基于 DiskOnChip 2000 的驱动程序设计及应用

陈纪东,杨 斌,朱六兵
(西南交通大学 计算机与通信工程学院,四川 成都 610031)

摘 要:介绍了 Nand Flash 和 DiskOnChip 2000 的原理及读写控制,在 S3C4510B 开发板上,为 uClinux 系统实现了 DiskOnChip 2000 的设备驱动程序,引入了专为 Flash 设计的 YAFFS 文件系统。

关键词:NAND Flash DOC uClinux YAFFS

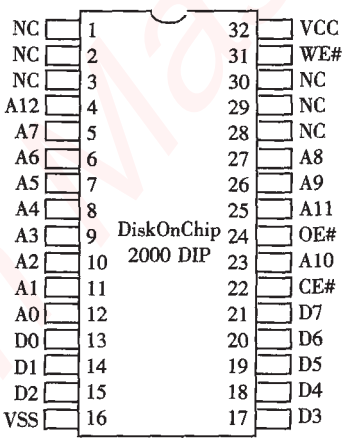
随着信息技术的飞速发展,形式多样的数字化产品已经成为继 PC 机后的信息处理工具。在这种数字化潮流下,嵌入式系统便成为当前研究和应用的热点之一。由于嵌入式系统的应用要求及成本因素决定了嵌入式系统在系统资源,包括硬件资源和软件资源方面都是非常精简和高效的。因此在嵌入式系统中的存储设备一般不会采用硬盘等大容量高功耗设备,而使用诸如 Flash、EPROM 等存储介质。但一般的闪存器(Flash Memory)只包含存储器部分,而不含控制器。M-system 公司推出的DiskOnChip 2000 系列是新一代闪存磁盘,为标准 32 脚 DIP 封装,它与标准的 E^2PROM 完全兼容,并且还有表面焊接封装(SMT、TSOP-Ⅱ)形式。它将控制器与存储器封装在同一块晶片上,因此不需要任何总线、插槽或连接器,只要在 CPU 主板上有一个标准的 32 脚 DIP 插座即可。这种存储器适用于嵌入式装置及要求高效能而体积小的产品。如 Internet 网际网络设备、Smart 电话等。因此,本文讨论 DiskOnChip 在 S3C4510B 开发板上、uClinux 系统下的设备驱动程序的编写和应用。

1 DiskOnChip 2000 介绍

1.1 工作原理

DiskOnChip 可以使系统的写入速度超过每秒 400KB,读取速度超过每秒 1.2MB,另外,不论向任何方向传输,都可以达到每秒 13.3MB 的传输速率。它还带具有错误检测/错误修正(EDC/ECC)功能的控制器。它支持不同的操作系统,也可以在它不支持的操作系统及根本没有操作系统的环境下工作。

图 1 是 DiskOnChip 2000 的引脚和内部方框图。系统接口单元为 DOC2000 提供了类似 SRAM 的接口,使之能够



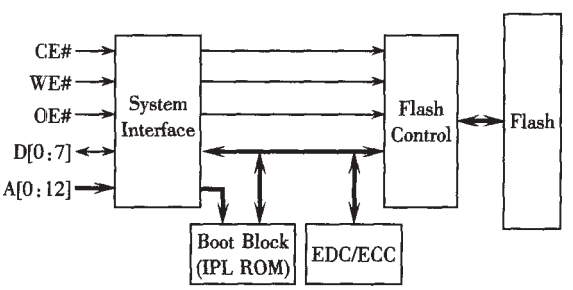


图 1 DiskOnchip2000 的引角和内部图

(接上页)

2001;9(1)
10 Syrdal A,Stylianou Y,Garisson L et al.TD-PSOLA versus harmonic plus noise model in diphose based speech synthesis.ICASSP,1998
11 Yegnanarayana B,Alessandro C R d,Darsinos V.An iterative algorithm for decomposition of speech signal into periodic and aperiodic components.IEEE Trans.On SAP,1998;6(1)
12 程云鹏.矩阵论.西安:西北工业大学出版社,1999
13 拉宾纳 L R,谢弗 R W 著,朱龙雪译.语音信号与数字处理.北京:国防工业出版社,1990
14 罗小冬,裘雪红,刘凯.语音信号的基音标注算法.计算机与现代化,2003;(1)

