EDDI
Electronic Design
Development Institute

에디로봇아카데미

임베디드 마스터 Lv1 과정

Ch10.

제 3기

2022. 3. 12

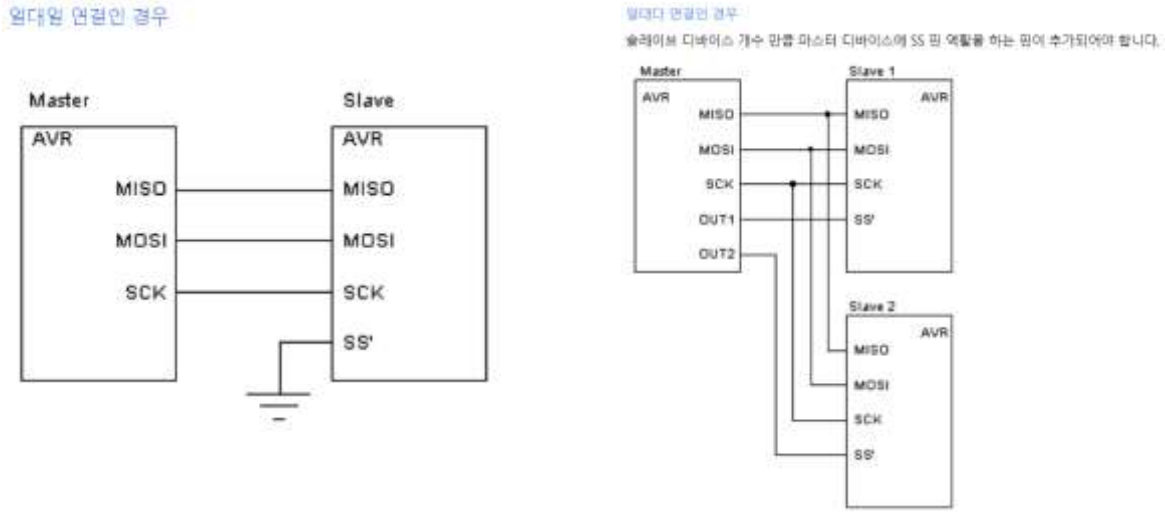
여건영

CONTENTS

SPI 란?

SPI(Serial Peripheral Interface)를 사용하면
MCU와 주변 장치 간 또는 여러 MCU 장치간에 고속 동기 데이터 전송이 가능하다.

1. 두 개의 장치가 SPI 통신을 하는 경우 한 대는 Master, 또 다른 한 대는 Slave 이다.
2. 하나의 마스터 장치와 여러 슬레이브 장치, 1:N 통신이 가능하지만
   이때 주의점은 동시에 두 대 이상의 마스터 장치가 SPI 통신을 시작하지 않도록 해야한다.
3. 두 대 이상의 마스터 장치가 슬레이브 장치와 연결될 수 도 있지만
   이때 주의점은 동시에 두 대 이상의 마스터 장치가 SPI 통신을 시작하지 않도록 해야한다.



마스터와 슬레이브 간의 데이터 전송

마스터와 슬레이브 유닛의 기본 구조는 동일하며, 마스터 유닛쪽에는 추가로 SPI 클럭 발생기가 있다.
두 유닛의 MISO, MOSI, CK 핀은 서로 연결되어 있다.

한 클럭 사이클에서 비트가 마스터 → 슬레이브, 슬레이브 → 마스터로 동시에 시프트되기 때문에
두 8비트 시프트 레지스터는 하나의 16비트 순환 시프트 레지스터로 간주 될 수 있다.

이것은 8개의 SCK 클럭 펄스 후에 마스터와 슬레이브 사이의 1바이트 데이터가 이동(교환)된다는 의미이다.

* 전송 될 새 바이트는 전체 시프트 사이클이 완료되기 전에 SPDR / 시프트 레지스터에 쓸 수가 없다.
  바이트에 대한 전송이 완료되어야 새로운 바이트를 전송할 수 있다.

