International Journal of Engineering Research and General Science Volume 3, Issue 5, September-October, 2015
ISSN 2091-2730

HARDWARE IMPLEMENTATION OF RC4 WITH TWO BYTES PER CLOCK CYCLE THROUGHPUT

Asha Krishna U K
Dept. of Electronics and Communication Engineering
MACE, Kothamangalam, Ernakulum.
Email:ashaushakrishna@gmail.com,+91-7356681700

Abstract- RC4 is very widely used as encryption algorithm in practical software applications. It comes under the category software stream ciphers since they are best suited in software platform. In this work, the study of RC4 stream ciphers in hardware platform is done. Basic RC4 output is improved to 2 RC4 key stream bytes per clock cycle, by merging the ideas of two technics. The technics involved in this are hardware pipelining and loop unrolling. All three of the proposed methods are designed in VHDL description, synthesized with Xilinx ISE Design Suit 14.1 and implemented in Spartan 5 FPGA trainer board.
Keywords- Loop unrolling, hardware pipelining, encryption, stream ciphers,RC4 keystream, throughput, pipelined-A, cipher.

1. INTRODUCTION

Encryption is the technology used to transmit data securely through a medium where third parties are present. So it enables secret communication. In encryption data to be send is converted to apparent nonsense in the transmission end using a secret key. This nonsense information (converted data or converted plain text) is send over the transmission medium to the receiver side. Any third person who receives this data can't understand any useful information from this cipher text. But the desired recipient have the key that used in encryption. The receiver can read the plain text using the same key and algorithm. A cipher is the sequence of steps that follows for performing encryption. If a cipher algorithm is using same key for encryption and decryption then such cipher is called symmetric cipher. RC4 is a symmetric key cipher.

Before RC4, the stream cipher technology was focused around linear feedback shift registers. The analysis of their security was an attractive topic because the device was very easy to study from a mathematical point of view.

In 1987, RC4 was developed by Ron Rivest. RC4, due to its byte operation is friendlier than linear feedback shift registers. Simplicity, compact implementation and reduced error prone are the reasons behind the popularity of RC4 stream ciphers. Despite of much research and many attacks, it's still secure enough for many applications.

In this paper, several hardware implementation aspects of RC4 is studied with respect to its efficiency. The paper presents three designing methods which is very fast to produce RC4 key stream.

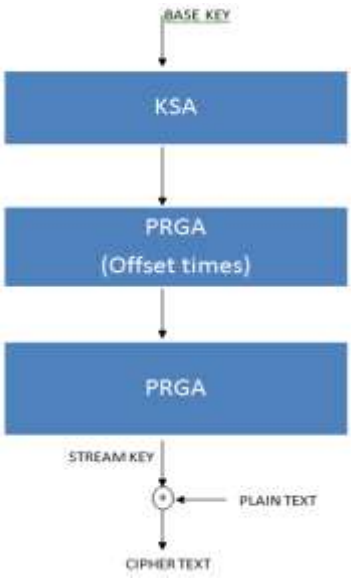

Fig.1. flow chart of RC4algorithm

2. BASIC FRAME WORK OF RC4 STREAM CIPHER

One can say RC4 as a finite state machine which have a finite number of internal states. Internal states will change according to a function of current state. The output stream is completely determined by the internal states of RC4. At first, the internal states are loaded with some initial values. are changed randomly according to some initial values. "key" is the input to the cipher. It can be variable in size. The state change of the finite state machine is completely depends on this input "key".

The complete RC4 stream cipher has 2 basic parts. First part is key scheduling algorithm. In short we can call it as KSA. It is this part where we give input. The input is a "key " which uses to randomize the internal states of RC4. Second part is called pseudo random generation algorithm (PRGA). It produces the output stream of the cipher, as a function of the internal states, after been shuffled by both KSA and PRGA.

