Novel Hardware Implementation of Modified RC4 Stream Cipher for Wireless Network Security

N. B. Hulle
Research Scholar
G. H. Raisoni College of Engineering, Nagpur

R. D. Kharadkar, PhD.
Principal
G. H. Raisoni Institute of Engineering and Technology, Pune

A. Y. Deshmukh, PhD.
HOD Electronics
G. H. Raisoni College of Engineering, Nagpur

ABSTRACT
This paper presents novel hardware implementation of modified RC4 stream cipher for wireless network security. The Modified RC4 algorithm proposes two changes in conventional RC4 stream cipher algorithm, one in Key Scheduling Algorithm (KSA) stage and other in Pseudorandom Generation Algorithm (PRGA) stage. This modification in KSA removed many weakness and produces random states, so that it will hard to identify any non-uniformity in the permutation after KSA. The other modification in PRGA has destroyed relation between internal states of S-boxes, which is base of many attacks on RC4. The proposed architecture uses variable key length from 1 byte to (256+256+8) 520 bytes but previous designs support maximum key length of 256 bytes. This architecture achieved throughput of 63.449 Mbps at a clock frequency of 190.349 MHz independent of key length. The system was implemented in hardware by using VHDL language and Xilinx FPGA device.

General Terms
Stream ciphers for wireless network security.

Keywords
KSA, PRGA, RC4, security, stream cipher, hardware.

1. INTRODUCTION
Wireless Local Area Network (WLAN) technology shows bright future for different wireless communications industries. The IEEE 802.11 is one of the most widely used wireless standards. IEEE 802.11 standard does not provide implementation details, but provides specifications for physical and Media Access Control (MAC) layers. This standard is widely used in ad-hoc and client/server networks but special attention should be given to the security of data in transmission channel. The security of this standard is based on Wired Equivalent Privacy (WEP) scheme. Currently, all the IEEE 802.11 products support WEP, but IEEE 802.11i working group introduced the Advanced Encryption Standard (AES), as the security scheme for the future IEEE 802.11 applications.

The WEP uses two basic components, Pseudorandom Number Generation (PRNG) and the integrity algorithm. The PRNG is the most important component because it is the actually original encryption core. WEP adopts RC4 stream cipher as PRNG unit and Cyclic Redundancy Check (CRC-32) as the integrity algorithm. WEP has several limitations and encryption procedure has no provision for key rotation, users have to transmit the data using same single key; which made cracking of WEP even easier. In response to growing wireless security importance in corporate field, the Wireless Fidelity (Wi-Fi) alliance proposed a new security protocol Wi-Fi Protected Access (WPA). WPA uses RC4 stream cipher as a security algorithm with new dynamic key management method, known as Temporal Key Integrity Protocol (TKIP). TKIP uses Message Integrity Code (MIC) instead of WEP's CRC-32 for ensuring data integrity. WPA is a secure solution for upgradable equipment of WEP but not supporting to WPA2. As far as transmission speed is concerned WPA has edge over WPA2 [1, 2, 3, 4, 5, 6].

RC4 stream cipher was designed by Ron Rivest of RSA Security in 1987 and officially termed "Rivest Cipher 4". The RC4 was initially a trade secret, but in September 1994 the details of RC4 was secretly posted to the cipher punk's mailing list so it leaked in 1994. It was soon posted on the sci.crypt newsgroup and many sites on the internet, the algorithm is known to most of the cryptanalysis, it was no longer a trade secret.

The proposed hardware supports variable key lengths from 1 byte to 520 bytes and it uses two 256 bytes S-arrays and two 256 bytes K-arrays for implementation, but previous design [7] supports key length from 1 byte to 256 bytes and uses three 256 bytes S-arrays and one 256 bytes K-array with conventional RC4 algorithm, the same can be implemented with only one S-array and one K-array [8]. The proposed implementation needs three clock cycles per byte generation in each stage of KSA and three clock cycles per byte generation in the PRGA.

This paper is arranged in following sections. Section 2 describes the conventional RC4 stream cipher algorithm and modified RC4 stream cipher algorithm. In section 3 the proposed novel hardware implementation of modified RC4 algorithm is presented and analyzed in all respects. The VLSI implementation, its results are discussed in section 4 and section 5 concludes the proposed novel architecture.

2. RC4 AND MODIFIED RC4 STREAM CIPHER
Conventional RC4 uses a variable length key from 1 byte to 256 bytes to initialize a 256-byte array. There are two 256-bytes arrays, S-Box and K-Box. The S-array is filled linearly such as S0=0, S1=1, S2=2……..S255=255. The K-array consists of the key, repeating as necessary times, in order to fill the array. It has the capability of using keys between 1 byte and 256 bytes. The RC4 algorithm works in two phases, KSA and PRGA. During KSA encryption key is used to

generate an encrypting variable using two arrays, S-array, K-array and N-number of mixing operations. These mixing operations consist of swapping bytes according to RC4 algorithm with the help of i and j pointers [7].

Fig. 1 and 2 shows the conventional RC4 algorithm, which uses two pointers i and j, which are initialized to zero value. In the key setup phase the S-box is being modified according to pseudo-code:

KSA: For i= 0 to 255, j = (j + Si + Ki) mod 256.

