A Generalized Crypto Logic Unit (GCLU) With Software and Hardware Implementations

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

1. General Organization of Remote Sensing (GORS),
Damascus, Syria.

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

Abstract: The Generalized Crypto Logic Unit (GCLU) is a key-driven encryption function modified from the Crypto Logic Unit (CLU) which is defined as the cipher engine of Metamorphic Stone Cipher. This Crypto Logic Unit uses eight bit-balanced operations. These operations are: XOR, INV, ROR, NOP, XNOR, SWAP, ROL, RevOr for bitwise xor, invert, rotate right, no operation, xnor, swap, rotate left, and reverse order respectively. In addition, we provide the Software and Field Programmable Gate Array (FPGA) implementation of the Generalized Crypto Logic Unit.

Keywords: Generalized Crypto Logic Unit, Metamorphic, Cipher, Cryptography, FPGA.

1. Introduction

The Crypto Logic Unit (CLU) is considered the cipher engine of the key-driven Stone Metamorphic Cipher [1], [2], and is used to modify many famous ciphers to increase the cipher's entropy and improve its security. These modified ciphers include the Metamorphic Twofish Cipher [3], the Metamorphic MARS Cipher [4], and the Metamorphic-Key-Hopping GOST Cipher [5]. The CLU is built using four low-level bit-balanced operations. 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 a plaintext bit without any change (NOP). The Generalized Crypto Logic Unit (GCLU), on the other hand, extrapolates the idea of using the bit-balanced four low-level operations in eight low-level bit-balanced operations. These are: the four operations of the CLU plus four other low-level operations. These newly-added operations are: XNORing a key bit with a plaintext bit (XNOR), swapping a plaintext bit with another one in a given plaintext word (SWAP), a left rotation operation (ROL), and the reverse order operation that reverses a plaintext word (RevOr). In the following sections, we discuss the GCLU structure, and its software and hardware implementations. Finally, we provide a summary and our conclusions.

2. Generalized Crypto Logic Unit (GCLU)

As discussed in the introduction section, the Generalized Crypto Logic Unit (GCLU) is a modified Crypto Logic Unit, which is defined in the key-driven Stone Metamorphic cipher, by adding four more operations to the CLU operations. The resulting eight low-level operations are:

- (XOR) by XORing a key bit with a plaintext bit,
- (INV) by inverting a plaintext bit,
- (NOP) by producing the plaintext without any change,
- (ROR) by exchanging one plaintext bit with another one in a given plaintext word using a right rotation operation,
- (XNOR) by XNORing a key bit with a plaintext bit,
- (SWAP) by exchanging one plaintext bit with another one in a given plaintext word using a swap operation,
- (ROL) by exchanging one plaintext bit with another one in a given plaintext word using a left rotation operation,
- (RevOr) by exchanging one plaintext bit with another one in a given plaintext word using a reverse order operation.

Figure 1 shows the basic generalized crypto logic unit and Table 1 demonstrates the details of each one of the GCLU operations

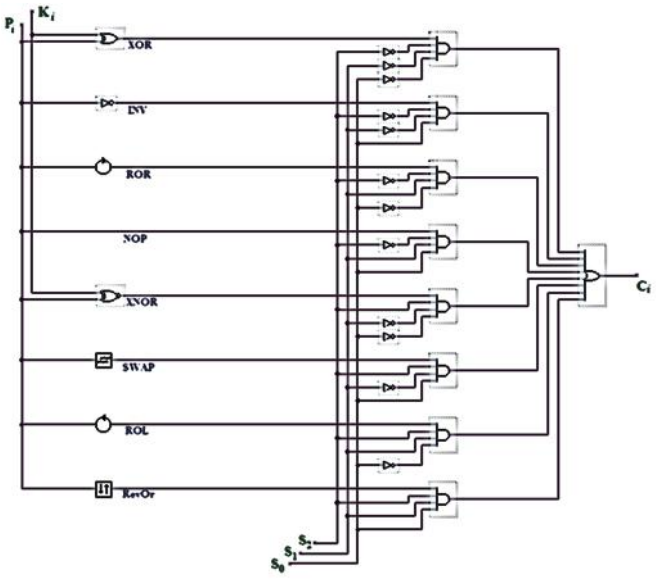

Figure 1. The basic generalized crypto logic unit (GCLC)

