Available online at www.elixirpublishers.com (Elixir International Journal)

Electrical Engineering

Elixir Elec. Engg. 51 (2012) 10700-10703

ISSN: 2229-712X

FPGA implementation of optimized the 64-bit RC5 encryption algorithm

Bahram Rashidi
University of Tabriz, Iran.

ARTICLE INFO

Article history:
Received: 2 August 2012;
Received in revised form:
20 September 2012;
Accepted: 27 September 2012;

Keywords
RC-5;
Encryption;
FPGA;
Barrel shifter.

ABSTRACT
This paper presents, a FPGA based the 64-bit RC5 encryption algorithm. One of complex operation in RC5 encryption is rotate thus we implementation this operation on FPGA using barrel shifter. We implement total of mathematic equations based optimized logic circuits until dynamic power consumption reduced, also for increase in speed and maximum operation frequency we using pipelining technique in proposed method. The results from the place and route report indicate that logic utilization by this architecture is 17% with a maximum clock frequency of 175.69 MHz.

Introduction

The use of cryptography is growing rapidly with the adoption of computer technology. The design of cryptographic ciphers is still not well understood; we cannot prove the security of an algorithm. Currently, the only way to be sure of the security of an algorithm is to study it for a long period of time and use the absence of attacks as evidence confirming its security. All ciphers are vulnerable to an exhaustive key search attack. An attacker can try every single possible key to check its correctness. This is time consuming, but feasible for several widely deployed ciphers. An obvious way to conduct an exhaustive key search attack is to write software that will check each key in turn. Current microprocessors have clock rates in the gigahertz range and can execute several instructions per clock cycle. They are also cheap, highly available and easy to program[1]. Implementation of RC-5 algorithm on FPGA has been proposed by some researchers [2-5], In[2], The goal is to combine traditional distributed computing concepts with reconfigurable hardware, where free resource area can be used to achieve computation speed-ups. Their approach is demonstrated on the base of a RC5 brute force key search analog to the distributed net project. In[3], proposes a new hardware dedicated to RC5, a typical cryptograph. In the proposed RC5 dedicated hardware, by introducing an architecture suitable for each operation used for the encryption, high-speed processing and area reduction can be realized. In[4], they discuss the options for brute-force cracking of the RC5 block cipher, that is, for revealing the unknown secret key, given a sample ciphertext and a portion of the corresponding plaintext. First, they summarize the methods employed by the current cracking efforts. Then, present two hardware architectures for finding the secret key using the “brute force” method. They implement the hardware in FPGA and ASIC. In[5], propose an efficient and reconfigurable hardware architecture for the RC5 block cipher implementation. The design can be reconfigured according to the different application requirements with variable parameters. In this paper use a FPGA device to implementation of a high speed RC5 encryption. FPGAs provide the functionality of a custom chip without the high upfront cost and lead time. They have much lower clock rates than general-purpose CPUs, but can be designed to perform one task exceptionally well. Parallelism can also be exploited to increase the overall encryption rate.

Description of RC5 Algorithm

The RC5 encryption algorithm was designed by Ronald Rivest of Massachusetts Institute of Technology (MIT) and it first appeared in December 1994. RSA Data Security, Inc. estimates that RC5 and its successor, RC6, are strong candidates for potential successors to DES. RC5 analysis (RSA Laboratories) is still in progress and is periodically updated to reflect any additional findings. RC5 is a symmetric block cipher designed to be suitable for both software and hardware implementation. It is a parameterised algorithm, with a variable block size, a variable number of rounds and a variable-length key. This provides the opportunity for great flexibility in both performance characteristics and the level of security. A particular RC5 algorithm is designated as RC5-w/r/b. The number of bits in a word, w, is a parameter of RC5. different choices of this parameter result in different RC5 algorithms. RC5 is iterative in structure, with a variable number of rounds. The number of rounds, r, is a second parameter of RC5. RC5 uses a variable-length secret key. The key length b (in bytes) is a third parameter of RC5. These parameters are summarized as follows [6]:

w: The word size, in bits. The standard value is 32bits; allowable values are 16, 32 and 64. RC5 encrypts two-word blocks so that the plaintext and ciphertext blocks are each 2w bits long.

r: The number of rounds. Allowable values of r are 0, 1, . . . , 255. Also, the expanded key table S contains t = 2 (r + 1) words.

b: The number of bytes in the secret key K. Allowable values of b are 0, 1, . . . , 255.

