AVR - Chapter 3

뉴테크놀로지 컴패니
대표 류 대 우
davidryu@newtc.co.kr

[I/O (GPIO) 제어]

- **I/O 제어**
  - **ATmega128의 I/O 구성 및 특징**
  - I/O PORT 구성
    - 8비트/양방향/범용/병렬 I/O포트 (PORT A ~PORT F) 6개
    - 5비트/양방향/범용/병렬 I/O포트 (PORT G) 1개
  - I/O PORT 특징
    - Read-modify-Write 동작
    - 최대 구동전류 40mA
    - 풀업저항(Pull-up resistor) 설정 가능

[I/O (GPIO) 제어]

- **I/O 관련 레지스터**
  - 각 포트에는 3개의 I/O레지스터(DDRx, PORTx, PINx) 영역을 가짐.

- **DDRA(Data Direction Register)**
  - PORT A의 입/출력 방향설정을 위한 레지스터
  - 레지스터를 SET(1)하면 출력으로 CLEAR(0)하면 입력으로 설정

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	

[I/O (GPIO) 제어]

- **PORTA(Data Register)**
  - 출력용 데이터 값을 위한 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	

- **PINA(Input Pins Address)**
  - 입력 핀에 해당하는 레지스터로서 입력된 값을 표시

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	

[I/O (GPIO) 제어]

- **SFIOR(Special Function I/O Register)**
  - **Bit2. PUD(Pull-up Disable)**
    - 모든 포트의 풀업저항 설정을 위한 레지스터
  - **1= Disable / 0=Enable**

Bit	7	6	5	4	3	2	1	0	
	TSM	–	–	–	ACME	PUD	PSR0	PSR321	SFIOR
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

[PORT 입/출력 제어]

- **PORT 출력 제어**
  - DDRx 레지스터의 각 비트를 1로 셋트 하면 해당포트의 핀이 출력 핀으로 설정
  - PORTx 레지스터의 각 비트를 1로 셋트하면 해당포트의 핀이 1값을 출력
  - PORTx 레지스터의 각 비트를 0로 셋트하면 해당포트의 핀이 0값을 출력한다.

[I/O 부가기능]

- I/O포트의 각 핀들은 대부분 기본적인 범용 I/O 기능 이외에 부수적인 기능을 가지고 있다.
  - PORT A
    - 시분할다중화된 데이터버스
    - 하위 어드레스 버스
  - PORT B
    - 타이머/카운터
    - SPI
  - PORT C
    - 상위 어드레스 버스

[I/O 부가기능]

- PORT D
  - 타이머/카운터
  - 외부인터럽트
  - USART1
  - TWI 직렬통신
- PORT E
  - 타이머카운터
  - 외부인터럽트
  - USART0

[I/O 부가기능]

- PORT F
  - A/D컨버터
  - JTAG
- PORT G
  - 타이머/카운터
  - 외부메모리 인터페이스

[PORT의 기본 설정]

```
void port_init(void)
{
        PORTA = 0x00;
        DDRA  = 0x00; // 입력모드로 설정
        PORTB = 0x00;
        DDRB  = 0xFF;  //출력 모드로 설정
        PORTC = 0x00; //m103 output only
        DDRC  = 0x00;
        PORTD = 0xFF; //PORTD의 기본 값을 1로 출력
        DDRD  = 0x00;
        PORTE = 0x00;
        DDRE  = 0x00;
        PORTF = 0x0F; //상위 (4~7)은 입력으로, 하위 (0~3)까지는 출력으로 설정
        DDRF  = 0x0F;  //상위 (4~7)은 0으로, 하위 (0~3)까지는 1로 설정
        PORTG = 0x00;
        DDRG  = 0x00;
}
```

[외부 포트의 초기화]

```
void ktm128_init(void)
{
    EX_SS_DATA=0x00;
    EX_SS_SEL=0x00;
    EX_DM_SEL=0x00;
    EX_DM_DATA=0x00;
    EX_LED=0x00;
    EX_STEPPING=0x00;
}
```

- 위 초기화는 써줘야 한다.

[사용자 초기화 헤더 파일-1]

```
#ifndef __KTM128__H__
#define __KTM128__H__

#define EX_LCD_DATA         (*(volatile unsigned char *)0x8000)
#define EX_LCD_CONTROL      (*(volatile unsigned char *)0x8001)
#define EX_SS_DATA          (*(volatile unsigned char *)0x8002)
#define EX_SS_SEL           (*(volatile unsigned char *)0x8003)
#define EX_DM_SEL           (*(volatile unsigned int  *)0x8004)
#define EX_DM_DATA          (*(volatile unsigned int  *)0x8006)
#define EX_LED              (*(volatile unsigned char *)0x8008)
#define EX_STEPPING         (*(volatile unsigned char *)0x8009)
#define EX_DCMOTOR          (*(volatile unsigned char *)0x800A)
#define EX_SERVO            (*(volatile unsigned char *)0x800B)
```

[사용자 초기화 헤더 파일-2]

```
void ktm128_init(void)
{
	EX_SS_DATA=0x00;
	EX_SS_SEL=0x00;
	EX_DM_SEL=0x00;
	EX_DM_DATA=0x00;
	EX_LED=0x00;
	EX_STEPPING=0x00;
}

void s_delay(int cnt){
	int i, j;
	for(i=0; i < cnt;i++){
		for(j=0; j < 265; j++)
				;
	}
}
```

[사용자 초기화 헤더 파일-3]

```
void delay(int cnt){
	int i, j;
	for(i=0; i < cnt;i++){
		for(j=0; j < 1000; j++)
			;
	}
}

void l_delay(int cnt){
	int i, j;
	for(i=0; i < cnt;i++){
		for(j=0; j < 2650; j++)
			;
	}
}

#endif __KTM128__H__
```

[사용자 초기화 헤더 파일-4]

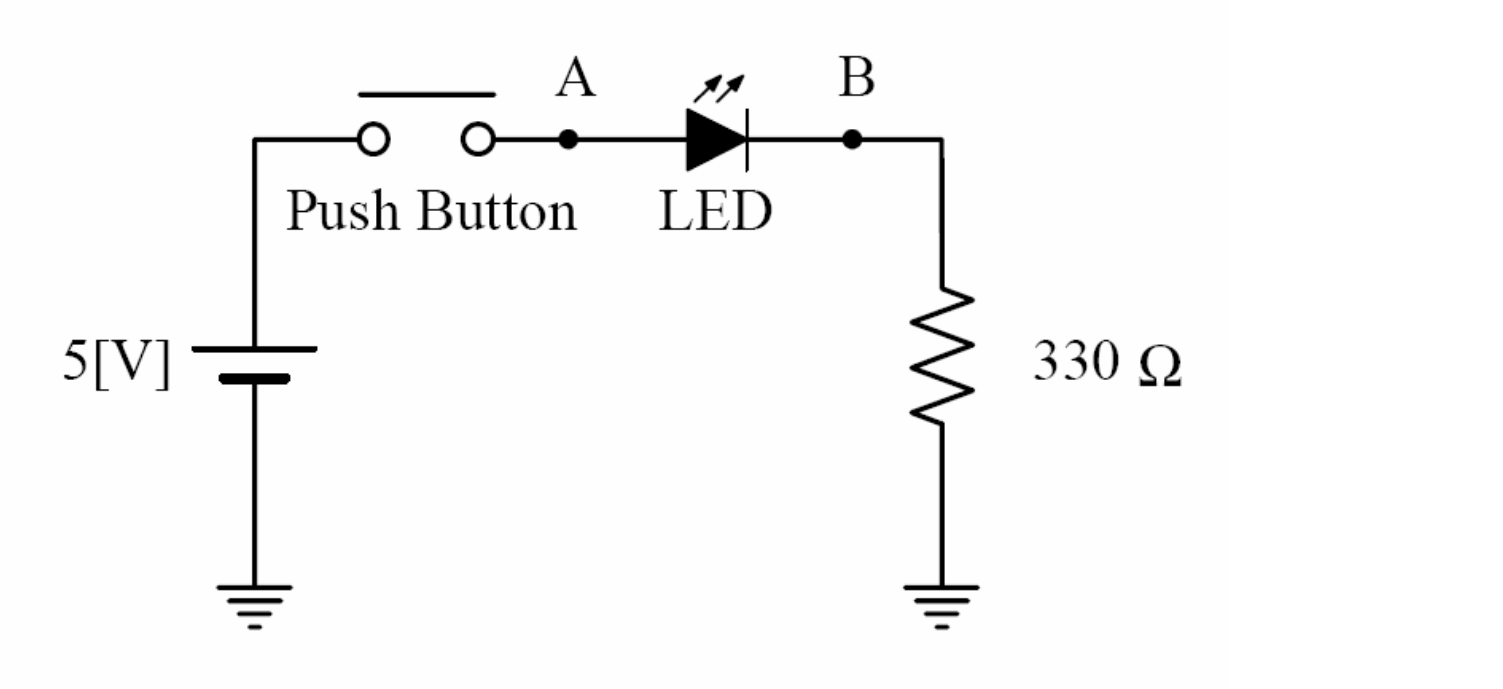
- 파일을 KTM128.h로 저장한다.

- 저장 위치
  - ICC AVR이 설치되어 있는 폴더의 하위 폴더인 include 폴더에 설치(복사)
    - ICCAVR 6.0 기본 설치 시 : C:\icc\include
    - ICCAVR 7.0 기본 설치 시 : C:\iccv7avr\include

- 이후부터 새 프로젝트를 만들면…
  - #include <ktm128.h> 쓰면 된다.

[LED]

- **LED(발광 다이오드)**
  - **전압이 인가되면 회로가 통하고, 역방향 전압이 인가되면 회로를 차단하는 특징을 가짐.**



[LED]

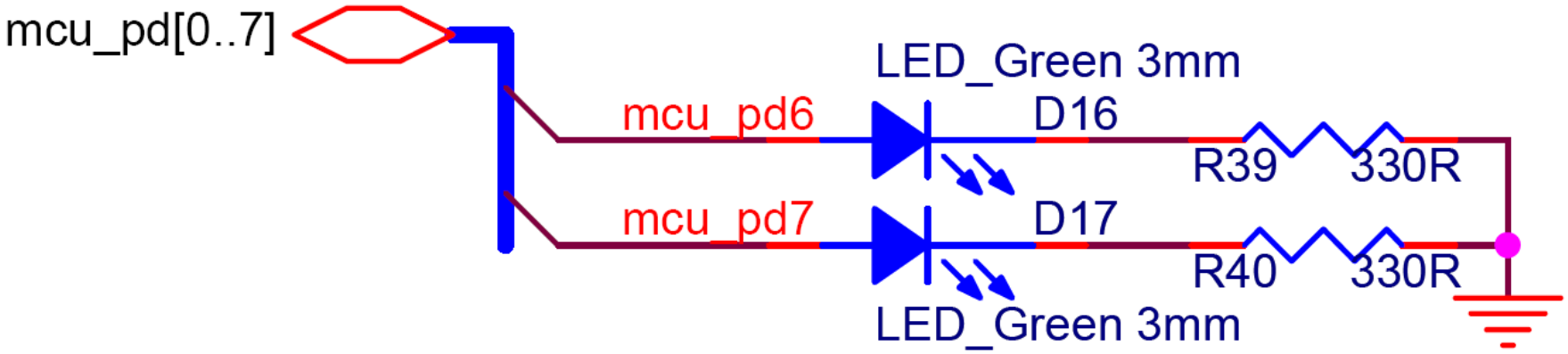
- LED 양단 전압은 다이오드 순방향전압(Vd)인 0.7[V]가 걸리게 된다.

