

Cortex – M0 MCU 中断向量重定位的高效方法

俞庆, 刘贯营

(南京苏美达智能技术有限公司, 南京 210032)

摘要: 为了在不具备中断向量硬件重定位的普通 Cortex – M0 架构 MCU 上实现 IAP 功能, 提出了一种中断向量重定位的通用方法。该方法不依赖于 MCU 的硬件特性, 而是基于软件实现梯子函数, 从而实现中断向量的重定位。文中深入研究了该方法并进行了改进, 大幅减小了中断向量重定位过程中的时间开销, 提升了中断响应速度。本文以 STM32F030 为对象, 详细介绍了该方法的原理, 给出了 Keil 环境下梯子函数的实现代码和分散加载文件的写法。目前, 该方法已经在实际产品中得到检验和推广。

关键词: Cortex – M0; IAP; Bootloader; 中断重定位; STM32F030

中图分类号: TP302

文献标识码: A

An Efficient Method of Interrupt Vector Relocation for Cortex – M0 MCU

Yu Qing, Liu Guanying

(Nanjing SUMEC Intelligence Technology Co., Ltd., Nanjing 210032, China)

Abstract: In order to implement the IAP function on the common Cortex-M0 MCU which does not support interrupt vector hardware relocation, a general method of interrupt vector relocation is proposed. This method does not depend on the hardware characteristics of MCU, but it is based on the ladder function implemented by software, and then realizes the relocation of interrupt vector with the ladder function. In this paper, the method is deeply studied and improved, which greatly reduces the time cost in the process of interrupt vector relocation and improves the interrupt response speed. Taking STM32F030 as the object, the paper introduces the principle of this method in detail, and gives the implementation code of ladder function and the writing method of scatter-loading file under Keil environment. At present, the method has been tested and promoted in products.

Keywords: Cortex – M0; IAP; Bootloader; interrupt relocation; STM32F030

0 引言

IAP (In Application Programming) 功能为产品软件升级提供了一个方便快捷的接口。用户通过串口、USB、CAN 总线^[1] (无需使用编程器) 即可实现产品的软件更新, 甚至可以通过以太网^[2] 或者无线网络^[3] 实现产品软件的远程升级, 大大方便了产品的功能迭代, 提升了产品的易用性。

Bootloader (以下简称 Boot) 俗称引导程序, 是实现 IAP 功能的核心。Application (简称 App) 即用户应用程序, 负责实现产品功能。Boot 和 App 这两个独立的程序在同一芯片中运行, 有各自的中断向量表, 但中断号相同。在由 Boot 程序进入到 App 程序后, 如何使 App 程序下的中断能正确响应? 这是所有 IAP 都需要面对的问题, 而解决问题的关键在于实现中断向量的重定位。根据自身硬件特性不同, 不同 MCU 实现中断向量重定位的方法不同。

1 Cortex – M3 架构 MCU 中断向量重定位

Nested Vectored Interrupt Controller (NVIC) 即嵌套向量中断控制器, 是 Cortex – M3 内核的一部分, 它管理和处理所有的异常和中断。Cortex – M3 中的 NVIC, 在地址 0xE000 ED08 处是一个 Vector Table Offset Register (VTOR) 即中断向量表偏移量寄存器, 通过修改它的值就能重新定位向量表^[4]。

进入 App 程序后, 只要在使能中断功能前设置 VTOR 寄存器值为 App 起始地址即可实现中断向量表的重定位。参考文献[5]中分析了 Cortex – M3 单片机中断向量重定位的执行过程。Cortex – M4 内核中也有 VTOR 寄存器, 因而在这些内核 MCU 上实现中断向量的重定位十分便捷、高效。

2 Cortex – M0 架构 MCU 中断向量重定位

适中的性能、极低的能耗、低廉的价格使得 Cortex – M0

架构 MCU 应用非常广泛,尤其是在物联网传感器、电动工具、电子测量、家电行业^[6]。因为 Cortex-M0 中的 NVIC 没有 VTOR 寄存器,所以此类型的 MCU 中断向量重定位变得不容易。

