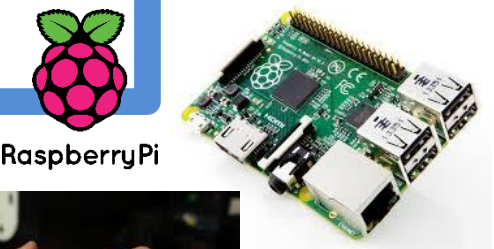
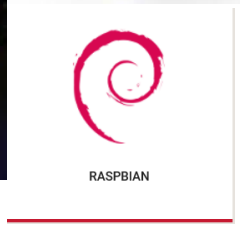
Sensor Programming
센서 프로그래밍

GPIO Piezo

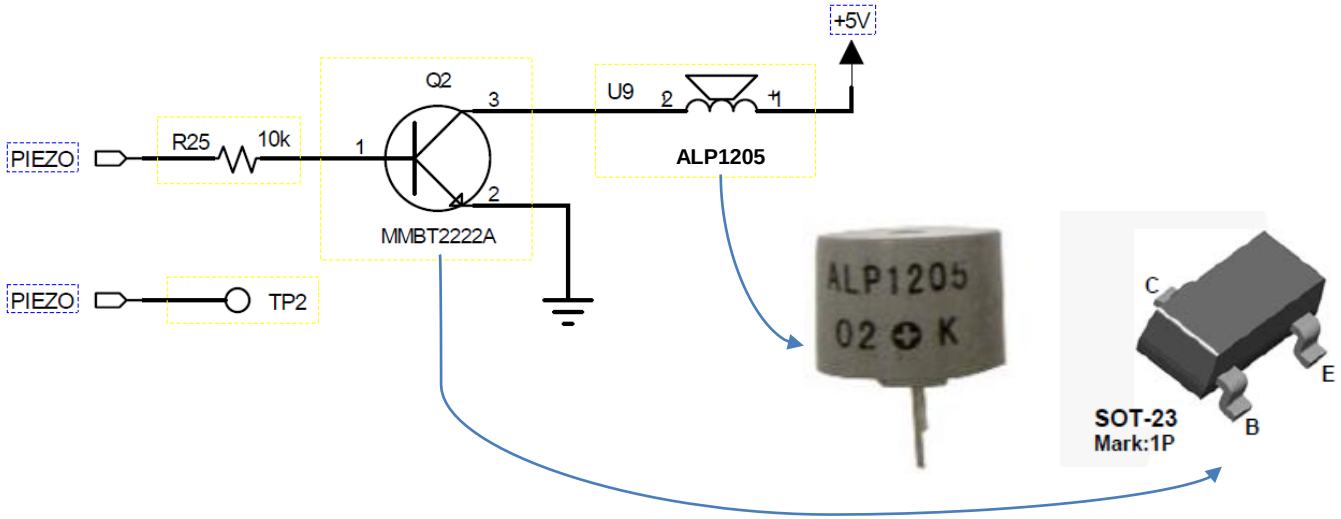




GPIO Piezo

- **Piezo 동작 제어**
  - Piezo는 Active type과 Passive type 으로 구분
    - Active type은 발진 회로 내장으로 정해진 Beep 음 발생
    - Passive type은 별도의 구동 회로 사용
      - 일정한 시간 간격으로 On/Off 할 때 음 발생
      - 발생되는 음은 음계별 표준 주파수 table 참조
      - 스위칭 주기(시간 간격) = on 시간 + off 시간

GPIO Piezo



- KPX-1205 : 마그네틱 부저
- MMBT2222A
  - IC 핀에서 흘릴 수 있는 전류는 보통 수십 mA
  - 부저같이 전류를 많이 필요로 하는 부품에 많은 양의 전류를 흘릴 수 있는 IC
  - 스팩은 최대 500mA까지 허용함.

GPIO Piezo

- **Piezo 동작 제어** (continued)
  - 발생되는 음은 음계별 표준 주파수 table

옥타브 / 음계	1	2	3	4	5	6	7	8
C(도)	32.7032	65.4064	130.8128	261.6256	523.2511	1046.502	2093.005	4186.009
C#	34.6478	69.2957	138.5913	277.1826	554.3653	1108.731	2217.461	4434.922
D(레)	36.7081	73.4162	146.8324	293.6648	587.3295	1174.659	2349.318	4698.636
D#	38.8909	77.7817	155.5635	311.1270	622.2540	1244.508	2489.016	4978.032
E(미)	41.2034	82.4069	164.8138	329.6276	659.2551	1318.510	2637.020	5274.041
F(파)	43.6535	87.3071	174.6141	349.2282	698.4565	1396.913	2793.826	5587.652
F#	46.2493	92.4986	184.9972	369.9944	739.9888	1479.978	2959.955	5919.911
G(솔)	48.9994	97.9989	195.9977	391.9954	783.9909	1567.982	3135.963	6271.927
G#	51.9130	103.8262	207.6523	415.3047	830.6094	1661.219	3322.438	6644.875
A(라)	55.0000	110.0000	220.0000	440.0000	880.0000	1760.000	3520.000	7040.000
A#	58.2705	116.5409	233.0819	466.1638	932.3275	1864.655	3729.310	7458.620
B(시)	61.7354	123.4708	246.9417	493.8833	987.7666	1975.533	3951.066	7902.133

GPIO Piezo

- **Piezo 소자 특징**
  - 실습 장치에 사용되는 Piezo 소자의 특징
    - Passive Piezo
    - 동작 주파수에 따라 음량 편차
      - 음계 별로 서로 다른 크기의 소리 발생
    - 고품질 음악 발생용으로는 부적합
    - 단순 멜로디 발생 용도
      - 동작 시작 / 종료 및 오류 상태 등의 상태 표시 용도

