Outline

日期	(週數) 主題
2/22	1. 嵌入式應用介紹
3/8	3. 樹莓派介紹與設定
3/15	**4. 樹莓派應用(倒車雷達)**
3/22	5. 樹莓派應用(人體活動偵測)
3/29	6. 網路攝影機 IP cam
4/12	8. 網路攝影機 + 影像辨識
4/19	9. 網路攝影機 + 機器學習影像辨識
4/26	10. Midterm (Project分組)
5/3	11. 語音助理. Google assistant
5/10	12. 網路應用、推播廣告(beacon)
5/17	13. 其他嵌入式系統
5/24	14. 其他嵌入式系統
5/31	15. Final Project – Proposal
6/14	17. Final Project prepare, Q&A, 補demo
6/21,28	18,19. Final Project demonstration (陽明, 交大) TBD

Last week

- 1. 根據安裝步驟, 使用TTL控制樹莓派
- 2. 建立VNC遠端桌面, 需開啟 “direct capture mode”
- 3. 練習wget、simpleHTTPserver與winscp傳輸檔案

嵌入式系統設計概論與實作

曾煜棋、吳昆儒、張凌燕

National Chiao Tung University

Outline

- 嵌入式應用: 倒車雷達
  - 1. 警示訊息 (LED燈)
  - 2. 超音波測距
  - 3. 溫度感測
- Raspberry PI
  - GPIO introduction
  - Python

HC-SR04

Ultrasonic

DHT-11

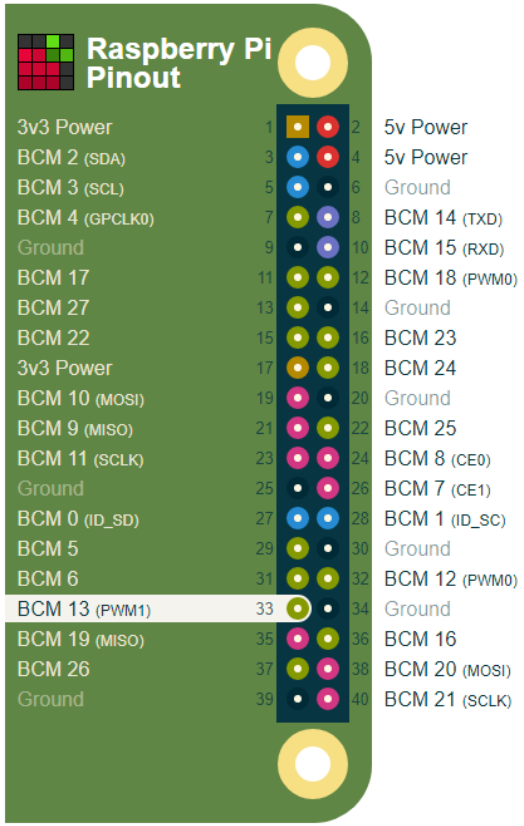
Thermometer + Hygrometer

Fig source: https://www.instructables.com/id/How-to-Create-a-Parking-Assist-System/

Parking Assist System

Outline

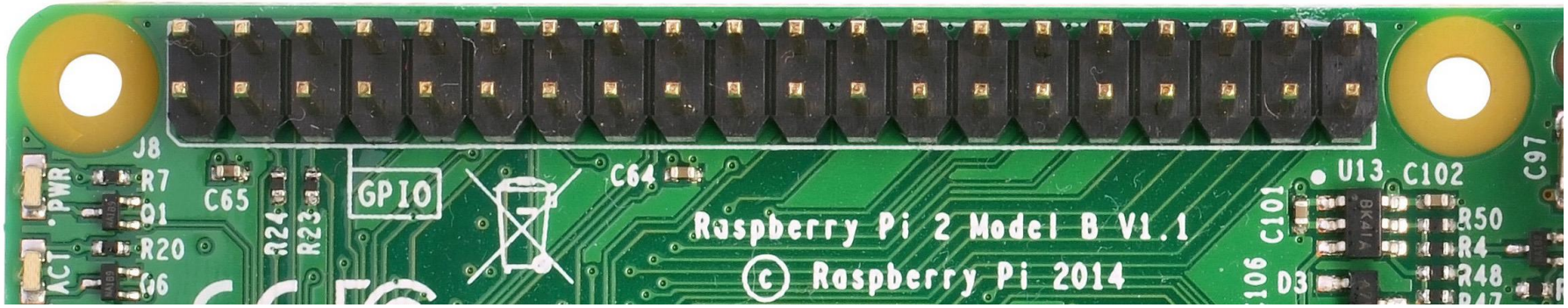
- Connect sensors to Raspberry PI
  - GPIO pins
  - Sensors
  - Write code (Python)
  - **Integrate them**



https://pinout.xyz/

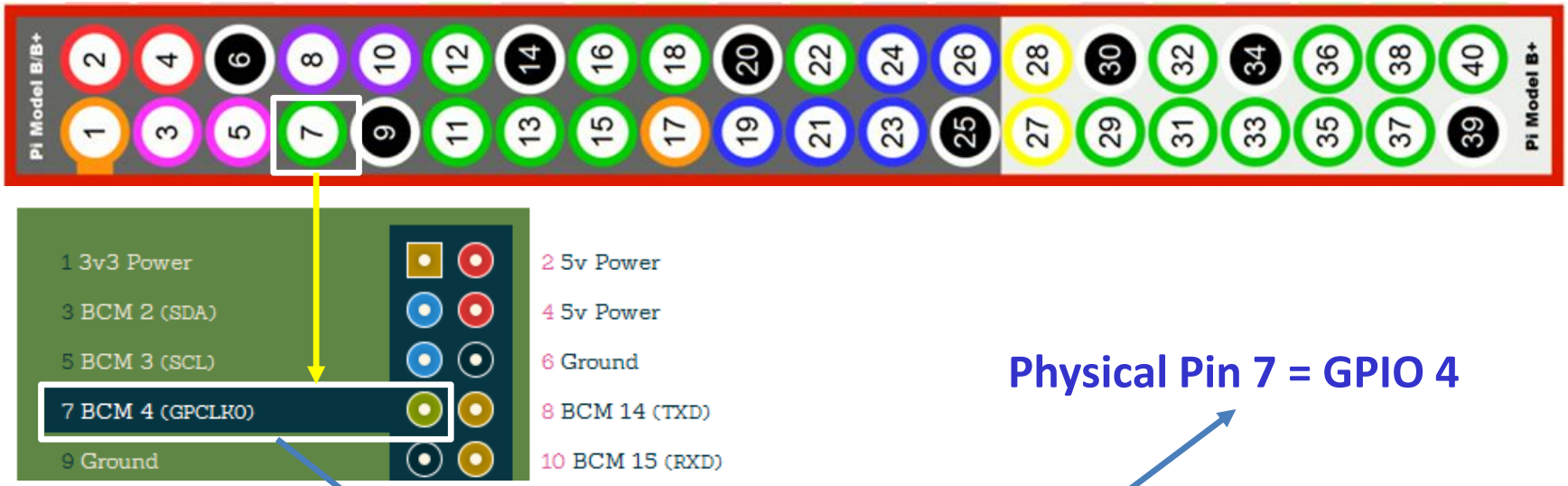
0. GPIO introduction

- General-purpose input/output (GPIO)
  - You can set PIN as **Input** or **Output** or **both**(Input and output)
    - Input: write a value on PIN
    - Output: Read the value on PIN



0. GPIO introduction

- Pin number != GPIO number
  - Physical numbering vs. GPIO numbering

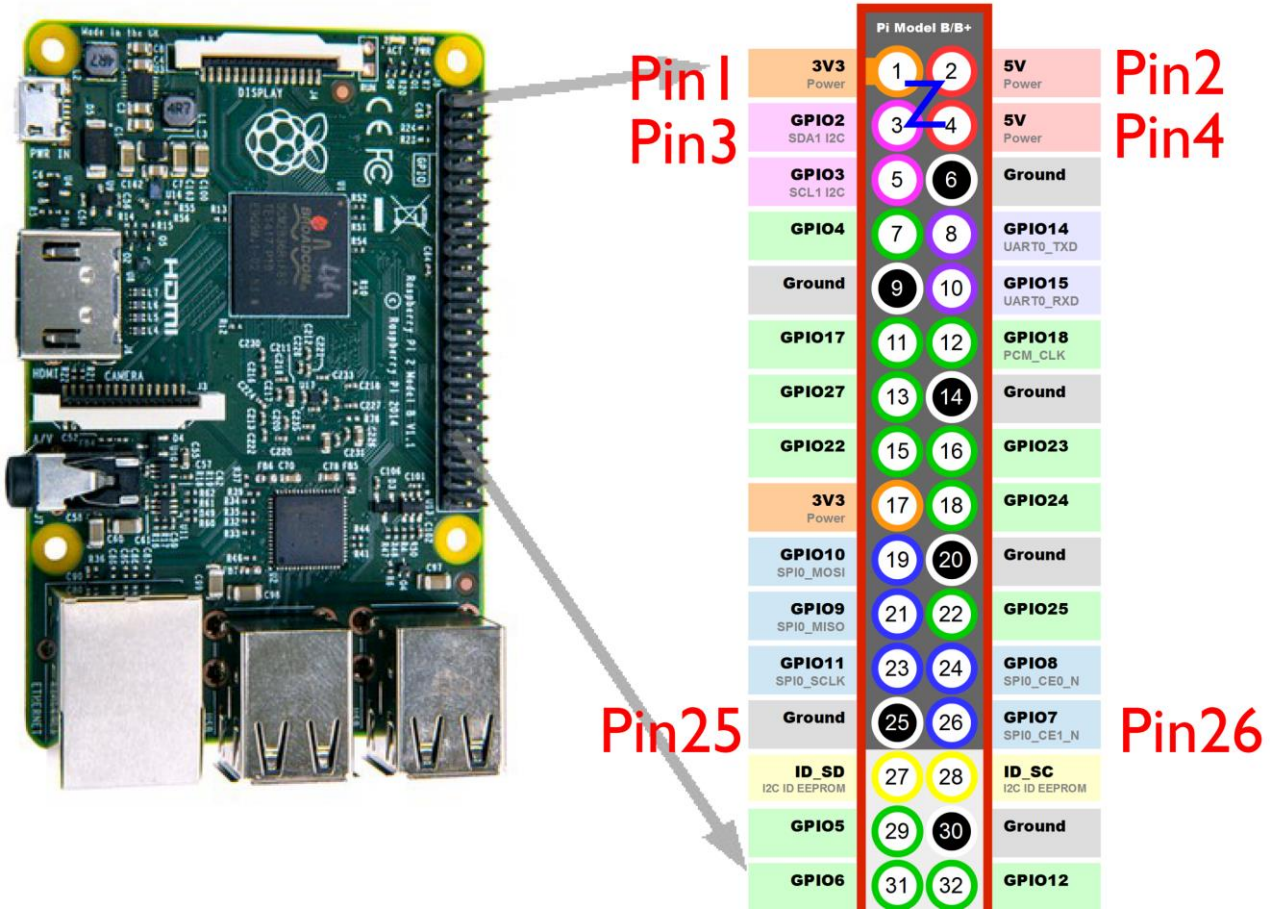


Physical Pin 7 = GPIO 4

BCM 4:
- Physical pin **7**
- BCM pin **4**
- Wiring Pi pin 7

0. GPIO introduction

- The PIN (Physical) numbering is in Z-shape



0. GPIO Limitations

- Do not put more than 3.3V on any GPIO pin being used as an input.
- Do not draw more than 16mA per output and keep the total for all outputs below 50mA in total for an older 26-pin Raspberry Pi, and below 100mA on a 40-pin Raspberry Pi.
- When using LEDs, 3mA is enough to light a red LED reasonably brightly with a 470Ω series resistor.
- Do not poke at the GPIO connector with a screwdriver or any metal object when the Pi is powered up.
- Do not power the Pi with more than 5V.
- Do not draw more than a total of 250mA from the 5V supply pins.