PRGA: For i= 0 to 255, j = (j + Si) mod 256.

From the previous research work on RC4, we know that security of RC4 depends on key used and internal states of S-box. In 2001 Fluhrer, Mantin and Shamir [9] proved that particular pattern of small number of key bits are sufficient to determine the large no of state bits (Invariance weakness) in conventional KSA, but same weakness can be removed by using three layered KSA presented by Subhamoy Maitra and Goutam Paul. [10].

```
KSA:

for i = 0 to N - l
S[i] = i;
j = 0;
for i = 0 to N-1
{
j = (j + S[i] + k[i]) mod N;
swap(S[i],S[j]);
i = i + 1
}
```

Figure 1: KSA of conventional RC4

```
PRGA:

i = j = 0;
Generation loop
{
i = (i + 1) mod N;
j = (j + S[i]) mod N;
swap(S[i],S[j]);
Output = S[(S[i] + S[j]) mod N];
}
```

Figure 2: PRGA of conventional RC4

Jian Xie and Xiaozhong Pan proposed improved RC4 algorithm [11] to destroy the relation between internal states of S-box in RC4 algorithm which is base of many attacks in PRGA. We are using three layered KSA proposed by Subhamoy Maitra and Goutam Paul [10] as shown in fig. 3 and PRGA of improved RC4 [11] as shown in fig. 4 in our design to provide more security with RC4 algorithm for data transmission.

```
KSA:
Initialization
For i = 0, . . . , N - 1
{
S1[i] = i;
S2[i] = i;
}
j1 = j2 = 0;

Layer 1: Basic Scrambling
For i = 0, . . . , N - 1
{
j1 = (j1 + S1[i] + K1[i]) mod N;
Swap (S1[i], S1[j]);

j2 = (j2 + S2[i] + K2[i]) mod N;
Swap (S2[i], S2[j]);
}

Layer 2: Scrambling with IV
For i =N/2 -1 Down to 0
{
j1 = (j1 + S1[i]) ⊕ (K1[i] + IV1[i]);
Swap(S1[i], S1[j]);

j2 = (j2 + S2[i]) ⊕ (K2[i] + IV2[i]);
Swap(S2[i], S2[j]);
}

For i =N/2 to N-1
{
 j1 = (j1 + S1[i]) ⊕ (K1[i] + IV1[i]);
Swap (S1[i], S1[j]);

j2 = (j2 + S2[i]) ⊕ (K2[i] + IV2[i]);
Swap (S2[i], S2[j]);
}

Layer 3: Zigzag Scrambling
For y = 0, . . . , N – 1
{
If y ≡ 0 mod 2 then
        i =y/2;
    else
        i = N – (y+1)/2;
j1 = (j1 + S1[i] + K1[i]);
Swap (S1[i], S1[j]);

j2 = (j2 + S2[i] + K2[i]);
Swap (S2[i], S2[j]);
}
```

Figure 3: KSA of modified RC4 Algorithm

The improved RC4 [11] chooses keys k1 and k2 as a secret key and two S-boxes S1 and S2. In improved RC4 only one layer basic scrambling is used in KSA but proposed design uses three layered KSA [10]. The first layer reduces key correlation with permutation bytes and other biases to some extent at the start of KSA. In second layer index i is incremented from middle to bottom and middle to top respectively, so that the values in the bottom quarter and top quarter permutation, which were biased to linear combination

of secret key bytes were swapped. This helps in removing the biases in the range of 0 to N-1 of S-box. In the third layer, scrambling is performed in zig-zag fashion. Index i takes the values in the form of 0, 255, 1, 254, …, 126, 129, 127,128[10].

Modified RC4 algorithm uses variable length key from 1 byte to 520 bytes to initialize two 256-byte k-arrays and eight bit as LFSR initialization vector. It consists of four 256-bytes arrays, two S-Boxes and two K-Boxes. Each S-array is filled linearly such as S0=0, S1=1, S2=2……...S255=255. The K-array consists of the key, repeating as necessary times, in order to fill the both arrays.

```
PRGA:
i = j1 = j2 = 0;
Generation loop
{
i = i + 1;
j1 = j1 + S1[i];
swap(S1 [i], S1[j1]);

j2 = j2 + S2[i];
swap(S2 [i], S2 [j2]);

Output = S1 [(S1 [i] + S1[j1]) mod N];
Output = S2 [(S2 [i] + S2 [j2]) mod N];

swap(S1[S2[j1]], S1[S2[j2]]);
swap(S2[S1 [j1]], S2[S1[j2]]);
}
```

Figure 4: PRGA of modified RC4

During KSA, encryption key is used to generate an encrypting variable using two arrays, S-array, K-array and N-number of mixing operations. These mixing operations consist of swapping bytes according to keys k1 and k2 as a secret key for two S-boxes S1 and S2 respectively having 256 elements from 0 to 255. The Proposed algorithm uses three pointers i, j1 and j2, which are initialized to zero value and updated as per following equations.

KSA: layer 1: For i= 0 to 255, j1 = (j1 + S1[i] + K1[i]) mod 256 and j2 = (j2 + S2[i] + K2[i]) mod 256.

