

3. 4자리 7세그먼트 모듈

앞 절에서 1자리 7세그먼트 표시장치를 제어하는 방법을 살펴보았다. 7세그먼트 표시장치에는 1자리 이외에도 2자리 이상의 숫자를 표시하기 위한 표시장치를 흔히 볼 수 있다. 하지만 1자리 7세그먼트 표시장치를 제어하기 위해 8개의 GPIO 핀이 필요했다는 것을 기억하는가? 2자리 7세그먼트 표시장치를 제어하기 위해서는 16개의 GPIO 핀이, 4자리 7세그먼트 표시장치를 제어하기 위해서는 32개의 GPIO 핀이 필요해지므로 20개의 GPIO 핀을 사용할 수 있는 아두이노 우노로는 제어가 불가능하다. 하지만 아두이노 우노로 4자리 7세그먼트 표시장치를 제어하는 예를 어렵지 않게 찾아볼 수 있다. 어떻게 가능할까? 4자리 7세그먼트 표시장치를 제어하기 위해서는 잔상효과를 이용하여야 한다. 물론 32개의 GPIO 핀을 사용한다면 가능하기는 하지만 32개의 핀을 사용하는 경우는 찾아볼 수 없다. 사람의 눈은 반응 속도가 느려 눈앞의 물건이 사라진 이후에도 잠깐 동안은 물건이 사라진 것을 눈치 채지 못한다. 망막에 맺힌 상이 잠깐 동안은 그대로 유지되며 이를 잔상 효과라고 한다. 잔상 효과를 이용하여 빠른 속도로 4자리 숫자를 반복적으로 켜준다면 사람의 눈은 4자리 숫자가 동시에 표시되는 것과 동일하게 느끼게 된다. 일반적으로 4자리 7세그먼트 표시장치는 12개의 핀으로 이루어지며²⁾ 1자리 7세그먼트 표시장치와 마찬가지로 소숫점이 있는 면을 아래로 했을 때 가장 왼쪽 핀이 1번이며 시계 반대 방향으로 핀 번호는 증가한다.

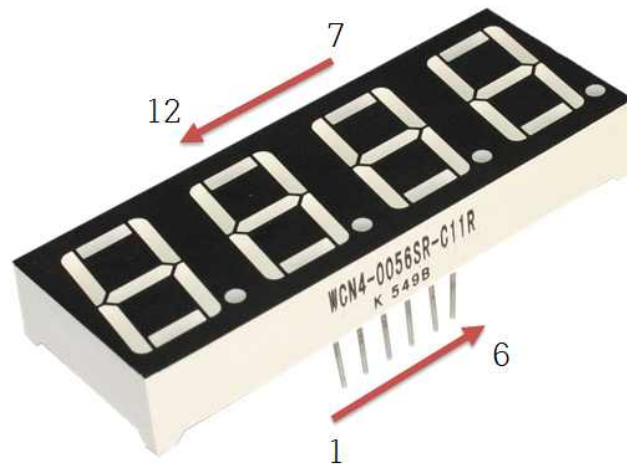


그림 5. 4자리 7세그먼트 표시장치

2) 4자리 7세그먼트 표시장치의 경우 시간 표시를 위해 콜론(:)이나 작은따옴표 등이 포함된 예도 흔히 볼 수 있으며, 이 경우 이들 부호를 제어하기 위해 별도의 제어핀을 사용하므로 12개 이상의 핀을 가질 수 있다.

12개의 핀 중 8개는 8개의 세그먼트 제어를 위해 사용된다. 이는 1자리 7세그먼트 표시장치의 경우와 동일하다. 나머지 4개는 4자리 숫자 중 특정 순간에 어느 자리에 출력할 것인지 결정하기 위해 사용한다. 4자리 7세그먼트 표시장치의 회로도에는 그림 6과 같다.

