LPC: How to place data/function/file in specified memory under

MCUXpresso IDE

Contents

During MCU development, placing data, function, and file in the specified memory address according to actual requirements is important for the memory usage. We Combine customer's frequent ask questions, explain how to operate these features step by step.

1. The default storage address space of code and data

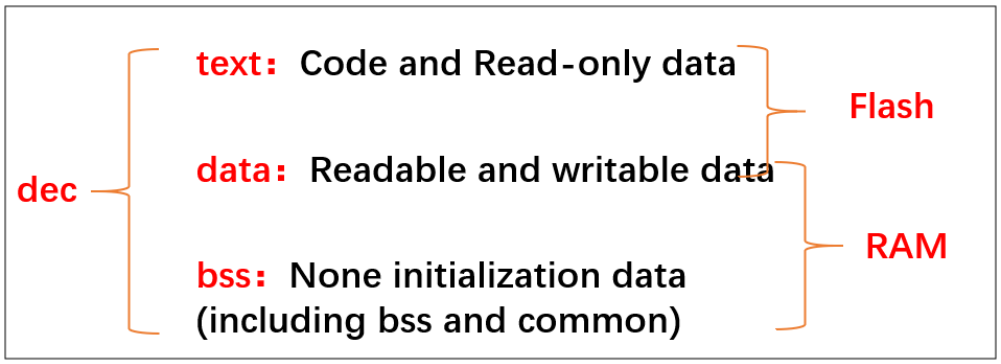
Take the hello world demo in LPC54628 as an example, and the development environment: MCUXpresso IDE.

After building, the memory allocation is as shown in the following console window:

```
Building target: lpcxpresso54628_2_hello_world.axf
Invoking: MCU Linker
arm-none-eabi-gcc -nostdlib -L"C:\shang\work\nxp\lpc54628\lpcxpresso54628_2_
Memory region         Used Size  Region Size  %age Used
  PROGRAM_FLASH:       10848 B        512 KB      2.07%
     BOARD_FLASH:          0 GB         16 MB      0.00%
   SRAM_UPPER:         8472 B        160 KB      5.17%
           SRAMX:          0 GB         32 KB      0.00%
         USB_RAM:          0 GB          8 KB      0.00%
     BOARD_SDRAM:          0 GB         16 MB      0.00%
Finished building target: lpcxpresso54628_2_hello_world.axf

make --no-print-directory post-build
Performing post-build steps
arm-none-eabi-size "lpcxpresso54628_2_hello_world.axf"; # arm-none-eabi-obj
   text    data     bss     dec     hex filename
  10844       4    8468   19316    4b74 lpcxpresso54628_2_hello_world.axf
```

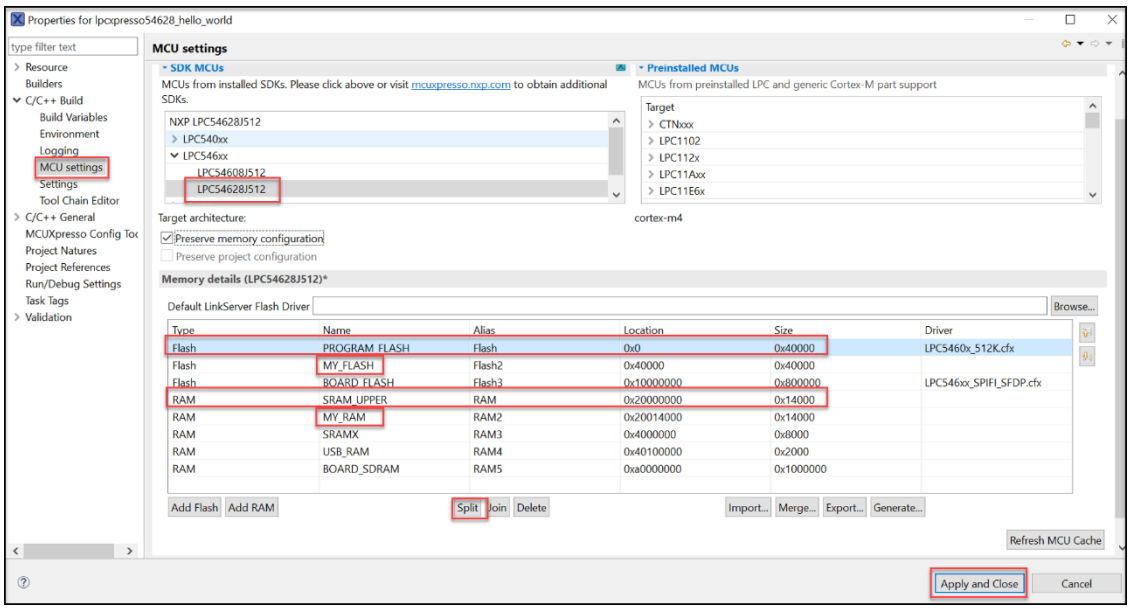
The relationship between .text, .data, .bss, .dec and Flash and RAM is as follows:



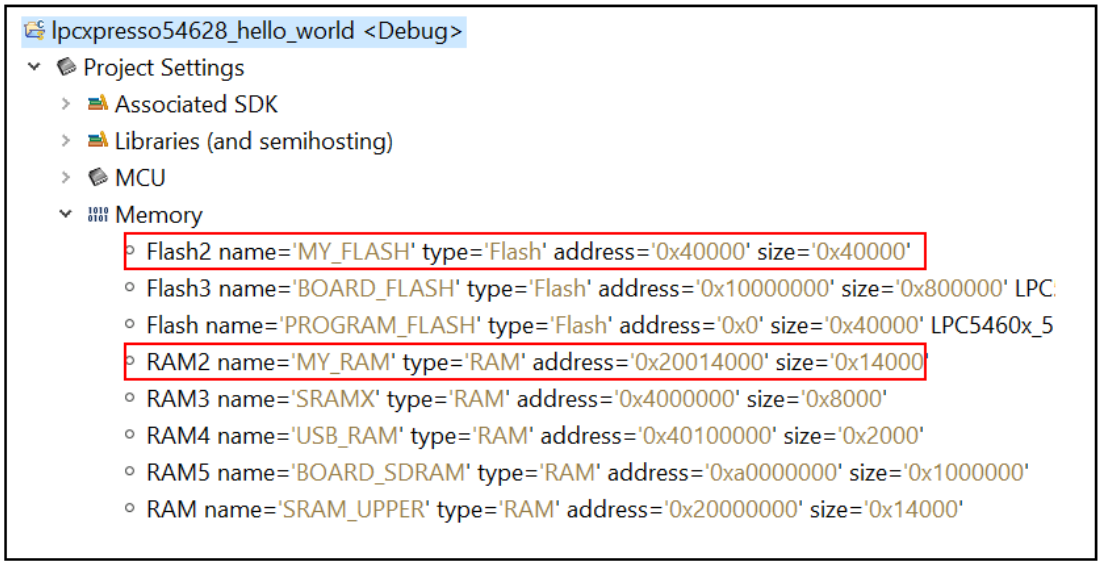
2. Customize Flash and RAM partitions

In order to place the data, function or file in the specified address space, we split some new partitions.

Open the project property setting interface, and split MY_FLASH and MY_RAM address spaces in the MCU settings option for testing. The size of these two address spaces can be customized, as follows:



After configuring Flash and RAM, click 'Apply and Close' button and you will see Flash2 and RAM2 in the project column, as follows:



3. Place the data in the specified address space

1) The default storage address space of variables and constants

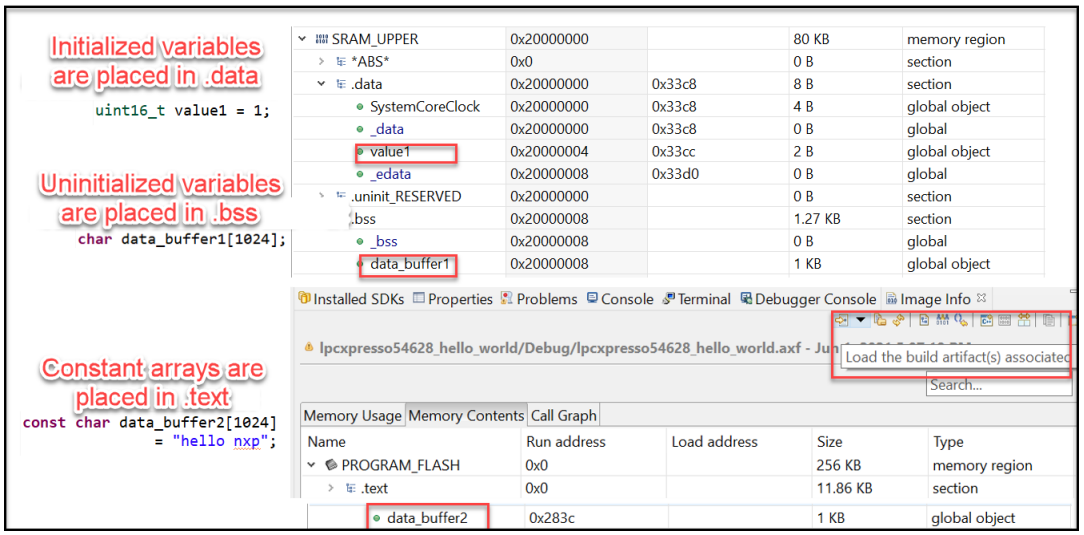
View the default address space of variables and arrays, as follows:

Initialized variable: `uint16_t value1 = 1;`

Uninitialized array: `char data_buffer1[1024];`

Constant array: `const char data_buffer2[1024] = "hello nxp";`

View storage address space of arrays using the Image Info window in MCUXpresso IDE, as follows:



Readable and writable variables and arrays are stored in RAM (0x20000000-0x20014000) named "SRAM_UPPER" by default, and const arrays are stored in Flash (0x0-0x40000) named "PROGRAM_FLASH".

2） Place the specified variables and constants in the specified address space

To place the array in custom Flash and RAM, you need to call the C language:

```
__attribute__ ((section(#type #bank)))
```

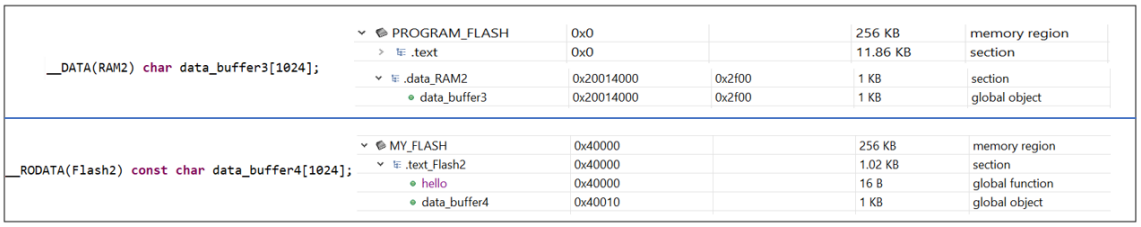
For example, place the data in .text of Flash2:

```
__attribute__ ((section("text_Flash2" ".$Flash2")))
```
+ data declaration

The NXP official has encapsulated this and defined it in cr_section_macros.h. __DATA(RAM2) means that the readable and writable array is placed into the .data section of RAM2, and __RODATA(Flash2) means that the read-only array is placed into the .rodata section of Flash2.

```
__DATA(RAM2) char data_buffer3[1024];
__RODATA(Flash2) const char data_buffer4[1024] = "hello nxp";
```

Note that you must `#include "cr_section_macros.h"`.



