AVR – HW8

임베디드스쿨1기

Lv1과정

2020. 11. 05

강경수

1. I2C통신

2020.11.03 강경수

1. I2C 통신이란?

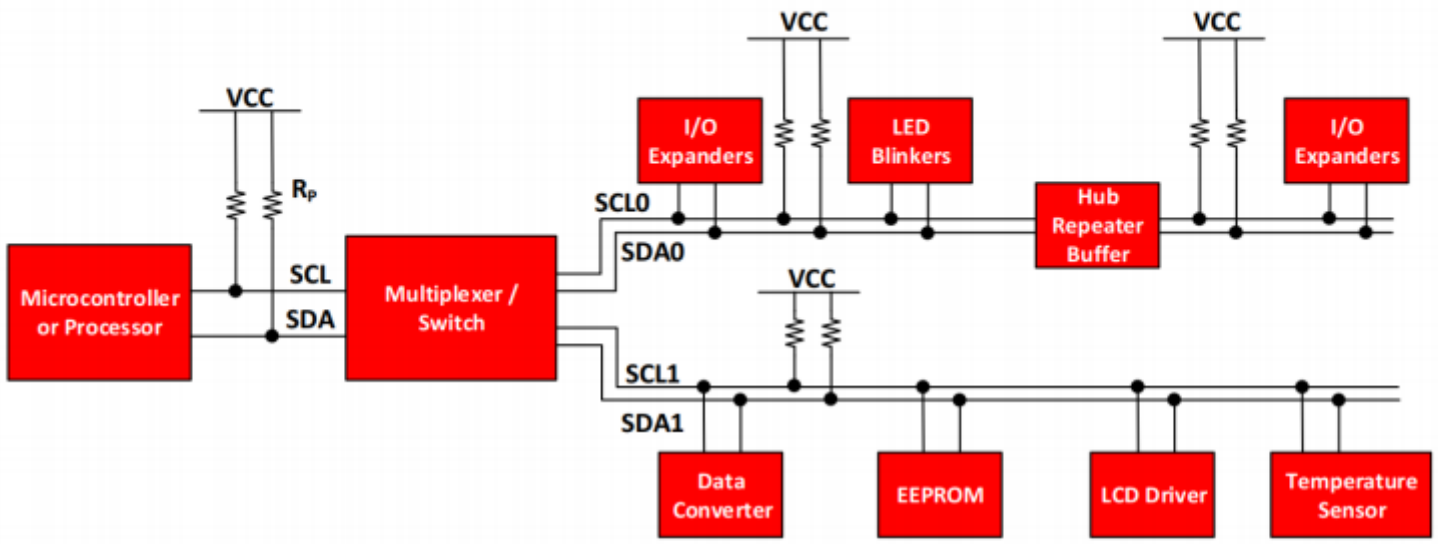
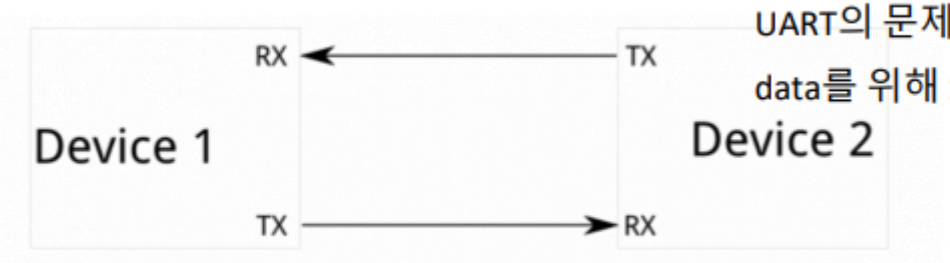


