

在 RAM 中执行中断函数使用指南

概述

在实际的使用场景中，为了运行效率，通常需要将某些函数放置在 RAM 中运行（因为 RAM 的取指速度比 Flash 快非常多），而在电机控制领域将中断函数放置在 RAM 的需求尤为普遍，这份手册将重点介绍在使用不同编译器时将中断函数放置在 RAM 中的方法。

适用范围	
SPC1125 系列	SPC1125, SPC1128, SPD1121
SPC1168 系列	SPC1155, SPC1156, SPC1158, SPC1168, SPD1148, SPD1178, SPD1188, SPD1163, SPM1173
SPC2168 系列	SPC2168, SPC2165, SPC2166, SPC1198
SPC1169 系列	SPC1169, SPD1179, SPD1176, SPD1177, SPD1179B
SPC2188 系列	SPC1185, SPC2188
SPC1198B 系列	SPC1198B

目录

- 1 典型中断函数调用关系6
- 2 KEIL 配置中断代码执行在 RAM7
 - 2.1 将 Code 配置为在 RAM 中运行 7
 - 2.2 将 Data 配置为存储在程序的静态存储区 8
- 3 GCC 配置中断代码执行在 RAM 10
 - 3.1 将 Code 配置为在 RAM 中运行 10
 - 3.2 将 Data 配置为存储在程序的静态存储区 11
- 4 IAR 配置中断代码执行在 RAM 13
 - 4.1 将 Code 配置为在 RAM 中运行 13
 - 4.2 将 Data 配置为存储在程序的静态存储区 15
- 5 程序调用 Flash Controller..... 16
 - 5.1 将中断向量表移动到 RAM 中..... 16

图片列表

图 1-1: 中断函数调用关系	6
图 2-1: TIMER1_IRQHandler/ A/ B/ B_1 执行在 RAM	8
图 2-2: TIMER1_IRQHandler/ A/ B/ B_1/ gi16SinTable 执行在 RAM	9
图 3-1: TIMER1_IRQHandler/ A/ B/ B_1 执行在 RAM	11
图 3-2: TIMER1_IRQHandler/ A/ B/ B_1/ gi16SinTable 执行在 RAM	12
图 4-1: TIMER1_IRQHandler/A/B/B_1 执行在 RAM	14
图 4-2: TIMER1_IRQHandler/A/B/B_1/gi16SinTable 执行在 RAM	15
图 5-1: Flash 存储器访问接口	16
图 5-2: 全部执行在 RAM	16

版本历史

版本	日期	作者	状态	变更
A/0	2023-11-23	HangSu	已过期	1. 首次发布。
A/1	2023-11-23	C.Chai	已过期	1. 文档改名。
C/0	2024-08-27	LemengZhou	已过期	1. 增加文档适用范围。 2. 增加 章节 0 ，描述中断向量表移动到 RAM 区的方法。 3. 修改为通用指南。
C/1	2025-03-31	HangSu	已发布	1. 增加文档适用范围。

