
Chapter 108. 아날로그-디지털 변환

1. ATmega328의 ADC

마이크로컨트롤러는 주변 환경으로부터 데이터를 획득하고 이를 처리하여 시스템을 제어하는 목적으로 사용된다. 이 때 주의할 점은 주변 환경으로부터 획득할 수 있는 데이터는 아날로그 데이터라는 점이다. 지금까지 살펴본 데이터들은 디지털 데이터였다. LED에 불을 켜거나 끄는 데이터, 스위치가 눌러졌거나 눌러지지 않은 데이터는 물론이거니와 UART 통신을 통해 컴퓨터로 전송되는 데이터 역시 HIGH나 LOW의 디지털 데이터 열로 전달되었다. 마이크로컨트롤러는 이진인 디지털 데이터만을 처리할 수 있다. 하지만 온도, 습도, 조도 등 주변 환경에서 측정할 수 있는 데이터들은 연속적인 아날로그 데이터로 이들 데이터를 처리하기 위해서는 먼저 디지털 데이터로 변환해주어야 하며, 아날로그 데이터를 디지털 데이터로 변환하는 장치를 아날로그-디지털 변환기(Analog-Digital Converter, ADC)라고 한다.

주변 환경으로부터 데이터를 획득하는 장치를 흔히 센서라고 이야기한다. 전압을 측정하는 회로는 간단하게 구현할 수 있으므로 대부분의 센서는 아날로그 데이터인 전압을 출력으로 낸다. 온도가 높은 경우에는 높은 전압을, 온도가 낮은 경우에는 낮은 온도를 출력으로 내는 식이다. 마이크로컨트롤러는 아날로그 전압을 입력으로 받아 이를 ADC를 거쳐 디지털 데이터로 변환한 이후에 CPU에서 처리한다. ATmega328 내부의 아날로그 데이터 입력 부분에 해당하는 블록 다이어그램은 그림 1과 같다.

