From: "Mang Erica" <emang@keysys.ro>
To: <AESround2@nist.gov>
Subject: RC6
Date: Thu, 11 May 2000 15:26:13 +0300
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook IMO, Build 9.0.2416 (9.0.2910.0)
Importance: Normal
X-MimeOLE: Produced By Microsoft MimeOLE V5.00.2314.1300

My name is Erica Mang and I am assistant professor at the University of Oradea, Romania.
I write you now to inform you that I implemented the RC6 cipher in a circuit named CRIPTOR. The circuit has off line self-test facilities. In the two files I attached this message I made an analysis of suitability for pseudorandom BIST for the RC6 Cipher, and present than shortly the hardware implementation of the cipher. The circuit was implemented first time last year in my laboratory, in Verilog and now in VHDL. For this implementation I used Xilinx Foundation Series 1.5i Software and the VIRTEX XCV1000 board family.

Yours sincerely,

Erica Mang

CRIPTOR1.0. VLSI Implementation of the RC6 Block Cipher

Erica Mang

Computers Department,
University of Oradea, 5 Armatei Romane Str., 3700, Oradea, Romania
E-mail: emang@keysys.ro

Abstract

In 1997, the National Institute of Standards and Technology (NIST) initiated a process to select a symmetric-key encryption algorithm to be used to protect sensitive Federal Information. In 1998 was announced the acceptance of fifteen candidate algorithms, and in 1999 five of them where selected. One of them is the RC6 cipher-block. In this paper I present a hardware implementation of this version of RC6 algorithm using VHDL Hardware Description Language. For this implementation we used Xilinx Foundation Series 1.5i Software and VIRTEX XCV1000 board family.

The RC6 block chipper is a fully parametrized family of encryption algorithms. RC6 is more accurately specified as RC6-*w/r/b* were the word size is *w* bits, encryption consists of *r* number of rounds and *b* denotes the lengths of the encryption key in bytes.

Since the AES submission is targeted at $w = 32$ and $r = 20$, we implemented this version of RC6 algorithm, using a 32 bits word size, 20 rounds and 32 bytes (256 bits) encryption key lengths.

In this paper we present a hardware implementation of this version of RC6 algorithm using VHDL hardware description Language. For this implementation we use Xilinx Foundation 1.5i Software and VIRTEX XCV1000 board family. We chose this board for its characteristics: more than one million equivalent gates and 512 input/output buffers.

In VHDL Hardware Description Language (the IEEE 1076 standard from 1987, reviewed in 1993) we have to define two elements in the code: the entity and the architecture. The entity in VHDL describes the interface to a hierarchical block, without defining its behavior. An architecture is always associated with an entity and it defines the behavior of the entity.

Because the board capacity did not allow us to design both processes of encryption and decryption we had to design two codes for two boards. In figure 1 we have the block structure for encryption/decryption circuit.

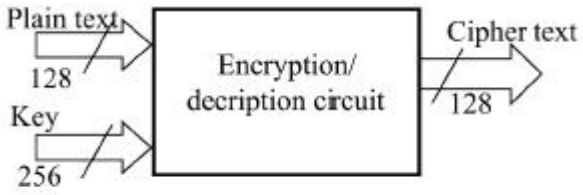

Figure 1. The block structure for encryption/decryption circuit

Both encryption and decryption circuits consist of three components: one module for subkeys generation, one for encryption/ decryption for each round and the command structure for encryption/decryption.

In figure 2 we have the circuit structure with every block that was mention before. Subkeys generation module for RC6 algorithm is the same for encryption and for decryption process. Subkeys generation module is part of the command structure for encryption/ decryption.