* 수신된 바이트는 전송이 완료된 직후 수신 버퍼에 기록이된다.

* 다음 바이트 전송이 완료되기 전에 수신 버퍼를 읽어야함, 그렇지 않으면 데이터가 손실된다.(덮어씌워짐)

* 데이터 레지스터(SPDR)을 읽으면 수신 버퍼의 데이터가 반환된다.

마스터와 슬레이브 간의 데이터 전송

아래 그림은 SPI를 사용하여 Master와 Slave CPU 간의 상호 연결을 보여주고 있다.
SPI 시스템은 마스터와 슬레이브에 각각 있는 8bit 시프트 레지스터와 마스터에만 있는 클럭 발생기로 구성된다.

Master device에서 통신을 원하는 Slave device의 SS pin을 LOW로 낮출 때, 두 디바이스간 SPI 통신이 가능한 상태가 된다.

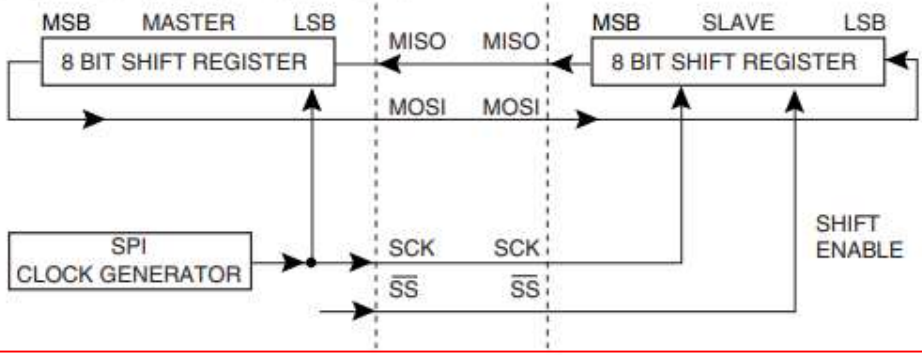
Master와 Slave는 각자의 시프트 레지스터에 전송할 데이터를 준비하고 Master는 데이터를 교환하기 위해 SCK 라인에 필요한 클럭 펄스를 발생시킨다.

데이터는 항 MOSI (Master Out – Slave In), MISO (Master In – Slave Out) 라인을 통해 이동한다.

전송과 수신이 동시에 일어나서 결과적으로 Master와 Slave는 한 byte씩 데이터를 교환하게 된다.
데이터 패킷 교환을 모두 끝낸 후
(한 byte 교환을 완료할때 마다),
Master는 SS 라인을 HIGH로 만들어
슬레이브와 동기화해야 한다.

성공적으로 SPI 통신을 하기 위해서
Master와 Slave의 통신속도는
반드시 같아야 한다.

Figure 66. SPI Master-slave Interconnection



마스터와 슬레이브 간의 데이터 전송

[마스터 디바이스]
디바이스가 Master로 설정된 경우 SPI 인터페이스는 SS 라인을 자동으로 제어하지 않는다.
SPI 통신을 시작하기 전에 소프트웨어 적으로 SS 라인을 제어해야한다.

SS 라인 제어 후, **SPI 데이터 레지스터(SPDR)**에 1 바이트를 쓰면 SPI 클럭 발생기가 작동되고 하드웨어는
데이터 8비트를 슬레이브 디바이스로 이동시킨다. 1 바이트 이동 후 SPI 클럭 발생기가 중지되며,
전송이 종료되었다는 것을 알리기 위해 **전송 종료 플래그(SPIF)**를 1로 설정함

SPCR 레지스터의 SPI 인터럽트 활성화 비트 (SPIE)가 1로 설정되면 인터럽트가 요청됨

Master는 SPI 데이터 레지스터(SPDR)에 다음 바이트를 기록하여 계속 한 바이트를 슬레이브로 이동시키거나
Slave Select 라인을 High 로 만들어서 슬레이브 디바이스에 패킷의 끝을 알릴 수 있다.

