

라즈베리파이 제어

- GPIO를 이용한 제어
- Interrupt를 이용한 제어
- I2C 통신을 이용한 제어
- SPI 통신을 이용한 제어

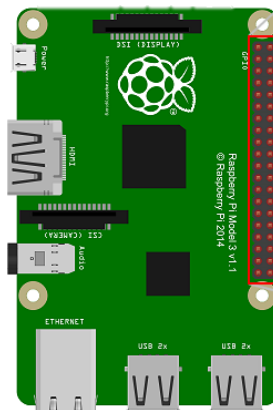


엣지아이랩

GPIO를 이용한 제어

GPIO를 이용한 제어

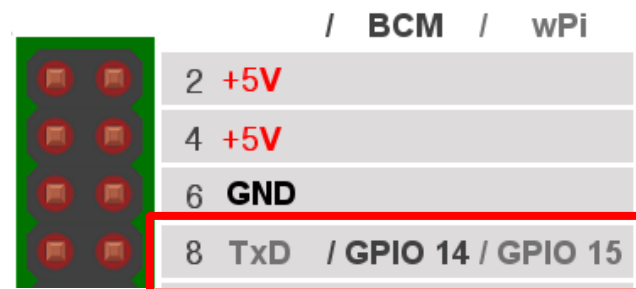
- 40개의 확장핀
 - 전원
 - 5V
 - GND
 - 외부 장치와 데이터 통신
 - UART
 - I2C
 - SPI
 - 외부장치 제어
 - PWM
 - GPIO



wPi	/	BCM	/				/	BCM	/	wPi
				+3.3V	1					2 +5V
GPIO 08	/	GPIO 02	/	SDA1	3					4 +5V
GPIO 09	/	GPIO 03	/	SCL1	5					6 GND
GPIO 07	/	GPIO 04	/		7					8 TxD / GPIO 14 / GPIO 15
				GND	9					10 RxD / GPIO 15 / GPIO 16
GPIO 00	/	GPIO 17	/		11					12 / GPIO 18 / GPIO 01
GPIO 02	/	GPIO 27	/		13					14 GND
GPIO 03	/	GPIO 22	/		15					16 / GPIO 23 / GPIO 04
				+3.3V	17					18 / GPIO 24 / GPIO 05
GPIO 12	/	GPIO 10	/	MOSI	19					20 GND
GPIO 13	/	GPIO 09	/	MISO	21					22 / GPIO 25 / GPIO 06
GPIO 14	/	GPIO 11	/	SCLK	23					24 CE0 / GPIO 08 / GPIO 10
				GND	25					26 CE1 / GPIO 07 / GPIO 11
GPIO 30	/	GPIO 00	/	SDA0	27					28 SCL0 / GPIO 01 / GPIO 31
GPIO 21	/	GPIO 05	/		29					30 GND
GPIO 22	/	GPIO 06	/		31					32 / GPIO 12 / GPIO 26
GPIO 23	/	GPIO 13	/		33					34 GND
GPIO 24	/	GPIO 19	/		35					36 / GPIO 16 / GPIO 27
GPIO 25	/	GPIO 26	/		37					38 / GPIO 20 / GPIO 28
				GND	39					40 / GPIO 21 / GPIO 29

GPIO를 이용한 제어

- 라즈베리파이의 GPIO 제어
 - BCM
 - BCM283x 칩의 물리적인 핀 번호 명칭
 - 기본 제공 '파이썬(Python)' 라이브러리가 사용
 - wPi
 - BCM283x 칩의 GPIO 핀 번호
 - C언어 라이브러리 'wiringPi' 에서 사용
 - GPIO 헤더 핀
 - 라즈베리파이에서 외부 연결이 용이하도록 확장시킨 헤더핀 번호
 - ex) GPIO 헤더 8번핀의 경우
 - 'TXD / GPIO 14 / GPIO 15' 표시
 - TxD : GPIO 입출력 기능 외에 부가기능으로 시리얼 통신 포트의 '송신'기능 사용 가능
 - GPIO 14 : BCM 기준 14번 핀
 - GPIO 15 : wPi 기준 15번 핀



GPIO를 이용한 제어

- WiringPi 라이브러리 설치
 - 터미널 창 "sudo apt-get install git-core" 입력
 - 오픈소스 기반의 분산형 소스관린 시스템 "git" 설치

```
pi@raspberrypi:~ $ sudo apt-get install git-core
```

- 터미널 창에 "git clone git ://git.drogon.net/wiringPi"
 - wiringPi 소스 복사

```
pi@raspberrypi:~ $ git clone git://git.drogon.net/wiringPi
```

```
pi@raspberrypi:~ $ git clone git://git.drogon.net/wiringPi
Cloning into 'wiringPi'...
remote: Counting objects: 1151, done.
remote: Compressing objects: 100% (957/957), done.
remote: Total 1151 (delta 804), reused 213 (delta 142)
Receiving objects: 100% (1151/1151), 366.50 KiB | 141.00 KiB/s, done.
Resolving deltas: 100% (804/804), done.
Checking connectivity... done.
pi@raspberrypi:~ $
```

GPIO를 이용한 제어

- WiringPi 라이브러리
 - 터미널 창에 "cd wiringPi" 입력
 - 복제된 폴더로 이동
 - wiringPi의 폴더에서 "./build" 입력
 - 라이브러리 빌드 및 설치

```
pi@raspberrypi:~ $ cd wiringPi
pi@raspberrypi:~/wiringPi $ ./build
```

```
pi@raspberrypi:~ $ cd wiringPi/
pi@raspberrypi:~/wiringPi $ ./build
wiringPi Build script
=====

WiringPi Library
[UnInstall]
[Compile] wiringPi.c
[Compile] wiringSerial.c
[Compile] wiringShift.c
[Compile] piHiPri.c
[Compile] piThread.c
```

