

EDGE-EMBEDDED 응용 실습

- Pi-Camera 제어
- 적외선(IR) 송수신 제어
- Gesture 제어
- UART 기반의 시리얼 통신 제어
- 블루투스 기반의 시리얼 통신 제어



엣지아이랩

PI-CAMERA 제어

Pi-Camera 제어

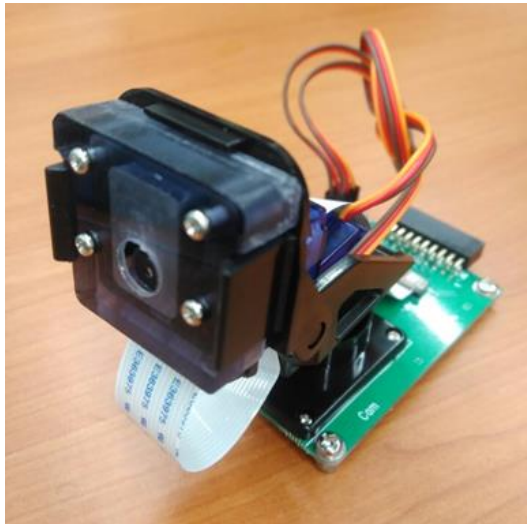
● Pi Camera V2

- 라즈베리파이 카메라 모듈 V2 은 8메가 픽셀의 고품질 소니 IMX219 이미지 센서를 사용한 라즈베리파이의 맞춤형 제품
- 성능
 - 사진의 경우 최대 3280 x 2464 pixel
 - 비디오의 경우, 1080p30, 720p60 등
- 15mm 리본 케이블을 통해 라즈베리파이 본체의 윗면의 CSI 인터페이스에 연결
- 카메라를 응용할 수 있는 분야 및 프로그램들이 많이 있으며, 경우에 따라 영상 처리 알고리즘이 사용될 수 있음
- 카메라 영상을 사용한 영상 처리는 오픈 소스에 많이 사용되는 OpenCV 라이브러리를 사용하는 경우가 많음
- **사용시 주의사항 : 반드시 전원을 분리한 상태에서 카메라 모듈을 결합**



Pi-Camera 제어

- Edge-Embedded의 카메라 모듈
 - Pi 카메라와 Servo 모터 2개를 결합한 제품
 - Servo 모터를 이용해 카메라를 상하좌우 회전할 수 있는 특징
 - Edge-Peripheral(Edge-Embedded의 좌측 센서부분)의 전원 스위치에 따라 2개의 Servo 모터 제어 방식이 달라짐
 - 전원 스위치가 "OFF" 되면, 카메라 모듈의 Servo 모터는 GPIO 17, GPIO 18(BCM Mode)로 제어가 가능

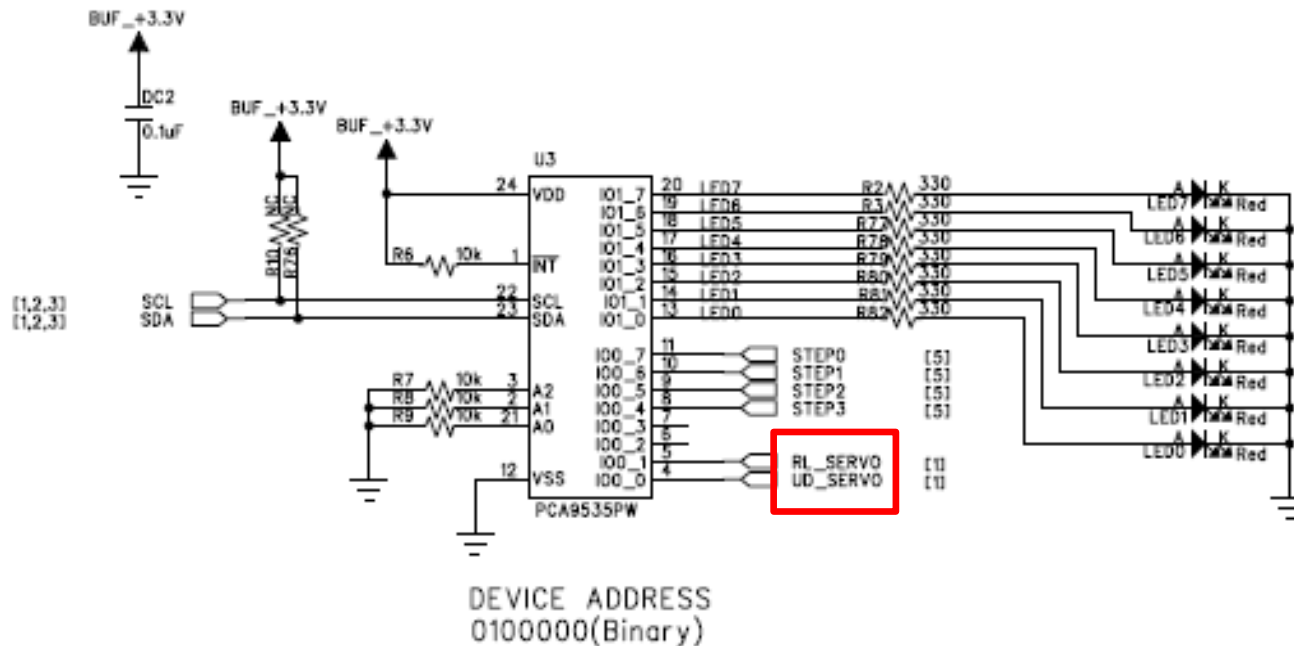


Connect	BCM	Name	Physical	Name	BCM	Connect
		+3.3V	1 2	+5V		
GPIO 02		SDA1	3 4	+5V		
GPIO 03		SCL1	5 6	GND		
GPIO 04		GPIO	7 8	TxD	GPIO 14	
		GND	9 10	RxD	GPIO 15	
Servo_UD	GPIO 17	GPIO	11 12	GPIO	GPIO 18	Servo_LR
GPIO 27		GPIO	13 14	GND		
GPIO 22		GPIO	15 16	GPIO	GPIO 23	
		+3.3V	17 18	GPIO	GPIO 24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

Pi-Camera 제어

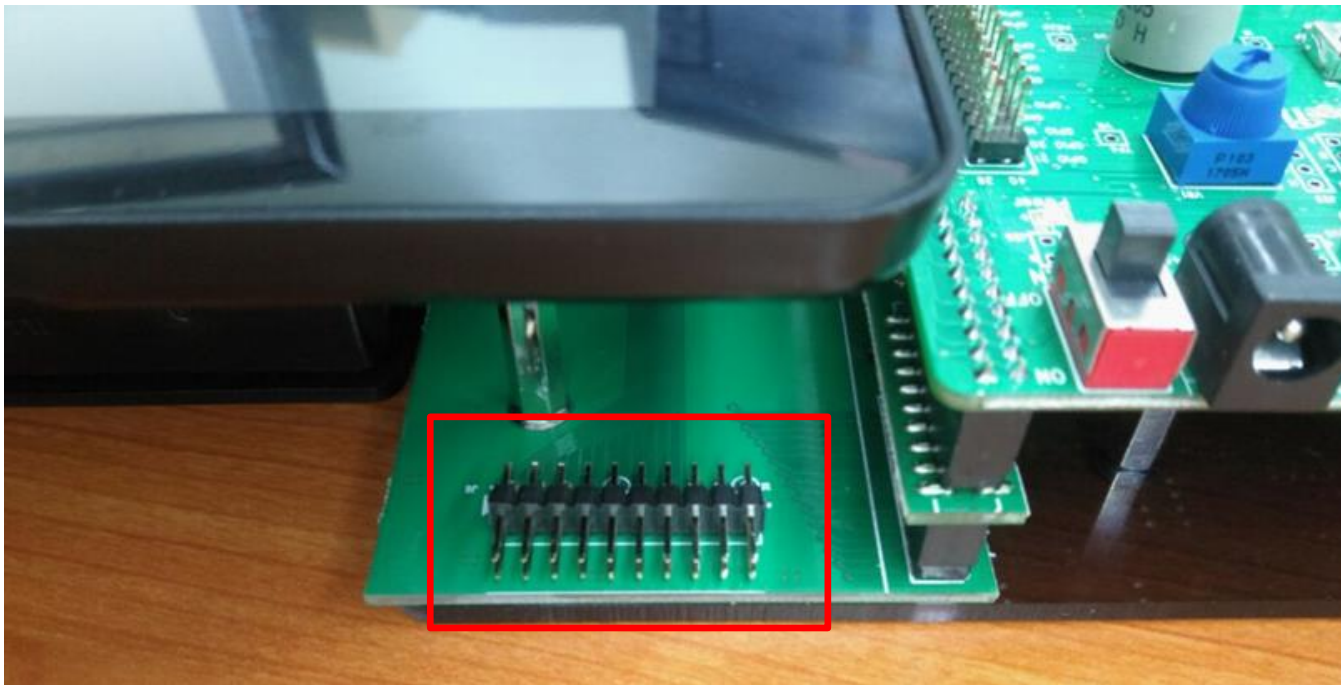
- Edge-Embedded의 카메라 모듈
 - 전원 스위치가 "ON" 되면, 카메라 모듈의 Servo 모터는 I2C 통신으로 제어 가능

8 LED & STEP



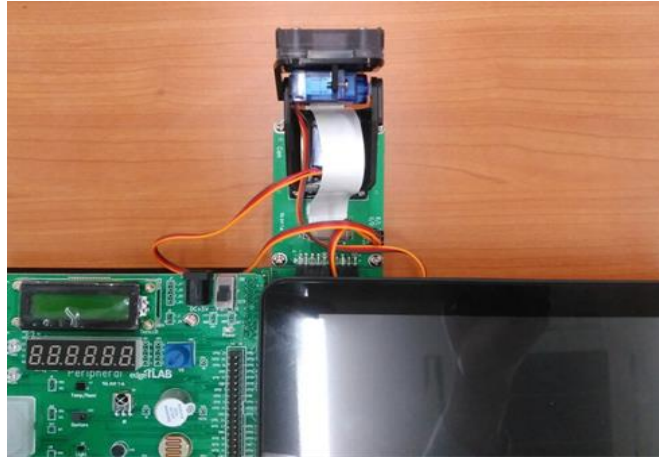
Pi-Camera 제어

- 카메라 활성화
 - Edge-Embedded 에는 카메라 모듈이 존재하며 이를 상단에 맞춰 끼우면 손쉽게 이용 가능
 - 단, 전원이 연결된 상태에서 결합할 경우, 카메라 모듈 및 라즈베리파이에도 치명적인 피해를 입히므로 반드시 전원을 분리한 상태에서 결합



Pi-Camera 제어

- 카메라 활성화



- 전원이 분리된 상태에서 카메라 모듈을 연결 한 후 전원을 연결



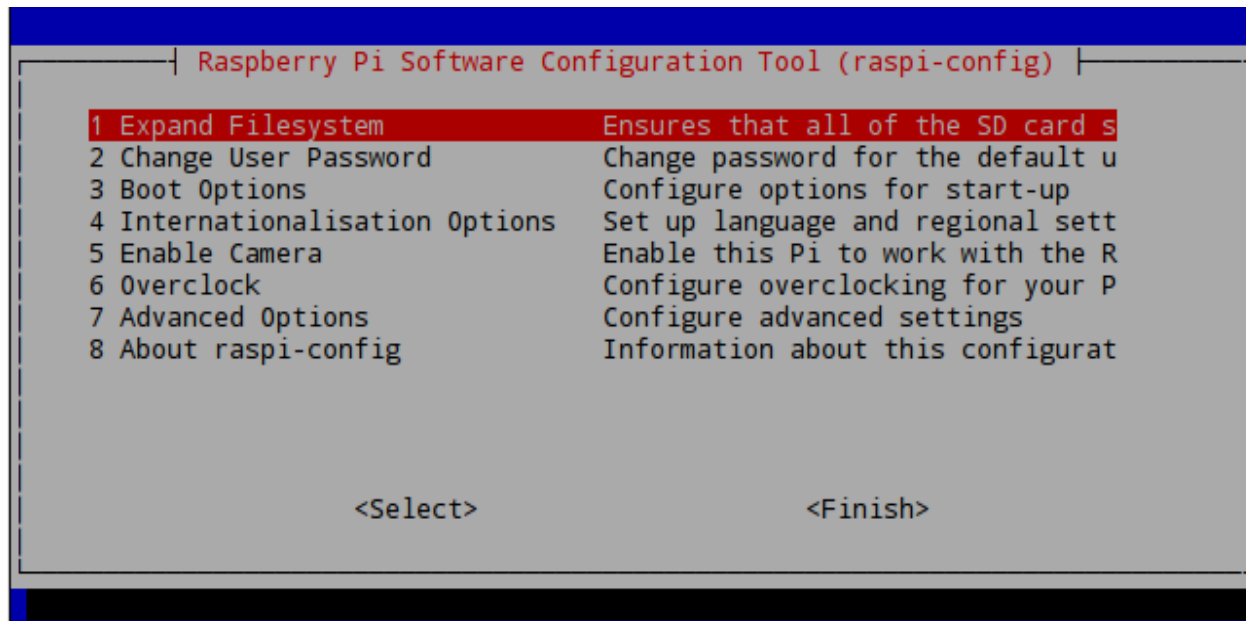
Pi-Camera 제어

- 카메라 활성화

- 카메라를 사용하기 위해서는 시스템 설정에서 카메라 사용을 활성화시켜줘야 함
- 다음과 같이 "camera" 디렉토리를 생성하고 명령어를 입력

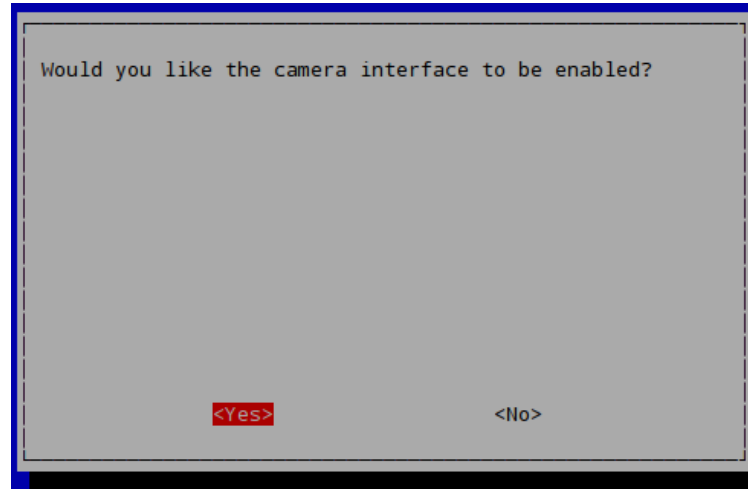
```
pi@raspberrypi:~ $ mkdir Camera
pi@raspberrypi:~ $ cd Camera
pi@raspberrypi:~/Camera $ sudo raspi-config
```

