, the actual value of the area can be calculated by multiplying the unit area values with the area covered by a 2-input NAND gate. Therefore, the unit area values give the equivalent gate count for the device.

The resulting device is not the fastest possible design. This is mainly due to the worst case operating conditions, which define the wire load models and the cell delays to be assigned extremely large values when compared to possible values that are available with current fabrication technologies. A best-case synthesis will also be run and the data obtained will be provided for comparison.

REFERENCES

[1] O.Y.H. Cheung, K.H. Tsoi, P.H.W. Leong, and M.P.Leong. "Tradeoffs in Parallel and Serial Implementations of the International Data Encryption Algorithm IDEA".

[2] Electronic Frontier Foundation. DES challenge III broken in record 22 hours, January 1999.

[3] X.Lai and J.Massay. A proposal for a new block encryption standard. In Advances in Cryptology, Proceedings of Eurocrypt 1990, pages 196-211, 1995

[4] H. Bonnenberg, A. Curiger, N.Felber, H.Kaeslin, and X.Lai. VLSI implementation of a new block cipher. In Proceedings of the IEEE International Conference on Computer Design:VLSI in Computer and Processors, pages 501-513, 1991.

[5] A.Curiger, H. Bonnenberg, R. Zimmerman, N.Felber, H. Kaeslin, and W.Fichtner. VINCI: VLSI implementation of the new secret-key block cipher IDEA. In Proceedings of the IEEE Custom Integrated Circuits Conference, pages 15.5.1-15.5.4,1993.

[6] R.Zimmerman, A.Curiger, H. Bonnenberg, H. Kaeslin, N.Felber and W.Fichtner. A 177 Mb/sec VLSI implementation of the International data encryption algorithm. IEEE Journal of Solid-State Circuits, 29 (3):303-307, March 1994.

[7] S.Wolter, H.Matz, A.Schubert, and R.Laur. On the VLSI implementation of the international data encryption algorithm IDEA. In Proceedings of the IEEE International Symposium on Circuits and Systems, volume 1, pages 397-400, 1995.
[8] A.Curiger, H. Bonnenberg, and H. Kaeslin. Regular VLSI-architectures for multiplication modulo (2^n+1). IEEE Journal of Solid-state Circuits, 26(7):990-994, July 1991.
[9] C.Meier and R.Zimmerman. A multiplier modulo (2^n+1). Diploma thesis, Institut fur Integrierte Systeme, ETH, Zurich, Switzerland, February 1991.
[10] http://www.cs.nps.navy.mil/curricula/tracks/security/notes/chap04_43.html
[11] http://strato.visgraf.impa.br/tron/Livros/books/book10/9312e/9312e.htm
[12] A. Curiger, H.Bonnenberg, R.Zimmerman, N.Felber, H.Kaeslin, and W.Fichtner. "VINCI:VLSI implementation of the new secret-key block cipher IDEA". IEEE Custom Integrated Circuits Conference, 1993.
[13] W.Stallings, "Cryptography and Network Secuirity: Principles and Practice, 3rd edition, Prentice Hall, Upper Saddle River, 2003.

Onur Tigli (M'97) Mr.Tigli was born in Istanbul, Turkey 1979. He received his BSc in Electrical and Electronics Engineering from the Middle East Technical University, Ankara Turkey, 2000 with a concentration on microelectronics and computer engineering. He started his graduate studies at The George Washington University, Electrical and Electronics Engineering Department and received his MS in Computer Engineering degree in 2002. During his masters he has worked for the National Institute of Standards and Technology (NIST) Semiconductor Electronics Division on system-on-a-chip project. He is currently pursuing his DSc degree at The George Washington University. His research interests include, ASIC design, MEMS and sensor readout design.