使用 STM32 设备的外部闪存编程

MDK 教程 AN333，2020 年秋季，v1.0

keil-feedback@arm.com

摘要

本应用笔记展示了一种通用的方法，用于通过 Keil MDK 对连接到 STM32 微控制器设备的外部闪存进行编程。

以使用STM32F769I-Discovery开发板为例，该开发板配备 STM32F769NIH6 微控制器和通过四线 SPI 连接的 MX25L51245G NOR 闪存。然而，所展示的概念可以类似地应用于其他 STM32 设备和闪存。

目录

摘要 1

前提条件 1

引言 2

在 Keil MDK 中编程 Flash 2

创建一个 Flash 编程算法 3

调试一个 Flash 编程算法 9

从外部闪存访问数据并执行代码 11

通过分散文件进行内存映射 15

程序外部闪存 16

摘要 17

参考文献和有用链接 17

先决条件

要复现本应用说明中描述的示例，需要以下组件：

Arm 公司的组件：

- Arm Keil MDK: 用于项目开发和调试的 IDE 和调试器。要创建闪存编程算法，需要 MDK-Essential 许可或更高版本。这里使用的是 MDK v5.31。
- ARM::CMSIS: CMSIS 包。版本 5.7.0 用于创建示例。
- Keil::STM32F7xx_DFP: STM32F7 设备的设备家族包（DFP）。除了其他项目外，还包含启动和系统文件以及示例应用程序。使用的是版本 2.13.0。

来自 ST 的组件：

- STM32F769I-Discovery 套件：示例中使用的目标硬件。
- STM32CubeMX：一个图形化工具，可轻松配置 STM32 微控制器。使用的是版本 6.0.1，包含 STM32F7 的 MCU 包 v 1.16.0。

示例项目

可在 keil.com/appnotes/docs/apnt_333.asp 下载 ZIP 文件。其中包含实现多个 STM32 Discovery 板闪存编程算法的项目，以及 Blinky 示例，展示如何在项目中使用它们。

引言

在大多数情况下，STM32 设备上的片上闪存足以运行用户应用程序。将程序加载到设备中是通过使用提供的闪存编程算法完成的。

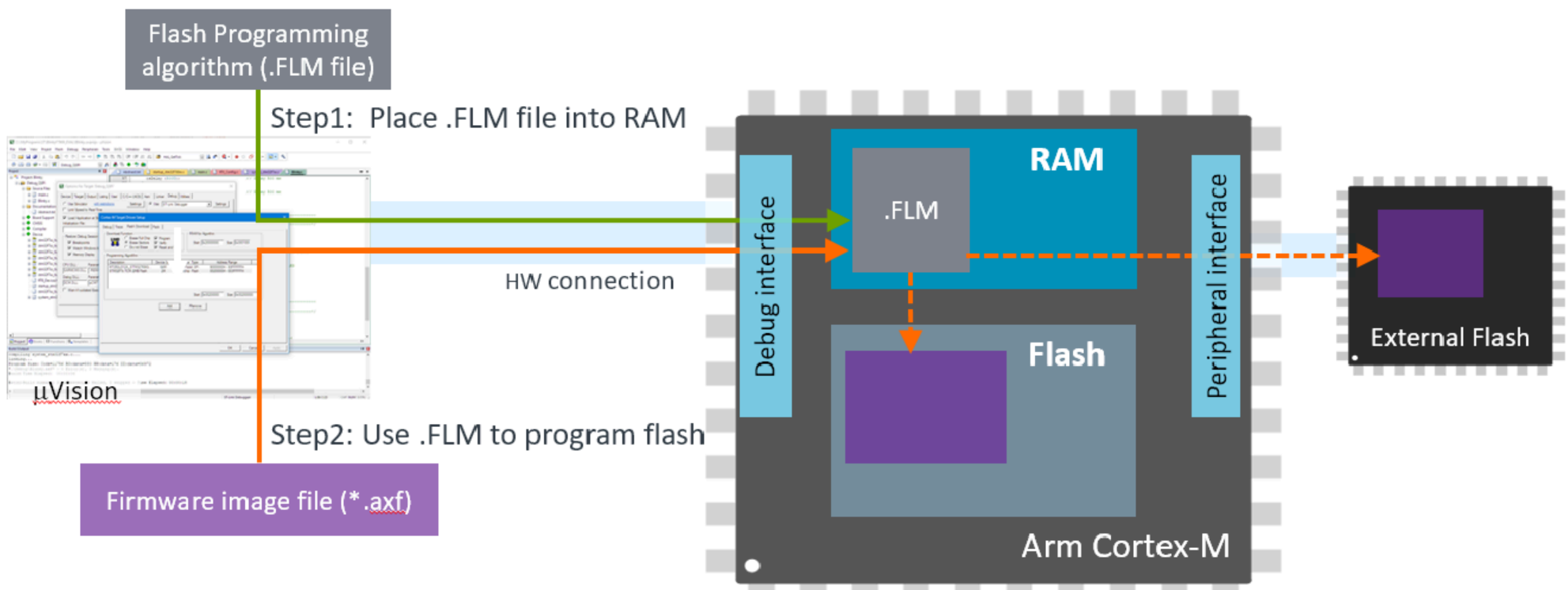
然而，当片上闪存的大小不足以满足需求时，可能需要使用外部闪存来存储常量数据或从中执行程序。例如，在带显示屏的嵌入式系统中，高质量图形内容需要大量内存，这些内容可以保存在外部闪存中。

该过程可以划分为几个不同的部分，这些部分在本文档的相关章节中有详细说明：

- 创建闪存编程算法：解释如何创建一个特殊程序，该程序可以将所需数据写入外部闪存设备。
- 调试闪存编程算法：提供设置项目的说明，该项目允许调试闪存编程算法。
- 从外部闪存访问数据并执行代码：给出一个使用外部闪存存储数据和程序代码的示例程序。
- 编程外部闪存：展示如何在 Keil MDK 中使用编程算法。

Keil MDK 中的闪存编程

为了对片上或外部闪存进行编程，Keil MDK 依赖于闪存编程算法——这是一种工具临时放置到 MCU 的 RAM 中的特殊软件，然后通过其接口向目标闪存提供数据并存储。Keil MDK 的闪存算法扩展名为 FLM。下图解释了 Keil MDK 中的闪存编程概念。



在µVision IDE 中编程设备时，首先将目标 ROM 区域的闪存编程算法（以 FLM 文件形式）放入微控制器的 RAM 中。之后，使用编程算法 API 将固件镜像文件写入相应的闪存——这可以是片上闪存，也可以是外部闪存设备。

应用笔记 334：MDK 闪存下载详细解释了闪存算法在 MDK 中如何用于擦除、下载和在闪存中验证应用程序。

对于目标微控制器，片上闪存的算法通常包含在相应的设备家族包（DFP）中，用户无需进行修改。STM32 设备也是这种情况。DFP 中提供的示例应用程序已经配置为使用这些算法进行内部闪存。

某些 STM32 DFP 还包含 FLM 文件，这些文件实现了针对特定开发板（如“Discovery”或“Eval”）上外部闪存设备的闪存编程算法。然而，提供的闪存算法文件将无法在具有其他设备变体和不同引脚排列的自定义 PCB 上工作。

对于此类自定义情况，用户需要按照下一章的说明创建闪存算法。

创建闪存编程算法

本章解释了创建自定义闪存编程算法的步骤，该算法能够将程序加载到连接到目标 STM32 微控制器的外部闪存设备。

注意：使用 MDK-Lite 创建闪存编程算法不受支持。

1. 从 ARM:CMSIS Pack 文件夹（默认情况下可用）复制_Template_Flash\文件夹。
   将 C:\Users\\Pack\ARM\CMSIS\\Device_Template_Flash)复制到一个新文件夹。这是一个用于闪存编程算法的模板项目。它被用作实现目标系统闪存编程算法的基础。

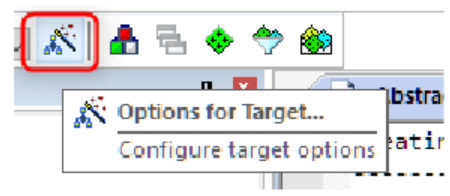
2. 移除复制目录的只读保护，包括其中所有的子目录和文件。

3. 将文件夹重命名为反映其用途。在我们的示例中，它被重命名为STM32F769I_Discovery_QPSI_MX25L51245G。

4. 将项目文件 NewDevice.uvprojx 重命名为代表新的闪存设备名称。在我们的示例中，它被重命名为 STM32F769I_Discovery_QPSI_MX25L51245G.uvprojx。

