|  |
| --- |
| **DIGITAL SYSTEM DESIGN LABORATORY** |
| **LAB 3** |

**IMPLEMENTATION OF SEQUENTIAL LOGIC CIRCUIT USING VERILOG IN FPGA KIT**

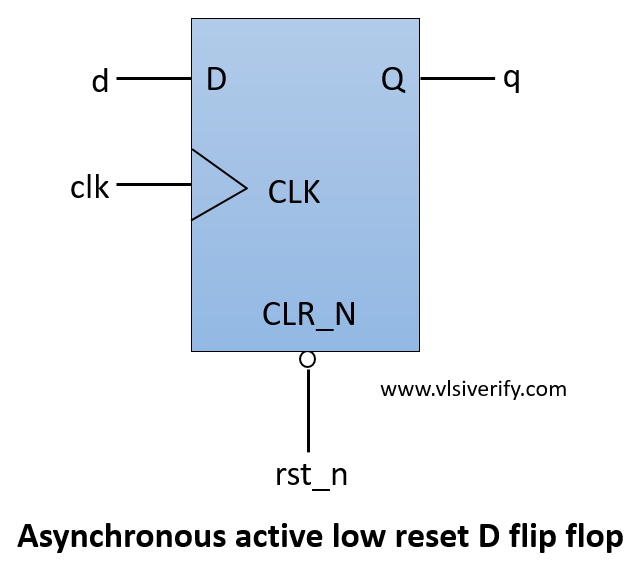
### I. LAB OBJECTIVES

### This Lab experiments are intended to implement Basic Sequential Circuits in Verilog. Students are require to write test bench to simulate the given example code and Top level module to implement these codes in DE2-115 FPGA Kit.

### II. LAB EXPERIMENT EXERCISES

**AIM: WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT THE FOLLOWING DIGITAL SEQUENTIAL LOGIC CIRCUITS:**

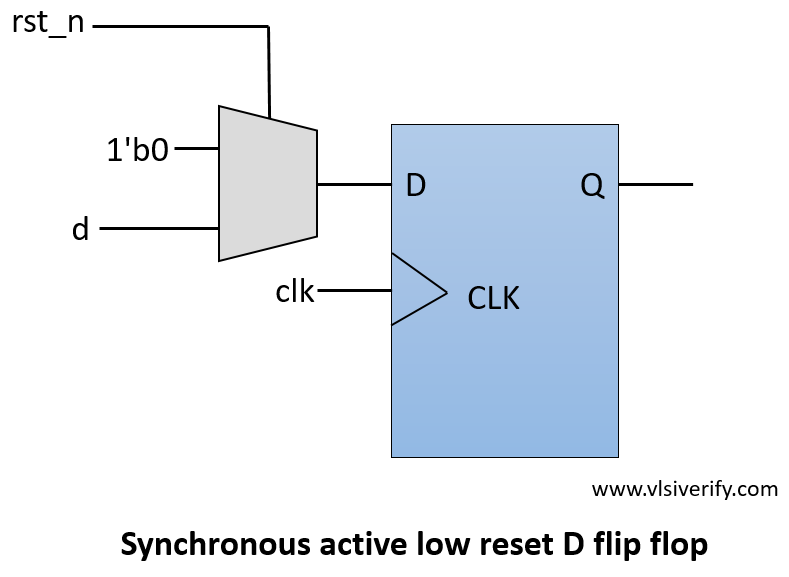
1. DFF with Asynchronous Reset



|  |  |  |  |
| --- | --- | --- | --- |
| Clk | Rst | D | Q |
| 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |

|  |
| --- |
| **Top module** |
| module lab3\_ex1(SW,LEDR,LEDG);  input [17:0] SW;  output [17:0] LEDR;  output [7:0] LEDG;    assign LEDR=SW;    D\_flipflop\_async(.d(SW[2]),.rst(SW[1]),.clk(SW[0]),.q(LEDG[7]));  endmodule |
| **Code** |
| module D\_flipflop\_async(  input clk, rst,  input d,  output reg q  );  always@(posedge clk or posedge rst)  begin  if(rst) q <= 0;  else q <= d;  end  endmodule |
| **Testbench** |
| module tb\_D\_flipflop\_async;  reg clk;  reg rst;  reg d;  wire q;  D\_flipflop\_async uut (  .clk(clk),  .rst(rst),  .d(d),  .q(q)  );  initial begin  clk = 0;  forever #5 clk = ~clk;  end  initial begin  rst = 0;  d = 0;  #10 rst = 1;  #10 rst = 0;  #10 d = 1;  #10 d = 0;  #10 d = 1;  #10 d = 0;  #10 rst = 1;  #10 rst = 0;  #10 d = 1;  #20 $finish;  end  initial begin  $monitor("Time = %t | clk = %b | rst = %b | d = %b | q = %b", $time, clk, rst, d, q);  end  endmodule |
|  |
|  |

