AVR – I2C통신
MS5611 모듈

임베디드스쿨 2기

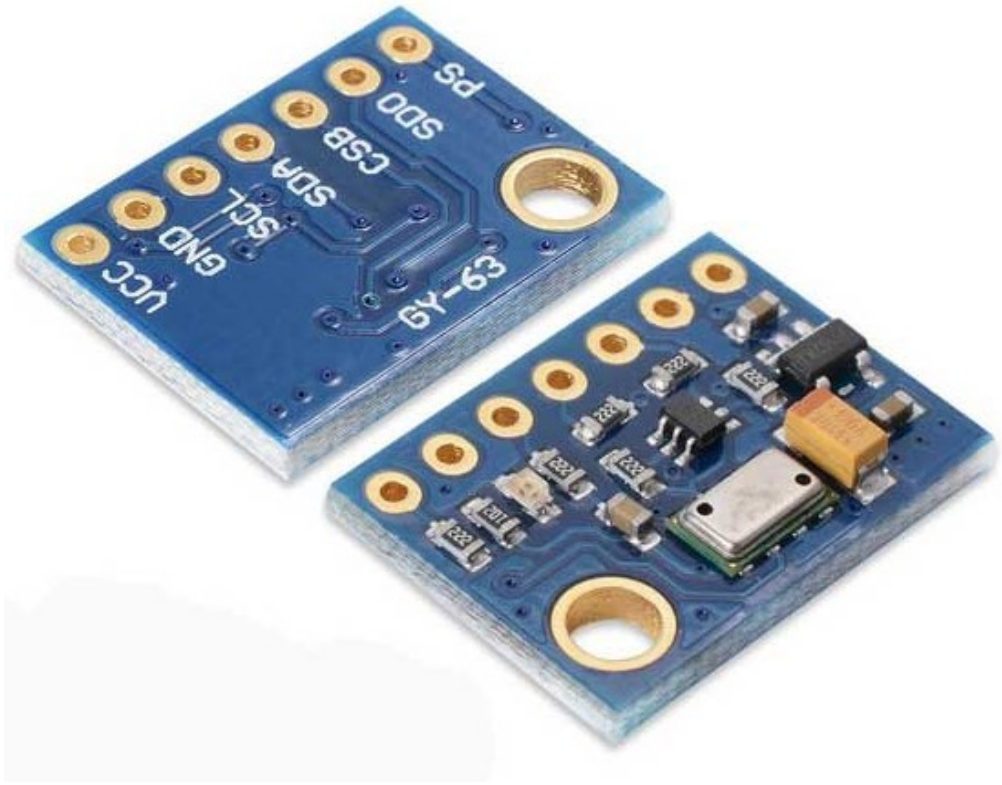
Lv1과정

2021. 07. 19

박태인

MS5611 - 대기압,고도계 센서 모듈

ㄴ 이번 시간에는 대기압 센서를 통해 I2C 통신을 하고 I2C 통신 방법 및 센서 작동 방법을 익혀 보도록 한다.



위 제품은 **MS5611 대기압 센서 모듈** 입니다.
24비트의 분해능을 가진 ADC를 내장하고 있어 매우 정교한 출력 값을 보여 줍니다.
3.3V/5V 시스템과 사용이 가능하며 I2C 를 이용하여 통신이 가능 합니다.

MS5611, AVR 연결 방법

(주의) CSB와 PS는 우측의 회로처럼
HIGH가 유지 되어야 한다.
(ms5611의 datasheet
에서 확보된 내용)

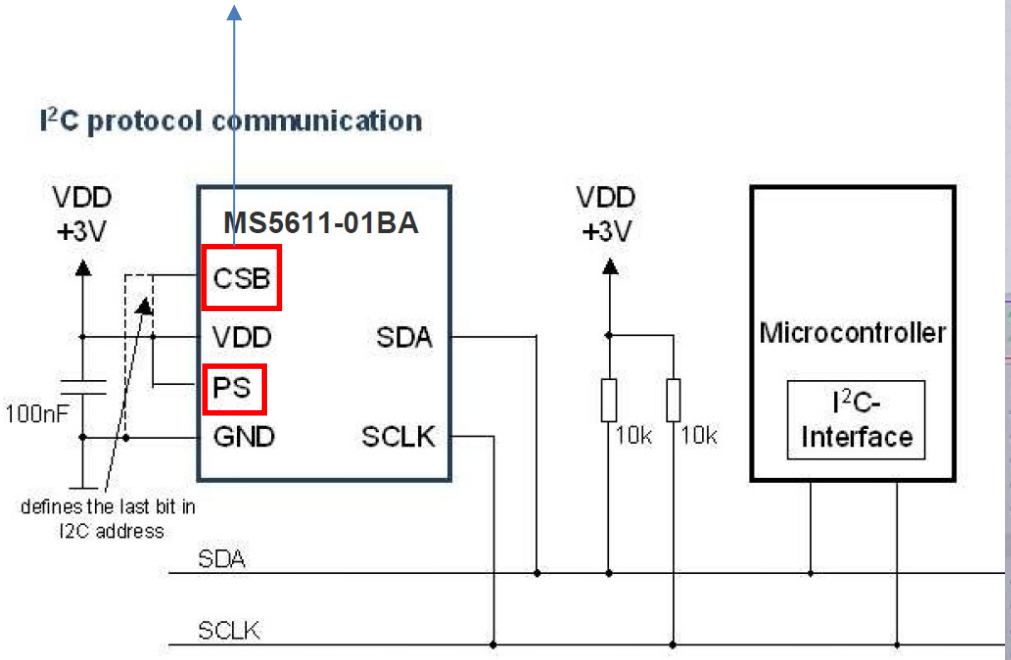
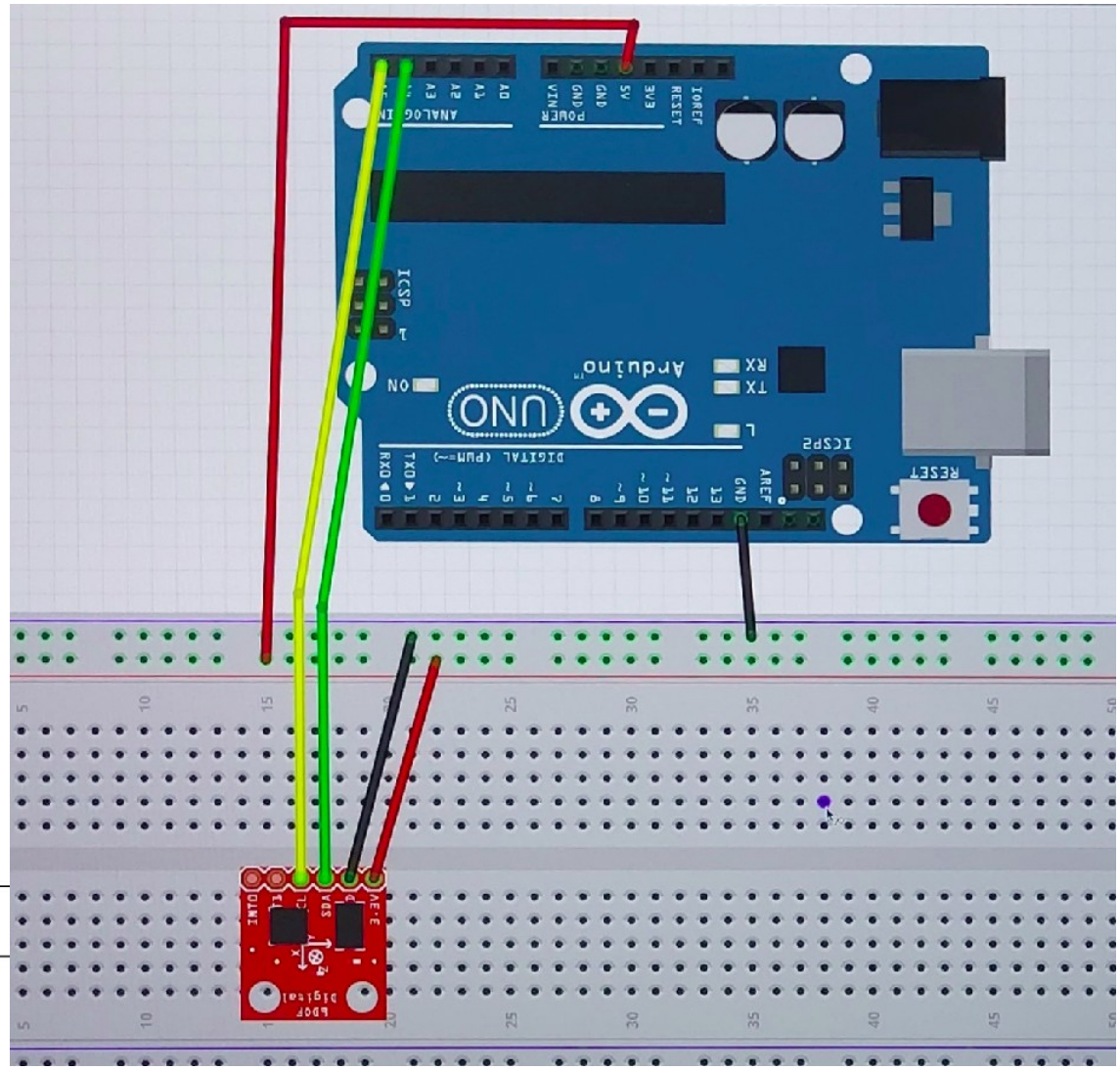
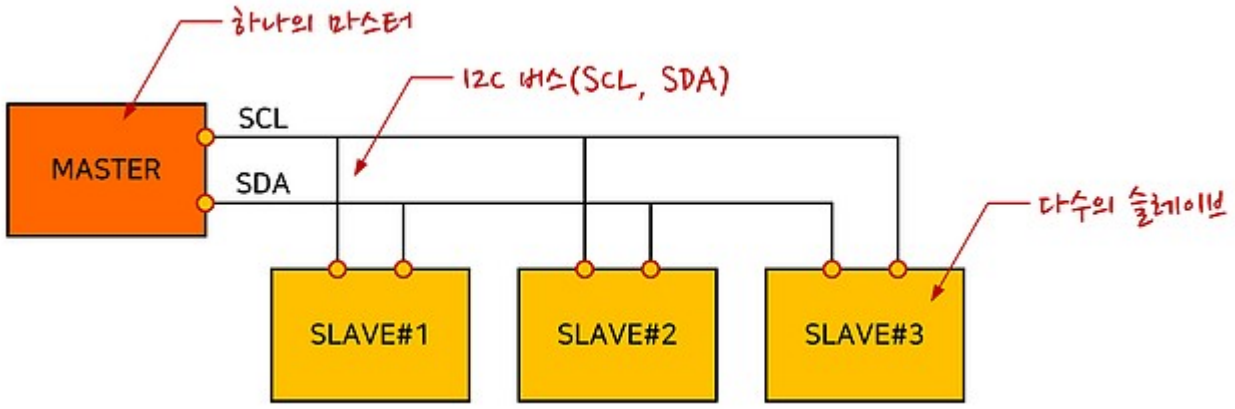


Figure 17: Typical application circuit with SPI / I²C protocol communication



I2C 통신이란?

- I2C는 두개의 신호선(SDA, SCL)으로 다수의 I2C 통신을 지원하는 디바이스와 데이터를 송/수신 할 수 있는 통신 방식입니다.

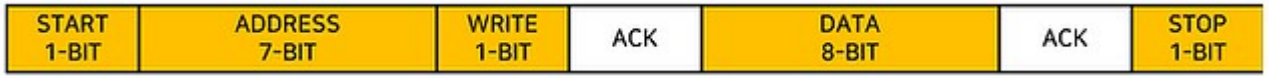


하나의 마스터와 다수의 슬레이브로 연결이 구성되며, 마스터에서 **기준 클럭(SCL)을 생성**하고, 이 클럭에 맞춰 **데이터(SDA)를 전송 및 수신** 합니다.

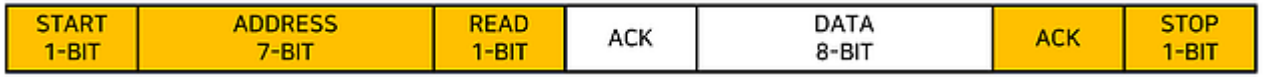
각 송신과 수신 구분(**송신과 수신이 동시에 이루어지지 않음**)되어 있는 반 이중(Half-Duplex)방식 입니다.