$$I = \frac{V_{dc} - V_d}{R} = \frac{5 - 0.7}{330} = 13.03\,[\mathrm{mA}]$$

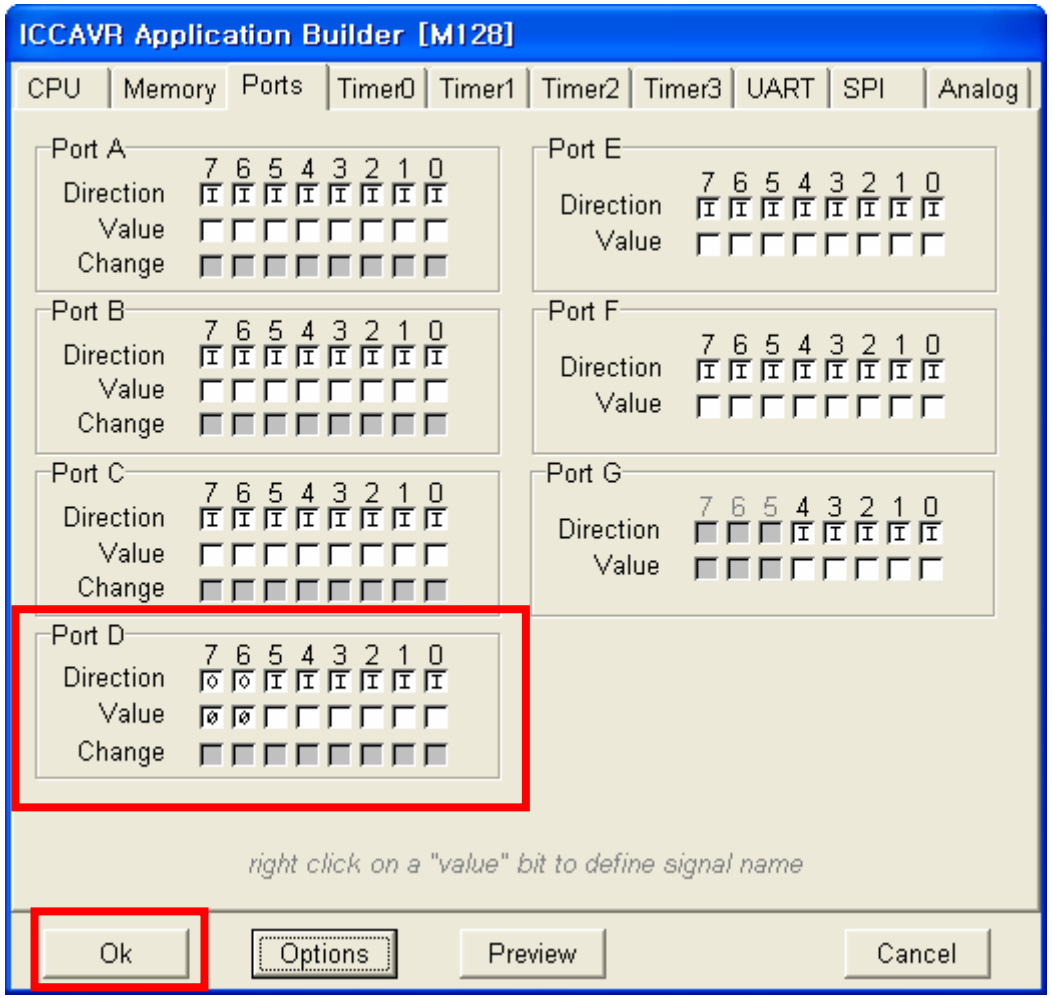
- LED는 13.03[mA]에 비례하는 밝기로 불이 켜진다.

[PORTD의 LED 제어]

- **LED 회로도**
  - **1일때 LED가 점등함.**



[PORTD의 LED 제어]

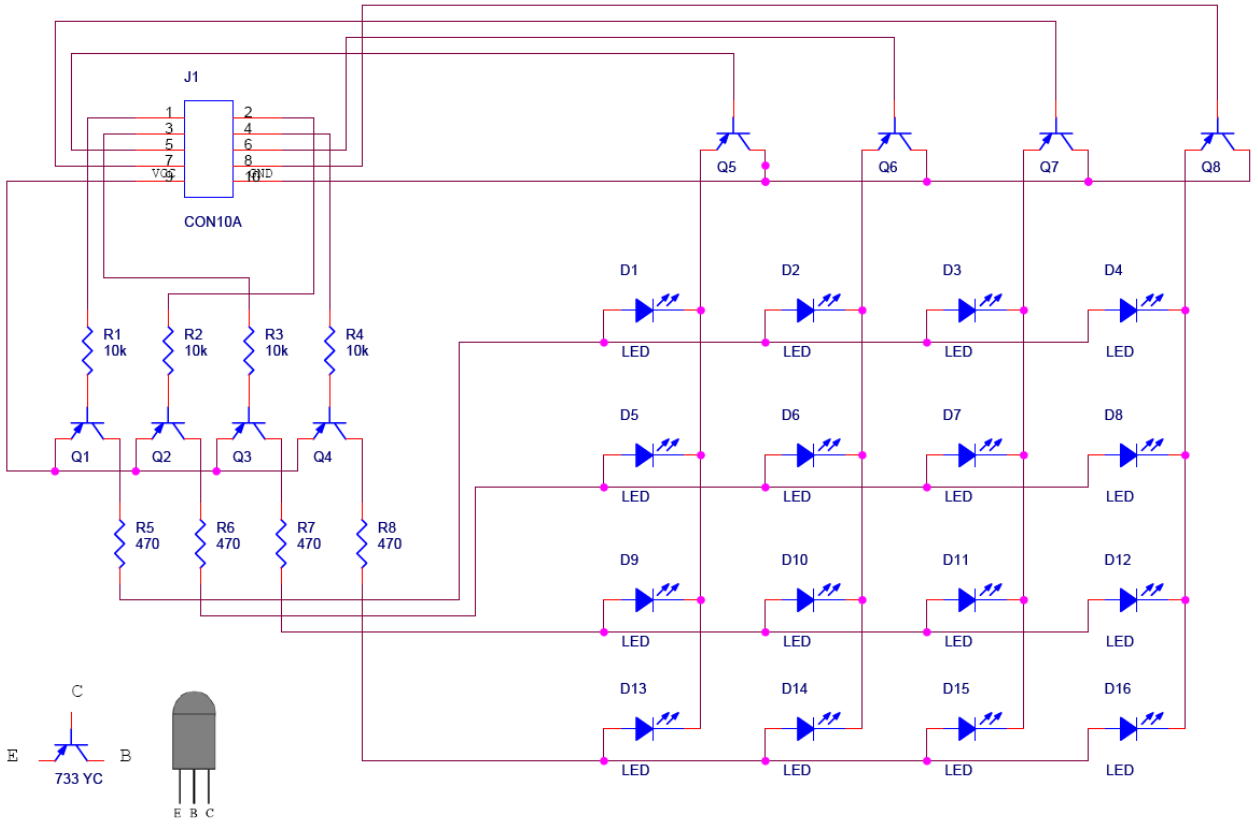


[PORTD의 LED 제어]

```
void main(void)
{
    init_devices();

    while(1){
        PORTD = 0x80;
        delay(100);
        PORTD = 0x40;
        delay(100);
    }
}
```

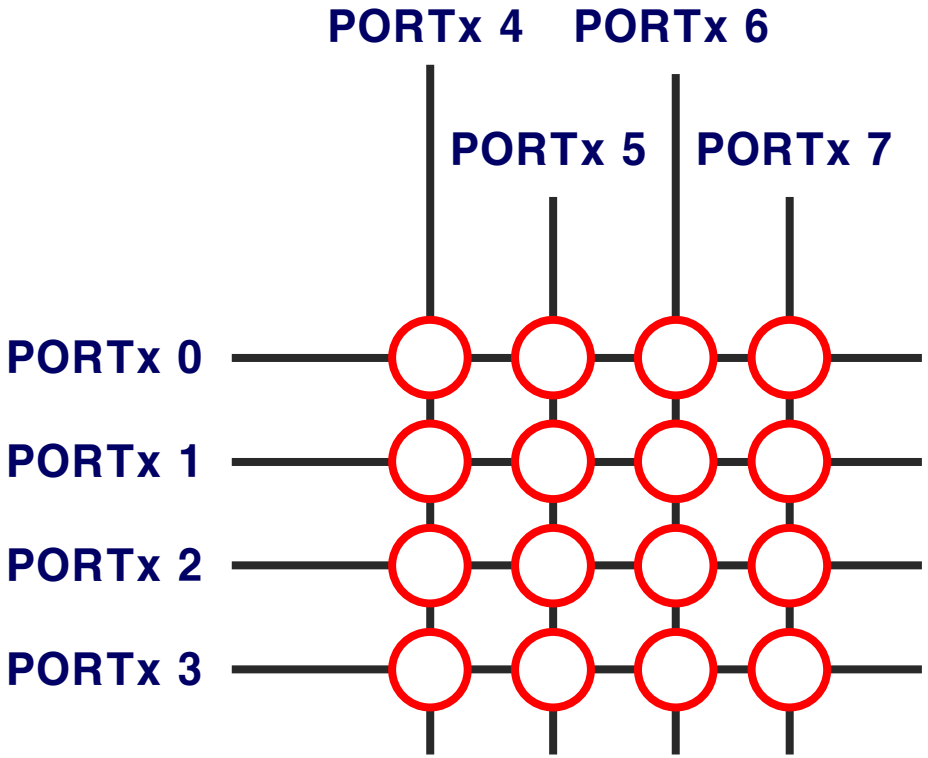
[4 x 4 매트릭스 LED]

- 4 x 4 매트릭스 LED 제작하기.(0일때 점등)
  - A733YC형이 Low Active



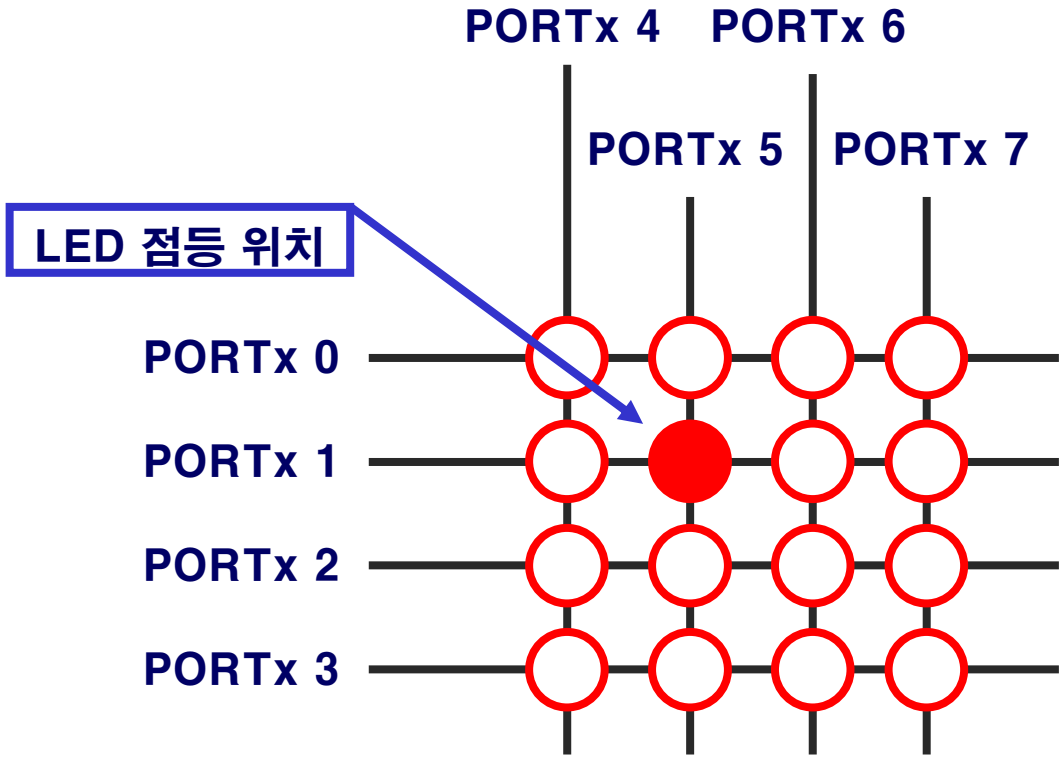
[4 x 4 매트릭스 LED]

