EFFICIENT ENCRYPTION USING RC4 STREAM CIPHER

Ameena Abdul Salam[1], Sharon Mathew[2]

[1]*P.G student,Electronics & Communication Department, IIET, Kerala,***ameenas786@gmail.com**

[2]*Assistant Professor,Electronics & Communication Department, IIET, Kerala,***sharoncrisbin@gmail.com**

Abstract

RC4 is the most popular stream cipher in the domain of cryptology. A systematic study of the hardware implementation of RC4 is presented, and proposed the fastest known architecture for the cipher. The ideas of hardware pipeline and loop unrolling is combined to design an architecture that produces 2 RC4 key stream bytes per clock cycle. I have optimized and implemented the proposed design using verilog description, to obtain a final RC4 key stream in the respective technologies. The algorithm for the cipher is intriguingly simple, and one can easily implement it within a few lines of code. This has been promoted RC4 as a natural choice by introducing Common Boolean Logic. This is the fastest known RC4 hardware architecture. The main advantage of this design it produces two bytes per clock cycle, this reduction in clock which will help to reduce the area and increase speed of operation, thus total performance of the system which can be increase with this design.

Index Terms: *Common Boolean logic , Cryptography, Loop Unrolling, Pipelining, RC4, Stream Cipher*

--- *** ---

I. INTRODUCTION

RC4 is a popular stream cipher to generate a stream of key characters where each character of the key being added to a character of the input stream to produce an output character. This cipher is widely used in network protocols such as Secure Socket Layer, Transport Layer Security, Wired Equivalent Protection and Wireless Protocol Access. The cipher also finds applications in Microsoft Windows, Lotus Notes, Oracle Secure Standard Query Language etc. Though several other efficient and secure stream ciphers have been discovered after RC4, it is still the most popular stream cipher algorithm due to its simplicity, ease of implementation and speed. In spite of several cryptanalysis attempts on RC4, the cipher stands secure if used properly.

RC4 has two components, namely the Key Scheduling Algorithm (KSA) and the Pseudo-Random Generation Algorithm (PRGA). The KSA uses the key K to generate a pseudorandom permutation S of {0 ,1, . . .,N _ 1} and PRGA uses this pseudorandom permutation to generate arbitrary number of pseudorandom key stream bytes. A detailed account of the design strategy and circuit analysis is presented. The implementation of the design has been done using verilog description, synthesized with 65 nm technology using Synopsys Design Compiler in topographical mode.

In the proposed system a completely new design of RC4 hardware using efficient hardware pipeline and loop unrolling simultaneously. This model provides a throughput of 2 bytes per cycle in RC4 PRGA, without losing the clock performance. This is the major contribution in the RC4 design. With strict clock period constraints, we could device a model based on final design.RC4 has been used numerous forms and shapes in a majority of cryptographic solutions based on stream ciphers. The algorithm for the cipher is intriguingly simple, and one can easily implement it within a few lines of code. This has been promoted RC4 as a natural choice by introducing Common Boolean Logic. This is the fastest known RC4 hardware architecture. The main advantage of this design is only need two byte per clock cycle, this reduction in clock which will help to reduce the area and increase speed of operation, thus total performance of the system which can be increase with this design.

II. SYSTEM OVERVIEW

The design is described in terms of its components, timing and throughput analysis, and implementation ideas. We consider the generation of two consecutive values of Z together, for the two consecutive plaintext bytes to be encrypted. Assume that the initial values of the variables i, j, and S are i0, j0, and S0, respectively. After the first execution of the PRGA loop, these values will be i1, j1, and S1, respectively, and the output byte is Z1, say. Similarly, after the second execution of the PRGA loop, these will be i2, j2, S2, and Z2, respectively. Thus, for the first two loops of execution to complete, we have to perform the operations.

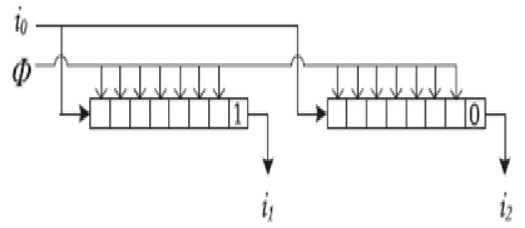

Fig- 1: Circuit to compute i1 and i2.