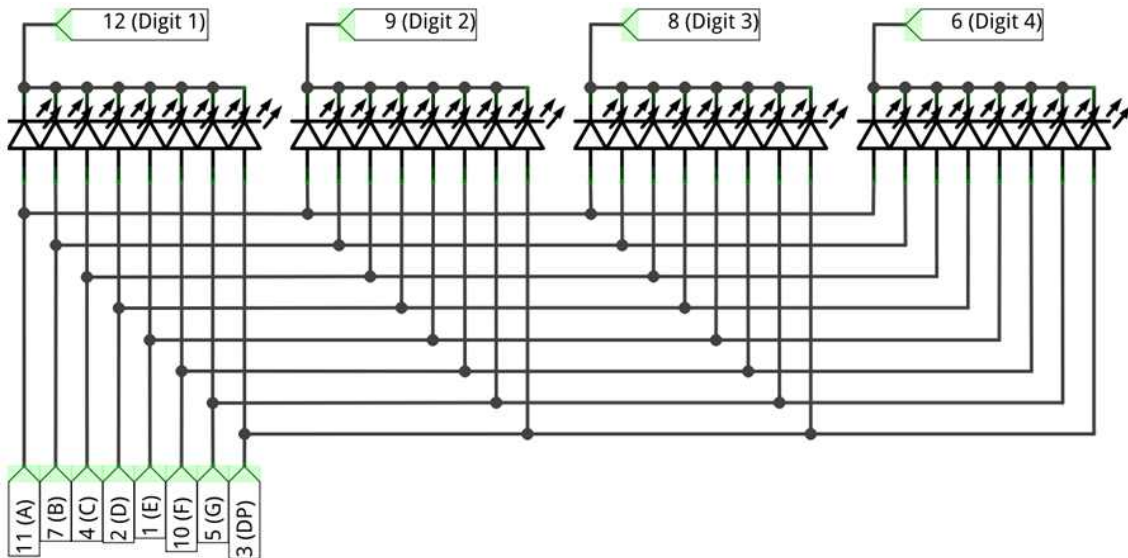


그림 6. 4자리 7세그먼트 표시장치 회로도

그림 6에서 아래쪽 8개 핀은 세그먼트 제어 핀으로 숫자는 그림 5에서의 핀 번호를 나타내고 괄호 안의 기호는 세그먼트의 이름을 나타낸다. 세그먼트 중에서 켜고 싶은 핀에는 1의 값을, 끄고 싶은 핀에는 0의 값을 가해주면 된다. 1자리 7세그먼트 표시장치에서와 마찬가지로 일반적인 세그먼트의 순서와 핀 번호 사이에 연관성이 없으므로 연결 시에 주의하여야 한다.

위쪽 4개 핀은 4자리 숫자 중 하나를 선택하기 위해 사용되는 핀으로 자리는 왼쪽을 1번으로 시작해서 오른쪽으로 증가한다. 특정 순간에는 하나의 자리에만 숫자를 표현하는 것이 일반적이다. 따라서 켜고 싶은 자리의 핀에는 0을, 끄고 싶은 자리의 핀에는 1을 가해준다. 이처럼 특정 자릿수 선택을 위해 0을 가해주어야 하는 4자리 7세그먼트 장치를 공통 음극 방식이라고 하며 이 장에서 사용할 장치 역시 공통 음극 방식이다. 반대로 자릿수 선택을 위해 1을 가해주어야 하는 공통 양극 방식의 4자리 7세그먼트 장치도 있지만 이 경우 세그먼트 제어핀 역시 해당 세그먼트를 켜기 위해서는 공통 음극 방식과는 반대로 0을 가해주어야 한다.³⁾

4자리 7세그먼트 모듈을 그림 7과 같이 연결해 보자.

3) 인터넷에서 찾아볼 수 있는 일부 문서의 경우 공통 양극 방식과 공통 음극 방식을 반대로 설명하는 경우도 있지만, 이 책에서는 자릿수 선택 핀에 가하는 전압에 따라 공통 양극 방식과 공통 음극 방식을 구별한다.

