-Timer/Counter2 레지스터 구성

Table 19. Reset and Interrupt Vectors

Vector No.	Program Address[2]	Source	Interrupt Definition
1	0x000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog Reset
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	TIMER2 COMP	Timer/Counter2 Compare Match
5	0x004	TIMER2 OVF	Timer/Counter2 Overflow
6	0x005	TIMER1 CAPT	Timer/Counter1 Capture Event
7	0x006	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	0x007	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	0x008	TIMER1 OVF	Timer/Counter1 Overflow
10	0x009	TIMER0 OVF	Timer/Counter0 Overflow
11	0x00A	SPI, STC	Serial Transfer Complete
12	0x00B	USART, RXC	USART, Rx Complete
13	0x00C	USART, UDRE	USART Data Register Empty
14	0x00D	USART, TXC	USART, Tx Complete
15	0x00E	ADC	ADC Conversion Complete
16	0x00F	EE_RDY	EEPROM Ready
17	0x010	ANA_COMP	Analog Comparator
18	0x011	TWI	Two-wire Serial Interface
19	0x012	INT2	External Interrupt Request 2
20	0x013	TIMER0 COMP	Timer/Counter0 Compare Match
21	0x014	SPM_RDY	Store Program Memory Ready

1. 제어 주기 만들 때 Interrupt를 발생하는 순서를 보면 Timer/Counter2가 순서가 빠르므로 Timer/Counter2를 쓴다.

Bit	7	6	5	4	3	2	1	0	
	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	TCCR2
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 51. Waveform Generation Mode Bit Description(1)

Mode	WGM21 (CTC2)	WGM20 (PWM2)	Timer/Counter Mode of Operation	TOP	Update of OCR2	TOV2 Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR2	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

2. Timer/Counter2 레지스터를 구성할 때 저번 Timer/Counter2 실험 할 때처럼 WGM를 Fast PWM모드로 쓴다. 즉 Bit 6,3은 1로 처리한다.

Table 53. Compare Output Mode, Fast PWM Mode(1)

COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Reserved
1	0	Clear OC2 on Compare Match, set OC2 at TOP (Non-Inverting).
1	1	Set OC2 on Compare Match, clear OC2 at TOP (Inverting).

3. 저번 Timer/Counter2 실험이랑 다르게 이번 실험은 OC2를 안 써서 COM모드인 Bit 5와 4를 0으로 처리한다.

Table 55. Clock Select Bit Description

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk_{T2S}/(No prescaling)
0	1	0	clk_{T2S}/8 (From prescaler)
0	1	1	clk_{T2S}/32 (From prescaler)
1	0	0	clk_{T2S}/64 (From prescaler)
1	0	1	clk_{T2S}/128 (From prescaler)
1	1	0	clk_{T2S}/256 (From prescaler)
1	1	1	clk_{T2S}/1024 (From prescaler)

4. 분주비가 제일 중요하다. 우리가 제어할 어떤 동작의 시간(control time)과 제어 주기(control period)를 관여하기 때문이다. 일단 control period가 control time 길어야 control time에서 우리가 구현하는 제어 동작이 작동된다. 일단 분주비는 1024로 설정했다

Timer/Counter Register – TCNT2

Bit	7	6	5	4	3	2	1	0	
	TCNT2[7:0]								TCNT2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Timer/Counter Register gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter. Writing to the TCNT2 Register blocks (removes) the Compare Match on the following timer clock. Modifying the counter (TCNT2) while the counter is running, introduces a risk of missing a Compare Match between TCNT2 and the OCR2 Register.

5. 제어주기 공식은 분주비/16MHz X (TOP-TCNT)이다. 일단 여기서 Uart를 미리 애기하자면 현재 소스에서 bps가 115200 이고 비트 수가 start+data+stop를 포함하면 10bit고 통신하는 개수가 22개이라서 총 합치면 10/115200 x 22 대략 2ms 가 나온다. 적어도 우리는 2ms보다 크게 주기를 만들어야 한다. 여유롭게 TCNT를 100를 잡아서 1024/16MHZ x (256-100)를 해서 대략 10ms 나오게 만들었다.

