如何提取与执行位置无关(Position-Independent Code)的Flash Driver

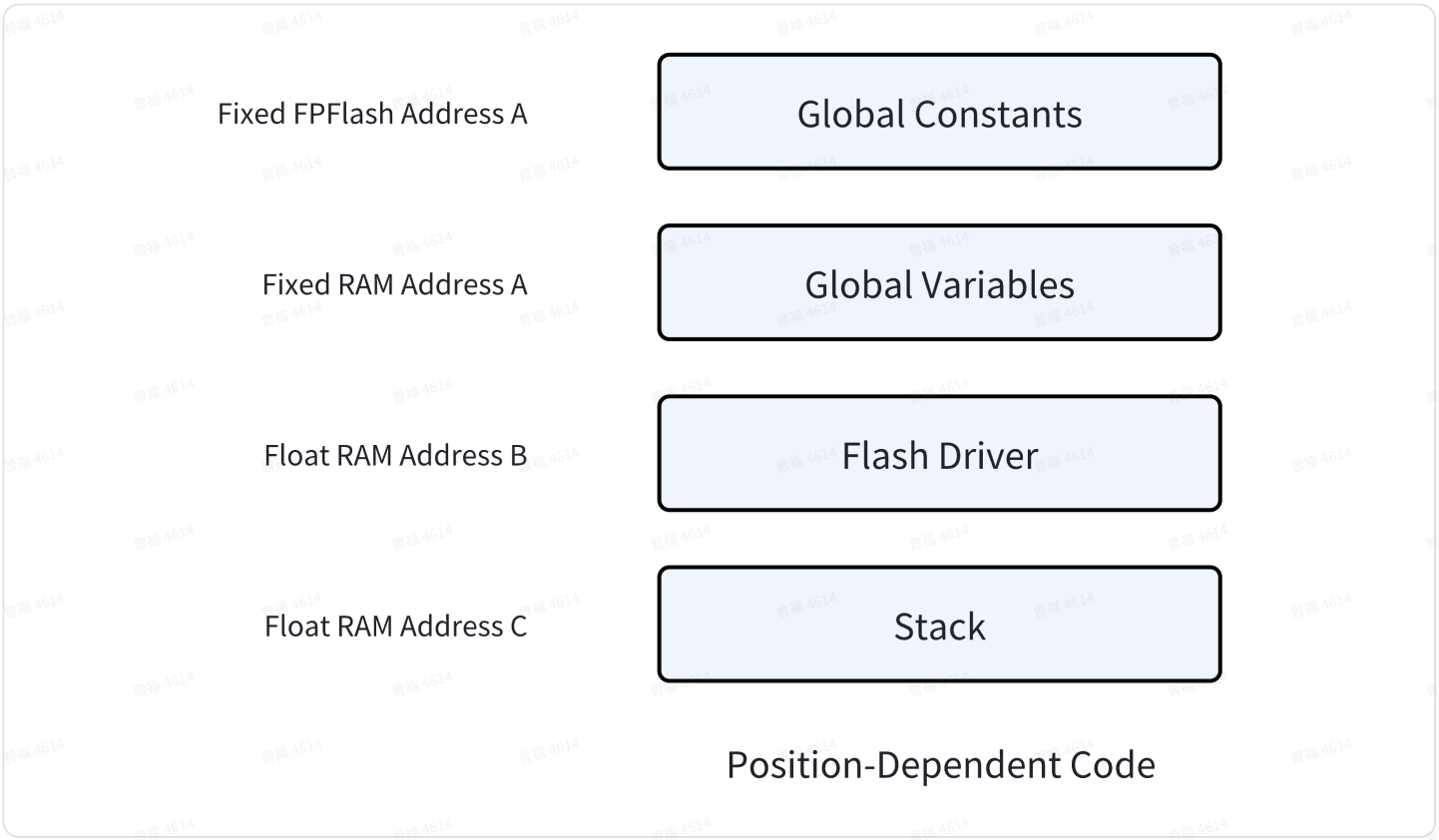
1. 为何要提取Flash Driver?

