



AVR - ADC

임베디드스쿨 2기

Lv1과정

2021. 07. 09

박태인

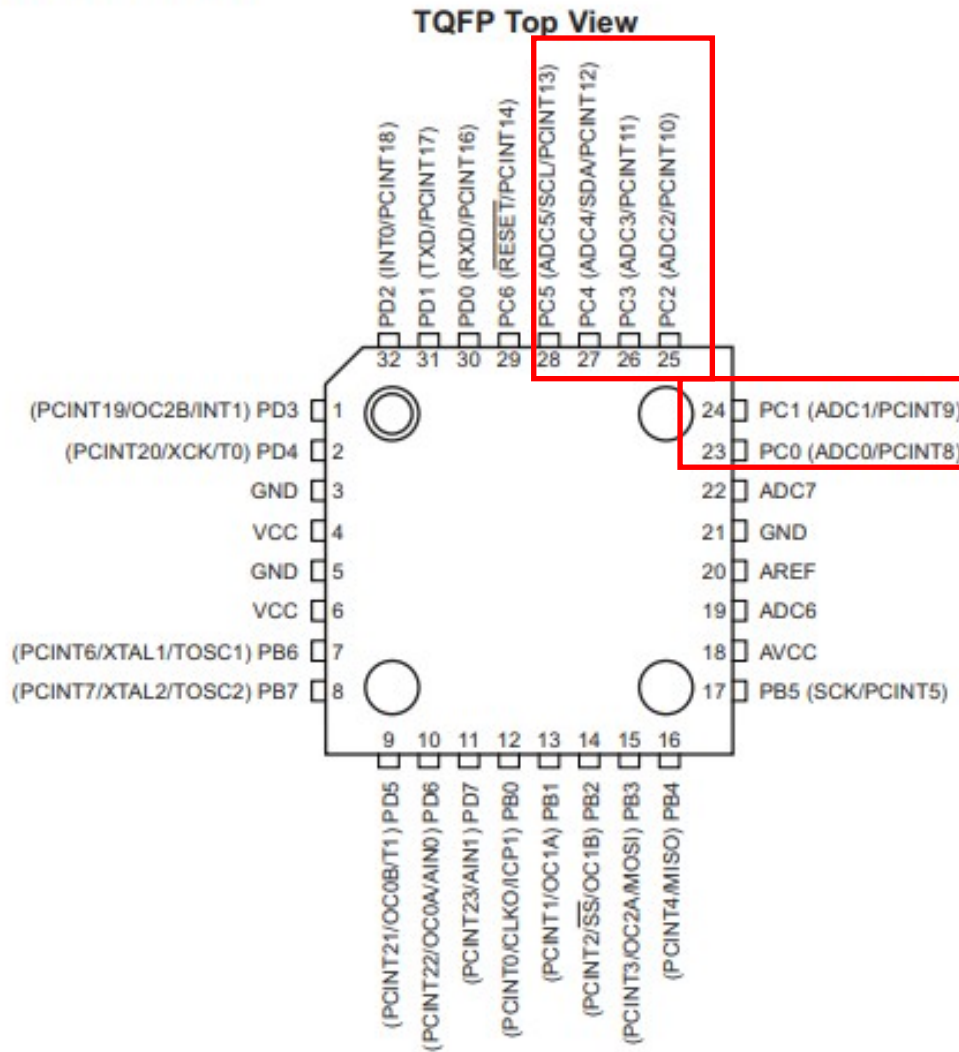
1. ADC

- ADC란?
 - ↳ A(Analog) D(Digital) C(Convert) 의 약자 이다.
즉, 아날로그 신호를 디지털 신호로 바꾸어 주는 역할을 한다.
- 이것을 하는 이유?
 - ↳ 빛의 밝기나 소리의 높낮이, 바람의 세기, 온도 등등 물리적인 현상들은 아날로그 값이다. 이 값들을 MCU 에서 받으려면 0,1 인 디지털 값으로 받아 처리하여야 한다.
- Atmega328 ADC 의 특징
 - ↳ 6 채널 10-bit 분해능 ADC 존재
(여기서 분해능이란 아날로그 변화 값을 표시 할 수 있는 단계의 수)
 - ↳ 10비트 -> 2의 10승 = **1024의 입력 값이 단계로 이루어 진다.**

