**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH**

**KHOA ĐIỆN ĐIỆN TỬ**

**BỘ MÔN KỸ THUẬT MÁY TÍNH - VIỄN THÔNG**

****

**Thực tập**

**Thiết kế hệ thống số và vi mạch tích hợp**

**NGÀNH CÔNG NGHỆ KỸ THUẬT**

**ĐIỆN TỬ - VIỄN THÔNG**

|  |  |
| --- | --- |
| Sinh viên: | Nguyễn Tấn Đạt |
| MSSV: | 22161110 |

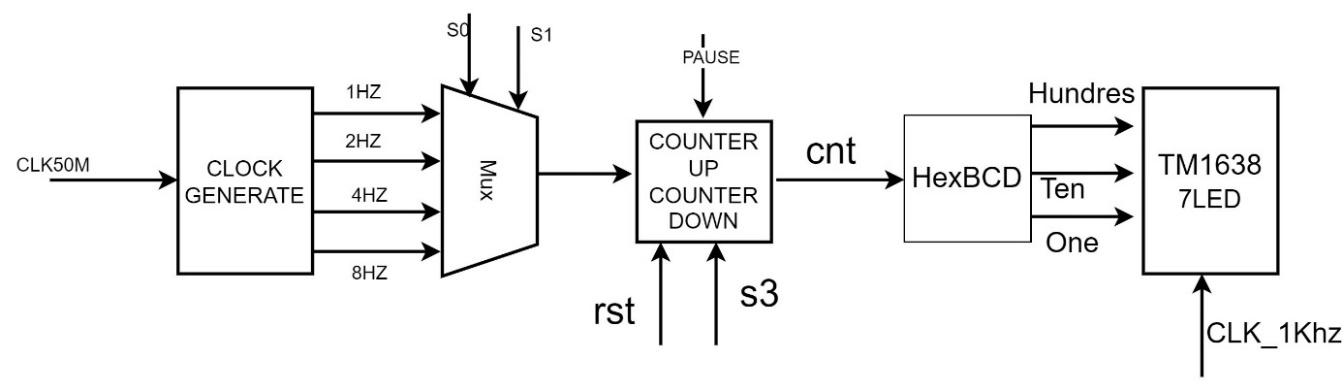
TP. HỒ CHÍ MINH – 12/2024

# 1. BÀI THỰC HÀNH SỐ 1.

## 1.1 Nội dung yêu cầu của bài thực hành.

Thiết kế mạch chia xung với ngõ vào 50 MHz, chia thành 4 xung clock ngõ ra với tần số f, 2f, 4f, 8f trong đó lựa chọn f = 1 Hz, 4 xung clock được hiển thị lên 4 led đơn. Sử dụng các xung clock này để thiết kế mạch đếm 8 bit có lựa chọn tần số đếm bằng 2 switch S1, S0, lựa chọn đếm lên hoặc đếm xuống bằng switch S3. Giá trị đếm hiển thị lên 3 led 7 đoạn trên module TM1638. Có tín hiệu cho phép dừng đếm (Pause).

## 1.2 Sơ đồ khối.

****

## 1.3 RTL code.

module Top(

input clk\_50M,

input wire [1:0] sw,

input wire mode,

input wire pause,

input wire rst,

output wire clk\_hz,

output wire clk,

output wire stb,

output wire dio

);

wire clk\_khz;

wire [7:0] q;

wire [3:0]clko;

wire [3:0] ones,tens,hundreds;

Clock\_div clock (.clk(clk\_50M),.q\_khz(clk\_khz),.q(clko));

Mux\_4\_to\_1 mux (.clk(clko), .sw(sw), .clk\_o(clk\_hz));

counter counter1(.clk(clk\_hz), .mode(mode), .pause(pause),.rst(rst), .q(q));

HexBcd hexbcd (.hex(q), .ones(ones), .tens(tens), .hundreds(hundreds));

wire [4:0] seg[7:0];

TM1638 tm (8'b1, ones, tens, hundreds, 15, 15, 15, 15, 15,

clk\_khz,

clk,

stb,

dio

);

endmodule

module Clock\_div

#(parameter N= 29, M = 50000000)

(input wire clk,

output wire q\_khz,

output wire [3:0]q

);

// signal declaration

reg [N-1:0] r\_reg1Hz,r\_reg2Hz,r\_reg4Hz,r\_reg8Hz,r\_regkhz;

wire [N-1:0] r\_next1Hz,r\_next2Hz,r\_next4Hz,r\_next8Hz, r\_nextkhz;

// body, register

always @(posedge clk) begin

r\_reg1Hz<=r\_next1Hz;

r\_reg2Hz<=r\_next2Hz;

r\_reg4Hz<=r\_next4Hz;

r\_reg8Hz<=r\_next8Hz;

r\_regkhz<=r\_nextkhz;

end

// next state logic

assign r\_next1Hz = (r\_reg1Hz==M)?0:r\_reg1Hz + 1;

assign r\_next2Hz = (r\_reg2Hz==M/2)?0:r\_reg2Hz + 1;

assign r\_next4Hz = (r\_reg4Hz==M/4)?0:r\_reg4Hz + 1;

assign r\_next8Hz = (r\_reg8Hz==M/8)?0:r\_reg8Hz +1;

assign r\_nextkhz = (r\_regkhz== M/1000)?0:r\_regkhz +1;

// output logic

assign q[0]=(r\_reg1Hz<M/2)?0:1;

assign q[1]=(r\_reg2Hz<M/4)?0:1;

assign q[2]=(r\_reg4Hz<M/8)?0:1;

assign q[3]=(r\_reg8Hz<M/16)?0:1;

assign q\_khz = (r\_regkhz<M/2000)?0:1;

endmodule

module Mux\_4\_to\_1

( input wire [3:0] clk,

input wire [1:0] sw,

output reg clk\_o

);

// signal declaration

// clk\_o ;

always @(clk,sw)

case (sw)

0: clk\_o <= clk[0];

1: clk\_o <= clk[1];

2: clk\_o <= clk[2];

3: clk\_o <= clk[3];

endcase

endmodule

module counter

(

input wire clk,

input wire mode,

input wire pause,

input wire rst,

output wire [7:0] q

);

wire set;

wire [7:0] r\_next;

reg [7:0] r\_reg;

initial r\_reg =0 ;

always @(posedge clk)

if(rst)r\_reg <= 0;

else

r\_reg <= r\_next;

assign r\_next = (pause == 1)? r\_reg :((set==1)?0:((mode == 0)? r\_reg +1:r\_reg -1));

assign q=r\_reg;

endmodule

module HexBcd

(

input [7:0] hex,

output wire [3:0] ones,

output wire [3:0] tens,

output wire [3:0] hundreds

);

wire [3:0] c1,c2,c3,c4,c5,c6,c7;

wire [3:0] d1,d2,d3,d4,d5,d6,d7;

assign d1 = {1'b0,hex[7:5]};

assign d2 = {c1[2:0],hex[4]};

assign d3 = {c2[2:0],hex[3]};

assign d4 = {c3[2:0],hex[2]};

assign d5 = {c4[2:0],hex[1]};

assign d6 = {1'b0,c1[3],c2[3],c3[3]};

assign d7 = {c6[2:0],c4[3]};

add3 m1(d1,c1);

add3 m2(d2,c2);

add3 m3(d3,c3);

add3 m4(d4,c4);

add3 m5(d5,c5);

add3 m6(d6,c6);

add3 m7(d7,c7);

assign ones = {c5[2:0],hex[0]};

assign tens = {c7[2:0],c5[3]};

assign hundreds = {c6[3],c7[3]};

endmodule

module add3(in,out);

input [3:0] in;

output [3:0] out;

reg [3:0] out;

always @ (in)

case (in)