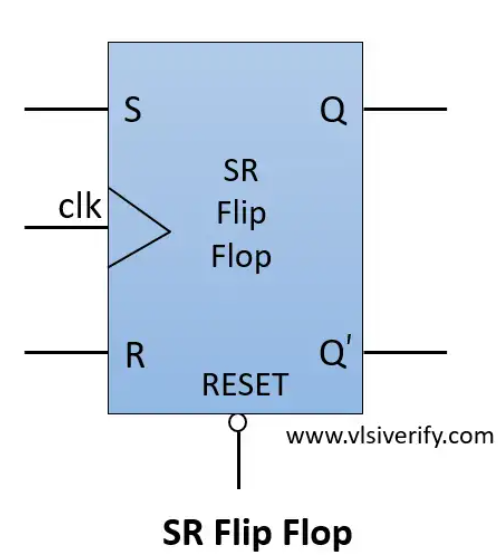
1. DFF with Synchronous Reset

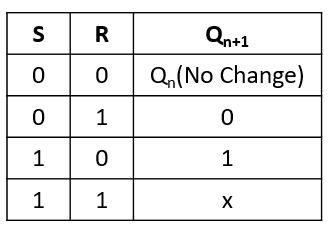


|  |  |  |  |
| --- | --- | --- | --- |
| Clk | Rst | D | Q |
| 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 |

|  |
| --- |
| **Top module** |
| module lab3\_ex2(SW,LEDR,LEDG);  input [17:0] SW;  output [17:0] LEDR;  output [7:0] LEDG;    assign LEDR=SW;    D\_flipflop\_sync(.d(SW[2]),.rst(SW[1]),.clk(SW[0]),.q(LEDG[7]));  endmodule |
| **Code** |
| module D\_flipflop\_sync (  input clk, rst,  input d,  output reg q  );  always@(posedge clk)  begin  if(rst) q <= 0;  else q <= d;  end  endmodule |
| **Testbench** |
| module tb\_D\_flipflop\_sync;  reg clk;  reg rst;  reg d;  wire q;  D\_flipflop\_sync uut (  .clk(clk),  .rst(rst),  .d(d),  .q(q)  );  initial begin  clk = 0;  forever #5 clk = ~clk;  end  initial begin  rst = 0;  d = 0;  #10 rst = 1;  #10 rst = 0;  #10 d = 1;  #10 d = 0;  #10 d = 1;  #10 d = 0;  #10 rst = 1;  #10 rst = 0;  #10 d = 1;      #20 $finish;  end  initial begin  $monitor("Time = %t | clk = %b | rst = %b | d = %b | q = %b", $time, clk, rst, d, q);  end  endmodule |
|  |
|  |

