Digital Systems Design Laboratory (EC29005)

Experiment 10

Submitted by:
Aradhya Chakrabarti (*2205880*)
(CSE03, Group-04)

Aim:
To implement the A5/1 synchronous stream cipher used in GSM communications using Verilog HDL via behavioral modeling.

Software(s) Required:
- Xilinx Vivado 2016.1

Theory/Working Principle:
A stream cipher is a *symmetric cipher* which operates with a time-varying transformation on individual plaintext digits. In a stream cipher, a sequence of plaintext digits, m_0m_1..., is encrypted into a sequence of ciphertext digits c_0c_1... as follows:

1. A pseudo-random sequence s_0s_1..., called the *keystream*, is produced by a finite state automaton whose
   initial state is determined by a secret *key* and a public parameter (*frame*).
2. The ith keystream digit only depends on the secret key and on the (i − 1) previous ciphertext digits.
3. Then, the ith ciphertext digit is obtained by combining the ith plaintext digit with the ith keystream digit.

A5/1 is the symmetric cipher used for encrypting over-the-air transmissions in the GSM standard. A5/1 is used in most European countries, whereas a weaker cipher, called A5/2, is used in other countries.

A5/1 is a synchronous stream cipher based on linear feedback shift registers (LFSRs). It has a 64-bit secret key. A GSM conversation is transmitted as a sequence of 228-bit frames (114 bits in each direction) every 4.6 milliseconds. Each frame is XOR-ed with a 228-bit sequence produced by the A5/1 running-key generator. The initial state of this generator depends on the 64-bit *secret key*, K, which is fixed during the conversation, and on a 22-bit *public frame number*, F.

The *Initialization Phase* is as follows:

1. The A5/1 running-key generator consists of 3 LFSRs of lengths 19, 22, and 23.
2. Their characteristic polynomials are
   - $X^{19}+X^5+X^2+X+1$
   - $X^{22}+X+1$ and.
   - $X^{23}+X^{15}+X^2+X+1$
3. For each frame transmission, the 3 LFSRs are first initialized to zero.

4. Then, at time t = 1...64, the LFSRs are clocked (shift operation), and the key bit K_t is XOR-ed to the feedback bit of each LFSR.
5. For t = 65...86, the LFSRs are clocked in the same fashion, but the $(t - 64)^{th}$ bit of the frame number is now XOR-ed to the feedback bits.

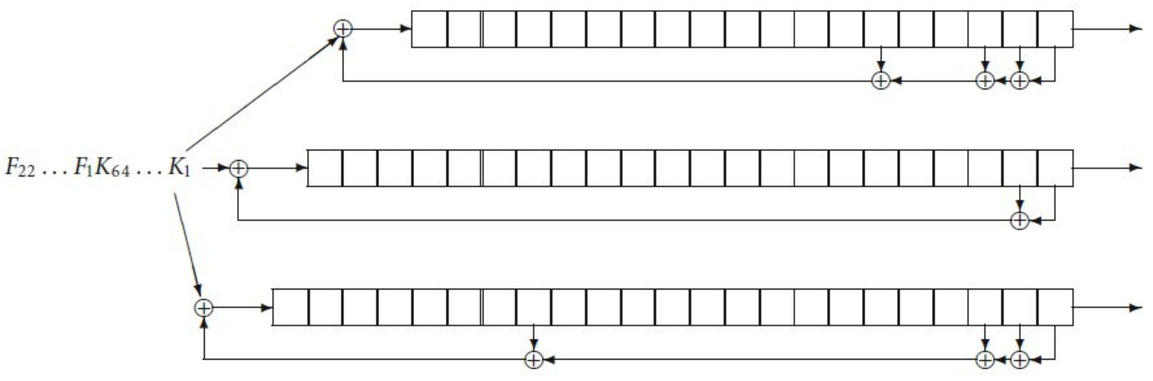


Fig1.1 Initialization of the A5/1 keystream generator

The *Keystream Generation* phase is as follows:

1. Each LFSR has a *clocking* tap: (*feedback* tap for each LFSR is at tap 0)
    a. At tap 8 for the first LFSR
    b. At tap 10 for the second LFSR and,
    c. At tap 10 for the the third LFSR
2. At each unit of time, the majority value *b* of the 3 clocking bits is computed. A LFSR is clocked *iff* its clocking bit is equal to *b*.
3. For example, if the 3 clocking bits are equal to (1, 0, 0), the majority value is 0. Then, the second and third LFSRs are clocked, but not the first one.
4. The output of the generator is then given by the XOR of the outputs of the 3 LFSRs. After the 86 initialization cycles, 328 bits are generated with the aforementioned pattern of irregular clocking.
5. The first 100 bits are discarded and the following 228 bits form the keystream.

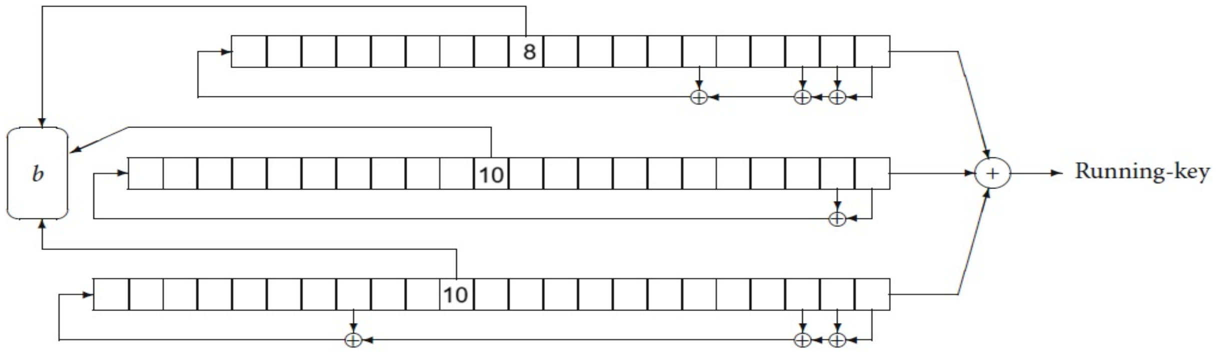
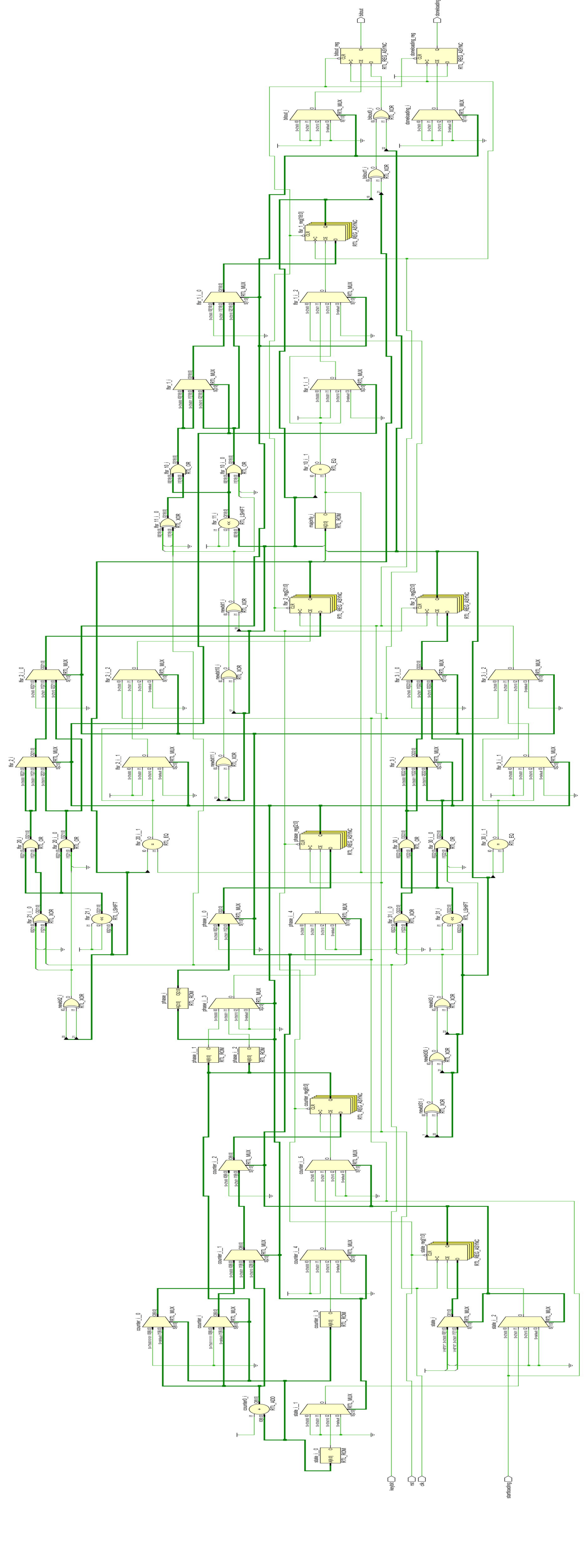