2.1 STM32F030 中断向量重定位

STM32F030 基于 Cortex-M0 架构,厂商为该系列的 MCU 赋予了一个特性:启动地址支持重映射。通过设置芯片 Boot 引脚电平,使得程序可从内部 Flash 或内部 RAM 中启动。芯片的 SYSCFG_CFGR1 寄存器也保存了设置状态,通过修改这个寄存器的值即可重新引导程序^[7]。

基于地址可以重新映射的特性,在 App 使能中断功能前,将中断向量表复制到内部 RAM 的起始地址,再设置 SYSCFG_CFGR1 寄存器,使其从 RAM 中启动,即可实现中断向量的重定位。

2.2 普通 Cortex-M0 架构 MCU 实现 IAP 的困境

并非所有 Cortex-M0 内核的单片机都支持地址重映射,因为这并不是内核功能的一部分^[8]。市场上存在大量使用 Cortex-M0 内核的专用 MCU,如某些电机控制的专用 MCU。它们不像 STM32F030 那样具备地址重新映射的特性,因此不能使用 2.1 小节中的方法实现中断向量的重定位。

因此,实现中断向量的重定位成为此类基于 Cortex-M0 架构 MCU 实现 IAP 功能的最大障碍。相关研究资料较少,参考文献[9]给出了一种基于 RAM 的中断跳转方法,不过效率有待提升。

3 中断向量重定位的通用方法

本文给出了实现中断向量重定位的一种通用方法。该方法不依赖于 MCU 自身硬件特性,而是基于纯软件实现。同时,结合 STM32F030 和 Keil 开发环境,详细介绍了该方法的原理,给出了相关代码。

3.1 梯子函数

给每一个中断向量增加这样一段程序:该程序负责获取 App 程序中对应中断服务函数的入口地址,然后跳转到该地址运行,从而运行 App 程序的中断服务程序。这种功能的程序像是从 Boot 中断向量通向 App 中断服务函数的梯子,因此称它为“梯子函数”。Boot 中断向量表中的每一个中断向量保存的是对应梯子函数的入口地址。

3.2 中断向量重定位过程

图 1 给出了基于该方法进行中断向量重定位的跳转流程图。App 运行期间产生中断,仍会从 Boot 中断向量表取得相应的中断向量。不过此时实际获取的是梯子函数的入口地址,然后借助梯子函数实现向 App 中断服务程序跳转的目标。

