2019 年 SEU-Xilinx 国际暑期学校团队项目设计文档

(Project Paper Submission Template)

作品名称	自动路标识别巡线平衡车
组员姓名	王文杰 许巾一 何群芳
房间号及桌号	717-14 桌

第一部分

小组成员分工

（各成员任务分配）

1　小组分工

	姓名	任务分配
组长	王文杰	（软件+SDK）串口通信调试/CNN 设计及训练/SDK 调试/运动参数调试
组员 1	许巾一	（Vivado）整体系统搭建及调试/运动控制 PID 的硬件设计实现
组员 2	何群芳	（Vivado_HLS）图像处理算法设计、IP 生成及优化/CNN 的 IP 实现
组员 3		

2　时间计划完成表

编号	类别	内容	状态	备注	人员
1	运动控制	平衡车STM32例程实测	成功	可保持直立	王文杰
2	串口通信	平衡车STM32串口通信协议解析	成功	可从PC端发送控制信号控制运动方向	王文杰
3	图像处理	SEA-S7摄像头图像处理DEMO实测	成功	摄像头采集图像可处理后在HDMI端显示，LUT>4000	王文杰
4	图像处理	寻线算法方案讨论		暂定两套：黑线取中值/目标追踪算法求方向	许巾一，何群芳
5	运动控制	PID算法编写、验证、生成IP	成功		许巾一
6	图像处理	编写图像处理算法（目标追踪IP）生成IP	暂弃	资源消耗过度，LUT>50000	何群芳
7	串口通信	SEA-S7设置串口发送端	成功	可在PC端接收到信号	王文杰
8	图像处理	编写图像处理算法（中值提取IP）生成IP	成功	AXI-Stream输入输出格式，LUT>2000	何群芳
9	图像处理	SEA-S7 视频流DEMO删减修改	成功	去除图像处理部分，LUT<3000	许巾一
10	串口通信	实现SEA-S7到STM32的串口通信，运动控制	成功	修改比特率，修改地址分配（Local MEMORY），保证串口缓存	王文杰
11	图像处理	图像处理IP在系统中的接口对接讨论		暂定两套：修改接口为AXI-Stream/修改HLS_ip接口	许巾一，何群芳
12	图像处理	图像处理IP端口传输格式调整，由stream转为pixel	失败	HLS无法仿真，verilog仿真无输出，实际逻辑有问题。	何群芳
13	图像处理	生成CNN训练集，用python代码训练提高准确性	成功	准确率≈90%	王文杰
14	图像处理	CNN的HLS实现	暂弃	预算资源超支，放弃该方法，尝试模板匹配	何群芳
15	系统调试	尝试添加IP在Bayer2RGB模块后，串口查看中线返回值	失败	串口输出一直为0	王文杰，许巾一
16	系统调试	尝试添加在摄像头后stream处添加图像处理寻线IP	失败	接口和时钟报错，无法综合	许巾一
17	系统调试	在Bayer2RGB模块后添加Stream转换IP，ila-Debug调试	成功	Debug输出正确中线值	王文杰，许巾一
18	运动控制	修改PID.v以适应图像处理模块输出，改输出为wire型	成功	输入由error转为LineLocation（error=LineLocation-MidLine）	许巾一
19	系统调试	在系统巡线模块后添加PID的IP，Debug测试	失败	Debug输入有正确中线值，输出一直为0	王文杰，许巾一
20	系统调试	直接将PID.v导入工程，生成RTL IP，Debug测试	成功	Debug输入有正确中线值，输出有信号	许巾一
21	系统调试	添加自定义lite IP，将PID输出信号经软件串口传出	成功	可观测，中线偏左负值，偏右正值	许巾一
22	图像处理	设计实现路标提取，将提取图像大小缩放为32*32	暂弃	放弃CNN路标识别，资源消耗过高	
23	图像处理	完成路标匹配IP设计优化，在HLS中封装	成功	LUT≈2000	何群芳
24	系统调试	将路标匹配IP加入系统，Debug调试	失败	BRAM使用超标，需进行系统优化，计划将instruction改为8K	许巾一
25	系统调试	将路标匹配IP加入系统，instruction改为8K	失败	SDK报错，程序存储空间不足	王文杰，许巾一
26	系统调试	将开发板和摄像头设计固定在平衡车上，布置测试场地	成功	摄像头位置影响图像处理结果，需调试；场地布置白底黑线	王文杰，许巾一
27	系统调试	将路标匹配IP加入系统，删除ila IP,instruction改为16K	成功	LUT>6000	许巾一
28	系统调试	将寻线算法和PID结合入系统，上车调试，实现寻线功能	成功	Delay(9800)	王文杰，许巾一
29	系统调试	将CNN路标识别加入系统，上车调试	暂弃	放弃CNN路标识别	
30	系统调试	将路标识别加入寻线系统，上车调试	ing	由于环境影响，识别结果不准确，继续优化算法	王文杰，何群芳

3　各成员部分任务工作展示

王文杰：

名称	修改日期	类型	大小
detectRoadSign	2019/7/18 10:11	文件夹	
line_single(1)	2019/7/16 16:31	文件夹	
line_stream	2019/7/16 16:30	文件夹	
Pix_count	2019/7/16 11:53	文件夹	
xilinx_com_hls_hls_line_1_0	2019/7/15 10:28	文件夹	
line_single(1).rar	2019/7/16 16:20	好压 RAR 压缩文件	2 KB
line_stream.rar	2019/7/16 16:20	好压 RAR 压缩文件	46 KB
vivado_hls.log	2019/7/18 10:32	Text Document	120 KB
xilinx_com_hls_hls_line_1_0.zip	2019/7/15 10:20	好压 ZIP 压缩文件	29 KB

许巾一：

2019年SEU-Xilinx国际暑期学校项目内容	2019/7/17 星期...	文件夹	
Demo	2019/7/13 星期...	文件夹	
ImgPro	2019/7/16 星期...	文件夹	
matlab	2019/7/15 星期...	文件夹	
PID_Error	2019/7/16 星期...	文件夹	
PID_LineLocation	2019/7/17 星期...	文件夹	
spartan_cam	2019/7/16 星期...	文件夹	
spartan_cam_NoImgPro	2019/7/16 星期...	文件夹	
spartan_cam_UART_ImgPro	2019/7/16 星期...	文件夹	
spartan_cam_UART_ImgPro_PID	2019/7/17 星期...	文件夹	
spartan_cam_UARTCar	2019/7/16 星期...	文件夹	
spatran_cam_UART_NoImgPro	2019/7/15 星期...	文件夹	
test	2019/7/17 星期...	文件夹	
pg260-mipi-csi2-tx.pdf	2019/7/14 星期...	Foxit Phantom P...	1,906 KB
STM32平衡车源码.rar	2019/7/11 星期...	360压缩 RAR 文件	5,580 KB
工作进度.xlsx	2019/7/17 星期...	Microsoft Excel ...	13 KB