Key scheduling algorithm is also called as key stream generation algorithm. It is Initialization:

for i = 0 to 255 do

S[i] = i

K [i] = KEY [i mod keylength]

Initial permutation

j = 0

for i = 0 to 255 do

$j = [j + S[i] + K[i]]$ *mod 256*

Swap $[S[i], S[j]]$

In this, i and j are two word sized registers. They are used to hold the index. KEY is the input to the KSA and is a variable. Here t is a register. We store the repeated KEY in it. Key length is the length of the KEY. S[] is called as substitution-box or S-box or simply S. it is basically an array of 256 words for 8-bit RC4 stream cipher. In initialization process the S-box contains 0 to 255 in its 9 to 256 locations. This is the initial state in the state machine. Then, according to the values in the word registers, i and j, the S-box values get permuted. The word register j is calculated with t-register which is a function of input KEY. So the S-box permutation is related to the input KEY. So we can say that the output of the KSA is the shuffled internal states of the S-box according to the KEY.

Pseudo random generation algorithm (PRGA) is

i=j=0

while (true) do

i=(i+1) mod 256

$j = [j + S[i]]$ *mod 256*

Swap $[S[i], S[j]]$

Temp = $S[S[i] + S[j]]$*mod 256*

Output = S [temp]

In PRGA the input is the random internal states . And in PRGA the S-box is again permuted to increase the randomness. And then output is generated. Output is a function of arithmetic addition of repeatedly permuted internal states.

All the addition in the algorithm is modulo-256 addition for the 8_bit RC4.

The output is then XOR-ed with the plain text(byte per byte) to form the encrypted text or cipher text at the sender. In the receiving side, the receiver will also have the information about the KEY. The sender and the receiver have the same KEY, because the RC4 is a symmetric stream cipher. Using the key the receiver will also generate the permuted internal states and the output stream. The cipher text which is received will XOR-ed with the output stream (byte per byte) to get back the plain text.

- RC4 generate a stream of output and each output byte is used to XOR-ed with consecutive bytes of plaintext.
- The paper proposes three methods to improve the efficiency of rc4 in terms of throughput.
- Design 1 can produce one byte per clock cycle. Loop unrolling is the technique we used in this.
- Design 2 can produce two bytes per clock cycles.
- Loop unrolling and hardware pipelining is the technique we used in METHOD 2.
- Design 3 is also produce two bytes per clock cycles. But it can release 32 bit or 64 bit per each iterations of PRGA.

3. DESIGN 1: ONE BYTE PER CLOCK CYCLE

We can see that two consecutive cycle's output bytes of RC4 can formed together for two consecutive plain-text bytes when the algorithm is analyzed in detail.

The first loop of PRGA algorithm is

$i1 = i0 + 1$

$j1 = [j0 + S0[i1]]$

$Swap\ [S0[i1], S0[j1]]$

$Z1 = S1[S0[i1], S0[j1]]$

The second loop of PRGA algorithm is

$i2 = i1 + 1 = i0 + 2$

$j2 = j1 + S1[i2]$

$= j0 + S1[i2] + S0[i1]$

$Swap\ [S1[i2], S1[j2]]$

$Z2 = S2\ [S1[i2], S1[j2]]$

Here Z is the output formed after EACH iteration and Z1 is the first output, Z2 is the second output, and so on. Likewise S0 is the initial S-box or non-permuted internal states. S1 is formed after one permutation and so on.

From the first and second loops of PRGA, we can see that i1 is formed from i0 and i2 is formed from i1 by increment i0 by 1. But in detailed analysis we can see that i2 can calculated from i0 by incrementing it by 2. Likewise j1 value is calculating from j0 and S0, and j2 is calculated by j1 and S1. But j2 can also calculated by j0 and S0. So from this analysis, we can come to the conclusion that i1, i2, j1 and j2 can calculate with the same set of input. So they can calculate in a single clock cycle. i1 and j1 is needed for the first output Z1 and i2 and j2 is needed for the second output Z2. So actually we can calculate Z1 and Z2 in a single clock cycle. So we can calculate two consecutive outputs in a single clock cycles.

3.1 Designing components

The S-box keeps the internal states of the cipher. It is eight bit register that used for holding each internal state and there are 256 such registers in total to form the whole S-box. Each state is accessed through a 256 to 1 multiplexer and the output of the multiplexer is set by the inputs i1, i2, j1 and j2.

