

Chapter 10. 디지털 데이터 출력

마이크로컨트롤러는 디지털 컴퓨터의 일종으로 디지털 데이터를 입력받아 이를 처리하고 그 결과를 출력하는 것을 기본으로 한다. 디지털 컴퓨터에서 데이터의 입출력 기본 단위는 비트이며 이는 마이크로컨트롤러의 핀 하나에 해당한다. 즉, 마이크로컨트롤러의 핀 하나로 1비트의 데이터를 주고받을 수 있다. 이 장에서는 ATmega128의 데이터 핀으로 1비트의 데이터를 출력하는 방법을 알아볼 것이며, 출력된 데이터의 확인을 위해서는 LED를 사용한다.

1. ATmega128의 데이터 핀

ATmega128은 64개의 핀을 가지는 마이크로컨트롤러이다. 이 중 전원, 크리스탈 등을 위한 11개의 핀을 제외한 53개의 핀은 디지털 데이터 입출력 핀으로 사용할 수 있다. 즉, ATmega128은 동시에 53 비트의 데이터를 주변장치와 교환하는 것이 가능하다. 이들 디지털 데이터 입출력 핀은 중앙처리장치(CPU)에서의 연산 단위인 워드(word)에 해당하는 포트(port)로 묶여져 있다. ATmega128에서 워드는 1바이트 크기를 가지고 있으므로 포트 역시 1바이트 크기를 가진다. 따라서 53개의 디지털 데이터 입출력 핀은 A에서 G까지 7개의 포트로 나뉜다. 53이 8의 배수가 아니므로 마지막 G 포트에 해당하는 ATmega128의 핀은 5개만 존재한다. 그림 1은 ATmega128 칩의 핀과 포트를 나타낸 것으로 포트 G를 제외하면 모두 연속된 핀으로 포트가 구성되어 있다.

