|  |
| --- |
| **DIGITAL SYSTEM DESIGN LABORATORY** |
| **REPORT LAB 2** |

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

Phạm Quang Vinh ITITIU21347

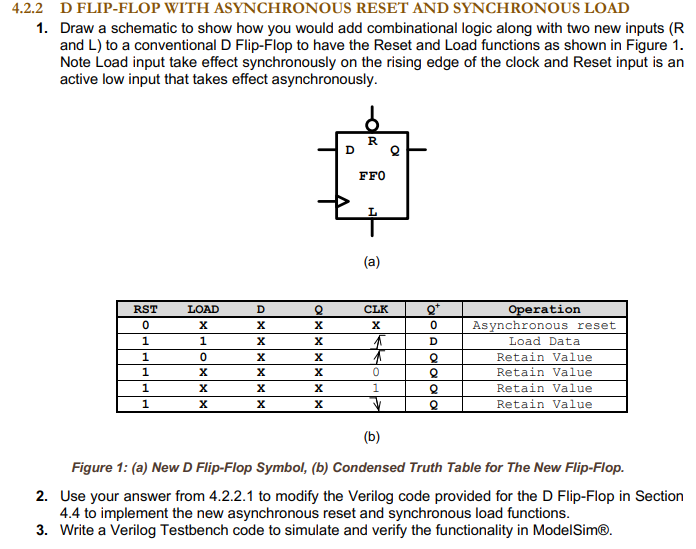
### I. LAB OBJECTIVES

### 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**



**Schematic**

**Truth Table**

**Verilog Code:**

module D\_flipflop (

input clk, rst\_n,

input d,

output reg q

);

always@(posedge clk or negedge rst\_n) begin

if(!rst\_n) q <= 0;

else q <= d;

end

endmodule

**Test bench Code:**

module tb;

reg clk, rst\_n;

reg d;

wire q;

D\_flipflop dff(clk, rst\_n, d, q);

always #2 clk = ~clk;

initial begin

clk = 0; rst\_n = 0;

d = 0;

#3 rst\_n = 1;

repeat(6) begin

d = $urandom\_range(0, 1);

#3;

end

rst\_n = 0; #3;

rst\_n = 1;

repeat(6) begin

d = $urandom\_range(0, 1);

#3;

end

$finish;

end

initial begin

$dumpfile("dump.vcd");

$dumpvars(1);

end

endmodule

**Result :**

A black and green rectangular object with lines

Description automatically generated with medium confidence

**Top-level Verilog Code :**

module top\_module (

input clk, rst\_n,

input [17:0] SW,

output reg [7:0] LEDG,

output reg [17:0] LEDR

);

reg [7:0] d; // Create an array of 8 bits to store D inputs for each flip-flop

wire [7:0] q; // Create an array of 8 bits to store outputs of flip-flops

// Instantiate 8 D\_flipflop modules, each for one LED

genvar i;

generate

for (i = 0; i < 8; i = i + 1) begin : DFF\_INST

D\_flipflop dff(

.clk(clk),

.rst\_n(rst\_n),

.d(d[i]),

.q(q[i])

);

end

endgenerate

// Connect switches to D inputs of flip-flops

assign d = SW[7:0];

// Connect flip-flop outputs to LEDs

assign LEDG = q;

assign LEDR = SW; // Connect all switches to the red LEDs directly

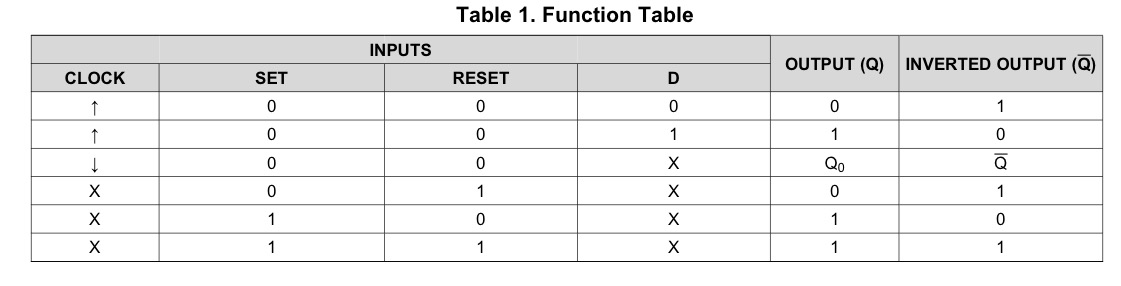
endmodule

1. **DFF with Synchronous Reset**

**A diagram of a block diagram

Description automatically generated**

**Schematic**



**Truth Table**

**Verilog Code:**

module D\_flipflop (

input clk, rst\_n,

input d,

output reg q

);

always@(posedge clk) begin

if(!rst\_n) q <= 0;

else q <= d;

end

endmodule

**Test bench Code:**

module tb;

reg clk, rst\_n;

reg d;

wire q;