Table 1. GCLU operations

Mnemonic	*Operation*	*Select Operation Code*
XOR	$C_i = K_i \oplus P_i$	"000"
INV	$C_i = \dot{E}\ P_i$	"001"
ROR	$C_i = P_i \ggg m$	"010"
NOP	$C_i = P_i$	"011"
XNOR	$C_i = K_i \odot P_i$	"100"
SWAP	$C_i = \# P_i$	"101"
ROL	$C_i = P_i \lll m$	"110"
RevOr	$C_i = \Downarrow\Uparrow P_i$	"111"

Similar to the CLU, the GCLU can be used as the encryptor and the decryptor where by changing the output cipher bit to become an input plaintext bit, the new output will be the same as the old plain text bit. But, this is a feature for XOR, INV, NOP, XNOR, SWAP, or RevOr functions. The exceptions are in the cases of the decryptor of ROR will use ROL, and the decryptor of ROL will use ROR. Appendix A shows the truth table of GCLU. Likewise, the operation_selection_bits (S_2 S_1 S_0) can be chosen from any three sub-key bits; the same idea applies for the rotation_selection_bits ($S'_n \cdots S'_0$). Figure 2 shows the locations of operation selection bits and rotation selection bits.

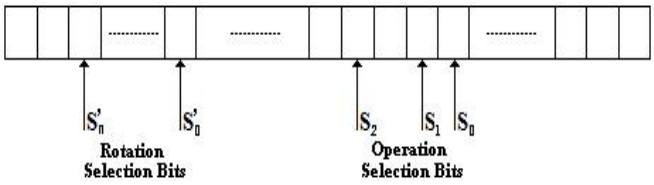

Figure 2. The proposed key format where the location of the operation and rotation selection bits is shown

3. The Algorithm

In this section, we provide the formal description of the Ulta Crypto Logic Unit as follows:

Function Ulta Crypto Logic Unit (GCLU)
Begin
1. *Read the next plaintext message P_i;*
2. *Read the next sub-key K_i;*
3. *Read n-bit rotation_selection_bits from sub-key where 2^n=Block size B;*
4. *Read 3-bit operation_selection_bits form sub-key;*
5. *Use operation selection & rotation selection bits to select and perform the operation:*
XOR *when operation_selection_bits="000"*
INV *when operation_selection_bits="001"*
ROR *when operation_selection_bits="010"*
NOP *when operation_selection_bits="011"*
XNOR *when operation_selection_bits="100"*
SWAP *when operation_selection_bits="101"*
ROL *when operation_selection_bits="110"*
RevOr *when operation_selection_bits="111";*
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. Software/Hardware Implementation

A pseudo C++ function [6] of the generalized crypto logic unit is applied representing the truth table of GCLU utilizing Microsoft Visual C++ 2010 Express. Appendix C provides a sample C++ code for the GCLU. Figure 3 shows the correct build solution of the C++ project of GCLU. Figure 4 is the execution screen of GCLU. Furthermore, a proof-of-concept FPGA-based implementation is used to encrypt a one byte plaintext using one byte sub-key word. We have implemented the GCLU applying the VHDL hardware description language and utilizing Altera design environment Quartus II 13.0 Service Pack 1 Web Edition [7]. The FPGA design was implemented using EP2C5AF256A7, Cyclone II family device. Appendix D represents the sample VHDL code for GCLU. The implementation results and the schematic diagram for GCLU are shown in Figure 5. The RTL screen and technology map viewer for GCLU are shown in Figures 6, and 7 respectively. Figure 8 demonstrates the floor plan for GCLU. The details of the analysis and synthesis summary and timing analyzer are shown in appendix B.

