文章编号:1671-4598(2025)03-0235-08 DOI:10.16526/j.cnki.11-4762/tp.2025.03.030 中图分类号:TP311.52 文献标识码:A

基于 FMQL45T900 文件系统研究与设计

宋书龙
(中国西南电子技术研究所，成都 610036)

摘要：针对装备的核心器件需用进口，导致设备存在安全及市场卡脖子问题，为降低对进口芯片的依赖，实现用国产芯片进行原位替代，并按照数据存储模块的相关要求，研究并提出了一种基于 FMQL45T900 的文件系统存储数据架构，给出了文件系统的总体设计和具体实现方法，应用层以文件和目录进行管理，驱动程序基于块设备驱动框架开发，并根据设备树信息动态配置和管理硬件设备，使得驱动程序更加模块化和可复用；通过测试表明，文件系统能够稳定可靠地运行，满足项目对数据存储的高效性、可靠性和安全性需求，对促进国产芯片的推广应用具有一定意义。

关键词：FMQL45T900；VxWorks；设备驱动；文件系统；VxBus 总线

Research and Design of File System Based on FMQL45T900

SONG Shulong
(Southwest China Institute of Electronic Technology, Chengdu 610036, China)

Abstract: Due to the reliance on imported core components, there are safety hazards and boottleneck issues with the equipment. In order to reduce dependence on imported chips and achieve in-situ substitution with domestically produced chips. According to the requirements of the data storage module, a file system storage data architecture based on FMQL45T900 is researched and proposed, which presents the overall design and specific implementation method of file system. The application layer is managed by files and directories, and the driver program is developed based on the block device driver framework, dynamically configuring and managing hardware devices according to device tree information, making the driver more modular and reusable. Tests show that the file system can run stably and reliably, meeting the project's requirements for efficiency, reliability, and security of data storage. It is of significance to promote the application of domestically produced chips.

Keywords: FMQL45T900; VxWorks; device drivers; file system; VxBus bus

0 引言

随着国际形势的变化以及外部对高端芯片的出口限制，国家和企业面临着芯片短缺重大挑战。在这种背景下，开发和推广国产芯片成为了当务之急。FMQL 系列芯片作为国产替代品，应运而生。它集成了 ARM 处理器和可编程逻辑，提供了与进口芯片相媲美的功能、性能和可靠性。

FMQL 系列芯片是国产的一款可编程片上系统（SoC，system on chip）芯片。该芯片结合了高性能处理系统（PS，processing system）和灵活的可编程逻辑（PL，programmable logic）模块，提供了丰富的功能和扩展性。在 PS 端，FMQL 芯片集成了四核 Cortex-A7ARM 处理器，这使得它能够处理复杂的计算任务和运行操作系统。还包括多种通用控制器，例如通用型输入输出、通用异步收发传输器等，这些控制器为微控制器与外部设备的通信提供了基础。在 PL 端则配备了可编程门阵列 FPGA，允许用户通过 FPGA 的逻辑资源实例化 IP 核。此外，FMQL 系列芯片还支持 VxWorks 操作系统，并提供了板级支持包（BSP，board support package），使其能够在多处理器（SMP，symmetrical multi-processing）模式下运行，这一特性使得 FMQL 芯片在复杂的嵌入式系统中表现出色，提供了灵活性和强大的计算能力，适用于各种高性能应用场景[1-2]。

VxWorks 在嵌入式系统领域占据着重要的地位，经过多年的发展和完善，具有很高的稳定性和可靠性，VxWorks 采用模块化的设计，可以方便地添加新的硬件驱动和应用程序。在 VxWorks 6.2 之前，VxWorks 的设备驱动程序开发完全依赖于 BSP，为了简化驱动开

收稿日期:2024-10-26； **修回日期**:2024-12-12。
作者简介:宋书龙(1989-),男,硕士研究生,工程师。
引用格式:宋书龙.基于 FMQL45T900 文件系统研究与设计[J].计算机测量与控制,2025,33(3):235-242.

发在 Workbench3.0 中首次引入基于总线的设备驱动模型，以组件的形式体现的，驱动程序被封装成独立的组件，便于管理和复用，方便开发人员根据需要进行驱动的添加、配置和删除，可以显著提高开发效率，缩短产品上市时间，复用现有的驱动组件可以降低开发成本，更容易维护和升级。

eMMC（embedded Multi-Media Card）作为一种嵌入式存储解决方案，通过内置的 NAND Flash 控制器简化了存储管理，并提供了统一的标准接口。随着协议版本的更新，已从 eMMC4.3 更新至当今的 eMMC5.1，传输速度、设备功能和功耗都得到了显著提升。这些改进使得 eMMC 在各种设备中得到了广泛应用，同时也提升了数据处理能力和效率[3]。

国外，嵌入式文件系统的研究起步较早，最早提出很多先进的嵌入式文件系统框架和优化技术。主要的研究方向包括：1）嵌入式文件系统的标准化与应用例如 FAT（File Allocation Table）文件系统、YAFFS（Yet Another Flash File System）和 JFFS2（Journaling Flash File System 2）文件系统。FAT 文件系统的设计简单、效率高，但在大规模文件管理、数据恢复、日志管理等方面存在局限。YAFFS 能够有效减少闪存的磨损，并具有较好的容错能力，特别适用于数据频繁写入和更新嵌入式系统。JFFS2 则通过日志机制，提高了数据一致性和系统恢复能力；2）嵌入式文件系统的性能优化，许多文件系统如 YAFFS 在写入操作和垃圾回收机制进行优化，以减少对闪存的写入负担，并延长闪存的使用寿命。FAT 文件系统则专注于提高存储介质的读写速度，确保文件操作能够在预定的时间内完成。