- LED 배열



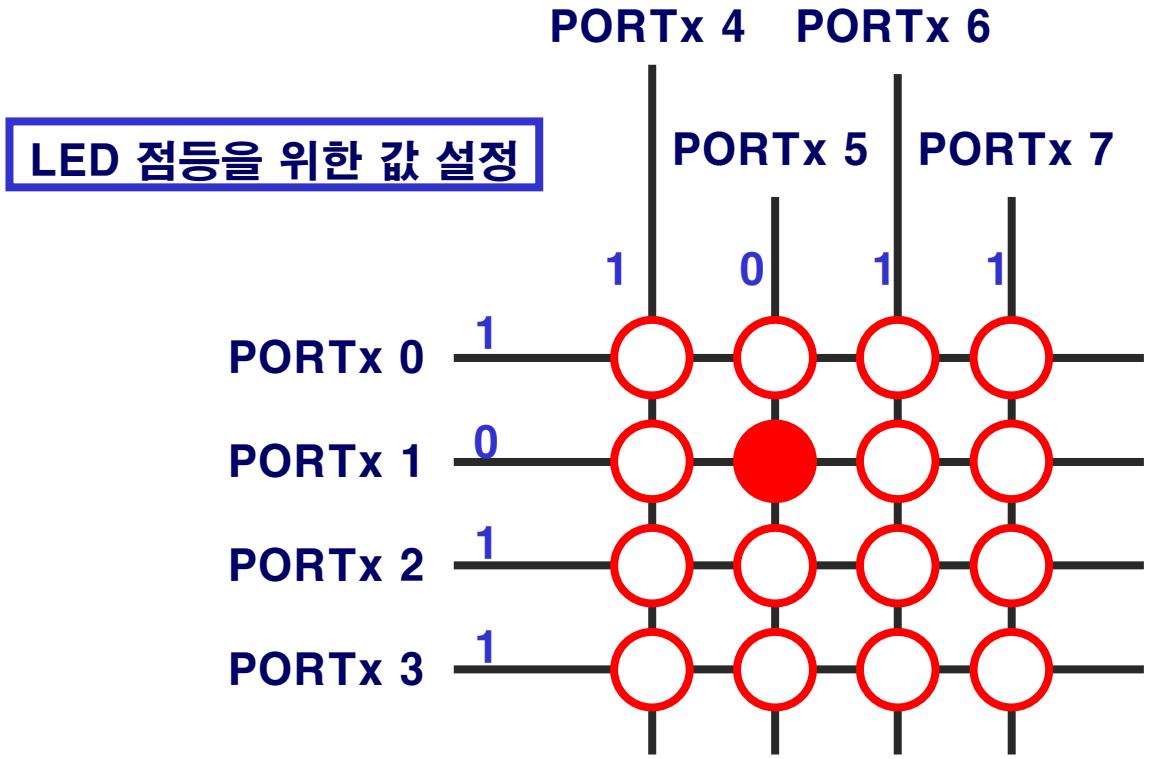
[4 x 4 매트릭스 LED]

- LED 배열

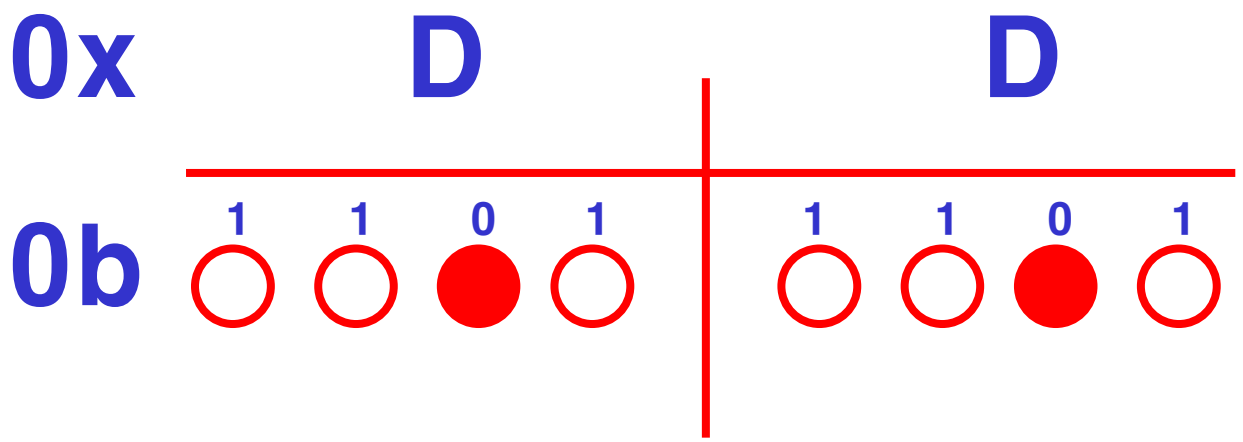


[4 x 4 매트릭스 LED]

- LED 배열

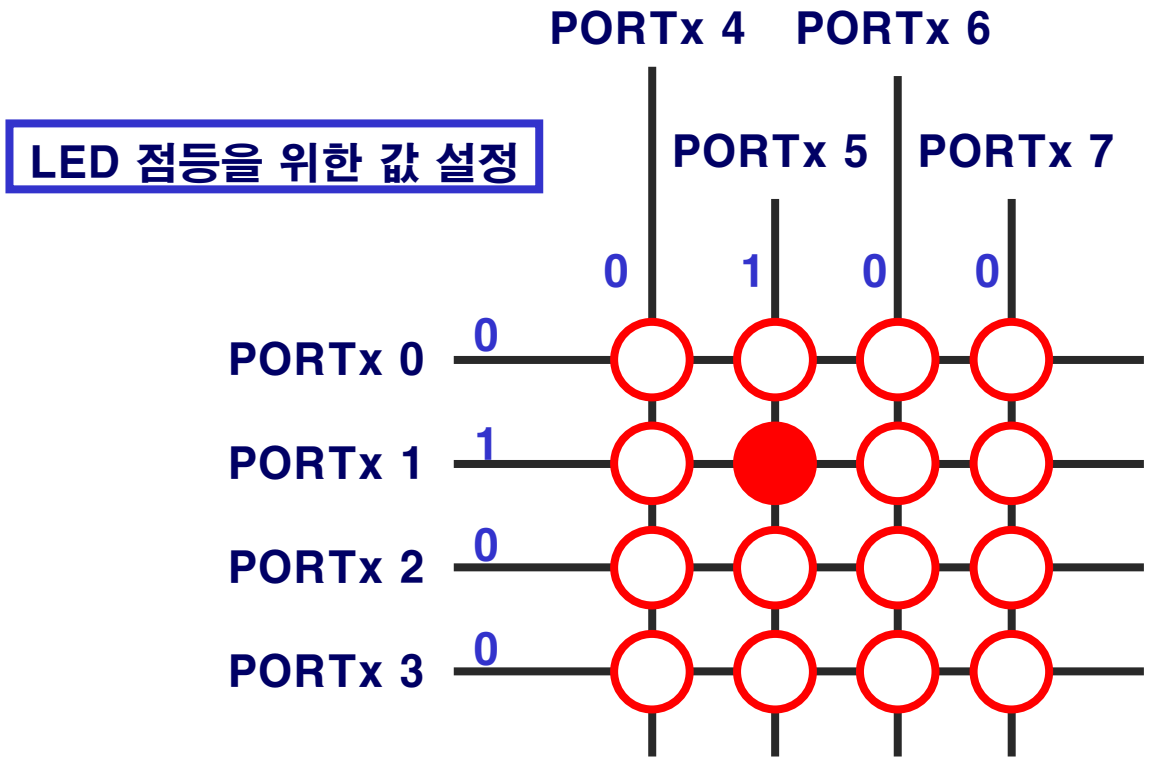


[4 x 4 매트릭스 LED]

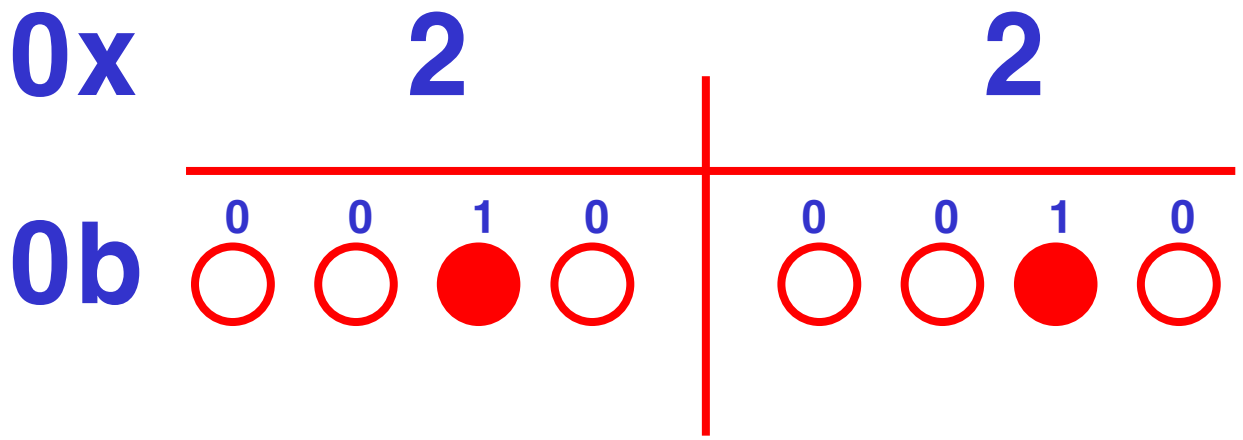


[4 x 4 매트릭스 LED]

- LED 배열(Low Active가 아닐 때)



[4 x 4 매트릭스 LED]

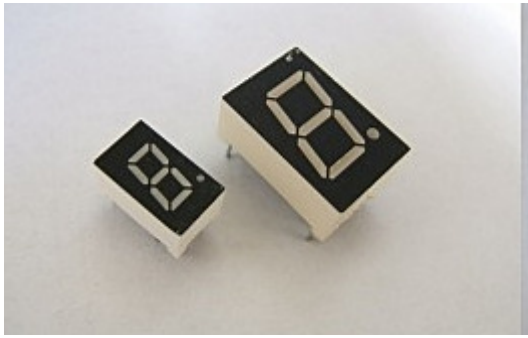


[4 x 4 매트릭스 LED 제어]

