KEIL 中如何让程序在 RAM 中运行

前言

最近老是遇到使用 KEIL 时需要将部分或者全部程序放到 RAM 中运行的问题。故此花了不少时间搜索资料和几番尝试，现将其总结在本篇文章中，也是为大家以后的工作节省时间罢。本文中会介绍通过 STM32F411Nucleo 的一个例子来介绍几种让程序在 RAM 中运行的方法。在该例子中，通过调用 ToggleLED 函数来翻转 LED2 亮灭。接下来，我们将通过多种方法将这段代码放在 RAM 中运行。

ToggleLED 函数从 Flash 中执行的情况

我们先来看看 ToggleLED 函数从 Flash 中执行的情况。下面是 ToggleLED 函数和它的调用情况。在 main 函数的 while（1）里调用 ToggleLED。

```
void ToggleLED(void)
{
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);

    /* Insert a 100ms delay */
    HAL_Delay(100);
}
```

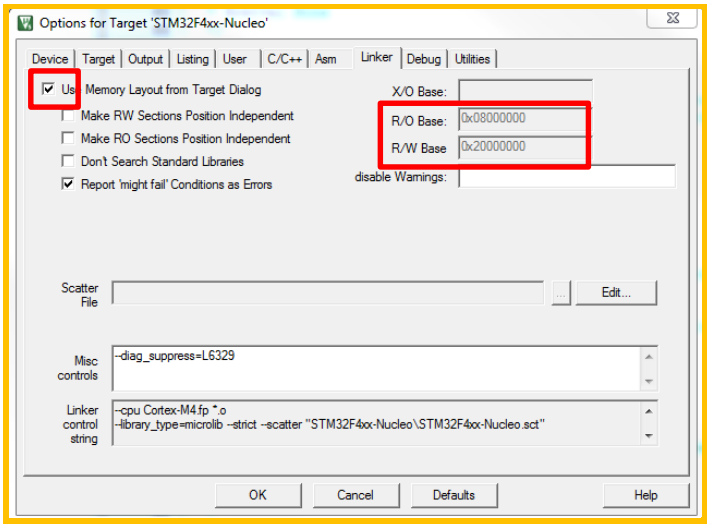
```
int main(void)
{
  ……

  /*##-3- Toggle PA05 IO in an infinite loop ##################################*/
  while (1)
  {
     ToggleLED();
  }
}
```

