Digital Design and Computer Organization Laboratory

UE19CS206

3rd Semester, Academic Year 2020-21

Date:

Name : Mahika Gupta	SRN : PES1UG20CS243	Section : D

Experiment Number: 6 Week # : 6

<u>Title of the Program: 16-BIT PROGRAM COUNTER</u>

Code:

1.pc.v:

```
// Write code for modules you need here

module pc (input wire clk, reset, inc, add, sub, input wire [15:0] offset, output wire [15:0] pc);

// Declare wires here
wire [15:0]cout;

// Instantiate modules here
```

```
slice_0 slice_00(offset[0], inc, sub,add, pc[0],cout[0]);

slice_1 silce_01(offset[1], inc, cout[0],add,pc[1],cout[1]);

slice_1 silce_02(offset[2], inc, cout[1],add,pc[2],cout[2]);

slice_1 silce_03(offset[3], inc, cout[2],add,pc[3],cout[3]);

slice_1 silce_04(offset[4], inc, cout[3],add,pc[4],cout[4]);

slice_1 silce_05(offset[5], inc, cout[4],add,pc[5],cout[5]);

slice_1 silce_06(offset[6], inc, cout[5],add,pc[6],cout[6]);

slice_1 silce_07(offset[7], inc, cout[6],add,pc[7],cout[7]);

slice_1 silce_08(offset[8], inc, cout[7],add,pc[8],cout[8]);

slice_1 silce_09(offset[9], inc, cout[8],add,pc[9],cout[9]);

slice_1 silce_01(offset[10], inc, cout[9],add,pc[10],cout[10]);

slice_1 silce_01(offset[11], inc, cout[10],add,pc[11],cout[11]);

slice_1 silce_01(offset[12], inc, cout[11],add,pc[12],cout[12]);

slice_1 silce_01(offset[13], inc, cout[12]],add,pc[13],cout[13]);

slice_1 silce_01(offset[14], inc, cout[13],add,pc[14],cout[14]);

slice_1 silce_01(offset[15], inc, cout[14],add,pc[15],cout[15]);



endmodule



module addsub(input wire sub ,pc,i,cin, output wire sum,cout);   //i - (offset+inc)   cin-(sub)


wire t;

xor2 xor_0(sub,i,t);

full_adder fa_0(t,pc,sub,sum,cout);

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 slice_0(input wire offset, inc, sub,add, output wire pc,cout);


wire load,t_or,t_as;


or2 or2_0(offset,inc,t_or);

or3 or3_0(inc,sub,add,load);

//xor2 xor2_0(t_or,sub,t_xor);

addsub addsub_0(sub,pc,t_or,sub,t_as,cout);

dfrl dfrl_0(clk,reset,load,pc);


endmodule


module slice_1(input wire offset, inc, sub,add, output wire pc,cout);


wire load,t_and,t_as,t_inc;


invert invert_0(inc,t_inc);

and2 and2_0(offset,t_inc,t_and);

or3 or3_0(inc,sub,add,load);


addsub addsub_0(sub,pc,t_and,sub,t_as,cout);

dfrl dfrl_0(clk,reset,load,pc);


endmodule
```