- raspi-config 실행



Pi-Camera 제어

- 카메라 활성화
 - 5. Enable Camera 선택



- “Yes”를 눌러 활성화 해준 다음, 시스템 설정에서 빠져 나와 시스템 재 시작

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

Pi-Camera 제어

- 쉘 명령어를 이용한 카메라 제어
 - 사진 촬영
 - raspistill : 라즈베리파이 기본 제공 명령어
 - 옵션을 붙여 사용
 - 터미널 창에 “raspistill -?” 입력
 - 옵션 확인

```
pi@raspberrypi:~/Camera $ raspistill -?
```

```
Image parameter commands
-?, --help      : This help information
-w, --width    : Set image width <size>
-h, --height   : Set image height <size>
-q, --quality   : Set jpeg quality <0 to 100>
-r, --raw       : Add raw bayer data to jpeg metadata
-o, --output    : Output filename <filename> (to write to stdout, use '-o -'). If
                  not specified, no file is saved
-l, --latest    : Link latest complete image to filename <filename>
-v, --verbose   : Output verbose information during run
-t, --timeout   : Time (in ms) before takes picture and shuts down (if not speci-
                  fied, set to 5s)
-th, --thumb    : Set thumbnail parameters (x:y:quality) or none
-d, --demo      : Run a demo mode (cycle through range of camera options, no cap-
                  ture)
-e, --encoding  : Encoding to use for output file (jpg, bmp, gif, png)
-x, --exif      : EXIF tag to apply to captures (format as 'key=value') or none
-tl, --timelapse : Timelapse mode. Takes a picture every <t>ms
-fp, --fullpreview : Run the preview using the still capture resolution (may
                  reduce preview fps)
-k, --keypress  : Wait between captures for a ENTER, X then ENTER to exit
-s, --signal    : Wait between captures for a SIGUSR1 from another process
-g, --gl        : Draw preview to texture instead of using video render componen-
                  t
-gc, --glcapture : Capture the GL frame-buffer instead of the camera imag-
                  e
-set, --settings : Retrieve camera settings and write to stdout
-cs, --camselect : Select camera <number>. Default 0
-bm, --burst     : Enable 'burst capture mode'
-md, --mode      : Force sensor mode. 0=auto. See docs for other modes available

Preview parameter commands
-p, --preview   : Preview window settings <'x,y,w,h'>
-f, --fullscreen : Fullscreen preview mode
-op, --opacity  : Preview window opacity (0-255)
-n, --nopreview : Do not display a preview window

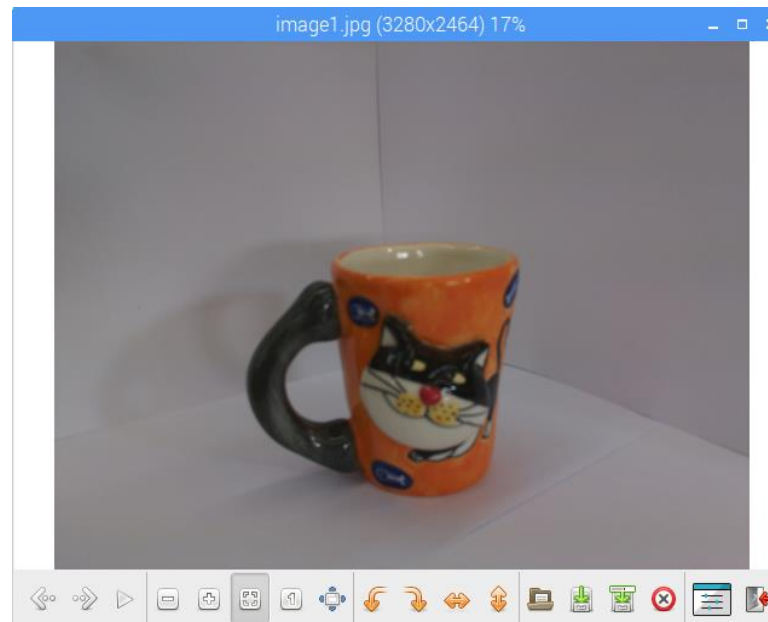
Image parameter commands
-sh, --sharpness : Set image sharpness (-100 to 100)
-co, --contrast  : Set image contrast (-100 to 100)
-br, --brightness : Set image brightness (0 to 100)
```

Pi-Camera 제어

- 셸 명령어를 이용한 카메라 제어
 - raspistill
 - 이미지 파일 저장
 - 사용방법 : raspistill -o [파일명]

```
pi@raspberrypi:~/Camera $ raspistill -o image1.jpg
```

- 프리뷰 화면을 보여주고 약 5초 후, 사진 촬영을 하고 프리뷰 화면이 닫힘
- 저장된 파일은 현재의 디렉터리에 저장

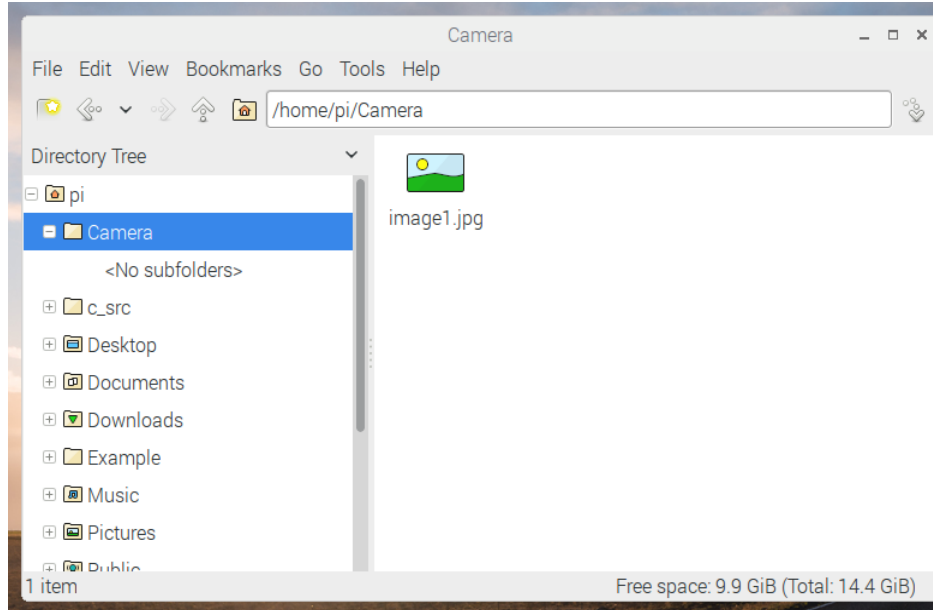


Pi-Camera 제어

- 셸 명령어를 이용한 카메라 제어
 - raspistill
 - 이미지 파일 열기
 - 사용방법 : gpicview [파일명]

```
pi@raspberrypi:~/Camera $ gpicview image1.jpg
```

- 또는 X-Windows 상에서 "File Manager"를 통해 img 파일 더블 클릭



Pi-Camera 제어

- 쉘 명령어를 이용한 카메라 제어

- raspistill

- 사진 촬영 전 시간 간격 조절
 - 기본적으로 5초 이후 촬영, 이를 조절 하기 위한 명령어
 - 사용 방법 : raspistill -t [msec]

```
pi@raspberrypi:~/Camera $ raspistill -t 1000 -o image2.jpg
```

- 위의 명령어를 입력하면 프리뷰 화면을 보여주고 1초후 사진을 촬영하고 화면이 닫힘
 - 옵션의 기본단위는 ms

Pi-Camera 제어

- 쉘 명령어를 이용한 카메라 제어
 - raspistill
 - 이미지 상하좌우 반전
 - Edge-Embedded의 카메라는 정면에서 봤을 때 반대로 영상을 찍고 있음



Pi-Camera 제어

- 셸 명령어를 이용한 카메라 제어
 - raspistill
 - 이미지 상하 반전
 - 사용 방법 : raspistill -vf

```
pi@raspberrypi:~/Camera $ raspistill -vf -t 5000 -o image3.jpg
```

- 이미지 좌우 반전
- 사용 방법 : raspistill -hf

```
pi@raspberrypi:~/Camera $ raspistill -hf -t 5000 -o image4.jpg
```

- 이미지 회전
- 사용 방법 : raspistill -rot [angle]

```
pi@raspberrypi:~/Camera $ raspistill -rot 90 -t 5000 -o image5.jpg
```

Pi-Camera 제어

- 쉘 명령어를 이용한 카메라 제어
 - 동영상 촬영
 - raspivid : 라즈베리파이 기본 제공 명령어
 - 옵션을 붙여 사용
 - 터미널 창에 "raspivid -?" 입력
 - 옵션 확인

```
pi@raspberrypi:~/Camera $ raspivid -?
```

```
pi@raspberrypi:~ $ raspivid -?
Display camera output to display, and optionally saves an H264 capture at requested bitrate

usage: raspivid [options]

Image parameter commands

-?, --help          : This help information
-w, --width         : Set image width <size>. Default 1920
-h, --height        : Set image height <size>. Default 1080
-b, --bitrate       : Set bitrate. Use bits per second (e.g. 10Mbits/s would be -b 10000000)
-o, --output        : Output filename <filename> (to write to stdout, use '-o -')
-v, --verbose       : Output verbose information during run
-t, --timeout       : Time (in ms) to capture for. If not specified, set to 5s. Zero to disable
-d, --demo          : Run a demo mode (cycle through range of camera options, no capture)
-fps, --framerate   : Specify the frames per second to record
-e, --penc          : Display preview image *after* encoding (shows compression artifacts)
-g, --intra         : Specify the intra refresh period (key frame rate/GoP size). Zero to produce an initial I-frame and then just P-frames.
-pf, --profile      : Specify H264 profile to use for encoding
-td, --timed        : Cycle between capture and pause. -cycle on,off where on is record time and off is pause time in ms
-s, --signal        : Cycle between capture and pause on Signal
-k, --keypress      : Cycle between capture and pause on ENTER
-i, --initial       : Initial state. Use 'record' or 'pause'. Default 'record'
-qp, --qp           : Quantisation parameter. Use approximately 10-40. Default 0 (off)
-ih, --inline       : Insert inline headers (SPS, PPS) to stream
-sg, --segment      : Segment output file in to multiple files at specified interval <ms>
-wr, --wrap         : In segment mode, wrap any numbered filename back to 1 when reach number
-sn, --start        : In segment mode, start with specified segment number
-sp, --split        : In wait mode, create new output file for each start event
```


Pi-Camera 제어

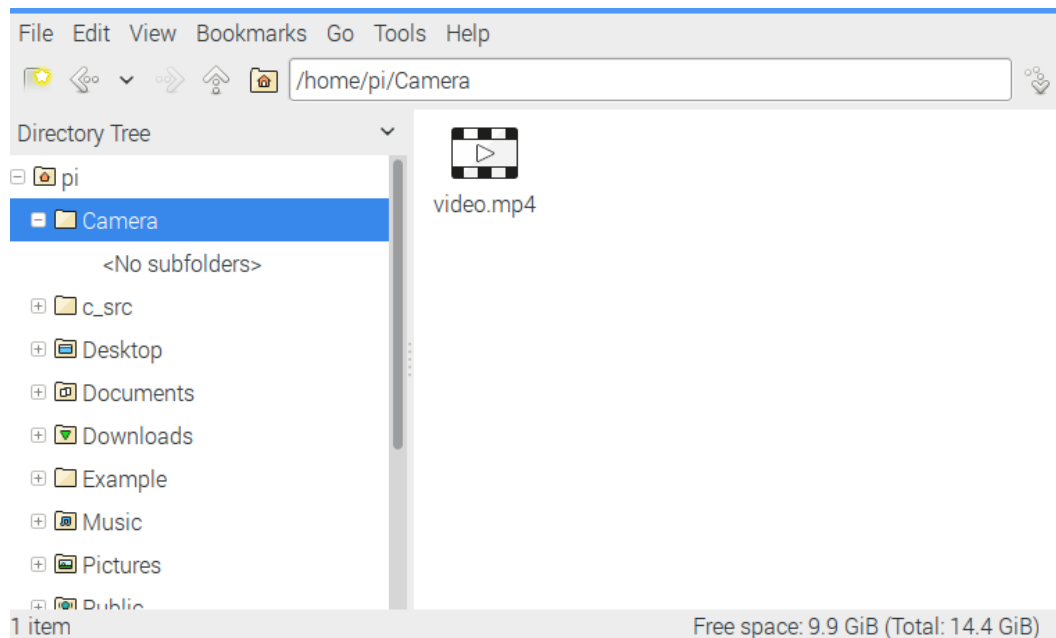
- 셸 명령어를 이용한 카메라 제어

- raspivid

- 동영상 파일 저장
 - 사용 방법 : raspivid -o [파일명]

```
pi@raspberrypi:~/Camera $ raspivid -o video.mp4
```