D\_flipflop dff(clk, rst\_n, d, q);

always #2 clk = ~clk;

initial begin

clk = 0; rst\_n = 0;

d = 0;

#3 rst\_n = 1;

repeat(6) begin

d = $urandom\_range(0, 1);

#3;

end

rst\_n = 0; #3;

rst\_n = 1;

repeat(6) begin

d = $urandom\_range(0, 1);

#3;

end

$finish;

end

initial begin

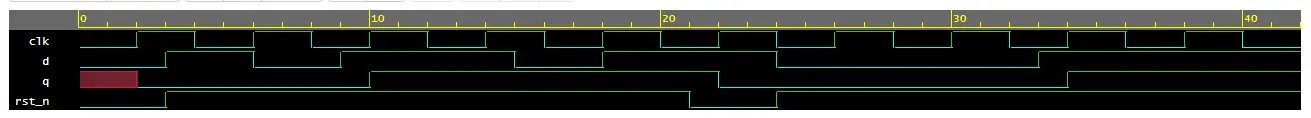
$dumpfile("dump.vcd");

$dumpvars(1);

end

endmodule

**Result :**



**Top-level Verilog Code :**

module top\_module (

input clk, rst\_n,

input [17:0] SW,

output reg [7:0] LEDG,

output reg [17:0] LEDR

);

reg [7:0] d; // Create an array of 8 bits to store D inputs for each flip-flop

wire [7:0] q; // Create an array of 8 bits to store outputs of flip-flops

// Instantiate 8 D\_flipflop modules, each for one LED

genvar i;

generate

for (i = 0; i < 8; i = i + 1) begin : DFF\_INST

D\_flipflop dff(

.clk(clk),

.rst\_n(rst\_n),

.d(d[i]),

.q(q[i])

);

end

endgenerate

// Connect switches to D inputs of flip-flops

assign d = SW[7:0];

// Connect flip-flop outputs to LEDs

assign LEDG = q;

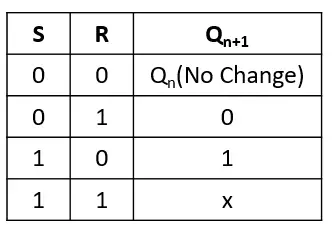
assign LEDR = SW; // Connect all switches to the red LEDs directly

endmodule

1. **A diagram of a flip flop

   Description automatically generatedSR Flip Flop**

**Schematic**

****

**Truth Table**

**Verilog Code:**

module SR\_flipflop (

input clk, rst\_n,

input s,r,

output reg q,

output q\_bar

);

// always@(posedge clk or negedge rst\_n) // for asynchronous reset

always@(posedge clk) begin // for synchronous reset

if(!rst\_n) 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

**Test bench Code:**

module tb;

reg clk, rst\_n;

reg s, r;

wire q, q\_bar;

SR\_flipflop dff(clk, rst\_n, s, r, q, q\_bar);

always #2 clk = ~clk;

initial begin

clk = 0; rst\_n = 0;

$display("Reset=%b --> q=%b, q\_bar=%b", rst\_n, q, q\_bar);

#3 rst\_n = 1;

$display("Reset=%b --> q=%b, q\_bar=%b", rst\_n, q, q\_bar);

drive(2'b00);

drive(2'b01);

drive(2'b10);

drive(2'b11);

#5;

$finish;

end

task drive(bit [1:0] ip);

@(posedge clk);

{s,r} = ip;

#1 $display("s=%b, r=%b --> q=%b, q\_bar=%b",s, r, q, q\_bar);

endtask

initial begin

$dumpfile("dump.vcd");

$dumpvars(1);

end

endmodule

**Result :**

A screenshot of a computer

Description automatically generated

**Top-level Verilog Code :**

module top\_module (

input clk, rst\_n,

input [1:0] SW, // Two switches

output reg [1:0] LED // Two LEDs

);

reg s\_internal, r\_internal;

// Connect the internal signals to the SR flip-flop inputs

assign s = s\_internal;

assign r = r\_internal;

// Test bench instance

tb tb\_inst (

.clk(clk),

.rst\_n(rst\_n),

.s(s\_internal),

.r(r\_internal)

);

// SR flip-flop instance

SR\_flipflop dff (

.clk(clk),

.rst\_n(rst\_n),

.s(s\_internal),

.r(r\_internal),

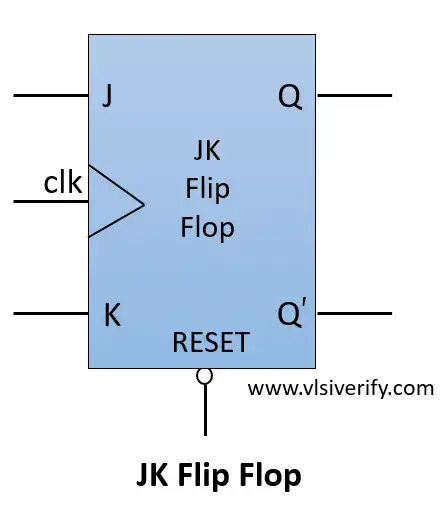
.q(LED[0]), // Connect the Q output to the first LED