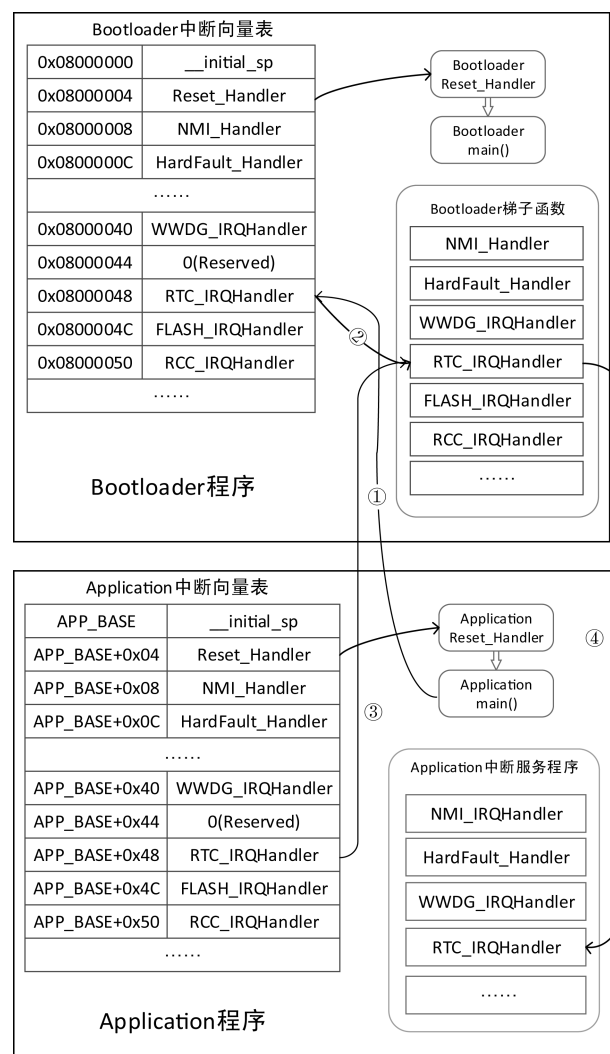


图 1 中断向量表重定位通用方法

以 RTC 中断为例,若 App 程序运行时产生了该中断,则会从 Boot 的中断向量表中找到 RTC 中断的梯子函数入口地址并跳转到该地址运行,如图 1 中箭头①和②所示。RTC 中断的梯子函数则会从 App 中的 RTC 中断向量加载对应的 RTC 中断服务程序的入口地址,如图 1 中箭头③所示。最后跳转到该地址,运行 App 的 RTC 中断服务程序,如图 1 中箭头④所示。

3.3 梯子函数的实现

梯子函数是实现从 Boot 中断向量到 App 中断服务程序跳转的关键。程序采用汇编代码实现,一个中断向量对应着一个梯子函数。部分梯子函数编写的示例如下:

1. WWDG_IRQHandler
2. ldr r0, =(0x08002000 + 0X40)
3. ldr r1, [r0]
4. bx r1
- 5.

```

6. RTC_IRQHandler
7.   ldr r0, =(0x08002000 + 0X48)
8.   ldr r1, [r0]
9.   bx r1

```

同样以 RTC_IRQHandler 为例, 0x0800 2000 为 App 的起始地址, 对应图 1 中 APP_BASE 的值。0x48 为 RTC 中断向量的偏移地址。Line7 将 RTC 中断向量偏移地址加载到 r0 寄存器。Line8 将 App 的 RTC 中断处理函数地址加载到 r1 寄存器。Line9 通过 bx 指令实现跳转到 App 的 RTC_IRQHandler 中断服务程序入口处的目标。

通过在 Boot 中为各个中断向量设置梯子函数, 借助梯子函数的桥梁作用实现中断向量表的重定位。基于该方法在 STM32F030 上进行了 IAP 功能验证, App 的中断功能运行正常, 验证了这一方法的可行性。

4 中断向量重定位改进方法 1

4.1 存在的问题

一个梯子函数需要两条 ldr 指令和一条 bx 指令, 分别需要 2 个和 3 个指令周期^[10], 因此一个梯子函数共需要 7 个指令周期。此外 Flash 的时钟频率一般比 CPU 时钟低很多。当 STM32F030 的 CPU 时钟频率高于 24 MHz 时, 访问 Flash 需要插入一个等待周期^[11]。

综上, App 中断服务函数运行前, 将会引入较大的时间开销。当要求中断能快速响应或者存在诸如 ADC 采样之类的高频中断时, 这种中断向量重定位方法会引发系统性能的明显下降, 为此需要对中断向量重定位的方法进行改进。

4.2 改进原理

改进分为两方面: 一是减少梯子函数所需的指令, 二是将梯子函数移到内部 RAM 中运行。

Boot 与 App 中各有一份梯子函数用于中断向量的重定位。Boot 中的梯子函数用于对无速度要求的中断向量重定位, 其工作原理和执行效果与前文相同。App 中的梯子函数为需要降低中断响应延迟、提高中断处理性能的中断重定位。改进方法 1 中断向量表重定位执行过程如图 2 所示。

仍以 RTC_IRQHandler 中断为例, 此时该中断向量的梯子函数 RTC_IRQHandler_App 位于 RAM 中, 且地址固定为 0x2000 0001。同时, Boot 中对应 RTC_IRQHandler 中断向量的单元保存了该地址。当 App 运行过程中发生了 RTC 中断时, MCU 会从 Boot 中的 RTC_IRQHandler 中断向量取得 0x2000 0001 地址, 并跳转到 App 下 RTC 中断的梯子函数入口运行, 对应图 2 中箭头①和②。梯子函数会从 App 的中断向量表中取得 RTC_

