Hardware Implementation of the Stone Metamorphic Cipher

Rabie A. Mahmoud[1], Magdy Saeb[2]

1. Department of Mathematics, Faculty of Science, Cairo University, Egypt.

2. Computer Engineering Department, Arab Academy for Science, Tech. & Maritime Transport (AAST), Alexandria, Egypt.
mail@magdysaeb.net

ABSTRACT: *The Stone Cipher is a metamorphic cipher that uses a variable word size and variable-size user's key. The cipher employs two basic functions; the encryption function and a Pseudo Random Number Generator (PRG) that is based on a specially-developed one-way hash function. Four bit-balanced operations are pseudo-randomly selected to generate the sequence of operations constituting the cipher. These operations are: XOR, INV, ROR, NOP for bitwise xor, invert, rotate right and no operation respectively. The user key is encrypted using the cipher encryption function with agreed-upon initial values then it is used to generate the bit stream required to select these operations. In this work, we provide a Field Programmable Gate Array (FPGA) hardware implementation of this cipher.*

Keywords: *FPGA, Cipher, Metamorphic, Cryptography, Hardware.*

1. Introduction

The Stone Cipher is a metamorphic cipher that is hardware implemented utilizing Field Programmable Gate Arrays (FPGA). The idea of this cipher is to use four low-level operations that are all bit-balanced to encrypt the plaintext bit stream. These operations are: xoring a key bit with a plaintext bit (XOR), inverting a plaintext bit (INV), exchanging one plaintext bit with another one in a given plaintext word using a right rotation operation (ROR) and producing the plaintext bit without any change (NOP). In addition, the internal sub-keys are generated using a combination of the encryption function itself and a one-way hash function. The generated key stream is used to select the various operations. In the following sections, we provide the structure of the cipher, the formal description of its algorithm, the details of our circuit design, discussion of the results of the FPGA implementation and finally a summary and our conclusions.

2. The Stone Cipher Structure

The conceptual block diagram of the implemented cipher is shown below in Figure 1. It is constructed of two basic functions; the encryption function and the sub-key generating one-way hash function. The pseudo random number generator is built using the same encryption function and the MDP-384 [4] one-way hash function in cascade. Two large numbers (a, b) are used to iteratively generate the sub-keys. The details of the substitution box S-orb can be found in [5].

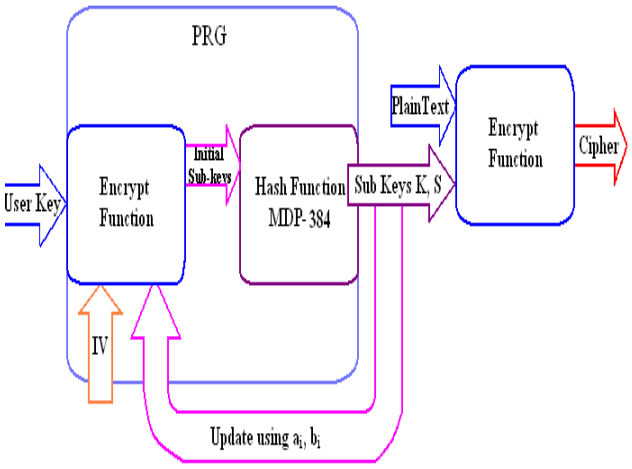

Figure 1. The structure of the cipher

The user key is first encrypted then the encrypted key is used to generate the sub-keys. The encryption function or the cipher engine is built using four low-level operations. Table1 demonstrates the details of each one of these operations.

Table 1: The basic cipher engine (Encryption function) operations

Mnemonic	*Operation*	*Select Operation code*
XOR	$C_i = K_i \oplus P_i$	00
INV	$C_i = \neg (P_i)$	01
ROR	$P_i \leftarrow P_i$	10
NOP	$C_i = P_i$	11

The basic crypto logic unit (CLU) is shown in Figure 2. All operations are at the bit level. The unit is to be repeated a number of times depending on the required word or block size. The rotation operation, referred to by the circular arrow which is performed using multiplexers and is shown in Figure 3. This CLU is used as the encryptor and the decryptor where by changing the output cipher bit to become an input plain text bit, the new output will be the same as the old plain text bit. Obviously, this is a feature of the applied functions namely XOR, INV or NOP. The only exception is in the case of ROR, the decryptor will use ROL [6].