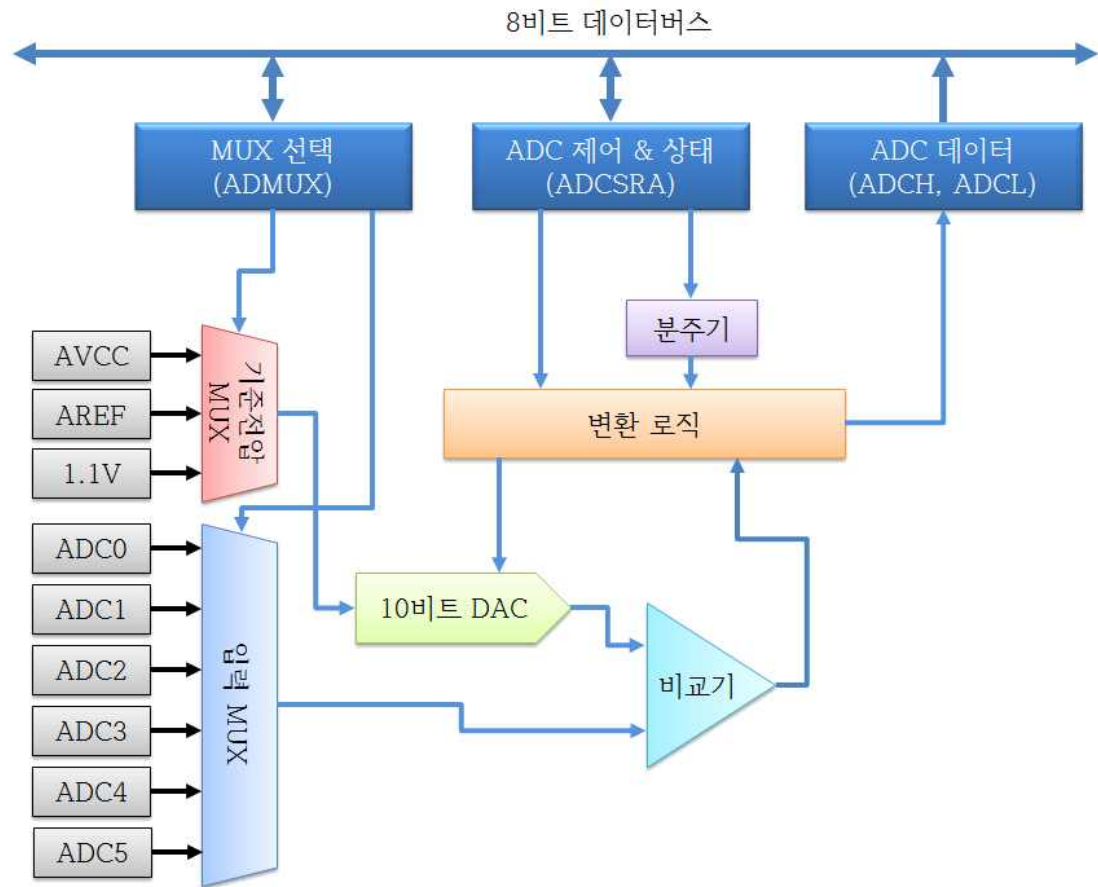


그림 1. 아날로그-디지털 변환 블록 다이어그램

ATmega328에는 10비트의 ADC가 포함되어 있다. 즉, 입력되는 아날로그 전압을 0에서 $1023 (= 2^{10} - 1)$ 사이의 디지털 값으로 변환할 수 있다. ADC 채널은 6개로 6개의 아날로그 출력 장치를 ATmega328에 연결할 수 있지만 6개의 채널은 하나의 AD 변환기에 멀티플렉서 (Multiplexer, MUX)로 연결되어 있으므로 한 번에 하나의 아날로그 입력만을 디지털로 변환할 수 있다. 6개의 채널은 포트 C의 PC0에서 PC5까지 연결되어 있으며, 아두이노에서는 핀 번호 14번에서 19번까지가 이에 해당한다.

센서가 아날로그 전압을 출력하고 이를 ADC가 0에서 1023 사이의 디지털 값으로 변환하는 과정에서 중요한 점은 몇 볼트의 전압을 1023의 값으로 변환할 것인가 하는 문제다. 이를 위해 기준 전압이 필요하며 기준전압은 AD 변환에서 1023의 디지털 값으로 변환될 값을 의미한다. 기준전압은 AVCC, AREF, 내부 1.1V의 3개 중 하나를 선택하여 사용할 수 있다. 아두이노 우노는 5V에서 동작하므로 AVCC의 경우 5V 값을 가지는 것이 일반적이며, AREF의 경우 5V가 아닌 다른 기준 전압을 사용하기 위해 별도의 외부 전원을 연결하여 사용한다. 실제 AD 변환은 선택된 입력 채널의 값과 기준 전압을 비교해 나가면서 이루어지며 이를 위해 필

요한 부분들이 DAC(Digital Analog Converter), 비교기 그리고 변환 로직에 해당한다. ATmega328은 축차비교형(successive approximation) ADC를 가지고 있다. 축차비교형 ADC는 MSB부터 LSB 순서로 각 비트를 1로 설정하여 목표값 v_{in} 보다 큰 경우 0으로, 목표값 v_{in} 보다 작은 경우 1로 설정함으로써 연속적으로 목표값 v_{in} 과 유사한 v_{out} 값을 구하는 방식이다.

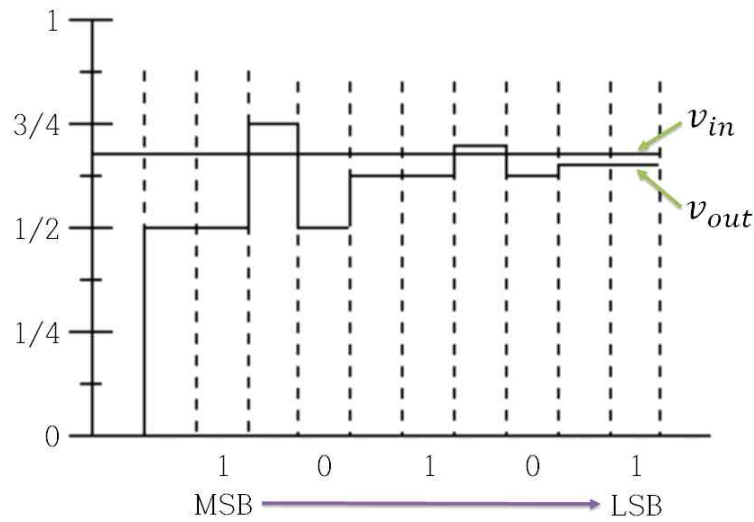


그림 2. 축차비교형 아날로그 디지털 변환

이처럼 AD 변환 과정은 시간이 많이 걸리는 작업으로 메인 클록, 즉, 아두이노 우노에서 사용하는 16MHz의 속도로 동작할 수는 없다. 따라서 메인 클록을 필요한 정도의 속도로 낮추어 사용하며 이를 위해 필요한 것이 분주기이다.