GPIO를 이용한 제어

- WiringPi 라이브러리
 - 터미널 창에 "gpio -v" 입력
 - Gpio 버전 확인

```
pi@raspberrypi:~ $ gpio -v
```

```
pi@raspberrypi:~ $ gpio -v
gpio version: 2.44
Copyright (c) 2012-2017 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty

Raspberry Pi Details:
Type: Pi 3, Revision: 02, Memory: 1024MB, Maker: Sony
* Device tree is enabled.
*--> Raspberry Pi 3 Model B Rev 1.2
* This Raspberry Pi supports user-level GPIO access.
pi@raspberrypi:~ $
```

GPIO를 이용한 제어

- WiringPi 라이브러리
 - 터미널 창에 “gpio readall” 입력
 - 라즈베리파이의 확장 핀 정보 확인

```
pi@raspberrypi:~ $ gpio readall
```

```
pi@raspberrypi:~ $ gpio readall
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi |   Name   | Mode | V | Physical | V | Mode |   Name   | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 2   | 8   | 3.3v     |      |   | 1   | 2   |      | 5v       |     |     |
| 3   | 9   | SDA.1    | ALTO | 1 | 3   | 4   |      | 5v       |     |     |
| 4   | 7   | SCL.1    | ALTO | 1 | 5   | 6   |      | 0v       |     |     |
|     |     | GPIO. 7  | IN   | 1 | 7   | 8   | 1   | TXD      | 15  | 14  |
|     |     | 0v       |      |   | 9   | 10  | 0   | RXD      | 16  | 15  |
| 17  | 0   | GPIO. 0  | OUT  | 0 | 11  | 12  | 0   | GPIO. 1  | 1   | 18  |
| 27  | 2   | GPIO. 2  | IN   | 0 | 13  | 14  |     | 0v       |     |     |
| 22  | 3   | GPIO. 3  | IN   | 0 | 15  | 16  | 0   | GPIO. 4  | 4   | 23  |
|     |     | 3.3v     |      |   | 17  | 18  | 1   | GPIO. 5  | 5   | 24  |
| 10  | 12  | MOSI     | ALTO | 0 | 19  | 20  |     | 0v       |     |     |
| 9   | 13  | MISO     | ALTO | 1 | 21  | 22  | 0   | GPIO. 6  | 6   | 25  |
| 11  | 14  | SCLK     | ALTO | 0 | 23  | 24  | 1   | CE0      | 10  | 8   |
|     |     | 0v       |      |   | 25  | 26  | 1   | CE1      | 11  | 7   |
| 0   | 30  | SDA.0    | IN   | 1 | 27  | 28  | 0   | SCL.0    | 31  | 1   |
| 5   | 21  | GPIO.21  | IN   | 0 | 29  | 30  |     | 0v       |     |     |
| 6   | 22  | GPIO.22  | IN   | 0 | 31  | 32  | 0   | GPIO.26  | 26  | 12  |
| 13  | 23  | GPIO.23  | IN   | 0 | 33  | 34  |     | 0v       |     |     |
| 19  | 24  | GPIO.24  | IN   | 1 | 35  | 36  | 0   | GPIO.27  | 27  | 16  |
| 26  | 25  | GPIO.25  | IN   | 0 | 37  | 38  | 0   | GPIO.28  | 28  | 20  |
|     |     | 0v       |      |   | 39  | 40  | 0   | GPIO.29  | 29  | 21  |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi |   Name   | Mode | V | Physical | V | Mode |   Name   | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
pi@raspberrypi:~ $
```


GPIO를 이용한 제어

- WiringPi GPIO 제어
 - GPIO란?
 - “General Purpose Input Output”의 약자
 - 범용 입출력 기능을 담당하는 핀
 - 기본적으로 GPIO로 사용되나 SW 설정에 따라 부가적으로 사용 가능
 - 출력(OUTPUT)
 - SW에서 핀 방향을 출력으로 설정
 - SW동작에 의해 GPIO 핀의 상태를 HIGH(3.3V)혹은 LOW(0V) 상태로 변경
 - GPIO핀에 연결된 하드웨어 제어 가능
 - 입력(INPUT)
 - SW에서 핀 방향을 입력으로 설정
 - SW동작에 의해 GPIO 핀의 상태가 HIGH(3.3V)혹은 LOW(0V) 상태인지 확인
 - SW에서 하드웨어의 상태 파악 가능

GPIO를 이용한 제어

- WiringPi를 이용한 제어
 - C언어 기반으로 GPIO 제어
 - 컴파일러는 GNU 라이선스 기반의 GCC 컴파일러 사용
 - GPIO 기본 함수들

GPIO 설정 함수	
int wiringPiSetup(void);	핀 번호 인덱싱을 wPi모드 기준으로 설정
int wiringPiSetupGpio(void);	핀 번호 인덱싱을 BCM 모드 기준으로 설정
void pinMode(int pin, int mode);	해당 핀의 입출력 모드 설정
<ul style="list-style-type: none">- pin : 핀 번호- mode : INPUT/OUTPUT	
GPIO 입력 함수	
int digitalRead(int pin);	해당 핀의 상태를 확인
<ul style="list-style-type: none">- pin : 핀 번호- 반환 값 : HIGH/LOW 혹은 1/0	

GPIO를 이용한 제어

- WiringPi를 이용한 제어
 - GPIO 기본 함수들

GPIO 출력 함수	
<code>void digitalWrite(int pin, int value);</code>	해당 핀의 상태를 설정
- pin : 핀 번호 - value : HIGH/LOW 혹은 1/0	
Timing 관련 함수	
<code>unsigned int millis(void);</code>	GPIO 설정 함수 중 하나가 호출된 이후로 ms 단위의 시간을 반환
<code>unsigned int micros(void);</code>	GPIO 설정 함수 중 하나가 호출된 이후로 us 단위의 시간을 반환
<code>void delay(unsigned int howLong);</code>	howLong(ms단위) 시간을 지연(현 상태 유지)
<code>void delayMicroseconds(unsigned int howLong);</code>	howLong(us 단위) 시간을 지연(현 상태 유지)

