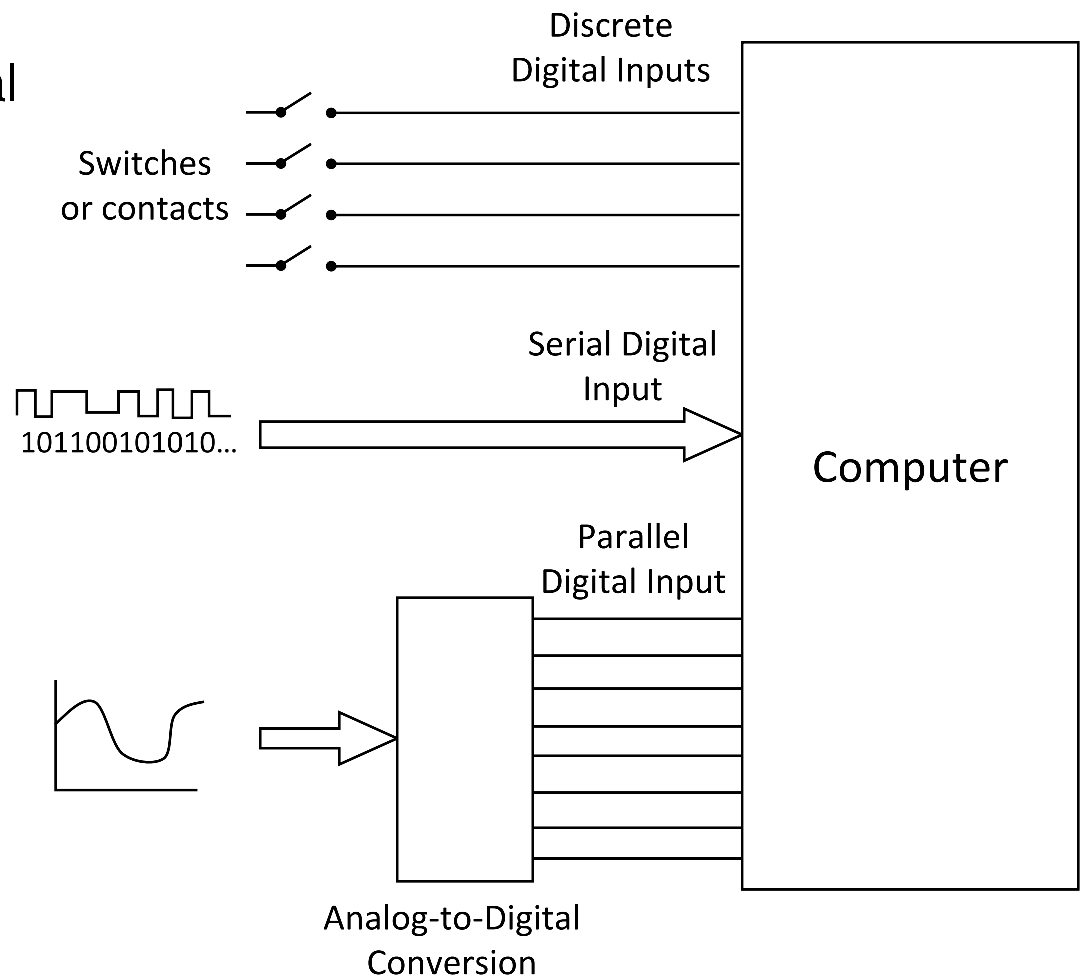
시스템 프로그래밍을 위한 C언어

- Hardware Memory Allocation to Access On-Chip Hardware via Memory Mapped I/O

현대자동차 입문교육
박대진 교수

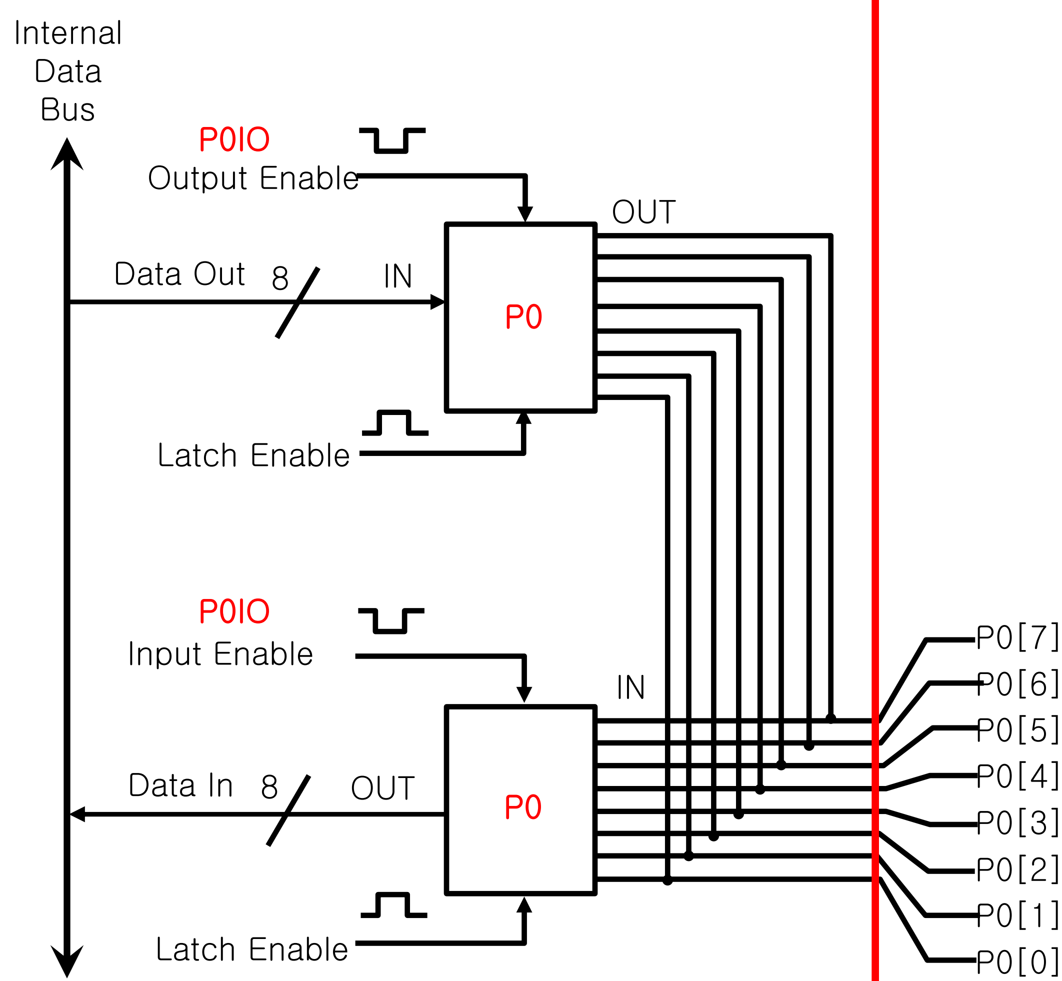
Real World Interfacing by using MCU

- Analog or Digital
