

# 임베디드 시스템을 위한 SW 구조 설계

(file #3 / 10) ver0.2

Yongseok Chi

# Course Objectives

---

## 1. Develop an understanding of technologies

about the micro controller & processor systems using evaluation kit

## 2. Skill up a **design ability the micro controller application systems**

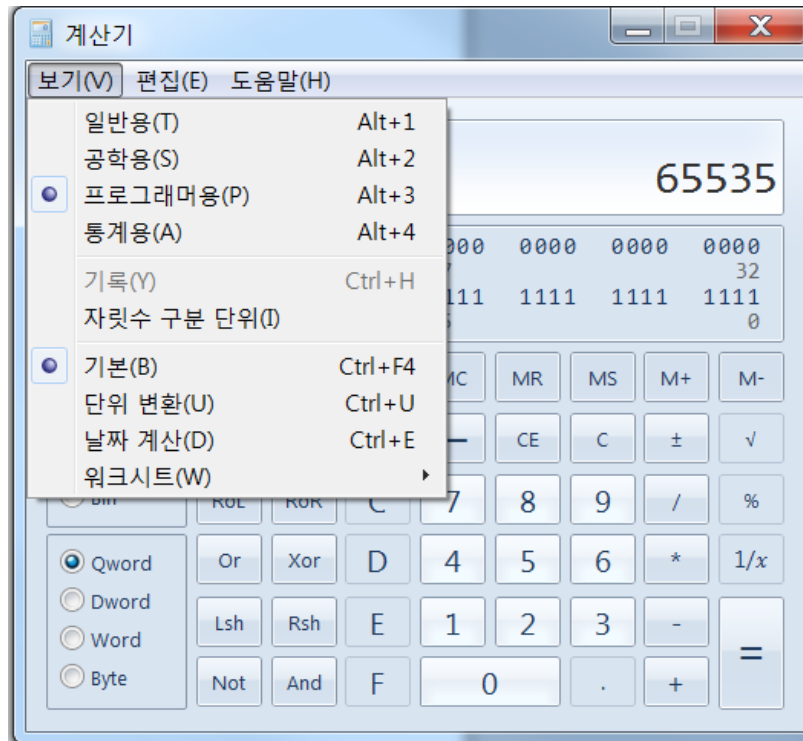
(1) technology of **the hardware and software** components

(2) **debugging** technology about the micro controller

(3) understanding of a **circuit design** skill

## 3. Develop an ability of design about the micro controller

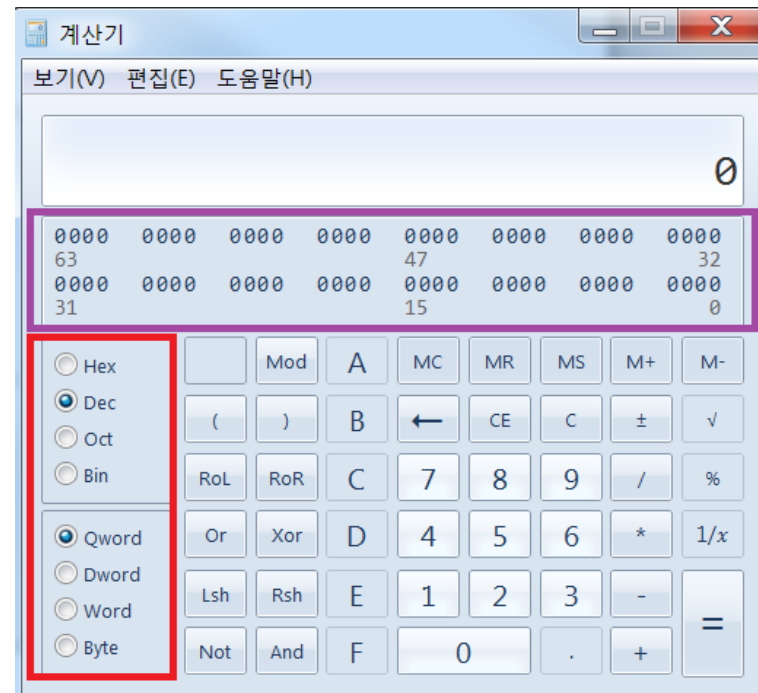
# 연산자 비트 연산자



1	0	1	0	1	0	1	0
$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
$1 \times 128$	$+ 0 \times 64$	$+ 1 \times 32$	$+ 0 \times 16$	$+ 1 \times 8$	$+ 0 \times 4$	$+ 1 \times 2$	$+ 0 \times 1 = 170$

# 연산자 비트 연산자

2진수	10진수	16진수
00000000	0	00
00000001	1	01
00000010	2	02
00000011	3	03
00001010	10	0A
00001111	15	0F
00010000	16	10
10101010	170	AA
11111111	255	FF
000011111111	255	0FF
0000000011111111	255	00FF
0000111111111111	4095	0FFF
1111111111111111	6535	FFFF



# 연산자 &

origin : 10101010	origin : 10101010	origin : 10101010	origin : 10101010
10000000	01000000	00100000	00010000
& -----	& -----	& -----	& -----
10000000	00000000	00100000	00000000

& 를 하면 두 값이 1일 경우를 제외하고 모두 0으로 만든다

7bit 6bit 5bit 4bit 3bit 2bit 1bit 0bit 중에서 4bit 값만 유지, 나머지는 0 만들기

& 0 0 0 1 0 0 0 0

if 4bit=0, 결과 0, 4bit=1 결과 00010000

16진수로 표현

AA	AA	AA	AA
& 80	& 40	& 20	& 10
결과 10	결과 00	결과 10	결과 00

# 연산자 |

origin : 10101010

10000000

| -----

10101010

origin : 10101010

01000000

| -----

11101010

origin : 10101010

00100000

| -----

10101010

origin : 10101010

00010000

| -----

10111010

| 를 하면 두 값이 0일 경우를 제외하고 모두 1으로 만든다

7bit 6bit 5bit 4bit 3bit 2bit 1bit 0bit 중에서 4bit 값을 무조건 1, 나머지는 유지하기

| 0 0 0 1 0 0 0 0

if 4bit=0, 결과 1,

4bit=1 결과 □ □ □ 1 □ □ □ □

16진수로 표현

AA

| 80

결과 AA

AA

| 40

결과 EA

AA

| 20

결과 AA

AA

| 10

결과 BA

# About This Course

---

1. Micro Processor 원리
2. Atmel사의 8bit Micro-controller
3. KUT0128 Evaluation Board 기능과 특징
4. IO Port 제어
5. External Interrupt 제어
6. Timer counter 제어
7. UART 제어
8. AD Converter 제어
9. Comparator 제어
10. EEPROM 제어 (IIC, Parallel method)
11. SPI 제어

# 4. IO Port 제어

## ATmega128 PIN MAP

PA, PB, PC, PD, PE, PF

: 8Ports

: 3 states @ reset

: Bi-direction

: Internal Pull\_up

PG

: 5Ports

: 3 states @ reset

: Bi-direction

: Internal Pull\_up

Serial IO

Analog  
Comparator

Timer Counter  
Comparator

Timer Counter  
Clock

Programming  
enable

ADC

JTAG

External memory

: Lower address

: Data 8bits

IO Port

: 40mA

External memory

: Upper address

IO Port

: 40mA

address 16개

→ 64 x 1024: 64K

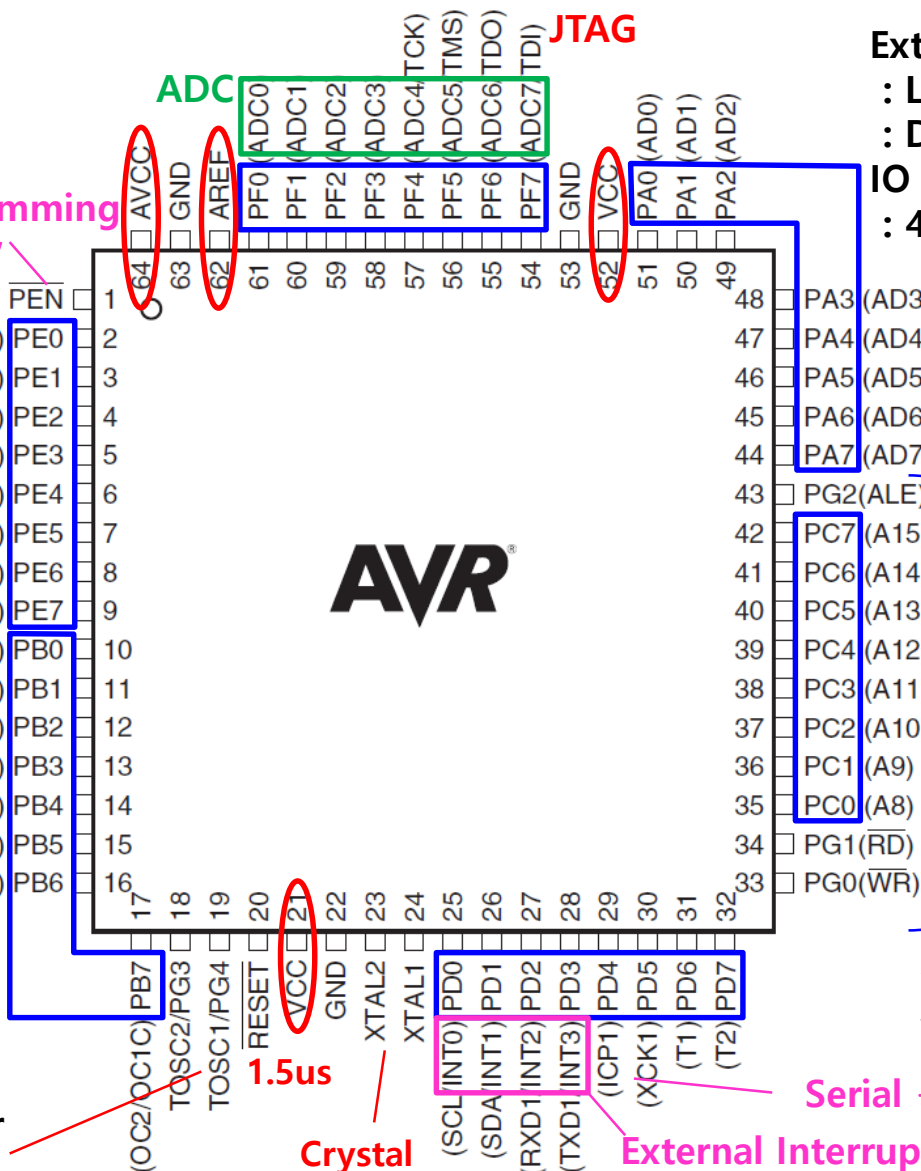
Address Latch Enable  
/Read, /Write

Serial 통신

External Interrupt

Sink current : input

Source current : output

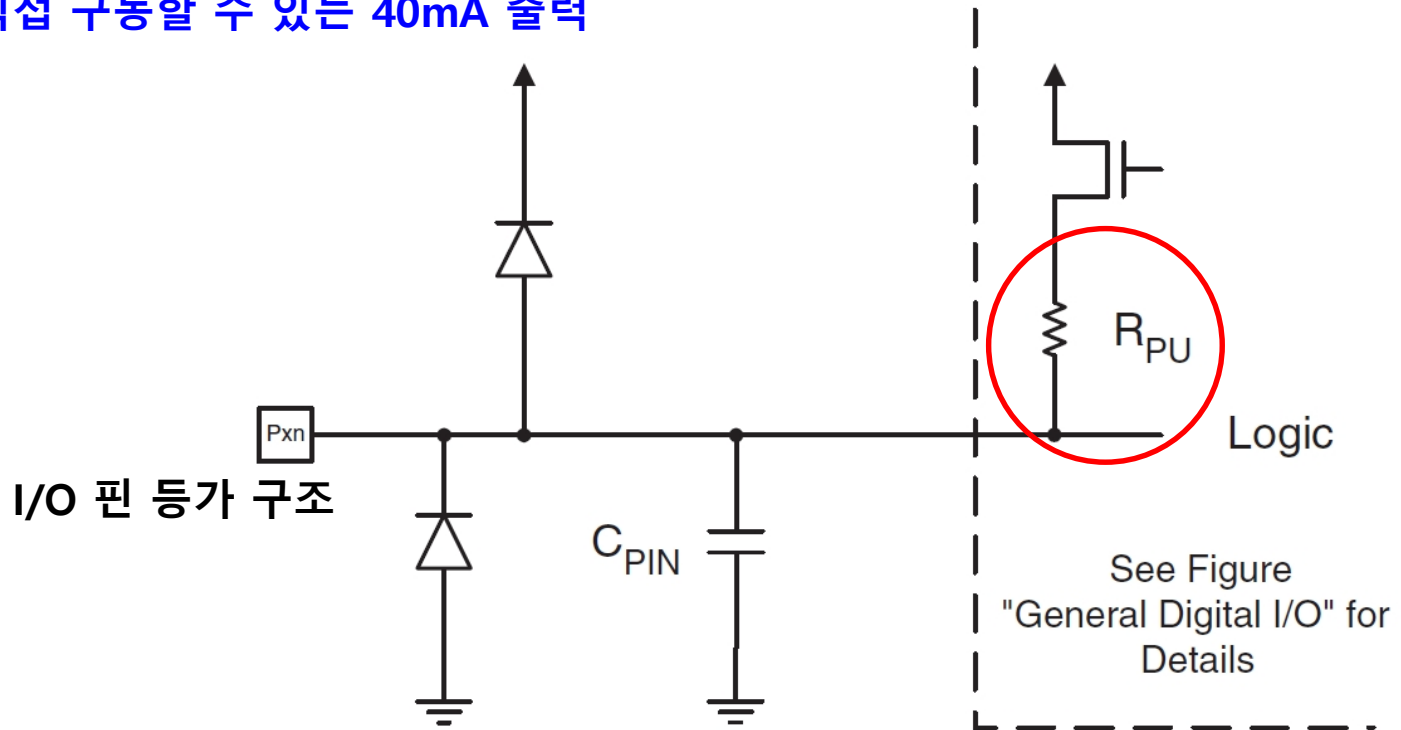




## 4. IO Port 제어

### I/O PORT의 특징

- 8비트 양방향 병렬 I/O포트 6개(A,B,C,D,E,F)와 5비트 양방향 병렬 I/O포트 (G)로 53개의 I/O포트
- 각 I/O핀은 보호용 다이오드와  $20\text{K}\Omega \sim 100\text{K}\Omega$ 의 내부 풀업 저항을 가지며
- LED를 직접 구동할 수 있는 40mA 출력



## 4. IO Port 제어

### I/O PORT의 특징(cont'd)

레지스터	기 능
DDRx	방향 설정 레지스터(비트단위 지정 1: 출력 ,0: 입력)
PORTx	포트 출력 레지스터 (출력용)
PINx	포트 입력 레지스터 (입력용)

※ 각 레지스터에서 **x**는 각 포트 **A,B,C,D,E,F,G** 를 나타낸다.

- PORTA는 PORTA7~PORTA0번까지 **8개의 비트로** 제어가 되며
- PORTG는 PORTG4~PORTG0번까지 **5개의 비트로** 제어된다.

