

## Chapter 99. ATmega328 마이크로컨트롤러

### 1. AVR

아트멜(Atmel)은 1984년 창립된 반도체 제조회사이다. 아트멜의 주력 상품은 플래시(flash)를 내장한 마이크로컨트롤러이지만 마이크로컨트롤러 이외에도 다양한 제품을 생산하여 판매하고 있다. 아트멜의 홈페이지<sup>1)</sup>의 제품 메뉴를 살펴보면 마이크로컨트롤러 이외에도 터치 솔루션, 자동차 관련 제품, 무선 통신 관련 제품, 메모리, 보안 관련 제품 등 다양한 제품을 판매하고 있음을 볼 수 있다. 이들 중 마이크로컨트롤러 제품은 8051 계열, ARM 계열, AVR 계열의 크게 3부류로 나뉜다. 8051은 1980년 인텔(Intel)에서 발표한 8비트 마이크로컨트롤러로 1990년대 초까지 모뎀, 게임기, 타자기 등에 광범위하게 사용되었으며 지금도 여러 회사에서 호환 제품을 생산하고 있다. 아트멜 역시 8051 호환 마이크로컨트롤러를 생산하는 회사 중 하나이다.

마이크로컨트롤러 시장 분석에 따르면 최근<sup>2)</sup> 32비트 마이크로컨트롤러의 판매 금액이 8비트나 16비트 마이크로컨트롤러를 앞지른 것으로 나타나고 있다. 판매량에서는 8비트나 16비트 마이크로컨트롤러가 32비트 마이크로컨트롤러에 비해 현재까지는 많다. 하지만 고성능 마이크로컨트롤러에 대한 수요 증가와 32비트 마이크로컨트롤러의 가격 하락으로 인해 판매량에 있어서도 32비트 마이크로컨트롤러가 8비트나 16비트 마이크로컨트롤러를 추월할 시점이 멀지 않은 것으로 예측되고 있다. 이러한 마이크로컨트롤러 시장의 변화를 주도하는 회사 중 하나가 ARM이며 아트멜은 AVR 계열의 32비트 마이크로컨트롤러뿐만 아니라 ARM 아키텍처를 채용한 32비트 마이크로컨트롤러도 생산하고 있다.

AVR은 아트멜에서 1996년 발표한 마이크로컨트롤러로 변형된 하버드 구조를 사용하는 8비트 RISC 구조를 채택한 마이크로프로세서로, Norwegian Institute of Technology의 학생이었던 Alf-Egil Bogen과 Vegard Wollan에 의해 처음 고안되었다. AVR은 8비트 마이크로컨트롤러에서 시작하였지만 2006년 이후에는 32비트 AVR 마이크로컨트롤러도 생산하고 있다. 32비트 AVR은 멀티미디어 처리를 포함하여 확장된 기능을 제공하며 ARM과 경쟁 관계에 있지만 8비트 AVR이나 ARM과 호환되지는 않는다.

AVR에도 여러 가지 계열이 존재하며 타이니(tiny) 계열, 메가(mega) 계열, X-메가(xmega) 계열의 크게 3가지로 나눌 수 있다.

1) <http://www.atmel.com>

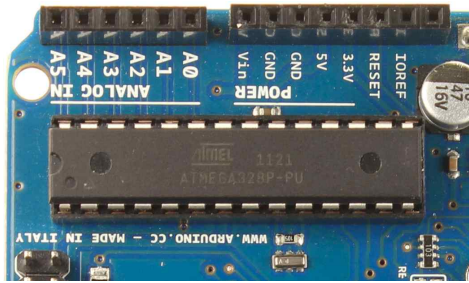
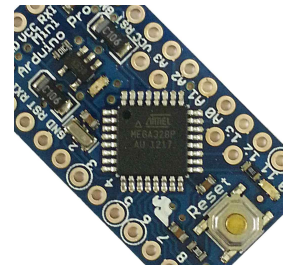
2) 2013년 기준

- 타이니 계열 : 0.5~16 KByte 크기의 프로그램 메모리와 6~32개의 핀을 가지는 마이크로컨트롤러로, 핀 수가 적어 한정된 주변 장치 연결 방법만을 지원하며 간단한 제어장치를 구성하기 위해 주로 사용된다.
- 메가 계열 : 4~512 KByte 크기의 프로그램 메모리와 28~100개의 핀을 가지는 마이크로컨트롤러로 확장된 명령어 집합(곱셈 명령, 프로그램 메모리 관리 명령 등)을 지원하고 타이니 계열에 비해 다양한 주변 장치 연결 방법을 제공한다.
- X-메가 계열 : 16~384 KByte 크기의 프로그램 메모리와 44~100개의 핀을 가지는 마이크로컨트롤러로 메가 계열에서 보다 확장된 기능을 제공하고 있다.

아두이노 우노(Arduino UNO)에 사용되었고 이 책에서 다루고자 하는 마이크로컨트롤러인 ATmega328과 학습용으로 널리 보급되어 있는 ATmega128 모두 메가 계열에 속하는 마이크로컨트롤러이다.