- 프리뷰 화면이 나타나고 5초후에 화면이 종료
 - 저장한 동영상 파일은 기본 디렉터리인 /home/pi/ 디렉터리에 저장



Pi-Camera 제어

- 셸 명령어를 이용한 카메라 제어
 - raspivid
 - 동영상 파일 재생
 - 사용 방법 : omxplayer [파일명]

```
pi@raspberrypi:~/Camera $ omxplayer video.mp4
```

- 5초간 저장된 영상 재생

Pi-Camera 제어

● Pi Camera Servo Motor 제어

- Edge-Embedded의 카메라 모듈에는 두 개의 Servo Motor가 장착되어 있어, 카메라를 상하좌우로 이동하여 촬영 가능
- Edge-Peripheral의 우측 상단에는 전원 스위치가 있고, 이 스위치의 On/Off에 따라 카메라 모듈의 Servo Motor를 GPIO, I2C통신으로 제어가능
- 전원 Off 시, GPIO17, 18(BCM Mode)번 과 연결

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

Pi-Camera 제어

- Pi Camera Servo Motor 제어 예제(1)

- 터미널 창에 “nano 18_CAMERA_01.c” 입력

```
pi@raspberrypi:~/Example $ nano 18_CAMERA_01.c
```

- 전원이 Off일 때, softPwm 라이브러리를 이용해 Servo Motor를 회전하고, 사진을 찍는 예제

```
1. // File : 18_CAMERA_01.c
2. #include <stdio.h>
3. #include <wiringPi.h>
4. #include <softPwm.h>           // Software PWM 라이브러리 참조
5. const int pinServoUD = 17;    // Up-Down Servo Motor 핀 설정
6. const int pinServoLR = 18;    // Left-Right Servo Motor 핀 설정
7. int main(void)
8. {
9.     wiringPiSetupGpio();       // 핀 번호를 BCM Mode로 설정
10.    softPwmCreate(pinServoUD, 0, 200); // 해당 핀을 PWM 핀, 0~200까지 범위 설정
11.    softPwmCreate(pinServoLR, 0, 200);
```

Pi-Camera 제어

- Pi Camera Servo Motor 제어 예제(1)

```
12. while(1)
13. {
14.     // Up Down Servo Motor 제어
15.     softPwmWrite(pinServoUD, 1);
16.     delay(1000);
17.     softPwmWrite(pinServoUD, 10);
18.     delay(1000);
19.     softPwmWrite(pinServoUD, 20);
20.     delay(1000);
21.     softPwmWrite(pinServoUD, 10);
22.     delay(1000);

23.     // Left Right Servo Motor 제어
24.     softPwmWrite(pinServoLR, 5);
25.     delay(1000);
26.     softPwmWrite(pinServoLR, 15);
27.     delay(1000);
28.     softPwmWrite(pinServoLR, 25);
29.     delay(1000);
30.     softPwmWrite(pinServoLR, 15);
31.     delay(1000);
```

Pi-Camera 제어

- Pi Camera Servo Motor 제어 예제(1)

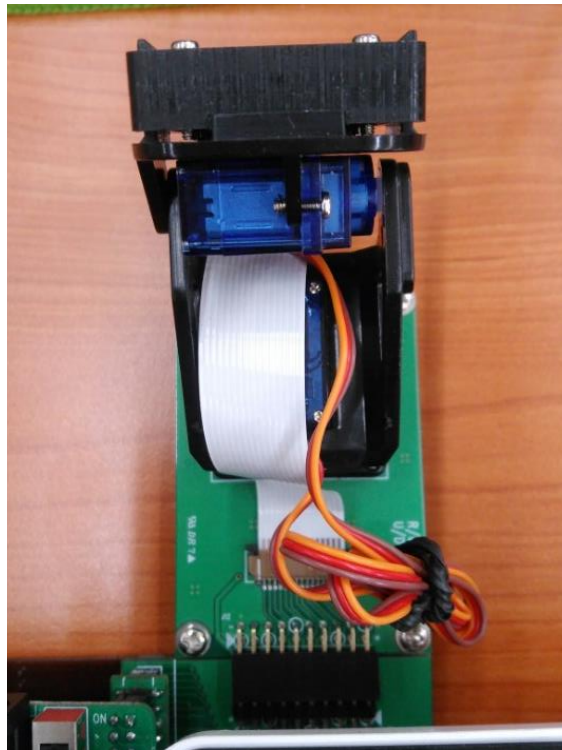
```
32. // 쉘스크립트 명령어를 프로그램 상에서 실행 시킬 경우 system() 사용
33. // 문자열 파라미터를 받아, 실행
34. system("raspistill -t 5000 -o image1.jpg");
35. }
36. return 0;
37. }
```

- 작성 후 "ctrl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
- GCC 컴파일러를 사용하여 빌드 및 생성된 "18_CAMERA_01" 파일 실행

```
pi@raspberrypi:~/Example $ gcc -o 18_CAMERA_01 18_CAMERA_01.c -lwiring
Pi
pi@raspberrypi:~/Example $ ./18_CAMERA_01
```

Pi-Camera 제어

- Pi Camera Servo Motor 제어 예제(1)
 - 결과
 - 전원이 Off인 상태에서 2개의 카메라 모듈의 Servo Motor가 회전을 하고, 5초 뒤 사진을 촬영



Pi-Camera 제어

- Pi Camera Servo Motor 제어 예제(2)

- 터미널 창에 “nano 18_CAMERA_02.c” 입력

```
pi@raspberrypi:~/Example $ nano 18_CAMERA_01.c
```

- 전원이 On일 때, I2C 통신을 이용해 Servo Motor를 회전하고, 사진을 찍는 예제

```
1. // File : 18_CAMERA_02.c
2. #include <stdio.h>
3. #include <wiringPi.h>
4. #include <wiringPiI2C.h>           // I2C 라이브러리 참조
5. #define SERVO_I2C_ADDR            0x20           // STEP I2C 주소
6. // 제어 레지스터
7. #define OUT_PORT0                  0x02
8. #define CONFIG_PORT0               0x06
9. const int period = 20000;           // 20msec 주기
10. ons tint servoUD = 0x01;           // IO0_0 UD Servo Pin
11. ons tint servoLR = 0x02;           // IO0_1 LR Servo Pin
12. int fd;                             // Servo Motor의 handle
```


Pi-Camera 제어

● Pi Camera Servo Motor 제어 예제(2)

```
13. int main(void)
14. {
15.     // I2C 시스템 초기화 설정
16.     if((fd = wiringPiI2Csetup(SERVO_I2C_ADDR)) < 0)
17.     {
18.         return -1;                // 설정이 안될 경우, 종료
19.     }

20.     // handle을 통해 Configuration 레지스터를 출력모드로 설정
21.     wiringPiI2CwriteReg16(fd, CONFIG_PORT0, 0x00);

22.     int i;

23.     while(1)
24.     {
25.         // 1초마다 UD 회전, 주기가 20msec이므로 20msec * 50 = 1 sec
26.         // 카메라 모듈 Up Down
27.         for(i=0; i<50; i++)
28.         {
29.             wiringPiI2CWriteReg16(fd, OUT_PORT0, servoUD);    // HIGH 상태로 설정
30.             delayMicroseconds(50);
31.             wiringPiI2CWriteReg16(fd, OUT_PORT0, 0x00);        // LOW 상태로 설정
32.             delayMicroseconds(period-50);
33.         }
```

Pi-Camera 제어

● Pi Camera Servo Motor 제어 예제(2)

```
34.     for(i=0; i<50; i++)
35.     {
36.         wiringPi2CWriteReg16(fd, OUT_PORT0, servoUD);    // HIGH 상태로 설정
37.         delayMicroseconds(600);
38.         wiringPi2CWriteReg16(fd, OUT_PORT0, 0x00);        // LOW 상태로 설정
39.         delayMicroseconds(period-600);
40.     }

41.     for(i=0; i<50; i++)
42.     {
43.         wiringPi2CWriteReg16(fd, OUT_PORT0, servoUD);    // HIGH 상태로 설정
44.         delayMicroseconds(1500);
45.         wiringPi2CWriteReg16(fd, OUT_PORT0, 0x00);        // LOW 상태로 설정
46.         delayMicroseconds(period-1500);
47.     }

48.     // 카메라 모듈 Left Right
49.     for(i=0; i<50; i++)
50.     {
51.         wiringPi2CWriteReg16(fd, OUT_PORT0, servoLR);    // HIGH 상태로 설정
52.         delayMicroseconds(50);
53.         wiringPi2CWriteReg16(fd, OUT_PORT0, 0x00);        // LOW 상태로 설정
54.         delayMicroseconds(period-50);
55.     }
```

Pi-Camera 제어

● Pi Camera Servo Motor 제어 예제(2)

```
56.     for(i=0; i<50; i++)
57.     {
58.         wiringPiI2CWriteReg16(fd, OUT_PORT0, servoLR);    // HIGH 상태로 설정
59.         delayMicroseconds(1000);
60.         wiringPiI2CWriteReg16(fd, OUT_PORT0, 0x00);        // LOW 상태로 설정
61.         delayMicroseconds(period-1000);
62.     }

63.     for(i=0; i<50; i++)
64.     {
65.         wiringPiI2CWriteReg16(fd, OUT_PORT0, servoLR);    // HIGH 상태로 설정
66.         delayMicroseconds(2000);
67.         wiringPiI2CWriteReg16(fd, OUT_PORT0, 0x00);        // LOW 상태로 설정
68.         delayMicroseconds(period-2000);
69.     }
70.     // 셸스크립트 명령어를 프로그램 상에서 실행 시킬 경우 system() 사용
71.     // 문자열 파라미터를 받아, 실행
72.     system("raspistill -t 5000 -o image2.jpg");

73. }
74. return 0;
75. }
```

Pi-Camera 제어

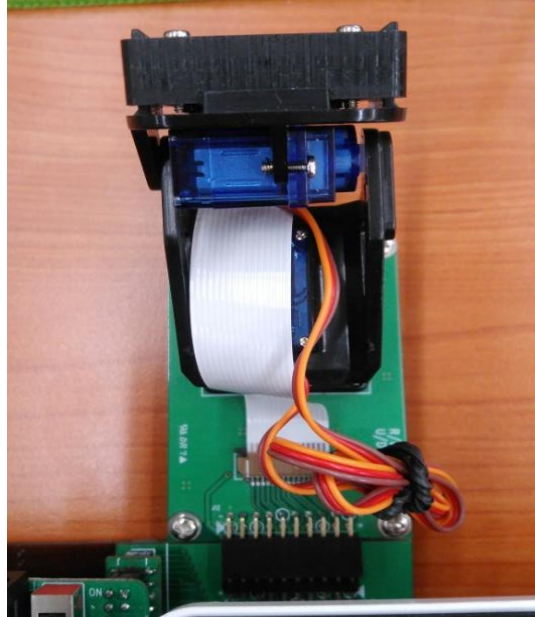
- Pi Camera Servo Motor 제어 예제(2)

- 작성 후 "ctrl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
- GCC 컴파일러를 사용하여 빌드 및 생성된 "18_CAMERA_02" 파일 실행

```
pi@raspberrypi:~/Example $ gcc -o 18_CAMERA_02 18_CAMERA_02.c -lwiringPi
pi@raspberrypi:~/Example $ ./18_CAMERA_02
```

- 결과

- 전원 스위치가 On인 상태에서 2개의 카메라 모듈의 Servo 모터가 회전하고, 5초 뒤에 사진을 촬영



적외선(IR) 송수신 제어

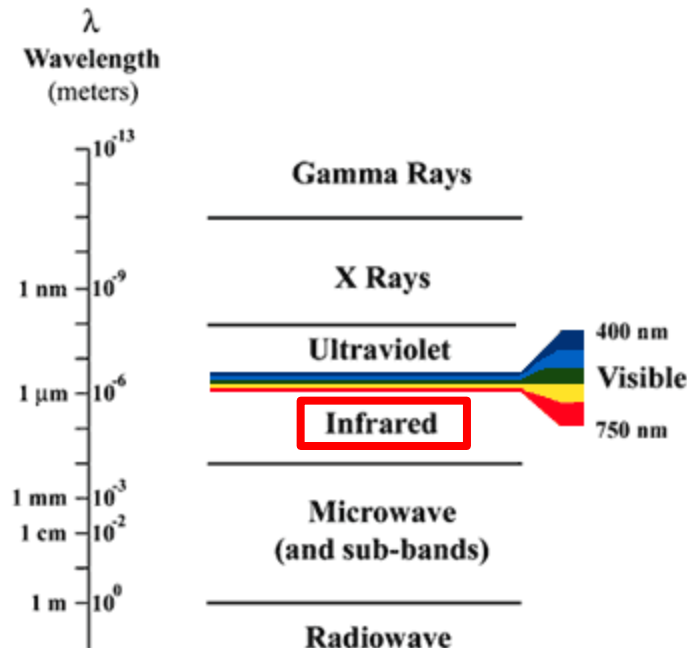
적외선(IR) 송수신 제어

- 적외선(InfraRed light)이란
 - 가시광선보다 파장이 긴 전자기파
 - 우리가 보통 "빛"이라고 생각하고 있는 것은 전자기파 중에서 눈으로 볼 수 있는 부분을 말한다.
 - 가시광선 영역에서 빨간 색 쪽으로 벗어나므로 적외선이라 부름
 - 적외선도 빛이지만 사람 눈에는 보이지 않음
 - 단 자외선과 마찬가지로 일부 동물과 곤충들은 적외선을 볼 수 있음
 - 파장의 범위는 $0.74 \sim 300 \mu\text{m}$ 정도