GPIO Piezo

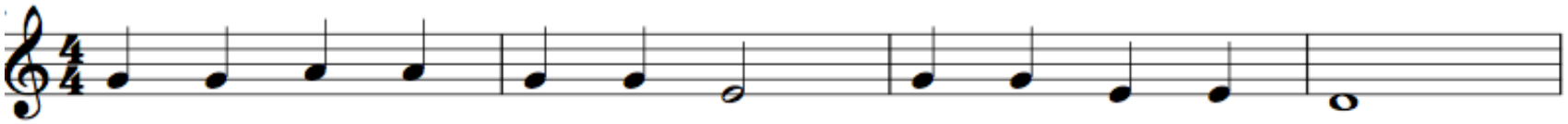
```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
gpio_pin=13
scale = [ 261, 294, 329, 349, 392, 440, 493, 523 ]
GPIO.setup(gpio_pin, GPIO.OUT)

try:
    p = GPIO.PWM(gpio_pin, 100)
    p.start(100)                # start the PWM on 100% duty cycle
    p.ChangeDutyCycle(90)  # change the duty cycle to 90%

    for i in range(8):
        print (i+1)
        p.ChangeFrequency(scale[i])
        time.sleep(1)

    p.stop()                    # stop the PWM output

finally:
    GPIO.cleanup()
```

GPIO Piezo

- scale = [261, 294, 329, 349, 392, 440, 493, 523]
  - 4옥타브 도 부터 5옥타브 도까지의 음계를 위한 list 선언
- p = GPIO.PWM(gpio_pin, 100)
  - gpio_pin의 주파수 100인 pwm 인스턴스를 생성
- p.start(100)
  - Duty cycle을 100%로 시작 설정
- p.ChangeDutyCycle(90)
  - Duty cycle을 90%로 변경
- for i in range(8):
  - 4옥타브 도부터 scale list의 음계를 반복 수행하기 위한 루프
- p.ChangeFrequency(scale[i])
  - 주파수를 scale의 음계로 변경
- p.stop()
  - pwm 정지

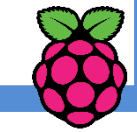
GPIO Piezo

- **Piezo 제어 실습**
  - 음계 발생
    - Piezo 연결된 BCM 13 핀의 ON/OFF
    - 일정한 시간 간격으로 High, Low 출력
  - 박자 제어
    - 발생 음의 지속 시간
    - 연속 음의 구분을 위한 일정 휴지 구간 설정
  - 아래의 악보를 실습 장치의 buzzer를 사용하여 음 발생 실습

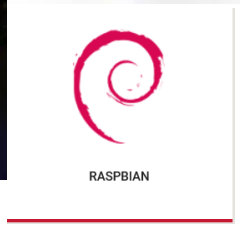


Sensor Programming
센서 프로그래밍

GPIO Character LCD



RaspberryPi

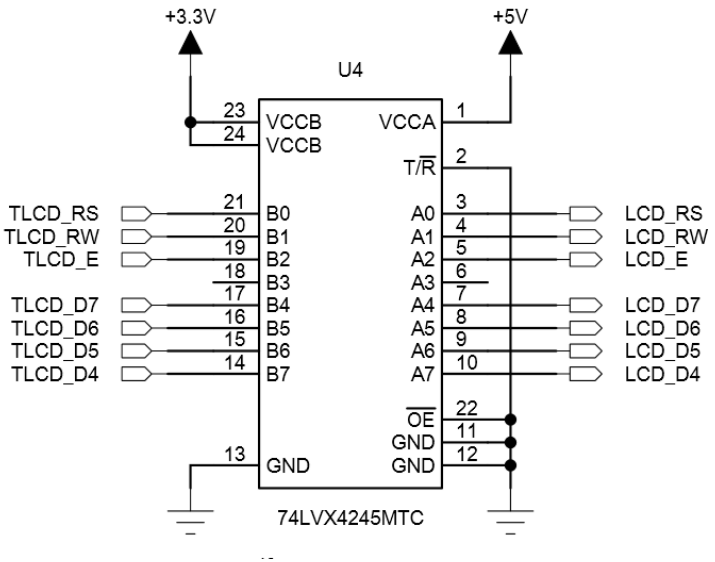


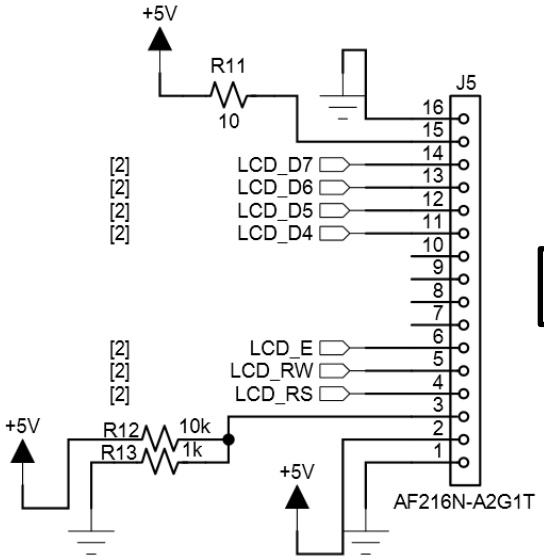
GPIO Character LCD

- Character LCD
  - 기억된 내용을 눈으로 직접 볼 수 있게 만든 디스플레이 장치.
  - 디스플레이 부와 제어 부가 하나로 통합된 모듈 형태로 판매.
  - 실습 보드는 16 x 2 문자 LCD 사용.
    - 4비트,8 비트 마이크로프로세서와 인터페이스 가능.
    - 5x8, 5x10 도트 디스플레이 가능.
    - 80x8비트의 디스플레이 램(최대 80글자).
    - 240 문자 폰트를 위한 문자 발생기 ROM.
    - 64x8비트 문자 발생기 RAM.
    - +5V 전원 사용.