KSA: Layer2: For i =256/2 -1 Down to 0, j1 =(j1 + S1[i]) ⊕ (K1[i] + IV1[i]) mod 256 , j2 =(j2 + S2[i]) ⊕ (K2[i] + IV2[i]) mod 256, For i =256/2 to 256-1, j1 = (j1 + S1[i]) ⊕ (K1[i] + IV1[i]) mod 256, j2 = (j2 + S2[i]) ⊕ (K2[i] + IV2[i]) mod 256,

KSA: Layer 3: i= 0, 255, 1, 254, …, 126, 129, 127,128. j1 = (j1 + S1[i] + K1[i]), j2 = (j2 + S2[i] + K2[i]).

During PRGA, S1 and S2 are used to generate two pointers j1 and j2 as shown, PRGA: For i= 0 to 255, j1 = (j1 + S1[i]) mod 256 and j2 = (j2 + S2[i]) mod 256 [10].

3. PROPOSED ARCHITECTURE

Fig. 5 shows the block diagram of proposed architecture consisting of four 256 bytes RAM and used as S-RAM1 (S1), SRAM2 (S2), K-RAM1 (K1), KRAM2 (K2) with necessary control unit and address generator logic. Due to parallel structure of this novel architecture two parallel streams are generated from S-RAM1 and S-RAM2. More confusion at the output side can be added by selecting one of the streams randomly from MUX out by using 8 bit LFSR as select lines, whose feedback function is represented by primitive polynomial x8 + x4 + x3 + x2 + 1. The MUX output is pseudorandom byte converted into serial form using key searilizer. The eight byte FIFO is used to store eight bytes of data and converted into serial form by data searilizer. Finally two streams are Ex-ORed to obtain encrypted serial data out [8, 12, 13].

Fig. 6 shows the state diagram of proposed architecture. It goes through following twelve states of architecture, such as Idle, Initial, Addr2Cal, Addr2Ld, SJ, SwapSI, SwapSJ, SwapS1S2, Addr2Gen, TCal, KByte and Encryption [12,13].

The block diagram showing complete architecture with detailed connections of K-RAM1, S-RAM1, K-RAM2, S-RAM2 and necessary swap logic is shown in figure 7. The KSA of RC4 stream cipher is divided into three layers and four steps.

In the first step two S-Boxes are linearly filled from 0 to 255 using two eight bit counters Addr11 and Addr12. When signals RST = 0, layer1Cntl=0, layer2Cntl=1, down=1 and layer3Cntl=1 then initial values of counter Addr11 and Addr12 are set to 0000 0000. The count value from Addr11 is used as address for S-RAM1, K-RAM1 and same is used as data for S-RAM1, similarly the count value from Addr12 is used as address for S-RAM2, K-RAM2 and data for S-RAM2. This step needs 256 clock cycles to fill 256 memory locations of S-RAM1, K-RAM1, S-RAM2 and K-RAM2. When Initialover=1 then system enters into step 2 [10, 12, 13].

In the second step both S-Boxes are randomized by using three pointers i, j1, j2 and four memory boxes S1, S2, K1, K2. During first clock cycle when RST = 0, layer1Cntl=0, layer2Cntl=1, down=1 and layer3Cntl=1 then Addr11 and Addr12 are set to 0000 0000. Addr11 is used as pointer i and gives address for K-RAM1, SRAM1, and Addr12 gives address for K-RAM2 and SRAM2. The contents from S-RAM1 and S-RAM2 are loaded into S_Reg1 and S_Reg3 by selecting proper value from each DataDMux. This loaded value acts as S1[i] and S2[i]. The Addr21 and Addr22 gives the current value of pointer j and Adder11 and Adder12 are used to calculate new values of the pointer j1= j1 + S1[i] + K1[i] mod 256 and j2 = j2 + S2[i] + K2[i] mod 256 respectively. These calculated values of pointer j1 and j2 are loaded into Addr21 and Addr22 in the second clock cycle, which are used as address for S-RAM, the contents of these locations are loaded into S_Reg2 and S_Reg4 by selecting