在汽车ECU bootloader中，需要擦写PFlash中的内容去更新应用固件。如果Flash Driver作为固件内容的一部分存放在PFlash中，增加了固件本身被意外擦除的风险。比如固件代码逻辑问题，导致Flash Driver被错误调用意外擦除了PFlash中的内容。所以汽车行业在bootloader更新PFlash中的固件时，把Flash Driver作为单独的数据通过Bootloader Master ECU远程下载到Bootloader Slave ECU的RAM中。在bootloader更新APP固件时已经成功使用Flash Driver烧写APP固件以后，并完成了相应的APP校验。软复位重启时，Flash Driver因为存放在RAM中会被启动代码的ECC初始化RAM程序覆盖掉。就这样保证了正常运行APP时，PFlash和RAM中没有Flash Driver擦写的代码。降低了PFlash内容在APP运行时被意外擦除的风险。

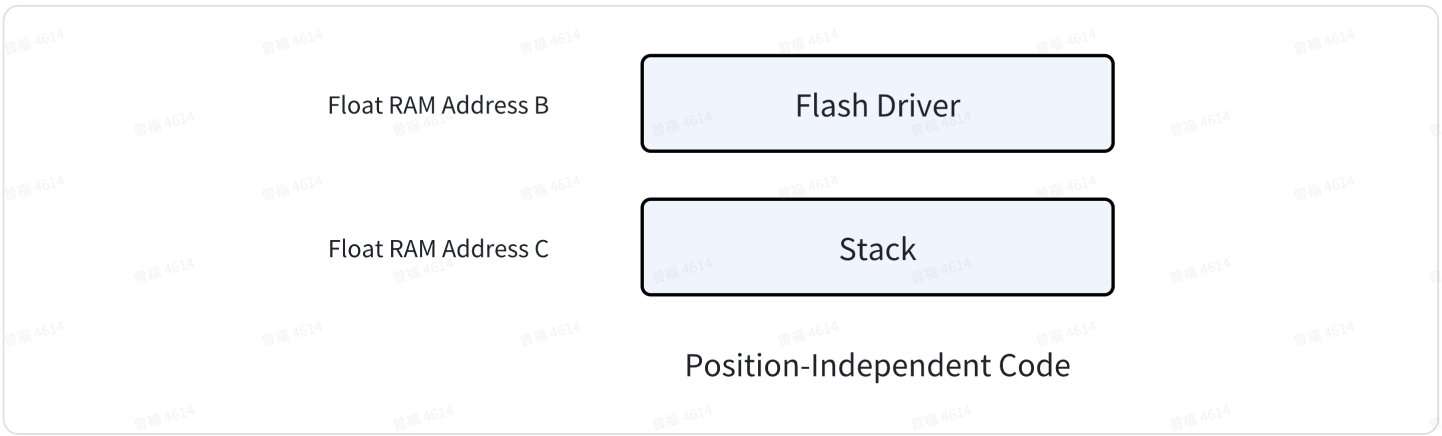
为了达到上述效果，Flash Driver需要支持在任意RAM位置空间都能运行，也就是与执行位置无关(Position-Independent Code)。

2. 什么是与执行位置无关(PIC)的代码?

如下图所示，Flash Driver在Float RAM Address B中执行，需要使用到Fixed PFlash Address A中的Global Constants和Fixed RAM Address A中的Global Variables。Global Constants和Variables的执行地址与当前工程的链接文件相关，它们的执行地址是固定的。这就导致执行Float RAM Address B中Flash Driver时，对其他固定地址Fixed PFlash Address A和Fixed RAM Address A有所依赖。这种情况下，Flash Driver是与执行位置相关的代码。



如下图所示，Flash Driver在Float RAM Address B中执行，只依赖于Float RAM Address C的Stack，Stack存在的地址不需要固定。这种情况下，Flash Driver是与执行位置无关的代码。



3. 如何提取与位置无关的Flash Driver?

