DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING, IIIT BHUBANESWAR

VLSI LAB RECORD

SUBMITTED BY:
SUDIP NAYAK
B. Tech (3rd YEAR, 6TH SEMESTER, B220061)

CONTENTS

EXPERIMENT 01

AIM

In this experiment, our aim was to simulate various logic gates (AND, OR, XOR) in Verilog using both Structural and Data Flow Modelling.

SOFTWARE USED

Vivado 2018.3

STRUCTURAL MODELLING

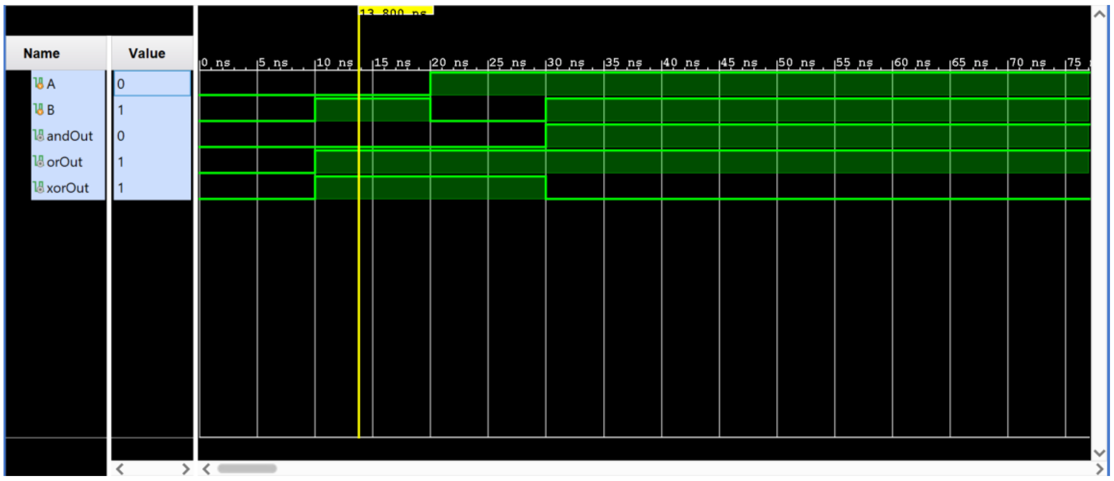
VERILOG CODE

```
`timescale 1ns / 1ps

module SM61(
   input A1,
   input B2,
   output andOut1,
   output orOut1,
   output xorOut1
   );
   and a3(andOut1, A1, B2);
   or o3(orOut1, A1, B2);
   xor x3(xorOut1, A1, B2);
endmodule
```

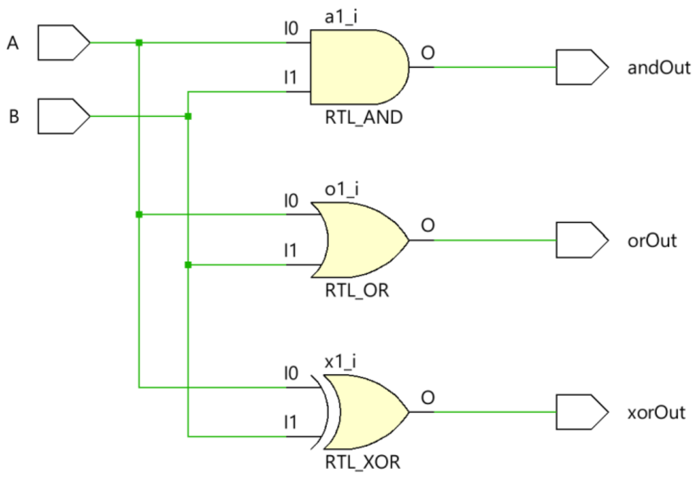
TESTBENCH

```
`timescale 1ns / 1ps

module StructuralTestBench61;
reg A1, B2;
wire andOut1, orOut1, xorOut1;
SM61 sm(.A1(A1), .B2(B2), .andOut1(andOut1), .orOut1(orOut1), .xorOut1(xorOut1));
   initial begin
      A1 = 0 ; B2 = 0 ;
      #10
      A1 = 0 ; B2 = 1 ;
      #10
      A1 = 1 ; B2 = 0 ;
      #10
      A1 = 1 ; B2 = 1 ;
   end
endmodule
```

WAVE WINDOW OUTPUT



SCHEMATIC



DATA FLOW MODELLING

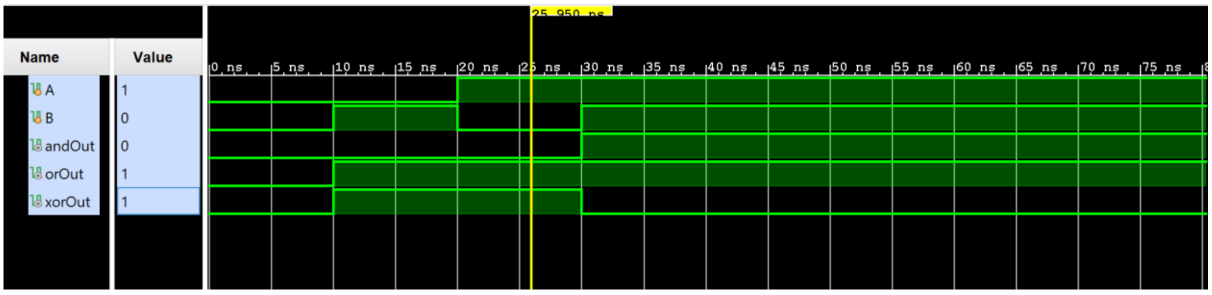
VERILOG CODE

```
`timescale 1ns / 1ps

module SM61(
   input A2,
   input B3,
   output andOut1,
   output orOut1,
   output xorOut1
   );
   assign andOut1 = A2 & B3 ;
   assign orOut1 =  A2 | B3 ;
   assign xorOut1 = A2 ^ B3 ;
endmodule
```

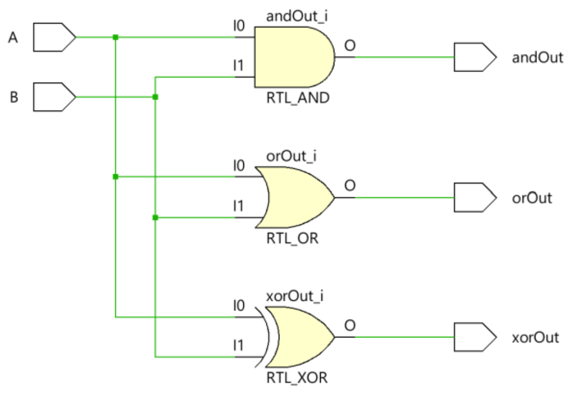
TESTBENCH

```
`timescale 1ns / 1ps

module BehaviouralTestBench22;
reg A2, B3;
wire andOut1, orOut1, xorOut1;
SM61 bm(.A2(A2), .B3(B3), .andOut1(andOut1), .orOut1(orOut1), .xorOut1(xorOut1));
  initial begin
     A2 = 0 ; B3 = 0 ;
     #10
     A2 = 0 ; B3 = 1 ;
     #10
     A2 = 1 ; B3 = 0 ;
     #10
     A2 = 1 ; B3 = 1 ;
  end
endmodule
```

WAVE WINDOW OUTPUT



SCHEMATIC



CONCLUSION

Hence, we successfully simulated various logic gates in Verilog using both structural and data flow modelling and observed the waveforms and schematic.

EXPERIMENT 02

AIM

In this experiment, our aim was to implement half adder and full adder in Verilog using both Structural and Data Flow modelling.

SOFTWARE USED

Vivado 2018.3

HALF ADDER USING STRUCTURAL MODELLING

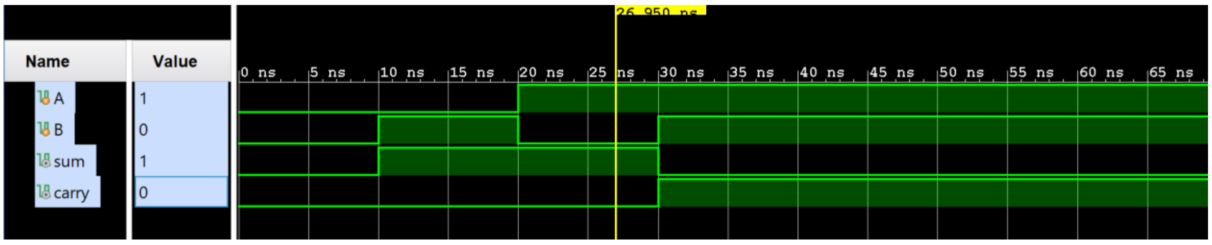
VERILOG CODE

```
`timescale 1ns / 1ps

module SM61(
   input Am,
   input Bn,
   output Sp,
   output Co
   );
   xor x3(Sp, Am, Bn);
   and a3(Co, Am, Bn);
endmodule
```

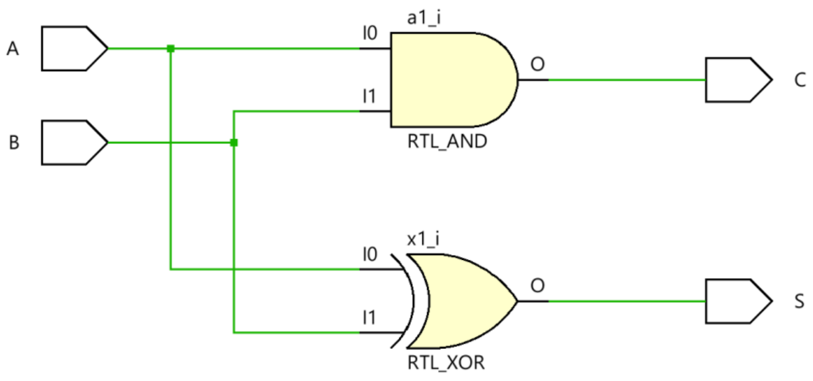
TESTBENCH

```
`timescale 1ns / 1ps

module StructuralTestBench;
reg Am, Bn;
wire sum, carry;
SM44 sm(.Am(Am), .Bn(Bn), .Sp(sum), .Co(carry));
initial begin
   Am = 0 ; Bn = 0 ;
   #10
   Am = 0 ; Bn = 1 ;
   #10
   Am = 1 ; Bn = 0 ;
   #10
   Am = 1 ; Bn = 1 ;
end
endmodule
```

