Overview of SNOW 1.0 [2]

We will get a brief overview of SNOW 1.0 - A New Stream Cipher which was proposed in 2000 :

Introduction –

In this paper a new word-oriented stream cipher, called SNOW, with a word size of 32 bits is proposed. The design of this stream cipher is simple and easy to implement as it consists a linear feedback shift register (LFSRs), feeding a finite state machine (FSM). The goal of this stream cipher was to produce a stream cipher significantly faster than AES with significantly lower implementation costs and security similar to that of AES. The fastest implementation of the design required only 1 clock cycle per running key bit which was very fast at that time but it had some flaws and disadvantages which will be discussed further along with design and weaknesses.

Design & Implementation –

In this stream cipher, the main part of the design was random keystream generation on which an operation usually XOR could be performed with the plaintext to produce ciphertext. For arithematics, the author assumed the "big endian" representation. The idea of the stream cipher came from a summation generator and a stream cipher E_0 in the Bluwtooth standard.
The cipher is described with two possible key sizes, 128 and 256 bits. As usual, the encryption starts with a key initialization, giving the components of the cipher their initial key values. In this description we will only concentrate on the cipher in operation. The details of the key initialization can be found in the original paper.
The generator is depicted in the schematic picture of SNOW 1.0 (Figure 1). It consists of a length 16 linear feedback shift register over $F_{2^{32}}$, feeding a finite state machine. The FSM consists of two 32 bit registers, called R1 and R2, as well as a some operations to calculate the output and the next state (the next value of R1 and R2).

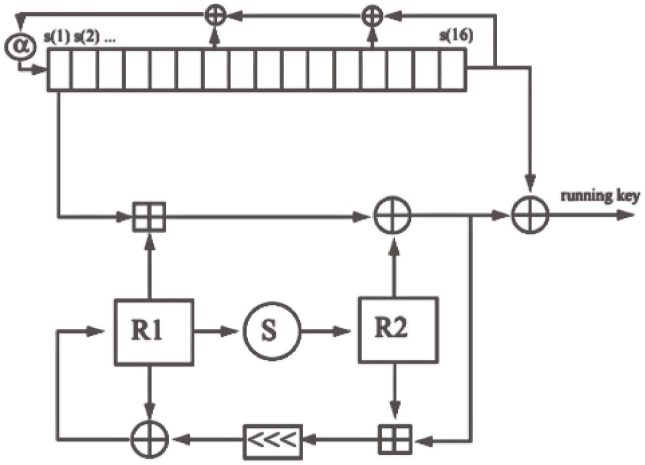


Figure 1: A schematic picture of SNOW 1.0

Operation –

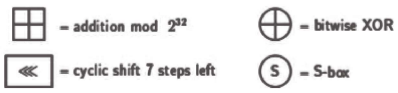


Figure 2: Meaning of Symblos Used

The operation of the cipher is as follows. First, key initialization is done. This procedure provides initial values for the LFSR as well as for the R1,R2 registers in the finite state machine. Next, the first 32 bits of the keystream is calculated by bitwise adding the output of the FSM and the last entry of the LFSR. After that the whole cipher is clocked once, and the next 32 bits of the keystream is calculated by again bitwise adding the output of the finite state machine and the last entry of the LFSR. We clock again and continue in this fashion. Returning to the previous figure, the LFSR has a primitive feedback polynomial over $F_{2^{32}}$ which is :-

$$p(x) \quad = \quad x^{16} \ + \ x^{13} \ + \ x^7 \ + \ \alpha^{-1},$$

where $F_{2^{32}}$ is generated by the irreducible polynomial

$$\pi(x) \quad = \quad x^{32} \ + \ x^{29} \ + \ x^{20} \ + \ x^{15} \ + \ x^{10} \ + \ x \ + \ 1,$$

over F_2, and $\pi(\alpha) = 0$. Furthermore let s(1), s(2),...s(16) $\in F_{2^{32}}$ be the state of the LFSR.
The output of the FSM, called FSM_{out}, is calculated as follows.

$$FSM_{out} \quad = \quad (s(1) \ \boxplus \ R1) \ \oplus \ R2.$$

The output of the FSM is XORed with s(16) to form the keystream, i.e.,

$$runningkey \quad = \quad FSM_{out} \ \oplus \ s(16).$$

The keystream is finally XORed with the plaintext, producing the ciphertext. Inside the FSM, the new values of R1 and R2 are given as follows,

$$\begin{aligned} newR1 \quad = \quad & ((FSM_{out} \ \boxplus \ R2) \quad \lll) \ \oplus \ R1, \\ R2 \quad = \quad & S(R1), \\ R1 \quad = \quad & newR1. \end{aligned}$$

By the notation $x \boxplus y$, we mean the integer addition of x and y $mod 2^{32}$. The notation $x \lll$ is a cyclic shift of x 7 steps to the left, and the addition sign in $x \oplus y$ represents bitwise addition (XOR) of the words x and y.
Finally, the S-box, denoted S(x), consists of four identical 8-to-8 bit S-boxes and a permutation of the resulting bits. The input x is split into 4 bytes, each byte enters a nonlinear mapping from 8 bits to 8 bits. After this mapping, the bits in the resulting word are permuted to form the final output of the S-box. A comprehensive description of the original design, including the key setup and modes of operation can be found in the original paper.

Weaknesses & Flaws –

We will now discuss the weaknesses found in the SNOW 1.0. Two people, P. Hawkes and F. Rose proposed a paper in 2002 describing a guess-and-determine attack on SNOW 1.0. The attack has data complexity of 2^{95} words and a process complexity of 2^{224} operations. Apart from some clever initial choices made by Hawkes and Rose, basically two properties in SNOW 1.0 are used to reduce the complexity of the attack below exhaustive key search. First, the fact that the FSM has only one input s(1). This enables an attacker to invert the operations in the FSM and derive more unknowns from only a few guesses. The second property is an unfortunate choice of feedback polynomial in SNOW 1.0.

A second weakness in the choice of the feedback polynomial emerges when considering bitwise linear approximations.

In a recent paper by D. Coppersmith, S. Halevi and C. Jutla , they found such a correlation and for the resulting distinguishing attack they need about 2^{95} words of output and the computational complexity is about 2^{100}. By computer search we have also found other smaller correlations, often involving similar bit positions. The strong correlations seem to be caused by an interaction between the permutation in the S-box and the cyclic shift by 7 in the FSM.

Even if this indeed is a security flaw unpredicted by the authors, one could argue about the relevance of such a distinguishing attack. Using e.g. AES (or any other block cipher with block length 128 bits) in counter mode, there is an almost trivial distinguishing attack after seeing about 2^{64} ciphertext blocks.

Conclusion –

The author proposed SNOW - a new stream cipher, estimated its software performance, and also gave an overview of the weaknesses and possible attacks to SNOW. The author concluded that there are still several properties yet to be considered, hardware implementations and optimization of code to be done in coming future.

Overview of SNOW 2.0 [1]

We will get a brief overview of SNOW 2.0 - A New Stream Cipher which was proposed in 2002, successor of SNOW 1.0 :

Introduction –

In this paper a new SNOW stream cipher, called SNOW 2.0, is proposed. It's implementation is bit faster and more secure than it's predecessor SNOW 1.0, which had some security flaws and weaknesses.
This new SNOW cipher is also same as previous with a word size of 32 bits The LFSR length is also unchanged i.e. 16, but the feeback polynomial has been changed to overcome the weaknesses of SNOW 1.0. The optimized C implementation of the design required 5-6 clock cycles per byte.