Fig1.2 Keystream Generation Phase

After the generation of the keystream, the plaintext to be encrypted is XOR-ed with the same.

RTL Schematic/Block Diagram:



Implementation:

Source Code (*Design*):

```
module a51(input clk, rst, keybit, startloading,
        output reg bitout, output reg doneloading);

reg [18:0] lfsr_1;
reg [21:0] lfsr_2;
reg [22:0] lfsr_3;

reg [1:0] state;
reg [6:0] counter;
reg [2:0] phase;

wire hi_1, hi_2, hi_3;
assign hi_1 = lfsr_1[18];
assign hi_2 = lfsr_2[21];
assign hi_3 = lfsr_3[22];

wire mid1, mid2, mid3;
assign mid1 = lfsr_1[8];
assign mid2 = lfsr_2[10];
assign mid3 = lfsr_3[10];

wire maj;
assign maj = majority(mid1, mid2, mid3);

wire newbit1, newbit2, newbit3;
assign newbit1 = ( lfsr_1[13] ^ lfsr_1[16] ^ lfsr_1[17] ^ lfsr_1[18] );
assign newbit2 = ( lfsr_2[20] ^ lfsr_2[21] ) ;
assign newbit3 = ( lfsr_3[7]  ^ lfsr_3[20] ^ lfsr_3[21] ^ lfsr_3[22] );

parameter IDLE=0, KEYING=1, RUNNING=2;

always @(posedge clk or negedge rst) begin
   if (!rst) begin: resetting
      $display("A5/1 Reset");
      doneloading <=0;
      bitout <=0;
      {lfsr_1, lfsr_2, lfsr_3} <= 64'h0;
      {state, counter, phase} <=0;
      end
   else begin
      case (state)
         IDLE: begin: reset_but_no_key
            if (startloading) begin: startloadingkey
               $display("Loading key starts at %0d ", $time);
               state <= KEYING;
               {lfsr_1, lfsr_2, lfsr_3} <= 64'h0;
               phase <=0; counter<=0;
```

```
                end
            end
        KEYING: begin
            case (phase)
                0: begin: load64andclock
                    clockallwithkey;
                    $display("Loading key bit %0b  %0d at %0d   %0x", Keybit, Counter, $time, lfsr_1);
                    if (counter == 63) begin
                        counter <= 0;
                        phase <= phase + 1;
                        $display(" ");
                    end
                    else counter <= counter + 1;
                end
                1: begin: load22andclock
                    $display("Loading frame bit %0b at %0d %0d   %0x", Keybit, Counter, $time, lfsr_1);
                    clockallwithkey;
                    if (counter == 21) begin
                        counter <= 0;
                        phase <= phase + 1;
                    end
                    else counter <= counter + 1;
                end
                2: begin: clock100
                    majclock;
                    if (counter == 100) begin
                        $display("Done keying, now running %0d\n", $time);
                        state <= RUNNING;
                    end
                    else counter <= counter + 1;
                end
            endcase
        end
        RUNNING: begin
            doneloading <= 1;
            bitout <= hi_1 ^ hi_2 ^ hi_3;
            majclock;
        end
    endcase
  end
end

function majority(input a,b,c); begin
  case({a,b,c})
    3'b000: majority=0;
    3'b001: majority=0;
    3'b010: majority=0;
    3'b011: majority=1;
    3'b100: majority=0;
```

```
        3'b101: majority=1;
        3'b110: majority=1;
        3'b111: majority=1;
    endcase
end
endfunction

task clock1; begin
    lfsr_1 <= ( lfsr_1 << 1 ) | newbit1;
end
endtask

task clock2; begin
    lfsr_2 <= (lfsr_2 << 1) | newbit2;
end
endtask

task clock3; begin
    lfsr_3 <= (lfsr_3 << 1) | newbit3;
end
endtask

task clockall; begin
    clock1;
    clock2;
    clock3;
end
endtask

task clockallwithkey; begin
    lfsr_1 <= ( lfsr_1 << 1 ) |  newbit1 ^ keybit;
    lfsr_2 <= ( lfsr_2 << 1 ) |  newbit2 ^ keybit;
    lfsr_3 <= ( lfsr_3 << 1 ) |  newbit3 ^ keybit;
end
endtask

task majclock; begin
    if (mid1 == maj) clock1;
    if (mid2 == maj) clock2;
    if (mid3 == maj) clock3;
end
endtask
endmodule
```