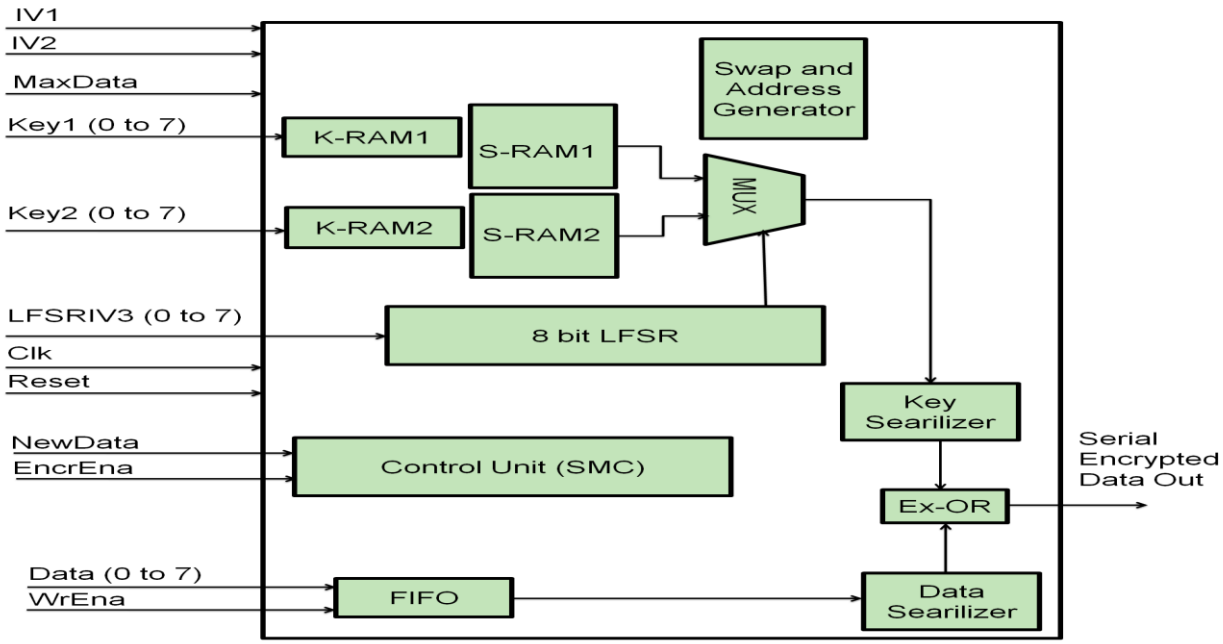

Figure 5: Block diagram of proposed architecture

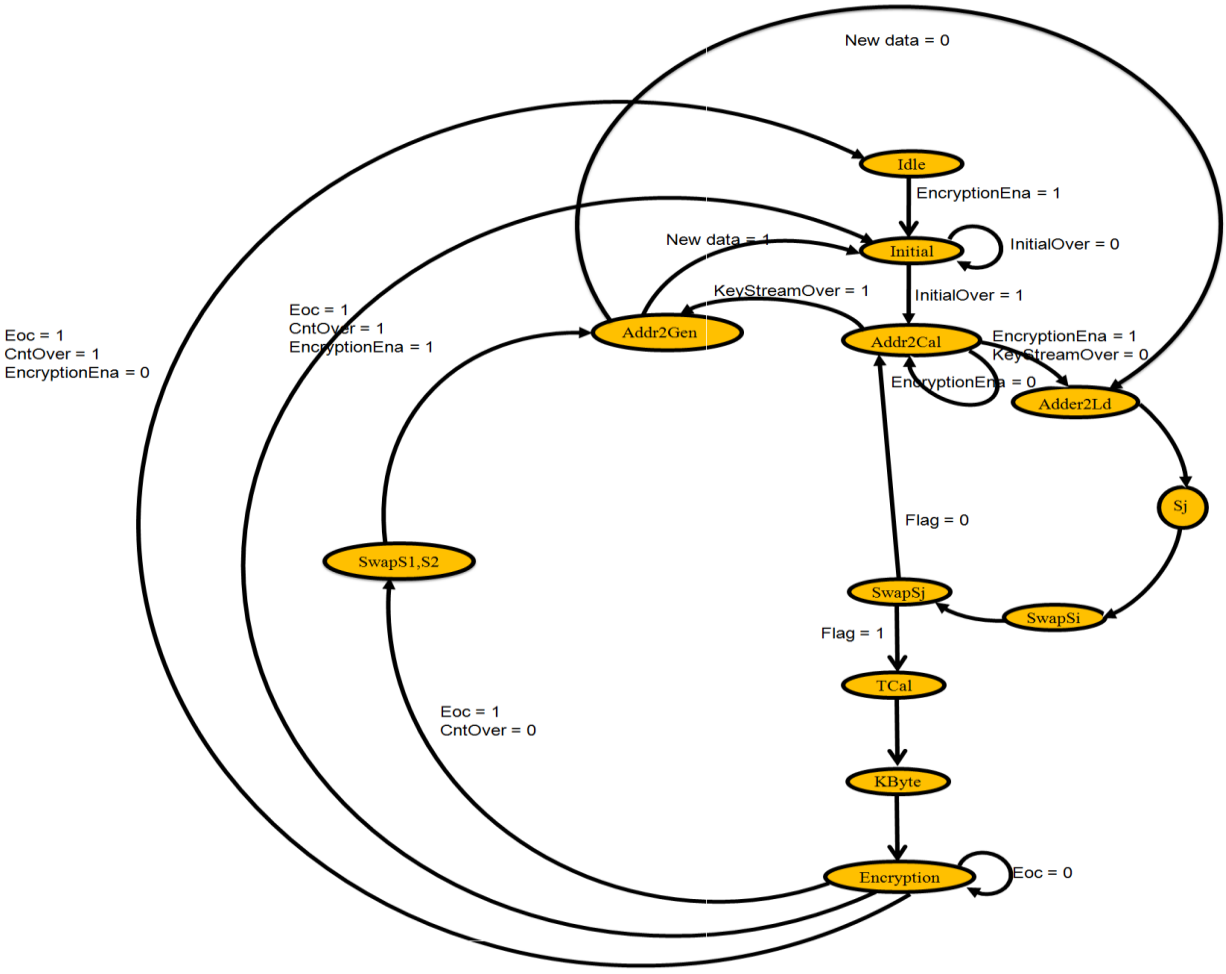

Figure 6: State diagram of proposed architecture

proper value from each DataDMux. In the third clock cycle the values of four shift registers are used for swapping the contents of S-RAM1 and S-RAM2 by selecting proper input from each AddrMux as an address. The shift registers S_Reg1 and S_Reg2 are used for swapping the contents of S-RAM1 and S_Reg3 and S_Reg4 are used to swap the contents of S-RAM2. These contents are swapped by selecting proper value from DataMux as a data and loaded into S-RAM1 and S-RAM2. This step needs three clock cycles per iteration. So, the total numbers of clock cycles needed for second step of KSA are 256*3 = 768 [10, 12, 13].