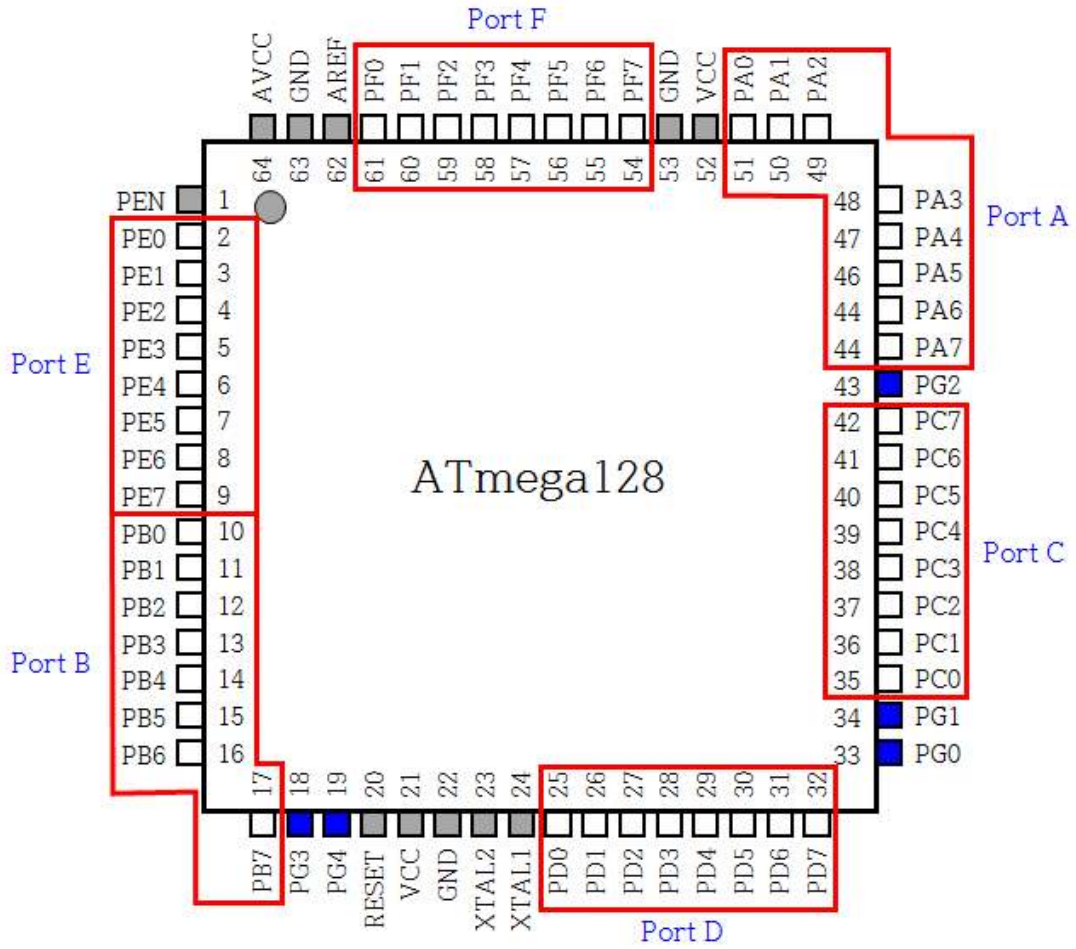

그림 1. ATmega128 칩의 핀과 포트

포트를 통해 디지털 데이터를 주고받기 위해서는 무엇이 필요할까? 마이크로컨트롤러는 마이크로프로세서와 다르게 주변장치와 데이터를 주고받는 기능이 칩 내에 포함되어 있으며, 주변장치와 입출력 핀을 통해 데이터를 교환하기 위해 사용되는 임시 기억 공간을 '입출력 레지스터'라고 한다. 마이크로프로세서에서의 레지스터는 주로 중앙처리장치 내에 연산 과정에서 사용되는 '범용 레지스터'를 나타낸다면 마이크로컨트롤러에서는 이와 더불어 주변장치와 데이터 교환을 위해 사용되는 입출력 레지스터를 함께 가리킨다. ATmega128에는 256개의 레지스터를 정의할 수 있으며 이 중 32개의 범용 레지스터와 105개의 입출력 레지스터가 정의되어 있다. ATmega128을 공부하는 것은 105개의 입출력 레지스터 사용 방법을 배우는 것이라 해도 과언은 아니다.

2. 디지털 데이터 출력을 위한 레지스터

포트를 통해 디지털 데이터를 출력하기 위해 사용되는 레지스터는 PORTx (x = A, ..., G) 레지스터이다. PORTx 레지스터의 구조는 그림 2와 같다. 7개의 포트 중 포트 G는 5개의 핀만을 사용할 수 있으므로 PORTG 레지스터에서도 상위 3비트는 사용할 수 없다.

비트	7	6	5	4	3	2	1	0
비트 이름	PORTx7	PORTx6	PORTx5	PORTx4	PORTx3	PORTx2	PORTx1	PORTx0
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

(a) PORTA ~ PORTF

비트	7	6	5	4	3	2	1	0
비트 이름	-	-	-	PORTG4	PORTG3	PORTG2	PORTG1	PORTG0
읽기/쓰기	R	R	R	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

(b) PORTG

그림 2. PORTx 레지스터 구조

ATmega128 칩의 7번 핀인 PE5 핀으로 1의 값을 출력하고 싶다면 PORTE 레지스터의 5번 비트를 1로 설정하면 된다. 간단하지 않은가?

* 포트와 핀

포트는 8개의 핀을 하나로 묶어서 관리하기 위해 사용된다. 하지만 마이크로컨트롤러에서 디지털 데이터 입출력의 기본 단위는 비트라고 이야기 하였다. 그렇다면 굳이 8개의 핀을 묶어서 포트로 관리하는 이유는 무엇일까?

포트 단위로 핀을 묶어서 관리하는 이유 중 하나는 CPU에서 데이터를 비트 단위로 처리할 수 없기 때문이다. CPU에서 데이터를 처리하는 단위는 워드(word)로 바이트의 정수배로 정해진다. ATmega128의 경우 1바이트의 워드 크기를 가지고 있으므로 ATmega128의 CPU에서 처리할 수 있는 데이터의 최소 크기는 1바이트가 된다.