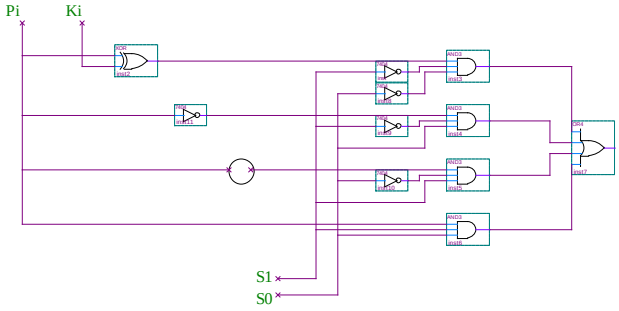

Figure 2. The basic crypto logic unit

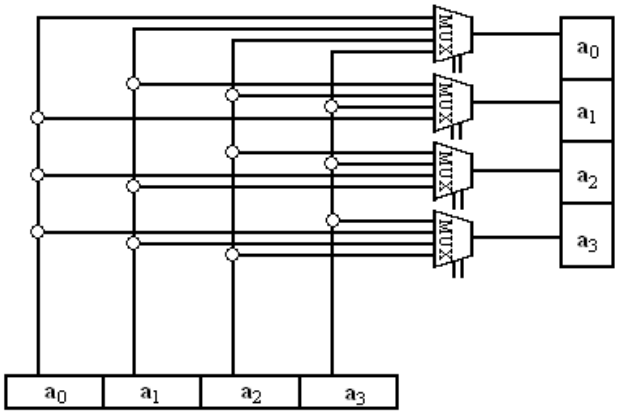

Figure 3. The rotation operation (ROTR) implementation using multiplexers

The operation selection bits (S$_1$ S$_0$) can be chosen from any two sub-key consecutive bits; as shown in Figure 4. The same idea applies for the rotation selection bits (S'$_1$ S'$_0$).

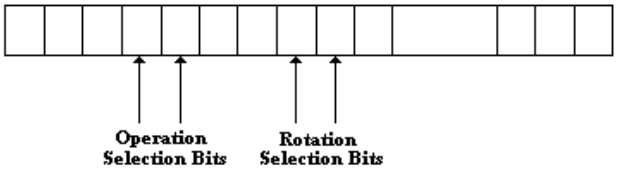

Figure 4. The proposed key format where the location of the selection bits is shown

3. The Algorithm

The formal description of the algorithm, as shown in [6], is summarized as follows:

Algorithm: STONEMETAMORPHIC
INPUT: *Plain text message P, User Key K,*
Block Size B
OUTPUT: *Cipher Text C*
Algorithm body:
Begin
Begin key schedule
1. *Read user key;*
2. Encrypt user key by calling encrypt function and using the initial agreed-upon values as the random input to this function;
3. Read the values of the large numbers a and b from the encrypted key;
4. Generate a sub-key by calling the hash one-way function;
5. Store the generated value of the sub-key;
6. Repeat steps 5 and 6 to generate the required number of sub-keys;
End key schedule;

Begin Encryption
7. Read a block B of the message P into the message cache;
8. Use the next generated 192-bit key to bit-wise encrypt the plain text bits by calling the encrypt function;
9. If message cache is not empty, Goto step 8;
10. Else if message cache is empty:
If message not finished
10.1 Load next block into message cache;
10.2 Goto 8;
Else if message is finished then halt;
End Encryption;
End Algorithm.

