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 的需求尤为旺盛，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。

本手册适用范围：

适用范围
SPC1169，SPD1179，SPD1176

本手册以 **64K SPD1179** 为例：

- 文中有关 Flash 及 RAM 地址范围的描述需要根据不同的产品进行设定。

SPINTROL

目录

1 典型中断函数调用关系 7

2 KEIL 配置中断代码执行在 RAM 8

2.1 将 Code 配置为在 RAM 中运行 8

2.2 将 Data 配置为存储在程序的静态存储区 9

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 12

4.1 将 Code 配置为在 RAM 中运行 12

4.2 将 Data 配置为存储在程序的静态存储区 13

5 程序调用 Flash Controller 14

SPINTROL

©2023 旋智电子科技（上海）有限公司

图片列表

图 1-1：中断函数调用关系 7
图 1-2：TIMER1_IRQHandler, A, B, B_1 执行在 RAM 7
图 1-3：A, B, B_1, C 执行在 RAM 7
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 10
图 3-2：TIMER1_IRQHandler, A, B, B_1, gi16SinTable 执行在 RAM 11
图 4-1：TIMER1_IRQHandler, A, B, B_1 执行在 RAM 12
图 4-2：TIMER1_IRQHandler, A, B, B_1, gi16SinTable 执行在 RAM 13
图 5-1：Flash 存储器访问接口 14
图 5-2：全部执行在 RAM 14

SPINTROL

表格列表

未找到图形项目表。

SPINTROL

©2023 旋智电子科技（上海）有限公司

版本历史

版本	日期	作者	状态	变更
A/0	2023-11-23	HangSu	Outdated	首次发布。
A/1	2023-11-23	C.Chai	Released	文档改名。

