电子技术课程设计

基于图像边缘检测的车牌数字识别系统

自 71 王思程 2017011534

自 71 尹小旭 2017011626

2019.7.13

一、选题背景

图像信息处理是现代电子信息发展中很重要的一个研究领域。图像边缘作为图像中的一个基本特征，能反应图像的形状、轮廓、纹理、位置等重要信息，因此其检测是许多图像处理任务的重要环节，边缘检测对于图像数据的压缩、传输和保存以及对于人工智能领域对于图像识别等算法的应用都有着很重要的作用。边缘检测的实质是采用某种算法提取出目标对象与背景的分界线，即图像中灰度发生急剧变化的区域边界。车牌识别广泛应用在停车场住宅区等场所，在车牌识别中，对摄像机所拍摄的车辆图像或者视频序列进行分析，通过边缘检测对图像简化，然后得到每一辆汽车唯一的车牌号码，从而完成识别过程

二、课题简介

本项目通过 OV7725 摄像头获取 RGB565 格式视频流输入 FPGA，用移位 IP 核获得 3*3 像素阵列，然后对视频流进行高斯滤波处理后用 Sobel 进行边缘检测，计算图像亮度函数的灰度近似值产生对应的灰度矢量，用 SDRAM 对处理后得到的视频流进行储存，将实时处理的黑白边缘图像显示在 VGA 显示器上。同时利用超声波传感器 HC-SR04 实时检测车辆到停车场道闸的距离，当距离达到设计好的值时，存下图像边缘检测后的视频的当前帧；同时，为了模拟实际状况，另外设计了以 LM324 运放和 MRF9011 三极管为核心的光控电路以实现亮度较暗时提供照明功能。

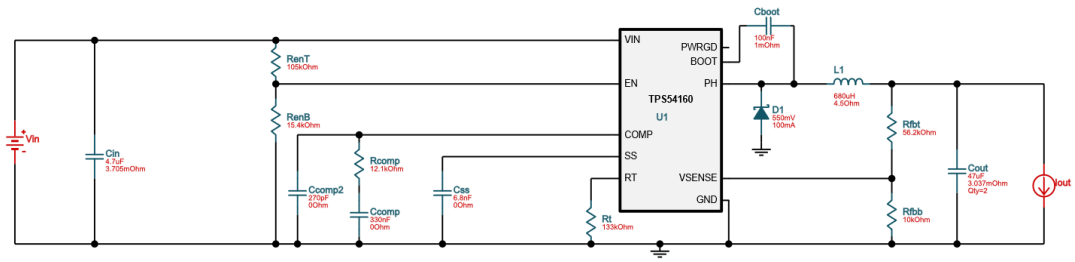
三、方案比较与选择

1.电源管理电路

本次实验中的电源管理是要实现 12V 转 5V 电压，经过 WEBENCH 设计有多个符合要求的电路，考虑到芯片的提供和电路的复杂程度，同时因为对于 PCB 的设计能力不足，因此主要选择的是可以利用转接板转为直插式芯片的电路，主要选择了三个电源管理电路：

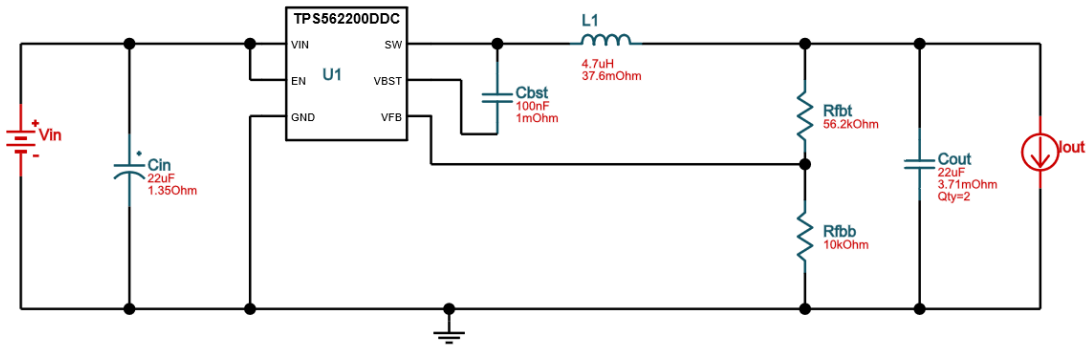
1.基于 TPS54160 的电源管理电路

该电路实现较为复杂，并且用到了肖特基二极管和电感等元器件，不容易调试电路，但是实验器件实验时都有提供，因此搭建比较方便。 WEBENCH 的电路设计如下：



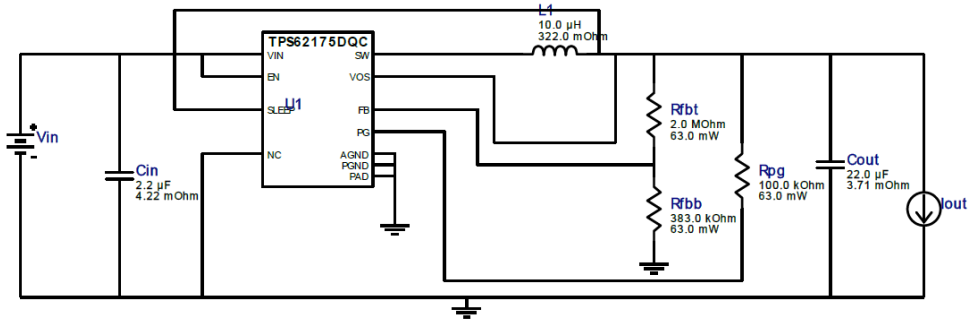
2.基于 TPS562200 的电源管理电路

该电源电路实现相对简单，使用的芯片也可以利用转接板得到直插式元件，元器件较少，方便调试电路。但是使用的芯片需要自己购买或者申请，所以提供的器件相对有限，所以要尽可能保证电路正确不损坏芯片。WEBENCH 电路设计如下：



3.基于 TPS62175 的电源管理电路

该电路的电路设计也比较简单，同时特性比较好，其设计如下：



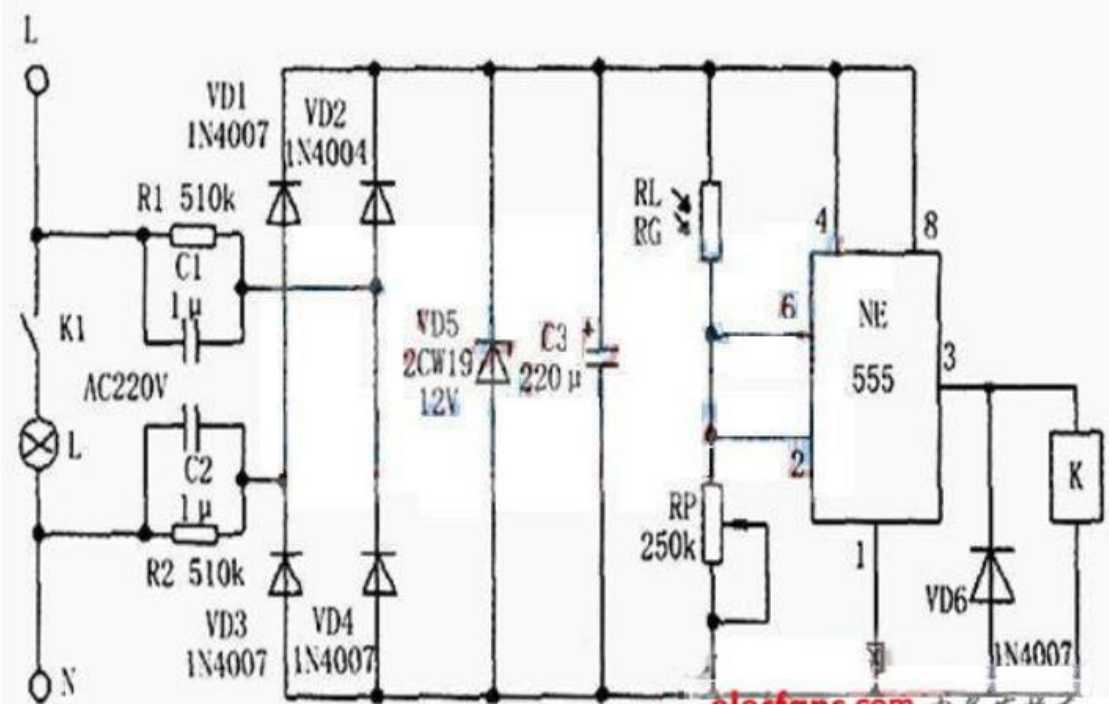
上述三种方案都各有优缺点，鉴于 TPS62175 芯片的封装比较特殊，不容易应用到电路中，最终在实验过程中对于其他两个电路都进行了搭建和调试，根据性能决定使用哪种方案。

2.光控电路

该电路是扩展功能，考虑到实际状况，为了在夜晚也能够识别车牌，设计了一个光控电路，当电路处在亮度较小的环境中时，led 灯亮，提供照明。该电路实现比较简单，实现的方案也较多。

对于电路核心器件的选择，常用的方案有利用 NE555、运放以及只利用简单的三极管构成的电路。对于检测器件的选择主要有光敏三极管和光敏电阻。

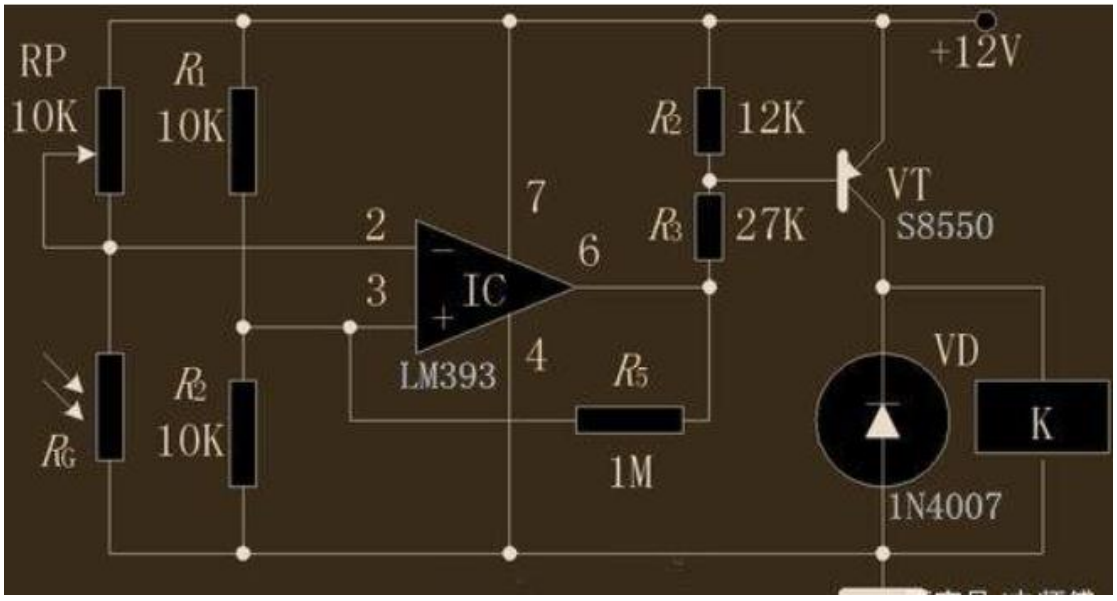
2.1 对于 NE555 电路，通过亮度变换引起光敏电阻阻值的变化，使得 3 脚的输出发生高低电平的转换从而实现了光控功能。



基于 NE555 的光控电路[1]

2.2 也可以利用运放构成电压比较器，光敏电阻的变化引起输入电压低于阈值电压，从而运放输出高电压，同时，输出端加入了三极管实现光控，在发射极搭配一定阻值的电阻也可以防止发光二极管烧坏。

下图所示电路与设计思路类似，利用三极管的导通和闭合实现对电路的控制。



基于运放的光控电路[2]

也可以直接使用三极管实现光控，通过光敏电阻的变化引起三极管的导通或截止，这种方法比较简单，不过不容易精确设置阈值，效果不够好。

---

[1] 简单的路灯自控电路图 http://www.elecfans.com/article/88/131/dg/2018/20180323651497_a.html

[2] 光敏电阻典型电路、自动光控调光电路、精密光控开关 http://baijiahao.baidu.com/s?id=1601794494970470345&wfr=spider&for=pc

考虑到电路的使用性以及实验元器件等问题，选择利用运放实现光控电路，因为电源管理电路没有设计-12V 电压，所以选用 LM324 的单极性运放，三级管选用实验室的 MRF9011L，使用供电电压为 3.3V 的发光二极管作为负载。

3.距离测量：

距离测量选择最为常用的超声波测距模块 HC~SR04，工作电压为 5V。使用最典型的测距方法，FPGA 对传感器发出触发信号，传感器接收到之后循环发出 8 个 40kHz 的脉冲，脉冲遇到物体后返回产生了回响信号，FPGA 的测距模块通过触发信号和回响信号的时间差计算得到物体距传感器的距离从而实现测距。

4.FPGA 图像边缘检测：

4.1 滤波

4.1.1 均值滤波：图片中一个方块区域（一般为 3*3）内，中心点的像素为全部点像素值的平均值。均值滤波就是对于整张图片进行以上操作。

优点：效率高，思路简单

缺点：（1）不保护图像细节，去噪同时破坏了图片细节，有使图像模糊的趋势。

（2）不能很好地去除椒盐噪声。

4.1.2 中值滤波：是一种非线性滤波，在图像中去 3*3 的矩阵，里面有 9 个像素点，我们将 9 个像素进行排序，最后将这个矩阵的中心点赋值为这九个像素的中值。中值滤波就是对于整张图片进行以上操作。

优点：（1）处理脉冲噪声和椒盐噪声的效果很好

（2）能有效保护图像的边缘信息

缺点：（1）简单地去实现中值滤波时间复杂度较高，需要有改进算法实现

（2）对高斯噪声的抑制效果不好

4.1.3 高斯滤波：是一种平滑线性滤波器，就是用带权重的滤波淹没确定领域内像素的平均灰度值去代替图像每个像素点的值，利用二维高斯函数的分布方式对图像进行平滑，是最常用的一种滤波算法。

优点：（1）二维高斯函数是旋转对称的，在各个方向上平滑程度相同，不会改变原图像的边缘走向。

（2）高斯函数是单值函数，高斯卷积核的锚点为极值，在所有方向上单调递减，锚点像素不会受到距离锚点较远像素的过大影响，保证了特征点和边缘的特性。

（3）在频域上，滤波过程不会被高频信号污染。

4.2.边缘检测

4.2.1 Sobel 算子：是一种离散性差分算子，用来运算图像亮度函数的灰度近似值。在图像的何一点使用此算子，将会产生对应的灰度矢量。

该算子包含两组 3x3 的矩阵，分别为横向及纵向，将之与图像作平面卷积，即可分别得出横向及纵向的亮度差分近似值。如果以 A 代表原始图像，Gx 及 Gy 分别代表经横向及纵向边缘检测的图像，其公式如下:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * A$$

图像的每一个像素的横向及纵向梯度近似值可用以下的公式结合，来计算梯度的大小：

$$G = \sqrt{G_x^2 + G_y^2}$$

可用以下公式计算梯度方向

$$\theta = arctan\left(\frac{G_y}{G_x}\right)$$

优点：（1）对于像素的位置的影响做了加权，可以降低边缘模糊程度

（2）Sobel 算子是滤波算子的形式，用于提取边缘，可以利用快速卷积函数， 简单有效

缺点：（1）没有基于图像灰度进行处理，所以并没有将图像主体与背景严格地区分

（2）没有严格模拟人的视觉生理特征，提取的图像轮廓有时并不令人满意

4.2.2 LoG 算子：先对图像做高斯滤波，抑制噪声，然后再求其拉普拉斯（Laplacian）二阶导数。最后，通过检测滤波结果的零交叉可以获得图像或物体的边缘。因而，也被业界简称为 Laplacian-of-Gaussian (LoG)算子。

常见的 3*3 和 5*5 整数模板为：

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \quad \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