根据第2章描述可以看出，要想Flash Driver的代码是与位置无关的，那么Flash Driver就不能依赖于Fixed Global Constants和Variables。最简单的做法就是Flash Driver不使用Global Constants和Variables。另一种做法是把Flash Driver使用到的Global Constants和Variables存放在Flash Driver所在的Float RAM Address B地址中，这样对于Flash Driver整体而言，还是与位置无关的。下面只介绍Flash Driver不使用Global Constants和Variables的情况。如果Flash Driver函数中存在子函数调用，子函数最好使用static inline关键字修饰，或者把子函数也放在Float RAM Address B地址。这里只介绍static inline子函数的情况。

3.1 Flash Driver Table

Bootloader在接收Flash Driver二进制代码到RAM后，需要从存放Flash Driver的RAM中找到相应的PFLASH_EraseSector和PFLASH_Program函数。其中基地址为Flash Driver存放在RAM中的首地址，偏移量就需要从其它地方获取，这个地方就叫Flash Driver Table。

```
/* fls.h */
/*!
 * @brief PFlash erase sector synchronously.
 */
typedef status_t (*PFLASH_EraseSector)(uint32_t dest, uint32_t size);
/*!
 * @brief PFlash program synchronously.
 */
typedef status_t (*PFLASH_Program)(uint32_t dest, uint32_t size, const uint32_t *pData);

/*!
 * @brief Flash driver table structure.
 *
 * This structure holds the function pointers for the flash APIs.
 */
typedef struct
{
    PFLASH_EraseSector EraseSector; /*!< Erase sector function pointer */
    PFLASH_Program Program;         /*!< Program function pointer */
} fls_drv_tbl_t;

/* fls.c */
 /* Base address of flash driver: the address of FLS_DRV_TEXT_start */
#define FLS_DRV_BASE_ADDR 0x400UL

/* Flash driver table: store the offsets of flash driver function pointers */
__attribute__((section (".fls_drv_tbl")))
const fls_drv_tbl_t fls_drv_tbl =
{
    .EraseSector = (PFLASH_EraseSector)((uint32_t)FLASH_DRV_EraseSector - FLS_DRV_BASE_ADDR),
    .Program = (PFLASH_Program)((uint32_t)FLASH_DRV_Program - FLS_DRV_BASE_ADDR)
};
```

3.2 在连接文件中创建Flash Driver相关的段

fls_drv_tbl段用来存放Flash Driver Table，内容是Flash Driver函数指针offset，fls_drv段来存放Flash Driver代码本身。

```
/* yt_link.ld */
MEMORY
{
    FLS_DRV_TEXT (RX) : ORIGIN = 0x400, LENGTH = 512
}

SECTIONS
{
    .FLS_DRV_TEXT : {
        FLS_DRV_TEXT_start = .;
        KEEP(*(.fls_drv_tbl))    /* Store offset of flash driver function pointer */
        KEEP(*(.fls_drv))        /* Store flash driver function */
        FLS_DRV_TEXT_end = .;
    } > FLS_DRV_TEXT
}
```

3.3 将Flash Driver函数放在指定的段

使用__attribute__((section (".fls_drv")))将Flash Driver函数放在fls_drv段。

```
/* flash_driver.c */

/* Use static inline with sub-functions. inline is a MUST to ensure the codes is linked to the pre-assigned memory section */
static inline status_t FLASH_LaunchCommandSequence(uint8_t command)
{
}

__attribute__((section (".fls_drv")))
status_t FLASH_DRV_EraseSector(uint32_t dest, uint32_t size)
{
}

__attribute__((section (".fls_drv")))
```

