SPI 통신을 활용한 예제 및 실습

- SPI 통신을 활용한 제어

엣지아이랩

SPI 통신을 활용한 제어

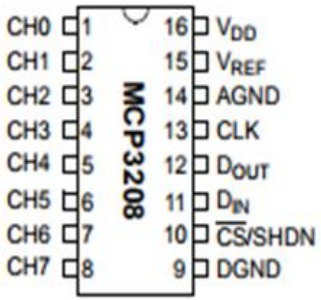
SPI 통신을 활용한 제어

- ADC 칩 MCP3208
  - 아날로그 출력 데이터를 디지털 값으로 변경
  - 라즈베리파이와 SPI로 연결
  - 8채널 12비트 아날로그 디지털 컨버터(ADC)
    - 0~3.3V 값을 0~4095(12bit)의 디지털 값으로 변환
  - 라즈베리파이에는 ADC 기능이 없음

SPI 통신을 활용한 제어

- ADC 칩 MCP3208

PDIP, SOIC



Name	Function
V_{DD}	+2.7V to 5.5V Power Supply
DGND	Digital Ground
AGND	Analog Ground
CH0-CH7	Analog Inputs
CLK	Serial Clock
D_{IN}	Serial Data In
D_{OUT}	Serial Data Out
$\overline{CS}$/SHDN	Chip Select/Shutdown Input
V_{REF}	Reference Voltage Input

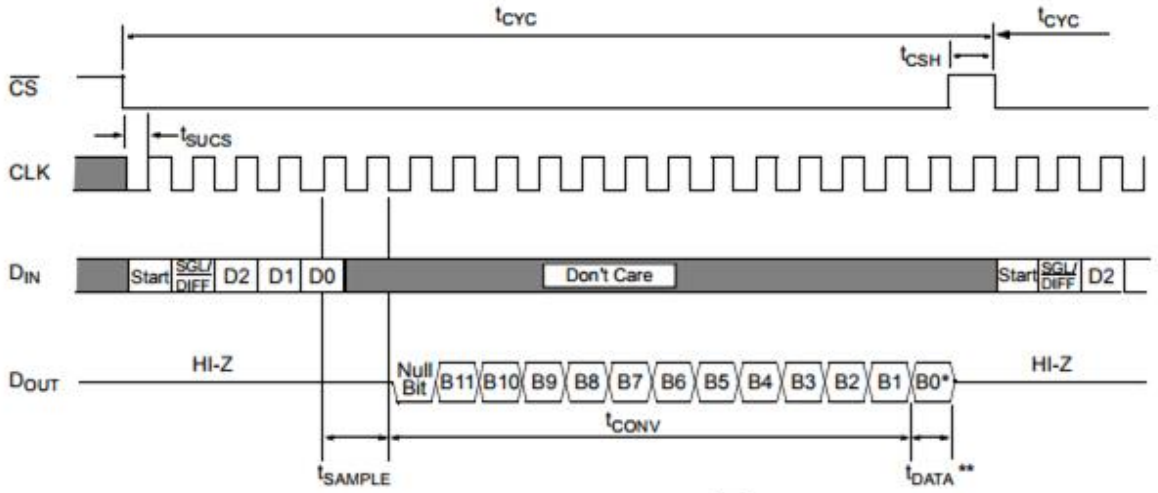
  - DGND / AGND
    - 내부 디지털 / 아날로그 회로의 접지 연결
  - CH0 ~ CH7
    - 멀티 플렉스 입력에 대한 채널 0~7의 아날로그 입력
    - 각 채널의 쌍은 단일 종단형 모드(Single-ended Mode)에서 두 개의 독립 채널로 사용
    - 하나의 채널이 IN+이고, 하나의 채널이 IN인 단일 반 차동 입력(Single pseudo-differential input)으로 사용되도록 프로그래밍 가능

SPI 통신을 활용한 제어

- ADC 칩 MCP3208
  - Serial Clock
    - SPI 클럭 핀은 변환을 초기화하거나, 발생하는 변환의 각 비트를 클럭 아웃(Clock Out)하는데 사용
  - Serial Data Input
    - SPI 포트 직렬 데이터 입력 핀은 장치의 채널 구성 데이터를 불러오는데 사용
  - Serial Data Output
    - SPI 직렬 데이터 출력 핀은 A/D 변환 결과를 옮기는데 사용
    - 변환이 일어날 때 마다 데이터는 항상 각 클럭의 하강 엣지에서 변경
  - CS / SHDN
    - CS/SHDN 핀은 Low로 끌어 올 때 장치와의 통신을 시작하는데 사용
    - High로 끌어 올 때 저전력 대기 상태로 전환
    - CS/SHDN 핀은 변환 사이에 High로 Pulling되어야 함

SPI 통신을 활용한 제어

- ADC 칩 MCP3208
  - 시리얼 통신(Serial Communication)
    - MCP3208 장치와의 통신은 표준SPI 호환 직렬 인터페이스를 사용하여 통신
    - CS 핀을 LOW로 하여 디바이스의 전원을 켜면 통신을 시작하기 위해 다시 HIGH로 되돌아와야 함
    - CS의 LOW와 DIN의 HIGH로 수신된 CS첫번째 클럭은 시작 비트를 구성
    - SGL/DIFF 비트는 시작 비트 다음에 오며, 변환이 단일 종단(Single –ended) 나 차동 입력 모드(Differential Input Mode)를 사용할지에 대해 결정



* After completing the data transfer, if further clocks are applied with $\overline{CS}$ low, the A/D converter will output LSB first data, followed by zeros indefinitely (see Figure 5-2 below).

** t_{DATA}: during this time, the bias current and the comparator power down while the reference input becomes a high impedance node, leaving the CLK running to clock out the LSB-first data or zeros.

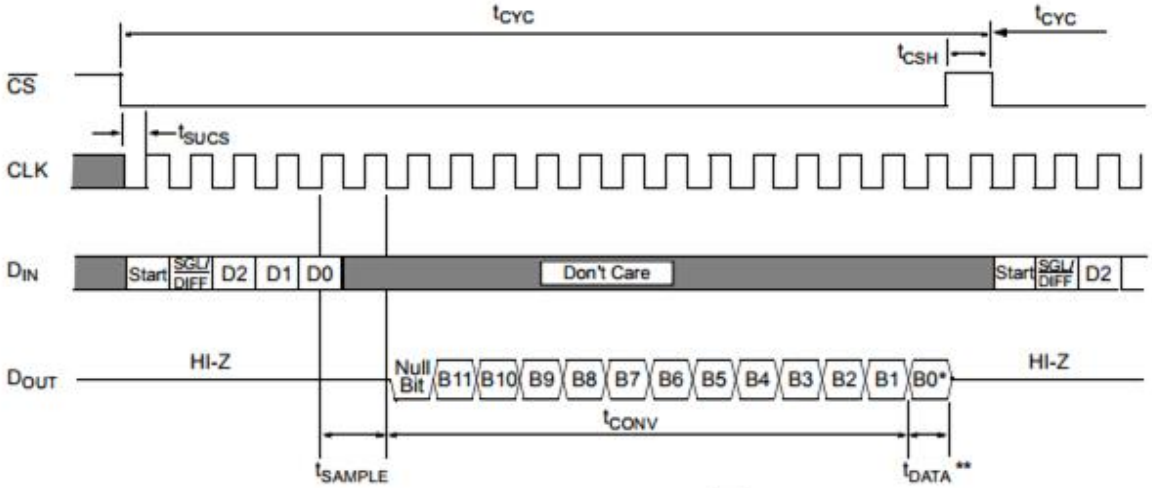
SPI 통신을 활용한 제어

- ADC 칩 MCP3208
  - 시리얼 통신(Serial Communication)
    - 다음 세 비트(D0, D1, D2)는 입력 채널 구성을 선택하는데 사용

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+

SPI 통신을 활용한 제어

- ADC 칩 MCP3208
  - 시리얼 통신(Serial Communication)
    - 디바이스는 시작 비트가 수신 된 후, 클럭의 네번째 상승 엣지에서 아날로그 입력을 샘플링하기 시작
    - 샘플 기간은 시작 비트 다음 5번째 클럭의 하강 엣지에서 완료
    - D0 비트가 입력되면 Sample & Hold 기간을 완료하기 위해 한번 더 클릭이 요구됨
    - 다음 클럭의 하강 엣지에서 디바이스는 낮은 null 비트를 출력
    - 그 다음 12 클럭은 아래와 같이 MSB로 변환한 결과를 출력



* After completing the data transfer, if further clocks are applied with $\overline{CS}$ low, the A/D converter will output LSB first data, followed by zeros indefinitely (see Figure 5-2 below).

** t_{DATA}: during this time, the bias current and the comparator power down while the reference input becomes a high impedance node, leaving the CLK running to clock out the LSB-first data or zeros.

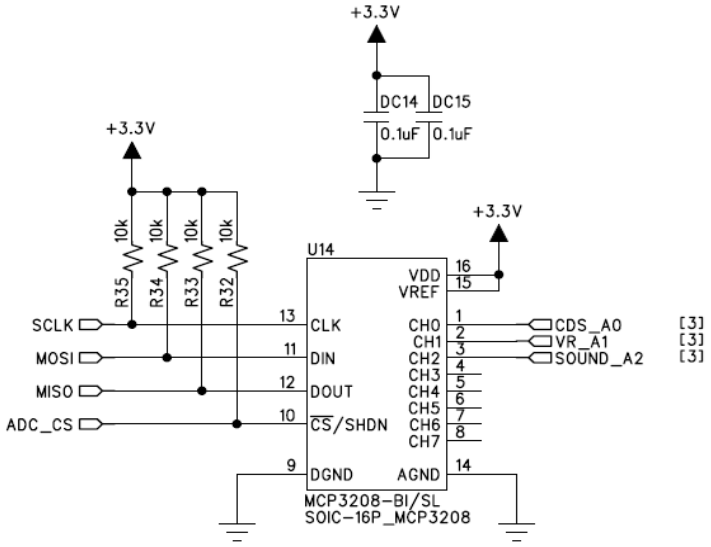
SPI 통신을 활용한 제어

- ADC 칩 MCP3208
  - 통신 예시
  - 라즈베리파이가 마스터이고, MCP3208이 슬레이브인 경우, 마스터에서 정해진 명령어를 슬레이브에게 전송하면 그에 해당되는 값이 전송
  - 송수신할 때에는 3byte를 이용하고, 이 3byte의 내용은 아래와 같음
  - 'x'는 어떠한 값도 상관없는 "Don't care"를 의미
  - 이렇게 3byte의 데이터를 보냄과 동시에 MCP3208로부터 3byte의 데이터를 받음
  - 그 데이터의 마지막 bit로부터 12비트가 ADC 결과값

비트	7	6	5	4	3	2	1	0
1 Byte	0	0	0	0	0	Start	SGL/Diff	D2
2Byte	D1	D0	X	X	X	X	X	X
3Byte	X	X	X	X	X	X	X	X

SPI 통신을 활용한 제어

- Edge-Embedded의 SPI
  - 라즈베리파이의 SPI 통신을 이용하여 Edge-Embedded(CE0)의 EEPROM과 직접 또는 MCP3208 칩(CE1)을 통해 간접적으로 연결
  - Edge-Embedded에 부착된 MCP3208 칩에는 CdS, VR, Sound 센서가 CH0~CH2 번과 연결

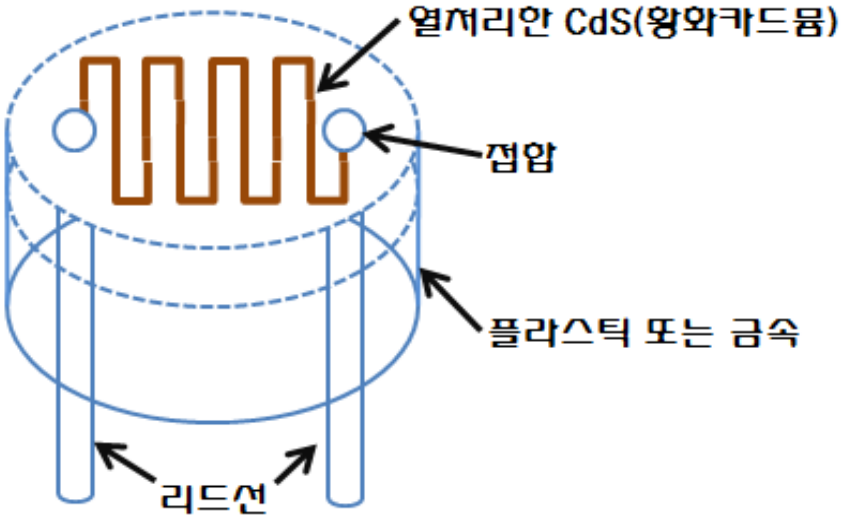


Connect	BCM	Name	Physical		Name	BCM	Connect
		+3.3V	1	2	+5V		
I2C	GPIO 02	SDA1	3	4	+5V		
I2C	GPIO 03	SCL1	5	6	GND		
DC_M	GPIO 04	GPIO	7	8	TxD	GPIO 14	UART
		GND	9	10	RxD	GPIO 15	UART
SERVO	GPIO 17	GPIO	11	12	GPIO	GPIO 18	TLCD
TLCD	GPIO 27	GPIO	13	14	GND		
TLCD	GPIO 22	GPIO	15	16	GPIO	GPIO 23	TLCD
		+3.3V	17	18	GPIO	GPIO 24	PIR
SPI	GPIO 10	MOSI	19	20	GND		
SPI	GPIO 09	MISO	21	22	GPIO	GPIO 25	DC_M
SPI	GPIO 11	SCLK	23	24	CE0	GPIO 08	SPI
		GND	25	26	CE1	GPIO 07	SPI
ULTRA	GPIO 00	SDA0	27	28	SCL0	GPIO 01	ULTRA
SWITCH	GPIO 05	GPIO	29	30	GND		
SWITCH	GPIO 06	GPIO	31	32	GPIO	GPIO 12	DC_M
PIEZO	GPIO 13	GPIO	33	34	GND		
IR	GPIO 19	GPIO	35	36	GPIO	GPIO 16	TLCD
TLCD	GPIO 26	GPIO	37	38	GPIO	GPIO 20	LED
		GND	39	40	GPIO	GPIO 21	LED