Third step works in two phases. In the first phase memory locations from 127 to 0 of both S-boxes are again randomized by using pointers i, j1, j2, four memory boxes S1, S2, K1, K2 and two initialization vectors IV1, IV2. During the first clock cycle of this step when RST = 0, layer1Cntl=1, layer2Cntl=0, down=0 and layer3Cntl=1 then initial values of counter Addr11 and Addr12 becomes 127 and these two counters works as down counter. Addr11 is used as pointer i and gives address for K-RAM1, SRAM1, and Addr12 gives address for K-RAM2 and SRAM2. The contents from S-RAM1 and S-RAM2 are loaded into S_Reg1 and S_Reg3 by selecting proper value from each DataDMux. This loaded value acts as S1[i] and S2[i]. The similar procedure is used as used in step two to calculate (j1 + S1[i]) mod 256 and (j2 + S2[i]) mod 256 only difference is that K1i and K2i are not selected for addition in Adder11 and Adder12 respectively. Similarly Adder31 and Adder32 calculates (K1i + IV1i) and (K2i + IV2i) respectively. These calculated values are Ex-ORed and used as new values of j1 and j2 and loaded into Addr21 and Addr22 in the second clock cycle, which are used as address for S-RAM, the contents of these locations are loaded into S_Reg2 and S_Reg4 by selecting proper value from each DataDMux. In the third clock cycle the swapping is achieved. This step needs three clock cycles per iteration. So, the total numbers of clock cycles needed for this phase are 128*3 = 384 [10, 12, 13].

In the second phase memory locations from 128 to 255 of both S-boxes are randomized by using the similar procedure as in first phase. During this step when RST = 0, layer1Cntl=1, layer2Cntl=0, down=1 and layer3Cntl=1 then initial values of counter Addr11 and Addr12 becomes 128 and two these two counters works as up counter. During first clock cycle Addr11 and Add12 provides value of pointer i and gives address for K-RAM, SRAM. The contents from S-RAM1 and S-RAM2 are loaded into S_Reg1 and S_Reg3 by selecting proper value from each DataDMux. This loaded value acts as S1[i] and S2[i]. The new value of j is calculated and loaded in Addr21 and Addr22 in the second clock cycle, which are used as address for S-RAM, the contents of these locations are loaded into S_Reg2 and S_Reg4 by selecting proper value from each DataDMux. In the third clock cycle swapping is carried out. This step also needs three clock cycles per iteration. So, the total numbers of clock cycles needed in this phase also 128*3 = 384 [10, 12, 13].

In the fourth step both S-Boxes are randomized again by using zig-zag pointer i. The other pointers for this step are j1, j2 and four memory boxes S1, S2, K1, K2. During first clock cycle when RST = 0, layer1Cntl=1, layer2Cntl=1, down=1 and layer3Cntl=0 then Addr11 and Addr12 are providing values in zig-zag manner (0, 255, 1, 254, …, 126, 129, 127,128). Addr11 and Addr12 are used as pointer i and gives address for K-RAM, SRAM in both stages respectively. The contents from S-RAM1 and S-RAM2 are loaded into S_Reg1 and S_Reg3. This loaded value acts as S1[i] and S2[i]. The Addr21 and Addr22 gives the current value of pointer j and Adder11 and Adder12 are used to calculate new values of the pointer j1= j1 + S1[i] + K1[i] mod 256 and j2 = j2 + S2[i] + K2[i] mod 256 respectively. These calculated values of pointer j1 and j2 are loaded into Addr21 and Addr22 in the second clock cycle, which are used as address for S-RAM, the contents of these locations are loaded into S_Reg2 and S_Reg4. In the third clock cycle the values of four shift registers are used for swapping the contents of S-RAM1 and S-RAM2 by selecting proper input from each AddrMux as a address. The shift registers S_Reg1 and S_Reg2 are used for swapping the contents of S-RAM1 and S_Reg3 and S_Reg4 are used to swap the contents of S-RAM2. This step needs three clock cycles per iteration. So, the total numbers of clock cycles needed for second step of KSA are 256*3 = 768 [10, 12, 13].

During PRGA same hardware is used but new value of j is calculated by using j= j + Si, for this Mux1 and Mux2 output is selected by each Sel, which is 0000 0000, added by same adder11 and adder12. Finally Adder21 and Adder22 are used to generate Si + Sj and given as address to S-RAM1 and S-RAM2. Two key streams are available, one at output of S-RAM1 and other at output of S-RAM2. One of the byte from S-RAM1 or S-RAM2 is selected by MuxOut by using eight bit LFSR as a select line of MUX and given to KeyStreamSerializer for converting it into serial form [8, 9, 11].