적외선(IR) 송수신 제어

- 적외선(InfraRed light)이란
 - 적외선은 여러 용도로 사용되고 있으며, 군사 분야에서는 표적 탐지 및 추적에 쓰이고 농업용이나 의료용으로도 쓰임
 - 실생활에서는 주로 가전 제품을 이용할 때 원격으로 제어할 수 있게끔 적외선 송수신기를 사용
 - Edge-Embedded에는 적외선 송신기(리모콘)과 적외선을 수신할 수 있는 IR Receiver가 제공



적외선(IR) 송수신 제어

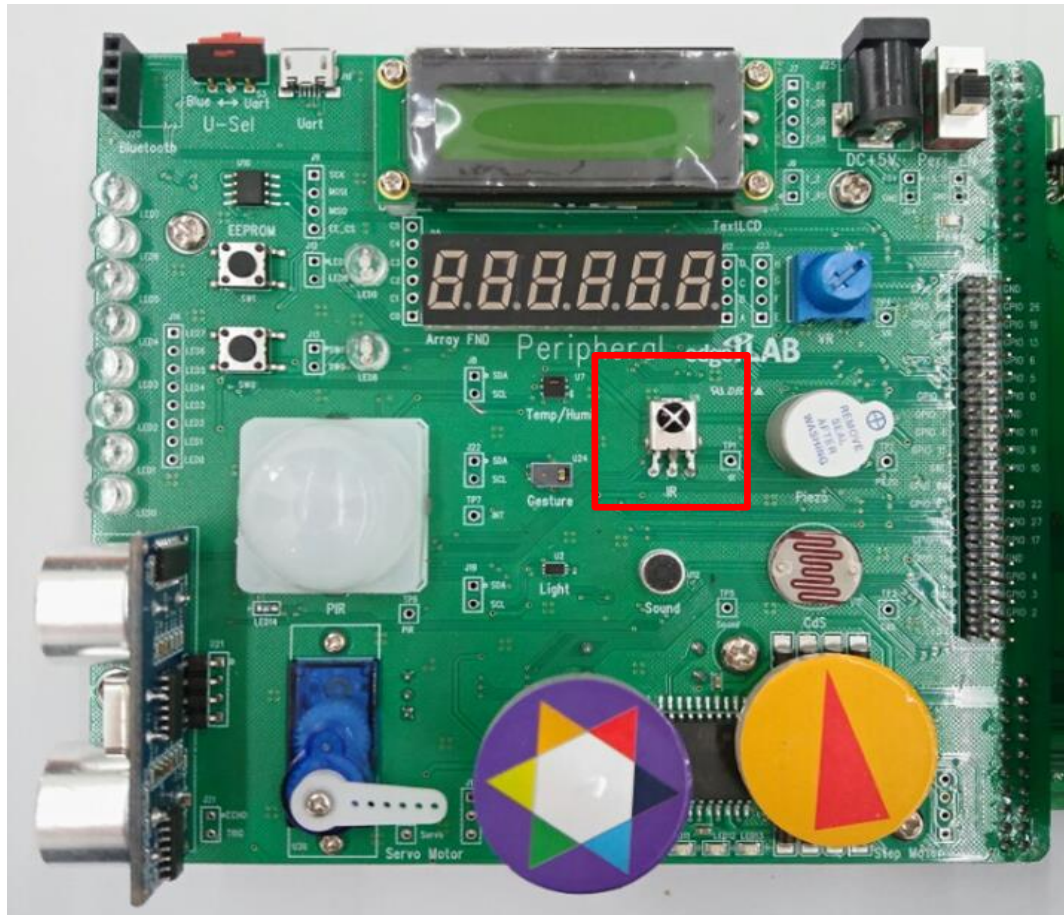
- 적외선(IR) 제어
 - Edge-Embedded에서 IR Reciver는 19번(BCM Mode)핀과 연결

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



적외선(IR) 송수신 제어

- 적외선(IR) 제어
 - Edge-Embedded의 IR Receiver 센서



적외선(IR) 송수신 제어

- 적외선(IR) 제어 예제(1)

- 터미널 창에 “nano 19_IR_01.c” 입력

```
pi@raspberrypi:~/Example $ nano 19_IR_01.c
```

- IR 수신기를 입력모드로 두고 리모콘으로 버튼을 누를 때마다 터미널 화면에 “Read”를 출력하는 예제

```
1. // File : 19_IR_01.c
2. #include <stdio.h>
3. #include <wiringPi.h>
4. const int pinIr = 19; // IR receiver가 연결된 핀(BCM Mode)
5. int main(void)
6. {
7.     wiringPiSetupGpio(); // 핀 번호를 BCM Mode로 설정
8.     pinMode(pinIr, INPUT); // IR 핀을 입력모드로 설정
9.     while(1)
10.    {
11.        if(!digitalRead(pinIr)) // IR 핀의 상태를 확인
12.        {
```

적외선(IR) 송수신 제어

● 적외선(IR) 제어 예제(1)

```
13.      printf("Read\n");           // "Read" 출력
14.      delay(100);
15.  }
16. }
17. return 0;
18. }
```

- 작성 후 "ctrl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
- GCC 컴파일러를 사용하여 빌드 및 생성된 "19_IR_01" 파일 실행

```
pi@raspberrypi:~/Example $ gcc -o 19_IR_01 19_IR_01.c -lwiringPi
pi@raspberrypi:~/Example $ ./19_IR_01
```

- 결과
 - IR Receiver로 수신되는 적외선이 있을 경우, 터미널 창에 "Read" 출력

```
pi@raspberrypi:~/Example $ ./19_IR_01
Read
Read
Read
```

적외선(IR) 송수신 제어

- LIRC 패키지

- LIRC(Linux Infrared Remote Control)란

- 일반적으로 사용되는 적외선 리모컨 신호를 디코딩하고 전송할 수 있게 제공하는 패키지
 - LIRC의 핵심은 장치 드라이버가 수신한 IR 신호를 디코딩하고 소켓에 정보를 제공하는 lircd 데몬

- LIRC 설치 및 설정

- 터미널 창에 “sudo apt-get install lirc liblircclient-dev” 입력

```
pi@raspberrypi:~ $ sudo apt-get install lirc liblircclient-dev
```

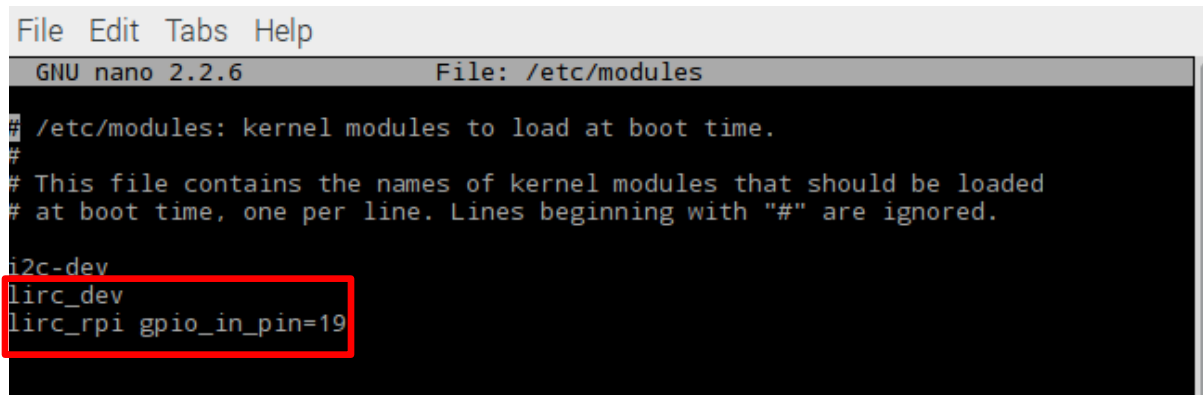
```
pi@raspberrypi:~ $ sudo apt-get install lirc liblircclient-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  lirc-x setserial ir-keytable
The following NEW packages will be installed:
  liblircclient-dev lirc
0 upgraded, 2 newly installed, 0 to remove and 233 not upgraded.
Need to get 0 B/374 kB of archives.
After this operation, 1,536 kB of additional disk space will be used.
Selecting previously unselected package liblircclient-dev.
(Reading database ... 124093 files and directories currently installed.)
Preparing to unpack .../liblircclient-dev_0.9.0~pre1-1.2_armhf.deb ...
Unpacking liblircclient-dev (0.9.0~pre1-1.2) ...
Selecting previously unselected package lirc.
Preparing to unpack .../lirc_0.9.0~pre1-1.2_armhf.deb ...
Unpacking lirc (0.9.0~pre1-1.2) ...
Processing triggers for man-db (2.7.0.2-5) ...
Processing triggers for systemd (215-17+deb8u5) ...
Setting up liblircclient-dev (0.9.0~pre1-1.2) ...
Setting up lirc (0.9.0~pre1-1.2) ...
Processing triggers for systemd (215-17+deb8u5) ...
```

적외선(IR) 송수신 제어

- LIRC 패키지
 - 설치 완료 후, 모듈에 GPIO 내용 추가
 - 터미널 창에 "sudo nano /etc/modules" 입력

```
pi@raspberrypi:~ $ sudo nano /etc/modules
```

- /etc/modules에 아래 내용 추가
- lirc_dev
lirc_rpi gpio_in_pin=19



```
File Edit Tabs Help
GNU nano 2.2.6 File: /etc/modules

# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.

i2c-dev
lirc_dev
lirc_rpi gpio_in_pin=19
```

적외선(IR) 송수신 제어

- LIRC 패키지

- Lirc에서 사용할 하드웨어 정보 수정
 - /etc/lirc/hardware.conf 내용 수정

LIRCD_ARGS="--input"

DRIVER="default"

DEVICE="/dev/lirc0"

MODULES="lirc_rpi"

- 터미널 창에 "sudo nano /etc/lirc/hardware.conf" 입력

```
pi@raspberrypi:~ $ sudo nano /etc/lirc/hardware.conf

GNU nano 2.2.6      File: /etc/lirc/hardware.conf      Modified

# /etc/lirc/hardware.conf
#
# Arguments which will be used when launching lircd
LIRCD_ARGS="--input"

#Don't start lircmd even if there seems to be a good config file
#START_LIRCMD=false

#Don't start irexec, even if a good config file seems to exist.
#START_IREXEC=false

#Try to load appropriate kernel modules
LOAD_MODULES=true

# Run "lircd --driver=help" for a list of supported drivers.
DRIVER="default"
# usually /dev/lirc0 is the correct setting for systems using udev
DEVICE="/dev/lirc0"
MODULES="lirc_rpi"

# Default configuration files for your hardware if any
LIRCD_CONF=""
LIRCMD_CONF=""
```

적외선(IR) 송수신 제어

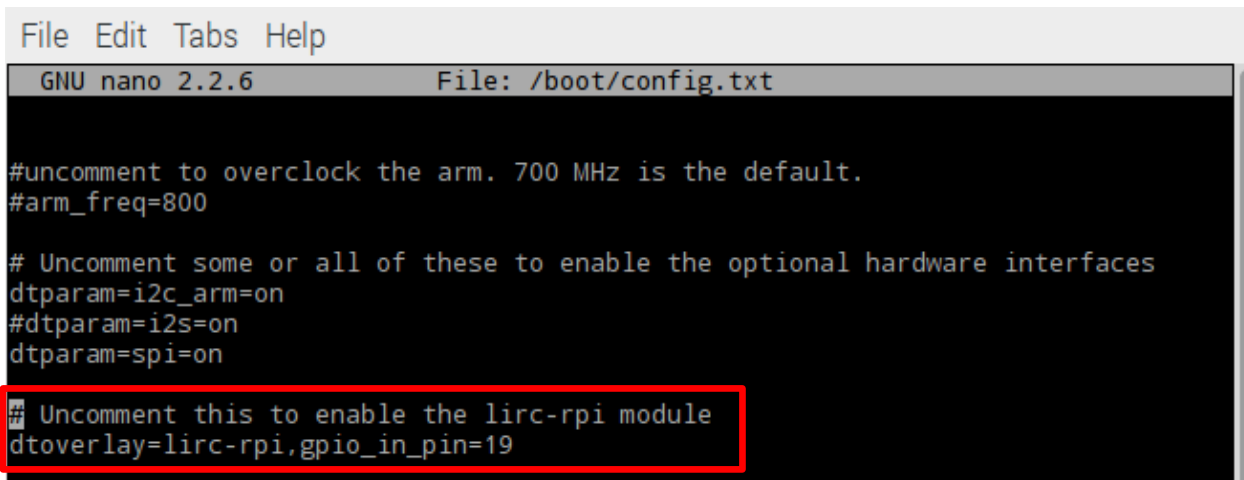
- LIRC 패키지

- config.txt 파일 수정

- 부팅 시, lirc-rpi module을 사용하기 위함
 - 터미널 창에 "sudo nano /boot/config.txt" 입력

```
pi@raspberrypi:~ $ sudo nano /boot/config.txt
```

- "#Uncomment this to enable the lirc-rpi module" 이라는 문장 밑에 내용을 추가
 - /boot/config.txt 내용 추가
dtoverlay=lirc-rpi,gpio_in_pin=19



```
File Edit Tabs Help
GNU nano 2.2.6 File: /boot/config.txt

#uncomment to overclock the arm. 700 MHz is the default.
#arm_freq=800

# Uncomment some or all of these to enable the optional hardware interfaces
dtparam=i2c_arm=on
#dtparam=i2s=on
dtparam=spi=on

# Uncomment this to enable the lirc-rpi module
dtoverlay=lirc-rpi,gpio_in_pin=19
```