From: Raspberry Pi Cookbook: Software and Hardware Problems and Solutions
https://books.google.com.tw/books?id=0skvDAAAQBAJ&pg=PT270&lpg=PT270#v=onepage&q&f=false

0. Programming language for GPIO

- C
- C + wiringPi
- C#
- Ruby
- Perl
- **Python**
- Scratch
- Java Pi4J Library
- Shell script

We use Python

0. Python example

```
#  pound key is annotation
i = 3                          #  set i to 3
print i
i = [1, 2, 3, 4, 5]            #  set i to an array
print i[2]                     #  print number 3 (start from 0)
i =  "abcde"                   #  set i to a string
print i[0]                     #  print a
```

```
# import MODULE
import RPi.GPIO

# import MODULE as ALIAS
import RPi.GPIO as GPIO
```

```
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.OUT)
```

```
for i in xrange(start, stop[, step]) :
    process
```

It does not use {} in the code.
It uses 4 space in for-loop, if-else…

4 space here

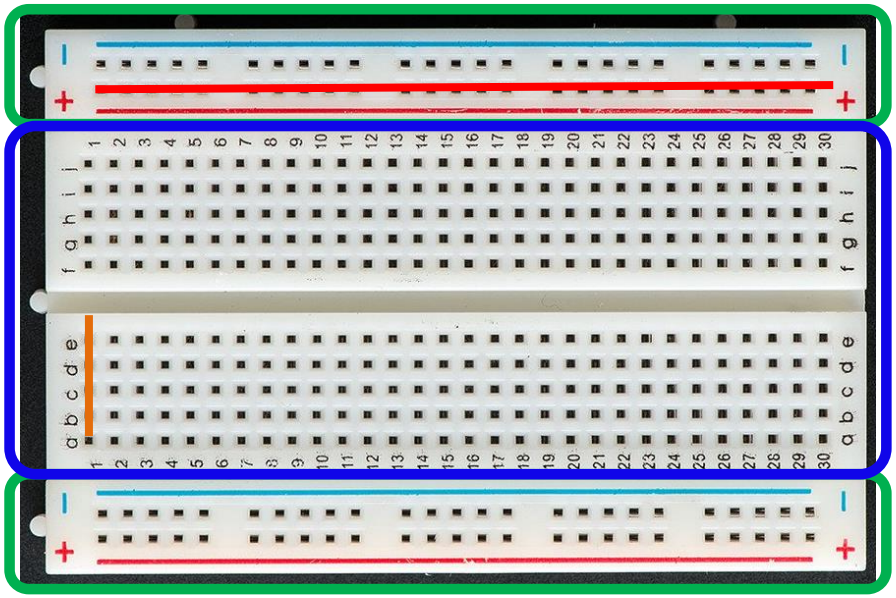
0. install Python

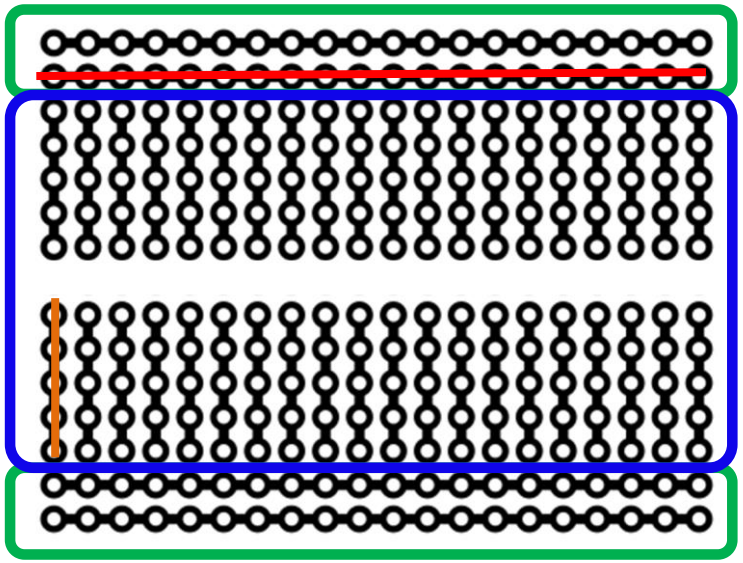
- Command
  - sudo apt-get update
  - sudo apt-get install -y python-dev python-pip python-rpi.gpio

- **Raspbian has already installed Python.**

Bread Board

Bus strips: one for ground (-) and one for a supply voltage (+)

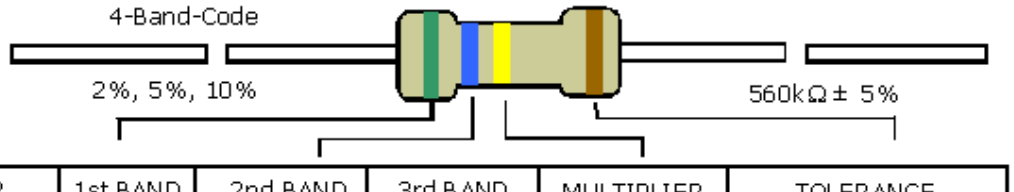




Terminal strips (ex: a1, b1, c1, d1 and e1 are connected)

Resistor

use color table to determine value



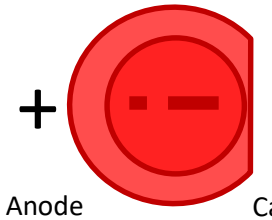
COLOR	1st BAND	2nd BAND	3rd BAND	MULTIPLIER	TOLERANCE	
Black	0	0	0	1Ω		
Brown	1	1	1	10Ω	± 1%	(F)
Red	2	2	2	100Ω	± 2%	(G)
Orange	3	3	3	1KΩ		
Yellow	4	4	4	10KΩ		
Green	5	5	5	100KΩ	±0.5%	(D)
Blue	6	6	6	1MΩ	±0.25%	(C)
Violet	7	7	7	10MΩ	±0.10%	(B)
Grey	8	8	8		±0.05%	
White	9	9	9			
Gold				0.1	± 5%	(J)
Silver				0.01	± 10%	(K)

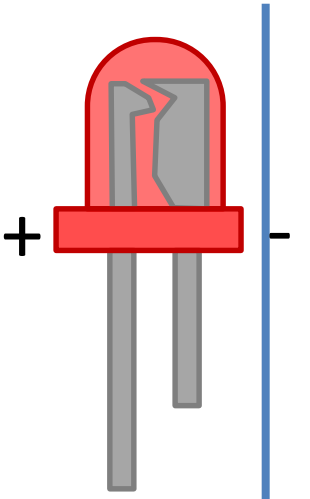
Ex: 20KΩ = Red, Black, Orange = 20*1000 (4-band)

= Red, Black, Black, Red = 200*100 (5-band)

LED

- The **LED** (**L**ight **E**mitting **D**iode) is a simple, digital **actuator**





- LEDs have a **short leg** (-) and a **long leg** (+) and it matters how they are oriented in a circuit
- To prevent damage, LEDs are used together with a 1KΩ **resistor** (or anything from 300Ω to 2KΩ)

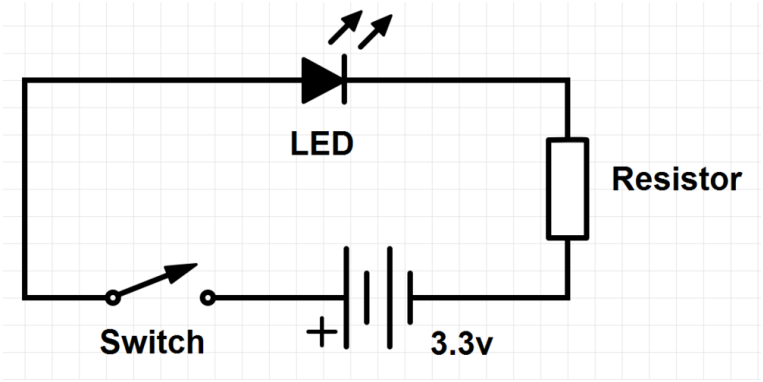
Electrical / Optical Characteristics at TA=25°C

Symbol	Parameter	Device	Typ.	Max.	Units	Test Conditions
λpeak	Peak Wavelength	Super Bright Red	660		nm	IF=20mA
λD [1]	Dominant Wavelength	Super Bright Red	640		nm	IF=20mA
Δλ1/2	Spectral Line Half-width	Super Bright Red	20		nm	IF=20mA
C	Capacitance	Super Bright Red	45		pF	VF=0V;f=1MHz
VF [2]	Forward Voltage	Super Bright Red	1.85	2.5	V	IF=20mA
IR	Reverse Current	Super Bright Red		10	uA	VR = 5V

Notes:
1.Wavelength: +/-1nm.
2. Forward Voltage: +/-0.1V.

LED

- 根據前面的參數
  - PI的GPIO腳位可以提供3.3V
  - LED的順向電壓是1.85V
  - LED的電流需要20mA



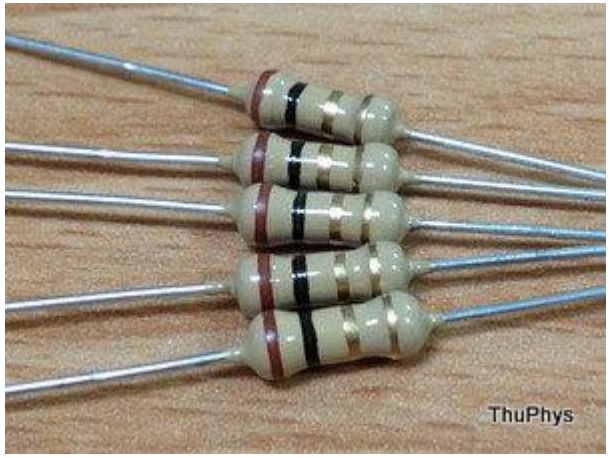
- 電阻公式: R=V/I
  - R =(3.3-1.85)/0.02=72.5 歐姆
  - 最少須接 72.5 歐姆的電阻，才可避免 LED燒毀

https://www.raspberrypi.org/documentation/usage/gpio/

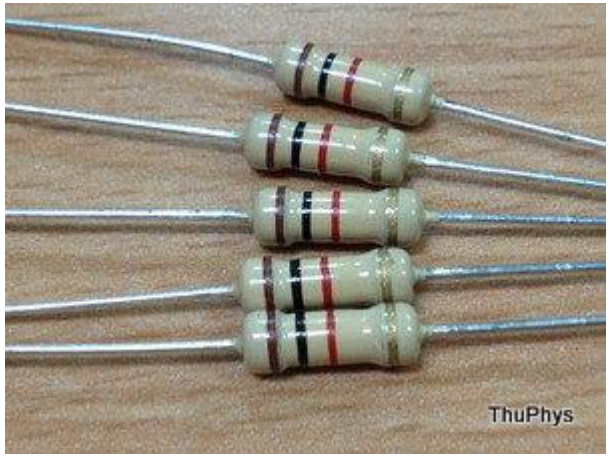
Discussion 1

- Identify the resistors. (Ω)
  - Write down the color which you see
  - Identify the corresponding resistor value
    - How to calculate the value?

