基于 PYNQ 的无人驾驶解决方案

孟凡鑫；吉家国

第一部分 设计概述

1.1 设计目的

200 字内

如今无人驾驶技术正在如火如荼的研发中，各大公司甚至已经开始上路测试，但是有人驾驶和无人驾驶必定在未来很长时间内共存。为此，现在通用的交通规则不能轻易更改，解决无人驾驶在行驶中遵循现行的交通规则迫在眉睫。由于涉及到人身安全，所以硬件的处理速度是第一条件，FPGA 平台的并行加速对于解决这一问题再适合不过。

1.2 应用领域

200 字内

模式一：

① 无人驾驶汽车识别行驶过程中的动态交通标识，并做出智能反应；

② 汽车导航中识别路途中交通标识并语音提示驾驶员，可作为插件安装在现有的导航中去；

模式二：

③ 机场、火车站等场所行李搬运，免除人员搬运，解放双手，自动跟随；

④ 实现超市手推车的自动跟随，可作为类似共享单车的商业项目，和各大商场达成合作，实现扫码使用。

1.3 主要技术特点

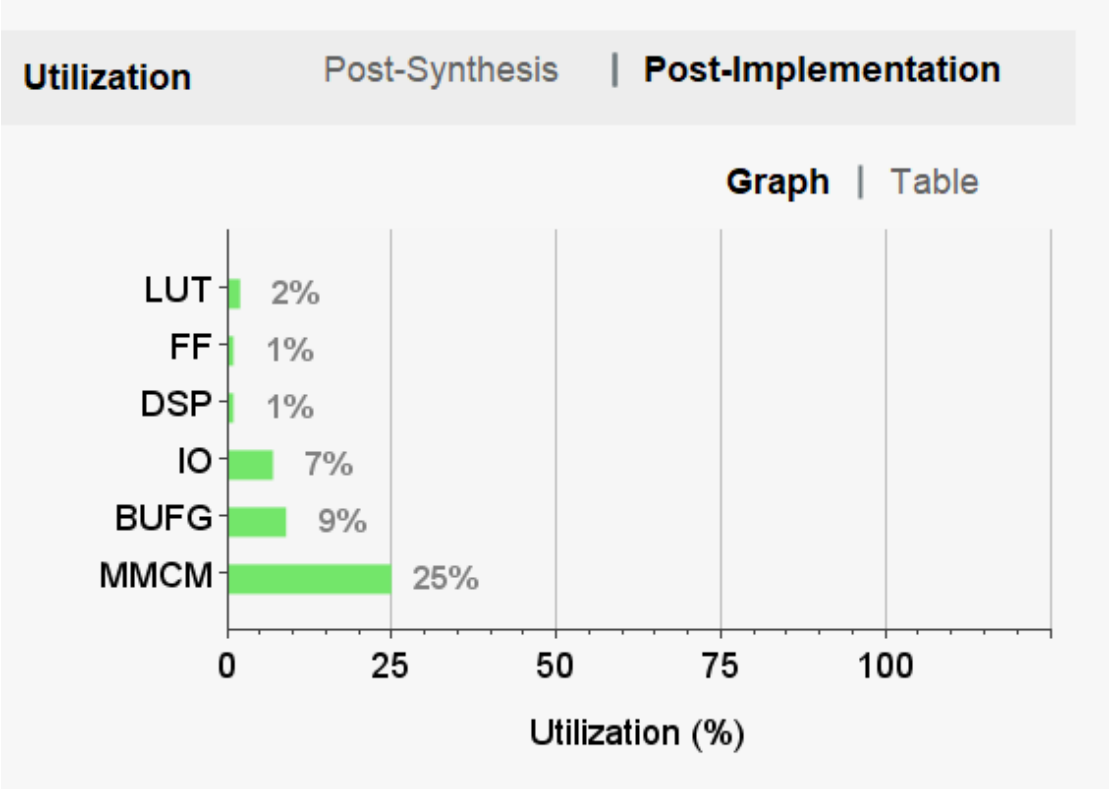
该作品主要分为电机驱动，传感器以及图像识别三大模块。其中电机用四路 PWM 波进行驱动实现车辆的前进，后退与转向。传感器为两个超声波接收器，接收发射源的超声波信号，通过串口将距离数据传输给单片机，单片机进行简单的计算将处理后的方向数据传输给 PYNQ 作为输入信号驱动电机进行相应的操作。图像识别基于 Opencv 算法，通过 python 高层次语言编写图像识别算法以及 HDMI 接口的通信，可识别左右转向标识，限速标识等常见交通警示牌。最终将底层硬件驱动部分封装为 IP 写入并重建 base Overlay，便于在 jupyter 中对电机驱动以及传感器的操作。