- Bit or Bus



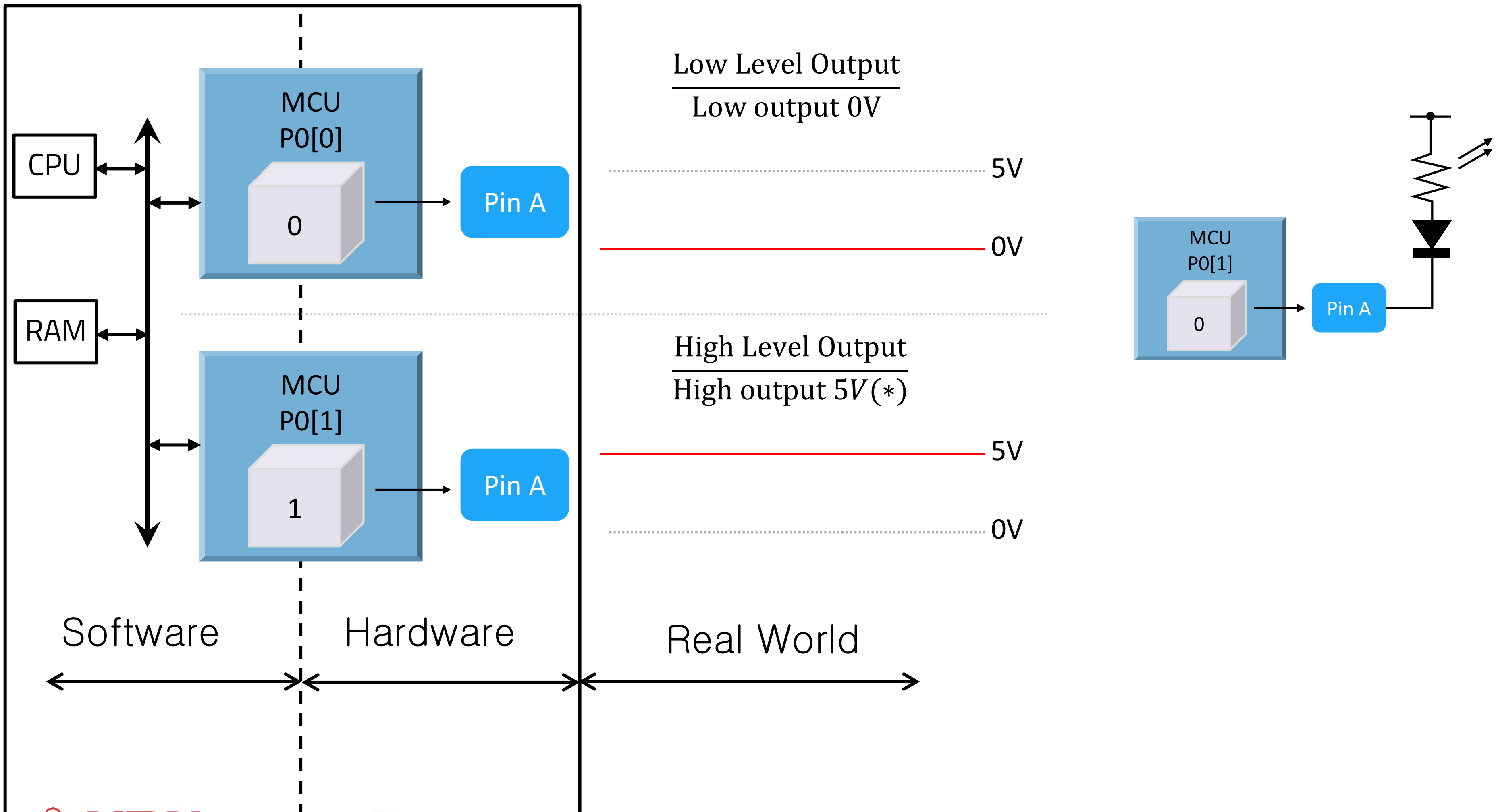
GPIO and Internal Bus

- General Purpose Input/Output
  - Interface Between Internal Bus and Outside world
  - Time-multiplexed Data Path (Input, output)
  - GPIO Port is mapped to registers in Memory Map



