

가변 저항: Analog-to-digital 변환(2)

김 동 훈

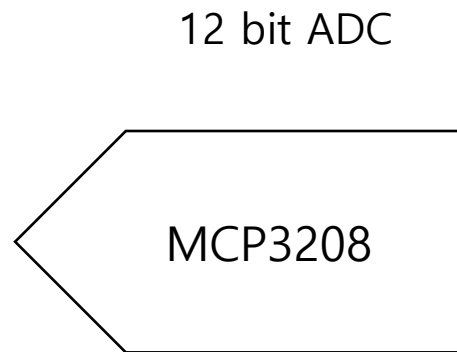
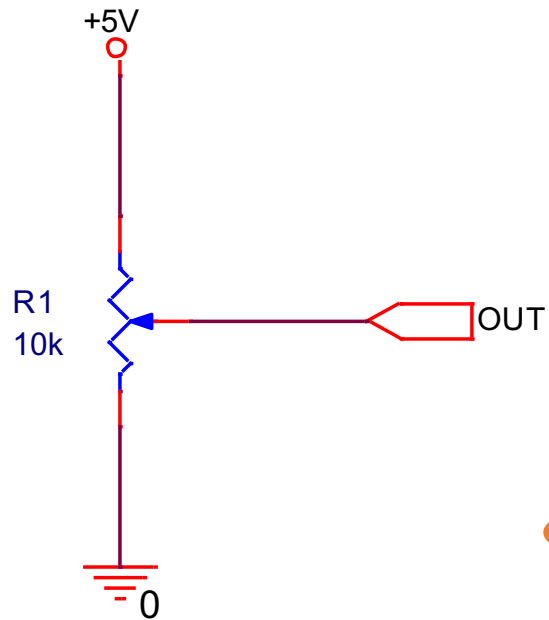
시작

• 강의 소개

- 이번 강의에서는 지난번 강의에 이어 ADC(analog-to-digital) 변환에 대한 내용을 학습합니다.
- 특히, 지난 강의와 연결하여 ADC의 인터페이스로 많이 사용되는 SPI 인터페이스를 학습하고 실습으로 ADC를 사용하여 저항 값의 변화에 따라 변화하는 전압 값을 읽어 들이는 실습을 진행 합니다.
- 실습 프로그램은 코드 리딩을 통해 프로그램의 흐름과 동작을 이해 할 수 있습니다.
- 실습 프로그램은 다음과 같이 진행하기 바랍니다.
 - 먼저 실습에서 주어진 문제를 읽고 이해하시기 바랍니다.
 - 실습코드를 공개 했으니 코드 리딩을 통해 프로그램의 흐름을 파악하시기 바랍니다.
 - 실습 코드의 흐름이 파악되면 그 동작을 이해 할 수 있습니다.
 - 이러한 과정은 프로그램 개발과정의 일부분이니 익숙해 지시는 것이 필요합니다.
 - 실습이 가능해지면 실습을 통해서 동작을 확인할 예정이니 큰 부담 갖지 말고 진행하시기 바랍니다.
 - 코드 리딩에 필요한 주석은 프로그램에 달려 있으니 꼼꼼히 확인하시기 바랍니다.

시작

• 가변 저항 및 ADC(Analog-to-digital converting) 구성



2 12 bit ADC를 통해 디지털화하고

1 가변 저항으로 변경된 아날로그 전압 값을

SPI
(Serial peripheral interface)

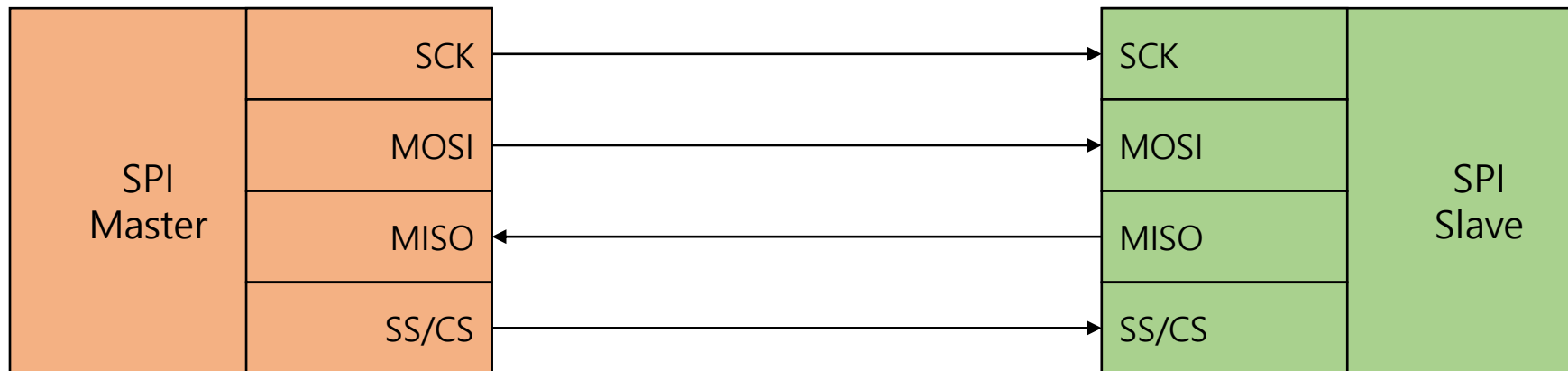


3 SPI 통신을 통해 raspberry pi로 읽어 온다

ADC(Analog-to-digital converting)

- **SPI**

- Serial peripheral interface 약어
- 1980년 중반 Motorola에 의해 개발
- On-board chip들 또는 근거리 장치 간의 동기화 된 시리얼 통신 인터페이스
- 4 wire serial bus
 - SCK: Serial Clock
 - MOSI: Master Output Slave Input
 - MISO: Master Input Slave Output
 - SS(CS): Slave Selection/Chip Selection



ADC(Analog-to-digital converting)

• SPI

- 4 wire serial bus

- SCK

- Serial Clock
 - SPI master가 serial interface의 동기를 위하여 생성하는 clock 신호
 - Serial data input/output은 SCK의 rising/falling edge에 sampling

- MOSI

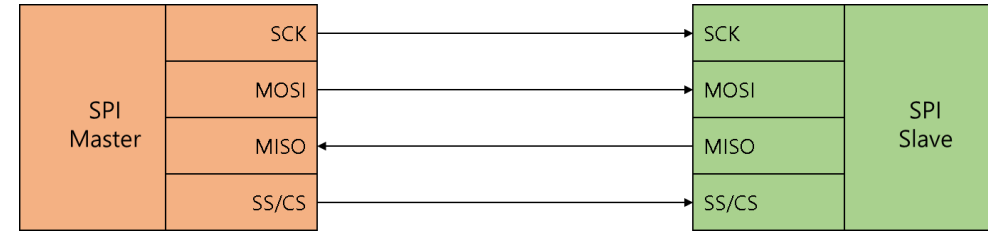
- Master Output Slave Input
 - SPI master serial 데이터 출력 ➔ SPI slave 데이터 입력
 - SPI master가 SCK에 동기 되어 데이터 출력
 - SPI slave가 SCK에 동기 되어 데이터 입력

- MISO: Master Input Slave Output

- Master Input Slave Output
 - SPI slave serial 데이터 출력 ➔ SPI master 데이터 입력
 - SS/CS에 의해 선택된 SPI slave가 SCLK에 동기 되어 데이터 출력
 - SPI master가 SCK에 동기 되어 데이터 입력

- SS/CS

- Slave Selection/Chip Selection
 - SPI master가 SPI slave를 선택하기 위한 선택 신호



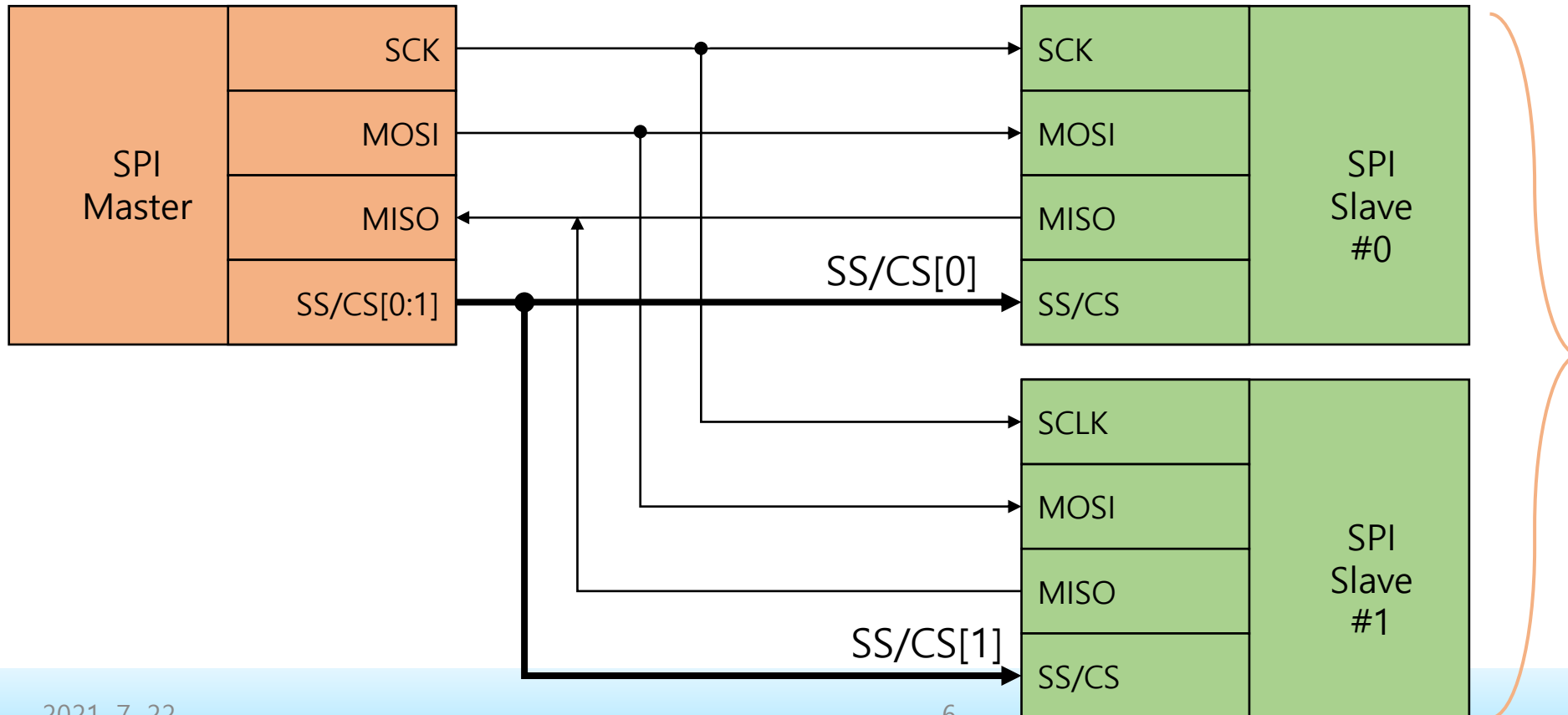
MOSI와 MISO가 분리되어 있어
양방향 통신 가능

ADC(Analog-to-digital converting)

- SPI

- Multi-drop

- 여러 개의 SPI slave가 동일한 SPI를 공유
 - SPI master의 SS/CS 신호에 의해 SPI slave 선택



Multi-drop
: CS에 의해 slave SPI 선택

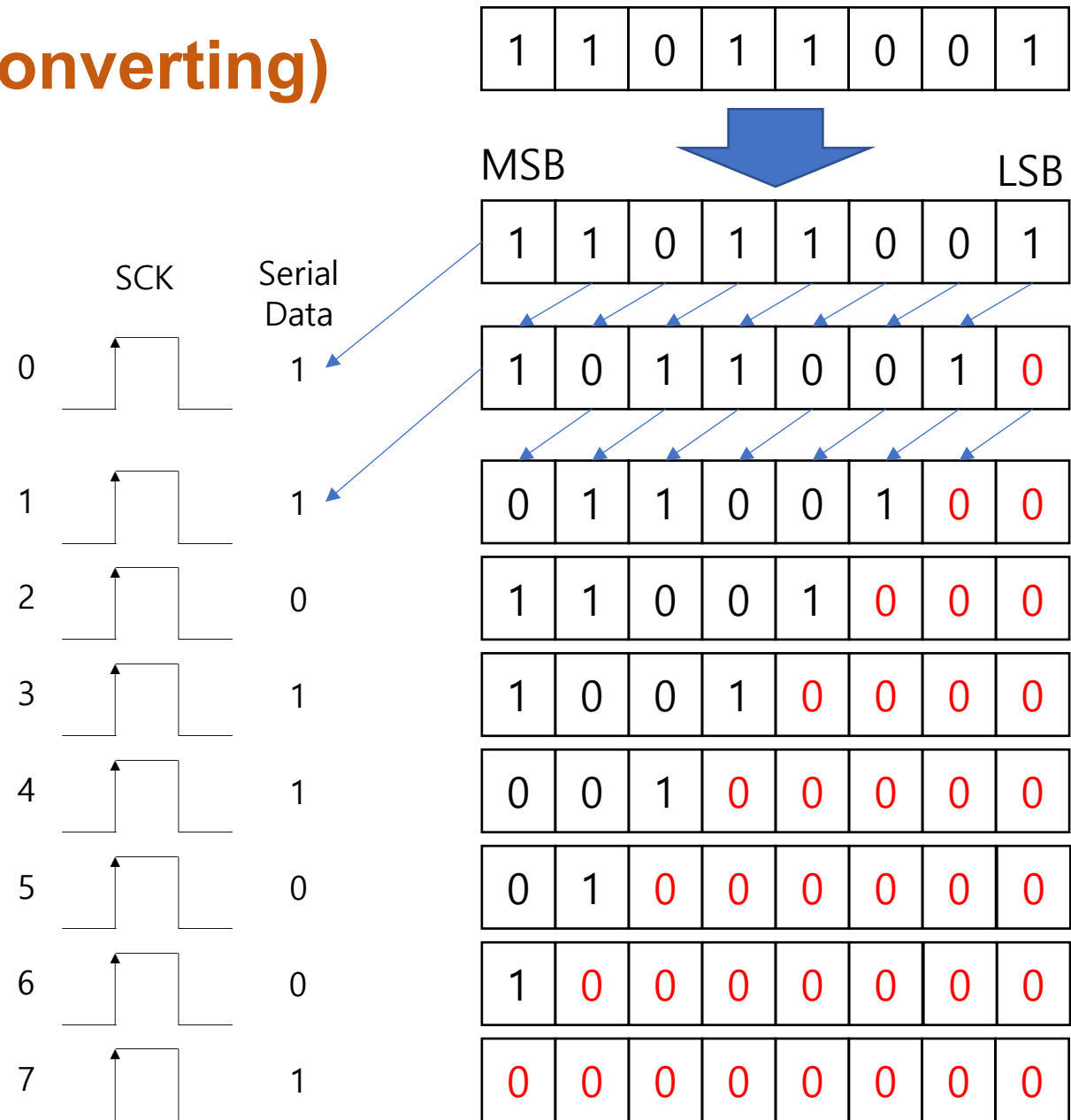
ADC(Analog-to-digital converting)

