LAB MANUAL 5

Getting Started with DFRobot Hat & Raspberry PI

Disclaimer: The content is curated from online/offline resources and used for educational purpose only

DFRobot I/O Expansion Hat

This IO Expansion HAT from DFRobot is the perfect companion for your Raspberry Pi 4B/3B+! It leads out all of the IO ports on Raspberry Pi including digital port, analog port, PWM, IIC, UART, SPI, and IIS. Besides that, the HAT is totally compatible with DFRobot Gravity Series which frees users from complicated connection work, and enables them to just concentrate on their projects building.

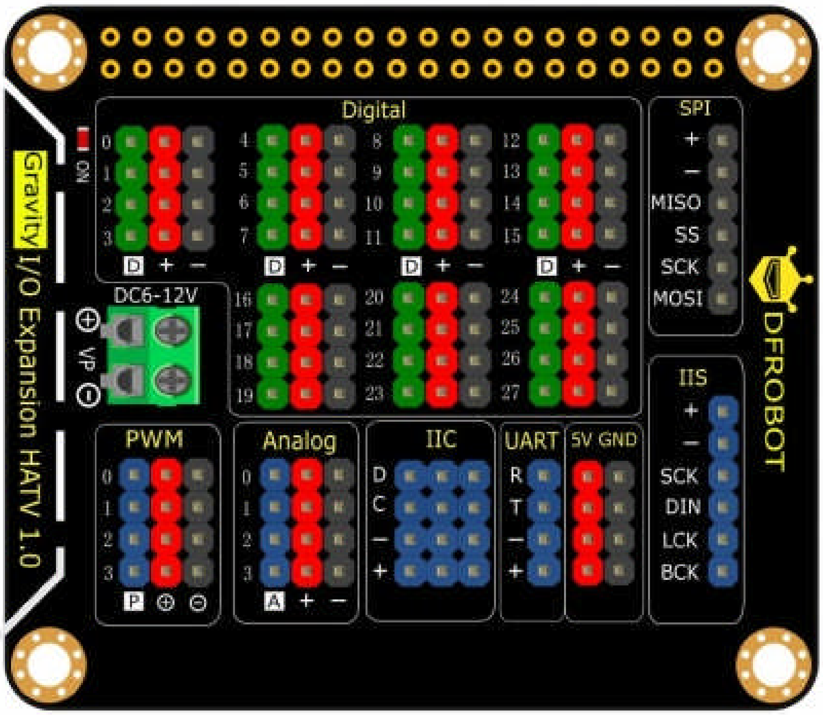
Raspberry Pi GPIO pins work with a maximum logic level of 3.3V. Besides the 3.3V power sensor and module, the product also supports:

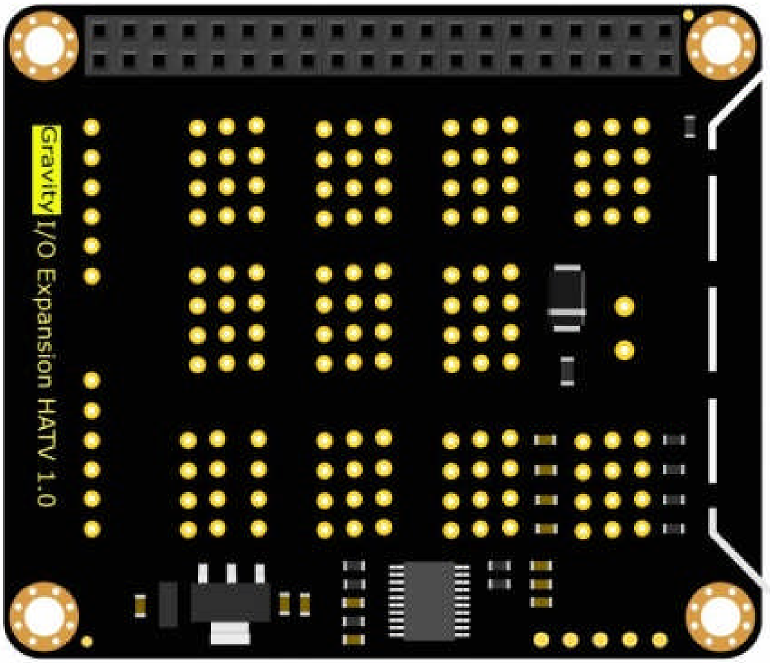
- Sensor and module with 5V power supply and 3.3V level
- PWM external power supply (6~12V)
- Controlling multiple servos

Specification

- Driver Main controller: STM32
- Operating Voltage: 5V
- PWM External Power Supply: 6-12V
- PWM Pin Voltage: 5V
- Sensor Interface Power Supply: 3.3V
- Communication Interface: 28 digital Ports, 4 groups of analog port, 3 groups of IIC port, 1 group of UART, 4 groups of 5V power interfaces, 1 group of SPI, 1 group of IIS port
- Device Address: 0x10
- Outline Dimension: 65×56mm/2.56×2.20”