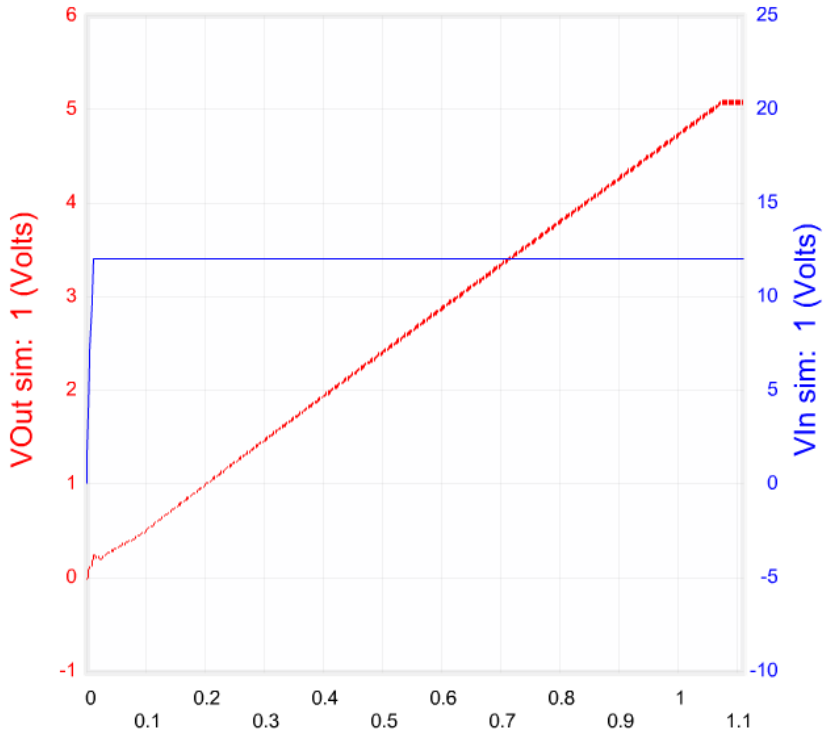
由于 Sobel 算子较于容易实现且有较好效果，决定使用 Sobel 算子。

三、基于 WEBENCH 的电源电路仿真：

因为设计中最终采用的是基于 TPS562200 的电源管理电路，故主要对该电路进行 WEBENCH 仿真。部分仿真结果如下：

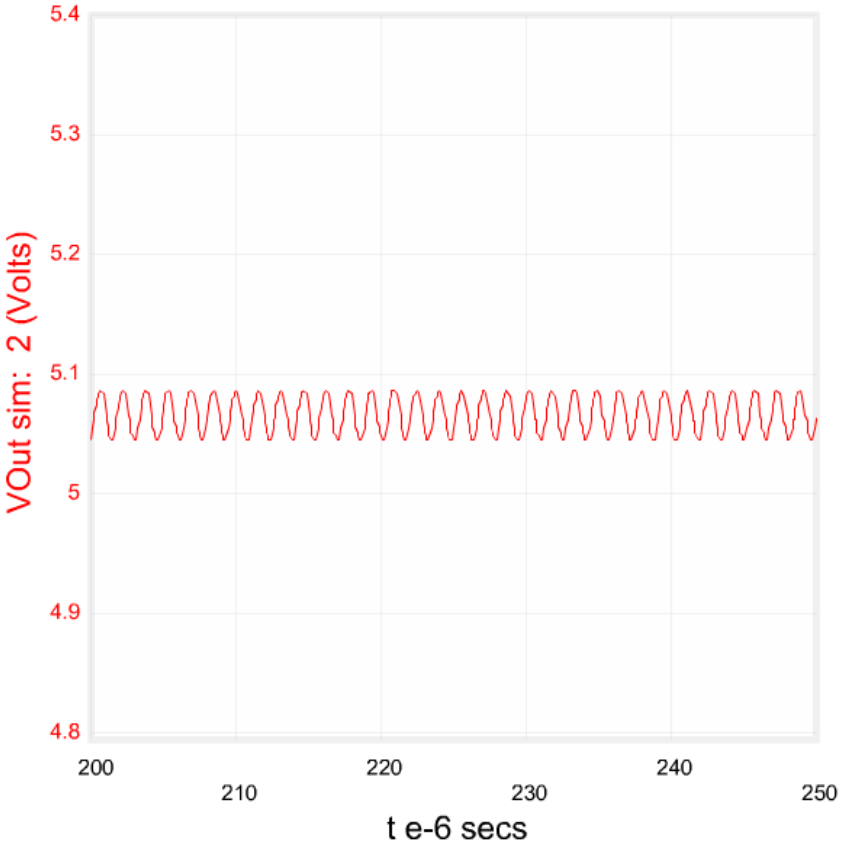
1.启动仿真

下图是电路的启动仿真，可以看到在电路输入 12V 的电压后，输出能在 1ms 后到达所需要的 5v 电压，因为电路对于启动速度要求不大，所以完全可以接受：

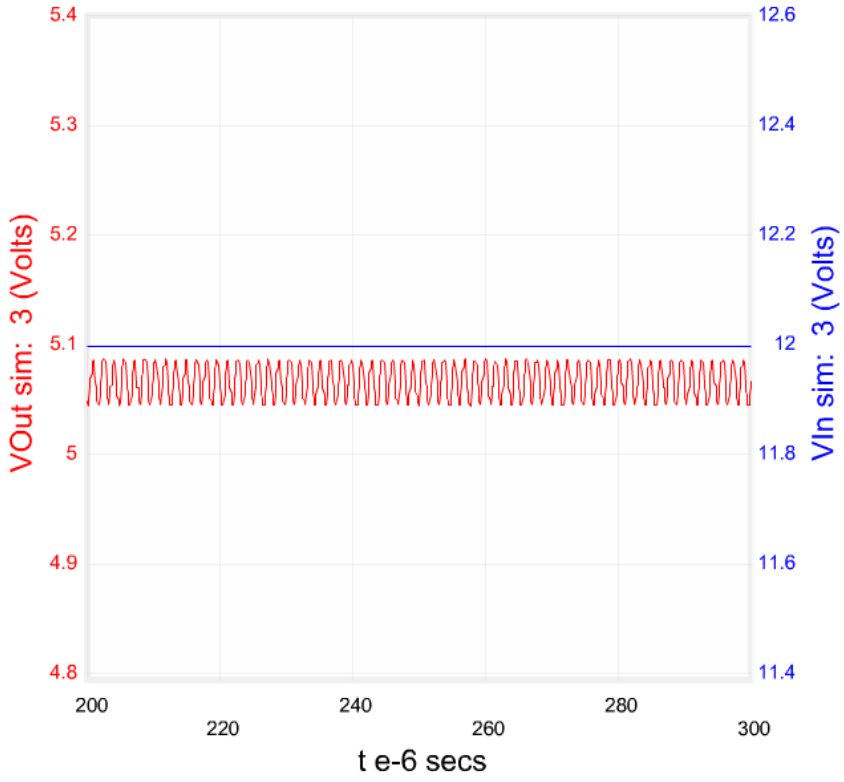


2.稳态仿真

下图是电路稳定工作时的输出电压，可以看到，正常情况下输出电压稳定在 5.05V~5.08V 左右，纹波为 0.03V，比较稳定：

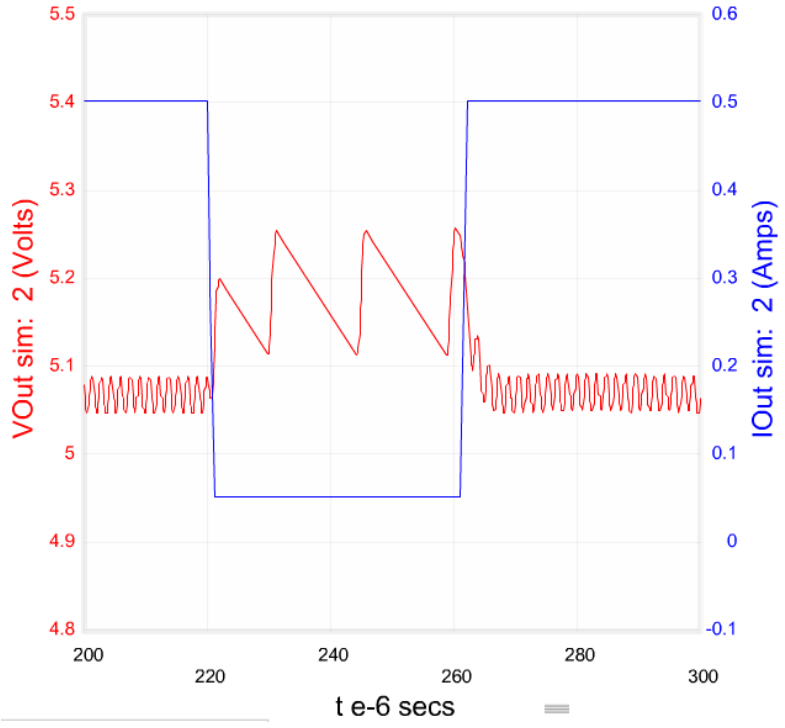


3.输入瞬态仿真：



4.负载瞬态仿真

下图是电路的负载瞬态仿真波形，可以看到在负载发生瞬变时电路输出电压最大可以跳到 5.25V 左右，相比之下波动比较小，电路的工作输出电压比较稳定。



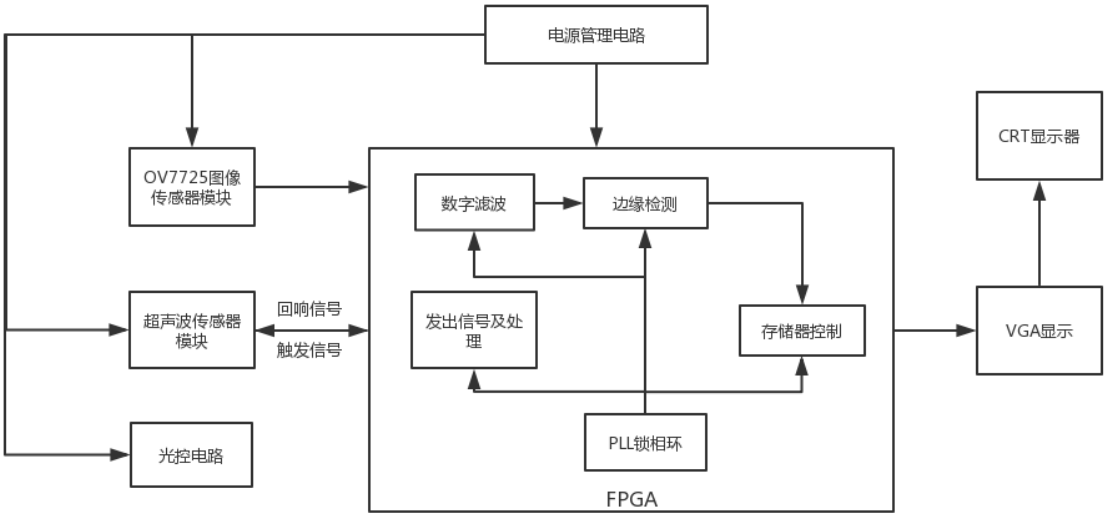
5.热仿真（针对设计好的 pcb）

可以看到电路工作时的芯片的最高温度不会超过 39℃，电路设计比较合理，用万能板焊接时也应当注意电路的热效应防止芯片被烧坏。

Vin	Iout	Tamb	Cu We	Highest Tei	Max Temp	Power Dissipation
12V	0.5A	30°C	2 OZ	39.0°C	Cin: 33.1°C Cout: 32.6°C IC - Die: 39.0°C IC - Top: 38.9°C L1: 32.6°C PCB - Bottom: 33.7°C PCB - Top: 38.8°C	Cin: 157uW Cout: 227uW IC - Die: 0.14W IC - Top: 0.14W L1: 0.01W

四、电路框图及各模块功能简介：

电路的整体设计框图如下:



电源管理电路对图像传感器模块、超声波传感器模块、光控电路模块和 FPGA 供电，每个模块与 FPGA 之间进行连接，FPGA 对于收到的数据进行滤波、存储控制等操作，将处理好的结果通过 VGA 接口在 CRT 显示器上显示。

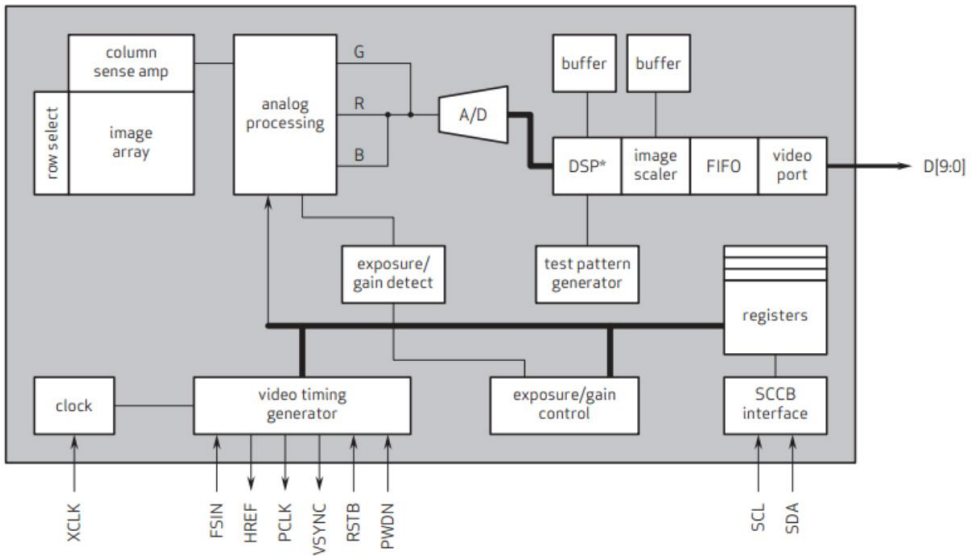
系统的核心功能是对摄像头拍摄到的有车牌的头像在显示屏上实时定位同时对车牌的每个字符进行分割，通过之前的边缘检测处理，最终得到的图像既定位了车牌还实现了二值化处理，这样可以很方便地实现车牌具体字符的识别（字符识别部分最终未能实现）；同时系统考虑到环境因素，为方便摄像头读取信息，采用光控模块实现光控照明；而超声波模块通过测距可以得出检测识别车牌的比较好的距离，并在可能的情况下发出声响提示识别完成。

以下给出 FPGA 外各模块的简介。

1.OV7725 图像传感器模块

OV7725 是一种图像传感器，它的感光阵列可以达到 640*800，其内部集成了很多图像处理的功能，并且支持多种不同分辨率图像的输出以及多种不同的数据像素格式，应用比较广泛。

这里是 OV7725 的功能框图，可以看到，感光阵列（image array）在时钟驱动下进行图像的采样，而时序发生器对模拟量进行一定的算法处理，再经过 AD 转换器后成功转换为数字信号，最终可以输出所配置格式的 10 位视频数据流。需要注意的是配置寄存器使用的是 SCCB 接口，其协议兼容 IIC 协议。



2.超声波传感器模块

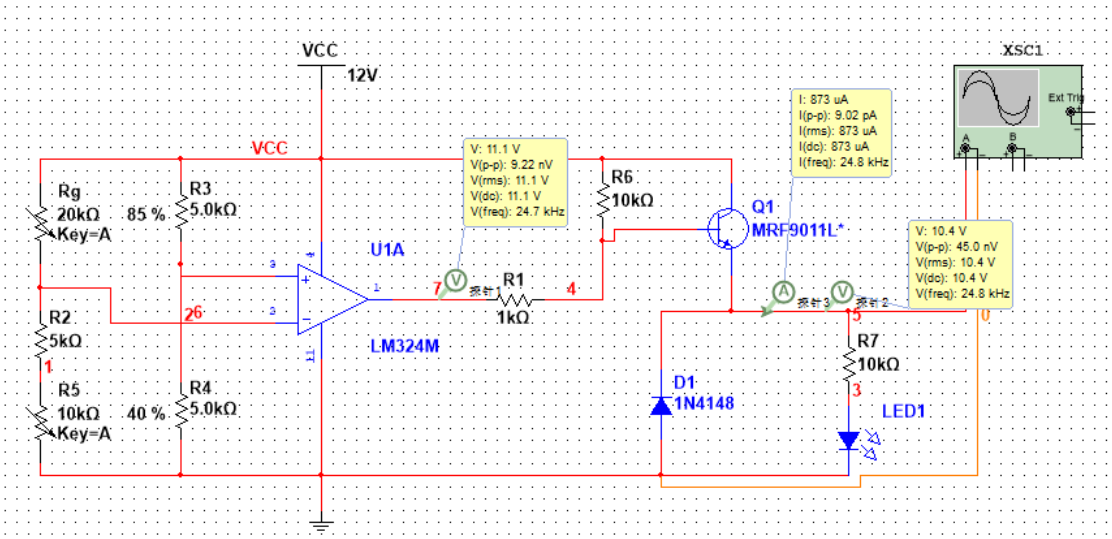
超声波传感器模块使用的是 HC-SR04，其内部结构主要由扬声器、压电晶片、共振板、运算放大器等构成，需要 5V 电压驱动，工作电流 15mA，trig 引脚接受 FPGA 的信号，echo 引脚向 FPGA 发送物体反射回来的信号。

