笔记 19 DIY 数码相框

一、背景介绍

所谓数码相框，其实用处并不大，在很多人看来它顶多只不过是一个昂贵的花瓶而已。之所以选择 DIY 数码相框，并不是期望可以做个多么像样的产品出来，毕竟消费类数码产品的市场竞争是很激烈的，任何一个成熟产品的成本都会被剥削到最低。市场上的数码相框一般是以带液晶驱动外设的 ARM 处理器为主实现上述的所有功能，而这个工程恰恰相反，所有的功能也都会使用 FPGA 来完成。但是话说回来，FPGA 是硬件，虽然有很多硬件固有特性所具备的优势，但是其设计灵活性方面还是和软件无法媲美的，所以这个工程项目最终实现的数码相框的功能会打一些折扣。和 DIY 逻辑分析仪一样，该项目的重点是希望通过这样的工程来掌握基于 FPGA 的开发流程和设计理念。

特权同学也没有花时间研究过电子城里琳良满目的数码相框，但是以特权同学做过的各种液晶屏的驱动来看，其实 DIY 数码相框也并非难事。一块电路板上无非是集成了一个可以读写 U 盘或是 SD 卡的控制模块，该控制模块内可能还需要完成一些文件系统控制和图片格式的解码；一个可以驱动液晶时序的模块。图 6.4 就是一个简单的数码相框功能框图。

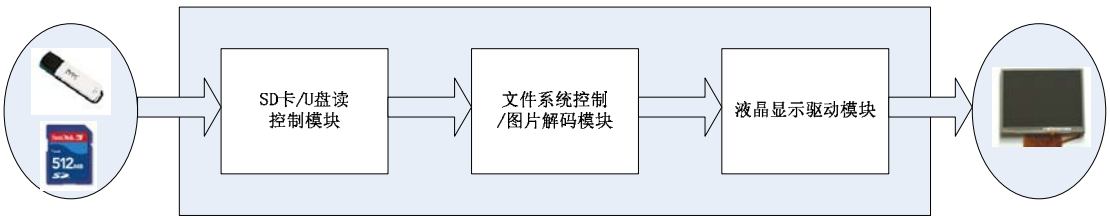


图 6.4 数码相框功能框图

二、功能需求及模块划分

该工程所要实现的数码相框的主要功能点如下所列。

① 使用 cyclone 系列的 EP1C3T144 为基础来实现所有的功能，以 SF-EP1C 开发板作为目标板。

② 从 SD 卡中读取图片（SPI 模式），SD 卡控制不涉及文件操作系统。

③ 使用电脑显示器（VGA）作为图片显示屏幕，工作在 60Hz/800*600 分辨率下，显示色彩为 256 色。

④ FPGA 中实现 bmp 的解码。

⑤ 使用 SDR SDRAM 作为显示图片缓存。

⑥ 循环显示 SD 卡中 10 幅图片。

设计者需要注意的点和《DIY 逻辑分析仪》中提到的内容是类似的，这里不再重复。下面重点看看 FPGA 内部具体的模块如何进行划分。

如图 6.5 所示，该工程的功能框图里明确了各个功能模块，5 个大的功能模块包括：系统时钟与复位模块、SD 卡相关模块（包括 SD 卡控制模块和 SPI 时序产生模块）、数据流控制模块（包括写 SDRAM 缓存 FIFO 模块、读 SDRAM 缓存 FIFO 模块和 BMP 色彩表模块）、SDRAM 控制器模块、VGA 显示驱动模块。

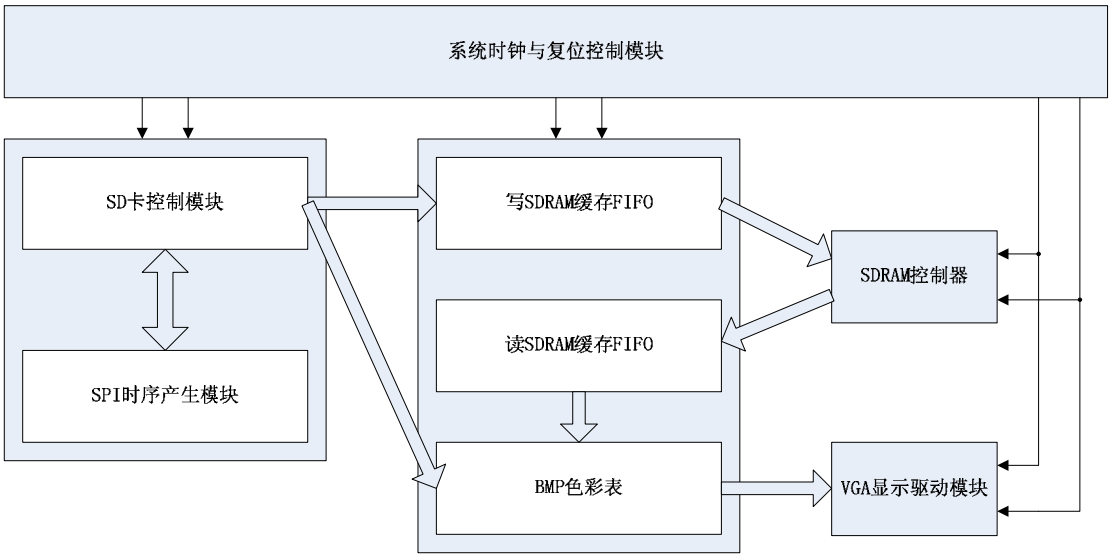


图 6.5 数码相框功能框图

系统时钟与复位模块主要完成 PLL 例化和复位控制。PLL 输出三个时钟：内部 SDRAM 控制器工作的 100MHz 时钟、外部 SDRAM 使用的有一定相位偏移的 100MHz 时钟和其他模块使用的 50MHz 时钟。复位控制部分对复位信号做“异步复位，同步释放”处理，保证系统有一个可靠稳定的复位信号。

SD 卡相关模块有两个子模块，SD 卡控制模块完成 SD 卡的一些基本控制，如 SD 卡的上电初始化、命令发送、数据读取等。FPGA 和 SD 卡之间数据或命令的传输是通过 SPI 口，这部分由 SPI 时序产生模块控制。

数据流控制模块用于衔接 SD 卡、SDRAM 以及 VGA 显示驱动模块。从 SD 卡中读取的 BMP 图片的色彩表数据将被缓存到 BMP 色彩表中，用于显示图片时译码使用。SD 卡的图片数据部分则被送入写 SDRAM 缓存 FIFO 中。读和写 SDRAM 缓存 FIFO 都直接和 SDRAM 控制器连接，他们一齐完成了高速数据的缓冲。由于 SD 卡本身速度较慢，无法满足 VGA 实时的数据扫描需求，所有需要先将 SD 卡中缓存的图片送入 SDRAM 中，VGA 显示器实时的从 SDRAM 中读取数据进行显示。

最后从读 SDRAM 缓存 FIFO 里输出的图片数据经过 BMP 色彩表译码后送到 VGA 显示驱动模块。VGA 显示模块直接驱动显示器进行图片显示。

DIY 数码相框工程源码配合 SF-EP1C 开发板使用，FPGA 管脚定义以及分配如表 6.5 所示。

表 6.5 DIY 数码相框管脚定义及分配

名称	方向	分配	作用
clk	input	PIN16	FPAG 输入时钟信号 25MHz
rst_n	input	PIN144	FPAG 输入复位信号，低电平有效
sdram_clk	output	PIN26	SDRAM 时钟信号
sdram_cke	output	PIN27	SDRAM 时钟有效信号，高电平有效
sdram_cs_n	output	PIN39	SDRAM 片选信号，低电平有效
sdram_ras_n	output	PIN38	SDRAM 行地址选通脉冲，低电平有效
sdram_cas_n	output	PIN37	SDRAM 列地址选通脉冲，低电平有效
sdram_we_n	output	PIN1	SDRAM 写选通信号，低电平有效
sdram_ba[0]	output	PIN40	SDRAM 的 L-Bank 地址线
sdram_ba[1]	output	PIN41	SDRAM 的 L-Bank 地址线
sdram_addr[0]	output	PIN47	SDRAM 地址总线
sdram_addr[1]	output	PIN48	SDRAM 地址总线
sdram_addr[2]	output	PIN49	SDRAM 地址总线
sdram_addr[3]	output	PIN50	SDRAM 地址总线
sdram_addr[4]	output	PIN36	SDRAM 地址总线
sdram_addr[5]	output	PIN35	SDRAM 地址总线
sdram_addr[6]	output	PIN34	SDRAM 地址总线
sdram_addr[7]	output	PIN33	SDRAM 地址总线
sdram_addr[8]	output	PIN32	SDRAM 地址总线
sdram_addr[9]	output	PIN31	SDRAM 地址总线
sdram_addr[10]	output	PIN42	SDRAM 地址总线
sdram_addr[11]	output	PIN28	SDRAM 地址总线
sdram_data[0]	inout	PIN132	SDRAM 数据总线
sdram_data[1]	inout	PIN133	SDRAM 数据总线
sdram_data[2]	inout	PIN134	SDRAM 数据总线
sdram_data[3]	inout	PIN139	SDRAM 数据总线
sdram_data[4]	inout	PIN140	SDRAM 数据总线
sdram_data[5]	inout	PIN141	SDRAM 数据总线
sdram_data[6]	inout	PIN142	SDRAM 数据总线
sdram_data[7]	inout	PIN143	SDRAM 数据总线
sdram_data[8]	inout	PIN11	SDRAM 数据总线

sdram_data[9]	inout	PIN10	SDRAM 数据总线
sdram_data[10]	inout	PIN7	SDRAM 数据总线
sdram_data[11]	inout	PIN6	SDRAM 数据总线
sdram_data[12]	inout	PIN5	SDRAM 数据总线
sdram_data[13]	inout	PIN4	SDRAM 数据总线
sdram_data[14]	inout	PIN3	SDRAM 数据总线
sdram_data[15]	inout	PIN2	SDRAM 数据总线
spi_miso	input	PIN51	SPI 主机输入从机输出数据信号
spi_mosi	output	PIN57	SPI 主机输出从机输入数据信号
spi_clk	output	PIN52	SPI 时钟信号，由主机产生
spi_cs_n	output	PIN58	SPI 从设备使能信号，由主机控制
hsync	Output	PIN61	VGA 行同步信号
vsync	Output	PIN62	VGA 场同步信号
vga_r[2]	Output	PIN74	VGA 色彩
vga_r[1]	Output	PIN73	VGA 色彩
vga_r[0]	Output	PIN72	VGA 色彩
vga_g[2]	output	PIN71	VGA 色彩
vga_g[1]	Output	PIN70	VGA 色彩
vga_g[0]	Output	PIN69	VGA 色彩
vga_b[1]	Output	PIN67	VGA 色彩
vga_b[0]	Output	PIN68	VGA 色彩

三、SPI 接口控制

SPI（Serial Peripheral Interface）即串行外围设备接口，是一种高速、全双工、同步的通信总线。只需要四条信号线即可，节约管脚，同时有利于 PCB 的布局。正是出于这种简单易用的特性，现在越来越多的芯片集成了这种通信协议。