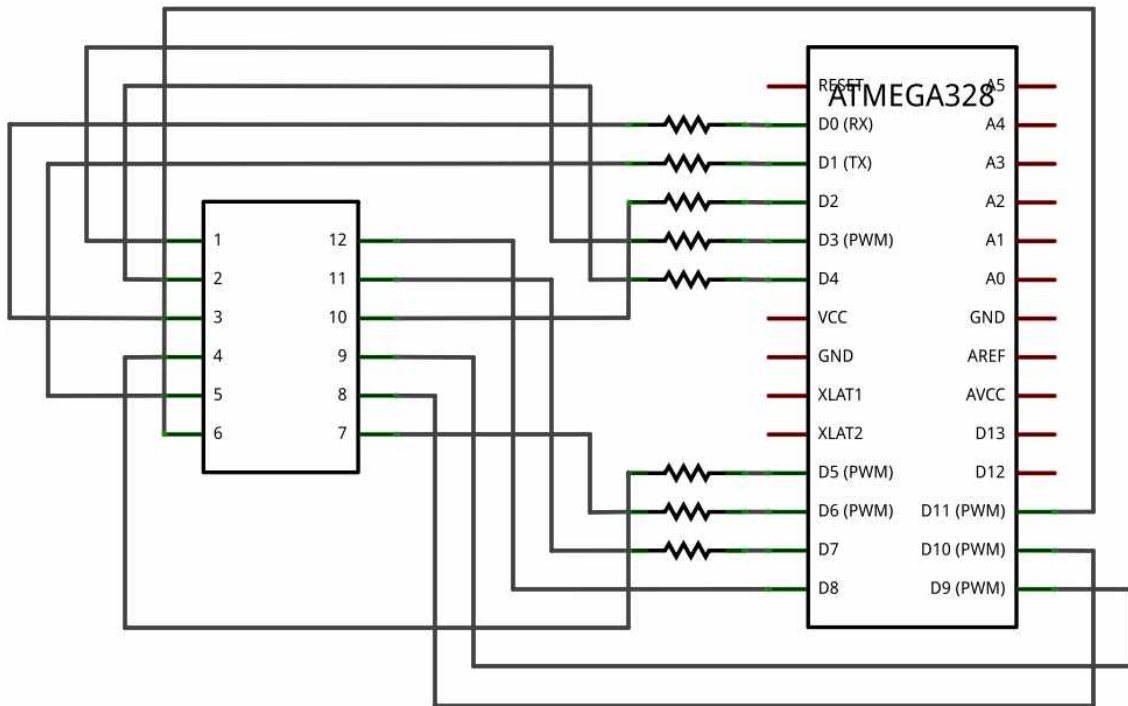


그림 7. 4자리 7세그먼트 모듈 연결 회로도

코드 6은 첫 번째 자리에 0부터 9까지 0.1초 간격으로 출력하고 오른쪽으로 이동하기를 반복하는 코드의 예이다.

코드 6. 4자리 7세그먼트 표시장치에 한 자리 숫자 나타내기

```
#define F_CPU 16000000L
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    uint8_t numbers[]
        = {0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE4, 0xFE, 0xE6};
    int i, j;

    DDRD = 0xFF; // 세그먼트 제어핀 8개를 출력으로 설정
```

```

    DDRB = 0x0F;                                // 자릿수 선택핀 4개를 출력으로 설정

    while(1)
    {
        for(i = 0; i < 4; i++){
            // 해당 자리에만 0(GND)을 출력하고 나머지에는 1을 출력하여 자리 선택
            PORTB = 0x0F & ~(1 << i);
            for(j = 0; j < 10; j++){
                PORTD = numbers[j];              // 숫자 데이터 출력
                _delay_ms(100);                  // 0.1초 간격으로 전환
            }
        }
    }

    return 1;
}

```

코드 6은 4자리 중 한 곳에만 숫자를 출력하고 있다. 여러 자리에 동시에 숫자를 출력하기 위해서는 잔상 효과를 이용하여야 하며 반복적으로 숫자를 출력하여 표시장치를 갱신하여야 한다. 따라서 `_delay_ms` 함수를 사용하게 되면 표시장치가 갱신되지 않아 한 자리의 숫자만 보이거나 희미하게 보이게 된다. 코드 7은 1000부터 0.1초 간격으로 1씩 증가하는 타이머를 `_delay_ms` 함수를 사용하여 구현한 예이다. LED는 반응 속도가 그리 빠르지 않으므로 `SEGMENT_DELAY`를 지정하여 각 자리가 완전히 켜지도록 하고 있다.