- **Shift register**

- Clock에 따라 데이터가 한 bit씩 자리 이동
- Output 동작
 - SCK clock에 동기 되어 한 bit씩 serial data로 출력
 - 최상위 bit(MSB)를 serial data로 출력
 - 각 bit 데이터는 한 bit씩 좌로 이동(left shift)
 - 최하위 bit(LSB) 0으로 채워 짐
- Input 동작
 - SCK clock에 동기 되어 serial data를 sampling하여 입력
 - 각 bit 데이터는 한 bit씩 좌로 이동
 - 최하위 bit(LSB)가 clock에 의해 sampling되어 입력
 - 최상위 bit(MSB)는 버림

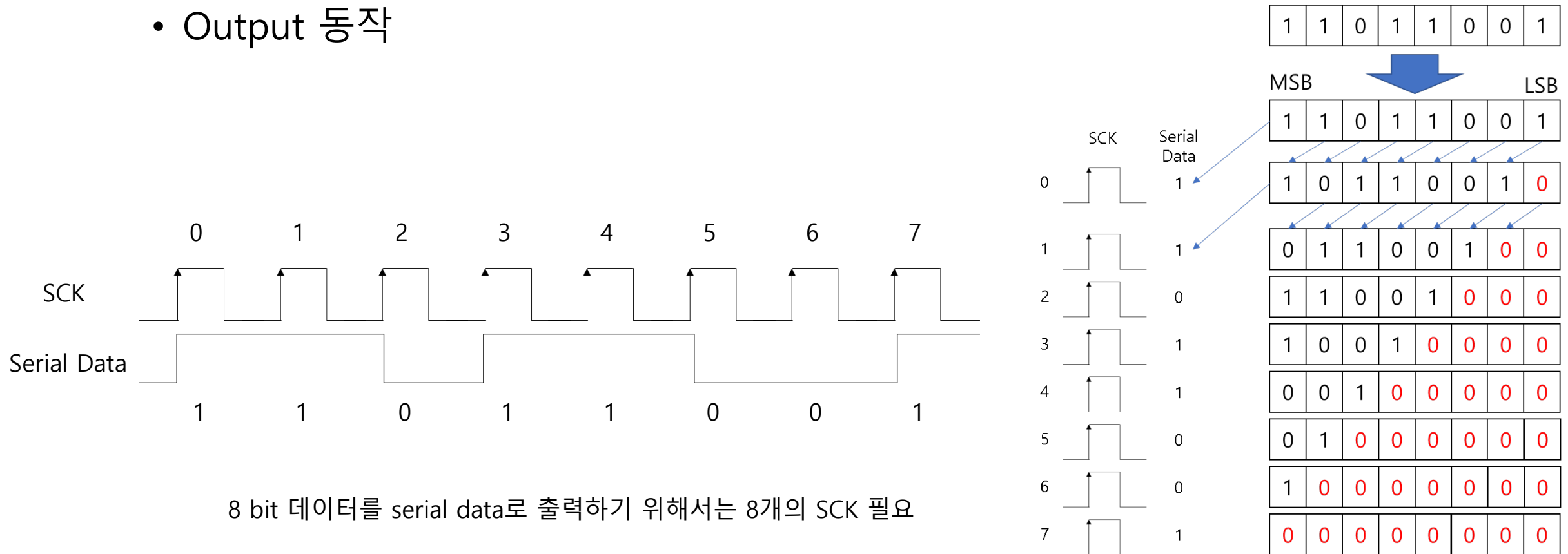
ADC(Analog-to-digital converting)

- Shift register
 - Output 동작



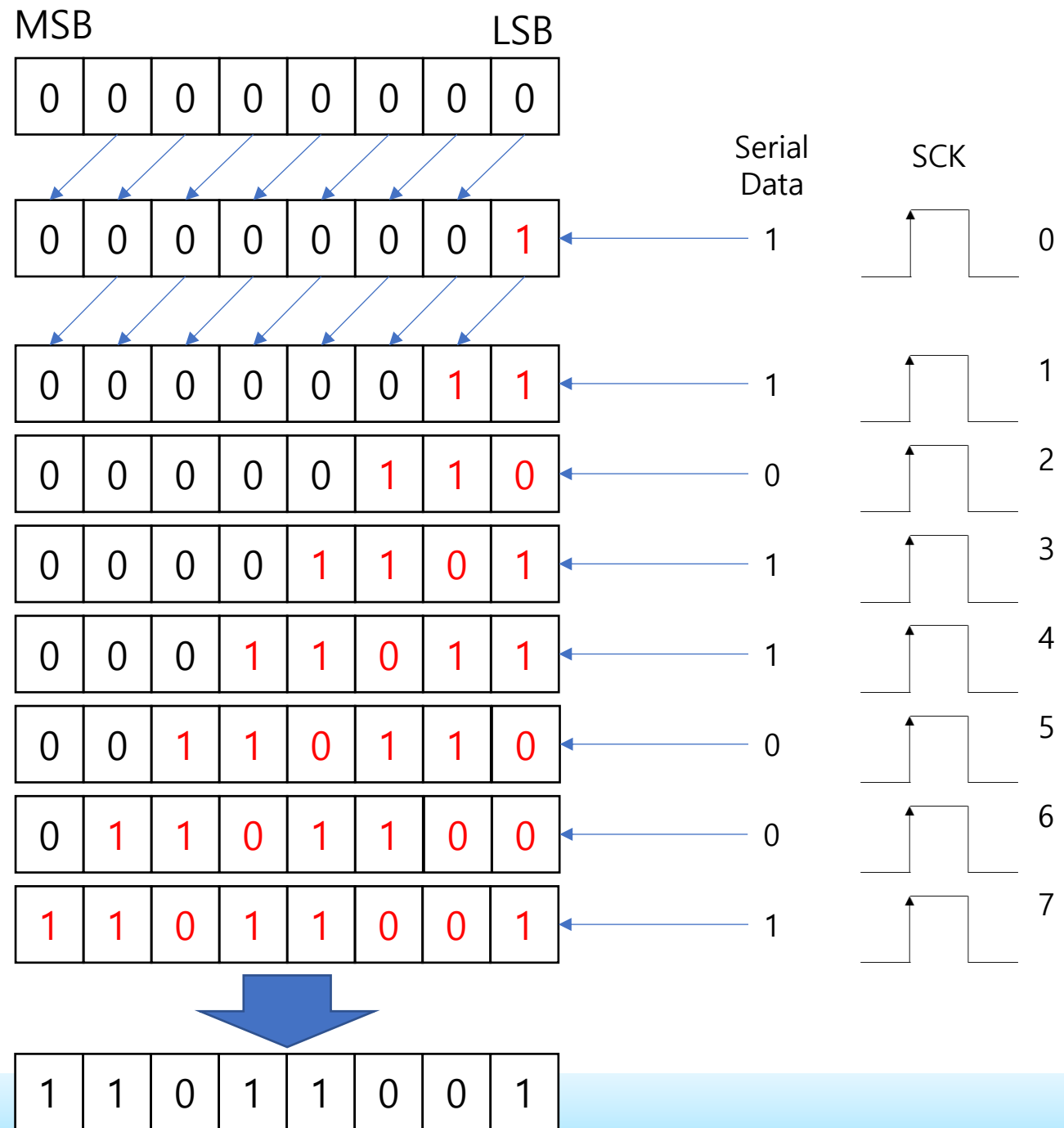
ADC(Analog-to-digital converting)

- Shift register
 - Output 동작



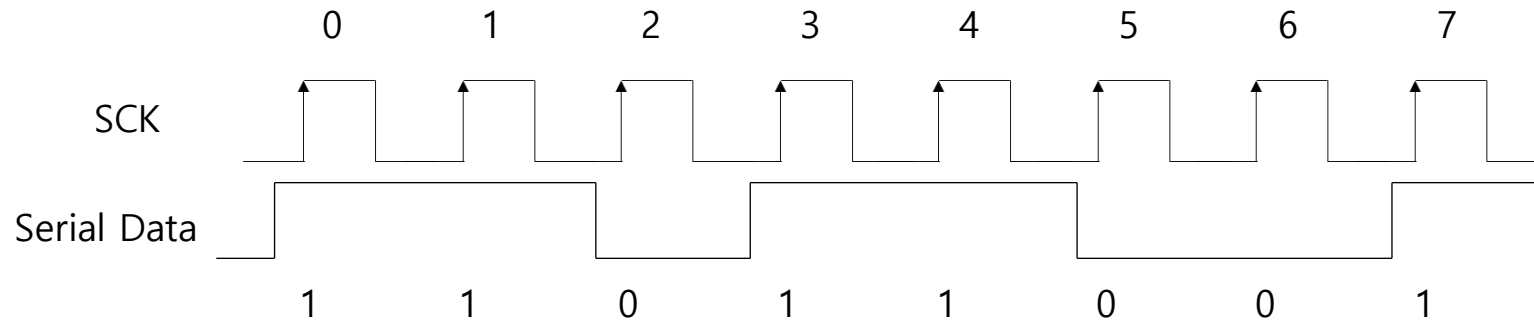
ADC

- Shift register
 - Input 동작

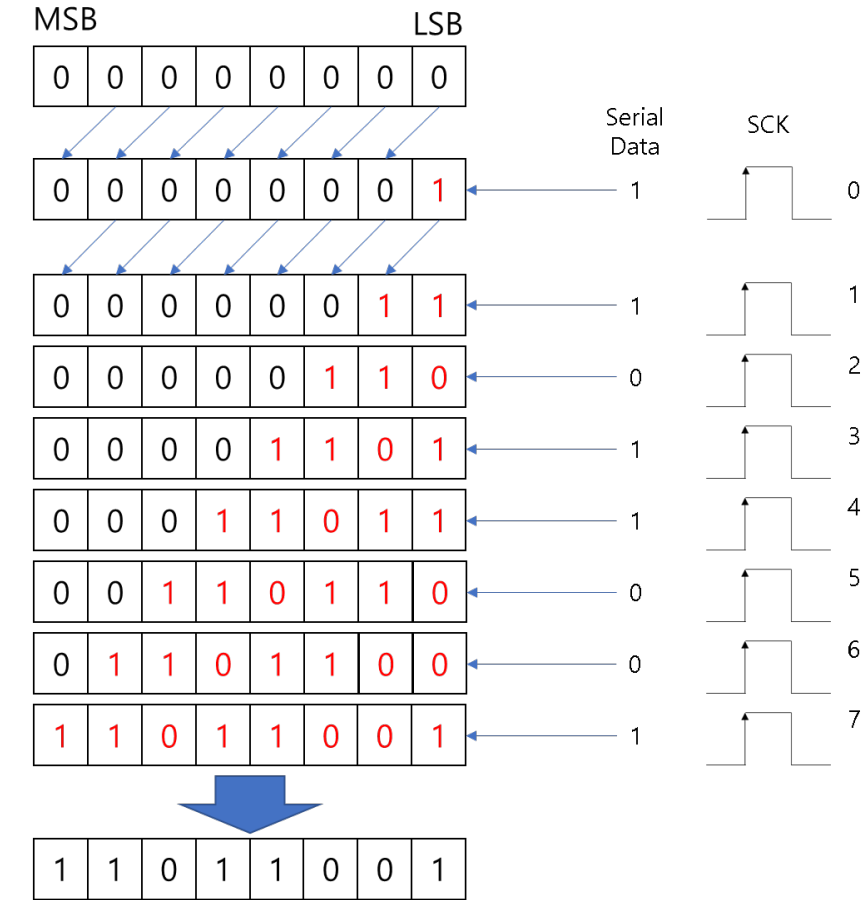


ADC(Analog-to-digital converting)

- Shift register
 - Input 동작



8 bit 데이터를 serial data로 부터 sampling하여 입력하기 위해서는 8개의 SCK 필요

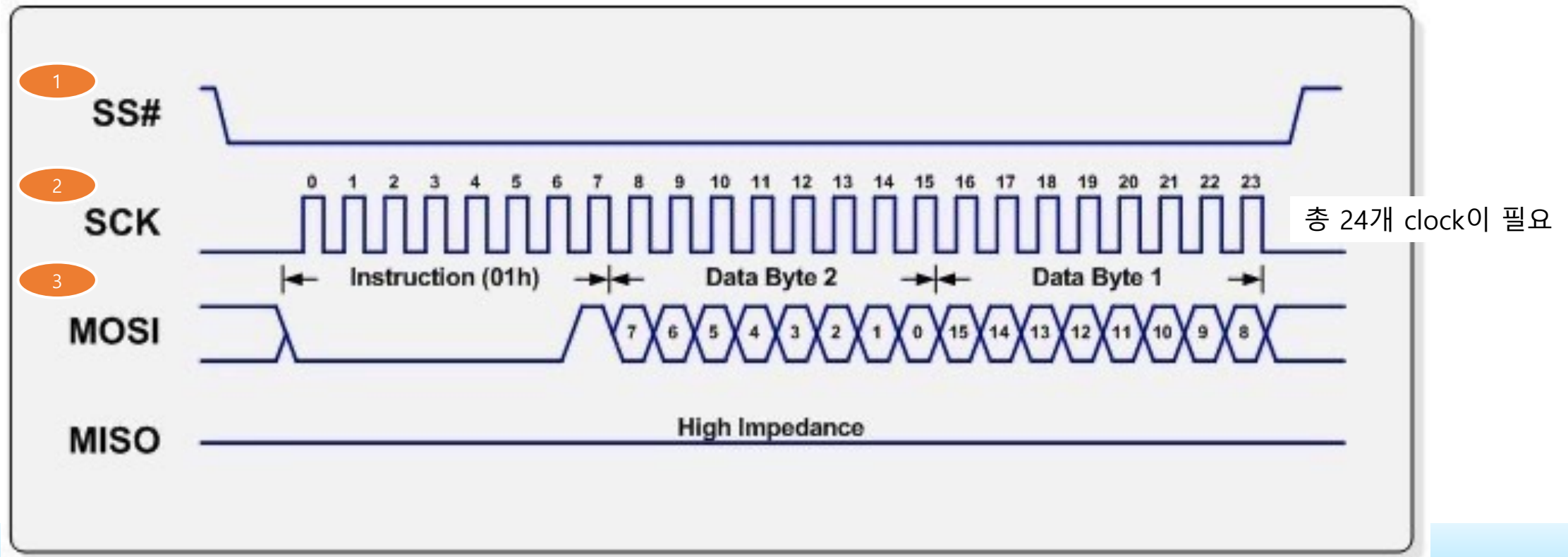


ADC(Analog-to-digital converting)

• SPI

- Data write 동작
 - Master에서 slave에 data를 쓰는 동작
 - 3 byte data write = 1 byte instruction + 2 byte data
 - 총 24 bit 데이터를 24개의 SCK clock에 동기 되어 전송

- 1 Master가 SS/CS신호를 low active시켜 slave 선택
- 2 Master가 동기 clock인 SCK를 출력
- 3 Master가 SCK에 동기 되어 data bit를 출력
- 4 선택된 Slave는 SCK에 동기 되어 data bit을 샘플링

