AVR - HW10

임베디드스쿨1기

Lv1과정

2020. 11. 20

박하늘

1. [Review] I2C

1) 특징 및 장단

Features

- Simple yet powerful and flexible communication interface, only two bus lines needed
- Both master and slave operation supported
- Device can operate as transmitter or receiver
- 7-bit address space allows up to 128 different slave addresses
- Multi-master arbitration support
- Up to 400kHz data transfer speed
- Slew-rate limited output drivers
- Noise suppression circuitry rejects spikes on bus lines
- Fully programmable slave address with general call support
- Address recognition causes wake-up when AVR® is in sleep mode
- Compatible with Phillips' I^2C protocol

장점:
- HW설치 용이, 총 3개선을 분기 시킴. 총 128(=2^7)개의 Slave 장치 연결 가능
- 비트레이트 사전 약속 불필요
- address 수신시 sleep-mode 에서 wake-up 지원
- slew-rate limiters 내장 (급 격한 전압 변동을 늦춰줌)

단점:
- 저속 통신 (단 하나의 data선으로 arbitration, target selection등 까지 수행 필요)
- S/W 의존적임 (장치 Select해야함)

1. [Review] I2C

2) ACK, NACK, Pull-up

2-1) ACK

매 8bit 전송후마다 Receiver는 SDA를 Low로 만들어 ACK를 발생시켜야 한다.

2-2) NACK(ACK비트가 발생하지 않는 경우)

- Receiver가 제대로 수신을 하지 못해 ACK를 발생시킬 타이밍을 놓친 경우
- Reveiver가 더이상 수신을 원치 않거나, 의도적으로 중단하여 ACK를 발생시키지 않은 경우

2-3) Pull-Up과 ACK

따로 제어를 하지 않으면 SDA는 Pull-up되어 High를 유지하기 때문에,
SDA가 Low라면 의도된 당겨진 상태를 의미 → ACK로 간주
SDA가 High라면 의도되지 않았을 수 있음 의미 → NACK로 간주
(수신 실패로 ACK 발생시킬 타이밍을 놓친 경우 NACK가 High레벨에 적합)

2. twi.c 분석

```c
#include "twi.h"

#define TW_STS      0xF8

#define TWI_SCL     0x20
#define TWI_SDA     0x10
#define TWI_RD      0x01
#define TWI_WR      0x00

#define TWI_START         0x08
#define TWI_RESTART         0x10

/* Master Transmitter */
#define TWI_MT_SLA_ACK      0x18
#define TWI_MT_SLA_NACK      0x20
#define TWI_MT_DATA_ACK      0x28
#define TWI_MT_DATA_NACK    0x30
#define TWI_MT_ARB_LOST      0x38

/* Master Receiver */
#define TWI_MR_ARB_LOST      0x38
#define TWI_MR_SLA_ACK      0x40
#define TWI_MR_SLA_NACK      0x48
#define TWI_MR_DATA_ACK      0x50
#define TWI_MR_DATA_NACK    0x58

void i2c_init(void)
{
    /*initialize TWI clockP 100khz clock, TWPS = 0 => prescaler = 1*/
    TWSR = 0x00;
    TWBR = 12;/*((F_CPU / SCL_CLOCK) - 16)/2 = TWBR, F_CPU = 4Mhz, SCL_CLOCK = 100K -- >12 */
}
```

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	

- **Bits 7..3 – TWS: TWI Status**

These 5 bits reflect the status of the TWI logic and the 2-wire serial bus. The different status codes are described later in this section. Note that the value read from TWSR contains both the 5-bit status value and the 2-bit prescaler value. The application designer should mask the prescaler bits to zero when checking the status bits. This makes status checking independent of prescaler setting. This approach is used in this datasheet, unless otherwise noted.

- **Bit 2 – Res: Reserved Bit**

This bit is reserved and will always read as zero.

- **Bits 1..0 – TWPS: TWI Prescaler Bits**

These bits can be read and written, and control the bit rate prescaler.