GPIO Character LCD

CPU





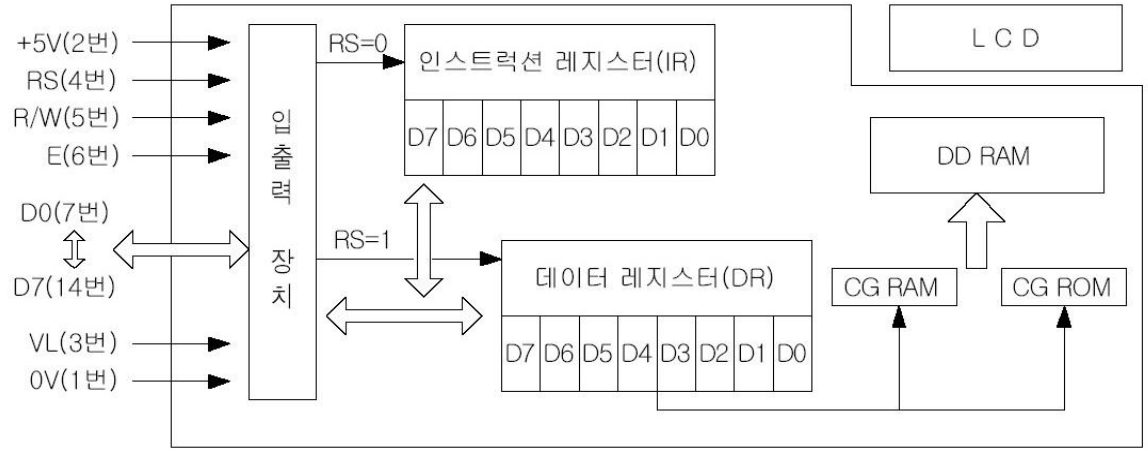
LCD 패널

74LVX4245MTC

- 3v(CPU, B port) 와 5v(LCD패널, A port) 버스 사이의 인터페이스를 위한 IC

GPIO Character LCD

- Character LCD 구조



- IR (instruction register) : LCD on/off, clear, function set 등을 설정하는 레지스터.
- DR (data register) : LCD 모듈에 문자를 나타내기 위한 데이터 값이 들어가는 레지스터.
- DD RAM : 최대 80 글자(80x8비트)를 출력할 수 있는 디스플레이 RAM
- CG RAM : 사용자 정의 64 글자(64x8비트) 발생기용 RAM
- CG ROM : 240 문자 폰트 ROM
- D0 ~ D7 : 명령(rs = 0) 또는 데이터(rs = 1) 입력 핀 (4비트일 경우 D4~D7)
- RS (resister select) : 명령 또는 데이터 선택 핀

GPIO Character LCD

pin	Signal Name	기 능	
1	VSS	전원 GND	
2	VDD	전원 +5VDC	
3	VEE	Contrast 제어 전압 레벨 (VDD-VEE = 13.5 ~ 0V)	
4	RS	Register Select (0 = instruction, 1 = data)	
5	R/W	Read/Write (0 = write, 1 = read)	
6	E	Enable Signal for read/write LCD	
7	DB0 (LSB)	4비트 데이터 버스 이용 시 사용 안함	8비트 데이터 버스 이용 시 모두 사용
8	DB1		
9	DB2		
10	DB3		
11	DB4	4비트 데이터 버스 이용 시 사용	
12	DB5		
13	DB6		
14	DB7 (MSB)		
15	A	+LED (backlight LED용 전원 +4.4V ~ +4.7V)	
16	K	-LED (backlight LED용 전원 GND)	

GPIO Character LCD

- RS와 RW 관계

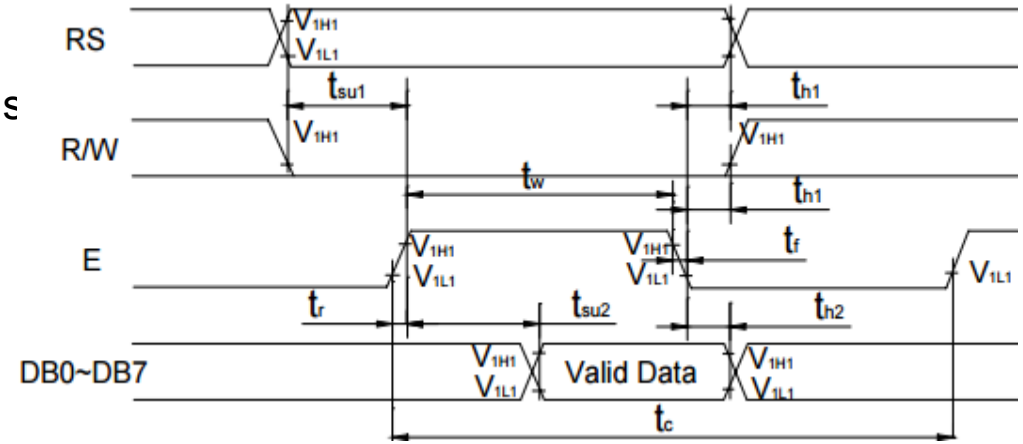
RS	R/W	LCD 동작 상태
0	0	IR 선택하여 제어 명령 쓰기 예) LCD 화면 클리어, 커서 시프트, LCD ON/OFF 등등.
0	1	D7로부터 비지 플래그 읽기 어드레스 카운터를 D0~D6으로부터 읽기
1	0	DR이 선택되어 데이터 값 쓰기
1	1	DR이 선택되어 데이터 값 읽기

- rs = 0 : 명령, rs = 1 : 데이터
- r/w = 0 : 쓰기, rw = 1 : 읽기
- 주소 버스는 없고 데이터 버스만 존재.
- 데이터 버스에 제어 명령과 데이터 정보가 함께 전달.
- rs로 데이터 버스에 실리는 정보가 제어 명령인지 데이터인지 구분 필요.

