Connor Cousineau
ECE 3700

Lab 5: CPU

The objective of this lab is to design a very basic CPU unit that will perform 10 instructions in a row. To do this it was broken into 2 units. The finite state machine which controls all the enables and the current instruction line. Followed by the computational unit whose job is to do the instruction tasks by storing and performing the stated tasks. When combined they form a very small CPU that has 4 switches as an input and two buttons which reset and cycle the clock.

One example shown in the demo was to activate the switch that represents the binary value 8. When the clock is cycled it eventually adds 8 + 3 which will display as b. A few more cycles and it will display 4. Finally it displays the number F or 15.Overall testing this particular lab is significantly more difficult to do as the sheer number of inputs to the computational unit and finite state machine are not insignificant. Each unit has an enable and some have a reset.

To ensure that the overall unit would function each unit was tested individually if needed. We have already tested the DFF/seven seg units and the ALU/Tri-buff verilog code was provided for the most part from the book and were not tested alone. There was an attempt to make a testbench for the CPU and FSM but I was unable to get either to work. The failed CPU Test bench is included in the code zone. In the end the way it was tested as a complete unit was to learn about an expected output and then run the code. At first it wasn't correct and I discovered that it had to do with the Registers being assigned wrong. If I had a test bench I could have spotted this. In much bigger projects this error may have been nearly impossible to spot.

There were a couple of complications that I ran into when I had finished combining all the units and no errors were showing. When put on the board it would cycle to the correct ending of F, but the values in the registers were permanently stuck at 0. Finding this bug was fairly involved as none of the units had issues shown in testing. The solution eventually surfaced when I discovered that due to labeling in my CPU unit and the FSM differing ever so slightly, the MUX that passes the value of three into the R2 was somehow connected to R1. Consequently, this meant that R2 was almost never getting the correct value making the whole machine stop functioning.

Everything was straight forward except for the Immediate value in R2, this part on it's own took approximately 25% of the time spent on this lab. It was only when we were told in the lab video to use a mux did it finally piece itself together. The last step was to assign all the input switches and buttons. With the clock set to be on the button, each step was able to be seen for the most part. When a value passes the third register, it's translated into 7 seg and displayed on the board. If the clock were set to the onboard unit, the steps would be almost impossible to see. Overall when combined the units functioned as intended. Even with the few complications the final product was entertaining to play with.

Images and code can be found below

Relevant Images

Failed attempt at CPU testbench

```
`timescale 1ps / 1ps
module tb_CPU;
//reg [3:0]in;
//reg option;
//wire [3:0]out;
reg[3:0]inpSwitch;
reg[1:0]ctrl;
reg T1en, T2en,T3en,T4en,R1en,R2en,R3en,clk,rst,Ictrl;
wire[3:0] Tout1,Tout2,bus,R1out,R2out,R3out,ALUout,Iout;
wire[0:6] seven_seg;
integer i;
//inpSwitch,ctrl, T1en, T2en,T3en,T4en,R1en,R2en,R3en,clk,rst,seven_seg,Ictrl
CPU uut (
.inpSwitch(inpSwitch),
.ctrl(ctrl),
.T1en(T1en),
.T2en(T2en),
.T3en(T3en),
.T4en(T4en),
.R1en(R1en),
.R2en(R2en),
.R3en(R3en),
.R4en(R4en),
.clk(clk),
.rst(rst),
.seven_seg(seven_seg),
.Ictrl(Ictrl)
);

initial begin

for(i = 0; i < 16; i = i+1)begin
        clk = ~clk;
        #9;
        $display("%d => %d option: %d", in, out,option);
        end
```

```
end

endmodule
```

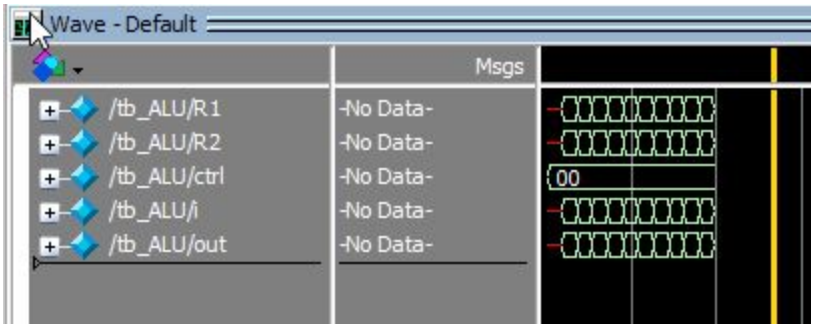
//Buffer code form book assumed it works as intended.