To store S-array in hardware, we use a bank of 8-bit registers, 256 in total.The output lines of any one of these 256 registers can be accessed through a 256 to 1 Multiplexer (MUX), with its control lines set to the required address i1, j1, i2, or j2. Thus, we need four such 256 to 1 MUX units to simultaneously read S[i]1, S[i2], S[j1], and S[j2]. Before that, it is necessary to know how to compute the increments of i and j at each level.

Step 1: Calculation of i and j. Incrementing i0 by 1 and 2 can be done by the same clock pulse applied to two synchronous 8-bit counters. The counter for i1 is initially loaded with 00000001 and the counter for i2 is loaded with 00000010, the initial states of these two indices.

Thereafter, in every other cycle, the clock pulse is applied to all the flip-flops except the ones at the LSB position values 1, 3, 5, . . . , and that of i2 assuming only the even values 2, 4, 6, . . . , as required in RC4. This is is assured as the LSB of i1 will always be 1 and that of i2 will always be 0, as shown in fig.1.For computing j2, there are two special cases as follows:

j2 = j0 + S0[i1] + S0[i2] if i2 ≠ j1, and j2 = j0 + S0[i1] + S0[i1] if i2 = i1.

The only change from S0 to S1 is the swap S0 [i1] ↔S0 [j1], and hence we need to check if i2 is equal to either of i1 or j1. Now, i2 cannot be equal to i1 as they differ only by 1 modulo 256. Therefore, S1 [i2] = S1 [j1] = S0 [i1] if i2 = j1, and S1 [i2] = S0 [i2] otherwise. In both the cases, three binary numbers are to be added.

Let us denote the k*th* bit of j0, S0 [i1], and S1 [i2] (either S0 [i2] or S0 [i1] by a_k; b_k, and c_k, respectively, where $0 \le k \le 7$. We first construct two 9-bit vectors R and C, where the k^{th} bits $(0 \le k \le 8)$ of R and C are given by

$$R_k = XOR(a_k, b_k, c_k) \text{ for } 0 \le k \le 7, R8 = 0, C0 = 0$$

$$C_k = a_{k-1}b_{k-1} + b_{k-1}c_{k-1} + c_{k-1}a_{k-1} \text{ for } 1 \le k \le 8.$$

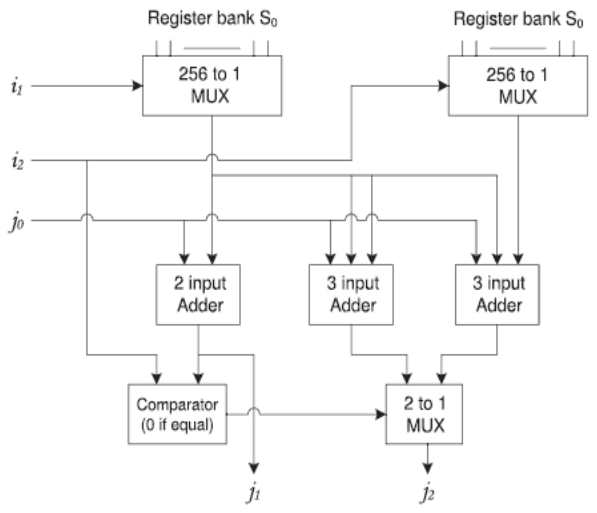

Fig- 2: Circuit to compute j1 and j2.

In RC4, all additions are done at modulo 256. Hence, we can discard the 9th bit (k = 8) of the vectors R, C while adding them together, and carry out normal 8-bit parallel addition considering $0 \le k \le 7$. Therefore, one may add R and C by a parallel full adder as used for j1.

Step 2: Swapping the S values. The two swap operations in the third row results in one of the following eight possible data transfer requirements among the registers of the S-register bank, depending on the different possible values of i1, j1, i2, and j2. We have to check if i2 and j2 can be equal to i1 or j1[7].

