

I2C 통신을 활용한 예제 및 실습

➤ I2C 통신을 활용한 제어



엣지아이랩

I2C 통신을 활용한 제어

I2C 통신을 활용한 제어

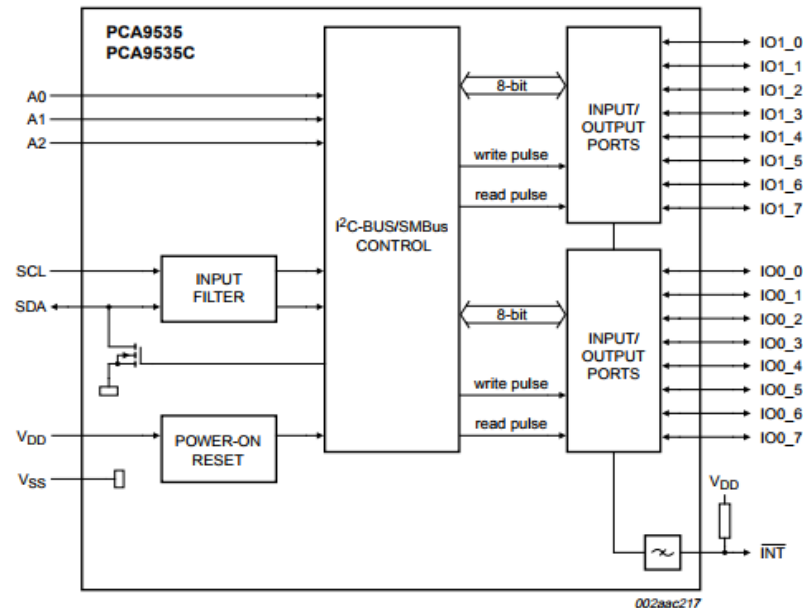
- PCA9535PW
 - I2C 인터페이스 확장 칩
 - 16비트 범용 입출력(GPIO)확장 제공
 - 2개의 8비트 입/출력 및 극성 반전 레지스터로 구성
 - NXP 반도체 제품의 I2C 버스 I/O 확장기 향상을 위해 개발된 24핀 CMOS 장치
 - 8LED, Step 모터, FND 센서 및 액추레이터 연결



I2C 통신을 활용한 제어

PCA9535PW

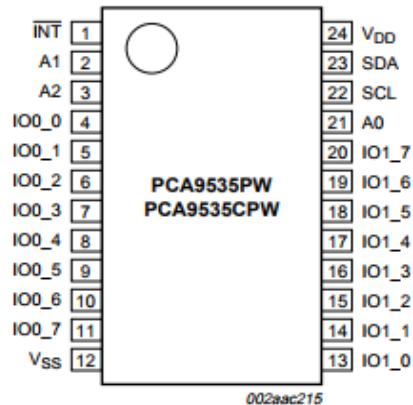
- 시스템 마스터는 I/O 설정 비트에 쓰기를 하므로써 입력 또는 출력으로 I/O를 활성화
- 각 입/출력에 대한 데이터는 해당 입/출력 레지스터에 저장
- 파워 온 리셋은 레지스터를 기본값으로 설정하고 장치를 초기화
- 세 개의 하드웨어 핀은 고정 I2C 버스 주소를 변경하고 최대 8개의 장치가 동일한 I2C 버스/SMBus를 공유할 수 있도록 함



Remark: All I/Os are set to inputs at reset.

I2C 통신을 활용한 제어

- PCA9535PW
 - Pin Out



Pin	Pin Name	Function	Pin	Pin Name	Function
1	INT	Interrupt output(open-drain)	13	IO1_0	Port 1 input/output
2	A1	Address input 1	14	IO1_1	
3	A2	Address input 2	15	IO1_2	
4	IO0_0	Port 0 input/output	16	IO1_3	
5	IO0_1		17	IO1_4	
6	IO0_2		18	IO1_5	
7	IO0_3		19	IO1_6	
8	IO0_4		20	IO1_7	
9	IO0_5		21	A0	Address input 0
10	IO0_6		22	SCL	Serial clock line
11	IO0_7		23	SDA	Serial data line
12	Vss	Supply ground	24	Vdd	Supply voltage

라즈베리파이에서의 I2C 사용

- PCA9535PW
 - 레지스터
 - 제어 레지스터(명령 바이트)
 - 쓰기 전송 중 주소 바이트 다음에 나오는 첫 번째 바이트
 - 포인터로 사용되어 다음 레지스터 중 쓰기 또는 읽기를 결정

Command	Register
0x00	Input port 0
0x01	Input port 1
0x02	Output port 0
0x03	Output port 1
0x04	Polarity Inversion port 0
0x05	Polarity Inversion port 1
0x06	Configuration port 0
0x07	Configuration port 1

라즈베리파이에서의 I2C 사용

● PCA9535PW

— 레지스터

■ 레지스터 0, 1 : Input port 레지스터

— 입력 전용 포트

— Configuration 레지스터에 의해 입력 또는 출력으로 정의되었는지 여부에 관계 없이 핀의 논리 상태('1' 또는 '0') 반영

— 기본값 'X'는 외부적으로 적용된 논리 상태에 의해 결정

비트	7	6	5	4	3	2	1	0
	I0.7	I0.6	I0.5	I0.4	I0.3	I0.2	I0.1	I0.0

기본값	X	X	X	X	X	X	X	X
-----	---	---	---	---	---	---	---	---

비트	7	6	5	4	3	2	1	0
	I1.7	I1.6	I1.5	I1.4	I1.3	I1.2	I1.1	I1.0

기본값	X	X	X	X	X	X	X	X
-----	---	---	---	---	---	---	---	---

라즈베리파이에서의 I2C 사용

- PCA9535PW
 - 레지스터
 - 레지스터 2, 3 : Output port 레지스터
 - 출력 전용 포트
 - Configuration 레지스터에 의해 출력으로 정의된 핀의 출력 논리 상태를 반영

비트	7	6	5	4	3	2	1	0
	O0.7	O0.6	O0.5	O0.4	O0.3	O0.2	O0.1	O0.0

기본값	1	1	1	1	1	1	1	1
-----	---	---	---	---	---	---	---	---

비트	7	6	5	4	3	2	1	0
	O1.7	O1.6	O1.5	O1.4	O1.3	O1.2	O1.1	O1.0

기본값	1	1	1	1	1	1	1	1
-----	---	---	---	---	---	---	---	---

라즈베리파이에서의 I2C 사용

- PCA9535PW
 - 레지스터
 - 레지스터 4, 5 : Polarity Inversion 레지스터
 - Configuration 레지스터에 의해 입력으로 정의된 핀의 극성 반정을 허용
 - 레지스터의 비트가 '1'인 경우, 해당 핀의 극성이 반전
 - 레지스터의 비트가 '0'인 경우, 해당 핀의 원래 극성 유지

비트	7	6	5	4	3	2	1	0
	N0.7	N0.6	N0.5	N0.4	N0.3	N0.2	N0.1	N0.0
기본값	0	0	0	0	0	0	0	0

비트	7	6	5	4	3	2	1	0
	N1.7	N1.6	N1.5	N1.4	N1.3	N1.2	N1.1	N1.0
기본값	0	0	0	0	0	0	0	0

라즈베리파이에서의 I2C 사용

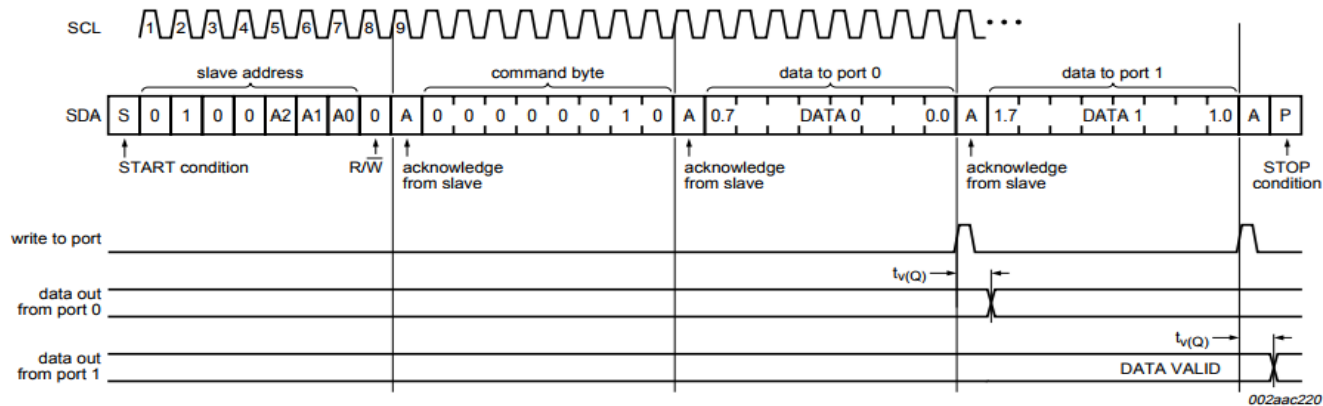
- PCA9535PW
 - 레지스터
 - 레지스터 6, 7 : Configuration 레지스터
 - I/O 핀의 방향 설정
 - 레지스터의 비트가 '1'인 경우, 해당 핀의 Port 입력 설정
 - 레지스터의 비트가 '0'인 경우, 해당 핀의 Port 출력 설정

비트	7	6	5	4	3	2	1	0
	C0.7	C0.6	C0.5	C0.4	C0.3	C0.2	C0.1	C0.0
기본값	1	1	1	1	1	1	1	1

비트	7	6	5	4	3	2	1	0
	C1.7	C1.6	C1.5	C1.4	C1.3	C1.2	C1.1	C1.0
기본값	1	1	1	1	1	1	1	1

라즈베리파이에서의 I2C 사용

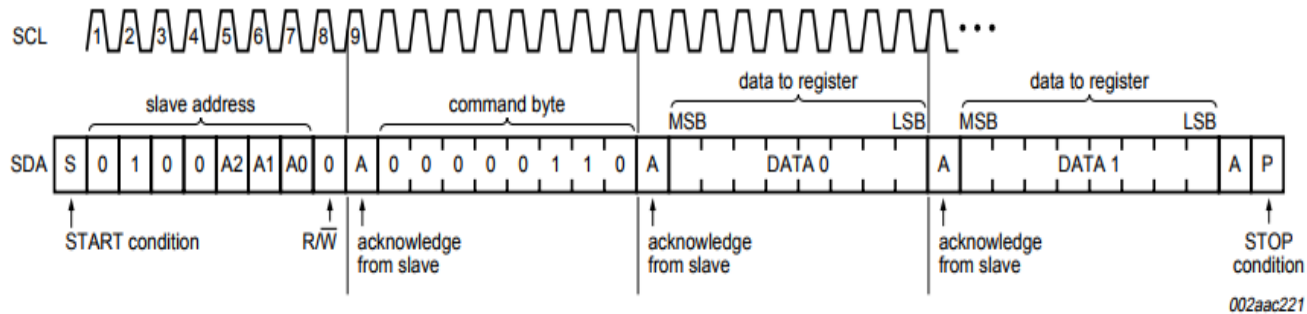
- PCA9535PW
 - Port 레지스터에 쓰기



- 데이터는 디바이스 어드레스를 전송하고 최하위 비트를 '0'으로 설정함으로써 PCA9535로 전송
- 명령 바이트는 주소 다음에 전송되며 명령 바이트 다음에 데이터를 수신 할 레지스터를 결정
- PCA9535 내의 8 개 레지스터는 4 개의 레지스터 쌍으로 작동하도록 구성
- 4 개의 쌍은 입력 포트, 출력 포트, 극성 반전 포트 및 구성 포트
- 하나의 레지스터에 데이터를 전송 한 후, 다음 데이터 바이트가 쌍의 다른 레지스터로 전송

라즈베리파이에서의 I2C 사용

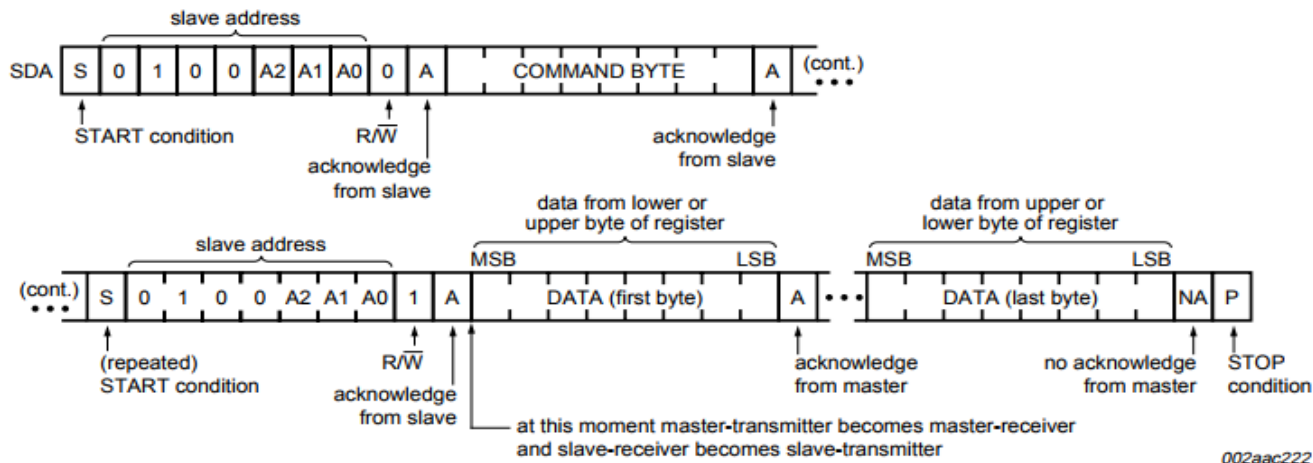
- PCA9535PW
 - Port 레지스터에 쓰기



- 예를 들어 첫 번째 바이트가 출력 포트 1 (레지스터 3)으로 전송되면 다음 바이트는 출력 포트 0 (레지스터 2)에 저장
- 하나의 쓰기 전송에서 전송되는 데이터 바이트 수에는 제한이 없고, 이러한 방식으로 각각의 8 비트 레지스터는 다른 레지스터와 독립적으로 업데이트 가능