1. ADC

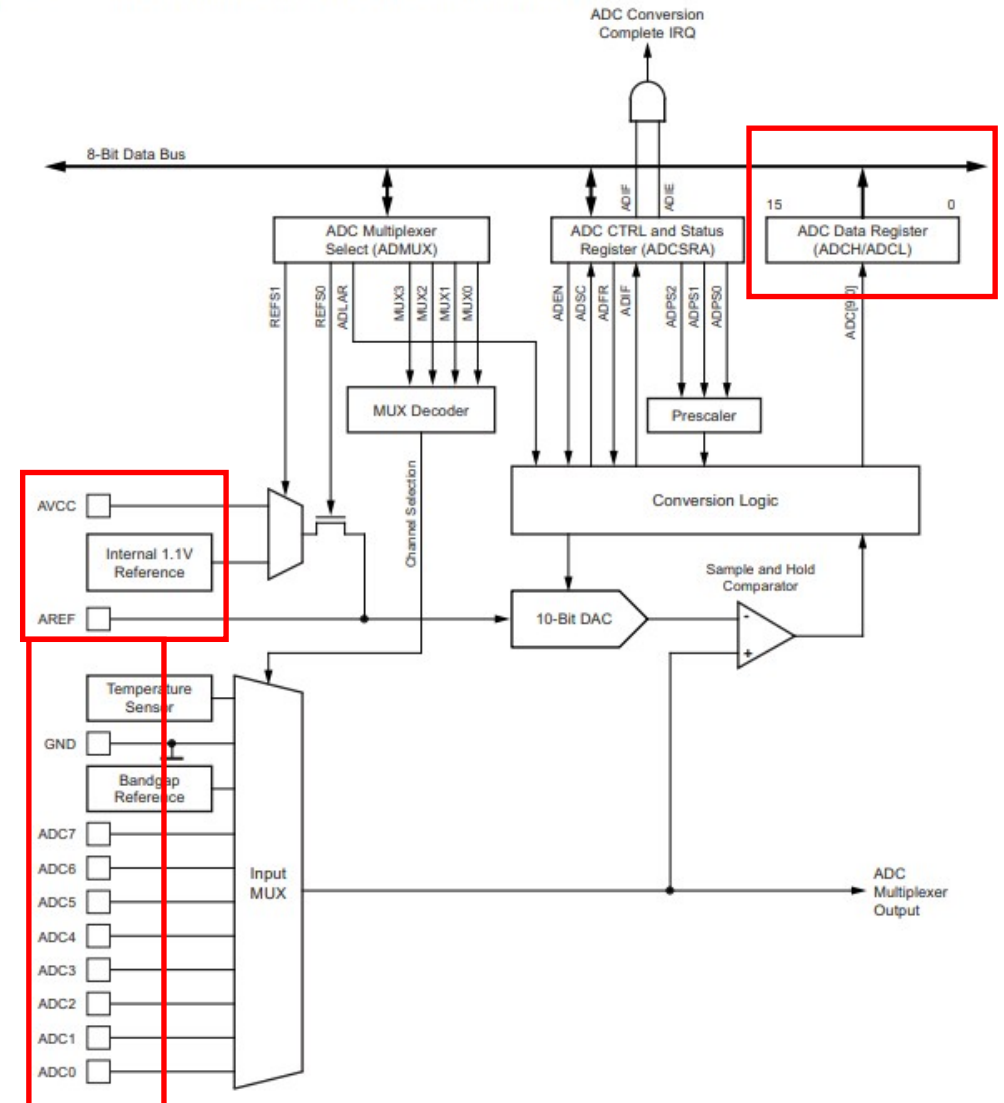
- ADC 핀 맵

Figure 1-1. Pinout



- ADC의 구성도

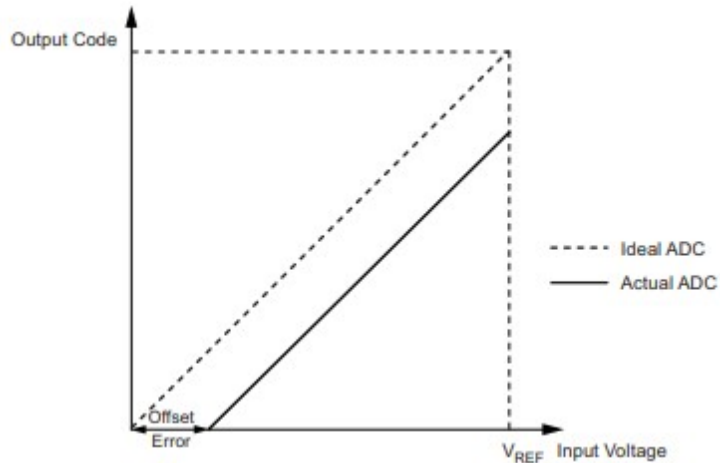
Figure 23-1. Analog to Digital Converter Block Schematic Operation



1. ADC

- 아날로그 -> 디지털 변환 오차

Figure 23-10. Offset Error

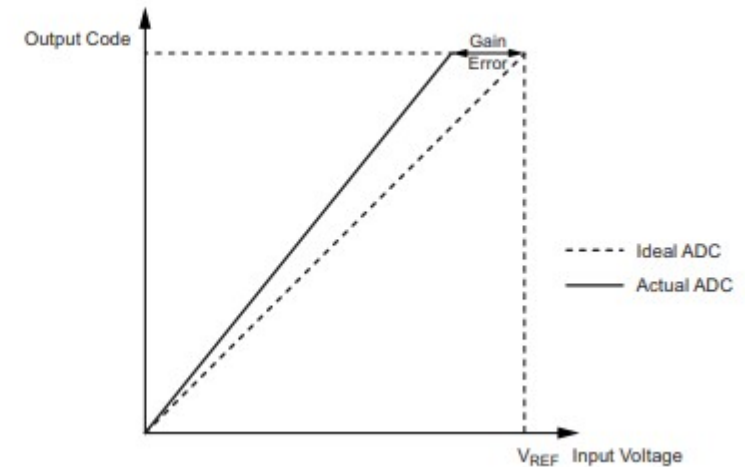


- Gain Error: After adjusting for offset, the gain error is found as the deviation of the last transition (0x3FE to 0x3FF) compared to the ideal transition (at 1.5LSB below maximum). Ideal value: 0LSB

- 오프셋오차(Offset Error) : 변환 결과가 디지털 값에서 일정한 양으로 벗어난 경우
- 해결방안 : 분해능을 높인 ADC 사용

- Integral Non-linearity (INL): After adjusting for offset and gain error, the INL is the maximum deviation of an actual transition compared to an ideal transition for any code. Ideal value: 0LSB.

Figure 23-11. Gain Error

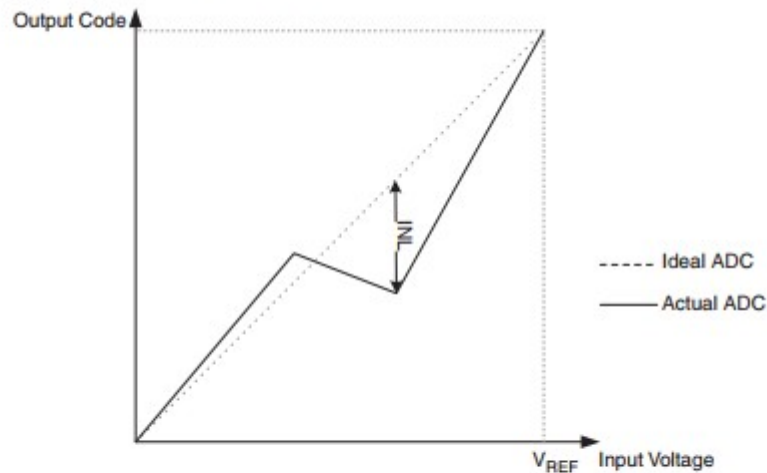


- 이득오차(Gain Error) : 변환 결과가 디지털 값에서 일정한 비율로 벗어난 경우
- 해결방안 : 변환된 디지털 값을 곱하거나 나눈다. (기울어진 축에 따라)

1. ADC

- 아날로그 -> 디지털 변환 오차

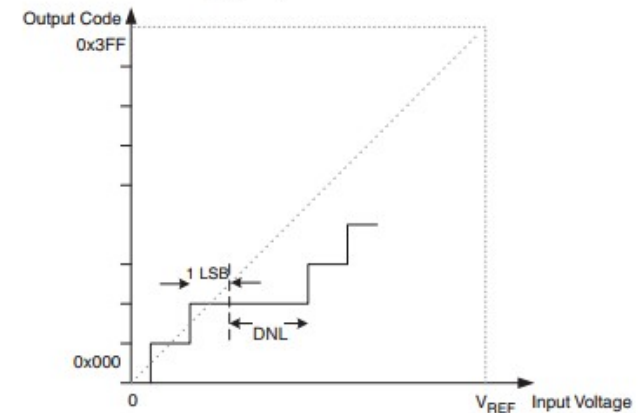
Figure 117. Integral Non-linearity (INL)



- Differential Non-linearity (DNL): The maximum deviation of the actual code width (the interval between two adjacent transitions) from the ideal code width (1LSB). Ideal value: 0LSB.