Figure 1. Example I²C Bus

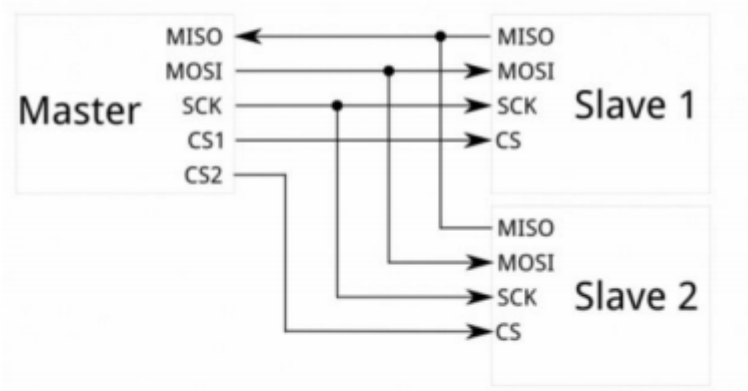
- SDA 양방향 직렬 통신
- SDA,SCL 모두 오픈컬렉터 방식의 출력단 구조
- CLOCK에 따라서 DATA를 송신하는 Synchoronous (SPI 처럼 MASTER SLAVE 1:1 Synchoronous 는 아님)
- SDA는 양방향 통식 가능 하다. 하지만 SCL은 항상 MASTER에서 생성한다.
- 전송 데이터는 항상 MSB부터 전송시작한다.
- 100kbps, 400kbps, 3.4Mbps 세가지 통신 속도
- I2C V2.1 부터는 MultiMater기능 사용 가능

1. I2C 통신

2. I2C를 사용하는 이유



UART의 문제점 : 비동기방식, DATA손실(8bit 의 data를 위해 10bit를 송신해야함)



SPI 의 문제점 : 많은 신호선을 필요로 함.

→ 즉 I2C는 UART의 문제점과 SPI의 문제점을 적절히 해결하는 통신 PROTOCOL
(UART보다 빠른 속도, 2가닥의 신호선으로 통신가능, Synchoronous)

1. I2C 통신

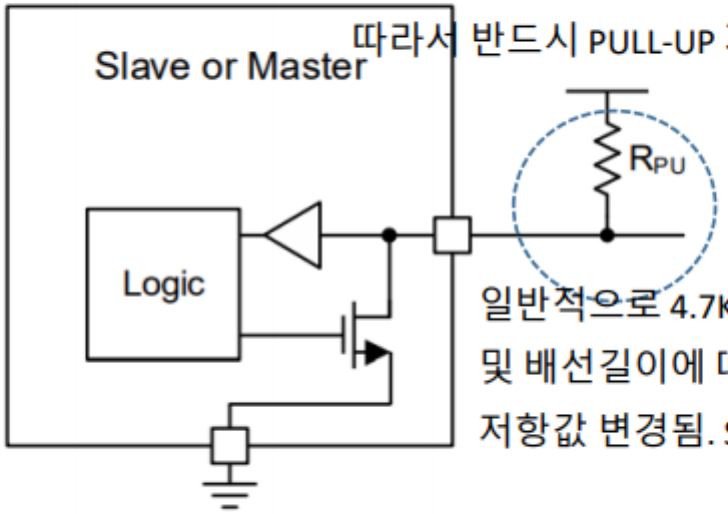
3. I2C 하드웨어 구조

OPEN DRAIN

OPEN COLLECTOR 구조

따라서 반드시 PULL-UP 저항 달아줘야 함.

OPEN DRAIN 구조이기
때문에 외부 전압에 의해서
HIGH LEVEL 결정 된다.

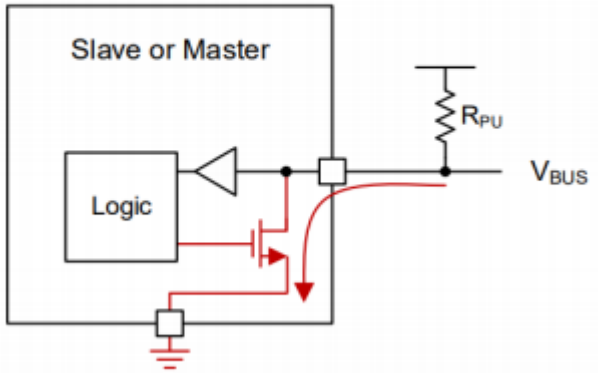


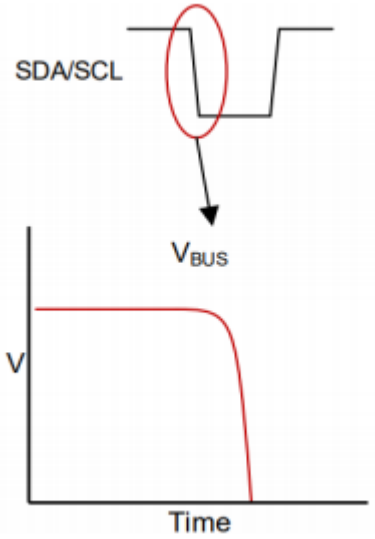
일반적으로 4.7K를 달아주나 I2C 통신속도
및 배선길이에 따라서
저항값 변경됨. SLVA689 TI 기술문서 참조