2. 가변저항 값 읽기

아두이노 우노 보드에는 20개의 핀(0~19번 핀)을 디지털 입출력으로 사용할 수 있으며 이 중 6개(14~19번 핀)는 아날로그 입력을 위해서도 사용이 가능하다. 아두이노에서는 아날로그 입력이 가능한 핀을 디지털 입출력만 가능한 핀과 구별하기 위해 14번 핀에 A0, 15번 핀에 A1 등으로 별도의 상수를 정의하여 두고 있으므로 아날로그 입력에 사용할 수 있는 핀을 쉽게 구별할 수 있다. 아날로그 입력 테스트를 위해 그림 3과 같이 PC0 핀 (아두이노 A0 핀)에 가변저항을 연결해 보자.

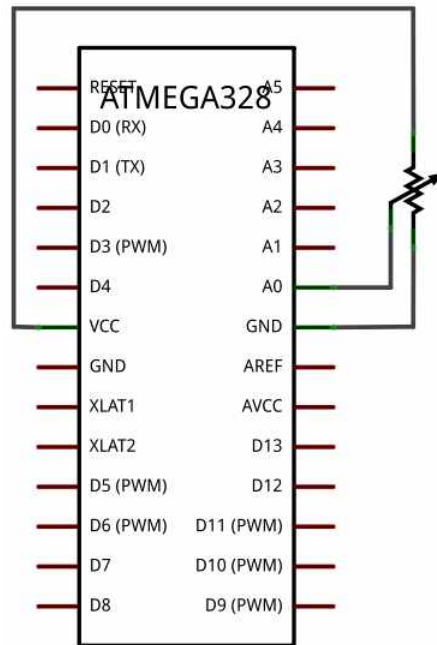


그림 3. 가변저항 연결 회로도

코드 1은 PC0 핀 (아두이노 A0 핀)에 연결된 가변저항의 값을 읽어 1초에 한 번 터미널 윈도우로 출력하는 프로그램의 예이다.

코드 1. 가변저항 읽기 - C 스타일

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

void UART_INIT(void) {
    UCSR0A |= _BV(U2X0);           // 2배속 모드

    UBRR0H = 0x00;                  // 통신 속도 (보율) 설정
    UBRR0L = 207;

    // 비동기, 8비트 데이터, 패리티 없음, 1비트 정지 비트 모드
    UCSR0C |= 0x06;
```

```
    UCSR0B |= _BV(RXEN0);           // 송수신 가능
    UCSR0B |= _BV(TXEN0);
}

void UART_transmit(unsigned char data)
{
    while( !(UCSR0A & (1<<UDRE0)) );    // 송신 가능 대기
    UDR0 = data;                        // 데이터 전송
}

void UART_transmit_string(char *str)
{
    while(*str){                       // 문자열의 끝 검사
        UART_transmit(*str);           // 문자 단위 전송
        str++;
    }

    UART_transmit('\n');               // 줄바꿈 문자 추가
}

void ADC_INIT(void)
{
    ADMUX |= 0x40;                     // AVCC를 기준 전압으로 선택

    ADCSRA |= (1 << ADPS1) | (1 << ADPS0);    // 분주비 설정
    ADCSRA |= (1 << ADEN);                  // ADC 활성화
    ADCSRA |= (1 << ADATE);                 // 자동 트리거 모드

    ADMUX |= ((ADMUX & 0xE0) | channel);    // 채널 선택
    ADCSRA |= (1 << ADSC);                  // 변환 시작
}

int read_ADC(unsigned char channel)
{
    while(!(ADCSRA & (1 << ADIF)));        // 변환 종료 대기
```

```

    return ADCW;                                // 10비트 값을 반환
}

void int_to_string(int n, char *buffer)
{
    sprintf(buffer, "%04d", n);                // ADC 값을 0으로 채워진 4자리 문자열로 변환
    buffer[4] = '\0';
}

int main(void)
{
    int read;
    char buffer[5];