Function ENCRYPT
Begin
1. Read next message bit;
2. Read next key bit from sub-key;
3. Read selection bits from sub-key;
4. Read rotation selection bits from sub-key;
5. Use selection & rotation bits to select and perform operation: XOR, INV, ROR, NOP;
6. Perform the encryption operation using plaintext bit and sub-key bit to get a cipher bit;
7. Store the resulting cipher bit;
End;

4. FPGA Implementation

The simplicity and lucidity of the encryption function of the Stone Cipher lead to a relatively easy-to-design FPGA-based implementation. We have implemented the cipher using VHDL hardware description language [2], [3], [7] and Quartus II 9.1 Service Pack 2 Web Edition [1], and utilizing Altera design environment. The metamorphic cipher implementation is based on the idea of encrypting 256-bit plaintext blocks using 256-bit user key and eight rotation selection bits, producing 256-bit ciphertext blocks. Every 256-bit sub-keys are extracted from the 384-bit of the output PRG. The schematic diagram for a demonstrative 256-bit encryption module is shown in Figure 5. The design was implemented using an EP2C70F896C6, Cyclone II family device. The worst case pin-to-pin delay was found to be equal to 28.420 ns. The longest pin-to-register delay was 7.787 ns and the shortest pin-to-register delay was 6.925 ns. The longest register-to-pin delay was 23.215 ns. A series of screen-captures of the different design software outputs are shown in Figures 6 to 12. Figures 6, 7, 8, and 9 provide indication of successful compilation, parts of RTL for metamorphic cipher respectively which are shown a repeated MUX's in different connections. Figure 10 displays the encryptor simulation showing the output encrypted bits where the pins of input plaintext, output ciphertext, and the selection bits of sub-keys to specify the operation of encryption function are highlighted in the screen. Figures 11 and 12 demonstrate the floor plan and the timing report respectively.