.q\_bar(LED[1]) // Connect the Q\_bar output to the second LED

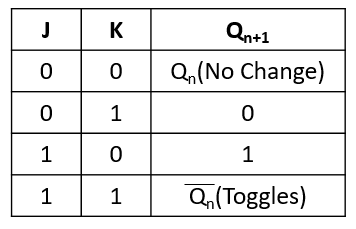
);

endmodule

1. **JK Flip Flop**

****

**Schematic**

****

**Truth Table**

**Verilog Code:**

module JK\_flipflop (

input clk, rst\_n,

input j,k,

output reg q,

output q\_bar

);

// always@(posedge clk or negedge rst\_n) // for asynchronous reset

always@(posedge clk) begin // for synchronous reset

if(!rst\_n) 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

**Test bench Code:**

module tb;

reg clk, rst\_n;

reg j, k;

wire q, q\_bar;

JK\_flipflop dff(clk, rst\_n, j, k, q, q\_bar);

always #2 clk = ~clk;

initial begin

clk = 0; rst\_n = 0;

$display("Reset=%b --> q=%b, q\_bar=%b", rst\_n, q, q\_bar);

#3 rst\_n = 1;

$display("Reset=%b --> q=%b, q\_bar=%b", rst\_n, 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(bit [1:0] ip);

@(posedge clk);

{j,k} = ip;

#1 $display("j=%b, k=%b --> q=%b, q\_bar=%b",j, k, q, q\_bar);

endtask

initial begin

$dumpfile("dump.vcd");

$dumpvars(1);

end

endmodule

**Result :**

A screenshot of a computer

Description automatically generated

**Top-level Verilog Code :**

module top\_module (

input clk, rst\_n,

input [1:0] SW, // Two switches

output reg [1:0] LED // Two LEDs

);

reg j\_internal, k\_internal;

// Connect the internal signals to the JK flip-flop inputs

assign j = j\_internal;

assign k = k\_internal;

// Test bench instance

tb tb\_inst (

.clk(clk),

.rst\_n(rst\_n),

.j(j\_internal),

.k(k\_internal)

);

// JK flip-flop instance

JK\_flipflop dff (

.clk(clk),

.rst\_n(rst\_n),

.j(j\_internal),

.k(k\_internal),

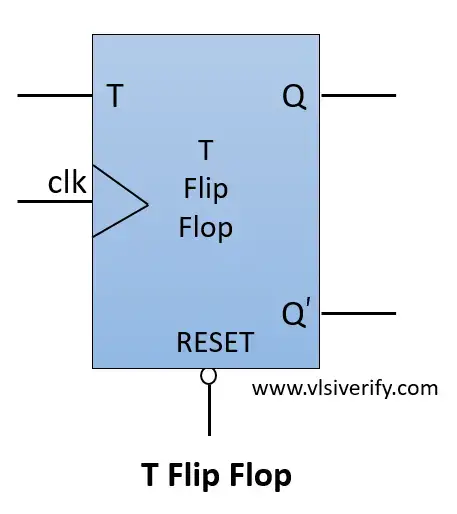
.q(LED[0]), // Connect the Q output to the first LED

.q\_bar(LED[1]) // Connect the Q\_bar output to the second LED

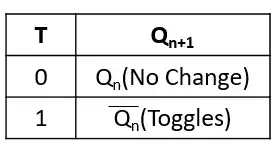
);

endmodule

1. **T Flip Flop**

****

**Schematic**

****

**Truth Table**

**Verilog Code:**

module T\_flipflop (

input clk, rst\_n,

input t,

output reg q,

output q\_bar

);

// always@(posedge clk or negedge rst\_n) // for asynchronous reset

always@(posedge clk) begin // for synchronous reset

if(!rst\_n) q <= 0;

else begin

q <= (t?~q:q);

end

end

assign q\_bar = ~q;

endmodule

**Test bench Code:**

module tb;

reg clk, rst\_n;

reg t;

wire q, q\_bar;

T\_flipflop dff(clk, rst\_n, t, q, q\_bar);

always #2 clk = ~clk;

initial begin

clk = 0; rst\_n = 0;

$display("Reset=%b --> q=%b, q\_bar=%b", rst\_n, q, q\_bar);

#3 rst\_n = 1;

$display("Reset=%b --> q=%b, q\_bar=%b", rst\_n, 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(bit ip);

@(posedge clk);

t = ip;

#1 $display("t=%b --> q=%b, q\_bar=%b",t, q, q\_bar);

endtask

initial begin

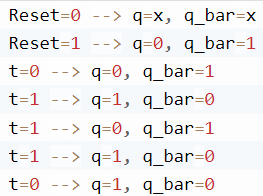
$dumpfile("dump.vcd");

