

13주차

임베디드 I2C/SPI 통신



최민

학습목표

라즈베리파이(Raspberry Pi) I2C 통신 사용법 이해하기

학습활동

라즈베리파이 SPI를 시스템 호출을 사용하여 구현할 수 있다.

라즈베리파이 I2C 를 활용하여 통신할 수 있다.

라즈베리파이 에서 디바이스 드라이버를 활용할 수 있다.

SPI와 I2C의 비교

I2C는 SDA 한 개 line으로 데이터 송수신 하므로 통신 속도가 SPI 방식보다 상대적으로 느리며, 데이터 송수신을 동시에 할 수 없음

 센서를 하나 더 추가 연결할 때, 전선 한 개를 버스선에 연결하면 됨

SPI는 MOSI와 MISO 두개의 line으로 데이터 송수신을 서로 분리하므로, I2C보다 통신속도가 빠른 장점, 또한, 데이터 송수신을 동시에 할 수 있음.

 센서 하나 더 추가 연결할 때, I2C 보다는 조금 더 많은 연결이 필요함.

임베디드 SPI 통신 예제

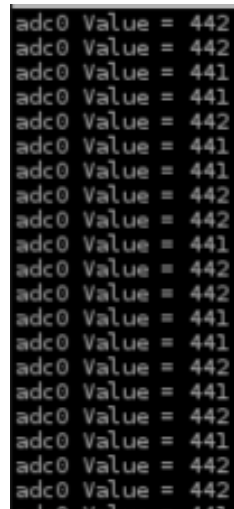
컴파일

```
Gcc -o wiringPi_spi wiringPi_spi.c -lwiringPi
```

실행방법

```
./wiringPi_spi
```

실행결과

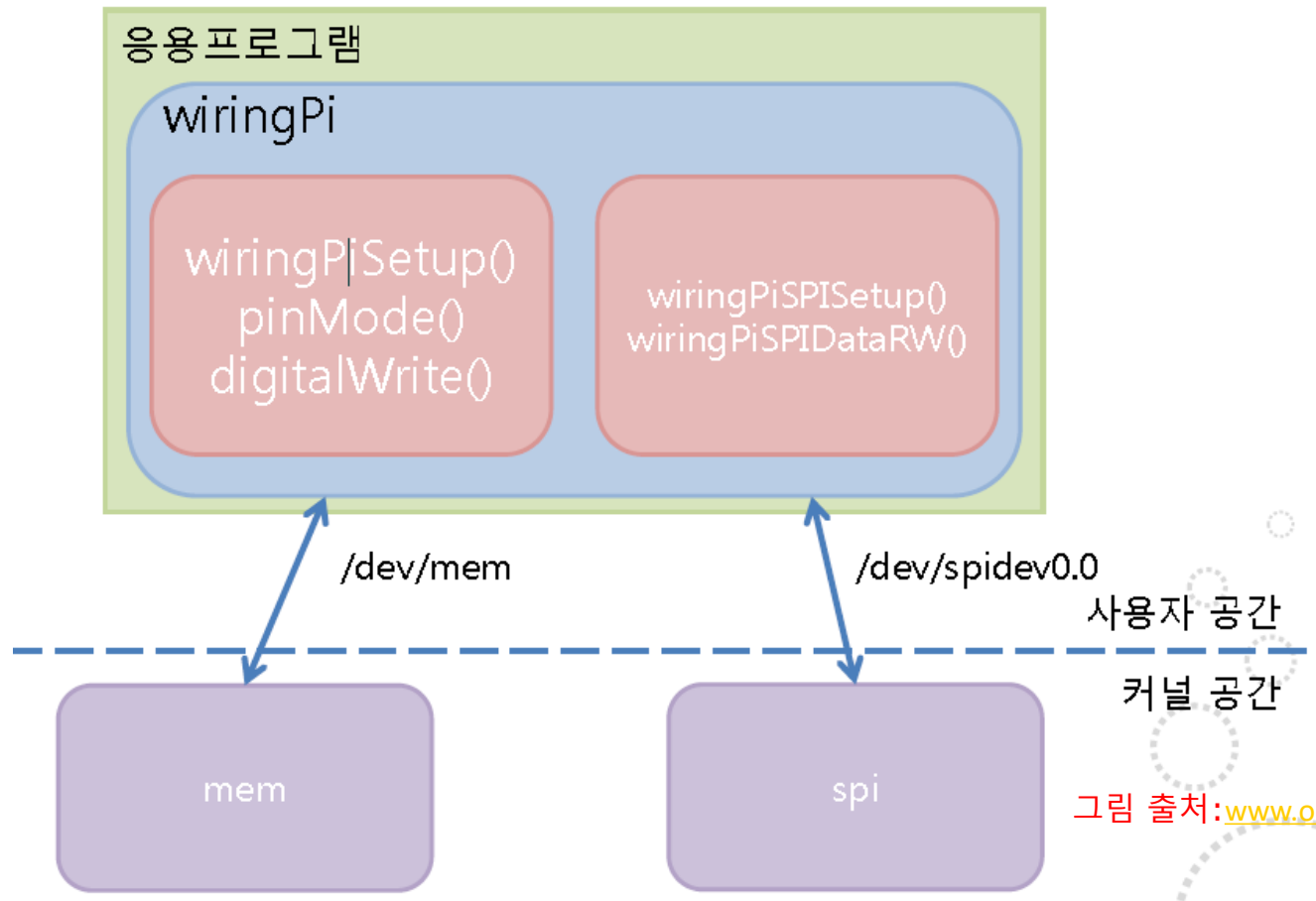
A screenshot of a terminal window with a black background and white text. It displays a series of 20 lines, each showing the text "adc0 Value = " followed by a number. The numbers alternate between 441 and 442, starting with 442 on the first line and ending with 442 on the 20th line.

```
adc0 Value = 442  
adc0 Value = 442  
adc0 Value = 441  
adc0 Value = 441  
adc0 Value = 442  
adc0 Value = 441  
adc0 Value = 441  
adc0 Value = 442  
adc0 Value = 442  
adc0 Value = 441  
adc0 Value = 442  
adc0 Value = 442  
adc0 Value = 441  
adc0 Value = 441  
adc0 Value = 442  
adc0 Value = 441  
adc0 Value = 442  
adc0 Value = 441  
adc0 Value = 442  
adc0 Value = 442
```

[https://www.icbanq.com/seminar2/board_View.aspx?number=269
&category=5&category2=&search_type=&search_text=&page=8](https://www.icbanq.com/seminar2/board_View.aspx?number=269&category=5&category2=&search_type=&search_text=&page=8)