5. 使用µVision 打开项目文件。

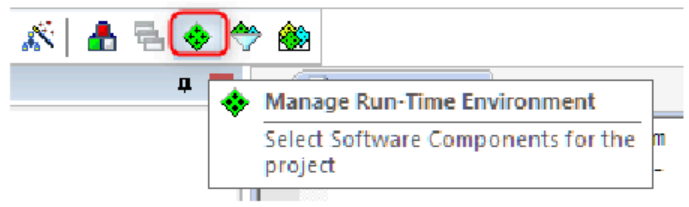
6. 转到项目 – 目标选项（Alt+F7）：



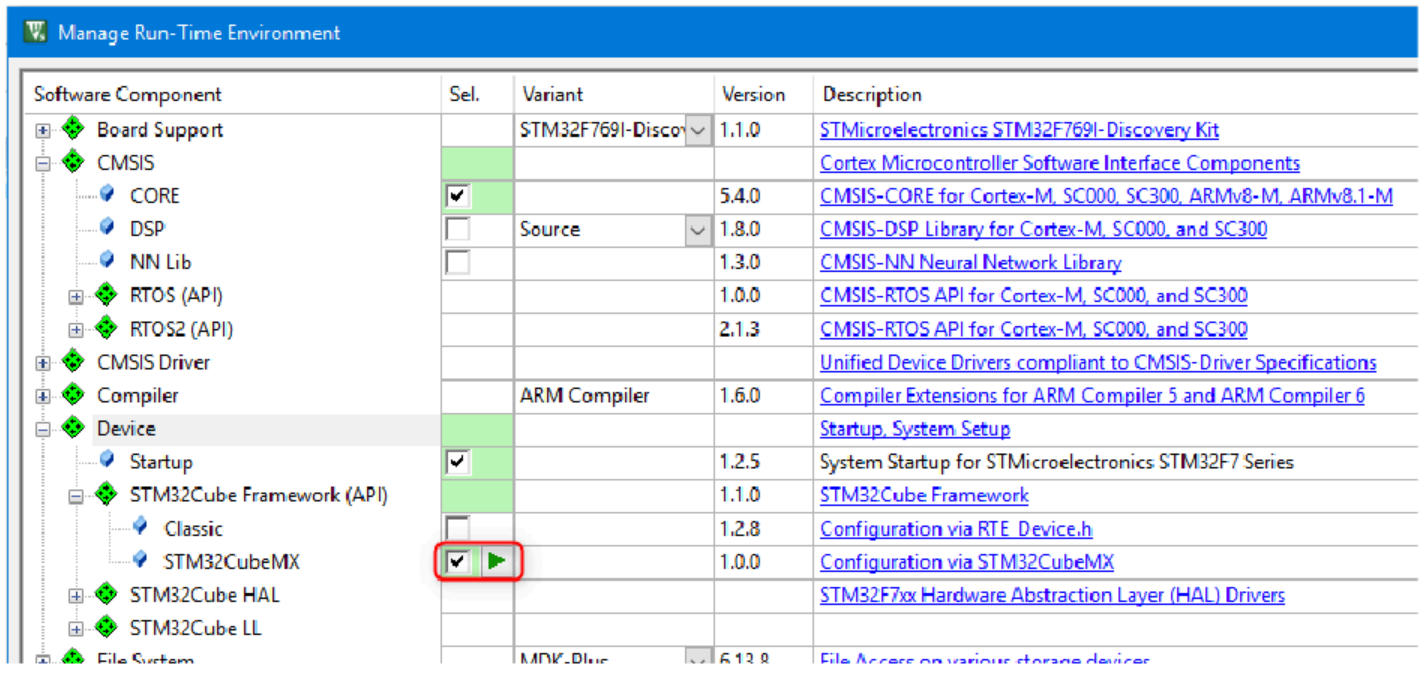
   - 在目标选项卡中，选择目标微控制器设备。在我们的示例中，它是 STM32F769NIHx。

     注意：该对话框仅显示已安装的 DFP 支持的设备。如果目标设备不可用，您需要使用包安装器工具安装其 DFP。

   - 确认在代码生成区域中，ARM 编译器字段选择使用默认编译器版本 6。
   - 在输出选项卡上：
     - 将可执行文件名称字段的内容更改为表示目标设备和闪存。名称不应超过 31 个字符，否则 µVision IDE 将无法检测到该文件。在我们的示例中，它被设置为 STM32F769I_Disco_MX25L51245G。
     - 确认调试信息标志已设置。这是确保 µVision 可以从生成的 FLM 文件中调用单个函数所必需的。
   - 按下 OK。



7. 打开管理运行时环境窗口：
   - 启用 CMSIS:CORE 组件。
   - 启用 Device:Startup 组件。
   - 启用设备：STM32Cube 框架(API)：STM32CubeMX 组件。
   - 通过点击其旁边的按钮启动 STM32CubeMX。