Board Overview





Silkscreen	Label	Description
+	+	3.3V Positive
-	-	Negative
⊕	⊕	Binding post connect to external power positive
⊖	⊖	Binding post connect to external power negative
Digital	0-27	Raspberry Pi GPIO0-GPIO27
PWM	0-3	PWM signal output pin 0-3
Analog	0-3	Analog signal input pin 0-3
IIC	C	IIC port clock line
	D	IIC port data line
UART	T	UART Transmit port
	R	UART Receive port
5V	5V	5V positive
GND	GND	Negative
IIS	SCK	IIS serial clock line
	DIN	IIS serial data input
	LCK	IIS L/R channel selection line
	BCK	IIS system clock line
SPI	MISO	SPI data output line
	SS	SPI enable pin
	SCLK	SPI serial clock line
	MOSI	SPI data input line

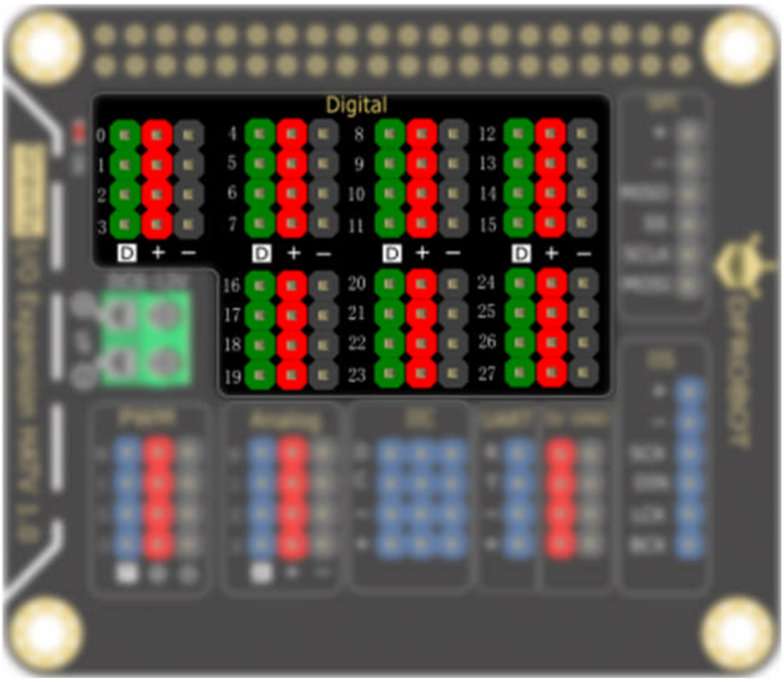
Note:

- The GPIO number in this board adopts BCM codes.
- When the VP port is not power by external power, the voltage of PWM ⊕ is 5V.

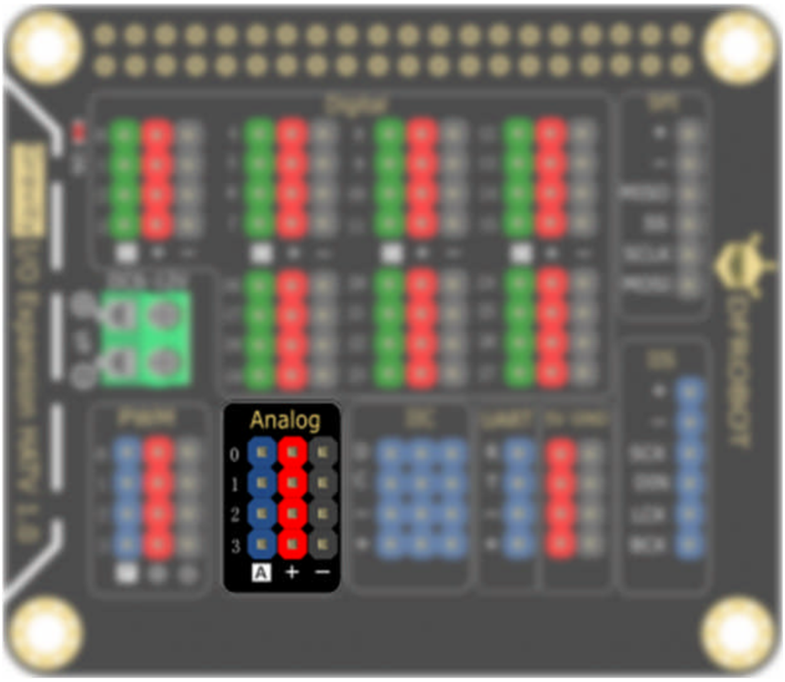
- When the VP port is powered by external power, the voltage of PWM ⊕ is equal to that of the VP external power (6-12V).