EEPROM
MCP3208

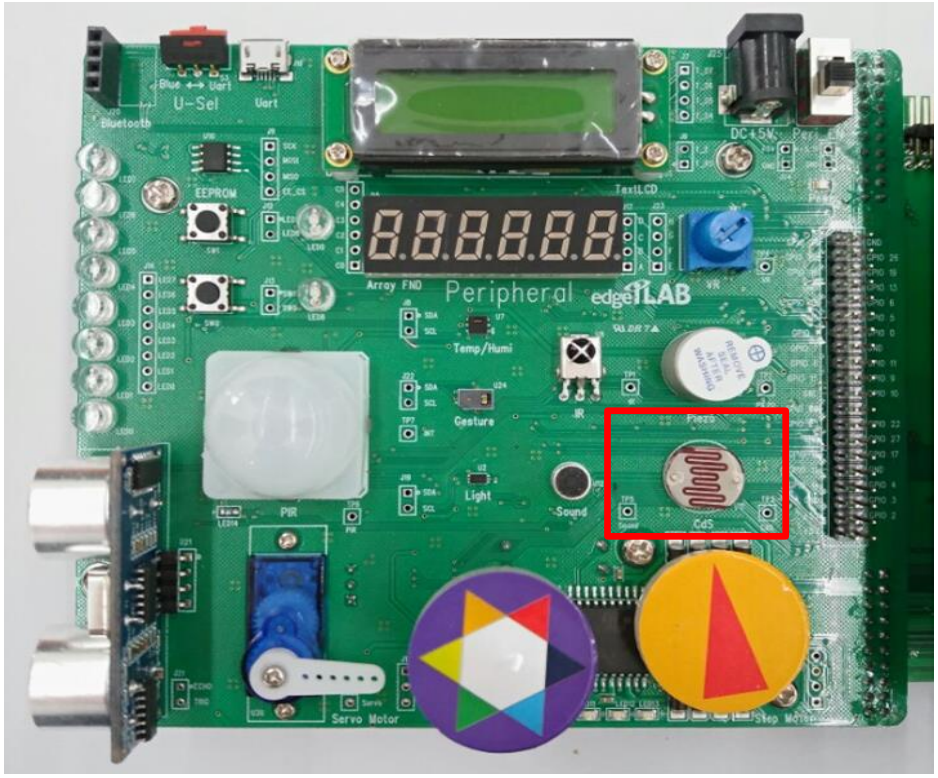
SPI 통신을 활용한 제어

- CdS(조도 센서) 제어
  - 조도센서(Photo Resistor)는 주변의 밝기를 측정하는 센서
  - 광 에너지(빛)을 받으면 내부에 움직이는 전자가 발생하여 전도율이 변하는 광전 효과를 가지는 소자를 사용
  - 황화카드뮴(CdS)를 소자로 사용하여, CdS 센서라고 함
  - CdS 센서는 작고 저렴하므로 어두워지면 자동으로 켜지는 가로등, 자동차의 헤드라이트 등 실생활에서 보편적으로 많이 사용



SPI 통신을 활용한 제어

- CdS(조도 센서) 제어
  - 조도센서는 극성은 없으나 빛의 양에 따라 전도율이 변하며, 전도율에 따라 저항이 변함
  - 즉, 빛의 양이 많은 곳일수록 내부 저항 값이 작아져 전류가 증가
  - 이러한 전도율이 밝기에 비례하지만, 선형적으로 증가하는 것이 아니기 때문에 밝고 어두운 정도만을 판별하기에는 적합



SPI 통신을 활용한 제어

- CdS(조도 센서) 제어 예제(1)
  - 터미널 창에 "nano 14_CDS_01.c" 입력

```
pi@raspberryPi:~/Example $ nano 14_CDS_01.c
```

  - 현재의 광량을 측정하는 예제

```
1.  // File : 14_CDS_01.c

2.  #include <stdio.h>
3.  #include <wiringPi.h>
4.  #include <wiringPiSPI.h>                    // SPI 라이브러리 참조

5.  #define CS_ADC              7              // ADC 칩 번호 설정
6.  #define SPI_CHANNEL         0              // SPI 통신 설정
7.  #define SPI_SPEED           1000000

8.  // 아날로그 Read 함수 정의
9.  int adcRead(char adcChannel)
10. {
11.     char buff[3];                          // 송수신용 데이터 변수 선언

12.     int adcValue = 0;
```

SPI 통신을 활용한 제어

- CdS(조도 센서) 제어 예제(1)

```
13.     // CdS가 연결된 채널은 CH0
14.     // 채널을 비트화 후 Shift 연산자를 통해 D2만 0번째 비트에 남겨둠
15.     // 0x60(Start 비트 : 1, Single-Endded 모드 비트 : 1)을 OR 연산
16.     buff[0] = 0x06 | ((adcChannel & 0x07) >> 2);

17.     // 채널을 비트화 후 Shift 연산자를 통해 D1, D0만 7, 6번째 비트에 남겨둠
18.     buff[1] = ((adcChannel & 0x07) << 6);

19.     buff[2] = 0x00;

20.     digitalWrite(CS_ADC, LOW);

21.     // SPI 통신을 통해 3byte 송수신
22.     // Single Ended 모드, CH0 전송
23.     // 2번째 Byte의 0~3비트, 3번째 Byte의 0~7비트에 ADC 결과값 저장
24.     wiringPiSPIDataRW(SPI_CHANNEL, buff, 3);

25.     // Mask(0x0F)를 이용해 2번째 Byte의 0~3비트 저장
26.     buff[1] = 0x0F & buff[1];

27.     // 2~3번째 Byte를 이용해 결과값 저장
28.     adcValue = (buff[1] << 8) | buff[2];
```

SPI 통신을 활용한 제어

- CdS(조도 센서) 제어 예제(1)

```
29.     digitalWrite(CS_ADC,HIGH);

30.     return adcValue;
31. }

32. int main(void)
33. {
34.     int adcValue_CDS = 0;                    // ADC 한 결과값 저장 변수 초기화
35.     wiringPiSetupGpio();                     // 핀 번호를 BCM Mode로 설정

36.     // SPI 시스템 초기화 설정
37.     if(wiringPiSPISetup(SPI_CHANNEL, SPI_SPEED) < 0 )
38.     {
39.         return -1;
40.     }

41.     pinMode(CS_ADC, OUTPUT);

42.     while(1)
43.     {
44.         adcValue_CDS = adcRead(0);
45.         printf("CDS = %u₩n", adcValue_CDS);
46.         delay(1000);
```

SPI 통신을 활용한 제어

- CdS(조도 센서) 제어 예제(1)

```
47.     }
48.     return 0;
49. }
```

  - 작성 후 "crtl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
  - GCC 컴파일러를 사용하여 빌드 및 생성된 "14_CDS_01" 파일 실행

```
pi@raspberryPi:~/Example $ gcc -o 14_CDS_01 14_CDS_01.c -lwiringPi
pi@raspberryPi:~/Example $ ./14_CDS_01
```

  - 결과
    - 현재 광량을 측정하여 터미널 화면에 출력

```
CDS = 3536
CDS = 3524
CDS = 1083
CDS = 903
CDS = 3520
```

SPI 통신을 활용한 제어

- CdS(조도 센서) 제어 예제(2)
  - 터미널 창에 "nano 14_CDS_02.c" 입력

```
pi@raspberryPi:~/Example $ nano 14_CDS_02.c
```

  - 현재의 광량을 측정하고 측정 값에 따라 8LED로 표시하는 예제

```
1.  // File : 14_CDS_02.c

2.  #include <stdio.h>
3.  #include <wiringPi.h>
4.  #include <wiringPiSPI.h>                    // SPI 라이브러리 참조
5.  #include <wiringPiI2C.h>                    // I2C 라이브러리 참조

6.  #define CS_ADC              7              // ADC 칩 번호 설정
7.  #define SPI_CHANNEL         0              // SPI 통신 설정
8.  #define SPI_SPEED           1000000

9.  // I2C 레지스터 설정
10. #define LED_I2C_ADDR     0x20
11. #define OUT_PORT1        0x03
12. #define CONFIG_PORT1     0x07

13. // 8LED 출력 데이터 설정
14. const int aLedData[8] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};
```

SPI 통신을 활용한 제어

- CdS(조도 센서) 제어 예제(2)

```
15.  int fd;                                          // 8LED의 handle

16.  // 아날로그 Read 함수 정의
17.  int adcRead(char adcChannel)
18.  {
19.      char buff[3];                                 // 송수신용 데이터 변수 선언

20.      int adcValue = 0;

21.      // CdS가 연결된 채널은 CH0
22.      // 채널을 비트화 후 Shift 연산자를 통해 D2만 0번째 비트에 남겨둠
23.      // 0x60(Start 비트 : 1, Single-Endded 모드 비트 : 1)을 OR 연산
24.      buff[0] = 0x06 | ((adcChannel & 0x07) >> 2);

25.      // 채널을 비트화 후 Shift 연산자를 통해 D1, D0만 7, 6번째 비트에 남겨둠
26.      buff[1] = ((adcChannel & 0x07) << 6);

27.      buff[2] = 0x00;

28.      digitalWrite(CS_ADC, LOW);
```

SPI 통신을 활용한 제어

- CdS(조도 센서) 제어 예제(2)

```
29.     // SPI 통신을 통해 3byte 송수신
30.     // Single Ended 모드, CH0 전송
31.     // 2번째 Byte의 0~3비트, 3번째 Byte의 0~7비트에 ADC 결과값 저장
32.     wiringPiSPIDataRW(SPI_CHANNEL, buff, 3);

33.     // Mask(0x0F)를 이용해 2번째 Byte의 0~3비트 저장
34.     buff[1] = 0x0F & buff[1];

35.     // 2~3번째 Byte를 이용해 결과값 저장
36.     adcValue = (buff[1] << 8) | buff[2];

37.     digitalWrite(CS_ADC,HIGH);

38.     return adcValue;
39. }

40. // 범위에 따른 LED점등 함수 정의
41. void ledOnByRange(int x, int min, int max, int ledNo)
42. {
43.     int i;
44.     int data = 0;                          //8LED 출력Data 저장변수 선언
45.     if( x < max && x >= min)               //변수 x가 min과 max 사이값일 경우
46.     {
```

SPI 통신을 활용한 제어

- CdS(조도 센서) 제어 예제(2)

```
47.         for(i=0; i< ledNo; i++)                    //ledNo 개수만큼 LED 점등
48.         {
49.             data = aLedData[i] | data;
50.         }
51.         wiringPiI2CWriteReg16(fd, OUT_PORT1, data);
52.     }
53. }

54. int main(void)
55. {
56.     int adcValue_CDS = 0;                          // ADC 한 결과값 저장 변수 초기화
57.     wiringPiSetupGpio();                           // 핀 번호를 BCM Mode로 설정

58.     // SPI 시스템 초기화 설정
59.     if(wiringPiSPISetup(SPI_CHANNEL, SPI_SPEED) < 0 )
60.     {
61.         return -1;
62.     }

63.     // I2C 시스템 초기화 설정
64.     if((fd = wiringPiI2CSetup(LED_I2C_ADDR)) < 0 )
65.     {
66.         return -1;
67.     }
```

SPI 통신을 활용한 제어

- CdS(조도 센서) 제어 예제(2)