1.4 关键性能指标

电机驱动与传感器：最大跟随距离：4.5m

识别交通标识最大可达 40 帧。

FPGA 资源利用情况如图所示：



1.5 主要创新点

（1）结合自动跟随与图像处理，可结合实际路况更智能地完成机场与商场手推车的任务。

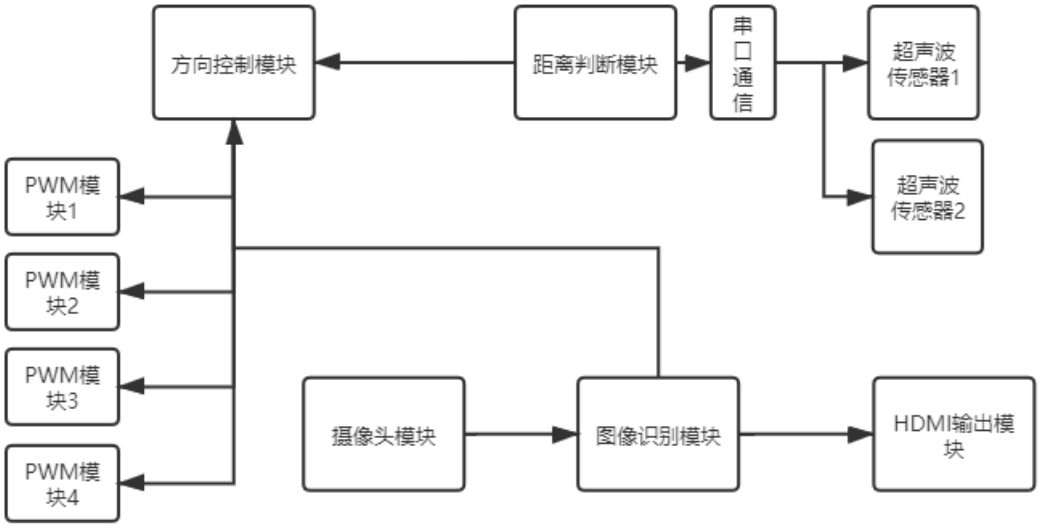
（2）其中的图像识别可应用于汽车，识别路上的标识并通过语音播报进行提醒。

（3）交通标识的训练模型由团队自主训练。

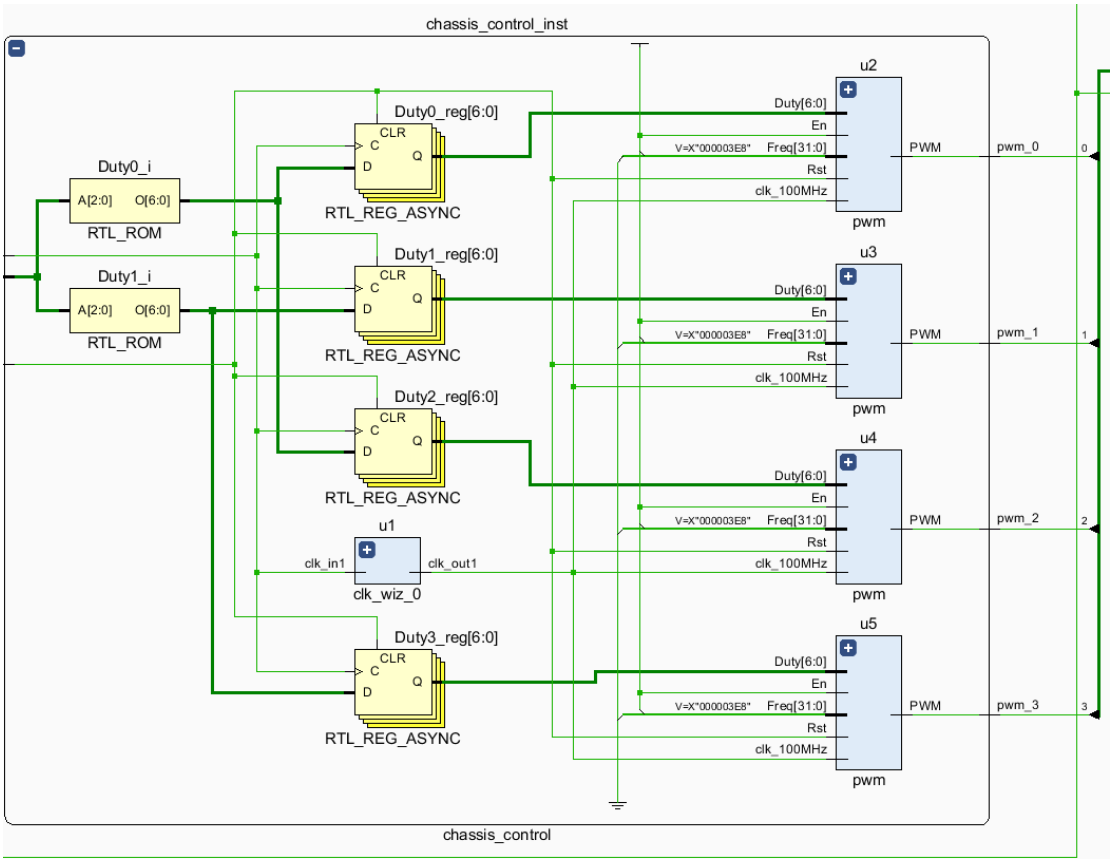
