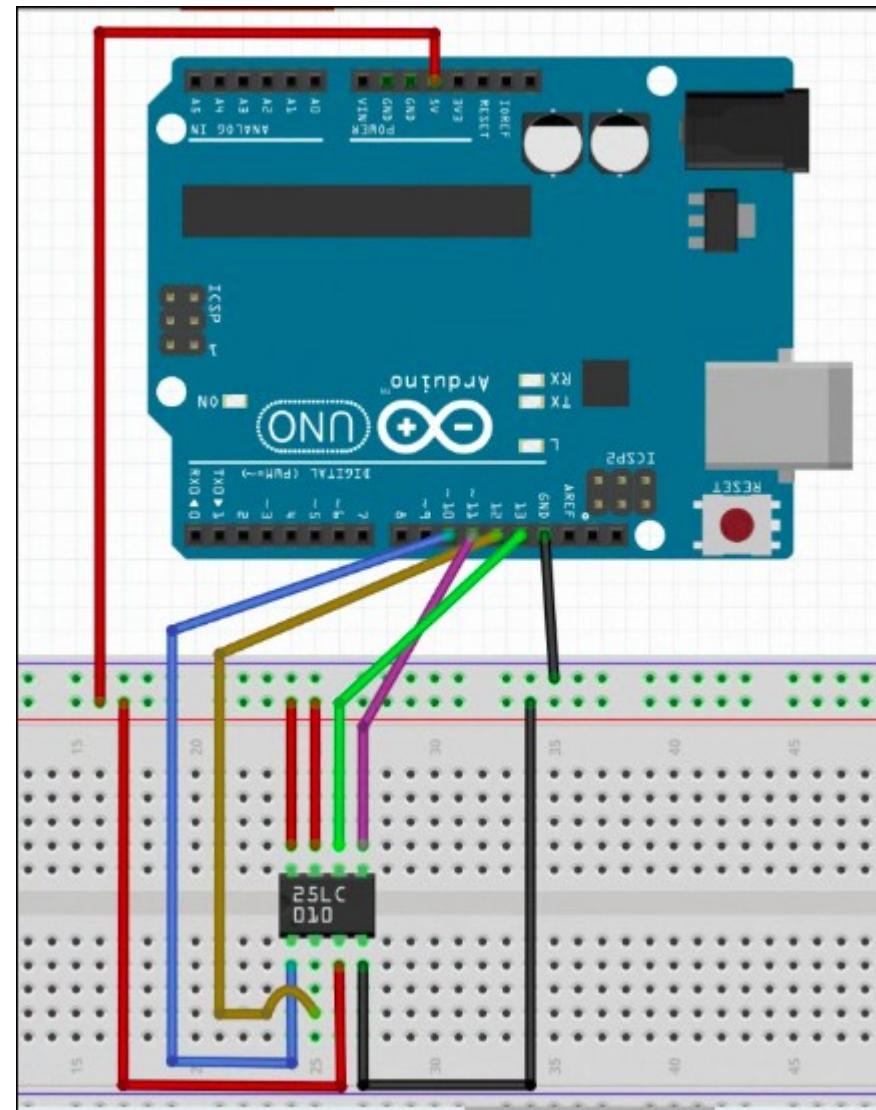




AVR – SPI
(with 외부 EEPROM)

임베디드스쿨2기
Lv1과정
2021. 07. 30
박태인

H/W 연결



포기하면 얻는 건 아무것도 없다.

SPI 통신이란?

SPI(Serial Peripheral Interconnect)

↳ 해석하면 ‘직렬 주변기기 인터페이스’ 뭔가 한글로 바꾸면 더 이상한 느낌이다.

아무튼,

SPI는 1:N 통신(Full duplex)을 지원하는 동기식 통신 방식이다.

SPI 통신을 위해서는 반드시 하나의 Master 와 하나 이상의 Slave 기기가 존재해야 한다.

통신을 위해서는 최소 4개의 선이 필요 한데, MISO, MOSI, SCK, SS가 그것이다.

하나씩 살펴 보면

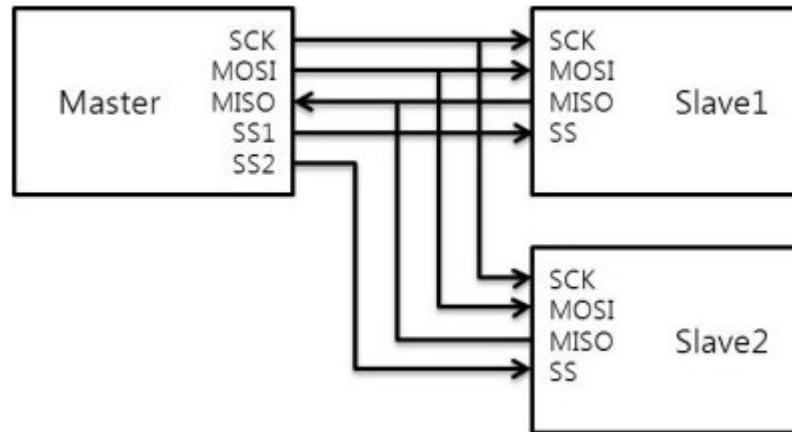
- **MOSI** : Master Out, Slave In – 마스터에서 데이터를 출력하기 위한 신호 선
- **MISO** : Master In, Slave Out – 슬레이브에서 데이터를 출력하기 위한 신호 선
- **SCK** : Clock 신호 선
- **SS** : Slave Select – 데이터를 송수신 할 슬레이브를 선택하기 위한 신호 선

데이터를 전송하고 수신하는 선이 따로 있기 때문에 전송과 수신이 동시에 이루어 질 수 있어서 송수신이 하나의 선으로 이루어지는 I2C 통신에 비해 속도가 빠르다. 그리고 Master 에서 출력하는 선과 Slave 에서 출력하는 선이 정해져 있어 시리얼 통신처럼 Rx, Tx 선이 맨날 헷갈릴 일도 없다.

속도가 빠르다는 장점 때문에 SPI는 주로 빠른 데이터 전송 속도를 필요로 하는 데에 많이 사용된다.
(대표적으로 이더넷 통신이나 SD 카드 같은)

SPI 통신이란?

SPI 통신 역시 I2C 통신처럼 하나의 Master에 여러 개의 Slave가 연결 될 수 있기 때문에, Slave를 선택하기 위한 솔루션이 필요하게 되는데, SPI는 그 방법으로 SS 신호를 사용하고 있다.
SS는 Slave Select 라는 뜻 그대로 Slave를 선택하는 선으로, 하나의 Slave에 하나의 SS 선을 사용한다.

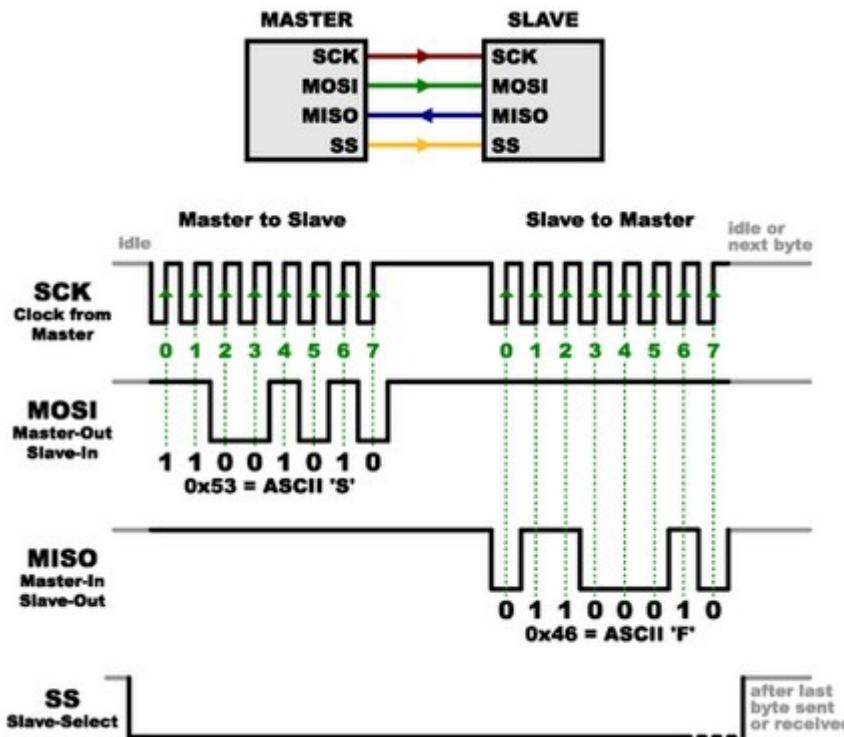


요렇게 SCK와 MOSI, MISO 신호 선은 공통으로 사용한다.

여러 개의 Slave 기기가 Master에 연결 될 수 있지만, Slave 개수 만큼 SS 신호선이 늘어나게 되므로 여러 개의 Slave가 존재 할 때에는 물리적으로 비효율적이다.

동기화 통신 방식으로는 통신에는 클럭 신호가 사용되며, 클럭 신호는 Master에서만 출력 된다.

SPI 통신이란?



SPI 특징 정리

- SPI 통신은 MOSI, MISO, SCK, SS 핀으로 이루어지는 동기식 직렬 통신 방식이다.
- 데이터 출력 신호 선과 입력 신호 선이 따로 있어 동시에 송 수신이 가능하다.
- Slave를 SS 신호로 선택 한다.
- I2C 통신보다 속도가 빠르다.
- SS 신호 선을 이용하면 여러 개의 Slave를 연결 할 수 있지만 Slave 개수 만큼 SS 신호 선이 필요하므로 Slave가 많아 질 경우에는 비효율적이다.

데이터 송수신을 위해 제일 먼저 SS 신호로 Slave를 선택 한 후 클럭 신호를 생성하고, 클럭 신호에 맞춰 데이터를 전송한다.