WAVE WINDOW OUTPUT



SCHEMATIC



HALF ADDER USING DATA FLOW MODELLING

VERILOG CODE

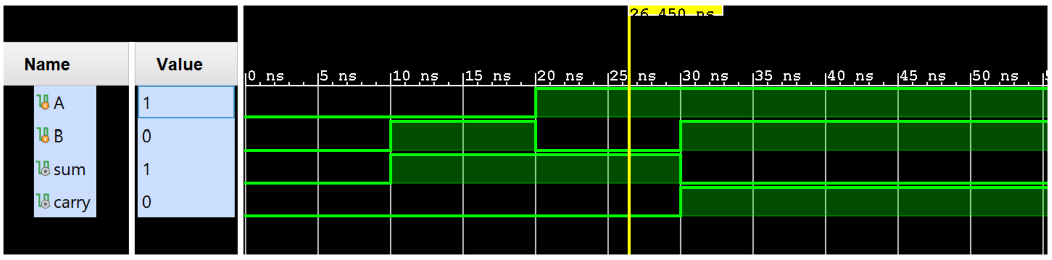
```
`timescale 1ns / 1ps

module DM61(
   input A1,
   input B2,
   output S3,
   output C4
   );
   assign S3 = A1 ^ B2 ;
   assign C4 = A1 & B2 ;
endmodule
```

TESTBENCH

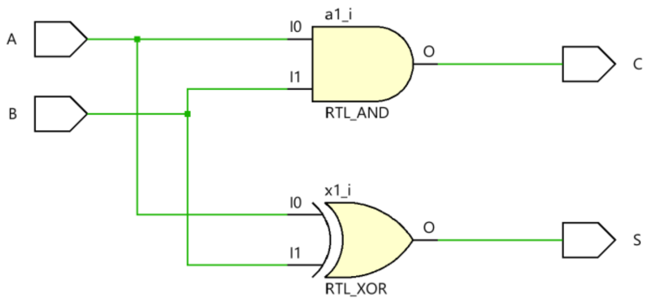
```
`timescale 1ns / 1ps

module DataFlowTestBench12;
reg A1, B2;
wire sum, carry;
DM44 dm(.A1(A1), .B2(B2), .S3(sum), .C4(carry));
initial begin
   A1 = 0 ; B2 = 0 ;
```

```
    #10
    A1 = 0 ; B2 = 1 ;
    #10
    A1 = 1 ; B2 = 0 ;
    #10
    A1 = 1 ; B2 = 1 ;
end
endmodule
```

WAVE WINDOW OUTPUT



SCHEMATIC



FULL ADDER USING STRUCTURAL MODELLING

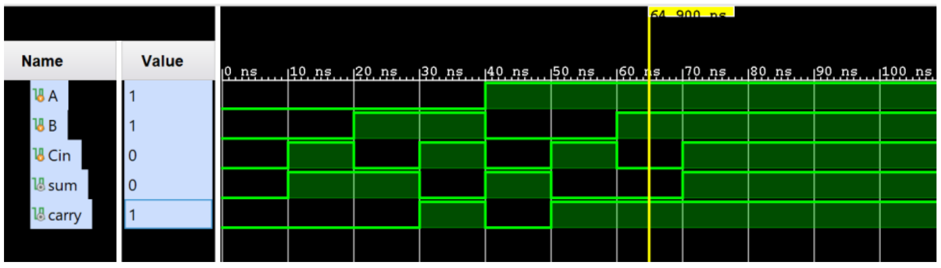
VERILOG CODE

```
`timescale 1ns / 1ps

module SM61(
    input Aa,
    input Bb,
    input Cin,
    output Ss,
    output Cout
    );
    wire a1, a2, x1 ;
    xor xor1(x1, Aa, Bb);
    xor xor2(S, x1, Cin);
    and and1(a1, x1, Cin);
    and and2(a2, Aa, Bb);
    or or1(Cout ,a1, a2) ;
endmodule
```

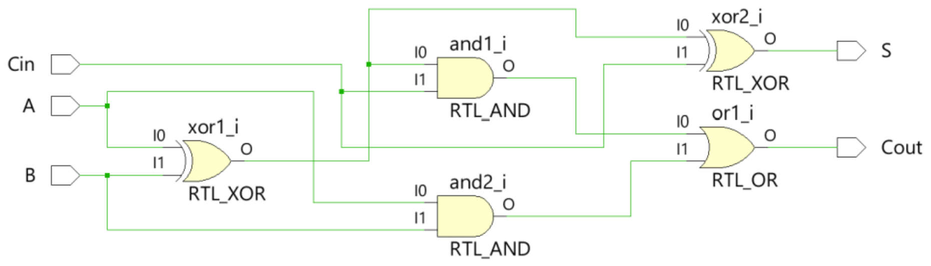
TESTBENCH

```
`timescale 1ns / 1ps

module StructuralTestBenchhh;
reg Aa, Bb, Cin;
wire sum, carry;
SM44 sm(.Aa(Aa), .Bb(Bb), .Cin(Cin), .Ss(sum), .Cout(carry));
    initial begin
        Aa = 0 ; Bb = 0 ; Cin = 0 ;
        #10
        Aa = 0 ; Bb = 0 ; Cin = 1 ;
        #10
        Aa = 0 ; Bb = 1 ; Cin = 0 ;
        #10
        Aa = 0 ; Bb = 1 ; Cin = 1 ;
        #10
        Aa = 1 ; Bb = 0 ; Cin = 0 ;
        #10
        Aa = 1 ; Bb = 0 ; Cin = 1 ;
        #10
        Aa = 1 ; Bb = 1 ; Cin = 0 ;
        #10
        Aa = 1 ; Bb = 1 ; Cin = 1 ;
    end
endmodule
```

WAVE WINDOW OUTPUT



SCHEMATIC



FULL ADDER USING DATA FLOW MODELLING

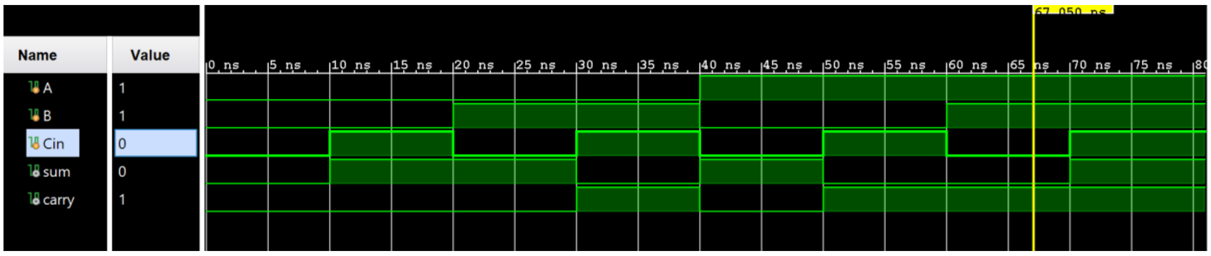
VERILOG CODE

```
`timescale 1ns / 1ps

module DM6(
   input Ai,
   input Bi,
   input Cin,
   output Si,
   output Cout
   );
   assign Si = Ai ^ Bi ^ Cin ;
   assign Cout = ((Ai ^ Bi) & Cin) | (Ai & Bi) ;
endmodule
```

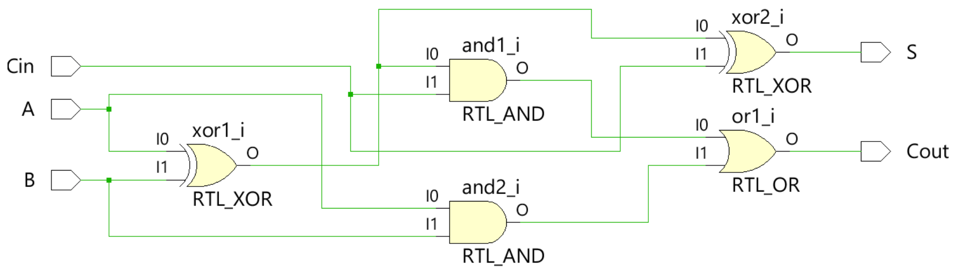
TESTBENCH

```
`timescale 1ns / 1ps
module DataFlowTestBench45;
reg At, Bt, Cin;
wire sum, carry;
DM61 dm(.At(At), .Bt(Bt), .Cin(Cin), .St(sum), .Cout(carry));
   initial begin
      At = 0 ; Bt = 0 ; Cin = 0 ;
      #10
      At = 0 ; Bt = 0 ; Cin = 1 ;
      #10
      At = 0 ; Bt = 1 ; Cin = 0 ;
      #10
      At = 0 ; Bt = 1 ; Cin = 1 ;
      #10
      At = 1 ; Bt = 0 ; Cin = 0 ;
      #10
      At = 1 ; Bt = 0 ; Cin = 1 ;
      #10
      At = 1 ; Bt = 1 ; Cin = 0 ;
      #10
      At = 1 ; Bt = 1 ; Cin = 1 ;
   end
endmodule
```

WAVE WINDOW OUTPUT



SCHEMATIC



CONCLUSION

Hence in this experiment, we successfully implemented half adder and full adder using both structural and data flow modelling in Verilog.

EXPERIMENT 03

AIM

In this experiment, our aim is to implement an 8-bit adder in Verilog using Structural Modelling.