SPINTROL

SPIN TROL

术语或缩写

术语或缩写	描述

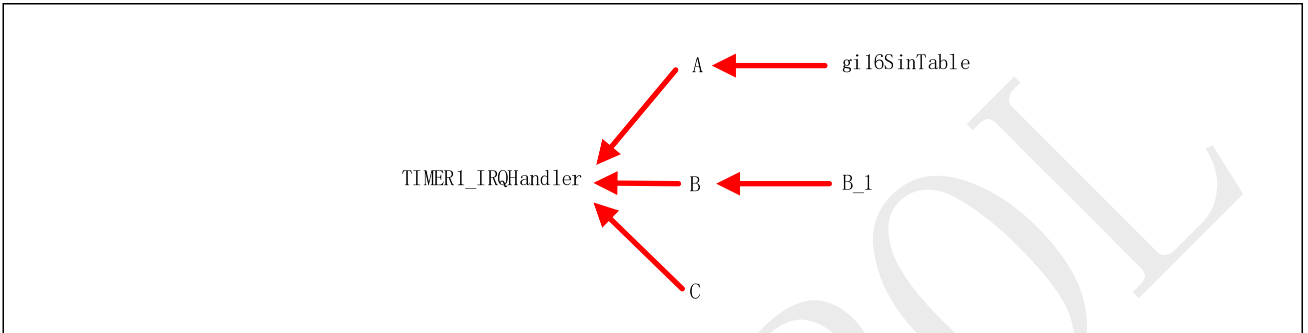
SPINTROL

©2023 旋智电子科技（上海）有限公司

1 典型中断函数调用关系

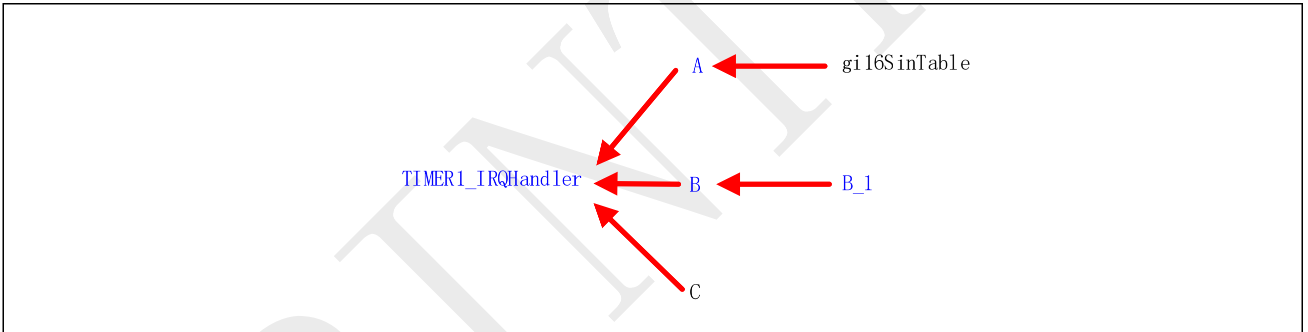
典型中断函数调用关系如图 1-1 所示。TIMER1_IRQHandler 中调用了函数 A，函数 B 和函数 C，在函数 A 中调用了 const 变量 gi16SinTable，在函数 B 中调用了子函数 B_1。

图 1-1：中断函数调用关系



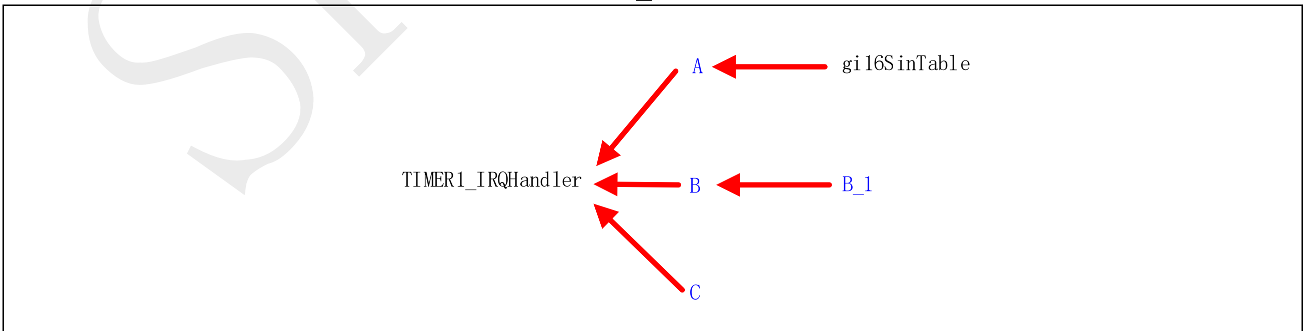
下面演示 TIMER1_IRQHandler, A, B, B_1 执行在 RAM，如图 1-2 所示。

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



也可以根据需要采用其他拓补结构，例如 A, B, B_1, C 执行在 RAM，如图 1-3 所示。

图 1-3：A, B, B_1, C 执行在 RAM



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 0x0000F000  {    ; load region size_region
  ER_IROM1 0x10000000 0x0000F000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x1FFFC000 0x00003000  {  ; RW data
   ; Must put all code refered in interrupt function into RAM
   *.o (RAMCODE)
   .ANY (+RW +ZI)
  }
}
```

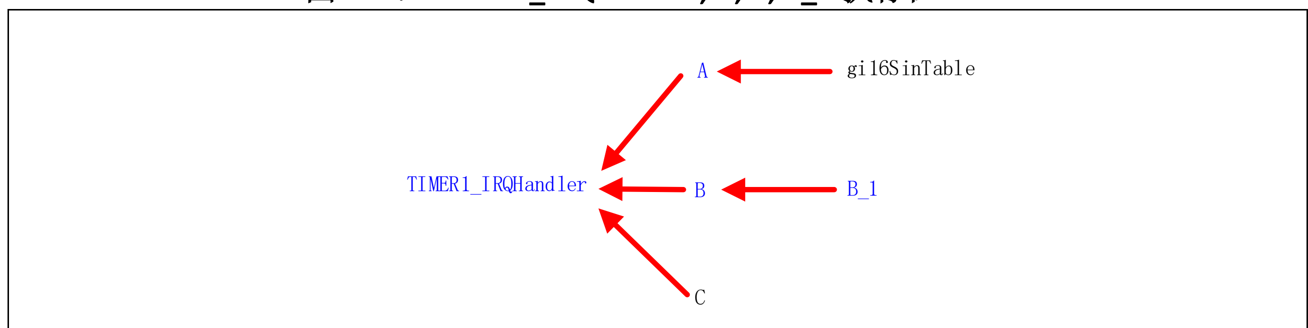
其次，采用__attribute__ ((section("RAMCODE")))关键字标记所有希望放置在“RAMCODE”段的函数。

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);
```