8. 在 STM32CubeMX 中，按照以下步骤配置 MCU 与外部闪存的连接并生成相应代码：

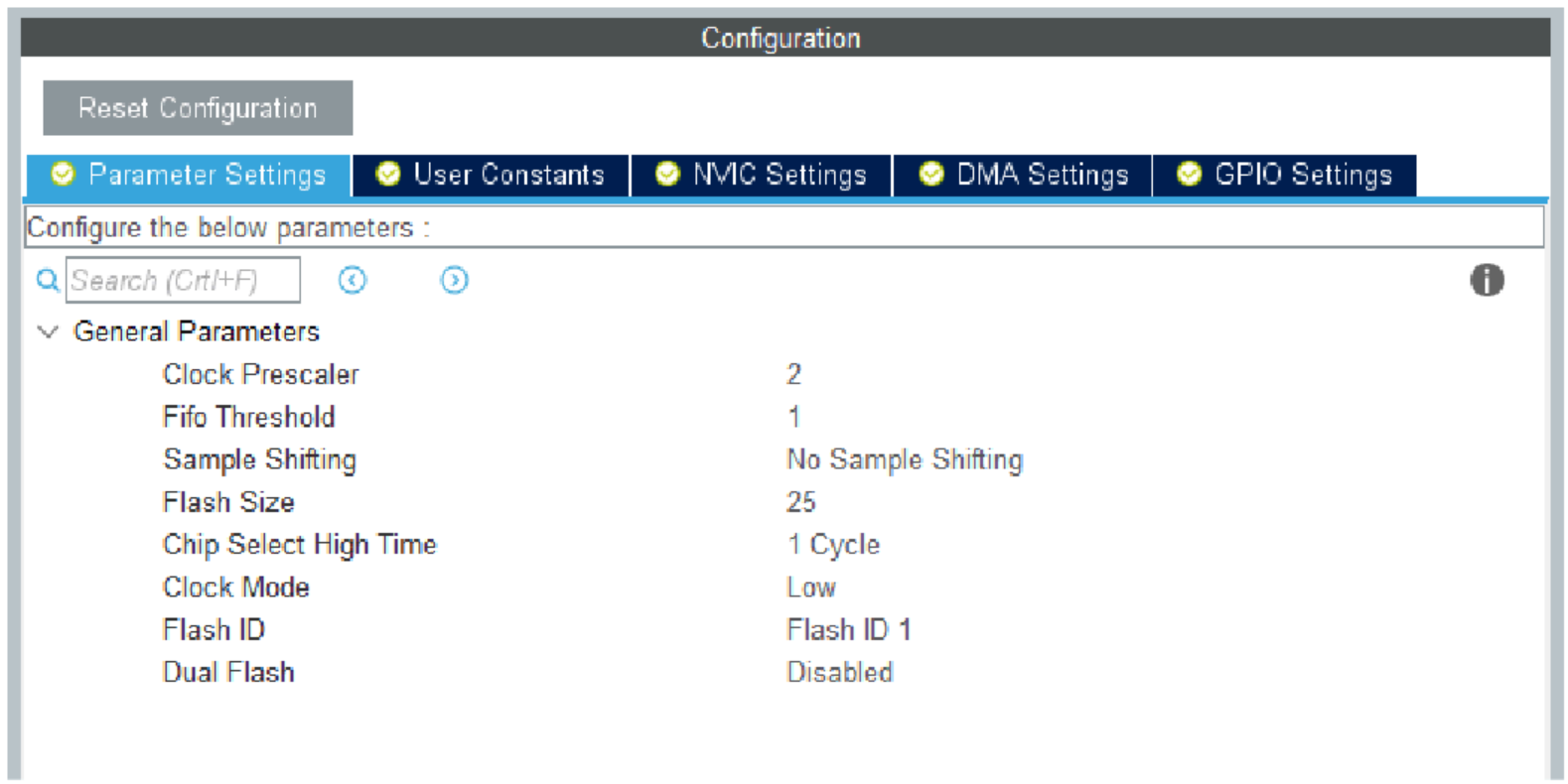
- 在引脚配置与参数设置选项卡中：

o 在分类视图展开连接性类别并选择 QUADSPI，因为这是我们示例中板载使用的接口。这将打开 QUADSPI 模式和配置

o 选择正确的 QuadSPI 模式（在我们的示例中为 Bank1，使用 Quad SPI 线路）

o 在配置部分——在参数设置选项卡中，根据目标设置设置正确的参数

在我们的示例中，这些参数如下：

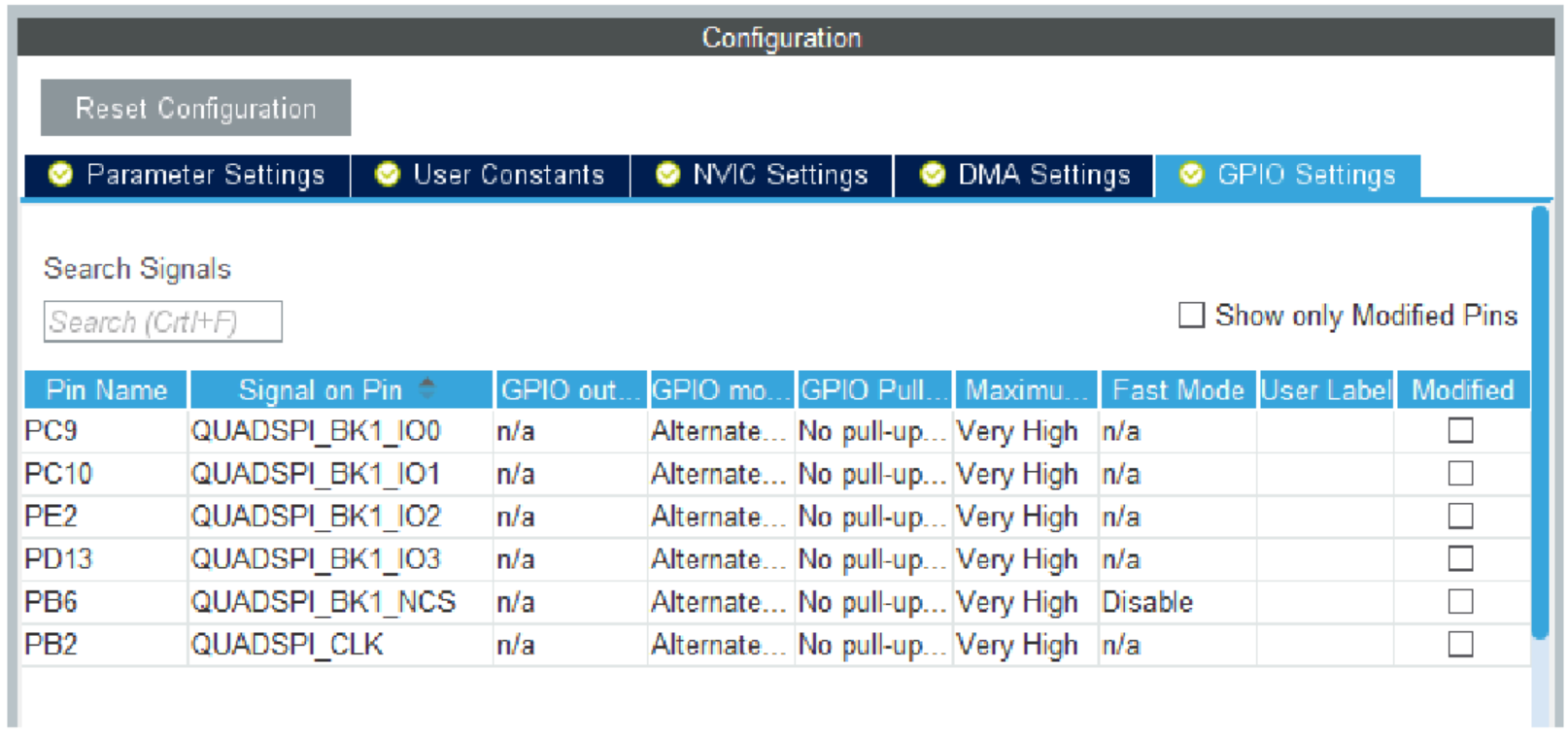


注意：请查阅设备内存数据手册以获取正确的 QSPI 参数设置。

o 在引脚视图：选择外部闪存连接的正确引脚。

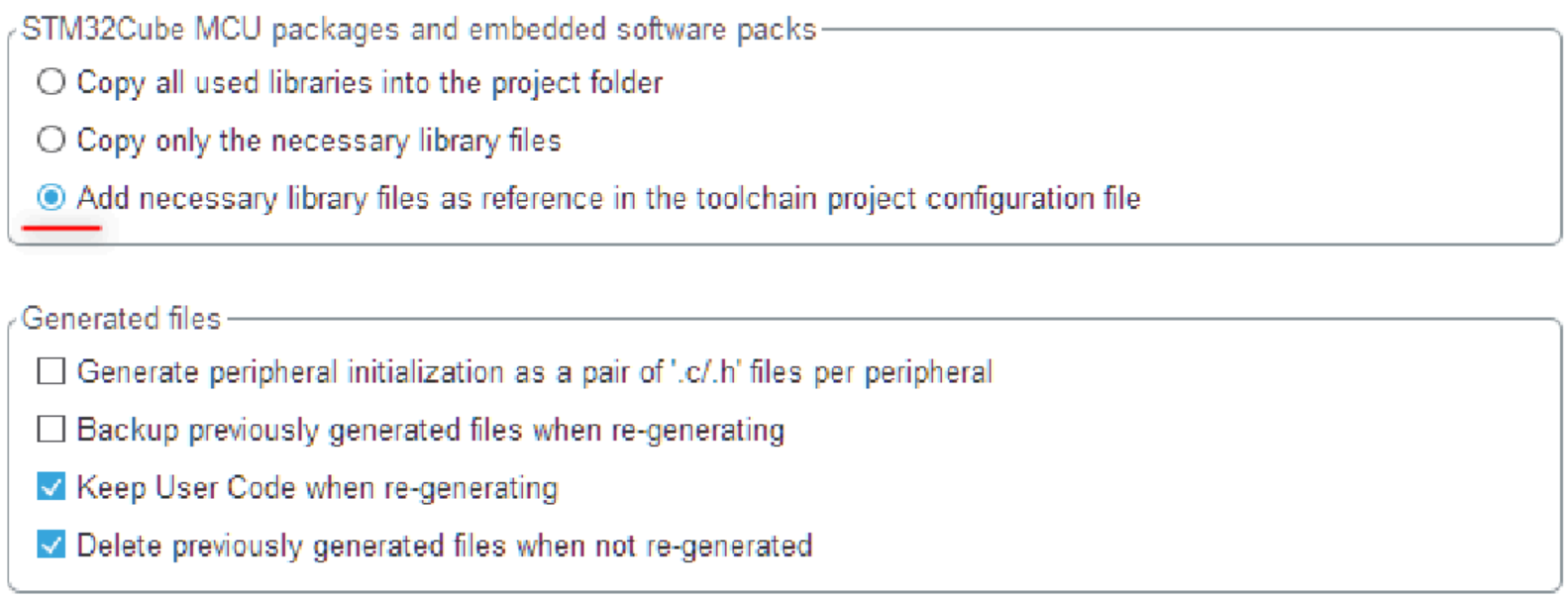
引脚配置也可以在配置部分的 GPIO 设置选项卡中观察到。

根据我们示例的STM32F769I-Discovery原理图，引脚需要按照以下方式重新配置：

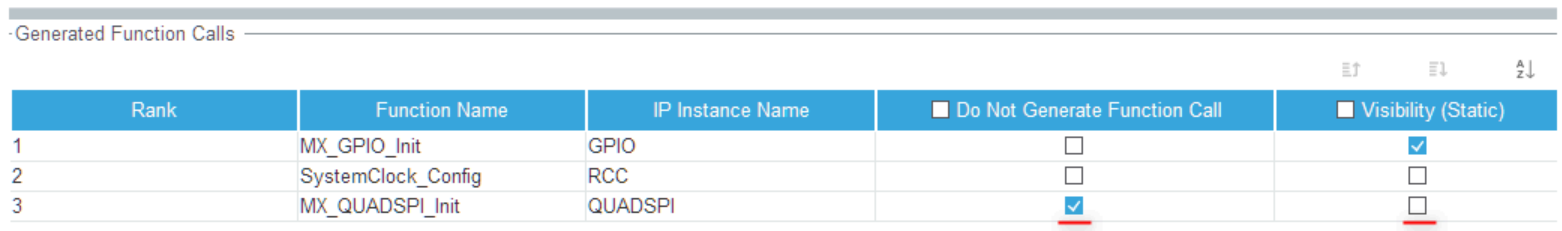


- 在时钟配置选项卡：无需更改默认设置。
- 在项目管理选项卡修改默认项目选项：
  - o 在项目类别中选择不生成 main()。

  - o 在代码生成类别中勾选在工具链项目配置文件中作为参考添加必要的库文件。



  - o 在 QUADSPI 驱动程序的"高级设置"类别中：
    - 启用"不生成函数调用"
    - 禁用可见性（静态）



- 点击"生成代码"按钮。
  - o 如有需要，下载目标 MCU 的软件包。
  - o 代码生成完成后，在代码生成对话框中点击关闭。

9. 切换回µVision 项目。
   - 在管理运行时环境窗口中点击确定。

- 在询问是否导入基于 STM32CubeMX 生成的代码的项目更改的对话框中点击"是"。

10. 在 FlashPrg.c 文件中实现闪存编程算法函数。

    文件组"程序函数"包含 FlashPrg.c 文件，其中需要实现 Keil MDK 中闪存编程部分所解释的高级闪存操作函数。

    该实现对于 STM32 设备相当通用，但针对所使用的通信接口（例如 QuadSPI 或 OctoSPI）具有特定性。它也独立于外部闪存设备。
    在应用笔记的 ZIP 文件中，您可以找到包含 FlashPrg.c 文件的项目 STM32F769I_Discovery_QPSI_MX25L51245G，该文件实现了我们目标系统的算法函数。只需使用这个文件代替默认模板文件。