4'b0000: out <= 4'b0000;

4'b0001: out <= 4'b0001;

4'b0010: out <= 4'b0010;

4'b0011: out <= 4'b0011;

4'b0100: out <= 4'b0100;

4'b0101: out <= 4'b1000;

4'b0110: out <= 4'b1001;

4'b0111: out <= 4'b1010;

4'b1000: out <= 4'b1011;

4'b1001: out <= 4'b1100;

default: out <= 4'b0000;

endcase

endmodule

module TM1638(

input wire [7:0] led , // 8 leds

input wire [3:0] seg7,seg6,seg5,seg4,seg3,seg2,seg1,seg0 ,//4 bit data for cathode commond LED

input clkinput,

output reg clk,

output reg stb,

output reg dio

);

/\* Hex-Digit to seven segment LED decoder

Author: Mr. Son

\*/

function [7:0] sseg;

input [3:0] hex;

begin

case (hex)

4'h0: sseg[7:0] = 8'b0111111;

4'h1: sseg[7:0] = 8'b0000110;

4'h2: sseg[7:0] = 8'b1011011;

4'h3: sseg[7:0] = 8'b1001111;

4'h4: sseg[7:0] = 8'b1100110;

4'h5: sseg[7:0] = 8'b1101101;

4'h6: sseg[7:0] = 8'b1111101;

4'h7: sseg[7:0] = 8'b0000111;

4'h8: sseg[7:0] = 8'b1111111;

4'h9: sseg[7:0] = 8'b1101111;

4'hA: sseg[7:0] = 8'b1110111;

4'hB: sseg[7:0] = 8'b1111100;

4'hC: sseg[7:0] = 8'b1011000;

4'hD: sseg[7:0] = 8'b1011110;

4'hE: sseg[7:0] = 8'b1111001;

default : sseg[7:0] = 8'b0000000; // 4'hF

endcase

end

endfunction

integer cs = 0;

integer i ;

reg [7:0] command1 =8'h40, command2 =8'hC0,command3 =8'h8F;

wire [127:0] leddata; // 1,3,5,7,9,11,13,15: single led; 0,2,4,6,8,10,12,14: seg LED (common cathode)

reg [127:0] leddatahold;

assign leddata[0\*8+7:0\*8+0] = sseg(seg0);

assign leddata[2\*8+7:2\*8+0] = sseg(seg1);

assign leddata[4\*8+7:4\*8+0] = sseg(seg2);

assign leddata[6\*8+7:6\*8+0] = sseg(seg3);

assign leddata[8\*8+7:8\*8+0] = sseg(seg4);

assign leddata[10\*8+7:10\*8+0] = sseg(seg5);

assign leddata[12\*8+7:12\*8+0] = sseg(seg6);

assign leddata[14\*8+7:14\*8+0] = sseg(seg7);

assign leddata[1\*8+7:1\*8+0] = led[0] ;

assign leddata[3\*8+7:3\*8+0] = led[1] ;

assign leddata[5\*8+7:5\*8+0] = led[2] ;

assign leddata[7\*8+7:7\*8+0] = led[3] ;

assign leddata[9\*8+7:9\*8+0] = led[4] ;

assign leddata[11\*8+7:11\*8+0] = led[5] ;

assign leddata[13\*8+7:13\*8+0] = led[6] ;

assign leddata[15\*8+7:15\*8+0] = led[7] ;

initial

begin

clk = 1 ;

stb = 1 ;

dio = 0 ;

end

always @(posedge clkinput)

begin

if (cs==0)

begin

stb = 0; // initial tm1638

command1 =8'h40; command2 =8'hC0;command3 =8'h8F;

leddatahold=leddata ;

end

else if ((cs >=1)&&(cs<=16))

begin

dio = command1[0];

clk = ~clk ;

if (clk) command1=command1>>1 ;

end

else if (cs==17)

stb = 1; // stop tm1638

else if (cs==18)

stb = 0; // ready to send the second command

// send second command

else if ((cs >=19)&&(cs<=34))

begin

dio = command2[0];

clk = ~clk ;

if (clk) command2=command2>>1 ;

end

else if ((cs >=35)&&(cs<=290))

begin

dio = leddatahold[0];

clk = ~clk ;

if (clk) leddatahold=leddatahold>>1 ;

end

else if (cs==291)

stb = 1; // stop tm1638 for end of data

else if (cs==292)

stb = 0; // ready to send the third command

// send last command

else if ((cs >=293)&&(cs<=308))

begin

dio = command3[0];

clk = ~clk ;

if (clk) command3=command3>>1 ;

end

else if (cs==309)

stb = 1; // End

else if (cs==310)

cs = -1 ; //repeat

// update cs

cs=cs+1;

end

endmodule

NET "clk\_50M" LOC = "C9" | IOSTANDARD = LVCMOS33;

NET "dio" LOC = "C7" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8;

NET "clk" LOC = "F8" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8;

NET "stb" LOC = "E8" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8;

NET "sw[0]" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP;

NET "sw[1]" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP;

NET "mode" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP;

NET "pause" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP;

NET "rst" LOC = "V16" | IOSTANDARD = LVTTL | PULLDOWN;

NET "clk\_hz" LOC = "F12" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8;

# 2. BÀI THỰC HÀNH SỐ 2.

## 2.1 Nội dung yêu cầu của bài thực hành.

Thiết kế mạch hiển thị lên 8 led đơn gồm các hiển thị sau: Trong đó S1, S2 lựa chọn 1 trong 4 tần số 1hz, 2hz, 4hz, 8hz và S3, S4 lựa chọn mode cho 8 led.