K: The b-byte secret key; K[0], K[1], . . . , K[b − 1]

RC5 consists of three components: a key expansion algorithm, an encryption algorithm and a decryption algorithm. These algorithms use the following three primitive operations:

Tele:
E-mail addresses: bahram88@ms.tabrizu.ac.ir

1. + Addition of words modulo 2^w
2. $\oplus$ Bit-wise exclusive-OR of words
3. <<< Rotation symbol: the rotation of x to the left by y bits is denoted by x <<< y.

One design feature of RC5 is its simplicity, which makes RC5 easy to implement. Another feature of RC5 is its heavy use of data-dependent rotations in encryption; this feature is very useful in preventing both differential or linear cryptanalysis. Given RC5-32/16/10. This particular RC5 algorithm has 32-bit words, 16 rounds, a 10-byte (80-bit) secret key variable and an expanded key table S of t = 2(r + 1) = 2(16 + 1) = 34 words. Rivest proposed RC5-32/12/16 as a block cipher providing a normal choice of parameters, i.e. 32-bit words, 12 rounds, 16-byte (128-bit) secret key variable and an expanded key table of 26 words.

Key Expansion

As explained in [6], the key-expansion algorithm expands the user's key K to fill the expanded key table S, so that S resembles an array of t = 2(r + 1) random binary words determined by K. It uses two word-size magic constants P_w and Q_w defined for arbitrary w as shown below:

P_w = Odd $((e - 2)2^w)$
Q_w = Odd $((\varphi - 1)2^w)$
where
e = 2.71828 . . . (base of natural logarithms)
$\varphi = (1 + \sqrt{5})/2 = 1.61803$ . . . (golden ratio)
Odd(x) is the odd integer nearest to x.

First algorithmic step of key expansion: This step is to copy the secret key K[0, 1, . . . ,b − 1] into an array L[0, 1, . . . , c − 1] of $c = \lceil b/u \rceil$ words, where u = w/8 is the number of bytes/word. This first step will be achieved by the following pseudocode operation: for i = b – 1 down to 0 do L[i/u] = (L[i/u] <<< 8) + K[i]; where all bytes are unsigned and the array L is initially zeroes. Second algorithmic step of key expansion: This step is to initialise array S to a particular fixed pseudo-random bit pattern, using an arithmetic progression modulo 2^w determined by two constants P_w and Q_w.

S[0] = P_w:
For i = 1 to t − 1 do S[i] = S[i − 1] + Q_w.

Third algorithmic step of key expansion: This step is to mix in the user's secret key in three passes over the arrays S and L. More precisely, due to the potentially different sizes of S and L, the larger array is processed three times, and the other array will be handled more after.

i = j = 0;
A = B = 0;
do 3∗ max (t, c) times:
A = S[i] = (S[i] + A + B) <<< 3
B = L[j] = (L[j] + A + B) <<< (A + B);
i = (i + 1) (mod t);
j = (j + 1) (mod c).

Note that with the key-expansion function it is not so easy to determine K from S, due to the one-wayness.