11. 修改 FlashDev.c 文件中的设备参数。

    该文件指定了目标闪存设备的参数，如页大小、扇区、总大小等。在我们的示例中，它需要包含以下内容：

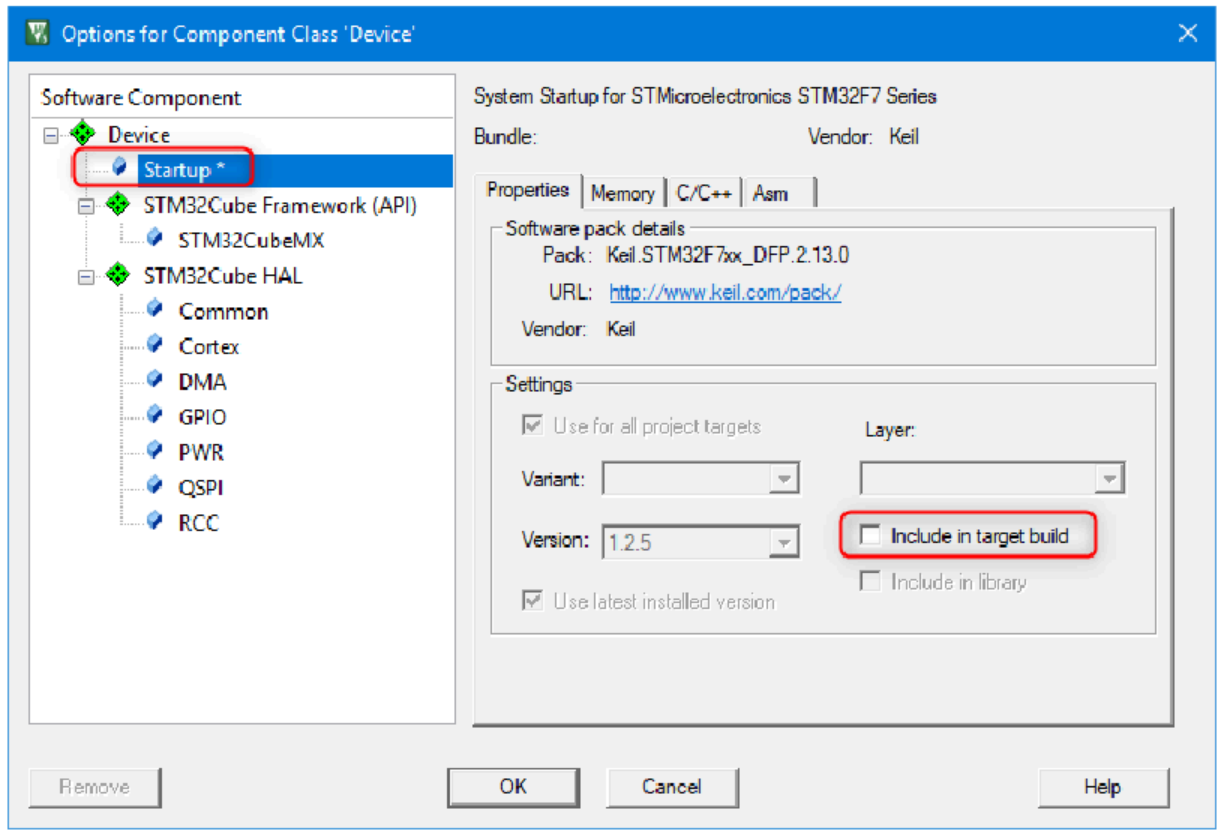
```
#include "FlashOS.h"                    // FlashOS Structures

struct FlashDevice const FlashDevice = { FLASH_DRV_VERS, // 驱动版本，不要修改!
"STM32F769I_Disco_MX25L51245G", // 设备名称 EXTSPI, // 设备类型 0x90000000, // 设备起始地址
0x04000000, // 设备字节数大小 (64MB) 0x00001000, // 编程页大小：4Kb (16*页大小) 0x00, // 保留，必须为
0 0xFF, // 擦除后内存的初始内容 10000, // 编程页超时 1000 mSec 10000, // 擦除扇区超时 1000 mSec



// 指定扇区的大小和地址 0x1000, 0x000000, // 扇区大小 4kB，扇区编号：16383 SECTOR_END };
```

12. 从构建中移除启动文件，并替换系统文件。

    我们需要从我们的闪存编程算法中排除启动文件，以确保没有向量表被定义，并且当加载时算法程序不会自动运行。我们还需要重新添加系统文件。

    在µVision 项目窗口中：

    - 在设备组上右键单击，然后选择“设备组件类”的“选项”。在打开的对话框中选择启动组件，取消勾选“包含在目标构建中”。点击“确定”。



- 在上一步中，我们还移除了构建所需的系统文件。因此，我们需要将其重新添加回来。

  o 右键点击目标文件夹（我们这里以 Cortex-M 为例），选择添加组。项目会添加一个名为 New Group 的组。双击它并重命名为 System File。

  在 System File 组上右键单击，并选择将现有文件添加到“System File”组中
  将项目中的“./RTE/Device//”目录下的系统文件添加到其中。在我们的示例中，它是“./RTE/Device/STM32F769NIHx/system_stm32f7xx.c”文件。

13. 添加内存连接接口的驱动。

    意法半导体在 GitHub 仓库 https://github.com/STMicroelectronics/stm32-external-loader/tree/contrib 中提供了多种外部四 SPI 闪存设备的驱动程序。