SOFTWARE USED

Vivado 2018.3

VERILOG CODE

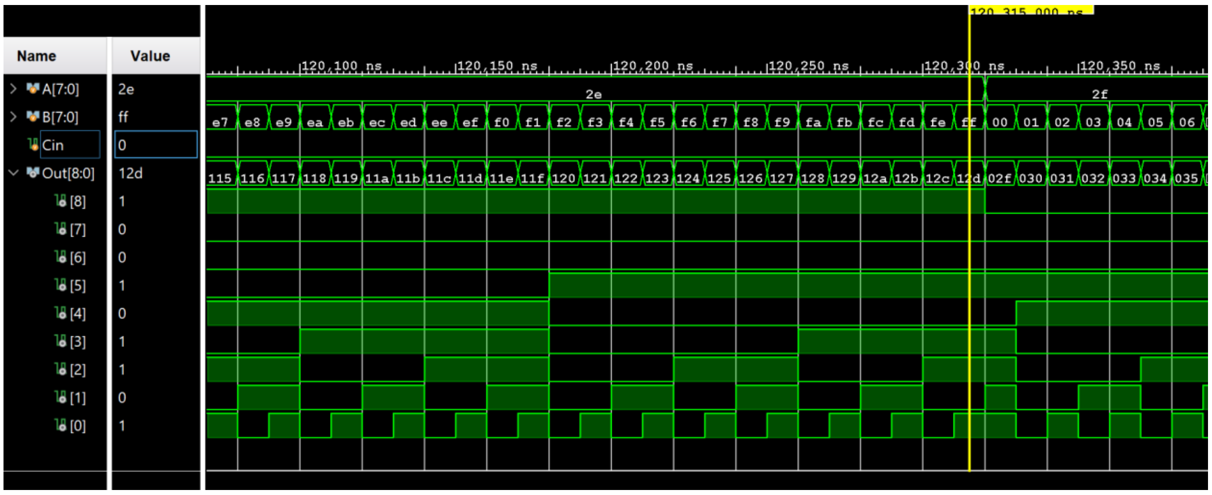
```
`timescale 1ns/1ps

module VerilogAdder61(
   input [7:0] Ain,
   input [7:0] Bin,
   input Cin,
   output [8:0] Out
   );
   wire[6:0] temp;
   FullAdder fa1(Ain[0], Bin[0], Cin, Out[0], temp[0]);
   FullAdder fa2(Ain[1], Bin[1], temp[0], Out[1], temp[1]);
   FullAdder fa3(Ain[2], Bin[2], temp[1], Out[2], temp[2]);
   FullAdder fa4(Ain[3], Bin[3], temp[2], Out[3], temp[3]);
   FullAdder fa5(Ain[4], Bin[4], temp[3], Out[4], temp[4]);
   FullAdder fa6(Ain[5], Bin[5], temp[4], Out[5], temp[5]);
   FullAdder fa7(Ain[6], Bin[6], temp[5], Out[6], temp[6]);
   FullAdder fa8(Ain[7], Bin[7], temp[6], Out[7], Out[8]);
Endmodule
```

TESTBENCH

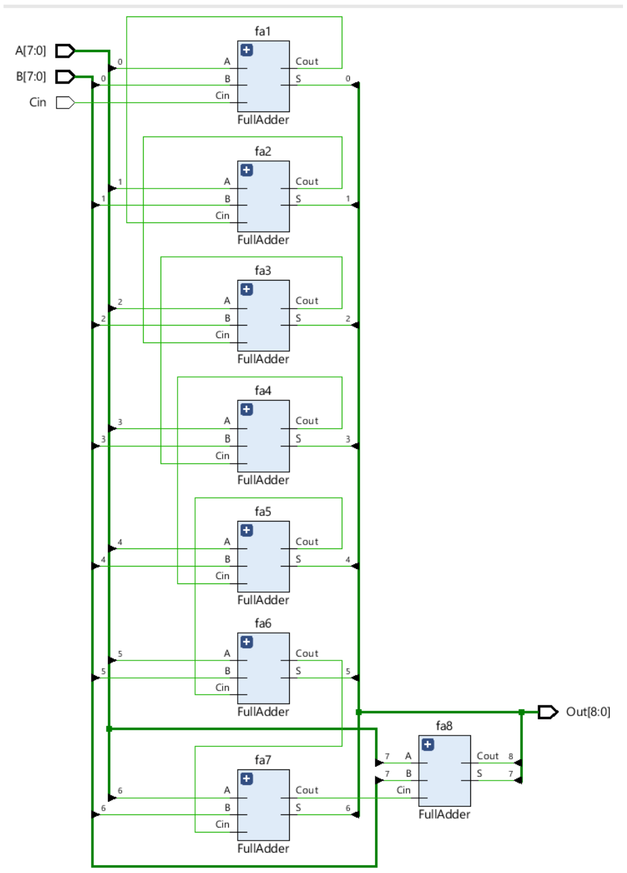
```
module AdderTestBench56;
reg[7:0] Ain;
reg[7:0] Bin;
reg Cin;
wire[8:0] Out;
VerilogAdder44 a1(.Ain(Ain), .Bin(B), .Cin(Cin), .Out(Out));
initial
begin
   Cin = 1'b0 ;
   for (Ain = 0; Ain < 8'b11111111; Ain = Ain + 1) begin
      for (Bin = 0; Bin < 8'b11111111; Bin = Bin + 1) begin
         #10;
      end
      #10;
   end
end
```

```
endmodule
```

WAVE WINDOW OUTPUT



SCHEMATIC



CONCLUSION

Hence in this experiment, we successfully implemented an 8-bit adder using structural Verilog.

EXPERIMENT 04

AIM

In this experiment, our aim was to implement a 4-bit multiplier using structural Verilog.

SOFTWARE USED

Vivado 2018.3

VERILOG CODE

XOR

```
`timescale 1ns / 1ps

module VerilogXOR61 (
   input A8,
   input B9,
   input C0,
   output O2
   );
   wire first1 ;
   xor(first1,A8,B9);
   xor(O2,first1,C0);
endmodule
```

AND

```
`timescale 1ns / 1ps

module AND61(
   input [3:0] An,
   input [3:0] Bn,
   output [3:0] Cn
   );
   and(Cn[0],An[0],Bn[0]);
   and(Cn[1],An[1],Bn[1]);
   and(Cn[2],An[2],Bn[2]);
   and(Cn[3],An[3],Bn[3]);
endmodule
```

FULL ADDER WITH CARRY

```
`timescale 1ns / 1ps

module VerilogCARRY61 (
   input Ae,
   input Be,
   input Ce,
   output Cout
   );
   wire w1, w2, w3, w4 ;
   and(w1,Ae,Be);
   and(w2,Ae,Ce);
   and(w3,Be,Ce);
   or(w4,w1,w2);
   or(Cout,w4,w3);
endmodule
```

STRUCTURAL ADDER

```
`timescale 1ns / 1ps

module VerilogFA61 (
   input Ao,
   input Bo,
   input Cin,
   output So,
   output Cout
   );
   VerilogXOR61x1(Ao,Bo,Cin,So);
   VerilogCARRY61 FC1(Ao,Bo,Cin,Cout);
endmodule
```

STRUCTURAL ADDER (4-BIT INPUT)

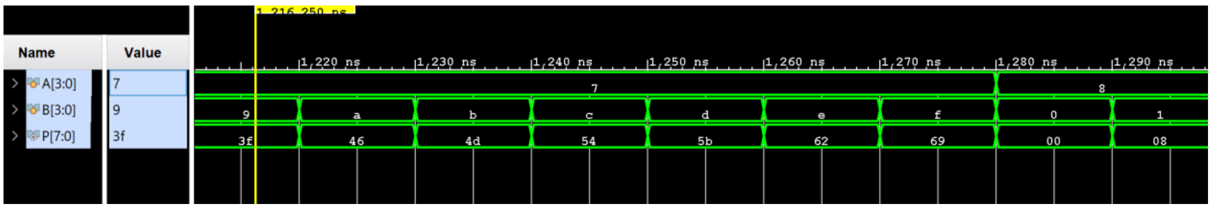
```
`timescale 1ns / 1ps

module VerilogADD44 (
   input [3:0] Ac,
   input [3:0] Bc,
   output Sc,
   output [3:0] Cout
   );
   wire C0,C1,C2 ;
   VerilogFA61 fa1(Ac[0],Bc[0],Sc,C0);
   VerilogFA61 fa2(Ac[1],Bc[1],C0,Cout[0],C1);
   VerilogFA61 fa3(Ac[2],Bc[2],C1,Cout[1],C2);
   VerilogFA61 fa4(Ac[3],Bc[3],C2,Cout[2],Cout[3]);
endmodule
```

MULTIPLIER

```
`timescale 1ns / 1ps
```

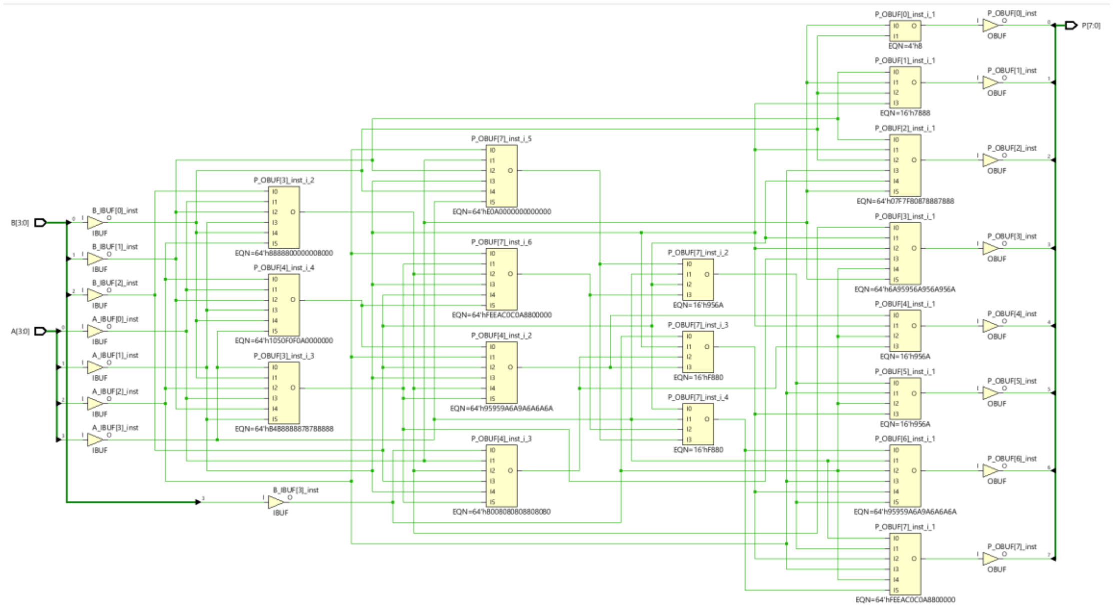
```
module VerilogMUL61 (
   input [3:0] Ar,
   input [3:0] Br,
   output [7:0] Pr
   );
   wire [3:0] L0, L1, L2, L3 ;
   wire [3:0] C0, C1, C2, C3 ;
   AND4 a40({0,Ar[3],Ar[2],Ar[1]},{Br[0],Br[0],Br[0],Br[0]},L0);
   AND4 a41(Ar,{Br[1],Br[1],Br[1],Br[1]},L1);
   AND4 a42(Ar,{Br[2],Br[2],Br[2],Br[2]},L2);
   AND4 a43(Ar,{Br[3],Br[3],Br[3],Br[3]},L3);

   and a1(Pr[0],Ar[0],Br[0]);
   VerilogADD61 sa40(L0,L1,P[1],C0);
   VerilogADD61 sa41(L2,C0,P[2],C1);
   VerilogADD61 sa42(L3,C1,Pr[3],{Pr[7],Pr[6],Pr[5],Pr[4]});
endmodule
```

