

# 基于 PYNQ 和物联网的水果缺陷检测与分级装置

许霜烨 李成浩 黄一帆

## 第一部分 设计概述

### 1.1 设计目的

目前国内市面上普通的水果分拣装置大多往往只能对水果的大小进行区分，有很多带有局部腐烂等缺陷的水果还大多还需要额外的人工筛选。人工成本和时间成本也导致了我国水果产量大但出口额却占比很小的状况。为此我们小组提出了一种基于 PYNQ 的低成本水果缺陷检测装置，通过机器视觉再加上 pynq 处理图像的简易性，对有缺陷的水果进行区分，实现水果分拣的自动化的同时提高水果分拣的精确程度，减少人工成本。

### 1.2 应用领域

本作品主要应用在水果后期的分拣处理流水线上，可以极大地减少人工成本，此分拣系统用摄像头代替了人眼，图像识别代了人脑地判断，舵机与电机取代了双手，并且可扩展性强，通过对算法的改变，实现对不同水果的识别。提高了水果后期的分拣精度与效率，节约了人工成本，提高我国水果国际市场的竞争力

### 1.3 主要技术特点

相较于传统水果采摘后的后期分拣线只能线下对水果的大小进行分类，本作品不仅联网实现了对水果种类的识别，还可实现对几种水果表层腐坏部分进行捕获。我们预置了一些好坏水果的图片，将其数据给搭建的小型全链接 ANN 网络的进行训练。在保证光线充足的情况下，将水果图片转化为 HSV 颜色空间的格式，根据环境设置背景色，通过 inRange 函数获取了整个水果的边界，消除阴影对水果边界识别的影响。同时将原有格式的水果图片通过 Canny 算子或 Sobel 算子获取水果范围内的边界梯度，由于坏的地方梯度密度较大，计算梯度的面积，通过调节设定梯度阈值，可找到梯度较大的一系列‘梯度怀疑区’。最后将各个怀疑区中心 10X10 的部分输入预先训练的 ANN 网络中，得到好坏的分类信息，并提前选择出一系列 10x10 的区域，极大地减少了网络所需参数，配合 FPGA 的并行高速运算可实现较快的识别速度，从而对好坏水果进行剔除与计数。

### 1.4 关键性能指标

通过 PYNQ 开发板，利用 FPGA 并行特性，结合 Python 与 OpenCV 对摄像头采集到的水果图像进行处理，从而可以对苹果，橘子，香蕉等水果进行种类识别，并可以判断其表层明显的腐烂缺陷，进行分类计数，通过 VGA 显示屏与串口屏来实现图像与结果。

### 1.5 主要创新点

- (1)相较于传统的水果分类机，本产品结合了 FPGA，Python，OpenCV 可以对水果表层缺陷进行快速甄别。
- (2)通过 PYNQ 开发板可以实现联网实时监控生产流水线，并进行控制。
- (3)可以实现对不同种类的水果缺陷的判断，对流水线上的水果进行计数方便后续操作。

## 第二部分 系统组成及功能说明

### 2.1 整体介绍

给出芯片或系统整体框图，各子模块标注清楚，并进行整体的文字说明，需要表达出各模块之间的关系。

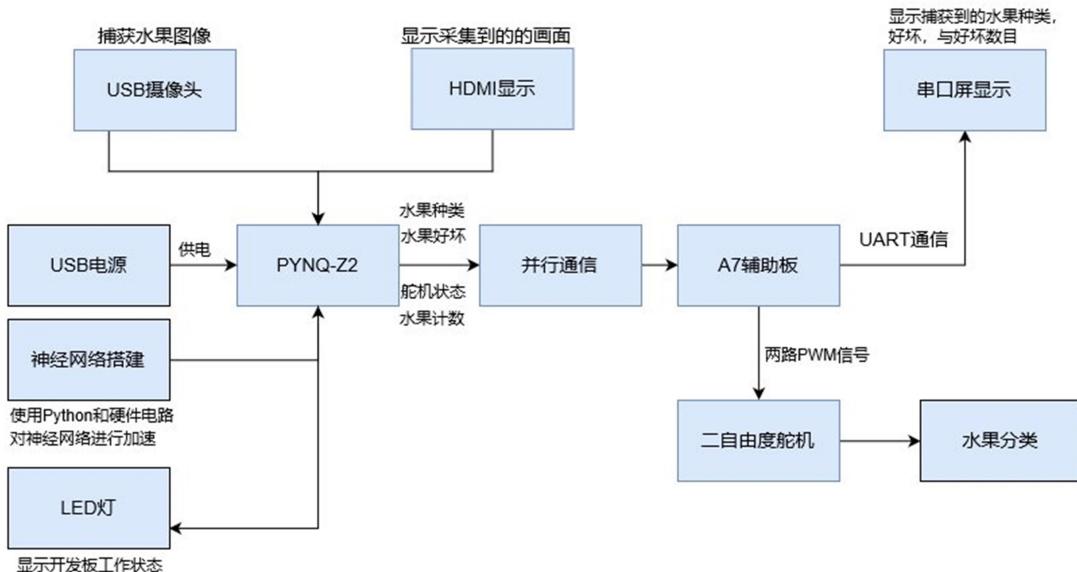


图 1 系统流程图

本系统主要由 Xilinx Zynq-7000 系列的 PYNQ-Z2 开发板作为主控中心，主要包含：USB 摄像头采集模块、A7 辅助板，图像处理模块、HDMI 显示模块、串口屏模块、舵机控制模块，组成。总体结构如图 1 所示。

首先，USB 摄像头采集到的图像发给 PYNQ 主控板，结合 Python 与 OpenCV，组件了一个小型全链接 ANN 训练网络，PYNQ 在搭建好的神经网络上对水果图像进行处理，利用 PL 执行并行算法的特性，能快速判断水果种类，好坏，通过 HDMI 接口在显示屏上显示水果图像并可以水果及其缺陷分别圈出，显示文字，然后发出种类判断，好坏判断，及舵机控制信号给辅助板，同时板载 LED 灯显示主控板工作状态。辅助板主要功能是将主控板发出的信号转化为具体的指令给外设，可以通过串口通信将数据发送给串口屏，串口屏实时显示水果种类，通过消抖模块接收脉冲可实现计数功能，同时还可以通过内部写的多路复用器发送不同的 PWM 波信号给二自由度舵机，使识别到的好坏水果朝不同方向分拣。

### 2.2 各模块介绍

#### 2.2.1 PYNQ 主控模块：

根据总体系统框图本系统主要由 Xilinx Zynq-7000 系列的 PYNQ-Z2 开发板作为主控中心，如图所示。其主芯片为 XC7Z020CLG400-1，该芯片是 FPGA+ARM 架构且可用 python 开发。同时提供了丰富的 python API 和完善的硬件设计流程，能实现较复杂的逻辑设计，13300 个逻辑 Slices，高达 650MHz 的工作频率，630K 的快速 Block RAM，220 个 DSP 切片，资源十分丰富。PYNQ 是一个开源框架，目标是使嵌入式编程人员能够在无需设计可编程逻辑电路的情况下即可充分发挥 Xilinx Zynq 全可编程 SoC 的功能。