此时拓补结构如图 2-1 所示。

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



©2023 旋智电子科技（上海）有限公司

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；

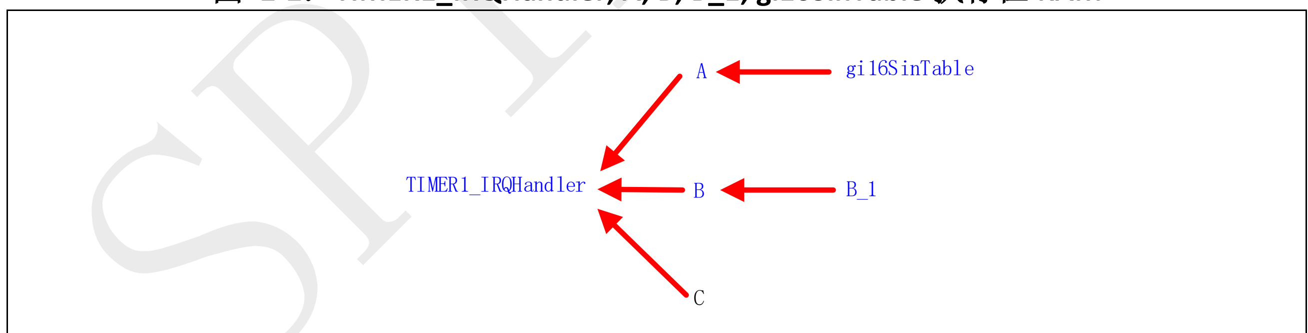
以上描述仅限于 ARMCC 默认状态下的行为，具体动作取决于编译器和编译器选项。

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
};
```

此时拓补结构如图 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 ld 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
```

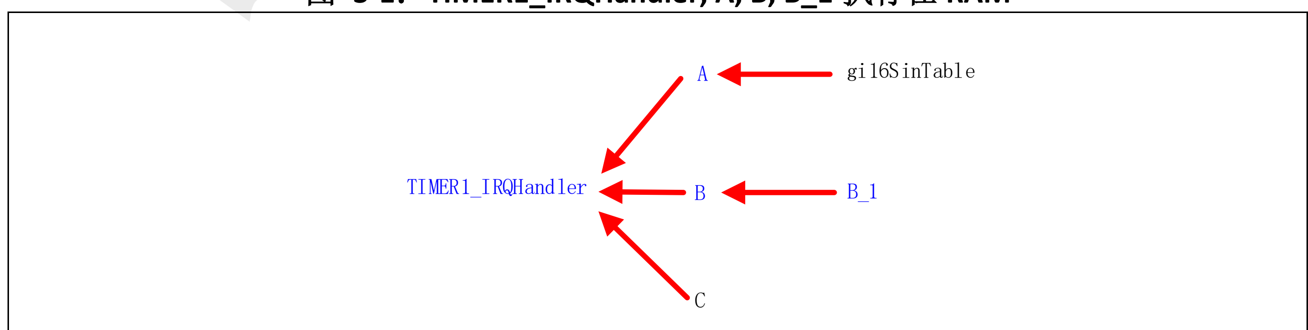
其次，采用__attribute__ ((section("RAMCODE")))关键字标记所有希望放置在“RAMCODE”段的函数。

