**Assignment 5**

**Name: Nikhil Girish**

**SRN: PES2UG21CS334**

**Sem: III Sec: F**

**Code:**

Alu.v:

module alu\_slice (input wire [1:0] op, input wire i0, i1, cin, output wire o, cout); wire t\_sumdiff, t\_and, t\_or, t\_andor; addsub \_i0 (op[0], i0, i1, cin, t\_sumdiff, cout); and2 \_i1 (i0, i1, t\_and); or2 \_i2 (i0, i1, t\_or); mux2 \_i3 (t\_and, t\_or, op[0], t\_andor); mux2 \_i4 (t\_sumdiff, t\_andor, op[1], o); endmodule

module alu (input wire [1:0] op, input wire [15:0] i0, i1, output wire [15:0] o, output wire cout); wire [14:0] c; alu\_slice \_i0 (op, i0[0], i1[0], op[0] , o[0], c[0]); alu\_slice \_i1 (op, i0[1], i1[1], c[0], o[1], c[1]); alu\_slice \_i2 (op, i0[2], i1[2], c[1], o[2], c[2]); alu\_slice \_i3 (op, i0[3], i1[3], c[2], o[3], c[3]); alu\_slice \_i4 (op, i0[4], i1[4], c[3], o[4], c[4]); alu\_slice \_i5 (op, i0[5], i1[5], c[4], o[5], c[5]); alu\_slice \_i6 (op, i0[6], i1[6], c[5], o[6], c[6]); alu\_slice \_i7 (op, i0[7], i1[7], c[6], o[7], c[7]); alu\_slice \_i8 (op, i0[8], i1[8], c[7], o[8], c[8]); alu\_slice \_i9 (op, i0[9], i1[9], c[8], o[9], c[9]); alu\_slice \_i10 (op, i0[10], i1[10], c[9] , o[10], c[10]); alu\_slice \_i11 (op, i0[11], i1[11], c[10], o[11], c[11]); alu\_slice \_i12 (op, i0[12], i1[12], c[11], o[12], c[12]); alu\_slice \_i13 (op, i0[13], i1[13], c[12], o[13], c[13]); alu\_slice \_i14 (op, i0[14], i1[14], c[13], o[14], c[14]); alu\_slice \_i15 (op, i0[15], i1[15], c[14], o[15], cout);

endmodule

lib.v: module invert (input wire i, output wire o); assign o = !i;

endmodule

module and2 (input wire i0, i1, output wire o); assign o = i0 & i1; endmodule

module or2 (input wire i0, i1, output wire o); assign o = i0 | i1;

endmodule

module xor2 (input wire i0, i1, output wire o); assign o = i0 ^ i1; endmodule

module nand2 (input wire i0, i1, output wire o); wire t; and2 and2\_0 (i0, i1, t); invert invert\_0 (t, o);

endmodule

module nor2 (input wire i0, i1, output wire o); wire t; or2 or2\_0 (i0, i1, t); invert invert\_0 (t, o);

endmodule

module xnor2 (input wire i0, i1, output wire o); wire t; xor2 xor2\_0 (i0, i1, t); invert invert\_0 (t, o);

endmodule

module and3 (input wire i0, i1, i2, output wire o); wire t; and2 and2\_0 (i0, i1, t); and2 and2\_1 (i2, t, o);

endmodule

module or3 (input wire i0, i1, i2, output wire o); wire t; or2 or2\_0 (i0, i1, t);

or2 or2\_1 (i2, t, o);

endmodule

module nor3 (input wire i0, i1, i2, output wire o); wire t; or2 or2\_0 (i0, i1, t); nor2 nor2\_0 (i2, t, o);

endmodule

module nand3 (input wire i0, i1, i2, output wire o); wire t; and2 and2\_0 (i0, i1, t); nand2 nand2\_1 (i2, t, o); endmodule

module xor3 (input wire i0, i1, i2, output wire o); wire t; xor2 xor2\_0 (i0, i1, t); xor2 xor2\_1 (i2, t, o);

endmodule

module xnor3 (input wire i0, i1, i2, output wire o); wire t; xor2 xor2\_0 (i0, i1, t); xnor2 xnor2\_0 (i2, t, o);

endmodule

module mux2 (input wire i0, i1, j, output wire o); assign o = (j==0)?i0:i1; endmodule

module mux3 (input wire [0:2] i, input wire j1, j0, output wire o); wire t; mux2 mux2\_0 (i[0], i[1], j0, t); mux2 mux2\_1 (t, i[2], j1, o);

endmodule

module mux4 (input wire [0:3] i, input wire j1, j0, output wire o); wire t0, t1; mux2 mux2\_0 (i[0], i[1], j1, t0); mux2 mux2\_1 (i[2], i[3], j1, t1); mux2 mux2\_2 (t0, t1, j0, o); endmodule