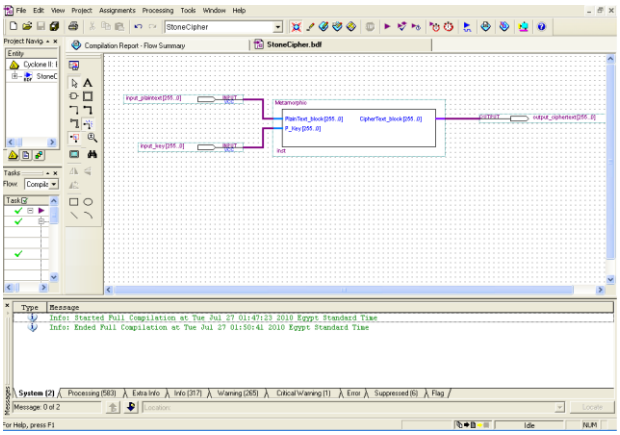

Figure 5. Schematic diagram of metamorphic cipher implementation

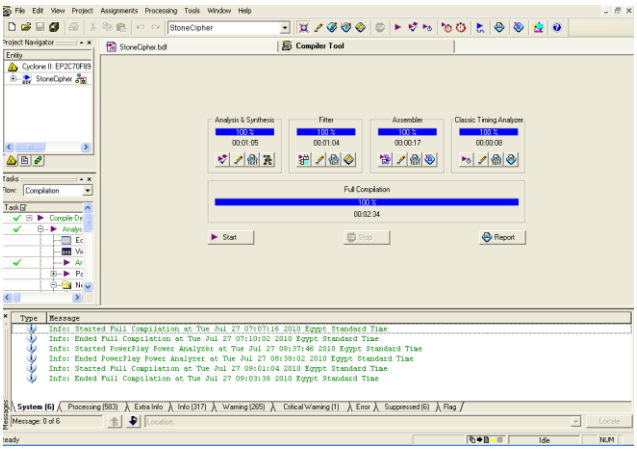

Figure 6. Compiler tool screen showing correct implementation

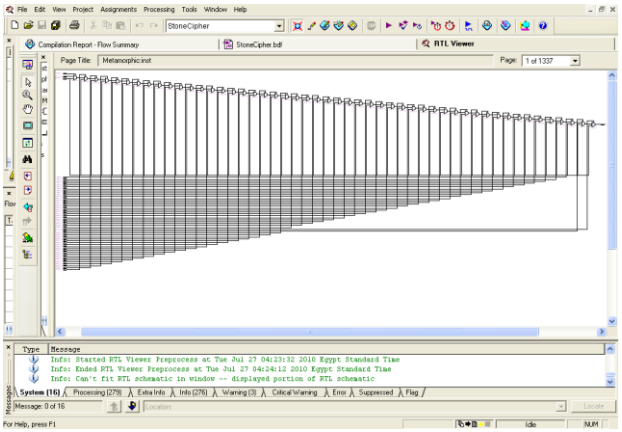

Figure 7. RTL screen for part of metamorphic cipher implementation

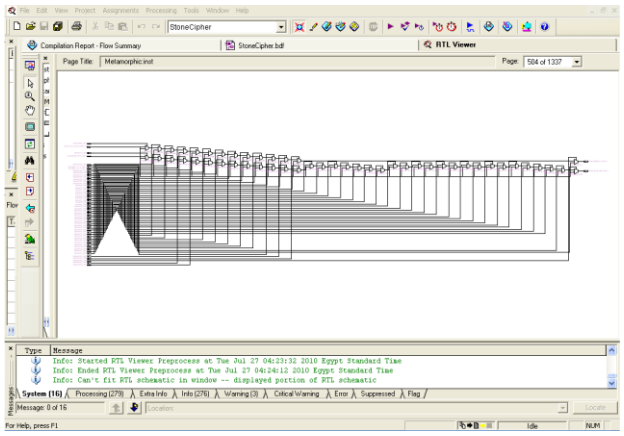

Figure 8. RTL screen for part of the resulting circuit diagram

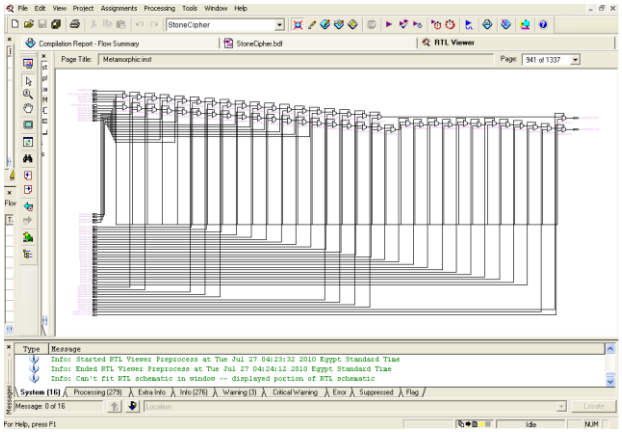

Figure 9. RTL screen for part of the cipher implementation

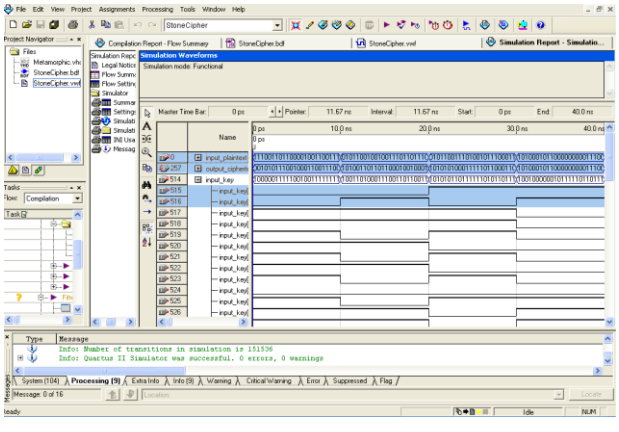

Figure 10. Simulator screen showing the output encrypted data based on the selection bits of user key

