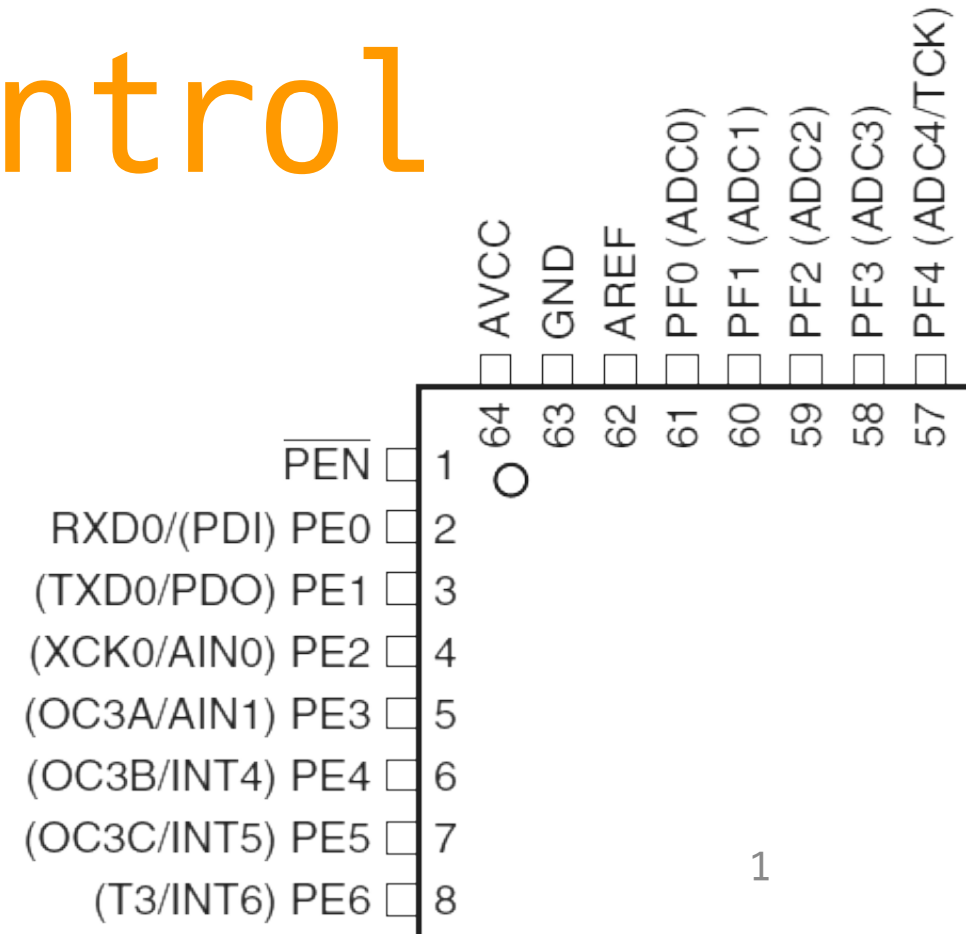
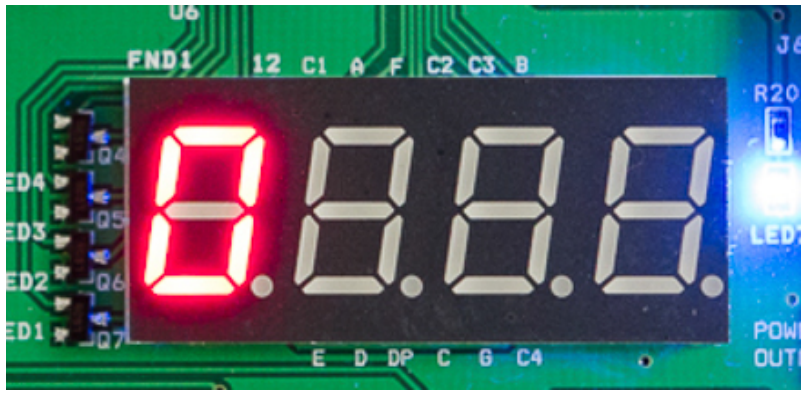
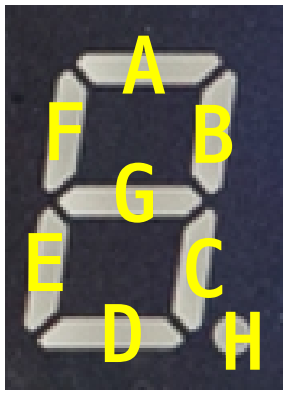
7 Segment Control