    UART_INIT();                               // UART 통신 초기화
    ADC_INIT();                                // AD 변환기 초기화

    while(1)
    {
        read = read_ADC(0);                    // 가변저항 값 읽기
        int_to_string(read, buffer);           // 정수값을 문자열로 변환
        UART_transmit_string(buffer);          // 문자열을 UART로 전송

        _delay_ms(1000);                       // 1초에 한 번 읽음
    }
}

```

UART 통신을 위한 내용은 시리얼 통신 장치의 내용을 그대로 사용하였다. 다만 문자열을 전송하기 위해 char형 배열을 매개변수로 가지는 UART_transmit_string 함수를 추가하였다. int_to_string 함수는 정수값을 길이 4의 문자열로 변환하는 함수로 문자열로 형식화된 내용을 출력하는 sprintf 함수를 이용하여 구현하였다. sprintf 함수의 사용 방법은 printf 함수와 기본적으로 동일하며, 변환된 문자열이 기록될 char형 배열이 매개변수에 추가된 점만 다르다.

코드 1을 살펴보면 AD 변환을 위해 ADMUX, ADCSRA 그리고 ADCW (ADCH와 ADCL의 조

합) 레지스터가 사용된 것을 볼 수 있다. 이들 레지스터는 그림 1의 블록 다이어그램에서도 확인할 수 있다. 레지스터의 설명은 잠깐 미루고 코드 1을 업로드 하고 가변저항을 돌리면서 0에서 1023의 결과가 출력되는지 확인해 보자.

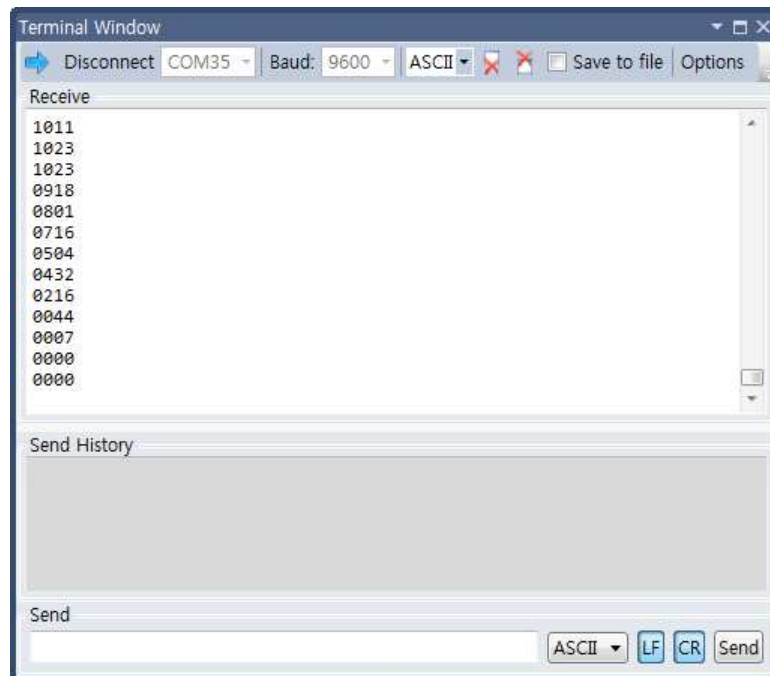


그림 4. 코드 1 실행 결과

3. AD 변환을 위한 레지스터

AD 변환을 위해 사용된 레지스터들을 살펴보자. AD 변환을 위해서는 기준 전압과 입력 채널 설정을 위한 ADMUX 레지스터, AD 과정을 제어하기 위한 ADCSRA 레지스터, 그리고 변환된 데이터를 저장하기 위한 ADCW (ADCH와 ADCL의 조합) 레지스터가 필요하다.

(1) ADCH, ADCL

ADCH와 ADCL 레지스터는 AD 변환된 데이터가 저장되는 레지스터이다. ATmega328의 ADC는 10비트의 해상도를 가지지만 레지스터는 8비트이므로 두 개의 레지스터를 사용하여 나누어 저장하고 있다. 따라서 전체 16비트 중에서 6비트는 사용되지 않으며 데이터를 정렬하는 방식에 따라 ADCH의 상위 6비트 또는 ADCL의 하위 6비트가 사용되지 않는다. 정렬 방식은 ADMUX 레지스터에 의해 결정된다. 그림 5는 ADCH 및 ADCL 레지스터의 구조를 나타

낸다.

비트	15	14	13	12	11	10	9	8
ADCH	-	-	-	-	-	-	ADC9	ADC8
ADCL	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