## 2. ATmega328 소개

이 책에서 다루고자 하는 마이크로컨트롤러는 아트멜의 ATmega328이다. 먼저 ATmega328 칩을 온라인 판매 사이트에서 검색해보자. ATmega328은 하나의 칩만을 가리키지는 않으며 ATmega328로 시작하면서 그 뒤에 다양한 기호가 붙은 여러 종류의 칩이 판매되고 있음을 볼 수 있다. 이들 ATmega328 제품은 크게 'ATmega328'로 시작하는 칩과 'ATmega328P'로 시작하는 칩의 두 종류가 있으며 후자는 'pico power'의 약자로 대기 모드에서 소비 전력을 줄인 개선된 버전에 해당한다. 아두이노 우노에서 사용되는 칩은 'ATmega328P-PU'가 일반적이며 두 번째 'P'는 PDIP (Plastic Dual In-line Package) 형태의 패키지를, 'U'는 환경 친화적인 무연(unleaded) 소재를 사용하였음을 나타낸다. 이 책에서도 아두이노 우노에 사용되는 ATmega328P-PU 형태의 칩을 사용한다. 하지만 작은 크기의 컨트롤 장치를 구성하고자 한다면 다른 형태의 패키지를 고려할 수도 있으며, 아두이노 공식 사이트에서도 TQFP (Thin Quad Flat Package) 패키지 형태의 ATmega328 칩을 사용한 작은 크기의 아두이노 나노(Nano), 아두이노 미니(Mini), 아두이노 프로(Pro) 등의 보드들도 볼 수 있다.

(a) PDIP 형태 - ATmega328P-PU<sup>3)</sup>

(b) TQFP 형태 - ATmega328P-AU

그림 1. ATmega328의 패키지 형태

ATmega328 칩과 유사한 칩으로는 ATmega168이 있다. ATmega168은 ATmega328과 핀 배열 및 기능 면에서 동일하며, 프로그램을 저장하기 위해 사용되는 플래시 메모리의 크기가 ATmega328의 절반인 16KByte라는 차이가 있다. 데이터 저장을 위한 SRAM과 EEPROM 역시 ATmega168은 ATmega328의 절반 크기를 갖는다. ATmega168이 ATmega328과 비교할 때 약간 저렴하므로 큰 메모리가 필요하지 않다면 ATmega168을 ATmega328 대용으로 사용할 수 있다. ATmega168을 사용하는 아두이노 보드가 아직 판매되고 있지만 대부분의 아두이노 보드들은 ATmega328을 사용하도록 변경되는 추세에 있다. 표 1은 ATmega328의 특징을 요약한 것으로 PDIP 패키지 기준이며 다른 형태의 패키지에서는 핀의 개수 등이 다를 수 있다. 그림 1을 자세히 살펴보면 TQFP 패키지의 경우 핀의 개수가 32개로 PDIP 패키지에 비해 많음을 알 수 있다.

3) 이미지 출처 : <http://arduino.cc/en/Main/Products>

| 항목       | 내용       | 비고                                                       |
|----------|----------|----------------------------------------------------------|
| 동작 전압    | 1.8~5.5V | 5V                                                       |
| 핀 수      | 28개      |                                                          |
| 비트       | 8        |                                                          |
| 입출력 가능 핀 | 23개      | 아두이노 우노에서는 20개의 핀만을 입출력으로 사용하며, 나머지는 외부 크리스탈과 리셋으로 사용한다. |
| 플래시 메모리  | 32KByte  |                                                          |
| SRAM     | 2KByte   |                                                          |
| EEPROM   | 1KByte   |                                                          |
| 동작 주파수   | 0~20MHz  | 아두이노 우노는 일반적으로 16MHz를 사용한다.                              |
| ADC 채널   | 6개       | 10비트 해상도를 가진다.                                           |
| PWM 채널   | 6개       |                                                          |

표 1. ATmega328의 특징

### 3. ATmega328 핀 배치

ATmega328P-PU 마이크로컨트롤러는 아두이노 보드 중에서 가장 기본이 되는 아두이노 우노에서 사용된 마이크로컨트롤러이다. 이 책에서는 ATmega328 마이크로컨트롤러를 이용하여 아두이노 스타일 및 일반 C 스타일로 프로그래밍 하는 방법을 비교하여 보여줄 것이므로 아두이노 우노에서 사용되는 ATmega328P-PU를 기준으로 한다.<sup>4)</sup>

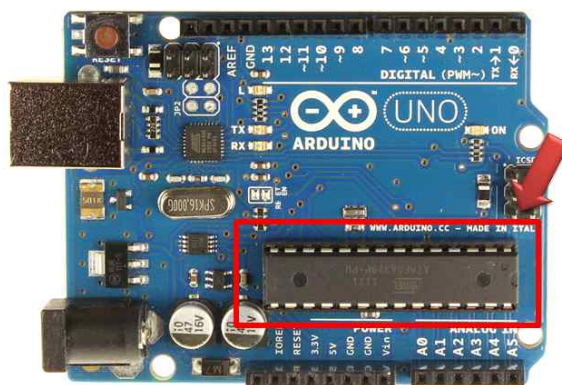


그림 2. 아두이노 우노에서 사용된 ATmega328P-PU

마이크로컨트롤러는 작은 컴퓨터로 마이크로컨트롤러의 핀을 통해 주변장치와 데이터를 주고

4) 특별히 언급하지 않는 한 이후 ATmega328은 ATmega328P-PU를 가리킨다.

받음으로써 컴퓨터로서 동작한다. 따라서 마이크로컨트롤러를 처음 접하게 되면 어느 핀이 어떤 역할을 하는 지 파악하는 것이 중요하다. ATmega328 마이크로컨트롤러의 핀 배치는 그림 3과 같다.

