**实验二：分频器**

`timescale 1ns / 1ps

module Timer(CLKin,RST,CLK1); //分频模块输出2Hz信号

input CLKin,RST; //系统时钟信号50MHz和复位信号

output CLK1; //分频后的脉冲信号

reg [26:0]counter; //计数器

reg tmp;

always @(negedge RST,posedge CLKin)

if(!RST)

begin

counter<=0;

tmp<=0;

end

else if(counter==12500000-1)

begin

counter<=0;

tmp<=~tmp;

end

else counter<=counter+1;

assign CLK1=tmp; //状态输出

endmodule

//先验证分频模块是否好用

module Div\_freg1(CLK1,CLK2);//将2Hz信号二分频变为1Hz

input CLK1;

output CLK2;

reg[1:0] count;

reg clk\_tmp;

parameter SET\_K=2'd2;

assign CLK2=clk\_tmp;

always@(posedge CLK1)

begin

if(count==SET\_K)

begin

count<=0;

clk\_tmp<=~clk\_tmp;

end

else count<=count+1;

end

endmodule

module Div\_freg2(CLK1,CLK3);//将2Hz信号四分频变为0.5Hz

input CLK1;

output CLK3;

reg[2:0] count;

reg clk\_tmp;

parameter SET\_K=3'd4;

assign CLK3=clk\_tmp;

always@(posedge CLK1)

begin

if(count==SET\_K)

begin

count<=0;

clk\_tmp<=~clk\_tmp;

end

else count<=count+1;

end

endmodule

module Div\_freg3(CLK1,CLK4);//将2Hz信号八分频变为0.25Hz

input CLK1;

output CLK4;

reg[3:0] count;

reg clk\_tmp;

parameter SET\_K=4'd8;

assign CLK4=clk\_tmp;

always@(posedge CLK1)

begin

if(count==SET\_K)

begin

count<=0;

clk\_tmp<=~clk\_tmp;

end

else count<=count+1;

end

endmodule

module Div\_freg4(CLK1,CLK5);//将2Hz信号十六分频变为0.125Hz

input CLK1;

output CLK5;

reg[4:0] count;

reg clk\_tmp;

parameter SET\_K=5'd16;

assign CLK5=clk\_tmp;

always@(posedge CLK1)

begin

if(count==SET\_K)

begin

count<=0;

clk\_tmp<=~clk\_tmp;

end

else count<=count+1;

end

endmodule

module KEY\_select(KEY,CLK2,CLK3,CLK4,CLK5,CLK);//信号选择模块

input [1:0] KEY;

input CLK2,CLK3,CLK4,CLK5;

output reg CLK;

always@(KEY)

begin

case(KEY)

2'b00:CLK<=CLK2;

2'b01:CLK<=CLK3;

2'b10:CLK<=CLK4;

2'b11:CLK<=CLK5;

endcase

end

endmodule

module top\_v(CLKin,RST,CLK,KEY);

input CLKin,RST;

input [1:0]KEY;

output CLK;

wire w0,w1,w2,w3,w4;

Timer U1(.CLKin(CLKin),.RST(RST),.CLK1(w0));

Div\_freg1 U2(.CLK1(w0),.CLK2(w1));

Div\_freg2 U3(.CLK1(w0),.CLK3(w2));

Div\_freg3 U4(.CLK1(w0),.CLK4(w3));

Div\_freg4 U5(.CLK1(w0),.CLK5(w4));

KEY\_select U6(.KEY(KEY),.CLK2(w1),.CLK3(w2),.CLK4(w3),.CLK5(w4),.CLK(CLK));

endmodule

**实验三：12小时模式**

`timescale 1ns / 1ps

module second(clk,clr,en,sech,secl,enmin);//秒钟计数模块

input clr,clk,en;

output [3:0]sech,secl;

output enmin;

reg [7:0]count;

assign sech=count[7:4];

assign secl=count[3:0];

assign enmin=((count==8'h59)&&(en==1'b1))?1:0;

always@(posedge clk,negedge clr)

begin

if(!clr) count<=8'h00;

else if(en)

begin

if(count==8'h59)count<=8'h00;

else if(count[3:0]==9)count<=count+7;

else count<=count+1;

end

end

endmodule

module minute (clk,clr,en,minh,minl,enhr); //分钟计数模块

input clr,clk,en;

output[3:0]minh,minl;

output enhr ;

reg [7:0] count ;

assign minh=count [7:4];//分钟十位

assign minl=count [3:0];//分钟个位

assign enhr=(count==8'h59&&en==1'b1)?1:0;//产生进位信号

always@(posedge clk,negedge clr )

begin

if(!clr) count<=8'h00;//低电平复位,异步复位

else if(en)

begin

if ( count==8'h59) count <=8'h00;//计满60清零

else if ( count [3:0]==9) count <= count +7;// BCD 码调整

else count <=count +1;

end

end

endmodule

module hour(clk,clr,en,hrh,hrl,judge);//时钟计数模块

input clr,clk,en;

