Verilog HDL

Assignment 4

By: Magdy Ahmed Abbas Abdelhamid

Question 1

Parameterized Shift Reg.

Verilog Code:

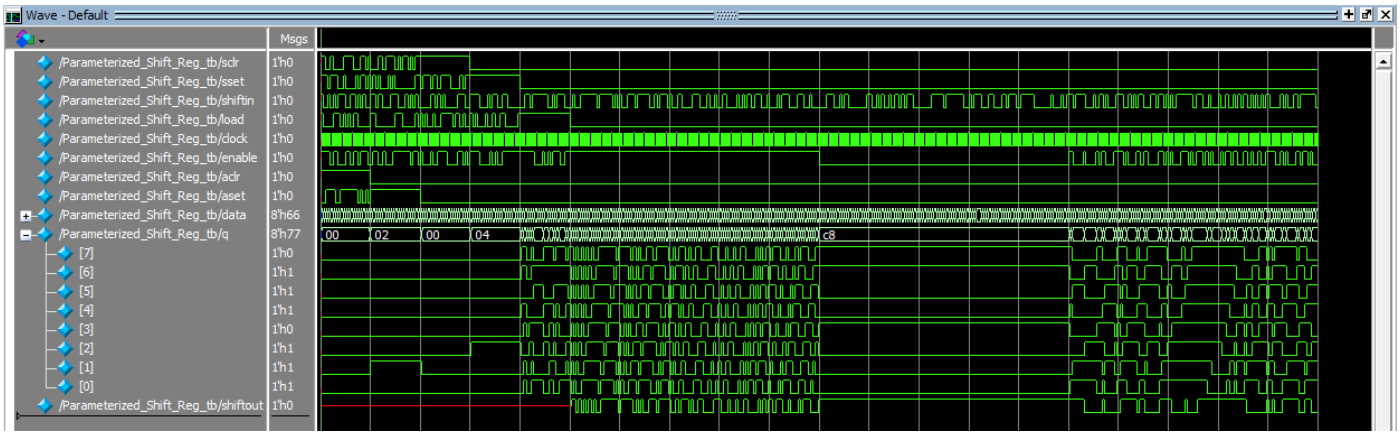
```
module Parameterized_Shift_Reg (sclr, sset, shiftin, load, data, clock, enable, aclr, aset, shiftout, q);

parameter SHIFT_DIRECTION = "LEFT";
parameter SHIFT_WIDTH = 8;
parameter LOAD_AVALUE = 2;
parameter LOAD_SVALUE = 4;
input sclr, sset, shiftin, load, clock, enable, aclr, aset;
input [SHIFT_WIDTH - 1 : 0] data;
output shiftout;
output [SHIFT_WIDTH - 1 : 0] q;
reg tmp_shiftout;
reg [SHIFT_WIDTH - 1 : 0] tmp_q;

always @(posedge clock or posedge aset or posedge aclr) begin
  if (aclr)
    tmp_q <= 0; // or {SHIFT_WIDTH{1'b0}};
  else if (aset)
    tmp_q <= LOAD_AVALUE;
  else if (sclr)
    tmp_q <= 0; // or {SHIFT_WIDTH{1'b0}};
  else if (sset)
    tmp_q <= LOAD_SVALUE;
  else begin
    if (enable) begin
      if (load)
        tmp_q <= data;
      else begin
        case (SHIFT_DIRECTION)
          "LEFT" : {tmp_shiftout, tmp_q} <= {tmp_q, shiftin};
          "RIGHT" : {tmp_q, tmp_shiftout} <= {shiftin, tmp_q};
        endcase
      end
    end
  end
end
// Output Assignment
assign q = tmp_q;
assign shiftout = tmp_shiftout;
endmodule
```

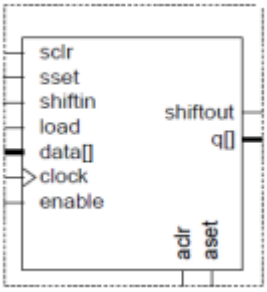
Testbench Code:

```
module Parameterized_Shift_Reg_tb ();
parameter SHIFT_DIRECTION = "LEFT";
parameter SHIFT_WIDTH = 8;
parameter LOAD_AVALUE = 2;
parameter LOAD_SVALUE = 4;
reg sclr, sset, shiftin, load, clock, enable, aclr, aset;
reg [SHIFT_WIDTH - 1 : 0] data;
wire [SHIFT_WIDTH - 1 : 0] q;

Parameterized_Shift_Reg #(SHIFT_DIRECTION, SHIFT_WIDTH, LOAD_AVALUE, LOAD_SVALUE)
DUT (sclr, sset, shiftin, load, data, clock, enable, aclr, aset, shiftout, q);

// Clock Generation
initial begin
  clock = 0;
  forever
    #25 clock = ~clock;
end

initial begin
  ////////////////////////
  // Test Async Signals //
  ////////////////////////
  aclr = 1; // aclr = 1
  repeat (20) begin
    aset = $random; sclr = $random; sset = $random; load = $random; enable = $random; data
              = $random; shiftin = $random;
    @(negedge clock);
    if (q != 0) begin
      $display("Async Control Signal - Q output is not correct with aclr");
      $stop;
    end
  end

  aclr = 0; aset = 1; // aclr = 0 & aset = 1
  repeat (20) begin
    sclr = $random; sset = $random; load = $random; enable = $random; data = $random; shiftin
              = $random;
    @(negedge clock);
    if (q != LOAD_AVALUE) begin
      $display("Async Control Signal - Q output is not correct with aset");
      $stop;
    end
  end
  ////////////////////////
  // Test Sync Signals //
  ////////////////////////
  aset = 0; sclr = 1; // sclr = 1 & aclr = 0 & aset = 0
```

```
  repeat (20) begin
    sset = $random; load = $random; enable = $random; data = $random; shiftin = $random;
    @(negedge clock);
    if (q != 0) begin
      $display("Sync Control Signal – Q output is not correct with sclr");
      $stop;
    end
  end
  sclr = 0; sset = 1; // sset = 1 & sclr = 0 & aclr = 0 & aset = 0
  repeat (20) begin
    load = $random; enable = $random; data = $random; shiftin = $random;
    @(negedge clock);
    if (q != LOAD_SVALUE) begin
      $display("Sync Control Signal – Q output is not correct with sset");
      $stop;
    end
  end
  ////////////////////
  // Test Load Signal //
  ////////////////////
  sset = 0; load = 1; // load = 1 & sset = 0 & sclr = 0 & aclr = 0 & aset = 0
  repeat (20) begin
    enable = $random; data = $random; shiftin = $random;
    @(negedge clock);
    if (q != data && enable == 1) begin
      $display("Load Control Signal – Q output is not correct with the load Signal");
      $stop;
    end
  end
  ////////////////////////////////
  // Test Shifting – Check Waveform //
  ////////////////////////////////  ////// High Enable – Shifting //////
  load = 0; enable = 1; // Enable = 1 & load = 0 & sset = 0 & sclr = 0 & aclr = 0 & aset = 0
  repeat (100) begin
    data = $random; shiftin = $random;
    @(negedge clock);
  End
  // Low Enable – Q doesn't Change //
  enable = 0; // Enable = 0 & load = 0 & sset = 0 & sclr = 0 & aclr = 0 & aset = 0
  repeat (100) begin
    data = $random; shiftin = $random;
    @(negedge clock);
  End
  // Randomized Enable
  repeat (100) begin
    enable = $random; data = $random; shiftin = $random;
    @(negedge clock);
  end
  $stop;
end
endmodule
```