Global variables and arrays are placed in custom RAM2 (0x20014000-0x20028000) named "MY_RAM", and const arrays are placed in custom Flash2 (0x40000-0x80000) named "MY_FLASH".

4. Place the function in the specified address space

1） The default storage address space of functions

The code is placed in the Flash (0x0-0x40000) named "PROGRAM_FLASH" by default, and the following function is defined:

```
int hello1(void)
{
    return 1;
}
```

2） Place the specified function in the specified address space

To place the function in custom Flash, you need to call the C language:

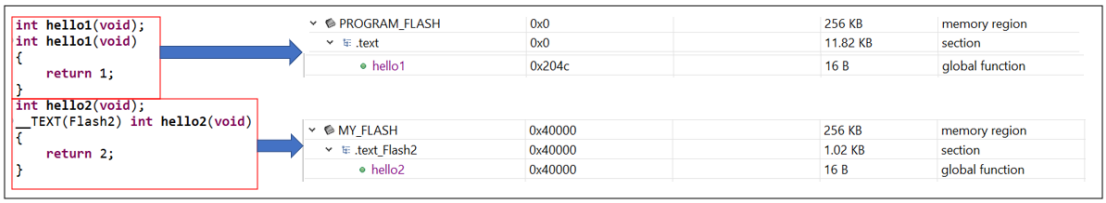
```
__attribute__ ((section(#type #bank)))
```

For example, place the function in .text of Flash2:

```
__attribute__ ((section("text_Flash2" ".$Flash2")))
```
+function declaration

The NXP official has also encapsulated this and defined it in cr_section_macros.h. The method to change the address space of the function is as follows, and place the function in a custom Flash named "MY_FLASH" (0x40000-0x80000).

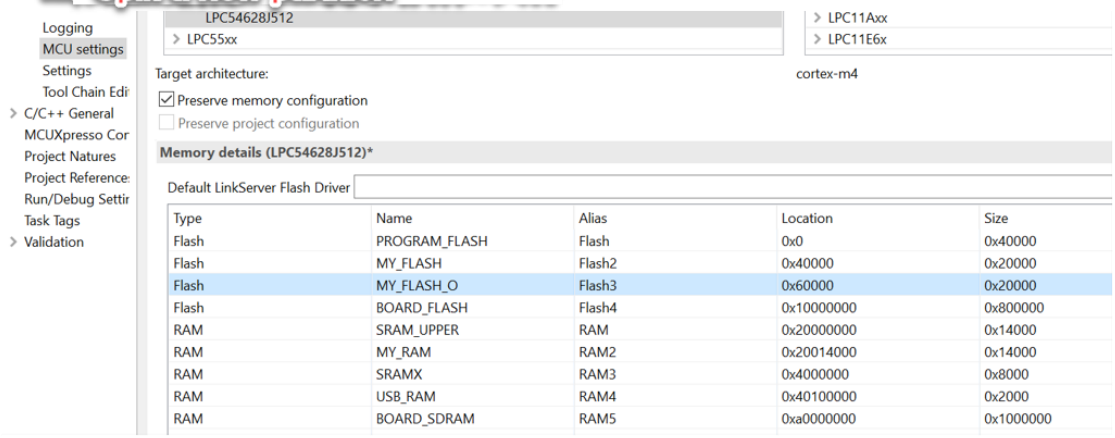
```
__TEXT(Flash2) int hello2(void)
{
    return 2;
}
```