Design & Operation –

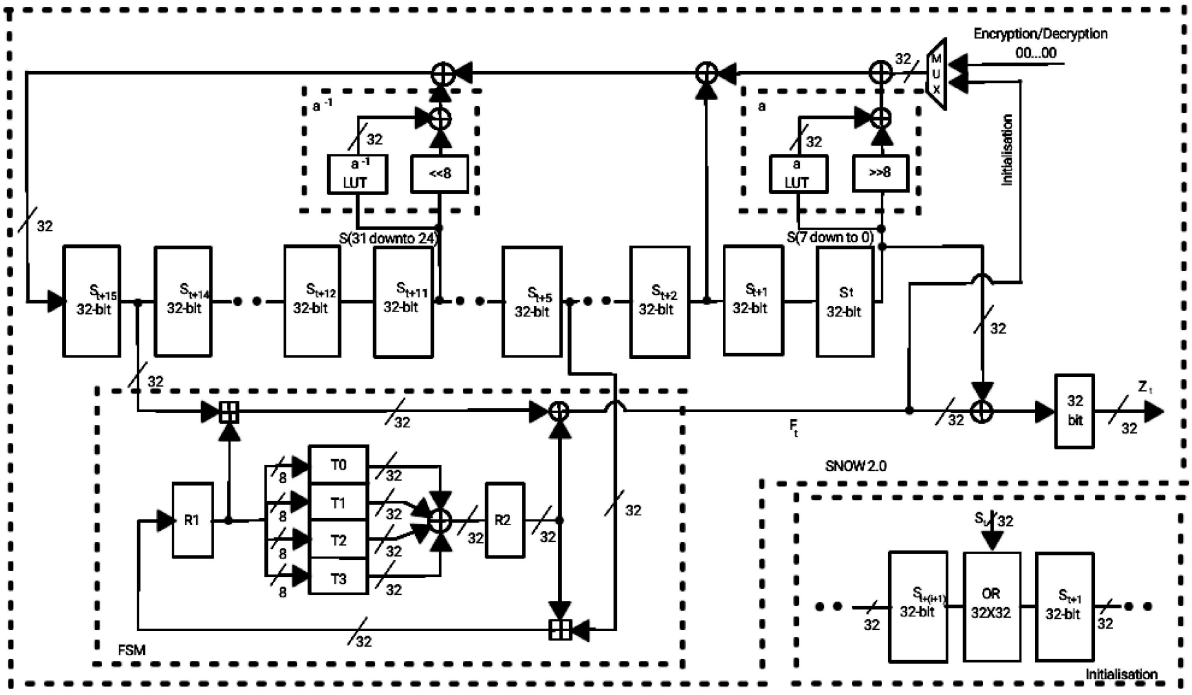


Figure 3: SNOW 2.0 Stream Cipher Architecture

Although word size and LFSR length of this cipher is same as it's predecessor the feedback polynomial has been changed. Also the Finite State Machine (FSM) now has two input words, taken from LFSR, and the running key is formed as the XOR between the FSM output and the last element of the LFSR, same as SNOW 1.0. Major implementation such as key initialization, startng states of LFSRs and initial values of R1 and R2 are also same as SNOW 1.0.
The main flaw of SNOW 1.0 was the feedback polynomial, which was chosen to give the fast realization in software but it openened the way for several attacks. In SNOW 2.0, the author proposed two different elements involved in the feedback loop, α and α^{-1}, where α now is a root of a primitive polynomial of degree 4 over F_{2^8}. TO be more precise, the feedback polynomial in SNOW 2.0 is given by :-

$$\pi(x) \quad = \quad \alpha x^{16} \ + \ x^{14} \ + \ \alpha^{-1}x^5 \ + \ 1 \quad \in \quad F_{2^{32}}[x],$$

where α is a root of $x^4 + \beta^{245}x^2 + \beta^{48}x + \beta^{239} \in F_{2^8}[x]$,
and β is a root of $x^8 + x^7 + x^5 + x^3 + 1 \in F_2[x]$. Let the state of the LFSR at time $t \geq 0$ be denoted

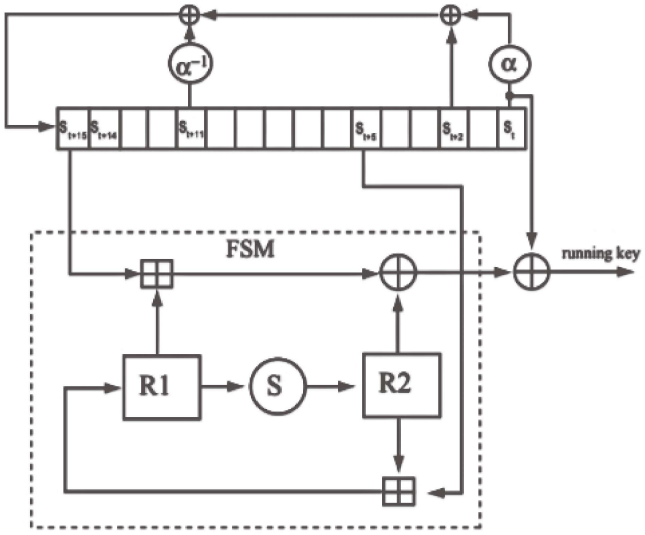


Figure 4: A schematic picture of SNOW 2.0

$(s_{t+15}, s_{t+14}, ..., s_t), s_{t+i} \in F_{2^{32}}, i \geq 0$. The element s_t is the rightmost element (or first element to exit) as indicated in Figure 2, and the sequence produced by the LFSR is (s0, s1, s2,...). By time t = 0, we mean the time instance directly after the key initialization. Then the cipher is clocked once before producing the first keystream symbol, i.e., the first keystream symbol, denoted z_1, is produced at time t = 1. The produced keystream sequence is denoted $(z_1, z_2, z_3, ...)$.

$$F_t \quad = \quad (s_{t+15} \ \boxplus \ R1_t) \ \oplus \ R2_t, t \quad \geq \quad 0$$

and the keystream is given by

$$z_t \quad = \quad F_t \ \oplus \ s_t, t \quad \geq \quad 1.$$

Here we use the notation $\boxplus$ for integer addition modulo 2_{32} and $\oplus$ for bitwise addition (XOR). The registers R1 and R2 are updated with new values according to

$$\begin{array}{rcl} R1_{t+1} & = & s_{t+5} \boxplus R2_t \text{ and} \\ R2_{t+1} & = & S(R1_t), t \geq 0. \end{array}$$

The S-Box –

The S-box, denoted by S(w), is a permutation on $Z_{2^{32}}$ based on the round function of Rijndael by J. Daemen V. Rijmen. Let $w = (w_3, w_2, w_1, w_0)$ be the input to the S-box, where $w_i, i = 0...3$ is the four bytes of w. Assume w_3 to be the most significant byte. Let

$$w = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{pmatrix}$$

be a vector representation of the input to the S-box. First we apply the Rijndael S-box, denoted S_R to each byte, giving us the vector

$$\begin{pmatrix} S_R[w_0] \\ S_R[w_1] \\ S_R[w_2] \\ S_R[w_3] \end{pmatrix}$$

In the MixColumn transformation of Rijndael's round function, each 4 byte word is considered a polynomial in y over F_{2^8} , defined by the irreducible polynomial $x_8+x_4+x_3+x+1 \in F_2[x]$. Each word can be represented by a polynomial of at most degree 3. Next we consider the vector in above eq as representing a polynomial over F_{2^8} and multiply with a fixed polynomial c(y)=(x+1)y3+y2+y+x F28 [y] modulo $y^4+1 \in F_2^8[y]$. This polynomial multiplication can (as done in Rijndael) be computed as a matrix multiplication,

$$\begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{pmatrix} = \begin{pmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{pmatrix} \begin{pmatrix} S_R[w_0] \\ S_R[w_1] \\ S_R[w_2] \\ S_R[w_3] \end{pmatrix},$$

where (r_3, r_2, r_1, r_0) are the output bytes from the S-box. These bytes are concatenated to form the word output from the S-box, r = S(w).