코드 7. 4자리 7세그먼트 표시장치에 4자리 숫자 나타내기 I

```

#define F_CPU 16000000L
#include <avr/io.h>
#include <util/delay.h>

#define SEGMENT_DELAY 5

int main(void)
{

```

```
uint8_t numbers[]
    = {0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE4, 0xFE, 0xE6};
int count = 1000;                                // 현재 표시할 숫자 (0 ~ 9)
int thousands, hundreds, tens, ones;

DDRD = 0xFF;                                     // 세그먼트 제어핀 8개를 출력으로 설정
DDRB = 0x0F;                                     // 자릿수 선택핀 4개를 출력으로 설정

while(1)
{
    thousands = count / 1000;                     // 천의 자리
    hundreds = count / 100 % 10;                 // 백의 자리
    tens = count / 10 % 10;                      // 십의 자리
    ones = count % 10;                           // 일의 자리

    PORTB = 0x0E;                                 // 천의 자리 출력
    PORTD = numbers[thousands];
    _delay_ms(SEGMENT_DELAY);

    PORTB = 0x0D;                                 // 백의 자리 출력
    PORTD = numbers[hundreds];
    _delay_ms(SEGMENT_DELAY);

    PORTB = 0x0B;                                 // 십의 자리 출력
    PORTD = numbers[tens];
    _delay_ms(SEGMENT_DELAY);

    PORTB = 0x07;                                 // 일의 자리 출력
    PORTD = numbers[ones];
    _delay_ms(SEGMENT_DELAY);

    _delay_ms(100);                               // 0.1초 대기

    count++;
    if(count == 9999) count = 1000;               // 1000 ~ 9999까지 반복
}
```

```

    return 1;
}

```

코드 7에서 문제가 되는 부분은 카운터 값을 증가시키기 이전에 0.1초 대기하는 `_delay_ms(100)` 문장으로 잔상 효과는 0.1초 보다 짧아 0.1초 이내에 출력을 갱신해야 하지만 출력을 갱신하지 못해 발생한다. 이를 `millis` 함수를 통해 수정한 예가 코드 8이다. 한 번에 하나의 자리만을 표시하는 것은 코드 7과 동일하지만 갱신 주기가 짧으므로 모든 자리가 동시에 켜져있는 것처럼 보인다.

코드 8. 4자리 7세그먼트 표시장치에 4자리 숫자 나타내기 II

```

#define F_CPU 16000000L
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define SEGMENT_DELAY 5

// 1 마이크로초 당 클록의 수
#define CLOCKS_PER_MICRO ( F_CPU / 1000000L )
// 클록 수를 마이크로초로 변환
#define CLOCKS_TO_MICROSECONDS(a) ( (a) / CLOCKS_PER_MICRO )
// 타이머0의 분주비를 64로 설정한 경우
// 오버플로 인터럽트가 발생하기까지의 마이크로초 단위 경과 시간
#define MICROSECONDS_PER_TIMER0_OVERFLOW
    ( CLOCKS_TO_MICROSECONDS(64 * 256) )
// 오버플로 인터럽트가 발생한 경우 밀리초 단위 증가 시간
#define MILLIS_INCREMENT_PER_OVERFLOW
    ( MICROSECONDS_PER_TIMER0_OVERFLOW / 1000 )
// 오버플로 인터럽트가 발생한 경우
// 밀리초 단위 증가를 제외하고 남는 마이크로초 단위 증가 시간
#define MICROS_INCREMENT_PER_OVERFLOW
    ( MICROSECONDS_PER_TIMER0_OVERFLOW % 1000 )