Port and Learning Guide

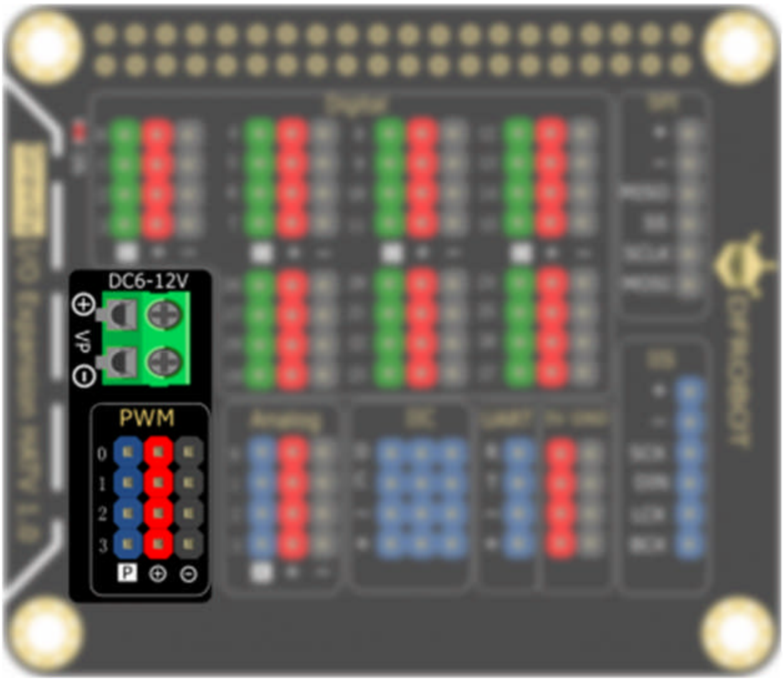
- Digital Port: IO expansion board offers 28 groups (D0-D27) of digital ports that are led out via Raspberry Pi ports GPIO0~GPIO27 (BCM codes).



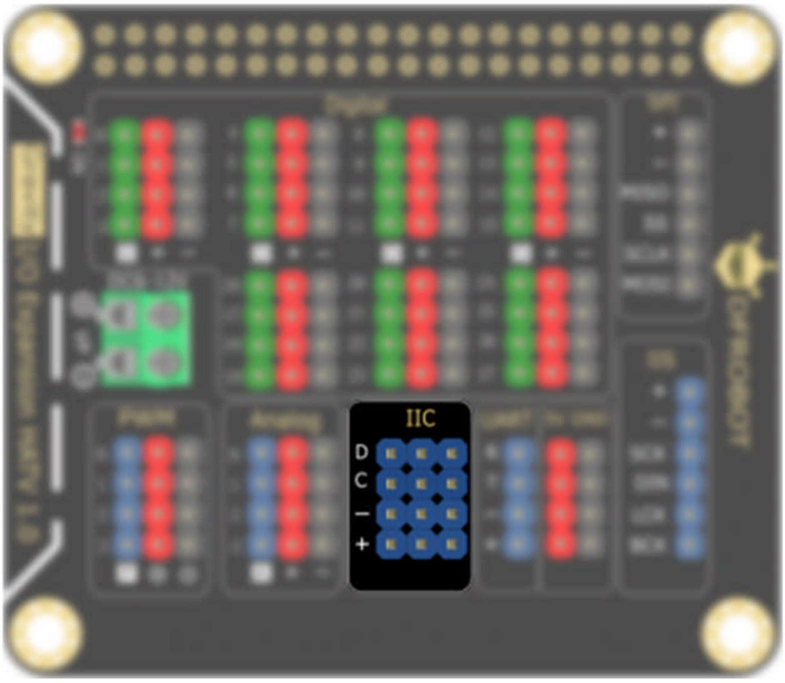
- Analog Port: IO expansion board has four groups of analog ports A0-A3. The board integrates on-board MCU STM32, and 12-bits ADC. The input voltage of analog sensor is 12-bit ADC. After the analog data is converted into digital data, it will be sent to Raspberry Pi via IIC communication.



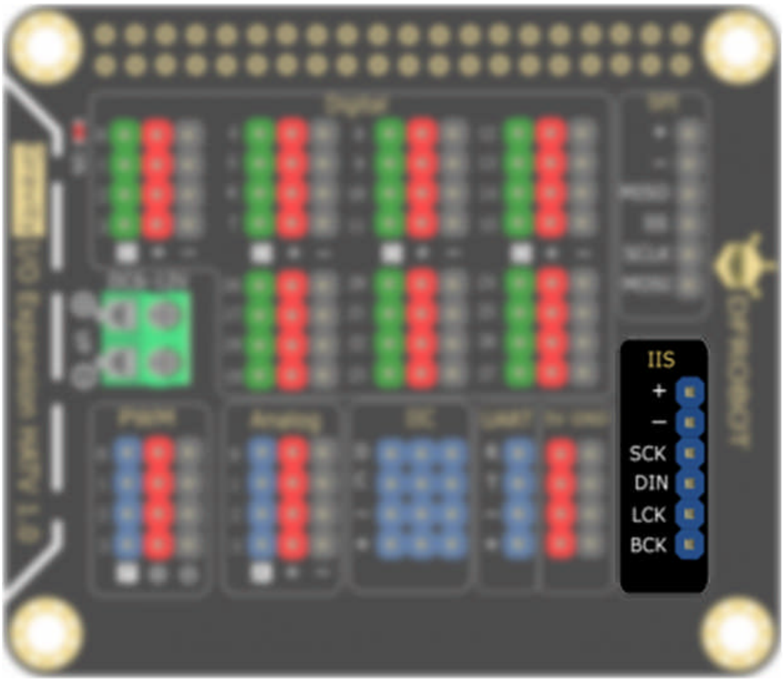
- PWM Port: IO expansion board provides four groups of PWM ports. Connect the STM32 to PWM. Raspberry Pi will send data to STM32 via IIC to control. VP port can supply 6-12V external power to PWM port. When not powered, the voltage of PWM ⊕ is 3.3V.