3.电源管理

在实际设计中发现基于 TPS54160 的管理电路结构比较复杂，波形很不稳定不容易调试，最终选择 TSP56220 作为电源管理电路的芯片，其电路设计图以及仿真效果在上面已经给出，可以调节电阻 Rfbb 和 Rfbt 来调整电路的输出电压，通过改变电容的值可以调节电路的滤波效果提升稳定性。

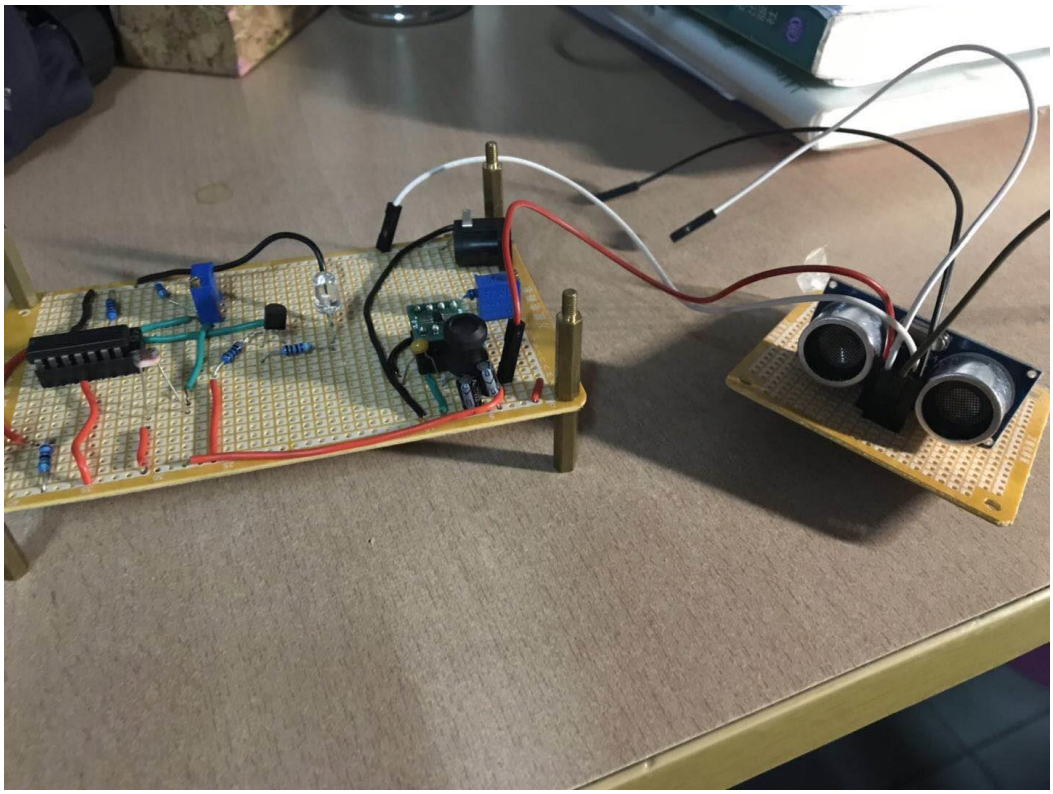
4.光控电路

根据方案比较与选择，选择以 LM324 运放和三极管 MRF9011L 为主的光控照明电路，光敏元件选择 10-20KΩ的光敏电阻，发光器件选择工作电压为 3.3V 的发光二极管，对于选择的方案稍作修改可以得到如下的电路设计:



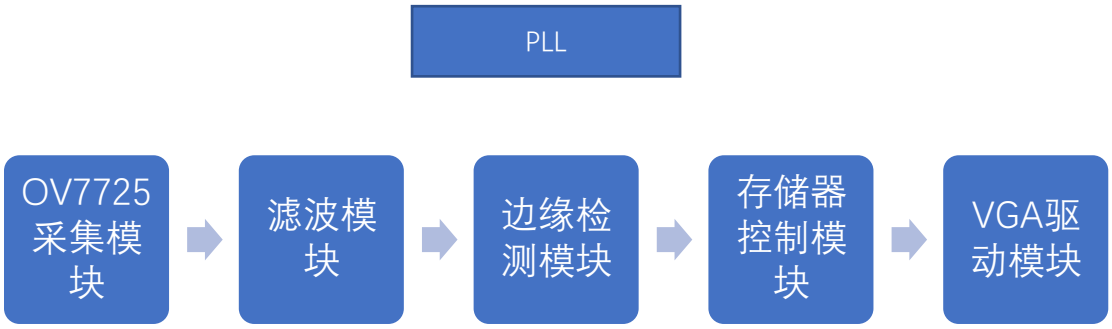
图中 Rg 为光敏电阻，受光照时电阻为 10~20KΩ，无光照时电阻为几百千欧，LM324 和前面的四个电阻构成了一个电压比较器，可以通过调节 R3 和 R4 的比值来改变电路的阈值电压，在调节合适的情况下，有光照的情况下（白天），由于 Rg 电阻小，运放反相输入端电压高于阈值电压，比较器输出接近 0V 的低电压，三极管无法导通，因此发光二极管不亮；在黑暗条件下，反相端输入电压低于阈值电压，运放输出高电压，使得三极管导通，因此二极管亮，从而实现了光控照明。同时，可以通过调节 R5 来调整电路对光的敏感程度，事实引入反馈则电路对于临界条件的处理会更加优秀，但本电路要求不高，故不再设计反馈调节。

最终搭建好的光控电路和电源管理电路如下：



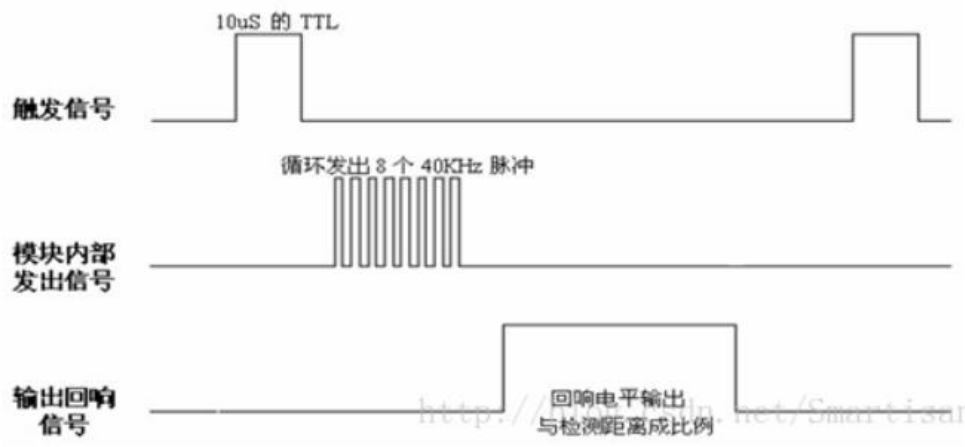
5.FPGA 模块：

核心控制是对图像信息处理的数字系统，详细解释将在下一部分给出。

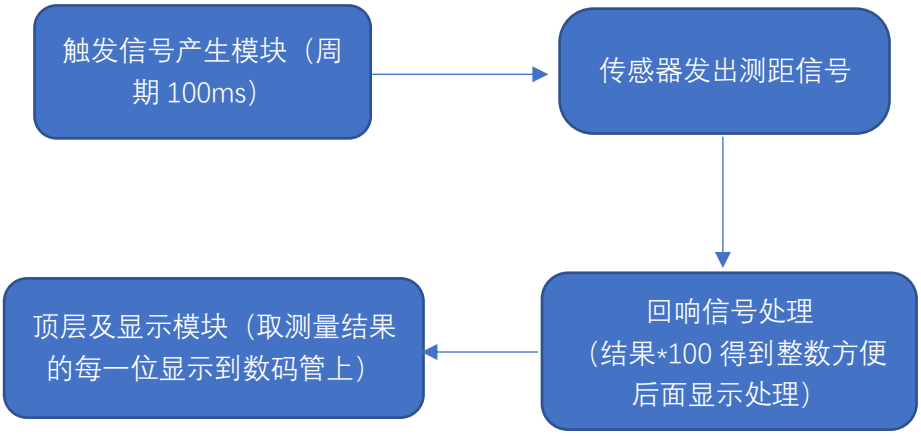


五、数字系统流程图

1.超声波模块设计：



超声波模块的工作原理：FPGA 板产生有 10us 脉冲的周期信号，为了保证测量的有效性，这个周期一般为几百 ms 或 1s，超声波模块接收到这个触发信号后向其检测的方向发出 8 个 40KHz 脉冲，脉冲遇到物体后被反弹，模块接收到反弹信号后处理得到回响信号发送给 FPGA，这个回响信号的高电平时间与检测的距离成正比，因此可以计算出物体到模块的距离。
可以设计如下的模块结构：（本模块采用的时钟信号为 50MHz）



1.1 触发信号模块比较容易，不再解释；

1.2 回响信号处理模块：

模块的声明和中间变量的设置如下：

模块设计采用了状态机结构，三个状态为初始状态、开始测距状态、结束测距状态，这三个状态的转变主要是靠回响信号的上升沿和下降沿，即 up 和 down，利用回响信号和比其早一个周期的信号来组合判断信号的上升沿和下降沿；count 保存计数结果，一个周期 20ns，而声速为 340m/s，距离为计数时间*声速/2，即得到

$$\text{distance} = \frac{\text{count} \times 20 \times 10^{-9} \times 340}{2} m$$

考虑到小数部分不好处理，将结果化为 cm 同时*100，最终得到

$$\text{distance} = \frac{\text{count} \times 20 \times 10^{-5} \times 340}{2} = count \times 0.034 = 2 \times count \times 0.017 = \frac{2 \times count}{58}$$

即代码中的结果。

```
module feedback(
    input clk,
    input echo,
    input rst_n,

    output [19:0] distance
);

parameter S0=2'b00,S1=2'b01,S2=2'b10;    //初始状态，开始测距，结束测距三个状态
reg [1:0] curr_st;
reg [1:0] next_st;

reg [20:0] cnt_time;     //对反射回的高电平时间计周期数
reg [20:0] count;        //保存计数结果
reg echo_1,echo_2;       //典型的利用各一个周期的信号来检测上升沿和下降沿方法

wire up,down;            //上升沿和下降沿信号
assign up=echo_1&(~echo_2);
assign down=(~echo_1)&echo_2;
assign distance=2*count/58;  //将ns换算得到以cm为单位同时*100倍的距离(方便得到小数部分)
```

1.3 显示模块：

模块使用到了七段译码器的例化，七段译码器程序很容易写，对应每个数字的 abcdefg 的值即可，因为每个数字需要 4 位的值来表示，num 就是需要放到数码管上某一位显示的值，程序也利用了状态机的思想，只是没有显示表达出来。

```
module display(
    input clk,
    input rst_n,
    input [19:0] data,

    output reg [3:0]sel,    //数码管的亮灭
    output [6:0]led,  //a,b,c,e,f,g
    output reg dot    //小数点
);

reg [15:0] cnt;
reg [3:0] num;
reg [1:0] sta;    //当前亮数码管的位置

sevenreg u_sevenreg(      //例化七段译码器
.num(num),
.led(led)
);
```

在 always 语句中根据当前数码管该亮的位置决定某一位数字，因为测试时在原 FPGA 上，只有四个数码管，故省略了最后一位，对 dot 赋值确定小数点的位置，同时每位数码管的扫描周期为 1ms，比较容易看清。

```
   begin
      case(sta)
      2'b00:
      begin
         num<=data[7:4];
         dot<=0;
         sel<=4'b0001;
      end
      2'b01:
      begin
         num<=data[11:8];
         dot<=1;
         sel<=4'b0010;
      end
      2'b10:
      begin
         num<=data[15:12];
         dot<=0;
         sel<=4'b0100;
      end
      2'b11:
      begin
         num<=data[19:16];
         dot<=0;
         sel<=4'b1000;
      end
      endcase
   end
```

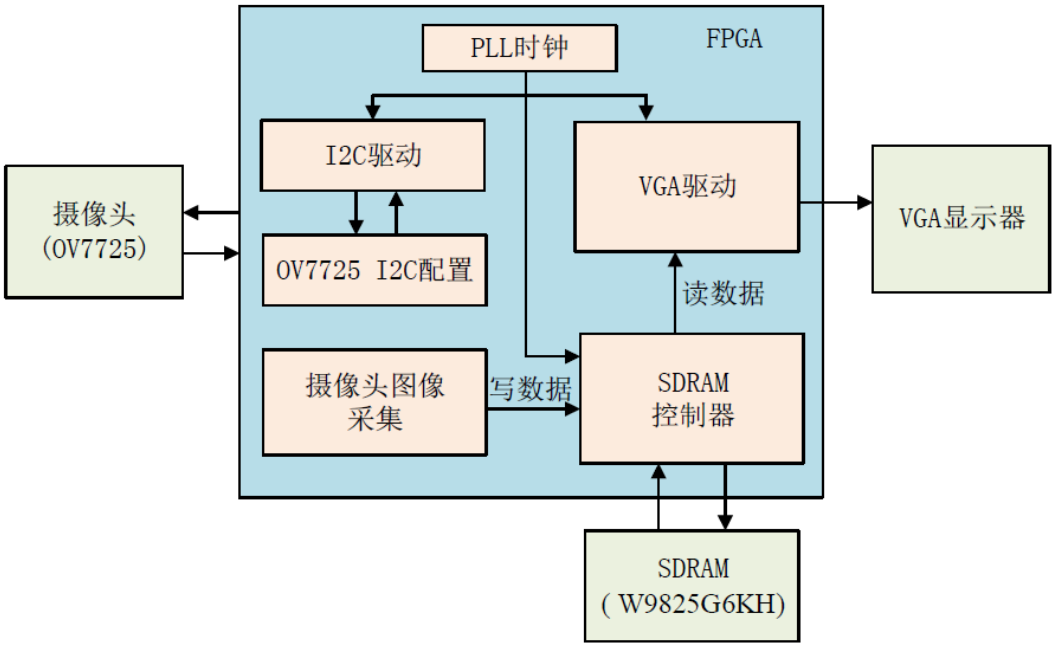
超声波模块封装好的文件：

```
module top_data(
    input clk,
    input rst_n,
    input echo,

    output [19:0] data,
    output trig
);

wire [19:0] distance;

trigger u_trigger(
    .clk(clk),
    .rst_n(rst_n),
    .trig(trig)
);

feedback u_feedback(
    .clk(clk),
    .echo(echo),
    .rst_n(rst_n),
    .distance(distance)
);

assign data[3:0] = distance%10; //百分位
assign data[7:4] = distance/10%10; //十分位
assign data[11:8] = distance/100%10; //个位
assign data[15:12] = distance/1000%10; //十位
assign data[19:16] = distance/10000;    //百位

endmodule
```

