AN0026
应用笔记

Extending SRAM in User's Program

前言

这篇应用指南描述了怎么在AT32F403/413芯片上执行程序来扩展SRAM。

支持型号列表：

支持型号	AT32F403xx
	AT32F403Axx
	AT32F413xx

目录

表目录

图目录

1 概述

AT32F403及AT32F413的片上SRAM，都有提供一个特别的SRAM扩展模式，可让用户通过设定选择字节中的EOPB0来调整SRAM的大小。

一般此扩展模式的设置都建议使用雅特力的ICP或ISP工具来完成，但在使用者无法使用ICP/ISP工具的情境下，就只能通过执行程序来完成。

本篇指南将说明如何在程序中正确的设置EOPB0来完成SRAM的扩展。

2 设置例程

基本上不管是AT32F403还是AT32F413，都是通过修改EOPB0的值来完成SRAM的扩展，但因为AT32F403及AT32F413能够调整的SRAM容量大小并不相同，所以个别都有提供一个例程供使用者参考。

其中，AT32F403的例程放在标准库BSP的\Project\AT_START_F403\Examples\SRAM\extend_SRAM目录；而AT32F413的例程则放在标准库BSP的\Project\AT_START_F413\Examples\SRAM\extend_SRAM目录。

2.1 AT32F403 例程

2.1.1 函数说明

AT32F403出厂预设的SRAM大小为96K字节，修改EOPB0后可扩展至224K字节，EOPB0的设定值如下：

表 1. AT32F403 EOPB0 设定值

EOPB0: 扩充的选择字节	
位7:0	224KB_MODE 0xFE：片上内存为224K 字节 0xFF：片上内存为96K 字节 其他设置保留

修改EOPB0的程序主要是写在 extend_SRAM( )函数内。

图 1. extend_SRAM 函数

```
void extend_SRAM(void)
{
/* Target set_SRAM_96K is selected */
#ifdef EXTEND_SRAM_96K
// check if RAM has been set to 96K, if not, change EOPB0
  if(((UOPTB->EOPB0)&0xFF)!=0xFF)
  {
    /* Unlock Option Bytes Program Erase controller */
    FLASH_Unlock();
    /* Erase Option Bytes */
    FLASH_EraseUserOptionBytes();
    /* Change SRAM size to 96KB */
    FLASH_ProgramUserOptionByteData((uint32_t)&UOPTB->EOPB0,0xFF);
    NVIC_SystemReset();
  }
#endif

/* Target set_SRAM_224K is selected */
#ifdef EXTEND_SRAM_224K
// check if RAM has been set to 224K, if not, change EOPB0
  if(((UOPTB->EOPB0)&0xFF)!=0xFE)
  {
    /* Unlock Option Bytes Program Erase controller */
    FLASH_Unlock();
    /* Erase Option Bytes */
    FLASH_EraseUserOptionBytes();
    /* Change SRAM size to 224KB */
    FLASH_ProgramUserOptionByteData((uint32_t)&UOPTB->EOPB0,0xFE);
    NVIC_SystemReset();
  }
#endif
}
```

此函数通过修改EOPB0，可将SRAM从默认的96K字节扩展到224K字节，或从224K字节改回96K字

节。须注意extend_RAM( )函数内，不可使用全局变量。通过 EXTEND_SRAM_224K 或 EXTEND_SRAM_96K的宏定义可以做选择。修改EOPB0之后，必须执行系统重置，新的EOPB0数值才会生效并真正的设定到所选的SRAM大小。

然后必须修改 startup_at32f403xxxx.s 的启动汇编代码，用户必须根据所选择型号的启动文件来作修改。此例程是用AT32F403ZGT6，范例中的 startup_at32f403zx_xl_ext_ram.s就是修改后的Reset_Handler。