与常规方式不同的是通过 PYNQ，设计人员可以通过 Python 语言和库，利用 Zynq 中融合可编程逻辑和微处理器的优势来快速构建更强大的嵌入式系统，其代码可直接在支持 PYNQ 的开发板上进行开发和测试。

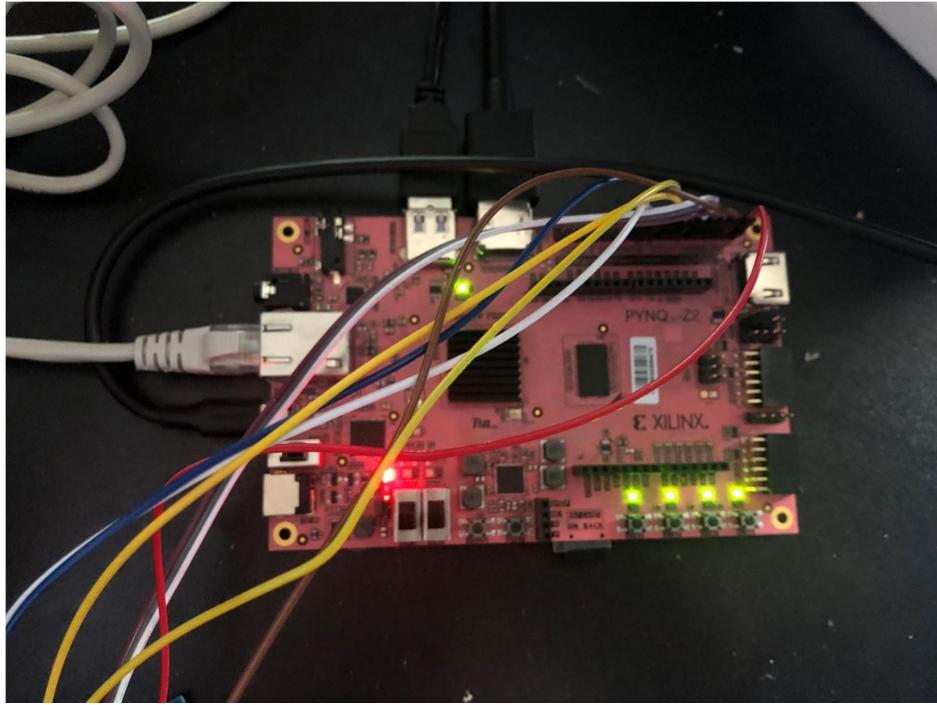


图 2 PYNQ 主控板

### 2.2.2 辅助控制模块：

本次辅助板采用得的是 XC7A35T-1FTG256C 黑板,负责执行 PYNQ 发送给他的指令，用其他的更廉价的 FPGA 开发板也可以实现本次功能。黑板与 PYQN 实现并行通信，PYNQ 将水果的种类及好坏指令发送给黑板，然后黑板发出两路 PWM 波信号来控制二自由度舵机，并且在串口屏上进行显示。

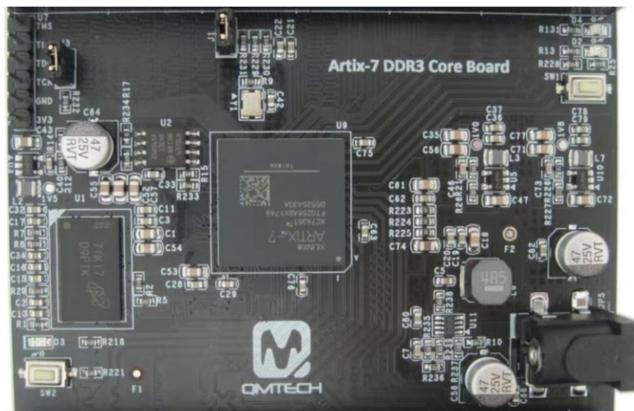


图 3 辅助板

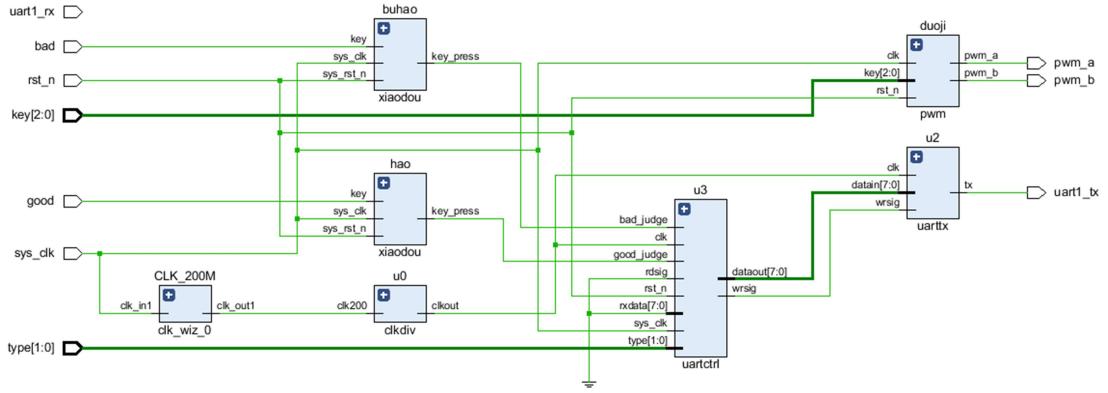


图 4 辅助系统原理图

### 2.2.3 摄像头模块：

该摄像头为 USB 免驱摄像头，参数如下：720P, 30 帧，2.6mm120°，低噪点，无畸变。可在 FPGA 开发板上调用对应的 python API 来控制摄像头使用。在摄像头采集模块中，我们将 python 内部 opencv 模块的图像处理函数加载至摄像头端，当摄像头采集视频流时可在显示屏实时观看所采集到的视频流。



