

Verilog Examples

Example 1: This module reads pushbutton on FPGA, and whenever push button is pressed leds shift to left and the led on the right side is turned on. If all leds are turned on then all of them are turned off.

Note: Pushbuttons of FPGA is active low and debounced.

Pin Planner:

	Node Name	Direction	Location
out	leds[7]	Output	PIN_L3
out	leds[6]	Output	PIN_B1
out	leds[5]	Output	PIN_F3
out	leds[4]	Output	PIN_D1
out	leds[3]	Output	PIN_A11
out	leds[2]	Output	PIN_B13
out	leds[1]	Output	PIN_A13
out	leds[0]	Output	PIN_A15
in	pb1	Input	PIN_J15

Input: 1 bit, connected to push button

Output: 8 bits, connected to leds.

```
module fpga_counter (leds, pb1);
output reg[7:0] leds;
input pb1;

always @(posedge pb1)
begin
    if (leds == 8'hFF)
        leds<=8'h00;
    else
        leds<= (leds<<1) + 1;
    end

initial
    leds=8'h00;





endmodule
```

Example 2: This module adds two one bit number. Numbers are read from dipswitches on FPGA and output of addition and carry bit are shown on leds.

Inputs: 2 bits, connected to dipswitches

Output: 2 bits, connected to leds

Pin Planner:

Node Name	Direction	Location
 leds[1]	Output	PIN_A13
 leds[0]	Output	PIN_A15
 sw1	Input	PIN_M1
 sw2	Input	PIN_T8

```
module onebitadder (sw1, sw2, leds);
```

```
input wire sw1, sw2;
```

```
output reg [1:0] leds;
```

```
always @(*)
```

```
begin
```

```
    leds[0]= sw1^sw2;
```

```
    if ((sw1==1)&&(sw2==1))
```

```
        leds[1]=1;
```

```
    else
```

```
        leds[1]=0;
```

```
end
```

```
endmodule
```

Same example, but here we use assign statement

```
module seqcirc ( sw1, sw2, leds);
```

```
input wire sw1, sw2;
```

```
output [1:0] leds;
```

```
    assign leds[0]= sw1 ^ sw2;
```

```
    assign leds[1]= (sw1&sw2) ? 1 :0;
```

```
endmodule
```

Example 3: Basic pushbutton example. In this module, when we press pb0, H33 is shown on leds and when we press pb1, HCC is shown on leds.

Input: 2 bits, connected to pushbuttons.

Output: 8 bits, connected to leds.

Pin assignment are same as previous examples.

```
module push_sv (pb, leds);
```

```
input [1:0] pb;
```

```
output reg [7:0] leds;
```

```
always @*
```

```
begin
```

```
    if (~pb[0]) leds=8'h33;
```

```
    else leds=leds;
```

```
    if (~pb[1]) leds=9'hCC;
```

```
    else leds=leds;
```

```
end
```

```
initial  leds=8'h00;
```













```
endmodule
```

Example 4: Basic ALU operations. This module reads ALU operation code from dipswitches and performs the operation. Result is shown on leds. The parameters of the operation are 7 bits local variables w1 and w2. Led[7] is used to show Zero Flag.

Inputs: 4 bits opcode, connected to dipswitches

Outputs: 7 bits result, 1 bit Zero flag, connected to leds.

Pin Planner:

Node Name	Direction	Location
 leds[7]	Output	PIN_L3
 leds[6]	Output	PIN_B1
 leds[5]	Output	PIN_F3
 leds[4]	Output	PIN_D1
 leds[3]	Output	PIN_A11
 leds[2]	Output	PIN_B13
 leds[1]	Output	PIN_A13
 leds[0]	Output	PIN_A15
 switch[3]	Input	PIN_M15
 switch[2]	Input	PIN_B9
 switch[1]	Input	PIN_T8
 switch[0]	Input	PIN_M1

```
module aluop(switch, leds);
```

```
output reg [7:0] leds;
```

```
input [3:0] switch;
```

```
reg [6:0] w1,w2;
```

```
always @*
```

```
begin
```

```
w1=7'h22;
```

```
w2=7'h11;
```

```
    case(switch)
```

```
        4'b0000: leds[6:0]=w1+w2; // 0 addition
```

```
        4'b0001: leds[6:0]=w1-w2; // 1 sub
```

```
        4'b0010: leds[6:0]=w1&w2; // 2 bitwise and
```

```
        4'b0011: leds[6:0]=w1|w2; // 3 bitwise or
```

```
        4'b0100: leds[6:0]=w1^w2; // 4 bitwise xor
```

```
        4'b0101: leds[6:0]=~w1; // 5 bitwise not
```

```
        4'b0110: leds[6:0]=w1; // 6 mov
```

```
        4'b0111: leds[6:0]= w1+1; // 7 increment
```

```
        4'b1000: leds[6:0]= w1-1; // 8 decrement
```

```
        default: leds[6:0] = 0;
```

```
    endcase
```

```
    if (leds[6:0]==0)
```

```
        leds[7]=1;
```

```
    else
```

```
        leds[7]=0;
```

```
end
```

```
endmodule
```