GPIO Character LCD

- 쓰기 모드 타이밍
  - rs를 high 또는 low로 설정
  - r/w를 low로 설정
  - 40ns 이후 e를 high로 설정
  - 80ns 이후 db0~7에 데이터 설정
  - 데이터 전송 완료
    - e를 low로 설정
  - 데이터 쓰기 구간은 최소 500ns

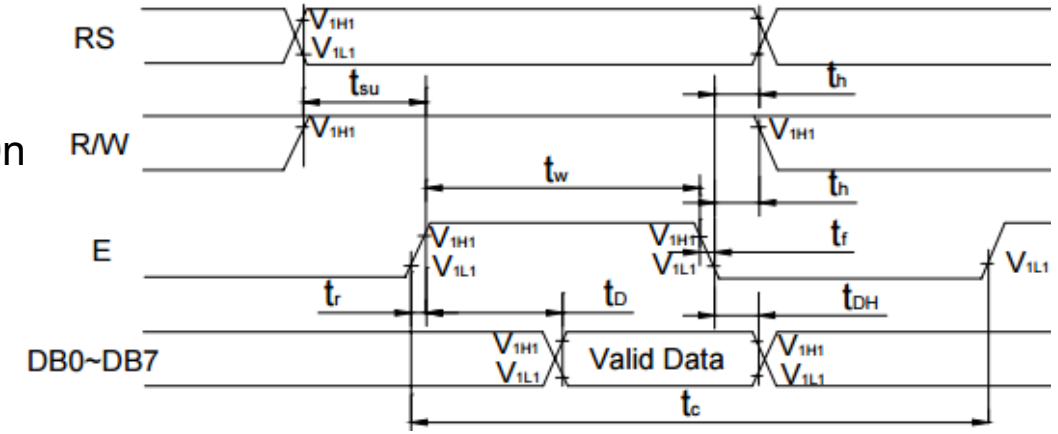
Mode	Symbol	Min.	Typ.	Max.	Unit
E Cycle Time	t_C	500	-	-	ns
E Rise / Fall Time	t_R, t_F	-	-	20	ns
E Pulse Width (High, Low)	t_W	230	-	-	ns
R/W and RS Setup Time	t_{su1}	40	-	-	ns
R/W and RS Hold Time	t_{H1}	10	-	-	ns
Data Setup Time	t_{su2}	80	-	-	ns
Data Hold Time	t_{H1}	10	-	-	ns



GPIO Character LCD

- 읽기 모드 타이밍
  - rs를 high 또는 low로 설정
  - r/w를 high로 설정
  - 40ns 이후 e를 high로 설정
  - 120ns 이후 db0~7에 데이터 읽기
  - 데이터 읽기 완료
    - e를 low로 설정
  - 데이터 읽기 구간은 최소 500n

Mode	Symbol	Min.	Typ.	Max.	Unit
E Cycle Time	t_C	500	-	-	ns
E Rise / Fall Time	t_R, t_F	-	-	20	ns
E Pulse Width (High, Low)	t_W	230	-	-	ns
R/W and RS Setup Time	t_{su1}	40	-	-	ns
R/W and RS Hold Time	t_{H1}	10	-	-	ns
Data Output Delay Time	t_{su2}	-	-	120	ns
Data Hold Time	t_{H1}	5	-	-	ns



GPIO Character LCD

- Character LCD 표시 제어 명령

기능	제어 신호		제어 명령							
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Clear Display	0	0	0	0	0	0	0	0	0	1
Return Home	0	0	0	0	0	0	0	0	1	0
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	0	0
Function Set	0	0	0	0	1	DL	N	F	0	0
Set CG RAM address	0	0	0	1	CG RAM address					
Set DD RAM address	0	0	1	DD RAM address						
Read busy flag and address	0	1	BF	Address Counter						
Data write to CG RAM or DD RAM	1	0	Write address							
Data read from CG RAM or DD RAM	1	1	Read address							

GPIO Character LCD

- Character LCD 표시 제어 명령 (continued)
  - Character LCD는 전송 받은 각 명령을 실행하기 위해 일정 시간 요구.
    - 다음 명령을 전송하기 전에 충분히 시간 지연.
    - BUSY 플래그를 읽어 선행된 명령이 완료되었는지 확인.
  - Clear Display
    - 모든 디스플레이 상태를 소거하고 커서를 Home 위치 이동.
  - Return Home
    - DD RAM의 내용은 변경하지 않고 커서만 Home으로 이동.
  - Entry Mode SET
    - 데이터를 읽거나 쓸 때 커서 위치 증가(I/D=1), 감소(I/D=0) 결정
    - 이때 화면을 이동 할지(S=1) 아닌지(S=0) 결정
  - Display ON/OFF Control
    - 화면 표시 ON/OFF(D), 커서 ON/OFF(C) 커서 깜박임(B) 설정

GPIO Character LCD

- Character LCD 표시 제어 명령 (continued)
  - Cursor or Display Shift
    - 화면(S/C=1) 또는 커서(S/C=0)를 오른쪽(R/L=1), 왼쪽(R/L=0)으로 이동.
  - Function SET
    - **데이터 길이**를 8비트(DL=1) 또는 **4비트(DL=0)로 지정.**
    - 화면 표시 행수를 2행(N=1) 또는 1행(N=0)으로 지정.
    - 문자 폰트를 5 x 10 도트(F=1) 또는 5 x 7도트(F=0)로 지정.
    - 전원 투입 후 초기화 코드에서 사용.
    - Character LCD reset에 약50ms가 소요되므로 충분히 기다린 후 명령 전송.
  - Set CG RAM Address
    - Character Generator RAM의 어드레스 지정.
    - 이후 송수신하는 데이터는 CG RAM 데이터

GPIO Character LCD

- Character LCD 표시 제어 명령 (continued)
  - Set DD RAM Address
    - Display Data RAM의 어드레스 지정.
    - 이후 송수신하는 데이터는 DD RAM 데이터.
  - Read Busy Flag & Address
    - Character LCD가 내부 동작 중임을 나타내는 Busy Flag(BF) 읽기.
    - 어드레스 카운터 내용 읽기.
    - Character LCD가 각 제어 코드를 실행하는 데는 일정한 시간이 필요.
    - 프로세서가 BF를 읽어 ‘1’이면 기다리고 ‘0’이면 다음 제어 코드 전송

GPIO Character LCD

- Character LCD 모듈 초기화
  - 전원 인가
  - Character LCD가 reset되려면 약 30ms 이상 소요되므로 이 시간 동안 대기.
  - Function set 명령(001X_XX00) 전송 후 39us 이상 대기.
  - Display ON/OFF control 명령(0000_1XXX) 전송 후 39us 이상 대기.
  - Display Clear 명령(0000_0001) 전송 후 1.53ms 이상 대기.
  - Entry mode set 명령(0000_01XX) 전송.
  - 필요에 따라 DD RAM address 전송 후 문자 데이터 연속 전송.