위 그림에서는 마치 데이터 전송 후 수신 받는 것처럼 되어 있지만, 사실은 데이터를 전송하는 중에도 MISO 신호 선으로 Slave의 데이터가 수신 될 수 있다. **전송과 수신이 동시에 이루어 질 수 있다는 것이 SPI의 가장 큰 장점!**

그림에서는 클럭 신호가 HIGH로 바뀔 때 데이터를 읽는 것으로 되어 있지만, SPI 통신에서는 HIGH로 바뀔 때 데이터를 읽을지 LOW로 바뀔 때 데이터를 읽을 지는 지정 할 수 있다. 그리고 클럭 신호가 HIGH로 먼저 시작하는지, LOW로 먼저 시작하는지 역시 결정 할 수 있다. 물론 Slave와 방식이 동일해야 한다.

EEPROM 이란

이번에 SPI 통신을 사용해서 통신 할 대상은 외부 EEPROM 이다.
여기서 잠깐 EEPROM이 무엇인지 알아 보도록 한다.

1. Memory 종류

- Memory는 데이터를 읽거나 쓰기를 위한 저장 공간입니다.
크게 ROM(롬)과 RAM(램)으로 구분 할 수 있습니다.

1) RAM

- ↳ Random Access Memory의 약자로 읽기도 가능하고, 데이터 저장도 가능합니다.
하지만 전원이 끊어지면 기억되어 있는 데이터들이 소멸 됩니다.

2) ROM

- ↳ Read Only Memory의 약자로 읽기만 가능한 메모리입니다.
전원이 끊어져도 기록된 데이터들이 소멸되지 않는 비휘발성 메모리입니다.
EEPROM 역시 ROM 안에 포함되는 것 입니다. 하지만 특징은 조금 다릅니다.

3) PROM

- ↳ Programmable ROM의 약자로 저장된 데이터를 1회 수정 할 수 있습니다.
내부에 퓨즈가 연결되어 있으며, 데이터를 기록 할 경우 퓨즈가 끊어지게 됩니다.
따라서 수정이 불가능 합니다.

4) EEPROM

- ↳ Electrically Erasable PROM 의 약자로 전기적인 기능을 통해 데이터를 지울 수 있는 ROM 입니다.
정격전압 보다 높은 고전압을 통해 데이터를 지울 수 있으며, 1 바이트 씩 데이터를 지우기 때문에 속도가 느린 편 입니다.
따라서 실시간으로 사용하는 메모리 라기 보다는 중요한 데이터를 백업해 놓는 형태로 사용하는 것이 적합 합니다.

MCU 내부에 있기도 하지만 주로 MCU 내부의 EEPROM은 메모리 크기가 매우 작은 편 입니다.
따라서 때에 따라 외부 EEPROM을 이용 하는 편 입니다.



외부 EEPROM

SPI 통신 코딩 분석(header)

```
main.c 25lc010.h 25lc010.c uart.h uart.c
1 #ifndef __25LC010_H__
2 #define __25LC010_H__
3
4 #include <avr/io.h>
5
6 #define SPI_SS      PB2
7 #define SPI_MOSI    PB3
8 #define SPI_MISO    PB4
9 #define SPI_SCK     PB5
10
11 #define EEPROM_SELECT()    PORTB &= ~(1 << SPI_SS)           SPI 통신 사용 핀 관련 setting
12 #define EEPROM_DESELECT()  PORTB |= (1 << SPI_SS)
13
14 #define EEPROM_READ      0b00000011
15 #define EEPROM_WRITE     0b00000010
16 #define EEPROM_WREN      0b00000110
17 #define EEPROM_RDSR      0b00000101
18
19 #define EEPROM_WRITE_IN_PROGRESS 0
20
21 #define EEPROM_PAGE_SIZE      16
22 #define EEPROM_TOTAL_BYTE     120
23
24 void spi_init(void);
25 void eeprom_change_byte(uint8_t);
26 void eeprom_send_address(uint8_t);
27 uint8_t eeprom_read_status(void);
28 void eeprom_write_enable(void);
29 uint8_t eeprom_read_byte(uint8_t);
30 void eeprom_write_byte(uint8_t, uint8_t);
31 void eeprom_erase_all(void);
32
33 #endif // __25LC010_H__
```

SPI 통신 사용 핀 관련 setting

EEPROM 핀 설정

EEPROM 모드 설정

EEPROM WRITE IN PROGRESS

EEPROM PAGE 및 TOTAL BYTE

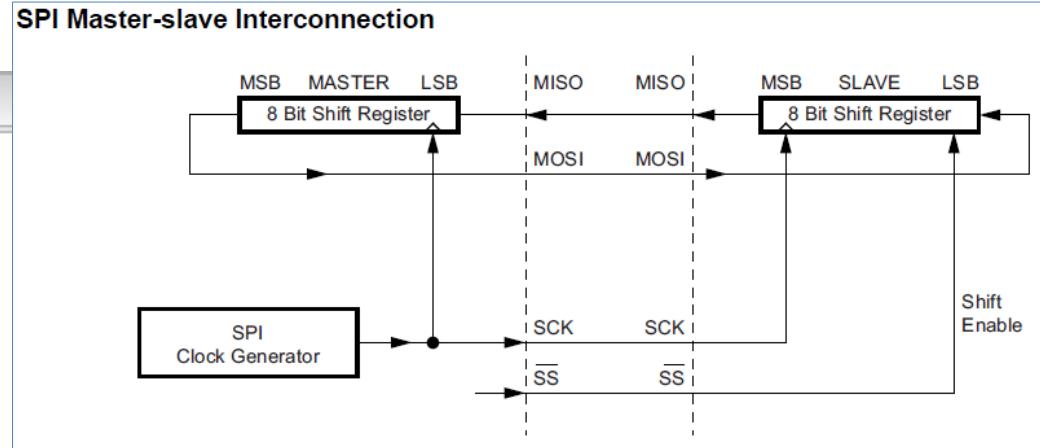
함수 원형 선언

SPI 통신 코딩 분석(25lc010.c)

```

main.c 25lc010.h *25lc010.c uart.h uart.c
1 #include "25lc010.h"
2
3 void spi_init(void) Spi_init
4 {
5     // datasheet 136 page SPI 통신 설명
6
7     // SS핀을 출력 설정
8     // PB2, chip select
9     DDRB |= (1 << SPI_SS); Select 출력
10    // SS핀을 HIGH로 설정하여 EEPROM이 선택되지 않은 상태로 시작!(내부 not 게이트)
11    // high를 넣어서 내부에 not 을 만듦.
12    PORTB |= (1 << SPI_SS);
13    // MOSI핀 출력
14    // master가 보내는거
15    DDRB |= (1 << SPI_MOSI); MOSI 출력
16    // MISO핀 입력
17    // slave가 보내는거
18    DDRB &= ~(1 << SPI_MISO); MISO 출력
19    // SCK핀 출력
20    DDRB |= (1 << SPI_SCK); SCK, Clock 출력
21
22    // SPCR 139,140 page, Master mode
23    // Master Mode
24    SPCR |= (1 << MSTR);
25    // SPI Enable
26    SPCR |= (1 << SPE); Master Mode, SPI enable
27 }

```



Select 출력

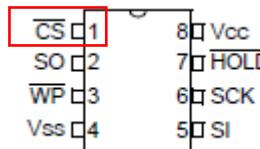
MOSI 출력
Master Out
Slave In

MISO 출력
Master In
Slave Out

SCK, Clock 출력

Master Mode, SPI enable

EEPROM의 선택이 LOW 신호를 입력 하였을 때 선택 되므로 HIGH 시 선택 되지 않음.



위에서 CS가 Chip Select로 SS와 같은 의미.

글자 위에 선이 그어져 있는 것은 LOW의 신호를 넣어 줘야 동작.

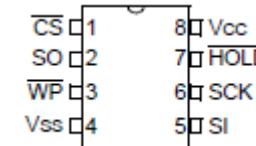
위 그림은 AVR328 136page에 있는 SPI 통신 관련 그림인데,
여기서 보듯이 원형 Q 형태를 그리고,

선생님께서 말씀해주신 부분 중에 주의해야 할 점이 있는데,

Master에서 송신하면 Slave에도 어떤 정보라도 보내야만 제대로 통신이 이루어진다고 하였다.
(위와 같은 원형 Q 형태이기 때문인 것으로 추측 된다.)

SPI 통신 코딩 분석(25lc010.c)

```
main.c 25lc010.h *25lc010.c uart.h uart.c
28
29 void eeprom_change_byte(uint8_t byte) Eeprom_change_byte
30 {
31     // datasheet 142
32     // SPDRO이 25lc로 갈거임 여기서 25lc 데이터 시트 확인 할 것. 1 page 설명
33     // 7번핀 HOLD는 안쓴다. 정지하지 않는다. → 7번 핀에 HIGH 신호를 주어 멈추지 않게 한다.
34     // 5 page serial input 타이밍도, 스코프 파형과 비교.
35
36     // 6 page 내용도 중요함.(2.1, 2.2, 2.3)
37     //데이터 전송 !
38     // 6 을 보냈다는건 write enable
39     SPDR = byte;
40     // 전송 완료에 대한 대기
41     loop_until_bit_is_set(SPSR, SPIF);
42 }
```



전송 완료
까지 대기

요 함수의 경우
다음 행동을 어떤걸 할건지를
정하는 용도로 많이 사용

즉, eeprom 의 모드 설정.

7번 핀에 HIGH 신호를 주어 멈추지 않게 한다.

스코프 파형 비교는 제일 마지막에

SPI 통신 데이터

SPDR – SPI Data Register

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

The SPI data register is a read/write register used for data transfer between the register file and the SPI shift register. Writing to the register initiates data transmission. Reading the register causes the shift register Receive buffer to be read.

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	

- Bit 7 – SPIF: SPI Interrupt Flag

When a serial transfer is complete, the SPIF Flag is set. An interrupt is generated if SPIE in SPCR is set and global interrupts are enabled. If SS is an input and is driven low when the SPI is in master mode, this will also set the SPIF flag. SPIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, the SPIF bit is cleared by first reading the SPI status register with SPIF set, then accessing the SPI data register (SPDR).