Example 5: Clock example. In this example, we will shift the bits to the left in each clock cycle. But it is not possible to see shift operation because clock is so fast. To overcome this problem we design another clock that is much more slower than the original one.

Input: 1 bit clock.

Output: 8 bit leds.

Pin assignment are same as previous examples.

```
module clock_ex (leds,clk);
```

```
input clk;
```

```
output reg [7:0] leds;
```

```
reg [20:0] clk1;
```

```
always @(posedge clk)
```

```
    clk1<=clk1+1;
```

```
always @(posedge clk1[20])
```

```
    leds<={leds[6:0],leds[7]};
```

```
initial leds=8'b1;
```

```
endmodule
```

Example 6: This module reads pushbuttons, if pb1 is pressed, leds are shifted to left and if pb2 is pressed leds are shifted to right.

Inputs: 3 bits, 2 of them connected to push buttons and we have 1 bit clock.

Output: 8 bits, connected to leds.

Pin Planner:

Node Name	Direction	Location
in clk	Input	PIN_R8
out leds[7]	Output	PIN_L3
out leds[6]	Output	PIN_B1
out leds[5]	Output	PIN_F3
out leds[4]	Output	PIN_D1
out leds[3]	Output	PIN_A11
out leds[2]	Output	PIN_B13
out leds[1]	Output	PIN_A13
out leds[0]	Output	PIN_A15
in pb 1	Input	PIN_E1
in pb2	Input	PIN_J15

First version of Verilog code. Here we have used clock to overcome bouncing problem.

```
module clk_ex (b1,b2,leds,clk);
```

```
input b1,b2;
```

```
input clk;
```

```
output reg [7:0] leds;
```

```
reg [26:0] clk1;
```

```
assign buttons={b1,b2};
```

```
always @(posedge clk)
```

```
clk1<=clk1+1;
```

```
always @(posedge clk1[25])
```

```
begin
```

```
    if (buttons==2'b10)
```

```
        leds<={leds[6:0], 1'b1};
```

```
    else if (buttons==2'b01)
```

```
        leds<={1'b0,leds[7:1]};
```

```
    else
```

```
        leds<=leds;
```

```
end
```

```
initial
```

```
leds=8'b111;
```

```
endmodule
```

Second version of the code. Here we desing a FSM to overcome bouncing problem.

```
module clock_ex (leds, pb1, pb2, clk);

input clk;
input pb1, pb2;
output reg [7:0] leds;
reg [1:0] state;
wire [1:0] buttons;
assign buttons={~pb1,~pb2};

always @(posedge clk)
begin
    case (state)

        2'b00:
            begin
                case(buttons)
                    2'b01:
                        begin
                            state<=2'b01;
                            leds<={leds[6:0],1'b1};
                        end
                    2'b10:
                        begin
                            state<=2'b10;
                            leds<={1'b0,leds[7:1]};
                        end
                endcase
            end

        2'b01:
            begin
                if (buttons==2'b00)
                    state<=2'b00;
            end

        2'b10:
            begin
                if (buttons==2'b00)
                    state<=2'b00;
            end
    endcase
end //end always
initial
begin
    leds=8'b111;
    state=2'b00;
end

endmodule
```

Example 7: Vending Machine. In this module pushbuttons are assigned to 5 and 10 cents. Whenever we pressed key0 or key1, we insert 5 cent or 10 cent into vending machine respectively. We assume the cost of all goods in vending machine is 20 cents. When total money in vending machine is greater than or equal to 20 cents the counter will be incremented and shown on leds. Balance is also stored.

Inputs: 3 bits, 2 of them connected to push buttons and we have 1 bit clock.

Output: 8 bits, connected to leds.

