# Digital Design and Computer Organisation Laboratory

**UE21CS251A**

# 3rd Semester, Academic Year 2022-23

## Date:

|  |  |  |
| --- | --- | --- |
| **Name:** Pranav Ajay Desai | **SRN:** PES2UG21CS385 | **Section:**  G |

Week# 7

## Program Number: 1

## TITLE:

**WRITE A VERILOG PROGRAM TO MODEL Control Logic .**

**GENERATE THE VVP OUTPUT AND SIMULATION WAVEFORM USING**

**GTKWAVE. VERIFY THE OUTPUT AND WAVEFORM WITH THE TRUTH TABLE**

## Verilog Code Screenshot

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

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

## alu.v

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

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

## 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'o000100; ram[1] = 16'o001201; ram[2] = 16'o002321; ram[3] = 16'o003432;

end

always @(wr) ram[addr] = din; assign dout = ram[addr];

endmodule

module mproc\_mem ( input wire clk, reset

);

wire [15:0] addr;

wire [15:0] ins;

ram\_128\_16 ram\_128\_16\_0 ( clk,

reset, 1'b0,

addr[6:0], 16'b0,

ins

);

mproc mproc\_0 ( clk, reset, ins,

addr

);

endmodule

mproc.v

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 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 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 pc\_inc, load\_ir,

wr\_reg

);

wire t, alu\_ins;

dfsl fetch ( clk, reset, 1'b1,

wr\_reg, pc\_inc

);

assign load\_ir = pc\_inc;

dfrl dec\_exec ( clk, reset, 1'b1,

load\_ir, t

);

nor5 nor5\_0 (

cur\_ins[15:11], alu\_ins

);

and2 and2\_0 (

t, alu\_ins, wr\_reg

);

assign rd\_addr\_a = cur\_ins[2:0];

assign rd\_addr\_b = cur\_ins[5:3];

assign wr\_addr = cur\_ins[8:6];

assign op = cur\_ins[10:9];

endmodule

module mproc ( input wire clk, reset,

input wire [15:0] ins, output wire [15:0] addr

);

wire pc\_inc, cout;

wire [2:0] rd\_addr\_a, rd\_addr\_b, wr\_addr; wire [1:0] op;

wire [15:0] cur\_ins, d\_out\_a, d\_out\_b;

pc pc\_0 (

clk, reset, pc\_inc, 1'b0,

1'b0,

16'b0,

addr

);

ir ir\_0 (

clk, reset, load\_ir, ins, cur\_ins

);

control\_logic control\_logic\_0 ( clk,

reset, cur\_ins,

rd\_addr\_a, rd\_addr\_b, wr\_addr, op, pc\_inc, load\_ir, wr\_reg

);

reg\_alu reg\_alu\_0 ( clk,

reset, 1'b1,

wr\_reg, op, rd\_addr\_a, rd\_addr\_b, wr\_addr, 16'b0,

d\_out\_a, d\_out\_b, cout

);

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 reg\_file\_2\_1 ( input wire clk, reset,

wr, rd\_addr\_a, rd\_addr\_b, wr\_addr, d\_in,

output wire d\_out\_a, d\_out\_b

);

wire l0, l1, o0, o1; dfrl dfrl\_0 (

clk, reset, l0,

d\_in,

o0

);

dfrl dfrl\_1 ( clk, reset, l1, d\_in,

o1

);

mux2 mux2\_a (

o0,

o1, rd\_addr\_a, d\_out\_a

);

mux2 mux2\_b (

o0,

o1, rd\_addr\_b, d\_out\_b

);

demux2 demux2\_0 ( wr,

wr\_addr, l0,

l1

);

endmodule

module \_reg\_file\_2\_1 ( input wire clk, reset,

wr, rd\_addr\_a, rd\_addr\_b, wr\_addr, d\_in,

output wire d\_out\_a, d\_out\_b

);

wire l0, l1, o0, o1; dfsl dfsl\_0 (

clk, reset, l0, d\_in, o0

);

dfrl dfrl\_1 ( clk, reset, l1,

d\_in, o1

);

mux2 mux2\_a (

o0,

o1, rd\_addr\_a, d\_out\_a

);

mux2 mux2\_b (

o0,

o1, rd\_addr\_b, d\_out\_b

);

demux2 demux2\_0 ( wr,

wr\_addr, l0,

l1

);

endmodule

module \_reg\_file\_4\_1 ( input wire clk, reset,

wr,

input wire [1:0] rd\_addr\_a, rd\_addr\_b,

wr\_addr,

input wire d\_in, output wire d\_out\_a, d\_out\_b

);

wire wr0, wr1, o0\_a, o0\_b, o1\_a, o1\_b;