使用时利用这个模块就可以实现对超声波的控制了。

2.摄像读写与显示模块

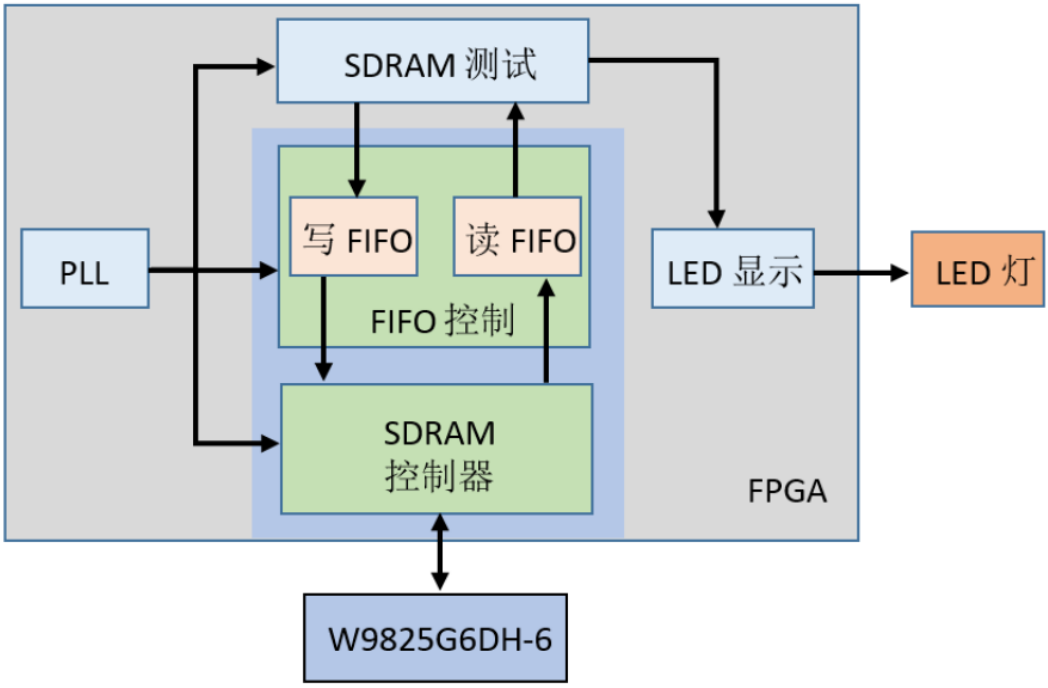
摄像读写与显示也是非常重要的一个部分。其总体的控制流程如下:



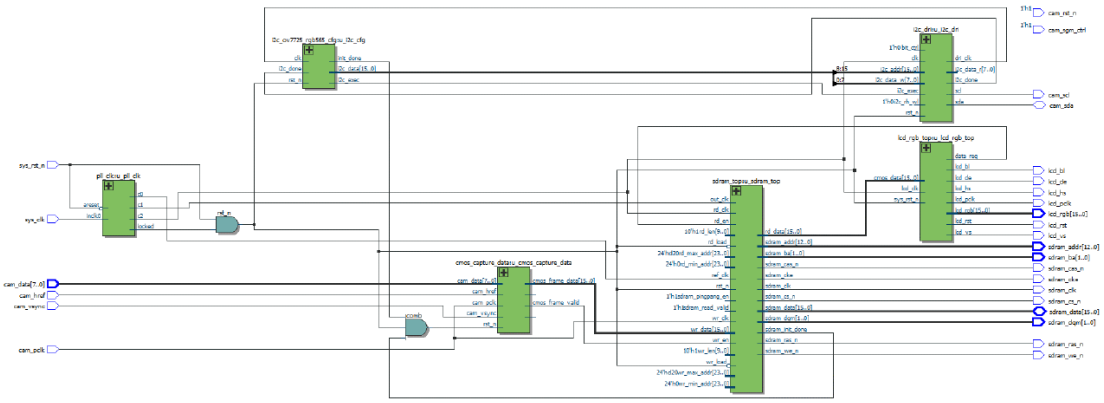
采用 OV7725 图像传感器实现对图像的实时采集，由上图可知，PLL 时钟模块为 VGA 驱动模块、SDRAM 读写控制模块以及 I2C 驱动模块提供驱动时钟，I2C 配置模块和 I2C 驱动模

块控制着传感器初始化的开始与结束，传感器初始化完成后图像采集模块将采集到的数据写入 SDRAM 读写控制模块，VGA 驱动模块从 SDRAM 控制模块中读出数据，完成了数据的采集、缓存与显示。需要注意的是图像数据采集模块是在 SDRAM 和传感器都初始化完成之后才开始输出数据的，避免了在 SDRAM 初始化过程中向里面写入数据。

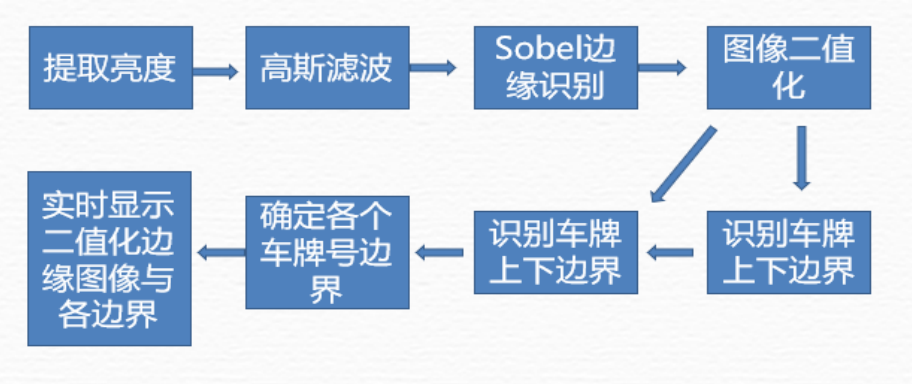
但是需要注意的是，SDRAM 虽然存储空间大，便于读写，但是其复杂的控制时序对用户不够友好，所以通常的做法是将其做一定的处理，封装成用户可以当做 FIFO 来读写的模块，这样使用起来就很方便，其模块封装如下：



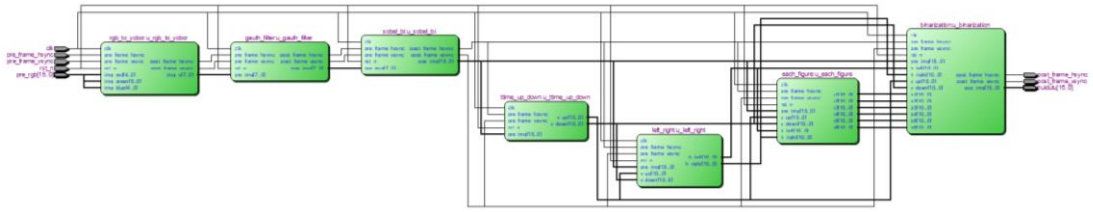
由于这一部分的代码实现相当复杂，原理也比较晦涩，同时又有现成的例程可以参考，所以设计时直接使用了参考例程，所以不在详述代码部分，该模块的顶层图如下：



3..图像处理核心模块的设计



图像处理顶层模块



图像处理模块顶层连线图

图像处理部分代码都由自己独立完成。图像处理模块接在 VGA 驱动模块后，处理后直接连到 VGA 显示器。第一个是流程图，首先是对 rgb565 格式的视频进行亮度提取，然后进行高斯滤波，Sobel 边缘检测，检测边缘后二值化，边缘为白色，非边缘部分为黑色，利用上上下边界以进一步检测出车牌的上下左右边界，进而确定各个车牌号的边界 ，最终实时显示二值化后的边缘图像和车牌号的各个边界。后面讲详细讲解各个部分。

代码:

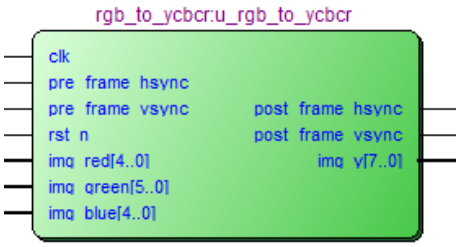
```
module chulidianlu(
 //module clock
    input               clk             ,   // 模块驱动时钟
    input               rst_n           ,   // 复位信号

    //图像处理前的数据接口
    input               pre_frame_vsync ,   // vsync信号
    input               pre_frame_hsync ,   // hsync信号

    input       [15:0]  pre_rgb,

    //图像处理后的数据接口
    output              post_frame_vsync,   // vsync信号
    output              post_frame_hsync,   // hsync信号
    output      [15:0]    huidutu             // 输出最终图像Y数据,并扩展到16位
    );


wire  [ 3:0]    outnum1;
wire  [ 3:0]    outnum2;
wire  [ 3:0]    outnum3;
wire  [ 3:0]    outnum4;
wire  [ 3:0]    outnum5;

wire [10:0]   h_left;
wire [10:0]   h_right;
wire [10:0]   v_up;
wire [10:0]   v_down;


wire          frame_hsync1;
wire          frame_vsync1;
wire [ 7:0]   img1;

wire          frame_hsync2;
wire          frame_vsync2;
wire [ 7:0]   img2;

wire          frame_hsync3;
wire          frame_vsync3;
wire [15:0]   img3;
```

```
  wire [10:0]   b1;
  wire [10:0]   b2;
  wire [10:0]   b3;
  wire [10:0]   b4;
  wire [10:0]   b5;
  wire [10:0]   b6;

  //assign huidutu= ( img2>8'b1001_0000 ? 16'b1111_1111_1111_1111:16'b0000000000000000  );
  //assign huidutu = 16'b1111111111100000;


  //例化模块1，转YCBCR
rgb_to_ycbcr u_rgb_to_ycbcr(
    .clk                (clk),
    .rst_n              (rst_n),
    .pre_frame_vsync    (pre_frame_vsync),
    .pre_frame_hsync    (pre_frame_hsync),
    .img_red            (pre_rgb[15:11] ),
    .img_green          (pre_rgb[10:5 ] ),
    .img_blue           (pre_rgb[ 4:0 ] ),
    .post_frame_vsync   (frame_vsync1),
    .post_frame_hsync   (frame_hsync1),
    .img_y              (img1)
    );
   //例化模块2，高斯滤波
gauth_filter u_gauth_filter(
    .clk                (clk),
    .pre_frame_vsync    (frame_vsync1),
    .pre_frame_hsync    (frame_hsync1),
    .pre_img            (img1),
    .rst_n              (rst_n),
    .post_frame_vsync   (frame_vsync2),
    .post_frame_hsync   (frame_hsync2),
    .pos_img            (img2)
    );
  //例化模块3，sobel后二值化
sobel_bi u_sobel_bi(
    .clk                (clk),
    .pre_frame_vsync    (frame_vsync2),
    .pre_frame_hsync    (frame_hsync2),
    .pre_img            (img2),
    .rst_n              (rst_n),
    .post_frame_vsync   (frame_vsync3),
```

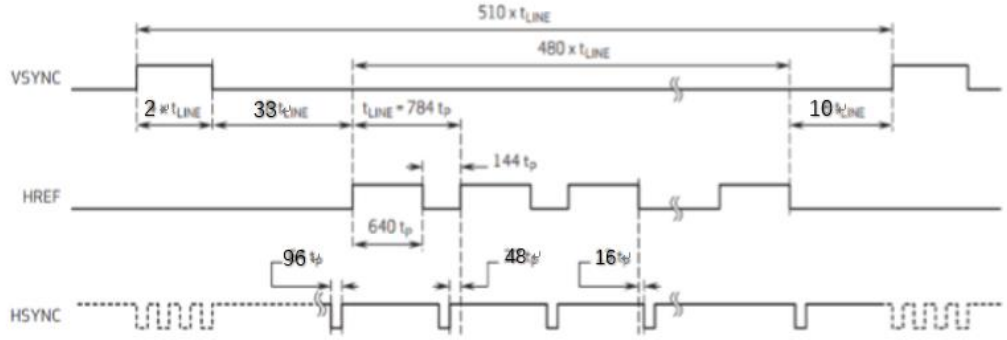
```
  //例化模块4，根据边缘检测后的图像和车牌边界信息只显示车牌
binarization u_binarization(
    .clk                (clk),
    .rst_n              (rst_n),
    .pre_frame_vsync    (frame_vsync3),
    .pre_frame_hsync    (frame_hsync3),
    .pre_img            (img3),
    .post_frame_vsync   (post_frame_vsync),
    .post_frame_hsync   (post_frame_hsync),
    .pos_img            (huidutu),
    .h_left             (h_left),
    .h_right            (h_right),
    .v_up               (v_up),
    .v_down             (v_down),
    .b1                 (b1),
    .b2                 (b2),
    .b3                 (b3),
    .b4                 (b4),
    .b5                 (b5),
    .b6                 (b6)
    );

/*//例化模块5，根据蓝色找车牌左右边界//用不了，摄像头像素偏白，发白的淡蓝和白色分不开。
blue_left_right u_blue_left_right(
    .clk                (clk),
    .rst_n              (rst_n),
    .pre_frame_vsync    (pre_frame_vsync),
    .pre_frame_hsync    (pre_frame_hsync),
    .pre_img            (pre_rgb),
    .h_left             (h_left),
    .h_right            (h_right)
    );
*/

   //例化模块6，根据交叉次数找车牌上下边界
ttime_up_down u_ttime_up_down(
    .clk                (clk),
    .rst_n              (rst_n),
    .pre_frame_vsync    (frame_vsync3),
    .pre_frame_hsync    (frame_hsync3),
    .pre_img            (img3),
    .v_up               (v_up),
    .v_down             (v_down)
    );
```

```
    //例化模块7，根据车牌长横线边界及已有车牌上下边界信息定出左右边界。但是要求水平程度好，不过现实中容易实现车牌相对摄像头很水平，也不是问题。
left_right u_left_right(
    .clk                (clk),
    .rst_n              (rst_n),
    .pre_frame_hsync    (frame_hsync3),
    .pre_frame_vsync    (frame_vsync3),
    .pre_img            (img3),
    .v_up               (v_up),
    .v_down             (v_down),
    .h_left             (h_left),
    .h_right            (h_right)
    );

    //例化模块8，对每个数字确定边界
each_figure u_each_figure(
    .clk                (clk),
    .rst_n              (rst_n),
    .pre_frame_vsync    (frame_vsync3),
    .pre_frame_hsync    (frame_hsync3),
    .pre_img            (img3),
    .v_up               (v_up),
    .v_down             (v_down),
    .h_left             (h_left),
    .h_right            (h_right),
    .b1                 (b1),
    .b2                 (b2),
    .b3                 (b3),
    .b4                 (b4),
    .b5                 (b5),
    .b6                 (b6)
    );

/*
    //例化模块9，识别每个数字     //白色占比法寄存器不够，特征识别交线的方法波动大用不了
num_figure2 u_num_figure2(
    .clk                (clk),
    .rst_n              (rst_n),
    .pre_frame_vsync    (frame_vsync3),
    .pre_frame_hsync    (frame_hsync3),
    .pre_img            (img3),
    .v_up               (v_up),
    .v_down             (v_down),
    .h_left             (h_left),
     .h_right            (h_right),
     .b1                 (b1),
     .b2                 (b2),
     .b3                 (b3),
     .b4                 (b4),
     .b5                 (b5),
     .b6                 (b6),
     .outnum1            (outnum1),
     .outnum2            (outnum2),
     .outnum3            (outnum3),
     .outnum4            (outnum4),
     .outnum5            (outnum5)
     );

    //例化模块10，显示识别的5个数字
shumaguan u_shumaguan(
    .clk                (clk),
    .rst_n              (rst_n),
    .num1               (outnum1),
    .num2               (outnum2),
    .num3               (outnum3),
    .num4               (outnum4),
    .num5               (outnum5),
    .sel                (sel),
    .led                (led)
    );
*/

endmodule
```

3.1 提取亮度模块



输入：Hsync 为行同步信号，Vsync 为场同步信号，rgb565 的像素信息
VGA 显示选 640*480 分辨率 60FPS 的参数来显示，在这种情况下行场同步信号的时序图如下



行同步信号拉低后经过一定延迟后，连续传输一行 640 个像素。
场同步信号拉低的时间内，传输 480 行。
一个场同步信号的周期就是完整传输一帧的时间。
由锁相环分得 VGA 驱动时钟为 25MHz，即$t_p = 4 \times 10^{-8}$秒。
一个行同步信号周期$t_{LINE} = 96t_p + 48t_p + 640t_p + 16t_p = 800t_p = 3.2 \times 10^{-5}$秒
一个场同步信号周期=$2t_{LINE} + 33t_{LINE} + 480t_{LINE} + 10t_{LINE}$ =$525t_{LINE} = 0.0168$秒 ，对应 60FPS 显示。
由于 FPGA 只能一个时钟读到一个像素，给图像处理带来很大挑战！

