Lesson 6 I2C Communication Tutorial

1. I2C Communication Introduction

"IIC" is also known as "I2C" (Inter-Integrated Circuit), and it is an abbreviation for "IICBus". Therefore, its full name is "Integrated Circuit Bus".

It is a type of serial communication bus that consists of a Date wire (SDA) and a clock wire (SCL), allowing for bidirectional communication between a CPU (microcontroller) and an I2C module or between multiple I2C modules for data transmission and reception.

I2C's characteristic is that it is a half-duplex, rather than a full-duplex.

I2C is a true multi-master bus (Unlike SPI which requires the determination of a master before each communication, I2C allows for changing of the master during the communication process). If two or more masters request access to the I2C simultaneously, it can detect conflicts and arbitrate to avoid damaging the I2C data.

Start and stop signals are sent by the master connected to devices on the I2C bus. If a device features the I2C hardware interface, it can easily detect the start and stop signals.

A 7-bit slave address and a 1-bit direction byte must be sent by the master after the start signal. Use "0" to represent data sent from the master and "1" to indicate data received by the master.

Whenever the master sends a byte of data to the slave, it always needs to wait for an acknowledgment signal sent from the slave to confirm if the data is successfully received.

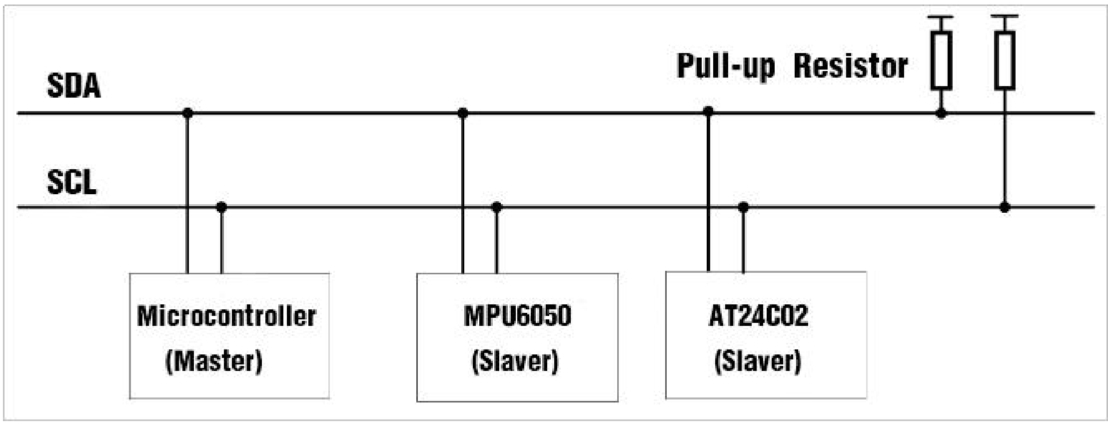
While the start signal is necessary, the stop and acknowledgment signals can be omitted depending on the actual situation.

- I2C Physical Connection

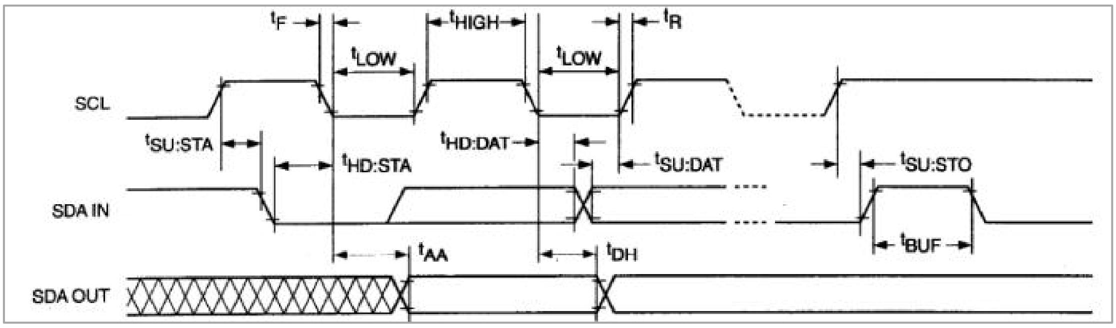
I2C communication is widely used in I2C devices such as the gyro-accelerometer MPU6050 and the EEPROM storage chip AT24C02, which can transmit data with microcontrollers via I2C bus.