该工程模块的 SPI 接口四条信号线分别定义为 spi_cs_n、spi_clk、spi_miso 和 spi_mosi。其中 spi_cs_n 是控制芯片是否被选中的，只有片选信号有效时（一般为低电平有效),对此芯片的操作才有效。这就使得在同一总线上连接多个 SPI 设备成为可能。spi_clk 是 SPI 同步时钟信号，数据信号在该时钟的控制下逐位进行传输。spi_miso 和 spi_mosi 是主从机进行通信的数据信号，spi_miso 即主机的输入或者说是从机的输出，spi_mosi 即主机的输出或者说是从机的输入。

SPI 的工作模式有两种：主模式和从模式。SPI 总线可以配置成单主单从、单主多从和

互为主从。该工程的 FPGA 是 SPI 主机，SD 卡是从机，处于单主单从模式。因此，FPGA 将控制产生 spi_cs_n 和 spi_clk 的时序。

一般而言，SPI 通信可以配置成四种不同的传输模式。 随便翻看一些内嵌有 SPI 接口外设的处理器的 datasheet 都会提到 CPOL 和 CPHA 这两个参数。如图 6.6 所示，CPOL=1 时，SPI 时钟信号 spi_clk 闲置时总是高电平，发起通信后的第一个时钟沿是下降沿；CPOL=0 时，SPI 时钟信号 spi_clk 闲置时总是低电平，发起通信后的第一个时钟沿是上升沿。而 CPHA 则用于控制数据与时钟的对齐模式，CPHA=1 时，时钟的第一个变化沿（上升沿或者下降沿）数据变化，那么也意味着时钟的第二个沿（与第一个沿相反）锁存数据；CPHA=0 时，时钟的第一个变化沿之前数据变化，那么也意味着时钟的第一个沿锁存数据。

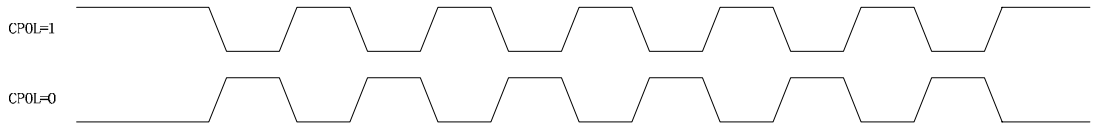


图 6.6 CPOL 配置 SPI 时钟

不同的 CPOL 和 CPHA 可以配置成 4 种 SPI 传输模式，其时序如图 6.7 所示。

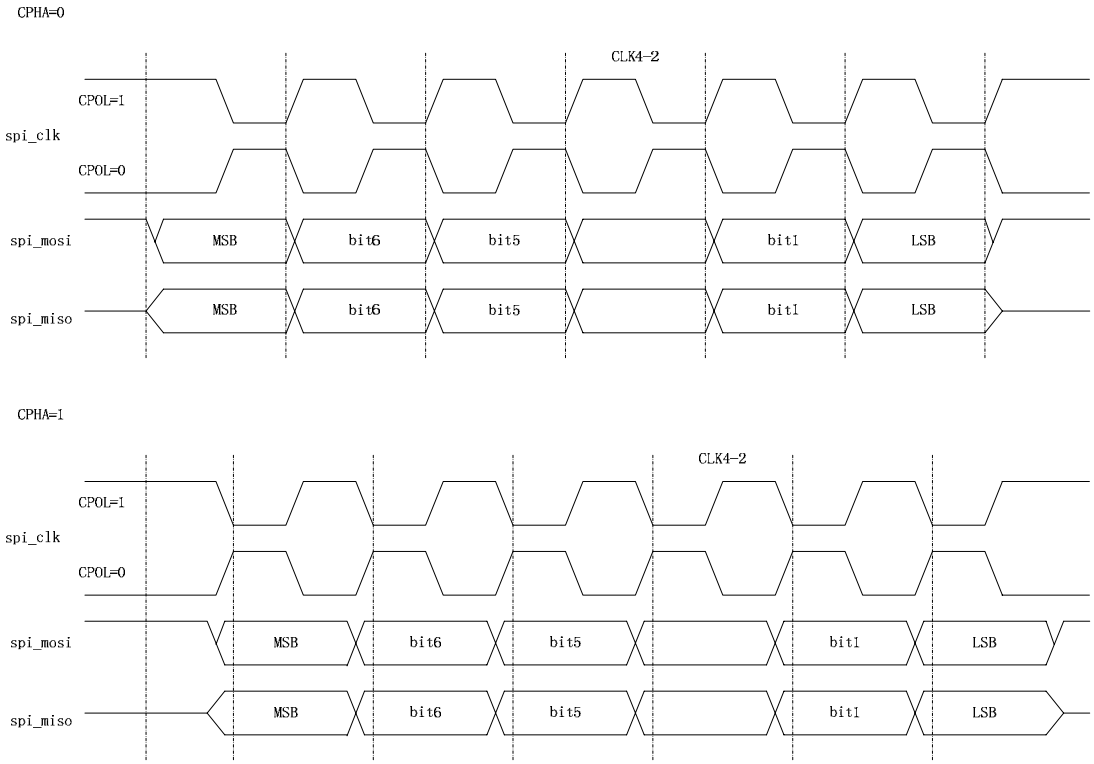


图 6.7 SPI 时序

SD 卡通信可以是 SD 模式或者 SPI 模式，该工程使用了 SD 卡的 SPI 模式进行通信。SD 卡在总线模式中唤醒，在接收复位命令时如果 CS 信号有效（拉低），那么将进入 SPI 模式。如果 SD 卡认为 SD 总线模式是必须的，那么它不会对命令做出响应并继续保持 SD 总线模式。如果需要 SPI 模式，SD 卡将切换到 SPI 模式并发出 SPI 模式下的 R1 响应。

返回 SD 总线模式唯一的方法是重新给 SD 卡上电。在 SPI 模式下，SD 卡协议状态机不

被检测。所有在 SD 总线模式支持的命令在 SPI 模式也是可用的。

SPI 模式下缺省的命令结构/协议是 CRC 检测关闭。随着 SD 卡在 SD 总线模式下上电，CMD0 必须紧跟着一个有效的 CRC 字节。一旦在 SPI 模式下，默认将关闭 CRC。

对于该设计中，SPI 的时序模式为 CPOL=1，CPHA=1，速率为 25Mb/s。SPI 模块的接口定义如表 6.6 所示。

表 6.6 SPI 模块接口定义

名称	方向	描述
clk	input	PLL 产生时钟信号 50MHz
rst_n	input	系统复位信号，低电平有效
spi_miso	input	SPI 主机输入从机输出数据信号
spi_mosi	output	SPI 主机输出从机输入数据信号
spi_clk	output	SPI 时钟信号，由主机产生
spi_tx_en	input	SPI 数据发送使能信号，高有效
spi_tx_rdy	output	SPI 数据发送完成标志位，高有效
spi_rx_en	input	SPI 数据接收使能信号，高有效
spi_rx_rdy	output	SPI 数据接收完成标志位，高有效
spi_tx_db[7:0]	input	SPI 数据发送寄存器
spi_rx_db[7:0]	output	SPI 数据接收寄存器

spi_cs_n 信号由 SD 卡命令驱动模块控制。当需要启动 SPI 模式进行数据传输时，先把待传输的数据放置到 SPI 数据发送寄存器 spi_tx_db[7:0]中，然后将 SPI 发送使能标志位 spi_tx_en 拉高，SPI 发送功能模块被启动。若干个时钟周期后，数据发生完毕，则 SPI 发送完成标志位 spi_tx_rdy 被拉高。此时外部模块检测到 spi_tx_rdy 为高电平，则拉低 spi_tx_en，SPI 模块在 spi_tx_en 拉低后也会清零内部的计数器，此时的 spi_tx_rdy 也会复位，从而完成一次数据传输。接收功能和发送功能类似，只要在 SPI 接口完成标志位 spi_rx_rdy 拉高后读取 spi_rx_db[7:0]的数据即可。

详细设计并不复杂，只要读者用心查看代码就能明白。该模块的关键是如何控制数据流的运转。

四、SD 卡数据存储结构与 FAT16 文件系统

首先，需要说明的一点是，SD 卡和 SDHC 卡其实还是有点区别的，不仅在容量上，对于实际的底层驱动上也是稍有区别的。SD 卡一般容量在 2GB 以下，使用 FAT16 的文件系统；而 SDHC 容量为 4GB 或更大，一般使用 FAT32 的文件系统。该工程针对 SD 卡进行初始化和通信。需要使用 SDHC 卡的朋友可以在此基础上更改代码。

在详细介绍 SD 卡的初始化以及其它相关驱动之前，特权同学将花费一些篇幅对 SD 卡以及 FAT16 文件系统做一些简单的介绍，该工程代码不涉及文件系统的操作，但是了解文件系统将有助于设计者进一步明确 SD 卡的数据存储结构。

1. FAT16 存储原理

当把一部分磁盘空间格式化为FAT文件系统时，FAT文件系统就将这个分区当成整块可分配的区域进行规划，以便于数据的存储。下文将把FAT16部分提取出来，详细进行描述。

FAT16是Microsoft较早推出的文件系统，具有高度兼容性，目前仍然广泛应用于个人电脑尤其是移动存储设备中，FAT16简单来讲由图6.8所示的六部分组成。首先是引导扇区(DBR)，紧随的便是FAT表，FAT表是FAT16用来记录磁盘数据区簇链结构的。FAT将磁盘空间按一定数目的扇区为单位进行划分，这样的单位称为簇。通常情况下，每扇区512字节的原则是不变的。簇的大小一般是2n (n为整数)个扇区的大小，如512B,1K,2K,4K,8K,16K,32K,64K，通常不超过32K。以簇为单位而不以扇区为单位进行磁盘的分配，是因为当分区容量较大时，采用大小为512B的扇区管理会增加FAT表的项数，对大文件存取会增加消耗，使文件系统效率不高。分区的大小和簇的取值是有关系的。

引导扇区	FAT1	FAT2	根文件夹	其他文件夹及所有文件	剩余扇区

图 6.8 FAT16 的组织形式

表 6.7 FAT16 分区大小与对应簇大小

分区空间大小	每个簇的扇区	簇空间大小
0MB-32MB	1	512 个字节
33MB-64MB	2	1k
65MB-128MB	4	2k
129MB-225MB	8	4k
256MB-511MB	16	8k
512MB-1023MB	32	16k
1024MB-2047MB	64	32k
2048MB-4095MB	128	64k

2. 引导扇区的信息：

DBR区(DOS BOOT RECORD)即操作系统引导记录区，通常占用分区的第0扇区共512个字节(特殊情况也要占用其它保留扇区)。在这512个字节中，其实又是由跳转指令，厂商标志和操作系统版本号，BPB(BIOS Parameter Block)，扩展BPB，OS引导程序，结束标志几部分组成。表6.8、表6.9和表6.10分别为FAT16分区上的引导扇区段、BPB字段、BPB扩展字段的重要信息列表。

表6.8 FAT16分区上的引导扇区段

字节位移	字段长度(字节)	字段名称
0x00	3	跳转指令(Jump Instruction)
0x03	8	OEM ID
0x0B	25	BPB
0x24	26	扩展 BPB
0x3E	448	引导程序代码(Bootstrap Code)
0x01FE	4	扇区结束标识符(0x55AA)