각 슬레이브는 개별 주소(어드레스)를 가지고 있으며, 이 주소를 통해 식별이 가능 합니다.
즉, **기준클럭과 데이터는 I2C 네이트워크의 모든 디바이스에 전달** 되고, 해당 주소를 가진 디바이스만 응답하는 방식으로
서로 데이터를 주고 받습니다.

마스터에서 슬레이브로 **1바이트 데이터를 쓸 때** 데이터 규격은 다음과 같습니다.
(주황색 : 마스터, 흰색 : 슬레이브)



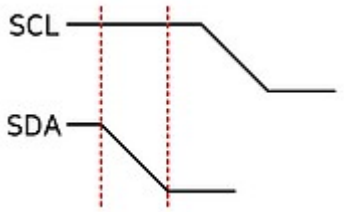
마스터에서 슬레이브로 **1바이트 데이터를 읽을 때** 데이터 규격은 다음과 같습니다.



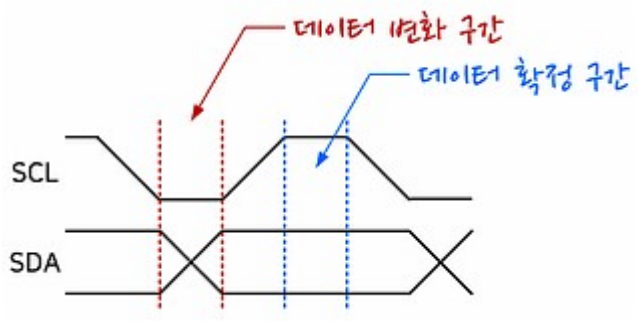
(다음페이지 계속)

I2C 통신이란?

START BIT는 SCL이 HIGH를 유지하고 있을 때, SDA가 HIGH에서 LOW로 변화(하강엣지) 하면 START로 인식합니다.

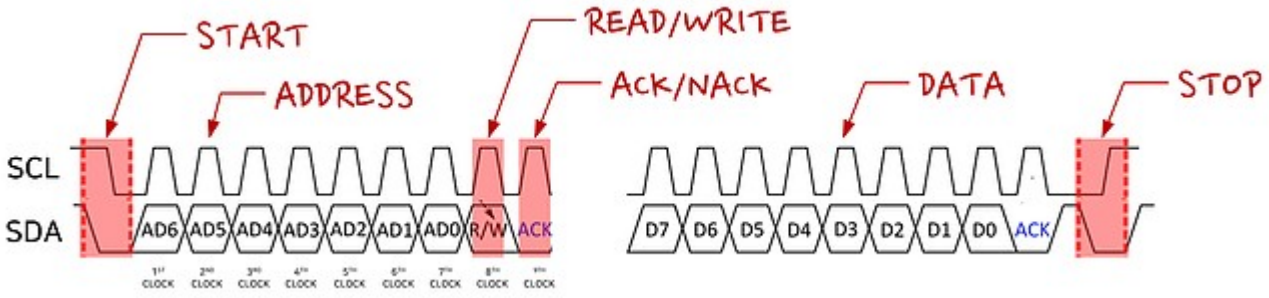


Address 필드는 **SCL이 LOW일 때 데이터를 바꾸고**, **SCL이 HIGH일 때 데이터를 확정** 합니다.



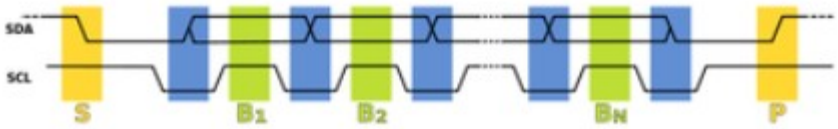
Write 필드는 “LOW”, READ는 “HIGH”
ACK 필드는 “LOW”, NACK는 “HIGH”

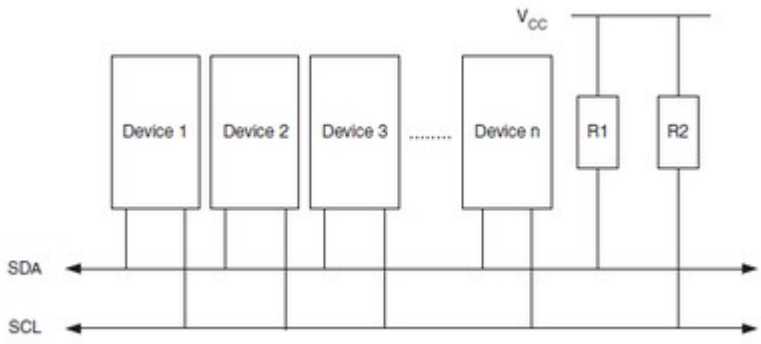
위에서 설명한 내용을 요약하면 I2C 데이터 규격은 다음과 같습니다.



I2C 통신이란?

뭔가 제대로 이해가 되지 않아 다른 설명을 더 추가해 보자.





SDA와 SCL 선의 신호는 풀업 저항에 의해 기본적으로 HIGH. 그러다가 SDA 신호만 LOW로 떨어지면 **시작 신호(S)**라고 판단한다.

그 후에 SCL선으로 클럭 신호가 만들어 지는데, 클럭 신호가 LOW일 때가 SDA 신호를 비트 신호로 바꾸는 시간(파란색 부분)
클럭 신호가 HIGH 일 때가 SDA 신호를 읽는 시간(녹색 부분).

다시 말해, **SCL 신호가 LOW가 되면 다음비트 신호로 바꾸고, 다시 HIGH에서 읽고.**

한 클럭에 한 비트씩 데이터 신호를 만들고, 모든 비트의 전송이 끝난 후 SCL 신호가 HIGH가 되면 SDA 신호 역시 HIGH로 만들어 **정지신호(P)**를 만든다.

시작 신호 뒤에 나오는 첫 7비트는 슬레이브의 주소값 이어야 하며,

8번째 비트는 데이터를 읽어오기 위한 신호인지, 쓰기위한 신호 인지 나타내는 비트로 사용된다.

슬레이브의 주소 값과 읽기/쓰기 비트는 마스터에서 생성 할 수 있으며,

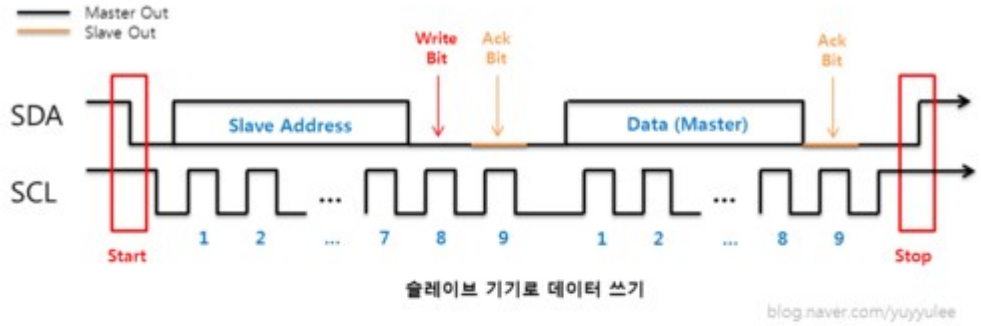
'쓰기'일 경우에 마스터에서 이후 데이터를 생성,'읽기 ' 일 경우에는 슬레이브에서 데이터를 생성한다.

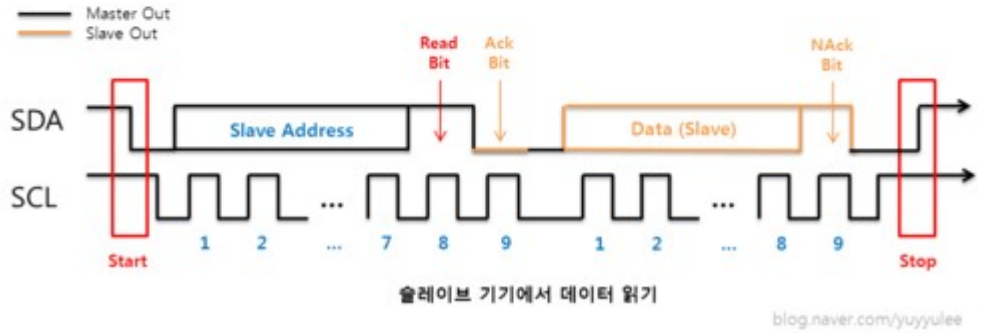
8비트 데이터 전송 후에는 슬레이브에서 응답 신호(ACK)를 만들어 수신을 확인해 준다.

응답 신호는 기본적으로 LOW 여야 하며, 만일 **슬레이브가 데이터를 전송하는 상태에서 모든 데이터의 전송이 끝났을 경우 HIGH상태**가 된다**. 이를 NACK**라고 하는데, 이 외의 경우에 NACK가 발생하는 경우는 통신 에러나 데이터 에러의 경우이다.

I2C 통신이란?

다음은 데이터를 쓸 경우와 읽는 경우의 신호를 그림으로 다시 간단히 나타낸 것이다.

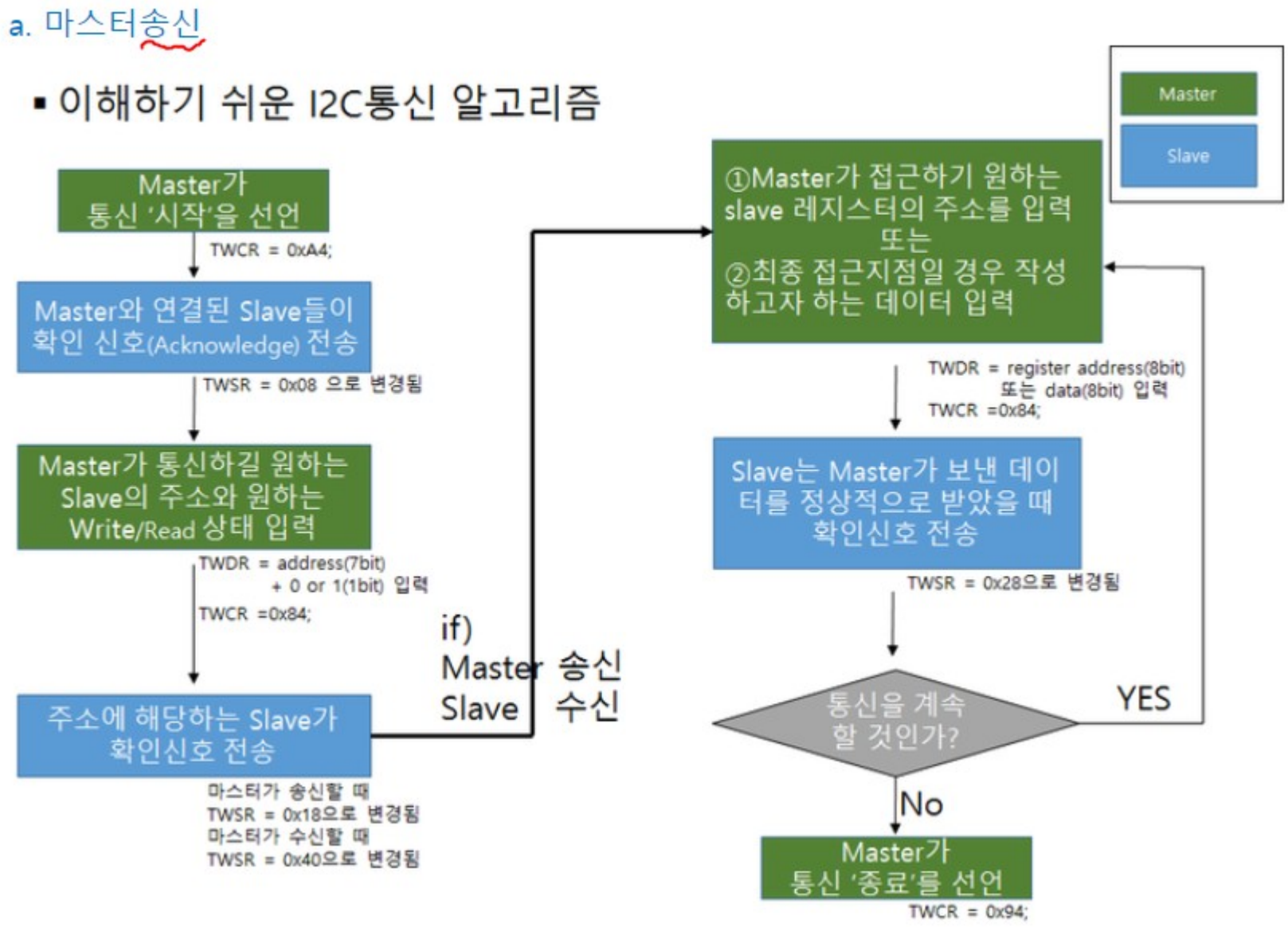




저 신호를 직접 다 만들어야 하는 것은 아니며..
대부분의 칩에서 I2C 통신은 하드웨어 기능으로 구성되어 있어서 I2C 관련 레지스터의 비트를 설정하는 것 만으로도
시작 신호와 데이터신호, 클럭 신호, 정지 신호를 출력 할 수 있다.