图 5 USB 摄像头

### 2.2.4 HDMI 显示屏模块：

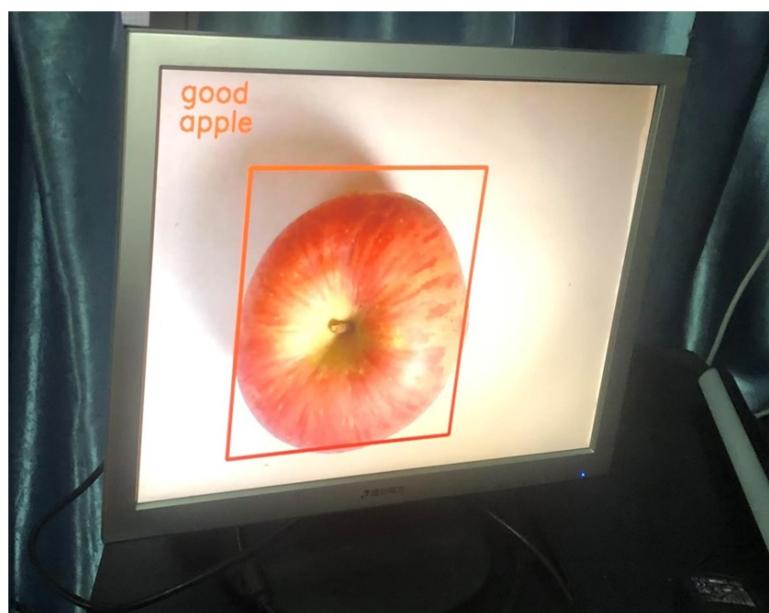


图 6 VGA 显示屏

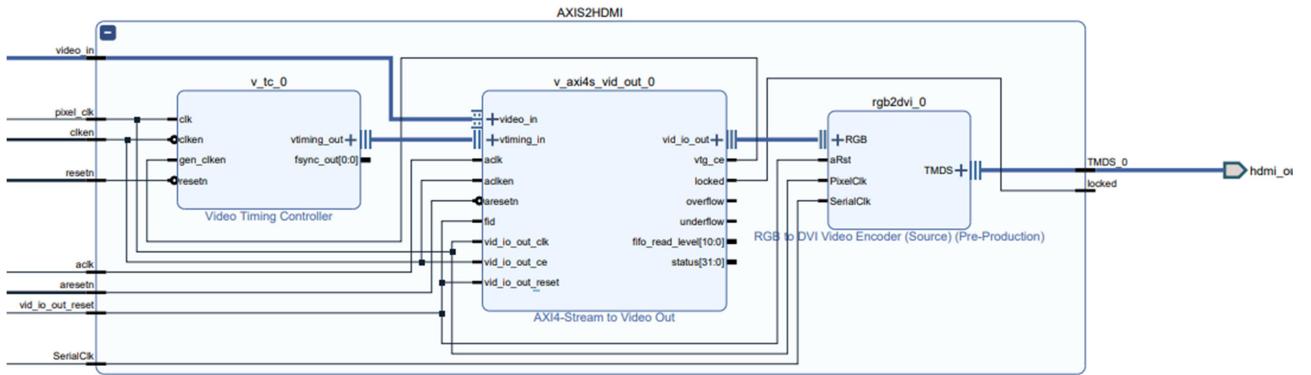


图 7 HDM IP 模块

本次显示屏采用的是清华同方显示屏，加上 VGA 转 HDMI 线可以实现显示的效果。

### 2.2.5 舵机模块：

系统设计中所使用的二自由度舵机云台为如图所示，可以实现二自由度运动，可以借此来控制水果缺陷检测完成后的去向。在非超载的情况下，舵机的转速、停止的位置只取决于脉冲信号的频率，而不受负载变化的影响，当步进驱动器接收到一个脉冲信号，它就驱动步进电机按设定的方向转动一定的角度，它的旋转的角度可通过控制脉冲的占空比进行改变调整。位置控制是通过发脉冲来控制的，位置控制模式一般是通过外部输入的脉冲的频率来确定转动速度的大小，通过脉冲的个数来确定转动的角度。一个周期是 20MS，一个周期的高脉冲范围在 0 到 2.5MS 时，上下舵机转动角度范围分别是 0 到 180 度和 0 到 270 度。只要设定相应转动角度对应的计数值，则可以实现相应的角度旋转。云台使用的舵机为 DS3115。

产品型号：DS3115

参数：

- 扭矩：4.8 – 6.8 – 7.2V 对应 13 – 15 – 17 kg / cm
- 死区：3 μ s
- 速度：4.8 – 6.8 – 7.2V 对应 0.16 – 0.14 – 0.13 sec/60°
- 工作电压：4.8–7.2V 直流电
- 重量：65 克
- 电机类型：直流电机
- 齿轮类型：金属齿轮
- 脉冲宽度：500–2500 μ s （中立信号值为 1500 μ s）
- 控制频率：50–330 Hz
- 尺寸：40x20x40.5mm

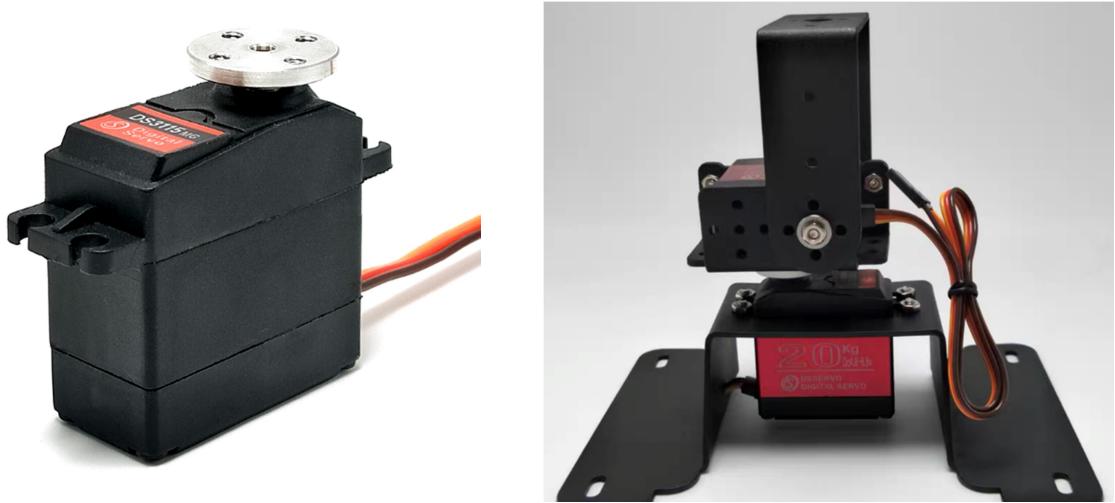


图 8 舵机

#### 2.2.6 串口屏显示模块：

原理：相较于传统 LCD 屏幕或是 VGA 显示屏，串口屏能极大地简化人们的开发过程，一切操作都可通过发送字符串等方式来完成，并且同时只用两根总线，占用端口资源少。



图 9 串口屏原理

我们在串口屏与 FPGA 信息交互时使用了 UART 通信协议。计算机和外部设备的连接，基本上使用了两类接口：串行接口和并行接口。并行接口是指数据的各个位同时进行传送，其特点是传输速度块，但当传输距离远、位数又多时，通信线路变复杂且成本提高。串行通信是指数据一位位地顺序传送，其特点是适合于远距离通信，通信线路简单，只要一对传输线就可以实现双向通信，从而大大降低了成本。会有指令数据通过 UART 通信协议发出，将传感器发出的信息传送给 FPGA 进行综合处理，串口屏会根据接收到的信息输出相应的数值。在这样的信息传输过程中是以 UART 通讯协议进行通信的。本作品的显示屏可以接收辅助板发送的指令，来显示需要水果的种类以及对好坏水果进行计数。