在国内，嵌入式文件系统的研究相较于国外起步较晚，但近年来随着嵌入式技术的快速发展，相关研究逐渐增多，并且取得了显著进展。主要的研究方向包括：1）嵌入式文件系统的性能优化，国内学者在嵌入式文件系统的优化方面开展了大量研究，尤其是在文件系统的性能优化、功耗管理等领域。在 FAT 和 YAFFS 的基础上提出了改进方案，保持原有文件系统优势的同时，提升其在嵌入式设备中的应用性能；2）基于闪存的文件系统的优化，一些研究关注如何通过减少系统调用的次数、优化磁盘访问顺序和提高缓存策略，来提高嵌入式文件系统的响应速度和存储效率；3）国内也有研究针对闪存的垃圾回收机制进行优化，减少因垃圾回收带来的性能开销。

近年来，随着国内嵌入式技术的快速发展和国家政策的支持，一些国内企业和科研机构也开始着手自主研发适用于国产嵌入式平台的文件系统。本文在于研究基于国产芯片的文件系统，并针对国产硬件平台的适配和优化，以支持自主可控的嵌入式设备。

国内外在嵌入式文件系统的研究方面均取得了重要进展。国外的研究多集中在文件系统优化和性能的提升等方面；而国内的研究则更多关注嵌入式文件系统的自主创新和国产平台的适配。随着嵌入式设备的广泛应用和存储需求的增长，嵌入式文件系统的研究逐渐成为嵌入式领域的重要课题。

1 系统研究与设计

1.1 VxWorks 系统结构

VxWorks 作为一款实时操作系统，为了保证系统对硬件的精细控制和高效利用，按照与硬件的依赖程度划分 3 个层级，分别是硬件、BSP 驱动和应用程序层，各层之间通过标准的接口进行交互，当需要修改或添加功能时，只需要修改相应的模块，对于新的硬件设备，只需要添加对应的驱动程序，使得系统具有更好的可维护性和可扩展性[4-6]。

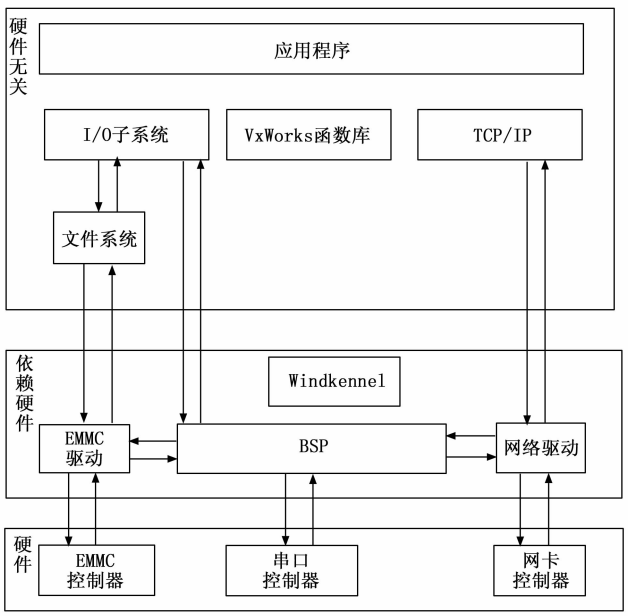

图 1 Vxworks 系统结构

系统各层之间的关系和作用：

硬件是整个系统的基础，包括 EMMC 控制器、串口控制器和网卡控制器等，BSP 驱动层完全依赖于硬件。

BSP 驱动层是针对不同的硬件平台进行高度定制化开发，将硬件的复杂性隐藏起来，为上层的驱动和应用程序提供一个统一的接口，在系统启动时，负责初始化硬件，实现对硬件设备的控制，如读写寄存器，驱动程序通过调用 BSP 提供的函数来访问硬件，同时驱动程序也提供一组标准的接口函数，供应用程序调用，是链接硬件和应用程序的桥梁。

应用程序是 VxWorks 系统最上层的用户程序，它

通过调用驱动程序提供的接口来控制硬件，应用程序只需调用一系列的 I/O 子系统和 EMMC 驱动程序，即可实现对磁盘的读写，转换成对文件的读写，使得应用程序无需关心具体的磁盘驱动细节。

VxWorks 通过分层的方式，将硬件、BSP 驱动和应用程序清晰地划分开来，使得系统结构更加清晰，便于开发和维护，这种分层结构也使得 VxWorks 具有很强的可扩展性和可移植性。

1.2　硬件原理图

eMMC 芯片是一种集成 NAND Flash 带控制器的管理功能的存储芯片，eMMC5.1 标准定义了其工作电压、外部接口规范以及内部时序控制等关键参数。根据图 2 硬件原理图可以很清楚地看到，EMMC _ D0～EMMC _ D3 表示数据线，EMMC _ RST _ N 表示硬件重置线，EMMC _ CLK 表示时钟线，EMMC _ CMD 表示命令线，EMMC _ DS 表示数据选通线，该信号线要在 HS400 模式下配置启用，本文设计的数据速率模式 SDR（Single Data Rate），兼容 MMC，总线位宽 4bit，时钟 25MHZ，VCCQ 给控制器供电，VCC 给 NAND Flash 存储单元供电。