라즈베리파이에서의 I2C 사용

- PCA9535PW
 - Port 레지스터에 읽기

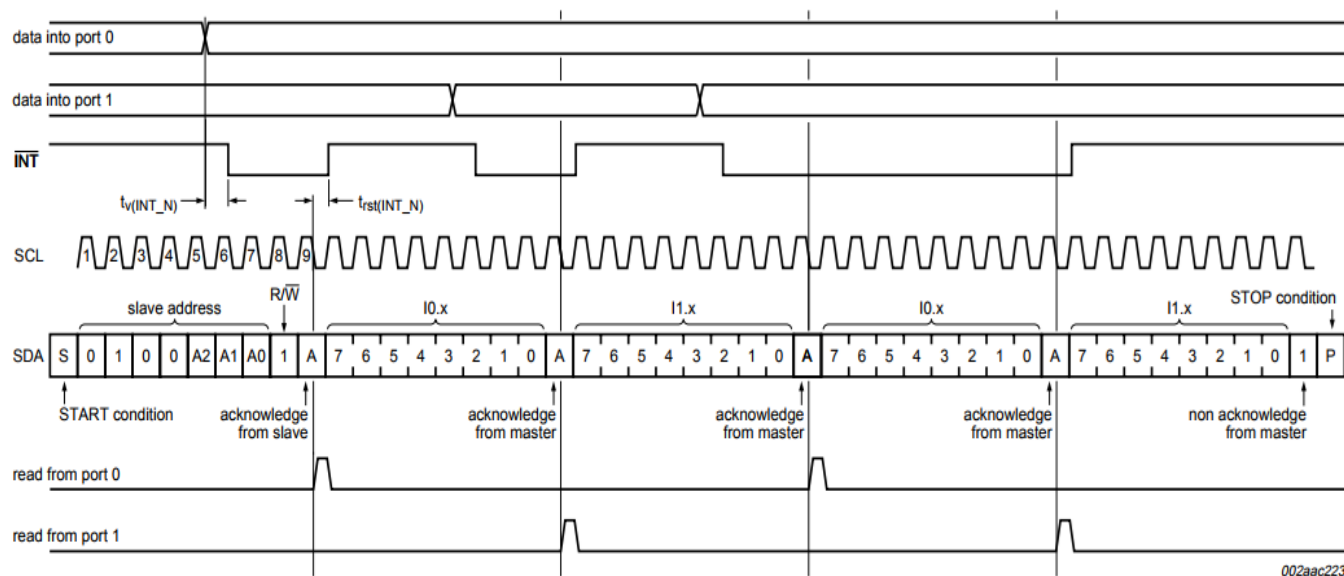


Remark: Transfer can be stopped at any time by a STOP condition.

- PCA9535에서 데이터를 읽으려면 버스 마스터가 최하위 비트가 논리 '0'으로 설정된 PCA9535 주소를 먼저 전송
- 명령 바이트는 주소 다음에 전송되며 액세스 할 레지스터를 결정
- 다시 시작한 후 장치 주소가 다시 전송되지만 이번에는 최하위 비트가 논리 1로 설정

라즈베리파이에서의 I2C 사용

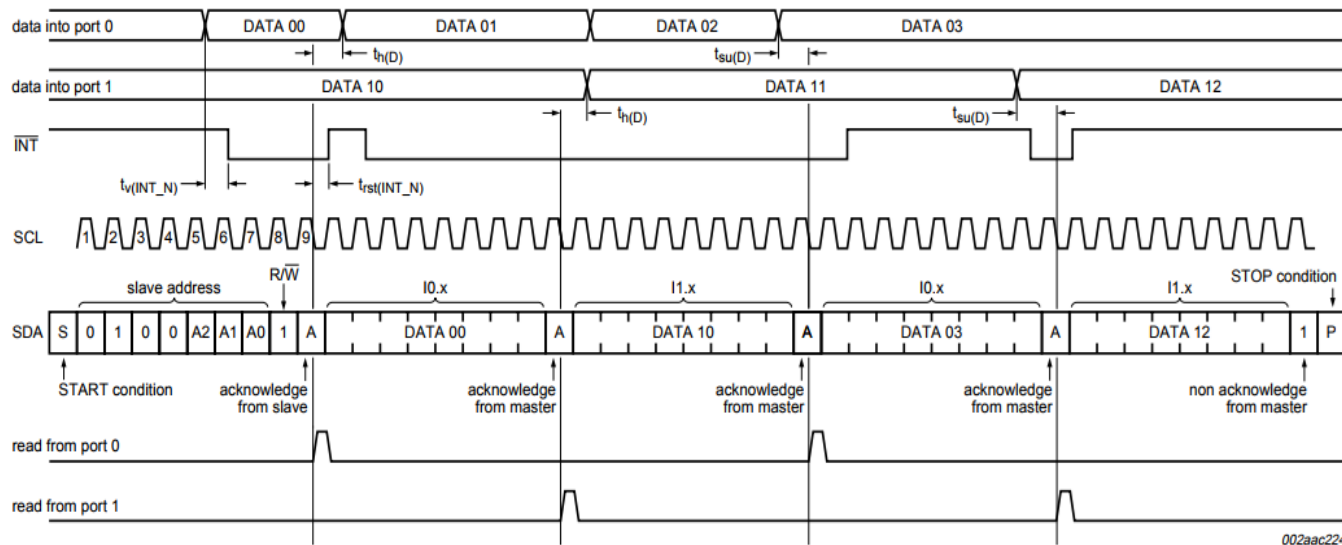
- PCA9535PW
 - Port 레지스터에 읽기



- 커맨드 바이트에 의해 정의 된 레지스터의 데이터는 PCA9535 에 의해 전송
- 데이터는 확인 클럭 펄스의 하강 엣지에서 레지스터로 기록
- 첫 번째 바이트를 읽은 후에는 추가 바이트를 읽을 수 있지만 데이터는 쌍의 다른 레지스터의 정보를 반영

라즈베리파이에서의 I2C 사용

- PCA9535PW
 - Port 레지스터에 읽기



- 예를 들어, 입력 포트 1을 읽으면 다음 바이트는 입력 포트 0이 됨
- 하나의 읽기 전송에서 수신된 데이터 바이트의 수에는 제한이 없음
- 마지막 바이트는 수신되므로 버스 마스터는 데이터를 확인하지 않아도 함

라즈베리파이에서의 I2C 사용

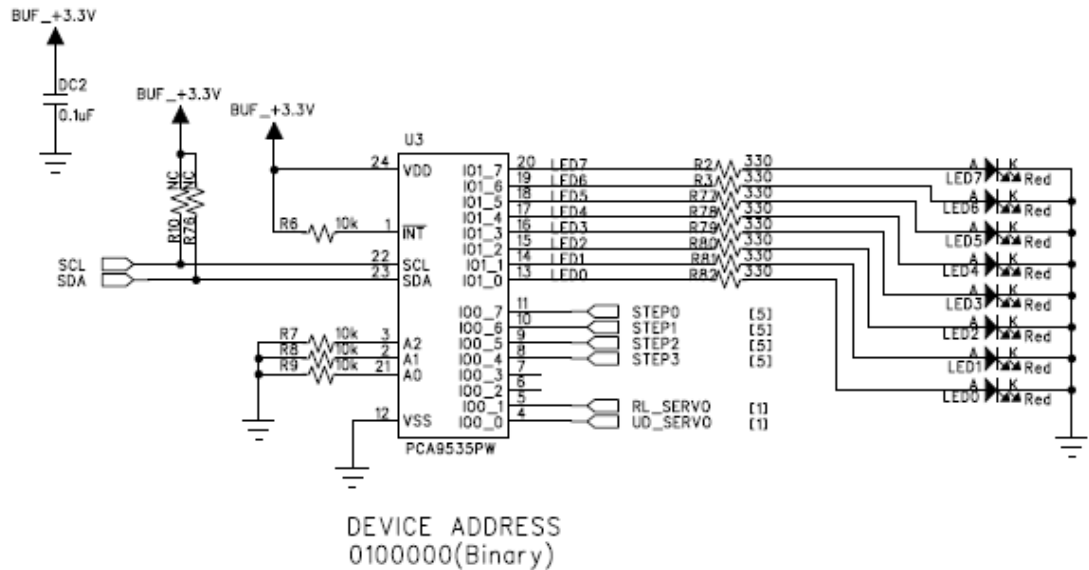
- Edge-Embedded 의 I2C
 - I2C 는 핀 02, 03 번(BCM Mode)와 연결
 - 디바이스 주소 참조

	Connect	BCM	Name	Physical	Name	BCM	Connect
I2C			+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
			GND	9	10	RxD	GPIO 15
	SERVO	GPIO 17	GPIO	11	12	GPIO	GPIO 18
	TLCD	GPIO 27	GPIO	13	14	GND	
	TLCD	GPIO 22	GPIO	15	16	GPIO	GPIO 23
			+3.3V	17	18	GPIO	GPIO 24
	SPI	GPIO 10	MOSI	19	20	GND	
	SPI	GPIO 09	MISO	21	22	GPIO	GPIO 25
	SPI	GPIO 11	SCLK	23	24	CE0	GPIO 08
			GND	25	26	CE1	GPIO 07
	ULTRA	GPIO 00	SDA0	27	28	SCL0	GPIO 01
	SWITCH	GPIO 05	GPIO	29	30	GND	
	SWITCH	GPIO 06	GPIO	31	32	GPIO	GPIO 12
	PIEZO	GPIO 13	GPIO	33	34	GND	
	IR	GPIO 19	GPIO	35	36	GPIO	GPIO 16
	TLCD	GPIO 26	GPIO	37	38	GPIO	GPIO 20
			GND	39	40	GPIO	GPIO 21

디바이스	주소
LED 8ea	0x20
Step Motor	0x20
6-Array FND	0x21
Light	0x23
Temp/Humi	0x40
Gesture	0x73

라즈베리파이에서의 I2C 사용

- 8LED 제어
 - PCA9535 칩의 IO 1 번 포트에 연결



라즈베리파이에서의 I2C 사용

- 8LED 제어 예제(1)
 - 터미널 창에 “nano 09_8LED_01.c” 입력

```
pi@raspberrypi:~/Example $ nano 09_8LED_01.c
```

- I2C통신을 이용하여 8LED를 제어하는 예제

```
1. // File : 09_8LED_01.c
2. #include <stdio.h>
3. #include <wiringPi.h>
4. #include <wiringPiI2C.h>                // I2C 라이브러리 참조
5. #define LED_I2C_ADDR          0x20      // 8LED I2C 주소
6. // 제어 레지스터 설정
7. #define IN_PORT0               0x00
8. #define IN_PORT1               0x01
9. #define OUT_PORT0              0x02
10. #define OUT_PORT1              0x03
11. #define POLARITY_IVE_PORT0     0x04
12. #define POLARITY_IVE_PORT1     0x05
13. #define CONFIG_PORT0           0x06
14. #define CONFIG_PORT1           0x07
```

라즈베리파이에서의 I2C 사용

● 8LED 제어 예제(1)

```
15. // 8개의 LED 데이터
16. const int aLedData[8] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};

17. int fd;                                // 8LED의 handle

18. int main(void)
19. {
20.     // I2C 시스템 초기화 설정
21.     if((fd = wiringPiI2CSetup(LED_I2C_ADDR)) < 0)
22.     {
23.         return -1;                      // 설정이 안될 경우, 종료
24.     }

25.     // handle을 통해 IO1 출력모드로 설정
26.     wiringPiI2CWriteReg16(fd, CONFIG_PORT1, 0x00);

27.     while(1)
28.     {
29.         int i;
30.         for(i=0; i<8; i++)
31.         {
```

라즈베리파이에서의 I2C 사용

● 8LED 제어 예제(1)

```
32.         // handle을 통해 출력 레지스터에 LED 데이터 값을 전송
33.         wiringPiI2CWriteReg16(fd, OUT_PORT1, aLedData[i]);
34.         delay(100);
35.     }
36. }
37. return 0;
38. }
```

- 작성 후 "ctrl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
- GCC 컴파일러를 사용하여 빌드 및 생성된 "09_8LED_01" 파일 실행

```
pi@raspberrypi:~/Example $ gcc -o 09_8LED_01 09_8LED_01.c -lwiringPi
pi@raspberrypi:~/Example $ ./09_8LED_01
```

라즈베리파이에서의 I2C 사용

- 8LED 제어 예제(1)
 - 결과
 - LED0~LED7 까지 순차적으로 점등



라즈베리파이에서의 I2C 사용

- 8LED 제어 예제(2)
 - 터미널 창에 “nano 09_8LED_02.c” 입력

```
pi@raspberrypi:~/Example $ nano 09_8LED_02.c
```

- SWITCH 2개를 이용해 SWITCH0을 누르면 LED0~ 7까지 순차적으로 점등되고, SWITCH1을 누르면 반대 방향으로 점등되는 예제

```
1. // File : 09_8LED_02.c
2. #include <stdio.h>
3. #include <wiringPi.h>
4. #include <wiringPiI2C.h>           // I2C 라이브러리 참조
5. #define LED_I2C_ADDR              0x20           // 8LED I2C 주소
6. // 제어 레지스터
7. #define IN_PORT0                   0x00
8. #define IN_PORT1                   0x01
9. #define OUT_PORT0                  0x02
10. #define OUT_PORT1                  0x03
11. #define POLARITY_IVE_PORT0         0x04
12. #define POLARITY_IVE_PORT1        0x05
13. #define CONFIG_PORT0               0x06
14. #define CONFIG_PORT1               0x07
```