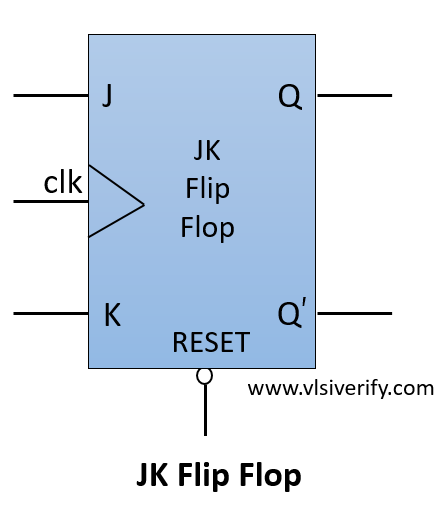
1. SR Flip Flop





|  |
| --- |
| **Top module** |
| module lab3\_ex3(SW,LEDR,LEDG);  input [17:0] SW;  output [17:0] LEDR;  output [7:0] LEDG;    assign LEDR=SW;    SR\_flipflop(.s(SW[1]),.r(SW[0]),.clk(SW[2]),.rst(SW[3]),.q(LEDG[7]),.q\_bar(LEDG[6]));  endmodule |
| **Code** |
| module SR\_flipflop (s,r,clk,rst,q,q\_bar);  input clk, rst, s, r;  output reg q;  output q\_bar;    // always@(posedge clk or posedge rst) // for asynchronous reset  always@(posedge clk) begin // for synchronous reset  if(rst) q <= 0;  else begin  case({s,r})  2'b00: q <= q; // No change  2'b01: q <= 1'b0; // reset  2'b10: q <= 1'b1; // set  2'b11: q <= 1'bx; // Invalid inputs  endcase  end  end  assign q\_bar = ~q;  endmodule |
| **Testbench** |
| module tb\_SR\_flipflop;  reg clk, rst, s, r;  wire q, q\_bar;    SR\_flipflop dff(s, r, clk, rst, q, q\_bar);    always #2 clk = ~clk;  initial begin  clk = 0; rst = 1;  $display("Reset=%b --> q=%b, q\_bar=%b", rst, q, q\_bar);  #3 rst = 0;  $display("Reset=%b --> q=%b, q\_bar=%b", rst, q, q\_bar);    drive(2'b00);  drive(2'b01);  drive(2'b10);  drive(2'b11);  #5;  $finish;  end    task drive(input [1:0] ip);  begin  @(posedge clk);  {s, r} = ip;  #1 $display("s=%b, r=%b --> q=%b, q\_bar=%b", s, r, q, q\_bar);  end  endtask  endmodule |
|  |
|  |

1. JK Flip Flop



A white rectangular grid with black text

Description automatically generated

|  |
| --- |
| **Top module** |
| module lab3\_ex4(SW,LEDR,LEDG);  input [17:0] SW;  output [17:0] LEDR;  output [7:0] LEDG;    assign LEDR=SW;    JK\_flipflop(.j(SW[1]),.k(SW[0]),.clk(SW[2]),.rst(SW[3]),.q(LEDG[7]),.q\_bar(LEDG[6]));  endmodule |
| **Code** |
| module JK\_flipflop (j, k,clk,rst,q,q\_bar);    input clk, rst, j, k;  output reg q;  output q\_bar;    // always@(posedge clk or posedge rst) // for asynchronous reset  always@(posedge clk) begin // for synchronous reset  if(rst) q <= 0;  else begin  case({j,k})  2'b00: q <= q; // No change  2'b01: q <= 1'b0; // reset  2'b10: q <= 1'b1; // set  2'b11: q <= ~q; // Toggle  endcase  end  end  assign q\_bar = ~q;  endmodule |
| **Testbench** |
| module tb\_JK\_flipflop;  reg clk, rst, j, k;  wire q, q\_bar;    JK\_flipflop dff(j, k, clk, rst, q, q\_bar);    always #2 clk = ~clk;  initial begin  clk = 0; rst = 1;  $display("Reset=%b --> q=%b, q\_bar=%b", rst, q, q\_bar);  #3 rst = 0;  $display("Reset=%b --> q=%b, q\_bar=%b", rst, q, q\_bar);    drive(2'b00);  drive(2'b01);  drive(2'b10);  drive(2'b11); // Toggles previous output  drive(2'b11); // Toggles previous output  #5;  $finish;  end    task drive(input [1:0] ip);  begin  @(posedge clk);  {j,k} = ip;  #1 $display("j=%b, k=%b --> q=%b, q\_bar=%b",j, k, q, q\_bar);  end  endtask    endmodule |
|  |
|  |

1. T Flip Flop

|  |
| --- |
| **Top module** |
| module lab3\_ex5(SW,LEDR,LEDG);  input [17:0] SW;  output [17:0] LEDR;  output [7:0] LEDG;    assign LEDR=SW;    T\_flipflop (.clk(SW[2]),.rst(SW[1]),.t(SW[0]),.q(LEDG[7]),.q\_bar(LEDG[6]));    endmodule |
| **Code** |
| module T\_flipflop (clk, rst, t, q, q\_bar);    input clk, rst, t;  output reg q;  output q\_bar;    // always@(posedge clk or posedge rst) // for asynchronous reset  always@(posedge clk) begin // for synchronous reset  if(rst) q <= 0;  else begin  q <= (t?~q:q);  end  end  assign q\_bar = ~q;  endmodule |
| **Testbench** |
| module tb\_T\_flipflop;  reg clk, rst, t;  wire q, q\_bar;    T\_flipflop dff(clk, rst, t, q, q\_bar);    always #2 clk = ~clk;  initial begin  clk = 0; rst = 1;  $display("Reset=%b --> q=%b, q\_bar=%b", rst, q, q\_bar);  #3 rst = 0;  $display("Reset=%b --> q=%b, q\_bar=%b", rst, q, q\_bar);    drive(0); // Same as previous output  drive(1); // Toggles previous output  drive(1); // Toggles previous output  drive(1); // Toggles previous output  drive(0); // Same as previous output  #5;  $finish;  end    task drive(input ip);  begin  @(posedge clk);  t = ip;  #1 $display("t=%b --> q=%b, q\_bar=%b",t, q, q\_bar);  end  endtask    endmodule |
|  |
|  |

