01205241 Digital Circuit and Logic Design

Academic Year 2020, First Semester

Mini Project 2

Combinational Logic Circuit Design

Present to

Asst.Prof. Dusit Thanapatay

Presented to

Miss Areeya Yiampanomkhun 6210554709 (sec 450)

Mr. Krit Jaithawin 6210554717 (sec 450)

Mr. Jakkaphob Kongthanarith 6210554725 (sec 450)

	Option		Neat&	Presentation	Report (25%)			
work	Bonus	Option Detail	Safety	Q&A	Method	Item&prices	Prob&Soln	Neat
(50%)	(20%)	(Add or reduce)	(10%)	(15%)	(15%)	(3%)	(5%)	(2%)

PREFACE

This report are part of secound digital project in subject Digital Circuits and Logic Design semester 1/2020 sec 450 presented to Asst prof Dusit Thanapatay. We describe the project process and how can it work. We build 3 modes in this project which begins with ready mode , input mode and display mode.

Areeya Yiampanomkhun

Krit Jaithawin

Jakkaphob Kongthanarith

Index

-preface 2

-ready mode 3

-input mode 4

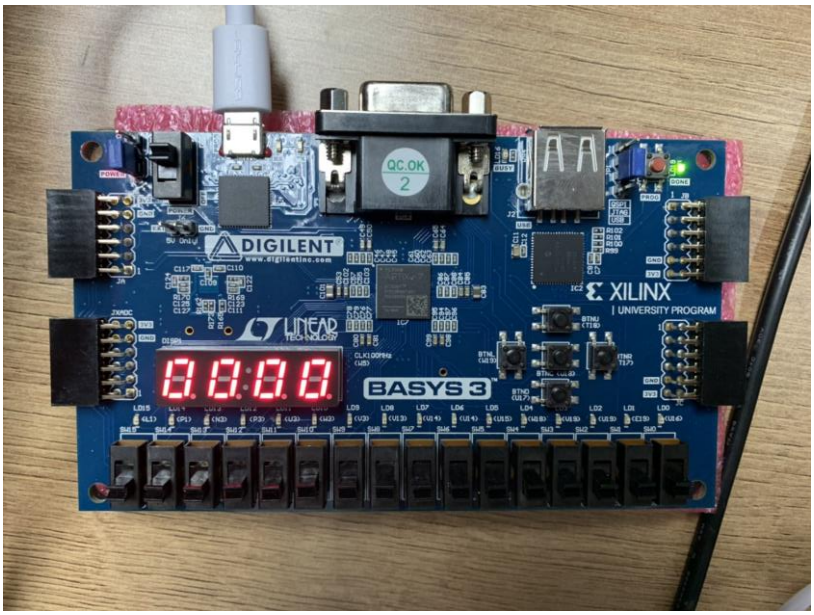
-display mode 5

Ready Mode

-When turn on the power , The circuit begins with **ready mode**

-In this mode , all 4 digits of seven-segment display start blinking 1 Hz of “0000” three times

-After that show “0000” without blinking until there is some assert input.



Code working process :

- Deferment clock form 10 Hz to 1 Hz (1 Hz = 1 sec)

- Count CLK until complete output blinking 3 times then CLK become 10 MHz

(Main Module)

```
`timescale 1ns / 1ps

module seven_segment_count(
input  clk, // Main Clock 25 MHz
input  btnU,btnL,btnR,btnD,btnC,

output [6:0]seg,
output [3:0]an
    );
wire [15:0] out_hex;
reg [15:0] i_hex;
reg [15:0] o_hex;
reg [15:0] main_hex;
wire [15:0] oo_hex;

wire switch;
reg clr;
reg [1:0] state = 2'b00;
wire w_Switch_5;

// apply ok  BTN to the center botton
Debounce_Switch Debounce_Switch_C
(.i_Clk(clk),
.i_Switch(btnC),
.o_Switch(w_Switch_5));

Clock_divider display_blink (
    .clk(clk),
    .divided_clk(switch)
);

Seven_Segment_counter ready_and_input (
.clk(clk),
.btnU(btnU),
.btnL(btnL),
.btnR(btnR),
.btnD(btnD),
.out_hex(out_hex)
);


always@ (posedge clk)
begin
```

First we have to declare CLK, all of tool, input and output

(Code : Blink)

```
3  module Clock_divider(
4  input wire clk,
5  output reg divided_clk);
6
7  localparam divider_value = 49999999;
8  integer Round_c = 0;
9  integer Value_of_counter = 0;
10
11 always@ (posedge clk)
12 begin
13     if (Value_of_counter == divider_value)
14         begin
15         Value_of_counter <= 0;
16         Round_c<=Round_c+1;
17         end
18     else
19         Value_of_counter <= Value_of_counter+1;
20 end
21
22 always@ (posedge clk)
23 begin
24 if(Round_c <= 5)
25     begin
26     if(Value_of_counter == divider_value)
27         divided_clk <= ~divided_clk;
28     else
29         divided_clk <= divided_clk;
30     end
31     else if(Round_c>6)
32         divided_clk = 1;
33 end
34 endmodule
35
```