The overall design can be divided into four separate parts. One part is to calculate i1 and i2. Only clock is needed for the first design with a initial state. For second design j0, i1 and i2 are the inputs. Third design is for swapping the S-box. The locations to be swapped is determined by the I and j values. Forming the output stream is the final design step.

3.1.1.First step of design1 : Design for calculating i1 and i2.

For formation of i1 and i2, two synchronous 8 bit counters is used and each bit is formed by a flip-flop. We can say that i1, i3, i5 etc. will be odd numbers where as i2, i4, i6 etc. will be even numbers. So we can say that, for i1 the least significant bit will always be a one. And for i2 the least significant bit will always be a zero. So actually a seven bit counter is only needed and the least significant flip-flop does not need a clock. It always carries a constant value, 0 for i2 and 1 for i1.

Initial value of i-counter is set differently for i1 and i2. It is 00000001 for i1 and 00000010 for i2. Calculation of i1 and i2 serves for first two rounds of KSA as well as PRGA.

Thereafter for every other clock pulse which gets for i1 and i2 the output is produced.

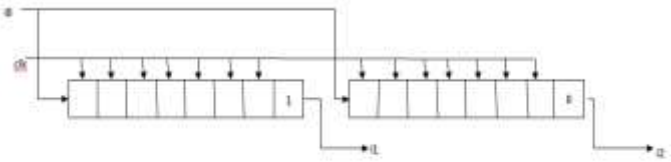

Fig.2. diagram of i1 and i2

i value is used to calculate the j value. Both are word sized registers used to provide the addresses from where values needed to be shuffled. The circuit diagram for i1 and i2 is there in fig.1.

3.1.2 Second step of design 1 : Design for calculating j1 and j2

Like I, j1 and j2 can also calculated simultaneously. The input to j-design is i1 , i2 and j0. j0 is the initial value for the j1 and j2. J0 is the initial state and it is same for both j1 and j2.

The two computed j-values are stored in two 8 bit registers. One component of circuit is 2-input adder which can construct by carry look ahead adder. Carry input is not needed and we can set it to one always. And only sum output I needed as output. Carry output can be set to 0 always or can avoid calculating carry which is a better method.

For calculating j, along with j0, s0 is also needed. S0 is the S-box which contains initial permutation. That means the 0^{th} register of total 256 contain a 0, 1^{st} register contain 1, 2^{nd} register contain 2,…254^{th} register contain 254 and 255^{th} register contain 255. We always want S-box values of the location specified by i-values. So for accessing S-box a 256 to 1 multiplexer is used. Two such multiplexer is needed because i1 and i2 is simultaneously access S-box to calculate j1 and j2 simultaneously.

There two cases to calculate j2

$j2 = j0 + S0[\ i1\ ] + S1\ [\ i2\ ]$

It can analyze in more detail like

if i2 ≠ j1 then

j2 = j0 + S0[i1] + S1 [i2]

if i2 = j1 then

j2 = j0 + S0[i1] + S1 [i1]

So for checking whether i2 equal to j1 or not, a comparator is used. A 8-bit comparator is used to compare and it will produce a single bit output according to the values. According to the comparator output j2 select its inputs. Actually both j2 is produced with two separate three input adder and produces both 8-bit outputs. We need to perform modulo-256 addition. So there is no need for a carry out in this. Both possible j2 values are giving as inputs of a 2-input multiplexer. The control input of the multiplexer is the output from the comparator

For designing the three input adder, denote the k-th bit of the j0, S [i1], S1 [i2] (either S0 [i2] or S0 [i1])by a_k, b_k, C_k . K can take any value from 0 to 7.from these input we can calculate 2 nine bit values, R and C.