Key Initialization –

SNOW 2.0 takes two parameters as input values; a secret key of either 128 or 256 bits and a publicly known 128 bit initialization variable IV . The IV value is considered as a four word input

$IV = (IV_3, IV_2, IV_1, IV_0)$, where IV_0 is the least significant word. The possible range for IV is thus 0...$2^1$281. This means that for a given secret key K, SNOW 2.0 implements a pseudo-random length-increasing function from the set of IV values to the set of possible output sequences. The use of a IV value is optional and applications requiring a IV value typically reinitialize the cipher frequently with a fixed key but the IV value is changed. This could be the case if two parties agreed on a common secret key but wish to communicate multiple messages, e.g. in a frame based setting. Frequent reinitialization could also be desirable from a resynchronization perspective in e.g. a radio based environment. Full key initialization can be found in the original SNOW 2.0 paper. [1]

Key Differences Between SNOW 1.0 SNOW 2.0 –

The key difference between SNOW 1.0 SNOW 2.0 is the feedback polynomial.

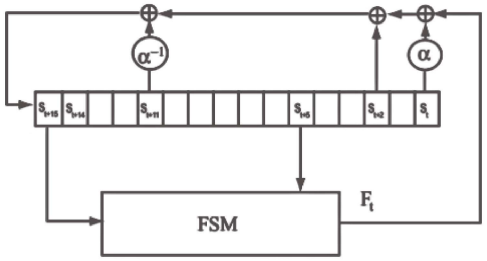


Figure 5: Cipher operation during key initialization.

In SNOW 1.0 the multiplication could be implemented by a single left shift of the word followed by a possible XOR with a known pattern of weight 6. This means that the resulting word was in many positions only a shift of the original word. In SNOW 2.0, the author defined $F_{2^{32}}$ as an extension field over F_{2^8} and each of the two multiplications can be implemented as a byte shift together with a unconditional XOR with one of 256 possible patterns. This results in a better spreading of the bits in the feedback loop, and improves the resistance against certain correlation attack. The use of two constants in the feedback loop also improves the resistance against bitwise linear approximation attacks. To the authors, there is no known method to manipulate the feedback polynomial such that the resulting linear recurrence hold for each bit position and have reasonably low weight. The unconditional XOR also seems to improve speed, by removing the possible branch prediction error in a pipelined processor.
The FSM in SNOW 2.0 takes two inputs. This makes a guess-anddetermine type of attack more difficult. Given the output of the FSM, together with R1 and R2 it is no longer possible to deduce the next FSM state directly. The update of R1 does not depend on the output of the FSM, but on a word taken from the LFSR.
The S-box in SNOW 1.0 was also byte oriented but the final bit permutation did not diffuse as much as the new design. In SNOW 1.0, each input byte to the S-box affected only 8 bits of the output word. The choice of the new S-box, based on the round function of Rijndael, provides a much stronger diffusion. Each output bit now depends on each input bit.

Conclusion –

The author proposed a new stream cipher called SNOW 2.0. The design of this cipher was based on the predecessor stream cipher - SNOW 1.0. The implementation of SNOW 2.0 is easier and encryption in faster than the SNOW 1.0. According to the paper proposed, typical encryption speed is over 3Gbits/sec on an Intel Pentium 4 running at 1.8GHz.
The complete design of SNOW 2.0 is discussed in the paper and it overcomes the weaknesses and flaws of SNOW 1.0 is also discussed.

Hardware Implementation of SNOW 2.0

SNOW 2.0 (Top Module)

```
`timescale 1ns / 1ps

module SNOW_2(
    input [127:0] s_key,
    input [127:0] IV,
    input clk,
    output [32:0] keystream
    );
    LFSR l1(.s_key(s_key), .IV(IV), .clk(clk), .keystream(keystream));
endmodule
```

Linear Feedback Shift Register (LFSR)

```
`timescale 1ns / 1ps

module LFSR(
    input [127:0] s_key,
    input [127:0] IV,
    input clk,
    output wire [31:0] keystream
    );

    reg [31:0] s0;
    reg [31:0] s1;
    reg [31:0] s2;
    reg [31:0] s3;
    reg [31:0] s4;
    reg [31:0] s5;
    reg [31:0] s6;
    reg [31:0] s7;
    reg [31:0] s8;
    reg [31:0] s9;
    reg [31:0] s10;
    reg [31:0] s11;
    reg [31:0] s12;
    reg [31:0] s13;
    reg [31:0] s14;
    reg [31:0] s15;

    wire [31:0] FSM_out;
    wire [31:0] alpha_out;
    wire [31:0] alpha_inv_out;

    always @ (posedge clk)
    begin
        s0 = (s_key[127:96] ^ IV[31:0]);
        s1 = s_key[95:64];
        s2 = s_key[63:32];
        s3 = (s_key[31:0] ^ IV[63:32]);
        s4 = (s_key[127:96] ^ 32'h11111111);
        s5 = (s_key[95:64] ^ 32'h11111111 ^ IV[95:64]);
        s6 = (s_key[63:32] ^ 32'h11111111 ^ IV[127:96]);
        s7 = (s_key[31:0] ^ 32'h11111111);
        s8 = s_key[127:96];
        s9 = s_key[95:64];
        s10 = s_key[63:32];
        s11 = s_key[31:0];
        s12 = (s_key[127:96] ^ 32'h11111111);
        s13 = (s_key[95:64] ^ 32'h11111111);
        s14 = (s_key[63:32] ^ 32'h11111111);
        s15 = (s_key[31:0] ^ 32'h11111111);
        #640;
        s0 <= (alpha_inv_out ^ (s13 ^ (alpha_out ^ FSM_out)));
```

```
        s1 <= s0;
        s2 <= s1;
        s3 <= s2;
        s4 <= s3;
        s5 <= s4;
        s6 <= s5;
        s7 <= s6;
        s8 <= s7;
        s9 <= s8;
        s10 <= s9;
        s11 <= s10;
        s12 <= s11;
        s13 <= s12;
        s14 <= s13;
        s15 <= s14;
    end

    assign keystream = (s15 ^ FSM_out);

    FSM f1(.in1(s0), .in2(s10), .clk(clk), .FSM_out(FSM_out));
    alpha_inv a1(.alpha_inv_in(s4), .alpha_inv_out(alpha_inv_out));
    alpha a2(.alpha_in(s15), .alpha_out(alpha_out));

endmodule
```

