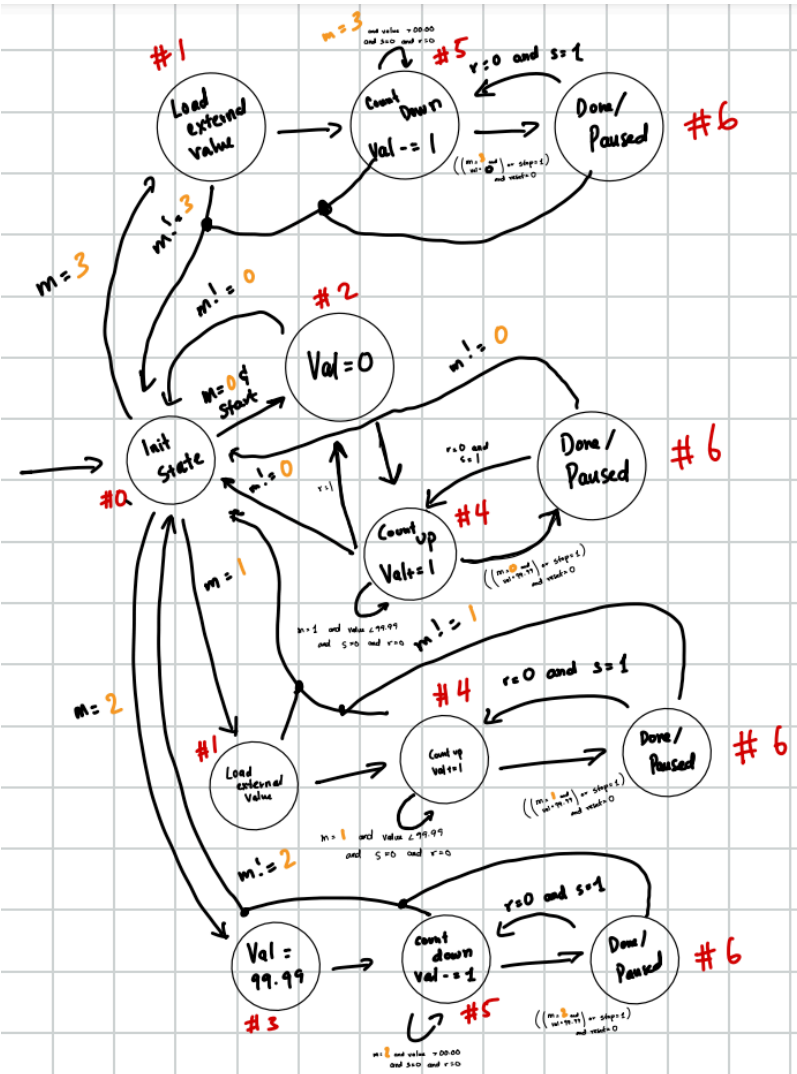
Brandon Lui BSL846 16730
Siddharth Kolukuluri SK55992 16730

HLSM

`



Inputs:

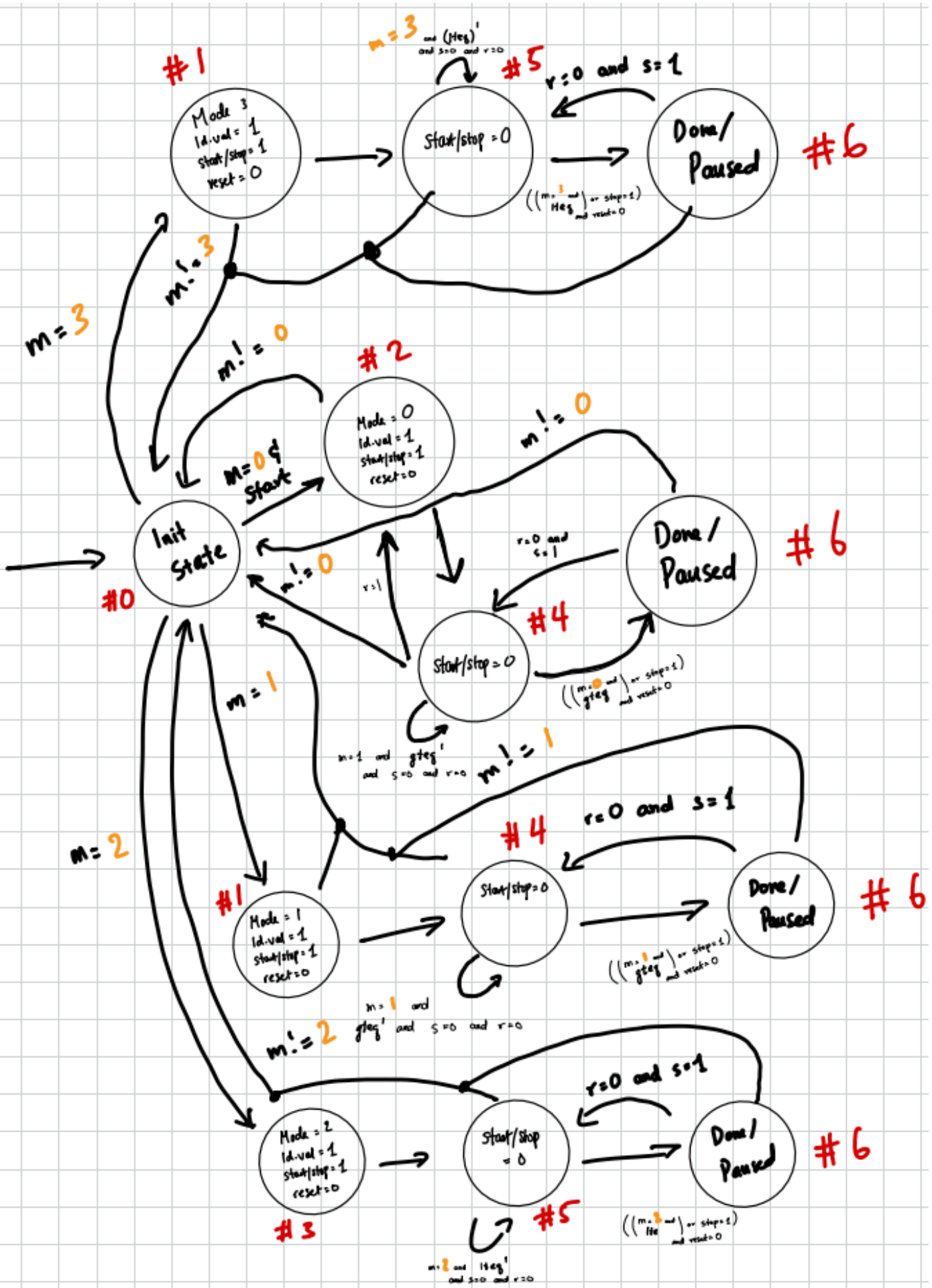
Mode → m — 2 switches

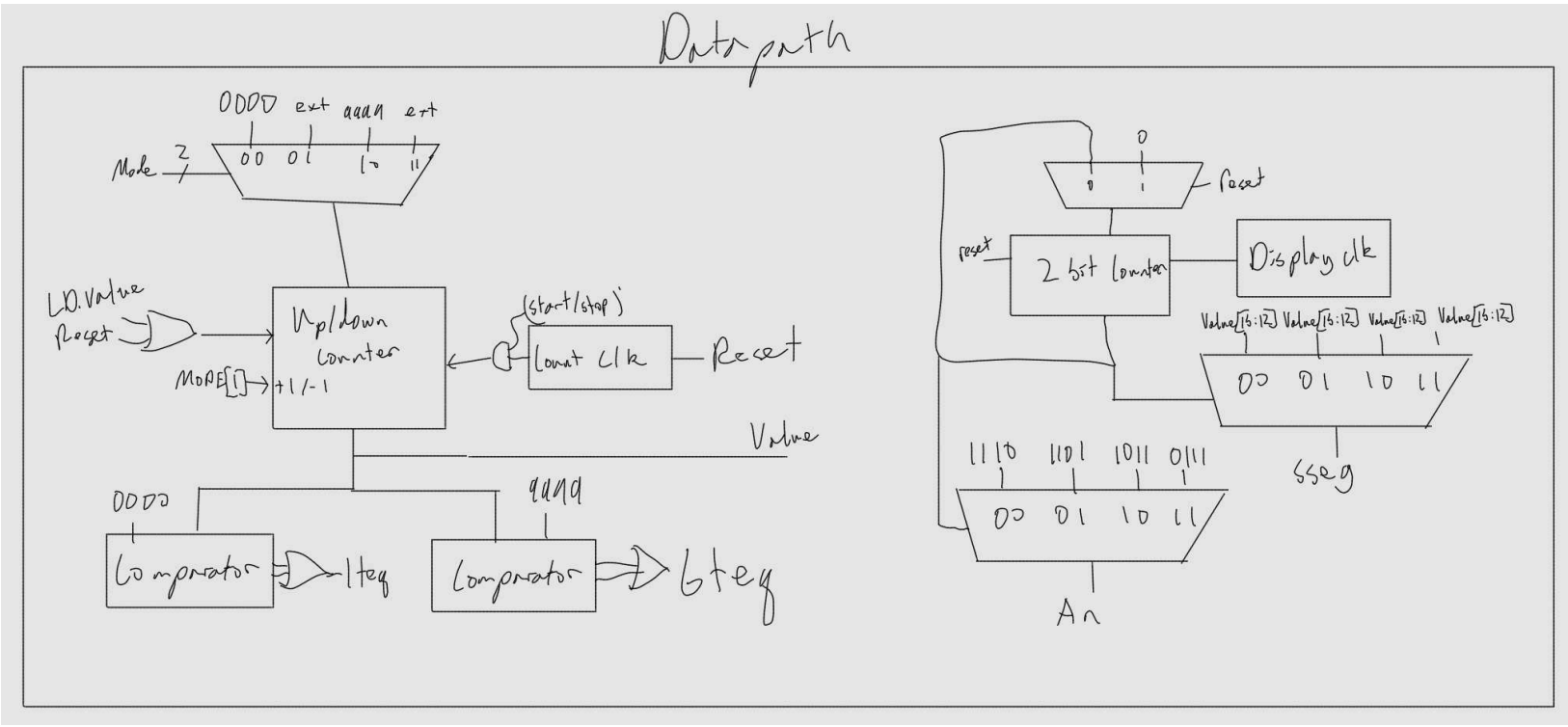
start/stop → s — 1 button

reset → r — 1 button

external load → e — 8 switches

Value → output to LED's


#1
Mode = 3
Id.val = 1
Start/stop = 1
reset = 0
m = 3 and (Meg)' and s=0 and r=0
#5
Start/stop = 0
r=0 and s=1
Done / Paused
#6
((m=3 and Meg) or stop=1) and reset=0
m != 3
m = 3
m != 0
#2
Mode = 0
Id.val = 1
Start/stop = 1
reset = 0
m=0 & start
Init State
#0
m != 0
r=0 and s=1
Done / Paused
#6
r=1
#4
Start/stop = 0
((m=0 and gteq) or stop=1) and reset=0
m = 1 and gteq' and s=0 and r=0
m != 1
m = 1
m = 2
#4
r=0 and s=1
Start/stop = 0
Done / Paused
#6
#1
Mode = 1
Id.val = 1
Start/stop = 1
reset = 0
((m=1 and gteq) or stop=1) and reset=0
m = 1 and gteq' and s=0 and r=0
m != 2
r=0 and s=1
Mode = 2
Id.val = 1
Start/stop = 1
reset = 0
Start/stop = 0
Done / Paused
#6
#3
#5
((m=2 and Meg) or stop=1) and reset=0
m=2 and Meg' and s=0 and r=0




```
`timescale 1ns / 1ps
module timer_state_machine(
    input fastclk,
    input clk,
    input reset,
    input stopstart,
    input [1:0] modesel,
    input [7:0] value,
    output reg[3:0] an,
    output reg[7:0] sseg
    );
    reg [0:0] stopping;
    reg [0:0] current_stopping;
    reg [0:0] resetting;
    reg [0:0] current_resetting;
    reg [1:0] alter;
    reg[2:0] state;
    reg[2:0] next_state;
    reg [15:0] val;
    reg [15:0] val_next;
    wire [7:0] in0, in1, in2, in3;
    hexto7segment c1 (.x(val[15:0]), .r1(in0), .r2(in1), .r3(in2) , .r4(in3));

    initial
    begin
```

```
    state <= 3'b000;
    alter = 0;
    val = 0;
    val_next = 0;
    stopping = 0;
    resetting = 0;
    current_stopping = 0;
    current_resetting = 0;
    end

always @(*) begin
    case(state)
        3'b000:
        begin
            if (modesel == 0)
                next_state = 3'b010;
            else if (modesel == 1)
                next_state = 3'b001;
            else if (modesel == 2)
                next_state = 3'b011;
            else if (modesel == 3)
                next_state = 3'b001;
        end

        3'b001:
        begin
            if (modesel != 1 && modesel != 3)
                next_state = 3'b000;
            else
                begin
                if (stopping == 0)
                    next_state = 3'b001;
                else if (modesel == 1)
                    next_state = 3'b100;
                else
                    next_state = 3'b101;
                end
            val_next = value[3:0]*100+value[7:4]*1000;
        end
        3'b010:
        begin
            if (modesel != 0)
                next_state = 3'b000;
            else
```

```
            begin
            if (stopping)
               next_state = 3'b100;
            else