I2C 통신이란?

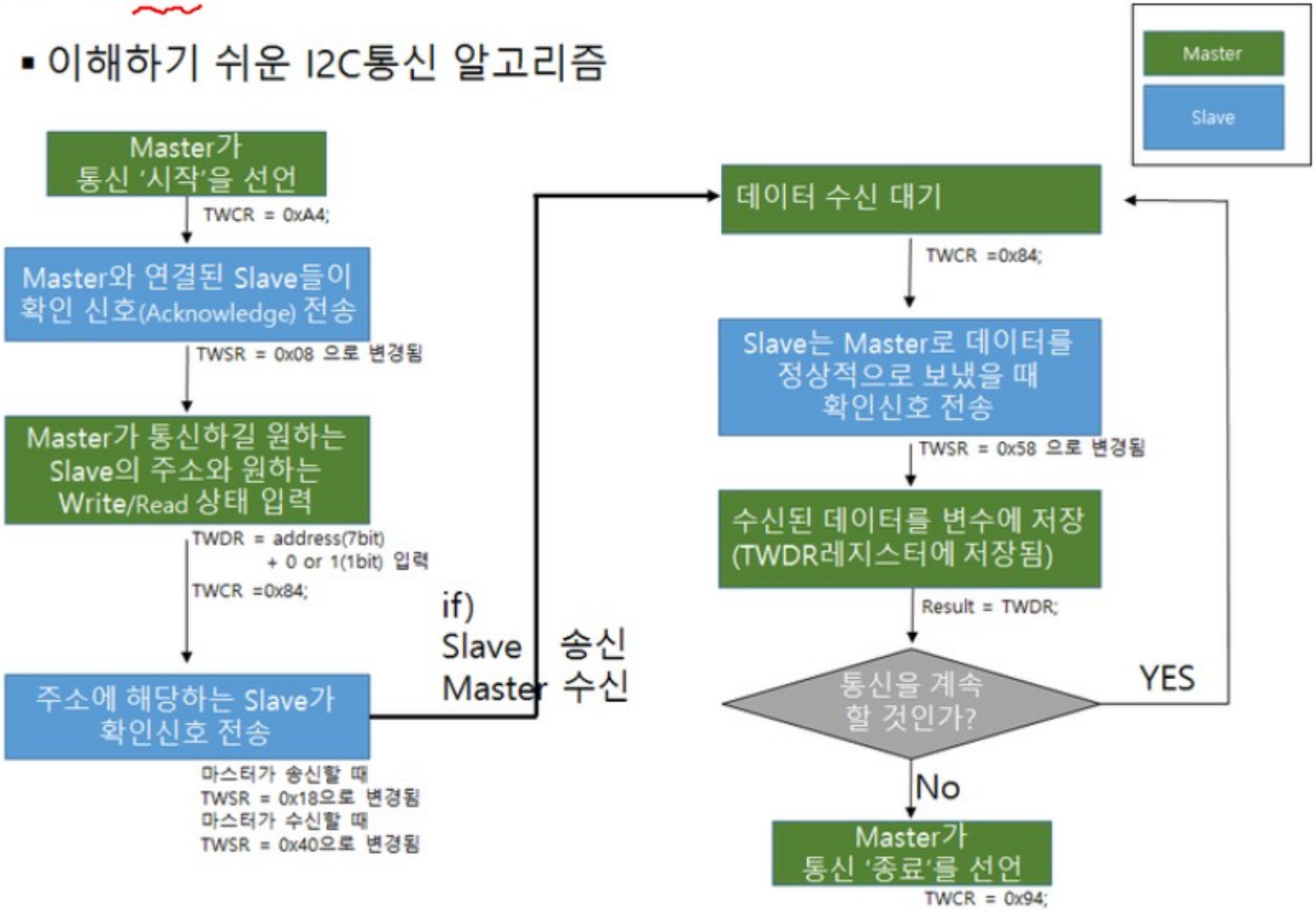
I2C 통신에 대한 알고리즘을 블록도로 나타낸 그림이 있어 아래에 참고 하면 좋을 듯 하다.(수신은 다음페이지)



I2C 통신이란?

b. 마스터수신

▪ 이해하기 쉬운 I2C통신 알고리즘



코딩(I2C 통신 설정) - twi.c

main.c | ms5611.c | ms5611.h | **twi.c** | twi.h | uart.c | uart.h

```
1
2   #include "twi.h"
3
4   #define TW_STS              0xF8
5
6   #define TWI_SCL             0x20
7   #define TWI_SDA             0x10
8   #define TWI_RD              0x01
9   #define TWI_WR              0x00
10
11  #define TWI_START           0x08
12  #define TWI_RESTART         0x10
13
14  /* 마스터 전송 */
15  #define TWI_MT_SLA_ACK      0x18
16  #define TWI_MT_SLA_NACK     0x20
17  #define TWI_MT_DATA_ACK     0x28
18  #define TWI_MT_DATA_NACK    0x30
19  #define TWI_MT_ARB_LOST     0x38
20
21  /* 마스터 수신 */
22  #define TWI_MR_ARB_LOST     0x38
23  #define TWI_MR_SLA_ACK      0x40
24  #define TWI_MR_SLA_NACK     0x48
25  #define TWI_MR_DATA_ACK     0x50
26  #define TWI_MR_DATA_NACK    0x58
```

헤더 및 define 구간

TWI 통신에서 사용될 용어 define

STS, SCL, SDA, RD, WR

STRAT, RESTART

전송 및 수신 관련 레지스터 설정

코딩(I2C 통신 설정) - twi.c

I2c_init

main.c | ms5611.c | ms5611.h | **twi.c** | twi.h | uart.c | uart.h

```
28  void i2c_init (void)
29  {
30      // 데이터시트 200 page
31      //
32      /* TWI Clock: x Hz */
33      TWSR = 0x00;     /* prescale 없음 */
34
35      // 데이터 시트 198 page , 계산 180 page
36      // gdb 계산, p/d 16000000 / (16 + 2 * 12 * 1) = 400000, maximum = 400kHz
37      TWBR = 12;
38  }
```

I2c 통신 설정 초기화
ㄴ 클럭 Hz (SCL)

SCL 계산 결과
'400kHz'

TWSR – TWI Status Register

Bit	7	6	5	4	3	2	1	0	
(0xB9)	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

Table 21-8. TWI Bit Rate Prescaler

TWPS1	TWPS0	Prescaler Value
0	0	1
0	1	4
1	0	16
1	1	64

TWBR – TWI Bit Rate Register

Bit	7	6	5	4	3	2	1	0	
(0xB8)	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
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	

21.5.2 Bit Rate Generator Unit

This unit controls the period of SCL when operating in a master mode. The SCL period is controlled by settings in the TWI bit rate register (TWBR) and the prescaler bits in the TWI status register (TWSR). Slave operation does not depend on bit rate or prescaler settings, but the CPU clock frequency in the slave must be at least 16 times higher than the SCL frequency. Note that slaves may prolong the SCL low period, thereby reducing the average TWI bus clock period. The SCL frequency is generated according to the following equation:

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \times (\text{PrescalerValue})}$$

코딩(I2C 통신 설정) - twi.c

I2c_start

```
40  unsigned char i2c_start (unsigned char address)
41  {
42      uint8_t twst;
43
44      // datasheet 199 page, 하드웨어에 의해 setting. 소프트웨어 적으로 setting 해야 하는 것도 있음.
45      // TWI 통신의 START 신호 + 활성화 + 끝 판정
46      TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
47
48      // TWINT는 끝 판정.
49      // 끝났으면 자동으로 뭐다 ? 1이 나온다는건 아직 처리를 하고 있다. 0은 처리가 끝났다.
50      // 그것을 not 을 통해 처리가 끝났는지를 판정함.
51      while (!(TWCR & (1 << TWINT)))
52      {
53          ;
54      }
```

TWCR – TWI Control Register

Bit	7	6	5	4	3	2	1	0	
(0xBC)	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – TWINT: TWI Interrupt Flag**

This bit is set by hardware when the TWI has finished its current job and expects application software response. If the I-bit in SREG and TWIE in TWCR are set, the MCU will jump to the TWI interrupt vector. While the TWINT flag is set, the SCL low period is stretched. The TWINT flag must be cleared by software by writing a logic one to it. Note that this flag is not automatically cleared by hardware when executing the interrupt routine. Also note that clearing this flag starts the operation of the TWI, so all accesses to the TWI address register (TWAR), TWI status register (TWSR), and TWI data register (TWDR) must be complete before clearing this flag.

- **Bit 5 – TWSTA: TWI START Condition Bit**

The application writes the TWSTA bit to one when it desires to become a master on the 2-wire serial bus. The TWI hardware checks if the bus is available, and generates a START condition on the bus if it is free. However, if the bus is not free, the TWI waits until a STOP condition is detected, and then generates a new START condition to claim the bus master status. TWSTA must be cleared by software when the START condition has been transmitted.

- **Bit 2 – TWEN: TWI Enable Bit**

The TWEN bit enables TWI operation and activates the TWI interface. When TWEN is written to one, the TWI takes control over the I/O pins connected to the SCL and SDA pins, enabling the slew-rate limiters and spike filters. If this bit is written to zero, the TWI is switched off and all TWI transmissions are terminated, regardless of any ongoing operation.

TWINT는 정확히 지금
새로운 동작이 가능하다는 준비 상태를 의미!
(결국 하나의 작업이 끝난 상태여야 가능)

TWINT를 설정하는 것의 의미는
인터럽트가 발생한다고 해도 클리어 되지 않고
처리 할 것이냐는 것을 사용자에게 위임
하는 것.
(처리 해야 한다는 것을 알려 준다 라고 보면 된다. -> 할 일이 있어!!! 같은 의미)
ㄴ **우리가 직접 1을 써줘서 비트 클리어를 시켜줘야 한다.**

즉, TWINT가 1일 때 비트 클리어 시킨다는 의미
이므로 직접 1을 쓰고 이중으로 while문 까지 써서
TWINT가 1이면 다음으로 넘어 가게 한 것이다.

코딩(I2C 통신 설정) - twi.c

I2c_start

유휴상태 : 송신측으로 부터 데이터가 전달 되기를 기다리고 있는 수신측의 상태

main.c | ms5611.c | ms5611.h | twi.c | twi.h | uart.c | uart.h

```
55
56      // START 혹은 repeat START 인 경우에만 아래로 내려감
57      // 200 page, status는 196 page 표 참조. 유휴상태 이거나 전송을 진행 하고 있는 것을 알려줌.
58      twst = TWSR & 0xF8;
59      // 0x08도 아니고 0x10이 아니면 1을 리턴 (TWSR), 189 page 표 참조.
60      if ((twst != TWI_START) && (twst != TWI_RESTART))
61      {
62          return 1;
63      }
64
```

twst에 현재의 TWSR의 값을 마스킹!!

TWSR – TWI Status Register

Bit	7	6	5	4	3	2	1	0	
(0xB9)	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

TWSR의 상태 값이 0x08(START)
이거나 0x10(repeated START)
가 아닌 경우는
Retrun 1 ;

1을 리턴 한다는건 문제가 있다는 것.