(收稿日期:2004-12-25)

RT Embedded http://www.kontronn.com

通过 CPU 的本地总线、ISA 总线以及 SRAM 总线建立与操作系统的连接。DOC 提供片选信号(CE#)、读写信号(WE#)、使能信号(OE#)以及 13 位宽的地址线(A[0:12])和 8 位宽的数据总线(D[0:7])。

本系统使用了 m-sys 公司的 DiskOnChip2000 系列，型号为 MD2200-D08。其中封装了二片 TOSHIBA 公司的 TC5832DC NAND FLASH，可以直接插在 32 针 DIP E^2PROM 插座上。在本设计中地址映射到以 0x3600000 为起始地址的 8KB 空间上。具体映象分布如下：

(1)启动块。该部分包含针对特定 CPU 的启动代码，可用大小是 64B，在 2KB 中重复该数据 32 次。启动块的其余代码在第二部分。

(2)启动块。该部分存储启动块的另外 64B。

(3)控制寄存器。该部分用来控制 DiskOnChip 2000 和内部的 Flash 介质。

(4)Flash 区。该部分用作对 Flash 介质读写数据的一个缓冲窗口。

1.2 NAND Flash 和 NOR Flash 介绍

NAND 和 NOR Flash(闪存)是目前市场上二种主要的非易失闪存芯片。这二种 Flash 的区别在于：NOR 类型 Flash 可以按照字节访问，所以存放在 Flash 中的程序可以直接执行；而 NAND 是串行访问的，它类似于硬盘的存取方式，需要先把程序读取到内存，然后在内存中运行。与 NOR 型相比，NAND 型闪存的优点是容量大，但速度比较慢。因为它的 I/O 端口只有 8(或 16)个，比 NOR 型的少，要完成地址和数据的传输就必需让这些信号轮流传送。这种串行传输的速率就比采用并行传输的 NOR 型芯片的速率慢很多。NAND 型的存储和传输是以页和块为单位的，比较适合大块数据的连续传输，同时也可以部分弥补串行传输的缺陷。因此，NAND 型闪存最适合的工作就是保存大容量的数据，如作为电子硬盘、移动存储介质等使用。

本系统中所使用的 NAND Flash 芯片为 TOSHIBA 公司的 TC58V32DC。该芯片的存储容量为 4MB+96KB，其中主数据区为 4MB，辅助数据区为 96KB，工作电压为 5V，I/O 端口宽度为 8 位。芯片内的(4MB+96KB)内存是按块和页的概念来组织的。一个芯片包括 1024 个块，每块包含 16 个页，每页内有 528B。芯片内具有一个容量为 528B 的数据寄存器，称为页寄存器，用来在数据存取时作为缓冲区。当对芯片内的某一页进行读写时，其数据首选被转移到此数据寄存器内，通过数据缓冲区与芯片外进行数据交换，以完成读写功能。页内的 528B 被划分为512B 的主数据区和 16B 的辅助数据区。主数据区存放用户数据，辅助数据区用来存储 ECC 代码(Error CorrectionCode，错误校验码)、坏块信息和文件系统相关代码。

NAND Flash 采用高度复用的访问接口。该接口既用作地址总线，又用作数据总线和命令输入接口。NAND Flash 的接口引脚主要分为数据引脚、控制引脚和状态引脚三类。本系统中所用芯片的数据引脚为 8 位，即 I/O1~I/O8，用来输入输出地址、数据和命令。有一个状态输出脚(R/_B)，用来表示设备的状态，当进行数据写入、擦除和随机读取时其输出为高电平，表明设备正处于忙状态，否则输出低电平。WP# 引脚用来禁止或允许写入操作，当其为低电平时禁止写操作，反之则允许写操作。控制引脚有 5 个，其中 ALE、CLE 说明如表 1。