SPI 통신 코딩 분석(25lc010.c)

```
44 void eeprom_send_address(uint8_t address)    Eeprom_send_address
45 {
46     eeprom_change_byte(address);      Eeprom 주소 변경(모드 변경)
47 }
48
49 uint8_t eeprom_read_byte(uint8_t address)    Eeprom_read_byte
50 {
51     // 쓰기 파형 다음으로 read 파형
52     // 101
53     //EEPROM 선택
54     EEPROM_SELECT();
55     // 읽기 명령 전송
56     eeprom_change_byte(EEPROM_READ);
57     // 메모리 주소를 전송      Eeprom_read 모드 명령 전송
58     // 50
59
60     eeprom_send_address(address);
61     // Master에서 바이트 값을 전송하여 읽어야 함
62     // 별다른 의미가 없지만 통신 방식이 엽기라 그냥 보내야함(마치 full-duplex 통신을 하기 위한 방법)
63     // SPI가 희안하게 half-duplex full-duplex 다 되지만 뭔가 예매하게 동작함.
64     // 아래 0은 의미 없는 0은 아니고 받아 오기 위한 0이다.
65     eeprom_change_byte(0);
66     //EEPROM 선택 해제
67     EEPROM_DESELECT();          데이터를 받아와서 읽기 위한 초기화
68     // EEPROM 선택 해제
69 }
```

Eeprom 선택

데이터를 받아와서 읽기 위한 초기화

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

Full-deplex 통신방식에서
데이터를 읽기 위해

어떠한 값이라도 보내야 함.

SPI 통신 코딩 분석(25lc010.c)

```
main.c 25lc010.h *25lc010.c uart.h uart.c
70
71     void eeprom_write_enable(void) Eeprom_write_enable
72 {
73     // Slave 선택
74     // low 지만 칩 셀렉트
75     EEPROM_SELECT();
76     // 쓰기 가능하게 설정
77     eeprom_change_byte(EEPROM_WREN);
78     // Slave Select를 HIGH
79     // data sheet 에 굉장히 충실히 감.
80     EEPROM_DESELECT();
81 }
```

흐름도에
따라

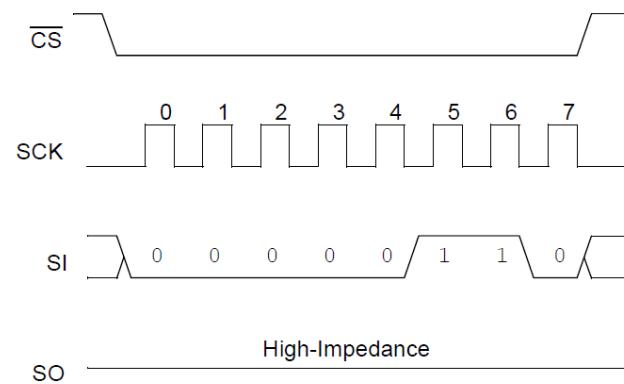
SS 선택 ->
'WREN'명령 ->
SS 해제.

Eeprom 선택

쓰기 가능
가능하게 설정

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

FIGURE 2-4: WRITE ENABLE SEQUENCE (WREN)

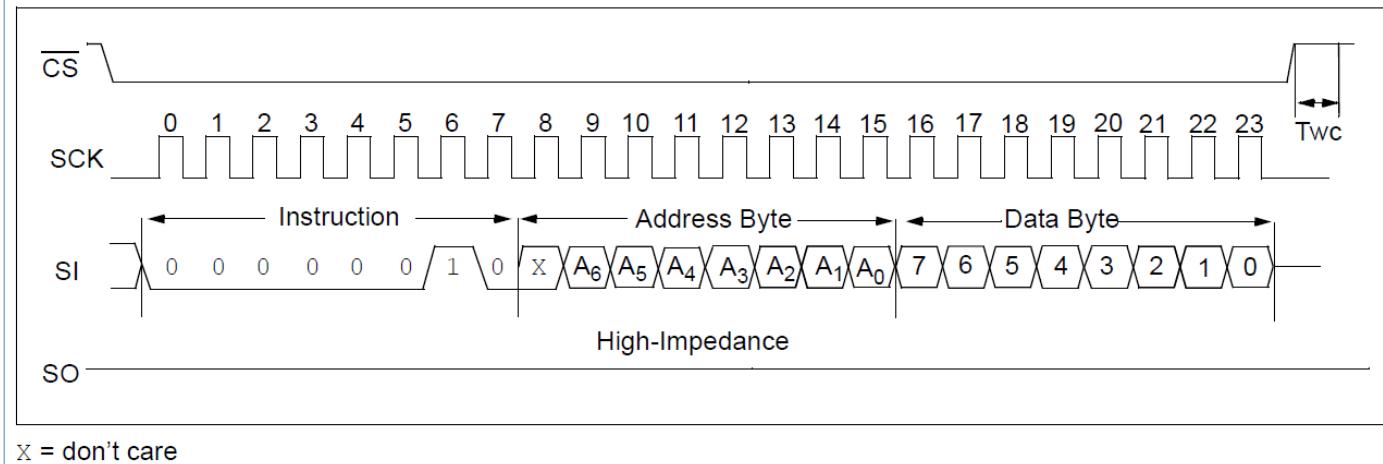


SPI 통신 코딩 분석(25lc010.c)

```
-- // 아래는 마스터가 슬레이브에게 보내는 것이다
83 // 아래는 마스터가 슬레이브에게 보내는 것이다
84 void eeprom_write_byte(uint8_t address, uint8_t data)
85 {
86     // 총 파형 확인 : enable -> EEPROM_WRITE -> address(50) -> data(50)
87
88     // 쓰기 모드
89     // 6= 110
90     eeprom_write_enable(); Write enable
91     // EEPROM 선택
92     EEPROM_SELECT(); Slave 선택
93     // 쓰기 명령 전송
94     // 2 = 00000010          'WRITE'
95     eeprom_change_byte(EEPROM_WRITE);
96     // 주소 전송
97     // 127 = 01111111
98     // 50 = 00110010
99     eeprom_send_address(address);
100    // 데이터 전송
101    eeprom_change_byte(data);
102    // EEPROM 선택 해제
103    EEPROM_DESELECT();
104
105    // 쓰기 완료까지 대기
106    // BV는 bit select, 해당 비트 설정
107    while (eeprom_read_status() & _BV(EEPROM_WRITE_IN_PROGRESS))
108    {
109        ;
110    }
111 }
```

Eeprom_write_byte

FIGURE 2-2: BYTE WRITE SEQUENCE



상태 레지스터 값이 0 과 & 되어 쓰기 완료 시 까지 대기.
read_status 는 뒤에 나옴.

SPI 통신 코딩 분석(25lc010.c)

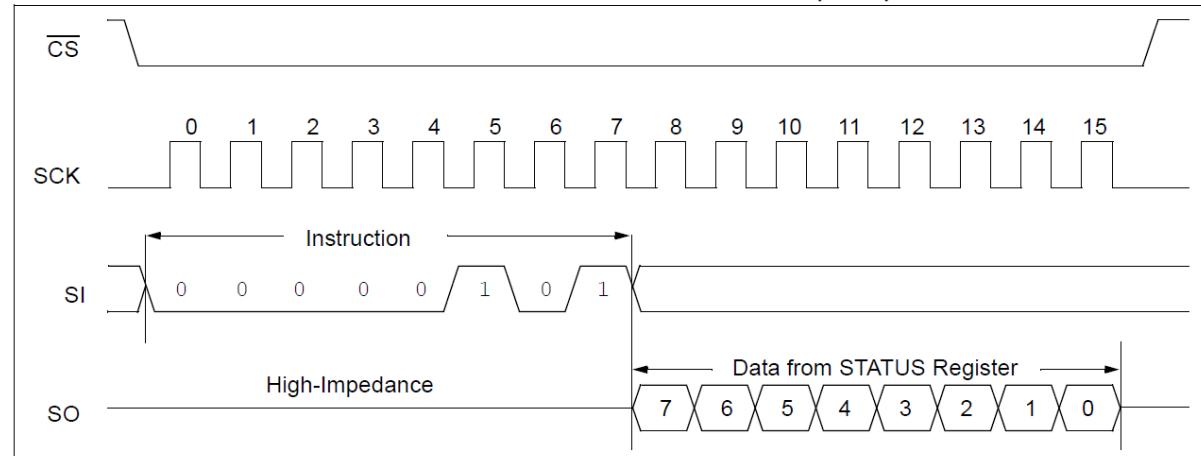
```
113     uint8_t eeprom_read_status(void)
114 {
115     // EEPROM 선택
116     EEPROM_SELECT();
117     // 상태 레지스터 읽기 명령 전송
118     eeprom_change_byte(EEPROM_RDSR);
119     // 상태 레지스터 값 읽기
120     eeprom_change_byte(0);
121     // EEPROM 선택 해제
122     EEPROM_DESELECT();
123
124     return SPDR; 상태 레지스터
125
126 }
```

데이터를 읽기 위해
임의의 '0'이라는
값을 전송

Eeprom_read_status

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

FIGURE 2-6: READ STATUS REGISTER TIMING SEQUENCE (RDSR)



SPI 통신 코딩 분석(25lc010.c)

```
127 void eeprom_erase_all(void) Eeprom_erase_all
```

```
128 {
129     uint8_t i;
130     uint16_t page_address = 0;
```

16byte 블록 단위 처리가 가능한 개념의 page

```
132     while (page_address < EEPROM_TOTAL_BYTE) EEPROM_TOTAL_BYTE는 128
```

```
133 {
134     // 쓰기 가능 모드
```

```
135     eeprom_write_enable(); 6(110)날림
```

```
136     // EEPROM 선택
```

```
137     EEPROM_SELECT();
```

```
138     // 쓰기 명령 전송
```

```
139     eeprom_change_byte(EEPROM_WRITE);
```

```
140     // 페이지 시작 주소 전송
```

```
141     eeprom_send_address(page_address);
```

```
142     // 페이지 단위 데이터 전송
```

```
143     for (i=0; i < EEPROM_PAGE_SIZE; i++)
```

```
144     {
145         eeprom_change_byte(0); 전체 EEPROM
```