处理：为提取亮度需要将 RGB565 信号转化为 YCbCr 模式。其中 Y 是指亮度分量，Cb 是指蓝色色度分量，Cr 指红色色度分量。
转换计算公式如下：
Y = 0.299R +0.587G + 0.114B
Cb = 0.568(B-Y) + 128 = -0.172R-0.339G + 0.511B + 128
CR = 0.713(R-Y) + 128 = 0.511R-0.428G -0.083B + 128
为计算时节省 FPGA 寄存器资源，将其化为
Y = (77 *R + 150*G + 29 *B)>>8
Cb = (-43*R - 85 *G + 128*B + 32768)>>8
Cr = (128*R - 107*G - 21 *B + 32768)>>8

由于转化的计算需要 3 个时钟周期，行场同步信号也需延迟相同周期

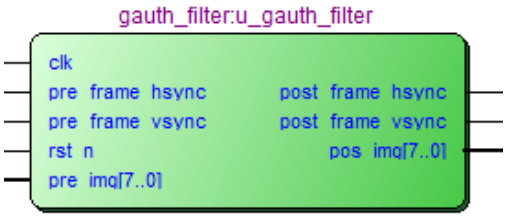
代码：

```v
module rgb_to_ycbcr(
    //module clock
    input               clk             ,   // 模块驱动时钟
    input               rst_n           ,   // 复位信号

    //图像处理前的数据接口
    input               pre_frame_vsync ,   // vsync信号
    input               pre_frame_hsync ,   // hsync信号
    input       [4:0]   img_red         ,   // 输入图像数据R
    input       [5:0]   img_green       ,   // 输入图像数据G
    input       [4:0]   img_blue        ,   // 输入图像数据B

    //图像处理后的数据接口
    output              post_frame_vsync,   // vsync信号
    output              post_frame_hsync,   // hsync信号
    output      [7:0]   img_y              // 输出图像Y数据

);

//reg define
reg  [15:0]   rgb_r_m0, rgb_r_m1, rgb_r_m2;
reg  [15:0]   rgb_g_m0, rgb_g_m1, rgb_g_m2;
reg  [15:0]   rgb_b_m0, rgb_b_m1, rgb_b_m2;
reg  [15:0]   img_y0 ;
reg  [15:0]   img_cb0;
reg  [15:0]   img_cr0;
reg  [ 7:0]   img_y1 ;
reg  [ 7:0]   img_cb1;
reg  [ 7:0]   img_cr1;
reg  [ 2:0]   pre_frame_vsync_d;
reg  [ 2:0]   pre_frame_hsync_d;


//wire define
wire [ 7:0]   rgb888_r;
wire [ 7:0]   rgb888_g;
wire [ 7:0]   rgb888_b;


//RGB565 to RGB 888
assign rgb888_r         = {img_red  , img_red[4:2]  };
assign rgb888_g         = {img_green, img_green[5:4]};
assign rgb888_b         = {img_blue , img_blue[4:2] };
//行场同步信号延迟用
assign post_frame_vsync = pre_frame_vsync_d[2]      ;
assign post_frame_hsync = pre_frame_hsync_d[2]      ;

assign img_y            = post_frame_hsync ? img_y1 : 8'd0;
```

```v

 //乘
always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        rgb_r_m0 <= 16'd0;
        rgb_r_m1 <= 16'd0;
        rgb_r_m2 <= 16'd0;
        rgb_g_m0 <= 16'd0;
        rgb_g_m1 <= 16'd0;
        rgb_g_m2 <= 16'd0;
        rgb_b_m0 <= 16'd0;
        rgb_b_m1 <= 16'd0;
        rgb_b_m2 <= 16'd0;
    end
    else begin
        rgb_r_m0 <= rgb888_r * 8'd77 ;
        rgb_r_m1 <= rgb888_r * 8'd43 ;
        rgb_r_m2 <= rgb888_r << 3'd7 ;
        rgb_g_m0 <= rgb888_g * 8'd150;
        rgb_g_m1 <= rgb888_g * 8'd85 ;
        rgb_g_m2 <= rgb888_g * 8'd107;
        rgb_b_m0 <= rgb888_b * 8'd29 ;
        rgb_b_m1 <= rgb888_b << 3'd7 ;
        rgb_b_m2 <= rgb888_b * 8'd21 ;
    end
end

 //加
always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        img_y0  <= 16'd0;
        img_cb0 <= 16'd0;
        img_cr0 <= 16'd0;
    end
    else begin
        img_y0  <= rgb_r_m0 + rgb_g_m0 + rgb_b_m0;
        img_cb0 <= rgb_b_m1 - rgb_r_m1 - rgb_g_m1 + 1
        img_cr0 <= rgb_r_m2 - rgb_g_m2 - rgb_b_m2 + 1
    end

end
```

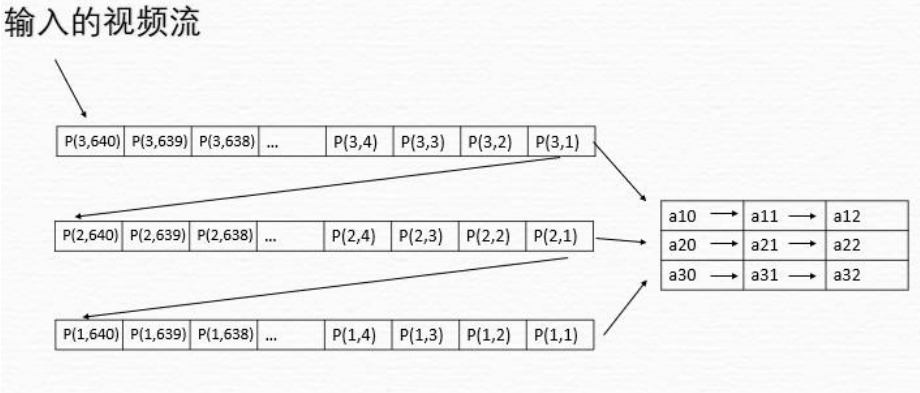
```
92    //移位，其实是取高位来实现
93  always @(posedge clk or negedge rst_n) begin
94      if(!rst_n) begin
95          img_y1  <= 8'd0;
96          img_cb1 <= 8'd0;
97          img_cr1 <= 8'd0;
98      end
99      else begin
100         img_y1  <= img_y0 [15:8];
101         img_cb1 <= img_cb0[15:8];
102         img_cr1 <= img_cr0[15:8];
103     end
104 end
105
106   //延时3拍以同步数据信号
107 always@(posedge clk or negedge rst_n) begin
108     if(!rst_n) begin
109         pre_frame_vsync_d <= 3'd0;
110         pre_frame_hsync_d <= 3'd0;
111     end
112     else begin
113         pre_frame_vsync_d <= {pre_frame_vsync_d[1:0], pre_frame_vsync};
114         pre_frame_hsync_d <= {pre_frame_hsync_d[1:0], pre_frame_hsync};
115
116     end
117 end
118
119 endmodule
120
```

3.2 高斯滤波模块



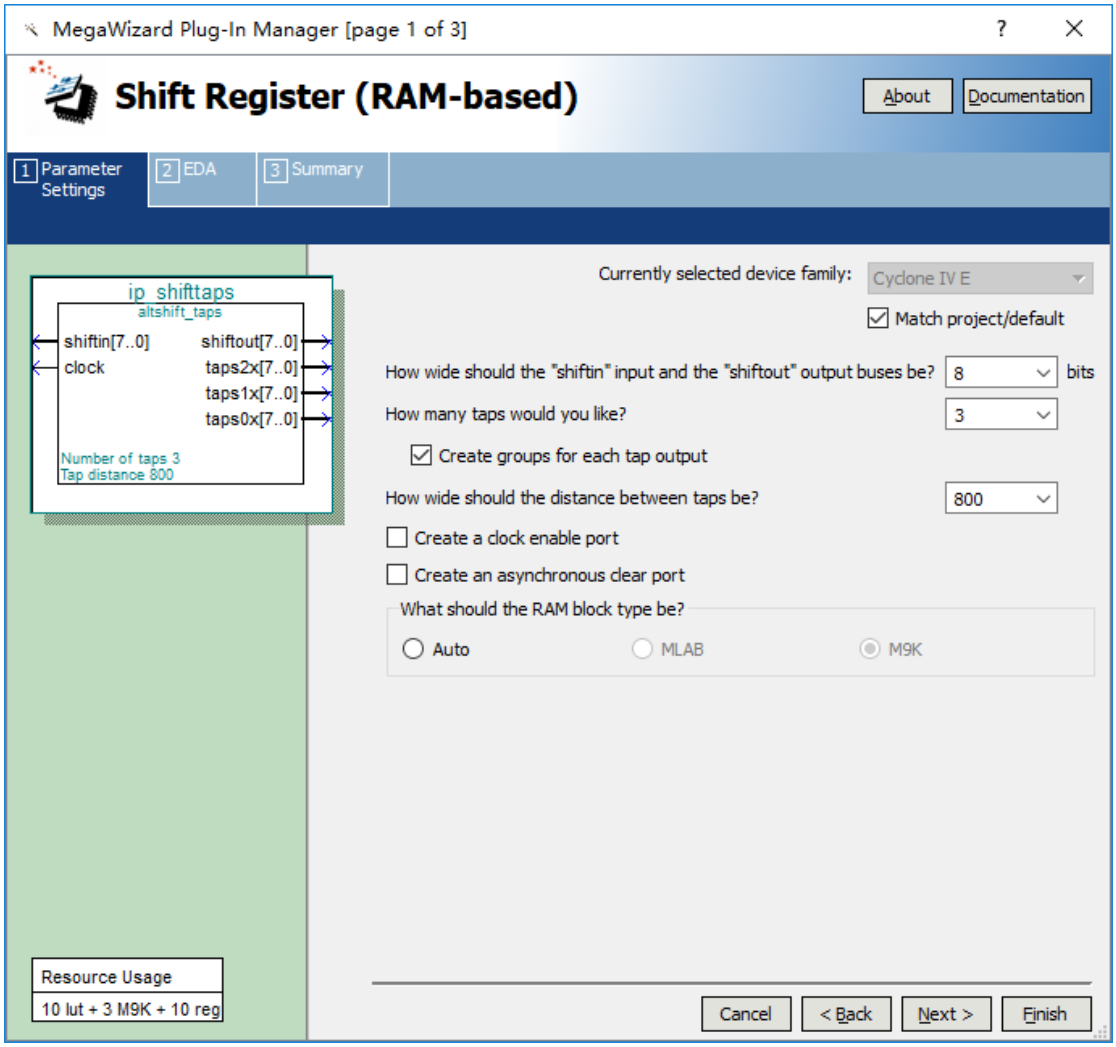
1	2	1
2	4	2
1	2	1

高斯滤波利用二维高斯函数的分布方式对图像进行平滑，是最常用的一种滤波算法。由于它是旋转对称的，所以不会改变原图像的边缘走向，用在需要边缘检测的场合非常合适。下面来讲一下具体的实施，这是假如一帧图像，我们将这个 3*3 的矩阵与图像上对应位置的数相乘后相加再处以 16，就是中间块的高斯滤波后的值。对整张图像的滤波就是将这个 3*3 的小滑窗在整帧图像上滑一遍。如何得到这样一个 3*3 小滑窗是一个难点，因为 FPGA 的图像信息是一个像素一个像素的给我们的。



可以通过三个移位寄存器来实现。三个移位寄存器分别存了三行的像素，每个像素最终移出时用这 9 个寄存器来存，就能实现。图中标出了数据的流动方向。比如下一个时钟上升沿，第一个移位寄存器出的数据会进入第二个移位寄存器和 a10，原本 a10 的数据进入 a11。

移位寄存器 IP 核的设置：



同提取亮度时一样要注意考虑行场同步信号的延时问题，后续每个模块都是，之后就不再赘述了。

代码:

```
module gauth_filter(
    //module clock
    input               clk             ,   // 模块驱动时钟
    input               rst_n           ,   // 复位信号

    //图像处理前的数据接口
    input               pre_frame_vsync ,   // vsync信号
    input               pre_frame_hsync ,   // hsync信号
    input   [7:0]       pre_img,

    //图像处理后的数据接口
    output              post_frame_vsync,   // vsync信号
    output              post_frame_hsync,   // hsync信号
    output      [7:0]   pos_img             // 输出图像Y数据
);

reg    [7:0]      a30;//3*3图像处理
reg    [7:0]      a31;
reg    [7:0]      a32;
reg    [7:0]      a20;
reg    [7:0]      a21;
reg    [7:0]      a22;
reg    [7:0]      a10;
reg    [7:0]      a11;
reg    [7:0]      a12;
reg    [4:0]      pre_frame_vsync_d;//用于延迟vsync和hsync，
reg    [4:0]      pre_frame_hsync_d;//延时5个周期
reg    [31:0]     sum1;
reg    [31:0]     sum2;
reg    [31:0]     sum3;
reg    [31:0]     sum123;
reg    [7:0]      after_gauth;

wire   [7:0]      line3;
wire   [7:0]      line2;
wire   [7:0]      line1;

//同步输出数据接口信号
assign post_frame_vsync = pre_frame_vsync_d[4]      ;
assign post_frame_hsync = pre_frame_hsync_d[4]      ;
assign pos_img          = post_frame_hsync ? after_gauth : 8'd0;


```

```
  //延时?拍以同步数据信号,3*3延时2拍,离斯3拍
always@(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        pre_frame_vsync_d <= 5'd0;
        pre_frame_hsync_d <= 5'd0;
    end
    else begin
        pre_frame_vsync_d <= {pre_frame_vsync_d[3:0], pre_frame_vsync};
        pre_frame_hsync_d <= {pre_frame_hsync_d[3:0], pre_frame_hsync};
    end
end

  //例化移位寄存器
ip_shifttaps u_ip_shifttaps(
      .clock          (clk),
      .shiftin        (pre_img),
      .shiftout       (),
      .taps0x         (line3),
      .taps1x         (line2),
      .taps2x         (line1)
      );

  //构成3*3
always@(posedge clk ) begin
    a32<=a31;
    a31<=a30;
    a30<=line3;
    a22<=a21;
    a21<=a20;
    a20<=line2;
    a12<=a11;
    a11<=a10;
    a10<=line1;
end

always@(posedge clk ) begin
    sum1<=a10+2*a11+a12;
    sum2<=2*a20+4*a21+2*a22;
    sum3<=a30+2*a31+a32;
end

always@(posedge clk ) begin
    sum123<=sum1+sum2+sum3;
end

always@(posedge clk ) begin
    after_gauth<=sum123[11:4];
end

  endmodule
```