AxisIP	2019/7/16 星期...	文件夹	
ip_repo	2019/7/16 星期...	文件夹	
MiddleSearch	2019/7/16 星期...	文件夹	
MidIIne	2019/7/15 星期...	文件夹	
spartan_cam_2	2019/7/17 星期...	文件夹	
Stream_RGB	2019/7/16 星期...	文件夹	
xilinx_com_hls_detect_1_0	2019/7/17 星期...	文件夹	
xilinx_com_hls_detect_2_0	2019/7/17 星期...	文件夹	
xilinx_com_hls_hls_line_1_0	2019/7/16 星期...	文件夹	
xilinx_com_hls_hls_line_2_0	2019/7/16 星期...	文件夹	
xilinx_com_hls_hls_line_3_0	2019/7/16 星期...	文件夹	
xilinx_com_hls_lite_1_0	2019/7/16 星期...	文件夹	
main.c	2019/7/13 星期...	C Source File	4 KB
platform.c	2019/7/13 星期...	C Source File	4 KB
platform.h	2019/7/13 星期...	C Header File	2 KB
platform_config.h	2019/7/13 星期...	C Header File	1 KB
xilinx.com_user_PID_1.0.zip	2019/7/17 星期...	360压缩 ZIP 文件	5 KB
xilinx_com_hls_detect_1_0(2).zip	2019/7/17 星期...	360压缩 ZIP 文件	64 KB
xilinx_com_hls_detect_1_0.zip	2019/7/17 星期...	360压缩 ZIP 文件	66 KB

何群芳：

Axilite	2019/7/15 15:46	文件夹	
detectRoadSign	2019/7/17 14:56	文件夹	
line_gray	2019/7/16 10:33	文件夹	
line_single	2019/7/16 16:33	文件夹	
line_stream	2019/7/16 16:20	文件夹	
matlab	2019/7/16 10:13	文件夹	
network	2019/7/15 15:08	文件夹	
spartan_cam	2019/7/14 17:31	文件夹	
vivado_hls.log	2019/7/17 16:02	文本文档	2 KB
截图.docx	2019/7/17 15:20	Microsoft Word 文档	361 KB

第二部分

设计概述 /Design Introduction

（请简要描述一下你的设计：1. 功能说明；2. 所用到的设备清单）

1 设计目的

设计目的

随着科技的发展，自动驾驶技术逐渐成为前沿科技中一个炙手可热的研究场景。自动驾驶汽车集成了物联网，人工智能，运动控制等多个研究领域于一体，旨在实现汽车以机器学习为基础的自动路况认知，解决方案推理和运动控制。其中，车辆需要自动识别车道、信号灯、路标等信息实现信息采集；整合采集的信息进行路线规划和速度方向控制；根据反馈信号实现车辆运动的平稳控制。在实现这些功能时，往往需要运用到复杂庞大的算法和数据，这会造成系统运行速度慢，响应延迟等致命的问题。

一方面，由于车载设备的体积限制，性能高成本高的计算机难以直接安装进汽车。另一方面，若远程控制，数据的通信会消耗大量时间，无法实现系统实时响应，且稳定性堪忧。因此，车载自动驾驶设备的实现需依托于边缘计算，物联网和嵌入式技术的支持。同时，为了提高系统的处理和响应时间，对算法进行 FPGA 加速是目前比较主流的一种自动驾驶解决方案。

考虑到 FPGA 相对较高的成本和数据处理特性，将全部系统在 FPGA 中实现在实际场景中并非最佳的解决方案。因此，基于 FPGA 并行处理的特性，对系统进行软硬件协同设计：将部分功能模块移植到 FPGA 中进行实现，其余模块使用软件实现。实现基于软硬件协同设计的自动驾驶系统，是自动驾驶技术实现实际场景应用的关键。

本项目旨在运用 FPGA 硬件加速技术，研究如何在资源有限的情况下，对具有自动路标识别和巡线功能的简单自动驾驶系统进行软硬件协同设计，并在嵌入式系统中进行实现。本项目将自动驾驶中的图像处理算法和部分运动控制算法划分到硬件中，实现实时控制。同时，由于该自动驾驶系统需整合 Spartan7，ESP32 和 STM32，，该项目也会对多处理器架构在自动驾驶领域的应用前景进行探索和研究。

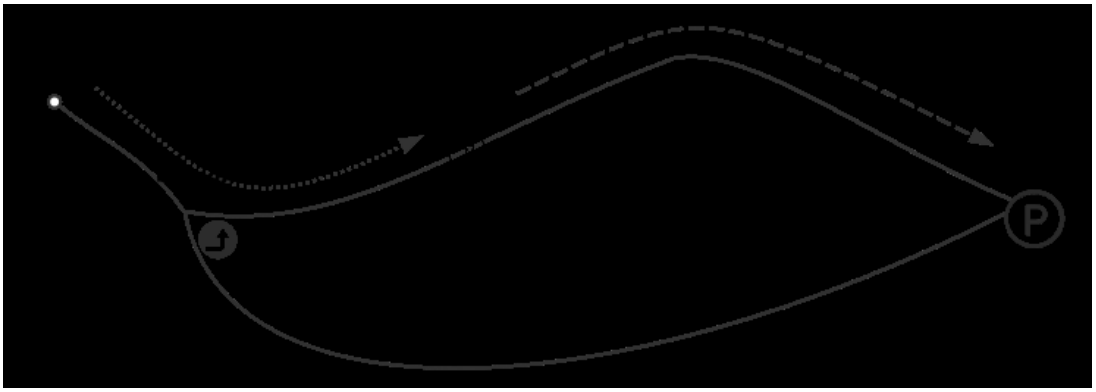
2 功能说明

通过 MIPI 摄像头获取视频流后，在 SEA-S7 开发板中进行图像处理，实现寻线和路标识别。根据图像处理信号，在 FPGA 中编写方向控制 PID，经由 SEA-S7 的串口与平衡车 STM32 通信，进行平衡车运动控制，最后实现平衡车巡线运动，遇到路标时能够识别并作出对应的反应。

(1) 实现图像的传输和处理，得到正确的路线规划和路标识别结果；

(2) 实现 PID 控制，根据路线偏移输出正确的控制信息。

(3) 结合上述功能，实现整体系统，使平衡车实现寻迹和路标识别功能。最终平衡车运行效果如下图所示：平衡车会通过车载 MIPI 摄像头实时识别黑线，并在黑线上前进，当遇到地面路标时（如图所示分叉路口）平衡车会根据路标信息进行转向/停车的运动。



3 设备清单

编号	设备名	数量	备注
1	SEA-S7 开发板	1	系统实现用开发板
2	SD 卡	1	下载存储 bitStream
3	平衡车套件	1	系统实现用车
4	MIPI 摄像头	1	采集图像
5	HDMI 显示屏	1	图像处理调试使用
6	HDMI 转 mini HDMI 转换线	1	图像处理调试使用