Timer/Counter Interrupt Mask Register – TIMSK

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 6 – TOIE2: Timer/Counter2 Overflow Interrupt Enable**

When the TOIE2 bit is written to one and the I-bit in the Status Register is set (one), the Timer/Counter2 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter2 occurs (i.e., when the TOV2 bit is set in the Timer/Counter Interrupt Flag Register – TIFR).

6. Overflow를 이용해 Interrupt를 발생시키므로 TIMSK에서 Timer/Counter2 Overflow Interrupt Enable 부분인 bit 6번을 1로 처리한다.

-Timer/Counter2 소스

```
void MCU_Init::Init_Timer2(void)
{
    TCCR2   = (1<<WGM20)|(1<<COM21)|(0<<COM20)|(1<<WGM21)|(1<<CS22)|(1<<CS21)|(1<<CS20);//fast pwm, 1024분주
    TCNT2   = 100;
    OCR2    = 0;
}
void MCU_Init::Init_TIMSK(void)
{
    TIMSK |=(1<<TOIE2); //Timer Interrupt 1 Enable
}
```

-Uart 레지스터 구성

USART Control and Status Register A – UCSRA

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

USART I/O Data Register – UDR

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (Read)
	TXB[7:0]								UDR (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – RXC: USART Receive Complete**

This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). If the Receiver is disabled, the receive buffer will be flushed and consequently the RXC bit will become zero. The RXC Flag can be used to generate a Receive Complete interrupt (see description of the RXCIE bit).

- **Bit 6 – TXC: USART Transmit Complete**

This flag bit is set when the entire frame in the transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer (UDR). The TXC Flag bit is automatically cleared when a Transmit Complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXC Flag can generate a Transmit Complete interrupt (see description of the TXCIE bit).

- **Bit 5 – UDRE: USART Data Register Empty**

The UDRE Flag indicates if the transmit buffer (UDR) is ready to receive new data. If UDRE is one, the buffer is empty, and therefore ready to be written. The UDRE Flag can generate a Data Register Empty interrupt (see description of the UDRIE bit).

UDRE is set after a reset to indicate that the Transmitter is ready.

1. UDR는 값을 넣어 보내주거나 값을 받아오는 역할을 한다.
2. UCSRA로 통해 값을 받거나 보낼 수 있다. UDRE는 UDR의 값이 비워져 있으면 1로 설정되고 값이 들어가져 있으면 0으로 설정된다. Bit 7번은 말 그대로 Uart가 받는 것을 완료하면 1 처리되고 아직 받는 중이면 0으로 처리된다. Bit 6번도 7번과 같은 맥락으로 생각하면 된다. 보내는 것을 완료하면 1, 보내는 중이면 0이다.
3. UCSRA를 쓰는 방법은 datasheet를 보면 예제소스가 있는데 아래 사진들이 그 받는거랑 보내는 예제소스다.

C Code Example(1)

```
unsigned char USART_Receive( void )
{
  /* Wait for data to be received */
  while ( !(UCSRA & (1<<RXC)) )
        ;
  /* Get and return received data from buffer */
  return UDR;
}
```

4. 받는 예제소스를 설명하자면 데이터를 받는 중이라 while 문에서 UCSRA는 대충 0b00000000일 것이고 (1<< RXC)는 0b10000000일 것이다. 둘이 AND로 비교하면 0b00000000일 테니 not 기호로 인해 while(1); 이랑 같은 구문이다. 즉 return UDR로 아직 안 간 상태다. 하지만 데이터가 받을 것을 완료하면 UCSRA, (1<< RXC) 둘 다 0b10000000일 테니 not 기호로 인해 while(0); 즉 return UDR로 가서 받는 값을 도출한다.

C Code Example(1)

```
void USART_Transmit( unsigned int data )
{
  /* Wait for empty transmit buffer */
  while ( !( UCSRA & (1<<UDRE)) )
        ;
  /* Copy 9th bit to TXB8 */
  UCSRB &= ~(1<<TXB8);
  if ( data & 0x0100 )
    UCSRB |= (1<<TXB8);
  /* Put data into buffer, sends the data */
  UDR = data;
}
```

5. 보내는 소스를 설명하자면 어떤 데이터를 매개변수인 unsigned int data에 넣으면 UDR에 값을 들어가게 만들어서 0으로 만들어준다. while문을 보면 UCSRA는 아직 값이 있을 수 있으므로 0b00000000일 것이고 (1<<UDRE)는 0b10000000일 것이다. 둘이 AND로 비교하면 0b00000000일 테니 not 기호로 인해 while(1); 이랑 같은 구문이다. 즉 UDR=data로 아직 안 간 상태다. 만약 앞선 값이 비워지면 UCSRA, (1<<UDRE) 둘 다 0b10000000 일 테니 not 기호로 인해 while(0); 즉 UDR=data로 들어가서 UDR이 값을 보내준다.