## 4. IO Port 제어

### PORTA 레지스터

포트 출력 레지스터

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

포트 방향 레지스터

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

포트 입력 레지스터

Bit	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

※ PORTB~F : 동일한 구조의 레지스터

## 4. IO Port 제어

### PORTG 레지스터

포트 출력 레지스터

Bit	7	6	5	4	3	2	1	0	
	–	–	–	PORTG4	PORTG3	PORTG2	PORTG1	PORTG0	PORTG
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

포트 방향 레지스터

Bit	7	6	5	4	3	2	1	0	
	–	–	–	DDG4	DDG3	DDG2	DDG1	DDG0	DDRG
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

포트 입력 레지스터

Bit	7	6	5	4	3	2	1	0	
	–	–	–	PING4	PING3	PING2	PING1	PING0	PING
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	N/A	N/A	N/A	N/A	N/A	

## 4. IO Port 제어

### I/O PORT의 기본 동작

※ x는 각 포트 A,B,C,D,E,F,G 를

- DDxn : **방향 설정 레지스터(DDRx)의 각 비트**      n은 각 Port의 pin(0,1,2,3,4,5,6,7)
  - 1 : 출력 핀으로 설정, 0 : 입력 핀으로 설정

DDRA = 0b00001111;      // PA7~PA4 입력, PA3~PA0 출력으로 설정, 2진수로 라이트

DDRA = 0x0f;      // 16진수로 라이트

7	6	5	4	3	2	1	0	
0	0	0	0	1	1	1	1	DDRA = 0FH
입력	입력	입력	입력	출력	출력	출력	출력	

DDRB = 0b10101010;      // PB7,PB5,PB3,PB1 입력, PB6,PB4,PB2,PB0을 출력으로 설정

DDRB = 0xaa;      // 16진수로 라이트

7	6	5	4	3	2	1	0	
1	0	1	0	1	0	1	0	DDRB = AAH
출력	입력	출력	입력	출력	입력	출력	입력	

## 4. IO Port 제어

### I/O PORT의 기본 동작 (cont'd)

- PORTxn : **출력** 설정된 포트의 **출력 레지스터의 각 비트**
  - 1을 쓰면 : 1출력, 0을 쓰면 : 0출력
  - **출력 전에 DDR 레지스터를 출력으로 설정해야 함**

```
DDRA = 0xff;           // 포트 A를 출력으로 설정  
PORTA = 0xfa;         // 포트 A에 fa를 출력  
DDRF = 0xff;           // 포트 F를 출력으로 설정  
PORTF = 0x2b;          // 포트 F에 2B를 출력
```

76543210 pin  
0b11111010

76543210 pin  
0b00101011

※ x는 각 포트 A,B,C,D,E,F,G 를  
n은 각 Port의 pin(0,1,2,3,4,5,6,7)

## 4. IO Port 제어

### I/O PORT의 기본 동작 (cont'd)

- PINxn : 입력 설정된 포트의 입력 레지스터

```
unsigned char var1;          // 변수 var1을 unsigned char형으로 선언
DDRA = 0x00;                // 포트 A를 입력으로 설정
var1 = PINA;                 // 포트 A로 입력받은 데이터를 변수 var1에 라이트

unsigned char var1;          // 변수 var1을 unsigned char형으로 선언
DDRD = 0x00;                // 포트 A를 입력으로 설정
var1 = PIND;                 // 포트 A로 입력받은 데이터를 변수 var1에 라이트
```

※ x는 각 포트 A,B,C,D,E,F,G 를  
n은 각 Port의 pin(0,1,2,3,4,5,6,7) 을

## 4. IO Port 제어

### I/O PORT의 기본 동작 (cont'd)

- SFIOR(Special Function Input Output Register)

: I/O 포트 핀의 내부 Pull-up 저항 사용 여부는 SFIOR의 PUD 비트로 설정

: PUD=1 Pull-up 저항 사용할 수 없음, PUD=0 Pull-up 저항을 사용

bit	7	6	5	4	3	2	1	0	
	TSM	—	—	—	ACME	PUD	PSR0	PSR321	SFIOR
Read/Write 초기값	R/W 0	R 0	R 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	

DDxn	PORTxn	PUD (in SFIOR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
입력 0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
입력 0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)



## 4. IO Port 제어

### I/O 포트의 alternate function (포트A)

PORTA 핀의 <b>alternate function</b>	
포트 핀	포트 A의 alternate function
PA0	AD0(외부 메모리 인터페이스 <b>어드레스와 데이터 비트0</b> )
PA1	AD1(외부 메모리 인터페이스 어드레스와 데이터 비트1)
PA2	AD2(외부 메모리 인터페이스 어드레스와 데이터 비트2)
PA3	AD3(외부 메모리 인터페이스 어드레스와 데이터 비트3)
PA4	AD4(외부 메모리 인터페이스 어드레스와 데이터 비트4)
PA5	AD5(외부 메모리 인터페이스 어드레스와 데이터 비트5)
PA6	AD6(외부 메모리 인터페이스 어드레스와 데이터 비트6)
PA7	AD7(외부 메모리 인터페이스 어드레스와 데이터 비트7)

## 4. IO Port 제어

### I/O 포트의 alternate function (포트B)

PORTB 핀의 alternate function	
포트 핀	포트 B의 alternate function
PB0	$\overline{SS}$ (SPI slave 선택 입력)
PB1	SCK (SPI 버스 직렬 클럭)
PB2	MOSI(SPI 버스 마스터 출력/slave 입력)
PB3	MISO(SPI 버스 마스터 입력/slave 출력)
PB4	OC0(타이머/카운터0 출력비교 매치 및 PWM 출력)
PB5	OC1A(타이머/카운터1 출력비교 매치 및 PWM 출력 A)
PB6	OC1B(타이머/카운터1 출력비교 매치 및 PWM 출력 B)
PB7	OC2/OC1C(타이머/카운터2 출력비교 매치 및 PWM 출력 또는 타이머/카운터1 출력비교 매치 및 PWM 출력 C)

## 4. IO Port 제어

### I/O 포트의 alternate function (포트C)

PORTC 핀의 alternate function	
포트 핀	포트 C의 alternate function
PC0	AD8(외부 메모리 인터페이스 어드레스와 데이터 비트8)
PC1	AD9(외부 메모리 인터페이스 어드레스와 데이터 비트9)
PC2	AD10(외부 메모리 인터페이스 어드레스와 데이터 비트10)
PC3	AD11(외부 메모리 인터페이스 어드레스와 데이터 비트11)
PC4	AD12(외부 메모리 인터페이스 어드레스와 데이터 비트12)
PC5	AD13(외부 메모리 인터페이스 어드레스와 데이터 비트13)
PC6	AD14(외부 메모리 인터페이스 어드레스와 데이터 비트14)
PC7	AD15(외부 메모리 인터페이스 어드레스와 데이터 비트15)

## 4. IO Port 제어

### I/O 포트의 alternate function (포트D)

PORTD 핀의 alternate function	
포트 핀	포트 D의 alternate function
PD0	INT0/SCL (외부 인터럽트 0 또는 TWI 시리얼 클럭)
PD1	INT1/SDA (외부 인터럽트 1 또는 TWI 시리얼 데이터)
PD2	INT2/RXD1(외부 인터럽트 2 또는 USART1 수신 단자)
PD3	INT3/TXD1(외부 인터럽트3 또는 USART1 송신 단자)
PD4	IC1(타이머/카운터1 입력 capture trigger 입력)
PD5	XCK1 (USART1 외부 클럭 입/출력)
PD6	T1 (타이머/카운터 1 클럭 입력)
PD7	T2 (타이머/카운터 2 클럭 입력)

## 4. IO Port 제어

### I/O 포트의 alternate function (포트E)

PORTE 핀의 alternate function	
포트 핀	포트 E의 alternate function
PE0	PDI/RXD0(프로그래밍 데이터 입력 또는 USART0 수신)
PE1	PDO/TXD0(프로그래밍 데이터 출력 또는 USART0 송신)
PE2	AIN0/XCK0(아날로그 + 입력 0 또는 USART 0 외부 클럭 입출력)
PE3	AIN1/OC3A(아날로그-입력1 또는 타이머/카운터 3 출력비교 및 PWM 출력 A)
PE4	INT4/OC3B(외부 인터럽트 4 또는 타이머/카운터 3 출력비교 및 PWM 출력 B)
PE5	INT5/OC3C(외부 인터럽트 5 또는 타이머/카운터 3 출력비교 및 PWM 출력 C)
PE6	INT6/T3(외부 인터럽트 6 또는 타이머/카운터 3외부 클럭 입력)
PE7	INT7/IC3(외부 인터럽트 7 또는 타이머/카운터 3입력 캡처 트리거 입력)

## 4. IO Port 제어

### I/O 포트의 alternate function (포트F)

PORTF 핀의 alternate function	
포트 핀	포트 F의 alternate function
PF0	ADC0 (ADC 입력 채널 0)
PF1	ADC1 (ADC 입력 채널 1)
PF2	ADC2 (ADC 입력 채널 2)
PF3	ADC3 (ADC 입력 채널 3)
PF4	ADC4/TCK (ADC 입력 채널 4 또는 JTAG테스트 클럭)
PF5	ADC5/TMS (ADC 입력 채널 5 또는 JTAG 테스트 모드 선택)
PF6	ADC6/TDO (ADC 입력 채널 6 또는 JTAG 테스트 데이터 출력)
PF7	ADC7/TDI (ADC 입력 채널 7 또는 JTAG 테스트 데이터 입력)

## 4. IO Port 제어

### I/O 포트의 alternate function (포트G)

PORTG 핀의 alternate function	
포트 핀	포트 G의 alternate function
PG0	$\overline{WR}$ (외부 데이터 메모리 쓰기 신호)
PG1	$\overline{RD}$ (외부 데이터 메모리 읽기 신호)
PG2	ALE(외부 메모리 어드레스 latch 신호)
PG3	TOSC2 (타이머/카운터 0의 Real Time Clock 발진기 출력)
PG4	TOSC1 (타이머/카운터 0의 Real Time Clock 발진기 입력)

## 4. IO Port 제어

---

[ Header files ] C:\wcvavreval\WINC

1. mega128a.h / mega128\_bits.h
2. avr\_compiler.h
3. compiler.h
4. ctype.h
5. interrupt.h
6. iobits.h

[ Documentation ]

1. C:\wcvavreval\WDocumentation

- ➔ CodeVisionAVR User Manual.pdf
- ➔ CVAVR Getting started with Atmel Studio.pdf



## 4. IO Port 제어

### Code Vision에서 레지스터 정의

- Code Vision 에서 Atmega128의 레지스터 명과 그 어드레스를 <mega128.h>에 정의한 헤더파일을 제공
- 인터럽트 함수를 정의할 때 사용하는 인터럽트 소스명도 함께 정의
- 사용자는 프로그램 처음에 #include와 함께 사용

예)        **#include <mega128.h>**  
             **#include <mega128a.h>**  
             #include <mega8.h>  
             #include <mega32a.h>

※ ATmega128을 사용한다면 mega128.h나 mega128a.h를 같이 사용해도 큰 문제는 없다  
ATmega128을 개선한 모델이 ATmega128a 이기 때문에 호환이 가능하다.  
다만 ATmega8이나 ATmega32a등 다른 모델을 사용하기 위해서는 그에 맞는 헤더파일을 입력해야 하며 이는 Code Vision 이 설치되어있는 폴더에 INC 폴더 안에 정리됨.

구성 라이브러리에 포함 공유 대상 새 폴더

즐거찾기

바탕 화면

라이브러리

홈 그룹

yongseok

컴퓨터

OS (C:)

네트워크

제어판

네트워크 및 인터넷

모든 제어판 항목

모양 및 개인 설정

사용자 계정 및 가족 보호

시계, 언어 및 국가별 옵션

시스템 및 보안

접근성

프로그램

하드웨어 및 소리

휴지통

## 헤더파일 폴더

이름	수정한 날짜	유형	크기
mega88a.h	2010-10-14 오전 8...	H 파일	5KB
mega88p.h	2009-04-14 오후 3...	H 파일	5KB
mega88pa.h	2010-10-14 오전 8...	H 파일	5KB
mega88pb.h	2014-12-16 오전 1...	H 파일	5KB
mega103.h	2009-04-14 오후 1...	H 파일	3KB
mega103_bits.h	2009-04-10 오전 8...	H 파일	24KB
mega128.h	2009-04-14 오후 1...	H 파일	5KB
mega128_bits.h	2009-04-09 오후 1...	H 파일	45KB
mega128a.h	2011-02-21 오전 1...	H 파일	5KB
mega128rfa1.h	2012-05-15 오후 2...	H 파일	14KB
mega128rfa1_bits.h	2010-03-04 오후 2...	H 파일	111KB
mega128rfr2.h	2013-08-09 오후 1...	H 파일	14KB
mega161.h	2009-04-14 오후 1...	H 파일	3KB
mega161_bits.h			26KB
mega162.h			4KB
mega162_bits.h			35KB
mega163.h			3KB
mega163_bits.h			27KB
mega164.h			5KB
mega164a.h			5KB
mega165.h			4KB
mega165_bits.h			39KB
mega165a.h			4KB
mega165a_bits.h	2012-05-16 오전 1...	H 파일	39KB
mega165p_bits.h	2012-05-16 오전 1...	H 파일	39KB
mega165pa.h	2010-10-14 오전 8...	H 파일	4KB

1. mega128a.h / mega128\_bits.h

2. avr\_compiler.h

3. compiler.h

4. ctype.h

5. interrupt.h / delay.h

6. iobits.h

## 4. IO Port 제어

### mega128a.h 정의

#### #pragma

- : compiler에게 명령하는 **전처리 명령어**.
- : 명령이 전달되어 **compile 이전에 실행됨**.
- : 기능은 다양함.

#### #ifdef

#### #define

#### Mega128a.h

TCNT2  
TCCR2  
ICR1L  
ICR1H  
ICR1

OCR1BH  
OCR1B  
OCR1AL  
OCR1AH  
OCR1A  
TCNT1L  
TCNT1H  
TCNT1  
TCCR1B  
TCCR1A  
ASSR  
OCR0  
TCNT0  
TCCR0  
MCUCSR  
MCUCR  
TIFR  
TIMSK  
EIFR  
EIMSK  
EICRB  
RAMPZ  
XDIV  
SPL  
SPH  
SREG

```
// CodeVisionAVR V2.05.1+ C Compiler  
// (C) 1998-2011 Pavel Haiduc, HP InfoTech S.R.L.  
  
// I/O registers definitions for the ATmega128A
```

```
#ifndef _MEGA128A_INCLUDED_  
#define _MEGA128A_INCLUDED_
```

```
#pragma used+  
sfrb PINF=0;  
sfrb PINE=1;  
sfrb DDRE=2;  
sfrb PORTE=3;  
sfrb ADCL=4;  
sfrb ADCH=5;  
sfrw ADCW=4; // 16 bit access  
sfrb ADCSRA=6;  
sfrb ADMUX=7;  
sfrb ACSR=8;  
sfrb UBRR0L=9;  
sfrb UCSROB=0xa;  
sfrb UCSROA=0xb;  
sfrb UDRO=0xc;  
sfrb SPCR=0xd;  
sfrb SPSR=0xe;  
sfrb SPDR=0xf;  
sfrb PIND=0x10;  
sfrb DDRD=0x11;  
sfrb PORTD=0x12;  
sfrb PINC=0x13;  
sfrb DDRC=0x14;
```

<https://msdn.microsoft.com/ko-KR/library/y4skk93w.aspx>

DDRG  
PORTG  
SPMCSR  
EICRA  
XMCRA  
OSCCAL  
TWBR  
TWSR  
TWAR  
TWRD  
TWCR  
OCR1CL  
OCR1CH  
TCCR1C

```
#define PORTG 0x15;  
sfrb PORTB=0x18;  
sfrb PINA=0x19;  
sfrb DDRA=0x1a;  
sfrb PORTA=0x1b;  
sfrb EECDR=0x1c;  
sfrb EEDR=0x1d;  
sfrb EEARL=0x1e;  
sfrb EEARH=0x1f;  
sfrw EEAR=0x1e; // 16 bit access  
sfrb SFIOR=0x20;  
sfrb WDTCSR=0x21;  
sfrb OCDR=0x22;
```

Microsoft API 및 참조 카탈로그

C/C++ 전처리기 참조

https://msdn.microsoft.com/ko-KR/library/y4skk93w.aspx

Microsoft | Developer Network

다운로드 프로그램 커뮤니티 설명서



- C++의 진화
- C++ 언어 참조
- C/C++ 전처리기 참조
  - 전처리기
  - 문법 요약
  - Pragma 지시문 및 \_\_Pragma 키워드
- C 언어 참조
- C 런타임 라이브러리 참조
- C++ 표준 라이브러리
- Component Extensions for Runtime Platforms
- C++ Attributes Reference
- SafeInt 라이브러리

# C/C++ 전처리기 참조

Visual Studio 2015 | 다른 버전

게시 날짜: 2016년 4월  
Visual Studio 2017 에 대한 최신 설명서는 Visual Studio 2017 설명서를 참조하세요.

C/C++ 전처리기 참조에서  
당 파일에 대한 준비 작업을  
고, 코드 섹션에 시스템별

## 단원 내용

전처리기 지시문  
여러 실행 환경에서 소스 코

전처리기 연산자  
#define 지시문의 컨텍스트

미리 정의된 매크로  
ANSI 및 Microsoft C++에

Pragma  
각 컴파일러가 C 및 C++ 소

## 관련 단원

C++ 언어 참조  
C++ 언어의 Microsoft 구

- 변환 단계
- 전처리기 지시문
    - #define 지시문
    - #error 지시문
    - #if, #elif, #else, and #endif 지시문
    - #ifdef 및 #ifndef 지시문
    - #import 지시문
    - #include 지시문
    - #line 지시문
    - Null 지시문
    - #undef 지시문
    - #using 지시문
  - 전처리 연산자
  - 매크로

# 전처리기 지시문

Visual Studio 2015 | 다른 버전

게시 날짜: 2016년 4월  
Visual Studio 2017 에 대한 최신 설명서는 Visual Studio 2017 설명서를 참조하세요.

#define 및 #ifndef와 같은 전처리기 지시문은 여러 실행 환경에서 소스 프로그램을 변경하기 쉽고 컴파일하기 용이하게 됩니다. 소스 파일의 지시문은 특정 작업을 수행하도록 전처리기에 지시합니다. 예를 들어 전처리기는 텍스트에서 토글을 소스 파일에 삽입하거나, 텍스트 섹션을 제거하여 파일 일부의 컴파일을 억제할 수 있습니다. 전처리기 코드 줄은 매크로입니다. 따라서 매크로가 전처리기 명령처럼 보이는 항목으로 확장되는 경우 해당 명령은 전처리기에서 인식되지 않습니다.

전처리기 문은 이스케이프 시퀀스가 지원되지 않는 것을 제외하고 소스 파일 문과 동일한 문자 집합을 사용합니다. 전처리기는 실행 문자 집합과 동일합니다. 전처리기는 음수 문자 값도 인식합니다.

전처리기는 다음 지시문을 인식합니다.

#define	#error	#import	#undef
#elif	#if	#include	#using
#else	#ifdef	#line	#endif
#ifndef	#pragma		

숫자 기호(#)는 지시문이 포함된 줄에서 공백이 아닌 첫 번째 문자여야 합니다. 공백 문자는 숫자 기호와 지시문의 첫 문자입니다. 일부 지시문에는 인수 또는 값이 포함됩니다. 지시문 다음에 오는 모든 텍스트(지시문의 일부인 인수 또는 값 제외)는 앞에 볼거나 주석 구분 기호(/\*)로 묶여야 합니다. 전처리기 지시문이 포함된 여러 줄은 줄의 끝 표식 바로 앞에 백슬러시입니다.