第三部分

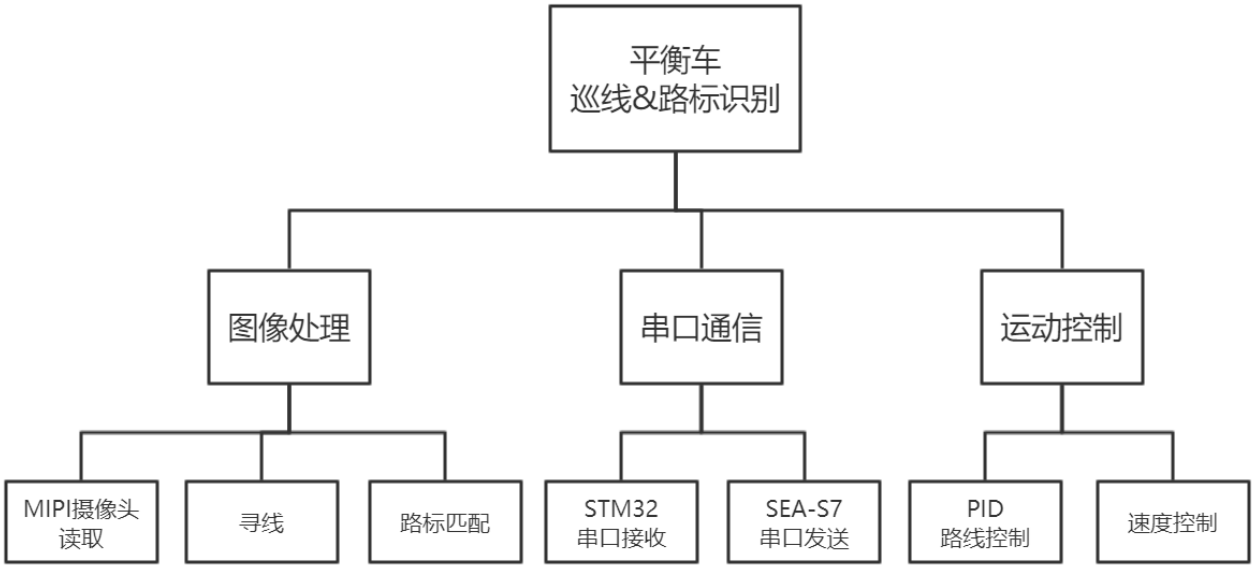
详细设计 /Detailed Design

（请详细说明你作品要实现的所有功能以及如何组建系统以实现该功能，还包括为实现该功能需要用到的所有参数和所有操作的详细说明，必要的地方多用图表形式表述）

如因文档篇幅限制无法详述，可提供附件。

1 系统功能

本平衡车系统功能划分如下图所示，可以看出本系统主要有三个功能：能够识别黑线和路标的图像处理功能；能够根据图像处理返回值控制平衡车运动方向的运动控制功能；能够将 PID 控制信号通过串口传递给平衡车 STM32 的串口通信功能。

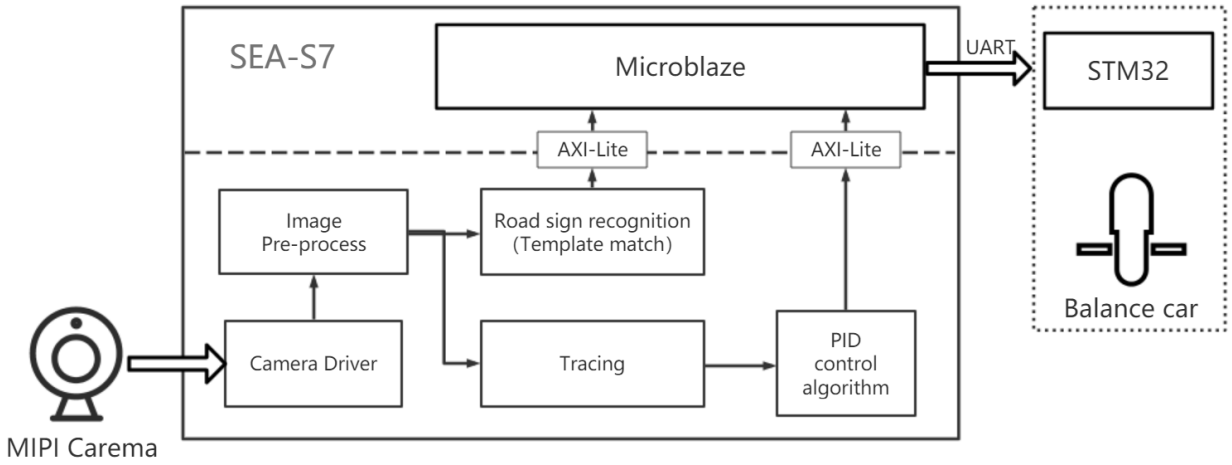


其中，图像处理功能又可分为三部分进行：首先由连接在 SEA-S7 开发板的 MIPI 摄像头进行图像采集，然后对得到的图像分别进行寻线处理和路标匹配处理。

运动控制主要分为两个部分，PID 路线控制根据寻线图像处理得到的黑线中线值，得到平衡车的偏移量，从而调节其运动方向；速度控制主要根据路标识别反馈的信息，由平衡车 STM32 开发板进行控制。

串口通信主要分为两个部分，首先要解析平衡车运动控制板 STM32 中的串口通信协议，其次调试 SEA-S7 的串口作为信号发送端，实现板到板的通信。

2 系统结构



系统结构主要如上图所示，图像处理和运动方向的 PID 控制主要在 SEA-S7 开发板上实现，通过串口通信发送相应的控制命令给 STM32，平衡车具体的电机控制和平衡控制由 STM32 中的三环 PID 实现。

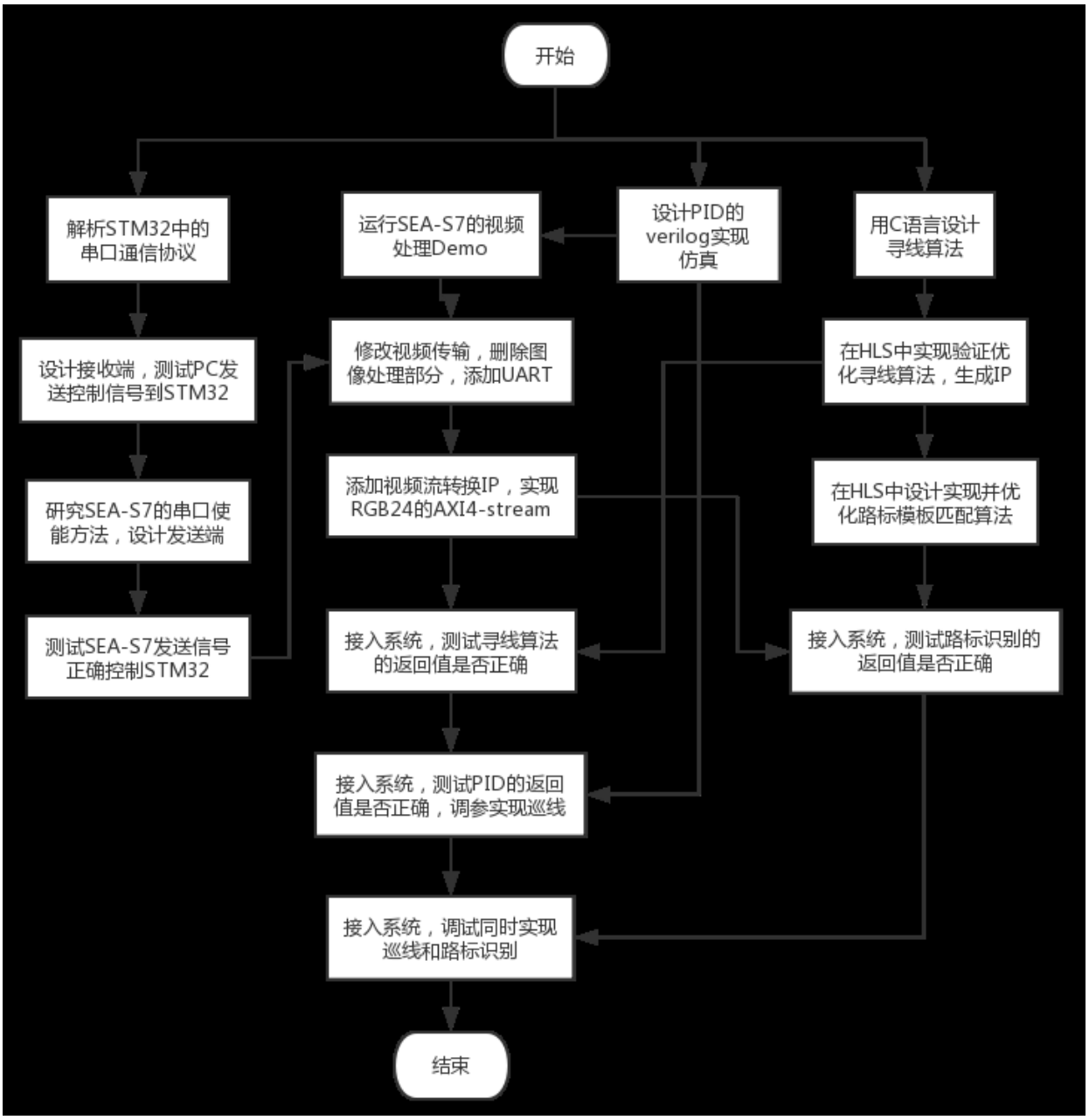
首先，摄像头捕获图像数据，经过一系列 IP 的数据格式转换，生成具有场同步和行同步信号的视频数据流。然后，由该视频数据流再分别视频流数据分别供寻线 IP 和路标匹配 IP 进行图像处理。