Consider RC5-32/12/16. Since w = 32, r = 12 and b = 16, we have
u = w/8 = 32/8 = 4 bytes/word
$c = \lceil b/u \rceil = \lceil 16/4 \rceil = 4$ words
t = 2(r + 1) = 2(12 + 1) = 26 words
The plaintext and the user's secret key are given as follows:
Plaintext = eedba521 6d8f4b15
Key = 915f4619be41b2516355a50110a9ce91
1. Key expansion Two magic constants
P_{32} = 3084996963 = 0xb7e15163
Q_{32} = 2654435769 = 0x9e3779b9
Step 1
For i = b − 1 down to 0 do L[i/u] = (L[i/u] <<< 8) + K[i] where b = 16, u = 4 and
L is initially 0.
L[i/4] = L[3] for i = 15, 14, 13 and 12.
L[3] = (L[3] <<< 8) + K[15] = 00 + 91 = 91
L[3] = (L[3] <<< 8) + K[14] = 9100 + ce = 91ce
L[3] = (L[3] <<< 8) + K[13] = 91ce00 + a9 = 91cea9
*L[3] = (L[3] <<< 8) + K[12] = 91cea900 + 10 = 91cea910
L[i/4] = L[2] for i = 11, 10, 9 and 8.
L[2] = (L[2] <<< 8) + K[11] = 00 + 01 = 01
L[2] = (L[2] <<< 8) + K[10] = 0100 + a5 = 01a5
L[2] = (L[2] <<< 8) + K[9] = 01a500 + 55 = 01a555
∗L[2] = (L[2] <<< 8) + K[8] = 01a55500 + 63 = 01a55563
L[i/4] = L[1] for i = 7, 6, 5 and 4.
L[1] = (L[1] <<< 8) + K[7] = 00 + 51 = 51
L[1] = (L[1] <<< 8) + K[6] = 5100 + b2 = 51b2
L[1] = (L[1] <<< 8) + K[5] = 51b200 + 41 = 51b241
∗L[1] = (L[1] <<< 8) + K[4] = 51b24100 + be = 51b241be
L[i/4] = L[0] for i = 3, 2, 1 and 0.
L[0] = (L[0] <<< 8) + K[3] = 00 + 19 = 19
L[0] = (L[0] <<< 8) + K[2] = 1900 + 46 = 1946
L[0] = (L[0] <<< 8) + K[1] = 194600 + 5f = 19465f
∗L[0] = (L[0] <<< 8) + K[0] = 19465f00 + 91 = 19465f91
Thus, converting the secret key from bytes to words (*) yields:
L[0] = 19465f91
L[1] = 51b241be
L[2] = 01a55563
L[3] = 91cea910
Step 2
S[0] = P_{32}. For i = 1 to 25, do S[i] = S[i − 1] + Q_{32}:
S[0] = b7e15163
S[1] = S[0] + Q_{32} = b7e15163 + 9e3779b9 = 5618cb1c
S[2] = S[1] + Q_{32} = 5618cb1c + 9e3779b9 = f45044d5
S[3] = S[2] + Q_{32} = f45044d5 + 9e3779b9 = 9287be8e
… …
S[25] = S[24] + Q_{32} = 8f14babb + 9e3779b9 = 2b4c3474
When the iterative processes continue up to t−1=2(r+1)−1=25, we can obtain the expanded key table S as shown below:

S[0] = b7e15163	S[09] = 47d498e4	S[18] = d7c7e065
S[1] = 5618cb1c	S[10] = e60c129d	S[19] = 75ff5a1e
S[2] = f45044d5	S[11] = 84438c56	S[20] = 1436d3d7
S[3] = 9287be8e	S[12] = 227b060f	S[21] = b26e4d90
S[4] = 30bf3847	S[13] = c0b27fc8	S[22] = 50a5c749
S[5] = cef6b200	S[14] = 5ee9f981	S[23] = eedd4102
S[6] = 6d2e2bb9	S[15] = fd21733a	S[24] = 8d14babb
S[7] = 0b65a572	S[16] = 9b58ecf3	S[25] = 2b4c3474
S[8] = a99d1f2b	S[17] = 399066ac	

Step 3
i = j = 0; A = B = 0;
3 × max(t, c) = 3 × 26 = 78 times
A = S[i] = (S[i] + A + B) <<< 3
B = L[j] = (L[j] + A + B) <<< (A + B)
i = i + 1(mod 26)
j = j + 1(mod 4)
A = S[0] = (b7e15163 + 0 + 0) <<< 3
= b7e15163 <<< 3 = bf0a8b1d
B = L[0] = (19465f91 + bf0a8b1d) <<< (A + B)
= d850eaae <<< bf0a8b1d = db0a1d55