术语或缩写

术语或缩写	描述
RAM	Random Access Memory, 随机存取存储器

1 典型中断函数调用关系

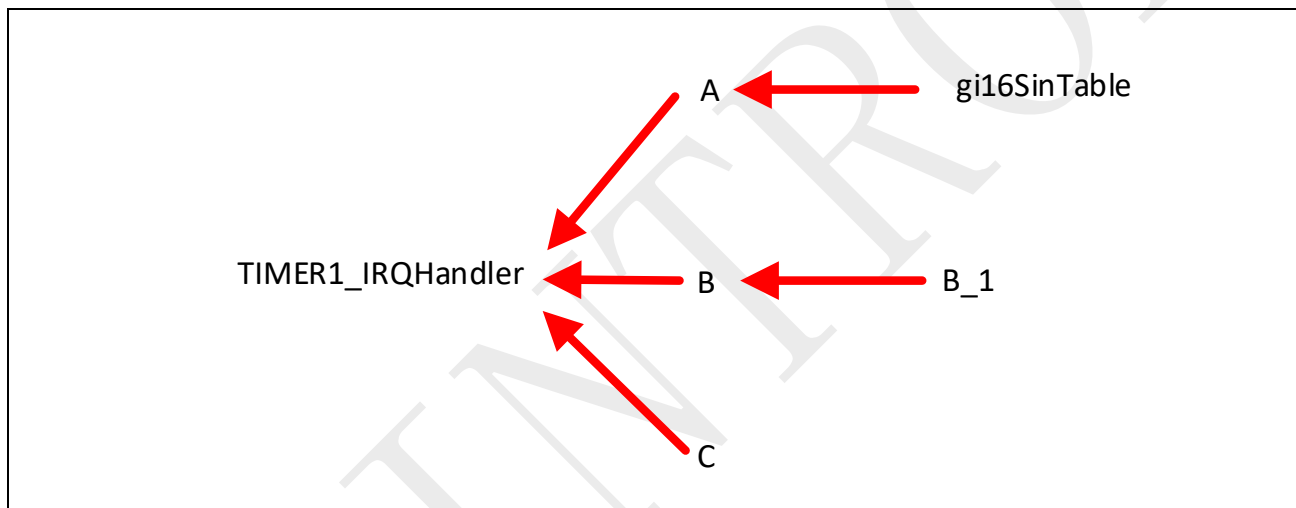
典型中断函数调用关系如图 1-1 所示。TIMER1_IRQHandler 中调用了函数 A，函数 B 和函数 C，在函数 A 中调用了 const 变量 gi16SinTable，在函数 B 中调用了子函数 B_1。

在实际的应用过程中：

1. 可能 TIMER1_IRQHandler、A、B、B_1 函数运行在 RAM；
2. 也有可能是 A、B、B_1、C 运行在 RAM；
3. 也还有可能是其它情况。

本文后续章节将描述不同 IDE 中配置函数运行在 RAM 中的详细方法。

图 1-1：中断函数调用关系



2 KEIL 配置中断代码执行在 RAM

2.1 将 Code 配置为在 RAM 中运行

首先，在 sct 文件中指定“RAMCODE”段的位置。

Project.sct

```
LR_IROM1 0x10000000 0x00010000 { ; load region size_region
ER_IROM1 0x10000000 0x00010000 { ; load address = execution address
    *.o (RESET, +First)
    *(InRoot$$Sections)
    .ANY (+RO)
}
RW_IRAM1 0x1FFFC000 0x00004000 { ; RW data
    ; Must put all code refered in interrupt function into RAM
    *.o (RAMCODE)
    .ANY (+RW +ZI)
}
}
```

- [1] 上方 Project.sct，是以 SPC1128 芯片为例。
 [2] 黄框内为 SPC1128 的代码段存储范围，红框内为数据段存储范围。

注意：

1. 代码段的存储范围，需要根据不同型号芯片 TRM 手册，查询其 FLASH 区在存储空间上的地址映射进行填写。
2. 数据段的存储范围，需要根据不同型号芯片 TRM 手册，查询其 RAM 区在存储空间上的地址映射进行填写。

其次，采用__attribute__ ((section("RAMCODE")))关键字标记所有希望放置在“RAMCODE”段的函数。

main.c

```
__attribute__ ((section("RAMCODE"))) void TIMER1_IRQHandler(void)
{
    i16Result = A(-16384);

    B(100, &i32_PWM_A, &i32_PWM_B, &i32_PWM_C);

    C(&i32_PWM);

    /* Clear the INT */
    TIMER_ClearInt(TIMER1);
}
```