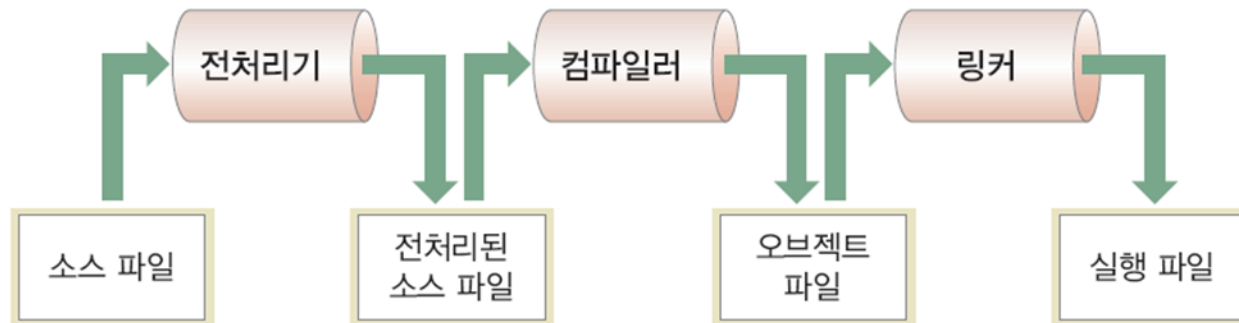
## 4. IO Port 제어 (Preprocessor)

- 전처리기(preprocessor)

- 소스(source) 프로그램을 오브젝트(object) 프로그램으로 컴파일하기 전에 수행되는 프로그램
- 소스 파일이 컴파일 될 수 있도록 준비하는 역할

- 전처리기 지시자

- 전처리기에게 내리는 지시자: '#'으로 시작
- 끝에 ;을 붙이지 않음
- 한 행에 한 지시자만 쓸 수 있음



## 4. IO Port 제어 (Preprocessor)

### ● #define 지시자

- 지정한 기호 상수를 프로그래머가 정의한 치환 문자열로 대체
- 매크로 상수
  - 인수 없이 단순히 치환만 함

형식

매크로 상수

#define 매크로명 치환할 값

예)

```
#define MAX 100           // MAX를 100으로 정의
#define NUM MAX-1         // NUM을 MAX-1 → 100-1로 정의
```

예)

```
#define PI 3.1415
area = PI * r * r;
```

## 4. IO Port 제어 (Preprocessor)

- 매크로 상수의 장점

- 프로그램 가독성 증가, 프로그램 수정 용이
- 100을 1000으로 변경하려면?

매크로 상수 사용시 매크로 정의만 수정하면 됨

```
// 매크로 사용 않은 경우
for (i=1; i<=100; i++)
:
avg = sum / 100;
```

```
// 매크로 사용 경우
#define MAX 100
for (i=1; i<=MAX; i++)
:
avg = sum / MAX;
```

- 정수 1(논리값 참에 해당) 보다는 TRUE라는 단어가 이해하기 쉬움

```
// 매크로 사용 않은 경우
if (leapyear == 1)
    printf("윤년");
```

```
// 매크로 사용 경우
#define TRUE 1

if (leapyear == TRUE)
    printf("윤년");
```

## 4. IO Port 제어 (Preprocessor)

### ● 매크로 함수

- 함수처럼 인수를 가짐
- 전처리기는  
매크로 함수 호출 자리를 정의된 함수 내용으로 대체함  
이 때 매크로 함수의 인수가 치환 내용 해당 자리로 매핑되어 대체됨
  - 함수처럼 실제 함수 호출이 발생하지 않고 전처리 단계에서 삽입된 코드가 실행되므로 실행 시 속도가 빠름

형식

매크로 함수

#define 매크로 함수명(인수1, 인수2, ...) 치환할 내용

예

```
#define ADD(x, y) ((x) + (y))
```

```
#define SQUARE(x) ((x) * (x))
```



## 4. IO Port 제어 (Preprocessor)

### ● #include 지시자

- 특정 파일을 현재 파일에 포함하기 위해 사용
- 특히 함수의 원형이 있는 헤더 파일 포함에 사용

형식


#include 지시자

#include <라이브러리 헤더 파일> 또는 #include "라이브러리 헤더 파일"

#include "사용자 정의 헤더 파일"

#include "사용자 정의 파일"

예

#include <stdio.h>  라이브러리 헤더 파일

#include <stdlib.h>

#include "C:\user\userlib.h" ← 사용자 정의 헤더 파일

#include "test.cpp" ← 사용자 정의 파일

## 4. IO Port 제어 (Preprocessor)

---

### ● 조건부 컴파일 지시자

- 특정 조건을 만족할 때만 특정 코드를 프로그램에 삽입함  
: 전처리를 마치면 필요한 코드만 삽입됨으로써,  
선택된 코드만 컴파일 된다.
- `#if`
- `#ifdef`
- `#ifndef`
- `#undef`

## 4. IO Port 제어 (Preprocessor)

- 조건식 결과값에 따라 특정 문장을 선택적으로 소스 파일에 삽입

형식 `#if`를 이용한 조건부 컴파일

```
#if 조건식
    문장 1;
#else
    문장 2;
#endif
```

조건식이 참 → '문장 1' 을 삽입  
거짓 → '문장 2' 를 삽입

- 프로그램이 한국, 미국, 유럽의 세 가지 버전으로 존재 시  
국가 별로 서로 다른 헤더 파일이 포함되게

```
#if (NATION == 1)
    #include "korea.h"
#elif (NATION == 2)
    #include "usa.h"
#else
    #include "europe.h"
#endif
```

또 다른 조건문을 검사하려면  
`#else`와 `#if`를 결합한 `#elif` 문을 사용

## 4. IO Port 제어 (Preprocessor)

### 형식 #ifdef를 이용한 조건부 컴파일

#ifdef 매크로명  
문장;

매크로명의 매크로가 정의되어 있다면 소스 코드에 포함할 내용

#else  
문장 2;

- 매크로가 정의되어 있지 않을 때 포함될 내용
- 필요 없다면 생략 가능
- #elif 는 사용 불가

#endif

예 #ifdef ADD  
printf("%d", x + y);  
#else  
printf("%d", x - y);  
#endif

### 형식 #undef를 이용한 매크로 정의 해제

#undef 매크로명

- 매크로 정의를 해제
- 이전에 정의된 매크로 정의를 무효화하고 새로 정의할 때 사용

## 4. IO Port 제어 (Preprocessor)

### ● #ifndef

- "if not defined"의 약어
- 매크로가 정의되어 있지 않은 경우에만  
**#ifndef ~#endif** 사이의 문장을 소스 파일에 삽입하여 컴파일 되게 함

```
1  #include <stdio.h>
2  #define PI 3.141592
3  #define R 5
4
5  int main()
6  {
7      double area;
8
9      #ifdef PI
10         printf("PI = 3.141592\n");
11     #endif
12
13     #undef R          // R의 정의 해제
14     #define R 3       // R을 3으로 재정의
15
16     area = PI * R * R;
17     printf("Radius = %d\nArea = %.2lf\n", R, area);
18
19     return 0;
20 }
```

실행결과

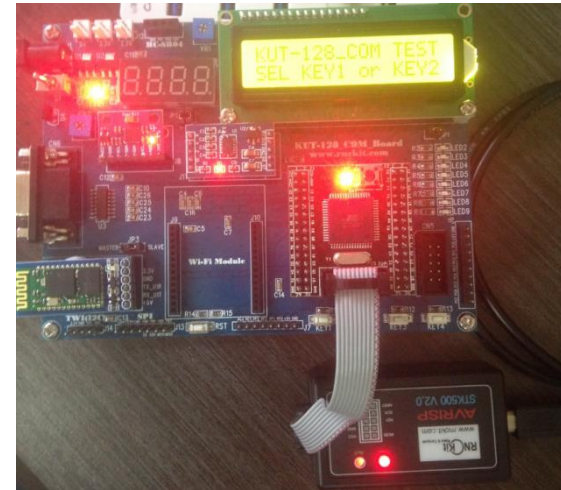
PI = 3.141592

Radius = 3

Area = 28.27

## 4. I/O 포트 사용법 요약

### 코드비전 설치 및 프로젝트 생성



### 코드비전 화면이 안 나올 경우

- : codevision 화면 닫기
- : 파일탐색기에서 숨김파일 보기
- : C – Program data – HP InfoTech
  - Codevision 내에서 .ini 파일 모두 삭제
- : codevision 다시 열기



## 4. I/O 포트 사용법 요약

- 사용할 포트에 대한 비트 단위의 입출력 방향 설정(0 : 입력, 1 : 출력)  
→ 사용 레지스터 **DDRx** ( x : A, B, C, D, E, F, G)
- 포트에 연결된 출력장치에 대한 초기상태 출력  
→ 사용 레지스터 **PORTx** ( x : A, B, C, D, E, F, G)
- 내부 풀업저항을 사용하는 경우에는 추가적으로 사용 설정  
→ 사용 레지스터 **SFIOR, PORTx** ( x : A, B, C, D, E, F, G)
- 입출력에 따라 입력 레지스터 또는 출력 레지스터를 이용  
→ 포트에 값을 출력할 때 **PORTx** ( x : A, B, C, D, E, F, G) 이용  
→ 포트로부터 값을 입력할 때 **PINx** ( x : A, B, C, D, E, F, G) 이용

## 4. LED 출력회로

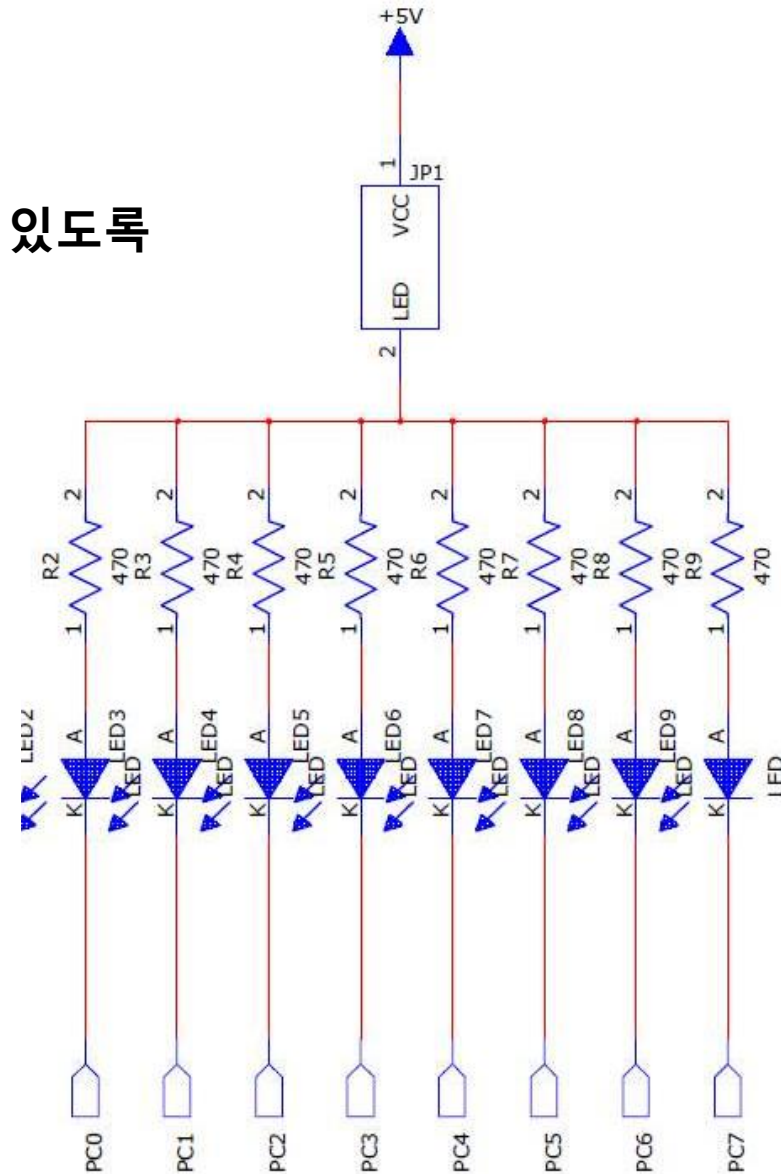
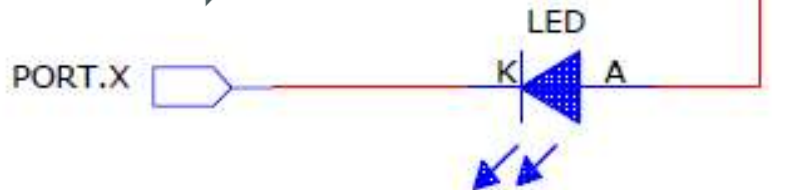
포트 출력에 대한 실험

: 포트 C에 8개의 LED를 연결하였으며

: 풀업저항에 연결된 전원을 ON/OFF할 수 있도록  
점퍼 JP1을 장착

5V → OFF

0V → ON





## 4. LED 출력회로

- (예제 3-1) LED ON/OFF 하기

LED가 연결되어 있는 포트 C에 **0x55 (0101 0101)**를 출력하여 다음과 같이 LED가 ON/OFF 되도록 프로그램을 작성하여라.

DDRC	DDRC7	DDRC6	DDRC5	DDRC4	DDRC3	DDRC2	DDRC1	DDRC0
방향	output	output	output	output	output	output	output	output
출력 값	1	1	1	1	1	1	1	1
DDRC = 0xFF								

PORC	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
연결 LED	LED7	LED6	LED5	LED4	LED3	LED2	LED1	LED0
출력 값	0	1	0	1	0	1	0	1
LED 상태	ON	OFF	ON	OFF	ON	OFF	ON	OFF
PORC = 0x55								

## 4. LED 출력회로

- (예제 3-1) LED ON/OFF 하기

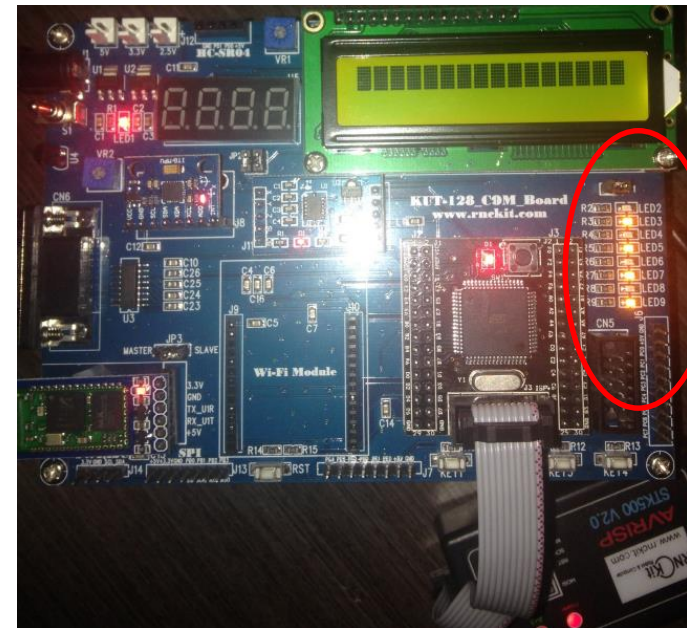
```
#include <mega128a.h>    // LED 4개 ON, 4개 OFF
```

```
void main(void)
```

```
{  DDRC = 0xFF;           // 포트C 출력 설정  
  PORTC = 0x55;          // 포트 C 0x55(또는 0101 0101) 출력
```

```
while(1);
```

```
}
```



## 4. LED 출력회로

- (예제 3-2) LED ON/OFF 점멸하기

LED가 연결되어 있는 포트 C에 **0x55(0101 0101)**와

**0xAA(1010 1010)**를 번갈아 출력하면서

점멸시키는 프로그램을 작성하라. 라이브러리 함수 **delay\_ms()**사용

👉 라이브러리 시간지연함수

**Void delay\_us(unsigned int n);**    //us단위 시간 지연 함수

**Void delay\_ms(unsigned int n);**    //ms단위 시간 지연 함수

**#include <delay.h>**    // delay\_ms(), delay\_us() 헤더파일  
                          // delay\_ms(지연값), delay\_us(지연값)

## 4. LED 출력회로

---

- (예제 3-2) LED ON/OFF 점멸하기

```
#include <mega128.h>      // LED 점멸하기(라이브러리 딜레이 함수 이용)
#include <delay.h>         // delay_ms(), delay_us() 정의 헤더파일

void main(void)
{
    DDRC = 0xFF;           // 포트C 출력 설정

    while(1){              // 무한 루프
        PORTC = 0x55;      // 0101 0101 출력
        delay_ms(500);     // 500ms 대기

        PORTC = 0xAA;      // 1010 1010 출력
        delay_ms(500);     // 500ms 대기
    }
}
```

## 4. LED 출력회로

---

- (예제 3-3) LED 쉬프트 하면서 순차 점멸하기 (1)

led = 0xFE (1111 1110)로 초기 설정한 후,

왼쪽으로 쉬프트 하면서 led값을 포트 C에 출력하는 프로그램 작성  
단, 쉬프트 8회를 한 주기로 하여 계속 반복하도록 한다.

led = 0xFE (1111 1110)

led = 0xFC (1111 1100)

led = 0xF8 (1111 1000)

led = 0xF0 (1111 0000)

led = 0xE0 (1110 0000)

led = 0xC0 (1100 0000)

led = 0x80 (1000 0000)

led = 0x00 (0000 0000)

## 4. LED 출력회로

---

- (예제 3-3) LED 쉬프트 하면서 순차 점멸하기 (1)

```
#include <mega128.h>
#include <delay.h>    // 딜레이 함수 정의 헤더 파일

void main(void)    // LED 순차 점멸하기(쉬프트 연산자)
{
    int i;
    unsigned char led;

    DDRC = 0xFF;                // 포트C 출력 설정
    while(1){                  // 무한 루프
        led = 0xFE;             // led 초기값
        for(i = 0; i < 8; i++){ // 8회 실행
            PORTC = led;         // led 값 출력
            delay_ms(500);        // 500ms 딜레이
            led = led << 1;      // 1비트 왼쪽 쉬프트
        }
    }
}
```

## 4. LED 출력회로

---

- (예제 3-4) LED 쉬프트 하면서 순차 점멸하기 (2)

led = 0xFE (1111 1110)로 초기 설정한 후,

1비트씩 왼쪽으로 쉬프트 한 후 OR연산을 통해 최하위 비트를  
강제로 1로 하여 포트 C에 출력하는 프로그램을 작성  
단, 쉬프트 8회를 한 주기로 하여 계속 반복하도록 한다.

led = 0xFE (1111 1110)

led = 0xFD (1111 1101)

led = 0xFB (1111 1011)

led = 0xF7 (1111 0111)

led = 0xEF (1110 1111)

led = 0xDF (1101 1111)

led = 0xBF (1011 1111)

led = 0x7F (0111 1111)

## 4. LED 출력회로

- (예제 3-4) LED 쉬프트 하면서 순차 점멸하기 (2)

```
#include <mega128.h>
#include <delay.h>    // 딜레이 함수 정의 헤더 파일

void main(void)    // LED 순차 점멸하기(쉬프트 연산자)
{
    int i;
    unsigned char led;

    DDRC = 0xFF;                // 포트C 출력 설정
    while(1){                  // 무한 루프
        led = 0xFE;             // led 초기값
        for(i = 0; i < 8; i++){ // 8회 실행
            PORTC = led;         // led 값 출력
            delay_ms(500);        // 500ms 딜레이
            led = led << 1;       // 1비트 왼쪽 쉬프트
            led = led | 0x01;     // 최하위 비트 셋
        }
    }
}
```



## 4. LED 출력회로

- (예제 3-4-1) LED 순차 점멸하기[배열이용]

```
#include <mega128.h>
#include <delay.h>    // 딜레이 함수 정의 헤더 파일
const unsigned char led[8] = {0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF, 0x7F};
```

```
void main(void)    // 배열 이용하여 순차 점멸하기(Flash 형 : 값 고정 이용)
{
    int    i;

    DDRC = 0xFF;    // 포트C 출력 설정

    while(1){        // 무한 루프
        for(i = 0; i < 8; i++){
            PORTC = led[i];    // i번째 led값 출력
            delay_ms(500);    // 500ms 딜레이
        }
    }
}
```

상수형 변수

프로그램이 실행되는 동안 값이 변하지 않는 상수형 변수는 플래시 메모리에 저장되며, 이를 위해 "flash" 또는 "const"를 데이터 형 앞에 붙여 사용  
반드시 전역변수로 함수 밖에서 선언

## 4. LED 출력회로

---

- (예제 3-5) LED 하나만 점멸하기

포트 C에 0x55를 출력하고, 포트 C의 0비트(**PORTC. 0**)에 연결되어 있는 LED0만을 점멸시키는 프로그램을 작성하라.