表 1 ALE.CLE 引脚证明

ALE	CLE	寄存器选择
0	0	数据寄存器
0	1	命令寄存器
1	0	地址寄存器
1	1	未定义

TC58V32DC 地址是通过复用 8 个 I/O 引脚送入芯片的。这样的设计显著减少了芯片的管脚数目，并为系统的升级带来了方便。在 CE# 为低时，将 WE# 置低可以把 TC58V32DC 的命令、地址和数据通过 I/O 口写入。数据在 WE# 的上升沿写入芯片。命令锁存使能 CLE 和地址锁存使能 ALE 以区分 I/O 口的数据是命令还是地址。所有的命令用一个总线周期，块擦除命令要用二个总线周期：一个周期用来做擦除建立，另一个周期在地址调入后做擦除执行。TC58V32DC 有 4MB，需要 22 位的地址，所以字节级的地址要用三个周期依次送入：列地址(A0~A7)、页地址(A9~A21；其中 A9~A12 为页在块中的地址，A13~A21 为块地址；A8 确定是高 256 字节还是低 256 字节)。页的读操作和页的编程操作都需要同样的三个地址周期紧跟在相应的命令输入之后。然而，在块擦除的操作中，只要二个地址周期。不同的操作通过往命令寄存器写不同的命令来区分。

2 DiskOnChip2000 在 S3C4510B 上的硬件设计

S3C4510B 提供了四个外部 I/O 组，每个组可寻址空间为 16KB。本设计将 DiskOnChip 2000 扩展为外部 I/O 组 0，分别使用 nECS[0]、nOE、nWBE[0]作为 DiskOnChip 2000 的片选、使能和写信号。其原理图见图 2。

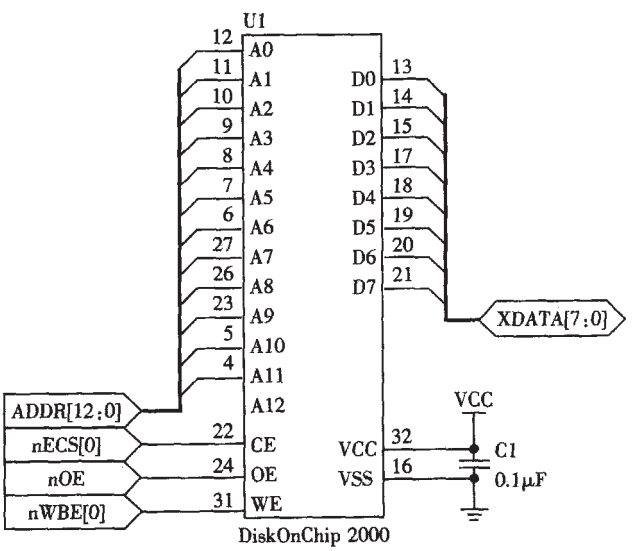


图 2 DiskOnChip 2000 在 SC34510B 上的硬件原理图

RT Embedded http://www.kontronn.com

3 uCLinux 系统下基于 DiskOnChip 的块设备驱动编程

3.1 DiskOnChip2000 读写函数的编写

因为 DiskOnChip 内可能有多个闪存器，所以其内的闪存器是按分层的方法来管理的，每层可能有多个闪存器，层和每层的编号都是从 0 开始的。因此可以通过 DiskOnChip 提供的选层寄存器（地址为 0x1003）确定要操作的层；再通过 DiskOnChip 提供的选片寄存器（地址为 0x1005）确定要操作的闪存器；然后通过向 DiskOnChip 2000 提供的 Flash 控制端口（地址为 0x1004）直接发送命令来控制 NAND Flash 的 WE、RE、WP、ALE、CLE 的电平。同时通过 DiskOnChip 的 I/O 端口（DiskOnChip2000 的 I/O 端口地址为 0x1800）向 NAND Flash 的 I/O1~I/O8 发送和读取数据。

例如，向 DiskOnChip 的 NAND Flash 发送命令和送入地址的函数实现为：

```
static inline int DoC_Command(unsigned char command)
{
    /* 向 DiskOnChip 的 Flash 控制寄存器发送命令，
                    置 NAND Flash 的 CE 和 CLE 有效 */
    /* 向 DiskOnChip 数据寄存器发送命令 */
    /* 向 DiskOnChip 的 Flash 控制寄存器发送命令，
                置 NAND Flash 的 CE 有效，CLE 无效 */
}
static inline int DoC_Address(unsigned long ofs)
{
    /* 向 DiskOnChip 的 Flash 控制寄存器发送命令，
                    置 NAND Flash 的 CE 和 ALE 有效 */
    /* 分三个周期向 DiskOnChip 数据寄存器发送地址
                        （擦出时只用二个周期） */
    /* 向 DiskOnChip 的 Flash 控制寄存器发送命令，
                置 NAND Flash 的 CE 有效，ALE 无效 */
}
```

