摘　　要

本报告介绍的实验利用Micro SD卡和VGA显示屏，使用Nexys4DDR开发板实现了图片数字识别功能。项目包括六个子系统，十个子模块包括SD卡初始化与读写，VGA显示，数码管显示，相关数据缓存与处理等功能模块。本报告阐述了实验过程中遇到的问题与解决办法，以及一些笔者课程学习的心得体会。

目　　录

1 实验内容

1.1 项目简述

本实验是基于 FPGA 以及 SD 卡的印刷体数字识别实验。

1.2 项目说明

将一些含有单个数字的图片文件（如图 1）转化为 Bin 文件（如图 2）存入 SD 卡（如图 3），通过 FPGA 将其依次显示在 VGA 上（如图 4），并对图片中**含有数字的区域进行框取并对数字进行识别显示在数码管上**。

图 1

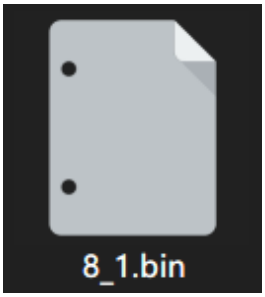


图 2

图 3

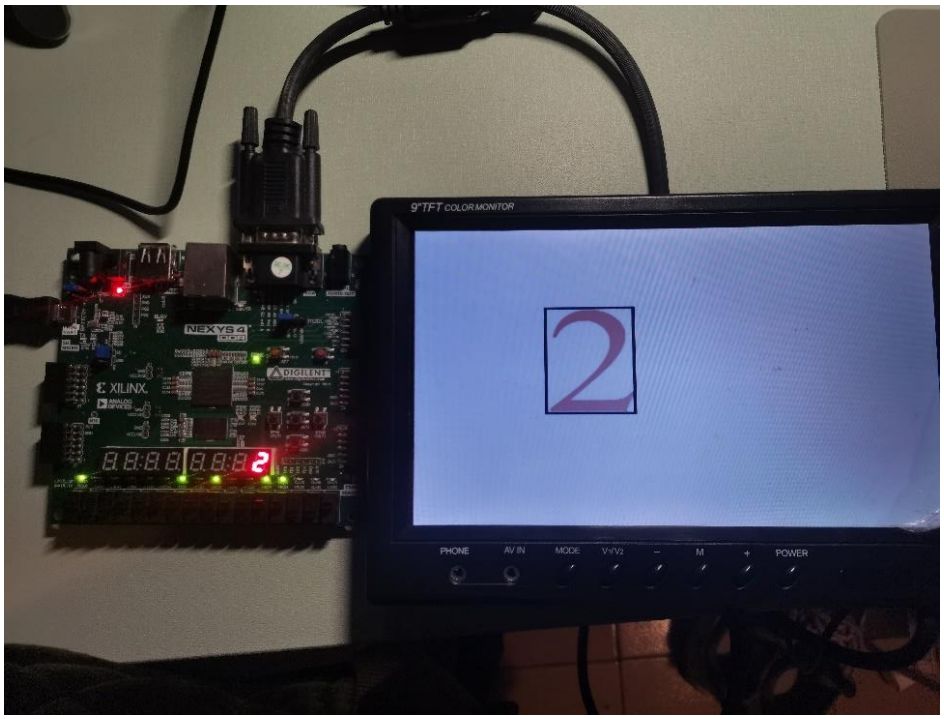


图 4

1.3 环境

1.3.1 硬件与外围模块

开发板：Nexys4 DDR™ FPGA Board Reference Manual

VGA：TFT LCD COLOR MONITOR

SD 卡：MICRO SD CRAD (SD2.0)

1.3.2 开发环境

开发工具：VIVADO 2016.2

1.3.3 软件工具

图片转换工具：Img2Lcd

磁盘读取工具：WinHex64

2 SD 卡读取数字图片识别系统

采用“自顶向下”的设计方式，SD 卡读取数字图片识别系统包括一个控制器子系统，时钟子系统、VGA 输出子系统、数码管输出子系统、图片读取控制系统、SD 卡输入子系统、数字识别子系统等 6 个子系统。

2.1 子系统框图

子系统框图

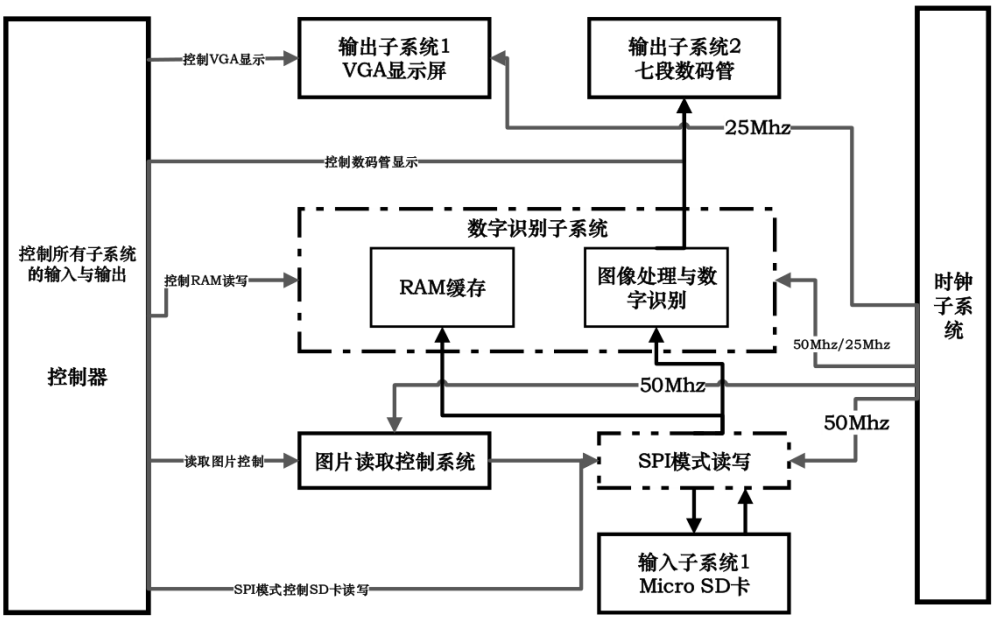


图 5

2.2 子系统功能简介

2.2.1 控制器子系统

控制器子系统控制各系统的输入输出与数据交互，包括 SD 卡的初始化与读取，RAM 的读写，控制图片数据的读取，控制数码管与 VGA 的显示

2.2.2 时钟子系统

利用系统时钟进行分配，分出两个时钟，50Mhz 与 25.175Mhz，一个用于 SD 卡的 SPI 模式读写，一个用于 VGA 的显示；同时，50Mhz 的时钟还用于图片读取控制系统；双端口 RAM 作为数据缓存写端口时钟使用的是 SPI 模式的 50Mhz 时钟，读端口使用的是 VGA 的 25.175Mhz 时钟。

2.2.3 VGA 输出子系统

通过读取 RAM 中的 12 位 RGB 像素数据将图片显示在 VGA 显示屏上，并且将在数字识别子系统中识别出的数字边框显示在显示屏上。

2.2.4 数码管输出子系统

通过数字识别子系统识别出的数字十进制数字获取信号，并通过七段数码管将该数字显示出来，还没有识别出来数字时或者识别出的数字发生错误时就不显示错误。

2.2.5 图片读取控制系统

给出 16 张图片在 SD 卡中的扇区地址，并且控制 SD 读取模块对 SD 卡的读取，每一次读取 512 个字节(即一个扇区)，读取一张图片（640*480*16/8/512）就需要读取 1200 次。每读取完一张图片则等待 2.5 秒，之后再读取下一张图片，不断循环读取 16 张图片。

2.2.6 SD 卡输入子系统

使用 SPI 模式对 SD 卡进行初始化与读操作，当 SD 卡初始化完成之后，再对 SD 卡进行读取，具体对 SD 卡的操作详见子模块设计。

2.2.7 数字识别系统

将从 SD 卡读取的像素数据（16 位）存入一个寄存器用于保存其对应的二值化后的数据，同时利用二值化的数据将图片中数字的上下左右边界的横纵坐标计算出来并控制其显示在 VGA 上，随后，在等待 SD 卡读取下一张图片的同时利用读时钟遍历一遍二值化数据寄存器，得出图片与三条特征线的交点个数，并且通过交点个数识别出对应的数字并传给数码管输出子系统。

3 子系统模块建模

本项目共有一个顶层模块（SD_VGA_DIG_TOP），实例化模块有：时钟分频模块（clk_div）,数码管显示模块（display_seg），VGA 显示模块（vga_driver）,数字识别模块

（dig_regnize），缓存双端口 RAM 模块（ram），图片处理模块（img_deal），图片读取控制模块（img_read_ctrl），SD 卡控制模块（sd_ctrl），SD 卡初始化模块（sd_init），SD 卡读模块（sd_read）等 10 个模块。