（4）利用底层自主封装的 IP 核与高层次的图像处理算法，充分发挥了 PYNQ 框架的优势。软硬结合，是 FPGA 未来的发展方向。

第二部分　系统组成及功能说明

2.1 整体介绍



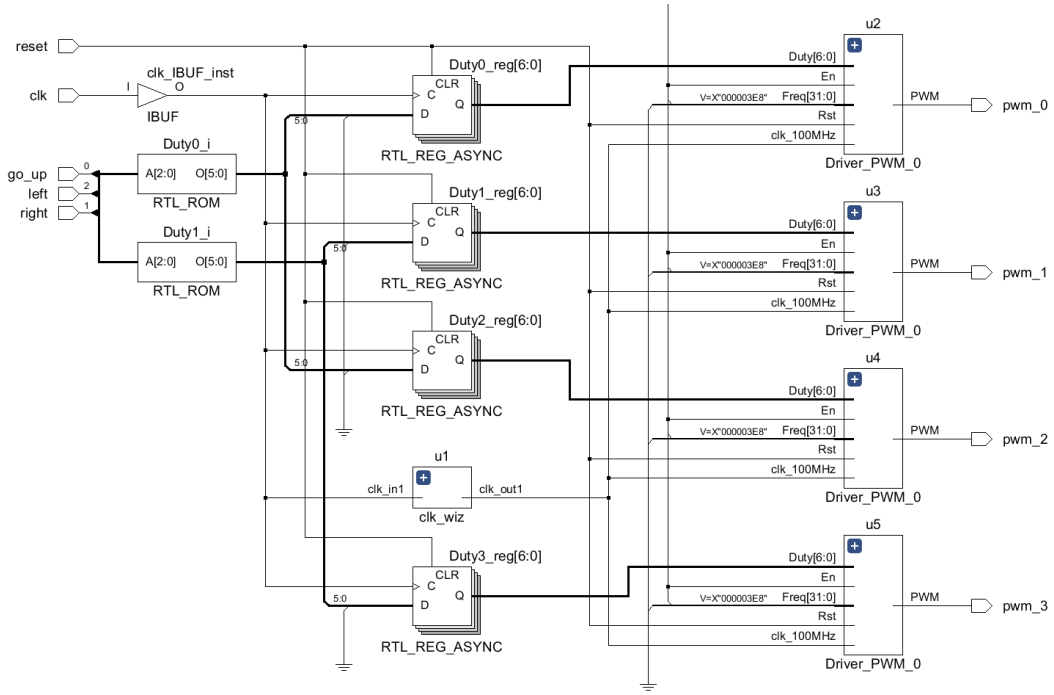
作品主要分为两大部分：电机驱动与图像识别。其中电机驱动的核心模块为方向控制模块。两个超声波传感器接收到信号源发送的信号，得到两组距离数据，两组数据在距离判断模块通过数值比较从而得到转向，前进与停止的方向控制信号，这些信号再控制四路 PWM 从而控制电机转动，实现车辆的运动。