When we execute the program by clicking ‘Program Device’ the program will run while checking the ‘Value_of_Counter’ valuable, if it reached the value we set (aka ‘ Divider_value’ variable) it will opposite it current state (if the light is on, it will be turned off, on the other hand, if it off it will be turned on) and resetting itself back to 0 will increase value of ‘Round_c’ by 1., otherwise, it will increase value of ‘Value_of_counter’ by 1. This operation will continue until variable ‘Round_c’ value is higher than 6.

(Code : Seven segment display)

```
`timescale 1ns / 1ps
module Display7(

input [15:0] dec,
input clk,
input clr,
input on_off,
output reg [6:0] seg,//LED on seven segment
output reg [3:0] an//we use 7-segment 4 bit
);

wire [1:0] refresh;
reg [3:0] digit;
wire [3:0] an_en;
reg [19:0] clkdiv;

assign refresh = clkdiv[19:18];
assign an_en = 4'b1111; // all turned off from the start

// quad 4to1 MUX.
always @(posedge clk)// or posedge clr)

case(refresh)
0:digit = dec[3:0]; // 00 -->0
1:digit = dec[7:4]; // 01 -->1
2:digit = dec[11:8]; //10 -->2
3:digit = dec[15:12]; //11 -->3

default:digit = dec[3:0];

endcase

always @(*)

case(digit)
0:seg = 7'b1000000;//0000
1:seg = 7'b1111001;//0001
2:seg = 7'b0100100;//0010
3:seg = 7'b0110000;//0011
4:seg = 7'b0011001;//0100
5:seg = 7'b0010010;//0101
6:seg = 7'b0000010;//0110
7:seg = 7'b1111000;//0111
8:seg = 7'b0000000;//1000
9:seg = 7'b0010000;//1001

default: seg = 7'b0000000;

endcase


always @(*)begin
an=4'b1111; //chage from 1111 to 0000
if(an_en[refresh] == 1 && on_off == 1)
an[refresh] = 0;
else if(on_off == 0)
an=4'b1111;
end

always @(posedge clk or posedge clr) begin//clkdiv
if ( clr == 1)
clkdiv <= 0;
else
clkdiv <= clkdiv+1;
end

endmodule
```

This code is use for display the seven segment.

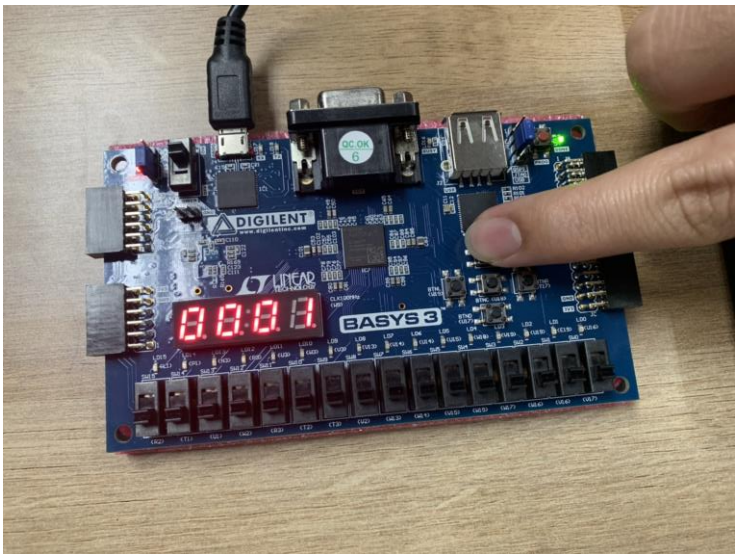
Input Mode

When there is asserted input (S1,S2,S3,S4) the circuit changes into **input mode** for each time when S1,S2,S3 or S4 is pressed, the 7-segment D1,D2,D3 and D4 shows 1 increment on its place in counting order 0-9 numbers

Working process :

- Build the debounce module for protect noise between press switch no error

- Build the variable to keep the data.



When we press the button, value of bit will be increased by 1, starting from 0.When the value increased to max(9). Pressing the button again will make it turn back to 1. This also applied to another button except the center button.