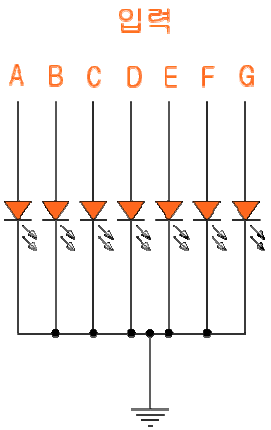
```
void main(void)
{
    init_devices();

    while(1){
        PORTF = 0xDD;
    }
}
```

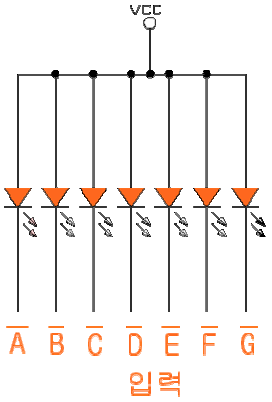
[FND]

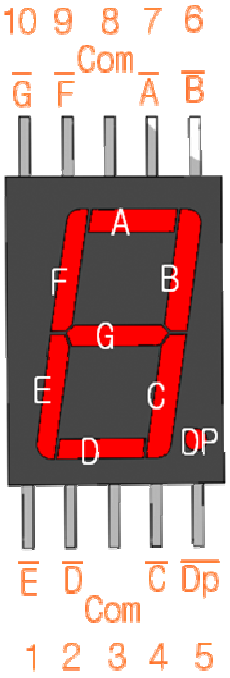
- **정식명칭**
  - 7-Segment LED
  - 7-segment Display
  - Multi- Segmented Display
  - 숫자표시기 줄여서 통상 FND 라고 부른다



[FND 제어]

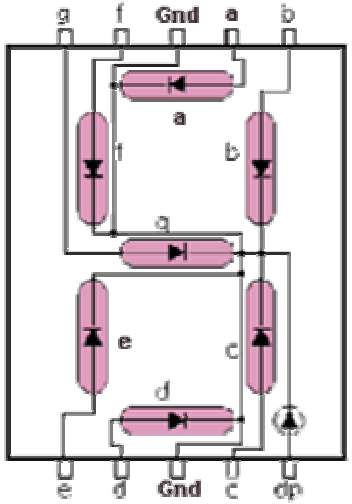






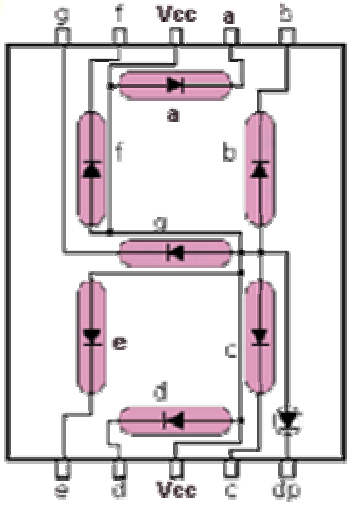
FND500

Common Cathode

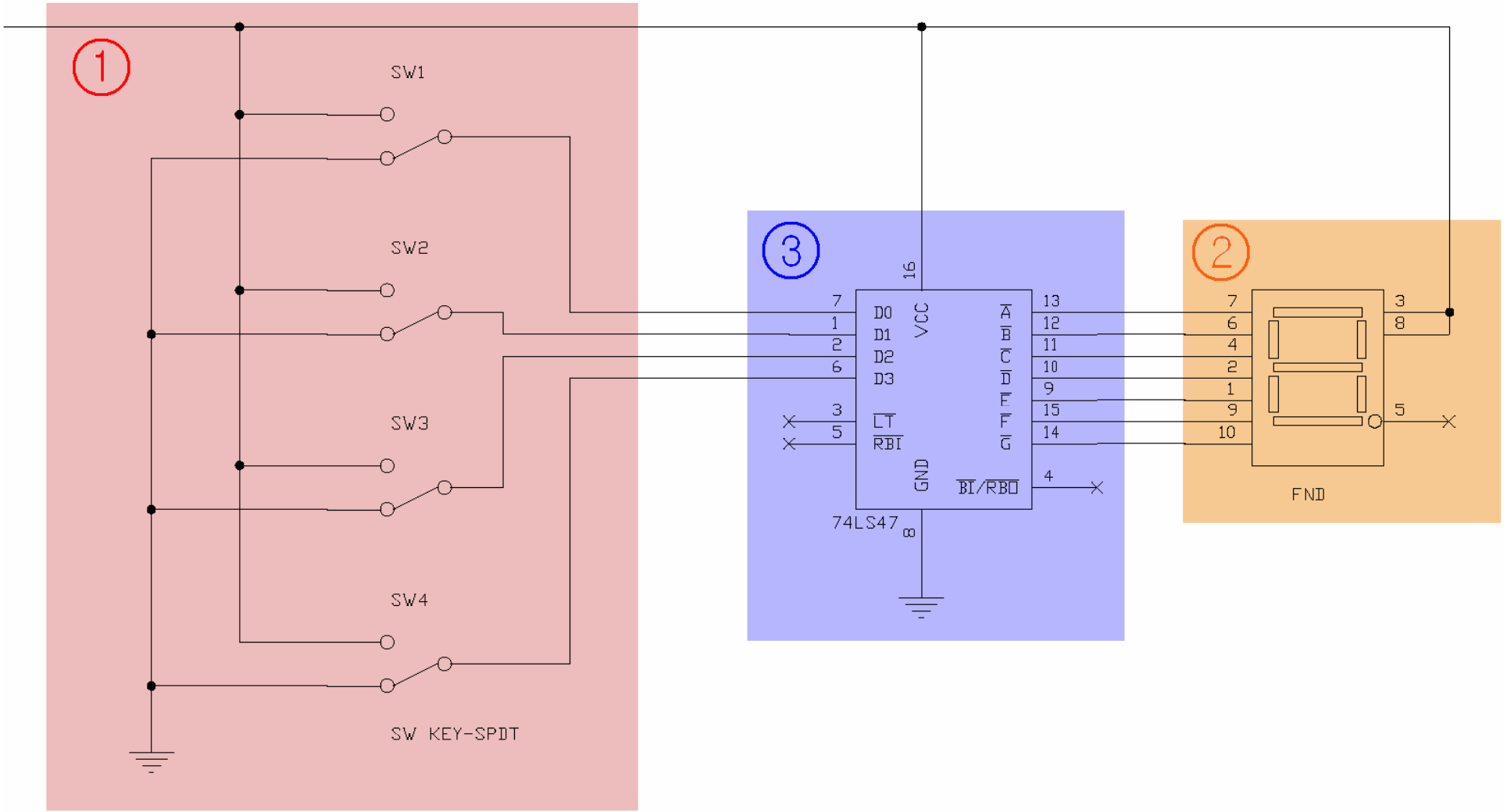


FND507

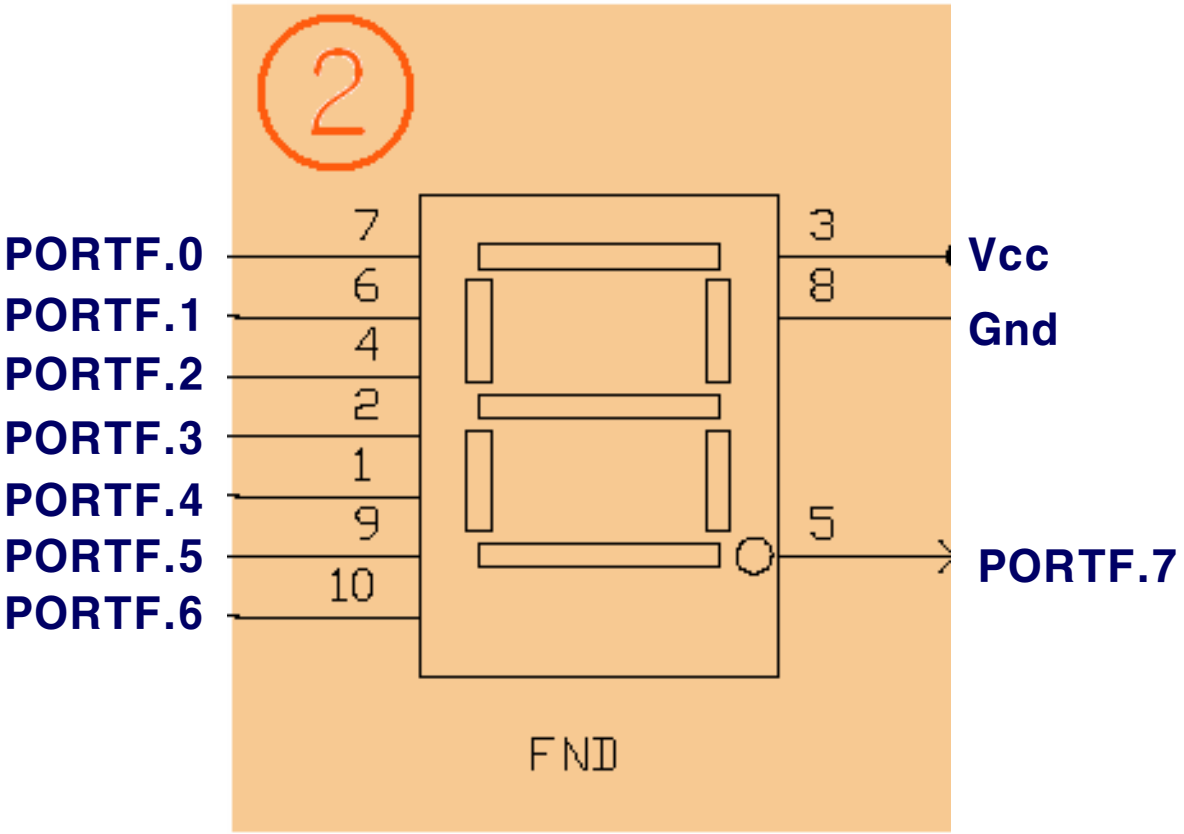
Common Anode



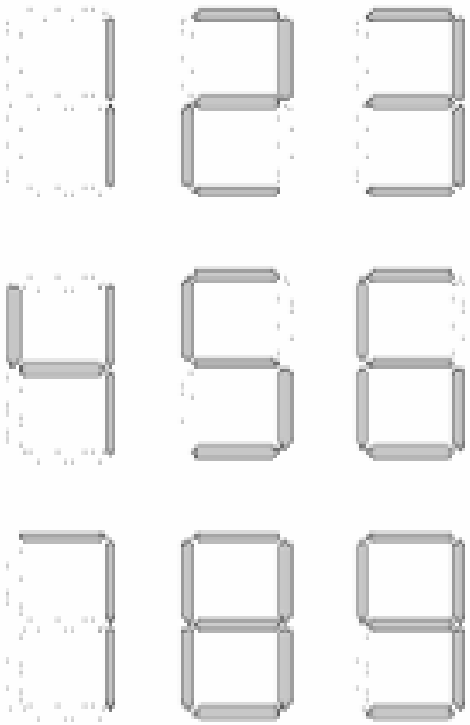
[FND 기본 회로도(1)]


①
SW1
SW2
SW3
SW4
SW KEY-SPDT
③
74LS47
D0
D1
D2
D3
LT
RBI
VCC
GND
BI/RBO
A
B
C
D
E
F
G
②
FND

[FND 기본 회로도(2)]


2
7
6
4
2
1
9
10
3
8
5
PORTF.0
PORTF.1
PORTF.2
PORTF.3
PORTF.4
PORTF.5
PORTF.6
Vcc
Gnd
PORTF.7
FND

[FND 제어]