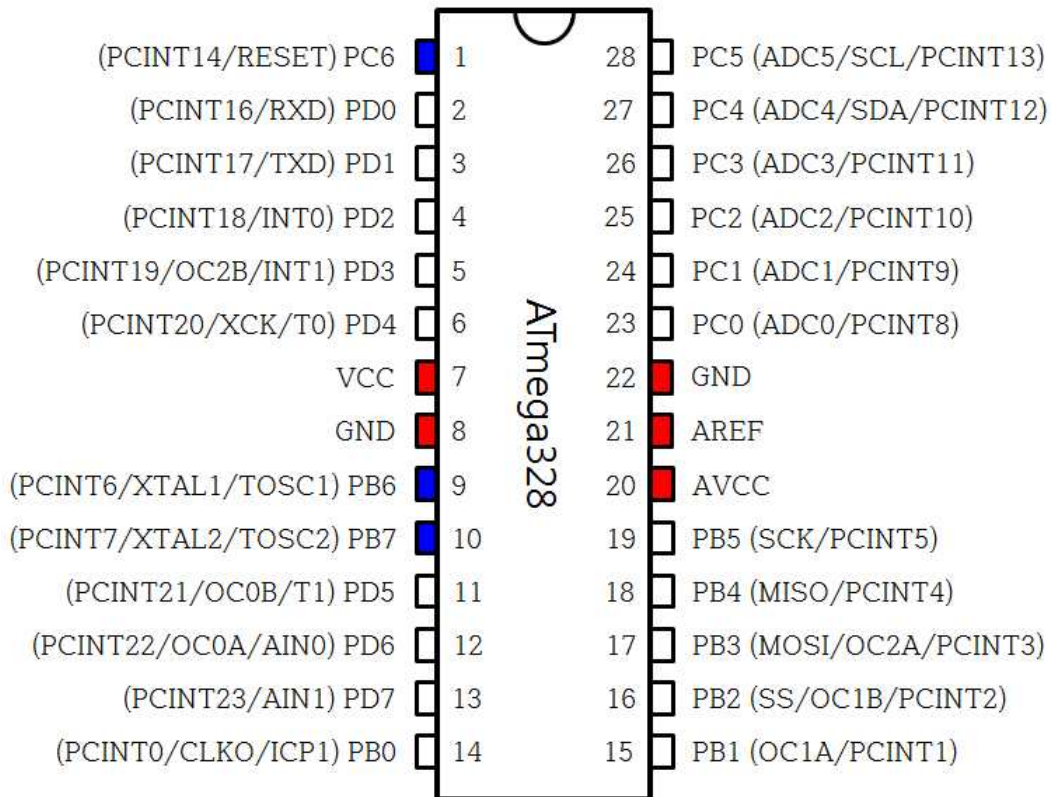


그림 3. ATmega328의 핀 배치

ATmega328는 28개의 핀을 가지고 있으며 그 중 디지털 전원 관련 핀 2개 (VCC, GND), 아날로그 전원 관련 핀 3개(AVCC, AREF, GND)를 제외한 23개의 핀은 주변장치와 데이터 교환을 위한 입출력으로 사용할 수 있다. 하지만 23개의 핀 중에서 아두이노 우노에서는 9번 (XTAL1) 및 10번(XTAL2) 핀에 16MHz의 크리스탈을 연결하여 ATmega328을 위한 클록을 공급하기 위해 사용되고, 1번 핀은 리셋(RESET) 용도로 사용한다. 따라서 아두이노 우노에서 사용할 수 있는 입출력 핀은 20개이다.

그림 3에서 볼 수 있듯이 ATmega328의 대부분 핀은 두 개 이상의 이름을 가지고 있다. 마이크로컨트롤러는 핀의 수가 제한되어 있으므로 하나의 핀이 두 개 이상의 기능을 수행하는 경우가 많으므로 필요에 따라 동일한 핀을 다른 기능으로 동작시키는 것이 가능하다. 그림 3에서 28개 핀에 붙어있는 이름을 이해하는 것이 ATmega328을 기능을 이해하는 것이라고 할

수 있다. 표 2는 각 핀의 약어들을 요약한 것이다.

| 약어     | 설명                                 | 비고                                  |
|--------|------------------------------------|-------------------------------------|
| PBn    | Port B                             | $n = 0, \dots, 7$                   |
| PCn    | Port C                             | $n = 0, \dots, 6$                   |
| PDn    | Port D                             | $n = 0, \dots, 7$                   |
| PCINTn | Pin change interrupt               | $n = 0, \dots, 14, 16, \dots, 23$   |
| INTn   | External interrupt                 | $n = 0, 1$                          |
| ADCn   | A/D Converter                      | $n = 0, \dots, 5$                   |
| OCnA   | Output compare match output        | $n = 0, 1, 2$ (n은 타이머/카운터 번호)       |
| OCnB   | Output compare match output        |                                     |
| AIN0   | Analog comparator positive input   |                                     |
| AIN1   | Analog comparator negative input   |                                     |
| XTALn  | Crystal                            | $n = 1, 2$ (시스템 클록을 위한 외부 오실레이터 연결) |
| TOSCn  | Timer oscillator                   | $n = 1, 2$ (타이머/카운터2를 위한 오실레이터 연결)  |
| XCK    | USART external clock input/output  |                                     |
| Tn     | Timer/counter source               | $n = 0, 1$ (타이머/카운터 0과 1의 외부 소스 연결) |
| CLKO   | Clock output                       | 시스템 클록 출력                           |
| ICP1   | Timer/Counter1 input capture input |                                     |
| RXD    | Receive data                       | USART 직렬 통신                         |
| TXD    | Transmit data                      |                                     |
| SCL    | 2-wire serial clock                | 2-wire 또는 I2C 직렬 통신                 |
| SDA    | 2-wire serial data                 |                                     |
| SCK    | SPI master clock                   | SPI 통신                              |
| MISO   | SPI master input slave output      |                                     |
| MOSI   | SPI master output slave input      |                                     |
| SS     | SPI master slave select            |                                     |

표 2. ATmega328의 핀 기능 설명

마이크로컨트롤러를 사용해 본 적이 있다면 표 2의 설명으로 대부분의 ATmega328 기능을

집작할 수 있을 것이다. 처음으로 표 2의 약어들을 접하고 있다면 모든 내용이 생소하겠지만 걱정하지 않아도 된다. 이후 장에서 표 2에 나타난 대부분의 내용들을 다루어 볼 것이다.