- 비선형 오차(Integral Non-Linearity Error(INL)) :
 - ↳ 변환 결과가 교정이 불가능한 상태
- 해결방안 : 없음

Figure 118. Differential Non-linearity (DNL)

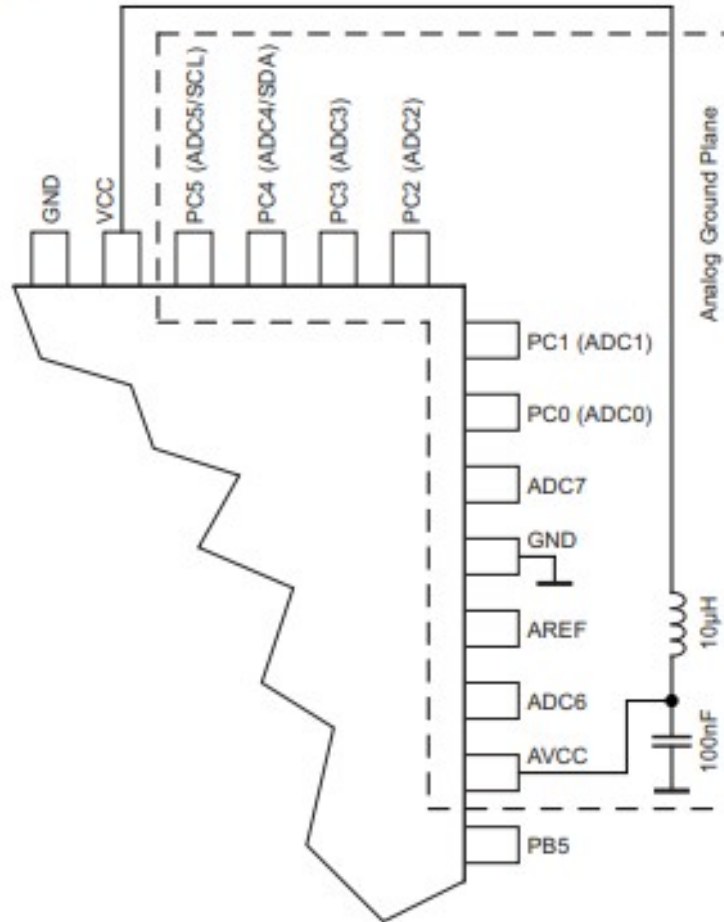


- Quantization Error: Due to the quantization of the input voltage into a finite number of codes, a range of input voltages (1LSB wide) will code to the same value. Always $\pm 0.5\text{LSB}$.
- Absolute Accuracy: The maximum deviation of an actual (unadjusted) transition compared to an ideal transition for any code. This is the compound effect of offset, gain error, differential error, non-linearity, and quantization error. Ideal value: $\pm 0.5\text{LSB}$.

- 차등 비선형 오차(Differential Non-Linearity Error(DNL)) :
 - ↳ 변환 결과가 교정이 불가능한 상태
- 해결방안 : 없음

1. ADC

Figure 23-9. ADC Power Connections

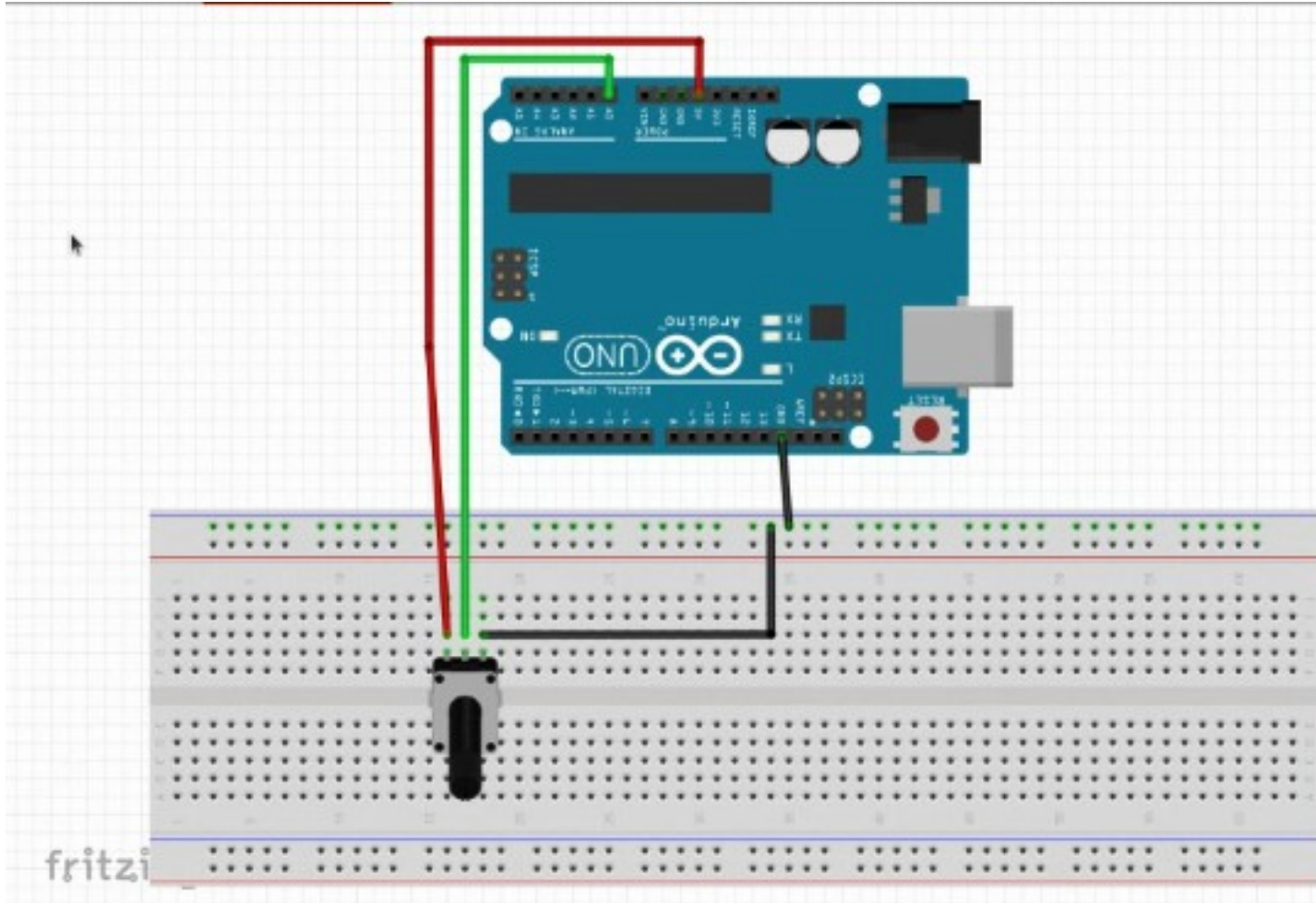


- AVCC = 독립적인 아날로그 회로 전원 단자
- AREF = 기준 전원 입력 단자
- AGND = 아날로그 회로 접지 단자

- 1) 아날로그 입력선은 최소한으로 짧게 하고 잡음의 영향이 없도록 회로를 구성한다.
- 2) 아날로그 전원단자 AVCC에 VCC를 인가 할 때는 LC 필터를 거쳐 안정화 시킨다.
- 3) 아날로그 회로의 모든 접지는 AGND에 접지하여 한 포인트에서만 GND와 접속 한다.
- 4) ADC 동작 중에는 병렬 I/O 포트의 논리상태를 스위칭 하지 않는다.
- 5) 잡음에 민감한 아날로그 소자의 ADC의 경우에는 **Adc Noise Reduction mode**를 사용한다.
- 6) 잡음이 심하여 결과값의 변동이 심하면 디지털 필터를 사용하거나 평균치를 구하여 사용한다.