3.3 Sobel 边缘检测及二值化模块

Sobel 算子：是一种离散性差分算子，该算子包含两组 3x3 的矩阵，分别为横向及纵向。

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * A$$

A 代表原始图像，Gx 及 Gy 分别代表经横向及纵向边缘检测的图像。

Sobel算子对于像素的位置的影响做了加权，可以降低边缘模糊程度，且是滤波算子的形式，可以利用快速卷积函数，简单有效。实现方法同高斯滤波。

图像的每一个像素的横向及纵向梯度近似值可用以下的公式来计算梯度的大小：

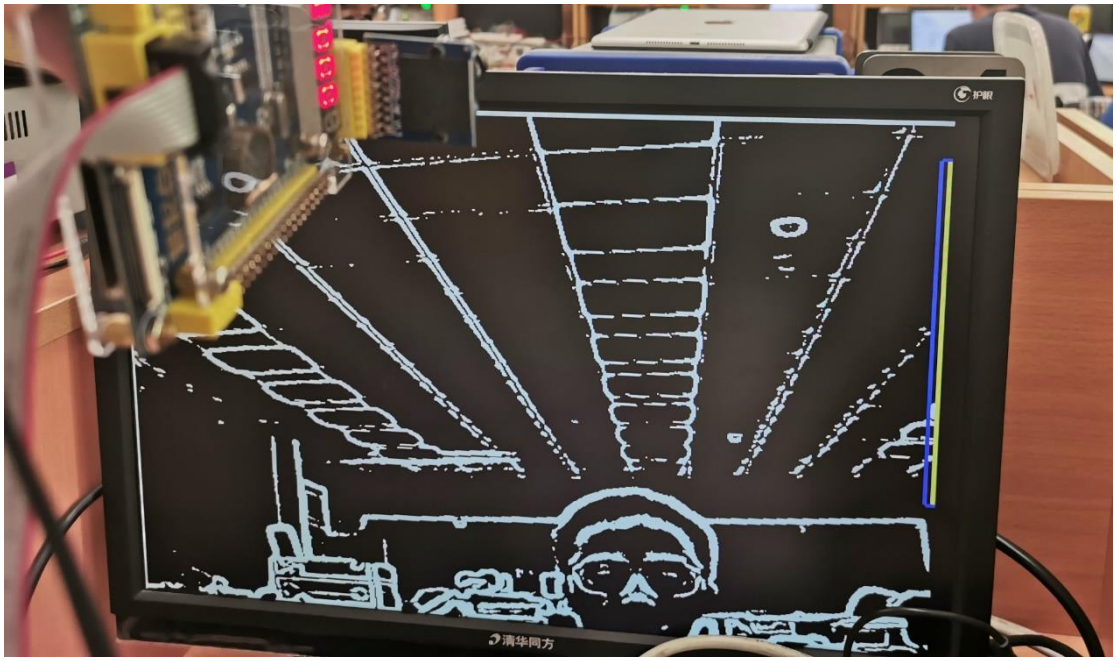
$$G = \sqrt{G_x^2 + G_y^2}$$

为节省 FPGA 资源，此次使用

$$G = |G_x| + |G_y|$$

Sobel 边缘检测后，图像边缘处值大，确定一个阈值，大于阈值赋白色，反之为黑色。

下图是 228 教室二值化后的边缘检测图像。

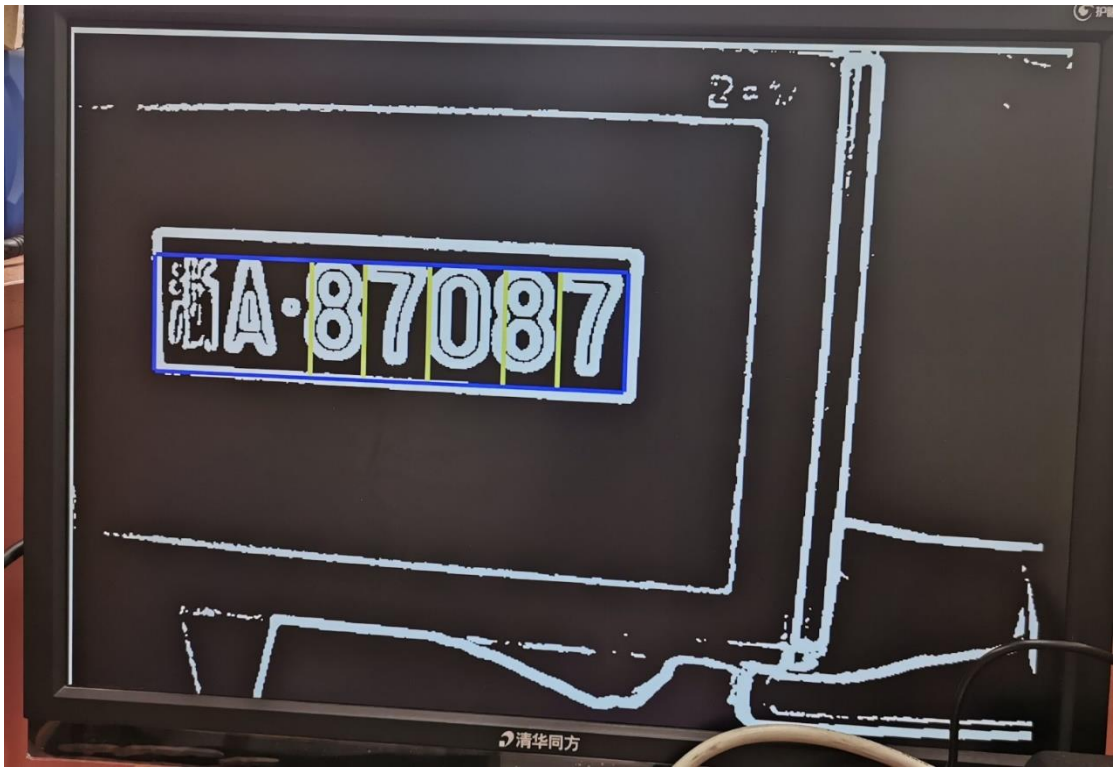


代码：

```v
module sobel_bi(
    //module clock
    input               clk             ,   // 模块驱动时钟
    input               rst_n           ,   // 复位信号

    //图像处理前的数据接口
    input               pre_frame_vsync ,   // vsync信号
    input               pre_frame_hsync ,   // hsync信号
    input   [7:0]       pre_img,

    //图像处理后的数据接口
    output              post_frame_vsync,   // vsync信号
    output              post_frame_hsync,   // hsync信号
    output      [15:0]   pos_img             // 输出图像Y数据
);


reg    [7:0]      a30;//3*3图像处理
reg    [7:0]      a31;
reg    [7:0]      a32;
reg    [7:0]      a20;
reg    [7:0]      a21;
reg    [7:0]      a22;
reg    [7:0]      a10;
reg    [7:0]      a11;
reg    [7:0]      a12;
reg    [5:0]      pre_frame_vsync_d;//用于延迟vsync和hsync，
reg    [5:0]      pre_frame_hsync_d;//延时6个周期
reg    [31:0]     gx1;
reg    [31:0]     gx2;
reg    [31:0]     absGx;
reg    [31:0]     gy1;
reg    [31:0]     gy2;
reg    [31:0]     absGy;
reg    [15:0]      G;
reg    [15:0]      after_bi;

wire   [ 7:0]      line3;
wire   [ 7:0]      line2;
wire   [ 7:0]      line1;

//同步输出数据接口信号
assign post_frame_vsync = pre_frame_vsync_d[5]      ;
assign post_frame_hsync = pre_frame_hsync_d[5]      ;
assign pos_img          = post_frame_hsync ? after_bi : 16'b0000_0000_0000_0000;

//延时?拍以同步数据信号,3*3延时2拍,sobel3拍,二值化1拍
always@(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        pre_frame_vsync_d <= 6'd0;
        pre_frame_hsync_d <= 6'd0;
    end
    else begin
        pre_frame_vsync_d <= {pre_frame_vsync_d[4:0], pre_frame_vsyn
        pre_frame_hsync_d <= {pre_frame_hsync_d[4:0], pre_frame_hsyn
    end
end

//例化移位寄存器
ip_shifttaps ul_ip_shifttaps(
      .clock            (clk),
      .shiftin          (pre_img),
      .shiftout         (),
      .taps0x           (line3),
      .taps1x           (line2),
      .taps2x           (line1)
      );

//构成3*3
always@(posedge clk ) begin
    a32<=a31;
    a31<=a30;
    a30<=line3;
    a22<=a21;
    a21<=a20;
    a20<=line2;
    a12<=a11;
    a11<=a10;
    a10<=line1;
end

always@(posedge clk ) begin
    gx1<=a10+2*a20+a30;
    gx2<=a12+2*a22+a32;
    gy1<=a30+2*a31+a32;
    gy2<=a10+2*a11+a12;
end


```

```
always@(posedge clk ) begin
    absGx<= gx1>gx2 ? gx1-gx2 : gx2-gx1;
    absGy<= gy1>gy2 ? gy1-gy2 : gy2-gy1;
end

always@(posedge clk ) begin
    G<=absGx+absGy;
end

/*always@(posedge clk ) begin
    after_bi<= G << 3'd7;
end*/
always@(posedge clk ) begin
    if(G<16'b0000_0000_0011_0000)
        after_bi<=16'b0000_0000_0000_0000;
    else
        after_bi<=16'b1111_1111_1111_1111;

end
/*always@(posedge clk ) begin
    if(G[15:0] < 16'b0000000011111111)
        after_bi<=16'b1111111111111111;
    else
        after_bi<=16'b0000000000000000;
end*/
endmodule
```

3.4 车牌上下边缘识别模块



车牌的边缘图像横线上由黑入白的次数大于 13 次，统计横线上的所有像素点都满足此条件的连续出现的行数，若大于阈值，则判定为车牌。

扫描过程中用两个寄存器记本时钟的像素是一帧中的多少行多少列，进满足条件的行时用一个寄存器存当时行，出的时候也存，并用两个寄存器标注此时有无满足的上下边。

出满足条件的行时对进出的行的值进行比较，满足大于一定阈值的条件就赋给输出用寄存器，不满足就回到没有进满足条件行的状态。

识别效果很好。

代码：

```v
module ttime_up_down(
    //module clock
    input               clk             ,   // 模块驱动时钟
    input               rst_n           ,   // 复位信号

    //图像处理前的数据接口
    input               pre_frame_vsync ,   // vsync信号
    input               pre_frame_hsync ,   // hsync信号
    input   [ 15:0]     pre_img,

    output  [ 10:0]     v_up,
    output  [ 10:0]     v_down
);

//parameter define

//640*480 60FPS_25MHz
parameter  H_SYNC   =  10'd96;    //行同步
parameter  H_BACK   =  10'd48;    //行显示后沿
parameter  H_DISP   =  10'd640;   //行有效数据
parameter  H_FRONT  =  10'd16;    //行显示前沿
parameter  H_TOTAL  =  10'd800;   //行扫描周期

parameter  V_SYNC   =  10'd2;     //场同步
parameter  V_BACK   =  10'd33;    //场显示后沿
parameter  V_DISP   =  10'd480;   //场有效数据
parameter  V_FRONT  =  10'd10;    //场显示前沿
parameter  V_TOTAL  =  10'd525;   //场扫描周期

//reg define
reg  [10:0] cnt_h;
reg  [10:0] cnt_v;

reg  [10:0] up_reg;
reg  [10:0] down_reg;

reg  [10:0] t_up;//记交叉穿过次数满足条件的上边，符合条件就给输出用reg

reg  [ 7:0] ttime;

reg         bw_delay;

reg         have_up;//状态量，是否确定上边界
reg         have_down;


wire     in_vga;//在图内否
wire     bw;//黑白


assign v_up = up_reg;
assign v_down = down_reg;

assign bw = pre_img[0];//取最低位，0黑，1白

//穿过次数检测的上升沿和清零+统计穿过次数
always @(posedge clk) begin
    bw_delay <= bw;
    if(cnt_h == H_SYNC+H_BACK+H_DISP-1)
        ttime<=0;
    else if (bw==0 && bw_delay==1)
        ttime<=ttime+1;
    else
        ttime<=ttime;
end


//记开始结束+判断上下边界距离是否满足要求,满足就输出，在下一次满足条件时输出的两个寄存器的值都不变
always @(posedge clk) begin
    if(ttime>=13 && have_up==0)begin
        t_up<=cnt_v;
        have_up<=1'b1;
        end
    else if(cnt_h == H_SYNC+H_BACK+H_DISP-2 && ttime<7 && have_up==1 && have_down==0 )begin
        if(cnt_v-t_up>50)begin
            up_reg<=t_up;
            down_reg<=cnt_v;
            have_down<=1;
            end
        else
            have_up<=0;
    end
    else if(cnt_h==H_TOTAL-1 && cnt_v==V_TOTAL-1)begin
        have_down<=0;
        have_up<=0;
    end

end
```

```
89
90
91
92      //行计数器对像素时钟计数
93      always @(posedge clk or negedge rst_n) begin
94          if (!rst_n)
95              cnt_h <= 10'd0;
96          else begin
97              if(cnt_h < H_TOTAL - 1'b1)
98                  cnt_h <= cnt_h + 1'b1;
99              else
100                 cnt_h <= 10'd0;
101         end
102     end
103
104     //场计数器对行计数
105     always @(posedge clk or negedge rst_n) begin
106         if (!rst_n)
107             cnt_v <= 10'd0;
108         else if(cnt_h == H_TOTAL - 1'b1) begin
109             if(cnt_v < V_TOTAL - 1'b1)
110                 cnt_v <= cnt_v + 1'b1;
111             else
112                 cnt_v <= 10'd0;
113         end
114     end
115     endmodule
116
```

3.5 车牌左右边缘识别模块

在原方案失败后决定利用上之前识别出的准确的车牌上下边界来帮助识别车牌左右边界，不过这样会导致识别出最后边界的时间多一帧图像的时间，即 1/60 秒。

之后的处理只看车牌上下边界内的像素：

设置两个 800 位的标志寄存器 allare 和 haveit 统计上下边界内每一列，allare 统计是否一整列都为白色，Haveit 统计一整列中是否有白色像素存在。

首先，一整列都是白色的才可能是车牌的左右边界，然后，利用上车牌内字符都一列中有白色但不是一整列都白的这一特征，可以区分出车牌左右边缘和其它白列。

代码：

```v
module left_right(
    //module clock
    input               clk             ,   // 模块驱动时钟
    input               rst_n           ,   // 复位信号

    //图像处理前的数据接口
    input               pre_frame_vsync ,   // vsync信号
    input               pre_frame_hsync ,   // hsync信号
    input   [ 15:0]     pre_img,

    input   [ 10:0]     v_up,
    input   [ 10:0]     v_down,

    output  [ 10:0]     h_left,
    output  [ 10:0]     h_right

);

//parameter define

//640*480 60FPS_25MHz
parameter  H_SYNC   =  10'd96;    //行同步
parameter  H_BACK   =  10'd48;    //行显示后沿
parameter  H_DISP   =  10'd640;   //行有效数据
parameter  H_FRONT  =  10'd16;    //行显示前沿
parameter  H_TOTAL  =  10'd800;   //行扫描周期

parameter  V_SYNC   =  10'd2;     //场同步
parameter  V_BACK   =  10'd33;    //场显示后沿
parameter  V_DISP   =  10'd480;   //场有效数据
parameter  V_FRONT  =  10'd10;    //场显示前沿
parameter  V_TOTAL  =  10'd525;   //场扫描周期

//reg define
reg  [10:0] cnt_h;
reg  [10:0] cnt_v;

reg  [10:0] left_reg;
reg  [10:0] right_leg;
reg  [10:0] t_left;
reg  [10:0] t_right;

reg         bw_delay;
reg         have_left;
reg         have_right;
wire        vga_en;

wire        bw;
assign h_left = left_reg;
assign h_right = right_leg;

assign bw = pre_img[0];//取最低位，0黑，1白
assign vga_en  = (((cnt_h >= H_SYNC+H_BACK) && (cnt_h < H_SYNC+H_BACK+H_DISP))
                 &&((cnt_v >= V_SYNC+V_BACK) && (cnt_v < V_SYNC+V_BACK+V_DISP)))
                 ?  1'b1 : 1'b0;

//在v_up~v_down里，统计哪些竖行整竖行都白色,通过车牌后有白字，纸后没东西判断

reg     [799:0]   allare;//0表示全白色，1表示不是全白
reg     [799:0]   haveit;//1表示有，0表示没有白色

always @(posedge clk)begin
    if(cnt_v==v_down-1)begin   //一帧的清算时间
        if(allare[cnt_h]==1 && allare[cnt_h+1]==0)begin  //出纸边或者车牌边
             if(haveit[cnt_h+20 +:30] != 30'd0 && have_left==0 && vga_en && cnt_h>H_SYNC+H_BACK+20)begin
                 left_reg<=cnt_h;
                 have_left<=1;
             end
             else if(have_left==1 && have_right==0 && haveit[cnt_h+20 +:20] == 20'd0 )begin
                 right_leg<=cnt_h;
                 have_right<=1;
             end
        end
    end

    else if(cnt_h==H_TOTAL-1 && cnt_v==V_TOTAL-1)begin//一帧结束归零为下一帧作准备
        have_left<=0;
        have_right<=0;
        haveit<=800'd0;
        allare<=800'd0;
    end

    else if(cnt_v>v_up && cnt_v<v_down)begin   //两个寄存器列的打标记
        if(bw==1)
            haveit[cnt_h]<=1;
        else
            allare[cnt_h]<=1;
    end
end
```

```
 //行计数器对像素时钟计数
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        cnt_h <= 10'd0;
    else begin
        if(cnt_h < H_TOTAL - 1'b1)
            cnt_h <= cnt_h + 1'b1;
        else
            cnt_h <= 10'd0;
    end
end
 //场计数器对行计数
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        cnt_v <= 10'd0;
    else if(cnt_h == H_TOTAL - 1'b1) begin
        if(cnt_v < V_TOTAL - 1'b1)
            cnt_v <= cnt_v + 1'b1;
        else
            cnt_v <= 10'd0;
    end
end
endmodule
```