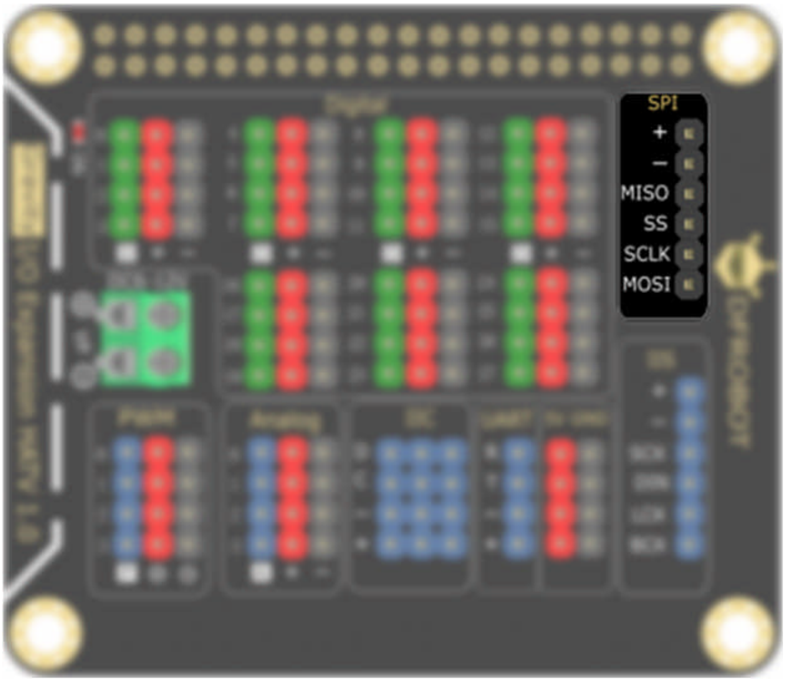
- IIC Port: IO expansion board has 3 groups of IIC ports that are led out via Raspberry Pi GPIO2（SDA.1）and GPIO3（SCL.1） （BCM code).



- IIS Port: there is 1 group of IIS port that is led out via Raspberry Pi GPIO ports: GIO21 (SCK), GPIO20(DIN), GPIO19(LCK), GPIO18(BCK).



- SPI Port: IO expansion board leads out a group of SPI ports via its GPIO port (BCM code): GPIO10(MOSI)and GPIO9(MISO), GPIO11(SCLK), GPIO8(SS).



Hardware

- Raspberry Pi Board x1
- IO Expansion HAT for Raspberry Pi x1
- HDMI Cable x1
- Display x1
- Keyboard and Mouse x1

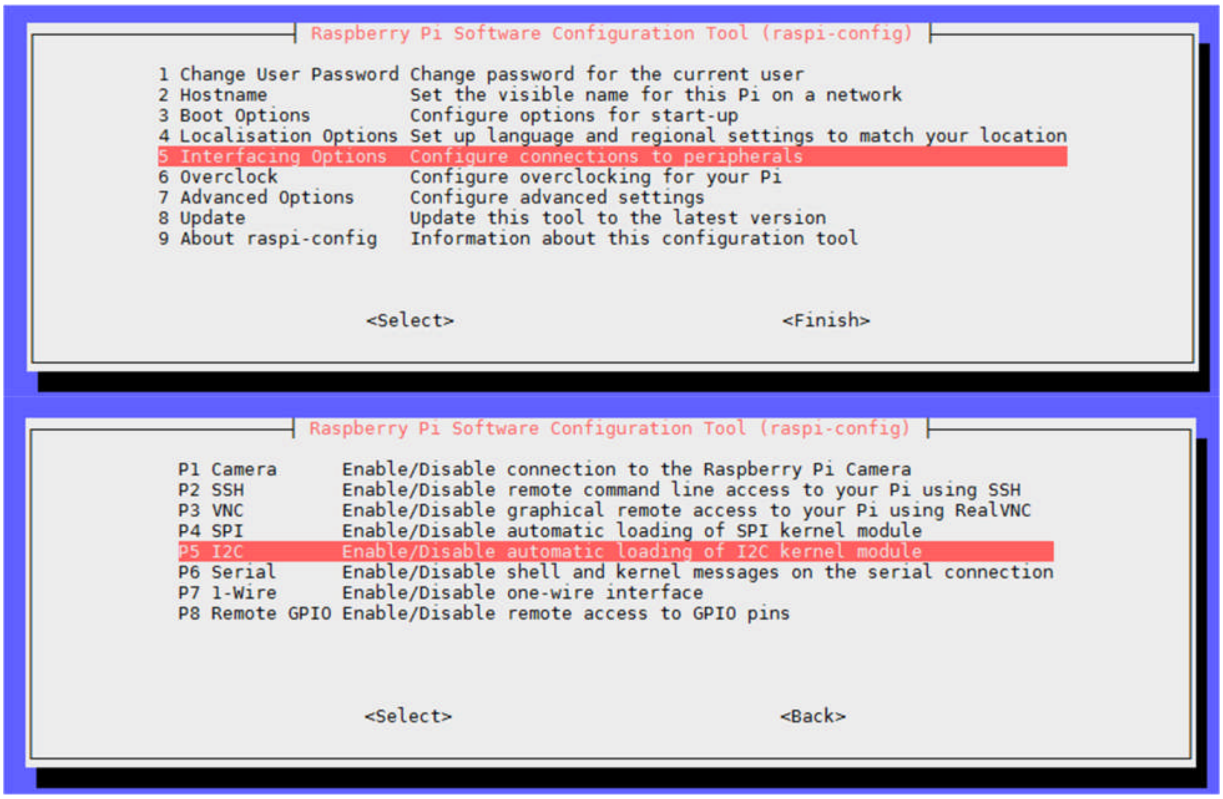
IIC Usage Operation and Program Execution Introduction