라즈베리파이에서의 I2C 사용

● 8LED 제어 예제(2)

```
15. // 8개의 LED 데이터
16. const int aLedData[8] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};

17. const int aPinSwitch[2] = {6, 5};           // SWITCH핀 설정
18. int fd;                                       // 8LED의 handle
19. int interruptFlag = 0;                       // interruptFlag 선언

20. // 인터럽트 서비스 루틴 정의
21. void ledAscending(void)                     // LED0~7까지 정방향
22. {
23.     interruptFlag = 0;
24. }

25. void ledDescending(void)                   // LED7~0까지 역방향
26. {
27.     interruptFlag = 1;
28. }

29. int main(void)
30. {
31.     wiringPiSetupGpio();                     // 핀 번호를 BCM Mode로 설정
```

라즈베리파이에서의 I2C 사용

● 8LED 제어 예제(2)

```
32. // I2C 시스템 초기화 설정
33. if((fd = wiringPiI2CSetup(LED_I2C_ADDR)) < 0)
34. {
35.     return -1;                // 설정이 안될 경우, 종료
36. }

37. // handle을 통해 Configuration 레지스터를 출력모드로 설정
38. wiringPiI2CWriteReg16(fd, CONFIG_PORT1, 0x00);

39. int i;
40. for(i=0; i<2; i++)
41. {
42.     pinMode(aPinSwitch[i], INPUT);
43. }

44. // 인터럽트 설정
45. wiringPiISR(aPinSwitch[0], INT_EDGE_RISING, ledAscending);
46. wiringPiISR(aPinSwitch[1], INT_EDGE_RISING, ledDescending);

47. while(1)
48. {
49.     if(interruptFlag == 0)
50.     {
```


라즈베리파이에서의 I2C 사용

- 8LED 제어 예제(2)

```
51.         for(i=0; i<8; i++)
52.         {
53.             // handle을 통해 출력 레지스터에 LED 데이터 값을 전송
54.             wiringPiI2CWriteReg16(fd, OUT_PORT1, aLedData[i]);
55.             delay(100);
56.         }
57.     }
58.     else
59.     {
60.         for(i=7; i>=0; i--)
61.         {
62.             wiringPiI2CWriteReg16(fd, OUT_PORT1, aLedData[i]);
63.             delay(100);
64.         }
65.     }
66. }
67. return 0;
68. }
```

라즈베리파이에서의 I2C 사용

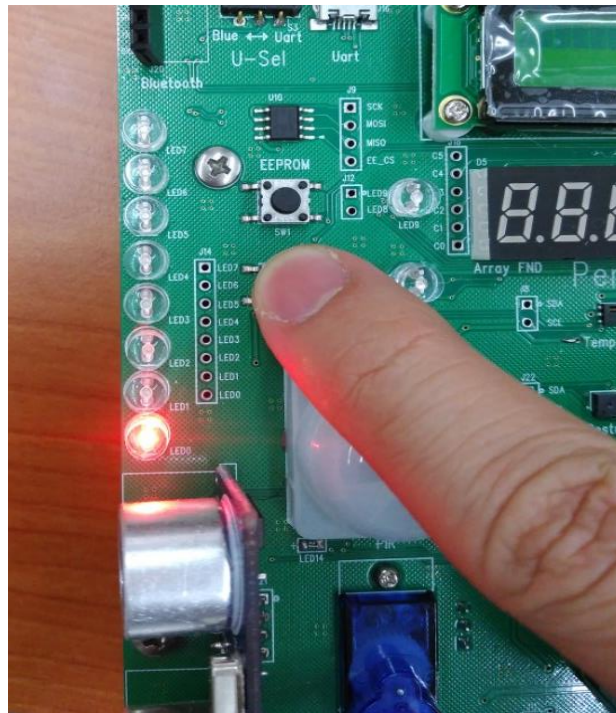
- 8LED 제어 예제(2)

- 작성 후 "ctrl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
- GCC 컴파일러를 사용하여 빌드 및 생성된 "09_8LED_02" 파일 실행

```
pi@raspberrypi:~/Example $ gcc -o 09_8LED_02 09_8LED_02.c -lwiringPi
pi@raspberrypi:~/Example $ ./09_8LED_02
```

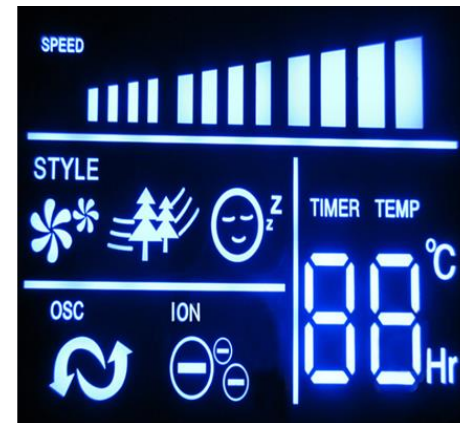
- 결과

- SWITCH에 의해 LED 방향이 결정되고 순차적으로 불빛이 켜짐



라즈베리파이에서의 I2C 사용

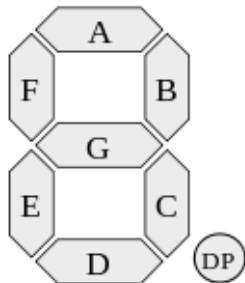
- FND 제어
 - FND(Flexible Numeberic Display)는 일종의 표시장치를 말함
 - 7-세그먼트(7-Segment)라고도 함
 - 7획으로 숫자나 문자를 나타낼 수 있음



라즈베리파이에서의 I2C 사용

● FND 제어

- 각 획은 맨 왼쪽 위쪽 가로 획부터 시계방향으로 A부터 마지막 가운데 가로 획까지 G라고 불림(DP는 경우에 따라 다름)
- 각 획에는 LED가 내장되어 있어 LED 점등에 따라 표시 가능
- 8개의 LED를 제어하여 숫자를 다음과 같이 표시



0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
또는	또는					또는	또는		또는
0	1					0	0		9

라즈베리파이에서의 I2C 사용

● FND 제어

- 7 세그먼트의 각 LED(비트)에 '0' 또는 '1'의 값을 입력하여 숫자 표시
- 예를 들어, FND에 숫자 '0'을 표시하기 위해선 A부터 F까지 '1' 입력 나머지는 '0' 입력
 - 2진수 : B00111111
 - 16진수 : 0x3F

표시	DP	G	F	E	D	C	B	A	2진수	16진수
0	0	0	1	1	1	1	1	1	0011 1111	0x3F
1	0	0	0	0	0	1	1	0	0000 0110	0x06
2	0	1	0	1	1	0	1	1	0101 1011	0x5B
3	0	1	0	0	1	1	1	1	0100 1111	0x4F
4	0	1	1	0	0	1	1	0	0110 0110	0x66
5	0	1	1	0	1	1	0	1	0110 1101	0x6D
6	0	1	1	1	1	1	0	1	0111 1101	0x7D
7	0	0	1	0	0	1	1	1	0010 0111	0x27
8	0	1	1	1	1	1	1	1	0111 1111	0x7F
9	0	1	1	0	1	1	1	1	0110 1111	0x6F

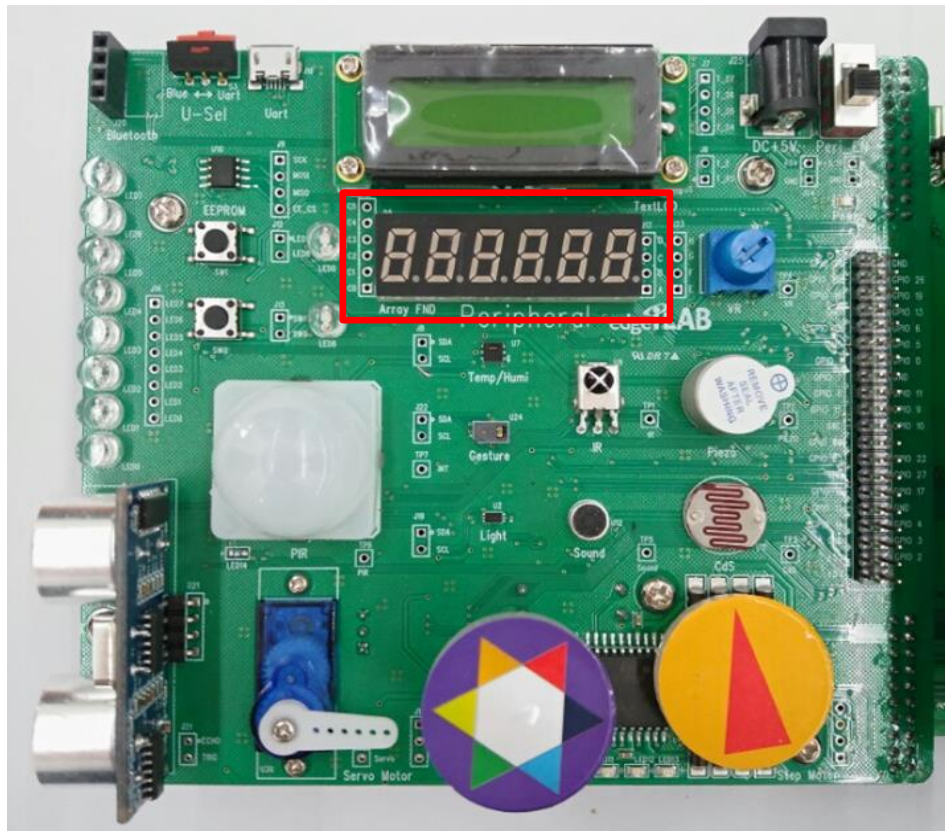
라즈베리파이에서의 I2C 사용

- FND 제어
 - 각 자릿수는 IO 0번 포트(2~7번 비트)의 SEG_C0~C5와 연결
 - '0'의 값을 줘야 해당 자릿수에 표시
 - 예를 들어, 첫 번째 자리만 표시를 하고 싶다면, 첫 번째 자리만 '0'을 입력하고, 나머지 자릿수는 '1'을 입력
 - 2진수 : B01111111
 - 16진수 : 0x7F

표시	C0	C1	C2	C3	C4	C5	-	-	2진수	16진수
8888888	0	1	1	1	1	1	1	1	0111 1111	0x7F
8888888	1	0	1	1	1	1	1	1	1011 1111	0xBF
8888888	1	1	0	1	1	1	1	1	1101 1111	0xDF
8888888	1	1	1	0	1	1	1	1	1110 1111	0xEF
8888888	1	1	1	1	0	1	1	1	1111 0111	0xF7
8888888	1	1	1	1	1	0	1	1	1111 1011	0xFB

라즈베리파이에서의 I2C 사용

- FND 제어
 - 6 Array FND 주소 : 0x21



라즈베리파이에서의 I2C 사용

- FND 제어 예제(1)
 - 터미널 창에 “nano 10_FND_01.c” 입력

```
pi@raspberrypi:~/Example $ nano 10_FND_01.c
```

- I2C 통신을 통해 1자리의 FND에 '0'~'9'을 출력하는 예제

```
1. // File : 10_FND_01.c
2. #include <stdio.h>
3. #include <wiringPi.h>
4. #include <wiringPiI2C.h>           // I2C 라이브러리 참조
5. #define FND_I2C_ADDR      0x21    // FND I2C 주소
6. // 제어 레지스터
7. #define IN_PORT0          0x00
8. #define IN_PORT1          0x01
9. #define OUT_PORT0         0x02
10. #define OUT_PORT1         0x03
11. #define POLARITY_IVE_PORT0 0x04
12. #define POLARITY_IVE_PORT1 0x05
13. #define CONFIG_PORT0      0x06
14. #define CONFIG_PORT1      0x07
```

라즈베리파이에서의 I2C 사용

● FND 제어 예제(1)

```
15. // FND 숫자 데이터(0~9) 초기화
16. const int aFndData[10] = {0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0x3E, 0xE0, 0xFE, 0xF6};

17. // FND 1의자리 데이터 초기화
18. const int fndSelect = 0xFB;

19. int fd;                                // FND의 handle

20. int main(void)
21. {

22.     // I2C 시스템 초기화 설정
23.     if( (fd = wiringPiI2CSetup(FND_I2C_ADDR)) < 0)
24.     {
25.         return -1;                        // 설정이 안될 경우, 종료
26.     }

27.     // handle을 통해 Configuration 레지스터를 출력모드로 설정
28.     // 레지스터는 한 쌍으로 이루어져 2byte데이터를 전송할 시,
29.     //정해진 레지스터에 1byte 저장하고, 나머지 1byte는 한 쌍의 다른 레지스터에 저장
30.     wiringPiI2CWriteReg16(fd, CONFIG_PORT0, 0x0000);
```

라즈베리파이에서의 I2C 사용

● FND 제어 예제(1)

```
30. while(1)
31. {
32.     int i;
33.     for(i=0; i<10; i++)
34.     {
35.         //OUT_PORT0 레지스터에는 fndSelect 데이터를 저장하고,
36.         //OUT_PORT1 레지스터에는 aFndData를 저장
37.         wiringPiI2CWriteReg16(fd, OUT_PORT0, (aFndData[i] << 8) | fndSelect);
38.         delay(1000);
39.     }
40. }
41. return 0;
42. }
```

- 작성 후 "ctrl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
- GCC 컴파일러를 사용하여 빌드 및 생성된 "10_FND_01" 파일 실행

```
pi@raspberrypi:~/Example $ gcc -o 10_FND_01 10_FND_01.c -lwiringPi
pi@raspberrypi:~/Example $ ./10_FND_01
```

라즈베리파이에서의 I2C 사용

- FND 제어 예제(1)
 - 결과
 - 6-Array FND의 1의 자리에 '0'~'9'까지 1초마다 순차적으로 출력



라즈베리파이에서의 I2C 사용

- FND 제어 예제(2)

- 터미널 창에 “nano 10_FND_02.c” 입력

```
pi@raspberrypi:~/Example $ nano 10_FND_02.c
```

- I2C 통신을 통해 6자리의 FND에 “123456”을 출력하는 예제
 - 한자리씩 값을 출력하고, 다음 자리의 값을 출력하기 직전에 짧은 시간 동안 지연시켜 잔상을 발생
 - 우리 눈에는 한번에 값의 뜨는 것처럼 착각하게 만들어야 함

```
1. // File : 10_FND_02.c
2. #include <stdio.h>
3. #include <wiringPi.h>
4. #include <wiringPiI2C.h>           // I2C 라이브러리 참조
5. #define FND_I2C_ADDR              0x21           // FND I2C 주소
6. // 제어 레지스터
7. #define IN_PORT0                   0x00
8. #define IN_PORT1                   0x01
9. #define OUT_PORT0                  0x02
10. #define OUT_PORT1                  0x03
```