A = S[1] = (5618cb1c + bf0a8b1d + db0a1d55) <<< 3
= f02d738e <<< 3 = 816b9c77
B = L[1] = (51b241be + 816b9c77 + db0a1d55) <<< (A + B)
= ae27fb8a <<< 5c75b9cc = 7fb8aae2
A = S[2] = (f45044d5 + 816b9c77 + 7fb8aae2) <<< 3
= f5748c2e <<< 3 = aba46177
B = L[2] = (01a55563 + aba46177 + 7fb8aae2) <<< (A + B)
= 2d0261bc <<< 2b5d0c59 = 785a04c3
A = S[3] = (9287be8e + aba46177 + 785a04c3) <<< 3
= b68624c8 <<< 3 = b4312645
B = L[3] = (91cea910 + b4312645 + 785a04c3) <<< (A + B)
= be59d418 <<< 2c8b2b08 = 59d418be
. . .
A = S[25] = (4e0d4c36 + f66a1aaf + 6d7f672f) <<< 3
= b1f6ce14, <<< 3 = 8fb670a5,
B = L[1] = (cdfc2657 + 8fb670a5 + 6d7f672f) <<< (A + B)
= cb31fe2b <<< fd35d7d4 = e2bcb31f

Encryption

The input block to RC5 consists of two w-bit words given in two registers, A and B. The output is also placed in the registers A and B. As explained in [6], RC5 uses an expanded key table, S[0, 1, . . . , t − 1], consisting of t = 2(r + 1) words. The key-expansion algorithm initializes S from the user's given secret key parameter K. However, the S table in RC5 encryption is not like an S-box used by DES. The encryption algorithm is given in the pseudo code as shown below:

A = A + S[0];
B = B + S[1];
for i = 1 to r do
A = ((A ⊕B) <<< B) + S[2i];
B = ((B ⊕A) <<< A) + S[2i + 1];
The output is in the registers A and B.

The decryption routine is easily derived from the encryption routine. The RC5 encryption algorithm is illustrated as shown in Figures 1, respectively.

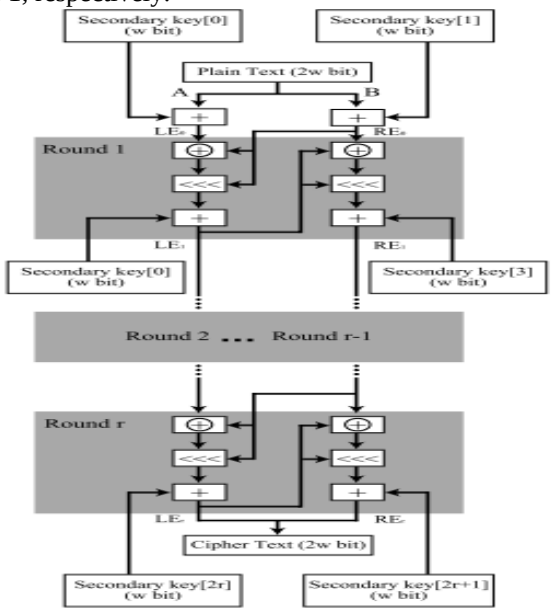

Figure 1: The RC5 encryption algorithm.