마지막으로 들어오는 바이트는 나중에 사용할 수 있도록 버퍼 레지스터에 보관됨

마스터와 슬레이브 간의 데이터 전송

[슬레이브 디바이스]
디바이스가 Slave로 구성되면 SS 핀이 High를 유지하는 동안 슬레이브의 SPI 인터페이스는 대기하게 됨

이 상태에서 소프트웨어는 **SPI 데이터 레지스터(SPDR)**의 내용을 업데이트 할 수 있지만 SS 핀이
Low 로 변경될 때까지 SCK 핀에서 들어노느 클록 펄스에 의해 데이터가 이동되지 않습니다.

SS핀이 Low가 되면 SCK 핀으로 입력되는 클럭펄스를 이용하여 1byte를 마스터로 이동시킴
한 바이트가 이동하고 나면, 전송이 종료되었다는 것을 알리기 위해 SPIF(전송 종료 플래그)를 1로 설정

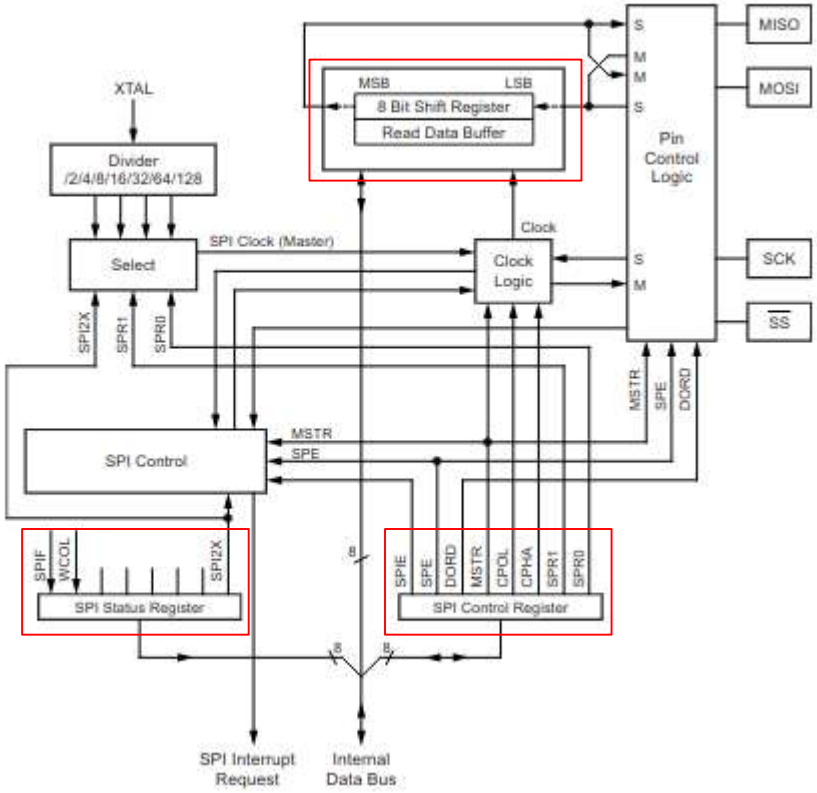
SPCR 레지스터의 SPI 인터럽트 활성화 비트(SPIE)가 1로 설정되면 인터럽트가 요청됨

Slave는 수신한 데이터를 읽기 전에 SPI 데이터 레지스터(SPDR)로 보낼 새 데이터를 계속 배치 할 수 있음
Slave는 계속해서 새로운 데이터를 SPI 데이터 레지스터(SPDR)에 넣어두고 새로운 데이터 입력을 대기함

마지막으로 들어오는 바이트는 나중에 사용할 수 있도록 버퍼 레지스터에 보관됨

SPI Block Diagram

Figure 18-1. SPI Block Diagram(1)