图像识别部分则在 jupyter 平台由 python 语言编写，摄像头模块获得摄像头采集到的数据，利用 Opencv 库中的图像处理算法，调用训练好的交通标识模型，实现对常见交通标识的识别，识别到左转（右转）标识也会对电机驱动模块发送左转（右转）信号，识别到限速标识则会强制改变电机的转速。

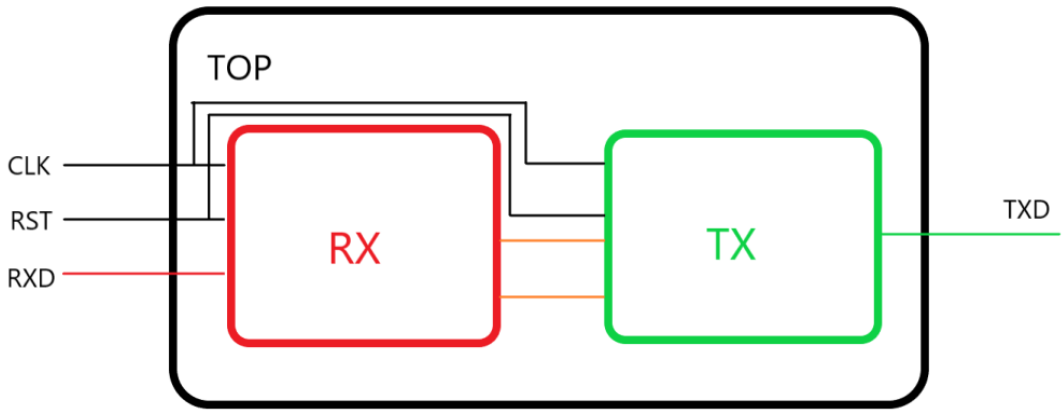
2.2 各模块介绍

四路 PWM 设计：



接收由串口接收到的信号，控制 PWM 的占空比，调节输出高低电平比例，从而实现 PWM 信号的有效输出，进而控制小车的方向。

UART 串口设计：



串口采用的是 RS232 协议，接收到超声波模块发来的 24 位数据后，判断数据帧头是否正确，如果正确解析后面的 16 位有效数据并计算得到距离信息，如果帧头不正确，则继续接收下一个信号。超声波模块发送数据的频

率为 50Hz，波特率为 115200bit/s。

摄像头模块：

在 jupyter 用 python 调用 baseoverlay 中的 video 函数，获取摄像头采集到的图像，进而利用 opencv，对采集到的图像进行识别，如果识别到特定的交通标识，从而返回一个有效值给 PWM 模块，控制小车的移动。

HDMI 模块：

该模块是为了更加直观的看到图像识别模块的识别效果添加的，同样在 jupyter 中调用 HDMI 函数，将 opencv 识别的效果时时传输到显示器显示，并将目标交通标识圈出来。

第三部分　完成情况及性能参数

具体完成的情况，可图文结合，具体的性能参数等量化指标

整个项目完成度百分之九十，图像识别的帧数可以通过优化代码进一步提高，目前帧数最高可达 40FPS，超声波传感器最大传输距离可达 4.5m，调试过程中 5m 之内小车运行稳定。

第四部分　总结

4.1 可扩展之处

基本功能已经初步实现，但仍然有很多可以扩展的地方。

由于时间的关系，团队只训练了两个交通标识（左转右转），采用的是欧美地区的交通标识，后面可以根据国内的标识多训练几个模型。识别的帧数方面也可以通过优化代码进一步提高。

自动跟随模式下，可以通过更换超声波传感器，采用雷达毫米波增大传输距离和灵敏度。

4.2 心得体会

做比赛的这段时间里，学会了很多诸如 UART 的协议并用 FPGA 完美复现出来，另外了解了很多 Linux 开发的知识，第一次用 python 开发了硬件。收获很大。