func.h

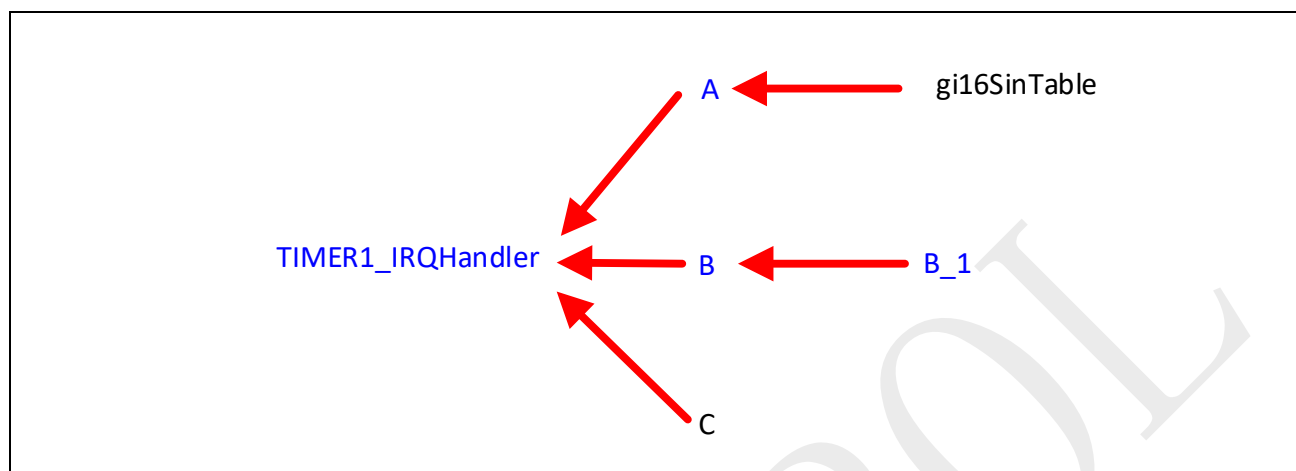
```
__attribute__ ((section("RAMCODE"))) int16_t A(int16_t i16Theta);

__attribute__ ((section("RAMCODE"))) void B(int32_t i32_PWM_Full_Scale, int32_t* pi32_PWM_A, int32_t* pi32_PWM_B, int32_t* pi32_PWM_C);

__attribute__ ((section("RAMCODE"))) int16_t B_1(int16_t in, int16_t sat);
```

若按照以上的步骤操作，此时 TIMER1_IRQHandler、A、B、B_1 函数将运行在 RAM 中，如图 2-1 所示。

图 2-1: TIMER1_IRQHandler/ A/ B/ B_1 执行在 RAM



2.2 将 Data 配置为存储在程序的静态存储区

ARMCC 编译器默认状态下对待 const 变量以及 static 变量的处理方式不同：

- const 变量会放置在只读存储区；
- static 变量会放置在静态存储区；

以上描述仅限于 ARMCC 默认状态下的行为，具体动作取决于编译器和编译器选项。

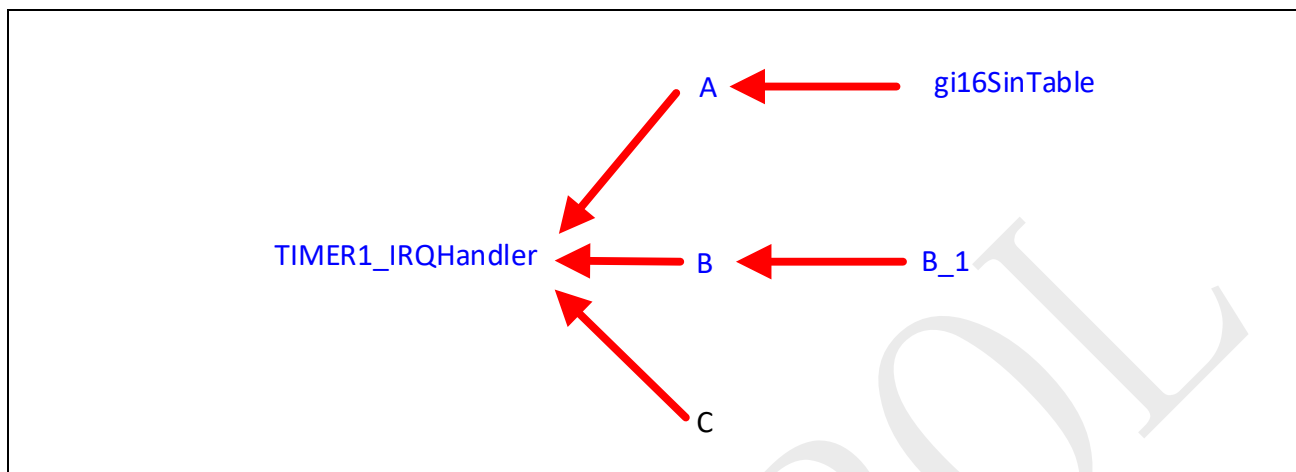
对于函数 A 中调用的 const 变量，编译器将其分配到 Flash 空间中。所以，若将程序代码放置在 RAM 之后发现还需进一步提升程序的执行效率，且 RAM 空间仍有富余，则可采用 static 关键字替代 const 关键字，从而将 gi16SinTable 存储在 RAM 的静态存储区。

