第三届

全国大学生集成电路创新创业大赛

CICIEC

ARM 杯（项目设计报告）

参赛题目：ARM 片上系统设计挑战赛

队伍编号：BHB88496

团队名称：倔强青年

目录

1.系统总体框架设计

本小组基于 Altera 平台的 DE1 开发板实现了搭载有 Cortex-M3 软核的车牌实时识别系统。系统框图如图 1 所示，使用 Arm Cortex-M3 DesignStart Eval 提供的处理器 IP，在 DE1 上搭建简单的 Cortex-M3 片上系统，并基于 Cortex-M3 的 AHB 总线协议将 LED 灯、拨码开关、LCD1602、硬件图像加速模块等外设挂载至 AHB 总线。在 Cortex-M3 通过 Debug 端口连接至 PC 端，通过 kei15 软件可对其进行程序的烧写以及在线的调试。Cortex-M3 上的两组 IO 口均挂载至 AHB 总线，分别用于控制 8 盏 LED 灯、8 个拨码开关，以及 LCD1602 显示屏。

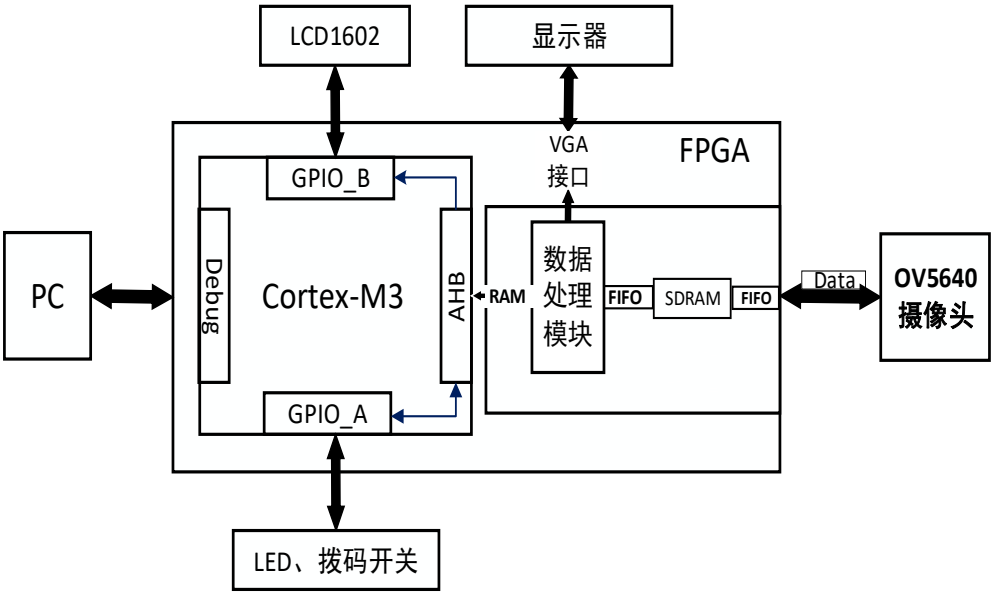

图 1.系统框图

图像硬件加速模块通过 IIC 协议配置 OV5640 摄像头，并接受 OV5640 摄像头输入的数字信号。通过写 FiFo 模块将图像信息写入 SDRAM 进行存储。数据处理模块则通过读 FiFo 模块读取 SDRAM 中的图像信息。经数据处理模块后，将视频信号输出至 VGA 接口供液晶显示器显示。同时，将车牌识别的结果写入存储空间为 400bits 的外部 RAM。将外部 RAM 的 20 位输出口作为 Cortex-M3 的外设，挂载至 AHB 总线，实现 Cortex-M3 软核对车牌数据的接收。

此系统最终实现的功能以及性能指标有，通过 PC 端对 Cortex-M3 软核进行在线调试以及程序的烧写，实现通过拨码开关控制 8 盏 LED 灯的闪烁模式。其次可动态识别蓝底白字的车牌，并通过 LCD1602 模块显示其识别结果。经多次

试验与测试，20 张车牌的平均识别所需时间大约为 14.8 秒，识别正确率超过 96%。实现了大赛题目所需的所有性能指标。

2.Cortex-M3 软核及其外设的设计与搭建

2.1 Cortex-M3 处理器的结构

Cortex-M3 微处理器的内部结构，如图 2 所示。ARM Cortex-M3 微处理器包括嵌套向量中断控制器（NVIC）、调试子系统、唤醒中断控制器、AHB-Lite 总线接口以及连接这些单元的内部总线系统。

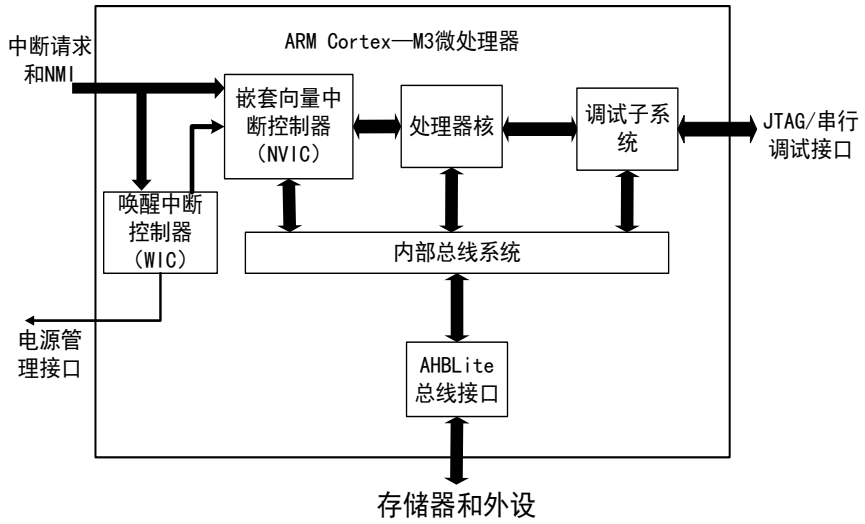

图 2.Cortex-M3 内部结构

1）处理器核

处理器核是 Cortex-M3 最核心的功能部件，它负责对数据进行处理。该处理器包含内部寄存器、算术逻辑单元（ALU）、数据通路和控制逻辑。处理器内部用于取指、译码和执行指令的指令通道采用三级流水结构，如图 3 所示。三级流水线结构显著提高了处理器指令通道的吞吐量和运行效率。

图 3.指令通道三级流水线结构

2）总线系统

总线系统用于将 Cortex-M3 内部的各个功能部件连接在一起。总线系统包含：

① 内部总线系统。

② 处理器核内部的数据通道。

③ AHB-Lite 接口单元

3）调试子系统

作为 Cortex-M3 处理器重要的一部分，调试子系统提供下面的功能：

① 管理调试控制、程序断点，以及数据监控点。

② 当产生调试事件时，它将处理器核设置为停止状态。此时可以在该点分析处理器的状态，如寄存器值和标志。

2.2 AHB-Lite 总线

AHB-Lite 总线结构如图 4 所示，Cortex-M3 DesignStart 的 CPU 共引出 3 组 AHB-Lite 总线主机信号：ICODE_BUS、DCODE_BUS 与 SYS_BUS。通常情况下，CPU 使用 ICODE_BUS 与 DCODE_BUS 总线访问 ROM 来读取指令与常数，同时使用 SYS_BUS 总线来访问 RAM 和外设。

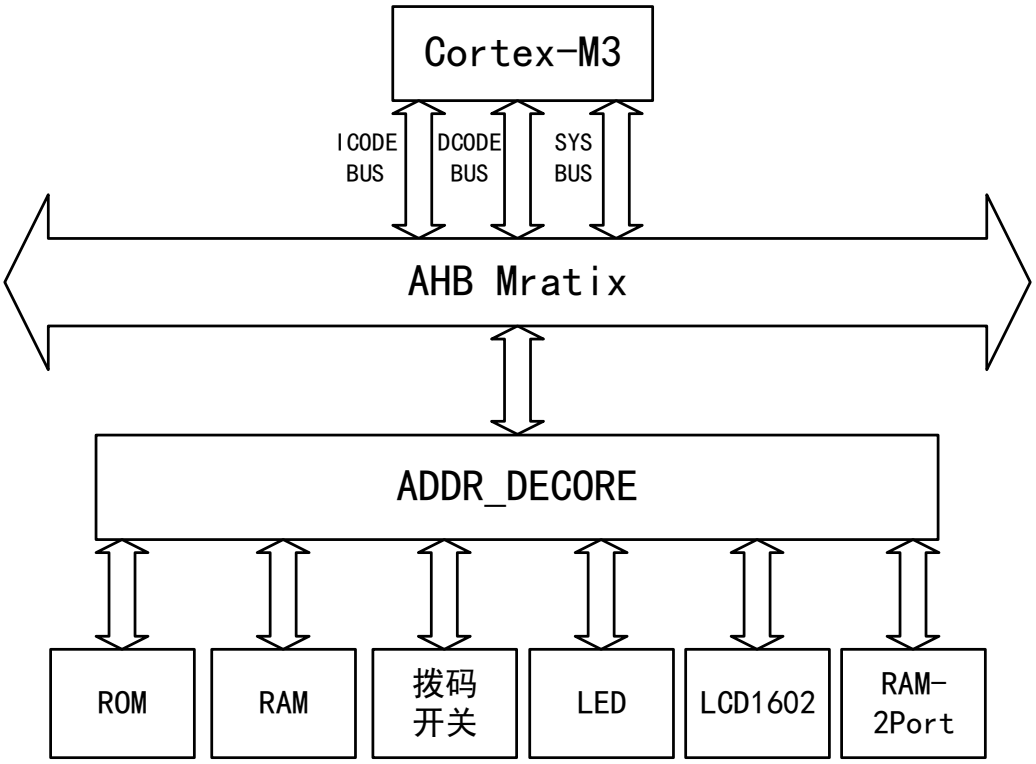

图 4.AHB-Lite 总线结构

ICODE_BUS 和 DCODE_BUS 总线在 CPU 内部已经做好了仲裁，这两个总线不会同时发出访问请求。顶层程序中只需要判断 htransd[1]信号并进行简单的数据选择即可。SYS_BUS 总线连接至包括 RAM 和各外设的多个从机，需要根据 HADDR 地址信号来生成每个从机的 HSEL 信号。Cortex-M3 DesignStart 通过 DAP 模块实现了调试与数据跟踪功能，需要处理 Serial Wire Debug（SWD）信号与 JTAG 信号的引脚复用。