Alpha

```
`timescale 1ns / 1ps

module alpha(
    input [31:0] alpha_in,
    output [31:0] alpha_out
    );
    wire [31:0] lut_out;
    wire [31:0] shift_out;

    reg [8:0] t0=8'd0;
    reg [8:0] t1=8'd0;
    reg [8:0] t2=8'd0;
    reg [8:0] t3=8'd0;

    function [7 : 0] al(input [7 : 0] z);
    begin
      case (z)
        8'h00: al = 8'h63;
        8'h01: al = 8'h7c;
        8'h02: al = 8'h77;
        8'h03: al = 8'h7b;
        8'h04: al = 8'hf2;
        8'h05: al = 8'h6b;
        8'h06: al = 8'h6f;
        8'h07: al = 8'hc5;
        8'h08: al = 8'h30;
        8'h09: al = 8'h01;
        8'h0a: al = 8'h67;
        8'h0b: al = 8'h2b;
        8'h0c: al = 8'hfe;
        8'h0d: al = 8'hd7;
        8'h0e: al = 8'hab;
        8'h0f: al = 8'h76;
        8'h10: al = 8'hca;
        8'h11: al = 8'h82;
        8'h12: al = 8'hc9;
        8'h13: al = 8'h7d;
        8'h14: al = 8'hfa;
        8'h15: al = 8'h59;
        8'h16: al = 8'h47;
        8'h17: al = 8'hf0;
```

```
42          8'h18: al = 8'had;
43          8'h19: al = 8'hd4;
44          8'h1a: al = 8'ha2;
45          8'h1b: al = 8'haf;
46          8'h1c: al = 8'h9c;
47          8'h1d: al = 8'ha4;
48          8'h1e: al = 8'h72;
49          8'h1f: al = 8'hc0;
50          8'h20: al = 8'hb7;
51          8'h21: al = 8'hfd;
52          8'h22: al = 8'h93;
53          8'h23: al = 8'h26;
54          8'h24: al = 8'h36;
55          8'h25: al = 8'h3f;
56          8'h26: al = 8'hf7;
57          8'h27: al = 8'hcc;
58          8'h28: al = 8'h34;
59          8'h29: al = 8'ha5;
60          8'h2a: al = 8'he5;
61          8'h2b: al = 8'hf1;
62          8'h2c: al = 8'h71;
63          8'h2d: al = 8'hd8;
64          8'h2e: al = 8'h31;
65          8'h2f: al = 8'h15;
66          8'h30: al = 8'h04;
67          8'h31: al = 8'hc7;
68          8'h32: al = 8'h23;
69          8'h33: al = 8'hc3;
70          8'h34: al = 8'h18;
71          8'h35: al = 8'h96;
72          8'h36: al = 8'h05;
73          8'h37: al = 8'h9a;
74          8'h38: al = 8'h07;
75          8'h39: al = 8'h12;
76          8'h3a: al = 8'h80;
77          8'h3b: al = 8'he2;
78          8'h3c: al = 8'heb;
79          8'h3d: al = 8'h27;
80          8'h3e: al = 8'hb2;
81          8'h3f: al = 8'h75;
82          8'h40: al = 8'h09;
83          8'h41: al = 8'h83;
84          8'h42: al = 8'h2c;
85          8'h43: al = 8'h1a;
86          8'h44: al = 8'h1b;
87          8'h45: al = 8'h6e;
88          8'h46: al = 8'h5a;
89          8'h47: al = 8'ha0;
90          8'h48: al = 8'h52;
91          8'h49: al = 8'h3b;
92          8'h4a: al = 8'hd6;
93          8'h4b: al = 8'hb3;
94          8'h4c: al = 8'h29;
95          8'h4d: al = 8'he3;
96          8'h4e: al = 8'h2f;
97          8'h4f: al = 8'h84;
98          8'h50: al = 8'h53;
99          8'h51: al = 8'hd1;
100         8'h52: al = 8'h00;
101         8'h53: al = 8'hed;
102         8'h54: al = 8'h20;
103         8'h55: al = 8'hfc;
104         8'h56: al = 8'hb1;
105         8'h57: al = 8'h5b;
106         8'h58: al = 8'h6a;
107         8'h59: al = 8'hcb;
108         8'h5a: al = 8'hbe;
109         8'h5b: al = 8'h39;
110         8'h5c: al = 8'h4a;
```

```
111         8'h5d: al = 8'h4c;
112         8'h5e: al = 8'h58;
113         8'h5f: al = 8'hcf;
114         8'h60: al = 8'hd0;
115         8'h61: al = 8'hef;
116         8'h62: al = 8'haa;
117         8'h63: al = 8'hfb;
118         8'h64: al = 8'h43;
119         8'h65: al = 8'h4d;
120         8'h66: al = 8'h33;
121         8'h67: al = 8'h85;
122         8'h68: al = 8'h45;
123         8'h69: al = 8'hf9;
124         8'h6a: al = 8'h02;
125         8'h6b: al = 8'h7f;
126         8'h6c: al = 8'h50;
127         8'h6d: al = 8'h3c;
128         8'h6e: al = 8'h9f;
129         8'h6f: al = 8'ha8;
130         8'h70: al = 8'h51;
131         8'h71: al = 8'ha3;
132         8'h72: al = 8'h40;
133         8'h73: al = 8'h8f;
134         8'h74: al = 8'h92;
135         8'h75: al = 8'h9d;
136         8'h76: al = 8'h38;
137         8'h77: al = 8'hf5;
138         8'h78: al = 8'hbc;
139         8'h79: al = 8'hb6;
140         8'h7a: al = 8'hda;
141         8'h7b: al = 8'h21;
142         8'h7c: al = 8'h10;
143         8'h7d: al = 8'hff;
144         8'h7e: al = 8'hf3;
145         8'h7f: al = 8'hd2;
146         8'h80: al = 8'hcd;
147         8'h81: al = 8'h0c;
148         8'h82: al = 8'h13;
149         8'h83: al = 8'hec;
150         8'h84: al = 8'h5f;
151         8'h85: al = 8'h97;
152         8'h86: al = 8'h44;
153         8'h87: al = 8'h17;
154         8'h88: al = 8'hc4;
155         8'h89: al = 8'ha7;
156         8'h8a: al = 8'h7e;
157         8'h8b: al = 8'h3d;
158         8'h8c: al = 8'h64;
159         8'h8d: al = 8'h5d;
160         8'h8e: al = 8'h19;
161         8'h8f: al = 8'h73;
162         8'h90: al = 8'h60;
163         8'h91: al = 8'h81;
164         8'h92: al = 8'h4f;
165         8'h93: al = 8'hdc;
166         8'h94: al = 8'h22;
167         8'h95: al = 8'h2a;
168         8'h96: al = 8'h90;
169         8'h97: al = 8'h88;
170         8'h98: al = 8'h46;
171         8'h99: al = 8'hee;
172         8'h9a: al = 8'hb8;
173         8'h9b: al = 8'h14;
174         8'h9c: al = 8'hde;
175         8'h9d: al = 8'h5e;
176         8'h9e: al = 8'h0b;
177         8'h9f: al = 8'hdb;
178         8'ha0: al = 8'he0;
179         8'ha1: al = 8'h32;
```

```
180         8'ha2: al = 8'h3a;
181         8'ha3: al = 8'h0a;
182         8'ha4: al = 8'h49;
183         8'ha5: al = 8'h06;
184         8'ha6: al = 8'h24;
185         8'ha7: al = 8'h5c;
186         8'ha8: al = 8'hc2;
187         8'ha9: al = 8'hd3;
188         8'haa: al = 8'hac;
189         8'hab: al = 8'h62;
190         8'hac: al = 8'h91;
191         8'had: al = 8'h95;
192         8'hae: al = 8'he4;
193         8'haf: al = 8'h79;
194         8'hb0: al = 8'he7;
195         8'hb1: al = 8'hc8;
196         8'hb2: al = 8'h37;
197         8'hb3: al = 8'h6d;
198         8'hb4: al = 8'h8d;
199         8'hb5: al = 8'hd5;
200         8'hb6: al = 8'h4e;
201         8'hb7: al = 8'ha9;
202         8'hb8: al = 8'h6c;
203         8'hb9: al = 8'h56;
204         8'hba: al = 8'hf4;
205         8'hbb: al = 8'hea;
206         8'hbc: al = 8'h65;
207         8'hbd: al = 8'h7a;
208         8'hbe: al = 8'hae;
209         8'hbf: al = 8'h08;
210         8'hc0: al = 8'hba;
211         8'hc1: al = 8'h78;
212         8'hc2: al = 8'h25;
213         8'hc3: al = 8'h2e;
214         8'hc4: al = 8'h1c;
215         8'hc5: al = 8'ha6;
216         8'hc6: al = 8'hb4;
217         8'hc7: al = 8'hc6;
218         8'hc8: al = 8'he8;
219         8'hc9: al = 8'hdd;
220         8'hca: al = 8'h74;
221         8'hcb: al = 8'h1f;
222         8'hcc: al = 8'h4b;
223         8'hcd: al = 8'hbd;
224         8'hce: al = 8'h8b;
225         8'hcf: al = 8'h8a;
226         8'hd0: al = 8'h70;
227         8'hd1: al = 8'h3e;
228         8'hd2: al = 8'hb5;
229         8'hd3: al = 8'h66;
230         8'hd4: al = 8'h48;
231         8'hd5: al = 8'h03;
232         8'hd6: al = 8'hf6;
233         8'hd7: al = 8'h0e;
234         8'hd8: al = 8'h61;
235         8'hd9: al = 8'h35;
236         8'hda: al = 8'h57;
237         8'hdb: al = 8'hb9;
238         8'hdc: al = 8'h86;
239         8'hdd: al = 8'hc1;
240         8'hde: al = 8'h1d;
241         8'hdf: al = 8'h9e;
242         8'he0: al = 8'he1;
243         8'he1: al = 8'hf8;
244         8'he2: al = 8'h98;
245         8'he3: al = 8'h11;
246         8'he4: al = 8'h69;
247         8'he5: al = 8'hd9;
248         8'he6: al = 8'h8e;
```

```
        8'he7: al = 8'h94;
        8'he8: al = 8'h9b;
        8'he9: al = 8'h1e;
        8'hea: al = 8'h87;
        8'heb: al = 8'he9;
        8'hec: al = 8'hce;
        8'hed: al = 8'h55;
        8'hee: al = 8'h28;
        8'hef: al = 8'hdf;
        8'hf0: al = 8'h8c;
        8'hf1: al = 8'ha1;
        8'hf2: al = 8'h89;
        8'hf3: al = 8'h0d;
        8'hf4: al = 8'hbf;
        8'hf5: al = 8'he6;
        8'hf6: al = 8'h42;
        8'hf7: al = 8'h68;
        8'hf8: al = 8'h41;
        8'hf9: al = 8'h99;
        8'hfa: al = 8'h2d;
        8'hfb: al = 8'h0f;
        8'hfc: al = 8'hb0;
        8'hfd: al = 8'h54;
        8'hfe: al = 8'hbb;
        8'hff: al = 8'h16;
      endcase
    end
    endfunction

    always @ (*)
    begin
        t0 <= alpha_in[7:0];
        t1 <= alpha_in[15:8];
        t2 <= alpha_in[23:16];
        t3 <= alpha_in[31:24];
    end

    assign shift_out = alpha_in >> 8;
    assign lut_out[7:0] = al(t0);
    assign lut_out[15:8] = al(t1);
    assign lut_out[23:16] = al(t2);
    assign lut_out[31:24] = al(t3);
    assign alpha_out = (lut_out ^ shift_out);

endmodule
```