表 6.9 FAT16 分区的 BPB 字段

字节位移	字段长度(字节)	例值	描述
0x0B	2	0x0200	扇区字节数(Bytes Per Sector) 硬件扇区的大小。本字段合法的十进制值有 512、1024、2048 和 4096。对大多数磁盘来说，本字段的值为 512。
0x0D	1	0x40	每簇扇区数(Sectors Per Cluster) 一个簇中的扇区数。由于 FAT16 文件系统只能跟踪有限个簇(最多为 65536 个)。因此，通过增加每簇的扇区数可以支持最大分区数。分区的缺省的簇的大小取决于该 分区的大小。本字段合法的十进制值有 1、2、4、8、16、32、64 和 128。导致簇大于 32KB(每扇区字节数*每簇扇区数)的值会引起磁盘错误和软件错误、
0x0e	2	0x0001	保留扇区数(Reserved Sector) 第一个 FAT 开始之前的扇区数，包括引导扇区。本字段的十进制值一般为 1。
0x10	1	0x02	FAT 数(Number of FAT)该分区上 FAT 的副本数。本字段的值一般为 2。
0x11	2	0x0200	根目录项数(Root Entries) 能够保存在该分区的根目录文件夹中的 32 个字节长的文件和文件夹名称项的总数。在一个典型的硬盘上，本字段的值为 512。其中一个项常常被用作卷标号(Volume Label)，长名称的文件和文件夹每个文件使用多个项。文件和文件夹项的最大数一般为 511，但是如果使用的长文件名，往往都达不到这个数。
0x13	2	0x0000	小扇区数(Small Sector) 该分区上的扇区数，表示为 16 位(<65536)。对大于 65536 个扇区的分区来说，本字段的值为 0，而使用大扇区数来取代它。

0x15	1	0xF8	媒体描述符(Media Descriptor)提供有关媒体被使用的信息。这个值如果是 0xF8 则表示硬盘，如果是 0xF0 则表示高密度的 3.5 寸软盘。
0x16	2	0x00FC	每 FAT 扇区数(Sectors Per FAT) 该分区上每个 FAT 所占用的扇区数。计算机利用这个数和 FAT 数以及隐藏扇区数来决定根目录在哪里开始。计算机还可以根据根目录中的项数(512)决定该 分区的用户数据区从哪里开始。
0x18	2	0x003F	每道扇区数(Sectors Per Trark)。
0x1A	2	0x0040	磁头数(Number of head)。
0x1C	4	0x00000 03F	隐藏扇区数(Hidden Sector) 该分区上引导扇区之前的扇区数。在引导序列计算到根目录和数据区的绝对位移的过程中使用了该值。
0x20	4	0x003EF 001	大扇区数(Large Sector) 如果小扇区数字段的值为 0，本字段就包含该 FAT16 分区中的总扇区数。如果小扇区数字段的值不为 0，那么本字段的值为 0。

表 6.9 FAT16 分区的扩展 BPB 字段

字节位移	字段长度(字节)	例值	描述
0x24	1	0x80	物理驱动器号(Physical Drive Number) 与 BIOS 物理驱动器号有关。如果是软盘驱动器则这个值被标识为 0x00，如果是物理硬盘则这个值被标识为 0x80。只有当该设备是一个引导设备时，这个值才有意义。
0x25	1	0x00	保留(Reserved) FAT16 分区一般将本字段的值设置为 0
0x26	1	0x29	扩展引导标签(Extended Boot Signature) 本字段必须要有能被 Windows 2000 所识别的值 0x28 或 0x29
0x27	2	0x52368 BA8	卷序号(Volume Serial Number) 在格式化磁盘时所产生的一个随机序号，它有助于区分磁盘
0x2B	11	"NO NAME"	卷标(Volume Label) 本字段只能使用一次，它被用来保存卷标号。现在，卷标被作为一个特殊文件保存在根目录中
0x36	8	"FAT16"	文件系统类型(File System Type) 根据该磁盘格式，该字段的值可以为 FAT、FAT12 或 FAT16

特权同学所使用的 SD 卡使用 Winhex 查看到的 DBR 区数据如图 6.9 所示。

- 偏移地址 00H，长度 3，内容：EB 3C 90 跳转指令。
- 偏移地址 03H，长度 8，内容：4D 53 44 4F 53 35 2E 30 为厂商标志和 OS 版本号，这里是 MSDOS5.0。
- 偏移地址 0BH,长度 2，内容：00 02。注意这里数据的布局，高地址放高字节，低地址放低字节（数据为小端格式组织），所以数据应该是 0200，即 512。表示的意思是，该磁盘每个扇区有 512 个字节。有的可能是 1024、2048、4096。
- 偏移地址 0DH，长度 1，内容：01。表示的意思是每个簇有 1 个扇区。这个值不能为 0，而且必须是 2 的整数次方，比如 1、2、4、8、16、32、64、128。但是这个值不能使每个簇超过 32KB 字节。
- 偏移地址 0EH，长度 2，内容：08 00。转换一下，就是 00 08，意思是保留区域中的保留扇区数为 8 个。那么就可以知道下面的 FAT1 区的开始的地址就是：0x08*0x200（每个扇区的字节数）=0x1000。
- 偏移地址 10H，长度 1，内容：02。表示此卷中的 FAT 结构的份数为 2，另外一个是备份的。
- 偏移地址 11H，长度 2，内容：00 02。转换一下，就是 0200H，表示根目录项数(Root Entries) 能够保存在该分区的根目录文件夹中的 32 个字节长的文件和文件夹名称项的总数。在一个典型的硬盘上，本字段的值为 512。通过该数据也可以算出根目录后的用户数据区的偏移量地址，即用户数据区首地址=根目录地址+512*32（十进制）。
- 偏移量地址 13H，长度 2，内容：4D ED。转换一下就是 ED4DH，即大约 32MB 的 SD 卡存储量。表示小扇区数(Small Sector) 。该分区上的扇区数，表示为 16 位(<65536)。对大于 65536 个扇区的分区来说，本字段的值为 0，而使用大扇区数来取代它。
- 偏移地址 16H，长度 2，内容：EC 00。转换一下为 00EC，表示每个 FAT 占用的扇区数。那么每个扇区占用的字节数就是 0x00EC*0x200=0x1D800。根据启动区、FAT1、FAT2、根目录、数据区的次序，可以依次计算出它们的地址了。
- 偏移量地址 20H，长度 2，内容：00 00。表示大扇区数(Large Sector) 。如果小扇区数字段的值为 0，本字段就包含该 FAT16 分区中的总扇区数。如果小扇区数字段的值不为 0，那么本字段的值为 0。
- 偏移量地址 36H，长度为 8，内容：46 41 54 31 36 20 20 20，对于 ASCII 码为“FAT16”，表示文件系统类型(File System Type) 根据该磁盘格式，该字段的值可以为 FAT、FAT12 或 FAT16。

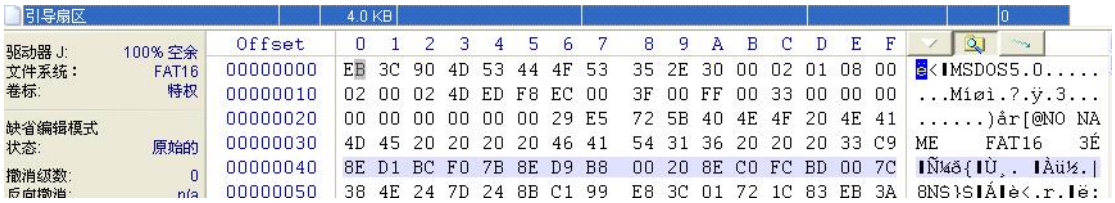


图 6.9 DBR 区数据

根据上面得到的信息可以进行以下的地址推导：

- 启动区地址理所当然是 0x00。
- FAT1 地址是 0x1000。
- FAT2 地址是 0x1000 + 0x1D800 = 0x1E800。
- 根目录区地址是 0x1E800 + 0x1D800 = 0x3C000。

根据上面的计算，可以看看是不是和实际的一致。如图 6.10、图 6.11 和图 6.12 所示。

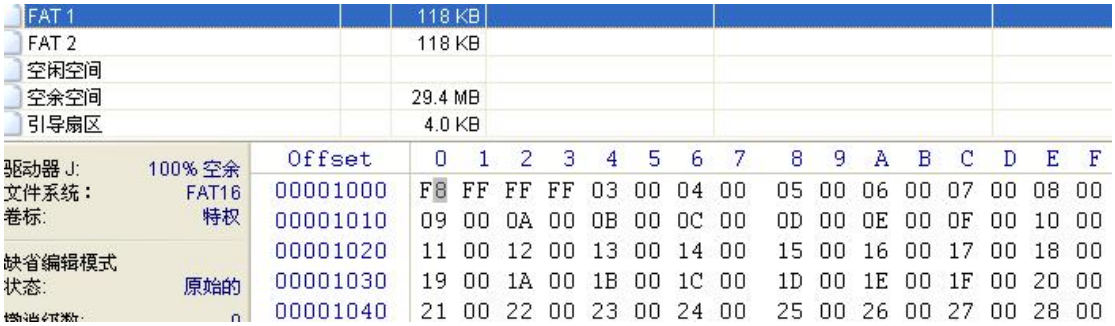


图 6.10 FAT1 地址

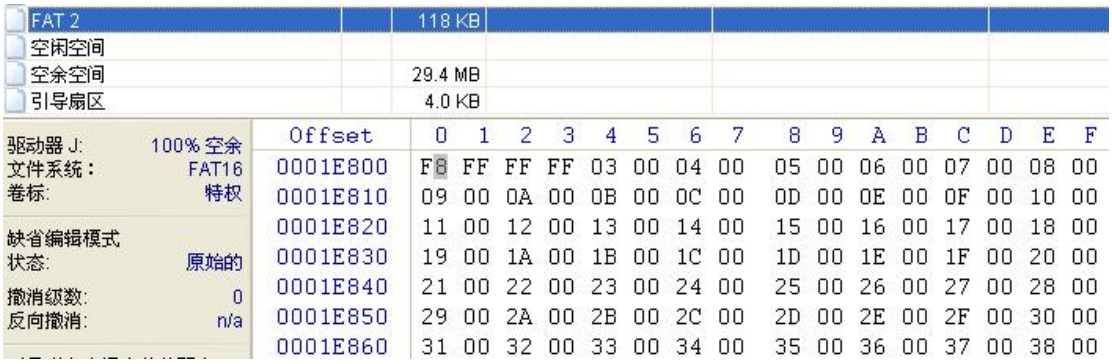


图 6.11 FAT2 地址

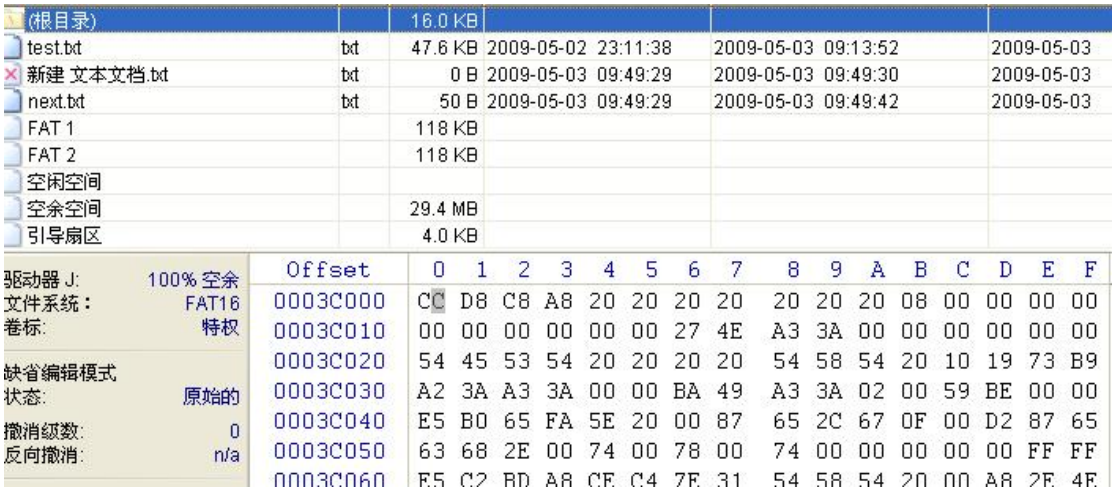


图 6.12 根目录地址

和前面计算的结果是一致的。SD 卡数据每次读取都是以一整个扇区 512 字节为单位。找出这些地址后，可以很方便的找到数据。

3. 分析根目录区的内容

FAT16 文件系统从根目录所占的 32 个扇区之后的第一个扇区开始以簇为单位进行数据的处理，这之前仍以扇区为单位。对于根目录之后的第一个簇，系统并不编号为第 0 簇或第 1 簇（可能是留作关键字），而是编号为第 2 簇，也就是说数据区顺序上的第 1 个簇也是编号上的第 2 簇。（下文所述的簇以此处定义为准。）