GPIO를 이용한 제어

- GPIO 출력 제어
 - Ex) LED가 0.5 초마다 On/Off 를 반복하는 예제

```
1. // gpio_output_test.c 파일
2. #include <stdio.h>
3. #include <wiringPi.h>           // wiringPi 헤더파일 참조
4. #define LED 21                  // 해당 핀에 LED 연결

5. int main(void)
6. {
7.     printf("Raspberry Pi - LED Blink\n");
8.     wiringPiSetupGpio();         // 핀 번호를 BCM모드로 설정
9.     pinMode(LED, OUTPUT);       // LED 핀을 출력모드로 설정

10.    while(1)                    // 무한 루프
11.    {
12.        digitalWrite(LED, HIGH); // LED 핀의 상태를 HIGH로 변경
13.        delay(500);              // 500msec 지연
14.        digitalWrite(LED, LOW);  // LED 핀의 상태를 LOW로 변경
15.        delay(500);
16.    }
17.    return 0;
18. }
```

GPIO를 이용한 제어

- GPIO 출력 제어

- 작성 완료 후, 터미널 창에 “gcc -o gpio_output_test gpio_output_test.c -lwiringPi” 입력
 - GCC 컴파일러를 사용하여 빌드
 - “-l” 옵션은 wiringPi 라이브러리를 링크 작업에 참여시켜 함께 빌드함을 의미

```
pi@raspberrypi:~ $ gcc -o gpio_output_test gpio_output_test.c -lwiringPi
```

- 빌드 과정이 끝나면 해당 디렉터리에 “gpio_output_test” 실행 파일 생성
- 터미널 창에 “./gpio_output_test” 입력
 - 현재 디렉터리의 gpio_output_test 파일 실행

```
pi@raspberrypi:~ $ ./gpio_output_test
```

GPIO를 이용한 제어

- GPIO 입력 제어
 - Ex) SWITCH를 눌렀을 경우 "Button is pressed" 라고 출력하는 예제

```
1. // gpio_input_test.c 파일
2. #include <stdio.h>
3. #include <wiringPi.h>
4. #define KEY 5 // 해당 핀에 SWITCH 연결

5. int main(void)
6. {
7.     printf("Raspberry Pi – Key Input Test\n");
8.     wiringPiSetupGpio(); // 핀 번호를 BCM모드로 설정
9.     pinMode(KEY, INPUT); // SWITCH 핀을 입력모드로 설정

10.    while(1) // 무한 루프
11.    {
12.        int val = digitalRead(KEY); // SWITCH 핀의 상태를 확인
13.        if(val == LOW) // 상태가 LOW 이면
14.        {
15.            printf("Button is pressed\n"); // "Button is Pressed" 출력
16.        }
17.        delay(1000); // 1000msec 지연
```

GPIO를 이용한 제어

- GPIO 입력 제어
 - Ex) SWITCH를 눌렀을 경우 "Button is pressed" 라고 출력하는 예제

```
18.     }  
19.     return 0;  
20. }
```

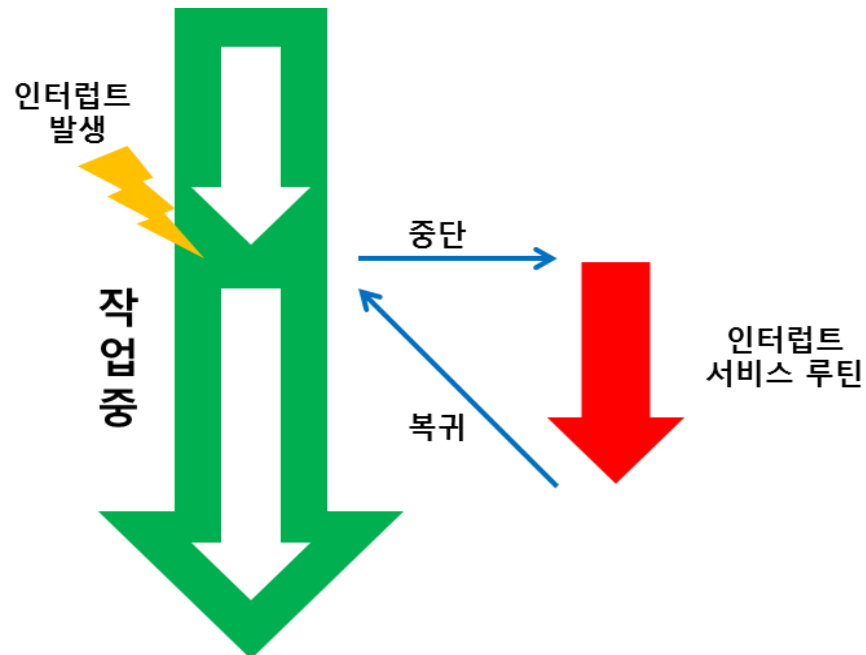
- 작성 완료 후, 터미널 창에 "gcc -o gpio_input_test gpio_input_test.c -lwiringPi" 입력
 - 실행 파일 생성
- 파일 생성 완료 후, 터미널 창에 "./gpio_input_test" 입력
 - 파일 실행

```
pi@raspberrypi:~ $ gcc -o gpio_input_test gpio_input_test.c -lwiringPi  
pi@raspberrypi:~ $ ./gpio_input_test
```

INTERRUPT를 이용한 제어

Interrupt를 이용한 제어

- 인터럽트란?
 - 인터럽트란 "방해하다"라는 의미
 - 전자 측에서는 프로세서의 즉각적인 처리를 필요로 하는 이벤트를 알리기 위해 발생하는 주변 하드웨어나 소프트웨어로부터 요청을 말함
 - 인터럽트 발생시, 운영체제 내의 제어프로그램에 있는 인터럽트 서비스 루틴 (Interrupt Service Routine)이 작동
 - 인터럽트 서비스 루틴 완료 후, 이전 상태로 복귀