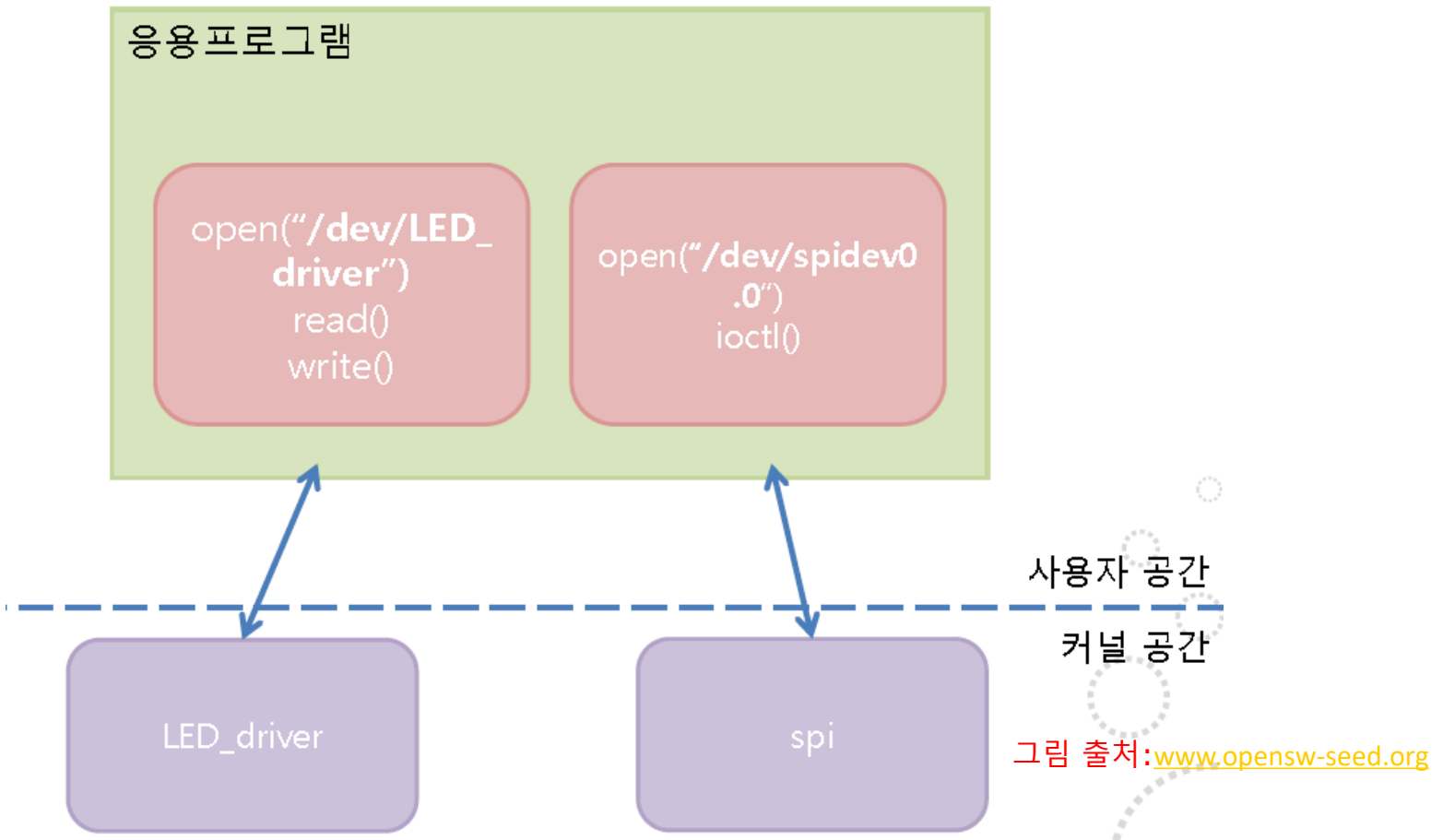
임베디드 SPI 통신 예제

wiringPi 라이브러리를 사용한 방법 대신, BCM라이브러리 시스템을 호출을 통해 spi드라이버에 접근하는 방법



임베디드 SPI 통신 예제

wiringPi 라이브러리를 사용한 방법 대신, BCM라이브러리 시스템 호출을 통해 spi드라이버에 접근하는 방법



임베디드 SPI 통신 예제

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <stdint.h>
#include <linux/spi/spidev.h>
#include <sys/ioctl.h>
#include <linux/types.h>
#include <linux/kdev_t.h>
#define DEV_NEWDEV_MAJOR_NUMBER 220
#define DEV_NEWDEV_NAME "LED_DRIVER"
#define DEV_FILE_NAME "/dev/led_driver"
#define SPI_SPEED 1000000
#define SPI_CHANNEL 0
```

임베디드 SPI 통신 예제

```
static uint32_t spi_speeds [2] ;
static int spi_fds [2] ;
const static uint8_t spi_mode = 0 ;
const static uint8_t spi_bpw = 8 ;
const static uint16_t spi_delay = 0 ;
int pi_spi_data_rw(int channel, unsigned char *data, int len) {
    struct spi_ioc_transfer spi ;
    channel &= 1 ;
    spi.tx_buf = (unsigned long)data ;
    spi.rx_buf = (unsigned long)data ;
    spi.len = len ;
    spi.delay_usecs = spi_delay ;
    spi.speed_hz = spi_speeds [channel] ;
    spi.bits_per_word = spi_bpw ;
    return ioctl (spi_fds [channel], SPI_IOC_MESSAGE(1), &spi) ;
}
```


임베디드 SPI 통신 예제

```
int main(void) {
    int spi_fd; int cs_fd;
    int speed = 1000000;
    int adc_val = 0;
    int adc_ch = 0;
    char write_data;
    unsigned char buff[3];
    mknod(DEV_FILE_NAME, ( S_IRWXU|S_IRWXG|S_IFCHR) ,
    MKDEV(DEV_NEWDEV_MAJOR_NUMBER,0 ));
    cs_fd = open(DEV_FILE_NAME, O_RDWR|O_NONBLOCK);
    if( cs_fd < 0 ) {
        printf("open error\n"); return 0;
    }
}
```

임베디드 SPI 통신 예제

```
spi_fd = open ("/dev/spidev0.0" , O_RDWR);  
if( spi_fd < 0 ) { printf("open error\n");  
return 0; }
```

```
spi_speeds[0] = SPI_SPEED;  
spi_fds[0] = spi_fd;  
ioctl (spi_fd , SPI_IOC_WR_MODE, &spi_mode);  
ioctl (spi_fd , SPI_IOC_WR_BITS_PER_WORD , &spi_bpw);  
ioctl (spi_fd , SPI_IOC_WR_MAX_SPEED_HZ, &speed);
```

임베디드 SPI 통신 예제

```
while(1) {  
    sleep(1);  
    buff[0] = 0x06 | ((adc_channel &0x07)>>2);  
    buff[1] = ((adc_channel &0x07)<<6);  
    buff[2] = 0x00;  
    write_data = 0;  
    write(cs_fd , &write_data , 1);  
    pi_spi_data_rw(SPI_CHANNEL , buff, 3);  
    buff[1] = 0x0f & buff[1];  
    adc_value = (buff[1] << 8) | buff[2];  
    write_data = 1;  
    write(cs_fd , &write_data , 1);  
    printf("adc0 Value = %u\n", adc_value);  
}  
close(spi_fd);  
close(cs_fd);  
return 0;
```

임베디드 SPI 통신 예제

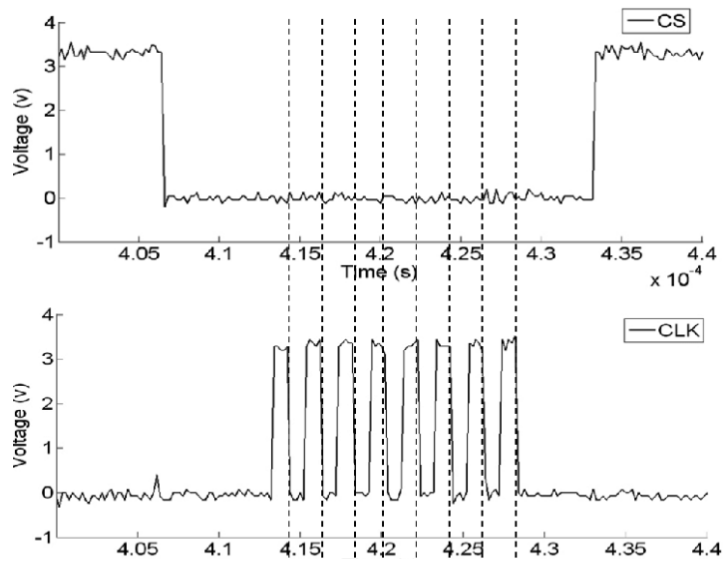
실행