크기 만큼
'0'값 write

```
146     }
```

```
147     // EEPROM 선택 해제
```

```
148     EEPROM_DESELECT();
```

```
149     // 페이지 변경
```

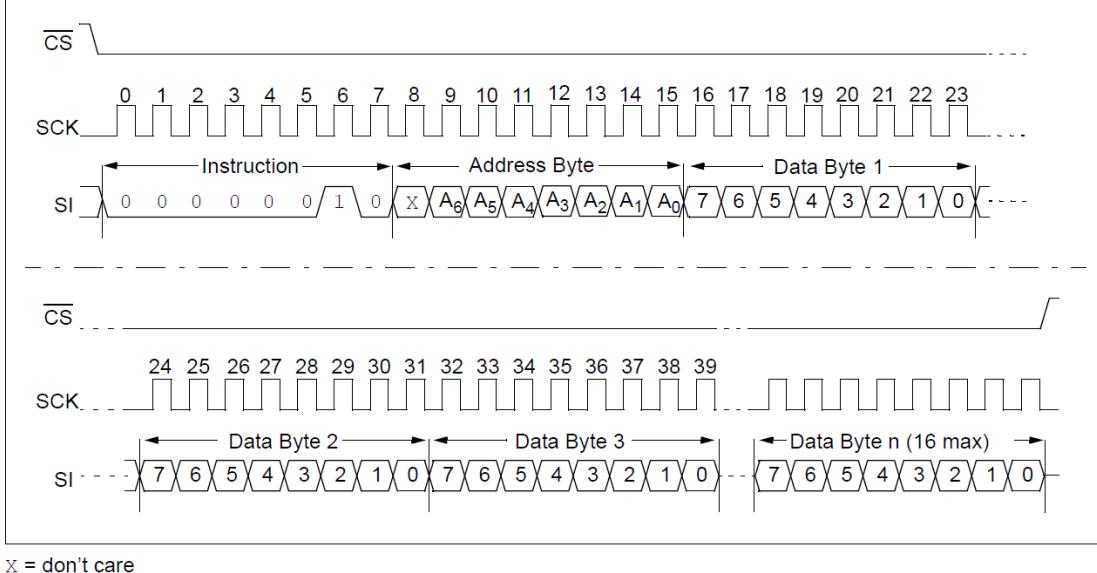
```
150     page_address += EEPROM_PAGE_SIZE; // 쓰기 완료 대기
```

```
151     while (eeprom_read_status() & _BV(EEPROM_WRITE_IN_PROGRESS))
```

쓰기 완료 대기

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

FIGURE 2-3: PAGE WRITE SEQUENCE



EEPROM 사이즈 만큼 값을 page_address에 저장
-> 다음 while문 빠져나가는 조건이 됨.

결론 : 전체 크기 128 byte에 전부 '0'의 값으로 초기화 하게 된다.

SPI 통신 코딩 분석(25lc010.c)

```
main.c 25lc010.h *25lc010.c uart.h uart.c
1  /*
2   */
3 #define F_CPU 16000000UL
4
5 #include <avr/io.h>
6 #include <string.h>          각종 헤더 선언
7 #include "uart.h"
8 #include "25lc010.h"
9
10 int main(void)           main
11 {
12     uint8_t i;
13
14     spi_init();            Spi 통신 init (with EEPROM)
15     UART_INIT();          UART 통신 init (with putty)
16
17     for(i=0; i < 128; i++)
18     {
19         eeprom_write_byte(i, i);      Write 첫번째 인자 : address
20                                     두번째 인자 : data
21
22         for(i=0; i < 128; i++)
23         {
24             uart_print_8bit_num(eeprom_read_byte(i));    즉, 1 ~ 128 까지 주소당 1 byte 씩 데이터 write
25             UART_transmit('\r');
26             UART_transmit('\n');
27         }
28
29     // Insert code
30 }
```

SPI 통신 코딩 분석(25lc010.c)

```
29 // Insert code
30
31 while(1)
32 {
33     for(i=127; i < 128; i++)
34     {
35         //eeprom_write_byte(i, i);
36         eeprom_write_byte(50, 50);
37     }
38
39     _delay_ms(1000); // 스코프 찍어 보기 위한 딜레이
40
41     for(i=127; i < 128; i++)
42     {
43         //uart_print_8bit_num(eeprom_read_byte(i));
44         uart_print_8bit_num(eeprom_read_byte(50));
45         UART_transmit('\r');
46         UART_transmit('\n');
47     }
48     _delay_ms(1000); // 스코프 찍어 보기 위한 딜레이
49
50
51     return 0;
52 }
53
```