最大的收获还是心态方面，面对诸多的 bugs，熬了很多个晚上，几近猝死，最终解决问题的心情是无以言表的。

第五部分　参考文献

[1]Eric Matthes，Python 编程从入门到实践[M].人民邮电出版社,2016-2018;
[2]张平，OpenCV 算法精解基于 Python 与 C++[M].电子工业出版社,2017/10;
[3]何宾，Xilinx FPGA 权威设计指南[M].电子工业出版社。

第六部分　附录

重要代码、推导过程等不便于在正文中体现的内容

模式一（图像识别）：

```
#   Load the Overlay
from pynq.overlays.base import BaseOverlay
from pynq.lib.video import *

base = BaseOverlay("base.bit")
#hdmi_in = base.video.hdmi_in
#hdmi_out = base.video.hdmi_out
```

```
#   monitor configuration: 640*480
Mode = VideoMode(640,480,24)
hdmi_out = base.video.hdmi_out
```

```
#   Initialize HDMI I/O
hdmi_out.configure(Mode,PIXEL_BGR)
hdmi_out.start()
frame_in_w = 640
frame_in_h = 480
```

```
#   Apply the face detection to the input
import cv2
import numpy as np
import time
```

```
#path = '\\home\\xilinx\\jupyter_notebooks\\base\\video\\data\\casecade,xml'
#frame = hdmi_in.readframe()
faceCascade = cv2.CascadeClassifier(
    '/home/xilinx/jupyter_notebooks/base/video/data/'
    'cascade.xml')
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, frame_in_w);
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, frame_in_h);
```

```
timeF = 10
```

```
globalimage=np.zeros((640,480,3), np.uint8)
```

```
while True:
    print("进入循环")
    ret, frame = cap.read()
    globalimage = frame
    img = frame.copy()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

```
    print("准备检测")
    rect = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.15,
        minNeighbors=3,
        minSize=(3, 3),
        flags=cv2.IMREAD_GRAYSCALE
    )
    print("检测完毕")
```

```
#    print(rect)
    if rect == ():
        print("没检测到")
    else:
        print("检测到了")
```

```
    #time.sleep(10)
    for (x, y, w, h) in rect:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)
#    cv2.imshow('frame', frame)
#   Show results on HDMI output
    outframe = hdmi_out.newframe()
    outframe[:] = frame
    hdmi_out.writeframe(outframe)

    print("已显示")
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

```
#   Release HDMI
hdmi_out.stop()
del hdmi_out
```

模式二（自动跟随）：

PWM 模块：

Top:

```
module pwm(
    input       clk,
    input       reset,
```

```
    input       left,
    input       right,
    input       go_up,

    output wire   pwm_0,
    output wire   pwm_1,
    output wire  pwm_2,
    output wire  pwm_3

    );

parameter grade0 = 0;
parameter grade1 = 20;
parameter grade2 = 40;
parameter grade3 = 60;
parameter grade4 = 80;
parameter grade5 = 100;




wire  [2:0] direction;

assign direction = {left,right,go_up}; //AR0   AR1   AR2



wire clk_out1;
reg [31:0] Freq  =  1000;

reg [31:0] cnt = 0;
reg [4:0] num = 0;

//duty < 100

reg [6:0] Duty0 = grade1;
reg [6:0] Duty1 = grade1;
reg [6:0] Duty2 = grade1;
reg [6:0] Duty3 = grade1;



always @(posedge clk or negedge reset) begin
    if (reset) begin
```

```
        Duty0 <= grade0;
        Duty1 <= grade0;
        Duty2 <= grade0;
        Duty3 <= grade0;
    end
    else begin
        case (direction)
            3'b000: begin Duty0 <= grade0; Duty1 <= grade0; Duty2
 <= grade0; Duty3 <= grade0; end
            3'b111: begin Duty0 <= grade2; Duty1 <= grade2; Duty2
 <= grade2; Duty3 <= grade2; end    //go
            3'b101: begin Duty0 <= grade3; Duty1 <= grade1; Duty2
 <= grade3; Duty3 <= grade1; end   //right
            3'b010: begin Duty0 <= grade1; Duty1 <= grade3; Duty2
 <= grade1; Duty3 <= grade3; end   //left
            default: begin Duty0 <= grade0; Duty1 <= grade0; Duty
2 <= grade0; Duty3 <= grade0; end
        endcase
    end
end


clk_wiz u1
(
    .clk_in1    (clk),
    .clk_out1   (clk_out1)

);

Driver_PWM_0 u2
(
    .clk_100MHz     (clk_out1),
    .Rst            (reset),
    .En             (1),
    .Freq           (Freq),
    .Duty           (Duty0),
    .PWM            (pwm_0)
);


Driver_PWM_0 u3
(
    .clk_100MHz     (clk_out1),
    .Rst            (reset),
    .En             (1),
```

```
    .Freq           (Freq),
    .Duty           (Duty1),
    .PWM            (pwm_1)
);

Driver_PWM_0 u4
(
    .clk_100MHz     (clk_out1),
    .Rst            (reset),
    .En             (1),
    .Freq           (Freq),
    .Duty           (Duty2),
    .PWM            (pwm_2)
);

Driver_PWM_0 u5
(
    .clk_100MHz     (clk_out1),
    .Rst            (reset),
    .En             (1),
    .Freq           (Freq),
    .Duty           (Duty3),
    .PWM            (pwm_3)
);


endmodule
```