Alpha Inverse

```
`timescale 1ns / 1ps

module alpha_inv(
    input [31:0] alpha_inv_in,
    output [31:0] alpha_inv_out
    );
    wire [31:0] lut_out;
    wire [31:0] shift_out;

    reg [8:0] t0=8'd0;
    reg [8:0] t1=8'd0;
    reg [8:0] t2=8'd0;
    reg [8:0] t3=8'd0;

    function [7 : 0] al(input [7 : 0] z);
    begin
      case (z)
        8'h00: al = 8'h63;
        8'h01: al = 8'h7c;
        8'h02: al = 8'h77;
```

```
21          8'h03: al = 8'h7b;
22          8'h04: al = 8'hf2;
23          8'h05: al = 8'h6b;
24          8'h06: al = 8'h6f;
25          8'h07: al = 8'hc5;
26          8'h08: al = 8'h30;
27          8'h09: al = 8'h01;
28          8'h0a: al = 8'h67;
29          8'h0b: al = 8'h2b;
30          8'h0c: al = 8'hfe;
31          8'h0d: al = 8'hd7;
32          8'h0e: al = 8'hab;
33          8'h0f: al = 8'h76;
34          8'h10: al = 8'hca;
35          8'h11: al = 8'h82;
36          8'h12: al = 8'hc9;
37          8'h13: al = 8'h7d;
38          8'h14: al = 8'hfa;
39          8'h15: al = 8'h59;
40          8'h16: al = 8'h47;
41          8'h17: al = 8'hf0;
42          8'h18: al = 8'had;
43          8'h19: al = 8'hd4;
44          8'h1a: al = 8'ha2;
45          8'h1b: al = 8'haf;
46          8'h1c: al = 8'h9c;
47          8'h1d: al = 8'ha4;
48          8'h1e: al = 8'h72;
49          8'h1f: al = 8'hc0;
50          8'h20: al = 8'hb7;
51          8'h21: al = 8'hfd;
52          8'h22: al = 8'h93;
53          8'h23: al = 8'h26;
54          8'h24: al = 8'h36;
55          8'h25: al = 8'h3f;
56          8'h26: al = 8'hf7;
57          8'h27: al = 8'hcc;
58          8'h28: al = 8'h34;
59          8'h29: al = 8'ha5;
60          8'h2a: al = 8'he5;
61          8'h2b: al = 8'hf1;
62          8'h2c: al = 8'h71;
63          8'h2d: al = 8'hd8;
64          8'h2e: al = 8'h31;
65          8'h2f: al = 8'h15;
66          8'h30: al = 8'h04;
67          8'h31: al = 8'hc7;
68          8'h32: al = 8'h23;
69          8'h33: al = 8'hc3;
70          8'h34: al = 8'h18;
71          8'h35: al = 8'h96;
72          8'h36: al = 8'h05;
73          8'h37: al = 8'h9a;
74          8'h38: al = 8'h07;
75          8'h39: al = 8'h12;
76          8'h3a: al = 8'h80;
77          8'h3b: al = 8'he2;
78          8'h3c: al = 8'heb;
79          8'h3d: al = 8'h27;
80          8'h3e: al = 8'hb2;
81          8'h3f: al = 8'h75;
82          8'h40: al = 8'h09;
83          8'h41: al = 8'h83;
84          8'h42: al = 8'h2c;
85          8'h43: al = 8'h1a;
86          8'h44: al = 8'h1b;
87          8'h45: al = 8'h6e;
88          8'h46: al = 8'h5a;
89          8'h47: al = 8'ha0;
```

```
90          8'h48: al = 8'h52;
91          8'h49: al = 8'h3b;
92          8'h4a: al = 8'hd6;
93          8'h4b: al = 8'hb3;
94          8'h4c: al = 8'h29;
95          8'h4d: al = 8'he3;
96          8'h4e: al = 8'h2f;
97          8'h4f: al = 8'h84;
98          8'h50: al = 8'h53;
99          8'h51: al = 8'hd1;
100         8'h52: al = 8'h00;
101         8'h53: al = 8'hed;
102         8'h54: al = 8'h20;
103         8'h55: al = 8'hfc;
104         8'h56: al = 8'hb1;
105         8'h57: al = 8'h5b;
106         8'h58: al = 8'h6a;
107         8'h59: al = 8'hcb;
108         8'h5a: al = 8'hbe;
109         8'h5b: al = 8'h39;
110         8'h5c: al = 8'h4a;
111         8'h5d: al = 8'h4c;
112         8'h5e: al = 8'h58;
113         8'h5f: al = 8'hcf;
114         8'h60: al = 8'hd0;
115         8'h61: al = 8'hef;
116         8'h62: al = 8'haa;
117         8'h63: al = 8'hfb;
118         8'h64: al = 8'h43;
119         8'h65: al = 8'h4d;
120         8'h66: al = 8'h33;
121         8'h67: al = 8'h85;
122         8'h68: al = 8'h45;
123         8'h69: al = 8'hf9;
124         8'h6a: al = 8'h02;
125         8'h6b: al = 8'h7f;
126         8'h6c: al = 8'h50;
127         8'h6d: al = 8'h3c;
128         8'h6e: al = 8'h9f;
129         8'h6f: al = 8'ha8;
130         8'h70: al = 8'h51;
131         8'h71: al = 8'ha3;
132         8'h72: al = 8'h40;
133         8'h73: al = 8'h8f;
134         8'h74: al = 8'h92;
135         8'h75: al = 8'h9d;
136         8'h76: al = 8'h38;
137         8'h77: al = 8'hf5;
138         8'h78: al = 8'hbc;
139         8'h79: al = 8'hb6;
140         8'h7a: al = 8'hda;
141         8'h7b: al = 8'h21;
142         8'h7c: al = 8'h10;
143         8'h7d: al = 8'hff;
144         8'h7e: al = 8'hf3;
145         8'h7f: al = 8'hd2;
146         8'h80: al = 8'hcd;
147         8'h81: al = 8'h0c;
148         8'h82: al = 8'h13;
149         8'h83: al = 8'hec;
150         8'h84: al = 8'h5f;
151         8'h85: al = 8'h97;
152         8'h86: al = 8'h44;
153         8'h87: al = 8'h17;
154         8'h88: al = 8'hc4;
155         8'h89: al = 8'ha7;
156         8'h8a: al = 8'h7e;
157         8'h8b: al = 8'h3d;
158         8'h8c: al = 8'h64;
```

```
159         8'h8d: al = 8'h5d;
160         8'h8e: al = 8'h19;
161         8'h8f: al = 8'h73;
162         8'h90: al = 8'h60;
163         8'h91: al = 8'h81;
164         8'h92: al = 8'h4f;
165         8'h93: al = 8'hdc;
166         8'h94: al = 8'h22;
167         8'h95: al = 8'h2a;
168         8'h96: al = 8'h90;
169         8'h97: al = 8'h88;
170         8'h98: al = 8'h46;
171         8'h99: al = 8'hee;
172         8'h9a: al = 8'hb8;
173         8'h9b: al = 8'h14;
174         8'h9c: al = 8'hde;
175         8'h9d: al = 8'h5e;
176         8'h9e: al = 8'h0b;
177         8'h9f: al = 8'hdb;
178         8'ha0: al = 8'he0;
179         8'ha1: al = 8'h32;
180         8'ha2: al = 8'h3a;
181         8'ha3: al = 8'h0a;
182         8'ha4: al = 8'h49;
183         8'ha5: al = 8'h06;
184         8'ha6: al = 8'h24;
185         8'ha7: al = 8'h5c;
186         8'ha8: al = 8'hc2;
187         8'ha9: al = 8'hd3;
188         8'haa: al = 8'hac;
189         8'hab: al = 8'h62;
190         8'hac: al = 8'h91;
191         8'had: al = 8'h95;
192         8'hae: al = 8'he4;
193         8'haf: al = 8'h79;
194         8'hb0: al = 8'he7;
195         8'hb1: al = 8'hc8;
196         8'hb2: al = 8'h37;
197         8'hb3: al = 8'h6d;
198         8'hb4: al = 8'h8d;
199         8'hb5: al = 8'hd5;
200         8'hb6: al = 8'h4e;
201         8'hb7: al = 8'ha9;
202         8'hb8: al = 8'h6c;
203         8'hb9: al = 8'h56;
204         8'hba: al = 8'hf4;
205         8'hbb: al = 8'hea;
206         8'hbc: al = 8'h65;
207         8'hbd: al = 8'h7a;
208         8'hbe: al = 8'hae;
209         8'hbf: al = 8'h08;
210         8'hc0: al = 8'hba;
211         8'hc1: al = 8'h78;
212         8'hc2: al = 8'h25;
213         8'hc3: al = 8'h2e;
214         8'hc4: al = 8'h1c;
215         8'hc5: al = 8'ha6;
216         8'hc6: al = 8'hb4;
217         8'hc7: al = 8'hc6;
218         8'hc8: al = 8'he8;
219         8'hc9: al = 8'hdd;
220         8'hca: al = 8'h74;
221         8'hcb: al = 8'h1f;
222         8'hcc: al = 8'h4b;
223         8'hcd: al = 8'hbd;
224         8'hce: al = 8'h8b;
225         8'hcf: al = 8'h8a;
226         8'hd0: al = 8'h70;
227         8'hd1: al = 8'h3e;
```

```
        8'hd2: al = 8'hb5;
        8'hd3: al = 8'h66;
        8'hd4: al = 8'h48;
        8'hd5: al = 8'h03;
        8'hd6: al = 8'hf6;
        8'hd7: al = 8'h0e;
        8'hd8: al = 8'h61;
        8'hd9: al = 8'h35;
        8'hda: al = 8'h57;
        8'hdb: al = 8'hb9;
        8'hdc: al = 8'h86;
        8'hdd: al = 8'hc1;
        8'hde: al = 8'h1d;
        8'hdf: al = 8'h9e;
        8'he0: al = 8'he1;
        8'he1: al = 8'hf8;
        8'he2: al = 8'h98;
        8'he3: al = 8'h11;
        8'he4: al = 8'h69;
        8'he5: al = 8'hd9;
        8'he6: al = 8'h8e;
        8'he7: al = 8'h94;
        8'he8: al = 8'h9b;
        8'he9: al = 8'h1e;
        8'hea: al = 8'h87;
        8'heb: al = 8'he9;
        8'hec: al = 8'hce;
        8'hed: al = 8'h55;
        8'hee: al = 8'h28;
        8'hef: al = 8'hdf;
        8'hf0: al = 8'h8c;
        8'hf1: al = 8'ha1;
        8'hf2: al = 8'h89;
        8'hf3: al = 8'h0d;
        8'hf4: al = 8'hbf;
        8'hf5: al = 8'he6;
        8'hf6: al = 8'h42;
        8'hf7: al = 8'h68;
        8'hf8: al = 8'h41;
        8'hf9: al = 8'h99;
        8'hfa: al = 8'h2d;
        8'hfb: al = 8'h0f;
        8'hfc: al = 8'hb0;
        8'hfd: al = 8'h54;
        8'hfe: al = 8'hbb;
        8'hff: al = 8'h16;
      endcase
    end
    endfunction

    always @ (*)
    begin
        t0 <= alpha_inv_in[7:0];
        t1 <= alpha_inv_in[15:8];
        t2 <= alpha_inv_in[23:16];
        t3 <= alpha_inv_in[31:24];
    end

    assign shift_out = alpha_inv_in << 8;
    assign lut_out[7:0] = al(t0);
    assign lut_out[15:8] = al(t1);
    assign lut_out[23:16] = al(t2);
    assign lut_out[31:24] = al(t3);
    assign alpha_inv_out = (lut_out ^ shift_out);

endmodule
```