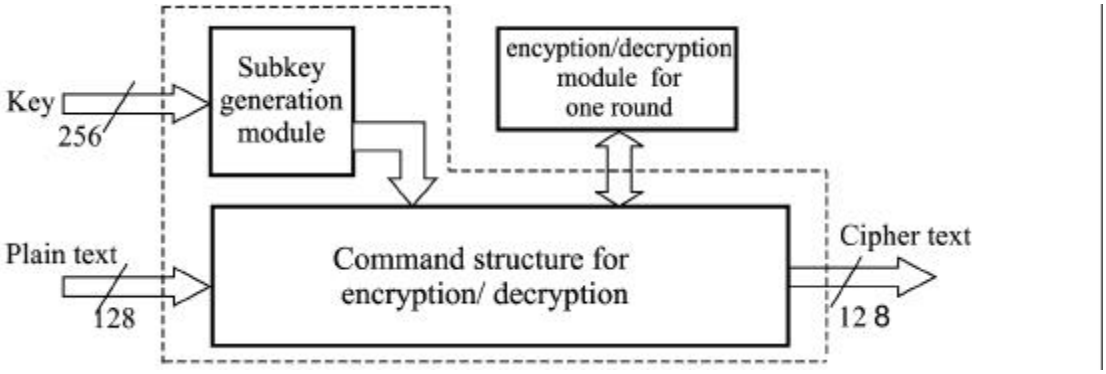

Figure 2. The encryption/decryption circuit

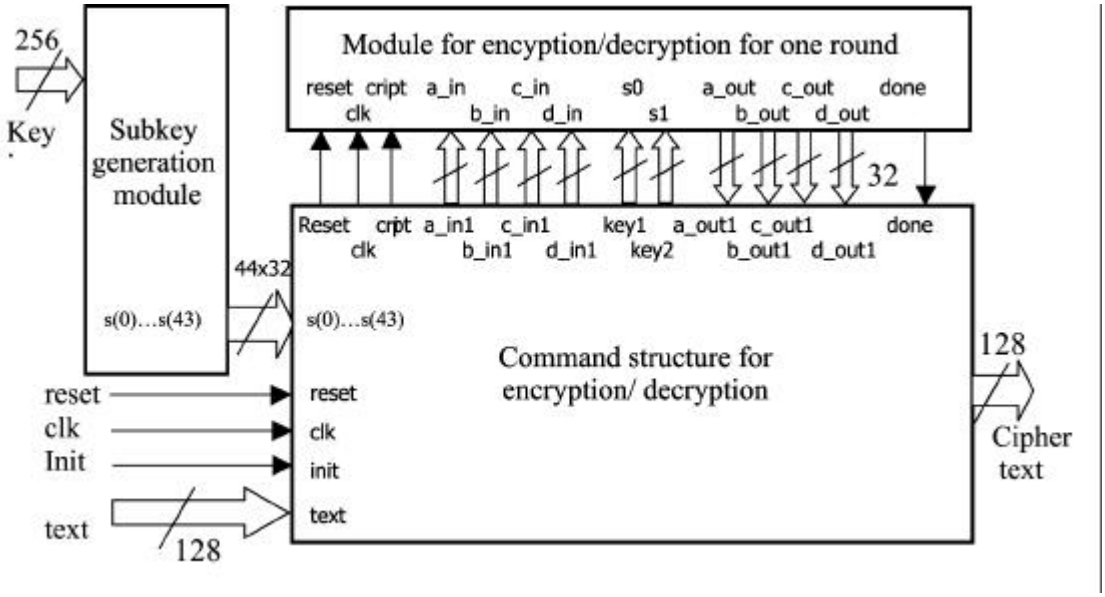

Figure 3. Detailed structure for encryption circuit

The command structure calls for every round the module for encryption/decryption for each round. The command structure sets the value for the four registers and for the two subkeys needed for the rounds calculation. The module for encryption/decryption for each round receives those data and after it finishes the calculation return the data to the command structure. Subkeys are available for the command structure from subkeys generation module.

In figure 3 we have the detailed structure with all the signals requested in the encryption circuit.

The command structure calls the module for encryption for each round, sets all data require and sets *cript* signal. After the module for encryption for each round finish the calculations, the results are set on the module outputs and in the meen time sets *done* signal. That means that the command structure has available data from the module.

After twenty rounds, on the encryption circuit outputs we have the cipher text.

We implemented the RC6 algorithm using Finite State Machine (FSM). The general architecture of an FSM consists of combinational block of next state logic, state registers, and combinational output logic.

Each module from figure 3 is described using an FSM. There are many ways to describe a finite state machine in VHDL. We use a process containing a case statement. The state of the machine is stored in a state

variable, and the possible states are represented with a user-defined enumeration type. The type declaration gives symbolic names to each of the states, but say nothing about their hardware implementation.

Finite state machine must be initialized by means of an explicit reset signal (*reset*). Otherwise, there is no reliable way to get the VHDL and gate level representation of the FSM into the same known state, and thus no way to verify their equivalence. The description of a finite state machine consists of a process, synchronized on a clock edge (*clk*).

The state transition diagram for encryption command structure is shown in figure 4. The possible states are: *idle*, *key_gen*, *call_cript*, *new_round*, and *final*.

