Програмування мікроконтролерів

ЛЕКЦІЯ 8

Робота зі scatter-файлом і мапування пам'яті

RAM та ROM

	ROM	RAM
Час доступу	Не залежить від адреси комірки;	Тим більший, чим більша адреса комірки (послідовний доступ)
Енергонезалежність	Так (при вимкненні живлення дані зберігаються)	Ні (при вимкненні живлення дані втрачаються)
Швидкість	Висока	Низька
Читання/запис	І читання, і запис	Лише читання; пам'ять програмується одноразово або ж дуже рідко
Різновиди	SRAM і DRAM	PROM EPROM EEPROM

SRAM та DRAM

SRAM	DRAM
+ вища швидкість	
+ менше енергоспоживання	
+ простіша схема	
Для 1 біт даних – 6 транзисторів	Для 1 біт даних – 1 транзистор + 1 конденсатор
	+ ціна

Мапування пам'яті

Мапування пам'яті — відповідність між ресурсами та адресами комірок пам'яті, у яких вони розміщені.

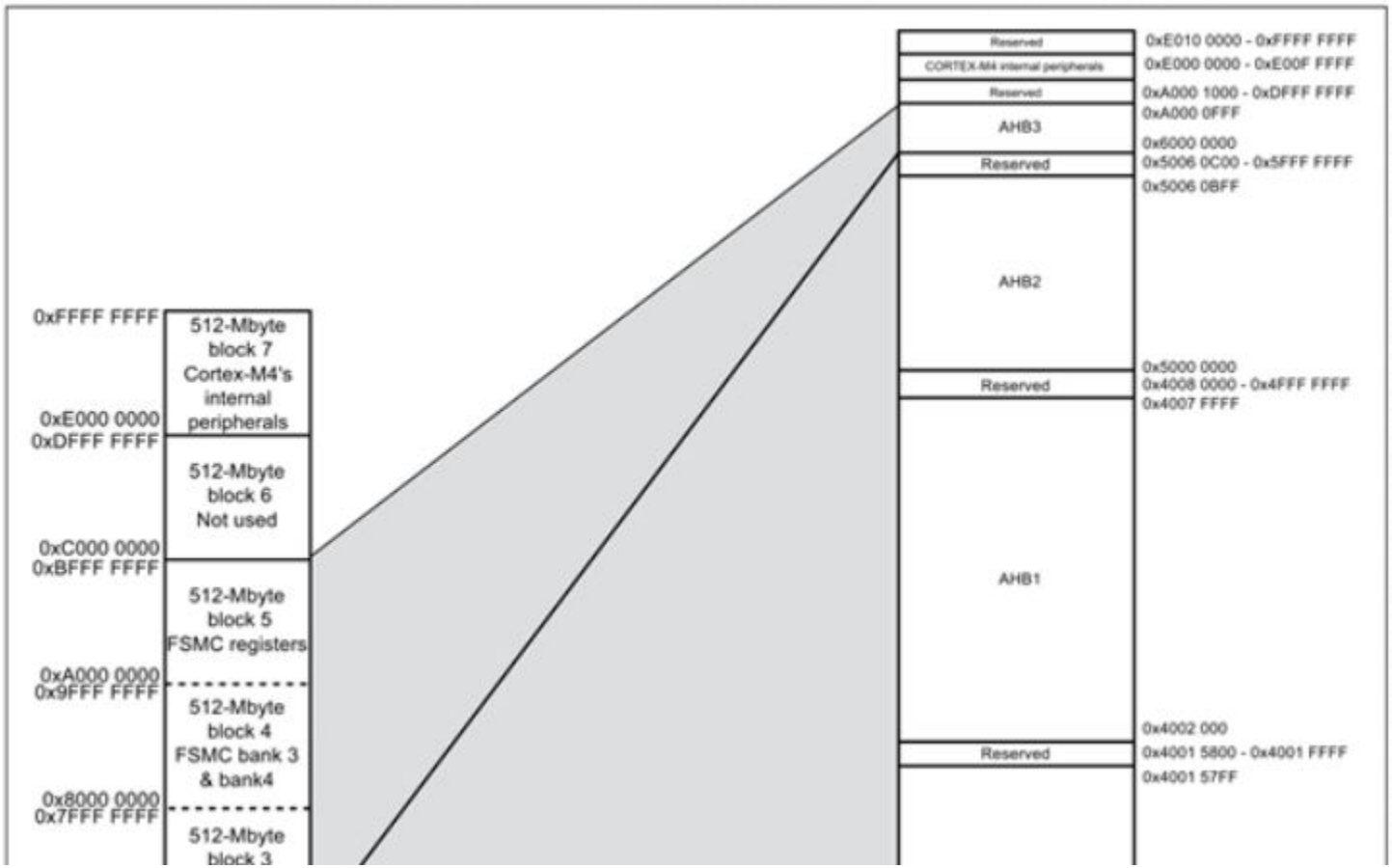
stm32f407vg

1 Мбайт

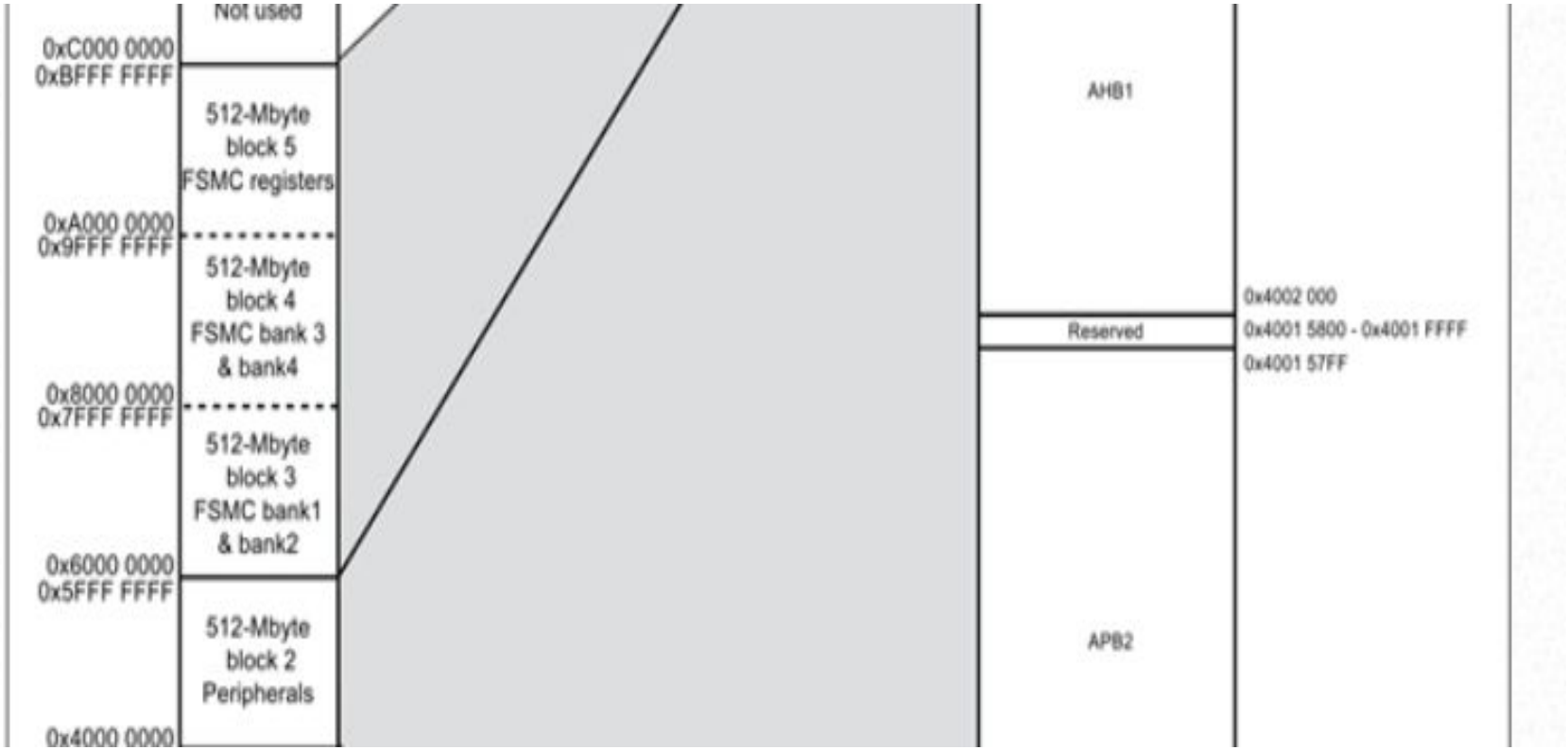
Поділена на сектори та підсектори

Сектори:
2 по 32 КБ
Далі - 64 кБ
Решта по 128 КБ

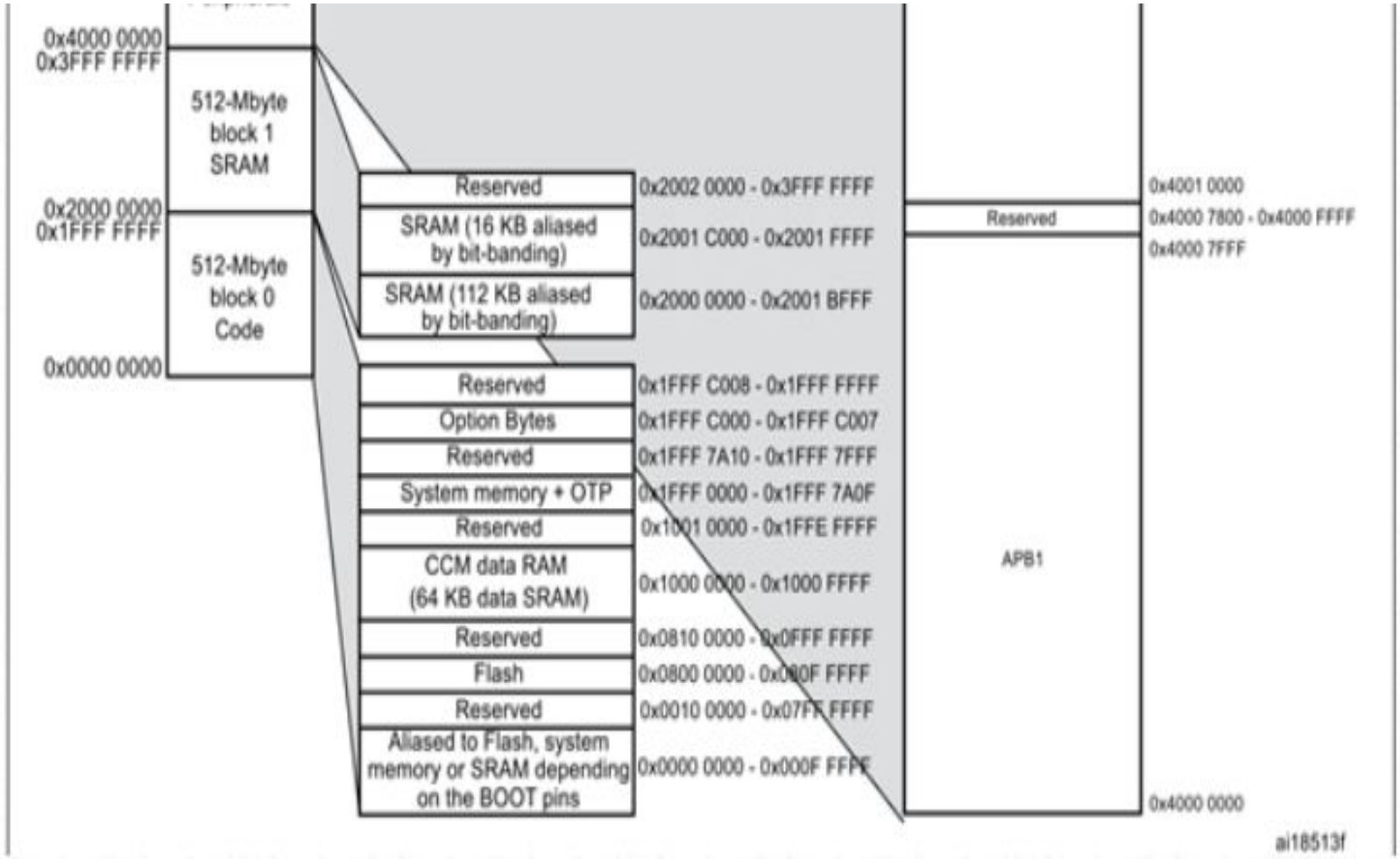
Мапування пам'яті (1)



Мапування пам'яті (2)

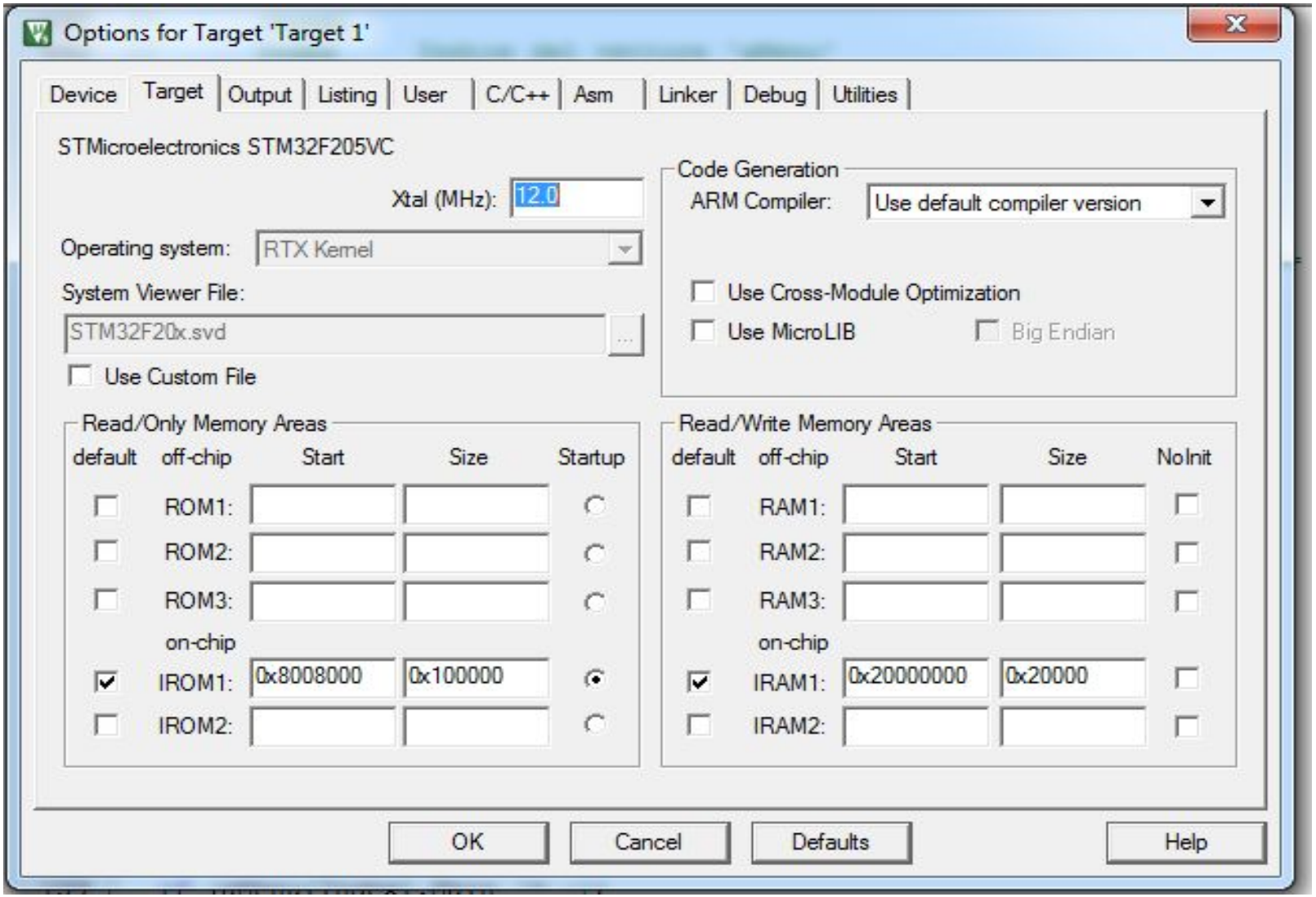


Мапування пам'яті (3)