1비트 데이터로 할 수 있는 일은 무엇일까? 마이크로컨트롤러를 배울 때 가장 먼저 해

보는 LED를 점멸하는 예나 버튼 상태를 읽어 들이는 예가 비트 단위의 데이터 입출력에 해당한다. LED나 버튼은 ON 또는 OFF의 두 가지 상태 중 하나만을 가지므로 비트 단위의 데이터로 표현하거나 제어할 수 있다. 하지만 실제로 CPU가 데이터를 처리하기 위해서는 최소한 바이트 단위의 데이터가 필요하며, 메모리 역시 바이트 단위로 주소가 정해져 있으므로 바이트 단위로만 읽거나 쓸 수 있다. C 언어에서 가장 작은 크기의 메모리를 필요로 하는 데이터 타입은 1바이트 크기를 갖는 char 타입이며 비트 단위의 데이터를 저장하는 데이터 타입은 존재하지 않는다.

CPU가 바이트 단위로 데이터를 처리한다면 비트 단위로 각각의 핀을 제어하는 것이 불가능해 보일 수도 있다. 하지만 비트 연산자를 생각해 보자. 비트 연산자는 바이트 이상의 데이터에서 특정 비트의 값만을 변경하기 위해 사용될 수 있다. 즉, 실제로 CPU 내에서 연산은 바이트 단위로 이루어지고 출력 역시 레지스터를 통해 바이트 단위로 이루어지지만 비트 연산자를 통해 비트 단위의 데이터만을 변경함으로써 특정 핀의 비트 단위 출력값을 변경할 수 있다.

ATmega128의 데이터 핀으로 데이터를 출력하기 위해 PORTx 레지스터를 사용하면 되지만 한 가지 잊지 말아야 할 점이 데이터 핀은 입출력이 가능한 핀이라는 점, 그리고 입력이나 출력으로 사용될 수 있지만 입력과 출력으로 동시에 사용될 수는 없다는 점이다. 따라서 데이터 핀을 사용하기 이전에 데이터 핀을 입력으로 사용할 것인지 또는 출력으로 사용할 것인지 먼저 결정하여야 한다. 데이터 핀을 입력 또는 출력으로 사용하기 위해 사용되는 레지스터는 DDRx (x = A, ..., G) (Data Direction Register) 레지스터이다. DDRx 레지스터의 구조는 그림 3과 같다. 7개의 포트 중 포트 G는 5개의 핀만을 사용할 수 있으므로 DDRG 레지스터 역시 PORTG 레지스터와 마찬가지로 상위 3비트는 사용할 수 없다.

비트	7	6	5	4	3	2	1	0
비트 이름	DDx7	DDx6	DDx5	DDx4	DDx3	DDx2	DDx1	DDx0
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

(1은 출력, 0은 입력 상태)

(a) DDRA ~ DDRF

비트	7	6	5	4	3	2	1	0
비트 이름	-	-	-	DDG4	DDG3	DDG2	DDG1	DDG0
읽기/쓰기	R	R	R	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

(1은 출력, 0은 입력 상태)

(b) DDRG

그림 3. DDRx 레지스터 구조

DDR 레지스터의 각 비트인 DDRxn (x = A, ..., G, n = 0, ..., 7) 비트는 PORTxn 비트에 해당하는 핀의 입력 또는 출력 방향을 결정한다. DDRxn 비트가 1로 설정되면 출력으로, 0으로 설정되면 입력으로 동작하게 된다.

3. 블링크

포트에 연결된 LED를 점멸하는 코드를 작성해 보자. LED 점멸을 위한 코드는 먼저 DDR 레지스터의 해당 비트에 1을 출력하여 해당 핀을 출력으로 설정한 후, PORT 레지스터에 1 또는 0의 값을 출력함으로써 LED를 켜거나 끌 수 있다. 먼저 포트 B에 8개 LED를 연결한다. 포트 B는 ATmega128 칩의 10번부터 17번까지 8개 핀에 해당한다.