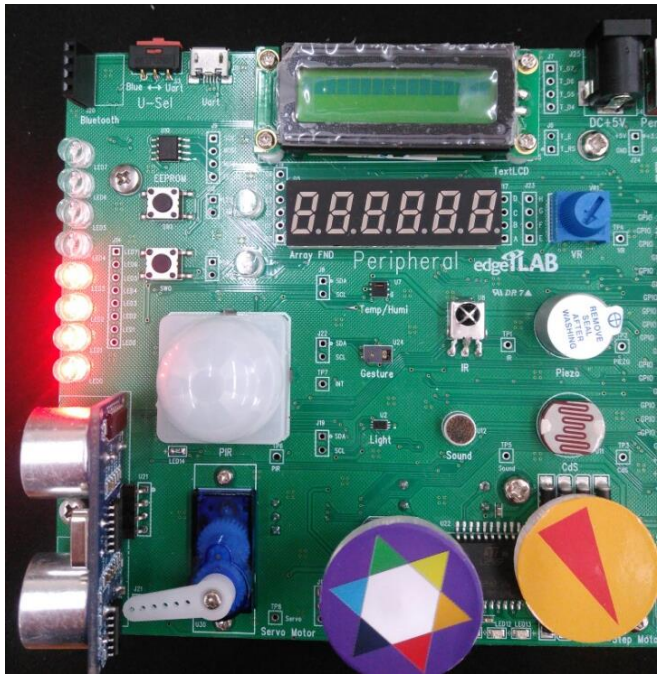
```
68.     pinMode(CS_ADC, OUTPUT);

69.     // handle을 통해 IO1 출력모드로 설정
70.     wiringPiI2CWriteReg16(fd, CONFIG_PORT1, 0x0000);

71.     while(1)
72.     {
73.         adcValue_CDS = adcRead(0);
74.         printf("CDS = %u₩n", adcValue_CDS);
75.         delay(100);

76.         // 3700 ~ 4000 값이면 LED 8개 On
77.         ledOnByRange(adcValue_CDS, 3700, 4000, 8);

78.         // 3200 ~ 3700 값이면 LED 4개 On
79.         ledOnByRange(adcValue_CDS, 3200, 3700, 4);

80.         // 900 ~ 3200 값이면 LED 1개 On
81.         ledOnByRange(adcValue_CDS, 900, 3200, 1);

82.     }
83.     return 0;
84. }
```

SPI 통신을 활용한 제어

- CdS(조도 센서) 제어 예제(2)
  - 작성 후 "crtl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
  - GCC 컴파일러를 사용하여 빌드 및 생성된 "14_CDS_02" 파일 실행

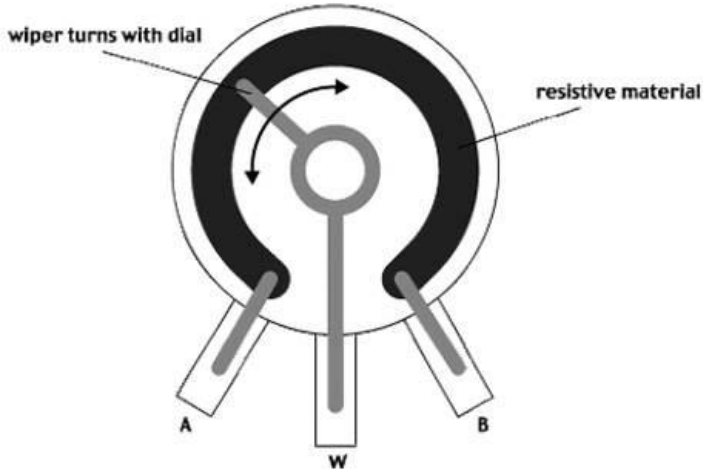
```
pi@raspberryPi:~/Example $ gcc -o 14_CDS_02 14_CDS_02.c -lwiringPi
pi@raspberryPi:~/Example $ ./14_CDS_02
```

  - 결과
    - 현재의 광량에 따라 LED의 점등 개수 변화



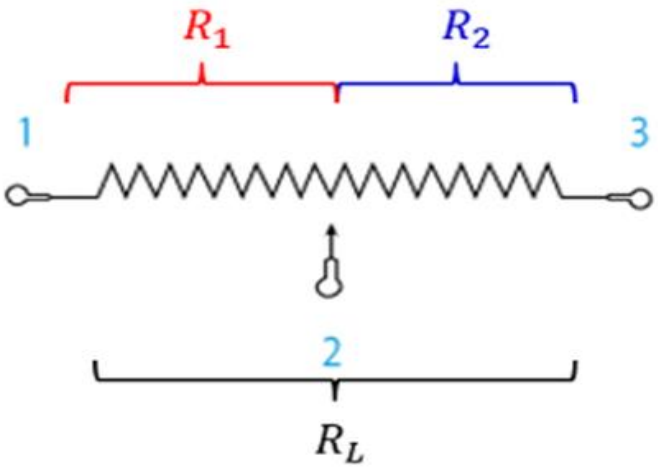
SPI 통신을 활용한 제어

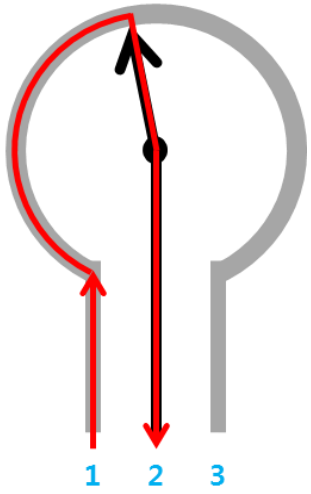
- VR(Variaable Resistor, 가변저항) 제어
  - 가변저항(Variable Resistor 혹은 Potentiometer)이란 사용자가 직접 저항 값을 임의로 바꿀 수 있는 저항기
  - 흔히 오디오 장비의 음량이나 동작 감지기의 감도, 밸런스 등을 조절할 때 사용
  - 가변저항은 사용하는 저항체의 종류에 따라 다양
  - 회전축을 중심으로 전극을 움직이거나, 좌우로 전극을 움직이는 방식으로 저항 값에 변화를 줌



SPI 통신을 활용한 제어

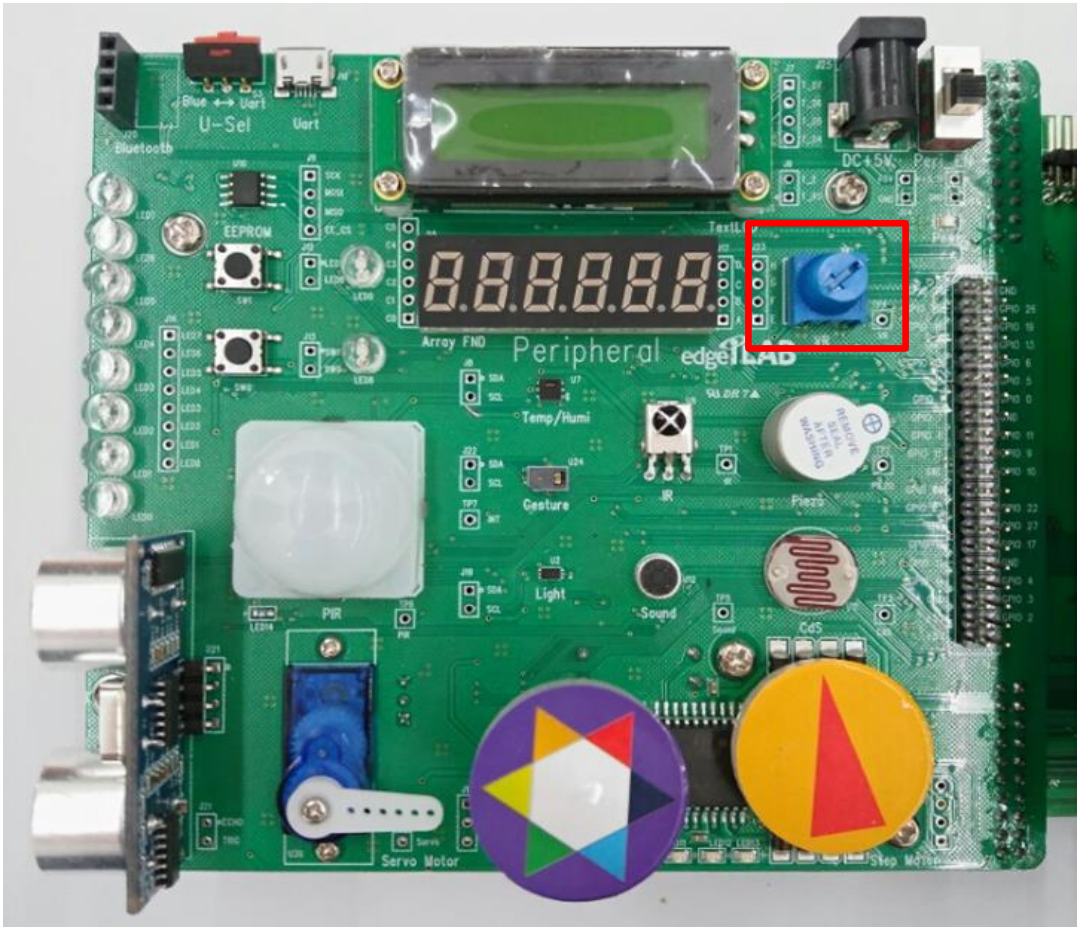
- VR(Variaable Resistor, 가변저항) 제어
  - 가변저항은 3개의 단자로 구성
  - 바깥 단자 2개는 내부 저항체의 양 끝에 연결
  - 이 저항체는 전도성 플라스틱 띠로 되어 있고, 이를 "트랙"이라고도 함
  - 중앙의 단자는 내부적으로 "와이퍼"라고 불리는 접점과 연결
  - 와이퍼는 띠와 접촉한 채로 축을 움직이면서 한쪽 끝에서 다른 쪽으로 이동
  - 저항체의 양쪽 끝 사이에 전위차가 걸려있으면 와이퍼가 움직이면서 와이퍼에 걸리는 전압이 변화
  - 이 때 가변저항은 저항 분압기와 같은 역할
  - 즉 양쪽 저항의 크기에 따라 전압 분배가 이루어짐





SPI 통신을 활용한 제어

- VR(Variaable Resistor, 가변저항) 제어
  - Edge-Embedded의 가변저항



SPI 통신을 활용한 제어

- VR(Variaable Resistor, 가변저항) 제어 예제(1)
  - 터미널 창에 "nano 15_VR_01.c" 입력

```
pi@raspberryPi:~/Example $ nano 15_VR_01.c
```

  - 현재 가변저항 값을 측정하는 예제

```
1.  // File : 15_VR_01.c

2.  #include <stdio.h>
3.  #include <wiringPi.h>
4.  #include <wiringPiSPI.h>                    // SPI 라이브러리 참조

5.  #define CS_ADC              7              // ADC 칩 번호 설정
6.  #define SPI_CHANNEL         0              // SPI 통신 설정
7.  #define SPI_SPEED           1000000

8.  // 아날로그 Read 함수 정의
9.  int adcRead(char adcChannel)
10. {
11.     char buff[3];                          // 송수신용 데이터 변수 선언

12.     int adcValue = 0;
```

SPI 통신을 활용한 제어

- VR(Variaable Resistor, 가변저항) 제어 예제(1)

```
13.     // VR이 연결된 채널은 CH1
14.     // 채널을 비트화 후 Shift 연산자를 통해 D2만 0번째 비트에 남겨둠
15.     // 0x60(Start 비트 : 1, Single-Endded 모드 비트 : 1)을 OR 연산
16.     buff[0] = 0x06 | ((adcChannel & 0x07) >> 2);

17.     // 채널을 비트화 후 Shift 연산자를 통해 D1, D0만 7, 6번째 비트에 남겨둠
18.     buff[1] = ((adcChannel & 0x07) << 6);

19.     buff[2] = 0x00;

20.     digitalWrite(CS_ADC, LOW);

21.     // SPI 통신을 통해 3byte 송수신
22.     // Single Ended 모드, CH0 전송
23.     // 2번째 Byte의 0~3비트, 3번째 Byte의 0~7비트에 ADC 결과값 저장
24.     wiringPiSPIDataRW(SPI_CHANNEL, buff, 3);

25.     // Mask(0x0F)를 이용해 2번째 Byte의 0~3비트 저장
26.     buff[1] = 0x0F & buff[1];

27.     // 2~3번째 Byte를 이용해 결과값 저장
28.     adcValue = (buff[1] << 8) | buff[2];
```

SPI 통신을 활용한 제어

- VR(Variaable Resistor, 가변저항) 제어 예제(1)

```
29.     digitalWrite(CS_ADC,HIGH);

30.     return adcValue;
31. }

32. int main(void)
33. {
34.     int adcValue_VR = 0;                    // ADC 한 결과값 저장 변수 초기화
35.     wiringPiSetupGpio();                    // 핀 번호를 BCM Mode로 설정

36.     // SPI 시스템 초기화 설정
37.     if(wiringPiSPISetup(SPI_CHANNEL, SPI_SPEED) < 0 )
38.     {
39.         return -1;
40.     }

41.     pinMode(CS_ADC, OUTPUT);

42.     while(1)
43.     {
44.         adcValue_VR = adcRead(1);
45.         printf("VR = %u₩n", adcValue_VR);
```

SPI 통신을 활용한 제어

- VR(Variaable Resistor, 가변저항) 제어 예제(1)

```
46.             delay(1000);
47.         }
48.         return 0;
49.     }
```

  - 작성 후 "crtl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
  - GCC 컴파일러를 사용하여 빌드 및 생성된 "15_VR_01" 파일 실행

```
pi@raspberryPi:~/Example $ gcc -o 15_VR_01 15_VR_01.c -lwiringPi
pi@raspberryPi:~/Example $ ./15_VR_01
```