```
# gcc -o app_spi app_spi.c
```

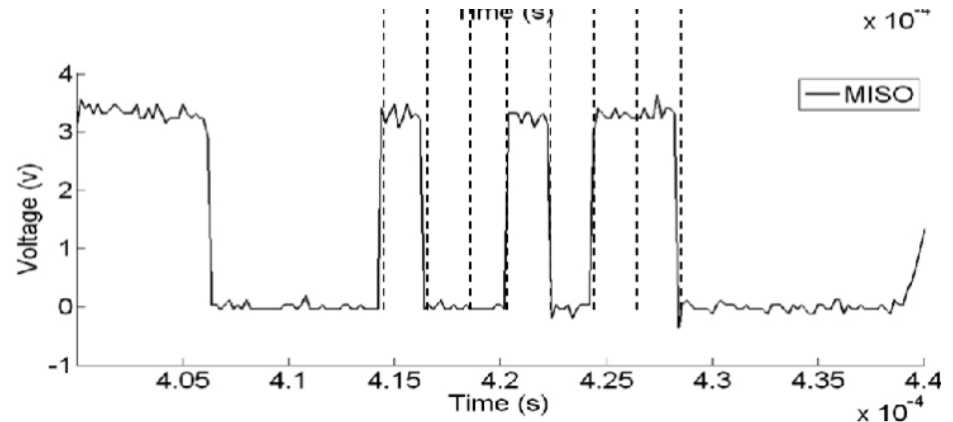
```
adc0 Value = 442  
adc0 Value = 442  
adc0 Value = 441  
adc0 Value = 441  
adc0 Value = 442  
adc0 Value = 441  
adc0 Value = 441  
adc0 Value = 442  
adc0 Value = 442  
adc0 Value = 441  
adc0 Value = 442  
adc0 Value = 442  
adc0 Value = 441  
adc0 Value = 441  
adc0 Value = 442  
adc0 Value = 441  
adc0 Value = 442  
adc0 Value = 441  
adc0 Value = 442  
adc0 Value = 442  
adc0 Value = 442
```

[https://www.icbanq.com/seminar2/board_View.aspx?number=269
&category=5&category2=&search_type=&search_text=&page=8](https://www.icbanq.com/seminar2/board_View.aspx?number=269&category=5&category2=&search_type=&search_text=&page=8)

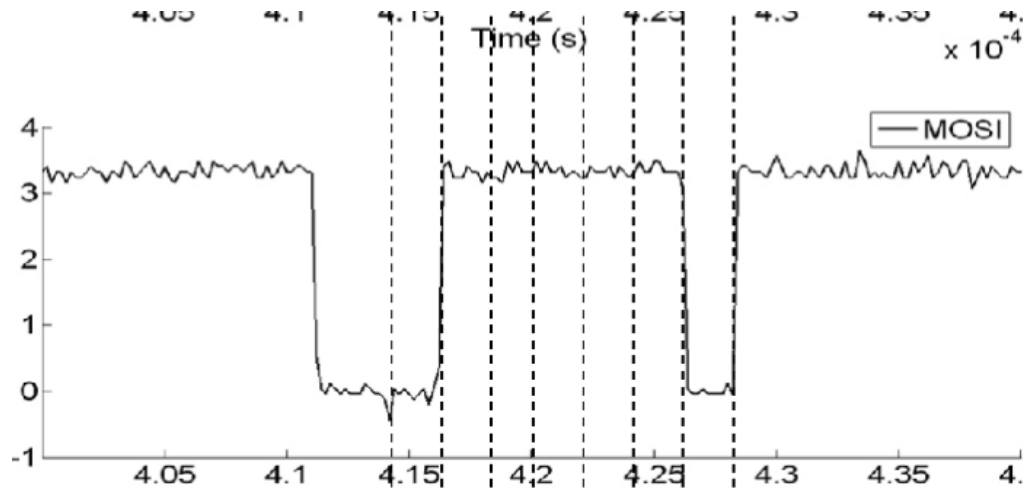
임베디드 SPI 통신 예제



CS & CLK



MISO



MOSI

임베디드 I2C 통신

I2C

Inter Integrated Circuit 버스

마이크로프로세서와 저속 주변장치 사이의 통신을 목적으로 함

Philips 에서 개발한 규격

2개의 line만으로 통신하기 때문에 TWI (Two Wire Interface)라고도 함

I2C는 양방향 오픈 드레인 선인 SCL(Serial Clock)과 SDA(Serial Data)로 이루어져 있음

Master-Slave 형태로 동작함

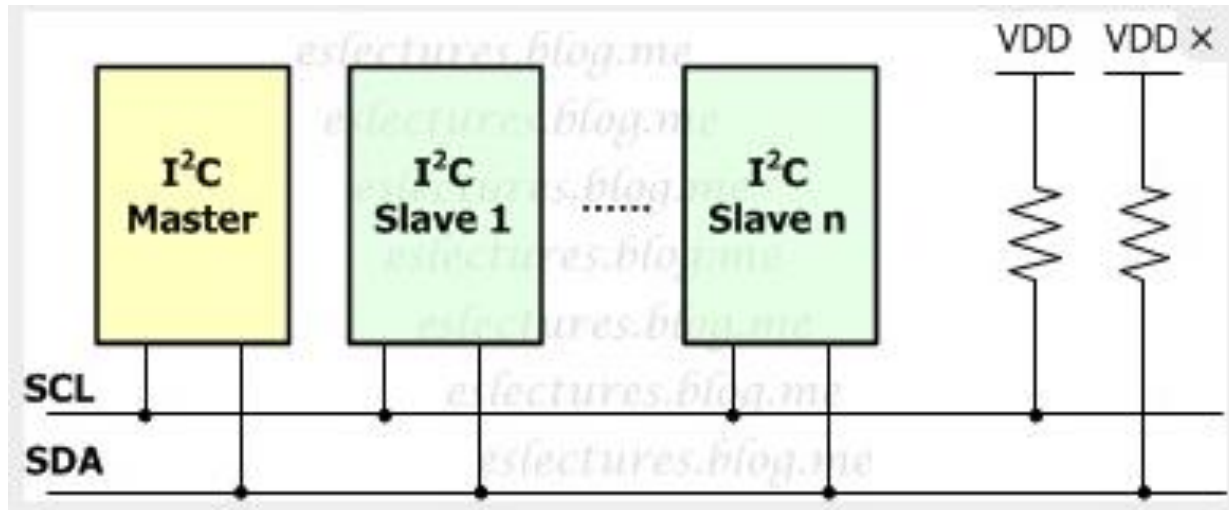
속도면에서는 다른 방식에 비하여 현저히 느리지만, 하드웨어적으로 간단한 구성으로 자주 활용됨

하나의 버스에 많은 수의 노드를 연결할 수 있다는 장점이 있음

임베디드 I2C 통신

SCL : 통신의 동기(synchronization)를 위한 클럭용 line

SDA : 데이터용 line



Master 는 SCL로 동기를 위한 클럭을 출력

Slave는 SCL로 출력되는 클럭에 맞추어 SDA를 통해 데이터를 출력하거나 입력받는다.

<https://eslectures.blog.me/>

임베디드 I2C 통신

SDA는 한 선으로만 데이터를 주고 받기 때문에 I2C 버스는 반이중(half duplex) 통신만 가능