\_reg\_file\_2\_1 reg\_file\_2\_1\_0 ( clk,

reset, wr0,

rd\_addr\_a[0], rd\_addr\_b[0], wr\_addr[0], d\_in,

o0\_a, o0\_b

);

reg\_file\_2\_1 reg\_file\_2\_1\_1 ( clk,

reset, wr1,

rd\_addr\_a[0], rd\_addr\_b[0], wr\_addr[0], d\_in,

o1\_a, o1\_b

);

mux2 mux2\_a ( o0\_a, o1\_a,

rd\_addr\_a[1], d\_out\_a

);

mux2 mux2\_b ( o0\_b,

o1\_b, rd\_addr\_b[1], d\_out\_b

);

demux2 demux2\_0 ( wr, wr\_addr[1], wr0,

wr1

);

endmodule

module reg\_file\_4\_1 ( input wire clk, reset,

wr,

input wire [1:0] rd\_addr\_a, rd\_addr\_b,

wr\_addr,

input wire d\_in, output wire d\_out\_a, d\_out\_b

);

wire wr0, wr1, o0\_a, o0\_b, o1\_a, o1\_b; reg\_file\_2\_1 reg\_file\_2\_1\_0 (

clk, reset, wr0,

rd\_addr\_a[0], rd\_addr\_b[0], wr\_addr[0], d\_in,

o0\_a, o0\_b

);

reg\_file\_2\_1 reg\_file\_2\_1\_1 ( clk,

reset, wr1,

rd\_addr\_a[0], rd\_addr\_b[0], wr\_addr[0], d\_in,

o1\_a, o1\_b

);

mux2 mux2\_a ( o0\_a, o1\_a,

rd\_addr\_a[1], d\_out\_a

);

mux2 mux2\_b ( o0\_b, o1\_b,

rd\_addr\_b[1], d\_out\_b

);

demux2 demux2\_0 ( wr, wr\_addr[1], wr0,

wr1

);

endmodule

module reg\_file\_8\_1 ( input wire clk, reset,

wr,

input wire [2:0] rd\_addr\_a, rd\_addr\_b,

wr\_addr,

input wire d\_in, output wire d\_out\_a, d\_out\_b

);

wire wr0, wr1, o0\_a, o0\_b, o1\_a, o1\_b;

\_reg\_file\_4\_1 reg\_file\_4\_1\_0 ( clk,

reset, wr0,

rd\_addr\_a[1:0], rd\_addr\_b[1:0], wr\_addr[1:0], d\_in,

o0\_a, o0\_b

);

reg\_file\_4\_1 reg\_file\_4\_1\_1 ( clk,

reset, wr1,

rd\_addr\_a[1:0], rd\_addr\_b[1:0], wr\_addr[1:0], d\_in,

o1\_a, o1\_b

);

mux2 mux2\_a ( o0\_a, o1\_a,

rd\_addr\_a[2], d\_out\_a

);

mux2 mux2\_b ( o0\_b, o1\_b,

rd\_addr\_b[2], d\_out\_b

);

demux2 demux2\_0 ( wr, wr\_addr[2], wr0,

wr1

);

endmodule

module reg\_file\_8\_4 ( input wire clk, reset,

wr,

input wire [2:0] rd\_addr\_a, rd\_addr\_b,

wr\_addr,

input wire [3:0] d\_in, output wire [3:0] d\_out\_a, d\_out\_b

);

reg\_file\_8\_1 reg\_file\_8\_1\_0 ( clk,

reset, wr,

rd\_addr\_a, rd\_addr\_b, wr\_addr, d\_in[0], d\_out\_a[0], d\_out\_b[0]

);

reg\_file\_8\_1 reg\_file\_8\_1\_1 ( clk,

reset, wr,

rd\_addr\_a, rd\_addr\_b, wr\_addr, d\_in[1], d\_out\_a[1], d\_out\_b[1]