Waveform:





Name	Value	Description
LOAD_AVALUE	Integer > 0	Value loaded with aset is high
SHIFT_DIRECTION	"LEFT" or "RIGHT"	Direction of the shift register. Default = "LEFT"
LOAD_SVALUE	Integer > 0	Value loaded with sset is high with the rising clock edge
SHIFT_WIDTH	Integer > 0	Width of data[] and q[] ports

Name	Type	Description
sclr	Input	Synchronous clear input. If both sclr and sset are asserted, sclr is dominant.
sset		Synchronous set input that sets q[] output with the value specified by LOAD_SVALUE. If both sclr and sset are asserted, sclr is dominant.
shiftin		Serial shift data input
load		Synchronous parallel load. High: Load operation with data[], Low: Shift operation
data[]		Data input to the shift register. This port is SHIFT_WIDTH wide
clock		Clock Input
enable		Clock enable input
aclr		Asynchronous clear input. If both aclr and aset are asserted, aclr is dominant.
aset		Asynchronous set input that sets q[] output with the value specified by LOAD_AVALUE. If both aclr and aset are asserted, aclr is dominant.
shiftout	Output	Serial Shift data output
q[]		Data output from the shift register. This port is SHIFT_WIDTH wide

Do File for Question 1

```
vlib work
vlog Parameterized_Shift_Reg. v Parameterized_Shift_Reg_tb. v
vsim  − voptargs = +acc work. Parameterized_Shift_Reg_tb
add wave  ∗
run  − all
```

Question 2

4 - Bit Ripple Counter

Verilog Code:

```
module Question2 (clk, set, out);

input clk, set;
output reg [3: 0] out;

always @(posedge clk or negedge set) begin
  if(~set)
    out <= 4'b1111;
  else
    out <= out + 1;
end
endmodule
```

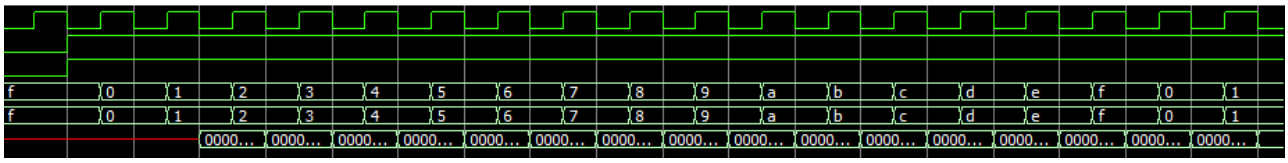
```
module D_FlipFlop (d, rstn, clk, q, qbar);

input d, rstn, clk;
output reg q; // output reg q, qbar;
output qbar;

always @(posedge clk or negedge rstn) begin
  if (~rstn)
    q <= 1'b0; // qbar <= 1'b1;
  else
    q <= d;  // qbar <= ~d;
end
assign qbar = ~q;
endmodule
```