Pin Planner:

Node Name	Direction	Location
in clk	Input	PIN_R8
out leds[7]	Output	PIN_L3
out leds[6]	Output	PIN_B1
out leds[5]	Output	PIN_F3
out leds[4]	Output	PIN_D1
out leds[3]	Output	PIN_A11
out leds[2]	Output	PIN_B13
out leds[1]	Output	PIN_A13
out leds[0]	Output	PIN_A15
in pb1	Input	PIN_E1
in pb2	Input	PIN_J15

```
module coins (key0, key1, leds, clk);
```

```
input key0, key1, clk;
```

```
output reg [7:0] leds;
```

```
reg [7:0] total;
```

```
wire [1:0] buttons;
```

```
assign buttons= {~key1, ~key0};
```

```
reg [1:0] bstate;
```

```
always @(posedge clk)
```

```
begin
```

```
    case (bstate)
```

```
        2'b00:
```

```
            begin
```

```
                if (buttons==2'b01)
```

```
                    begin
```

```
                        bstate<= 2'b01;
```

```
                        total<=total+5;
```

```
                    end
```

```
                else if (buttons==2'b10)
```

```
                    begin
```

```
                        bstate<= 2'b10;
```

```
                        total<=total+10;
```

```
                    end
```

```
                if (total>= 20)
```

```
                    begin
```

```
                        leds<=leds+1;
```

```
                        total <= total-20;
```

```
                    end
```

```
            end
```

```
        endcase
```

```
    end
```



```

                                end
                        end
2'b01:
    begin
        if (buttons==2'b00)
            bstate<=2'b00;
        else
            bstate<=2'b01;
        end
2'b10:
    begin
        if (buttons==2'b00)
            bstate<=2'b00;
        else
            bstate<=2'b10;
        end
    endcase
end

initial begin
    leds=8'b0;
    bstate=2'b00;
    total=0;
end
endmodule

```

Example 8: Seven Segment Display. In this module we implement a seven segment display. Dipswitches choose which of the 7segment unit turn on. The value to be shown on 7segment unit is stored in local variable “data”.

Input: 4 bits, connected to dipswitches

Output: 4 bits, to control the ground of each 7 segment display unit. 7 bits, leds on 7 segment display, 1 bit for clock.

Pin Assignment:

in	clk	Input	PIN_R8
out	display[6]	Output	PIN_F8
out	display[5]	Output	PIN_D8
out	display[4]	Output	PIN_E6
out	display[3]	Output	PIN_C6
out	display[2]	Output	PIN_D6
out	display[1]	Output	PIN_A6
out	display[0]	Output	PIN_D5
in	ds[3]	Input	PIN_M15
in	ds[2]	Input	PIN_B9
in	ds[1]	Input	PIN_T8
in	ds[0]	Input	PIN_M1
out	grounds[3]	Output	PIN_B4
out	grounds[2]	Output	PIN_A3
out	grounds[1]	Output	PIN_C3
out	grounds[0]	Output	PIN_D3

```
module switch_segment (display, grounds, ds);
```

```
output reg [6:0] display;
```

```
output reg [3:0] grounds;
```

```
input [3:0] ds;
```

```
reg [3:0] data;
```

```
always @(*)
```

```
    grounds=ds;
```

```
always @(*)
```

```
    begin
```

```
        case (data)
```

```
            0:display=7'b1111110; //starts with a, ends with g
```

```
            1:display=7'b0110000;
```

```
            2:display=7'b1101101;
```

```
            3:display=7'b1111001;
```

```
            4:display=7'b0110011;
```

```
            5:display=7'b1011011;
```

```
            6:display=7'b1011111;
```

```
            7:display=7'b1110000;
```

```
            8:display=7'b1111111;
```

```
            9:display=7'b1111011;
```

```
            'ha:display=7'b1110111;
```

```
            'hb:display=7'b0011111;
```

```
            'hc:display=7'b1001110;
```

```
            'hd:display=7'b0111101;
```

```
            'he:display=7'b1001111;
```

```
            'hf:display=7'b1000111;
```

```
            Default: display=7'b1111111;
```

```
endcase  
end
```

```
initial data=8'ha;
```

```
endmodule
```

Example 9: In this example, whenever we press key0 or key 1, data shown on 7 segment display unit is shifted to the left or right.

Input: 2 bits, connected to pushbuttons

Output: 4 bits, to control the ground of each 7 segment display unit. 7 bits, leds on 7 segment display, 1 bit for clock.