下面给出 VIVADO Design Files 结构图、模块框图以及系统的部分 RTL 图。

3.1 Design Files 结构图

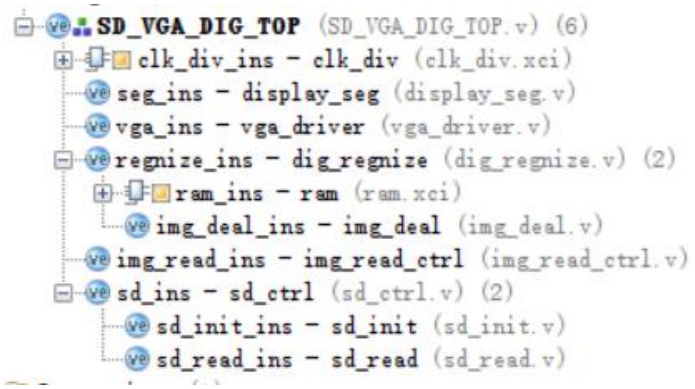


图 6　Design Files 结构图

3.2 模块框图

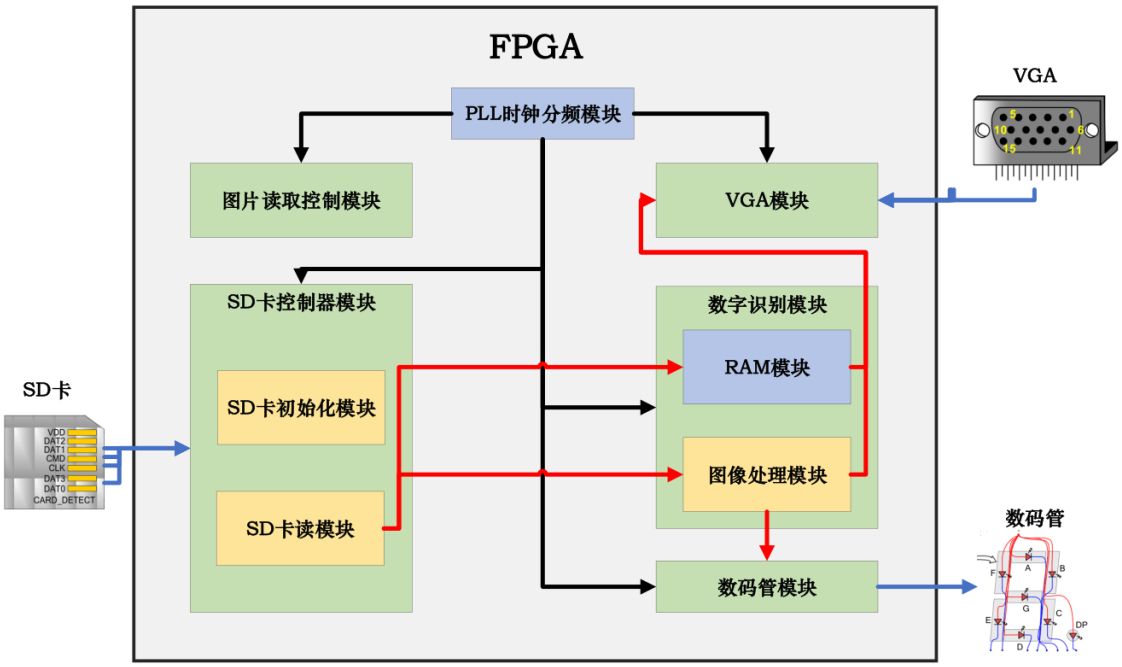


图 7

3.3 部分 RTL 图

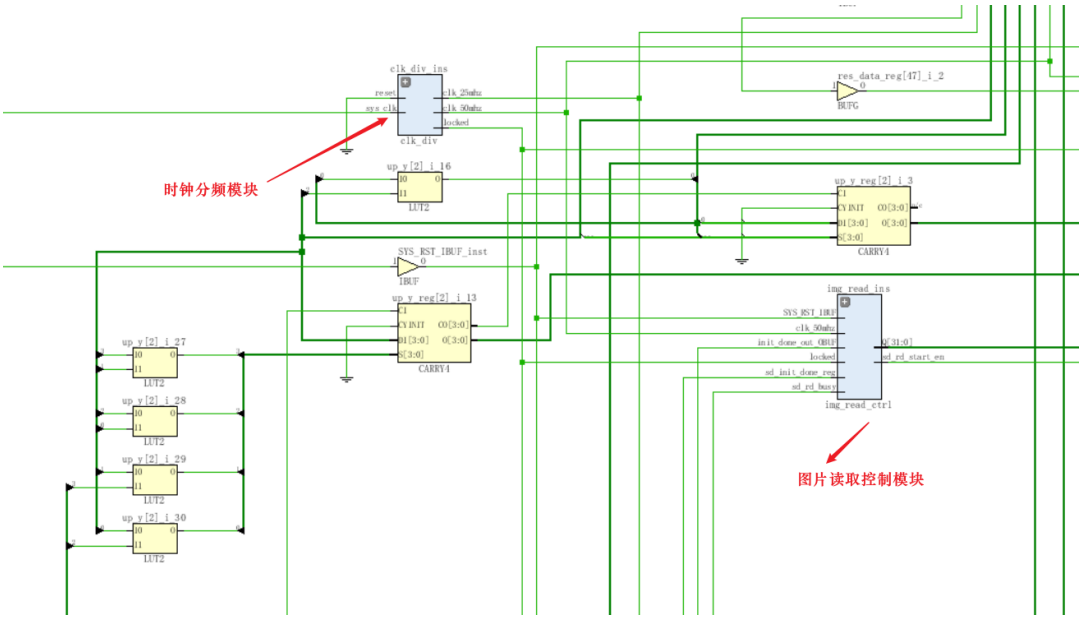


图 8

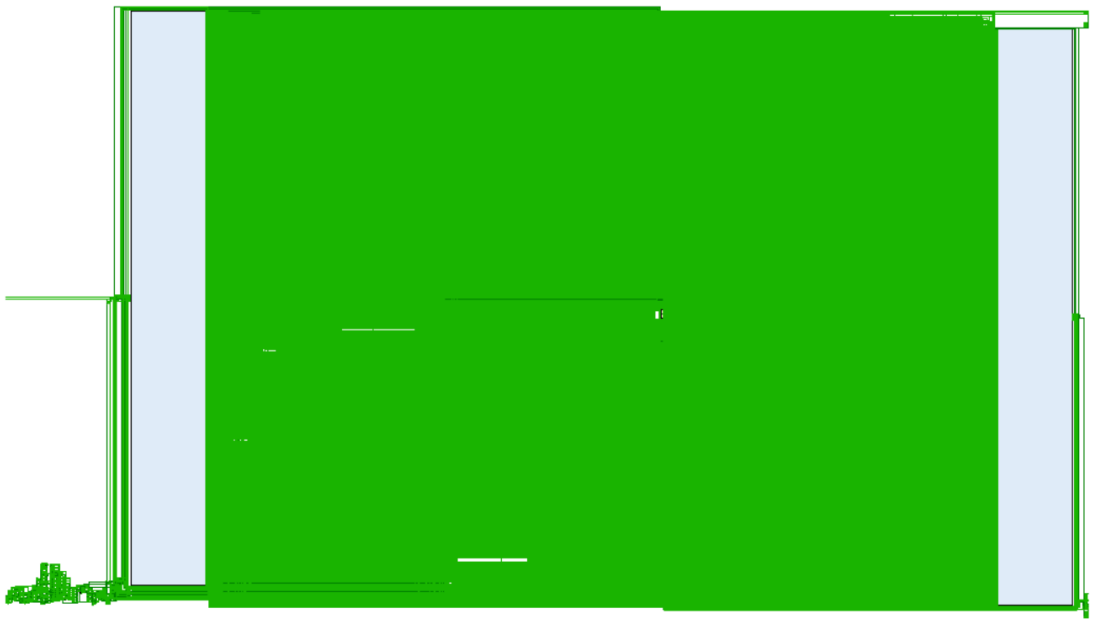


图 9

3.4 子模块建模

3.4.1 顶层模块（SD_VGA_DIG_TOP）

模块接口定义：

```
module SD_VGA_DIG_TOP(
  // 系统接口
  input        SYS_CLK ,            // 系统时钟
  input        SYS_RST ,            // 系统复位

  // SD 卡外部接口
  input        SD_CD       ,
  output       SD_RESET    ,
  output       SD_SCK      ,
  output       SD_CMD      ,
  inout   [3:0]   SD_DATA ,

  // VGA 外部接口
  output          VGA_HS   ,         // 行同步信号
  output          VGA_VS   ,         // 场同步信号
  output  [11:0]  VGA_RGB,           // 红绿蓝颜色输出

  // Debug 接口
  output  init_done_out,             // SD 卡初始化完成信号
  output  [1:0]   n_node,            // 竖直交点个数
  output  [1:0]   m1_node_l,         // 横左交点个数
  output  [1:0]   m1_node_r,         // 横右交点个数
  output  [1:0]   m2_node_l,         // 横左交点个数
  output  [1:0]   m2_node_r,         // 横右交点个数

  output  [7:0]   DIG,               // 数码管信号(识别出来的数字)
  output  [7:0]   BIT_CTRL           // 数码管位控信号(只显示一位)
  );
```

接口简介：

接口	性质	描述
SYS_CLK	input	系统的 100Mhz 时钟，用于分频成 VGA 和 SD 卡时钟
SYS_RST	input	系统的复位，用户接口位一个 SW 开关
SD 卡	input/output	FPGA 与 SD 卡相连，并且通过 SPI 模式进行与 SD 卡的数

		据交互
VGA	output	FPGA 与 VGA 相连，在 VGA 上显示图片
DIG	output	与数码管相连，将识别的数字显示在数码管上
Debug	output	连接 LED 灯，包括 SD 卡是否初始化，数字识别中数字与各特征线的交点

模块描述：

本模块是顶层模块，直接与各外围模块接口和用户控制接口相连（**其中与 VGA 的 RGB 颜色和行场同步信号相连，与 SD 卡的 CD，RESET，SCK，CMD 和 4 位 DATA 相连，与数码管 8 位显示接口和 8 位位控接口相连，还有一些与 LED 灯相连作为 Debug 接口**）。顶层模块实例化了各子模块包含了总体功能的具体实现方法，表明了各子模块之间是如何相互连接的。

3.4.2 时钟分频 PLL 模块（IP 核实现）（clk_div）

接口定义（由于调用 IP 核，故只写出实例化）：

```
wire                    clk_25mhz       ;   // 25.175mhz 时钟
wire                    clk_50mhz       ;   // 50.000mhz 时钟
wire                    locked          ;   // 锁
wire                    rst             ;   // 内部复位信号

clk_div clk_div_ins(
    .sys_clk(SYS_CLK),
    .clk_50mhz(clk_50mhz),
    .clk_25mhz(clk_25mhz),
    .reset(0),
    .locked(locked)
);
```

接口简介：

接口	性质	描述
SYS_CLK	input	系统的 100Mhz 时钟，将其分频为 50Mhz，25Mhz
reset（0）	input	复位，始终保持低电平（始终不复位）
clk_50mhz	input/output	50Mhz 时钟，用于 SD 卡的 SPI 模式

clk_25mhz	output	25Mhz 时钟，用于 VGA
locked	output	锁，等分频后的时钟稳定后被拉高，用于更新复位

模块描述：

本模块是实例化的 VIVADO IP 核 Clocking Wizard（5.3），输出锁相环，本想使用平时作业中写的时钟分频模块，但是由于 VGA 需要的时钟为 25.175Mhz，使用自实现的分频模块不好分出 25.175Mhz 的时钟，而且方便起见，使用自带的 IP 核实现。