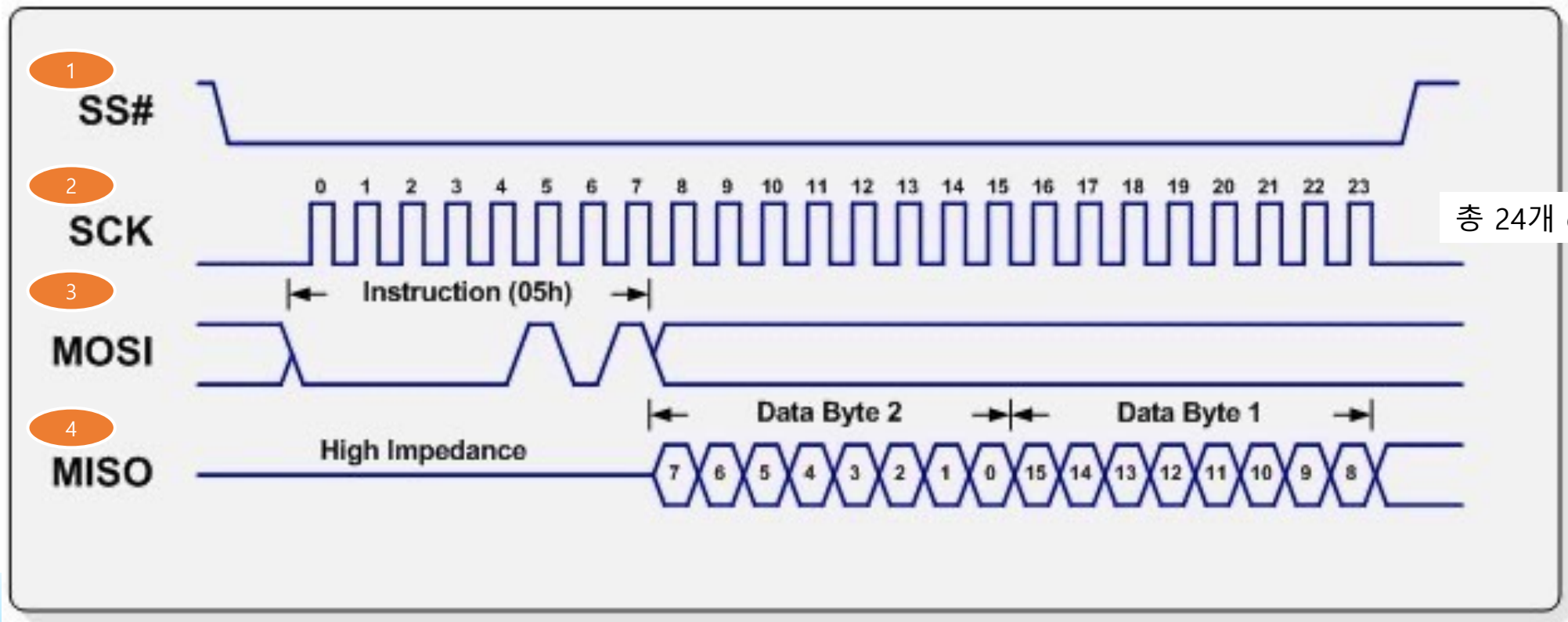


ADC(Analog-to-digital converting)

• SPI

- Data read 동작
 - Master가 Slave로 부터 데이터를 읽어 오는 동작
 - 1 Byte 명령을 master가 slave에 전송하고 slave는 2 byte 데이터를 master에게 전송
 - 총 24 bit 데이터를 24개의 SCK clock에 동기 되어 전송

- 1 Master가 CS(SS)신호를 low active시켜 slave 선택
- 2 Master가 동기 clock인 SCK를 출력
- 3 Master가 SCK에 동기 되어 instruction bit를 출력
- 4 선택된 Slave는 SCK에 동기 되어 data bit을 출력



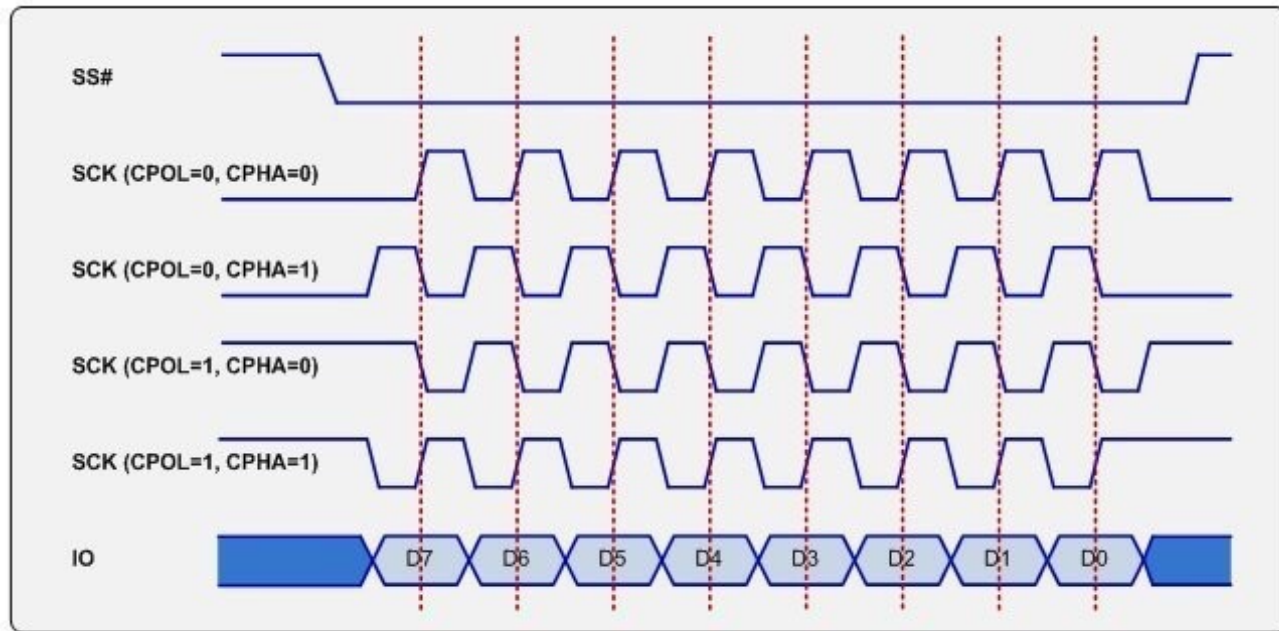
총 24개 clock이 필요

ADC(Analog-to-digital converting)

• SPI

• SPI mode

- SCK 상태와 데이터 sampling을 위한 SCK rising/falling edge를 설정
 - CPOL(Clock POLarity)
 - SCK 상태 설정
 - CPHA(Clock PHAse)
 - Data sampling falling/rising edge 설정



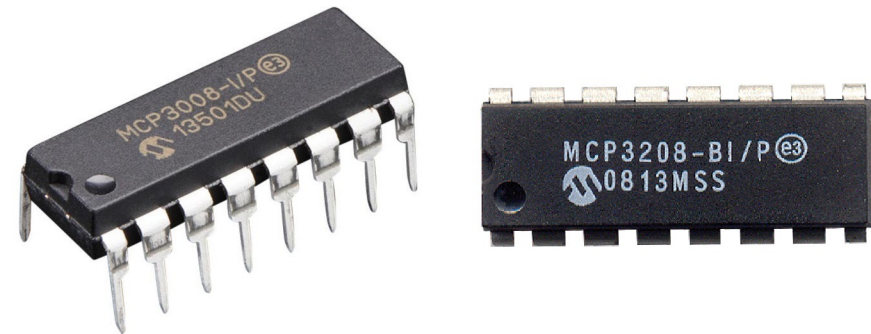
구분	0	1
CPOL	SCK 상태 Low 유지	SCK 상태 High 유지
CPHA	Rising edge	Falling edge

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

ADC(Analog-to-digital converting)

- **MCP3208 ADC**

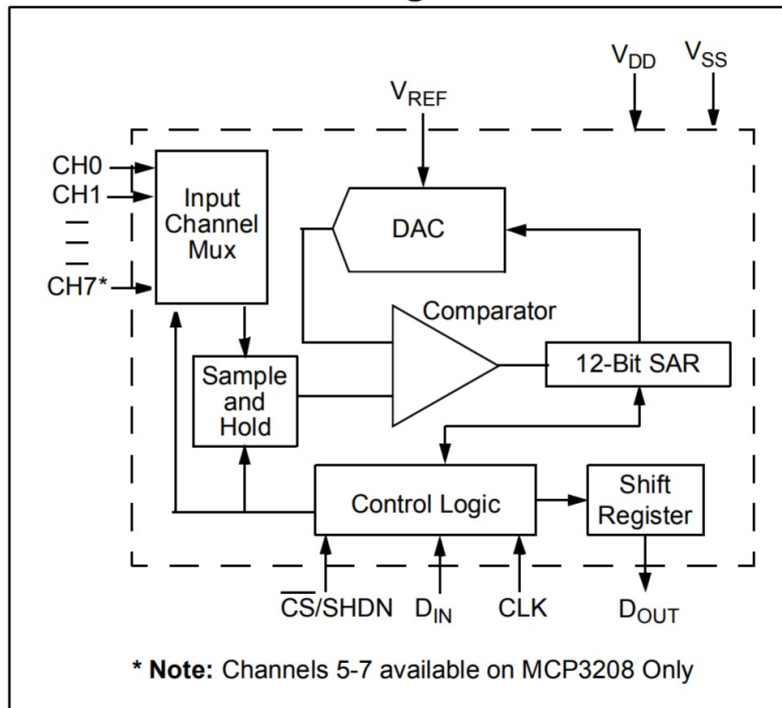
- 12-bit resolution
- 8ch ADC
- On-chip sample and hold
- Sampling rate
 - 100 ksps max.@VDD 5V
 - 50 ksps max.@VDD 2.7V
- Low power CMOS technology
 - 500nA typical standby current
 - 400yA max. active current
- Interface
 - SPI protocol



ADC(Analog-to-digital converting)

• MCP3208 ADC

Functional Block Diagram



구분	설명	규격
CH0~CH7	ADC channel	<ul style="list-style-type: none">Single ended: 8chDifferential: 4ch
V_{DD}	전원	<ul style="list-style-type: none">+5V
V_{SS}	Ground	
V_{REF}	Reference voltage	<ul style="list-style-type: none">+5V
CLK	SCK(Clock)	<ul style="list-style-type: none">2MHz max
D_{IN}	MOSI	<ul style="list-style-type: none">Serial 데이터 입력
D_{OUT}	MISO	<ul style="list-style-type: none">Serial 데이터 출력
$\overline{CS}/SHDN$	Chip selection	

ADC(Analog-to-digital converting)

- Reference voltage

$$V_{ref} = +5V$$

ADC 변환 가능 최대 전압

0V

$V_{ref} = +5V$ 이고 ADC bit resolution이 12 bit이면
+5V를 $(2^{12} - 1)$ 단계로 나눠서 ADC 수행

$(2^{12} - 1)$ 단계 존재

$$\text{Resolution}(V/bit) = \frac{V_{ref}}{(2^n - 1)}$$
$$\frac{5}{(2^{12} - 1)} = 0.001221V/bit$$

ADC(Analog-to-digital converting)

- **MCP3208 ADC**

- $V_{REF} = +5V$, ADC bit resolution $n=12$ bit

- Resolution

$$Resolution(v/bit) = \frac{V_{ref}}{(2^n - 1)}$$
$$\frac{5}{(2^{12} - 1)} = 0.001221V/bit$$

V_{ref} : Reference voltage(ADC 최대 전압 범위)
 n : 최대 bit 수

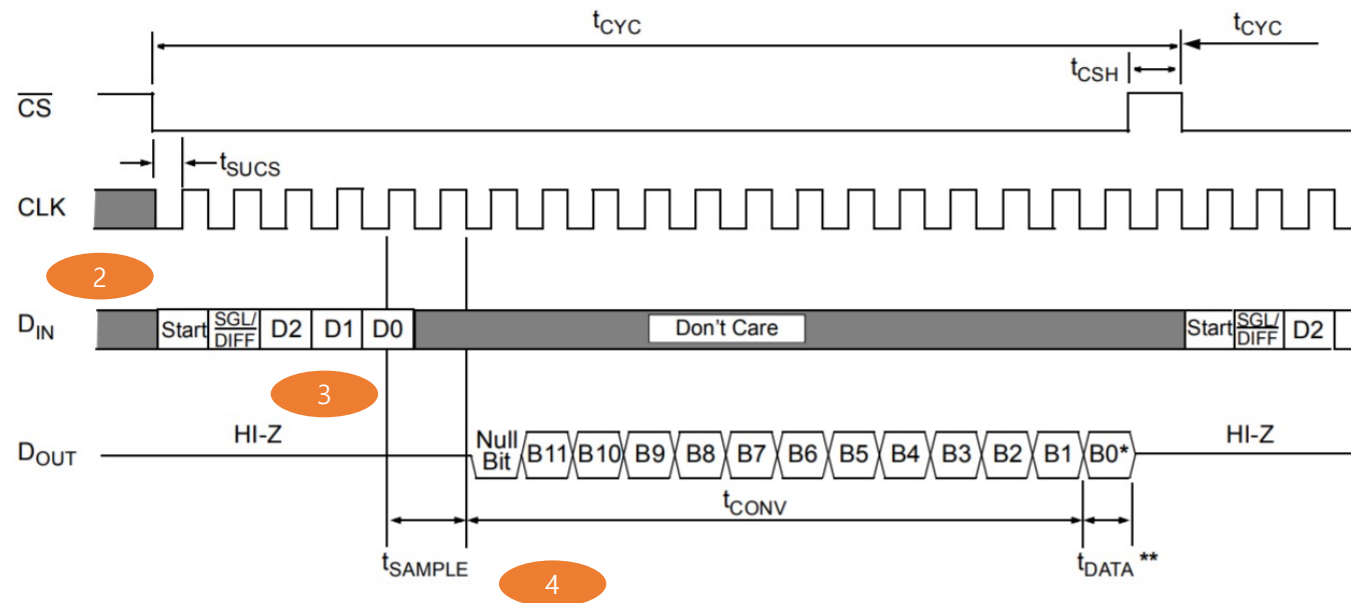
- ADC값 환산

$$V_{ADC}(V) = ADC \times Resolution$$
$$= ADC \times \frac{V_{ref}}{(2^n - 1)}$$
$$= ADC \times \frac{5}{(2^{12} - 1)}$$

ADC(Analog-to-digital converting)