The new idea used in the proposed architecture to break relations between states of S-Boxes [11], for that after every 256 byte generation of key, the contents of S-RAM1 and S-RAM2 are swapped by using secrete pointer S1[S2[j1]], S1[S2[j2]] for S-RAM1 and S2[S1[j1]], S2[S1[j2]] for S-RAM2, this will destroys base of many attacks on RC4 stream cipher. Besides, this design uses two S-boxes, it generates two output byte in one loop which double that of conventional RC4 [7] and internal states of S-boxes becomes N!XN!, hence output stream generated by new RC4 algorithm is more secure than conventional RC4. Another important aspect of this new RC4 algorithm is the parallel structure. Two S-RAM blocks generates the pseudorandom key in parallel form, so new RC4 implementation takes the same time as that of previous structure with more security in PRGA.

The data is taken from data base and eight bytes of data are fetched in advance and stored in FIFO. One byte at a time is given to DataSerializer, to convert in data stream. Finally, key stream is taken from key stream generation module and data stream is taken from data stream generation module, which is plain text. These two streams are Ex-ORed to generate encrypted data and transmitted on RF interface.

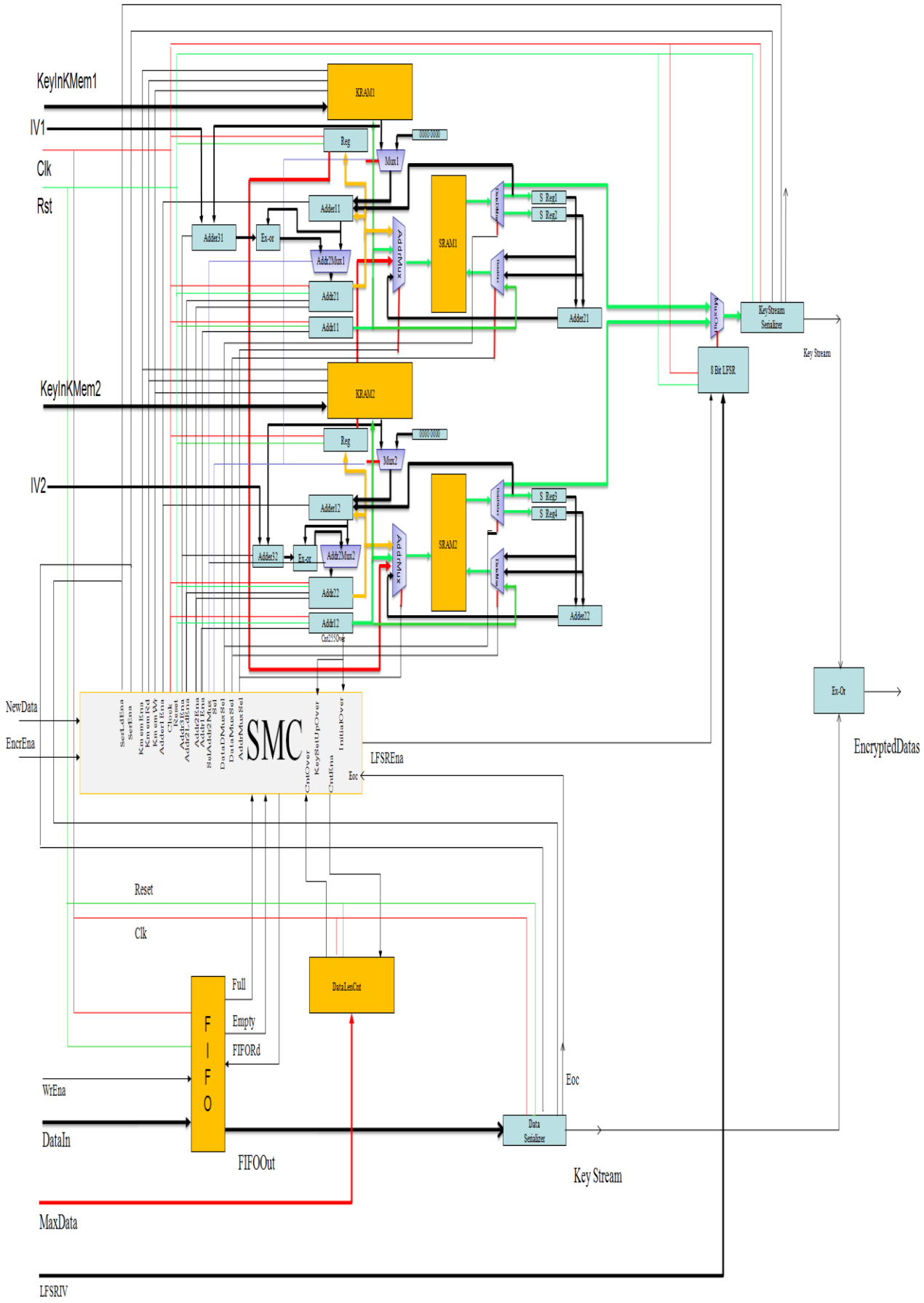

Figure 7: Architecture of proposed system

4. VLSI IMPLEMENTATION RESULTS OF MODIFIED RC4 STREAM CIPHER

The proposed architecture was made in VHDL [13,14] language on Xilinx Virtex4 series chips[15]. The software used was PC version Xilinx ISE 8.1i and used device was XC4VSX25, with package of FF668-12. All the system components were designed by behavioral and data flow modeling. These components were connected on top module by using structural modeling. The complete design was tested by test vectors [16] and synthesized, placed and routed by above device. Design information and device utilization summary for the proposed architecture is as shown below.