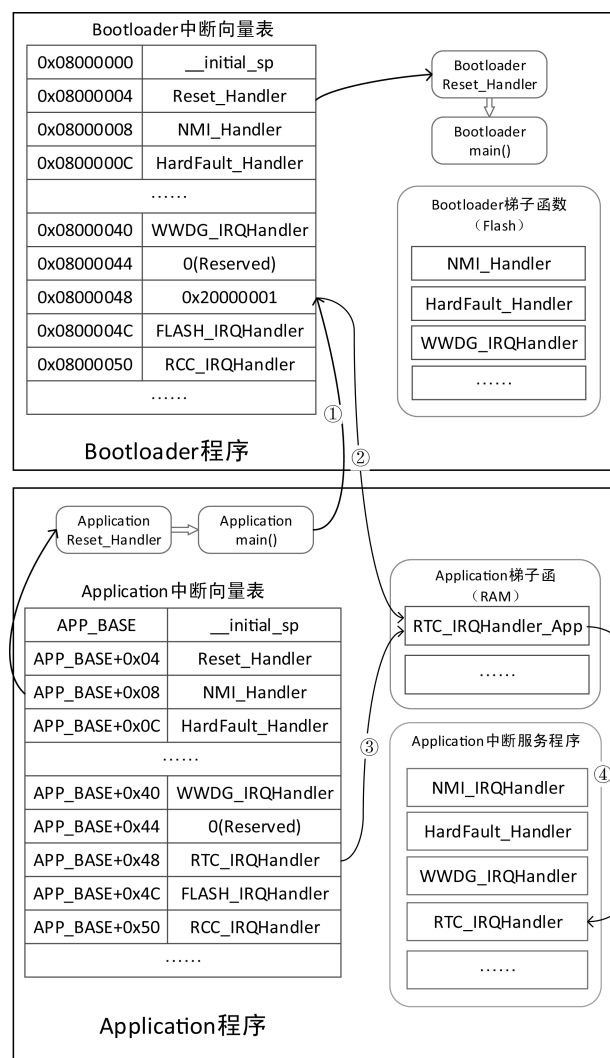


图 2 改进方法 1 中断向量表重定位执行过程

IRQHandler 中断服务函数地址, 如图 2 中箭头③所示, 最终跳转到 App 的 RTC 中断服务程序入口处运行, 如图 2 中箭头④所示。

4.3 App 中的梯子函数

App 所使用的梯子函数示例如下:

```

1. AREA ladder_app, CODE, READONLY
2.
3.   EXPORT RTC_IRQHandler_App
4.   IMPORT RTC_IRQHandler
5.
6. RTC_IRQHandler_App
7.   LDR R0, =RTC_IRQHandler
8.   BX R0
9.
10.  EXPORT Timer3_IRQHandler_App
11.  IMPORT TIM3_IRQHandler
12.

```



```

13. Timer3_IRQHandler_App
14.   LDR R0, =TIM3_IRQHandler
15.   BX R0
16.
17.   END

```

其中,Line1 定义了名为“ladder_app”的代码段。Lin6 定义了 RTC_IRQHandler_App 函数,共两条汇编指令。首先将 App 下的 RTC_IRQHandler 中断服务函数地址加载到 R0 寄存器中,接着跳转到该地址运行。相比于改进之前,此时只需要 LDR 和 BX 两条指令,总计只需要 5 个指令周期。同时该段代码在 RAM 中运行,不受 Flash 等待周期限制。

4.4 App 的分散加载设置

通过分散加载可使得 APP 中的梯子函数始终位于 RAM 固定地址,而与具体的应用程序无关。STM32F030 在 Keil 下的分散加载文件如下:

```

1.  LR_IROM1 0x08002000 0x00010000 {
2.    ER_IROM1 0x08002000 0x00010000 {
3.      *.o (RESET, +First)
4.      * (InRoot $ $ Sections)
5.      .ANY (+RO)
6.      .ANY (+XO)
7.    }
8.
9.    RW_IRAM1 0x20000000 0x00000080 {
10.     * (ladder_app, +First)
11.   }
12.
13.   RW_IRAM2 0x20000080 0x00001F80 {
14.    .ANY (+RW +ZI)
15.  }
16. }

