天津大学

本科生毕业论文

学　　院　智能与计算学部

专　　业　软件工程

年　　级　2015 级

姓　　名　杨兴锋

指导教师　毕重科

2019 年 5 月 6 日

摘　　要

中文摘要一般在 400 字以内，简要介绍毕业论文的研究目的、方法、结果和结论，语言力求精炼。中英文摘要均要有关键词，一般为 3 — 7 个。字体为小四号宋体，各关键词之间要有分号。英文摘要应与中文摘要相对应，字体为小四号 Times New Roman，详见模板。

关键词： RTOS； CORTEX-M； ARM； SwitchContex； Scheduing

ABSTRACT

The upper bound of the number of Chinese characters is 400. The abstract aims at introducing the research purpose, research methods, research results, and research conclusion of graduation thesis, with refining words. Generally speaking, both the Chinese and English abstracts require the keywords, the number of which varies from 3 to 7, with a semicolon between adjacent words. The font of the English Abstract is Times New Roman, with the size of 12pt(small four).

Keywords: RTOS; CORTEX-M; ARM; SwitchContex; Scheduing

目　　录

第一章　　绪论

谷雨过后长安的清晨, 一片雾蒙蒙的, 隐约可见刚冒出小芽的青草顶着晶莹的露珠, 附身的翰墨慢慢蹲下, 撩了撩挡住视线的头发, 抬起头说道“小草真像一个个背着包袱的小人儿,”

花无邪低头微微一笑, 殊不知, 这一笑, 便是整个世界.

就在这时, 一个偏瘦的男子, 头戴斗笠, 右手握着滴血的剑已经下垂, 略显疲惫, 骑着马向这边飞奔而来, 不时回头看.

第二章　　设计

2.0.1　初衷

2.0.2　架构设计

第三章　实现

3.1 StartUp

3.1.1 Interrupt Vector Table

首先我们从中断向量表 (既中断服务程序入口地址) 开始, 当中断或异常发生的时候, CPU 自动将 PC 指向一个特定的地址, 这个地址就是中断向量表。Arm 的中断向量表一般位于内存 0x00000000~0x0000001C 处, 结构如下:

表 3-1　Arm 中断向量表

Address	Interrupt Handler
...	...
0x00000040	WWDG_IRQHandler
0x0000003C	SysTick_Handler
0x00000038	PendSV_Handler
0x00000034	Reserved
0x00000030	DebugMon_Handler
0x0000002C	SVC_Handler
0x00000028	Reserved
0x00000024	Reserved
0x00000020	Reserved
0x0000001C	Reserved
0x00000018	UsageFault_Handler
0x00000014	BusFault_Handler
0x00000010	MemManage_Handler
0x0000000C	HardFault_Handler
0x00000008	NMI_Handler
0x00000004	Reset_Handler
0x00000000	Top_Of_Stack

内存中的第一个 4 字节用来初始化 Main Stack Pointer(MSP), 第二个 4 字节是指向 reset handler 函数的指针, 是起始地址, 或者被称为 reset handler 函数的入口点, 用来初始化 Program Counter(PC).

3.1.2 BOOT

当处理器启动时首先读取两个引脚, 包括引脚 boot0 和引脚 boot1, 处理器根据这两个引脚确定引导模式. 然后, 处理器将存储在地址 0x00000000 的值复制到 Main Stack Pointer(MSP), 这一步基本上完成了 Main Stack Pointer(MSP) 的初始化. 接着处理器将存储在地址 0x00000004 的值复制到 Program Counter(PC). 程序计数器始终保存下一个要由处理器执行的操作的内存地址, 因此, 在处理器启动后, 处理器将立即开始执行 reset handler. 通常,reset handler 首先执行一些硬件初始化, 例如数据段和 BSS 段的初始化, 然后 reset handler 调用 main() 函数将控件传递给 main 函数.

表 3-2 Boot Mode

BOOT1	BOOT0	BOOT MODE
x	0	Boot from main flash memory
0	1	Boot from system memory(bootloader)
1	1	Boot from embeded SRAM

大多数 Cortex-M 处理器支持至少三种不同的引导模式, 处理器可以从片内存储器, 系统存储器或片内 SRAM 启动. 存储在系统存储器中的代码称为引导加载程序 (bootloader), 引导加载程序 (bootloader) 通常由芯片制造商提供. 引导加载程序 (bootloader) 可以升级内部闪存内的固件. 所有 STM32 微处理器在只读存储器区域 (ROM) 中都带有预编程的引导加载程序 (bootloader), 该 ROM 区域称为系统存储器.

