C basic language

임베디드스쿨 2기

Lv1과정

2021. 7. 9

김효창

Analog to Digital Converter

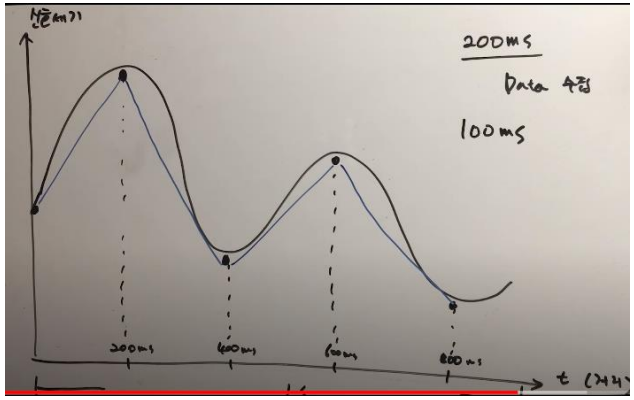
1초 내에 발생한 아날로그 신호

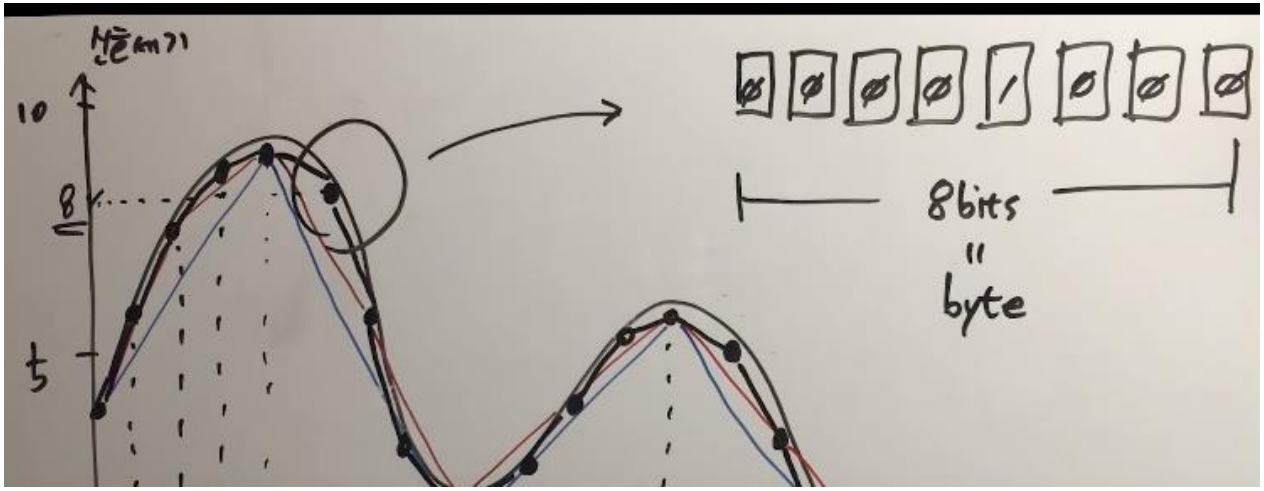
아날로그 신호를 분석해서 디지털 신호로 바꾸는 작업

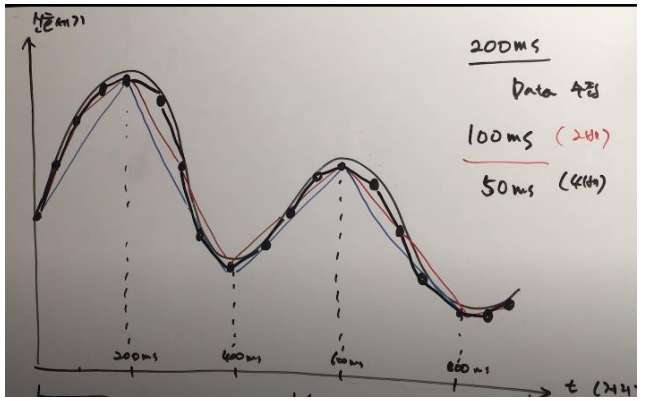
시스템은 200 ms 마다 Data 를 수집 1초에 5번 (왜곡이 있다)
100 ms 마다 Data 를 수집 1초에 10번
50 ms 마다 Data 를 수집 1초에 20번 (아날로그와 비슷)

시간이 짧을수록 정밀도가 높다.