I2C only consists of two wires for communication. The high and low voltage level of the data wire SDA transmits binary data, and the clock wire SCL uses a square wave signal to provide a clock pulse.

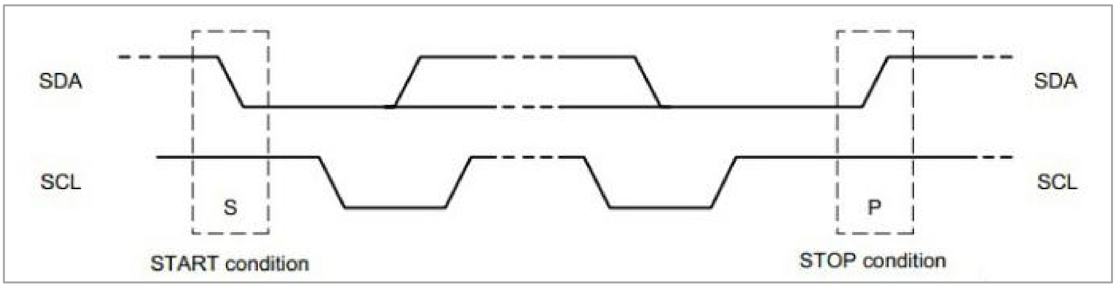
Multiple I2C devices can be connected in parallel on the I2C. Each device has its specific address for time-divisional sharing of the I2C. In actual use, the I2C also needs to be connected to a power supply and a common GND.



- I2C Sequence



- I2C Start and Stop Signals

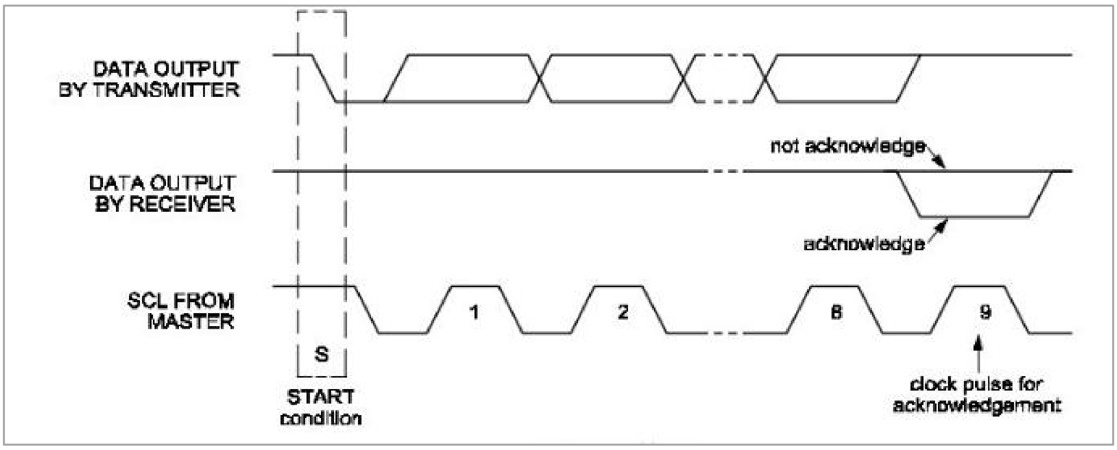


Start: When the clock wire SCL is high, the data wire SDA is from high to low.

Stop: When the clock wire SCL is high, the data wire SDA is from low to high.

Note: When SCL and SDA are both high, the I2C is in an inactive state.

- I2C Acknowledgment



The waveform below: SCL, the clock pulse generated by the master.