라즈베리파이에서의 I2C 사용

● FND 제어 예제(2)

```
11. #define POLARITY_IVE_PORT0    0x04
12. #define POLARITY_IVE_PORT1    0x05
13. #define CONFIG_PORT0          0x06
14. #define CONFIG_PORT1          0x07

15. // FND 숫자 데이터(0~9) 초기화
16. const int aFndData[10] = {0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0x3E, 0xE0, 0xFE, 0xF6};

17. // FND 자리 데이터 초기화
18. const int aFndSelect[6] = {0x7E, 0xBF, 0xDF, 0xEF, 0xF7, 0xFB};

19. int fd;                                // FND의 handle

20. int main(void)
21. {
22.     // I2C 시스템 초기화 설정
23.     if((fd = wiringPiI2CSetup(FND_I2C_ADDR)) < 0)
24.     {
25.         return -1;                        // 설정이 안될 경우, 종료
26.     }
```

라즈베리파이에서의 I2C 사용

● FND 제어 예제(2)

```
27. // handle을 통해 Configuration 레지스터를 출력모드로 설정
28. wiringPiI2CWriteReg16(fd, CONFIG_PORT0, 0x0000);

29. int count = 0;

30. while(1)
31. {
32.     // 표현하려는 숫자와 자리를 temp변수에 저장
33.     int temp = (aFndData[count+1]<<8) | aFndSelect[count];

34.     // 저장된 temp 값을 출력 레지스터에 저장
35.     wiringPiI2CWriteReg16(fd, OUT_PORT0, temp);

36.     count++; // 표현하려는 숫자와 자리수 증가
37.     delay(1); // 잔상을 처리하기 위해 0.001초 지연

38.     if(count > 5) // 자리가 6번째를 넘어가면
39.         count = 0; // count 0으로 초기화
40. }
41. return 0;
42. }
```

라즈베리파이에서의 I2C 사용

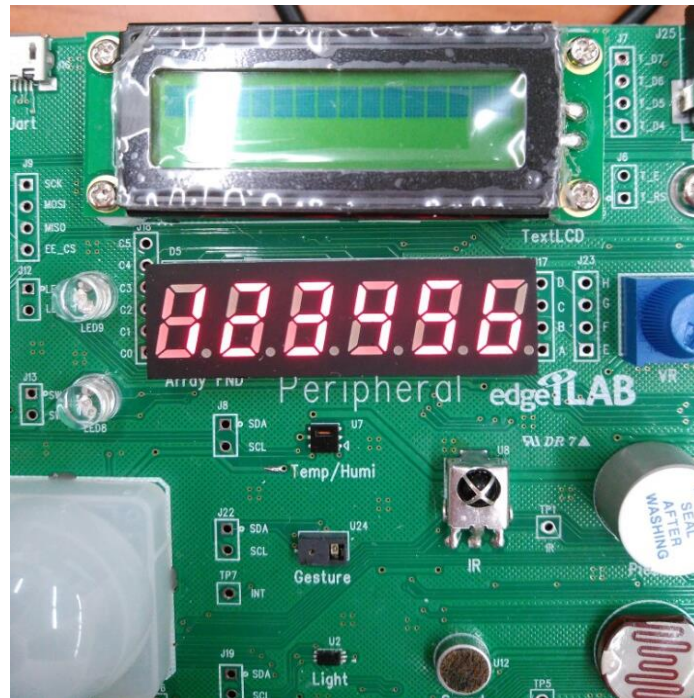
- FND 제어 예제(2)

- 작성 후 "ctrl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
- GCC 컴파일러를 사용하여 빌드 및 생성된 "10_FND_02" 파일 실행

```
pi@raspberrypi:~/Example $ gcc -o 10_FND_02 10_FND_02.c -lwiringPi
pi@raspberrypi:~/Example $ ./10_FND_02
```

- 결과

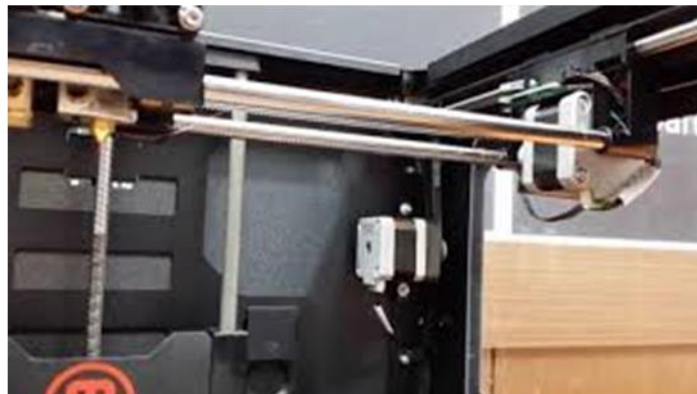
- 6-Array FND에 "123456" 출력



라즈베리파이에서의 I2C 사용

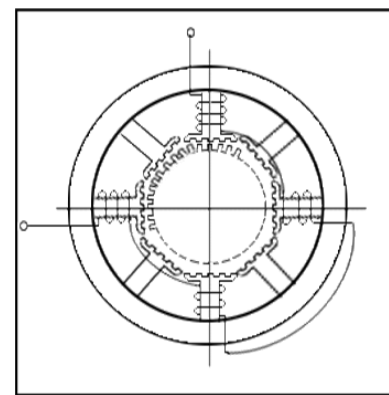
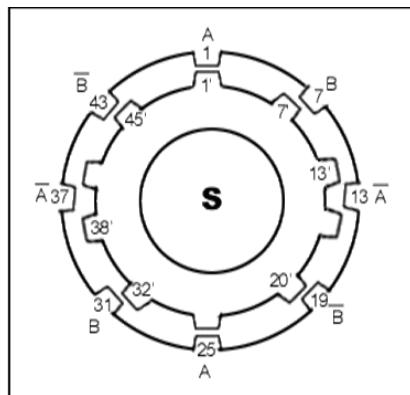
● STEP MOTOR 제어

- 스텝모터는 일정한 간격으로 외부의 DC전압 또는 전류의 양(펄스)을 조절하여 입력시켜 줌에 따라 일정한 각도로 회전을 하는 모터
- 디지털 제어방식의 기기로, 디지털 펄스형식의 제어에 적합
- 스텝 모터는 모터의 샤프트(shaft, 축)의 위치를 검출하기 위한 Feedback 없이, 정해진 각도를 회전하고, 상당히 높은 정확도로 정지 가능
- 정지 시 매우 큰 유지토크(정지토크)가 있기 때문에 전자 브레이크 등의 유지 기구를 필요로 하지 않음
- 회전 속도가 펄스의 속도에 비례해서 간편하게 제어 가능



라즈베리파이에서의 I2C 사용

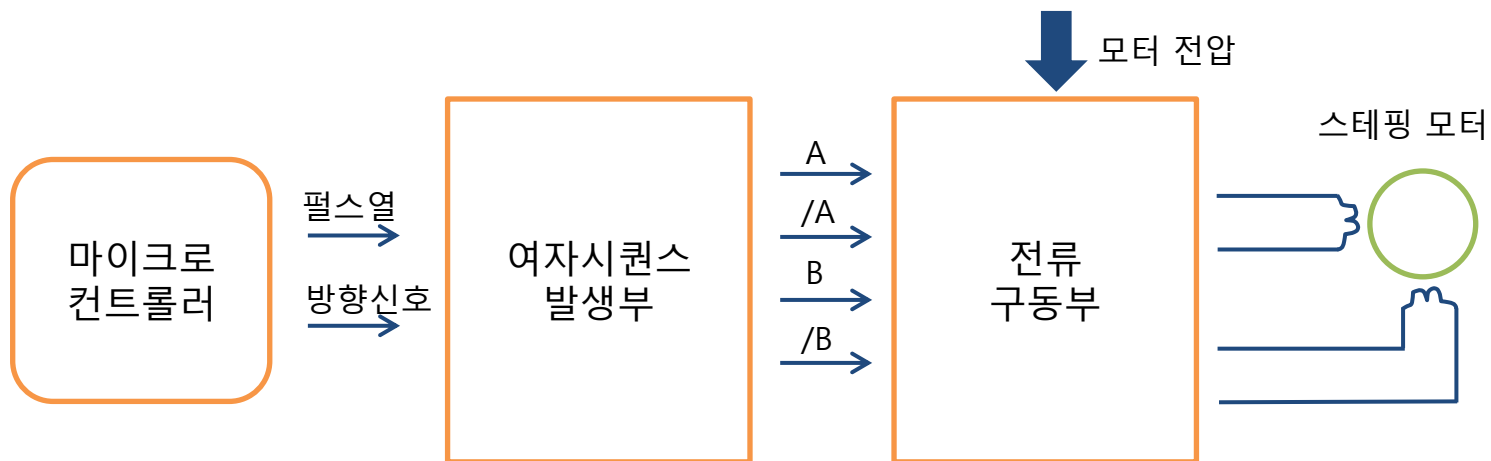
- STEP MOTOR 제어
 - 동작원리
 - 회전자(Rotor)둘레에 50개의 치(Tooth)가 있고, 고정자(Stator)의 내경에 48개의 치가 있는 모터의 경우
 - 고정자의 치-피치(Tooth-pitch, 간격) 7.5도($360/48 = 7.5$)
 - 회전자의 치-피치 7.2도($360/50 = 7.2$)
 - 따라서 A상을 여자(코일에 전류를 입력해서 자석을 만들어줌)시켰을 때 회전자 7번째와 고정자 7번째의 각도차는 $(7.5 - 7.2) * 6 = 1.8$ 도 이기 때문에 기본 회전 스텝각 1.8도
 - 회전방향을 반대로 하려면 상기의 여자 순서를 반대로 동작



라즈베리파이에서의 I2C 사용

● STEP MOTOR 제어

- 스텝모터의 제어회로는 펄스를 인가하여 각 상의 여자 신호를 발생시키는 여자 신호 발생부와 그 신호를 받아서 여자 전류를 흘려주기 위한 구동 회로부로 구성
- GPIO에서 방향 신호와 펄스열(array)을 인가하면, 여자 신호를 발생시켜 스텝핑 모터를 회전
- 여자신호를 소프트웨어적으로 발생시킬 수 있으면 여자 신호 발생부는 생략 가능
- 방식으로는 1상, 2상, 1-2상 여자 방식이 있음



라즈베리파이에서의 I2C 사용

- STEP MOTOR 제어
 - 1상 여자 방식
 - 스텝모터를 구동하기 위한 최소한의 구동방법
 - A -> B -> /A -> /B 의 순서로 1개의 코일만을 차례로 여자하는 방식
 - 소비전력이 낮고 1스텝당 정밀도가 높지만 감쇠 진동이 크고 탈조(펄스가 빨라지면 모터 축이 변화를 따라가지 못하고 멈춰있는 현상)하기 쉬움

구분	1	2	3	4	5	6	7	8	9
A	1	0	0	0	1	0	0	0	1
B	0	1	0	0	0	1	0	0	0
/A	0	0	1	0	0	0	1	0	0
/B	0	0	0	1	0	0	0	1	0

1주기

구분	1	2	3	4	5	6	7	8	9
A									
B									
/A									
/B									

라즈베리파이에서의 I2C 사용

● STEP MOTOR 제어

— 2상 여자 방식

- 항상 2상이 여자 되므로 기동 토크가 주어져 난조(축이 흔들리는 현상)가 일어나기 어려움
- 항상 2개의 상에서 전류가 흐르도록 해야 함
- 상 전환 시에도 반드시 1상은 여자되어 있으므로 동작 시에 제동 효과가 있음

구분	1	2	3	4	5	6	7	8	9
A	1	0	0	1	1	0	0	1	1
B	1	1	0	0	1	1	0	0	1
/A	0	1	1	0	0	1	1	0	0
/B	0	0	1	1	0	0	1	1	0

1주기

구분	1	2	3	4	5	6	7	8	9
A									
B									
/A									
/B									

라즈베리파이에서의 I2C 사용

● STEP MOTOR 제어

— 1-2상 여자 방식

- 하나의 상과 두 개의 상에 교대로 전류를 흐르게 하는 방식
- 1상 여자 구동에 비해 1.5배 전류가 필요
- 1, 2상 여자방식의 스텝 각이 1/2이므로 각도를 정밀하게 제어할 때 주로 사용

구분	1	2	3	4	5	6	7	8	9
A	1	1	0	0	0	0	0	1	1
B	0	1	1	1	0	0	0	0	0
/A	0	0	0	1	1	1	0	0	0
/B	0	0	0	0	0	1	1	1	0

1주기

구분	1	2	3	4	5	6	7	8	9
A									
B									
/A									
/B									

-

라즈베리파이에서의 I2C 사용

- STEP MOTOR 제어 예제(1)
 - 터미널 창에 “nano 11_STEP_01.c” 입력

```
pi@raspberrypi:~/Example $ nano 11_STEP_01.c
```

- Step 모터를 1상 여자 방식으로 정방향 회전시키는 예제

```
1. // File : 11_STEP_01.c
2. #include <stdio.h>
3. #include <wiringPi.h>
4. #include <wiringPiI2C.h>                // I2C 라이브러리 참조
5. #define STEP_I2C_ADDR          0x20      // STEP I2C 주소
6. // 제어 레지스터
7. #define IN_PORT0                0x00
8. #define IN_PORT1                0x01
9. #define OUT_PORT0               0x02
10. #define OUT_PORT1              0x03
11. #define POLARITY_IVE_PORT0     0x04
12. #define POLARITY_IVE_PORT1    0x05
13. #define CONFIG_PORT0           0x06
14. #define CONFIG_PORT1          0x07
```