```

```
// 프로그램 시작 이후의 경과 시간
volatile unsigned long timer0_millis = 0;
volatile int timer0_micros = 0;

ISR(TIMER0_OVF_vect)
{
    unsigned long m = timer0_millis;
    int f = timer0_micros;

    m += MILLIS_INCREMENT_PER_OVERFLOW;    // 밀리초 단위 시간 증가
    f += MICROS_INCREMENT_PER_OVERFLOW;    // 마이크로초 단위 시간 증가

    // 마이크로초가 1000을 넘어가면 밀리초를 증가시킴
    int micro_to_millis = f / 1000;
    m += micro_to_millis;
    f = f % 1000;

    timer0_millis = m;
    timer0_micros = f;
}

unsigned long millis()
{
    unsigned long m;
    uint8_t oldSREG = SREG;                // 상태 레지스터 값 저장

    // timer0_millis 값을 읽는 동안
    // timer0_millis 값이 변하지 않도록 인터럽트를 비활성화
    cli();

    m = timer0_millis;

    SREG = oldSREG;                        // 이전 상태 레지스터 값 복원

    return m;                             // 프로그램 시작 후 경과 시간
}
```

```

}

int main(void)
{
    uint8_t numbers[]
        = {0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE4, 0xFE, 0xE6};
    int count = 1000;                // 현재 표시할 숫자 (0 ~ 9)
    int thousands, hundreds, tens, ones;

    TCCR0B |= (1 << CS01) | (1 << CS00); // 분주비를 64로 설정
    TIMSK0 |= (1 << TOIE0);              // 오버플로 인터럽트 허용
    sei();                                // 전역적으로 인터럽트 허용

    unsigned long time_previous, time_current;
    time_previous = millis();             // 시작 시간

    DDRD = 0xFF;                         // 세그먼트 제어핀 8개를 출력으로 설정
    DDRB = 0x0F;                         // 자릿수 선택핀 4개를 출력으로 설정

    while(1)
    {
        thousands = count / 1000;        // 천의 자리
        hundreds = count / 100 % 10;    // 백의 자리
        tens = count / 10 % 10;          // 십의 자리
        ones = count % 10;               // 일의 자리

        PORTB = 0x0E;                   // 천의 자리 출력
        PORTD = numbers[thousands];
        _delay_ms(SEGMENT_DELAY);

        PORTB = 0x0D;                   // 백의 자리 출력
        PORTD = numbers[hundreds];
        _delay_ms(SEGMENT_DELAY);

        PORTB = 0x0B;                   // 십의 자리 출력
        PORTD = numbers[tens];
    }
}

```

```

    _delay_ms(SEGMENT_DELAY);

    PORTB = 0x07;                // 일의 자리 출력
    PORTD = numbers[ones];
    _delay_ms(SEGMENT_DELAY);

    time_current = millis();      // 현재 시간

    if(time_current - time_previous > 100){
        time_previous = time_current;
        count++;
        if(count == 9999) count = 1000;    // 1000 ~ 9999까지 반복
    }
}

return 1;
}

```

4. 4자리 7 세그먼트 표시장치 - 아두이노

아두이노 역시 delay 함수를 사용하여 4자리 7세그먼트 표시장치를 제어할 수 없으며 millis 함수를 사용하여야 한다. 코드 9는 그림 8과 동일한 동작을 아두이노 스타일로 구현한 예이다.

코드 9. 4자리 7세그먼트 표시장치에 4자리 숫자 나타내기 - 아두이노

```

#define SEGMENT_DELAY 5

// 세그먼트 연결 핀 'dp → g → f → e → d → c → b → a' 순서
int segment_pins[] = {0, 1, 2, 3, 4, 5, 6, 7};
int digit_pins[] = {8, 9, 10, 11};    // 자리수 선택 핀

uint8_t numbers[]
    = {0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE4, 0xFE, 0xE6};

```

```
unsigned long time_previous, time_current;
int count = 1000;

void setup(){
    for(int i = 0; i < 8; i++)                // 세그먼트 제어핀을 출력으로 설정
        pinMode(segment_pins[i], OUTPUT);
    for(int i = 0; i < 4; i++)                // 자릿수 선택핀을 출력으로 설정
        pinMode(digit_pins[i], OUTPUT);

    time_previous = millis();                // 시작 시간
}

void loop(){
    int thousands, hundreds, tens, ones;

    thousands = count / 1000;                // 천의 자리
    hundreds = count / 100 % 10;            // 백의 자리
    tens = count / 10 % 10;                 // 십의 자리
    ones = count % 10;                      // 일의 자리

    show_digit(1, thousands);
    show_digit(2, hundreds);
    show_digit(3, tens);
    show_digit(4, ones);

    time_current = millis();                // 현재 시간

    if(time_current - time_previous > 100){
        time_previous = time_current;
        count++;
        if(count == 9999) count = 1000;      // 1000 ~ 9999까지 반복
    }
}

// 지정한 위치 position (1 ~ 4)에 지정한 숫자 number (0 ~ 9) 출력
void show_digit(int position, int number){
```

```
for(int i = 0; i < 4; i++){
    if(i + 1 == position) digitalWrite(digit_pins[i], LOW);
    else digitalWrite(digit_pins[i], HIGH);
}

for(int i = 0; i < 8; i++){
    boolean on_off = bitRead(numbers[number], i);    // 비트값 읽기
    digitalWrite(segment_pins[i], on_off);           // 세그먼트 제어
}
delay(SEGMENT_DELAY);
}
```