GPIO Character LCD

```
#!/usr/bin/python

import RPi.GPIO as GPIO
import time

# Define GPIO to LCD mapping
LCD_RS = 23
LCD_RW = 24
LCD_E  = 26
LCD_D4 = 17
LCD_D5 = 18
LCD_D6 = 27
LCD_D7 = 22
```

GPIO Character LCD

```
# Define some device constants
LCD_WIDTH = 16    # Maximum characters per line
LCD_CHR = True
LCD_CMD = False

LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line

# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005
```

GPIO Character LCD

```
def main():
  GPIO.setwarnings(False)
  GPIO.setmode(GPIO.BCM)       # Use BCM GPIO numbers
  GPIO.setup(LCD_E, GPIO.OUT)  # E
  GPIO.setup(LCD_RS, GPIO.OUT) # RS
  GPIO.setup(LCD_D4, GPIO.OUT) # DB4
  GPIO.setup(LCD_D5, GPIO.OUT) # DB5
  GPIO.setup(LCD_D6, GPIO.OUT) # DB6
  GPIO.setup(LCD_D7, GPIO.OUT) # DB7

  # Initialise display
  lcd_init()
```

GPIO Character LCD

```
# def main():  (continued)
  while True:

    # Send some test
    lcd_string("Rasbperry Pi",LCD_LINE_1)
    lcd_string("16x2 LCD Test",LCD_LINE_2)
    time.sleep(3) # 3 second delay

    # Send some text
    lcd_string("1234567890123456",LCD_LINE_1)
    lcd_string("abcdefghijklmnop",LCD_LINE_2)
    time.sleep(3) # 3 second delay
```

GPIO Character LCD

```
def lcd_init():
  # Initialise display
  lcd_byte(0x33,LCD_CMD) # 110011 Initialise
  lcd_byte(0x32,LCD_CMD) # 110010 Initialise
  lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction
  lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off
  lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  time.sleep(E_DELAY)
```

GPIO Character LCD

```
def lcd_byte(bits, mode):
  # Send byte to data pins
  # bits = data
  # mode = True  for character
  #             False for command

  GPIO.output(LCD_RS, mode) # RS
```

GPIO Character LCD

```
# def lcd_byte (bits, mode): (continued)
 # High bits
  GPIO.output(LCD_D4, False)
  GPIO.output(LCD_D5, False)
  GPIO.output(LCD_D6, False)
  GPIO.output(LCD_D7, False)
  if bits&0x10==0x10:
    GPIO.output(LCD_D4, True)
  if bits&0x20==0x20:
    GPIO.output(LCD_D5, True)
  if bits&0x40==0x40:
    GPIO.output(LCD_D6, True)
  if bits&0x80==0x80:
    GPIO.output(LCD_D7, True)
  # Toggle 'Enable' pin
  lcd_toggle_enable()
```

GPIO Character LCD

```
# def lcd_byte (bits, mode): (continued)
  # Low bits
  GPIO.output(LCD_D4, False)
  GPIO.output(LCD_D5, False)
  GPIO.output(LCD_D6, False)
  GPIO.output(LCD_D7, False)
  if bits&0x01==0x01:
    GPIO.output(LCD_D4, True)
  if bits&0x02==0x02:
    GPIO.output(LCD_D5, True)
  if bits&0x04==0x04:
    GPIO.output(LCD_D6, True)
  if bits&0x08==0x08:
    GPIO.output(LCD_D7, True)
  # Toggle 'Enable' pin
  lcd_toggle_enable()
```

GPIO Character LCD

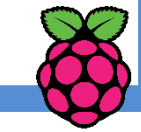
```
def lcd_toggle_enable():
  # Toggle enable
  time.sleep(E_DELAY)
  GPIO.output(LCD_E, True)
  time.sleep(E_PULSE)
  GPIO.output(LCD_E, False)
  time.sleep(E_DELAY)


def lcd_string(message,line):
  # Send string to display
  message = message.ljust(LCD_WIDTH," ")
  lcd_byte(line, LCD_CMD)
  for i in range(LCD_WIDTH):
    lcd_byte(ord(message[i]),LCD_CHR)
```

GPIO Character LCD

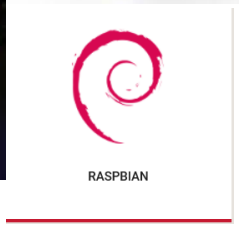
```
if __name__ == '__main__':
  try:
    main()
  except KeyboardInterrupt:
    pass
  finally:
    lcd_byte(0x01, LCD_CMD)
    lcd_string("Goodbye!",LCD_LINE_1)
    GPIO.cleanup()
```