(Code : Debounce)

```
1  `timescale 1ns / 1ps
2  module Debounce_Switch
3  (input i_Clk, input i_Switch, output o_Switch);
4
5  parameter c_DEBOUNCE_LIMIT = 250000; // 10 ms at 25 MHz
6
7  reg [17:0] r_Count = 0;
8  reg r_State = 1'b0;
9
10 always @(posedge i_Clk)
11 begin
12
13 if (i_Switch !== r_State && r_Count < c_DEBOUNCE_LIMIT)
14 r_Count <= r_Count + 1;
15
16 // End of counter reached, switch is stable, register it, reset cou
17 else if (r_Count == c_DEBOUNCE_LIMIT)
18 begin
19 r_State <= i_Switch;//same state
20 r_Count <= 0;//reset counter
21 end
22
23 else
24 r_Count <= 0;// just reset
25
26 assign o_Switch = r_State;
27
28 endmodule
```

This program will erase all the noise that may cause an accident to programs, making it display the wrong input we expect.

(Code : Seven segment counter)

```
`timescale 1ns / 1ps

module Seven_Segment_counter
(input  clk, //25 MHz
input  btnU,btnL,btnR,btnD,
output reg [15:0] out_hex

);
wire switch;
reg clr;

wire w_Switch_1;
wire w_Switch_2;
wire w_Switch_3;
wire w_Switch_4;
reg r_Switch_1 = 1'b0;
reg r_Switch_2 = 1'b0;
reg r_Switch_3 = 1'b0;
reg r_Switch_4 = 1'b0;

reg [3:0] r_Count1 = 4'b0000;
reg [3:0] r_Count2 = 4'b0000;
reg [3:0] r_Count3 = 4'b0000;
reg [3:0] r_Count4 = 4'b0000;


Debounce_Switch Debounce_Switch_U
(.i_Clk(clk),
.i_Switch(btnU),
.o_Switch(w_Switch_1));

Debounce_Switch Debounce_Switch_R
(.i_Clk(clk),
.i_Switch(btnR),
.o_Switch(w_Switch_2));

Debounce_Switch Debounce_Switch_D
(.i_Clk(clk),
.i_Switch(btnD),
.o_Switch(w_Switch_3));

Debounce_Switch Debounce_Switch_L
(.i_Clk(clk),
.i_Switch(btnL),
.o_Switch(w_Switch_4));

// Purpose: When Switch is pressed, increment count.
// When counter gets to 9, turn to 0 again.
always @(posedge clk)
begin
r_Switch_1 <= w_Switch_1;
r_Switch_2 <= w_Switch_2;
r_Switch_3 <= w_Switch_3;
r_Switch_4 <= w_Switch_4;

// Increment Count when switch is pushed down
if (w_Switch_1 == 1'b1 && r_Switch_1 == 1'b0)
    begin
    if (r_Count1 == 9)
        r_Count1 <= 0;
    else
        r_Count1 <= r_Count1 + 1;
    end

else if (w_Switch_2 == 1'b1 && r_Switch_2 == 1'b0)
    begin
    if (r_Count2 == 9)
        r_Count2 <= 0;
    else
        r_Count2 <= r_Count2 + 1;
    end

else if (w_Switch_3 == 1'b1 && r_Switch_3 == 1'b0)
    begin
    if (r_Count3 == 9)
        r_Count3 <= 0;
    else
        r_Count3 <= r_Count3 + 1;
    end

else if (w_Switch_4 == 1'b1 && r_Switch_4 == 1'b0)
    begin
    if (r_Count4 == 9)
        r_Count4 <= 0;
    else
        r_Count4 <= r_Count4 + 1;
    end
end
reg [3:0] num1;

always @(*)
begin
if ((btnU==1) || (btnL==1) || (btnR==1) ||(btnD==1))
out_hex[3:0] = r_Count1;
out_hex[7:4] = r_Count2;
out_hex[11:8] = r_Count3;
out_hex[15:12] = r_Count4;

end
```

This part make program remember which button we want to inscribe our data at.

(Code : Debounce switch)

```
1  `timescale 1ns / 1ps
2  module Debounce_Switch
3  (input i_Clk, input i_Switch, output o_Switch);
4
5  parameter c_DEBOUNCE_LIMIT = 250000; // 10 ms at 25 MHz
6
7  reg [17:0] r_Count = 0;
8  reg r_State = 1'b0;
9
10 always @(posedge i_Clk)
11 begin
12
13 if (i_Switch !== r_State && r_Count < c_DEBOUNCE_LIMIT)
14 r_Count <= r_Count + 1;
15
16 // End of counter reached, switch is stable, register it, reset counter
17 else if (r_Count == c_DEBOUNCE_LIMIT)
18 begin
19 r_State <= i_Switch;//same state
20 r_Count <= 0;//reset counter
21 end
22
23 else
24 r_Count <= 0;// just reset
25
26 assign o_Switch = r_State;
27
28 endmodule
```

This is debounce of each switch. Making the program more accurately and reduce amount of mistake.