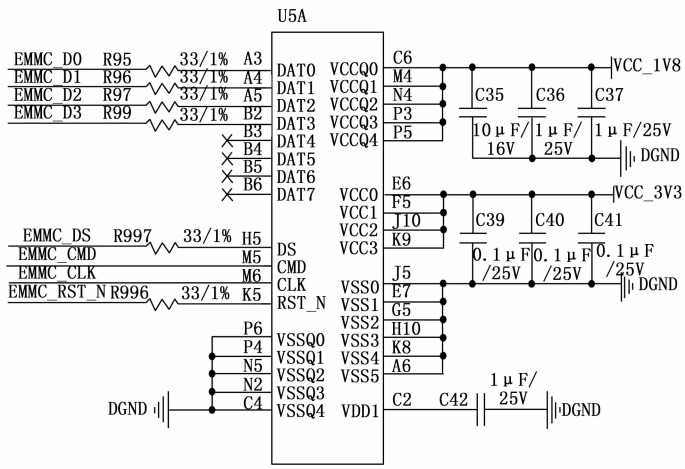

图 2　存储芯片原理图

1.3　外设接口配置

外设接口在系统功能实现中扮演着关键角色，通过使用特定寄存器或编程来配置外设 I/O 引脚，能够有效地控制各种硬件设备，实现处理器与外设的数据交互。EMMC 数据读写接口有两种，一种 SPI 接口和一种 SDIO（Secure Digital I/O）接口，SDIO 接口可以挂载各种外接设备实现不同功能，是利用标准的 SD 总线来进行数据传输；SPI 接口是利用 SPI 总线来进行数据传输，本文选择 SDIO 模式如图 3 所示。

FMQL45T900 端的 MIO40 引脚连接外设 EMMC _ CLK 引脚，MIO41 引脚连接外设 EMMC _ CMD 引脚，MIO42 引脚连接外设 EMMC _ D0 引脚，MIO43 引脚连接外设 EMMC _ D1 引脚，MIO44 引脚连接外设 EMMC _ D2 引脚，MIO45 引脚连接外设 EMMC _ D3 引脚。

1.4　文件系统总体结构

操作系统上电调用 usrDosfsInit、dosFsCacheLibInit 和 dosFsShowInit 等函数初始化 DosFS（Dos File System）层，调用 xbdInit、fsMonitorInit 和 fsEventUtilLibInit 等函数初始化 XBD（eXtened Block Device）中间层。

应用程序对磁盘的读写操作实质是调用标准的 I/O 库对文件的读写，VxWorks 所有的 I/O 设备都被当做文件来存取，应用程序可以直接对内核函数进行调用，没有权限上的限制，标准的 I/O 库是由 ioLib 提供，包括 open、read、write、ioctl、close 等，其中调用 open 函数创建一个文件描述符，其他的函数以文件描述符作为参数来指定对应的文件，文件描述符是全局句柄，调用 close 函数文件被关闭，文件描述符被释放。VxWorks 内部维护一张文件描述符表，文件描述符就是表的索引，对文件描述表的操作就是对文件的操作[7]。

IO 子系统层位于应用程序下面，应用程序调用 ioLib 的函数会发起一个 I/O 请求，I/O 请求会进一步向操作系统内核进行传递，ioLib 层之下的就是 iosLib 层，一般将 ioLib 层称为应用层，iosLib 称为 IO 子系统层，该层接口对应用不可见，向下管理着所有类型的设备驱动，字符设备驱动和块设备驱动都必须向 IO 子系统进行注册方可被内核访问。应用程序并不需要知道它访问的是一个什么硬件设备，操作系统会根据不同的设备类型调用相应的驱动程序，IO 子系统层为各种设备提供了简单、统一，与设备无关的访问接口。

字符设备驱动可以直接挂接到 I/O 子系统，而块设备驱动和 I/O 子系统之间存在文件系统层，如图 4 所示 I/O 子系统与底层块设备驱动之间存在 DosFS 层和 XBD 层，DosFS 层是块设备驱动和 I/O 子系统接口，DosFS 通过 XBD 中间层来管理底层块设备驱动，实现对底层块设备 IO 的操作，DosFS 层与 XBD 层通过 xbdStrategy 函数进行数据读写交换，XBD 层与底层块设备驱动通过 blkXbdStrategy 函数进行数据读写交换，两个函数在不同的层之间起到衔接作用。

2　设备驱动层

设备驱动主要负责实现对硬件设备的操作如读写寄存器，而硬件设备的描述信息通过设备树（DT，device tree）获取，设备驱动程序通过设备树获取设备的配置信息，从而实现对硬件的操作。设备树用于描述硬件设备的数据结构和语言，它以树形结构描述系统中的设备，包括设备的类型、属性、寄存器等信息，以. dts 为后缀，将硬件配置信息从内核代码中分离出来，内核