WAVE WINDOW OUTPUT



```
A[3:0]  7   | 7                                              | 8
B[3:0]  9   | 9  | a  | b  | c  | d  | e  | f  | 0  | 1
P[7:0]  3f  | 3f | 46 | 4d | 54 | 5b | 62 | 69 | 00 | 08
```

SCHEMATIC



CONCLUSION

Hence in this experiment, we successfully implemented a 4-bit multiplier using structural Verilog.

EXPERIMENT 05

AIM

In this experiment, our aim is to implement an 8:3 Encoder and a 3:8 Decoder in Verilog using Behavioral Modelling.

SOFTWARE USED

Vivado 2018.3

VERILOG CODE

ENCODER 3:8

```
`timescale 1ns / 1ps

module VerilogEnc61 (
   input [8:0] Yj,
   output reg [3:0] Outi
   );
   always @(Yj) begin
      case(Yj)
         8'b00000001: Outi = 3'b000;
         8'b00000010: Outi = 3'b001;
         8'b00000100: Outi = 3'b010;
         8'b00001000: Outi = 3'b011;
         8'b00010000: Outi = 3'b100;
         8'b00100000: Outi = 3'b101;
         8'b01000000: Outi = 3'b110;
         8'b10000000:Outi = 3'b111;
         default: Outi = 3'bx;
      endcase
   end
endmodule
```

TEST BENCH

```
`timescale 1ns / 1ps

module EncoderTestBench9;
reg[7:0] Yj;
wire[3:0] Outi;
VerilogEnc61 enc(.Yj(Yj), .Outj(Outj));
initial
begin
   Y = 8'b00000001; #10;
   Y = 8'b00000010; #10;
   Y = 8'b00000100; #10;
   Y = 8'b00001000; #10;
   Y = 8'b00010000; #10;
   Y = 8'b00100000; #10;
   Y = 8'b01000000; #10;
   Y = 8'b10000000; #10;
end
endmodule
```

DECODER 8:3

```
`timescale 1ns / 1ps

module VerilogDec61 (
   input [2:0] Yi,
   output reg [7:0] Outj
   );
   always @(Yi) begin
      case(Yi)
         3'b000: Outj = 8'b00000001;
         3'b001: Outj = 8'b00000010;
         3'b010: Outj = 8'b00000100;
         3'b011: Outj = 8'b00001000;
         3'b100: Outj = 8'b00010000;
         3'b101: Outj = 8'b00100000;
         3'b110: Outj = 8'b01000000;
         3'b111: Outj = 8'b10000000;
      endcase
   end
endmodule
```

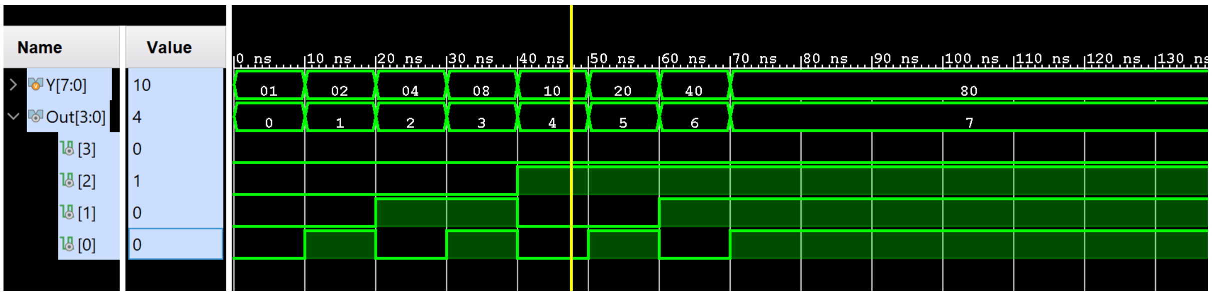
TEST BENCH

```
`timescale 1ns / 1ps


module DecoderTestBench12;
reg [2:0] Yi;
wire [7:0] Outj ;
VerilogDec61 decod(.Yi(Yi), .Outj(Outj));
initial begin
   Yi = 3'b000; #10;
   Yi = 3'b001; #10;
   Yi = 3'b010; #10;
   Yi = 3'b011; #10;
   Yi = 3'b100; #10;
   Yi = 3'b101; #10;
```

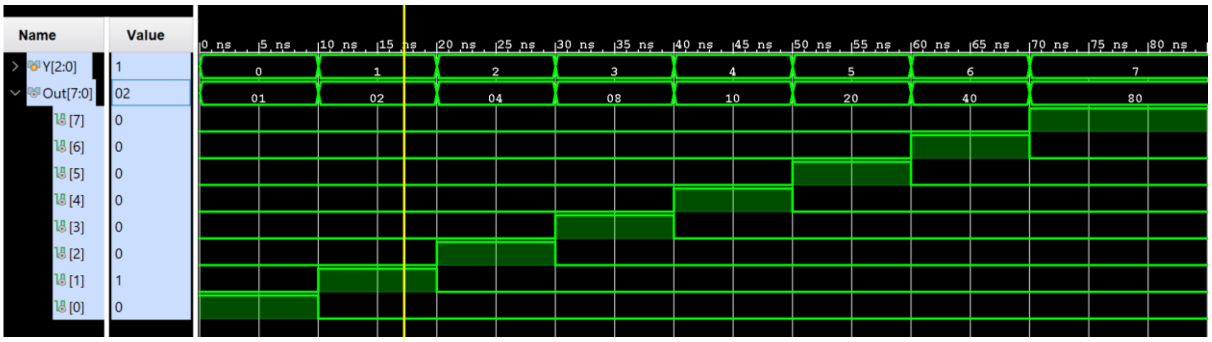
```
    Yi = 3'b110; #10;
    Yi = 3'b111; #10;
end
endmodule
```