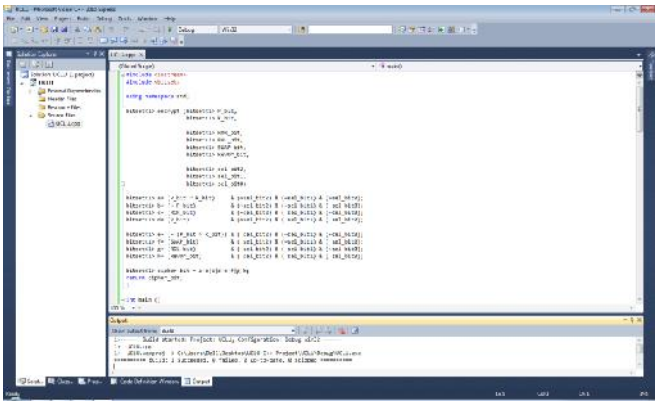

Figure 3. C++ project of GCLU showing the correct build solution

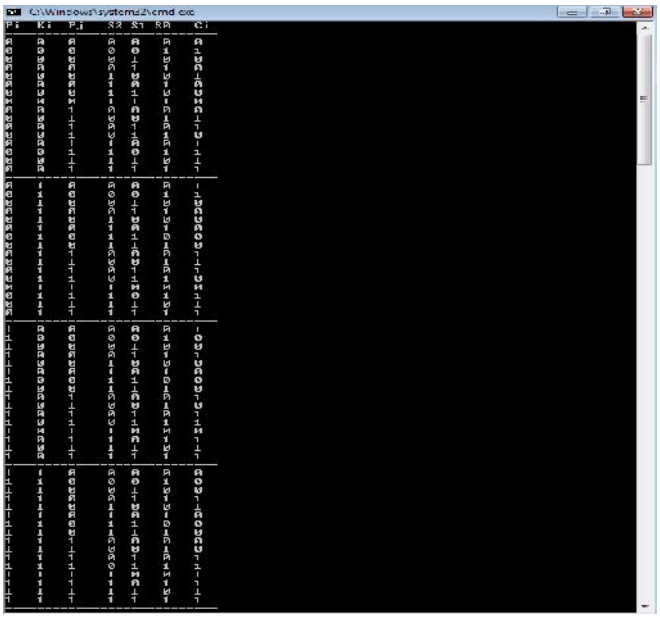

Figure 4. Execution screen of the C++ project of GCLU showing the truth table of the GCLU