2.2.1 主从设备关系

在 AHB-Lite 主设备需要提供地址和控制信息，用于初始化读和写操作。然后主设备接收来自从设备的响应信息，包括数据、准备信号和响应信号。如图 5 所示，HADDR[31:0]由主设备指向从设备以及译码器 Decoder，经译码器根据不同地址的值进行译码后，产生对应的从器件 HSEL 选择信号。其中 HWDATA[31:0]由主设备指向从设备，作为写数据总线，用于在写操作周期内将数据从主设备发送到从设备。HRDATA[31:0]由多路复用器指向主设备，将从器件所发出的数据传输至处理器。

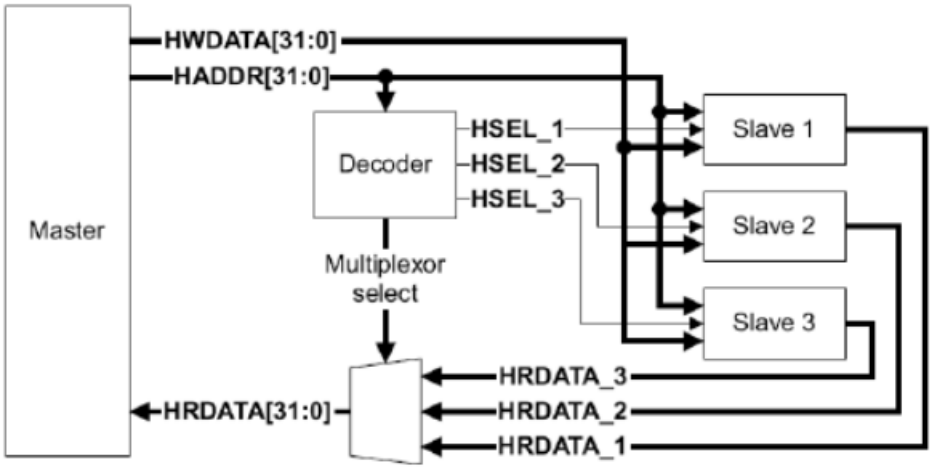

图 5.AHB-Lite 主从关系图

依据 AHB-Lite 主从关系，在 AHB 总线上挂载 LED、拨码开关、LCD、ROM，以及外部 RAM 等外设。并根据如图 6 所示的地址表，对外设进行地址的分配。在基于 Verilog 语言编写的 Decoder 模块中配置各个从器件的首地址。综合得到如图 7 所示的总体 RTL 视图。

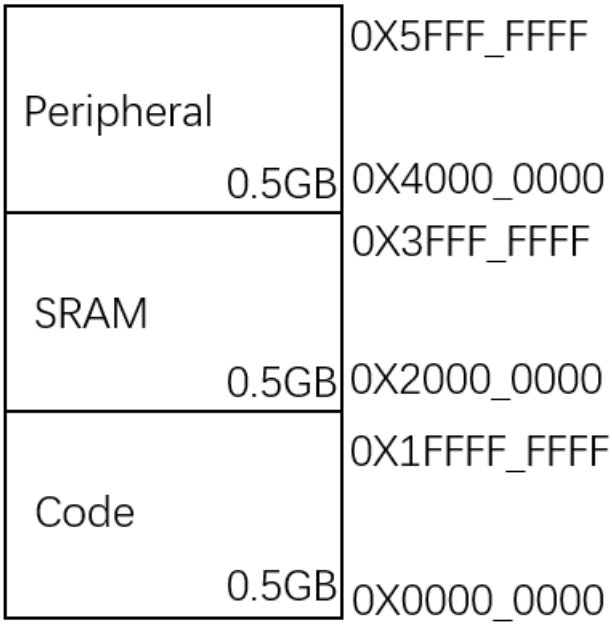

图 6.Cortex m3 内核的存储器部分映射地址

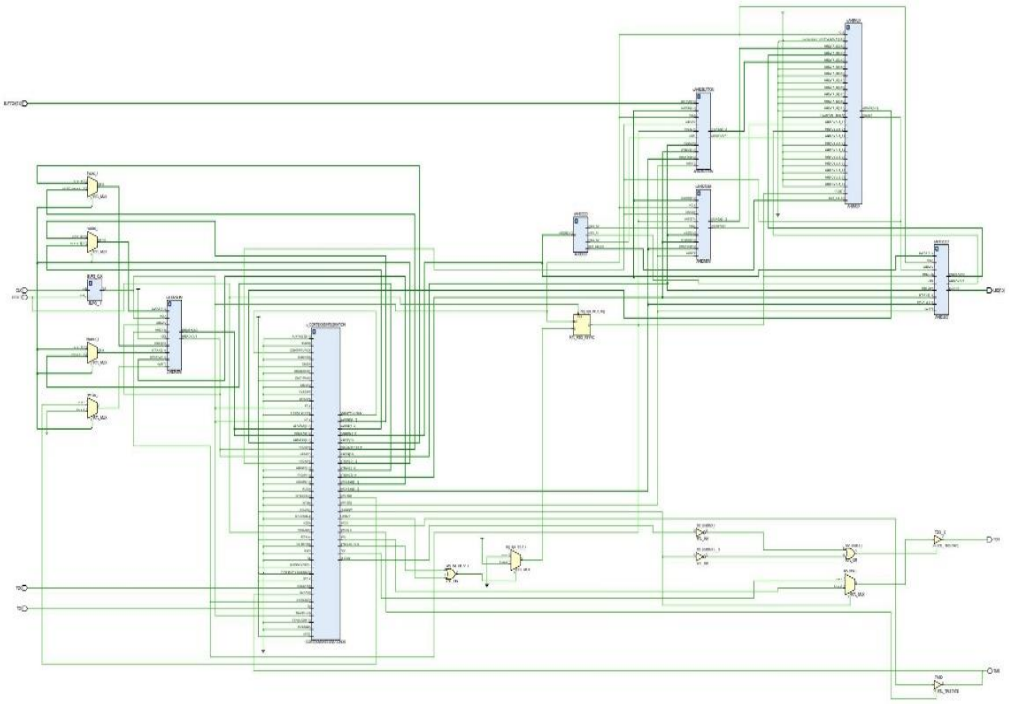

图 7. 系统 RTL 视图

2.2.2 AHB-Lite 读写时序

AHB-Lite 传输包括两个阶段，分别为第一时钟周期的地址阶段和第二个时钟周期的数据阶段。地址阶段只持续一个 HCLK 周期，而数据阶段则可为数个 HCLK 周期。使用 HREADY 信号来控制完成传输所需要的周期数。

1）无等待的基本读时序

在地址阶段中，主设备给出地址和控制信号，并将 HWRITE 设置为 0。在数据阶段，从设备将主设备所要读取的数据放置在 HRDATA 总线上。在读传输过程中，没有等待状态。如图 8 中，没有插入等待状态，表示从设备可以持续提供读取的数据，HREADY 信号持续有效。

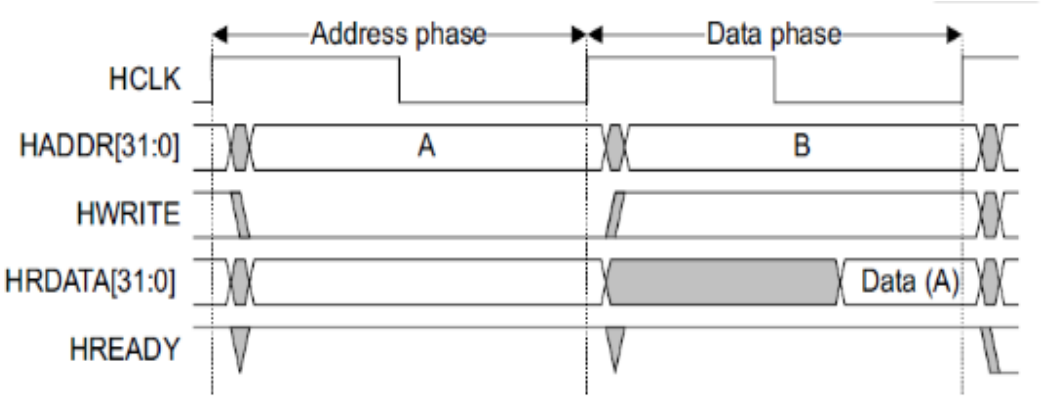

图 8.无等待的基本读时序

2）无等待的基本写时序

无等待的基本写时序如图 9 所示，在地址阶段给出地址和控制信号，将 HWRITE 设置为 1。在数据阶段，主设备将要写到从设备的数据放到 HWDATA 上。在无等待的写传输过程中，没有等待信号。也就是说，没有插入等待状态，表示从设备可以持续接收数据。

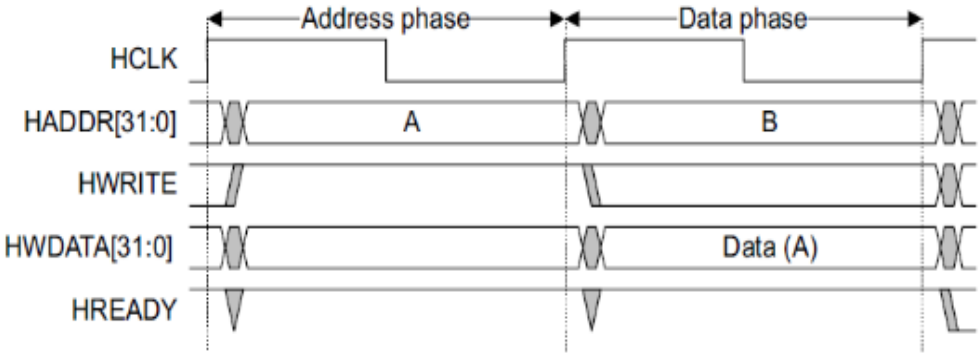

图 9.无等待的基本写时序

2.3 Cortex-M3 系统外设

2.3.1 拨码开关状态读取

为通过拨码开关的值来控制 LED 灯的工作状态，设置拨码开关为 00、01、10、11 这四个值时对应不同的控制子程序，分别对应 8 盏 LED 的全亮状态、向左流水状态、向右流水状态，以及整体闪烁状态。