其中，寻线处理 IP 会返回图象中的黑色中线值，PID IP 利用这个中线值进行 PID 计算，得到控制信号输出；路标识别 IP 会对图像中固定范围内是否有路标和路标是什么进行判断推理，得到相应的控制指令作为输出。

最后，两股控制信号会通过 AXI4-Lite 传输到搭建好的 MicroBlaze 软核中，在软核里，这些控制信号以一个特定的周期通过串口通信传输至 STM32 中去，从而实现平衡车的运动控制。

3 实现流程

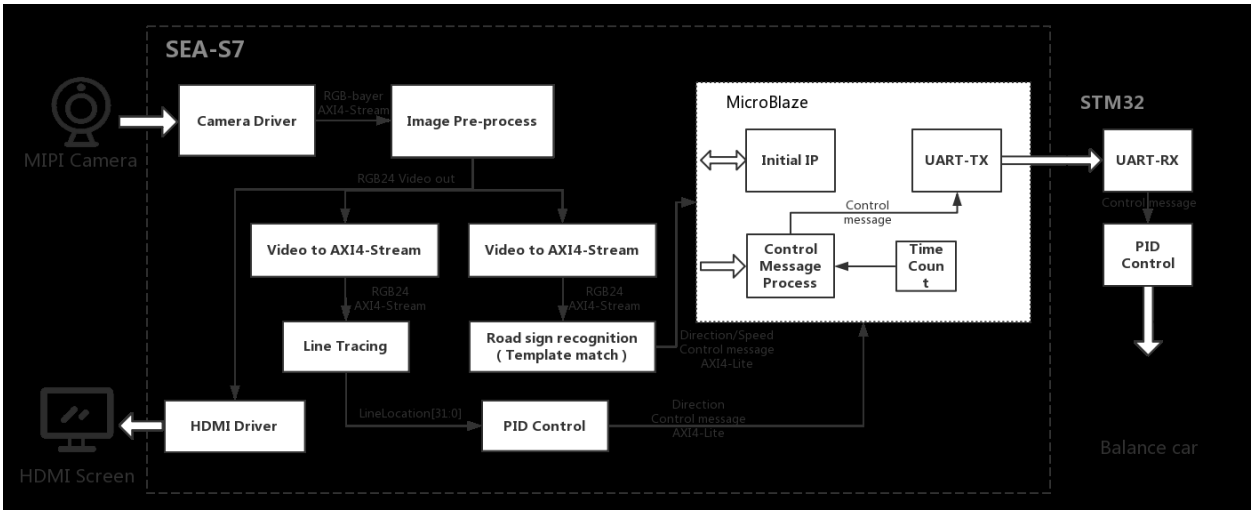
实现流程如下图所示，在开始时，首先测试所有的 Demo 和例程，确保其正确性。然后根据系统的功能划分，分三个不同的模块分别独立并行进行设计实现（串口调试，PID 运动控制，图像处理）。其中 PID 部分仿真实现后，进行 Demo 的研究和修改，减少不需要的功能 IP 节省资源。在系统中带各部分的功能单独测试实现后，再两两依次进行联调，不断完善和添加新的功能到已测试的系统中去，直到实现全部的功能。图像处理部分首先实现寻线功能，然后加入系统调节参数，使平衡车实现巡线运动。之后再将图像处理的路标识别部分添加入已完成的系统进行调试。



4 系统实现

4.1 整体框架

根据上述流程安排，最终本项目完成的系统工程在 Vivado 中的整体框架图如下图所示：



由于 SEA-S7 的板载芯片是 xilinx 的 Spartan7 系列芯片，芯片本身只有 FPGA 资源，所以为了使部分功能在软件上实现，本项目在 FPGA 上搭建了 MicroBlaze 软核作为 PS 端，用于部分 HLS 生成 IP 核的初始化、数据读取处理和与 STM32 的串口通信。

MIPI 摄像头从 PL 端进行驱动和数据读取，经由 IP 转换为 2 Lane 的 AXI4-Stream 格式。由于此处生成的 AXI4-Stream 流数据协议较为复杂，且该项目的图像处理模块分两模块并行执行，因此该 AXI4-Stream 流数据无法直接进行图像处理。需再经过两个 RTL IP 进行 RGB-bayer 到 RGB 的数据格式的转换，生成具有场同步信号和行同步信号的 pixel 视频流，最后转换为 2 股 RGB24 的 AXI4-Stream 流分别用于寻线 IP 和路标匹配 IP 进行图像处理。

寻线 IP 会在处理后返回 32 位的图像黑线中线值，PID PI 将根据识别出的黑线中线值与图像中值的差距进行闭环控制，存储一个 32 位的偏移控制信息到寄存器，供 MicroBlaze 读取使用。路标匹配 IP 将对采集到的图像进行模板匹配，如识别到路标，则将对应的动作指令存储到寄存器，供 MicroBlaze 读取使用。由 MicroBlaze 对数据进行简单处理后以一定的频率通过串口传递给平衡车 STM32 进行运动方向控制。

在调试过程中，往往需要直观观察图像，因此本项目在 SEA-S7 中设置了 HDMI 驱动，可将图像直接投影到 HDMI 显示屏上进行观测。需要注意的是，本系统中处理的图像大小为 720P.

4.2 系统图像处理部分设计

4.2.1 寻迹 IP 设计

寻迹 IP 采用 AXI4-STREAM 接口读取到 720P 大小的彩色图像，然后将彩色图像通过设定阈值转换成二值图像。通过判断图像中连续 N 个像素点的值都为黑色确定路线的坐标。最后通过各行的坐标判断选出最佳的坐标值返回。

部分代码如下：

```
//#include "line.h"
#include "gray2binValue.h"
int hls_line(AXI_STREAM_IN& src_axi, int rows, int cols){
#pragma HLS INTERFACE axis port=src_axi
    int output;
    RGB_IMAGE img_01(rows, cols);
    //RGB_IMAGE img_03(rows, cols);
```

```
    GRAY_IMAGE img1(rows, cols);
    //GRAY_IMAGE img2(rows, cols);
    //GRAY_IMAGE img3(rows, cols);
#pragma HLS dataflow
    hls::AXIvideo2Mat(src_axi, img_01);
    hls::CvtColor<HLS_RGB2GRAY>(img_01, img1);
    //hls::Duplicate(img1,img3,img4);

    //hls::Point_<int> c_point;
    //c_point.x=-1;
    //c_point.y=-1;
    //滤波
    //hls::Filter2D(img1,img2,smooth_kernel,c_point);
    gray2bin(img1,output);
    //drawLine(img4,img5,value);
    //hls::CvtColor<HLS_GRAY2RGB>(img5, img_03);
    //hls::HoughLines2()
    //printf("%d\f",value);
    //hls::Mat2AXIvideo(img_03, dst_axi);
```