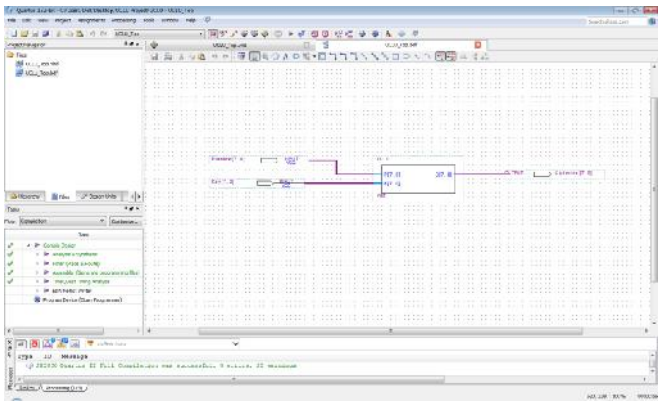

Figure 5. Schematic diagram of GCLU showing also correct implementation

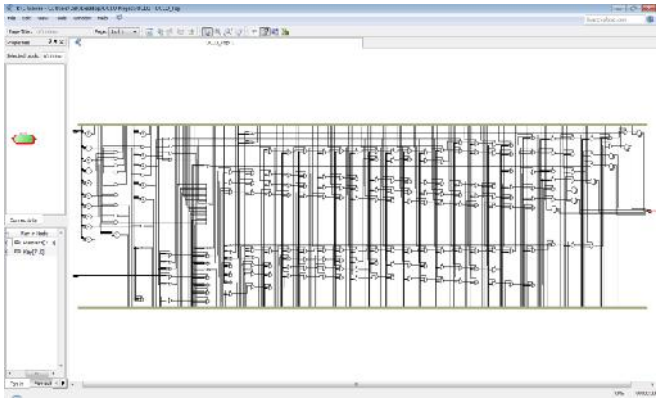

Figure 6. RTL screen for GCLU

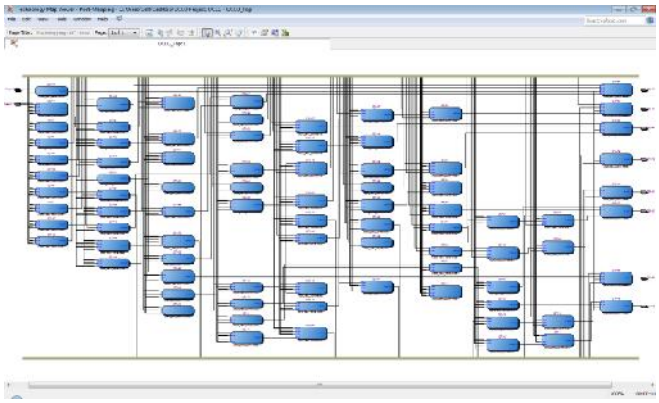

Figure 7. Technology Map viewer of GCLU

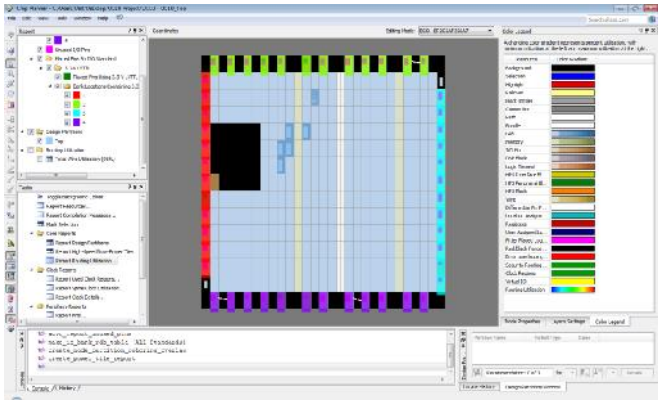

Figure 8. Floor-plan of Chip of GCLU

5. Summary and Conclusions

We have presented the Generalized Crypto Logic Unit (GCLU) which is a modified version of the crypto logic unit (CLU) of the key-driven Stone Metamorphic Cipher. The GCLU is constructed using eight bit-balanced operations. The eight low-level operations are pseudo-randomly chosen using three key-dependent selection bits. These operations are: bitwise xor, invert, rotate right, no operation, xnor, swap, rotate left, and reverse order. In addition, we have shown that the generalized crypto logic unit can be implemented as Software or FPGA-based Hardware. We have included a proof-of-concept software and FPGA hardware implementations. The aim of modifying the CLU to be GCLU is to increase a cipher's entropy by providing a higher degree of randomness and thus an enhanced security. This GCLU is then utilized to modify well-known ciphers in order to achieve key-dependent encryption.

References

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

[2] Rabie A. Mahmoud, Magdy Saeb, "Hardware Implementation of the Stone Metamorphic Cipher," International Journal of Computer Science and Network Security (IJCSNS), Vol.10, No.8, pp.54-60, 2010.

[3] Rabie A. Mahmoud, Magdy Saeb, "A Metamorphic-Enhanced Twofish Block Cipher And Associated FPGA Implementation," International Journal of Computer Science and Communication Security (IJCSCS), Vol.2, No.1, Jan., 2012.

[4] Ahmed Helmy, Magdy Saeb, A. Baith Mohamed, "A Metamorphic-Enhanced MARS Block Cipher," International Journal of Computer Science and Communication Security (IJCSCS), Vol.3, No.4, Jul., 2013.

[5] Rabie A. Mahmoud, Magdy Saeb, "A Metamorphic-Key-Hopping GOST Cipher and Its FPGA Implementation," International Journal of Computer Science and Communication Security (IJCSCS), Vol.3, No.7, Oct., 2013.

[6] C++ site: http://www.cplusplus.com

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

Appendix A: The truth table of the GCLU

P_i	K_i	$\rightarrow P_j$	S_2	S_1	S_0	Operation	C_i
0	0	0	0	0	0	XOR	0
0	0	0	0	0	1	INV	1
0	0	0	0	1	0	ROR	0
0	0	0	0	1	1	NOP	0
0	0	0	1	0	0	XNOR	1
0	0	0	1	0	1	SWAP	0
0	0	0	1	1	0	ROL	0
0	0	0	1	1	1	RevOr	0
0	0	1	0	0	0	XOR	0
0	0	1	0	0	1	INV	1

