基于 AUTOSAR 规范的 Flash 驱动程序的研究与实现①

项　晨　张谚华　涂时亮 (复旦大学 计算机科学技术学院 上海 200433)

摘　要： AUTOSAR（汽车开放式系统架构）平台可分为 3 层：应用层、运行时环境和基础软件，其中基础软件又包括系统服务、ECU 抽象层和 uC 抽象层。所有驱动程序都包含在 uC 抽象层和 ECU 抽象层中。存储器驱动提供了对不同存储设备的访问接口。根据 AUTOSAR 规范，在 MPC5633M 微控制器上，设计并实现了 Flash 驱动程序。提出的设计与实现的方法同样适用于其他设备驱动的开发。

关键词： AUTOSAR（汽车开放式系统架构）; ECU（电子控制单元）; 微控制器抽象层; Flash 驱动; MPC563xM

Research and Implementation of AUTOSAR Flash Driver

XIANG Chen, ZHANG Yan-Hua, TU Shi-Liang

(School of Computer Science and Technology, Fudan University, Shanghai 200433, China)

Abstract: The AUTOSAR platform consists of three layers: the application layer, the runtime environment, and the basic software which includes system services, EAL(ECU abstraction layer) and uCAL(uC abstraction layer). Device drivers belong to uCAL and EAL. The memory device driver provides access to different memory devices. In the paper, the specific design of AUTOSAR is followed and a Flash driver on MPC5633M microcontroller is implemented. The proposed procedure is applicable to other device drivers.

Keywords: AUTOSAR(Automotive Open System Architecture); ECU(Electronic Control Unit); uCAL(Microcontroller Abstraction Layer); flash driver; MPC563xM

1　简介

1.1 AUTOSAR 简介

AUTOSAR —— AUTomotive Open System Architecture（汽车开放式系统架构），定义了一套支持分布式的，功能驱动的汽车电子软件开发方法和电子控制单元上的软件架构标准化方案，以便应用于不同的汽车和平台。

图 1 简单展示了 AUTOSAR ECU 的软件架构。AUTOSAR 软件架构可分为 3 层：应用层，AUTOSAR RTE（Run-time Environment），和基础软件（BSW）。基础软件又包括：服务层（包括 OS，系统服务和通信），ECU 抽象层和 uC 抽象层[1]。各个层的实现都运用了模块化的方式。应用层位于 RTE 之上，包括了所有 AUTOSAR 软件组件：普通应用软件，传感器，执行器组件。应用层内的各个组件之间，以及应用层与其他层的通信都是通过 RTE 来实现的。位于 RTE 之下的都是 AUTOSAR BSW。其中服务层提供了操作系统，网络通信，存储等功能。ECU 抽象层封装了 uC 提供的各种片内设备的驱动，也实现了外部设备的驱动程序。uC 抽象层提供了各种片内设备的驱动模块，这些模块直接访问 uC 内部的外设（如 Flash，CAN 总线，SPI，ADC 等）。本文实现的 Flash 驱动模块属于 uC 抽象层。