$R_k = XOR\ (\ a_k, b_k, c_k\ )$ for k = 0 to 7

$R_8 = 0$

$C_0 = 0$
$C_k = abk\text{-}1 + bck\text{-}1 + ack\text{-}1$ for k = 0 to 7

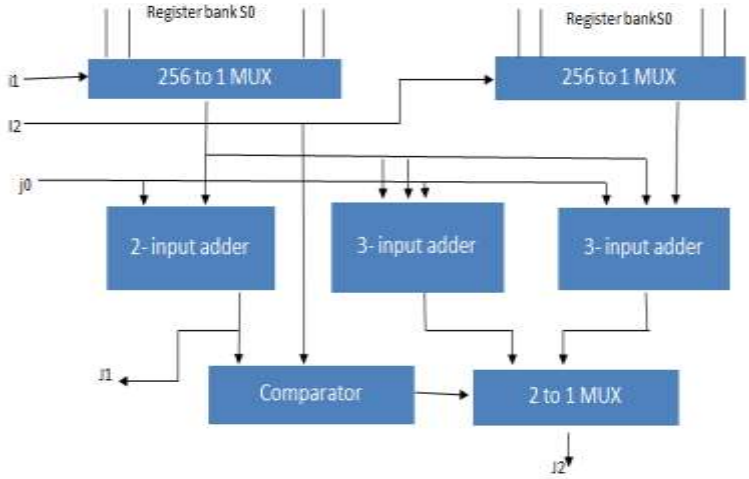

Fig.3. diagram to calculate j1 and j2

Even though the addition contains two 9 bit values, we can simply add them together to form a nice bit value. We need to perform a modulo-256 addition. We need to take out only two 8 bit values from the circuit by discarding the 8th bit.then the two bytes are added together for the desired output.this two input addition can done by the parallel adder we constructed for calculating j1. Fig.3. shows the circuit diagram for calculating j1 and j2.

3.1.3 Second step of design 1: permutation of the S-box

S-box contains 256 eight bit values. Initial state of the S-box is containing a certain internal state. These internal state values are permuted according to i1, i2, j1 and j2. J1 and j2 index is random. The values containing in the random address locations are shuffled according to the i1, i2, j1, and j2. Which two address locations will be shuffled is determined by some conditions. Conditions are checked by comparators. Total three comparators are needed. Since the three conditions are checked to make decisions about the locations to be shuffled.

Here S0 is used to find out S1 and S1 is used to find out S2. But we know i2 and j2 also with i1 and i2. So we can calculate S2 directly from S0 without calculating S1.

The conditions to be checked and the corresponding swap operation is specified in the table.1

Data to be stored in S-box is taken from any of the four 8 to 1 multiplexer. And the output line of the multiplexer is controlled by three comparator. Each checks one condition from the total 3 conditions. The 8 to one multiplexer contains 3 control lines. It forms by combining the outputs from all three comparator.

It is better to construct the 256 eight bit registers using master slave JK-flip-flop because it will help to save data simultaneously.

Conditions and corresponding shuffles of S- box are

i2 ≠ j1 and j2 ≠ i1 and j2 ≠ j1

S0[i1] → S0[j1]

S0[j1] → S0[i1]

S0[i2] → S0[j2]

S0[j2] → S0[i2]

i2 ≠ j1 and j2 ≠ i1 and j2 = j1

S0[i1] → S0[i2]

S0[i2] → S0[j1] =S0[j2]

S0[j1] → S0[i1]

i2 ≠ j1 and j2 = i1 and j2 ≠ j1

S0[i1] → S0[j1]

S0[i2] → S0[i1] = S0[j2]

S0[j1] → S0[i2]

i2 ≠ j1 and j2 = i1 and j2 = j1

S0[i1] → S0[i2]

S0[i2] → S0[i1] = S0[j1] = S0[j2]

i2 = j1 and j2 ≠ i1 and j2 ≠ j1

S0[i1] → S0[j2]

S0[j2] → S0[j1] = S0[i2]

S0[j1] → S0[i1]

i2 = j1 and j2 ≠ i1 and j2 = j1

S0[i1] → S0[j1] =S0[i2] = S0[j2]

S0[j1] → S0[i1]

i2 = j1 and j2 = i1 and j2 ≠ j1

No permutation.

No data transfer is shuffling.

i2 = j1 and j2 = i1 and j2 = j1

It is an impossible condition, because i1 will never be equal to i2

3.1.4 Fourth step of the design: design for calculatingZ1 and Z2.

Like all other designs Z can also make use of loop unrolling. So we can find out Z1 and Z2 in a single clock cycle. For finding them we need S0, S2, i1, i2, j1, j2. Those values are available.