3.4.3 驱动 VGA 显示模块（vga_driver）

接口定义：

```
1.  module vga_driver(
2.   input           vga_clk,          // VGA 驱动时钟
3.   input           rst,              // 复位信号
4.
5.   output          vga_hs,           // 行同步信号
6.   output          vga_vs,           // 场同步信号
7.   output  [11:0]  vga_rgb,          // 红绿蓝输出
8.
9.   input   [11:0]  pix_data,         // 像素点数据
10.  output  [10:0]  pix_xpos,         // 像素点横坐标
11.  output  [10:0]  pix_ypos          // 像素点纵坐标
12.  );
```

接口简介：

接口	性质	描述
vga_clk	input	50Mhz 时钟
rst	input	复位，低电平有效
vga_hs	output	输出给 VGA 的行同步信号
vga_vs	output	输出给 VGA 的场同步信号
vga_rgb	output	输出给 VGA 的 12 位像素数据
pix_data	input	从 ram 中读取的 12 位像素数据用于输出给 VGA
pix_xpos/pix_ypos	output	输出当前 VGA 需要显示像素的横纵坐标

模块描述与建模过程：

学校领取的 VGA 支持两种分辨率一种是 640*480，还有一种是 1024*768，一开始我选择

的是 1024*768，但是考虑到如果使用高分辨率的话，会给内存的使用带来压力，并且板子给的资源就比较少，所以采用了 640*480 的分辨率。

通过查找资料，获得了 VGA 相关参数（如图），于是定义了相关参数如下：

VGA Signal 640 x 480 @ 60 Hz Industry standard timing

General timing

Screen refresh rate	60 Hz
Vertical refresh	31.46875 kHz
Pixel freq.	25.175 MHz

Horizontal timing (line)

Polarity of horizontal sync pulse is negative.

Scanline part	Pixels	Time [µs]
Visible area	640	25.422045680238
Front porch	16	0.63555114200596
Sync pulse	96	3.8133068520357
Back porch	48	1.9066534260179
Whole line	800	31.777557100298

Vertical timing (frame)

Polarity of vertical sync pulse is negative.

Frame part	Lines	Time [ms]
Visible area	480	15.253227408143
Front porch	10	0.31777557100298
Sync pulse	2	0.063555114200596
Back porch	33	1.0486593843098
Whole frame	525	16.683217477656

图 10

```
1.  /************************************************************
2.  参数(分辨率: 640*480 时钟频率: 25.175mhz 位宽:10 位)
3.  ************************************************************/
4.  parameter  H_SYNC   =  10'd96;    // 行同步
5.  parameter  H_BACK   =  10'd48;    // 行显示后沿
6.  parameter  H_DISP   =  10'd640;   // 行有效数据
7.  parameter  H_FRONT  =  10'd16;    // 行显示前沿
8.  parameter  H_TOTAL  =  10'd800;   // 行扫描周期
9.
10. parameter  V_SYNC   =  10'd2;     // 场同步
11. parameter  V_BACK   =  10'd33;    // 场显示后沿
12. parameter  V_DISP   =  10'd480;   // 场有效数据
13. parameter  V_FRONT  =  10'd10;    // 场显示前沿
14. parameter  V_TOTAL  =  10'd525;   // 场扫描周期
```

VGA 的时序较为简单，分为行时序和场时序（如图），一个行扫描周期实现一行像素点的

显示，一个场扫描周期实现一帧图像的显示。拿行时来说，我们通过 VGA 的时钟对行计数器进行计数当处于同步阶段时就拉高行同步信号用于同步数据，随后在有效数据阶段拉高 VGA 显示使能信号将一行的像素点数据显示出来。同理，场时序也是如此。

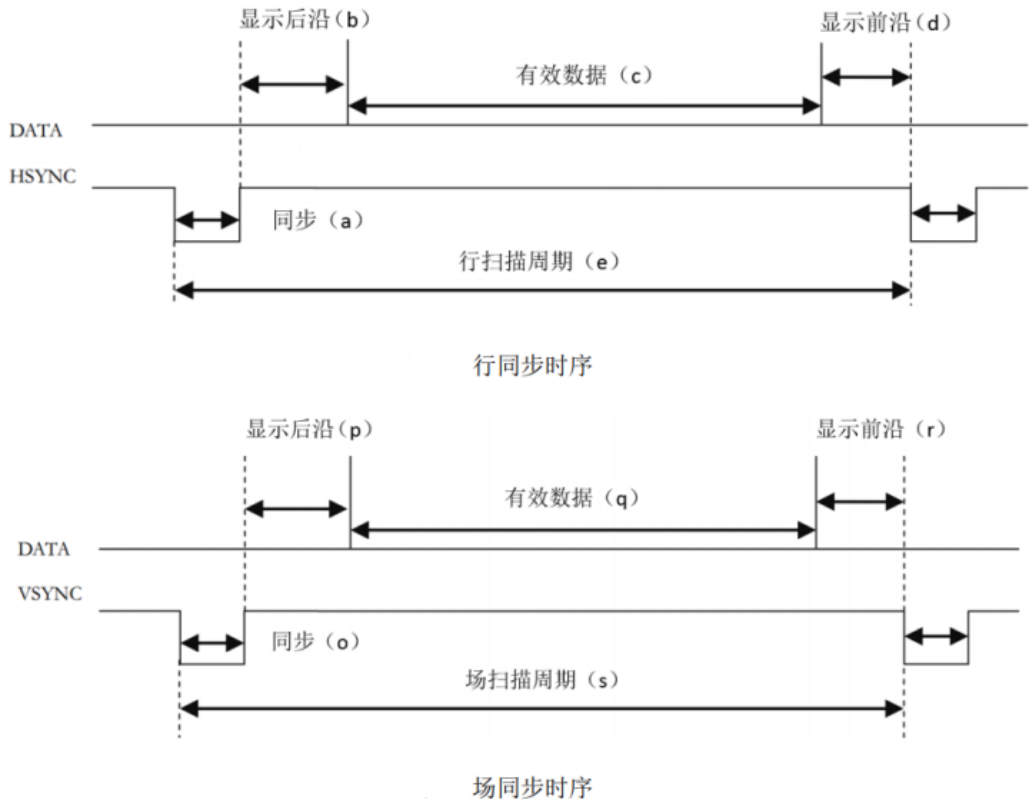


图 11

此外，值得一提的是，多数 VGA 都是以 RGB565 的格式显示像素数据的，但是我们的开发板支持的 VGA 是 RGB444 的格式，而且从 SD 中读取的像素数据也是 RGB565 格式，所以需要对 RGB565 进行裁剪与处理，而这个工作是我在像素数据存入 RAM 之前做的，所以输入的数据也就是 RGB444。

3.4.4 数码管显示模块（display_sig）

接口定义：

```
1. module display_seg(
2.  input       [3:0]   num     ,   // 需要显示的数字
3.  output reg  [7:0]   dig     ,   // 输出的数码管信号
4.  output      [7:0]   bit_ctrl    // 输出的数码管位控
5.  );
```

接口简介：

接口	性质	描述
num	input	输入需要显示的数字
dig	output	显示信号，用于七段数码管的显示
bit_ctrl	output	位控信号，用于八位数码管的控制

模块描述与建模过程：

我们的平时实验写过七段数码管，因此可以搬用，但是我们的 display7.v 是将一个数字显示在八个数码管上，没有进行位控，而本次我只需要将识别出的数字（0-9）显示在一位数码管上，因此需要加入位控。根据开发板提供的数码管资料（如图）进行模块的建模和约束文件编写。

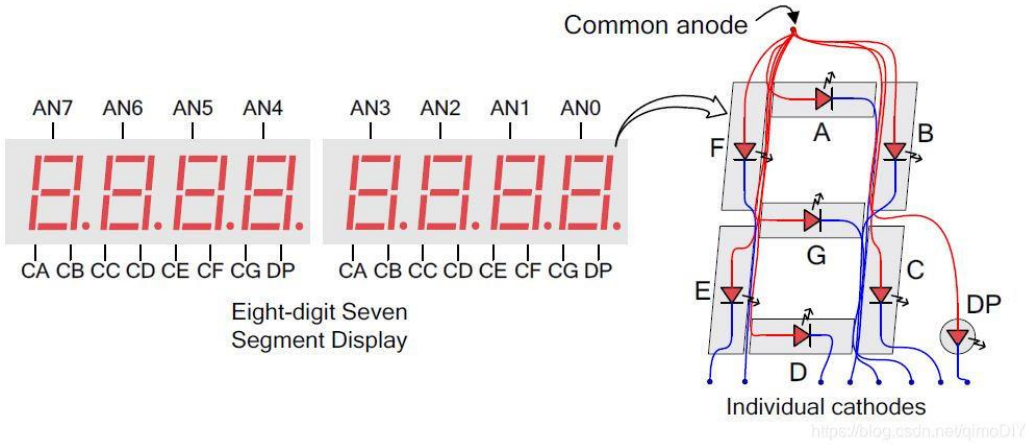


图 12

3.4.5 SD 卡控制模块（sd_ctrl）

接口定义：

```
1.  module sd_ctrl(
2.    input                clk_ref      ,  // 时钟信号
3.    input                rst          ,  // 复位信号,低电平有效
4.
5.    // SD 卡接口
6.    input                SD_CD        ,
7.    output               SD_RESET     ,
8.    output               SD_SCK       ,
9.    output               SD_CMD       ,
10.   inout    [3:0]       SD_DATA      ,
11.
12.   // 读 SD 卡接口
```

```
13.  input               rd_start_en    ,    // 开始读 SD 卡数据信号
14.  input   [31:0]      rd_sec_addr    ,    // 读数据扇区地址
15.  output              rd_busy        ,    // 读数据忙信号
16.  output              rd_val_en      ,    // 读数据有效信号
17.  output  [15:0]      rd_val_data    ,    // 读数据
18.  output  [18:0]      ram_wr_addr    ,    // 写入 ran 的地址
19.
20.  output              sd_init_done        // SD 卡初始化完成信号
21.  );
```