module mux8 (input wire [0:7] i, input wire j2, j1, j0, output wire o); wire t0, t1; mux4 mux4\_0 (i[0:3], j2, j1, t0); mux4 mux4\_1 (i[4:7], j2, j1, t1); mux2 mux2\_0 (t0, t1, j0, o); endmodule

module demux2 (input wire i, j, output wire o0, o1); assign o0 = (j==0)?i:1'b0; assign o1 = (j==1)?i:1'b0; endmodule

module demux4 (input wire i, j1, j0, output wire [0:3] o); wire t0, t1; demux2 demux2\_0 (i, j1, t0, t1); demux2 demux2\_1 (t0, j0, o[0], o[1]); demux2 demux2\_2 (t1, j0, o[2], o[3]); endmodule

module demux8 (input wire i, j2, j1, j0, output wire [0:7] o); wire t0, t1; demux2 demux2\_0 (i, j2, t0, t1); demux4 demux4\_0 (t0, j1, j0, o[0:3]); demux4 demux4\_1 (t1, j1, j0, o[4:7]); endmodule

module df (input wire clk, in, output wire out); reg df\_out; always@(posedge clk) df\_out <= in; assign out = df\_out; endmodule

module dfr (input wire clk, reset, in, output wire out); wire reset\_, df\_in; invert invert\_0 (reset, reset\_); and2 and2\_0 (in, reset\_, df\_in); df df\_0 (clk, df\_in, out);

endmodule

module dfrl (input wire clk, reset, load, in, output wire out); wire \_in; mux2 mux2\_0(out, in, load, \_in);

dfr dfr\_1(clk, reset, \_in, out);

endmodule

module dfs (input wire clk, set, in, output wire out); wire dfr\_in,dfr\_out; invert invert\_0(in, dfr\_in); invert invert\_1(dfr\_out, out); dfr dfr\_2(clk, set, dfr\_in, dfr\_out);

endmodule

module dfsl (input wire clk, set, load, in, output wire out); wire \_in; mux2 mux2\_0(out, in, load, \_in); dfs dfs\_1(clk, set, \_in, out);

endmodule

module fa (input wire i0, i1, cin, output wire sum, cout); wire t0, t1, t2; xor3 \_i0 (i0, i1, cin, sum); and2 \_i1 (i0, i1, t0); and2 \_i2 (i1, cin, t1); and2 \_i3 (cin, i0, t2); or3 \_i4 (t0, t1, t2, cout);

endmodule

module addsub (input wire addsub, i0, i1, cin, output wire sumdiff, cout); wire t; fa \_i0 (i0, t, cin, sumdiff, cout); xor2 \_i1 (i1, addsub, t);

endmodule

mproc.v: module nor5 (input wire [0:4] i, output wire o); wire t; or3 or3\_0 (i[0], i[1], i[2], t); nor3 nor3\_0 (t, i[3], i[4], o);

endmodule

module ir (input wire clk, reset, load, input wire [15:0] din, output wire [15:0] dout); dfrl dfrl\_0 (clk, reset, load, din['h0], dout['h0]); dfrl dfrl\_1 (clk, reset, load, din['h1], dout['h1]); dfrl dfrl\_2 (clk, reset, load, din['h2], dout['h2]); dfrl dfrl\_3 (clk, reset, load, din['h3], dout['h3]);

dfrl dfrl\_4 (clk, reset, load, din['h4], dout['h4]); dfrl dfrl\_5 (clk, reset, load, din['h5], dout['h5]); dfrl dfrl\_6 (clk, reset, load, din['h6], dout['h6]); dfrl dfrl\_7 (clk, reset, load, din['h7], dout['h7]); dfrl dfrl\_8 (clk, reset, load, din['h8], dout['h8]); dfrl dfrl\_9 (clk, reset, load, din['h9], dout['h9]); dfrl dfrl\_a (clk, reset, load, din['ha], dout['ha]); dfrl dfrl\_b (clk, reset, load, din['hb], dout['hb]); dfrl dfrl\_c (clk, reset, load, din['hc], dout['hc]); dfrl dfrl\_d (clk, reset, load, din['hd], dout['hd]); dfrl dfrl\_e (clk, reset, load, din['he], dout['he]); dfrl dfrl\_f (clk, reset, load, din['hf], dout['hf]);

endmodule

module control\_logic (input wire clk, reset, input wire [15:0] cur\_ins, output wire [2:0] rd\_addr\_a, rd\_addr\_b, wr\_addr, output wire [1:0] op, output wire sel, jump, pc\_inc, load\_ir, wr\_reg);

// Copy your assignment 3 logic here and modify.