## 4. LED 출력회로

- (예제 3-5) LED 하나만 점멸하기

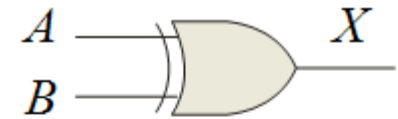
포트 C에 0x55를 출력하고, 포트 C의 0비트(PORTC.0)에 연결되어 있는 LED0만을 점멸시키는 프로그램을 작성하라.

```
#include <mega128.h>
#include <delay.h>    // 딜레이 함수 정의 헤더 파일

void main(void)
{
    DDRC = 0xFF;           // 포트C 출력 설정

    PORTC = 0x55;          // 포트C 0x55 출력

    while(1){              // 무한 루프
        PORTC = PORTC ^ 0x01; // PORTC의 0비트만 반전
        delay_ms(500);       // 500ms 딜레이
    }
}
```



Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

```
0x55 0101 0101
0x01 0000 0001 (XOR)
-----
0101 0100
0x01 0000 0001 (XOR)
0101 0101
```

## 4. LED 출력회로

---



### [ 비트 단위 지정 ]

비트 단위 조작은 64개의 I/O 레지스터만 가능하며 (데이터 메모리 맵 참조),  
비트 지정은 [ I/O 레지스터 어드레스. 비트번호 ]의 형식을 갖는다.

그러나 컴파일러의 헤더파일([mega128a.h](#))에는  
레지스터의 어드레스가 레지스터명으로 모두 선언되어 있으므로  
[ I/O 레지스터명.비트번호 ]의 형식으로 비트를 지정

## 4. LED 출력회로

- (예제 3-5-1) LED 하나만 점멸하기

```
#include <mega128.h>
```

```
#include <delay.h>
```

```
void main(void)
```

```
{
```

```
    DDRC = 0xFF;        // 포트C 출력 설정
```

```
    PORTC = 0x55;       // 포트C 0x55 출력
```

```
    while(1){           // 무한 루프
```

```
        PORTC.0 = 1;     // PORTC의 0비트 OFF
```

```
        delay_ms(500);    // 500ms 딜레이
```

```
        PORTC.0 = 0;     // PORTC의 0비트 ON
```

```
        delay_ms(500);    // 500ms 딜레이
```

```
    }
```

```
}
```

[ 비트 단위 지정 ]

## 4. LED 출력회로

- (예제 3-5-2) LED 하나만 점멸하기

```
#include <mega128.h>
```

```
#include <delay.h>
```

```
#define LED0 PORTC.0
```

```
#define ON 0
```

```
#define OFF 1
```

```
void main(void) // LED0, ON, OFF 값 정의(define문 이용)
```

```
{
```

```
    DDRC = 0xFF; // 포트C 출력 설정
```

```
    PORTC = 0x55; // 포트C 0x55 출력
```

```
    while(1){
```

```
        LED0 = 1; // LED0 OFF
```

```
        delay_ms(500); // 500ms 딜레이
```

```
        LED0 = 0; // LED0 ON
```

```
        delay_ms(500); // 500ms 딜레이
```

```
    }
```

```
}
```

**[ 비트 단위 지정 ]**

## 4. LED 출력회로

---

- (예제 3-5-3) LED 하나만 점멸하기

```
#include <mega128.h>
```

```
#include <delay.h>
```

```
void main(void)                // 포트C의 0비트만 점멸(비트 반전 연산자 이용)
{
    DDRC = 0xFF;               // 포트C 출력 설정

    PORTC = 0x55;              // 포트C 0x55 출력

    while(1){
        PORTC.0 = !PORTC.0;    // PORTC의 0비트 반전
        delay_ms(500);         // 500ms 딜레이
    }
}
```

[ 비트 단위 지정 ]

## 4. LED 출력회로

---

- (예제 3-5-4) LED 하나만 점멸하기

```
#include <mega128.h>
#include <delay.h>
```

```
#define LED0 PORTC.0
```

```
void main(void)          // LED0 정의(비트반전 이용)
```

```
{
    DDRC = 0xFF;          // 포트C 출력 설정

    PORTC = 0x55;         // 포트C 0x55 출력

    while(1){
        LED0 = !LED0;     // LED0 반전 출력          [ 비트 단위 지정 ]
        delay_ms(500);     // 500ms 딜레이
    }
}
```



## 4. 스위치 입력회로

4개의 스위치 : Pull up + Capacitor

- Capacitor는 De-bouncing 역할

스위치 :

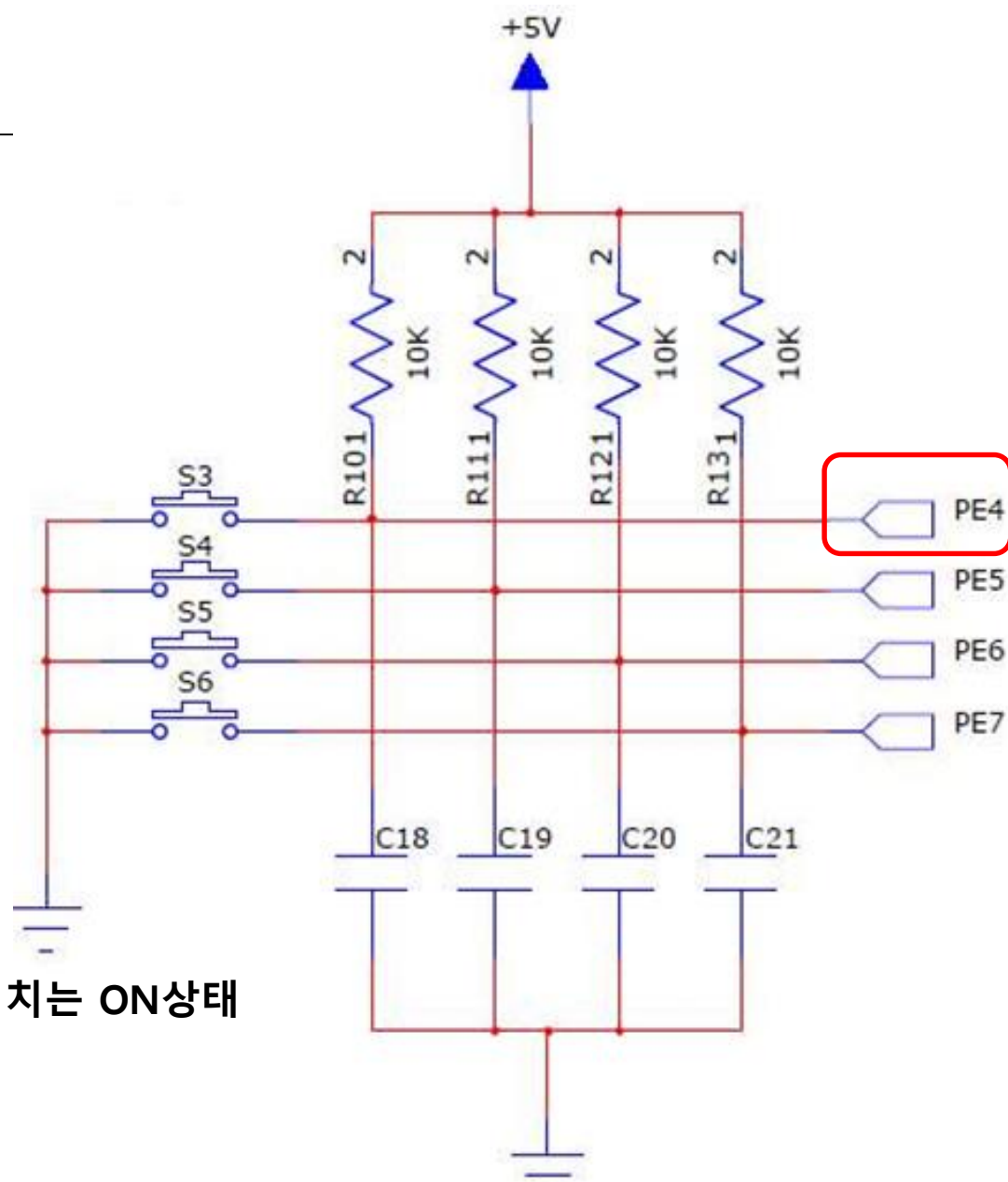
- 스위치가 OFF인 경우

Pull up 저항의 5V가 포트에 입력,  
5V(논리1)가 되며

- 스위치가 ON인 경우

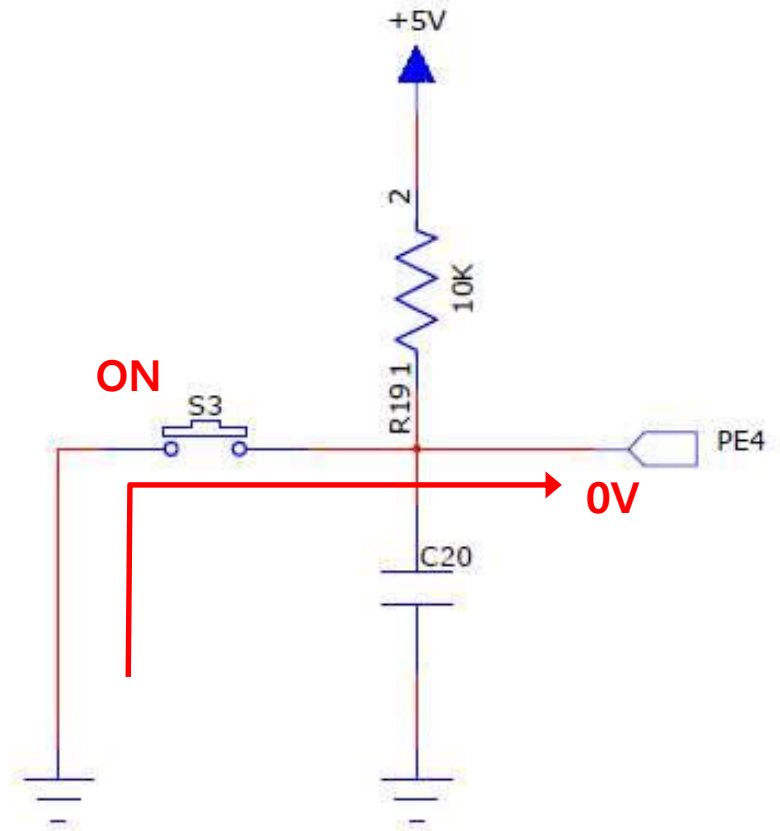
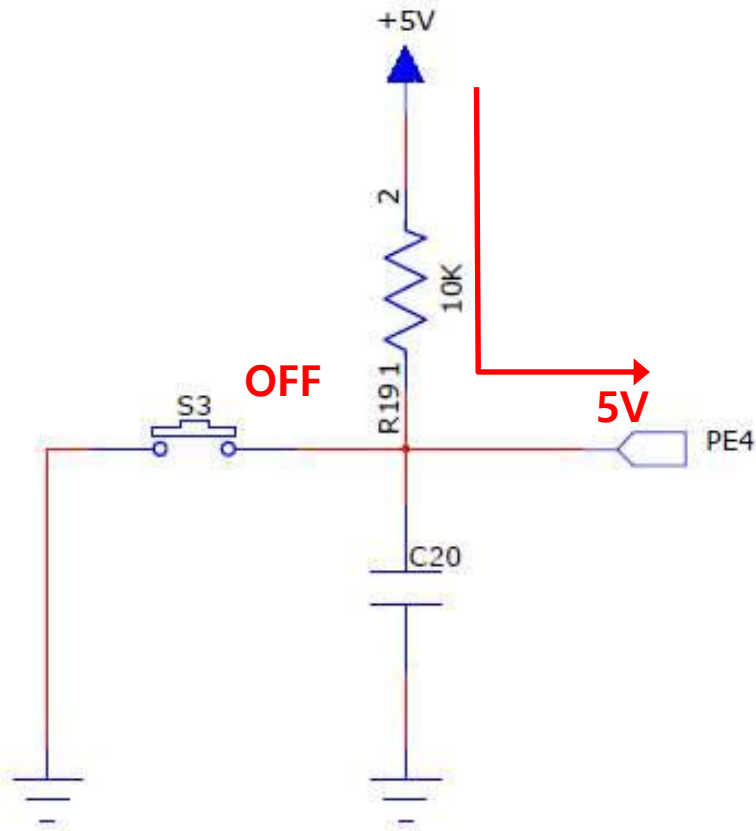
스위치 양쪽의 전압이 같아 지면서  
0V(논리값0)의 값이 포트에 입력

- 스위치에 연결된 비트 값이 0이면 스위치는 ON상태  
1이면 스위치는 OFF상태를 나타냄



## 4. 스위치 입력회로

스위치 ON/OFF 동작에 따른 포트 값



## 4. 스위치 입력회로

- (예제 3-6) 스위치 상태 읽기

PE4에 연결되어 있는 SW1이 ON이면 (0이 되면),

포트 C에 연결되어 있는 LED를 모두 ON시키고,

SW1이 OFF이면 포트 C에 연결되어 있는 LED를 모두 OFF시킴

☞ <힌트>

### \* 초기설정

포트E 입력 설정 (SW)

포트C 출력 설정 (LED)

### \* 반복부분

- 포트E 읽기(PINE이용)

- PE4 비트 1(SW OFF)이면 LED 모두 OFF  
0(SW ON)이면 LED 모두 ON

## 4. 스위치 입력회로

- (예제 3-6) 스위치 상태 읽기

**PE4**에 연결되어 있는 SW1이 ON이면

```
#include <mega128.h>
```

```
void main(void)
```

```
{
```

```
    unsigned char  sw;
```

```
    DDRC = 0xFF;
```

```
// 포트C 출력 설정
```

```
    DDRE = 0x00;
```

```
// 포트E 입력 설정
```

```
    PORTC = 0xFF;
```

```
// LED 모두 off (output=1, off)
```

```
    while(1){
```

```
        sw = PINE & 0b00010000;
```

```
// PE4비트 추출 (PORT E의 4번 입력핀)
```

```
        if(sw != 0) PORTC = 0xFF;
```

```
// 입력이 1이면, SW OFF -> LED 모두 OFF
```

```
        else PORTC = 0x00;
```

```
// 입력이 0이면, SW ON -> LED 모두 ON
```

```
    }
```

```
}
```

## 4. 스위치 입력회로

---

- (예제 3-6-1) 스위치 상태 읽기

```
#include <mega128.h>
```

```
void main(void)                                // 비트 단위 입력처리
{
    DDRC = 0xFF;                                // 포트C 출력=1 설정
    DDRE = 0x00;                                // 포트E 입력=0 설정

    PORTC = 0xFF;                               // LED 모두 off

    while(1){

        if(PINE.4 == 1) PORTC = 0xFF;          // PE 4입력이 1이면, SW OFF -> LED OFF
        else PORTC = 0x00;                      // PE 4입력이 0이면, SW ON -> LED ON
    }
}
```

## 4. 스위치 입력회로

- (예제 3-6-2) 스위치 상태 읽기

```
#include <mega128.h>
```

```
#define SW0 PINE.4
```

```
#define ON 0
```

```
void main(void)                                // 비트 단위 입력처리
{
    DDRC = 0xFF;                                // 포트C 출력 설정
    DDRE = 0x00;                                // 포트E 입력 설정

    PORTC = 0xFF;                                // LED 모두 off

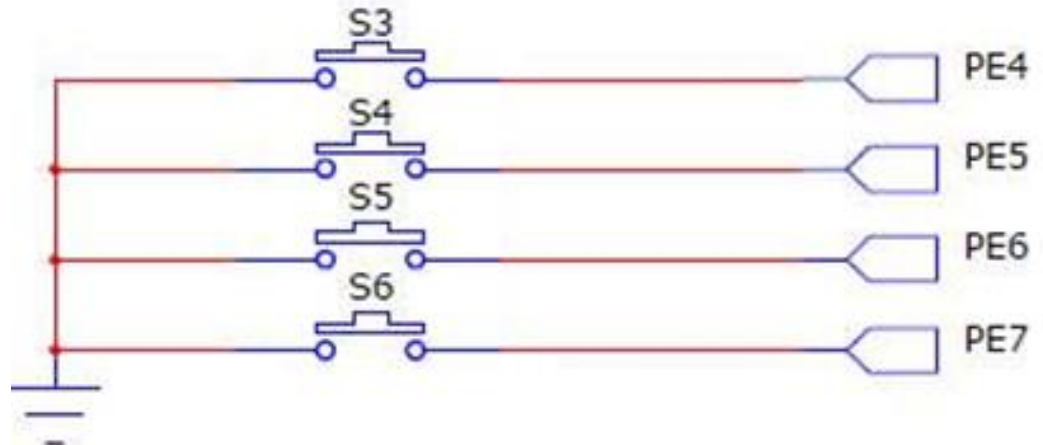
    while(1){
        if(SW0 != ON) PORTC = 0xFF; // PE 4입력이 1이면, SW OFF -> LED OFF
        else PORTC = 0x00;           // PE 4입력이 0이면, SW ON -> LED ON
    }
}
```

## 4. 스위치 입력회로

- (예제 3-7) 스위치 상태에 따라 다르게 출력하기 (**switch문** -> **if문**)

4개의 스위치 입력에 따라 다음과 같이 동작하는 프로그램을 작성하라.