4.2.2 路标匹配 IP 设计

路标 IP 采用 AXI4-STREAM 接口读取到 720P 大小的彩色图像，通过收集彩色图像的颜色直方图与先前存储在 IP 中的四个路标的颜色直方图进行对比，判断出目前是否出现路标并判断出该路标属于哪个路标。其中四个路别分别为停止、慢、左转和右转。

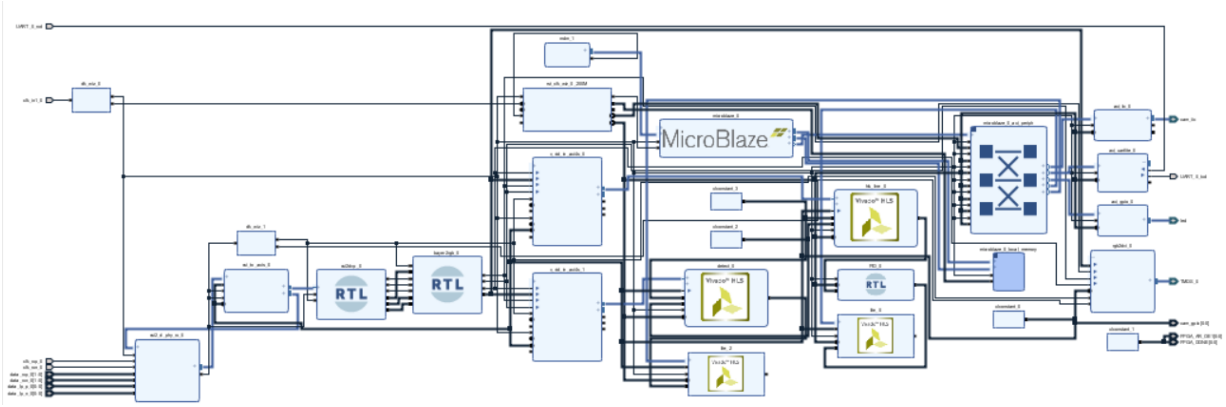
部分代码如下：

```
#include "detect.h"
#include "segmentation.h"
#include "math.h"
int detect(AXI_STREAM_IN& src_axi,int rows, int cols){
#pragma HLS INTERFACE axis port=src_axi
    int output;
    RGB_IMAGE img_01(rows, cols);
    GRAY_IMAGE img1(rows, cols);
    //hls::Mat<SEG_HEIGHT, SEG_WIDTH, HLS_8UC1> img2;
#pragma HLS dataflow
    hls::AXIvideo2Mat(src_axi, img_01);
    //提取红色通道中的值
    hls::CvtColor<HLS_RGB2GRAY>(img_01, img1);
    //图像分割
    segment(img1,output);
    //hls::Mat2AXIvideo(img2, dst_axi);
    return output;
}
```

```
#include "segmentation.h"
void segment(hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC1> &_src,int
&output)
{
	GRAY gray;
	int sum=0;
	loop_height: for(int i= 0;i< MAX_HEIGHT;i++)
	{
		loop_width: for (int j= 0;j< MAX_WIDTH;j++)
		{
#pragma HLS PIPELINE
			_src.data_stream[0]>>gray;
			if(gray<THRES){
				sum++;
			}
		}
	}
	if(abs(sum-LEFT)<=ERROR){
		output=2;
	}else{
		if(abs(sum-RIGHT)<=ERROR){
			output=3;
		}else{
			if(abs(sum-STOP)<=ERROR){
				output=1;
			}else{
				if(sum<MIN){
					output=0;
				}else{
					output=sum;
				}
			}
		}
	}
}
```

4.3 系统 PL 部分搭建

系统的 PS 部分工程图如下图所示，并具体分析其中主要的两个部分：PID 设计和视频通路。



4.3.1 PID IP 设计

PID 会根据平衡车在黑线上的偏移量来不断调整平衡车的运动方向。它主要由 P,I,D 三参数的值结合返回信号做一定的运算进行控制。

比例（P）控制

比例控制是一种最简单的控制方式。其控制器的输出与输入误差信号成比例关系。当仅有比例控制时系统输出存在稳态误差（Steady-state error）。

积分（I）控制

在积分控制中，控制器的输出与输入误差信号的积分成正比关系。对一个自动控制系统，如果在进入稳态后存在稳态误差，则称这个控制系统是有稳态误差的 或简称有差系统（System with Steady-state Error）。为了消除稳态误差，在控制器中必须引入“积分项”。积分项对误差取决于时间的积分，随着时间的增加，积分项会增大。这样，即便误差很小，积 分项也会随着时间的增加而加大，它推动控制器的输出增大使稳态误差进一步减小，直到接近于零。因此，比例+积分（PI）控制器，可以使系统在进入稳态后几乎无稳 态误差。

微分（D）控制

在微分控制中，控制器的输出与输入误差信号的微分（即误差的变化率）成正比关系。自动控制系统在克服误差的调节过程中可能会出现振荡甚至失稳。其原因是由于存在有较大惯性组件（环节）或有滞后（delay）组件，具有抑制误差的作用， 其变化总是落后于误差的变化。解决的办法是使抑制误差的作用的变化“超前”，即在误差接近零时，抑制误差的作用就应该是零。这就是说，在控制器中仅引入 “比例”项往往是不够的，比例项的作用仅是放大误差的幅值，而需要增加的是“微分项”，它能预测误差变化的趋势，这样，具有比例+微分的控制器，就能 够提前使抑制误差的控制作用等于零，甚至为负值，从而避免了被控量的严重超调。所以对有较大惯性或滞后的被控对象，比例+微分（PD）控制器能改善系统在调节过程中的动态特性。

在本系统里，在每个时钟周期，该 IP 会获取寻线 IP 传来的黑线中值，与标准图像的中值求差，作为平衡车的偏移量，即输入的误差信号。计算后输出一个有效的控制信号。主要的 PID 计算部分代码如下：

```
`timescale 1ns / 1ps
module PID(
        input clk,
        input rst,
        input signed [31:0] LineLocation,
        output wire signed [31:0] ConMsg
```

```
    );
    parameter P=1;
    parameter I=0;
    parameter D=2;
    parameter M=2;
    parameter MIDLINE=640;
    wire signed [31:0] error;
    reg Mclk=0;
    reg [31:0] count=0;
    reg signed [31:0] DifError=0,SumError=0,LastError=0;
    assign error = LineLocation-MIDLINE;
    assign ConMsg = P*error+I*SumError+D*DifError;
  ……

    always@(posedge clk)
    begin
        if(!rst)
begin
            SumError<=0; DifError<=0;SumError<=0;LastError<=0;
        end
        else
        begin
        if (LineLocation<30)
        begin
            SumError<=SumError-640;
            DifError<=LastError+640;
            LastError<=+640;
        end
        else if (LineLocation>1250)
             begin
                SumError<=SumError+640;
                DifError<=LastError-640;
                LastError<=640;
             end
             else
             begin
                SumError<=SumError+error;
                DifError<=LastError-error;
                LastError<=error;
             end

        end
    end
endmodule
```

考虑到本项目中 PID 算法用于实现寻线方向控制，根据经验将 I 值设为 0。同时为了防止过度偏移并匹配寻线算法，需要对偏移量进行限幅：当中线值小于 30 或大于 1250 时，视误差为-640 或 640。经调试后，系统设置 P=2，D=2。