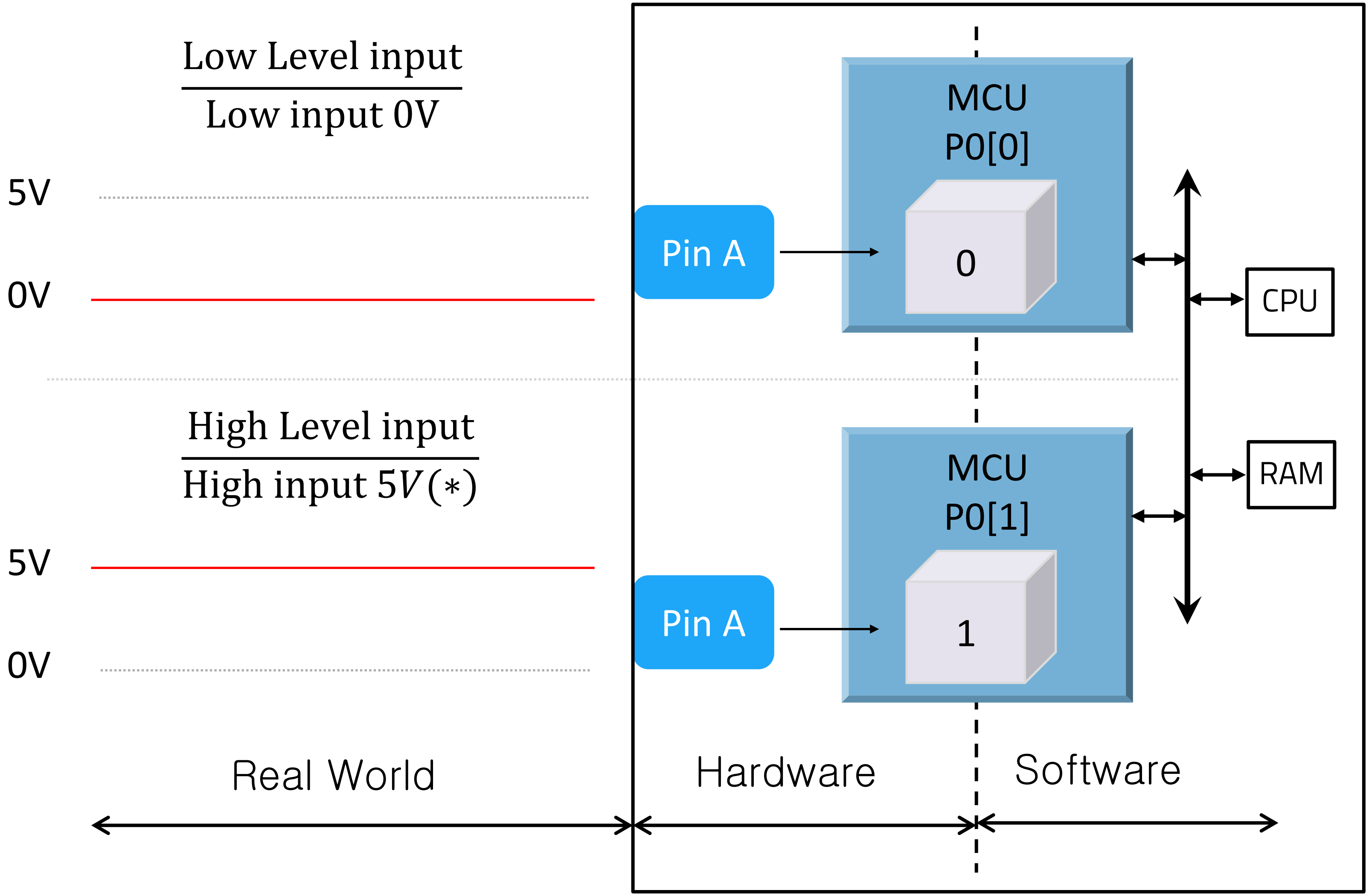
Interfacing via PORT Register on Memory Map : Write Mode

- Write value on Register → Control the output voltage



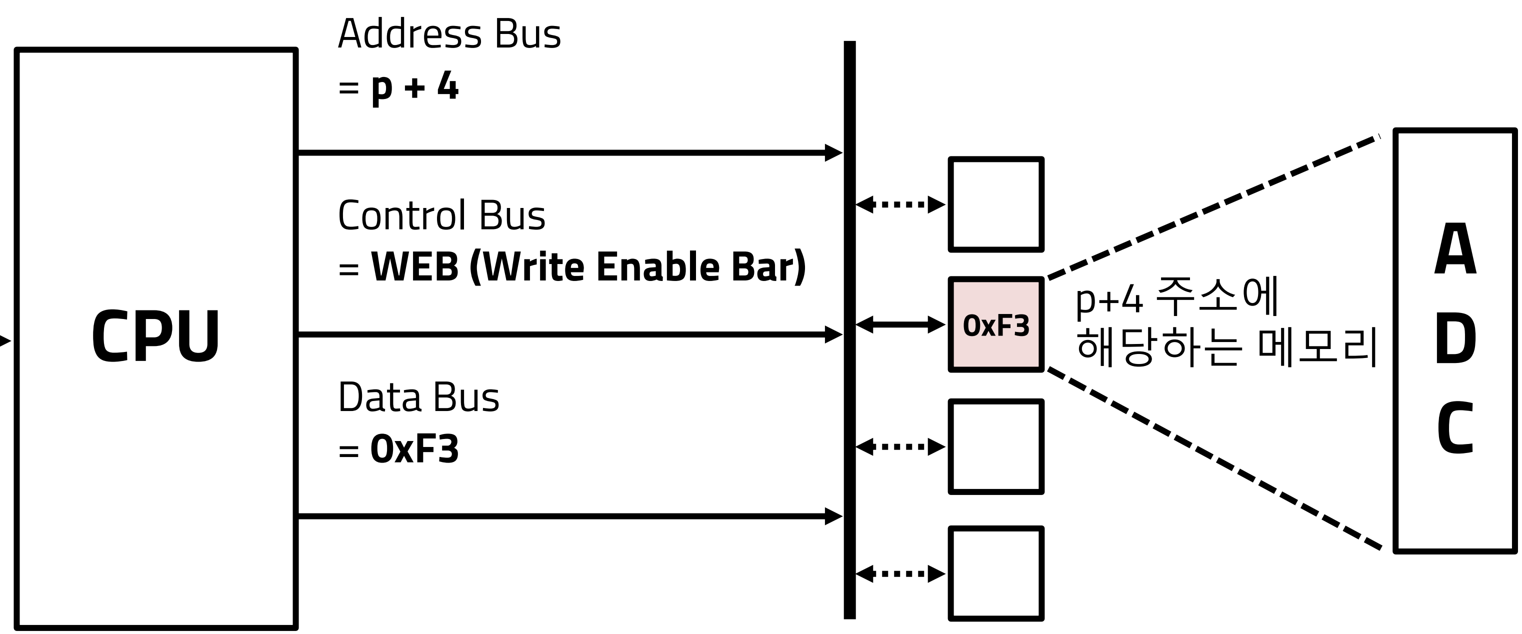
Interfacing via PORT Register on Memory Map: Read Mode