1 МВ = 1048576 байт; 32КБ = 32768 байт

Налаштування адрес у Keil uVision



Структура map-файлу (1)

```
hal_arm.o(.emb_text) refers to rt_system.o(i.os_sys_manager) for os_sys_manager
hal_arm.o(.emb_text) refers to SWI_Table.o(SWI_TABLE) for SWI_Table
hal_arm.o(i.__SWI_0) refers to hal_arm.o(i.os_stk_check) for os_stk_check
hal_arm.o(i.__SWI_0) refers to rtx_config.o(.text) for tsk_unlock
hal_arm.o(i.__SWI_0) refers to hal_arm.o(.emb_text) for os_switch_tasks_ret
hal_arm.o(i.__SWI_0) refers to rt_task.o(.data) for os_runtask
hal_arm.o(i.os_init_stack) refers to rtx_config.o(.constdata) for os_stackinfo
hal_arm.o(i.os_stk_check) refers to rtx_config.o(.text) for os_error
hal_arm.o(i.os_stk_check) refers to rt_task.o(.data) for os_runtask
```

```
Checking duplicate symbol definitions

================================================================================

Removing Unused input sections from the image.

    Removing sn.o(.text), (26 bytes).
    Removing univio.o(.constdata), (2 bytes).
    Removing lpc24xx_pincfg.o(.constdata), (296 bytes).
    Removing rtx_can.o(.text), (592 bytes).
    Removing rtx_can.o(.bss), (844 bytes).
    Removing rtx_can.o(.data), (12 bytes).
    Removing can_lpc23xx.o(.text), (1160 bytes).
    Removing can_lpc23xx.o(.text), (272 bytes).
    Removing can_lpc23xx.o(.constdata), (68 bytes).
    Removing can_lpc23xx.o(.data), (12 bytes).
    Removing c2374.o(.text), (3784 bytes).
    Removing c2374.o(.bss), (20 bytes).
```

Структура map-файлу (2)

```
Adding Veneers to the image

    Adding TA veneer (4 bytes, Inline) for call to '_printf_percent' from __printf_flags_ss_wp.o(.text).
    Adding AT veneer (8 bytes, Inline) for call to '__rt_lib_init' from rtentry2.o(.ARM.Collect$$rtentry$$0000000A).
    Adding TA veneer (4 bytes, Inline) for call to 'SetTextFgColor' from auxfunct.o(.text).
```

```
Stack Usage for IRMCommandTask 0x0 bytes.
Stack Usage for NIR_Exec_Cmd 0x78 bytes.
Stack Usage for NIR_InitPort 0x18 bytes.
Stack Usage for NIR_Init 0x10 bytes.
Stack Usage for NIR_Acquire 0x28 bytes.
Stack Usage for NIR_GetInfo 0x0 bytes.
Stack Usage for NIR_GetTemp 0x8 bytes.
```

```
Potential Stack Usage Inaccuracies.

Functions with no stack size information.

 * Reset_Handler
 * os_sys_manager_ret
 * os_switch_tasks_ret
 * __user_initial_stackheap
 * Undef_Handler
 * PAbt_Handler
 * DAbt_Handler
 * IRQ_Handler
 * FIQ_Handler
 * Switch_task

Mutually recursive functions.

 * Undef_Handler => Undef_Handler
 * PAbt_Handler => PAbt_Handler
 * DAbt_Handler => DAbt_Handler
 * IRQ_Handler => IRQ_Handler
 * FIQ_Handler => FIQ_Handler
```