```

其中 Line10 将“ladder_app”代码段加载到 0x2000 0000 地址,长度为 128 字节。若 App 中梯子函数较多,则预留空间需要相应增大,反之亦然。

梯子函数的入口地址可以从工程编译成功后的 map 文件获取。如图 3 所示,RTC_IRQHandler_App 函数的入口地址为 0x2000 0001,这就是前文 Boot 中 RTC_IRQHandler 向量保存的地址来源。

RTC_IRQHandler_App	0x20000001
Timer3_IRQHandler_App	0x20000005
RTC_IRQHandler	0x080036a5
TIM3_IRQHandler	0x080036b1

图 3 梯子函数与中断服务函数入口地址

5 中断向量重定位改进方法 2

前文讨论了将梯子函数搬移到 RAM 中运行,从而大

幅减小进入用户中断服务程序的时间开销的改进方法。本节讨论再改进的方法,进一步压缩梯子函数的时间开销,提高进入中断程序的效率。

5.1 梯子函数的改进

经过改进的梯子函数代码如下:

```

1. AREA ladder_app, CODE, READONLY
2.
3.   EXPORT RTC_IRQHandler_App
4.   IMPORT RTC_IRQHandler
5.
6.   RTC_IRQHandler_App
7.   B RTC_IRQHandler
8.
9.   EXPORT Timer3_IRQHandler_App
10.  IMPORT TIM3_IRQHandler
11.
12.  Timer3_IRQHandler_App
13.   B TIM3_IRQHandler
14.
15.   END

```

可以看出,只需要通过一条 B 指令就可以跳转到中断服务函数入口。该指令执行只需要 3 个时钟周期,进一步降低了梯子函数运行所需要的时间开销。

B 指令是相对跳转指令,用于无条件跳转,其支持跳转地址范围为 PC-2046~PC+2046^[4]。因此中断服务函数必须也位于内部 RAM 中,且距对应的梯子函数的跳转指令地址必须在 2046 字节之内。这一条件是容易满足的,毕竟不是所有的中断服务程序都需要这么做,而只是个别时间敏感、高频的中断需要按照这种方法处理。

5.2 中断服务函数约束条件

为实现中断处理函数的地址约束,在编写相关中断处理函数时需要添加约束属性。通过为函数设置 section 属性将中断服务程序放入自定义名称为“ramfunc_isr”的输入段。除此之外,与普通的中断处理函数无其他差别。如在 Keil 环境下,对于 RTC 中断处理函数写法如下:

```
attribute__((section("ramfunc_isr"))) void RTC_IRQHandler(void)
```

5.3 分散加载文件

改进方法 2 的分散加载文件设置如下:

```

1.  LR_IROM1 0x08002000 0x00010000 {
2.    ER_IROM1 0x08002000 0x00010000 {
3.      *.o (RESET, +First)
4.      * (InRoot $ $ Sections)
5.      .ANY (+RO)
6.      .ANY (+XO)
7.    }
8.    RW_IRAM1 0x20000000 0x00000080 {
9.     * (ladder_app, +First)

```

```

10.     }
11.     RW_IRAM2 0x20000080 0x00001F80 {
12.         *(ramfunc_isr, +First)
13.         .ANY (+RW +ZI)
14.     }
15. }
```

与改进方法 1 的分散加载文件相比,多了 Line12 的内容,即从地址 0x2000 0080 开始放置“ramfunc_isr”段的内容。

图 4 给出了 map 文件的部分内容,可以看出,梯子函数 RTC_IRQHandler_App 入口地址为 0x2000 0001,Timer3_IRQHandler_App 入口地址为 0x2000 0003。此时 App 的一个中断梯子函数占用存储空间大小仅为 2 字节。实际的中断处理函数的入口地址分别为 0x200 00081 和 0x2000 008d,均位于 RAM 中,紧接在梯子函数之后,相对地址也远小于 2046 字节,满足该改进方法的使用要求。

```

RTC_IRQHandler_App      0x20000001
Timer3_IRQHandler_App   0x20000003
RTC_IRQHandler           0x20000081
TIM3_IRQHandler          0x2000008d
```