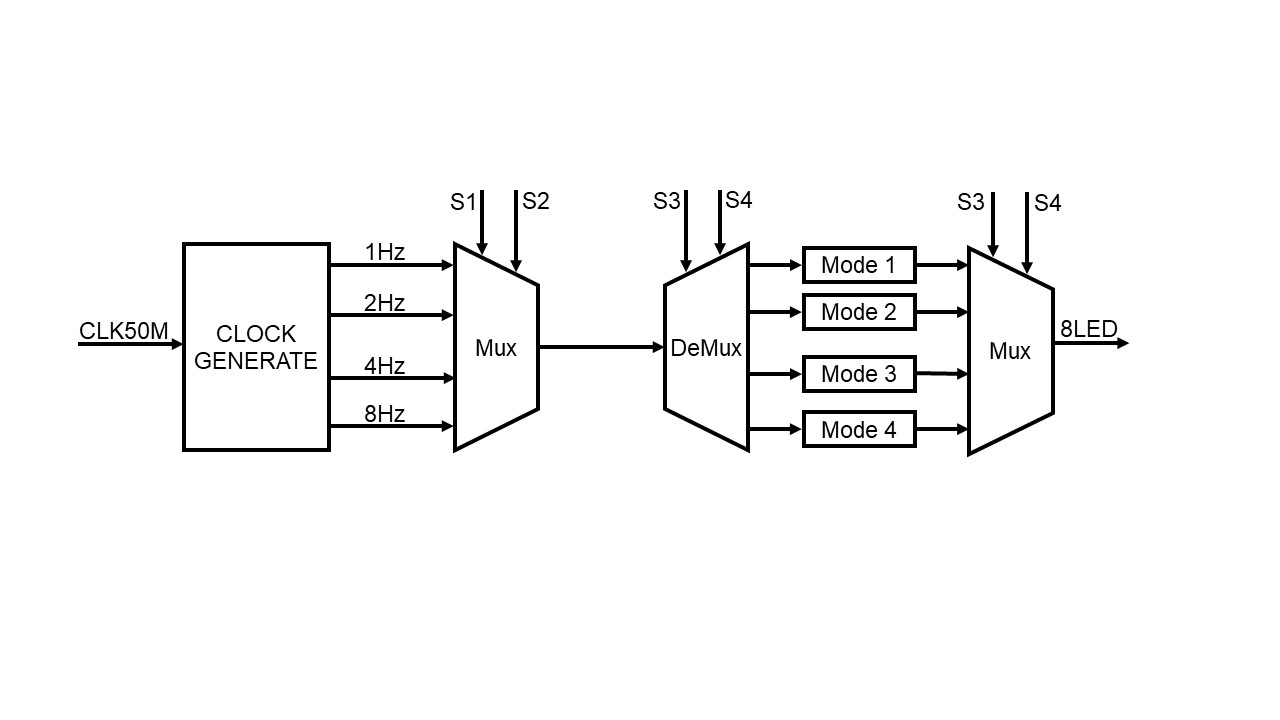
Mode =1: 8 led chớp tắt,

Mode =2: 8 led sáng dần, tắt dần,

Mode = 3: 1 led sáng chạy từ trái sang phải, rồi từ phải sang trái.

Mode = 4: 8 led sáng dồn

## 2.2 Sơ đồ khối.



## 2.3 RTL code.

module Led\_4\_Mode(clk\_50,rst,sw,mode,led);

input clk\_50;

input wire rst;

input wire [1:0] sw;

input wire [1:0] mode;

output wire [7:0] led;

wire clk1,clk2,clk3,clk4;

wire clk\_demux1, clk\_demux2, clk\_demux3, clk\_demux4;

wire clk\_in;

wire [7:0] in1, in2, in3, in4;

clock\_1hz2hz4hz8hz clock ( .clk(clk\_50),.rst(rst),.clkout({clk1, clk2, clk3, clk4}));

Mux\_4\_to\_1 mux (.clk({clk1, clk2, clk3, clk4}),.sw(sw),.clk\_o(clk\_in));

Demux\_1\_to\_4 demux (.clk(clk\_in),.mode(mode),.out1(clk\_demux1),.out2(clk\_demux2),.out3(clk\_demux3),.out4(clk\_demux4));

Mode\_1 mode1 (.clk(clk\_demux1),.rst(rst),.out(in1));

Mode\_2 mode2 (.clk(clk\_demux2),.rst(rst),.out(in2));

Mode\_3 mode3 (.clk(clk\_demux3),.rst(rst),.out(in3));

Mode\_4 mode4 (.clk(clk\_demux4),.rst(rst),.out1(in4[3:0]),.out2(in4[7:4]));

mux4to1 mux4\_1 (.in1(in1),.in2(in2),.in3(in3),.in4(in4),.mode(mode),.out(led));

endmodule

module clock\_1hz2hz4hz8hz(

input clk,

input rst,

output [3:0] clkout

);

clock\_generate gen1hz (clk,rst,clkout[0]);

clock\_generate gen2hz (clk,rst,clkout[1]);

defparam gen2hz.div = 12500000;

clock\_generate gen3hz (clk,rst,clkout[2]);

defparam gen3hz.div = 6250000;

clock\_generate gen4hz (clk,rst,clkout[3]);

defparam gen4hz.div = 3125000;

// clock\_generate gen5hz (clk,rst,clkout[4]);

// defparam gen5hz.div = 1562500;

endmodule

module clock\_generate ( clk\_50,

rstn,

clk\_out

);

parameter div = 24\_999\_999;

input clk\_50;

input rstn;

output clk\_out;

reg clk\_out;

reg [26:0] counter;

wire equal;

assign equal = (counter == div)? 1'b1 : 1'b0;

always @ (posedge clk\_50)

begin

if (!rstn) counter <= 26'd0;

else if (equal) counter <= 26'd0;

else counter <= counter + 26'b1;

end

always @ (posedge clk\_50)

begin

if (!rstn) clk\_out <= 1'd0;

else if (equal) clk\_out <= ~ clk\_out;

end

endmodule

module Mux\_4\_to\_1

( input wire [3:0] clk,

input wire [1:0] sw,

output reg clk\_o

);

// signal declaration

always @(clk,sw)

case (sw)

0: clk\_o <= clk[0];

1: clk\_o <= clk[1];

2: clk\_o <= clk[2];

3: clk\_o <= clk[3];

endcase

endmodule

module Demux\_1\_to\_4(

clk,

out1,

out2,

out3,

out4,

mode

);

input wire clk;

input wire [1:0] mode;

output reg out1,out2,out3,out4;

always @(\*)

begin

case (mode)

2'b00: out1 = clk;

2'b01: out2 = clk;

2'b10: out3 = clk;