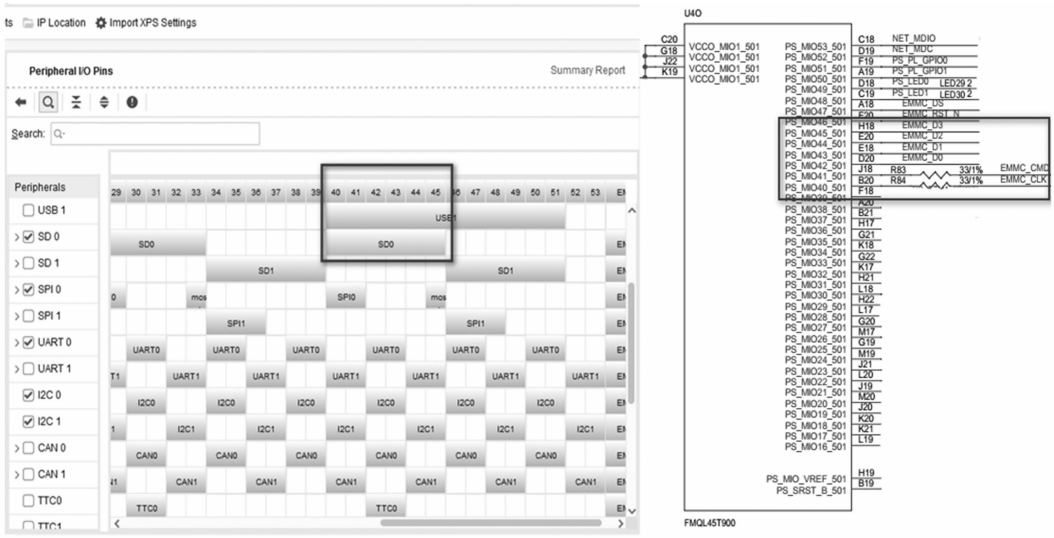

图 3 FMQL45T900 外设接口配置

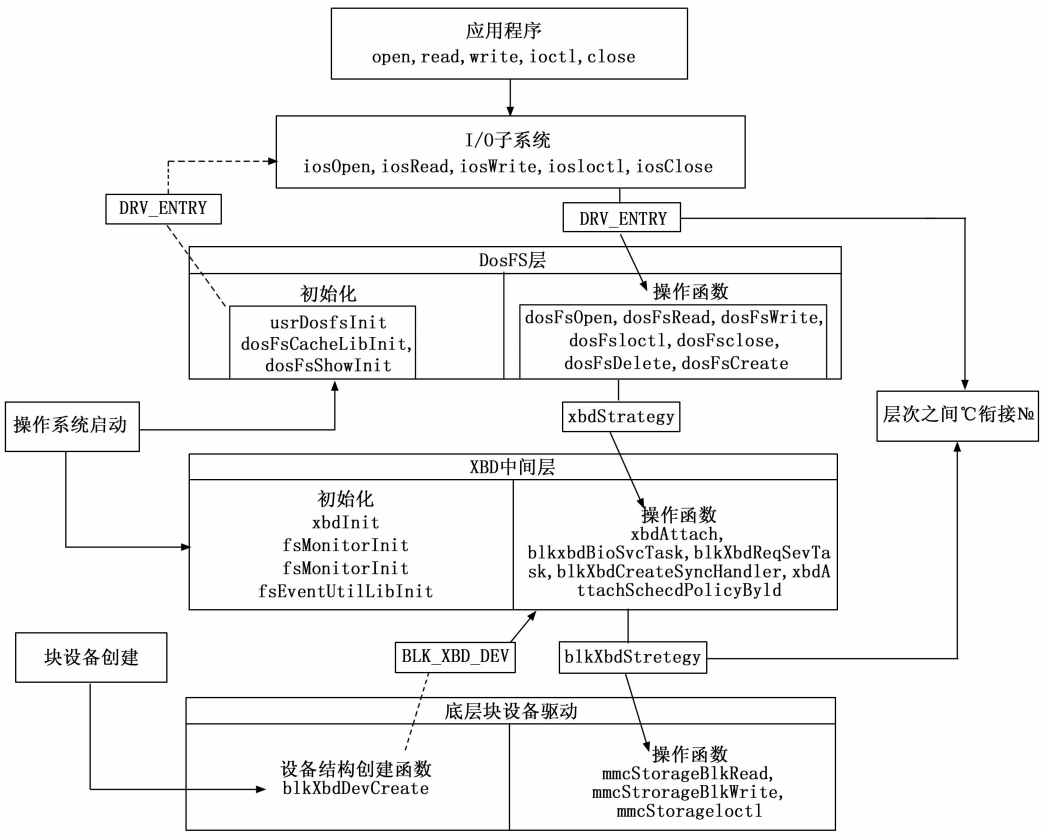

图 4 文件系统总体结构

根据设备树的信息来初始化硬件设备。VxBus 是硬件设备抽象为一系列的资源，操作系统中的一种设备驱动模型，VxBus 会根据设备树中的信息创建一个设备对象，并为其分配相应的资源，VxBus 找到匹配的驱动程序，并将其绑定到该设备上。驱动程序可以通过读取设备树中的属性来获取设备的配置信息[8-10]。

mmc0：dwmmc@e0043000 {

compatible = "fmsh,psoc-dw-mshc";/* 厂商复旦微和模块对应的驱动名字 */

reg = <0xe0043000 0x1000>;/* 设备定义的起始地址 0xe0043000，长度为 0x1000 */

clocks = <&clkc NCLK_AHB_SDIO0>, <&clkc NCLK_SDIO0>;/* 时钟配置为使用两个时钟源，分别为 clkc NCLK_

```
AHB_SDIO0 和 clkc NCLK_SDIO0 * /
    clock-names = "biu", "ciu";/ * "biu":表示第一个时钟名称,"ciu":表示第二个时钟名称 * /
      address-cells = <1>;
      size-cells = <0>;
    data-addr = <0x100>;/ * 表示数据的起始地址为 0x100 * /
    fifo-depth = <0x20>;/ * FIFO 深度 * /
    bus-width = <4>;
    status = "disabled";
    cap-mmc-highspeed;/ * 指定设备支持高速 MMC 标准 * /
    };
```

