ARM Coretex-M

펌웨어 설계

자가 진단 SW 설계

한국기술교육대학교
온라인평생교육원

학습목표

- STM32F429 레지스터들의 C언어 표현을 이해할 수 있다.
- 하드웨어 자가 진단 테스트를 위한 소프트웨어를 설계할 수 있다.

학습내용

- STM32F429의 레지스터 접근
- 자가 진단 SW 설계

STM32F429의 레지스터 접근

포인터를 사용한 레지스터 접근

포인터

Pointer

무언가를 가리킨다는 뜻

무엇을 가리키나?

▼

▼

▼

메모리나 레지스터의 주소

- 모든 메모리는 주소를 가짐
- 메모리는 바이트 단위로 액세스됨
  - 첫 번째 바이트의 주소는 0, 두 번째 바이트는 1,…

포인터는 메모리나 레지스터의 주소를 말함

- 모든 레지스터는 주소를 가짐
  - 단, CPU core에 내장된 레지스터들은 제외

STM32F429의 레지스터 접근

포인터를 사용한 레지스터 접근

포인터 연산자

C언어

&연산자	*연산자
• &다음에 오는 변수(메모리)나 레지스터의 주소를 반환함	• *다음에 오는 값(주소)의 내용을 반환

⋯› 5회차 강의 GPIO 제어 SW설계에서 확인된 레지스터 주소

- GPIOB_MODER레지스터의 주소는 0x40020400임

⋯› 0x40020400 주소의 내용을 확인하려면 간단하게 *(0x40020400)라고 표현할 수 있음

- 다만 *(0x40020400)만 표기하면 컴파일러가 0x40020400을 시작주소로 몇 바이트를 접근해야 하는지 알 수 없기 때문에 변수형을 지정

STM32F429의 레지스터 접근

포인터를 사용한 레지스터 접근

포인터 연산자

- 0x40020400 시작 주소 내용 확인 방법

한 바이트 내용

- *((unsigned char*)0x40020400) 이라고 표현

두 바이트 내용

- *((unsigned short*)0x40020400) 이라고 표현

네 바이트의 내용

- *((unsigned int*)0x40020400) 이라고 표현

GPIOB_MODER 레지스터는 32비트인 네 바이트 레지스터이므로 주로 *((unsigned int*)0x40020400)이라는 표현을 쓰게 됨

STM32F429의 레지스터 접근

포인터를 사용한 레지스터 접근

포인터를 사용하여 레지스터 접근하기

GPIOx_MODER 레지스터

GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

GPIOB_MODER레지스터의 MODER1[1:0]에 해당하는
2번 비트를 1, 3번 비트를 0으로 셋팅하려는 경우

▼
▼
▼

```
*((unsigned int*)0x40020400)
= 0b0100;
```

STM32F429의 레지스터 접근

포인터를 사용한 레지스터 접근

포인터를 사용하여 레지스터 접근하기

GPIOx_MODER 레지스터

GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

위의 2진수 표현을 16진수로 수정하는 경우

▼
▼
▼

```
*((unsigned int*)0x40020400)
= 0x4;
```

STM32F429의 레지스터 접근

포인터를 사용한 레지스터 접근

포인터를 사용하여 레지스터 접근하기

GPIOx_MODER 레지스터

GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

GPIOB_MODER 레지스터에 0x04값을 쓰는 C코드는 대개 매크로를 사용하는 경우

▼
▼
▼

```
#define GPIOB_MODER *((unsigned int *) 0x40020400

GPIOB_MODER = 0x04;
```

STM32F429의 레지스터 접근

포인터를 사용한 레지스터 접근

포인터를 사용하여 레지스터 접근하기

GPIOx_MODER 레지스터

GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

0x20000000 의 주소를 가지는 TEST_REG라는 이름을 가진 32비트 레지스터에 0xabcd값을 쓰는 C코드의 표현

```
#define TEST_REG *((unsigned int *) 0x20000000

TEST_REG = 0xabcd;
```

STM32F429의 레지스터 접근

포인터를 사용한 레지스터 접근

32비트 레지스터의 표현

비트 #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16 진수																																
2 진수																																

- 제일 위의 숫자는 비트 번호임. 0번 비트부터 31번 비트까지 있음
- 보통 오른쪽에서 왼쪽으로 0부터 증가하는 방향으로 비트들을 표현
- 제일 아래의 2진수는 각 비트를 0이나 1로 표현한 값
- 중간의 16진수는 2진수를 4비트씩 표기할 수 있는 속성을 이용하여 그림과 같이 표기

비트 #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16 진수	0				0				0				0				a				b				c				d			
2 진수	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	1	1	1	0	0	1	1	0	1

- 만약에 32비트 레지스터에 0xabcd라는 값을 쓴다면 표를 이용하여 각 비트별 값을 쉽게 확인할 수 있음

자가 진단 SW 설계

STM32 라이브러리의 레지스터 표현

STM32 라이브러리

- STM32 시리즈의 펌웨어 라이브러리를 STM사에서 기본 제공함
- CubeMX를 통해 생성된 코드에 STM32 라이브러리가 포함되어 있으며 무료로 자유롭게 사용할 수 있음
- STM32 라이브러리의 레지스터 표현을 분석함으로써 정확한 코딩이 가능함
- STM32 라이브러리는 레지스터 접근을 위해 C언어의 구조체, 포인터, typedef지정자를 사용함
- GPIO 관련 라이브러리 함수중 기본인 HAL_GPIO_Init()함수의 매개변수를 분석함