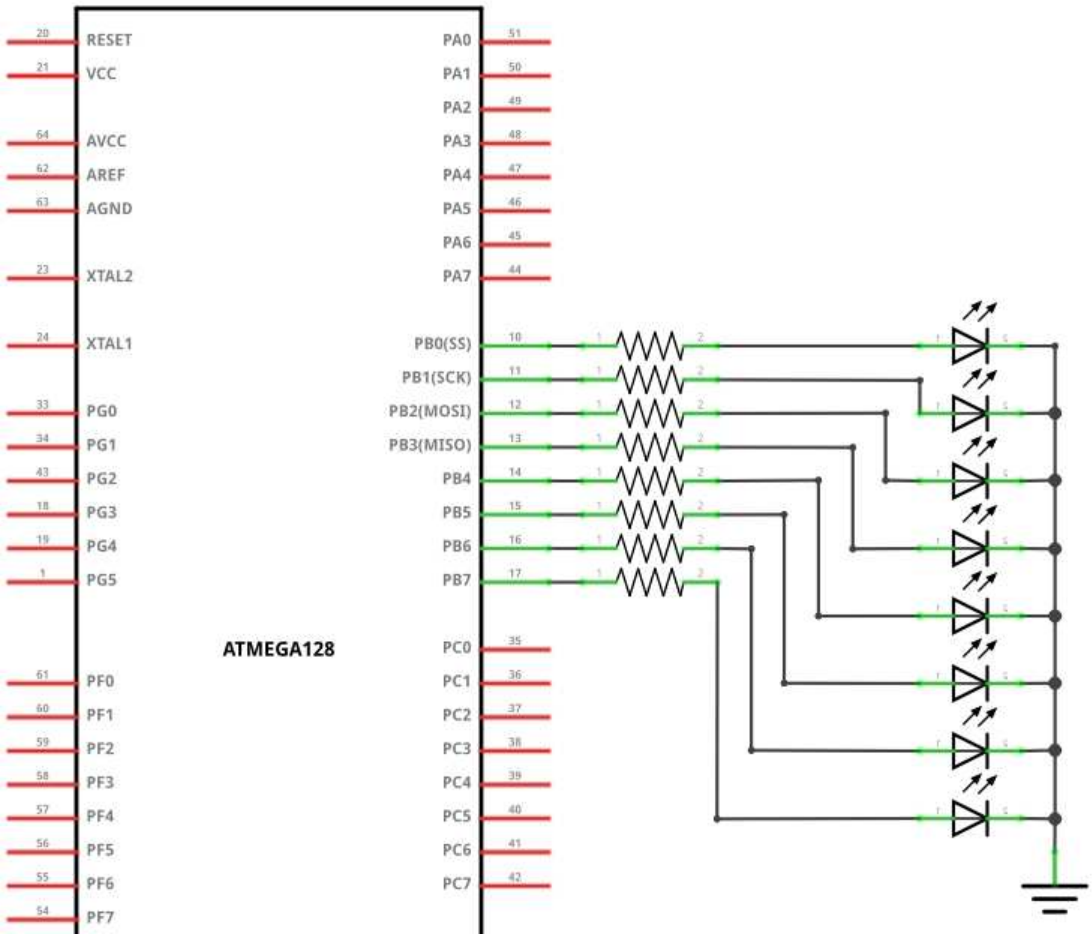

그림 4. 포트 B에 LED 연결 회로도[1)]

1) 가능한 간단하고 이해하기 쉽게 회로를 나타내기 위해 ATmega128 칩과 주변장치는 GND와 VCC에 연결된 것으로 가정하고 별도로 GND와 VCC 연결선을 표시하지는 않는다.