Pin Assignment

in	clk	Input	PIN_R8
out	display[6]	Output	PIN_F8
out	display[5]	Output	PIN_D8
out	display[4]	Output	PIN_E6
out	display[3]	Output	PIN_C6
out	display[2]	Output	PIN_D6
out	display[1]	Output	PIN_A6
out	display[0]	Output	PIN_D5
out	grounds[3]	Output	PIN_B4
out	grounds[2]	Output	PIN_A3
out	grounds[1]	Output	PIN_C3
out	grounds[0]	Output	PIN_D3
in	keys[1]	Input	PIN_E1
in	keys[0]	Input	PIN_J15

```
module switch_segment (display, grounds, clk, keys);
```

```
output reg [6:0] display;
```

```
output reg [3:0] grounds;
```

```
input clk;
```

```
input [1:0] keys;
```

```
wire [1:0] buttons;
```

```
assign buttons={~keys[1],~keys[0]};
```

```
reg [1:0] state;
```

```
reg [3:0] data;
```

```
always @(posedge clk)
```

```
begin
```

```
case (state)
```

```
2'b00:
```

```
begin
```

```
case (buttons)
```

```
2'b01:
```

```
begin
```

```
state<=2'b01;
```

```
grounds<={grounds[2:0],grounds[3]};
```

```
end
```

```
2'b10:
```

```
begin
```

```
state<=2'b10;
```

```
grounds<={grounds[0],grounds[3:1]};
```

```
end
```

```

                endcase
            end

2'b01:
    begin
        if (buttons==2'b00)
            state<=2'b00;
        end
    end

2'b10:
    begin
        if (buttons==2'b00)
            state<=2'b00;
        end
    end

endcase
end

always @(*)
    begin
        case (data)
            0:display=7'b1111110; //starts with a, ends with g
            1:display=7'b0110000;
            2:display=7'b1101101;
            3:display=7'b1111001;
            4:display=7'b0110011;
            5:display=7'b1011011;
            6:display=7'b1011111;
            7:display=7'b1110000;
            8:display=7'b1111111;
            9:display=7'b1111011;
            'ha:display=7'b1110111;
            'hb:display=7'b0011111;
            'hc:display=7'b1001110;
            'hd:display=7'b0111101;
            'he:display=7'b1001111;
            'hf:display=7'b1000111;
            Default: display=7'b1111111;
        endcase
    end

initial begin
    data=8'hc;
    grounds=4'b1110;
    state=2'b00;
end

endmodule

```

Example 10: 16 bit Counter. In this module we have 16 bit number. In each posedge of clk1[23] it will be incremented and the number is shown on 7 segment display.

Input: 1 bit clock

Output: 4 bits for ground, 7 bits for leds on 7segment display.

Pin assignment is same as previous example.

```
module seventsegment(grounds, display, clk);

output reg [3:0] grounds;
output reg [6:0] display;
input clk;

reg [3:0] data [3:0] ; //number to be printed on display
reg [1:0] count;      //which data byte to display.
reg [25:0] clk1;
reg [15:0] number;

always @(posedge clk1[15]) //25 slow //19 wavy //15 perfect
begin
    grounds<={grounds[2:0],grounds[3]};
    count<=count+1;

end

always @(posedge clk)
    clk1<=clk1+1;

always @(posedge clk1[23])
    number<=number+1;

always @(*)
    case(data[count])
        0:display=7'b1111110; //starts with a, ends with g
        1:display=7'b0110000;
        2:display=7'b1101101;
        3:display=7'b1111001;
        4:display=7'b0110011;
        5:display=7'b1011011;
        6:display=7'b1011111;
        7:display=7'b1110000;
        8:display=7'b1111111;
        9:display=7'b1111011;
        'ha:display=7'b1110111;
        'hb:display=7'b0011111;
        'hc:display=7'b1001110;
        'hd:display=7'b0111101;
        'he:display=7'b1001111;
        'hf:display=7'b1000111;
        default display=7'b1111111;
    endcase
```

```
always @(*)  
begin  
    data[0]=number[15:12];  
    data[1]=number[11:8];  
    data[2]=number[7:4];  
    data[3]=number[3:0];  
end
```

```
initial begin  
    number = 0;  
    count = 2'b0;  
    grounds=4'b1110;  
    clk1 = 0;  
end
```

```
endmodule
```

Example 11: In this module, one of the push button is assigned to 1 and other one is assigned to 10. Whenever we press one of the push buttons, the value assigned to pushbuttons is added to total value and shown on 7 segment display.