SW6	SW5	SW4	SW3	LED
OFF	OFF	OFF	ON	All ON
OFF	OFF	ON	OFF	All OFF
OFF	ON	OFF	OFF	짝수 LED ON
ON	OFF	OFF	OFF	홀수 LED OFF



- (예제 3-7) 스위치 상태에 따라 다르게 출력하기

```
- #include <mega128.h>
```

```
void main(void) {
    unsigned char key;
```

```
    DDRC = 0xFF;
```

```
// 포트 C 출력 설정
```

```
    DDRE = 0x00;
```

```
// 포트 E 입력 설정
```

```
    PORTC = 0xFF;
```

```
// LED 모두 OFF
```

```
    while(1){
```

```
        key = PINE & 0xF0;
```

```
// 포트 E 4.5.6.7 읽어오기.
```

```
        switch(key){
```

```
            case 0b11100000 :
```

```
// SW1(E.4)이 눌리면 LED 모두 ON
```

```
                PORTC = 0x00;
```

```
                break;
```

```
            case 0b11010000 :
```

```
// SW2(E.5)가 눌리면 LED 모두 OFF
```

```
                PORTC = 0xFF;
```

```
                break;
```

```
            case 0b10110000 :
```

```
// SW3(E.6)이 눌리면 짝수번째 LED ON
```

```
                PORTC = 0b01010101;
```

```
                break;
```

```
            case 0b01110000 :
```

```
// SW4(E.7)가 눌리면 홀수번째 LED ON
```

```
                PORTC = 0b10101010;
```

```
                break;
```

```
            default:
```

```
                break;
```

```
        } // end of switch
```

```
    } // end of while
```

```
} // end of main
```



## 4. 스위치 입력회로

- (예제 3-7-1) 스위치 상태에 따라 다르게 출력하기 (if문)

```
#include <mega128.h>
```

```
void main(void)
```

```
{
```

```
    unsigned char key;
```

```
    DDRC = 0xFF;
```

```
    // 포트 C 출력 설정
```

```
    DDRE = 0x00;
```

```
    // 포트 E 입력 설정
```

```
    PORTC = 0xFF;
```

```
    // LED 모두 OFF
```

```
    while(1){
```

```
        key = PINE & 0xF0;
```

```
        // 포트 E 4.5.6.7 읽어오기.
```

```
        if(key == 0b11100000) PORTC = 0x00;
```

```
        // SW1이 눌리면 LED 모두 ON
```

```
        else if(key == 0b11010000) PORTC = 0xFF;
```

```
        // SW2가 눌리면 LED 모두 OFF
```

```
        else if(key == 0b10110000) PORTC = 0b01010101;
```

```
        // SW3이 눌리면 짝수번째 LED ON
```

```
        else if(key == 0b01110000) PORTC = 0b10101010;
```

```
        // SW4가 눌리면 홀수번째 LED ON
```

```
    }
```

```
}
```

## 4. 스위치 입력회로

- (예제 3-8) 스위치 OFF에서 ON으로 될 때마다 LED 순차 점멸하기  
: PE4에 연결되어 있는 SW1이 OFF(1)에서 ON(0)으로 될 때마다  
포트 C에 연결되어 있는 8개의 LED를 순차 점멸하는 프로그램을 작성  
➔ 짧은 시간차를 두고 스위치의 상태를 읽어  
두 상태의 값으로부터 스위치의 상태를 아래 표와 같이 파악.  
➔ 맨 처음에만 연속해서 두 번 스위치의 상태를 읽고  
그 다음부터는 1회씩 읽으면서 스위치의 상태파악

스위치 상태	이전 상태	현재 상태	비 고
OFF	OFF	OFF	
OFF➔ ON	OFF	ON	Falling Edge (1➔ 0)
ON	ON	ON	
ON ➔ OFF	ON	OFF	Rising Edge (0➔1)

- (예제 3-8) 스위치 OFF → ON 될 때마다 LED 순차 점멸하기

```
#include <mega128.h>
```

```
void main(void)
```

```
{
```

```
    unsigned char  old_sw, new_sw;
```

```
    unsigned char  led = 0xFE;
```

```
    DDRC = 0xFF;
```

```
    DDRE = 0x00;
```

```
    PORTC = led;
```

```
    old_sw = PINE & 0b00010000;
```

```
    while(1){
```

```
        new_sw = PINE & 0b00010000;
```

```
        if((old_sw != 0) && (new_sw == 0)){ // OFF(1) -> ON(0) 되는 순간 체크
```

```
            led = (led << 1) | 0x01;
```

```
            if(led == 0xFF) led = 0xFE;
```

```
            PORTC = led;
```

```
        }
```

```
        old_sw = new_sw;
```

```
    }
```

```
}
```

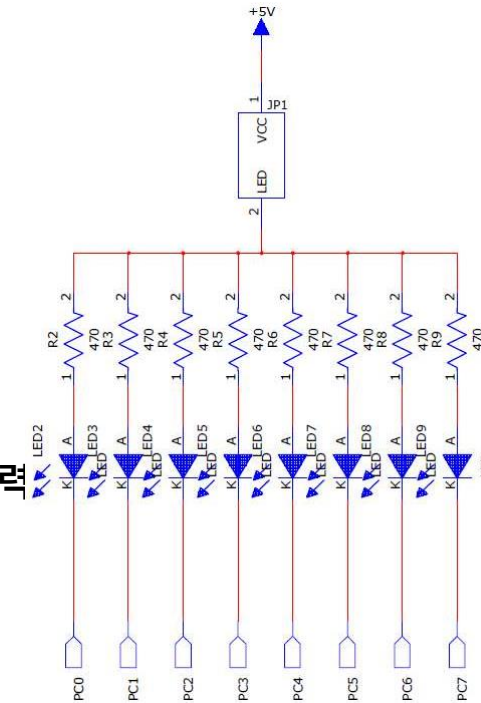
```
// 포트C 출력 설정
```

```
// 포트E 입력 설정
```

```
// 포트C.0=0 led on 초기값 출력
```

```
// PE4 SW1 상태값 추출
```

```
// 이전상태 <- 현재상태
```



- (예제 3-8-1) 스위치 OFF→ON 될 때마다 LED 순차 점멸하기

```
#include <mega128.h>

const unsigned char led[8] =
    {0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF, 0x7F};

void main(void)
{
    unsigned char o_sw, n_sw;
    unsigned char state = 0;           // LED 출력 상태(0번째)

    DDRC = 0xFF;                       // 포트C 출력 설정
    DDRE = 0x00;                       // 포트E 입력 설정

    PORTC = led[state];                // PORTC = led[0] → 0번째 출력 0
    o_sw = PINE & 0b00010000;         // SW1=PE4 상태값

    while(1){
        n_sw = PINE & 0b00010000;
        if(o_sw != 0 && n_sw == 0){    // OFF -> ON 되는 순간 체크
            state = (state + 1) % 8;    // state 다음 상태로 갱신(7 다음 0)
            PORTC = led[state];
        }
        o_sw = n_sw;                   // 이전상태 <- 현재상태
    }
}
```

<대체 구문>

```
state = state+1;
if(state==8) state=0;
```

St 값	(st+1) 값	St(새로운 값) =(st+1)%8
0	1	1
1	2	2
2	3	3
3	4	4
4	5	5
5	6	6
6	7	7
7	8	0

👉 State = (state + 1) % 8

- (예제 3-8) 스위치 OFF → ON 될 때마다 LED 순차 점멸하기

```
#include <mega128.h>
```

```
void main(void)
```

```
{
```

```
    unsigned char  old_sw, new_sw;
```

```
    unsigned char  led = 0xFE;    → unsigned short 로 변경되면 8번 후, LED 동작 안 함.
```

```
    0x00FE
```

```
        : 0000 0000 1111 1110
```

```
        : 0000 0001 1111 1101
```

```
        : 0000 0011 1111 1101
```

```
        : 0111 1111 0111 1111 (0xFF 되지 않음, if 문에 들어가지 않음)
```

```
    while(1){
```

```
        new_sw = PINE & 0b00010000;
```

```
        if((old_sw != 0) && (new_sw == 0)){ // OFF(1) -> ON(0) 되는 순간 체크
```

```
            led = (led << 1) | 0x01;    // 1비트 쉬프트, 0비트 1로 채움
```

```
            if(led == 0xFF) led = 0xFE;    // LED 모두 off상태이면 초기값 재설정
```

```
            PORTC = led;
```

```
        }
```

```
        old_sw = new_sw;    // 이전상태 <- 현재상태
```

```
    }
```

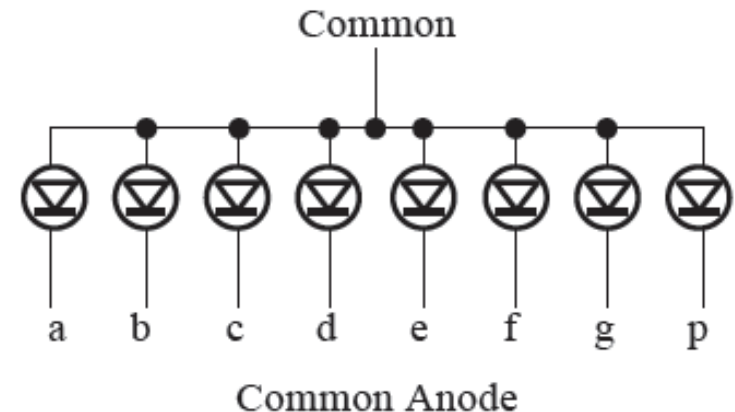
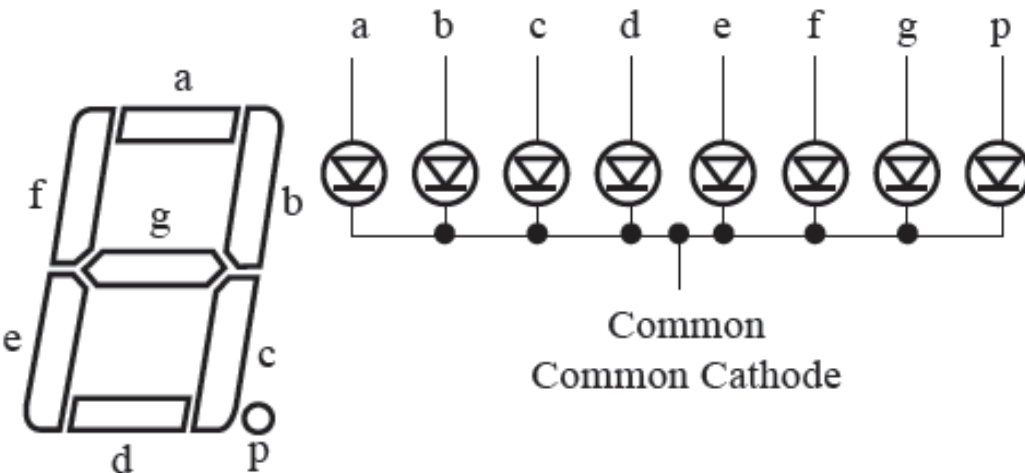
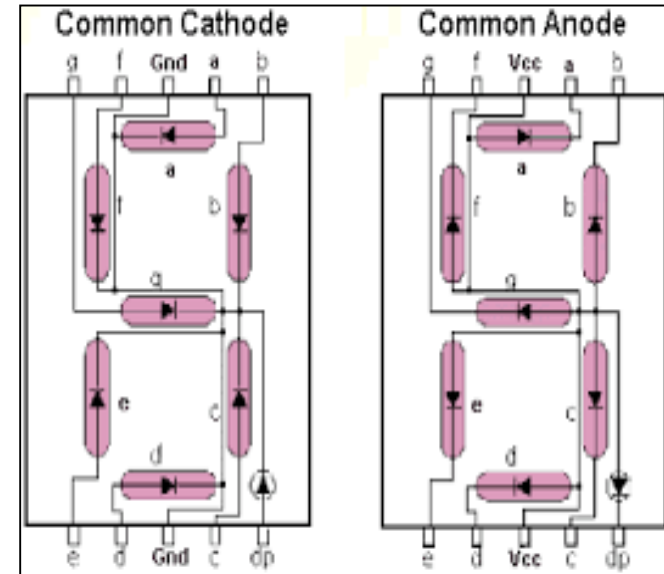
```
}
```

## 4. 7-Segment 구동회로

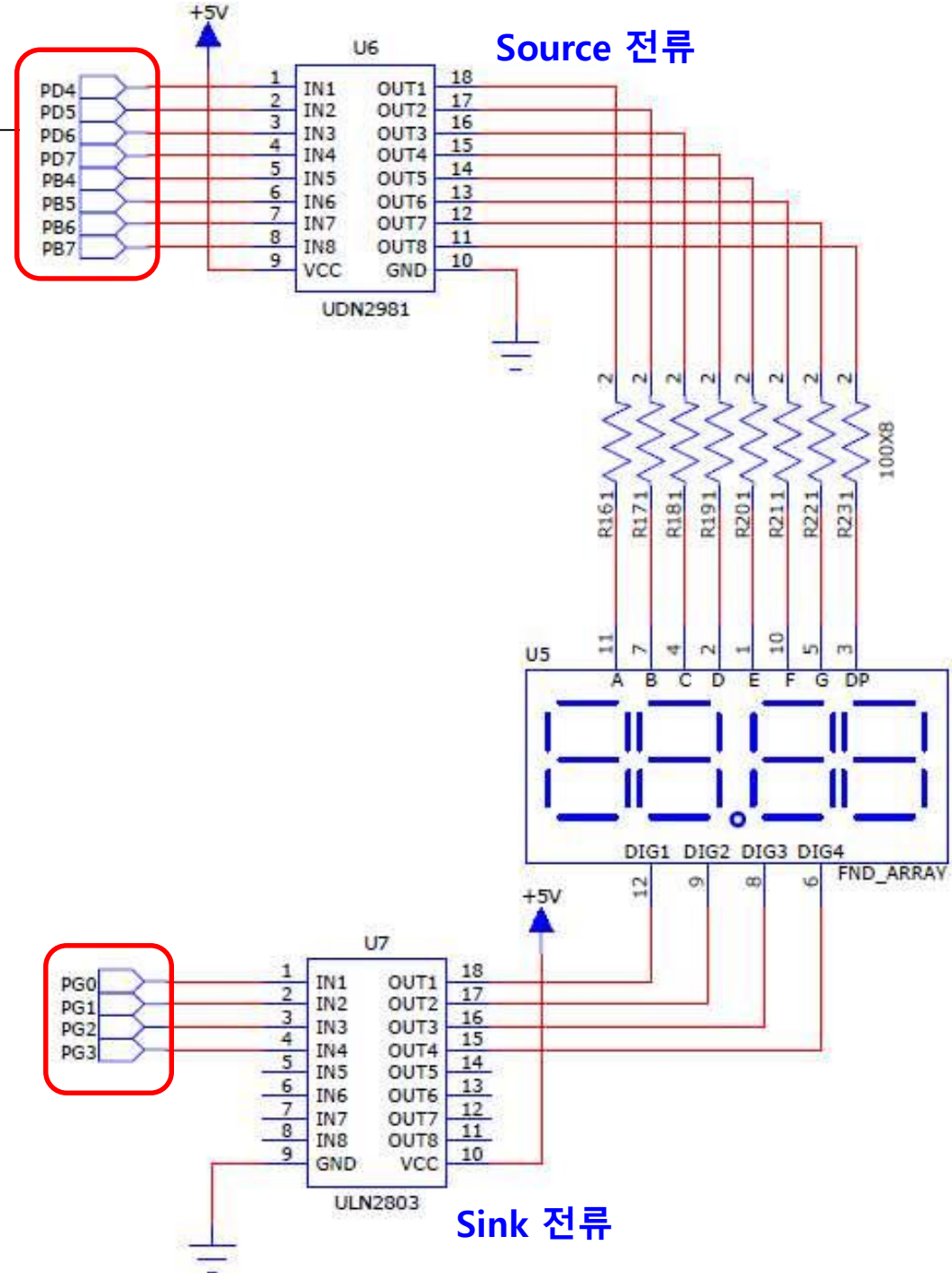
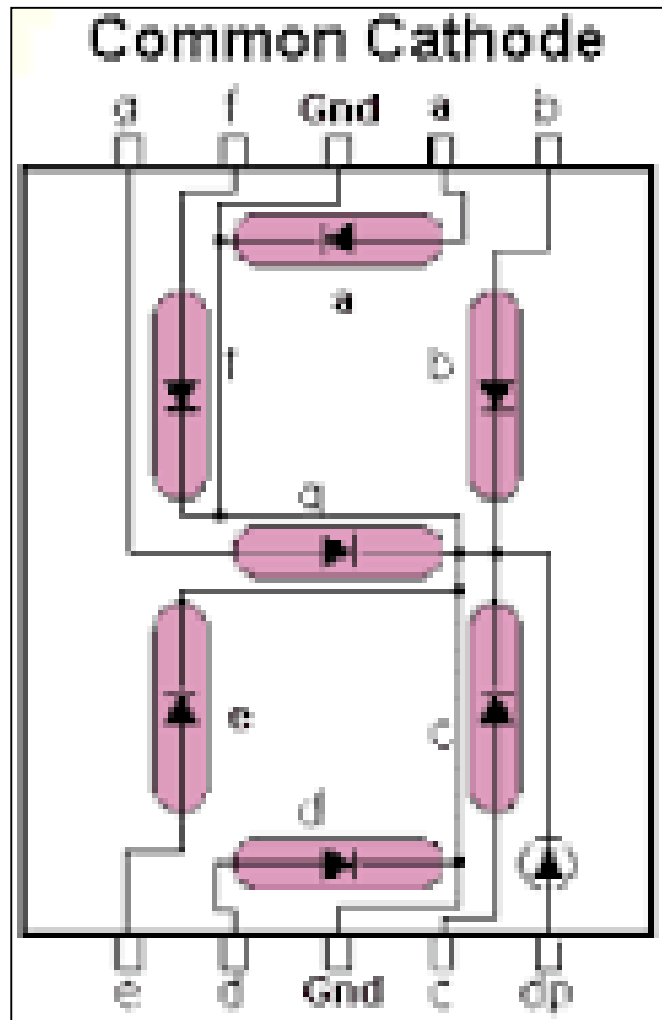
### 7-Segment : FND(Flexible Numeric Display)