컴퓨터가 데이터를 저장하는 최소 단위 : Byte







Analog to Digital Converter

ADC 사용하려면
디지털로 바꾸려는 아날로그 값의 범위 ?
디지털로 바꿀 때, 몇 개의 구간으로 분할?
얼마나 자주 바꿀 것인가?

아날로그 값의 범위
0 ~ V_{CC} , 0 ~ V_{REF} , 0 ~ 2.56V 등등

몇 개의 구간으로 분할?
분해능 : 전압을 얼마의 크기로 분해할 것인가
ATmega328 → 10 Bit Resolution → 1024 개 구간으로 분할 (0 ~ 1023)

얼마나 자주 바꿀 것인가?
Single Conversion : 1번 변환
Free Running : 계속 변환

Analog to Digital Converter

```
int main(void)
{
	int read;
	char buffer[5];
	uart_init();
	adc_init(0); // ADMUX |= ((ADMUX & 0xE0) | channel); A0 Pin 을 입력으로 사용


	while(1)
	{
		read = read_adc();
		int_to_string(read, buffer);
		uart_string_transmit(buffer);
		uart_string_transmit("₩n");  다음 줄로 이동
		uart_string_transmit("₩r");   해당 줄 맨 앞으로 이동
		_delay_ms(1000);
	}

	return 0;
}
```

sprintf( ) 함수

출력 값을 문자열에 저장하는 함수
통신을 하거나 buffer 에 원하는 문자열을 삽입, 이어붙이기 할 때 사용

buffer : 출력 값을 저장할 문자열
%04d : 4자리 크기로 할당 , 빈 공간은 0 으로 할당
n : 10 Bit ADC 값

ADC값을 0으로 채워진 4자리 문자열로 변환해서 buffer 에 저장
문자열은 항상 NULL 로 끝나야 한다.
buffer[0] = '1'
buffer[1] = '0'
buffer[2] = '2'
buffer[3] = '3'
buffer[4] = ' \0 '

```
void int_to_string(int n, char *buffer)
{
	sprintf(buffer, "%04d", n);
	buffer[4] = ' \0 ';
}
```

buffer 에 저장된 문자열을 uart 를 이용하여 putty 에 표시

Analog to Digital Converter

```
void adc_init(unsigned char channel)
{
	ADMUX |= 0x40;
	ADCSRA |= 0x07;
	ADCSRA |= (1 << ADEN);
	ADCSRA |= (1 << ADATE);
	ADMUX |= ((ADMUX & 0xE0) | channel);
	ADCSRA |= (1 << ADSC);
}

int read_adc(void)
{
	while (!(ADCSRA & (1 << ADIF))); // 변환 종료 대기
	return ADC; // 10 Bit 값을 반환
}
```

- ADC 활성화
- 아날로그 값을 읽을 PIN 설정
- 기준 전압 설정
- ADC 변환 주기 설정 (분주비)
- AD 변환 결과 값 정렬 방식 설정
- ADC 변환 시작

AD 변환 처음에는 25클럭 , 이후에는 13클럭

128분주:16 MHz/128 →125 kHz → 8 us
8us × 13 cycle = 104us → 9600 Hz
125 kHz / 13 cycle = 9615.385
13cycle당 1회 샘플링 - 약 9.6 kHz

폴링 polling 방식 :ADIF 를 읽어서 flag 뜰 때 까지 대기
ADIF = 1 되면, while 탈출
ADIF 재활용 하기 위해서는 Clear 해야 한다.
ADIF 플래그는 AD Conversion 이 완료된 후 ADCL/ADCH 레지스터에 값이 업데이트 되면 SET 된다.
read_adc 함수는 해당 레지스터가 set 될 때까지 대기 후 값을 반환한다.
2개의 Low/High 레지스터가 사용되기 때문에 Low 를 먼저 읽고 High 를 읽는다.
반대로 접근하면 Low 값이 Clear 된다.