output reg judge;//十二小时判断

output [3:0] hrh,hrl;

reg [7:0] count;

assign hrh=count [7:4];//时钟十位

assign hrl=count [3:0];//时钟个位

always@(posedge clk,negedge clr)

begin

if(!clr)count<=8'h00;//低电平复位,异步复位

else if(en )

begin

if (count==8'h12)

begin

if(judge==0)

begin judge<=~judge;count<=8'h01;end//记满12点之后变为下午一点

else if(judge==1)

begin judge<=~judge;count<=8'h01;end//记满下午12点之后变为凌晨1点

end//计满12清零

else if (count [3:0]==9)count<=count+7;// BCD 码调整

else count <=count +1;

end

end

endmodule

module div\_freg(clk,clk1hz,clk\_scan); //分频模块

input clk;//系统时钟50MHz

output clk1hz, clk\_scan;//clk1hz为1Hz, clk \_ scan 为数码管扫描时钟

reg[24:0]count;//计数器,最大计数值为25000000-1

reg clk\_tmp ;

parameter SET\_K =25000-1;//设置分频系数SET\_K

assign clk1hz=clk\_tmp;

assign clk\_scan=count[14];//分频比为2^15

//逻辑功能实现

always@(posedge clk)

begin

if(count==SET\_K)//判断计数值

begin

count<=0;//到达计数值,计数器将会被清零

clk\_tmp<=~clk\_tmp ;//同时改变计数寄存器的状态

end

else count<=count+1;//计数器加1

end

endmodule

module seg\_select(clk,sech,secl,minh,minl,hrh,hrl,seg\_sel,q,judge); //数码管选通模块

input [3:0] sech,secl,minh,minl,hrh,hrl;//待显示的字符

input judge;

input clk;//数码管扫描时钟

output reg [3:0] q;//当前显示的字符

output reg [2:0] seg\_sel;//数码管位选信号

reg [2:0]cnt;

always @( posedge clk )

begin

if(cnt==3'd7)cnt<=0;

else cnt<=cnt+1;

case(cnt)

3'd0: begin q<= secl;seg\_sel<=3'b000; end //扫描秒钟个位

3'd1: begin q<= sech;seg\_sel<=3'b001; end //扫描秒钟十位

3'd2: begin q<= minl;seg\_sel<=3'b010; end //扫描分钟个位

3'd3: begin q<= minh;seg\_sel<=3'b011; end //扫描分钟十位

3'd4: begin q<= hrl;seg\_sel<=3'b100; end //扫描时钟个位

3'd5: begin q<= hrh;seg\_sel<=3'b101; end //扫描时钟十位

3'd6: begin q<= 4'b1010;seg\_sel<=3'b110; end //分隔符

3'd7:

begin

if(judge==0)begin q<=4'b1011;seg\_sel<=3'b111;end

else if(judge==1)begin q<=4'b1100;seg\_sel<=3'b111;end

end //上下午

default: q<= 4'b1111;//数码管共阳，不显示

endcase

end

endmodule

module decoder (q,seg);//数码管显示模块

input [3:0] q;//当前显示的字符

output reg[7:0] seg;//数码管段选信号输出

always@(q)

begin

case(q)

4'b0000:seg<=8'b11000000;//数码管共阳,显示数据"0"

4'b0001:seg<=8'b11111001;//数码管共阳,显示数据"1"

4'b0010:seg<=8'b10100100;//数码管共阳,显示数据"2"

4'b0011:seg<=8'b10110000;//数码管共阳,显示数据"3"

4'b0100:seg<=8'b10011001;//数码管共阳,显示数据"4"

4'b0101:seg<=8'b10010010;//数码管共阳,显示数据"5"

4'b0110:seg<=8'b10000010;//数码管共阳,显示数据"6"

4'b0111:seg<=8'b11111000;//数码管共阳,显示数据"7"

4'b1000:seg<=8'b10000000;//数码管共阳,显示数据"8"

4'b1001:seg<=8'b10010000;//数码管共阳,显示数据"9"

4'b1010:seg<=8'b10111111;//数码管共阳,显示数据"-"

4'b1011:seg<=8'b10001000;//数码管共阳,显示数据"A"

4'b1100:seg<=8'b10001100;//数码管共阳,显示数据"P"

default:seg<=8'b11111111;//数码管共阳,不显示

endcase

end

endmodule

module top\_v(clk,clrn,seg,seg\_sel);//顶层模块

input clk,clrn;//clk为系统时钟输入,clrn 为异步清零端号

output [7:0] seg ; //数码管段选信号输出

output [2:0] seg\_sel ;//数码管位选信号输出

wire w0,w1,w2,w3,w5,w12;

wire[3:0] w4,w6,w7,w8,w9,w10,w11;

assign w2=1;//秒钟模块计数使能端设置为高电平,允许计数

minute U1(.clk(w0),.clr(clrn),.en(w1),.enhr(w3),.minh(w8),.minl(w9));

second U2(.clk(w0),.clr(clrn),.en(w2),.enmin(w1),.sech(w10),.secl(w11));