func.c

```
static int16_t gi16SinTable[513] =
{
    0, 201, 402, 603, 804, 1005, 1206, 1407, 1608, 1809, 2009,
    2210, 2410, 2611, 2811, 3012, //16
    3212, 3412, 3612, 3811, 4011, 4210, 4410, 4609, 4808, 5007, 5205,
    5404, 5602, 5800, 5998, 6195, //32
    6393, 6590, 6786, 6983, 7179, 7375, 7571, 7767, 7962, 8157, 8351,
    8545, 8739, 8933, 9126, 9319, //48
    .....
    6393, 6195, 5998, 5800, 5602, 5404, 5205, 5007, 4808, 4609, 4410,
    4210, 4011, 3811, 3612, 3412, //496
    3212, 3012, 2811, 2611, 2410, 2210, 2009, 1809, 1608, 1407, 1206,
    1005, 804, 603, 402, 201, //512
    0
};
```


若按照以上的步骤操作，此时 TIMER1_IRQHandler、A、B、B_1 函数将运行在 RAM 中，且 gi16SinTable 存储在 RAM，如图 2-2 所示。

图 2-2: TIMER1_IRQHandler/ A/ B/ B_1/ gi16SinTable 执行在 RAM



3 GCC 配置中断代码执行在 RAM

3.1 将 Code 配置为在 RAM 中运行

首先，GCC 在 Id 文件中指定“RAMCODE”段的位置。

project.ld

```
/* Initialized data sections goes into RAM, load LMA copy after code */
.data :
{
    . = ALIGN(4);
    _sdata = .;          /* create a global symbol at data start */
    *(.data)              /* .data sections */
    *(.data*)              /* .data* sections */
    KEEP (* (RAMCODE))
    . = ALIGN(4);
    _edata = .;          /* define a global symbol at data end */
} >RAM AT> FLASH
```

其次，采用__attribute__((section("RAMCODE")))关键字标记所有希望放置在“RAMCODE”段的函数。

main.c

```
__attribute__((section("RAMCODE"))) void TIMER1_IRQHandler(void)
{
    i16Result = A(-16384);

    B(100, &i32_PWM_A, &i32_PWM_B, &i32_PWM_C);

    C(&i32_PWM);

    /* Clear the INT */
    TIMER_ClearInt(TIMER1);
}
```

func.h

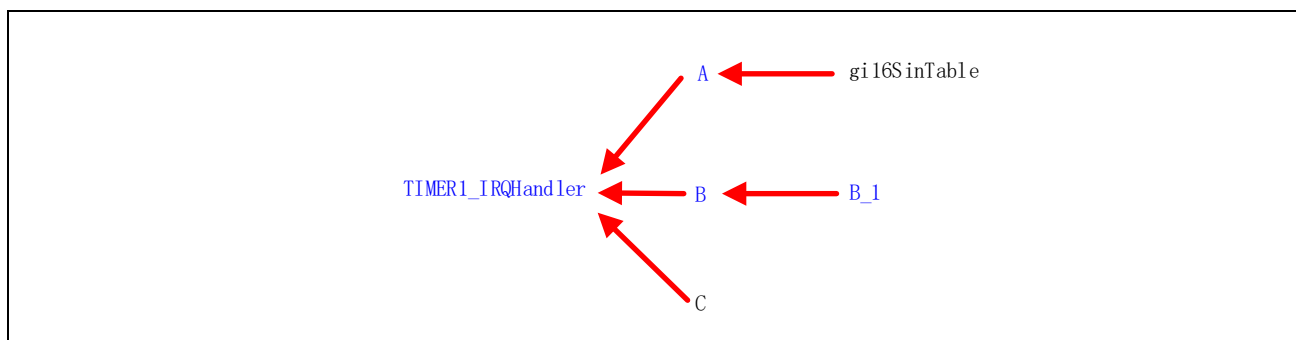
```
__attribute__((section("RAMCODE"))) int16_t A(int16_t i16Theta);

__attribute__((section("RAMCODE"))) void B(int32_t i32_PWM_Full_Scale, int32_t*
pi32_PWM_A, int32_t* pi32_PWM_B, int32_t* pi32_PWM_C);

__attribute__((section("RAMCODE"))) int16_t B_1(int16_t in, int16_t sat);
```