한 가지 유념해야 할 점은 그림 3의 핀 이름에서 괄호 안에 들어있지 않은 이름이 전원 관련 핀을 제외하고는 모두 PBn, PCn, PDn이라는 점이다. 'P'는 'Port'의 약자로 데이터가 입출력 되는 통로를 의미한다. 전원 관련 핀 이외의 모든 핀이 PBn, PCn, PDn 중 하나를 가진다는 것은 주변 장치와의 데이터 교환이 마이크로컨트롤러의 기본적인 기능 중 하나라는 의미로 해석할 수 있다. 괄호 안에 들어 있는 이름들도 대부분 데이터 입출력을 설명하는 이름들이지만 특정 목적을 위해서만 사용되는 이름인 반면 PBn, PCn, PDn는 범용의 데이터 입출력을 의미하므로 핀의 기능을 대표하는 이름으로 표시되어 있다.

'B, C, D'는 포트의 이름을 나타낸다. ATmega328 마이크로컨트롤러는 8비트 CPU를 가지고 있다. 즉, 한 번에 처리할 수 있는 데이터의 크기는 8비트이며 데이터는 8비트의 정수배 단위로 처리된다. 마이크로컨트롤러는 비트 단위로 핀을 통해 주변장치와 데이터를 교환하지만 실제로 CPU 내부에서 비트 단위로 데이터를 처리할 수는 없으며 8비트 단위로 데이터가 처리되고 입출력 핀으로 전달된다. 따라서 ATmega328은 핀을 8개씩 묶어서 각각 'Port B', 'Port C', 'Port D'로 이름을 붙여 관리한다. 하지만 입출력으로 사용할 수 있는 핀의 개수인 23은 8의 정수배가 아니다. 따라서 포트 C에는 7개의 핀만이 할당되어 있다. 다른 두 포트가 0에서 7사이의 'n' 값을 가지는 반면 포트 C는 0에서 6 사이의 값을 가진다. 마지막 'n'은 포트에서의 비트 번호를 나타내며  $n = 0$ 은 LSB(Least Significant Bit)를,  $n = 7$ 은 MSB(Most Significant Bit)를 의미한다.

#### 4. 레지스터

마이크로컨트롤러 프로그래밍을 하면서 가장 어려운 점은 레지스터(register)의 이름을 기억하는 일이 아닐까 싶다. 레지스터란 CPU 내부에 있는 임시 저장 공간을 가리키는 말로 실행할 명령어의 피연산자나 계산 결과를 임시로 저장하기 위해 사용한다. 일반적인 C 언어 프로그래밍에서 레지스터를 알고 있어야 하는 일은 거의 없다. 마이크로컨트롤러 프로그래밍에서도 현재 실행하는 명령어가 무엇인지, 계산 결과가 잠시 저장되었다가 사라지는 레지스터가 무엇인지 관심을 가질 필요는 전혀 없다. 하지만 마이크로컨트롤러 프로그래밍에서는 C 언어 프로그래밍에서와 다르게 외부의 주변 장치와 데이터를 교환하는 것이 주된 작업 중 하나이며 이를 위해서 데이터를 교환하기 위해 사용하는 레지스터는 알고 있어야 한다. 피연산자나 계산 결과를 임시로 저장하는 레지스터를 범용 레지스터라고 하는 반면 주변 장치와 데이터 교환을 위해 사용되는 레지스터를 전용 레지스터로 구별하여 부르며 마이크로컨트롤러 프로그래밍에

서 중요한 레지스터는 전용 레지스터들이다.

C 언어에서 키보드로부터 키 값을 읽어 들이는 `getchar` 함수를 생각해 보자.

#### 코드 1. 문자 읽기

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    char ch;

    ch = getchar();
    printf("%c\n", ch);

    return 0;
}
```

`getchar` 함수는 키보드로부터 문자를 읽어 반환하는 함수로 코드 1에서 반환된 값은 문자형 변수인 `ch`에 저장한다. 실제로 `getchar` 함수는 입력 장치인 키보드와 데이터를 교환하는 ‘통신’을 통해 눌러진 키 값을 전달받지만 실제 통신 과정은 Windows에서 알아서 처리해 주므로 C 프로그래밍에서 신경 쓸 필요는 없다. 하지만 마이크로컨트롤러에는 Windows와 같은 친절한 운영체제가 존재하지 않는다. 키가 눌러졌다면 이를 읽어서 전달해주는 역할까지도 프로그램에서 처리해 주어야 하며 이를 위해서 필요한 것이 바로 전용 레지스터들이다. 특히 데이터 교환을 위한 입출력에 사용되는 레지스터를 ‘입출력 레지스터 (I/O register)’라고 한다. 마이크로컨트롤러에 친절한 운영체제가 존재하지는 않지만 그렇다고 그렇게 불친절하지만은 않다. 포트 B에 키보드를 연결하고 눌러진 키의 8비트 값이 포트 B로 전달된다고 가정해 보자. 전달된 8비트의 값은 포트 B로 들어오는 데이터가 저장되는 전용 입출력 레지스터인 ‘PINB’에 저장되고 이 값은 C 언어 프로그래밍에서와 마찬가지로 변수에 대입하여 사용할 수 있다.

```
keyValue = PINB;
```

유사하게 포트 D에 한 문자를 표시할 수 있는 LCD가 연결되어 있다고 가정해 보자. 포트 D로 출력할 문자의 8비트 값을 전달하면 전달된 값은 LCD로 전달되고 화면에 글자가 표시된다. 이 때 코드에서는 핀으로 직접 데이터를 출력할 수 없고 데이터 출력을 위한 전용 입출력 레지스터인 'PORTD'로 데이터를 출력한다.