0	0	1	0	1	0	ROR	1
0	0	1	0	1	1	NOP	0
0	0	1	1	0	0	XNOR	1
0	0	1	1	0	1	SWAP	1
0	0	1	1	1	0	ROL	1
0	0	1	1	1	1	RevOr	1
0	1	0	0	0	0	XOR	1
0	1	0	0	0	1	INV	1
0	1	0	0	1	0	ROR	0
0	1	0	0	1	1	NOP	0
0	1	0	1	0	0	XNOR	0
0	1	0	1	0	1	SWAP	0
0	1	0	1	1	0	ROL	0
0	1	0	1	1	1	RevOr	0
0	1	1	0	0	0	XOR	1
0	1	1	0	0	1	INV	1
0	1	1	0	1	0	ROR	1
0	1	1	0	1	1	NOP	0
0	1	1	1	0	0	XNOR	0
0	1	1	1	0	1	SWAP	1
0	1	1	1	1	0	ROL	1
0	1	1	1	1	1	RevOr	1
1	0	0	0	0	0	XOR	1
1	0	0	0	0	1	INV	0
1	0	0	0	1	0	ROR	0
1	0	0	0	1	1	NOP	1
1	0	0	1	0	0	XNOR	0
1	0	0	1	0	1	SWAP	0
1	0	0	1	1	0	ROL	0
1	0	0	1	1	1	RevOr	0
1	0	1	0	0	0	XOR	1
1	0	1	0	0	1	INV	0
1	0	1	0	1	0	ROR	1
1	0	1	0	1	1	NOP	1
1	0	1	1	0	0	XNOR	0
1	0	1	1	0	1	SWAP	1
1	0	1	1	1	0	ROL	1
1	0	1	1	1	1	RevOr	1
1	1	0	0	0	0	XOR	0
1	1	0	0	0	1	INV	0
1	1	0	0	1	0	ROR	0
1	1	0	0	1	1	NOP	1
1	1	0	1	0	0	XNOR	1
1	1	0	1	0	1	SWAP	0
1	1	0	1	1	0	ROL	0
1	1	0	1	1	1	RevOr	0
1	1	1	0	0	0	XOR	0
1	1	1	0	0	1	INV	0
1	1	1	0	1	0	ROR	1
1	1	1	0	1	1	NOP	1
1	1	1	1	0	0	XNOR	1
1	1	1	1	0	1	SWAP	1
1	1	1	1	1	0	ROL	1
1	1	1	1	1	1	RevOr	1

Appendix B: The analysis & synthesis and fitter report details

FPGA synthesis of GCLU for 1-byte inputs consumes 83 logic elements to perform multiplexers with no registers, and needs 17.255 ns from input port "Plaintext[7]" to output port "Ciphertext[7]". Table 2 and Table 3 show the number of usage logic elements and the interconnections between them in Area, Speed, and Balanced optimization technique. Figure 9 shows the delays in the design of the GCLU.

Analysis & Synthesis and Fitter Summary

•Family: Cyclone II
•Device: EP2C5AF256A7
• Nominal Core Voltage: 1.20 V
• Minimum Core Junction Temperature: -40 °C
• Maximum Core Junction Temperature: 125 °C.

•Optimization Technique: Balanced
•Total logic elements: 83 out of 4,608 (2%)
-- Combinational with no register:83
-- Register only:0
-- Combinational with a register:0
Logic element usage by number of LUT inputs
-- 4 input functions: 47
-- 3 input functions: 34
-- <=2 input functions: 2
-- Register only: 0
Logic elements by mode
-- Normal mode: 83
-- Arithmetic mode: 0
• Total LABs: 6 out of 288 (2 %)
• Total fan-out: 302
• Average fan-out: 2.75
• Highest non-global fan-out: 27
• Maximum fan-out: 27

•Block interconnects: 80 out of 15,666 (< 1 %)
•C16 interconnects: 7 out of 812 (< 1 %)
•C4 interconnects: 69 out of 11,424 (< 1 %)
•Direct links: 7 out of 15,666 (< 1 %)
•Global clocks: 0 out of 8 (0 %)
•Local interconnects: 48 out of 4,608 (1 %)
•R24 interconnects: 7 out of 652 (1 %)
•R4 interconnects: 34 out of 13,328 (< 1 %)

Table 2. A synthesis comparison between optimization technique implementations of GCLU

	Balanced	Area	Speed
Total logic elements	83	84	83
Total combinational functions	83	84	83
4 input functions	47	44	60
3 input functions	34	39	19
<=2 input functions	2	1	4
Total fan-out	302	303	313
Maximum fan-out	27	23	31
Average fan-out	2.75	2.73	2.85

Table 3. A fitter comparison between optimization technique implementations of GCLU

	Balanced	Area	Speed
Block interconnects	80	81	103
C16 interconnects	7	4	13
C4 interconnects	69	69	68
Direct links	7	2	5
Global clocks	0	0	0
Local interconnects	48	49	50
R24 interconnects	7	5	8
R4 interconnects	34	39	53

Timing Analyzer Summary

In Balanced Optimization

- Longest propagation delay was 17.255 ns from input port "Plaintext[7]" to output port "Ciphertext[7]".
- Longest minimum propagation delay was 7.182 ns from input port "Plaintext[1]" to output port "Ciphertext[7]".