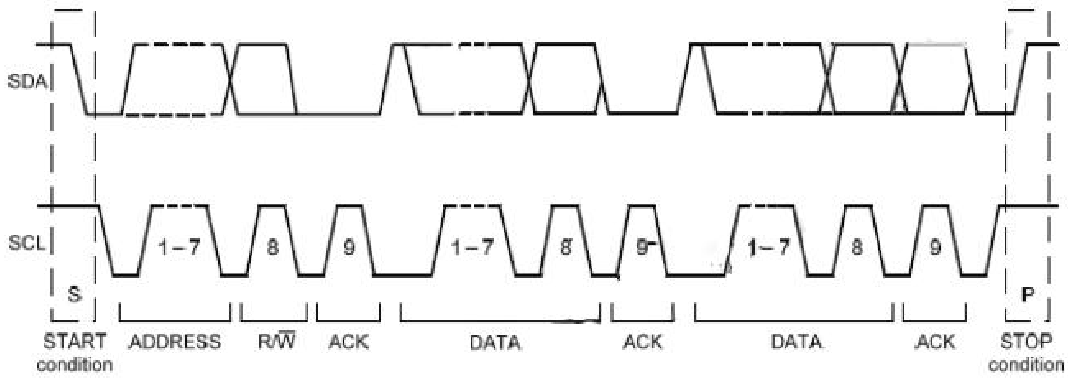
The waveform above: SDA, the 8-bit data sent from the master.

The waveform in the middle: SDA, the slave responses at the ninth clock signal pulling down the SDA wire, which indicates that the data sent from the master has been received. If the response is pulled up, it means that the slave is not acknowledging the data.

Note: Actually, the waveform above and the waveform in the middle are the same SDA wire shown separately in the diagram for clarity. This is because I2C acknowledgment is an interrelation. The microcontroller sends data to the I2C device which must acknowledge the data is received. Similarly, if the microcontroller receives data sent from the I2C device, it must also give an acknowledgment of receiving the data to the I2C device.

If they both need each other's acknowledgment after sending data, then when is it appropriate to not acknowledge each other's message? That is after reading the current data and there is no need for further reading, the one receiving and reading the data sends a no-acknowledgment signal to the other who sends the data, and it assumes no receiving of the data to stop sending data.

- I2C Complete Transmission Sequence



After the start sign (S) is sent, the master sends a 7-bit slave address, followed by an eighth bit named as Read or Write (R/W) bit.

The R/W bit represents whether the master is receiving the data from the slave or writing data to it.

After that, the master releases the SDA wire to wait for the acknowledgment signal (ACK) sent from the slave. An ACK bit follows the transmission of each byte.

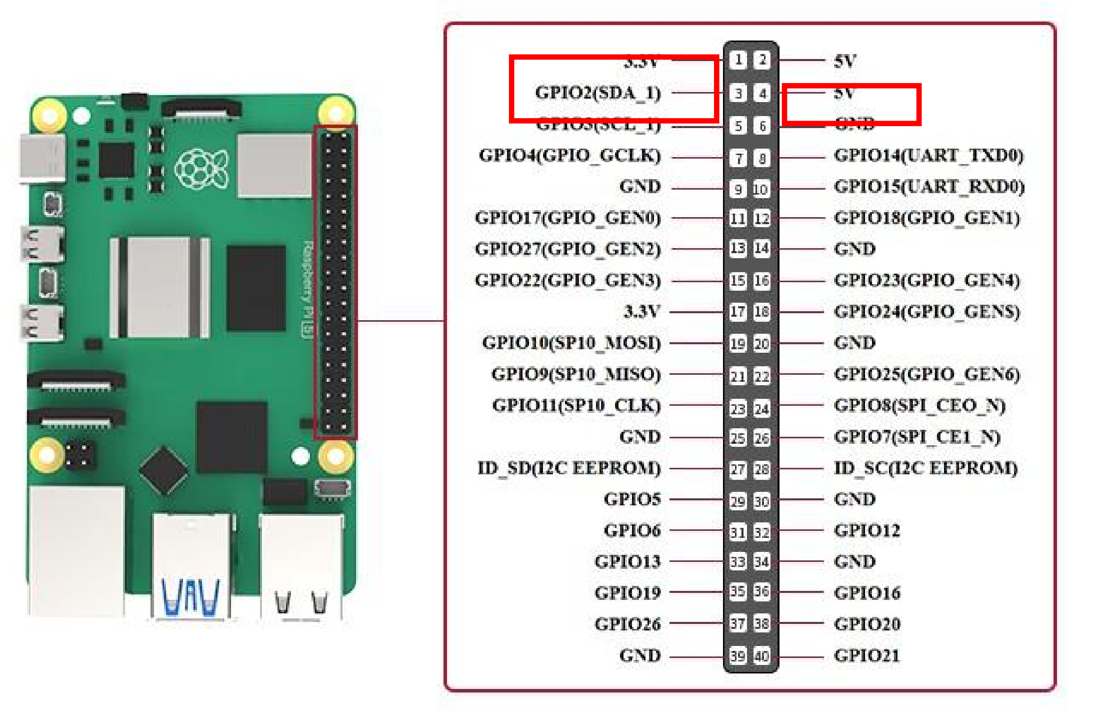
Upon the ACK being generated, the slave pulls down the SDA voltage level and remains low while the SCL has a high voltage level.