WAVE WINDOW OUTPUT

ENCODER 3:8

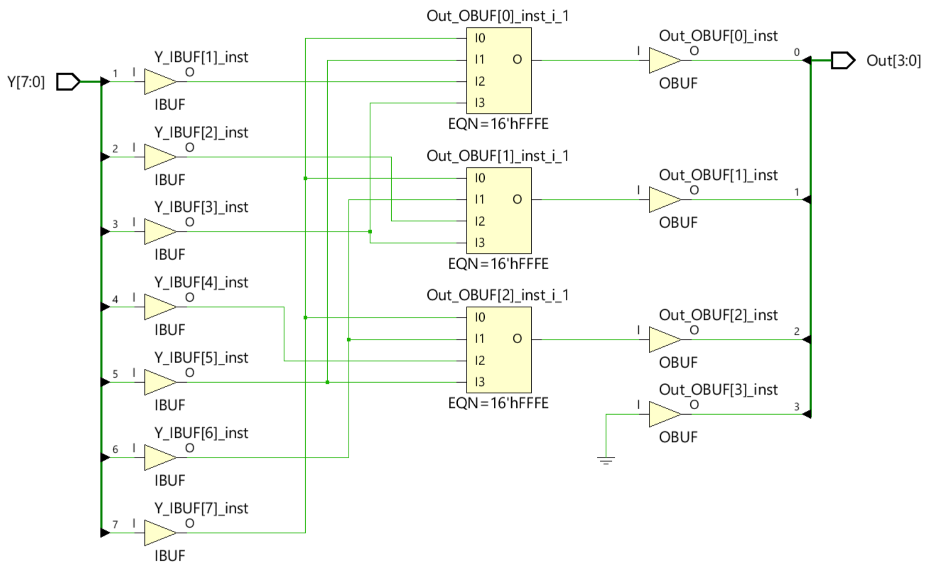


DECODER 8:3

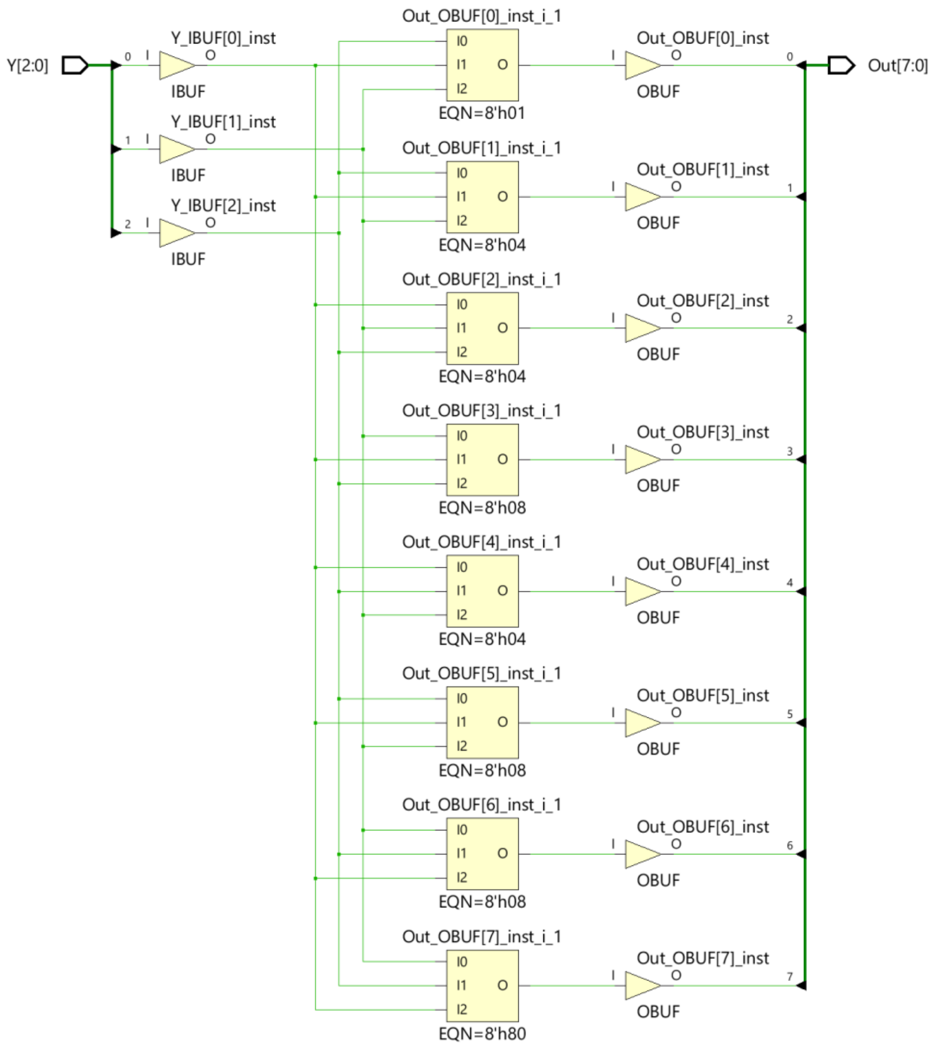


SCHEMATIC

ENCODER 3:8



DECODER 8:3



CONCLUSION

Hence in this experiment, we successfully implemented a 3:8 Decoder and a n 8:3 Encoder using Behavioral Modelling in Verilog.

EXPERIMENT 06

AIM

In this experiment, our aim is to implement latches (SR Latch & D Latch) & Flip-Flops (JK Flip-Flop & T Flip-Flop) in Verilog using Behavioral Modelling.

SOFTWARE USED

Vivado 2018.3

VERILOG CODE

SR LATCH

```
`timescale 1ns / 1ps

module VerilogSR_LATCH61 (
   input Sn, Rm,
   output reg Q1, Q_not2
   );
   always @ (Sn or Rm) begin
   if (Sn == 1 && Rm == 0) begin
      //Set
      Q1 = 1;
      Q_not2 = 0;
   end else if (Sn == 0 && Rm == 1) begin
      //Reset
      Q1 = 0;
      Q_not2 = 1;
   end else if (Sn == 0 && Rm == 0) begin
      //Hold State
      Q1 = Q1 ;
      Q_not2 = Q_not2 ;
   end else begin
      // invalid state
      Q1 = 1 ;
      Q_not2 = 1 ;
   end
end
endmodule
```

TEST BENCH

```
`timescale 1ns / 1ps

module SR_LatchTestBench6;
reg Sn, Rm  ;
wire Q1, Q_not2;
```

```
VerilogSR_LATCH61 sr(Sn, Rm, Q1, Q_not2);
initial
   begin
      Sn = 0;
      Rm = 0 ;
      repeat(4) begin
         #10 Sn = ~Sn;
         #10 Rm = ~Rm;
      end
   $stop;
end
endmodule
```

D LATCH

```
`timescale 1ns / 1ps

module VerilogD_LATCH61 (
   input D1,
   input Eno,
   output reg Q2,
   output reg Q_not3
   );
   always @ (D1 or Eno) begin
   if (Eno == 1) begin
      Q2 = D1;
      Q_not3 = ~D1 ;
   end
end
endmodule
```

TEST BENCH

```
`timescale 1ns / 1ps

module D_LatchTestBench1;
reg D1, Eno;
wire Q2, Q_not3;
VerilogD_LATCH61 dl(D1, Eno, Q2, Q_not3);
always #5 Eno = ~Eno;
initial
   begin
      D1 = 0 ;
      Eno = 0;
      repeat(4) begin
         #10 D1 = ~D1 ;
      end
   $stop;
end
endmodule
```

JK FLIP FLOP

```
`timescale 1ns / 1ps

module VerilogJK_LATCH61 (
   input Ji,
   input Ki,
   input clk,
   output reg Qi,
   output reg Q_not1
   );
   always @(posedge clk) begin
      if(Ji == 0 && Ki == 1) begin
         Qi = 0 ;
         Q_not1 = 1 ;
      end else if(Ji == 1 && Ki == 0) begin
         Qi = 1 ;
         Q_not1 = 0 ;
      end else if(Ji == 1 && Ki == 1) begin
         Qi = ~Qi;
         Q_not1 = ~Qi ;
      end
   end
endmodule
```

TEST BENCH

```
`timescale 1ns / 1ps

module JKFlipFlopTestBench13;
reg Ji Ki, clk ;
wire Qi, Q_not1;
VerilogJK_LATCH61 jk(Ji, Ki, clk, Qi, Q_not1);
always #5 clk = ~clk ;
initial
   begin
      Ji = 0 ;
      Ki = 0 ;
      clk = 0 ;
      repeat(8) begin
         #10 Ji = ~Ji ;
         #10 Ki = ~Ki ;
      end
   $stop;
```

```
end
endmodule
```

T FLIP FLOP

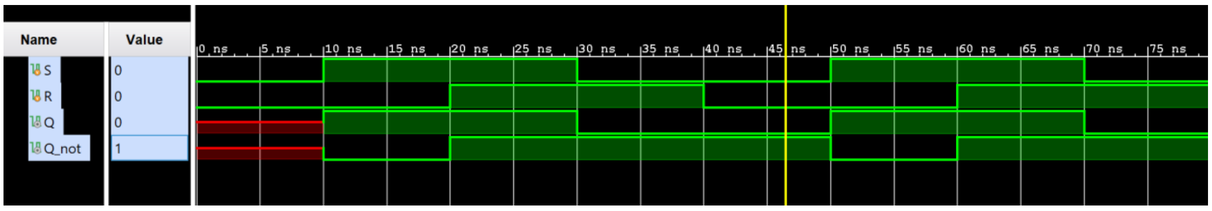
```
`timescale 1ns / 1ps

module VerilogT_LATCH61 (
   input Tn,
   input clk,
   output reg Q1,
   output reg Q_not2
   );
   initial begin
      Q1 = 0 ;
      Q_not2 = 1 ;
   end
   always @(posedge clk) begin
      if(Tn == 1) begin
         Q1 = ~Q1 ;
         Q_not2 = ~Q_not2 ;
      end
   end
endmodule
```

TEST BENCH

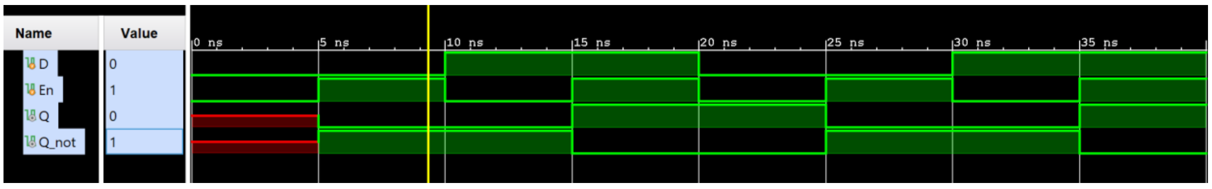
```
`timescale 1ns / 1ps

module TFlipFlopTestBench14;
reg Tn, clk ;
wire Q1, Q_not2 ;
VerilogT_LATCH61 T1(.Tn(Tn), .clk(clk), .Q1(Q1), .Q_not2(Q_not2));
always #5 clk = ~clk ;
initial begin
   Tn = 0 ;
   clk = 0 ;
   repeat(4) begin
      #10 ;
      Tn = ~Tn ;
   end
end
endmodule
```