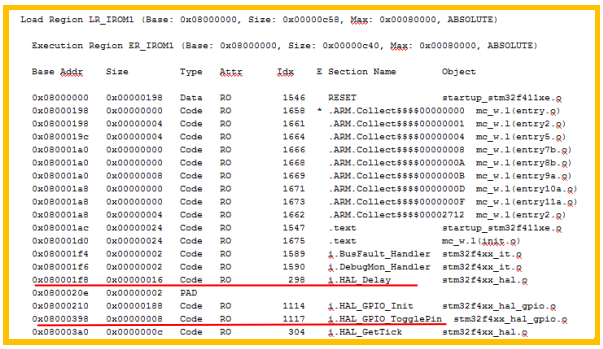
Linker 的配置为，见下图：

Flash 起始地址：0x08000000

RAM 起始地址：0x20000000



编译后从 map 文件可以看到，ToggleLED 以及其中调用到的 HAL_GPIO_TogglePin 和 HAL_Delay 函数的地址都在 FLASH 中。



```
Load Region LR_IROM1 (Base: 0x08000000, Size: 0x00000c58, Max: 0x00080000, ABSOLUTE)

  Execution Region ER_IROM1 (Base: 0x08000000, Size: 0x00000c40, Max: 0x00080000, ABSOLUTE)

  Base Addr    Size         Type   Attr      Idx    E Section Name        Object

  0x08000000   0x00000198   Data   RO         1546    RESET               startup_stm32f411xe.o
  0x08000198   0x00000000   Code   RO         1658  * .ARM.Collect$$$$00000000  mc_w.l(entry.o)
  0x08000198   0x00000004   Code   RO         1661    .ARM.Collect$$$$00000001  mc_w.l(entry2.o)
  0x0800019c   0x00000004   Code   RO         1664    .ARM.Collect$$$$00000004  mc_w.l(entry5.o)
  0x080001a0   0x00000000   Code   RO         1666    .ARM.Collect$$$$00000008  mc_w.l(entry7b.o)
  0x080001a0   0x00000000   Code   RO         1668    .ARM.Collect$$$$0000000A  mc_w.l(entry8b.o)
  0x080001a0   0x00000008   Code   RO         1669    .ARM.Collect$$$$0000000B  mc_w.l(entry9a.o)
  0x080001a8   0x00000000   Code   RO         1671    .ARM.Collect$$$$0000000D  mc_w.l(entry10a.o)
  0x080001a8   0x00000000   Code   RO         1673    .ARM.Collect$$$$0000000F  mc_w.l(entry11a.o)
  0x080001a8   0x00000004   Code   RO         1662    .ARM.Collect$$$$00002712  mc_w.l(entry2.o)
  0x080001ac   0x00000024   Code   RO         1547    .text               startup_stm32f411xe.o
  0x080001d0   0x00000024   Code   RO         1675    .text               mc_w.l(init.o)
  0x080001f4   0x00000002   Code   RO         1589    i.BusFault_Handler  stm32f4xx_it.o
  0x080001f6   0x00000002   Code   RO         1590    i.DebugMon_Handler  stm32f4xx_it.o
  0x080001f8   0x00000016   Code   RO          298    i.HAL_Delay         stm32f4xx_hal.o
  0x0800020e   0x00000002   PAD
  0x08000210   0x00000188   Code   RO         1114    i.HAL_GPIO_Init     stm32f4xx_hal_gpio.o
  0x08000398   0x00000008   Code   RO         1117    i.HAL_GPIO_TogglePin  stm32f4xx_hal_gpio.o
  0x080003a0   0x0000000c   Code   RO          304    i.HAL_GetTick       stm32f4xx_hal.o
```

```
  0x08000aba   0x00000002   PAD
  0x08000abc   0x00000080   Code   RO         1553    i.SystemClock_Config  main.o
  0x08000b3c   0x00000054   Code   RO          260    i.SystemInit          system_stm32f4xx.o
  0x08000b90   0x00000018   Code   RO         1554    i.ToggleLED           main.o
  0x08000ba8   0x00000002   Code   RO         1597    i.UsageFault_Handler  stm32f4xx_it.o
  0x08000baa   0x0000000e   Code   RO         1679    i.__scatterload_copy  mc_w.l(handlers.o)
  0x08000bb8   0x00000002   Code   RO         1680    i.__scatterload_null  mc_w.l(handlers.o)
  0x08000bba   0x0000000e   Code   RO         1681    i.__scatterload_zeroinit  mc_w.l(handlers.o)
  0x08000bc8   0x00000048   Code   RO         1555    i.main                main.o
```

将翻转 LED 的程序放到 SRAM 中执行

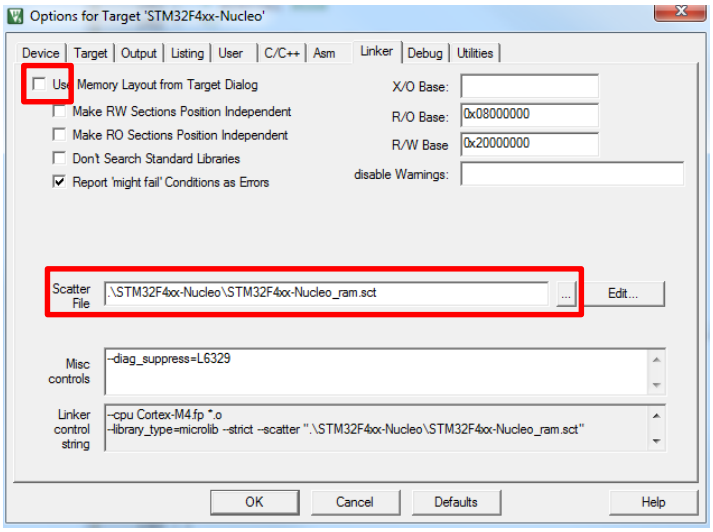
方法一：通过#pragma arm section code = “RAMCODE ”和#pragma arm section。参考 Example1 代码。

这种方式，可以同时将多个函数放到指定的 section。具体方法如下：

1. 修改.sct 文件，自定义一个叫做 RAMCODE 的 section，放在 RW_IRAM1 执行区域，地址范围 0x20000000~0x20020000。

```
LR_IROM1 0x08000000 0x00080000  {    ; load region size_region
  ER_IROM1 0x08000000 0x00080000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x20000000 0x00020000  {  ; RW data
  *.o(RAMCODE)
    .ANY (+RW +ZI)
  }
}
```

2. 在工程中使用前面修改的.sct 文件



3.以#pragma arm section code = “RAMCODE” 开头，以#pragma arm section 结尾。将所有需要放到 RAMCODE section 的函数包括进来。编译时，编译器会自动将这些函数放到 RAMCODE 所在 0x20000000 开始的区域。

```
#pragma arm section code = "RAMCODE"
void ToggleLED(void)
{
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);

    /* Insert a 100ms delay */
    HAL_Delay(100);
}

void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
  /* Check the parameters */
  assert_param(IS_GPIO_PIN(GPIO_Pin));

  GPIOx->ODR ^= GPIO_Pin;
}

uint32_t HAL_GetTick(void)
{
  return tick;
}

void HAL_Delay(__IO uint32_t Delay)
{
  uint32_t tickstart = 0;
  tickstart = HAL_GetTick();
  while((HAL_GetTick() - tickstart) < Delay)
  {
  }
}
#pragma arm section
```

