Main

module main(

input clk, red\_ctrl, enable,//clk为系统自带时钟，red\_ctrl接受红外遥控器发射的信号，enable调节小车运动模式

input rl1, rl2, rl3, rl4,//红外避障和悬空检测的信号，各两个

input [2:0] speed\_ctrl,//控制小车运动速度

output head\_out,//舵机控制信号

output reg [1:0] en,//控制左右轮的两个电机的电源

output reg [3:0] in//in影响控制左右轮的两个电机的运动状态

);

reg do\_auto, do\_ctrl;//小车两种运动模式工作与否

wire [1:0] en\_auto, en\_ctrl;//小车两种运动模式的电机开关信号

wire [3:0] in\_auto, in\_ctrl;//小车两种运动模式的电机控制信号

reg [2:0] head\_order=3'b100;//初始舵机方向为90°

wire head\_enable=1;//红外控制的使能信号

wire [7:0] data;//红外控制信号转换的指令

wire led;//红外控制接收的信号

red\_receive rr0(clk, head\_enable, red\_ctrl, data, led);//连接红外控制模块

automove auto(clk, do\_auto, red\_ctrl, rl1, rl2, rl3, rl4, en\_auto, in\_auto);//连接自走模块

contralmove ctrl(clk, do\_ctrl, speed\_ctrl, rl1, rl2, red\_ctrl, en\_ctrl, in\_ctrl);//连接即时操作模块

head\_ctrl head(clk,head\_order,head\_out);//连接舵机模块

always@(posedge clk) begin

if(enable==0) begin//切换为即时操作模式

do\_auto<=0;

do\_ctrl<=1;

en<=en\_ctrl;

in<=in\_ctrl;

end else begin//切换为自走模式

do\_auto<=1;

do\_ctrl<=0;

en<=en\_auto;

in<=in\_auto;

end

end

always@(\*) begin//红外遥控舵机方向

case(data)

8'b00011001: head\_order=3'b010;//居中

8'b00010110: head\_order=3'b100;//左转

8'b00001101: head\_order=3'b001;//右转

8'b00011000: head\_order=3'b010;//居中

8'b00001000: head\_order=3'b100;//左转

8'b01011010: head\_order=3'b001;//右转

default: head\_order=3'b010;//居中

endcase

end

endmodule

redlight

module redlight(

input in1, in2, in3, in4,clk,//分别为前方两个对管，下方两个对管的信号及开发板时钟

output reg out//停止运动信号

);

reg red\_avoid, red\_load;//分别为红外避障及悬空检测的返回信号

always@(posedge clk) begin

red\_avoid <= !(in1 & in2);//红外避障的返回信号

red\_load <= in3 | in4;//悬空检测的返回信号

out <= red\_avoid | red\_load;//遇到障碍物或者悬空时小车停止运动

end

endmodule

red­\_receive

//该代码参考自https://www.freesion.com/article/1395552981/

//该代码用于从连续不断的有规律的红外信号中提取出8位指令码，进而判断按下的是哪一个键

//虽然该代码的注释十分详细，但我们还是花了不少精力去理解

//源代码的时钟与板子自带的时钟频率不同，我们修改了和时钟有关的参数，未对其他部分做出修改

//该代码的input rst\_n起到的并不是reset的作用，更贴近于enable的效果

module red\_receive(

input clk,

input rst\_n,

input remote\_in,

output reg [7:0] data,

output reg led

);

parameter IDLE = 6'b00000;

parameter START\_9MS = 6'b00001;

parameter START\_4\_5MS\_OR\_2\_5MS = 6'b00010;

parameter START\_RECEIVE\_DATA = 6'b00011;

parameter START\_REPEAT = 6'b00100;

reg [5:0] state\_c;

reg [5:0] state\_n;

wire sidle29ms\_start;

reg s9ms2s4\_5ms\_or\_2\_5ms\_start;

reg s9ms2idle\_start;

reg s4\_5ms2receive\_data\_start;

reg s4\_5ms2repeat\_code\_start;

reg s4\_5ms2idle;

wire sreceive\_data2idle\_start;

wire srepeat2idle\_start;