WAVE WINDOW OUTPUT

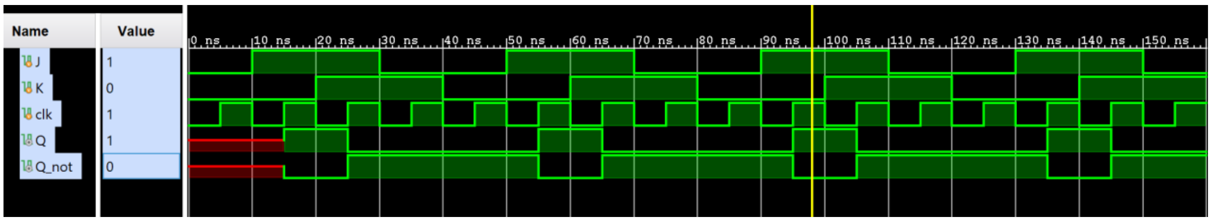
SR LATCH



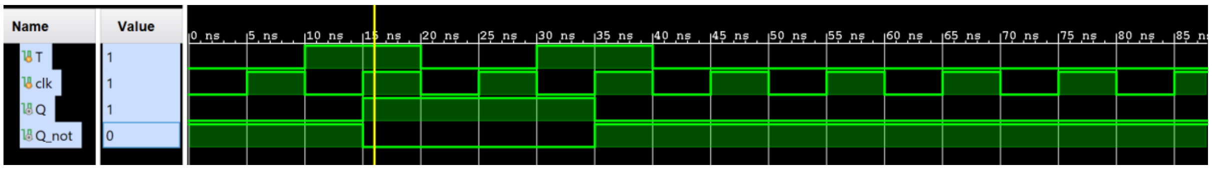
D LATCH



JK FLIP FLOP

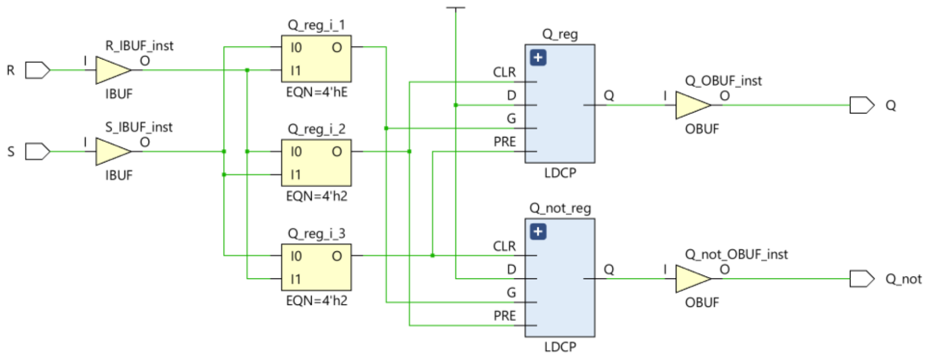


T FLIP FLOP

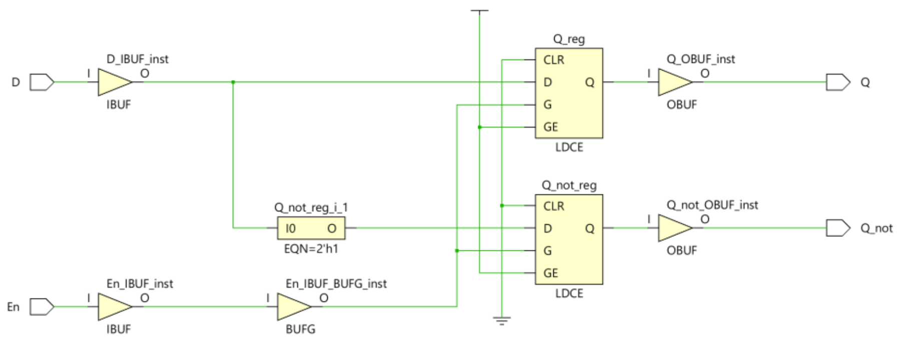


SCHEMATIC

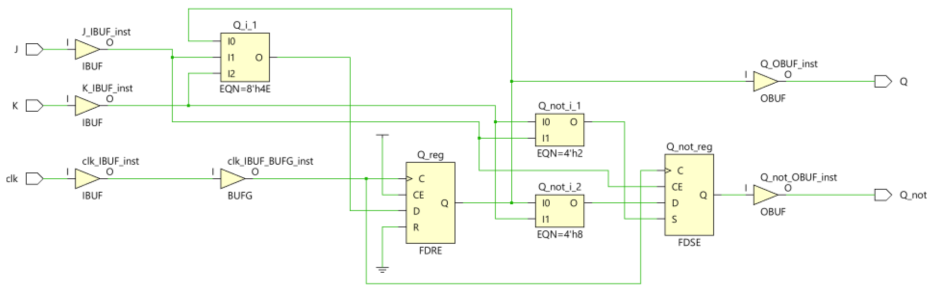
SR LATCH



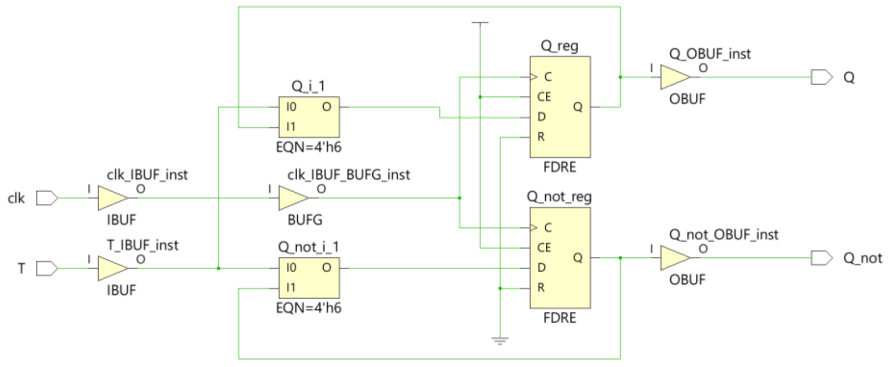
D LATCH



JK FLIP FLOP



T FLIP FLOP



CONCLUSION

Hence in this experiment, we successfully implemented Latches and Flip Flops using Behavioral Modelling in Verilog.

EXPERIMENT 07

AIM

In this experiment, our aim is to implement Shift Registers (SISO, SIPO, PISO & PIPO) in Verilog.

SOFTWARE USED

Vivado 2018.3

VERILOG CODE

SISO REGISTER

```
`timescale 1ns / 1ps

module VerilogSISO_61 (
   input In,
   input clk,
   input clear,
   output Out
   );
   reg [3:0] temp ;
   always @ (posedge clk)
   begin
      if(clear)
         temp <= 4'b0000;
      else
         temp <= {temp[2:0], In};
   end
   assign Out = temp[3];
endmodule
```

TEST BENCH

```
`timescale 1ns / 1ps

module SISO6TestBench;
reg In, clk, clear;
wire Out ;
VerilogSISO_61 s4(.In(In), .clk(clk), .clear(clear), .Out(Out));
always #10 clk = ~clk ;
initial
   begin
      clk = 0;
      In = 1;
      clear = 0;
      repeat(32) begin
         In = ~In ;
         #20;
      end
   $stop;
end
endmodule
```

SIPO SHIFT REGISTER

```
`timescale 1ns / 1ps

module VerilogSIPO_61 (
   input In,
   input clk,
   input clear,
   output [3:0] Out
   );
   reg [3:0] temp ;
   always @ (posedge clk)
   begin
      if(clear)
         temp <= 4'b0000;
      else
         temp <= {temp[2:0], In};
   end
   assign Out = temp;
endmodule
```

TEST BENCH

```
`timescale 1ns / 1ps

module SIPO4TestBench;
reg In, clk, clear;
wire[3:0] Out ;
VerilogSIPO_61 s4(.In(In), .clk(clk), .clear(clear), .Out(Out));
always #10 clk = ~clk ;
initial
   begin
      clk = 0;
      In = 1;
      clear = 0;
      repeat(32) begin
         In = ~In ;
         #20;
      end
   $stop;
end
endmodule
```

PISO SHIFT REGISTER (4-BIT)

```
`timescale 1ns / 1ps

module VerilogPISO_61 (
   input [3:0] In,
   input clk,
   input load,
   output reg Out
   );
   reg [3:0] temp ;
   always @(posedge clk) begin
      if(load)
         temp = In ;
      else
         temp = {temp[2:0], 1'b0} ;
      Out = temp[3] ;
   end
endmodule
```

TEST BENCH

```
`timescale 1ns / 1ps

module PISO4TestBench;
reg [3:0] In;
reg clk, load;
wire Out ;
VerilogPISO_61 p4(.In(In), .clk(clk), .load(load), .Out(Out));
always #10 clk = ~clk ;
initial
   begin
      clk = 0;
      In = 4'b0111; load = 1 ; #20;
      load = 0 ;  #80
      In = 4'b1010; load = 1 ; #20 ;
      load = 0 ; #80
      In = 4'b0101 ; load = 1 ; #20;
      load = 0 ; #80;
   $stop;
end
endmodule
```

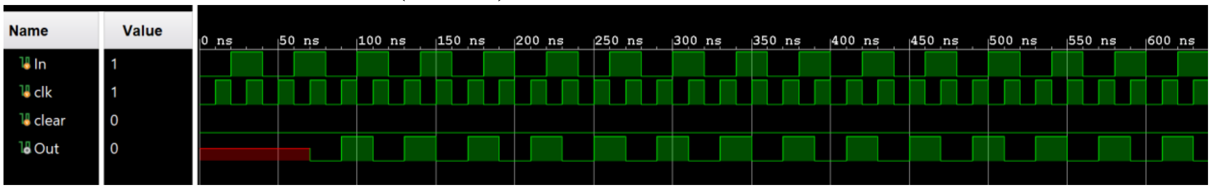
PIPO SHIFT REGISTER (4-BIT)

```
`timescale 1ns / 1ps

module VerilogPIPO_61 (
   input [3:0] In,
   input clk,
   input reset,
   output reg [3:0] Out
   );
   always @ (posedge clk or negedge reset)
   begin
      if (~reset)
         Out <= 4'b0000;
      else
         Out <= {Out[2:0], In};
   end
endmodule
```

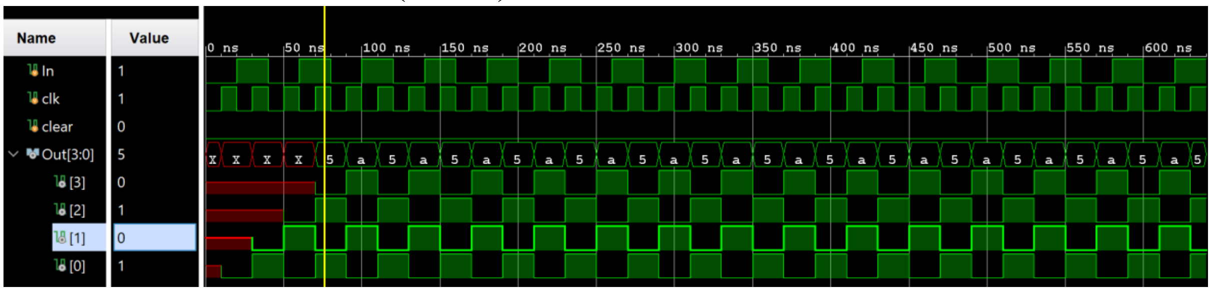
TEST BENCH

```
`timescale 1ns / 1ps

module PIPO4TestBench;
reg [3:0] In;
reg clk, reset;
wire[3:0] Out ;
VerilogPIPO_61  p4(.In(In), .clk(clk), .reset(reset), .Out(Out));
always #10 clk = ~clk ;
initial
   begin
      clk = 0;
      reset = 1 ;
      In = 4'b0101 ; #40 ;
      reset = 0; #10 ;
      reset = 1 ; In = 4'b1111; #40 ;
      In = 4'b1010; #40 ;
   $stop;
end
endmodule
```