In Area Optimization

- Longest propagation delay was 17.639 ns from input port "Plaintext[0]" to output port "Ciphertext[2]".
- Longest minimum propagation delay was 7.229 ns from input port "Plaintext[2]" to output port "Ciphertext[4]".

In Speed Optimization

- Longest propagation delay was 15.911 ns from input port "Plaintext[4]" to output port "Ciphertext[3]".
- Longest minimum propagation delay was 6.708 ns from input port "Plaintext[4]" to output port "Ciphertext[5]".

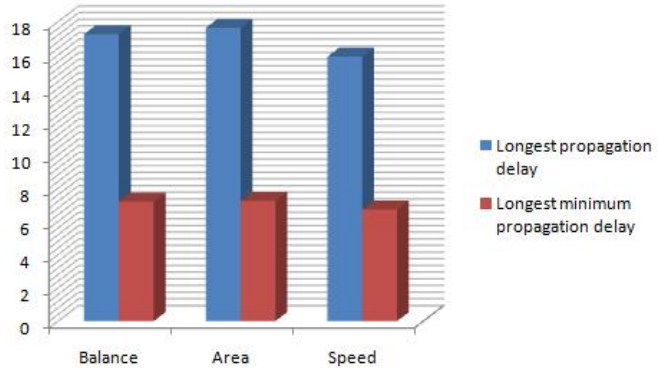

Figure 9. Delays in the design of the GCLU

Appendix C: Sample C++ code for GCLU

```
#include <iostream>
#include <bitset>
using namespace std;

bitset<1>  encrypt (bitset<1>  P_bit,
                    bitset<1>  k_bit,
                    bitset<1>  ROR_bit,
                    bitset<1>  ROL_bit,
                    bitset<1>  SWAP_bit,
                    bitset<1>  RevOr_bit,
                    bitset<1>  sel_bit2,
                    bitset<1>  sel_bit1,
                    bitset<1>  sel_bit0)
{
bitset<1>  a,b,c,d,e,f,g,h;
a=(P_bit ^ k_bit)  & (~sel_bit2) &(~sel_bit1)&(~sel_bit0);
b=(~ P_bit)        & (~sel_bit2) &(~sel_bit1)&(  sel_bit0);
c=(ROR_bit)        & (~sel_bit2) &(  sel_bit1)&(~sel_bit0);
d=(P_bit)          & (~sel_bit2) &(  sel_bit1)&(  sel_bit0);
e=(~ (P_bit ^ k_bit)) &(sel_bit2)&(~sel_bit1)&(~sel_bit0);
f=(SWAP_bit)       &(sel_bit2)&(~sel_bit1)&(  sel_bit0);
g=(ROL_bit)        &(sel_bit2)&(  sel_bit1)&(~sel_bit0);
h=(RevOr_bit)      &(sel_bit2)&(  sel_bit1)&(  sel_bit0);

bitset<1>  cipher_bit = a|b|c|d|e|f|g|h;
return cipher_bit;
}
int main ()
{
   cout << encrypt(0,0,0,0,0,0,0,0,0) << endl;
   cout << encrypt(0,0,0,0,0,0,0,0,1) << endl;
   cout << encrypt(0,0,0,0,0,0,0,1,0) << endl;
   cout << encrypt(0,0,0,0,0,0,0,1,1) << endl;
   cout << encrypt(0,0,0,0,0,0,1,0,0) << endl;
   cout << encrypt(0,0,0,0,0,0,1,0,1) << endl;
   cout << encrypt(0,0,0,0,0,0,1,1,0) << endl;
   cout << encrypt(0,0,0,0,0,0,1,1,1) << endl;
      .
      .
      .
   cout << encrypt(1,1,1,1,1,1,0,0,0) << endl;
   cout << encrypt(1,1,1,1,1,1,0,0,1) << endl;
   cout << encrypt(1,1,1,1,1,1,0,1,0) << endl;
   cout << encrypt(1,1,1,1,1,1,0,1,1) << endl;
   cout << encrypt(1,1,1,1,1,1,1,0,0) << endl;
   cout << encrypt(1,1,1,1,1,1,1,0,1) << endl;
   cout << encrypt(1,1,1,1,1,1,1,1,0) << endl;
   cout << encrypt(1,1,1,1,1,1,1,1,1) << endl;
return 0;
}
```