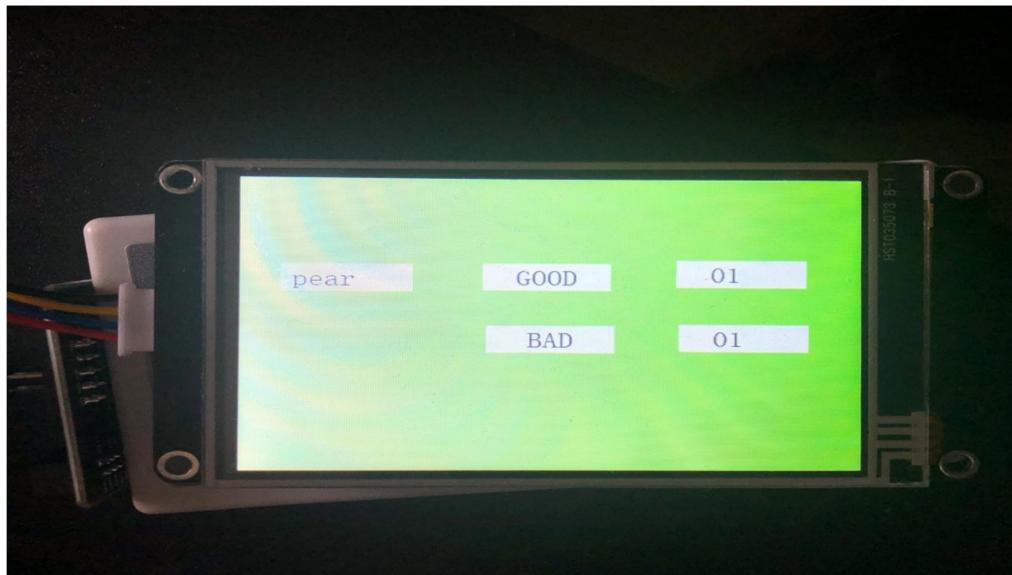


图 10 串口屏实物图

### 2.2.7 转接板：

为了连线方便，本次实验用洞洞板做了一个简易转接板，方便主控板 ZYNQ 与舵机，辅助板，串口屏连接。

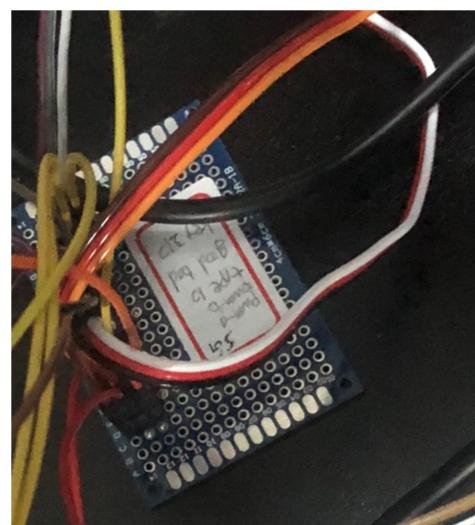


图 11 转接板

### 第三部分 完成情况及性能参数

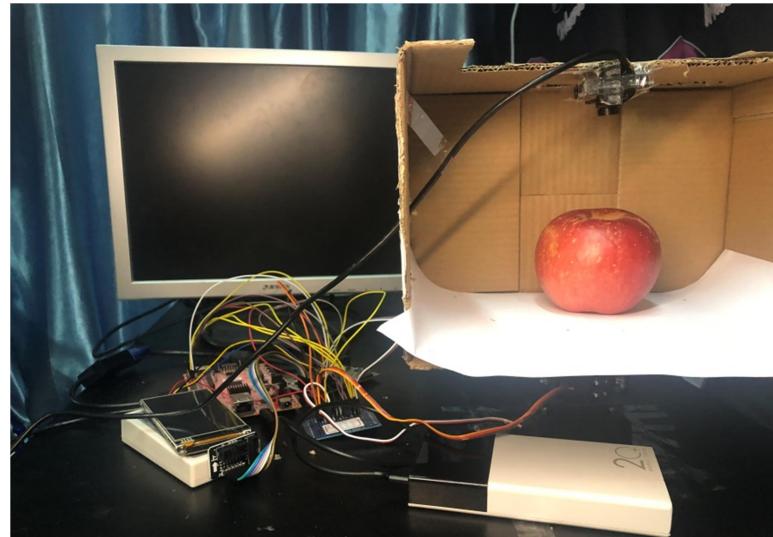


图 12 作品总览图

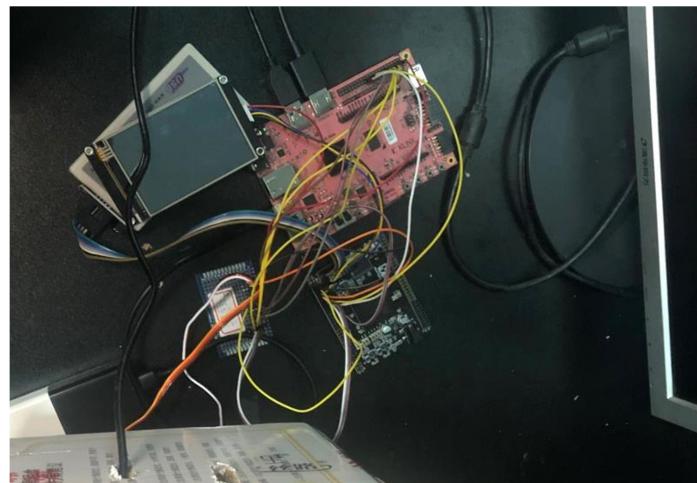


图 13 电路俯视图

作品总览图如图 10 所示，识别结果如下：图 12-14 分别对好橘子，好苹果，好香蕉进行测试，其测试结果完全正确并能够框出水果区域。图 15-图 16 为检测坏香蕉的结果，可以发现系统可以将香蕉与其腐烂处分别圈出，舵机也能进行相应地朝向（朝左转为好水果，朝右转为剔除坏水果）。图 17-图 18 为测试坏苹果，运行结果同香蕉。