- Reading Register Value → Can Identify the input voltage



포인터 변수를 통해 메모리 버스에 접근

```
…
*(p + 4) = 0xF3
…
…
```

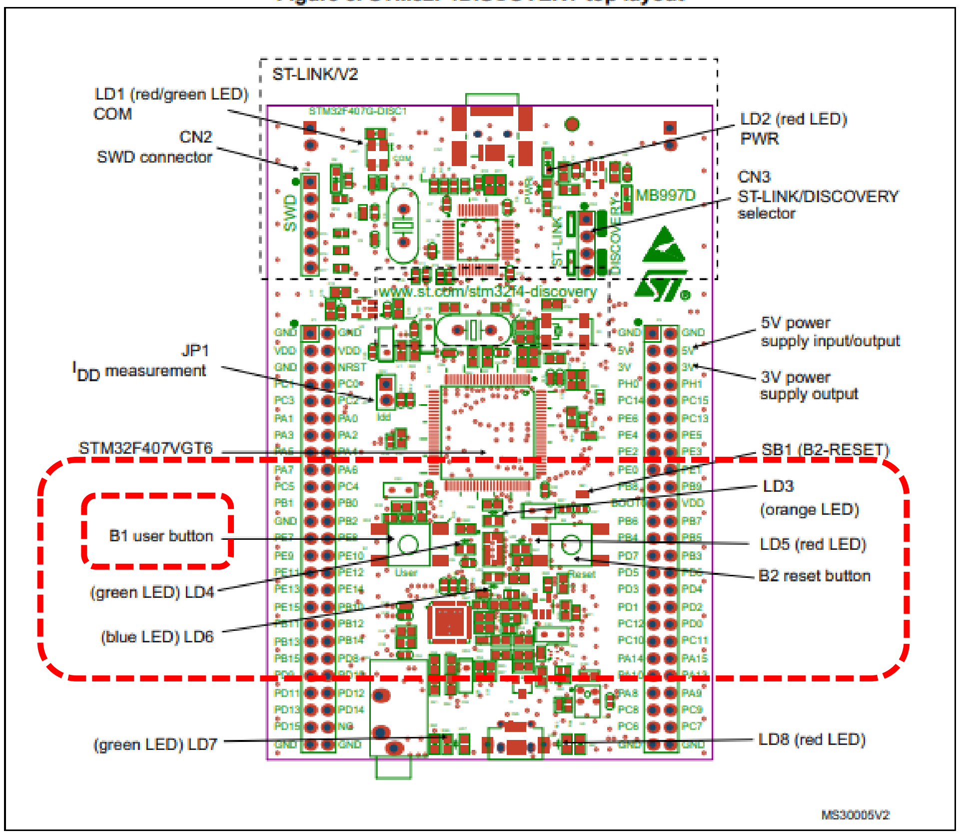


- 메모리의 한 영역에 값을 쓰게 되면
- 버스에 값이 실리게 되고
- 그 주소에 매칭되는 회로 (메모리도 그중에 하나)가 반응함
- 온칩에 내장된 회로들이 이렇게 버스에 묶여 있어서 자기 주소에 대응된다면 SW와 연동된다.

코드 설명 – GPIO 설정

- 보드의 LED, 버튼을 사용하기 위한 하드웨어 회로 파악
  - 4개 LED : LD3 ~ 6 는 PD12 ~ 15에 연결되어 있음 (PD, **Port D**)
  - 버튼 B1은 PA0에 연결되어 있음 (PA, **Port A**)

Figure 3. STM32F4DISCOVERY top layout



6.3 LEDs

- LD1 COM: LD1 default status is red. LD1 turns to green to indicate that communications are in progress between the PC and the ST-LINK/V2.
- LD2 PWR: red LED indicates that the board is powered.
- User LD3: orange LED is a user LED connected to the I/O PD13 of the STM32F407VGT6.
- User LD4: green LED is a user LED connected to the I/O PD12 of the STM32F407VGT6.
- User LD5: red LED is a user LED connected to the I/O PD14 of the STM32F407VGT6.
- User LD6: blue LED is a user LED connected to the I/O PD15 of the STM32F407VGT6.
- USB LD7: green LED indicates when VBUS is present on CN5 and is connected to PA9 of the STM32F407VGT6.
- USB LD8: red LED indicates an overcurrent from VBUS of CN5 and is connected to the I/O PD5 of the STM32F407VGT6.

6.4 Push buttons

- B1 USER: User and Wake-Up buttons are connected to the I/O PA0 of the STM32F407VG.
- B2 RESET: Push button connected to NRST is used to RESET the STM32F407VG.

코드 설명 – GPIO 설정

- GPIO Port A, Port D를 사용하기 위한 GPIO 레지스터 설정

Table 1. STM32F4xx register boundary addresses (continued)

Boundary address	Peripheral	Bus	Register map
0x5006 0800 - 0x5006 0BFF	RNG	AHB2	*Section 24.4.4: RNG register map on page 772*
0x5006 0400 - 0x5006 07FF	HASH		*Section 25.4.9: HASH register map on page 796*
0x5006 0000 - 0x5006 03FF	CRYP		*Section 23.6.13: CRYP register map on page 764*
0x5005 0000 - 0x5005 03FF	DCMI		*Section 15.8.12: DCMI register map on page 476*
0x5000 0000 - 0x5003 FFFF	USB OTG FS		*Section 34.16.6: OTG_FS register map on page 1326*
0x4004 0000 - 0x4007 FFFF	USB OTG HS	AHB1	*Section 35.12.6: OTG_HS register map on page 1469*
0x4002 B000 - 0x4002 BBFF	DMA2D		*Section 11.5: DMA2D registers on page 352*
0x4002 8000 - 0x4002 93FF	ETHERNET MAC		*Section 33.8.5: Ethernet register maps on page 1240*
0x4002 6400 - 0x4002 67FF	DMA2		*Section 10.5.11: DMA register map on page 335*
0x4002 6000 - 0x4002 63FF	DMA1		
0x4002 4000 - 0x4002 4FFF	BKPSRAM		
0x4002 3C00 - 0x4002 3FFF	Flash interface register		*Section 3.9: Flash interface registers*
0x4002 3800 - 0x4002 3BFF	RCC		*Section 7.3.24: RCC register map on page 265*
0x4002 3000 - 0x4002 33FF	CRC		*Section 4.4.4: CRC register map on page 115*
0x4002 2800 - 0x4002 2BFF	GPIOK		*Section 8.4.11: GPIO register map on page 286*
0x4002 2400 - 0x4002 27FF	GPIOJ		
0x4002 2000 - 0x4002 23FF	GPIOI		*Section 8.4.11: GPIO register map on page 286*
0x4002 1C00 - 0x4002 1FFF	GPIOH		
0x4002 1800 - 0x4002 1BFF	GPIOG		
0x4002 1400 - 0x4002 17FF	GPIOF		
0x4002 1000 - 0x4002 13FF	GPIOE		
0x4002 0C00 - 0x4002 0FFF	GPIOD		
0x4002 0800 - 0x4002 0BFF	GPIOC		
0x4002 0400 - 0x4002 07FF	GPIOB		
0x4002 0000 - 0x4002 03FF	GPIOA		
0x4001 6800 - 0x4001 6BFF	LCD-TFT	APB2	*Section 16.7.26: LTDC register map on page 510*
0x4001 5800 - 0x4001 5BFF	SAI1		*Section 29.17.9: SAI register map on page 966*
0x4001 5400 - 0x4001 57FF	SPI6		*Section 28.5.10: SPI register map on page 928*
0x4001 5000 - 0x4001 53FF	SPI5		

GPIOD 레지스터

GPIOA 레지스터

8.4.11 GPIO register map

The following table gives the GPIO register map and the reset values.