FND(Flexible Numeric Display) : 숫자 표시용 소자

- 7개의 세그먼트(a, b, c, d, e, f, g)와 도트(p)로 구성
  - 각각의 세그먼트는 LED로 구성
  - CC(Common Cathode)형 : (-)극이 공통(common)
  - CA(Common Anode)형 : (+)극이 공통(anode)



## 4. 7-Segment 구동회로



## 4. 7-Segment 구동회로

- 디지털시계의 제작, A/D변환기의 변환 값 표시

: **Common Cathode** 형 4채널 Segment

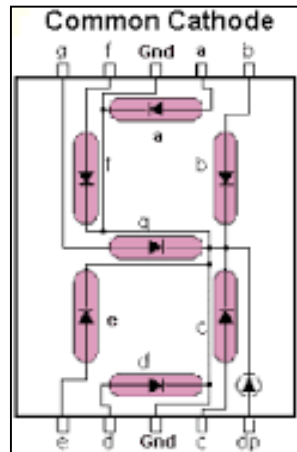
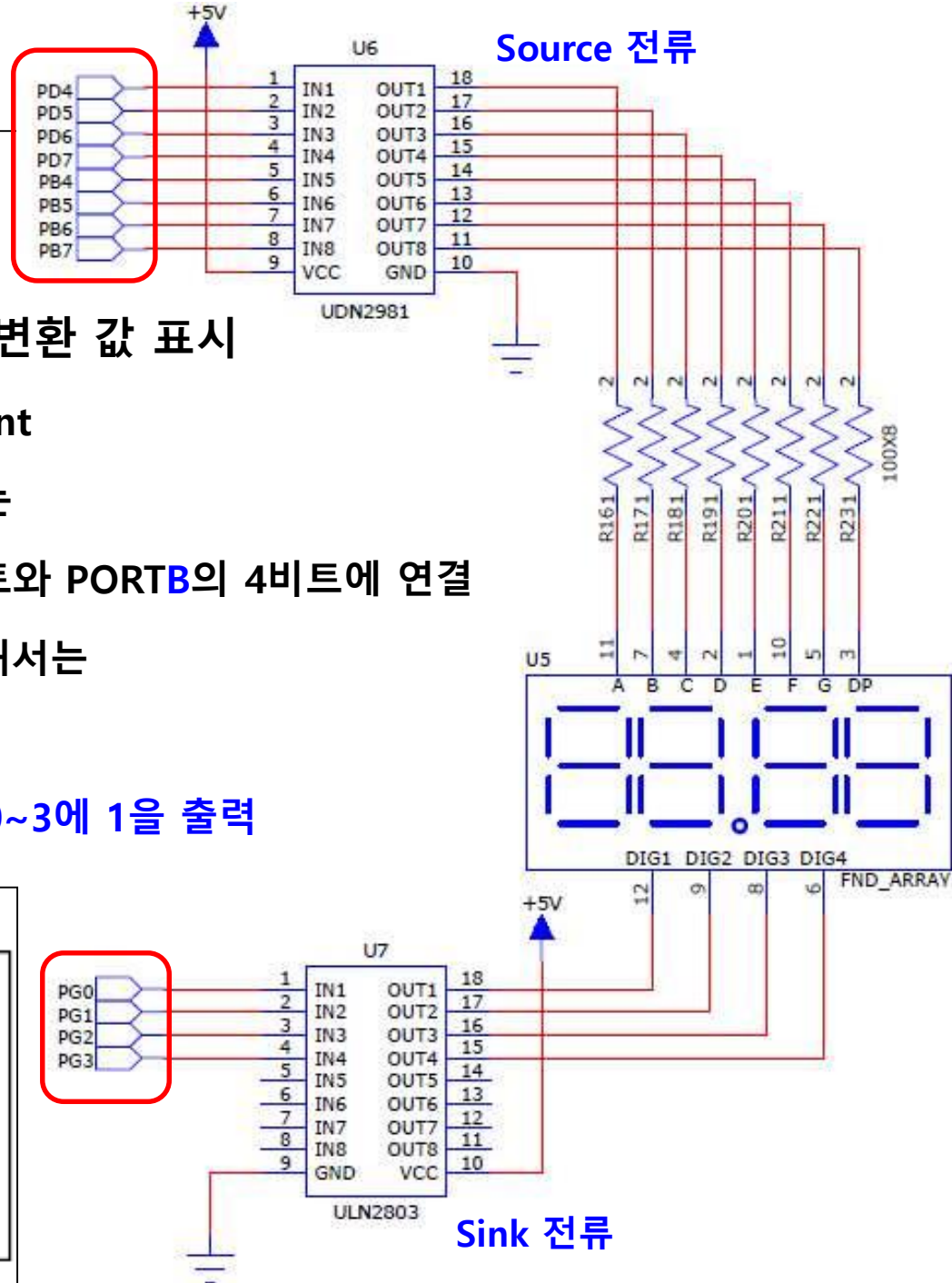
: 각 Segment LED(A,B,C,D,E,F,G,DP)는

UDN2981을 거쳐 포트 PORTD 4비트와 PORTB의 4비트에 연결

: 각 Segment의 LED를 ON시키기 위해서는

PORTB, PORTD의 해당비트에 1출력

: Segment를 ON시키기 위해서는 PG0~3에 1을 출력





## 4. 7-Segment 구동회로

---

- Sink

: 출력 소자가 **부하(load)**와 **GND**에 있어서 **GND** 쪽을 **ON/OFF** 하는 출력 회로로  
소스 출력 회로보다 큰 전류를 컨트롤함

- ULN2803

: 출력전류 500mA의 **Sink전류**가 가능한 **open collector NPN**형 트랜지스터 array

- Source

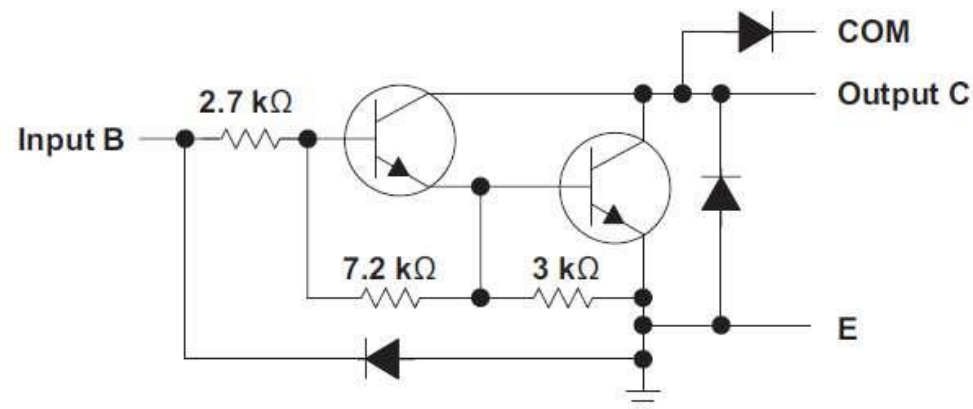
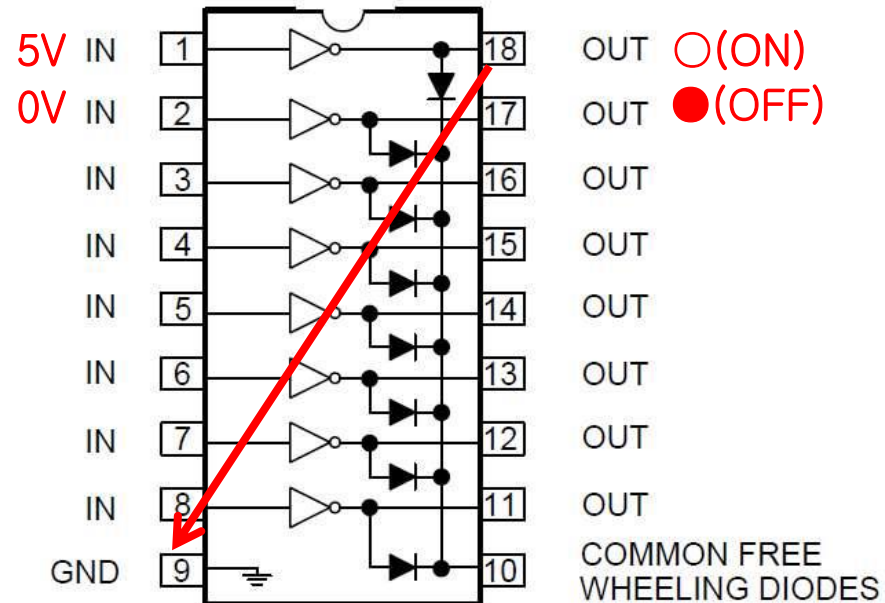
: 출력 소자가 **전원과 부하(load)** 사이에 있어서 **전원** 측을 **ON/OFF** 하는 출력 회로

- UDN2981

: 출력전류 500mA의 **Source전류**가 가능한 **open emitter NPN**형 트랜지스터 array

## 4. 7-Segment 구동회로

- 스위칭용 IC(ULN2803) - Sink전류
- ULN2803은 달링톤(Darlington)트랜지스터가 내장되어 있고, 출력 핀당 500mA의 전류를 구동할 수 있는 오픈콜렉터형 스위칭용 IC
- Darlington TR은 2개의 트랜지스터를 직렬 연결하여 높은 전류를 흐르도록 만든 트랜지스터를 말함
- 입력핀에 5V가 입력되면 해당 제어 출력핀이 9번핀의 GND와 연결되어 구동장치가 동작하게되며, 반대로 입력핀에 0V가 입력되면 해당 제어 출력핀이 GND와 연결되지 않아 구동장치가 동작하지 않음

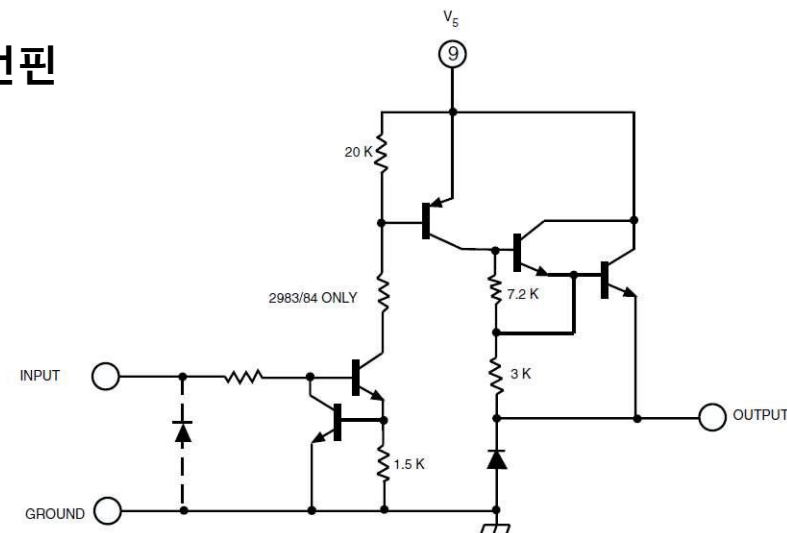
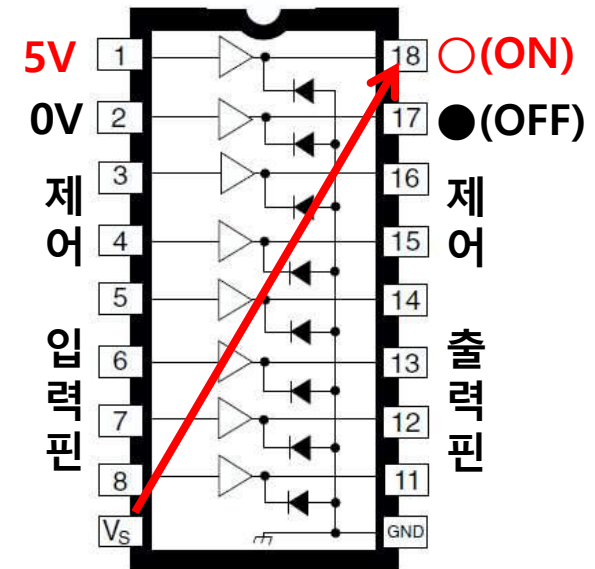


## 4. 7-Segment 구동회로

### 스위칭용 IC(UDN2981) – Source 전류

- 내부에 8개의 Darlington 트랜지스터가 내장되어 있고 출력핀당 500mA의 전류를 구동할 수 있는 오픈 에미터형 스위칭용 IC
- 1~8번 핀은 제어신호를 입력하는 입력핀  
**11~18번 핀은 외부 구동장치의 (+)쪽을 연결하는 출력핀**
  - 1번핀 → 18번핀 제어
  - 2번핀 → 17번핀 제어
- **입력핀에 5V가 입력되면 해당 제어 출력 핀이 9번핀의 제어장치 구동전압과 연결되어 구동장치(여기서는 7Segment)가 동작**  
입력핀에 0V가 입력되면 해당 제어 출력핀은 9번핀과 연결되지 않아 제어장치가 동작하지 않음

UDN2981 내부회로



# 4. 7-Segment 구동회로

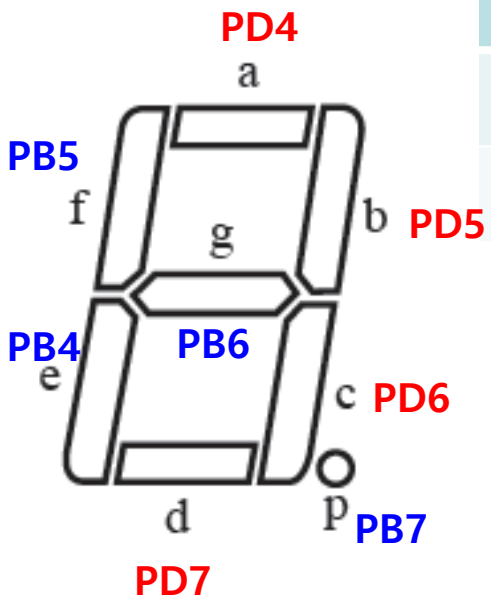
5021

- (예제 3-9) 7-Segment 구동연습 1
  - (1) 각 segment를 1개씩 점멸하기
  - (2) 7-Segment의 8개 LED A, B, C, D, E, F, G, DP를 순차적으로 점멸시키는 프로그램 작성

LED 연결 포트

7-Seg	A	B	C	D	E	F	G	DP
연결 포트	PD4	PD5	PD6	PD7	PB4	PB5	PB6	PB7

1의 값을 출력하면 세그먼트의 LED가 ON (UDN2981)