Interrupt를 이용한 제어

- 프로그래밍 방식
 - 폴링(Polling) 방식
 - 프로그램 안에서 사용자가 명령어를 이용하여 해당 핀의 값을 정기적으로 읽어 상태 변화를 알아내는 방법
 - 인터럽트 방식
 - 외부 또는 내부에서 발생하는 사건에 대해 하드웨어의 도움을 받아 자동으로 감지하고, 인터럽트 서비스 루틴을 실행하는 방법
 - WiringPi를 이용한 제어
 - 첫 번째 인자 pin은 INPUT 모드
 - INT_EDGE_FALLING은 해당 핀의 상태가 HIGH 에서 LOW로 떨어지는 순간 인터럽트 서비스 루틴 호출
 - INT_EDGE_RISING은 해당 핀의 상태가 LOW에서 HIGH로 올라가는 순간 인터럽트 서비스 루틴 호출
 - INT_EDGE_BOTH는 두 경우 모두 해당

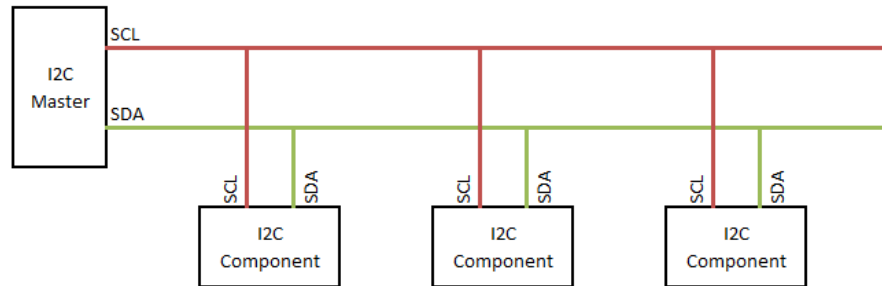
설정 함수	
<pre>int wiringPiISR(int pin, int mode, void (*function)(void));</pre> <p>- 반환 값 : 실패시 '0'보다 작은 수</p>	<p>지정된 핀에서 수신된 인터럽트에 함수를 등록</p> <p>Mode는 "INT_EDGE_FALLING, INT_EDGE_RISING, INT_EDGE_BOTH"가 있음</p>

I2C 통신을 이용한 제어

I2C 통신을 이용한 제어

● I2C란?

- I2C(Inter-Integrated Circuit 또는 TWI-Two Wire Interface)는 직렬 컴퓨터 버스를 의미
- 마더보드, 임베디드, 휴대전화 등에 저속의 주변기기를 연결하기 위해 사용

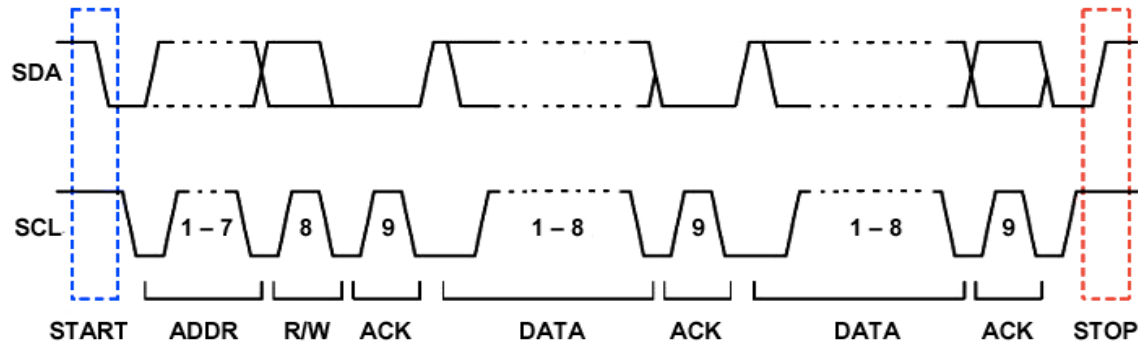


- 하나의 마스터 장치와 여러 슬레이브 장치들 간의 데이터 송수신을 위한 통신 방식
- I2C는 클럭과 데이터 두 개의 신호로 구성
 - 클럭 : 마스터 장치에서 슬레이브 장치로 출력
 - 데이터 : 하나의 신호선으로 양방향 통신이 이루어짐

I2C 통신을 이용한 제어

● I2C 통신

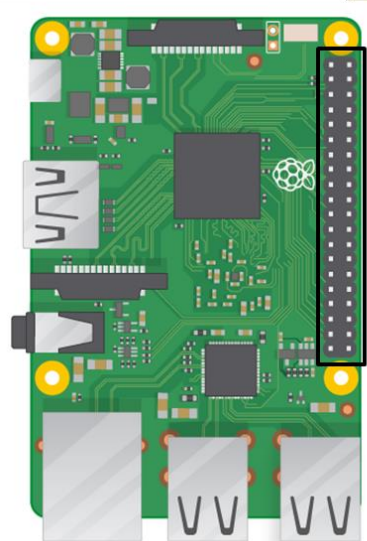
- SCL(클럭) 신호 : 마스터에서 출력
- SDA(데이터) : 양방향 신호, 마스터 또는 슬레이브에서 출력



- I2C 기반의 전송은 주소 전송 구간과 한 바이트(8bit) 데이터 전송구간으로 나뉨
- 마스터 장치는 주소 전송 구간과 데이터 전송 구간의 마지막 비트(9번째 비트)에 항상 슬레이브로부터의 응답(ACK-Acknowledge)를 기다림
 - 응답이 오면 전송 시작
 - 응답이 없으면 오류로 처리
- 마스터 장치에서 사용되는 클럭은 보통 100kHz의 데이터 통신
 - 고속 데이터 통신에 부적합
- 주소 전송 구간이 7비트로 최대 128개의 슬레이브 연결 가능
 - 각 슬레이브 자아치들은 고유의 주소 필요