25LC010A – 1K(1024) bit, (128) byte EEPROM

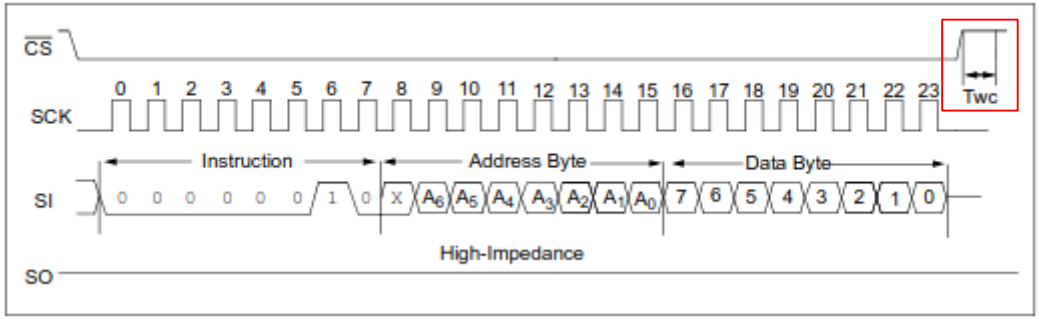
EEPROM – On Board 상태에서 사용자가 내용을 byte 단위로 Read 하거나 Write 하여
사용할 수 있는 비 휘발성 메모리이다.

Read 동작은 access 동작이 다소 느릴지라도 SRAM과 유사한데 반해,
Write 동작을 수행하는 경우 1byte를 write 할 때마다 수 ms 이상의 시간 지연이 필요하다.
따라서 실시간으로 사용되는 변수를 저장하는 메모리나 스택 메모리로는 사용 될 수 없다.

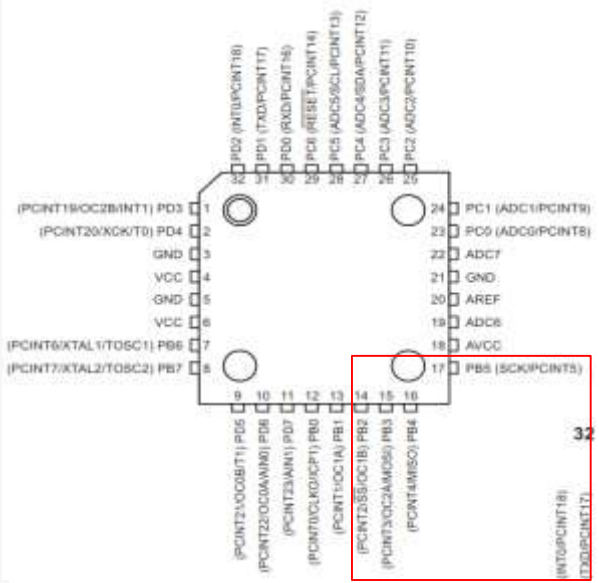
Byte write 시퀀스에서 Twc를 확인 할 수 있다.
Twc : byte write 시퀀스 후 CS의 상승 edge에서 시작하고 내부 쓰기 주기가 완료되면 끝이 난다.
Byte 단위 대신 page 단위로 기록하면, 훨씬 짧은 시간에 많은 바이트의 값을 기록할 수 있다.

20	Twc	Internal write cycle time (byte or page)	—	5	ms	(Note 3)

FIGURE 2-2: BYTE WRITE SEQUENCE



spi_init




```
#define SPI_SS      PB2
#define SPI_MOSI    PB3
#define SPI_MISO    PB4
#define SPI_SCK     PB5
```

```
void spi_init (void)
{
    // CS(Chip Select) 출력 설정
    DDRB |= (1 << SPI_SS);
    // CS를 HIGH 로 띄워서 아무 선택이 발생하지 않게 함
    PORTB |= (1 << SPI_SS);
    // MOSI
    DDRB |= (1 << SPI_MOSI);
    // MISO
    DDRB &= ~(1 << SPI_MISO);
    // SCK
    DDRB |= (1 << SPI_SCK);
    // Master Mode
    // MSTR: Master/Slave Select 비트가 1 설정되면 Master
    SPCR |= (1 << MSTR);
    // SPI 활성화
    // SPE: SPI Enable 비트가 설정됨으로써 SPI 모듈이 활성화됨
    SPCR |= (1 << SPE);
}
```

각 기능에 맞는 port를 연결하고,
입, 출력을 설정하여 초기화 시켜준다.