Design Information

Command Line: C:\Xilinx\bin\nt\map.exe-ise E:/HulleRC4/RC4STREAMCIPHRE.ise-intstyle ise -p xc4vsx25-ff668-12 -cm area -pr b -k 4 -c 100 -o TopModule_map.ncd TopModule.ngd TopModule.pcf

Target Device : xc4vsx25

Target Package : ff668

Target Speed : -12

Mapper Version : virtex4 -- $Revision: 1.34 $

Mapped Date : Sun May 08 07:44:24 2011

==========

Device utilization summary:

Selected Device : 4vsx25ff668-12

Number of Slices:	7626 out of 10240	74%
Number of Slice Flip Flops:	8627 out of 20480	42%
Number of 4 input LUTs:	13837 out of 20480	67%
Number of bonded IOBs:	42 out of 320	13%
Number of FIFO16/RAMB16s:	1 out of 128	0%
Number used as RAMB16s:	1	
Number of GCLKs:	2 out of 32	6%

==========

Fig. 8 shows RTL schematic diagram of top module and Fig. 9 shows simulated output results of top module using ISE simulator.

The proposed architecture needs 256 clock cycles in first step of KSA, (768 +768 + 768 = 2304) clock cycles are needed in second, third and fourth step of KSA and 3*n clock cycles are needed in PRGA. (n is number of bytes of plain text). The implementation [7] needs 256 + 768 + 3*n clock cycles, 256 cycles to fill S-Box linearly and 768 cycles for second step of KSA. The implementation [17] needs 1280 + 4*n clock cycles and implementation [18] needs 768 + 3*n cycles but it works with fixed key.

Comparison between the modified version of improved RC4 implementation and some of other reported implementations [7,17,18] are given in table I, but previous implementations [7,8,17,18] has two major drawbacks invariance weakness in KSA and fixed relations between states of S-Boxes in PRGA.

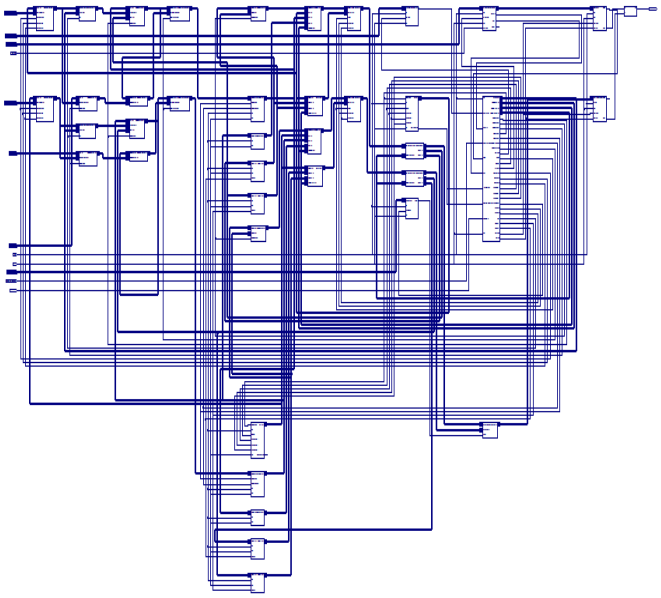

Figure 8: RTL Schematic diagram of proposed Architecture

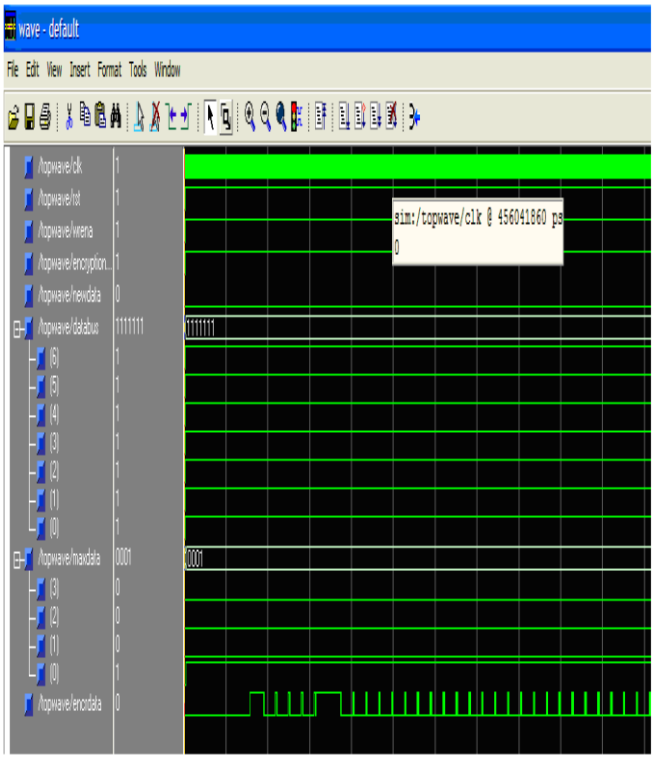

Figure 9: Output waveforms of proposed Architecture

Table 1: RC4 Implementations Comparison

RC4 Architecture	7	17 Software	18	Proposed
CLB Slices	138	-	225	7626
Frequency (MHz)	64	150	17.8	190.349
Throughput (MB/s)	22	20	2.22	63.449