라즈베리파이에서의 I2C 사용

● STEP MOTOR 제어 예제(1)

```
15. // 1상 여자 방식 회전 데이터(A, B, /A, /B)
16. const int aPhase_1[4] = {0x80, 0x40, 0x20, 0x10};

17. int fd;                                // Step Motor의 handle

18. int main(void)
19. {
20.     // I2C 시스템 초기화 설정
21.     if((fd = wiringPiI2CSetup(STEP_I2C_ADDR)) < 0)
22.     {
23.         return -1;                        // 설정이 안될 경우, 종료
24.     }

25.     // handle을 통해 Configuration 레지스터를 출력모드로 설정
26.     wiringPiI2CWriteReg16(fd, CONFIG_PORT0, 0x00);

27.     int i;

28.     while(1)
29.     {
```

라즈베리파이에서의 I2C 사용

● STEP MOTOR 제어 예제(1)

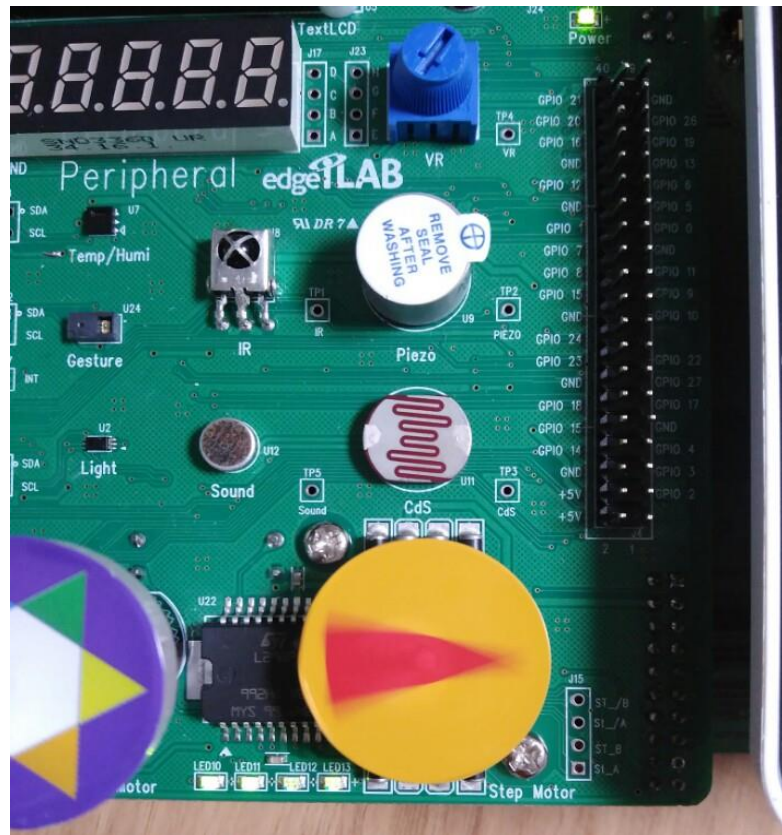
```
30. // 순차적으로 Step Motor의 위상을 조절
31. for(i=0; i<4; i++)
32. {
33.     // A, B, /A, /B 상에 '1'의 상태로 만들어 회전시킴
34.     wiringPiI2CWriteReg16(fd, OUT_PORT0, aPhase_1[i]);
35.     delay(10);
36. }
37. }
38. return 0;
39. }
```

- 작성 후 "ctrl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
- GCC 컴파일러를 사용하여 빌드 및 생성된 "11_STEP_01" 파일 실행

```
pi@raspberrypi:~/Example $ gcc -o 11_STEP_01 11_STEP_01.c -lwiringPi
pi@raspberrypi:~/Example $ ./11_STEP_01
```

라즈베리파이에서의 I2C 사용

- STEP MOTOR 제어 예제(1)
 - 결과
 - Step 모터가 1상 여자 방식으로 반시계 방향 회전



라즈베리파이에서의 I2C 사용

- STEP MOTOR 제어 예제(2)
 - 터미널 창에 "nano 11_STEP_02.c" 입력

```
pi@raspberrypi:~/Example $ nano 11_STEP_02.c
```

- Interrupt를 이용해 SWITCH를 누르면 Step모터의 방식(2상 여자 방식: 반시계 방향, 1-2상 여자방식: 시계방향)을 바꿔 회전시키는 예제

```
1. // File : 11_STEP_02.c
2. #include <stdio.h>
3. #include <wiringPi.h>
4. #include <wiringPiI2C.h>                // I2C 라이브러리 참조
5. #define STEP_I2C_ADDR          0x20      // STEP I2C 주소
6. // 제어 레지스터
7. #define IN_PORT0                0x00
8. #define IN_PORT1                0x01
9. #define OUT_PORT0               0x02
10. #define OUT_PORT1              0x03
11. #define POLARITY_IVE_PORT0      0x04
12. #define POLARITY_IVE_PORT1      0x05
13. #define CONFIG_PORT0           0x06
14. #define CONFIG_PORT1           0x07
```

라즈베리파이에서의 I2C 사용

● STEP MOTOR 제어 예제(2)

```
15. const int aPinSwitch[2] = {6, 5};

16. // 2상 여자 방식 회전 데이터(A, B, /A, /B)
17. const int aPhase_2[4] = {0xC0, 0x60, 0x30, 0x90};

18. // 1-2상 여자 방식 회전 데이터(A, B, /A, /B)
19. const int aPhase_12[8] = {0x10, 0x30, 0x20, 0x60, 0x40, 0xC0, 0x80, 0x90};

20. int fd;                                // Step Motor의 handle

21. int interruptFlag = 0;

22. // 인터럽트 서비스 루틴 정의
23. void changePhaseTo2(void)
24. {
25.     interruptFlag = 0;
26. }

27. void changePhaseTo12(void)
28. {
29.     interruptFlag = 1;
30. }
```

라즈베리파이에서의 I2C 사용

● STEP MOTOR 제어 예제(2)

```
31. int main(void)
32. {
33.     // I2C 시스템 초기화 설정
34.     if((fd = wiringPiI2CSetup(STEP_I2C_ADDR)) < 0)
35.     {
36.         return -1;                // 설정이 안될 경우, 종료
37.     }

38.     wiringPiSetupGpio();           // 핀 번호를 BCM Mode로 설정

39.     int i;
40.     for(i=0; i<2; i++)
41.     {
42.         pinMode(aPinSwitch[i], INPUT); // SWITCH 핀을 입력 모드로 설정
43.     }

44.     // handle을 통해 Configuration 레지스터를 출력모드로 설정
45.     wiringPiI2CWriteReg16(fd, CONFIG_PORT0, 0x00);

46.     // 인터럽트 설정
47.     wiringPiISR(aPinSwitch[0], INT_EDGE_RISING, changePhaseTo2);
48.     wiringPiISR(aPinSwitch[1], INT_EDGE_RISING, changePhaseTo12);
```

라즈베리파이에서의 I2C 사용

● STEP MOTOR 제어 예제(2)

```
49. while(1)
50. {
51.     if(interrupt == 0)                // 인터럽트 플래그가 '0'일 경우
52.     {
53.         for(i=0; i<4; i++)
54.         {
55.             // 2상 여자 방식으로 반시계방향 회전
56.             wiringPiI2CWriteReg16(fd, OUT_PORT0, aPhase2[i]);
57.             delay(10);
58.         }
59.     }
60.     else                                // 인터럽트 플래그가 '1'일 경우
61.     {
62.         for(i=7; i>=0; i--)
63.         {
64.             // 1-2상 여자 방식으로 시계방향 회전
65.             wiringPiI2CWriteReg16(fd, OUT_PORT0, aPhase12[i]);
66.             delay(10);
67.         }
68.     }
69. }
70. return 0;
71. }
```


라즈베리파이에서의 I2C 사용

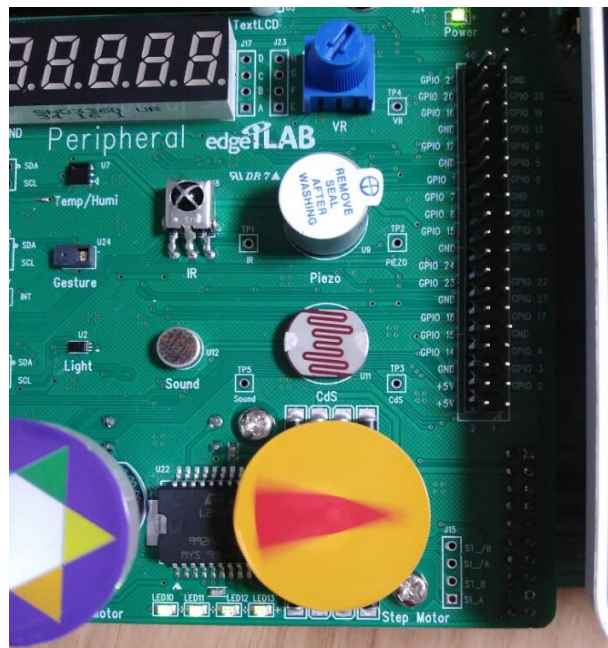
- STEP MOTOR 제어 예제(2)

- 작성 후 "ctrl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
- GCC 컴파일러를 사용하여 빌드 및 생성된 "11_STEP_02" 파일 실행

```
pi@raspberrypi:~/Example $ gcc -o 11_STEP_02 11_STEP_02.c -lwiringPi
pi@raspberrypi:~/Example $ ./11_STEP_02
```

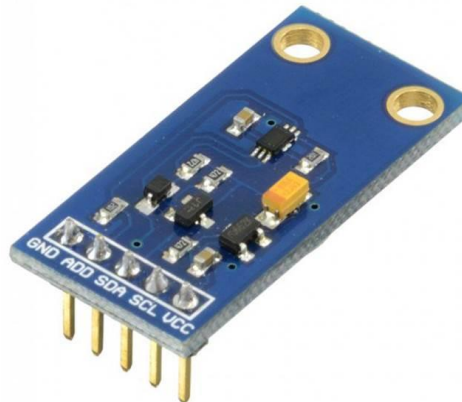
- 결과

- SWITCH1을 누르면 2상 여자 방식으로 반시계 방향 회전
- SWITCH2를 누르면 1-2상 여자 방식으로 시계 방향 회전



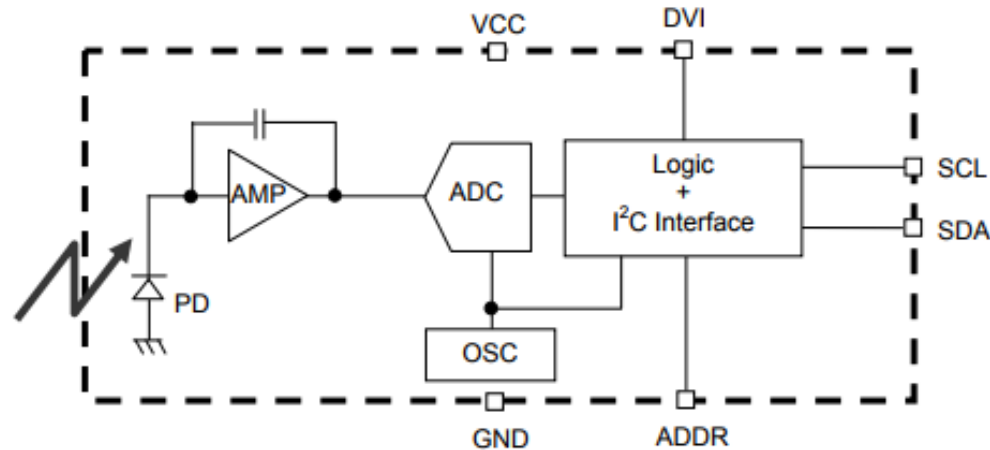
라즈베리파이에서의 I2C 사용

- LIGHT(광) 센서 제어
 - 광 센서는 주변의 광량을 측정하는 디지털 센서
 - 센서 안에 부착된 ROHM사의 BH1750FVI 은 I2C 인터페이스 용 디지털 IC 칩
 - 측정범위는 0~65535 lux(빛의 조명도를 나타내는 단위로 1럭스는 1칸델라의 광 원으로부터 1m 떨어진 곳에 광원과 직각으로 놓인 면의 밝기)
 - 고해상도의 넓은 범위를 감지
 - 주로 휴대 전화의 LCD 및 Keypad BackLight의 전원을 조정하기 위한 주변 광량 데이터를 얻는데 적합



라즈베리파이에서의 I2C 사용

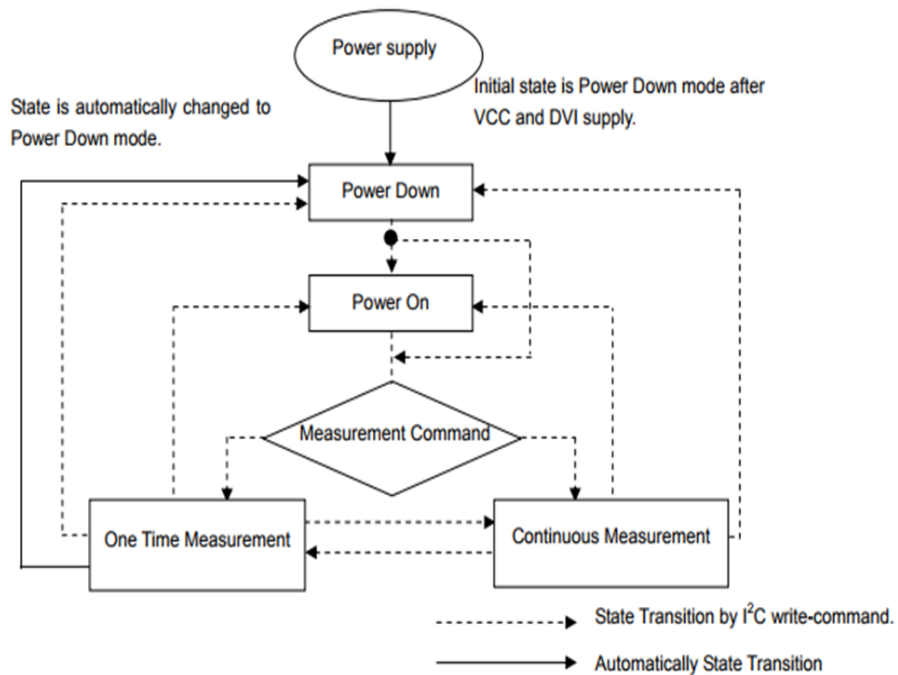
● LIGHT(광) 센서 제어



디바이스	기능
PD	포토 다이오드
AMP	PD의 전류를 전압으로 변환하기 위한 OPAMP
ADC	16비트 아날로그-디지털 변환기
Logic + I2C interface	주변 광 계산 및 I2C 인터페이스, 데이터 레지스터 -> 주변 광 데이터의 등록을 위한 레지스터, 초기값은 0000_0000_0000_0000 측정 시간 레지스터 -> 측정 시간의 등록을 위한 레지스터, 초기값은 0100_0101
OSC	내부 오실레이터