但是, 有时您需要开发自定义引导加载程序 (bootloader), 例如: 您需要加密固

表 3-3 Memory Map

Size	Address	Memory
0.5G	0xE0000000~0xFFFFFFFF	System
1G	0xA0000000~0xE0000000	External Device
1G	0x60000000~0xA0000000	External RAM
0.5G	0x40000000~0x60000000	Peripheral
0.5G	0x20000000~0x40000000	Internal SRAM
0.5G	0x00000000~0x20000000	Code

件并将其放在 Internet 上, 以便客户可以升级固件. 在这种情况下, 您必须编写自定义引导加载程序 (bootloader) 来解码加密的固件. 表 3-3 是 Arm Cortex-M 处理器的内存映射:

每个存储区的地址范围是固定的. 接下来看一下代码区域 (表 3-4), 代码区域的范围是 0x00000000 到 0x1FFFFFFF. 顶部区域 (0x1FFF000~0x1FFFFFFF) 是保留用于存储引导加载程序的 ROM 区域. 中间区域是片内闪存. 底部区域是可以物理映射到内部闪存, 系统内存的区域. 可以物理映射到内部闪存, 系统内存和内部 SRAM 的区域.

内部闪存, 系统存储器和内部 SRAM 的起始地址也是固定的. 具体来说, 内部闪存的起始地址为 0x08000000, 系统内存的地址从 0x1FFF0000 开始.

表 3-4 Code Area

Size	Address	Memory
todo	0x1FFF77FF~0x1FFFFFFF	Options Bytes
todo	0x1FFF0000~0x1FFF77FF	System memory (bootloader)
todo	0x080X0000~0x1FFF0000	Reserved
todo	0x08000000~0x080X0000	Internal Flash
todo	0x00000000~0x08000000	Alias to flash, system memory or SRAM

现在, 我们回头再看一下引导模式 (表 3-2). 引导模式由引脚 boot1 和引脚 boot0 上的电压决定.

如果引脚 boot0 接地, 则处理器将从内部闪存引导. 如果引脚 boot0 接地, 处理器将物理映射内部闪存到底部区域. 例如, 内存地址 0x08000000 将物理映射到地址 0x00000000. 换句话说, 闪存内容可以从地址 0x00000000 或 0x08000000 访问. 所以, 当处理器启动时, 它总是从内存地址 0x00000000 和 0x00000004 分别获取堆栈指针 (SP) 和程序计数器 (PC) 的值. 因为内部闪存已经被物理映射到了起始地址 0x00000000, 所以实际上闪存就是是启动内存。

当引脚 boot1 为低电平且引脚引导 0 为高电平时, 系统存储器将物理映射到底部区域. 当处理器从内存地址 0x00000004 获取程序计数器 (PC) 的值时, 处理器实际上是从系统存储器中获取值. 也就是说, 系统存储器被选为引导存储器. 在此启动模式下, 处理器可以重新编程闪存或执行设备固件升级.

当引脚 boot1 和引脚 boot0 都为高电平时, 内部 SRAM 物理映射到底部区域, 内存地址 0x20000000 物理映射到内存地址 0x00000000 因此处理器从 SRAM 引导.

简单一点总结就是 arm 处理器从哪儿引导由 pcb 设计说了算, 然后根据 boot0/1 两个针脚的电压决定把内存的哪块映射到 0 地址区域. 然后处理器取 4 地址的值放到 pc 寄存器开始运行.