main

While 동작으로
'50'의 address에 data를 write 후

Delay 1초

Eeprom에 Write 된 data를

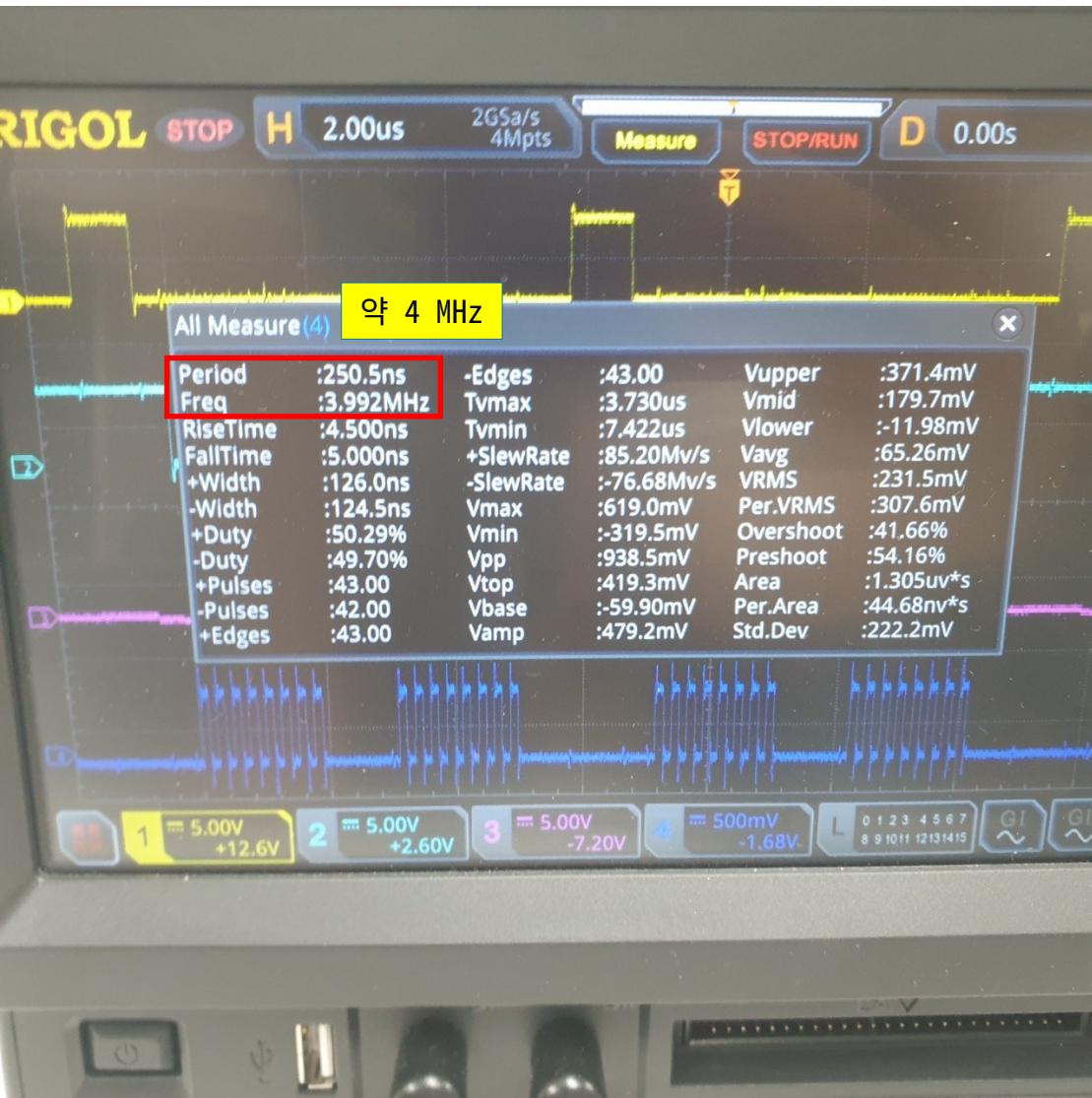
Read 하여 uart에 출력하고

Delay 1초

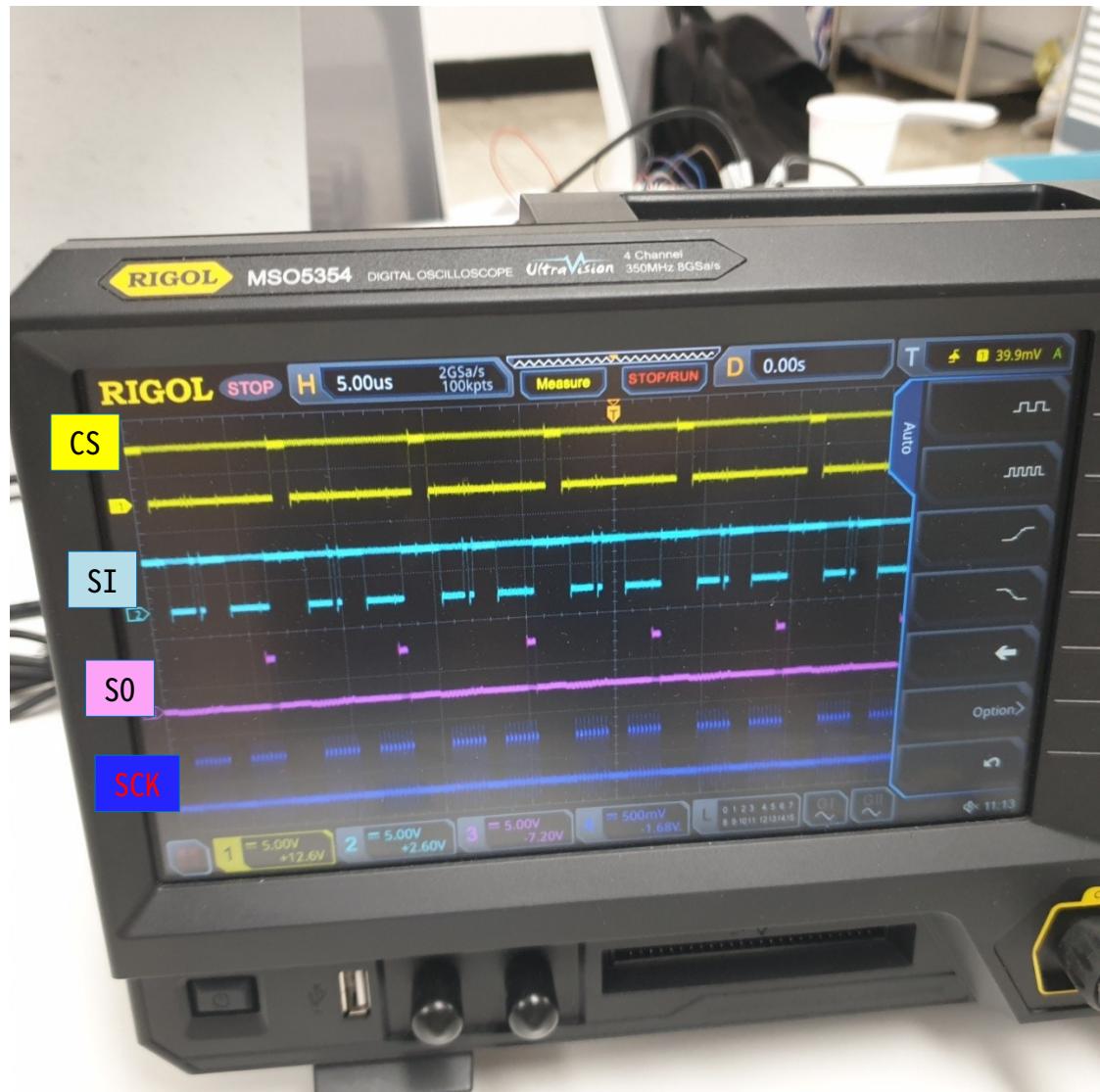
각각의 write와 read 동작 사이에 1초씩
Delay를 준 이유는 스코프에서 동작 파형을 살펴 보기 위함.

동작 파형 분석(Scope)

SCK 클럭 주파수 확인



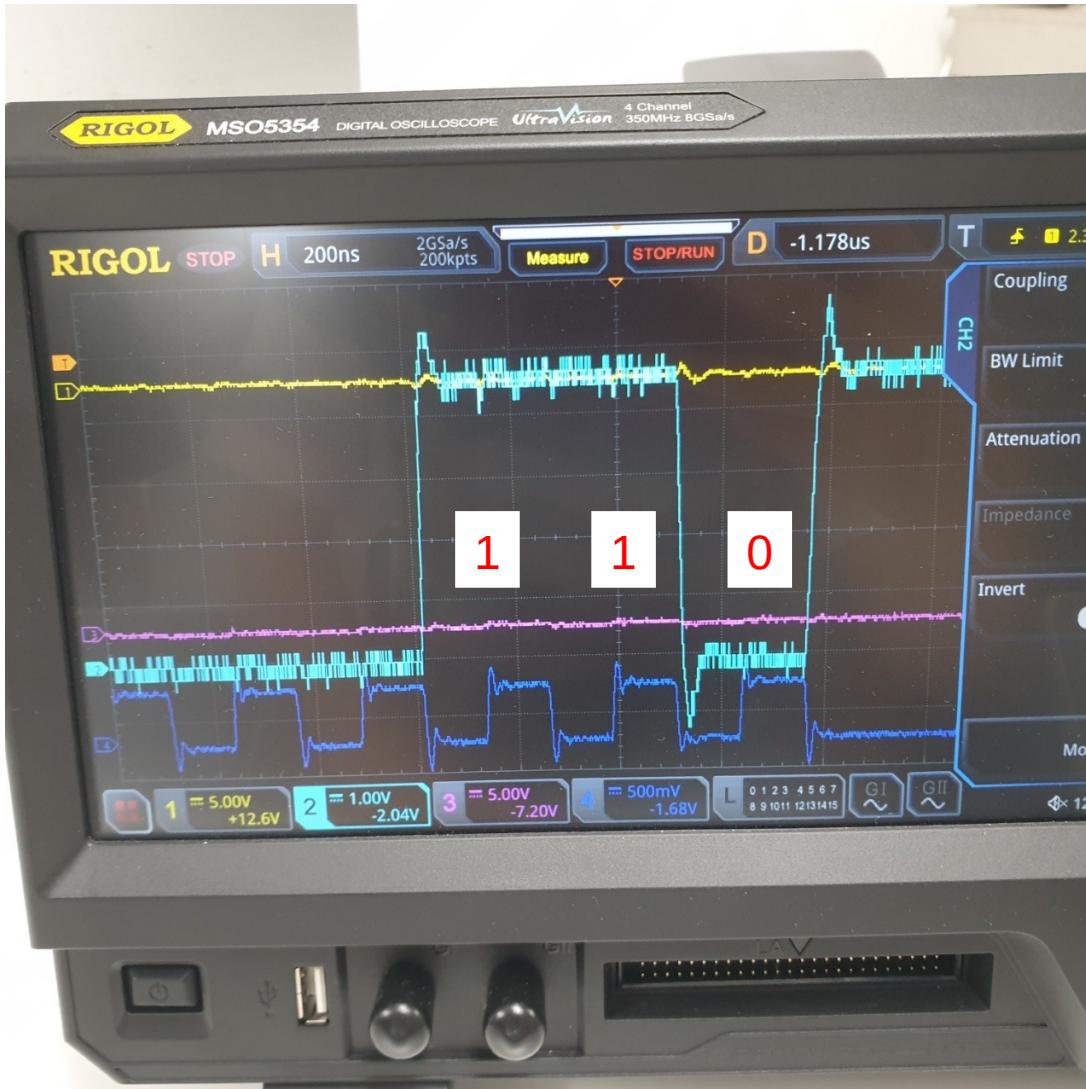
전체적 파형 모습



동작 파형 분석(Scope)

Write enable 단계

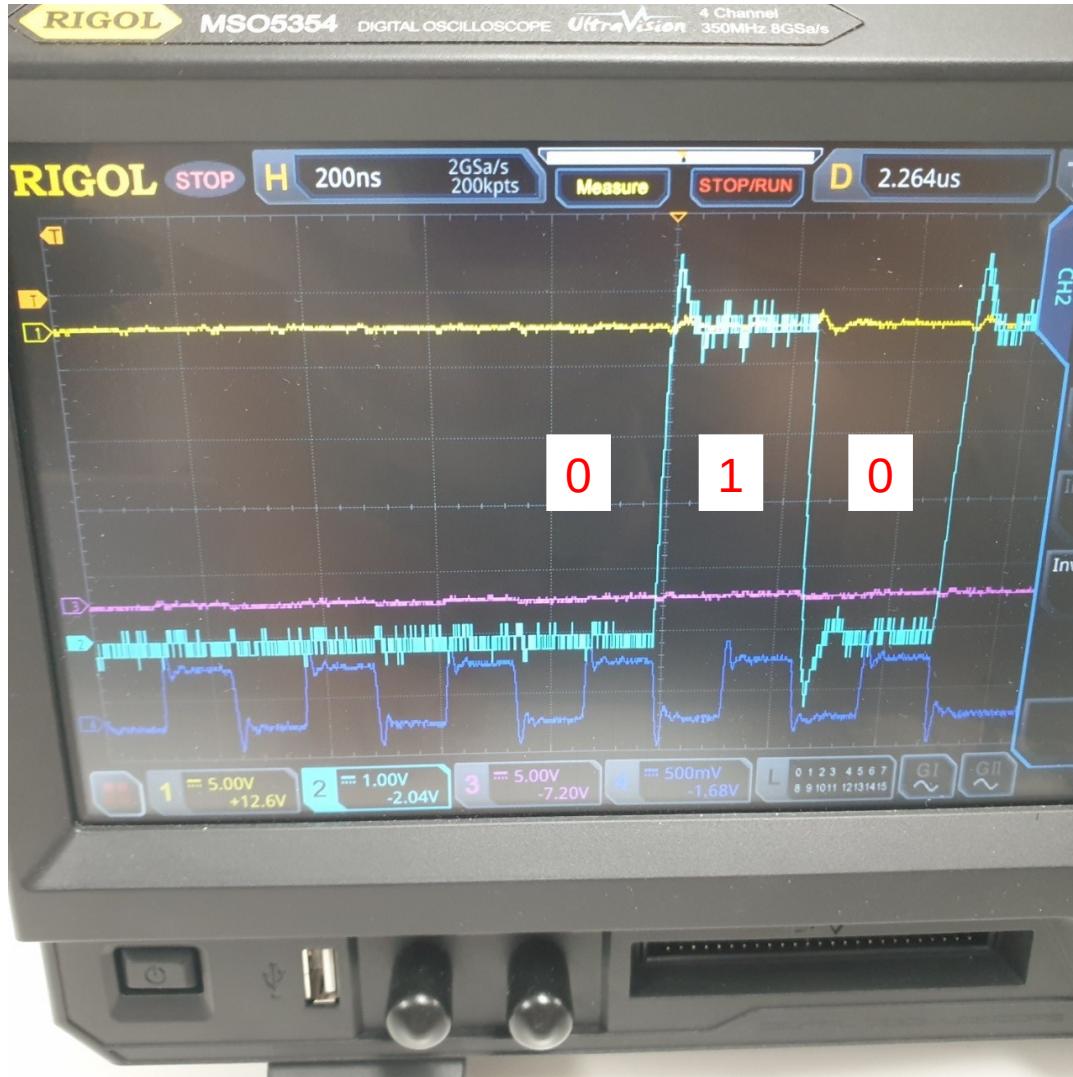
지금부터 write 후 read 하는 과정을 파형으로 관찰해 본다.



