Съпоставяне на С структури към адреси

В програмите на С периферните модули се описват със структури.

На всеки един регистър съответства по една променлива от структурата (map).

Компилаторът разполага променливите на С структурите на последователни адреси.

Съпоставяне на С структури към адреси

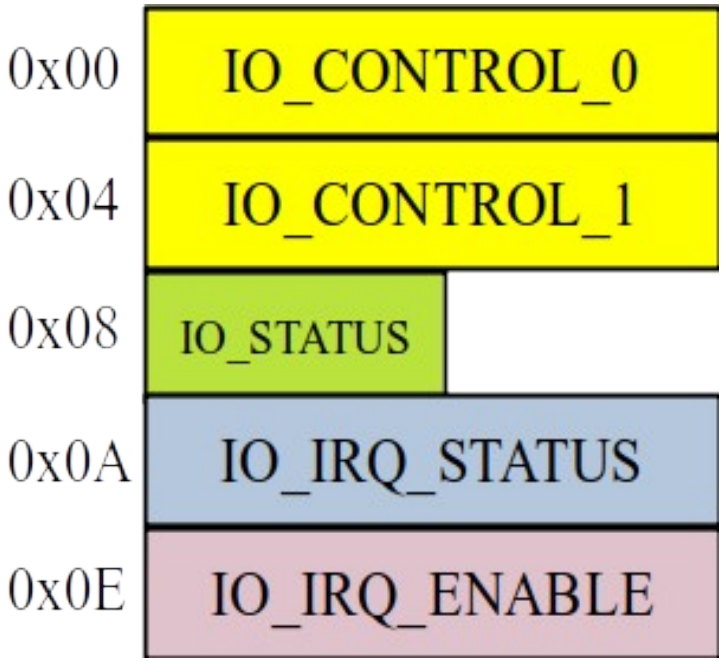
За да се съпостави софтуера към хардуера, трябва да са изпълнени условията:

*променливите са еднакви по размер с регистрите, към които ще бъдат съпоставени
*броят на променливите от С структурата е равен на броя на регистрите от хардуерния модул
*адресът на първата променлива от структурата и първия регистър от хардуерния модул съвпадат
*разполагането на данните в паметта (endianess) на μPU и генерираните променливи от компилатора трябва да съвпадат

Съпоставяне на C структури към адреси

```
typedef struct {
    uint32_t IO_CONTROL[2];
    uint16_t IO_STATUS;
    uint32_t IO_IRQ_STATUS;
    uint32_t IO_IRQ_ENABLE;
}my_io_module_t;
```