Analog to Digital Converter

변환 시간 동안 아날로그 입력전압을 일정하게 하는 Sample/Hold회로를 내장 하고 있다.

Single Ended : 0 ~ VREF의 입력전압이 공급되어 0x000~0x3FF(0~1023)의 디지털 값으로 변환된다.
Differential Channel : 2의 보수 0x200~0x1FF(-512~511)로 변환된다.

변환결과는 16비트 A/D변환기 데이터 레지스터에 좌측 또는 우측으로 정렬되어 저장된다.

ADCSRA의 ADEN(ADC Enable 비트)을 Set 하면 활성화 된다.
ADEN이 Clear 되면 ADC는 비 활성화 되어 전력을 소비하지 않는다.

입력전압 = GND → 0 , 입력전압 = AV_{CC} → $2^{10} - 1 = 1023$

AV_{CC} : ADC pin 동작을 전체적으로 관리하는 전압
V_{REF} : ADC 변환 시 기준으로 사용하는 전압
AV_{CC} 사용(V_{REF}에 세라믹 콘덴서 연결 권장) ,
AVR에 내장된 1.1V 정전압을 사용,
외부 전원을 V_{REF} 에 인가하여 사용

$$ADC = \frac{V_{IN} \times 1024}{V_{REF}}$$

$$V_{IN} = \frac{ADC \times V_{REF}}{1024}$$

Features

- 10-bit resolution
- 0.5 LSB integral non-linearity
- ±2 LSB absolute accuracy
- 65 to 260µs conversion time
- Up to 15kSPS
- 6 multiplexed single ended input channels
- 2 additional multiplexed single ended input channels
- Temperature sensor input channel
- Optional left adjustment for ADC result readout
- 0 to V_{CC} ADC input voltage range
- Selectable 1.1V ADC reference voltage
- Free running or single conversion mode
- Interrupt on ADC conversion complete
- Sleep mode noise canceler

Analog to Digital Converter

높은 Clock frequency를 선택 하면 변환 속도는 빨라지고 정확도는 떨어지기 때문에 변환 속도와 정확도를 고려 하여 적당한 주파수를 선택한다.

ADC가 켜진 후 첫 번째 변환은 아날로그 회로를 초기화하기 위해 25 ADC 클럭 사이클이 필요하다.
정상 상태에서의 변환은 13 ADC 클록 사이클을 필요하다.

A/D 변환이 완료 되면 결과가 ADC Data register에 Write 되고 ADIF가 Set 된다.

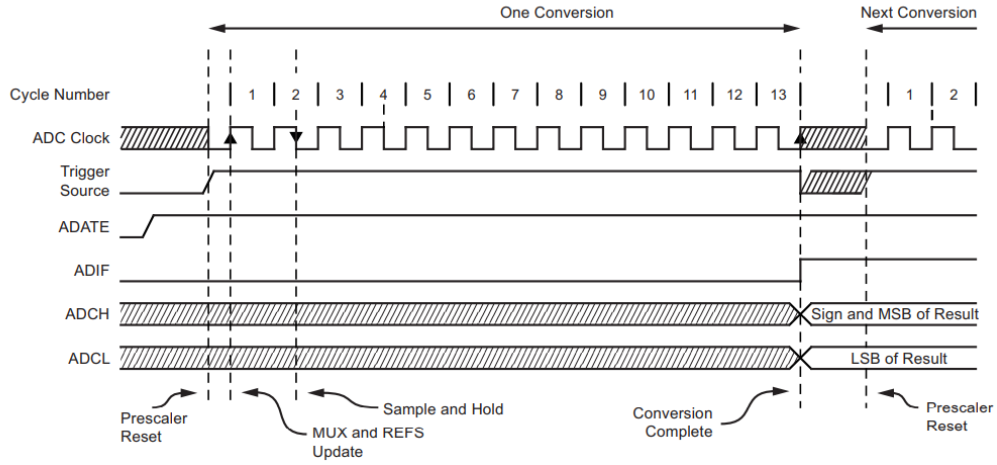
Single conversion mode에서는 변환이 완료 되면 ADSC는 Clear 된다.
다시 A/D 변환을 하고자 하는 경우에는 ADSC 를 Set 하여야 한다.