Z1 is selected from the internal states of the S-box after permutation. The factors deciding which internal state will come to the Z1 is the internal states of identity permuted internal states selected using the word registers j1 and i1.

if S0[j1] + S0[i1] = j2 then

Z1 = S2 [i2]

if S0[j1] + S0[i1] = i2 then

Z1 = S2 [j2]

if S0[j1] + S0[i1] ≠ j2 ≠ i2 then

Z1 = S2 [S0[j1] + S0[i1]]

S0 [j1] and S0 [i1] will added first to check the conditions. They are compared with word registersi2 and j2. We can give adder inputs to a 4 to 1 multiplexer. And comparator outputs to its control inputs. According to the address specified in the output of multiplexer, the data is taken out from the register bank. It can do by a 256 to 1 decoder. The control line of the decoder is given from the output of the multiplexer.

Z2 = S2 $\left[S2[i2] + S2[j2]\right]$= S2 $\left[S1[i2] + S1[j2]\right]$

3.2 The complete circuit

Fig.4. shows the complete design. It includes all 4 design components. First circuit works in the trailing edge of first clock cycle. In the next cycle circuit 2 produces outputs and the output latches releases values of circuit 2 and circuit 1 to the input of circuit 3.

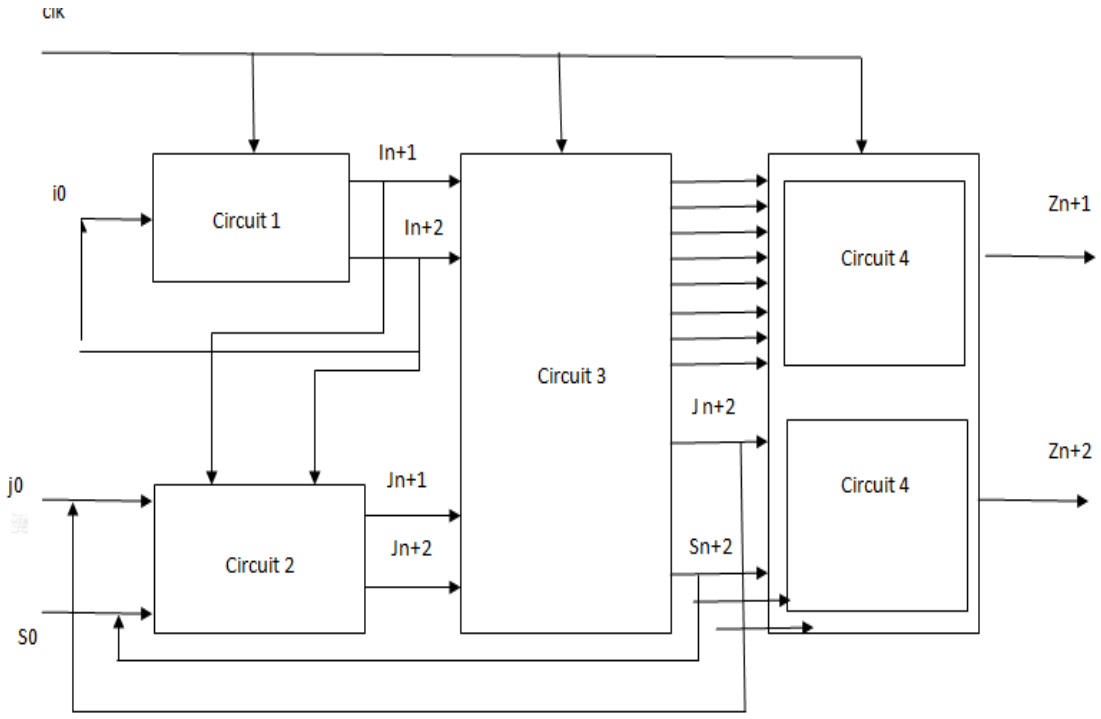

Fig.4. complete design

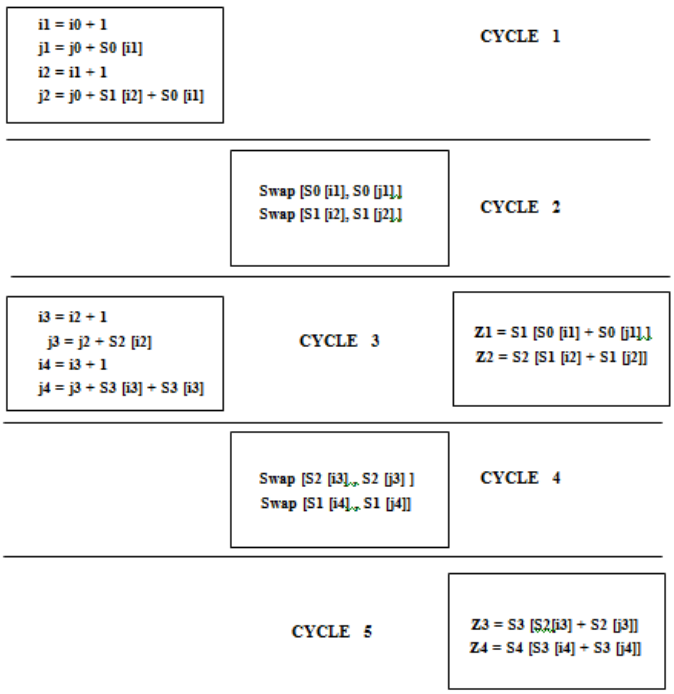

Fig.5. pipelining structure in first design