拨码开关接口 RTL 视图如 10 所示，位宽为 8bits 的 BUTTON 信号线用于输入拨码开关当前的状态。由 HRDATA[31:0]将数据传输给处理器。

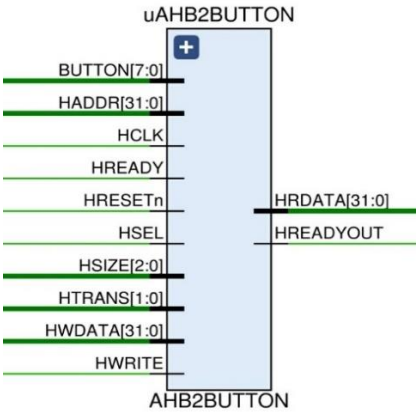

图 10.拨码开关接口 RTL 视图

根据 C 语言所编写的程序，当 BUTTON 拨码开关的值为 8'h01 时，经译码器译码后选中地址为 32'h5100_0000 的拨码开关外设。此后在下一个周期拉高读使能信号，将数据总线 HRDATA_BUTTON 的值变为 32'h0000_0001。如图 11 所示的仿真结果与设计情况一致，从而验证 BUTTON 模块功能正常。

图 11.BUTTON、LED 模块仿真时序图

2.3.2 LED 的闪烁模式控制

LED 接口 RTL 视图如图 12 所示，位宽为 8bits 的 LED 信号线用于输出 LED 灯的当前状态。

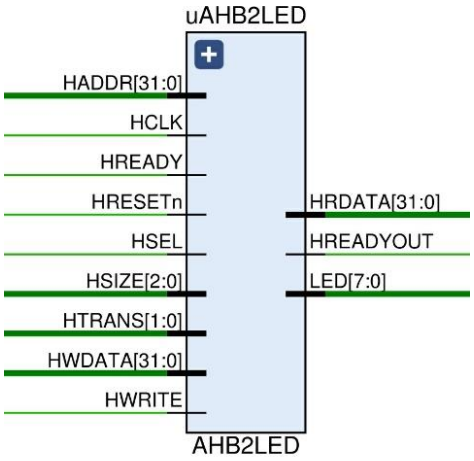

图 12. LED 接口 RTL 视图

在 Keil5 软件中对 4 种不同的 LED 工作模式进行编程，其中两种工作状态程序如图 13、图 14 所示，分别为 LED 灯全亮状态和 LED 向左循环流水闪烁。

```
void mode0()/*mode0:AHB_BUTTON == 0x00*/
{
    *(unsigned int*) AHB_LED_BASE = 0xff;
}
```

图 13. LED 灯全亮工作状态

```
void mode1()/*mode1:AHB_BUTTON == 0x01*/
{
    unsigned char cnt = 0;
    *(unsigned int*) AHB_LED_BASE = 0x01;
    delay(2000);
    for(;cnt<=7;cnt++)
    {
        *(unsigned int*) AHB_LED_BASE = *(unsigned int*) AHB_LED_BASE << 1 ;
        delay(2000);
    }
    cnt = 0;
}
```

图 14. LED 灯循环流水工作程序

当 BUTTON 值为 8'h00 时，在设计 LED 模块时将 BUTTON 拨码开关的值设置为 8'h00，在下一个周期将数据总线 HRDATA 的值设为 32'h0000_0000，数据总线经

RAM 模块处理后将总线 HRDATA 的值设为 32’h0000_00FF，一段时间后地址总线经译码器选中 32’h5000_0000 即 LED 模块，在下一个周期拉高 rHWRITE，并读取数据总线数据 32’h0000_00FF，在下一个周期将读取的数据低 8 位传给 LED。仿真结果如图 15 与理论设计一致。实现的功能为当拨码开关的值为 8’ h00 时，8 个 LED 灯全亮。

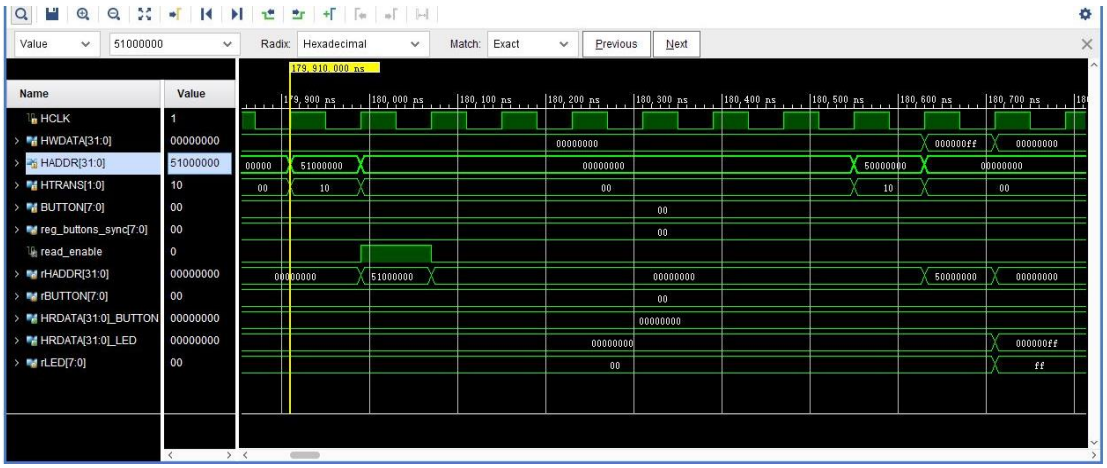

图 15.LED 全亮工作模式仿真图

当 BUTTON 为 8’h01，在设计 LED 模块时将 BUTTON 拨码开关的值设置为 8’h01，在下一个周期将数据总线 HRDATA 的值设为 32’h0000_0001，数据总线经 RAM 模块处理后将总线 HRDATA 的值设为 32’h0000_0001，并将数据总线 HWDATA[31:0]上的数据经每隔 131.36ms 左移一位，一段时间后地址总线经译码器选中 32’h5000_0000 即 LED 模块，在下一个周期拉高 rHWRITE，并读取数据总线数据，在下一个周期将数据低 8 位传给 LED。仿真结果如图 11，图 16 所示与理论设计一致，实现 8 盏 LED 灯的流水闪烁。

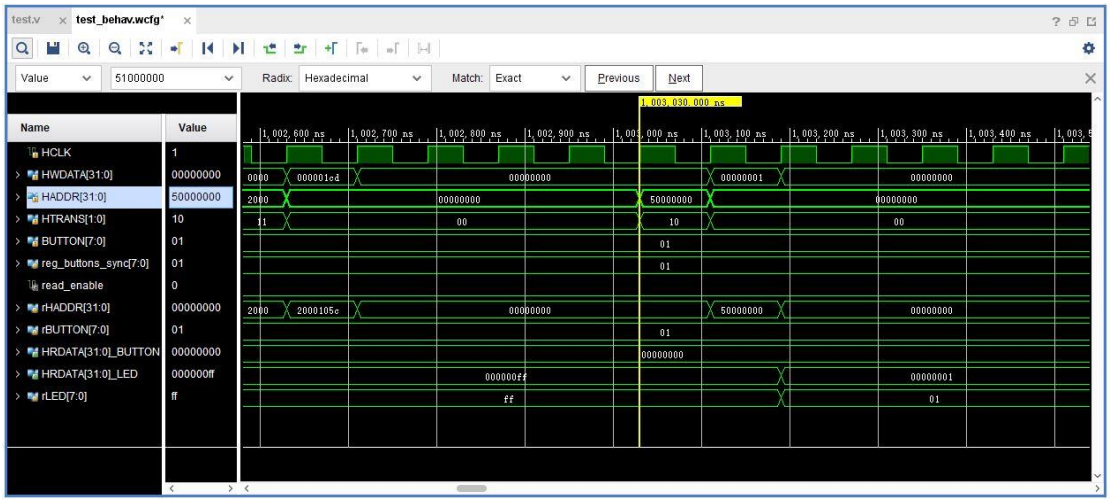

图 16. LED 灯循环流水工作模式仿真图

2.3.3 外部 RAM 的访问

由于硬件加速模块所生成的 20 个车牌数据结果为 400 位的二进制数据，但 AHB 总线位宽为 32 位，故需要对数据的读取进行位宽的设置。设置为 400bits 的存储容量，传输位宽为 20bits，即每传输一次对应一张车牌的 5 个数字。外部 RAM 的 RTL 视图如图 17 所示。

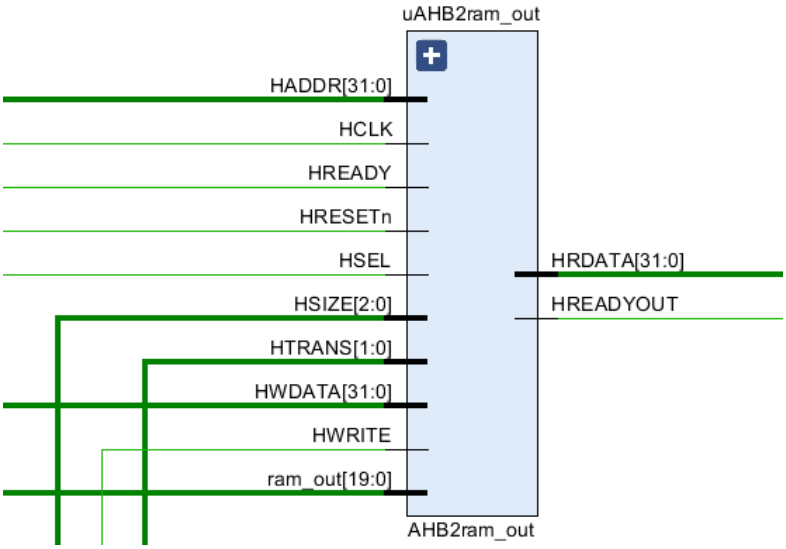

图 17. 外部 RAM 接口 RTL 视图

在 Quartus 软件中调用外部 RAM 的 IP 核，IP 核模块如图 18 所示。通过 5bits 位宽的地址线对其中的 20 个车牌数据进行寻址，从而控制数据的写入与输出。