3.6 各个车牌号边界确定模块

由于车牌号的大小间距都是成固定的比例的，所以在准确识别出车牌左右边界后可以简单地通过计算得到各个数字的边界。

代码：

```
always @(posedge clk) begin
    b1_reg<= 29*h_left/44+15*h_right/44;
    b2_reg<= 6*h_left/11+5*h_right/11;
    b3_reg<= 9*h_left/22+13*h_right/22;
    b4_reg<= h_left/4+3*h_right/4;
    b5_reg<= 3*h_left/22+19*h_right/22;
    b6_reg<= h_right;
end
```

3.7 实时显示二值化边缘图像与各边界

显示二值化边缘图像，并将车牌边缘用蓝色标出，车牌号边界用黄色标出

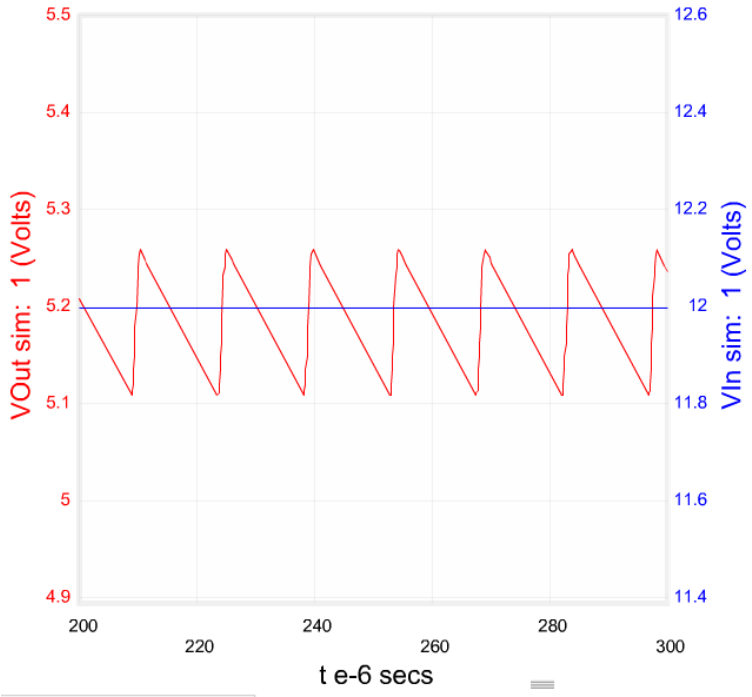
代码：

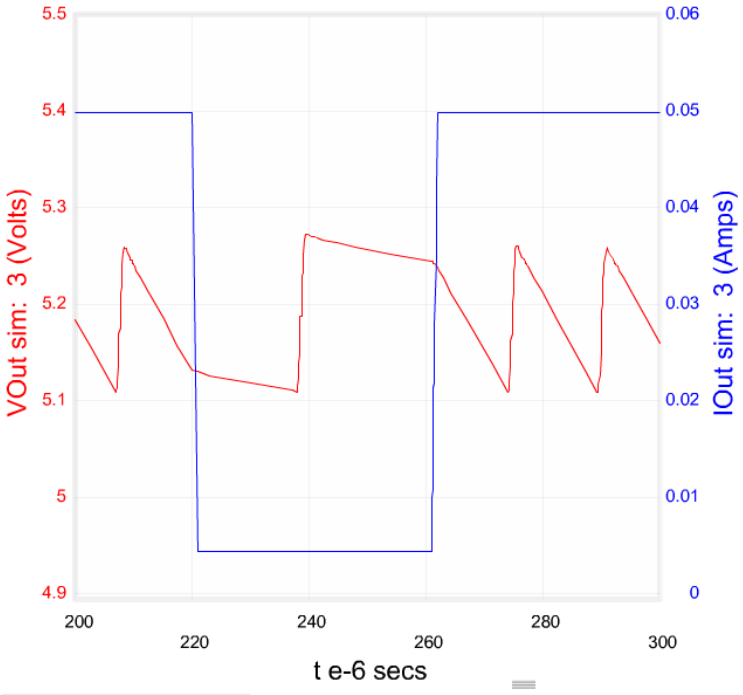
```
parameter  BLUE     =  16'b00000_000000_11111;
parameter  YELLOW   =  16'b11111_111111_00000;
//reg define
reg  [10:0] cnt_h;
reg  [10:0] cnt_v;
reg  [15:0] pre_img_d;
//wire define
wire        vga_en;

reg vs;
reg hs;
assign pos_img    = vga_en ? pre_img_d : 16'b0000_0000_0000_0000;//
assign post_frame_hsync=hs;
assign post_frame_vsync=vs;
assign vga_en  = (((cnt_h >= H_SYNC+H_BACK) && (cnt_h < H_SYNC+H_BACK+H_DISP))
                 &&((cnt_v >= V_SYNC+V_BACK) && (cnt_v < V_SYNC+V_BACK+V_DISP)))
                 ?  1'b1 : 1'b0;
always@(posedge clk ) begin
    vs<=pre_frame_vsync;
    hs<=pre_frame_hsync;
    if(v_up < cnt_v && cnt_v < v_down &&  h_left-2<cnt_h && cnt_h < h_left+2)
        pre_img_d<=BLUE;
    else if(v_up < cnt_v && cnt_v < v_down &&  h_right-2<cnt_h && cnt_h < h_right+2)
        pre_img_d<=BLUE;
    else if(v_up-2 < cnt_v && cnt_v < v_up+2 &&  h_left<cnt_h && cnt_h < h_right)
        pre_img_d<=BLUE;
    else if(v_down-2 < cnt_v && cnt_v < v_down+2 &&  h_left<cnt_h && cnt_h < h_right)
        pre_img_d<=BLUE;
    else if(v_up < cnt_v && cnt_v < v_down &&  b1-2<cnt_h && cnt_h < b1+2)
        pre_img_d<=YELLOW;
    else if(v_up < cnt_v && cnt_v < v_down &&  b2-2<cnt_h && cnt_h < b2+2)
        pre_img_d<=YELLOW;
    else if(v_up < cnt_v && cnt_v < v_down &&  b3-2<cnt_h && cnt_h < b3+2)
        pre_img_d<=YELLOW;
    else if(v_up < cnt_v && cnt_v < v_down &&  b4-2<cnt_h && cnt_h < b4+2)
        pre_img_d<=YELLOW;
    else if(v_up < cnt_v && cnt_v < v_down &&  b5-2<cnt_h && cnt_h < b5+2)
        pre_img_d<=YELLOW;
    else if(v_up < cnt_v && cnt_v < v_down &&  b6-2<cnt_h && cnt_h < b6+2)
        pre_img_d<=YELLOW;
    else
        pre_img_d<=pre_img;
end
```

六、课题开发与调试中出现的问题分析

问题一：电源管理电路输出电压不稳定且容易烧坏芯片
在面包板上调节电源管理电路时输出比较稳定，但是焊接到万能板上时输出波动很大，波形为锯齿波，峰峰值可以达到 1V，同时有很多的毛刺，芯片也易发热甚至烧坏。

原因分析：经过仔细分析发现，电路的输出结果与参数选择有很大的关系，这一点可以从 WEBENCH 仿真中看到，在输出电流较大时，电路输出电压很稳定，纹波在 0.03V 左右，而随着输出电流变小，电路的波动也变得非常大，到了 0.2V 左右，其对负载的变化也表现地不够好，可以从下面小电流输出的仿真图看出。这说明这个电源管理电路由于芯片本身的性质比较适用于有较大输出电流的负载，而本系统使用的负载工作电流仅为 15mA，当然会有很大波动；而对于毛刺，除了电路的特性，与电路的稳定也有很大关系，这也是造成芯片容易烧坏的原因（出现虚焊、连接不稳定、电路元件集中等）。





解决：可以适当调节电路参数，调节滤波电容的容值以减少毛刺，使输出稳定；在焊接时注意元件布局，保证接线良好，同时可以增加负载以增大输出电流来解决输出不稳的问题。通过调整最终得到的电源电路能够正常工作，输出电压稳定在 0.1V，比较理想。

问题二：光控电路无论怎样调节敏感程度都无法实现光控
分析与解决：分析发现尽管电路元件都没有损坏，但是光控电阻的阻值特性与所标定的相差比较大，其有光照时阻值仅为 2KΩ，无光照时阻值为几十千欧，所以这样很难达比较器的阈值电压，在光敏电阻一段再串一个电阻以降低反相输入端的电压就可以解决这个问题。

问题三：直接给最终要输出的 rgb 信号赋定值时显示黑色，用 assign 语句赋值则只显示第一种颜色，该为第二种颜色的地方为黑色，且第二种颜色的值会影响到第一种颜色。比如，assign rgb = 条件 A? 16'b11111_111111_11111 : 16'b00000_000000_11111 应该是满足条件时显示白色，不满足时显示蓝色，结果满足条件显示为黄色，不满足时为黑色。因为调试时经常会用到用什么条件来影响直接给最终输出赋值，所以这个问题导致我大半天几乎没有进展，不过这大半天进行了完整的试验，为后续调试打下扎实基础。
一天调试的发现：
1.通过在 VGA 显示彩条的样例代码中各种尝试分析得出：如果设一个 16 位的寄存器，给这个寄存器赋一个定值，然后再 assign 输出的 rgb 等于这个寄存器的值，则可以正常显示。但如果直接给输出的 rgb 赋定值就会黑屏。这个虽然不知道理由，但是在使用时注意下，多转一道就可以避开这个问题。
2.直接用 assign 语句给输出赋一个条件判断后选择的值时，eg assign rgb= A? c1:c2 满足条件会显示 记位置 B 为 c2 中为 1 的各位，将 c1 里在位置 B 上的 1 改为 0 的颜色。所以在后续需要用到此类型语句时都将 c2 设为 16'd0，即黑色，即可不影响后续调试。

问题四：高斯滤波出来的值不对
原因与解决：1.一开始用的 2 个移位寄存器来实现，可以实现但思路不够清晰，同时也不清

楚移位寄存器 IP 核除了两个移位寄存器外的那个 shiftout 的时钟延迟情况，也没想清楚它的两个 tap 哪个是第一行哪个是第二行，最终决定用三个移位寄存器来实现。
2.最开始想错了，以为用 640 深度的就行，但想清楚发现这样是实现不了 3*3 小滑块的。改成深度 800，并通过行列的寄存器来判断是否是在屏幕上要显示的像素部分

问题五：车牌左右边界识别。

最初方案是不用边缘识别后的图像而是直接使用最初的 RGB 图像，通过进出车牌的蓝色块来判断左右边界。
在不考虑复杂光照环境和蓝色喷漆的车的情况下，本来按说应该是会较为精准的一个方法，结果调试一上午后放弃。

失败原因：通过直接把摄像头本身采集到的画面不加处理地显示在屏幕上发现，摄像头本身原因导致视频画面偏淡发白，通过设置 rgb 的红色绿色蓝色阈值来判断无法将车牌的蓝色和近白色的颜色无法区分开，无论怎么调整三个阈值，识别成功率都相当低，故舍弃这种方法，自己又想了最终使用的这个方案，以牺牲一点速度为代价换取高识别准确度和精度，为后续各个数字的分割和识别打好基础。

原方通过蓝色识别方案代码核心部分：

```
reg  [10:0] b_l;//记蓝色进出，符合条件就给输出用reg
reg  [10:0] b_r;
reg         surelr;
wire in_vga;//在图内否
wire is_blue;
reg is_blue_delay;//用于检测进出蓝块

assign h_left  = left_reg;
assign h_right = right_reg;
//assign is_blue =  ( pre_img[15:11] < 5'b01100   &&  pre_img[10:5] < 6'b100000   &&    pre_img[4:0] > 5'b10000);
//assign is_blue = pre_img[4:0]>5'b10000;
assign  is_blue = (pre_img[15:11] < 5'b10011 && pre_img[4:0]>5'b01100 && pre_img[4:0]>pre_img[9:5]);
assign in_vga  = (((cnt_h >= H_SYNC+H_BACK) && (cnt_h < H_SYNC+H_BACK+H_DISP))
                 &&((cnt_v >= V_SYNC+V_BACK) && (cnt_v < V_SYNC+V_BACK+V_DISP)))
                 ?  1'b1 : 1'b0;

//统计进出蓝块+蓝块确认是车牌输出+蓝块左右边临时寄存器清零
always @(posedge clk or negedge rst_n) begin
    if(!rst_n)
        is_blue_delay<=0;
    else begin
        is_blue_delay<=is_blue;
        if(cnt_h==H_TOTAL-1 && cnt_v==V_TOTAL-1)begin
            b_l<=0;
            b_r<=0;
            surelr<=0;
        end
        else if(cnt_h==H_SYNC+H_BACK+H_DISP-1)begin  //一行结束最终的出蓝块和进蓝块比
            if(b_r-b_l>100 && surelr==0)begin
                left_reg<=b_l;
                right_reg<=b_r;
                surelr<=1;
                end
            else begin
                b_l<=0;
                b_r<=0;
                end
         end
         else begin
             if(!is_blue_delay && is_blue && in_vga && b_l==0)//第一次进入蓝色块记左边
                 b_l<=cnt_h;
             else if (is_blue_delay && !is_blue && in_vga)
                 b_r<=cnt_h;
             else
                 b_r <= b_r;
         end
    end
end
```

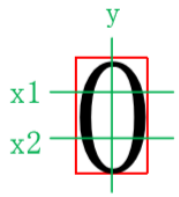
问题六：每个数字的边界划分

原方案：通过竖直方向投影，车牌内字符间有空隙，空隙在车牌上下边界内的二值化边缘图像是一整列没有白色像素的来判断每个字符的左右边界。
但是，经边缘处理后原本就不大的缝隙更小了，对车牌角度要求过高，歪一点在缝隙内就找不到没白色像素的一整列了。识别率太差，最终弃掉这种方法。

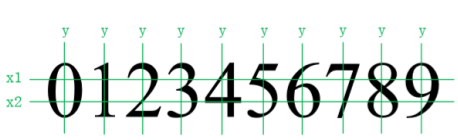
原方案代码核心部分：

```
76   //该帧像素最后一行扫描标出5个边界，出黑块时标
77  always @(posedge clk) begin
78      bw_delay <= bw;
79
80      if(!rst_n) begin//列有无数字，为标边界奠基
81          h_border<=0;
82          segment<=0;
83          end
84      else if(bw==0)
85          h_border[cnt_h]<=1;
86
87      if(cnt_v==v_down-1)begin
88           if(h_border[cnt_h]==0 && h_border[cnt_h-1]==1)begin//出有黑字的列
89              segment<=segment+1;
90              case(segment)
91                  5'd3: begin
92                      b1_reg<=cnt_h;
93                      end
94                  5'd4: begin
95                      b2_reg<=cnt_h;
96                      end
97                  5'd5: begin
98                      b3_reg<=cnt_h;
99                      end
100                 5'd6: begin
101                     b4_reg<=cnt_h;
102                     end
103                 5'd7: begin
104                     b5_reg<=cnt_h;
105                     end
106                 5'd8: begin
107                     b6_reg<=cnt_h;
108                     end
109                 default: begin
110                     b1_reg<=b1_reg;
111                     end
112             endcase
113          end
114     end
115
116     else if(cnt_h==h_right-1 && cnt_v==v_down-1)begin      //清零为下一帧准备
117         segment<=0;
118         h_border<=0;
119     end
120 end
121
```

问题七：识别已经切分出边缘的车牌号（只考虑数字，暂时做不到英文和汉字）
方案一：考虑到车牌号是打印体，所以可以使用数字特征识别来判断。



统计数字中心的竖线与数字的交叉点，在横线 x1 上 y 左边的交点，横线 x1 上 y 右边的交点，横线 x2 上 y 左边的交点，横线 x2 上 y 右边的交点 这五个信息便可确定出数字。但是，经过边缘处理后的图像 和 网上常用的讲数字特征识别的数字图对比