USART Control and Status Register B – UCSRB

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 4 – RXEN: Receiver Enable**

Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxD pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FE, DOR, and PE Flags.

- **Bit 3 – TXEN: Transmitter Enable**

Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxD pin when enabled. The disabling of the Transmitter (writing TXEN to zero) will not become effective until ongoing and pending transmissions are completed, (i.e., when the transmit Shift Register and transmit Buffer Register do not contain data to be transmitted). When disabled, the Transmitter will no longer override the TxD port.

- **Bit 2 – UCSZ2: Character Size**

The UCSZ2 bits combined with the UCSZ1:0 bit in UCSRC sets the number of data bits (Character Size) in a frame the Receiver and Transmitter use.

6. UCSRB에서 설정해줘야 할 것은 Bit 3,4번이다. 둘 다 datasheet를 보면 보내는 것과 받는 것을 활성화 시켜주는 비트다. 애네 둘 다 1로 설정해주면 활성화가 된다. Bit2번은 데이트 비트 수 설정하는데 비트데 5~8bit 보낼거면 다 0으로 설정되어있으므로 그냥 0으로 설정하면 된다. Character size는 주로 UCSRC에서 설정해준다.

USART Control and Status Register C – UCSRC(1)

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

- **Bit 6 – UMSEL: USART Mode Select**

This bit selects between asynchronous and synchronous mode of operation.

Table 64. UMSEL Bit Settings

UMSEL	Mode
0	Asynchronous Operation
1	Synchronous Operation

7. UMSEL는 통신 방법을 설정하는 것인데 Asynchronous Operation는 비동기식이고 Synchronous Operation 동기식이다. 간단하게 설명하면 비동기 식은 일반 통행이고 동기는 답을 보내면 그 답에 대해 반응해야한다. 반응을 안 할시 통신이 못 넘어간다. 그러므로 우리는 일반 통행인 비동기로 설정한 다.

Table 65. UPM Bits Settings

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

8. parity 모드는 데이터 비트 수를 검사 하는거라 생각하면 된다. 만약 우리가 짝수를 검사하는 EVEN Paritiy를 설정할 때 만약 비트 수가 홀수로 검출되면 통신에 오류가 생겼다고 알려준다. Odd paritiy는 그 반대인 홀수를 검사하는데 비트 수가 짝수로 검출되면 오류 났다고 알려준다.

Table 66. USBS Bit Settings

USBS	Stop Bit(s)
0	1-bit
1	2-bit

9. 여기서 우리는 stop bit를 1 bit만 쓸거니 0으로 처리해준다.

Table 67. UCSZ Bits Settings

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

10. 앞서 UCSRB에서 얘기한 데이트 비트 수 정하는거다. 우리는 데이터를 8비트 보낼거니 **UCSZ1 UCSZ0**를 둘 다 1로 처리한다.

USART Baud Rate Registers – UBRRL and UBRRH(1)

Bit	15	14	13	12	11	10	9	8	
	URSEL	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Baud Rate (bps)	f_{osc} = 16.0000 MHz			
	U2X = 0		U2X = 1	
	UBRR	Error	UBRR	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%
28.8k	34	-0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	-0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	-3.5%	16	2.1%
230.4k	3	8.5%	8	-3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max (1)	1 Mbps		2 Mbps	

11. 통신속도를 정하는 것인데 우리의 실험보드는 16MHz이므로 datasheet에서 이 표를 보면 된다. 앞에서 Timer/Counter2 레지스터 설 정할 때 통신속도를 115200bps로 설정했다고 했고 255보다 작은 값인 8이므로 UBRRH = 0; UBRRL = 8; 으로 설정해준다.