1. ADC

- H/W 연결

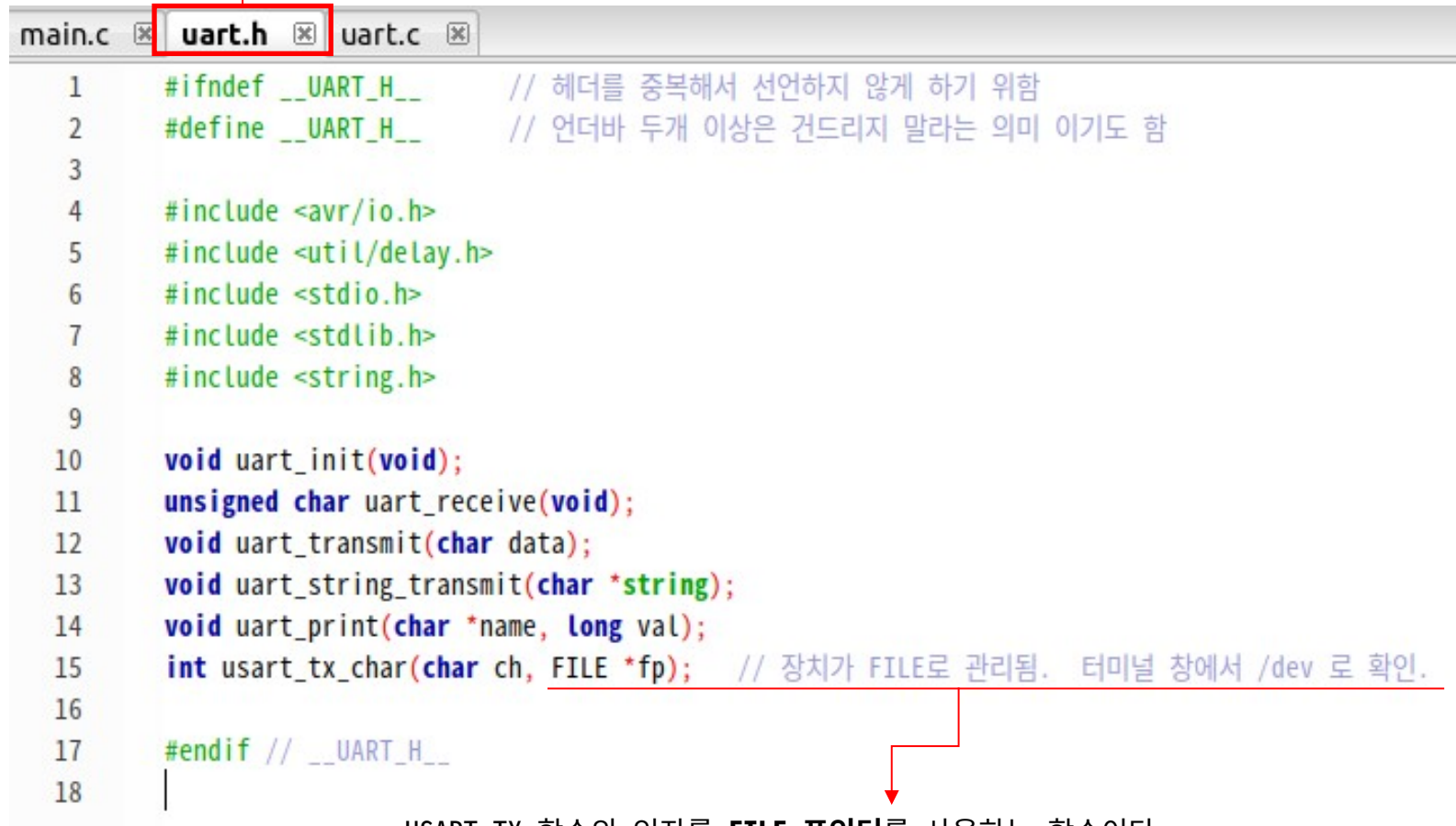


가변 저항을 이용해서 저항 값에 따른 ADC 변환을 측정하기 위한 H/W 회로 연결이다.

1. ADC - 코딩

- 지금부터는 코딩을 분석 하면서

이번에는 앞 선 수업에서 해보았던 UART 설정은 위와 같이 header 파일로 추가 한다.



```
1  #ifndef __UART_H__      // 헤더를 중복해서 선언하지 않게 하기 위함
2  #define __UART_H__      // 언더바 두개 이상은 건드리지 말라는 의미 이기도 함
3
4  #include <avr/io.h>
5  #include <util/delay.h>
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <string.h>
9
10 void uart_init(void);
11 unsigned char uart_receive(void);
12 void uart_transmit(char data);
13 void uart_string_transmit(char *string);
14 void uart_print(char *name, long val);
15 int usart_tx_char(char ch, FILE *fp); // 장치가 FILE로 관리됨. 터미널 창에서 /dev 로 확인.
16
17 #endif // __UART_H__
18
```

USART_TX 함수의 인자를 **FILE 포인터**를 사용하는 함수이다.
FILE 포인터는 낫설기에 우선 한번 알아 보고 넘어가도록 한다.

1. ADC - 코딩(파일 포인터)

- 파일 포인터

└ 여기서 파일은 실제로 흔히 아는 그 파일을 뜻한다.

지금까지 데이터를 변수에만 저장하여 활용 했는데, **사실 변수에 저장한 데이터는 프로그램이 종료되면 유지되지 않기 때문에 영구적으로 기억 할 수 없습니다.**

└ 이러한 파일을 사용할 수 있도록 C 에서 지원 하고 있습니다.

- 파일 처리

프로그램에서 파일 처리를 할 때는 반드시 루틴을 따라줘야 합니다.

open
(파일 열기) → 파일 읽기 / 쓰기 → *close*
(파일 닫기)

파일을 읽거나 쓰기와 같은 처리를 하기 전에 파일을 **open** 해야 하고, 모든 처리 후에는 반드시 닫아줘야 합니다.

• 파일을 여는 함수 : **fopen()**

└ FILE *fopen(const char *name, const char* mode)

fopen은 주어진 파일 **name**으로 파일을 생성하거나 있다면 읽어와서 **mode** 에 맞게 처리를 할 수 있도록 합니다.

즉, **FILE** 포인터를 반환하여 해당 파일에 대한 정보를 돌려 줍니다.

FILE은 **stdio.h** 에 선언 되어 있는 구조체 입니다.

해당 구조체에는 파일에 대한 열기, 읽기, 쓰기, 닫기에 관련된 모든 상태 정보가 들어 있습니다.