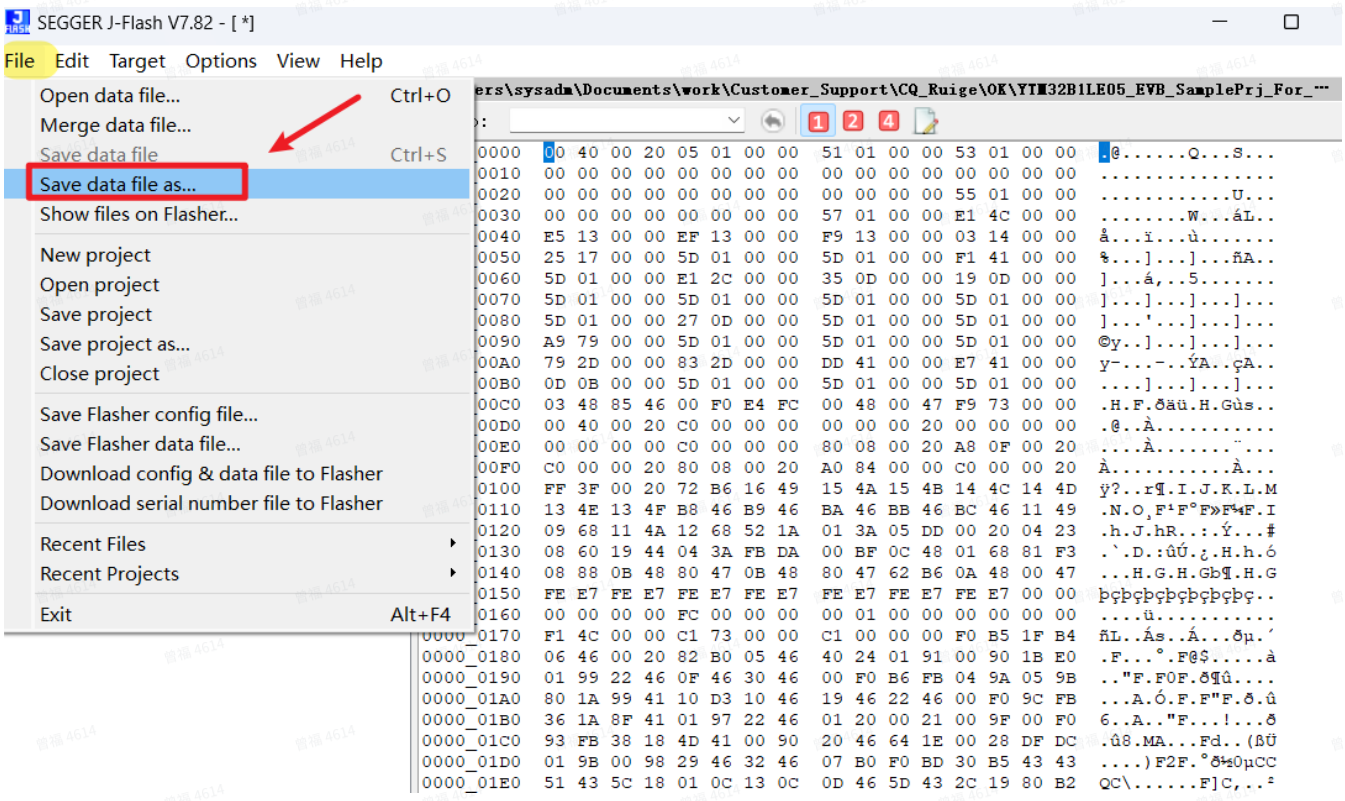
```
14 status_t FLASH_DRV_Program(uint32_t dest, uint32_t size, const uint32_t *
   pData)
15 {
16 }
```