endmodule

module mproc (input wire clk, reset, input wire [15:0] d\_in, output wire [6:0] addr, output wire [15:0] d\_out); wire pc\_inc, cout, cout\_, sub, sel, sel\_addr; wire [2:0] rd\_addr\_a, rd\_addr\_b, wr\_addr; wire [1:0] op; wire [8:0] \_addr; wire [15:0] cur\_ins, d\_out\_a, d\_out\_b;

and2 and2\_0 (jump, cout, sub); pc pc\_0 (clk, reset, pc\_inc, 1'b0, sub, {8'b0, cur\_ins[7:0]}, {\_addr, addr}); ir ir\_0 (clk, reset, load\_ir, d\_in, cur\_ins); control\_logic control\_logic\_0 (clk, reset, cur\_ins, rd\_addr\_a, rd\_addr\_b, wr\_addr, op, sel, jump, pc\_inc, load\_ir, wr\_reg); reg\_alu reg\_alu\_0 (clk, reset, sel, wr\_reg, op, rd\_addr\_a, rd\_addr\_b, wr\_addr, d\_in, d\_out\_a, d\_out\_b, cout); assign d\_out = d\_out\_a; endmodule

mproc\_mem.v:

module ram\_128\_16 (input wire clk, reset, wr, input wire [6:0] addr, input wire [15:0] din, output wire [15:0] dout); reg [0:127] ram [15:0];

initial begin

ram[0]=16'b10xxxxx000xxxxxx; // load r0, 1 ram[1]=16'h01; ram[2]=16'b10xxxxx001xxxxxx; // load r1, 21 ram[3]=16'h15; ram[4]=16'b10xxxxx010xxxxxx; // load r2, 1 ram[5]=16'h01; ram[6]=16'b10xxxxx011xxxxxx; // load r3, 1 ram[7]=16'h01; ram[8]=16'b0000000100010011; // add r4, r2, r3 ram[9]=16'b0000010010011011; // move r2, r3 (and r2, r3, r3) ram[10]=16'b0000010011100100; // move r3, r4 (and r3, r4, r4) ram[11]=16'b0000001001000001; // sub r1, r0, r1 ram[12]=16'b01xxxxxx00000101; // jbc ram[8] ram[13]=16'b0000010000100100; // move r0, r4 (and r0, r4, r4) end always @(wr) ram[addr]=din; assign dout=ram[addr]; endmodule

module mproc\_mem (input wire clk, reset); wire wr; wire [6:0] addr; wire [15:0] d\_in, d\_out;

ram\_128\_16 ram\_128\_16\_0 (clk, reset, 1'b0, addr, d\_out, d\_in); mproc mproc\_0 (clk, reset, d\_in, addr, d\_out); endmodule

pc.v:

module pc\_slice (input wire clk, reset, cin, load, inc, sub, offset, output wire cout, pc); wire in, inc\_; invert invert\_0 (inc, inc\_); and2 and2\_0 (offset, inc\_, t); addsub addsub\_0 (sub, pc, t, cin, in, cout); dfrl dfrl\_0 (clk, reset, load, in, pc);

endmodule

module pc\_slice0 (input wire clk, reset, cin, load, inc, sub, offset, output wire cout, pc); wire in; or2 or2\_0 (offset, inc, t); addsub addsub\_0 (sub, pc, t, cin, in, cout); dfrl dfrl\_0 (clk, reset, load, in, pc);

endmodule

module pc (input wire clk, reset, inc, add, sub, input wire [15:0] offset, output wire [15:0] pc); input wire load; input wire [15:0] c; or3 or3\_0 (inc, add, sub, load); pc\_slice0 pc\_slice\_0 (clk, reset, sub, load, inc, sub, offset[0], c[0], pc[0]); pc\_slice pc\_slice\_1 (clk, reset, c[0], load, inc, sub, offset[1], c[1], pc[1]); pc\_slice pc\_slice\_2 (clk, reset, c[1], load, inc, sub, offset[2], c[2], pc[2]); pc\_slice pc\_slice\_3 (clk, reset, c[2], load, inc, sub, offset[3], c[3], pc[3]); pc\_slice pc\_slice\_4 (clk, reset, c[3], load, inc, sub, offset[4], c[4], pc[4]); pc\_slice pc\_slice\_5 (clk, reset, c[4], load, inc, sub, offset[5], c[5], pc[5]); pc\_slice pc\_slice\_6 (clk, reset, c[5], load, inc, sub, offset[6], c[6], pc[6]); pc\_slice pc\_slice\_7 (clk, reset, c[6], load, inc, sub, offset[7], c[7], pc[7]); pc\_slice pc\_slice\_8 (clk, reset, c[7], load, inc, sub, offset[8], c[8], pc[8]); pc\_slice pc\_slice\_9 (clk, reset, c[8], load, inc, sub, offset[9], c[9], pc[9]); pc\_slice pc\_slice\_10 (clk, reset, c[9], load, inc, sub, offset[10], c[10], pc[10]); pc\_slice pc\_slice\_11 (clk, reset, c[10], load, inc, sub, offset[11], c[11], pc[11]); pc\_slice pc\_slice\_12 (clk, reset, c[11], load, inc, sub, offset[12], c[12], pc[12]); pc\_slice pc\_slice\_13 (clk, reset, c[12], load, inc, sub, offset[13], c[13], pc[13]); pc\_slice pc\_slice\_14 (clk, reset, c[13], load, inc, sub, offset[14], c[14], pc[14]); pc\_slice pc\_slice\_15 (clk, reset, c[14], load, inc, sub, offset[15], c[15], pc[15]); endmodule