FAT 文件系统之所以有 12，16，32 不同的版本之分，其根本在于 FAT 表用来记录任意一簇链接的二进制位数。以 FAT16 为例，每一簇在 FAT 表中占据 2 字节(二进制 16 位)。所以，FAT16 最大可以表示的簇号为 0xFFFF(十进制的 65535)，以 32K 为簇的大小的话，FAT32 可以管理的最大磁盘空间为：32KB×65535=2048MB，这就是为什么 FAT16 不支持超过 2GB 分区的原因。

FAT 表实际上是一个数据表，以 2 个字节为单位，我们暂将这个单位称为 FAT 记录项，通常情况其第 1、2 个记录项(前 4 个字节)用作介质描述。从第三个记录项开始记录除根目录外的其他文件及文件夹的簇链情况。根据簇的表现情况 FAT 用相应的取值来描述，如表 6.10 所示。

表 6.10 FAT16 记录项的取值含义

FAT16 记录项的取值（16 进制）	对应簇的表现情况
0000	未分配的簇
0002~FFEF	已分配的簇
FFF0~FFF6	系统保留
FFF7	坏簇
FFF8~FFFF	文件结束簇

这里使用的是FAT16短文件目录项，每32个字节表示一个文件（文件夹也是），32个字节的表示定义分别如表6.11所示。

表 6.11 FAT16 目录项 32 个字节的表示定义

字节偏移(16 进制)	字节数	定义	
0x0~0x7	8	文件名	
0x8~0xA	3	扩展名	
0xB	1	属性字节	00000000(读写)
			00000001(只读)
			00000010(隐藏)
			00000100(系统)
			00001000(卷标)
			00010000(子目录)
			00100000(归档)
0xC~0x15	10	系统保留	
0x16~0x17	2	文件的最近修改时间	
0x18~0x19	2	文件的最近修改日期	
0x1A~0x1B	2	表示文件的首簇号	
0x1C~0x1F	4	表示文件的长度	

对上表中的一些取值进行说明：

- 对于短文件名，系统将文件名分成两部分进行存储，即主文件名+扩展名。0x0~0x7 字节记录文件的主文件名，0x8~0xA 记录文件的扩展名，取文件名中的 ASCII 码值。不记录主文件名与扩展名之间的“.”。主文件名不足 8 个字符就以空白符(20H)填充，扩展名不足 3 个字符也同样以空白符(20H)填充。0x0 偏移处的取值若为 00H，表明目录项为空；若为 E5H，表明目录项曾被使用，但对应的文件或文件夹已被删除。(这也是误删除后恢复的理论依据)。文件名中的第一个字符若为“.”或“..”表示这个簇记录的是一个子目录的目录项。“.”代表当前目录；“..”代表上级目录(和我们在 dos 或 windows 中的使用意思是一样的，如果磁盘数据被破坏，就可以通过这两个目录项的具体参数推算磁盘的数据区的起始位置，猜测簇的大小等等，故而是比较重要的)。
- 0xB 的属性字段：可以看作系统将 0xB 的一个字节分成 8 位，用其中的一位代表某种属性的有或无。这样，一个字节中的 8 位每位取不同的值就能反映各个属性的不同取值了。如 00000101 就表示这是个文件，属性是只读、系统。
- 0xC~0x15 在原 FAT16 的定义中是保留未用的。在高版本的 WINDOWS 系统中有时也用它来记录修改时间和最近访问时间。那样其字段的意义和 FAT32 的定义是相同的。
- 0x16~0x17 中的时间=小时*2048+分钟*32+秒/2。得出的结果换算成 16 进制填入即可。也就是：0x16 字节的 0~4 位是以 2 秒为单位的量值；0x16 字节的 5~7 位

和 0x17 字节的 0~2 位是分钟；0x17 字节的 3~7 位是小时。

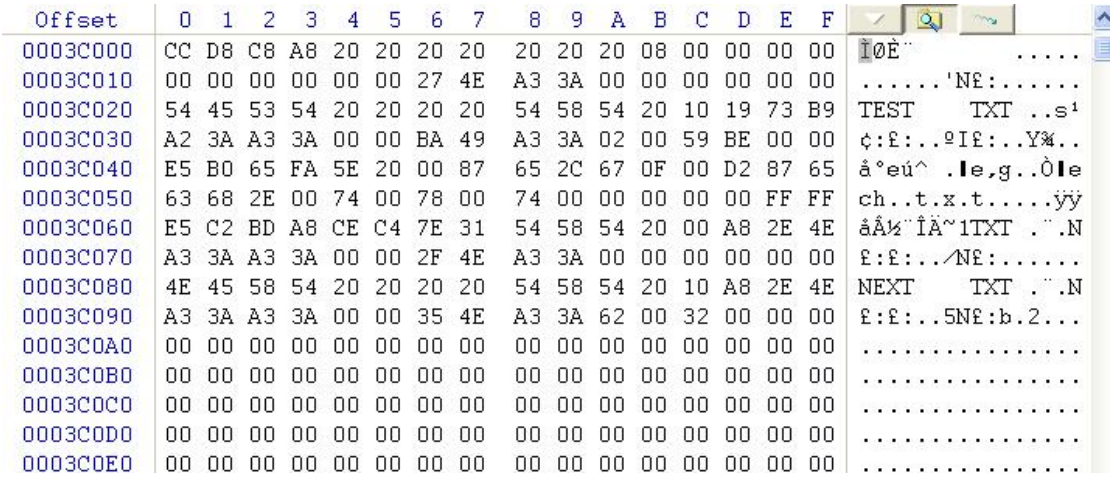
- 0x18~0x19 中的日期=(年份-1980)*512+月份*32+日。得出的结果换算成 16 进制填入即可。也就是：0x18 字节 0~4 位是日期数；0x18 字节 5~7 位和 0x19 字节 0 位是月份；0x19 字节的 1~7 位为年号，原定义中 0~119 分别代表 1980~2099，目前高版本的 Windows 允许取 0~127，即年号最大可以到 2107 年。
- 0x1A~0x1B 存放文件或目录的表示文件的首簇号，系统根据掌握的首簇号在 FAT 表中找到入口，然后再跟踪簇链直至簇尾，同时用 0x1C~0x1F 处字节判定有效性。就可以完全无误的读取文件(目录)了。
- 普通子目录的寻址过程也是通过其父目录中的目录项来指定的，与数据文件(指非目录文件)不同的是目录项偏移 0xB 的第 4 位置 1，而数据文件为 0。
- 对于整个 FAT 分区而言，簇的分配并不完全总是分配干净的。如一个数据区为 99 个扇区的 FAT 系统，如果簇的大小设定为 2 扇区，就会有 1 个扇区无法分配给任何一个簇。这就是分区的剩余扇区，位于分区的末尾。有的系统用最后一个剩余扇区备份本分区的 DBR，这也是一种好的备份方法。
- 早的 FAT16 系统并没有长文件名一说，Windows 操作系统已经完全支持在 FAT16 上的长文件名了。FAT16 的长文件名与 FAT32 长文件名的定义是相同的。

根目录下的数据如图 6.13 所示，其详细含义及说明如下：

- 偏移地址 00H，长度 8，内容：驱动器的名称，8 个字节。这里的 CCD8 对应国标码“特”，而 C8A8 对应国标码“权”，即特权同学给该 SD 卡起的“特权”一名。
- 偏移地址 20H，长度 8，内容：54 45 53 54 20 20 20 20。表示第一个文件名：TEST (空缺部分是空格)。
- 偏移地址 80H，长度 8，内容：4E 45 58 54 20 20 20 20。表示第二个文件名：NEXT (空缺部分是空格)。
- 偏移地址 28H（88H 也一样），长度 3，内容：54 58 54。表示文件类型，为 ASCII 字符表示。
- 偏移地址 2BH（8BH 也一样），长度 1，内容：20。表示文件属性，00000000(读写)；00000001(只读)；00000010(隐藏)；00000100(系统)；00001000(卷标)；00010000(子目录)；00100000(归档)。
- 偏移地址 36H，长度 2，内容为 BA 49。表示时间=小时*2048+分钟*32+秒/2。得出的结果换算成 16 进制填入即可。也就是：36H 字节的 0~4 位是以 2 秒为单位的量值；36H 字节的 5~7 位和 37H 字节的 0~2 位是分钟；37H 字节的 3~7 位是小时。
- 偏移地址 38H，长度 2，内容为 A3 3A。表示日期=(年份-1980)*512+月份*32+日。

得出的结果换算成 16 进制填入即可。也就是：38H 字节 0~4 位是日期数；38H 字节 5~7 位和 39H 字节 0 位是月份；39H 字节的 1~7 位为年号，原定义中 0~119 分别代表 1980~2099，目前高版本的 Windows 允许取 0~127，即年号最大可以到 2107 年。

- 偏移地址 3AH，长度 2，为该文件开始簇号，这里也是用了小端格式组织。转换下为 00 02，根据这个就可以找到文件 TEST.txt 下一个簇号在 FAT1 中的位置了。1000H+02H*02H（因为 2 个字节存一个簇号）= 1004H。
- 偏移地址 3AH，长度 2，为该文件开始簇号，这里也是用了小端格式组织。转换下为 00 62，根据这个就可以找到文件 NEXT.txt 下一个簇号在 FAT1 中的位置了。1000H+62H*02H（因为 2 个字节存一个簇号）= 10C4H。
- 偏移地址 3CH，长度 4，内容：59 BE 00 00。表示文件长度，转换后为 00 00 BE 59 就是 48729 字节。
- 偏移地址 9CH，长度 4，内容：32 00 00 00。表示文件长度，转换后为 00 00 00 32 就是 50 字节。