);

reg\_file\_8\_1 reg\_file\_8\_1\_2 ( clk,

reset,

wr, rd\_addr\_a, rd\_addr\_b, wr\_addr, d\_in[2], d\_out\_a[2], d\_out\_b[2]

);

reg\_file\_8\_1 reg\_file\_8\_1\_3 ( clk,

reset, wr,

rd\_addr\_a, rd\_addr\_b, wr\_addr, d\_in[3], d\_out\_a[3], d\_out\_b[3]

);

endmodule

module reg\_file ( input wire clk, reset,

wr,

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

);

reg\_file\_8\_4 reg\_file\_8\_4\_0 ( clk,

reset, wr,

rd\_addr\_a, rd\_addr\_b, wr\_addr,

d\_in[3:0],

d\_out\_a[3:0], d\_out\_b[3:0]

);

reg\_file\_8\_4 reg\_file\_8\_4\_1 ( clk,

reset, wr,

rd\_addr\_a, rd\_addr\_b, wr\_addr, d\_in[7:4],

d\_out\_a[7:4], d\_out\_b[7:4]

);

reg\_file\_8\_4 reg\_file\_8\_4\_2 ( clk,

reset, wr,

rd\_addr\_a, rd\_addr\_b, wr\_addr, d\_in[11:8], d\_out\_a[11:8], d\_out\_b[11:8]

);

reg\_file\_8\_4 reg\_file\_8\_4\_3 ( clk,

reset, wr,

rd\_addr\_a, rd\_addr\_b, wr\_addr, d\_in[15:12], d\_out\_a[15:12], d\_out\_b[15:12]

);

endmodule

module mux2\_4 (

input wire [3:0] i0, i1,

input wire j, output wire [3:0] 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]

);

endmodule

module mux2\_16 (

input wire [15:0] i0, i1,

input wire j,

output wire [15:0] o

);

mux2\_4 mux2\_4\_0 (

i0[3:0],

i1[3:0],

j, o[3:0]

);

mux2\_4 mux2\_4\_1 ( i0[7:4],

i1[7:4],

j, o[7:4]

);

mux2\_4 mux2\_4\_2 ( i0[11:8],

i1[11:8],

j, o[11:8]

);

mux2\_4 mux2\_4\_3 ( i0[15:12],

i1[15:12],

j, o[15:12]

);

endmodule

module reg\_alu ( input wire clk, reset,

sel, wr,

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; wire cout\_0;

alu alu\_0 ( op, d\_out\_a, d\_out\_b, d\_in\_alu, cout\_0

);

reg\_file reg\_file\_0 ( clk,

reset, wr,

rd\_addr\_a, rd\_addr\_b, wr\_addr, d\_in\_reg, d\_out\_a, d\_out\_b

);

mux2\_16 mux2\_16\_0 ( d\_in,

d\_in\_alu, sel, d\_in\_reg

);

dfr dfr\_0 ( clk, reset, cout\_0, cout

);