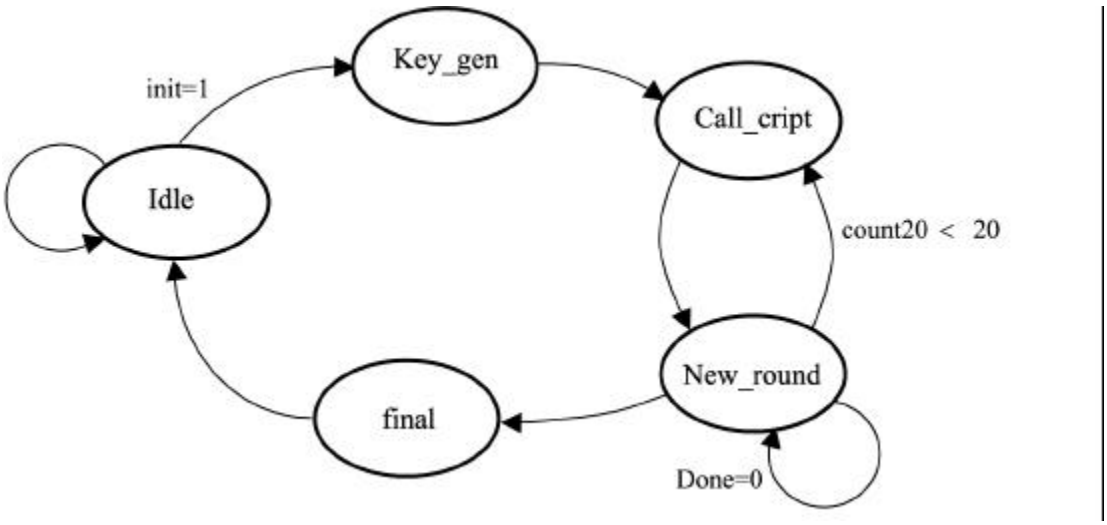

Figure 4. The state transition diagram for encryption command structure

All state transitions occur on a rising edge of a global master clock (*clk*). Some of the transitions depends on signals such us *init, count20* and *done*.

Here we make a short description of the way that the state machine from figure no. 4 works. *Idle* is a waiting state, the initialization state for the finite state machine. On the first rising edge of the clock, after the *init* signal is set *key_gen* becomes current state. In this state the variable are initialize and the subkeys table is generate. On the next rising edge of the master clock *call_cript* becomes current state, *cript* signal is set, the data (plaintext) from A, B, C, D registers and the two subkeys for the first round are available for the module for encryption for each round. Becouse *cript* signal is set the module for encryption for each round leaves *idle* state. On the next rising edge the *cript* signal is reset and *new_round* becomes current state. The FSM leaves this state only after the module for encryption for each round sets *done* signal. Depending on the value of *count20* counter, the next state is *call_cript* (if the counter is less then 20) or the next state is *final* (if *count20* is 20). The counter *count20* shows the round number we reach. If current state is *call_cript* the process works for the next round else, if current state is *final* on the outputs we have the ciphertext.

Reset and *init* are external asynchronous signals, first for general reset and the second for starting the encryption process.

The architecture for this command structure consists of processes. In VHDL a process contains sequential statements. While each process executes its statements in sequence, multiple processes interact with each other concurrently. The command structure for encryption have the next processes: *syncronous_p* process synchronizes the transitions to the next state with *clk* signal and follow the value of *reset* signal, *async_p* process describe the state trasition diagram. *Gen_keys_p* initiate the 44 subkeys for

encryption, *encryption_p* control the data transfer between the comand structure and the module for encryption for each round, *encryp_p* set or reset the *cript* signal when it is necessarily, *counter20_p* increase the *count20* counter and the last one, *final_p* puts on the outputs the chipertext.

The subkeys generation module is also design using finite state machine. The finite state machine is shown in figure 5. The subkeys generation module consists of following states: *idle, initialization, gen_keys* and *final*.

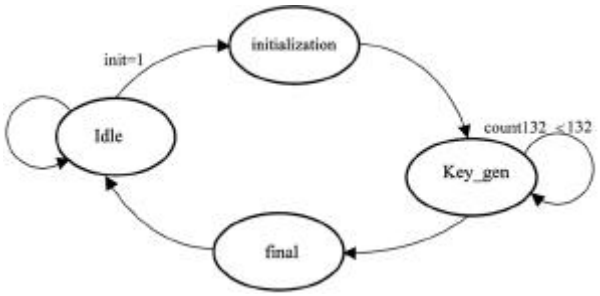

Figure 5. The state transition diagram for subkeys generation module

From *idle* state, on the first rising edge of the clock after the signal *init* is set, *initialization* become current state. On *initialization* state all variables takes their initial values. On the next rising edge of the clock *gen_keys* become current state. The module leaves this state only after all subkeys were generated when the *count132* counter reaches value 132. This value is calculated from the next formula of the RC6 algorithm:

$$v=3 \times max\{c, 2r+3\}$$

where *c* is the number of blocks of 32 bits of the key, *r* is the number of rounds. For our version *c*=8 and *r*=20. Because those calculations are simple we made them in order not to overload the board with unnecessary gates. When the counter reach the 132 value, the next state is *final.* In this state all subkeys are generated and they are ready to be use in the encryption process.