-Uart 소스

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//UART
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void MCU_Init::Init_UART(void)
{

    //UCSRA = 0x00;
    UCSRB=(1<<RXEN)|(1<<TXEN);   //RX, TX Enable
    UCSRC=0x06;                 //Stop bit 1, Data size 8

    UBRRH=0x00;
    UBRRL=8;                //115200bps
}
void MCU_Init::Uart_Putch(unsigned char data)
{
    while(!(UCSRA &(1<<UDRE)));   //UDR empty flag
    UDR = data;
}
unsigned char MCU_Init::Uart_Getch(void)
{
    while(!(UCSRA &(1<<RXC)));   //RX complete flage

    return UDR;
}
```

-ADC 레지스터 구성

ADC Multiplexer Selection Register – ADMUX

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – REFS1:0: Reference Selection Bits**

These bits select the voltage reference for the ADC, as shown in Table 84. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

Table 84. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

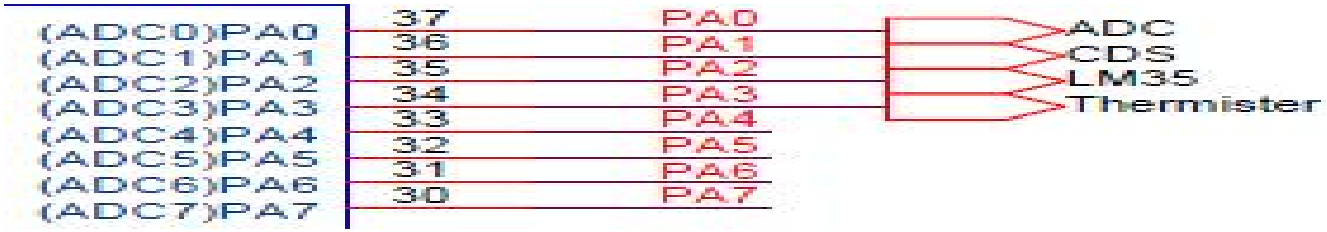
1. 외부 AVCC를 사용하므로 표에서 나온 것처럼 REFS1과 REFS0를 0,1로 설정해준다.

- **Bits 4:0 – MUX4:0: Analog Channel and Gain Selection Bits**

The value of these bits selects which combination of analog inputs are connected to the ADC. These bits also select the gain for the differential channels. See Table 85 for details. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set).

Table 85. Input Channel and Gain Selections

MUX4..0	Single Ended Input	Pos Differential Input	Neg Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			



2. 우리는 ADC를 주로 개별적으로 쓰므로 datasheet에 나오는 위의 표 사진의 나머지 표는 무시하고 이것을 참고하면 된다. 회로도를 보면 MUX 번호를 알려주는 핀이 포트 A이르므로 이 포트에서 센서를 제어하면 된다.

The ADC Data Register – ADCL and ADCH

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	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	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	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	

3. 하위 비트로 ADC 핀을 고르기 때문에 ADLAR는 0으로 설정해준다.

ADC Control and Status Register A – ADCSRA

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **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.

- **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.

- **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.

4. Bit 7번은 ADC를 활성화 시켜주는 포트이므로 1로 설정하면 활성화된다.

5. Bit 6번은 아날로그 신호에서 디지털 신호로 변환해주는 것을 설정하는데 1로 설정하면 각각의 ADC로 변환을 시작한다. 만약 앞선 ADC0
의 변환이 끝나면 그 다음 ADC의 변환이 시작된다.

6. Bit 4번은 ADC가 끝나면 Interrupt를 발생시킨다라는 것으로 1로 설정하면 활성화된다.

Table 86. 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

7. ADPS는 ADC의 분주비를 설정해주는 것이다. 처음에 128로 설정하면 큰 값과 작은 값을 다 볼 수 있지만 노이즈가 심해 작동이 잘 안 될 수 있다. 그래서 처음에 분주비를 크게 잡고 작동이 잘 안되면 분주비를 낮추면 된다.

-ADC 소스

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// ADC
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void MCU_Init::Init_ADC(void)
{
    ADMUX = (0<<REFS1)|(1<<REFS0); //AVCC 전압 reference
    ADCSRA = (1<<ADEN)|(7<<ADPS0); //ADC enable, 128분주비
}
unsigned int MCU_Init::ADC_Read(unsigned char input)
{
    ADMUX = (input|(0<<REFS1)|(1<<REFS0));//ADC 핀 설정
    ADCSRA |=(1<<ADSC);//ADC start conversion
    while(!(ADCSRA&(1<<ADIF)));//ADC 변환 완료되면 interrupt 발생
    return ADC;
}
```

-소스 설명

1. ADCSRA = (1<<ADEN)|(7<<ADPS0); 에서 (7<<ADPS0)는 (1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)랑 같다. 그 이유는 7를 이진법으로 바꾸면 0b00000111인데 이 비트를 시프트연산자로 인해 ADPS0 부터 왼쪽으로 채워지면 ADCSRA 는 0b00000111이 된다.