若按照以上的步骤操作，此时 TIMER1_IRQHandler、A、B、B_1 函数将运行在 RAM 中，如图 3-1 所示。

图 3-1: TIMER1_IRQHandler/ A/ B/ B_1 执行在 RAM



3.2 将 Data 配置为存储在程序的静态存储区

GCC 编译器默认状态下对待 `const` 全局变量以及 `static` 全局变量或普通全局变量的处理方式不同：

- `const` 全局变量会放置在只读存储区；
- `static` 全局变量或普通全局变量会放置在静态存储区；

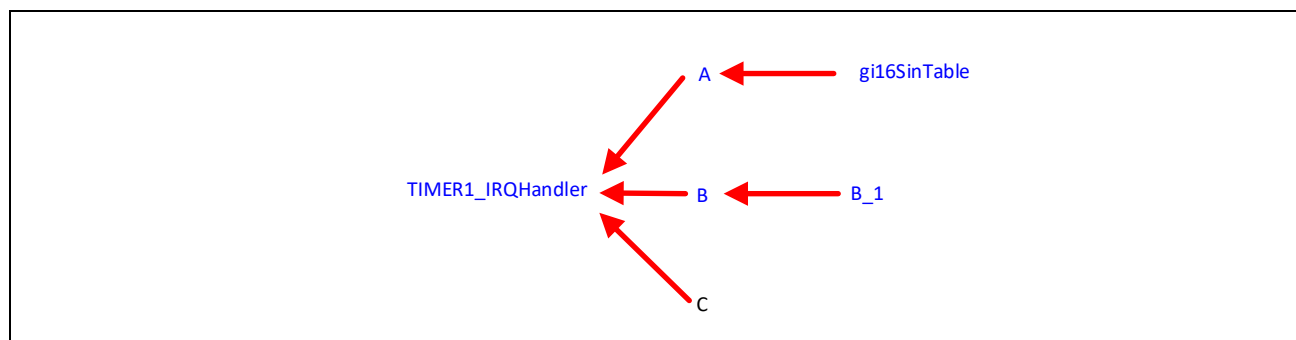
以上描述仅限于 ARMCC 默认状态下的行为，具体动作取决于编译器和编译器选项。

对于函数 A 中调用的 `const` 变量，编译器将其分配到 Flash 空间中。所以，若将程序代码放置在 RAM 之后发现还需进一步提升程序的执行效率，且 RAM 空间仍有富余，则可采用删除 `const` 关键字，或替换为 `static` 关键字，从而将 `gi16SinTable` 存储在 RAM 的静态存储区。

```
static int16_t gi16SinTable[513] =
{
    0, 201, 402, 603, 804, 1005, 1206, 1407, 1608, 1809, 2009,
    2210, 2410, 2611, 2811, 3012, //16
    3212, 3412, 3612, 3811, 4011, 4210, 4410, 4609, 4808, 5007, 5205,
    5404, 5602, 5800, 5998, 6195, //32
    6393, 6590, 6786, 6983, 7179, 7375, 7571, 7767, 7962, 8157, 8351,
    8545, 8739, 8933, 9126, 9319, //48
    .....
    6393, 6195, 5998, 5800, 5602, 5404, 5205, 5007, 4808, 4609, 4410,
    4210, 4011, 3811, 3612, 3412, //496
    3212, 3012, 2811, 2611, 2410, 2210, 2009, 1809, 1608, 1407, 1206,
    1005, 804, 603, 402, 201, //512
    0
};
```

若按照以上的步骤操作，此时 `TIMER1_IRQHandler`、`A`、`B`、`B_1` 函数将运行在 RAM 中，且 `gi16SinTable` 存储在 RAM，如图 3-2 所示。