4.从 map 文件里，可以看到这四个函数都已经被放到了 SRAM 中。

```
HAL_GetTick                             0x20000001   Thumb Code     6  main.o(RAMCODE)
HAL_Delay                               0x20000007   Thumb Code    16  main.o(RAMCODE)
HAL_GPIO_TogglePin                      0x20000017   Thumb Code     8  main.o(RAMCODE)
ToggleLED                               0x2000001f   Thumb Code    14  main.o(RAMCODE)
```

方法二：通过__attribute__((section(“name ”)))

在 KEIL 中可以通过__attribute__((at(address)))的方式将变量放到指定的位置。

通过__attribute__((section(“name ”)))的方式将变量或者函数放到指定的位置。

下面我们来看看如何通过这种方式将程序放到 SRAM 中执行。可以参考 Example2 的代码。

1.同样，我们需要修改.sct 文件，自定义一个叫做 RAMCODE 的 section，并在工程选项的 linker 页面中，选择定义好的.sct 文件。（见方法一中的第 1，2 步）

```
LR_IROM1 0x08000000 0x00080000  {    ; load region size_region
  ER_IROM1 0x08000000 0x00080000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x20000000 0x00020000  {  ; RW data
  *.o(RAMCODE)
    .ANY (+RW +ZI)
  }
}
```

2.在需要放到 RAM 中的函数前，用__attribute__((section("RAMCODE")))声明该函数放在 RAMCODE section 中。**注意，该函数中调用到的所有函数也要放到 RAMCODE section 中。**

```
__attribute__((section("RAMCODE")))
void ToggleLED(void)
{
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);

    /* Insert a 100ms delay */
    HAL_Delay(100);
}
```

```
__attribute__((section("RAMCODE")))
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
  /* Check the parameters */
  assert_param(IS_GPIO_PIN(GPIO_Pin));

  GPIOx->ODR ^= GPIO_Pin;
}
```

```
__attribute__((section("RAMCODE")))
__weak uint32_t HAL_GetTick(void)
{
  return uwTick;
}

__attribute__((section("RAMCODE")))
__weak void HAL_Delay(__IO uint32_t Delay)
{
  uint32_t tickstart = 0;
  tickstart = HAL_GetTick();
  while((HAL_GetTick() - tickstart) < Delay)
  {
  }
}
```

3.从编译后的 map 文件可以看出，ToggleLED 以及它调用到的所有函数都被到了 RAM 中。

```
  HAL_GetTick                              0x20000001   Thumb Code     6  stm32f4xx_hal.o(RAMCODE)
  HAL_Delay                                0x20000007   Thumb Code    24  stm32f4xx_hal.o(RAMCODE)
  HAL_GPIO_TogglePin                       0x20000025   Thumb Code     8  stm32f4xx_hal_gpio.o(RAMCODE)
  ToggleLED                                0x2000002d   Thumb Code    20  main.o(RAMCODE)
```

方法二可以覆盖方法一，也就是说如果你同时用方法一和方法二对同一个函数的执行区域做了说明。最终起作用的是方法二。我们可以通过 Example3 来说明

1.修改.sct 文件。将 SRAM 分为两个执行区 RW_IRAM1 和 RW_IRAM2。Section RAMCODE1，RAMCODE2 分别位于 0x20000000 开始，和 0x20010000 开始的两个 64KB 的区域。

```
LR_IROM1 0x08000000 0x00080000  {    ; load region size_region
  ER_IROM1 0x08000000 0x00080000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x20000000 0x00010000  {  ; RW data
  *.o(RAMCODE1)
    .ANY (+RW +ZI)
  }

  RW_IRAM2 0x20010000 0x00010000  {
  *.o(RAMCODE2)
  }

}
```

2.在代码中, HAL_GetTick 被放在了#pragma 的作用域内被声明放在 RAMCODE1 section，同时又用__attribute__((section ("RAMCODE2"))) 将其放在 RAMCODE2 的 section 内。

```
#pragma arm section code = "RAMCODE1"
void ToggleLED(void)
{
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);

    /* Insert a 100ms delay */
    HAL_Delay(100);
}

void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
  /* Check the parameters */
  assert_param(IS_GPIO_PIN(GPIO_Pin));

  GPIOx->ODR ^= GPIO_Pin;
}

__attribute__( ( section ( "RAMCODE2" ) ) )
uint32_t HAL_GetTick(void)
{
  return tick;
}


void HAL_Delay(__IO uint32_t Delay)
{
  uint32_t tickstart = 0;
  tickstart = HAL_GetTick();
  while((HAL_GetTick() - tickstart) < Delay)
  {
  }
}
#pragma arm section
```