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);
```

此时拓补结构如图 3-1 所示。

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



©2023 旋智电子科技（上海）有限公司

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 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。

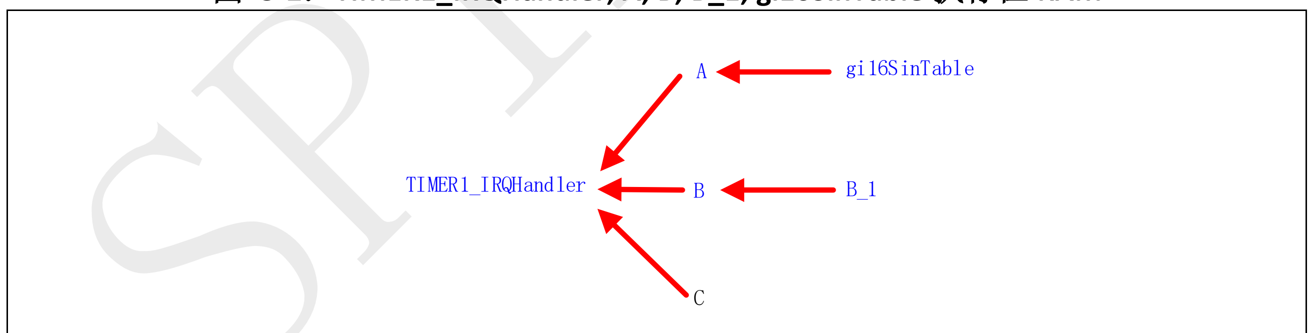
对于函数 A 中调用的 const 变量，编译器将其分配到 Flash 空间中。所以，若将程序代码放置在 RAM 之后发现还需进一步提升程序的执行效率，且 RAM 空间仍有富余，则可采用 static 关键字替代 const 关键字，从而将 gi16SinTable 存储在 RAM 的静态存储区。

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
};
```

此时拓补结构如图 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);
}
```

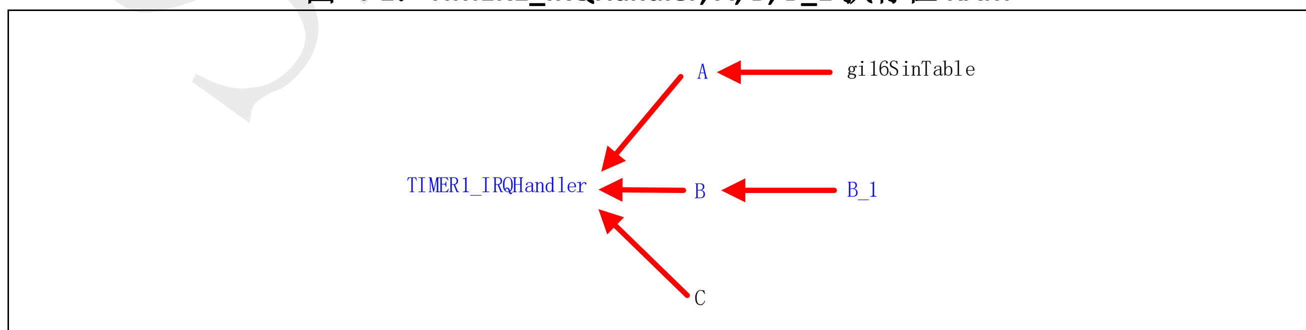
func.h

```
__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。

此时拓补结构如图 4-1 所示。

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



©2023 旋智电子科技（上海）有限公司

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：

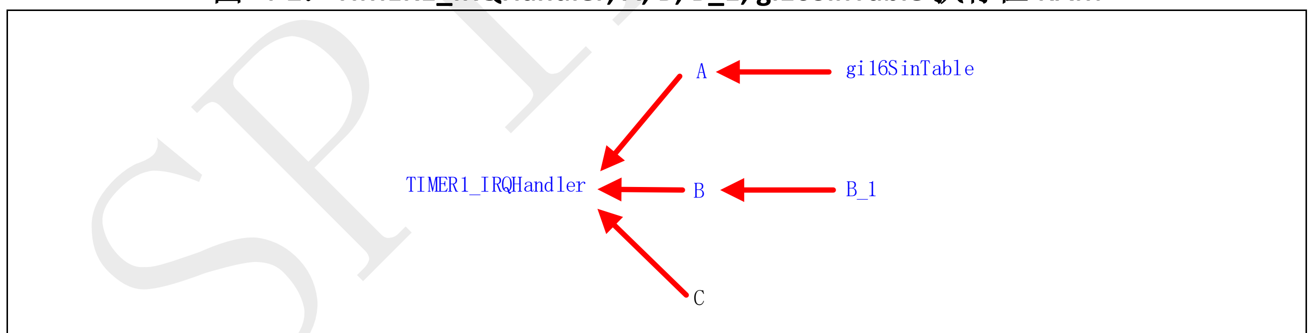
- const 变量会放置在只读存储区；
- static 变量会放置在静态存储区；