비트	7	6	5	4	3	2	1	0
읽기/쓰기	R	R	R	R	R	R	R	R
	R	R	R	R	R	R	R	R
초깃값	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0

(a) 오른쪽 정렬

비트	15	14	13	12	11	10	9	8
ADCH	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2
ADCL	ADC1	ADC0	-	-	-	-	-	-
비트	7	6	5	4	3	2	1	0
읽기/쓰기	R	R	R	R	R	R	R	R
	R	R	R	R	R	R	R	R
초깃값	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0

(b) 왼쪽 정렬

그림 5. ADCH 및 ADCL 레지스터 구조

두 개의 레지스터를 사용하는 것은 불편하므로 ADCH와 ADCL 레지스터로 이루어지는 16비트 크기의 ADCW를 사용할 수 있다. 오른쪽 정렬을 사용하는 경우에는 ADCW 값을 통해 10비트의 값을 직접 얻을 수 있다. 왼쪽 정렬을 사용한다면 ADCW 값을 오른쪽으로 6비트만큼 이동시켜 사용하면 된다. 디폴트로 오른쪽 정렬이 사용된다.

(2) ADMUX

ADMUX 레지스터는 AD 변환을 위한 기준 전압과 입력 채널을 선택하기 위해 사용된다. ADMUX 레지스터의 구조는 그림 6과 같다.

비트	7	6	5	4	3	2	1	0
	REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0
읽기/쓰기	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

그림 6. ADMUX 레지스터 구조

비트 7과 REFS1과 비트 6 REFS0는 AD 변환의 기준 전압을 선택하기 위해 사용된다. 비트 값에 따른 기준 전압은 표 1과 같다. 코드 1에서는 외부 AVCC로 5V를 기준 전압으로 사용하고 있다.

REFS1	REFS0	설명
0	0	외부 AREF 핀 입력을 기준 전압으로 사용한다.
0	1	외부 AVCC 핀 입력을 기준 전압으로 사용한다.
1	0	-
1	1	내부 1.1V를 기준 전압으로 사용한다.

표 1. REFS1과 REFS0 설정에 따른 기준 전압

비트 5 ADLAR(ADC Left Adjust Result)은 변환된 디지털 데이터의 저장 방법을 설정하기 위해 사용되며 ADLAR = 1이면 왼쪽 정렬, ADLAR = 0이면 오른쪽 정렬이 사용된다. 디폴트 값은 오른쪽 정렬이며 코드 1에서도 오른쪽 정렬을 사용하고 있다.

3번 비트 MUX3부터 0번 비트 MUX0까지는 채널 선택을 위해 사용된다. 아두이노 우노에서는 0번부터 (아날로그 A0 핀, PC0) 5번까지 (아날로그 A5번 핀, PC5) 6개의 채널을 사용할 수 있으므로 사용하고자 하는 번호를 설정하면 된다. 채널 선택을 위해 코드 1에서는 먼저 MUX3:MUX0의 4비트를 0으로 만든 이후 선택한 채널 번호와 OR 시키고 있다.