图 3-2: TIMER1_IRQHandler/ A/ B/ B_1/ gi16SinTable 执行在 RAM



4 IAR 配置中断代码执行在 RAM

IAR 使用的是 IAR C/C++编译器，这个编译器将中断函数放置在 RAM 中执行的方法与之前描述的 ARMCC 以及 GCC 方法不同，需要特别注意。

4.1 将 Code 配置为在 RAM 中运行

采用__ramfunc 关键字标记所有希望执行在 RAM 中的函数。

main.c

```
__ramfunc ((section("RAMCODE"))) void TIMER1_IRQHandler(void)
{
    i16Result = A(-16384);

    B(100, &i32_PWM_A, &i32_PWM_B, &i32_PWM_C);

    C(&i32_PWM);

    /* Clear the INT */
    TIMER_ClearInt(TIMER1);
}
```

func.h

```
__ramfunc ((section("RAMCODE"))) int16_t A(int16_t i16Theta);

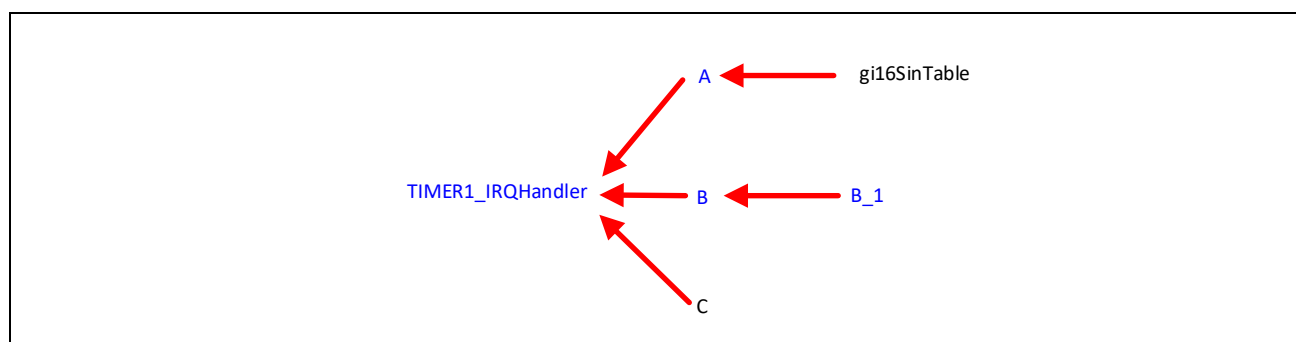
__ramfunc ((section("RAMCODE"))) void B(int32_t i32_PWM_Full_Scale,int32_t*
pi32_PWM_A,int32_t* pi32_PWM_B,int32_t* pi32_PWM_C);

__ramfunc ((section("RAMCODE"))) int16_t B_1(int16_t in,int16_t sat);
```

注意： 若 TIMER1_IRQHandler 标记为__ramfunc，但是其子函数 C 没有标记为__ramfunc，会伴有编译警告” Call to a non __ramfunc function (C) from within a __ramfunc function”。若程序执行效率已达到需求，可忽略该警告。