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

동작 파형 분석(Scope)

Write 단계

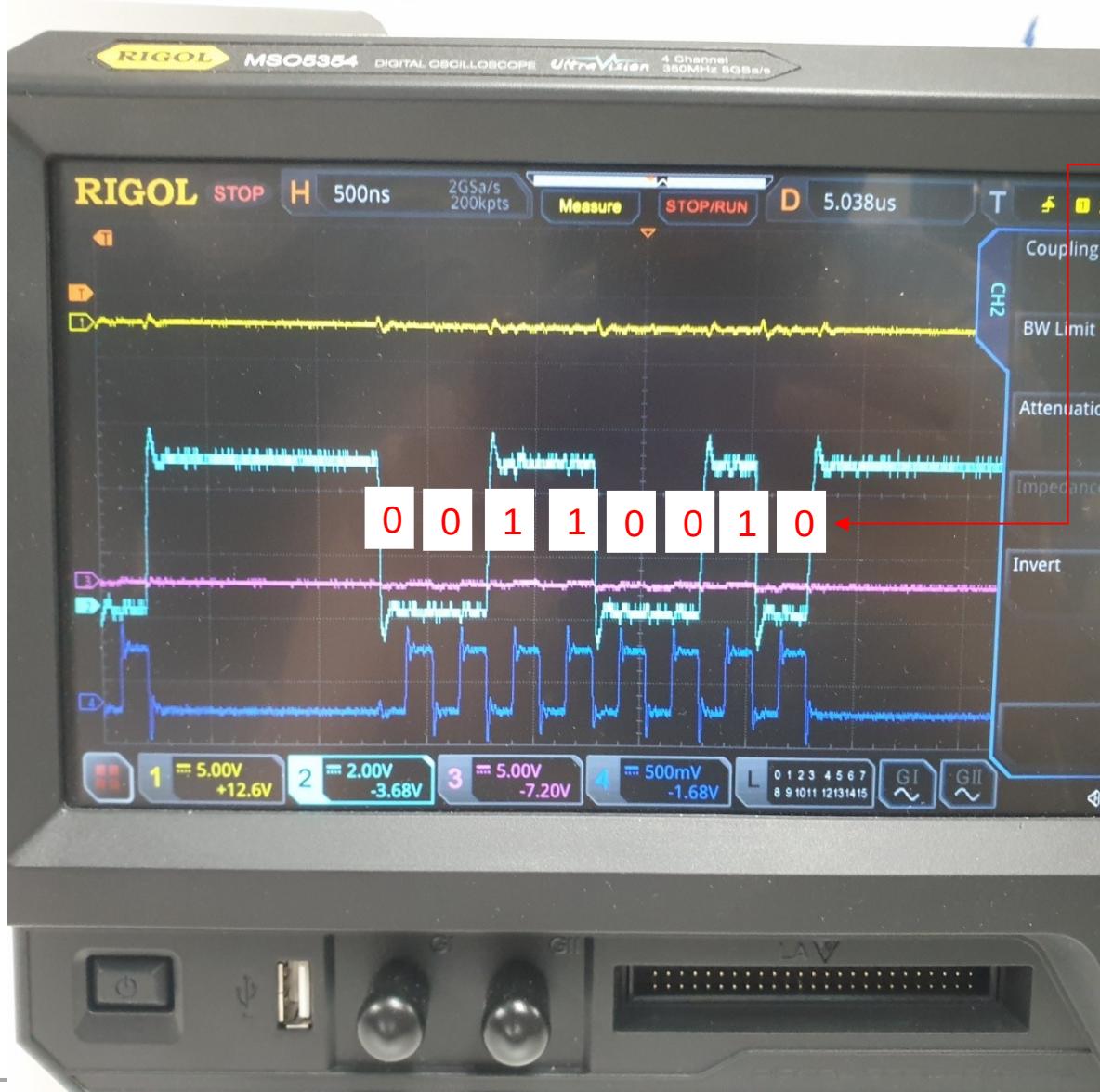


Instruction Name	Instruction Format	Description
READ	0000 x011	Read data from memory array beginning at selected address
WRITE	0000 x010	Write data to memory array beginning at selected address
WRDI	0000 x100	Reset the write enable latch (disable write operations)
WREN	0000 x110	Set the write enable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

동작 파형 분석(Scope)

Write 주소(50) 날림

50

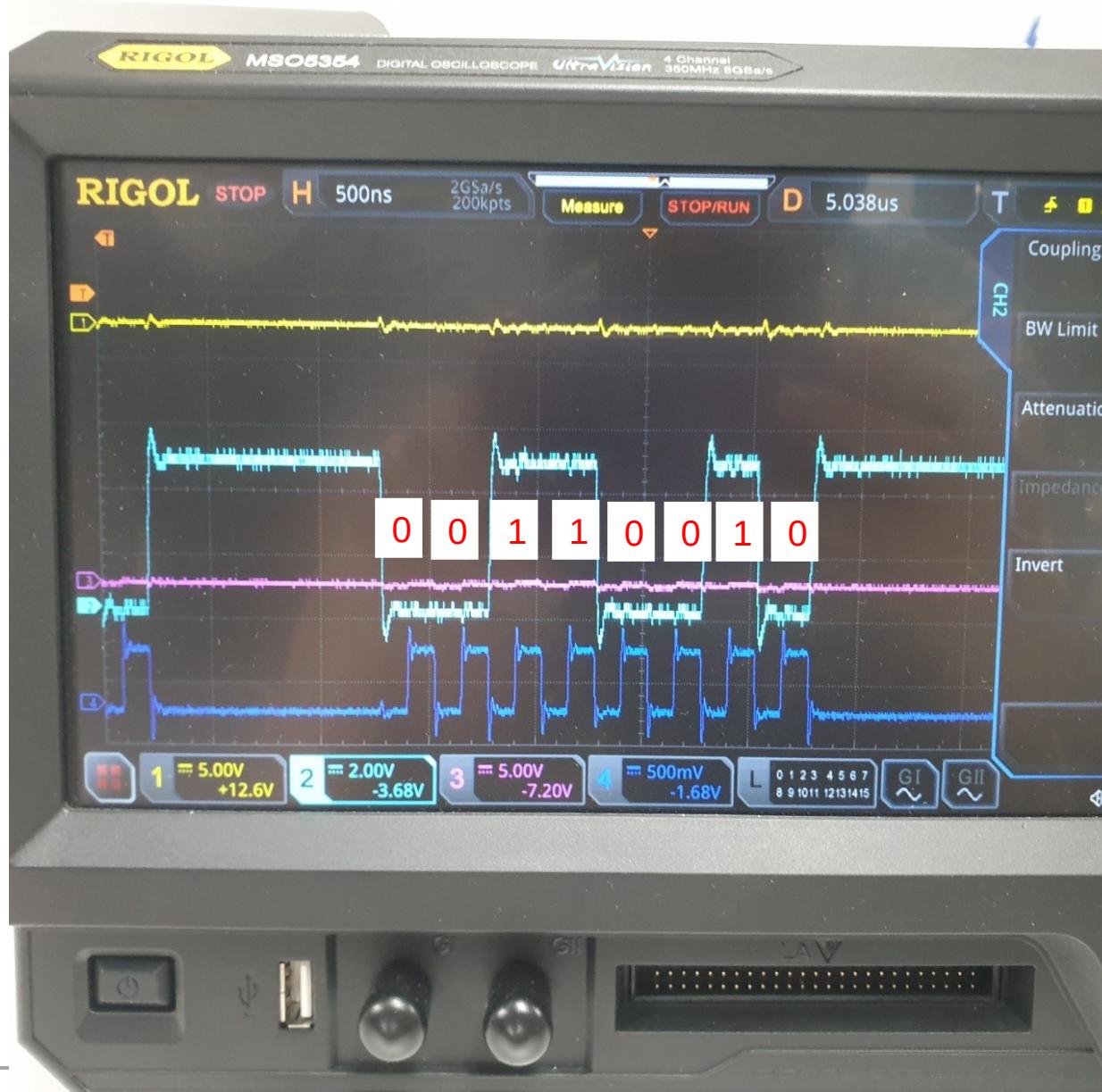


HEX 32
DEC 50
OCT 62
BIN 0011 0010

포기하면 얻는 건 아무것도 없다.

동작 파형 분석(Scope)

50 값 전송



50의 값이 클럭 동기에 맞춰 전송 되는 것을 볼 수 있다.

포기하면 얻는 건 아무것도 없다.