Source Code (*Simulation*):

```
module test_a51();
reg clk, rst, keybit, startloading;
wire bitout;
wire doneloading;
reg [0:7] key [7:0];
reg [22:0] frame;
a51 CKT1 (clk, rst, keybit, startloading, bitout, doneloading);
integer i,j;
initial begin #5
   key[0]= 8'h12;
   key[1]= 8'h23;
   key[2]= 8'h45;
   key[3]= 8'h67;
   key[4]= 8'h89;
   key[5]= 8'hAB;
   key[6]= 8'hCD;
   key[7]= 8'hEF;
   frame <= 22'h134;
   clk <= 0; rst <= 1; startloading <= 0;
   keybit <= 0;
   #10 rst <= 0; #10 rst <= 1; #100
   startloading <= 1; $display("Starting to key %0d", $time);
   for (i = 0; i < 8; i = i + 1) begin
      for (j = 0; j < 8; j = j + 1) begin
         #10 startloading <= 0;
         keybit <= key[i] >> j;
         end
   end
   for (i = 0; i < 22; i = i + 1) begin
      #10 keybit <= frame[i];
   end
   wait(Doneloading); $display("Done keying %0d", $time);
   $write("\nBits out: \n");
   repeat (32) #10 $write("%b", Bitout);
   $display("\nKnown value = \n%b", 32'h534EAA58);
   #1000 $display("\nSim done."); $finish;
end
always @(clk) #5 clk <= ~clk;
endmodule
```