Proposed Method For Implementation Rc5 on FPGA

One of complex operation in RC5 encryption is rotate thus must be designed and implementation with a optimized circuit now we for implementation of this opreation use a barrel shifter based on 32-bit architecture this circuit is very simple and its logic utilaization is low. Proposed code for this operation is shown in below:

```
case rot is
when 0 => x1<=x1;
when 1 => x1(0)<=x1(31); x1(31 downto 1)<=x1(30 downto 0);
when 2 => x1(1 downto 0)<=x1(31 downto 30); x1(31 downto 2)<=x1(29 downto 0);
when 3 => x1(2 downto 0)<=x1(31 downto 29); x1(31 downto 3)<=x1(28 downto 0);
when 4 => x1(3 downto 0)<=x1(31 downto 28); x1(31 downto 4)<=x1(27 downto 0);
when 5 => x1(4 downto 0)<=x1(31 downto 27); x1(31 downto 5)<=x1(26 downto 0);
when 6 => x1(5 downto 0)<=x1(31 downto 26); x1(31 downto 6)<=x1(25 downto 0);
when 7 => x1(6 downto 0)<=x1(31 downto 25); x1(31 downto 7)<=x1(24 downto 0);
when 8 => x1(7 downto 0)<=x1(31 downto 24); x1(31 downto 8)<=x1(23 downto 0);
when 9 => x1(8 downto 0)<=x1(31 downto 23); x1(31 downto 9)<=x1(22 downto 0);
when 10 => x1(9 downto 0)<=x1(31 downto 22); x1(31 downto 10)<=x1(21 downto 0);
when 11 => x1(10 downto 0)<=x1(31 downto 21); x1(31 downto 11)<=x1(20 downto 0);
when 12 => x1(11 downto 0)<=x1(31 downto 20); x1(31 downto 12)<=x1(19 downto 0);
when 13 => x1(12 downto 0)<=x1(31 downto 19); x1(31 downto 13)<=x1(18 downto 0);
when 14 => x1(13 downto 0)<=x1(31 downto 18); x1(31 downto 14)<=x1(17 downto 0);
when 15 => x1(14 downto 0)<=x1(31 downto 17); x1(31 downto 15)<=x1(16 downto 0);
when 16 => x1(15 downto 0)<=x1(31 downto 16); x1(31 downto 16)<=x1(15 downto 0);
when 17 => x1(16 downto 0)<=x1(31 downto 15); x1(31 downto 17)<=x1(14 downto 0);
when 18 => x1(17 downto 0)<=x1(31 downto 14); x1(31 downto 18)<=x1(13 downto 0);
when 19 => x1(18 downto 0)<=x1(31 downto 13); x1(31 downto 19)<=x1(12 downto 0);
when 20 => x1(19 downto 0)<=x1(31 downto 12); x1(31 downto 20)<=x1(11 downto 0);
when 21 => x1(20 downto 0)<=x1(31 downto 11); x1(31 downto 21)<=x1(10 downto 0);
when 22 => x1(21 downto 0)<=x1(31 downto 10); x1(31 downto 22)<=x1(9 downto 0);
when 23 => x1(22 downto 0)<=x1(31 downto 9); x1(31 downto 23)<=x1(8 downto 0);
when 24 => x1(23 downto 0)<=x1(31 downto 8); x1(31 downto 24)<=x1(7 downto 0);
when 25 => x1(24 downto 0)<=x1(31 downto 7); x1(31 downto 25)<=x1(6 downto 0);
when 26 => x1(25 downto 0)<=x1(31 downto 6); x1(31 downto 26)<=x1(5 downto 0);
when 27 => x1(26 downto 0)<=x1(31 downto 5); x1(31 downto 27)<=x1(4 downto 0);
when 28 => x1(27 downto 0)<=x1(31 downto 4); x1(31 downto 28)<=x1(3 downto 0);
when 29 => x1(28 downto 0)<=x1(31 downto 3); x1(31 downto 29)<=x1(2 downto 0);
when 30 => x1(29 downto 0)<=x1(31 downto 2); x1(31 downto 30)<=x1(1 downto 0);
```

when 31 => x1(30 down to 0)<=x1(31 down to 1); x1(31)<=x1(0); when others => x1<=x"00000000";
end case;

Proposed method for implementation RC-5 algorithm is based on one ASM chart. Proposed ASM chart is according to RC-5 algorithm. Proposed ASM chart for RC-5 algorithm is shown in Figure 2. We applied pipelining technique in proposed ASM chart.