图 2. Reset_Handler

```
                AREA    |.text|, CODE, READONLY

; Reset handler
Reset_Handler   PROC
                EXPORT  Reset_Handler                   [WEAK]
                IMPORT  __main
                IMPORT  SystemInit
                IMPORT  extend_SRAM

                MOV32   R0, #0x20001000
                MOV     SP, R0
                LDR     R0, =extend_SRAM
                BLX     R0
                MOV32   R0, #0x08000000
                LDR     SP, [R0]

                LDR     R0, =SystemInit
                BLX     R0
                LDR     R0, =__main
                BX      R0
                ENDP
```

需注意的两个重点:

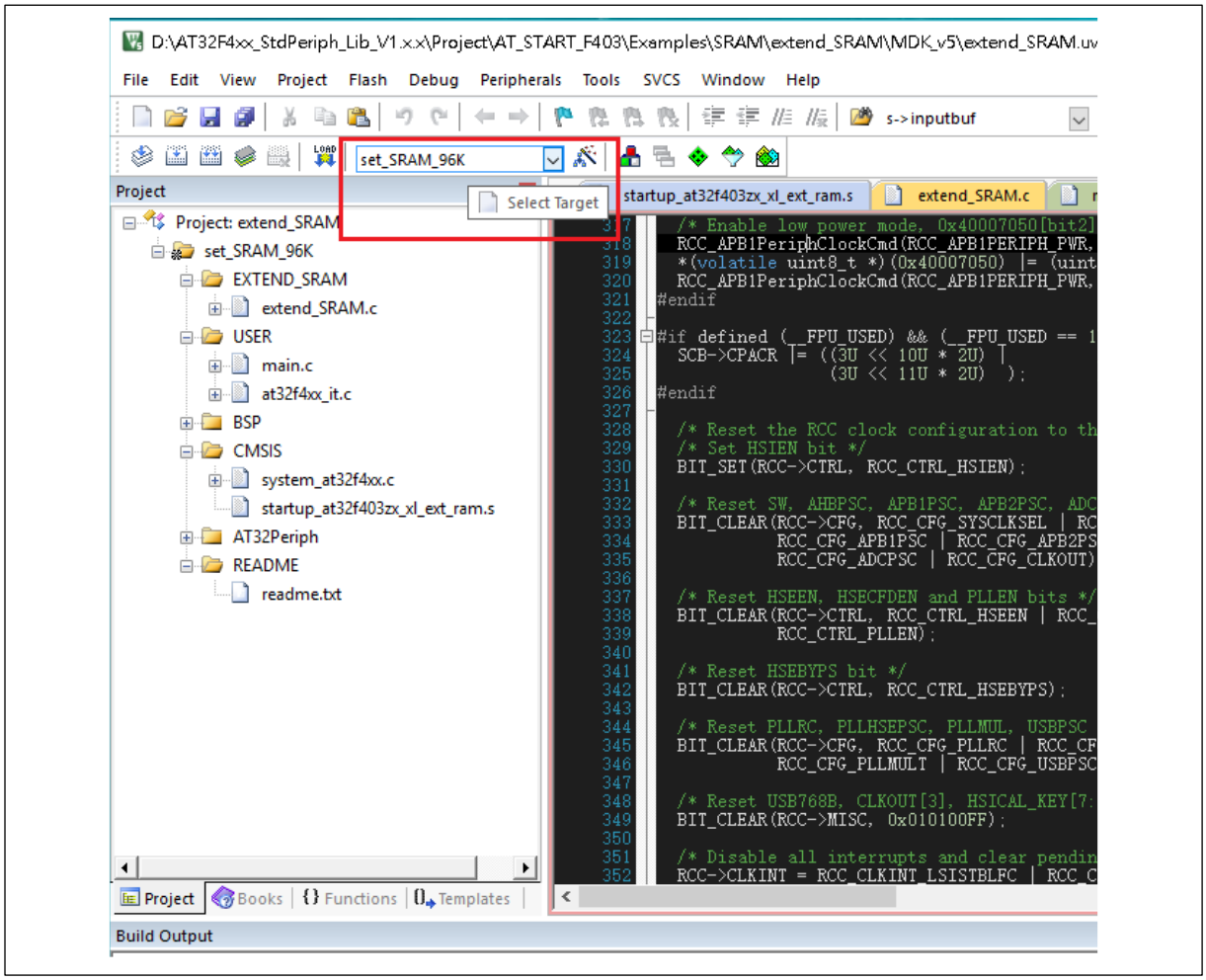
1) 必须在Reset_Handler的一开头就去做EOPB0的修改，不能在SystemInit( )函数里头做设置主要是因为用户一开始在Keil/IAR开发环境设定的SRAM范围，就可能是以扩充后的224K字节作设定，且实际用到的SRAM可能超过了默认的96K，此时堆栈(STACK)的指针初始值会被设定到96K之后的地址，执行SystemInit( )时就会出错，甚至发生HardFault而造成死机。