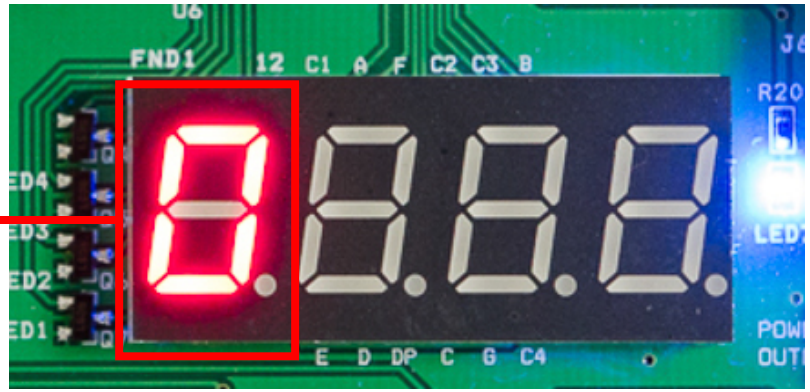
•7 Segment

- 숫자 표시를 위해 7+1개의 LED를 묶어 한 자리의 숫자를 표시할 수 있도록 제작한 소자
  - 숫자 표시를 위해 7개의 LED + 소수점 표시를 위해 1개의 LED 사용
  - 각 LED Segment에 A ~ H로 이름을 붙임.
- 설계에 따라 Common Anode형과 Common Cathode형으로 나뉨
  - 각 LED의 Anode 또는 Cathode를 하나로 묶어서 사용함.
  - 7-SEG를 구동할 때 필요한 Pin 개수를 줄일 수 있음. (후술)





•7 Segment - Common Anode & Common Cathode



하나의 숫자를 제어하기 위해서 8개의 신호가 필요

: 그러면 4자리 숫자는 32개의 신호가 필요

: GPIO를 직접 7-segment에 연결한다면 AVR의 대부분의 GPIO를 사용하게 됨.(ATmega 128의 전체 핀은 몇 개?)

: 다른 방법이 필요함.