Simulation Log:

```
Vivado Simulator 2016.1
Time resolution is 1 ps
A5/1 Reset
A5/1 Reset
Starting to key 125
Loading key starts at 130
Loading key bit 0  0 at 140   0
Loading key bit 1  1 at 150   0
Loading key bit 0  2 at 160   1
Loading key bit 0  3 at 170   2
Loading key bit 1  4 at 180   4
Loading key bit 0  5 at 190   9
Loading key bit 0  6 at 200   12
Loading key bit 0  7 at 210   24
Loading key bit 1  8 at 220   48
Loading key bit 1  9 at 230   91
Loading key bit 0  10 at 240   123
Loading key bit 0  11 at 250   246
Loading key bit 0  12 at 260   48c
Loading key bit 1  13 at 270   918
Loading key bit 0  14 at 280   1231
Loading key bit 0  15 at 290   2462
Loading key bit 1  16 at 300   48c5
Loading key bit 0  17 at 310   918b
Loading key bit 1  18 at 320   12316
Loading key bit 0  19 at 330   2462d
Loading key bit 0  20 at 340   48c5b
Loading key bit 0  21 at 350   118b7
Loading key bit 1  22 at 360   2316f
Loading key bit 0  23 at 370   462df
Loading key bit 1  24 at 380   c5be
Loading key bit 1  25 at 390   18b7d
Loading key bit 1  26 at 400   316fa
Loading key bit 0  27 at 410   62df5
Loading key bit 0  28 at 420   45beb
Loading key bit 1  29 at 430   b7d7
Loading key bit 1  30 at 440   16fae
Loading key bit 0  31 at 450   2df5d
Loading key bit 1  32 at 460   5bebb
Loading key bit 0  33 at 470   37d76
Loading key bit 0  34 at 480   6faed
Loading key bit 1  35 at 490   5f5db
Loading key bit 0  36 at 500   3ebb6
Loading key bit 0  37 at 510   7d76d
Loading key bit 0  38 at 520   7aedb
Loading key bit 1  39 at 530   75db6
Loading key bit 1  40 at 540   6bb6c
Loading key bit 1  41 at 550   576d8
Loading key bit 0  42 at 560   2edb0
Loading key bit 1  43 at 570   5db60
Loading key bit 0  44 at 580   3b6c1
Loading key bit 1  45 at 590   76d83
Loading key bit 0  46 at 600   6db07
```