• MCP3208 ADC 동작

- SPI 인터페이스에 의해 제어
 - 3 byte 데이터 구성

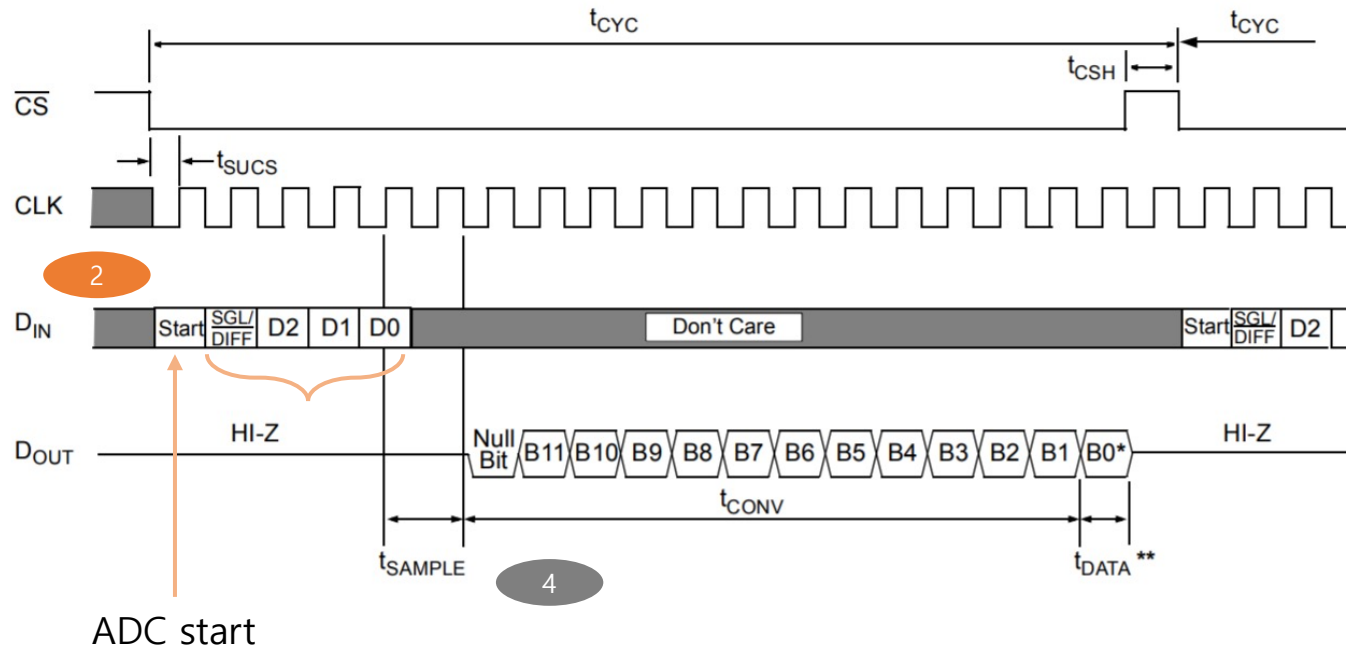


순서	동작	방향
1	Chip selection	master → ADC
2	ADC 시작 및 ADC 설정 데이터 write (3 byte data write)	master → ADC
3	ADC 수행	
4	ADC data read (3 byte data read)	ADC → master

ADC(Analog-to-digital converting)

• MCP3208 ADC 동작

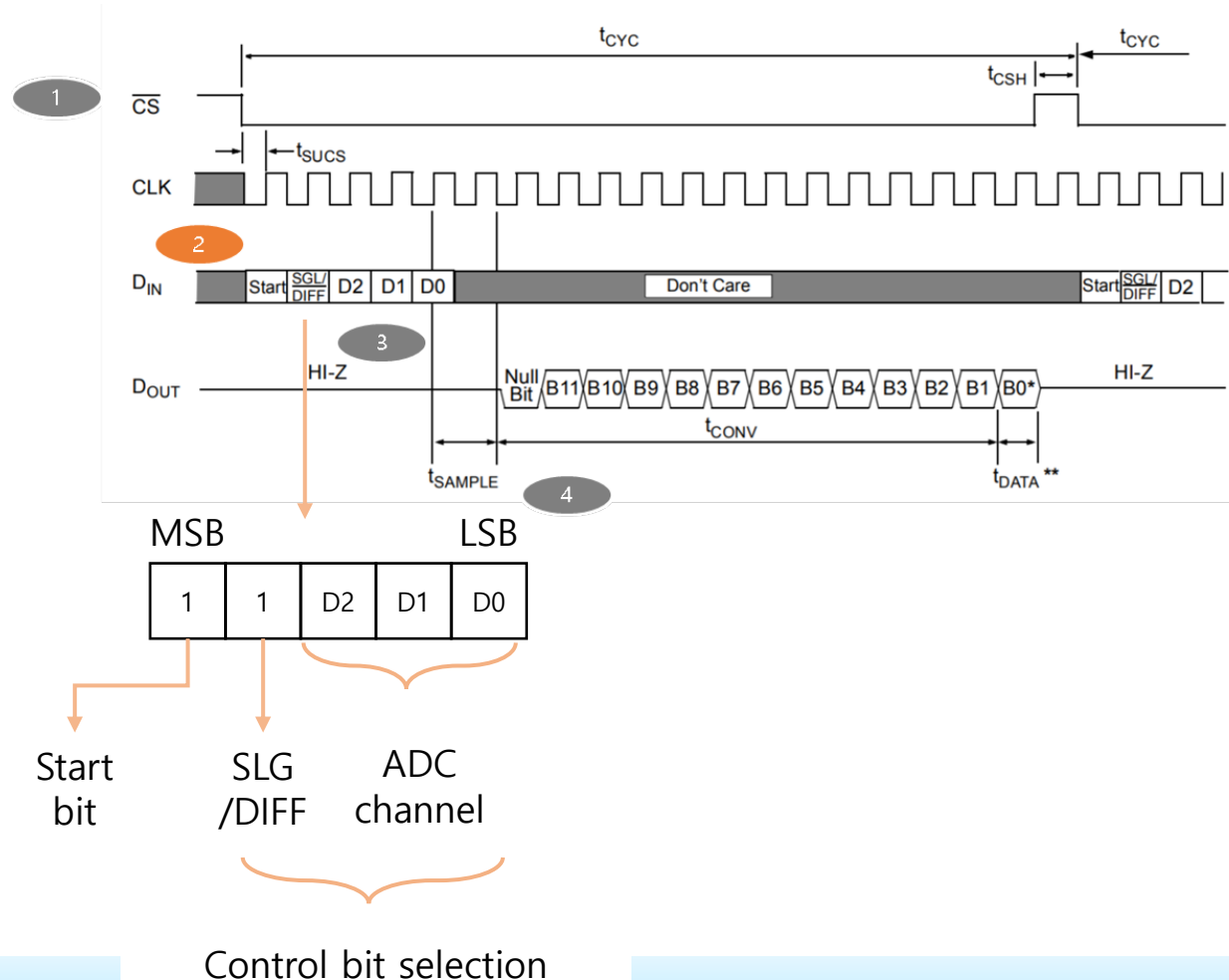
- ADC 시작 및 ADC 설정 데이터 write
 - master가 ADC에게 변환의 시작을 알림
 - ADC의 입력 조건 설정
 - ADC를 어떤 channel로 수행 할지 결정



ADC(Analog-to-digital converting)

• MCP3208 ADC 동작

- ADC 시작 및 ADC 설정 데이터 write



Control Bit Selections				Input Configuration	Channel Selection
Single/Diff	D2	D1	D0		
1	0	0	0	single-ended	CH0
1	0	0	1	single-ended	CH1
1	0	1	0	single-ended	CH2
1	0	1	1	single-ended	CH3
1	1	0	0	single-ended	CH4
1	1	0	1	single-ended	CH5
1	1	1	0	single-ended	CH6
1	1	1	1	single-ended	CH7
0	0	0	0	differential	CH0 = IN+ CH1 = IN-
0	0	0	1	differential	CH0 = IN- CH1 = IN+
0	0	1	0	differential	CH2 = IN+ CH3 = IN-
0	0	1	1	differential	CH2 = IN- CH3 = IN+
0	1	0	0	differential	CH4 = IN+ CH5 = IN-
0	1	0	1	differential	CH4 = IN- CH5 = IN+
0	1	1	0	differential	CH6 = IN+ CH7 = IN-
0	1	1	1	differential	CH6 = IN- CH7 = IN+

8개의 ADC channel 중 하나 선택

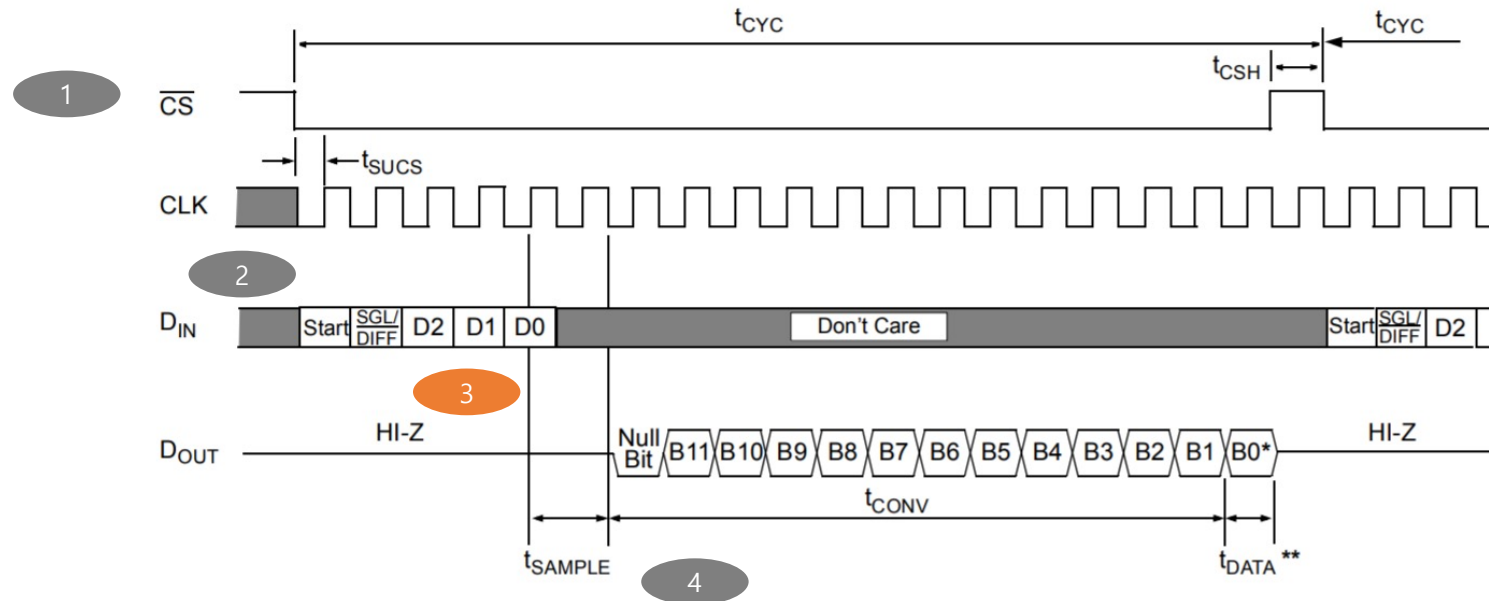
ADC(Analog-to-digital converting)

- MCP3208 ADC 동작

- ADC 수행

- t_{sample} 시간동안 입력 전압에 대하여 sampling 수행

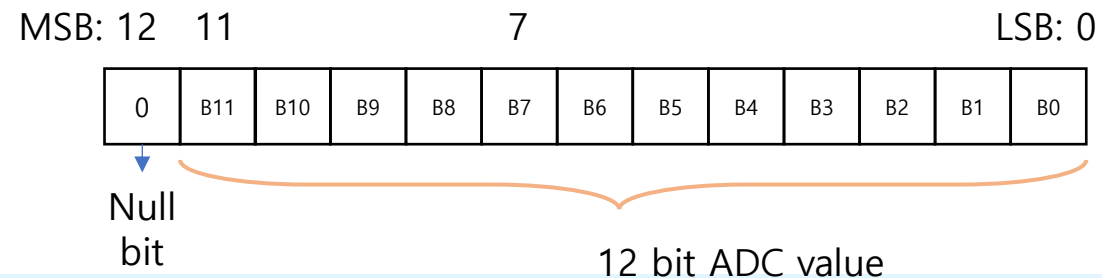
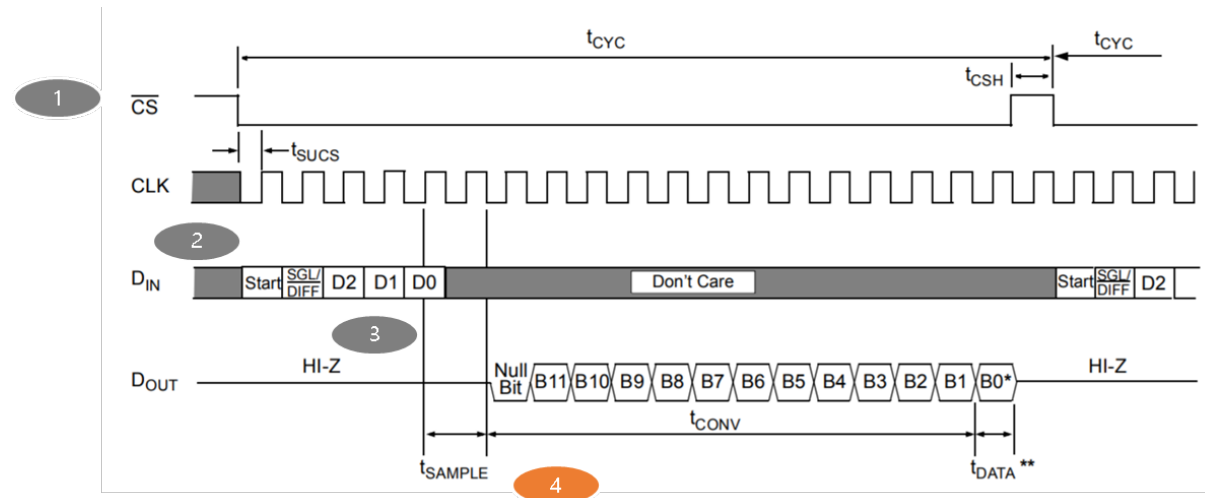
Analog Input Sample Time	t_{SAMPLE}	1.5	clock cycles	
--------------------------	---------------------	-----	--------------	--



ADC(Analog-to-digital converting)

• MCP3208 ADC 동작

- ADC data read
 - ADC 결과를 master로 읽어오는 단계
 - null bit(1 bit) + 12 bit ADC value



ADC(Analog-to-digital converting)

- **Wiring Pi SPI 함수**

- SPI setup function
 - Raspberry Pi GPIO핀을 SPI interface로 설정
 - Raspberry Pi에서 2개의 SPI interface를 지원
 - wiringPiSPI.h 포함시켜야 함

SPI setup function

int wiringPiSPISetup (int channel, int speed)

입력:

channel: SPI channel (0 또는 1)

speed: SPI interface clock speed

SPI clock frequency를 Hz 단위로 표시

500,000 ~ 32,000,000 범위

출력:

-1: error 발생

ADC(Analog-to-digital converting)

- **Wiring Pi SPI 함수**

- SPI data write/read function
 - SPI interface를 사용하여 data를 write/read
 - Data buffer에 ADC 제어 값을 써서 ADC 수행 시작
 - Data buffer에 ADC 된 결과 값을 넣어 줌
 - wiringPiSPI.h 포함 시켜야 함

SPI data read/write function

int wiringPiSPIDataRW (int channel, unsigned char *data, int len)

입력:

channel: SPI channel (0 또는 1)

data: SPI data write/read를 위한 메모리 포인터

함수가 호출 되면 ADC된 결과 값이 포인터가 가리키는 메모리에 저장

len: SPI data write/read를 위한 메모리 길이

출력:

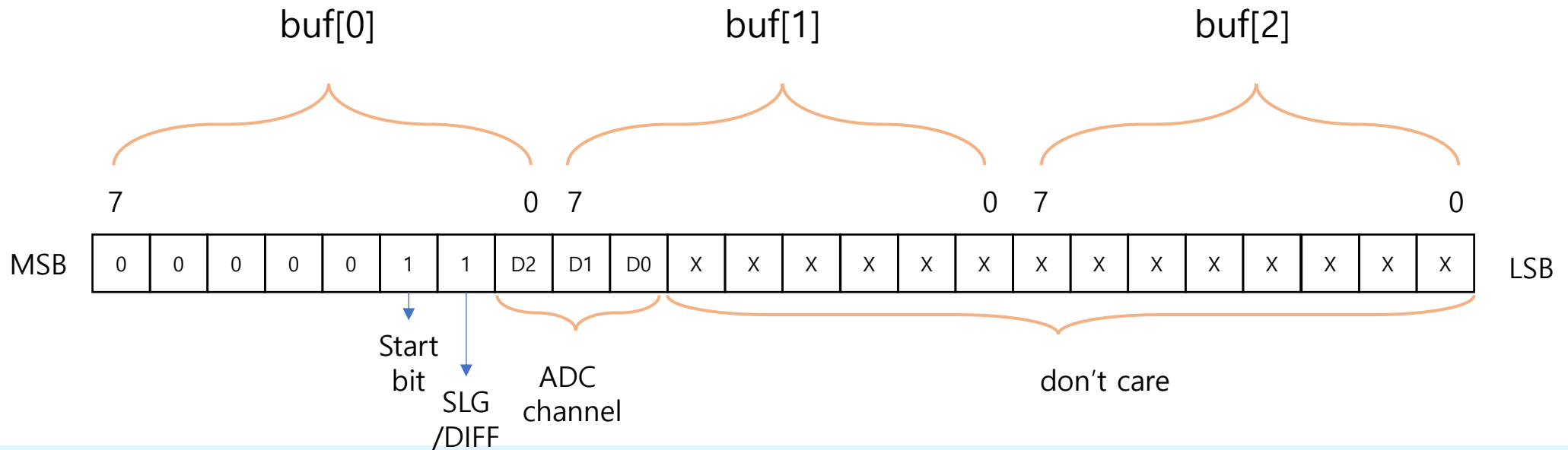
-1: error 발생

ADC(Analog-to-digital converting)

- **Wiring Pi SPI 함수**

- SPI data write/read function
 - ADC 시작
 - 3 byte data write

```
uint8_t buf[3]; // 8 bit buffer  
wiringPiSPIDataRW (SPICH, buf, 3);
```



ADC(Analog-to-digital converting)

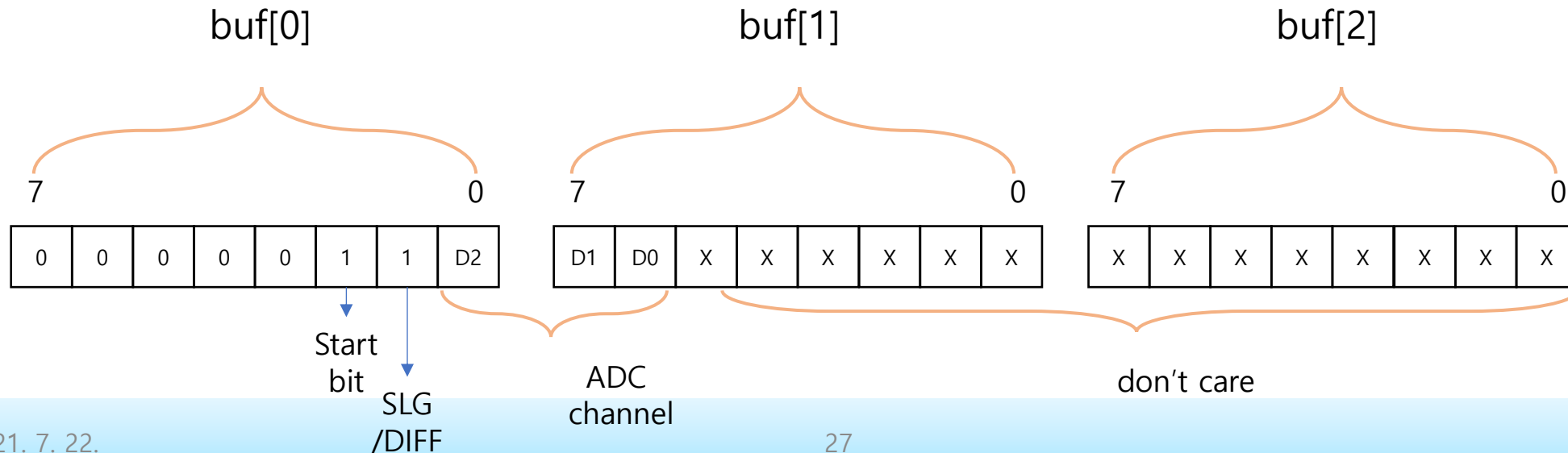
- **Wiring Pi SPI 함수**

- SPI data write/read function
 - ADC 시작

```
#define START_BIT 1
#define SLG_DIFF_BIT 1
#define ADCCH 3

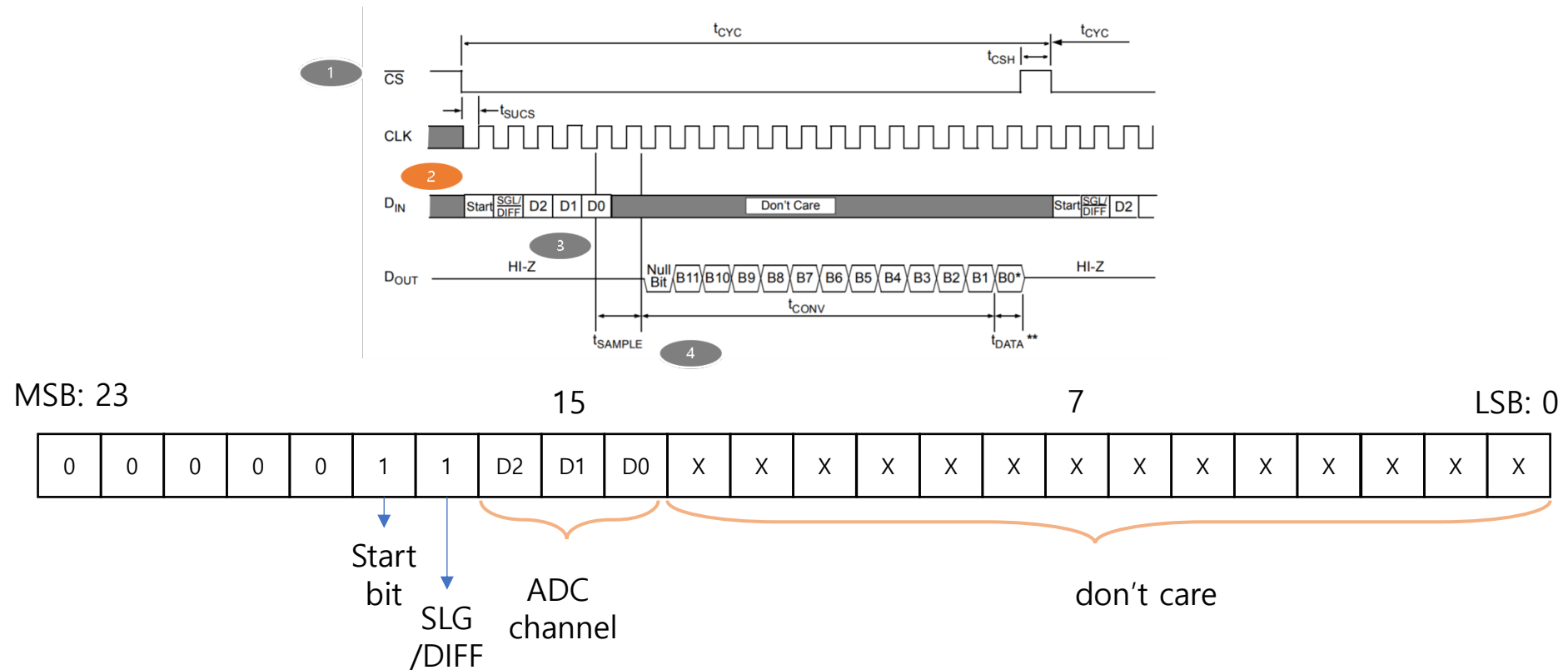
...
buf[0]=ADCCH>>2;
buf[0]=START_BIT<<2 | SLG_DIFF_BIT<<1 | buf[0];
buf[1]=ADCCH<<6;
buf[2]=0;

...
wiringPiSPIDataRW (ADCCH, buf, 3);
```



ADC(Analog-to-digital converting)

- Wiring Pi SPI 함수
 - SPI data write/read function
 - ADC 시작
 - 3 byte data write



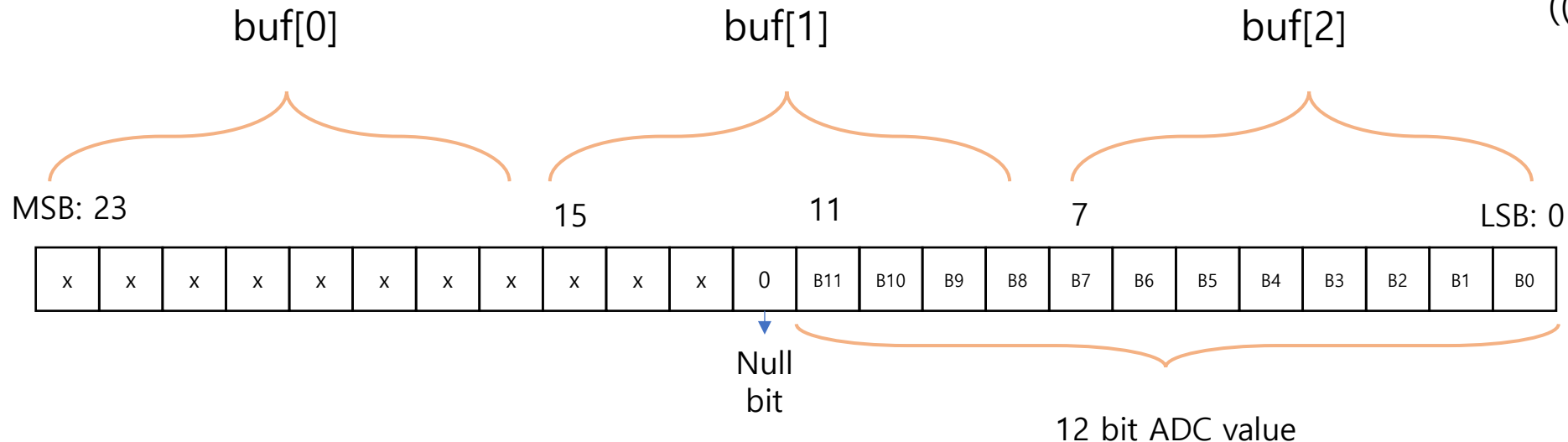
ADC(Analog-to-digital converting)

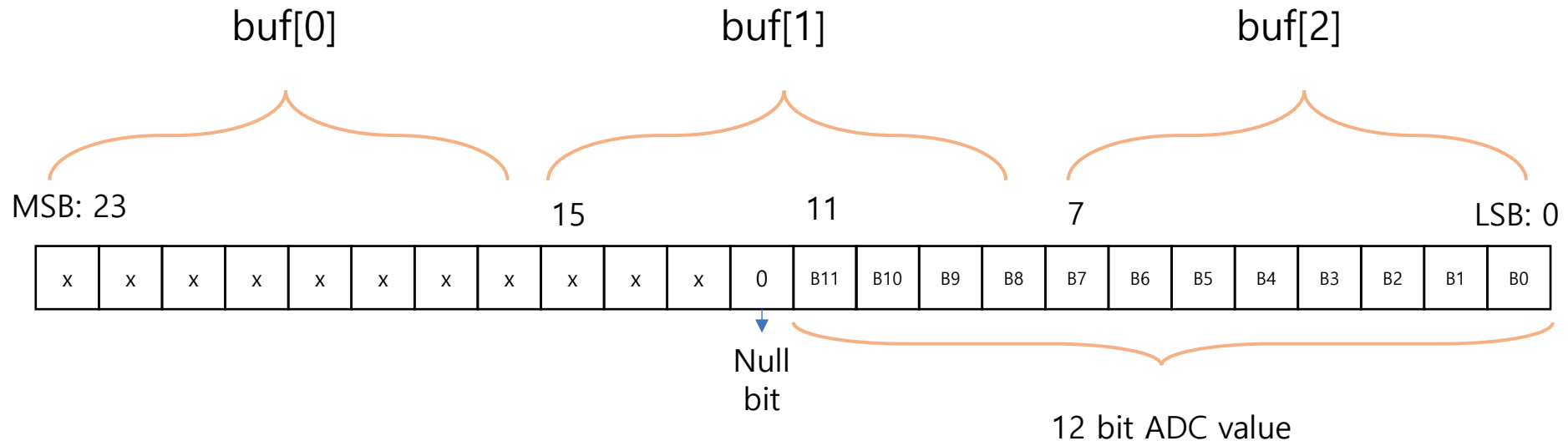
- **ADC 동작**

- SPI data write/read function
 - ADC data read
 - 3 byte data read → null bit(1 bit) + 12 bit ADC value

```
uint8_t buf[3]; // 8 bit buffer  
wiringPiSPIDataRW (SPICH, buf, 3);
```

adcValue =
((buf[1]&0xF)<<8)|buf[2];





```
adcValue = ((buf[1]&0xF)<<8)|buf[2];
```

ADC(Analog-to-digital converting)

- **Raspberry Pi ADC SPI 설정**

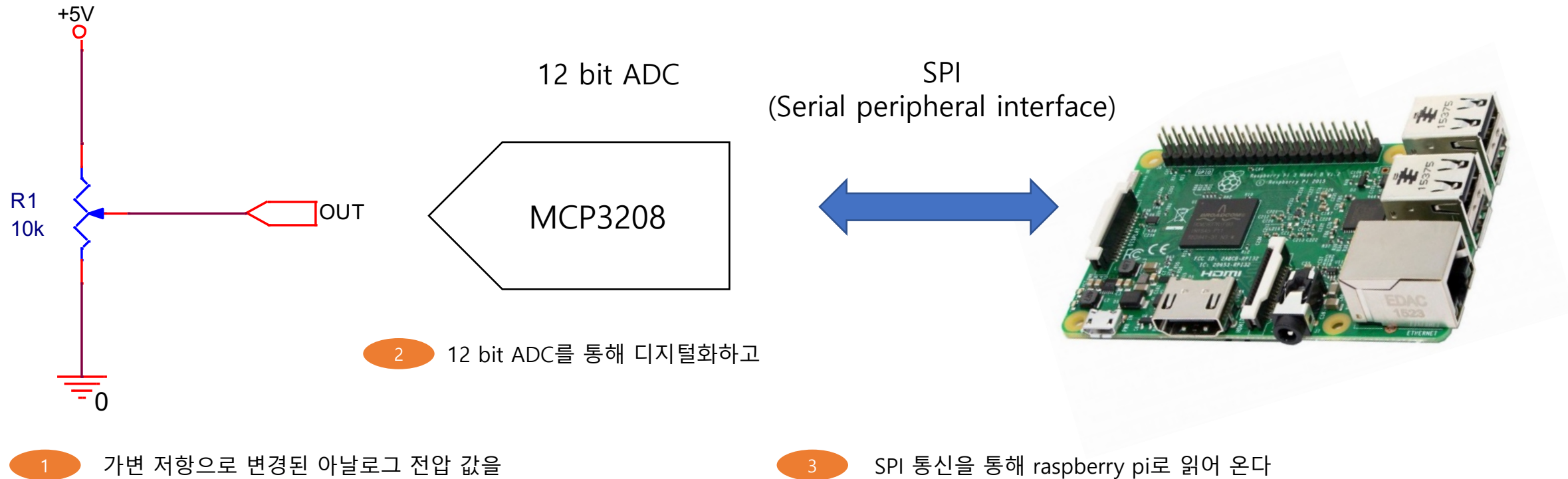
- Raspberry Pi는 2개의 SPI chip selection 제공
 - 실습에서는 ADC를 SPI chip selection 0에 연결
 - SPI chip selection 0은 GPIO_8에 할당