  - 결과
    - 현재의 가변저항 값을 측정하여 터미널 화면에 출력

```
VR = 1582
VR = 1491
VR = 934
VR = 695
VR = 1375
```

SPI 통신을 활용한 제어

- VR(Variaable Resistor, 가변저항) 제어 예제(2)
  - 터미널 창에 "nano 15_VR_02.c" 입력

```
pi@raspberryPi:~/Example $ nano 15_VR_02.c
```

  - 현재 가변저항 값에 따라 Piezo의 음을 다르게 출력하는 예제

```
1.  // File : 15_VR_02.c

2.  #include <stdio.h>
3.  #include <wiringPi.h>
4.  #include <wiringPiSPI.h>                    // SPI 라이브러리 참조
5.  #include <softTone.h>                       // softTone 라이브러리 참조

6.  #define CS_ADC              7              // ADC 칩 번호 설정
7.  #define SPI_CHANNEL         0              // SPI 통신 설정
8.  #define SPI_SPEED           1000000

9.  const int pinPiezo = 13;                   // Piezo 핀 설정

10. // 아날로그 Read 함수 정의
11. int adcRead(char adcChannel)
12. {
13.     char buff[3];                          // 송수신용 데이터 변수 선언
```

SPI 통신을 활용한 제어

- VR(Variaable Resistor, 가변저항) 제어 예제(2)

```
14.     int adcValue = 0;

15.     // VR이 연결된 채널은 CH1
16.     // 채널을 비트화 후 Shift 연산자를 통해 D2만 0번째 비트에 남겨둠
17.     // 0x60(Start 비트 : 1, Single-Endded 모드 비트 : 1)을 OR 연산
18.     buff[0] = 0x06 | ((adcChannel & 0x07) >> 2);

19.     // 채널을 비트화 후 Shift 연산자를 통해 D1, D0만 7, 6번째 비트에 남겨둠
20.     buff[1] = ((adcChannel & 0x07) << 6);

21.     buff[2] = 0x00;

22.     digitalWrite(CS_ADC, LOW);

23.     // SPI 통신을 통해 3byte 송수신
24.     // Single Ended 모드, CH0 전송
25.     // 2번째 Byte의 0~3비트, 3번째 Byte의 0~7비트에 ADC 결과값 저장
26.     wiringPiSPIDataRW(SPI_CHANNEL, buff, 3);

27.     // Mask(0x0F)를 이용해 2번째 Byte의 0~3비트 저장
28.     buff[1] = 0x0F & buff[1];
```

SPI 통신을 활용한 제어

- VR(Variaable Resistor, 가변저항) 제어 예제(2)

```
29.     // 2~3번째 Byte를 이용해 결과값 저장
30.     adcValue = (buff[1] << 8) | buff[2];

31.     digitalWrite(CS_ADC,HIGH);

32.     return adcValue;
33. }

34. int main(void)
35. {
36.     int adcValue_VR = 0;                    // ADC 한 결과값 저장 변수 초기화
37.     wiringPiSetupGpio();                    // 핀 번호를 BCM Mode로 설정

38.     // SPI 시스템 초기화 설정
39.     if(wiringPiSPISetup(SPI_CHANNEL, SPI_SPEED) < 0 )
40.     {
41.         return -1;
42.     }

43.     pinMode(CS_ADC, OUTPUT);

44.     softToneCreate(pinPiezo);               // 해당 핀을 Tone 핀으로 설정
```

SPI 통신을 활용한 제어

- VR(Variaable Resistor, 가변저항) 제어 예제(2)

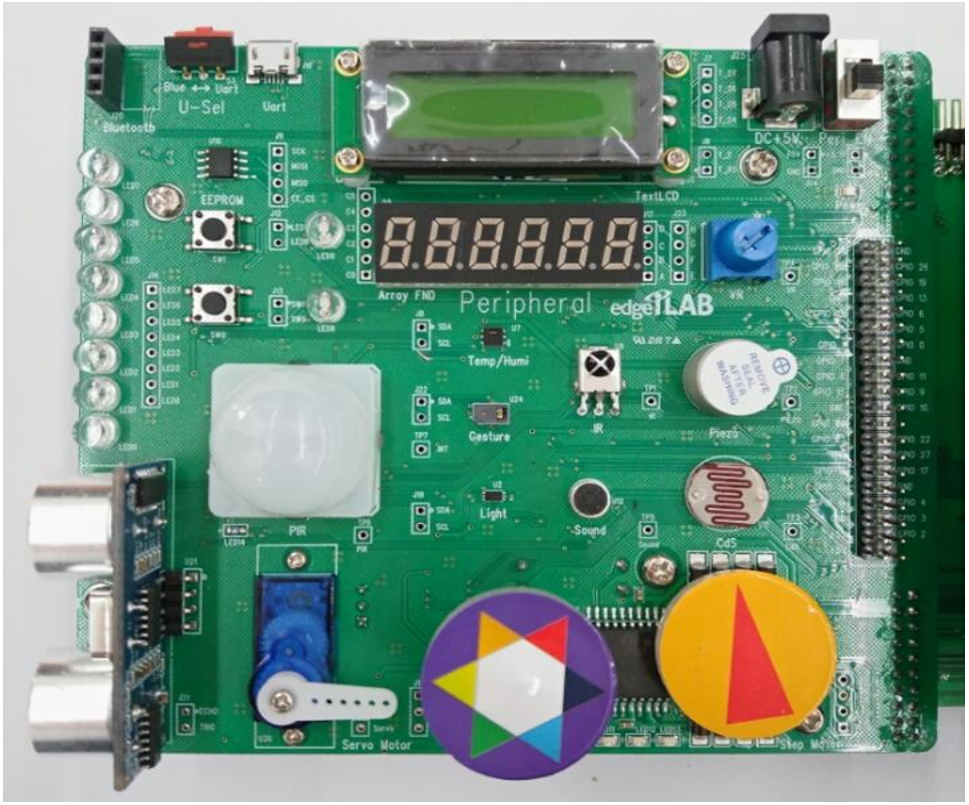
```
45.     while(1)
46.     {
47.         adcValue_VR = adcRead(1);
48.         printf("VR = %u₩n", adcValue_VR);

49.         softToneWrite(pinPiezo, adcValue_VR);
50.         delay(100);
51.     }
52.     return 0;
53. }
```

- 작성 후 "crtl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
- GCC 컴파일러를 사용하여 빌드 및 생성된 "15_VR_02" 파일 실행

```
pi@raspberryPi:~/Example $ gcc -o 15_VR_02 15_VR_02.c -lwiringPi
pi@raspberryPi:~/Example $ ./15_VR_02
```

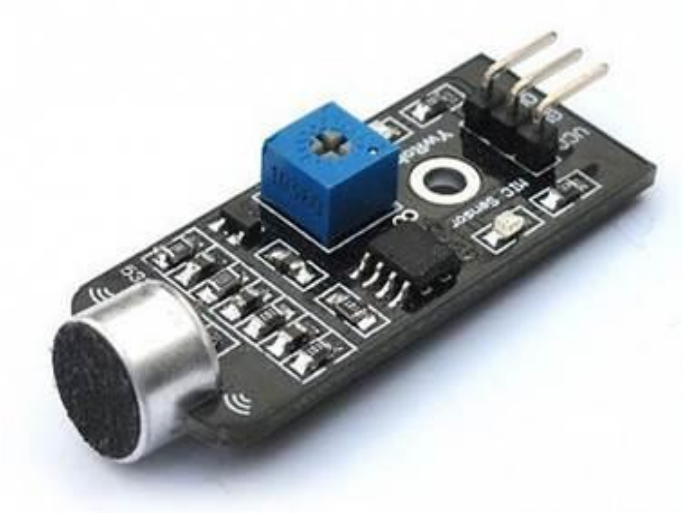
SPI 통신을 활용한 제어

- VR(Variaable Resistor, 가변저항) 제어 예제(2)
  - 결과
    - VR값에 따라 음(Pitch)을 출력



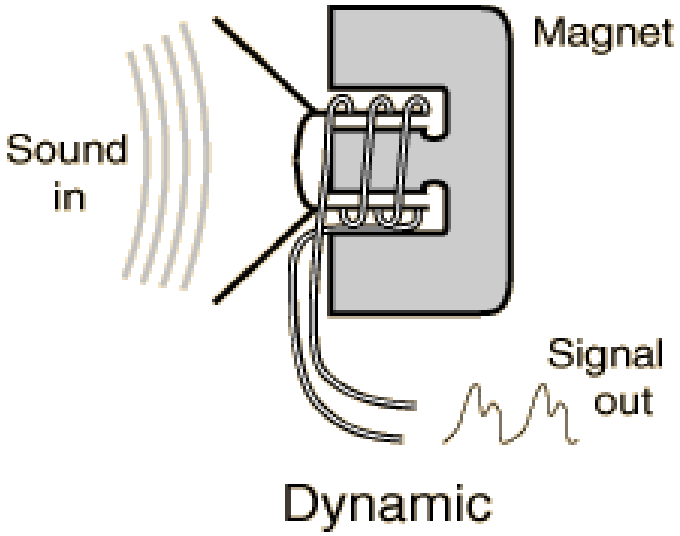
SPI 통신을 활용한 제어

- SOUND 제어
  - 사운드(음량) 감지 센서는 주변에서 발생하는 소리를 마이크로 모아 그 크기를 측정하는 센서
  - 여기서 소리는 공기의 진동을 통해서 전달되는 파동이며 진동이 공기를 통해 우리의 청각기관인 귀를 자극해서 '듣게'되는 것
  - 우리가 알고 있는 마이크와 같이 소리를 전기적 신호로 변환하여 입력 받는 원리
  - 소리를 전기적 신호로 변환하는 많은 방법은 다양함



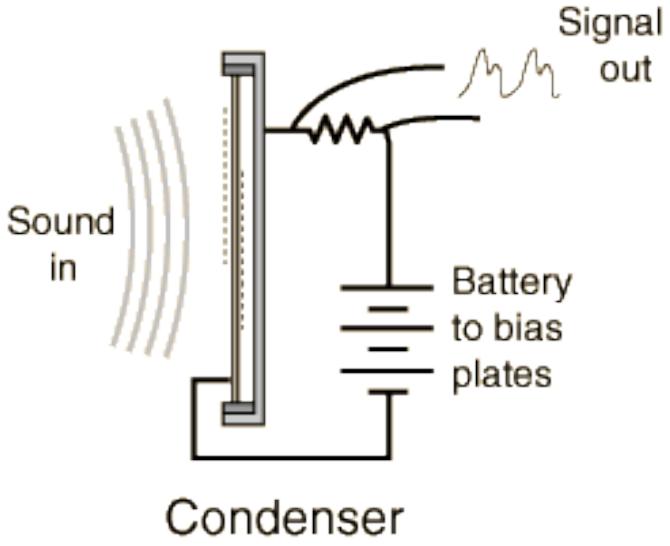
SPI 통신을 활용한 제어

- SOUND 제어
  - 다이나믹 방식
    - 다이나믹 방식은 발전기와 원리가 비슷함
    - 진동판에 연결된 코일이 소리에 의해 자석 사이에서 상-하로 움직이게 되면 "전자기유도 법칙"에 의해 전자운동이 생기며 전기적 신호가 발생
    - 이때, 코일이 얼마나 움직이냐에 따라 전기적 신호가 변하게 되고 변하는 정도에 따라 소리의 크기나 음의 높낮이 등을 알 수 있는 원리