$dumpvars(1);

end

endmodule

**Result :**



**Top-level Verilog Code :**

module top\_module (

input clk, rst\_n,

input [1:0] SW, // Two switches

output reg [1:0] LED // Two LEDs

);

reg t\_internal;

// Connect the internal signal to the T flip-flop input

assign t = t\_internal;

// Test bench instance

tb tb\_inst (

.clk(clk),

.rst\_n(rst\_n),

.t(t\_internal)

);

// T flip-flop instance

T\_flipflop dff (

.clk(clk),

.rst\_n(rst\_n),

.t(t\_internal),

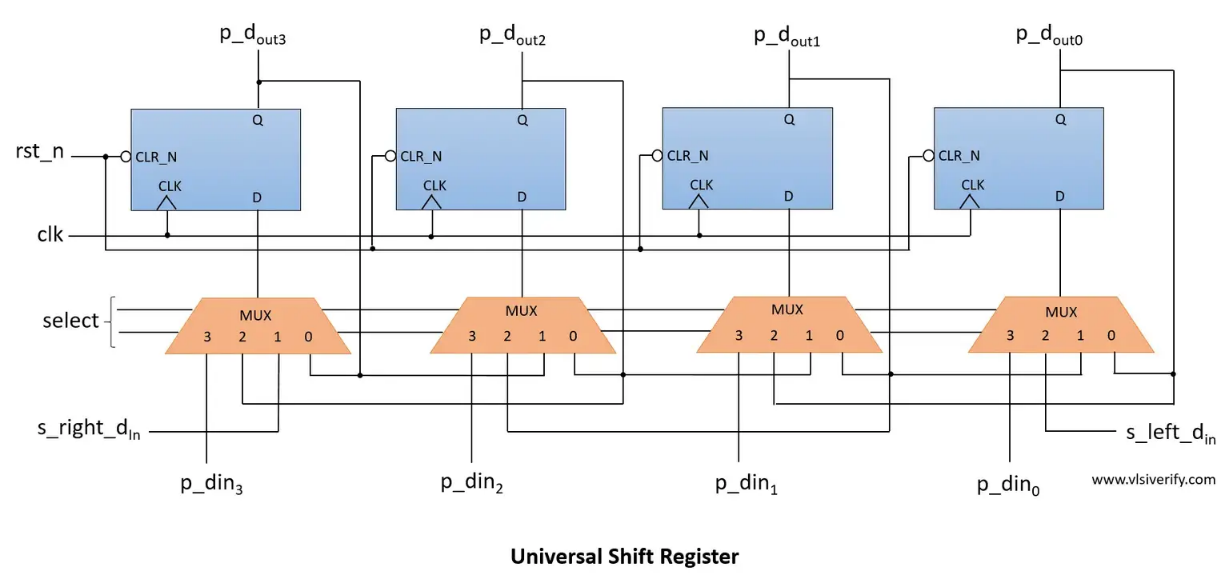
.q(LED[0]), // Connect the Q output to the first LED

.q\_bar(LED[1]) // Connect the Q\_bar output to the second LED

);

endmodule

1. **Universal Shift Register 4-bit**

****

**Schematic**

**A screenshot of a computer

Description automatically generated**

**Truth Table**

**Verilog Code:**

module universal\_shift\_reg(

input clk, rst\_n,

input [1:0] select, // select operation

input [3:0] p\_din, // parallel data in

input s\_left\_din, // serial left data in

input s\_right\_din, // serial right data in

output reg [3:0] p\_dout, //parallel data out

output s\_left\_dout, // serial left data out

output s\_right\_dout // serial right data out

);

always@(posedge clk) begin

if(!rst\_n) p\_dout <= 0;

else begin

case(select)

2'h1: p\_dout <= {s\_right\_din,p\_dout[3:1]}; // Right Shift

2'h2: p\_dout <= {p\_dout[2:0],s\_left\_din}; // Left Shift

2'h3: p\_dout <= p\_din; // Parallel in - Parallel out

default: p\_dout <= p\_dout; // Do nothing

endcase

end

end

assign s\_left\_dout = p\_dout[0];

assign s\_right\_dout = p\_dout[3];

endmodule

**Test bench Code:**

module TB;

reg clk, rst\_n;

reg [1:0] select;

reg [3:0] p\_din;

reg s\_left\_din, s\_right\_din;

wire [3:0] p\_dout; //parallel data out

wire s\_left\_dout, s\_right\_dout;

universal\_shift\_reg usr(clk, rst\_n, select, p\_din, s\_left\_din, s\_right\_din, p\_dout, s\_left\_dout, s\_right\_dout);

always #2 clk = ~clk;

initial begin

$monitor("select=%b, p\_din=%b, s\_left\_din=%b, s\_right\_din=%b --> p\_dout = %b, s\_left\_dout = %b, s\_right\_dout = %b",select, p\_din, s\_left\_din, s\_right\_din, p\_dout, s\_left\_dout, s\_right\_dout);

clk = 0; rst\_n = 0;