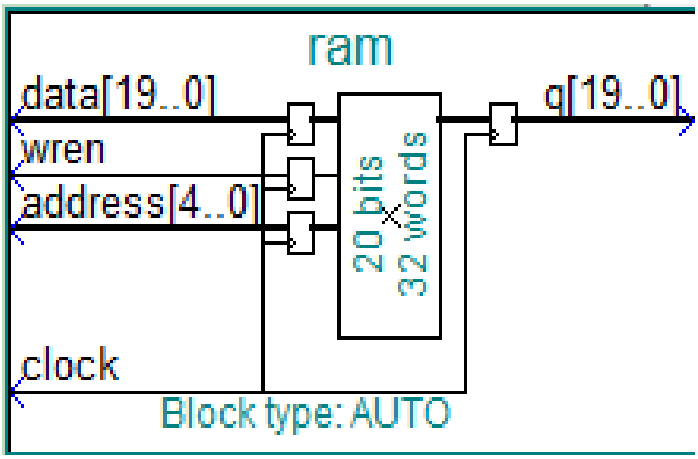

图 18. 外部 RAM IP 核模块

其中 data[19:0]以及 address[4:0]两个信号均挂载至 AHB-Lite 总线，通过 AHB 总线依次读取 20 次位宽为 20bits 的车牌识别结果。

2.3.4 LCD1602 显示

通过 kei15 软件对 LCD1602 模块进行编程。实现每幅 LCD1602 的页面显示 4 个车牌号码，共显示 5 幅页面，实际显示效果如图 19 所示。

图 19.LCD 显示效果图

由于 LCD1602 属于低速器件，故在代码设计时，加入延时子函数，防止上一次写操作未完成就进行下一次写操作。因此不必对 LCD1602 进行读操作，来判断其是否为忙状态。对于控制液晶对比度的 VL 管脚，使用一个 FPGA 的 GPIO 口单独对其控制，输出占空比为 50%的 PWM 方波信号，使其字符显示清晰。

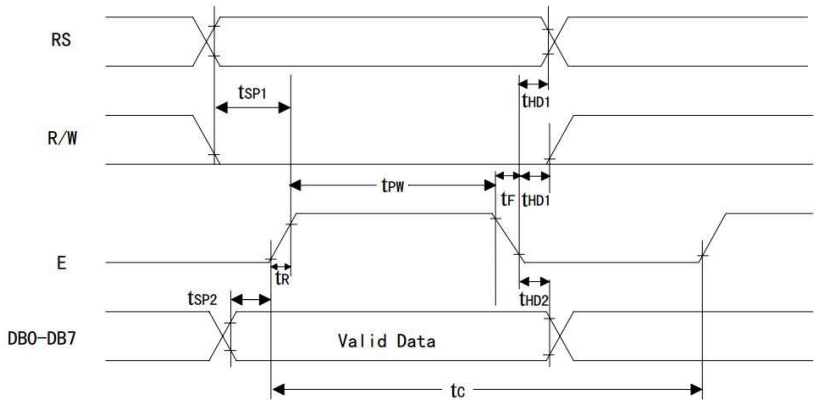

图 20.LCD1602 写时序

在 LCD 上电后，首先需要对其进行初始化操作。基础的控制指令如图 20 所示。LCD 初始化顺序为，显示模式设置、显示关闭、显示清屏、显示光标移动设置、显示开及光标设置。初始化完成后，即可等待 20 张车牌识别完成信号，进而对 20 幅车牌识别结果进行显示。

2.3.5 程序设计

系统程序框图如图 21 所示，系统上电复位后，进行 20 张车牌是否完成识别的判断，若 20 车牌未识别完成，立即对拨码开关的状态进行判断。程序中设置

了 4 种不同的状态模式，由最低位的两个拨码控制，其拨码值分别为 00、01、10、11，分别对应 8 盏 LED 的全亮状态、向左流水状态、向右流水状态，以及整体闪烁状态。

当检测到20张车牌识别完毕,此时开始读取写入到外部RAM中的车牌数据。一张车牌数据对应 20bits 的位宽大小，重复读取 20 次后，将 400 位的数据信息存入自定义的数组变量中，以便 LCD1602 显示使用。此后进入 LCD1602 的显示阶段，循环检测按键按下的次数，按键每按下一次，即显示下一副 LCD 页面。显示 5 幅页面显示完毕，识别结束。

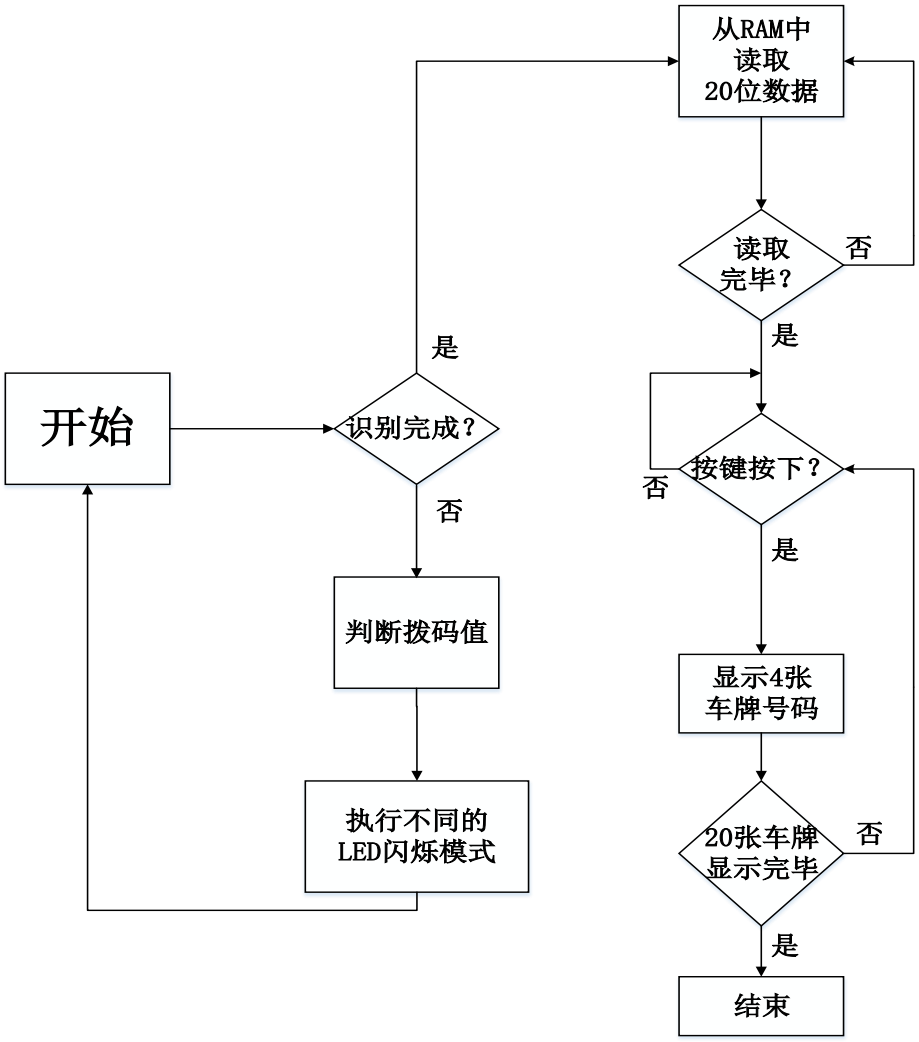

图 21.程序设计流程图

3.车牌识别硬件加速算法实现

本系统通过 Verilog 硬件描述语言在硬件平台实现车牌识别算法，通过 Verilog 的语法特点可以大大提升识别算法的执行速度。首先，本系统将车牌识别分为三个部分：图像数据采集、图像数据输出显示、车牌定位和数字识别算法。每个部分又分别由多个子模块组成，如图 22 所示为车牌识别部分的整体 RTL 图。首先通过 i2c_ov5640_rgb565_cfg 和 i2c_dri 模块配置 ov5640 的相关寄存器，并通过 cmos_capture_data 模块将摄像头输出的 8 位数据转换成 16 位的 RBG565 数据，并通过 sdram_top 模块将 RGB565 实时写入 sdram 中。通过 img_handle 模块读取 SDRAM 中的数据，以及对图像数据进行相关算法处理，最终通过 vga_driver 将处理后的图像数据输出到 VGA 显示器上。由于 SDRAM 的读写时钟和 VGA 驱动时钟均高于系统时钟且不同，所以本系统通过调用 PLL 的 IP 核来满足多个模块的时钟需求。