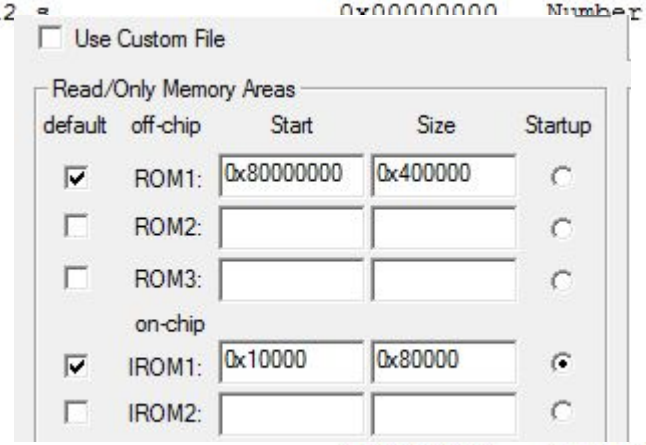
Структура map-файлу (3)

```
Image Symbol Table

    Local Symbols

    Symbol Name                                Value     Ov Type        Size  Object(Section)

    ../../angel/boardlib.s                     0x00000000   Number         0  boardinit1.o ABSOLUTE
    ../../angel/boardlib.s                     0x00000000   Number         0  boardinit2.o ABSOLUTE
    ../../angel/boardlib.s                     0x00000000   Number         0  boardinit3.o ABSOLUTE
    ../../angel/dczerorl2.s                    0x00000000   Number         0  __dczerorl2.o ABSOLUTE
```




```
    RESET                                      0x00010010   Section      860  lpc2400.o(RESET)
    Undef_Handler                              0x00010050   ARM Code       4  lpc2400.o(RESET)
    PAbt_Handler                               0x00010054   ARM Code       4  lpc2400.o(RESET)
    DAbt_Handler                               0x00010058   ARM Code       4  lpc2400.o(RESET)
    IRQ_Handler                                0x0001005c   ARM Code       4  lpc2400.o(RESET)
    FIQ_Handler                                0x00010060   ARM Code       4  lpc2400.o(RESET)
    !!!main                                    0x0001036c   Section        8  __main.o(!!!main)
    !!!scatter                                 0x00010374   Section       60  __scatter.o(!!!scatter)
    !!dczerorl2                                0x000103b0   Section       92  __dczerorl2.o(!!dczerorl2)
    _decompress_loop                           0x000103bb   Thumb Code     0  __dczerorl2.o(!!dczerorl2)
    _decompress_zerolen                        0x000103c9   Thumb Code     0  __dczerorl2.o(!!dczerorl2)
    _decompress_litn                           0x000103d1   Thumb Code     0  __dczerorl2.o(!!dczerorl2)
```

Структура map-файлу (4)

```
Memory Map of the image

  Image Entry point : 0x00010010

  Load Region LR_IROM1 (Base: 0x00010010, Size: 0x0003b524, Max: 0x0007fff0, ABSOLUTE, COMPRESSED[0x00035b30])

    Execution Region ER_IROM1 (Base: 0x00010010, Size: 0x00035950, Max: 0x0007fff0, ABSOLUTE)

    Base Addr    Size         Type   Attr      Idx    E Section Name        Object

    0x00010010   0x0000035c   Code   RO           3  * RESET               lpc2400.o
    0x0001036c   0x00000008   Code   RO        3294  * !!!main             c_t.l(__main.o)
    0x00010374   0x0000003c   Code   RO        3860    !!!scatter          c_t.l(__scatter.o)
    0x000103b0   0x0000005c   Code   RO        3858    !!dczerorl2         c_t.l(__dczerorl2.o)
    0x0001040c   0x0000002c   Code   RO        3862    !!handler_zi        c_t.l(__scatter_zi.o)
    0x00010438   0x00000004   Ven    RO        3197    .ARM.Collect$$_printf_percent$$00000000  c_t.l(_printf_percent
    0x0001043c   0x00000000   Code   RO        3197    .ARM.Collect$$_printf_percent$$00000000  c_t.l(_printf_percent
    0x0001043c   0x00000008   Code   RO        3196    .ARM.Collect$$_printf_percent$$00000003  c_t.l(_printf_f.o)
    0x00010444   0x00000008   Code   RO        3193    .ARM.Collect$$_printf_percent$$00000008  c_t.l(_printf_i.o)
    0x0001044c   0x00000008   Code   RO        3194    .ARM.Collect$$_printf_percent$$00000009  c_t.l(_printf_d.o)
    0x00010454   0x00000008   Code   RO        3195    .ARM.Collect$$_printf_percent$$0000000A  c_t.l(_printf_u.o)
    0x0001045c   0x00000008   Code   RO        3192    .ARM.Collect$$_printf_percent$$0000000B  c_t.l(_printf_o.o)
```

Структура map-файлу (5)

```
0x4000ab84   0x00000030   Zero   RW          2448    .bss               RTX_ARM_L.LIB(rt_task.o)
0x4000abb4   0x00000030   Zero   RW          3022    .bss               RTX_ARM_L.LIB(rt_list.o)
0x4000abe4   0x00000060   Zero   RW          3117    .bss               c_t.l(libspace.o)
0x4000ac44   0x00000050   Zero   RW          3222    .bss               c_t.l(stdio_streams.o)
0x4000ac94   0x00000050   Zero   RW          3223    .bss               c_t.l(stdio_streams.o)
0x4000ace4   0x00000050   Zero   RW          3224    .bss               c_t.l(stdio_streams.o)
0x4000ad34   0x00000004   PAD
0x4000ad38   0x00001000   Zero   RW             2    HEAP               lpc2400.o
0x4000bd38   0x00000740   Zero   RW             1    STACK              lpc2400.o


Execution Region RW_RAM1 (Base: 0xa0000000, Size: 0x00000000, Max: 0x00400000, ABSOLUTE)

**** No section assigned to this execution region ****


Execution Region RW_RAM2 (Base: 0xa0400000, Size: 0x000056f4, Max: 0x00100000, ABSOLUTE, COMPRESSED[0x000000b0])

Base Addr    Size         Type   Attr      Idx    E Section Name        Object

0xa0400000   0x000056f4   Data   RW         2086    BUFFERS SECT        vars.o
  ================================================================================


      Code (inc. data)   RO Data    RW Data    ZI Data      Debug

    207760      16250      11728      23508      58532     720114   Grand Totals
    207760      16250      11728        480      58532     720114   ELF Image Totals (compressed)
    207760      16250      11728        480          0          0   ROM Totals

  ================================================================================

    Total RO  Size (Code + RO Data)                219488 ( 214.34kB)
    Total RW  Size (RW Data + ZI Data)              82040 (  80.12kB)
    Total ROM Size (Code + RO Data + RW Data)      219968 ( 214.81kB)

  ================================================================================
```