1. Right Shift Register 4-bit

|  |
| --- |
| **Top module** |
| module lab3\_ex6(SW,LEDR,LEDG);  input [17:0] SW;  output [17:0] LEDR;  output [7:0]LEDG;  assign LEDR=SW;  shift\_right\_register DUT(.clk(SW[2]), .rst(SW[1]),.d\_in(SW[0]),.d\_out(LEDG[3:0]));  endmodule |
| **Code** |
| module shift\_right\_register(  input clk, // clock input  input rst, // reset input  input d\_in, // data input  output reg [3:0] d\_out // data output  );  always @(posedge clk or posedge rst) begin  if (rst) begin  d\_out <= 4'b0000;  end  else  d\_out<={d\_in, d\_out[3:1]};  end  endmodule |
| **Testbench** |
| module tb\_shift\_right\_register;  reg clk, rst, d\_in;  wire [3:0] d\_out;  shift\_right\_register uut (  .clk(clk),  .rst(rst),  .d\_in(d\_in),  .d\_out(d\_out)  );  always #1 clk = ~clk;  initial begin  clk = 0;  rst = 1;  $display("Reset=%b --> d\_out=%b", rst, d\_out);  #10 rst = 0;  $display("Reset=%b --> d\_out=%b", rst, d\_out);  drive(1);  drive(0);  drive(1);  drive(1);  drive(0);  #5;  $finish;  end  task drive(input value);  begin  d\_in = value;  #2;  $display("d\_in=%b --> d\_out=%b", d\_in, d\_out);  end  endtask  endmodule |
|  |
|  |

1. Left Shift Register 4-bit

|  |
| --- |
| **Top module** |
| module lab3\_ex7(SW,LEDR,LEDG);  input [17:0] SW;  output [17:0] LEDR;  output [7:0]LEDG;  assign LEDR=SW;  shift\_left\_register DUT(.clk(SW[2]), .rst(SW[1]),.d\_in(SW[0]),.d\_out(LEDG[3:0]));  endmodule |
| **Code** |
| module shift\_left\_register(  input clk, // clock input  input rst, // reset input  input d\_in, // data input  output reg [3:0] d\_out // data output  );  always @(posedge clk or posedge rst) begin  if (rst) begin  d\_out <= 4'b0000;  end  else  d\_out<={d\_out[2:0],d\_in};  end  endmodule |
| **Testbench** |
| module tb\_shift\_left\_register;  reg clk, rst, d\_in;  wire [3:0] d\_out;  shift\_left\_register uut (  .clk(clk),  .rst(rst),  .d\_in(d\_in),  .d\_out(d\_out)  );  always #1 clk = ~clk;  initial begin  clk = 0;  rst = 1;  $display("Reset=%b --> d\_out=%b", rst, d\_out);  #10 rst = 0;  $display("Reset=%b --> d\_out=%b", rst, d\_out);  drive(1);  drive(0);  drive(1);  drive(1);  drive(0);  #5;  $finish;  end  task drive(input value);  begin  d\_in = value;  #2;  $display("d\_in=%b --> d\_out=%b", d\_in, d\_out);  end  endtask  endmodule |
|  |
|  |