endmodule

## tb\_mproc\_mem.v

`timescale 1 ns / 100 ps

`define TESTVECS 4

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

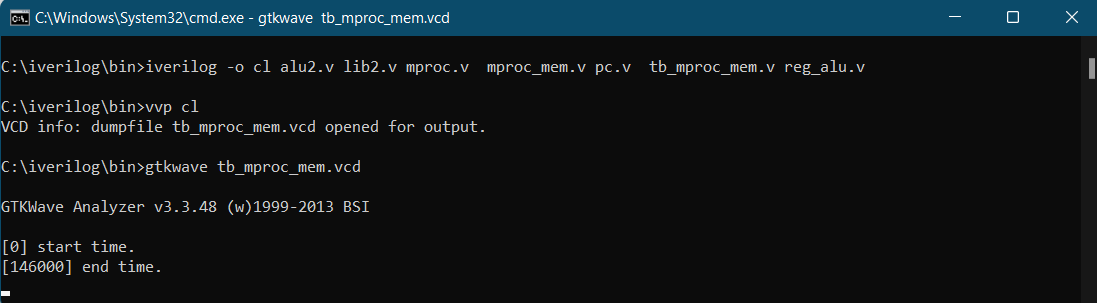
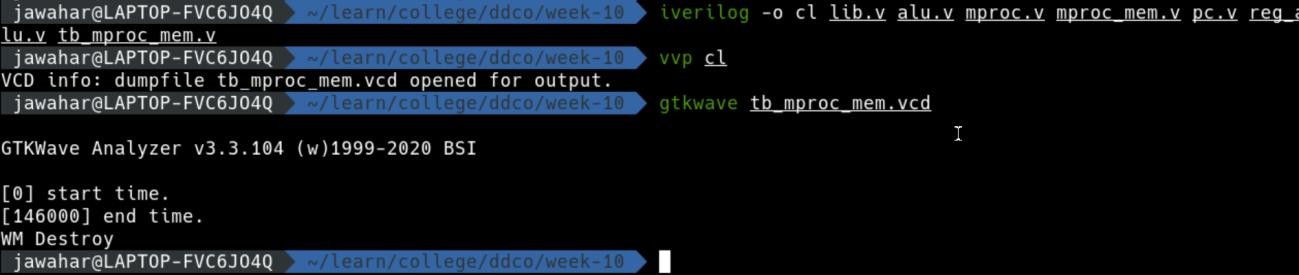
for (i = 0; i < `TESTVECS; i = i + 1) begin #10;