               next_state = 3'b010;
            end
         val_next = 0;
      end
      3'b011:
      begin
         if (modesel != 2)
            next_state = 3'b000;
         else
         begin
         if (stopping)
            next_state = 3'b101;
         else
            next_state = 3'b011;
         end
         val_next = 9999;
      end
      3'b100:
      begin
         if (modesel != 1 && modesel != 0)
            next_state = 3'b000;
         else if (resetting == 1)
            begin
            if (modesel == 1)
               next_state = 3'b001;
            else
               next_state = 3'b010;
            end
         else if ((modesel == 1||modesel ==0) && (val == 9999 || stopping == 1) && resetting !=
1)
            begin
            next_state = 3'b110;
            val_next = val;
            end
         else
            begin
            next_state = 3'b100;
            val_next = val + 1;
            end
```

```
end
3'b101:
begin
   if (modesel != 2 && modesel != 3)
      next_state = 3'b000;
   else if (resetting == 1)
      begin
      if (modesel == 2)
         next_state = 3'b011;
      else
         next_state = 3'b001;
      end
   else if ((modesel == 2||modesel ==3) && (val == 0 || stopping == 1) && resetting != 1)
   begin
      next_state = 3'b110;
      val_next = val;
   end
   else
   begin
      next_state = 3'b101;
      val_next = val - 1;
   end
end
3'b110:
begin
   if (resetting == 1)
      begin
      if (modesel == 2)
         next_state = 3'b011;
      else if (modesel == 3 || modesel == 1)
         next_state = 3'b001;
      else
         next_state = 3'b010;
      end
   else if (stopping == 0)
      next_state = 3'b110;
   else
     begin
     if (modesel == 0 || modesel == 1)
         next_state = 3'b100;
     else if (modesel == 2 || modesel == 3)
         next_state = 3'b101;
     end
   val_next = val;
```

```
            end

        endcase
    end
    always @(*) begin
        case(alter)
            2'b00: begin sseg = in0; an = 4'b1110; end
            2'b01: begin sseg = in1; an = 4'b1101; end
            2'b10: begin sseg = in2; an = 4'b1011; end
            2'b11: begin sseg = in3; an = 4'b0111; end
        endcase
    end

    always @(posedge fastclk)
    begin
         alter = alter + 1;
    end
    always @(posedge reset or negedge clk)
    begin
        if(reset)
          current_resetting = reset;
        else
          current_resetting = 0;
    end


   always @(posedge clk or posedge stopstart)
   begin
        if (stopstart)
            current_stopping <= 1;
        else
        begin
        resetting <= current_resetting;
        stopping <= current_stopping;
        state <= next_state;
        if (current_stopping)
            val <= val;
        else