Table 21-8. TWI Bit Rate Prescaler

TWPS1	TWPS0	Prescaler Value
0	0	1

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

2. twi.c 분석

```
unsigned char i2c_start(unsigned char address)
{
    uint8_t   twst;

    // send START condition
    TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

    // wait until transmission completed
    while(!(TWCR & (1<<TWINT)));

    // check value of TWI Status Register. Mask prescaler bits.
    twst = TWSR & 0xF8;
    if ( (twst != TWI_START) && (twst != TWI_RESTART)) return 1;

    // send device address
    TWDR = address;
    TWCR = (1<<TWINT) | (1<<TWEN);

    // wail until transmission completed and ACK/NACK has been received
    while(!(TWCR & (1<<TWINT)));

    // check value of TWI Status Register. Mask prescaler bits.
    twst = TWSR & 0xF8;
    if ( (twst != TWI_MT_SLA_ACK) && (twst != TWI_MR_SLA_NACK) ) return 1;

    return 0;
}
unsigned char i2c_rep_start(unsigned char address)
{

    return i2c_start(address);

}
```

1. start 조건을 보낸다. / TWI 의 전반적인 동작을 제어

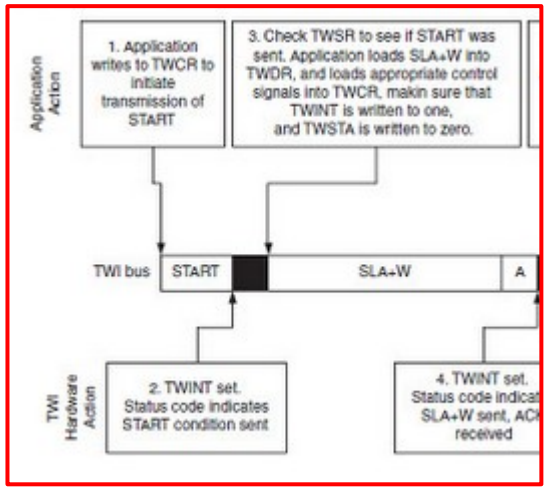
2. TWINT가 1로 설정되기를 기다린다.

3. 11111000(하위 3BIT 마스크)
상위 5BIT 상태 체크 후 Start, Restart가 아니면 1

4. TWDR레지스터에 슬레이브의 주소를 써넣는다.
TWCR레지스터는 TWINT를 클리어 시키려고 새로 써넣는다.

5. TWINT가 1로 설정되기를 기다린다.

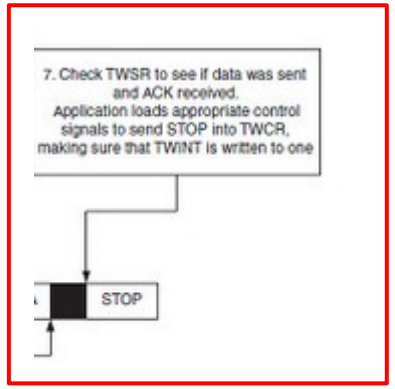
6. 상위 5BIT 상태 체크 후
ack, nack가 아니면 1



2. twi.c 분석

```
void i2c_stop(void)
{

    TWCR = (1<<TWINT) | (1<<TWEN) | ( 1<< TWSTO);

    while(TWCR & (1<<TWSTO));

}
unsigned char i2c_write(unsigned char data)
{

    uint8_t twst;

    TWDR = data;
    TWCR = (1 << TWINT) | (1<<TWEN);

    while(!(TWCR & (1<<TWINT)));

    twst = TWSR & 0xF8;
    if((twst != TWI_MT_DATA_ACK)) return 1;

    return 0;
}
/*************************************************************************
Read one byte from the I2C device, request more data from device

Return : byte read from I2C device
*************************************************************************/
unsigned char i2c_readAck(void)//데이터 제대로 받을때 ACK
{

    TWCR = (1<<TWINT) | (1<<TWEN) | ( 1<< TWEA);
    while(!(TWCR & ( 1<<TWINT)));

    return TWDR;
}
```

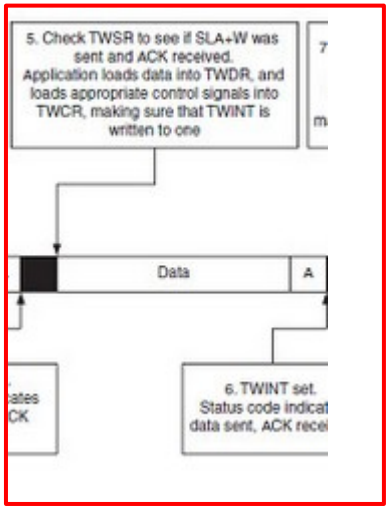
6. STOP 조건을 만든다.
TWSTO가 1이 설정되기를 기다린다.