2) 在调用extend_SRAM( )函数前，要将堆栈(STACK)的指针先改到 96K字节内(例程中是修改到(0x20001000)，避免因STACK的指针初始值被设定到96K之后的地址，而造成extend_SRAM( )执行时发生错误。

2.1.2 例程展示

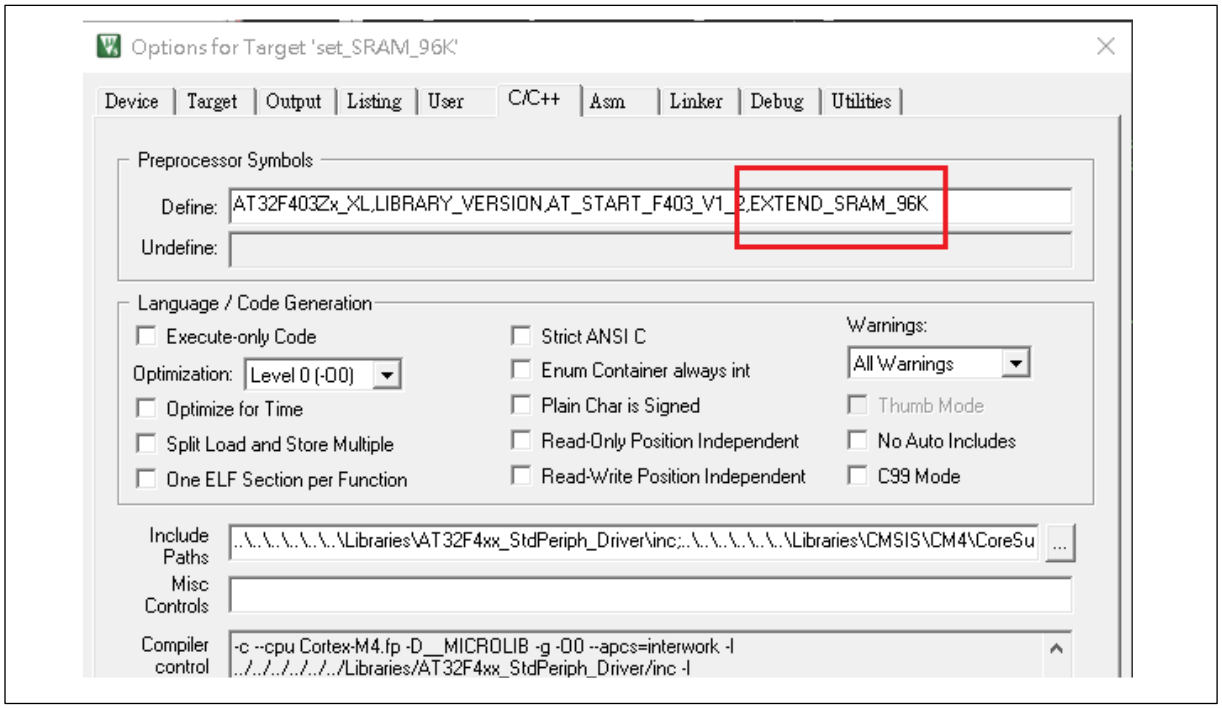
在同一个例程项目内，可以通过”Select Target”窗口选择96K 字节或224K 字节的范例程序。如下图：