Tb_alu/waveform

```
`timescale 1ps / 1ps
module tb_ALU;
reg [3:0]R1,R2;
reg[1:0]ctrl;
wire [3:0]out;
integer i;

//ctrl, R1, R2, out
ALU4B uut (
.ctrl(ctrl),
.R1(R1),
.R2(R2),
.out(out)
);

initial begin
ctrl=0; #9;


for(i = 0; i < 16; i = i+1)begin
        R1 = i;
        R2 = i+1;
        #9;
        $display("%d => %d out: %d ctrl: %d", R1, R2, out,ctrl);
        end
end
endmodule
```

```
 0 =>  1 out:  1 ctrl: 0
#  1 =>  2 out:  3 ctrl: 0
#  2 =>  3 out:  5 ctrl: 0
```

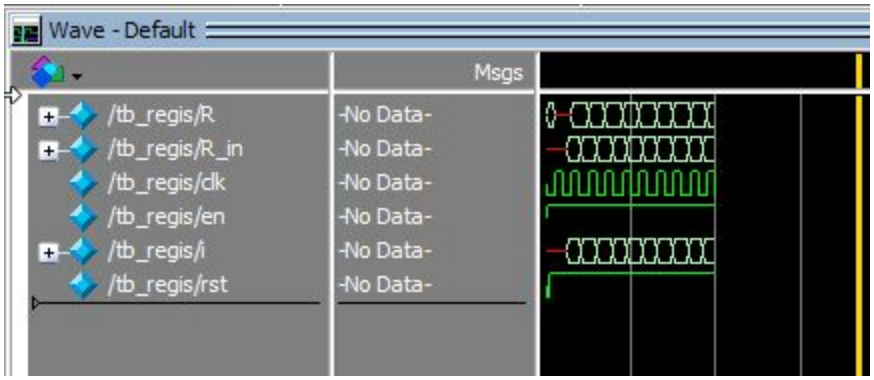
```
#  3 =>  4 out:  7 ctrl: 0
#  4 =>  5 out:  9 ctrl: 0
#  5 =>  6 out: 11 ctrl: 0
#  6 =>  7 out: 13 ctrl: 0
#  7 =>  8 out: 15 ctrl: 0
```



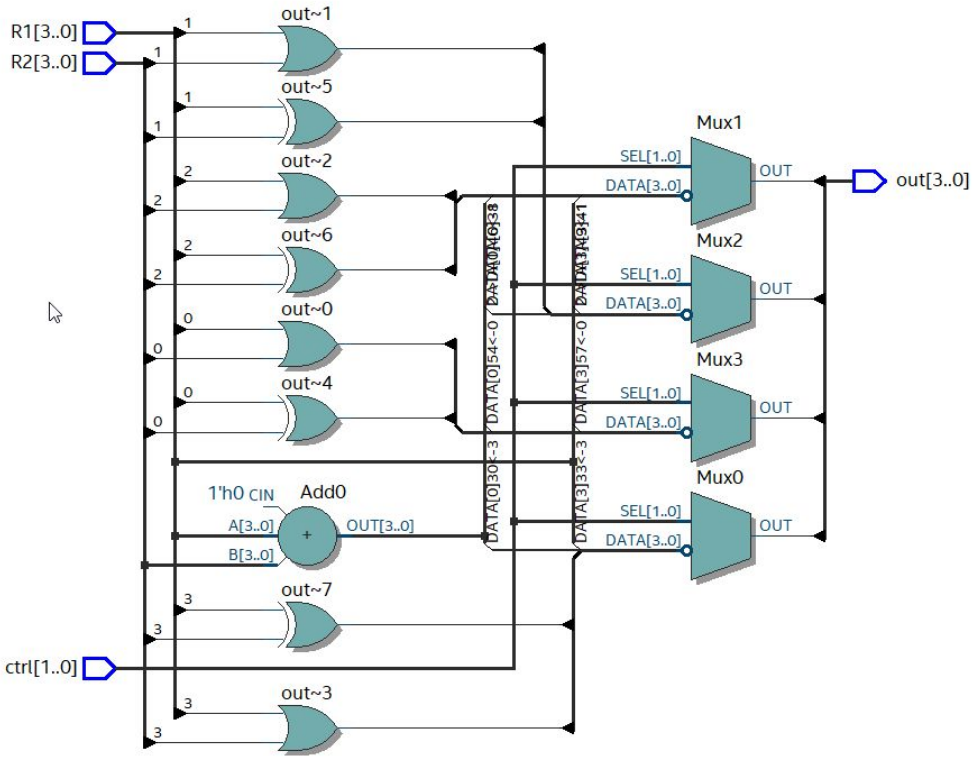
Testbench:regis/waveform

```
`timescale 1ps / 1ps
module tb_regis;
reg [3:0]R_in;
reg clk,rst,en;
wire [3:0]R;
integer i;

//R_in,en,R,clk,rst
regis uut (
.R_in(R_in),
.en(en),
.clk(clk),
.rst(rst),
.R(R)
);

initial begin
//rst = 0; clk=0;#5;
//rst = 1; clk=1;#5;
//rst = 0; clk=0;#5;
clk =0;
en = 1;
rst=0; #2;
rst=1; #5;
rst=1; #5;
```