4.3.2 图像处理通路设计

由于 MIPI Camera 输入到 HDMI 输出的通路已在 Demo 中完成，此处不再多做赘述。主要分析如何将自定义的图像处理 IP 添加到原有的视频通路中去。

MIPI 摄像头从 PL 端进行驱动和数据读取，图像大小为 720P,经由 IP 转换为 2 Lane 的 AXI4-Stream 格式。由于此处生成的 AXI4-Stream 流数据协议较为复杂，且该项目的图像处理模块分两模块并行执行，因此该 AXI4-Stream 流数据无法直接进行图像处理。需再经过两个 RTL IP 进行 RGB-bayer 到 RGB 的数据格式的转换，生成具有场同步信号和行同步信号的 pixel 视频流，最后转换为 2 股 RGB24 的 AXI4-Stream 流分别用于寻线 IP 和路标匹配 IP 进行图像处理。寻线 IP 会在处理后返回 32 位的图像黑线中线值

4.4 系统 PS 部分搭建

4.4.1 串口通信

串口通信（Serial Communications）的概念非常简单，串口按位（bit）发送和接收字节。典型地，串口用于 ASCII 码字符的传输。通信使用 3 根线完成，分别是地线、发送、接收。由于串口通信是异步的，端口能够在一根线上发送数据同时在另一根线上接收数据。其他线用于握手，但不是必须的。串口通信最重要的参数是波特率、数据位、停止位和奇偶校验。对于两个进行通信的端口，这些参数必须匹配。

波特率

这是一个衡量符号传输速率的参数。指的是信号被调制以后在单位时间内的变化，即单位时间内载波参数变化的次数，如每秒钟传送 240 个字符，而每个字符格式包含 10 位（1 个起始位，1 个停止位，8 个数据位），这时的波特率为 240Bd，比特率为 10 位*240 个/秒=2400bps。一般调制速率大于波特率，比如曼彻斯特编码）。在本系统中 SEA-S7 与 PC 通信时比特率为 115200，与 STM32 通信时比特率需设置为 9600.

数据位

这是衡量通信中实际数据位的参数。当计算机发送一个信息包，实际的数据往往不会是 8 位的，标准的值是 6、7 和 8 位。如何设置取决于想传送的信息。比如，标准的 ASCII 码是 0～127（7 位）。扩展的 ASCII 码是 0～255（8 位）。如果数据使用简单的文本（标准 ASCII 码），那么每个数据包使用 7 位数据。每个包是指一个字节，包括开始/停止位，数据位和奇偶校验位。

停止位

用于表示单个包的最后一位。典型的值为 1，1.5 和 2 位。由于数据是在传输线上定时的，并且每一个设备有其自己的时钟，很可能在通信中两台设备间出现了小小的不同步。因此停止位不仅仅是表示传输的结束，并且提供计算机校正时钟同步的机会。适用于停止位的位数越多，不同时钟同步的容忍程度越大，但是数据传输率同时也越慢。

奇偶校验位

在串口通信中一种简单的检错方式。有四种检错方式：偶、奇、高和低。当然没有校验位也是可以的。对于偶和奇校验的情况，串口会设置校验位（数据位后面的一位），用一个值确保传输的数据有偶个或者奇个逻辑高位。例如，如果数据是 011，那么对于偶校验，校验位为 0，保证逻辑高的位数是偶数个。如果是奇校验，校验位为 1，这样就有 3 个逻辑高位。高位和低位不真正的检查数据，简单置位逻辑高或者逻辑低校验。这样使得接收设备能够知道一个位的状态，有机会判断是否有噪声干扰了通信或者是否传输和接收数据是否不同步。

STM32 端串口通信协议经解析后，得到的部分传输协议如下：

通信串口：UART1

起始位：'$'

终止位：'#'

第一位字符是平衡车的运动控制信号，如下：

```
#define      run_car      '1'//按键前
#define      back_car     '2'//按键后
#define      left_car     '3'//按键左
#define      right_car    '4'//按键右
#define      stop_car     '0'//按键停
```

以此开发板为接收端，将 SEA-S7 作为发送端进行串口开发，首先根据提供的开发板 datasheet，注意到使能 SEA-S7 的串口需将 FPGA_AR_OE 和 FPGA_DONE 置高，并在 MicroBlaze 中设置波特率为 9600 以匹配 STM32 的波特率，从而实现串口通信。

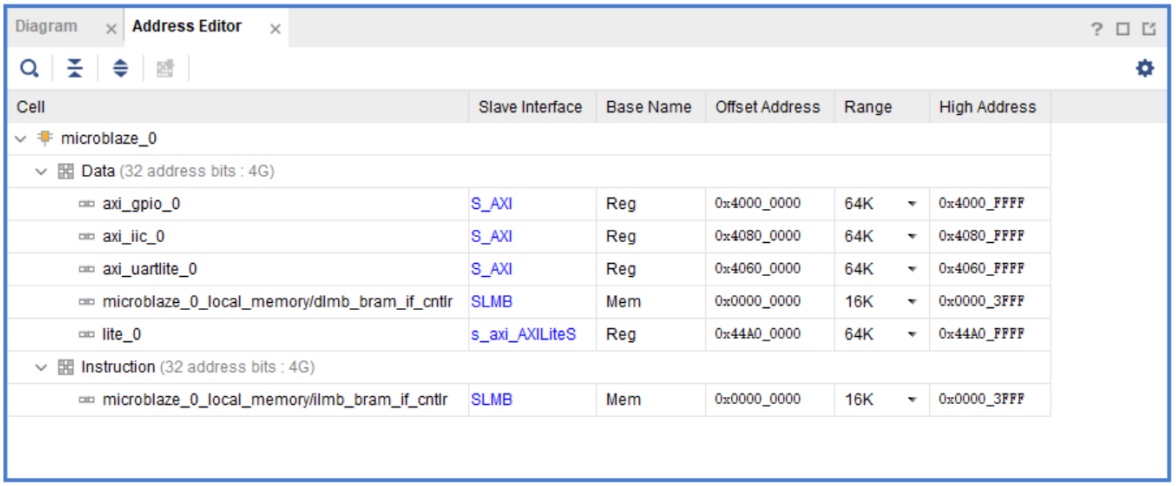
需要注意的是，由于 SEA-S7 开发板无内存，需要对 LocalMemory 进行设置，使用 BRAM 进行缓存。

4.4.2 控制信号处理

由于 PL 端信号处理速度快且处理模块并行执行的特性，在 MicroBlaze 中需设置控制信号的采样周期和并协调不同控制信号之间的优先级，以防过度频繁的控制指令无法在 STM32 中得到及时处理。

主要代码如下在本项目中，经多次调试，最佳的采样周期为 0.49s。控制指令处理优先级为：路标匹配控制信息优先于寻线控制信息。

需要注意的是，由于 SEA-S7 开发板 BRAM 资源又十分有限。所以 instruction 在 Vivado 中设置为 16K，过大和过小均会造成系统问题。具体的地址分配如下图所示：



Cell	Slave Interface	Base Name	Offset Address	Range	High Address
microblaze_0					
Data (32 address bits : 4G)					
axi_gpio_0	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
axi_iic_0	S_AXI	Reg	0x4080_0000	64K	0x4080_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
microblaze_0_local_memory/dlmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	16K	0x0000_3FFF
lite_0	s_axi_AXILiteS	Reg	0x44A0_0000	64K	0x44A0_FFFF
Instruction (32 address bits : 4G)					
microblaze_0_local_memory/ilmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	16K	0x0000_3FFF