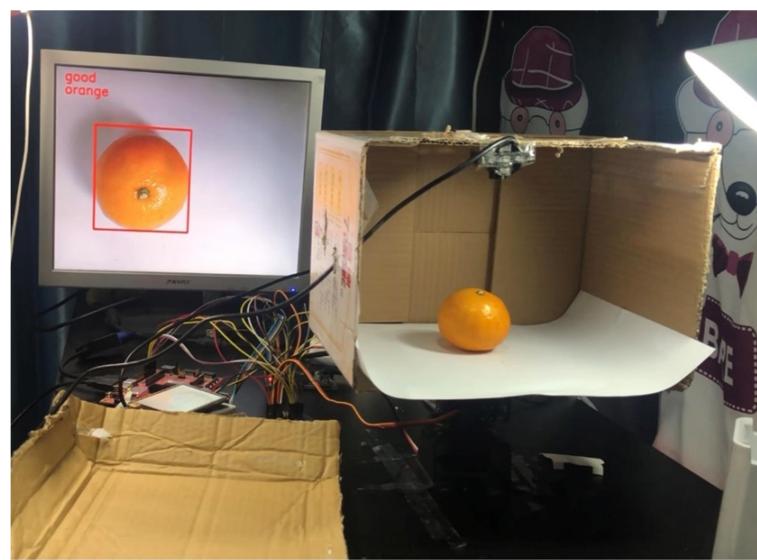


图 14 好橘子识别结果

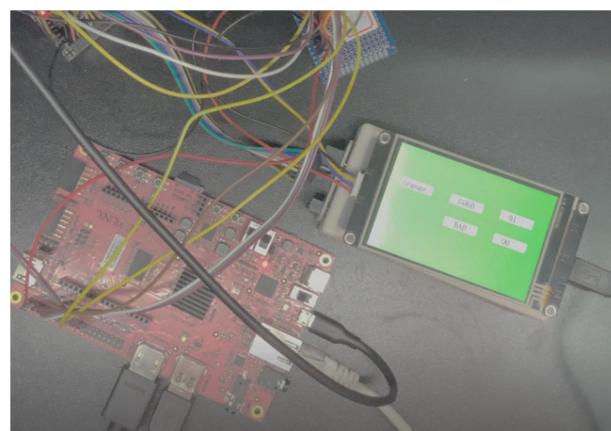


图 15 好橘子串口屏显示结果

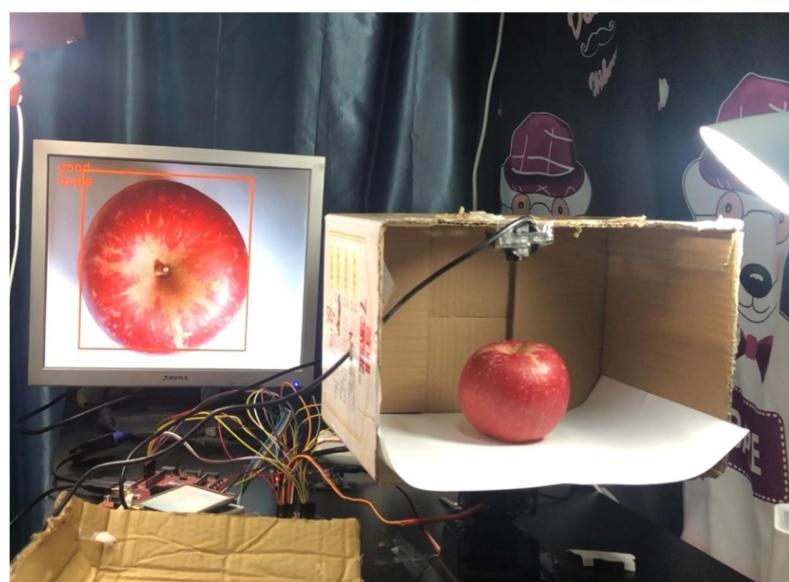


图 16 好苹果识别结果

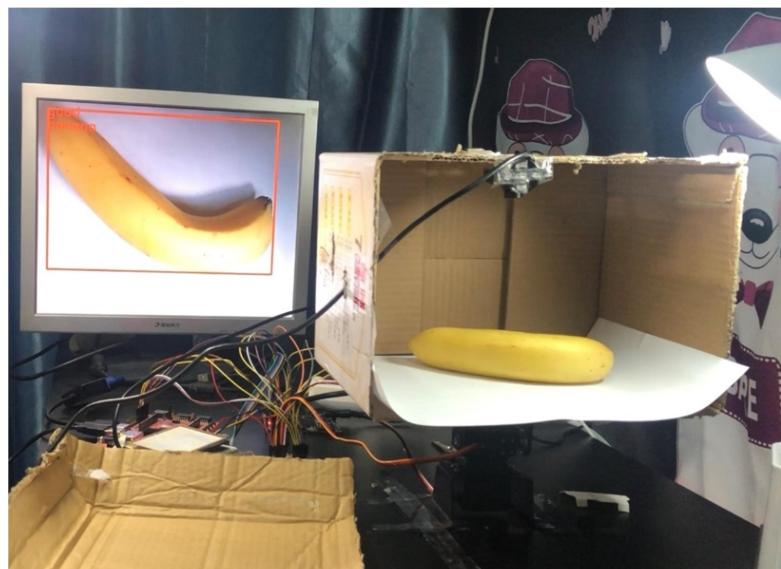


图 17 好香蕉识别结果



图 18 坏香蕉识别结果

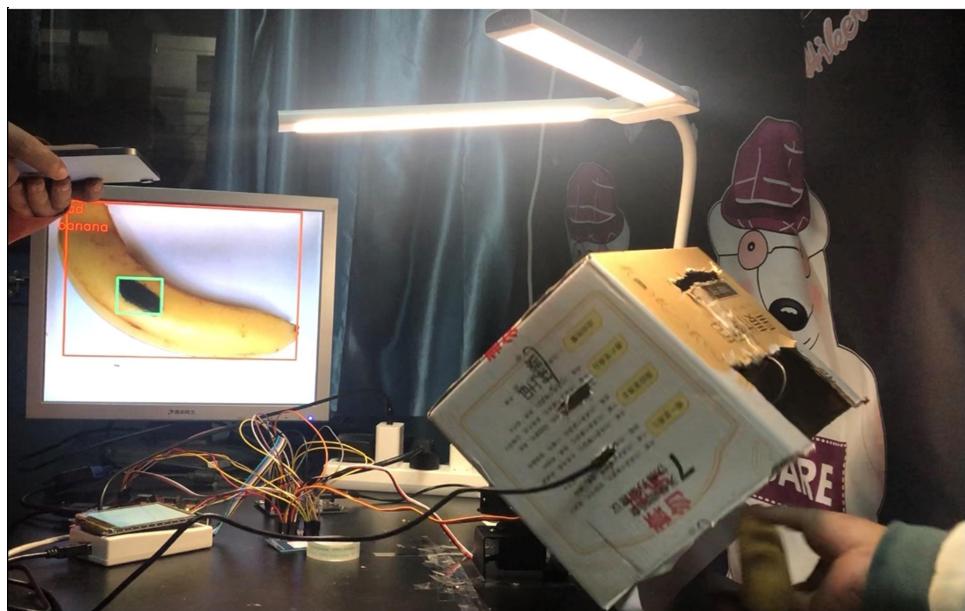


图 19 坏香蕉舵机转向

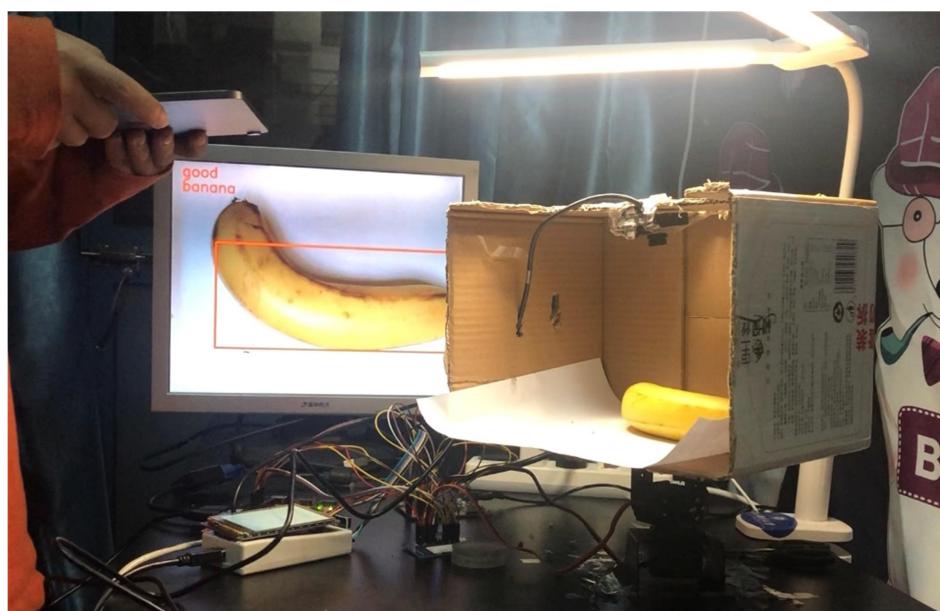


图 20 好香蕉舵机转向

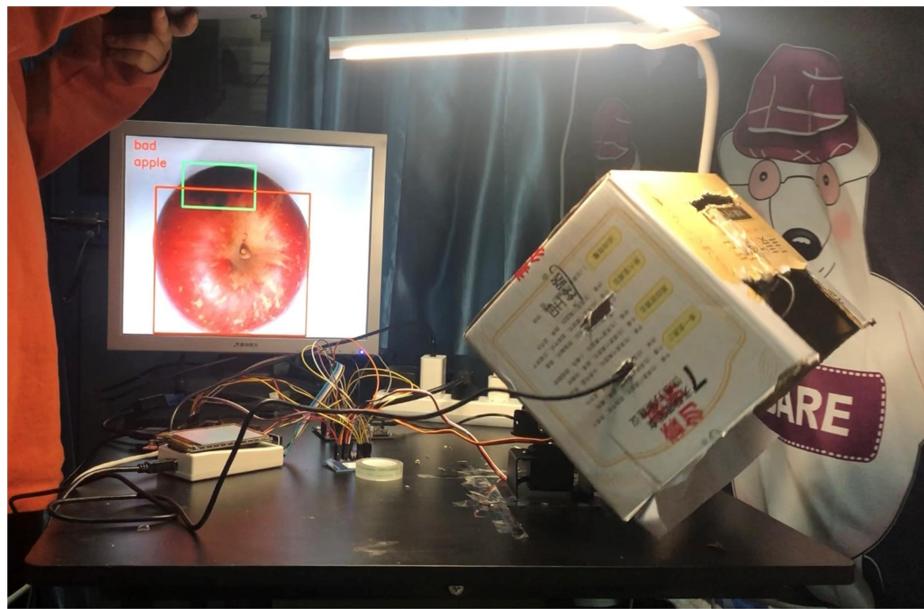


图 21 坏苹果识别结果

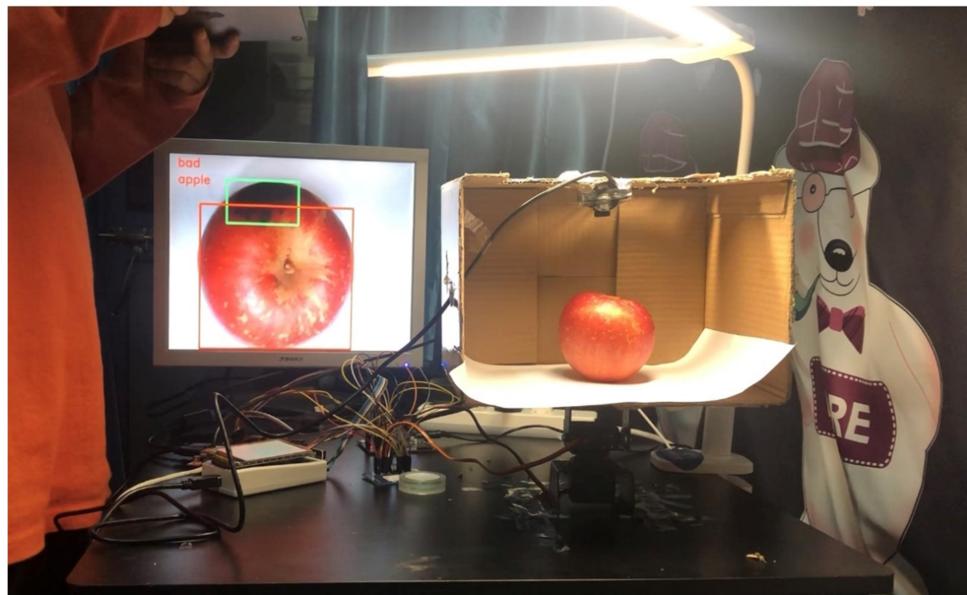


图 22 坏苹果舵机转向

总结一下，本系统可以联网识别不同种类的水果，并且能对其好坏进行判断，将坏的部分圈出，然后舵机进行相应的剔除操作，缺陷检测效果已经基本达到。

## 第四部分 总结

### 4.1 可扩展之处

本次试验对苹果，香蕉，橘子进行了实验，能够对这几种水果腐烂较明显的部分进行标记然后分拣。考虑到除去腐烂缺陷，有些水果还会出现发霉等不同情况的缺陷，因此我们未来将实现更精准的缺陷分类，优化算法，同时更换云台实现对水果多角度的图像捕获，对流水线上的水果进行更快速地精准到某一种类缺陷的判别，最终实现对具体某一种水果品质的自动化分级。