•7 Segment – Common Anode & Common Cathode

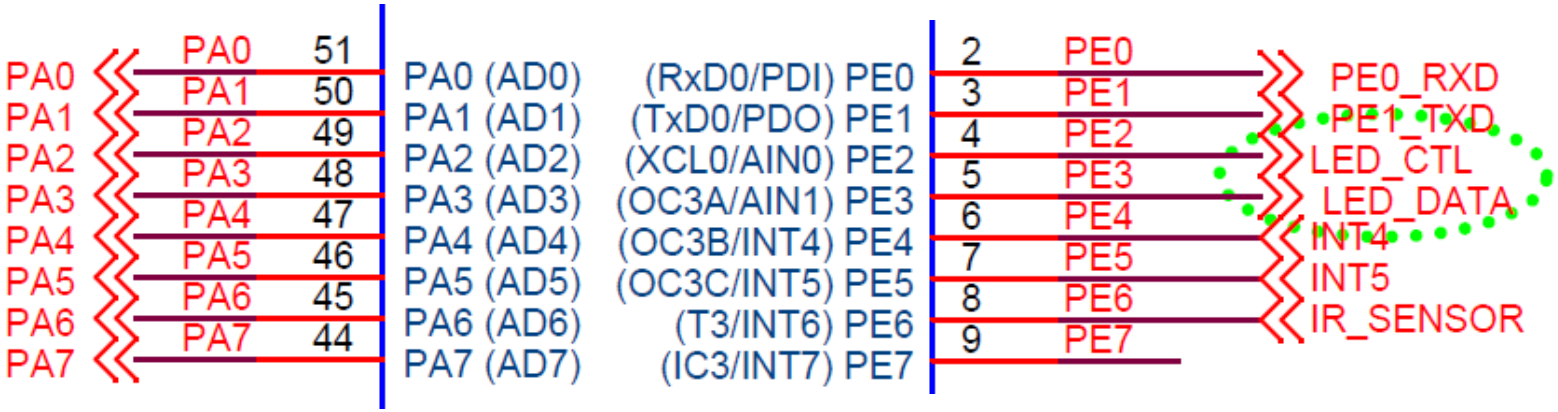
➢Schematic 1, 4 Page 참고

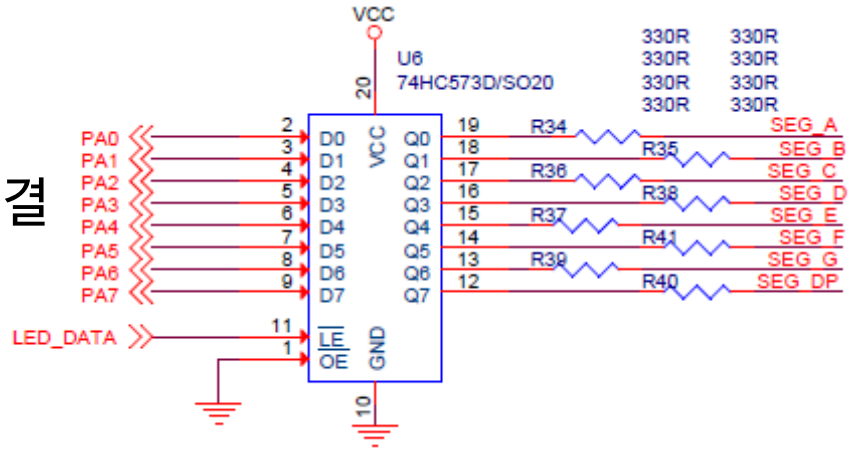
➢7 Segment 구동에 Port A 전체와 PE2, PE3사용

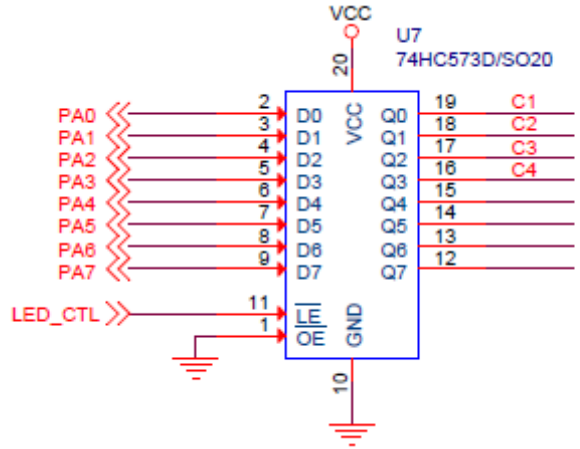