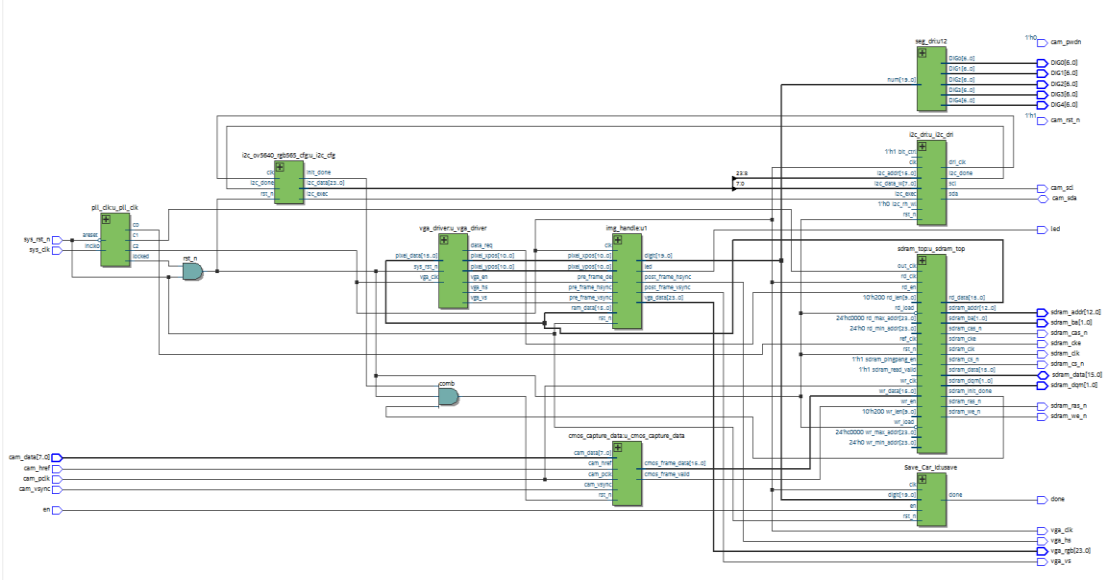

图 22. 整体 RTL 视图

3.1 图像数据的采集

首先，FPGA 通过 IIC 总线协议对 OV5640 摄像头的相关寄存器进行配置，使其在系统所需的模式下进行工作，之后摄像头对图像信息进行采集，将采集到的数据通过 FPGA 的 GPIO 输入到 FPGA 中，FPGA 通过 FIFO 将数据写入 SDRAM 中完成图像数据的采集和存储。

3.1.1 摄像头寄存器配置

OV5640 摄像头寄存器，通过 IIC 总线协议进行配置。本系统将摄像头寄存器配置分为两个模块，其中一个模块为摄像头内部寄存器地址的查找表，另一个模块为 IIC 总线控制器。通过 IIC 通信输入将指令输入摄像头，并对摄像头的工作模式以及各个参数进行配置。

3.1.2 摄像头数据处理

当摄像头寄存器全部配置完成后，先等待 10 帧数据，等待寄存器配置生效后再开始采集数据。并将摄像头的 8 位输出数据转换成 16 位的 RGB565 格式的图像数据。其核心代码如下：

```
always @(posedge cam_pclk or negedge rst_n) begin
    if(!rst_n) begin
        cmos_data_t <= 16'd0;
        cam_data_d0 <= 8'd0;
        byte_flag <= 1'b0;
    end
    else if(cam_href) begin         //摄像头行同步信号
        byte_flag <= ~byte_flag;
        cam_data_d0 <= cam_data;    //摄像头数据寄存
        if(byte_flag)
            cmos_data_t <= {cam_data_d0,cam_data};
        else;
    end
    else begin
        byte_flag <= 1'b0;
        cam_data_d0 <= 8'b0;
    end
end
```

3.1.3 SDRAM 存储模块

SDRAM 用于实时存储摄像头采集的数据，方便后续对图像数据进行处理。SDRAM 的读写实现主要由两个模块实现：SDRAM 的读写模块和 SDRAM 控制模块。其中读写模块例化了两个 FIFO 分别对 SDRAM 的数据进行读和写。而 SDRAM 控制模块用于对 SDRAM 的状态、SDRAM 的命令和 SDRAM 的数据读写进行控制。如图 23 所示为 SDRAM 顶层模块的 RTL 图。

其主要的接口有：

wr_clk ： 写端口 FIFO 时钟

wr_en ： 写端口 FIFO 使能

[15:0] wr_data ： 写端口 FIFO 数据

[23:0] wr_min_addr ： 写 SDRAM 的起始地址

[23:0] wr_max_addr ： 写 SDRAM 的结束地址

rd_clk ： 读端口 FIFO 时钟

rd_en ： 读端口 FIFO 使能

[15:0] rd_data ： 读端口 FIFO 数据

[23:0] rd_min_addr ： 读 SDRAM 的起始地址

[23:0] rd_max_addr ： 读 SDRAM 的结束地址

sdram_read_valid ： SDRAM 读使能

sdram_pingpang_en ： SDRAM 乒乓操作使能

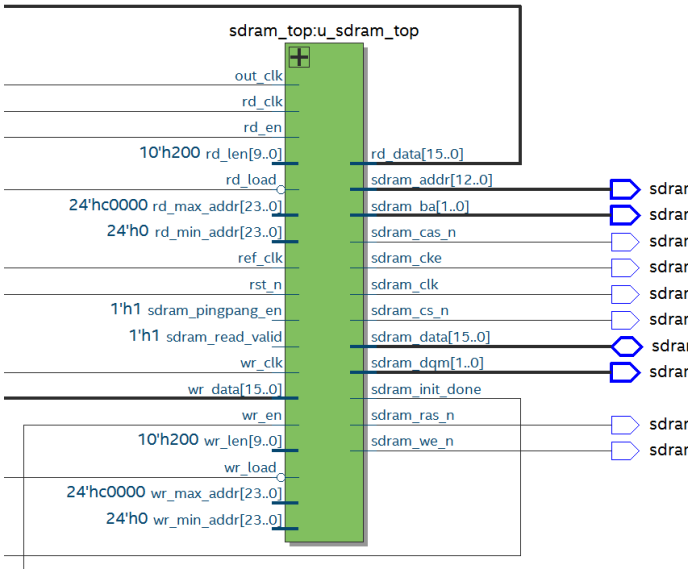

图 23. SDRAM 顶层模块 RTL 图

另外，为了防止读取和写入 SDRAM 时，读取 SDRAM 的速度低于 SDRAM 的写入速度导致两帧画面同时重叠出现在一帧画面中，所以对 SDRAM 进行了一个乒乓操作，将 SDRAM 分为两个 bank，一个 bank 进行写操作，一个 bank 进行读操作，具体原理见 3.3.1 节。

3.2 车牌识别算法

车牌的识别算法主要包括两个部分：车牌定位算法和数字识别算法。首先我们通过 RGB 转灰度、图像二值化、中值滤波、垂直投影和水平投影等多个模块将车牌的具体位置锁定，然后通过字符分割和字符识别模块将车牌的数字识别出来，并通过数码管将识别到的数字动态的显示出来。

3.2.1 图像灰度处理

首先，通过 FIFO 读取 SDRAM 中存取的每一帧图像的 RGB565 值，然后通过对颜色分量高位补低位的算法将 RGB565 的图像数据格式转换为 RGB888 的数据格式。RGB888_R[7:0]={RGB565_R[4:0]RGB565_R[4:2]};

RGB888_G[7:0]={RGB565_G[5:0],RGB565_G[5:4]};

RGB888_B[7:0]={RGB565_B[4:0],RGB565_B[4:2]};

将 RGB565 转换成 RGB888，再将 RGB888 转为 YCbCr。在 YCbCr 中，Y 表示亮度值（灰度值），Cb 表示蓝色色度分量，Cr 表示红色色度分量。其与 RGB 色彩空间可以用以下公式转换：

Y = 0.299R + 0.587G + 0.114B

Cb = -0.1687R-0.3313G+0.5B+128

Cr=0.5R-0.4187G-0.0813B+128

3.2.2 基于灰度图像的二值化处理

由灰度处理模块得到图像的灰度值以后，图像数据由原来 RGB888 格式的 24 位数据转换为 8 位的灰度数据，数值由 0-255 分别表示由黑色到白色的图像灰度值。其中车牌蓝底的 RGB888 值为{6,0,134}，对应的灰度值为 17；车牌孔洞的 RGB888 的值为{187,173,113}，对应的灰度值为 171；车牌数字的 RGB888 值为{255,255,255}，对应的灰度值为 255。根据公式，最佳阈值=（最大灰度值-（最大灰度值-最小灰度值）/3），进而计算出适合车牌进行二值化处理的最佳阈值为

175。所以当像素点的灰度值大于 175 时，将灰度数据赋为 0，视为黑色；当像素点的灰度值小于 175 时，将灰度值赋为 255，视为白色。通过此全局阈值，可以很好的将车牌的孔洞滤去，只留下车牌号和车牌背景区域。如图 24 所示，为二值化处理后的图像。

图 24.二值化效果图

3.2.3 中值滤波

从二值化后的图像，我们不难看出图像存在椒盐噪声，所以本系统采用中值滤波去除椒盐噪。所谓中值滤波，即对以本像素点为中心得 3*3 矩阵进行排序，用排序中值取代中心点的像素值。同时，为了简化模块在读取数据时对 RAM 地址的计算并且充分利用 FPGA 的并行特性，对图像数据进行顺序读取，需要将逐行顺序读取的数据转化成 3 行的并行数据进行处理。在此我们巧妙的使用 Quartus 提供的移位寄存器 IP 核来实现串行数据的并行化。移位寄存器的原理如图 25 所示，移位寄存器就是由多个 FIFO 进行实现，每个通道即为一个 FIFO，数据填满三个 FIFO 后输出的数据即为并行数据。当数据在三个 FIFO 全部填满时，三个通道数据输出，由于本系统采用 1024*768 分辨率的 VGA 显示，所以每个 FIFO 的容量应为 1024，每个数据为 8 位。由于我们选用的 3*3 矩阵进行中值滤波，所以需要定义 9 个 reg 型变量对数据进行暂存，并利用以下中值计算核心代码对 9 个 reg 型变量进行排序操作。

```
assign max = (A1>=A2)?((A1>=A3)?A1:A3):((A2>=A3)?A2:A3);
assign min = (A1<=A2)?((A1<=A3)?A1:A3):((A2<=A3)?A2:A3);
```

```
assign middle
=((A1<=A2)&&(A1>=A3))||((A1>=A2)&&(A1<=A3))?A1:(((A2<=A1)&&(A2>=A3))|
|((A2>=A1)&&(A2<=A3))?A2:A3)
```