5. CONCLUSION

A novel hardware implementation of the modified RC4 stream cipher algorithm for wireless network security is presented in this paper. The proposed architecture provides maximum throughput of 63.449 MB/s at a clock frequency of 190.349 MHz after initial 2304 clock cycles. The key used is variable from 1 byte to 520 bytes but previous design [7,8,18] supports only for key length of 1 byte to 256 bytes or less. The similar architecture using previous design [7] may consume total memory of 2048 bytes but proposed architecture is implemented by using 1024 bytes of RAM [8]. This new RC4 algorithm removed key correlation with permutation bytes, other biases and invariance weakness in KSA and destroyed relations between the states of S-boxes, which is the base of many attacks in PRGA stage of RC4 algorithm. The proposed design is more secure and flexible in comparison with previous designs.

6. REFERENCES

[1] K. Sali, T. Hamalainen, J. Knuutila, J. Saarinen, “Security Design for A New Wireless Local Area Network TUTWLAN”, The Ninth IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC’98), Septenber 8-11, 1998, Boston, USA, pp. 1540-1544.

[2] B. Schneier, "Applied Cryptography Protocols, Algorithms and Source Code in C", Second Edition, John Wiley and Sons, New York, 1996. pp. 171-184.

[3] A. Menezes, P. van Oorschot, and S. Vanstone, "Handbook of Applied Cryptography", CRC Press, 1996. pp. 482-504.

[4] Andrew S. Tanenbaum, “Computer Networks”, Fourth edition, Peaeson Education , 2005. pp. 292-302.

[5] Overview of IEEE 802.11b security, Intel technology Journal Q2, 2000.

[6] Jesse Walker, “Overview of IEEE 802.1lb Security”, Intel Technology Journal Q2, 2000. pp. 1062-1067.

[7] P.kitsos, G. Kostopoulos, N. Sklavos and O.Koufopavlou “Hardware implementation of the RC4 stream cipher”.VLSI design laboratory, 0-7803-8294-3/04/$20.00/2004 IEEE. pp. 1363-1366.

[8] Vandana Malode, Hulle Nagnath “Hardware Implementation of RC4 Stream Cipher for Wi-Fi Security” National Conference of Mobile and Pervasive Computing-2009, Department of Information Science and Engineering, NMAM Institute of Technology, Nitte Dist. Udupi, (Karanataka). pp. 26-30

[9] S. Fluhrer, I. Mantin, Shamir. "Weaknesses in the key scheduling algorithm of RC4 ". In Proc. 8ih Workshop on Selected Areas in Cryptography, LNCS 2259. Springer-Verlag, 2001. pp. 231-237.

[10] Subhamoy Maitra, Goutam Paul, “Analysis of RC4 and Proposal of Additional Layers for Better Security Margin” International Conference on Cryptology in India (INDOCRYPT 2008), IIT, Kharagpur.

[11] Jian Xie and Xiaozhong Pan “An Improved RC4 Stream Cipher”, Department of Electronic Technology, Xi'an, China, 978-1-4244-7237-6//10/$26.00/2010 IEEE. pp. V7 156-159.

[12] Morris Mano, “Digital Design”, Third edition, Prentice Hall of India, 2006. pp. 291-384.

[13] Stephen Brown Zvonk Vransic, “Fundamentals of Digital Logic Design with VHDL”, Second editation, Tata Mcgraw Hill, 2005. pp. 315-724.

[14] Douglas A. Pucknell, Kamran Eshraghian, “Basic VLSI design”, 3rd Edition, Prentice Hall of India, 2004. pp. 118-274.

[15] Xilinx Inc., San Jose, Calif., "Virtex, 2.5 V Field Programmable Gate Arrays," 2003, www.xilinx.com

[16] Confirmed Test Vector for RC4, http://www.qrst.de/html/dsds/rc4.htm

[17] B.Schneier, D.Whiting,"Fast Software Encryption : Designing Encryption Algorithms for Optimal Software Speed on the Intel Pentium Processor ", Fast Software Encryption workshop (FSE97), LNCS, Vol.1267, pp.242-259, Springer-Verlag, Haifa, Israel, January20-22,1997.

[18] P. Hamalainen, M. Hannikainen, T. Hamalainen and J. Snarinen, "Hardware Implementation of the Improved WEP and RC4 Encryption Algorithms for Wireless Terminals", The European Signal Processing Conference (EUSIPCO'2000), September 5-8, 2000, Tampere, Finland, pp. 2289-2292.