reg\_alu.v:

module dfrl\_16 (input wire clk, reset, load, input wire [0:15] in, output wire [0:15] out); dfrl dfrl\_0(clk, reset, load, in[0], out[0]); dfrl dfrl\_1(clk, reset, load, in[1], out[1]); dfrl dfrl\_2(clk, reset, load, in[2], out[2]); dfrl dfrl\_3(clk, reset, load, in[3], out[3]); dfrl dfrl\_4(clk, reset, load, in[4], out[4]); dfrl dfrl\_5(clk, reset, load, in[5], out[5]); dfrl dfrl\_6(clk, reset, load, in[6], out[6]); dfrl dfrl\_7(clk, reset, load, in[7], out[7]); dfrl dfrl\_8(clk, reset, load, in[8], out[8]); dfrl dfrl\_9(clk, reset, load, in[9], out[9]); dfrl dfrl\_10(clk, reset, load, in[10], out[10]); dfrl dfrl\_11(clk, reset, load, in[11], out[11]); dfrl dfrl\_12(clk, reset, load, in[12], out[12]); dfrl dfrl\_13(clk, reset, load, in[13], out[13]); dfrl dfrl\_14(clk, reset, load, in[14], out[14]); dfrl dfrl\_15(clk, reset, load, in[15], out[15]);

endmodule

module mux2\_16 (input wire [15:0] i0, i1, input wire j, output wire [15:0] o); mux2 mux2\_0 (i0[0], i1[0], j, o[0]); mux2 mux2\_1 (i0[1], i1[1], j, o[1]); mux2 mux2\_2 (i0[2], i1[2], j, o[2]); mux2 mux2\_3 (i0[3], i1[3], j, o[3]); mux2 mux2\_4 (i0[4], i1[4], j, o[4]); mux2 mux2\_5 (i0[5], i1[5], j, o[5]); mux2 mux2\_6 (i0[6], i1[6], j, o[6]); mux2 mux2\_7 (i0[7], i1[7], j, o[7]); mux2 mux2\_8 (i0[8], i1[8], j, o[8]); mux2 mux2\_9 (i0[9], i1[9], j, o[9]); mux2 mux2\_10 (i0[10], i1[10], j, o[10]); mux2 mux2\_11 (i0[11], i1[11], j, o[11]); mux2 mux2\_12 (i0[12], i1[12], j, o[12]); mux2 mux2\_13 (i0[13], i1[13], j, o[13]); mux2 mux2\_14 (i0[14], i1[14], j, o[14]); mux2 mux2\_15 (i0[15], i1[15], j, o[15]); endmodule

module mux8\_16 (input wire [0:15] i0, i1, i2, i3, i4, i5, i6, i7, input wire [0:2] j, output wire [0:15] o); mux8 mux8\_0({i0[0], i1[0], i2[0], i3[0], i4[0], i5[0], i6[0], i7[0]}, j[0], j[1], j[2], o[0]); mux8 mux8\_1({i0[1], i1[1], i2[1], i3[1], i4[1], i5[1], i6[1], i7[1]}, j[0], j[1], j[2], o[1]); mux8 mux8\_2({i0[2], i1[2], i2[2], i3[2], i4[2], i5[2], i6[2], i7[2]}, j[0], j[1], j[2], o[2]); mux8 mux8\_3({i0[3], i1[3], i2[3], i3[3], i4[3], i5[3], i6[3], i7[3]}, j[0], j[1], j[2], o[3]); mux8 mux8\_4({i0[4], i1[4], i2[4], i3[4], i4[4], i5[4], i6[4], i7[4]}, j[0], j[1], j[2], o[4]); mux8 mux8\_5({i0[5], i1[5], i2[5], i3[5], i4[5], i5[5], i6[5], i7[5]}, j[0], j[1], j[2], o[5]); mux8 mux8\_6({i0[6], i1[6], i2[6], i3[6], i4[6], i5[6], i6[6], i7[6]}, j[0], j[1], j[2], o[6]); mux8 mux8\_7({i0[7], i1[7], i2[7], i3[7], i4[7], i5[7], i6[7], i7[7]}, j[0], j[1], j[2], o[7]); mux8 mux8\_8({i0[8], i1[8], i2[8], i3[8], i4[8], i5[8], i6[8], i7[8]}, j[0], j[1], j[2], o[8]); mux8 mux8\_9({i0[9], i1[9], i2[9], i3[9], i4[9], i5[9], i6[9], i7[9]}, j[0], j[1], j[2], o[9]); mux8 mux8\_10({i0[10], i1[10], i2[10], i3[10], i4[10], i5[10], i6[10], i7[10]}, j[0], j[1], j[2], o[10]); mux8 mux8\_11({i0[11], i1[11], i2[11], i3[11], i4[11], i5[11], i6[11], i7[11]}, j[0], j[1], j[2], o[11]); mux8 mux8\_12({i0[12], i1[12], i2[12], i3[12], i4[12], i5[12], i6[12], i7[12]}, j[0], j[1], j[2], o[12]); mux8 mux8\_13({i0[13], i1[13], i2[13], i3[13], i4[13], i5[13], i6[13], i7[13]}, j[0], j[1], j[2], o[13]); mux8 mux8\_14({i0[14], i1[14], i2[14], i3[14], i4[14], i5[14], i6[14], i7[14]}, j[0], j[1], j[2], o[14]);