(뒷 페이지 계속)

1. ADC - 코딩(파일 포인터)

- 파일 여는 함수 fopen()

```
FILE *fp;  
fp = fopen("test.txt", "w");
```

좌측과 같은 방식으로 사용 할 수 있습니다.
파일 열기가 실패하면 NULL을 반환 합니다.

Nmae에는 파일의 이름, 경로를 지정한 이름이 들어 갈 수 있습니다.

Mode에는 아래 표와 같이 필요한 설정이 가능 합니다.

모드	설명
"r"	읽기 모드로 파일을 연다.
"w"	쓰기 모드로 파일 생성, 파일이 존재하면 기존 내용이 지워진다.
"a"	추가 모드로 파일을 생성. 파일이 있으면 데이터가 끝에 추가 된다.
"r+"	읽기와 쓰기 모드로 파일을 연다. 파일이 반드시 존재해야한다.
"w+"	읽기와 쓰기 모드로 파일을 생성, 파일이 존재하면 새 데이터가 기존 데이터를 덮어쓴다.
"a+"	읽기와 추가 모드로 파일을 연다. 파일이 존재하면 데이터가 파일 끝에 추가된다. 읽기는 어떤 위치에서나 가능
"b+"	이진 파일 모드로 파일을 연다.

모드가 굉장히 다양합니다.

1. ADC - 코딩(파일 포인터)

- 파일 닫는 함수 `fclose()`
 - ↳ 앞서 설명에 나오듯 파일을 열었으면 항상 작업을 마치고 닫아 줘야 합니다.
역시 `stdio.h`에 정의 되어 있습니다.
`int fclose(FILE* stream)`
정상적으로 파일을 닫는 경우 0을 반환, 실패 할 경우 -1을 반환

```
FILE *fp;  
fp = fopen("test.txt", "w");  
fclose(fp);
```

좌측과 같이 `fopen` 후에 `fclose`를 꼭 해주어야 합니다.

```
#include<stdio.h>  
  
int main()  
{  
    FILE *fp;  
  
    fp = fopen("test.txt", "w");  
  
    if(fp == NULL){  
        printf("파일열기 실패\n");  
    } else {  
        printf("파일열기 성공\n");  
    }  
  
    fclose(fp);  
  
    return 0;  
}
```

`Text.txt` 파일을 쓰기모드로 여는 예제 입니다.

파일이 없으면 파일이 생성 됩니다.

파일은 현재 소스코드 위치에 생성되고,

`Test.txt` 앞에 경로를 지정하면 원하는 곳에서 불러올 수 있습니다.

1. ADC - 코딩(파일 포인터)

- 텍스트 파일 읽기/쓰기

종류	입력	출력
문자 단위	int fgetc(FILE *fp)	int fputc(int c, FILE *fp)
문자열 단위	char *fgets(char *buf, int n, FILE *fp)	int fputs(const char *buf, FILE *fp)
타입지정 입 출력	int fscanf(FILE *fp, ...)	int fprintf(FILE *fp, ...)
이진 데이터	fread(char *buf, int size, int count, FILE *fp)	fwrite(char *buf, int size, int count, FILE *fp)

표준 입출력과 거의 비슷한데,
앞에 f가 붙어서 파일임을 나타내 줍니다.
그리고 매개변수에 **파일 포인터**가
추가 되어 있습니다.

- 문자 단위 입/출력

```
#include<stdio.h>

int main()
{
    FILE *fp;

    fp = fopen("test.txt", "w");

    if(fp == NULL){
        printf("파일열기 실패\n");
    } else {
        printf("파일열기 성공\n");
    }

    fputc('a', fp);
    fputc('b', fp);
    fputc('x', fp);

    fclose(fp);

    return 0;
}
```

결과 값 :

파일열기 성공

test.txt 파일 값 :

abx

포기하면 얻는 건 아무것도 없다.

1. ADC - 코딩(파일 포인

터)

- 파일 내용 읽어 오기

```
#include<stdio.h>

int main()
{
    FILE *fp;
    int c;

    fp = fopen("test.txt", "w");

    if(fp == NULL){
        printf("파일열기 실패\n");
    } else {
        printf("파일열기 성공\n");
    }

    while((c = fgetc(fp)) != EOF){
        putchar(c);
    }

    fclose(fp);

    return 0;
}
```

결과 값 :

```
파일열기 성공
abx
```

이전에 저장한 문자들이 불러와 집니다.

좀 특이한 부분이 있는데, 바로 while문의 EOF 입니다.

EOF란 End of File의 약자로 파일의 끝을 나타내는 특수 문자 입니다.

즉, fgetc를 통해서 파일 포인터가 이동하며 최종적으로 EOF에 도달하면

반복문에서 빠져나와 파일 내용 전부를 출력하는 형태 입니다.

1. ADC - 코딩(파일 포인터)

파일 내용 읽어 오기(문자열 단위 입출력)

```
#include<stdio.h>
#include<memory.h>
#include<string.h>

int main()
{
    FILE *fp;
    char file_buff[100];
    fp = fopen("test_line.txt", "w");

    if (fp == NULL) printf("파일열기 실패\n");
    else printf("파일열기 성공\n");

    int i;

    for (i = 1; i <= 5; i++) {
        printf("파일에 적을 내용을 입력하세요 (%d번째 라인)\n", i);
        memset(file_buff, 0, sizeof(file_buff));
        scanf("%s", file_buff);
        file_buff[strlen(file_buff)] = '\n';

        fputs(file_buff, fp);
    }
    fclose(fp);

    fp = fopen("test_line.txt", "r");

    if (fp == NULL) printf("파일열기 실패\n");
    else printf("파일열기 성공\n");

    while(fgets(file_buff, sizeof(file_buff), fp) != NULL){
        printf("%s", file_buff);
        memset(file_buff, 0, sizeof(file_buff));
    }
    fclose(fp);

    return 0;
}
```

결과값 :

```
파일열기 성공
파일에 적을 내용을 입력하세요 (1번째 라인)
1번째라인이다
파일에 적을 내용을 입력하세요 (2번째 라인)
2번째라인이다
파일에 적을 내용을 입력하세요 (3번째 라인)
3번째라인이다
파일에 적을 내용을 입력하세요 (4번째 라인)
4번째라인이다
파일에 적을 내용을 입력하세요 (5번째 라인)
5번째라인이다
파일열기 성공
1번째라인이다
2번째라인이다
3번째라인이다
4번째라인이다
5번째라인이다
```

몇가지 추가 설명을 하자면,

Memset 함수를 이용해서 file buf에 있는 값을 0으로 초기화 하는작업을 반복 했습니다.

Memset의 경우 memory.h 헤더파일에 포함된 함수 입니다.

File_buff[strlen(file_buff)] = “\n”, 의 경우

File buf에 줄바꿈 문자인 \n을 마지막 라인에 포함하여 해당 버퍼를 Fputs 을 이용해서 파일에 쓰도록 하였습니다.