VxWorks 系统下的设备通常分为 3 类：字符设备，块设备和网络设备。

字符设备：以字节为单位进行数据传输，没有固定的块大小限制，数据传输是顺序的，例如串口、鼠标和键盘等；

块设备：以块为单位进行数据传输，块的大小是固定，数据可以随机访问，数据传输有固定块大小的限制，例如 EMMC、SD 和硬盘等；

网络设备：以数据包为单位进行发送和接收，需要处理网络协议，管理网络接口，例如网卡、无线网卡等。

VxWorks 对设备驱动程序的管理有着一套完善的机制，它通过维护多个表来跟踪系统中的设备和驱动程序，确保它们之间能够正确地关联并协同工作，VxWorks 内核对设备、驱动和文件管理主要通过三张表进行维护，分别是系统设备表、系统驱动表和文件描述符表，系统设备表是一个链表，包含系统中所有设备的结构体，通过遍历这个链表，可以获取系统中所有设备的信息。系统驱动表是一个数组，每个数组元素指向一个驱动函数表，每个设备都关联一个驱动函数表索引，从而确定该设备对应的驱动程序。文件描述符表用于管理打开的设备，每个打开的设备都有一个对应的文件描述符，该描述符指向设备在系统设备表中的条目。

2.1　块设备驱动

块设备驱动以块为单位，本文定义块大小为 512 字节，每次只能以块的整数倍进行数据写入和读取，对于应用层而言，块设备以文件系统的形式存在，用户以操作文件和目录的方式访问块设备，所以块设备可以随机读写访问，字符设备驱动只能从当前位置开始顺序读写访问，字符设备比较简单，直接受 IO 子系统管理，与 IO 子系统进行数据交互不经过任何中间层，块设备驱动与 IO 子系统之间数据交互需要通过 DosFS 层、XBD 层之间的转换。

如上文中图 4 所示的底层块设备驱动与 XBD 层之间需要建立数据交换，BLK _ XBD _ DEV 结构体中进行初始化操作保存了 XBD 层所需的所有信息，这个结构作为桥梁，完成上层与底层驱动之间的关键信息传递，在块设备驱动中 BLK _ XBD _ DEV 结构体非常重要，在初始化一个块设备时，需要对这个结构体的成员进行赋值，以准确地反映设备的特性和功能，XBD 保存块的大小，块的数量等信息，BLK _ XBD _ OPS 块设备的读、写操作[11]。对块设备初始化操作，调用 blkXbdDevCreate 函数完成块设备的创建。EMMC 块设备创建完成，将 EMMC 存储设备的读写和控制操作与 XBD 中间层关联起来，当系统对这个 XBD 设备进行读写或控制操作时，就会调用相应的块驱动函数来实现。

2.2　XBD 中间层

VxWorks 5.5 版本使用 CBIO（Cached Block I/O）辅助组件作为缓存，Vxworks6.4 以后使用 XBD 是一种用于管理块设备的缓存中间层，本文使用的 XBD 作为中间层。XBD 主要针对块设备，提供一个统一接口，多用于硬盘、EMMC 卡，对于传输大块数据，效率较高。CBIO 提供更底层的字符设备访问方式，以字节为单位对设备进行操作例如串口设备，主要应用于小量数据传输。本文传输数据都以块为单位，采用扩展块设备作为中间层[12-13]。

XBD 层对块设备进行封装，把填充块的信息通过 blkXbdParams 放到 BLK _ XBD _ DEV 对象，然后会调用 xbdAttach 函数将一个 XBD 设备附加到 DosFS 文件系统层，该函数以一个 xbd 结构体作为支点。在 XBD 中间层包含两个主要数据结构：

```
xbd 设备结构体和 bio 块结构体；
struct xbd {
struct device xbd_dev; / * 设备结构体包含了设备名称、类型等信息 * /
struct xbd_funcs * xbd_funcs; / * 指向一个 XBD 函数结构体的指针,该结构体定义了对该 XBD 设备可以进行读写操作 * /
unsigned xbd_blocksize;/ * xbd 设备块大小 * /
sector_t xbd_nblocks;/ * 总共的块数 * /
…
} XBD;
struct bio {
device_t bio_dev; / * 本次 I/O 操作所针对设备 * /
sector_t bio_blkno; / * 本次 I/O 操作起始块号 * /
unsigned bio_bcount; / * 本次 I/O 操作传输总的字节数 * /
void * bio_data; / * 存放读取或写入的数据 * /
unsigned bio_resid; / * 本次 I/O 操作剩余未传输的字节数 * /
};
```

bio 块结构体包含了关于一次 I/O 请求的所有必要信息，例如设备、数据缓冲区、传输字节数等。blkXbdFuncs 函数定义对 XBD 设备的读取数据、写入数据等操作。

LOCAL struct xbd_funcs blkXbdFuncs ={blkXbdIoctl,blkXbdStrategy,blkXbdDump};

xbdStrategy 提取并解析 DosFS 文件系统层传递过来的 I/O 请求，例如读数据请求或者写数据请求，将它们作为参数传递给 XBD 层的 blkXbdStrategy 函数，将子请求转换为具体的设备操作，交给设备驱动程序，完成数据的读写，最终实现指定 I/O 任务。