Free Running mode에서는 변환이 한번 시작 되면 A/D 변환을 계속 한다.
A/D 변환을 정지 하고자 하는 경우에는 하는 경우에는 ADSC 를 Clear 해야 한다.

Table 23-1. ADC Conversion Time

Condition	Sample and Hold (Cycles from Start of Conversion)	Conversion Time (Cycles)
First conversion	13.5	25
Normal conversions, single ended	1.5	13
Auto triggered conversions	2	13.5

Figure 23-6. ADC Timing Diagram, Auto Triggered Conversion



Analog to Digital Converter

ADMUX – ADC Multiplexer Selection Register
Bit 7:6 – REFS1:0 : 기준 전압 설정
ADMUX |= 0x40;
AVCC 를 기준 전압으로 설정
기준 전압을 어떤 것으로 선택 (AREF/AVCC)

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

Bit 5 – ADLAR: AD 변환이 완료된 데이터의 정렬 방식을 지정.
ADMUX |= ((ADMUX & 0xE0) | channel);
1 = 왼쪽 정렬 , 0 = 오른쪽 정렬
사용자가 원하는 채널을 설정
ADC 의 변환 정밀도 10 Bit 이므로
ADCH(8 Bit), ADCL(8 Bit) 두 개의 레지스터로 나누어 읽는다.
ADC 변환 값 10 Bit 모두 사용하는 경우 컴파일러에 의해
ADCL 부터 읽는다.

23.9.3.1 ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

23.9.3.2 ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
(0x79)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
(0x78)	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Bits 3:0 – MUX3:0: ADC 입력 채널 선택
ADMUX |= 0x40;
A0 Pin 을 입력으로 사용 (PORT 지정)
아두이노 Uno 는 Pin 수가 6개로 제한되어 있다. (A0 ~ A5)
ADC8 은 내장된 온도 센서 읽는 용도

Table 23-4. Input Channel Selections

MUX3..0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8(1)
1001	(reserved)
1010	(reserved)
1011	(reserved)
1100	(reserved)
1101	(reserved)
1110	1.1V (V_{BG})
1111	0V (GND)

Note: 1. For temperature sensor.

Analog to Digital Converter

ADCSRA – ADC Control and Status Register A

ADEN : ADC 사용하기 위해 1로 설정
ADSC : ADC 변환 시작 = 1 , 변환 끝나면 = 0
AD 변환이 끝나면 ADSC는 0 으로 Clear 된다.
ADATE : 자동 트리거 모드 , 선택된 트리거 신호에 의해 변환이 시작. 0 = 단일 변환

Bit 4 – ADIF: ADC Interrupt Flag
변환이 완료되고 결과가 Data 레지스터에 Write 되면 Set 된다.
SREG 의 I , ADIE 가 Set 되어 있으면 A/D변환 완료 Interrupt 가 실행 된다.
A/D변환 완료 Interrupt 가 실행 되면 Hardware 적으로 Clear 된다

ADIE : AD 변환 끝났을 때 인터럽트 발생을 허용하는 비트 , SREG 1로 SET 되어 있어야 한다.
ADPS : 변환 속도를 설정 , ADC Clock 을 위한 분주율 설정

Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits
ADC 변환을 빠르게 할까? 천천히 할까? 선택
16MHz 를 얼마의 값으로 나누어서 ADC Clock 을 사용할 것인지 선택.

```
ADCSRA |= 0x07;
```

ADC 의 주파수 16000000/128 = 125kHz

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

Analog to Digital Converter

16MHz 의 ATmega328p 에서 ADC 를 사용하려고 한다.
데이터시트에는 일반적인 ADC 클럭 주파수 범위가 50kHz ~ 200kHz 라고 나와 있다.

최대 해상도를 얻기 위해 50kHz에서 200kHz 사이의 입력 클록 주파수가 필요하다.
10비트보다 낮은 분해능이 필요한 경우, ADC에 대한 입력 클럭 주파수가 200kHz보다 높을 수 있으므로 샘플링 속도를 높일 수 있다.