Table 21-4. Status Codes for Master Receiver Mode

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x08	A START condition has been transmitted	Load SLA+R	0	0	1	X	SLA+R will be transmitted ACK or NOT ACK will be received
0x10	A repeated START condition has been transmitted	Load SLA+R or	0	0	1	X	SLA+R will be transmitted ACK or NOT ACK will be received
		Load SLA+W	0	0	1	X	SLA+W will be transmitted logic will switch to master transmitter mode

코딩(I2C 통신 설정) - twi.c

I2c_start

main.c | ms5611.c | ms5611.h | twi.c | twi.h | uart.c | uart.h

```
55
56      // START 혹은 repeat START 인 경우에만 아래로 내려감
57      // 200 page, status는 196 page 표 참조. 유휴상태 이거나 전송을 진행 하고 있는 것을 알려줌.
58      twst = TWSR & 0xF8;
59      // 0x08도 아니고 0x10이 아니면 1을 리턴 (TWSR), 189 page 표 참조.
60      if ((twst != TWI_START) && (twst != TWI_RESTART))
61      {
62          return 1;
63      }
64
65      // TWDR에 사용할 장치 주소 설정. (0x76, ms5611 주소)
66      TWDR = address;
67      TWCR = (1 << TWINT) | (1 << TWEN);
68
69      // 처리가 완료 될 때 까지 대기
70      while (!(TWCR & (1 << TWINT)))
71      {
72          ;
73      }
74
75      // check value of TWI Status Register, ACK는 계속한다. NACK는 이제 끝났다.
76      // 0x18과 0x20, 186 page 참조. 18은 한방에 보내는거 20은 repaeat start.(?)
77      // ACK 혹은 NACK 판단.
78      twst = TWSR & 0xF8;
79      if ((twst != TWI_MT_SLA_ACK) && (twst != TWI_MR_SLA_NACK))
80      {
```

TWDR – TWI Data Register

Bit	7	6	5	4	3	2	1	0	
(0xBB)	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

Address (Slave의 주소)를 TWDR에 넣는다.
TWCR 에 다시 한번 TWI 인터럽트 설정 및 Enable

TWINT가 1 일 때 까지 대기(비트 클리어 시 까지)

현재의 TWSR 레지스터 값을 마스킹

ACK 인지 NACK인지 검사
- ㄴ ACK : 계속 한다.
- ㄴ NACK : 이제 끝났다.

코딩(I2C 통신 설정) - twi.c

I2c_start

main.c | ms5611.c | ms5611.h | twi.c | twi.h | uart.c | uart.h

```
75      // check value of TWI Status Register, ACK는 계속한다. NACK는 이제 끝났다.
76      // 0x18과 0x20, 186 page 참조. 18은 한방에 보내는거 20은 repeat start.(?)
77      // ACK 혹은 NACK 판단.
78      twst = TWSR & 0xF8;
79      if ((twst != TWI_MT_SLA_ACK) && (twst != TWI_MR_SLA_NACK))
80      {
81          return 1;
82      }
83
84      return 0;
85  }
```

이건 무시.

현재의 TWSR 레지스터 값을 마스킹

I2c_repeat_start

```
87  unsigned char i2c_repeat_start (unsigned char address)
88  {
89      return i2c_start(address);
90  }
```

앞서 했던 strart 주소 값을 그대로 다시 전송 하여 repeat_start.

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x18	SLA+W has been transmitted; ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset
0x20	SLA+W has been transmitted; NOT ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset

I2c_stop

```
92  // 198 page
93  void i2c_stop (void)
94  {
95      // 활성화 + STOP 비트 + 끝 판정
96      TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
97
98      // 올바르게 전송이 완료 될 때 까지 대기.
99      while (TWCR & (1 << TWSTO))
100     {
```

- **Bit 4 – TWSTO: TWI STOP Condition Bit**

Writing the TWSTO bit to one in master mode will generate a STOP condition on the 2-wire serial bus. When the STOP condition is executed on the bus, the TWSTO bit is cleared automatically. In slave mode, setting the TWSTO bit can be used to recover from an error condition. This will not generate a STOP condition, but the TWI returns to a well-defined unaddressed slave mode and releases the SCL and SDA lines to a high impedance state.

STOP 비트 설정이 제대로 됐는지 확인 후 넘어감.

이번 특이점은 이 STOP 비트 설정

코딩(I2C 통신 설정) - twi.c

main.c | ms5611.c | ms5611.h | twi.c | twi.h | uart.c | uart.h

```
92  // 198 page
93  void i2c_stop (void)
94  {
95      // 활성화 + STOP 비트 + 끝 판정
96      TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
97
98      // 올바르게 전송이 완료 될 때 까지 대기.
99      while (TWCR & (1 << TWSTO))
100     {
101         ;
102     }
103 }
104
105 unsigned char i2c_write (unsigned char data)
106 {
107     uint8_t twst;
108
109     TWDR = data;
110     TWCR = (1 << TWINT) | (1 << TWEN);
111
112     while (!(TWCR & (1 << TWINT)))
113     {
114         ;
115     }
116
117     twst = TWSR & 0xF8;
```

I2c_stop

다시 한번 끝 판정

끝 판정이 TWSTO 가 제대로 됐는지 확인.

I2c_write

인수로 받은 data를 TWDR 레지스터에 삽입.
TWCR Control 레지스터에 TWI 활성화 및 enable on

새로운 동작이 가능한 상태 인지를 판별 (TWINT)

Twst 변수에 현재 TWSR(상태 레지스터)에 상태 값 저장.

코딩(I2C 통신 설정) - twi.c

main.c | ms5611.c | ms5611.h | twi.c | twi.h | uart.c | uart.h

```
102     }
103 }       I2c_write
104
105 unsigned char i2c_write (unsigned char data)
106 {
107     uint8_t twst;
108
109     TWDR = data;
110     TWCR = (1 << TWINT) | (1 << TWEN);
111
112     while (!(TWCR & (1 << TWINT)))
113     {
114         ;
115     }
116
117     twst = TWSR & 0xF8;
118     if (twst != TWI_MT_DATA_ACK)
119     {
120         return 1;
121     }
122
123     return 0;
124 }
125
```

자, 다시 한번 write 함수를 전체적으로 봤을 때..

인수로 받은 data를 TWDR 레지스터에 삽입.

TWCR Control 레지스터에 TWI 활성화 및 enable on

새로운 동작이 가능한 상태 인지를 판별 (TWINT)

Twst 변수에 현재 TWSR(상태 레지스터)에 상태 값 저장.

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x28	Data byte has been transmitted; ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset

Data를 보내고 ACK를 받았는지 확인 하는 절차.
ㄴ 못받으면 return 1 : 에러

코딩(I2C 통신 설정) - twi.c

main.c | ms5611.c | ms5611.h | **twi.c** | twi.h | uart.c | uart.h

```
126  unsigned char i2c_read_ack (void)
127  {
128      // ACK 신호 전송
129      TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWEA);
130      // 올바르게 처리되었는지 대기
131      while (!(TWCR & (1 << TWINT)))
132      {
133      }
134      // 읽은 데이터가 TWDR에 들어오기 때문에 이것을 리턴.
135      return TWDR;
136  }
137
138  unsigned char i2c_read_nack (void)
139  {
140      // 왜 이렇게 두개가 들어 올까? PROM이 16비트 데이터 이므로 (readAck + readNak)
141      TWCR = (1 << TWINT) | (1 << TWEN);
142      // 잘 수신하였는지 대기
143      while (!(TWCR & (1 << TWINT)))
144      {
145          ;
146      }
147      // 마지막 8 비트 수신
148      // 만약에 24 비트면 Nak를 한번더 해야 겠지 (nak는 연속적인거 뒤에 쭉쭉)
149      return TWDR;
150  }
151
```

I2c_read_ack

ACK 신호 수신 받는

- **Bit 6 – TWEA: TWI Enable Acknowledge Bit**

The TWEA bit controls the generation of the acknowledge pulse. If the TWEA bit is written to one, the ACK pulse is generated on the TWI bus if the following conditions are met:

1. The device's own slave address has been received.
2. A general call has been received, while the TWGCE bit in the TWAR is set.
3. A data byte has been received in master receiver or slave receiver mode.

By writing the TWEA bit to zero, the device can be virtually disconnected from the 2-wire serial bus temporarily. Address recognition can then be resumed by writing the TWEA bit to one again.

새로운 동작이 가능한 상태 인지를 판별 (TWINT)

요게 좀 특이점인데, 읽은 데이터가 TWDR에 들어 와서 이것을 리턴 한다!

I2c_read_nack

새로운 동작이 가능한 상태 인지를 판별 (TWINT)

한번에 받는 것이 8 비트니 2번 받아서 16비트를 만든다고 보면 된다.

최종 받이트에서 NACK을 보낸다.

TCP에서도 마찬가지로 사용하는 개념인데 ACK는 보통 잘 수신하였음 NACK는 문제가 생김 혹은 끝났음을 의미함

이 말은 틀렸다.. Ack ack ack ... nack 방식이다.
ack 후 마지막에 nack가 붙는.

Nack에서도 역시 Data를 return.

코딩(I2C 통신 설정) - twi.h

main.c | ms5611.c | ms5611.h | twi.c | **twi.h** | uart.c | uart.h

```
1
2   #ifndef __TWI_H__
3   #define __TWI_H__
4
5   #include <avr/io.h>
6   #include <util/delay.h>
7   #include <stdio.h>
8   #include <stdlib.h>
9   #include <string.h>
10
11  void i2c_init (void);
12  unsigned char i2c_start (unsigned char);
13  unsigned char i2c_repeat_start (unsigned char);
14  void i2c_stop (void);
15  unsigned char i2c_write (unsigned char);
16  unsigned char i2c_read_ack (void);
17  unsigned char i2c_read_nack (void);
18
19  #endif
20
```

Twi.c 에서 선언한 함수들의 원형을 헤더 파일에 선언하고 다른 .c에서 활용 할 수 있도록 한다.

요런 형식은 자주 이용되므로 잘 봐두는 것이 좋다.

코딩(I2C 통신 설정) - ms5611.c

이번에는 본격적으로 slave센서인 ms5611 코딩 부분이다.

```
#define F_CPU 16000000UL

#include "ms5611.h"
#include "twi.h"
#include "uart.h"

//TWI 전체적인 흐름 : 주종관계 맺기, 커맨드 날리기, 데이터 읽기

int32_t _ms5611_temp;
uint32_t _ms5611_pres;

struct _ms5611_cal
{
    uint16_t sens, off, tcs, tco, tref, tsens;
}
ms5611_cal;

void ms5611_reset (void)
{
    // MS5611_ADDR 더블 클릭해보면 0x76 이라고 나오는데, 이것은 ms5611-01ba datasheet를 살펴 보면 12 page
    // 111011(NOT CSB) ===> I2C => 0x76, SPI = 0x77
    // START 비트 날리고 ms5611 장치 선택
    i2c_start((MS5611_ADDR << 1) | I2C_WRITE);
    // 야 ms5611 이제 구동해봐! (전원 활성화)
    i2c_write(RESET);
```

일단은 TWI의 전체적인 흐름을 인지 하자.
ㄴ 주종관계 맺기, 커맨드 날리기, 데이터 읽기.

구조체 변수로 **sens, off, tcs, tref, tsens** 등의 값을 선언한다.

Ms_5611_reset

Ms5611 작동을 위한 reset 함수 인데
자세한 동작에 대해 다음 페이지에서 서술한다.

코딩(I2C 통신 설정) - ms5611.c

main.c | ms5611.c | ms5611.h | twi.c | twi.h | uart.c | uart.h

```
18
19  void ms5611_reset (void)
20  {
21      // MS5611_ADDR 더블 클릭해보면 0x76 이라고 나오는데, 이것은 ms5611-01ba datasheet를 살펴 보면 12 page
22      // 111011(NOT CSB) ===> I2C => 0x76, SPI = 0x77
23      // START 비트 날리고 ms5611 장치 선택
24      i2c_start((MS5611_ADDR << 1) | I2C_WRITE);
25      // 야 ms5611 이제 구동해봐! (전원 활성화)
26      i2c_write(RESET);
27      // 주종 관계 성립 (Atmega328 : Master, MS5611 : Slave)
28      i2c_stop();
29      _delay_ms(10);
30  }
31
```

Ms_5611_reset

MS5611 가 작동하도록 하는 유일한 방법은 여러 SCLK를 전송한 후 리셋 시퀀스를 보내거나 전원 켜기 리셋을 반복 하는 것!

Salve 주소 설정

RESET SEQUENCE

The reset can be sent at any time. In the event that there is not a successful power on reset this may be caused by the SDA being blocked by the module in the acknowledge state. The only way to get the MS5611-01BA to function is to send several SCLKs followed by a reset sequence or to repeat power on reset.