➢각 Pin은 7 Segment에 직접 연결되지 않음

➢대신 U6, U7으로 연결되고, U6, U7의 Qn이 7-Seg에 연결

➢74HC573D은 무슨 소자인지 조사하여 레포트에 반영

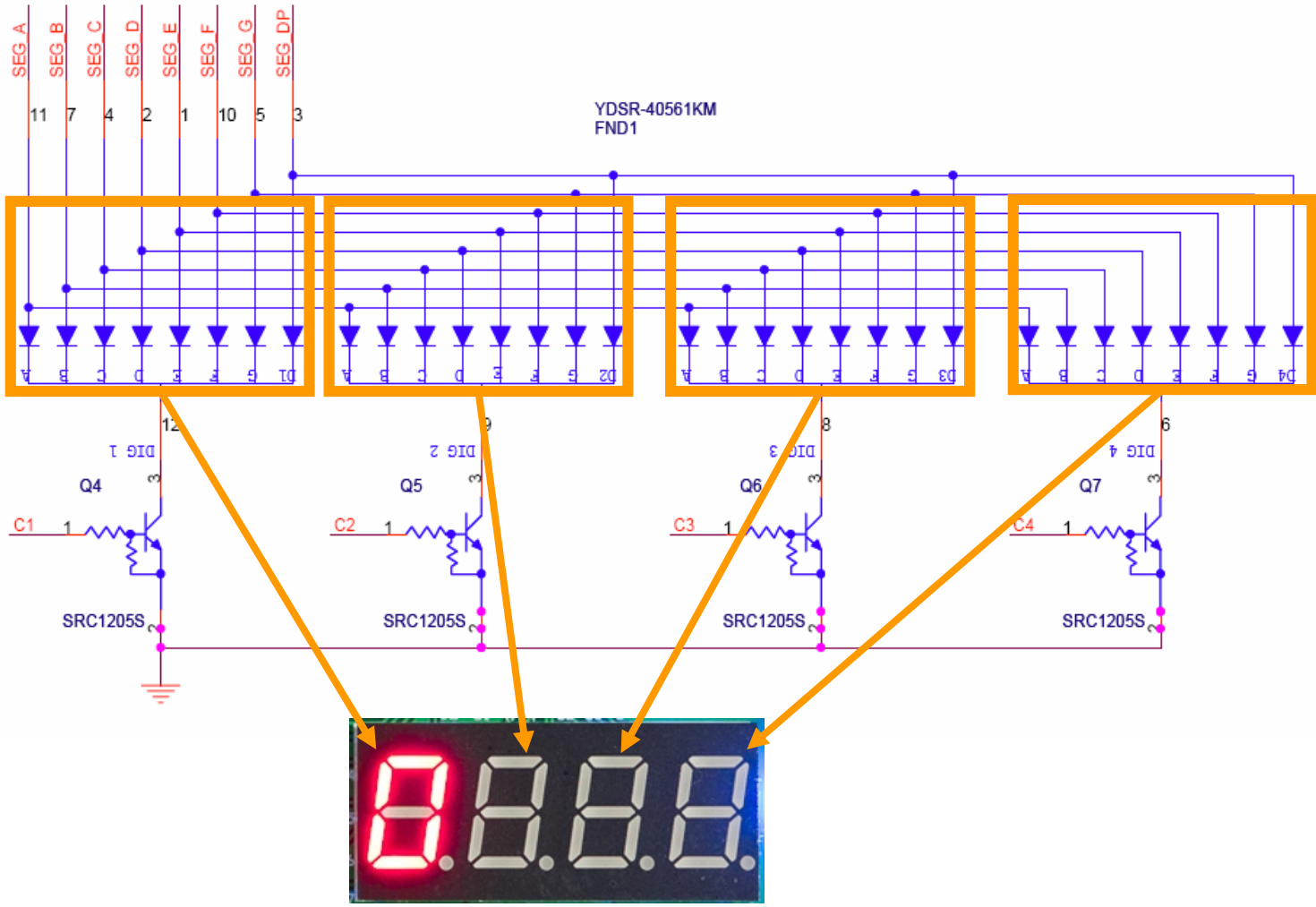






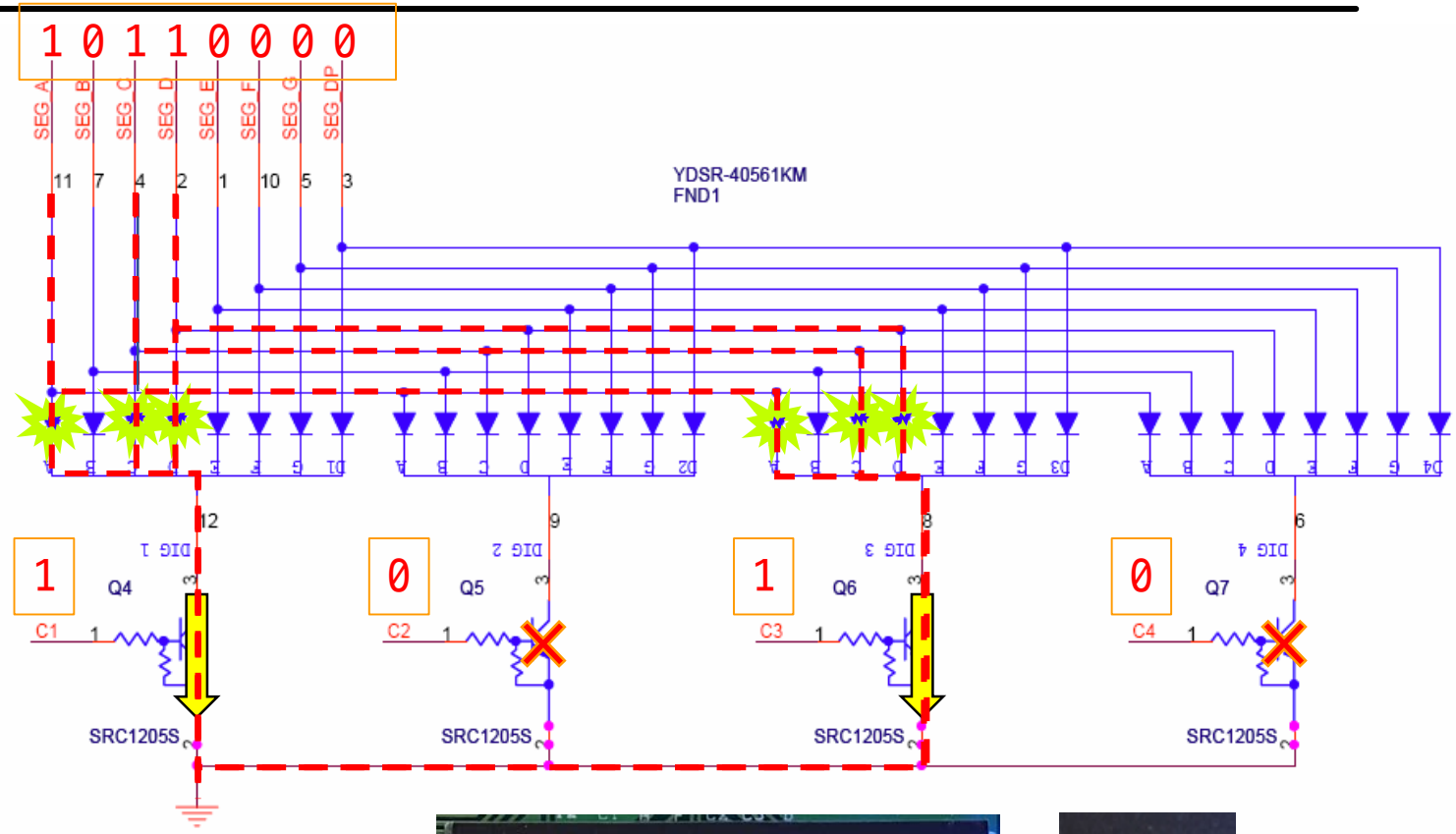
•7 Segment – Common Anode & Common Cathode