hour U3(.clk(w0),.clr(clrn),.en(w3),.hrh(w6),.hrl(w7),.judge(w12));

div\_freg U4(.clk(clk),.clk1hz(w0),.clk\_scan(w5));

decoder U5(.q(w4),.seg(seg));

seg\_select U6(.clk(w5),.hrh(w6),.hrl(w7),.minh(w8),.minl(w9),.sech(w10),.secl(w11),.q(w4),.seg\_sel(seg\_sel),.judge(w12));

endmodule

**实验三：24小时模式**

`timescale 1ns / 1ps

module second(clk,clr,en,sech,secl,enmin);//秒钟计数模块

input clr,clk,en;

output [3:0]sech,secl;

output enmin;

reg [7:0]count;

assign sech=count[7:4];

assign secl=count[3:0];

assign enmin=((count==8'h59)&&(en==1'b1))?1:0;

always@(posedge clk,negedge clr)

begin

if(!clr) count<=8'h00;

else if(en)

begin

if(count==8'h59)count<=8'h00;

else if(count[3:0]==9)count<=count+7;

else count<=count+1;

end

end

endmodule

module minute (clk,clr,en,minh,minl,enhr); //分钟计数模块

input clr,clk,en;

output[3:0]minh,minl;

output enhr ;

reg [7:0] count ;

assign minh=count [7:4];//分钟十位

assign minl=count [3:0];//分钟个位

assign enhr=(count==8'h59&&en==1'b1)?1:0;//产生进位信号

always@(posedge clk,negedge clr )

begin

if(!clr) count<=8'h00;//低电平复位,异步复位

else if(en)

begin

if ( count==8'h59) count <=8'h00;//计满60清零

else if ( count [3:0]==9) count <= count +7;// BCD 码调整

else count <=count +1;

end

end

endmodule

module hour(clk,clr,en,hrh,hrl);//时钟计数模块

input clr,clk,en;

output [3:0] hrh,hrl;

reg [7:0] count;

assign hrh=count [7:4];//时钟十位

assign hrl=count [3:0];//时钟个位

always@(posedge clk,negedge clr)

begin

if(!clr)count<=8'h00;//低电平复位,异步复位

else if(en )

begin

if(count==8'h23)count<=8'h00;//计满24清零

else if (count [3:0]==9)count<=count+7;// BCD 码调整

else count <=count +1;

end

end

endmodule

module div\_freg(clk,clk1hz,clk\_scan); //分频模块

input clk;//系统时钟50MHz

output clk1hz, clk\_scan;//clk1hz为1Hz, clk \_ scan 为数码管扫描时钟

reg[24:0]count;//计数器,最大计数值为25000000-1

reg clk\_tmp ;

parameter SET\_K =25000000-1;//设置分频系数SET\_K

assign clk1hz=clk\_tmp;

assign clk\_scan=count[14];//分频比为2^15

//逻辑功能实现

always@(posedge clk)

begin

if(count==SET\_K)//判断计数值

begin

count<=0;//到达计数值,计数器将会被清零

clk\_tmp<=~clk\_tmp ;//同时改变计数寄存器的状态

end

else count<=count+1;//计数器加1

end

endmodule

module seg\_select(clk,sech,secl,minh,minl,hrh,hrl,seg\_sel,q); //数码管选通模块

input [3:0] sech,secl,minh,minl,hrh,hrl;//待显示的字符

input clk;//数码管扫描时钟

output reg [3:0] q;//当前显示的字符

output reg [2:0] seg\_sel;//数码管位选信号

reg [2:0]cnt;

always @( posedge clk )

begin

if(cnt==3'd7)cnt<=0;

else cnt<=cnt+1;

case(cnt)

3'd0: begin q<= secl;seg\_sel<=3'b000; end //扫描秒钟个位

3'd1: begin q<= sech;seg\_sel<=3'b001; end //扫描秒钟十位

3'd2: begin q<= 4'b1010;seg\_sel<=3'b010; end //分隔符

3'd3: begin q<= minl;seg\_sel<=3'b011; end //扫描分钟个位

3'd4: begin q<= minh;seg\_sel<=3'b100; end //扫描分钟十位

3'd5: begin q<= 4'b1010;seg\_sel<=3'b101; end //分隔符

3'd6: begin q<= hrl;seg\_sel<=3'b110; end //扫描时钟个位

3'd7: begin q<= hrh;seg\_sel<=3'b111; end //扫描时钟十位

default: q<= 4'b1111;//数码管共阳，不显示

endcase

end

endmodule

module decoder (q,seg);//数码管显示模块

input [3:0] q;//当前显示的字符

output reg[7:0] seg;//数码管段选信号输出

always@(q)

begin

case(q)

4'b0000:seg<=8'b11000000;//数码管共阳,显示数据"0"

4'b0001:seg<=8'b11111001;//数码管共阳,显示数据"1"

4'b0010:seg<=8'b10100100;//数码管共阳,显示数据"2"

4'b0011:seg<=8'b10110000;//数码管共阳,显示数据"3"