* CS pin은 active low 이므로
초기에는 꼭 High 상태로 두자

```
#define EEPROM_SELECT()     PORTB &= ~(1 << SPI_SS)
#define EEPROM_DESELECT()   PORTB |= (1 << SPI_SS)

#define EEPROM_READ     0b00000011
#define EEPROM_WRITE    0b00000010
#define EEPROM_WREN     0b00000110
#define EEPROM_RDSR     0b00000101

#define EEPROM_WRITE_IN_PROGRESS        0

#define EEPROM_PAGE_SIZE            16
#define EEPROM_TOTAL_BYTE           128
```

아래 data

TABLE 2-1: INSTRUCTION SET

Instruction Name	Instruction Format	Description
READ	0000 x011	Read data from memory array beginning at selected address
WRITE	0000 x010	Write data to memory array beginning at selected address
WRDI	0000 x100	Reset the write enable latch (disable write operations)
WREN	0000 x110	Set the write enable latch (enable write operations)
RDSR	0000 x101	Read STATUS register
WRSR	0000 x001	Write STATUS register

x = don't care

spi_init

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_init (void)
{
    // CS(Chip Select) 출력 설정
    DDRB |= (1 << SPI_SS);
    // CS를 HIGH 로 띄워서 아무 선택이 발생하지 않게 함
    PORTB |= (1 << SPI_SS);
    // MOSI
    DDRB |= (1 << SPI_MOSI);
    // MISO
    DDRB &= ~(1 << SPI_MISO);
    // SCK
    DDRB |= (1 << SPI_SCK);
    // Master Mode
    // MSTR: Master/Slave Select 비트가 1 설정되면 Master
    SPCR |= (1 << MSTR);
    // SPI 활성화
    // SPE: SPI Enable 비트가 설정됨으로써 SPI 모듈이 활성화됨
    SPCR |= (1 << SPE);
}
```

* Bit 7 - SPIE : SPI Interrupt Enable
  SPSR 레지스터의 SPIF 비트가 설정되고 SREG의 글로벌 인터럽트 활성화 비트가 설정된 경우,
  이 비트가 설정되면 SPI 인터럽트가 실행되도록 함

* Bit 6 - SPE : SPI Enable
  SPE 비트가 1에 기록되면 SPI가 활성화됨

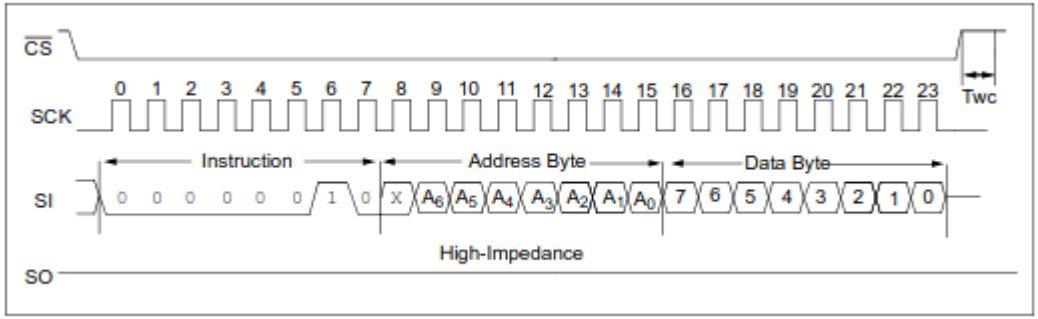
* Bit 4 - MSTR : Master/Slave Select - 마스터 또는 슬레이브 모드 선택
  이 비트가 1로 설정되면 마스터 SPI 모드가 선택되고 이 비트에 0을 설정하면 슬레이브 SPI 모드가 선택됨

  SS 핀이 입력으로 설정되고 MSTR이 1로 설정된 상태에서 SS 핀이 Low 상태로 구동되면
  MSTR이 0으로 클리어되고 SPI 상태 레지스터(SPSR)의 SPIF가 1로 설정됨
  사용자는 SPI 마스터 모드를 다시 활성화하도록 MSTR을 1로 설정해야 함

* Bit 1,0 - SPR1, SPR0 : SPI Clcok Rate Select 1 and 0
  두 비트는 마스터로 구성된 장치의 SCK 속도를 제어함

쓰기

FIGURE 2-2: BYTE WRITE SEQUENCE



X = don't care