3.4 提取Flash Driver

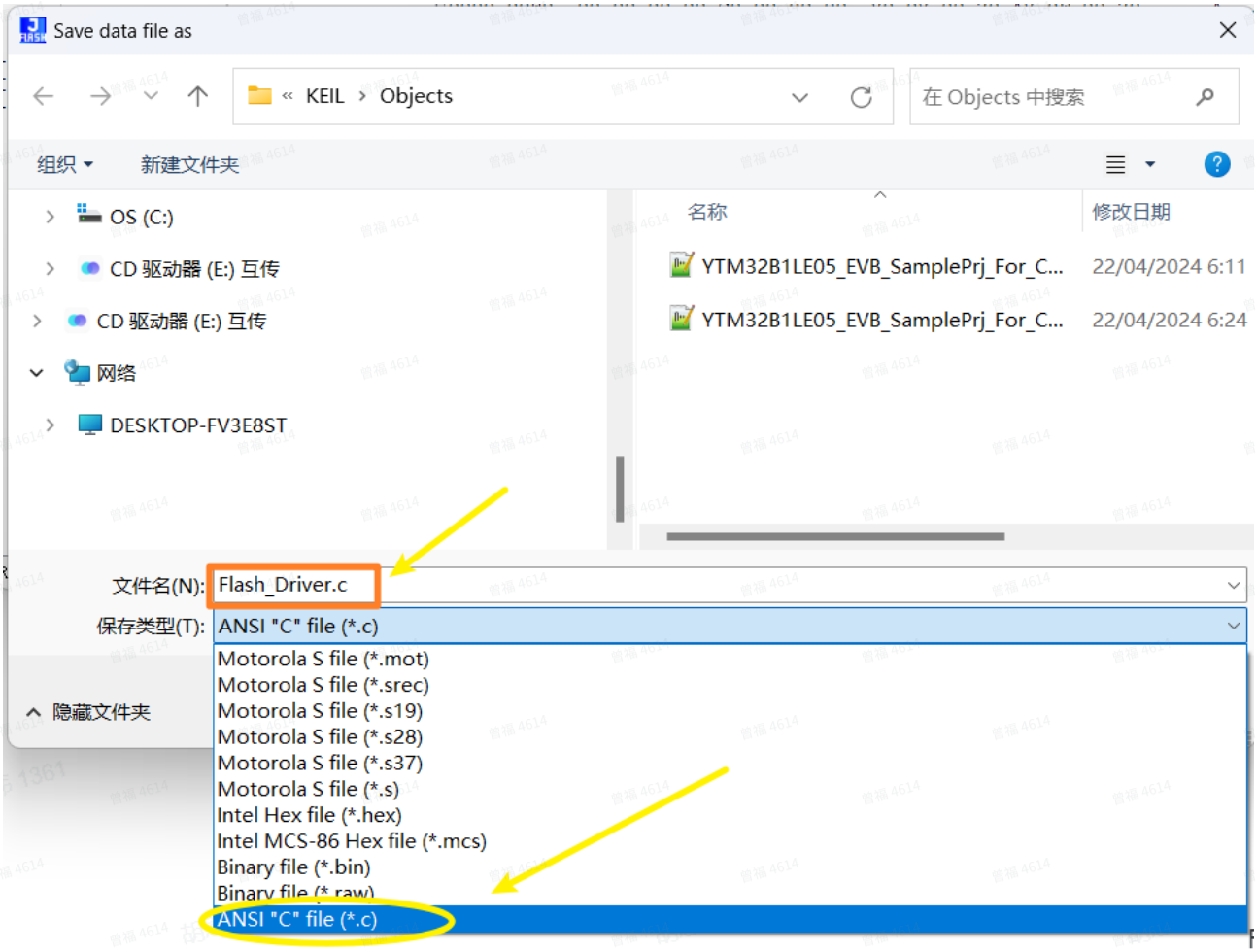
- 方式1：将存放Flash Driver的FLS_DRV_TEXT地址的内容Copy到一个code_ram数组中，直接使用该数组。

```
1 __attribute__((section (".code_ram")))
2 uint32_t fls_drv_bin[512U] =
3 {
4 }
```