    我们示例的 STM32F769I-Discovery 开发板搭载 MX25L51245G 闪存设备，因此我们使用来自仓库的相应 QSPI 驱动程序。

    - 在我们的 µVision 项目根目录下创建一个以闪存设备名称命名的文件夹（例如 MX25L51245G)并将 QSPI 驱动程序存储库中的 quadspi.c 和 quadspi.h 文件复制到其中。
    - 在µVision 的项目窗口中，右键单击目标文件夹（在我们的例子中是 Cortex-M），然后选择添加组。项目会添加一个名为 New Group 的组。将其重命名为 QUADSPI Memory。
    - 右键单击 QUADSPI Memory 组，选择添加现有文件到组“QUADSPI Memory”…然后将“.\QUADSPI Memory”目录中的现有 quadspi.c 文件添加到其中。
    - 转到项目 – 目标选项（Alt+F7） – C/C++(AC6)选项卡，并在包含路径字段中添加包含内存连接驱动程序的文件夹的路径。在我们的例子中它是“.\\MX25L51245G”。

14. 编辑 quadspi.c 和 quadspi.h 文件。

    默认的 QSPI 驱动文件需要稍作修改以确保正常使用。

    - 修改 quadspi.c 并在顶部添加：

      ```
      #include "quadspi.h" extern
      QSPI_HandleTypeDef hqspi;
      ```

    - 修改 quadspi.h：

      o 在顶部添加：#include "main.h"

    如果你需要再次确认，请参考应用笔记 ZIP 文件中的示例项目。

15. 在 main.c 中添加 HAL Tick 函数：

    HAL Tick 函数的默认实现依赖于 SysTick。然而，在闪存编程期间，我们需要确保不会发生中断，并避免不必要的额外外设初始化。

    因此，HAL Tick 函数的新实现需要放置在标记为/* USER CODE BEGIN 0 */和/* USER CODE END 0 */之间的部分。这将覆盖 HAL 中的弱实现。

    - 添加 HAL_InitTick 函数：

```
/** * 覆盖默认 HAL_InitTick 函数 */ HAL_StatusTypeDef HAL_InitTick(uint32_t
TickPriority) { return HAL_OK; }
```

    - 添加 HAL_GetTick 函数：

```
/** * 重写默认的 HAL_GetTick 函数 */ uint32_t
HAL_GetTick (void) { static uint32_t ticks = 0U;

            uint32_t i;

/* 如果内核未运行，则等待大约 1 毫秒，然后增加并返回辅助滴答计数器值 */ for (i =
(SystemCoreClock >> 14U); i > 0U; i--) { __NOP(); __NOP(); __NOP(); __NOP(); __NOP();
__NOP(); __NOP(); __NOP(); __NOP(); __NOP(); __NOP(); } return ++ticks; }
```

    - 添加 HAL_Delay 函数：

```
/** * 覆盖默认的 HAL_InitTick 函数 */ void
HAL_Delay(uint32_t Delay) { uint32_t tickstart =
HAL_GetTick(); uint32_t wait = Delay;

  /* 添加一个周期以保证最小等待时间 */ if (wait <
  HAL_MAX_DELAY) { wait += (uint32_t)(HAL_TICK_FREQ_DEFAULT);
  }

while((HAL_GetTick() - tickstart) < wait) { __NOP(); } }
```

16. 编辑 main.h

    添加 SystemClock_Config() 和 MX_QUADSPI_Init() 函数的定义：

```
/* USER CODE BEGIN EFP */ void
SystemClock_Config(void); void
MX_QUADSPI_Init (void); /* USER CODE END
EFP */
```

17. 使用  项目 – 构建目标 (F7)”来生成新的闪存编程算法。用户特定的后构建过程会在项目根目录中创建 FLM 文件。在我们的示例中，它是STM32F769I_Disco_MX25L51245G.FLM

18. 输出 FLM 文件需要复制到 MDK 安装目录下的./ARM/Flash 目录中。这样算法就可以在项目中使用了。

本章创建的闪存算法可用于将固件镜像放置到外部闪存中
在“程序外部闪存”章节中解释。在“访问外部数据并执行代码”章节中
flash 展示了如何在应用程序项目中使用外部闪存来存储数据和代码。如果算法不工作，请参考第 Debug a flash programming algorithm 章节，该章节解释了如何设置一个用于调试目的的测试项目。

Debug a flash programming algorithm

前一章的项目实现了闪存编程算法，但是，如果出现任何问题，它不允许调试代码。本章解释了如何创建一个单独的测试项目，该项目可用于调试目的。

在我们的示例中，我们将所有代码和数据放在 RAM 中，并启动调试会话。

1. 将章节“创建闪存编程算法”中创建的示例项目复制到单独的目录中。这将作为具有调试功能的测试项目的基础。

2. 确保测试程序从 RAM 执行：

   转到项  目 – 目标选项（Alt+F7）：

   - 在 C/C++（AC6）选项卡中，取消勾选“只读位置独立”和“读写位置独立”选项。

   - 在链接器选项卡中，点击编辑按钮并修改链接器脚本，以便所有代码和数据都放置在 RAM 中。在我们的示例中，这是通过以下方式完成的：

```
; 链接器控制文件（分散加载）；


LR_ROM 0x20000000 0x0000F000 { ; 加载区域大小_region ER_ROM 0x20000000 0x0000F000 { ; 加载地址
= 执行地址
      *.o (RESET, +First)
      *(InRoot$$Sections)
      * (+RO +XO)
   }

RW_IRAM1 0x2000F000 0x00002000 { ; RW data .ANY (+RW +ZI) }
}
```

请参考链接器用户指南中的“散文件语法”章节，了解更多关于散文件的信息。

- 在调试选项卡中：
  - o 在使用行从下拉菜单中选择目标调试适配器。在我们的案例中这是 ST-LINK。
  - o 取消勾选启动时加载应用程序标志。
  - o 添加一个名为 Dbg_RAM.ini 的初始化文件，内容如下：

```
/*-------------------------------------------------------------Setup() 配置 PC 和 SP 用于 RAM 调
试 *-------------------------------------------------------------*/ FUNC void Setup (void) { SP =
_RDWORD(0x20020000); // 设置堆栈指针 PC = _RDWORD(0x20020004); // 设置程序计数器
_WDWORD(0xE000ED08, 0x20020000); // 设置向量表偏移寄存器 }



/*-------------------------------------------------------------OnResetExec() 重置后配置
PC 和 SP 用于 RAM 调试 *-------------------------------------------------------------*/
FUNC void OnResetExec (void) { Setup(); }


LOAD %L 增量加载                    // 加载应用程序
Setup();                            // 运行设置
//g, main
```

  - 在工具选项卡中取消勾选“调试前更新目标”选项。
  - 按下 OK。

3. 重新启用设备家族包中的启动代码。
   这会撤销第 12 步，来自章节“创建闪存编程算法”。在项目窗口中：
   - 排除 STM32CubeMX 提供的系统文件：o 找到系统文件组。
     - o 右键点击位于该位置的系统文件（以我们的示例 system_stm32f7xx.c 为例） o 选择“system_stm32f7xx.c”文件的“选项” o 在对话框窗口中取消勾选“包含在目标构建中” o 点击“确定”。
   - 从 DFP 中包含设备启动：o 在设备组中找到系统文件（在我们的例子中是 system_stm32f7xx.c）。
     - o 右键点击它并选择“组件类‘Device’的选项..
     - o 确保在左侧的软件组件部分中选择了 Startup。启用包含在目标构建中。
     - o 按下 OK。

4. 创建一个包含测试代码的文件。
   - 在项目窗口中，右键点击目标文件夹（例如 Cortex-M），然后选择添加组。
     项目会添加一个名为 New Group 的组。将其重命名为 Test Code。
   - 在“Test Code”组上右键单击，选择“添加新项到‘Test Code’组”。在对话框中选择“C 文件(.c)”选项，并指定文件名，例如 FlashTest.c。单击“添加”按钮并关闭对话框。

- 在 FlashTest.c 文件中填充一个 main()函数，用于测试 FlashPrg.c 文件中的 API 函数。它应该实现 CMSIS-Pack 文档（算法函数部分）和应用程序说明 334：MDK 闪存下载中描述的Vision 中使用的流程。例如，测试闪存擦除操作：

```
#include "RTE_Components.h" #include
CMSIS_device_header #include
"FlashOS.h"

extern struct FlashDevice const FlashDevice;

volatile int ret;                          // 返回码

/* 错误处理函数 */ void stop_on_error(uint32_t
cond) { if(cond) {


__BKPT(0x1); // 执行过程中发生错误 while(1){} } }



/*------------------------------主函数 *----------
----------------------*/ int main(void) { /* 测试
Flash 擦除操作 */ ret = Init(FlashDevice.DevAdr, 0,
1); stop_on_error (ret);


   ret = EraseChip();
   stop_on_error (ret);

   ret = UnInit(1);
   stop_on_error(ret);

   while(1){}
}
```