Sensor Programming
센서 프로그래밍

Ultrasonic Sensor



RaspberryPi



Ultrasonic Sensor

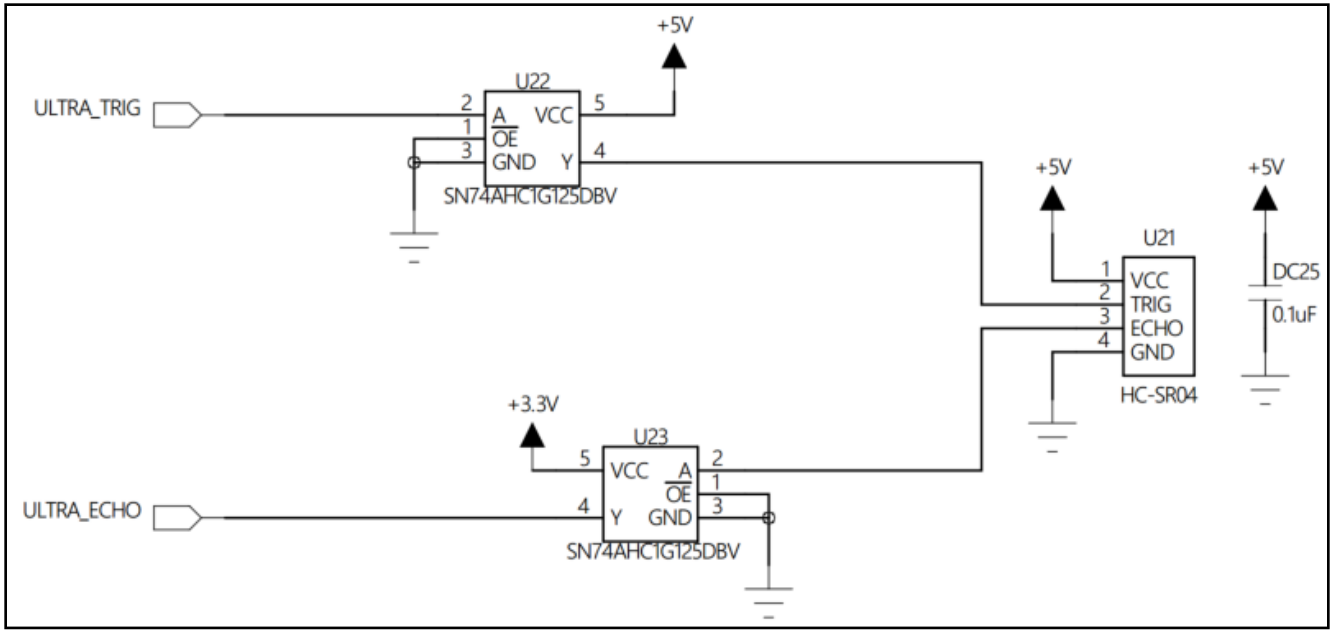
- **Ultrasonic Sensor : 초음파 센서**

  초음파 센서는 박쥐처럼 초음파를 공기 중에 방사한 후 물체에 반사되어 돌아오는 시간을 계산해 거리를 파악한다. Peri0 모듈에 포함된 초음파 센서(HC-SR04)는 송수신기가 결합된 모듈 타입으로 햇빛이나 검은색 물질에 영향을 덜 받으며 2cm ~ 400cm 범위의 거리를 측정할 수 있다.

Ultrasonic Sensor

- 회로 구성

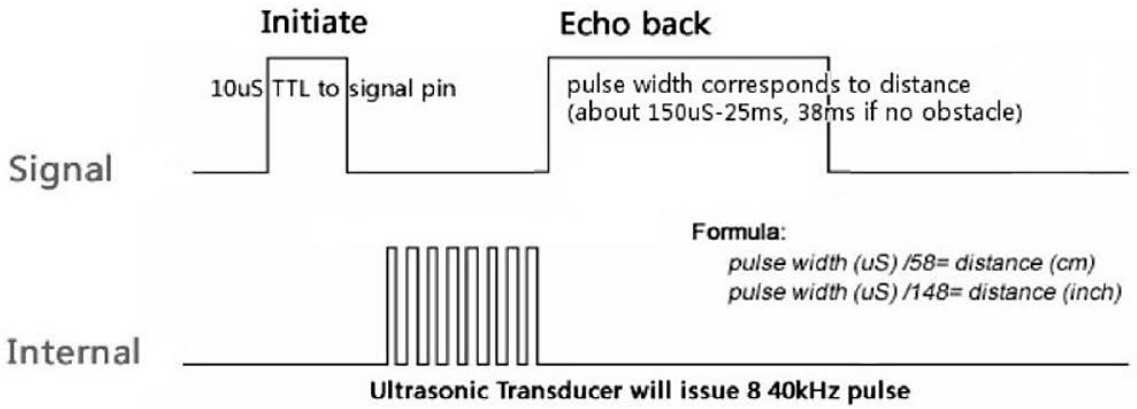
  동작 전압은 5V이므로 송신기와 연결되는 ULTRA_TRIG는 3.3V GPIO 출력을 5V로 변환하기 위해 레벨 버퍼 U22를 거치며 수신기와 연결되는 ULTRA_ECHO는 5V 출력을 GPIO의 유효 입력 전압인 3.3V로 변경하기 위해 레벨 버퍼 U23을 거쳐 들어온다.



Ultrasonic Sensor

- TRIG/ECHO 신호

초음파 센서의 TRIG 핀에 최소한 10마이크로 초 동안 HIGH 신호를 전달하면 송신기는 40kHz 초음파 8개를 방사한 후 반사되어 돌아오는 것을 기다린다. 수신기에서 반사되어 돌아온 초음파를 감지하면 ECHO를 HIGH로 설정한 다음 거리에 비례하는 마이크로 초(ms) 동안 유지(Time)한다. 따라서 ECHO 핀으로 수신되는 약 150us ~ 25ms 범위의 HIGH 레벨 펄스 폭은 거리에 해당하며, 장애물이 없는 경우 38ms 동안 유지된다.



Ultrasonic Sensor

- 거리 계산
  초음파의 속도는 약 340m/s (1초에 340m 이동)

  속도 = 거리 / 시간
  거리(distance) = 속도 * 시간

  공식에 따라 속도는 340m/s, 거리는 distance ,
  시간은 duration /2 (왕복거리이므로)

  distance = 340 * 100 * duration / 2 = duration* 17000

  단위가 meter였으므로 cm로 나타내기위해 100을 곱하면 17000
  round를 이용하여 소숫점 아래 3째자리에서 반올림

Ultrasonic Sensor

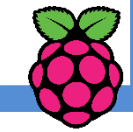
```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

trig = 0
echo = 1

GPIO.setup(trig, GPIO.OUT)
GPIO.setup(echo, GPIO.IN)                    # Peri v 2.1
#GPIO.setup(echo, GPIO.IN,GPIO.PUD_UP)    # Peri v 2.0

try :

  while True :

    GPIO.output(trig, False)
    time.sleep(0.5)

    GPIO.output(trig, True)
    time.sleep(0.00001)
    GPIO.output(trig, False)
```

Ultrasonic Sensor

```
    while GPIO.input(echo) == False :   # Peri v 2.1
    #while GPIO.input(echo) == True :  # Peri v 2.0
      pulse_start = time.time()

    while GPIO.input(echo) == True :    # Peri v 2.1
    #while GPIO.input(echo) == False : # Peri v 2.0
      pulse_end = time.time()

    pulse_duration = pulse_end - pulse_start
    distance = pulse_duration * 17000
    distance = round(distance, 2)

    print ("Distance : ", distance, "cm")

except :
  GPIO.cleanup()
```