I2C 통신을 이용한 제어

- 라즈베리파이에서의 I2C 사용
 - 라즈베리파이 확장 핀 3, 5번 핀에는 I2C 채널이 존재



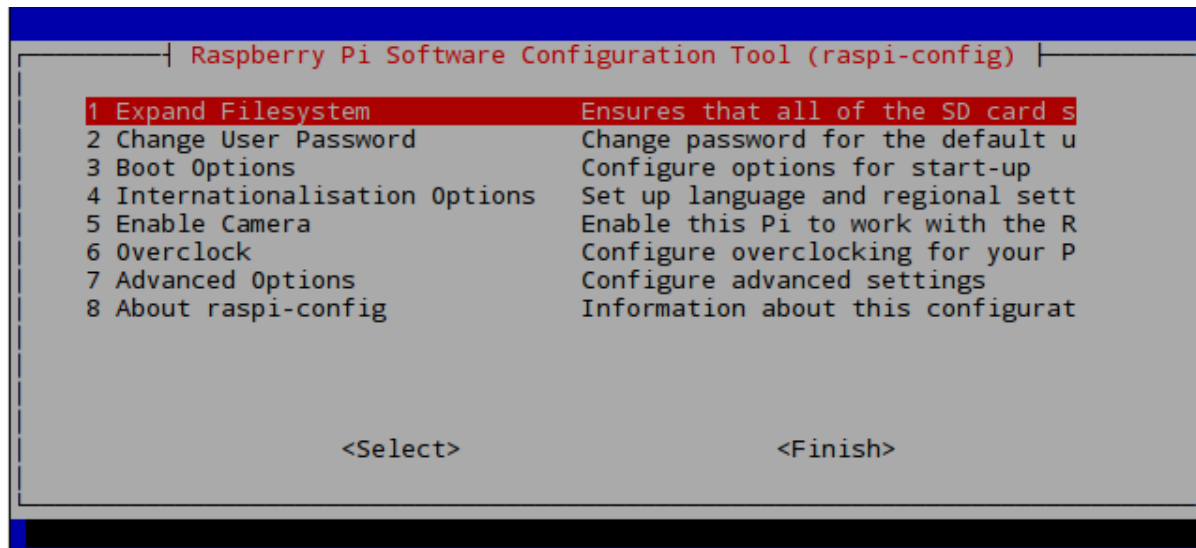
Name	Physical	Name
+3.3V	1	+5V
SDA1	3	+5V
SCL1	5	GND
GPIO	7	TxD
GND	9	RxD
GPIO	11	GPIO
GPIO	13	GND
GPIO	15	GPIO
+3.3V	17	GPIO
MOSI	19	GND
MISO	21	GPIO
SCLK	23	CE0
GND	25	CE1
SDA0	27	SCL0
GPIO	29	GND
GPIO	31	GPIO
GPIO	33	GND
GPIO	35	GPIO
GPIO	37	GPIO
GND	39	GPIO

I2C 통신을 이용한 제어

- I2C 통신 활성화
 - 터미널 창에 “sudo raspi-config” 입력
 - I2C 활성화를 위한 설정

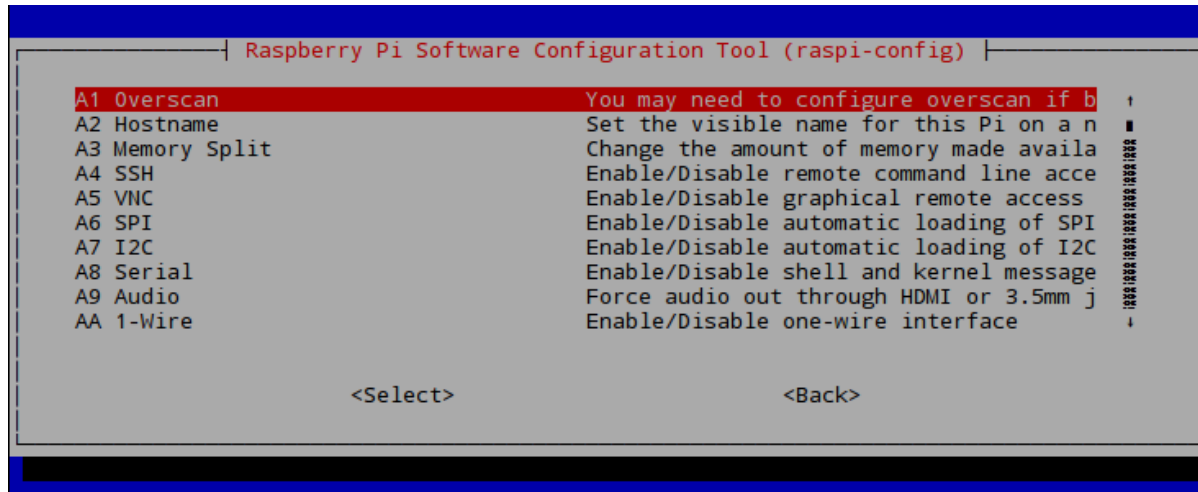
```
pi@raspberrypi:~ $ sudo raspi-config
```

- 7. Advanced Options 선택

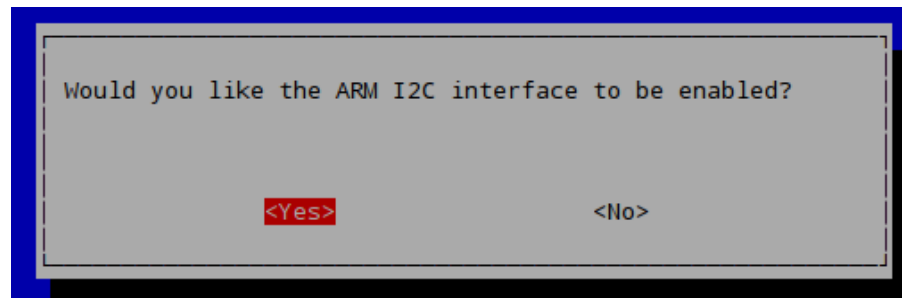


I2C 통신을 이용한 제어

- I2C 통신 활성화
 - A7. I2C 선택



- "Yes" 선택
 - I2C 활성화



I2C 통신을 이용한 제어

- I2C 통신 활성화

- 터미널 창에 “sudo reboot” 입력
 - 시스템 재시작

```
pi@raspberrypi:~ $ sudo reboot
```

- 시스템 재시작 완료 후, 터미널 창에 “sudo apt-get install i2c-tools” 입력
 - I2C 관련 응용 도구 다운로드

```
pi@raspberrypi:~ $ sudo apt-get install i2c-tools
```