Inputs: 1 bit for clock, 2 bits for pushbuttons

Outputs: 4 bit for grounds, 7 bit for leds on 7 segment display

Pin assignments are same as previous examples.

```

module seventsegment(grounds, display, clk, keys);

output reg [3:0] grounds;
output reg [6:0] display;
input [1:0] keys;
input clk;

reg [3:0] data [3:0] ; //number to be printed on display
reg [1:0] count;      //which data byte to display.
reg [25:0] clk1;
reg [15:0] number;
wire [1:0] buttons;
assign buttons={~keys[1],~keys[0]};
reg [1:0] state;

always @(posedge clk1[15]) //25 slow //19 wavy //15 perfect
begin
    grounds<={grounds[2:0],grounds[3]};
    count<=count+1;

end

always @(posedge clk)
begin
    case (state)
    2'b00:
        begin
            case (buttons)
            2'b01:
                begin
                    state<=2'b01;
                    number<=number+1;
                end
            2'b10:
                begin
                    state<=2'b10;
                    number<=number+10;
                end
            endcase
        end
    2'b01:

```



```

        begin
            if (buttons==2'b00)
                state<=2'b00;
            end
2'b10:
        begin
            if (buttons==2'b00)
                state<=2'b00;
            end
        endcase
    end

    always @(posedge clk)
        clk1<=clk1+1;

    always @(*)
        case(data[count])
            0:display=7'b1111110; //starts with a, ends with g
            1:display=7'b0110000;
            2:display=7'b1101101;
            3:display=7'b1111001;
            4:display=7'b0110011;
            5:display=7'b1011011;
            6:display=7'b1011111;
            7:display=7'b1110000;
            8:display=7'b1111111;
            9:display=7'b1111011;
            'ha:display=7'b1110111;
            'hb:display=7'b0011111;
            'hc:display=7'b1001110;
            'hd:display=7'b0111101;
            'he:display=7'b1001111;
            'hf:display=7'b1000111;
            default display=7'b1111111;
        endcase

    always @(*)
        begin
            data[0]=number[15:12];
            data[1]=number[11:8];
            data[2]=number[7:4];
            data[3]=number[3:0];
        end

    initial begin
        number = 0;
        count = 2'b0;
        grounds=4'b1110;
        clk1 = 0;
    end

endmodule

```

Example 12: In this module we implement a dipswitch controlled shift operation. Whenever we turn on dipswitch led is shifted to the left once. Because of bouncing problem we have to read input till we get stable signal.

Input: 4 bits dipswitches, (one of them is used for his example)

Output: 8 bits leds and 1 bit clock.

Pin assignments are same as previous examples. Dip switch is connected to any GPIO port. Ground and Vcc those are necessary for dipswitches are also connected to appropriate GPIO ports.

```
module dipswitchex (
    input [3:0] ds,
    output reg [7:0] leds,
    input clk);

reg ds0;
reg state;
reg [7:0] check;

always @(posedge clk)
begin
    check<={check[6:0], ds[0]};
    if (check==8'b11111111)
        ds0<=1;
    else
        ds0<=0;
end

always @(posedge clk)
begin
    case (state)
        1'b0:
            begin
                if (ds0==1)
                    begin
                        state<=1;
                        leds<={leds[6:0], leds[7]};
                    end
                end
            begin
                if (ds0==0)
                    state<=0;
                end
            endcase
    end

initial begin
    state=0;
    leds=8'b00000001;
    check=0;
end
endmodule
```

Example 13: In this example dipswitches are used to add 1, 2, 4 and 8 to total value. Initially all dipswitches are off and whenever a dipswitch is turned on, assigned value to this switch is added to total value. To add a new number, all switches should be turned off. Total value is shown on 7 segment display. Switch position also can be seen on leds.

Inputs, outputs and pin assignments are same as previous example

```
module dipswitches (ds, leds, display, grounds,clk);
output reg [3:0] leds;
output reg [6:0] display;
output reg [3:0] grounds;
input [3:0] ds;
input clk;

reg [3:0] data [3:0] ;
reg [1:0] count;
reg [25:0] clk1;
reg [15:0] number;
reg [3:0] state;
wire [3:0] switch;

assign switch=ds;

always @(posedge clk)
    clk1<=clk1+1;

always @(posedge clk1[15]) //25 slow //19 wavy //15 perfect
begin
    grounds<={grounds[2:0],grounds[3]};
    count<=count+1;
end

always @(*)
case(data[count])
    0:display=7'b1111110; //starts with a, ends with g
    1:display=7'b0110000;
    2:display=7'b1101101;
    3:display=7'b1111001;
    4:display=7'b0110011;
    5:display=7'b1011011;
    6:display=7'b1011111;
    7:display=7'b1110000;
    8:display=7'b1111111;
    9:display=7'b1111011;
    'ha:display=7'b1110111;
    'hb:display=7'b0011111;
    'hc:display=7'b1001110;
    'hd:display=7'b0111101;
    'he:display=7'b1001111;
    'hf:display=7'b1000111;
```

```

        default display=7'b1111111;
    endcase

    always @(*)
    begin
        data[0]=number[15:12];
        data[1]=number[11:8];
        data[2]=number[7:4];
        data[3]=number[3:0];
        leds=ds;                //To see the switch position on leds
    end

    always @(posedge clk1[20])
    begin
        case (state)
            4'b0000:
                begin
                    if (switch==4'b0001)
                        begin
                            number<=number+1;
                            state<=4'b0001;
                        end
                    else if (switch==4'b0010)
                        begin
                            number<=number+2;
                            state<=4'b0010;
                        end
                    else if (switch==4'b0100)
                        begin
                            number<=number+4;
                            state<=4'b0100;
                        end
                    else if (switch==4'b1000)
                        begin
                            number<=number+8;
                            state<=4'b1000;
                        end
                end
            4'b0001:
                begin
                    if (switch==4'b0000)
                        state<=4'b0000;
                    else
                        state<=4'b0001;
                end
            4'b0010:
                begin
                    if (switch==4'b0000)
                        state<=4'b0000;
                    else
                        state<=4'b0010;
                end
        end
    end

