
성 균 관 대 학 교
SIOR
로 봇 학 회

2022년 05월 08일

EMBEDDED

4 주 차

목차

atmega328

버튼 입력

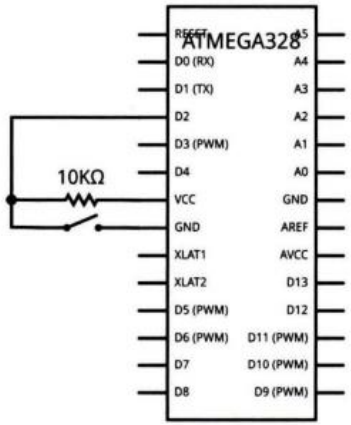

그림 11-2 풀업 저항을 사용한 버튼 연결 회로도

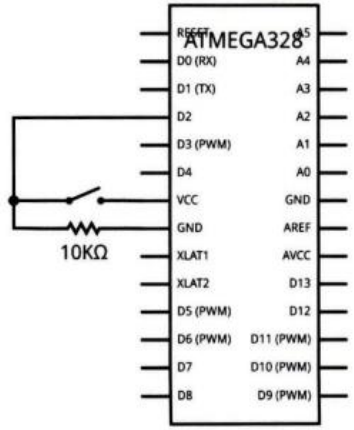

그림 11-3 풀다운 저항을 사용한 버튼 연결 회로도

표 11-1 풀업 및 풀다운 저항 사용에 의한 디지털 입력

	스위치 ON	스위치 OFF
풀다운 저항	1	0
풀업 저항	0	1

ATmega328에는 풀업 저항 포함

① DDRx : 데이터 방향(입력 or 출력)

② PINx : 디지털 데이터 읽어오기

③ PORTx : 출력값 기록(DDRx=1),
풀업 저항 사용 여부
(DDRx=0)