1.2 MPC563xM 微处理器及片内 Flash 简介

MPC563xM 是飞思卡尔公司推出的 32 位，RISC 架构的微处理器，主要用于汽车电子。MPC563xM 系

① 收稿时间:2009-12-22;收到修改稿时间:2010-01-18

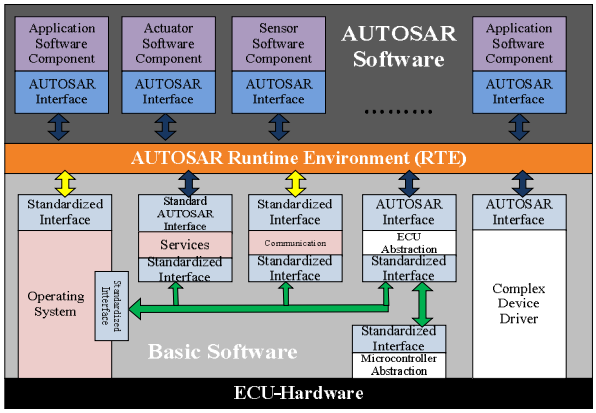

图 1 AUTOSAR ECU 软件架构

列基于高性能的 Power Architecture e200z335 内核。最高总线频率可达 80MHz，片内最多集成 94KB SRAM 和 1.5MB Flash 存储器，没有 EEPROM。图 2 展示了 MPC563xM 的 Flash 组织。1.5MB 的 Flash 被划分成三个独立的阵列，每个阵列大小为 512KB，阵列 0 由 512KB+16KB shadow block 组成，512KB 被分为 8 个小块和 2 个 128KB 的大块。阵列 1 和阵列 2 没有小块，被划分为 4 个 128KB 的大块。

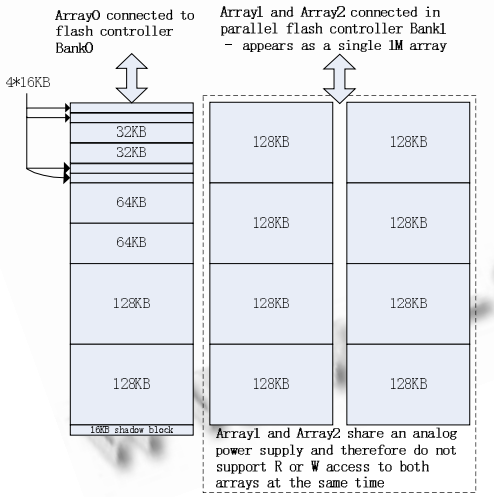

图 2 MPC563xM 的 Flash 组织

阵列 0 连接在 Flash 控制器 Bank0 上；阵列 1 和阵列 2 并行连接在 Flash 控制器 Bank1 上，而且它们共用一个模拟供电，因此不能同时访问阵列 1 和阵列 2，它们对外相当于一个 1MB 的 Flash 阵列。MPC563xM 允许在读一个阵列 1 或阵列 2 的数据的同时，对阵列 0 进行擦除或编程。这一功能可以用于 Flash 模拟 EEPROM。本文使用的 MPC5633M 芯片，片内集成 64K SRAM 和 1MB Flash，没有阵列 2[2]。

1.3 AUTOSAR Flash 驱动简介

AUTOSAR Flash 驱动提供了四种对 Flash 设备的操作：读，写，擦除，比较。ECU 抽象层中的 FEE（Flash EEPROM Emulation）模块将 Flash 设备模拟成 EEPROM。用户能访问到的存储器 API 都是针对 EEPROM 的。

AUTOSAR Flash 驱动有两种工作模式：slow 和 fast，后者在一次读写周期内可以处理更大的数据块。

AUTOSAR Flash 驱动有两种操作触发方式：中断和轮询。中断方式下，一旦有操作请求，会立刻产生中断，来处理操作。轮询方式下，需要对操作处理函数设置定时调用，如每隔 10ms 调用一次。AUTOSAR OS 提供了 BSW 调度器，它将各个驱动模块的主操作函数创建成任务，定时调用这些任务。写某 Flash 块，Flash 驱动调用 Fls_Write()函数将写命令提交给 Flash 主操作函数 Fls_MainFunction()。提交命令是异步的，真正的写操作是同步的，它由包含 Fls_MainFunction()的 OS 任务完成。

2 uC抽象层和ECU抽象层

2.1 uC 抽象层

uC 抽象层由各种片内设备的驱动模块组成，可直接访问硬件。它为上层软件抽象出了片内设备，避免了上层软件直接访问微控制器的寄存器。如图 3 所示，uC 抽象层由 I/O 驱动，通信驱动，存储器驱动和微控制器外设驱动构成。存储器驱动模块包括所有片内存储器驱动[3]。