            val <= val_next;
        current_stopping = 0;
        end
   end
endmodule
```

```
`timescale 1ns / 1ps
module timer_main(
    input clk,
    input reset,
    input stopstart,
    input [1:0] modesel,
    input [7:0] value,
    output [3:0] an,
    output [7:0] sseg
    );
    wire slow_clk;
    wire fastclk;

    clkdiv c5 (.clk(clk), .reset(reset), .divider(1000000), .clk_out(slow_clk));
    clkdiv c1 (.clk(clk), .reset(reset),  .divider(2000),.clk_out(fastclk));

    timer_state_machine timey(
    .fastclk(fastclk),
    .clk(slow_clk),
    .reset(reset),
    .stopstart(stopstart),
    .modesel(modesel),
    .value(value),
    .an(an),
    .sseg(sseg));

endmodule

`timescale 1ns / 1ps

module clkdiv(
    input clk,
    input [62:0] divider,
    input reset,
    output clk_out
    );

    reg [30:0] COUNT;
    reg dived;
    assign clk_out = dived;
    initial
        COUNT = 0;
    always @(posedge clk)
    begin
```

```
    if(reset)
        COUNT = 0;
    else
        COUNT = COUNT + 1;
    if(COUNT == divider)
    begin
        dived = 1;
        COUNT = 0;
    end
    else
        dived = 0;
    end
endmodule

`timescale 1ns / 1ps


module hexto7segment(
    input [15:0] x,
    output reg [7:0] r1,
    output reg [7:0] r2,
    output reg [7:0] r3,
    output reg [7:0] r4
    );

    always @(*)begin
        case(x/1000)
            16'h0: r4 = 8'b00000011;
            16'h1 : r4 = 8'b10011111;
            16'h2 : r4 = 8'b00100101;
            16'h3 : r4 = 8'b00001101;
            16'h4 : r4 = 8'b10011001;
            16'h5 : r4 = 8'b01001001;
            16'h6 : r4 = 8'b01000001;
            16'h7 : r4 = 8'b00011111;
            16'h8 : r4 = 8'b00000001;
            16'h9 : r4 = 8'b00011001;
       endcase
       case((x/100)%10)
            16'h0: r3 = 8'b00000010;
            16'h1 : r3 = 8'b10011110;
            16'h2 : r3 = 8'b00100100;
            16'h3 : r3 = 8'b00001100;
            16'h4 : r3 = 8'b10011000;
```

```
            16'h5 : r3 = 8'b01001000;
            16'h6 : r3 = 8'b01000000;
            16'h7 : r3 = 8'b00011110;
            16'h8 : r3 = 8'b00000000;
            16'h9 : r3 = 8'b00011000;
        endcase
        case((x/10)%10)
            16'h0: r2 = 8'b00000011;
            16'h1 : r2 = 8'b10011111;
            16'h2 : r2 = 8'b00100101;
            16'h3 : r2 = 8'b00001101;
            16'h4 : r2 = 8'b10011001;
            16'h5 : r2 = 8'b01001001;
            16'h6 : r2 = 8'b01000001;
            16'h7 : r2 = 8'b00011111;
            16'h8 : r2 = 8'b00000001;
            16'h9 : r2 = 8'b00011001;
        endcase
        case(x%10)
            16'd0: r1 = 8'b00000011;
            16'd1 : r1 = 8'b10011111;
            16'd2 : r1 = 8'b00100101;
            16'd3 : r1 = 8'b00001101;