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0003C000	CC	D8	C8	A8	20	20	20	20	20	20	20	08	00	00	00	00
0003C010	00	00	00	00	00	00	27	4E	A3	3A	00	00	00	00	00	00
0003C020	54	45	53	54	20	20	20	20	54	58	54	20	10	19	73	B9
0003C030	A2	3A	A3	3A	00	00	BA	49	A3	3A	02	00	59	BE	00	00
0003C040	E5	B0	65	FA	5E	20	00	87	65	2C	67	0F	00	D2	87	65
0003C050	63	68	2E	00	74	00	78	00	74	00	00	00	00	00	FF	FF
0003C060	E5	C2	BD	A8	CE	C4	7E	31	54	58	54	20	00	A8	2E	4E
0003C070	A3	3A	A3	3A	00	00	2F	4E	A3	3A	00	00	00	00	00	00
0003C080	4E	45	58	54	20	20	20	20	54	58	54	20	10	A8	2E	4E
0003C090	A3	3A	A3	3A	00	00	35	4E	A3	3A	62	00	32	00	00	00
0003C0A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0003C0B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0003C0C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0003C0D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0003C0E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

图6.13 根目录地址的内容

4. 计算出该文件放置空间

从文件的大小可以计算出，需要占用多少个簇。根据前面的数据，每个簇放1个扇区，每个扇区512个字节，那么一个簇的空间就是512字节了。那么48729字节需要96个簇，这96个簇的开始的地址就可以计算出来了。

如图6.14所示，TEST.txt占用了48KB的空间，NEXT占用了512B的空间。文件是按照整簇来存放的，不够一个簇的大小（由上面算得，一个簇为一个扇区即512B），也要给一个簇的空间。

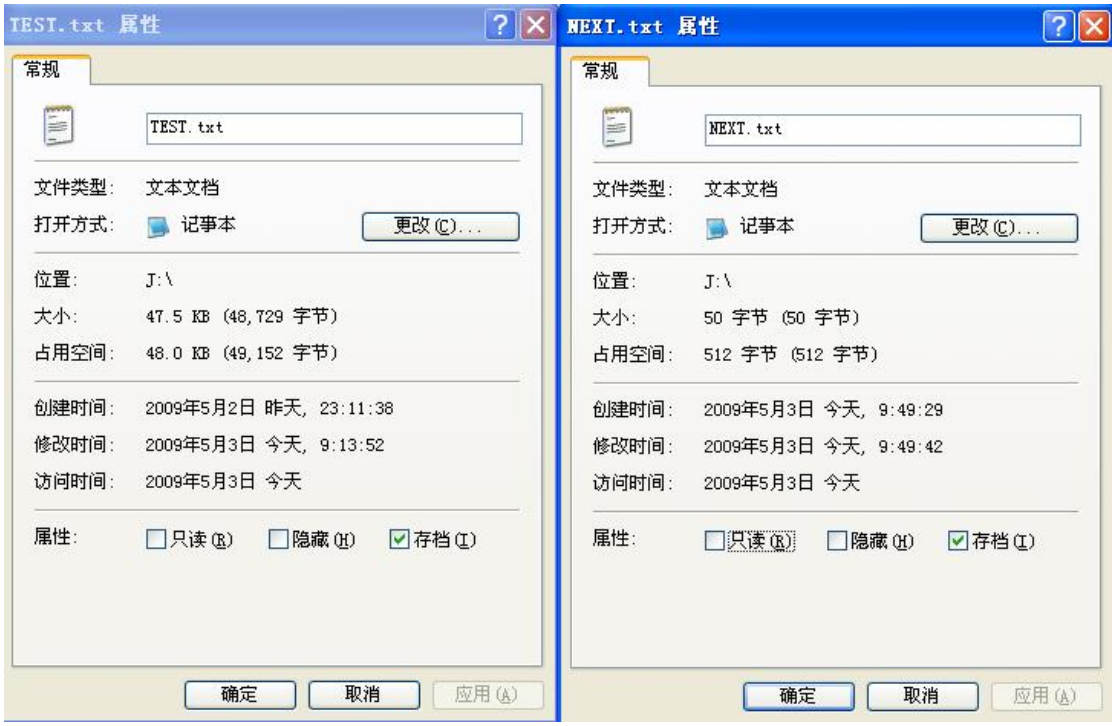


图6.14 文件存储空间占用

如图6.15和图6.16所示，分别为在Winhex下查看到的文件TEST.txt和NEXT.txt的数据。下面就要简单的算算他们的数据存放地址是否与图示一致。

上面已经知道TEST.txt开始簇地址存放在FAT1中的偏移量了：02H，由此可以先计算出TEST.txt的第一簇数据存放地址为：3C000H(根目录地址)+20H*200H(前面提到的用户数据偏移量)+(02H-02H)*01H(1个簇有1个扇区)*200H=40000H。把偏移量-02H意思是簇号在FAT1中存储都是从02H开始的。

TEST.txt的第一个簇数据所在的地址指针的地址存放在FAT1中的：1000H（FAT1起始地址）+02H*02H=1004H。而1004H地址上的数据为：03 00，转换后为0003H，那么由此可以计算出TEST.txt的第二个簇数据所在地址为：3C000H（根目录区地址）+20H*200H(前面提到的用户数据偏移量)+(03H-02H)*01H(1个簇有1个扇区)*200H=40200H（紧接着第一个簇）。依此类推，一直到FAT1中偏移量为C2H处出现了数据FF FF，这表示TEST.txt文件存储结束，那么前面的0061H就是文件最后一个簇偏移量。可以由此算一下文件大小为：
（0061H-0002H+0001H(补偿)）=96个簇，和实际相符。

同样的道理可以算出NEXT.txt文件的存放地址。首地址偏移量为62H，由此可以先计算出NEXT.txt的第一簇数据存放地址为：3C000H(根目录地址)+20H*200H(前面提到的用户数据偏移量)+(62H-02H)*01H(1个簇有1个扇区)*200H=4C000H。而第一个簇数据所在的地址指针的地址存放在FAT1中的：1000H（FAT1起始地址）+62H*02H=10C4H。而10C4H地址上的数据为：FF FF,即结束了，也就是说由于NTXT.txt不满一个簇，那么只能分配到一个簇的地址空间。

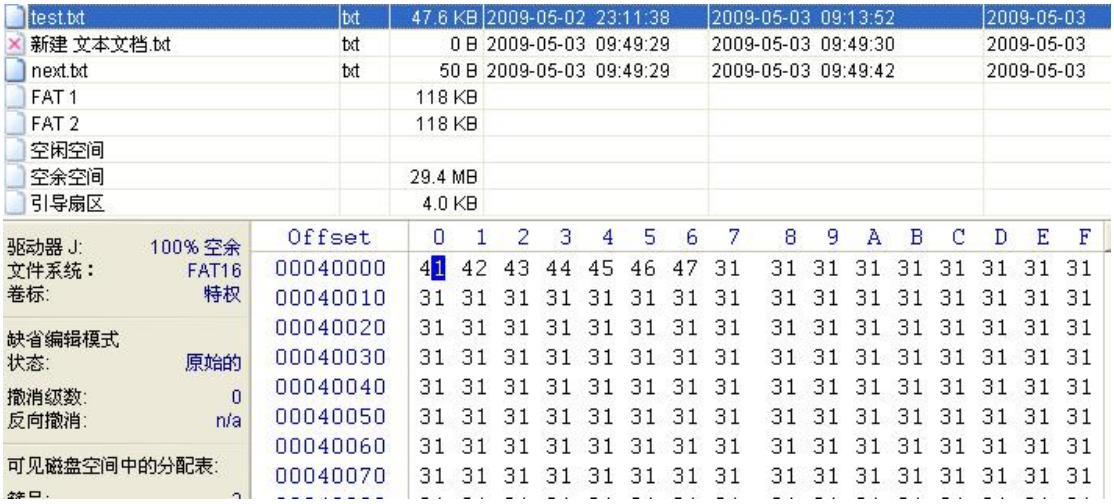


图6.14 test.txt数据

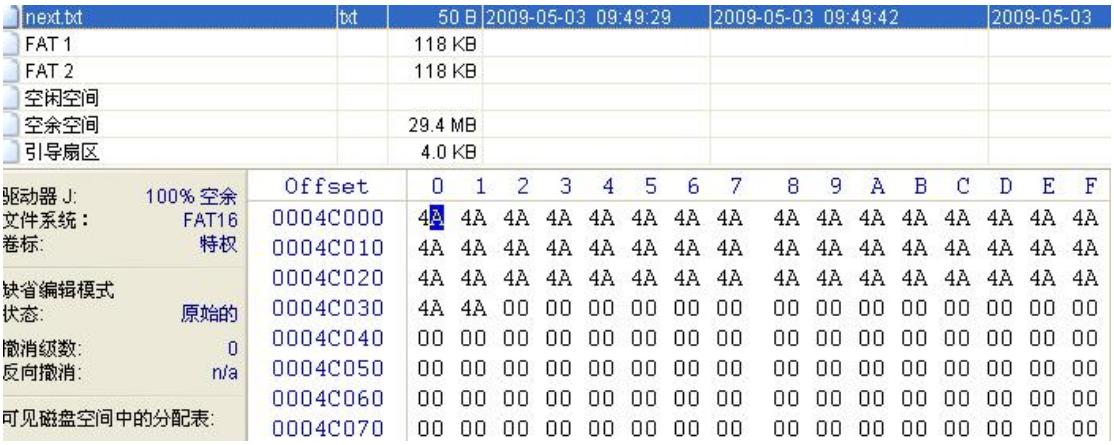


图6.15 next.txt数据

简单的介绍了文件系统和 SD 卡的数据存储结构，将有助于理解后面将涉及到的图片存储，比如一幅图片存储空间大小，第 1 幅和第 2 幅图片之间的地址差等问题。

五、SD 卡初始化及读操作

SD 卡的操作并不简单,需要设计者花心思好好把 SD 卡和 SPI 协议研究透。对于 SD 卡的设计输入和调试，需要事先独立于整个系统工程。特权同学的初步调试是通过串口把读出的 SD 卡数据上传，在确认读取出来的 SD 卡数据正确无误后再和系统的其他模块整合调试，这也是设计中很有效可行的一个设计方法。

这个测试所用的串口模块在最后的系统中是不需要的，但是现在的初步调试会用到它。所以要先对这个串口模块精心包装定制一番，再整合前面设计好 SPI 协议控制模块，能收能发全双工，速率 25Mb/s，标称的最大速度（初步调试时可以考虑先降额一半或更低的速率可能更好一些，这里由于叙述的方便所以直接用 25Mb 的速率，特权同学也是从低速调试到高速的）。SD 卡控制的模块里除了 SPI 协议模块外，当然还得用几个又臭又长的状态机来控制 SD 卡的上电初始化和扇区读取控制。

如果不先理清思路，状态机写得还挺费劲的，想起当初调试 SDRAM 的时候也差不多的样子，这也只能说是 verilog 的局限的，做一些顺序执行的任务比较麻烦（软硬各有利弊）。对于这样的外设控制，有时还真不一定非得是可编程器件管用，软件的优势还是更大一些。

初步测试的模块划分如图 6.16 所示。SD 接收来的数据要送给串口模块发出去，这期间的数据交换就都交给 FIFO 来处理了。输出处理串口发送信号 rs232_tx 外，还有 led[3:0] 连接到 4 个发光二极管，用于调试 SD 卡初始化的状态机时的状态指示。