更进一步的话，如果配上更高级的光源与视觉设备等，还可以对其他农工业产品进行分级。

### 4.2 心得体会

这是我们三个人第一次参加 FPGA 大赛，大家都是 PYNQ 小白，对于 FPGA 的学习也全都靠热爱。参加这次比赛前，我们中有 2 人参加了电赛，学了点单片机与 FPGA，还有一兄弟参加了数模比赛，擅长用 Python 进行数据可视化分析与图像处理，于是我们最终选择了用 Xilinx 的 PYNQ 开发板。这次比赛，我们从准备到最后的作品提交前后大约用了一个多月的时间，由于专业课程较多，这个比赛几乎占用了我们绝大多数的课余时间。经过不断的改进设计和调试，我们的系统慢慢能够实现预期的功能，看到成品后，大家都认为付出是值得的。

项目整个系统完全由小组成员自主搭建设计完成，首先感谢主办方提供如此丰富的学习资料，能够让我们快速上手 PYNQ，然后比较辛苦的是有一些基本功能给的几个教程里并没有提到，需要我们从官方手册里一个个扒出来，然后再找一些博客来明确使用方法，这期间有很多简单的问题卡了我们很久，不如预装一个新的 Python 库等，由于身边没有人使用 PYNQ 可以来给我们提供教导，但最后大家也都齐心协力想办法解决了各种问题。从开始想实现一个水果分类机器的思想萌发，到最后真正的做出来实物，并完善一个个功能，力求用现有的知识去实现好每一个步骤，在学习的过程中深切体会到了先苦后甜的滋味，在每个模块调试成功的时候，欣慰感、成就感油然而生。同时也是因为这个竞赛，让我们有机会能够真正的化一个个零碎的知识为整体，付诸于实践同时也感受到了团队合作的重要性，