接口简介：

接口	性质	描述
clk_ref	input	SD 的 SPI 模式参考时钟
rst	iutput	复位信号，低电平有效
SD_CD	input	SD 卡保留接口
SD_RESET	output	SD 卡复位，保持低电平
SD_SCK	output	连接 SD 的时钟
SD_CMD	output	连接 SD 的命令和响应信号
SD_DATA	inout	连接 SD 的 4 根数据线
rd_start_en	input	从图片读取模块中输出的开始读 SD 卡信号
rd_sec_addr	input	从图片读取模块中输出的读 SD 卡的扇区地址
rd_busy	output	当 SD 卡正在读数据的时候，拉高读忙信号
rd_val_en	output	从 SD 卡中读取的数据的有效信号
rd_val_data	output	从 SD 卡中读取的 16 位数据
ram_wr_addr	output	根据读取的像素数据的个数输出的写入 ram 的地址

SPI 模式简介：

我们对 SD 的读写协议一般有 SPI 模式和 SDIO 模式两种，由于 SPI 在芯片管脚上只占用四根线，而且 SPI 实现 SD 卡读写只需要修改一些 SPI 借口逻辑就可以实现，因此，本实验采用的是 SPI 模式对 SD 卡进行操作。

SPI 是串行外设接口(Serial Peripheral Interface)的缩写。是 Motorola 公司推出的一种同步串行接口技术，是一种高速的，全双工，同步的通信总线。SPI 具有支持全双工通信；通信简单；数据传输速率快；高速、同步、全双工、非差分、总线式；主从机通信模式等特点。

SPI 模式的信号线有：CS、CLK、MISO、MOSI，4 根线。

The Nexys4 DDR provides a microSD slot for both FPGA configuration and user access. The on-board Auxiliary Function microcontroller shares the SD card bus with the FPGA. Before the FPGA is configured the microcontroller must have access to the SD card via SPI. Once a bit file is downloaded to the FPGA (from any source), the microcontroller power cycles the SD slot and relinquishes control of the bus. This enables any SD card in the slot to reset its internal state machines and boot up in SD native bus mode. All of the SD pins on the FPGA are wired to support full SD speeds in native interface mode, as shown in Figure 21. The SPI is also available, if needed. Once control over the SD bus is passed from the microcontroller to the FPGA, the SD_RESET signal needs to be actively driven low by the FPGA to power the microSD card slot. For information on implementing an SD card controller, refer to the SD card specification available at www.sdcard.org.

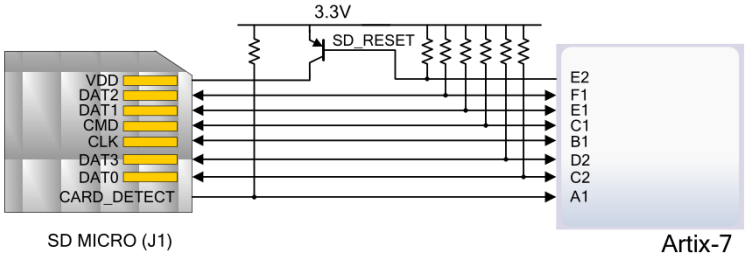
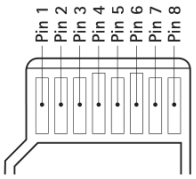


图 13

由于板子只能插入 TF 卡即 MicroSD card，我查阅相关资料得到 TF 卡的 SPI 模式接口（如图），通过这些接口，将 SD 卡和 FPGA 通过 SPI 模式连接起来对其进行操作。

**Specifications and product offerings are subject to change without notice

***Available in Q1 '05



SD MODE

Pin No.	Name	Type	Description
1	DAT2	I/O/PP	Data Line (bit 2)
2	CD/DAT3	I/O/PP	Card Detect/Data line (Bit 3)
3	CMD	PP	Command Response
4	VDD	S	Supply Voltage
5	CLK	I	Clock
6	VSS	S	Supply Voltage Ground
7	DAT0	I/O/PP	Data Line (bit 0)
8	DAT1*	I/O/PP	Data Line (bit 1)

*The extended DAT line (DAT1) is an input on power-up. It starts to operate as DAT lines after SET_BUS_WIDTH command is sent.

SPI MODE

Pin No.	Name	Type	Description
1	RSV	–	–
2	CS	I	Chip Select (neg true)
3	DI	I	Data In
4	VDD	S	Supply Voltage
5	SCLK	I	Clock
6	VSS	S	Supply Voltage Ground
7	DO	O/PP	Data Out
8	RSV	–	–

图 14

SPI 模式连接：

```
1.  wire    spi_miso    ;       // MISO 接口
2.  wire    spi_clk     ;       // CLK 接口
3.  reg     spi_cs      ;       // CS 接口
4.  reg     spi_mosi    ;       // MOSI 接口
5.
6.  // SPI 模式接口
7.  assign      SD_RESET        =   0;
8.  assign      SD_DATA[1]      =   1;
9.  assign      SD_DATA[2]      =   1;
```

```
assign      SD_DATA[3]      =   spi_cs    ;
assign      SD_CMD          =   spi_mosi  ;
assign      SD_SCK          =   spi_clk   ;
assign      spi_miso        =   SD_DATA[0];
```

3.4.6 SD 卡的初始化与读取（sd_init、sd_read）

SD 卡操作初始化与读取流程控制：

在上面的 SD 卡控制模块中，我们有一个 always 块用于控制 SD 卡的操作流程，当 SD 卡还未初始化的时候，我们对其进行初始化，初始化完成之后才可以进行读操作，当读忙信号被读模块拉高时，进行读操作，否则 SD 卡将一直处于空闲状态（此时 CS 片选信号和 MOSI 信号都为高电平）。

注：CS、CLK 和 MOSI 为 SD 卡控制模块输出给 SD 卡的信号，当对 SD 卡进行不同的操作时，我们需要选择对应的输出，而 MISO 为 SD 卡输入，不需进行更改直接输入到初始化和读子模块。

```
wire    init_clk    ;       // 初始化 SD 卡时的低速时钟
wire    init_cs     ;       // 初始化模块 SD 片选信号
wire    init_mosi   ;       // 初始化模块 SD 数据输出信号
wire    rd_cs       ;       // 读数据模块 SD 片选信号
wire    rd_mosi     ;       // 读数据模块 SD 数据输出信号

// SD 卡的 SPI_CLK
assign  clk_ref_neg = ~clk_ref;
assign  spi_clk = (sd_init_done == 1'b0)  ?  init_clk  :  clk_ref_neg
;

/*************************************************************
SD 卡接口的信号选择
*************************************************************/
always @(*) begin
    if(sd_init_done == 1'b0) begin
        spi_cs = init_cs;
        spi_mosi = init_mosi;
    end
    else if(rd_busy) begin
        spi_cs = rd_cs;
        spi_mosi = rd_mosi;
    end
    else begin
```

```
        spi_cs = 1'b1;
        spi_mosi = 1'b1;
    end
end
```

在对 SD 卡进行操作之前，我们首先需要对 SD 卡的命令有一些了解。SD 卡的 command 占 6 bit，一般叫 CMDx 或 ACMDx，比如 CMD1 就是 1，CMD13 就是 13，ACMD41 就是 41，依此类推。Command Argument 占 4 byte，并不是所有命令都有参数，没有参数的话该位一般就用置 0。最后一个字节由 7 bit CRC 校验位和 1 bit 停止位组成。在 SPI 模式下，CRC 校验是被忽略的，可以都置 1 或置 0.但是发送 CMD0 和 CMD8 时要加上 CRC 校验（因为发送 CMD0 时还未进入 SPI 模式）。

- **In idle state**: The card is in idle state and running the initializing process.
- **Erase reset:** An erase sequence was cleared before executing because an out of erase sequence command was received.
- **Illegal command**: An illegal command code was detected.
- **Communication CRC error**: The CRC check of the last command failed.
- **Erase sequence error**: An error in the sequence of erase commands occurred.
- **Address error**: A misaligned address that did not match the block length was used in the command.
- **Parameter error**: The command's argument (e.g. address, block length) was outside the allowed range for this card.

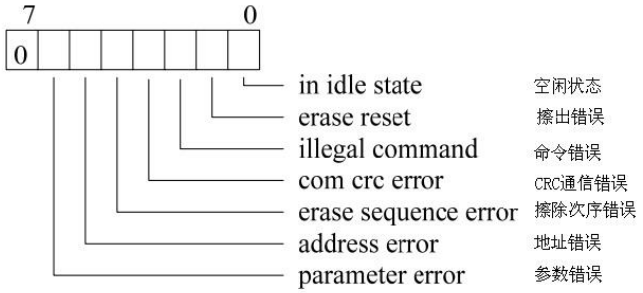


Figure 7-9: R1 Response Format

图 15

SD 卡初始化过程：

在对 SD 卡初始化之前，我们需要对其上电后复位。复位的方法为：

- 拉高 CS，发送至少 74 个时钟周期来使 SD 卡达到正常工作电压和进行同步
- 选低 CS，发送 CMD0，需要收到回应 0x01 表示成功进入等待状态
- 拉高 CS，发送 8 个时钟

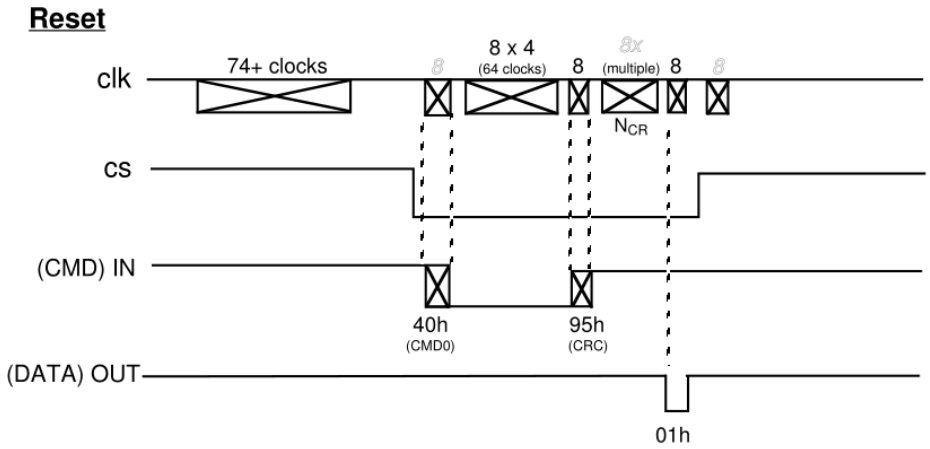


图 16

复位成功后我们就可以对其进行初始化了。初始化的过程较复杂，状态较多，看的资料也比较复杂。对其过程总结如下:

- 使用 CMD，发送 CMD1，收到 0x00 表示成功
- 使用 CMD55+ACMD41
- 发送 CMD55（表示接下来要对其使用 ACMDx 类的命令），收到 0x01
- 发送 ACMD41，收到 0x00 表示成功

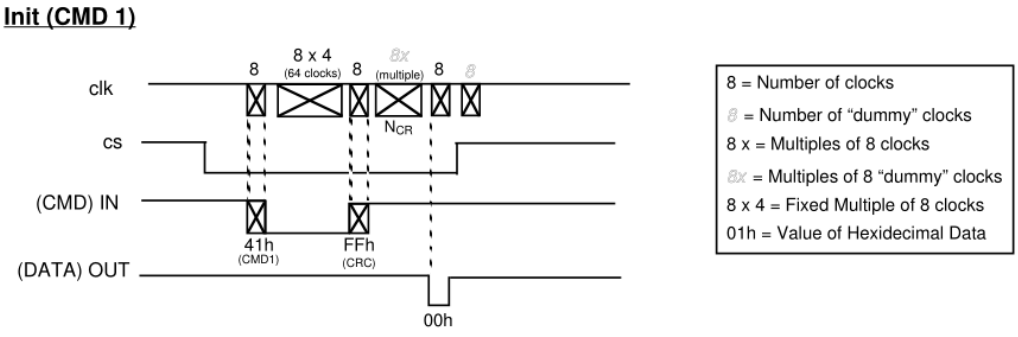


图 17

总结一下，SD 的初始操作可以用以下流程图来表示，其实，无疑就是不断向 SD 卡发送命令，然后接收 SD 卡返回的数据，如果数据符合要求就代表我们可以发送下一个命令或者初始化成功，如果 SD 卡返回的数据有错误则重新发送命令，直到 SD 卡返回正确的数据，当发现 SD 卡返回的数据和我们要求的数据有偏差时，说明我们的初始化操作失败了（比如电压要求不符，SD 协议不符等等原因）。

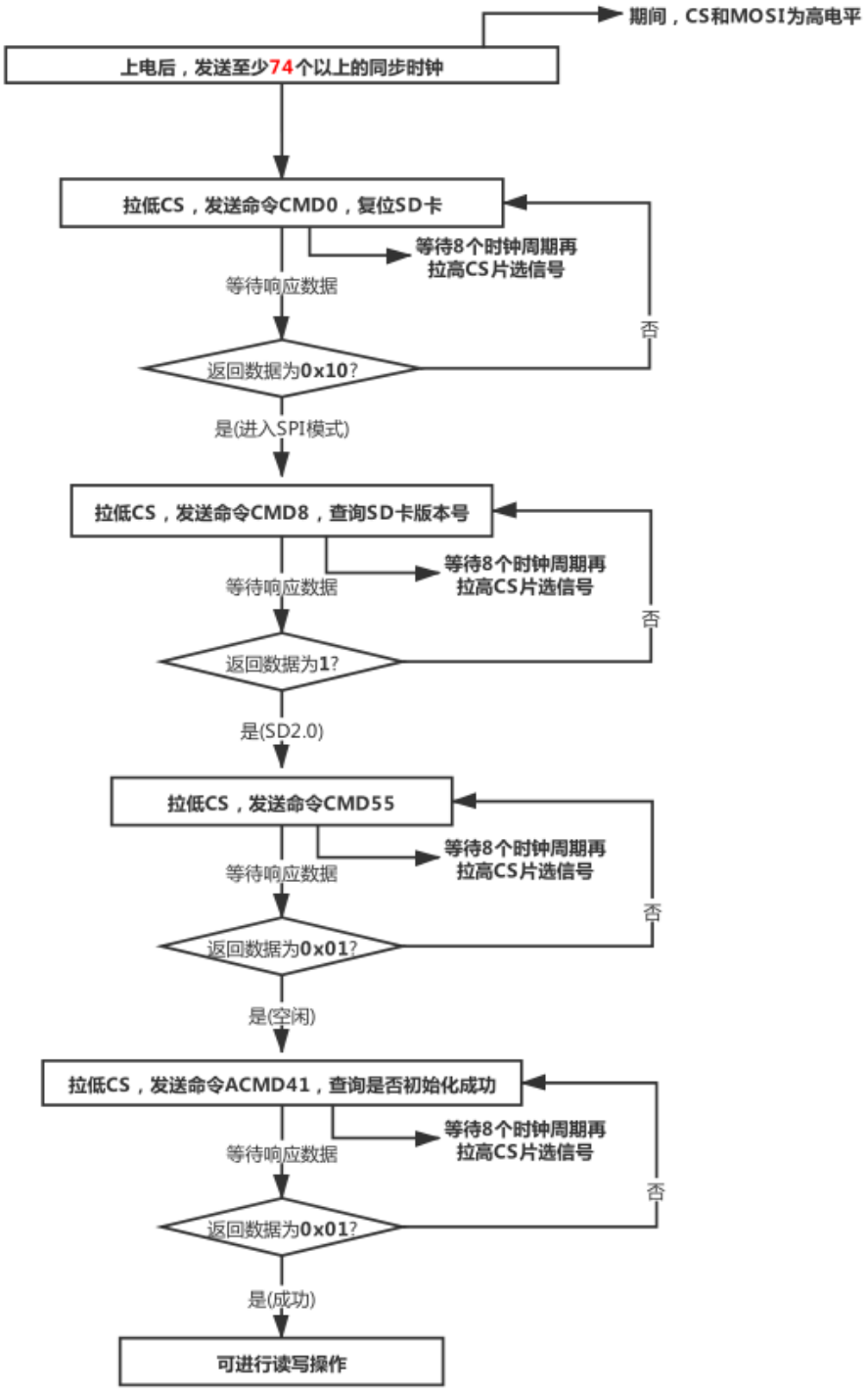


图 18

SD 初始化模块接口：

```
1. module sd_init(
2.   input          clk_ref       ,  // 参考时钟
3.   input          rst           ,  // 复位信号,低电平有效
4.
```

```
    input           sd_miso       ,  // SD 卡 SPI 串行输入数据信号
    output          sd_clk        ,  // SD 卡 SPI 时钟信号
    output  reg     sd_cs         ,  // SD 卡 SPI 片选信号
    output  reg     sd_mosi       ,  // SD 卡 SPI 串行输出数据信号

    output  reg     sd_init_done      // SD 卡初始化完成信号
);
```

接口描述：

接口	性质	描述
clk_ref	input	SD 的 SPI 模式参考时钟
rst	input	复位信号，低电平有效
sd_miso	input	SPI 模式 MISO 接口
sd_clk	output	SD 卡复位，保持低电平
sd_cs	output	连接 SD 的时钟
sd_mosi	output	连接 SD 的命令和响应信号
sd_init_done	output	SD 卡初始化成功信号

值得注意的是：SD 卡初始化过程中的 SPI 时钟需要使用低速时钟（最好小于 400Khz），所以需要对 clk_ref 进行分频成 250Khz 输出，这时就可以直接利用平时是实验中的时钟分频模块对 50Mhz 的时钟进行分频了。

SD 卡命令定义与状态机定义：

```
/*****************************************************************
参数定义
*****************************************************************/
// SD 卡命令定义
parameter  CMD0  = {8'h40,8'h00,8'h00,8'h00,8'h00,8'h95};
parameter  CMD8  = {8'h48,8'h00,8'h00,8'h01,8'haa,8'h87};
parameter  CMD55 = {8'h77,8'h00,8'h00,8'h00,8'h00,8'hff};
parameter  ACMD41 = {8'h69,8'h40,8'h00,8'h00,8'h00,8'hff};

// 状态编码
parameter  sta_idle         = 7'b000_0001;  // 上电等待 SD 卡稳定
parameter  sta_send_cmd0    = 7'b000_0010;  // 发送软复位命令
parameter  sta_wait_cmd0    = 7'b000_0100;  // 等待 SD 卡响应
parameter  sta_send_cmd8    = 7'b000_1000;  // 检测 SD 卡是否满足电压范围
parameter  sta_send_cmd55   = 7'b001_0000;  // 告诉 SD 卡接下来的命令是应用相关命令
parameter  sta_send_acmd41 = 7'b010_0000;  // 发送操作寄存器(OCR)内容
```

```
parameter  sta_init_done   = 7'b100_0000;  // SD 卡初始化完成
```

SD 卡初始化的过程具有七个状态，我们使用三段状态机进行状态的转移，当 SD 卡处于最后一个状态（sta_init_done）时，我们认为 SD 卡已经初始化完成了，把 sd_init_done 信号拉高，后续就可以对 SD 卡进行读取操作了。

SD 卡读取模块：

SD 卡读取的过程和初始化流程有很多相似的地方，也就是向 SD 卡发送命令然后接收 SD 返回的响应数据的过程，只是有一段 SD 返回的有效数据段是我们需要的数据。SD 卡读过程总结（由于我们已经对 SD 卡初始化，所以可以不对 SD 卡返回的响应数据进行检验）：

- 发送 CMD17
- 连续读直到读到开始字节 0xFE（可以只检测读到最后一个 0）
- 读 512 个字节的有效数据即一个扇区
- 读两个 CRC 校验字节