A



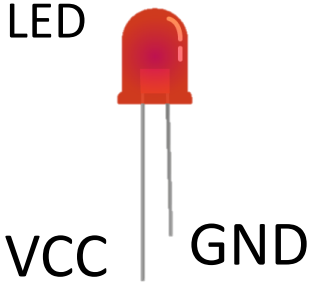
B

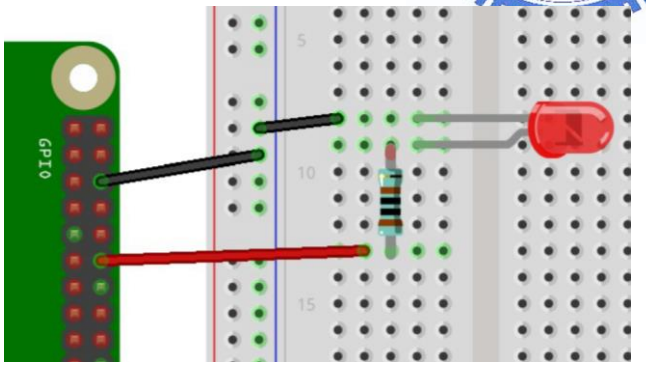


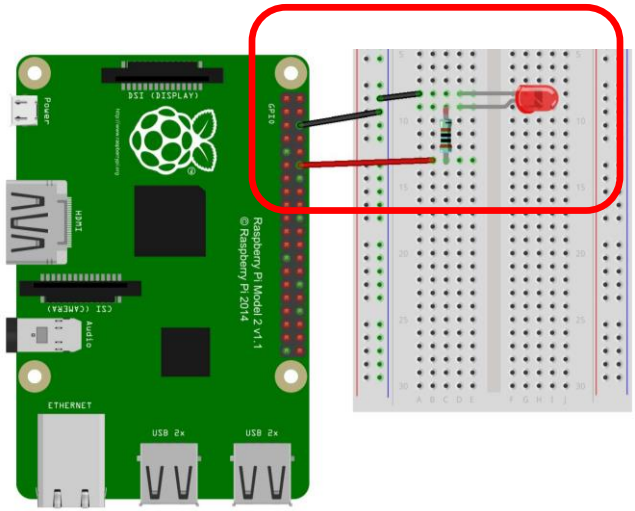
C

1. Control LED

- Goal: create a simplest thing to control GPIO and see physical output
- Output: it blinks an LED.
- Hardware Required
  - Raspberry PI
  - LED







fritzing

1. Control LED

```
import RPi.GPIO as GPIO
import time
```

Load GPIO library

```
LED_PIN = 12
GPIO.setmode(GPIO.BOARD)
GPIO.setup(LED_PIN, GPIO.OUT)
```

LED is on pin 12 by pin numbering (z-shape)

```
try:
    while True:
        print("LED is on")
        GPIO.output(LED_PIN, GPIO.HIGH)
        time.sleep(1)
        print("LED is off")
        GPIO.output(LED_PIN, GPIO.LOW)
        time.sleep(1)
```

the try clause (the statement(s) between the try and except keywords) is executed.

```
except KeyboardInterrupt:
    print "Exception: KeyboardInterrupt"
```

a user-generated interruption is signaled
(ctrl + c)

```
finally:
    GPIO.cleanup()
```

A finally clause is always executed before leaving the try statement, whether an exception has occurred or not.

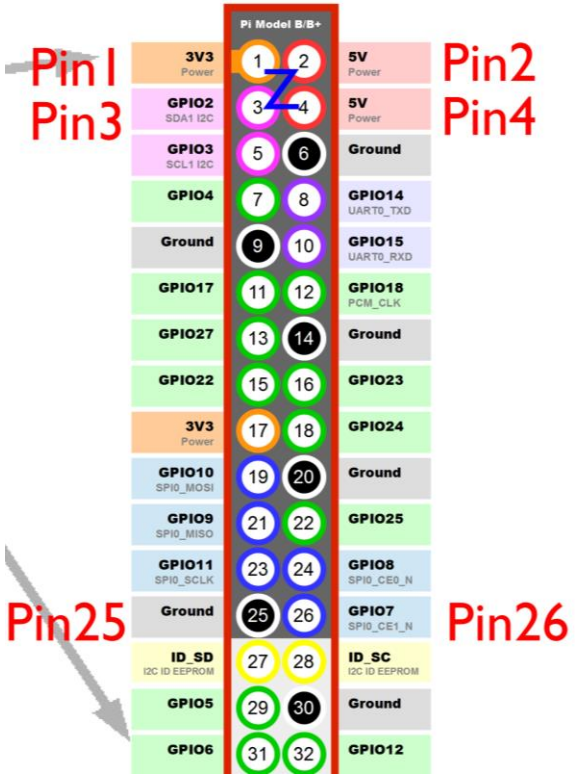
Syntax - GPIO

- GPIO.setmode(GPIO.BOARD) // two parameters
  - GPIO.BOARD: Define the pins by the number of the pin plug (z-shape)
  - GPIO.BCM: Define the pins by the "Broadcom SOC channel" number

- GPIO.setup(LED_PIN, GPIO.OUT)
  - Set LED_PIN to output mode

- GPIO.cleanup()
  - clean up all the ports you've used

[Sad story] *when you have a port set HIGH as an output and you accidentally connect it to GND (LOW), which would short-circuit the port and possibly fry it.*



http://raspi.tv/2013/rpi-gpio-basics-3-how-to-exit-gpio-programs-cleanly-avoid-warnings-and-protect-your-pi

Syntax – Python error statement

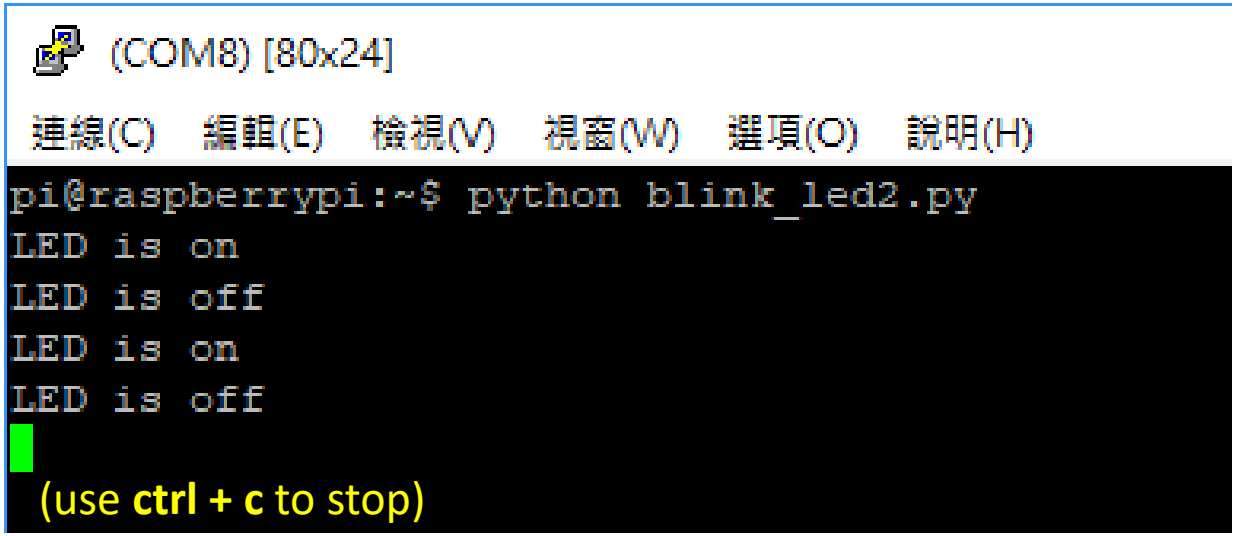
- The try statement works as follows.
  - First, the try clause (the statement(s) between the try and except keywords) is executed.
  - If no exception occurs, the except clause is skipped and execution of the try statement is finished.
  - If an exception occurs during execution of the try clause, the rest of the clause is skipped. Then if its type matches the exception named after the except keyword, the except clause is executed, and then execution continues after the try statement.
  - If an exception occurs which does not match the exception named in the except clause, it is passed on to outer try statements; if no handler is found, it is an unhandled exception and execution stops with a message as shown above.

```
try:
    while True:
        print("LED is on")
        GPIO.output(LED_PIN, GPIO.HIGH)
        time.sleep(1)
        print("LED is off")
        GPIO.output(LED_PIN, GPIO.LOW)
        time.sleep(1)

except KeyboardInterrupt:
    print "Exception: KeyboardInterrupt"

finally:
    GPIO.cleanup()
```

https://docs.python.org/2/tutorial/errors.html

How to write Python code on PI?

- Use text editor (ex: nano, vim … etc)
  - Ex: **nano blink.py**
- Write code, then save it
  - In nano, press **ctrl + x** to exit
- Execution (use **ctrl + c** to stop)
  - **sudo python blink.py**

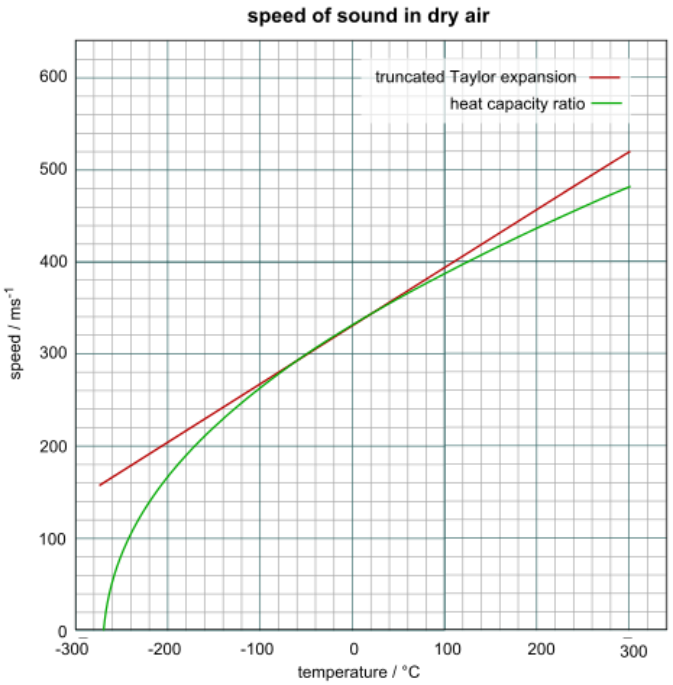


2. Ultrasonic (HC-SR04)

- Speed of sound
  - At 20°C (68°F), the speed is 343 m/s.
  - The approximate speed of sound (c) can be calculated from:

$$\mathbf{c_{air} = (331.3 + 0.606 * \theta)\ (m/s)}$$

where $\boldsymbol{\theta}$ is the temperature in degrees Celsius (°C).