Appendix D: Sample VHDL code for GCLU

```
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 GCLU IS
PORT( P :  in   std_logic_vector (7 downto 0);
      K : in   std_logic_vector (7 downto 0);
      C : out std_logic_vector (7 downto 0));
END GCLU;

ARCHITECTURE behavioral OF GCLU IS

SIGNAL Operation_sel_bits: std_logic_vector (7 downto 0);
SIGNAL Rortaion_sel_bits  : std_logic_vector (7 downto 0);

SIGNAL SWAP_P : std_logic_vector (7 downto 0);
SIGNAL RevOr_P : std_logic_vector (7 downto 0);

BEGIN

Operation_sel_bits <= K(7) & K(5) & K(3);
Rortaion_sel_bits  <= K(1) & K(0) & K(4);

C <= P XOR K     WHEN Operation_sel_bits="000" ELSE
     NOT P       WHEN Operation_sel_bits="001" ELSE
     P           WHEN Operation_sel_bits="011" ELSE
     P XNOR K    WHEN Operation_sel_bits="100" ELSE
     SWAP_P      WHEN Operation_sel_bits="101" ELSE
```

```
      RevOr_P     WHEN Operation_sel_bits="111"  ELSE

---ROR------
      P                                  WHEN
Operation_sel_bits="010" AND Rortaion_sel_bits="000"
                                                ELSE
      P(0) & P(7 downto 1)               WHEN
Operation_sel_bits="010" AND Rortaion_sel_bits="001"
                                                ELSE
      P(1 downto 0) & P(7 downto 2)  WHEN
Operation_sel_bits="010" AND Rortaion_sel_bits="010"
                                                ELSE
      P(2 downto 0) & P(7 downto 3)  WHEN
Operation_sel_bits="010" AND Rortaion_sel_bits="011"
                                                ELSE
      P(3 downto 0) & P(7 downto 4)  WHEN
Operation_sel_bits="010" AND Rortaion_sel_bits="100"
                                                ELSE
      P(4 downto 0) & P(7 downto 5)  WHEN
Operation_sel_bits="010" AND Rortaion_sel_bits="101"
                                                ELSE
      P(5 downto 0) & P(7 downto 6)  WHEN
Operation_sel_bits="010" AND Rortaion_sel_bits="110"
                                                ELSE
      P(6 downto 0) & P(7)               WHEN
Operation_sel_bits="010" AND Rortaion_sel_bits="111"
                                                ELSE
---ROL------
      P                                  WHEN
Operation_sel_bits="110" AND Rortaion_sel_bits="000"
                                                ELSE
      P(6 downto 0) & P(7)               WHEN
Operation_sel_bits="110" AND Rortaion_sel_bits="001"
                                                ELSE
      P(5 downto 0) & P(7 downto 6)  WHEN
Operation_sel_bits="110" AND Rortaion_sel_bits="010"
                                                ELSE
      P(4 downto 0) & P(7 downto 5)  WHEN
Operation_sel_bits="110" AND Rortaion_sel_bits="011"
                                                ELSE
      P(3 downto 0) & P(7 downto 4)  WHEN
Operation_sel_bits="110" AND Rortaion_sel_bits="100"
                                                ELSE
      P(2 downto 0) & P(7 downto 3)  WHEN
Operation_sel_bits="110" AND Rortaion_sel_bits="101"
                                                ELSE
      P(1 downto 0) & P(7 downto 2)  WHEN
Operation_sel_bits="110" AND Rortaion_sel_bits="110"
                                                ELSE
      P(0) & P(7 downto 1)               WHEN
Operation_sel_bits="110" AND Rortaion_sel_bits="111";

END behavioral;
```

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. He is a professor and former head of 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). He is the Chief Technology Officer of an Information Security Company GWIS. He holds five International Patents in Cryptography. His current research interests include Cryptography, FPGA Implementations of Cryptography and Steganography Data Security Techniques, Encryption Processors, Mobile Agent Security. www.magdysaeb.net

Rabie A. Mahmoud received the B.Sc. Degree, Faculty of Science, Tishreen University, Latakia-Syria, in 2001, the MS. and Ph.D.in Computational Science, Faculty of Science, Cairo University, Egypt, in 2007 and 2011 respectively. Currently, he is working in General Organization of Remote Sensing (GORS), Damascus, Syria. His current interests include Cryptography, FPGA Implementations of Cryptography and Data Security Techniques. rabiemah@yahoo.com