```
ADMUX |= ((ADMUX & 0xE0) | channel);
```

(3) ADCSRA

ADCSRA 레지스터는 ADC Control and Status Register A로 AD 변환의 상태를 나타내거나 AD 변환 과정을 제어하기 위해 사용된다. ADCSRA 레지스터의 구조는 그림 7과 같다.

비트	7	6	5	4	3	2	1	0
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
읽기/쓰기	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

그림 7. ADCSRA 레지스터 구조

7번 비트 ADEN(ADC Enable)은 ADC를 활성화시키는 비트로 디폴트값이 0이므로 ADC를 사용하기 위해서는 반드시 1로 설정하여야 한다.

AD 변환은 필요할 때에만 한 번 AD 변환을 수행하는 단일 변환 (single conversion) 모드와 연속적으로 AD 변환이 수행되는, 즉, 이전 변환이 끝나면 자동으로 다음 변환이 시작되는 프리러닝 (free running) 모드가 있다. 6번 비트 ADSC(ADC Start Conversion)는 AD 변환을 시작하도록 하는 비트로 단일 변환 모드에서는 각 변환을 시작하기 위해, 프리러닝 모드의 경우에는 첫 번째 변환을 시작하기 위해 사용된다. AD 변환이 끝나면 ADSC는 0으로 클리어된다.

5번 비트 ADATE(ADC Auto Trigger Enable)는 선택된 트리거 신호에 의해 변환이 시작되도록 설정하기 위해 사용된다. ADATE 비트가 0인 경우에는 단일 변환 모드에 해당한다. 트리거 소스의 선택은 ADCSRB 레지스터의 설정에 따른다.

4번 비트 ADIF(ADC Interrupt Flag)는 AD 변환이 종료되고 데이터 레지스터가 업데이트 되면 1로 세트된다. 3번 비트 ADIE(ADC Interrupt Enable)은 AD 변환이 끝났음을 알리는 인터럽트 발생을 허용하는 비트이다. 하지만 실제 인터럽트가 발생하기 위해서는 SREG 레지스터의 I 비트가 세트되어 있어야 한다. 즉, AD 변환이 끝나면 ADIF 비트는 자동으로 세트되고, ADIF 비트가 세트되었을 때 ADIE 비트와 SREG의 I 비트가 세트되어 있다면 AD 변환 완료 인터럽트가 발생한다.¹⁾

ADCSRA 레지스터의 나머지 3비트는 ADC 클럭을 위한 분주율을 설정하기 위해 사용된다. 분주율은 16MHz 시스템 클럭을 얼마의 값으로 나누어서 ADC 클럭으로 사용할 것인지를 나타내며 표 2는 ADPSn (n = 0, 1, 2) 값에 따른 분주율을 나타낸다.

1) 인터럽트에 관한 자세한 내용은 뒷부분에서 다룰 것이며 이 장에서 인터럽트를 사용하지는 않는다.

ADPS2	ADPS1	ADPS0	분주율
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

표 2. ADPSn (n = 0, 1, 2) 값에 따른 분주율

표 3은 ADCSRA 레지스터의 의미를 요약한 것이다.

비트	이름	설명
7	ADEN	ADC를 활성화시킨다. (ADC Enable)
6	ADSC	AD 변환을 시작한다. (ADC Start Conversion)
5	ADATE	선택된 트리거 신호에 의해 변환이 시작되도록 설정한다. 트리거 신호 선택은 ADCSRB 레지스터 설정에 따른다. (ADC Auto Trigger Enable)
4	ADIF	AD 변환이 종료되었음을 알리는 플래그이다. (ADC Interrupt Flag)
3	ADIE	AD 변환이 종료되었을 때 인터럽트 발생을 허용한다. (ADC Interrupt Enable)
2	ADPS2	ADC를 위한 분주율을 설정한다. (ADC Prescaler Select)
1	ADPS1	
0	ADPS0	

표 3. ADCSRA 레지스터 비트

(4) ADCSRB

코드 1에서는 사용하지 않았지만 AD 변환을 위해서는 ADCSRB 레지스터 역시 사용된다. ADCSRA 레지스터의 ADATE 비트 설정에 따라 트리거 소스가 선택되며 트리거 소스 선택은 ADCSRB 레지스터에 의해 설정된다고 설명한 것을 기억하는가? ADCSRB 레지스터의 구조는 그림 8과 같다.

비트	7	6	5	4	3	2	1	0
	-	ACME	-	-	-	ADTS2	ADTS1	ADTS0
읽기/쓰기	R	R/W	R	R	R	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

그림 8. ADCSRB 레지스터구조

6번 비트 ACME(Analog Comparator Multiplexer Enable)는 이름에서 알 수 있듯이 비교기를 위해 사용되는 비트로 여기서는 자세히 설명하지 않는다. 하위 3비트인 ADTSn ($n = 0, 1, 2$) (ADC Auto Trigger Source)는 AD 변환을 시작하는 트리거 신호를 선택할 수 있도록 해 주며 설정 값에 따른 소스는 표 4에 나타내었다.

ADTS2	ADTS1	ADTS0	트리거 소스
0	0	0	프리러닝 모드
0	0	1	아날로그 비교기
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match A
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

표 4. ADTSn ($n = 0, 1, 2$) 설정에 따른 트리거 소스

표 3에서 볼 수 있듯이 ADTSn = 000_2 이외에는 비교기, 외부 인터럽트 요청, 타이머/카운터 등에 의해 트리거가 발생하는 경우로 이들은 뒤에서 다시 다룬다. 트리거 소스의 디폴트값은 000_2 으로 프리러닝 모드로 설정되어 있다. 따라서 ADCSRA 레지스터의 ADATE 비트가 0인 경우에는 단일 변환 모드, ADATE 비트가 1인 경우에는 프리러닝 모드인 점은 기억하기 바란다.

4. 아두이노의 AD 변환

아날로그 데이터를 디지털 값으로 변환하여 받아들이기 위해서는 앞 절에서 살펴본 것처럼 3

개에서 4개의 레지스터를 사용하여야 한다. UART 시리얼 통신을 위해 5개의 레지스터를 사용하여야 하지만 아두이노에서는 Serial 클래스를 통해 간단히 수행할 수 있었던 것을 기억하는가? AD 변환 역시 마찬가지이다. 아두이노에서는 analogRead라는 함수를 통해 간단하게 아날로그 데이터를 디지털로 변환하여 받아들일 수 있다. 코드 2는 코드 1과 동일한 기능을 수행하는 아두이노 코드이다.

코드 2. 가변저항 읽기 - 아두이노 스타일

