Memory 分区

- 在CCS和IAR环境下都可以设置自己的memory分区来根据应用的需求最大化的利用FRAM资源.
- Memory分区是由IDE相关的linker文件来决定的。
  - 每个CCS project 都会有独立的linker文件(.cmd)存储在项目文件夹下
  - IAR的项目通常会共享在C:\<IAR installation directory>\430\config\linker\ folder路径下的同一个.xcl linker文件
  - IAR project需要额外配置使得此项目使用你所定义的这个.xcl 文件。
- 在low level initialization函数中禁止FRAM写保护

```
int _system_pre_init()        //CCS low level initial function
Int __low_level_init()        // IAR low level initial function
```

修改CCS的linker文件 (1)

- 在项目文件夹下找到.cmd linker文件
- 按如下描述修改memory map

```
/*****************************************************************************/
/* Specify the system memory map                                             */
/*****************************************************************************/

MEMORY
{
    SFR                     : origin = 0x0000, length = 0x0010
    PERIPHERALS_8BIT        : origin = 0x0010, length = 0x00F0
    PERIPHERALS_16BIT       : origin = 0x0100, length = 0x0100
    RAM                     : origin = 0x2000, length = 0x0800
    INFOA                   : origin = 0x1800, length = 0x0200
    FRAM                    : origin = 0xC400, length = 0x3B80
    JTAGSIGNATURE           : origin = 0xFF80, length = 0x0004, fill = 0xFFFF
    BSLSIGNATURE            : origin = 0xFF84, length = 0x0004, fill = 0xFFFF
    INT00                   : origin = 0xFF88, length = 0x0002
```

```
/*****************************************************************************/
/* Specify the system memory map                                             */
/*****************************************************************************/

MEMORY
{
    SFR                     : origin = 0x0000, length = 0x0010
    PERIPHERALS_8BIT        : origin = 0x0010, length = 0x00F0
    PERIPHERALS_16BIT       : origin = 0x0100, length = 0x0100
    RAM                     : origin = 0x2000, length = 0x0800
    INFOA                   : origin = 0x1800, length = 0x0100
    F2S_RAM                 : origin = 0x1900, length = 0x0100
    FRAM                    : origin = 0xC400, length = 0x3B80
    JTAGSIGNATURE           : origin = 0xFF80, length = 0x0004, fill = 0xFFFF
    BSLSIGNATURE            : origin = 0xFF84, length = 0x0004, fill = 0xFFFF
    INT00                   : origin = 0xFF88, length = 0x0002
```

增加 F2S_RAM段，将information memory分成两个部分。注意根据需要修改相应地址和长度

修改CCS的linker文件 (2)

- 将相应的section分配到多个memory segment中。新创建的F2S_RAM为这些section增加了可使用的RAM空间

```
.bss        : {} > RAM              /* Global & static vars             */
.data       : {} > RAM              /* Global & static vars             */
.TI.noinit  : {} > RAM              /* For #pragma noinit               */
.cio        : {} > RAM              /* C I/O buffer                     */
.sysmem     : {} > RAM              /* Dynamic memory allocation area   */
.stack      : {} > RAM (HIGH)       /* Software system stack            */
```

```
.bss        : {} > F2S_RAM | RAM    /* Global & static vars             */
.data       : {} > F2S_RAM | RAM    /* Global & static vars             */
.TI.noinit  : {} > F2S_RAM | RAM    /* For #pragma noinit               */
.cio        : {} > F2S_RAM | RAM    /* C I/O buffer                     */
.sysmem     : {} > F2S_RAM | RAM    /* Dynamic memory allocation area   */
.stack      : {} > RAM (HIGH)       /* Software system stack            */
```

用 “|” 来分配多个memory segment。如图中，实际会先使用 F2S_RAM， 不够才会使用RAM。当然可以通过改变顺序来优先使用 (RAM | F2S_RAM)

- 注意，堆栈仍仅指向RAM
- 因为堆栈会不停地被访问，使用FRAM会造成额外的功耗

修改CCS的 Low level Init 函数

- 将 system_pre_init.c 添加到项目中。用 **_system_pre_init()** 函数来使能FRAM的写操作

```
int _system_pre_init(void)
{
        /* Insert your low-level initializations here */

        /* Disable Watchdog timer to prevent reset during */
        /* long variable initialization sequences. */
        WDTCTL = WDTPW | WDTHOLD;
        SYSCFG0 &= ~DFWP;    // Data FRAM write enable
//      SYSCFG0 &= ~PFWP;    // Program FRAM write enable

        /*=================================*/
        /* Choose if segment initialization */
        /* should be done or not.          */
        /* Return: 0 to omit initialization */
        /* 1 to run initialization         */
        /*=================================*/
        return 1;
}
```

FRAM的写使能操作会在调用main函数前完成。在这种情况下应该小心不要在程序中对FRAM进行意外写操作

完成以上三个步骤，就可以将部分的FRAM作为RAM来使用了。得益于FRAM，我们可以将SRAM的size在不增加成本的情况下有所增加

修改IAR的linker文件

- IAR的项目会共享C:\<IAR installation directory>\430\config\linker文件夹内的同一个.xcl 文件
- 复制原来的lnk430fr4133.xcl文件并重命名为lnk430fr4133_customized.xcl.
- 分别修改information memory和RAM的所属地址范围
- 0x1900-0x19FF地址空间被从information FRAM划给了SRAM.

```
// ------------------------------------------------
// Information memory
//
-Z(CONST)INFO=1800-19FF
-Z(CONST)INFOA=1800-19FF

// ------------------------------------------------
// RAM memory
//
-Z(DATA)DATA16_I,DATA16_Z,DATA16_N,TLS16_I=2000-27FF
-Z(DATA)CODE_I
-Z(DATA)DATA20_I,DATA20_Z,DATA20_N
-Z(DATA)CSTACK+_STACK_SIZE#
```