//接收数据

reg [31:0] receive\_data; //接收的数据

reg [6:0] cnt;

wire add\_cnt;

wire end\_cnt;

//上升沿，下降沿

wire pos\_remote\_in,neg\_remote\_in;

reg remote\_in\_d0,remote\_in\_d1;

//led控制

reg led\_flag;

wire add\_cnt1;

wire end\_cnt1;

reg [24:0] cnt1;

//计时

reg [20:0] neg\_cnt,pos\_cnt;

//20ns 50个是1us=1000\*50

//9ms

parameter TIME\_10MS = 21'd1000\_000; //50000\*10

parameter TIME\_8MS = 21'd800\_000;

//4.5ms

parameter TIME\_4MS = 21'd400\_000;

parameter TIME\_5MS = 21'd500\_000;

//2.25 逻辑'1'

parameter TIME\_1\_4MS = 21'd140\_000;

parameter TIME\_3MS = 21'd300\_000;

//上升沿检测,下降沿检测,很常规的一种方法

assign pos\_remote\_in = (~remote\_in\_d1) & remote\_in\_d0; //上升沿检测,如果检测到上升沿，有0到1，之后下个周期为0

assign neg\_remote\_in = remote\_in\_d1 & (~remote\_in\_d0); //下降沿

always @(posedge clk or negedge rst\_n)begin

if(rst\_n==1'b0)begin

remote\_in\_d0<=1'b0;

remote\_in\_d1<=1'b0;

end

else begin

remote\_in\_d0<=remote\_in;

remote\_in\_d1<=remote\_in\_d0;

end

end

//状态机初值

always@(posedge clk or negedge rst\_n)begin

if(!rst\_n)begin

state\_c <= IDLE;

end

else begin

state\_c <= state\_n;

end

end

//在接收数据的各种状态

always@(\*)begin

case(state\_c)

IDLE:begin

if(sidle29ms\_start)begin //9ms的下降沿开始，准备开始计数

state\_n = START\_9MS;

end

else begin

state\_n = state\_c;

end

end

START\_9MS:begin

if(s9ms2s4\_5ms\_or\_2\_5ms\_start)begin

state\_n = START\_4\_5MS\_OR\_2\_5MS;

end

else if(s9ms2idle\_start)begin

state\_n = IDLE; //计数出错了

end

else begin

state\_n = state\_c;

end

end

START\_4\_5MS\_OR\_2\_5MS:begin

if(s4\_5ms2receive\_data\_start)begin

state\_n = START\_RECEIVE\_DATA; //开始接收数据

end

else if(s4\_5ms2repeat\_code\_start)begin

state\_n = START\_REPEAT; //重复码

end

else if(s4\_5ms2idle)begin

state\_n = IDLE; //计数出错了

end

else begin

state\_n = state\_c;

end

end

START\_RECEIVE\_DATA:begin

if(sreceive\_data2idle\_start)begin

state\_n = IDLE; //回到空闲状态

end

else begin

state\_n = state\_c;

end

end

START\_REPEAT:begin

if(srepeat2idle\_start)begin

state\_n = IDLE; //回到空闲状态

end

else begin

state\_n = state\_c;

end

end

default:begin

state\_n = IDLE;

end

endcase

end

assign sidle29ms\_start =state\_c==IDLE && neg\_remote\_in; //检测到下降沿，准备开始对9ms低电平计时

assign sreceive\_data2idle\_start =state\_c==START\_RECEIVE\_DATA && end\_cnt; //40位数据接收完成

assign srepeat2idle\_start =state\_c==START\_REPEAT && pos\_remote\_in; //重复码结束标志位