若按照以上的步骤操作，此时 TIMER1_IRQHandler、A、B、B_1 函数将运行在 RAM 中，如图 4-1 所示。

图 4-1: TIMER1_IRQHandler/A/B/B_1 执行在 RAM



4.2 将 Data 配置为存储在程序的静态存储区

IAR C/C++编译器默认状态下对待 const 变量以及 static 变量的处理方式不同：

- const 变量会放置在只读存储区；
- static 变量会放置在静态存储区；

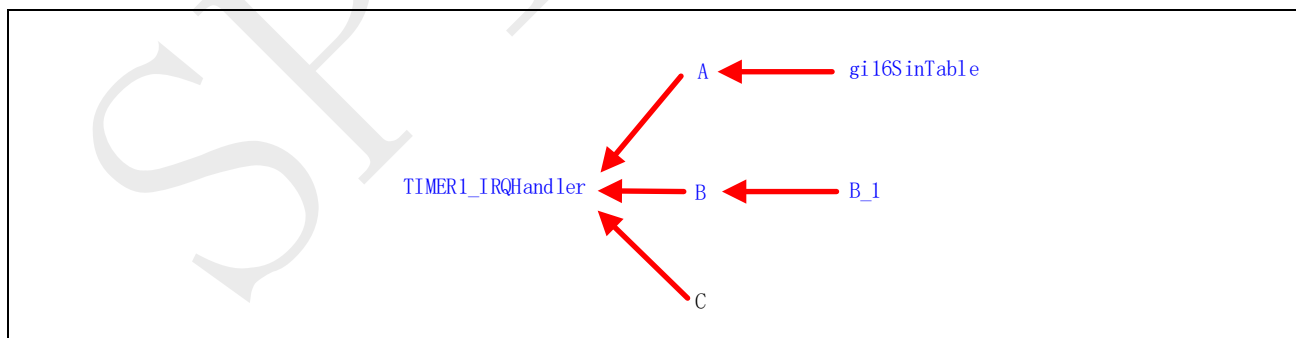
对于函数 A 中调用的 const 变量，编译器将其分配到 Flash 空间中，并伴有编译警告“Possible rom access (gi16SinTable) from within a __ramfunc function”。所以，若将程序代码放置在 RAM 之后发现还需进一步提升程序的执行效率，且 RAM 空间仍有富余，则可采用 static 关键字替代 const 关键字，从而将 gi16SinTable 存储在 RAM 的静态存储区。

func.c

```
static int16_t gi16SinTable[513] =
{
    0,    201,   402,   603,   804,   1005,   1206,   1407,   1608,   1809,   2009,
  2210,  2410,  2611,  2811,  3012,  //16
    3212,  3412,  3612,  3811,  4011,  4210,  4410,  4609,  4808,  5007,  5205,
  5404,  5602,  5800,  5998,  6195,  //32
    6393,  6590,  6786,  6983,  7179,  7375,  7571,  7767,  7962,  8157,  8351,
  8545,  8739,  8933,  9126,  9319,  //48
    .....
    6393,  6195,  5998,  5800,  5602,  5404,  5205,  5007,  4808,  4609,  4410,
  4210,  4011,  3811,  3612,  3412,  //496
    3212,  3012,  2811,  2611,  2410,  2210,  2009,  1809,  1608,  1407,  1206,
  1005,   804,   603,   402,   201,  //512
    0
};
```

若按照以上的步骤操作，此时 TIMER1_IRQHandler、A、B、B_1 函数将运行在 RAM 中，且 gi16SinTable 存储在 RAM，如图 4-2 所示。

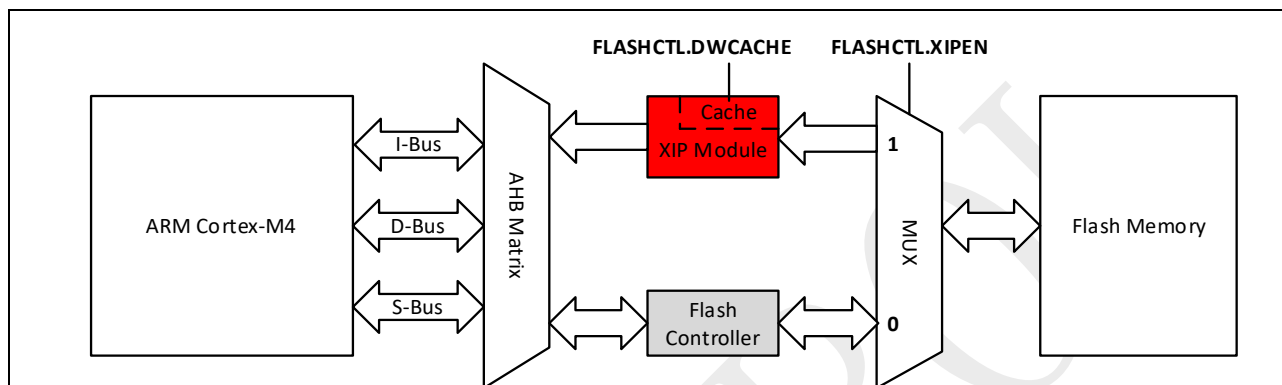
图 4-2：TIMER1_IRQHandler/A/B/B_1/gi16SinTable 执行在 RAM