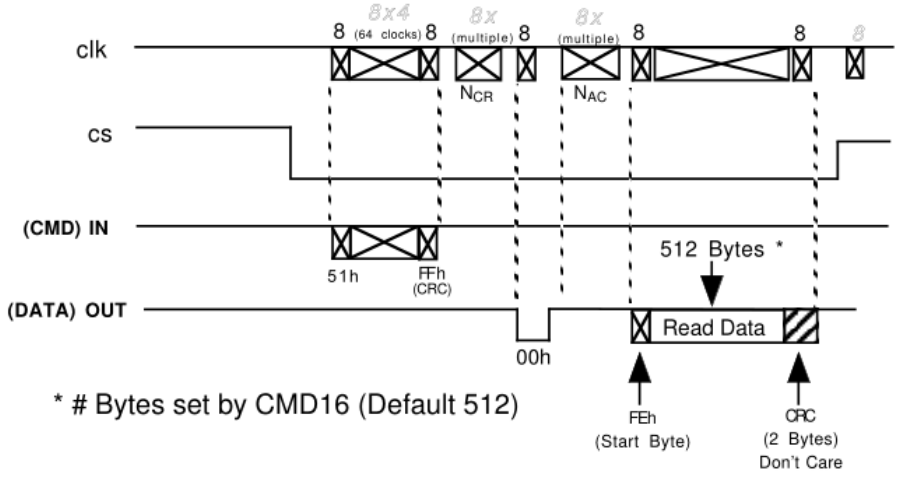


图 19

SD 卡读取模块接口定义：

```
module sd_read(
  input                 clk_ref         ,  // 时钟信号
  input                 rst             ,  // 复位信号

  input                 sd_miso         ,  // SD 卡 SPI 串行输入数据信号
  output  reg           sd_cs           ,  // SD 卡 SPI 片选信号
  output  reg           sd_mosi         ,  // SD 卡 SPI 串行输出数据信号

  input                 rd_start_en     ,  // 开始读 SD 卡数据信号
```

```
10.  input        [31:0]  rd_sec_addr    ,  // 读数据扇区地
     址
11.  output  reg          rd_busy        ,  // 读数据忙信号
12.  output  reg          rd_val_en      ,  // 读数据有效信号
13.  output  reg  [15:0]  rd_val_data    ,  // 读数据
14.
15.  output  reg  [18:0]  ram_wr_addr       // 读 ram 地址
16.  );
```

3.4.7 读取图片控制模块（img_read_ctrl）

接口定义：

```
1.  module img_read_ctrl(
2.   input                clk            ,  // 时钟信号
3.   input                rst            ,  // 复位信号
4.
5.   input                rd_busy        ,  // SD 卡读忙
6.   output  reg          rd_start_en    ,  // 开始写 SD 卡数据信号
7.   output  reg  [31:0]  rd_sec_addr       // 读数据扇区地址
8.   );
```

接口描述：

接口	性质	描述
clk	input	50Mhz 操作时钟
rst	input	复位信号，低电平有效
rd_busy	input	SD 卡读忙信号，通过采集其下降沿来控制扇区地址
rd_start_en	output	输出开始读取 SD 卡数据信号
rd_sec_addr	output	输出 SD 卡读取的扇区地址

模块建模过程：

在此之前，我们先谈一谈在开始本实验前需要做一些图片准备工作：

- 准备若干张（此处为 16 张）分辨率为 640*480 的图片，通过 Img2Lcd 软件将其转化为 bin 文件（如图选择合适的尺寸，RGB565 格式，并选择高位在前）

图 20

- 将 bin 文件存入 SD 卡中
- 通过 WinHex 软件查看 SD 卡内部的数据与扇区地址（如图），将每张照片的扇区地址记录下来，这将是我们循环读取照片的依据。

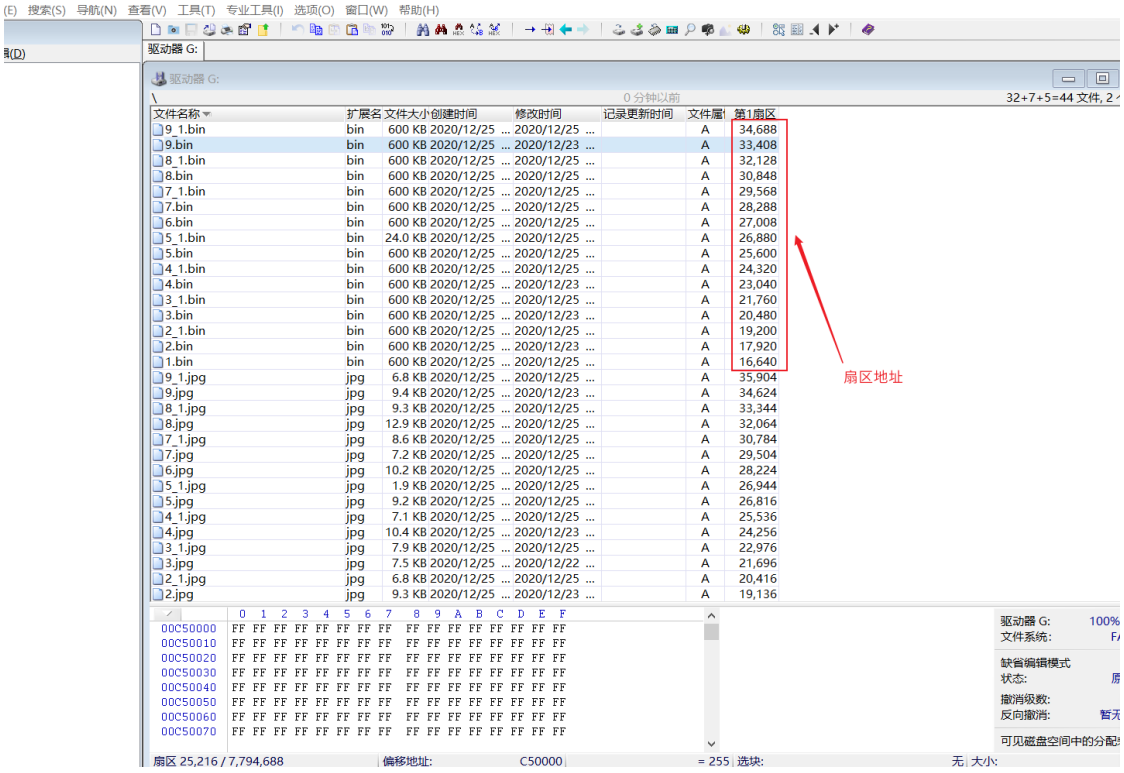


图 21

这样我们就得到了 16 张图片对应的扇区地址，然后我们将其写入到图片读取控制模块。

```
/************************************************************
参数定义(16 张图片的地址)
************************************************************/
parameter PIC_ADDR0 = 32'd23040 ;
parameter PIC_ADDR1 = 32'd28288 ;
parameter PIC_ADDR2 = 32'd34688 ;
parameter PIC_ADDR3 = 32'd32128 ;
parameter PIC_ADDR4 = 32'd29568 ;
parameter PIC_ADDR5 = 32'd26880 ;
parameter PIC_ADDR6 = 32'd24320 ;
parameter PIC_ADDR7 = 32'd16640 ;
parameter PIC_ADDR8 = 32'd19200 ;
parameter PIC_ADDR9 = 32'd17920 ;
parameter PIC_ADDR10 = 32'd25664 ;
parameter PIC_ADDR11 = 32'd30848 ;
parameter PIC_ADDR12 = 32'd35968 ;
parameter PIC_ADDR13 = 32'd33408 ;
parameter PIC_ADDR14 = 32'd20480 ;
parameter PIC_ADDR15 = 32'd21760 ;
```

这样就可以使用一个 4 位的寄存器变量来指示当前读取的图片并且可以实现 16 张图片循环读取了，每读取完一张就等待 2.5 秒，随后再读取下一张图片。

3.4.8 数字识别模块（dig_regnize）

接口定义：

```
module dig_regnize(
  input   w_en,          // 写使能
  input   clk_w,         // 写时钟
  input   clk_r,         // 读时钟
  input   [18:0]  addr_w, // 写地址
  input   [18:0]  addr_r, // 读地址
  input   [15:0]  dat_w   ,  // 16 位像素传入
  output  [11:0]  dat_r   ,  // 12 位像素读出
  output  [1:0]   n_node  ,       // 竖直交点个数
  output  [1:0]   m1_node_l,      // 横左交点个数
  output  [1:0]   m1_node_r,      // 横右交点个数
  output  [1:0]   m2_node_l,      // 横左交点个数
  output  [1:0]   m2_node_r,      // 横右交点个数
  output  [3:0]   iden_num
```

```
15.  );
```

接口描述：

接口	性质	描述
w_en	input	ram 写使能
clk_w	input	ram 写时钟，50Mhz
clk_r	input	ram 读使能，25Mhz
addr_w	output	ram 写地址，由 SD 卡读模块给出
addr_r	output	ram 读地址，通过 VGA 的横纵坐标计算出
dat_w	input	16 位的原始像素数据输入
dat_r	output	12 位的像素数据读出给 VGA 显示模块
iden_num	output	识别出来的数字
node…	output	图像与各特征线的交点

本模块实例化了两个模块，分别为图像处理与数字识别模块和双口 RAM 模块。首先来介绍双口 RAM，双口 RAM 的作用是缓存从 SD 中读取的数据，由于 SD 卡的时钟与 VGA 的时钟不一样，因此需要一个双口 RAM 先将数据缓存才可以实现两个模块之间的数据交互。由于我们的平时实验曾经做过单端口的 RAM，所以想实现一个简单的双口 RAM，但是在做的过程中，需要定义一个 640*480*12（3686400）大小的寄存器，综合的时候会报错，发现无法定义这么大容量的寄存器。于是我选择使用 VIVADO IP 核：Block Memory Generator 8.3

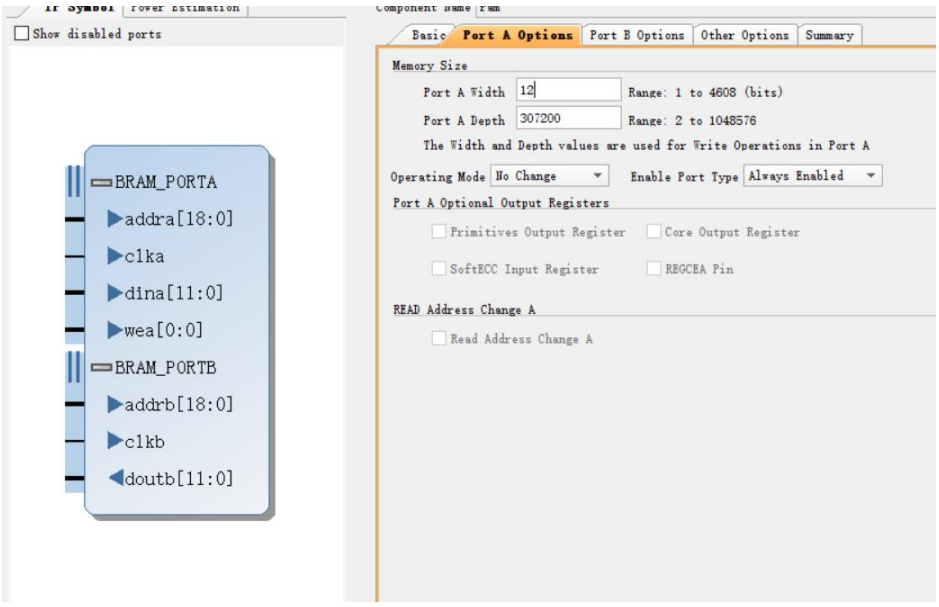


图 22

ram 模块接口定义（由于调用的 IP 核所以给出实例化）：

```
// 适应 VGA 的 12 位像素数据
assign dat_w_pix = { dat_w[15:12], dat_w[10:7], dat_w[4:1] };
/*************************************************************
实例化双端口 ram 模块
*************************************************************/
ram ram_ins(
    .clka (clk_w)    ,      // 写时钟
    .wea  (w_en)     ,      // 使能
    .addra(addr_w)   ,      // 写地址
    .dina (dat_w_pix),      // 写数据
    .clkb (clk_r)    ,      // 读时钟
    .addrb(addr_r)   ,      // 读地址
    .doutb(dat_r_pre)       // 读数据
);
```