```
PORTD = characterValue;
```

이처럼 입출력 핀 또는 포트를 통한 데이터 입출력은 항상 전용의 레지스터를 거쳐서 이루어지며 전용 레지스터는 C 프로그래밍에서의 변수와 동일하게 데이터를 읽거나 쓸 수 있다. 입출력 레지스터는 (프로그램 코드인) 소프트웨어와 (실제 출력이 이루어지는 마이크로컨트롤러의 핀인) 하드웨어 사이에서 중계자 역할을 수행한다고 볼 수 있다. 레지스터의 이름은 'io.h' 파일에 정의되어 있다. 하지만 'io.h' 파일을 열어 보면 실제 레지스터의 정의는 찾아볼 수 없으며 마이크로컨트롤러의 종류에 따라 포함시켜야 하는 헤더 파일들이 나열되어 있을 뿐이다. ATmega328P-PU의 경우 'iom328p.h' 파일을 포함하도록 정의되어 있으며 ATmega328P-PU의 실제 레지스터 이름들은 'iom328p.h' 파일에 정의되어 있다.

```
#elif defined (__AVR_ATmega328P__)  
# include <avr/iom328p.h>
```

'iom328p.h' 파일을 열어보자. 수많은 레지스터 정의가 보이는가? 앞에서 이야기 했던 'PORTB' 레지스터 정의를 살펴보자.