atmega328

버튼 입력

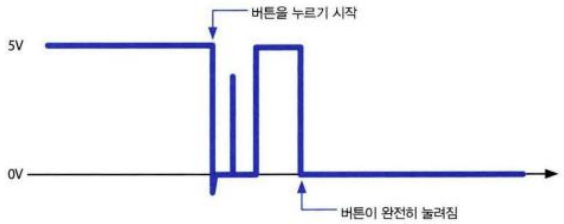

그림 11-8 채터링 현상

① 버튼 누르고 일정시간 입력 무시
② 짧게 2번 검사

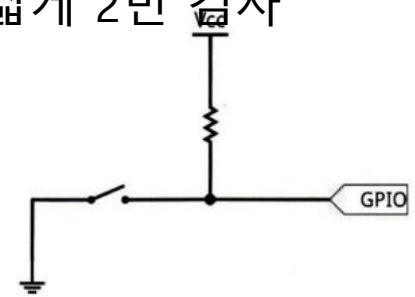

그림 11-9 풀업 저항을 사용한 버튼 회로

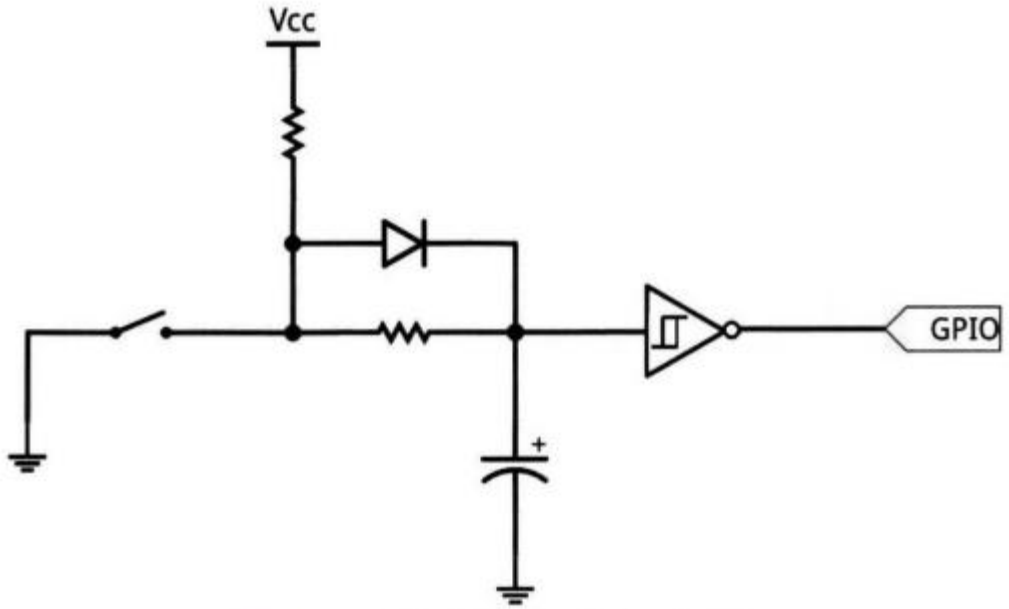

그림 11-12 디바운스 회로를 포함하는 버튼 회로 2

C : 진동에 의한 전압 변동 일부 흡수
Diode : 충전속도 빠르게
R : 방전속도 빠르게
슈미트 트리거: 완만한 전압 변화

atmega328

버튼 입력

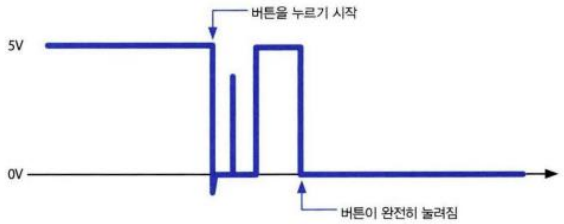

그림 11-8 채터링 현상

① 버튼 누르고 일정시간 입력 무시
② 짧게 2번 검사

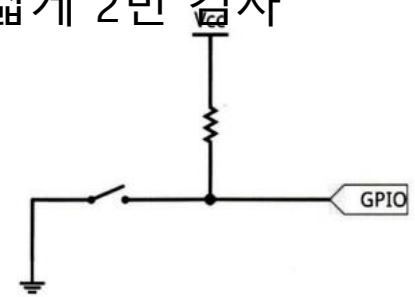

그림 11-9 풀업 저항을 사용한 버튼 회로

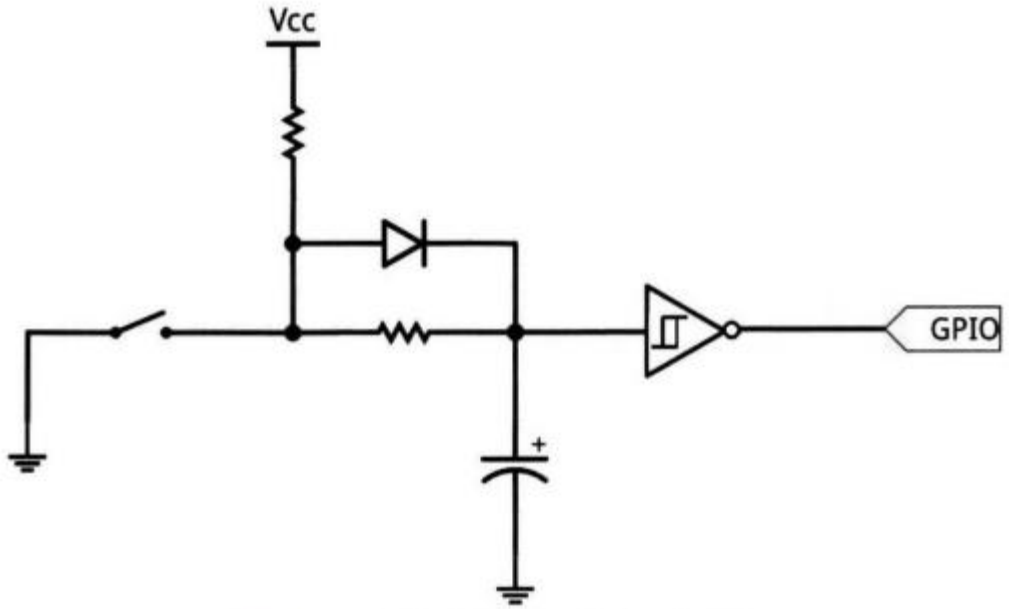

그림 11-12 디바운스 회로를 포함하는 버튼 회로 2

C : 진동에 의한 전압 변동 일부 흡수
Diode : 충전속도 빠르게
R : 방전속도 빠르게
슈미트 트리거: 완만한 전압 변화

EMBEDDED

ATmega328의 ADC

마이크로 컨트롤러: 주변 환경으로부터 데이터를 획득, 이를 처리하여 시스템을 제어하는 목적으로 사용
ADC 필요한 이유: 주변 환경으로부터 획득할 수 있는 데이터(주로 센서로 획득)는 아날로그 데이터지만 마이크로 컨트롤러는 binary의 디지털 데이터만 처리 가능
아날로그 전압 입력->ADC(디지털데이터로 변환)->CPU
*ADC(Analog-Digital Converter, 아날로그-디지털 변환기)