$$Speed = \frac{Distance}{Time}$$

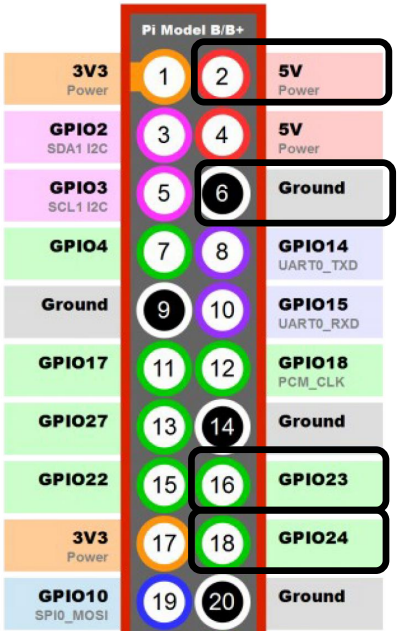
$$34300 = \frac{Distance}{Time/2}$$

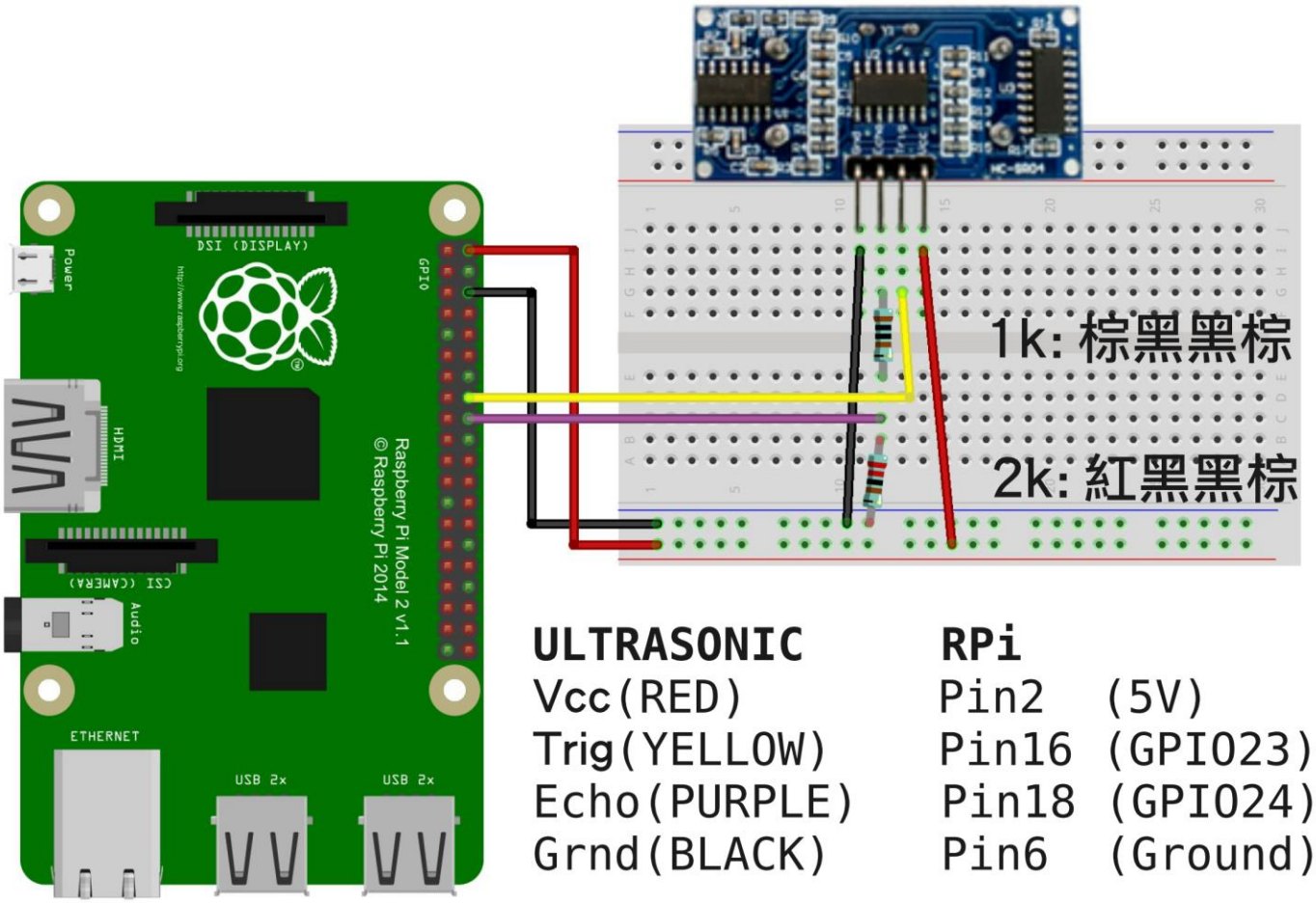
$$17150 = \frac{Distance}{Time}$$

$$17150 \times Time = Distance$$

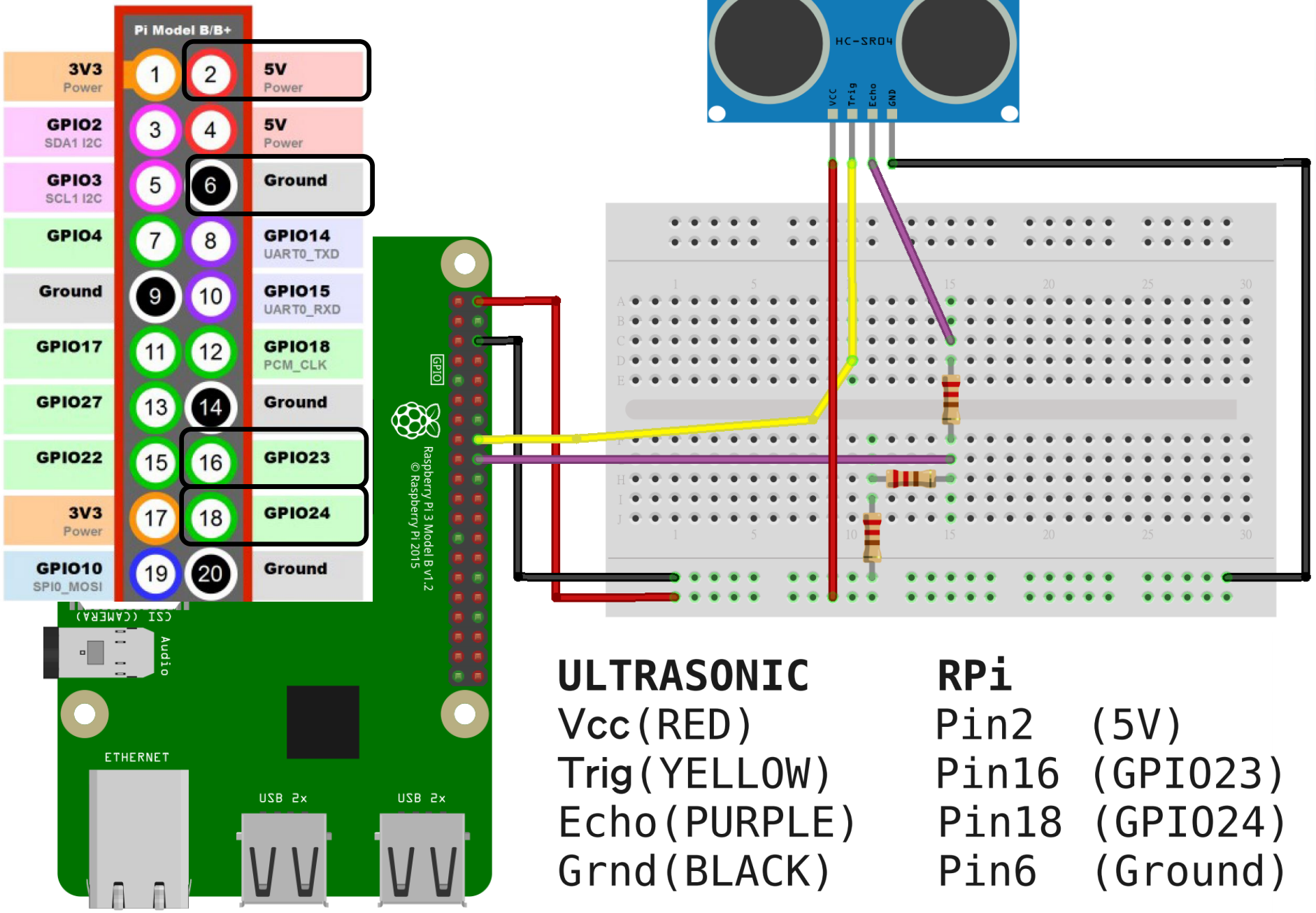
http://en.wikipedia.org/wiki/Speed_of_sound
https://www.modmypi.com/blog/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi

2. Ultrasonic (HC-SR04)





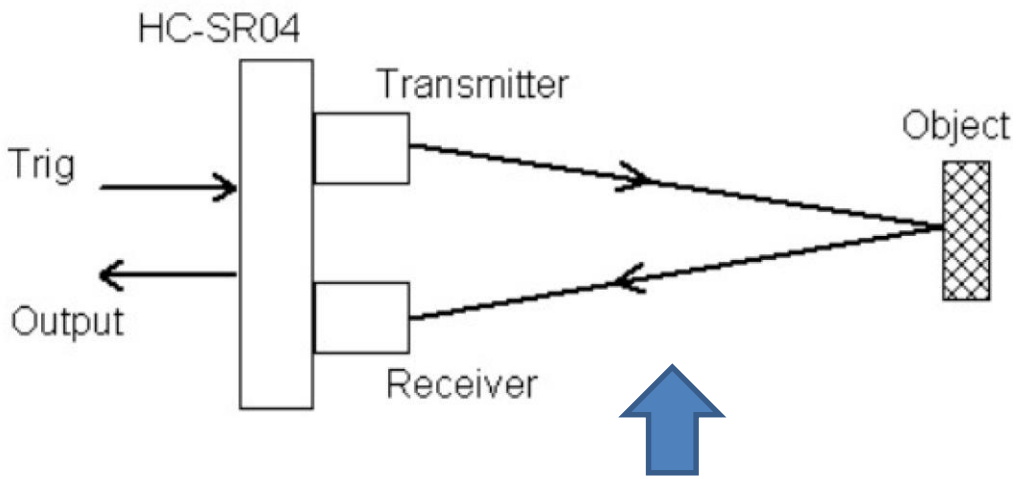
ULTRASONIC	RPi
Vcc(RED)	Pin2 (5V)
Trig(YELLOW)	Pin16 (GPIO23)
Echo(PURPLE)	Pin18 (GPIO24)
Grnd(BLACK)	Pin6 (Ground)