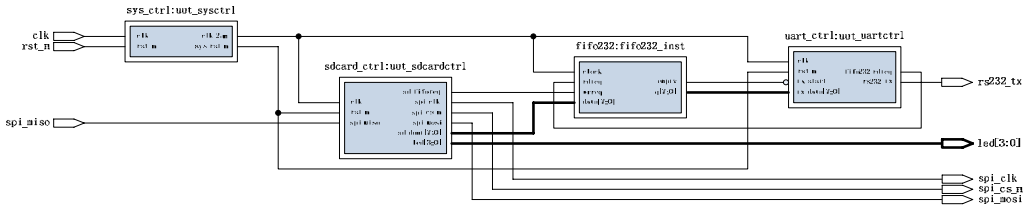


图 6.16 SD 卡调试模块 RTL 视图

特权同学在调试过程中还是出了一些问题，能够读出物理 0 扇区的内容（如图 6.17 所示），用 Winhex 比对无误。但是除了 0 扇区以外再要读取别的扇区，送完 CMD17 和扇区号状态机就等 8’hfe 等死了，当时也搞不明白为什么。网上一搜发现有不少人遇到过同样的问题，只是没看到问题答案。

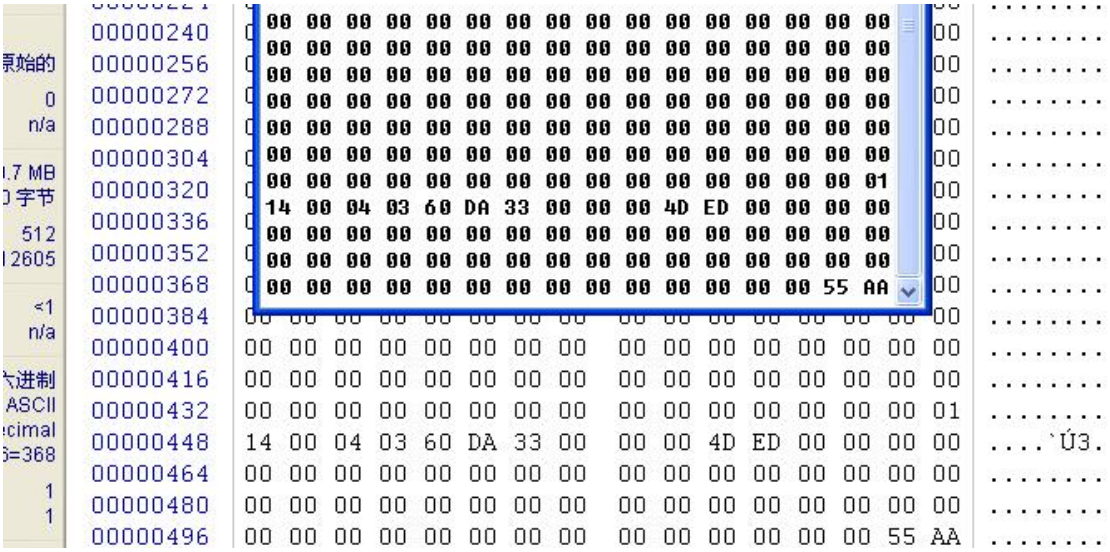


图 6.17 读 SD 卡 0 扇区内容比对

问题最终还是被解决了，其实是因为 CMD 后面跟的 arg 地址是字节地址，而之前写入地址时特权同学一直把这个地址假想成是扇区数了。如果设置每次读 512B 数据，那么这个地址只能是以扇区为单位，即 512B 的倍数，如果不是，结果只能是等不到读数据的起始数据 8’hfe 了。

SD 卡控制模块包括了一个 SD 卡初始化和扇区读取控制的模块和一个 SPI 通信模块，它的具体接口定义和描述如表 6.12 所示。

表 6.12 SD 卡模块接口定义

名称	方向	描述
clk	input	PLL 产生时钟信号 50MHz
rst_n	input	系统复位信号，低电平有效
spi_miso	input	SPI 主机输入从机输出数据信号
spi_mosi	output	SPI 主机输出从机输入数据信号
spi_clk	output	SPI 时钟信号，由主机产生
sd_dout[7:0]	output	从 SD 读出的待放入 FIFO 数据
sd_fifowr	output	sd 读出数据写入 FIFO 使能信号，高有效
sdwrad_clr	output	SDRAM 写控制相关信号清零复位信号，高有效

SD 卡的上电初始化过程可以分为以下五个步骤：

① 适当延时等待 SD 就绪。

② 发送 74+个 spi_clk，且保持 spi_cs_n=1,spi_mosi=1。

③ 发送 CMD0 命令并等待响应 R1=8'h01：将卡复位到 IDLE 状态。

④ 发送 CMD1 命令并等待响应 R1=8'h00：激活卡的初始化进程。

⑤ 发送 CMD16 命令并等待响应 R1=8'h00：设置一次读写 BLOCK 的长度为 512 个字节。

SD 数据读取操作大体分为以下三个步骤：

① 发送命令 CMD17。

② 接收读数据起始令牌 0xfe。

③ 读取 512Byte 数据以及 2Byte 的 CRC。

SD 命令 CMD 发送控制大体分为以下五个步骤：

① 发送 8 个时钟脉冲。

② SD 卡片选 CS 拉低,即片选有效。

③ 连续发送 6 个字节命令。

④ 接收 1 个字节响应数据。

⑤ SD 卡片选 CS 拉高,即关闭 SD 卡。

发送总共 6 个字节命令的格式如图 6.18 所示。

起始位	表示主机	命令	地址	CRC 校验	结束位
1’b0	1’b1	bit5-0	bit31-0	bit6-0	1’b1

Start bit	Host	Command	Argument	CRC	End bit
1'b0	1'b1	bit5-0	bit31-0	bit6-0	1'b1

图 6.18 SD 卡命令格式

详细设计可以参考工程源码，有详尽的注释和说明。

六、SDRAM 控制器设计

SDRAM 控制器的详细设计请参考笔记 17 的第十节。由于之前的设计中模块划分明确，特权同学没有费多少时间就把原来的 SDRAM 控制器模块移植到这个新工程中，再添油加醋做一些定制化的完善后就能使用。

该 SDRAM 控制器的接口定义与说明如表 6.13 所示。

表 6.13 SDRAM 控制器接口定义

名称	方向	描述
clk	input	PLL 产生时钟信号，100MHz
rst_n	input	系统复位信号，低电平有效
sdram_wr_req	input	系统读 SDRAM 请求信号，高电平有效
sdram_rd_req	input	系统写 SDRAM 请求信号，高电平有效
sys_wraddr[21:0]	input	写 SDRAM 时地址暂存器，(bit21-20)L-Bank 地址:(bit19-8)为行地址，(bit7-0)为列地址
sys_rdaddr[21:0]	input	读 SDRAM 时地址暂存器，(bit21-20)L-Bank 地址:(bit19-8)为行地址，(bit7-0)为列地址
sys_data_in[15:0]	input	写 SDRAM 时数据暂存器，4 个突发读写字数据，默认为 00 地址 bit15-0;01 地址 bit31-16;10 地址 bit47-32;11 地址 bit63-48
sdwr_byte[8:0]	input	突发写 SDRAM 字节数（1-256 个）
sdrd_byte[8:0]	input	突发读 SDRAM 字节数（1-256 个）
sdram_wr_ack	output	系统写SDRAM响应信号,作为wrFIFO的输出有效信号，高电平有效
sdram_rd_ack	output	系统读 SDRAM 响应信号，高电平有效
sys_data_out[15:0]	output	读 SDRAM 时数据暂存器
sdram_cke	output	SDRAM 时钟有效信号
sdram_cs_n	output	SDRAM 片选信号，低电平有效

sdram_ras_n	output	SDRAM 行地址选通脉冲，低电平有效
sdram_cas_n	output	SDRAM 列地址选通脉冲，低电平有效
sdram_we_n	output	SDRAM 写允许位，低电平有效
sdram_ba[1:0]	output	SDRAM 的 L-Bank 地址线
sdram_addr[11:0]	output	SDRAM 地址总线
sdram_data[15:0]	inout	SDRAM 数据总线

有一点需要提醒读者注意的，因为之前的 SDRAM 读写设置为 8 个字节。而该工程使用了页模式下进行读写。这个页读写模式也并不简单，没有认真消化 datasheet 很容易就出问题，特权同学就吃了这个亏，调试了好久才搞定它。

特权同学刚开始页操作模式一直没调通，用串口接收传过来的数据总有些问题。因此无奈之下查看了不同厂家的类似型号 SDRAM 的 datasheet，不经意看到一句话，说是 SDRAM 在页操作模式下必须使用突发停止命令停止其操作。否则，地址会不停的从 0-255 循环，而设计者的本意是在第一次 0-255 地址递增期间会分别送 0-255 的数据到数据总线上，但是如果不发突发停止命令，那么在下一次命令到来之前，地址总线会重新不停的进行 0-255 的循环变化，而最后写入 SDRAM 的数据则取决于你最后一个 0-255 地址变化周期内的数据。可想而知，如果 FPGA 不发送停止命令，而假设其操作完成回到 IDEL 状态，那么最后写入的数据肯定就是一串 FF 或者 00，再或者是最后一个写入的一个数据（如果在写入后数据总线不释放的话）。实践证明，结果确实如此。

解决问题之后，就要充分发挥页模式的灵活性和高效性，那么就要做成一个由外部输入数据控制其一次性操作的字节数，也就是说，外部在读写数据前事先控制一个寄存器，往寄存器写入需要操作的字节数，而进入读写操作后，SDRAM 控制器根据外部给出的字节数在适当的时候发出突发停止命令，这样做到了 SDRAM 的读写操作的字节可以在 1-256 范围内灵活调整，增强了通用性。

七、BMP 格式图片显示

1. BMP 格式解析

位图（Bitmap）是 Windows 显示图片的基本格式。在 Windows 下，任何各式的图片文件（包括视频播放）都要转化为位图后才能显示出来。关于这点，做过一些液晶驱动器的特权同学还是深有体会的。不过以前做过的驱动部分大都是人家送数据过来我放到 SRAM 或者 SDRAM 里，然后每次显示从 SRAM 或 SDRAM 搬数据这样的活，相对比较简单，没什么真正意义上的图片结构的成分，只是自己定义好了起始和结束地址就好了。这样的数据也很容易得到，用字模或者图片取模软件转一下就可以。下面就要先好好学习一下 BMP 图片的格式。

下面我们来看看位图文件（*.bmp）的格式。位图文件主要分为如表 6.14 所示的 3 个部

分。

表 6.14 位图文件结构

块名称	windows 结构体定义	大小（Byte）
文件信息头	BITMAPFILEHEADER	14
位图信息头	BITMAPINFOHEADER	40
彩色表	RGBQUAD	一个 800*600 的 8bit 位图的色彩表为：1024B。不同位宽的图片彩色表大小应该是不同的
RGB 颜色阵列	BYTE*	由图像长宽尺寸决定

文件信息头 BITMAPFILEHEADER 的结构体定义如下：

```
typedef struct tagBITMAPFILEHEADER {        /* bmfh */
    UINT bfType;
    DWORD bfSize;
    UINT bfReserved1;
    UINT bfReserved2;
    DWORD bfOffBits;
} BITMAPFILEHEADER;
```