接下来是图像处理与数字识别模块：

接口定义：

```
module img_deal(
    input       w_en,                 // 写使能，高电平有效
    input       clk_w,                // 写时钟 50mhz
    input       [18:0] addr_w,        // 写地址
    input       [11:0] dat_w,         // 12 位像素传入

    output reg  [10:0] left_x,        // 数字的左边界
    output reg  [10:0] right_x,       // 数字的左边界
    output reg  [10:0] up_y,          // 数字的左边界
    output reg  [10:0] down_y,        // 数字的左边界

    output reg  [1:0] n_node,         // 竖直交点个数
    output reg  [1:0] m1_node_l,      // 横左交点个数
    output reg  [1:0] m1_node_r,      // 横右交点个数
    output reg  [1:0] m2_node_l,      // 横左交点个数
    output reg  [1:0] m2_node_r,      // 横右交点个数
    output reg [3:0] iden_num        // 识别的数字
);
```

本实验的数字识别主要是采用特征识别，遵循以下几个步骤：

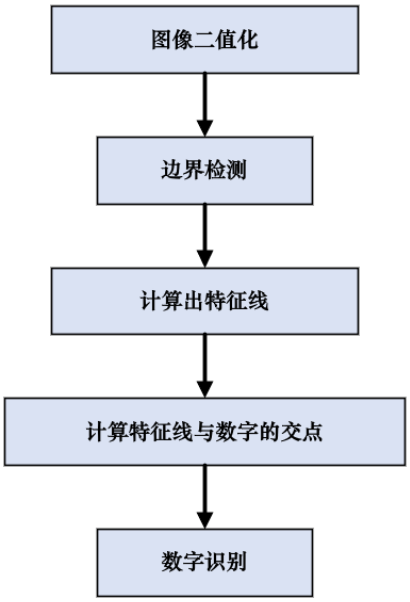


图 23

- **第一步，将图像二值化。**我们的图像数据是 RGB565 格式，然后经过我们的处理转化为 RGB444，接下来，我们要做的就是将 RGB444 转换为灰度图。由于我们的最终目的是得到二值化后的像素（只有 0 和 1），而且我们所要求的灰度值精度并不是很高，所以直接采用 **R*4+G*10+B*2>>4** 的方式得到 4 位的灰度值，然后再与 8 进行比较，大于 8 的置 1，小于等于 8 的置 0.

```
// 二值化处理
wire    [3:0]   red;        // 红颜色分量
wire    [3:0]   green;      // 绿颜色分量
wire    [3:0]   blue;       // 蓝颜色分量
wire    [3:0]   grey;       // 灰度数据
wire            bin_dat;    // 二值化数据
assign  red     =   dat_w[11:8];
assign  green   =   dat_w[7:4];
assign  blue    =   dat_w[3:0];
assign  grey    =   (red * 4 + green * 10 + blue * 2) >> 4;
assign  bin_dat =   (grey > 4'd8) ? 1'b1 : 1'b0;
```

- **第二步，识别出数字的边界。**我们最后需要得到上下左右四个边界，我的做法是数据写进入时，将其二值化，如果其为 1（为 1 即为数字部分），就将四个边界更新，所有像素写入完毕时，就可以得出数字的四个边界了。

```
reg     bin_pix [0:307199]  ;   // 储存二值化后的图像数据
/*************************************************************
```

```
3. 利用投影分割获取数字边界
4. ************************************************************/
5. always@(posedge clk_w) begin
6.     if(w_en) begin
7.         bin_pix[addr_w] <= bin_dat;
8.         if(addr_w == 19'd1) begin
9.             left_x <= 11'd639;
10.            right_x <= 11'd0;
11.            up_y <= 11'd479;
12.            down_y <= 11'd0;
13.        end else if( !bin_dat ) begin
14.            if(addr_w % 11'd640 > right_x) right_x <= addr_w % 11'd64
   0;
15.            else right_x <= right_x;
16.            if(addr_w % 11'd640 < left_x) left_x <= addr_w % 11'd640;

17.            else left_x <= left_x;
18.            if(addr_w / 11'd640 > down_y) down_y <= addr_w / 11'd640;

19.            else down_y <= down_y;
20.            if(addr_w / 11'd640 < up_y) up_y <= addr_w / 11'd640;
21.            else up_y <= up_y;
22.        end
23.    end else begin
24.        bin_pix[addr_w] <= bin_pix[addr_w];
25.    end
26. end
```

同时，该模块计算出的边界将显示在 VGA 上：

```
1. // 显示数字边框
2. assign    dat_r = (((xpos >= left_x-
   2 - 3) && (xpos <= right_x+2 + 3) && (ypos >= up_y-
   2 - 3) && (ypos <= up_y-2))||((xpos >= left_x-
   2 - 3) && (xpos <= right_x+2 + 3) && (ypos >= down_y+2) && (ypos <= d
   own_y+2 + 3))|| ((ypos >= up_y-
   2 - 3) && (ypos <= down_y+2 + 3) && (xpos >= left_x-
   2 - 3) && (xpos <= left_x-2))|| ((ypos >= up_y-
   2 - 3) && (ypos <= down_y+2 + 3) && (xpos >= right_x+2) && (xpos <= r
   ight_x+2 + 3)) )? 12'b0000_0000_0000 : dat_r_pre;
```

- **第三步，通过边界计算出三条特征线。**假设左右边界为 l，r，上下边界为 u，d，则中心特征线 y 的坐标为（l+r）/2，特征线 x1 的纵坐标为 u+2/5*（d-u），x2 的纵坐标为 u+2/3*（d-u）。

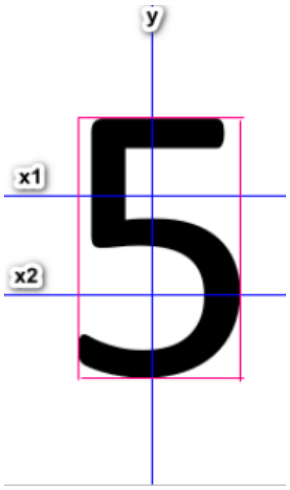


图 24

- **第四步，计算出数字与特征线的交点个数与分布，通过这些信息，对数字进行识别。**就拿上图的数字“5”而言，“5”与竖直线 y 的交点个数为 3，与 x1 的交点个数为 1 且位于 y 的左边，与 x2 的交点个数为 1 且位于 x2 的右边，这样我们就得到了 5 的交点特征值“3 1 0 0 1”，分别代表数字与 y 的交点，与 x1 且位于 y 的左边的交点，与 x1 且位于 y 的右边的交点，与 x2 且位于 y 的左边的交点，与 x2 且位于 y 的右边的交点。下面给出各数字对应的特征交点值，通过以下的特征就可以将印刷体数字识别出来了。

数字	与 y 的交点	与 x1 的左右交点		与 x2 的左右交点	
0	2	1	1	1	1
1	1	1	0	1	0
2	3	0	1	1	0
3	3	0	1	0	1
4	2	1	1	1	0
5	3	1	0	0	1
6	3	1	0	1	1
6	3	2	0	1	1
7	2	0	1	1	0
7	2	0	1	0	1
8	3	1	1	1	1
9	3	1	1	0	1

4 测试模块建模

4.1 时钟分频测试模块

测试模块代码：

```
`timescale 1ps / 1ps
module pll_test();
 reg clk_100;
 wire clk_50;
 wire clk_25_175;
 reg reset;
 wire locked;

 always #10 clk_100=~clk_100;
 initial
 begin
    reset<=0;
    clk_100<=0;
 end

 clk_wiz_0 pll_test
   (
  // Clock in ports
   .clk_100(clk_100),
   // Clock out ports
   .clk_50(clk_50),
   .clk_25_175(clk_25_175),
   // Status and control signals
   .reset(reset),
   .locked(locked)
   );
endmodule
```

仿真波形图如下：

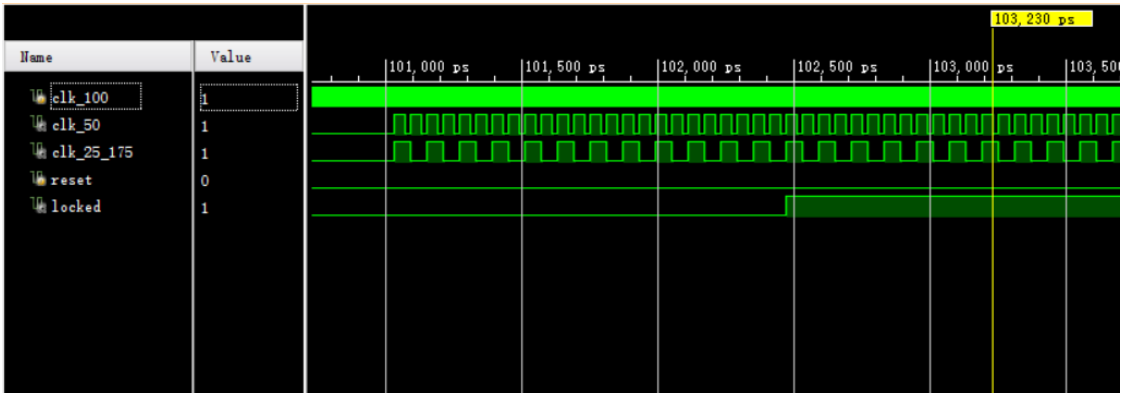


图 25

4.2 数码管测试模块

测试模块代码：

```
1. `timescale 1ns/1ns
2. module display7_tb;
3.     reg [3:0] iData;
4.     wire [6:0] oData;
5.
6.     initial
7.     begin
8.         iData=4'b0000;
9.         #20 iData=4'b0001;
10.        #20 iData=4'b0010;
11.        #20 iData=4'b0011;
12.        #20 iData=4'b0100;
13.        #20 iData=4'b0101;
14.        #20 iData=4'b0110;
15.        #20 iData=4'b0111;
16.        #20 iData=4'b1000;
17.        #20 iData=4'b1001;
18.    end
19.
20.    display7 utt(.iData(iData),.oData(oData));
21.endmodule
```