EMBEDDED

ATmega328의 ADC

10비트 해상도의 ADC
입력되는 아날로그 전압을 0~1023-1까지 표현 가능

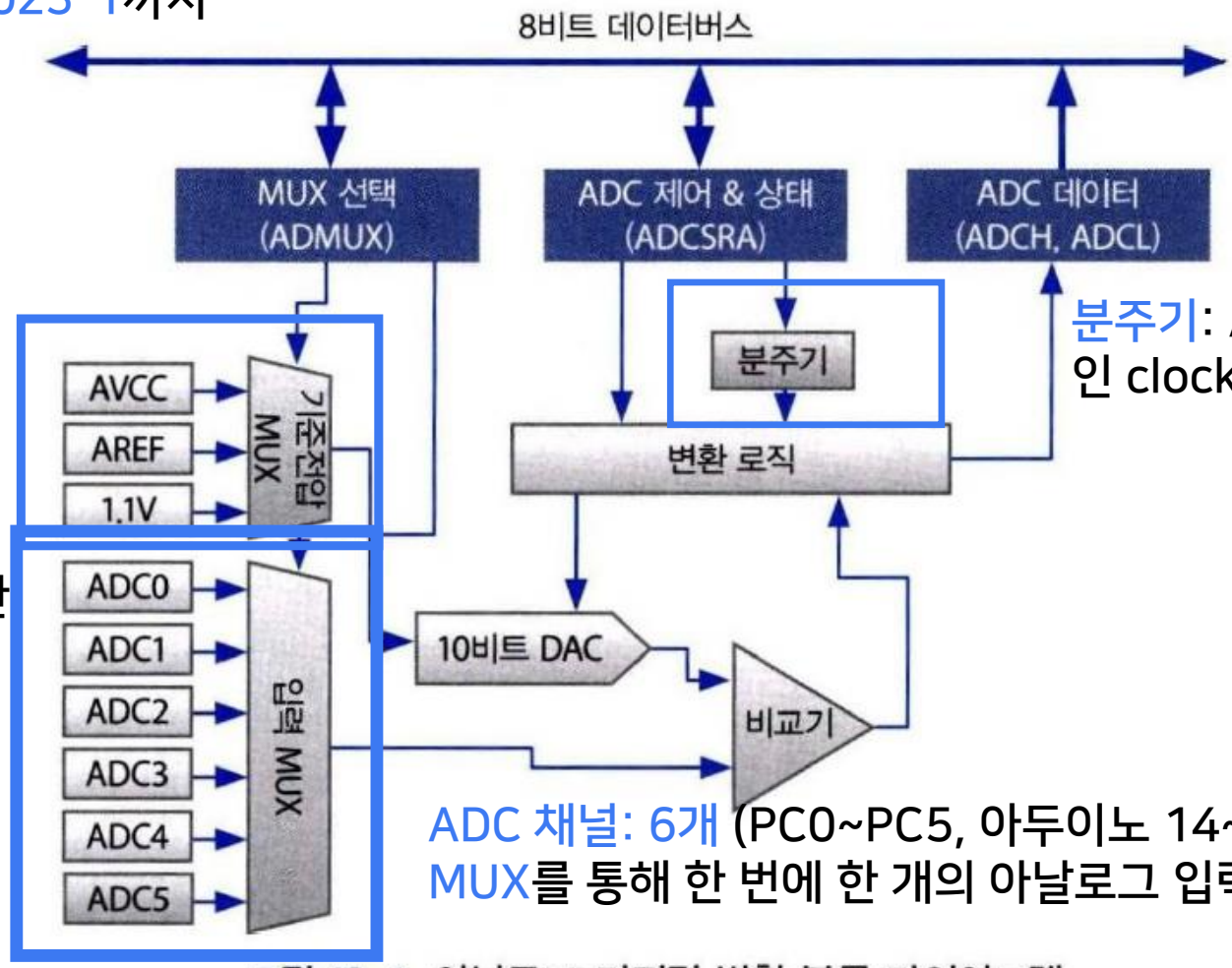

그림 12-1 아날로그-디지털 변환 블록 다이어그램

기준전압: AD변환에서 1023의 디지털 값으로 변환할 값 의미
입력 채널의 값과 기준전압을 비교해 가며 변환이 이루어짐
*기준 전압 5V라면, 구별 가능한 전압의 차이는 약 5/1023=4.89mV 정도이다.