The data transmission ends with a stop signal (P) to release the I2C. However, the master can also generate repetitive start signals to operate another slave without sending an end signal.

All SDA signal changes must occur when the clock wire SCL is at a low voltage level, except the start and end signals.

2. Hardware Wiring

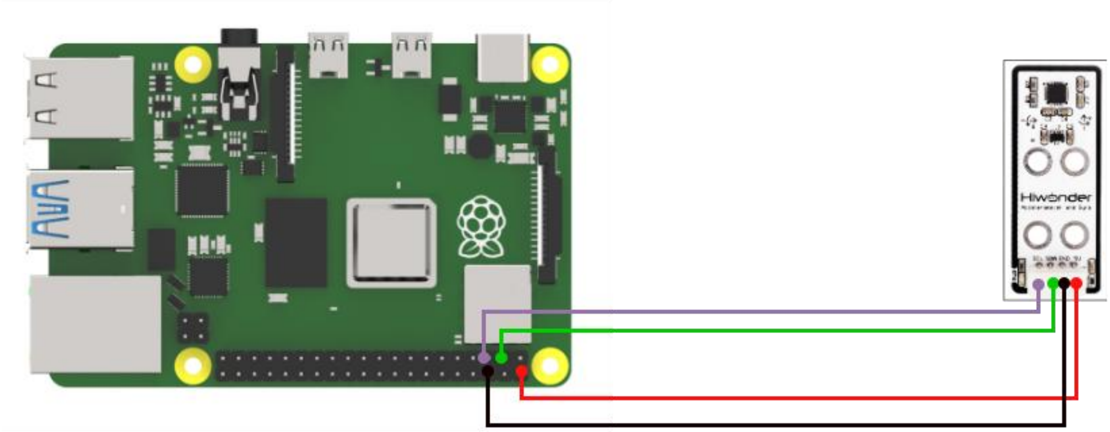
According to the Raspberry Pi 5 pin diagram, you will use the following pins:



Use an accelerometer sensor with an I2C interface as an example. (If you have a different module with the I2C interface, you can also refer to this wiring method.) Connect the accelerometer sensor to Raspberry Pi 5 with female-to-female DuPont wires, as shown in the diagram below:

Note: The Raspberry Pi 5 is powered by 5V 3A. Therefore, there is no need to connect VCC.