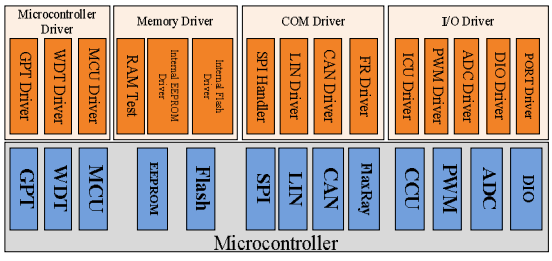

图 3 AUTOSAR uC 抽象层结构

2.2 ECU 抽象层

ECU 抽象层封装了 uC 抽象层提供的片内设备驱动程序接口，还实现了外部设备的驱动程序。ECU 抽

象层使上层软件的实现不依赖于 ECU 硬件。

图 4 展示了 AUTOSAR 存储器模块的抽象。可以看到 ECU 抽象层实现了通过 SPI 总线连接的外部 EEPROM 的驱动，外部 Flash 驱动，EEPROM 抽象，FEE 以及所有存储器的抽象[4,5]。

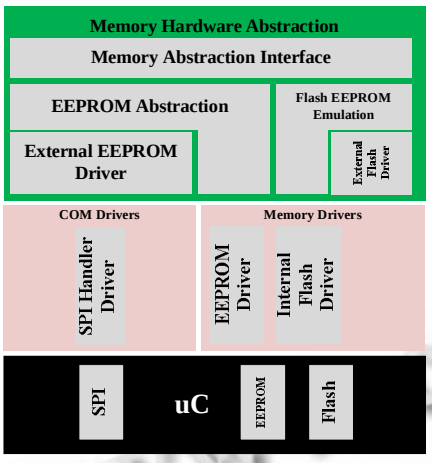

图 4　AUTOSAR 存储器模块抽象

3　Flash驱动程序的设计与实现

3.1 Flash 驱动程序文件结构[6]

根据 AUTOSAR 规范，Flash 驱动程序的文件结构如图 5 所示。Flash 驱动程序由 8 个文件组成，其中 Fls.c，Fls_Ac.c，Fls.h，Fls_Cfg.h 四个文件中实现了 Flash 的主要功能，这四个文件是每个 Flash 驱动程序所必需的。另外 4 个文件为 Fls_Lcfg.c，Fls_PBcfg.c，Fls_Irq.c，Fls_Cbk.h，它们是配置文件和可配置文件，是否需要这 4 个文件应根据驱动程序的需求。图 5 中剩余文件从其他模块引用而来：Spi.h 引用了 Spi 驱动程序，如果 Flash 是通过 Spi 总线外接的，那么就要包含 Spi.h。表 1 描述了 Flash 驱动程序中各个文件的内容。