The architecture for subkeys generation module consists of the following processes: *sync_p, async_p* similar with the processes with the same names from command structure, *counter_p* used to count the iteration value (from 0 to *v*) and *gen_subkey_p* for computing subkeys using the for loop from algorithm description displayed bellow:

```
for s=1 to v do
{
  A=S[i]=(S[i]+A+B)<<<3
  B=L[j]=(L[j]+A+B)<<<(A+B)
  i=(i+1) mod (2r+4)
   j=(j+1) mod c
}
```

The *for* loop was implemented using FSM and the *count132* counter. The FSM leaves *gen_keys* state only when s (*count132*) reachs the maximum value (132). This condition is writen in *async_p* process:

```
when gen_keys =>
  if count132=132 then
     next_s=final;
  else
     next_s=gen_keys;
  end if;
```

Fixed rotations were made using concatenation operator '&'. Variable rotation were made using a case syntax and concatenation operator:

```
when 7 =>
 temp:=interm2(24 downto 0)&
       interm2(31 downto 25);
```

where temp and interm2 are internal variable of *gen_subkeys_p* process.

The subkeys generation module is the same for encryption and decryption. This module is called only when we change the encryption or decryption key.

The module for encryption for each round is declared as a component at the beginning of the command structure. The command structure calls the module for encryption for each round using following lines:

```
u1:m_cript
port map (reset, clk, crip, a_in1,
   b_in1, c_in1, d_in1, key1, key2,
   a_out1, b_out1, c_out1, d_out1,
   done);
```

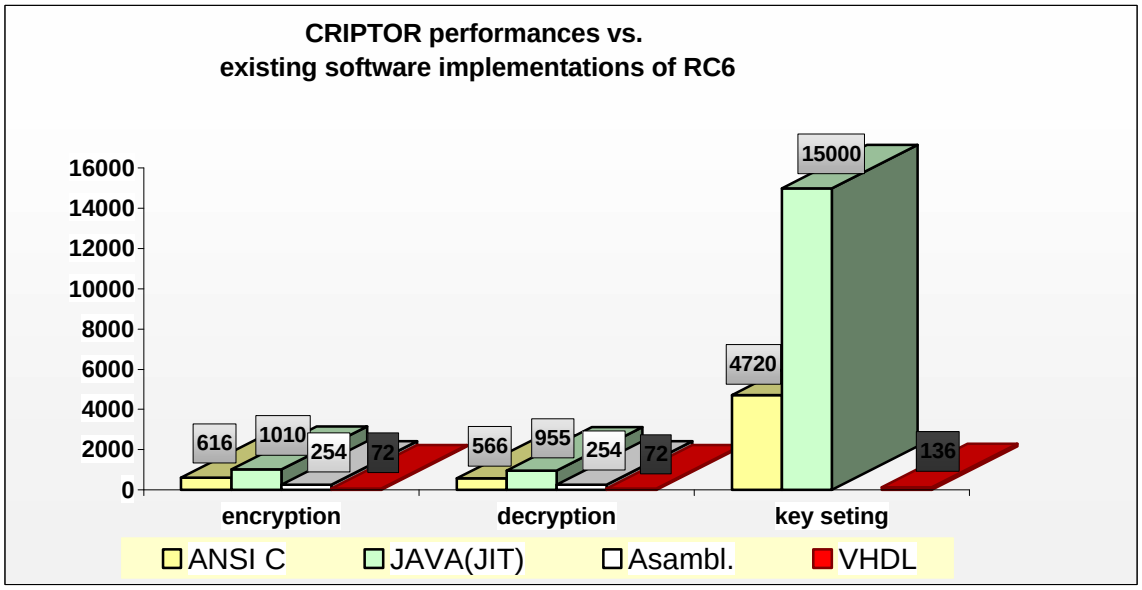

Chart 1. Comparison of CRIPTOR 1.0 performances with other known software implementations

The signals from the port map match, in the same order, with the signals from entity declaration of the module for encryption for each round.

The module for encryption for each round is computing the next lines of the algorithm:

```
t=(Bx(2B+1))<<<lg w
u=(Dx(2D+1))<<<lg w
A=((A⊕t)<<<u)+S[2i]
C=((C⊕u)<<<t)+S[2i+1]
```

also design using finite state machine. *w* is the number of bit from the register (32 for our chip, meaning that both rotations are fixed and are with 5 bits to left. This rotation is implemented also using the concatenation operator. Multiplications are implemented using shift left operations.