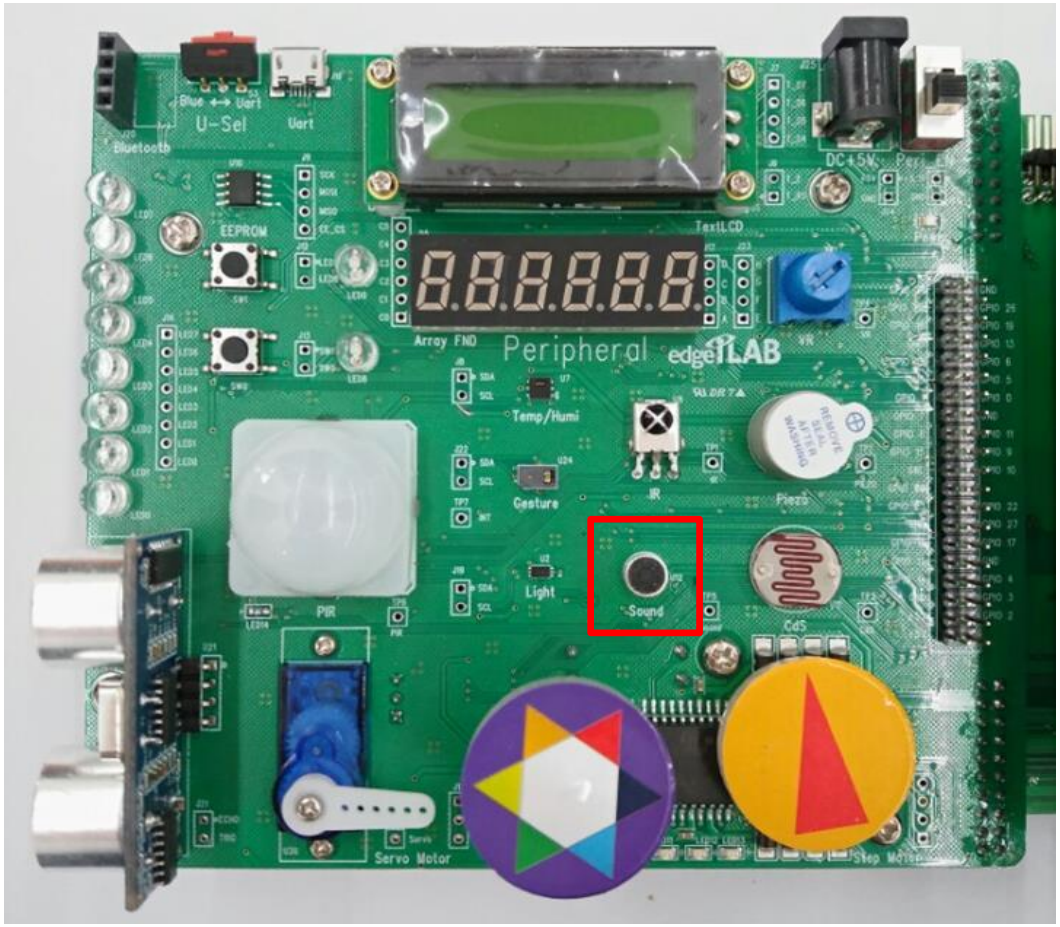
SPI 통신을 활용한 제어

- SOUND 제어
  - 콘덴서 방식
    - 콘덴서 방식은 축전기의 원리와 비슷함
    - 진동판이 소리에 의해 움직이게 되면 고정된 전극판과의 거리차에 의해 축전되는 전류량이 변화함
    - 축전기의 원리에 의해 두 전극판이 가까우면 발생하는 전기에너지가 커지고, 두 전극판이 멀어지면 전기에너지가 작아짐
    - 소리에 따라 전극판의 거리변화가 생기게 되고, 이 변화에 따라 음향의 차이를 확인하는 원리



SPI 통신을 활용한 제어

- SOUND 제어
  - Edge-Embedded의 Sound 센서



SPI 통신을 활용한 제어

- SOUND 제어 예제(1)
  - 터미널 창에 "nano 16_SOUND_01.c" 입력

```
pi@raspberryPi:~/Example $ nano 16_SOUND_01.c
```

  - 현재의 음량을 측정하는 예제

```
1.  // File : 16_SOUND_01.c

2.  #include <stdio.h>
3.  #include <wiringPi.h>
4.  #include <wiringPiSPI.h>                    // SPI 라이브러리 참조

5.  #define CS_ADC              7              // ADC 칩 번호 설정
6.  #define SPI_CHANNEL         0              // SPI 통신 설정
7.  #define SPI_SPEED           1000000

8.  // 아날로그 Read 함수 정의
9.  int adcRead(char adcChannel)
10. {
11.     char buff[3];                           // 송수신용 데이터 변수 선언

12.     int adcValue = 0;
```

SPI 통신을 활용한 제어

- SOUND 제어 예제(1)

```
13.     // Sound가 연결된 채널은 CH2
14.     // 채널을 비트화 후 Shift 연산자를 통해 D2만 0번째 비트에 남겨둠
15.     // 0x60(Start 비트 : 1, Single-Endded 모드 비트 : 1)을 OR 연산
16.     buff[0] = 0x06 | ((adcChannel & 0x07) >> 2);

17.     // 채널을 비트화 후 Shift 연산자를 통해 D1, D0만 7, 6번째 비트에 남겨둠
18.     buff[1] = ((adcChannel & 0x07) << 6);

19.     buff[2] = 0x00;

20.     digitalWrite(CS_ADC, LOW);

21.     // SPI 통신을 통해 3byte 송수신
22.     // Single Ended 모드, CH0 전송
23.     // 2번째 Byte의 0~3비트, 3번째 Byte의 0~7비트에 ADC 결과값 저장
24.     wiringPiSPIDataRW(SPI_CHANNEL, buff, 3);

25.     // Mask(0x0F)를 이용해 2번째 Byte의 0~3비트 저장
26.     buff[1] = 0x0F & buff[1];

27.     // 2~3번째 Byte를 이용해 결과값 저장
28.     adcValue = (buff[1] << 8) | buff[2];
```

SPI 통신을 활용한 제어

- SOUND 제어 예제(1)

```
29.     digitalWrite(CS_ADC,HIGH);

30.     return adcValue;
31. }

32. int main(void)
33. {
34.     int adcValue_SOUND = 0;                    // ADC 한 결과값 저장 변수 초기화
35.     wiringPiSetupGpio();                       // 핀 번호를 BCM Mode로 설정

36.     // SPI 시스템 초기화 설정
37.     if(wiringPiSPISetup(SPI_CHANNEL, SPI_SPEED) < 0 )
38.     {
39.         return -1;
40.     }

41.     pinMode(CS_ADC, OUTPUT);

42.     while(1)
43.     {
44.         adcValue_SOUND = adcRead(2);
45.         printf("SOUND = %u₩n", adcValue_SOUND);
46.         delay(1000);
```

SPI 통신을 활용한 제어

- SOUND 제어 예제(1)

```
47.     }
48.     return 0;
49. }
```

  - 작성 후 "crtl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
  - GCC 컴파일러를 사용하여 빌드 및 생성된 "16_SOUND_01" 파일 실행

```
pi@raspberryPi:~/Example $ gcc -o 16_SOUND_01 16_SOUND_01.c -lwiringPi
pi@raspberryPi:~/Example $ ./16_SOUND_01
```

  - 결과
    - 현재의 음량을 측정하여 터미널 화면에 출력
    - 사람의 소리보다 낮은 주파수의 소리에서 인식률일 높음

```
SOUND = 1865
SOUND = 2459
SOUND = 2061
SOUND = 1726
SOUND = 1814
SOUND = 2061
```

SPI 통신을 활용한 제어

- SOUND 제어 예제(2)
  - 터미널 창에 "nano 16_SOUND_02.c" 입력

```
pi@raspberryPi:~/Example $ nano 16_SOUND_02.c
```

  - 현재의 음량에 따라 8LED를 다르게 점등하는 예제

```
1.  // File : 16_SOUND_02.c

2.  #include <stdio.h>
3.  #include <wiringPi.h>
4.  #include <wiringPiSPI.h>                    // SPI 라이브러리 참조
5.  #include <wiringPiI2C.h>                    // I2C 라이브러리 참조

6.  #define CS_ADC              7              // ADC 칩 번호 설정
7.  #define SPI_CHANNEL         0              // SPI 통신 설정
8.  #define SPI_SPEED           1000000

9.  // I2C 레지스터 설정
10. #define LED_I2C_ADDR    0x20
11. #define OUT_PORT1       0x03
12. #define CONFIG_PORT1    0x07

13. // 8LED 출력 데이터 설정
14. const int aLedData[8] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};
```

SPI 통신을 활용한 제어

- SOUND 제어 예제(2)

```
15.  int fd;                                                    // 8LED의 handle

16.  // 아날로그 Read 함수 정의
17.  int adcRead(char adcChannel)
18.  {
19.      char buff[3];                                          // 송수신용 데이터 변수 선언

20.

21.      int adcValue = 0;

22.      // SOUND가 연결된 채널은 CH2
23.      // 채널을 비트화 후 Shift 연산자를 통해 D2만 0번째 비트에 남겨둠
24.      // 0x60(Start 비트 : 1, Single-Endded 모드 비트 : 1)을 OR 연산
25.      buff[0] = 0x06 | ((adcChannel & 0x07) >> 2);

26.      // 채널을 비트화 후 Shift 연산자를 통해 D1, D0만 7, 6번째 비트에 남겨둠
27.      buff[1] = ((adcChannel & 0x07) << 6);

28.      buff[2] = 0x00;

29.      digitalWrite(CS_ADC, LOW);
```

SPI 통신을 활용한 제어

- SOUND 제어 예제(2)

```
29.     // SPI 통신을 통해 3byte 송수신
30.     // Single Ended 모드, CH0 전송
31.     // 2번째 Byte의 0~3비트, 3번째 Byte의 0~7비트에 ADC 결과값 저장
32.     wiringPiSPIDataRW(SPI_CHANNEL, buff, 3);

33.     // Mask(0x0F)를 이용해 2번째 Byte의 0~3비트 저장
34.     buff[1] = 0x0F & buff[1];

35.     // 2~3번째 Byte를 이용해 결과값 저장
36.     adcValue = (buff[1] << 8) | buff[2];

37.     digitalWrite(CS_ADC,HIGH);

38.     return adcValue;
39. }

40. // 범위에 따른 LED점등 함수 정의
41. void ledOnByRange(int x, int min, int max, int ledNo)
42. {
43.     int i;
44.     int data = 0;                          //8LED 출력Data 저장변수 선언
45.     if( x < max && x >= min)               //변수 x가 min과 max 사이값일 경우
46.     {
```

SPI 통신을 활용한 제어

- SOUND 제어 예제(2)

```
47.         for(i=0; i< ledNo; i++)                    //ledNo 개수만큼 LED 점등
48.         {
49.             data = aLedData[i] | data;
50.         }
51.         wiringPiI2CWriteReg16(fd, OUT_PORT1, data);
52.     }
53. }

54. int main(void)
55. {
56.     int adcValue_SOUND = 0;                    // ADC 한 결과값 저장 변수 초기화
57.     wiringPiSetupGpio();                       // 핀 번호를 BCM Mode로 설정

58.     // SPI 시스템 초기화 설정
59.     if(wiringPiSPISetup(SPI_CHANNEL, SPI_SPEED) < 0 )
60.     {
61.         return -1;
62.     }

63.     // I2C 시스템 초기화 설정
64.     if((fd = wiringPiI2CSetup(LED_I2C_ADDR)) < 0 )
65.     {
66.         return -1;
67.     }
```

SPI 통신을 활용한 제어

- SOUND 제어 예제(2)

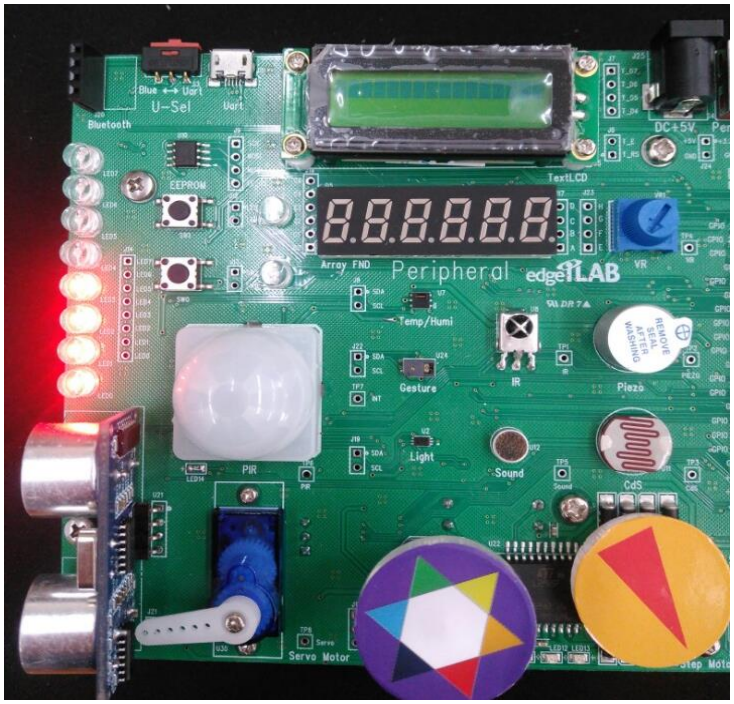
```
68.     pinMode(CS_ADC, OUTPUT);

69.     // handle을 통해 IO1 출력모드로 설정
70.     wiringPiI2CWriteReg16(fd, CONFIG_PORT1, 0x0000);

71.     while(1)
72.     {
73.         adcValue_SOUND = adcRead(2);
74.         printf("SOUND = %u₩n", adcValue_ SOUND);
75.         delay(100);

76.         // 측정 값의 범위에 따라 LED로 표시(센서마다 민감도 다를 수 있음)
77.         ledOnByRange(adcValue_SOUND, 2700, 4096, 0);
78.         ledOnByRange(adcValue_SOUND, 2500, 2700, 1);
79.         ledOnByRange(adcValue_SOUND, 2400, 2500, 2);
80.         ledOnByRange(adcValue_SOUND, 2300, 2400, 3);
81.         ledOnByRange(adcValue_SOUND, 2200, 2300, 4);
82.         ledOnByRange(adcValue_SOUND, 2100, 2200, 5);
83.         ledOnByRange(adcValue_SOUND, 1900, 2100, 6);
84.         ledOnByRange(adcValue_SOUND, 1700, 1900, 7);
85.         ledOnByRange(adcValue_SOUND, 1500, 1700, 8);
86.     }
87.     return 0;
88. }
```