虽然边缘识别后很清楚人眼可以清楚认出，但是用特征识别法时可以预感到精准度会很成问题，但一开始没想到别的可行方法，就还是硬着头皮先尝试了下
原方案代码核心部分：

```
//负责检测出5个数字给出，由超声测距模块判断什么时候读了显示
module num_figure(
    //module clock
    input               clk             ,   // 模块驱动时钟
    input               rst_n           ,   // 复位信号

    //视频流信息
    input               pre_frame_vsync ,   // vsync信号
    input               pre_frame_hsync ,   // hsync信号
    input   [15:0]      pre_img,

    //车牌边界信息
    input   [10:0]      v_up ,
    input   [10:0]      v_down,
    input   [10:0]      h_left,
    input   [10:0]      h_right,

    //各个数字边界信息
    input  [10:0]       b1,
    input  [10:0]       b2,
    input  [10:0]       b3,
    input  [10:0]       b4,
    input  [10:0]       b5,
    input  [10:0]       b6,

    //输出5个4位BCD码的数字
    output [ 3:0]       outnum1,
    output [ 3:0]       outnum2,
    output [ 3:0]       outnum3,
    output [ 3:0]       outnum4,
    output [ 3:0]       outnum5
);
```

```
116     reg             y11_delay;//检测列方向变化
117     reg             y22_delay;
118     reg             y33_delay;
119     reg             y44_delay;
120     reg             y55_delay;
121
122     wire  [ 10:0]   y1;//要检测的5个竖线
123     wire  [ 10:0]   y2;
124     wire  [ 10:0]   y3;
125     wire  [ 10:0]   y4;
126     wire  [ 10:0]   y5;
127     wire  [ 10:0]   x1;//要检测的两条横线
128     wire  [ 10:0]   x2;
129     wire            bw;
130
131     assign y1=(b1+b2)/2;
132     assign y2=(b2+b3)/2;
133     assign y3=(b3+b4)/2;
134     assign y4=(b4+b5)/2;
135     assign y5=(b5+b6)/2;
136     assign x1=3*v_up/5+2*v_down/5;
137     assign x2=v_up/3+2*v_down/3;
138     assign bw=pre_img[0];
139     assign outnum1 = num1;
140     assign outnum2 = num2;
141     assign outnum3 = num3;
142     assign outnum4 = num4;
143     assign outnum5 = num5;
```

```
else if(cnt_v==x1)begin //统计x1交点
    if(bw_delay && !bw)begin
        if(b1<cnt_h && cnt_h<y1)
            tnumx11_r<=tnumx11_r+1;
        else if(y1<cnt_h && cnt_h<b2)
            tnumx11_l<=tnumx11_l+1;

        else if(b2<cnt_h && cnt_h<y2)
            tnumx12_r<=tnumx12_r+1;
        else if(y1<cnt_h && cnt_h<b2)
            tnumx12_l<=tnumx12_l+1;

        else if(b3<cnt_h && cnt_h<y3)
            tnumx13_r<=tnumx13_r+1;
        else if(y1<cnt_h && cnt_h<b2)
            tnumx13_l<=tnumx13_l+1;

        else if(b4<cnt_h && cnt_h<y4)
            tnumx14_r<=tnumx14_r+1;
        else if(y1<cnt_h && cnt_h<b2)
            tnumx14_l<=tnumx14_l+1;

        else if(b5<cnt_h && cnt_h<y5)
            tnumx15_r<=tnumx15_r+1;
        else if(y1<cnt_h && cnt_h<b2)
            tnumx15_l<=tnumx15_l+1;

        else
            tnumx11_r<=tnumx11_r;
    end
end

/统计所有列线交点
lways @(posedge clk or negedge rst_n) begin
  if (!rst_n)begin
      tnumy1<=0;
      tnumy2<=0;
      tnumy3<=0;
      tnumy4<=0;
      tnumy5<=0;
      end
  else begin
      case(cnt_h)
          y1:begin
              y11_delay<=bw;
              if(y11_delay && !bw)
                  tnumy1<=tnumy1+1;
              else
                  tnumy1<=tnumy1;
             end
          y2:begin
              y22_delay<=bw;
              if(y22_delay && !bw)
                  tnumy2<=tnumy2+1;
              else
                  tnumy1<=tnumy1;
             end
          y3:begin
              y33_delay<=bw;
              if(y33_delay && !bw)
                  tnumy3<=tnumy3+1;
              else
                  tnumy1<=tnumy1;
```

统计列向上穿过交点数也就是多设 5 个寄存器存下上一行的这 5 列的像素便可判断。

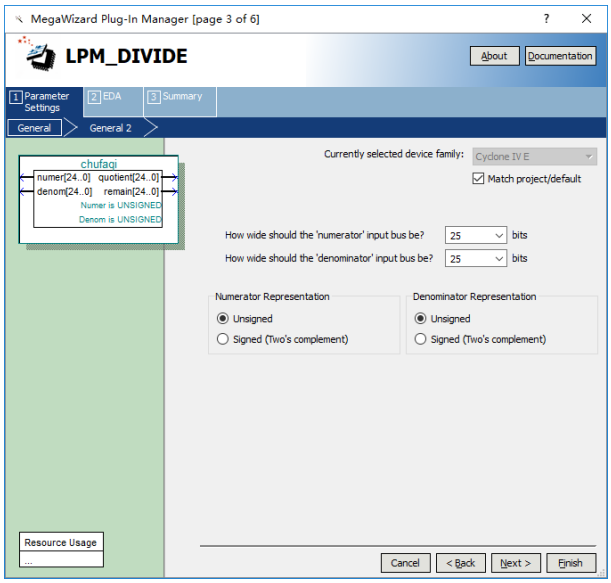
按说一帧结束前各个位置交点数都清楚了，可通过对比各个交点数和数字对应的表来确定是什么数字。那么首先就要一个一个数字实测得出这个表。调试测试时让 5 个交点数分别显示在数码管上。

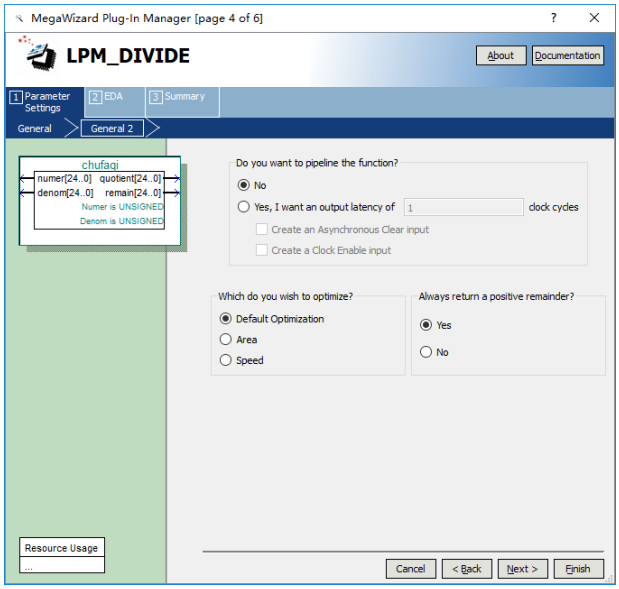
但是惨痛地发现数字跳动飞快，根本看不成，果断放弃了这种方法。

方案二：因为查的资料主流就两种，一种是上述特征识别法，另一个就是用神经网路，查了下神经网路法步骤大概是在电脑上训练神经网络好了后把参数矩阵存入 RAM，FPGA 读 RAM 让参数矩阵与输入图像向量进行运算得到最终结果，考虑这个实现任务量较大，就目前我组对神经网络和 FPGA 读写 RAM 的熟练程度来看，规定时间内肯定实现不出来，故放弃神经网络法，另辟蹊径。考虑到之前的车牌和各个数字的识别和边界分割较为准确，尝试利用测一个数字的白色像素点占整个边界方框的百分比不同来区分各个数字。

一开始写完这部分代码编译时说 FPGA 内寄存器数量不够，故改成例化除法的运算器来实现除，取余操作。

除法器ＩＰ核设置：





代码核心部分：

```
chufaqi ul1_chufaqi(
        .denom          (allv1),
        .numer          (valuel*10),
        .quotient       (radiol1),
        .remain         ()
        );

chufaqi ul21_chufaqi(
        .denom          (allv1),
        .numer          (valuel*100),
        .quotient       (temp1),
        .remain         ()
        );
chufaqi ul22_chufaqi(
        .denom          (10),
        .numer          (temp1),
        .quotient       (),
        .remain         (radiol2)
        );
```

白色像素占比就是通过统计白色像素块和总块数从而计算出占的比例的那个小数的第一二三四五位（后续发现只需要 2 位即可）

调试过程中测单个数字的白像素占比结果并显示到数码管上：第二位小数上有 1 的波动，波动很小，各个数字所占比例可以分开！制定出的判定区间如下

```
 //num1
always @(posedge clk)begin
    if(cnt_h>b1 && cnt_h<b2 && cnt_v>v_up && cnt_v<v_down)begin
        valuel<=valuel+bw;
        allv1<=allv1+1;
    end
    else if(cnt_h==H_TOTAL-1 && cnt_v==V_TOTAL-1)begin
        valuel<=0;
        allv1<=0;
        if(radiol1==4)begin
            if( radiol2==2 || radiol2==3 )
                num1<=4'd3;
            else if( radiol2==0 || radiol2==1 )
                num1<=4'd9;
            else if( radiol2==5 || radiol2==6 )
                num1<=4'd5;
            else if( radiol2==8 || radiol2==9 )
                num1<=4'd4;
        end
        else if(radiol1==5)begin
            if( radiol2<=5 )
                num1<=4'd0;
            else
                num1<=4'd8;
        end
        else if(radiol1==6)begin
            num1<=8;
        end
        else if(radiol1==3)begin
            if(radiol2<=5 )
                num1<=4'd7;
            else
                num1<=4'd2;
        end
        else
            num1<=4'd1;
    end
end
```

但是，此方法使用时对距离有限制，测得的这个标准是在一定距离范围内可用的，因为测量时发现打印的车牌到摄像头的距离改变，对比例能有 0.03 甚至 0.04 的改变，不过由于我们设计了一个超声波测距模块，当测的距离为我们制作如上标准时，再读取数字识别的结果便可较为准确。

但是，同时统计 5 个数字的比例并在数码管上显示时，FPGA 板的寄存器数量不够了 . . . 最初不加数字识别模块时，已经用了 8000 多个寄存器，加上后达到了 16000，即是将除法和取余用 IP 核实现，要计算 5 个小数的一二位需要例化 15 个除法器，最终还是超寄存器数量了（购买的板子寄存器 10000 个）。

将超声波模块的部分用之前的绿板子去运行时（超声波模块要用 1400 个寄存器），在新买的板子上对单个数字进行调试和测量还是够的，这个方法虽然精度不高且有距离要求，但至少验证了是可行的！没能实现 5 个数字识别显示在数码管上是本项目很大的一个遗憾了。

七、创新点

1.对于车牌左右边界的识别，由于摄像头本身的问题，视频中的蓝色比较淡，利用传统的边界检测办法很难准确地找到车牌边缘，所以系统创新性地通过图像中横线有黑至白的次数来判定是否是边界，准确性得到了很大的提高，具体实现思路见数字系统设计的边缘检测模块。
2.对于车牌字符的分割，常用方法是通过图像竖直方向的投影判断某一列无白色像素点来判断字符空隙，但若车牌是歪的则很难识别；因为常用车牌的长宽标准接近，因此直接计算好比例来分割字符，同时经过微调能有很好的效果，该方法准确性也非常高

八、不足与展望

1.本次系统设计中因为比较注重数字系统的设计，同时使用的模块集成度比较高，所以模拟部分的设计显的相对简单，以后可以通过考虑实际环境，加入电机驱动、语音通知系统、蓝牙传输系统等更好地实现整体性的车牌识别和其扩展功能；
2.本次设计的最初目的是实现对于车牌字符的识别，然而由于 FPGA 的寄存器数量不够，最终没有实现，之后希望可以一方面精简程序减少寄存器数量，同时搭载性能更加强大的 FPGA 实现字符的识别。
3.对于字符识别部分，本系统设计的方法只能机械地识别标准的车牌，但实际上利用神经网络可以实现识别各种字形的车牌，大大增强系统的可应用性，同时通过查阅资料知道是可以利用 FPGA 实现神经网络的，也许以后会向这一方面的研究靠近。

九、参考文献：

[1]基于 FPGA 的 Sobel 算子实现实时图像边缘检测系统的设计[D].李涛.北京交通大学 2013
[2]图像锐化与边缘检测之 Roberts 算子、Prewitt 算子、Sobel 算子和 Laplacian 算子 CSDN 博客：https://blog.csdn.net/Eastmount/article/details/89001702
[3]图像处理常用边缘检测算子：https://blog.csdn.net/qq_37995260/article/details/79746924
图像滤波算法——均值、中值、高斯：
https://blog.csdn.net/qq_37568748/article/details/80344709
[4]基于 FPGA 的车牌识别系统. 与非网. 2014-07-24. https://www.eefocus.com/fpga/329632
[5]杨鼎鼎,陈世强,刘静漪.基于车牌背景和字符颜色特征的车牌定位算法[J].计算机应用与软件,2018,35(12):216-221.
[6]刘伟,刘广文.基于 FPGA 的车牌识别算法研究[J].科技资讯,2014,12(24):26.
[7]张晓峰,曾飞,孙荣,吴仪.基于 FPGA 的车牌识别系统设计[J].物流工程与管理,2016,38(03):207-209.
[8]王凯. 基于 FPGA 的车牌识别算法的实现[D].福州大学,2016

十、工作日志：

7.1	上午	完成预习报告答辩验收。 在网上搜查资料并完成绝大部分器件的购买。 利用面包板初步搭好电源管理电路以便后续仿真。
	下午	ALIENTEK 开拓者 FPGA 开发板硬件资源学习。 复习并拓展 modelsim 相关知识。 Verilog 语法复习。

7.2	上午	pll 和 ram 的 ip 核的学习。
	下午	VGA 原理、VGA 读 ROM 显示的程序。 下载 PCB 设计软件 Altium Designer，电源管理电路的 PCB 原理图处理。
7.3	上午	PCB 制图知识的了解和相关的微调。 FIFO 的 ip 核的学习，SDRAM 使用的学习（学了一半，没完全弄明白）。
	下午	I2C 学习，研究写 I2C 驱动模块代码。 测验新买的 FPGA 的各个功能正常，没有硬件问题。 修改电源管理电。
7.4	上午	继续学习使用 SDRAM 将其改成 FIFO 用户接口方便以后使用 修改电源管理电路，更换样片使得可以利用转接板焊接到万能板上。
	下午	研究 OV7725 如何使用。 学会图像由 RGB565 格式转为 YCBCR 格式并进一步二值化。
7.5	上午	查移位寄存器资料，编写用移位寄存器得到图像 3*3 的滑块用于后续处理。 查找实现光控电路的相关资料。
	下午	编写完高斯滤波代码和 sobel 边缘检测代码。
7.6	完成超声波模块以及数码管显示模块的代码实现以及调试。 编写通过交线次数判定车牌上下边界的模块代码，编写通过蓝色检测判定车牌左右检测的模块代码。	
7.7	完成确定每个数字的边界的模块的相关代码，通过数字特征识别判断每个数字。 购买光控电路所需的相关器件如光敏电阻等。	
7.8	上午	提取图像灰度后在高斯滤波模块出现问题，对此进行调试，利用面包板搭电源管理电路并进行调试。
	下午	调好了高斯滤波模块和 Sobel 边缘检测，焊接电源管理电路并进行调试。
7.9	上午	调好通过交叉次数定上下边界的模块，调整各种参数尝试后发现通过蓝色检测车牌左右边界不好用，决定换方法判定左右边界；电源电路调试中发现焊接问题和芯片损坏，重新焊接电路。
	下午	尝试其他方法定左右边界，效果不好。 设计、搭建及调试光控电路。
7.10	上午	实现实用的左右边界检测方法。 焊接完成并调试好光控电路。

	下午	测验发现数字特征匹配法在边缘检测后的数字图像上不好用，换新方法实现数字识别。 进行了模拟电路部分和部分数字电路的验收。
7.11	上午	整合电路和 FPGA，形成一个系统。 尝试新的数字识别的方法，但是识别精度不够且识别同时识别 5 个车牌号时 FPGA 寄存器不够用，除法改用 IP 核后还是寄存器数量不够。
	下午	由于 FPGA 板本身的资源配置不足问题，最终放弃最后一步的数字识别。 调试整合好的代码，完成全部验收。 上交 FPGA 板和面包板以及元器件，拍摄课程设计视频。
7.12	上午	答辩交流。
	下午	写报告。