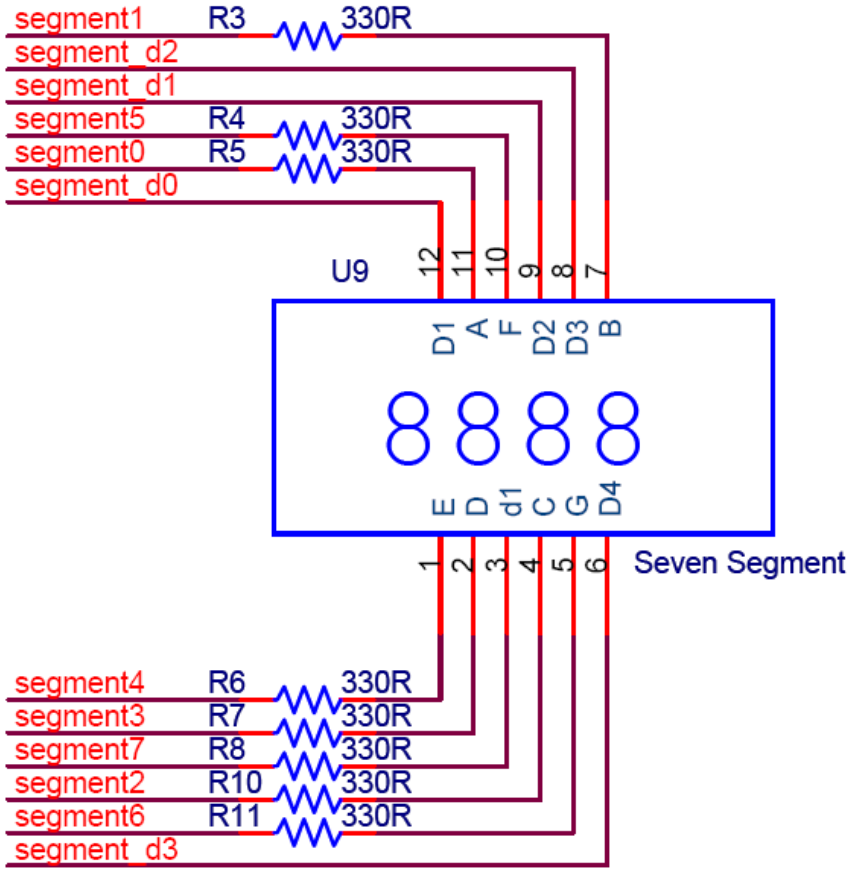
Digit Shown	Illuminated Segment (1 = illumination)						
	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

[FND 제어]

```
const char digit[] =
    {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7c,0x07,0x
    7f,0x67};


void dis(unsigned char num)
{
    PORTF =~digit[num]; //NOT게이트
}
```

[FND 매트릭스 제어]

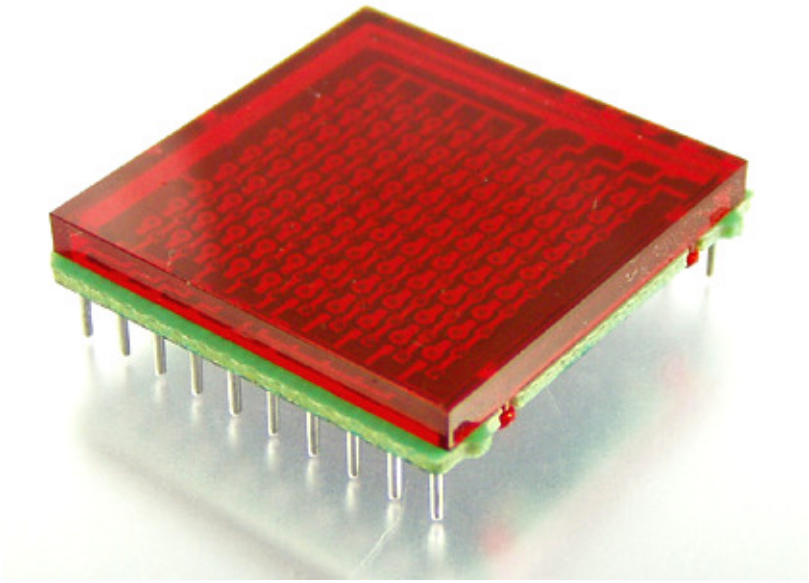
- FND 회로도

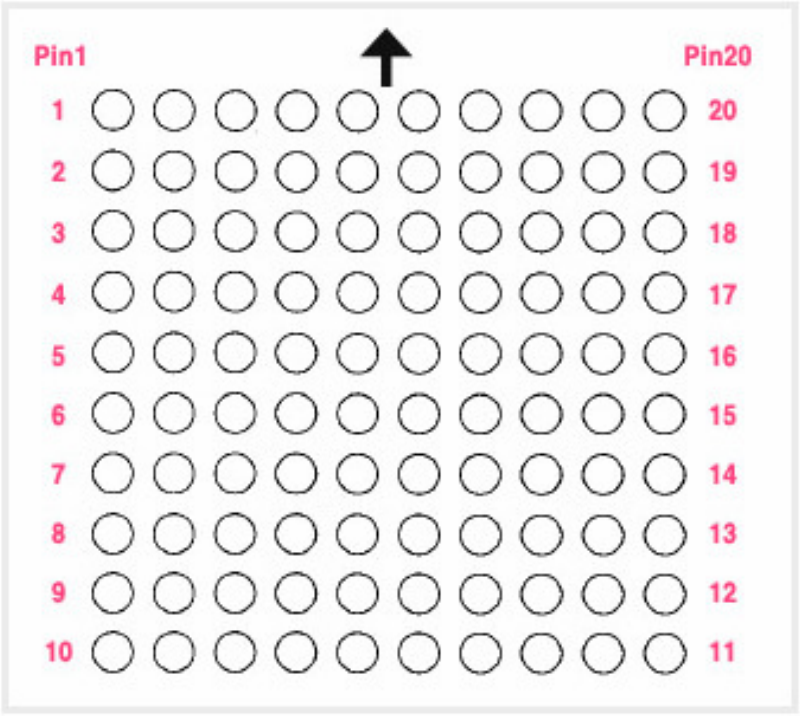


[FND 매트릭스 제어]

```
const char digit[] =
    {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7c,0x07,0x
    7f,0x67};
void runSevenSegment(char i,char num)
{
    EX_SS_SEL = 0x0f;
    EX_SS_DATA = digit[num];
    EX_SS_SEL = ~(0x01 << i);
}
```

[도트 매트릭스]

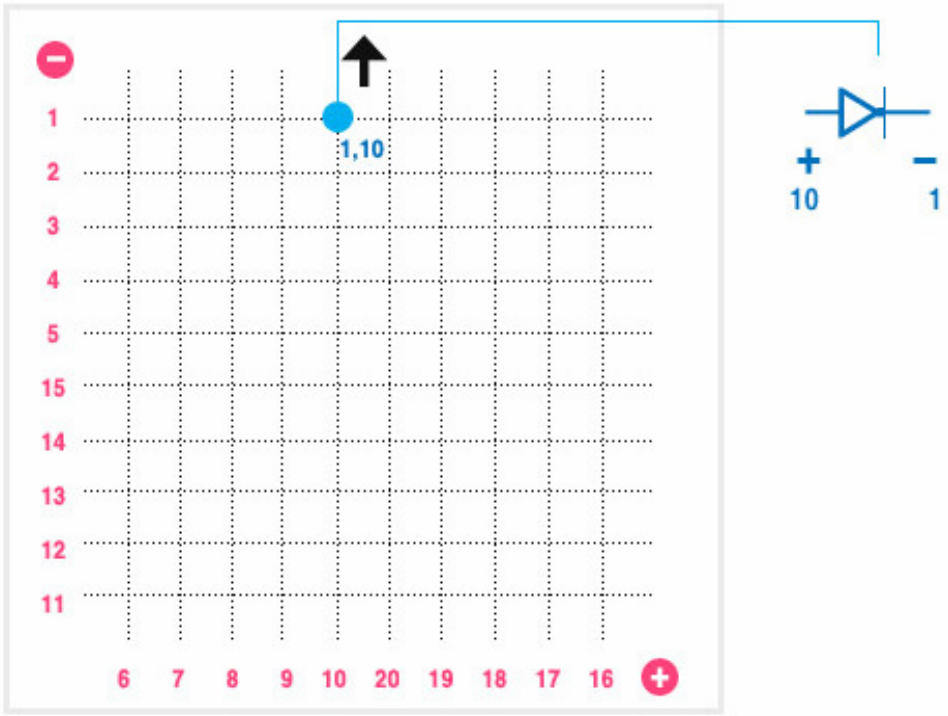




[도트 매트릭스]

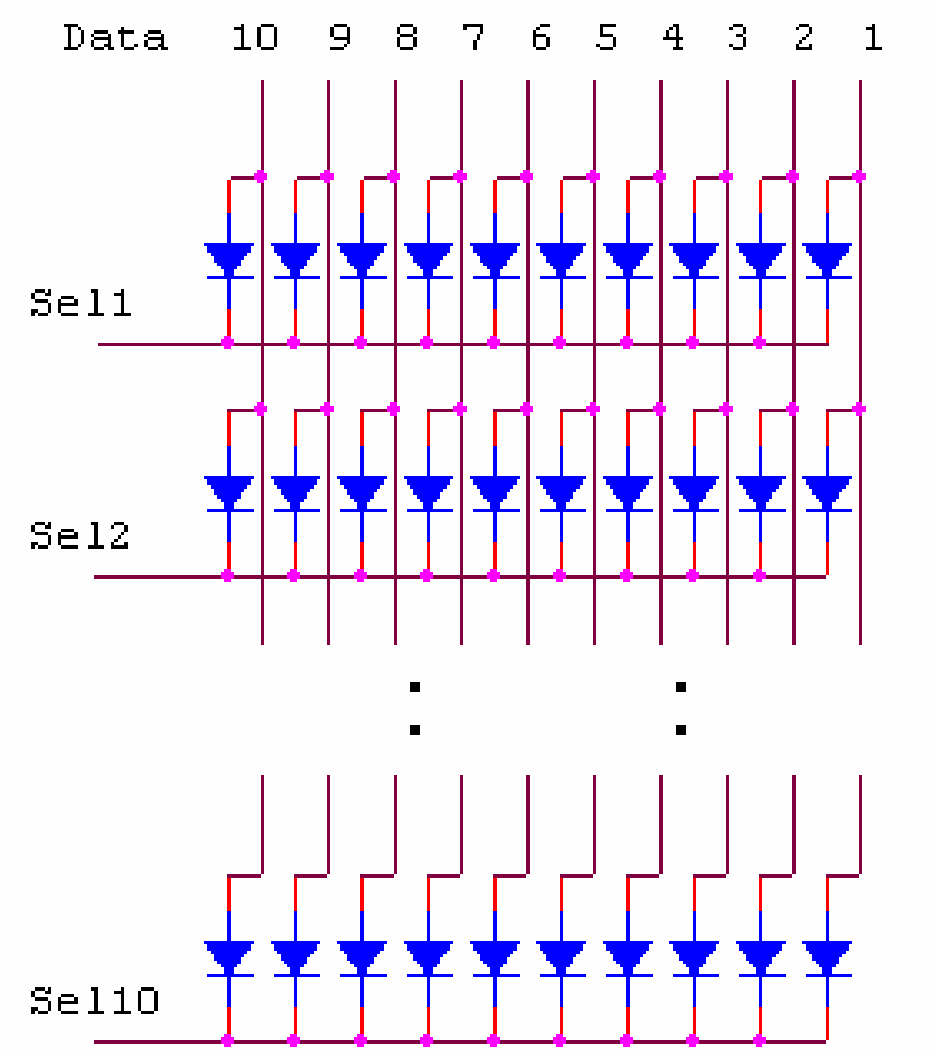
- 크기
  - 10X10, LED : 100 개 사용

- LED 접속 방식



[도트 매트릭스]

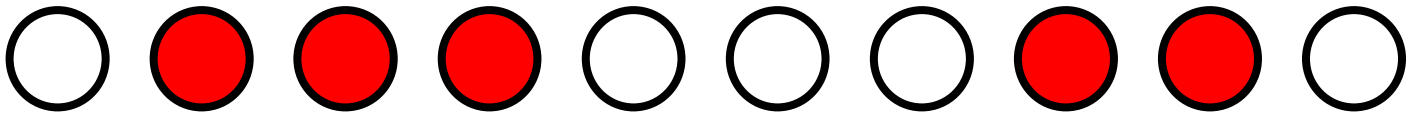
- Data 신호에 첫째 줄의 데이터를 출력하고 그 줄의 Sel1 신호를 0으로 만들어 주면 첫째줄만 켜지게 된다.
- 다음에 두번째줄의 데이터를 출력하고 Sel2 신호를 0으로 만들면 두번째 줄만 켜지게 된다.
- 10개를 순서대로 빠른 속도로 켜면 동시에 켜진 것으로 보이게 된다.