사전 크기 조정은 ADCSRA의 ADPS 비트에 의해 설정된다.
프리스케일러는 ADCSRA에서 ADEN 비트를 설정하여 ADC가 켜지는 순간부터 계산을 시작한다.

ADC 클럭의 프리스케일러는 2 ~ 128 범위
ADC는 CLK/128부터 CLK/4까지 모든 클럭 주파수에 대해 작동하지만 CLK/2 에서는 작동하지 않는다.

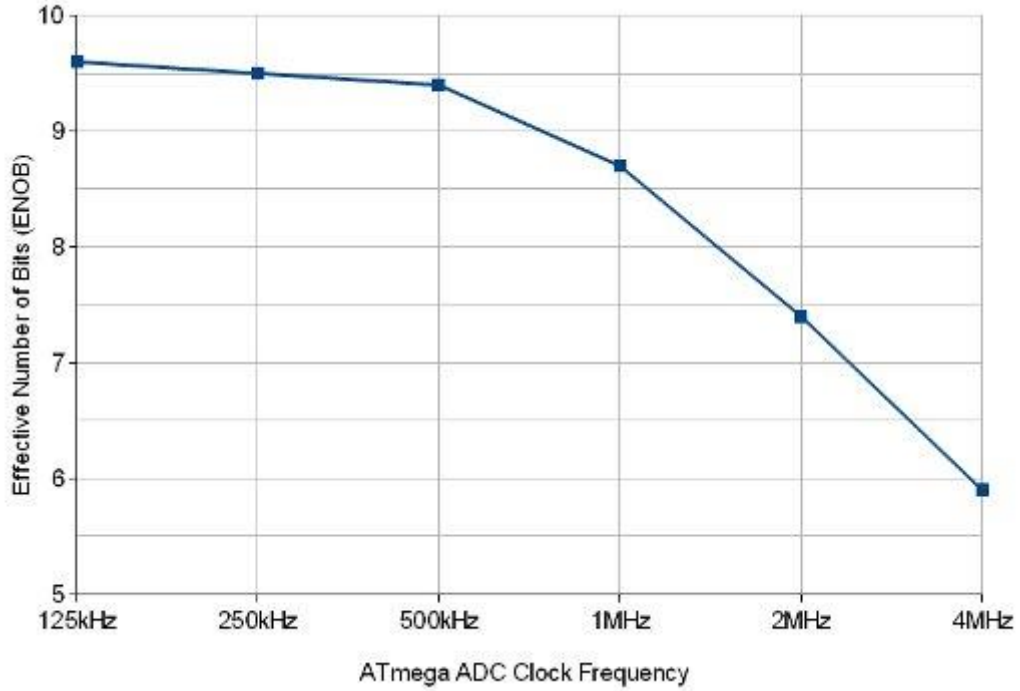
권장되는 최대값을 초과하여 ADC를 실행하고 있기 때문인가? 아니면 알아야 할 다른 사항인가?

데이터 시트에는 ADC 클럭이 1MHz를 초과해서는 안된다고 분명히 나와 있다.
규칙을 어기면 동작이 정의되지 않는다.
작동 할 수도, 작동하지 않을 수도 있고, 온도가 23.7 ° C 이상일 때만 작동 할 수도 있다.
분명히 클럭이 16MHz이면 16보다 낮은 분배기를 사용해서는 안된다.

	Conversion Time	Free Running Conversion	13		260	µs
	Clock Frequency		50		1000	kHz

Analog to Digital Converter

ADC와 클럭 주파수의 유효 분해능은 다음과 같다.