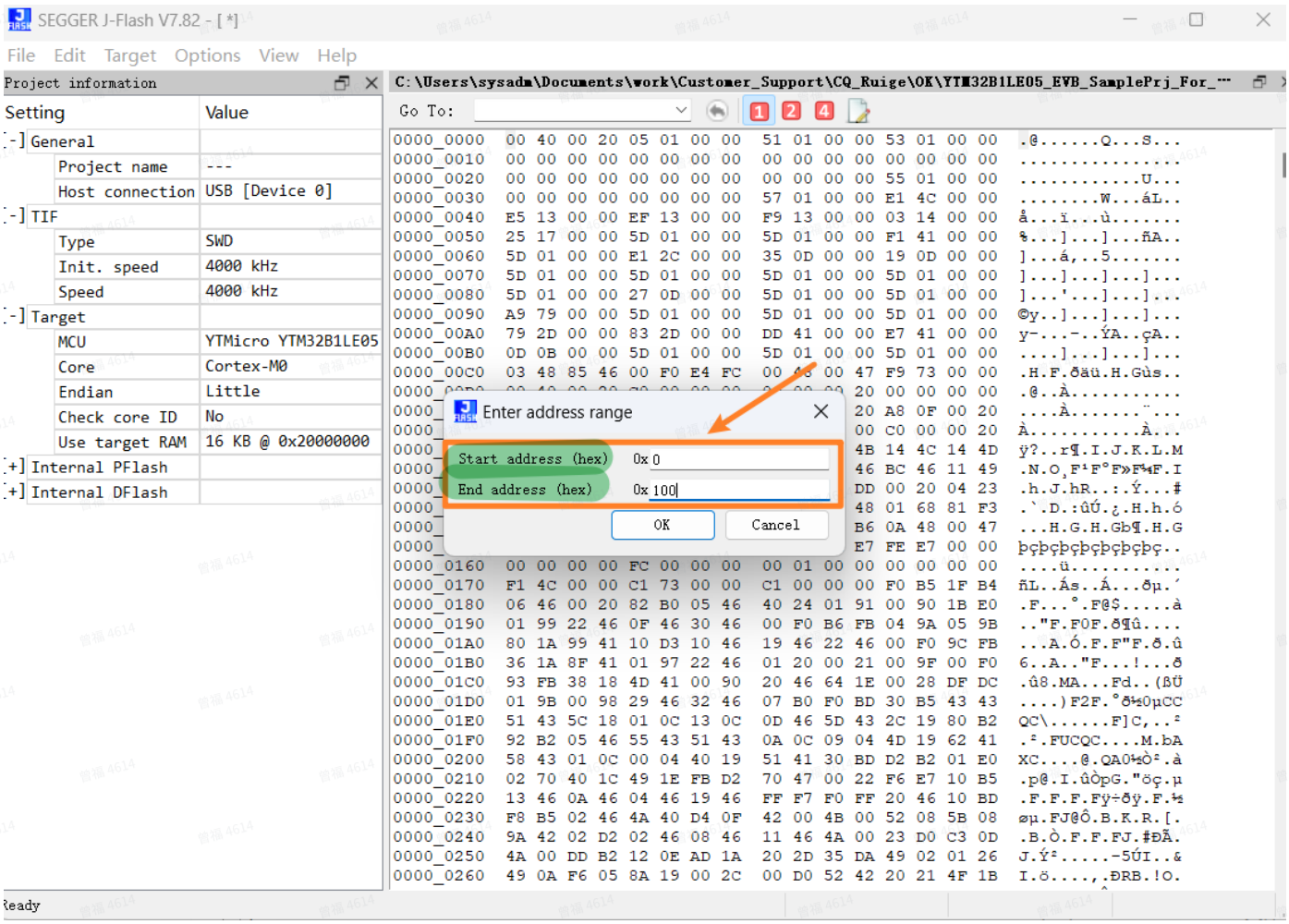
- 方式2：生成S19/HEX文件，将地址从FLS_DRV_TEXT_start到FLS_DRV_TEXT_end部分的代码转换成BIN文件。
- ***Tips***：提取Flash driver为数组或者hex/bin文件可以使用Segger的J-Flash软件，具体操作步骤如下：
  - ①先使用JFlash打开包含Flash driver编译结果的S19/Hex文件；
  - ②然后菜单选择**File** -> "**Save data file as..**"