2.3 添加驱动组件

VxWorks 为了与各种硬件平台和设备配合使用，通过添加驱动组件，可以扩展其功能，提升系统的灵活性和适应能力。需要为文件系统添加驱动组件包括如下。

INCLUDE_DOSFS：启用对 DOSFS 文件系统的支持。

INCLUDE_DOSFS_MAIN：一个更具体的配置选项，用于控制 DOSFS 模块的编译和初始化。

INCLUDE_DOSFS_FMT：用于控制 DOSFS 文件系统的格式化功能。

INCLUDE_DOSFS_CHKDSK：用于控制 DOSFS 文件系统的磁盘检查。

INCLUDE_DOSFS_DIR_VFAT：使其支持 VFAT 格式支持长文件名。

INCLUDE_DOSFS_DIR_FIXED：使用固定大小的目录项来存储文件名和文件属性。

INCLUDE_DOSFS_SHOW：专门用于控制 DOSFS 文件系统的调试信息输出。

INCLUDE_FS_MONITOR：使其能够监控文件系统的变化，对文件系统的各种事件进行跟踪。

INCLUDE_XBD：使其支持 XBD。

INCLUDE_DEVICE_MANAGER：使其支持设备管理器。

DRV_MMCSTORAGE_CARD：用于驱动和管理 EMMC 卡的设备驱动程序。

在包含宏定义后，内核会对相关组件进行初始化，DosFS 组件初始化完成后，就可以向 I/O 子系统注册 DosFS 文件系统中间驱动层。

2.4 DosFs 层

VxWorks 为块设备提供了两种文件系统：一种与 MS-DOS 文件系统相兼容 DosFS 文件系统，另一种原始文件系统 RawFS。这些文件系统的支持库分别为 dosFsLib 和 RawFsLib。RawFs 文件系统将整个磁盘视为一个大型文件，将整个磁盘区域视为单一文件。可以通过任何为设备打开的文件描述符访问该区域，所有对磁盘的读写操作都是相对于磁盘起始块的字节偏移量，而不维护目录信息。DosFS 文件系统提供了层次化的文件和目录管理方式，使得文件组织更加有序和便捷。它兼容 VFAT（Volume Label for FAT）长文件名，允许使用更具描述性的文件名，从而提升文件管理的灵活性和可读性。支持 FAT16 和 FAT32 文件格式，使其能够处理不同大小和需求的存储设备。这种文件系统适合于通用文件存储，因其结构简单且易于实现，从而具备了较好的兼容性和易用性，广泛应用于多种操作系统和设备中。因为 DosFs 文件系统与 MS-DOS 相兼容特性，有目录结构和文件信息，对于 windows 系统操作人员来讲，维护简单，便于操作，所以本文采用 DosFs 文件系统[14-15]。

创建 DosFs 层设备：

在 VxWorks 中，提供了一个统一、灵活的 I/O 访问方式，无论是传统的块设备、还是 DosFs 文件系统，都会被抽象为一个设备。这种统一的模型使得应用程序可以通过相同的接口来访问不同类型的设备，大大简化了 I/O 编程，DosFs 文件系统层在 VxWorks 中也被看作一个设备。通过调用函数 DosFsDevCreate（）来在指定分区上创建 DosFs 文件系统，调用 fsmProbeInstall 函数是将文件系统的监测函数注册到文件系统管理（FSM，filesystem manager）中，使得系统能够识别 DosFs 文件系统，并监控 DosFs 文件系统的活动，它会对 DosFs 文件系统上的各种操作进行监控，例如文件的创建、删除、修改、读取等。当用户或系统执行文件系统操作时，文件系统内核会检查是否有已注册的回调函数，如果有注册的回调函数，文件系统内核会调用该函数，并将相关操作信息作为参数传递给它，回调函数接收到相关操作信息后，可以进行相应处理。

3 文件系统实现

3.1 读设备块驱动实现

底层驱动读函数提供了读取 EMMC 卡数据的基本功能，图 5 读取设备单个块过程，首先设置块的长度，通过命令从给定地址开始将数据块读到缓存[16-18]。

主机在命令指令线上发送一个命令 CMD17，需要读取的地址就会包含这个命令中，设备接收到该命令时，它会将响应放入 R1 寄存器，并同时将数据块放到数据线上去，发送给主机，每个数据块的末尾都会附加一个 CRC 校验码，用于检测数据在传输过程中的错误，以确保传输过程中数据的完整性和正确性。命令和数据在不同的线路进行并行传输，这种设计提高了数据读取的效率，使得设备能够在回应命令的同时进行数据传输，显著减少了延迟。在数据块的传输过程中，主机发送 CMD12 命令用于终止数据传输，确保能够准确收到信息，并处理接收到的数据。