3.1.3 StartUp code

那么, 我们从开发者的角度看一下 Start Up 是如何做到的, 以及如何将启动代码放到目标引导内存中:

当链接器 (Linker) 将对象和库文件组合成单个可执行文件时, 链接器脚本 (Linker Script) 提供两种关键类型的操作, 如何对数据和代码段进行操作以及每个部分应放在内存中的位置进行操作. 编程人员可以修改链接描述文件 (Linker Script) 以将代码放在目标引导内存中. 例如这个项目中的链接描述文件:

```
MEMORY {
    FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 256K
    RAM (rwx) : ORIGIN = 0x20000000, LENGTH = 32K
}
SECTIONS {
     PROVIDE( _stack_ptr = ORIGIN(RAM) + LENGTH(RAM));
    .text :
    {   _text = .;
        KEEP(*(.vector_table))
        *(.text*)
        *(.rodata*)
        _etext = .; } > FLASH
    .data :
    {   _data = .;
        *(.data*)
        _edata = .; } > RAM AT >FLASH
    .bss :
    {   _bss = .;
        *(.bss*)
        *(COMMON)
        _ebss = .; } > RAM
}
```

从上面的描述文件可以看到代码段, 数据段再内存里面的位置.(此处应有解

释)

然后，我们再看一下如何将值 (Main Stack Pointer(SP) 和 Program Counter(PC)) 存在内存的 0x00000000 地址和 0x00000004 地址, 也就是启动代码, 比如启动汇编文件中放置了这几个值, 例如:

```
Stack_Size      EQU     0x00000400
                AREA  STACK, NOINIT, READWRITE, ALIGN=3
Stack_Mem       SPACE Stack_Size
__initial_sp
...
__Vectors       DCD     __initial_sp    ;Top of Stack
                DCD     Reset_Handler   ;ResetHandler
...
```

代码中 __initial_sp 便是 Stack Pointer(SP),Reset_Handler 便是 Program Counter(PC) 的初始值.

3.2 Interrupt

接下来我们看一下中断是如何在 Arm Cortex-M 微处理器上工作的, 以及为什么需要中断, 假设我们需要开发一个程序, 按下按钮打开红色 LED 灯, 有两种方法可以监视连接到按钮的输入引脚的逻辑状态, 一种是轮询, 另一种是中断, 轮询方法就像是每隔几秒接起电话来查看是不是有电话打进来, 中断方法就像等待电话铃响, 显而易见, 中断方法更高效. 你可以做任何事情, 知道电话响了再把它接起来. 下面是简化了的轮询代码:

```
while(1){
    read_botton_input;
    if(pushed){
        exit;
    }
}
turn_on_LED;
```

在这个循环中, 程序不断读取连接到按钮的引脚, 直到按键被按下, 程序跳出循环, 打开 LED 灯. 轮询方法是一种忙等待的方法, 处理器不停地读取输入直到按钮被按下, 显然, 轮询方法很简单但是低效. 中断方法比轮询更高效, 如果用户按下按钮, 则产生称为中断请求的电信号, 当处理器收到中断请求时, 它会自动暂停正常程序的执行, 并开始执行一个称为中断处理程序的特殊定义函数. 在中断

处理程序完成后, 处理器从暂停的地方重新启动执行常规程序.

然后我们回头看一下 Cortex-M4 的内存映射 (表 3-3), 以及代码区 (表 3-4) 和中断向量表 (表 3-1), 中断向量表保存一个存储器地址数组, 中断表中的每个条目长度为 4 个字节, 每个条目包含一个中断服务程序的起始地址, 简单地说, 中断表包含一个函数指针数组. 为每个中断类型分配一个编号, 称为中断号. 中断号用于索引中断向量表, 当触发中断 x 时,NVIC 使用中断号 x 作为索引值来查找中断 x 的相应中断服务程序的地址, 并强制处理器跳转并执行该中断服务程序.

表 3-5 Interrupt Vector Table

Interrupt Number (8 bits)	Memory Address of ISR (32 bits)
1	Interrupt Service Routine for interrupt 1
2	Interrupt Service Routine for interrupt 2
3	Interrupt Service Routine for interrupt 3
4	Interrupt Service Routine for interrupt 4
5	Interrupt Service Routine for interrupt 5
...	...

那么,NVIC 控制器如何使用中断号来查找中断向量表.

3.2.1 SysTick

3.2.2 PendSV

3.3 TCB

3.4 Context Switch

3.5 Thread Scheduling

3.6 Critical Section

3.7 Semphore

3.8 Mutex

3.9 MQ

结　论

杨兴锋是个大傻逼

外文资料

Here follows the English paper.

...

中文译文

这里就是外文资料的中文翻译。

...

致　谢

我就做了三件微小的事情，谢谢大家。