```
#define PORTB _SFR_IO8(0x05)
#define PORTB0 0
#define PORTB1 1
#define PORTB2 2
#define PORTB3 3
#define PORTB4 4
#define PORTB5 5
#define PORTB6 6
#define PORTB7 7
```

‘PORTB’라는 레지스터의 이름이 정의되어 있을 뿐만이 아니라 PORTB 레지스터의 각 비트에도 이름이 정해져 있어 필요에 따라서는 정의된 이름을 사용하여 쉽게 사용할 수 있도록 하고 있다.<sup>5)</sup>

기억해야 할 것은 마이크로컨트롤러가 주변장치와 정보를 교환하기 위해 마이크로컨트롤러의 핀을 사용한다는 점, 핀을 통해 실제로 정보를 교환하기 위해서는 목적에 맞는 레지스터를 사용한다는 점, 그리고 레지스터의 이름은 헤더 파일에 정의되어 있으며 정의된 이름들은 C 언어에서의 일반 변수와 동일하게 읽고 쓸 수 있다는 점은 반드시 기억해야 한다.

한 가지 더 추가하자면 마이크로컨트롤러의 핀 하나에는 연관된 레지스터가 두 개 이상이 있을 수 있다는 점이다. ATmega328의 핀은 두 가지 이상의 목적으로 사용되는 경우가 대부분인 것은 이미 그림 3을 통해 살펴보았다. ATmega328의 18번 핀의 이름은 ‘PB4’로 포트 B의 다섯 번째 비트에 해당하므로 데이터 출력을 위해 ‘PORTB’ 레지스터와 연관되어 있다. 이외에 18번 핀의 부가 기능 중 하나가 ‘MISO’이다. MISO는 Master Input Slave Output의 약자로 SPI (Serial Peripheral Interface) 방식의 직렬 통신을 위해 사용되며, SPI와 관련된 레지스터 중 하나가 SPCR (SPI Control Register)이다. 즉, SPI 통신을 위해 18번 핀을 사용하는 경우에는 PORTB 레지스터뿐만이 아니라 SPCR 레지스터의 영향을 받게 된다. 그렇다면 과연 ATmega328에는 몇 개의 레지스터가 존재할까? ATmega328에서 정의할 수 있는 레지스터의 개수는 256개이고 이 중 실제로 정의되어 사용되는 레지스터는 119개이고 나머지 137개는 추후 사용을 위해 남겨져(reserved) 있다. 그렇다면 이들 119개의 레지스터를 다 기억해야 한다는 말인가? 이들 모두를 기억하는 것은 거의 불가능하며 필요할 때마다 데이터시

5) 마이크로컨트롤러에서 레지스터는 비트 별로 그 의미가 다르고 사용 방법이 다른 경우를 흔히 볼 수 있다. 따라서 비트 단위의 조작이 필요한 경우가 많으므로 비트별로 이름을 정의해 놓으면 비트의 실제 위치를 기억하는 것에 비해 기억하고 사용하기 쉬운 장점이 있다. 비트 조작은 다른 장에서 자세히 다룬다.

트를 찾아보는 방법밖에는 없다. 그러니 항상 ATmega328의 데이터시트를 곁에 두기를 추천한다.

처음으로 돌아가 보자. 마이크로컨트롤러 프로그래밍을 하면서 가장 어려운 점은 레지스터의 이름을 기억하는 일이다. 마이크로컨트롤러를 처음 접하는 사람들은 혼란에 빠진다. 몇 개의 알파벳으로 축약되어 있는 레지스터 이름에서 그 의미를 짐작해낼 수 있기까지는 상당한 시간이 걸린다. 레지스터 이름을 기억하지 않고 마이크로컨트롤러를 위한 프로그램을 작성할 수는 없을까? 있다, 바로 아두이노가 그것이다. 아두이노는 비전공자들이 쉽게 마이크로컨트롤러를 사용할 수 있도록 복잡한 레지스터 조작을 추상화된 함수의 형태로 제공하고 있어 마이크로컨트롤러 프로그램을 시작하기에 좋은 출발점이 되는 것이 사실이다. 또한 다양한 주변장치를 사용하기 위한 라이브러리가 준비되어 있어 어렵지 않게 복잡한 제어 장치를 구성하는 것도 가능하다. 그렇다면 굳이 번거롭고 기억하기 어려운 레지스터를 이야기하는 이유는 무엇일까? 아쉽지만 아두이노가 만능은 아니다. 지금도 누군가 아두이노를 보다 쉽게 사용할 수 있도록 애쓰고 있을 것은 분명하며, 그러기 위해 데이터시트에서 레지스터를 찾아보고 있으리라 것도 어렵지 않게 짐작할 수 있다. 그만큼 레지스터는 마이크로프로세서 프로그래밍에서 빼놓을 수 없는 부분이다.

## 5. ATmega328의 메모리

마이크로컨트롤러는 컴퓨터의 메인 보드의 기능과 ‘일부’ 하드 디스크의 기능을 하나의 칩으로 구현한 것이다. 여기서 ‘일부’ 하드 디스크의 기능이라고 이야기 한 이유는 하드 디스크와 유사하면서도 다른 점이 있기 때문이다. 먼저 컴퓨터에서 프로그램이 실행되는 순서를 생각해 보자. ATmega328을 위한 프로그램을 작성하기 위해서는 어떤 과정을 거치는가? 먼저 Atmel Studio 6 프로그램을 설치하여야 한다. 프로그램은 하드 디스크에 설치된다. 설치가 끝나면 아이콘을 더블클릭 하여 프로그램을 실행시킨다. 프로그램이 실행되면 하드 디스크에 설치된 프로그램은 메인 메모리로 읽혀지고 메인 메모리로 이동한 프로그램은 다시 CPU로 이동하여 화면에 멋진 사용자 인터페이스를 보여주면서 코드 입력을 기다리고 있을 것이다. 하지만 마이크로컨트롤러에서 프로그램이 실행되는 과정은 컴퓨터와 기본적으로 동일하면서도 세부 사항에서 약간의 차이가 있다. 프로그램을 실행하기 이전에 프로그램을 설치하여야 하는 것은 동일하다. 하지만 마이크로컨트롤러에는 하드 디스크가 없다. 그렇다면 어디에 프로그램이 설치될까? 바로 메모리에 설치된다. 컴퓨터에서 메모리라고 하면 메인 메모리를 가리키며 전원이 꺼지면 그 내용이 지워지는 휘발성 메모리를 가리킨다. 반면 마이크로컨트롤러에서 프로그램이 설치되는 메모리는 전원이 꺼져도 내용이 지워지지 않는 비휘발성 메모리이다. 컴퓨터에

서와 같이 휘발성 메모리를 사용한다면 전원을 켜 때마다 프로그램을 다시 설치해야 할 것이다. 표 1을 유심히 살펴보면 눈치 챌겠지만 ATmega328에는 세 종류의 메모리가 존재한다. 그 중 프로그램이 설치되는 메모리가 32KByte 크기의 플래시 메모리 또는 프로그램 메모리에 해당한다.

프로그램이 설치된 후 실행을 위해서는 프로그램을 메인 메모리로 읽어 와야 하지만 마이크로컨트롤러의 경우 이미 프로그램이 메모리에 위치해 있으므로 바로 실행이 가능하다. 프로그램이 실행되는 도중 값이 변하는 변수들의 값 역시 메인 메모리에 저장된다. 예를 들어, 코드 1에서 변수 ch의 값은 프로그램이 실행된 후 키가 눌릴 때까지 알 수 없으며 키가 눌린 후에야 그 값이 메인 메모리에 저장된다. 컴퓨터의 경우 하드 디스크에서 읽혀진 프로그램과 변수값은 동일한 메인 메모리에 저장된다. 하지만 마이크로컨트롤러의 경우 변수 값이 저장되는 메모리는 프로그램이 저장되는 메모리와는 다른 SRAM(Static Random Access Memory)에 저장된다. 컴퓨터에서와 같이 프로그램과 (프로그램 실행 도중 값이 변하는 변수로 대표되는) 데이터가 동일한 메모리에 저장되는 구조를 폰 노이만 구조라고 하고, 마이크로컨트롤러에서와 같이 프로그램과 데이터가 서로 다른 메모리에 저장되는 구조를 하버드 구조라고 한다. SRAM은 컴퓨터의 메인 메모리와 동일하게 휘발성 메모리이므로 전원이 꺼지면 그 내용은 사라진다. 하지만 컴퓨터의 메인 메모리인 DRAM (Dynamic RAM)에 비해 SRAM은 훨씬 빠른 속도로 읽고 쓸 수 있다.

컴퓨터에서 프로그램을 실행시키면서 실행 결과를 하드 디스크에 기록하여 다음 번 실행 시에 이전 실행 결과를 참고하는 경우가 종종 있다. 워드프로세서를 사용하면서 이전 문서 목적이 자동으로 메뉴에 추가되어 있는 것을 본 기억이 있는가? 하드 디스크의 어딘가에 이전에 편집한 문서의 목적이 저장되어 있으며 프로그램에서는 프로그램 자체와 별도로 문서 목록을 읽어서 사용하고 있다. 마이크로컨트롤러에서 이와 유사하게 현재 상태를 기록한다고 생각해 보자. 어디에다 기록해야 할까? SRAM은 휘발성이므로 데이터를 기록해 놓을 수 없다. 그렇다면 플래시 메모리에 기록하면 어떨까? 플래시 메모리는 ROM(Read Only Memory)에 가까워 프로그램이 실행 중인 동안 정보를 기록할 수 없다. 플래시 메모리에 정보를 기록하는 것은 프로그램을 다운로드 하여 설치할 때만 가능하다. 따라서 제 3의 메모리인 EEPROM(Electrically Erasable Programmable Read Only Memory)이 필요하다. EEPROM은 플래시 메모리와 같이 비휘발성이면서도 읽고 쓰기가 자유로운 메모리로 프로그램이 실행 중인 동안에도 자유롭게 읽고 쓸 수 있어 보존이 필요한 정보를 기록해 두기에 적합하다. 한 가지 의문이 생기지 않는가? 그렇다면 플래시 메모리를 EEPROM으로 대체하면 되지 않을까? EEPROM이 기능적인 면에서 흠잡을 데가 없긴 하지만 속도가 느린 단점이 있어 프로그램을 설치하고 읽어 오는 용도로는 적합하지 않다. 따라서 EEPROM은 세 종류의 메