Figure 2. Basic Internal Structure of SDA/SCL Line

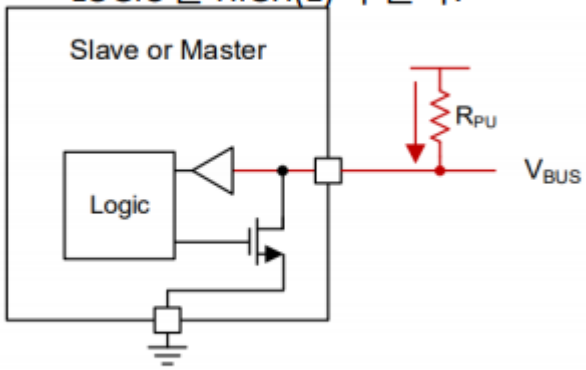
1. I2C 통신

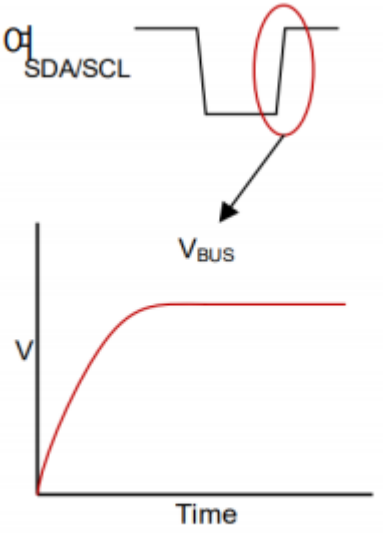
옆 그림에서 보여지듯이 FET ON 될시
LOGIC은 0이 되게 된다.





FET OFF 될시 PULL UP RESISTOR에 의하여
LOGIC 은 HIGH(1)이 된다.