분주기: AD변환은 시간 많이 걸려서 메인 clock 속도를 낮추는데 사용

ADC 채널: 6개 (PC0~PC5, 아두이노 14~19번 핀)
MUX를 통해 한 번에 한 개의 아날로그 입력만 디지털로 변환 가능

EMBEDDED

AD 변환을 위한 레지스터

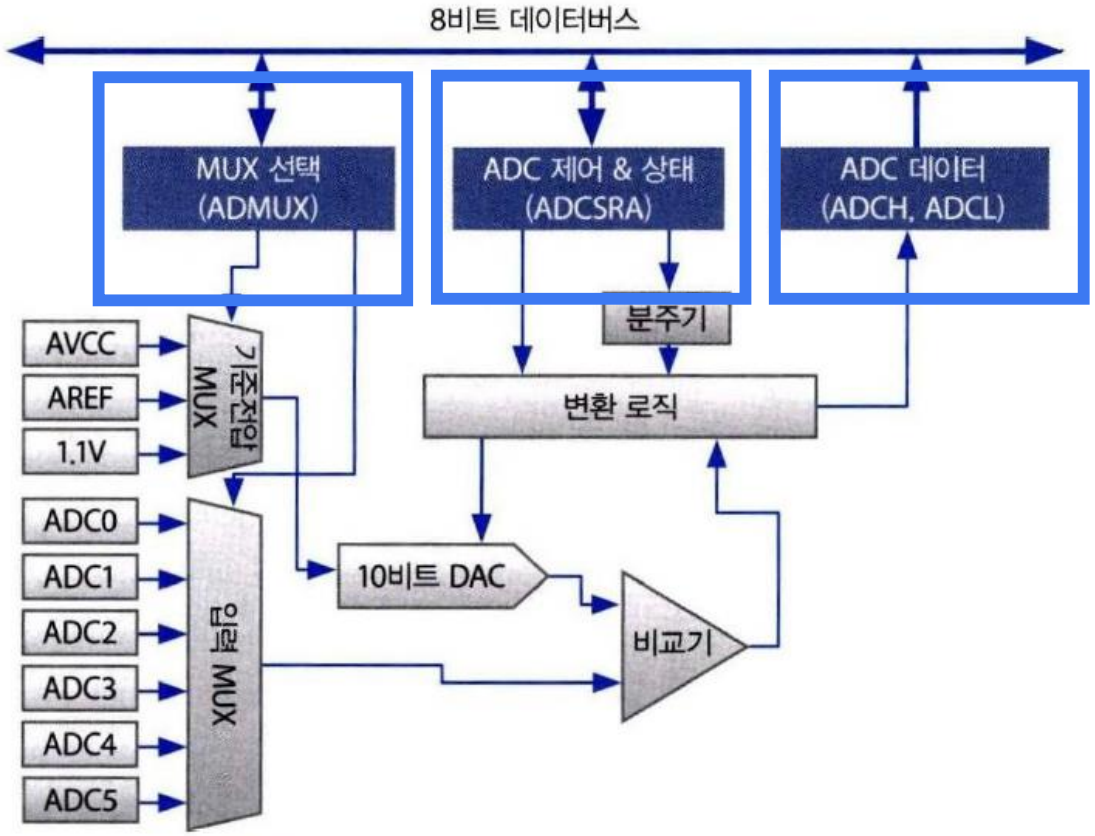

그림 12-1 아날로그-디지털 변환 블록 다이어그램

ADMUX: AD변환 위해 기준 전압과 입력 채널 설정을 위한 레지스터

ADCSRA: AD 과정을 제어하기 위한 레지스터

ADC: 변환된 데이터를 저장하기 위한 레지스터

1. ADC(저장 R) (ADCH, ADCL 조합)

ADC: 해상도가 10bit
ADCH, ADCL: 레지스터는 8bit이므로 2개 사용 (변환된 데이터를 나누어 저장), 6bit는 사용X

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 12-4 ADCH 및 ADCL 레지스터의 구조

데이터 정렬 방식 2가지(ADMUX에 의해 결정됨)
오른쪽 정렬: ADCH의 상위 6it사용 X
왼쪽 정렬: ADCL의 하위 6bit 사용 X
두 개 합한 것을 16bit 크기의 가상 레지스터 ADC로 사용
오른쪽 정렬이 디폴트 값

2. ADMUX

ADMUX: AD변환을 위한 기준 전압과 입력 채널 선택 위해 사용

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 12-5 ADMUX 레지스터의 구조

비트 7~6 REFS0

: 내부 또는 외부 기준 전압 선택

비트 3~0 MUX3~0

: 입력 채널 선택(Channel Selection)을 위해 사용

0000 ~ 0101, ADC0 ~ ADC5까지 6개 채널 사용가능

표 12-1 REFS1, REFS0 설정에 따른 기준 전압

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.

3. ADCSRA

ADCSRA: ADC Control and Status Register A
AD변환의 상태를 나타내거나 AD 변환 과정을 제어하기 위해 사용된다.

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 12-6 ADCSRA 레지스터의 구조

비트 7 – ADEN(ADC Enable)

비트 6 – ADSC(ADC Start Conversion)

비트 5 – ADATE(ADC Auto Trigger Enable)

비트 4 – ADIF(ADC Interrupt Flag)

비트 3 – ADIE(ADC Interrupt Enable)

비트 2~0 – ADPS2~0(ADC Prescaler Select Bits)

표 12-2 ADPSn(n = 0, 1, 2) 값에 따른 분주율

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

아두이노 우노 clock=16MHz ->너무 빠름

$$50 \times 10^3 Hz < \frac{16 \times 10^6 Hz}{\text{분주율}} < 200 \times 10^3 Hz$$

4. ADCSRB

ADCSRB: 트리거 소스 선택을 설정

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 12-7 ADCSRB 레지스터의 구조

*000제외하고는 모두 트리거가 발생하는 경우임

비트6 ACME(Analog Cpmparator Multipl
: 2개 값을 비교하는 비교기를 위해 사용되는

비트2~0 ADTSn(n-0,1,2) (ADC Auto Trigger Source)
: AD 변환을 시작하는 트리거 신호를 선택할 수 있도록 해준다.
트리거 소스의 디폴트 값은 $000_{(2)}$: 프리러닝 모드
(ADCSRA 레지스터의 ADATE 비트가 1인 경우 프리러닝 모드
0이면 단일 모드 임을 기억)