Підходи до мапування пам’яті

Розміщення за певними потрібними адресами у пам'яті

To place a function or variable at a specific address it must be placed in its own section. There are several ways to do this:

- Place the function or data item in its own source file.
- Use `__attribute__((at(address)))` to place variables in a separate section at a specific address.
- Use `__attribute__((section("name")))` to place functions and variables in a named section.
- Use the `AREA` directive from assembly language. In assembly code, the smallest locatable unit is an `AREA`.
- Use the `--split_sections` compiler option to generate one ELF section for each function in the source file.
  This option results in a small increase in code size for some functions because it reduces the potential for sharing addresses, data, and string literals between functions. However, this can help to reduce the final image size overall by enabling the linker to remove unused functions when you specify `armlink --remove`.

Приклади розміщення в пам’яті

int variable __attribute__((section("foo"))) = 10;

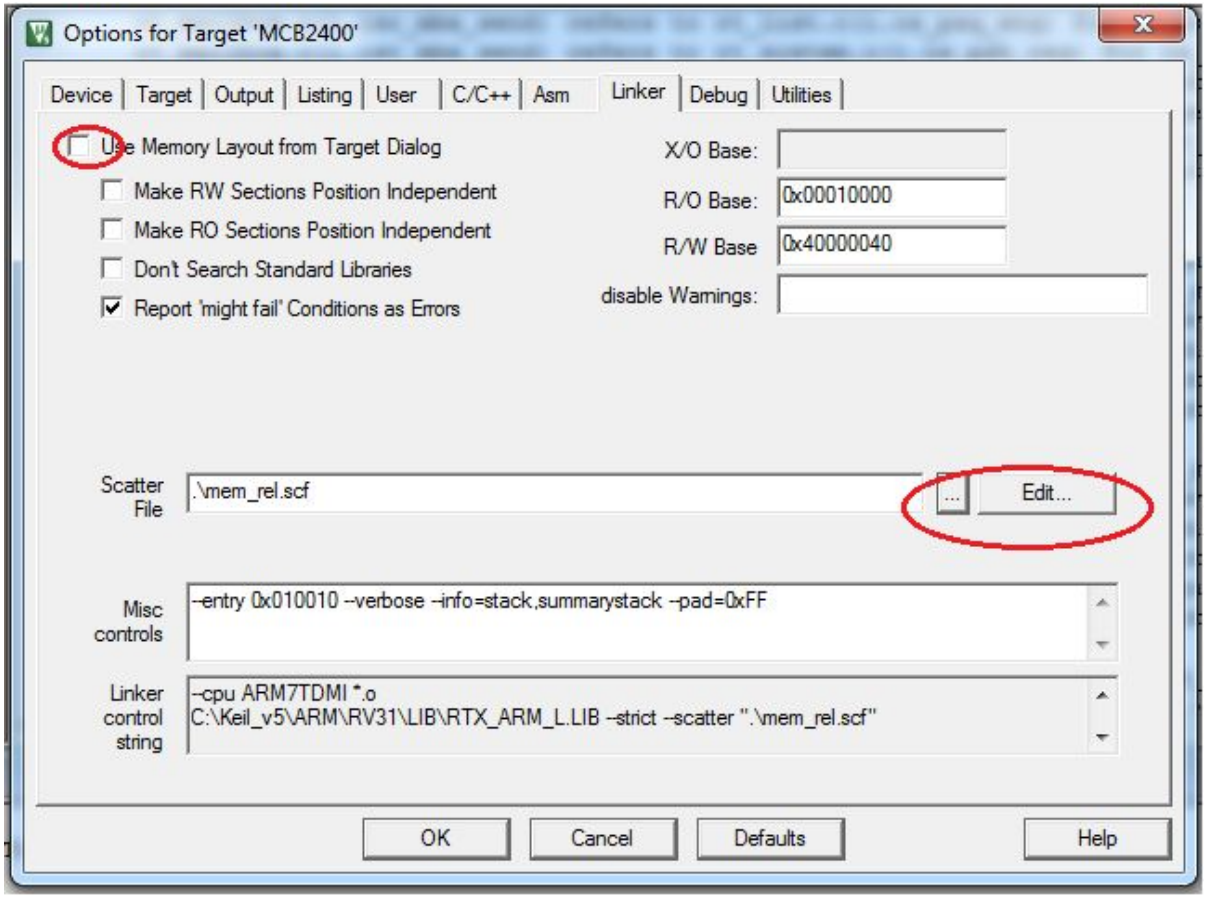
```
FLASH 0x24000000 0x4000000
{
    …                                   ; rest of code
    ADDER 0x08000000
    {
        file.o (foo)                     ; select section foo from file.o
    }
}
```

```
// place variable1 in a section called .ARM.__AT_0x00008000
int variable1 __attribute__((at(0x8000))) = 10;
// place variable2 in a section called .ARM.__at_0x8000
int variable2 __attribute__((section(".ARM.__at_0x8000"))) = 10;
```