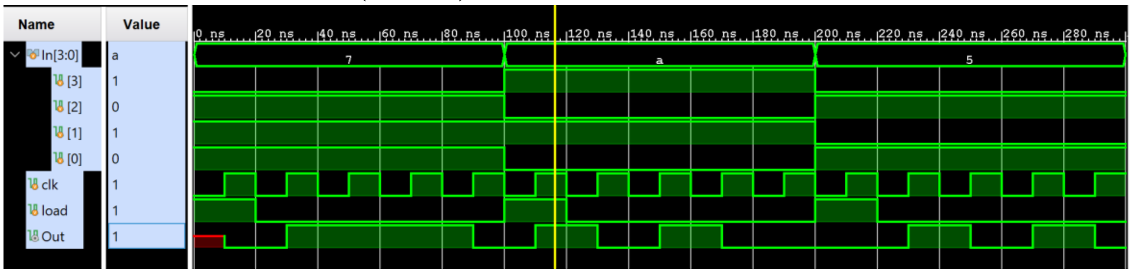
WAVE WINDOW OUTPUT
SISO SHIFT REGISTER (4-BIT)



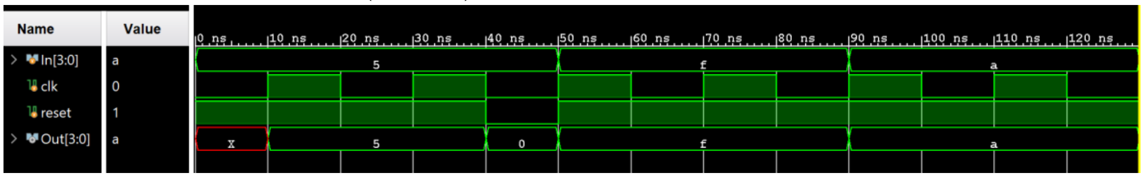
SIPO SHIFT REGISTER (4-BIT)



PISO SHIFT REGISTER (4-BIT)

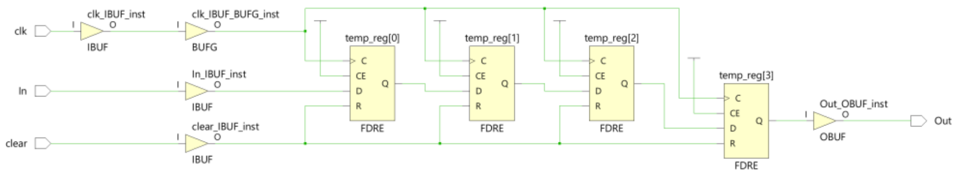


PIPO SHIFT REGISTER (4-BIT)

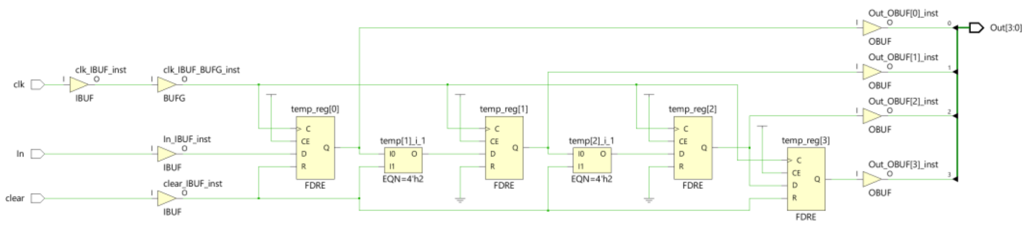


SCHEMATIC

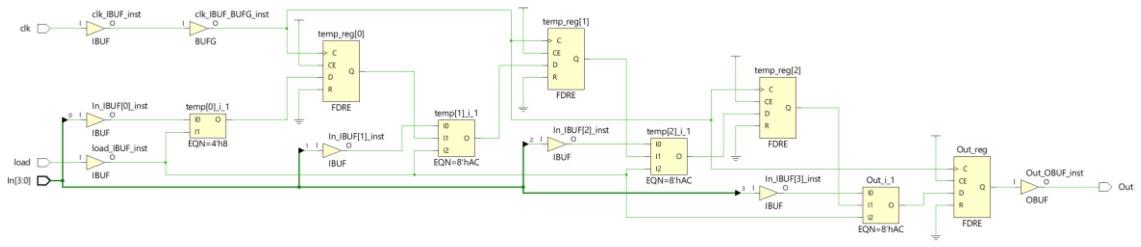
SISO SHIFT REGISTER (4-BIT)



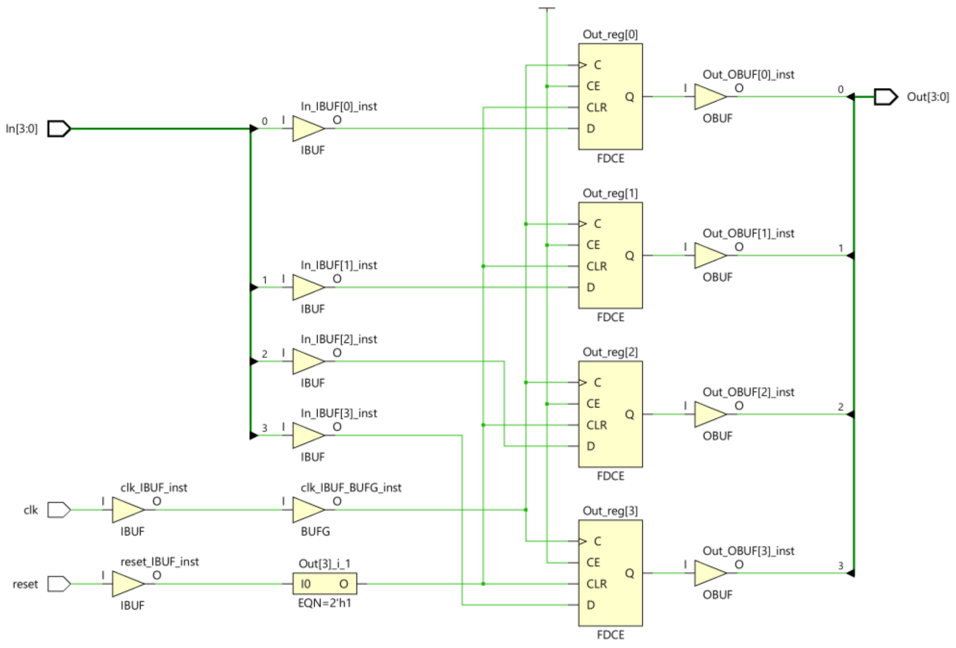
SIPO SHIFT REGISTER (4-BIT)



PISO SHIFT REGISTER (4-BIT)



PIPO SHIFT REGISTER (4-BIT)



CONCLUSION

Hence in this experiment, we successfully implemented 4-bit Shift Registers in Verilog.

EXPERIMENT 08

AIM

In this experiment, our aim is to implement a Universal Shift Register in Verilog.

SOFTWARE USED

Vivado 2018.3

VERILOG CODE

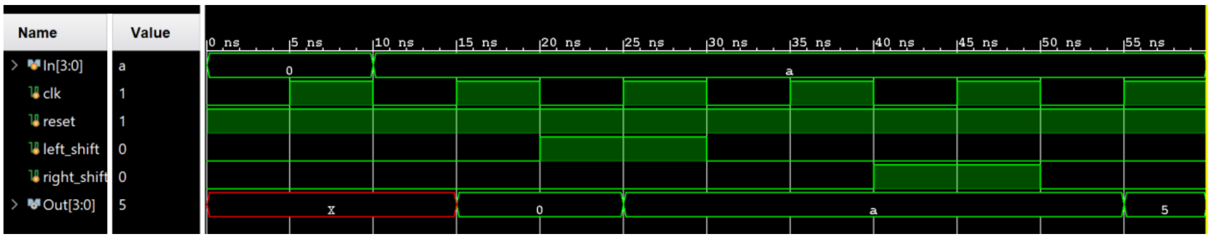
```
`timescale 1ns / 1ps

module SHIFT_REG61 (
   input [3:0] In,
   input clk,
   input reset,
   input left_shift,
   input right_shift,
   output reg [3:0] Out
   );
   reg [3:0] shift_reg;
   always @(posedge clk) begin
      if (~reset) begin
         shift_reg <= 4'b0;
      end else begin
         if (left_shift) begin
            //Left Shift
            shift_reg <= {shift_reg[2:0], In};
         end else if (right_shift) begin
            //Right Shift
            shift_reg <= {In, shift_reg[3:1]};
         end else begin
            //Parallel Loading
            shift_reg <= In;
         end
      end
      Out = shift_reg ;
   end
endmodule
```

TEST BENCH

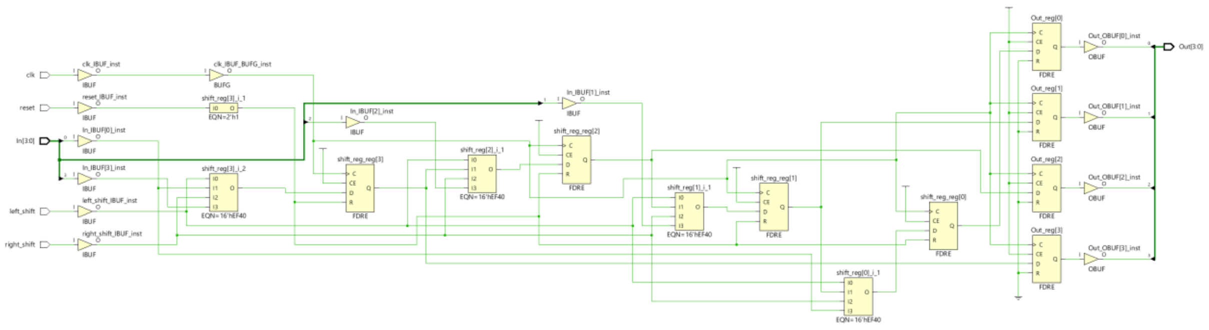
```
`timescale 1ns / 1ps

module UniversalShiftRegisterTestBench78;
   reg [3:0] In;
   reg clk;
```

```
    reg reset;
    reg left_shift;
    reg right_shift;
    wire [3:0] Out;
    SHIFT_REG61 usr(
      .In(In), .clk(clk), .reset(reset), .left_shift(left_shift),
      .right_shift(right_shift), .Out(Out));
    initial begin
      clk = 0;
      forever #5 clk = ~clk;
    end
    initial begin
      reset = 1;
      In = 4'b0;
      left_shift = 0;
      right_shift = 0;
      #10 In = 4'b1010;
      #10 left_shift = 1;
      #10 left_shift = 0;
      #10 right_shift = 1;
      #10 right_shift = 0;
      #10 $finish;
    end
endmodule
```

WAVE WINDOW OUTPUT



SCHEMATIC



CONCLUSION

Hence in this experiment, we successfully implemented a 4-bit Bi-directional Universal Shift Register in Verilog.