적외선(IR) 송수신 제어

- 동작 여부 확인

- 터미널 창에 “dmesg | grep lirc” 입력

- 시스템 부팅 메시지에서 “lirc”가 들어가 있는 모든 행을 출력

```
pi@raspberrypi:~$ dmesg | grep lirc
```

- dmesg란

- display message 또는 driver message로, 시스템 부팅 메시지를 확인하는 명령어

- grep란

- 파일 내에서 지정한 패턴이나 문자열을 찾은 후, 그 패턴을 포함하고 있는 모든 행을 출력하는 명령어

File Edit Tabs Help

```
pi@raspberrypi:~$ dmesg | grep lirc
[ 2.815825] lirc_dev: IR Remote Control driver registered, major 245
[ 3.394018] lirc_rpi: module is from the staging directory, the quality is unknown, you have been warned.
[ 4.348461] lirc_rpi: auto-detected active low receiver on GPIO pin 19
[ 4.348813] lirc_rpi lirc_rpi: lirc_dev: driver lirc_rpi registered at minor = 0
[ 4.348828] lirc_rpi: driver registered!
[ 6.359600] input: lircd as /devices/virtual/input/input1
```


적외선(IR) 송수신 제어

- 동작 여부 확인
 - 터미널 창에 “sudo /etc/init.d/lirc stop” 입력
 - lirc 서비스 중지

```
pi@raspberrypi:~ $ sudo /etc/init.d/lirc stop
```

- /etc/init.d 디렉터리는 서비스 관리 프로그램인 init 프로세스가 사용하는 스크립트들을 포함
- Init프로세스는 커널이 초기화되고 나서 가장 처음 실행되는 프로세스
- 특정한 서비스(bluetooth, mysql, lirc 등)들을 시작, 정지 가능

File Edit Tabs Help

```
pi@raspberrypi:~ $ sudo /etc/init.d/lirc stop  
[ ok ] Stopping lirc (via systemctl): lirc.service.
```

적외선(IR) 송수신 제어

- 동작 여부 확인
 - 터미널 창에 "mode2 -d /dev/lirc0"입력
 - 수신되는 적외선 출력

```
pi@raspberrypi:~ $ mode2 -d /dev/lirc0
```

- Mode2란
 - Lirc 드라이버에서 사용 가능한 커널 출력을 표시
- 옵션 -d
 - 지정된 디바이스로부터 값을 읽는다는 의미
 - "dev/lirc0"디바이스로부터 입력 받은 값을 출력한다는 의미

```
pi@raspberrypi:~ $ mode2 -d /dev/lirc0
space 2765695
pulse 9144
space 4549
pulse 569
space 597
pulse 572
space 597
pulse 571
space 597
pulse 570
```

- "ctrl + C"를 눌러 중지

적외선(IR) 송수신 제어

- 동작 여부 확인

- 적외선 리모컨 설정

- 터미널 창에 "irrecord -d /dev/lirc0 ./lircd.conf" 입력

- 키 값 설정

```
pi@raspberrypi:~ $ irrecord -d /dev/lirc0 ./lircd.conf
```

- 현재 기본 디렉터리인 "/home/pi"에 "lircd.conf"파일 생성

```
pi@raspberrypi:~ $ irrecord -d /dev/lirc0 ./lircd.conf
irrecord - application for recording IR-codes for usage with lirc
Copyright (C) 1998,1999 Christoph Bartelmus(lirc@bartelmus.de)

This program will record the signals from your remote control
and create a config file for lircd.

A proper config file for lircd is maybe the most vital part of this
package, so you should invest some time to create a working config
file. Although I put a good deal of effort in this program it is often
not possible to automatically recognize all features of a remote
control. Often short-comings of the receiver hardware make it nearly
impossible. If you have problems to create a config file READ THE
DOCUMENTATION of this package, especially section "Adding new remote
controls" for how to get help.

If there already is a remote control of the same brand available at
http://www.lirc.org/remotes/ you might also want to try using such a
remote as a template. The config files already contain all
parameters of the protocol used by remotes of a certain brand and
knowing these parameters makes the job of this program much
easier. There are also template files for the most common protocols
available in the remotes/generic/ directory of the source
distribution of this package. You can use a template files by
providing the path of the file as command line parameter.

Please send the finished config files to <lirc@bartelmus.de> so that I
can make them available to others. Don't forget to put all information
that you can get about the remote control in the header of the file.

Press RETURN to continue.
```

적외선(IR) 송수신 제어

- 동작 여부 확인
 - 적외선 리모컨 설정
 - 엔터를 누르고 마침표가 줄을 채울 때 까지 여러 버튼들을 1초에 한번씩 눌러 입력

```
Now start pressing buttons on your remote control.  
  
It is very important that you press many different buttons and hold them  
down for approximately one second. Each button should generate at least one  
dot but in no case more than ten dots of output.  
Don't stop pressing buttons until two lines of dots (2x80) have been  
generated.  
  
Press RETURN now to start recording.  
.....
```

- 정상적으로 인식되었으면 "Now enter the names for the buttons"이라는 문구가 출력되고 키의 이름 등록 가능

```
Press RETURN now to start recording.  
.....  
Found const length: 108639  
Please keep on pressing buttons like described above.  
.....  
Space/pulse encoded remote control found.  
Signal length is 67.  
Found possible header: 9156 4532  
Found trail pulse: 565  
Found repeat code: 9142 2260  
Signals are space encoded.  
Signal length is 32  
Now enter the names for the buttons.  
  
Please enter the name for the next button (press <ENTER> to finish recording)  
|
```

적외선(IR) 송수신 제어

- 동작 여부 확인
 - 적외선 리모컨 설정
 - 키의 이름을 "KEY_0"으로 설정

```
Now enter the names for the buttons.  
  
Please enter the name for the next button (press <ENTER> to finish recording)  
KEY_0  
  
Now hold down button "KEY_0".  
  
Please enter the name for the next button (press <ENTER> to finish recording)
```

- 리모컨의 '0' 버튼을 등록하고 싶다면
 - "KEY_0" 을 입력 후 엔터
 - 리모컨의 '0' 버튼을 1초정도 누르면 인식 완료
- KEY_0부터 KEY_9까지 입력 후, 마지막에는 키의 이름 없이 Enter

```
Checking for toggle bit mask.  
Please press an arbitrary button repeatedly as fast as possible.  
Make sure you keep pressing the SAME button and that you DON'T HOLD  
the button down!.  
If you can't see any dots appear, then wait a bit between button presses.  
  
Press RETURN to continue.  
.....  
Toggle bit mask is 0x2020.  
Successfully written config file.
```

적외선(IR) 송수신 제어

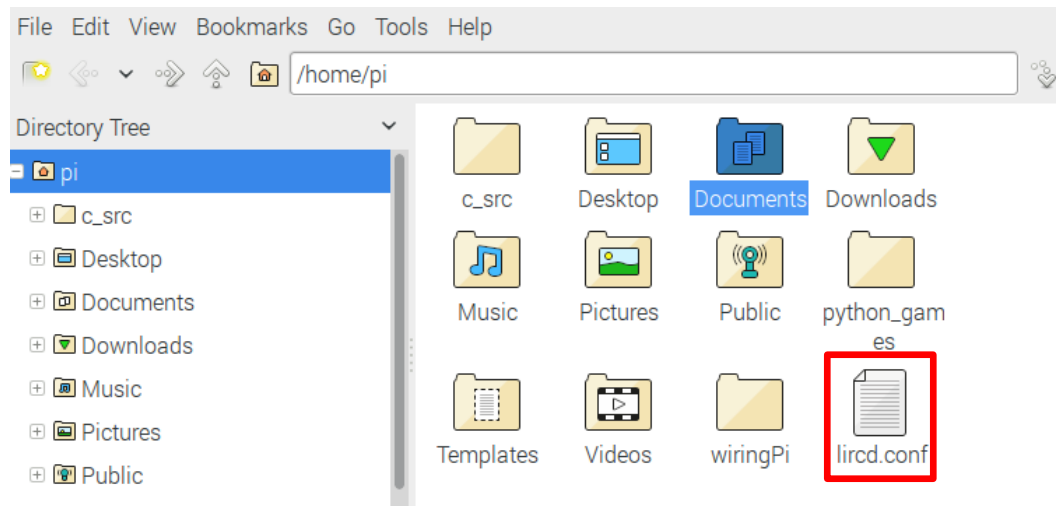
- 동작 여부 확인
 - 적외선 리모컨 설정
 - 마지막으로, 임의의 버튼 하나를 계속 눌러 Toggle 비트 마스크를 체크
 - 키의 이름은 표준 네임스페이스에 존재하는 이름으로만 작성하는 것을 권장
 - 터미널 창에 "irrecord -list -namespace" 입력
 - 사용 가능한 표준 네임스페이스 확인

```
pi@raspberrypi:~ $ irrecord -list-namespace
```

```
pi@raspberrypi:~ $ irrecord --list-namespace
KEY_0
KEY_102ND
KEY_1
KEY_2
KEY_3
KEY_4
KEY_5
KEY_6
KEY_7
KEY_8
KEY_9
KEY_A
KEY_AB
KEY_ADDRESSBOOK
KEY_AGAIN
KEY_ALTERASE
KEY_ANGLE
KEY_APOSTROPHE
```

적외선(IR) 송수신 제어

- 동작 여부 확인
 - 적외선 리모컨 설정
 - 정상적으로 처리가 완료되면, 기본 경로 디렉터리에 "lircd.conf"파일이 생성됐음을 확인 가능



- 터미널 창에 "cat ./lircd.conf"입력

```
pi@raspberrypi:~ $ cat ./lircd.conf
```

적외선(IR) 송수신 제어

- 동작 여부 확인
 - 적외선 리모컨 설정

```
begin remote

  name    ./lircd.conf
  bits    16
  flags   SPACE_ENC|CONST_LENGTH
  eps     30
  aeps    100

  header   9156 4532
  one      566 1687
  zero     566 602
  ptrail   565
  repeat   9142 2260
  pre_data_bits 16
  pre_data 0xFF
  gap      108639
  toggle_bit_mask 0x2020

  begin codes
    KEY_0      0x6897
    KEY_1      0x30CF
    KEY_2      0x18E7
    KEY_3      0x7A85
    KEY_4      0x10EF
    KEY_5      0x38C7
    KEY_6      0x5AA5
    KEY_7      0x42BD
    KEY_8      0x4AB5
    KEY_9      0x52AD
  end codes

end remote
```


적외선(IR) 송수신 제어

- 동작 여부 확인
 - 적외선 리모컨 설정
 - 이는 우리가 사용하고 있는 적외선 리모컨 버튼의 주파수를 코드화 한 것
 - 터미널 창에 "sudo cp ./lircd.conf /etc/lirc/lircd.conf" 입력
 - 터미널 창에 "sudo /etc/init.d/lirc restart" 입력
 - lirc 서비스에 복사하고 lirc 서비스를 재시작

```
pi@raspberrypi:~ $ sudo cp ./lircd.conf /etc/lirc/lircd.conf
pi@raspberrypi:~ $ sudo /etc/init.d/lirc restart
```

```
File Edit Tabs Help
pi@raspberrypi:~ $ sudo /etc/init.d/lirc restart
[ ok ] Restarting lirc (via systemctl): lirc.service.
```

적외선(IR) 송수신 제어

- 동작 확인
 - 터미널 창에 "irw" 입력
 - 수신되는 적외선 값 및 키 이름 확인

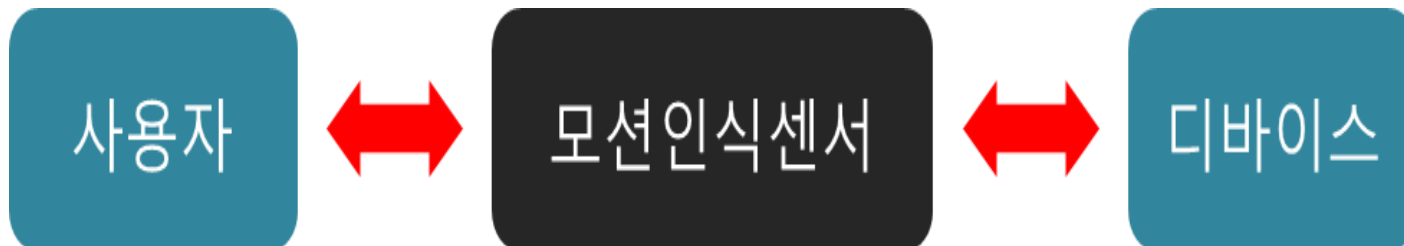
```
pi@raspberrypi:~ $ irw
```

```
File Edit Tabs Help
pi@raspberrypi:~ $ irw
0000000000ff6897 00 KEY_0 ./lircd.conf
0000000000ff6897 01 KEY_0 ./lircd.conf
0000000000ff30cf 00 KEY_1 ./lircd.conf
0000000000ff30cf 00 KEY_1 ./lircd.conf
0000000000ff18e7 00 KEY_2 ./lircd.conf
0000000000ff4ab5 00 KEY_8 ./lircd.conf
0000000000ff52ad 00 KEY_9 ./lircd.conf
```