1. Universal Register 4 bit

|  |
| --- |
| **Top module** |
| module lab3\_ex8(SW,LEDR,LEDG);  input [17:0] SW;  output [17:0] LEDR;  output [7:0]LEDG;  assign LEDR=SW;  Universal\_shift\_register DUT(.clk(SW[3]), .rst(SW[2]),.shift\_control(SW[1]),.d\_in(SW[0]),.d\_out(LEDG[3:0]));  endmodule |
| **Code** |
| module Universal\_shift\_register(  input clk, // clock input  input rst, // reset input  input shift\_control,  input d\_in, // data input  output reg [3:0] d\_out // data output  );  always @(posedge clk or posedge rst) begin  if (rst) begin  d\_out <= 4'b0000;  end  else  begin  if(shift\_control) begin  d\_out<={d\_out[2:0],d\_in}; // shift left  end  else begin  d\_out<={d\_in,d\_out[3:1]}; // shift right  end  end  end  endmodule |
| **Testbench** |
| module tb\_Universal\_shift\_register;  reg clk, rst, shift\_control, d\_in;  wire [3:0] d\_out;  Universal\_shift\_register DUT (  .clk(clk),  .rst(rst),  .shift\_control(shift\_control),  .d\_in(d\_in),  .d\_out(d\_out)  );  always #1 clk = ~clk; // Clock with a period of 10 time units  initial begin  clk = 0; rst = 1; shift\_control = 0; d\_in = 0;  $display("Reset=%b, d\_out=%b", rst, d\_out);  #10 rst = 0;  $display("Reset=%b, d\_out=%b", rst, d\_out);  // Test shifting operations  drive(1, 1); // Shift left with d\_in = 1  drive(1, 1); // Shift left with d\_in = 1  drive(0, 0); // Shift right with d\_in = 0  drive(0, 1); // Shift right with d\_in = 1  drive(1, 1); // Shift left with d\_in = 0  #10;  $finish;  end  task drive(input shift\_ctl, input din);  begin  shift\_control = shift\_ctl;  d\_in = din;  #2; // Wait for one clock cycle  $display("shift\_control=%b, d\_in=%b, d\_out=%b", shift\_control, d\_in, d\_out);  end  endtask  endmodule |
|  |
|  |