EXPERIMENT 09

AIM

In this experiment, our aim is to implement 4-bit Counters (Synchronous Up & Down) in Verilog.

SOFTWARE USED

Vivado 2018.3

VERILOG CODE

SYNCHRONOUS UP COUNTER (4-BIT)

```
`timescale 1ns / 1ps

module UP_COUNTER61 (
   input clk1,
   input reset1,
   output reg [3:0] Out1
   );
   always @(posedge clk1 or posedge reset1) begin
      if (reset1) begin
         Out1 <= 4'b0000;
      end else begin
         Out1 <= Out + 1;
      end
   end
endmodule
```

TEST BENCH

```
`timescale 1ns / 1ps

module UpCounterTestBench56;
   reg clk1, reset1;
   wire [3:0] Out1;
   UP_COUNTER61 up(.clk1(clk1), .reset1(reset1), .Out1(Out1));
   always #5 clk1 = ~clk1 ;
   initial begin
      clk1 = 0 ;
      reset1 = 1 ;
      #5 reset1 = 0 ;
   end
endmodule
```

SYNCHRONOUS DOWN COUNTER (4-BIT)

```
`timescale 1ns / 1ps
```

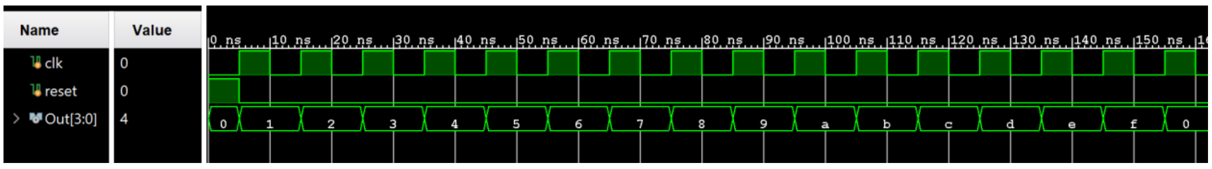
```
module DOWN_COUNTER61 (
   input clk1,
   input reset1,
   output reg [3:0] Out1
   );
   always @(posedge clk1 or posedge reset1) begin
      if (reset1) begin
         Out1 <= 4'b0000;
      end else begin
         Out1 <= Out1 - 1;
      end
   end
endmodule
```

TEST BENCH

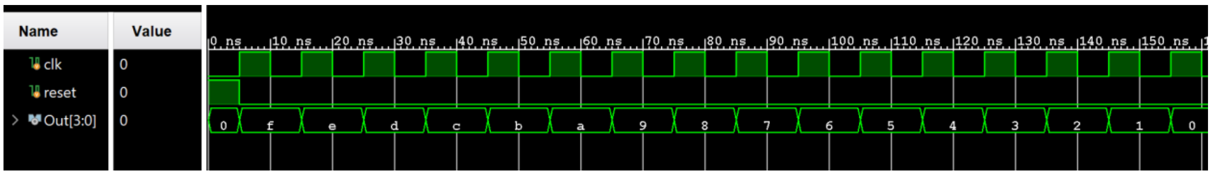
```
`timescale 1ns / 1ps

module DownCounterTestBench34;
   reg clk1, reset1;
   wire [3:0] Out1 ;
   DOWN_COUNTER61 down(.clk1(clk1), .reset1(reset1), .Out1(Out1));
   always #5 clk1 = ~clk1 ;
   initial begin
      clk1 = 0 ;
      reset1 = 1 ;
      #5 reset1 = 0 ;
   end
endmodule
```

WAVE WINDOW OUTPUT

SYNCHRONOUS UP COUNTER (4-BIT)

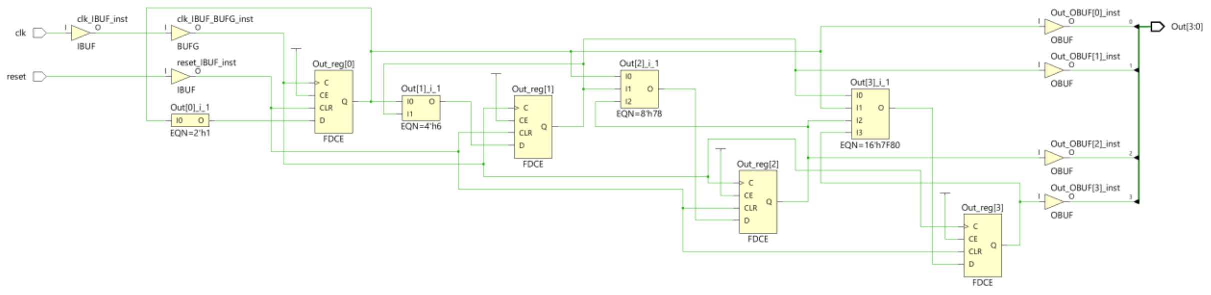

Name
Value
clk 0
reset 0
Out[3:0] 4
0 ns 10 ns 20 ns 30 ns 40 ns 50 ns 60 ns 70 ns 80 ns 90 ns 100 ns 110 ns 120 ns 130 ns 140 ns 150 ns
0 1 2 3 4 5 6 7 8 9 a b c d e f 0

SYNCHRONOUS DOWN COUNTER (4-BIT)

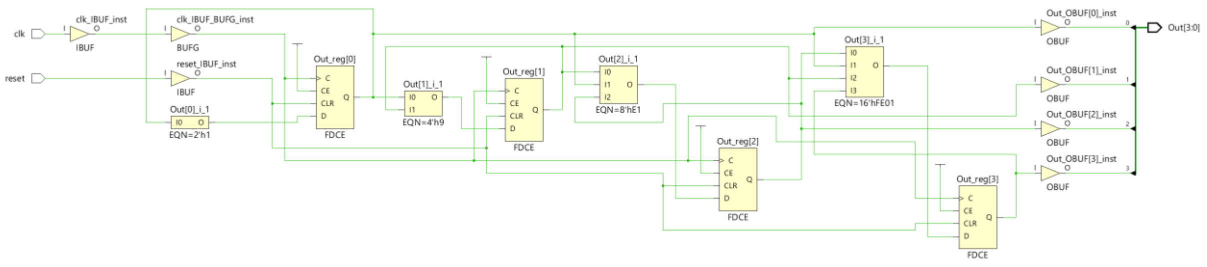

Name
Value
clk 0
reset 0
Out[3:0] 0
0 ns 10 ns 20 ns 30 ns 40 ns 50 ns 60 ns 70 ns 80 ns 90 ns 100 ns 110 ns 120 ns 130 ns 140 ns 150 ns
0 f e d c b a 9 8 7 6 5 4 3 2 1 0

SCHEMATIC

SYNCHRONOUS UP COUNTER (4-BIT)



SYNCHRONOUS DOWN COUNTER (4-BIT)



CONCLUSION

Hence in this experiment, we successfully implemented Synchronous Up/Down Counters in Verilog.

EXPERIMENT 10

AIM

In this experiment, our aim is to implement a PRBS (Pseudo Random Binary Sequence) Generator in Verilog.

SOFTWARE USED

Vivado 2018.3

VERILOG CODE

PRBS GENERATOR

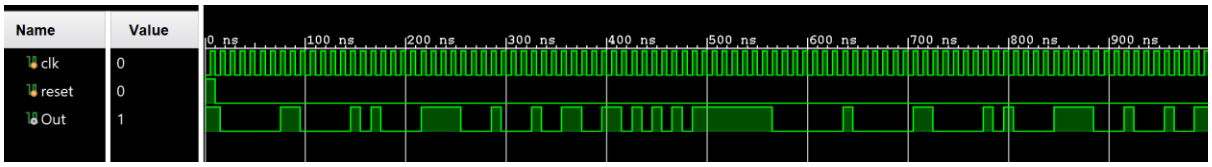
This generator uses an 8-bit feedback shift register and the feedback polynomial is x_7+x_6+1.

```
`timescale 1ns / 1ps

module PRBS_GEN_61 (
   input clk1,
   input reset1,
   output Out1
   );
   reg [7:0] shift_reg;
   wire feedback = shift_reg[7] ^ shift_reg[6];

   always @(posedge clk1 or posedge reset1) begin
      if (reset1) begin
         shift_reg <= 8'b00000001;
      end else begin
         shift_reg <= {shift_reg[6:0], feedback};
      end
   end
   assign Out1 = shift_reg[0];
endmodule
```

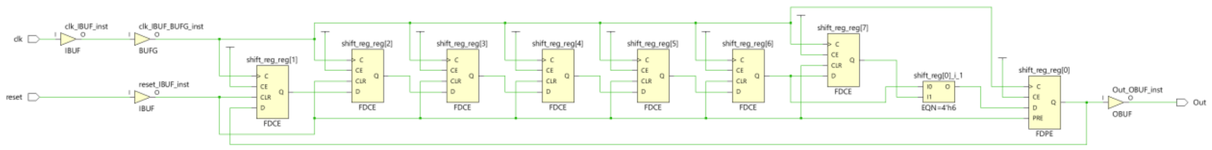
TEST BENCH

```
`timescale 1ns / 1ps

module PRBSTestBench123;
   reg clk, reset;
   wire Out ;
   PRBS_GEN_61 prbs(.clk1(clk1), .reset1(reset1), .Out1(Out1));
   initial begin
      clk1 = 0;
      forever #5 clk1 = ~clk1;
   end
   initial begin
      reset = 1;
      #10 reset = 0;
      #1000 $finish;
   end
endmodule
```

WAVE WINDOW OUTPUT



SCHEMATIC



CONCLUSION

Hence in this experiment, we successfully implemented a PRBS using a custom feedback polynomial and an 8-bit shift register in Verilog.