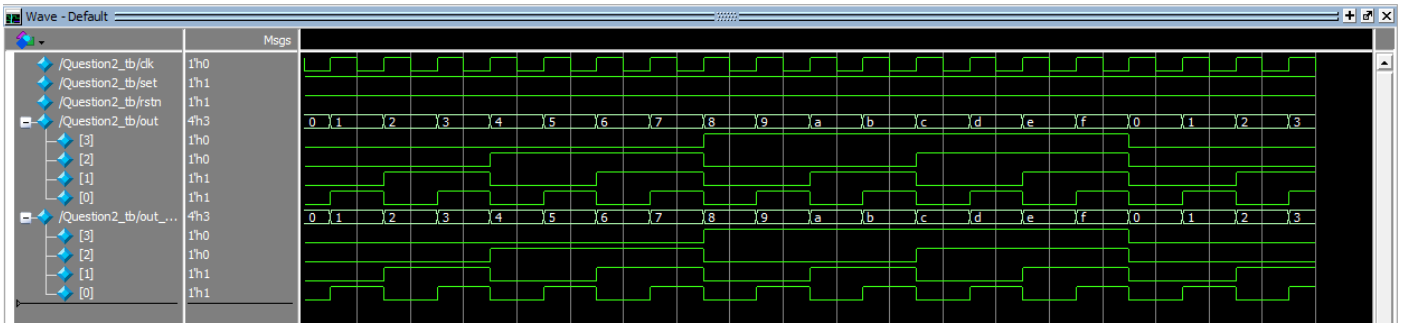
```
module RippleCounter4bits (CLK, RSTN, OUT);

parameter N = 4;
input CLK, RSTN;
output [N − 1: 0] OUT;
wire Q0, Q1, Q2, Q3;
wire Qn0, Qn1, Qn2, Qn3;
D_FlipFlop ff1 (. d(Qn0), . rstn(RSTN), . clk(CLK), . q(Q0), . qbar(Qn0));
D_FlipFlop ff2 (. d(Qn1), . rstn(RSTN), . clk(Q0), . q(Q1), . qbar(Qn1));
D_FlipFlop ff3 (. d(Qn2), . rstn(RSTN), . clk(Q1), . q(Q2), . qbar(Qn2));
D_FlipFlop ff4 (. d(Qn3), . rstn(RSTN), . clk(Q2), . q(Q3), . qbar(Qn3));
assign OUT = {Qn3, Qn2, Qn1, Qn0};

endmodule
```

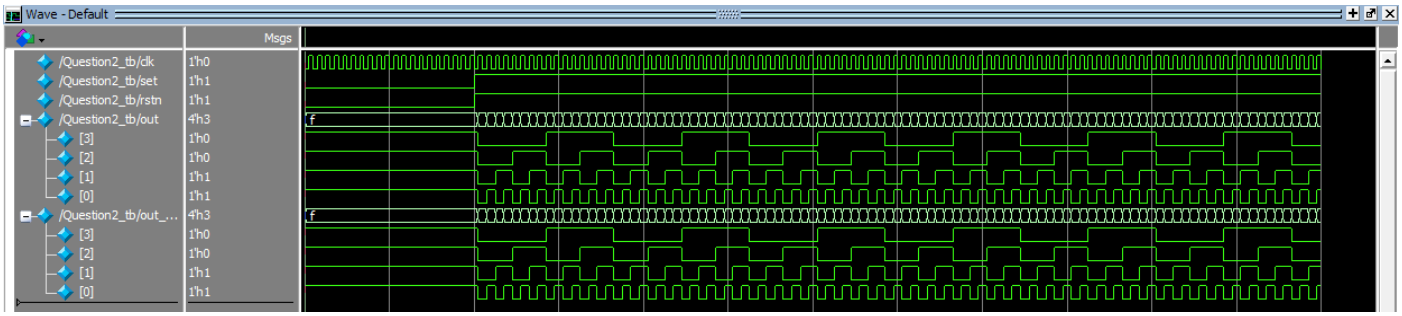
Testbench Code:

```
module Question2_tb ();

reg clk, set, rstn;
wire [3:0] out, out_ref;

Question2 DUT (clk, set, out);
RippleCounter4bits DUT_Ref (clk, rstn, out_ref);

initial begin
   clk = 0;
   forever
      #25 clk = ~clk;
end

initial begin
   set = 0; rstn = 0;
   repeat (20) begin
      @(negedge clk);
      if (out != out_ref) begin
         $display("ASync Control Signal – Q output is not correct with set");
         $stop;
      end
   end

   set = 1; rstn = 1;
   repeat (100) begin
      @(negedge clk);
      if (out != out_ref) begin
         $display("The 4 Bit Ripple Counter Design is Wrong");
         $stop;
      end
   end
   $stop;
end

initial
$monitor("time = %0t, CLK = %b, Set = %b, Out = %b", $time, clk, set, out);

endmodule
```

Waveform:







Do File for Question 2

```
vlib work
vlog Question2. v Question2_tb. v
vsim  − voptargs = +acc work. Question2_tb
add wave  ∗
run  − all
```

```
# time = 2000, CLK = 0, Set = 1, Out = 0011
# time = 2025, CLK = 1, Set = 1, Out = 0100
# time = 2050, CLK = 0, Set = 1, Out = 0100
# time = 2075, CLK = 1, Set = 1, Out = 0101
# time = 2100, CLK = 0, Set = 1, Out = 0101
# time = 2125, CLK = 1, Set = 1, Out = 0110
# time = 2150, CLK = 0, Set = 1, Out = 0110
# time = 2175, CLK = 1, Set = 1, Out = 0111
# time = 2200, CLK = 0, Set = 1, Out = 0111
# time = 2225, CLK = 1, Set = 1, Out = 1000
# time = 2250, CLK = 0, Set = 1, Out = 1000
# time = 2275, CLK = 1, Set = 1, Out = 1001
# time = 2300, CLK = 0, Set = 1, Out = 1001
# time = 2325, CLK = 1, Set = 1, Out = 1010
# time = 2350, CLK = 0, Set = 1, Out = 1010
# time = 2375, CLK = 1, Set = 1, Out = 1011
# time = 2400, CLK = 0, Set = 1, Out = 1011
# time = 2425, CLK = 1, Set = 1, Out = 1100
# time = 2450, CLK = 0, Set = 1, Out = 1100
# time = 2475, CLK = 1, Set = 1, Out = 1101
# time = 2500, CLK = 0, Set = 1, Out = 1101
# time = 2525, CLK = 1, Set = 1, Out = 1110
# time = 2550, CLK = 0, Set = 1, Out = 1110
# time = 2575, CLK = 1, Set = 1, Out = 1111
# time = 2600, CLK = 0, Set = 1, Out = 1111
# time = 2625, CLK = 1, Set = 1, Out = 0000
# time = 2650, CLK = 0, Set = 1, Out = 0000
# time = 2675, CLK = 1, Set = 1, Out = 0001
# time = 2700, CLK = 0, Set = 1, Out = 0001
# time = 2725, CLK = 1, Set = 1, Out = 0010
# time = 2750, CLK = 0, Set = 1, Out = 0010
# time = 2775, CLK = 1, Set = 1, Out = 0011
# time = 2800, CLK = 0, Set = 1, Out = 0011
```