표 12-4 ADTSn(n = 0, 1, 2) 설정에 따른 트리거 소스

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

가변저항 값 읽기 실습

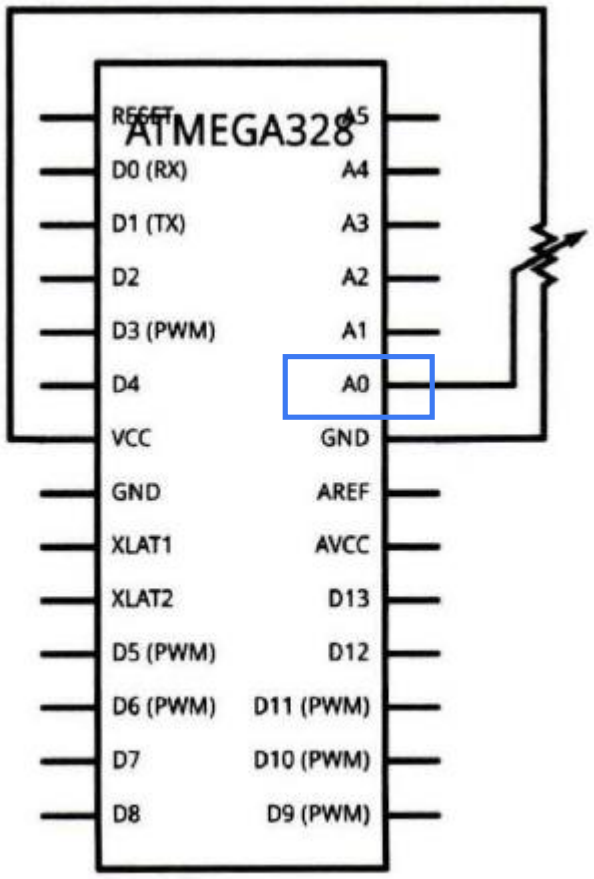

그림 12-2 가변저항 연결 회로도

코드 12-1 가변저항 읽기 – C 스타일

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

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

    ADCSRA |= 0x07;                          // 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(void)
{
    while(!(ADCSRA & (1 << ADIF)));          // 변환 종료 대기

    return ADC;                              // 10비트 값을 반환
}
```

```c
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(0);                             // AD 변환기 초기화

    while(1)
    {
        read = read_ADC();                   // 가변저항 값 읽기

        int_to_string(read, buffer);         // 정수값을 문자열로 변환
        UART_printString(buffer);            // 문자열을 UART로 전송
        UART_printString("\n");              // 줄 바꿈

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

A0에 연결된 가변저항 값을 읽어 1초에 한 번 출력해주는 코드
결과: 0~1023까지 값 출력됨

가변저항 값 읽기 실습

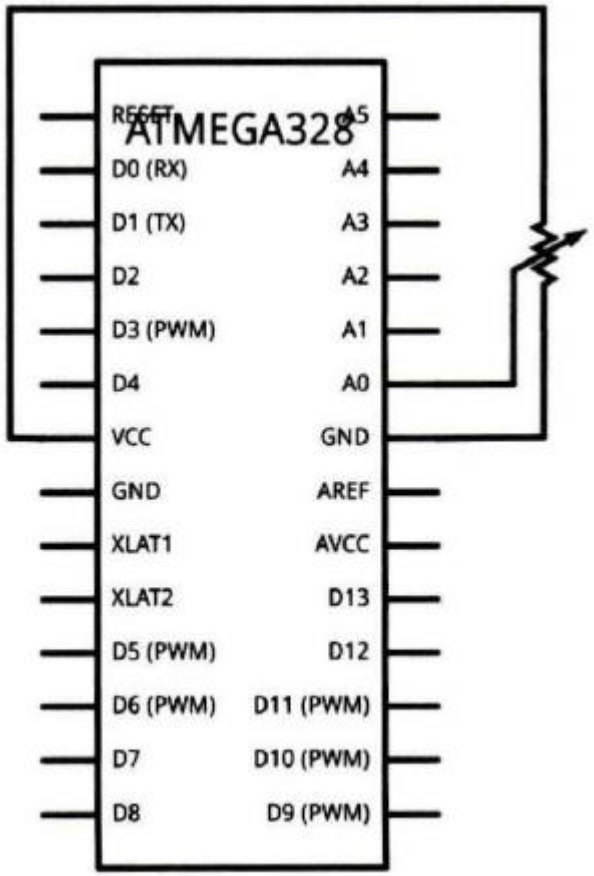

그림 12-2 가변저항 연결 회로도

코드 12-2 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
}
```

*아두이노에서는 analogRead 함수를 통해 간단하게 아날로그 데이터를 디지털로 변환하여 읽어들일 수 있다.

EMBEDDED

가변저항 값 읽기 실습

*analogRead 함수: 디폴트로 기준전압 AVCC 사용

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

*analogReference 함수: 기준 전압 변경 가능, 3가지 경우 가능

```
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.