1. ADC - 코딩(파일 포인터)

- 파일 지정 입출력

```
#include<stdio.h>

int main()
{
    FILE *fp;
    char name[20];
    int age;
    double height;
    double ave_age = 0;

    if ((fp = fopen("user_info.txt", "w")) == NULL) {
        printf("파일열기 실패\n");
    }

    int i;
    for (i = 0; i < 5; i++) {
        printf("%d번째 유저정보를 입력 (나이, 이름, 키 순): ", i);
        scanf("%d %s %lf", &age, name, &height);

        fprintf(fp, "%d %s %lf\n", age, name, height);
    }
    fclose(fp);

    if ((fp = fopen("user_info.txt", "r")) == NULL) {
        printf("파일열기 실패\n");
    }

    for (i = 0; i < 5; i++) {
        fscanf(fp, "%d %s %lf", &age, name, &height);
        ave_age += (double)age;
    }

    printf("유저들의 평균나이 : %.2lf\n", ave_age / 5);
    fclose(fp);

    return 0;
}
```

결과값 :

```
0번째 유저정보를 입력 (나이, 이름, 키 순) : 24 PARK 175.5
1번째 유저정보를 입력 (나이, 이름, 키 순) : 27 JIN 178.7
2번째 유저정보를 입력 (나이, 이름, 키 순) : 31 JUNG 180.54
3번째 유저정보를 입력 (나이, 이름, 키 순) : 25 CHOI 165.31
4번째 유저정보를 입력 (나이, 이름, 키 순) : 29 KIM 169.8
유저들의 평균나이 : 27.20
```

user_info.txt 파일 :

```
24 PARK 175.500000
27 JIN 178.700000
31 JUNG 180.540000
25 CHOI 165.310000
29 KIM 169.800000
```

지금 까지는 파일에 문자데이터를 읽고 쓰는 법만 했었고,
다양한 자료형(int, float, double 등)은 쓸 수 없는 걸까?
Printf와 scanf를 활용해서 좌측 예제와 같이
활용이 가능하다.

1. ADC - 코딩(파일 포인

터)

- 이진 파일 읽기/쓰기

이진 파일 읽기/쓰기를 하기 전에 이진파일은 정확히 무엇 일까요?

이진 파일은 말 그대로(Binary)로 이루어진 파일 입니다. 텍스트가 아닌 0과 1로 구성된 데이터 입니다.

하지만 이것만으로 우리 눈으로는 무슨 내용인지 파악하기가 어렵습니다.

그런데 무슨 이점이 있길래 이진파일 읽기/쓰기를 지원하는 것 일까요??

예를 들어 12라는 정수 데이터가 있다고 하면

텍스트 파일의 경우 모두 문자열로 바꾸어서 저장을 합니다.

반면 이진파일의 경우 12라는 정수 데이터를 그대로 저장합니다. **속도면에서 굉장히 유리하죠**

1byte로 표현하면 0000 1100 으로 구성됩니다.

컴퓨터의 계산기를 이용해서 12를 이진수 형태로 봅시다.

12			
0000	0000	0000	0000
47			32
0000	0000	0000	1100
15			0

문자열로 표현하면 2 byte 가량이 필요한데,

는 1100 4bit만 있으면 12가 표현이 가능합니다.

저장공간 면에서도 유리 합니다.

, 데이터 크기와 속도가 중요한 경우 이진파일을 사용하고는 합니다.

1. ADC - 코딩(파일 포인터)

이진 파일 읽기/쓰기(예제)

```
#include<stdio.h>

int main()
{

    int buff[] = { 10, 20, 30, 40 ,50 };
    int output[6];
    FILE *fp;
    int size, count;

    fp = fopen("binary.bin", "wb");
    if (fp == NULL) printf("파일 열기 실패\n");

    size = sizeof(int);
    count = sizeof(buff) / sizeof(int);

    fwrite(&buff, size, count, fp);
    fclose(fp);

    fp = fopen("binary.bin", "rb");
    if (fp == NULL) printf("파일 열기 실패\n");
    int i;
    for (i = 0; i < count; i++) {
        fread(&output[i], size, 1, fp);
        printf("%d ", output[i]);
    }
    fclose(fp);

    return 0;
}
```

결과값:

10 20 30 40 50

정수형 배열을 **fwrite** 를 통해서 이진 데이터로 저장 할 수 있다.

Size 값은 한번에 저장할 항목의 크기이고,
카운트는 저장할 값의 전체 크기 입니다.

즉, 배열 한 인덱스의 크기인 int 크기와
카운트는 전체 배열의 크기 이다.

실제로 bin 파일을 메모장에서 열어보면
알수 없는 이상한 값들이 담겨져 있습니다.

이 데이터들은 **fread**를 통해서 읽어 올 수 있습니다.

1. ADC - 코딩

```
main.c x uart.h x uart.c x
1  #define F_CPU 16000000UL
2  #include "uart.h"
3
4  #define sbi(PORTX, BITX) (PORTX |= (1 << BITX))
5  #define cbi(PORTX, BITX) (PORTX |= ~(1 << BITX))
6
7  #define UART_BUF 10
8
9  void uart_init(void)
10 {
11     sbi(UCSR0A, U2X0);
12
13     UBRROH = 0x00;
14     UBRROL = 207;
15
16     UCSROC |= 0x06;
17
18     sbi(UCSR0B, RXEN0);
19     sbi(UCSR0B, TXEN0);
20 }
21
22 unsigned char uart_receive(void)
23 {
24     while(!(UCSR0A & (1 << RXC0)))
25     {
26     }
27
28     return UDR0;
29 }
30
```

```
30
31 void uart_transmit(char data)
32 {
33     while(!(UCSR0A & (1 << UDRE0)))
34     {
35     }
36
37     UDR0 = data;
38 }
39
40 void uart_string_transmit(char *string)
41 {
42     while(*string != '\0')
43     {
44         uart_transmit(*string);
45         string++;
46     }
47 }
48
```

UART는 앞서 정리한 내용이 있기때문에 간단하게 정리 후 넘어 가도록 하겠습니다.

- Baudrate 9600
- UDR0에 Receive 받을 데이터가 있을 시 수신(polling)
- Transmit 송신 준비가 되었을 때 데이터를 UDR0에 송신(polling)
- Uart_string_transmit 함수는 문자열의 끝인 NULL이 들어 올 때 까지 문자 출력.

1. ADC - 코딩

```
30
31 void uart_transmit(char data)
32 {
33     while(!(UCSRA & (1 << UDRE0))){
34         ;
35     }
36
37     UDR0 = data;
38 }
39
40 void uart_string_transmit(char *string)
41 {
42     while(*string != '\0')
43     {
44         uart_transmit(*string);
45         string++;
46     }
47 }
48
49 void uart_print(char *name, long val)
50 {
51     char debug_buffer[UART_BUF] = {'\0'};
52
53     uart_string_transmit(name);
54     uart_string_transmit(" = ");
55
56     itoa((val), debug_buffer, UART_BUF);
57     uart_string_transmit(debug_buffer);
58     uart_string_transmit("\n");
59 }
60
```

- uart_print 함수
 - ↳ 디버그용 버퍼가 있는 걸 보아 디버그용 함수 인듯 하다.
 - ↳ 'name = '을 transmit 함수로 출력
 - ↳ itoa 함수를 활용한 int 형을 아스키로 변환

```
char * itoa(int val, char * buf, int radix);
```

itoa 함수의 원형은 위와 같으며 자세한 정보는 아래와 같습니다.