모리 중 가장 작은 크기인 1KByte 용량을 가지며 반드시 기록해서 보존해야만 하는 정보에 한해 사용하기를 추천한다.

|                    | 플래시 메모리    | SRAM   | EEPROM      |
|--------------------|------------|--------|-------------|
| 크기 (KByte)         | 32         | 2      | 1           |
| 주 용도               | 프로그램 저장    | 데이터 저장 | 데이터 저장      |
| 비휘발성               | ○          | ×      | ○           |
| 프로그램 실행 중<br>변경 가능 | 불가능        | 가능     | 가능          |
| 속도                 | 중간         | 가장 빠름  | 가장 느림       |
| 수명                 | 10,000회 쓰기 | 반영구적   | 100,000회 쓰기 |

표 3. ATmega328의 메모리

세 종류의 메모리는 각기 별도의 메모리 주소를 가지고 있다. 그 중 가장 간단한 구조를 가지는 메모리는 EEPROM으로 바이트 단위로 메모리 주소가 정해지며 입출력 역시 바이트 단위로 이루어진다.

플래시 메모리는 32KByte 크기를 가진다. ATmega328이 8비트 CPU를 사용하고 있지만 모든 AVR 명령어는 16비트 또는 32비트 크기를 가지고 있다. ATmega328에서 사용할 수 있는 기계어 명령어는 130여개로 8비트로 표현할 수 있지만 피연산자(operand)를 포함하게 되면 8비트로 표현할 수 없으므로 2바이트 또는 4바이트 명령어 길이를 가지고 있다. 따라서 플래시 메모리의 주소는 다른 메모리들과 다르게 2바이트 단위로 정해져 있으며 16K개의 주소 ( $32\text{KByte} \div 2\text{Byte}$ )를 나타내기 위해 14비트의 주소( $16K = 16 \times 2^{10} = 2^4 \times 2^{10}$ )를 사용하고 있다.

플래시 메모리는 두 개의 영역으로 나뉘어져 있다. 플래시 메모리는 프로그램을 저장하기 위해 사용되는 공간으로 사용자가 작성한 프로그램을 저장하기 위해 주로 사용되지만 부트로더(bootloader)라고 불리는 특별한 프로그램을 위해 일부 공간을 사용할 수 있다. 부트로더 역시 사용자가 작성한 프로그램인 점은 동일하지만 'boot + loader'라는 이름에서 알 수 있듯이 마이크로컨트롤러가 '부팅'되는 시점에서 자동으로 프로그램을 '다운로드'하여 설치할 수 있도록 하는 용도로 흔히 사용되는 프로그램으로 사용자가 작성한 프로그램의 자동 업데이트 용도로 많이 사용된다. 아두이노에서는 프로그램을 다운로드하기 위해 부트로더를 사용하고 있으며 이는 다른 마이크로컨트롤러에서는 찾아보기 어려운 특징 중 하나이다. 그림 4는 ATmega328의 플래시 메모리 구조를 나타내고 있다. 부트로더의 크기는 설정에 따라 최소

0.5Kbyte에서 최대 4Kbyte의 크기를 가질 수 있으며 프로그램 메모리의 가장 뒤쪽에 위치한다.

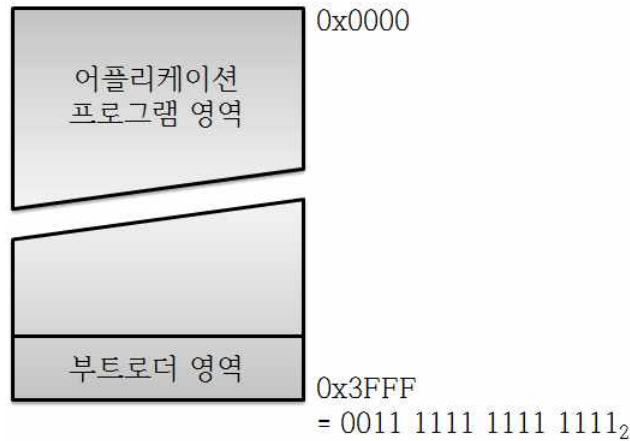


그림 4. ATmega328의 프로그램 메모리 구조

SRAM은 세 종류의 메모리 중 가장 복잡한 구조를 가지고 있다. 사실 그 구조가 복잡하지는 않지만 SRAM은 레지스터와 메모리 주소를 함께 사용하고 있어 이해하기 까다로울 뿐이다. 앞서도 설명하였지만 마이크로컨트롤러 프로그래밍에서 가장 귀찮은 부분은 레지스터이다. ATmega328에는 256개의 레지스터를 정의할 수 있으며 이는 다시 세 부분으로 나뉜다. 컴퓨터에서 레지스터라고 하면 연산을 위해 필요한 피연산자와 연산의 결과를 임시로 저장하기 위해 사용되는 CPU 내의 기억 장치를 가리키는 경우가 많으며 이를 범용 레지스터라고 한다. ATmega328에도 32개의 범용 레지스터가 준비되어 있으며 이를 제외한 나머지 224개 레지스터는 입출력 레지스터에 해당한다. 224개 입출력 레지스터는 다시 64개의 기본 입출력 레지스터와 160개의 확장 입출력 레지스터로 나뉘며, 일반적으로 입출력 레지스터는 기본 입출력 레지스터를 가리킨다.<sup>6)</sup>

6) 이 책에서는 특별히 구별하여야 할 필요가 없는 경우에는 기본 입출력 레지스터와 확장 입출력 레지스터 모두를 입출력 레지스터라고 지칭한다.

|             | 사용 가능한<br>개수 | 실제 정의되어<br>사용하는 개수 | 추후 사용을 위해<br>예약된 개수 |
|-------------|--------------|--------------------|---------------------|
| 범용 레지스터     | 32           | 32                 | 0                   |
| 입출력 레지스터    | 64           | 39                 | 25                  |
| 확장 입출력 레지스터 | 160          | 48                 | 112                 |
| 합계          | 256          | 119                | 137                 |

표 4. 레지스터 사용 현황

SRAM은 레지스터와 메모리 주소를 함께 사용하고 있다고 이야기 하였다. 그림 5를 보면 이해가 빠를 것이다. 먼저 범용 레지스터 32개가 0번지부터 할당되고, 다음으로 기본 입출력 레지스터와 확장 입출력 레지스터가 할당되어 있으며, 그 뒤에 SRAM의 주소가 할당되어 있다. AVR에서는 레지스터와 SRAM으로 이루어지는 메모리를 데이터 메모리라고 부른다. EEPROM 역시 데이터 저장을 위해 사용되는 메모리이지만 별도의 주소를 가지고 있으므로 주소를 이야기하는 경우 데이터 메모리에 EEPROM을 포함시키지는 않는 것이 일반적이다.

|                        |                 |
|------------------------|-----------------|
| 32개 범용 레지스터            | 0x0000 - 0x001F |
| 64개<br>입출력 레지스터        | 0x0020 - 0x005F |
| 160개<br>확장 입출력<br>레지스터 | 0x0060 - 0x00FF |
| 2K개<br>SRAM            | 0x0100 - 0x08FF |

그림 5. ATmega328의 데이터 메모리 구조

ATmega328에는 세 종류의 메모리가 포함되어 있으며 세 종류의 주소를 사용하고 있다. 엄청나게 복잡한 것으로 보일 수도 있지만 사실 그리 걱정할 필요는 없다. 컴퓨터에서 프로그램을 작성하면서 프로그램이 적재되는 메모리 주소나 현재 계산 결과를 저장할 변수가 위치하는 메모리 주소를 걱정해 본 적이 있는가? 대부분은 생각조차 해 본 적이 없을 것이고 생각해 볼 필요가 없는 것이 사실이다. 이는 마이크로컨트롤러에서도 마찬가지다. 프로그램이 저장되

는 플래시 메모리와 변수의 값이 저장되는 SRAM의 주소를 걱정할 필요는 없다. 데이터 메모리의 일부를 차지하는 레지스터는 어떤가? 모든 레지스터에는 이름이 있으므로 레지스터 사용을 위해 번지를 사용할 필요는 없다. 하지만 EEPROM은 다르다. EEPROM은 유일하게 번지만 접근이 가능한 메모리이므로 데이터를 읽거나 쓰기 위해서는 번지를 사용하여야 한다. 앞에서 'PORTB' 레지스터의 정의를 살펴본 것이 기억나는가? PORTB의 정의는 다음과 같다.