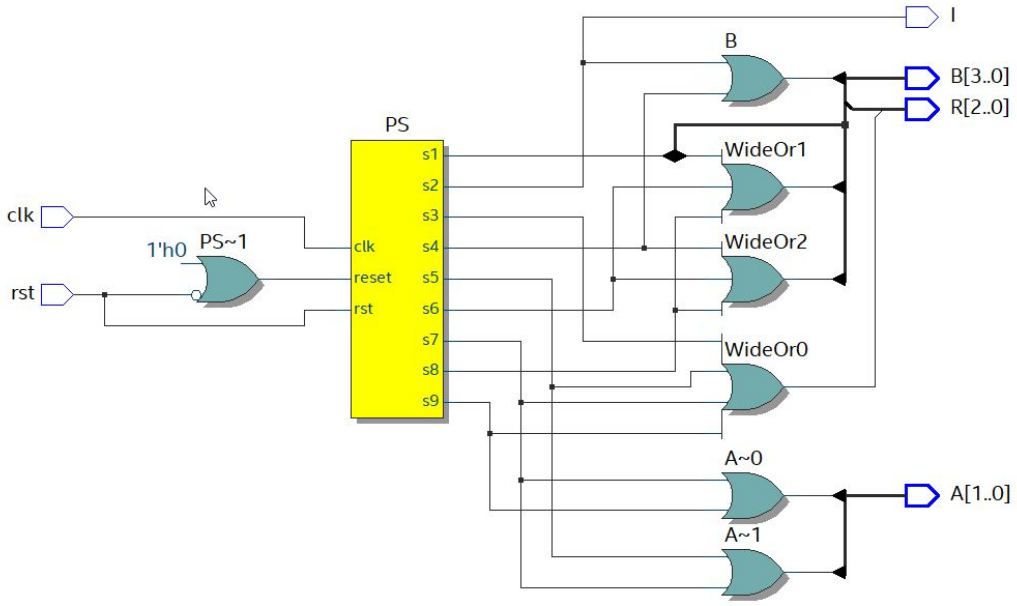
```
for(i = 0; i < 16; i = i+1)begin
        R_in = i;
        #9;
        $display("%d => %d clk: %d rst: %d en: %d", R_in, R, clk, rst, en);
        end
end

always
begin
#5;
clk <= ~clk;
end

endmodule
```