[도트 매트릭스]

- 도트메트릭스 출력할 데이터 만드는 방법

도트메트릭스 한줄 출력



출력 값

0x1C6

[도트 매트릭스]

도트 메트릭스 셀렉트(하위) 출력 메모리 번지 : 0x8004 (Write 전용)

7	6	5	4	3	2	1	0
Sel[7]	Sel[6]	Sel[5]	Sel[4]	Sel[3]	Sel[2]	Sel[1]	Sel[0]

도트 메트릭스 셀렉트(상위) 출력 메모리 번지 : 0x8005 (Write 전용)

7	6	5	4	3	2	1	0
						Sel[9]	Sel[8]

Sel[n]　도트메트릭스 n 번째 줄 선택 (H:Enable, L:Disable)

[도트 매트릭스]

도트 메트릭스 데이터(하위) 출력 메모리 번지 : 0x8006 (Write 전용)

7	6	5	4	3	2	1	0
Data[7]	Data[6]	Data[5]	Data[4]	Data[3]	Data[2]	Data[1]	Data[0]

도트 메트릭스 데이터(상위) 출력 메모리 번지 : 0x8007 (Write 전용)

7	6	5	4	3	2	1	0
						Data[9]	Data[8]

Data[n] 도트메트릭스 한줄의 데이터 (H:On L:Off)

※ 0x8006, 0x8004 의 Integer Pointer 를 사용하여 하위 10bit 를 사용하여 프로그래밍 할 수 있습니다.

[도트 매트릭스]

도트 메트릭스 출력 포트 지정

```
#define DM_SEL          (*(volatile unsigned int  *)0x8004)
#define DM_DATA         (*(volatile unsigned int  *)0x8006)
```

도트메트릭스의 Data가 매핑되어 있는 0x8006 번지를 unsigned int 형 포인터의 포인터 변수로 지정하고 Sel신호가 매핑되어 있는 0x8004 번지를 unsigned int 형 포인터의 포인터 변수로 지정한 것이다.

앞에 volatile 은 해당 메모리가 레지스터의 주소이기 때문에 해당 메모리 번지에 대하여 최적화를 하지 않도록 컴파일러에게 알리는 것이다.

[도트 매트릭스]

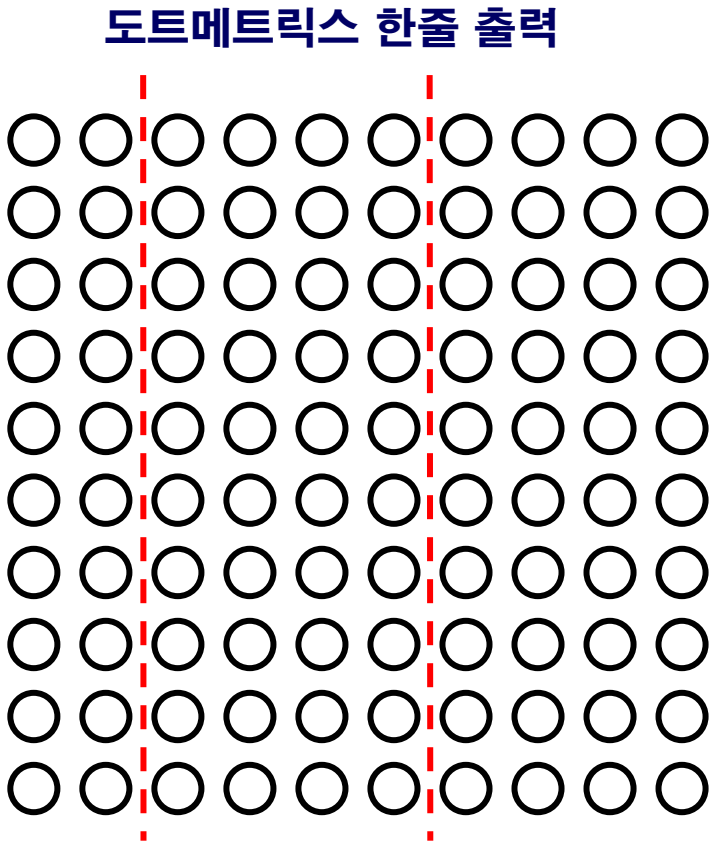
```
// 도트메트릭스 출력 데이터
int dm_data[10] = {0x000, 0x0cc, 0x132, 0x201, 0x205, 0x10a,
    0x084, 0x048, 0x030, 0x000};
int dmi=0;                                  // 현재 출력할 줄 선택
//실행 할 때 마다 한줄씩 켜짐
void runDotMatrix(void)
{
    EX_DM_SEL = 0;
    EX_DM_DATA = dm_data[dmi];     // dmi 번째 줄 데이터 출력
    EX_DM_SEL = 1<<dmi;            // dmi 번째 줄 선택
    dmi++;
    if(dmi>9)   dmi=0;
}
```

[도트 매트릭스]

```
void main(void)
{
    while(1){
        runDotMatrix();
        delay(10);
    }
}
```

[숙제1]

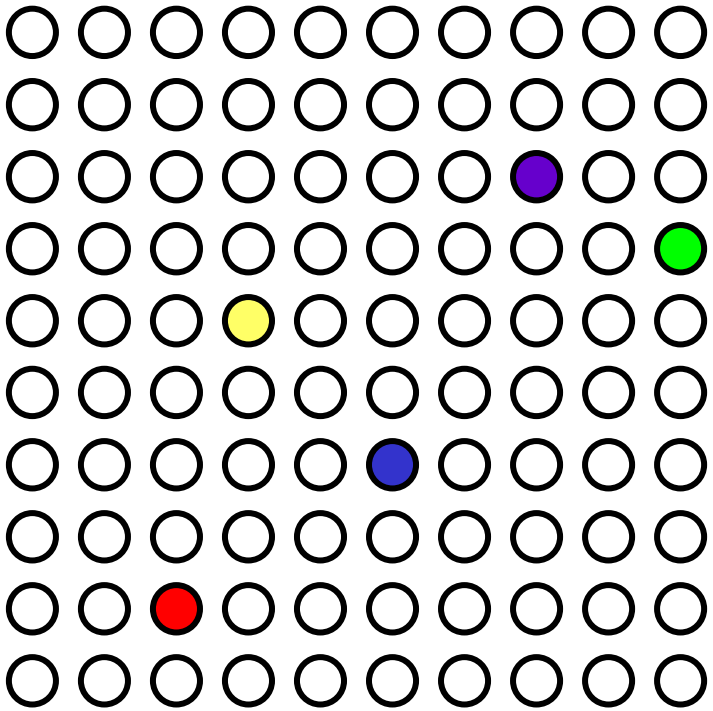
- 다음 데이터 값을 채우시오.



Data	Sel
( )	0x001
( )	0x002
( )	0x004
( )	0x008
( )	0x010
( )	0x020
( )	0x040
( )	0x080
( )	0x100
( )	0x200

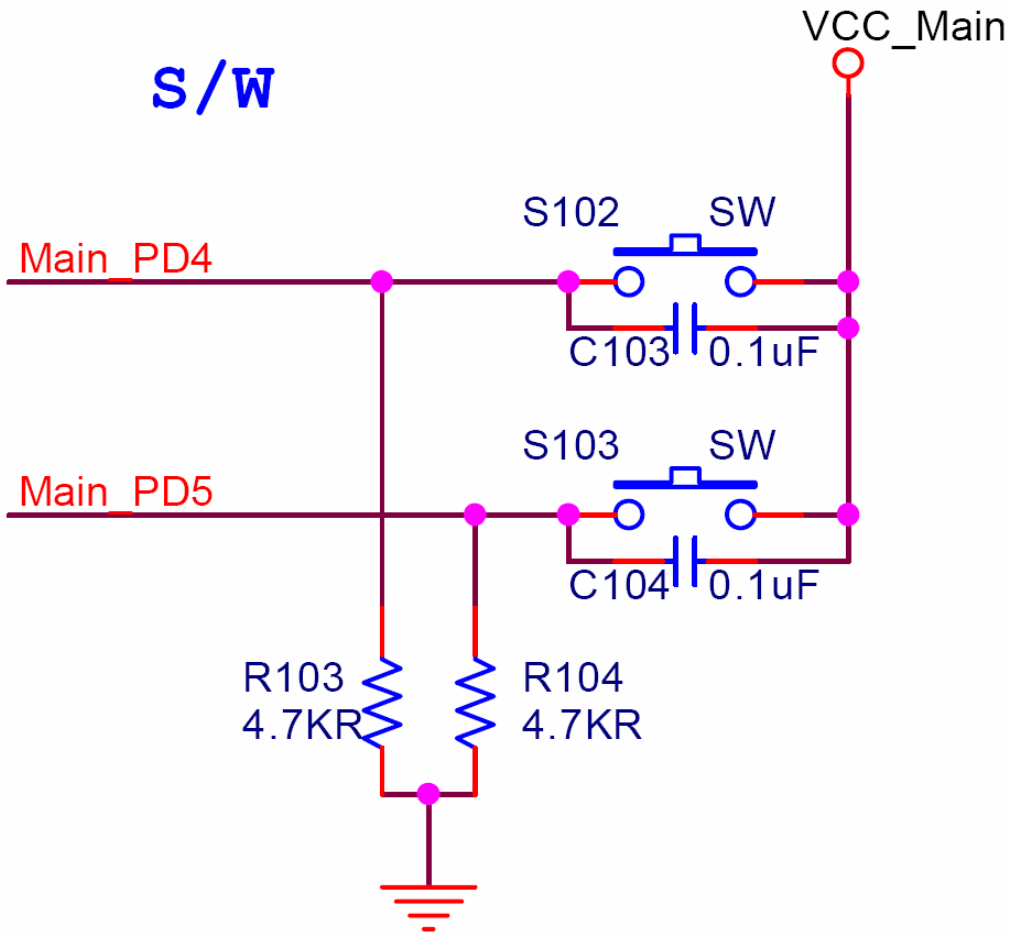
[숙제2]

- 다음 LED가 순서대로 점등되도록 만드시오.
  - 순서는 빨강 -> 노랑 -> 파랑 -> 보라 -> 녹색
  - 빨강이 점등 된 후에 노랑이 점등 될 때 빨강은 계속 켜져 있도록 한다.
  - 최종 녹색이 들어오면 빨강부터 다시 한다.
  - 오른쪽은 녹색까지 전부 점등된 화면입니다.