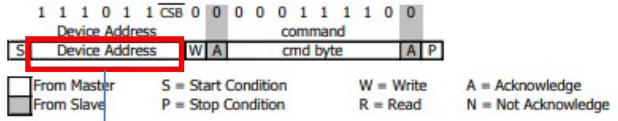
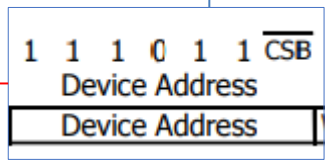


Figure 10: I²C Reset Command



MS5611_ADDR -> 0x76
I2C_WRITE -> 0

MS5611_ADDR을 좌측 << 1을 하는 이유는?
0x76이라는게 원래는 0111 0110 인데, Datasheet를 보면 최상단 0 값은 필요가 없다.
따라서 << 1을 해준 것이고

I2C_WRITE가 0 이 된 이유는 'W' Write가 0 값을 주어야 하기 때문이다.
111 011 | 0 -> 1110110

데이터 시트를 잘 살펴 보면 'R' Read의 경우는 1 값을 주어야 한다.

: RESET 코드가
0001 1110 -> 즉, 0x1E
를 i2c_write 함수로 전송.

전체적 흐름은
Slave 주소 설정 후, 리셋 명령어 날림.

코딩(I2C 통신 설정) - ms5611.c

main.c | **ms5611.c** | ms5611.h | twi.c | twi.h | uart.c | uart.h

```
31
32  // 캘리브리에이션          Ms_read_cal_reg
33  uint32_t ms5611_read_cal_reg (uint8_t reg)
34  {
35      uint8_t PROM_dat1;
36      uint8_t PROM_dat2;
37
38      uint16_t data;
39  // ------------------------------------------- 이부분은 바뀌는 확률이 높다.
40      i2c_start((MS5611_ADDR << 1) | I2C_WRITE);
41      i2c_write(MS5611_CMD_PROM(reg));
42      i2c_repeat_start(MS5611_ADDR << 1 | I2C_READ);
43
44      PROM_dat1 = i2c_read_ack();
45      PROM_dat2 = i2c_read_nack();
46      // ------------------------------------------------
47      // 24 비트면 ack 이 하나더 붙이면 되겠지.
48      i2c_stop();   I2c_stop
49
50      printf("PROM_dat1:%d, %d\n", PROM_dat1, PROM_dat2);   PROM_data 출력
51
52      // 완성된 16 비트 데이터.
53      data = ( PROM_dat1 << 8 ) + (uint16_t)PROM_dat2;
54
55
56      return data;
57  }
```

캘리브레이션이란 ? 무게를 측정하기 전에 바늘이 0을 가르키도록 조정하거나,
시간의 추진 속도를 조정하는 것을 가리킨다.

데이터 1 byte 단위로 dat1, 2 선언.

자, 일단 전체적인 스토리는
I2c_start로 slave 설정 및 write 모드를 끊고,
I2c_write로 데이터(명령어) 를 전송한 뒤
I2c_repeat_start로 재시작을 해서 read 모드를 한 뒤

데이터를 읽는다.

여기에 MS5611_CMD_PROM(reg) 에 대해서는 헤더와
데이터 시트를 참고해야 하므로 다음 페이지에 설명 한다.

총 계산 크기의 데이터는 16 bit 이므로,
8 비트를 shift 시키고,
dat2를 16bit 타입으로 형 변환.
(참고 : 앞에 쉬프트를 하고 뒤에만 형 변환해도 상관 없다.)
신경 쓰이면 앞 쪽의 dat1도 형 변환 해줘도 된다.

코딩(I2C 통신 설정) - ms5611.c

main.c | **ms5611.c** | ms5611.h | twi.c | twi.h | uart.c | uart.h

Ms_read_cal_reg

```
31
32    // 캘리브리에이션
33    uint32_t ms5611_read_cal_reg (uint8_t reg)
34    {
35        uint8_t PROM_dat1;
36        uint8_t PROM_dat2;
37
38        uint16_t data;
39    // ------------------------------------------- 이부분은 바뀌는 확률이 높다.
40        i2c_start((MS5611_ADDR << 1) | I2C_WRITE);
41        i2c_write(MS5611_CMD_PROM(reg));
42        i2c_repeat_start(MS5611_ADDR << 1 | I2C_READ);
```

아래는 header

```
18    #define CMD_ADC_READ          0x00
19    // reg << 1 이니까 2가 되고, 0xA2가 되지. ms5611 10 page 데이터 시트 표, 6,8,11(read sequence) page 참고(6가지의 값 세팅). 0번 비트는 건드릴 필요 없기에 shift 했다.
20    // 캘리브리에이션이 뭔가? 바깥의 노이즈 성분을 계산해서 성능 개선
21    #define MS5611_CMD_PROM(reg)    (0xA0 + ((reg) << 1))
22    #define CMD_PROM_READ           0xA0
23
```

함수형 매크로 형식으로,
Reg는 우리가 입력하는 값에 따라 변합니다.
-> 하드웨어 상태를 그대로 적용해 놨다고 보면 됩니다.

이러한 부분은 어떤 하드웨어도 커버하겠다는 의도로
다형성을 추구하기 위해 추상화 될 필요가 있다고 한다.
ㄴ 이 부분에 대해서 다시 문의 드려 볼 생각.

코딩(I2C 통신 설정) – ms5611.c

Ms5611_init

main.c | ms5611.c | ms5611.h | twi.c | twi.h | uart.c | uart.h

```
59  void ms5611_init (void)
60  {
61      ms5611_reset();
62      UART_string_transmit("ms5611 reset ok\n");
63
64      ms5611_cal.sens  = ms5611_read_cal_reg(1);
65      ms5611_cal.off   = ms5611_read_cal_reg(2);
66      ms5611_cal.tcs   = ms5611_read_cal_reg(3);
67      ms5611_cal.tco   = ms5611_read_cal_reg(4);
68      ms5611_cal.tref  = ms5611_read_cal_reg(5);
69      ms5611_cal.tsens = ms5611_read_cal_reg(6);
70
71      _delay_ms(1000);
72  }
73
74  uint32_t ms5611_conv_read_adc (uint8_t command)
75  {
76      uint8_t rv1;
77      uint8_t rv2;
78      uint8_t rv3;
79
80      uint32_t adc_data;
81
```

Ms5611_reset() 함수로 센서 작동
Reset 이 ok 됨을 uart로 전송

함수형 매크로 형식인데,
예를 들어 ms5611_read_cal_reg(5) 이면
0000 1001 << 1 이므로, 0001 0010 이 되어서 (12) -> C가 된다.

한번 더 해보면
ms5611_read_cal_reg(6) 이면
0000 1010 << 1 이므로, 0001 0100 이 되어서 (14) -> E가 된다.

```
18  #define CMD_ADC_READ            0x00
19  // reg << 1 이니까 2가 되고, 0xA2가 되지. ms5611 10 page 데이터 시트
20  // 캘리브리에이션이 뭔가? 바깥의 노이즈 성분을 계산해서 성능 개선
21  #define MS5611_CMD_PROM(reg)    (0xA0 + ((reg) << 1))
22  #define CMD_PROM_READ           0xA0
23
```

구조체 변수에 각각의 값 저장.

Ms5611_conv_read_adc

변수 rv1~3 선언.

32bit adc_data 변수 선언.
ㄴ 뒤에 계속.

코딩(I2C 통신 설정) – ms5611.c

Ms5611_conv_read_adc

main.c | ms5611.c | ms5611.h | twi.c | twi.h | uart.c | uart.h

```
82
83
84      i2c_start((MS5611_ADDR << 1) | I2C_WRITE);
85      i2c_write(command);
86      i2c_stop();
87      _delay_ms(10); //conversion Time delay
88
89      i2c_start((MS5611_ADDR << 1) | I2C_WRITE);
90      i2c_write(CMD_ADC_READ);
91      i2c_repeat_start(MS5611_ADDR <<1 | I2C_READ);
92
93      // 끝 날 때가 nack 다.
94      rv1 = i2c_read_ack();
95      rv2 = i2c_read_ack();
96      rv3 = i2c_read_nack();
97      i2c_stop();
98
99      adc_data = ((uint32_t)rv1 << 16) + ((uint32_t)rv2 << 8) + (uint32_t)rv3;
100
101     return adc_data;
102 }
```

Slave 선택 및 write 모드

Command 날림

stop

Conversion Time

다시 한번 Slave 선택 및 write

ADC 값 Read command

Slave 선택 및 read 명령어

Rv1~3 변수에 8 bit씩 단위 입력

총 24 bit 양의 데이터

Adc_data 변수에 총 변환한 데이터 저장.

ADC 값 반환.

Read digital pressure and temperature data

D1	Digital pressure value	unsigned int 32	24	0	16777216	9085466
D2	Digital temperature value	unsigned int 32	24	0	16777216	8569150

코딩(I2C 통신 설정) – ms5611.c

Ms5611_measure

main.c | ms5611.c | ms5611.h | twi.c | twi.h | uart.c | uart.h

```
102     }
103
104     void ms5611_measure (void)
105     {
106         int32_t temp_raw, press_raw, dt;
107         int64_t sens, off;
108
109         temp_raw = ms5611_conv_read_adc(CONV_D2_4096);
110         press_raw = ms5611_conv_read_adc(CONV_D1_4096);
111
112         dt = temp_raw - ((int32_t)ms5611_cal.tref << 8);
113         _ms5611_temp = 2000 + ((dt*((int64_t)ms5611_cal.tsens)) >> 23);
114         off = ((int64_t)ms5611_cal.off << 16) + (((int64_t)dt*(int64_t)ms5611_cal.tco) >> 7);
115         sens = ((int64_t)ms5611_cal.sens << 15) + ((int64_t)ms5611_cal.tcs*dt >> 8);
116         _ms5611_pres = ((((uint64_t)press_raw*sens) >> 21) - off) >> 15;
117     }
118
119     long int ms5611_getAltitude (void)
120     {
121         long int alt;
122         alt = (1 - pow(_ms5611_pres / (long int)101325, 0.1903)) / 0.0000225577;
123         return alt;
124     }
```

값 측정.

	Command byte								hex value
Bit number	0	1	2	3	4	5	6	7	
Bit name	PRM	COV	-	Typ	Ad2/ Os2	Ad1/ Os1	Ad0/ Os0	Stop	
Command									
Reset	0	0	0	1	1	1	1	0	0x1E
Convert D1 (OSR=256)	0	1	0	0	0	0	0	0	0x40
Convert D1 (OSR=512)	0	1	0	0	0	0	1	0	0x42
Convert D1 (OSR=1024)	0	1	0	0	0	1	0	0	0x44
Convert D1 (OSR=2048)	0	1	0	0	0	1	1	0	0x46
Convert D1 (OSR=4096)	0	1	0	0	1	0	0	0	0x48
Convert D2 (OSR=256)	0	1	0	1	0	0	0	0	0x50
Convert D2 (OSR=512)	0	1	0	1	0	0	1	0	0x52
Convert D2 (OSR=1024)	0	1	0	1	0	1	0	0	0x54
Convert D2 (OSR=2048)	0	1	0	1	0	1	1	0	0x56
Convert D2 (OSR=4096)	0	1	0	1	1	0	0	0	0x58
ADC Read	0	0	0	0	0	0	0	0	0x00
PROM Read	1	0	1	0	Ad2	Ad1	Ad0	0	0xA0 to 0xAE

변수 크기 선언 이유

Calculate temperature

dT	Difference between actual and reference temperature [2] $dT = D2 - T_{REF} = D2 - C5 * 2^8$	signed int 32
TEMP	Actual temperature (-40…85°C with 0.01°C resolution) $TEMP = 20°C + dT * TEMPSENS = 2000 + dT * C6 / 2^{23}$	signed int 32

Calculate temperature compensat

OFF	Offset at actual temperature [3] $OFF = OFF_{T1} + TCO * dT = C2 * 2^{16} + (C4 * dT) / 2^7$	signed int 64
SENS	Sensitivity at actual temperature [4] $SENS = SENS_{T1} + TCS * dT = C1 * 2^{15} + (C3 * dT) / 2^8$	signed int 64

코딩(I2C 통신 설정) – ms5611.c

Ms5611_measure 계산 분석.

```
112     dt = temp_raw - ((int32_t)ms5611_cal.tref << 8);
113     _ms5611_temp = 2000 + ((dt*((int64_t)ms5611_cal.tsens)) >> 23);
114     off = ((int64_t)ms5611_cal.off << 16) + (((int64_t)dt*(int64_t)ms5611_cal.tco) >> 7);
115     sens = ((int64_t)ms5611_cal.sens << 15) + ((int64_t)ms5611_cal.tcs*dt >> 8);
116     _ms5611_pres = ((((uint64_t)press_raw*sens) >> 21) - off) >> 15;
117 }
```

dT	Difference between actual and reference temperature [2] $dT = D2 - T_{REF} = D2 - C5 * 2^8$

D2 : temp_raw

곱하기 2^8 은 << 좌측 쉬프트 연산!