Driver_PWM_0:

```
module Driver_PWM_0(
    input clk_100MHz,
    input [31:0]Freq,
    input [6:0]Duty,
    input Rst,
    input En,
    output reg PWM = 1
    );
    reg [31:0]Period=0;      //PWM period
    reg [31:0]Period_Cnt=0; //PWM cycle count
    reg [31:0]Duty_Num=0;   //Number of duty cycles

    //Calculated cycle
    always @ (*)
```

```
        begin
            Period=100000000/Freq;
            if(Duty<100)
                Duty_Num=Duty*Period/100;
            else
                Duty_Num=Period;
        end

    //Cycle frequency division
    always@(posedge clk_100MHz or negedge Rst)
        begin
            if(Rst)
                Period_Cnt<=0;
            else
                begin
                    if(Period_Cnt<Period-1)
                        Period_Cnt<=Period_Cnt+1;
                    else
                        Period_Cnt<=0;
                end
        end

    //Generate PWM
    always @ (posedge clk_100MHz or negedge Rst)
        begin
            if(Rst)
                PWM<=0;
            else
                begin
                    //If enabled, duty cycle adjustment
                    if(En)
                        begin
                            if(Period_Cnt<Duty_Num)
                                PWM<=1;
                            else
                                PWM<=0;
                        end
                    else
                        PWM<=0;
                end
        end
endmodule
```

UART(RS232) + distance 判断 模块：

Top:

```
module UART_top(
    input sys_clk,
    input sys_rst,
    input rx1,
    input rx0,



    output          left,
    output          right,
    output          go,

    output          tx,
    output          tx1
    );

    wire [23:0]   po_data1;
    wire [23:0]   pi_data1;
    wire [23:0]   po_data0;
    wire [23:0]   pi_data0;
    wire          po_flag0;
    wire          pi_flag0;
    wire          po_flag1;
    wire          pi_flag1;
    wire [15:0]   distance_0;
    wire [15:0]   distance_1;


    UART_rx0 UART_rx0
    (
        .sys_clk        (sys_clk)   ,
        .sys_rst        (sys_rst)   ,
        .rx             (rx0)        ,
        .po_data0       (po_data0)   ,
        .po_flag0       (po_flag0)
    );

    UART_rx1 UART_rx1
    (
        .sys_clk        (sys_clk)   ,
        .sys_rst        (sys_rst)   ,
```

```
        .rx             (rx1)        ,
        .po_data1       (po_data1)   ,
        .po_flag1       (po_flag1)
    );

    UART_tx UART_tx
    (
        .sys_clk        (sys_clk)   ,
        .sys_rst        (sys_rst)   ,
        .tx             (tx)        ,
        .pi_data0       (po_data0)   ,
        .pi_flag0        (po_flag0)
    );

    UART_tx1 UART_tx1
    (
        .sys_clk        (sys_clk)   ,
        .sys_rst        (sys_rst)   ,
        .tx1             (tx1)        ,
        .pi_data1        (po_data1)   ,
        .pi_flag1        (po_flag1)
    );

    distance0 distance0
    (
        .sys_clk        (sys_clk)   ,
        .sys_rst        (sys_rst)   ,
        .distance       (distance_0)  ,
        .po_data0       (po_data0)   ,
        .po_flag0       (po_flag0)
    );

    distance1 distance1
    (
        .sys_clk        (sys_clk)   ,
        .sys_rst        (sys_rst)   ,
        .distance       (distance_1)  ,
        .po_data1       (po_data1)   ,
        .po_flag1       (po_flag1)
    );

    vs2pwmsign vs2pwmsign
    (
        .sys_clk        (sys_clk)   ,
```

```
        .sys_rst        (sys_rst)   ,
        .distance_0     (distance_0) ,
        .distance_1     (distance_1) ,
        .left           (left)      ,
        .right          (right)     ,
        .go             (go)
    );

endmodule
```