仿真波形图如下：

图 26

4.3 VGA 显示测试模块

掌握了 VGA 的时序之后，我立刻编写了 VGA 显示纯色实验，发现可以显示，就将此作为 VGA 的测试下图为 VGA 的显示纯色实验。

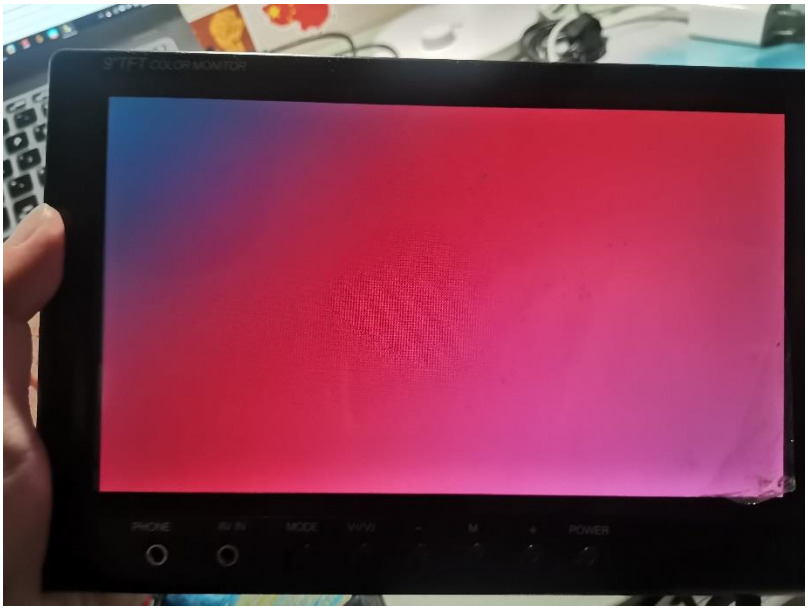


图 27

4.4 SD 卡模块测试

SD 卡模块的测试主要是通过输出信号进行测试（比如将初始化完成信号 sd_init_done 连接到 LED 灯显示），当观察到 LED 灯点亮时，说明此时 SD 已经初始化完成。使用这种方法进行调试与测试的原因是，对于 SD 卡的调试，很难编写 test_bench 文件，因为在与 SD 卡进行数据交互的过程中需要 SD 卡的响应，而这种响应数据是较难编写成测试文件的，本想使用

VIVADO 自带的在线逻辑分析仪进行测试，但是由于本人能力有限，并不熟悉在线分析仪的操作，于是就选择了较麻烦的调试方法。

5 实验结果

5.1 下板结果

结果如下图所示，VGA 显示数字图片并且用黑框将其框出，数码管显示程序识别出来的数字。

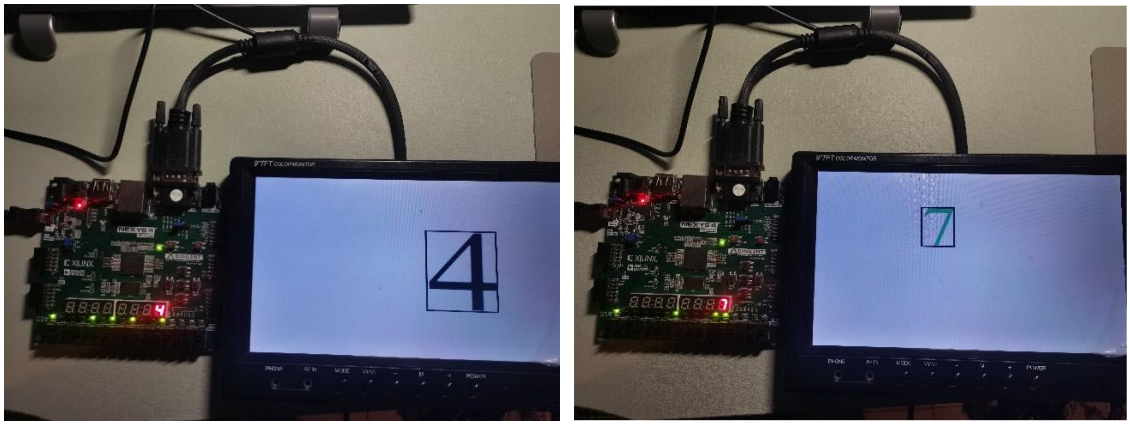


图 28

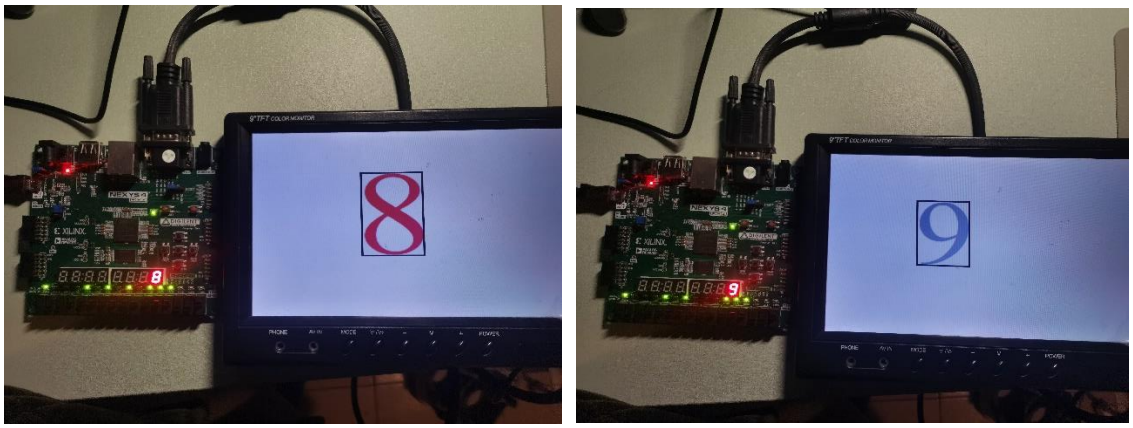


图 29

6 心得体会

6.1 程序调试心得

大作业开始之前，我先翻阅了先前学长学姐的大作业，发现一位学长本打算做一个摄像头

获取图片再储存到 SD 的项目，但是限于精力，只完成了摄像头显示实验。于是我突发奇想想完成他未完成的实验，买来了 SD（TF）卡。一开始研究 SD 卡模块读写协议时一头雾水，根本不知道从何下手，查阅大量资料和博客文章，差不多看了一个星期左右才开始动手。

刚开始写 SD 卡时，才发现这是个大坑。首先，最重要也是最基本的是需要先对 SD 卡进行初始化（对 SD 进行操作时，有两种模式，一种是 SDIO 模式，还有一种是 SPI 模式，由于 SPI 模式较简单，只需要根据 SPI 的端口进行连接所以我选择的是 SPI 模式），利用 SPI 向 SD 卡发送命令和接收 SD 返回的响应数据（见上文 SD 卡初始化模块），最终对 SD 卡进行初始化。SD 卡的初始化模块是我花时间最多的模块，原因之一是其状态较多而且比较复杂，其二是难以调试，由于没法使用在线逻辑分析仪因此有时只能通过输出查看结果并且调试浪费了很多时间。完成了初始化后才可以进行对 SD 卡的读写，读写操作和初始化类似只不过多了发送有效数据和读取有效数据环节。

做完了这些工作本想就直接结束，但发现还有一些时间就想可不可以从 SD 卡读取图片，然后通过查阅资料完成了从 SD 卡读取 bit 图片文件数据然后通过缓存到 RAM 再显示在 VGA 上；但是期间还遇到与一个问题就是 RAM 缓存的问题：直接在自己的模块中定义一个 RAM 空间发现已经超出最大内存限制（一张 RGB565bit 文件大小为 640*480*16），因此参考往届对摄像头图像数据的缓存我使用了 VIVADO 的 IP 核实现。最后发现是不是可以在图片的基础上实现一些功能，首先想的是对图片进行图像处理，由于往届的学长有许多已经将图像处理做的很好了所以我选择了特征识别数字。

数字识别有很大的意义，一些实例比如车牌识别已经被广泛应用起来，我也想借此机会体会数字识别的过程与原理。我选择了最基本的特征识别，直接对缓存在 RAM 中的像素数据进行处理与识别，过程还比较顺利，但是本实验只对印刷体数字识别效果较好，算法与准确性还需要改进。

6.2 课程学习体会

郭老师第一节课就和我们讲了我们为什么要学习硬件，一开始我也对硬件比较抵触，一开始的课程也学的云里雾里，但是通过一个学期的学习，我感觉我还是收获很大，不管是对知识点的理解还是对观念的转变，我认为我已经理解了很多东西。

记得在 BILIBILI 看过一个视频《国产芯片的明显短板：FPGA》，视频中阐述了 FPGA 的

重要性与我国芯片的短板。目前整个中国芯片领域，有的可能只有一些技术、人才、刻蚀机。而缺少的是芯片架构、EDA 软件、原材料、光刻机等设备，怕的自然也就是这些架构不给授权了、EDA 软件没法用了，还有原材料被限制了，以及光刻机等设备无法进口了。这让我对我们为什么要学习硬件有了答案，通过本次大作业也让我对课程所学内容有了更深一层的了解。

参考资料

[1] SD卡数据手册：https://www.sdcard.org/

[2] Nexys4 DDR™ FPGA Board Reference Manual：

https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/start

[3] VGA 参数时序： http://www.tinyvga.com/vga-timing

[4] Micro SD 卡（TF卡） SPI 模式：https://blog.csdn.net/ming1006/article/details/7281597

[5] TF卡资料：http://www.sandisk.com

[6] 基于FPGA的数字识别：https://cloud.tencent.com/developer/article/1528938