//9MS

always @(posedge clk or negedge rst\_n)begin

if(rst\_n==1'b0)begin

s9ms2s4\_5ms\_or\_2\_5ms\_start<=0;

s9ms2idle\_start<=0;

end

else if(state\_c==START\_9MS && pos\_remote\_in) begin //下降沿开始接收数据

if(neg\_cnt>TIME\_8MS && neg\_cnt<TIME\_10MS) begin

s9ms2s4\_5ms\_or\_2\_5ms\_start<=1;

end

else begin

s9ms2idle\_start<=1;

end

end

else begin

s9ms2s4\_5ms\_or\_2\_5ms\_start<=0;

s9ms2idle\_start<=0;

end

end

//4.5MS

always @(posedge clk or negedge rst\_n)begin

if(rst\_n==1'b0)begin

s4\_5ms2receive\_data\_start<=0;

s4\_5ms2repeat\_code\_start<=0;

s4\_5ms2idle<=0;

end

else if(state\_c==START\_4\_5MS\_OR\_2\_5MS && neg\_remote\_in) begin //下降沿开始接收数据

if(pos\_cnt>TIME\_4MS&&pos\_cnt<TIME\_5MS) begin

s4\_5ms2receive\_data\_start<=1;

end

else if(pos\_cnt>TIME\_1\_4MS&&pos\_cnt<TIME\_3MS)begin

s4\_5ms2repeat\_code\_start<=1;

end

else begin

s4\_5ms2idle<=1;

end

end

else begin

s4\_5ms2receive\_data\_start<=0;

s4\_5ms2repeat\_code\_start<=0;

s4\_5ms2idle<=0;

end

end

///////////////////////////////////接收数据//////////////////////////////////////////////////

//需要接收32位的数据

always @(posedge clk or negedge rst\_n)begin

if(!rst\_n)begin

cnt <= 0;

end

else if(add\_cnt)begin

if(end\_cnt)

cnt <= 0;

else

cnt <= cnt + 1;

end

end

assign add\_cnt = state\_c==START\_RECEIVE\_DATA&&neg\_remote\_in; //成功接收一个

assign end\_cnt = add\_cnt && cnt==32-1;

//接收数据

always @(posedge clk or negedge rst\_n)begin

if(rst\_n==1'b0)begin

receive\_data<=0;

end

else if(state\_c==START\_RECEIVE\_DATA&&neg\_remote\_in) begin

if(pos\_cnt>TIME\_1\_4MS&&pos\_cnt<TIME\_3MS) begin //在2.25ms左右

receive\_data[cnt]<=1;

end

else begin

receive\_data[cnt]<=0;

end

end

end

///////////////////////////////////接收数据////////////////////////////////////////////////////

//下降沿来了,开始计数，这个方法很好用，专门检测低电平的时间

//当低电平来了，低电平neg\_cnt清零并开始自己计数，当上升沿来的那刻，去读neg\_cnt这个值，就是低电平的时间

always @(posedge clk or negedge rst\_n)begin

if(rst\_n==1'b0)begin

neg\_cnt<=0;

end

else begin

if(neg\_remote\_in) //重新清零

neg\_cnt<=0;

else

neg\_cnt<=neg\_cnt+1;

end

end

//上升沿来了,开始计数，

always @(posedge clk or negedge rst\_n)begin

if(rst\_n==1'b0)begin

pos\_cnt<=0;

end

else begin

if(pos\_remote\_in) //重新清零

pos\_cnt<=0;

else

pos\_cnt<=pos\_cnt+1;

end

end

//下面是做测试用的

always @(posedge clk or negedge rst\_n)begin

if(rst\_n==1'b0)begin

led\_flag<=0;

end

else if(srepeat2idle\_start) begin

led\_flag<=1;

end

else if(end\_cnt1) begin

led\_flag<=0;

end

end

always @(posedge clk or negedge rst\_n)begin

if(!rst\_n)begin

cnt1 <= 0;

end

else if(add\_cnt1)begin

if(end\_cnt1)

cnt1 <= 0;

else

cnt1 <= cnt1 + 1;