Sensor Programming
센서 프로그래밍

IR(Infrared) Receiver



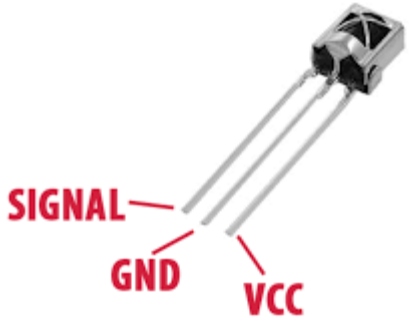
RaspberryPi

RASPBIAN

IR Receiver

- IR (infrared) Receiver

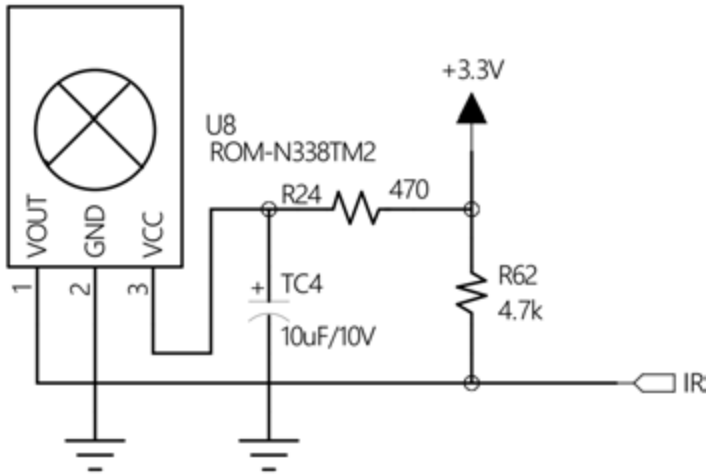
적외선 Infrared Rays은 파장이 약 780nm ~ 1mm 범위의 전자기파를 일컫는 말로 가시광선 영역에서 파장이 가장 긴 빨간색 밖에 위치한다. 적외선은 다시 파장이 길어지는 순서로 근적외선, 중적외선, 원적외선으로 나뉘며, 이 중 대부분의 영역은 열선이라 불리는 원적외선(4㎛~1mm)이 차지한다. 780nm ~ 2000nm 범위의 근적외선은 가시광선보다 파장이 길어서 회절이 잘 일어나고 사람의 눈으로 인지할 수 없으며 원적외선과 달리 열을 발생시키지 않으므로 가전제품의 제어에 주로 사용한다.



IR Receiver

- 회로 구성

Peri0의 IR Receiver는 적외선을 감지하는 포토다이오드Phototransistor와 대역 통과 필터Band Pass Filter가 통합된 모듈 타입으로 수신되는 적외선 신호 중 특정 주파수의 적외선 신호만 분리해 출력 핀(VOUT)으로 내보낸다. 입력 핀(IR)이 풀 업 상태이므로 기본값은 HIGH이고 적외선 신호가 감지될 때마다 Low(1) 신호가 IR로 전달된다.



IR Receiver

- LIRC(Linux Infrared Remote Control)

/etc/lirc/ 디렉터리에 위치하는 iCORE-SDP의 기존 LEG 리모컨 용 키 맵 파일 lircd.conf를 백업한 후 새로 생성한 키 맵 파일을 복사한다.

tea@planx:~ $ sudo mv /etc/lirc/lircd.conf /etc/lirc/lge_lircd.conf

tea@planx:~ $ sudo cp /etc/lirc/car_lird.conf /etc/lirc/lircd.conf

lircrc는 lircd.conf에 정의된 키 이름을 특정 문자열로 변환해 응용프로그램에 전달할 때 참조한다. /dev/lirc/ 디렉터리에 위치하는 iCORE-SDP의 기존 LEG 리모컨 용 lircrc를 백업한 후 새로 작성한다.

tea@planx:~ $ sudo mv /etc/lirc/lircrc /etc/lirc/lge_lircrc

tea@planx:~ $ sudo mv /etc/lirc/car_lircrc /etc/lirc/lircrc

<… 각 항목은 begin ~ end로 묶이고 주요 요소는 button, prog, config임 …>

IR Receiver

- LIRC(Linux Infrared Remote Control)

  LIRC 서비스를 재 시작한 후 새로 만든 키 맵을 테스트한다.

  tea@planx:~ $ sudo service lirc restart

  tea@planx:~ $ irw

<수신기 키를 누를 때마다 코드와 연속 횟수, 이름, 키 맵 이름 순으로 출력>

0000000000ffa25d 00 KEY_CHANNELDOWN custom.conf
0000000000ffa25d 01 KEY_CHANNELDOWN custom.conf
0000000000ffa25d 02 KEY_CHANNELDOWN custom.conf
0000000000ffe21d 00 KEY_CHANNELUP custom.conf
0000000000ffe21d 01 KEY_CHANNELUP custom.conf

…

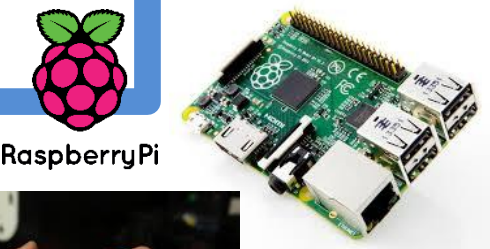
<Ctrl> + <C>

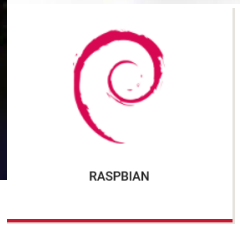
IR Receiver

```
import lirc
import time

sockid = lirc.init("Peri0","/etc/lirc/lircrc",blocking=False)

while True:
            try:
        button = lirc.nextcode()

        if len(button) == 0:
                    continue

        print(button[0])


           except KeyboardInterrupt:
                    lirc.deinit()
                    break
```