设计这个作品初衷是为了提高我国水果后期分拣流水线的分拣精度与效率，减少人力成本，增强我国水果在国际市场上的竞争力，让多余的物力人力去投入对水果实际品质的提高上。同时由于相较于国外高昂的光学设备，本作品成本较低，适合个体果农来使用，市场潜力大，在现有的流水线装置上配上我们的系统，可以在根据重量分拣水果的同时，实现对表层有缺陷的水果的剔除。

通过这一次比赛，我们收获颇多，这是一个挑战自我，提升自我的契机，同时也锻炼了我们的团队合作能力，并能让我们所学得到了应用。同时我们相信，PYNQ 这款开发板具有巨大的潜力，在这个物理网与信息爆炸的时代，其优点必将能被发挥得淋漓尽致。本水果分拣装置也仅仅是用了其中的一些功能罢了，所以接下来的日子里，我组的成员仍然会不断实践学习，增加本作品的功能，提高识别精度，扩大识别范围，使得系统更加完善。同时抽出时间来继续学习 FPGA 相关的知识，不断地提升自己。

## 第五部分 参考文献

1. PYNQ\_Z2\_User\_Manual\_v1.1
2. 黑金 AX7102 开发板 Verilog 教程
3. 夏宇闻-Verilog 经典教程第三版
4. 汤勇明. 搭建你的数字积木—数字电路与逻辑设计

## 第六部分 附录

1.顶层模块:

```
module fruit_top(
    input sys_clk,
    input rst_n,
    input [2:0]key,
    input [1:0]type,
    input good,
    input bad,
    input uart1_rx,           // UART 串口接收
    output uart1_tx,
    output pwm_a,
    output pwm_b
);
    wire wrsig;
    wire rdsig;
    wire sys_clk_ibufg;      //clock for 9600 uart port
    wire [7:0] txdata,rxdata; //串口发送数据和串口接收数据
    wire clk;//串口时钟
    clk_wiz_0 CLK_200M
    (
        // Clock out ports
        .clk_out1(sys_clk_ibufg), // output clk_out1 200m
        // Clock in ports
        .clk_in1(sys_clk));
//产生时钟的频率为 16*9600
clkdiv u0 (
    .clk200 (sys_clk_ibufg), //200Mhz 的晶振输入
    .clkout (clk)           //16 倍波特率的时钟
);
//串口接收程序
/*uartrx u1 (
    .clk (clk),             //16 倍波特率的时钟
    .rx (uart1_rx),          //串口接收
    .dataout (rxdata),        //uart 接收到的数据,一个字节
);
```

```

    .rdsig          (rdsig),           //uart 接收到数据有效
    .dataerror      (0),
    .frameerror    (0)
);/*/


//串口发送程序
uarttx u2 (
    .clk            (clk),           //16 倍波特率的时钟
    .tx             (uart1_tx),       //串口发送
    .datain        (txdata),         //uart 发送的数据
    .wrsig          (wrsig),          //uart 发送的数据有效
    .idle           (0)

);

wire good_press;
wire bad_press;
//串口数据发送控制程序
uartctrl u3 (
    .clk            (clk),
    .rst_n          (rst_n),          // .key(key),
    .good_judge(good_press),
    .bad_judge(bad_press),
    .type(type),
    .sys_clk(sys_clk),
    .rdsig          (rdsig),          //uart 接收到数据有效
    .rxdata         (rxdata),         //uart 接收到的数据
    .wrsig          (wrsig),          //uart 发送的数据有效
    .dataout        (txdata)          //uart 发送的数据，一个字节

);

pwm duoji(
    .clk(sys_clk),
    .rst_n(rst_n),
    .key(key),//前两位控制水平，后两位控制垂直
    .pwm_a(pwm_a),
    .pwm_b(pwm_b)
);

xiaodou hao(
    .sys_clk(sys_clk),           //外部 50M 时钟
    .sys_rst_n(rst_n),           //外部复位信号，低有效
    .key(good),                  //外部按键输入
    .key_press(good_press)        //按键数据有效信号

```

```

);
xiaodou buhao(
    .sys_clk(sys_clk),           //外部 50M 时钟
    .sys_rst_n(rst_n),          //外部复位信号，低有效
    .key(bad),                  //外部按键输入
    .key_press(bad_press)       //按键数据有效信号

);

endmodule

```

## 2. 分频模块

```

module clkdiv(clk200, clkout);
input clk200;                      //系统时钟
output clkout;                     //采样时钟输出
reg clkout;
reg [15:0] cnt;

//分频进程,对 200Mhz 的时钟 1302 分频
always @(posedge clk200)
begin
    if(cnt == 16'd560)//560
        begin
            clkout <= 1'b1;
            cnt <= cnt + 16'd1;
        end
    else if(cnt == 16'd1301)//1301
        begin
            clkout <= 1'b0;
            cnt <= 16'd0;
        end
    else
        begin
            cnt <= cnt + 16'd1;
        end
end
endmodule

```

## 3. 串口发送模块

```

module uarttx(clk, datain, wrsig, idle, tx);
input clk;                          //UART 时钟
input [7:0] datain;                 //需要发送的数据
input wrsig;                       //发送命令, 上升沿有效
output idle;                       //线路状态指示, 高为线路忙, 低为线路空闲
output tx;                         //发送数据信号