- 필요 헤더: `stdlib.h`
- 파라미터:
 1. 변환할 정수
 2. 변환받을 문자열을 저장할 문자열 변수
 3. 변환할 진수 (10을 입력하면 10진수, 2를 입력하면 2진수)
- 반환값: 변환된 문자열의 주소 (2번째 파라미터인 buf의 주소)
- 기타: 비표준함수 (마이크로소프트 VS에서만 이용 가능)

Itoa는 마이크로소프트 비주얼스튜디오에서만 정의해 놓은 함수이다. 따라서 우분투 등 다른 플랫폼에서 개발할 경우 사용 할 수 없을 것.

- debug_buffer 내용 전달
- 다음줄 넘어가는 “\n” 출력.

1. ADC - 코딩

```
61 int usart_tx_char(char ch, FILE *fp)
62 {
63     while(!(UCSROA & (1 << UDRE0)))
64     {
65         ;
66     }
67
68     UDR0 = ch;
69     return 0;
70 }
71
```

- usart_tx_char 함수

어떤 문자를 File 포인터가 가르키는 곳으로 송신 하기 위한 함수 인듯 하다.

송신 문자를 UDR0에 넣어서 전송.

1. ADC - 코딩

*main.c uart.h uart.c

```
1  #define F_CPU 16000000UL
2
3  #include <avr/io.h>
4  #include <util/delay.h>
5  #include <stdio.h>
6
7  #include "uart.h"
8  #include <avr/io.h>
9
10 // ADC 값이 10 bit 이기 때문에 putty 상에서 최대 1024 정도의 값이 뜬다.
11
12 void adc_init(unsigned char channel)
13 {
14     // AVCC 기준 전압, ADC 전압을 위한 기준 전압.
15     ADMUX |= 0x40;
16
17     // 분주비, 16000000 / 128 = 125000 -> 1/125000 = 8 us
18     ADCSRA |= 0x07;
19
20     // ADC 활성화
21     ADCSRA |= (1 << ADEN);
22
23     // 자동 트리거 set, 여기서 상승 엣지, 8us 단위로(위에서 설정된)
24     ADCSRA |= (1 << ADSCF);
25
26     // 채널 선택, 채널 0, 내부 모드 사용해서 기준이 1.1 v 가 되어서 실제로는 1.1~5v를 ADC 하게 됨.
27     ADMUX |= ((ADMUX & 0xE0) | channel);
28
29     // 컨버트 시작 !
30     ADCSRA |= (1 << ADSC);
31
32 }
```

- adc_init 함수

이제 본격적으로 ADC 초기화 관련 코드를 살펴 보도록 한다.

우선 기존 필요 및 코딩 했던 uart 헤더를 선언 한다.

- ADMUX

↳ AVCC 기준 전압, ADC 전압을 위한 기준 전압.

ADMUX – ADC Multiplexer Selection Register

Bit (0x7C)	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	ADMUX
Initial Value	0	0	0	0	0	0	0	0	

MUX3..0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4

1. ADC - 코딩

*main.c uart.h uart.c

```
1  #define F_CPU 16000000UL
2
3  #include <avr/io.h>
4  #include <util/delay.h>
5  #include <stdio.h>
6
7  #include "uart.h"
8  #include <avr/io.h>
9
10 // ADC 값이 10 bit 이기 때문에 putty 상에서 최대 1024 정도의 값이 뜬다.
11
12 void adc_init(unsigned char channel)
13 {
14     // AVCC 기준 전압, ADC 전압을 위한 기준 전압.
15     ADMUX |= 0x40;
16
17     // 분주비, 16000000 / 128 = 125000 -> 1/125000 = 8 us
18     ADCSRA |= 0x07;
19
20     // ADC 활성화
21     ADCSRA |= (1 << ADEN);
22
23     // 자동 트리거 set, 여기서 상승 엣지, 8us 단위로(위에서 설정된)
24     ADCSRA |= (1 << ADSC);
25
26     // 채널 선택, 채널 0, 내부 모드 사용해서 기준이 1.1 v 가 되어서 실제로는 1.1~5v를 ADC 하게 됨.
27     ADMUX |= ((ADMUX & 0xE0) | channel);
28
29     // 컨버트 시작 !
30     ADCSRA |= (1 << ADSC);
31
32 }
```

ADCSRA – ADC Control and Status Register A

Bit (0x7A)	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	ADCSRA
Initial Value	0	0	0	0	0	0	0	0	

- ADCSRA
 - ADC 샘플링 분주비 설정

Table 23-5. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
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

- ADCSRA
 - ADC 활성화(Enable)

• Bit 7 – ADEN: ADC Enable

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

1. ADC - 코딩

*main.c uart.h uart.c

```
1  #define F_CPU 16000000UL
2
3  #include <avr/io.h>
4  #include <util/delay.h>
5  #include <stdio.h>
6
7  #include "uart.h"
8  #include <avr/io.h>
9
10 // ADC 값이 10 bit 이기 때문에 putty 상에서 최대 1024 정도의 값이 뜬다.
```

```
12 void adc_init(unsigned char channel)
```

```
13 {
14     // AVCC 기준 전압, ADC 전압을 위한 기준 전압.
15     ADMUX |= 0x40;
16
17     // 분주비, 16000000 / 128 = 125000 -> 1/125000 = 8 us
18     ADCSRA |= 0x07;
19
20     // ADC 활성화
21     ADCSRA |= (1 << ADEN);
22
23     // 자동 트리거 set, 여기서 상승 엣지, 8us 단위로(위에서 설정된)
24     ADCSRA |= (1 << ADSCF);
25
26     // 채널 선택, 채널 0, 내부 모드 사용해서 기준이 1.1 v 가 되어서 실제로는 1.1~5v를 ADC 하게 됨.
27     ADMUX |= ((ADMUX & 0xE0) | channel);
28
29     // 컨버트 시작 !
30     ADCSRA |= (1 << ADSC);
31
32     ADC 컨버트 시작.
```

ADMUX – ADC Multiplexer Selection Register

Bit (0x7C)	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 23-3. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, internal V_{REF} turned off
0	1	AV_{CC} with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V voltage reference with external capacitor at AREF pin

내부 모드 사용해서 기준이 1.1 V가 되어
실제로는 1.1~5V를 ADC 하게 된다.