We already presented here the design for encryption. The structure for decryption is similar with this. The keys used for decryption are the same with the keys from encryption but there are used in opposite order. There is no difference between those two processes in terms of time performance.

In chart 1 we show the performances of the CRIPTOR1.0 circuitry by comparing them with the dates offered by RSA Laboratories regarding the known software implementations [Biha-99] [NBDDFR-99] [SKWWHF-99]. In this chart we specify the number of clock cycles needed for encryption, decryption and key settings. It can be observed the high performances of our circuitry.

Now we are working for a new implementation of the cryptochip, with better time performances for encryption, decryption and key settings.

This structure that I presented, was than a little modified like in figure 6. So, the cryptochip will have now also off-line test capabilities. The new structure I proposed will need about 20% more circuits.

As depicted in figure 6, a pseudorandom master key and a fixed set of pseudorandom input data are applied directly behind the input pads instead of as external inputs during self-test. Signature analysis is applied on data just before they leave the chip. The self-test controller simulate normal operation for all other hardware subblocks and is thus the only unit aware of self-test operation. The advantages of this scheme are twofold. First, the input is 64 bits wide, but the input port is only a 16-bit bus. A pseudo-random patern of 16 bits width can be generated faster and with less overhead than

one of 64-bit size. Second, all the inputs, subkey generating and scheduling, and outputs circuitry is included in the self-test process.

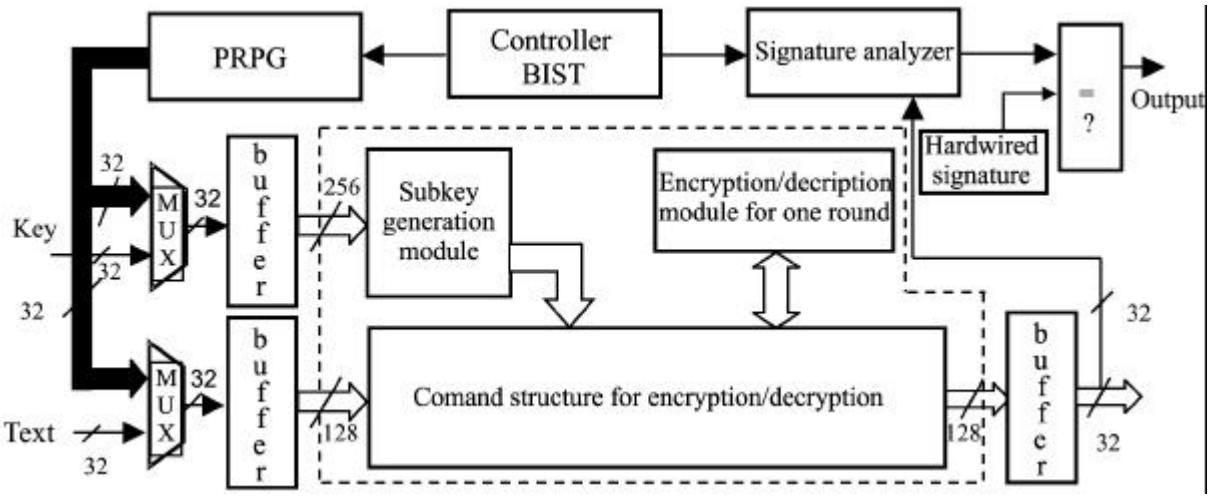

Figure 6. Off-line BIST scheme for CRIPTOR 1.0

References

[Biha-99] E. Biham. A note on comparing the AES candidates, The Second AES Conference, March 22-23, pag 85-92; 1999.

[NBDDFR-99] J. Nechvatal, E. Barker, D. Dodson, M. Dworkin, J. Foti, E. Roback – Status Report on the First Round of the Development of the Advanced Encryption Standard, 1999.

[RRSY-98] R. Rivest, M.J.B. Robshaw, R. Sidney, Y.L..Yin, The RC6 Block Cipher, M.I.T.Laboratory for Computer Science, RSA Laboratories, 1998.

[SKWWHF-99] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, Performance Comparision of the AES Submisions, The second AES Conference, pag 15-34; 1999.

[Xili-98a] Xilinx Coorporation. Foundation Series User Guide, 1998.

[Xili-98d] Xilinx Coorporation. Hardware User Guide, 1998.

[Xili-98g] Xilinx Coorporation. VHDL Reference Guide, 1998.

[Yarb-97] J. Yarbrough. Digital Logic, Applications and Design, Oregon Institute of Technology, West Publishing Company, 1997.