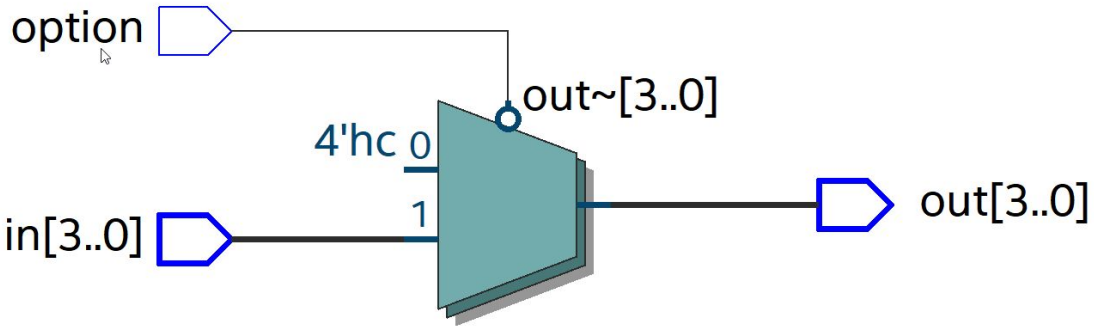
ALU



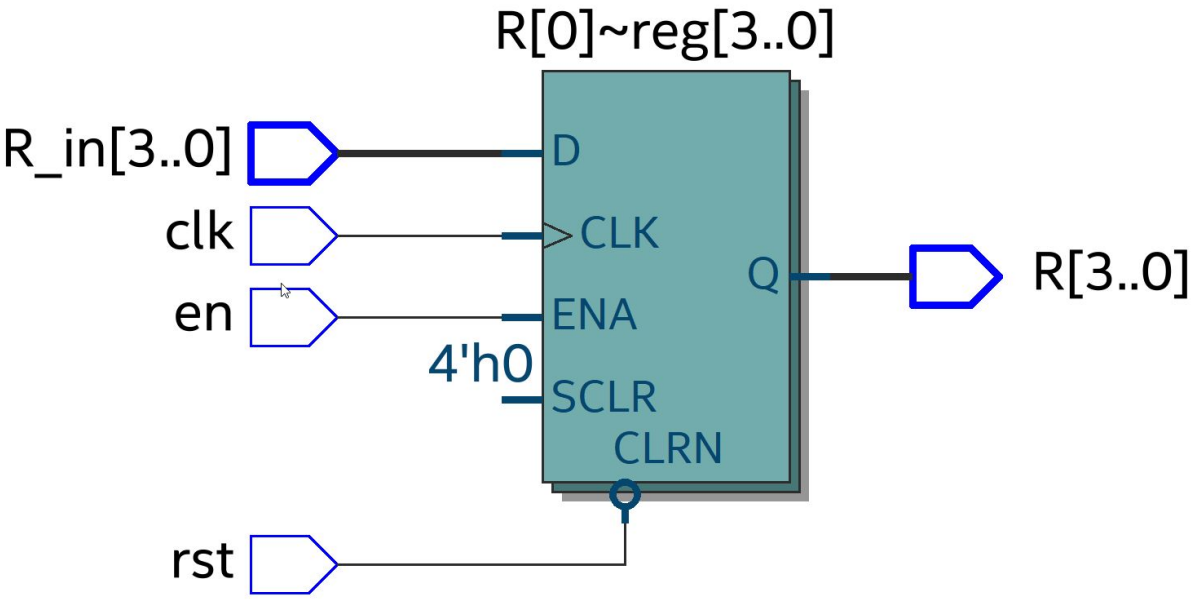
Finite State Machine



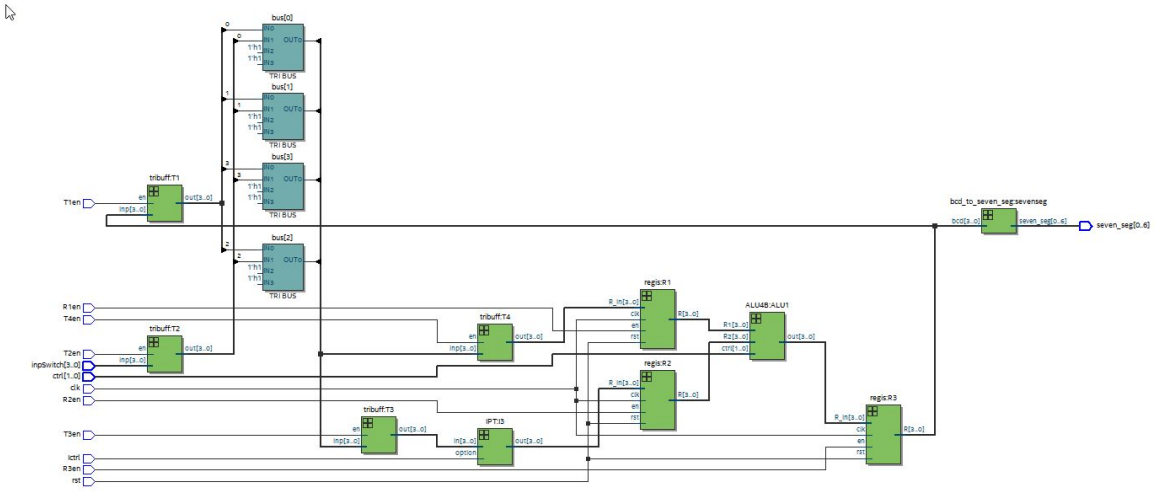
Immediate Pass through Unit/Mux



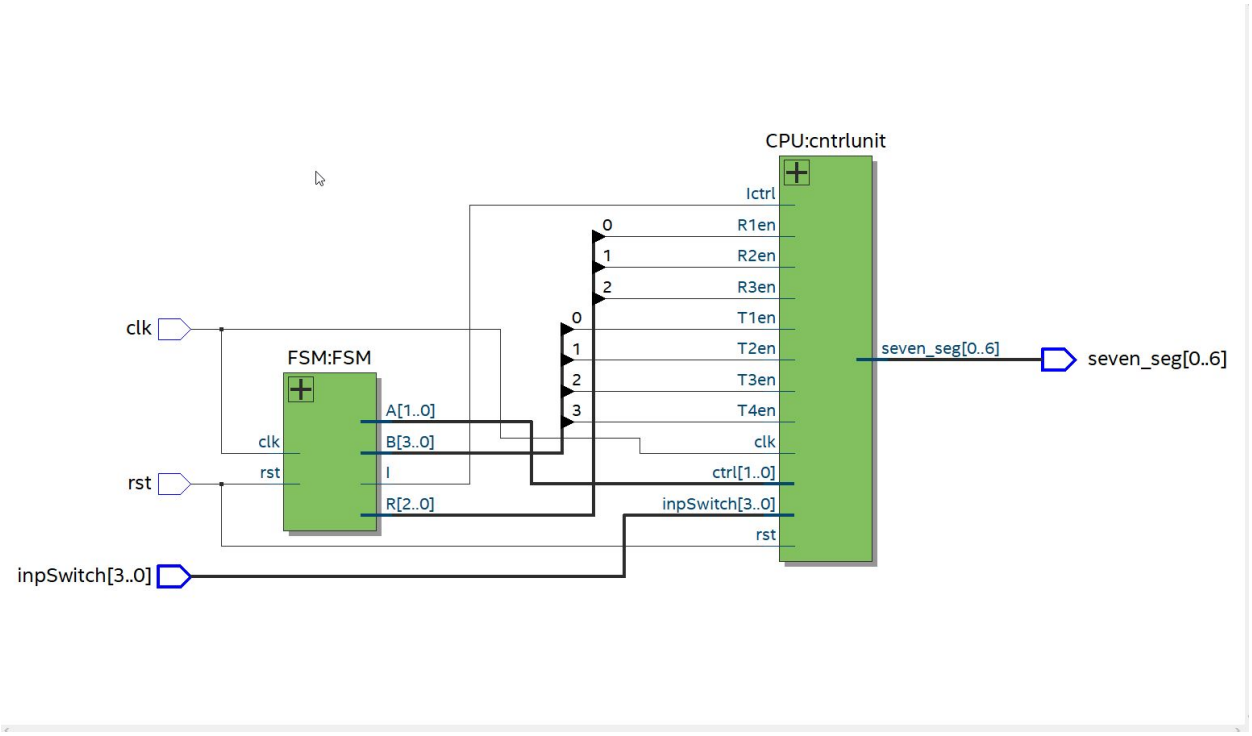
Registers



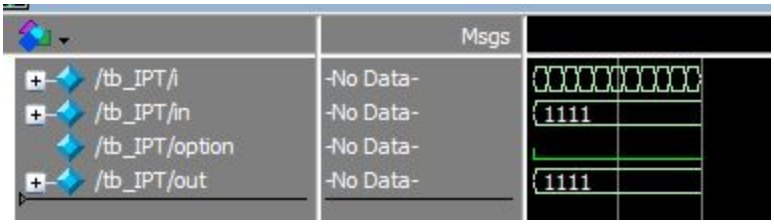
CPU



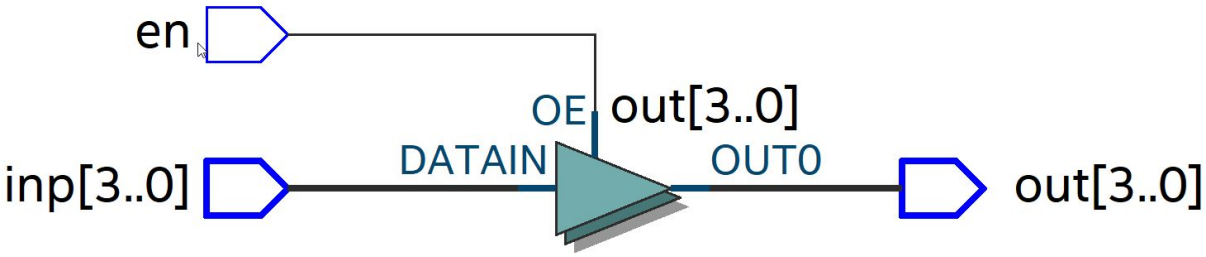
Full machine Image



Immediate pass through wave form



Tri State buffer



Code

CPU Combined unit

```
module CPU(inpSwitch,ctrl, T1en, T2en,T3en,T4en,R1en,R2en,R3en,clk,rst,seven_seg,Ictrl);
input[3:0]inpSwitch;
input[1:0]ctrl;
input T1en, T2en,T3en,T4en,R1en,R2en,R3en,clk,rst,Ictrl;
wire[3:0] Tout1,Tout2,bus,R1out,R2out,R3out,ALUout,Iout;