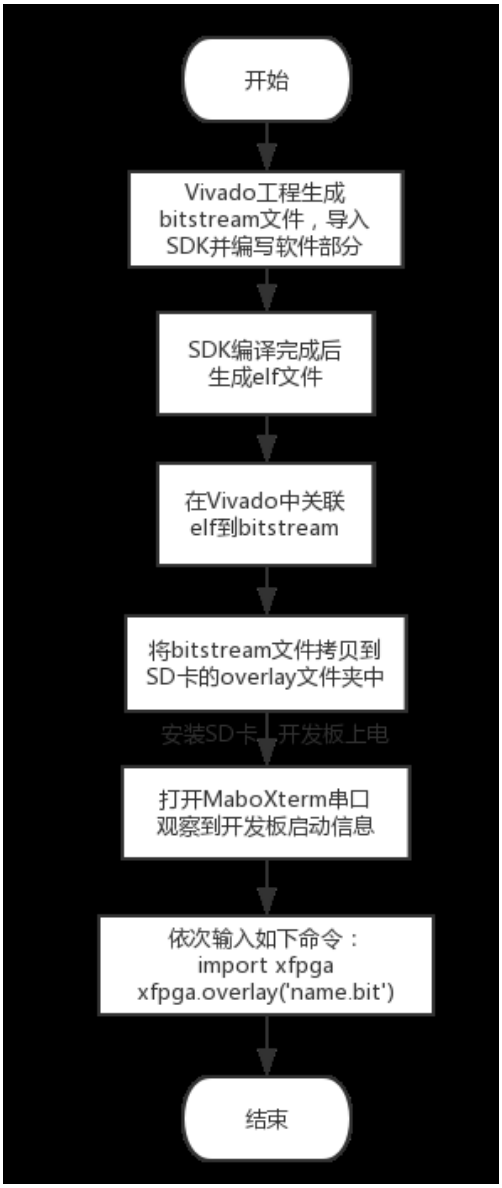
SDK 部分代码如下：

```
#define XLITE_ADDR    XPAR_LITE_0_S_AXI_AXILITES_BASEADDR
#define XLITE2_ADDR XPAR_LITE_2_S_AXI_AXILITES_BASEADDR
static XLite xlite; //巡线
static XLite xlite2;//路标检测
……
void writeReg(u16 reg_addr, u8 reg_data)
u8 readReg(u16 reg_addr)
void delay(long ms)
void control(signed int pid_data)
{
    if(pid_data < 0 && pid_data > -620)
    {
        //左转
```

```
            print("$3#\r\n");
            delay((-pid_data)*60);
            //前进
            print("$1#\r\n");
        }
        if(pid_data > 0 && pid_data < 750)
        {
            //右转
            print("$4#\r\n");
            delay((pid_data)*60);
            //前进
            print("$1#\r\n");
        }
        if(pid_data <= -620 || pid_data >= 750)
        {
            //停车  左侧线出界或右侧线出界
            //print("$0#\r\n");
        }
}

int main()
{
        signed int data, detect;
        data = readReg(0x300a);
        //xil_printf("%x\r\n", data);
        data = readReg(0x300b);
        //xil_printf("%x\r\n", data);

        int i;
        for(i = 0; cfg_ov5647_init[i].addr != 0xffff; i++)
            writeReg(cfg_ov5647_init[i].addr, cfg_ov5647_init[i].data);

        //initial xlite
        int state = XLite_Initialize(&xlite, XPAR_LITE_0_DEVICE_ID);
        if(state != 0)
        {
            xil_printf("XLite_Initialize Failed!\r\n");
        }
        int state2 = XLite_Initialize(&xlite2, XPAR_LITE_2_DEVICE_ID);
        if(state2 != 0)
        {
            xil_printf("XLite2_Initialize Failed!\r\n");
        }
```

5 系统调试

本项目中 SEA-S7 开发板的启动流程如下：



第四部分

完成情况及性能参数 /Final Design & Performance Parameters

（作品完成情况，附照片/视频链接）

1 图像处理 IP 的资源利用率

Synthesis(solution1)

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	11.000	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
925964	926644	925957	926644	dataflow

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	32
FIFO	0	-	50	324
Instance	0	3	1698	2342
Memory	-	-	-	-
Multiplexer	-	-	-	36
Register	-	-	6	-
Total	0	3	1754	2734
Available	20	20	16000	8000
Utilization (%)	0	15	10	34

寻线 IP

Synthesis(solution1)

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	11.000	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
921612	926644	921604	926644	dataflow

Detail

Instance

Loop

Utilization Estimates

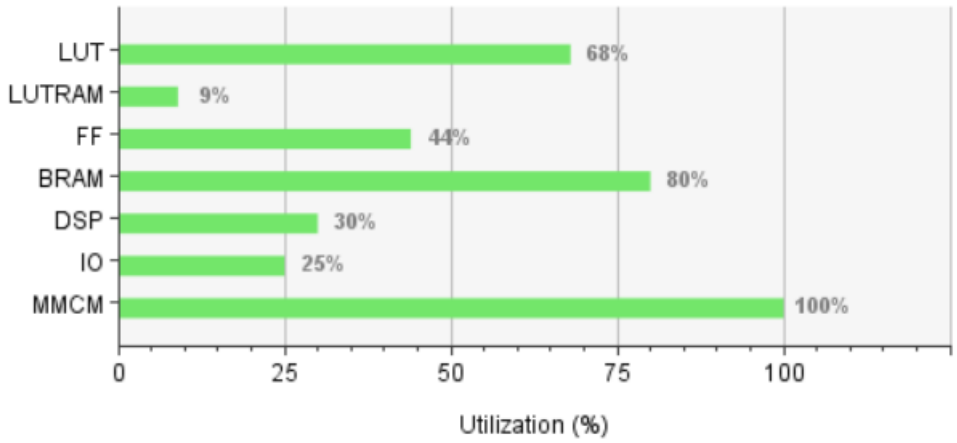
Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	32
FIFO	0	-	50	322
Instance	0	3	817	1659
Memory	-	-	-	-
Multiplexer	-	-	-	36
Register	-	-	6	-
Total	0	3	873	2049
Available	20	20	16000	8000
Utilization (%)	0	15	5	25