GESTURE 제어

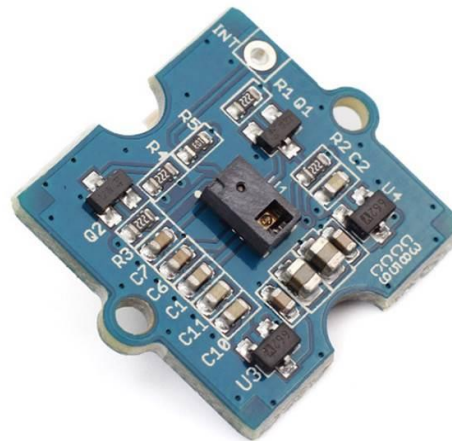
Gesture 제어

- 모션인식 기술이란
 - 특정한 물체의 움직임이나 위치를 인식하는 센서를 이용한 기술을 통칭하는 기술
 - 센서는 신체 움직임을 인식하고, 이를 디바이스와 상호작용하는 중간 연결고리 기능을 담당
 - 모션 인식 기술을 가능하게 하는 센서들의 종류
 - 자이로스코프 : 가속도를 측정해 물체의 방향 변화를 감지
 - 가속도 센서 : 지표면을 기준으로 가속도와 기울기를 이용해 물체의 움직임을 측정
 - 제스처 센서 : 빛을 방출하고 물체로부터 반사된 반사광을 감지하여 물체의 움직임을 판독



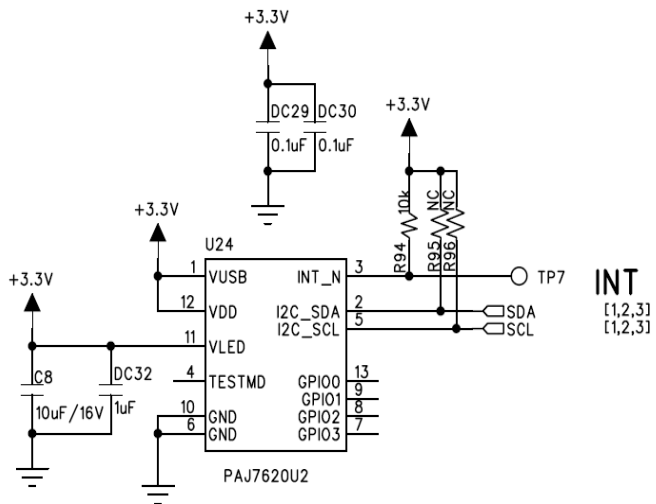
Gesture 제어

- PAJ7620U2
 - Grove-gesture센서는 일반적인 I2C인터페이스와 제스처 인식 기능을 단일 칩에 통합한 PAJ7620U2를 사용한 센서
 - PAJ7620U2는 IR LED 및 광학 렌즈가 내장되어 있는 비 접촉 방식의 제스처 센서로 모듈화된 형태로 제작
 - 사람의 손동작 제스처에 따라 9가지 동작을 인식
 - Up, Down, Left, Right, Forward, Backward, Clockwise, Count Clockwise, Wave
 - 5~15cm 범위에서 센서에 접근하거나 멀어지는 물체 감지 가능
 - 제스처 센서에 근접 감지 기능 내장
 - 저전력으로 동작, 절전 모드 사용 가능



Gesture 제어

- Edge-Embedded의 Gesture 센서
 - Google 검색
 - “PAJ7602U2” Data Sheet 참조



Pin #	Symbol	Type	Function
1	Vbus	Power	BUS power supply
2	SDA	IN/OUT	I2C data pin (Open Drain)
3	INT_N	OUT	Interrupt pin(Active low) (Open Drain)
4	TESTMD	IN	For Module Test Only
5	SCL	IN	I2C clock pin (Open Drain)
6, 10	GND	GND	Ground
7	GPIO3 (SPI_DATA)	SPI Mode : OUT GPIO Mode : IN/OUT	SPI Mode : Data out of SPI master GPIO Mode : GPIO
8	GPIO2 (SPI_SCK)	SPI Mode : OUT GPIO Mode : IN/OUT	SPI Mode : SCK signal of SPI master GPIO Mode : GPIO
9	GPIO1 (SPI_nCS)	SPI Mode : OUT GPIO Mode : IN/OUT	SPI Mode : Chip select signal of SPI master (Active low) GPIO Mode : GPIO
11	VLED	POWER	LED power input
12	VDD	POWER	Main Power supply
13	GPIO0 (SPI_MCLK)	SPI Mode : IN GPIO Mode : IN/OUT	SPI Mode : External clock input for SPI GPIO Mode : GPIO

Gesture 제어

- Gesture 센서 제어
 - 라이브러리
 - https://github.com/DBOpenSource/db_samples/tree/master/Linaro/Sample-PAJ7620
 - "Paj7620.cpp" 및 "Paj7620.h" 다운로드
 - 다운로드 후, 외장 USB 메모리를 이용하여 파일 이동

DBOpenSource / db_samples

Watch 4 Star 3 Fork 5

Code Issues 0 Pull requests 0 Projects 0 Insights

Branch: master db_samples / Linaro / Sample-PAJ7620 /

Create new file Find file History

Sujai SyrilRaj Initial version of the dragonboard 410c sensor samples. Latest commit e431a44 on 10 Dec 2015

..		
.settings	Initial version of the dragonboard 410c sensor samples.	2 years ago
.cproject	Initial version of the dragonboard 410c sensor samples.	2 years ago
.project	Initial version of the dragonboard 410c sensor samples.	2 years ago
PAJ7620.cpp	Initial version of the dragonboard 410c sensor samples.	2 years ago
PAJ7620.h	Initial version of the dragonboard 410c sensor samples.	2 years ago
README.md	Initial version of the dragonboard 410c sensor samples.	2 years ago
i2c-dev.h	Initial version of the dragonboard 410c sensor samples.	2 years ago
main.cpp	Initial version of the dragonboard 410c sensor samples.	2 years ago

README.md

Gesture 제어

- Gesture 센서 제어
 - 라이브러리
 - 또는 "git clone" 명령어를 이용해 다운
 - 터미널 창에 "git clone http://github.com/DBOpenSource/db_samples.git" 입력

```
pi@raspberrypi:~ $ git clone https://github.com/DBOpenSource/db_sample
s.git
```

```
pi@raspberrypi:~ $ git clone https://github.com/DBOpenSource/db_samples
Cloning into 'db_samples'...
remote: Counting objects: 245, done.
remote: Total 245 (delta 0), reused 1 (delta 0), pack-reused 244
Receiving objects: 100% (245/245), 3.03 MiB | 539.00 KiB/s, done.
Resolving deltas: 100% (56/56), done.
Checking connectivity... done.
pi@raspberrypi:~ $ cd db_samples/Linaro/Sample-PAJ7620/
pi@raspberrypi:~/db_samples/Linaro/Sample-PAJ7620 $ ls
i2c-dev.h  main.cpp  PAJ7620.cpp  PAJ7620.h  README.md
pi@raspberrypi:~/db_samples/Linaro/Sample-PAJ7620 $
```

- 다운로드 후, 현재의 디렉터리를 /db_samples/Linaro/Sample-PAJ7620/으로 옮기면 "PAJ7620.cpp"와 "PAJ7620.h"파일 확인
- 이 파일들과 "i2c-dev.h"파일을 /Example 디렉터리로 이동
 - 터미널 창에 "mv PAJ7620.* i2c-dev.h ~/Example/" 입력

```
pi@raspberrypi:~/db_samples/Linaro/Sample-PAJ7620 $ mv PAJ7620.*
i2c-dev.h ~/Example/
```


Gesture 제어

- Gesture 센서 제어 예제(1)
 - 터미널 창에 “nano 20_GESTURE_01.cpp” 입력
 - “PAJ7620.cpp”파일은 C++언어이므로, 같이 빌드 하기 위해선 C++로 작성

```
pi@raspberrypi:~/Example $ nano 20_GESTURE_01.cpp
```

- Gesture 센서를 이용해 9가지 동작을 실행하는 예제

```
1. // File : 20_GESTURE_01.cpp
2. #include <stdio.h>
3. #include <unistd.h>
4. #include "PAJ7620.h"                // 현재 디렉터리에 있는 헤더파일 참조
5. #define I2C0      "/dev/i2c-1"      //I2C의 버스 설정
6. Paj7620 gesture_sensor(I2C0);       //gesture 센서의 객체 생성
7. //gesture센서의 초기화 함수 선언
8. int init_gesture()
9. {
10.    //초기화는 I2C 버스 연결 - 센서 wake up - 레지스터 읽기 - 주소 설정 순으로 진행
11.    //성공시 1, 실패시 0반환
12.    return gesture_sensor.initSensor();
13. }
```

Gesture 제어

- Gesture 센서 제어 예제(1)

```
14. //센서값을 읽어오는 함수 선언
15. int read_gesture()
16. {
17.     //readSensor는 gesture에 해당하는 레지스터 값을 읽어 그에 해당하는 값을 반환하는
        함수
18.     int value = gesture_sensor.readSensor();
19.     return value;
20. }

21. //센서의 값을 출력하는 함수 선언
22. void print_gesture(int value)
23. {
24.     char *gesture = NULL;

25.     switch(value)
26.     { //각 값에 해당하는 문자열 출력
27.         case GESTURE_RIGHT :
28.             printf("Right %d\n");
29.             break;

30.         case GESTURE_LEFT :
31.             printf("Left %d\n");
32.             break;
```

Gesture 제어

- Gesture 센서 제어 예제(1)

```
33.     case GESTURE_UP :  
34.         printf("Up %n");  
35.         break;  
  
36.     case GESTURE_DOWN :  
37.         printf("Down %n");  
38.         break;  
  
39.     case GESTURE_FORWARD :  
40.         printf("Forward %n");  
41.         break;  
  
42.     case GESTURE_BACKWARD :  
43.         printf("Backward %n");  
44.         break;  
  
45.     case GESTURE_CLOCKWISE :  
46.         printf("Clockwise %n");  
47.         break;  
  
48.     case GESTURE_COUNTER_CLOCKWISE:  
49.         printf("Counter Clockwise %n");  
50.         break;
```

Gesture 제어

- Gesture 센서 제어 예제(1)

```
51.     case GESTURE_WAVE:
52.         printf("Wave %d\n");
53.         break;

54.     default:
55.         break;
56. }
57. }

58. int main(void)
59. {
60.     printf("GESTURE SENSOR TEST\n");

61.     int result;

62.     //센서 초기화 실패시 -1을 반환
63.     if(result = init_gesture() == 0)
64.     {
65.         return -1;
66.     }

67.     while(1)
68.     {
```

Gesture 제어

● Gesture 센서 제어 예제(1)

```
69. //gesture센서의 값을 읽어옴
70. int value = read_gesture();

71. //gesture센서의 값을 출력
72. print_gesture(value);
73. //0.1초 대기
74. usleep( 100 * 1000 );
75. }
76. return 0;
77. }
```

- 작성 후 "ctrl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
- GCC 컴파일러를 사용하여 빌드 및 생성된 "20_GESTURE_01" 파일 실행

```
pi@raspberrypi:~/Example $ gcc -o 20_GESTURE_01 20_GESTURE_01.cpp PAJ7
620.cpp
pi@raspberrypi:~/Example $ ./20_GESTURE_01
```

- 결과
 - Gesture센서에서 약 5~15cm 떨어진 지점에서 손바닥으로 제스처를 취하면 동작을 인식하여 터미널 화면에 출력

```
Counter Clockwise
Right
Forward
Backward
Left
Right
Left
```

UART 기반의 시리얼 통신 제어

UART 기반의 시리얼 통신 제어

● 라즈베리파이의 UART

- 라즈베리파이에 사용되는 CPU는 BCM2837이고, 두 개의 UART 채널이 존재
- 두 개의 UART 채널 중에서 하나는 확장 핀으로 연결되어 있으며, 나머지 하나는 블루투스 모듈로 연결
- 확장 핀인 UART1 포트를 이용해 외부 장치와 통신

UART 채널	시리얼 포트	장치 이름	기능
UART0	serial1	ttyAMA0	블루투스
UART1	serial0	ttyS0	콘솔 혹은 시리얼

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

UART 기반의 시리얼 통신 제어

- 환경설정

- 터미널 창에 “sudo nano /boot/config.txt” 입력

- 확장 포트에 있는 UART기능을 사용하려면 “enable_uart=1” 문구 확인

```
pi@raspberrypi:~ $ sudo nano /boot/config.txt
```

```
# Enable audio (loads snd_bcm2835)
dtparam=audio=on
start_x=1
gpu_mem=128
enable_uart=1
```

- 터미널 창에 “sudo nano /boot/cmdline.txt” 입력

- UART 기능의 핀을 통신용으로 사용하기 위해 콘솔 사용 중지

```
pi@raspberrypi:~ $ sudo nano /boot/cmdline.txt
```

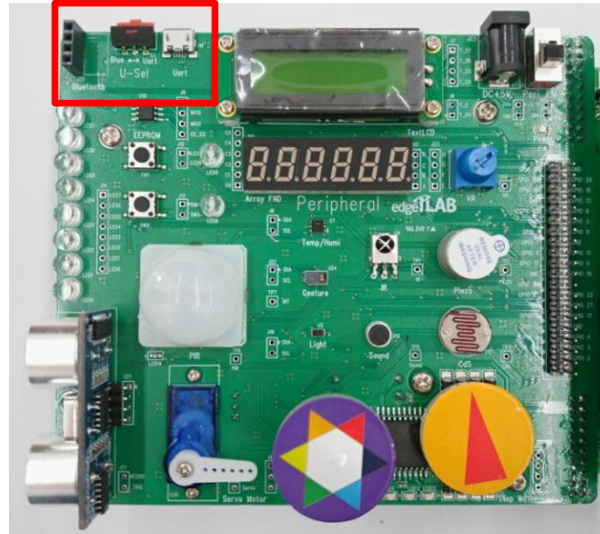
- “console=serial0, 115200” 부분 삭제

```
dwc_otg.lpm_enable=0 console=serial0,115200 console=tty1 root=/dev/mmcblk0p2 ro$
```