SCL선과 SDA선은 모두 오픈 드레인이므로 두 선에는 각각 풀업 저항을 연결해 주어야 한다.

모든 I2C Master와 Slave 장치들의 SCL, SDA는 서로 연결되고, 모든 장치들이 SCL, SDA를 각각 공유하므로 Master가 Slave를 개별적으로 지정하기 위한 방법이 있어야 한다.

Master는 주소로 원하는 Slave를 지정한다.

주소의 길이는 7비트(bit)이므로 Master는 최대 127개의 슬레이브 장치들과 연결가능

물론, 이 때 각 Slave 장치들의 주소는 모두 달라야 한다.

임베디드 I2C 통신

I2C 통신방식

한 순간에는 오직 하나의 Master와 하나의 Slave 만이 통신할 수 있음.
모든 장치들의 SCL과 SDA는 각각 wired AND로 연결되어 있음.

- ▶ 이때, 어느 한 장치라도 0을 출력하면 해당 신호의 상태는 논리 0이 됨.
만일 어떤 장치가 논리 0을 출력하면 다른 장치가 그 신호의 상태를 논리 1로 만들 방법이 없음
- ▶ 따라서, 통신에 참여하지 않는 장치가 SCL이나 SDA로 0을 출력하면 Master가 정상적으로 통신할 수 없음.
- ▶ 따라서, I2C 버스에 연결되어 있지만 현재 통신에 참여하지 않고 있는 장치들은 모두 자신의 출력을 floating 상태로 유지해야 함.

임베디드 I2C 통신

I2C 통신방식

통신이 진행되지 않는 상황에서 모든 장치의 출력은 floating 상태이므로, SCL과 SDA의 상태는 모두 논리 1임.

이 상황에서 I2C 버스의 사용을 원하는 Master는 SCL과 SDA로 시작 조건을 출력하여 버스 소유권을 주장하고 통신을 시작할 수 있음

두 신호의 상태가 모두 논리 1이 아니라면 현재 다른 Master가 버스 소유권을 가지고 통신을 진행 중에 있다는 것을 뜻함.

따라서, 그 Master가 버스 소유권을 반납할 때 까지는 새로운 통신이 시작될 수 없다.

임베디드 I2C 통신

데이터 안정화 구간

I2C 프로토콜의 시작 조건, 정지 조건, 그리고 데이터 안정구간

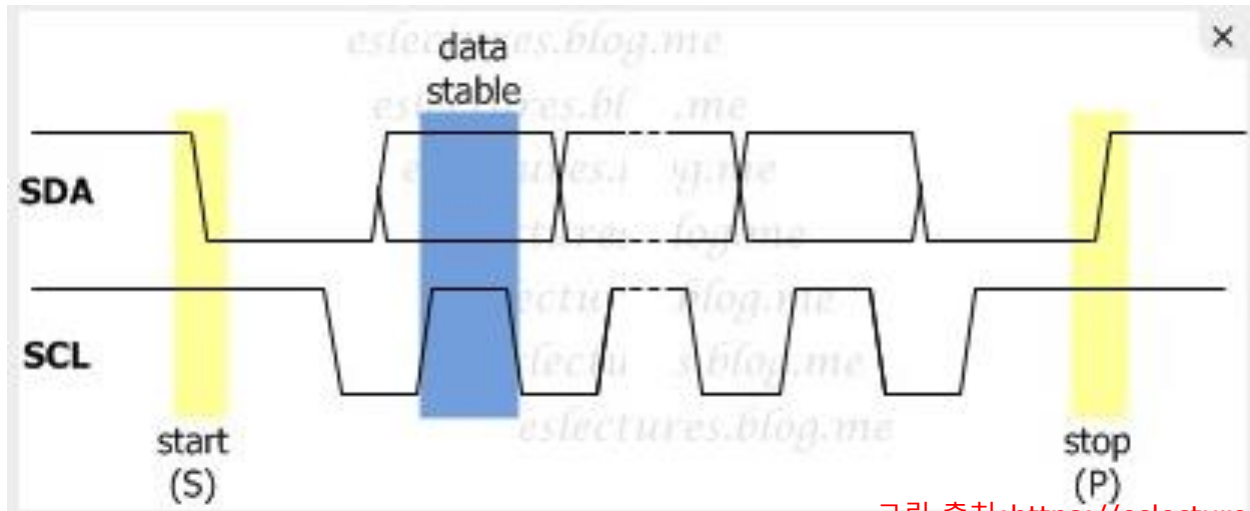


그림 출처: <https://eslectures.blog.me/>

I2C 프로토콜에서 SCL이 0인 구간에서는 SDA의 상태변화가 허용되지만

SCL이 1인 구간에서는 SDA는 안정된 논리 상태를 유지해야 한다.

Master가 Slave로 데이터를 출력할 때(Slave가 데이터를 출력하고 Master가 그 데이터를 읽을 때도 동일) SCL이 0인 구간에서 SDA의 비트전환을 하여 SCL이 1인 구간에서는 SDA의 상태를 그대로 유지

임베디드 I2C 통신

I2C 프로토콜의 시작

Start Condition

- ▶ SCL이 1인 동안에 SDA가 1에서 0으로 바뀌는 상황
- ▶ 통신의 시작을 다른 장치에 알리는 효과

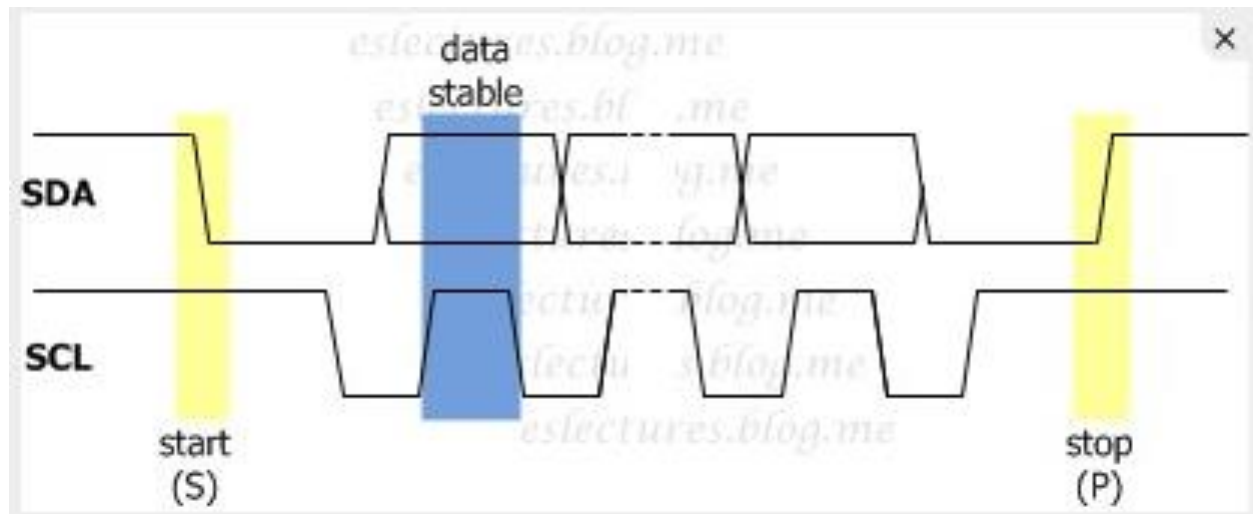