• Bit 6 – ADSC: ADC Start Conversion

In single conversion mode, write this bit to one to start each conversion. In free running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

1. ADC - 코딩

```
34 int read_adc(void)
35 {
36     //ADC 변환이 완료 되었나요? 완료 됐으면 while loop 통과, polling 방식.
37     while(!(ADCSRA & (1 << ADIF))) // ADC 인터럽트 플래그, 우리는 인터럽트는 이전에 활성화 시키진 않았다.(배터리 체크 정도) 인터럽트는 보통 ADC를 한 이후 바로 가져와야 할 정도 일 때 사용(ex: 레이더 시스템)
38     {
39     };
40 }
41
42 return ADC;
43 }
```

ADC 변환이 완료 됐을 경우 ADIF는 1의 값이 되고, while 문을 빠져 나온 뒤, ADC 값을 반환한다.

```
44
45 void int_to_string(int n, char *buffer)
46 {
47     sprintf(buffer, "%04d", n); // sprintf가 여기서 딱이다. atoi나 이런것들 중에 하나가 막혀 있음., 버퍼에다가 n 이라는 숫자 값을 버퍼화 시켜 주세요.
48     buffer[4] = '\0';
49 }
```

Sprintf 를 간단히 이해 해보자.

• Bit 4 – ADIF: ADC Interrupt Flag

This bit is set when an ADC conversion completes and the data registers are updated. The ADC conversion complete interrupt is executed if the ADIE bit and the I-bit in SREG are set. ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag. Beware that if doing a read-modify-write on ADCSRA, a pending interrupt can be disabled. This also applies if the SBI and CBI instructions are used.

함수 원형

```
int sprintf( char *buffer, const char *format [, argument] ... );
```

오른쪽은 sprintf의 사용 예시 이다.

Sprintf은 I 값을 buf에 복사하고 가운데 항의 형식으로 출력한다.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

void main(void)
{
    int i=5;
    char buff[100];

    printf("buff 에 복사 전 : i의 값은 %d\n",i);
    // i의 값은 5를 출력
    sprintf(buff,"i의 값은 %d\n",i);// i의 값은 5를 buff에 복사
    printf("buff 에 복사 후 : %s",buff);
    //물론 똑같은걸 복사했으니까 출력은 i의 값은 5가 출력

    getch();
}
```

갔다.

1. ADC - 코딩

```
52 int main(void)
53 {
54     int read;
55     char buffer[5];
56     // Insert code
57
58     uart_init();
59     adc_init(0);
60
61     while(1)
62     {
63         read = read_adc();
64         int_to_string(read, buffer);
65         uart_string_transmit(buffer);
66         uart_string_transmit("\n");
67         uart_string_transmit("\r");
68
69         _delay_ms(1000);
70     }
71
72     return 0;
73 }
74
```

이제 마지막으로 위의 함수들을 이용해서 main 문에서 실행 시켜 보도록 합니다.

Int 형 변수 read
Char 배열[5] buffer

urat 초기화
Adc 채널 0 번 초기화

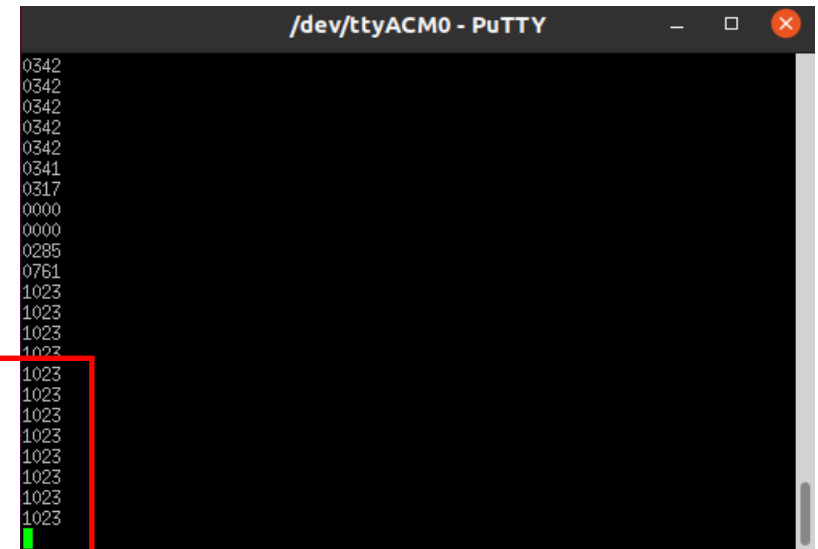
Read_adc 함수를 통해 현재 adc 값을 read 변수에 저장
Int_to_string 함수를 통해 int 형 read 상수를 문자 배열 buffer에 저장

Uart_string_transmit 함수를 통해 buffer에 들어 있는 문자열을 출력

커서 다음 줄 명령어 출력
커서 제일 앞으로 명령어 출력

딜레이 함수(1초)

결과 창



ADC가 10bit 샘플링 방식이므로
값의 최대값이 1023이 되는 것
을 확인 할 수 있다.

Q. 질문

```
main.c x uart.h x uart.c x
1  #ifndef __UART_H__    // 헤더를 중복해서 선언하지 않게 하기 위함
2  #define __UART_H__    // 언더바 두개 이상은 건드리지 말라는 의미 이기도 함
3
4  #include <avr/io.h>
5  #include <util/delay.h>
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <string.h>
9
10 void uart_init(void);
11 unsigned char uart_receive(void);
12 void uart_transmit(char data);
13 void uart_string_transmit(char *string);
14 void uart_print(char *name, long val);
15 int usart_tx_char(char ch, FILE *fp); // 장치가 FILE로 관리됨. 터미널 창에서 /dev 로 확인.
16
17 #endif // __UART_H__
18 |
```

```
char * itoa(int val, char * buf, int radix);
```

질문1)

Uart_string 함수는 인자를 char* 형태로 받는 것에 비해

Usart_tx_char는 인자를 FILE * 로 받는 이유가 있을 까요??

질문2)

Itoa는 마이크로소프트 비주얼스튜디오에서만 정의해 놓은 함수이다
·
따라서 우분투 등 다른 플랫폼에서 개발할 경우 사용 할 수 없을 것
·

이라고 하는데 이 것 대신 다른 것을 사용하는 방법이 있나요??

Q. 질문

ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• Bit 5 – ADLAR: ADC Left Adjust Result

The ADLAR bit affects the presentation of the ADC conversion result in the ADC data register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC data register immediately, regardless of any ongoing conversions. For a complete description of this bit, see [Section 23.9.3 “ADCL and ADCH – The ADC Data Register”](#) on page 219.

```
// 채널 선택, 채널 0 , 내부 모드 사용해서 기준이 1.1 v 가 되어서 실제로는 1.1~5v를 ADC 하게 됨.  
ADMUX |= ((ADMUX & 0xE0) | channel);
```

질문3)

여기에서 비트5가 ADLAR 이라는 레지스터 인데,

이것을 설정 하였을 경우 ADC 값에 영향을 준다고 하는데

어떤 영향을 주는 건지 잘 모르겠습니다..

그리고 이것을 설정한 이유를 잘 모르겠습니다!