So, the input data (1 byte) for each of the 256 registers in the S-register bank will be taken from the output of an 8 to 1 MUX unit, whose data inputs are taken from S0[i1] , S0[j1] , S0[i2] , S0[j2] and the control inputs are taken from the outputs of three comparators comparing 1) i2 and j1, 2) j2 and i1, 3) j2 and j1.

For the simultaneous register-to-register data transfer during the swap operation, we propose the use of Master-Slave JK flip-flops to construct the registers in the S-register bank. This way, the read and write operations will respect the required order of functioning, and the synchronization can be performed at the end of each clock cycle to update the S-state.

Step 3: Calculation of Z1 and Z2. The main idea to get the most out of loop unrolling in RC4 is to completely bypass the generation of S1, and move directly from S0 to S2, as discussed right before. However, note that we require the state S1 for computing the output byte.

Table- CCCLXXXIII: Different cases for Register to Register Transfer

No.	Condition	Register-to-Register Transfers
1	i2 ≠ j1 & j2 ≠i1 & j2 ≠ j1	S0[i1] → S0[j1], S0[j1] → S0[i1], S0[i2] → S0[j2], S0[j2] → S0[i2]
2	i2 ≠ j1 & j2 ≠ i1 & j2 = j1	S0[i1] → S0[i2], S0[i2] → S0[j1] = S0[j2], S0[j1] → S0[i1]
3	i2 ≠ j1 & j2 = i1 & j2 ≠ j1	S0[i1] → S0[j1], S0[i2] → S0[i1] = S0[j2], S0[j1] → S0[i2]
4	i2 ≠j1 & j2 = i1 & j2 = j1	S0[i1] → S0[i2], S0[i2] → S0[i1] = S0[j1] = S0[j2]
5	i2 = j1 & j2 ≠ i1 & j2 ≠ j1	S0[i1] → S0[j2], S0[j2] → S0[j1] = S0[i2], S0[j1] → S0[i1]
6	i2 = j1 & j2 ≠i1 & j2 = j1	S0[i1] → S0[j1] = S0[i2] = S0[j2], S0[j1] → S0[i1]
7	i2 = j1 & j2 = i1 & j2 ≠ j1	Identity permutation, no data transfer
8	i2 = j1 & j2 = i1 & j2 = j1	Impossible, as it implies i1 = i2 = i1 +1

The Z values are read from the S-registers after the completion of the double-swap, and using the loop unrolling logic from the first one-byte-per-clock design. That is, two consecutive values of the output byte Z are read from the same state S by using some suitable combinational logic. Similarly, the increment of two consecutive i and j values are done simultaneously using the combinational logic of the original one-byte-per-clock design.

Z1 = S1[S0[j1] + S0[i1]] = Z1 = S2[S0[j1] + S0[i1]].

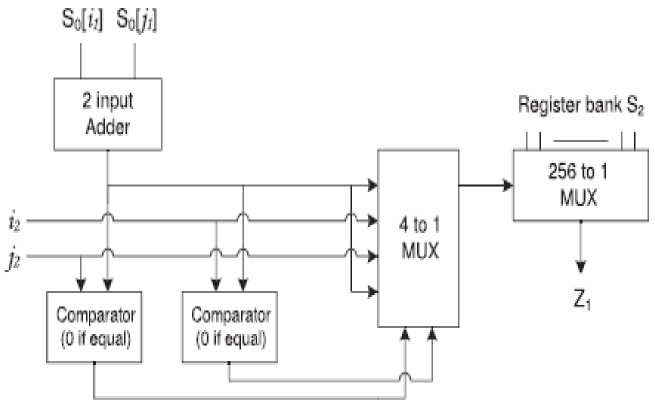

Fig-3: Circuit to compute Z1.

Computing Z1 involves adding S0[i1] and S0[j1] first, which can be done using a 2-input parallel adder. The 256 to 1 MUX, which is used to extract appropriate data from S2,will be controlled by another 4 to 1 MUX as shown in fig.3. This 4 to 1 MUX is in turn controlled by the outputs of two comparators comparing

- S0[j1] + S0[i1] and i2,
- S0[j1] + S0[i1] and j2

Computation of Z2 involves adding S1[i2] , S1[j2] The conditions can be realized using an 8 to 1 MUX unit controlled by the outputs of three comparators comparing 1) i2 and j1, 2) j2 and i1, 3) j2 and j1. We can use the same control lines as in case of the swapping operation. The circuit is as shown in fig.4.