UART_rx0:

```
module UART_rx0
#(
    parameter uart_bps = 'd115200,
    parameter clk = 'd125000000
)

(
    input   wire            sys_clk,
    input   wire            sys_rst,
    input   wire            rx,

    output  reg [23:0]      po_data0,
    output  reg             po_flag0
);

parameter baud_cnt_max = clk / uart_bps;

reg                 rx_reg1;
reg                 rx_reg2;
reg                 rx_reg3;
reg     [23:0]      rx_data;
reg                 rx_flag;
reg                 start_flag;
reg                 work_en;
reg     [15:0]      baud_cnt;
reg                 bit_flag;
reg     [4:0]       bit_cnt;



always@(posedge sys_clk or negedge sys_rst)
```

```
begin
    if(!sys_rst)
    begin
        rx_reg1 <= 1;
    end
    else
        rx_reg1 <= rx;
end



always@(posedge sys_clk or negedge sys_rst)
begin
    if(!sys_rst)
    begin
        rx_reg2 <= 1;
    end
    else
        rx_reg2 <= rx_reg1;
end

always@(posedge sys_clk or negedge sys_rst)
begin
    if(!sys_rst)
    begin
        rx_reg3 <= 1;
    end
    else
        rx_reg3 <= rx_reg2;
end

always@(posedge sys_clk or negedge sys_rst)
begin
    if(!sys_rst)
        start_flag <= 0;
    else if((rx_reg3 == 1)&& (rx_reg2 == 0) && (work_en == 0))
        start_flag <= 1;
    else
        start_flag <= 0;
end

always@(posedge sys_clk or negedge sys_rst)
    if(!sys_rst)
        work_en <= 0;
```

```
    else if(start_flag == 1)
        work_en <= 1;
    else if(bit_cnt == 24 && bit_flag == 1)
        work_en <= 0;
    else
        work_en <= work_en;

always@(posedge sys_clk or negedge sys_rst)
    if(!sys_rst)
        baud_cnt <= 0;
    else if(baud_cnt == baud_cnt_max - 1 || work_en == 0)
        baud_cnt <= 0;
    else
        baud_cnt <= baud_cnt + 1;

always@(posedge sys_clk or negedge sys_rst)
    if(!sys_rst)
        bit_flag <= 0;
    else if(baud_cnt == baud_cnt_max / 2 - 1)
        bit_flag <= 1;
    else
        bit_flag <= 0;

always@(posedge sys_clk or negedge sys_rst)
    if(!sys_rst)
        bit_cnt <= 0;
    else if(bit_cnt == 24 && bit_flag == 1)
        bit_cnt <= 0;
    else if(bit_flag == 1)
        bit_cnt <= bit_cnt + 1;

always@(posedge sys_clk or negedge sys_rst)
    if(!sys_rst)
        rx_data <= 8'b0;
    else if(bit_cnt >= 1 && bit_cnt <= 24 && bit_flag == 1)
        rx_data <= {rx_reg3,rx_data[7:1]};

always@(posedge sys_clk or negedge sys_rst)
    if(!sys_rst)
        rx_flag <= 0;
    else if(bit_cnt == 24 && bit_flag == 1)
        rx_flag <= 1;
    else
        rx_flag <= 0;
```

```
always@(posedge sys_clk or negedge sys_rst)
    if(!sys_rst)
        po_data0 <= 0;
    else if(rx_flag == 1)
        po_data0 <= rx_data;

always@(posedge sys_clk or negedge sys_rst)
    if(!sys_rst)
        po_flag0 <= 0;
    else
        po_flag0 <= rx_flag;


endmodule
```