#3 rst\_n = 1;

p\_din = 4'b1101;

s\_left\_din = 1'b1;

s\_right\_din = 1'b0;

select = 2'h3; #10;

select = 2'h1; #20;

p\_din = 4'b1101;

select = 2'h3; #10;

select = 2'h2; #20;

select = 2'h0; #20;

$finish;

end

// To enable waveform

initial begin

$dumpfile("dump.vcd"); $dumpvars;

end

endmodule

**Result :**

select=xx, p\_din=xxxx, s\_left\_din=x, s\_right\_din=x --> p\_dout = xxxx, s\_left\_dout = x, s\_right\_dout = x

select=xx, p\_din=xxxx, s\_left\_din=x, s\_right\_din=x --> p\_dout = 0000, s\_left\_dout = 0, s\_right\_dout = 0

select=11, p\_din=1101, s\_left\_din=1, s\_right\_din=0 --> p\_dout = 0000, s\_left\_dout = 0, s\_right\_dout = 0

select=11, p\_din=1101, s\_left\_din=1, s\_right\_din=0 --> p\_dout = 1101, s\_left\_dout = 1, s\_right\_dout = 1

select=01, p\_din=1101, s\_left\_din=1, s\_right\_din=0 --> p\_dout = 1101, s\_left\_dout = 1, s\_right\_dout = 1

select=01, p\_din=1101, s\_left\_din=1, s\_right\_din=0 --> p\_dout = 0110, s\_left\_dout = 0, s\_right\_dout = 0

select=01, p\_din=1101, s\_left\_din=1, s\_right\_din=0 --> p\_dout = 0011, s\_left\_dout = 1, s\_right\_dout = 0

select=01, p\_din=1101, s\_left\_din=1, s\_right\_din=0 --> p\_dout = 0001, s\_left\_dout = 1, s\_right\_dout = 0

select=01, p\_din=1101, s\_left\_din=1, s\_right\_din=0 --> p\_dout = 0000, s\_left\_dout = 0, s\_right\_dout = 0

select=11, p\_din=1101, s\_left\_din=1, s\_right\_din=0 --> p\_dout = 0000, s\_left\_dout = 0, s\_right\_dout = 0

select=11, p\_din=1101, s\_left\_din=1, s\_right\_din=0 --> p\_dout = 1101, s\_left\_dout = 1, s\_right\_dout = 1

select=10, p\_din=1101, s\_left\_din=1, s\_right\_din=0 --> p\_dout = 1101, s\_left\_dout = 1, s\_right\_dout = 1

select=10, p\_din=1101, s\_left\_din=1, s\_right\_din=0 --> p\_dout = 1011, s\_left\_dout = 1, s\_right\_dout = 1

select=10, p\_din=1101, s\_left\_din=1, s\_right\_din=0 --> p\_dout = 0111, s\_left\_dout = 1, s\_right\_dout = 0

select=10, p\_din=1101, s\_left\_din=1, s\_right\_din=0 --> p\_dout = 1111, s\_left\_dout = 1, s\_right\_dout = 1

select=00, p\_din=1101, s\_left\_din=1, s\_right\_din=0 --> p\_dout = 1111, s\_left\_dout = 1, s\_right\_dout = 1

**Top-level Verilog Code :**

module top\_module (

input clk, rst\_n,

input [1:0] select, // Select operation

input [3:0] p\_din, // Parallel data in

input s\_left\_din, // Serial left data in

input s\_right\_din, // Serial right data in

output reg [3:0] p\_dout, // Parallel data out

output reg s\_left\_dout, // Serial left data out

output reg s\_right\_dout // Serial right data out

);

// Universal Shift Register instance

universal\_shift\_reg usr\_inst (

.clk(clk),

.rst\_n(rst\_n),

.select(select),

.p\_din(p\_din),

.s\_left\_din(s\_left\_din),

.s\_right\_din(s\_right\_din),

.p\_dout(p\_dout),

.s\_left\_dout(s\_left\_dout),

.s\_right\_dout(s\_right\_dout)

);

endmodule

1. **Universal Shift Register 8-bit**

**Verilog Code:**

module universal\_shift\_reg\_8bit(

input clk, rst\_n,

input [1:0] select, // select operation

input [7:0] p\_din, // parallel data in

input s\_left\_din, // serial left data in

input s\_right\_din, // serial right data in

output reg [7:0] p\_dout, //parallel data out

output s\_left\_dout, // serial left data out

output s\_right\_dout // serial right data out

);

always@(posedge clk) begin

if(!rst\_n) p\_dout <= 0;

else begin

case(select)

2'h1: p\_dout <= {s\_right\_din,p\_dout[7:1]}; // Right Shift

2'h2: p\_dout <= {p\_dout[6:0],s\_left\_din}; // Left Shift

2'h3: p\_dout <= p\_din; // Parallel in - Parallel out

default: p\_dout <= p\_dout; // Do nothing

endcase

end

end

assign s\_left\_dout = p\_dout[0];

assign s\_right\_dout = p\_dout[7];

endmodule

**Test bench Code:**

module universal\_shift\_reg\_8bit\_tb;