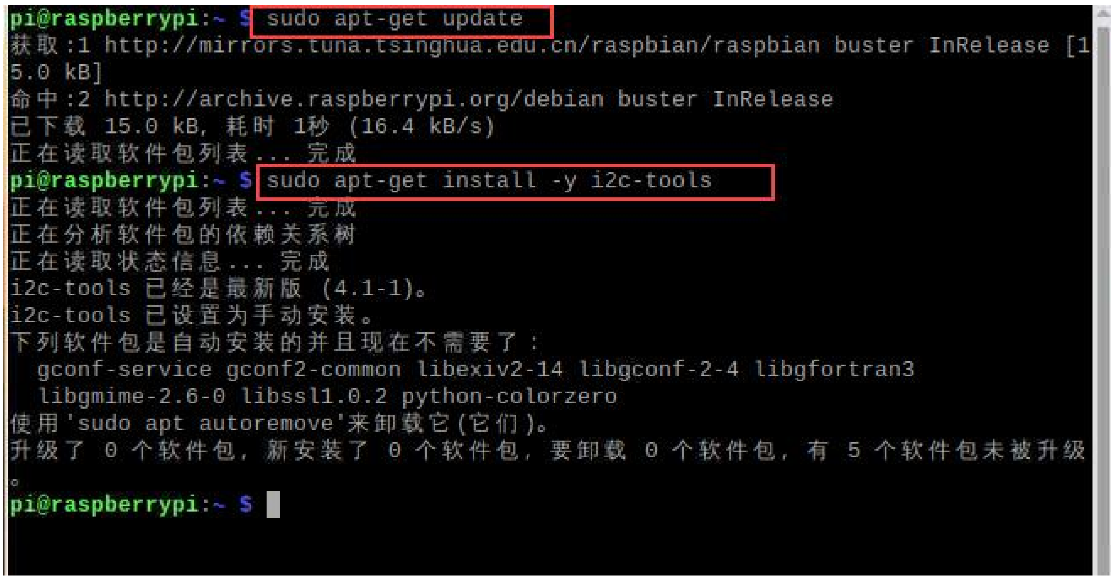
Pin3 on the Raspberry Pi 5（SDA） <--> Accelerometer (SDA)
Pin5 on the Raspberry Pi 5（SCL） <--> Accelerometer (SCL)
Pin6 on the Raspberry Pi 5（GND） <--> Accelerometer (GND)



3. I2C Library File Installation

Start Raspberry Pi 5 and press “Ctrl+Alt+T” to open the command line terminal. Then enter the two commands shown below in turn to install the I2C library.

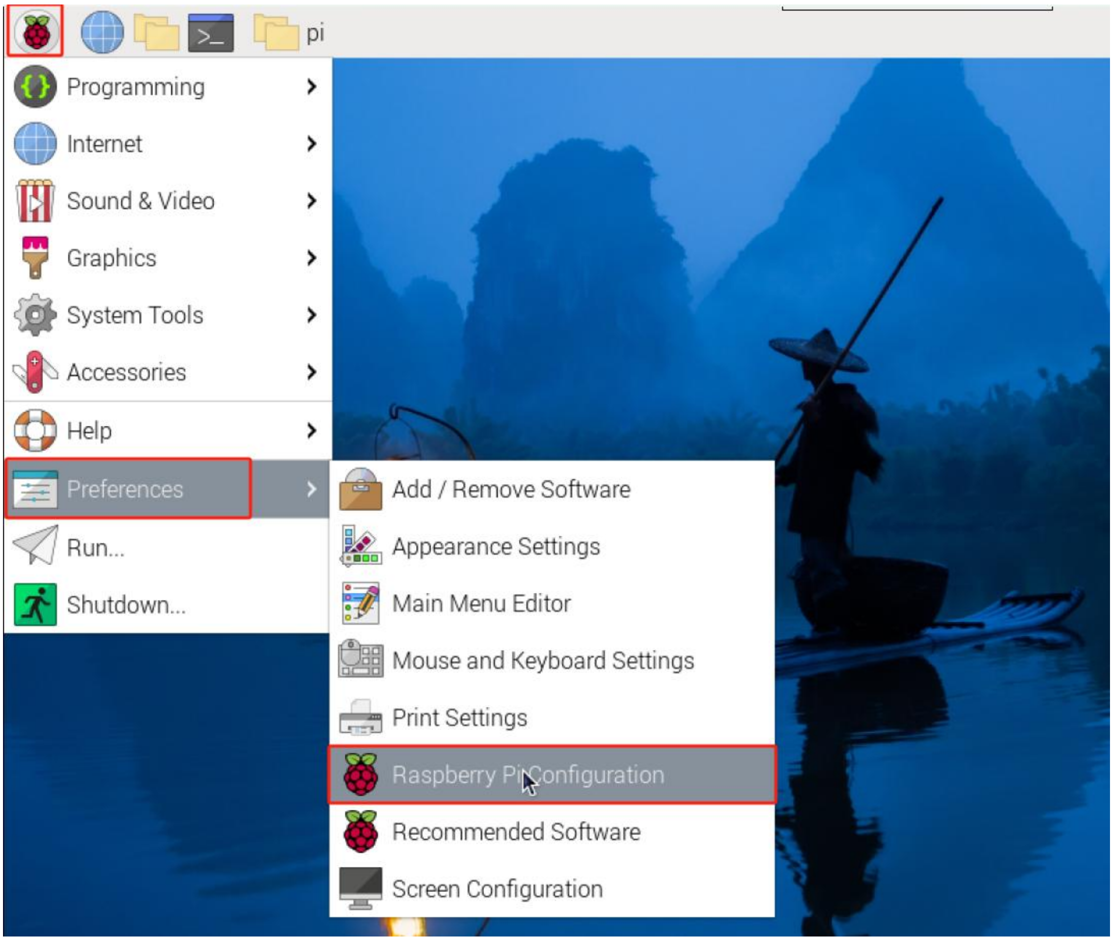
```
sudo apt-get update

sudo apt-get install -y i2c-tools
```