2'b11: out4 = clk;

endcase

end

endmodule

module Mode\_1(clk,rst,out);

input clk,rst;

output wire [7:0] out;

wire [7:0] reg\_next;

reg [7:0] reg\_out;

always @(posedge clk or negedge rst)

begin

if (!rst)

reg\_out <= 8'b0;

else

reg\_out <= reg\_next;

end

assign reg\_next = ~ reg\_out;

assign out = reg\_out;

endmodule

module Mode\_2(clk,rst,out);

input wire clk, rst;

output wire [7:0] out;

wire s\_in;

reg [7:0] reg\_out;

wire [7:0] reg\_next;

always @(posedge clk or negedge rst)

begin

if (!rst) reg\_out <= 8'b0;

else reg\_out <= reg\_next;

end

assign s\_in = ~reg\_out[0];

assign reg\_next = {s\_in, reg\_out[7:1]};

assign out= reg\_out;

endmodule

module Mode\_3 (

input wire clk,

input wire rst,

output reg [7:0] out

);

reg tt;

always @(posedge clk or negedge rst)

begin

if (!rst)

out <= 8'b10000000;

else

out <= tt ? (out >> 1) : (out << 1);

end

always @(negedge clk or negedge rst)

begin

if (!rst)

tt <= 1;

else if (out == 8'b10000000)

tt <= 1;

else if (out == 8'b00000001)

tt <= 0;

end

endmodule

module Mode\_4 (

input wire clk,

input wire rst,

output wire [3:0] out1,

output wire [3:0] out2

);

reg tt1;

reg tt2;

reg [3:0] reg\_out1;

reg [3:0] reg\_out2;

wire [3:0] reg\_next1 = tt1 ? (reg\_out1 >> 1) : (reg\_out1 << 1);

always @(posedge clk or negedge rst)

begin

if (!rst)

reg\_out1 <= 4'b1000;

else

reg\_out1 <= reg\_next1;

end

always @(negedge clk or negedge rst)

begin

if (!rst)

tt1 <= 1;

else if (reg\_out1 == 4'b1000)

tt1 <= 1;

else if (reg\_out1 == 4'b0001)

tt1 <= 0;

end

assign out1 = reg\_out1;

wire [3:0] reg\_next2 = tt2 ? (reg\_out2 << 1) : (reg\_out2 >> 1);

always @(posedge clk or negedge rst)

begin

if (!rst)

reg\_out2 <= 4'b0001;

else

reg\_out2 <= reg\_next2;

end

always @(negedge clk or negedge rst)

begin

if (!rst)

tt2 <= 1;

else if (reg\_out2 == 4'b0001)

tt2 <= 1;

else if (reg\_out2 == 4'b1000)

tt2 <= 0;

end

assign out2 = reg\_out2;

endmodule

module mux4to1(in1,

in2,

in3,

in4,

mode,

out

);

input wire [7:0] in1, in2, in3, in4;

input wire [1:0] mode;

output reg [7:0] out;

always @(\*)

begin

case (mode)

2'b00: out = in1;

2'b01: out = in2;

2'b10: out = in3;

2'b11: out = in4;

default: out = in1;

endcase

end

endmodule

NET "clk\_50" LOC = "C9" | IOSTANDARD = LVCMOS33;

NET "led<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

NET "led<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

NET "led<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

NET "led<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

NET "led<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

NET "led<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

NET "led<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

NET "led<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

NET "sw<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;

NET "sw<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP ;

NET "mode<0>" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ;

NET "mode<1>" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP ;

NET "rst" LOC = "V16" | IOSTANDARD = LVTTL | PULLUP ;

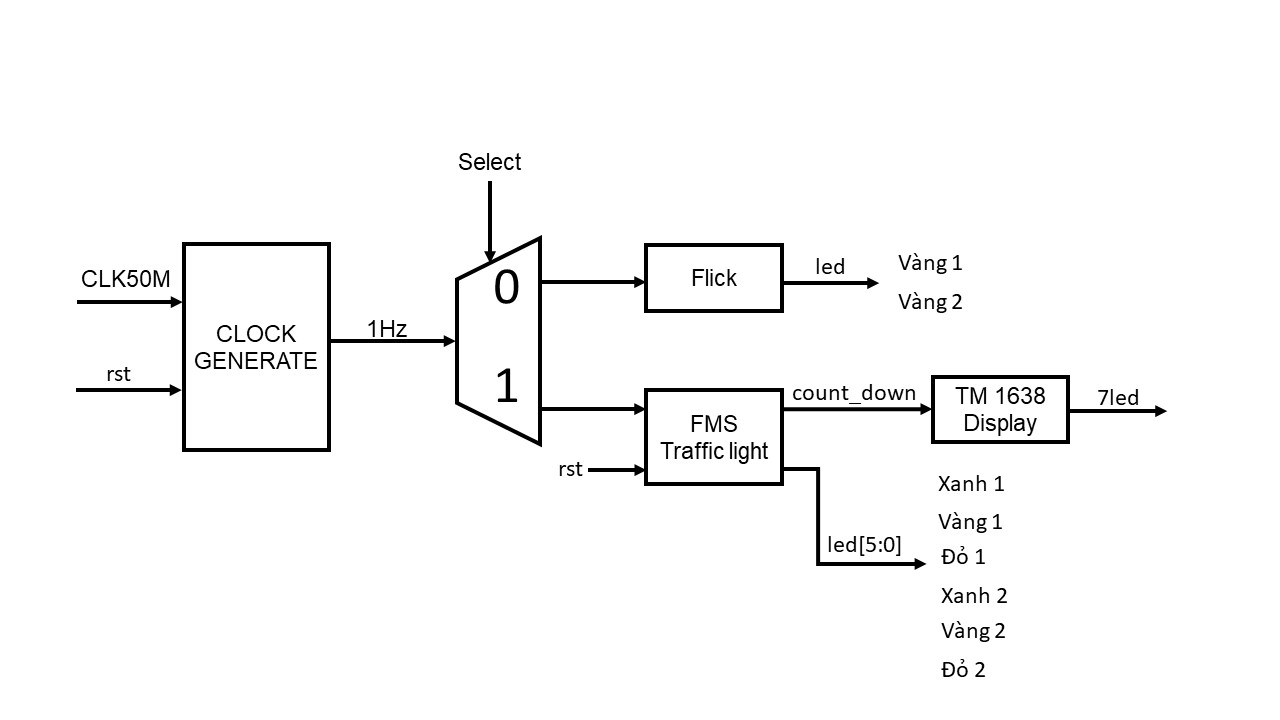
# 3. BÀI THỰC HÀNH SỐ 3.

## 3.1 Nội dung yêu cầu của bài thực hành.

Thiết kế mạch điều khiển đèn giao thông:

Thiết kế mạch điều khiển 2 cột đèn giao tại ngả tư của hai giao lộ, mỗi cột gồm 3 đèn: một đèn xanh, một đèn đỏ và một đèn vàng. Cột đèn 1 có đèn xanh sáng 27 giây, đèn vàng sáng 3 giây, đèn đỏ sáng 20. Cột đèn 2 có đèn xanh sáng 17 giây, đèn vàng sáng 3 giây, đèn đỏ sáng 30 giây. Mạch có bộ đếm thời gian, đếm xuống, hiển thị giá trị đếm xuống trên mỗi cột đèn. Mạch có ngõ vào lựa chọn 2 chế độ: hoạt động bình thường và chế độ chớp tắt với tần số 1hz.