```
#define PORTB _SFR_IO8(0x05)
```

앞에서 언급하지 않고 슬쩍 지나친 부분이 바로 '\_SFR\_IO8'이라는 매크로이다. \_SFR\_IO8은 입출력 주소를 데이터 메모리 주소로 변환하는 기능을 한다. '입출력 주소'는 앞에서 설명한 세 가지 주소와는 다른 또 다른 주소로 기본 입출력 레지스터로만 구성되는 공간에 할당되는 주소를 말한다. 그림 5에서 기본 입출력 레지스터는 데이터 메모리의 0x20부터 0x5F까지의 주소에 할당되어 있음을 확인할 수 있다. 입출력 주소는 데이터 메모리 주소에서 0x20을 뺀 값, 즉, 첫 번째 기본 입출력 레지스터의 주소가 영(zero)이 되도록 만들어 놓은 주소이다. 레지스터 목록을 살펴보면 기본 입출력 레지스터에는 두 개의 번지가 할당되어 있음을 확인할 수 있다. 이 중 하나는 입출력 주소이고 다른 하나는 데이터 메모리 주소에 해당한다. 반면 확장 입출력 레지스터는 데이터 메모리 주소로만 정의된다.

메모리 주소에 대하여 명확하게 이해할 수 없다면 처음에는 ATmega328은 세 개의 메모리를 가지고 있다는 것과 세 개 메모리의 주된 기능을 이해하는 것으로 충분하다. 메모리 주소에 대해 자세하게 언급한 이유는 데이터시트를 참고할 경우 데이터시트에서 설명하는 방식을 이해할 수 있도록 하기 위한 목적이 크다. ATmega328이나 다른 아트멜 마이크로컨트롤러의 데이터시트를 한번쯤 살펴본 독자라면 공감할 수도 있으리라 생각한다. 아트멜의 데이터시트는 다른 데이터시트에 비해 그리 친절하지 않아 이해하기가 쉽지 않다. 처음 데이터시트를 접하게 되면 무슨 이야기인지 도저히 이해할 수 없더라도 실망하지는 말길 바란다. 필자도 그랬고 사실 지금도 그러하다. 하지만 결국 가장 정확한 정보를 얻을 수 있는 곳은 데이터시트임도 명심하기 바란다.