图 4 梯子函数与中断服务函数入口地址

6 结 语

表 1 对文中提出的中断向量重定位的通用方法及其两种改进方法进行了对比,总结了 3 种方法的各自特点,实际上 3 种方法可以混合使用。比如将高频的 ADC 采样中断使用改进方法 2 进行 ADC 中断向量的重定位,此时 ADC 中断额外的时间开销最小,同时由于该中断服务程序在 RAM 中运行,性能也将大幅提高。将中频的定时器中断使用改进方法 1 进行中断向量重定位,其他普通的中断向量采样通用方法进行重定位。

表 1 中断向量重定位 3 种方法对比

重定位方法	梯子函数	App 中断服务程序	时间开销	特点
通用方法	在 Boot 程序的 Flash 中运行, 3 条指令	无特殊要求	7 个指令周期 0.29 μ s@48MHz	不占用 RAM,时间 开销最大
改进方法 1	在 App 程序的 RAM 中运行, 2 条指令	无特殊要求	5 个指令周期 0.10 μ s@48MHz	时间开销中等,占用 少量 RAM
改进方法 2	在 App 程序的 RAM 中运行, 1 条指令	相对中断向量地址偏移量 需小于 2046B	3 个指令周期 0.06 μ s@48MHz	时间开销最小,中断程序执行最快, 占用 RAM 最多

基于本文提出的中断向量表重定位方法,在 STM32F030 和 Keil 开发环境下进行了验证,结果正确。此外,在其他品牌的基于 Cortex-M0 内核的 MCU 上进行了验证,也取得了成功。目前,基于该方法实现的 IAP 功能已经在产品中批量使用,效果较好。

虽然文中提出的中断向量重定位的方法是以 Cortex-M0 内核的 MCU 为对象进行讨论和测试的,但是其不依赖于硬件的特殊性,因此也可以在其他内核的 MCU 上进行推广应用。

参考文献

- [1] 王义泽,蔡弘. 基于 CAN 总线的 AVR 单片机 IAP 设计实现[J]. 电子技术,2018(1):57-60.
- [2] 石盼,何东伟,徐博铭. 基于 LWIP 的 IAP 远程升级技术研究[J]. 机电一体化,2017(5):31-34.
- [3] 罗晶晶. 基于 4G 网络的嵌入式设备远程升级系统设计与实现[D]. 长春:吉林大学,2021:1-2.
- [4] Joseph Yiu. Cortex-M3 权威指南[M]. 宋岩,译. 北京:北京航空航天大学出版社,2009:83-107.

- [5] 马建,黄增波,李泽芳. 煤矿安全监控系统传感器在线升级技术研究[J]. 煤矿安全,2022(4):136-137.
- [6] 范云龙,方安平,李宁. Cortex-M0 处理器初探[J]. 单片机与嵌入式系统应用,2010(6):78-81.
- [7] 意法半导体. Reference manual STM32F030, 2017: 144-146.
- [8] Joseph Yiu. ARM Cortex-M0 权威指南[M]. 吴常玉,等译. 北京:清华大学出版社,2013:104-105.
- [9] 韩雨泓,陈良勇. 无中断向量重定位单片机中实现 IAP 和 APP 中断的方法[J]. 信息技术与网络安全,2020(2): 53-56.
- [10] 意法半导体. Cortex-M0 Technical Reference Manual, 2009.
- [11] 意法半导体. STM32F030x Datasheet,2013.

俞庆(工程师),主要从事机器人产品的电机控制和嵌入式应用开发。通信作者:俞庆,yuqing@sumec.com.cn。

(责任编辑:薛士然 收稿日期:2023-05-11)