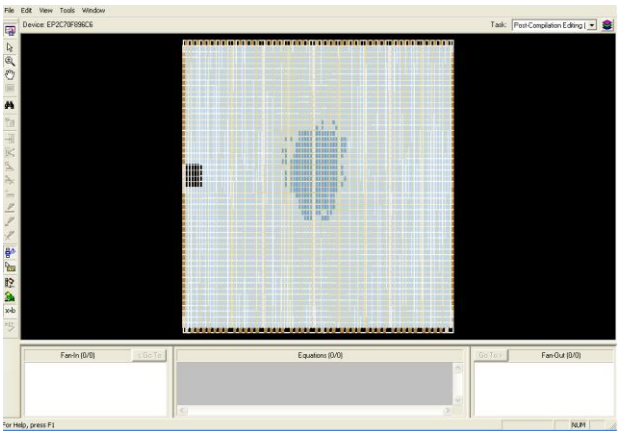

Figure 11. Floor-plan of cipher implementation

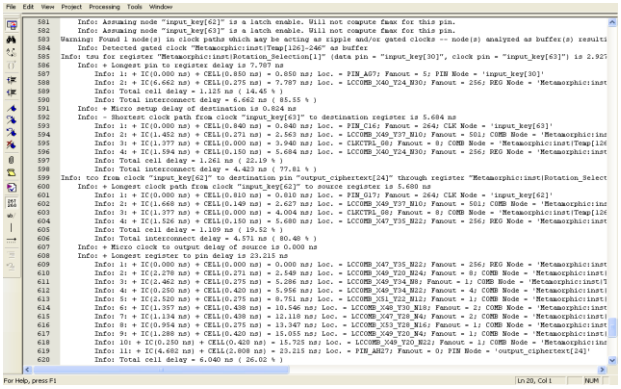

Figure 12. Timing report

The details of the analysis and synthesis report are shown in appendix B.

Summary & Conclusion

We have furnished a brief discussion of the hardware implementation of the Stone Metamorphic Cipher. Various modules were implemented applying VHDL and then joined together using the schematic editor. The resulting circuit provides a proof-of-concept FPGA implementation. It was shown that the worst case pin-to-pin delay is equal to 28.420 ns. Moreover, area and speed optimization were performed and it is shown that the worst case pin-to-pin delay is equal to 26.341 ns in the case of area optimization and 27.233 ns in speed optimization. One may inquire why the maximum delay in case of speed optimization is marginally greater than in the case of area optimization. This can be explained by noting that the speed optimization objective is to increase the maximum operating frequency (f_{max}) that is dependent on the register-to-register delay not pin-to-pin delay.

Moreover, high fan-out reduces the usage of global interconnection resources. Therefore, the speed optimization decreases the use of global resources. This is clearly demonstrated in the synthesis report of our design. A comparison with other implementations is not applicable since this is the first time the cipher is FPGA-implemented. This and other related issues will be dealt with in future development of the device.

References

[1] Altera's user-support site: http://www.altera.com/support/examples/vhdl/vhdl.html

[2] Enoch O. Hwang, "Digital Logic and Microprocessor Design with VHDL," La Sierra University, Riverside, California, USA, 2005.

[3] Pong P. Chu, "RTL Hardware Design Using VHDL," John Wiley & Sons, Inc., New Jersey, 2006.

[4] Magdy Saeb, "Design & Implementation of the Message Digest Procedures MDP-192 and MDP-384," ICCCIS2009, International Conference on Cryptography, Coding & Information Security, Paris, June 24-26, 2009.

[5] Magdy Saeb, "The Chameleon Cipher-192: A Polymorphic Cipher," SECRYPT2009, International Conference on Security & Cryptography, Milan, Italy, July, 2009.

[6] Magdy Saeb, "The Stone Cipher-192 (SC-192): A Metamorphic Cipher," The International Journal on Computers and Network Security (IJCNS), Vol.1 No.2, pp. 1-7, Nov., 2009.

[7] Volnei A. Pedroni, "Circuit Design with VHDL," MIT Press, 2004.

Appendix A:

Sample VHDL code for a four-bit encryption module

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;

ENTITY Metamorphic IS
  Port (PlainText : in std_logic_vector (3 downto 0);
        Key : in std_logic_vector (3 downto 0);
        CipherText: out std_logic_vector (3 downto 0));
END Metamorphic ;

ARCHITECTURE behavioral OF Metamorphic IS
 signal Temp: std_logic_vector (3 downto 0);
 signal Rotation_Selection : std_logic_vector(1 downto 0);
BEGIN

 process (PlainText, Key, Temp)
            variable R : integer range 3 downto 0;
 begin
   if    Key(3) = '0' and Key(2 )= '0' then
            Temp <= PlainText xor Key;

  elsif Key(3) = '0' and Key(2) = '1' then
            Temp <= not PlainText;

  elsif Key(3) = '1' and Key(2 )= '1' then
            Temp <= PlainText;

  elsif Key(3) = '1' and Key(2 )= '0' then
       Rotation_Selection <= Key(0)  &  Key(1) ;
       R := conv_integer (Rotation_Selection);