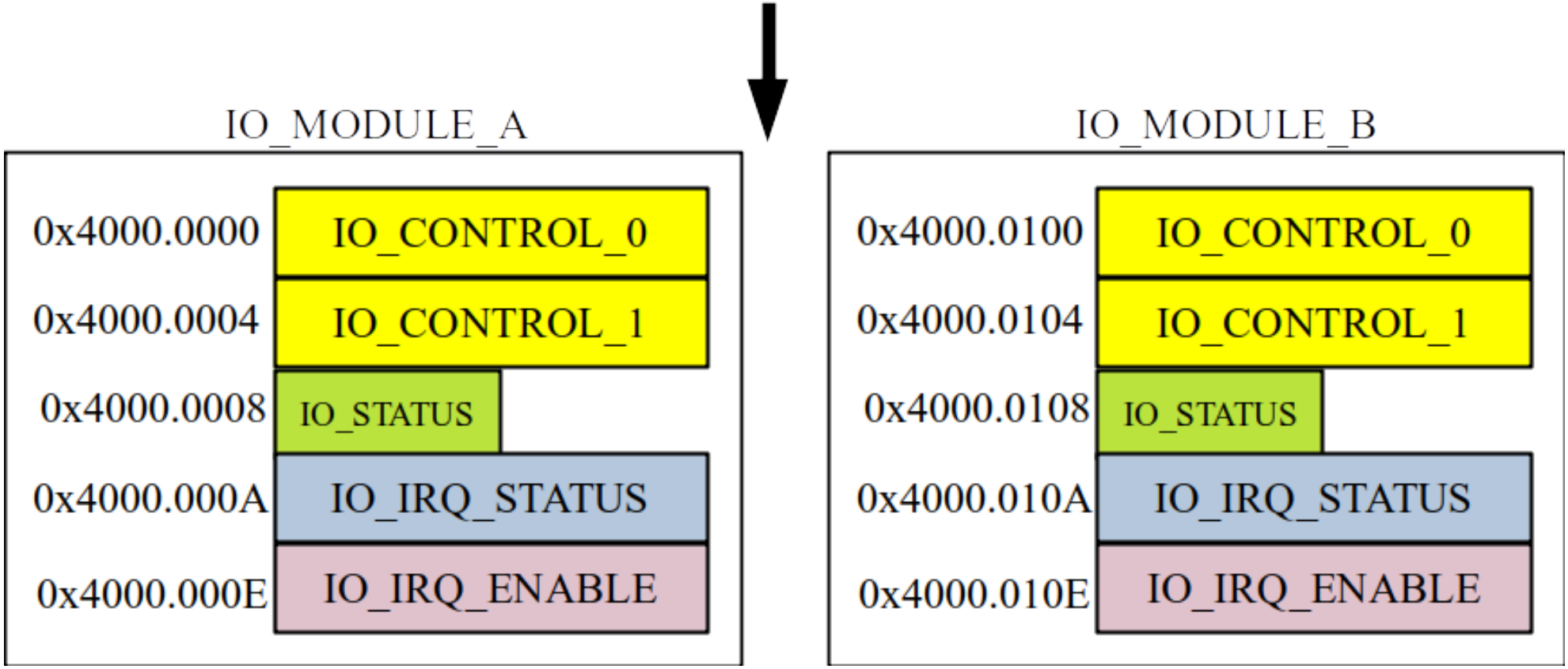
Компилатор



Съпоставяне на C структури към адреси

Съпоставяне (map) на структура към адрес

```
#define IO_A   ((my_io_module_t *)0x40000000)
#define IO_B   ((my_io_module_t *)0x40000100)
```



Съпоставяне на С структури към адреси

След съпоставянето достъпът до регистрите става по следния начин:

```
//Променяне (запис) на всички битове от контролен
//регистър 0
IO_A→IO_CONTROL[0] = 0x4350003F;

//Четене на всички битове от статус регистъра
uint32_t io_stat;
io_stat = IO_A→IO_STATUS;
```

Съпоставяне на C структури към адреси

```
//Вдигане само на един бит (#4) в логическа 1 от
//регистъра за разрешаване на прекъсванията
IO_IRQ_ENABLE = IO_IRQ_ENABLE | 0x10;

//Сваляне само на един бит (#7) в логическа 0 от
//регистъра за разрешаване на прекъсванията
IO_IRQ_ENABLE = IO_IRQ_ENABLE & ~0x80;

//Преобръщане само на един бит (#2) от контролен
//регистър 1
IO_CONTROL[1] = IO_CONTROL[1] ^ 0x04;
```

Съпоставяне на С структури към адреси

Пример – в библиотеката HAL на STM32L011 се използва съпоставяне на адреси.

В technical reference manual на STM32L011 може да бъде намерен списък с всички регистри на GPIO модулите. Адресите им се дават като отместване спрямо базов адрес, защото регистрите са еднотипни и се използват в няколко GPIO модула, всеки с уникален базов адрес.

Отместване (спрямо базов адрес)	Име на регистър
0x00	GPIOA_MODER
0x04	GPIOx_OTYPER
0x08	GPIOA_OSPEEDR
0x0C	GPIOA_PUPDR
0x10	GPIOx_IDR
0x14	GPIOx_ODR
0x18	GPIOx_BSRR
0x1C	GPIOx_LCKR
0x20	GPIOx_AFRL
0x24	GPIOx_AFRH
0x28	GPIOx_BRR

Съпоставяне на С структури към адреси

В technical reference manual на STM32L011 може да бъде намерена картата на паметта. По-долу е дадена извадка от нея.