라즈베리파이에서의 I2C 사용

- LIGHT(광) 센서 제어
 - 슬레이브의 주소는 ADDR터미널(Vcc값)에 의해 결정
 - ADDR = 'H'(ADDR \geq 0.7 Vcc) -> '1011100'
 - ADDR = 'L'(ADDR \leq 0.3 Vcc) -> '0100011'



명령어	Code	설 명
Power Down	0000_0000	비활성화 상태
Power On	0000_0001	측정 명령 대기
Reset	0000_0111	데이터 레지스터 초기화
Continuously H-Resolution Mode	0001_0000	1x의 해상도에서 측정 측정시간 120ms
One Time H-Resolution Mode	0010_0000	1x의 해상도에서 측정 측정 시간 120ms 측정 후, Power Down 모드로 돌입

라즈베리파이에서의 I2C 사용

- LIGHT(광) 센서 제어
 - Continuously H-Resolution 모드 측정 진행
 - 마스터에서 슬레이브의 주소(ADDR='L')로 "Continuously H-resolution Mode"을 명령
 - "Continuously H-resolution Mode"가 완료될 때까지 최대 180ms 대기
 - 슬레이브에서 마스터로 2Byte(High, Low Byte)의 데이터 값 전송
 - 만약 측정한 데이터의 값이 High Byte = "00000001", Low Byte = "00010000"이면 $(2^8 + 2^4)/1.2 \approx 227[\text{lx}]$

ex1) Continuously H-resolution mode (ADDR = 'L')

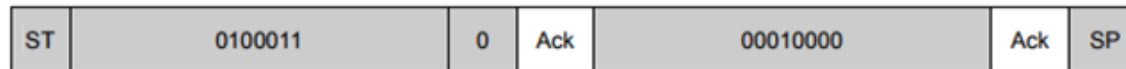


from Master to Slave



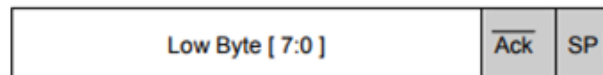
from Slave to Master

① Send "Continuously H-resolution mode " instruction



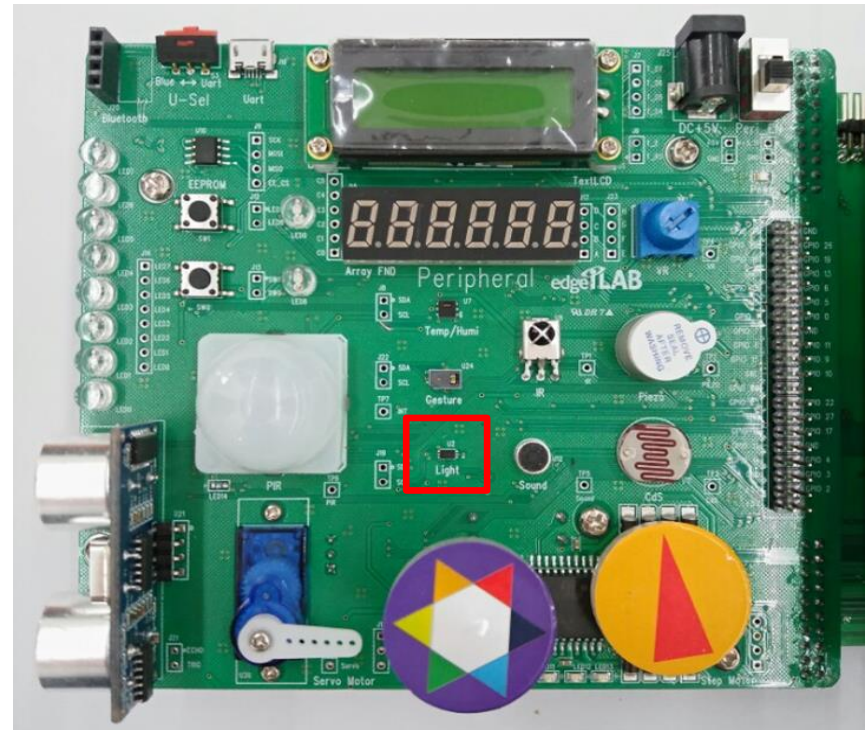
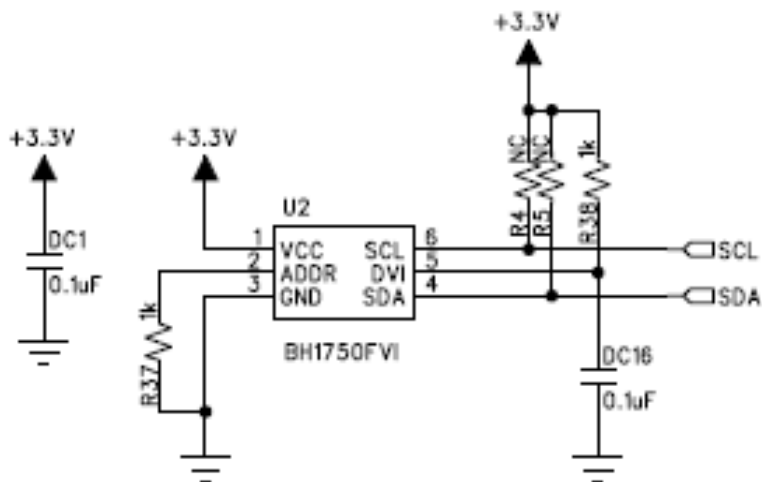
② Wait to complete 1st H-resolution mode measurement.(max. 180ms.)

③ Read measurement result.



라즈베리파이에서의 I2C 사용

- LIGHT(광) 센서 제어
 - Edge-Embedded Light 센서 회로도
 - Light 센서 주소 : 0x23



라즈베리파이에서의 I2C 사용

- LIGHT(광) 센서 제어 예제(1)
 - 터미널 창에 "nano 12_LIGHT_01.c" 입력

```
pi@raspberrypi:~/Example $ nano 12_LIGHT_01.c
```

- Continuously H-Resolution Mode를 이용해 현재의 광량을 측정하는 예제

```
1. // File : 12_LIGHT_01.c
2. #include <stdio.h>
3. #include <wiringPi.h>
4. #include <wiringPiI2C.h>           // I2C 라이브러리 참조
5. #define LIGHT_I2C_ADDR            0x23   // LIGHT I2C 주소
6. // 명령 코드
7. #define LIGHT_I2C_POWER_DOWN      0x00
8. #define LIGHT_I2C_POWER_ON        0x01
9. #define LIGHT_I2C_RESET            0x07
10. #define LIGHT_I2C_CON_HR_MODE     0x10
11. int fd;                          // Light 센서의 handle
```

라즈베리파이에서의 I2C 사용

● LIGHT(광) 센서 제어 예제(1)

```
1. int main(void)
2. {
3.     float lightValue = 0.0;
4.     int aData[2];
5.     int i;
6.     int value;

7.     // I2C 시스템 초기화 설정
8.     if((fd = wiringPiI2CSetup(LIGHT_I2C_ADDR)) < 0)
9.     {
10.         return -1;                // 설정이 안될 경우, 종료
11.     }

12.    // LIGHT 센서 초기화 설정
13.    wiringPiI2CWrite(fd, LIGHT_I2C_RESET);
14.    delay(50);

15.    while(1)
16.    {
17.        // 측정 모드 설정
18.        wiringPiI2CWrite(fd, LIGHT_I2C_CON_HR_MODE);
19.        delay(260);
```


라즈베리파이에서의 I2C 사용

● LIGHT(광) 센서 제어 예제(1)

```
20.     for(i=0; i<2; i++)
21.     {
22.         aData[i] = wiringPiI2CRead(fd); // LIGHT 센서로부터 측정된 데이터를 저장
23.     }

24.     // 측정된 데이터를 계산
25.     value = (data[0]*256) + data[1];
26.     lightValue = (int)value / 1.2;

27.     printf("LIGHT : %.1fWn", (float)lightValue);
28.     delay(1000);
29. }
30. return 0;
31. }
```

- 작성 후 "ctrl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
- GCC 컴파일러를 사용하여 빌드 및 생성된 "12_LIGHT_01" 파일 실행

```
pi@raspberrypi:~/Example $ gcc -o 12_LIGHT_01 12_LIGHT_01.c -lwiringPi
pi@raspberrypi:~/Example $ ./12_LIGHT_01
```

라즈베리파이에서의 I2C 사용

- LIGHT(광) 센서 제어 예제(1)
 - 결과
 - 현재의 광량을 측정하여 터미널 화면에 데이터 출력

```
pi@raspberrypi:~/Example $ ./12_LIGHT_01
LIGHT : 443.3
LIGHT : 5320.0
LIGHT : 3990.0
LIGHT : 3768.3
LIGHT : 3768.3
```

라즈베리파이에서의 I2C 사용

- LIGHT(광) 센서 제어 예제(2)
 - 터미널 창에 "nano 12_LIGHT_02.c" 입력

```
pi@raspberrypi:~/Example $ nano 12_LIGHT_02.c
```

- 현재의 광량을 측정하여 FND로 표시하는 예제

```
1. // File : 12_LIGHT_01.c
2. #include <stdio.h>
3. #include <wiringPi.h>
4. #include <wiringPiI2C.h>           // I2C 라이브러리 참조
5. #define LIGHT_I2C_ADDR             0x23       // LIGHT I2C 주소
6. // 명령 코드
7. #define LIGHT_I2C_POWER_DOWN      0x00
8. #define LIGHT_I2C_POWER_ON        0x01
9. #define LIGHT_I2C_RESET           0x07
10. #define LIGHT_I2C_CON_HR_MODE     0x10
11. #define FND_I2C_ADDR              0x21       // FND I2C 주소
```

라즈베리파이에서의 I2C 사용

- LIGHT(광) 센서 제어 예제(2)

```
12. // 제어 레지스터
13. #define OUT_PORT0          0x02
14. #define CONFIG_PORT0       0x06

15. // FND 숫자 데이터(0~9) 초기화
16. const int aFndData[10] = {0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0x3E, 0xE0, 0xFE, 0xF6};

17. // FND 자리 데이터 초기화
18. const int aFndSelect[6] = {0x7E, 0xBF, 0xDF, 0xEF, 0xF7, 0xFB};

19. int fd_FND;                // FND의 handle

20. int fd_LIGHT;              // Light 센서의 handle

21. // FND Display함수 정의
22. void displayNumber(float number)
23. {
24.     int i;
25.     int aPosition[6] = {0, };

26.     printf("%.2f\n", number);
```

라즈베리파이에서의 I2C 사용

- LIGHT(광) 센서 제어 예제(2)

```
27. // 자리수 표현
28. aPosition[0] = (int)number/1000;           // 1000의 자리
29. aPosition[1] = (int)number%1000/100;       // 100의 자리
30. aPosition[2] = (int)number%1000%100/10;    // 10의 자리
31. aPosition[3] = (int)number%10;             // 1의 자리
32. aPostion[4] = (int)(number*10)%10;          // 소수점 첫째자리
33. aPostion[5] = (int)(number*100)%10;        // 소수점 둘째자리

34. for(i=0; i<6; i++)
35. {
36.     // 임시 변수에 표현할 숫자와 자리를 저장
37.     int temp = (aFndData[aPosition[i]] << 8) | aFndSelect[i];

38.     if(i == 3)
39.     {
40.         temp = temp | 0x0100;                // 소수점 자리 표현
41.     }

42.     // 저장된 temp 값을 출력 레지스터에 저장
43.     wiringPiI2CWriteReg16(fd_FND, OUT_PORT0, temp);
44.     delay(1);
45. }
46. }
```

라즈베리파이에서의 I2C 사용

- LIGHT(광) 센서 제어 예제(2)

```
47. int main(void)
48. {
49.     float lightValue = 0.0;
50.     int aData[2];
51.     int i;
52.     int value;

53.     // I2C 시스템 초기화 설정
54.     if((fd_LIGHT = wiringPiI2CSetup(LIGHT_I2C_ADDR)) < 0)
55.     {
56.         return -1;                // 설정이 안될 경우, 종료
57.     }

58.     if((fd_FND = wiringPiI2CSetup(FND_I2C_ADDR)) < 0)
59.     {
60.         return -1;                // 설정이 안될 경우, 종료
61.     }

62.     // handle을 통해 Configuration 레지스터를 출력모드로 설정
63.     wiringPiI2CWriteReg16(fd_FND, CONFIG_PORT0, 0x0000);
```

라즈베리파이에서의 I2C 사용

● LIGHT(광) 센서 제어 예제(2)

```
64. // LIGHT 센서 초기화 설정
65. wiringPiI2CWrite(fd_LIGHT, LIGHT_I2C_RESET);
66. delay(50);

67. // 측정 모드 설정
68. wiringPiI2CWrite(fd_LIGHT, LIGHT_I2C_CON_HR_MODE);
69. delay(260);

70. while(1)
71. {
72.     for(i=0; i<2; i++)
73.     {
74.         aData[i] = wiringPiI2CRead(fd_LIGHT); // 측정된 데이터를 저장
75.     }

76.     // 측정된 데이터를 계산
77.     value = (data[0]*256) + data[1];
78.     lightValue = (float)value / 1.2;

79.     displayNumber(lightValue);
80. }
81. return 0;
82. }
```