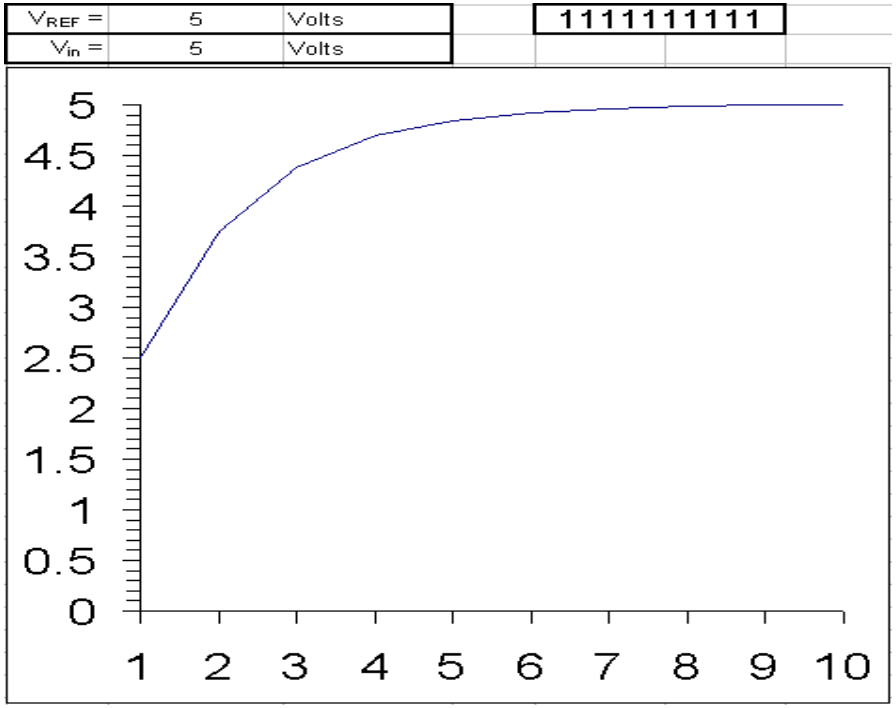


Analog to Digital Converter

Atmega328p 는 successive approximation 축차 비교 방식

The Atmel® ATmega328P features a 10-bit successive approximation ADC.
The ADC converts an analog input voltage to a 10-bit digital value through successive approximation.

축차비교형 ADC는 SAR(Successive Approximation Register)을 사용하여
최상위 비트부터 순서대로 하위 비트 쪽으로 수정하여 가는 방법
변환 중에 아날로그 값(비교 값)이 유지되어야 하므로
샘플/홀드 회로가 필요하다



uart.c

U2X0 : USART Double Transmission Speed
비동기 전송 모드에서만 사용
1 = 2배속 , 0 = 1배속

Table 74. Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

Baud Rate [bps]	f_{osc} = 16.0000MHz			
	U2Xn = 0		U2Xn = 1	
	UBRRn	Error	UBRRn	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	-0.1%
19.2k	51	0.2%	103	0.2%

```c
#define F_CPU 16000000UL
#include "uart.h"

#define sbi(PORTX, BITX) (PORTX |= (1 << BITX))
#define cbi(PORTX, BITX) (PORTX |= ~(1 << BITX))

#define UART_BUF 10

void uart_init(void)
{
    sbi(UCSR0A, U2X0); // 2배속 모드

    UBRR0H = 0x00;
    UBRR0L = 207; // 통신속도 9600 설정

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

    sbi(UCSR0B, RXEN0); // 수신 허용
    sbi(UCSR0B, TXEN0); // 송신 허용
}
```

```c
unsigned char uart_receive(void) // 1 바이트 수신
{
    while(!(UCSR0A & (1 << RXC0))); // 데이터 수신 대기
    return UDR0;
}

void uart_transmit(char data) // 1 바이트 송신
{
    while(!(UCSR0A & (1 << UDRE0))); // 송신 가능 대기
    UDR0 = data;
}
```