Table 39. GPIO register map and reset values

Offset	Register	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
0x00	GPIOA_ MODER	MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[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]	
	Reset value	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x00	GPIOB_ MODER	MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[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]	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
0x00	GPIOx_MODER (where x = C..I/J/K)	MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[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]	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

전체 레지스터 map에서 base address를 찾아 GPIO 모듈의 세부 레지스터 map으로 이동

코드 설명 – GPIO 설정

- 레지스터 설정 값을 결정하기 위한 스펙 문서 이해
  - GPIO의 여러 레지스터 중, 1개의 예시

8.4.1 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

2. 사용하려는 component를 위해서 전체 레지스터 중, 어디에 write해야 하는지 파악

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

1. 원하는 동작을 위해서 어떤 값을 사용해야 하는지 파악

코드 설명 – GPIO 설정

- GPIO 레지스터를 설정하는 코드

```c
int main (void) {

    clk();

    RCC_CFGR |= 0x04600000;

    /* PORT A */
    RCC_AHB1ENR  |= 1<<0; //RCC clock enable register
    GPIOA_MODER  |= 0<<0; // input mode
    GPIOA_OTYPER |= 0<<0; // output push-pull
    GPIOA_PUPDR  |= 0<<0; // no pull-up, pull-down

    /* PORT D */
    RCC_AHB1ENR  |= 1<<3;       // PORTD enable
    GPIOD_MODER  |= 1<<24;      // PORTD 12 general output mode
    GPIOD_MODER  |= 1<<26;      // PORTD 13 general output mode
    GPIOD_MODER  |= 1<<28;      // PORTD 14 general output mode
    GPIOD_MODER  |= 1<<30;      // PORTD 15 general output mode
    GPIOD_OTYPER |= 0x00000000;
    GPIOD_PUPDR  |= 0x00000000;

    GPIOD_ODR |= 1<<12;

    while(1) {
        if( GPIOA_IDR & 0x00000001 ) {
            GPIOD_ODR ^= 1 << 13;
            GPIOD_ODR ^= 1 << 14;
            GPIOD_ODR ^= 1 << 15;
        }
    }
}
```

PA0 (Port A의 pin 0)을
입력으로 사용하기 위한 설정
- Port A 모듈에 clock 인가
- 핀 입출력 방향 설정

PD12~15 (Port D의 pin 12~15)을 입력으로
사용하기 위한 설정
- Port D 모듈에 clock 인가
- 핀 입출력 방향 설정

출력 핀에 원하는 값 인가

코드 설명 – 전체 동작

- 버튼으로 입력되는 값에 따라 LED toggle 여부를 결정

```c
int main (void) {

    clk();

    RCC_CFGR |= 0x04600000;

    /* PORT A */
    RCC_AHB1ENR  |= 1<<0; //RCC clock enable register
    GPIOA_MODER  |= 0<<0; // input mode
    GPIOA_OTYPER |= 0<<0; // output push-pull
    GPIOA_PUPDR  |= 0<<0; // no pull-up, pull-down

    /* PORT D */
    RCC_AHB1ENR  |= 1<<3;       // PORTD enable
    GPIOD_MODER  |= 1<<24;      // PORTD 12 general output mode
    GPIOD_MODER  |= 1<<26;      // PORTD 13 general output mode
    GPIOD_MODER  |= 1<<28;      // PORTD 14 general output mode
    GPIOD_MODER  |= 1<<30;      // PORTD 15 general output mode
    GPIOD_OTYPER |= 0x00000000;
    GPIOD_PUPDR  |= 0x00000000;

    GPIOD_ODR |= 1<<12;

    while(1) {
        if( GPIOA_IDR & 0x00000001 ) {
            GPIOD_ODR ^= 1 << 13;
            GPIOD_ODR ^= 1 << 14;
            GPIOD_ODR ^= 1 << 15;
        }
    }
}
```

버튼으로 입력되는 값이 1이라면,

3개의 LED (PD13, PD14, PD15)의
상태를 toggle
- on되어 있으면 off
- off되어 있으면 on

여러가지 SFR 접근 방법

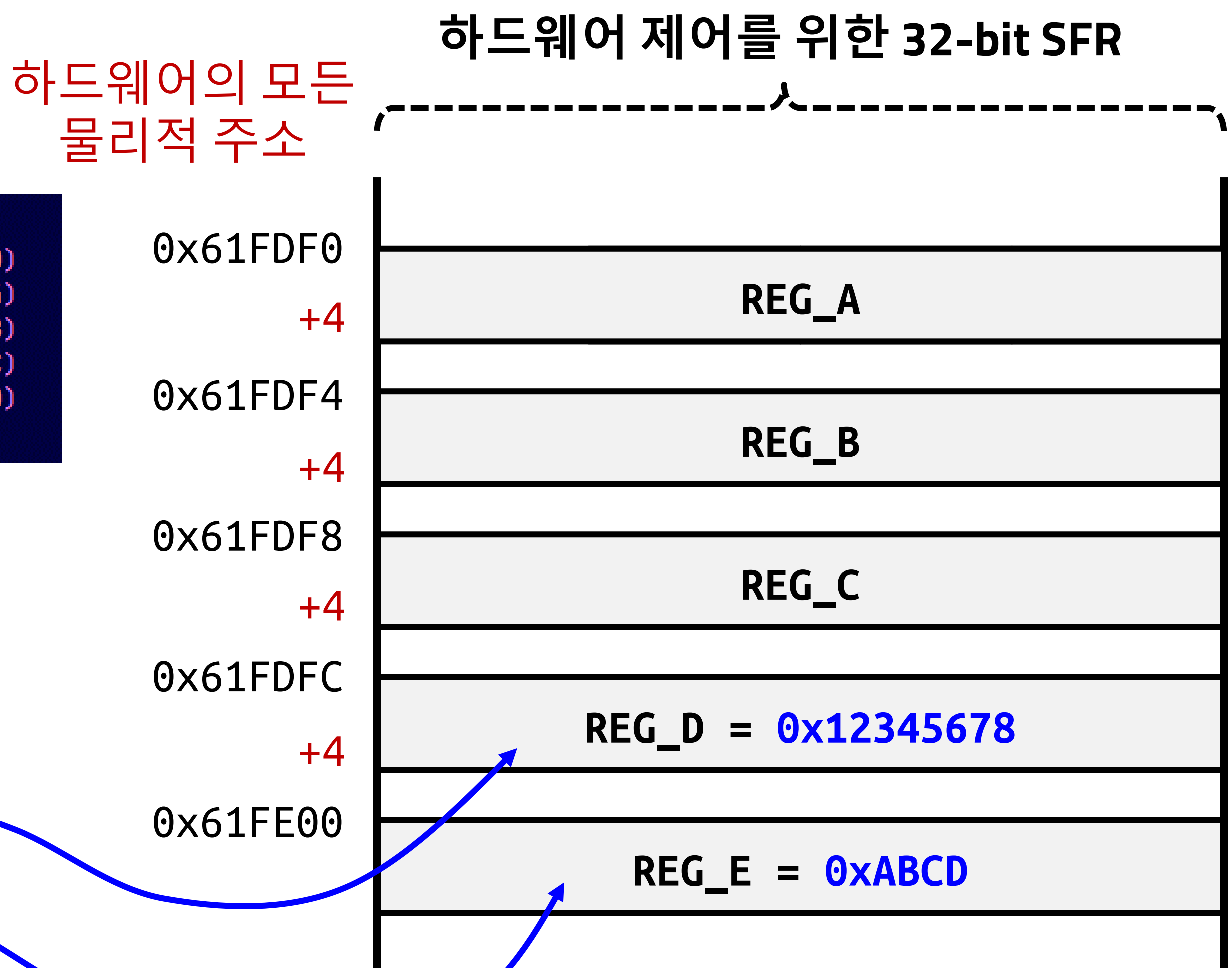
Memory Mapped된 하드웨어 영역의 특정 주소를 직접 지정

```
#define REG_A (*(volatile unsigned int*)0x61FDF0)
#define REG_B (*(volatile unsigned int*)0x61FDF4)
#define REG_C (*(volatile unsigned int*)0x61FDF8)
#define REG_D (*(volatile unsigned int*)0x61FDFC)
#define REG_E (*(volatile unsigned int*)0x61FE00)
```