1. I2C 통신

4. 일반적인 I2C 동작

1) START, STOP 조건

START 조건 : SCL HIGH 에서 SDA LOW로 변화

STOP 조건 : SCL LOW에서 SDA HIGH로 변화

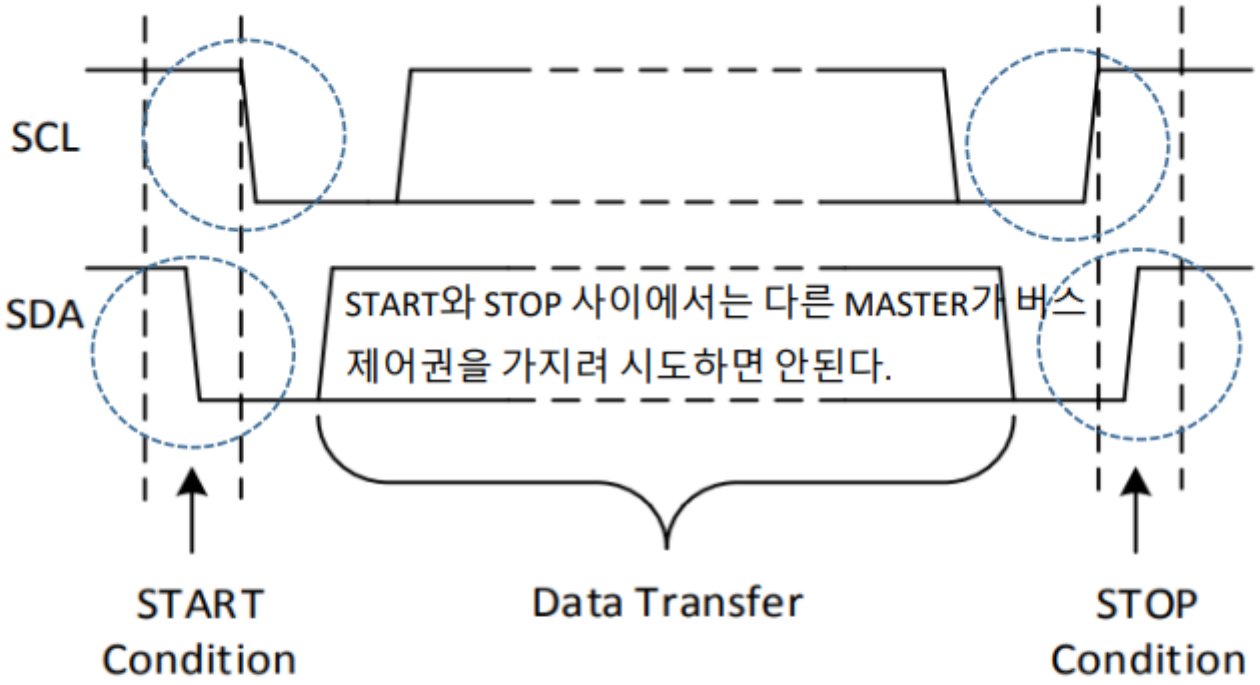
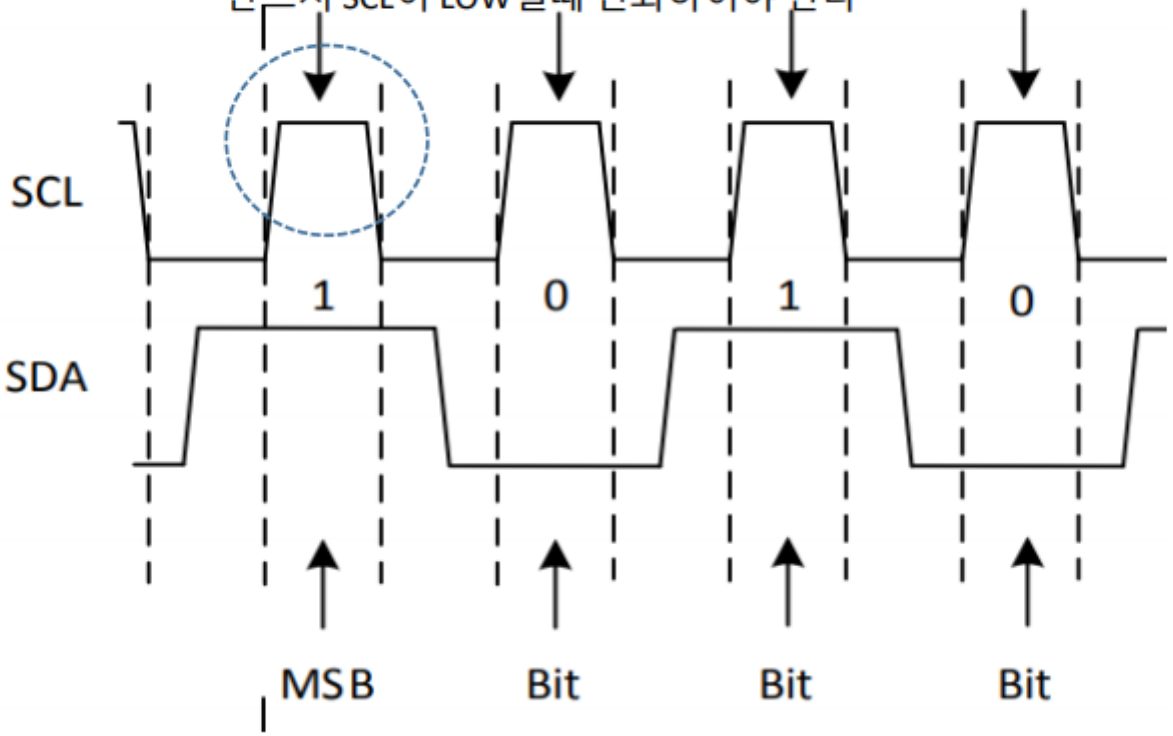


Figure 5. Example of START and STOP Condition

START조건 이후 STOP조건 전에 또 다시 START조건
시작 될 수 있는데 이는 REPEATED START라한다.
REPEATED START는 버스 제어권을 포기하지 않고
새로운 전송을 시작하는 것일 뿐이다. 모든것은
START와 동일함!