图 3. 程序 SRAM 大小选择



当选择set_SRAM_96K 时，EXTEND_SRAM_96K 的宏定义就会被设置在 C/C++ →Preprocessor Symbols 的定义框里面，编译时extend_SRAM( )函数就会选择96K SRAM 的配置。

图 4. C/C++/Preprocessor Symbols 配置



同样的，当选择set_SRAM_224K 时，EXTEND_SRAM_224K 的宏定义就会被设置，编译时extend_SRAM( )函数就会选择224K SRAM 的配置。

扩展完成并进入main( )函数时，会检查EOPB0 的数值以确认是否有正确的去配置成所选择的SRAM大小，并且从USART1 串口打印配置的结果。

2.2 AT32F413 例程

2.2.1 函数说明

AT32F413出厂预设的SRAM大小为32K字节，修改EOPB0后可减少为16K字节或扩展至64K字节，EOPB0的设定值如下

表 2. AT32F413 EOPB0 设定值

EOPB0: 扩充的选择字节	
位1：0	0x0：片上SRAM 64K 字节 0x1：片上SRAM 16K 字节 0x2：片上SRAM 64K 字节 0x3：片上SRAM 32K 字节

修改EOPB0的程序一样是写在 extend_SRAM( )函数内，一样用宏定义来选择SRAM的大小。

图 5. extend_SRAM( )函数

```
void extend_SRAM(void)
{
/* Target set_SRAM_16K is selected */
#ifdef EXTEND_SRAM_16K
  // check if RAM has been set to 16K, if not, change EOPB0
  if(((UOPTB->EOPB0)&0x03)!=0x01)
  {
    /* Unlock Option Bytes Program Erase controller */
    FLASH_Unlock();
    /* Erase Option Bytes */
    FLASH_EraseUserOptionBytes();
    /* Change SRAM size to 16KB */
    FLASH_ProgramUserOptionByteData((uint32_t)&UOPTB->EOPB0,0xFD);
    NVIC_SystemReset();
  }
#endif

/* Target set_SRAM_32K is selected */
#ifdef EXTEND_SRAM_32K
  // check if RAM has been set to 32K, if not, change EOPB0
  if(((UOPTB->EOPB0)&0x03)!=0x03)
  {
    /* Unlock Option Bytes Program Erase controller */
    FLASH_Unlock();
    /* Erase Option Bytes */
    FLASH_EraseUserOptionBytes();
    /* Change SRAM size to 32KB */
    FLASH_ProgramUserOptionByteData((uint32_t)&UOPTB->EOPB0,0xFF);
    NVIC_SystemReset();
  }
```

```
  #endif

/* Target set_SRAM_64K is selected */
#ifdef EXTEND_SRAM_64K
  // check if RAM has been set to 64K, if not, change EOPB0
  if(((UOPTB->EOPB0)&0x01))
  {
    /* Unlock Option Bytes Program Erase controller */
    FLASH_Unlock();
    /* Erase Option Bytes */
    FLASH_EraseUserOptionBytes();
    /* Change SRAM size to 64KB */
    FLASH_ProgramUserOptionByteData((uint32_t)&UOPTB->EOPB0,0xFC);
    NVIC_SystemReset();
  }
#endif
```

然后必须修改 startup_at32f413xxxx.s 启动汇编代码，用户必须根据所选择型号的启动文件来作修改。此例程是用AT32F413RCT7，范例中的 startup_at32f403rx_hd_ext_ram.s就是修改后的Reset_Handler。

需注意的两个重点:

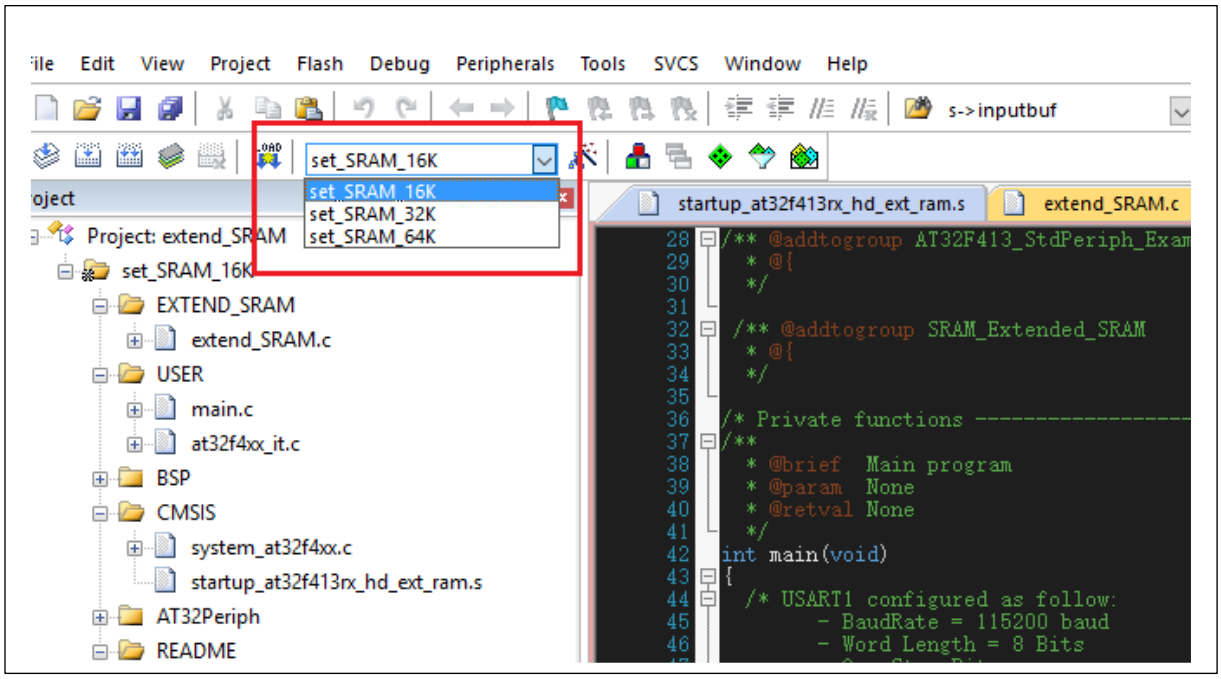
1) 必须在Reset_Handler的一开头就去做EOPB0的修改，不能在SystemInit( )函数里做设置主要是因为用户一开始在Keil/IAR开发环境设定的SRAM范围，就可能是以扩充后的64K字节作设定，且实际用到的SRAM可能超过了默认的32K字节，此时堆栈(STACK)的指针初始值会被设定到32K之后的地址，执行SystemInit( )时就会出错，甚至发生HardFault而造成死机。

2) 因为当前的SRAM大小有可能是16K字节，在调用extend_SRAM( )函数前，要将堆栈(STACK)的指针先改到16K字节内(例程中是修改到0x20001000)，避免STACK的指针初始值被设定到16K之后的地址，而造成extend_SRAM( )执行时发生错误。