7-Segment COM 단자 연결포트

SEG COM(+)	DIG1 (맨 왼쪽)	DIG2	DIG3	DIG4
연결 포트	PG0	PG1	PG2	PG3

1을 출력하면 세그먼트의 COM(-) 단자에 GND가 연결(ULN2803)

```

#include <mega128.h>
#include <delay.h>

void main(void)
{
    char i, value;

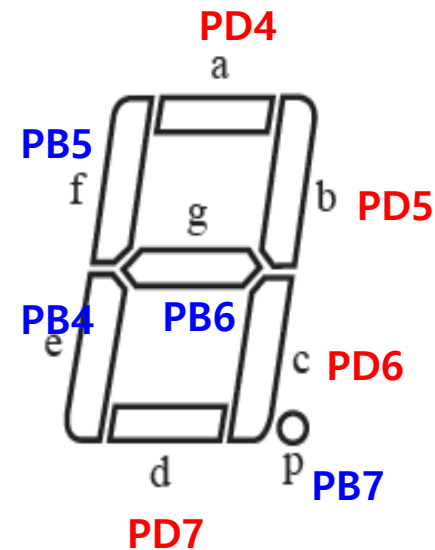
    DDRB = 0xF0;           // PORTB 4,5,6,7 비트 출력 설정
    DDRD = 0xF0;           // PORTD 4,5,6,7 비트 출력 설정

    DDRG = 0x0F;           // PORTG 0,1,2,3 비트 출력 설정

    PORTG = 0b00001000;    // 맨 우측 7-Segment DIG4 ON (PORTG3=1)
    PORTB = 0x00;           // 초기값 PORTB 4,5,6,7 출력 0
    PORTD = 0x00;           // 초기값 PORTD 4,5,6,7 출력 0

    while(1){
        PORTD = 0b00010000;
        delay_ms(500);
        PORTD = 0b00000000;
    }
}

```



```

#include <mega128.h>
#include <delay.h>

void main(void)
{
    char i, value;

    DDRB = 0xF0;           // PORTB 4,5,6,7 비트 출력 설정
    DDRD = 0xF0;           // PORTD 4,5,6,7 비트 출력 설정

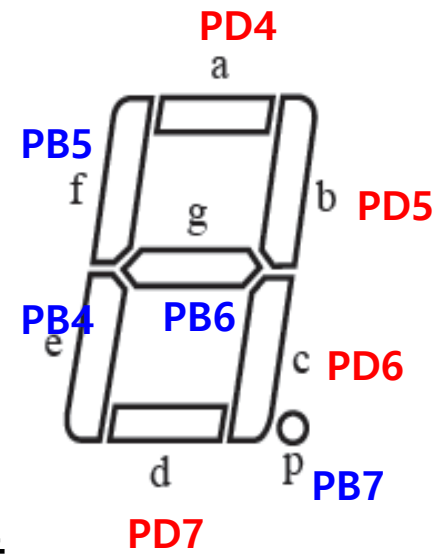
    DDRG = 0x0F;           // PORTG 0,1,2,3 비트 출력 설정

    PORTG = 0b00001000;    // 맨 우측 7-Segment DIG4 ON (PORTG3=1)
    PORTB = 0x00;           // 초기값 PORTB 4,5,6,7 출력 0
    PORTD = 0x00;           // 초기값 PORTD 4,5,6,7 출력 0

    while(1){
        value= 0b00000001;

        for(i = 0; i < 8; i++){
            PORTD = (value & 0x0F) << 4; // PD4-PD7 : A-D
            PORTB = (value & 0xF0);        // PB4-PB7 : E,F,G,DP
            delay_ms(500);
            value = value << 1;           // 출력할 값 1비트 왼쪽 쉬프트
        }
    }
}

```



## 4. 7-Segment 구동회로

---

- (예제 3-10) 7-Segment 구동연습 2
  - 맨 우측(PG3)의 7-Segment부터 좌측(PG0)의 7-Segment로 이동해가며 모든 LED를 0.5초 간격으로 ON시키는 프로그램을 작성하라
  - 모든 프로그램 작성후 아래와 같이 수정해서 실행한 후 어떤 차이가 있는지 확인
    - (1) `delay_ms(500)` → `delay_ms(10)`
    - (2) `delay_ms(500)` → `delay_ms(5)`

• (예제 3-10) 7-Segment 구동연습 2

```
#include <mega128.h>
#include <delay.h>

void main(void)
{
    DDRB = 0xF0;           // PORTB 4,5,6,7 비트 출력 설정
    DDRD = 0xF0;           // PORTD 4,5,6,7 비트 출력 설정

    DD RG = 0x0F;          // PORTG 0,1,2,3 비트 출력 설정

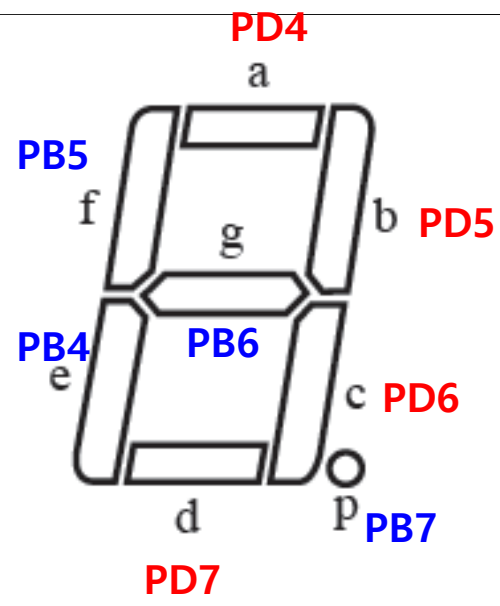
    PORTB = 0b11110000;    // E, F, G, DP on
    PORTD = 0b11110000;    // A, B, C, D on

    while(1){
        PORTG = 0b00001000; // DIG4 7-Segment ON (PG3=1) 우측
        delay_ms(500);

        PORTG = 0b00000100; // DIG3 7-Segment ON (PG2=1)
        delay_ms(500);

        PORTG = 0b00000010; // DIG2 7-Segment ON (PG1=1)
        delay_ms(500);

        PORTG = 0b00000001; // DIG1 7-Segment ON (PG0=1) 좌측
        delay_ms(500);
    }
}
```



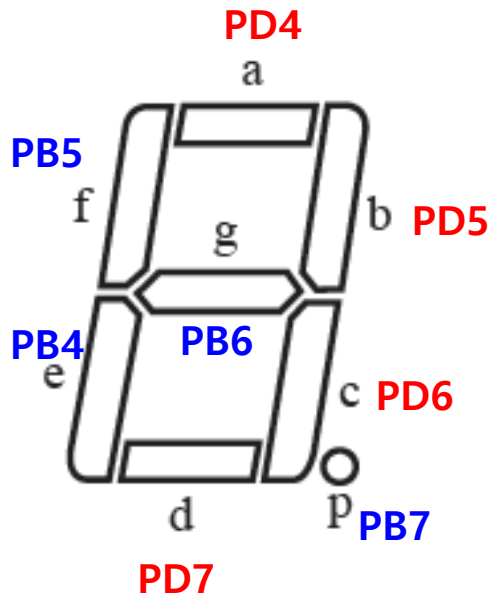


## 4. 7-Segment 구동회로

- (예제 3-11) 7-Segment 구동연습 3

- 맨 우측의 7-Segment에 대해  
16진수값 '0~F'를 순차 표시하는 프로그램을 작성하라

예, 0x3F중에 F 하위 4bit,  
3 상위 3bit



표시 문자	PB6	PB5	PB4	PD7	PD6	PD5	PD4	
	G	F	E	D	C	B	A	
0	0	1	1	1	1	1	1	0x3F
1	0	0	0	0	1	1	0	0x06
2	1	0	1	1	0	1	1	0x5B
3	1	0	0	1	1	1	1	0x4F
4	1	1	0	0	1	1	0	0x66
5	1	1	0	1	1	0	1	0x6D
6	1	1	1	1	1	0	1	0x7D
7	0	0	0	0	1	1	1	0x07
8	1	1	1	1	1	1	1	0x7F
9	1	1	0	1	1	1	1	0x6F
A	1	1	1	0	1	1	1	0x77
b	1	1	1	1	1	0	0	0x7C
C	0	1	1	1	0	0	1	0x39
d	1	0	1	1	1	1	0	0x5E
E	1	1	1	1	0	0	1	0x79
F	1	1	1	0	0	0	1	0x71

## • (예제 3-11) 7-Segment 구동연습 3

```
#include <mega128.h>
```

```
#include <delay.h>
```

```
const unsigned char seg_pat[16]  
    = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07,  
       0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71};
```

```
void main(void) {
```

```
    int i;
```

```
    DDRB = 0xF0;           // 포트 B 상위 4비트 출력 설정
```

```
    DDRD = 0xF0;           // 포트 D 상위 4비트 출력 설정
```

```
    DDRC = 0x0F;           // 포트 C 하위 4비트 출력 설정
```

```
    PORTG = 0b00001000;     // 맨 우측 7-Segment DIG4 ON(PG3=1)
```

```
    PORTB = 0x00;           // E, F, G, DP off (초기값)
```

```
    PORTD = 0x00;           // A, B, C, D off (초기값)
```

```
    while(1){               // 16진수 순차 표시
```

```
        for(i = 0; i < 16; i++){
```

```
            PORTD = ((seg_pat[i] & 0x0F) << 4) | (PORTD & 0x0F); // seg_pat 하위 4bit A, B, C, D
```

```
            PORTB = (seg_pat[i] & 0x70) | (PORTB & 0x0F); // seg_pat 상위 3bit E, F, G
```

```
            delay_ms(1000);
```

```
        }
```

```
    }
```

```
}
```

PORTD 하위 4bits

PORTB 하위 4bits

변경되지 않게

예, 0x3F중에 F 하위 4bit,

3 상위 3bit

## 4. 7-Segment 구동회로

- (예제 3-12) 7-Segment 구동연습 4

: 맨 우측 2개의 7-Segment를 이용하여 0부터 99까지 표시하는 프로그램

```
#include <mega128.h>
```

```
#include <delay.h>
```

```
const char seg_pat[10]= {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f};
```

```
void Seg2_out(int);
```

```
void main(void) {
```

```
    int num = 0;
```

```
    DDRB = 0xF0;           // 포트 B 상위 4비트 출력 설정
```

```
    DDRD = 0xF0;           // 포트 D 상위 4비트 출력 설정
```

```
    DDRG = 0x0F;           // 포트 G 하위 4비트 출력 설정
```

```
    PORTB = 0x0;           // E, F, G, DP off
```

```
    PORTD = 0x0;           // A, B, C, D off
```

```
    while(1){
```

```
        Seg2_out(num);
```

```
        num++;
```

```
        if(num > 99) num = 0;
```

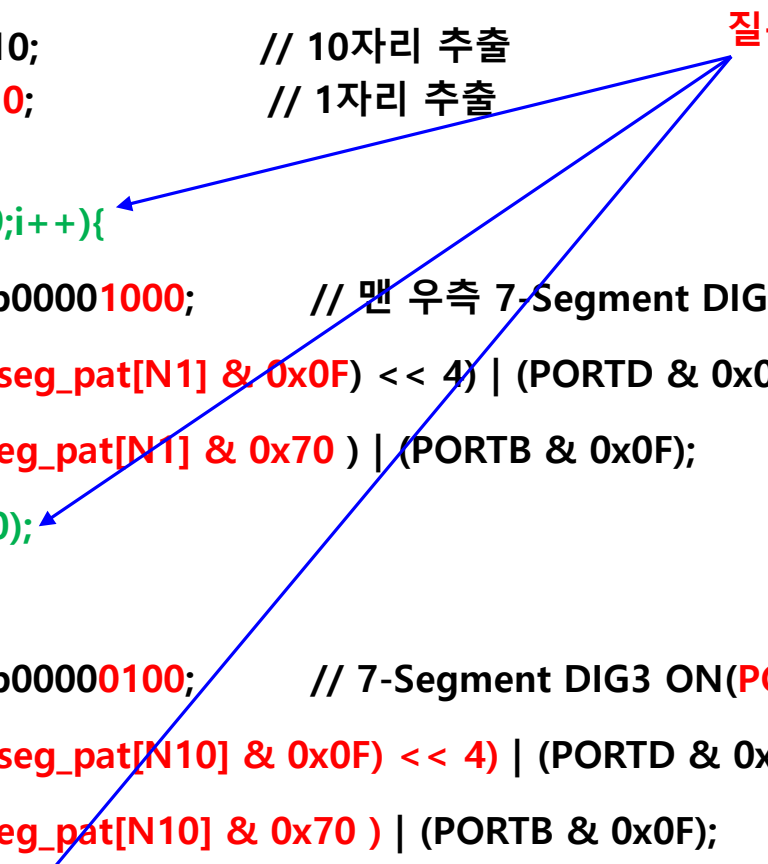
```
    }
```

## 4. 7-Segment 구동회로

```
void Seg2_out(int num) // 두 자리 정수 출력
```

```
{  
    int i, N10, N1;  
  
    N10 = num / 10;           // 10자리 추출  
    N1 = num % 10;           // 1자리 추출  
  
    for(i = 0; i < 49; i++){  
        PORTG = 0b00001000; // 맨 우측 7-Segment DIG4 ON(PG3=1)  
        PORTD = ((seg_pat[N1] & 0x0F) << 4) | (PORTD & 0x0F); // A, B, C, D 표시  
        PORTB = (seg_pat[N1] & 0x70) | (PORTB & 0x0F); // E, F, G 표시(P 사용 안 함)  
        delay_ms(10);  
  
        PORTG = 0b00000100; // 7-Segment DIG3 ON(PG2=1)  
        PORTD = ((seg_pat[N10] & 0x0F) << 4) | (PORTD & 0x0F); // A, B, C, D 표시  
        PORTB = (seg_pat[N10] & 0x70) | (PORTB & 0x0F); // E, F, G 표시  
        delay_ms(10);  
    }  
}
```

질문: 왜 있는 것일까?



## 4. 7-Segment 구동회로

- (예제 3-13) 7-Segment 구동연습 5

: 7-Segment를 이용하여 0부터 9999까지 표시하는 프로그램

```
#include <mega128.h>
```

```
#include <delay.h>
```

```
const char seg_pat[10]= {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f};
```

```
void Seg4_out(int);
```

```
void main(void) {
```

```
    int num = 0;
```

```
    DDRB = 0xF0;           // 포트 B 상위 4비트 출력 설정
```

```
    DDRD = 0xF0;           // 포트 D 상위 4비트 출력 설정
```

```
    DDRG = 0x0F;           // 포트 G 하위 4비트 출력 설정
```

```
    PORTB = 0x0;           // E, F, G, DP off
```

```
    PORTD = 0x0;           // A, B, C, D off
```

```
    while(1){
```

```
        Seg4_out(num);
```

```
        num++;
```

```
        if(num > 9999) num = 0;
```

```
    }
```

```
}
```

```
void Seg4_out(int num) // 네 자리 정수 출력
```

```
{
```

```
    int i, N1000, N100, N10, N1, buf;
```

```
    N1000 = num / 1000;           // 1000자리 추출
```

```
    buf = num % 1000;
```

```
    N100 = buf / 100;            // 100자리 추출
```

```
    buf = buf % 100;
```

```
    N10 = buf / 10;              // 10자리 추출
```

```
    N1 = buf % 10;              // 1자리 추출
```

```
    for(i = 0; i < 2; i++){
```

```
        PORTG = 0b00001000;
```

```
        PORTD = ((seg_pat[N1] & 0x0F) << 4) | (PORTD & 0x0F);
```

```
        PORTB = (seg_pat[N1] & 0x70) | (PORTB & 0x0F);
```

```
        delay_ms(5);
```

```
        PORTG = 0b00000100;
```

```
        PORTD = ((seg_pat[N10] & 0x0F) << 4) | (PORTD & 0x0F);
```

```
        PORTB = (seg_pat[N10] & 0x70) | (PORTB & 0x0F);
```

```
        delay_ms(5);
```

```
        PORTG = 0b00000010;
```

```
        PORTD = ((seg_pat[N100] & 0x0F) << 4) | (PORTD & 0x0F);
```

```
        PORTB = (seg_pat[N100] & 0x70) | (PORTB & 0x0F);
```

```
        delay_ms(5);
```

```
        PORTG = 0b00000001;
```

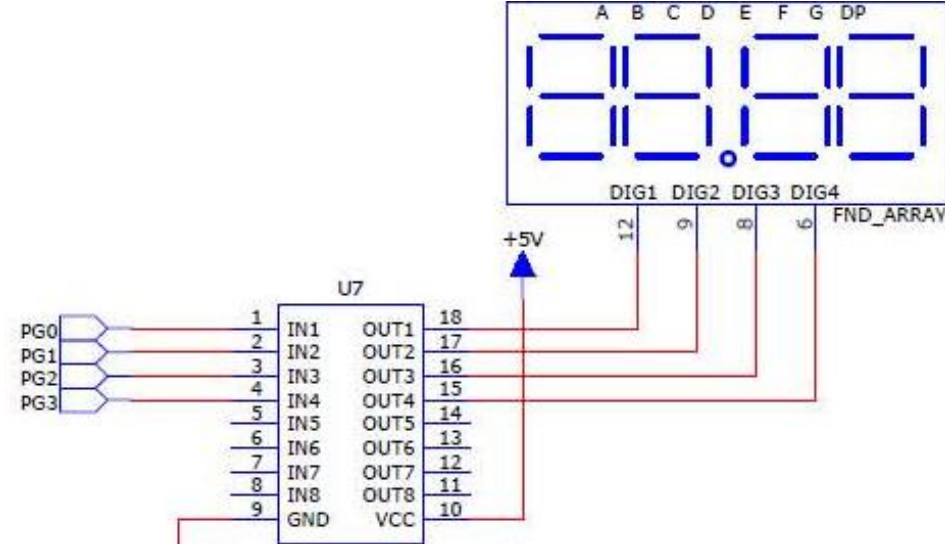
```
        PORTD = ((seg_pat[N1000] & 0x0F) << 4) | (PORTD & 0x0F);
```

```
        PORTB = (seg_pat[N1000] & 0x70) | (PORTB & 0x0F);
```

```
        delay_ms(5);
```

```
    }
```

```
}
```



```
// 맨 우측 7-Segment DIG4 ON(PG3=1)
```

```
// A, B, C, D 표시
```

```
// E, F, G 표시
```

```
// 7-Segment DIG3 ON(PG2=1)
```

```
// A, B, C, D 표시
```

```
// E, F, G 표시
```

```
// 7-Segment DIG2 ON(PG1=1)
```

```
// A, B, C, D 표시
```

```
// E, F, G 표시
```

```
// 7-Segment DIG1 ON(PG0=1)
```

```
// A, B, C, D 표시
```

```
// E, F, G 표시
```