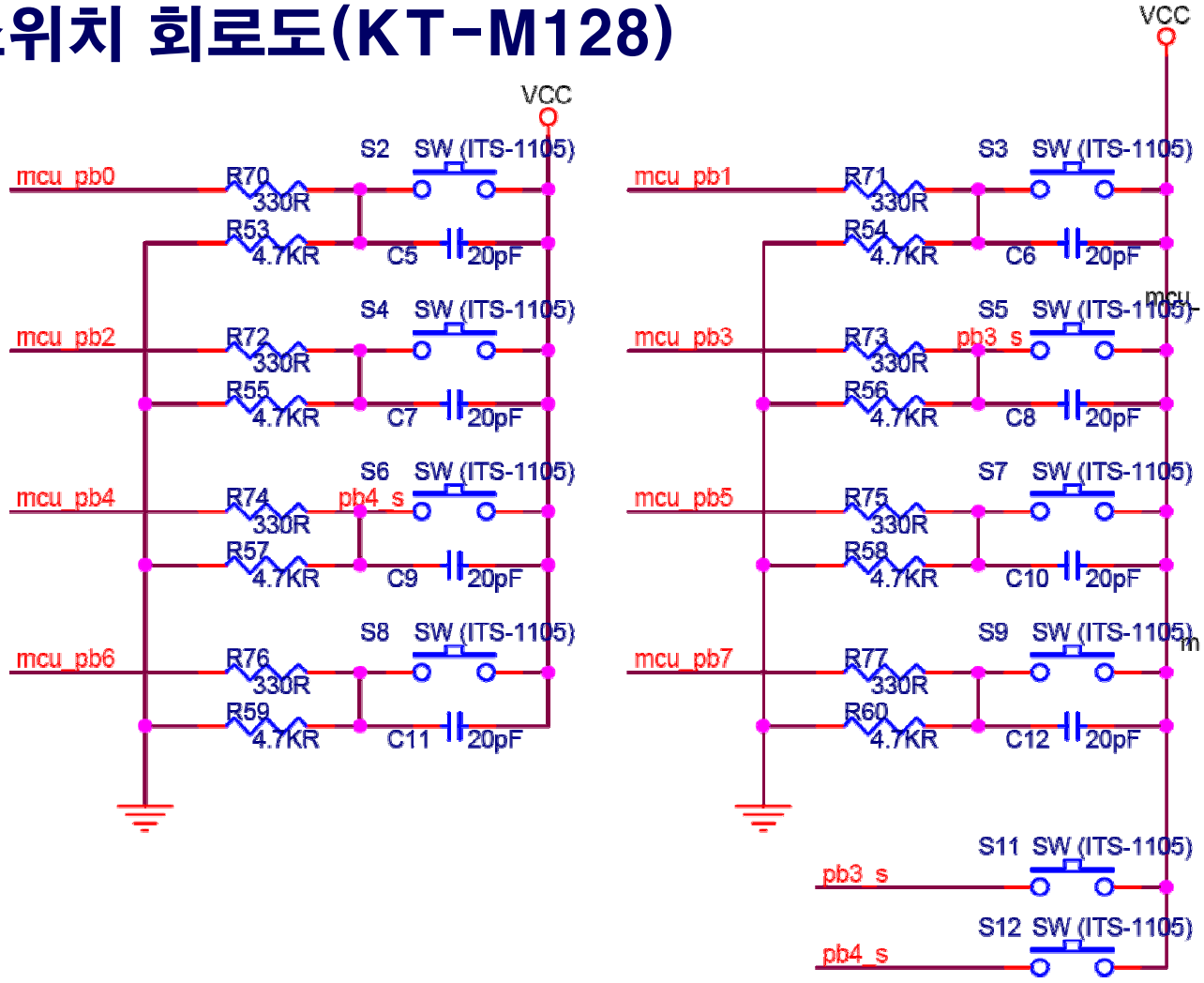
[스위치]

- 스위치 회로도(KD-128)



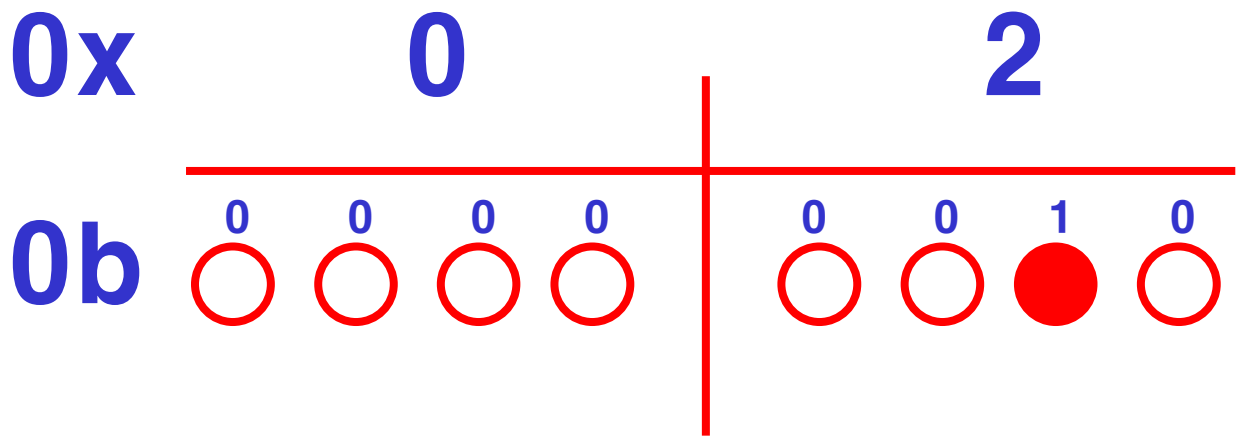
[스위치]

- 스위치 회로도(KT-M128)



[스위치 제어]

- 스위치를 눌렀을 경우 해당 비트를 1로 반환한다.



[스위치 제어]

```
void main(void)
{
	volatile unsigned char i, count=0;
	volatile unsigned char *sw_in;
	volatile unsigned char *led_out;
	init_devices();
	sw_in=(volatile unsigned char *)0x36;
	led_out=(volatile unsigned char *)0x8008;

	printf("\n\r\n\nKT-M128 V01\n\r");
	printf("Trainning Board Lab5 Example.\n\r");

	while(1){
		*led_out = *sw_in;
		delay(200);
		printf("count : %d\n\r", count);
		count++;
	}
}
```

[스위치 제어]

- 스위치를 1번 누르면 해당 LED가 점등되고, 다시 1번 누르면 소멸되게 작성하시오.

[스위치 채터링]

- 스위치를 한번을 눌렀을 경우에도 여러 번 눌렀다고 판단한다.
  - 스위치를 누른 횟수는 크리스탈 클럭과 시간에 비례한다.

- 여러 번 눌렸을 경우 한번만 인식하기 위해 채터링을 사용한다.
  - 채터링의 종류
    - 선 채터링
    - 후 채터링

[선 채터링 예제]

- 채터링으로 인하여 대기 한 후, 스위치를 떼고 나면 처리내용을 실행
  - 스위치에서 누르고 있는 경우 while에서 대기, 떼고 나면 처리내용 실행

```
if(PINB & 0b00010000){
        while(PINB & 0b00010000)
                ;
        //처리내용
}
```

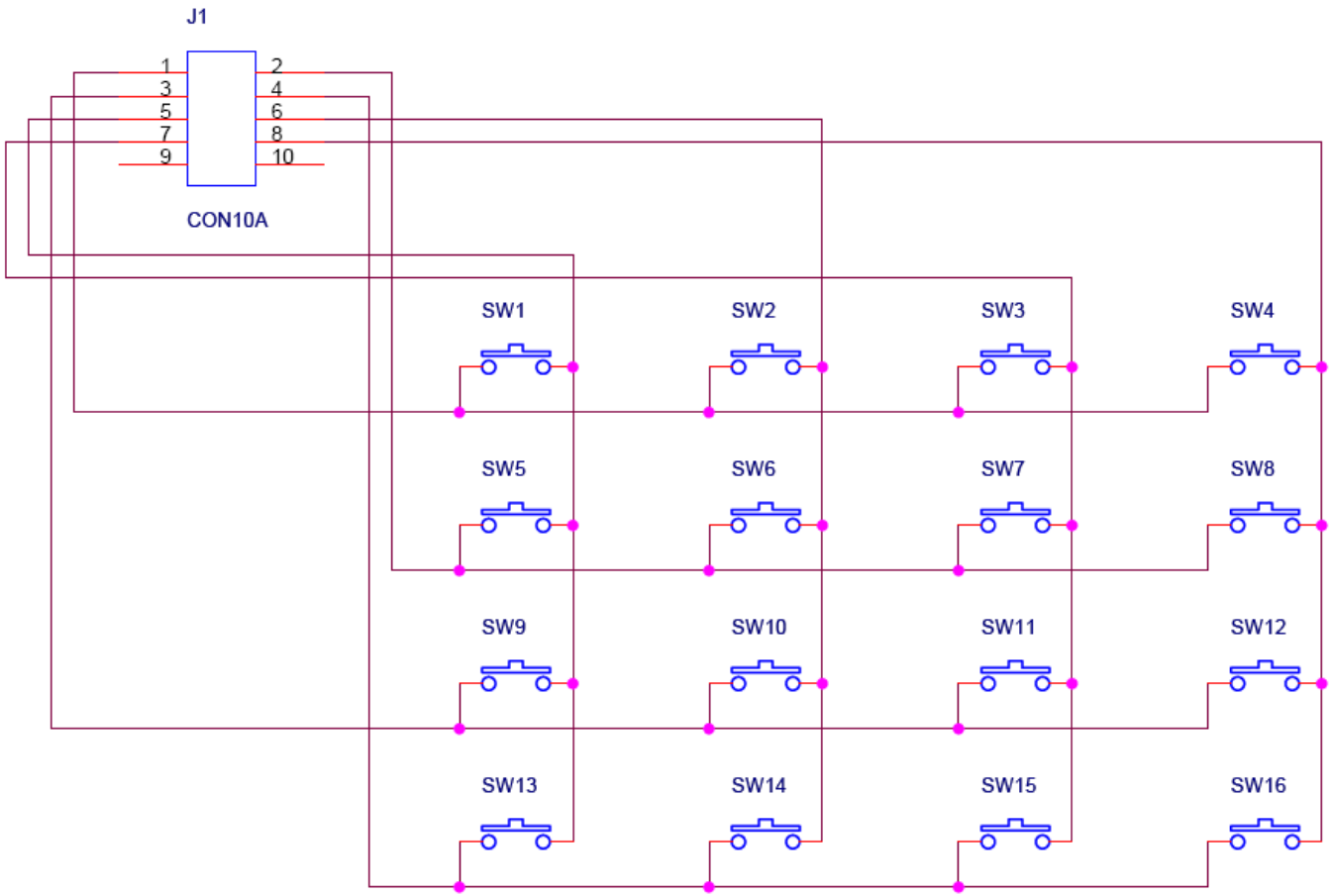
[후 채터링 예제]