Sensor Programming
센서 프로그래밍

PIR(Passive Infrared) Sensor





PIR Sensor

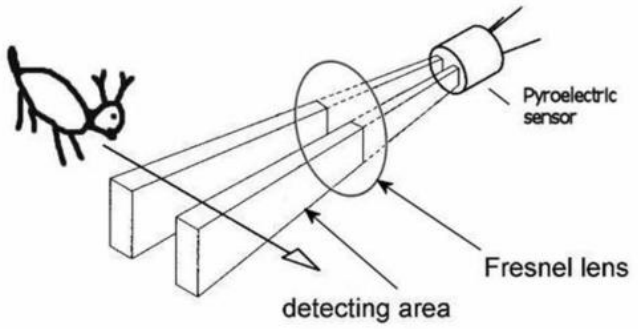
- PIR (Passive infrared) Sensor

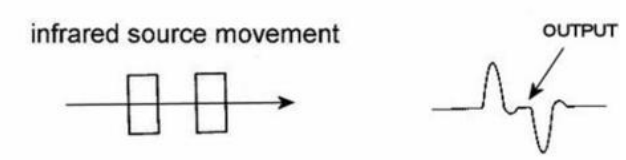
  절대 온도를 초과하는 모든 물체는 사람 눈에는 보이지 않는 적외선 파장과 함께 열에너지를 방출하는데 PIR는 물체에서 방출되거나 반사되는 적외선 파장의 움직임에 반응하는 센서이다.

PIR Sensor

- 움직임 감지

  PIR Sensor는 프레넬 렌즈 Fresnel Lens 와 적외선 검출기(U20), 판별 회로부로 구성되는데 프레넬 렌즈가 적외선만 모아 적외선 검출기로 전달하면 적외선 검출기는 수평 면을 기준으로 첫 번째 감지 점과 두 번째 감지 점을 구분해 판별 회로로 전달한다. 판별 회로는 두 입력 값의 변화를 비교한 후 디지털 결과를 출력한다.



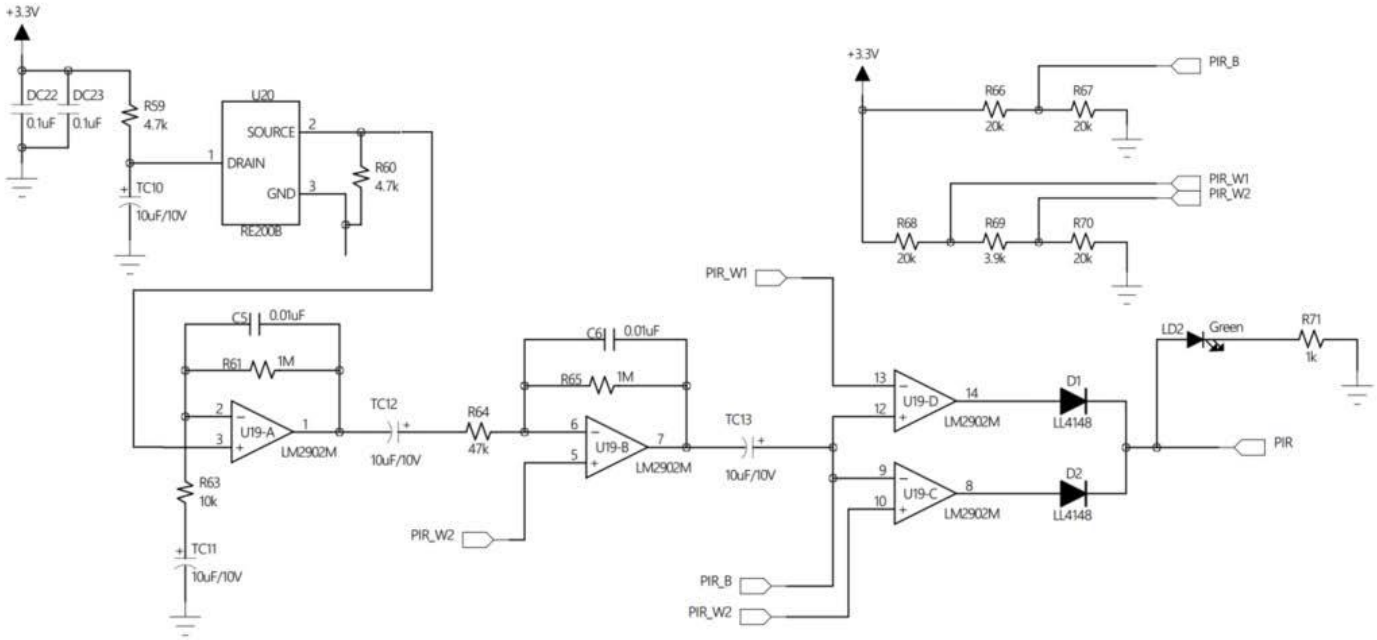


PIR Sensor

- 회로 구성

Peri0의 적외선 검출기 소스는 다운Pull Down 상태이므로 적외선의 움직임을 감지하면 미세 전류를 오른쪽 증폭기(U19-A, U18-B)로 보낸다. 첫 번째 증폭기는 입력 신호와 피드백을 통해 노이즈를 제거한 증폭 신호 만들어 두 번째 증폭기로 보내고, 두 번째 증폭기는 입력 신호와 기준 전압(PIR_W2)을 통해 노이즈를 제거한 증폭 신호를 만들어 비교기로(U19-C, U19-D)로 보낸다

비교기는 2개의 기준 전압(PIR_W1, PIR_W2)과 입력 신호의 비교 결과를 다이오드(D1, D2)로 보내면 다이오드를 통해 움직임이 감지될 때마다 최종 결과인 High 값이 출력 핀(PIR)으로 보내진다. 출력 핀은 풀 다운 상태이므로 움직임이 감지되지 않으면 LOW, 감지되면 HIGH가 된다.



PIR Sensor

```
import RPi.GPIO as GPIO
import time

pir = 24
GPIO.setmode(GPIO.BCM)
GPIO.setup(pir, GPIO.IN)

def loop():
    cnt = 0
    while True:
        if (GPIO.input(pir) == True):
            print ('detected %d' %cnt)
            cnt+=1
        time.sleep(0.1)

try:
    loop()

except KeyboardInterrupt:
                pass
finally:
    GPIO.cleanup()
```