동작 파형 분석(Scope)

Read 상태 레지스터 읽기



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

동작 파형 분석(Scope)

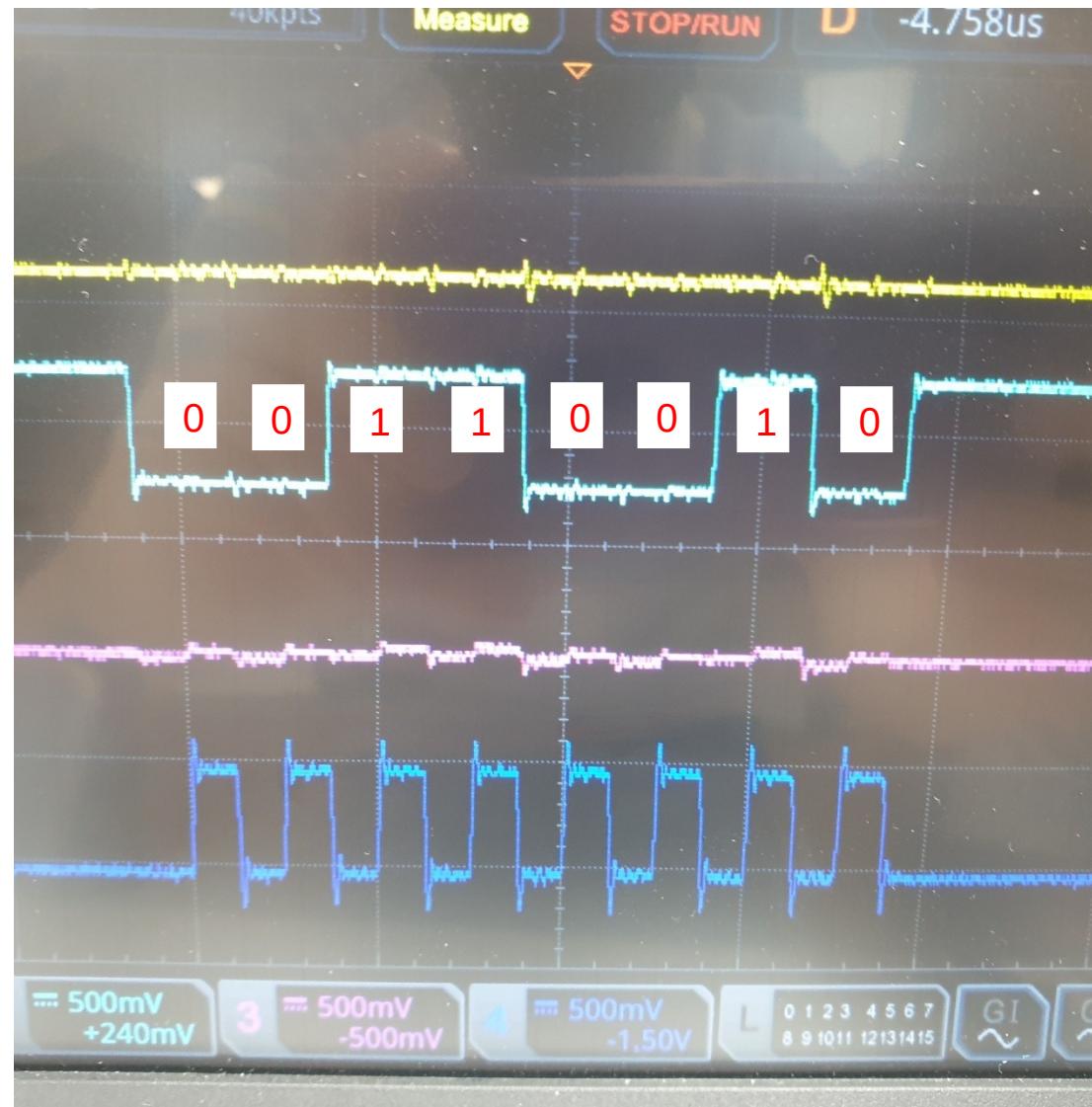
Read 모드



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

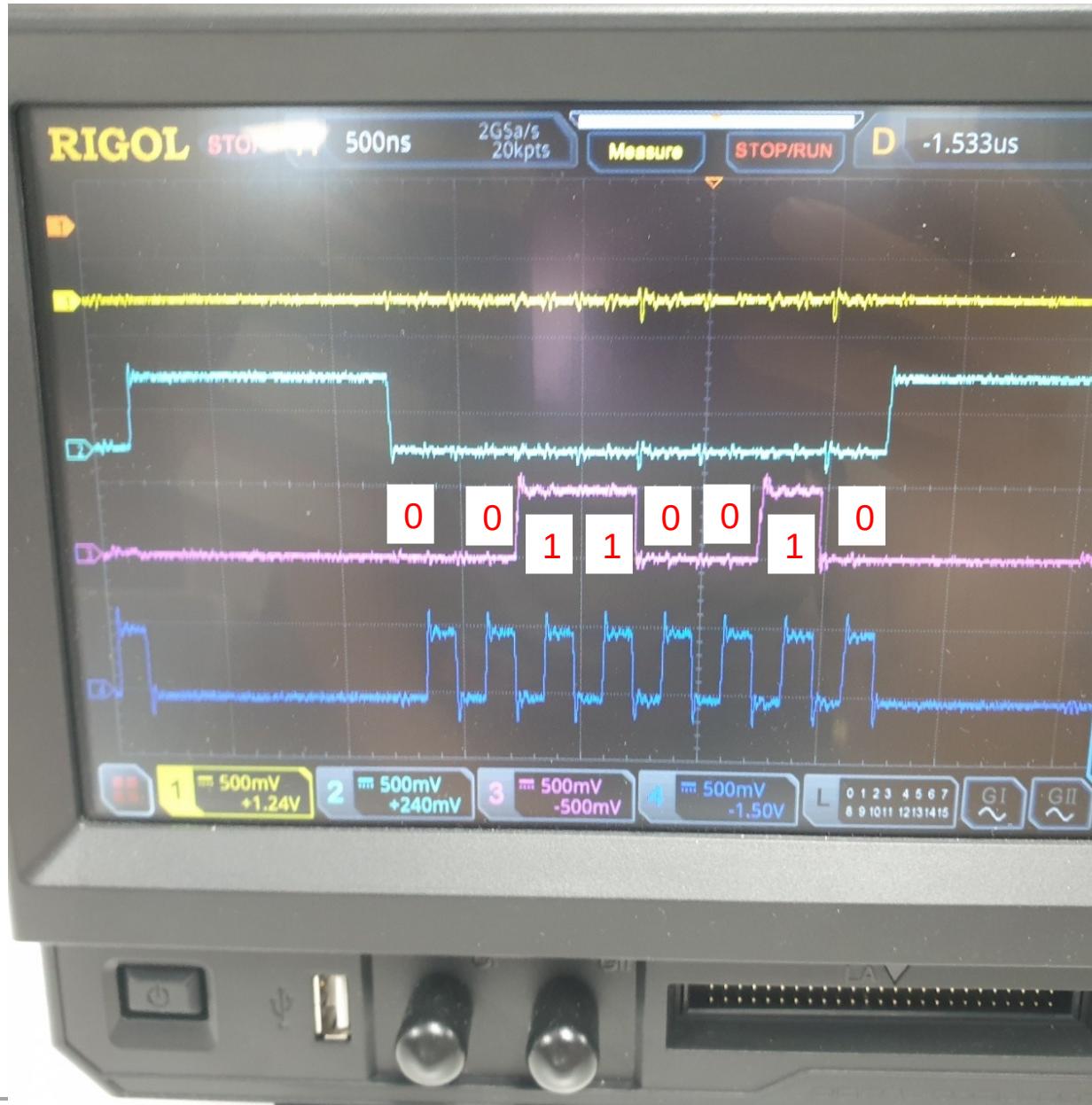
동작 파형 분석(Scope)

Read 할 Address(50) 날림

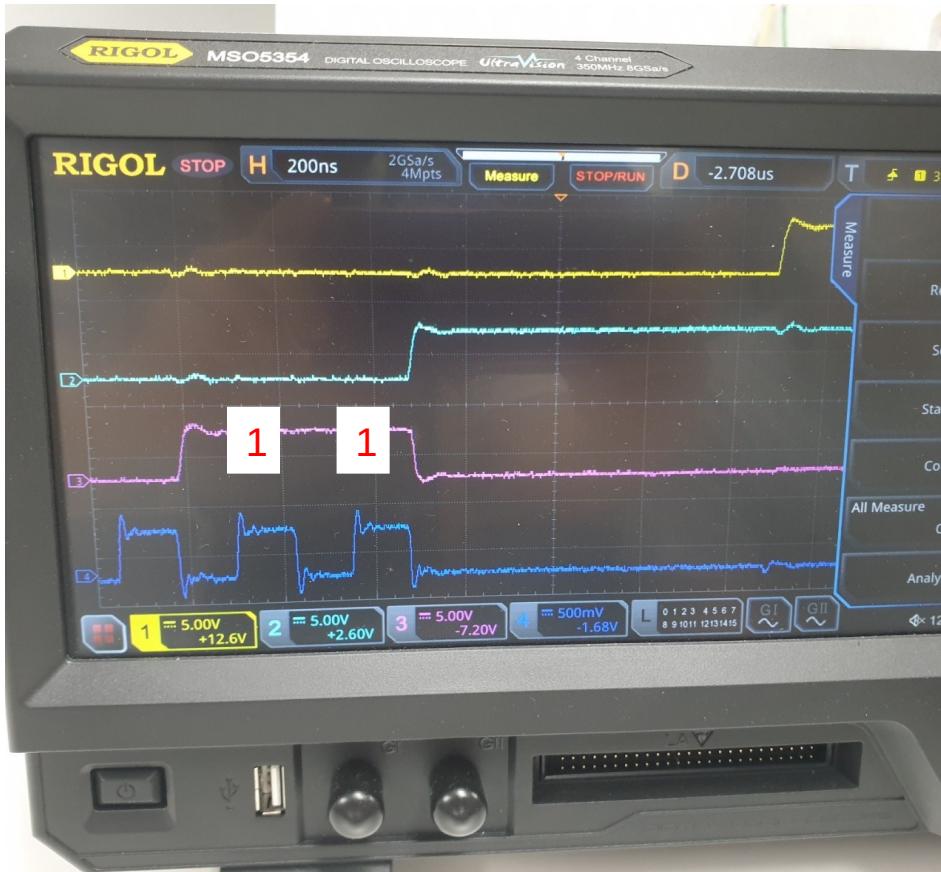


동작 파형 분석(Scope)

최종 출력(50) - 분홍선



동작 파형 분석(Scope) – Raed 시 파형 주의



EEPROM에서 데이터를 Read 할 때

S0(분홍색) 선으로 신호가 나오게 되는데,
우측과 같이 신호가 제대로 나오지 않고
1의 값이 찍혀 나올 때가 있다.