end

end

assign add\_cnt1 = led\_flag ;

assign end\_cnt1 = add\_cnt1 && cnt1==10000000-1; //100ms

always @(posedge clk or negedge rst\_n)begin

if(rst\_n==1'b0)begin

led<=0;

end

else if(led\_flag&&cnt1<8000000-1)begin //80ms亮

led<=1;

end

else begin

led<=0;

end

end

always @(posedge clk or negedge rst\_n)begin

if(rst\_n==1'b0)begin

data<=0;

end

else if(sreceive\_data2idle\_start) begin

data<=receive\_data[23:16]; //最后接收的数据,将数据显示到数码管上

end

end

endmodule

contralmove

module contralmove(

input clk,//开发板时钟

input enable,//使能信号

input [2:0] speed\_ctrl,//速度控制信号

input rl1,rl2,//悬空检测信号

input red\_ctrl,//红外遥控接收信号

output reg [1:0] en,//电机开关信号

output reg [3:0] in//电机运转信号

);

reg [40:0]count;//计数器

wire stop;//stop表示是否需要暂停

wire [7:0] ctrl;//用于存储红外遥控器发射的8位指令码

wire finish;//finish表示对红外遥控器发射的信号的翻译是否结束

red\_receive rrc1(clk, enable, red\_ctrl, ctrl,finish);//连接红外遥控模块

redlight\_stop r1(rl1, rl2, clk, stop);//连接红外避障模块

reg [2:0] state,next\_state;//控制模块的状态

reg [2:0] run\_state;//运动模块的状态

reg [1:0] speed = 0;//速度档位，共三档

always@(posedge clk)

begin

if(!stop)begin

if(ctrl==8'b01010010 || ctrl==8'b00001000 || ctrl==8'b01011010) begin//使得小车遇到障碍时不能前进，但可以后退、左右转

state<=next\_state;

end else begin

state<=0;

end

end else state<=next\_state;

if(next\_state==2)count<=0;else count<=count+1;//count记时实现小车运动信号的延时

end

always@(\*)

begin

case(state)

0:next\_state=1;

1:next\_state=2;

2:if(finish==1)next\_state=(ctrl==8'b00011000)?3:(ctrl==8'b01010010)?4:(ctrl==8'b00001000)?6:(ctrl==8'b01011010)?5:0;else next\_state=2;//判断跳转到使小车哪个运动状态，4个ctrl的值分别对应红外遥控器的上下左右键，state3、4、5、6分别对应小车前进、后退、左转、右转

3:if(count==10000000)next\_state=0;else next\_state=3;

4:if(count==10000000)next\_state=0;else next\_state=4;

5:if(count==10000000)next\_state=0;else next\_state=5;

6:if(count==10000000)next\_state=0;else next\_state=6;

default:next\_state=0;

endcase

end

always@(posedge clk)

begin

case(next\_state)

0:run\_state<=0;//电机停止

1:run\_state<=0;//电机停止

2:run\_state<=0;//电机停止

3:run\_state<=1;//小车前进

4:run\_state<=2;//小车后退

5:run\_state<=3;//小车左转

6:run\_state<=4;//小车右转

default:run\_state<=0;//小车停止

endcase

end

always@(\*) //通过改变占空比来实现小车运动速度的变化

begin

if( speed == 0 ) begin//全速

case(run\_state)

0:begin en=0;in=0;end//电机关

1:begin en=3;in=4'b1001;end//左电机逆时针，右电机顺时钟

2:begin en=3;in=4'b0110;end//左电机顺时针，右电机逆时钟

3:begin en=3;in=4'b0101;end//左、右电机顺时针

4:begin en=3;in=4'b1010;end//左、右电机逆时针

default:begin en=0;in=0;end//电机关

endcase end

else if( speed == 1 ) begin//半速，以0.1s为周期，电机运行0.05s，静止0.05s

if(count<5000000) begin

case(run\_state)