            16'd4 : r1 = 8'b10011001;
            16'd5 : r1 = 8'b01001001;
            16'd6 : r1 = 8'b01000001;
            16'd7 : r1 = 8'b00011111;
            16'd8 : r1 = 8'b00000001;
            16'd9 : r1 = 8'b00011001;
        endcase
        end
endmodule


set_property PACKAGE_PIN W5 [get_ports {clk}]
 set_property IOSTANDARD LVCMOS33 [get_ports {clk}]
 create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clk}]
set_property PACKAGE_PIN V17 [get_ports {modesel[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {modesel[0]}]
set_property PACKAGE_PIN V16 [get_ports {modesel[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {modesel[1]}]
set_property PACKAGE_PIN W16 [get_ports {value[0]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {value[0]}]
set_property PACKAGE_PIN W17 [get_ports {value[1]}]
```

```
 set_property IOSTANDARD LVCMOS33 [get_ports {value[1]}]
set_property PACKAGE_PIN W15 [get_ports {value[2]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {value[2]}]
set_property PACKAGE_PIN V15 [get_ports {value[3]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {value[3]}]
set_property PACKAGE_PIN W14 [get_ports {value[4]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {value[4]}]
set_property PACKAGE_PIN W13 [get_ports {value[5]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {value[5]}]
set_property PACKAGE_PIN V2 [get_ports {value[6]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {value[6]}]
set_property PACKAGE_PIN T3 [get_ports {value[7]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {value[7]}]
set_property PACKAGE_PIN W7 [get_ports {sseg[7]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sseg[7]}]
set_property PACKAGE_PIN W6 [get_ports {sseg[6]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sseg[6]}]
set_property PACKAGE_PIN U8 [get_ports {sseg[5]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sseg[5]}]
set_property PACKAGE_PIN V8 [get_ports {sseg[4]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sseg[4]}]
set_property PACKAGE_PIN U5 [get_ports {sseg[3]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sseg[3]}]
set_property PACKAGE_PIN V5 [get_ports {sseg[2]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sseg[2]}]
set_property PACKAGE_PIN U7 [get_ports {sseg[1]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sseg[1]}]
set_property PACKAGE_PIN V7 [get_ports {sseg[0]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sseg[0]}]
set_property PACKAGE_PIN U2 [get_ports {an[0]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]
set_property PACKAGE_PIN T18 [get_ports {stopstart}]
 set_property IOSTANDARD LVCMOS33 [get_ports {stopstart}]
set_property PACKAGE_PIN T17 [get_ports {reset}]
 set_property IOSTANDARD LVCMOS33 [get_ports {reset}]
```