## 3.2 Sơ đồ khối.



## 3.3 RTL code.

module den\_giao\_thong(clk\_50,sw0,X1,V1,D1,X2,V2,D2,nhapnhap,clk,stb,dio);

input clk\_50;

input sw0;

output X1,V1,D1,X2,V2,D2,nhapnhap;

output wire clk;

output wire stb;

output wire dio;

wire clk\_khz,clk\_hz;

//output reg [5:0] Hex0;

//output reg [5:0] Hex1;

wire [5:0] cnt\_up;

wire [5:0] cnt\_down;

wire clock;

wire [3:0] ones1,tens1,hundreds1 ;

wire [3:0] ones2,tens2,hundreds2 ;

chia\_xung CLOCK (.clk\_in(clk\_50),.rst(sw0),.clk\_out(clock));

assign nhapnhay = clock;

ClockDiv div\_clk (.clki(clk\_50),.clk\_khz(clk\_khz),.clk\_hz(clk\_hz));

dieukhien dieu\_khien(.clock (clock),.rst(sw0),.X1 (X1),.D1 (D1),.V1 (V1),.X2 (X2),.D2 (D2),.V2 (V2),.cnt\_up (cnt\_up),.cnt\_down(cnt\_down));

HexBcd hex\_bcd1 (.hex(cnt\_up),.ones1(ones1),.tens1(tens1),.hundreds1(hundreds1));

HexBcd hex\_bcd2 (.hex(cnt\_down),.ones2(ones2),.tens2(tens2),.hundreds2(hundreds2));

wire [4:0] seg [7:0];

TM1638 tm(8'b1,ones1,tens1,hundreds1,15,15,ones2,tens2,hundreds2,

clk\_khz,

clk,

stb,

dio);

endmodule

module chia\_xung(

clk\_in,

rst,

clk\_out

);

parameter div = 24\_999\_999;

input clk\_in;

input rst;

output clk\_out;

reg clk\_out;

reg [26:0] counter;

wire equal;

assign equal = (counter == div)? 1'b1 : 1'b0;

always @ (posedge clk\_in)

begin

if (!rst) counter <= 26'd0;

else if (equal) counter <= 26'd0;

else counter <= counter + 26'b1;

end

always @ (posedge clk\_in)

begin

if (!rst) clk\_out <= 1'd0;

else if (equal) clk\_out <= ~ clk\_out;

end

endmodule

module ClockDiv(

input wire clki,

output wire clk\_khz,clk\_hz

);

wire [26:0] r\_next ;

reg [26:0] r\_reg;

initial begin r\_reg =0 ;end

always @(posedge clki)

r\_reg = r\_next;

assign r\_next =(r\_reg==50000000)?0: r\_reg + 1 ;

assign clk\_khz=r\_reg[5]; /\*781.250 Khz\*/

assign clk\_hz=(r\_reg<=50000000/2)?0:1; /\*781.250 Khz\*/

endmodule

module dieukhien(clock,rst,X1,V1,D1,X2,V2,D2,cnt\_up,cnt\_down);

parameter S0 = 2'b00;

parameter S1 = 2'b01;

parameter S2 = 2'b10;

parameter S3 = 2'b11;

input clock,rst;

output X1,V1,D1,X2,V2,D2;

output cnt\_up;

output cnt\_down;

reg X1,V1,D1,X2,V2,D2;

reg [1:0] state, next\_state;

reg [5:0] cnt\_up;

reg [5:0] cnt\_down;

always @ (posedge clock)

begin

if (!rst) cnt\_up <= 6'd0;

else

case (state)

S0:

begin

if (cnt\_up == 6'd27) cnt\_up <= 6'd0;

else

cnt\_up <= cnt\_up +1'b1;

end

S1:

begin

if (cnt\_up == 6'd3) cnt\_up <= 6'd0;

else

cnt\_up <= cnt\_up +1'b1;

end

S2:

begin

if (cnt\_up == 6'd17) cnt\_up <= 6'd0;

else

cnt\_up <= cnt\_up +1'b1;

end

S3:

begin

if (cnt\_up == 6'd3) cnt\_up <= 6'd0;

else

cnt\_up <= cnt\_up +1'b1;

end

default: cnt\_up <= 6'd0;

endcase

end

// counter\_down

always @ (posedge clock)

begin

if (!rst) cnt\_down <= 6'd27;

else

case (state)

S0:

begin

if (cnt\_down == 6'd0) cnt\_down <= 6'd27;

else

cnt\_down <= cnt\_down - 1'b1;

end

S1:

begin

if (cnt\_down == 6'd0) cnt\_down <= 6'd3;

else

cnt\_down <= cnt\_down - 1'b1;

end

S2:

begin

if (cnt\_down== 6'd0) cnt\_down<= 6'd17;

else

cnt\_down <= cnt\_down -1'b1;

end

S3:

begin

if (cnt\_down == 6'd0) cnt\_down <= 6'd3;

else

cnt\_down <= cnt\_down- 1'b1;

end

default: cnt\_down<= 6'd27;

endcase

end

always @ (\*)

begin

X1 = 1'b0;

D1 = 1'b0;

D1 = 1'b0;

X2 = 1'b0;

D2 = 1'b0;

V2 = 1'b0;

next\_state = S0 ;

case (state)

S0:

begin

X1 = 1'b1;

D2 = 1'b1;

if (cnt\_up == 6'd27) next\_state = S1;

else next\_state = S0;

end

S1:

begin

V1 = 1'b1;

D2 = 1'b1;

if (cnt\_up == 6'd3) next\_state = S2;

else next\_state = S1;

end

S2:

begin

D1 = 1'b1;

X2 = 1'b1;

if (cnt\_up == 6'd17) next\_state = S3;

else next\_state = S2;

end

S3:

begin

D1 = 1'b1;

V2 = 1'b1;

if (cnt\_up== 6'd3) next\_state = S0;

else next\_state = S3;

end

default:

begin

X1 = 1'b1;

D2 = 1'b1;

next\_state = S0;

end

endcase

end

always @(posedge clock)

begin

if (!rst) state <= S0;

else state <= next\_state;

end

endmodule

module HexBcd(

input [7:0] hex,

output wire [3:0] ones1,ones2,

output wire [3:0] tens1,tens2,

output wire [3:0] hundreds1,hundreds2

);

wire [3:0] c1,c2,c3,c4,c5,c6,c7;

wire [3:0] d1,d2,d3,d4,d5,d6,d7;

assign d1 = {1'b0,hex[7:5]};

assign d2 = {c1[2:0],hex[4]};

assign d3 = {c2[2:0],hex[3]};

assign d4 = {c3[2:0],hex[2]};

assign d5 = {c4[2:0],hex[1]};

assign d6 = {1'b0,c1[3],c2[3],c3[3]};

assign d7 = {c6[2:0],c4[3]};

add3 m1(d1,c1);

add3 m2(d2,c2);

add3 m3(d3,c3);

add3 m4(d4,c4);

add3 m5(d5,c5);

add3 m6(d6,c6);

add3 m7(d7,c7);

assign ones = {c5[2:0],hex[0]};

assign tens = {c7[2:0],c5[3]};

assign hundreds = {c6[3],c7[3]};

endmodule

module add3(in,out);

input [3:0] in;

output [3:0] out;

reg [3:0] out;

always @ (in)

case (in)

4'b0000: out <= 4'b0000;

4'b0001: out <= 4'b0001;