- 설치 완료 후, 터미널 창에 “lsmod” 입력
 - I2C 관련 모듈 설정 확인

```
pi@raspberrypi:~ $ lsmod
```

I2C 통신을 이용한 제어

- I2C 통신 활성화
 - Lsmod 실행
 - i2c_bcm2708이 보이면 설정이 되었다는 의미

```
pi@raspberrypi:~ $ lsmod
Module                  Size  Used by
bnep                    10340  2
hci_uart                17943  1
btbcm                   5929   1 hci_uart
bluetooth              326105  22 bnep,btbcm,hci_uart
uinput                  7454   1
cfg80211               427855  0
rfkill                 16037   3 cfg80211,bluetooth
evdev                  11396   4
joydev                 9024   0
snd_bcm2835            20447   3
snd_pcm                75762   1 snd_bcm2835
snd_timer              19288   1 snd_pcm
spi_bcm2835            6678   0
snd                    51908   9 snd_bcm2835,snd_timer,snd_pcm
i2c_bcm2708            4834   0
bcm2835_gpiomem         3040   0
bcm2835_wdt            3225   0
```

I2C 통신을 이용한 제어

- I2C 통신 활성화
 - 터미널 창에 “i2cdetect -y 1” 입력
 - 라즈베리파이의 I2C 디바이스(i2C-1)에 연결된 채널 정보 확인

```
pi@raspberrypi:~ $ i2cdetect -y 1
```

```
pi@raspberrypi:~ $ i2cdetect -y 1
   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
20: 20 21 -- 23 --  --  --  --  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40: 40  --  --  --  --  --  --  --  --  --  --  --  --  --  --
50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
60:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
70:  --  --  -- 73  --  --  --  --  --  --  --  --  --  --  --
pi@raspberrypi:~ $
```

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

I2C 통신을 이용한 제어

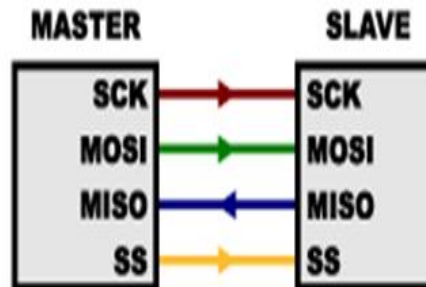
- WiringPi I2C 라이브러리
 - 라이브러리 제공 함수

초기화 함수	
<pre>int wiringPiI2CSetup(int devid);</pre> <p>- 반환 값 : 성공시 'handle' 값, 실패시 '-1'</p>	장치의 식별자(devid)로 I2C 시스템을 초기화. devid는 장치의 I2C 주소
제어 함수	
<pre>int wiringPiI2CRead(int fd);</pre>	해당 디바이스(핸들)을 읽음, 또는 data를 반환
<pre>int wiringPiI2CWrite(int fd, int data);</pre>	해당 디바이스에 data 값을 설정 일부 장치는 레지스터에 접근하지 않고 사용 가능
<pre>int wiringPiI2CWriteReg8(int fd, int reg, int data);</pre> <pre>int wiringPiI2CWriteReg16(int fd, int reg, int data);</pre>	해당 디바이스 레지스터에 8또는 16bit data 값을 설정
<pre>int wiringPiI2CReadReg8(int fd, int reg);</pre> <pre>int wiringPiI2CReadReg16(int fd, int reg);</pre>	지정된 디바이스 레지스터로부터 8 또는 16비트 값을 읽음

SPI 통신을 이용한 제어

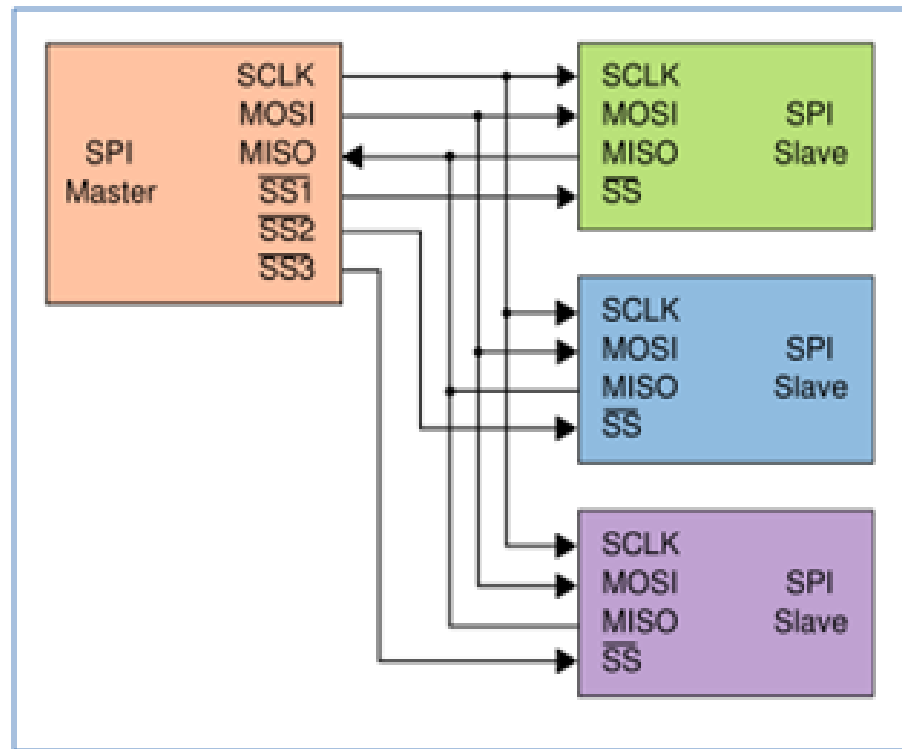
SPI 통신을 이용한 제어

- SPI(Serial Peripheral Interface)란?
 - 데이터 통신 방식 중 하나
 - 하드웨어 적으로 전송 속도를 최대로 얻기 위한 목적으로 개발
 - 빠른 데이터 전송 속도
 - 하나의 마스터 장치와 여러 슬레이브 장치가 연결되어 있는 구조
 - 클럭을 이용하여 동기화된 직렬 통신 방식 사용
 - 네 개의 신호선 으로 구성
 - SCLK(Serial Clock) : 클럭 신호 선
 - MOSI(Master Out, Slave In) : 마스터에서 데이터를 출력
 - MISO(Master In, Slave Out) : 슬레이브에서 데이터 출력
 - SS(Slave Select) : 데이터 송수신할 슬레이브 선택