라즈베리파이에서의 I2C 사용

- LIGHT(광) 센서 제어 예제(2)

- 작성 후 "ctrl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
- GCC 컴파일러를 사용하여 빌드 및 생성된 "12_LIGHT_02" 파일 실행

```
pi@raspberrypi:~/Example $ gcc -o 12_LIGHT_02 12_LIGHT_02.c -lwiringPi
pi@raspberrypi:~/Example $ ./12_LIGHT_02
```

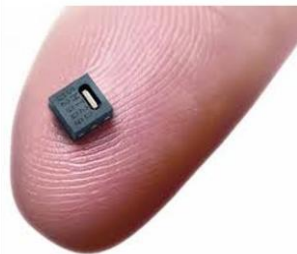
- 결과

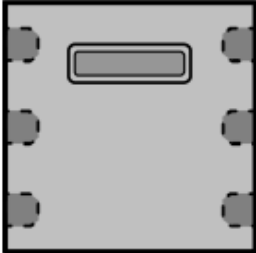
- 현재 광량을 측정하여 6Array FND에 출력



라즈베리파이에서의 I2C 사용

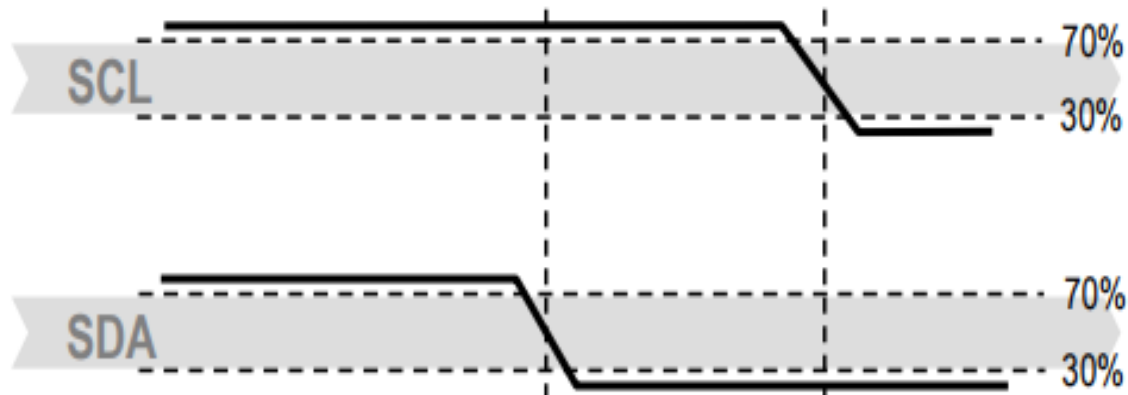
- Temp/Humi(온습도) 센서 제어
 - SHT20
 - 상대습도 및 상대 온도 측정
 - 이슬점(Dew Point)측정 가능
 - I2C 인터페이스를 사용하여 최대 400kHz의 SCL 주파수 사용 가능
 - 섭씨 -40도 ~ 105도, 상대습도 10~95% 측정 가능
 - 온도 오차 $\pm 0.5^{\circ}\text{C}$, 습도 오차 $\pm 3.0\%\text{RH}$
 - 6개의 핀으로 동작



Pin	Name	Comment		
1	SDA	Serial Data, bidirectional		
2	VSS	Ground		
5	VDD	Supply Voltage		
6	SCL	Serial Clock, bidirectional		
3,4	NC	Not Connected		

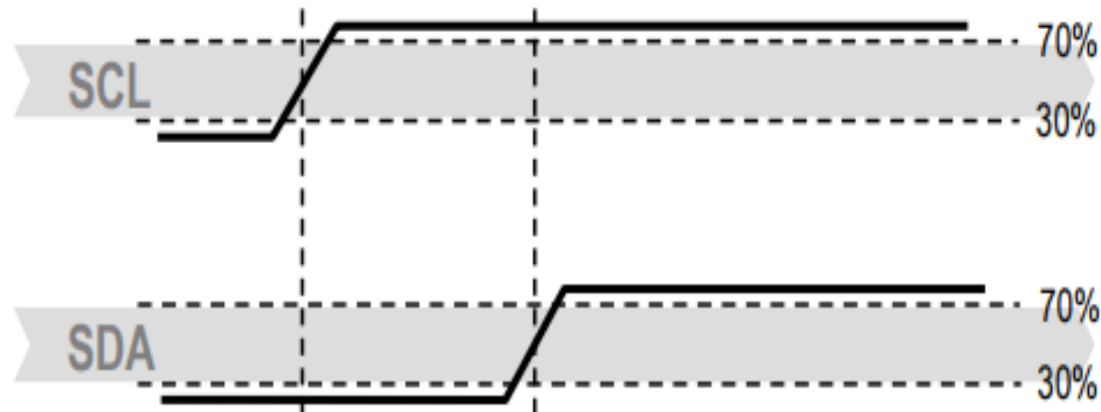
라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어
 - 센서 시작
 - VDD에 전역 공급
 - 마스터(MCU)에서 명령을 받을 수 있도록 SCL이 High가 될 때까지(IDLE상태) 15ms 대기
 - 센서는 측정이나 통신을 수행하지 않을 경우, 자동으로 IDLE상태 전환
 - 시작 신호
 - 시작 신호(S)(SCL이 High인 상태일 때, SDA를 High에서 Low로 상태를 변화)를 발생시켜 전송의 시작을 알림
 - 시작 조건은 마스터에 의해서 생성된 버스의 고유한 상태로, 전송 순서의 시작을 슬레이브에게 알림



라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어
 - 정지 신호
 - 정지 신호(P)(SCL이 High인 상태일 때, SDA를 Low에서 High로 상태를 변화)를 발생시켜 전송의 끝을 알림
 - 정지 조건은 마스터에 의해서 생성된 버스의 고유한 상태로, 전송 순서의 끝을 슬레이브에게 알림



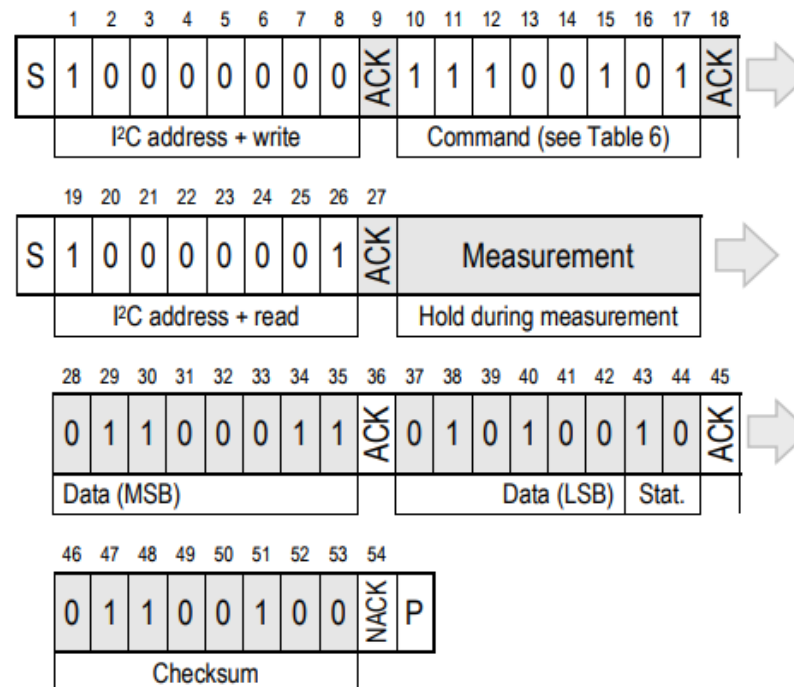
라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어
 - 명령 전달
 - 시작 조건을 전송한 후, I2C 헤더는 7비트 I2C 디바이스의 주소 '1000'000'과 SDA의 방향 비트(Read R : '1', Write W : '0')으로 구성
 - 측정 명령(온도의 경우, '1110'0011, 습도의 경우 '1110'0101')이 발생한 후, MCU는 측정이 완료될 때까지 대기

Command	Code	Comment
Trigger T measurement	1111 0011	Hold Master
Trigger RH measurement	1110 0101	Hold Master
Trigger T measurement	1111 0011	No Hold Master
Trigger RH measurement	1111 0101	No Hold Master
Write user register	1110 0110	
Read user register	1110 0111	
Soft reset	1111 1110	

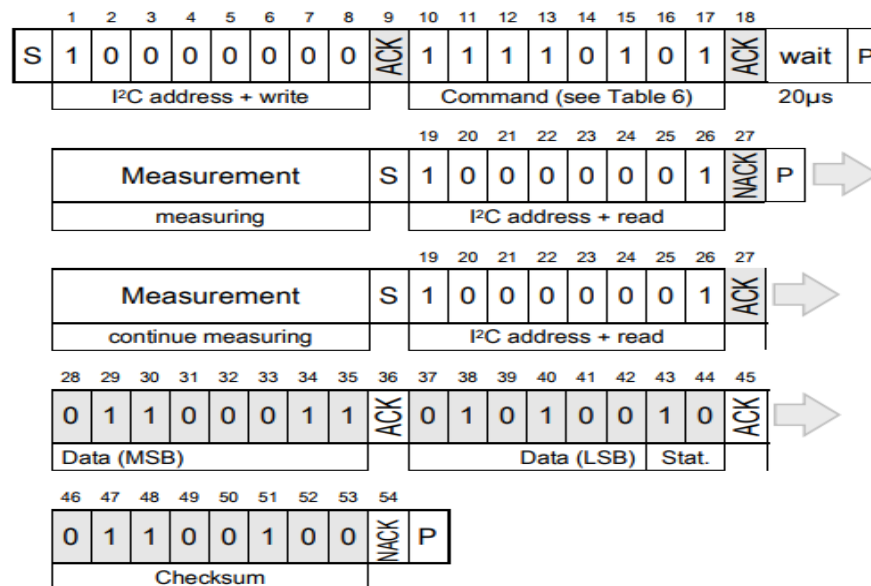
라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어
 - Hold / No Hold Master Mode
 - Hold Master
 - 측정 중일 때 SCL 이 차단
 - SCL을 풀다운(차단)하면서 마스터를 대기 상태로 전환
 - SCL이 해제하면 센서가 내부 처리가 종료되므로, 전송 진행 가능



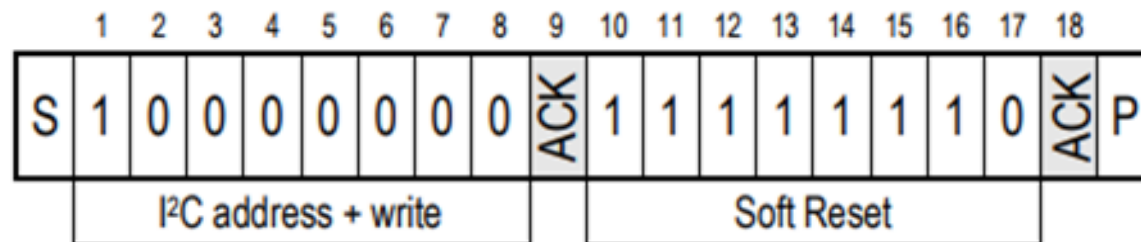
라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어
 - Hold / No Hold Master Mode
 - No Hold Master
 - SCL 이 다른 통신을 위해 열린 상태로 유지
 - MCU 센서의 내부 프로세싱 종료를 위해 폴링
 - 내부 프로세싱이 끝나면, 센서는 MCU의 폴링을 확인 후 데이터 읽음
 - 측정이 완료되지 않은 경우, 센서는 ACK 비트에 응답하지 않고 시작 조건을 다시 한번 발생
 - 센서의 ACK 비트(18비트) 수신 후와 정지 조건 사이에 20us의 대기시간을 포함하는 것을 권장



라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어
 - Hold / No Hold Master Mode
 - 두 모드 최대 분해능이 14비트이기 때문에, 43과 44 비트는 상태 정보를 전송하는데 사용
 - 이 비트 중 43비트는 측정 유형('0': 온도, '1': 습도)를 나타내고, 44 비트는 할당되지 않음
 - Soft Reset
 - 이 명령은 전원 On/Off하지않고 센서 시스템을 재 부팅하는데 사용
 - 이 명령을 수신하면 센서 시스템은 사용자 레지스터의 Heat Bit를 제외하고 기본설정에 따라 다시 초기화
 - 소요시간은 약 15ms 미만



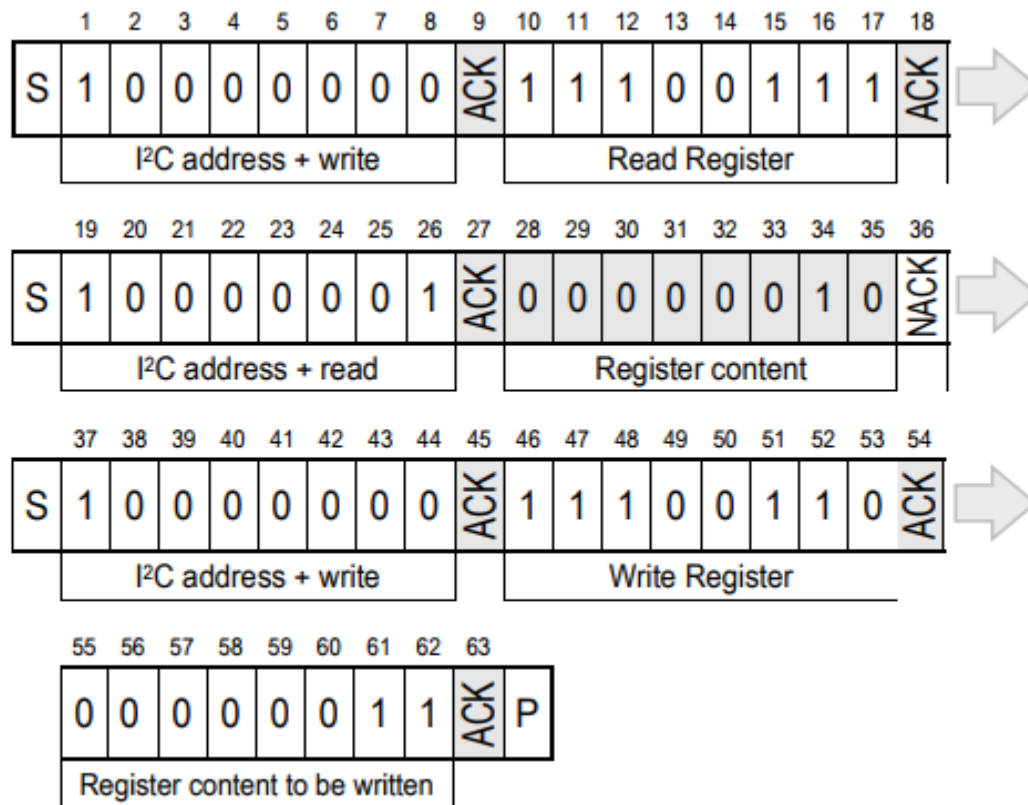
라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어
 - User Register
 - Reserved 비트는 변경하지 말아야 하므로, 사용자 레지스터에 기록할 때에는 reserved 비트의 기본 값을 먼저 읽어야 함
 - 6비트는 측정할 때마다 배터리의 양을 체크하여서 2.25V 이하가 되면 '1'로 Set
 - 2비트는 온도 상승 시 상대 습도가 저하되는 특성을 위해 사용
 - 1비트는 안전 기능이며, 모든 측정 전에 히터 비트(2비트)를 제외하고 전체 OTP 설정을 레지스터에 읽어옴