TEMP	Actual temperature (-40…85°C with 0.01°C resolution) $TEMP = 20°C + dT * TEMPSENS = 2000 + dT * C6 / 2^{23}$

20도 = 2000, 나누기 2^23은 >> 우측 쉬프트 연산!

OFF	Offset at actual temperature [3] $OFF = OFF_{T1} + TCO * dT = C2 * 2^{16} + (C4 * dT) / 2^7$

이번에도 곱하기 나누기 연산 주의 하여 연산.

SENS	Sensitivity at actual temperature [4] $SENS = SENS_{T1} + TCS * dT = C1 * 2^{15} + (C3 * dT) / 2^8$

마찬가지.

P	Temperature compensated pressure (10…1200mbar with 0.01mbar resolution) $P = D1 * SENS - OFF = (D1 * SENS / 2^{21} - OFF) / 2^{15}$

마지막 압력 값 계산.

코딩(I2C 통신 설정) - ms5611.c

Ms5611_getAltitude

```
118
119  long int ms5611_getAltitude (void)
120  {
121      long int alt;
122      alt = (1 - pow(_ms5611_pres / (long int)101325, 0.1903)) / 0.0000225577;
123      return alt;
124  }
```

압력 값을 이용한 대기압 값 출력.

여기서 issue)
ㄴ 131325를 double 형이 아닌 long int로 해야 숫자 출력이 제대로 될 수 있었다.(환경상 문제)

```
125
126  uint32_t ms5611_getPress (void)
127  {
128      return _ms5611_pres;
129  }
130
131  int32_t ms5611_getTemp (void)
132  {
133      return _ms5611_temp;
134  }
135
```

Press 값 리턴

Temp 값 리턴

코딩(I2C 통신 설정) - ms5611.h

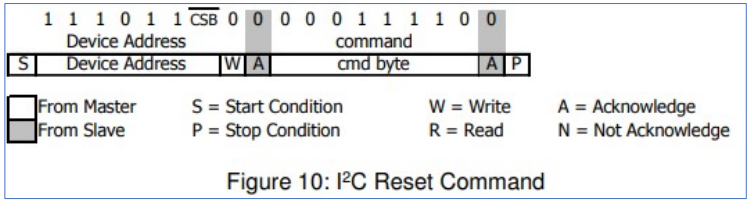


Figure 10: I²C Reset Command

main.c | ms5611.c | **ms5611.h** | twi.c | twi.h | uart.c | uart.h

```
24  // Conversion Time
25  #define OSR_256                 1
26  #define OSR_512                 2
27  #define OSR_1024                3
28  #define OSR_2048                5
29  #define OSR_4096                10
30
31  #define CONV_D1_256         0x40
32  #define CONV_D1_512         0x42
33  #define CONV_D1_1024        0x44
34  #define CONV_D1_2048        0x46
35  #define CONV_D1_4096        0x48
36  #define CONV_D2_256         0x50
37  #define CONV_D2_512         0x52
38  #define CONV_D2_1024        0x54
39  #define CONV_D2_2048        0x56
40  #define CONV_D2_4096        0x58
41
42  void ms5611_reset (void);
43  uint32_t ms5611_read_cal_reg (uint8_t reg);
44  void ms5611_init (void);
45  uint32_t ms5611_conv_read_adc (uint8_t command);
46  void ms5611_measure (void);
47  long int ms5611_getAltitude (void);
48  uint32_t ms5611_getPress (void);
49  int32_t ms5611_getTemp (void);
```

	Command byte								hex value
Bit number	0	1	2	3	4	5	6	7	
Bit name	PR M	COV	-	Typ	Ad2/ Os2	Ad1/ Os1	Ad0/ Os0	Stop	
Command									
Reset	0	0	0	1	1	1	1	0	0x1E
Convert D1 (OSR=256)	0	1	0	0	0	0	0	0	0x40
Convert D1 (OSR=512)	0	1	0	0	0	0	1	0	0x42
Convert D1 (OSR=1024)	0	1	0	0	0	1	0	0	0x44
Convert D1 (OSR=2048)	0	1	0	0	0	1	1	0	0x46
Convert D1 (OSR=4096)	0	1	0	0	1	0	0	0	0x48
Convert D2 (OSR=256)	0	1	0	1	0	0	0	0	0x50
Convert D2 (OSR=512)	0	1	0	1	0	0	1	0	0x52
Convert D2 (OSR=1024)	0	1	0	1	0	1	0	0	0x54
Convert D2 (OSR=2048)	0	1	0	1	0	1	1	0	0x56
Convert D2 (OSR=4096)	0	1	0	1	1	0	0	0	0x58
ADC Read	0	0	0	0	0	0	0	0	0x00
PROM Read	1	0	1	0	Ad2	Ad1	Ad0	0	0xA0 to 0xAE

main.c | ms5611.c | **ms5611.h** | twi.c | twi.h | uar

```
1
2   #ifndef __MS5611_H__
3   #define __MS5611_H__
4
5   #include <avr/io.h>
6   #include <util/delay.h>
7   #include <stdio.h>
8   #include <stdlib.h>
9   #include <string.h>
10  #include <math.h>
11
12  #define I2C_WRITE               0
13  #define I2C_READ                1
14
15  #define MS5611_ADDR             0x76
16
17  #define RESET                   0x1E
18  #define CMD_ADC_READ            0x00
19  // reg << 1 이니까 2가 되고, 0xA2가 되지. ms5611 10 page 데이터 시트 표, 6,8,11(read sequence) page 참고(6가지의 값 세팅). 0번 비트는 건드릴 필요 없기에 shift 했다.
20  // 캘리브리에이션이 뭔가? 바깥의 노이즈 성분을 계산해서 성능 개선
21  #define MS5611_CMD_PROM(reg)    (0xA0 + ((reg) << 1))
22  #define CMD_PROM_READ           0xA0
23
```

코딩(I2C 통신 설정) – uart.c

main.c | ms5611.c | ms5611.h | twi.c | twi.h | **uart.c** | uart.h

```
1
2   #define F_CPU 16000000UL
3   //#define F_CPU 8000000UL
4   #include "uart.h"
5
6   #define sbi(PORTX, BitX) (PORTX |= (1<<BitX))
7   #define cbi(PORTX, BitX) (PORTX &= ~(1<<BitX))
8
9   #define UART_BUFLEN 10
10
11  void UART_INIT(void){
12      sbi(UCSR0A, U2X0); // U2X0 = 1 --> Baudrate 9600 = 207
13
14      UBRR0H = 0x00;
15      UBRR0L = 207;   // Baudrate 9600
16
17      UCSR0C |= 0x06; // 1stop bit, 8bit data
18
19      sbi(UCSR0B, RXEN0);  // enable receiver and transmitter
20      sbi(UCSR0B, TXEN0);
21  }
22
23  unsigned char UART_receive(void)
24  {
25      while(!(UCSR0A & (1<<RXC0))); // wait for data to be received
26      return UDR0; // get and return received data from buffer
27  }
```

UART_init

Logs & others

UART 코딩 부분은 앞서 했던 부분들과 크게 다를게 없으므로

자세한 설명은 생략.

Baudrate 9600 설정.

1 spot bit, 8 bit data

TX, RX enable

코딩(I2C 통신 설정) - uart.c

main.c | ms5611.c | ms5611.h | twi.c | twi.h | **uart.c** | uart.h

UART_receive

```
unsigned char UART_receive(void)
{
    while(!(UCSR0A & (1<<RXC0))); // wait for data to be received
    return UDR0; // get and return received data from buffer
}
```

받을 data가 있는지 확인
UDR에 저장된 데이터 return.

UART_transmit

```
void UART_transmit( char data)
{
    while(!(UCSR0A & (1<<UDRE0))); // wait for empty transmit buffer
    UDR0 = data; // put data into buffer, sends the data
}
```

인자로 받은 data를
송신 준비가 되었다면
UDR0 에 송신.

UART_string_transmit

```
void UART_string_transmit(char *string)
{
    while(*string != '\0')
    {
        UART_transmit( *string );
        string++;
    }
}

void UART_PRINT(char *name, long val)
{
    char debug_buffer[UART_BUFLEN] = {'\0'};

    UART_string_transmit(name);
```

캐릭터 포인터 변수로 인자를 받은 뒤
위의 char 단위 송신 함수를 활용해
널 까지 while문을 반복하여 문장 출력.

코딩(I2C 통신 설정) – uart.c

main.c | ms5611.c | ms5611.h | twi.c | twi.h | **uart.c** | uart.h

UART_PRINT

```
45  void UART_PRINT(char *name, long val)
46  {
47      char debug_buffer[UART_BUFLEN] = {'\0'};
48
49      UART_string_transmit(name);
50      UART_string_transmit(" = ");
51
52      ltoa((val), debug_buffer, UART_BUFLEN);
53      UART_string_transmit(debug_buffer);
54      UART_string_transmit("\n");
55  }
```

앞선 UART 코드 들과 다른 부분.

Deug_buffer 생성.

"Name =" 을 송신.

Int형 val 값을 아스키 값으로 변경 후 debug_buffer에 UART_BUFLEN 크기 만큼 변환.

계산 한 온도나 대기압 값 등의 int형 숫자 값은 아스키 값으로 변경 후 출력하기 위함.

uartTxChar

```
56  int usartTxChar(char ch, FILE *fp) {  // for printf
57      while (!(UCSR0A & (1 << UDRE0)));
58
59      UDR0 = ch;
60
61      return 0;
62  }
63
64  void uart_print_8bit_num(uint8_t no)
65  {
66      char num_string[4] = "0";
67      int i, index = 0;
68
69      if (no > 0)
```

파일 포인터의 char 인자 값을 받고,
송신할 준비가 되었다면
받은 ch 값을 UDR0에 저장.

코딩(I2C 통신 설정) – uart.c

main.c | ms5611.c | ms5611.h | twi.c | twi.h | **uart.c** | uart.h

```
61      return 0;
62  }
63
64  void uart_print_8bit_num(uint8_t no)
65  {
66      char num_string[4] = "0";
67      int i, index = 0;
68
69      if (no > 0)
70      {
71          for (i = 0; no != 0; i++)
72          {
73              num_string[i] = no % 10 + '0';
74              no = no / 10;
75          }
76
77          num_string[i] = '\0';
78          index = i - 1;
79      }
80
81      for (i = index; i >= 0; i--)
82      {
83          UART_transmit(num_string[i]);
84      }
85  }
86
```

Uart_print_8bit_num

인자로 8bit 숫자 값을 받는다.

Char형 배열 num_string[4]의 배열을 생성.
Int형 변수 I, index

No 값이 0 보다 클 경우

For문을 통해 0 이 아닐 때 까지 값을
자릿수 별로 string[i]에 저장.

이것을 하나씩 역순으로 uart_transmit을 통해 송신.

코딩(I2C 통신 설정) – uart.h

main.c | ms5611.c | ms5611.h | twi.c | twi.h | uart.c | **uart.h**

```
1
2   #ifndef UART_H_
3   #define UART_H_
4
5   #include <avr/io.h>
6   #include <util/delay.h>
7   #include <stdio.h>
8   #include <stdlib.h>
9   #include <string.h>
10
11  void UART_INIT(void);
12  unsigned char UART_receive(void);
13  void UART_transmit( char data);
14  void UART_string_transmit(char *string);
15  void UART_PRINT(char *name, long val);
16  int usartTxChar(char ch, FILE *fp);  // for printf
17  void uart_print_8bit_num(uint8_t);
18
19  #endif /* UART_H_ */
20
```

Uart 통신 헤더 파일 선언.