BCM (Raspberry Pi)	wPi (Wiring Pi)	ADC 연결	설명
GPIO_8	10	ADC_CS	ADC chip selection
GPIO_9	13	ADC_MISO	ADC MISO
GPIO_10	12	ADC_MOSI	ADC MOSI
GPIO_11	14	ADC_SCK	ADC clock

ADC 연결	가변 저항
ADC_CH0	OUT

ADC(Analog-to-digital converting)

- Raspberry Pi ADC SPI 설정
 - 가변 저항 및 ADC 구성



ADC(Analog-to-digital converting)

• 실습 1

- MCP3208 ADC를 사용하여 가변저항의 저항 값 변화에 따른 전압을 측정
 - Raspberry Pi SPI channel: 0
 - SPI interface clock speed: 500 kHz
 - ADC channel: 0 channel
 - Reference voltage: 5V
 - Bit Resolution: 12 bit
- MCP3208 사용하여 ADC를 하기 위한 제어 함수 작성
 - SPI를 통해서 ADC 변환을 제어하고 ADC된 값을 반환하는 함수

int32_t ADCRead(int SPI_CH, uint32_t ADC_CH)

ADC(Analog-to-digital converting)

- 실습 1

- SPI를 통해서 ADC 변환을 제어하고 ADC된 값을 반환하는 함수

int32_t ADCRead(int SPI_CH, uint32_t ADC_CH)

입력:

- SPI_CH: Raspberry Pi SPI channel [0:1]
- ADC_CH: MCP3208 ADC channel 선택 [0:7]

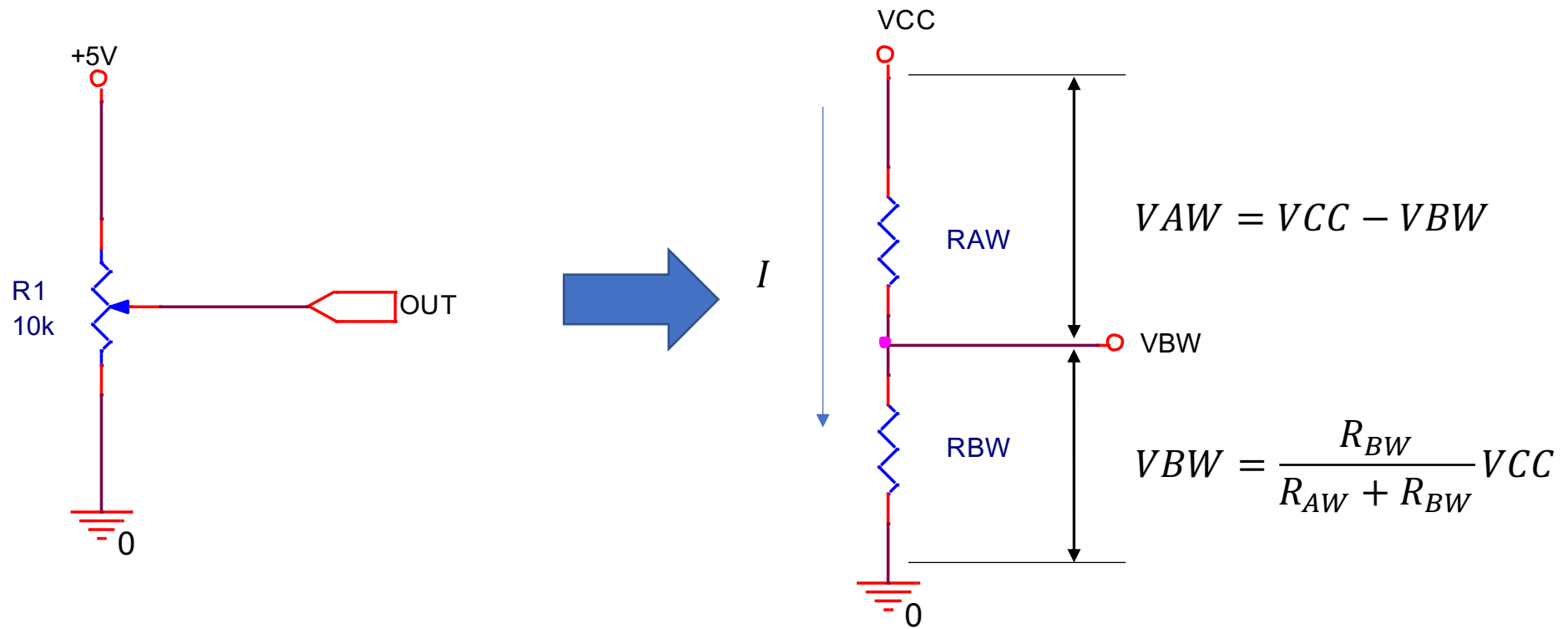
리턴:

- SPI로 부터 읽어 들인 ADC 결과 값

ADC(Analog-to-digital converting)

• 실습 1

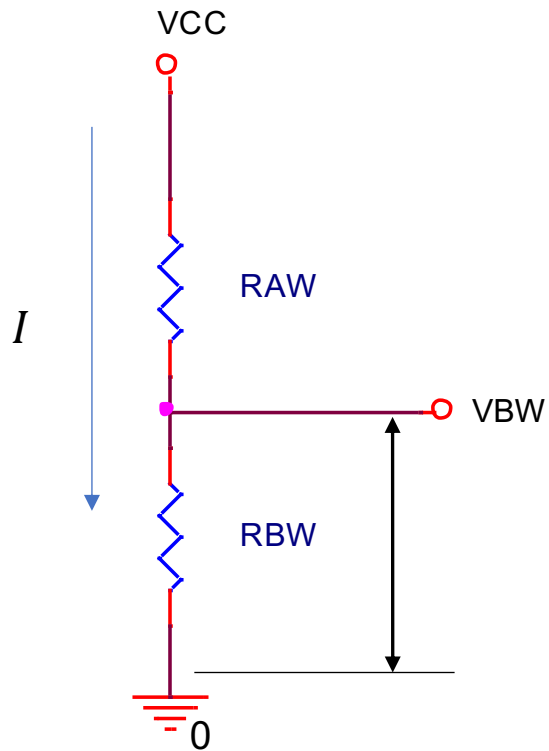
- MCP3208 ADC를 사용하여 가변저항의 저항 값 변화에 따른 전압을 측정



ADC(Analog-to-digital converting)

• 실습 1

- MCP3208 ADC를 사용하여 가변저항의 저항 값 변화에 따른 전압을 측정



$$VBW = \frac{R_{BW}}{R_{AW} + R_{BW}} VCC$$

$$VBW = \frac{R_{BW}}{10k\Omega} VCC$$

저항 값 가변 범위: $0 \leq R_{BW} \leq 10k\Omega$

전압의 가변 범위: $0 \leq VBW \leq VCC$

ADC(Analog-to-digital converting)

- 실습 1

전압의 가변 범위: $0 \leq VBW \leq VCC$

$$ADC\ Value = \frac{VBW}{Resoultion}$$

$$\begin{aligned} Resoultion &= \frac{V_{REF}}{(2^n - 1)} \\ &= \frac{5V}{(2^{12} - 1)} \\ &\cong 0.001221\ V/bit \end{aligned}$$

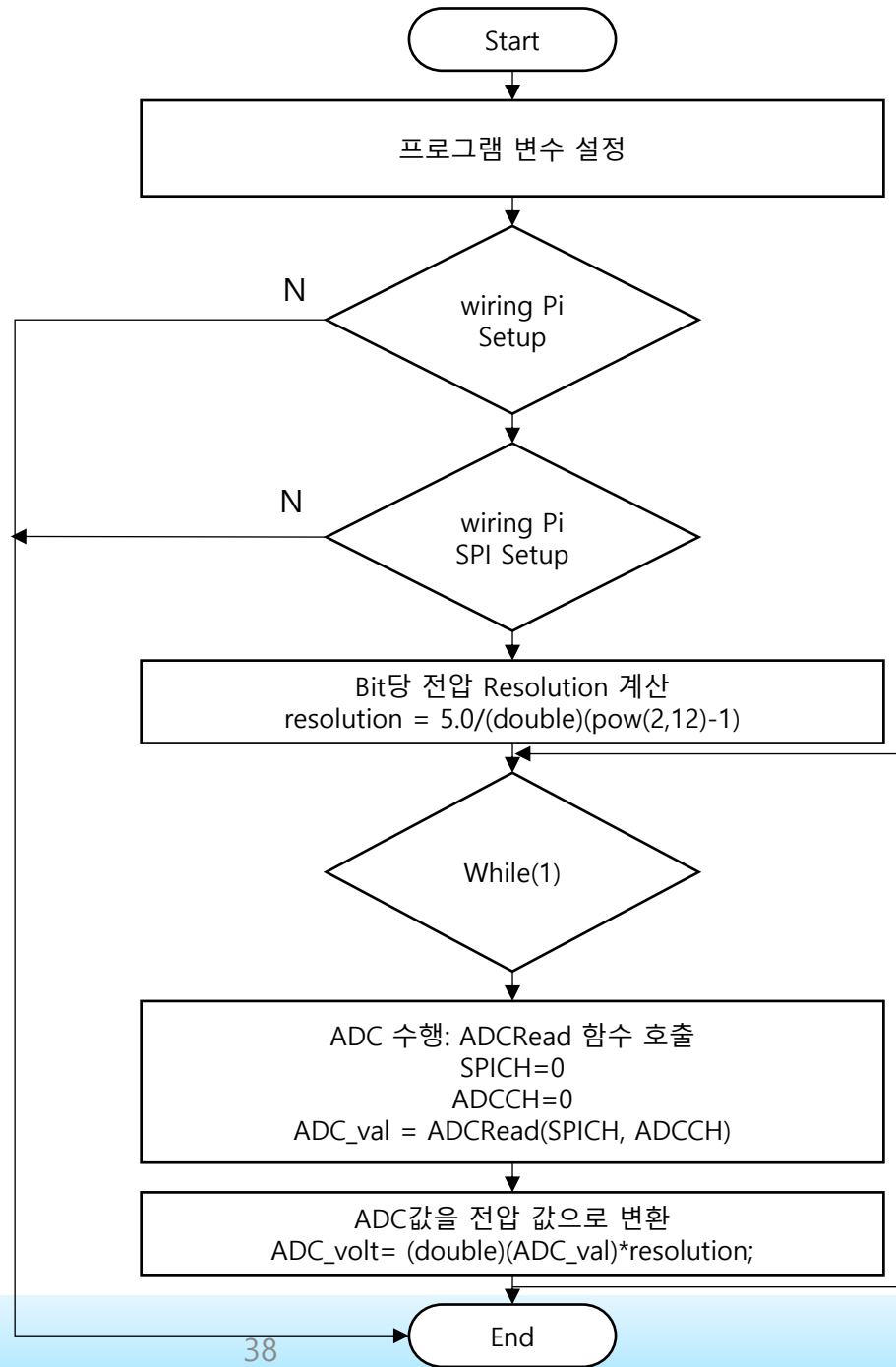
$$VBW(V) = ADC\ Value \times Resoultion$$



V_{REF} : ADC reference voltage
 n : ADC bit resoultion

ADC

• 실습 1



ADC(Analog-to-digital converting)

- 실습 1

```
#include <stdint.h>
#include <stdio.h>
#include <wiringPi.h>    // wiringPi
#include <wiringPiSPI.h> // SPI
#include <math.h>        // sqrt

#define SPICH          0 //SPI Channel 0
#define ADCCH          0 //ADC Channel 0
#define SPI_SCK        500000 //SPI Clock 500kHz

#define START_BIT      1 // Start bit 1
#define SLG_DIFF_BIT   1 // SLG/DIFF bit

int32_t ADCRead(int SPI_CH, uint32_t ADC_CH);    // SPI를 사용한 ADC 함수 선언
```

ADC(Analog-to-digital converting)

- 실습 1

```
int main(void)
{
    uint32_t ADC_val;           // ADC 값
    double ADC_volt;           // ADC 값을 전압으로 변환한 값
    double resolution;         // Bit 당 해상도

    if(wiringPiSetup()==-1)
        return 1;

    if(wiringPiSPISetup(SPICH, SPI_SCK)==-1)
    {
        printf("SPI set-up 실패\n");
        return 1;
    }

    // Bit당 resolution 구하기
    resolution = 5.0/(double)(pow(2,12)-1);

    while(1)
    {
        ADC_val = ADCRead(SPICH,ADCCH); // SPI를 통해서 ADC 값 읽어 오기
        ADC_volt= (double)(ADC_val)*resolution;
        printf("ADC value:%d\t전압(V):%f\n", ADC_val,ADC_volt);
    }
}
```


ADC(Analog-to-digital converting)

- 실습 1

```
// ADC값을 SPI를 통해 읽어오기
int32_t ADCRead(int SPI_CH, uint32_t ADC_CH)
{
    uint8_t buf[3];
    int32_t adcValue = 0;

    buf[0] = (START_BIT<<2)|(SLG_DIFF_BIT<<1)|(ADCCH>>2);
    buf[1] = (ADC_CH&3)<<6;
    buf[2] = 0;

    wiringPiSPIDataRW(SPI_CH, buf, 3);

    adcValue = ((buf[1]&0xF)<<8)|buf[2];

    return adcValue;
}
```

ADC(Analog-to-digital converting)

• 실습 1

<파일명>

mcp3208_vr.c

<Compile 명령>

gcc mcp3208_vr.c -o mcp3208_vr -lwiringPi -lm

<실행>

sudo ./mcp3208_vr

<출력>

```
ADC value:3671 전압(V):4.482295
ADC value:3674 전압(V):4.485958
ADC value:3687 전압(V):4.501832
ADC value:3671 전압(V):4.482295
ADC value:3667 전압(V):4.477411
ADC value:3677 전압(V):4.489621
ADC value:3687 전압(V):4.501832
ADC value:3679 전압(V):4.492063
ADC value:3668 전압(V):4.478632
ADC value:3680 전압(V):4.493284
ADC value:3687 전압(V):4.501832
ADC value:3680 전압(V):4.493284
ADC value:3672 전압(V):4.483516
ADC value:3674 전압(V):4.485958
ADC value:3693 전압(V):4.509158
ADC value:3667 전압(V):4.477411
ADC value:3675 전압(V):4.487179
ADC value:3679 전압(V):4.492063
ADC value:3690 전압(V):4.505495
ADC value:3671 전압(V):4.482295
ADC value:3671 전압(V):4.482295
```