Bit	# Bits	Description / Coding	Default
7, 0	2		'00'
		RH	
		T	
		'00'	
		'01'	
		'10'	12bit
		'11'	
6	1	Status : End of Battery '0' : VDD > 2.25V '1' : VDD < 2.25V	'0'
3,4,5	3	Reserved	
2	1	Enable on-chip heater	'0'
1	1	Disable OTP Reload	'1'

라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어
 - Read & Write 레지스터 사용 예



라즈베리파이에서의 I2C 사용

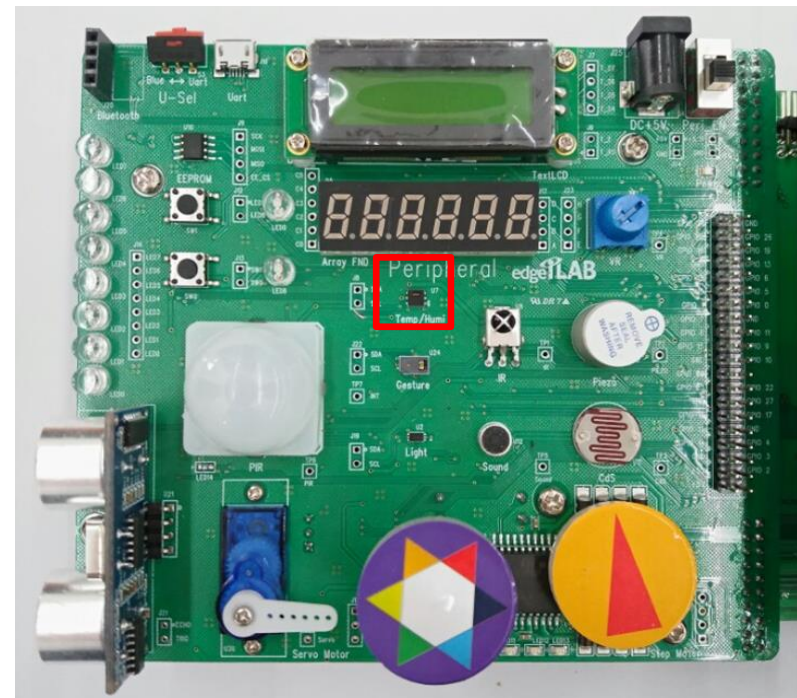
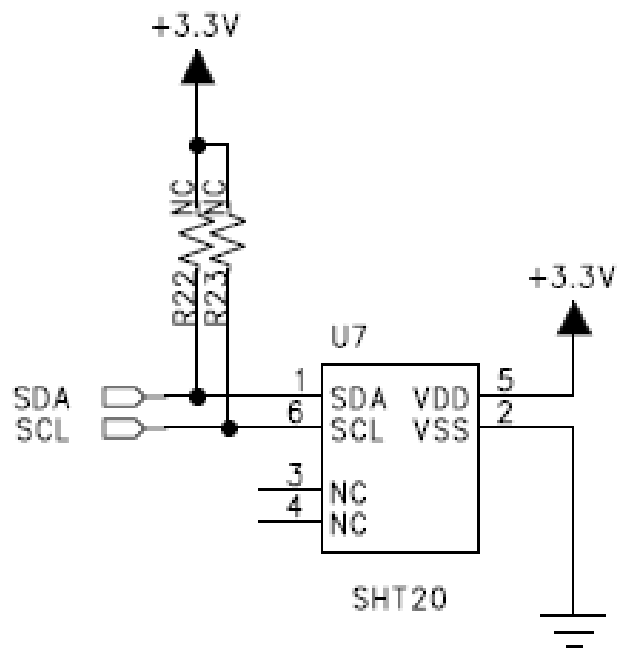
- Temp/Humi(온습도) 센서 제어
 - Read & Write 레지스터 사용 예
 - 계산 방법
 - 기본 해상도는 12비트 상대습도 및 14비트 온도 읽기 값으로 설정
 - 측정된 데이터는 2바이트 패키지, 최상위 비트(MSB)가 먼저 전송되는 프레임(8비트)으로 전송
 - 각 바이트는 ACK 비트에 따라 옴
 - 최하위 비트(LSB)의 마지막 비트 2개는 계산하기 전에 '0'으로 설정
 - 공식

$$RH = -6 + 125 \times \frac{S_{RH}}{2^{16}} \quad [\%], \quad (S_{RH} : \text{Relative humidity signal output})$$

$$T = -46.85 + 175.72 \times \frac{S_T}{2^{16}} \quad [^{\circ}\text{C}], \quad (S_T : \text{Temperature signal output})$$

라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어
 - Edge-Embedded의 SHT20 회로도
 - SHT20 센서 주소 : 0x40



라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어 예제(1)
 - 터미널 창에 "nano 13_TEMP HUMI_01.c" 입력

```
pi@raspberrypi:~/Example $ nano 13_TEMP HUMI_01.c
```

- 현재의 온습도를 측정하는 예제

```
1. // File : 13_TEMP HUMI_01.c
2. #include <stdio.h>
3. #include <wiringPi.h>
4. #include <wiringPiI2C.h>           // I2C 라이브러리 참조
5. #define SHT20_I2C_ADDR             0x40  // SHT20 I2C 주소
6. // SHT20 명령 코드
7. #define SHT20_I2C_CMD_MEASURE_TEMP 0xF3
8. #define SHT20_I2C_CMD_MEASURE_HUMI 0xF5
9. #define SHT20_SOFT_RESET           0xFE
10. int fd;                           // SHT20의 handle
11. int main(void)
12. {
```

라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어 예제(1)

```
13. float temp = 0.0;
14. float humi = 0.0;

15. int aData[2];
16. int value;
17. int i;

18. // I2C 시스템 초기화 설정
19. if((fd = wiringPiI2CSetup(SHT20_I2C_ADDR) < 0)
20. {
21.     return -1;
22. }

23. // SHT20 센서 초기화 설정
24. wiringPiI2CWrite(fd, SHT20_SOFT_RESET);
25. delay(50);

26. while(1)
27. {
28.     // 온도 측정 모드
29.     wiringPiI2CWrite(fd, SHT20_I2C_CMD_MEASURE_TEMP);
30.     delay(260);
```

라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어 예제(1)

```
31. // SHT20으로부터 측정된 데이터를 저장
32. for(i=0; i<2; i++)
33. {
34.     data[i] = wiringPiI2CRead(fd);
35. }
36. // 측정된 데이터를 온도로 계산
37. value = aData[0] << 8 | aData[1];
38. temp = -46.85 + 175.72/65536*(int) value;

39. // 습도 측정 모드
40. wiringPiI2CWrite(fd, SHT20_I2C_CMD_MEASURE_HUMI);
41. delay(260);

42. // SHT20으로부터 측정된 데이터를 저장
43. for(i=0; i<2; i++)
44. {
45.     data[i] = wiringPiI2CRead(fd);
46. }

47. // 측정된 데이터를 습도로 계산
48. val = data[0] << 8 | data[1];
49. humi = -6.0 + 125.0/65536*(int)val;
```

라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어 예제(1)

```
50.     printf("Temp : %.1fWn", (float)temp);
51.     printf("Humi : %.1fWn", (float)humi);

52.     delay(1000);
53. }
54. return 0;
55. }
```

- 작성 후 "ctrl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
- GCC 컴파일러를 사용하여 빌드 및 생성된 "13_TEMPHUMI_01" 파일 실행

```
pi@raspberrypi:~/Example $ gcc -o 13_TEMPHUMI_01 13_TEMPHUMI_01.c -
lwiringPi
pi@raspberrypi:~/Example $ ./13_TEMPHUMI_01
```

라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어 예제(1)
 - 결과
 - 현재의 온습도를 측정하여 터미널 화면에 데이터 출력

```
Temp : 30.6  
Humi : 35.7  
Temp : 30.5  
Humi : 35.7  
Temp : 30.5  
Humi : 35.7  
Temp : 31.9  
Humi : 42.6
```


라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어 예제(2)
 - 터미널 창에 “nano 13_TEMP HUMI_02.c” 입력

```
pi@raspberrypi:~/Example $ nano 13_TEMP HUMI_02.c
```

- 현재 온습도 측정값을 TextLCD에 출력하고, 측정값이 30도 이상이거나 습도가 50% 이상일 경우 DC Motor를 회전하는 예제

```
1. // File : 13_TEMP HUMI_01.c
2. #include <stdio.h>
3. #include <wiringPi.h>
4. #include <wiringPiI2C.h>           // I2C 라이브러리 참조
5. #include <lcd.h>                   // lcd 라이브러리 참조
6. #define SHT20_I2C_ADDR             0x40 // SHT20 I2C 주소
7. // SHT20 명령 코드
8. #define SHT20_I2C_CMD_MEASURE_TEMP 0xF3
9. #define SHT20_I2C_CMD_MEASURE_HUMI 0xF5
10. #define SHT20_SOFT_RESET           0xFE
```

라즈베리파이에서의 I2C 사용

● Temp/Humi(온습도) 센서 제어 예제(2)

```
11. const int pinEnable = 12;
12. const int pinPositive = 4;
13. const int pinNegative = 25;

14. int fd;                                // SHT20의 handle

15. int main(void)
16. {
17.     float temp = 0.0;
18.     float humi = 0.0;

19.     int aData[2];
20.     int value;
21.     int i;

22.     wiringPiSetupGpio();                // 핀 번호를 BCM Mode로 설정

23.     pinMode(pinEnable, OUTPUT);
24.     pinMode(pinPositive, OUTPUT);
25.     pinMode(pinNegative, OUTPUT);

26.     // Text LCD 설정
27.     int lcd = lcdInit(2, 16, 4, 16, 26, 18, 27, 22, 23, 0, 0, 0, 0);
```

라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어 예제(2)

```
28. // I2C 시스템 초기화 설정
29. if((fd = wiringPiI2CSetup(SHT20_I2C_ADDR) < 0)
30. {
31.     return -1;
32. }

33. // SHT20 센서 초기화 설정
34. wiringPiI2CWrite(fd, SHT20_SOFT_RESET);

35. lcdClear(lcd);
36. delay(50);

37. while(1)
38. {
39.     // 온도 측정 모드
40.     wiringPiI2CWrite(fd, SHT20_I2C_CMD_MEASURE_TEMP);
41.     delay(260);

42.     // SHT20으로부터 측정된 데이터를 저장
43.     for(i=0; i<2; i++)
44.     {
```

라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어 예제(2)

```
45.         data[i] = wiringPiI2CRead(fd);
46.     }

47.     // 측정된 데이터를 온도로 계산
48.     value = aData[0] << 8 | aData[1];

49.     temp = -46.85 + 175.72/65536*(int) value;

50.     // 습도 측정 모드
51.     wiringPiI2CWrite(fd, SHT20_I2C_CMD_MEASURE_HUMI);
52.     delay(260);

53.     // SHT20으로부터 측정된 데이터를 저장
54.     for(i=0; i<2; i++)
55.     {
56.         data[i] = wiringPiI2CRead(fd);
57.     }

58.     // 측정된 데이터를 습도로 계산
59.     val = data[0] << 8 | data[1];
60.     humi = -6.0 + 125.0/65536*(int)val;
```

라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어 예제(2)

```
61. // lcd화면에 온습도 출력
62. lcdPosition(lcd, 0, 0);
63. lcdPrintf(lcd, "Temp : %.1f [C]", temp);

64. lcdPosition(lcd, 0, 1);
65. lcdPrintf(lcd, "Humi : %.1f [%]", humi);

66. delay(500);

67. // 온도 30.0도 또는 습도 50.0% 이상일 경우 DC Motor 회전
68. if(temp >= 30.0 || humi >= 50.0)
69. {
70.     digitalWrite(pinEnable, HIGH);
71.     digitalWrite(pinPositive, HIGH);
72.     digitalWrite(pinNegative, LOW);
73. }
74. else
75. {
76.     digitalWrite(pinEnable, LOW);
77. }
78. }
79. return 0;
80. }
```

라즈베리파이에서의 I2C 사용

- Temp/Humi(온습도) 센서 제어 예제(2)
 - 작성 후 "ctrl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
 - GCC 컴파일러를 사용하여 빌드 및 생성된 "13_TEMP HUMI_02" 파일 실행

```
pi@raspberrypi:~/Example $ gcc -o 13_TEMP HUMI_02 13_TEMP HUMI_02.c  
-lwiringPi -lwiringPiDev  
pi@raspberrypi:~/Example $ ./13_TEMP HUMI_02
```

- 결과
 - 현재 온습도가 Text LCD에 표시되고, 조건에 따라 DC 모터 동작