SPI 통신을 활용한 제어

- SOUND 제어 예제(2)
  - 작성 후 "crtl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
  - GCC 컴파일러를 사용하여 빌드 및 생성된 "16_SOUND_02" 파일 실행

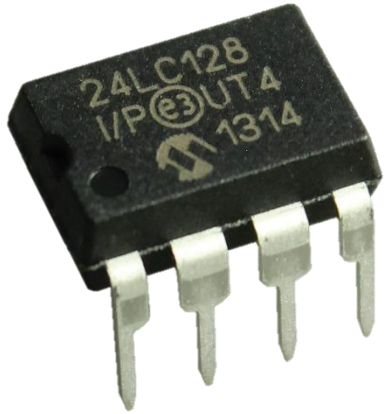
```
pi@raspberryPi:~/Example $ gcc -o 16_SOUND_02 16_SOUND_02.c -lwiringPi
pi@raspberryPi:~/Example $ ./16_SOUND_02
```

  - 결과
    - 현재의 음량을 측정하여 8LED로 음량의 크기를 표시



SPI 통신을 활용한 제어

- EEPROM 제어
  - EEPROM(Electrically Erasable Programmable Read Only Memory)이란
    - 전원이 OFF되어도 내용이 지워지지 않는 비휘발성 장치
  - 기록된 데이터는 전기적으로 소거하여 재 기록이 가능
  - 따라서 프로그램을 재 기록할 필요가 있는 응용분야에서 널리 사용
  - 칩을 구성하는 소자의 전하를 적기적으로 변화시킴으로써 데이터를 기록 및 소거
  - 재 기록하는데 다른 기억장치에 비해 시간이 많이 소모되고 기억용량이 작으며 횟수의 제한이 있지만, 전원 없이도 장기간 보존가능

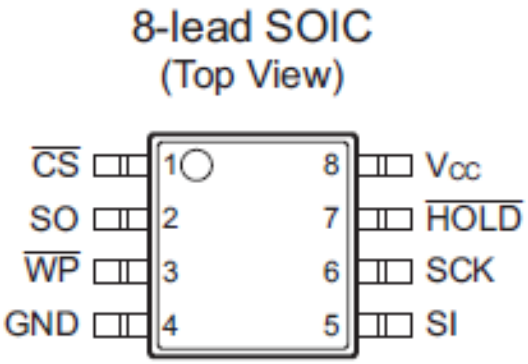


SPI 통신을 활용한 제어

- EEPROM 제어
  - AT25160B
    - SPI 지원
    - 저전력(1.8~5.5V)
    - 20MHz 클럭
    - Write Protect(WP) 기능 지원
    - 높은 신뢰성(1백만번 저장, 100년 보존)
    - 16,384 비트의 직렬 EEPROM 로, 각 8비트 2048 워드로 구성
    - 저전력/저전압 제품이 필요한 산업에 널리 사용
    - 사용법은 Chip Select(CS) 핀을 통해 활성화 되고 직렬 데이터 입력(SI), 직렬 데이터 출력(SO) 및 직렬 클럭(SCK)으로 구성된 3 wire 인터페이스를 통해 접근
    - 쓰기 전에는 별도의 소거 사이클이 필요

SPI 통신을 활용한 제어

- EEPROM 제어
  - AT25160B 직렬 인터페이스 기능



Pin Name	Function
$\overline{CS}$	Chip Select
GND	Ground
$\overline{HOLD}$	Suspends Serial Input
SCK	Serial Data Clock
SO	Serial Data Output
SI	Serial Data Input
$\overline{WP}$	Write Protect
Vcc	Power Supply

Interface	Description
Chip Select	AT25160B는 CS핀이 Low상태일 때 선택된다. 선택이 되지 않으면 SI핀을 통해 데이터가 받을 수 없고, SO핀이 High Impedance 상태를 유지한다.
HOLD	CS핀과 함께 AT25160B를 선택하고 직렬 시퀀스가 진행 중일때 HOLD 핀을 사용하여 직렬 시퀀스를 재설정하지 않고 마스터 장치와의 직렬 통신을 일시 중지 할 수 있다. 일시 정지하려면 SCK핀이 Low상태일때 HOLD 핀도 Low상태여야 한다. 직렬 통신을 재개하려면 SCK 핀이 Low일때 HOLD핀이 High 상태로 설정한다.
Write Protect	WP핀이 High상태로 유지되면 읽기/쓰기 작업을 허용한다. WP핀이 Low상태고 WPEN 비트가 '1'일 때, 상태 레지스터에 대한 모든 쓰기 동작이 금지된다. WP 핀 기능은 상태 레지스터의 WPEN 비트가 '0'일 때 차단된다. 즉 WPEN 비트가 '1'일 경우만 WP핀의 기능은 활성화된다.

SPI 통신을 활용한 제어

- EEPROM 제어
  - AT25160B 직렬 인터페이스 기능
    - AT25160B는 동기식 직렬 주변기기 인터페이스(SPI)와 직접 인터페이스 하도록 설계
    - AT25160B는 8비트 명령어 레지스터를 사용
    - 모든 명령어, 주소 및 데이터는 먼저 MSB(Most Significant Bit, 전송과 수신할 때의 첫 번째 비트)와 함께 전송되며 CS핀이 High에서 Low로 전환과 함께 시작

Instruction Name	Instruction Format	Operation
WREN	0000 X 110	Set Write Enable Latch
WRDI	0000 X 100	Reset Write Enable Latch
RDSR	0000 X 101	Read Status Register
WRSR	0000 X 001	Write Status Register
Read	0000 X 011	Read Data from Memory Array
Write	0000 X 010	Write Data to Memoy Array

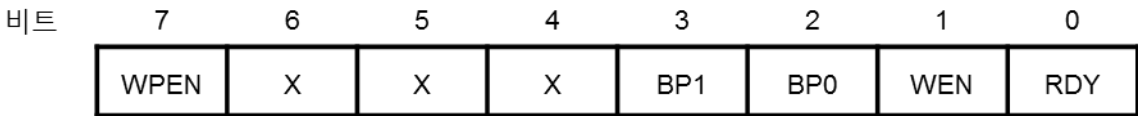
SPI 통신을 활용한 제어

- EEPROM 제어
  - AT25160B 직렬 인터페이스 기능
    - Write Enable(WREN)
      - Vcc가 적용되면 Write Disable 상태에서 장치의 전원을 킴
      - 즉, 모든 프로그래밍 명령어 앞에 Write Enable 명령어가 있어야 함
    - Write Disable(WRDI)
      - 의도하지 않은 쓰기로부터 장치를 보호하기 위한 명령어로 모든 프로그래밍 모드를 비활성화
      - WRDI 명령은 WP 핀의 상태와 독립적
    - Read Status Register(RDSR)
      - RDSR 명령은 상태 레지스터에 대한 접근을 제공
      - RDSR 명령을 통해 장치의 준비/바쁨/ Write Enable 상태 판별 가능
      - 유사하게, Block Write Protection 비트는 이용된 보호의 범위를 나타냄
      - 이 비트는 WRSR 명령을 사용하여 설정

비트	7	6	5	4	3	2	1	0
	WPEN	X	X	X	BP1	BP0	WEN	RDY

SPI 통신을 활용한 제어

- EEPROM 제어
  - AT25160B 직렬 인터페이스 기능



Bit	설 명
Bit 0 (RDY)	'0' 일 경우, 디바이스 준비 '1' 일 경우, 쓰기 사이클 진행 중
Bit 1 (WEN)	'0' 일 경우, 디바이스 쓰기 비활성화 '1' 일 경우, 디바이스 쓰기 활성화
Bit 2 (BP0)	아래 표 참조
Bit 3 (BP1)	아래 표 참조
Bit 4 to 6	디바이스가 내부 쓰기 사이클에 있지 않을 경우, '0'을 가리킴
Bit 7 (WPEN)	아래 표 참조
Bits 0 to 7	내부 쓰기 사이클 중에는 '1'을 가리킴

SPI 통신을 활용한 제어

- EEPROM 제어
  - AT25160B 직렬 인터페이스 기능
    - Write Status Register(WRSR)
      - WRSR 명령을 통해 사용자는 4가지 보호 레벨 중 하나 선택 가능
      - AT25160B는 4개의 세그먼트 열로 나뉨
      - 1/4, 1/2 또는 모든 메모리 세그먼트 보호 가능
      - 선택한 세그먼트 내의 모든 데이터는 읽기 전용
      - 블록 쓰기 보호 레벨과 해당 상태 레지스터 제어 비트는 아래 표 참조
      - 3개의 비트 BP0, BP1 및 WPEN
        - 정규 메모리 셀과 동일한 특성 및 기능을 갖는 비휘발성 셀

Level	Status Register Bits		Array Addresses Protected
	BP1	BP0	
0	0	0	None
1(1/4)	0	0	0600 – 07FF
2(1/2)	1	0	0400 – 07FF
3(All)	1	1	0000 – 07FF

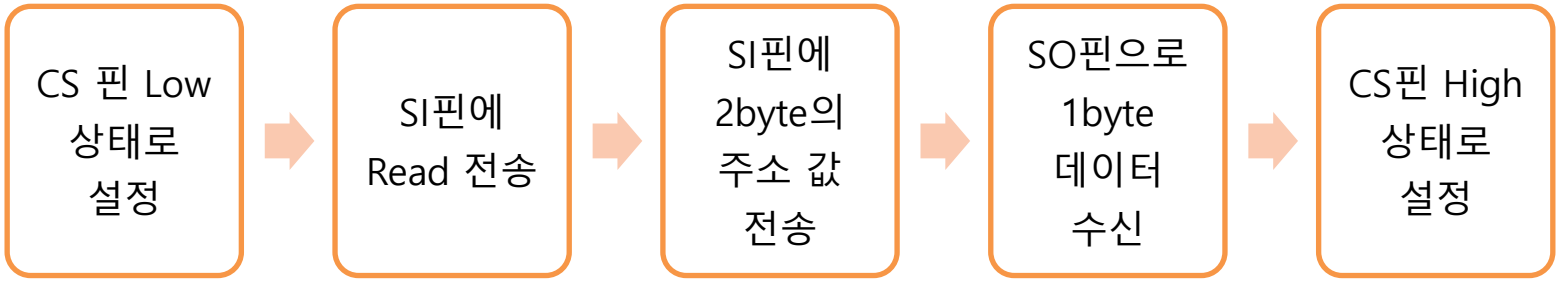
SPI 통신을 활용한 제어

- EEPROM 제어
  - AT25160B 직렬 인터페이스 기능
    - WRSR 명령을 통해 사용자는 WPEN(Write Protect Enable) 비트를 사용하여 WP(Write Protect) 핀을 활성화 및 비활성화 가능
    - 하드웨어 쓰기 보호
      - WP 핀이 Low이고, WPEN 비트가 '1'일 때 활성화
      - WP 핀이 High이거나 WPEN 비트가 '0'일 때 비활성화
    - 디바이스가 하드웨어 쓰기 보호인 경우, 블록 보호 비트 및 WPEN 비트를 포함하여 상태 레지스터에 쓰기와 메모리 배열의 블록 보호 섹션은 비활성화
    - 쓰기는 블록 보호되지 않은 메모리 섹션에만 허용
    - 또한 WPEN 비트가 하드웨어 쓰기 보호인 경우, WP 핀이 Low로 유지되는 한 다시 '0'으로 변경 불가능

WPEN	WP	WEN	Protected Blocks	Unprotected Blocks	Status Register
0	X	0	Protected	Protected	Protected
0	X	1	Protected	Writable	Writable
1	Low	0	Protected	Protected	Protected
1	High	1	Protected	Writable	Protected
X	High	0	Protected	Protected	Protected
X	High	1	Protected	Writable	Writable

SPI 통신을 활용한 제어

- EEPROM 제어
  - AT25160B 직렬 인터페이스 기능
    - Read Sequence(Read)
      - 직렬 출력(SO)핀을 통해 AT25160B를 읽으려면 순서가 필요
      - CS 핀이 Low로 설정한 후, 읽기 연산 코드(READ OPCODE)는 SI 핀을 통해 전송되고 2Byte 주소(A15-A0)가 읽힘
      - 완료되면 SI 핀의 모든 데이터는 무시
      - 지정된 주소의 데이터(D7~ D0)가 SO 핀으로 수신
      - 1Byte 만 읽어 데이터가 수신되면, CS 핀을 High로 설정
      - 바이트 주소가 자동으로 증가되고 데이터가 계속 이동되므로 Read Sequence 지속 가능
      - 가장 높은 주소에 도달하면 주소 카운터가 가장 낮은 주소로 되돌아가 전체 메모리를 하나의 연속 읽기 주기로 읽기 가능