output[0:6] seven_seg;
bcd_to_seven_seg sevenseg(R3out,seven_seg);
ALU4B ALU1(ctrl,R1out,R2out,ALUout);
regis R1(Tout2,R1en,R1out,clk,rst);
regis R2(Iout,R2en,R2out,clk,rst);
regis R3(ALUout,R3en,R3out,clk,rst);
IPT I3(Tout1,Iout,Ictrl);
tribuff T1(R3out,T1en,bus);//R3
tribuff T2(inpSwitch, T2en, bus);//switches
tribuff T3(bus, T3en,Tout1);//R2
tribuff T4(bus, T4en,Tout2);//R1

endmodule
```

IPT Test bench

```
`timescale 1ps / 1ps
module tb_IPT;
reg [3:0]in;
reg option;
wire [3:0]out;
integer i;
IPT uut (
.in(in),
.out(out),
.option(option)
);

initial begin
option = 0;
in = 4'b1111;
for(i = 0; i < 16; i = i+1)begin
        //D = clk;
```

```
        #9;
        $display("%d => %d option: %d", in, out,option);
        end
end

endmodule
```

Immediate Pass through unit

```
module IPT(in,out,option);
input[3:0]in;
input option;
output reg[3:0]out;

always @(*)
begin
if(option == 0)
        out = in;
        else
        out = 4'b0011;
end
endmodule
```

Buffer

```
module tribuff(inp, en, out);
input[3:0] inp;
input en;
output [3:0]out;
assign out = en?inp: 4'bz;
endmodule
```

ALU

```
module ALU4B(ctrl, R1, R2, out);
input [1:0] ctrl;
input [3:0] R1, R2;
output reg [3:0] out;
always @(ctrl, R1, R2)
case (ctrl)
```

```
0: out = R1+R2;
1: out = R1|R2;
2: out = R1^R2;
3: out = ~R1;

endcase
endmodule
```