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
};
```

此时拓补结构如图 4-2 所示。

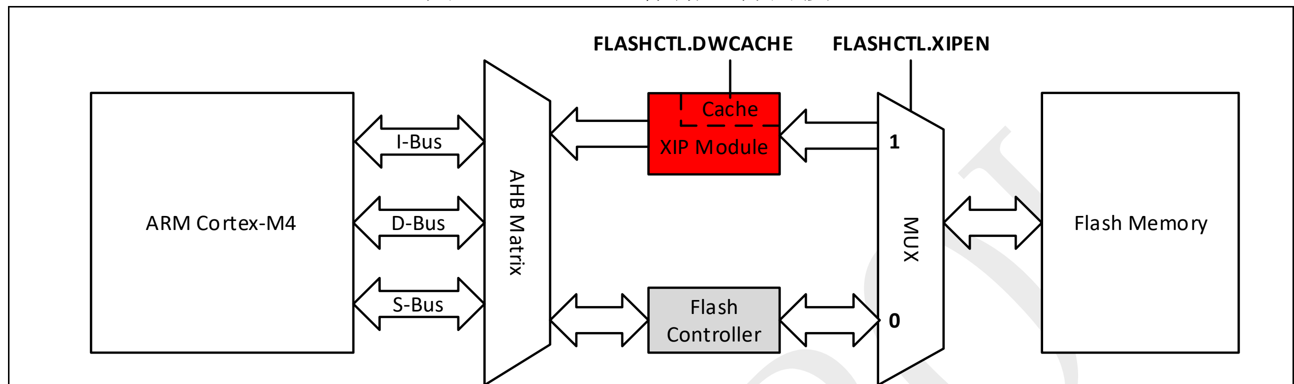
图 4-2：TIMER1_IRQHandler, A, B, B_1, gi16SinTable 执行在 RAM



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。

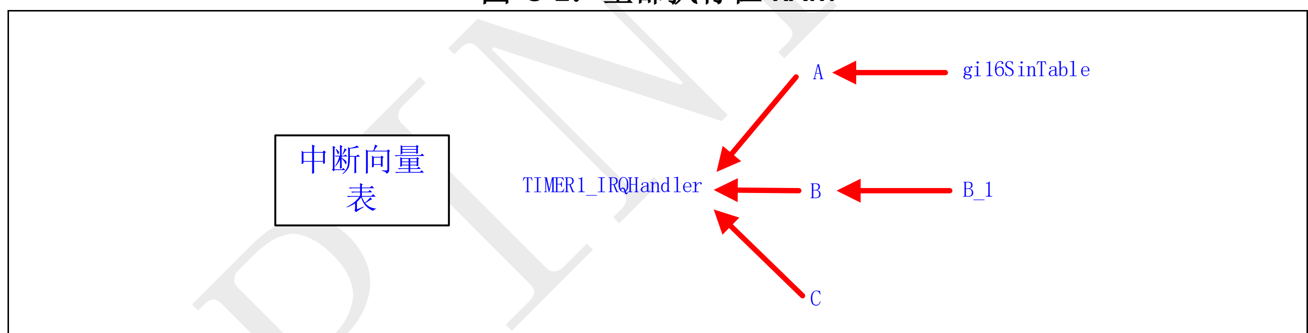
图 5-1：Flash 存储器访问接口



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、 gi16SinTable 都放到 RAM 的存储区。

此时拓补结构如图 5-2 所示。

图 5-2：全部执行在 RAM



©2023 旋智电子科技（上海）有限公司