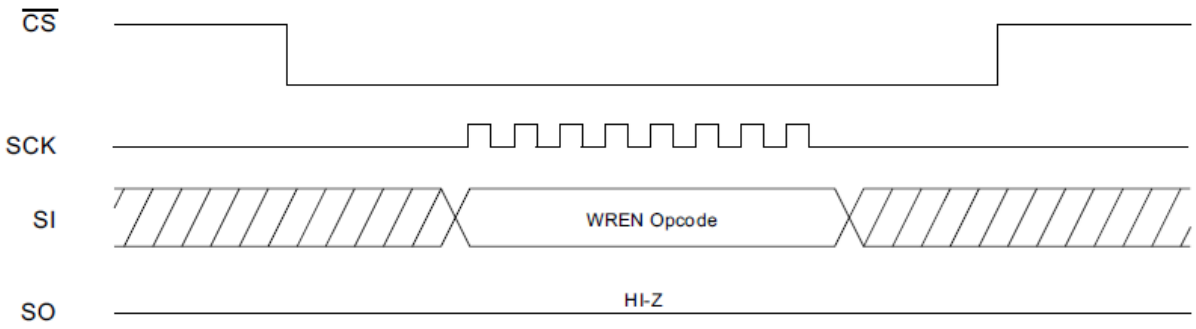


SPI 통신을 활용한 제어

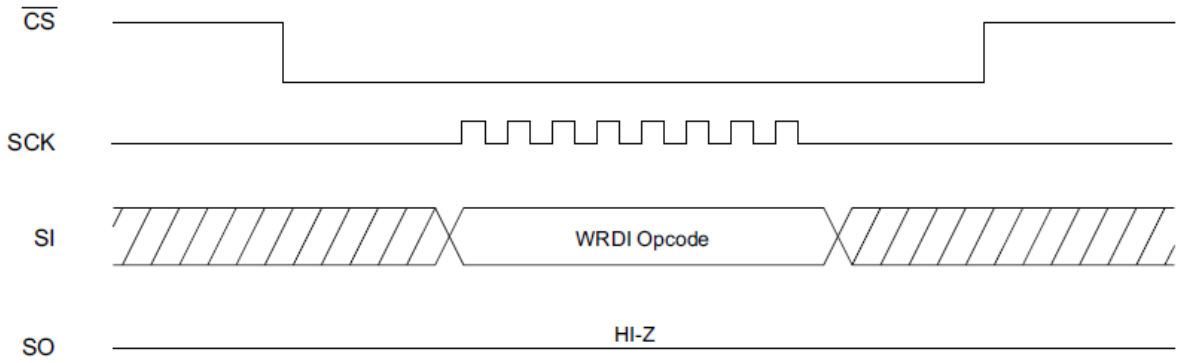
- EEPROM 제어
  - AT25160B 직렬 인터페이스 기능
    - Write Sequence(Write)
      - AT25160B를 프로그래밍하려면 두 개의 명령을 실행
        - 먼저, 장치는 WREN 명령을 통해 쓰기가 가능해야 하며, 그 후 Write 명령을 실행할 수 있음
        - 프로그래밍할 메모리 위치의 주소는 블록 쓰기 보호 레벨에 의해 선택된 주소 필드 위치의 외부에 있어야 함
        - 내부 쓰기 사이클 중에는 RDSR 명령을 제외한 모든 명령이 무시
      - 데이터를 저장하기 위해서는 다음과 같이 진행
        - CS 핀을 Low 상태로 설정한 후, SI 핀을 통해 쓰기 연산 코드(WRITE OPCODE)가 전송되고, 그 뒤에 바이트 주소(A15-1A0)와 프로그래밍할 데이터(D7-D0)을 전송
        - CS핀이 High가 된 후 프로그래밍이 시작
        - CS 핀의 Low에서 High로의 이동은 D0(LSB) 데이터 비트의 클로킹 직후 SCK Low 시간동안 발생
      - 장치의 준비/사용 중 상태는 RDSR 명령을 시작하여 확인
        - 비트 0이 '1'이면, 쓰기 사이클은 진행 중을 의미
        - 비트 0이 '0'이면, 쓰기 사이클이 끝남