1. I2C 통신

2) 유효 데이터 조건 SCL이 HIGH 상태일때만 SDA는 유효하며 SDA의 변경은 반드시 SCL이 LOW일때 변화하여야 한다



1. I2C 통신

3) 일반적인 데이터 쓰기(WRITE)

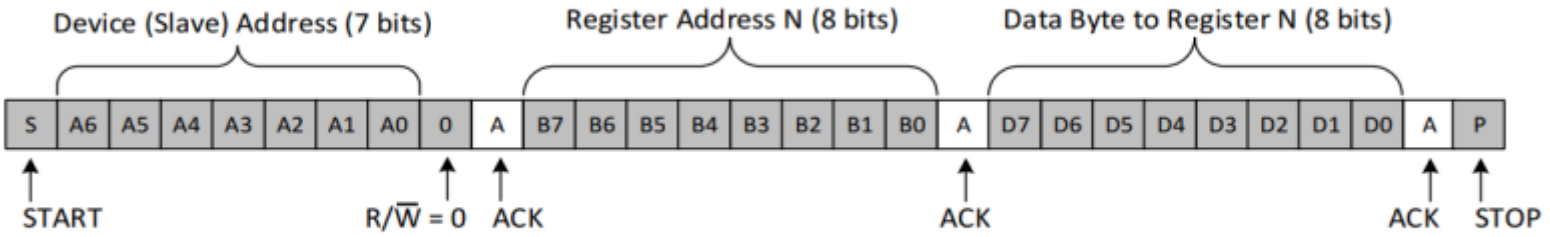


Figure 8. Example I²C Write to Slave Device's Register

중간중간 ACK 즉 SLAVE가 정상적으로 DATA를 수신했다는 신호를 준다. 이때 정상 수신은 0을 반환하며 1을 반환할시 NACK 즉 정상적으로 DATA가 전송되지 못했음을 의미한다.

4) 일반적인 데이터 읽기(READ)

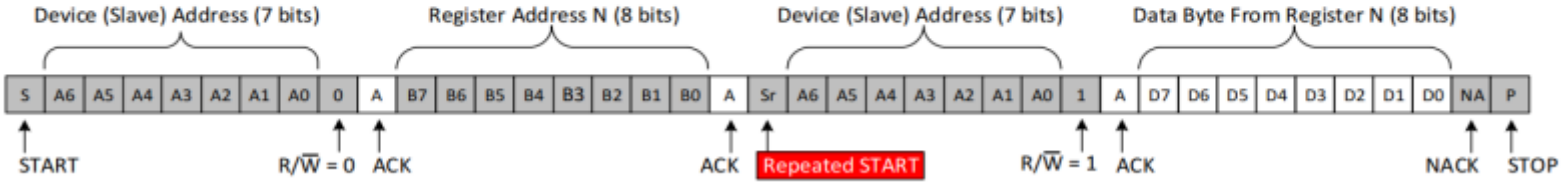
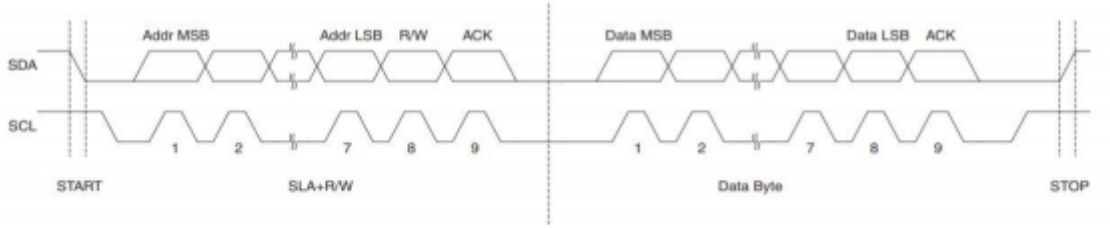


Figure 9. Example I²C Read from Slave Device's Register

1. I2C 통신

Q. 질문 : DATASHEET를 보면 Slave Address 이후 data를 바로 전송하거나
data를 수신하는 것을 볼 수 있는데(r/w, ack제외)
제가 참고한 위 기술자료에는 Register Address가 존재합니다.
이는 Register Address 영역이 별도로 존재하는 Slave(Device)가 따로 있다고
생각하면 될까요?