특정 SFR의 주소에
직접적으로 접근

```
REG_D = 0x12345678;
REG_E = 0xABCD;
```

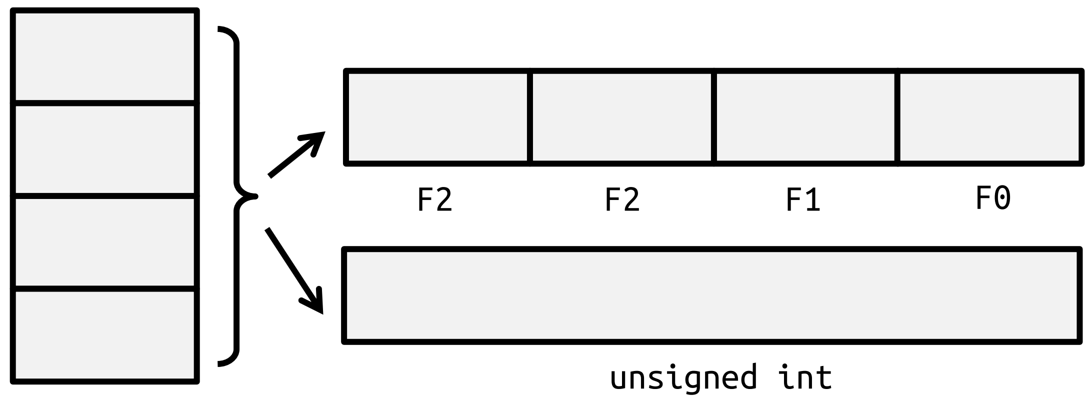
하드웨어에 값 할당



struct, union이용한 하드웨어 영역 표현

```c
struct REG_BITS {
    unsigned int F0 : 8;
    unsigned int F1 : 8;
    unsigned int F2 : 8;
    unsigned int F3 : 8;
};
union ADC_CONTROL {
    unsigned int U;
    struct REG_BITS B;
};
```

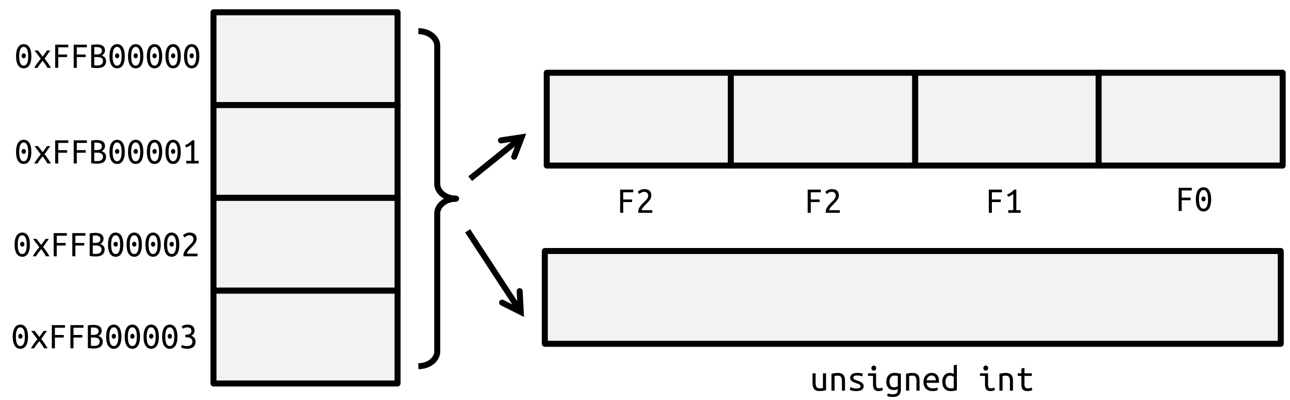
물리적인 메모리를 다음의 4개 바이트
혹은 32비트 unsigned int로 볼수 있다



포인터 타입을 이용하여 해당 영역의 주소를 표현

```c
struct REG_BITS {
    unsigned int F0 : 8;
    unsigned int F1 : 8;
    unsigned int F2 : 8;
    unsigned int F3 : 8;
};
union ADC_CONTROL {
    unsigned int U;
    struct REG_BITS B;
};
```