// Parameters

parameter CLK\_PERIOD = 10; // Clock period in simulation time units

// Signals

reg clk, rst\_n;

reg [1:0] select;

reg [7:0] p\_din;

reg s\_left\_din, s\_right\_din;

wire [7:0] p\_dout;

wire s\_left\_dout, s\_right\_dout;

// Instantiate the DUT

universal\_shift\_reg\_8bit dut (

.clk(clk),

.rst\_n(rst\_n),

.select(select),

.p\_din(p\_din),

.s\_left\_din(s\_left\_din),

.s\_right\_din(s\_right\_din),

.p\_dout(p\_dout),

.s\_left\_dout(s\_left\_dout),

.s\_right\_dout(s\_right\_dout)

);

// Clock generation

always #((CLK\_PERIOD / 2)) clk = ~clk;

// Stimulus

initial begin

// Initialize inputs

clk = 0;

rst\_n = 0;

select = 2'b00; // Do nothing initially

p\_din = 8'h00;

s\_left\_din = 0;

s\_right\_din = 0;

// Reset

#20 rst\_n = 1;

// Test shifting to the right

select = 2'b01; // Right shift

p\_din = 8'b10101010; // Input data

#50;

// Test shifting to the left

select = 2'b10; // Left shift

s\_left\_din = 1; // Serial left input

#50;

// Test parallel in - parallel out

select = 2'b11; // Parallel in - parallel out

p\_din = 8'b11001100; // Input data

#50;

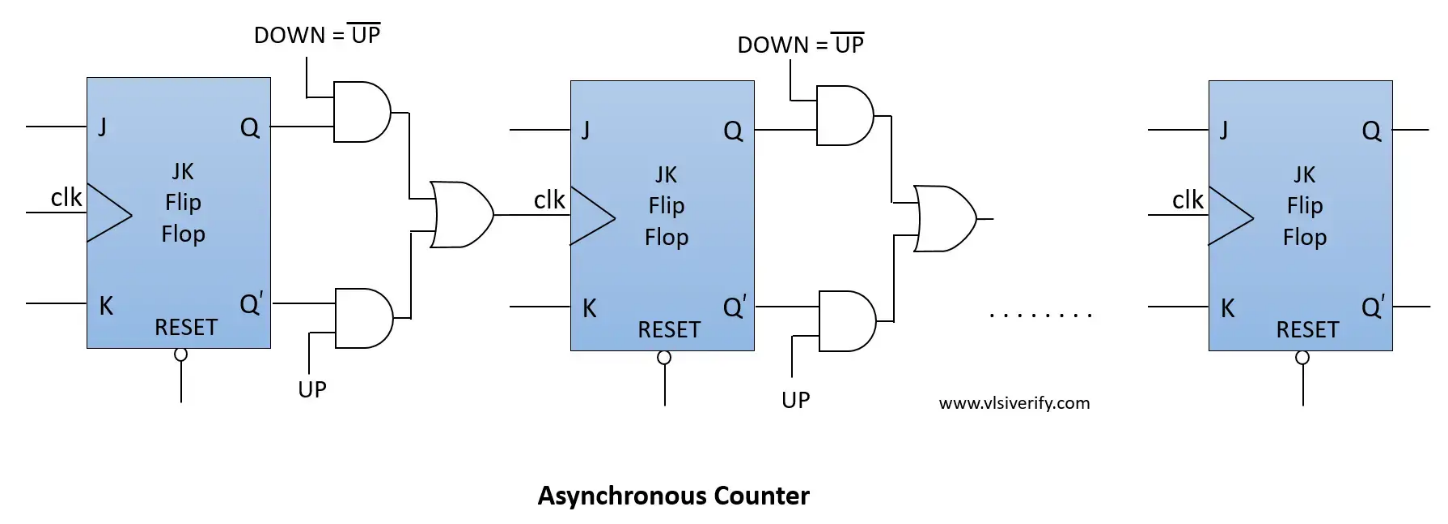
// End simulation

$finish;

end

endmodule

1. **Asynchronous Counter 4-bit**

****

**Schematic**

**Verilog Code:**

module JK\_flipflop (

input clk, rst\_n,

input j,k,

output reg q, q\_bar

);

always@(posedge clk or negedge rst\_n) begin // for asynchronous reset

if(!rst\_n) 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

module updown\_selector(input q, q\_bar, bit up, output nclk);

assign nclk = up?q\_bar:q;

endmodule

module asynchronous\_counter #(parameter SIZE=4)(

input clk, rst\_n,

input j, k,

input up,

output [3:0] q, q\_bar

);

wire [3:0] nclk;

genvar g;

/\*

// Up Counter (at output q)

JK\_flipflop jk1(clk, rst\_n, j, k, q[0], q\_bar[0]);