Scatter-файл

англ. Scatter — розкидати

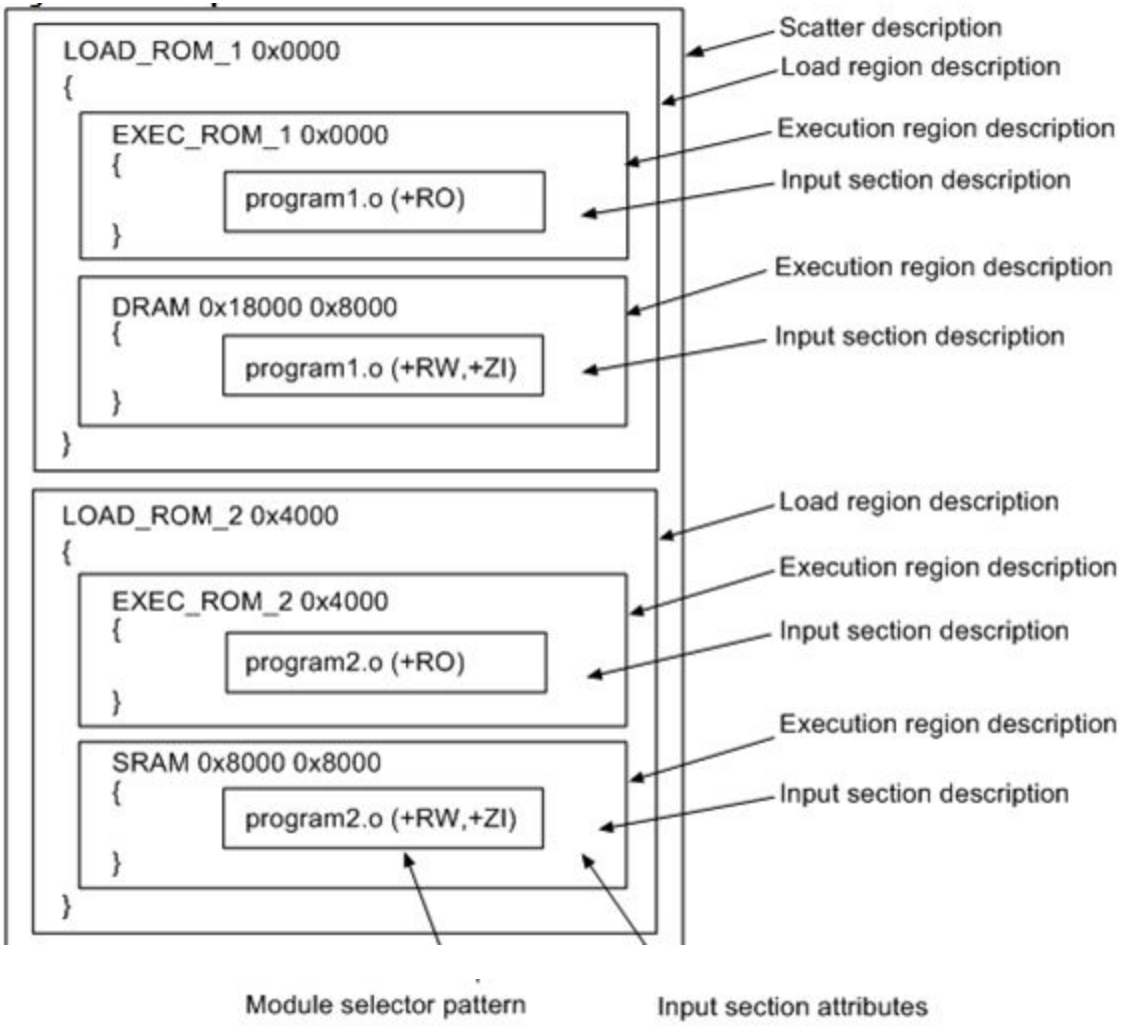
розширення — .sct



Синтаксис BNF

Symbol	Description
"	Quotation marks indicate that a character that is normally part of the BNF syntax is used as a literal character in the definition. The definition B"+"C, for example, can only be replaced by the pattern B+C. The definition B+C can be replaced by, for example, patterns BC, BBC, or BBBC.
A ::= B	Defines A as B. For example, A::= B"+" | C means that A is equivalent to either B+ or C. The ::= notation defines a higher level construct in terms of its components. Each component might also have a ::= definition that defines it in terms of even simpler components. For example, A::= B and B::= C | D means that the definition A is equivalent to the patterns C or D.
[A]	Optional element A. For example, A::= B[C]D means that the definition A can be expanded into either BD or BCD.
A+	Element A can have one or more occurrences. For example, A::= B+ means that the definition A can be expanded into B, BB, or BBB.
A*	Element A can have zero or more occurrences.
A | B	Either element A or B can occur, but not both.
(A B)	Element A and B are grouped together. This is particularly useful when the | operator is used or when a complex pattern is repeated. For example, A::=(B C)+ (D | E) means that the definition A can be expanded into any of BCD, BCE, BCBCD, BCBCE, BCBCBCD, or BCBCBCE.