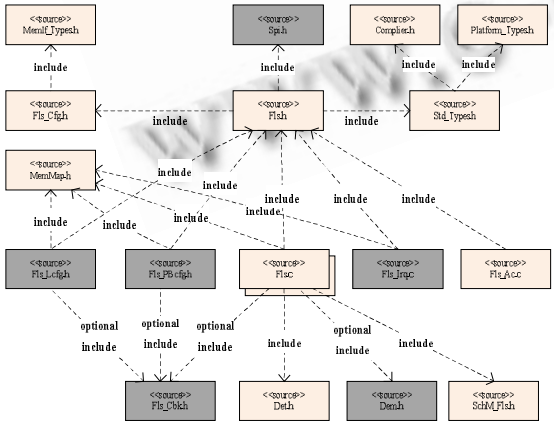

图 5　AUTOSAR Flash 驱动程序文件结构

表 1　AUTOSAR Flash 驱动程序文件内容描述

文件名	内容
Fls.h	为 Flash 驱动定义 API 和宏。
Fls.c	Flash 驱动 API 实现。
Fls_Cfg.h	定义数据结构，结构体，可配置参数，设置在预编译阶段需要用到的配置参数。
Fls_Ac.c	Flash 驱动的访问代码 API 实现。访问代码（access code）是指保存在 Flash 设备中的 Flash 驱动程序代码。我们在访问这部分代码是必须保护好它们。
Fls_Lcfg.c	设置链接阶段需要用到的配置参数。
Fls_PBcfg.c	设置运行时需要用到的配置参数。
Fls_Irq.c	中断请求的实现。
Fls_Cbk.h	Flash 驱动引用的上层软件提供的回调函数。
MemIf_Types.h	存储器接口层的类型定义。
Spi.h	SPI 总线驱动 API 及宏定义。
Std_Types.h	标准类型定义。
Complier.h	编译器相关信息。
Platform_Types.h	平台相关信息。
MemMap.h	存储数据和代码的映射。
Det.h	开发错误跟踪器的定义。
Dem.h	诊断事件管理器的定义。
SchM_Fls.h	调度管理器的定义。

3.2 Flash 驱动程序类型及 API 定义

根据 AUTOSAR 规范，本文定义了 Flash 驱动程序所需的数据类型，以及要实现的 API。表 2 描述了 Flash 数据类型，表 3 描述了 Flash API。值得一提的是，Fls_Write()，Fls_Read()，Fls_Erase()，Fls_Compare()这四个函数并不直接对 Flash 进行操作，而是向模块提交一个操作命令，真正执行操作是在 Fls_MainFunction()这个轮询函数中进行的。

表 2 Flash 数据类型描述

名称	类型	范围	描述
Fls_ConfigType	struct	硬件相关	初始化 Flash 的数据。
Fls_AddressType	uint8, uint16, uint32	8/16/32bits	访问某一 Flash 存储块的偏移地址（基地址已配置）。
Fls_LengthType	uint8, uint16, uint32	8/16/32bits	读/写/擦/比较操作的字节数。

表 3 Flash API 描述

API	内容
Fls_Init()	初始化 Flash 驱动。
Fls_Erase()	擦除 Flash 块。
Fls_Write()	写一个或多个 Flash 页。
Fls_Cancel()	取消一个正在进行的操作。
Fls_GetStatus()	返回驱动的状态。
Fls_GetJobResult()	返回最后一个操作的结果。
Fls_Read()	从 Flash 读数据。
Fls_Compare()	比较应用程序缓冲区和 Flash 某一区域的内容。
Fls_SetMode()	设置 Flash 驱动的工作模式。
Fls_GetVersionInfo()	返回版本信息。
Fls_MainFunction()	执行操作 Flash 的主函数。

3.3 AUTOSAR Flash 驱动的具体实现[2,6]

3.3.1 Fls_Init()函数的实现

Fls_Init(const FlsConfigType* ConfigPtr)，完成 Flash 驱动初始化，它接受一个配置信息结构体的指针。根据 AUTOSAR 规范和 MPC5633M 的 Flash 组织，本文设计了 Fls_ConfigType 结构体：

```
typedef struct Fls_ConfigType
{
    void (*FlsAcErase)();      //擦除 Flash 的访问代码指针
    void (*FlsAcWrite)();      //写 Flash 的访问代码指针
    uint32_t FlsCallCycle; //主调度函数的周期
    void (*FlsJobEndNotification)();      //工作完通告函数
    void (*FlsJobErrorNotification)();      //出错通告函数
    uint32_t FlsMaxReadFastMode;      //最大字节数
    uint32_t FlsMaxReadNormalMode;
    uint32_t FlsMaxWriteFastMode;
    uint32_t FlsMaxWriteNormalMode;
    FlsSector FlsSectorList[14];      //MPC5633M 的 14
                                      //Flash 块信息
}Fls_ConfigType;
```

Fls_Init()函数定义：

```
void Fls_Init(const Fls_ConfigType *ConfigPtr)
{
  //check error: check ConfigPtr and Module status
  //init hardware: MPC5633M flash init code. Init software
  FlsConfig = *ConfigPtr;
  //set module state and job result
  Fls_ModuleStatus = MEMIF_IDLE;
  Fls_JobResult = MEMIF_JOB_OK;
  return;
}
```