코딩(I2C 통신 설정) – main.c

main.c | ms5611.c | ms5611.h | twi.c | twi.h | uart.c | uart.h

```
2    * TWI_MS5611.c
3    *
4    * Created: 2020-05-09 오후 4:13:36
5    * Author : user
6    */
7   // SCL-RX, SDA - TX : H/W 연결시
8   #define F_CPU 16000000UL
9   //#define F_CPU 8000000UL
10
11  #include "uart.h"
12  #include "twi.h"
13  #include "ms5611.h"
14
15  int main(void)
16  {
17      //extern FILE *fdevopen(int (*__put)(char, FILE*), int (*__get)(FILE*));
18      // extern 외부 파일에 선언된 전역 데이터를 가져올 때 사용 한다.
19      // FILE * 를 리턴하고
20      // 입력으로 int (*__put)(char, FILE*) 형태의 함수 포인터, int 형 반환하고 char와 FILE* 입력으로 받는 put 은 이름인 포인터
21      // int (*__get)(FILE*)  형태의 함수 포인터 즉, 총 2개 선언됨.
22
23      // 리눅스 파일디스크립터 리다이렉션과 같은 개념임
24      // 모니터 출력 대신 usartTxchar가 동작 하다고 보면 된다.
25      // 즉, usart에 의한 출력이 putty 같은 터미널 프로그램들은 이를 반영하여 결과를 출력 할 것이다.
26      // 즉, 키보드 입력 -> usart 출력 -> 화면에 반영 되서 디스플레이. (요런 느낌의 절차)
27      // https://github.com/vancegroup-mirrors/avr-libc/blob/master/avr-libc/libc/stdio/fdevopen.c
28      FILE* fpStdio = fdevopen(usartTxChar, NULL); // for printf
```

Uart, twi, ms5611 관련 헤더 선언

코딩(I2C 통신 설정) - main.c

```
15  int main(void)
16  {
17      //extern FILE *fdevopen(int (*__put)(char, FILE*), int (*__get)(FILE*));
18      // extern 외부 파일에 선언된 전역 데이터를 가져올 때 사용 한다.
19      // FILE * 를 리턴하고
20      // 입력으로 int (*__put)(char, FILE*) 형태의 함수 포인터, int 형 반환하고 char와 FILE* 입력으로 받는 put 은 이름인 포인터
21      // int (*__get)(FILE*)  형태의 함수 포인터 즉, 총 2개 선언됨.
22
23      // 리눅스 파일디스크립터 리다이렉션과 같은 개념임
24      // 모니터 출력 대신 usartTxchar가 동작 하다고 보면 된다.
25      // 즉, usart에 의한 출력이 putty 같은 터미널 프로그램들은 이를 반영하여 결과를 출력 할 것이다.
26      // 즉, 키보드 입력 -> usart 출력 -> 화면에 반영 되서 디스플레이. (요런 느낌의 절차)
27      // https://github.com/vancegroup-mirrors/avr-libc/blob/master/avr-libc/libc/stdio/fdevopen.c
28      FILE* fpStdio = fdevopen(usartTxChar, NULL); // for printf
```

함수 포인터 fdevopen 관련 설명.

일단 fdevopen은

`Extern FILE *fdevopen(int (*__put)(char, FILE*), int(*__get)(FILE*))` 가 기본 stdio.h 헤더 상에 선언 되어 있다.

반환형 / 포인터 이름 / 입력받는 인자

코딩(I2C 통신 설정) - main.c

main.c | ms5611.c | ms5611.h | twi.c | twi.h | uart.c | uart.h

```
22
23      // 리눅스 파일디스크립터 리다이렉션과 같은 개념임
24      // 모니터 출력 대신 usartTxchar가 동작 하다고 보면 된다.
25      // 즉, usart에 의한 출력이 putty 같은 터미널 프로그램들은 이를 반영하여 결과를 출력 할 것이다.
26      // 즉, 키보드 입력 -> usart 출력 -> 화면에 반영 되서 디스플레이. (요런 느낌의 절차)
27      // https://github.com/vancegroup-mirrors/avr-libc/blob/master/avr-libc/libc/stdio/fdevopen.c
28      FILE* fpStdio = fdevopen(usartTxChar, NULL); // for printf
29      int i;
30      unsigned long D1;
31      unsigned long D2;
32      unsigned int C[8];
33
34      UART_INIT();
35      UART_string_transmit("uart init ok\n");
36
37      // TWI(Two-Wire Serial Interface)
38      // 필립스에서 만든 IIC(Inter Integrated Circuit)라는 용어를 많이 사용함
39      // I2C(아이스퀘어씨) 라고 부르고 발음함
40      // 두 개의 선을 이용해서 데이터를 받음
41      // 하나는 SDA로 데이터를 주는 선이며,
42      // 또 다른 하나는 SCL로 클럭을 제공하는 선이다.
43      // SCL이 필요한 이유는 동기화 시켜서 어디가 시작이고 어디가 끝인질 알아야 하기 때문
44      // 데이터를 주고 받는 선이 하나 밖에 없으므로 반이중(Half Duplex) 통신이다.
45      // SPI와 마찬가지로 Master-Slave 통신 방식을 가진다.
46      // 근래 들어서 SPI Master/Slave - Slave/Master
47      // 즉, 장치가 고정적으로 Master만 되거나 Slave만 되는 것이 아닌 모드 전환이 가능해 졌다.
```

변수 I, D1, D2
C[8] 배열

UART 초기화,
초기화 완료 시 메시지 전송

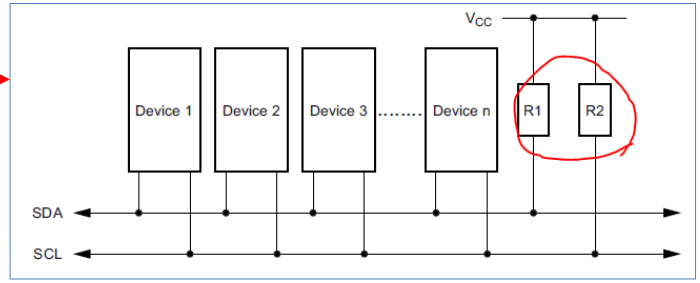
I2C에 대한 정의 내용 정리
- ㄴ 동작원리(SDA,SCL)
- ㄴ 통신 방식(반 이중 통신)
- ㄴ Master 하나에 여러 Slave 통신 가능
  (동시는 아니고 하나씩)
- ㄴ 참고로 UART는 I2C 처럼 여러 Slave를 병렬식으로 통신이 되지 않는다.
  (오직 1대1 로만)

코딩(I2C 통신 설정) - main.c

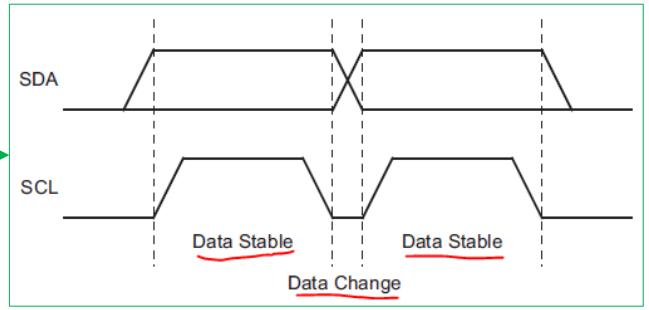
main.c | ms5611.c | ms5611.h | twi.c | twi.h | uart.c | uart.h

```
48
49      // AVR m328p datasheet 173,4,5,6 page에 설명 있다.
50      // SDA, SCL은 IDLE (유휴 상태, 아무것도 안하는 상태)상태에서 pull-up 저항인 High를 유지해야만 한다.
51      // SDA 데이터는 SCL이 High 로 동일한 상태를 유지해야 하고
52      // SCL이 LOW 인 상태에서만 SDA 신호가 바뀔 수 있다.(Low로 가면서 변경 가능 해진다)
53
54      // TWI 통신을 시작하기 위해 START 신호를 보내고
55      // 그 다음은 어떤 슬레이브와 통신할 것인지 알려주기 위해 Slave 주소를 보낸다.(이 부분이 중요)
56      // TWI 통신은 항상 MSB 비트 부터 보내도록 되어 있다.
57      // Master에서 Slave 주소를 내보낼 때 읽기 동작을 할 것인지, 쓰기를 할 것인지 미리 알려줘야 한다.
58      // 읽기 동작은 '1'을 보내고 쓰기 동작은 '0'을 보낸다.
59
60      // 선택된 Slave에서 주소를 제대로 인식 시켰다면
61      // 다음 SCL 클럭에서 ACK 신호를 보내줘야 한다.
62      // 제대로 수신하였음을 알려주기 위해 SDA 신호를 LOW로 만든다.
63      // 만약 Master에서 주소 값을 보낸 후 ACK 신호를 검출 하였을 때
64      // 이 값이 0이 아니면 Slave에서 제대로 주소를 받지 못했다고 판단 한다.
65
```

M328p datasheet 상에 I2C 통신 관련 설명이 나온다.



R1, R2와 같은 pull up 저항을 사용해서 IDLE 상태는 High를 유지한다.



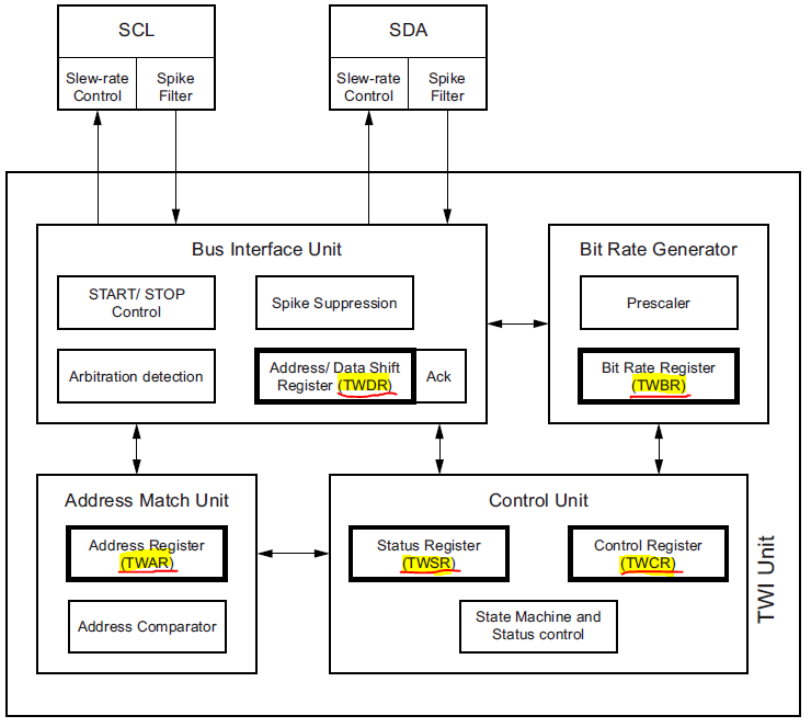
START 신호 -> Slave 주소 전송 -> Slave에서 주소 인식 -> ACK 신호 응답
- ㄴ 항상 MSB 비트 부터
- ㄴ 읽기(1), 쓰기(0) 동작 차이

코딩(I2C 통신 설정) - main.c

```
53
54      // TWI 통신을 시작하기 위해 START 신호를 보내고
55      // 그 다음은 어떤 슬레이브와 통신할 것인지 알려주기 위해 Slave 주소를 보낸다.(이 부분이 중요)
56      // TWI 통신은 항상 MSB 비트 부터 보내도록 되어 있다.
57      // Master에서 Slave 주소를 내보낼 때 읽기 동작을 할 것인지, 쓰기를 할 것인지 미리 알려줘야 한다.
58      // 읽기 동작은 '1'을 보내고 쓰기 동작은 '0'을 보낸다.
59
60      // 선택된 Slave에서 주소를 제대로 인식 시켰다면
61      // 다음 SCL 클럭에서 ACK 신호를 보내줘야 한다.
62      // 제대로 수신하였음을 알려주기 위해 SDA 신호를 LOW로 만든다.
63      // 만약 Master에서 주소 값을 보낸 후 ACK 신호를 검출 하였을 때
64      // 이 값이 0이 아니면 Slave에서 제대로 주소를 받지 못했다고 판단 한다.
65
66      // 여기서 블록도를 봐야 하는데, page 179 에서 진하게 되어 있는 부분이 중요하다
67      // 데이터시트 상의 굵은 표시가 되어 있는 것들이 CPU로 제어 할 수 있는 레지스터들이다.
68      // 읽기/쓰기를 위한 TWDR 레지스터/
69      // TWI 제어를 위한 TWCR, TWSR 레지스터
70      // TWI 통신 속도를 위한 TWBR 레지스터
```

```
72      // TWAR 레지스터는 Slave로 동작 될 때 사용하는 주소를 써놓음(마스터 동작에선 필요 없음)
73      // 마지막 경우는 사실상 누군가 AVR을 Slave로 쓰고 싶을 때 경우를 말하는 건데, 그래서 위에 4가지 레지스터를 다루는게 더 중요함.
74      i2c_init();
```