The combinational circuit in the circuit 3 works in 2nd cycle. The latch in the output of the circuit 3 activates. And in the next cycle, circuit 4 get activates and two consecutive outputs will get in 3rd clock cycle.

Along with the hardware unrolling we can see a pipeline structure in design 1. First and second circuit works together in a single cycle. Swapping done in next cycle. And outputs calculated in next half cycle. But first cycle can repeat in third One, so after

the initial delay of two half cycle we can see that our circuit is producing two outputs in alternating clock cycles. That makes it a output per cycle architecture. Figure 6 shows the diagram of the architecture.

4. DESIGN 2: TWO BYTE PER CLOCK CYCLE

Along with the concept of loop unrolling there is a technique called hardware pipelining.

In pipelining concept, in 1st cycle, calculation of i and j along with swapping is done. But index of only first loop can find. So in 1st cycle i1, j1 and S1 can find. In second cycle output byte is calculated. Only Z1 can find out. But during the 2nd cycle itself we can calculate i2, j2, and S2. By doing so we could find out next output, Z2 in 3rd cycle. That means it also provide one byte per clock cycle with an initial delay of one cycle.Fig.6.shows the structure of pipelined-architecture.

We already saw the concept of loop unrolling. It can provide two outputs in one cycle. In detail view of two concepts:

Loop unrolling steps are

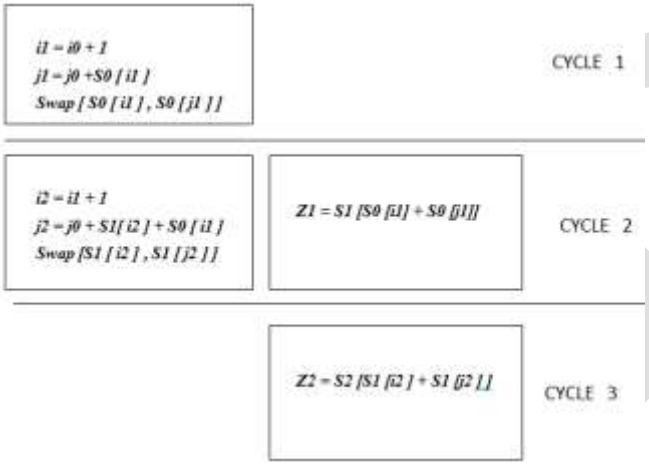

Fig.6. pipelined-A architecture

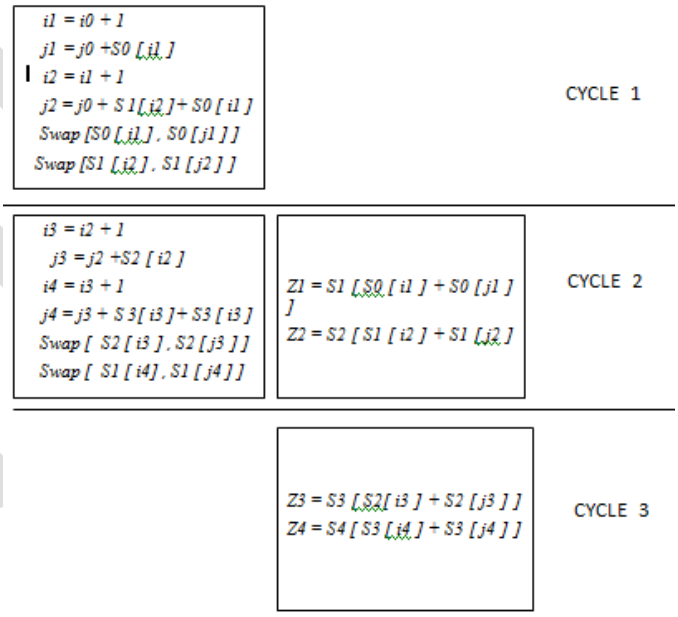

Fig.7. pipeline structure for design 2

- Calculating two consecutive values of word registers i s and j.
- Finding consecutive permutations by swapping.
- Read two consecutive output bytes Z1 and Z2 simultaneously.

Pipelined-A architecture steps are

- Calculating value of word registers i and j. then perform swap operation of internal states.
- Read output corresponding to the first step.

So the advantage of loop unrolling is that it provides two outputs in a single clock cycle. But the problem is it gets us output only in alternate clock cycles. While pipelined-A gives output in all clock cycle. But it can only give away one output per clock cycle.

So if we could combine the two concepts of loop unrolling and pipelined-A then we can produce two outputs in every single clock cycle. Pipeline structure of this concept is given in the figure 6.

Consecutive loops for KSA are

- First loop of algorithm

  $i1 = i0 + 1$

  $j1 = j0 + S0[\,i1\,] + KEY[i1]$

  $Swap\ [S0[i1], S0[j1]]$

- Second loop of algorithm

  $i2 = i1 + 1 = i0 + 2$

  $j2 = j1 + S1[\,i2\,] + KEY[i2]$

  $= j0 + S0[\,i1\,] + S1[\,i2\,] + KEY[i2]$

  $Swap\ [S1[i2], S1[j2]]$

The double swap operation starts in the first stage in this design. So we need to use pipelined registers to maintain read and write. After the swap operation Z values are reading from the internal states. Two outputs are getting from a clock cycle because of loop unrolling. The shared structure is shown in figure.8.

5. DESIGN 4: TWO 16 BIT OUTPUT PER CLOCK CYCLES

The proposed structure enables us to release 16 bits or 32 bits, in each iterations of PRGA loop. Even though the architecture keeps S-box size smaller than 2^{32} or 2^{16} . So from this we can see that output stream is increased without taking too much area. Here the size of i or j is not increasing considerably. N is the size of the counters and the S_box states. And M is the output word size.

PRGA algorithm for 8-bit word registers and S-box and 16 bit output.

$i = 0$

$j = 0$

while (true)

$i = [\,i+1\,] \bmod 2^8$

$j = [\,j + S[\,i\,]\,] \bmod 2^8$

$k = k + S[\,j\,]\,] \bmod 2^{16}$

$output = [\,S[\,[S[\,i\,] + S[\,J\,]\,] \bmod 2^8] + k\,] \bmod 2^{16}$

By incorporating loop unrolling to it we can calculate 2 outputs simultaneously. So we could get two 16 bit output in a single clock cycle. Due to pipelining we will get two outputs in all clock cycles.

Algorithm which gives two output in one clock cycle.