通过上述代码，对三个值进行比较，计算出其最大、最小及中值，而中值计算模块由 7 个上述子模块组合而成，通过子模块的组合最终实现模块中值的计算。

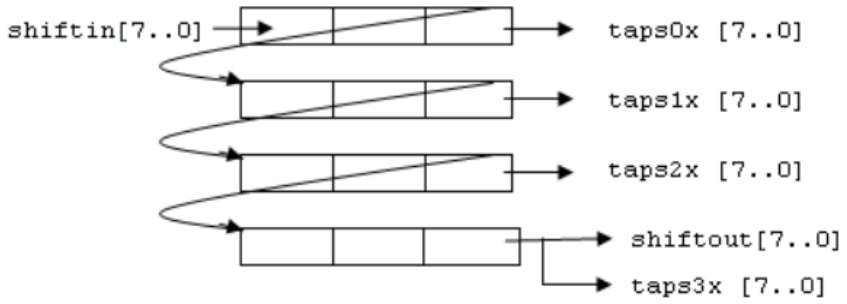

图 25.移位寄存器工作原理

通过中值滤波后，二值化图像如图 26 所示，与之前没有进行中值滤波的二值图像进行对比，可以明显的看出椒盐噪声得到了很好的改善。

图 26.中值滤波后二值化效果图

3.2.4 车牌定位

在本系统中我们采用投影法来实现对车牌的定位，因为车牌是放在白色背景的 PPT 中，所以车牌周围环境比较单一，利用投影法可以很好的确定车牌的具体区域，投影法的原理图如图 27 所示。

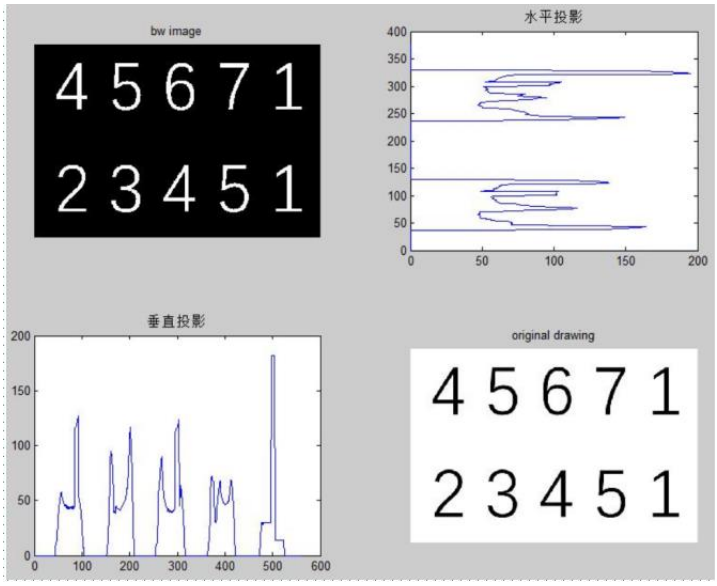

图 27. 投影操作原理图

投影法的具体实现方式如下：首先通过调用两个 ram 来分别存取一帧图像对应的列像素点数据和行像素点数据，由于本次系统采用的是 1024*768 的高分辨率 VGA 来显示图像，所以 ram 不对 8 位的图像灰度数据进行存储，而是通过 1 位数据代替，其中 1 表示像素点为黑色、0 表示像素点为白色，这样做的目的是为了节省存储空间，同时提高运算速度。而图像进过二值化后，PPT 的白色背景为黑色，车牌的底色为白色，所以白色区域即为目标区域。因此，每次都通过三帧图像来完成一次投影，并通过状态机来实现每一帧图像的对应操作，其中第一帧图像用于对两个 ram 进行初始化，两个 ram 的所有位都置为 0；第二帧图像用于存储坐标点的像素值，只有当坐标点为目标时才将两个 ram 对应的地址（坐标）的数据值写为 1；经过第二帧图像后，第三帧通过对行 ram 进行遍历。当行 ram 的前一个数据为白，后一个数据为黑即可断定此处为车牌的上边界；当行 ram 的前一个数据为黑，后一个数据为白即可断定此处为车牌的下边界；同理，通过对列 ram 的遍历同样也可以确定车牌的左右边界。所以，第三帧图像通过遍历第二帧图像处理后的 ram 进而可以确定车牌的上下左右边界。又因为车牌的背景部分会有一个白色的框会对车牌号的识别进行干扰，所以我们在确定车牌的具体区域后，将车牌的边界往中间缩小了一部分，这样做的目的是为了去除车牌边缘的白框对后续算法的影响。如图 28 所示，当车牌出现在蓝色框区域内时，绿色的框就能动态的确定车牌的位置。

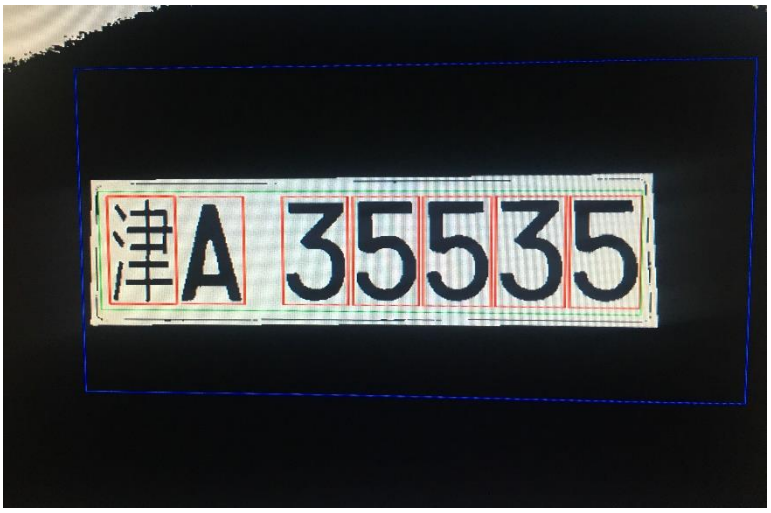

图 28. 车牌定位效果图

3.2.5 字符分割和识别

通过车牌定位后，可以准确的确定车牌的具体位置，在确定的车牌区域进行水平投影和垂直投影可以将车牌的每个字符很好的分割出来，然后通过特征识别算法将具体的数字值计算出来。

特征识别法的关键是找出字符的形状及结构等几何特征，通过统计字符特征来识别相应的字符。针对车牌的字体，为实现更加稳定的识别我们选取了三条线，分别位于数字行宽的 1/3 处、2/3 处，以及数字列宽的 1/2 处，并通过选取数字与行 1/3 划线的前沿交点、数字与行 2/3 划线的后沿交点、以及数字与列 1/2 处的后沿交点的位置和个数来确定数字的值。如图 29 所示是对字符 0 的特征识别统计，用一条纵线 y 和两条横线 x1、x2 穿过字符。y 在字符的 1/2 宽度处与 0 有两个交点；x1 在字符 1/3 高度处有两个交点而第一个交点的前沿位于 y 轴的左侧；第二个交点的前沿位于 y 轴的右侧；x2 在字符 2/3 高度处与 0 的第一个交点的后沿位于 y 轴的左侧，第二交点的后沿位于 y 轴的右侧。

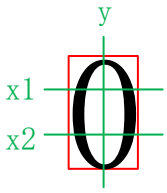

图 29. 算法原理图

通过此方法对其他字符与特征线的交点进行统计，并针对车牌的字体，我们做出了其中 Y 表示数字在列 1/2 处的交点个数，X1L 表示数字在行 1/3 处的左交

点个数，X1R 表示数字在行 1/3 处的右交点个数、X2L 表示数字在行 2/3 处的左交点个数、X2R 表示数字在 2/3 处的右交点个数。特征值为 6 位 2 进制数，高两位代表与 Y 的交点，低四位分别代表与两条 X 的交点情况。

数字	Y	X1L	X1R	X2L	X2R	特征值
0	2	1	1	1	1	6'b10_1_1_1_1
1	1	0	1	1	0	6'b01_1_0_1_0
2	3	0	1	1	0	6'b11_0_1_1_0
3	3	0	1	0	1	6'b11_0_1_0_1
4	2	0	1	1	0	6'b10_1_1_1_0
5	3	1	0	0	1	6'b11_1_0_0_1
6	3	1	0	1	1	6'b11_1_0_1_1
7	2	0	1	1	0	6'b10_0_1_1_0
8	3	1	1	1	1	6'b11_1_1_1_1
9	3	1	1	0	1	6'b11_1_1_0_1

表 1.数字特征值统计表

通过上述算法，可以很好的实现车牌的实时识别，如图 30 为车牌识别算法的效果图。

图 30.车牌识别效果图

3.3 图像数据的输出

首先通过 VGA 驱动程序将摄像头采集的数据从 SDRAM 中实时读取出来，并利用 SDRAM 的乒乓操作实现对数据实时的采集和读取。车牌识别的最终显示效果如图 31 所示。蓝色框为算法区域，当车牌出现在此区域，绿色框会动态的将车牌的位置框住，红色框会动态的将车牌字符的位置框住。

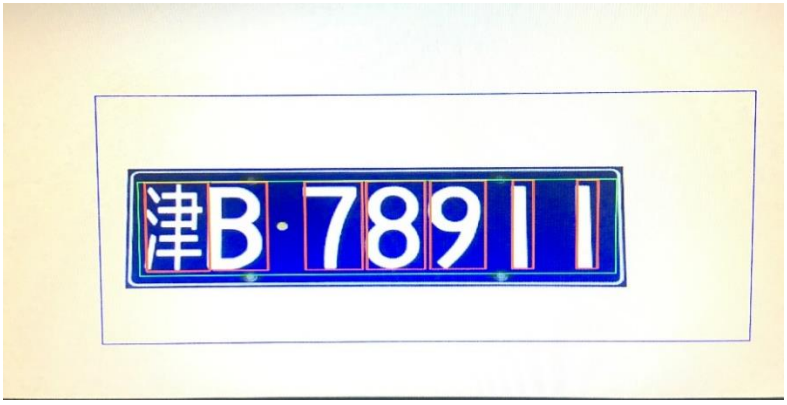

图 31. 车牌识别最终效果图

3.3.1 SDRAM 的乒乓操作

摄像头输入的视频源为 1024*768@84MHz，VGA 的驱动显示为 1024*768@65MHz，由于写 SDRAM 的速度和读 SDRAM 的速度不一致，所以可能会出现两帧重叠现象。为了达到显示的实时性和同步性，设计 SDRAM 里的两片 bank 来进行乒乓操作。如图 32 所示为乒乓操作的原理示意图。