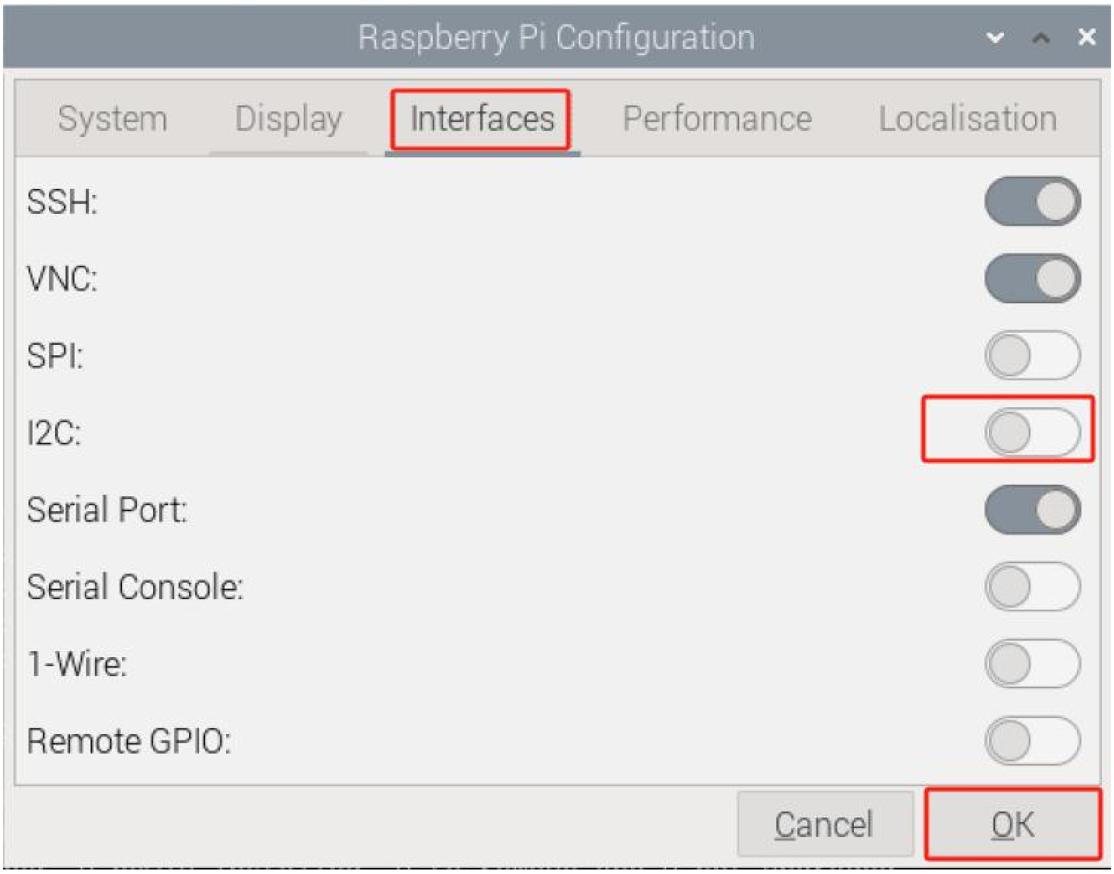
```
pi@raspberrypi:~ $ sudo apt-get update
获取:1 http://mirrors.tuna.tsinghua.edu.cn/raspbian/raspbian buster InRelease [15.0 kB]
命中:2 http://archive.raspberrypi.org/debian buster InRelease
已下载 15.0 kB, 耗时 1秒 (16.4 kB/s)
正在读取软件包列表... 完成
pi@raspberrypi:~ $ sudo apt-get install -y i2c-tools
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
i2c-tools 已经是最新版 (4.1-1)。
i2c-tools 已设置为手动安装。
下列软件包是自动安装的并且现在不需要了：
  gconf-service gconf2-common libexiv2-14 libgconf-2-4 libgfortran3
  libgmime-2.6-0 libssl1.0.2 python-colorzero
使用'sudo apt autoremove'来卸载它(它们)。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 5 个软件包未被升级。
pi@raspberrypi:~ $
```