mux8 mux8\_15({i0[15], i1[15], i2[15], i3[15], i4[15], i5[15], i6[15], i7[15]}, j[0], j[1], j[2], o[15]);

endmodule

module reg\_file (input wire clk, reset, wr, input wire [0:2] rd\_addr\_a, rd\_addr\_b, wr\_addr, input wire [0:15] d\_in, output wire [0:15] d\_out\_a, d\_out\_b); wire [0:7] load; wire [0:15] dout\_0, dout\_1, dout\_2, dout\_3, dout\_4, dout\_5, dout\_6, dout\_7; dfrl\_16 dfrl\_16\_0(clk, reset, load[0], d\_in, dout\_0); dfrl\_16 dfrl\_16\_1(clk, reset, load[1], d\_in, dout\_1); dfrl\_16 dfrl\_16\_2(clk, reset, load[2], d\_in, dout\_2); dfrl\_16 dfrl\_16\_3(clk, reset, load[3], d\_in, dout\_3); dfrl\_16 dfrl\_16\_4(clk, reset, load[4], d\_in, dout\_4); dfrl\_16 dfrl\_16\_5(clk, reset, load[5], d\_in, dout\_5); dfrl\_16 dfrl\_16\_6(clk, reset, load[6], d\_in, dout\_6); dfrl\_16 dfrl\_16\_7(clk, reset, load[7], d\_in, dout\_7); demux8 demux8\_0(wr, wr\_addr[2], wr\_addr[1], wr\_addr[0], load); mux8\_16 mux8\_16\_9(dout\_0, dout\_1, dout\_2, dout\_3, dout\_4, dout\_5, dout\_6, dout\_7, rd\_addr\_a, d\_out\_a); mux8\_16 mux8\_16\_10(dout\_0, dout\_1, dout\_2, dout\_3, dout\_4, dout\_5, dout\_6, dout\_7, rd\_addr\_b, d\_out\_b); endmodule

module reg\_alu (input wire clk, reset, sel, wr, /\*add\_sub,\*/ input wire [1:0] op, input wire [2:0] rd\_addr\_a, rd\_addr\_b, wr\_addr, input wire [15:0] d\_in, output wire [15:0] d\_out\_a, d\_out\_b, output wire cout); wire [15:0] d\_in\_alu, d\_in\_reg, pc\_in, pc\_out, alu\_in\_a, alu\_in\_b; wire cout\_0; reg\_file reg\_file\_0 (clk, reset, wr, rd\_addr\_a, rd\_addr\_b, wr\_addr, d\_in\_reg, alu\_in\_a, alu\_in\_b); mux2\_16 mux2\_16\_0 (d\_in, d\_in\_alu, sel, d\_in\_reg); alu alu\_0 (op, alu\_in\_a, alu\_in\_b, d\_in\_alu, cout\_0); dfrl dfrl\_0 (clk, reset, /\*add\_sub\*/1'b1, cout\_0, cout); endmodule

tb\_mproc\_mem.v:

`timescale 1 ns / 100 ps

`define TESTVECS 500

module tb;