       if     R = 0 then Temp <= PlainText;
       elsif R = 1 then Temp <= PlainText(0)  &
                                PlainText (3 downto 1);
       elsif R = 2 then Temp <= PlainText (1 downto 0) &
                                PlainText (3 downto 2);
       elsif R = 3 then Temp <= PlainText (2 downto 0) &
                                PlainText(3);
        end if;
    end if;
 end process;
      CipherText <= Temp;
END behavioral;
```

Sample VHDL code for a MDP-1 module of hash function

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;

ENTITY MDP_1 IS
Port (input_to_hash: in std_logic_vector (31 downto 0);
       M_D_P_1: out std_logic_vector (383 downto 0));
END MDP_1;

ARCHITECTURE behavioral OF MDP_1 IS
    signal W0 : std_logic_vector (31 downto 0);

    signal K0 : std_logic_vector(31 downto 0) :=
                "01100000011100010100100110001111";

    signal a0 : std_logic_vector(31 downto 0) :=
                "01011110111111110100010111001100";
    signal b0 : std_logic_vector(31 downto 0) :=
```

```
                "00110110010010111101000001001100";
    signal c0 : std_logic_vector(31 downto 0) :=
                "00100011111001010000111001110000";
    signal d0 : std_logic_vector(31 downto 0) :=
                "01001100000010000001110010000000";
    signal e0 : std_logic_vector(31 downto 0) :=
                "00100011100110111110011111101001";
    signal f0 : std_logic_vector(31 downto 0) :=
                "00010100101101111111010010000000";

    signal a0, b0, c0, d0, e0, f0, a1, b1, c1, d1, e1, f1 :
                            std_logic_vector(31 downto 0);

BEGIN
W0 <= input_to_hash (31 downto 0);

a1 <= (a0(30 downto 0) & a0(31)) + ((a0 and b0) or ((not
      a0) and c0)) + ((c0 and d0) or ((not c0) and e0))
      + f0 + W0 + K0;
b1 <= (e0(26 downto 0) & e0(31) & e0(30) & e0(29) &
      e0(28) & e0(27));
c1 <= (a0(30 downto 0) & a0(31));
d1 <= (b0(29 downto 0) & b0(31) & b0(30)) xor (a0(30
      downto 0) & a0(31));
e1 <= (c0(28 downto 0) & c0(31) & c0(30) & c0(29)) xor
      (b0(29 downto 0) & b0(31) & b0(30));
f1 <= (d0(27 downto 0) & d0(31) & d0(30) & d0(29) &
      d0(28)) xor (c0(28 downto 0) & c0(31) & c0(30) &
      c0(29));

M_D_P_1 <=  a1 & b1 & c1 & d1 & e1 & f1 & a1 &
            b1 & c1 & d1 & e1 & f1;

END behavioral;
```

Appendix B: The analysis and synthesis report details:

Family: Cyclone II
Device: EP2C70F896C6
Total logic elements: 3085 out of 68,416 (5 %)
Total combinational functions: 3085
Logic element usage by number of LUT inputs
-- 4 input functions: 2562
-- 3 input functions: 517
-- <=2 input functions: 6
Dedicated logic registers: 0
Total memory bits: 0 out of 1,152,000 (0 %)
Embedded Multiplier 9-bit elements: 0 out of 300 (0 %)
Total PLLs: 0 out of 4 (0 %)
Optimization Technique: Balanced
Maximum fan-out: 1025
Total fan-out: 12067
Average fan-out: 3.30

Fitter Summary
Block interconnects: 4080 out of 197,592 (2 %)
C16 interconnects: 508 out of 6,270 (8 %)
C4 interconnects: 3337 out of 123,120 (3 %)
Direct links: 273 out of 197,592 (< 1 %)
Global clocks: 1 out of 16 (6 %)
Local interconnects: 1139 out of 68,416 (2 %)
R24 interconnects: 779 out of 5,926 (13 %)
R4 interconnects: 3708 out of 167,484 (2 %)
Nominal Core Voltage: 1.20 V
Low Junction Temperature: 0 °C
High Junction Temperature: 85 °C.