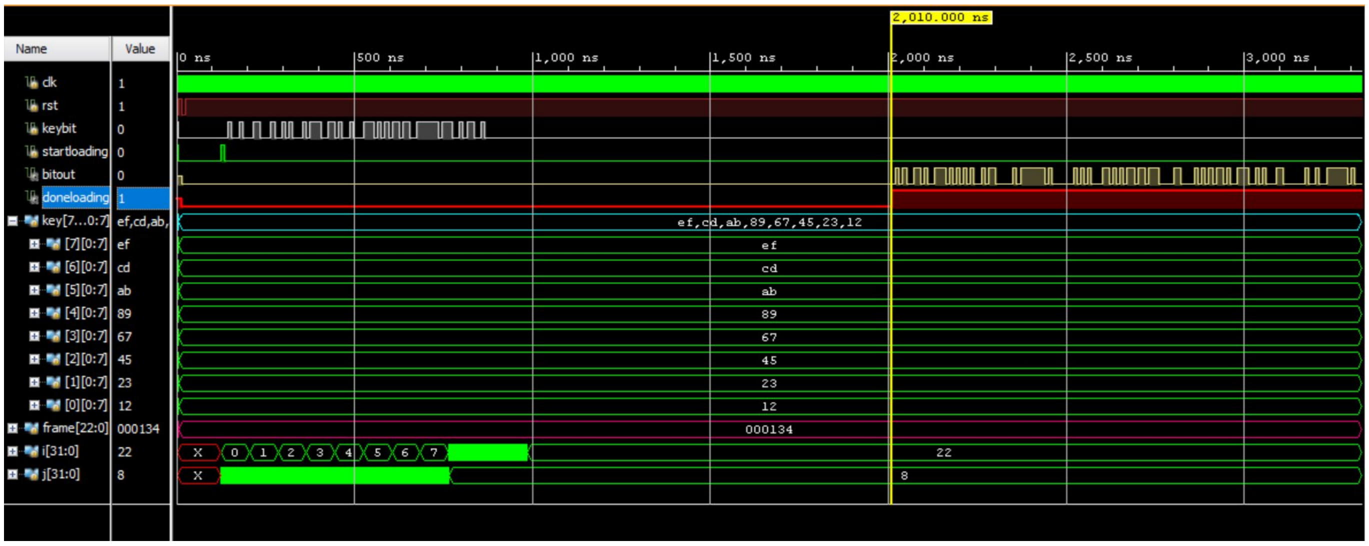
Behavioral Simulation:

```
Loading key bit 1  47 at 610   5b60e
Loading key bit 1  48 at 620   36c1c
Loading key bit 0  49 at 630   6d838
Loading key bit 1  50 at 640   5b070
Loading key bit 1  51 at 650   360e0
Loading key bit 0  52 at 660   6c1c0
Loading key bit 0  53 at 670   58380
Loading key bit 1  54 at 680   30700
Loading key bit 1  55 at 690   60e01
Loading key bit 1  56 at 700   41c03
Loading key bit 1  57 at 710   3806
Loading key bit 1  58 at 720   700c
Loading key bit 1  59 at 730   e018
Loading key bit 0  60 at 740   1c030
Loading key bit 1  61 at 750   38061
Loading key bit 1  62 at 760   700c3
Loading key bit 1  63 at 770   60186

Loading frame bit 0 at 0 780   4030d
Loading frame bit 0 at 1 790   61b
Loading frame bit 1 at 2 800   c36
Loading frame bit 0 at 3 810   186d
Loading frame bit 1 at 4 820   30da
Loading frame bit 1 at 5 830   61b4
Loading frame bit 0 at 6 840   c368
Loading frame bit 0 at 7 850   186d0
Loading frame bit 1 at 8 860   30da1
Loading frame bit 0 at 9 870   61b43
Loading frame bit 0 at 10 880   43686
Loading frame bit 0 at 11 890   6d0c
Loading frame bit 0 at 12 900   da19
Loading frame bit 0 at 13 910   1b432
Loading frame bit 0 at 14 920   36864
Loading frame bit 0 at 15 930   6d0c9
Loading frame bit 0 at 16 940   5a192
Loading frame bit 0 at 17 950   34325
Loading frame bit 0 at 18 960   6864a
Loading frame bit 0 at 19 970   50c94
Loading frame bit 0 at 20 980   21928
Loading frame bit 0 at 21 990   43251
Done keying, now running 2000

Done keying 2010
Bits out:
0101001101001110101010101001011000
Known value =
0101001101001110101010101001011000
Sim done.
$finish called at time : 3330 ns : File
"C:/Users/AradhyaPC/Desktop/Digital Systems
Design ab/expt10/expt10.srcs/sim_1/new/test_a51.v"
Line 37
```

Waveform Output:



Observation:
It can be observed in the waveform output that the encryption process through the A5/1 cipher has mainly 3 parts, initialization, keystream generation (with and without the majority function) and XOR-ing of the keystream and plaintext. 64 clock cycles are used to feed the key bits to the 3 LFSRs, 22 to feed the known frame, and 100 additional clock cycles are used to initialize with the majority function. Finally, 228 clock cycles are required to generate the final keystream through the LFSRs to XOR with the plaintext. As can be seen in the *Simulation Log*, the encrypted value (Bits out) and the known value turn out to be the same, implying that this Verilog HDL implementation functions correctly.

Conclusion:
The theoretical working principle of the A5/1 stream cipher is straightforward (and arguably, devoid of much mathematical consideration). It is easy to implement via hardware and the short size of the key and frame make it quick in its functioning. However, it is not very strong or secure and can be easily broken through currently available computing power, and this vulnerability is still extensively exploited. A5/1 has largely been replaced by the A5/3 block cipher (used in GPRS/EDGE communications).