3.3.2 Fls_Write()函数的实现

Fls_Write()的作用是向驱动模块发送写命令，并设置源地址，目标地址和长度。Fls_Write()函数定义：

```
Std_ReturnType Fls_Write(Fls_AddressType TargetAddress, const uint8* SourceAddressPtr, Fls_LengthType Length){
  //check error: 页对齐，源地址是否为 NULL 等。
  //set cmd word, store param, set module state
  cmd = FLS_WRITE_CMD;
  offset = TargetAddress;
  Fls_ModuleStatus = MEMIF_JOB_PENDING;
  return E_OK;
}
```

3.3.3 Fls_MainFunction()函数的实现

Fls_MainFunction()函数真正执行对 Flash 设备的四种操作：读/写/擦除/比较。它被包含在 OS BSW 调度器创建的定时任务中。Fls_MainFunction()函数定义：

```
void Fls_MainFunction(){
  switch(cmd)
  {
     case FLS_WRITE_CMD:
     //call Fls_Ac_Write(), if error occurs, report
       error and
     //return. If success, set module state and job
       result
        break;
     case FLS_READ_CMD:
  }
  return;
}
```

Fls_Ac_Write()函数在 Fls_Ac.c 中，它是 Flash 访问代码，实现了对物理 Flash 设备的写操作。由于写/擦除 Flash 时，不允许读 Flash，因此写/擦除 Flash 的代码必须在另一块 FLASH 或被装载到 RAM 中运行。Fls_MainFunction()通过已配置的装载地址，调用 Fls_Ac_Write()函数。Fls_Ac_Write()的具体实现可参考文献[2]。

3.4 Flash 模拟 EEPROM

AUTOSAR 存储器模块提供的接口都是针对 EEPROM 的。EEPROM 以字节为操作单元，Flash 的读写单元是页，擦除单元是块，写时必须先擦除整个块。根据 MPC5633M 的 Flash 组织，可采用以下方法来实现 Flash 模拟 EEPROM。

如果要写的块比较小（16/32KB），可以将整个块读入 SRAM（64KB），在 SRAM 中修改相应字节，再擦除 Flash 块，最后写回。

如果我们要写 64/128KB 的块，那么需要借助阵列 1 中的某个块（128KB）。具体步骤为：1）擦除阵列 1 中的块。2）将阵列 0 的页（128 位）读入 SRAM，进行修改。3）将修改好的页写入阵列 1（2，3 两步需循环 1K 次）。4）擦除阵列 0 的块，将阵列 1 的块写回阵列 0。

4 总结

本文首先介绍了 AUTOSAR 软件架构、MPC563xM 系列微处理器及其 Flash 组织、Flash 驱动在 AUTOSAR 软件架构中的位置。接着，根据 AUTOSAR 规范，设计并实现 Flash 驱动的文件结构、数据类型及函数接口；结合 MPC5633M 的硬件特性，设计 Flash 驱动的配置结构体，实现了 Flash 驱动初始化函数、发送操作命令、Flash 主操作函数。最后介绍了 Flash 模拟 EEPROM。

利用本文提出的方法，可以实现其他 AUTOSAR 设备驱动程序。

参考文献

1 Won WJ, Son J, Park G, Kum D, Lee S. Design and Implementation Procedure of the AUTOSAR I/O Driver Cluster. ICROS-SICE International Joint Conference. 2009.

2 Semiconductor F. MPC5634M Microcontroller Reference Manual. http://www.freescale.com/. 2009.

3 Park G, Kum D, Jin S, Jung W. Implementation of AUTOSAR I/O Driver Modules for a SSPS System. International Conference on Control, Automation and Systems. 2008.

4 AUTOSAR Administration. Layered Software Architecture v2.2.2. AUTOSAR GbR. 2008.

5 AUTOSAR Administration. Specification of Module Memory Abstraction Interface v1.2.2. AUTOSAR GbR 2008.

6 AUTOSAR Administration. Specification of Module Flash Driver v2.2.3. AUTOSAR GbR. 2008.