- First loop of algorithm

$i0 = 0$

$j0 = 0$

while (true)

$i1 = [\, i0 + 1 \,] \bmod 2^8$

$j1 = \left[j0 + S0[i1]\right] \bmod 2^8$

$k1 = \left[k1 + S0[j1]\right] \bmod 2^{16}$

$$Z1 = \left[S1\left[\left[S0[i1] + S0[j1]\right] \bmod 256\right] + k1\right] \bmod 2^{16}$$

- Second loop of algorithm

$i2 = [\, i1 + 1 \,] \bmod 2^8$

$= [\, i0 + 1 \,] \bmod 2^8$

$j2 = [j0 + S0[i1] + S1[i2]] \bmod 2^8$

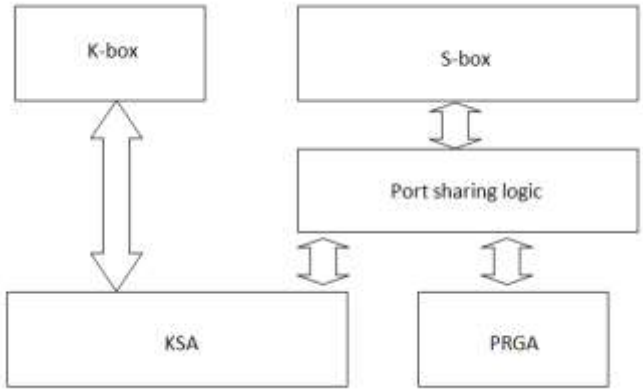

fig.8. shared access of PRGA and KSA.

$k2 = k2 + S[\, j2 \,]\,] \bmod 2^{16}$

$= k2 + S[[j0 + S1[i2] + S0[\, i1 \,]] \bmod 2^8]\,] \bmod 2^{16}$

$$Z2 = \left[S2\left[\left[S1[i2] + S1[j2]\right] \bmod 256\right] + k2\right] \bmod 2^{16}$$

However now the contents of array S does not undergo complete permutation. In RC4 , swapping is done to update output in a random manner. In this approach we use modulo 2^{16} integer addition. We update the output by replacing the internal state by a random number. The most important fact is that the size of the array is a very small fraction of the all possible numbers in the Z.

The main difference is the presence of k , a third variable. k, along with i and j. the k has two significant.

- It masks the output so that it does not simply represent value stored in the array.
- It ensure that the new value in the update step

6. CONCLUSION

RC4 is a very popular stream cipher which is very popular in the domain of cryptography. It is a software cipher. This paper is based on its hardware implementation. RC4 and three methods to improve its throughput with respect to clock cycle is done. In first design one byte per clock cycle and in second design two byte per clock cycle is obtained. In third design output stream of rather large size can be get. The size of the array is a very small fraction of all the possible numbers in the output. This variation is done without complicating or enlarging the size of the design. The four algorithm designs are coded in VHDL description, simulated with Xilinx ISE Design Suit 14.1 and was implemented in Spartan 5 FPGA trainer board.

REFERENCES:

1. Chang N. Zhang and Qian Yu, "an rc4 based light weight secure protocol for sensor networks" Department of Computer Science, University of Regina3737 Wascana Parkway, Regina, SK S4S 0A2 Canada {zhang, yu209}@cs.uregina.ca
2. Souradyuti Paul and Bart Preneel, "A New Weakness in the RC4 KeystreamGenerator and an Approach to Improve theSecurity of the Cipher?".
3. Matthew E. McKague," Design and Analysis of RC4-like StreamCiphers".
4. Guang Gong, Kishan Chand Gupta, Martin Hell and Yassir Nawaz,"Towards a General RC4-like KeystreamGenerator".
5. Ilya Mironov," (Not So) Random Shu_es of RC4".
6. Lae Lae Khine," A New Variant of RC4 Stream Cipher".
7. Sourav Sen Gupta, Anupam Chattopadhyay," High-Performance Hardware Implementation for RC4 Stream Cipher". ieee transactions on computers, vol. 62, no. 4, april 2013
8. Rick Wash ,"Lecture Notes on Stream Ciphers and RC4", ieee transactions on computers, vol. 62, no. 4, april 2013
9. Scott fluhrer, Sdi Shamir , "Weaknesses of key sheduling algorithm of rc4, ".
10. Machalis Galanis,paris Kitsos,giorgos Kostopoulos,Nickolas Sklavos,and Costas Goutis ,"Comparison of haedware implementation of sream ciphers",
11. Panu hamalainen,Marko Hannikainen, Jukka Saaarineen, Timo Hamaalaineen, "Hardware implementation of improved WEP and RC$ encryption algorithms for wireless terminals".
12. Ilya Mironov, "(Not So) Random shuffles of RC4".