Finite-State Machine (FSM)

```
`timescale 1ns / 1ps

module FSM (
  input [31:0] in1,
  input [31:0] in2,
  input clk,
  output [31:0] FSM_out
);
  reg [31:0] R1 = 32'd0;
  reg [31:0] R2 = 32'd0;
  wire [31:0] x0;
  wire [31:0] x1;
  wire [31:0] x2;

  always @ (posedge clk)
  begin
    R1 <= x1;
    R2 <= x2;
  end

  assign FSM_out = (x0 ^ R2);

  Modulo_Addition m1(.a(in1), .b(R1), .clk(clk), .sum(x0));
  Modulo_Addition m2(.a(in2), .b(R2), .clk(clk), .sum(x1));
  S_Box s1(.in(R1), .clk(clk), .out(x2));
endmodule
```

S-Box

```
`timescale 1ns / 1ps

module S_Box(
    input [31:0] in,
    input clk,
    output [31:0] out
);
  reg [8:0] t0=8'd0;
  reg [8:0] t1=8'd0;
  reg [8:0] t2=8'd0;
  reg [8:0] t3=8'd0;

  function [7 : 0] sbox(input [7 : 0] z);
    begin
      case (z)
        8'h00: sbox = 8'h63;
        8'h01: sbox = 8'h7c;
        8'h02: sbox = 8'h77;
        8'h03: sbox = 8'h7b;
        8'h04: sbox = 8'hf2;
        8'h05: sbox = 8'h6b;
        8'h06: sbox = 8'h6f;
        8'h07: sbox = 8'hc5;
        8'h08: sbox = 8'h30;
        8'h09: sbox = 8'h01;
        8'h0a: sbox = 8'h67;
        8'h0b: sbox = 8'h2b;
        8'h0c: sbox = 8'hfe;
        8'h0d: sbox = 8'hd7;
        8'h0e: sbox = 8'hab;
        8'h0f: sbox = 8'h76;
        8'h10: sbox = 8'hca;
        8'h11: sbox = 8'h82;
        8'h12: sbox = 8'hc9;
        8'h13: sbox = 8'h7d;
        8'h14: sbox = 8'hfa;
        8'h15: sbox = 8'h59;
```

```
38          8'h16: sbox = 8'h47;
39          8'h17: sbox = 8'hf0;
40          8'h18: sbox = 8'had;
41          8'h19: sbox = 8'hd4;
42          8'h1a: sbox = 8'ha2;
43          8'h1b: sbox = 8'haf;
44          8'h1c: sbox = 8'h9c;
45          8'h1d: sbox = 8'ha4;
46          8'h1e: sbox = 8'h72;
47          8'h1f: sbox = 8'hc0;
48          8'h20: sbox = 8'hb7;
49          8'h21: sbox = 8'hfd;
50          8'h22: sbox = 8'h93;
51          8'h23: sbox = 8'h26;
52          8'h24: sbox = 8'h36;
53          8'h25: sbox = 8'h3f;
54          8'h26: sbox = 8'hf7;
55          8'h27: sbox = 8'hcc;
56          8'h28: sbox = 8'h34;
57          8'h29: sbox = 8'ha5;
58          8'h2a: sbox = 8'he5;
59          8'h2b: sbox = 8'hf1;
60          8'h2c: sbox = 8'h71;
61          8'h2d: sbox = 8'hd8;
62          8'h2e: sbox = 8'h31;
63          8'h2f: sbox = 8'h15;
64          8'h30: sbox = 8'h04;
65          8'h31: sbox = 8'hc7;
66          8'h32: sbox = 8'h23;
67          8'h33: sbox = 8'hc3;
68          8'h34: sbox = 8'h18;
69          8'h35: sbox = 8'h96;
70          8'h36: sbox = 8'h05;
71          8'h37: sbox = 8'h9a;
72          8'h38: sbox = 8'h07;
73          8'h39: sbox = 8'h12;
74          8'h3a: sbox = 8'h80;
75          8'h3b: sbox = 8'he2;
76          8'h3c: sbox = 8'heb;
77          8'h3d: sbox = 8'h27;
78          8'h3e: sbox = 8'hb2;
79          8'h3f: sbox = 8'h75;
80          8'h40: sbox = 8'h09;
81          8'h41: sbox = 8'h83;
82          8'h42: sbox = 8'h2c;
83          8'h43: sbox = 8'h1a;
84          8'h44: sbox = 8'h1b;
85          8'h45: sbox = 8'h6e;
86          8'h46: sbox = 8'h5a;
87          8'h47: sbox = 8'ha0;
88          8'h48: sbox = 8'h52;
89          8'h49: sbox = 8'h3b;
90          8'h4a: sbox = 8'hd6;
91          8'h4b: sbox = 8'hb3;
92          8'h4c: sbox = 8'h29;
93          8'h4d: sbox = 8'he3;
94          8'h4e: sbox = 8'h2f;
95          8'h4f: sbox = 8'h84;
96          8'h50: sbox = 8'h53;
97          8'h51: sbox = 8'hd1;
98          8'h52: sbox = 8'h00;
99          8'h53: sbox = 8'hed;
100         8'h54: sbox = 8'h20;
101         8'h55: sbox = 8'hfc;
102         8'h56: sbox = 8'hb1;
103         8'h57: sbox = 8'h5b;
104         8'h58: sbox = 8'h6a;
105         8'h59: sbox = 8'hcb;
106         8'h5a: sbox = 8'hbe;
```

```
107         8'h5b: sbox = 8'h39;
108         8'h5c: sbox = 8'h4a;
109         8'h5d: sbox = 8'h4c;
110         8'h5e: sbox = 8'h58;
111         8'h5f: sbox = 8'hcf;
112         8'h60: sbox = 8'hd0;
113         8'h61: sbox = 8'hef;
114         8'h62: sbox = 8'haa;
115         8'h63: sbox = 8'hfb;
116         8'h64: sbox = 8'h43;
117         8'h65: sbox = 8'h4d;
118         8'h66: sbox = 8'h33;
119         8'h67: sbox = 8'h85;
120         8'h68: sbox = 8'h45;
121         8'h69: sbox = 8'hf9;
122         8'h6a: sbox = 8'h02;
123         8'h6b: sbox = 8'h7f;
124         8'h6c: sbox = 8'h50;
125         8'h6d: sbox = 8'h3c;
126         8'h6e: sbox = 8'h9f;
127         8'h6f: sbox = 8'ha8;
128         8'h70: sbox = 8'h51;
129         8'h71: sbox = 8'ha3;
130         8'h72: sbox = 8'h40;
131         8'h73: sbox = 8'h8f;
132         8'h74: sbox = 8'h92;
133         8'h75: sbox = 8'h9d;
134         8'h76: sbox = 8'h38;
135         8'h77: sbox = 8'hf5;