2.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
```

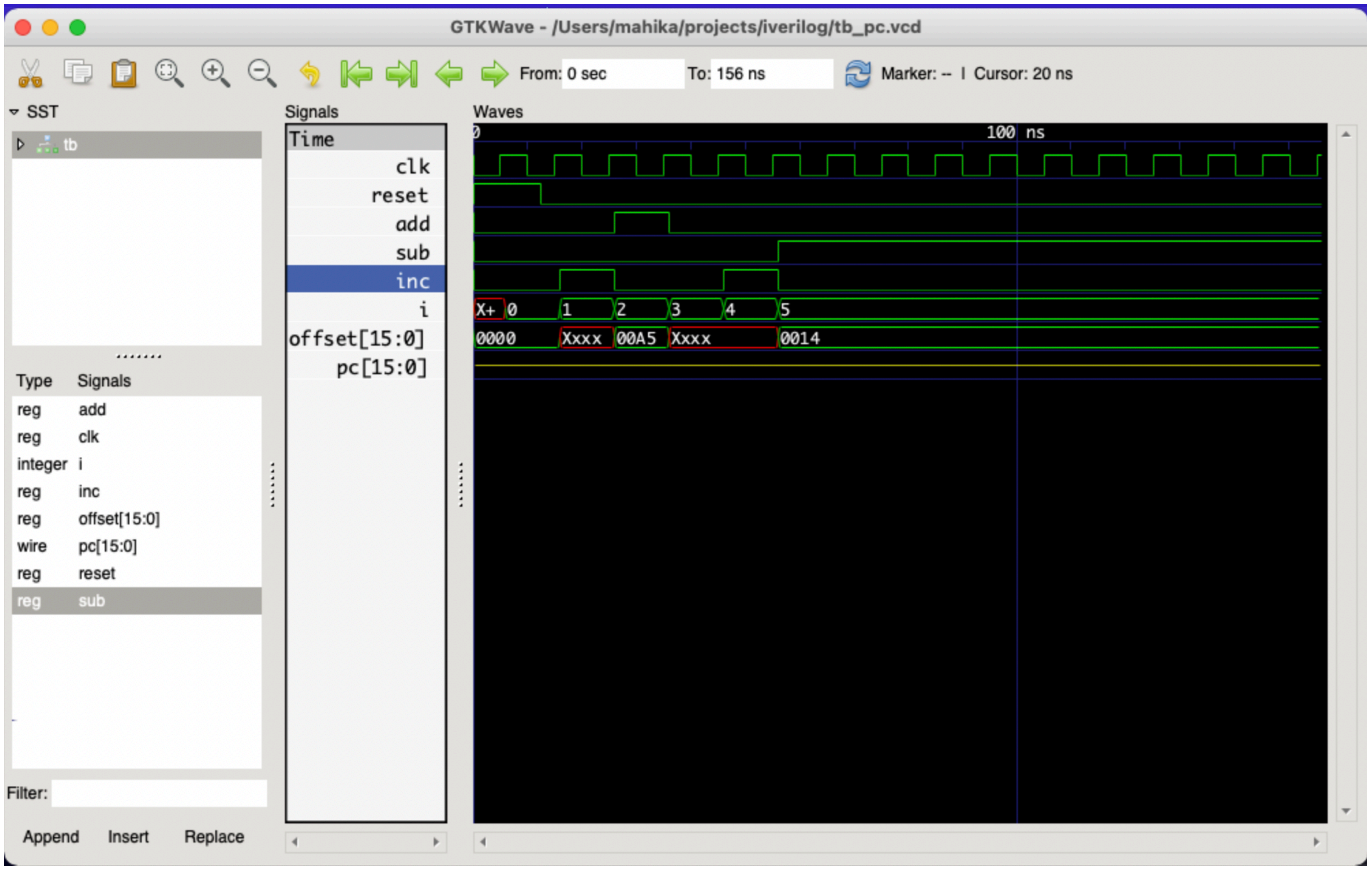
```
module xor3 (input wire i0, i1, i2, output wire o);
   wire t;
   xor2 xor2_0 (i0, i1, t);
   xor2 xor2_1 (i2, t, o);
endmodule

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 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
```

3.tb_pc.v

```
`timescale 1 ns / 100 ps
`define TESTVECS 5

module tb;
  reg clk, reset, inc, add, sub;
  reg [15:0] offset;
  wire [15:0] pc;
  reg [18:0] test_vecs [0:(`TESTVECS-1)];
  integer i;
  initial begin $dumpfile("tb_pc.vcd"); $dumpvars(0,tb); end
  initial begin reset = 1'b1; #12.5 reset = 1'b0; end
  initial clk = 1'b0; always #5 clk =~ clk;
  initial begin
    test_vecs[0][18] = 1'b1; test_vecs[0][17] = 1'b0; test_vecs[0][16] = 1'b0;
    test_vecs[0][15:0] = 15'hxx;
    test_vecs[1][18] = 1'b0; test_vecs[1][17] = 1'b1; test_vecs[1][16] = 1'b0;
    test_vecs[1][15:0] = 15'ha5;
    test_vecs[2][18] = 1'b0; test_vecs[2][17] = 1'b0; test_vecs[2][16] = 1'b0;
    test_vecs[2][15:0] = 15'hxx;
    test_vecs[3][18] = 1'b1; test_vecs[3][17] = 1'b0; test_vecs[3][16] = 1'b0;
    test_vecs[3][15:0] = 15'hxx;
    test_vecs[4][18] = 1'b0; test_vecs[4][17] = 1'b0; test_vecs[4][16] = 1'b1;
    test_vecs[4][15:0] = 15'h14;
  end
  initial {inc, add, sub, offset} = 0;
  pc pc_0 (clk, reset, inc, add, sub, offset, pc);
  initial begin
    #6 for(i=0;i<`TESTVECS;i=i+1)
      begin #10 {inc, add, sub, offset}=test_vecs[i]; end
    #100 $finish;
  end
endmodule
```

Output waveform



Type	Signals
reg	add
reg	clk
integer	i
reg	inc
reg	offset[15:0]
wire	pc[15:0]
reg	reset
reg	sub