```
void eeprom_write_byte(uint8_t address, uint8_t data)
{
    // 쓰기 락 해제(쓰기 허용)
    eeprom_write_enable();

    EEPROM_SELECT();

    eeprom_change_byte(EEPROM_WRITE);

    eeprom_send_address(address);

    eeprom_change_byte(data);

    EEPROM_DESELECT();
    // 쓰기가 완료될 때까지 대기
    // _BV 는 해당 비트를 1로 만듦
    while (eeprom_read_status() & _BV(EEPROM_WRITE_IN_PROGRESS))
    {
        ;
    }
}
```

쓰기의 경우 WREN 명령을 통해 쓰기 활성화를 한다.
명령어의 8비트가 모두 전송된 후, CS는 쓰기 활성화를 위해 HIGH 구동되어야 한다.
CS가 HIGH로 구동되지 않고 WREN 명령 이후 쓰기 작업을 한다면 데이터가 올바로 처리되지 않는다.
쓰기 활성화 이후 WRITE 명령을 실행한 다음 나머지 주소를 전송하고 다음 기록할 데이터를 작성한다.

- CS pin을 Low로 낮춘다.
- 쓰기를 하기 위해 WREN(6) 명령을 보낸다.
- CS pin을 High로 만든다 . // 쓰기가 활성화 됨
- 다시 CS pin을 Low로 낮춘다.
- WRITE(2) 명령을 보낸다.
- 주소값을 보낸 후, 주소값에 기록할 데이터를 보낸다.
- CS pin을 High로 만든다.

쓰기

SPI Status Register - SPSR

Bit	7	6	5	4	3	2	1	0	
	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 eeprom_change_byte (uint8_t byte)
{
    // 전송 관점에서는 내가 전송할 데이터를 기록한다.
    // 수신 관점에서는 수신된 정보가 SPDR에 존재함을 의미하기도 한다.
    SPDR = byte;
    // 위의 데이터 전송 완료까지 대기한다.
    // SPI Status Register 의 SPIF 비트 설정을 확인
    // SPIF: SPI Interrupt Flag
    // 전송이 완료되면 알아서 설정되므로
    // 이것이 설정되는 것을 기다리는 것을 의미함
    loop_until_bit_is_set(SPSR, SPIF);
}
```

* Bit 7 - SPIF : SPI Interrupt Flag

직렬 전송이 완료되면 SPIF 플래그가 설정된다. SPCR의 SPIE가
설정되고 글로벌 인터럽트가 Set 되면 인터럽트가 발생한다.

SS가 입력이고 SPI가 마스터 모드에 있을 때,
LOW로 구동되면 SPIF 플래그도 설정된다.
SPIF는 해당 인터럽트 처리 벡터를 실행할 때 하드웨어에 의해 Clear 된다

또는 SPIF가 설정된 SPI 상태 레지스터를 먼저 읽은
다음 SPI 데이터 레지스터 (SPDR)에 엑세스하여 SPIF 비트를 지운다.

loop_until_bit_is_set

```
#define loop_until_bit_is_set(  sfr,
                                bit
                             )    do { } while (bit_is_clear(sfr, bit))
```

```
#include <avr/io.h>
```

Wait until bit bit in IO register sfr is set.