同理，可根据 NAND Flash 的读写逻辑编写出相应的读写和擦除函数。

```
int doc_read(struct DiskOnChip * doc, loff_t from, size_t
    len, size_t * retlen, u_char * buf); //DiskOnChip 读函数
int doc_write(struct DiskOnChip * doc, loff_t to, size_t len,
  size_t * retlen, const u_char * buf); //DiskOnChip 写函数
int doc_erase (struct DiskOnChip * doc, loff_t ofs, size_t
                        len); //DiskOnChip 擦除函数
int doc_read_oob(struct DiskOnChip * doc, loff_t ofs,
                size_t len, size_t * retlen, u_char * buf);
                        //DiskOnChip 读辅助数据函数
int doc_write_oob(struct DiskOnChip * doc, loff_t ofs,
            size_t len, size_t * retlen, const u_char * buf);
                        //DiskOnChip 写辅助数据函数
```

3.2 uClinux 下 DiskOnChip2000 驱动的实现

Linux 中输入/输出设备被分为三类：块设备、字符设备和网络设备。由于 DiskOnChip 也是以页方式组织数据的高速设备，所以可将它作为一个块设备。下面就 DiskOnChip 的驱动程序设计作一介绍。

(1)向系统注册和注销驱动程序

使用 devfs_register_blkdev 和 devfs_unregister_blkdev 向设备管理子系统注册和注销块设备驱动程序。其中注册驱动程序必须在初始化时进行，具体用法如下：

```
int devfs_register_blkdev(unsigned int major, const char
        * name, strcut block_device_operations * bdops);
```

参数 bdops 指针指向一个初始化的结构 block_device_operations，该结构定义如下：

```
struct block_device_operations{
int ( * open)(struct inode * , struct file * ); //设备打开函数
int ( * release)(strcut inode * , struct file * ); //设备关闭函数
int ( * ioctl)(struct inode * , struct file * , unsigned,
                        unsigned long); //设备控制函数
int ( * check_media_change)(kdev_t);
struct module * owner;
};
```

DiskOnChip 设备驱动程序实现了其中的三个函数：open、release、ioctl。open、release 函数的实现只是分别简单地增加和减少了使用设备的用户数。而由于 doc 设计的特殊性，即每页是由 512 字节的数据区和 16 字节的辅助数据组成，所以获取（或写入）16 字节的辅助数据就是通过 ioctl 实现的，同时其中也实现了 doc 的擦除和 doc 容量的获取。

驱动程序的注销一般在模块的注销函数中实现，具体用法如下：

```
int devfs_unregister_blkdev (unsigned int major, const
                        char * name);
```

其功能是向 Linux 设备管理子系统注销以前注册的设备驱动程序，对应的设备不能再被访问。参数 major 和 name 与注册函数时使用的前二个参数一致。

(2)驱动程序对数据的处理

DiskOnChip 驱动程序的 block_device_operations 定义如下：

```
static struct block_device_operations doc_fops =
{
    owner:          THIS_MODULE,
    open:           doc_open,
    release:        doc_release,
    ioctl:          doc_ioctl
};
```

显然，该结构未提供块设备的读写操作函数。实际上，当系统对块设备进行读写操作时，仅仅是通过块设备通用的读写操作函数 genric_file_read()/genric_file_write()，

RT Embedded http://www.kontronn.com

将这一请求发送给对应的设备，并保存在该设备的请求队列中，然后调用设备对应的 request 函数处理请求。所以在块设备注册成功后，一定会初始化该设备的请求队列，以通知内核本设备的请求函数地址，保证适当的时候可以调用。

通常块设备都有一个与主设备号对应的请求队列，该队列的初始化和注销分别调用 blk_init_queue 和 blk_cleanup 函数。DiskOnChip 的等待队列的初始化函数为：

```
blk_init_queue(BLK_DEFAULT_QUEUE(MAJOR_NR),
                                    &doc_request),
```