4'b0100:seg<=8'b10011001;//数码管共阳,显示数据"4"

4'b0101:seg<=8'b10010010;//数码管共阳,显示数据"5"

4'b0110:seg<=8'b10000010;//数码管共阳,显示数据"6"

4'b0111:seg<=8'b11111000;//数码管共阳,显示数据"7"

4'b1000:seg<=8'b10000000;//数码管共阳,显示数据"8"

4'b1001:seg<=8'b10010000;//数码管共阳,显示数据"9"

4'b1010:seg<=8'b10111111;//数码管共阳,显示数据"-"

default:seg<=8'b11111111;//数码管共阳,不显示

endcase

end

endmodule

module top\_v(clk,clrn,seg,seg\_sel);//顶层模块

input clk,clrn;//clk为系统时钟输入,clrn 为异步清零端号

output [7:0] seg ; //数码管段选信号输出

output [2:0] seg\_sel ;//数码管位选信号输出

wire w0,w1,w2,w3,w5;

wire[3:0] w4,w6,w7,w8,w9,w10,w11;

assign w2=1;//秒钟模块计数使能端设置为高电平,允许计数

minute U1(.clk(w0),.clr(clrn),.en(w1),.enhr(w3),.minh(w8),.minl(w9));

second U2(.clk(w0),.clr(clrn),.en(w2),.enmin(w1),.sech(w10),.secl(w11));

hour U3(.clk(w0),.clr(clrn),.en(w3),.hrh(w6),.hrl(w7));

div\_freg U4(.clk(clk),.clk1hz(w0),.clk\_scan(w5));

decoder U5(.q(w4),.seg(seg));

seg\_select U6(.clk(w5),.hrh(w6),.hrl(w7),.minh(w8),.minl(w9),.sech(w10),.secl(w11),.q(w4),.seg\_sel(seg\_sel));

endmodule

**实验四：红外传感器控制步进电机**

//---------------------------------------------------------------------------

//-- 描述 : 红外驱动模块

//---------------------------------------------------------------------------

/\*Timing control.

`define HEAD\_HIGH 24'h6\_DDD0 // 9.000ms @ 50MHz, standard is 24'h6\_DDD0.

`define HEAD\_LOW 24'h3\_6EE8 // 4.500ms @ 50MHz, standard is 24'h3\_6EE8

`define BIT\_0\_HIGH 24'h6D60 // 0.560ms @ 50MHz, standard is 24'h6D60

`define BIT\_0\_LOW 24'h6E5A // 0.565ms @ 50MHz, standard is 24'h6E5A

`define BIT\_1\_HIGH 24'h6D60 // 0.560ms @ 50MHz, standard is 24'h6D60

`define BIT\_1\_LOW 24'h1\_4A14 // 1.685ms @ 50MHz, standard is 24'h1\_4A14

`define REP\_HEAD\_HIGH 24'h6\_DDD0 // 9.000ms @ 50MHz, standard is 24'h6\_DDD0.

`define REP\_HEAD\_LOW 24'h1\_B774 // 2.250ms @ 50MHz, standard is 24'h1\_B774

`define REP\_BIT\_HIGH 24'h6D60 // 0.560ms @ 50MHz, standard is 24'h6D60

`define REP\_BIT\_LOW 24'hF4240 // 20.00ms @ 50MHz, standard is 24'hF4240 \*/

//high\_time 为红外引导码9ms低电平，(high\_time[23:14] == `HEAD\_HIGH)

//24'h6\_DDD0 = (0110\_11)01\_1101\_1101\_0000 的[23:14]是(01\_1011) = 10'h1B

//1B+14个0=6C000 X 20ns = 8.85ms

`define HEAD\_HIGH 10'h1B //(0110\_11)01\_1101\_1101\_0000 约 8.85ms

`define HEAD\_LOW 10'hD //(0011\_01)10\_1110\_1110\_1000 约 4.26ms

`define BIT\_0\_HIGH 10'h1 //(0000\_01)10\_1101\_0110\_0000 约 0.33ms

`define BIT\_0\_LOW 10'h1 //(0000\_01)10\_1110\_0101\_1010 约 0.33ms

`define BIT\_0\_LOW2 10'h2 //(0000\_10)00\_0000\_0000\_0000 约 0.66ms

`define BIT\_1\_HIGH 10'h1 //(0000\_01)10\_1101\_0110\_0000 约 0.33ms

`define BIT\_1\_LOW 10'h5 //(0001\_01)00\_1010\_0001\_0100 约 1.64ms

`define BIT\_1\_LOW2 10'h4 //(0001\_00)00\_0000\_0000\_0000 约 1.31ms

`define REP\_HEAD\_HIGH 10'h1B //(0110\_11)01\_1101\_1101\_0000 约 8.85ms

`define REP\_HEAD\_LOW 10'h6 //(0001\_10)11\_0111\_0111\_0100 约 1.97ms

`define REP\_BIT\_HIGH 10'h1 //(0000\_01)10\_1101\_0110\_0000 约 0.33ms

`define REP\_BIT\_LOW 10'h3D //(1111\_01)00\_0010\_0100\_0000 约19.99ms

module Ir\_Module

(

//输入端口

CLK\_50M,RST\_N,IR\_DATA,

//输出端口

o\_ir\_data

);