자가 진단 SW 설계

STM32 라이브러리의 레지스터 표현

GPIO 관련 레지스터의 라이브러리 함수

- HAL_GPIO_Init()함수의 매개변수인 GPIO_TypeDef를 분석하면 GPIO 관련 레지스터의 접근 방식을 알 수 있음

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

```
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 register,      Address offset: 0x18      */
  __IO uint32_t LCKR;     /*!< GPIO port configuration lock register, Address offset: 0x1C      */
  __IO uint32_t AFR[2];   /*!< GPIO alternate function registers,     Address offset: 0x20-0x24 */
} GPIO_TypeDef;
```

레지스터 접근 방식

- typedef지정자는 기존 자료구조를 다르게 표현할 사용

```
typedef unsigned int U32INT;

U32INT a;
```

레지스터 접근 방식

자가 진단 SW 설계

STM32 라이브러리의 레지스터 표현

GPIO 관련 레지스터의 라이브러리 함수

- 위의 예제에서 unsigned int a;로 선언할 변수를 U32INT a;로 표현할 수 있음

```
typedef struct{
  unsigned int a;
  unsigned int b;
} GPIO_TypeDef;

GPIO_TypeDef gpio;
```

레지스터 접근 방식

기존의 자료형에 새로운 이름을 붙이는 것은 큰 이점이 없으나 구조체의 같은 복잡한 형식을 사용할 때는 유용하게 사용됨

자가 진단 SW 설계

STM32 라이브러리의 레지스터 표현

GPIO 관련 레지스터의 라이브러리 함수

구조체의 멤버변수들을 분석

```
__IO uint32_t MODER;
```

```
#define    __IO    volatile
typedef unsigned int uint32_t
```

레지스터 접근 방식

- __IO 는 #define __IO volatile 으로 선언된 것으로, 즉 volatile지정자임
- C언어의 volatile지정자는 컴파일 옵션에 의한 최적화가 일어나지 않게 하는 지정자임
- 그 다음의 uint32_t는 typedef unsigned int uint32_t

```
HAL_GPIO_Init(GPIOB,  &GPIO_InitStruct);
```

```
#define GPIOB              ((GPIO_TypeDef *) GPIOB_BASE)
#define GPIOB_BASE         (AHB1PERIPH_BASE + 0x0400UL)
#define AHB1PERIPH_BASE    (PERIPH_BASE + 0x00020000UL)
#define PERIPH_BASE        0x40000000UL
```

레지스터 접근 방식

- HAL_GPIO_Init()함수는 위와 같이 사용됨

자가 진단 SW 설계

STM32 라이브러리의 레지스터 표현

GPIO 관련 레지스터의 라이브러리 함수

- 구조체의 멤버변수들을 분석

```
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

#define GPIOB              ((GPIO_TypeDef *) GPIOB_BASE)
#define GPIOB_BASE         (AHB1PERIPH_BASE + 0x0400UL)
#define AHB1PERIPH_BASE    (PERIPH_BASE + 0x00020000UL)
#define PERIPH_BASE        0x40000000UL
```

레지스터 접근 방식

매개변수 GPIOB

- GPIOB_BASE의 값을 가지는 GPIO_TypeDef 구조체 포인터형 변수임
- GPIOB는 분석해보면 결국 0x40020400의 값
- 즉, GPIOB는 0x40020400을 시작주소로 하는 GPIO관련 레지스터를 표현한 값

GPIOB, MODER

- MODER레지스터의 주소인 0x40020400
- GPIOB->MODER은 *(GPIOB.MODER)과 같은 값이므로 0x40020400의 주소에 저장된 내용을 말함

자가 진단 SW 설계

자가 진단 테스트 소프트웨어 설계

교수님 자가 진단 테스트 소프트웨어 설계 실습 영상

1. STM32F429의 GPIO 관련 레지스터의 값을 읽고 쓰면서 하드웨어 테스트

자가 진단 SW 설계

STM32 라이브러리의 레지스터 표현

GPIO관련 레지스터의 라이브러리 함수

⋯› HAL_GPIO_Init()함수의 2번째 매개변수인 GPIO_InitTypeDef는 위와 같은 구조

```
typedef struct
{
  uint32_t Pin;
  uint32_t Mode;
  uint32_t Pull;
  uint32_t Speed;
  uint32_t Alternate;
}GPIO_InitTypeDef;
```

레지스터 접근 방식

- 이 자료형은 앞의 매개변수인 GPIO_TypeDef 로 접근되는 레지스터에 실제 설정하고자 하는 값을 가짐

⋯› GPIO_InitTypeDef 자료형을 가진 GPIO_InitStruct 변수를 선언

```
GPIO_InitTypeDef GPIO_InitStruct = {0};

GPIO_InitStruct.Pin = LD3_Pin|LD2_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
```

레지스터 접근 방식

- Pin, Mode, Pull, Speed 변수에 설정하고자하는 설정 셋팅

요점노트

1. STM32F429의 레지스터 접근

- STM32F429의 레지스터 접근
  - &연산자 다음에 오는 변수(메모리)나 레지스터의 주소를 반환함
  - *연산자 다음에 오는 값(주소)의 내용을 반환함
  - 포인터는 메모리나 레지스터의 주소를 말하며 모든 메모리와 레지스터는 주소를 가짐
  - C언어의 포인터를 통해 원하는 주소의 레지스터와 메모리에 접근할 수 있음

요점노트

2. 자가 진단 SW 설계

- 자가 진단 SW 설계
  - STM32 시리즈의 펌웨어 라이브러리를 STM사에서 기본 제공함
  - CubeMX를 통해 생성된 코드에 STM32 라이브러리가 포함되어 있으며 무료로 자유롭게 사용할 수 있음
  - STM32 라이브러리는 레지스터 접근을 위해 C언어의 구조체, 포인터, typedef 지정자를 사용함