가변저항으로 LED 제어 실습

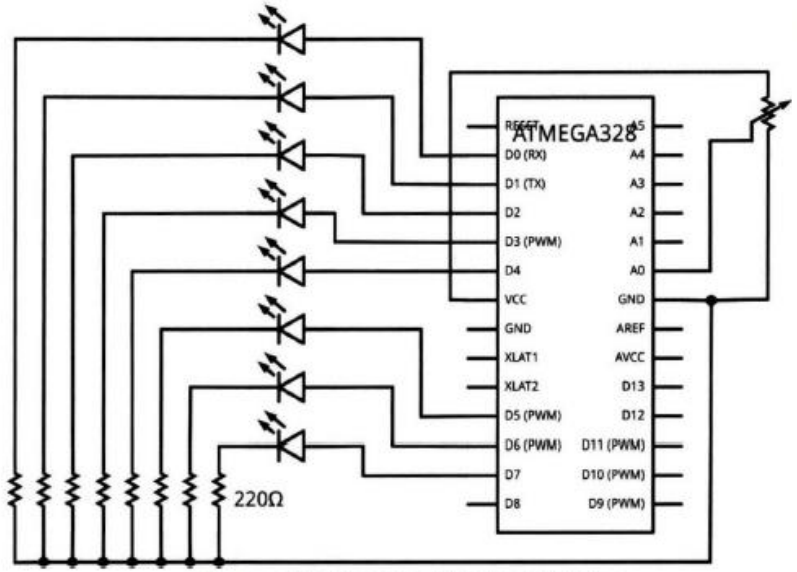

그림 12-9 가변저항으로 LED를 제어하기 위한 회로

코드 12-3 가변저항으로 LED 개수 제어 – C 스타일

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

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

    ADCSRA |= 0x07;                         // 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(void)
{
    while(!(ADCSRA & (1 << ADIF)));         // 변환 종료 대기

    return ADC;                             // 10비트 값을 반환
}

void PORT_INIT(void)
{
    DDRD = 0xFF;                            // LED 연결 핀을 출력으로 설정
    PORTD = 0x00;                           // LED는 꺼진 상태로 시작
}

int main(void)
{
    int value;
    uint8_t on_off;

    ADC_INIT(0);                            // ADC 초기화
    PORT_INIT();

    while(1)
    {
        value = read_ADC() >> 7;            // AD 변환된 값의 최상위 3비트만 남김

        on_off = 0;
        for(int i = 0; i <= value; i++){    // 8개의 LED 점멸 패턴 결정
            on_off |= (0x01 << i);
        }
        PORTD = on_off;                     // LED 제어
    }
}
```

*idea: AD 변환된 값은 10bit이므로 7비트 만큼 오른쪽으로 이동해 최상위 3bit를 이용해서 8개의 led 조절 가능

EMBEDDED

가변저항으로 LED 제어 실습

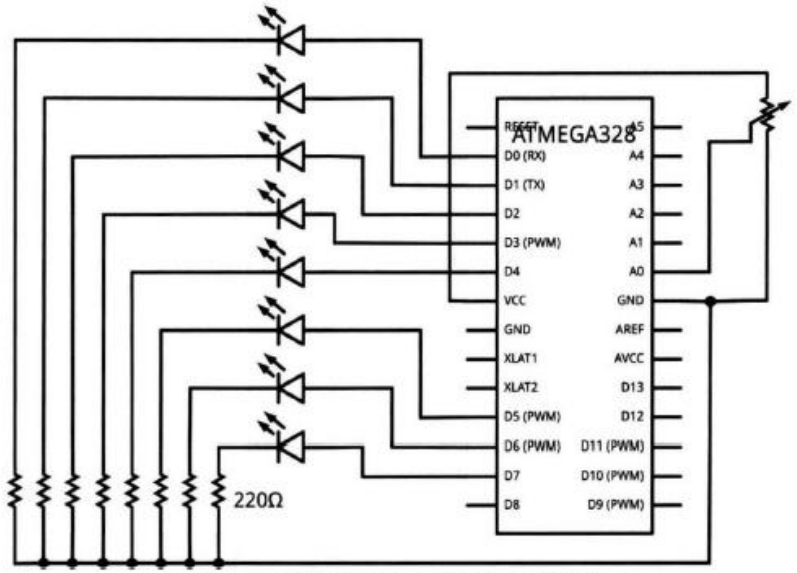

그림 12-9 가변저항으로 LED를 제어하기 위한 회로

코드 12-4 가변저항으로 LED 개수 제어 – 아두이노 스타일

```
int LED_pins[] = {0, 1, 2, 3, 4, 5, 6, 7};          // LED 연결 핀

void setup()
{
    for(int i = 0; i < 8; i++){
        pinMode(LED_pins[i], OUTPUT);               // LED 연결 핀을 출력으로 설정
        digitalWrite(LED_pins[i], LOW);             // LED는 꺼진 상태로 시작
    }
}

void loop()
{
    int value = analogRead(A0) >> 7;                // AD 변환된 값의 최상위 3비트만 남김