2. ADMUX = (input|(0<<REFS1)|(1<<REFS0));에서 input는 우리가 원하는 ADC 핀 번호데 만약 우리가 7번을 원한다면 7의 이진법인 0b00000111과 (0<<REFS1)|(1<<REFS0) 의 값인 0b01000000를 OR로 연산하면 ADMUX 는 0b01000111로 MUX에서 ADC 7번핀을 쓰게 된다.

3. while(!(ADCSRA&(1<<ADIF)));에서 ADCSRA 의 ADIF가 아직 ADC변환이 완료가 안되었으면 0이므로 (1<<ADIF)인 0b00010000랑 AND 연산하면 0b00000000이고 NOT 기호에 의해 while(1);이랑 같아 return ADC;으로 안 내려간다. 완료가 되면 ADCSRA 의 ADIF가 1이므로 다시 AND로 연산하면 0b00010000로 나오고 NOT 기호에 의해 while(0);이라서 return ADC;으로 내려간다. 즉 ADC 값을 도출한다.

*참고

```
void MCU_Init::Uart_num(int data)
{
    Uart_Putch((data/1000)+48);
    Uart_Putch((data%1000)/100+48);
    Uart_Putch((data%100)/10+48);
    Uart_Putch((data%10)+48);
}
```

시리얼 통신으로 ADC 즉 센서 값을 보고 싶으면 일단 총 ADC 값의 범위인 0~1023를 문자형으로 변환해줘야 한다. 왜냐하면 Uart는 문 자형 기호인 char로 데이터를 주고 받기 때문이다. 그래서 처음에 천의 자리를 표현하기 위해 data를 1000으로 나누고 0의 아스키코드인 48을 더해주면 그 해당되는 천의 자리 숫자가 시리얼통신에 나올 것이다. 100의 자리는 1000으로 나눈 그 나머지를 100으로 나누고 아까 처럼 48을 더해주고 10고 마찬가지로 100으로 나눈 그 나머지를 10으로 나누고 48을 더해주고 1의 자리는 10으로 나눈 그 나머지를 48로 더하면 전체적인 센서 값을 볼 수 있다.

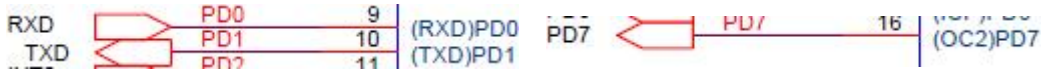
-전체적인 소스

```
//////////////////////// Define ////////////////////////////////
#define       sbi(PORTX,BitX) PORTX|=(1<<BitX)    //비트 set    명령 정의
#define       cbi(PORTX,BitX) PORTX&=~(1<<BitX)   //비트 clear 명령 정의


////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//IO
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void MCU_Init::Init_DDR(void){
    //input
    DDRA = 0x00;//ADC
    cbi(DDRD,DDD0);//UART_RXD

    //output
    DDRC = 0xff;//Test LED
    sbi(DDRD,DDD7);//control Period
    sbi(DDRD,DDD1);//UART_TXD
}
```

1. 클래스 파일인 MCU_Init.cpp에서 DDR를 설정해주는 클래스 멤버 함수다. ADC는 센서값을 읽어와야하기 때문에 입력설정인 0으로 설정해준다.

2. sbi는 어떤 포트를 설정하고 그 포트의 번호를 1로 설정해주게 정의한 것이다. 그래서 PORTX |=(1<<BitX)로 설정해 OR 연산으로 항상 1로 처리된다.

3. cbi는 sbi의 반대다. 어떤 포트를 설정하고 그 포트의 번호를 0으로 설정해주게 정의한 것이다. 그래서 PORTX&=~(1<<BitX)에서 ~(1<<BitX)는 그 포트의 번호가 0이 되고 AND 연산이라 PORTX가 어떤 값이 있든 무조건 0으로 처리된다.