3.2 写设备块驱动实现

底层驱动写函数提供了将指定的数据块写入到 EMMC 卡中，图 6 主机将单个块数据写入到设备的过程，首先设置块的长度，通过命令将数据块写入设备的块中。

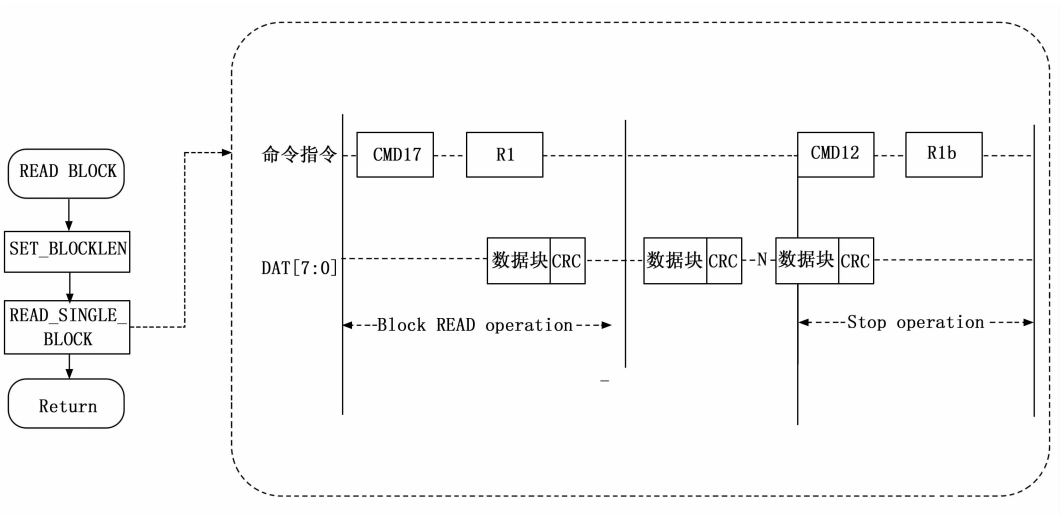

图 5　读函数驱动

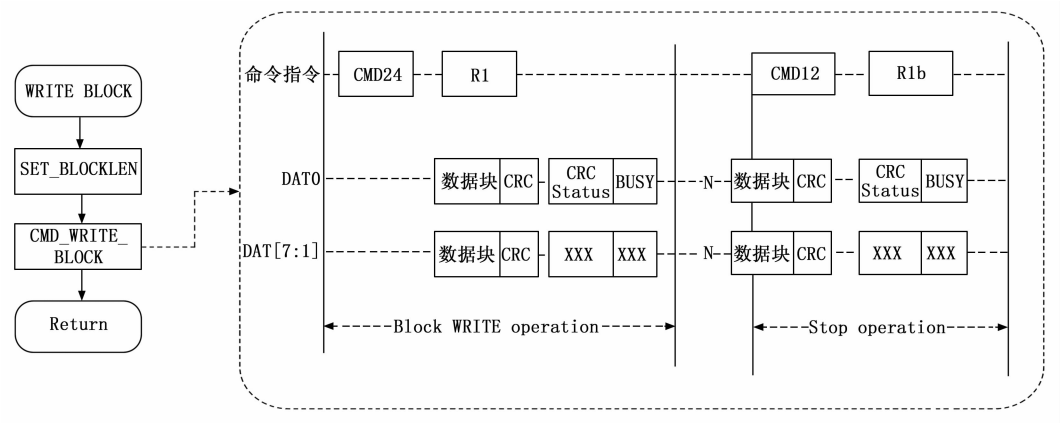

图 6　写函数驱动

一般情况下，CRC 校验码会与数据一起通过 DATA0 线传输，设备发送完 CRC 校验码后，会拉低 DATA0 数据线，表示设备处于忙状态，在这个时候设备会把接收到的数据块写入存储器中去，当写入操作完成后，设备会释放 DATA0 数据线，表示空闲，单个数据块写入完成，可以进行下一块的操作。

3.3　VxBus 总线注册

VxBus 总线提供了一种统一的方式来管理和访问系统中的各种硬件设备，它为设备驱动程序提供了强大的支持。设备驱动程序在初始化时向 VxBus 总线注册，VxBus 会扫描系统中的硬件设备，并将它们与已注册的驱动程序进行匹配，如果找到了匹配的驱动程序，VxBus 会将驱动程序绑定到设备上。

在 hwconf. c 文件中添加驱动所需的资源，包括寄存器基地址和中断号等，这些信息在驱动程序初始化时会被读取，从而让驱动程序知道如何与硬件设备进行交互。向 VxBus 总线注册驱动程序，操作系统访问 EMMC 的时候能够识别，从而实现读写数据等操作[19-21]。

系统初始化完成后，使用 vxBusShow 命令查看驱动信息，输入 devs 命令可见 EMMC 挂载盘符“/mmc0:0”，说明设备挂载成功。

4　测试与验证

设备与驱动关联匹配成功，用户可以正常操作该硬件设备，通过调用 mmcStorageBlkRead 函数对硬件设备进行读驱动进行测试，使用 FTP 工具把本地 test. txt 文件上传到文件系统，测试程序调用标准 IO 接口 read 函数，读取 test. txt 文件内容，经过层层调用后，最终调用底层 mmcStorageBlkRead 驱动函数读取设备数据，驱动函数和测试程序把读到结果通过串口输出到超级终端，通过打印比较测试程序读到数据与驱动读取数据一致，测试程序读取内容与本地 test. txt 内容比较完全一致，如图 7 所示。

通过调用 mmcStorageBlkWrite 函数对硬件设备进行写驱动测试，测试程序新建 test. txt 文件，调用标准 IO 接口 write 函数往文件中写入递增数，经过层层调用