ADC(Analog-to-digital converting)

• 실습 2

- AD 변환된 전압 값의 평균 및 표준 편차를 구하는 프로그래밍
 - 100번 측정된 ADC값을 사용하여 평균 및 표준편차를 구한다
 - MCP3208 설정
 - SPI channel:0
 - SPI interface clock speed: 500 kHz
 - ADC channel: 0 channel
 - Reference voltage: 5V
 - Bit Resolution: 12 bit

ADC(Analog-to-digital converting)

• 실습 2

```
#include <stdint.h>
#include <stdio.h>
#include <wiringPi.h>    // wiringPi
#include <wiringPiSPI.h> // SPI
#include <math.h>        // sqrt

#define SPICH          0        //SPI Channel 0
#define ADCCH          0        //ADC Channel 0
#define SPI_SCK 500000        //SPI Clock 500kHz

#define START_BIT      1        // Start bit 1
#define SLG_DIFF_BIT 1        // SLG/DIFF bit

#define NUM_MEASURE 100        // 측정 횟수

int32_t ADCRead(int SPI_CH, uint32_t ADC_CH);    // SPI를 사용한 ADC 함수 선언
double mean(double* x, int size);    // 평균함수 선언
double STD(double* x, int size);    // 표준 편차 함수 선언
```

ADC(Analog-to-digital converting)

• 실습 2

```
int main(void)
{
    uint32_t ADC_val;          // ADC 값
    double ADC_volt;           // ADC 값을 전압으로 변환한 값
    double resolution; // Bit 당 해상도
    int i=0;
    double ADC_mea[NUM_MEASURE]; // 측정 전압을 저장
    double volt_mean, volt_std; // 측정 전압의 평균과 표준 편차

    // Bit당 resolution 구하기
    resolution = 5.0/(double)(pow(2,12)-1);

    if(wiringPiSetup() == -1)
        return 1;

    if(wiringPiSPISetup(SPICH, 1000000) == -1)
    {
        printf("SPI set-up 실패\n");
        return 1;
    }
}
```

ADC(Analog-to-digital converting)

- 실습 2

```
while(1)
{
    while(i < NUM_MEASURE)
    {
        ADC_val = ADCRead(SPICH,ADCCH); // SPI를 통해서 ADC 값 읽어 오기
        ADC_volt= (double)(ADC_val)*resolution;
        ADC_mea[i]=ADC_volt;
        i++;
    }

    volt_mean=mean(ADC_mea,sizeof(ADC_mea)/sizeof(ADC_mea[0]));
    volt_std=STD(ADC_mea,sizeof(ADC_mea)/sizeof(ADC_mea[0]));

    printf("측정 전압 평균(V):%f\t측정 전압 표준 편차(V):%f\n",volt_mean,volt_std);
    i=0;
}
```

ADC(Analog-to-digital converting)

- 실습 2

```
// ADC값을 SPI를 통해 읽어 오기
int32_t ADCRead(int SPI_CH, uint32_t ADC_CH)
{
    uint8_t buf[3];
    int32_t adcValue = 0;

    buf[0] = (START_BIT<<2)|(SLG_DIFF_BIT<<1)|(ADCCH>>2);
    buf[1] = (ADC_CH&3)<<6;
    buf[2] = 0;

    wiringPiSPIDataRW(SPI_CH, buf, 3);

    adcValue = ((buf[1]&0xF)<<8)|buf[2];

    return adcValue;
}
```

ADC(Analog-to-digital converting)

- 실습 2

```
// 산술 평균 구하기
double mean(double* x, int size)
{
    double sum = 0.0;        // sum 변수
    double mu = 0.0;         // 평균 결과
    int i = 0;                // 샘플 인덱스

    while(i < size)
    {
        sum += x[i]; // sum 누적
        i++;
    }

    mu = sum / size;          // 평균 계산

    return mu;
}
```


ADC(Analog-to-digital converting)

- 실습 2

```
// 표준 편차 구하기
double STD(double* x, int size)
{
    double sum = 0.0;        // sum 변수
    double sigma = 0.0;      // 표준 편차
    double diff;
    double mu = 0;
    int i = 0;

    // 배열 요소가 1개 이하 일 때는
    // -1로 리턴
    if (size < 2) return sqrt(-1.0);

    // 평균 계산
    mu = mean(x, size);

    while(i < size)
    {
        diff = x[i] - mu;
        sum += diff * diff;
        i++;
    }

    sigma = sqrt(sum / (size - 1));

    return sigma;
}
```

ADC(Analog-to-digital converting)

<파일명>

mcp3208_vr_mean.c

<Compile 명령>

gcc mcp3208_vr_mean.c -o mcp3208_vr_mean -lwiringPi -lm

<실행>

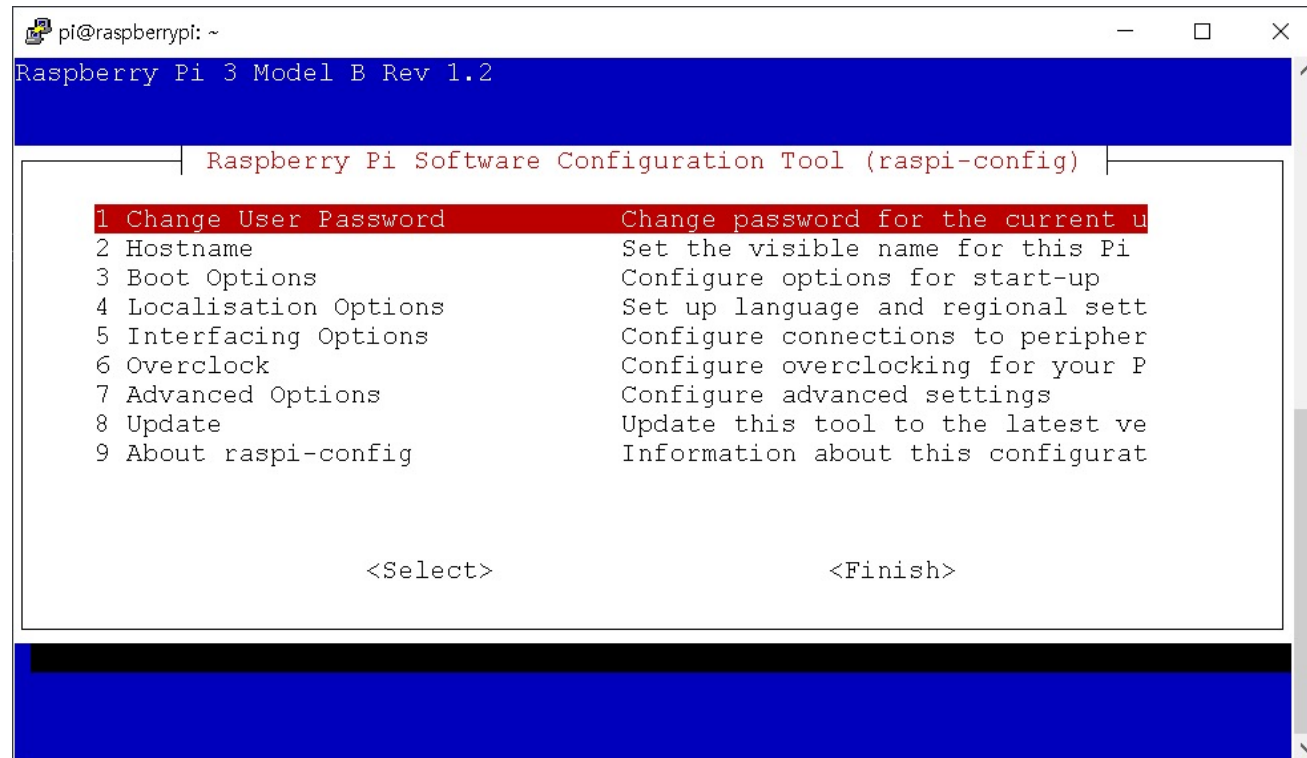
sudo ./mcp3208_vr_mean

측정 전압	평균(V):4.493541	측정 전압	표준 편차(V):0.008190
측정 전압	평균(V):4.492650	측정 전압	표준 편차(V):0.008298
측정 전압	평균(V):4.493468	측정 전압	표준 편차(V):0.008198
측정 전압	평균(V):4.493284	측정 전압	표준 편차(V):0.007829
측정 전압	평균(V):4.492759	측정 전압	표준 편차(V):0.008205
측정 전압	평균(V):4.492979	측정 전압	표준 편차(V):0.008220
측정 전압	평균(V):4.493114	측정 전압	표준 편차(V):0.008184
측정 전압	평균(V):4.492808	측정 전압	표준 편차(V):0.008060
측정 전압	평균(V):4.493480	측정 전압	표준 편차(V):0.008134
측정 전압	평균(V):4.493175	측정 전압	표준 편차(V):0.007947
측정 전압	평균(V):4.492625	측정 전압	표준 편차(V):0.008380
측정 전압	평균(V):4.492662	측정 전압	표준 편차(V):0.008044
측정 전압	평균(V):4.493053	측정 전압	표준 편차(V):0.008464
측정 전압	평균(V):4.492955	측정 전압	표준 편차(V):0.008270
측정 전압	평균(V):4.493175	측정 전압	표준 편차(V):0.008131
측정 전압	평균(V):4.493175	측정 전압	표준 편차(V):0.007944
측정 전압	평균(V):4.493016	측정 전압	표준 편차(V):0.008222
측정 전압	평균(V):4.493272	측정 전압	표준 편차(V):0.008297
측정 전압	평균(V):4.492723	측정 전압	표준 편차(V):0.008336
측정 전압	평균(V):4.492869	측정 전압	표준 편차(V):0.008140
측정 전압	평균(V):4.492955	측정 전압	표준 편차(V):0.008503
측정 전압	평균(V):4.493529	측정 전압	표준 편차(V):0.007589
측정 전압	평균(V):4.493150	측정 전압	표준 편차(V):0.008151
측정 전압	평균(V):4.493162	측정 전압	표준 편차(V):0.008227
측정 전압	평균(V):4.492662	측정 전압	표준 편차(V):0.008766
측정 전압	평균(V):4.492930	측정 전압	표준 편차(V):0.008347
측정 전압	평균(V):4.493407	측정 전압	표준 편차(V):0.008416
측정 전압	평균(V):4.493797	측정 전압	표준 편차(V):0.008201

ADC(Analog-to-digital converting)

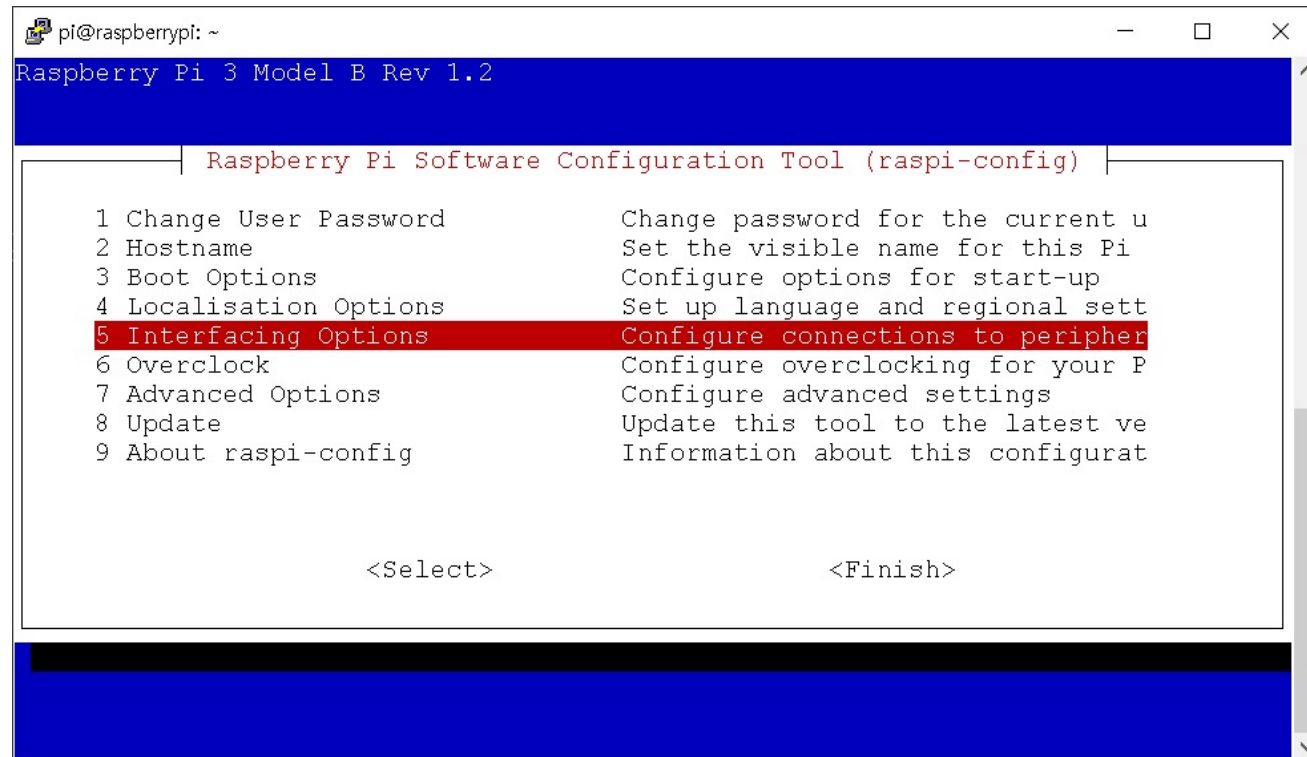
- SPI를 위한 Raspberry Pi setup
 - Raspberry Pi SPI module 활성화

```
pi@raspberrypi:~$ sudo raspi-config
```