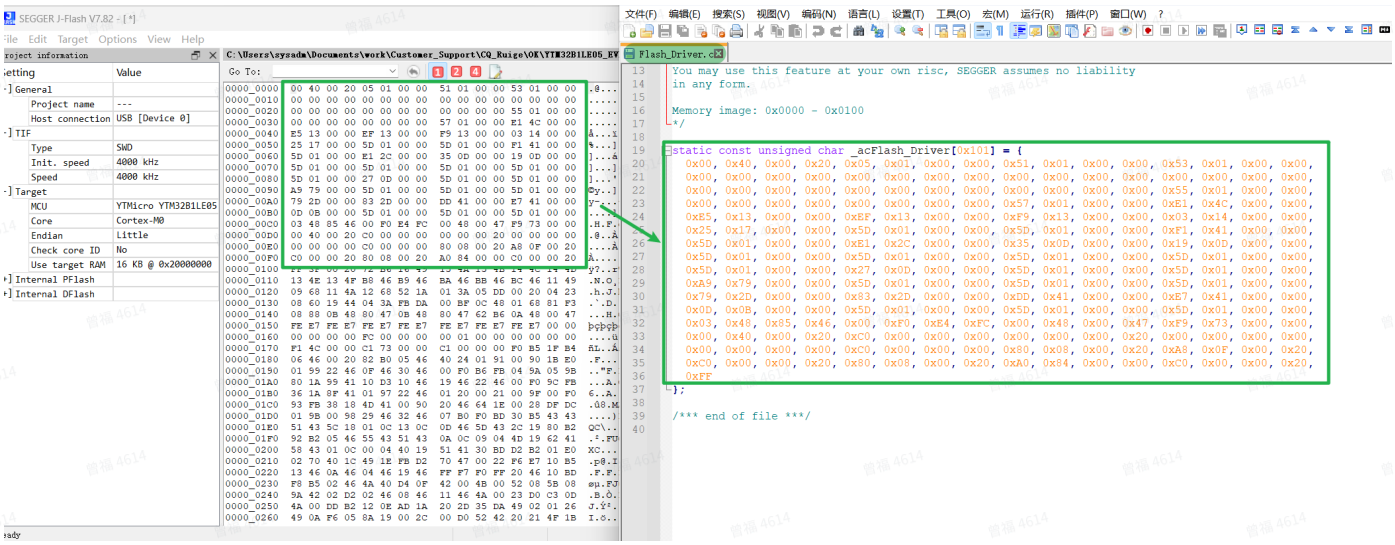
- ③输入文件名，选择保存类型为:ANSI “C” file(*.c)