Question 3

Counter Dividers

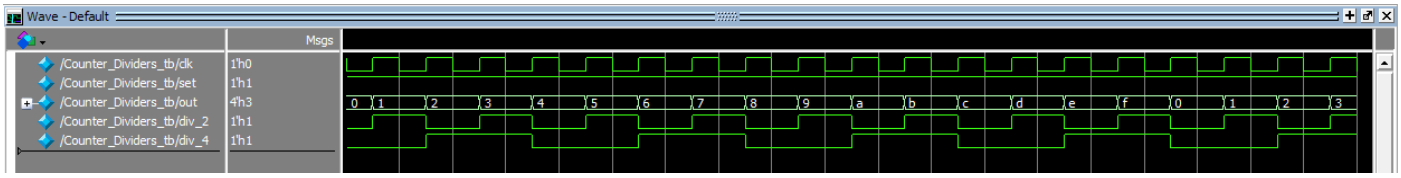
Verilog Code:

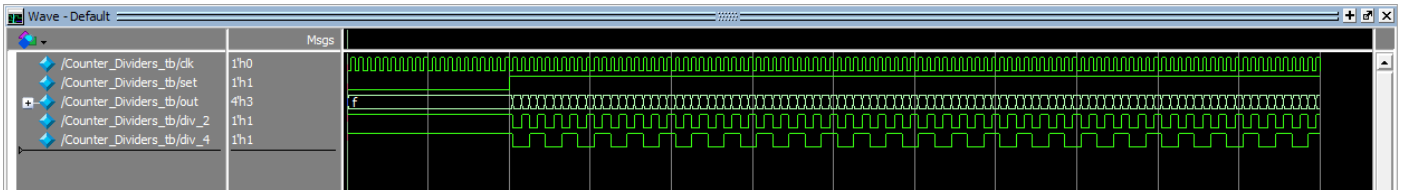
```
module Counter_Dividers (clk, set, out, div_2, div_4);

input clk, set;
output div_2, div_4;
output reg [3: 0] out;

always @(posedge clk or negedge set) begin
   if(~set)
      out <= 4'b1111;
   else
      out <= out + 1;
end

assign div_2 = out[0];
assign div_4 = out[1];

endmodule
```

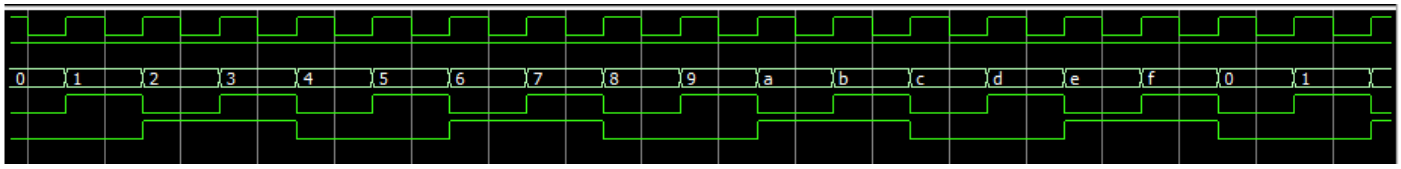
Testbench Code:

```
module Counter_Dividers_tb ();

reg clk, set;
wire [3:0] out;
wire div_2, div_4;

Counter_Dividers DUT (clk, set, out, div_2, div_4);

initial begin
   clk = 0;
   forever
      #25 clk = ~clk;
end

initial begin
   set = 0;
   repeat (20) begin
      @(negedge clk);
      if (out != 4'b1111) begin
         $display("ASync Control Signal - Q output is not correct with set");
         $stop;
      end
   end

   set = 1;
   repeat (100) @(negedge clk);

   $stop;
end

initial
$monitor("time = %0t, CLK = %b, Set = %b, Out = %b, Div2 = %b,
                                          Div4 = %b", $time, clk, set, out, div_2, div_4);

endmodule
```