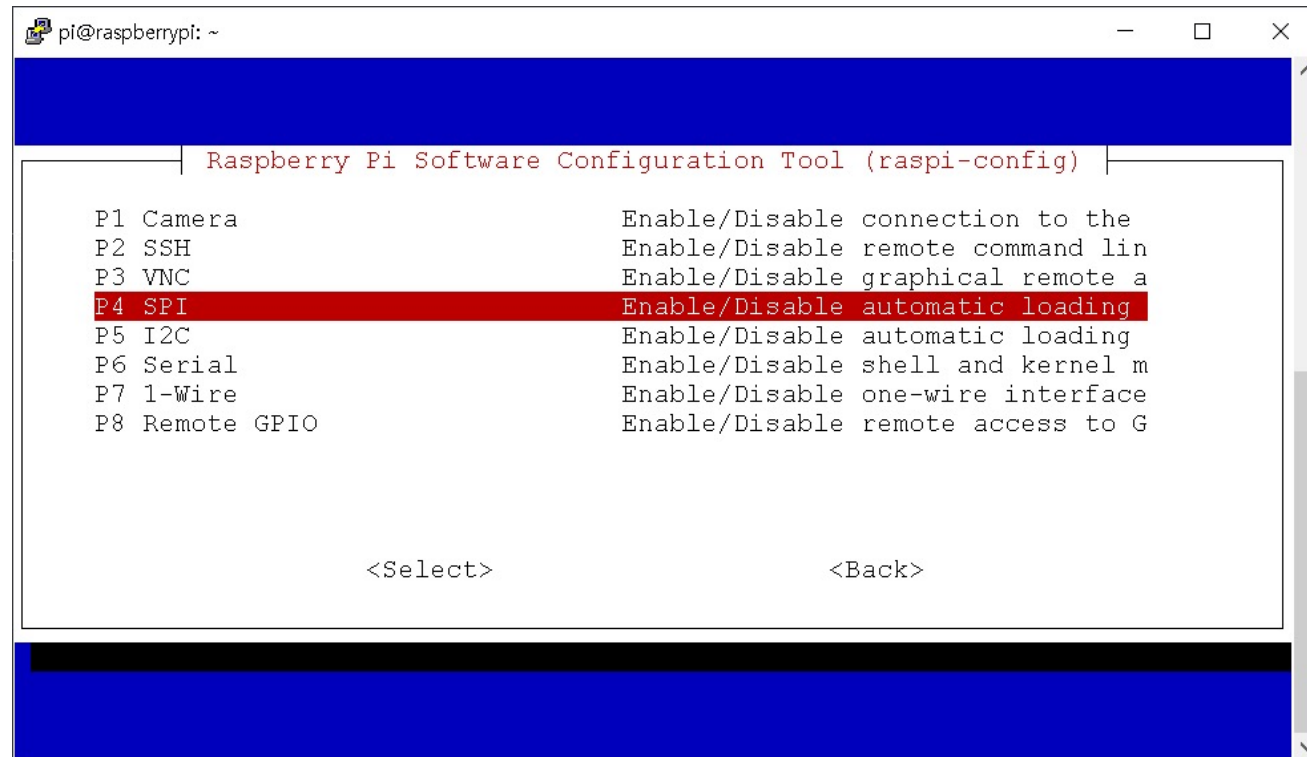
ADC(Analog-to-digital converting)

- SPI를 위한 Raspberry Pi setup
 - Raspberry Pi SPI module 활성화



ADC(Analog-to-digital converting)

- SPI를 위한 Raspberry Pi setup
 - Raspberry Pi SPI module 활성화



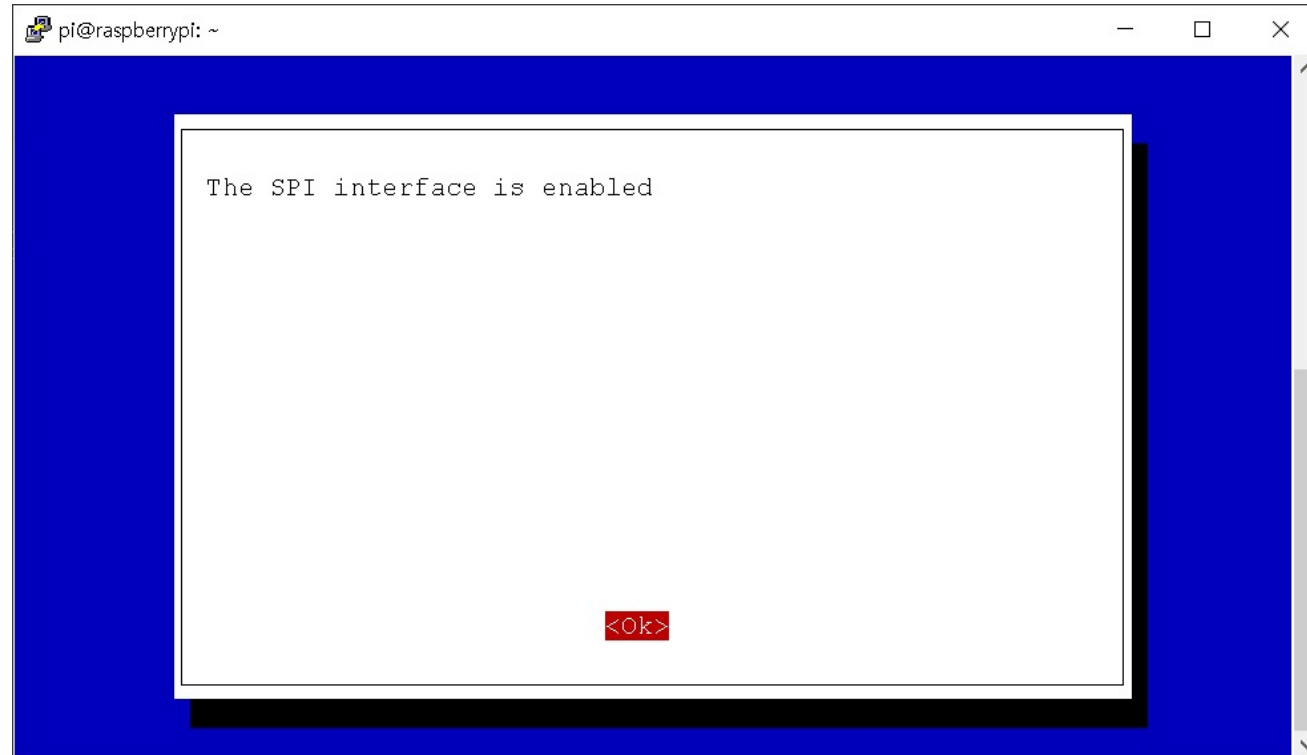
ADC(Analog-to-digital converting)

- SPI를 위한 Raspberry Pi setup
 - Raspberry Pi SPI module 활성화



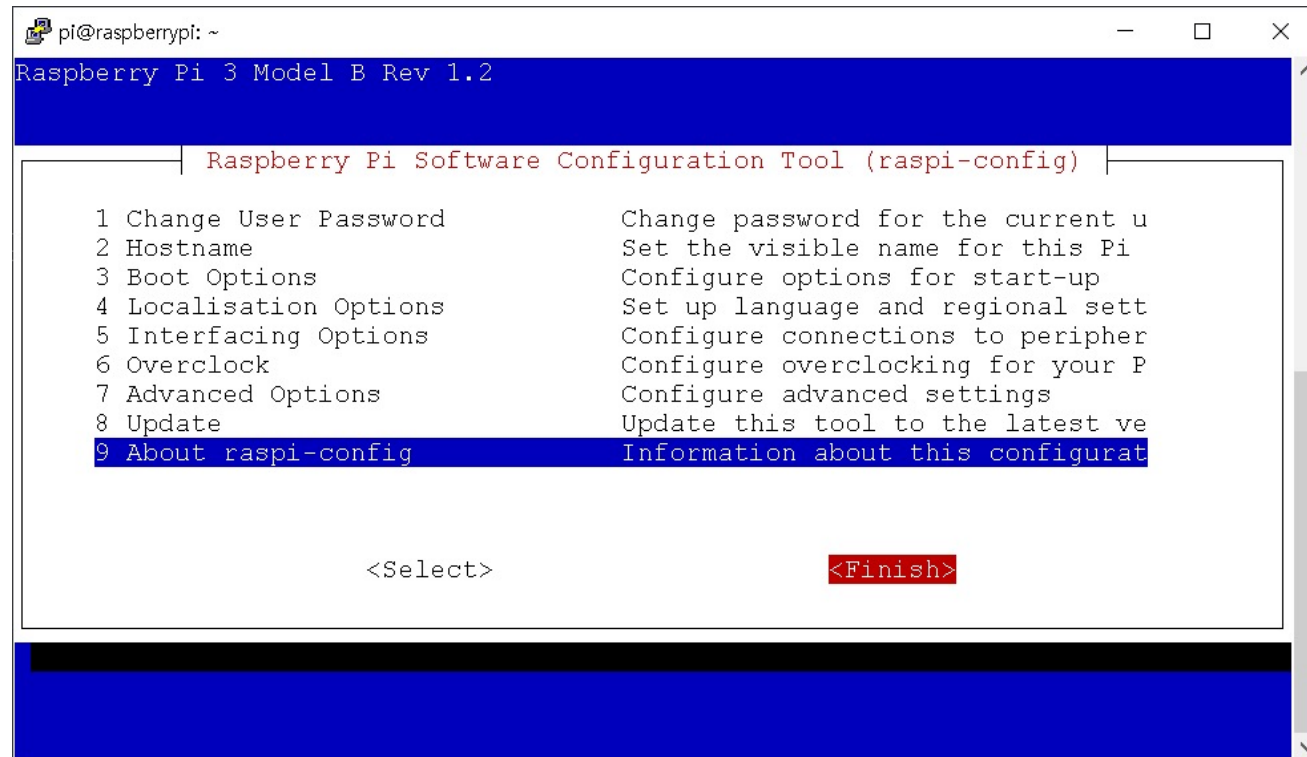
ADC(Analog-to-digital converting)

- SPI를 위한 Raspberry Pi setup
 - Raspberry Pi SPI module 활성화



ADC(Analog-to-digital converting)

- SPI를 위한 Raspberry Pi setup
 - Raspberry Pi SPI module 활성화

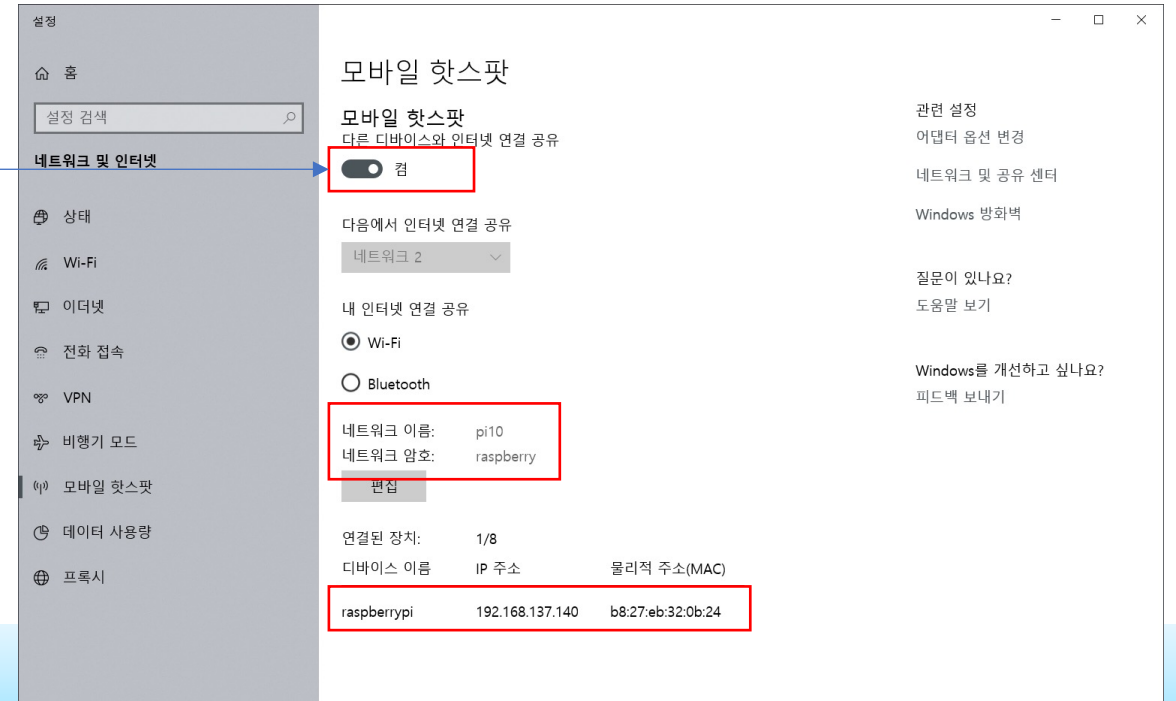
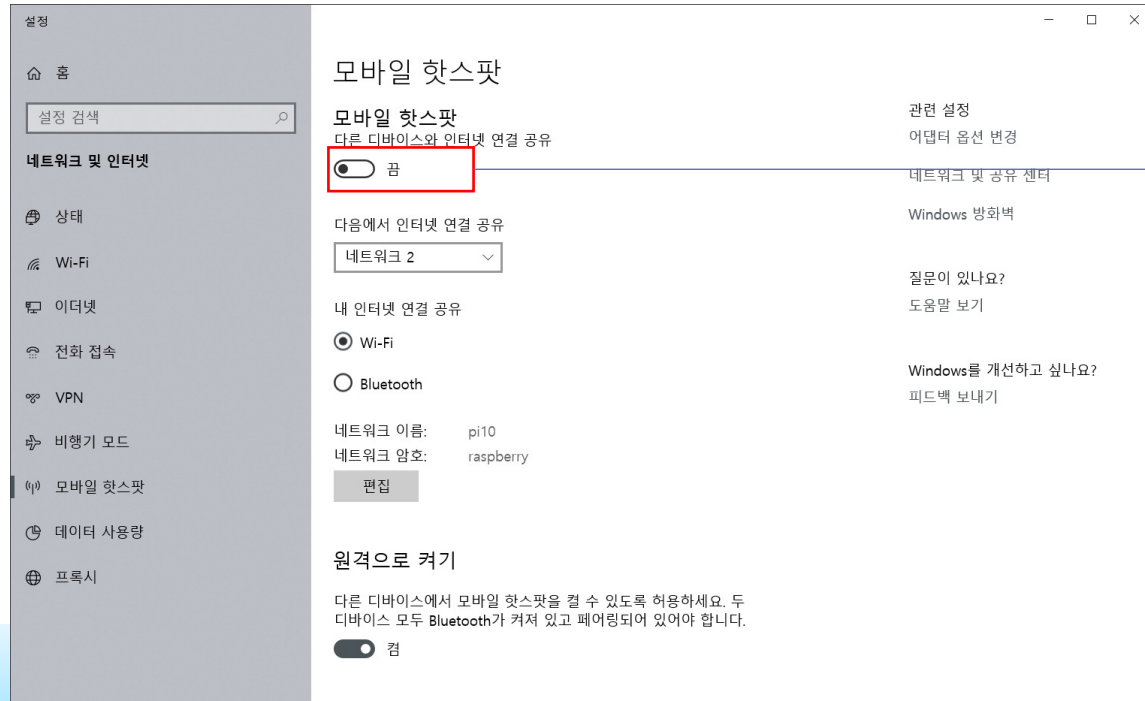


```
pi@raspberrypi: ~  
Raspberry Pi 3 Model B Rev 1.2  
  
Raspberry Pi Software Configuration Tool (raspi-config)  
  
1 Change User Password      Change password for the current u  
2 Hostname                  Set the visible name for this Pi  
3 Boot Options              Configure options for start-up  
4 Localisation Options      Set up language and regional sett  
5 Interfacing Options        Configure connections to peripher  
6 Overclock                 Configure overclocking for your P  
7 Advanced Options          Configure advanced settings  
8 Update                    Update this tool to the latest ve  
9 About raspi-config         Information about this configurat  
  
<Select>          <Finish>
```


참고 사항

• 수업 전 확인사항

- 무선랜 카드를 PC에 설치 및 SD card를 Raspberry Pi에 삽입
- PC의 모바일 핫스팟을 **컴**으로 설정
 - Raspberry Pi의 전원을 켜
- 네트워크 이름 및 네트워크 암호 설정 확인
- 연결된 장치의 IP 주소 확인



참고 사항

- Raspberry Pi 끝 때

sudo shutdown -t now



Superuser 권한으로 명하니 Raspberry Pi를 꺼라 언제? 지금 당장!