2.2.2 例程展示

在同一个例程项目内，可以通过”Select Target”窗口选择16K 字节、32K 字节或64K 字节的范例程序。如下图

图 6. 程序 SRAM 大小选择



SRAM 扩展完成并进入main( )函数时，会检查EOPB0 的数值以确认是否有正确的去配置成所选择的SRAM 大小，并且从USART1 串口打印配置的结果。

2.3 AT32F403A 例程

2.3.1 函数说明

AT32F403A出厂预设的SRAM大小为96K字节，修改EOPB0后可扩展至224K字节，EOPB0的设定值如下：

表 3. AT32F403A EOPB0 设定值

EOPB0: 扩充的选择字节	
位7:0	224KB_MODE 0xFE：片上内存为224K 字节 0xFF：片上内存为96K 字节 其他设置保留

修改EOPB0的程序一样是写在 extend_SRAM( )函数内，一样用宏定义来选择SRAM的大小。

图 7. extend_SRAM 函数

```
void extend_SRAM(void)
{
/* Target set_SRAM_96K is selected */
#ifdef EXTEND_SRAM_96K
// check if RAM has been set to 96K, if not, change EOPB0
  if(((UOPTB->EOPB0)&0xFF)!=0xFF)
  {
    /* Unlock Option Bytes Program Erase controller */
    FLASH_Unlock();
    /* Erase Option Bytes */
    FLASH_EraseUserOptionBytes();
    /* Change SRAM size to 96KB */
    FLASH_ProgramUserOptionByteData((uint32_t)&UOPTB->EOPB0,0xFF);
    NVIC_SystemReset();
  }
#endif

/* Target set_SRAM_224K is selected */
#ifdef EXTEND_SRAM_224K
// check if RAM has been set to 224K, if not, change EOPB0
  if(((UOPTB->EOPB0)&0xFF)!=0xFE)
  {
    /* Unlock Option Bytes Program Erase controller */
    FLASH_Unlock();
    /* Erase Option Bytes */
    FLASH_EraseUserOptionBytes();
```

```
    /* Change SRAM size to 224KB */
    FLASH_ProgramUserOptionByteData((uint32_t)&UOPTB->EOPB0,0xFE);
    NVIC_SystemReset();
  }
#endif
}
```

此函数通过修改EOPB0，可将SRAM从默认的96K字节扩展到224K字节，或从224K字节改回96K字节。须注意extend_RAM( )函数内，不可使用全局变量。通过 EXTEND_SRAM_224K 或 EXTEND_SRAM_96K的宏定义可以做选择。修改EOPB0之后，必须执行系统重置，新的EOPB0数值才会生效并真正的设定到所选的SRAM大小。

然后必须修改 startup_at32f403Axxxx.s 的启动汇编代码，用户必须根据所选择型号的启动文件来作修改。此例程是用AT32F403AVGT7，范例中的 startup_at32f403avgt7_ext_ram.s就是修改后的Reset_Handler。

图 8. Reset_Handler

```
                AREA    |.text|, CODE, READONLY

; Reset handler
Reset_Handler   PROC
                EXPORT  Reset_Handler                   [WEAK]
                IMPORT  __main
                IMPORT  SystemInit
                IMPORT  extend_SRAM

                MOV32   R0, #0x20001000
                MOV     SP, R0
                LDR     R0, =extend_SRAM
                BLX     R0
                MOV32   R0, #0x08000000
                LDR     SP, [R0]

                LDR     R0, =SystemInit
                BLX     R0
                LDR     R0, =__main
                BX      R0
                ENDP
```

需注意的两个重点:

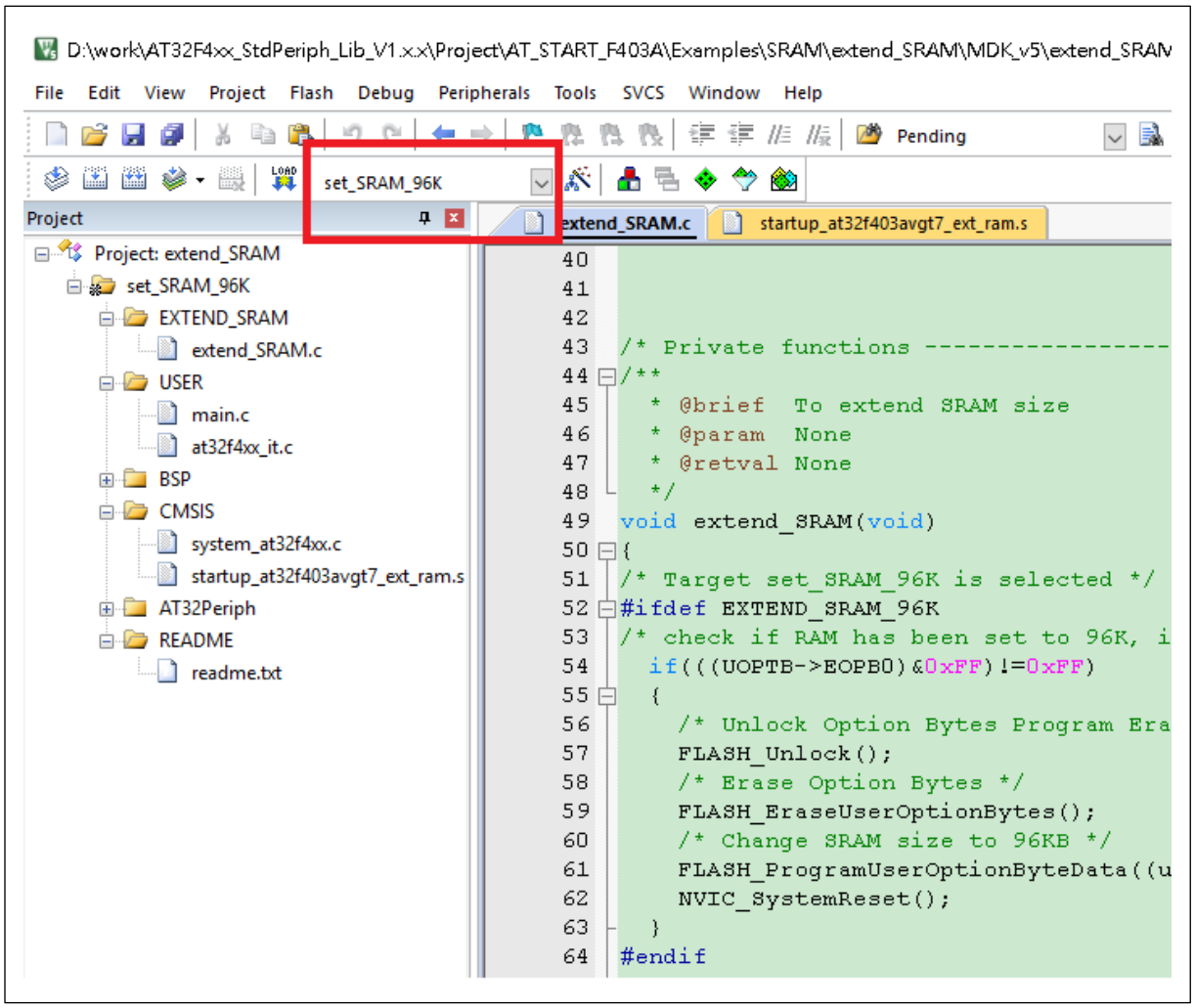
3) 必须在Reset_Handler的一开头就去做EOPB0的修改，不能在SystemInit( )函数里头做设置主要是因为用户一开始在Keil/IAR开发环境设定的SRAM范围，就可能是以扩充后的224K字节作设定，且实际用到的SRAM可能超过了默认的96K，此时堆栈(STACK)的指针初始值会被设定到96K之后的地址，执行SystemInit( )时就会出错，甚至发生HardFault而造成死机。

4) 在调用extend_SRAM( )函数前，要将堆栈(STACK)的指针先改到 96K字节内(例程中是修改到(0x20001000)，避免因STACK的指针初始值被设定到96K之后的地址，而造成extend_SRAM( )执行时发生错误。