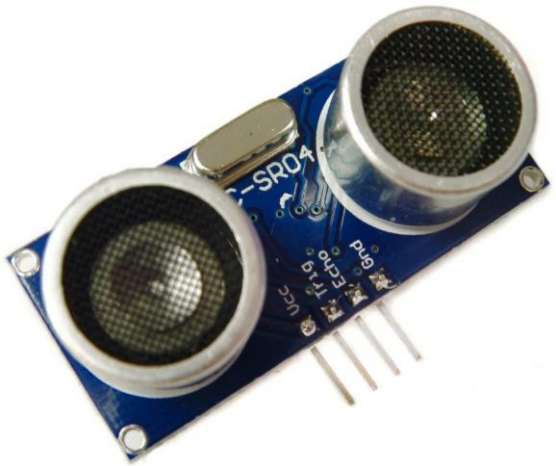
ULTRASONIC	RPi
Vcc(RED)	Pin2 (5V)
Trig(YELLOW)	Pin16 (GPIO23)
Echo(PURPLE)	Pin18 (GPIO24)
Grnd(BLACK)	Pin6 (Ground)

2. Ultrasonic (HC-SR04)

- 內建發射 (40kHz) 與接收電路
- 根據發射與接收的時間差計算距離
- 特殊功能 :US-020(長距離) 、 US-100(溫度補償)



可得到時間差



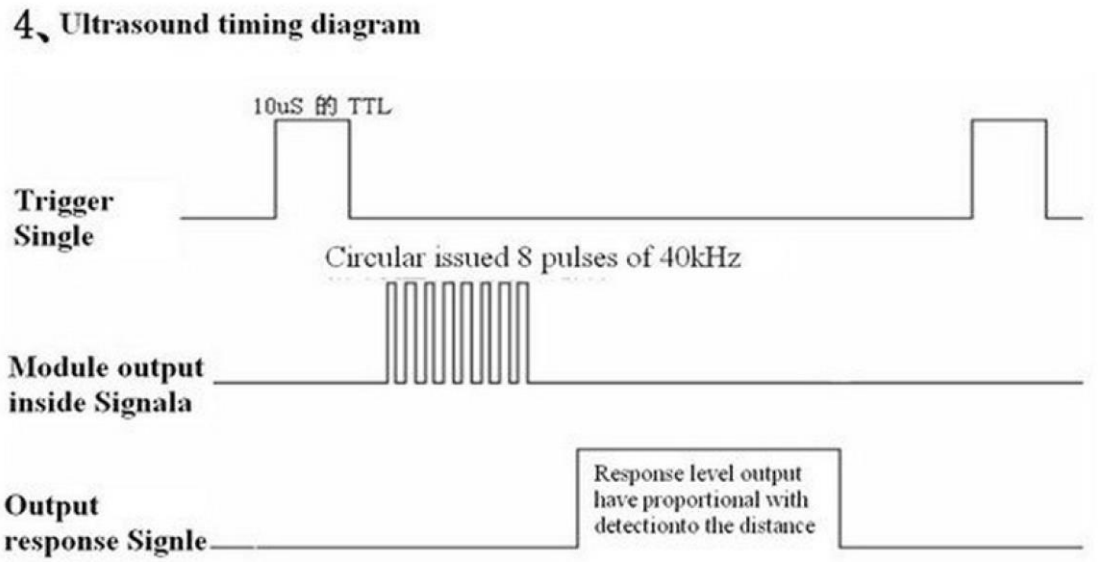
VCC, Trigger, Echo, GND

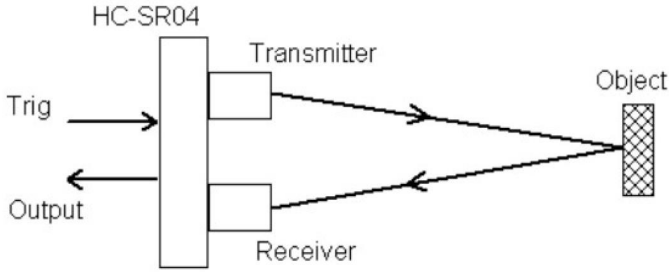
2. Ultrasonic (HC-SR04)

Specification

Working Voltage	DC 5 V
Working Current	15mA
Working Frequency	40Hz
Max Range	4m
Min Range	2cm
MeasuringAngle	15 degree
Trigger Input Signal	10uS TTL pulse
Echo Output Signal	Input TTL lever signal and the range in proportion
Dimension	45*20*15mm

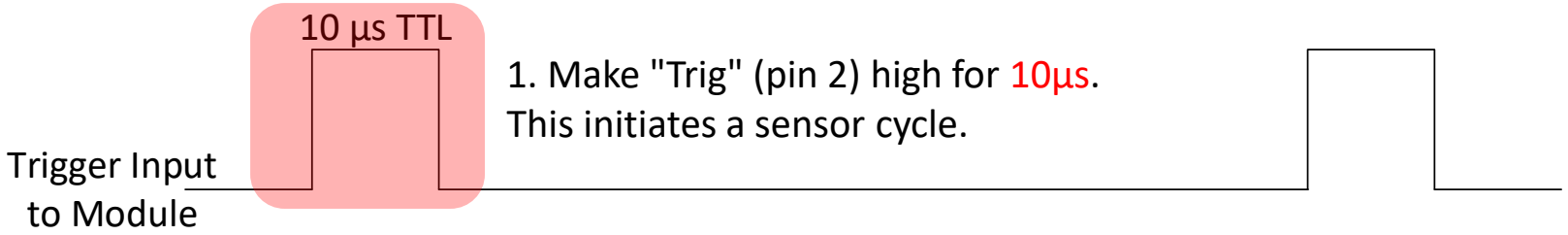
Timing diagram



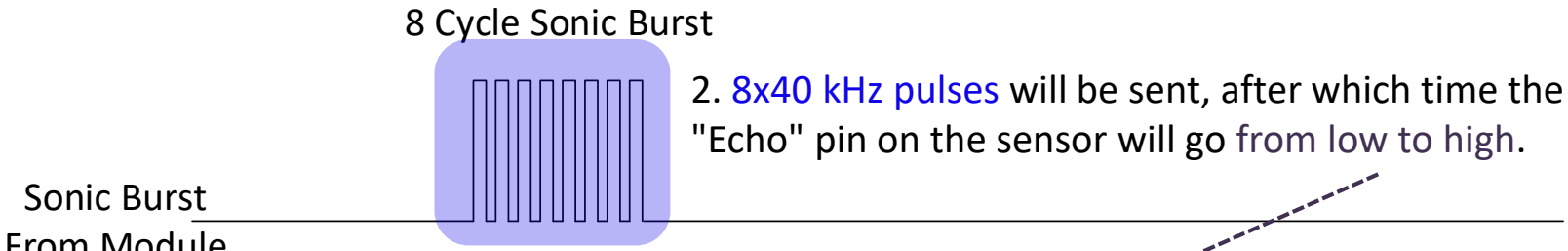


3. The 40kHz sound wave will bounce off the nearest object and return to the sensor.

VCC, Trigger, Echo, GND

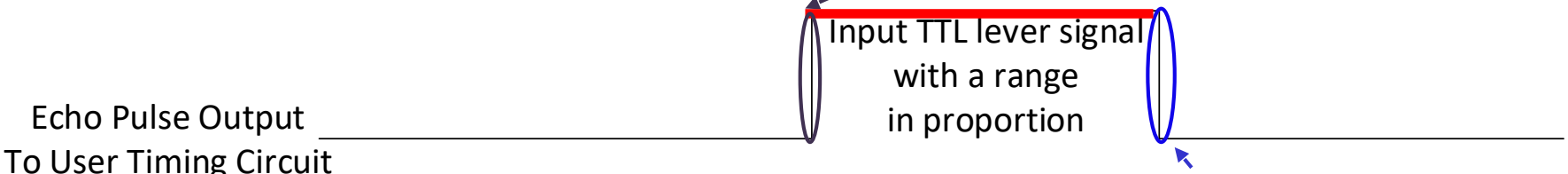


1. Make "Trig" (pin 2) high for 10µs. This initiates a sensor cycle.



2. 8x40 kHz pulses will be sent, after which time the "Echo" pin on the sensor will go from low to high.

5. The **distance** between the sensor and the detected object can be calculated based on **the length of time the Echo pin is high**.



4. When the sensor detects the reflected sound wave, the Echo pin will go low again.

http://www.micropik.com/PDF/HCSR04.pdf

2. Ultrasonic (HC-SR04)

1. Make "Trig" (pin 2) high for 10µs. This initiates a sensor cycle.
2. 8x40 kHz pulses will be sent, after which time the "Echo" pin on the sensor will go from low to high.
3. The 40kHz sound wave will bounce off the nearest object and return to the sensor.
4. When the sensor detects the reflected sound wave, the Echo pin will go low again.
5. The distance between the sensor and the detected object can be calculated based on the length of time the Echo pin is high.
6. If no object is detected, the Echo pin will stay high for 38ms and then go low.

Datasheet: http://www.micropik.com/PDF/HCSR04.pdf

2. Ultrasonic (HC-SR04)

- TRIG 腳位收到高電位 (3.3V) 後發送超聲波
- ECHO 腳位維持低電位 (0V), 收到回應後拉到高電位 (5V)
- Raspberry Pi 腳位的容忍電位為 3.3V
  - => 將 ECHO 腳位的 5V 降壓為 3.3V 左右

$$\frac{3.3}{5} = \frac{R2}{1000 + R2}$$

$$0.66 = \frac{R2}{1000 + R2}$$

$$0.66(1000 + R2) = R2$$

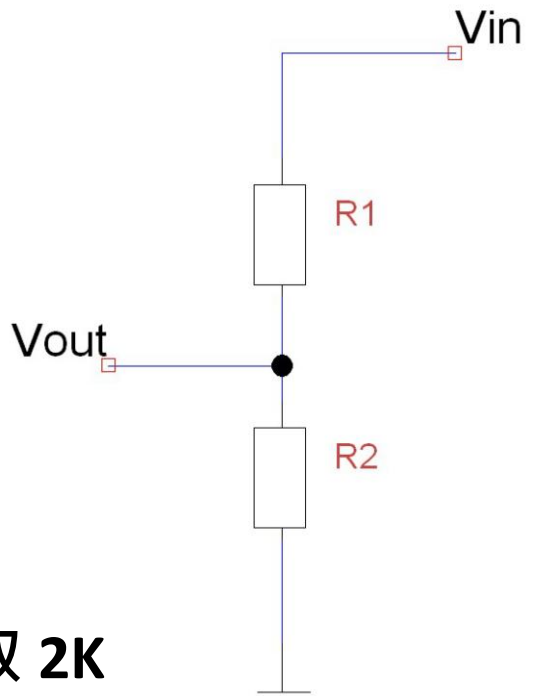
$$660 + 0.66R2 = R2$$

$$660 = 0.34R2$$

$$1941 = R2$$

$$Vout = Vin \times \frac{R2}{R1 + R2}$$

$$\frac{Vout}{Vin} = \frac{R2}{R1 + R2}$$



計算結果: R1=1K, R2 取 2K

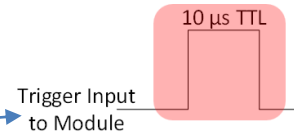
https://www.modmypi.com/blog/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi

2. Ultrasonic (HC-SR04)

```python
#!/usr/sbin/env python
import RPi.GPIO as GPIO
import time

v = 343          # (331 + 0.6*20)
TRIGGER_PIN = 16
ECHO_PIN = 18
GPIO.setmode(GPIO.BOARD)
GPIO.setup(TRIGGER_PIN,GPIO.OUT)
GPIO.setup(ECHO_PIN,GPIO.IN)

def measure() :
    GPIO.output(TRIGGER_PIN, GPIO.HIGH)
    time.sleep(0.00001)      # 10uS
    GPIO.output(TRIGGER_PIN, GPIO.LOW)

    while GPIO.input(ECHO_PIN) == GPIO.LOW:
        pulse_start = time.time()
    while GPIO.input(ECHO_PIN) == GPIO.HIGH:
        pulse_end = time.time()
    t = pulse_end - pulse_start
    d = t * v
    d = d/2
    return d*100

print measure()
GPIO.cleanup()
```

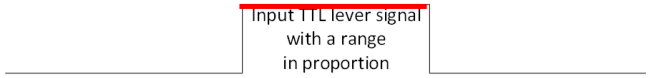
Load library

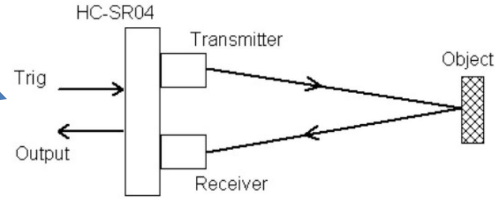
ULTRASONIC	RPi
Vcc(RED)	Pin2 (5V)
Trig(YELLOW)	Pin16 (GPIO23)
Echo(PURPLE)	Pin18 (GPIO24)
Grnd(BLACK)	Pin6 (Ground)



1. Make "Trig" (pin 2) high for 10µs. This initiates a sensor cycle.

5. The **distance** between the sensor and the detected object can be calculated based on **the length of time the Echo pin is high**.