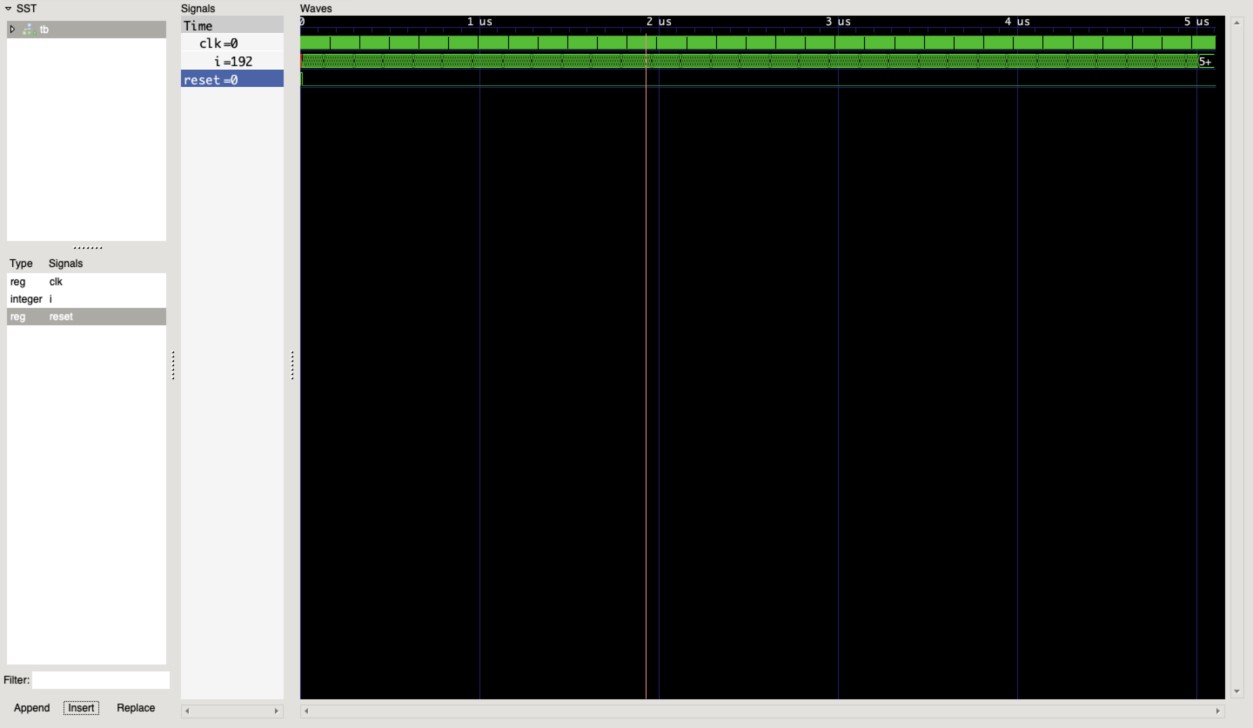
reg clk, reset; integer i; initial begin $dumpfile("tb\_mproc\_mem.vcd"); $dumpvars(0,tb); end initial begin reset = 1'b1; #12.5 reset = 1'b0; end initial clk = 1'b0; always #5 clk =~ clk; mproc\_mem mproc\_mem\_0 (clk, reset);

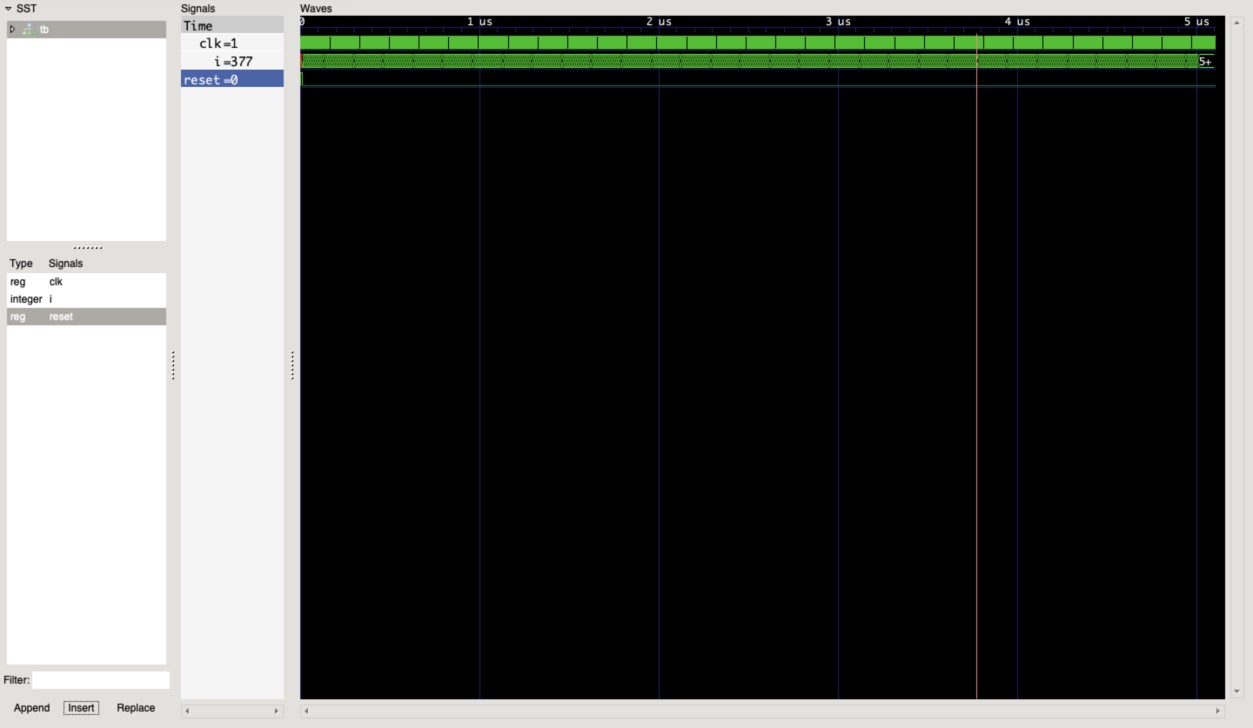
initial begin

#6 for(i=0;i<`TESTVECS;i=i+1) begin #10; end #100 $finish; end

endmodule







**ASSIGNMENT 4B**

**Digital Design and Computer Organization**

**UE20CS251A**

**3rd Semester, Academic Year 2021-22**

Date: 14/11/22

|  |  |  |
| --- | --- | --- |
| Name:  Nikhil Girish | SRN:  PES2UG21CS334 | Section: F |

**Question:**

Convert the following program from machine language into MIPS assembly language. The numbers on the left are the instruction addresses in memory, and the numbers on the right give the instruction at that address. Then reverse engineer a high-level program that would compile into this assembly language routine and write it. Explain in words what the program does. $a0 is the input, and it initially contains a positive number, n. $v0 is the output.