JK\_flipflop jk2(q\_bar[0], rst\_n, j, k, q[1], q\_bar[1]);

JK\_flipflop jk3(q\_bar[1], rst\_n, j, k, q[2], q\_bar[2]);

JK\_flipflop jk4(q\_bar[2], rst\_n, j, k, q[3], q\_bar[3]);

// Down Counter (at output q)

JK\_flipflop jk1(clk, rst\_n, j, k, q[0], q\_bar[0]);

JK\_flipflop jk2(q[0], rst\_n, j, k, q[1], q\_bar[1]);

JK\_flipflop jk3(q[1], rst\_n, j, k, q[2], q\_bar[2]);

JK\_flipflop jk4(q[2], rst\_n, j, k, q[3], q\_bar[3]);

// Up and Down Counter (at output q)

JK\_flipflop jk1(clk, rst\_n, j, k, q[0], q\_bar[0]);

updown\_selector ud1(q[0], q\_bar[0], up, nclk[0]);

JK\_flipflop jk2(nclk[0], rst\_n, j, k, q[1], q\_bar[1]);

updown\_selector ud2(q[1], q\_bar[1], up, nclk[1]);

JK\_flipflop jk3(nclk[1], rst\_n, j, k, q[2], q\_bar[2]);

updown\_selector ud3(q[2], q\_bar[2], up, nclk[2]);

JK\_flipflop jk4(nclk[2], rst\_n, j, k, q[3], q\_bar[3]);

\*/

// Using generate block

JK\_flipflop jk0(clk, rst\_n, j, k, q[0], q\_bar[0]);

generate

for(g = 1; g<SIZE; g++) begin

updown\_selector ud1(q[g-1], q\_bar[g-1], up, nclk[g-1]);

JK\_flipflop jk1(nclk[g-1], rst\_n, j, k, q[g], q\_bar[g]);

end

endgenerate

endmodule

**Test bench Code:**

module tb;

reg clk, rst\_n;

reg j, k;

reg up;

wire [3:0] q, q\_bar;

asynchronous\_counter(clk, rst\_n, j, k, up, q, q\_bar);

initial begin

clk = 0; rst\_n = 0;

up = 1;

#4; rst\_n = 1;

j = 1; k = 1;

#80;

rst\_n = 0;

#4; rst\_n = 1; up = 0;

#50;

$finish;

end

always #2 clk = ~clk;

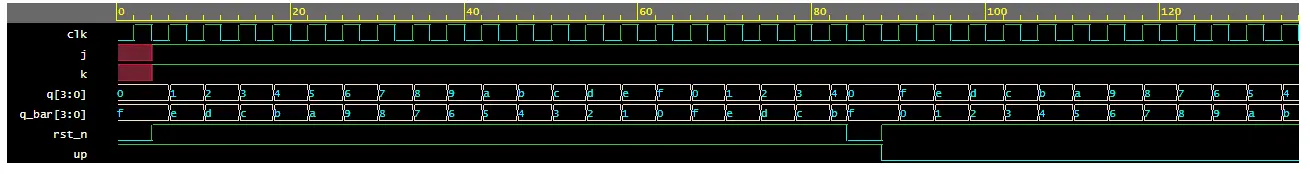
initial begin

$dumpfile("dump.vcd"); $dumpvars;

end

endmodule

**Result :**



1. **Asynchronous Counter 8-bit**

**Verilog Code:**

module JK\_flipflop (

input clk, rst\_n,

input j, k,

output reg q, q\_bar

);

always@(posedge clk or negedge rst\_n) begin // for asynchronous reset

if(!rst\_n) 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

module updown\_selector(

input q, q\_bar,

input bit up,

output nclk

);

assign nclk = up ? q\_bar : q;

endmodule

module asynchronous\_counter\_8bit #(parameter SIZE=8)(

input clk, rst\_n,

input j, k,

input up,

output reg [SIZE-1:0] q, q\_bar

);

wire [SIZE-1:0] nclk;

genvar g;

// Using generate block

JK\_flipflop jk0(clk, rst\_n, j, k, q[0], q\_bar[0]);

generate

for(g = 1; g < SIZE; g++) begin

updown\_selector ud1(q[g-1], q\_bar[g-1], up, nclk[g-1]);

JK\_flipflop jk1(nclk[g-1], rst\_n, j, k, q[g], q\_bar[g]);

end

endgenerate

endmodule

module asynchronous\_counter\_8bit\_tb;

// Parameters

parameter CLK\_PERIOD = 10; // Clock period in simulation time units

parameter SIZE = 8; // Counter size

// Signals

reg clk, rst\_n;

reg [1:0] j, k;

reg up;

wire [SIZE-1:0] q;

wire [SIZE-1:0] q\_bar;

// Instantiate the DUT

asynchronous\_counter\_8bit #(.SIZE(SIZE)) dut (

.clk(clk),

.rst\_n(rst\_n),

.j(j),

.k(k),

.up(up),

.q(q),

.q\_bar(q\_bar)