5 程序调用 Flash Controller

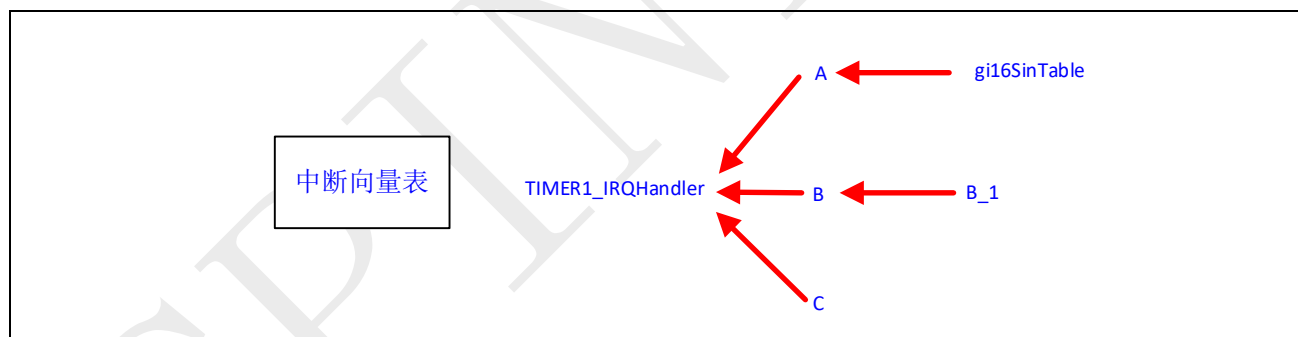
当调用 ROM 中的 Flash 驱动库（pHWLIB->XXX 函数）时，这些函数执行期间会关闭 Main Flash XIP，此时当中断来临，CPU 无法通过 XIP 访问 Main Flash 中 Code 或 Data，从而造成异常，如图 5-1 所示。

图 5-1: Flash 存储器访问接口



如果此时仍要响应所有中断，需要把中断向量表、TIMER1_IRQHandler、A、B、B_1、C、sgi16SinTable 都放到 RAM 的存储区，如图 5-2 所示。

图 5-2: 全部执行在 RAM



5.1 将中断向量表移动到 RAM 中

将中断向量表移动到 RAM 中，需要进行如下步骤：

- 通过创建全局变量，为中断向量表开辟 RAM 区中的存储空间；
- 将中断向量表内容从源 FLASH 区位置搬移到目标 RAM 区空间；
- 将中断向量表重映射到目标 RAM 区空间首地址；

通过如下操作，在工程进入 main 函数后，首先进行中断向量表的移动，再进行其他初始化及逻辑动作。

main.c

```
/* Define vector number */
#define VECTOR_NUM (45u)
/* Define vector table start address */
#define VectorTableLoadAddr (0x10000000u)
/* Define vector table RAM region */
uint32_t u32aRAMVector[VECTOR_NUM] __attribute__((aligned(0x400))) = {0};

/* Other Code */
//TODO

/* Copy vector table from FLASH to RAM */
for(u32VectorCnt = 0; u32VectorCnt < VECTOR_NUM; u32VectorCnt++)
{
    u32aRAMVector[u32VectorCnt] = *((volatile uint32_t *) (VectorTableLoadAddr \
                                                             + u32VectorCnt * 4));
}
/* Retarget the vector table start address */
SCB->VTOR = (uint32_t)u32aRAMVector;
```

- [1] 上方代码段，是以 SPC1128 芯片为例。
- [2] 黄框内为 SPC1128 中断向量表个数，红框内为中断向量表默认首地址。

- 注意：
1. 中断向量表的个数，需要根据不同型号芯片 TRM 手册，查询获取。
 2. 中断向量表的起始位置，需要根据不同型号芯片 TRM 手册，默认位于 FLASH 区的首地址。
 3. 中断向量表 RAM 区存储空间，需要根据不同型号芯片 TRM 手册，获取其 RAM 区范围，需注意所定义的 RAM 区空间地址需要按照 0x400 地址对齐。