UART_tx:

```
module UART_tx
#(
    parameter uart_bps = 'd115200,
    parameter clk = 'd125000000
)
(
    input wire      sys_clk,
    input wire      sys_rst,
    input wire [23:0]pi_data0,
    input wire      pi_flag0,

    output reg         tx
    );

parameter baud_cnt_max = clk / uart_bps;


    reg         work_en;
    reg [15:0]  baud_cnt;
    reg [5:0]   bit_cnt;
    reg         bit_flag;

    always @(posedge sys_clk or negedge sys_rst) begin
        if (!sys_rst) begin
            work_en <= 0;
        end
```

```
        else if(pi_flag0) begin
            work_en <= 1;
        end
        else if(bit_cnt == 25 && bit_flag == 1) begin
            work_en <= 0;
        end
    end

    always @(posedge sys_clk or negedge sys_rst) begin
        if (!sys_rst)
            baud_cnt <= 0;
        else if (work_en == 0 || baud_cnt == baud_cnt_max) begin
            baud_cnt <= 0;
        end
        else if (work_en == 1) begin
            baud_cnt <= baud_cnt + 1;
        end
    end

    always @(posedge sys_clk or negedge sys_rst)
        if (!sys_rst)
            bit_flag <= 0;
        else if (baud_cnt == 1) begin
            bit_flag <=1;
        end
        else begin
            bit_flag <= 0;
        end

    always @(posedge sys_clk or negedge sys_rst)
        if (!sys_rst)
            bit_cnt <= 0;
        else if (bit_cnt == 25 && bit_flag == 1) begin
            bit_cnt <= 0;
        end
        else if (work_en == 1 && bit_flag == 1) begin
            bit_cnt <= bit_cnt + 1;
        end

    always @(posedge sys_clk or negedge sys_rst)
        if (!sys_rst)
            tx <= 1;
        else if (bit_flag == 1) begin
            case (bit_cnt)
```

```
                0: tx <= 1'b0;
                1: tx <= pi_data0[0];
                2: tx <= pi_data0[1];
                3: tx <= pi_data0[2];
                4: tx <= pi_data0[3];
                5: tx <= pi_data0[4];
                6: tx <= pi_data0[5];
                7: tx <= pi_data0[6];
                8: tx <= pi_data0[7];
                9: tx <= 1'b1;
                default: tx <= 1'b1;
            endcase
        end

endmodule
```

distance:

```
module distance0(
    input       sys_clk,
    input       sys_rst,
    input [23:0] po_data0,
    input       po_flag0,

    output reg [15:0] distance
);

    always @(posedge sys_clk or negedge sys_rst) begin
        if (!sys_rst) begin
            distance <= 0;
        end
        else if (!po_flag0) begin
            distance <= 0;
        end
        else begin
            distance <= po_data0[1] << 8 | po_data0[2];
        end
    end

endmodule
```

vs2pwmsign

```
module vs2pwmsign(
    input           sys_clk,
```

```
    input           sys_rst,
    input [15:0]    distance_0,
    input [15:0]    distance_1,

    output reg      left,
    output reg      right,
    output reg      go
    );

    parameter distance_min = 500;

    wire  stop_flag;

    assign stop_flag = (distance_0 == distance_1 && distance_0 ==
distance_min) ? 1 : 0;

    always @(posedge sys_clk or negedge sys_rst) begin
        if (!sys_rst) begin
            left <= 0;
        end
        else begin
            if(distance_0 < distance_1) begin
                left <= 1;
            end
            else begin
                left <= 0;
            end
        end
    end

    always @(posedge sys_clk or negedge sys_rst) begin
        if (!sys_rst) begin
            right <= 0;
        end
        else begin
            if(distance_0 > distance_1) begin
                right <= 1;
            end
            else begin
                right <= 0;
            end
        end
    end
```

```
endmodule
```