- Enable Raspberry Pi I2C interface. (Way to enable SPI is the same with IIC). Skip this step if it is already enabled.

Open Terminal and input the following commands, press "Enter":

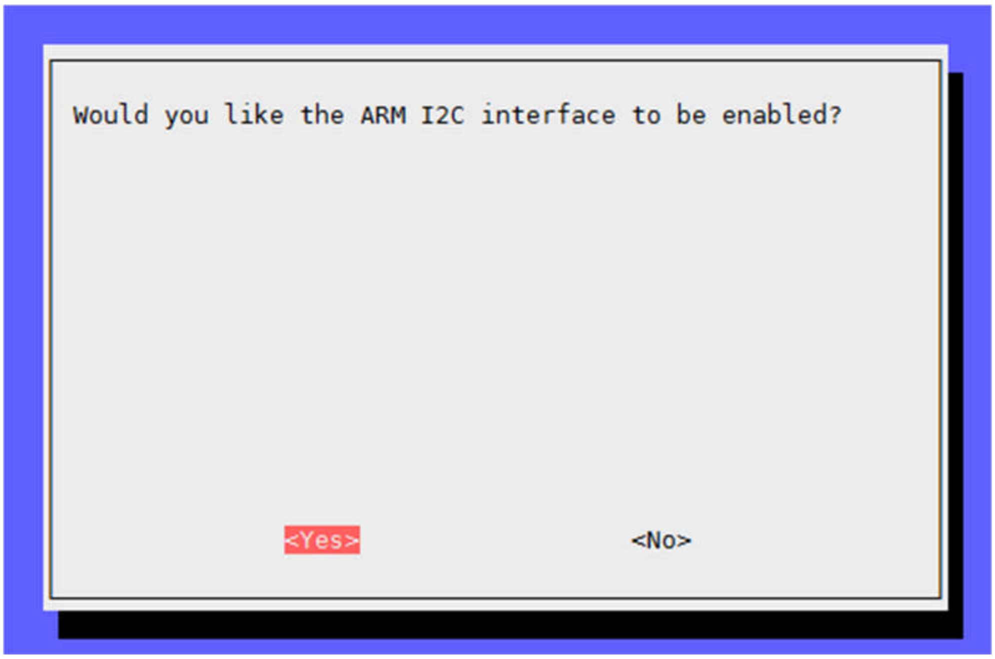
```
sudo raspi-config
```

Use the "Enter" key to select: [Interfacing Options] or ([Advanced Options])->[I2C]->[Yes]->[OK]->[Finish]:

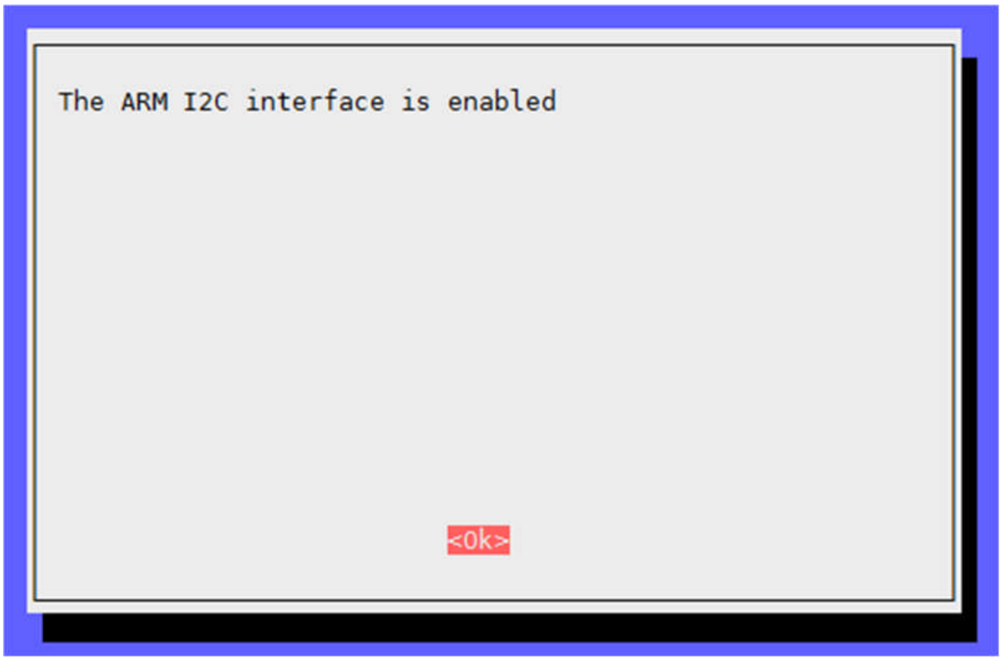


```
Raspberry Pi Software Configuration Tool (raspi-config)

1 Change User Password Change password for the current user
2 Hostname             Set the visible name for this Pi on a network
3 Boot Options         Configure options for start-up
4 Localisation Options Set up language and regional settings to match your location
5 Interfacing Options  Configure connections to peripherals
6 Overclock            Configure overclocking for your Pi
7 Advanced Options     Configure advanced settings
8 Update               Update this tool to the latest version
9 About raspi-config   Information about this configuration tool

                <Select>                            <Finish>
```

```
Raspberry Pi Software Configuration Tool (raspi-config)

P1 Camera      Enable/Disable connection to the Raspberry Pi Camera
P2 SSH         Enable/Disable remote command line access to your Pi using SSH
P3 VNC         Enable/Disable graphical remote access to your Pi using RealVNC
P4 SPI         Enable/Disable automatic loading of SPI kernel module
P5 I2C         Enable/Disable automatic loading of I2C kernel module
P6 Serial      Enable/Disable shell and kernel messages on the serial connection
P7 1-Wire      Enable/Disable one-wire interface
P8 Remote GPIO Enable/Disable remote access to GPIO pins

             <Select>                            <Back>
```



```
Would you like the ARM I2C interface to be enabled?



           <Yes>                      <No>
```