这里 main()函数初始化外部存储器，执行完整内存擦除操作，然后取消与闪存设备的连接。如果闪存编程函数执行失败，程序将在 stop_on_error()中的断点处停止。

闪存编程和闪存验证流程可以类似地实现，并在返回代码不成功时进行调试。

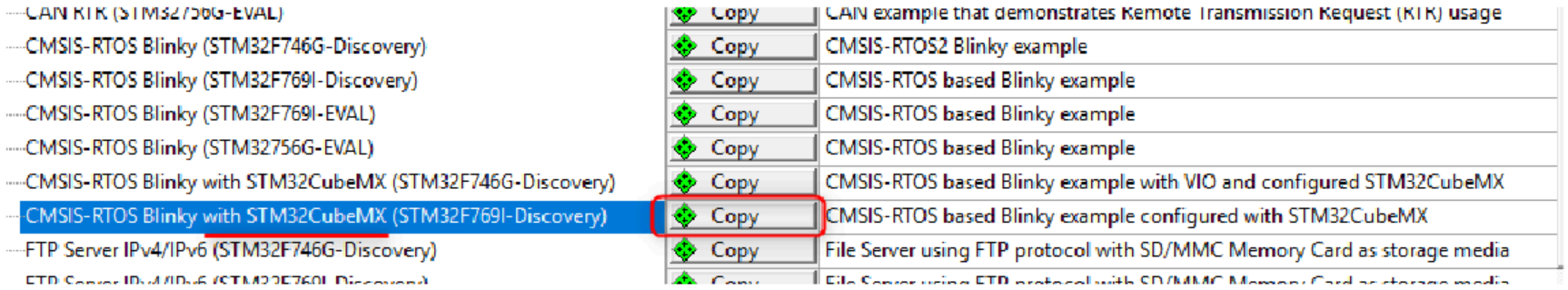
从外部闪存访问数据并执行代码

本章展示了如何扩展现有应用程序，以便将数据常量或程序代码存储在外部闪存中。

以下所有步骤都是通用的，可以应用于任何项目。在我们的示例中，我们从 CMSISRTOS2 Blinky 与 STM32CubeMX 项目开始，该项目包含在 STM32F7 系列设备家族包中的STM32F769I-Discovery板上。

应用说明的 ZIP 存档包含使用外部闪存的修改后的 Blinky 示例。

1. 使用 Pack Installer 复制STM32F769IDiscovery板的 CMSIS-RTOS2 Blinky with STM32CubeMX 项目。



注意，对于 STM32F2/F4/F7 系列，您需要复制名称中包含“with STM32CubeMX”的 Blinky 示例。只有这个项目默认可通过 STM32CubeMX 进行配置。对于其他系列，标准的 CMSIS-RTOS Blinky 示例支持 STM32CubeMX，并且可以按照本章所述流程扩展以使用外部闪存。

2. 使用 STM32CubeMX 配置 QuadSPI 接口。

   为了能够使用外部闪存，我们需要使用 STM32CubeMX 工具在项目中添加和配置 QSPI 驱动接口。

   目标硬件与用于闪存算法的硬件相同，因此只需重复“创建闪存编程算法”章节中的步骤 7、8 和 9。

   注意，在这种情况下，时钟设置以及应用程序使用的其他外设也可以在 STM32CubeMX 中配置。

3. 将 Quad-SPI 驱动文件添加到项目中。

   quadspi.c 和 quadspi.h 文件可以从闪存算法项目中完全复用，如下所示：

   - 将 flash algorithm 项目中的.\MX25L51245G-directory复制到 Blinky 项目的根目录下。

   - 在项目窗口中，右键单击目标文件夹并选择添加组。项目会添加一个新组。将其重命名为 QUADSPI Memory。
   - 右键单击 QUADSPI Memory 组，选择添加现有文件到组“QUADSPI Memory”…，并将.\MX25L51245G 目录中的现有 quadspi.c 文件添加到其中。
   - 转到项目 – 目标选项（Alt+F7） – C/C++(AC6)选项卡，并在包含路径字段中添加包含内存连接驱动程序的文件夹的路径。在我们的例子中它是“.\\MX25L51245G”。

4. 修改 main.h 文件：

   添加 MX_QUADSPI_Init()函数的定义：

```
/* 用户代码开始 EFP */ void
MX_QUADSPI_Init (void); /* 用户代码结束
EFP */
```

5. 修改 main.c 文件：

   - 在用户包含目录下添加 #include "quadspi.h"：

```
  …
/* 用户代码开始 Includes */ #include
"quadspi.h" /* 用户代码结束 Includes */
```

   - 在 main()中启用内存映射模式：

     为了无缝访问位于外部闪存中的数据和执行代码，需要启用 QUADSPI 的内存映射模式。

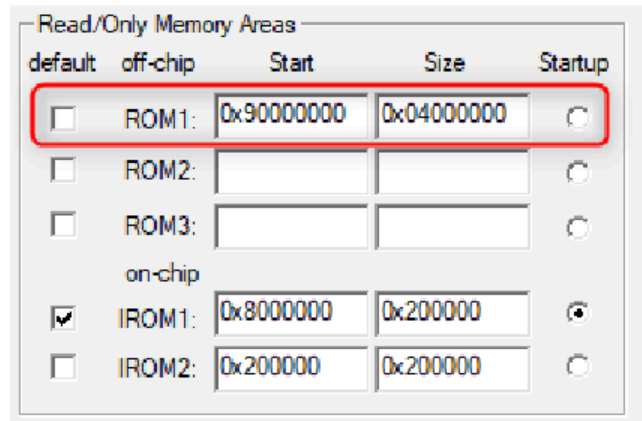
为此，在/* USER CODE BEGIN 2 */部分中的 main()函数中添加对相应 CSP 函数的调用，如下所示：

```
/* USER CODE BEGIN 2 */ CSP_QUADSPI_Init();
CSP_QSPI_EnableMemoryMappedMode();

...
```

6. 扩展内存布局。

我们需要定义与系统中使用的外部闪存对应的内存区域，以便所需数据或代码可以链接到其中。

进入 Project – Options for Target (Alt+F7) – Target 选项卡。在只读内存区域，使用 Start 和 Size 字段指定一个额外的 ROM 区域，该区域对应于闪存编程算法中使用的 FlashDev.c 文件中指定的外部闪存。在我们的示例中，我们使用 ROM1 来指定外部闪存区域。



默认复选框定义默认使用的内存。
启动单选按钮选择用于启动代码的区域。

两者都必须启用，以确保内部 ROM 能够启动并初始化 QSPI 接口。

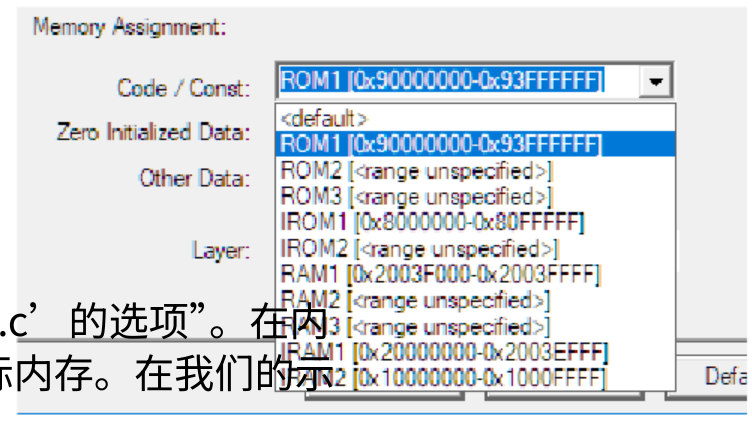
7. 向外部闪存添加要放置的常量数据。

- 在项目窗口中，右键单击源组，然后选择“添加新项到组”“源”…
- 选择 C 文件（.c），提供一个文件名（例如 Data.c），然后点击添加。
- 将以下内容添加到 Source 组中出现的空 Data.c 文件中：

```
/* 延迟常量 */ const unsigned int delay_data[10]={100,200,300,400,500,600,700,800,900,1000};
```