Z2 = S2[S2[i2] + S2[j2]] = S2[S1[j2] + S1[i2]].

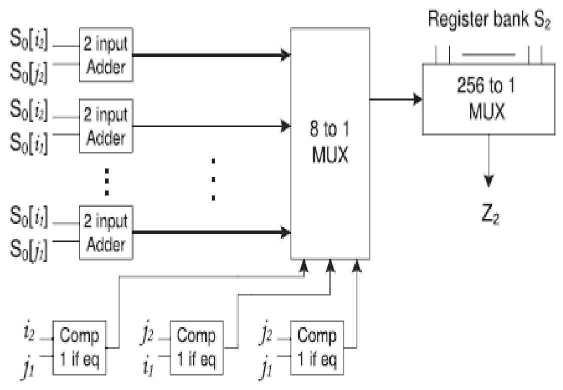

Fig-4: Circuit to compute Z2

III. PIPELINE STRUCTURE

For an intuitive pipeline architecture and timing analysis of this new design, one needs to recall the pipeline structures of the individual designs based on loop unrolling and hardware pipelining.

- Increment of indices I and j.
- Swap operation in the S-register.
- Read output byte Z from S-register.

In case of hardware pipeline, we used the idea of pipeline registers to control the read-after-write sequence during S-box swaps, and that resulted in a natural two stage pipeline model for RC4 PRGA. In case of loop unrolling, this idea of pipeline registers was not used at all, but the same throughput was obtained by merging two consecutive rounds of RC4 PRGA.

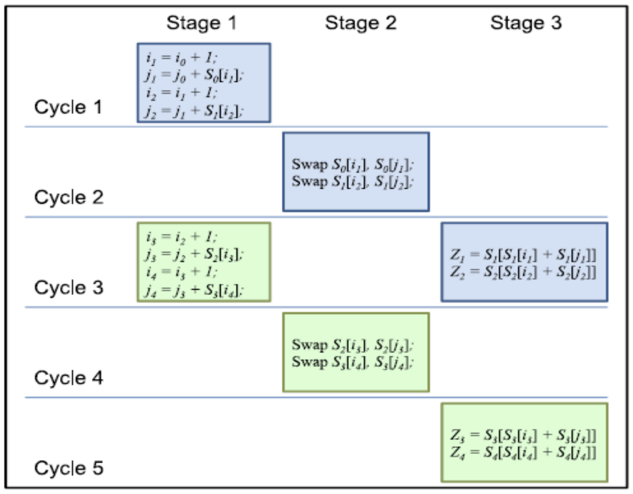
Stage 1
Stage 2
Stage 3
Cycle 1
$i_1 = i_0 + 1;$
$j_1 = j_0 + S_0[i_1];$
$i_2 = i_1 + 1;$
$j_2 = j_1 + S_1[i_2];$
Cycle 2
Swap $S_0[i_1]$, $S_0[j_1]$;
Swap $S_1[i_2]$, $S_1[j_2]$;
Cycle 3
$i_3 = i_2 + 1;$
$j_3 = j_2 + S_2[i_3];$
$i_4 = i_3 + 1;$
$j_4 = j_3 + S_3[i_4];$
$Z_1 = S_1[S_1[i_1] + S_1[j_1]]$
$Z_2 = S_2[S_2[i_2] + S_2[j_2]]$
Cycle 4
Swap $S_2[i_3]$, $S_2[j_3]$;
Swap $S_3[i_4]$, $S_3[j_4]$;
Cycle 5
$Z_3 = S_3[S_3[i_3] + S_3[j_3]]$
$Z_4 = S_4[S_4[i_4] + S_4[j_4]]$

Fig-5: Pipeline structure for 2-byte-per-clock design

In the design for 2 bytes per clock cycle throughput, we propose a fusion of the two ideas, to generate a 2-stage pipeline architecture, as shown in Fig.5. The double swap operation starts at Stage 1 in this case, and takes the help of pipeline registers to maintain the read after-write ordering during the swap operations.