```

```

reg idle, tx;
reg send;
reg wrsigbuf, wrsigrise;
reg presult;
reg[7:0] cnt;           //计数器
parameter paritymode = 1'b0;

//检测发送命令是否有效，判断 wrsig 的上升沿
always @(posedge clk)
begin
    wrsigbuf <= wrsig;
    wrsigrise <= (~wrsigbuf) & wrsig;
end

always @(posedge clk)
begin
    if(wrsigrise && (~idle)) //当发送命令有效且线路为空闲时，启动新的数据发送进程
        begin
            send <= 1'b1;
        end
    else if(cnt == 8'd168)      //一帧资料发送结束
        begin
            send <= 1'b0;
        end
end

///////////////////////////////
//使用 168 个时钟发送一个数据（起始位、8 位数据、奇偶校验位、停止位），每位占用 16 个
时钟,结束位 8 个时钟//
/////////////////////////////
always @(posedge clk)
begin
    if(send == 1'b1)  begin
        case(cnt)           //tx 变低电平产生起始位，0~15 个时钟为发送起始位
        8'd0: begin
            tx <= 1'b0;
            idle <= 1'b1;
            cnt <= cnt + 8'd1;
        end
        8'd16: begin
            tx <= datain[0];   //发送数据位的低位 bit0,占用第 16~31 个时钟
            presult <= datain[0]^paritymode;
            idle <= 1'b1;
            cnt <= cnt + 8'd1;
        end
        8'd32: begin
    
```

```

tx <= datain[1];      //发送数据位的第 2 位 bit1,占用第 47~32 个时钟
presult <= datain[1]^presult;
idle <= 1'b1;
cnt <= cnt + 8'd1;
end
8'd48: begin
    tx <= datain[2];      //发送数据位的第 3 位 bit2,占用第 63~48 个时钟
    presult <= datain[2]^presult;
    idle <= 1'b1;
    cnt <= cnt + 8'd1;
end
8'd64: begin
    tx <= datain[3];      //发送数据位的第 4 位 bit3,占用第 79~64 个时钟
    presult <= datain[3]^presult;
    idle <= 1'b1;
    cnt <= cnt + 8'd1;
end
8'd80: begin
    tx <= datain[4];      //发送数据位的第 5 位 bit4,占用第 95~80 个时钟
    presult <= datain[4]^presult;
    idle <= 1'b1;
    cnt <= cnt + 8'd1;
end
8'd96: begin
    tx <= datain[5];      //发送数据位的第 6 位 bit5,占用第 111~96 个时钟
    presult <= datain[5]^presult;
    idle <= 1'b1;
    cnt <= cnt + 8'd1;
end
8'd112: begin
    tx <= datain[6];      //发送数据位的第 7 位 bit6,占用第 127~112 个时钟
    presult <= datain[6]^presult;
    idle <= 1'b1;
    cnt <= cnt + 8'd1;
end
8'd128: begin
    tx <= datain[7];      //发送数据位的第 8 位 bit7,占用第 143~128 个时钟
    presult <= datain[7]^presult;
    idle <= 1'b1;
    cnt <= cnt + 8'd1;
end
8'd144: begin
    tx <= presult;        //发送奇偶校验位, 占用第 159~144 个时钟
    presult <= datain[0]^paritymode;
    idle <= 1'b1;
    cnt <= cnt + 8'd1;

```

```

end
8'd160: begin
    tx <= 1'b1;           //发送停止位， 占用第 160~167 个时钟
    idle <= 1'b1;
    cnt <= cnt + 8'd1;
end
8'd168: begin           //结束位 8 个时钟
    tx <= 1'b1;
    idle <= 1'b0;        //一帧资料发送结束
    cnt <= cnt + 8'd1;
end
default: begin
    cnt <= cnt + 8'd1;
end
endcase
end
else begin
    tx <= 1'b1;
    cnt <= 8'd0;
    idle <= 1'b0;
end
end
endmodule

```

4.PWM 信号生成模块;

```

module pwm(
    input clk,
    input rst_n,
    input [2:0]key,//前两位控制水平 , 后两位控制垂直
    output reg pwm_a,
    output reg pwm_b
);
// input clk_in1
// parameter Hori=2'b00;//水平 left midde right
//parameter Vert=2'd1;//垂直 normal down
//parameter Hori_normal=3'b00000;
parameter Hori_left=3'b001;
parameter Hori_right=3'b010;
// parameter Vert_normal=3'b100;
parameter Vert_down=3'b011;
parameter fuwei=3'b000;
// parameter bad=2'd2;
parameter CLK_DIV=6'd50;//50 分频为 1MHz 时钟
parameter MAX_NUM=20'd2_0000;
reg dri_clk;
reg [5:0]div_cnt;

```

```

reg [20:0]cnt;
//50 分频电路,得到 1MHz 时钟
always@(posedge clk or negedge rst_n)begin
    if(!rst_n)begin
        div_cnt<=6'b0;
        dri_clk<=1'b0;
    end
    else if(div_cnt==CLK_DIV/2-1'b1)begin
        div_cnt<=6'd0;
        dri_clk<=~dri_clk;
    end
    else begin
        div_cnt<=div_cnt+1'b1;
        dri_clk<=dri_clk;
    end
end

```

```

reg [15:0]shuipin_num;
reg [15:0]chuizhi_num;

always@(posedge clk or negedge rst_n)begin
    if(!rst_n)begin
        shuipin_num<=16'd2120;
        chuizhi_num<=16'd1550;
    end
    else begin
        case(key)
            /*fuwei:begin//000
            shuipin_num<=16'd2120;
            chuizhi_num<=16'd1550;
            end*/
            Hori_left:begin//001
            shuipin_num<=16'd1750;
            chuizhi_num<=chuizhi_num;
            end
            Hori_right:begin//010
            shuipin_num<=16'd2500;
            chuizhi_num<=chuizhi_num;
            end
            Vert_down:begin//100
            chuizhi_num<=16'd1950;
            shuipin_num<=shuipin_num;
            end
            default:begin
            shuipin_num<=16'd2120;
            end
        endcase
    end
end

```

```

        chuizhi_num<=16'd1550;
    end
endcase
end

end

//产生 50Hz, 20ms 的时钟
always@(posedge dri_clk or negedge rst_n)begin
    if(!rst_n)begin
        cnt<=21'd0;
    end
    else if(cnt==20'd2_0000-1'b1)
        cnt<=21'd0;
    else cnt<=cnt+1'b1;
end

//水平 2100 倾倒 175
always@(posedge dri_clk or negedge rst_n)begin
    if(!rst_n)begin
        pwm_a<=1'b0;
    end
    else if(cnt<shuipin_num)
        pwm_a<=1'b1;
    else pwm_a<=1'b0;
end

always@(posedge clk or negedge rst_n)begin
    if(!rst_n)begin
        pwm_b<=1'b0;
    end

    else if(cnt<chuizhi_num)
        pwm_b<=1'b1;
    else pwm_b<=1'b0;
end

endmodule

```

## 5. 消抖模块

```

module xiaodou(
    input          sys_clk,           //外部 50M 时钟
    input          sys_RST_N,         //外部复位信号, 低有效

```

```

input          key,           //外部按键输入
output         key_press      //按键数据有效信号

);

reg key_flag;
reg key_value;
//reg define
reg [31:0] delay_cnt;
reg      key_reg;
always @(posedge sys_clk or negedge sys_rst_n) begin
  if (!sys_rst_n) begin
    key_reg   <= 1'b1;
    delay_cnt <= 32'd0;
  end
  else begin
    key_reg <= key;
    if(key_reg != key)           //一旦检测到按键状态发生变化(有按键被按下或释放)
      delay_cnt <= 32'd1000000; //给延时计数器重新装载初始值（计数时间为 20ms）
    else if(key_reg == key) begin //在按键状态稳定时，计数器递减，开始 20ms 倒计时
      if(delay_cnt > 32'd0)
        delay_cnt <= delay_cnt - 1'b1;
      else
        delay_cnt <= delay_cnt;
    end
  end
end
end

always @(posedge sys_clk or negedge sys_rst_n) begin
  if (!sys_rst_n) begin
    key_flag  <= 1'b0;
    key_value <= 1'b1;
  end
  else begin
    if(delay_cnt == 32'd1) begin //当计数器递减到 1 时，说明按键稳定状态维持了 20ms
      key_flag  <= 1'b1;       //此时抖动过程结束，给出一个时钟周期的标志信号
      key_value <= key;        //并寄存此时按键的值
    end
    else begin
      key_flag  <= 1'b0;
      key_value <= key_value;
    end
  end
end
end

assign key_press=key_flag && key_value;

```

endmodule