물리적인 메모리를 다음의 4개 바이트
혹은 32비트 unsigned int로 볼수 있다

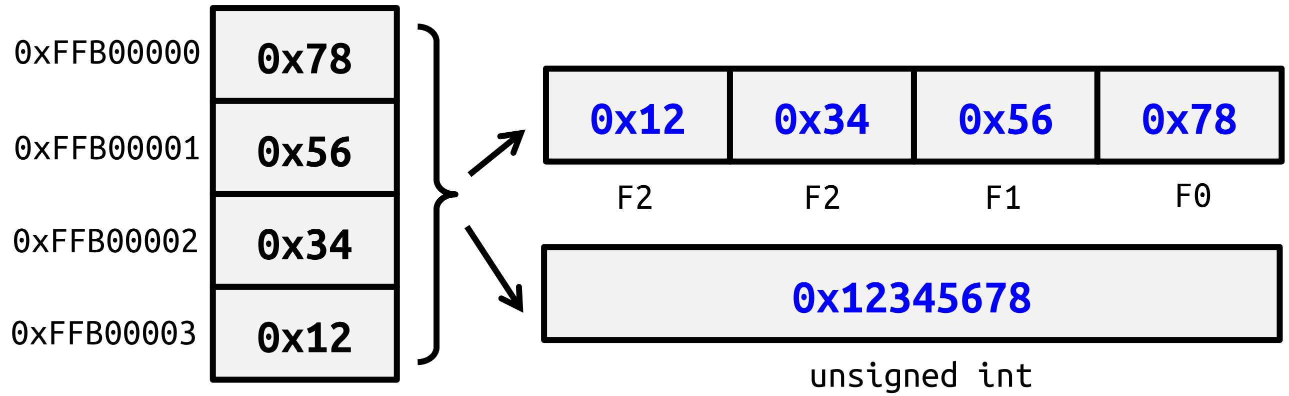



```c
#define HW_EMULATION 1
#ifdef HW_EMULATION
  union ADC_CONTROL adc_control1;
  #define ADCC (*(volatile union ADC_CONTROL*)&adc_control1)
#else
  #define ADCC (*(volatile union ADC_CONTROL*)0xFFB00000)
//#define P    (*(                         int*)0xFFCC0000)
#endif
```

해당주소영역에 값을 쓰기

```c
struct REG_BITS {
    unsigned int F0 : 8;
    unsigned int F1 : 8;
    unsigned int F2 : 8;
    unsigned int F3 : 8;
};
union ADC_CONTROL {
    unsigned int U;
    struct REG_BITS B;
};
```

물리적인 메모리를 다음의 4개 바이트
혹은 32비트 unsigned int로 볼수 있다



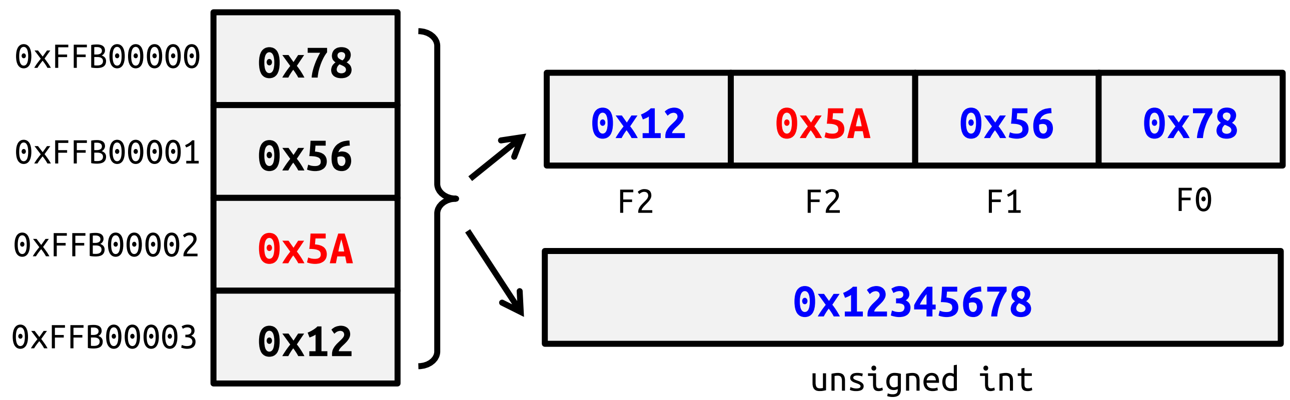

```c
#define ADCC (*(volatile union ADC_CONTROL*)0xFFB00000)
```

```c
ADCC.U = 0x12345678;
```

해당주소영역을 구조체기반 해당 bit slice에 접근

```c
struct REG_BITS {
    unsigned int F0 : 8;
    unsigned int F1 : 8;
    unsigned int F2 : 8;
    unsigned int F3 : 8;
};
union ADC_CONTROL {
    unsigned int U;
    struct REG_BITS B;
};
```

물리적인 메모리를 다음의 4개 바이트
혹은 32비트 unsigned int로 볼수 있다



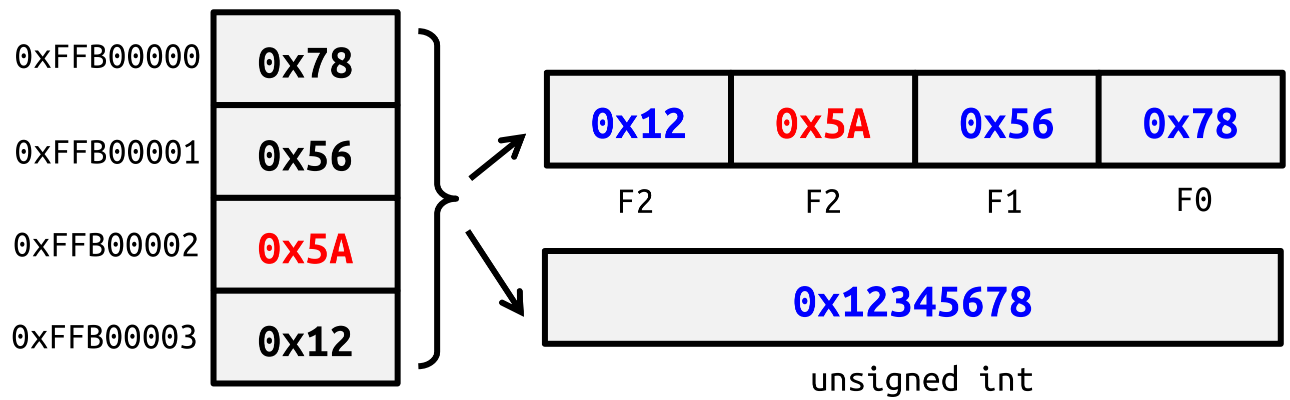

```c
#define ADCC (*(volatile union ADC_CONTROL*)0xFFB00000)
```

```c
ADCC.B.F2 = 0x5A;
```

해당주소영역을 masking통한 bit slice에 접근

물리적인 메모리를 다음의 4개 바이트
혹은 32비트 unsigned int로 볼수 있다

```c
struct REG_BITS {
    unsigned int F0 : 8;
    unsigned int F1 : 8;
    unsigned int F2 : 8;
    unsigned int F3 : 8;
};
union ADC_CONTROL {
    unsigned int U;
    struct REG_BITS B;
};
```




```c
#define ADCC (*(volatile union ADC_CONTROL*)0xFFB00000)
```