0x00400000 0x20080000 0x00400004 0x20090001

0x00400008 0x0089502A

0x0040000C 0x15400003

0x00400010 0x01094020 0x00400014 0x21290002 0x00400018 0x08100002

0x0040001C 0x01001020

0x00400020 0x03E00008

**Answer:**

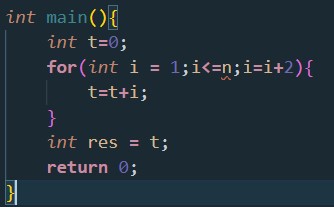
After converting machine language to MIPS assembly language:

|  |  |  |  |
| --- | --- | --- | --- |
| [0x00400000] 0x20080000 [0x00400004] 0x20090001  [0x00400008] 0x0089502A |  | for: | addi $t0, $0, 0 addi $t1, $0, 1 slt $t2, $a0, $t1 |
| [0x0040000C] 0x15400003  [0x00400010] 0x01094020 |  |  | bne $t2, $0, done  add $t0, $t0, $t1 |
| [0x00400014] 0x21290002  [0x00400018] 0x08100002 |  |  | addi $t1, $t1, 2  j for |
| [0x0040001C] 0x01001020 |  | done: add $v0, $t0, $0 | |

Last line [0x00400020] 0x03E00008, is R-type jump instruction to register 11111 (31) ($ra) which signifies return statement.

**C program:**

t = $t0, i = $t1, n = $a0, res = $v0



The program will sum up all the odd integers from 1 to $a0 and stores the final sum in $v0.

<https://youtu.be/c2_LwMuPs6U> Refer this link

Name: Nikhil Girish

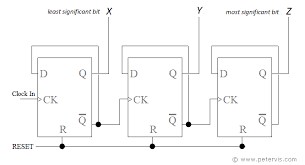
SRN: PES2UG21CS334

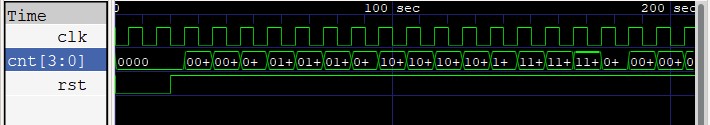
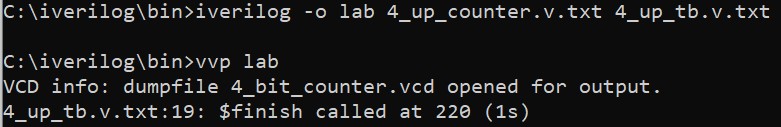
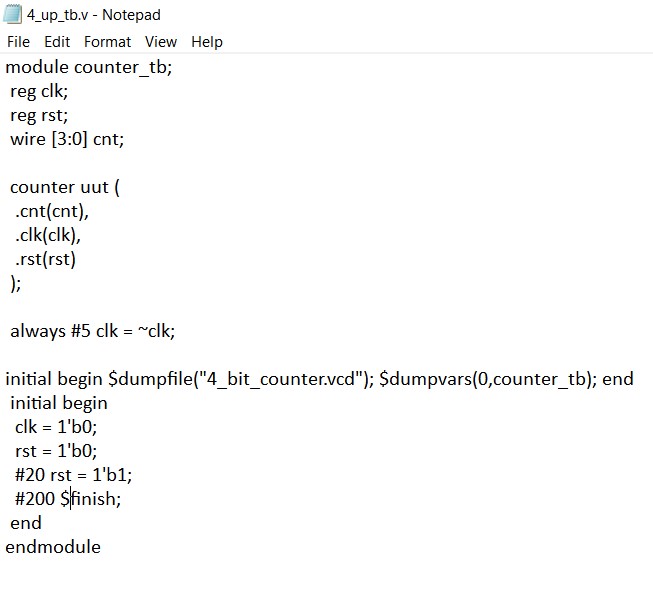
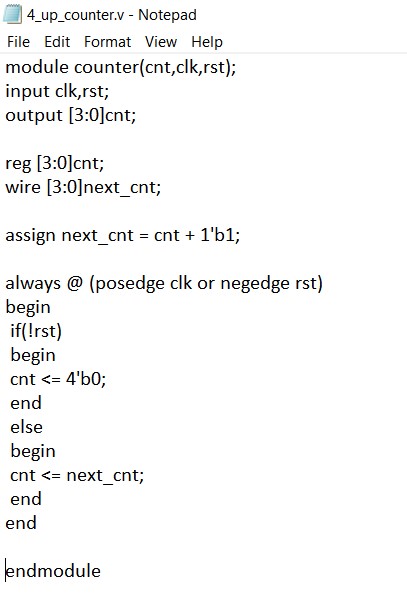
Section: F