그림 출처: <https://eslectures.blog.me/>

Stop Condition

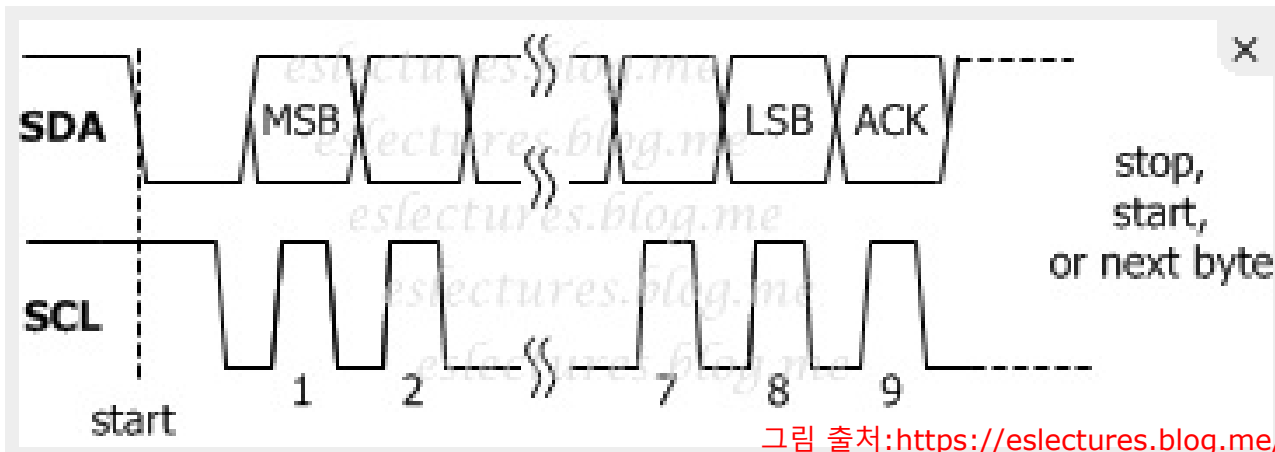
- ▶ SCL이 1인 동안에 SDA가 0에서 1로 바뀌는 상황
- ▶ 버스 소유권 반납을 다른 장치에 알리는 효과

임베디드 I2C 통신

I2C 통신 패킷 형식

ACK를 포함한 9비트(bit)가 I2C 규격에서 통신의 기본 단위임.

Master는 시작조건을 출력하면서 통신의 시작을 알리고, 시작 조건 이후부터는 SCL 상태가 0인 구간에서만 SDA의 논리 값이 바뀐다

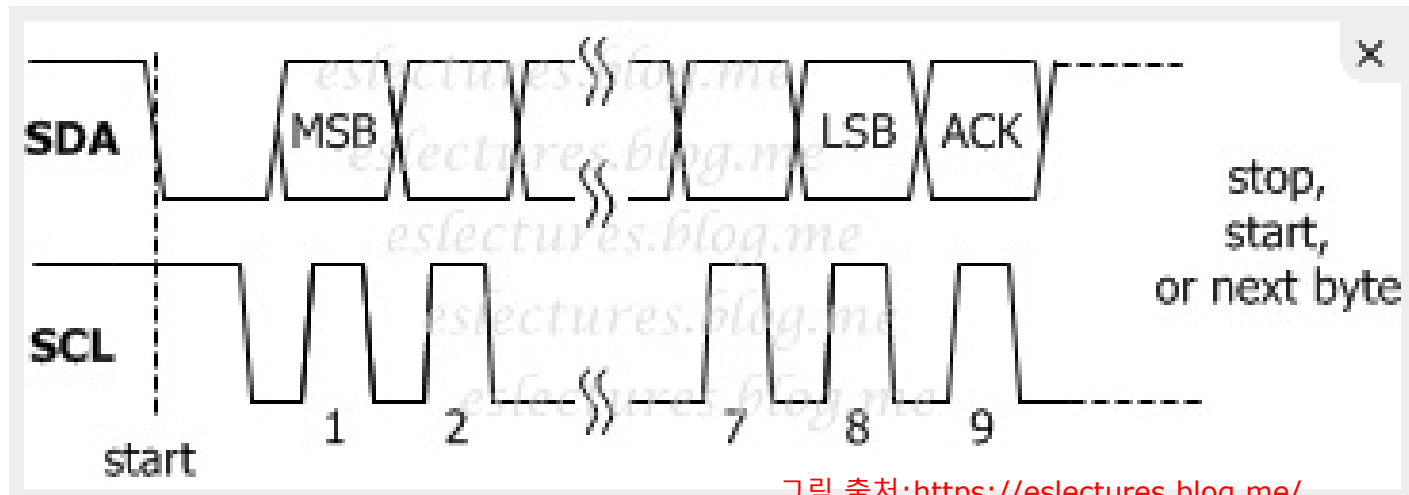


Master가 SCL로 출력하는 클럭에 동기를 맞추어 SDA로는 데이터가 MSB부터 한 비트씩 출력됨

임베디드 I2C 통신

8비트 데이터가 8 클럭 사이클 동안 SDA로 출력되면 그 데이터를 수신한 쪽에서 9번째 클럭에 맞추어 그 8비트 데이터의 수신여부를 확인해주는 ACK를 보냄

- ▶ ACK 비트가 0이면 정상 수신
- ▶ ACK 비트가 1이면 비정상 수신 (값이 0인 ACK와 구분하기 위해서 값이 1인 NACK라 함)



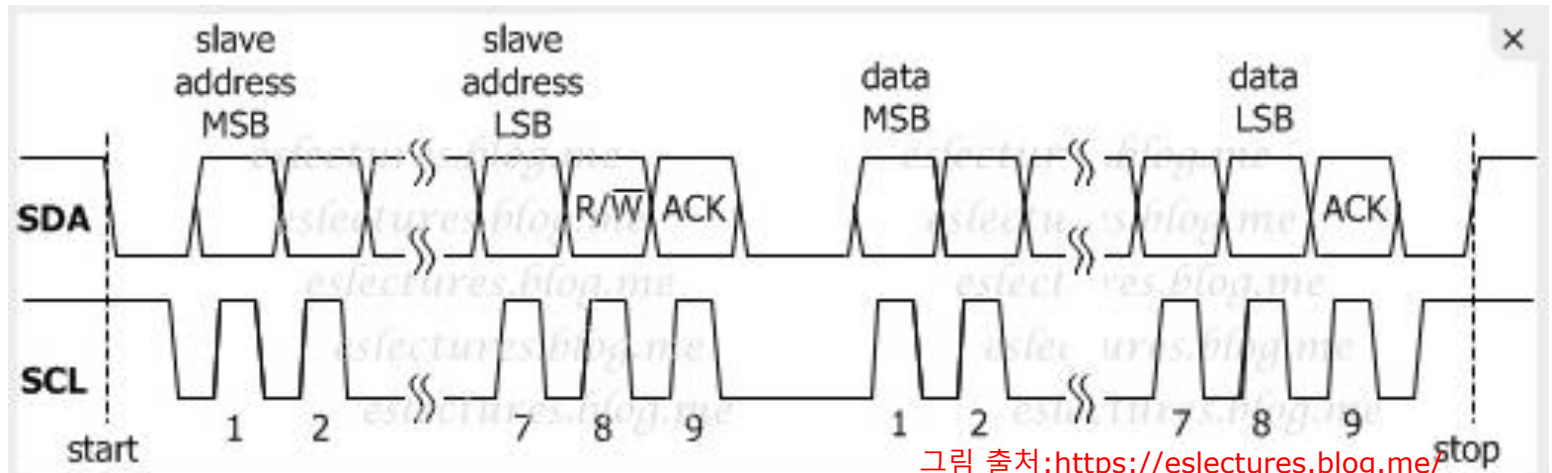
8비트 정보와 이어지는 ACK 비트의 전송이 끝난 다음에는 Master가 정지 조건을 출력할 수도 있고, 시작 조건을 다시 출력할 수도 있고, 다음 바이트 전송이 이어질 수도 있음.

임베디드 I2C 통신

I2C 주소 지정 형식

모든 I2C Slave 장치는 7비트의 고유 주소를 가지며, Master는 이 주소를 사용하여 상대 Slave 장치를 지정한다.

Master는 시작조건에 이어서 자신이 원하는 Slave의 7비트 주소를 출력



버스에 연결된 모든 Slave는 SDA를 계속 감시하면서 Master가 출력한 주소가 자신의 주소와 일치하는 지 검사.

만일 Master가 출력한 주소가 자신의 주소와 같으면 그 Slave는 ACK 비트에 0을 출력하여 Master에게 응답한다.

- ▶ 아무도 일치하지 않으면 아무도 ACK비트로 0을 출력하지 않으므로 1상태를 유지 → NACK 상태

임베디드 I2C 통신

I2C 주소 지정 방식

Slave 주소 7비트 다음에 오는 8번째 비트는 다음 동작이 Master의 읽기인지 쓰기인지 가리킴.

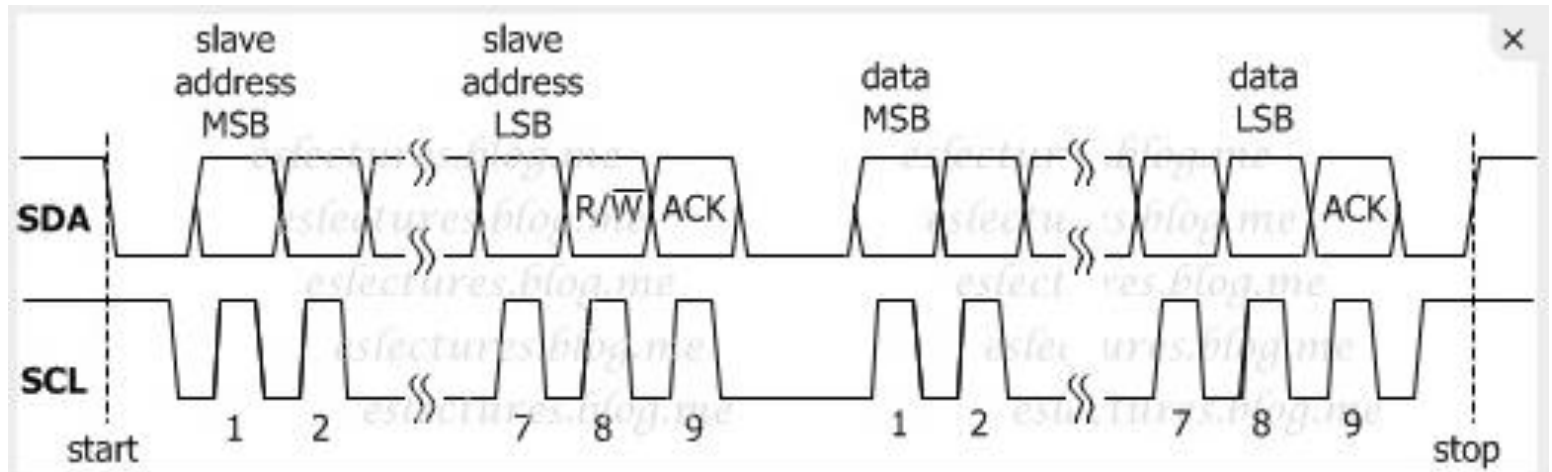


그림 출처: <https://eslectures.blog.me/>

임베디드 I2C 통신

예)

Slave 주소는 7비트이나 상위 7비트로 정렬되고 마지막에 R/W 비트가 추가됨.

DS1037이라는 RTC칩의 I2C 주소는 1101000b로 고정이라고 하자. 즉, DS1036의 I2C 주소는 0x68 (01101000b) 이다.