    for(int i = 0; i <= value; i++){
        digitalWrite(LED_pins[i], HIGH);
    }

    for(int i = value + 1; i < 8; i++){
        digitalWrite(LED_pins[i], LOW);
    }
}
```

AVCC 읽기 실습

*USB를 통해 전원을 공급받아 사용하는데, 그 전압을 측정해보면 정확하게 5V가 X
고정X, 컴퓨터의 동작상태에 따라 변함->출력 전압 범위가 좁은 경우 오류로 이어질 가능성 有
*ATmega328에서는 AVCC값 계산 가능
원리: 내부 1.1V 전압을 기준으로 현재 AVCC값을 측정한다.

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 12-5 ADMUX 레지스터의 구조

ADC 채널 선택할 때 사용하는 ADMUX의 하위 4bit를 이용하여 내부 기준 전압 1.1V를 읽어들일 수 있다. (1110, 14번 채널 선택하면 됨)
내부 1.1V는 외부 AVCC에 비해 안정적->1.1V기준으로 비교

A D A D
1.1 : read_from_ch_14 = AVCC : 1023

AVCC = (1.1 * 1023 / read_from_ch_15)

표 12-5 MUX3...0 비트 선택에 따른 채널 선택

MUX3	MUX2	MUX1	MUX0	입력
0	0	0	0	ADC0
0	0	0	1	ADC1
0	0	1	0	ADC2
0	0	1	1	ADC3
0	1	0	0	ADC4
0	1	0	1	ADC5
0	1	1	0	ADC6
0	1	1	1	ADC7
1	0	0	0	ADC8
1	0	0	1	-
1	0	1	0	-
1	0	1	1	-
1	1	0	0	-
1	1	0	1	-
1	1	1	0	1.1V
1	1	1	1	0V(GND)

*실습을 해보면 5V와 가깝긴 하지만 값이 계속 변함을 확인할 수 있다.

EMBEDDED

AVCC 읽기 실습

코드 12-5 AVCC 읽기

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

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

    ADCSRA |= 0x07;                         // 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(void)
{
    while(!(ADCSRA & (1 << ADIF)));         // 변환 종료 대기

    return ADC;                             // 10비트 값을 반환
}

int main(void)
{
    int read, actualVCC;

    ADC_INIT(0x0E);                         // 14번 채널 선택
    UART_INIT();                            // UART 통신 초기화

    while(1)
    {
        read = read_ADC();                  // 14번 채널 값 읽기
        actualVCC = 1125300L / read;        // mV 단위로 변환

        UART_print16bitNumber(actualVCC); // AVCC 값 출력
        UART_printString("\n");             // 줄 바꿈

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

코드 12-6 AVCC 읽기 - 아두이노 스타일

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

    ADCSRA |= 0x07;                         // 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(void)
{
    while(!(ADCSRA & (1 << ADIF)));         // 변환 종료 대기

    return ADC;                             // 10비트 값을 반환
}

void setup()
{
    ADC_INIT(0x0E);                         // 14번 채널 선택
    Serial.begin(9600);                     // UART 통신 초기화
}

void loop()
{
    int read, actualVCC;

    read = read_ADC();                      // 14번 채널 값 읽기
    actualVCC = 1125300L / read;            // mV 단위로 변환

    Serial.println(actualVCC);              // AVCC 값 출력

    delay(1000);
}
```

EMBEDDED

난수 생성 실습

*마이크로컨트롤러에서 생성되는 난수는 실제 난수가 아닌 계산에 의해 생성되는 의사 난수
항상 동일한 순서로 만들어짐
해결방안: 시작값을 현재 시간으로 설정하여 난수를 발생시키자
*Atmega328에는 현재 시간 알아낼 수 있는 장치X
대안: 회로가 연결되지 않은 아날로그 입력 핀의 입력값을 사용

*C
rand함수
srand 함수

*아두이노
random함수
randomSeed함수

코드 12-7 난수 생성

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

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

    ADCSRA |= 0x07;                         // 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(void)
{
    while(!(ADCSRA & (1 << ADIF)));         // 변환 종료 대기

    return ADC;                             // 10비트 값을 반환
}

int main(void)
{
    UART_INIT();                            // 시리얼 통신 초기화
    ADC_INIT(0);                            // ADC 초기화
    srand(read_ADC());                      // 난수열 초기화

    UART_printString("** Start generating random number...\n");

    while(1)
    {

        int randomNumber = rand() % 100 + 1;    // 1~100 사이 난수 생성

        UART_print8bitNumber(randomNumber);
        UART_transmit('\n');

        _delay_ms(1000);
    }
}
```

코드 12-8 난수 생성 - 아두이노 스타일

```
void setup()
{
    Serial.begin(9600);                         // 시리얼 통신 초기화
    randomSeed(analogRead(A0));                 // 아날로그 값 읽기
    Serial.println("** Start generating random number...");
}

void loop()
{
    Serial.println(random(1, 101));             // 1~100 사이 난수 생성
    delay(1000);
}
```

```
long random(long max)
long random(long min, long max)
    min: 생성될 난수의 최솟값
    max: max – 1이 생성될 난수의 최댓값
- 반환값: [min, max – 1] 범위의 난수값
```