4'b0010: out <= 4'b0010;

4'b0011: out <= 4'b0011;

4'b0100: out <= 4'b0100;

4'b0101: out <= 4'b1000;

4'b0110: out <= 4'b1001;

4'b0111: out <= 4'b1010;

4'b1000: out <= 4'b1011;

4'b1001: out <= 4'b1100;

default: out <= 4'b0000;

endcase

endmodule

module TM1638(

input wire [7:0] led , // 8 leds

input wire [3:0] seg7,seg6,seg5,seg4,seg3,seg2,seg1,seg0 ,//4 bit data for cathode commond LED

input clkinput,

output reg clk,

output reg stb,

output reg dio

);

/\* Hex-Digit to seven segment LED decoder

Author: Mr. Son

\*/

function [7:0] sseg;

input [3:0] hex;

begin

case (hex)

4'h0: sseg[7:0] = 8'b0111111;

4'h1: sseg[7:0] = 8'b0000110;

4'h2: sseg[7:0] = 8'b1011011;

4'h3: sseg[7:0] = 8'b1001111;

4'h4: sseg[7:0] = 8'b1100110;

4'h5: sseg[7:0] = 8'b1101101;

4'h6: sseg[7:0] = 8'b1111101;

4'h7: sseg[7:0] = 8'b0000111;

4'h8: sseg[7:0] = 8'b1111111;

4'h9: sseg[7:0] = 8'b1101111;

4'hA: sseg[7:0] = 8'b1110111;

4'hB: sseg[7:0] = 8'b1111100;

4'hC: sseg[7:0] = 8'b1011000;

4'hD: sseg[7:0] = 8'b1011110;

4'hE: sseg[7:0] = 8'b1111001;

default : sseg[7:0] = 8'b0000000; // 4'hF

endcase

end

endfunction

integer cs = 0;

integer i ;

reg [7:0] command1 =8'h40, command2 =8'hC0,command3 =8'h8F;

wire [127:0] leddata; // 1,3,5,7,9,11,13,15: single led; 0,2,4,6,8,10,12,14: seg LED (common cathode)

reg [127:0] leddatahold;

assign leddata[0\*8+7:0\*8+0] = sseg(seg0);

assign leddata[2\*8+7:2\*8+0] = sseg(seg1);

assign leddata[4\*8+7:4\*8+0] = sseg(seg2);

assign leddata[6\*8+7:6\*8+0] = sseg(seg3);

assign leddata[8\*8+7:8\*8+0] = sseg(seg4);

assign leddata[10\*8+7:10\*8+0] = sseg(seg5);

assign leddata[12\*8+7:12\*8+0] = sseg(seg6);

assign leddata[14\*8+7:14\*8+0] = sseg(seg7);

assign leddata[1\*8+7:1\*8+0] = led[0] ;

assign leddata[3\*8+7:3\*8+0] = led[1] ;

assign leddata[5\*8+7:5\*8+0] = led[2] ;

assign leddata[7\*8+7:7\*8+0] = led[3] ;

assign leddata[9\*8+7:9\*8+0] = led[4] ;

assign leddata[11\*8+7:11\*8+0] = led[5] ;

assign leddata[13\*8+7:13\*8+0] = led[6] ;

assign leddata[15\*8+7:15\*8+0] = led[7] ;

initial

begin

clk = 1 ;

stb = 1 ;

dio = 0 ;

end

always @(posedge clkinput)

begin

if (cs==0)

begin

stb = 0; // initial tm1638

command1 =8'h40; command2 =8'hC0;command3 =8'h8F;

leddatahold=leddata ;

end

else if ((cs >=1)&&(cs<=16))

begin

dio = command1[0];

clk = ~clk ;

if (clk) command1=command1>>1 ;

end

else if (cs==17)

stb = 1; // stop tm1638

else if (cs==18)

stb = 0; // ready to send the second command

// send second command

else if ((cs >=19)&&(cs<=34))

begin

dio = command2[0];

clk = ~clk ;

if (clk) command2=command2>>1 ;

end

else if ((cs >=35)&&(cs<=290))

begin

dio = leddatahold[0];

clk = ~clk ;

if (clk) leddatahold=leddatahold>>1 ;

end

else if (cs==291)

stb = 1; // stop tm1638 for end of data

else if (cs==292)

stb = 0; // ready to send the third command

// send last command

else if ((cs >=293)&&(cs<=308))

begin

dio = command3[0];

clk = ~clk ;

if (clk) command3=command3>>1 ;

end

else if (cs==309)

stb = 1; // End

else if (cs==310)

cs = -1 ; //repeat

// update cs

cs=cs+1;

end

endmodule

NET "clk\_50" LOC = "C9" | IOSTANDARD = LVCMOS33;

NET "dio" LOC = "C7" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8;

NET "clk" LOC = "F8" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8;

NET "stb" LOC = "E8" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8;

NET "sw0" LOC = "V16" | IOSTANDARD = LVTTL | PULLUP;

NET "X1" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

NET "V1" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

NET "D1" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

NET "X2" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

NET "V2" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

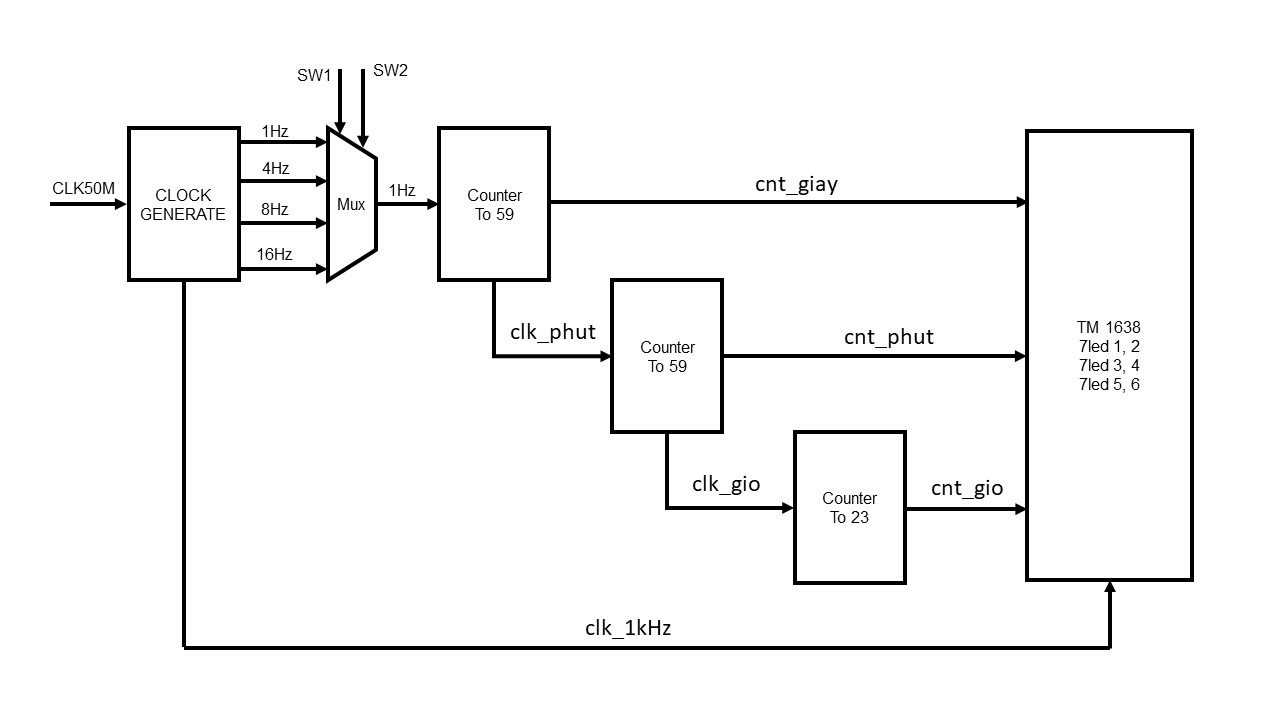
NET "D2" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