SPI 통신을 활용한 제어

- EEPROM 제어
  - AT25160B 직렬 인터페이스 기능
    - 타이밍 다이어그램
      - WREN 타이밍

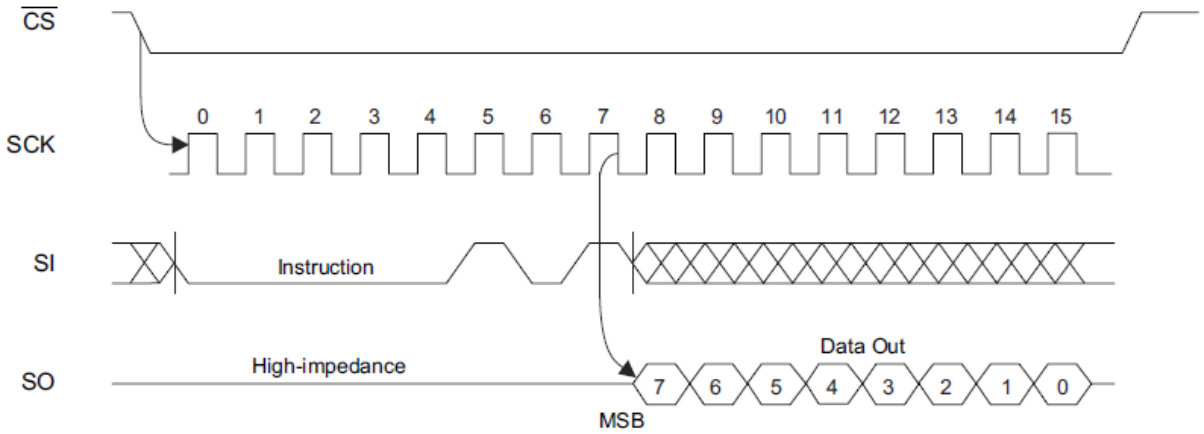


      - WRDI 타이밍

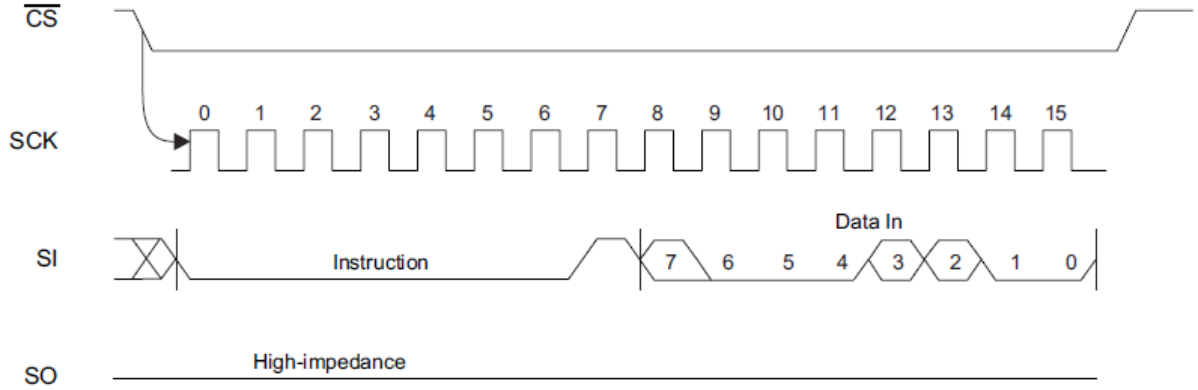


SPI 통신을 활용한 제어

- EEPROM 제어
  - AT25160B 직렬 인터페이스 기능
    - 타이밍 다이어그램
      - RDSR 타이밍

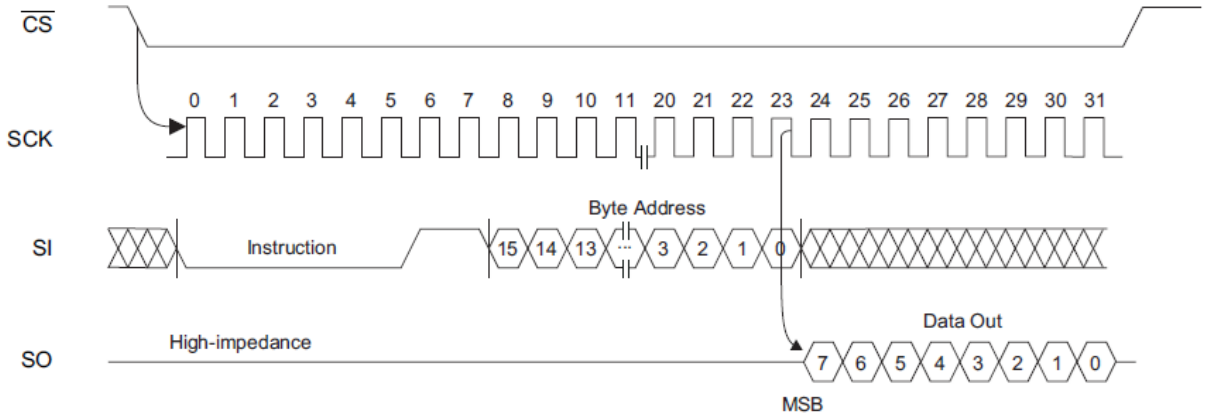


      - WRSR 타이밍

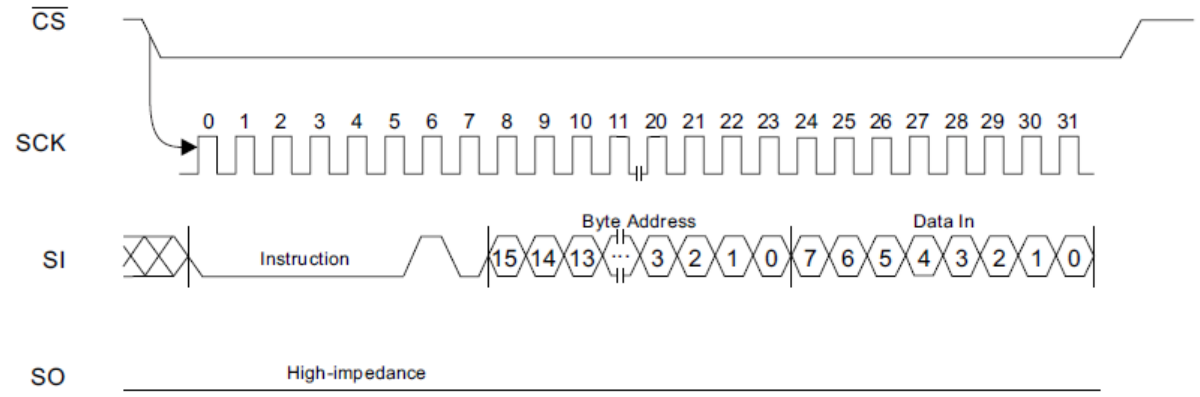


SPI 통신을 활용한 제어

- EEPROM 제어
  - AT25160B 직렬 인터페이스 기능
    - 타이밍 다이어그램
      - Read 타이밍

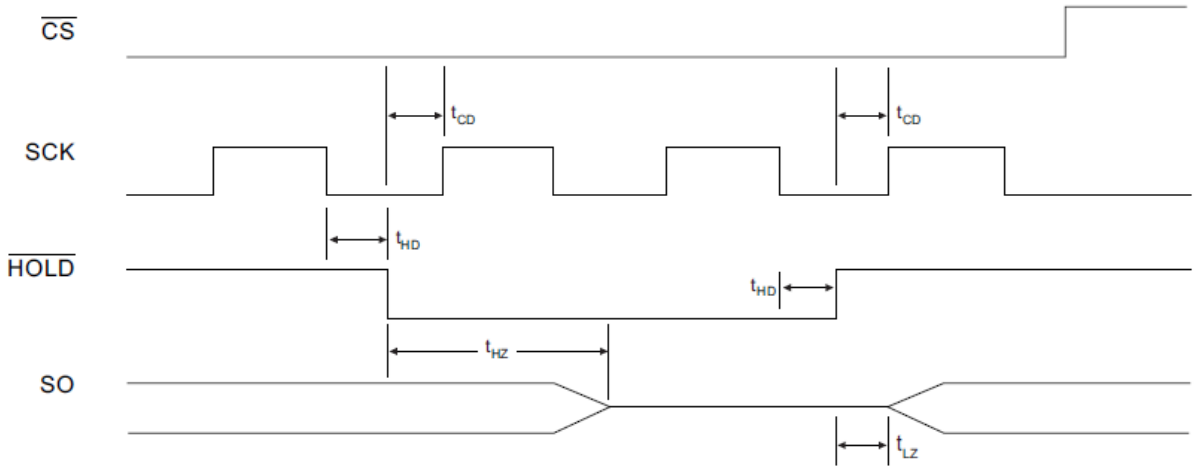


      - Write 타이밍



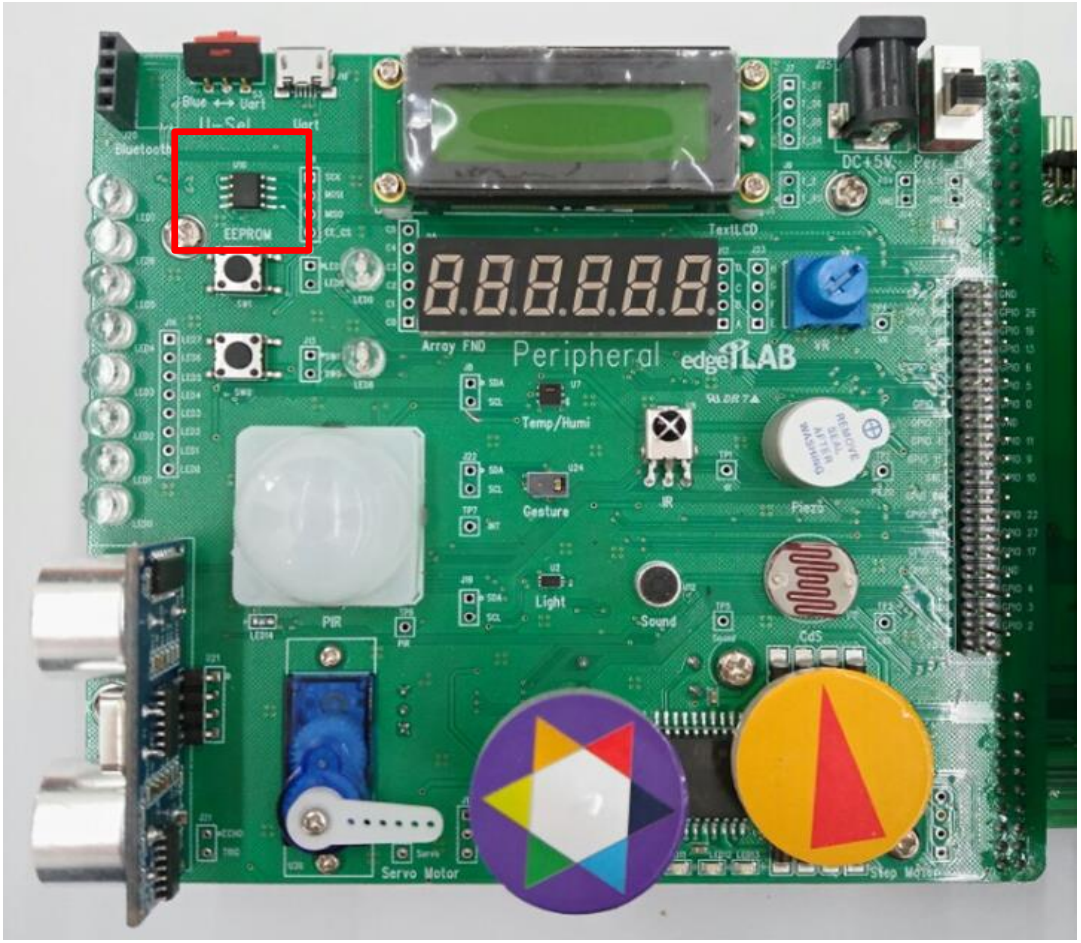
SPI 통신을 활용한 제어

- EEPROM 제어
  - AT25160B 직렬 인터페이스 기능
    - 타이밍 다이어그램
      - HOLD 타이밍



SPI 통신을 활용한 제어

- EEPROM 제어
  - Edge-Embedded의 EEPROM



SPI 통신을 활용한 제어

- EEPROM 제어 예제(1)
  - 터미널 창에 "nano 17_EEPROM_01.c" 입력

```
pi@raspberryPi:~/Example $ nano 17_EEPROM_01.c
```

  - EEPROM에 "RASPI"를 저장하고, 저장된 값을 출력하는 예제

```
1.  // File : 17_EEPROM_01.c

2.  #include <stdio.h>
3.  #include <string.h>
4.  #include <wiringPi.h>
5.  #include <wiringPiSPI.h>                    // SPI 라이브러리 참조

6.  #define CS_EEPROM              8
7.  #define SPI_CHANNEL            0
8.  #define SPI_SPEED              1000000

9.  // EEPROM 레지스터 설정
10. #define WREN                   0x06
11. #define WRDI                   0x04
12. #define RDSR                   0x05
13. #define WRSR                   0x01
14. #define READ                   0x03
15. #define WRITE                  0x02
```

SPI 통신을 활용한 제어

- EEPROM 제어 예제(1)

```
16. int main(void)
17. {
18.     char buf[9];                                    // 송수신용 데이터 변수 선언
19.     wiringPiSetupGpio();                            // 핀 번호를  BCM Mode로 설정

20.     // SPI 시스템 초기화 설정
21.     if(wiringPiSPISetup(SPI_CHANNEL, SPI_SPEED) < 0)
22.     {
23.         return -1;
24.     }

25.     pinMode(CS_EEPROM, OUTPUT);
26.     digitalWrite(CS_EEPROM, HIGH);

27.     // EEPROM 프로그래밍 모드 활성화
28.     digitalWrite(CS_EEPROM, LOW);
29.     delayMicroseconds(1);
30.     buff[0] = WREN;
31.     wiringPiSPIDataRW(SPI_CHANNEL, buff, 1);
32.     delayMicroseconds(1);
33.     digitalWrite(CS_EEPROM, HIGH);
34.     delay(1);
```

SPI 통신을 활용한 제어

- EEPROM 제어 예제(1)

```
35.     // EEPROM에 데이터 쓰기
36.     printf("SAVE : RASPI₩n");
37.     digitalWrite(CS_EEPROM, LOW);
38.     delayMicroseconds(1);
39.     buff[0] = WRITE;
40.     buff[1] = 0x00;              // address [15:8]
41.     buff[2] = 0x11;              // address [7:0], 주소 지정
42.     buff[3] = 'R';
43.     buff[4] = 'A';
44.     buff[5] = 'S';
45.     buff[6] = 'P';
46.     buff[7] = 'I';
47.     buff[8] = '₩n';

48.     // SPI 통신으로9byte의 데이터를 EEPROM에 전송
49.     wiringPiSPIDataRW(SPI_CHANNEL, buff, 9);
50.     delayMicroseconds(1000);
51.     digitalWrite(CS_EEPROM, HIGH);
52.     delay(5000);
```

SPI 통신을 활용한 제어

- EEPROM 제어 예제(1)

```
53.     // EEPROM 프로그래밍 모드 비활성화
54.     digitalWrite(CS_EEPROM, LOW);
55.     delayMicroseconds(1);
56.     buff[0] = WRDI;

57.     wiringPiSPIDataRW(SPI_CHANNEL, buff, 1);
58.     delayMicroseconds(1);
59.     digitalWrite(CS_EEPROM, HIGH);
60.     delay(1);

61.     // EEPROM으로부터 데이터 읽기
62.     digitalWrite(CS_EEPROM, LOW);
63.     delayMicroseconds(1);
64.     buff[0] = READ;
65.     buff[1] = 0x00;              // address [15:8]
66.     buff[2] = 0x11;              // address [7:0], 주소 지정

67.     // SPI통신을 통해 읽기,쓰기 동시에 진행
68.     // EEPROM에 저장된 값을 나머지 6byte에 저장
69.     wiringPiSPIDataRW(SPI_CHANNEL, buff, 9);
70.     delayMicroseconds(1000);
71.     digitalWrite(CS_EEPROM, HIGH);
72.     delay(1000);
```

SPI 통신을 활용한 제어

- EEPROM 제어 예제(1)

```
73.     printf("READ : ");

74.     int i;
75.     for(i=3; i<9; i++)
76.     {
77.         // buff[3]~buff[8]까지 저장된 값 출력
78.         printf("%c", buff[i]);
79.     }
80.     printf("₩n");
81.     return 0;
82. }
```

  - 작성 후 "crtl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
  - GCC 컴파일러를 사용하여 빌드 및 생성된 "17_EEPROM_01" 파일 실행

```
pi@raspberryPi:~/Example $ gcc -o 17_EEPROM_01 17_EEPROM_01.c -lwiring
Pi
pi@raspberryPi:~/Example $ ./17_EEPROM_01
```

SPI 통신을 활용한 제어

- EEPROM 제어 예제(1)
  - 결과
    - EEPROM에 "RASPI"를 저장하고, 이를 다시 읽어와 터미널 화면에 출력

```
pi@raspberrypi:~/Example $ ./17_EEPROM_01
SAVE : RASPI
READ : RASPI
```

SPI 통신을 활용한 제어

- EEPROM 제어 예제(2)
  - 터미널 창에 "nano 17_EEPROM_02.c" 입력

```
pi@raspberryPi:~/Example $ nano 17_EEPROM_02.c
```

  - 사용자로부터 문자열(5글자)을 입력 받아, EEPROM에 이를 저장하고 저장된 값을 출력하는 예제

```
1.  // File : 17_EEPROM_02.c

2.  #include <stdio.h>
3.  #include <string.h>
4.  #include <wiringPi.h>
5.  #include <wiringPiSPI.h>                    // SPI 라이브러리 참조

6.  #define CS_EEPROM           8
7.  #define SPI_CHANNEL        0
8.  #define SPI_SPEED          1000000
```

SPI 통신을 활용한 제어

- EEPROM 제어 예제(2)

```
9.  // EEPROM 레지스터 설정
10. #define WREN                0x06
11. #define WRDI                0x04
12. #define RDSR                0x05
13. #define WRSR                0x01
14. #define READ                0x03
15. #define WRITE               0x02

16. int main(void)
17. {
18.     char buff[9];                          // 송수신용 데이터 변수 선언
19.     wiringPiSetupGpio();                   // 핀 번호를  BCM Mode로 설정

20.     // SPI 시스템 초기화 설정
21.     if(wiringPiSPISetup(SPI_CHANNEL, SPI_SPEED) < 0)
22.     {
23.         return -1;
24.     }

25.     pinMode(CS_EEPROM, OUTPUT);
26.     digitalWrite(CS_EEPROM, HIGH);
```

SPI 통신을 활용한 제어

- EEPROM 제어 예제(2)

```
27.     // EEPROM 프로그래밍 모드 활성화
28.     digitalWrite(CS_EEPROM, LOW);
29.     delayMicroseconds(1);
30.     buff[0] = WREN;

31.     wiringPiSPIDataRW(SPI_CHANNEL, buff, 1);
32.     delayMicroseconds(1);

33.     digitalWrite(CS_EEPROM, HIGH);
34.     delay(1);

35.     // 사용자로부터 문자열 입력받고 , EEPROM에 데이터 저장
36.     printf("Input String : ");
37.     digitalWrite(CS_EEPROM, LOW);
38.     delayMicroseconds(1);

39.     buff[0] = WRITE;
40.     buff[1] = 0x00;              // address [15:8]
41.     buff[2] = 0x11;              // address [7:0], 주소 지정

42.     int i;
43.     for(i=3; i<9; i++)           // i = 3부터 9가 될 때까지 buff[i]에 char변수 저장
44.     {
```

SPI 통신을 활용한 제어

- EEPROM 제어 예제(2)

```
45.         scanf("%c", &buff[i]);
46.     }

47.     // SPI 통신으로9byte의 데이터를 EEPROM에 전송
48.     wiringPiSPIDataRW(SPI_CHANNEL, buff, 9);
49.     delayMicroseconds(1000);

50.     digitalWrite(CS_EEPROM, HIGH);
51.     delay(5000);

52.     // EEPROM 프로그래밍 모드 비활성화
53.     digitalWrite(CS_EEPROM, LOW);
54.     delayMicroseconds(1);
55.     buff[0] = WRDI;

56.     wiringPiSPIDataRW(SPI_CHANNEL, buff, 1);
57.     delayMicroseconds(1);
58.     digitalWrite(CS_EEPROM, HIGH);
59.     delay(1);

60.     // EEPROM으로부터 데이터 읽기
61.     digitalWrite(CS_EEPROM, LOW);
62.     delayMicroseconds(1);
```

SPI 통신을 활용한 제어

- EEPROM 제어 예제(2)

```
63.     buff[0] = READ;
64.     buff[1] = 0x00;              // address [15:8]
65.     buff[2] = 0x11;              // address [7:0], 주소 지정

66.     // SPI통신을 통해 읽기, 쓰기 동시에 진행
67.     // EEPROM에 저장된 값을 나머지 6byte에 저장
68.     wiringPiSPIDataRW(SPI_CHANNEL, buff, 9);
69.     delayMicroseconds(1000);

70.     digitalWrite(CS_EEPROM, HIGH);
71.     delay(1000);

72.     printf("READ : ");

73.     for(i=3; i<9; i++)
74.     {
75.         // buff[3]~buff[8]까지 저장된 값 출력
76.         printf("%c", buff[i]);
77.     }
78.     printf("₩n");
79.     return 0;
80. }
```

SPI 통신을 활용한 제어

- EEPROM 제어 예제(2)
  - 작성 후 "crtl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
  - GCC 컴파일러를 사용하여 빌드 및 생성된 "17_EEPROM_02" 파일 실행

```
pi@raspberryPi:~/Example $ gcc -o 17_EEPROM_02 17_EEPROM_02.c -lwiring
Pi
pi@raspberryPi:~/Example $ ./17_EEPROM_02
```

  - 결과
    - 사용자로부터 문자열(5글자)을 입력받아 EEPROM에 저장하고, 다시 EEPROM을 읽어 문자열을 출력

```
pi@raspberrypi:~/Example $ ./17_EEPROM_02
Input String : HELLO
READ : HELLO
```