4. I2C Interface Configuration

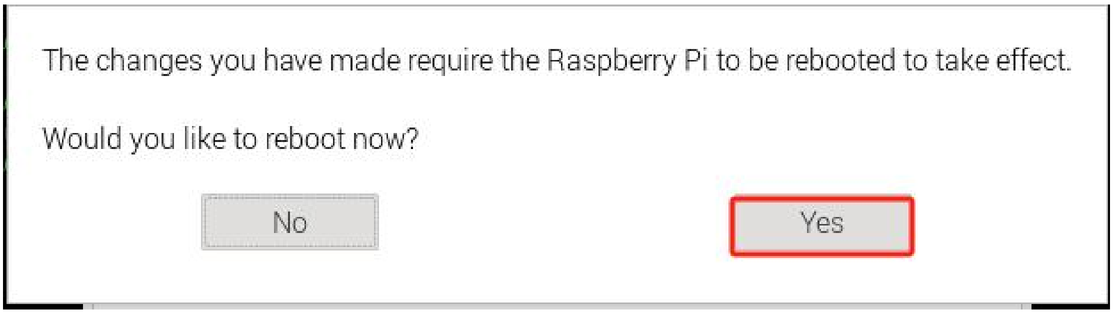
1) Configure the Raspberry Pi to enable the I2C interface and open I2C after booting up. Click on the the Raspberry Pi logo at the top left corner of the screen to select “Preferences”, and then “Raspberry Pi Configuration”.



2) Select “Interfaces” to start “Serial Port” and close “Serial Console”, then click “OK”.



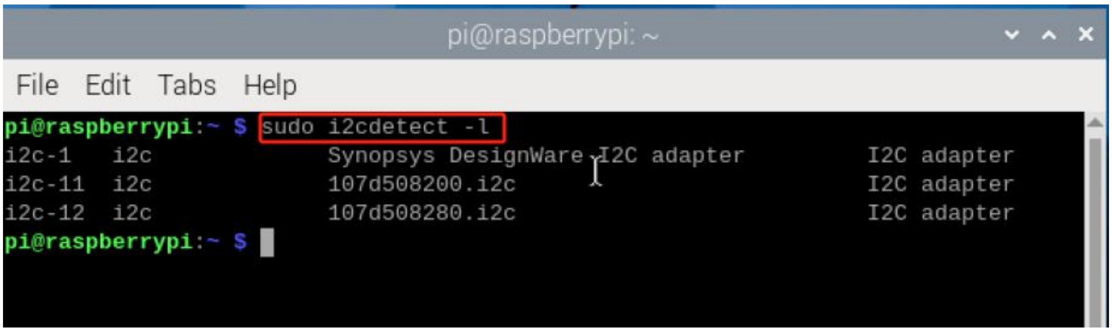
3) Click "Yes" to start the I2C.