Waveform:








```
# time = 700, CLK = 0, Set = 0, Out = 1111, Div2 = 1, Div4 = 1
# time = 725, CLK = 1, Set = 0, Out = 1111, Div2 = 1, Div4 = 1
# time = 750, CLK = 0, Set = 0, Out = 1111, Div2 = 1, Div4 = 1
# time = 775, CLK = 1, Set = 0, Out = 1111, Div2 = 1, Div4 = 1
# time = 800, CLK = 0, Set = 0, Out = 1111, Div2 = 1, Div4 = 1
# time = 825, CLK = 1, Set = 0, Out = 1111, Div2 = 1, Div4 = 1
# time = 850, CLK = 0, Set = 0, Out = 1111, Div2 = 1, Div4 = 1
# time = 875, CLK = 1, Set = 0, Out = 1111, Div2 = 1, Div4 = 1
# time = 900, CLK = 0, Set = 0, Out = 1111, Div2 = 1, Div4 = 1
# time = 925, CLK = 1, Set = 0, Out = 1111, Div2 = 1, Div4 = 1
# time = 950, CLK = 0, Set = 0, Out = 1111, Div2 = 1, Div4 = 1
# time = 975, CLK = 1, Set = 0, Out = 1111, Div2 = 1, Div4 = 1
# time = 1000, CLK = 0, Set = 1, Out = 1111, Div2 = 1, Div4 = 1
# time = 1025, CLK = 1, Set = 1, Out = 0000, Div2 = 0, Div4 = 0
# time = 1050, CLK = 0, Set = 1, Out = 0000, Div2 = 0, Div4 = 0
# time = 1075, CLK = 1, Set = 1, Out = 0001, Div2 = 1, Div4 = 0
# time = 1100, CLK = 0, Set = 1, Out = 0001, Div2 = 1, Div4 = 0
# time = 1125, CLK = 1, Set = 1, Out = 0010, Div2 = 0, Div4 = 1
# time = 1150, CLK = 0, Set = 1, Out = 0010, Div2 = 0, Div4 = 1
# time = 1175, CLK = 1, Set = 1, Out = 0011, Div2 = 1, Div4 = 1
# time = 1200, CLK = 0, Set = 1, Out = 0011, Div2 = 1, Div4 = 1
# time = 1225, CLK = 1, Set = 1, Out = 0100, Div2 = 0, Div4 = 0
# time = 1250, CLK = 0, Set = 1, Out = 0100, Div2 = 0, Div4 = 0
# time = 1275, CLK = 1, Set = 1, Out = 0101, Div2 = 1, Div4 = 0
# time = 1300, CLK = 0, Set = 1, Out = 0101, Div2 = 1, Div4 = 0
# time = 1325, CLK = 1, Set = 1, Out = 0110, Div2 = 0, Div4 = 1
# time = 1350, CLK = 0, Set = 1, Out = 0110, Div2 = 0, Div4 = 1
# time = 1375, CLK = 1, Set = 1, Out = 0111, Div2 = 1, Div4 = 1
# time = 1400, CLK = 0, Set = 1, Out = 0111, Div2 = 1, Div4 = 1
# time = 1425, CLK = 1, Set = 1, Out = 1000, Div2 = 0, Div4 = 0
# time = 1450, CLK = 0, Set = 1, Out = 1000, Div2 = 0, Div4 = 0
# time = 1475, CLK = 1, Set = 1, Out = 1001, Div2 = 1, Div4 = 0
# time = 1500, CLK = 0, Set = 1, Out = 1001, Div2 = 1, Div4 = 0
```

Question 4

LFSR

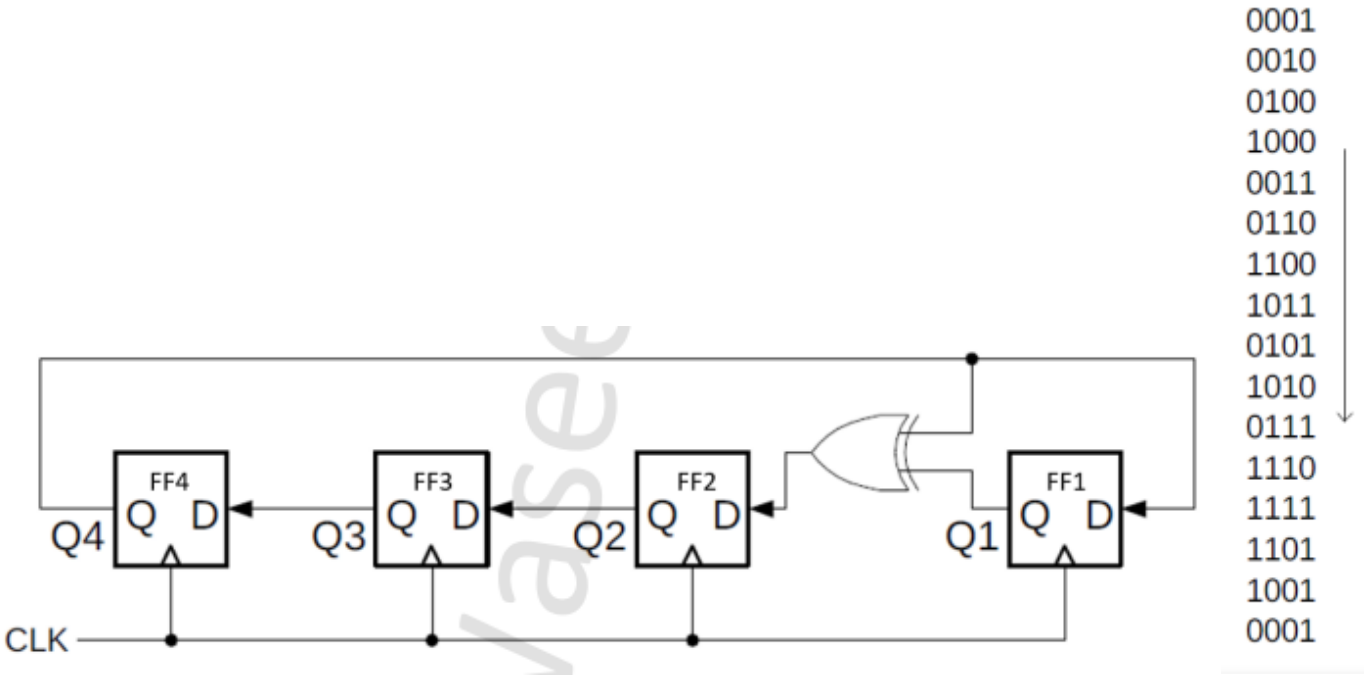
Verilog Code:

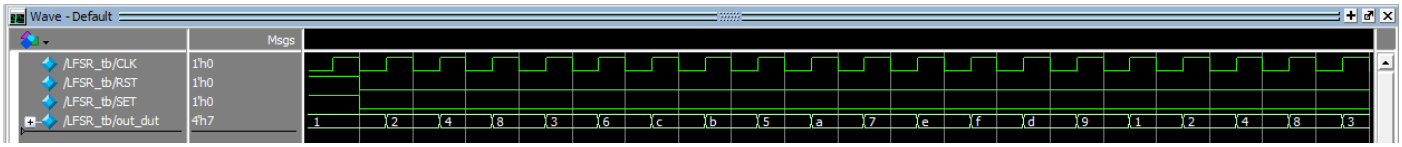
```
/*
module LFSR (clk, rst, set, out);
input clk, rst, set;
output [3: 0] out;
reg [3: 0] tmp_out;
always @(posedge clk or posedge rst) begin
   if (rst)
      tmp_out[3: 1]  <=  3'b000;
   else begin
      tmp_out[3]  <=  tmp_out[2];
      tmp_out[2]  <=  tmp_out[1];
      tmp_out[1]  <=  tmp_out[3] ^ tmp_out[0];
   end
end
always @(posedge clk or posedge set) begin
   if (set)
      tmp_out[0]  <=  1'b1;
   else
      tmp_out[0]  <=  tmp_out[3];
end
assign out  =  tmp_out;
endmodule
*/
module LFSR (clk, rst, set, out);
input clk, rst, set;
output [3: 0] out;
reg Q1, Q2, Q3, Q4;
always @(posedge clk or posedge rst) begin
   if (rst) begin
      Q2  <=  0; Q3  <=  0; Q4  <=  0;
   end else begin
      Q2  <=  Q1 ^ Q4; Q3  <=  Q2; Q4  <=  Q3;
   end
end
always @(posedge clk or posedge set) begin
   if (set) begin
      Q1  <=  1;
   end else begin
      Q1  <=  Q4;
   end
end
assign out  =  {Q4, Q3, Q2, Q1};
endmodule
```

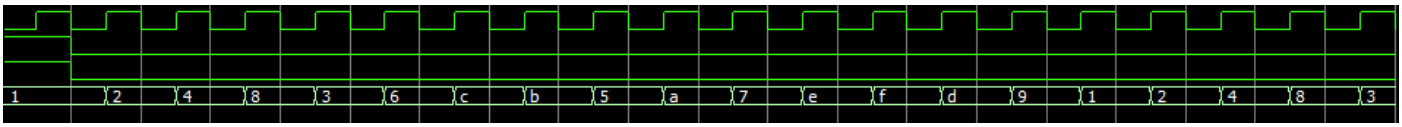
Testbench Code:

```
module LFSR_tb ();

reg CLK, RST, SET;
wire [3: 0] out_dut;

LFSR DUT (CLK, RST, SET, out_dut);

initial begin
   CLK  =  0;
   forever
      #25 CLK  =  ~CLK;
end

initial begin
   RST  =  1; SET  =  1;
   repeat(10) begin
      @(negedge CLK);
      if (out_dut ! =  4'b0001) begin
         $display("The LFSR Design is Wrong! ");
         $stop;
      end
   end

   RST  =  0; SET  =  0;
   repeat(100) @(negedge CLK);
   $stop;

end

initial
   $monitor ("OUT  =  %b", out_dut);

endmodule
```

Waveform:







```
# OUT = 0001
# OUT = 0010
# OUT = 0100
# OUT = 1000
# OUT = 0011
# OUT = 0110
# OUT = 1100
# OUT = 1011
# OUT = 0101
# OUT = 1010
# OUT = 0111
# OUT = 1110
# OUT = 1111
# OUT = 1101
# OUT = 1001
# OUT = 0001
# OUT = 0010
# OUT = 0100
# OUT = 1000
# OUT = 0011
# OUT = 0110
# OUT = 1100
# OUT = 1011
# OUT = 0101
# OUT = 1010
# OUT = 0111
# OUT = 1110
# OUT = 1111
# OUT = 1101
# OUT = 1001
```

Question 5

Parameterized FA / HA

Verilog Code:

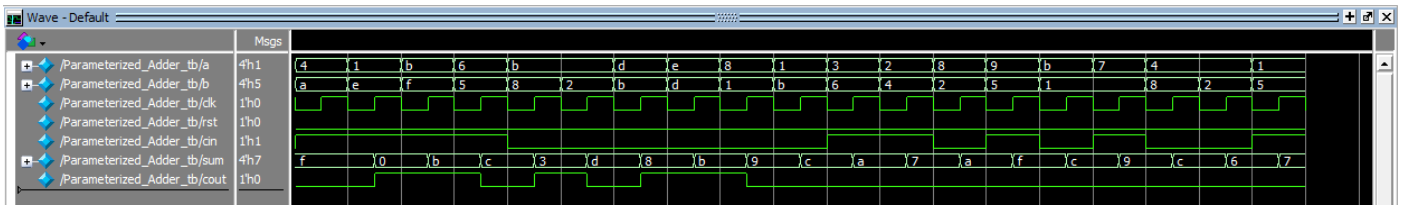
```
module Parameterized_Adder (a, b, clk, cin, rst, sum, cout);

parameter WIDTH = 4;
parameter PIPELINE_ENABLE = 1;
parameter USE_FULL_ADDER = 1;

input clk, rst, cin;
input [WIDTH - 1 : 0] a, b;
output reg [WIDTH - 1 : 0] sum;
output reg cout;
wire HA_Cout_Comb, FA_Cout_Comb;
wire [WIDTH - 1 : 0] HA_Sum_Comb, FA_Sum_Comb;
reg HA_Cout_Seq, FA_Cout_Seq;
reg [WIDTH - 1 : 0] HA_Sum_Seq, FA_Sum_Seq;

// Adder Dataflow Modeling
assign {HA_Cout_Comb, HA_Sum_Comb} = a + b;
assign {FA_Cout_Comb, FA_Sum_Comb} = a + b + cin;
// Adder Behavioral Pipelined Modeling
always @(posedge clk) begin
  if (rst) begin
    HA_Cout_Seq <= 0;
    HA_Sum_Seq <= 0;
    FA_Cout_Seq <= 0;
    FA_Sum_Seq <= 0;
  end begin
    {HA_Cout_Seq, HA_Sum_Seq} = a + b;
    {FA_Cout_Seq, FA_Sum_Seq} = a + b + cin;
  end
end
```