7. TWI모듈의 송수신모드에서 다음에 Write/Read할 바이트를 저장
TWDR 레지스터에 슬레이브의 주소를 써넣는다.
TWCR 레지스터는 TWINT를 클리어 시키기 위해 새로 써넣는다.

8. TWINT를 1로 설정되기를 기다린다 .

9. 상위 5BIT 상태 체크 후 ack 이면 0, 아니면 1



10. ACK조건을 만든다.
TWINT가 1이 설정되기를 기다린다.

2. twi.c 분석

```
/*************************************************************************
Read one byte from the I2C device, read is followed by a stop condition

Return : byte read from I2C device
*************************************************************************/
unsigned char i2c_readNak(void) //데이터를 제대로 받지 못하였을때
{
    TWCR = (1<<TWINT)|(1<<TWEN);
    while(!(TWCR & ( 1<<TWINT)));

    return TWDR;
}
```

10. NACK조건을 만든다.
TWINT가 1이 설정되기를 기다린다.

- TWINT 0 → 1 일때, start condition
- TWINT 1이 될 때까지 기다렸다가 상위 5BIT 상태 확인 후 START되었는지 확인
- TWDR에 address저장 후 TWINT 다시 0으로 Clear, TWI Enable
- TWINT 1이 될 때까지 기다렸다가 상위 5bit 상태 확인 후 ACK, NACK되었는지 확인

3. MS5611.C 분석

1) 형식은 자유롭게~~~

<글

```
#define F_CPU 16000000UL
#include "ms5611.h"

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)
{
   i2c_start( (MS5611_ADDR << 1) | I2C_WRITE );
   i2c_write( RESET );
   i2c_stop();
   _delay_ms(10);
}
uint32_t ms5611_read_cal_reg(uint8_t reg)
{
   uint8_t PROM_dat1;
   uint8_t PROM_dat2;
   uint16_t data;

   i2c_start( (MS5611_ADDR << 1) | I2C_WRITE );
   i2c_write(MS5611_CMD_PROM(reg));
   i2c_rep_start(MS5611_ADDR <<1 | I2C_READ);

   PROM_dat1 = i2c_readAck();
   PROM_dat2 = i2c_readNak();
   i2c_stop();

   printf("PROM_dat1:%d, %d\n", PROM_dat1, PROM_dat2);
   /*
      PROM_dat1:180, 246
      PROM_dat1:188, 144
      PROM_dat1:111, 211
      PROM_dat1:101, 87
      PROM_dat1:126, 66
      PROM_dat1:108, 68
```

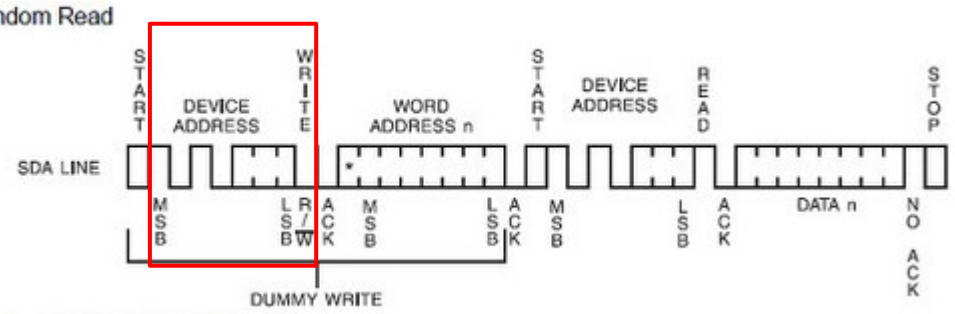
```
#define MS5611_ADDR 0x76
```

```
#define RESET       0x1E
```

01110110 << 1
= 11101100

Slave에 해당 address를 써넣겠다.
그것은 바로 reset 비트
Stop 사인 보내고, delay 10ms

Random Read




```
#define MS5611_CMD_PROM(reg)   (0xA0 + ((reg) << 1))
```

01110110 << 1
= 11101100

Slave에 해당 address를 써넣겠다.
그것은 바로 CMD_PROM(Reg)비트
Stop 사인 보내고, delay 10ms
이번엔 바로 Slave에 해당 address를 읽겠다.