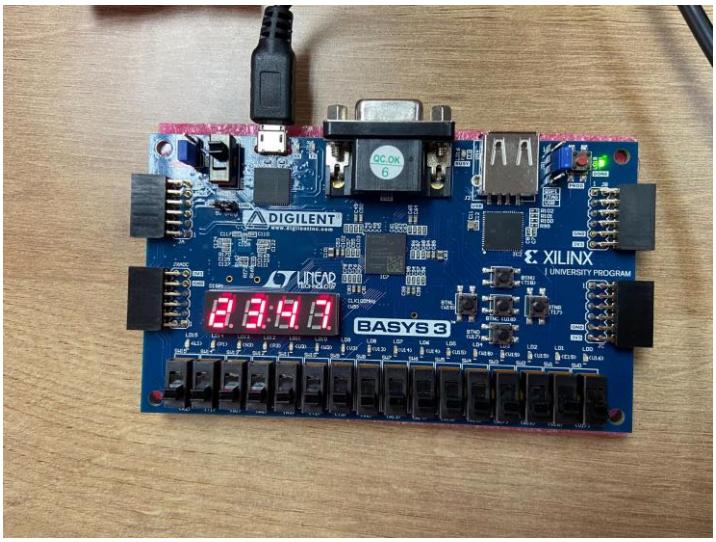
(Code : Sorting)

```
1  `timescale 1ns / 1ps
2  module sorting(
3  input wire clk,
4  input [15:0] i_hex,
5  output reg [15:0] o_hex
6      );
7  wire switch;
8  reg clr;
9  reg [3:0] num1,num2,num3,num4;
10 always@(posedge clk)
11 begin
12 num1 <= i_hex[3:0];
13 num2 <= i_hex[7:4];
14 num3 <= i_hex[11:8];
15 num4 <= i_hex[15:12];
16 end
17
18 integer i,j;
19 reg [3:0] temp;
20 reg [3:0] NUM [1:4];
21 always @*
22 begin
23   NUM[1] = num1;
24   NUM[2] = num2;
25   NUM[3] = num3;
26   NUM[4] = num4;
27   for (i = 5; i > 0; i = i - 1)
28   begin
29     for (j = 1 ; j < i; j = j + 1)
30     begin
31           if (NUM[j] < NUM[j + 1])
32           begin
33             temp = NUM[j];
34             NUM[j] = NUM[j + 1];
35             NUM[j + 1] = temp;
36           end
37     end
38   end
39 end
40 always @(posedge clk)
41 begin
42       o_hex[3:0] <= NUM[1];
43       o_hex[7:4] <= NUM[2];
44       o_hex[11:8] <= NUM[3];
45       o_hex[15:12] <= NUM[4];
46
47 end
48
49
50 endmodule
51
```

This process will compare number stored in each digit. Program will choose the most far-right number and then compare to the number next to the left, put the number that have least value on the left side and continue comparing. When the number of the most far-left side is surely a least value of number, the program will started again until the logic that we have written is all right and rearrange it .

Display Mode

When OK is pressed switch the program will go to this **display mode**



(Code : Display sorting)

```
`timescale 1ns / 1ps

module Display_7seg(

input [15:0] hex,
input clk,
input clr,
input on_off,
output reg [6:0] seg,
output reg [3:0] an
);

wire [1:0] refresh;
reg [3:0] digit;
wire [3:0] an_en;
reg [19:0] clkdiv;

assign refresh = clkdiv[19:18];
assign an_en = 4'b1111; // all turned off from start

// quad 4to1 MUX.
always @(posedge clk)

case(refresh)
0:digit = hex[3:0];
1:digit = hex[7:4];
2:digit = hex[11:8];
3:digit = hex[15:12];

default:digit = hex[3:0];

endcase

always @(*)

case(digit)
0:seg = 7'b1000000;//0000
1:seg = 7'b1111001;//0001
2:seg = 7'b0100100;//0010
3:seg = 7'b0110000;//0011
4:seg = 7'b0011001;//0100
5:seg = 7'b0010010;//0101
6:seg = 7'b0000010;//0110
7:seg = 7'b1111000;//0111
8:seg = 7'b0000000;//1000
9:seg = 7'b0010000;//1001

default: seg = 7'b0000000; // U

endcase


always @(*)begin
an=4'b1111; //chage from 1111 to 0000
if(an_en[refresh] == 1 && on_off == 1)
an[refresh] = 0;
else if(on_off == 0)
an=4'b1111;
end

//clkdiv
always @(posedge clk or posedge clr) begin
if ( clr == 1)
clkdiv <= 0;
else
clkdiv <= clkdiv+1;
end

endmodule
```

The code is used for making bord display value that have been stored in the current variable that we select.

Reference:

https://www.nandland.com/vhdl/modules/binary-to-7-segment.html

https://youtu.be/iei1EugtQvQ

How to Debounce a Switch in an FPGA (nandland.com)

DIGITAL SYSTEM DESIGN WITH FPGA book from Mc graw hill Edutcation