- ➢Schematic 4 Page 참고
  - ➢각 자리수의 LED의 Cathode가 공통배선 되어 있음
    - ➢Common Cathode 방식의 7-Seg.
  - ➢각 자리의 Cathode는 TR에 연결되어 있음
    - ➢Cn이 Low인 경우, Cathode는 GND와 Hi-Z상태임.
    - ➢즉, SEG x가 Hi여도 LED가 켜질 수 없음.



•7 Segment – Common Anode & Common Cathode

- ➢예를 들어...
  - ➢C1, C3 = Hi
  - ➢C2, C4 = Low
  - ➢SEG A, SEG C, SEG D = Hi
  - ➢나머지 SEG = Low라면...
  - ➢첫째 자리, 셋째 자리의 7-SEG만 켜 짐.
  - ➢또한 각 자리의 SEG A, SEG C, SEG D만 켜 짐.



• 예제 1

```
#include <avr/io.h>
#include <util/delay.h>

unsigned char
FND_SEGNP[10]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x27,0x7F,0x6F};
unsigned char
FND_SEGWP[10]={0xBF,0x86,0xDB,0xCF,0xE6,0xED,0xFD,0xA7,0xFF,0xEF};
unsigned char FND_SEGPOS[4]={0x01,0x02,0x04,0x08};

void init_port()
{
    DDRA = 0xFF;
    DDRE = 0x0C;
    PORTE = 0x04;
    PORTA = 0x0F;
}
```

```
int main()
{
    init_port();
    int i;
    while(1)
    {
        for(i=0;i<10;i++)
        {
            PORTE = 0x08;
            PORTA = FND_SEGNP[i];
            _delay_ms(500);
        }
    }
    return 0;
}
```

• 예제 1의 동작

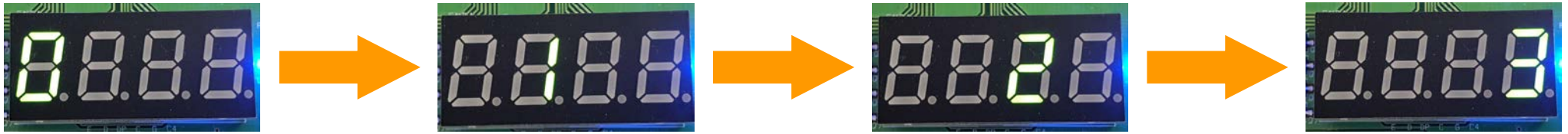
- 매 0.5초마다 숫자가 바뀜
  - 각 자리마다 같은 숫자가 출력
    - 왜? -> 레포트 반영
  - PORTA = FND_SEGNP[i]; 를 PORTA = FND_SEGWP[i]; 로 바꾸어 빌드하여 실행해본다.
    - 실행 결과가 바뀌는가?
      - 왜? -> 레포트 반영

•7 Segment - 숫자 표시

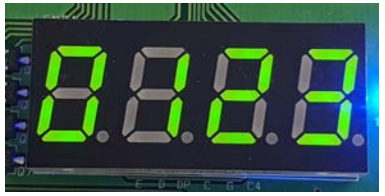
➢각 자리마다 다른 숫자를 표시하려면...?