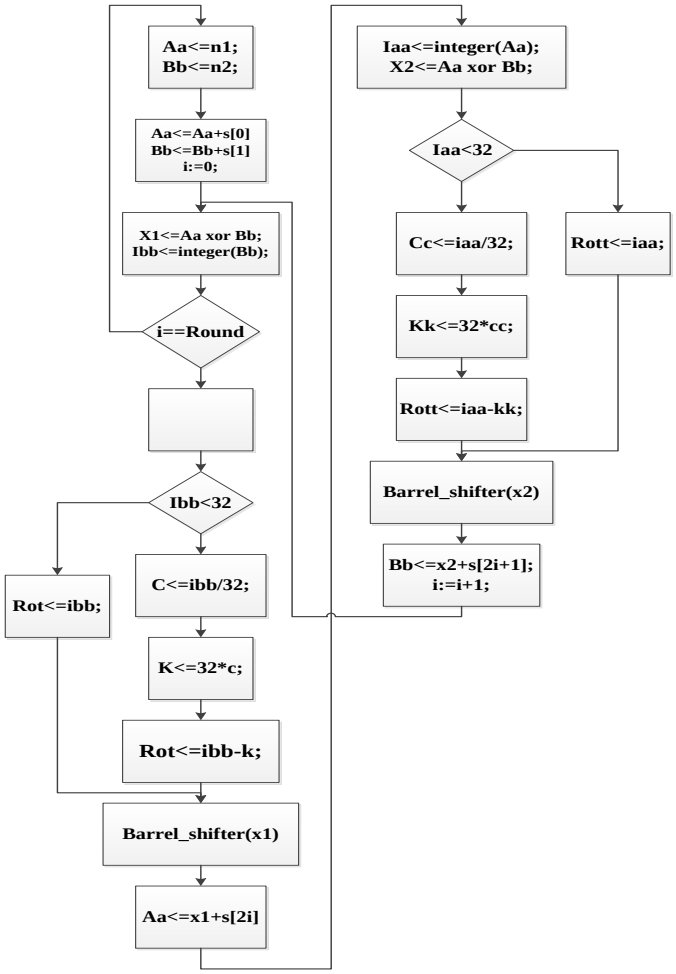

Figure 2: Proposed ASM chart for implementation RC-5 algorithm

The proposed ASM chart is based on below pseudo code:

A = A + S[0];
B = B + S[1];
for i = 1 to r do
A = ((A ⊕B) <<< B) + S[2i];
B = ((B ⊕A) <<< A) + S[2i + 1];

Comparison

We design a FPGA implementation of the 64-bit RC5 encryption algorithm. In this paper proposed method has been writed by VHDL hardware description language. In order to get actual numbers for the hardware usage thus this work was synthesized and implemented using Quartus II 9.1 software, Stratix II FPGA to target device EP2S15F484C3. Table I shows logic utilization of proposed method and Table II shows logic utilization of other works.

Conclusion

We have designed and implementation hardware realization of the RC5 encryption on FPGA. Proposed method is based on a new behavioral level design. Approaches used for increase performance are include pipelining technique, rotate operation is implemented with barrel shifter, also we implement total of algorithm in one ASM chart. Proposed method have more speed than other work.

References

[1] Ian Howson, "A Cost/Performance Study of Modern FPGAs in Cryptanalysis", thesis Bachelor of software Engineering, Electrical and Information Engineering University of Sydney, October 2003.

[2] Dirk Koch, Matthias K¨orber, and J¨urgen Teich, "Searching RC5-Keys with Distributed Reconfigurable Computing", This work was partly supported by DFG (Deutsche Forschungsge-meinschaft) under grant Te163/10-2

[3] Masaya Yoshikawa, Koichi Sakaue, "Dedicated hardware for RC5 cryptography and its implementation", This research was supported by Japan Science and Technology Agency (JST), Core Research for Evolutional Science and Technology (CREST).

[4] J. Buček, J. Hlaváč, M. Matušková, R. Lórencz, "Cost-Effective Architectures for RC5 Brute Force Cracking", Czech Technical University in Prague, Acta Polytechnica Vol. 45 No. 2/2005.

[5] Hua Li; Jianzhou Li; Jing Yang, "An efficient and reconfigurable architecture for RC5", Conference of Electrical and Computer Engineering, 2005. Canadian, IEEE, pp. 1648-1651, 2005.

[6] Man Young Rhee, "Internet Security Cryptographic Principles, Algorithms and Protocols", published in Wiley, 2003.

Table I: logic utilization of proposed method

Implementation	device	Combinational ALUTs	registers	Logic utilization	F_{Max}(MHz)
Proposed method	EP2S15F484C3	1787	440	17%	175.69

Table II: logic utilization of [2] and [3]

Method	device	slices	LUTs	FFs	F_{Max}(MHz)
[2]	XC4VLX25	9388(87%)	---	---	---
[3]	XC5VLX30/50	2488(51%)	8893(46%)	2281	100.8