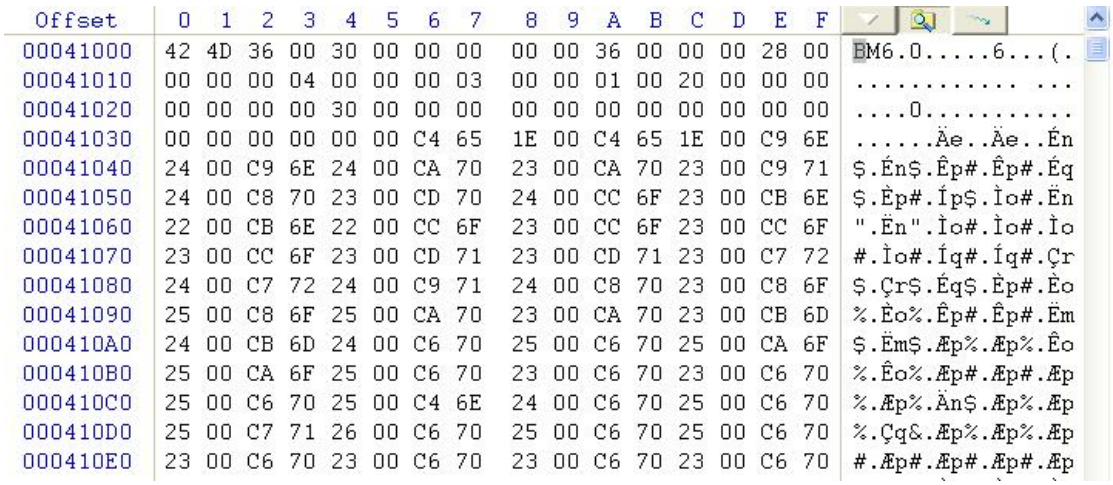
表 6.15 位图文件信息头定义

变量	描述
bfType	说明文件的类型，该值必需是 0x4D42，也就是字符'BM'。
bfSize	说明该位图文件的大小，单位为字节。
bfReserved1	保留，必须设置为 0。
bfReserved2	保留，必须设置为 0。
bfOffBits	说明从文件头开始到实际的图像数据之间字节数的偏移量。这个参数是非常有用的，因为位图信息头和调色板的长度会根据不同情况而变化，所以可以用这个偏移值迅速的从文件中读取到位数据。

看完理论不够，再来看实际的 Winhex 里读取到的一副 BMP 图片的数据是什么，如图 6.19 所示。

偏移地址 0-1 是 4D42H，即“BM”；偏移地址 2-5 是 00300036H，即该图片大小为十六进制的 300036 字节，换算一下大约 3MB 左右，和 winhex 里的根目录的数值是一样的；偏移地址 6-7、8-9 都是 0000H；偏移地址 a-d 是 00000036H，也就是说从图片开始字节地址 41000H 往后的偏移量为 36H 字节的数据才是真正的图片数据。认真算一下会发现前 36H 地址存放的

54 个字节数据正好是文件信息头（14B）和位图信息头（40B）的数据。



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00041000	42	4D	36	00	30	00	00	00	00	00	36	00	00	00	28	00	BM6.0.....6...(.
00041010	00	00	00	04	00	00	00	03	00	00	01	00	20	00	00	00	....
00041020	00	00	00	00	30	00	00	00	00	00	00	00	00	00	00	00	....0.....
00041030	00	00	00	00	00	00	C4	65	1E	00	C4	65	1E	00	C9	6E	......Äe..Äe..Én
00041040	24	00	C9	6E	24	00	CA	70	23	00	CA	70	23	00	C9	71	$.Én$.Êp#.Êp#.Éq
00041050	24	00	C8	70	23	00	CD	70	24	00	CC	6F	23	00	CB	6E	$.Èp#.Íp$.Ìo#.Ën
00041060	22	00	CB	6E	22	00	CC	6F	23	00	CC	6F	23	00	CC	6F	".Ën".Ìo#.Ìo#.Ìo
00041070	23	00	CC	6F	23	00	CD	71	23	00	CD	71	23	00	C7	72	#.Ìo#.Íq#.Íq#.Çr
00041080	24	00	C7	72	24	00	C9	71	24	00	C8	70	23	00	C8	6F	$.Çr$.Éq$.Èp#.Èo
00041090	25	00	C8	6F	25	00	CA	70	23	00	CA	70	23	00	CB	6D	%.Èo%.Êp#.Êp#.Ëm
000410A0	24	00	CB	6D	24	00	C6	70	25	00	C6	70	25	00	CA	6F	$.Ëm$.Æp%.Æp%.Êo
000410B0	25	00	CA	6F	25	00	C6	70	23	00	C6	70	23	00	C6	70	%.Êo%.Æp#.Æp#.Æp
000410C0	25	00	C6	70	25	00	C4	6E	24	00	C6	70	25	00	C6	70	%.Æp%.Än$.Æp%.Æp
000410D0	25	00	C7	71	26	00	C6	70	25	00	C6	70	25	00	C6	70	%.Çq&.Æp%.Æp%.Æp
000410E0	23	00	C6	70	23	00	C6	70	23	00	C6	70	23	00	C6	70	#.Æp#.Æp#.Æp#.Æp

图 6.19 实际 BMP 文件信息头

位图信息头 BITMAPINFOHEADER 的结构体定义如下：

```
typedef struct tagBITMAPINFOHEADER {        /* bmih */
    DWORD biSize;
    LONG biWidth;
    LONG biHeight;
    WORD biPlanes;
    WORD biBitCount;
    DWORD biCompression;
    DWORD biSizeImage;
    LONG biXPelsPerMeter;
    LONG biYPelsPerMeter;
    DWORD biClrUsed;
    DWORD biClrImportant;
} BITMAPINFOHEADER;
```

表 6.16 位图信息头定义

变量	描述
biSize	说明 BITMAPINFOHEADER 结构所需要的字数。
biWidth	说明图像的宽度，以象素为单位。
biHeight	说明图像的高度，以象素为单位。注：这个值除了用于描述图像的高度之外，它还有另一个用处，就是指明该图像是倒向的位图，还是正向的位图。如果该值是一个正数，说明图像是倒向的，如果该值是一个负数，则说明图像是正向的。大多数的 BMP 文件都是倒向的位图，也就是时，高度值是一个正数。

biPlanes	为目标设备说明位面数，其值将总是被设为 1。
biBitCount	说明比特数/象素，其值为 1、4、8、16、24、或 32。但是由于我们平时用到的图像绝大部分是 24 位和 32 位的，所以我们只讨论这两类图像。
biCompression	说明图像数据压缩的类型，我们只讨论没有压缩的类型：BI_RGB。
biSizeImage	说明图像的大小，以字节为单位。当用 BI_RGB 格式时，可设置为 0。
biXPelsPerMeter	说明水平分辨率，用象素/米表示。
biYPelsPerMeter	说明垂直分辨率，用象素/米表示。
biClrUsed	说明位图实际使用的彩色表中的颜色索引数（设为 0 则说明使用所有调色板项）。
biClrImportant	说明对图像显示有重要影响的颜色索引的数目，如果是 0，表示都重要。

再来看如图 6.20 所示偏移地址 0eH 开始的信息头的数据。偏移地址 0e-11H 为 00000028H，即 BITMAPINFOHEADER 结构所需要的字数为 28H；偏移地址 12-15H 为 00000400H（即 1024），偏移地址 16-19H 为 00000300H（即 768），这两个参数对应图片的像素是 1024*768；偏移地址 1AH-1BH 是 0001H，即 biPlanes=1；偏移地址 1CH-1DH 是 0020H，即该 32bit/像素；偏移地址 1eH-21H 是 00000000H，应该是表示没有压缩的图像；偏移地址 22H-25H 是 00300000H；表示图像大小为 3MB；偏移地址 26H-29H 是 00000000H，偏移地址 2aH-2dH 是 00000000H；偏移地址 2eH-31H 是 00000000H；偏移地址 32H-35H 是 00000000H。

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00041000	42	4D	36	00	30	00	00	00	00	00	36	00	00	00	28	00	BM6.0.....6...(.
00041010	00	00	00	04	00	00	00	03	00	00	01	00	20	00	00	00	....
00041020	00	00	00	00	30	00	00	00	00	00	00	00	00	00	00	00	....0.....
00041030	00	00	00	00	00	00	C4	65	1E	00	C4	65	1E	00	C9	6E	......Äe..Äe..Én
00041040	24	00	C9	6E	24	00	CA	70	23	00	CA	70	23	00	C9	71	$.Én$.Êp#.Êp#.Éq
00041050	24	00	C8	70	23	00	CD	70	24	00	CC	6F	23	00	CB	6E	$.Èp#.Íp$.Ìo#.Ën
00041060	22	00	CB	6E	22	00	CC	6F	23	00	CC	6F	23	00	CC	6F	".Ën".Ìo#.Ìo#.Ìo
00041070	23	00	CC	6F	23	00	CD	71	23	00	CD	71	23	00	C7	72	#.Ìo#.Íq#.Íq#.Çr
00041080	24	00	C7	72	24	00	C9	71	24	00	C8	70	23	00	C8	6F	$.Çr$.Éq$.Èp#.Èo
00041090	25	00	C8	6F	25	00	CA	70	23	00	CA	70	23	00	CB	6D	%.Èo%.Êp#.Êp#.Ëm
000410A0	24	00	CB	6D	24	00	C6	70	25	00	C6	70	25	00	CA	6F	$.Ëm$.Æp%.Æp%.Êo
000410B0	25	00	CA	6F	25	00	C6	70	23	00	C6	70	23	00	C6	70	%.Êo%.Æp#.Æp#.Æp
000410C0	25	00	C6	70	25	00	C4	6E	24	00	C6	70	25	00	C6	70	%.Æp%.Än$.Æp%.Æp
000410D0	25	00	C7	71	26	00	C6	70	25	00	C6	70	25	00	C6	70	%.Çq&.Æp%.Æp%.Æp
000410E0	23	00	C6	70	23	00	C6	70	23	00	C6	70	23	00	C6	70	#.Æp#.Æp#.Æp#.Æp

图 6.20 实际 BMP 位图信息头

下面对 biBitCount 做一些补充说明。

- biBitCount=1 表示位图最多有两种颜色，缺省情况下是黑色和白色，可以自己定义这两种颜色。图像信息头的调色板中将有两个调色板项，称为索引 0 和索引 1。图像数据阵列中的每一位表示一个象素。如果一个位是 0，显示时就使用索引