```
void randomSeed(unsigned int seed)
    seed: 의사 난수의 시작 위치 결정을 위한 값
- 반환값: 없음
```

폴링 방식

- 어떤 특정한 사건이 발생하였는지를 반복적으로 검사하고 사건이 발생하였을 때 특정한 동작을 수행하는 방식

- 단점 – 두 가지 작업 동시 진행 불가

- → 인터럽트

인터럽트

- 하드웨어 / 소프트웨어 인터럽트
- 마이크로컨트롤러에서의 인터럽트 – 하드웨어

- 하드웨어 인터럽트 – 외부 / 내부

- 내부 – ATmega328 내부의 장치에 의해 발생
- 외부 – ATmega328의 범용 입출력 핀에 가해지는 입력의 변화

인터럽트 핀

- 인터럽트 발생
- 인터럽트 번지로 이동
- 인터럽트를 처리할 ISR의 주소 찾기
- ISR로 이동하여 처리

Interrupt Service Routine - ISR

표 13-1 인터럽트 벡터 테이블

벡터 번호	프로그램 주소	소스	인터럽트 정의
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

인터럽트 발생 조건

- SREG 레지스터
- → 전역적인 인터럽트 허용

- + ADC 인터럽트

- ADCSRA 레지스터의 ADIE
- → 개별 인터럽트 허용

표 13-2 상태 레지스터 비트

비트 번호	비트 이름	설명
7	I	Global Interrupt Enable: 전역적인 인터럽트 발생을 허용한다.
6	T	Bit Copy Storage: 비트 복사를 위한 BLD(Bit Load), BST(Bit Store) 명령에서 사용한다. BST 명령에 의해 비트 값을 비트 T에 저장할 수 있으며, BLD 명령에 비트 T의 내용을 읽어 올 수 있다.
5	H	Half Cary Flag: 산술 연산에서의 보조 캐리 발생을 나타낸다. 보조 캐리는 바이트 단위 연산에서 하위 니블(nibble)로부터 발생하는 캐리를 말한다.
4	S	Sign Bit: 부호 비트로 음수 플래그(N)와 2의 보수 오버플로 플래그(V)의 배타적 논리합(XOR)으로 설정된다. ($S = N \oplus V$)
3	V	2's Complement Overflow Flag: 2의 보수를 이용한 연산에서 자리 올림이 발생하였음을 나타낸다.
2	N	Negative Flag: 산술 연산이나 논리 연산에서 결과가 음수임을 나타낸다.
1	Z	Zero Flag: 산술 연산이나 논리 연산에서 결과가 영(zero)임을 나타낸다.
0	C	Carry Flag: 산술 연산이나 논리 연산에서 캐리가 발생하였음을 나타낸다.

코드 13-3 인터럽트 처리 루틴

```
#include <avr/interrupt.h>

ISR(ADC_vect)
{
    // 인터럽트 처리 코드
}
```

표 13-3 인터럽트 벡터 이름

벡터 번호	벡터 이름	인터럽트 정의
1	-	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	INT0_vect	External Interrupt Request 0
3	INT1_vect	External Interrupt Request 1
4	PCINT0_vect	Pin Change Interrupt Request 0
5	PCINT1_vect	Pin Change Interrupt Request 1
6	PCINT2_vect	Pin Change Interrupt Request 2
7	WDT_vect	Watchdog Time-out Interrupt
8	TIMER2_COMPA_vect	Timer/Counter2 Compare Match A
9	TIMER2_COMPB_vect	Timer/Counter2 Compare Match B
10	TIMER2_OVF_vect	Timer/Counter2 Overflow
11	TIMER1_CAPT_vect	Timer/Counter1 Capture Event
12	TIMER1_COMPA_vect	Timer/Counter1 Compare Match A
13	TIMER1_COMPB_vect	Timer/Counter1 Compare Match B
14	TIMER1_OVF_vect	Timer/Counter1 Overflow
15	TIMER0_COMPA_vect	Timer/Counter0 Compare Match A
16	TIMER0_COMPB_vect	Timer/Counter0 Compare Match B
17	TIMER0_OVF_vect	Timer/Counter0 Overflow
18	SPI_STC_vect	SPI Serial Transfer Complete
19	USART_RX_vect	USART Rx Complete
20	USART_UDRE_vect	USART, Data Register Empty
21	USART_TX_vect	USART, Tx Complete
22	ADC_vect	ADC Conversion Complete
23	EE_READY_vect	EEPROM Ready
24	ANALOG_COMP_vect	Analog Comparator
25	TWI_vect	2-wire Serial Interface
26	SPM_READY_vect	Store Program Memory Ready

인터럽트 주의 사항

1. ISR 내부에 또다른 인터럽트 지양
- 메모리 문제, 중요한 인터럽트 놓칠 가능성
- (전역 인터럽트 활성 비트인 I 비트 자동 클리어)

2. 낮은 번호의 비트가 우선순위

인터럽트 관련 명령어

- Volatile
- 변수가 다른 위치에서 변할 수 있게

- ATOMIC_BLOCK – 한 클록에 데이터 처리 불가 대안
- ATOMIC_BLOCK 내부에서는 인터럽트 불가
- ATOMIC_FORCEON, ATOMIC_RESTORESTATE (변수)
- 상태 레지스터 값 보존 여부

외부 인터럽트

• INT0 (디지털 2핀만), INT1 (디지털 3핀만)
(RESET을 제외한 우선 순위 짱)
→ EIMSK

• 인터럽트 발생 시점
→ EICRA
LOW
CHANGE
RISING
FALLING

표 13-4 ISC 비트 설정에 따른 인터럽트 발생 시점(n = 0 또는 1)

ISCn1	ISCn0	인터럽트 발생 시점
0	0	INTn 핀의 입력값이 LOW일 때 인터럽트가 발생한다(low level).
0	1	INTn 핀의 입력값이 LOW에서 HIGH 또는 그 반대로 변할 때 인터럽트가 발생한다(logical change).
1	0	입력값의 하강 에지에서 인터럽트가 발생한다(falling edge).
1	1	입력값의 상승 에지에서 인터럽트가 발생한다(rising edge).

성 균 관 대 학 교

Thank You

로 봇 동 아 리