其中：参数 BLK_DEFAULT_QUEUE(MAJOR_NR)为定义的块设备的缺省请求队列，doc_request 即为 DiskOnChip 的请求函数。

DiskOnChip 的数据分为主数据和辅助数据。通常，人们只关心对主数据区的读写，而不关注辅助数据，所以对主数据和辅助数据的读写就不能使用统一的函数接口。Linux 系统提供了 ioctl 系统调用，其对应的内核实现在本设备驱动程序中就是 doc_ioctl 函数。其具体实现如下。

宏和结构的定义为：

```
#define DOC_READ_OOB        _IOR("d",0x01,oob_t)
#define DOC_WRITE_OOB       _IOW("d",0x01,oob_t)
#define DOC_ERASE           _IOW("d",0x02,oob_t)
#define DOC_SIZE            _IOR("d",0x02,oob_t)
typedef struct{
    int ofs;//页地址
    union{
        char oob_data[16];                    //辅助数据
        unsigned int len;      //擦除区大小，16 页(0x2000)
                                              //的整数倍
        unsigned int size;                    //doc 的容量
    };
}oob_t;
```

doc_ioctl 函数的详细定义略。

4 Yaffs 文件系统的移植

Yaffs 文件系统非常简单，适用于 NAND Flash，它节约资源、起动迅速。它如同 JFFSx 一样有日志功能，使得它比 FAT 更健壮。

由于标准的 YAFFS 只支持 MTD 设备，而今需要支持定制的 DiskOnChip 2000 驱动程序，所以需修改相应的读写和擦除扇区函数、相应的主设备号及设备类型指针（在 yaffs_fs.c 中）。同时，由于本 DiskOnChip 驱动实现没有加入分区功能，所以假定除 DiskOnChip 的第一块以外的其余块均交给 YAFFS 文件系统管理。

需重新实现的函数如下：

```
int nandmtd_WriteChunkToNAND(yaffs_Device * dev,int
                chunkInNAND,const __u8 * data,
                yaffs_Spare * spare);//写扇区
int nandmtd_ReadChunkFromNAND(yaffs_Device * dev,
                int chunkInNAND,__u8 * data,
                yaffs_Spare * spare);//读扇区
int nandmtd_EraseBlockInNAND(yaffs_Device * dev,
                int blockNumber);//擦除块
int nandmtd_InitialiseNAND(yaffs_Device * dev);
                //初始化设备
```

各具体函数的实现可调用 3.1 中提到的 DiskOnChip 的读写擦除函数。

5 结束语

本驱动程序的代码编译后在压缩内核中约占 27.3KB 的空间。由于嵌入式系统的特殊性，裁掉了原内核支持 DiskOnChip2000 的所有模块（包括 mtd、nftl 及额外的文件系统，如 ext2、fat 等），在压缩内核中可节约 30KB 以上的空间（考虑原系统用 ext2），因此在嵌入式 Linux 系统下有一定的实用价值。

参考文献

1 鲁比尼，科比特，魏永明.Linux 设备驱动程序.北京：中国电力出版社，2002

2 杜春雷.ARM 体系结构与编程.北京：清华大学出版社，2003

3 贾智平.微机原理与接口技术.北京：中国水利水电出版社，1999

4 李驹光，聂雪媛.ARM 应用系统开发详解.北京：清华大学出版社，2003

（收稿日期：2004-12-28）

来稿选摘

基于 OWL 共享本体的动态 Web 服务组合

Web 服务组合就是把现有的基本 Web 服务联合起来，用以提供具有附加价值的新功能。

在自动的动态 Web 服务组合中，如何提高匹配成功率是个很重要的课题。本文提出：通过结合 OWL-S 语义 Web 服务本体语言和 Web 本体语言 OWL，将语义引入 Web 服务组合，可以提高服务发现、匹配的效率并可以方便地实现 Web 服务组合的自动化。本文所介绍的系统，即是以 OWL-S 与 OWL 语言为基础开发的一个自动的动态服务组合平台。系统由 Web 服务组合器与组合 Web 服务执行引擎构件组成。文章介绍了系统的工作原理及系统框架，较详细地描述了基于 OWL 共享本体的服务匹配原理及具体实现。

须文波　周中成

（江南大学 信息工程学院，江苏 无锡 214036）

RT Embedded http://www.kontronn.com