136         8'h78: sbox = 8'hbc;
137         8'h79: sbox = 8'hb6;
138         8'h7a: sbox = 8'hda;
139         8'h7b: sbox = 8'h21;
140         8'h7c: sbox = 8'h10;
141         8'h7d: sbox = 8'hff;
142         8'h7e: sbox = 8'hf3;
143         8'h7f: sbox = 8'hd2;
144         8'h80: sbox = 8'hcd;
145         8'h81: sbox = 8'h0c;
146         8'h82: sbox = 8'h13;
147         8'h83: sbox = 8'hec;
148         8'h84: sbox = 8'h5f;
149         8'h85: sbox = 8'h97;
150         8'h86: sbox = 8'h44;
151         8'h87: sbox = 8'h17;
152         8'h88: sbox = 8'hc4;
153         8'h89: sbox = 8'ha7;
154         8'h8a: sbox = 8'h7e;
155         8'h8b: sbox = 8'h3d;
156         8'h8c: sbox = 8'h64;
157         8'h8d: sbox = 8'h5d;
158         8'h8e: sbox = 8'h19;
159         8'h8f: sbox = 8'h73;
160         8'h90: sbox = 8'h60;
161         8'h91: sbox = 8'h81;
162         8'h92: sbox = 8'h4f;
163         8'h93: sbox = 8'hdc;
164         8'h94: sbox = 8'h22;
165         8'h95: sbox = 8'h2a;
166         8'h96: sbox = 8'h90;
167         8'h97: sbox = 8'h88;
168         8'h98: sbox = 8'h46;
169         8'h99: sbox = 8'hee;
170         8'h9a: sbox = 8'hb8;
171         8'h9b: sbox = 8'h14;
172         8'h9c: sbox = 8'hde;
173         8'h9d: sbox = 8'h5e;
174         8'h9e: sbox = 8'h0b;
175         8'h9f: sbox = 8'hdb;
```

```
176         8'ha0: sbox = 8'he0;
177         8'ha1: sbox = 8'h32;
178         8'ha2: sbox = 8'h3a;
179         8'ha3: sbox = 8'h0a;
180         8'ha4: sbox = 8'h49;
181         8'ha5: sbox = 8'h06;
182         8'ha6: sbox = 8'h24;
183         8'ha7: sbox = 8'h5c;
184         8'ha8: sbox = 8'hc2;
185         8'ha9: sbox = 8'hd3;
186         8'haa: sbox = 8'hac;
187         8'hab: sbox = 8'h62;
188         8'hac: sbox = 8'h91;
189         8'had: sbox = 8'h95;
190         8'hae: sbox = 8'he4;
191         8'haf: sbox = 8'h79;
192         8'hb0: sbox = 8'he7;
193         8'hb1: sbox = 8'hc8;
194         8'hb2: sbox = 8'h37;
195         8'hb3: sbox = 8'h6d;
196         8'hb4: sbox = 8'h8d;
197         8'hb5: sbox = 8'hd5;
198         8'hb6: sbox = 8'h4e;
199         8'hb7: sbox = 8'ha9;
200         8'hb8: sbox = 8'h6c;
201         8'hb9: sbox = 8'h56;
202         8'hba: sbox = 8'hf4;
203         8'hbb: sbox = 8'hea;
204         8'hbc: sbox = 8'h65;
205         8'hbd: sbox = 8'h7a;
206         8'hbe: sbox = 8'hae;
207         8'hbf: sbox = 8'h08;
208         8'hc0: sbox = 8'hba;
209         8'hc1: sbox = 8'h78;
210         8'hc2: sbox = 8'h25;
211         8'hc3: sbox = 8'h2e;
212         8'hc4: sbox = 8'h1c;
213         8'hc5: sbox = 8'ha6;
214         8'hc6: sbox = 8'hb4;
215         8'hc7: sbox = 8'hc6;
216         8'hc8: sbox = 8'he8;
217         8'hc9: sbox = 8'hdd;
218         8'hca: sbox = 8'h74;
219         8'hcb: sbox = 8'h1f;
220         8'hcc: sbox = 8'h4b;
221         8'hcd: sbox = 8'hbd;
222         8'hce: sbox = 8'h8b;
223         8'hcf: sbox = 8'h8a;
224         8'hd0: sbox = 8'h70;
225         8'hd1: sbox = 8'h3e;
226         8'hd2: sbox = 8'hb5;
227         8'hd3: sbox = 8'h66;
228         8'hd4: sbox = 8'h48;
229         8'hd5: sbox = 8'h03;
230         8'hd6: sbox = 8'hf6;
231         8'hd7: sbox = 8'h0e;
232         8'hd8: sbox = 8'h61;
233         8'hd9: sbox = 8'h35;
234         8'hda: sbox = 8'h57;
235         8'hdb: sbox = 8'hb9;
236         8'hdc: sbox = 8'h86;
237         8'hdd: sbox = 8'hc1;
238         8'hde: sbox = 8'h1d;
239         8'hdf: sbox = 8'h9e;
240         8'he0: sbox = 8'he1;
241         8'he1: sbox = 8'hf8;
242         8'he2: sbox = 8'h98;
243         8'he3: sbox = 8'h11;
244         8'he4: sbox = 8'h69;
```

```
        8'he5: sbox = 8'hd9;
        8'he6: sbox = 8'h8e;
        8'he7: sbox = 8'h94;
        8'he8: sbox = 8'h9b;
        8'he9: sbox = 8'h1e;
        8'hea: sbox = 8'h87;
        8'heb: sbox = 8'he9;
        8'hec: sbox = 8'hce;
        8'hed: sbox = 8'h55;
        8'hee: sbox = 8'h28;
        8'hef: sbox = 8'hdf;
        8'hf0: sbox = 8'h8c;
        8'hf1: sbox = 8'ha1;
        8'hf2: sbox = 8'h89;
        8'hf3: sbox = 8'h0d;
        8'hf4: sbox = 8'hbf;
        8'hf5: sbox = 8'he6;
        8'hf6: sbox = 8'h42;
        8'hf7: sbox = 8'h68;
        8'hf8: sbox = 8'h41;
        8'hf9: sbox = 8'h99;
        8'hfa: sbox = 8'h2d;
        8'hfb: sbox = 8'h0f;
        8'hfc: sbox = 8'hb0;
        8'hfd: sbox = 8'h54;
        8'hfe: sbox = 8'hbb;
        8'hff: sbox = 8'h16;
      endcase
    end
  endfunction

  always @ (posedge clk)
  begin
    t0 <= in[7:0];
    t1 <= in[15:8];
    t2 <= in[23:16];
    t3 <= in[31:24];
  end

  assign out[7:0] = sbox(t0);
  assign out[15:8] = sbox(t1);
  assign out[23:16] = sbox(t2);
  assign out[31:24] = sbox(t3);
endmodule
```