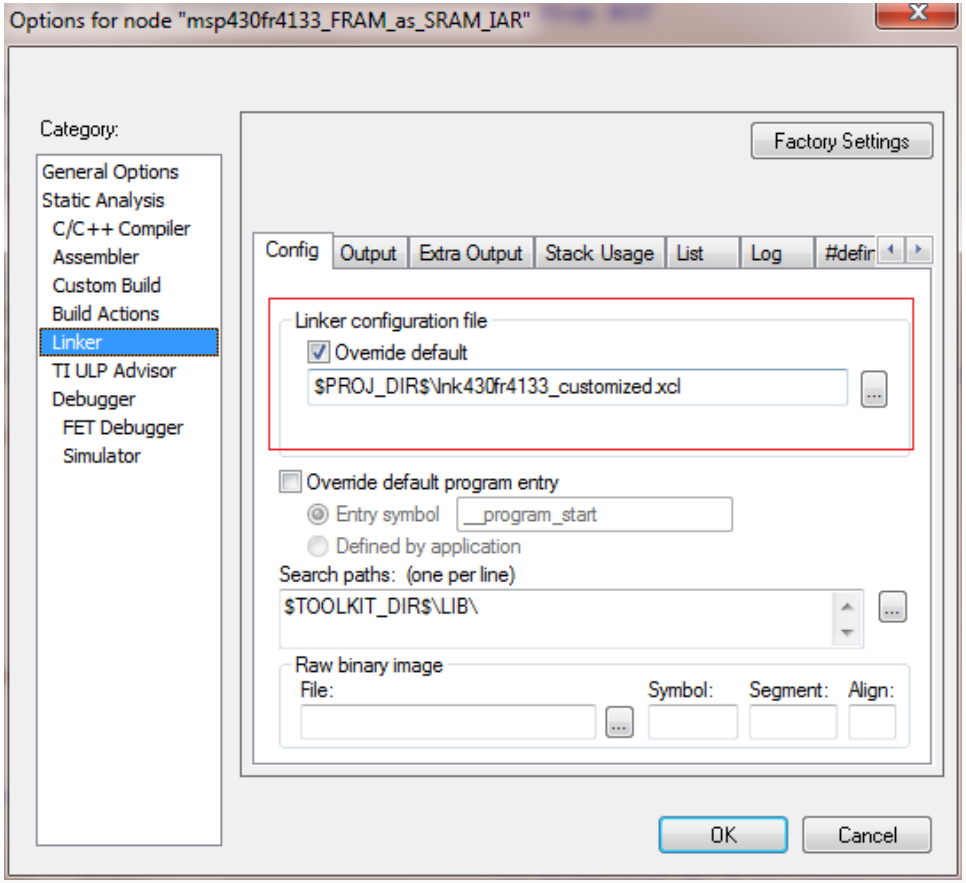
```
// ------------------------------------------------
// Information memory
//
-Z(CONST)INFO=1800-18FF
-Z(CONST)INFOA=1800-18FF

// ------------------------------------------------
// RAM memory
//
-Z(DATA)DATA16_I,DATA16_Z,DATA16_N,TLS16_I=1900-19FF,2000-27FF
-Z(DATA)CODE_I
-Z(DATA)DATA20_I,DATA20_Z,DATA20_N
-Z(DATA)CSTACK+_STACK_SIZE#
```

指向自定义的linker文件

- 将lnk430fr4133_customized.xcl添加到项目中.
- 如图，将lnk430fr4133_customized.xcl指定为需要使用的linker文件

  Options -> Linker -> Config -> Linker configuration file .



不使用默认的linker文件，指定lnk430fr4133_customized.xcl作为linker文件

使用 $PROJ_DIR$可确保你在复制项目到别处后仍可有效使用

修改IAR中的Low level Init 函数

- 在IAR环境下, low level initialization函数是**low_level_init.c** 文件的__low_level_init()
- Enable FRAM write in 在___low_level_init () 函数中使能FRAM写操作，并将low_level_init.c 文件添加到项目中

```
#include <intrinsics.h>
#include "msp430.h"

int __low_level_init(void)
{
  /* Insert your low-level initializations here */
  WDTCTL = WDTPW | WDTHOLD;
  SYSCFG0 &= ~DFWP;     //Data FRAM write enable
//SYSCFG0 &= ~PFWP;     //Program FRAM write enable
  /*
   * Return value:
   *
   *  1 - Perform data segment initialization.
   *  0 - Skip data segment initialization.
   */

  return 1;
}
```

FRAM写操作会在调用main()函数前被使能。应用程序要比较小心不要再程序中意外修改了不应修改的FRAM.

完成以上三个步骤，就可以将部分的FRAM作为RAM来使用了。得益于FRAM，我们可以将SRAM的size在不增加成本的情况下有所增加