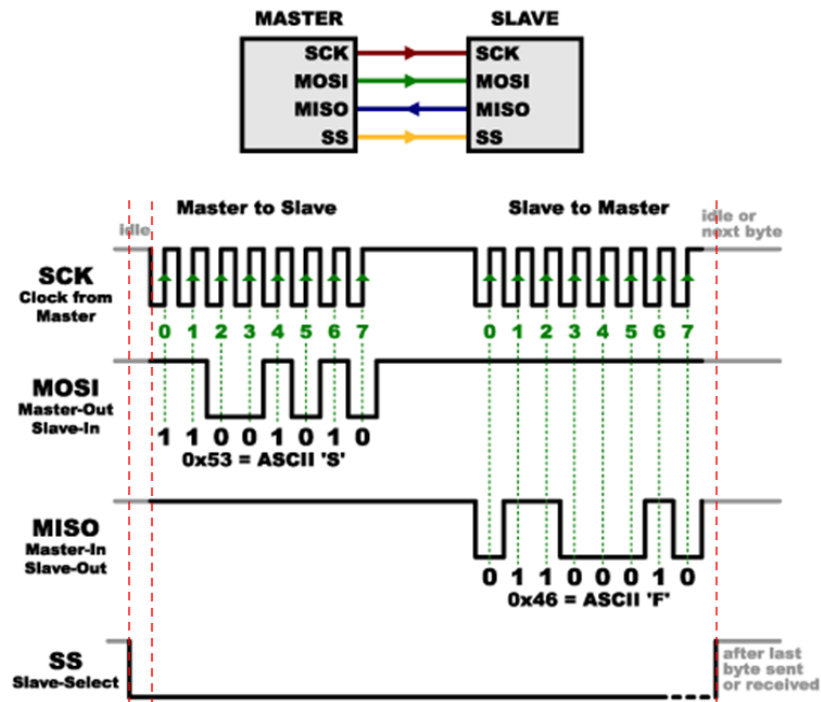
SPI 통신을 이용한 제어

- SPI 통신
 - 데이터 전송과 수신 신호선이 따로 존재
 - 전송과 수신이 동시에 가능
 - 슬레이브 장치만큼 신호선이 증가하므로 장치가 많을 땐 비효율적



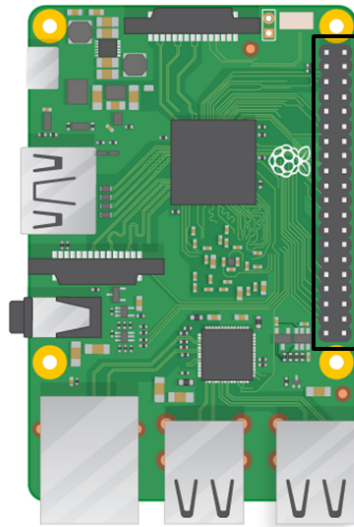
SPI 통신을 이용한 제어

- SPI 통신 순서
 - 마스터에서 SS 신호로 슬레이브 장치 선택
 - 통신 시작 시, SS 신호가 HIGH에서 LOW로 변함
 - 통신 종료 시, SS 신호가 LOW에서 HIGH로 변함
 - 클럭 신호 생성 및 데이터 전송
 - 클럭 신호는 마스터 장치에서만 발생
 - 슬레이브 장치가 가 임의적으로 데이터 전송 불가



SPI 통신을 이용한 제어

- 라즈베리파이에서의 SPI 사용
 - 라즈베리파이 확장 핀 19, 21, 23, 24, 26번 핀에는 SPI 채널이 2개 존재
 - CLK, MOSI, MISO 신호선 공유
 - 두 채널 CE0, CE1(Chip Enable) 따로 사용



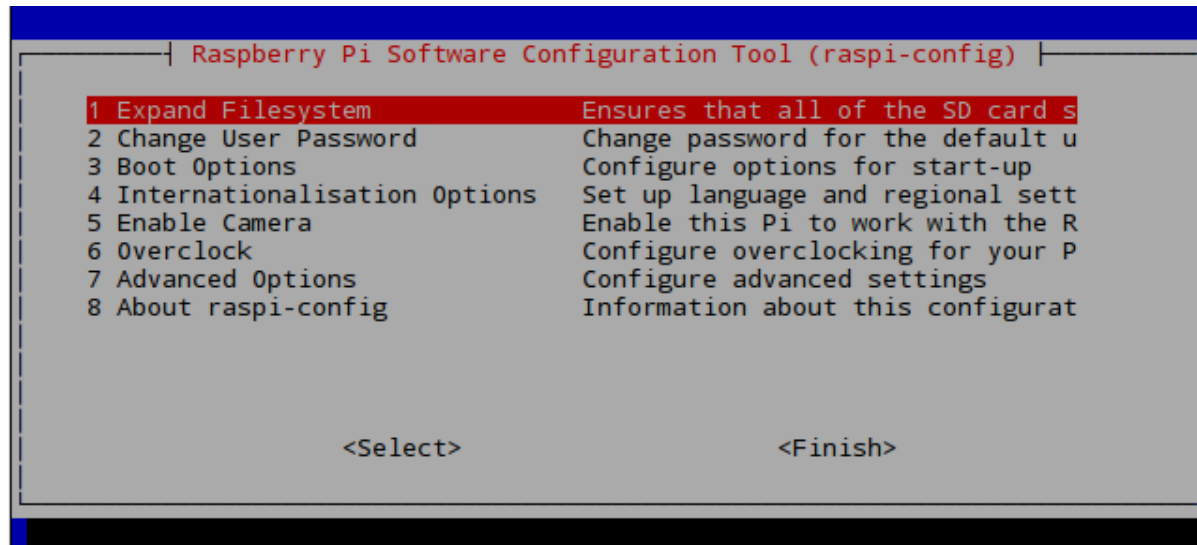
Name	Physical	Name
+3.3V	1 2	+5V
SDA1	3 4	+5V
SCL1	5 6	GND
GPIO	7 8	TxD
GND	9 10	RxD
GPIO	11 12	GPIO
GPIO	13 14	GND
GPIO	15 16	GPIO
+3.3V	17 18	GPIO
MOSI	19 20	GND
MISO	21 22	GPIO
SCLK	23 24	CE0
GND	25 26	CE1
SDA0	27 28	SCL0
GPIO	29 30	GND
GPIO	31 32	GPIO
GPIO	33 34	GND
GPIO	35 36	GPIO
GPIO	37 38	GPIO
GND	39 40	GPIO