```

```

        4'b0100:
            begin
                if (switch==4'b0000)
                    state<=4'b0000;
                else
                    state<=4'b0100;
            end
        4'b1000:
            begin
                if (switch==4'b0000)
                    state<=4'b0000;
                else
                    state<=4'b1000;
            end
        default: state<=4'b0000;
    endcase
end

initial begin
    number=0;
    state=4'b0000;
    count=0;
    grounds=4'b1110;
    clk1=0;
end
endmodule

```

Example 14: This code also works as previous example but here switch position is not important. Whenever turn on any switch, the value is added. Here we define a module “checkdipswitch” that will decide to add number or not. We instantiate it 4 times. In each instantiation we check one of the switches. For bouncing problem we have used slower clock in checkdipswitch module.

```
module dipswitches (ds, leds, display, grounds,clk);
output reg [3:0] leds;
output reg [6:0] display;
output reg [3:0] grounds;
input [3:0] ds;
input clk;

reg [3:0] data [3:0] ; //number to be printed on display
reg [1:0] count;      //which data byte to display.
reg [25:0] clk1;
reg [15:0] number;
wire [3:0] switch;

assign switch=ds;
wire [3:0] adornnot_main;

always @(posedge clk)
    clk1<=clk1+1;

always @(posedge clk1[15]) //25 slow //19 wavy //15 perfect
begin
    grounds<={grounds[2:0],grounds[3]};
    count<=count+1;
end

always @(*)
case(data[count])
    0:display=7'b1111110; //starts with a, ends with g
    1:display=7'b0110000;
    2:display=7'b1101101;
    3:display=7'b1111001;
    4:display=7'b0110011;
    5:display=7'b1011011;
    6:display=7'b1011111;
    7:display=7'b1110000;
    8:display=7'b1111111;
    9:display=7'b1111011;
    'ha:display=7'b1110111;
    'hb:display=7'b0011111;
    'hc:display=7'b1001110;
    'hd:display=7'b0111101;
    'he:display=7'b1001111;
    'hf:display=7'b1000111;
    default display=7'b1111111;
endcase
```

```

always @(*)
begin
data[0]=number[15:12];
data[1]=number[11:8];
data[2]=number[7:4];
data[3]=number[3:0];
leds=ds;
end

checkdipswitch inst0(.dipswin(switch[0]), .addornot(addornot_main[0]), .switchnumber(2'b00),
.clk1(clk1),);
checkdipswitch inst1(.dipswin(switch[1]), .addornot(addornot_main[1]), .switchnumber(2'b01),
.clk1(clk1));
checkdipswitch inst2(.dipswin(switch[2]), .addornot(addornot_main[2]), .switchnumber(2'b10),
.clk1(clk1));
checkdipswitch inst3(.dipswin(switch[3]), .addornot(addornot_main[3]), .switchnumber(2'b11),
.clk1(clk1));

always @(posedge clk1[20])
begin
        number<=number + addornot_main;

end

initial begin
number=0;
count=0;
grounds=4'b1110;
clk1=0;
end

endmodule

module checkdipswitch (dipswin, addornot, switchnumber, clk1);
input [25:0] clk1;
input dipswin;
output reg addornot;
input [1:0] switchnumber;
reg state [3:0];

always @(posedge clk1[20])
begin
        case (state[switchnumber])
                1'b0:
                        begin
                                if (dipswin==1)
                                        begin

```

```

                                addornot<=1;
                                state[switchnumber]<=1;
                                end
                                else
                                addornot<=0;
                                end
                                1'b1:
                                begin
                                if (dipswin==0)
                                begin
                                state[switchnumber]<=0;
                                addornot<=0;
                                end
                                else
                                addornot<=0;
                                end
                                endcase
                                end
                                initial
                                begin
                                state[0]=0;
                                state[1]=0;
                                state[2]=0;
                                state[3]=0;
                                end
                                endmodule