2. Ultrasonic (HC-SR04)

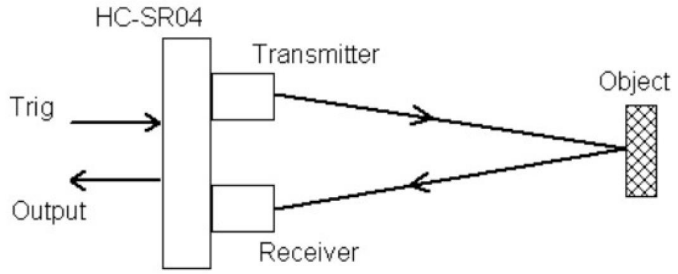
- Create a new Python code
  - nano ultrasonic_distance.py
- Run the code
  - sudo python ultrasonic_distance.py

```
#!/usr/sbin/env python
import RPi.GPIO as GPIO
import time

v = 343         # (331 + 0.6*20)
TRIGGER_PIN = 16
ECHO_PIN = 18
GPIO.setmode(GPIO.BOARD)
GPIO.setup(TRIGGER_PIN,GPIO.OUT)
GPIO.setup(ECHO_PIN,GPIO.IN)

def measure() :
    GPIO.output(TRIGGER_PIN, GPIO.HIGH)
    time.sleep(0.00001)     # 10uS
    GPIO.output(TRIGGER_PIN, GPIO.LOW)
    pulse_start = time.time()
    while GPIO.input(ECHO_PIN) == GPIO.LOW:
```

This is picture!
Try to write code by yourself.

Discussion 2

- How does **Trigger** and **Echo** pins work together?
- Do some experiments. What is the **max distance** that your ultrasonic sensor can detect?
- In addition to distance measurement, is there any ultrasonic application?



max distance??

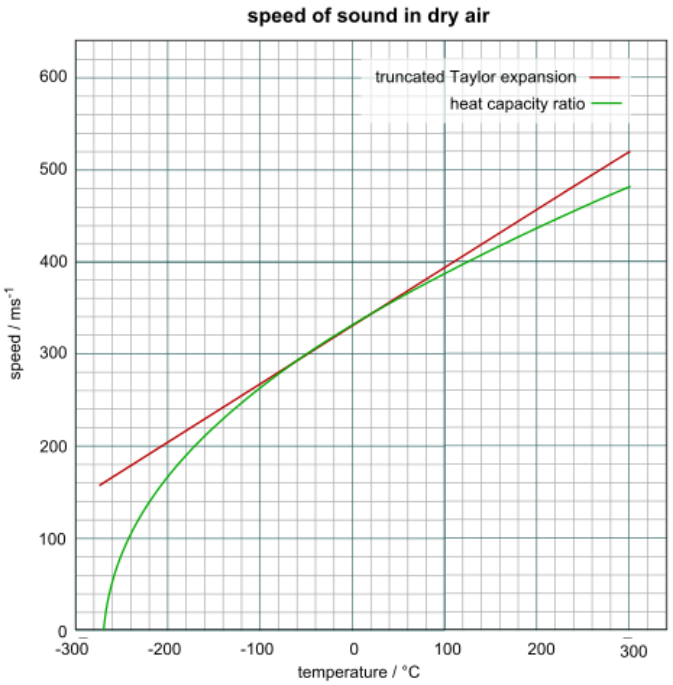
VCC, **Trigger, Echo**, GND

3. Temperature (DHT-11)

- Speed of sound
  - At 20°C (68°F), the speed is 343 m/s.
  - The approximate speed of sound (c) can be calculated from:

$$\mathbf{c_{air} = (331.3 + 0.606 * \theta)\ (m/s)}$$

where θ is the temperature in degrees Celsius (°C).



$$Speed = \frac{Distance}{Time}$$

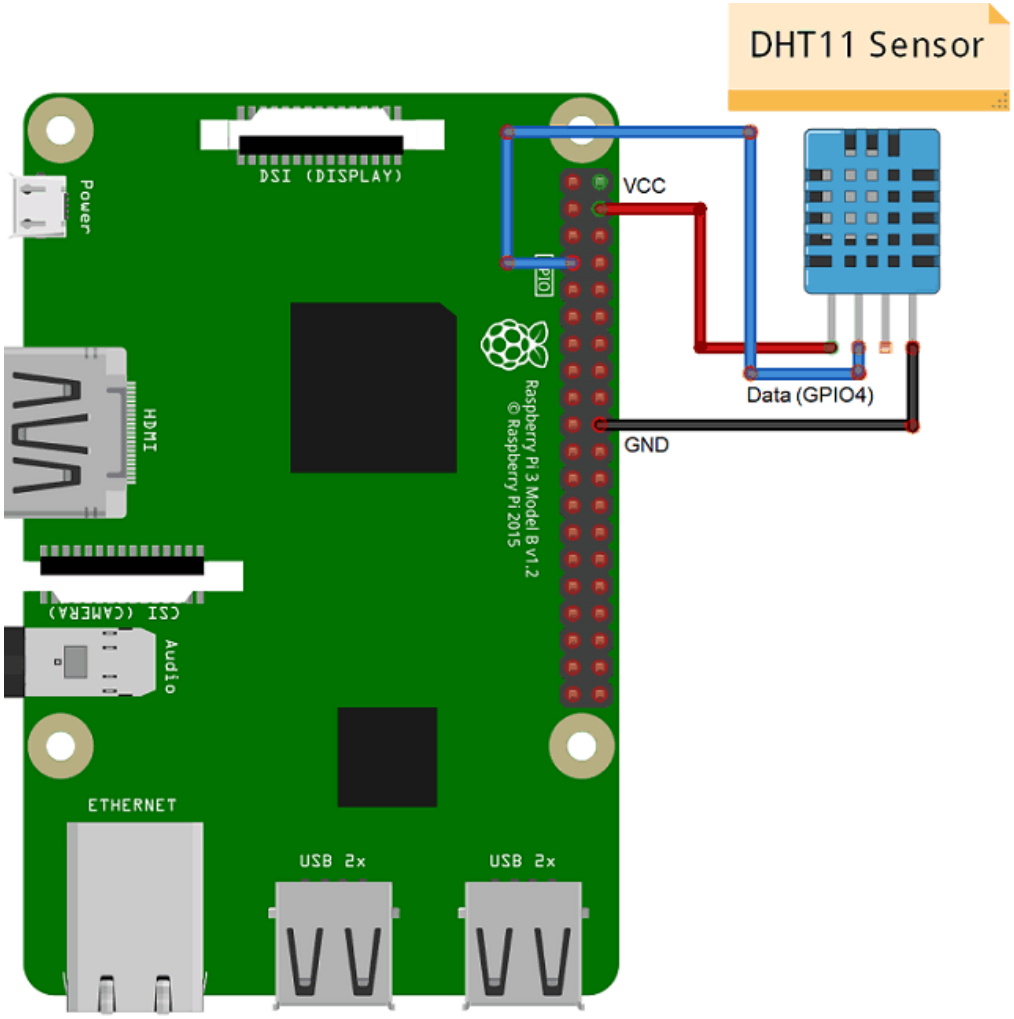
$$34300 = \frac{Distance}{Time/2}$$

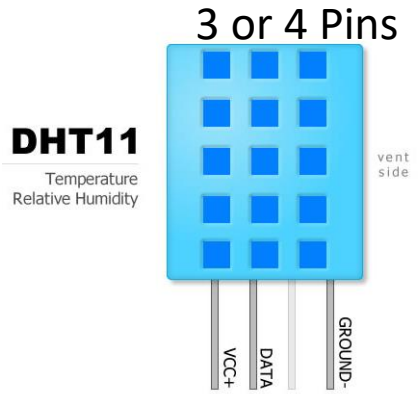
$$17150 = \frac{Distance}{Time}$$

$$17150 \times Time = Distance$$

http://en.wikipedia.org/wiki/Speed_of_sound
https://www.modmypi.com/blog/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi

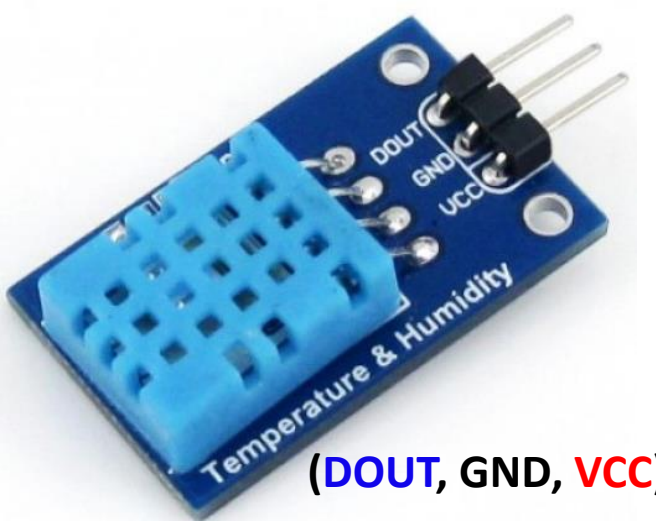
3. DHT-11



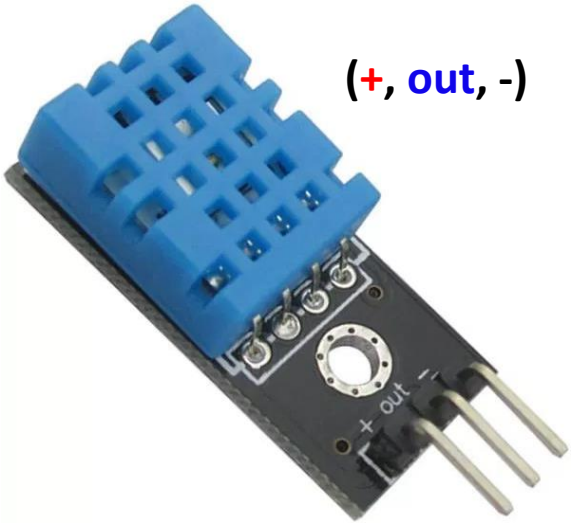


VCC, Data, GND
(+, out, -)