SPI 통신을 이용한 제어

- SPI 통신 활성화
 - 터미널 창에 “sudo raspi-config” 입력
 - SPI 통신 활성화를 위한 설정

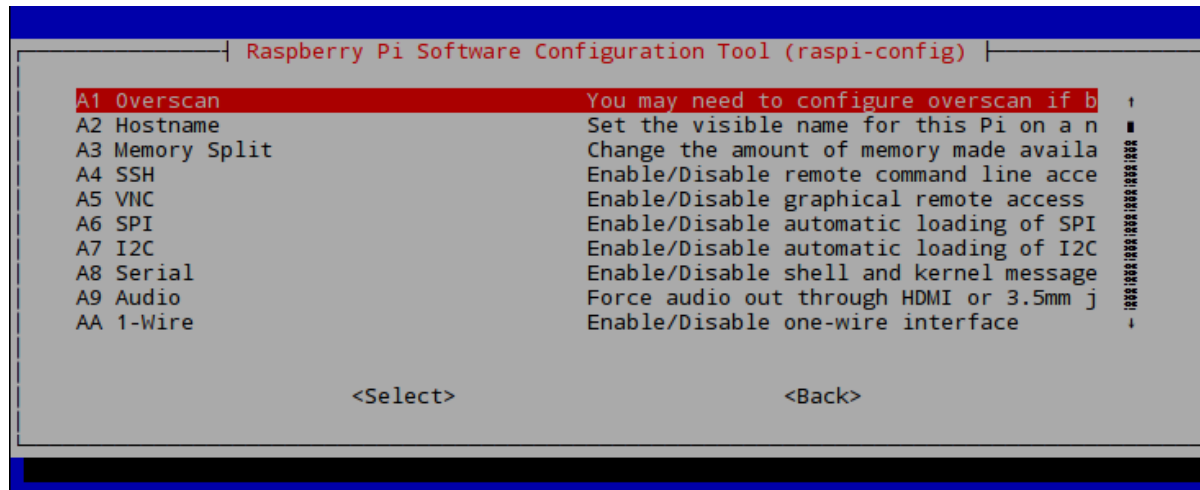
```
pi@raspberrypi:~ $ sudo raspi-config
```

- 7. Advanced Options 선택



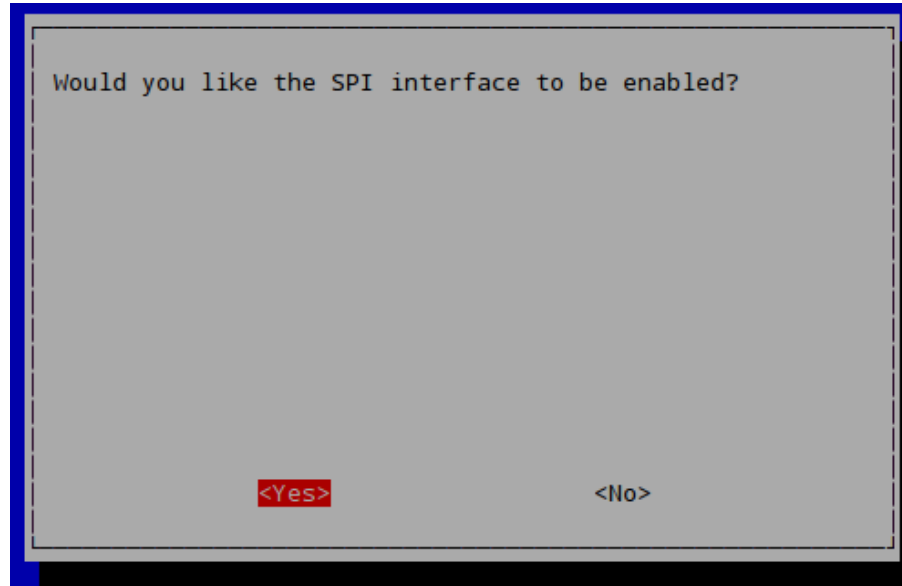
SPI 통신을 이용한 제어

- SPI 통신 활성화
 - A6. SPI 선택



SPI 통신을 이용한 제어

- SPI 통신 활성화
 - “Yes” 선택
 - SPI 활성화



- 터미널 창에 “sudo reboot” 입력
 - 시스템 재시작

```
pi@raspberrypi:~ $ sudo reboot
```

SPI 통신을 이용한 제어

- wiringPi SPI 라이브러리
 - 라이브러리 제공 함수

초기화 함수	
<pre>int wiringPiSPISetup(int channel, int speed);</pre> <p>- 반환 값 : 실패시 '-1'</p>	<p>채널의 초기화를 설정, 라즈베리파이에는 0과 1, 2개의 채널이 존재, Speed는 500,000에서 32,000,000 사이의 정수이며, Hz단위의 SPI 클럭 속도를 의미</p>
제어 함수	
<pre>int wiringPiSPIDataRW(int chan nel, unsigned char *data, int len);</pre>	<p>SPI를 통해 동시에 Read/Write 를 수행, 버퍼 안의 data는 SPI에서 반환된 데이터로 덮어씀</p>