The PRGA and KSA will not run in parallel, so we shared the read and write ports of S-box and K-box between PRGA and KSA. From KSA, two read accesses to K-box are required as two loops are merged per cycle. Further, four read and four write accesses to S-box are needed for the double swap operation. From PRGA, six read accesses to S-box and four write accesses to S-box are required.

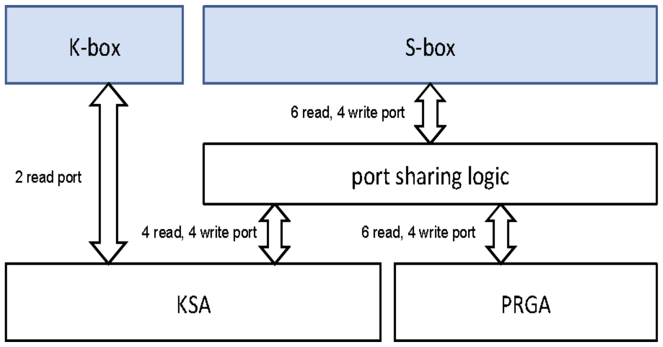

Fig-6: Port-sharing of KSA and PRGA

The two read accesses correspond to simultaneous generation of two Z values at the last stage of PRGA, while the four read and four write accesses correspond to the double swap operation. While sharing the mutually exclusive accesses, all the accesses from KSA can be merged among the PRGA accesses. Therefore, the total number of read ports to K-box is 2, the total number of read ports to S-box is 6 and the total number of write ports to S-box is 4.

The port-sharing logic reduces the multiplexer area significantly. It should be noted that the multiplexer logic to the register banks claims the major share of area. The port-sharing logic reduces a major share of this combinational area in our design[5]. In fig.6, we illustrate the circuit structure for the port-sharing logic that operates with the S-box during KSA and PRGA. Note that K-box accesses are only made by KSA, and there is no question of port sharing in that context. We illustrate the port-sharing logic, using just one read and one write port for simplicity.

IV. IMPLEMENTATION

The schematic diagrams for PRGA and KSA circuits in the proposed design are shown in Figs. 6.4 and 6.5, respectively. The PRGA circuit operates as per the 2-stage pipeline structure, where the increment of indices takes places in the first stage, and so does the double-swap operation for the S-box.

In the same stage, the addresses for the two consecutive output bytes ZnandZn+1 are calculated as the swap does not change the outcomes of the addition S [in] + S [jn] or S [in+1] + S [jn+1].In the second stage of the pipeline, the output addresses Z $_{n\_addr}$ and z $_{n+1\_addr}$ are used to read the appropriate key stream bytes from the updated S-box.

The circuit for KSA operates similarly, but has no pipeline feature as the operation happens in a single stage. Here, the increment of indices and swap are done for two consecutive rounds of KSA in a single clock cycle, thereby producing a speed of 2-rounds-per-cycle.

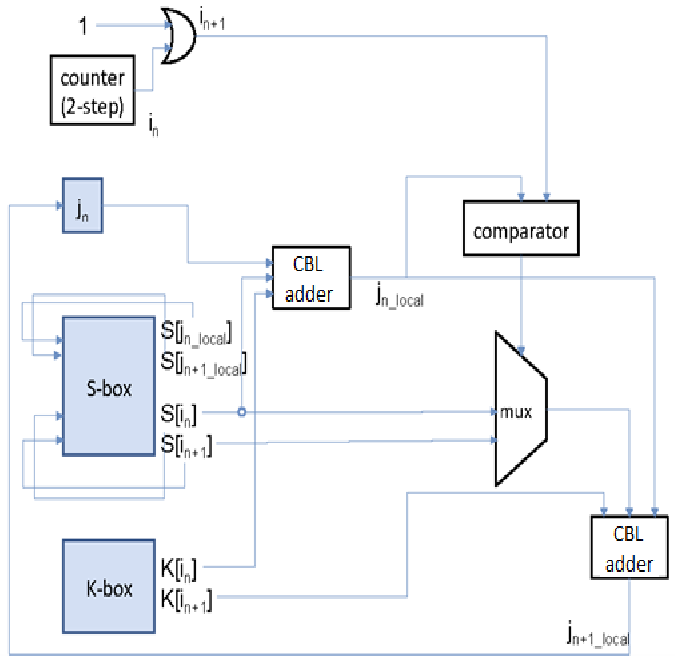

Fig-7: KSA circuit structure

Issues with KSA: Note that the general KSA routine runs for 256 iterations to produce the initial permutation of the S-box. Moreover, the steps of KSA are quite similar to the steps of PRGA, apart from the following:

- Calculation of j involves key Kalong with S and I.
- Computing Z1, Z2 is neither required nor advised.