Синтаксис scatter-файлу



Опис області load

```
load_region_description ::=
  load_region_name  (base_address  | ("+" offset)) [attribute_list] [max_size]
       "{"
          execution_region_description+
       "}"
```

Атрибути:

ABSOLUTE
ALIGN <число байт, степінь 2>
NOCOMPRESS
OVERLAY
PI
PROTECTED
RELOC

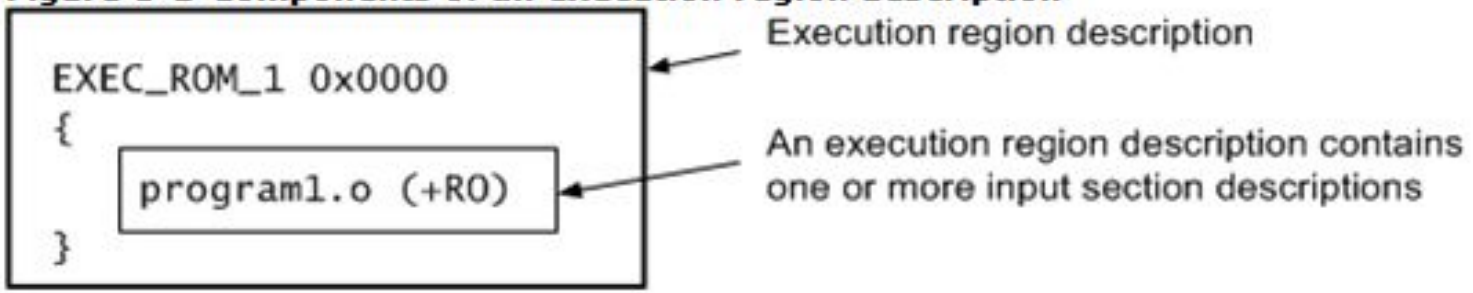
Опис області execution (1)

An execution region description has:

- A name (used by the linker to identify different execution regions).
- A base address (either absolute or relative).
- Attributes that specify the properties of the execution region.
- An optional maximum size specification.
- One or more input section descriptions (the modules placed into this execution region).

The following figure shows the components of a typical execution region description:

Figure 8-3 Components of an execution region description



Опис області execution (2)

```
execution_region_description ::=
  exec_region_name (base_address | "+" offset) [attribute_list] [max_size | length]
        "{"
            input_section_description*
        "}"
```

```
input_section_description ::=
  module_select_pattern
        [ "(" input_section_selector ( "," input_section_selector )* ")" ]  input_section_selector ::=
        ("+" input_section_attr | input_section_pattern | input_symbol_pattern | section_properties)
```

Частина scatter-файлу (3)

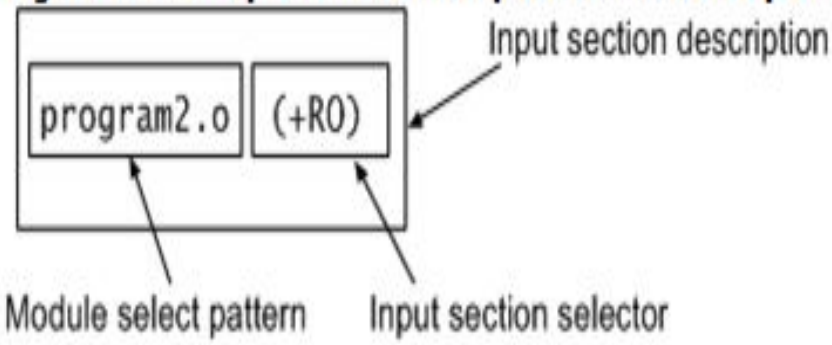
An input section description is a pattern that identifies input sections.

An input section description identifies input sections by:

- Module name (object filename, library member name, or library filename). The module name can use wildcard characters.
- Input section name, or input section attributes such as `READ-ONLY`, or `CODE`. You can use wildcard characters for the input section name.
- Symbol name.

The following figure shows the components of a typical input section description.

Figure 8-4 Components of an input section description



Зміст стандартного scatter-файлу

```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

LR_IROM1 0x08008000 0x00100000  {    ; load region size_region
  ER_IROM1 0x08008000 0x00100000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x20000000 0x00020000  {  ; RW data
   .ANY (+RW +ZI)
  }
}
```

Приклади scatter-файлів (1)

```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

LR_IROM1 0x08010000 0x001F0000  {    ; load region size_region
         image.bin 0x08010000 0x001F0000  {  ; load address = execution address
         *.o (RESET, +First)
         *(InRoot$$Sections)
         .ANY (+RO)
         }

         ;External ram 8MB
         RW_RAM1 0xD0680000 0x00080000 { ; RW data -> vars.h     (about 512 KB)
         *(BUFFERS_SECT)
         }

         RW_IRAM1 0x20000000 0x00030000  {  ; RW data
         .ANY (+RW +ZI)
         }
}
```

Приклади scatter-файлів (2)

```
LR_ROM1 0x00000000 0x00100000 {
        unicode.bin 0x00000000 0x00100000 { ; load address != execution address (about 1MB)
        Unicode.o (+RO)
        }
}

LR_ROM2 0x00000000 0x00200000 {
        irbcomponents.bin 0x00000000 0x00200000 { ; load address != execution address (about 2MB)
        Components.o (+RO)
        }
}

LR_ROM3 0x00000000 0x00100000 {
        irbsensor.bin 0x00000000 0x00100000 { ; load address != execution address (about 1MB)
        Sensor.o (+RO)
        }
}
```

Вирази у scatter-файлах

```
LR1 0x8000
{
    ER1 (defined(version1) ? 0x8000 : 0x10000)    ; Base address is 0x8000
                                                  ; if version1 is defined
                                                  ; 0x10000 if not
    {
        *(+RO)
    }
    ER2 +0
    {
        *(+RW +ZI)
    }
}
```