);

// Clock generation

always #((CLK\_PERIOD / 2)) clk = ~clk;

// Stimulus

initial begin

// Initialize inputs

clk = 0;

rst\_n = 0;

j = 2'b00;

k = 2'b00;

up = 1'b0;

// Reset

#20 rst\_n = 1;

// Test Up Counter

up = 1;

#50;

// Test Down Counter

up = 0;

#50;

// Test Up-Down Counter

up = 1;

#50;

up = 0;

#50;

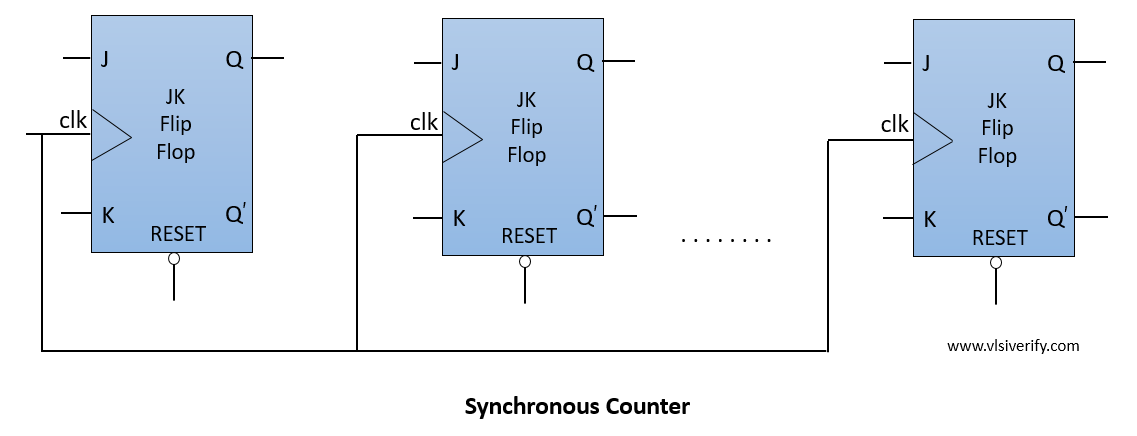
// End simulation

$finish;

end

endmodule

1. **Synchronous Counter 4-bit**

****

**Schematic**

**Verilog Code:**

module synchronous\_counter #(parameter SIZE=4)(

input clk, rst\_n,

input up,

output reg [3:0] cnt

);

always@(posedge clk) begin

if(!rst\_n) begin

cnt <= 4'h0;

end

else begin

if(up) cnt <= cnt + 1'b1;

else cnt <= cnt - 1'b1;

end

end

endmodule

**Test bench Code:**

module tb;

reg clk, rst\_n;

reg up;

wire [3:0] cnt;

synchronous\_counter(clk, rst\_n, up, cnt);

initial begin

clk = 0; rst\_n = 0;

up = 1;

#4; rst\_n = 1;

#80;

rst\_n = 0;

#4; rst\_n = 1; up = 0;

#50;

$finish;

end

always #2 clk = ~clk;

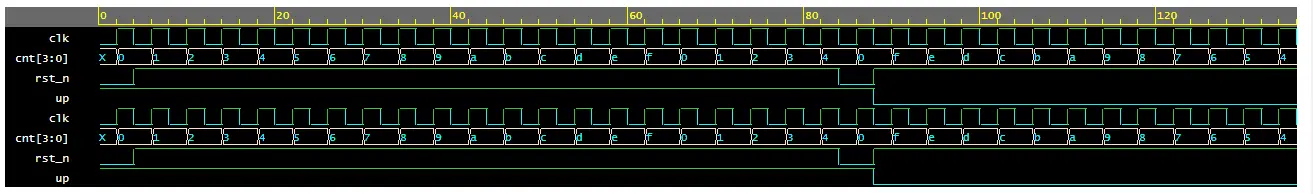
initial begin

$dumpfile("dump.vcd"); $dumpvars;

end

endmodule

**Result :**



1. **Synchronous Counter 8-bit**

**Verilog Code:**

module synchronous\_counter\_8bit #(parameter SIZE=8)(

input clk, rst\_n,

input up,

output reg [SIZE-1:0] cnt

);

always@(posedge clk) begin

if(!rst\_n) begin

cnt <= SIZE'd0;

end

else begin

if(up) cnt <= cnt + 1'b1;

else cnt <= cnt - 1'b1;

end

end

endmodule

1. **Synchronous Counter n-bit**

**Verilog Code:**

module synchronous\_counter #(parameter SIZE=4)(

input clk, rst\_n,

input up,

output reg [SIZE-1:0] cnt

);

always @(posedge clk or negedge rst\_n) begin

if (!rst\_n) begin

cnt <= {SIZE{1'b0}}; // Initialize to all zeros

end

else begin

if (up)

cnt <= cnt + 1'b1;

else

cnt <= cnt - 1'b1;

end

end

endmodule