This design obviously provides two output bytes per clock cycle, after an initial lag of one cycle, as is evident from Fig.6.3. Thus, for the generation of 2N key stream bytes in RC4 PRGA, the circuit has to operate for just N+1 clock cycles, thereby producing an asymptotic throughput of 2-bytes-per-clock. In KSA, we simply omit Stage 2 of the pipeline structure, and obtain a speed of two KSA rounds per clock cycle. Thus, KSA is completed within 128 cycles in this design.

Based on this schematic diagram for the circuits, and the port-sharing logic described earlier, we now attempt the hardware implementation of our new design.

The proposed structure for RC4 stream cipher is implemented using synthesizable verilg description .The S-register box and K-register box are implemented as array of master-slave flip-flops . Carry Select Adder (CSLA) is one of the fastest adders used in many data-processing processors to perform fast arithmetic functions.

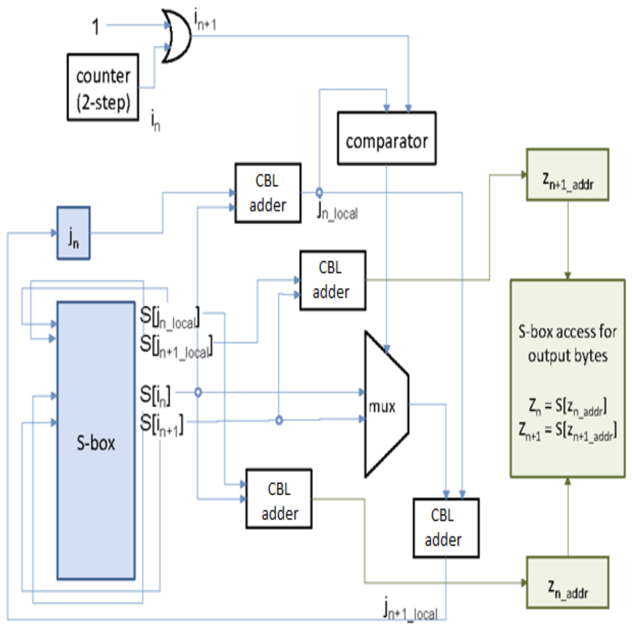

Fig- 8: PRGA circuit structure

From the structure of the CSLA, it is clear that there is scope for reducing the area and power consumption in the CSLA. This work uses a simple and efficient gate-level modification to significantly reduce the area and power of the CSLA. Based on this modification, the proposed design has reduced area and power. This work evaluates the performance of the proposed designs in terms of delay, area, power, and their products by hand with logical effort and through custom design and layout.

In modified architecture, an area-efficient carry select adder by sharing the common Boolean logic term to remove the duplicated adder cells in the conventional carry select adder is shown in this way, it saves many transistor counts and achieves a low power. Through analyzing the truth table of a single bit full adder, to find out the output of summation signal as carry-in signal is logic ‘0’ is the inverse signal of itself as carry-in signal is logic ‘1’.

By sharing the common Boolean logic term in summation generation, a proposed carry select adder design is illustrated in the figure. To share the common Boolean logic term it needs to implement one OR gate with INVERTER gate