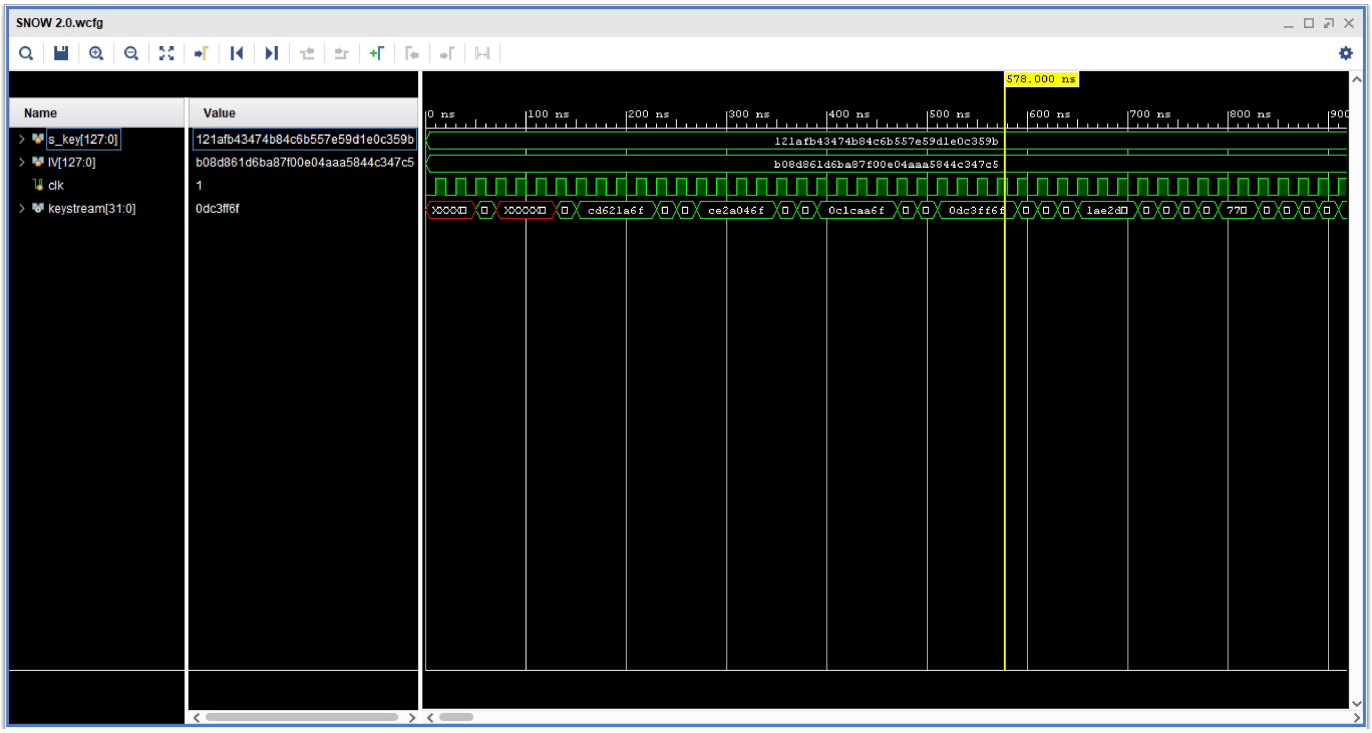
Modulo Addition

```
`timescale 1ns / 1ps

module Modulo_Addition #(parameter two_32=40'd2147483648)(
    input [31:0] a,
    input [31:0] b,
    input clk,
    output reg [31:0] sum
);
    integer a1,b1,sum1;

    always@(posedge clk)
    begin
        a1 <= a;
        b1 <= b;
        sum1 <= a1 + b1;
        if (sum1 < two_32)
            sum <= sum1;
        else if (sum1 == two_32)
            sum <= 32'd0;
        else
            sum <= sum1 % two_32;
    end
endmodule
```

Testbench

```
`timescale 1ns / 1ps

module testbench;
    reg [127:0] s_key;
    reg [127:0] IV;
    reg clk = 0;
    wire [31:0] keystream;

    SNOW_2 tb(.s_key(s_key), .IV(IV), .clk(clk), .keystream(keystream));

    initial
    begin
        s_key = 128'h121afb43474b84c6b557e59d1e0c359b;
        IV = 128'hb08d861d6ba87f00e04aaa5844c347c5;
        clk=0;
        forever #10 clk = ~clk;
    end

endmodule
```

Simulation of SNOW 2.0 Algorithm



"SNOW 2.0" – Test values Result of Simulation

Test values of s_key IV taken in testbench –

- S_key = 128'h121afb43474b84c6b557e59d1e0c359b
- IV = 128'hb08d861d6ba87f00e04aaa5844c347c5
- Clock = 50 MHz (20 ns time period)

Result –

- From 0 ns to 130 ns, there is no output in on the keystream for security and initialization.
- From 130 ns, keystream starts giving output as : ..,cd621a6f,..,ce2a046f,. . . ,0c1caa6f,. . . ,0dc3ff6f,. . . .,1aa2da62,...an so on.