DS1037에 데이터를 기록해야 할 때, Master는 I2C 버스로 시작 조건을 출력한 다음 1101000⁰을 출력한다.

DS1037의 데이터를 읽고자 하면, Master는 I2C 버스로 시작 조건을 출력한 다음 1101000¹b를 출력한다.

임베디드 I2C 통신 예제

I2C (Inter Integrated Circuit)

I2C는 마이크로프로세서와 저속 주변장치 사이의 통신을 용도로 Philip사에서 개발한 규격

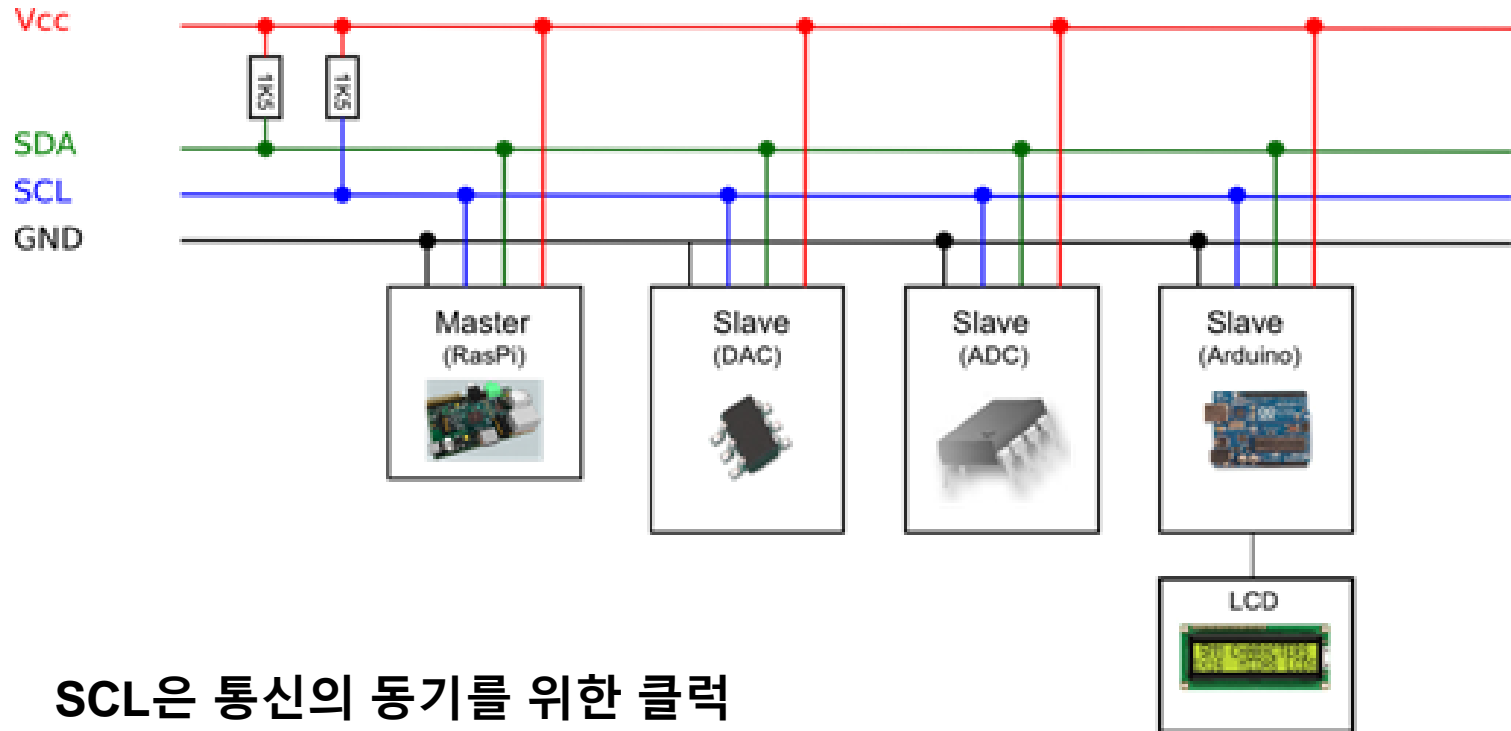
I2C통신은 양방향 선인 SCL(Serial Clock)와 SDA(Serial Data)로 구성. 속도면에서 다른 방식 보다 느리지만 하드웨어적으로 간단한 구성과 하나의 버스에 많은 수의 노드를 연결.

임베디드 I2C 통신 예제

wiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	wiringPi Pin
—	—	3.3v	1 2	5v	—	—
8	R1:0/R2:2	SDA0	3 4	5v	—	—
9	R1:1/R2:3	SCL0	5 6	0v	—	—
7	4	GPIO7	7 8	TxD	14	15
—	—	0v	9 10	RxD	15	16
0	17	GPIO0	11 12	GPIO1	18	1
2	R1:21/R2:27	GPIO2	13 14	0v	—	—
3	22	GPIO3	15 16	GPIO4	23	4
—	—	3.3v	17 18	GPIO5	24	5
12	10	MOSI	19 20	0v	—	—
13	9	MISO	21 22	GPIO6	25	6
14	11	SCLK	23 24	CE0	8	10
—	—	0v	25 26	CE1	7	11
wiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	wiringPi Pin

그림 출처: <http://acb-luddite-technology.blogspot.com/2015/10/raspberry-pi-teletype-project.html>

임베디드 I2C 통신 예제



SCL은 통신의 동기를 위한 클럭

SDA는 데이터 통신을 위한 라인. 그림 출처: <https://github.com/gism/Arduino-Workshop/wiki/OLED>

Master는 SCL로 동기를 위한 클럭을 출력, Slave는 SCL로 출력되는 클럭에 맞추어 SDA를 통해 데이터를 출력하거나 입력

임베디드 I2C 통신 예제

I2C데이터 전송

I2C는 시리얼 전송 방식을 사용하기 때문에 데이터의 전달은 기본적으로 비트 정보를 전달

I2C에서는 데이터를 전달하기 위해 아래 형식을 따름

1. Master가 slave에 전송을 시작한다는 표현 -Start(1bit)
2. 데이터 목적지 주소 - Address (7bit)
3. 전송 목적 - R/'W (읽기 또는 쓰기) (1bit)
4. 전송데이터 - Data (8bit)
5. slave가 정상적으로 데이터를 수신했다는 응답 - Ack (1bit)