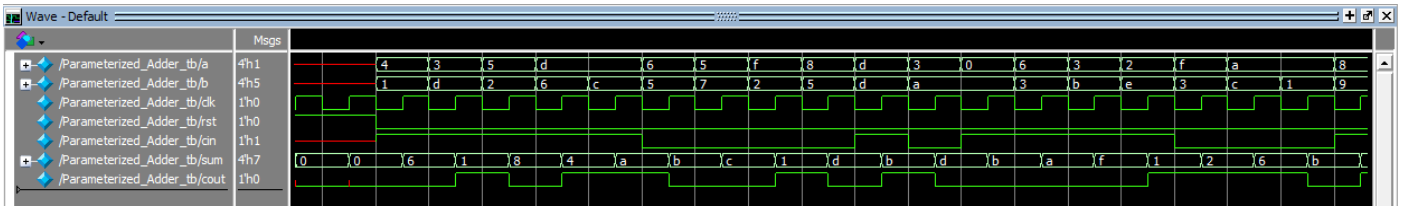
```
always @(*) begin
  if (PIPELINE_ENABLE == 1) begin
    if (USE_FULL_ADDER == 1) begin
      sum = FA_Sum_Seq;
      cout = FA_Cout_Seq;
    end
    else begin
      sum = HA_Sum_Seq;
      cout = HA_Cout_Seq;
    end
  end
  else begin
    if (USE_FULL_ADDER == 1) begin
      sum = FA_Sum_Comb;
      cout = FA_Cout_Comb;
    end
    else begin
      sum = HA_Sum_Comb;
      cout = HA_Cout_Comb;
    end
  end
end

endmodule // N - Bit Parameterized Full / Half Adder
```

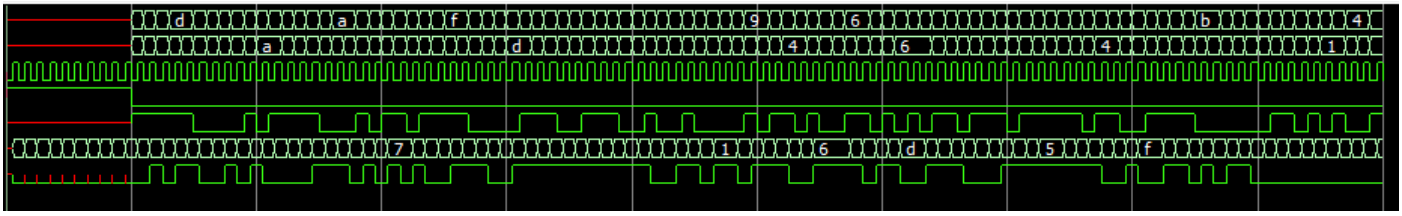
Testbench Code:

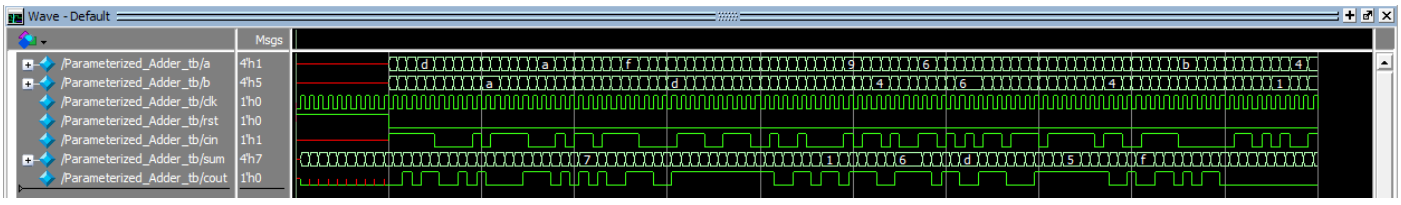
```
module Parameterized_Adder_tb();

parameter W1 = 4;
parameter W2 = 1;
parameter W3 = 1;
reg [W1 − 1: 0] a, b;
reg clk, rst, cin;
wire [W1 − 1: 0] sum;
wire cout;

Parameterized_Adder #(.WIDTH(W1), .PIPELINE_ENABLE(W2), .USE_FULL_ADDER(W3))
DUT(.a(a), .b(b), .cin(cin), .clk(clk), .rst(rst), .sum(sum), .cout(cout));

initial begin
clk = 0;
forever
#25 clk = ~clk;
end

initial begin
rst = 1;
repeat (10) @(negedge clk);

rst = 0;
repeat (100) begin
a = $random;
b = $random;
cin = $random;
@(negedge clk);
end
$stop;
end

initial
$monitor("time = %0t, a = %b, b = %b, Cin = %b, Sum = %b, Cout = %b",
                                                    $time, a, b, cin, sum, cout);

endmodule
```

Waveform:


Wave - Default
Msgs
/Parameterized_Adder_tb/a
/Parameterized_Adder_tb/b
/Parameterized_Adder_tb/clk
/Parameterized_Adder_tb/rst
/Parameterized_Adder_tb/cin
/Parameterized_Adder_tb/sum
/Parameterized_Adder_tb/cout


Wave - Default
Msgs
/Parameterized_Adder_tb/a
/Parameterized_Adder_tb/b
/Parameterized_Adder_tb/clk
/Parameterized_Adder_tb/rst
/Parameterized_Adder_tb/cin
/Parameterized_Adder_tb/sum
/Parameterized_Adder_tb/cout




Wave - Default
Msgs
/Parameterized_Adder_tb/a
/Parameterized_Adder_tb/b
/Parameterized_Adder_tb/clk
/Parameterized_Adder_tb/rst
/Parameterized_Adder_tb/cin
/Parameterized_Adder_tb/sum
/Parameterized_Adder_tb/cout

```
# time = 0, a = xxxx, b = xxxx, Cin = x, Sum = xxxx, Cout = x
# time = 25, a = xxxx, b = xxxx, Cin = x, Sum = 0000, Cout = 0
# time = 75, a = xxxx, b = xxxx, Cin = x, Sum = 0000, Cout = 0
# time = 125, a = xxxx, b = xxxx, Cin = x, Sum = 0000, Cout = 0
# time = 175, a = xxxx, b = xxxx, Cin = x, Sum = 0000, Cout = 0
# time = 225, a = xxxx, b = xxxx, Cin = x, Sum = 0000, Cout = 0
# time = 275, a = xxxx, b = xxxx, Cin = x, Sum = 0000, Cout = 0
# time = 325, a = xxxx, b = xxxx, Cin = x, Sum = 0000, Cout = 0
# time = 375, a = xxxx, b = xxxx, Cin = x, Sum = 0000, Cout = 0
# time = 425, a = xxxx, b = xxxx, Cin = x, Sum = 0000, Cout = 0
# time = 475, a = xxxx, b = xxxx, Cin = x, Sum = 0000, Cout = 0
# time = 500, a = 0100, b = 0001, Cin = 1, Sum = 0000, Cout = 0
# time = 525, a = 0100, b = 0001, Cin = 1, Sum = 0110, Cout = 0
# time = 550, a = 0011, b = 1101, Cin = 1, Sum = 0110, Cout = 0
# time = 575, a = 0011, b = 1101, Cin = 1, Sum = 0001, Cout = 1
# time = 600, a = 0101, b = 0010, Cin = 1, Sum = 0001, Cout = 1
# time = 625, a = 0101, b = 0010, Cin = 1, Sum = 1000, Cout = 0
# time = 650, a = 1101, b = 0110, Cin = 1, Sum = 1000, Cout = 0
# time = 675, a = 1101, b = 0110, Cin = 1, Sum = 0100, Cout = 1
# time = 700, a = 1101, b = 1100, Cin = 1, Sum = 0100, Cout = 1
# time = 725, a = 1101, b = 1100, Cin = 1, Sum = 1010, Cout = 1
# time = 750, a = 0110, b = 0101, Cin = 0, Sum = 1010, Cout = 1
# time = 775, a = 0110, b = 0101, Cin = 0, Sum = 1011, Cout = 0
# time = 800, a = 0101, b = 0111, Cin = 0, Sum = 1011, Cout = 0
# time = 825, a = 0101, b = 0111, Cin = 0, Sum = 1100, Cout = 0
# time = 850, a = 1111, b = 0010, Cin = 0, Sum = 1100, Cout = 0
# time = 875, a = 1111, b = 0010, Cin = 0, Sum = 0001, Cout = 1
# time = 900, a = 1000, b = 0101, Cin = 0, Sum = 0001, Cout = 1
# time = 925, a = 1000, b = 0101, Cin = 0, Sum = 1101, Cout = 0
# time = 950, a = 1101, b = 1101, Cin = 1, Sum = 1101, Cout = 0
# time = 975, a = 1101, b = 1101, Cin = 1, Sum = 1011, Cout = 1
# time = 1000, a = 0011, b = 1010, Cin = 0, Sum = 1011, Cout = 1
# time = 1025, a = 0011, b = 1010, Cin = 0, Sum = 1101, Cout = 0
# time = 1050, a = 0000, b = 1010, Cin = 1, Sum = 1101, Cout = 0
# time = 1075, a = 0000, b = 1010, Cin = 1, Sum = 1011, Cout = 0
# time = 1100, a = 0110, b = 0011, Cin = 1, Sum = 1011, Cout = 0
# time = 1125, a = 0110, b = 0011, Cin = 1, Sum = 1010, Cout = 0
# time = 1150, a = 0011, b = 1011, Cin = 1, Sum = 1010, Cout = 0
# time = 1175, a = 0011, b = 1011, Cin = 1, Sum = 1111, Cout = 0
# time = 1200, a = 0010, b = 1110, Cin = 1, Sum = 1111, Cout = 0
# time = 1225, a = 0010, b = 1110, Cin = 1, Sum = 0001, Cout = 1
# time = 1250, a = 1111, b = 0011, Cin = 0, Sum = 0001, Cout = 1
# time = 1275, a = 1111, b = 0011, Cin = 0, Sum = 0010, Cout = 1
# time = 1300, a = 1010, b = 1100, Cin = 0, Sum = 0010, Cout = 1
```

Name	Type	Description
a	Input	Data input a of width determined by WIDTH parameter
b		Data input b of width determined by WIDTH parameter
clk		Clk input
cin		Carry in bit
rst		Active high synchronous reset
sum	Output	sum of a and b input of width determined by WIDTH parameter
cout		Carry out bit