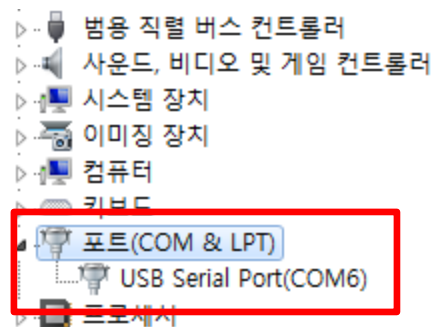
- 저장 및 재시작

UART 기반의 시리얼 통신 제어

- UART 핀 연결
 - Edge-Embedded의 UART 통신부와 Micro 5핀 케이블을 이용하여 연결



- 장치관리자에서 Serial Port번호 확인(COMx)

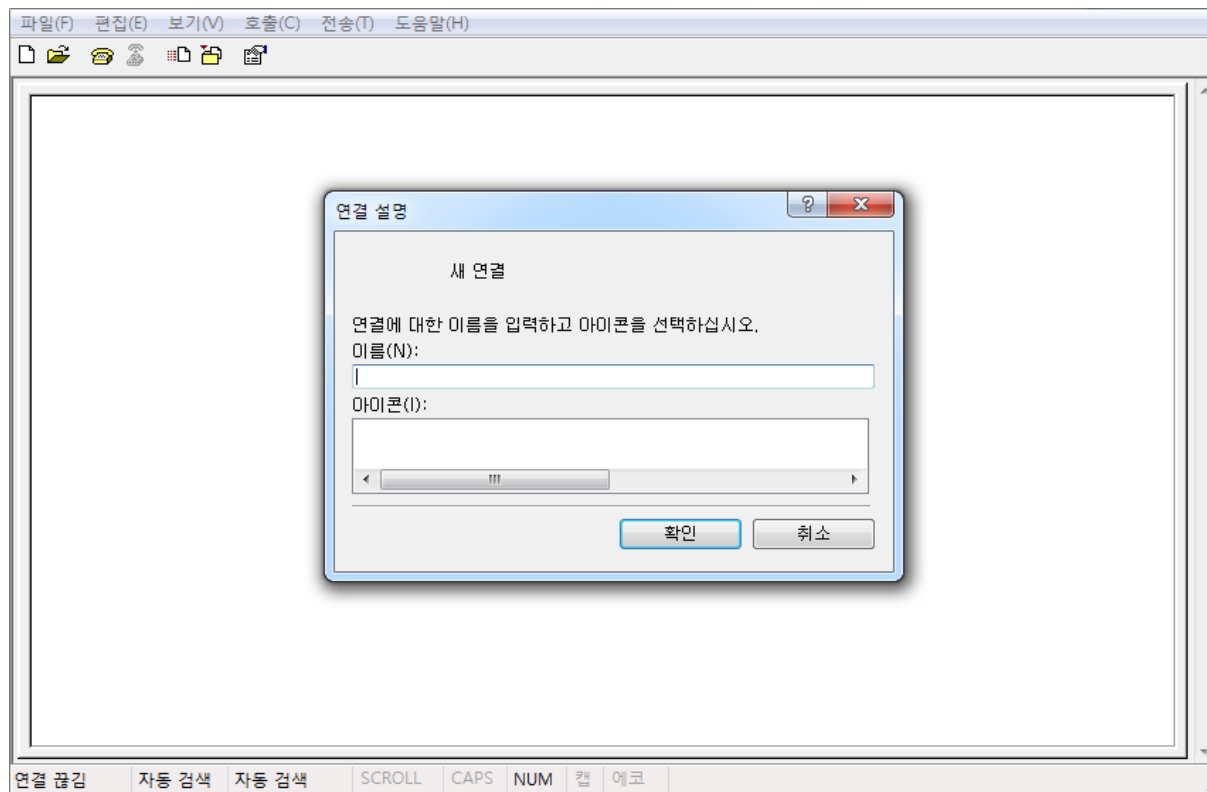


UART 기반의 시리얼 통신 제어

- 동작 테스트

- PC에서 하이퍼 터미널 실행

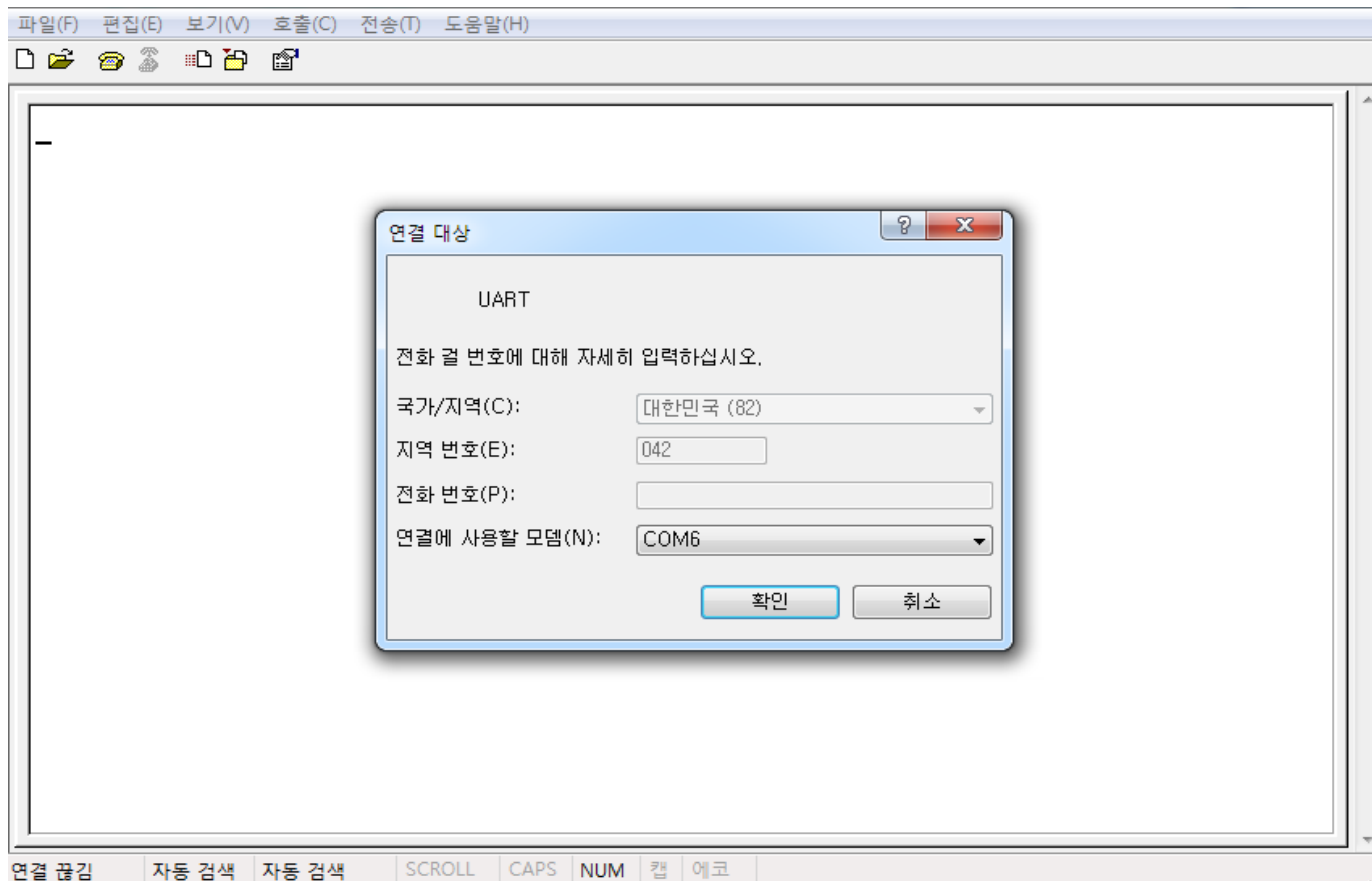
- 이름에 자신이 원하는 통신의 이름을 입력하고 확인
 - 윈도우 7 이상의 OS에서는 하이퍼 터미널이 내장되어 있지 않으므로, 구글 검색을 이용하여 다운받아 사용



UART 기반의 시리얼 통신 제어

- 동작 테스트

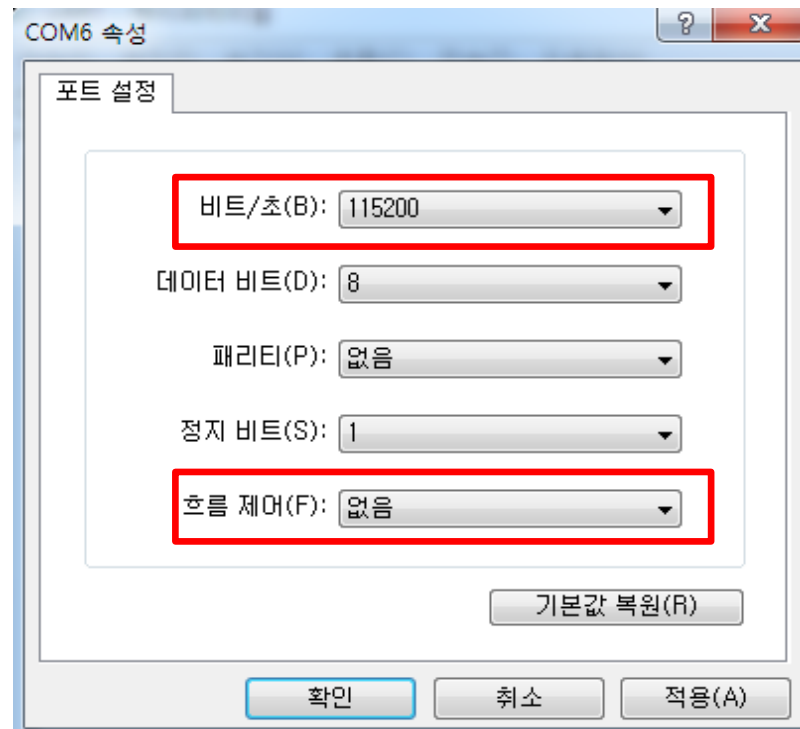
- 연결한 포트 대상을 설정하는 부분으로 PC와 Edge-Embedded를 연결한 USB 포트를 지정(장치관리자에서 포트 번호 확인)



UART 기반의 시리얼 통신 제어

- 동작 테스트

- UART 기반의 통신은 비동기 방식이므로 송신 측과 수신 측이 동일한 통신 속도 지정
 - 통신속도 : 115200bps
 - 흐름제어 : 없음



UART 기반의 시리얼 통신 제어

- 동작 테스트

- PC에서의 통신 설정이 완료하였으면, Edge-Embedded 에서도 동일하게 설정
 - 터미널 창에 "stty -F /dev/serial0 115200" 입력

```
pi@raspberrypi:~ $ stty -F /dev/serial0 115200
```

- 터미널 창에 "cat /dev/serial0" 입력
 - Serial0장치파일로 수신되는 문자열 확인

```
pi@raspberrypi:~ $ cat /dev/serial0
```

- 문자 출력 대기 상태

```
pi@raspberrypi:~ $ cat /dev/serial0
```

UART 기반의 시리얼 통신 제어

- 동작 테스트

- Edge-Embedded가 대기 상태에서, PC의 하이퍼 터미널로 돌아가 "THIS IS FROM PC" 입력



- Edge-Embedded의 serial0장치로 수신
 - 터미널 창에 수신된 문자열 출력

```
pi@raspberrypi:~ $ cat /dev/serial0  
THIS IS FROM PC.
```

- "ctrl + C"를 눌러 강제 종료

UART 기반의 시리얼 통신 제어

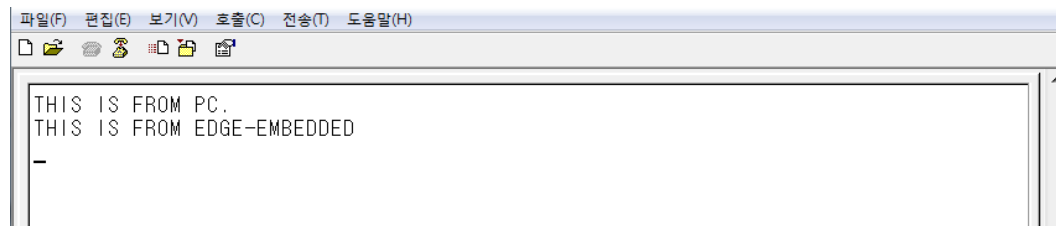
- 동작 테스트

- 터미널 창에 "echo "THIS IS FROM EDGE-EMBEDDED" > /dev/serial0" 입력
 - Echo 명령어를 이용하여 명시된 장치파일로 문자열 전송

```
pi@raspberrypi:~ $ echo "THIS IS FROM EDGE-EMBEDDED" > /dev/serial0
```

```
pi@raspberrypi:~ $ cat /dev/serial0
THIS IS FROM PC.
^C
pi@raspberrypi:~ $ echo "THIS IS FROM EDGE-EMBEDDED" > /dev/serial0
pi@raspberrypi:~ $
```

- PC의 하이퍼 터미널 창에 Edge-Embedded로부터 수신된 문자열 출력



UART 기반의 시리얼 통신 제어

- wiringPiSerial 라이브러리
 - Serial 라이브러리 제공 함수

초기화 함수	
int serialOpen(char *device, int baud); - 반환 값 : 성공시 'handle' 값, 실패시 '-1'	시리얼 장치파일을 baud변수에 명시된 속도로 설정
void serialClose(int fd);	열린 시리얼 장치를 닫음
제어 함수	
void serialFlush(int fd);	수신 버퍼를 비우는 작업을 수행
int serialGetchar(int fd); - 반환 값 : 10초 대기 후 수신문 자가 없으면 '-1' 반환	열린 시리얼 장치로 수신된 하나의 문자를 읽음
void serialPutchar(int fd, char c);	열린 시리얼 장치로 하나의 문자를 전송