쓰기

```
uint8_t eeprom_read_status (void)
{
    // EEPROM 선택
    EEPROM_SELECT();
    // 상태 읽기
    eeprom_change_byte(EEPROM_RDSR);
    // 읽기
    eeprom_change_byte(0);
    // EEPROM 선택 해제
    EEPROM_DESELECT();

    return SPDR;
}
```

2.5 Read Status Register Instruction (RDSR)

The Read Status Register instruction (RDSR) provides access to the STATUS register. See Figure 2-6 for the RDSR timing sequence. The STATUS register may be read at any time, even during a write cycle. The STATUS register is formatted as follows:

TABLE 2-2: STATUS REGISTER

7	6	5	4	3	2	1	0
–	–	–	–	W/R	W/R	R	R
X	X	X	X	BP1	BP0	WEL	WIP
W/R = writable/readable. R = read-only.							

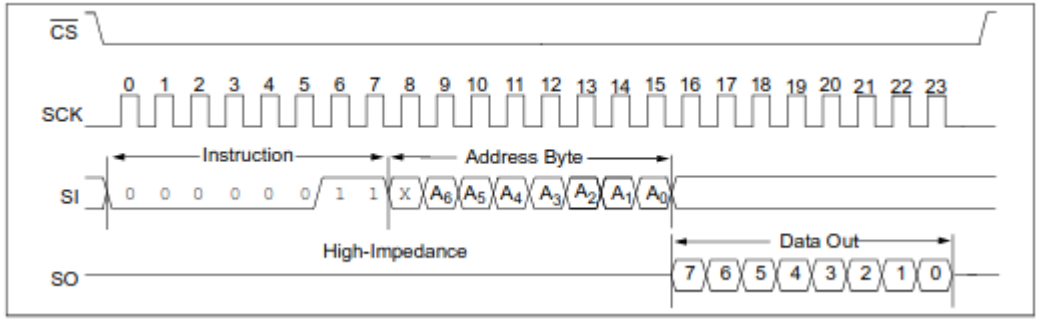
The **Write-In-Process (WIP)** bit indicates whether the 25XX010A is busy with a write operation. When set to a '1', a write is in progress, when set to a '0', no write is in progress. This bit is read-only.

```
void eeprom_write_byte(uint8_t address, uint8_t data)
{
    // 쓰기 락 해제(쓰기 허용)
    eeprom_write_enable();

    EEPROM_SELECT();

    eeprom_change_byte(EEPROM_WRITE);

    eeprom_send_address(address);

    eeprom_change_byte(data);

    EEPROM_DESELECT();
    // 쓰기가 완료될 때까지 대기
    // _BV 는 해당 비트를 1로 만듦
    while (eeprom_read_status() & _BV(EEPROM_WRITE_IN_PROGRESS))
    {
        ;
    }
}
```

데이터까지 전송이 완료되고, 쓰기가 완료되었는지
RDSR(5) 명령을 통해 지금 Write_In_Process 인지 확인하고
0이면 반복문을 빠져나간다.

읽기

FIGURE 2-1: READ SEQUENCE




```
uint8_t eeprom_read_byte (uint8_t address)
{
    EEPROM_SELECT();

    eeprom_change_byte(EEPROM_READ);

    eeprom_send_address(address);

    eeprom_change_byte(0);

    EEPROM_DESELECT();

    return SPDR;
}
```

읽기의 경우 CS가 LOW로 떨어져 있어야하며 8비트 주소가 전송되어야 한다.
READ 명령과 주소가 전송된 이후 선택한 주소의 메모리에 저장된 데이터가 SO pin에서 시프트 된다.
다음 주소의 메모리에 저장된 데이터는 Slave에 클럭 펄스를 계속 제공하여 순차적으로 읽을 수가 있다.
내부 주소 포인터는 데이터의 각 byte가 시프트 된 이후 자동으로 상위 주소로 증가하므로
별도의 변경이 필요없다.

- CS pin을 Low로 낮춘다.
- 읽기명령 READ(3) 명령을 보낸다.
- 읽을 주소를 보낸다.
- 0을 보낸다. // 상호간에 동기화가 되어 있어, Master에서 아무것도 아닌 텅빈 데이터를 전송할 필요가 있
- CS pin을 High로 만든다.