3.编译完成后，我们再看看 map 文件中 HAL_GetTick 到底被放到了哪个 section。

```
HAL_Delay                       0x20000001   Thumb Code    16  main.o(RAMCODE1)
HAL_GPIO_TogglePin              0x20000011   Thumb Code     8  main.o(RAMCODE1)
ToggleLED                       0x20000019   Thumb Code    14  main.o(RAMCODE1)
SystemCoreClock                 0x20000030   Data           4  system_stm32f4xx.o(.data)
AHBPrescTable                   0x20000034   Data          16  system_stm32f4xx.o(.data)
tick                            0x20000048   Data           4  main.o(.data)
__initial_sp                    0x20000460   Data           0  startup_stm32f411xe.o(STACK)
HAL_GetTick                     0x20010001   Thumb Code     6  main.o(RAMCODE2)
```

从 map 里可以看到，最终 HAL_GetTick 被放在了 RAMCODE2 section 中。

将整个程序放到 SRAM 中执行

前面我们介绍了将一个或多个程序放到指定地址执行的方法。如果需要放到指定地址的程序比较多，我们还可以将这些需要放到指定地址的程序集中放到一个或几个 C 文件中，然后在.sct 文件中将这些 C 文件生成的目标文件放到指定地址。

在这里，我们将尝试将整个程序放到 SRAM 中执行。复位后程序从 FLASH 启动，之后将从 SRAM 执行所有的程序。下面是具体的步骤。可以参考 Example4 的代码。

1.将中断向量表和中断处理程序放到 SRAM 中

新建一个 startup_stm32f411xe_ram.s 文件，放到 0x20000000 开始的位置（在.sct 文件中修改）。注意这里是新建，而不是直接将原来的文件放到 SRAM 中，为什么呢？大家可以思考一下。在 startup_stm32f411xe_ram.s 里定义新的 SECTION，叫做 RESET_ram（还有其他的修改，请对照参考代码）。在后面的.sct 中将把 RESET_ram 这个 section 放到 SRAM 开始的位置上（见第 3 步）。

```
; Vector Table Mapped to Address 0 at Reset
                AREA    RESET_ram, DATA, READONLY
                EXPORT  __Vectors_ram
                EXPORT  __Vectors_End_ram
                EXPORT  __Vectors_Size_ram

__Vectors_ram  DCD     0                          ; Top of Stack
                DCD     0                          ; Reset Handler
                DCD     NMI_Handler                ; NMI Handler
……
```

2.在 SystemInit 中将中断向量表的偏移地址设置为 0x20000000。使能 VECT_TAB_SRAM。

```
#ifdef VECT_TAB_SRAM
  SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal SRAM */
#else
  SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal FLASH
*/
#endif
```

3.修改.sct 文件，将运行时需要的所有目标文件都放到 SRAM 执行区中。这里中断向量表有同样的两份，一份在 0x08000000 开始的位置，一份在 0x20000000 开始的位置。

```
LR_IROM1 0x08000000 0x00080000  {    ; load region size_region
  ER_IROM1 0x08000000 0x00080000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x20000000 0x00020000  {  ; RW data
  *.o (RESET_ram, +First)
  startup_stm32f411xe_ram.o(+RO)
  main.o(+RO +RW)
  stm32f4xx_it.o(+RO +RW)
  stm32f4xx_hal.o(+RO +RW)
  stm32f4xx_hal_gpio.o(+RO +RW)
```

```
  stm32f4xx_hal_rcc.o(+RO +RW)
  stm32f4xx_hal_cortex.o(+RO +RW)
    .ANY (+RW +ZI)
  }
}
```

4. 编译完成后，从 map 文件或者跟踪调试的结果都可以看到。系统复位以后，从 main 函数开始，所有的程序都在 RAM 中运行了。

另外，如果你的程序中有用到 ARM 底层的库，可以在.sct 文件中加入*armlib*(+RO)来将所有用到的库文件放到 SRAM 中。

重要通知 – 请仔细阅读

意法半导体公司及其子公司（“ST”）保留随时对ST 产品和/ 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于ST 产品的最新信息。ST 产品的销售依照订单确认时的相关ST 销售条款。

买方自行负责对ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的ST 产品如有不同于此处提供的信息的规定，将导致ST 针对该产品授予的任何保证失效。

ST 和ST 徽标是ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。