➢한 번에 한 자리 씩 숫자를 켜고 끄는 것을 매우 빠르게 반복한다.

➢잔상때문에 동시에 켜져 있는 것처럼 보임.



매우 빠르게 반복



• 예제 2

```
#include <avr/io.h>
#include <util/delay.h>

unsigned char
FND_SEGNP[10]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,
0x27,0x7F,0x6F};
unsigned char
FND_SEGWP[10]={0xBF,0x86,0xDB,0xCF,0xE6,0xED,0xFD,
0xA7,0xFF,0xEF};
unsigned char FND_SEGPOS[4]={0x01,0x02,0x04,0x08};

void init_port()
{
    DDRA = 0xFF;
    DDRE = 0x0C;
    PORTE = 0x04;
    PORTA = 0x0F;
}
```

```
int main(void)
{
    init_port();
    int j;
    int k=70;
    for(j = 0; j<4 ; j++)
    {
        PORTA = FND_SEGPOS[j];
        PORTE = 0X04;
        PORTE = 0X00;
        PORTA = FND_SEGNP[j];
        PORTE = 0X08;
        PORTE = 0X00;
        _delay_ms(1000);
    }
```

```
    while(1)
    {
        for(j = 0; j<4 ; j++)
        {
            PORTA = FND_SEGPOS[j];
            PORTE = 0X04;
            PORTE = 0X00;
            PORTA = FND_SEGNP[j];
            PORTE = 0X08;
            PORTE = 0X00;
            _delay_ms(k);
        }
        if(k>=2)
            k=k-1;
    }
    return 0;
}
```

• 예제 2의 동작

- 한 자리 씩 숫자가 번갈아 가며 출력됨
  - 시간이 지나면서 속도가 점점 빨라 짐
    - _delay_ms(k)의 k값이 줄어들면서 점점 빨라 짐
  - 속도가 일정 속도 이상 빨라지면, 모두 켜져 있는 것 처럼 보임.
    - 실제로는 한 자리 씩 켜져 있지만, 잔상때문에 모두 켜져 있는 것 처럼 보임.
- WinAVR대신 AVR toolchain을 사용하는 경우
  - 빌드할 때 최적화 옵션을 -O0로 설정
    - 6주차 강의자료 참고
    - 본 예제만 -O0로 설정하여 동작시킵니다.
      - 과제 참고 사항 참고
- CPU 동작 주파수 반드시 설정
  - 6주차 강의자료 참고

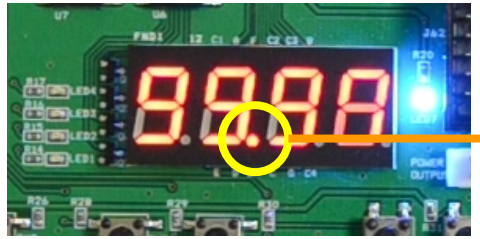
• 과제

➢ _delay_ms() 함수를 이용하여 적당히 정확한 초시계 만들기

➢ 또는 _delay_us() 함수 사용

10s	1s	1/10s	1/100s

(99.97초에서 99.98초로 넘어가는 순간)



7-segment에 점(point)을 찍어서 정수 부분과 소수 부분을 구분해 봅시다.

➢ 맨 처음 시작할 때(= reset버튼을 눌렀을 때) 숫자 0000이 떠있는 상태 유지

➢ 0000에서 증가하거나 감소하지 않습니다.

➢ SW5을 누를 경우 START

➢ 숫자가 증가.

➢ 동작 중에 SW5을 누를 경우 숫자 증가 정지 및 현재 시간 값 표시.

➢ 정지상태에서 다시 누를 경우, 현재 값부터 숫자가 다시 증가함. 즉 SW5는 증가/정지를 바꾸는 역할을 함.