QnA

SPI Status Register - SPSR

Bit	7	6	5	4	3	2	1	0	
	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 eeprom_change_byte (uint8_t byte)
{
    // 전송 관점에서는 내가 전송할 데이터를 기록한다.
    // 수신 관점에서는 수신된 정보가 SPDR에 존재함을 의미하기도 한다.
    SPDR = byte;
    // 위의 데이터 전송 완료까지 대기한다.
    // SPI Status Register 의 SPIF 비트 설정을 확인
    // SPIF: SPI Interrupt Flag
    // 전송이 완료되면 알아서 설정되므로
    // 이것이 설정되는 것을 기다리는 것을 의미함
    loop_until_bit_is_set(SPSR, SPIF);
}
```

* Bit 7 - SPIF : SPI Interrupt Flag

직렬 전송이 완료되면 SPIF 플래그가 설정된다. SPCR의 SPIE가 설정되고 글로벌 인터럽트가 Set 되면 인터럽트가 발생한다.

SS가 입력이고 SPI가 마스터 모드에 있을 때,
LOW로 구동되면 SPIF 플래그도 설정된다.
SPIF는 해당 인터럽트 처리 벡터를 실행할 때 하드웨어에 의해 Clear 된다

또는 SPIF가 설정된 SPI 상태 레지스터를 먼저 읽은
다음 SPI 데이터 레지스터 (SPDR)에 엑세스하여 SPIF 비트를 지운다.

loop_until_bit_is_set

```
#define loop_until_bit_is_set(sfr, bit) do { } while (bit_is_clear(sfr, bit))
```

```
#include <avr/io.h>
```

Wait until bit bit in IO register sfr is set.

(QnA) SPIF bit는 다음 전송을 위해 SPDR에 보낼 새 data를 쓰거나 읽는?(엑세스) 할 때 자동으로 Clear 된 다는 말인지 질문 드립니다.

QnA

```
uint8_t eeprom_read_status (void)
{
    // EEPROM 선택
    EEPROM_SELECT();
    // 상태 읽기
    eeprom_change_byte(EEPROM_RDSR);
    // 읽기
    eeprom_change_byte(0);
    // EEPROM 선택 해제
    EEPROM_DESELECT();

    return SPDR;
}
```

2.5 Read Status Register Instruction (RDSR)

The Read Status Register instruction (RDSR) provides access to the STATUS register. See Figure 2-6 for the RDSR timing sequence. The STATUS register may be read at any time, even during a write cycle. The STATUS register is formatted as follows:

TABLE 2-2: STATUS REGISTER

7	6	5	4	3	2	1	0
–	–	–	–	W/R	W/R	R	R
X	X	X	X	BP1	BP0	WEL	WIP
W/R = writable/readable. R = read-only.							

The **Write-In-Process (WIP)** bit indicates whether the 25XX010A is busy with a write operation. When set to a '1', a write is in progress, when set to a '0', no write is in progress. This bit is read-only.

```
void eeprom_write_byte(uint8_t address, uint8_t data)
{
    // 쓰기 락 해제(쓰기 허용)
    eeprom_write_enable();

    EEPROM_SELECT();

    eeprom_change_byte(EEPROM_WRITE);

    eeprom_send_address(address);

    eeprom_change_byte(data);

    EEPROM_DESELECT();
    // 쓰기가 완료될 때까지 대기
    // _BV 는 해당 비트를 1로 만듦
    while (eeprom_read_status() & _BV(EEPROM_WRITE_IN_PROGRESS))
    {
        ;
    }
}
```

데이터까지 전송이 완료되고, 쓰기가 완료되었는지
RDSR(5) 명령을 통해 지금 Write_In_Process 인지 확인하고
0이면 반복문을 빠져나간다.

(QnA) EEPROM_WRITE_IN_PROGRESS를 _BV하여 RDSR과 비교를하여 쓰기가 완료되었는지 확인했는데, 후에 코드 적으로 다시 Clear를 해줘야 되는 것인지 질문 드립니다.