- 처리내용이 1번 실행 되고, 채터링으로 인하여 다음 명령어를 실행하지 못하게 함.
  - 처리내용을 먼저 실행 후 while에서 대기

```
if(PINB & 0b00010000){
        //처리내용
        while(PINB & 0b00010000)
                ;

}
```

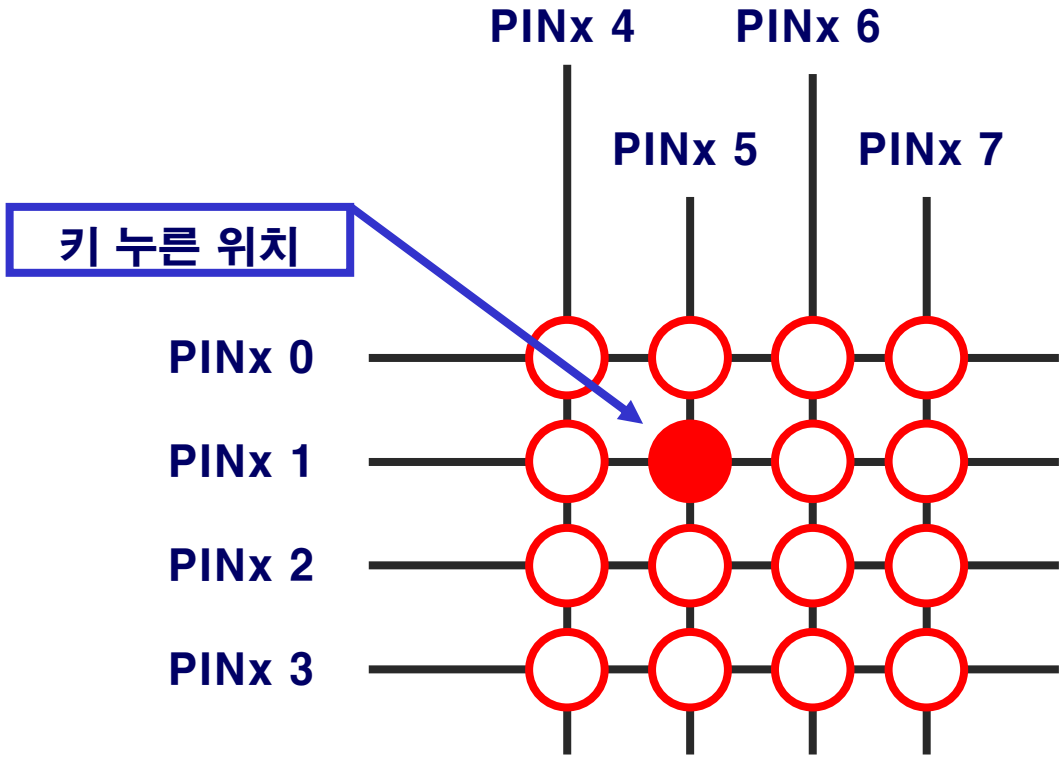
[4 x 4 위치(키) 매트릭스]

- 회로도



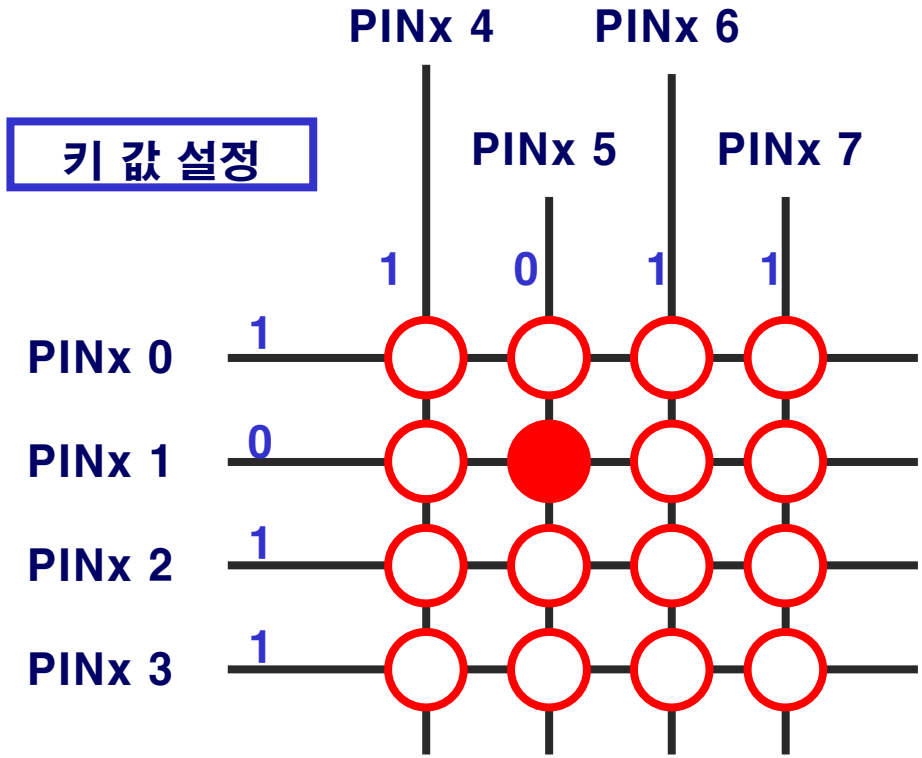
[4 x 4 스위치(키) 매트릭스]

- 스위치 배열



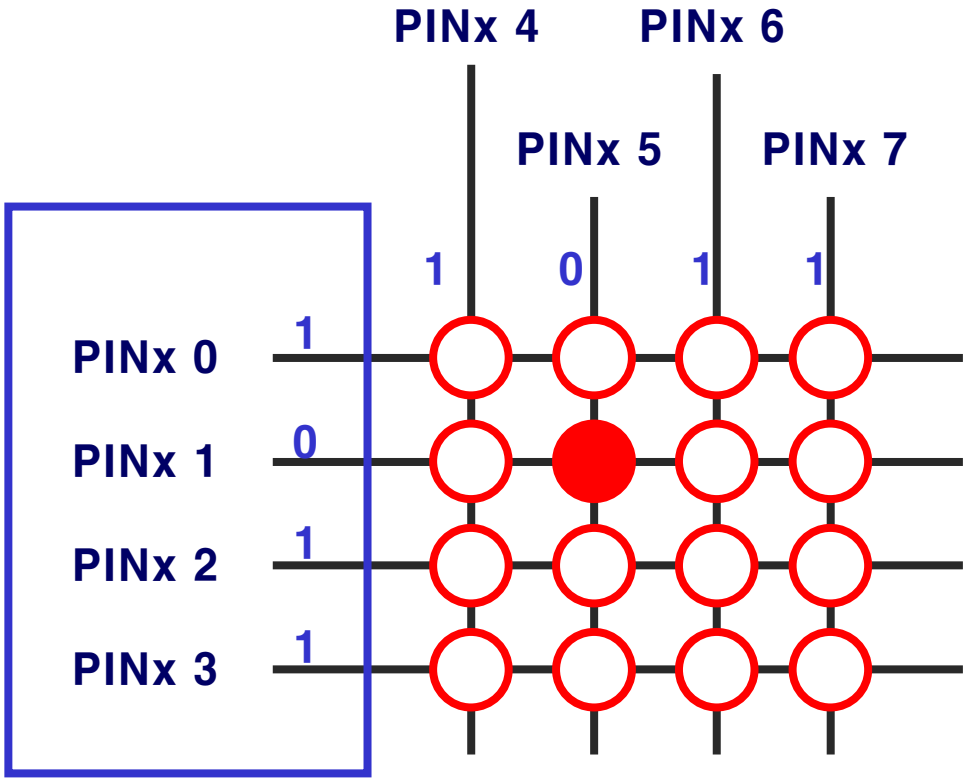
[4 x 4 스위치(키) 매트릭스]

- 스위치 배열



[4 x 4 스위치(키) 매트릭스]

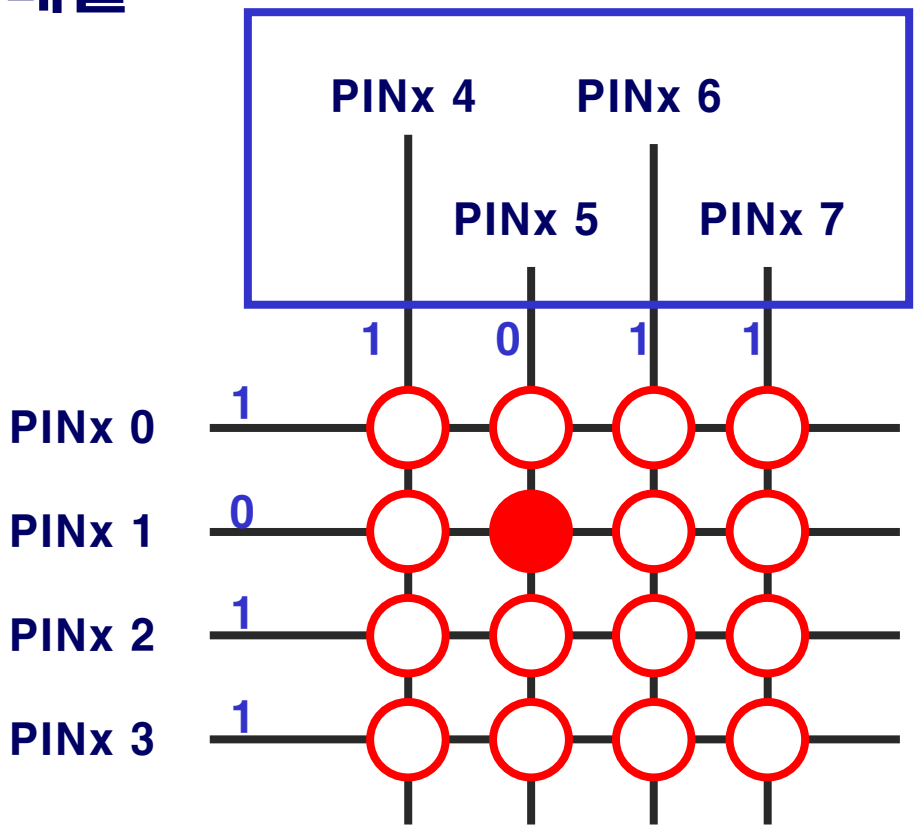
- 스위치 배열



1. 하위 4비트를 먼저 읽어 온다.

[4 x 4 스위치(키) 매트릭스]

- 스위치 배열



2. 상위 4비트를 읽어온다

[4 x 4 스위치(키) 매트릭스]

```
unsigned char PORTC_KEY(){
        unsigned char left, right, result;

        DDRC    = 0x0F;
        PORTC   = 0x0F;
        left    = PINC;

        DDRC    = 0xF0;
        PORTC   = 0xF0;
        right   = PINC;

        result          = left | right;
        return          result;
}
```

[숙제]

- 스위치(키) 매트릭스와 LED 매트릭스를 연결하여, 매트릭스로 누른 키와 매칭되는 LED 매트릭스의 LED를 점등 시키시오. (KD-128)

- 스위치(키) 매트릭스를 눌렀을 경우 해당 키가 2 x 2 라고 가정할 때, FND를 이용하여 “22”라고 표현될 수 있도록 작성하시오.(KT-M128)