Figure 21-9. Overview of the TWI Module



코딩(I2C 통신 설정) - main.c

main.c | ms5611.c | ms5611.h | twi.c | twi.h | uart.c | uart.h

```
66      // 여기서 블록도를 봐야 하는데, page 179 에서 진하게 되어 있는 부분이 중요하다
67      // 데이터시트 상의 굵은 표시가 되어 있는 것들이 CPU로 제어 할 수 있는 레지스터들이다.
68      // 읽기/쓰기를 위한 TWDR 레지스터/
69      // TWI 제어를 위한 TWCR, TWSR 레지스터
70      // TWI 통신 속도를 위한 TWBR 레지스터
71
72      // TWAR 레지스터는 Slave로 동작 될 때 사용하는 주소를 써놓음(마스터 동작에선 필요 없음)
73      // 마지막 경우는 사실상 누군가 AVR을 Slave로 쓰고 싶을 때 경우를 말하는 건데, 그래서 위에 4가지 레지스터를 다루는게 더 중요함.
74      i2c_init();
75      UART_string_transmit("i2c init ok\n");
76
77
78      ms5611_init();
79      UART_string_transmit("ms5611 init ok\n");
80      _delay_ms(1000);
81
82      while (1)
83      {
84          ms5611_measure();
85          printf("press : %u\n\r" , ms5611_getPress());
86          printf("temp : %d\n\r" , ms5611_getTemp());
87          printf("alt : %ld\n\r" , (long int)ms5611_getAltitude());
88          _delay_ms(1000);
89      }
90  }
91
```

I2c 통신 설정 init
I2c init 메시지 uart로 전송

Ms5611 센서 관련 설정 init
Ms5611 센서 init 메시지 uart 전송

Ms5611 센서 값 측정
측정 값 출력 (압력, 온도, 대기압 값 출력)

여기서 주의 했어야 했던 사항
- 대기압 값 출력이 double 형 이었는데, putty상에서 값이 출력되지 않아 long int로 형 변환 후 제대로 출력 됨을 확인 할 수 있었다. (리눅스 환경상 문제 일 듯 하다)

질문(1)

- URL 주소
https://cafe.naver.com/eddicorp/74

TWI 통신 ACK NACK 관련 질문

질문

pti1360 일반멤버
2021.07.15. 16:41 조회 10

댓글 2 URL 복

main.c | ms5611.c | ms5611.h | twi.c | twi.h | uart.c | uart.h

```
126  unsigned char i2c_read_ack (void)
127  {
128      // ACK 신호 전송
129      TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWEA);
130      // 올바르게 처리되었는지 대기
131      while (!(TWCR & (1 << TWINT)))
132      {
133          ;
134      }
135      // 읽은 데이터가 TWDR에 들어오기 때문에 이것을 리턴.
136      return TWDR;
137  }
138
139  unsigned char i2c_read_nack (void)
140  {
141      // 왜 이렇게 두개가 들어 올까? PROM이 16비트 데이터 이므로 (readAck + readNak)
142      TWCR = (1 << TWINT) | (1 << TWEN);
143      // 잘 수신하였는지 대기
144      while (!(TWCR & (1 << TWINT)))
145      {
146          ;
147      }
148      // 마지막 8 비트 수신
149      // 만약에 24 비트면 Nak를 한번더 해야 겠지 (nak는 연속적인거 뒤에 쭉쭉)
150      return TWDR;
151  }
```

수업 시간에 들었던 내용을 주석으로 남긴 것인대요. ACK가 NACK가 이렇게 두개로 나뉘어서 들어오는 방식의 이유가 atmega에서 TWI 통신 자체가 8비트 타입이고 PROM이 16비트 데이터라고 들었었던 것 같습니다.

여기서 PROM이 16비트라는 말이 slave로 사용하고 있는 센서의 데이터 길이가 16 bit 이란 뜻이어서 atmega의 내부적 TWI 통신이 8 bit 형식이니 두개씩 들어 온다고 의미 인걸까요?

그리고 여기서 두개씩이라는 말이 ACK NACK 이렇게 두개를 의미 하는 걸까요?
아니면 16 bit의 경우 ACK ACK NACK 가 되는 거여서, ACK가 두개가 된다. 라는 의미 인걸까요??
(NACK는 데이터 전송의 끝?을 의미 하는 듯 해서요~)

답변

댓글 등록순 최신순

링크쌤
넵 맞습니다.
한 번에 받는 것이 8비트니 2번 받아서 16비트를 만든다고 보면 되겠습니다.

최종 바이트에서 NACK를 보내게 됩니다.

TCP에서도 마찬가지로 사용하는 개념인데 ACK는 보통 잘 수신하였음
NACK는 도중에 문제가 생김 혹은 끝났음을 의미합니다.
비슷한 개념으로 접근해도 좋을것 같습니다.
2021.07.15. 21:31 답글쓰기

pti1360 작성자
옙 깔금한 정리 감사드립니다
2021.07.15. 21:58 답글쓰기

질문(2)

- URL 주소
https://cafe.naver.com/eddicorp/75

ms5611 센서 관련 header 관련 질문

질문

pti1360 일반멤버
2021.07.15. 19:20 조회 9

댓글 1 URL 복사

main.c | ms5611.c | ms5611.h | twi.c | twi.h | uart.c | uart.h

```
#ifndef __MS5611_H__
#define __MS5611_H__

#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define I2C_WRITE               0
#define I2C_READ                1

#define MS5611_ADDR             0x76

#define RESET                   0x1E
#define CMD_ADC_READ            0x00
// reg << 1 이니까 2가 되고, 0xA2가 되지. ms5611 10 page 데이터 시트 표, 6,8,11(read sequence) page 참고(6가지의 값 세팅). 0번 비트는 건드릴 필요 없기에 shift 했다.
// 캘리브레이션이 뭔가? 바깥의 노이즈 성분을 계산해서 성능 개선
#define MS5611_CMD_PROM(reg)    (0xA0 + ((reg) << 1))
#define CMD_PROM_READ           0xA0
```

COMMANDS

Size of each command is 1 byte (8 bits) as described in the table below. After ADC read commands the device will return 24 bit result and after the PROM read 16bit result. The address of the PROM is embedded inside of the PROM read command using the a2, a1 and a0 bits.

	Command byte								hex value
Bit number	0	1	2	3	4	5	6	7	
Bit name	PRM	COV	-	Typ	Ad2/Os2	Ad1/Os1	Ad0/Os0	Stop	
Command									
Reset	0	0	0	1	1	1	1	0	0x1E
Convert D1 (OSR=256)	0	1	0	0	0	0	0	0	0x40
Convert D1 (OSR=512)	0	1	0	0	0	0	1	0	0x42
Convert D1 (OSR=1024)	0	1	0	0	0	1	0	0	0x44
Convert D1 (OSR=2048)	0	1	0	0	0	1	1	0	0x46
Convert D1 (OSR=4096)	0	1	0	0	1	0	0	0	0x48
Convert D2 (OSR=256)	0	1	0	1	0	0	0	0	0x50
Convert D2 (OSR=512)	0	1	0	1	0	0	1	0	0x52
Convert D2 (OSR=1024)	0	1	0	1	0	1	0	0	0x54
Convert D2 (OSR=2048)	0	1	0	1	0	1	1	0	0x56
Convert D2 (OSR=4096)	0	1	0	1	1	0	0	0	0x58
ADC Read	0	0	0	0	0	0	0	0	0x00
PROM Read	1	0	1	0	Ad2	Ad1	Ad0	0	0xA0 to 0xAE

위에 보이는 그림의 MS5611_CMD_PROM(reg)관련 질문 입니다.
위에서 보면 해당 define이 (0xA0 + ((reg << 1)) 로 되어 있고
reg << 1 은 0000 이 0010 이 되어 2가 되며,
0xA0 + 2 가 되어 결국은 0xA2의 의미가 되는 것 같습니다..

여기서 궁금한 점은 reg 라는 것을 다른 곳에서 선언 한 적이 없는데 자동적으로 0000 이 초기 값이 되는걸까요?
그리고 위와 같이 데이터 시트를 보면 COMMANDS 에 PROM Read 가 A0 ~ AE 로 되어 있는데,
위와 같이 PROM(reg) 밑 줄에서 A0은 따로 선언하고,
위에서는 reg 라는 것을 이용해서 A2를 만들어서 따로 선언하는 이유가 있을 까요??
(만약 A2를 이용해야 한다면, 그냥 0xA2를 선언한다던지 하는 거랑 어떤 점이 다른건지 궁금합니다.)

추가적으로, PROM Read가 A0~AE라고 되어 있는데, 현재는 A0과 A2만 사용하게 된 것이고,
A1은 넘어간 이유가 있을까요?

마지막으로, 필요 하다면 다음 A3~E 더 선언해서 사용하면 되는 걸까요?

댓글 등록순 최신순

답변

링크쌤
함수형 매크로로 reg는 우리가 입력하는 값에 따라 변합니다.
하드웨어 상태를 코드에 그대로 적용해놨다고 보면 됩니다.
사실 이러한 부분은 다형성을 추구하기 위해서는 좀 더 추상화될 필요가 있긴합니다.
어떤 하드웨어던지 커버하겠다는 의도를 가지고 있다면 말이죠.
2021.07.15. 21:34 답글쓰기

추가이해
ㄴ 이 부분에 대한 추가적 이해가 필요하다.

A0 다음 사용을 왜 A2가 되게 했을까?
결론적으로 PROM이 16bit를 반환하기 때문이다.

주소 값 하나 움직일 때 1byte. 두개 움직일 때 2 byte(16비트)
이게 왜 주소 값 하나 일 움직일 때 마다 1byte(8bit) 일까? -> AVR이 8bit 체제이기 때문이다.

이해를 돕기 위해 어셈블리 당시 포인터 값의 이동 할 때 가상 메모리 주소가 이동시에 8 byte(64bit 체계) 값 씩 움직였던 것을 떠올리면 이해가 더 잘 된다!

COMMANDS

Size of each command is 1 byte (8 bits) as described in the table below. After ADC read commands the device will return 24 bit result and after the PROM read 16bit result. The address of the PROM is embedded inside of the PROM read command using the a2, a1 and a0 bits.

	Command byte								hex value
Bit number	0	1	2	3	4	5	6	7	
Bit name	PRM	COV	-	Typ	Ad2/Os2	Ad1/Os1	Ad0/Os0	Stop	
Command									
Reset	0	0	0	1	1	1	1	0	0x1E
Convert D1 (OSR=256)	0	1	0	0	0	0	0	0	0x40
Convert D1 (OSR=512)	0	1	0	0	0	0	1	0	0x42
Convert D1 (OSR=1024)	0	1	0	0	0	1	0	0	0x44
Convert D1 (OSR=2048)	0	1	0	0	0	1	1	0	0x46
Convert D1 (OSR=4096)	0	1	0	0	1	0	0	0	0x48
Convert D2 (OSR=256)	0	1	0	1	0	0	0	0	0x50
Convert D2 (OSR=512)	0	1	0	1	0	0	1	0	0x52
Convert D2 (OSR=1024)	0	1	0	1	0	1	0	0	0x54
Convert D2 (OSR=2048)	0	1	0	1	0	1	1	0	0x56
Convert D2 (OSR=4096)	0	1	0	1	1	0	0	0	0x58
ADC Read	0	0	0	0	0	0	0	0	0x00
PROM Read	1	0	1	0	Ad2	Ad1	Ad0	0	0xA0 to 0xAE

질문.

1)

ANALOG DIGITAL CONVERTER (ADC)

Parameter	Symbol	Conditions	Min.	Typ.	Max	Unit
Output Word				24		bit
Conversion time	t_c	OSR				ms
		4096	7.40	8.22	9.04	
		2048	3.72	4.13	4.54	
		1024	1.88	2.08	2.28	
		512	0.95	1.06	1.17	
		256	0.48	0.54	0.60	

여기서 말하는 컨디션은 ADC의 계산 샘플링 능력을 말하는 걸까요?
Ex) 10 bit -> 1024

```
118
119     long int ms5611_getAltitude (void)
120     {
121         long int alt;
122         alt = (1 - pow(_ms5611_pres / (long int)101325, 0.1903)) / 0.0000225577;
123         return alt;
124     }
```

2) 대기압 구하는 공식은 데이터 시트 상으로는 보이지 않는데..
일반적인 공식 인걸까요??

3) 계산 과정중 (101325, 0.1903) 은 어떤 의미 일까요?
pow 함수 사용법을 보고 참고 했을 때는

$$(1-(\frac{ms5611pres}{101325})^{0.1903})/0.0000225577$$

가 되는게 맞을까요? Ex) pow(2,3) = 2^3