6. 전송 종료 - Stop (1bit)

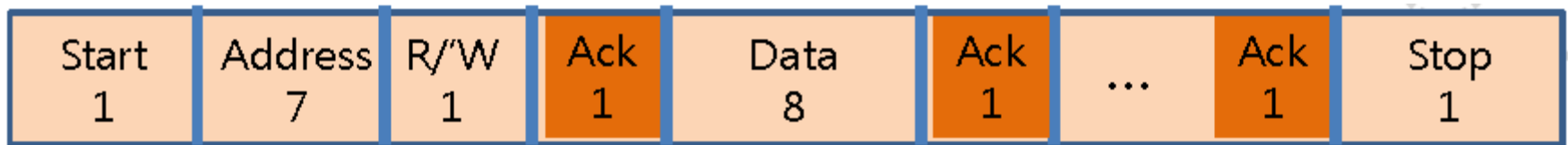
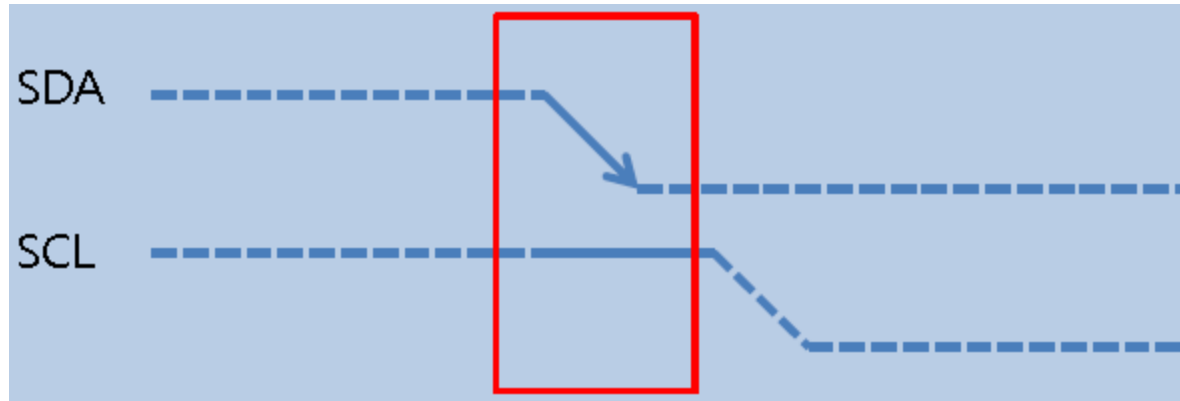


그림 출처: www.opensw-seed.org

임베디드 I2C 통신 예제

Start표현 -Start는 Master가 slave에 전송 시작을 알리기 위한 것



Stop표현 -Stop은 전송 종료를 알리기 위한 것

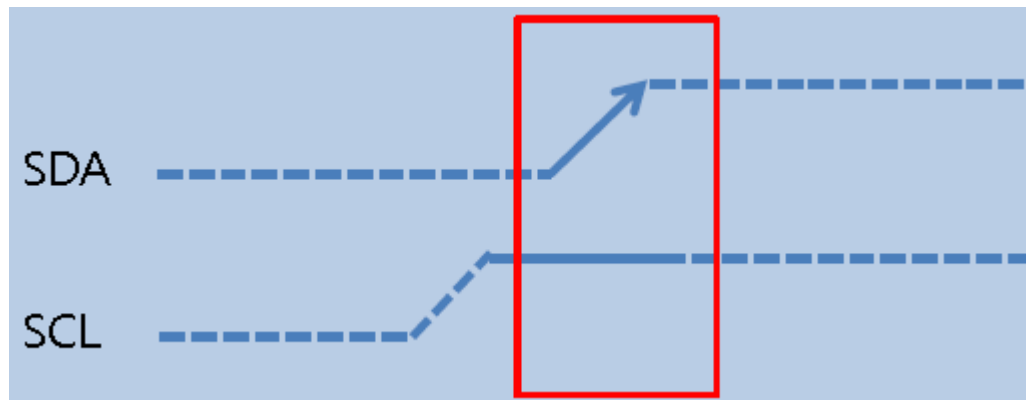
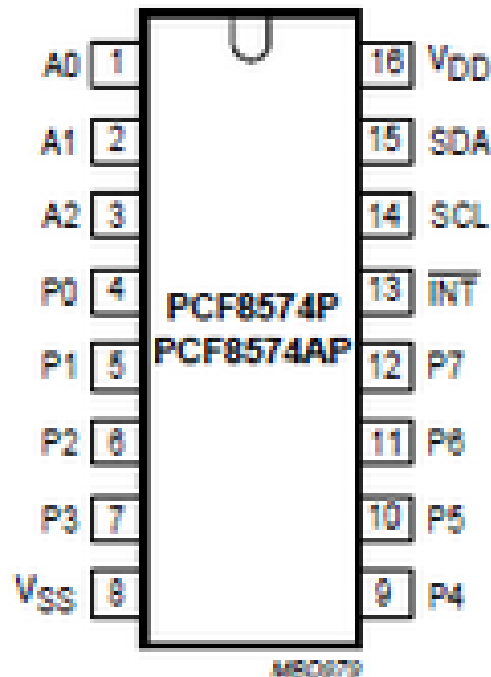


그림 출처: www.opensw-seed.org

임베디드 I2C 통신 예제

I2C에 대해 알아보았으니 라즈베리 파이에서 제어
I2C 칩인 PCF8574AP를 사용하여 제어



SYMBOL	PIN	DESCRIPTION
A0	1	address input 0
A1	2	address input 1
A2	3	address input 2
P0	4	quasi-bidirectional I/O 0
P1	5	quasi-bidirectional I/O 1
P2	6	quasi-bidirectional I/O 2
P3	7	quasi-bidirectional I/O 3
Vss	8	supply ground
P4	9	quasi-bidirectional I/O 4
P5	10	quasi-bidirectional I/O 5
P6	11	quasi-bidirectional I/O 6
P7	12	quasi-bidirectional I/O 7
INT	13	interrupt output (active LOW)
SCL	14	serial clock line
SDA	15	serial data line
VDD	16	supply voltage

그림 출처: www.opensw-seed.org

임베디드 I2C 통신 예제

PCF8574AP

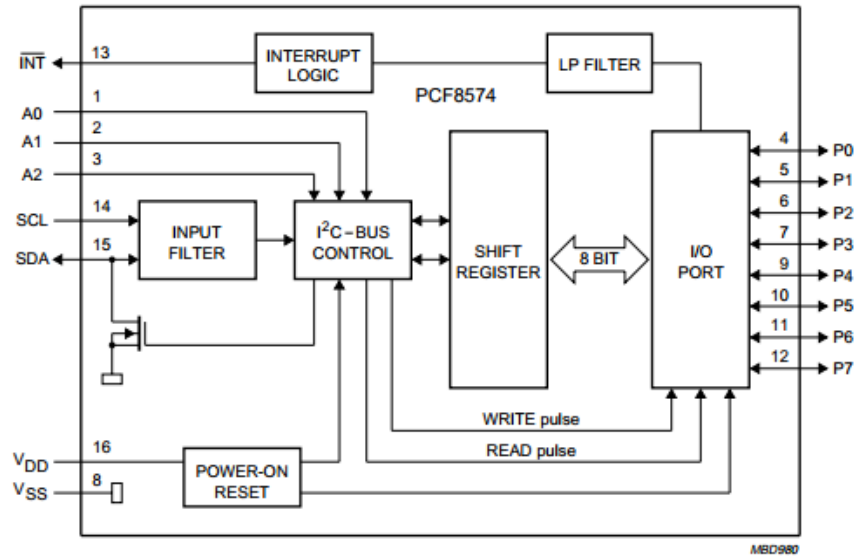


Fig.1 Block diagram (pin numbers apply to DIP16 and SO16 packages).

그림 출처: www.opensw-seed.org

임베디드 I2C 통신 예제

PCF8574AP

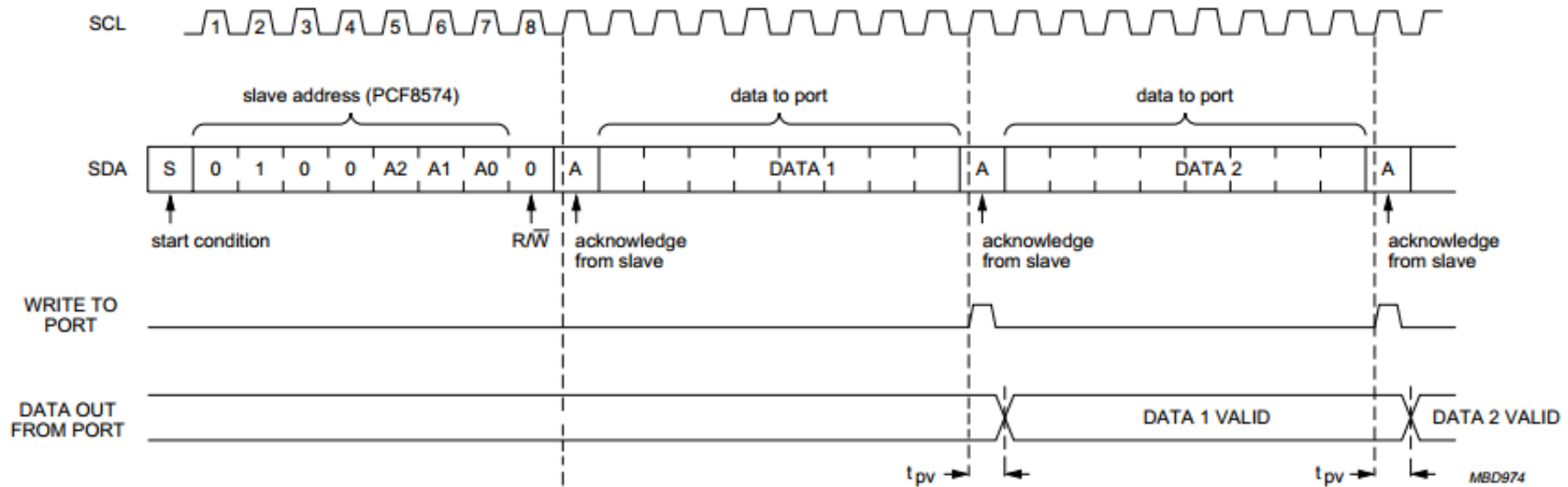


그림 출처: www.opensw-seed.org

Write mode (output)

임베디드 I2C 통신 예제

PCF8574AP

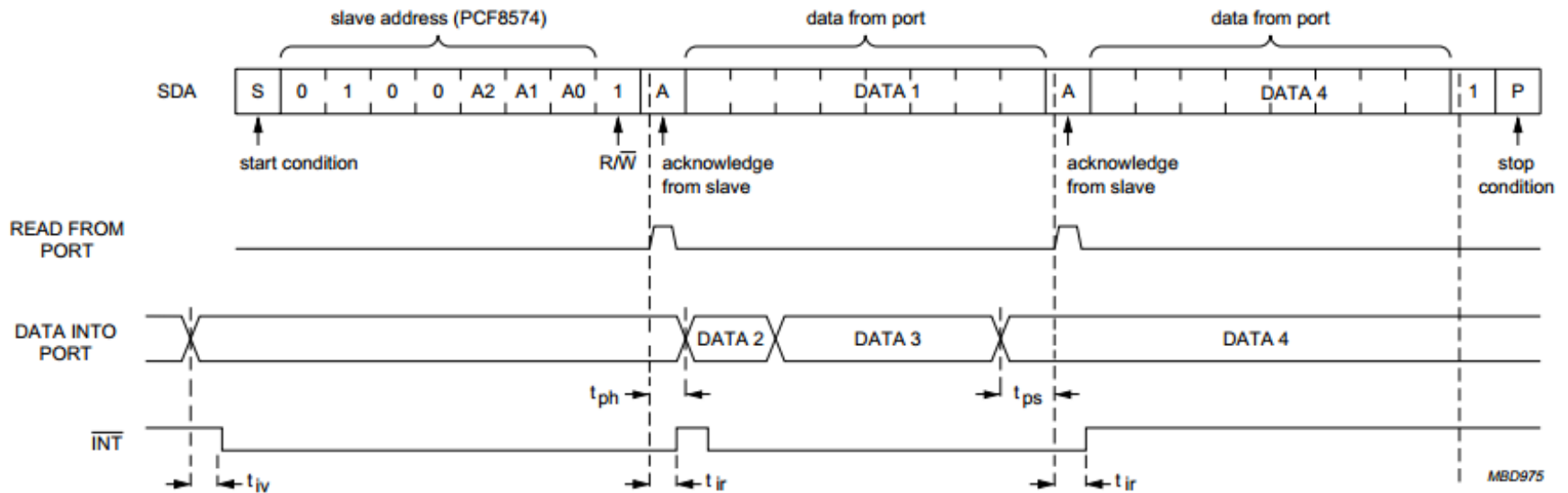


그림 출처: www.opensw-seed.org

Read mode (input)

임베디드 I2C 통신 예제

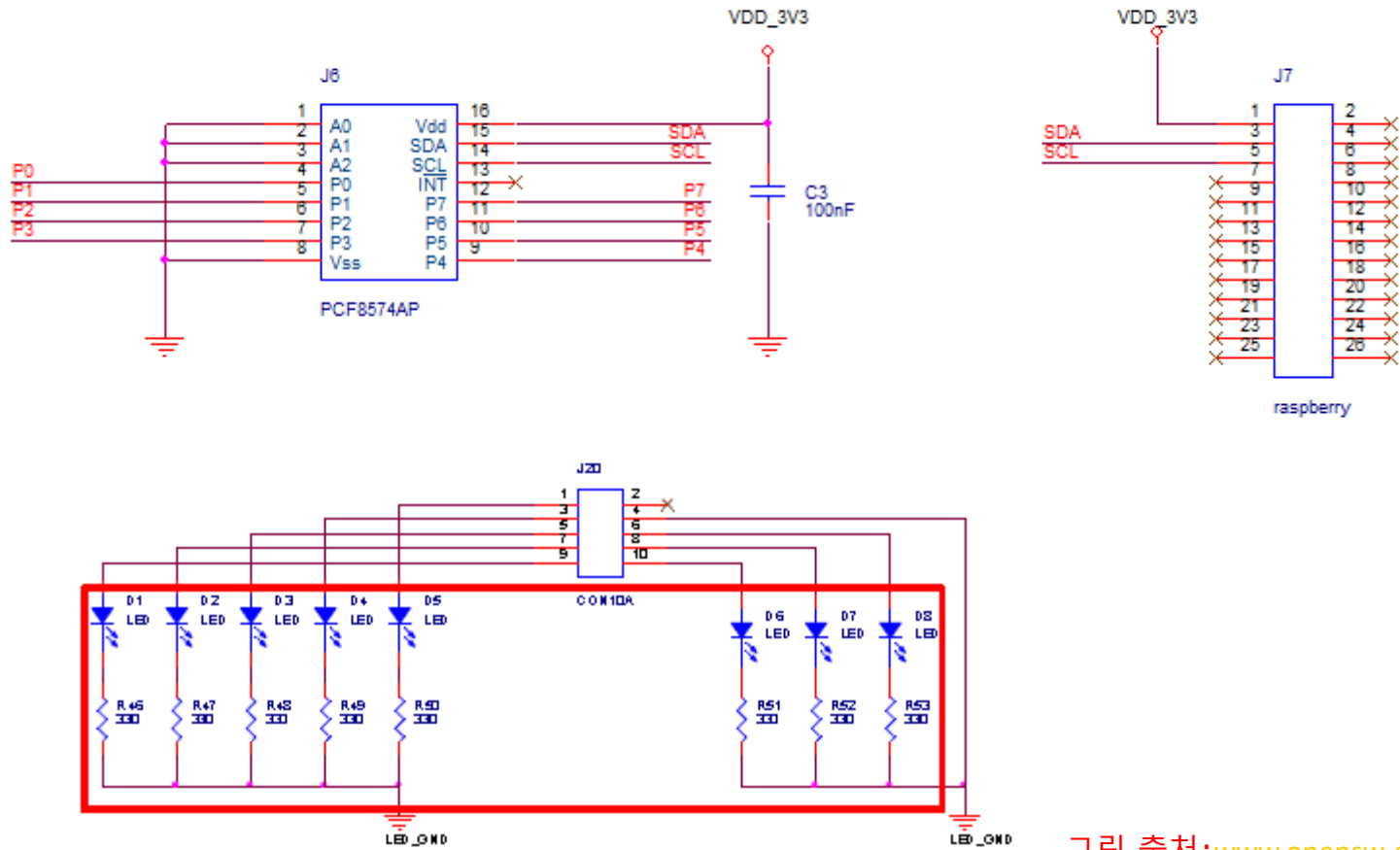


그림 출처: www.opensw-seed.org

임베디드 I2C 통신 예제