0:begin en=0;in=0;end

1:begin en=3;in=4'b1001;end

2:begin en=3;in=4'b0110;end

3:begin en=3;in=4'b0101;end

4:begin en=3;in=4'b1010;end

default:begin en=0;in=0;end

endcase end

else en=0;

end

else if( speed == 2 ) begin

if(count<2500000) begin//慢速，以0.1s为周期，电机运行0.025s，静止0.075s

case(run\_state)

0:begin en=0;in=0;end

1:begin en=3;in=4'b1001;end

2:begin en=3;in=4'b0110;end

3:begin en=3;in=4'b0101;end

4:begin en=3;in=4'b1010;end

default:begin en=0;in=0;end

endcase end

else en=0;

end

case(speed\_ctrl)//速度控制信号转换为速度状态

3'b100:speed=0;//全速

3'b010:speed=1;//半速

3'b001:speed=2;//慢速

default:speed=0;

endcase

end

endmodule

module redlight\_stop(

input in1,in2, clk,//前方两个红外对管信号

output reg out//返回到小车

);

always@(posedge clk) begin

out<=in1 & in2;//遇到障碍物时in1、in2返回0信号，in1&in2，当左右各一个的红外检测灯有一个检测到障碍，out就返回1信号

end

endmodule

automove

module automove(

input clk,//开发板时钟

input enable,//使能信号

input redlight,//红外遥控接收信号

input rl1, rl2, rl3, rl4,//红外避障及悬空检测信号

output reg[1:0]en,//电机开关信号

output reg[3:0]in//电机运转信号

);

wire [7:0]data;

reg [7:0]olddata;

wire led;

wire red\_stop;

reg [50:0]timecount;

wire [50:0]actiontime;

reg [4:0]count;

reg [4:0]outcount;

reg [159:0]action;

reg start;

reg oldstart;

reg [2:0]run\_state;

reg [29:0] scount;//运动时间计时器

red\_receive red1(clk,enable,redlight,data,led);//连接红外遥控模块

redlight rr1(rl1, rl2, rl3, rl4, clk, red\_stop);//连接红外避障及悬空检测模块

always@(posedge clk)

begin

oldstart<=start;//用于消除连点同一个键，连发同一个信号的影响

if(start==0)

begin

if(led==1)olddata<=data;//led表示data接受到了一个新的指令码

if(start==0&&oldstart==1)action<=0;//上一个运动周期结束，清空存储的action指令

else if(led==1&&olddata!=data&&(data==8'b00011000||data==8'b01010010||data==8'b00001000||data==8'b01011010||data==8'b01000101||data==8'b01000110||data==8'b01000111))//表示按下了上下左右键或者123键

begin

action[count\*8+7]<=data[7];

action[count\*8+6]<=data[6];

action[count\*8+5]<=data[5];

action[count\*8+4]<=data[4];

action[count\*8+3]<=data[3];

action[count\*8+2]<=data[2];

action[count\*8+1]<=data[1];

action[count\*8]<=data[0];

if(count<20)count<=count+1;//最多20条指令，奇数条记录动作，偶数条记录该动作进行时间

end

end

else begin count<=0;end//清零记录运动指令条数的count，为下次输入运动指令做准备（不会对小车运动造成影响，应为

if(led==1&&olddata!=data&&data==8'b00011100)//表示按下了ok键，开始执行运动指令

begin

start<=1;//表示开始运动的信号

end

else if(outcount==20)start<=0;//执行完所有运动指令，结束运动

if(start)

begin

if(red\_stop) begin

run\_state<=0;//暂停信号为1，强行使小车停止

end

else begin

if(!action[outcount\*8]&&!action[outcount\*8+1]&&!action[outcount\*8+2]&&action[outcount\*8+3]&&action[outcount\*8+4]&&!action[outcount\*8+5]&&!action[outcount\*8+6]&&!action[outcount\*8+7])//判断是否对应上键的指令码

run\_state<=1;