uart.c

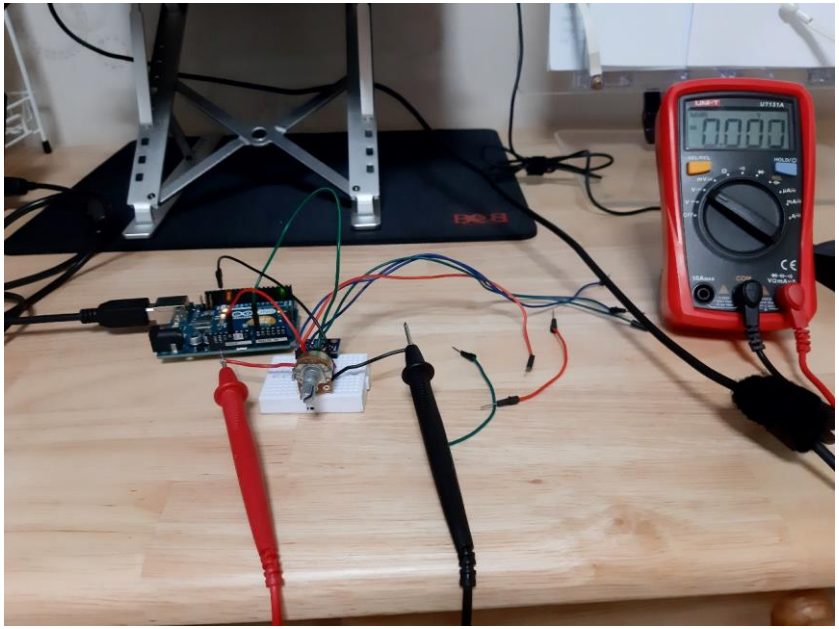
```c
void uart_string_transmit(char *string) // 문자열 송신
{
    while(*string != '\0') // '\0' 만날 때까지 반복
    {
        uart_transmit(*string); // ADC 값 1 Byte 단위로 출력
        string++; // buffer[0] , buffer[1] , buffer[2] , buffer[3], buffer[4] : 5 Byte
    }
}
```

```c
void uart_print(char *name, long val)
{
    char debug_buffer[UART_BUF] = {'\0'}; // debug_buffer[10] -> debug_buffer[0] ~ [9] 까지 초기화

    uart_string_transmit(name);
    uart_string_transmit(" = ");

    /* itoa ( int value , char *str , int radix );
    value 는 변환 시키고 싶은 숫자
    str 은 전환될 문자열
    radix 는 진수를 결정  2 = 2진수 , 10 = 10진수 형태로 출력
    */
    itoa((val), debug_buffer, UART_BUF);  // debug_buffer 를 반환하여 10 Bit ADC 값을 10진수 형태로 출력
    uart_string_transmit(debug_buffer);
    uart_string_transmit("\n");
}
```

Analog to Digital Converter

코드 함수 동작 흐름과 실험 측정

1. void uart_init(void);
   void adc_init(unsigned char channel)
2. 가변저항 돌리기
3. unsigned char uart_receive(void);
4. int read_adc(void)
5. void uart_transmit(char data);
6. void int_to_string(int n, char *buffer)
7. void uart_string_transmit(char *string);
8. void uart_print(char *name, long val);
9. int usart_tx_char(char ch, FILE *fp);



DMM	PUTTY
0 [v]	0000
0.051 [v]	0008
0.188 [v]	0036
0.797 [v]	0158
1.489 [v]	0297
1.875 [v]	0374
2.35 [v]	0470
2.8 [v]	0560
3.35 [v]	0670
3.77 [v]	0755
4.38 [v]	0878
5 [v]	1001
5.12 [v]	1023

질의응답

Q : ADC 실습 Single Conversion 모드로 설정하였는가???
A : ADCSRB를 통해 모드 변경이 가능합니다. 일단 별도의 지정이 없으므로 Free Running이 맞습니다.
뭐가 되었던 준비 시간이 필요하다.

Q : *fp
A : file pointer의 약자

Q : int usart_tx_char(char ch, FILE *fp) 의 char ch 는 putty 에 마지막으로 전달해 주는 것들인가요?
A : 매서드명대로 문자 1개 전송입니다.

Q : usart_tx_char 함수 동작 질문

```
int usart_tx_char(char ch, FILE *fp)
{
    while(!(UCSR0A & (1 << UDRE0)));
    UDR0 = ch;
    return 0;
}
```

A : 내부적인 콜백 함수를 확인해야 한다.
추가로 fdevopen , fpStdio 2 개 분석해야 한다.

Q : char debug_buffer[UART_BUF] = {'₩0'};
문자열의 끝을 함수 시작부터 넣는 이유는?
A : C언어 학습한 배열 생각.
[0] ~ [9] 까지의 배열 초기화

End of Document