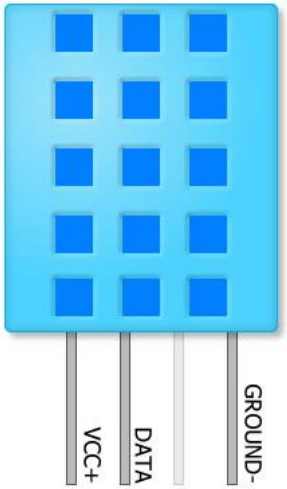
3. All kinds of DHT-11



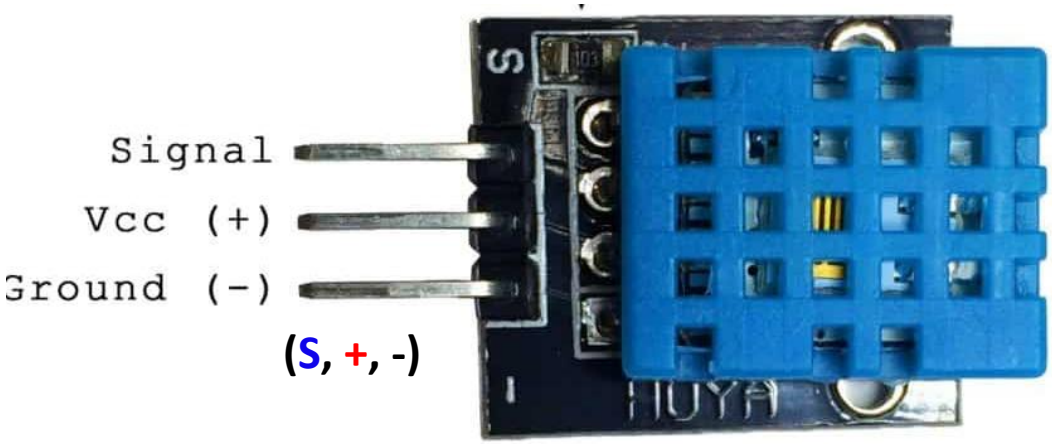
(DOUT, GND, VCC)



(+, out, -)



(VCC, Data, GND)

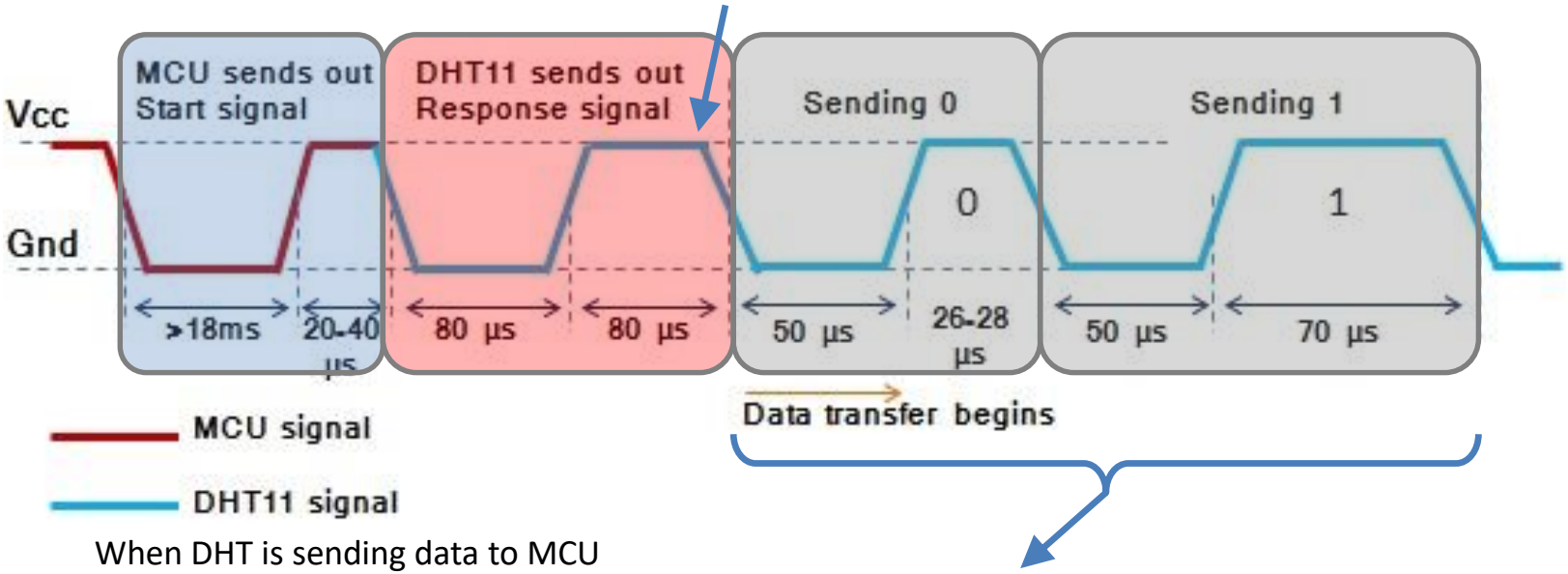


(S, +, -)

3. DHT-11: Communication Process

- "**Start**" and "**Response**" signals.
- **Data** (40-bit) = Integer Byte of RH + Decimal Byte of RH + Integer Byte of Temp. + Decimal Byte of Temp. + Checksum Byte.

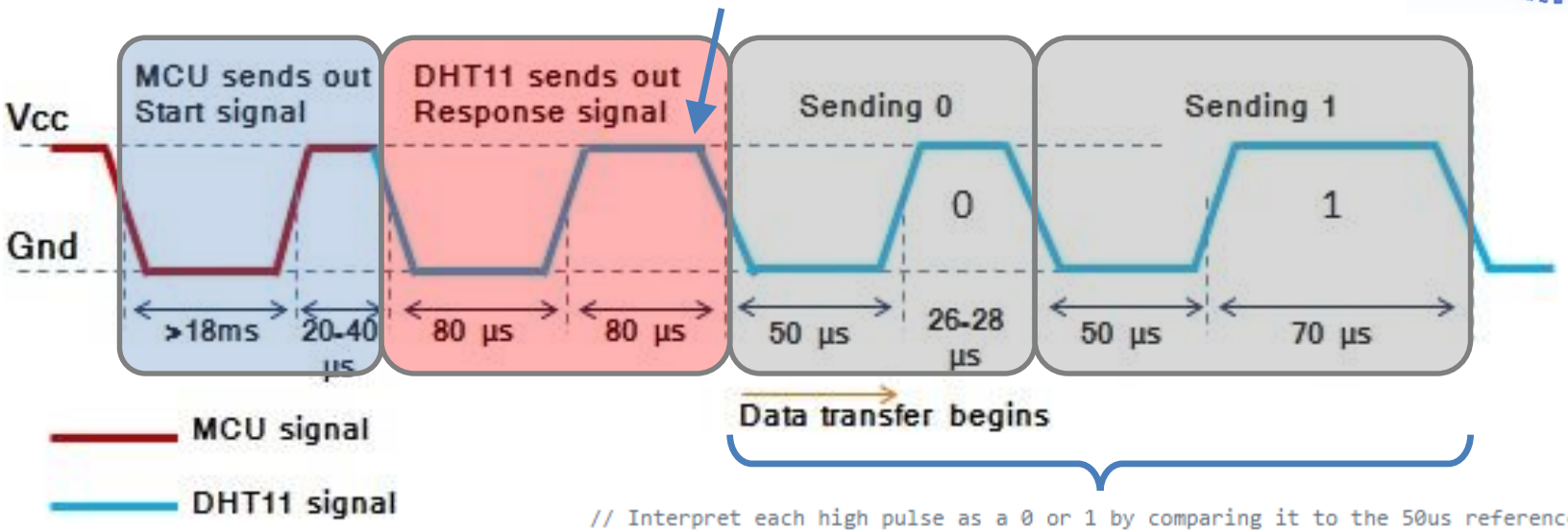
pull up voltage and keeps it for 80us and prepares for data transmission



When DHT is sending data to MCU

- every bit of data begins with the 50us low-voltage-level
- the length of the following high-voltage-level signal determines whether data bit is "0" or "1"

3. DHT-11: Communication Process




```
// Interpret each high pulse as a 0 or 1 by comparing it to the 50us reference.
// If the count is less than 50us it must be a ~28us 0 pulse, and if it's higher
// then it must be a ~70us 1 pulse.
uint8_t data[5] = {0};
for (int i=3; i < DHT_PULSES*2; i+=2) {
  int index = (i-3)/16;
  data[index] <<= 1;
  if (pulseCounts[i] >= threshold) {
    // One bit for long pulse.
    data[index] |= 1;
  }
  // Else zero bit for short pulse.
}
```

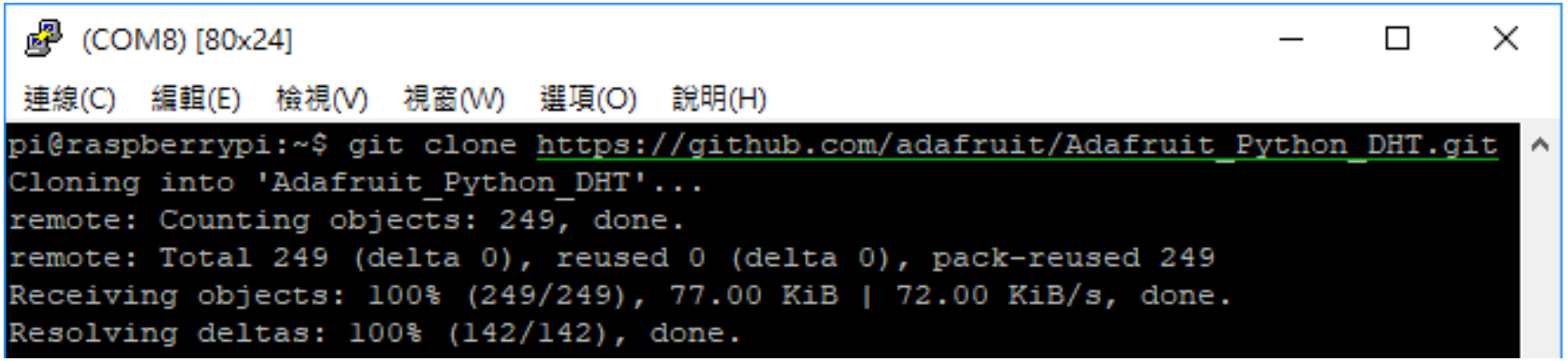
source/Raspberry_Pi_2/pi_2_dht_read.c

3. DHT-11

- Install Adafruit source code
  - Source code: https://github.com/adafruit/Adafruit_Python_DHT
- Install related tools
  - sudo apt-get update
  - sudo apt-get install git-core build-essential python-dev

3. DHT-11

- Download source code
  - Enter the command on terminal
  - **git clone https://github.com/adafruit/Adafruit_Python_DHT.git**



- Install Adafruit_Python_DHT
  - **cd Adafruit_Python_DHT**
  - **sudo python setup.py install**