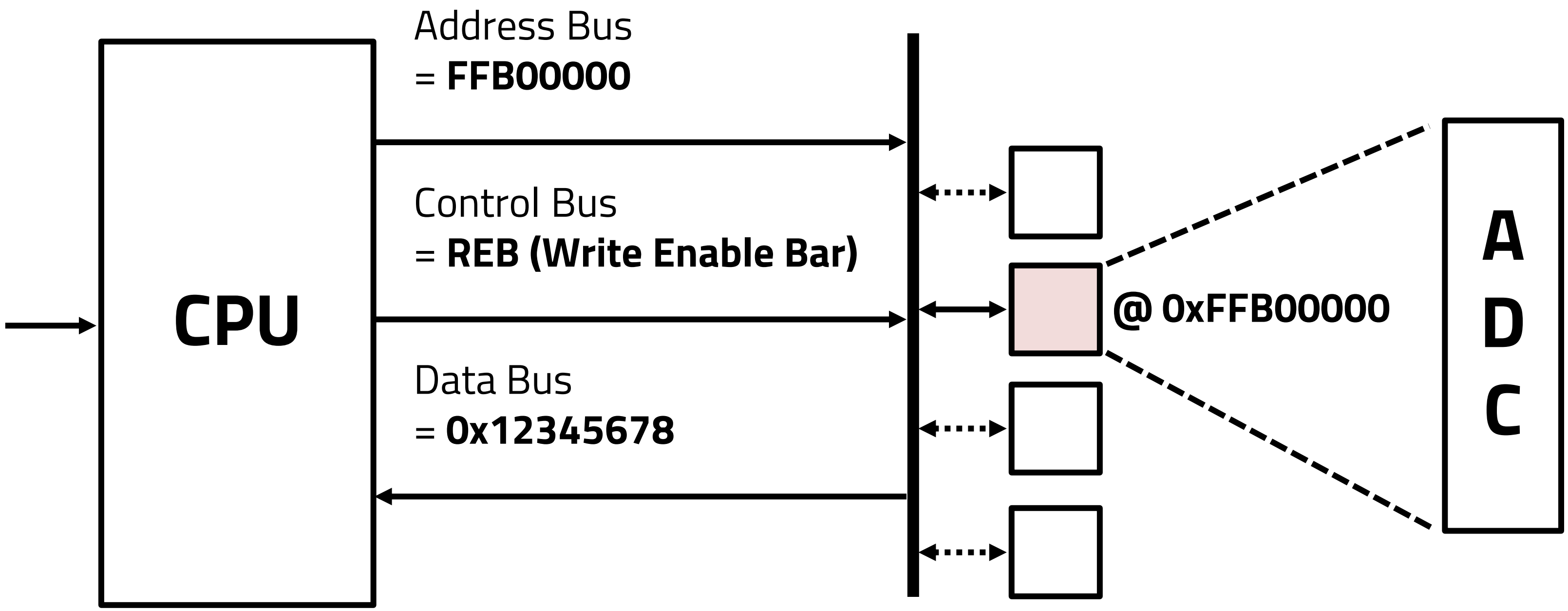
```c
#define F2_IDX 16
```

```c
ADCC.U  = 0x12345678;
ADCC.U &= ~(0xFF << F2_IDX); // clear bit slide using mask pattern generation
ADCC.U |= 0x5A << F2_IDX;    // then we can set bit slice
```

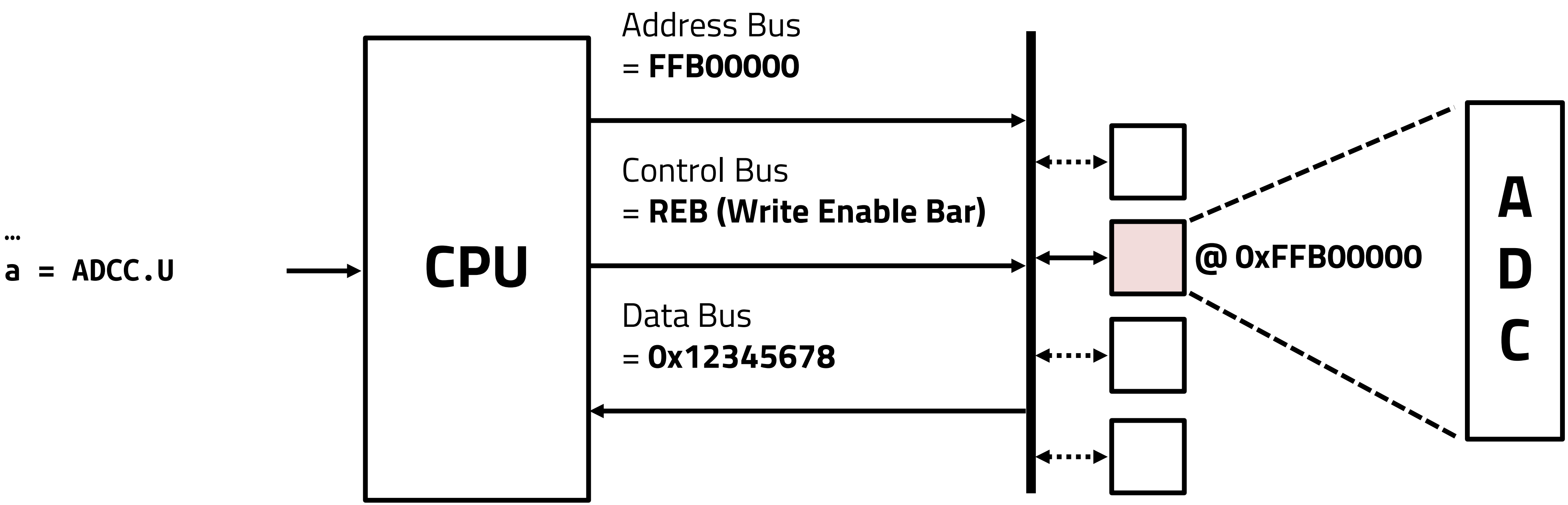
버스에 연결된 HW에 접근시 온칩 버스

```c
#define ADCC (*(volatile union ADC_CONTROL*)0xFFB00000)
```

```c
…
a = ADCC.U
…
```



HW 상태를 지속적으로 체크




```c
#define ADCC (*(volatile union ADC_CONTROL*)0xFFB00000)
// wait here until ADCC[3] is 1
while( (ADCC.U & (1<<EOC_IDX)) == 0);

// check hardare via bus with memory-mapped hardware
// ADCC[3] is 1, go through here
printf("End of conversion\n");
```