# 4. BÀI THỰC HÀNH SỐ 4.

## 4.1 Nội dung yêu cầu của bài thực hành.

Thiết kế mạch đồng hồ.

## 4.2 Sơ đồ khối.



## 4.3 RTL code

module clock\_generate(

input wire clki,

input wire reset,

output wire clk\_1Hz,

output wire clk\_4Hz,

output wire clk\_8Hz,

output wire clk\_16Hz

);

reg [24:0] counter\_1Hz = 0; // tai ngo ra thu q\_n thi f\_in/[2^(n+1)]

reg [22:0] counter\_4Hz = 0;

reg [21:0] counter\_8Hz = 0;

reg [20:0] counter\_16Hz = 0;

always @(posedge clki or posedge reset) begin

if (reset) begin

counter\_1Hz <= 0;

counter\_4Hz <= 0;

counter\_8Hz <= 0;

counter\_16Hz <= 0;

end else

begin

counter\_1Hz <= counter\_1Hz + 1;

counter\_4Hz <= counter\_4Hz + 1;

counter\_8Hz <= counter\_8Hz + 1;

counter\_16Hz <= counter\_16Hz + 1;

end

end

assign clk\_1Hz = counter\_1Hz[24];

assign clk\_4Hz = counter\_4Hz[22];

assign clk\_8Hz = counter\_8Hz[21];

assign clk\_16Hz = counter\_16Hz[20];

endmodule

module multiplexer(

input wire clk\_1Hz,

input wire clk\_4Hz,

input wire clk\_8Hz,

input wire clk\_16Hz,

input wire [1:0] count\_freq\_select,

output wire clk\_selected

);

assign clk\_selected = (count\_freq\_select == 2'b00) ? clk\_1Hz :

(count\_freq\_select == 2'b01) ? clk\_4Hz :

(count\_freq\_select == 2'b10) ? clk\_8Hz :

clk\_16Hz;

endmodule

module CLOCK (

input wire clk\_out,

input wire rst,

input wire pause,

output wire [7:0] giay,

output wire [7:0] phut,

output wire [7:0] gio

);

wire tick\_seconds;

wire tick\_minutes;

giay dem\_giay (clk\_out, rst, pause, giay);

assign tick\_seconds = (giay == 8'd59);// luc dau tick\_second = 0 nhung khi giay dat den 59 giay thi tick\_second = 1, tao thanh xung lock cap vao module phut

phut dem\_phut (tick\_seconds, rst, phut);

assign tick\_minutes = (giay == 8'd59) && (phut == 8'd59);

gio dem\_gio (tick\_minutes, rst, gio); // luc dau tick\_minutes = 0 nhung khi phut dat den 59 phut thi tick\_minutes = 1, tao thanh xung lock cap vao module phut

endmodule

module giay(

input wire clki,

input wire rst,

input wire pause,

output reg [7:0] cnt\_giay

);

initial cnt\_giay =0;

always@(posedge clki or posedge rst)

begin

if(rst) begin

cnt\_giay <= 8'b0;

end

else begin

if(pause) cnt\_giay <= cnt\_giay;

else

begin

if(cnt\_giay == 8'd59)

begin

cnt\_giay <= 8'b0;

end

else

begin

cnt\_giay <= cnt\_giay + 1;

end

end

end

end

endmodule

module phut(

input wire clki,

input wire rst,

output reg [7:0] cnt\_phut

);

initial cnt\_phut = 0;

always@(posedge clki or posedge rst)

begin

if(rst) begin

cnt\_phut <= 8'b0;

end

else begin

if(cnt\_phut == 8'd59)

begin

cnt\_phut <= 8'b0;

end

else

begin

cnt\_phut <= cnt\_phut + 1;

end

end

end

endmodule

module gio(

input wire clki,

input wire rst,

output reg [7:0] cnt\_gio

);

initial cnt\_gio = 0;

always@(posedge clki)

begin

if(rst) cnt\_gio <= 8'b0;

else begin

if(cnt\_gio == 8'd23) begin

cnt\_gio <= 8'b0;

end

else

begin

cnt\_gio <= cnt\_gio + 1;

end

end

end

endmodule

module topmodule(

input wire clk\_50M,

input wire rst,

input wire pause,

input wire [1:0] clock\_ctrl,

output wire clk,

output wire stb,

output wire dio

);

wire clk\_giay, clk\_phut, clk\_gio;

wire [7:0] gio, phut, giay;

clock\_generate chia\_clock (clk\_50M, rst, clk\_1Hz, clk\_4Hz, clk\_8Hz, clk\_16Hz);

multiplexer chon\_clk (clk\_1Hz, clk\_4Hz, clk\_8Hz, clk\_16Hz, clock\_ctrl, clk\_giay);

CLOCK dongho (clk\_giay, rst, pause, giay, phut, gio);

wire clk\_khz,clk\_hz;

wire [3:0] ones1,tens1,hundreds1 ;

wire [3:0] ones2,tens2,hundreds2 ;

wire [3:0] ones3,tens3,hundreds3 ;

wire [4:0] seg [7:0];

ClockDiv clock (clk\_50M, clk\_khz,clk\_hz);

HexBcd hex\_bcd1 (giay,ones1,tens1,hundreds1);

HexBcd hex\_bcd2 (phut,ones2,tens2,hundreds2);

HexBcd hex\_bcd3 (gio,ones3,tens3,hundreds3);

TM1638 tm (8'b11111111,ones1,tens1,15,ones2,tens2,15,ones3,tens3,clk\_khz,

clk,

stb,

dio

);

endmodule

////////////////////////////////////////////////////

module ClockDiv(

input wire clki,

output wire clk\_khz,clk\_hz

);

wire [26:0] r\_next ;

reg [26:0] r\_reg;

initial begin r\_reg =0 ;end

always @(posedge clki)

r\_reg = r\_next;

assign r\_next =(r\_reg==50000000)?0: r\_reg + 1 ;

assign clk\_khz=r\_reg[5]; /\*781.250 Khz\*/

assign clk\_hz=(r\_reg<=50000000/2)?0:1; /\*781.250 Khz\*/

endmodule

//////////////////////////////////////////////////////

module add3(in,out);

input [3:0] in;

output [3:0] out;

reg [3:0] out;

always @ (in)

case (in)