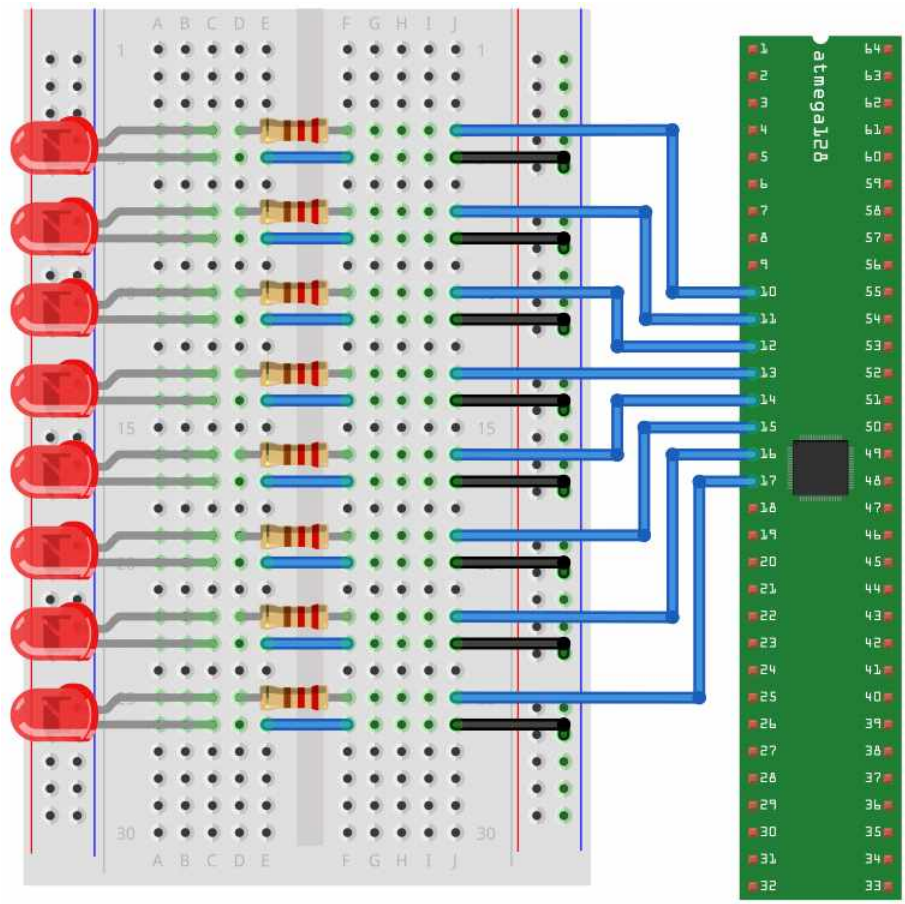

그림 5. 포트 B에 LED 연결 회로

코드 1은 포트 B에 연결된 8개의 LED를 1초 간격으로 점멸하는 예이다.

코드 1. Blink

```
1   #define F_CPU 16000000L
2   #include <avr/io.h>
3   #include <util/delay.h>
4
5   int main(void)
6   {
7     DDRB = 0xFF;
8
9     while (1)
10    {
```

```
11          PORTB = 0x00;
12          _delay_ms(1000);
13          PORTB = 0xFF;
14          _delay_ms(1000);
15      }
16  }
```

- Line 1 : 마이크로컨트롤러의 동작 주파수를 정의한다. 이 책에서는 ATmega128을 16MHz로 동작시키는 것을 기준으로 하므로 이에 해당하는 값을 정의한다. ATmega128에서 int 타입은 2바이트이므로 16M의 값의 표현할 수 없으므로 상수 뒤에 'L'을 붙여 long 타입임을 표시한다.
- Line 2 : 'io.h' 헤더 파일을 포함시킨다. io.h 헤더 파일에는 마이크로컨트롤러에 따른 레지스터 이름과 레지스터 비트 이름 등이 정의되어 있으므로 반드시 포함시켜야 한다. 프로젝트를 생성하면 io.h 헤더 파일을 포함하는 문장은 기본적으로 포함되어 있다.
- Line 3 : 시간 지연 함수 _delay_ms를 사용하기 위한 'delay.h' 헤더 파일을 포함시킨다.
- Line 7 : 포트 B의 8개 핀을 출력으로 설정한다.
- Line 9~15 : 메인 루프인 무한 루프에 해당한다. 마이크로컨트롤러 프로그램은 전원이 주어진 동안에는 종료하지 않고 계속 실행되는 것을 기본으로 한다.
- Line 11 : 포트 B의 8개 핀에 연결된 LED를 끈다.
- Line 12, 14 : 밀리초 단위로 지연 시간을 설정한다. 시간 지연 함수를 사용하기 위해서는 'delay.h' 파일이 반드시 포함되어야 하며, 마이크로초 단위의 지연 시간을 설정할 수 있는 _delay_us 함수도 함께 정의되어 있다.

그다지 복잡해 보이지는 않는다. 프로그램을 컴파일 하고 업로드 하여 8개 LED가 점멸하는지의 여부를 확인해 보자.