➢ 동작 중 / 정지 상태에서 SW6을 누를 경우 0000으로 초기화 및 동작 정지

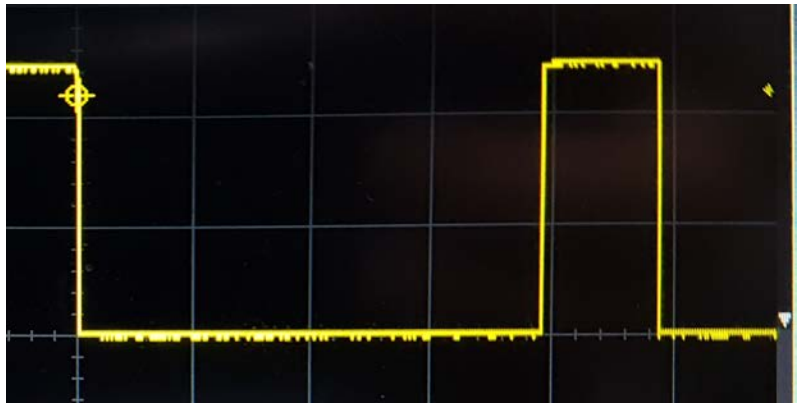
•과제 참고 사항

➢버튼을 한 번 눌렀는데 두 번 인식해요!!

- ➢스위치 바운싱
  - ➢ 또는 채터링 현상이라고 함.
- ➢스위치의 접점이 서로 붙거나 떨어지는 과정에서 물리적으로 문제가 생기기 때문에 스위치가 여러 번 눌리는 것처럼 작동함.

1. 회로를 설계할 때 RC 회로, FF, Latch등을 추가하여 해결
2. 프로그램을 짤 때 스위치입력에 대하여 여러 처리를 하여 해결
   1. SW 입력을 받자 마자 작동하지 말고 Delay를 가진 후 작동하는 작동 하는 것으로 해결
   2. SW 입력을 받고 나서 입력을 일정 시간 이상 유지하면 입력된 것으로 판단하여 해결

- ➢ 이 외에도 다양한 해결 방법이 있음.



•과제 참고 사항

➢C언어 연산자 우선 순위?

- ➢연산자 우선 순위를 다 외울 수 있나요?
  - ➢틀리지 않고 항상 정확하게?
- ➢항상 괄호를 써서 연산 순위를 명확하게!
  - ➢`while((PING&0x0F) != 0x0F);`
  - ➢`while(PING&0x0F != 0x0F);`
    - ➢이 둘의 연산 결과는 다름

```
../          .c:88: warning: suggest parentheses around comparison in operand of &
```

- ➢`while(PING&0x0F != 0x0F);`를 컴파일 할 때 생기는 경고
- ➢컴파일 할 때 뜨는 경고를 반드시 확인합시다.
  - ➢경고 내용을 반드시 확인하고 의도한 동작인지 다시 생각해봅니다.

•과제 참고 사항

➢부동소수점 연산

- ➢ATmega128에는 부동소수점 연산을 위한 FPU가 없음
  - ➢따라서 double, float형 연산이 대단히 느림
  - ➢또한 math.h에 있는 다양한 수학 함수들의 동작이 매우 느림.
    - ➢sin, cos, pow...
  - ➢만약 수학 함수 값을 써야 한다면...
    1. 느린 것을 감안하고 그냥 함수를 사용한다.
    2. 특정 수학 함수 값만 사용할 것으로 예상된다면 미리 그 값을 계산하여 배열 등으로 선언하고, 그 값을 사용한다.
       - ➢Look Up Table이라고 함.
       - ➢int pow10[5] = {1,10,100,1000,10000};

```
// pow(10,x)에서 x가 0, 1, 2, 3, 4 중 하나로 고정되는 경우
// pow10[x] 형태로 참고 가능
```

•과제 참고 사항

- _delay_ms, _delay_us함수의 동작
  - 컴파일러의 최적화 옵션에 따라 delay함수의 동작이 달라집니다.
    - 컴파일러 옵션을 바꾸면서 적당한 최적화 옵션을 찾아봅니다.
      - -O1, -O2, -O3로 설정하고 동작시키면 큰 문제는 없습니다.
  - CPU 동작 주파수 설정
    - delay함수는 컴파일 시 CPU의 동작 주파수를 참고하여 빌드됩니다.
      - 따라서 6주차 강의자료를 참고하여 동작 주파수를 설정해야합니다.