Date: 21/10/22

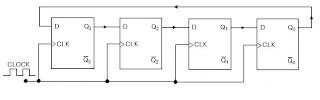
Assignment-3: Counters with D flipflop

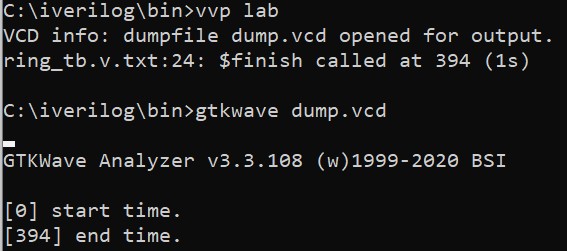
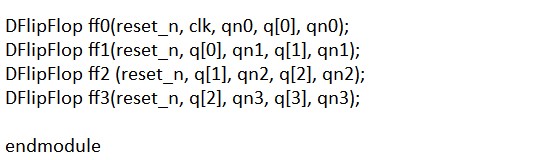
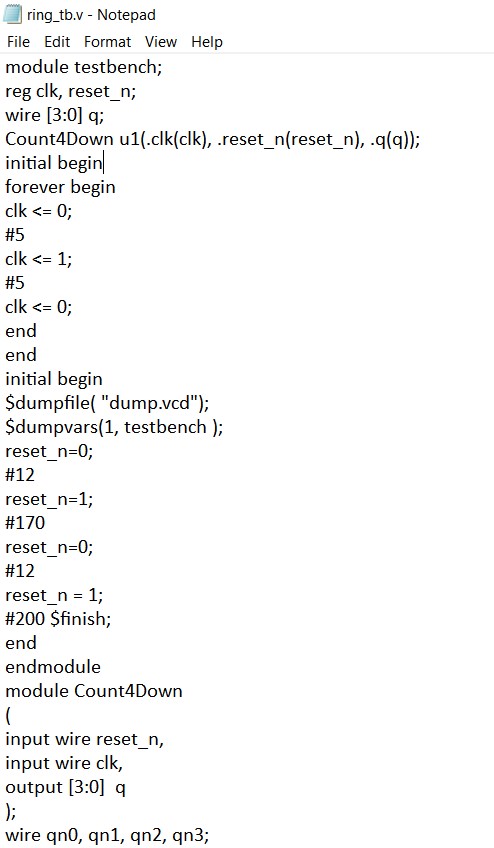
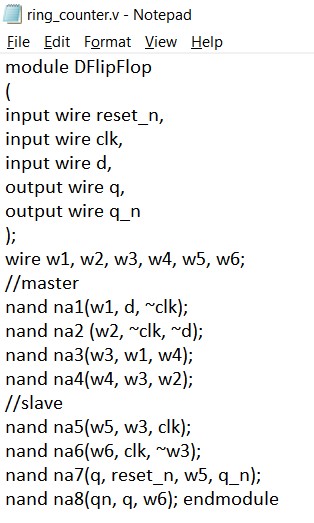
Write a Verilog code and test bench for 4 bit Binary up counter designed using D Flip Flops

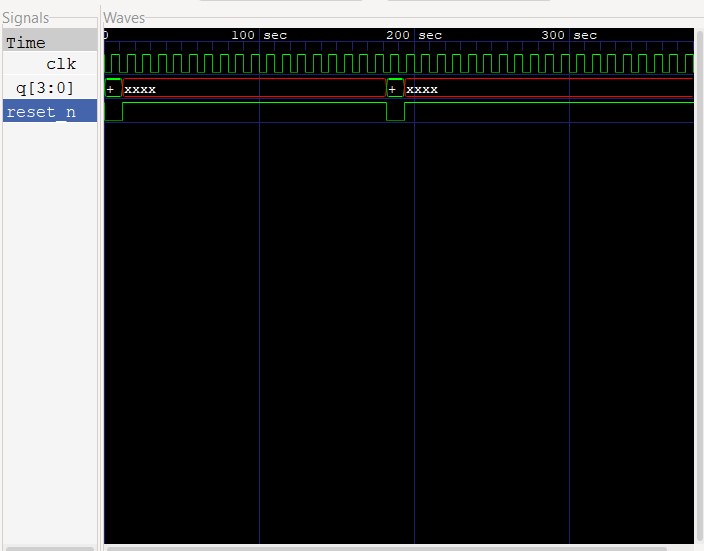




Write a Verilog code and test bench for 4-bit Ring Counter by D flipflop







Write a Verilog code and test bench for 4-bit Johnson Counter by D flipflop