```

Example 15: In this example we have added a debouncing module to previous example. Dipswitch signal is the input of debounced module and debounced signal is the output. This output is used in checkdipswitch module as an input.

```
module dipswitches (ds, leds, display, grounds,clk);
output reg [3:0] leds;
output reg [6:0] display;
output reg [3:0] grounds;
input [3:0] ds;
input clk;

reg [3:0] data [3:0] ; //number to be printed on display
reg [1:0] count;      //which data byte to display.
reg [25:0] clk1;
reg [15:0] number;
wire [3:0] switch;
wire [3:0] addornot_main;
wire [3:0] debounced_ds;

assign switch=ds;

always @(posedge clk)
    clk1<=clk1+1;

always @(posedge clk1[15]) //25 slow //19 wavy //15 perfect
begin
    grounds<={grounds[2:0],grounds[3]};
    count<=count+1;
end

always @(*)
case(data[count])
    0:display=7'b1111110; //starts with a, ends with g
    1:display=7'b0110000;
    2:display=7'b1101101;
    3:display=7'b1111001;
    4:display=7'b0110011;
    5:display=7'b1011011;
    6:display=7'b1011111;
    7:display=7'b1110000;
    8:display=7'b1111111;
    9:display=7'b1111011;
    'ha:display=7'b1110111;
    'hb:display=7'b0011111;
    'hc:display=7'b1001110;
    'hd:display=7'b0111101;
    'he:display=7'b1001111;
    'hf:display=7'b1000111;
    default display=7'b1111111;
endcase

always @(*)
```

```
begin
data[0]=number[15:12];
data[1]=number[11:8];
data[2]=number[7:4];
data[3]=number[3:0];
leds=ds;
end

debounced dinst0 (.dsin(switch[0]), .dsout(debounced_ds[0]),.clk(clk));
debounced dinst1 (.dsin(switch[1]), .dsout(debounced_ds[1]),.clk(clk));
debounced dinst2 (.dsin(switch[2]), .dsout(debounced_ds[2]),.clk(clk));
debounced dinst3 (.dsin(switch[3]), .dsout(debounced_ds[3]),.clk(clk));

checkdipswitch inst0(.dipswin(debounced_ds[0]), .addornot(addornot_main[0]),
.switchnumber(2'b00), .clk(clk),);
checkdipswitch inst1(.dipswin(debounced_ds[1]), .addornot(addornot_main[1]),
.switchnumber(2'b01), .clk(clk));
checkdipswitch inst2(.dipswin(debounced_ds[2]), .addornot(addornot_main[2]),
.switchnumber(2'b10), .clk(clk));
checkdipswitch inst3(.dipswin(debounced_ds[3]), .addornot(addornot_main[3]),
.switchnumber(2'b11), .clk(clk));

always @(posedge clk)
begin
    number<=number + addornot_main;
end

initial begin
number=0;
count=0;
grounds=4'b1110;
clk1=0;
end
endmodule

module checkdipswitch (dipswin, addornot, switchnumber, clk);
input clk;
input dipswin;
output reg addornot;
input [1:0] switchnumber;
reg state [3:0];

always @(posedge clk)
begin
    case (state[switchnumber])
        1'b0:
            begin
                if (dipswin==1)
                    begin
                        addornot<=1;
                        state[switchnumber]<=1;
                    end
                else
                    addornot<=0;
            end
        default:
            addornot<=0;
    endcase
end
endmodule
```

```

                                end
                            else
                                addornot<=0;
                            end
                        end
                    1'b1:
                        begin
                            if (dipswin==0)
                                begin
                                    state[switchnumber]<=0;
                                    addornot<=0;
                                end
                            else
                                addornot<=0;
                            end
                        end
                    endcase
                end

                initial begin
                    state[0]=0;
                    state[1]=0;
                    state[2]=0;
                    state[3]=0;
                end
            endmodule

            module debounced(dsin, dsout, clk);
            input dsin;
            input clk;
            output reg dsout ;
            reg ds_sync_0;
            reg ds_sync_1;
            reg [16:0] cnt;

            always @(posedge clk)
                ds_sync_0<=dsin;

            always @(posedge clk)
                ds_sync_1<=ds_sync_0;

            always @(posedge clk)
                begin
                    if (dsout==ds_sync_1)
                        cnt<=0;
                    else
                        begin
                            cnt<=cnt+1;
                            if (cnt==16'hFFFF)
                                dsout<=~dsout;
                        end
                    end
                end
            end
        endmodule