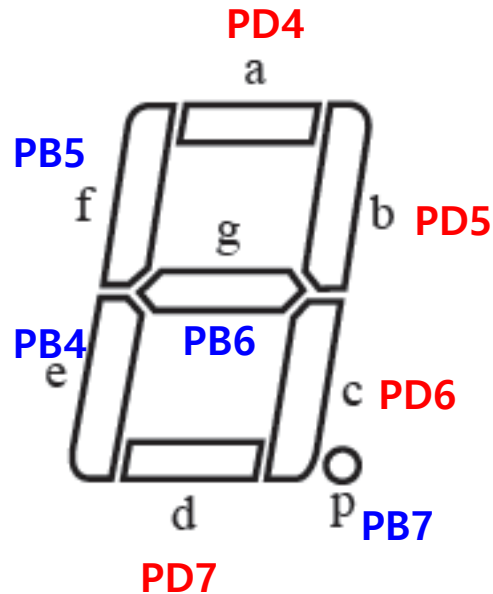
## 4. 7-Segment 구동회로

- (예제 3-14) 입력에 따른 7-Segment 구동연습

: SW1(PE4)이 눌러질(H->L) 때마다

맨 우측(PG3) Segment에 표시되는 값이 0 1 2 3 4... 9 0

표시되도록 하는 프로그램을 작성하라.



```

#include <mega128.h>
#include <delay.h>

const unsigned char seg_pat[10]= {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f};

void main(void)
{
    unsigned char oldkey, key, num = 0;

    DDRB = 0xF0;           // 포트 B 상위 4비트 출력 설정
    DDRD = 0xF0;           // 포트 D 상위 4비트 출력 설정
    DDRC = 0x0F; DDRE = 0x00; // 포트 G 하위 4비트 출력 설정, 포트 E 입력설정

    PORTG = 0b00001000;    // 맨 우측 7-Segment DIG4 ON(PG3=1)
    PORTB = 0x0;           // E, F, G, DP off
    PORTD = 0x0;           // A, B, C, D off
    oldkey = PINE & 0b00010000; // SW1 상태만 추출

    while(1){
        PORTD = ((seg_pat[num] & 0x0F) << 4) | (PORTD & 0x0F); // A, B, C, D 표시
        PORTB = (seg_pat[num] & 0x70) | (PORTB & 0x0F); // E, F, G 표시

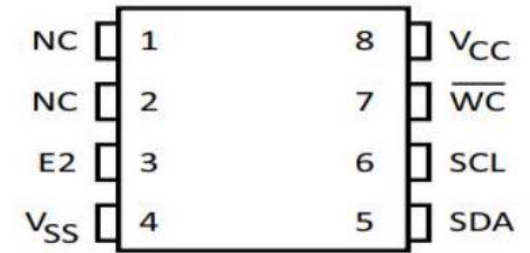
        key = PINE & 0b00010000; // SW1 상태만 추출

        if(oldkey != 0 && key == 0){ // 먼저 상태 OFF, 현 상태 ON ?
            num = (num + 1) % 10; // num값 1증가 (9 다음은 0)
        }
        oldkey = key; // 먼저 상태 <- 현상태
    }
}

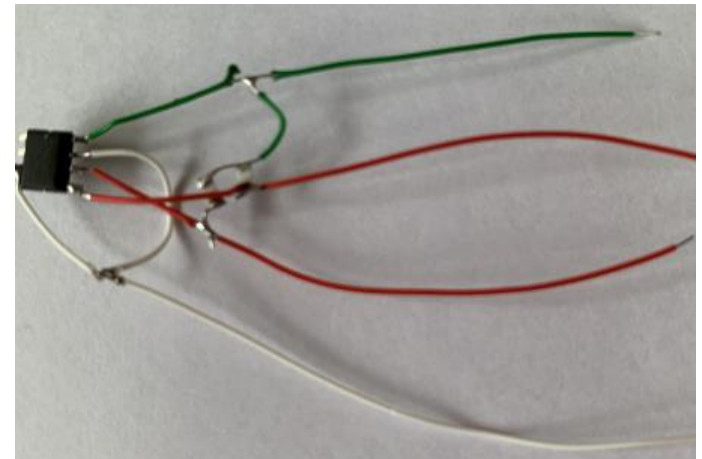
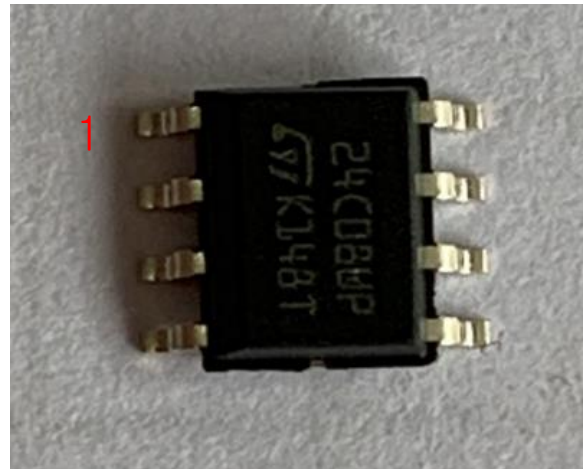
```



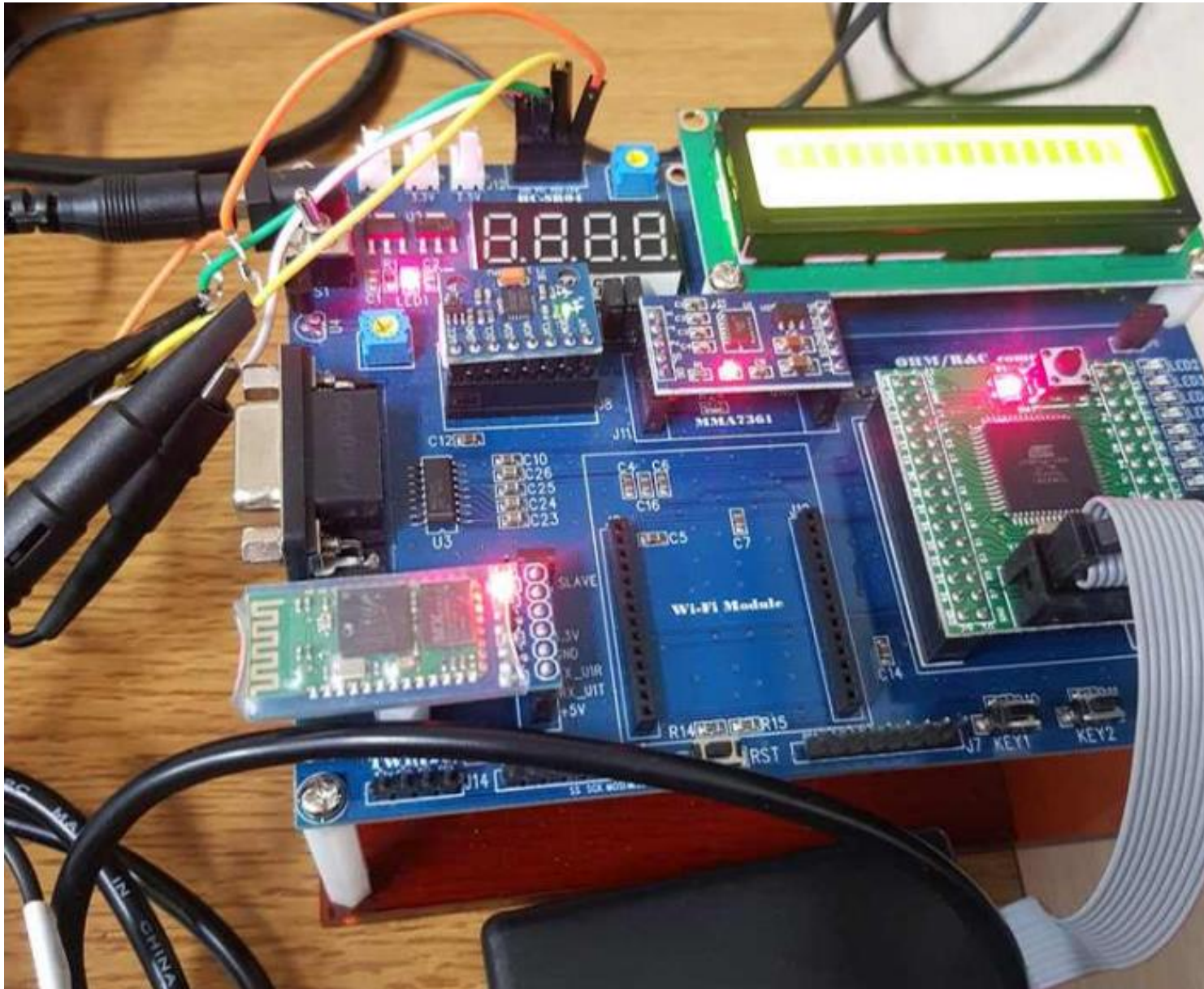
# Sub project-1 : IIC-BUS Programming



VSS,E2,WC - GND VCC - +5V SCL - PD0 SDA - PD1 연결



## Sub project-1 : IIC-BUS Programming

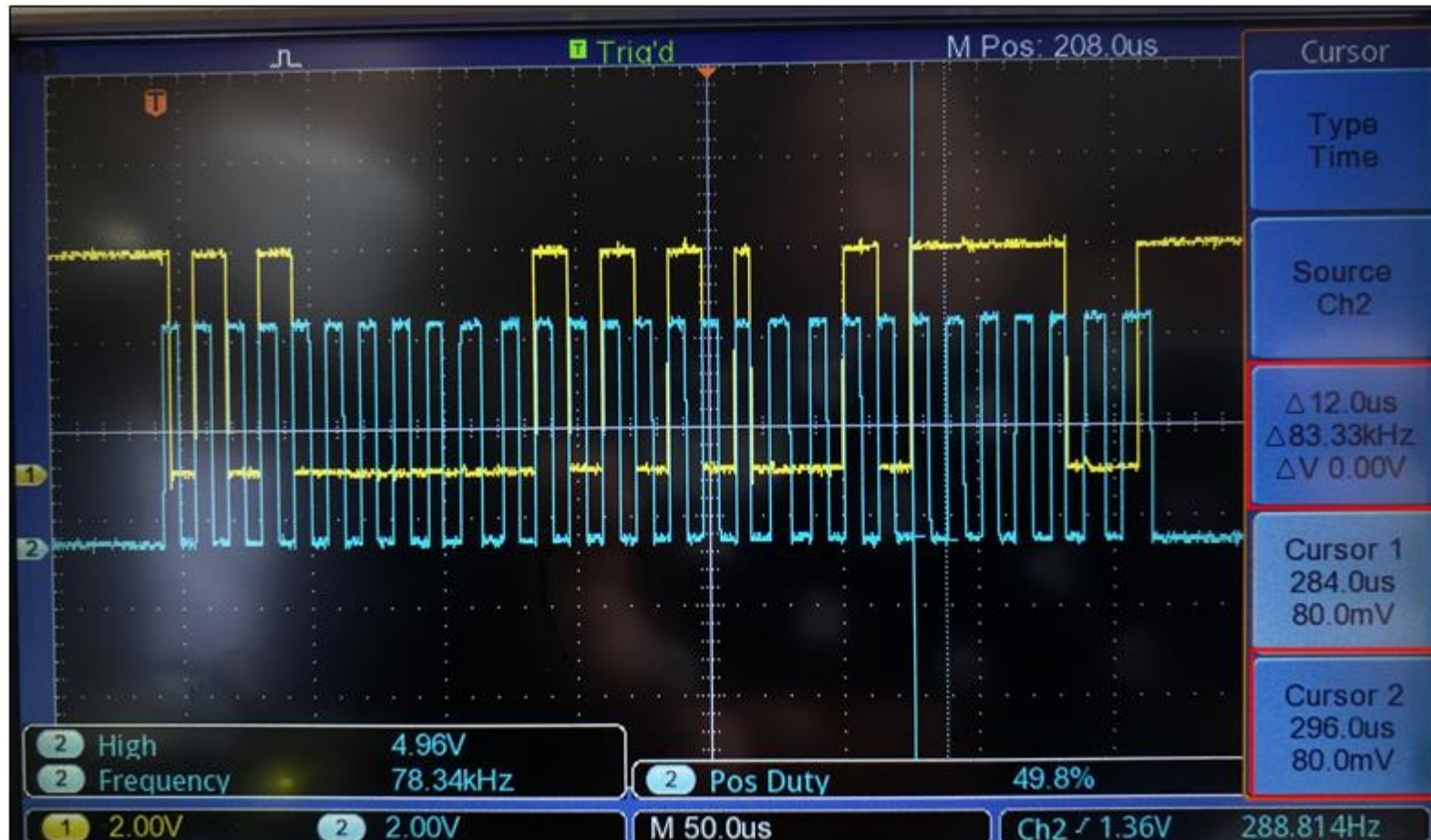




# Sub project-1 : IIC-BUS Programming

Write

기입 : U8 DEV\_ADD\_W = 0xA0;



U8 DEV\_ADD\_R = 0xA1;

U8 IIC\_ADD = 0x55;

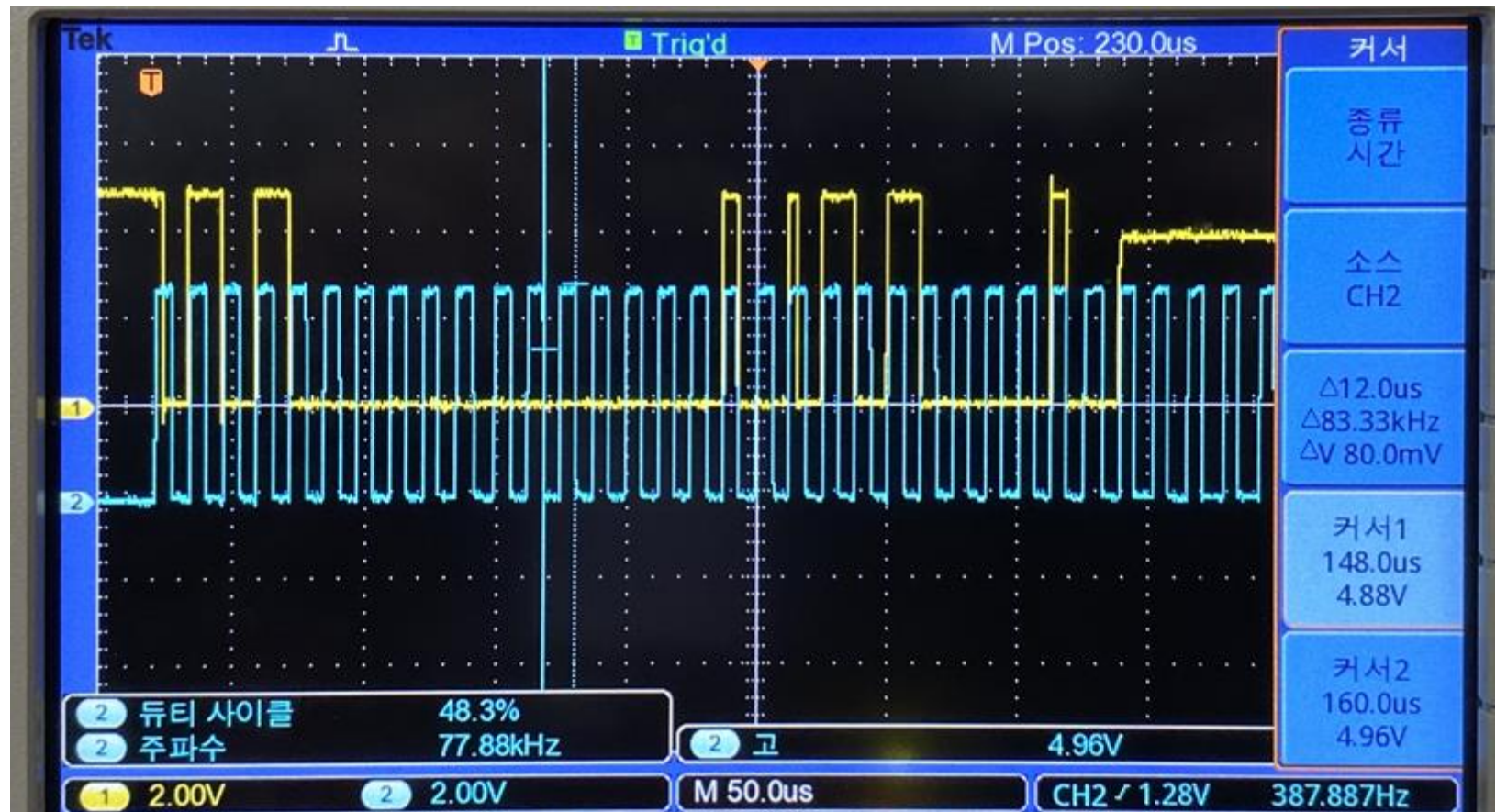
U8 IIC\_DAT = 0x5F;

# Sub project-1 : IIC-BUS Programming

기입 : U8 DEV\_ADD\_W = 0xA0;



# Sub project-1 : IIC-BUS Programming



U8 DEV\_ADD\_R = 0xA1;

U8 IIC\_ADD = 0x01;

U8 IIC\_DAT = 0xFF;

---

감사합니다