0 的 RGB 值，如果位是 1，则使用索引 1 的 RGB 值。

- biBitCount=4 表示位图最多有 16 种颜色。每个象素用 4 位表示，并用这 4 位作为彩色表的表项来查找该象素的颜色。例如，如果位图中的第一个字节为 0x1F，它表示有两个象素，第一象素的颜色就在彩色表的第 2 表项中查找，而第二个象素的颜色就在彩色表的第 16 表项中查找。此时，调色板中缺省情况下会有 16 个 RGB 项。对应于索引 0 到索引 15。
- biBitCount=8 表示位图最多有 256 种颜色。每个象素用 8 位表示，并用这 8 位作为彩色表的表项来查找该象素的颜色。例如，如果位图中的第一个字节为 0x1F，这个象素的颜色就在彩色表的第 32 表项中查找。此时，缺省情况下，调色板中会有 256 个 RGB 项，对应于索引 0 到索引 255。
- biBitCount=16 表示位图最多有 216 种颜色。每个色素用 16 位（2 个字节）表示。这种格式叫作高彩色，或叫增强型 16 位色，或 64K 色。它的情况比较复杂，当 biCompression 成员的值是 BI_RGB 时，它没有调色板。16 位中，最低的 5 位表示蓝色分量，中间的 5 位表示绿色分量，高的 5 位表示红色分量，一共占用了 15 位，最高的一位保留，设为 0。这种格式也被称作 555 16 位位图。如果 biCompression 成员的值是 BI_BITFIELDS，那么情况就复杂了，首先是原来调色板的位置被三个 DWORD 变量占据，称为红、绿、蓝掩码。分别用于描述红、绿、蓝分量在 16 位中所占的位置。在 Windows 95（或 98）中，系统可接受两种格式的位域：555 和 565，在 555 格式下，红、绿、蓝的掩码分别是：0x7C00、0x03E0、0x001F，而在 565 格式下，它们则分别为：0xF800、0x07E0、0x001F。你在读取一个像素之后，可以分别用掩码"与"上像素值，从而提取出想要的颜色分量（当然还要再经过适当的左右移操作）。在 NT 系统中，则没有格式限制，只不过要求掩码之间不能有重叠。（注：这种格式的图像使用起来是比较麻烦的，不过因为它的显示效果接近于真彩，而图像数据又比真彩图像小的多，所以，它更多的被用于游戏软件）。
- biBitCount=24 表示位图最多有 2 的 24 次方种颜色。这种位图没有调色板（bmiColors 成员尺寸为 0），在位数组中，每 3 个字节代表一个象素，分别对应于颜色 R、G、B。
- biBitCount=32 表示位图最多有 232 种颜色。这种位图的结构与 16 位位图结构非常类似，当 biCompression 成员的值是 BI_RGB 时，它也没有调色板，32 位中有 24 位用于存放 RGB 值，顺序是：最高位—保留，红 8 位、绿 8 位、蓝 8 位。这种格式也被成为 888 32 位图。如果 biCompression 成员的值是 BI_BITFIELDS 时，原来调色板的位置将被三个 DWORD 变量占据，成为红、绿、蓝掩码，分别用于描述红、绿、蓝分量在 32 位中所占的位置。在 Windows 95(or 98)中，系统只

接受 888 格式，也就是说三个掩码的值将只能是：0xFF0000、0xFF00、0xFF。而在 NT 系统中，你只要注意使掩码之间不产生重叠就行。（注：这种图像格式比较规整，因为它是 DWORD 对齐的，所以在内存中进行图像处理时可进行汇编级的代码优化（简单））。

彩色表包含的元素与位图所具有的颜色数相同，象素的颜色用 RGBQUAD 结构来定义。对于 24 位真彩色图像就不使用彩色表（同样也包括 16 位、和 32 位位图），因为位图中的 RGB 值就代表了每个象素的颜色。彩色表中的颜色按颜色的重要性排序，这可以辅助显示驱动程序为不能显示足够多颜色数的显示设备显示彩色图像。RGBQUAD 结构描述由 R、G、B 相对强度组成的颜色，定义如下：

```
typedef struct tagRGBQUAD { /* rgbq */
    BYTE rgbBlue;
    BYTE rgbGreen;
    BYTE rgbRed;
    BYTE rgbReserved;
} RGBQUAD;
```

表 6.16 位图色彩表定义

变量	描述
rgbBlue	指定蓝色强度
rgbGreen	指定绿色强度
rgbRed	指定红色强度
rgbReserved	保留

有关 RGB 三色空间大家可能都很熟悉了，在 Windows 下 RGB 颜色阵列存储的格式其实是 BGR。也就是说，对于 24 位的 RGB 位图像素数据格式是：

蓝色 B 值	绿色 G 值	红色 R 值

对于 32 位的 RGB 位图像素数据格式是：

蓝色 B 值	绿色 G 值	红色 R 值	透明通道 A 值

透明通道也称 Alpha 通道，该值是该像素点的透明属性，取值在 0（全透明）到 255（不透明）之间。对于 24 位的图像来说，因为没有 Alpha 通道，即整个图像都不透明。

紧跟在彩色表之后的是图像数据字节阵列。图像的每一扫描行由表示图像象素的连续的字节组成，每一行的字节数取决于图像的颜色数目和用象素表示的图像宽度。扫描行是由底向上存储的，这就是说，阵列中的第一个字节表示位图左下角的象素，而最后一个字节表示位图右上角的象素。

2. 基于 verilog 的 BMP 解码设计

补充讲解了 BMP 图片的基础知识后，下面要真枪实弹的看看如何对 BMP 格式的图片进行解码，然后转换为最后可以显示到液晶屏上的数据。虽然 BMP 格式内的数据本身就是 RGB 格式的，但是由于它存储的顺序是从图片的左下角开始右上角结束，而且对于如 8bit 的位图还有色彩表的概念，而送到液晶显示器一端的数据一般是以从左到右、从上到下的顺序扫描数据的，所以 BMP 格式的解码还是有点文章可作的。

SD 卡控制模块直接寻址到某一幅图片后将其发送到数据流控制模块，同时会有一个图片数据有效标志位 wrf_wrreq 做指示，wrf_wrreq 只在每次发送的数据有效后保持一个时钟周期高脉冲，从而让数据流控制模块对该数据做相应处理。因此一副分辨率为 800*600 的 8bit 位宽图片共有 481078 字节（54 个字节信息头数据、1024 字节色彩表数据和 480000 字节图像数据）的数据，在读取 SD 卡过程中，wrf_wrreq 也就会产生对应的 481078 个单时钟周期的高脉冲。

图片数据处理的大体流向如图 6.21 所示。由于该设计中只处理 BMP 格式图片，所以对文件信息头和位图信息头就不做判断处理，色彩表数据存储到 FPGA 内部 IP core 例化的一个 RAM(rgb_ram)中，将 1024 字节的色彩表译码为 256 字节的不同色彩数据。两个名为 wrfifo 和 rdfifo 的 FIFO 和 SDRAM 控制模块接口。图像数据首先被送入 wrfifo 中进行缓存，当需要实时显示当前图像的时候，数据再通过 rdfifo 读出。

从 rdfifo 出来的图像数据再通过 rgb_ram 译码后送到后端的 VGA 驱动模块显示。

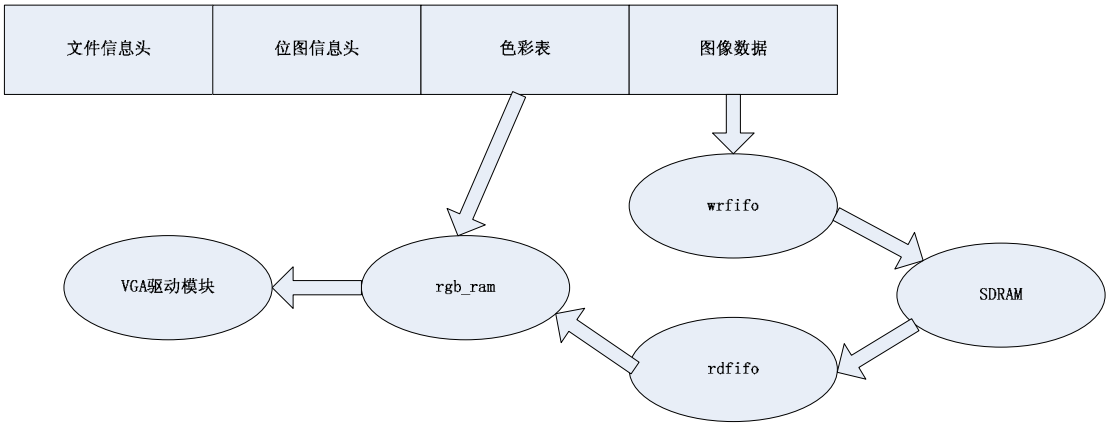


图 6.21 图片数据流示意图

数据流控制模块和 VGA 显示驱动模块的接口定义与描述分别如表 6.17 和表 6.18 所示。

表 6.17 数据流控制模块接口定义

名称	方向	描述
clk_50m	input	PLL 输出 50MHz 时钟
clk_100m	input	PLL 输出 100MHz 时钟
rst_n	input	系统复位信号，低有效
wrf_wrreq	input	（用于 SDRAM 数据写入缓存的）FIFO 写请求信

		号，高电平有效
sdram_wr_ack	input	系统写SDRAM响应信号,作为wrFIFO的输出有效信号，高电平有效
wrf_din[15:0]	input	（用于 SDRAM 数据写入缓存的）FIFO 写数据总线
rdf_rdreq	input	（用于 SDRAM 数据读出缓存的）FIFO 读请求信号，高电平有效
sdram_rd_ack	input	系统读SDRAM响应信号,作为rdFIFO的输出有效信号，高电平有效
sys_data_out[15:0]	input	（用于 SDRAM 数据读出缓存的）FIFO 写数据总线
sdram_rd_req	output	系统读 SDRAM 请求信号，高电平有效
vga_valid	input	用于使能 SDRAM 读数据单元进行寻址或地址清零，高电平有效
sdwrad_clr	input	SDRAM 写入控制寄存器的清零复位信号，高电平有效
sdram_wr_req	output	系统写 SDRAM 请求信号
sys_data_in[15:0]	output	（用于 SDRAM 数据写入缓存的）FIFO 读数据总线，即写 SDRAM 时的数据暂存器
sys_wraddr[21:0]	output	写 SDRAM 时地址暂存器，(bit21-20)L-Bank 地址:(bit19-8)为行地址，(bit7-0)为列地址
sys_rdaddr[21:0]	output	读 SDRAM 时地址暂存器，(bit21-20)L-Bank 地址:(bit19-8)为行地址，(bit7-0)为列地址
dis_data[7:0]	output	VGA 显示数据

表 6.18 VGA 显示驱动模块接口定义

名称	方向	描述
clk	input	PLL 输出 50MHz 时钟
rst_n	input	系统复位信号，低有效
dis_data[7:0]	input	VGA 显示数据
disp_ctrl	input	外部控制 LCD 显示使能信号，高电平有效
vga_valid	output	用于使能 SDRAM 读数据单元进行寻址或地址清

		零，高电平有效
rdf_rdreq	output	SDRAM 数据读出缓存 FIFO 数据输出请求，高电平有效
hsync	output	VGA 行同步信号
vsync	output	VGA 场同步信号
vga_r[2:0]	output	VGA 色彩
vga_g[2:0]	output	VGA 色彩
vga_b[1:0]	output	VGA 色彩

整个设计工程的最终效果是能够将存储在 SD 卡里的 10 幅图片循环显示。其中一幅 256 色的图片显示效果如图 6.22 所示。

图 6.22 256 色图片显示效果

欢迎加入 EDN 网站 FPGA/CPLD 助学小组 http://group.ednchina.com/1375/

购买 BJ-EPM240 CPLD 学习板 http://group.ednchina.com/1375/23842.aspx

购买 SF-EP1C FPGA 开发板 http://group.ednchina.com/1375/27650.aspx

北航出版社将于 2010 年 3 月份前后出版《深入浅出玩转 FPGA》一书，欢迎各位网友到时购买，作为本视频和学习板/开发板的参考教材

特权

2009.11.11