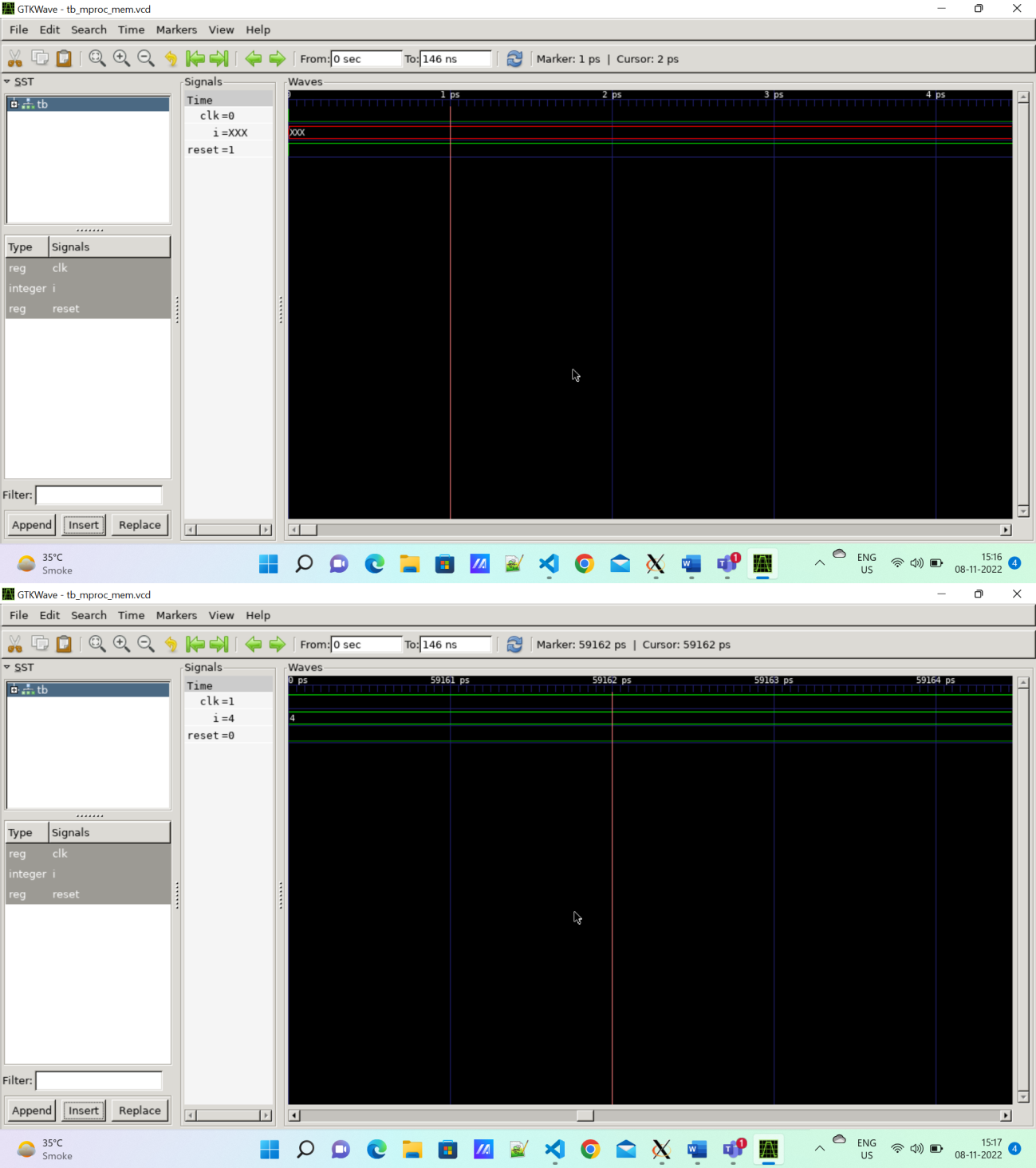
end

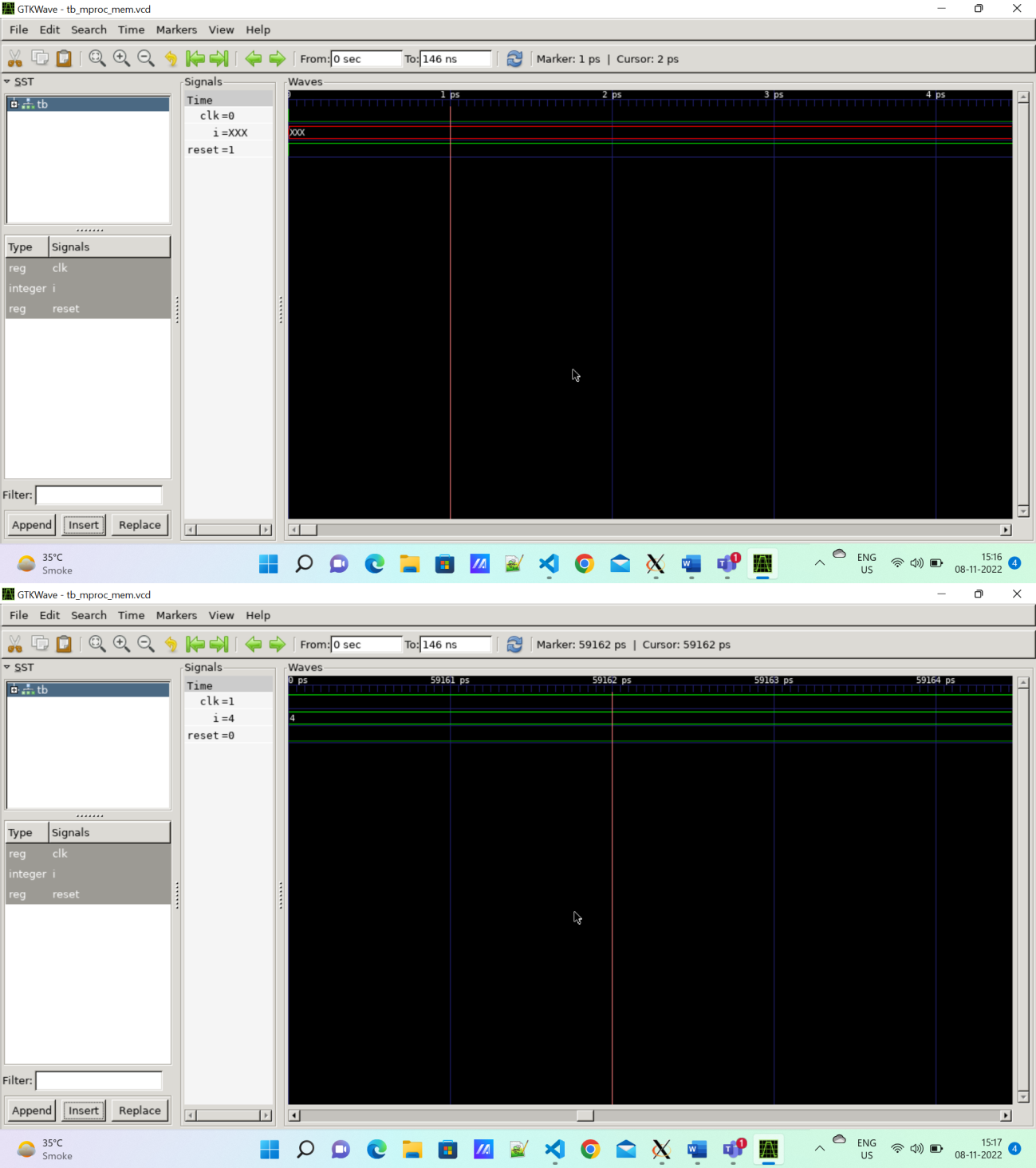
#100 $finish;