위는 ATEMAG328P DATASHEET에서 발췌한 TIMING 표

2. SPI REVIEW

▣ SPI REVIEW

2020.11.03 강경수

```
void SPI_Slave_Init(void)
{
    cbi(DDRB,5);
    cbi(DDRB,3);
    cbi(DDRB,2);
    sbi(DDRB,4);
    PORTB = 0xff;
    sbi(SPCR,SPE);
}
```

SPCR – SPI Control Register

Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
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	

```
 void SPI_Master_Init(void)
 {
     sbi(DDRB,5);
     sbi(DDRB,3);
     sbi(DDRB,2);
     cbi(DDRB,4);
     PORTB = 0xff;
     sbi(SPCR,SPE);
     sbi(SPCR,MSTR);
 }
```

SPCR – SPI Control Register

Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
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	

SPE : SPI ENABLE
MSTR : 0 → SLAVE
MSTR : 1 → MASTER

2. SPI REVIEW

```
unsigned char SPI_TxRx(unsigned char Data)
{
    SPDR = Data;
    while(!(SPSR&(1<<SPIF)));
    _delay_ms(100);
    return SPDR;
}
```

SPDR : DATA 레지스터

SPDR에 값을 쓰면 1bit 씩 전송된다.

SPDR – SPI Data Register

Bit	7	6	5	4	3	2	1	0	
0x2E (0x4E)	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

DATA 전송이 완려되면

SPSR 의 SPIF레지스터가

1로 SET된다.

SPSR – SPI Status Register

Bit	7	6	5	4	3	2	1	0	
0x2D (0x4D)	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

```
void UART_INIT(void)
{
    sbi(UCSR0A,U2X0);

    UBRR0H = 0x00;
    UBRR0L = 207;

    UCSR0C |= 0x06;

    sbi(UCSR0B, RXEN0);
    sbi(UCSR0B, TXEN0);
}
```

8BIT, RX TX ENABLE

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%

BAUD RATE : 9600

UCSRnC – USART Control and Status Register n C

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
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	1	1	0	

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit

2. SPI REVIEW

```
unsigned char UART_receive(void)
{
    while(!(UCSR0A&(1<<RXC0)));
    return UDR0;
}
```

UCSRnA – USART Control and Status Register n A

Bit	7	6	5	4	3	2	1	0
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn
Read/Write	R	R/W	R	R	R	R	R/W	R/W
Initial Value	0	0	1	0	0	0	0	0

UART 완료 FLAG

```
unsigned char UART_transmit(unsigned char data)
{
    while(!(UCSR0A & (1<<UDRE0)));
    UDR0 = data;
}
```

UART DATA EMPTY FLAG

```
int main(void)
{
    /* Replace with your application code */
    unsigned char data;
    UART_INIT();
    SPI_Master_Init();
//  SPI_Slave_Init();

    cbi(PORTB,2);//master
    while (1) //master
    {
        SPI_TxRx('A');
        data = SPI_TxRx(0x00);

        UART_transmit(data);
        _delay_ms(1000);
    }
```

```
while(1) //slave
```

전체적인 설명:
MASTER 에서 'A'를 전송하면
SLAVE는 'a'로 변환하고
Master에서 0x00을 전송하면
'a'가 1bit씩 밀려와서 MASTER는
'a'를 갖게 된다.

MASTER CODE 수정

대문자 A 전송후 100ms 대기후
0x00으로 변환된 'a' data값에 저장

기존 코드보다 조금 더 'a'가
출력됐으나 결국 대문자 A로 바뀌었음

2. SPI REVIEW

```
cbi(PORTB,2);//master
while (1) //master
{
    SPI_TxRx('A');
    data = SPI_TxRx(0x00);

    UART_transmit(data);
    _delay_ms(1000);
}
```

```
while(1) //slave
{
    while(!(SPSR & (1<<SPIF)));
    SPDR = SPDR+ 0x20;
}
```

MASTER CODE 수정

대문자 A 전송후 100ms 대기후
0x00으로 변환된 'a' data값에 저장

기존 코드보다 조금 더 'a'가
출력됐으나 결국 대문자 A로 바뀌었음

Q. PORTB의 INPUT부분에 PULL-UP을 해줘야하는 이유가 있나요?
신호 안정화를 인한 조취라고 보면 되는지요?