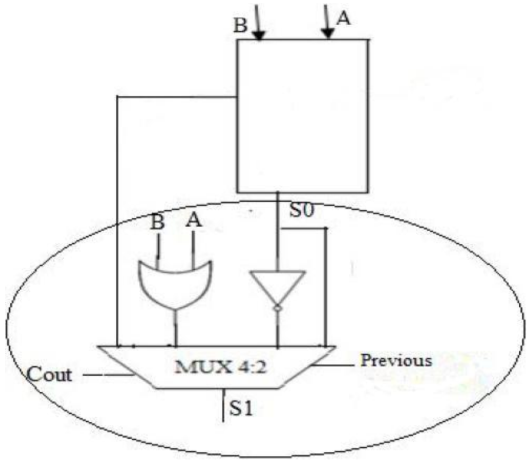

Fig-9: Single bit full adder with CB

to generate the carry signal and summation signal pair. Once the carry-in signal is ready, then select the correct carry-out output according to the logic state of carry-in signal. In this way, the summation and carry circuits can be kept parallel. The results analysis shows the improvement in the design.

V. COMPARISON

We have already seen the main RC4 designs requires 257 cycles to complete KSA and generates 2-bytes-percycle in PRGA, with an initial lag of two cycles. This design produces N key stream bytes in approximately 257 +(N/2 + 2) = N/2 + 259 clock cycles. For large N, this increases and also the hardware proposed. In the earlier designs that provides a key stream throughput of 1 byte-per cycle ,the design produces N bytes in approx. N + 259 cycles[1].

A precise comparison of area requirements with could not be made due to mismatch in implementation platforms, and does not specify any area figures at all. we synthesized the RAM using 65 nm technology. Power and area reduction has become a popular design goal for advanced design application, whether mobile or not. Reducing power consumption in chips enables better, cheaper products to be designed and power-related chip failures to be minimized. As a result, how to minimize power consumption has become an important design goal that every chip designer must take care.

VI. CONCLUSION

A completely new design of RC4 hardware is designed using an area-efficient carry select adder by sharing the common Boolean logic term, it saves many transistor counts and achieves a low power. This is the fastest known RC4 hardware architecture till date .It uses efficient hardware pipeline and loop unrolling simultaneously.

This model provides a throughput of 2 bytes per cycle in RC4 PRGA, without losing the clock performance. This is the major contribution in the RC4 design. RC4 is one of the widely used stream ciphers which is used in network protocols such as SSL, TLS, WEP and WPA. The cipher also finds applications in Microsoft Windows, Lotus Notes, Oracle Secure SQL etc. In spite of several cryptanalysis attempts on RC4, the cipher stands secure if used properly. There are obvious scopes for further improvement in throughput, using more loop unrolling or better pipelining, and these or not optimum in terms of area and speed. In the future, the further look into the hardware architecture for optimizing the area is needed and enhance the encryption process..

REFERENCES

[1]. S. Sen Gupta, K. Sinha, S. Maitra, and B.P. Sinha, “One Byte per Clock: A Novel RC4 Hardware,” Proc.INDOCRYPT’10,vol.6498,pp. 347-363, 2010.

[2]. I. Mantin,(2005) “Predicting and Distinguishing Attacks on RC4 Key stream Generator,” Proc. 24th Ann. Int’l Conf. Theory and Applications of Cryptographic Techniques (EUROCRYPT ’05), vol. 3494, pp. 491-506.

[3]. J. Golic, (1997) “Linear Statistical Weakness of Alleged RC4 Key stream Generator,” Proc. Advances in Cryptology EUROCRYPT, vol. 1233,pp. 226-238.

[4]. J.-D. Lee and C.-P. Fan, “Efficient Low-Latency RC4 Architecture Designs for IEEE 802.11i WEP” Proc. Int’l Symp. Intelligent Signal Processing and Comm. Systems (ISPACS ’07), pp. 56-59.

[5]. M.D. Galanis, P. Kitsos, G. Kostopoulos, N. Sklavos, and C.E. Goutis, (2005), “Comparison of the Hardware Implementation of Stream Ciphers,” Int’l Arab J. Information Technology, vol. 2, no. 4, pp. 267- 274.

[6]. A New Robust Enrichment Symmetric Stream Cipher Approach for Confidentiality Based on RC4 Stream Cipher Algorithm, Jagdeep Singh, Kundan Munjal.

[7]. P. Kitsos, G. Kostopoulos, N. Sklavos, and O. Koufopavlou,“Hardware Implementation of the RC4 Stream Cipher,” Proc.IEEE 46th Midwest Symp. Circuits and Systems , 2003.