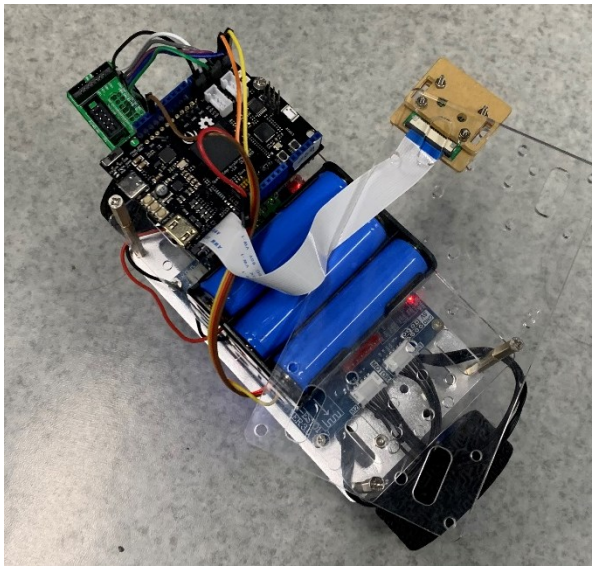
路标识别 IP

2 整个系统的资源利用率

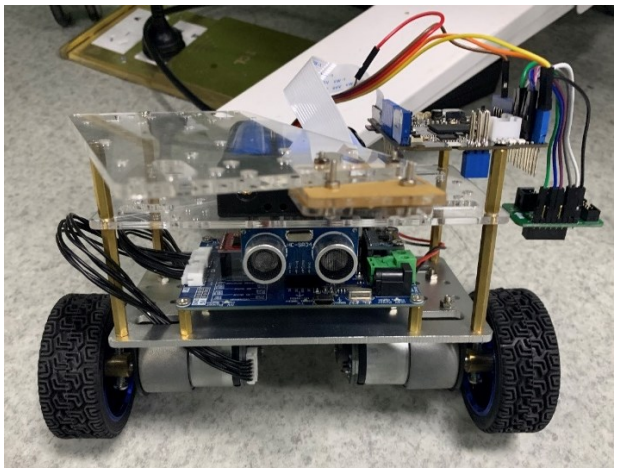
Resource	Utilization	Available	Utilization %
LUT	5477	8000	68.46
LUTRAM	222	2400	9.25
FF	7055	16000	44.09
BRAM	8	10	80.00
DSP	6	20	30.00
IO	25	100	25.00
MMCM	2	2	100.00



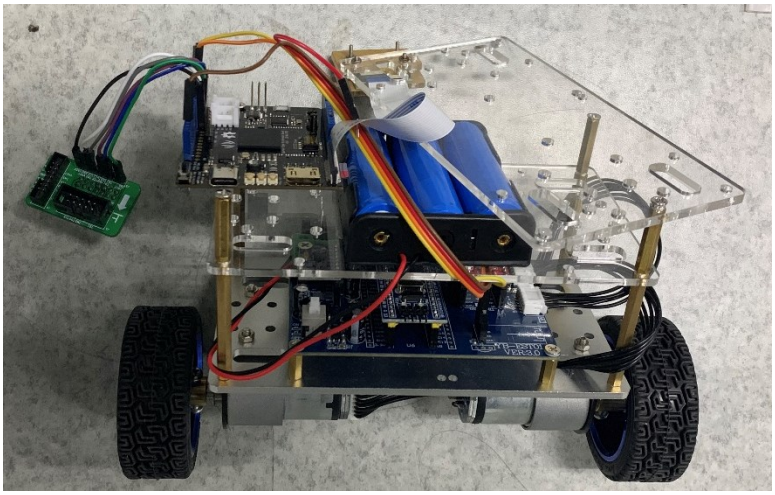
3 作品实物展示



正面图



侧面图



背面图

第五部分

项目总结 /Conclusions

(项目中所用到的知识点，项目收获及心得)

1 知识点

分类	具体内容	环境	备注
软件的使用	Vivado		SEA-S7
	Vivado_HLS		SEA-S7
	SDK		SEA-S7
	Keil		STM32
	MobaXterm		注意波特率设置
调试方法	Debug	Vivado	添加 ila IP,Xilinx 下载调试器
	生成并关联 elf 到 bitstream 文件	Vivado	
	SD 卡启动	MobaXterm	SD 卡，Type-C 线，串口调试工具
传输协议	AXI4-Stream	Vivado&Vvado_HLS	
	AXI4-Lite	Vivado&SDK	
	UART	Vivado&SDK	
硬件知识	verilog 封装 IP	Vivado	
	时序与系统时钟	Vivado	
	MicroBlaze 的设置	Vivado	
	SEA-S7 开发板串口的使用	Vivado	SEA-S7,杜邦线
软件知识	MicroBlaze 的使用	SDK	
	MicroBlaze 读取寄存器	SDK	
	MicroBlaze 使能串口,做发送端	SDK	
	Vivado_HLS 设计生成 IP	Vivado_HLS	
	IP 优化	Vivado_HLS	

2 项目收获及心得

王文杰：

时间如同白驹过隙，转眼间为期 12 天的 Xilinx 暑期学校即将结束。通过本次暑期学校，既开拓了我的眼界也提升了我在 FPGA 开发领域的认知。前期的 FPGA 课程，让我进一步理解了 HLS 开发的原理和相关优化指令的用法和区别；也让我在逻辑时许方面有了更深的认识。此外，DPU 的课程拓展了我的眼界，让我对 FPGA 的应用领域有了新的认知。后期的小组项目，我们小组的三位组员通力合作，分工明确，最终也顺利实现了项目的基本目标。在整个项目的执行过程中，我们遇到过模块之间接口不统一的问题，遇到过数据格式传输不统一以及不同模块间的时钟同步等等问题；最终，通过组员不间断地努力和熬夜，这些问题逐一被我们攻破。

从前期的理论学习到后期的项目实践，Xilinx 暑期学校让我体会到了理论到动手实践的整个过程。感受到了理论与实践的差别，同时也让我进一步熟悉了 FPGA 开发的开发流程，从逻辑到系统再到软件，也是一个完整的从底层硬件到上层应用的开发过程。这也进一步增强了我在嵌入式开发方面的理解。此外，团队合作的形式也进一步培养了我的团队合作能力，懂得了分工执行的重要性。总的来说，此次 Xilinx 暑期学校让我受益匪浅，感受颇多，在此感谢为此次活动无私奉献的老师和同学。

许巾一：

这次项目有幸能够和两位非常有能力的同学合作完成使我收获颇丰。一方面短时间内工作量巨大的项目要求我们以高强度高效率的方法设计和实现，从身心上都得到了很好的锻炼，作为项目进度的调控

者，我的时间规划能力也得到了提升；另一方面，作为一个学习硬件的同学，能够实际的和学习软件的同学分模块完成这样一个软硬件划分的系统，带给我新的体验，让我更深的认识到软硬件思维的不同，交流讨论之中也让我对软件有了更好的理解。

在项目从无到有创建的过程中，有很多我之前不甚了解的技术知识点得到了巩固和加强。比如 AXI4-Stream 流和一些 Xilinx，之前虽然我会使用，但对内部的协议和逻辑并不了解，在这次项目中我加深了对这些协议和 IP 的理解，能够更加灵活地使用它们。另外，我跟同组的同学学会了很多之前不知道的技巧：开发板的 Debug 调试方法，HLS 的一些接口设计和优化方法。也对前四天在暑期学校学习的一些内容进行了实际应用和练习，是我更理解之前学习的知识了。

另外，我也特别感谢这次暑期学校，经过两个星期的学习和动手操作，我觉得自己的能力得到了很高的提升。认识了很多技术高手，开阔了眼界。非常感谢为这个暑期学校的顺利举办，每天辛苦工作的工作人员和班委，非常高兴能和各位在这个暑期相遇。

何群芳：

通过这次项目，我收获了很多。了解了 HLS 各个接口的具体含义以及使用场景，并且知道了不同接口协议在 IP 核之间的具体传递流程，以及各个接口信号的作用。知道了如何通过 vivado 工具对设计进行调试，以及如何通过 vivado 工具查看设计的资源使用情况。了解了如何在没有处理器的纯 FPGA 器件上进行系统设计以及调试。学习了如何通过串口和平衡车进行通信和控制。知道了怎么通过 PID 对信号进行控制。

这次项目过程中遇到了很多问题，但经过一系列的调试以及交流，最后终于得到了不错的项目结果。所以通过这次项目了解到了自己的不足，加剧了自己学习的欲望，并学会了坚持，相信坚持就会成功。

第六部分

源代码/Source code

（作品源码 Github 链接，github 操作步骤见 *Github 项目上传简单教程*）

第七部分（技术总结/心得/笔记等分享）

（可分享的技术文档附件或已发表的文章链接）