```
void setup()
{
    Serial.begin(9600);           // UART 통신 초기화
}

void loop()
{
    int read = analogRead(A0);    // 아날로그 값 읽기
    Serial.println(read);         // 아날로그 값을 시리얼 모니터로 출력

    delay(1000);                 // 1초 대기
}
```

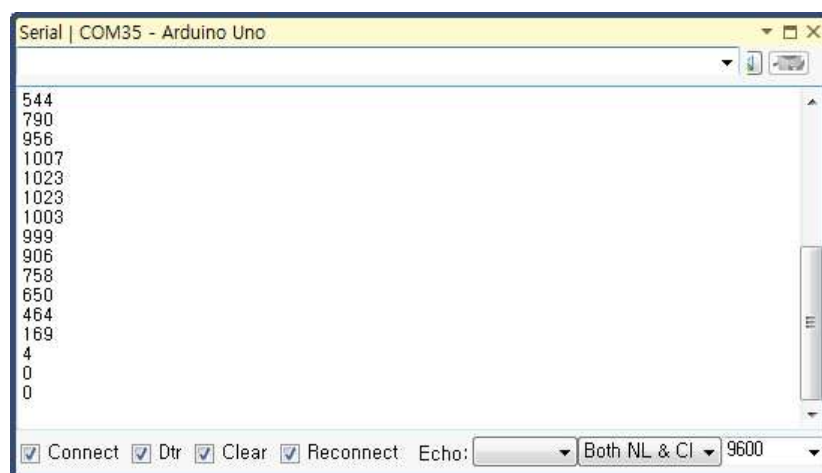


그림 9. 코드 2 실행 결과

아날로그 값을 읽어 들이기 위한 아두이노 함수는 analogRead이다. analogRead 함수는 핀 번호만을 지정해 주면 지정한 핀으로부터 아날로그 값을 읽어 디지털 값으로 변환한 후 반환

해 준다.

```
int analogRead(uint8_t pin)
매개변수
    pin : 핀 번호
반환값 : 0에서 1023 사이의 정수값
```

analogRead 함수는 디폴트로 AVCC를 기준 전압으로 사용한다. 다른 기준 전압을 사용하고 싶다면 analogReference 함수를 통해 바꿀 수 있다.

```
void analogReference(uint8_t type)
매개변수
    type : DEFAULT, INTERNAL, EXTERNAL 중 한 가지
반환값 : 없음
```

- ♦ DEFAULT : AVCC 5V 값을 기준 전압으로 설정한다.
- ♦ INTERNAL : 내부 1.1V를 기준 전압으로 설정한다.
- ♦ EXTERNAL : AREF 핀에 인가된 0V ~ 5V 사이의 전압을 기준 전압으로 설정한다.

이외에도 몇 가지 type이 더 있지만 아두이노 우노에서 사용할 수 없는 옵션들이므로 생략하였다.