4. 위 회로도를 보면 D0은 Uart 통신에서 데이터 값을 받게 해주는거 D1은 Uart 통신에서 데이터 값을 보낼 수 있게 해주는거 D7 은 Timer/Counter2의 주기를 알 수 있다. 그래서 cbi(DDRD,DDD0);는 데이터 값을 읽어오는 것으로 0으로 설정하고 sbi(DDRD,DDD1);는 데이터 값을 출력하는 것이므로 1로 설정해주었고 주기 파형을 출력하기 위해 sbi(DDRD,DDD7)를 1로 설정해주었다.

```
//////////////////// Main ////////////////////////////
int main(void)
{

//////////////////// Register Init ////////////////////////////
    mcu_init.Init_DDR();
    mcu_init.Init_Timer2();
    mcu_init.Init_TIMSK();
    mcu_init.Init_UART();
    mcu_init.Init_ADC();


//////////////////// Interrupt Enable ////////////////////////////
    sei();

    while(1)
    {

    }
}
```

5. 앞에서 보여준 모든 DDR, TIMER/COUNTER2, UART, ADC 설정한 MCU_Init.cpp의 멤버함수를 main 문에 선언해준다.

6. TIMER/COUNTER2, UART, ADC 모두 Interrupt가 발생하므로 sei(); 를 설정해준다.

```
//////////////////// Include ////////////////////////////
#include <avr/io.h>
#include <avr/interrupt.h>
#include "MCU_Init.h"

//////////////////// Define /////////////////////////////
#define      sbi(PORTX,BitX) PORTX|=(1<<BitX)   //비트 set    명령 정의
#define      cbi(PORTX,BitX) PORTX&=~(1<<BitX)   //비트 clear 명령 정의

//////////////////// Global Var /////////////////////////////
MCU_Init mcu_init;
static int m_send_period=0;

//////////////////// Interrupt /////////////////////////////
SIGNAL(TIMER2_OVF_vect)
{
    ////////// Timer Start ////////////////////////////
    sbi(PORTD,PORTD7);
    cli();

    ////////// Local Var ////////////////////////
    unsigned int a_ADC_data[4]={0,};

    ////////// ADC Read ////////////////////////////
        for(int i=0;i<4;i++)
           a_ADC_data[i] = mcu_init.ADC_Read(i);

    /////// Uart Send ////////////////////////////
           m_send_period++;
           m_send_period%=50;

           if(m_send_period==0)//500ms
          {
           mcu_init.Uart_num(a_ADC_data[0]); mcu_init.Uart_Putch(',');//데이터구분
           mcu_init.Uart_num(a_ADC_data[1]); mcu_init.Uart_Putch(',');//데이터구분
           mcu_init.Uart_num(a_ADC_data[2]); mcu_init.Uart_Putch(',');//데이터구분
           mcu_init.Uart_num(a_ADC_data[3]); mcu_init.Uart_Putch(',');//데이터구분
           mcu_init.Uart_Putch(0x0D);//ENTER

          }

    //// Control Period Set ////////////////////////////
           TCNT2=100;
    //// Timer End ////////////////////////////
           cbi(PORTD,PORTD7);
           sei();
}
```

1. MCU_Init mcu_init;는 클래스인 MCU_Init 의 변수를 선언해 준거라고 생각하면 된다.

2. cli();는 Interrupt를 활성화를 안한다는 뜻이다.

3. a_ADC_data[i] = mcu_init.ADC_Read(i); 를 반복문으로 돌려서 ADC 핀 번호를 대입한다.