The usage number of logic elements and their connections in the device can be changed depending on the optimization technique which is used for synthezing the cipher. Table B1 shows the number of usage logic elements and the interconnections between them in Area, Speed, and Balanced optimization technique. We noticed that the number of usage logic elements in the device increased in speed optimization technique comparing with balanced optimization technique to provide more block, local, and direct interconnections and applied 3.34 average fan-out, while less consuming of logic elements in area optimization technique gives less average fun-out 3.26 comparing with average fun-out 3.30 in balanced optimization technique.

Table B1: A comparison between optimization technique implementations of the stone cipher

	Balance	Area	Speed
Total logic elements	3085	2870	3226
Total combinational functions	3085	2870	3226
4 input functions	2562	2394	2808
3 input functions	517	465	390
<=2 input functions	6	11	28
Maximum fan-out	1025	517	1025
Total fan-out	12067	11249	12714
Average fan-out	3.30	3.26	3.34
Block interconnects	4080	3998	4338
C16 interconnects	508	539	533
C4 interconnects	3337	3153	3179
Direct links	273	358	380
Global clocks	1	1	1
Local interconnects	1139	910	1229
R24 interconnects	779	821	781
R4 interconnects	3708	3529	3969

The comparison between optimization techniques was extracted from the timing reports of implementing area and speed optimization. Figure B.1 shows a comparison chart between various implementation delays.

● in area optimization, the worst case pin-to-pin delay was found to be equal to 26.341 ns. The longest pin-to-register delay was 7.885 ns and the shortest pin-to-register delay was 6.840 ns. The longest register-to-pin delay was 22.930 ns.

● in speed optimization, the worst case pin-to-pin delay was found to be equal to 27.233 ns. The longest pin-to-register delay was 8.232 ns and the shortest pin-to-register delay was 6.912 ns. The longest register-to-pin delay was 22.230 ns.

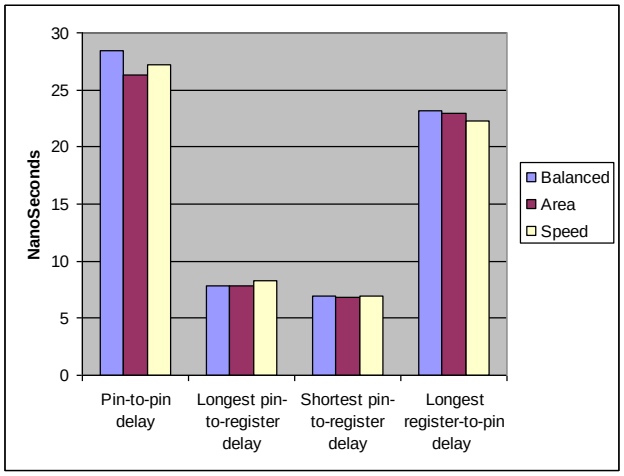

Figure B.1. Delays in our design of the cipher implementation

Magdy Saeb received the BSEE, School of Engineering, Cairo University, in 1974, the MSEE, and Ph.D. degrees in Electrical & Computer Engineering, University of California, Irvine, in 1981 and 1985, respectively. He was with Kaiser Aerospace and Electronics, Irvine California, and The Atomic Energy Establishment, Anshas, Egypt. Currently, he is a professor in the Department of Computer Engineering, Arab Academy for Science, Technology & Maritime Transport, Alexandria, Egypt; He was on-leave working as a principal researcher in the Malaysian Institute of Microelectronic Systems (MIMOS). His current research interests include Cryptography, FPGA Implementations of Cryptography and Steganography Data Security Techniques, Encryption Processors, Mobile Agent Security. www.magdysaeb.net.

Rabie Mahmoud received the BS. Degree, Faculty of Science, Tishreen University, Syria, in 2001, the MS. in Computational Science, Faculty of Science, Cairo University, Egypt in 2007. Currently, he is working on his Ph.D. Dissertation in the Department of Mathematics, Faculty of Science, Cairo University, ., His current interests include Image processing, Cryptography, FPGA Implementations of Cryptography and Data Security Techniques. rabiemah@yahoo.com