else if(!action[outcount\*8]&&action[outcount\*8+1]&&!action[outcount\*8+2]&&!action[outcount\*8+3]&&action[outcount\*8+4]&&!action[outcount\*8+5]&&action[outcount\*8+6]&&!action[outcount\*8+7])//判断是否对应下键的指令码

run\_state<=2;

else if(!action[outcount\*8]&&!action[outcount\*8+1]&&!action[outcount\*8+2]&&action[outcount\*8+3]&&!action[outcount\*8+4]&&!action[outcount\*8+5]&&!action[outcount\*8+6]&&!action[outcount\*8+7])//判断是否对应左键的指令码

run\_state<=3;

else if(!action[outcount\*8]&&action[outcount\*8+1]&&!action[outcount\*8+2]&&action[outcount\*8+3]&&action[outcount\*8+4]&&!action[outcount\*8+5]&&action[outcount\*8+6]&&!action[outcount\*8+7])//判断是否对应右键的指令码

run\_state<=4;

else

run\_state<=0;

end

end

end

always@(\*)

begin

if(scount<7500000) begin//速度控制，以0.1s为周期，电机正常运行0.075s，静止0.025s，避免速度过快

case(run\_state)

0:begin en=0;in=0;end//静止

1:begin en=3;in=4'b1001;end//前进

2:begin en=3;in=4'b0110;end//后退

3:begin en=3;in=4'b1010;end//左转

4:begin en=3;in=4'b0101;end//右转

default:begin en=0;in=0;end//静止

endcase

end

else begin//静止

en=0;

in=0;

end

end

assign actiontime=(action[outcount\*8+8]&&!action[outcount\*8+9]&&action[outcount\*8+10]&&!action[outcount\*8+11]&&!action[outcount\*8+12]&&!action[outcount\*8+13]&&action[outcount\*8+14]&&!action[outcount\*8+15])?100000000:(!action[outcount\*8+8]&&action[outcount\*8+9]&&action[outcount\*8+10]&&!action[outcount\*8+11]&&!action[outcount\*8+12]&&!action[outcount\*8+13]&&action[outcount\*8+14]&&!action[outcount\*8+15])?200000000:(action[outcount\*8+8]&&action[outcount\*8+9]&&action[outcount\*8+10]&&!action[outcount\*8+11]&&!action[outcount\*8+12]&&!action[outcount\*8+13]&&action[outcount\*8+14]&&!action[outcount\*8+15])?300000000:0;//判断按下的是1，2还是3，获得运动时间

always@(posedge clk)

begin

if(start==0)begin//运动结束时，对记录运动过程的变量进行清零

timecount<=0;

outcount<=0;

end

else if(start==1)begin

if(timecount==actiontime)//运动到指定时间后，进入下一条指令

begin

timecount<=0;

if(outcount<=20) begin

outcount<=outcount+2;

end

end

else begin

if(red\_stop) timecount<=timecount;//停止运动时停止计时

else timecount<=timecount+1;

end

end

end

always@(posedge clk)

begin

if(scount==10000000) scount<=0;//从接电开始计时，以0.1s为一个周期

else scount<=scount+1;

end

endmodule

head\_ctrl

module head\_ctrl(input clk,[2:0] sel,//舵机方向控制，共三个角度

output reg out//舵机控制输出信号，通过信号的占空比实现控制

);//舵机控制

reg [19:0] count=0;//计数器

always@(posedge clk)

begin

if(count==2000000) count<=0;//以0.02s为一个周期

else count<=count+1;

end

always@(\*)

begin

case(sel)

3'b001:begin

if(count>=0&&count<150000) out=1;//占空比7.5%，舵机转向45°

else out=0;

end

3'b010:begin

if(count>=0&&count<200000) out=1;//占空比10%，舵机转向90°

else out=0;

end

3'b100:begin

if(count>=0&&count<250000) out=1;//占空比12.5%，舵机转向135°

else out=0;

end

default:out=0;//不改变当前舵机角度

endcase

end

endmodule