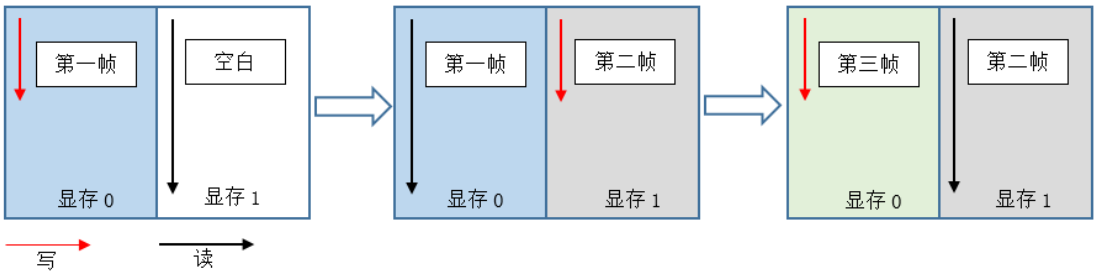

图 32. 乒乓操作的原理示意图

为了确保当前显示的一帧是完整的一帧，使用两个 bank 进行分别存取，在刚开始的时候摄像头给 banko 写入数据，写满 bank0 然后切换到 bank1 写数据，VGA 读取 bank1 的数据，读完一帧切换到 bank0 读取数据。这样基本上可以保证两帧交叠的概率降低。

4 系统仿真及测试

4.1 Cortex-M3 软核测试

如图 33 所示，在 Keil5 软件中打开 Cortex-M Target Driver Setup 界面，观察到在 SW Device 栏可以识别到 IDCODE 为 0x2BA01477 的器件，证明可以正确访问 Cortex-M3 器件。

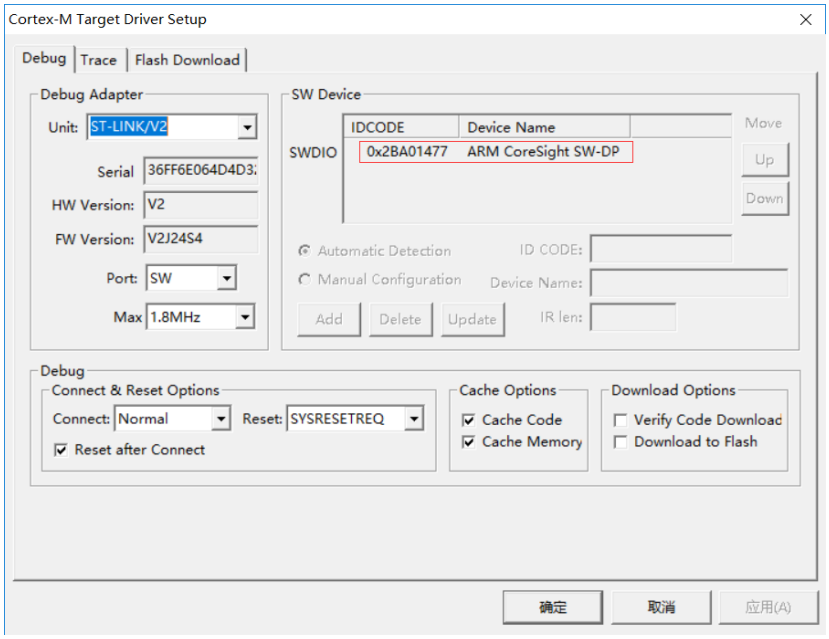

图 33.Keil 5 器件设置界面

进入 Debug 界面，对程序进行断点调试，以验证调试接口的功能。如图 34 所示，对 LED 的工作模式一进行断点测试，将 AHB_LED 以及 cnt 设置为观察对象，对它们的值进行跟踪。

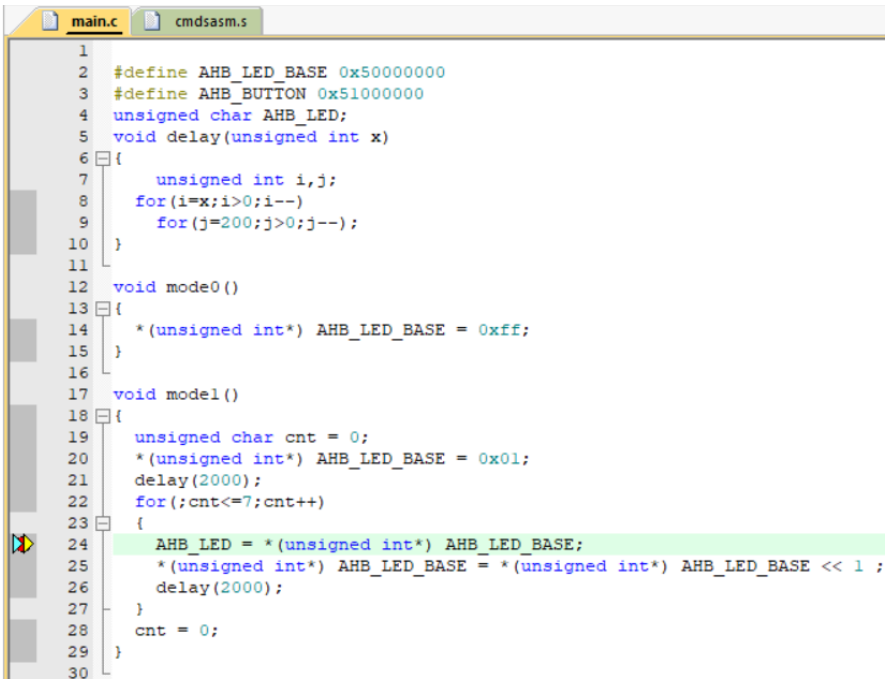

图 34. LED 模式中断点调试界面

点击运行按钮两次，分别观察两者的值并记录，如图 35 所示，cnt 自加 1，表明循环了一次，此时 LED 灯应该左移一位。由 AHB_LED 的可以看出，两次循环后 LED 的值均正确，证明了硬件电路以及软件程序的正确性。

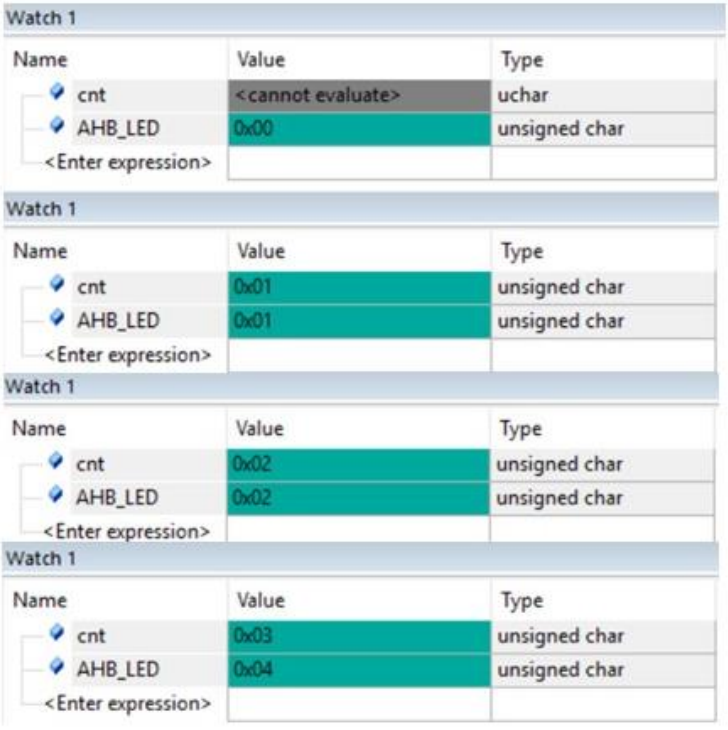

图 35. LED 参数断点调试结果

4.2 Cortex-M3 外设驱动仿真测试

当 BUTTON 拨码开关的值为 8'h01 时，经译码器译码后选中地址为 32'h5100_0000 的拨码开关外设。此后在下一个周期拉高读使能信号，将数据总线 HRDATA_BUTTON 的值变为 32'h0000_0001。如图 36 所示的仿真结果与设计情况一致，从而验证 BUTTON 模块功能正常。

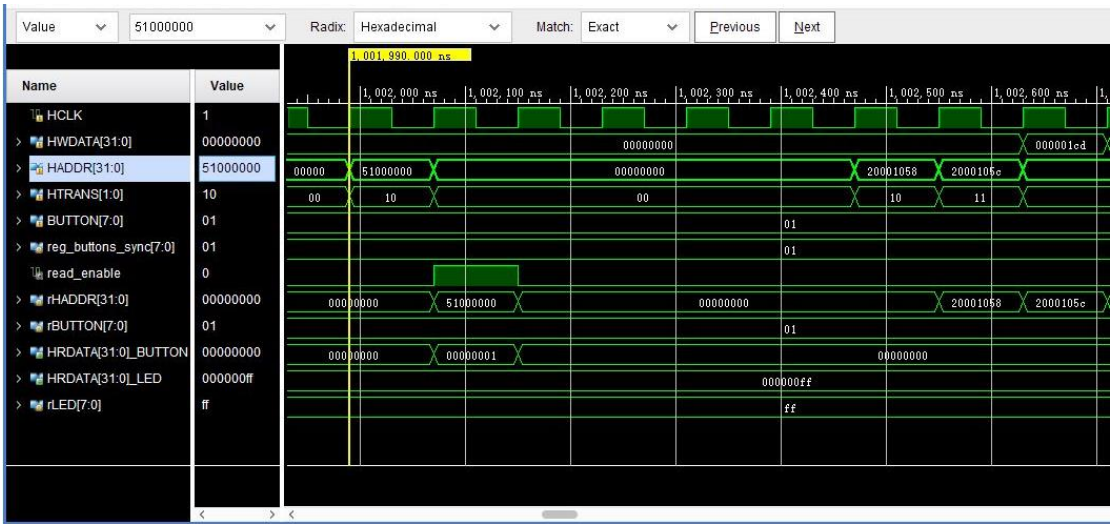

图 36.BUTTON 拨码开关仿真时序图

当 BUTTON 值为 8'h00 时，在设计 LED 模块时将 BUTTON 拨码开关的值设置为 8'h00，在下一个周期将数据总线 HRDATA 的值设为 32'h0000_0000，数据总线经 RAM 模块处理后将总线 HRDATA 的值设为 32'0000_00FF，一段时间后地址总线经译码器选中 32'h5000_0000 即 LED 模块，在下一个周期拉高读使能信号，并读取数据总线数据 32'h0000_00FF，在下一个周期将读取的数据低 8 位传给 LED。仿真结果如图 37 所示，与理论设计结果一致。