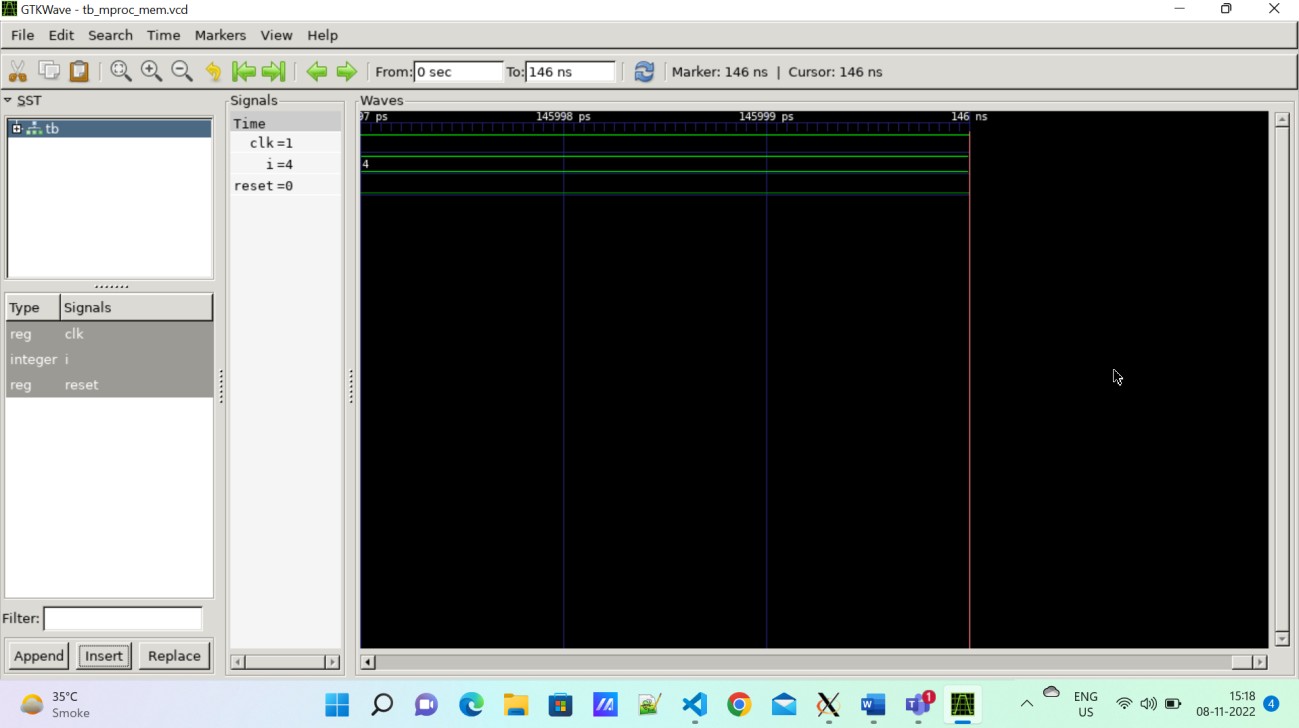
end endmodule

## Verilog VVP Output Screen Shot



1. GTKWAVE Screenshot





xxx

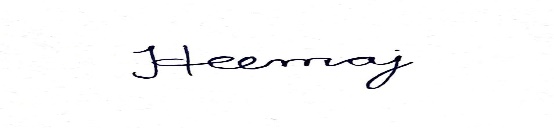
# Disclaimer:

## The programs and output submitted is duly written, verified and executed by me.

* I have not copied from any of my peers nor from the external resource such as internet.

## If found plagiarized, I will abide with the disciplinary action of the University.

Signature:



Name:HEEMAJ YADAV

SRN: PES2UG21CS198

Date:15-11-2022