3. DHT-11

- Message of installing Adafruit_Python_DHT

(COM8) [80x24]

連線(C) 編輯(E) 檢視(V) 視窗(W) 選項(O) 說明(H)

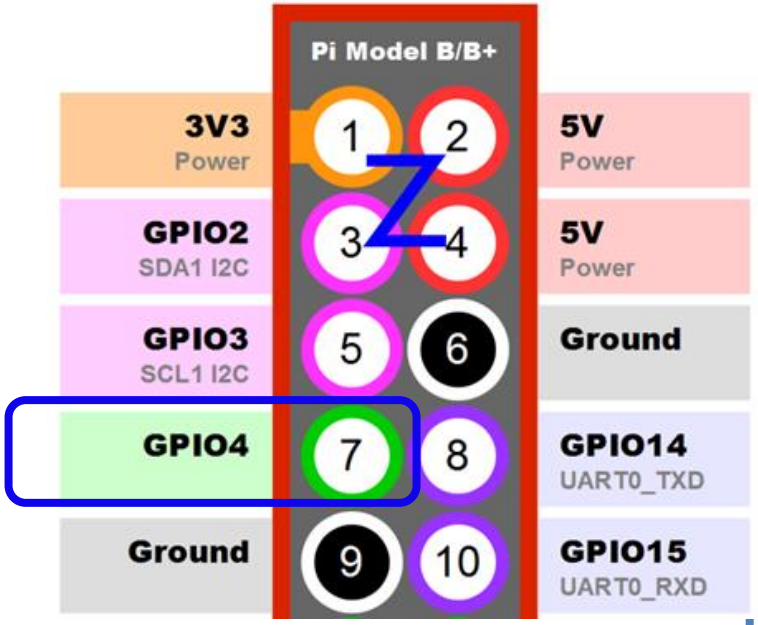
```
pi@raspberrypi:~$ cd Adafruit_Python_DHT
pi@raspberrypi:~/Adafruit_Python_DHT$ sudo python setup.py install
running install
running bdist_egg
running egg_info
creating Adafruit_DHT.egg-info
writing Adafruit_DHT.egg-info/PKG-INFO
writing top-level names to Adafruit_DHT.egg-info/top_level.txt
```

```
Copying Adafruit_DHT-1.3.2-py2.7-linux-armv7l.egg to /usr/local/lib/python2.7/di
st-packages
Adding Adafruit-DHT 1.3.2 to easy-install.pth file

Installed /usr/local/lib/python2.7/dist-packages/Adafruit_DHT-1.3.2-py2.7-linux-
armv7l.egg
Processing dependencies for Adafruit-DHT==1.3.2
Finished processing dependencies for Adafruit-DHT==1.3.2
pi@raspberrypi:~/Adafruit_Python_DHT$
```

Finish

3. DHT-11

- Execute sample code
  - **cd examples**
  - **sudo ./AdafruitDHT.py 11 4**
    - **11**: use DHT 11
      (it also supports DHT 22, DHT 2302)
    - **4**: use **GPIO 4** (Pin 7)
      - It use **GPIO.BCM** to define the pins
      - "Broadcom SOC channel" number




```
pi@raspberrypi ~ $ cd Adafruit_Python_DHT/examples/
pi@raspberrypi ~/Adafruit_Python_DHT/examples $ sudo ./AdafruitDHT.py 11 4
Temp=26.0*  Humidity=37.0%
```

Temp=26.0*C Humidity=37.0%

3. DHT-11

```
import sys
import Adafruit_DHT
```

Load libraries

```
sensor_args = { '11': Adafruit_DHT.DHT11,
                '22': Adafruit_DHT.DHT22,
                '2302': Adafruit_DHT.AM2302 }
```

Parse the parameters after .py
$ sudo ./AdafruitDHT.py 11 4

```
if len(sys.argv) == 3 and sys.argv[1] in sensor_args:
    sensor = sensor_args[sys.argv[1]]
    pin = sys.argv[2]
else:
    print('usage: sudo ./Adafruit_DHT.py [11|22|2302] GPIOpin#')
    print('example: sudo ./Adafruit_DHT.py 2302 4 - Read from an AM2302 connected to GPIO #4')
    sys.exit(1)
```

Print the usage

```
humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
```

Read sensor data

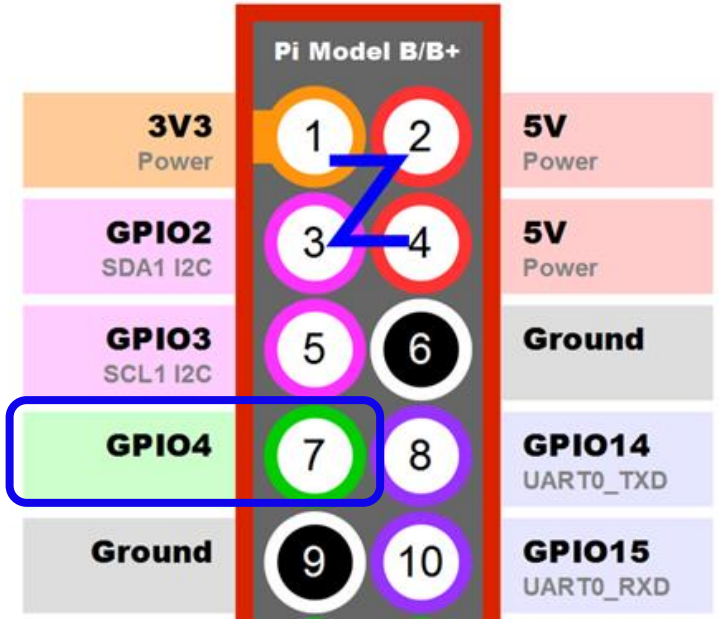
```
if humidity is not None and temperature is not None:
    print('Temp={0:0.1f}*  Humidity={1:0.1f}%'.format(temperature, humidity))
else:
    print('Failed to get reading. Try again!')
    sys.exit(1)
```

Return sensor data

Read more: Adafruit_DHT/source/Raspberry_Pi_2/pi_2_dht_read.c

Discussion 3

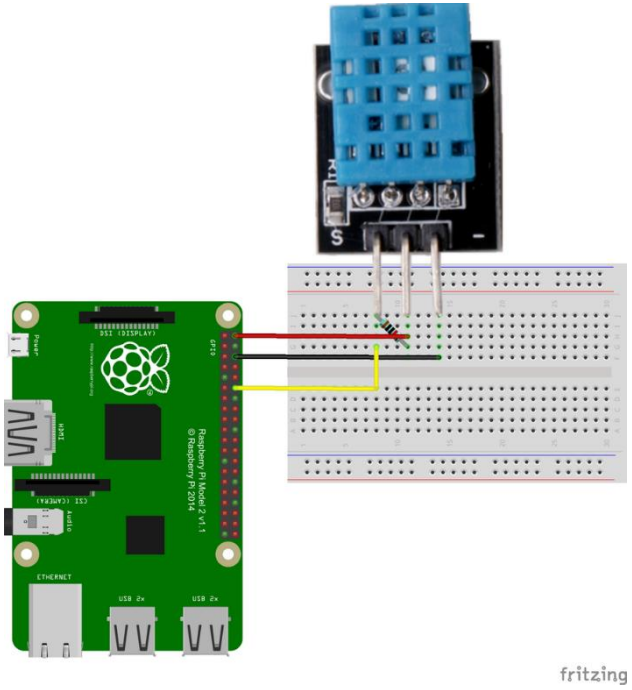
- Execute sample code
  - **cd examples**
  - **sudo ./AdafruitDHT.py 11 4**
    - **11**: use DHT 11
      (it also supports DHT 22, DHT 2302)
    - **4**: use **GPIO 4** (Pin 7)
      - It use **GPIO.BCM** to define the pins
      - "Broadcom SOC channel" number

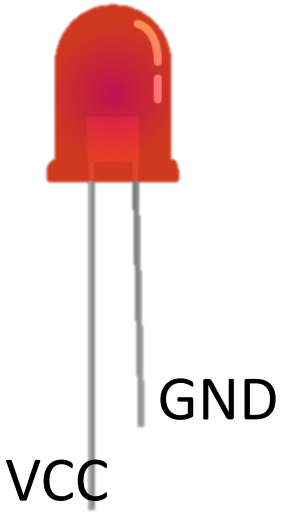


If we want to use Physical PIN number, how to modify the code?

Quiz 1

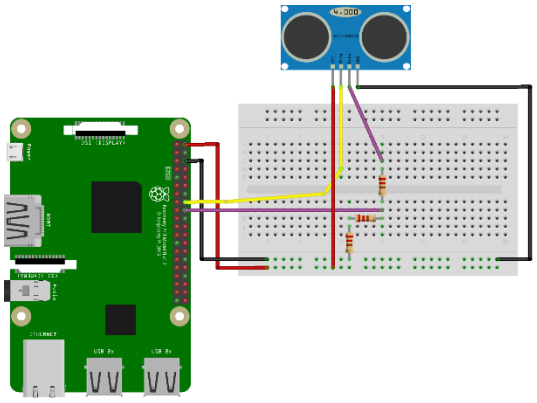
- Temperature alarm (溫度警示燈)
  - When the temperature exceeds the threshold (ex: 26.0*C), turn on the LED.

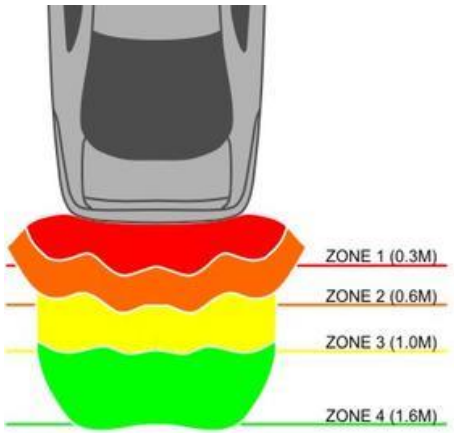


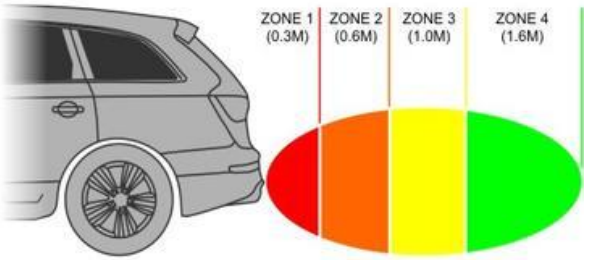


Quiz 2

- Design a Parking Assist System (倒車雷達)
  - use ultrasonic sensor, speaker
  - Divide the detecting distance into three parts: **a) Safe; b) Be careful; and c) Dangerous**.
  - Use the blinking LED to reminder the driver.
    - **Safe**: no response (> 1m)
    - **Be careful**: blinking (0.3 to 1m)
    - **Dangerous**: fast blinking (<0.3 m)







Summary

- Practice Lab (LED, ultrasonic , DHT11)
- Write down the answer for discussion
  - Discussion 1: Identify the resistors
  - Discussion 2: Ultrasonic
  - Discussion 3: How to assign Physical PIN number
- Write code for Quiz 1 - 2, then demonstrate it to TAs
  - Quiz1: Temperature alarm
  - Quiz2: Design a Parking Assist System