//---------------------------------------------------------------------------

//-- 外部端口声明

//---------------------------------------------------------------------------

input CLK\_50M; //系统时钟

input RST\_N; //系统复位

input IR\_DATA; //红外输入管脚

output [ 7:0] o\_ir\_data; //从红外读出的数据

//---------------------------------------------------------------------------

//-- 内部端口声明

//---------------------------------------------------------------------------

reg [ 3:0] ir\_fsm\_cs; //状态机的当前状态

reg [ 3:0] ir\_fsm\_ns; //状态机的下一个状态

reg [23:0] time\_cnt; //计时器

reg [23:0] time\_cnt\_n; //time\_cnt的下一个状态

reg [23:0] low\_time; //低电平计时器（实际是高电平）

reg [23:0] low\_time\_n; //low\_time的下一个状态

reg [23:0] high\_time; //高电平计时器（实际是低电平）

reg [23:0] high\_time\_n; //high\_time的下一个状态

reg [ 7:0] bit\_cnt; //用来记录8位串行红外数据组成一个字节

reg [ 7:0] bit\_cnt\_n; //bit\_cnt的下一个状态

reg [ 1:0] detect\_edge; //检测边沿寄存器

wire [ 1:0] detect\_edge\_n; //detect\_edge的下一个状态

reg [31:0] ir\_data; //从红外读出的数据

reg [31:0] ir\_data\_n; //ir\_data的下一个状态

reg [31:0] ir\_data\_reg; //红外数据的缓存寄存器

reg [31:0] ir\_data\_reg\_n; //ir\_data\_reg的下一个状态

reg posedge\_reg; //检测上升沿

wire posedge\_reg\_n; //posedge\_reg的下一个状态

wire head\_code; //红外引导码

wire bit\_0\_code; //逻辑0（实际逻辑1）

wire bit\_1\_code; //逻辑1（实际逻辑0）

wire rep\_head\_code; //重复引导码

wire rep\_bit\_code; //重复码

parameter FSM\_IDLE = 4'h0; //空闲状态

parameter FSM\_DATA = 4'h1; //串行数据接收状态

parameter FSM\_DATA\_END = 4'h2; //数据接收完成状态

parameter FSM\_REP\_BIT = 4'h3; //处理重复码状态

parameter FSM\_REP\_BIT\_END = 4'h4; //重复码处理完成状态

//时序电路,用来给detect\_edge寄存器赋值

always @ (posedge CLK\_50M or negedge RST\_N)

begin

if(!RST\_N) //判断复位

detect\_edge <= 2'h0; //初始化detect\_edge值

else

detect\_edge <= detect\_edge\_n; //用来给detect\_edge赋值

end

//组合电路,检测上升沿

assign detect\_edge\_n = {detect\_edge[0] , {~IR\_DATA}};//将红外信号取反并接收

//时序电路,用来给posedge\_reg寄存器赋值

always @ (posedge CLK\_50M or negedge RST\_N)

begin

if(!RST\_N) //判断复位

posedge\_reg <= 1'h0; //初始化posedge\_reg值

else

posedge\_reg <= posedge\_reg\_n; //用来给posedge\_reg赋值

end

//组合电路,判断上升沿,如果detect\_edge等于01,posedge\_reg\_n就置1

assign posedge\_reg\_n = (detect\_edge == 2'b01) ? 1'b1 : 1'b0;

//时序电路,用来给time\_cnt寄存器赋值

always @ (posedge CLK\_50M or negedge RST\_N)

begin

if(!RST\_N) //判断复位

time\_cnt <= 24'h0; //初始化time\_cnt值

else

time\_cnt <= time\_cnt\_n; //用来给time\_cnt赋值

end

//组合电路,计数器用于记录高电平或者低电平的脉冲宽度

always @ (\*)

begin

if(detect\_edge[0] != detect\_edge[1])//判断电平变化

time\_cnt\_n = 24'h0; //如果红外信号发生变化,time\_cnt\_n就从0开始计数

else

time\_cnt\_n = time\_cnt + 24'h1; //否则,time\_cnt就加1

end

//时序电路,用来给high\_time寄存器赋值

always @ (posedge CLK\_50M or negedge RST\_N)

begin

if(!RST\_N) //判断复位

high\_time <= 24'h0; //初始化high\_time值

else

high\_time <= high\_time\_n; //用来给high\_time赋值

end

//组合电路,实际记录的是IR\_DATA上的低电平宽度，因为上面对IR\_DATA做了一次取反操作

always @ (\*)

begin

if(detect\_edge == 2'b10) //判断下降沿

high\_time\_n = time\_cnt; //如果判断为下降沿，则开始计数

else

high\_time\_n = high\_time; //否则保持不变

end

//时序电路,用来给low\_time寄存器赋值

always @ (posedge CLK\_50M or negedge RST\_N)