The ARM I2C interface is enabled

Test the Expansion board installation

- Open Thonny IDE and run below code

Code

```
import time

_PWM_CHAN_COUNT = 4
_ADC_CHAN_COUNT = 4

class DFRobot_Expansion_Board:

  _REG_SLAVE_ADDR = 0x00
  _REG_PID = 0x01
  _REG_VID = 0x02
  _REG_PWM_CONTROL = 0x03
  _REG_PWM_FREQ = 0x04
  _REG_PWM_DUTY1 = 0x06
  _REG_PWM_DUTY2 = 0x08
  _REG_PWM_DUTY3 = 0x0a
  _REG_PWM_DUTY4 = 0x0c
  _REG_ADC_CTRL = 0x0e
  _REG_ADC_VAL1 = 0x0f
  _REG_ADC_VAL2 = 0x11
  _REG_ADC_VAL3 = 0x13
  _REG_ADC_VAL4 = 0x15

  _REG_DEF_PID = 0xdf
  _REG_DEF_VID = 0x10

  ''' Enum board Analog channels '''
  A0 = 0x00
  A1 = 0x01
  A2 = 0x02
  A3 = 0x03

  ''' Board status '''
  STA_OK = 0x00
  STA_ERR = 0x01
  STA_ERR_DEVICE_NOT_DETECTED = 0x02
  STA_ERR_SOFT_VERSION = 0x03
  STA_ERR_PARAMETER = 0x04

  ''' last operate status, users can use this variable
to determine the result of a function call. '''
  last_operate_status = STA_OK

  ''' Global variables '''
  ALL = 0xffffffff

  def _write_bytes(self, reg, buf):
    pass

  def _read_bytes(self, reg, len):
    pass

  def __init__(self, addr):
    self._addr = addr
    self._is_pwm_enable = False

  def begin(self):
    '''
      @brief    Board begin
      @return   Board status
    '''
    pid = self._read_bytes(self._REG_PID, 1)
    vid = self._read_bytes(self._REG_VID, 1)
    if self.last_operate_status == self.STA_OK:
      if pid[0] != self._REG_DEF_PID:
        self.last_operate_status =
self.STA_ERR_DEVICE_NOT_DETECTED
      elif vid[0] != self._REG_DEF_VID:
        self.last_operate_status =
self.STA_ERR_SOFT_VERSION
      else:
        self.set_pwm_disable()
        self.set_pwm_duty(self.ALL, 0)
        self.set_adc_disable()
    return self.last_operate_status

  def set_addr(self, addr):
    '''
      @brief    Set board controler address, reboot
module to make it effective
      @param address: int    Address to set, range in 1
to 127
    '''
    if addr < 1 or addr > 127:
      self.last_operate_status = self.STA_ERR_PARAMETER
      return
    self._write_bytes(self._REG_SLAVE_ADDR, [addr])

  def _parse_id(self, limit, id):
    ld = []
    if isinstance(id, list) == False:
      id = id + 1
      ld.append(id)
    else:
      ld = [i + 1 for i in id]
    if ld == self.ALL:
      return range(1, limit + 1)
    for i in ld:
      if i < 1 or i > limit:
        self.last_operate_status =
self.STA_ERR_PARAMETER
        return []
    return ld

  def set_pwm_enable(self):
    '''
      @brief    Set pwm enable, pwm channel need
external power
    '''
    self._write_bytes(self._REG_PWM_CONTROL, [0x01])
    if self.last_operate_status == self.STA_OK:
      self._is_pwm_enable = True
    time.sleep(0.01)

  def set_pwm_disable(self):
    '''
      @brief    Set pwm disable
    '''
    self._write_bytes(self._REG_PWM_CONTROL, [0x00])
    if self.last_operate_status == self.STA_OK:
      self._is_pwm_enable = False
    time.sleep(0.01)

  def set_pwm_frequency(self, freq):
    '''
      @brief    Set pwm frequency
      @param freq: int    Frequency to set, in range 1
- 1000
    '''
    if freq < 1 or freq > 1000:
      self.last_operate_status = self.STA_ERR_PARAMETER
      return
    is_pwm_enable = self._is_pwm_enable
    self.set_pwm_disable()
    self._write_bytes(self._REG_PWM_FREQ, [freq >> 8,
freq & 0xff])
    time.sleep(0.01)
    if is_pwm_enable:
      self.set_pwm_enable()

  def set_pwm_duty(self, chan, duty):
    '''
      @brief    Set selected channel duty
      @param chan: list     One or more channels to 
set, items in range 1 to 4, or chan = self.ALL
      @param duty: float    Duty to set, in range 0.0 
to 100.0
    '''
    if duty < 0 or duty > 100:
      self.last_operate_status = self.STA_ERR_PARAMETER
      return
    for i in self._parse_id(_PWM_CHAN_COUNT, chan):
      self._write_bytes(self._REG_PWM_DUTY1 + (i - 1) * 
2, [int(duty), int((duty * 10) % 10)])

  def set_adc_enable(self):
    '''
      @brief    Set adc enable
    '''
    self._write_bytes(self._REG_ADC_CTRL, [0x01])

  def set_adc_disable(self):
    '''
      @brief    Set adc disable
    '''
    self._write_bytes(self._REG_ADC_CTRL, [0x00])

  def get_adc_value(self, chan):
    '''
      @brief    Get adc value
      @param chan: int    Channel to get, in range 1 to 
4, or self.ALL
      @return :list       List of value
    '''
    for i in self._parse_id(_ADC_CHAN_COUNT, chan):
      rslt = self._read_bytes(self._REG_ADC_VAL1 + (i - 
1) * 2, 2)
```

```
    return ((rslt[0] << 8) | rslt[1])

  def detecte(self):
    '''
      @brief    If you forget address you had set, use
this to detecte them, must have class instance
      @return   Board list conformed
    '''
    l = []
    back = self._addr
    for i in range(1, 127):
      self._addr = i
      if self.begin() == self.STA_OK:
        l.append(i)
    for i in range(0, len(l)):
      l[i] = hex(l[i])
    self._addr = back
    self.last_operate_status = self.STA_OK
    return l

class DFRobot_Epansion_Board_Digital_RGB_LED():

  def __init__(self, board):
    '''
      @param board: DFRobot_Expansion_Board   Board
instance to operate digital rgb led, test LED:
https://www.dfrobot.com/product-1829.html
                                              Warning:
LED must connect to pwm channel, otherwise may destory
Pi IO
    '''
    self._board = board
    self._chan_r = 0
    self._chan_g = 0
    self._chan_b = 0

  def begin(self, chan_r, chan_g, chan_b):
    '''
      @brief    Set digital rgb led color channel,
these parameters not repeat
```