```

Example 16: In this example 7 segment display module is added.

```
module dipswitches (ds, leds, display, grounds,clk);
output reg [3:0] leds;
output wire [6:0] display;
output wire [3:0] grounds;
input [3:0] ds;
input clk;

reg [25:0] clk1;
reg [15:0] number;
wire [3:0] switch;
wire [3:0] addornot_main;
wire [3:0] debounced_ds;

assign switch=ds;
assign leds=ds;

debounced dint0 (.dsin(switch[0]), .dsout(debounced_ds[0]),.clk(clk));
debounced dint1 (.dsin(switch[1]), .dsout(debounced_ds[1]),.clk(clk));
debounced dint2 (.dsin(switch[2]), .dsout(debounced_ds[2]),.clk(clk));
debounced dint3 (.dsin(switch[3]), .dsout(debounced_ds[3]),.clk(clk));

checkdipswitch inst0(.dipswin(debounced_ds[0]), .addornot(addornot_main[0]),
.switchnumber(2'b00), .clk1(clk));
checkdipswitch inst1(.dipswin(debounced_ds[1]), .addornot(addornot_main[1]),
.switchnumber(2'b01), .clk1(clk));
checkdipswitch inst2(.dipswin(debounced_ds[2]), .addornot(addornot_main[2]),
.switchnumber(2'b10), .clk1(clk));
checkdipswitch inst3(.dipswin(debounced_ds[3]), .addornot(addornot_main[3]),
.switchnumber(2'b11), .clk1(clk));

sevensegment ss(.datain(number), .grounds(grounds), .display(display), .clk(clk));

always @(posedge clk)
    number<=number + addornot_main;

initial number=0;
endmodule

module checkdipswitch (dipswin, addornot, switchnumber, clk1);
input clk1;
input dipswin;
output reg addornot;
input [1:0] switchnumber;
reg state [3:0];

always @(posedge clk1)
begin
    case (state[switchnumber])
        1'b0:
```

```

        begin
            if (dipswin==1)
                begin
                    addornot<=1;
                    state[switchnumber]<=1;
                end
            else
                addornot<=0;
            end
        end
1'b1:
        begin
            if (dipswin==0)
                begin
                    state[switchnumber]<=0;
                    addornot<=0;
                end
            else
                addornot<=0;
            end
        end
    endcase
end
initial begin
    state[0]=0;
    state[1]=0;
    state[2]=0;
    state[3]=0;
end
endmodule

```

```

module debounced(dsin, dsout, clk);
input dsin;
output reg dsout;
input clk;

reg ds_sync_0;
reg ds_sync_1;
reg [15:0] cnt;

always @(posedge clk)
    ds_sync_0<=dsin;

always @(posedge clk)
    ds_sync_1<=ds_sync_0;

always @(posedge clk)
begin
    if (dsout==ds_sync_1)
        cnt<=0;
    else
        begin
            cnt<=cnt+1;

```

```

                if (cnt==16'hFFFF)
                    dsout<=~dsout;
            end
        end
    end
endmodule

```

```

module sevensegment(datain, grounds, display, clk);

```

```

    input wire [15:0] datain;
    output reg [3:0] grounds;
    output reg [6:0] display;
    input clk;

```

```

    reg [3:0] data [3:0];
    reg [1:0] count;
    reg [25:0] clk1;

```

```

    always @(posedge clk1[15])
        begin
            grounds <= {grounds[2:0],grounds[3]};
            count <= count + 1;
        end

```

```

    always @(posedge clk)
        clk1 <= clk1 + 1;

```

```

    always @(*)
        case(data[count])
            0:display=7'b1111110; //starts with a, ends with g
            1:display=7'b0110000;
            2:display=7'b1101101;
            3:display=7'b1111001;
            4:display=7'b0110011;
            5:display=7'b1011011;
            6:display=7'b1011111;
            7:display=7'b1110000;
            8:display=7'b1111111;
            9:display=7'b1111011;
            'ha:display=7'b1110111;
            'hb:display=7'b0011111;
            'hc:display=7'b1001110;
            'hd:display=7'b0111101;
            'he:display=7'b1001111;
            'hf:display=7'b1000111;
            default display=7'b1111111;
        endcase

```

```

    always @*
        begin
            data[0] = datain[15:12];
            data[1] = datain[11:8];

```

```
        data[2] = datain[7:4];
        data[3] = datain[3:0];
    end

    initial begin
        count = 2'b0;
        grounds = 4'b1110;
        clk1 = 0;
    end
endmodule
```