5. I2C-Tools

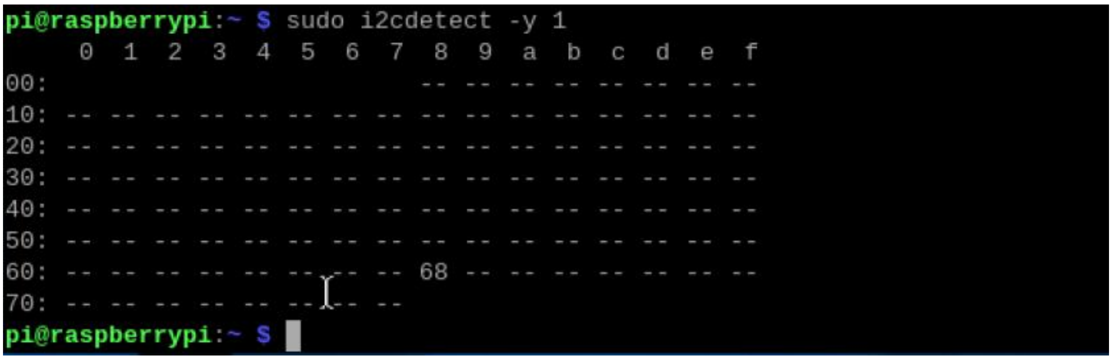
5.1 i2cdetect

Press "Ctrl+Alt+T" to open the command line terminal, and enter "sudo i2cdetect -l" to list all installed buses.



Enter the command “sudo i2cdetect -y l” to scan the device loaded on the bus. Such as the No.1 bus.

Where there is a device, there is a list of the device address. For example, there is a 0x68 device connected to the No.1 bus (refer to “2. Hardware Wiring”).



5.2 i2cdump

The i2cdump is used to view the register value of the device. Such as the used accelerometer MPU6050, the diagram of its register is as follows:

WHO_AM_I

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
75	117	-	WHO_AM_I[6:1]						-

Note: The register is used to identify the device identity. **Who_AM_I** is the first **six bit** in the **seven-bit I2C** address of MPU-60X0. The last bit is decided by **AD0**. The value of **AD0** pin is not related to the register.0

The default value of the register is **0x68**.

Retain bit **0** and **7**. (The hard-coding is **0**.)

In the command line terminal, enter the command “i2cdump -y 1 0x68” to check all the register values of No.1 bus 0x68.

```
pi@raspberrypi:~ $ i2cdump -y 1 0x68
No size specified (using byte-data access)
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: 81 01 81 c2 d3 0e ff e0 f8 fd 04 8e 28 7f XX XX    ??????.?????(?XX
10: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
20: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
30: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
40: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
50: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
60: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
70: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
80: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
90: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
a0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
b0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
c0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
d0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
e0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
f0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
pi@raspberrypi:~ $
```

5.3 i2cget

The “i2ccget” is used to check to single register values of the device and can be applied to single-byte registers. In the command line terminal, enter the command “i2cget -y 1 0x68 0x6b” to read the values of register 0x6b in the device with the address 0x68.

```
pi@raspberrypi:~ $ i2cget -y 1 0x68 0x6b
0x40
pi@raspberrypi:~ $
```

5.4 i2ctransfer

The “i2ctransfer” is used to read and write register values of a device, and can be applicable to double-byte registers. Generally, registers have an 8-bit address. And “i2cdump”, “i2cget”, and “i2cset” are all set to read the addresses with 8 bits. If an address exceeds 8 bits, the “i2ctransfer” should be used. In the command line terminal, enter the command “i2ctransfer -f -y 1 w2@0x68 0x01 0x6b r16” to read and write the data with 16 bytes starting from the register address 0x016b on the device with address 0x68 on No.1 bus.

```
pi@raspberrypi:~ $ i2ctransfer -f -y 1 w2@0x68 0x01 0x6b r16
0x81 0xc2 0xd3 0x0e 0xff 0xe0 0xf8 0xfd 0x04 0x8e 0x28 0x6f 0x6f 0x93 0x92 0x00
pi@raspberrypi:~ $
```