```
#include <stdio.h>
```

```
#include <unistd.h>
```

```
#include <wiringPi.h>
```

```
#include <pcf8574.h>
```

```
#define EXTEND_BASE 100
```

```
int main (void) {
```

```
    wiringPiSetup( );
```

```
    pcf8574Setup( EXTEND_BASE, 0x38 );
```

```
    int i;
```

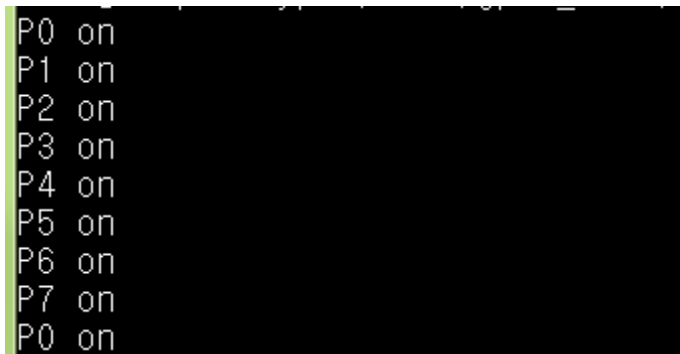
임베디드 I2C 통신 예제

```
for ( i = 0 ; i < 8 ; i++ ) {  
    ▶ pinMode( EXTEND_BASE + i, OUTPUT );  
}  
for (;;) {  
    ▶ for( i = 0 ; i < 8; i++ ) {  
        – digitalWrite ( EXTEND_BASE + i, HIGH);  
        – printf(“P%d on\n”, i);  
        – sleep(1);  
        – digitalWrite ( EXTEND_BASE + i, LOW);  
    ▶ }  
}  
return 0 ;  
}
```

실행

```
# gcc -o wiringPi_i2c wiringPi_i2c.c -lwiringPi
```

```
# ./wiringPi_i2c
```



```
P0 on  
P1 on  
P2 on  
P3 on  
P4 on  
P5 on  
P6 on  
P7 on  
P0 on
```

그림 출처: www.opensw-seed.org

임베디드 디바이스 드라이버 제어

라즈베리파이를 위한 나만의 디바이스 드라이버(device driver)를 만들어봅시다

임베디드 디바이스 드라이버 제어

```
#include <linux/module.h>
#include <linux/version.h>
#include <linux/kernel.h>
static int __init ofd_init(void) /* Constructor */
{
    printk(KERN_INFO "rako: driver registered");
    return 0;
}
static void __exit ofd_exit(void) /* Destructor */
{
    printk(KERN_INFO "rako: driver unregistered");
}
```

파일 이름 : rako.c

임베디드 디바이스 드라이버 제어

```
module_init(ofd_init);  
module_exit(ofd_exit);  
MODULE_LICENSE("GPL");  
MODULE_AUTHOR("Raspberry Pi ");  
MODULE_DESCRIPTION("Device Driver Example");
```

파일 이름 : rako.c

임베디드 디바이스 드라이버 제어

obj-m :=rako.o

KDIR :=/home/pi/raspberry/linux

PWD :=\$(shell pwd)

ARCH =ARCH=arm

CROSS_COMPILE =CROSS_COMPILE=arm-linux-gnueabi-

MAKEARCH = \$(MAKE) \$(ARCH) \$(CROSS_COMPILE)

default:

\$(MAKEARCH) -C \$(KDIR) SUBDIRS=\$(PWD) modules

clean:

\$(MAKEARCH) -C \$(KDIR) SUBDIRS=\$(PWD) clean

파일 이름 : Makefile

임베디드 디바이스 드라이버 제어

컴파일

make

컴파일 된 것 지우기

make clean

컴파일 성공시 다음 파일 존재

확장자 ko 파일

Mod.o 파일

.o 파일

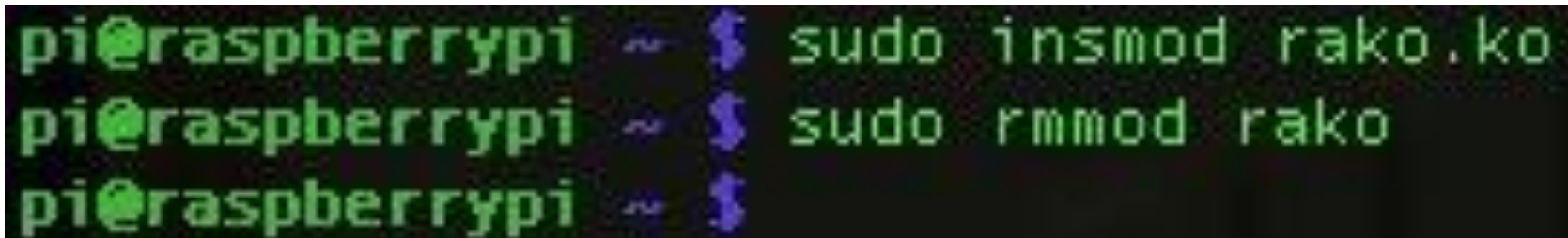
```
erry/rako_driver$ ls
rako.c  rako.ko  rako.mod.c  rako.mod.o  rako.o
erry/rako_driver$
```

임베디드 디바이스 드라이버 제어

컴파일 완료된 rako.ko 파일을 사용하는 방법

`sudo insmod rako.ko` #모듈올리기

`sudo rmmod rako` #모듈해제



```
pi@raspberrypi ~ $ sudo insmod rako.ko
pi@raspberrypi ~ $ sudo rmmod rako
pi@raspberrypi ~ $
```

디바이스 드라이버를 load하였다가 unload하였을 때 출력된 메시지 보기

`sudo cat /proc/kmsg`

임베디드 디바이스 드라이버 제어

```
pi@raspberrypi ~ $ sudo cat /proc/kmsg
<6>[ 1730.583518] rako: driver registered
<6>[ 1827.078967] rako: driver unregisteredrako: driver registered
<6>[ 1870.225730] rako: driver unregistered
```