5. Place the specified file in the specified address space

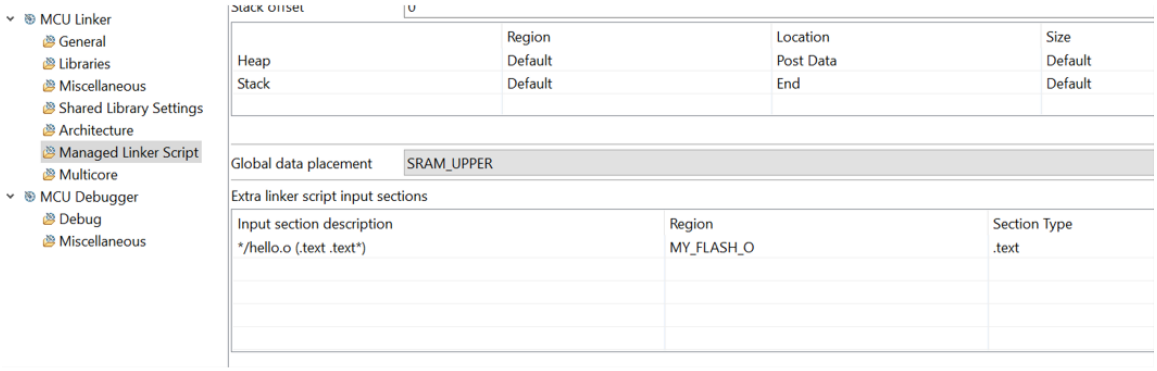
When there are many functions that need to be placed in the specified Flash, it is a little clumsy to use the __TEXT(Flash) method to set each function. If you need to place all the functions in the c file in the specified Flash, you only need to place the compiled .o file in the specified Flash. Split a new partition named "MY_FLASH_O" , create a new hello.c under the source folder, compile and generate hello.o, and configure Linker Script to place hello.o in the partitioned Flash, as follows:

1. Split a new partition

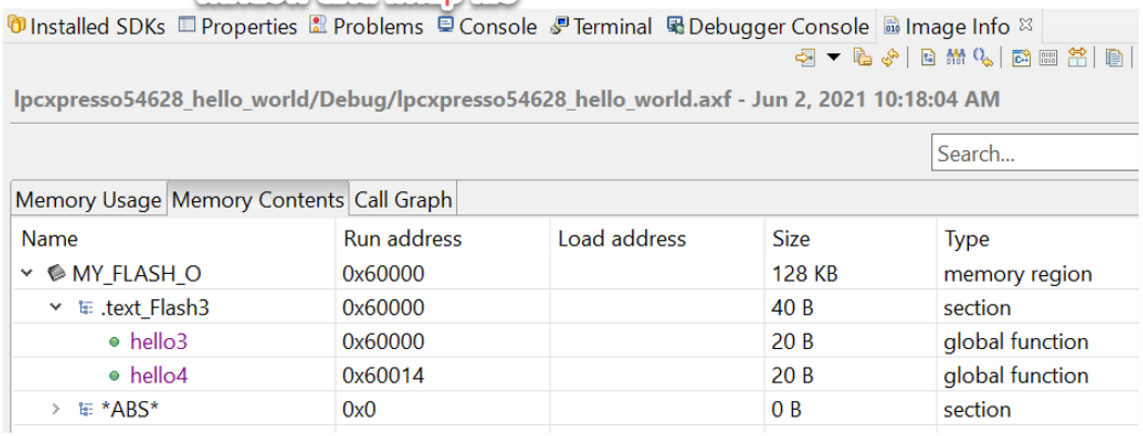


2. Create a new .c file and compile it

3. Configure Linker Script



4. View the result using Image Info window and .map file



lpcxpresso54628_hello_world.map

```
1867
1868 .text_Flash3    0x00060000        0x28
1869  FILL mask 0xff
1870  */hello.o(SORT_BY_ALIGNMENT(.text) SORT_BY_ALIGNMENT(.text*))
1871  .text.hello3   0x00060000        0x14 ./source/hello.o
1872                 0x00060000                hello3
1873  .text.hello4   0x00060014        0x14 ./source/hello.o
1874                 0x00060014                hello4
```

Reference:
https://mcuoneclipse.com/2021/05/26/placing-code-in-sections-with-managed-gnu-linker-scripts/
Relocating Code and Data Using the CW GCC Linker File for Kinetis .pdf