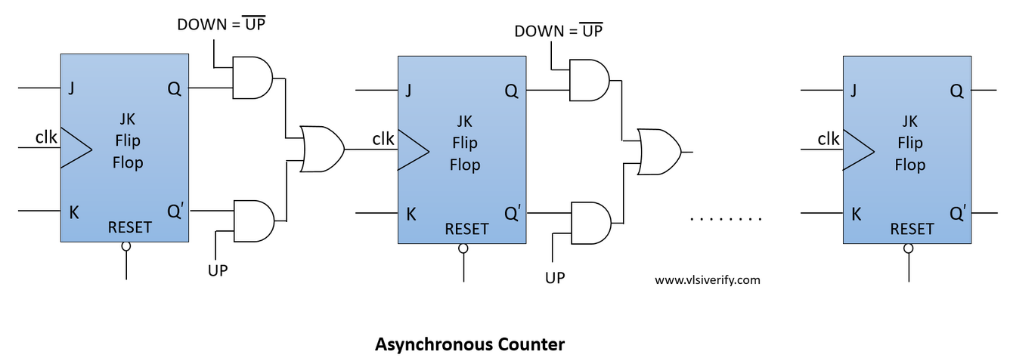
1. Universal Shift Register N-bit

A diagram of a computer scheme

Description automatically generated

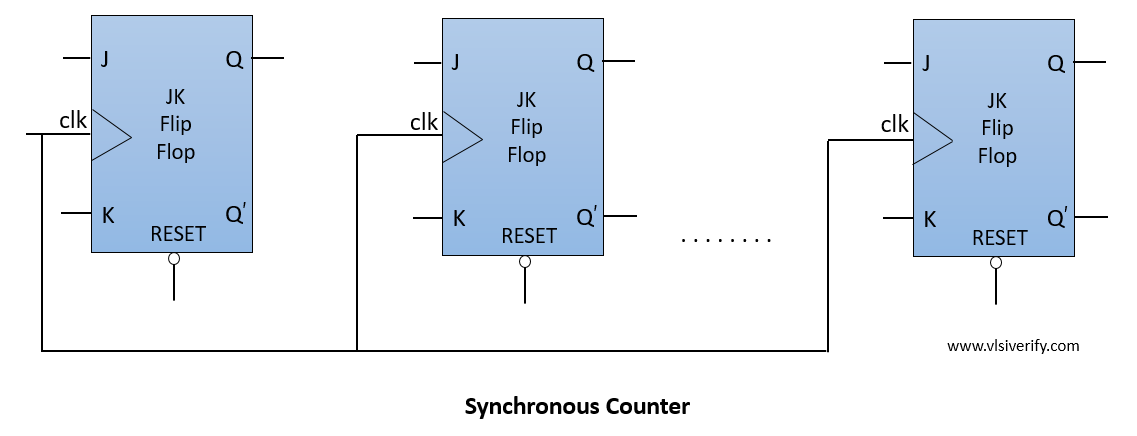
|  |
| --- |
| **Top module** |
| module lab3\_ex9(SW,LEDR,LEDG);  input [17:0] SW;  output [17:0] LEDR;  output [7:0]LEDG;  assign LEDR=SW;  Universal\_shift\_register\_N\_bit(.clk(SW[2]), .rst(SW[1]),.d\_in(SW[0]),.d\_out(LEDG[N-1:0]));  defparam DUT.N=4;  endmodule |
| **Code** |
| module Universal\_shift\_register #(parameter N = 4) (  input clk, // clock input  input rst, // reset input  input shift\_control, // control for shift direction  input d\_in, // data input  output reg [N-1:0] d\_out // data output  );  always @(posedge clk or posedge rst) begin  if (rst) begin  d\_out <= {N{1'b0}}; // Reset the output to 0's  end  else begin  if (shift\_control) begin  d\_out <= {d\_out[N-2:0], d\_in}; // Shift left (MSB gets d\_in)  end  else begin  d\_out <= {d\_in, d\_out[N-1:1]}; // Shift right (LSB gets d\_in)  end  end  end  endmodule |
| **Testbench** |
| module tb\_Universal\_shift\_register\_N\_bit;  reg clk, rst, shift\_control, d\_in;  wire [N-1:0] d\_out;  parameter N = 4; // Can change to any desired size  Universal\_shift\_register #(.N(N)) DUT (  .clk(clk),  .rst(rst),  .shift\_control(shift\_control),  .d\_in(d\_in),  .d\_out(d\_out)  );  always #1 clk = ~clk; // Clock with a period of 2 time units  initial begin  clk = 0; rst = 1; shift\_control = 0; d\_in = 0;  $display("Reset=%b, d\_out=%b", rst, d\_out);  #10 rst = 0;  $display("Reset=%b, d\_out=%b", rst, d\_out);  // Test shifting operations  drive(1, 1); // Shift left with d\_in = 1  drive(1, 1); // Shift left with d\_in = 1  drive(0, 0); // Shift right with d\_in = 0  drive(0, 1); // Shift right with d\_in = 1  drive(1, 1); // Shift left with d\_in = 1  #10;  $finish;  end  task drive(input shift\_ctl, input din);  begin  shift\_control = shift\_ctl;  d\_in = din;  #2; // Wait for one clock cycle  $display("shift\_control=%b, d\_in=%b, d\_out=%b", shift\_control, d\_in, d\_out);  end  endtask  endmodule |
|  |
|  |

1. Asynchronous Counter N-bit



|  |
| --- |
| **Top module** |
| module lab3\_ex10(SW,LEDR,LEDG);  input [17:0] SW;  output [17:0] LEDR;  output [7:0] LEDG;  assign LEDR=SW;  parameter N = 4;  asynchronous\_counter DUT(.clk(SW[2]),.rst(SW[1]),.up\_down(SW[0]),.cnt(LEDG[N-1:0]));  defparam DUT.N=N;  endmodule |
| **Code** |
| module asynchronous\_counter #(parameter N = 4) (  input clk,  input rst,  input up\_down,  output reg [N-1:0] cnt  );  always @(posedge clk or posedge rst) begin  if (rst) begin  cnt <= {N{1'b0}};  end  else begin  if (up\_down && cnt < {N{1'b1}}) begin  cnt <= cnt + 1'b1;  end  else if (!up\_down && cnt > 0) begin  cnt <= cnt - 1'b1;  end  end  end  endmodule |
| **Testbench** |
| module tb\_asynchronous\_counter;  parameter N = 4; // Declare 'N' parameter her  reg clk, rst, up\_down;  wire [N-1:0] cnt; // Wire 'cnt' width is now based on 'N'  asynchronous\_counter #(.N(N)) DUT (  .clk(clk),  .rst(rst),  .up\_down(up\_down),  .cnt(cnt)  );  always #5 clk = ~clk; // Clock with a period of 10 time units  initial begin  // Initial values  clk = 0; rst = 1; up\_down = 1;  $display("rst=%b, up\_down=%b, cnt=%b", rst, up\_down, cnt);    // Apply reset  #10 rst = 0;  $display("rst=%b, up\_down=%b, cnt=%b", rst, up\_down, cnt);  // Test counting up  drive(1); // cnt = 1  drive( 1); // cnt = 2  drive( 1); // cnt = 3  drive(1); // cnt = 4  // Test counting down  up\_down = 0;  drive( 0); // cnt = 3  drive( 0); // cnt = 2  drive( 0); // cnt = 1  drive( 0); // cnt = 0  // Test reset  rst = 1;  #10 rst = 0;  $display("rst=%b, up\_down=%b, cnt=%b", rst, up\_down, cnt);  #10;  $finish;  end  // Task to drive up\_down and observe the counter  task drive(input up\_dn);  begin  up\_down = up\_dn;  #10;  $display("rst=%b, up\_down=%b, cnt=%b", rst, up\_down, cnt);  end  endtask  endmodule |
|  |
|  |