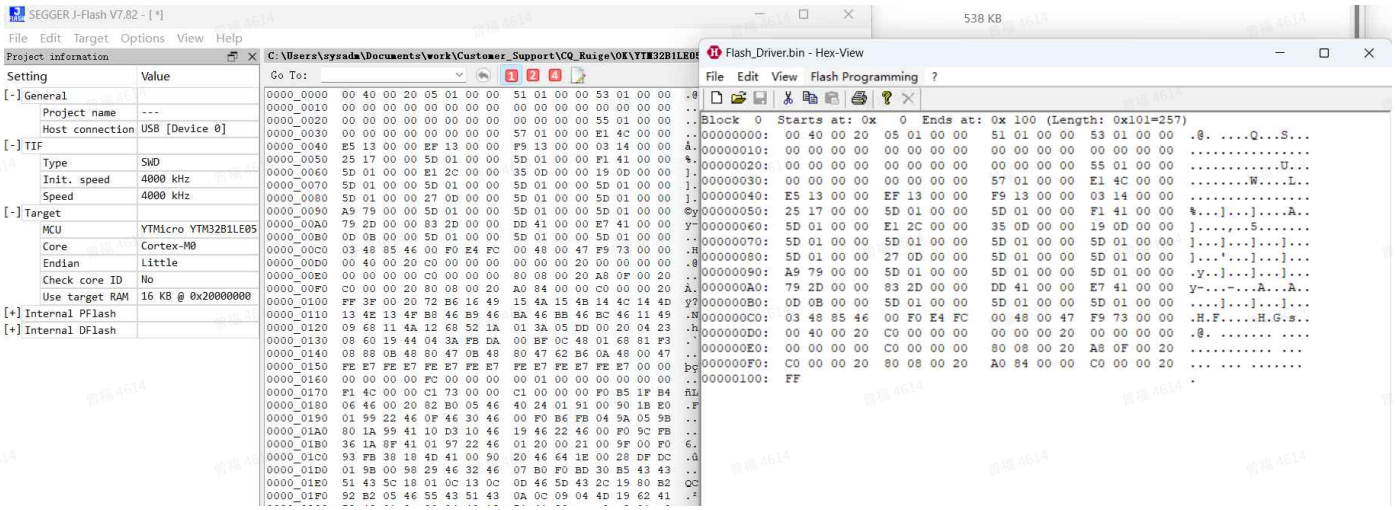
- ④输入要转换的起始地址（hex），比如0x00~0x100：



保存结果如下：



如果是要通过CAN/LIN总线下载，则可以将Flash driver编译结果地址（从map文件查看）选择导出为所需的bin/hex文件。



4. 如何使用提取的Flash Driver?

Flash Driver下载到RAM中以后，基地址就是存放Flash Driver的地址，offset是fls_drv_tbl中的内容。Flash Driver函数调用地址=基地址+offset。使用时需要先计算出函数的调用地址，然后再进行常规的函数调用。

```
/* main.c */
#if (RUN_FLASH_DRIVER == RUN_FLASH_DRIVER_BIN)
    status_t status;
    fls_drv_tbl_t * pfls_drv_tbl = (fls_drv_tbl_t *)fls_drv_bin;

    /* Flash driver function pointer address = (flash driver function pointer
     * offset in fls_drv_bin) + (the base address of fls_drv_bin: fls_drv_bin
     */
    for (uint8_t i = 0; i < sizeof(fls_drv_tbl_t)/sizeof(uint32_t *); ++i)
    {
        fls_drv_bin[i] += (uint32_t)fls_drv_bin;
    }

    /* Erase PFlash1 Sector 0 */
    status = pfls_drv_tbl->EraseSector(FLASH_TEST_ADDR,
FEATURE_EFM_MAIN_ARRAY_SECTOR_SIZE);
    DEV_ASSERT(status == STATUS_SUCCESS);
    /* Verify PFlash data after erasing. */
    for (uint8_t i = 0U; i < FLASH_TEST_SIZE; ++i)
    {
        ReadBuffer[i] = *(volatile uint8_t *)(FLASH_TEST_ADDR + i);
        DEV_ASSERT(ReadBuffer[i] == 0xFFU);
    }
    /* Write data to PFlash1 Sector 0 */
    status = pfls_drv_tbl->Program(FLASH_TEST_ADDR, FLASH_TEST_SIZE, (const
uint32_t *)WriteBuffer);
    DEV_ASSERT(status == STATUS_SUCCESS);
```

```
26      /* Verify PFlash data after programming. */
27      for (uint8_t i = 0U; i < FLASH_TEST_SIZE; ++i)
28      {
29          ReadBuffer[i] = *(volatile uint8_t *)(FLASH_TEST_ADDR + i);
30          DEV_ASSERT(ReadBuffer[i] == WriteBuffer[i]);
31      }
32  #endif
```