8. 将目标文件映射到外部闪存。

在µVision IDE 中，可以指定软件组件、文件组和单个文件的目标内存。
在我们的示例中，我们希望将 Data.c 文件放置在外部闪存上。



在项目窗口中，右键单击 Data.c 文件，然后选择“文件‘Data.c’的选项”。在内存分配区域的对话框中，从下拉菜单中选择 Code/Const 的目标内存。在我们的示例中，它应该是 ROM1。单击“确定”。

类似的方法可用于将程序代码放置到外部闪存中（例如 Blinky.c 文件）。

9. 或者，可以使用散列文件进行内存映射。参见章节"通过散列文件进行内存映射"。

注意，Quad-SPI 驱动程序以及启动代码（直到内存映射模式初始化之前）都需要从内部闪存中启动。

10. 更新 Blinky.c 以使用 delay_data。

    我们修改了默认逻辑，使得当摇杆被按下时，从外部闪存中存储的 delay_data 数组中的下一个值被用作 LED 闪烁的间隔。

    该示例使用了一些特定于板卡的 peripherals，例如 LED 和摇杆。

```
...
extern const uint32_t delay_data[10]; // 延迟值数组 static uint32_t delay = 0U; // 当前延
迟值

/*----------------------------------------------------------------------------thrLED: 闪烁 LED *-----------
----------------------------------------------------------*/ __NO_RETURN void thrLED (void *argument) {



   for (;;) {
LED_On (0U); // 打开 LED osDelay (delay); // 延时 LED_Off (0U);
// 关闭 LED osDelay (delay); // 延时 } }


/*----------------------------------------------------------------------------thrBUT: 检查按钮状态 *-------
----------------------------------------------------------------*/ __NO_RETURN static void thrBUT(void
*argument) { uint32_t last = 0; uint32_t state = 0; uint32_t index = 0;



   delay = delay_data[index];

   for (;;) { state = (Buttons_GetState () & 1U); // 获取按下的按钮状态 if (state != last){ //
   仅在状态变化时响应

        if (state == 1){ // 仅在新按下时响应 index++; if (index == (sizeof(delay_data) /
        sizeof(delay_data[0]))){

            index = 0U;
} delay = delay_data[index]; // 从外部闪存获取下一个延时值 } } last = state; osDelay (100U); } }
```

11. 进入项目 – 构建目标 (F7)，并在构建输出窗口中观察其正确构建，没有错误或警告。

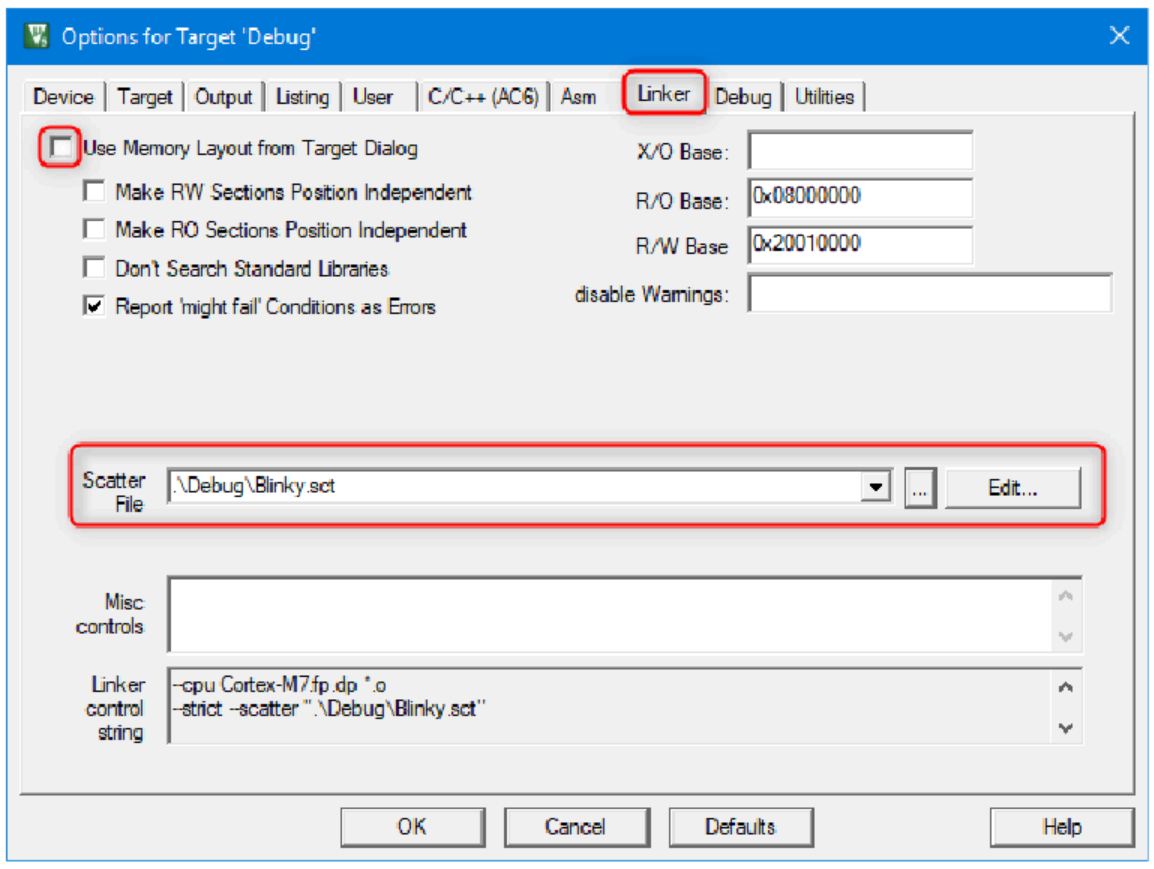
现在，固件镜像可以按照第“程序外部闪存”章节中所述的方法加载到内部和外部闪存中。

通过分散文件进行内存映射

对于复杂情况，可以使用分散文件来定义数据和代码在内存中的位置：

- 进入项目 – 目标选项（Alt+F7） – 链接器选项卡。
  - o 取消勾选使用目标对话框中的内存布局。

    散列文件会根据目标选项卡上指定的当前内存布局自动创建。该文件会显示在散列文件字段中。
  - o R/O Base 和 R/W Base 字段中的值未被使用，可以忽略。
  - o 如有必要，使用散列文件行中的“…”按钮更改散列文件的路径。
  - o 点击编辑按钮，在 µVision 编辑器中打开散列文件。



在我们的示例中，散文件将包含以下内容，其中 Data.c 和 Blinky.c 文件映射到外部数据闪存（ROM1）：

```
; ************************************************************* ; *** 由 uVision 生成
的散装加载描述文件 *** ; *************************************************************


LR_IROM1 0x08000000 0x00200000 { ; 加载区域大小_region ER_IROM1 0x08000000 0x00200000 { ; 加载地
址 = 执行地址 *.o (RESET, +First) *(InRoot$$Sections) .ANY (+RO) .ANY (+XO) } RW_IRAM1 0x20021000
0x0005F000{ ; RW 数据 .ANY (+RW +ZI) } RW_RAM1 0x20020000 UNINIT 0x00001000 {

EventRecorder.o (+ZI) } }

LR_ROM1 0x90000000 0x04000000 { ER_ROM1 0x90000000 0x04000000 { ; 加载地址 = 执行地址
    Data.o (+RO)
    Blinky.o (+RO)
  }
}
```

请参考链接器用户指南中的“散文件语法”章节，了解更多关于散文件的信息。

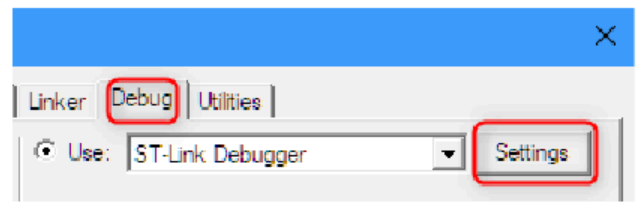
编程外部闪存

本章解释了如何在 MDK 中使用闪存编程算法，并编程一个需要外部内存闪存的固件镜像。

1. 打开一个构建时将代码或常量映射到外部内存闪存的项目。章节"从外部闪存访问数据并执行代码"解释了如何在程序中使用外部闪存内存以及如何相应地配置项目。

2. 将闪存编程算法添加到调试驱动程序设置中。

   - 进入项目 – 目标选项 (Alt+F7) – 调试选项卡。
   - 在"使用"行从下拉菜单中选择目标调试适配器。在我们的案例中这是 ST-LINK。点击设置按钮。