Table 3. STM32L0x1 peripheral register boundary addresses(1)

Bus	Boundary address	Size (bytes)	Peripheral	Peripheral register map
IOPORT	0X5000 1C00 - 0X5000 1FFF	1K	GPIOH	*Section 8.4.12: GPIO register map*
	0X5000 1400 - 0X5000 1BFF	2 K	Reserved	-
	0X5000 1000 - 0X5000 13FF	1K	GPIOE	*Section 8.4.12: GPIO register map*
	0X5000 0C00 - 0X5000 0FFF	1K	GPIOD	*Section 8.4.12: GPIO register map*
	0X5000 0800 - 0X5000 0BFF	1K	GPIO C	*Section 8.4.12: GPIO register map*
	0X5000 0400 - 0X5000 07FF	1K	GPIOB	*Section 8.4.12: GPIO register map*
	0X5000 0000 - 0X5000 03FF	1K	GPIOA	*Section 8.4.12: GPIO register map*
	0X4002 6400 - 0X4002 FFFF	49 K	Reserved	-
	0X4002 6000 - 0X4002 63FF	1 K	AES (Cat. 1, 2 and 5 with AES only)	*Section 15.7.13: AES register map*
	0X4002 5400 - 0X4002 5FFF	3 K	Reserved	-
	0X4002 4400 - 0X4002 53FF	3 K	Reserved	-

Съпоставяне на C структури към адреси

stm32l011xx.h

```
==============================================================
#define PERIPH_BASE           ((uint32_t)0x40000000U)
#define IOPPERIPH_BASE        (PERIPH_BASE + 0x10000000U)
------------------------------------------------------------------------
#define GPIOA_BASE            (IOPPERIPH_BASE + 0x00000000U)
#define GPIOB_BASE            (IOPPERIPH_BASE + 0x00000400U)
#define GPIOC_BASE            (IOPPERIPH_BASE + 0x00000800U)
------------------------------------------------------------------------
#define GPIOA                 ((GPIO_TypeDef *) GPIOA_BASE)
#define GPIOB                 ((GPIO_TypeDef *) GPIOB_BASE)
#define GPIOC                 ((GPIO_TypeDef *) GPIOC_BASE)
------------------------------------------------------------------------
typedef struct{
  __IO uint32_t MODER;       /*!< GPIO port mode register,                Address offset: 0x00 */
  __IO uint32_t OTYPER;      /*!< GPIO port output type register,         Address offset: 0x04 */
  __IO uint32_t OSPEEDR;     /*!< GPIO port output speed register,        Address offset: 0x08 */
  __IO uint32_t PUPDR;       /*!< GPIO port pull-up/pull-down register,   Address offset: 0x0C */
  __IO uint32_t IDR;         /*!< GPIO port input data register,          Address offset: 0x10 */
  __IO uint32_t ODR;         /*!< GPIO port output data register,         Address offset: 0x14 */
  __IO uint32_t BSRR;        /*!< GPIO port bit set/reset registerBSRR,   Address offset: 0x18 */
  __IO uint32_t LCKR;        /*!< GPIO port configuration lock register,  Address offset: 0x1C */
  __IO uint32_t AFR[2];      /*!< GPIO alternate function register,       Address offset: 0x20-0x24 */
  __IO uint32_t BRR;         /*!< GPIO bit reset register,                Address offset: 0x28 */
}GPIO_TypeDef;
```

Съпоставяне на С структури към адреси

stm32l0xx_hal_gpio.h

==========

```
void HAL_GPIO_Init(GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_Init){
    ....

temp = GPIOx->AFR[position >> 3U];

temp &= ~((uint32_t)0xFU << ((uint32_t)(position & (uint32_t)0x07U) * 4U)) ;

temp |= ((uint32_t)(GPIO_Init->Alternate) << (((uint32_t)position & (uint32_t)0x07U) * 4U)) ;

     GPIOx->AFR[position >> 3U] = temp;
    }

    ....
}
```