2.3.2 例程展示

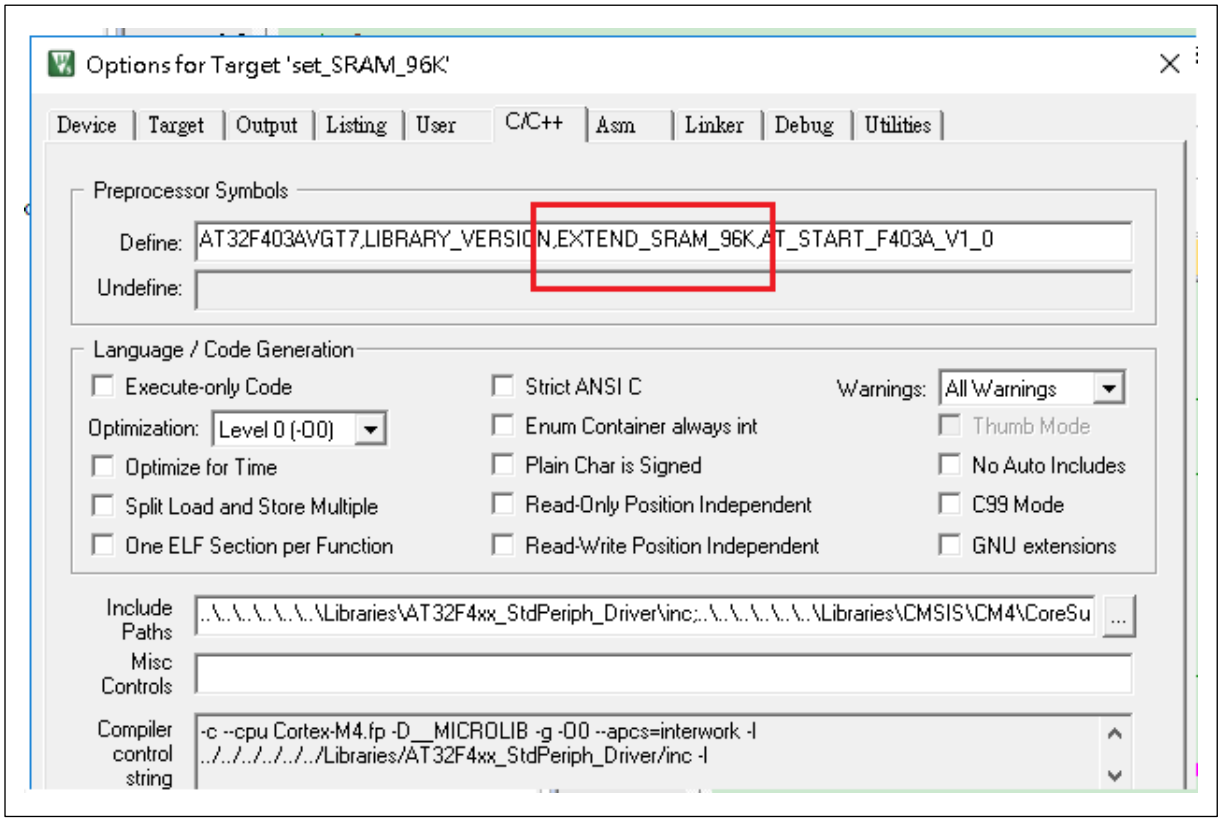
在同一个例程项目内，可以通过”Select Target”窗口选择96K 字节或224K 字节的范例程序。如下图：

图 9. 程序 SRAM 大小选择



当选择set_SRAM_96K 时，EXTEND_SRAM_96K 的宏定义就会被设置在 C/C++ →Preprocessor Symbols 的定义框里面，编译时extend_SRAM( )函数就会选择96K SRAM 的配置。

图 10. C/C++/Preprocessor Symbols 配置



同样的，当选择set_SRAM_224K 时，EXTEND_SRAM_224K 的宏定义就会被设置，编译时extend_SRAM( )函数就会选择224K SRAM 的配置。

扩展完成并进入main( )函数时，会检查EOPB0 的数值以确认是否有正确的去配置成所选择的SRAM大小，并且从USART1 串口打印配置的结果。

3 版本历史

表 4. 文档版本历史

日期	版本	变更
2019.11.12	1.0.0	最初版本

重要通知 - 请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和/或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途(及其依据任何司法管辖区的法律的对应情况)，或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力产品并非设计或专门用于下列用途的产品：(A) 对安全性有特别要求的应用，如：生命支持、主动植入设备或对产品功能安全有要求的系统；(B) 航空应用；(C) 汽车应用或汽车环境；(D) 航天应用或航天环境，且/或(E) 武器。因雅特力产品不是为前述应用设计的，而采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，风险由购买者单独承担，并且独力负责在此类相关使用中满足所有法律和法规要求。

经销的雅特力产品如有不同于本文档中提出的声明和/或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。