UART 기반의 시리얼 통신 제어

- UART 기반의 시리얼 통신 제어 예제(1)
 - 터미널 창에 "nano 21_UART_01.c" 입력

```
pi@raspberrypi:~/Example $ nano 21_UART_01.c
```

- PC와 시리얼 통신을 하는 예제

```
1. // File : 21_UART_01.c
2. #include <stdio.h>
3. #include <wiringPi.h>      // wiringPi.h 만 선언해도 빌드에 문제 없다.
4. #define SER_PORT          "/dev/serial0"    // 시리얼 장치 설정
5. #define BAUD_RATE          115200          // 시리얼 통신 속도 설정
6. int main(void)
7. {
8.     int dev;                    // 시리얼 장치의 "handle"
9.     if((dev = serialOpen(SER_PORT, BAUD_RATE) < 0)
10.    {
11.        return -1;
12.    }
13.    printf("Port Open.\n");
14.    serialFlush(dev);           // 수신 버퍼 비우기
```

UART 기반의 시리얼 통신 제어

- UART 기반의 시리얼 통신 제어 예제(1)

```
15. while(1)
16. {
17.     char ch = serialGetchar(dev);           // dev로부터 하나의 문자를 읽어 저장
18.                                           // 수신 데이터가 있을 때까지 10초간 대기
19.     if(ch == 'x')                          // 수신된 문자가 'x'인 경우
20.     {
21.         break;                             // while 루프를 빠져나옴
22.     }
23.     else                                    // 수신된 문자가 'x'가 아닌 경우
24.     {
25.         fputc(ch, stderr);                  // 표준에러(stderr)에 문자를 입력
26.         // 버퍼링 없이 바로 터미널 화면에 출력
27.         serialPutchar(dev, ch);             // 수신된 문자를 다시 전송
28.     }
29. }
30. printf("Port Closed.\n");
31. serialClose(dev);

32. return 0;
33. }
```

UART 기반의 시리얼 통신 제어

- UART 기반의 시리얼 통신 제어 예제(1)
 - 작성 후 "ctrl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
 - GCC 컴파일러를 사용하여 빌드 및 생성된 "21_UART_01" 파일 실행

```
pi@raspberrypi:~/Example $ gcc -o 21_UART_01 21_CDS_01.c -lwiringPi
pi@raspberrypi:~/Example $ ./21_UART_01
```

- 결과
 - PC에서 하이퍼 터미널을 통해 "Hello World" 전송 및 수신



```
pi@raspberrypi:~/Example $ ./21_UART_01
Port Open
Hello World Port Closed.
pi@raspberrypi:~/Example $
```

- 'x'문자 수신 시 종료

블루투스 기반의 시리얼 통신 제어

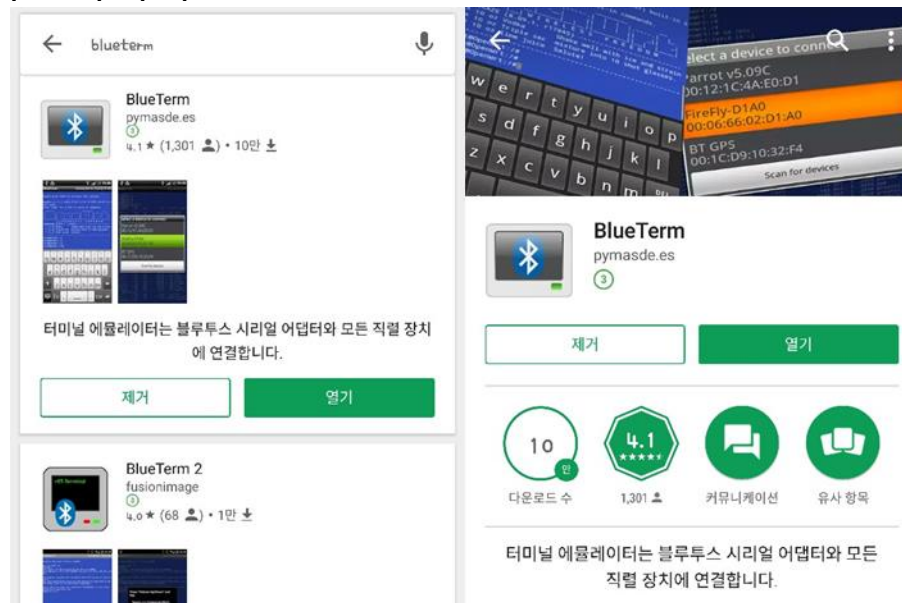
블루투스 기반의 시리얼 통신 제어

● 라즈베리파이의 블루투스

- 블루투스는 UART0채널에 연결되어 있고, 장치 이름은 serial1 또는 ttyAMA0

UART 채널	시리얼 포트	장치 이름	기능
UART0	serial1	ttyAMA0	블루투스
UART1	serial0	ttyS0	콘솔 혹은 시리얼

- Edge-Embedded와 블루투스 통신을 하기 위해 스마트폰(안드로이드)의 "Blueterm" 어플리케이션 활용



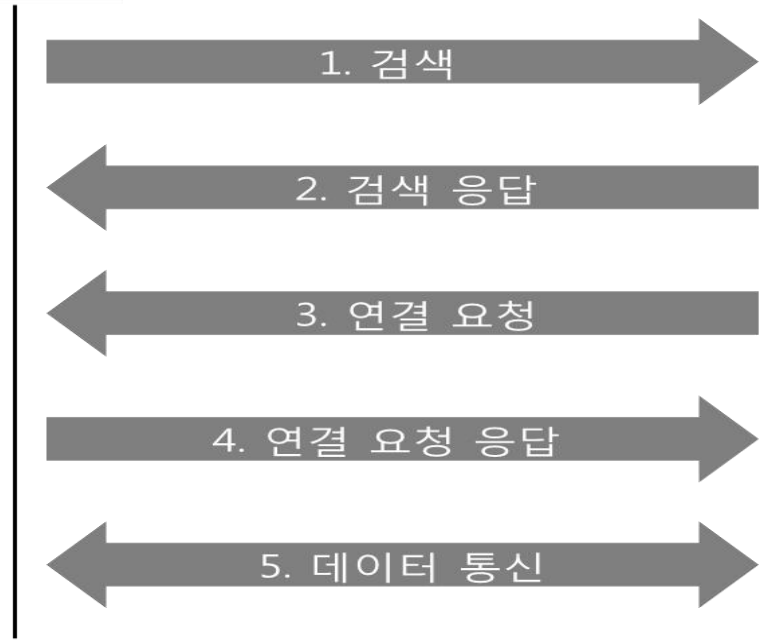
블루투스 기반의 시리얼 통신 제어

- 연결 과정
 - 서버 : Edge-Embedded
 - 클라이언트 : 스마트 폰

Client



Server



블루투스 기반의 시리얼 통신 제어

- 연결 과정

- 터미널 창에 “sudo rfcomm watch all &” 입력
 - 외부 장치가 블루투스 채널을 통해 연결 요청이 있는지 확인

```
pi@raspberrypi:~ $ sudo rfcomm watch all &
```

```
pi@raspberrypi:~ $ sudo rfcomm watch all &
[1] 1694
pi@raspberrypi:~ $ Waiting for connection on channel 1
pi@raspberrypi:~ $
```

- “Waiting for connection on channel 1” 이 출력되면 Enter키를 눌러 종료
 - 백그라운드에서 수행중
- 터미널 창에 “ps -ef | grep 'rfcomm'” 입력
 - 백그라운드에서 수행되는 프로그램 확인

```
pi@raspberrypi:~ $ ps -ef | grep 'rfcomm'
root      1649   1637   0 07:26 pts/0    00:00:00 sudo rfcomm watch all
root      1653   1649   0 07:26 pts/0    00:00:00 rfcomm watch all
root      1655     2   0 07:26 ?        00:00:00 [krfcommd]
pi        1665   1637   0 07:28 pts/0    00:00:00 grep --color=auto rfcomm
```

블루투스 기반의 시리얼 통신 제어

- 연결 과정
 - 페어링 되어있는 스마트 폰에 설치했던 "Blueterm" 어플리케이션을 실행하여 연결 요청하면 자동적으로 연결



블루투스 기반의 시리얼 통신 제어

- 동작 테스트

- 터미널 창에 "ls -al /dev/rfcomm0" 입력
 - 장치 파일이 생성되었는지 확인

```
pi@raspberrypi:~ $ ls -al /dev/rfcomm0
```

```
pi@raspberrypi:~ $ ls -al /dev/rfcomm0  
crw-rw---- 1 root dialout 216, 0 Oct 26 07:30 /dev/rfcomm0
```

- rfcomm이란

- 데이터 송수신을 위한 블루투스 스택이 내장된 장치
- 외부 블루투스 장치와 프로토콜에 기반한 정상적인 통신을 하기 위해서는 블루투스와 직접 연결된 "serial1"장치를 열면 안되고 그보다 상위의 "rfcomm"장치를 열어야 함
- 터미널 창에 "echo "THIS IS FROM EDGE-EMBEDDED" > /dev/rfcomm0" 입력
 - 해당 장치파일로 문자열 입력

```
pi@raspberrypi:~ $ echo "THIS IS FROM EDGE-EMBEDDED" > /dev/rfcomm0
```

```
BlueTerm                                     connected: raspberrypi  
THIS IS FROM EDGE-EMBEDDED
```

블루투스 기반의 시리얼 통신 제어

- 동작 테스트
 - 터미널 창에 "cat /dev/rfcomm0" 입력
 - 클라이언트에서 입력한 문자열 출력

```
pi@raspberrypi:~ $ cat /dev/rfcomm0
```

```
pi@raspberrypi:~ $ cat /dev/rfcomm0  
this is from smartphone
```

블루투스 기반의 시리얼 통신 제어

- 블루투스 기반의 시리얼 통신 제어 예제(1)
 - 터미널 창에 "nano 22_BLUETOOTH_01.c" 입력

```
pi@raspberrypi:~/Example $ nano 22_BLUETOOTH_01.c
```

- 스마트 폰과 블루투스 통신하는 예제

```
1. // File : 22_BLUETOOTH_01.c
2. #include <stdio.h>
3. #include <wiringPi.h>      // wiringPi.h 만 선언해도 빌드에 문제 없다.
4. #define BLUE_PORT         "/dev/rfcomm0"      // 시리얼 장치 설정
5. #define BAUD_RATE         115200             // 시리얼 통신 속도 설정
6. int main(void)
7. {
8.     int dev;                                // 블루투스 파일장치의 "handle"
9.     if((dev = serialOpen(BLUE_PORT, BAUD_RATE) < 0)
10.    {
11.        return -1;
12.    }
13.    printf("Port Open.\n");
14.    serialFlush(dev);                       // 수신 버퍼 비우기
```

블루투스 기반의 시리얼 통신 제어

● 블루투스 기반의 시리얼 통신 제어 예제(1)

```
15. while(1)
16. {
17.     char ch = serialGetchar(dev);           // dev로부터 하나의 문자를 읽어 저장
18.                                           // 수신 데이터가 있을 때까지 10초간 대기
19.     if(ch == 'x')                          // 수신된 문자가 'x'인 경우
20.     {
21.         break;                             // while 루프를 빠져나옴
22.     }
23.     else                                    // 수신된 문자가 'x'가 아닌 경우
24.     {
25.         fputc(ch, stderr);                  // 표준에러(stderr)에 문자를 입력
26.         // 버퍼링 없이 바로 터미널 화면에 출력
27.         serialPutchar(dev, ch);             // 수신된 문자를 다시 전송
28.     }
29. }
30. printf("Port Closed.\n");
31. serialClose(dev);

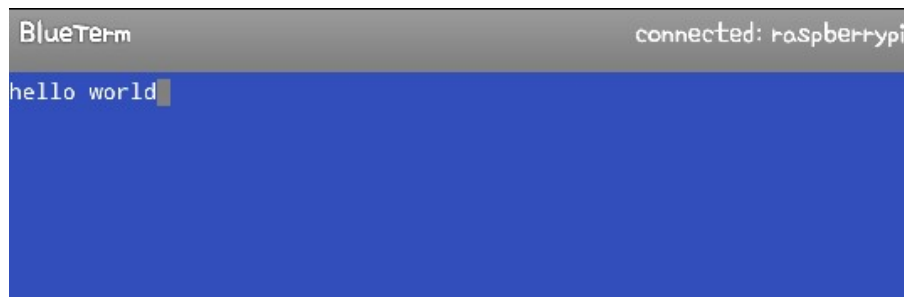
32. return 0;
33. }
```

블루투스 기반의 시리얼 통신 제어

- 블루투스 기반의 시리얼 통신 제어 예제(1)
 - 작성 후 "ctrl + o" 를 눌러 저장 및 "ctrl + X"를 눌러 종료
 - GCC 컴파일러를 사용하여 빌드 및 생성된 "22_BLUETOOTH_01" 파일 실행

```
pi@raspberrypi:~/Example $ gcc -o 22_BLUETOOTH_01 22_BLUETOOTH_01.c  
-lwiringPi  
pi@raspberrypi:~/Example $ ./22_BLUETOOTH_01
```

- 결과
 - 클라이언트에서 블루투스를 통해 "hello world" 전송 및 수신



- 클라이언트로부터 수신된 문자 출력 및 전송
- 'x' 문자 수신 시 종료

```
pi@raspberrypi:~/Example $ ./22_BLUETOOTH_01  
Port Open  
hello world Port Closed.
```