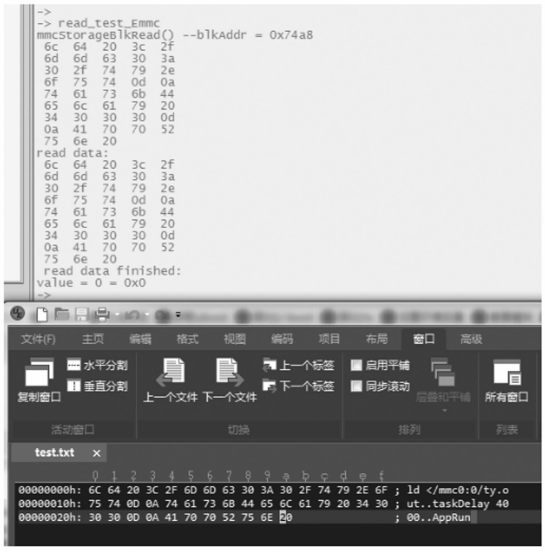

图 7 读驱动测试

后，最终调用底层 mmcStorageBlkWrite 驱动函数把数据写入设备，通过打印比较应用程序写入数据与驱动写入数据一致，使用 FTP 工具把文件系统 test.txt 文件下载到本地，比较测试程序写入内容与下载到本地内容完全一致，如图 8 所示。

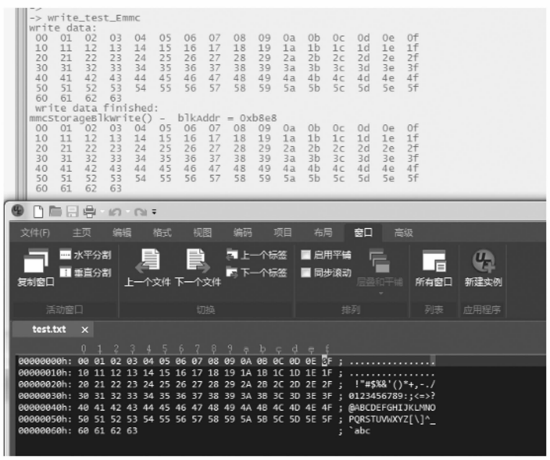

图 8 写驱动测试

5 结束语

本文针对国产芯片提出了一种基于 FMQL45T900 的文件系统设计方案，概述了文件系统实现的具体方法，并实现了底层块设备驱动的读写函数。根据文件系统的工作原理，应用层通过系统调用向文件系统发出读写请求，文件系统则调用相应的驱动完成操作，并将结果返回给用户。测试结果表明，应用层与驱动层的数据完全一致，说明文件系统能够满足需求。未来，该方案还有望进一步完善和扩展，以在国产芯片领域发挥更重要的作用。

参考文献：

[1] 张钰尧，张 前，郭俊麟. 基于 ZYNQ-7000 嵌入式平台的 Flash 驱动的设计与实现 [J]. 信息通信，2019 (4)：49 - 50.

[2] 吴凤柱，王 斌，王永全等. ZYNQ7000 全可编程 SoC 中的 EMMC 应用技术研究 [J]. 技术纵横，2020 (2)：37 - 39.

[3] 赵昶宇. VxWorks 系统下提高 SATA 硬盘传输速度的方法 [J]. 科技与创新计，2023 (18)：83 - 85.

[4] 付月生，王 丽. 基于 VxBus 的驱动程序架构分析 [J]. 计算技术与自动化，2012，31 (2)：98 - 102.

[5] 吴 云. 基于 VxWorks 下 dosFS 的块设备开发研究 [J]. 信息通信，2017 (11)：136 - 137.

[6] 石 炜，孟金芳. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法 [J]. 电子设计工程，2015，24 (23)：139 - 145.

[7] 王 博，王夕臣，章诗晨. 一种嵌入式平台下的高可靠文件系统设计 [J]. 航空电子技术，2021，52 (1)：16 - 20.

[8] Wind River. Vxworks bsp developers guide 6.9 [DB/OL]. http：//www.windriver.com，2015.

[9] Wind River. Vxbus device driver developers guide 6.9 [DB/OL]. http：//www.windriver.com，2015.

[10] Wind River. Vxworks kernel programmers guide 6.9 [DB/OL]. http：//www.windriver.com，2015.

[11] Wind River. usb for Vxworks6 Programmers guide [DB/OL]. http：//www.windriver.com，2015.

[12] WindRiver. Vxworks application programmers guide 6.9 [DB/OL]. http：//www.windriver.com，2015.

[13] WindRiver. Vxworks architecture supplement 6.9 [DB/OL]. http：//www.windriver.com，2015

[14] 曹桂平. VxWorks 设备驱动开发详解 [M]. 北京：电子工业出版社，2021.

[15] 陈智育，温延军，陈 琪. VxWorks 程序开发实践 [M]. 北京：人民邮电出版社，2004.

[16] JEDEC. Embedded Multimedia Card (eMMC)，Electrical Standard5.1 [S/OL]. http：//www.jedec.org,2014.

[17] 陈 婷. EMMC 测试系统的软件设计 [D]. 哈尔滨：黑龙江大学，2018.

[18] 张 诚. 面向 EMMC 协议的 SoC 系统设计 [D]. 哈尔滨：黑龙江大学，2018.

[19] 丁红晖，马游春，苏庆庆. 基于高速 EMMC 存储的弹载记录仪设计 [J]. 现代电子技术，2018，41 (6)：70 - 73.

[20] 盘勇军，黄伯铮. VxWorks6.8 操作系统下 NVMe 驱动设计 [J]. 航空电子技术，2017，48 (4)，32 - 37.

[21] 贺小琳，张善从. 基于 VxWorks 的 SD 卡驱动程序的设计与实现 [J]. 计算机工程与设计，2010，31 (16)：3573 - 3575.