3. MS5611.C 분석

1)

```
    data = ( PROM_dat1 << 8 ) + (uint16_t)PROM_dat2;

    return data;
}
void ms5611_init(void)
{
    ms5611_reset();
    UART_string_transmit("ms5611 reset ok\n");

    ms5611_cal.sens  = ms5611_read_cal_reg(1);
    ms5611_cal.off   = ms5611_read_cal_reg(2);
    ms5611_cal.tcs   = ms5611_read_cal_reg(3);
    ms5611_cal.tco   = ms5611_read_cal_reg(4);
    ms5611_cal.tref  = ms5611_read_cal_reg(5);
    ms5611_cal.tsens = ms5611_read_cal_reg(6);

    _delay_ms(1000);
}
uint32_t ms5611_conv_read_adc(uint8_t command)
{
    uint8_t rv1;
    uint8_t rv2;
    uint8_t rv3;

    uint32_t adc_data;

    i2c_start( (MS5611_ADDR << 1) | I2C_WRITE );
    i2c_write(command);
    i2c_stop();
    _delay_ms(10); //conversion Time delay

    i2c_start((MS5611_ADDR << 1) | I2C_WRITE);
    i2c_write(CMD_ADC_READ);
    i2c_rep_start(MS5611_ADDR <<1 | I2C_READ);
```

1. ack(상위8bit)와 nack(하위8bit) 값을 반환한다.

2 calibration 값

Read calibration data (factory calibrated) from PROM

Variable	Description | Equation	Recommended variable type	Size [1] [bit]	Value min	Value max	Example / Typical
C1	Pressure sensitivity | $SENS_{T1}$	unsigned int 16	16	0	65535	40127
C2	Pressure offset | OFF_{T1}	unsigned int 16	16	0	65535	36924
C3	Temperature coefficient of pressure sensitivity | TCS	unsigned int 16	16	0	65535	23317
C4	Temperature coefficient of pressure offset | TCO	unsigned int 16	16	0	65535	23282
C5	Reference temperature | T_{REF}	unsigned int 16	16	0	65535	33464
C6	Temperature coefficient of the temperature | TEMPSENS	unsigned int 16	16	0	65535	28312

3. 앞에와 상동

3. MS5611.C 분석

1)

```
    rv1 = i2c_readAck();
    rv2 = i2c_readAck();
    rv3 = i2c_readNak();
    i2c_stop();

    adc_data = ((uint32_t)rv1 << 16) + ((uint32_t)rv2 << 8) + (uint32_t)rv3;
    /*
    printf("rv1 : %d\n", rv1);
    printf("rv2 : %d\n", rv2);
    printf("rv3 : %d\n", rv3);
    */

    /*
    rv1 : 135
    rv2 : 28
    rv3 : 72
    */

    return adc_data;
}

void ms5611_measure(void)
{
    int32_t temp_raw, press_raw, dt;
    int64_t sens, off;

    temp_raw = ms5611_conv_read_adc(CONV_D2_4096);
    press_raw = ms5611_conv_read_adc(CONV_D1_4096);

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

1. ack(최상위 8bit)와 ack(상위8bit)와 nack(하위8bit) 값을 반환한다.

2. bit shift를 이용하여 cal 한 값

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

Calculate temperature

dT	Difference between actual and reference temperature [2] $dT = D2 - T_{REF} = D2 - C5 * 2^8$	signed int 32	25	-16776960	16777216	2366
TEMP	Actual temperature (-40…85°C with 0.01°C resolution) $TEMP = 20°C + dT * TEMPSENS = 2000 + dT * C6 / 2^{23}$	signed int 32	41	-4000	8500	2007 = 20.07 °C

Calculate temperature compensated pressure

OFF	Offset at actual temperature [3] $OFF = OFF_{T1} + TCO * dT = C2 * 2^{16} + (C4 * dT) / 2^7$	signed int 64	41	-8589672450	12884705280	2420281617
SENS	Sensitivity at actual temperature [4] $SENS = SENS_{T1} + TCS * dT = C1 * 2^{15} + (C3 * dT) / 2^8$	signed int 64	41	-4294836225	6442352640	1315097036
P	Temperature compensated pressure (10…1200mbar with 0.01mbar resolution) $P = D1 * SENS - OFF = (D1 * SENS / 2^{21} - OFF) / 2^{15}$	signed int 32	58	1000	120000	100009 = 1000.09 mbar

3. MS5611.C 분석

1)

```
double ms5611_getAltitude(void)
{
   double alt;
   alt = (1 - pow(_ms5611_pres/(double)101325, 0.1903)) / 0.0000225577;
   return alt;
}

uint32_t ms5611_getPress(void)
{
   return _ms5611_pres;
}

int32_t ms5611_getTemp(void)
{
   return _ms5611_temp;
}
```

1. 고도를 환산한다. Math 라이브러리 사용

감사합니다.