Registers

```
module regis(R_in,en,R,clk,rst);
input[3:0]R_in;
input en,rst,clk;
output reg[3:0]R;

always @(posedge clk,negedge rst)begin
if(rst == 0)
        R <= 4'b000;
else if(en == 1)
                        R <= R_in;
                else
                        R <= R;
end
endmodule
```

Finite State Machine

```
 module FSM(clk,rst,B,R,A,I);
input clk, rst;
output reg [3:0]B;
output reg[2:0]R;
output reg[1:0]A;
output reg I;
reg[3:0]PS,NS;
parameter[3:0] s0 = 4'b0000,s1 = 4'b0001,s2 = 4'b0010,s3 = 4'b0011,s4 = 4'b0100,s5 =
4'b0101,s6 = 4'b0110,s7 = 4'b0111,s8 = 4'b1000,s9 = 4'b1001;
//next state
// PI primary input
//PS present state
//rst Reset
always @(PS,rst)
```

```
begin
if(rst == 0)NS = s0;
else begin
case(PS)
s0: NS = s1;
s1: NS = s2;
s2: NS = s3;
s3: NS = s4;
s4: NS = s5;
s5: NS = s6;
s6: NS = s7;
s7: NS = s8;
s8: NS = s9;
s9: NS = s9;
endcase
end
end
//Dff logic
always @(posedge clk,negedge rst)begin
if(rst == 0) PS <= s0;
else PS <= NS;
end
//output
always @(PS)
begin
case(PS)
s0:
begin
R=3'b000;
A=2'b00;
B=4'b0000;
I = 0;
end
s1: begin
R=3'b001;
A=2'b00;
B=4'b1010;
I=0;
end
s2: begin
R=3'b010;
A=2'b00;
B=4'b0100;
```

```
I=1;
end
s3: begin
R=3'b100;
A=2'b00;
B=4'b0000;
I=0;
end
s4: begin
R=3'b010;
A=2'b00;
B=4'b0101;
I=0;
end
s5: begin
R=3'b100;
A=2'b01;
B=4'b0000;
I=0;
end
s6: begin
R=3'b001;
A=2'b00;
B=4'b1001;
I=0;
end
s7: begin
R=3'b100;
A=2'b11;
B=4'b0000;
I=0;
end
s8: begin
R=3'b001;
A=2'b00;
B=4'b1001;
I=0;
end
s9: begin
R=3'b100;
A=2'b10;
B=4'b0000;
I=0;
```

```
end
endcase
end
endmodule
```

Final Combined unit

```
module Fullunit(clk,rst,inpSwitch,seven_seg);
input clk,rst;
input [3:0]inpSwitch;
wire[3:0]B;
wire[2:0]R;
wire[1:0]A;
wire I;
output [0:6]seven_seg;
FSM FSM(clk,rst,B,R,A,I);
CPU cntrlunit(inpSwitch,A, B[0], B[1],B[2],B[3],R[0],R[1],R[2],clk,rst,seven_seg,I);
endmodule
```

Seven seg

```
module bcd_to_seven_seg(bcd,seven_seg);

input [3:0] bcd;
output reg [0:6] seven_seg;

always @*
 begin
  case (bcd)
   0 : begin seven_seg  = ~7'b1111110; end
   1 : begin seven_seg  = ~7'b0110000; end
   2 : begin seven_seg  = ~7'b1101101; end
   3 : begin seven_seg  = ~7'b1111001; end
   4 : begin seven_seg  = ~7'b0110011; end
   5 : begin seven_seg  = ~7'b1011011; end
   6 : begin seven_seg  = ~7'b1011111; end
   7 : begin seven_seg  = ~7'b1110000; end
         8 : begin seven_seg  = ~7'b1111111; end
   9 : begin seven_seg  = ~7'b1110011; end
         10 : begin seven_seg = ~7'b1110111; end
         11: begin seven_seg  = ~7'b0011111; end
```

```
        12 : begin seven_seg = ~7'b1001110; end
        13 : begin seven_seg = ~7'b0111101; end
        14: begin seven_seg  = ~7'b1001111; end
        15 : begin seven_seg = ~7'b1000111; end
   default : begin seven_seg = ~7'b0000000; end
  endcase
 end
endmodule
```