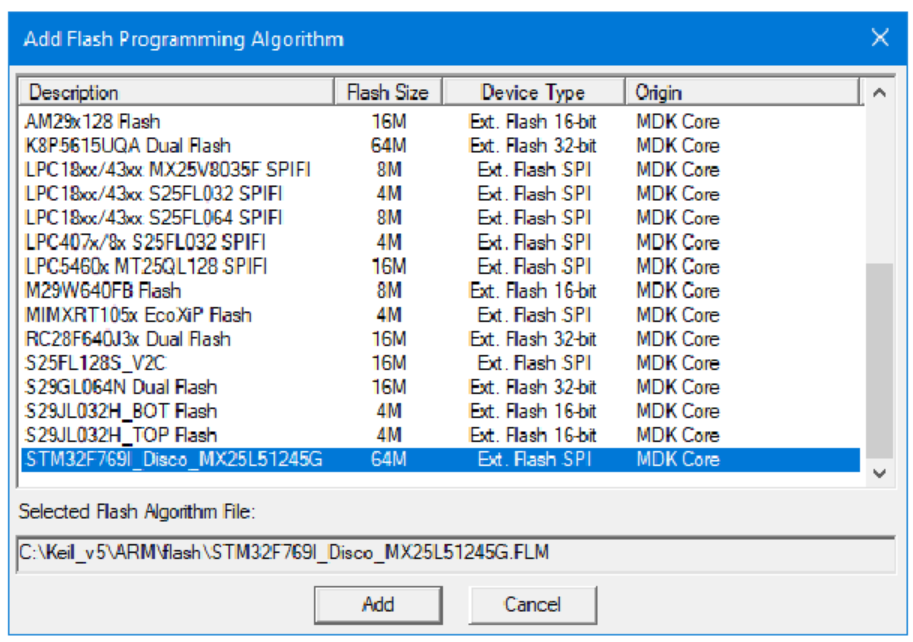


   请注意，设置仅适用于选定的调试驱动器，如果使用不同的调试适配器（例如 ULINKpro），则需要重复这些设置。

   - 进入闪存下载选项卡。
   - 点击添加按钮。它会打开一个可供选择的闪存编程算法列表。

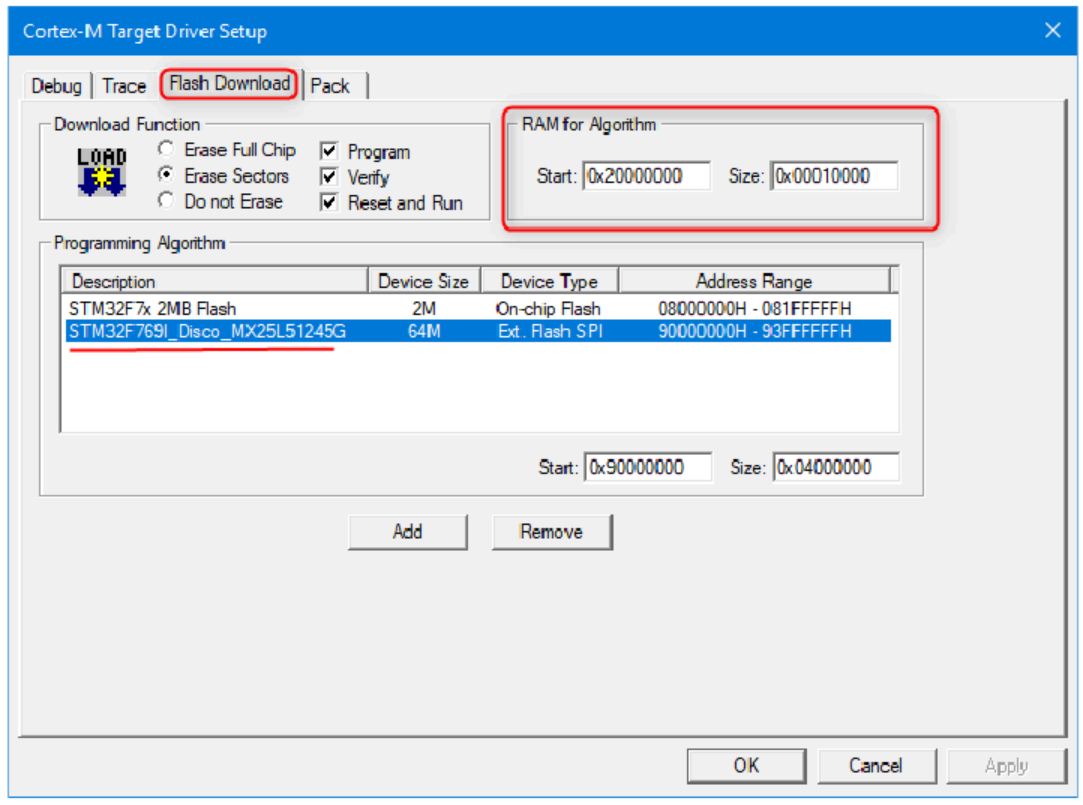
     来源列显示算法的位置。其中一些算法随设备系列包提供，其他算法是 MDK-Core 安装的一部分。

     找到并选择适用于您系统的目标闪存编程算法。它的名称与 FlashDev.c 中指定的字符串匹配。当选择算法时，您还可以看到它的路径并验证是否使用了正确的 FLM 文件。在我们的示例中，我们需要添加算法STM32F769I_Disco_MX25L51245G。



   - 点击添加。该算法会出现在调试适配器将要使用的编程算法列表中。

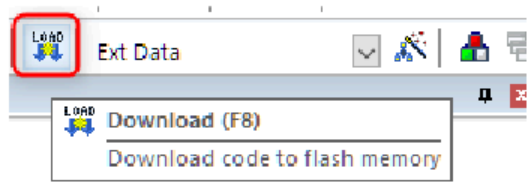
   - 验证算法部分在 RAM 中指定的值。特别是 Size 值需要足够大，以确保有足够的空间用于算法。在我们的示例中，它被设置为 0x00010000。



- 按下 OK。

参见《µVision 用户指南》中“目标驱动程序设置”部分，了解闪存下载调试驱动程序设置的详细信息。

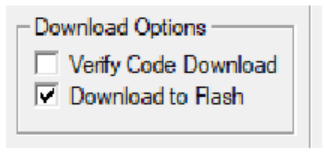
3. 转到“项目” – “构建目标”（F7），并将图像文件（.axf）下载（F8）到目标。



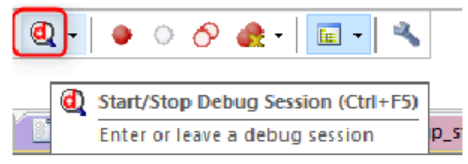
- 观察“构建输出”窗口中的状态。内存擦除、编程和验证应成功。

```
Build Output
Load "C:\\MyPrograms\\STM32F769 Discovery Blinky\\Blinky\\Debug\\Blinky.axf"
Erase Done.
Programming Done.
Verify OK.
Application running ...
Flash Load finished at 15:08:25
```

4. 现在也可以调试程序了。



- 确保在"目标选项..."对话框中，调试选项卡，目标调试器设置中，禁用"验证代码下载"。
- 转到"调试 – 开始/停止调试会话"(Ctrl+F5)。按配置，调试器会在main()函数处停止。



摘要

在本应用笔记中，我们解释了如何使用 Keil MDK 对连接到 STM32 微控制器的外部闪存设备进行编程。它提供了逐步指导，用于创建并使用自定义闪存编程算法，将固件加载到外部闪存设备中。此外，它还展示了一个示例，演示了如何将项目数据代码映射到外部闪存，以及如何在应用程序中访问它。

参考文献和有用链接

[1] µVision 用户指南
[2] µVision 应用笔记 334：MDK 闪存下载 [3] 使用 STM32CubeMX 与 Keil MDK 项目