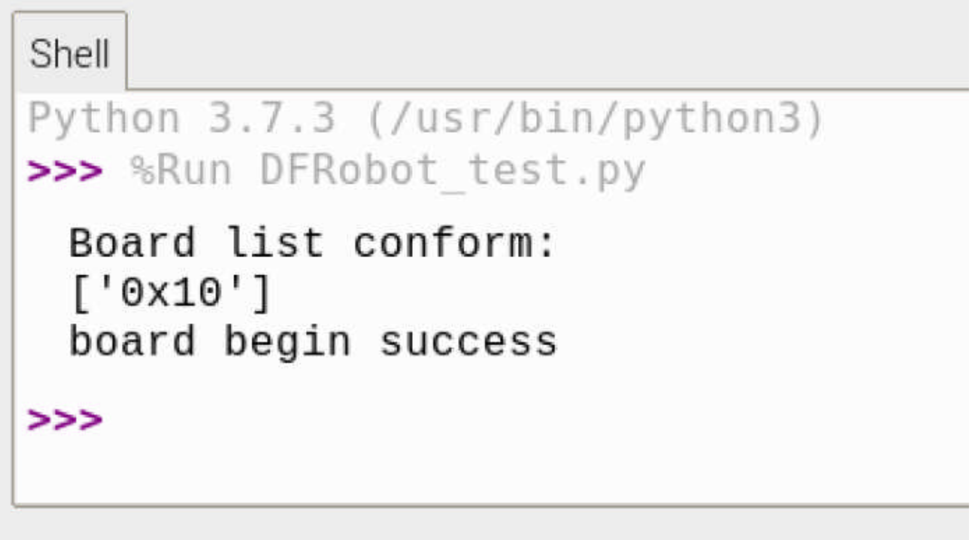
```
      @param chan_r: int    Set color red channel id,
in range 1 to 4
      @param chan_g: int    Set color green channel id,
in range 1 to 4
      @param chan_b: int    Set color blue channel id,
in range 1 to 4
    '''
    if chan_r == chan_g or chan_r == chan_b or chan_g
== chan_b:
      return
    if chan_r < _PWM_CHAN_COUNT and chan_g <
_PWM_CHAN_COUNT and chan_b < _PWM_CHAN_COUNT:
      self._chan_r = chan_r
      self._chan_g = chan_g
      self._chan_b = chan_b
      self._board.set_pwm_enable()
      self._board.set_pwm_frequency(1000)
      self._board.set_pwm_duty(self._board.ALL, 100)

  def color888(self, r, g, b):
    '''
      @brief    Set LED to true-color
      @param r: int   Color components red
      @param g: int   Color components green
      @param b: int   Color components blue
    '''
    self._board.set_pwm_duty([self._chan_r], 100 - (r &
0xff) * 100 // 255)
    self._board.set_pwm_duty([self._chan_g], 100 - (g &
0xff) * 100 // 255)
    self._board.set_pwm_duty([self._chan_b], 100 - (b &
0xff) * 100 // 255)

  def color24(self, color):
    '''
      @brief    Set LED to 24-bits color
      @param color: int   24-bits color
    '''
    color &= 0xffffff
    self.color888(color >> 16, (color >> 8) & 0xff,
color & 0xff)

  def color565(self, color):
    '''
      @brief    Set LED to 16-bits color
      @param color: int   16-bits color
    '''
    color &= 0xffff
    self.color888((color & 0xf800) >> 8, (color &
0x7e0) >> 3, (color & 0x1f) << 3)

class DFRobot_Expansion_Board_Servo():

  def __init__(self, board):
    '''
      @param board: DFRobot_Expansion_Board   Board
instance to operate servo, test servo:
https://www.dfrobot.com/product-255.html
                                              Warning:
servo must connect to pwm channel, otherwise may
destory Pi IO
    '''
    self._board = board

  def begin(self):
    '''
      @brief    Board servo begin
    '''
    self._board.set_pwm_enable()
    self._board.set_pwm_frequency(50)
    self._board.set_pwm_duty(self._board.ALL, 0)

  def move(self, id, angle):
    '''
      @brief    Servos move
      @param id: list     One or more servos to set,
items in range 1 to 4, or chan = self.ALL
      @param angle: int   Angle to move, in range 0 to
180
```

```
    '''
    if 0 <= angle <= 180:
      self._board.set_pwm_duty(id, (0.5 + (float(angle)
/ 90.0)) / 20 * 100)

import smbus

class
DFRobot_Expansion_Board_IIC(DFRobot_Expansion_Board):

  def __init__(self, bus_id, addr):
    '''
      @param bus_id: int   Which bus to operate
      @oaram addr: int     Board controler address
    '''
    self._bus = smbus.SMBus(bus_id)
    DFRobot_Expansion_Board.__init__(self, addr)

  def _write_bytes(self, reg, buf):
    self.last_operate_status =
self.STA_ERR_DEVICE_NOT_DETECTED
    try:
      self._bus.write_i2c_block_data(self._addr, reg,
buf)
      self.last_operate_status = self.STA_OK
    except:
      pass

  def _read_bytes(self, reg, len):
    self.last_operate_status =
self.STA_ERR_DEVICE_NOT_DETECTED
    try:
      rslt = self._bus.read_i2c_block_data(self._addr,
reg, len)
      self.last_operate_status = self.STA_OK
      return rslt
    except:
      return [0] * len
```

```
board = DFRobot_Expansion_Board_IIC(1, 0x10)    #
Select i2c bus 1, set address to 0x10

def board_detect():
  l = board.detecte()
  print("Board list conform:")
  print(l)

''' print last operate status, users can use this
variable to determine the result of a function call.
'''
def print_board_status():
  if board.last_operate_status == board.STA_OK:
    print("board status: everything ok")
  elif board.last_operate_status == board.STA_ERR:
    print("board status: unexpected error")
  elif board.last_operate_status ==
board.STA_ERR_DEVICE_NOT_DETECTED:
    print("board status: device not detected")
  elif board.last_operate_status ==
board.STA_ERR_PARAMETER:
    print("board status: parameter error")
  elif board.last_operate_status ==
board.STA_ERR_SOFT_VERSION:
    print("board status: unsupport board framware
version")

board_detect()    # If you forget address you had set,
use this to detected them, must have class instance


while board.begin() != board.STA_OK:    # Board begin
and check board status
    print_board_status()
    print("board begin faild")
    time.sleep(2)
print("board begin success")
```

If the board is properly installed it should confirm the board status

Output-



```
Python 3.7.3 (/usr/bin/python3)
>>> %Run DFRobot_test.py

  Board list conform:
  ['0x10']
  board begin success

>>>
```

Disclaimer: The content is curated from online/offline resources and used for educational purpose only