begin

if(!RST\_N) //判断复位

low\_time <= 24'h0; //初始化low\_time值

else

low\_time <= low\_time\_n; //用来给low\_time赋值

end

//组合电路,实际记录的是IR\_DATA上的高电平宽度，因为上面对IR\_DATA做了一次取反操作

always @ (\*)

begin

if(IR\_DATA) //判断高电平

low\_time\_n = time\_cnt; //如果判断为高电平，则开始计数

else

low\_time\_n = low\_time; //当IR\_DATA变成0时就保持不变

end

//低电平至少8.85ms,高电平至少4.26ms,就被认为是引导码,head\_code就为1

assign head\_code = (high\_time[23:14] == `HEAD\_HIGH) && (low\_time[23:14] == `HEAD\_LOW) && posedge\_reg;

//低电平至少0.33ms,高电平至少0.33ms或者0.66ms,被认为是逻辑"0",bit\_0\_code就为1

assign bit\_0\_code = (high\_time[23:14] == `BIT\_0\_HIGH) && ((low\_time[23:14] == `BIT\_0\_LOW) || (low\_time[23:14] == `BIT\_0\_LOW2)) && posedge\_reg;

//低电平至少0.33ms,高电平至少1.31ms或者1.66ms,被认为是逻辑"1",bit\_1\_code就为0

assign bit\_1\_code = (high\_time[23:14] == `BIT\_1\_HIGH) && ((low\_time[23:14] == `BIT\_1\_LOW) || (low\_time[23:14] == `BIT\_1\_LOW2)) && posedge\_reg;

//重复引导码

assign rep\_head\_code = (high\_time[23:14] == `REP\_HEAD\_HIGH) && (low\_time[23:14] == `REP\_HEAD\_LOW) && posedge\_reg;

//重复码

assign rep\_bit\_code = (high\_time[23:14] == `REP\_BIT\_HIGH) && (low\_time[23:14] == `REP\_BIT\_LOW) && posedge\_reg;

//时序电路,用来给bit\_cnt赋值的

always @ (posedge CLK\_50M or negedge RST\_N)

begin

if(!RST\_N) //判断复位

bit\_cnt <= 8'h0; //初始化bit\_cnt

else

bit\_cnt <= bit\_cnt\_n; //用来给bit\_cnt赋值

end

//组合电路,用来记录8位串行红外数据组成一个字节

always @ (\*)

begin

if(ir\_fsm\_cs != FSM\_DATA) //判断状态机当前状态是否在接收数据状态

bit\_cnt\_n = 8'h0; //如果不等于,bit\_cnt\_n则清零

else if((ir\_fsm\_cs == FSM\_DATA) && posedge\_reg)//判断状态机当前状态是否在接收数据状态以及是否在上升沿

bit\_cnt\_n = bit\_cnt + 8'h1; //如果条件成立,则记录8位串行红外数据

else

bit\_cnt\_n = bit\_cnt; //否则保持不变

end

//时序电路,用来给ir\_fsm\_cs赋值的

always @ (posedge CLK\_50M or negedge RST\_N)

begin

if(!RST\_N) //判断复位

ir\_fsm\_cs <= FSM\_IDLE; //初始化ir\_fsm\_cs的值

else

ir\_fsm\_cs <= ir\_fsm\_ns; //用来给ir\_fsm\_cs赋值

end

//组合电路,状态机的控制核心

always @ (\*)

begin

case(ir\_fsm\_cs) //判断当前的状态

FSM\_IDLE:

if(head\_code) //收到引导码后

ir\_fsm\_ns = FSM\_DATA; //进入串行接收状态

else if(rep\_head\_code) //收到重复码后

ir\_fsm\_ns = FSM\_REP\_BIT; //进入处理重复码状态

else

ir\_fsm\_ns = ir\_fsm\_cs; //否则保持不变

FSM\_DATA:

if(bit\_cnt == 8'h20) //接收4个字节(地址码，地址反码，命令码，命令反码)

ir\_fsm\_ns = FSM\_DATA\_END; //接收完毕后，进入数据完成状态

else if(rep\_head\_code || head\_code || rep\_bit\_code) //判断重复码

ir\_fsm\_ns = FSM\_IDLE; //进入空闲状态

else

ir\_fsm\_ns = ir\_fsm\_cs; //否则保持不变

FSM\_DATA\_END:

ir\_fsm\_ns = FSM\_IDLE; //进入空闲状态

FSM\_REP\_BIT:

if(rep\_bit\_code) //判断重复码

ir\_fsm\_ns = FSM\_REP\_BIT\_END; //进入重复码处理完成状态

else if(rep\_head\_code || head\_code || bit\_0\_code || bit\_1\_code) //判断重复码

ir\_fsm\_ns = FSM\_IDLE; //进入空闲状态

else

ir\_fsm\_ns = ir\_fsm\_cs; //否则保持不变

FSM\_REP\_BIT\_END:

ir\_fsm\_ns = FSM\_IDLE; //进入空闲状态

default:ir\_fsm\_ns = FSM\_IDLE; //进入空闲状态

endcase

end

//时序电路,用来给ir\_data\_reg赋值的

always @ (posedge CLK\_50M or negedge RST\_N)

begin

if(!RST\_N) //判断复位

ir\_data\_reg <= 32'h0; //初始化ir\_data\_reg

else

ir\_data\_reg <= ir\_data\_reg\_n; //用来给ir\_data\_reg赋值的

end

//组合电路,记录接收到的串行码32bit,每接收一位，判断是0还是1后移位保存。

always @ (\*)

begin

if(ir\_fsm\_cs == FSM\_IDLE) //判断状态机的状态

ir\_data\_reg\_n = 32'hFFFF;

else if((ir\_fsm\_cs == FSM\_DATA) && (bit\_1\_code))

ir\_data\_reg\_n = {ir\_data\_reg[30:0] , 1'h1};

else if((ir\_fsm\_cs == FSM\_DATA) && (bit\_0\_code))

ir\_data\_reg\_n = {ir\_data\_reg[30:0] , 1'h0};

else

ir\_data\_reg\_n = ir\_data\_reg; //否则保持不变

end

//时序电路,用来给ir\_data赋值的

always @ (posedge CLK\_50M or negedge RST\_N)

begin

if(!RST\_N) //判断复位

ir\_data <= 32'h0; //初始化ir\_data

else

ir\_data <= ir\_data\_n; //用来给ir\_data赋值

end

//组合电路,解码完成的状态就可以读取值了

always @ (\*)

begin

if(ir\_fsm\_ns == FSM\_DATA\_END) //判断状态机的状态

ir\_data\_n = ir\_data\_reg; //解码完成的状态就可以读取值了

else

ir\_data\_n = ir\_data; //否则保持不变

end

assign o\_ir\_data = ir\_data[15:8];

endmodule

//---------------------------------------------------------------------------

//-- 描述 : 数码管静态显示

//---------------------------------------------------------------------------

module SMG

(

//输入端口

VAL,

//输出端口

SMG\_DATA,SMG\_SEL

);

//---------------------------------------------------------------------------

//-- 外部端口声明

//---------------------------------------------------------------------------

input [7:0] VAL; //模块的输入端口VAL

output reg[7:0] SMG\_DATA; //模块的输出端口SMG\_DATA

output reg[2:0] SMG\_SEL; //模块的输出端口SMG\_SEL

always@(VAL) //检测信号输入是否有变化

begin

SMG\_SEL <= 3'b000; //数码管位选为00.底板有三八译码器，经过译码后，选择最右端的数码管

case (VAL) //根据输入信号，数码管显示对应值,比如VAL输入4'h5,数码管显示"5"

8'h68: SMG\_DATA <=8'b11000000; //数码管共阳，显示数据"0"

8'h30: SMG\_DATA <=8'b11111001; //数码管共阳，显示数据"1"

8'h18: SMG\_DATA <=8'b10100100; //数码管共阳，显示数据"2"

8'h7A: SMG\_DATA <=8'b10110000; //数码管共阳，显示数据"3"

8'h10: SMG\_DATA <=8'b10011001; //数码管共阳，显示数据"4"

8'h38: SMG\_DATA <=8'b10010010; //数码管共阳，显示数据"5"

8'h5A: SMG\_DATA <=8'b10000010; //数码管共阳，显示数据"6"

8'h42: SMG\_DATA <=8'b11111000; //数码管共阳，显示数据"7"

8'h4A: SMG\_DATA <=8'b10000000; //数码管共阳，显示数据"8"

8'h52: SMG\_DATA <=8'b10010000; //数码管共阳，显示数据"9"

default: SMG\_DATA <=8'b11111111; //数码管共阳，不显示

endcase

end

endmodule

//---------------------------------------------------------------------------

//-- 描述 : 步进电机

//---------------------------------------------------------------------------

module Step\_Motor(CLK,KEY,Motor);

input CLK;

input [1:0]KEY;// 菠码开关

output reg[3:0]Motor;// 电机端口输出

reg [23:0] CLK\_Count; //电机频率计数器

reg [1:0] State; //电机转的状态

wire Motor\_EN;

wire Direction;

assign Motor\_EN=KEY[0]; //拨码开关DK0控制步进电机使能

assign Direction=KEY[1]; //拨码开关DK1控制步进电机正转还是反转

always@(posedge CLK)

begin

if(Motor\_EN) //电机使能: DKO=1控制步进电机转动、否则停止

begin

if(Direction)//DK1控制步进电机转动方向

State<=State+1'b1;//DK1=1电机正转

else

State<=State-1'b1;//DK1=0,反转

case(State) //控制电机转动步代

2'b00: Motor<=4'b0001;

2'b01: Motor<=4'b0010;

2'b10: Motor<=4'b0100;