1. Synchronous Counter n-bit



|  |
| --- |
| **Top module** |
| module lab3\_ex11(SW,LEDR,LEDG);  input [17:0] SW;  output [17:0] LEDR;  output [7:0] LEDG;  assign LEDR=SW;  parameter N = 4;  synchronous\_counter DUT(.clk(SW[2]),.rst(SW[1]),.up\_down(SW[0]),.cnt(LEDG[N-1:0]));  defparam DUT.N=N;  endmodule |
| **Code** |
| module synchronous\_counter(clk, rst, up\_down, cnt);  parameter N = 4;  input clk, rst;  input up\_down;  output reg [N-1:0] cnt;  always @(posedge clk) begin  if (rst) begin  cnt <= {N{1'b0}};  end  else begin  if (up\_down && cnt < {N{1'b1}}) begin  cnt <= cnt + 1'b1;  end  else if (!up\_down && cnt > 0) begin  cnt <= cnt - 1'b1;  end  end  end |
| **Testbench** |
| module tb\_synchronous\_counter;  parameter N = 4;    reg clk, rst, up\_down;  wire [N-1:0] cnt;    synchronous\_counter DUT (  .clk(clk),  .rst(rst),  .up\_down(up\_down),  .cnt(cnt)  );  always #5 clk = ~clk;  initial begin  clk = 0; rst = 1; up\_down = 1;  $display("rst=%b, up\_down=%b, cnt=%b", rst, up\_down, cnt);    #10 rst = 0;  $display("rst=%b, up\_down=%b, cnt=%b", rst, up\_down, cnt);    // Test counting up  drive(1); // cnt = 1  drive(1); // cnt = 2  drive(1); // cnt = 3  drive(1); // cnt = 4    // Test counting down  up\_down = 0;  drive(0); // cnt = 3  drive(0); // cnt = 2  drive(0); // cnt = 1  drive(0); // cnt = 0    // Test reset  rst = 1;  #10 rst = 0;  $display("rst=%b, up\_down=%b, cnt=%b", rst, up\_down, cnt);    #10;  $finish;  end  task drive(input up\_dn);  begin  up\_down = up\_dn;  #10;  $display("rst=%b, up\_down=%b, cnt=%b", rst, up\_down, cnt);  end  endtask  endmodule |
|  |
|  |

For each circuit, Do the following steps:

Step 1 : Draw the Schematic of this circuit

### (Show Schematic in Lab report)

Step 2 : Write the truth Table for this circuit

### (Show The Truth Table in Lab report)

Step 3 : Write the Verilog Module to implement this circuit ( using structural, data flow, and behavior modeling)

### (Show Verilog codes of this module in Lab report)

Step 4 : Write the testbench to simulate the Verilog modules of this circuit

### (Show simulation results in Lab report)

Step 5 : Write the Top-level Verilog Code to implement the Verilog modules of this circuit in DE2-FPGA Kit

### (Show implementation results in Lab report)

**IV. LAB REPORT GUIDELINES**

Students write up a report on the Verilog HDL implementation experiment projects created in this lab. The lab report should include Circuit Schematics, Verilog Module Codes, Verilog test bench codes, Top level module to implement the required circuit in FPGA KIT and evidences of data output evidences to validate the experiments (The Captured Screens, Photo of FPGA Kit implementation results).