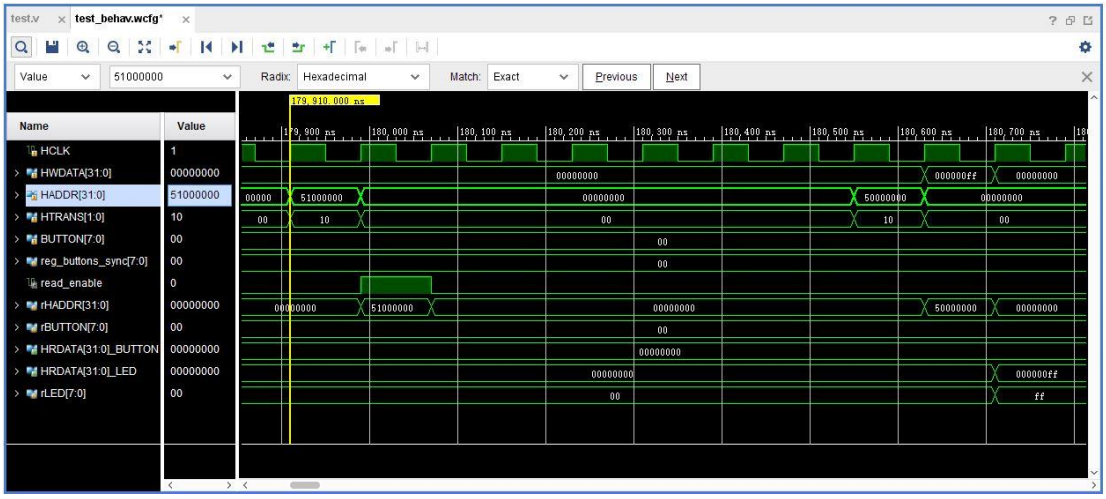

图 37.BUTTON 为 8'h00 时 LED 仿真时序图

当BUTTON为8'h01,在设计LED模块时将BUTTON拨码开关的值设置为8'h01，在下一个周期将数据总线 HRDATA 的值设为 32'h0000_0001，数据总线经 RAM 模块处理后将总线 HRDATA 的值设为 32'h0000_0001，并将数据总线 HWDATA[31:0]上的数据经每隔 131.36ms 左移一位，一段时间后地址总线经译码器选中 32'h5000_0000 即 LED 模块，在下一个周期拉高读使能信号，并读取数据总线数据，在下一个周期将数据低 8 位传给 LED。仿真结果如图 36、图 38 所示，且与理论设计一致。

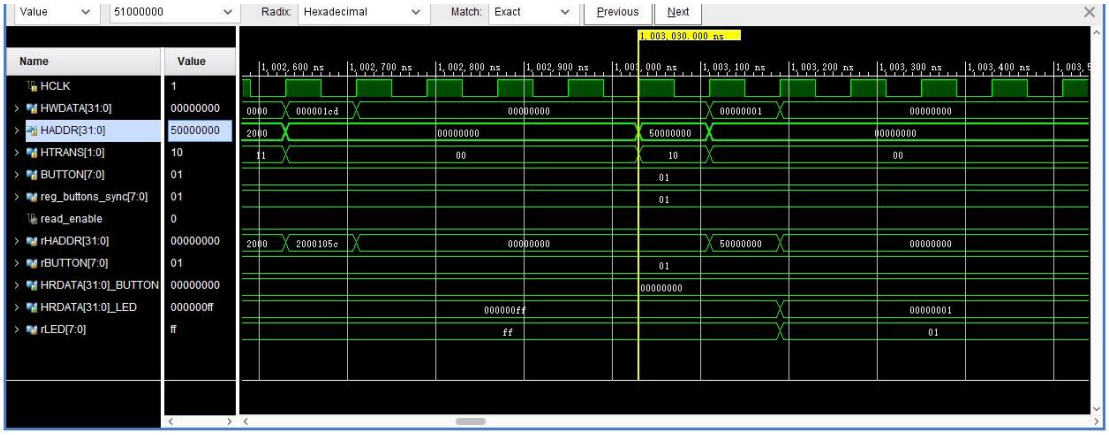

图 38.BUTTON 为 8'h01 时 LED 仿真时序图

4.3 车牌识别算法获取结果测试

搭建好测试的环境,预置 4 页存放车牌的 PPT,并设置 PPT 为自动播放方式。如图 39 所示为测试时通过 VGA 显示器观察到的车牌数字准确定位效果图。通过效果图可以发现，当车牌出现在蓝色框框中的区域时，绿色框会将车牌的所有字符准确的框住，红色框会准确的将车牌的单个字符分别准确框住。

图 39.车牌数字准确定位效果图

另外，每幅车牌的识别结果会动态显示在开发板的数码管中，如图 40 所示。通过验证可以发现 4 幅车牌识别完全正确。

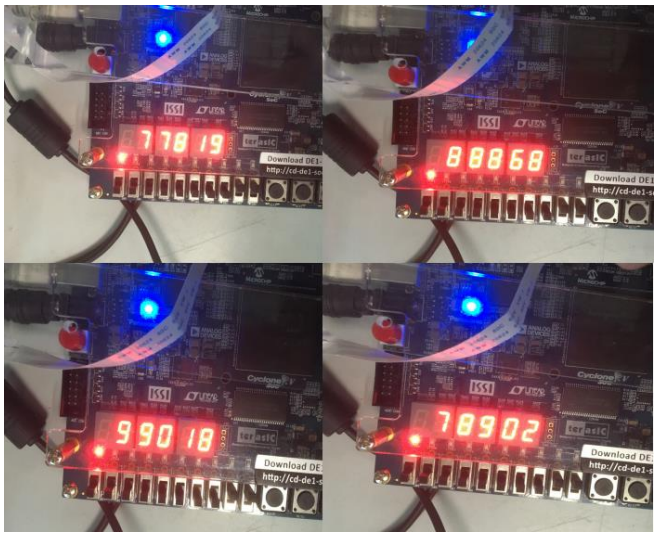

图 40.数码管动态显示结果图

最后，车牌识别完成后，需要将识别的结果传递至 Arm 软核，并由软核驱动 LCD 将识别结果显示出来。通过 Quartus 自带的 Signal Tap 工具，可以观察到数据可以稳定的传递至 Cortex-M3。最终，通过片上 Cortex-M3 驱动 LCD1602，并将识别结果显示出来。如图 41 所示，可以发现识别结果准确，数据读写准确。

图 41.LCD1602 显示识别结果图

4.4 系统整体测试

为测试系统识别车牌的正确率以及 20 张车牌识别速度，对系统进行 10 组实验，每组检测 20 张随机生成的车牌。并记录错误的个数，统计平均识别正确率。

经过多次重复测试，发现当 PPT 变换速度为 0.7 秒每幅图片时，系统的正确率和识别速度均可得到保障。

测试前，在 PPT 中预置 24 张车牌，并设置 PPT 以 0.5 秒每幅图片的速度播放，20 张车牌识别完毕将用时 10 秒。进行 50 次整体性能测试，记录测试数据，并计算出正确率。测试结果如表 2 所示。

组次	识别车牌总数	错误个数	正确率
第一组	20 张	0	100%
第二组	20 张	1	95%
第三组	20 张	0	100%
第四组	20 张	0	100%
第五组	20 张	0	100%
第六组	20 张	1	95%
第七组	20 张	0	100%
第八组	20 张	0	100%
第九组	20 张	1	95%
第十组	20 张	0	100%
合计	200 张	3	98.5%

表 2.十组系统测试结果

由测试结果可知，当 PPT 变换速度为 0.5 秒一副图片时，每次测试识别正确率均在 95%以上，且平均正确率为 98.5%，实现了系统的总体性能目标。

参考文献：

[1]王一楠.基于 AMBA2.0 的 AHB Matrix 总线架构设计[D].西安理工大学,2018.

[2]郭君成.基于 AMBA 总线的 CAN 控制器的设计与验证[D].电子科技大学,2018.

[3]罗惠文,吴斌,尉志伟,叶甜春.AHB Matrix 互连总线 IP 的设计与实现[J].微电子学与计算机,2015,32(10):54-57.

[4]范肖飞.基于 FPGA 的车牌识别系统设计与实现[D].成都理工大学,2018.

[5]强书连.车牌识别系统中字符识别算法的 FPGA 实现[D].福州大学,2017.

[6]赵亮,刘鹏,王晓曼,刘美.基于 FPGA 快速中值滤波算法的硬件实现[J].长春理工大学学报(自然科学版),2018,41(05):97-100+115.

[7]张雨沐.基于 FPGA 的视频图像处理系统的设计与实现[D].华南理工大学,2017.

[8]罗鑫. 基于 ARM 的嵌入式运动控制系统的研究[D].上海交通大学,2010.

[9]陶友龙,赵安璞,陈海波.基于 Cortex-M3 核的 SoC 架构设计及性能分析[J].电子技术应用,2012,38(08):53-55.

[10]张松. 兼容 Cortex-M3 指令集嵌入式微处理器设计[D].南京理工大学,2013.