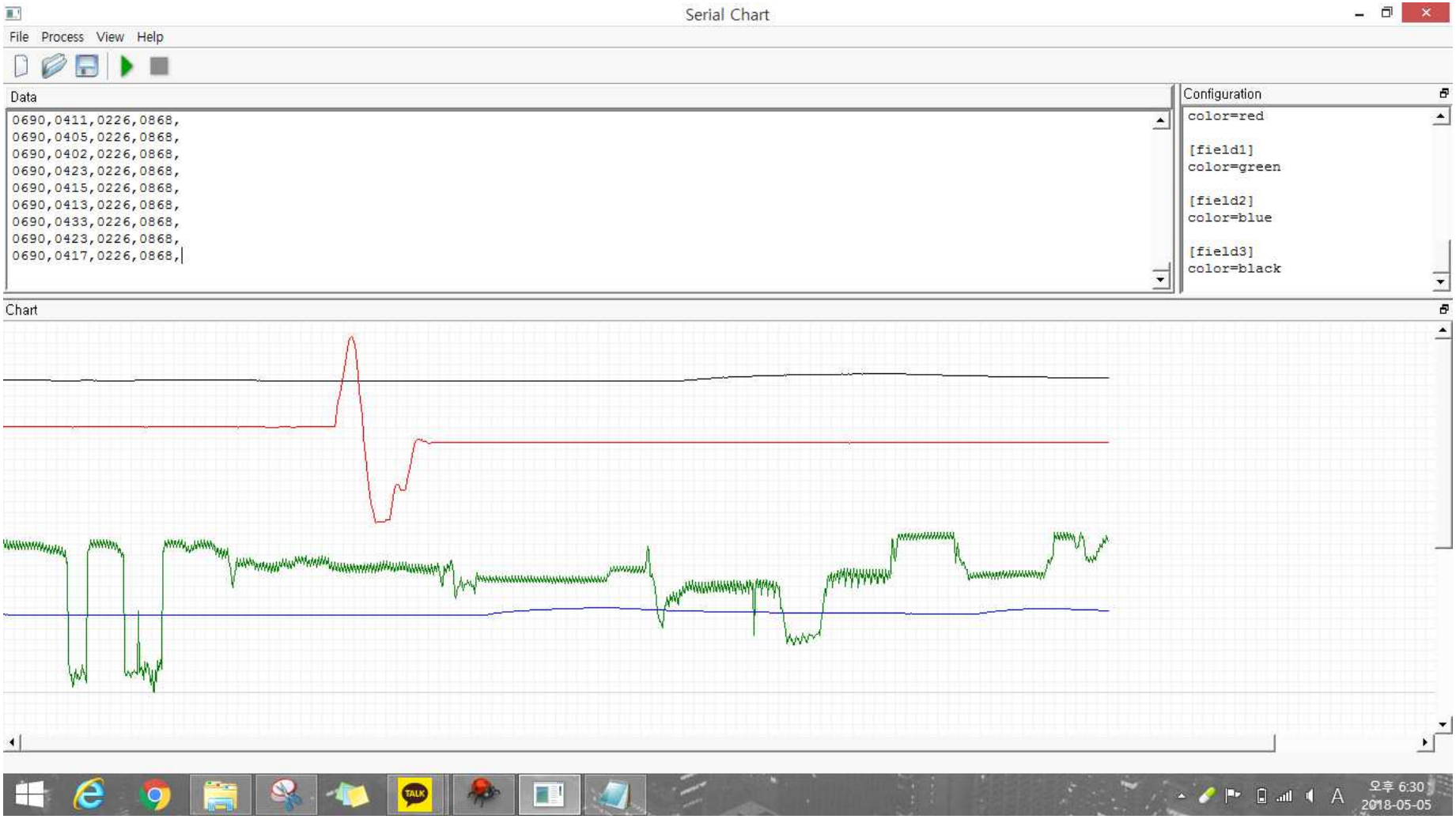
4. m_send_period++; %=50;는 버퍼 방지용 소스다. 통신할 때 버퍼가 생기면 통신이 지연될 수 있어 원하는 시간에 그래프가 잘 안 나올 수 있다. 하지만 제어주기의 동작시간이 평상시보다 길어진다. 이거 안 쓰고 통신해보니 시리얼 통신 프로그램이 버퍼 때문에 렉이 많이 걸렸다.

5. ADC의 값을 Uart_num()의 함수로 통해 통신으로 보내준다.

6. TCNT 값을 또 다시 넣어주므로써 계속해서 카운팅 할 때 TCNT 값에서 다시 시작된다.

7. cbi를 설정하므로써 앞에서 설정한 Timer를 시작해주는 sbi에서 PORTD7를 0으로 설정한다.

-결과



1. ADC0(빨간색)은 가변저항, ADC1(초록색)는 조도센서, ADC2(파랑색)는 LM35, ADC3(검은색)는 Themister다.