2'b11: Motor<=4'b1000;

endcase

end

end

endmodule

//---------------------------------------------------------------------------

//-- 描述 : 分频器1

//---------------------------------------------------------------------------

module Timer1(CLK\_50M,RST\_N,CLK1); //分频模块输出250Hz信号

input CLK\_50M,RST\_N; //系统时钟信号50MHz和复位信号

output CLK1; //分频后的脉冲信号

reg [23:0]counter; //计数器

reg tmp;

always @(negedge RST\_N,posedge CLK\_50M)

if(!RST\_N)

begin

counter<=0;

tmp<=0;

end

else if(counter==24'd200000-1)

begin

counter<=0;

tmp<=~tmp;

end

else counter<=counter+1;

assign CLK1=tmp; //状态输出

endmodule

//---------------------------------------------------------------------------

//-- 描述 : 分频器2

//---------------------------------------------------------------------------

module Timer2(CLK\_50M,RST\_N,CLK2); //分频模块输出125Hz信号

input CLK\_50M,RST\_N; //系统时钟信号50MHz和复位信号

output CLK2; //分频后的脉冲信号

reg [23:0]counter; //计数器

reg tmp;

always @(negedge RST\_N,posedge CLK\_50M)

if(!RST\_N)

begin

counter<=0;

tmp<=0;

end

else if(counter==24'd400000-1)

begin

counter<=0;

tmp<=~tmp;

end

else counter<=counter+1;

assign CLK2=tmp; //状态输出

endmodule

//---------------------------------------------------------------------------

//-- 描述 : 分频器3

//---------------------------------------------------------------------------

module Timer3(CLK\_50M,RST\_N,CLK3); //分频模块输出500Hz信号

input CLK\_50M,RST\_N; //系统时钟信号50MHz和复位信号

output CLK3; //分频后的脉冲信号

reg [23:0]counter; //计数器

reg tmp;

always @(negedge RST\_N,posedge CLK\_50M)

if(!RST\_N)

begin

counter<=0;

tmp<=0;

end

else if(counter==24'd100000-1)

begin

counter<=0;

tmp<=~tmp;

end

else counter<=counter+1;

assign CLK3=tmp; //状态输出

endmodule

//---------------------------------------------------------------------------

//-- 描述 : 信号选择模块

//---------------------------------------------------------------------------

module KEY\_select(KEY,CLK1,CLK2,CLK3,CLK);//信号选择模块

input [7:0] KEY;

input CLK1,CLK2,CLK3;

output reg CLK;

always@(KEY)

begin

case(KEY)

8'h30: CLK=CLK1; //250Hz

8'h18: CLK=CLK2; //125Hz

8'h7A: CLK=CLK3; //500Hz

endcase

end

endmodule

//---------------------------------------------------------------------------

//-- 描述 : 顶层模块

//---------------------------------------------------------------------------

module IR

(

//输入端口

CLK\_50M,RST\_N,IR\_DATA,KEY,

//输出端口

SMG\_DATA,SMG\_SEL,Motor

);

//---------------------------------------------------------------------------

//-- 外部端口声明

//---------------------------------------------------------------------------

input CLK\_50M; //时钟的端口,开发板用的50M晶振

input RST\_N; //复位的端口,低电平复位

input IR\_DATA; //红外端口

input [1:0] KEY;

output [3:0] Motor;

output [7:0] SMG\_DATA; //数码管段选

output [2:0] SMG\_SEL; //数码管位选

//---------------------------------------------------------------------------

//-- 内部端口声明

//---------------------------------------------------------------------------

wire [7:0] o\_ir\_data; //接收到红外的完整数据

wire w0,w1,w2,w3;

//---------------------------------------------------------------------------

//-- 逻辑功能实现

//---------------------------------------------------------------------------

//例化红外模块

Ir\_Module Ir\_Module\_Init

(

.CLK\_50M (CLK\_50M ), //时钟端口

.RST\_N (RST\_N ), //复位端口

.IR\_DATA (IR\_DATA ), //红外端口

.o\_ir\_data (o\_ir\_data ) //接收的红外的完整数据

);

//例化数码管模块

SMG SMG\_int

(

.VAL (o\_ir\_data), //输入提取到的按键值

.SMG\_DATA (SMG\_DATA), //数码管段数据

.SMG\_SEL (SMG\_SEL) //数码管位选

);

Step\_Motor Step\_Motor\_int

(

.CLK (w3),

.KEY (KEY),

.Motor (Motor)

);

Timer1 Timer1\_int

(

.CLK\_50M (CLK\_50M),

.RST\_N (RST\_N),

.CLK1 (w0)

);

Timer2 Timer2\_int

(

.CLK\_50M (CLK\_50M),

.RST\_N (RST\_N),

.CLK2 (w1)

);

Timer3 Timer3\_int

(

.CLK\_50M (CLK\_50M),

.RST\_N (RST\_N),

.CLK3 (w2)

);

KEY\_select KEY\_select\_int

(

.KEY (o\_ir\_data),

.CLK1 (w0),

.CLK2 (w1),

.CLK3 (w2),

.CLK (w3)

);

endmodule