4'b0000: out <= 4'b0000;

4'b0001: out <= 4'b0001;

4'b0010: out <= 4'b0010;

4'b0011: out <= 4'b0011;

4'b0100: out <= 4'b0100;

4'b0101: out <= 4'b1000;

4'b0110: out <= 4'b1001;

4'b0111: out <= 4'b1010;

4'b1000: out <= 4'b1011;

4'b1001: out <= 4'b1100;

default: out <= 4'b0000;

endcase

endmodule

////////////////////////////////////////////////////////

module HexBcd(

input [7:0] hex,

output wire [3:0] ones,

output wire [3:0] tens,

output wire [3:0] hundreds

);

wire [3:0] c1,c2,c3,c4,c5,c6,c7;

wire [3:0] d1,d2,d3,d4,d5,d6,d7;

assign d1 = {1'b0,hex[7:5]};

assign d2 = {c1[2:0],hex[4]};

assign d3 = {c2[2:0],hex[3]};

assign d4 = {c3[2:0],hex[2]};

assign d5 = {c4[2:0],hex[1]};

assign d6 = {1'b0,c1[3],c2[3],c3[3]};

assign d7 = {c6[2:0],c4[3]};

add3 m1(d1,c1);

add3 m2(d2,c2);

add3 m3(d3,c3);

add3 m4(d4,c4);

add3 m5(d5,c5);

add3 m6(d6,c6);

add3 m7(d7,c7);

assign ones = {c5[2:0],hex[0]};

assign tens = {c7[2:0],c5[3]};

assign hundreds = {c6[3],c7[3]};

endmodule

///////////////////////////////////////////////////

module TM1638(

input wire [7:0] led , // 8 leds

input wire [3:0] seg7,seg6,seg5,seg4,seg3,seg2,seg1,seg0 ,//4 bit data for cathodecommond LED

input clkinput,

output reg clk,

output reg stb,

output reg dio

);

/\* Hex-Digit to seven segment LED decoder

Author: Mr. Son

\*/

function [7:0] sseg;

input [3:0] hex;begin

case (hex)

4'h0: sseg[7:0] = 8'b0111111;

4'h1: sseg[7:0] = 8'b0000110;

4'h2: sseg[7:0] = 8'b1011011;

4'h3: sseg[7:0] = 8'b1001111;

4'h4: sseg[7:0] = 8'b1100110;

4'h5: sseg[7:0] = 8'b1101101;

4'h6: sseg[7:0] = 8'b1111101;

4'h7: sseg[7:0] = 8'b0000111;

4'h8: sseg[7:0] = 8'b1111111;

4'h9: sseg[7:0] = 8'b1101111;

4'hA: sseg[7:0] = 8'b1110111;

4'hB: sseg[7:0] = 8'b1111100;

4'hC: sseg[7:0] = 8'b1011000;

4'hD: sseg[7:0] = 8'b1011110;

4'hE: sseg[7:0] = 8'b1111001;

default : sseg[7:0] = 8'b0000000; // 4'hF

endcase

end

endfunction

integer cs = 0;

integer i ;

reg [7:0] command1 =8'h40, command2 =8'hC0,command3 =8'h8F;

wire [127:0] leddata; // 1,3,5,7,9,11,13,15: single led; 0,2,4,6,8,10,12,14: seg LED (commoncathode)

reg [127:0] leddatahold;

assign leddata[0\*8+7:0\*8+0] = sseg(seg0);

assign leddata[2\*8+7:2\*8+0] = sseg(seg1);

assign leddata[4\*8+7:4\*8+0] = sseg(seg2);

assign leddata[6\*8+7:6\*8+0] = sseg(seg3);

assign leddata[8\*8+7:8\*8+0] = sseg(seg4);

assign leddata[10\*8+7:10\*8+0] = sseg(seg5);

assign leddata[12\*8+7:12\*8+0] = sseg(seg6);

assign leddata[14\*8+7:14\*8+0] = sseg(seg7);

assign leddata[1\*8+7:1\*8+0] = led[0] ;

assign leddata[3\*8+7:3\*8+0] = led[1] ;

assign leddata[5\*8+7:5\*8+0] = led[2] ;

assign leddata[7\*8+7:7\*8+0] = led[3] ;

assign leddata[9\*8+7:9\*8+0] = led[4] ;

assign leddata[11\*8+7:11\*8+0] = led[5] ;

assign leddata[13\*8+7:13\*8+0] = led[6] ;

assign leddata[15\*8+7:15\*8+0] = led[7] ;

initial

begin

clk = 1 ;

stb = 1 ;

dio = 0 ;

end

always @(posedge clkinput)

begin

if (cs==0)

begin

stb = 0; // initial tm1638

command1 =8'h40; command2 =8'hC0;command3 =8'h8F;

leddatahold=leddata ;

end

else if ((cs >=1)&&(cs<=16))

begin

dio = command1[0];

clk = ~clk ;

if (clk) command1=command1>>1 ;

end

else if (cs==17)

stb = 1; // stop tm1638

else if (cs==18)

stb = 0; // ready to send the second command

// send second command

else if ((cs >=19)&&(cs<=34))

begin

dio = command2[0];

clk = ~clk ;

if (clk) command2=command2>>1 ;

end

else if ((cs >=35)&&(cs<=290))

begin

dio = leddatahold[0];

clk = ~clk ;

if (clk) leddatahold=leddatahold>>1 ;

end

else if (cs==291)

stb = 1; // stop tm1638 for end of data

else if (cs==292)

stb = 0; // ready to send the third command

// send last command

else if ((cs >=293)&&(cs<=308))

begin

dio = command3[0];

clk = ~clk ;

if (clk) command3=command3>>1 ;

end

else if (cs==309)

stb = 1; // End

else if (cs==310)

cs = -1 ; //repeat

// update cs

cs=cs+1;

end

endmodule

# PlanAhead Generated physical constraints

NET "clock\_ctrl[0]" LOC = L13 | IOSTANDARD = LVTTL | PULLUP;

NET "clock\_ctrl[1]" LOC = L14 | IOSTANDARD = LVTTL | PULLUP;

NET "pause" LOC = H18 | IOSTANDARD = LVTTL | PULLUP;

NET "rst" LOC = N17 | IOSTANDARD = LVTTL | PULLUP;

NET "clk" LOC = F8;

NET "clk\_50M" LOC = C9;

NET "dio" LOC = C7;

NET "stb" LOC = E8;

NET "clk\_50M" IOSTANDARD = LVCMOS33;

NET "clk" IOSTANDARD = LVCMOS33;

NET "stb" IOSTANDARD = LVCMOS33;

NET "dio" IOSTANDARD = LVCMOS33;