이것은 아직 EEPROM이 앞의 수행을 덜 하거나 했을 경우
데이터를 Out 할 준비가 덜 되었기 때문이다.

(우측 내용 참조.)

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.

질문

```
main.c 25lc010.h 25lc010.c uart.h uart.c
1 #ifndef __25LC010_H__
2 #define __25LC010_H__
3
4 #include <avr/io.h>
5
6 #define SPI_SS    PB2
7 #define SPI_MOSI  PB3
8 #define SPI_MISO  PB4
9 #define SPI_SCK   PB5
10
11 #define EEPROM_SELECT()  PORTB &= ~(1 << SPI_SS)
12 #define EEPROM_DESELECT() PORTB |= (1 << SPI_SS)
13
14 #define EEPROM_READ    0b00000011
15 #define EEPROM_WRITE   0b00000010
16 #define EEPROM_WREN    0b00000110
17 #define EEPROM_RDSR    0b00000101
18
19 #define EEPROM_WRITE_IN_PROGRESS  0
20
21 #define EEPROM_PAGE_SIZE      16
22 #define EEPROM_TOTAL_BYTE     120
23
24 void spi_init(void);
25 void eeprom_change_byte(uint8_t);
26 void eeprom_send_address(uint8_t);
27 uint8_t eeprom_read_status(void);
28 void eeprom_write_enable(void);
29 uint8_t eeprom_read_byte(uint8_t);
30 void eeprom_write_byte(uint8_t, uint8_t);
31 void eeprom_erase_all(void);
32
33 #endif // __25LC010_H__
```

Features:

- 10 MHz max. Clock Frequency
- Low-Power CMOS Technology:
 - Max. Write Current: 5 mA at 5.5V, 10 MHz
 - Read Current: 5 mA at 5.5V, 10 MHz
 - Standby Current: 5 µA at 5.5V
- 128 x 8-bit Organization
- Write Page mode (up to 16 bytes)
- Sequential Read

eeprom에서 의미 하는 page는 어떤 의미로 사용 하였다고 이해 하면 될까요??

eeprom의 총 용량이 $128 \text{ byte} \times 8 = 1024 \text{ byte} = 1 \text{ KB}$ 라면

생각에는 page는 8

Total byte는 128의 값이 되어야 할 것 같은데

16과 120이 되는 이유가 무엇 일까요?



링크쌤

답변

보편적으로 I/O 장치들은 속도가 느리기 때문에 특정한 블록(Block)단위로 데이터를 처리합니다.
그리고 위에 적혀 있듯이 128 x 8 bit Organization이므로 128 바이트가 맞습니다.

Memory Hierarchy 관점에서 I/O 발생을 최소화하기 위해

쓰기의 경우 16바이트까지 몰아서 처리하는 것이 가능하다는 표현입니다.

저희는 저런 블록 기능은 고려하지 않고 단순히 1바이트씩 처리했습니다.

이번에 레벨2에서 진행하는 리눅스 커널의 경우도 DRAM의 페이지 프레임이 이와 동일한 개념을 가집니다.
(물론 용량은 좀 더 큅니다 ^^)

128이랑 120이 얼핏보면 비슷해보일 수 있을것 같네요.
제가 작성한 코드쪽을 살펴보니 128로 되어 있습니다.
오태내신것 같습니다.

128이 되는 것이 맞습니다!

여기서 page는 eeprom datasheet에서 16 byte라고 되어 있기 때문에 16인가요?

질문

```
main.c 25lc010.h *25lc010.c uart.h uart.c
28
29     void eeprom_change_byte(uint8_t byte)
30 {
31     // datasheet 142
32     // SPDR이 25lc로 갈거임 여기서 25lc 데이터 시트 확인 할 것. 1 page 설명
33     // 7번핀 HOLD는 안쓴다. 정지하지 않는다.
34     // 5 page serial input 타이밍도, 스코프 파형과 비교.
35
36     // 6 page 내용도 중요함.(2.1, 2.2, 2.3)
37     //데이터 전송 !
38     // 6 을 보냈다는건 write enable
39     SPDR = byte;
40     // 전송 완료에 대한 대기
41     loop_until_bit_is_set(SPSR, SPIF);
42 }
```

질문1) Change byte 함수의 역할은 Spi 로 전송할 데이터를 SPDR에 넣고 전송 완료 했는지 보고 받는 것 인가요?

답변

맞습니다.

```
49     uint8_t eeprom_read_byte(uint8_t address)
50 {
51     // 쓰기 파형 다음으로 read 파형
52     // 101
53     //EEPROM 선택
54     EEPROM_SELECT();
55     // 읽기 명령 전송
56     eeprom_change_byte(EEPROM_READ);
57     // 메모리 주소를 전송
58     // 50
59     eeprom_send_address(address);
60     // Master 에서 바이트 값을 전송하여 읽어야 함
61     // 별다른 의미가 없지만 통신 방식이 업기라 그냥 보내야함(마치 full-duplex 통신을 하기 위한 방법)
62     // SPI가 희안하게 half-duplex full-duplex 다 되지만 원가 애매하게 동작함.
63     // 아래 0 은 의미 없는 0 은 아니고 받아 오기 위한 0 이다.
64     eeprom_change_byte(0);
65     //EEPROM 선택 해제
66     EEPROM_DESELECT();
67
68     return SPDR;
69 }
```

질문2) [빨간 밑줄] 읽기 통신을 하기 위해 임의의로 값을 날리는 것이 아닌 Change byte면 SPDR spi 데이터 레지스터에 0을 넣는다는 건데 이것은 무엇인가를 읽어 오기 위해 0 으로 setting 해둔다는 것으로 이해 하면 될까요?

답변

네, 맞습니다.

질문

```
49     uint8_t eeprom_read_byte(uint8_t address)
50 {
51     // 쓰기 파형 다음으로 read 파형
52     // 101
53     //EEPROM 선택
54     EEPROM_SELECT();
55     // 읽기 명령 전송
56     eeprom_change_byte(EEPROM_READ);
57     // 메모리 주소를 전송
58     // 50
59     eeprom_send_address(address);
60     // Master에서 바이트 값을 전송하여 읽어야 함
61     // 별다른 의미가 없지만 통신 방식이 업기라 그냥 보내야함(마치 full-duplex 통신을 하기 위한 방법)
62     // SPI가 희안하게 half-duplex full-duplex 다 되지만 원가에 매하게 동작함.
63     // 아래 0은 의미 없는 0은 아니고 받아 오기 위한 0이다.
64     eeprom_change_byte(0);
65     //EEPROM 선택 해제
66     EEPROM_DESELECT();
67
68     return SPDR;
69 }
```

답변

3번: 저희 수업 시간에 스코프를 가지고 파형을 찍었습니다. 밑으로 가라앉아 있는 상태에서 데이터가 나왔었죠? 데이터시트의 타이밍도와 파형 사진 찍어놓은 부분을 다시 한 번 차분하게 살펴보시길 바랍니다.

아래 파형에서 볼 수 있듯이 S0(분홍)가 '0'일 때
SI(하늘)가 나오듯이
데이터가 나오는 타이밍에
파형과 같은
값이 필요하다!

질문2) [빨간 밑줄] 읽기 통신을 하기 위해 임의의로 값을 날리는 것이 아닌 Change byte면 SPDR spi 데이터 레지스터에 0을 넣는다는 건데 이것은 무엇인가를 읽어 오기 위해 0으로 setting 해둔다는 것으로 이해하면 될까요?

질문3) [초록 밑줄] 위에서 eeprom_change_byte 로 0 값을 날렸기 때문에 SPDR에는 무조건 0의 값이 return 되는 것 일까요?



포기하면 얻는 건 아무것도 없다.

질문

```
83 // 아래는 마스터가 슬레이브에게 보내는 것이다
84 void eeprom_write_byte(uint8_t address, uint8_t data)
85 {
86     // 총 파형 확인 : enable -> EEPROM_WRITE -> address(50) -> data(50)
87
88     // 쓰기 모드
89     // 6= 110
90     eeprom_write_enable();
91     // EEPROM 선택
92     EEPROM_SELECT();
93     // 쓰기 명령 전송
94     // 2 = 00000010
95     eeprom_change_byte(EEPROM_WRITE);
96     // 주소 전송
97     // 127 = 01111111
98     // 50 = 00110010
99     eeprom_send_address(address);
100    // 데이터 전송
101    eeprom_change_byte(data);
102    // EEPROM 선택 해제
103    EEPROM_DESELECT();
104
105    // 쓰기 완료까지 대기
106    // BV는 bit select, 해당 비트 세팅
107    while (eeprom_read_status() & _BV(EEPROM_WRITE_IN_PROGRESS))
108    {
109        ;
110    }
111 }
```

답변

4번: 수업시간에도 설명드렸지만 _BV는 1 << 쉬프팅비트로 구성됩니다. 그러므로 0이 아닌 1이겠죠 ?

↳ BV에 대한 이해가 부족 했었다고 볼 수 있다.

질문4) EEPROM_WRITE_IN_PROGRESS 는 0 으로 define 되어 있는데,
이 값과 status 값을 &(and)시키면
Status 값에 상관 없이 무조건 while 조건은 0 의 값을
띄워서 대기 상태가 되는 경우는 없게 되는 걸까요??

질문

```
113     uint8_t eeprom_read_status(void)
114     {
115         // EEPROM 선택
116         EEPROM_SELECT();
117         // 상태 레지스터 읽기 명령 전송
118         eeprom_change_byte(EEPROM_RDSR);
119         // 상태 레지스터 값 읽기
120         eeprom_change_byte(0);
121         // EEPROM 선택 해제
122         EEPROM_DESELECT();
123
124         return SPDR;
125     }
126
```

질문5) 여기서 change_byte 로 0 값을 날려 주는 것은 Read 동작을 하기위해 임의로 0 값을
날려 주는 것 일까요? (SPI 통신 특징상)

답변

5번: 3번 질문과 연결되는 질문으로 맞습니다.