# 1. Using GPIO



## 1. Import GPIO libaray

The development board comes with a pre-installed GPIO Python library `Hobot.GPIO`. Users can import the GPIO library using the following command.

```
sunrise@ubuntu:~$ sudo python3
Python 3.8.10 (default, Mar 15 2022, 12:22:08)
Type "help", "copyright", "credits" or "license" for more information.
>>> import Hobot.GPIO as GPIO
Get board ID: 0x504
>>> GPIO.VERSION
'0.0.2'
>>> GPIO.model
'X3PI'
```

## 2. Set pin encoding mode

There are 4 modes for the pin coding of the development board.

- BOARD: physical pin number, corresponding to the silkscreen number of the development board.
- BCM: GPIO naming rules formulated according to Broadcom SoC.
- CVM: Use strings instead of numbers, corresponding to the signal names of the CVM/CVB connector.
- SOC: The corresponding number is the GPIO pin number of the X3M chip, which corresponds to the chip data sheet.

This article recommends that users use the `BOARD` coding mode, and the coding method is as follows.

```
GPIO.setmode(GPIO.BOARD)
# or
GPIO.setmode(GPIO.BCM)
# or
GPIO.setmode(GPIO.CVM)
# or
GPIO.setmode(GPIO.SOC)
```

Query encoding method.

```
mode = GPIO.getmode()
```

The program will output one of the results: `BOARD, BCM, CVM, SOC or None`.

## 3. Warning message

When running the code in the following cases, there will be warning log output, but it will not affect normal functions.

- The GPIO that the user is trying to use has been used in other applications;
- Before setting the mode and channel, try to call `GPIO.cleanup` to clean up the pin;

If you want to block the warning message, you can use the following command.

```
GPIO.setwarnings(False)
```

## 4. Pin configuration

Before using the GPIO pin, you need to configure it accordingly, as follows.

The way to set it as input is as follows.

```
GPIO.setup(channel, GPIO.IN)
```

The way to set it to output is as follows.

```
GPIO.setup(channel, GPIO.OUT)
```

You can also specify an initial value for the output channel, for example.

```
GPIO.setup(channel, GPIO.OUT, initial=GPIO.HIGH)
```

In addition, the tool supports setting multiple output channels at the same time, for example.

```
# set gpio(18,12,13) to output
channels = [18, 12, 13]
GPIO.setup(channels, GPIO.OUT)
```

## 5. Input operation

To read the value of a channel:

```
GPIO.input(channel)
```

The command return value is 0 or 1.

0 represents GPIO.LOW, 1 represents GPIO.HIGH.

# 6. Output operation

To set the output value of a channel.

```
GPIO.output(channel, state)
```

State can be GPIO.LOW or GPIO.HIGH.

# 7. Clean up pin occupation

Before exiting the program, you can clean up the channel.

```
GPIO.cleanup()
```

If you want to clean only a specific channel, use the following command.

```
# Clear a single channel
GPIO.cleanup(channel)
# Clear a group of channels
GPIO.cleanup( (channel1, channel2) )
GPIO.cleanup( [channel1, channel2] )
```

# 8. Check pin status

This function allows you to check the functionality of the corresponding GPIO channel.

```
GPIO.gpio_function(channel)
```

该函数返回 IN 或 OUT。

# 9. Edge detection and interrupt

An edge is a change in an electrical signal from low to high (rising edge) or from high to low (falling edge). This change can be regarded as an event that can be used to trigger a CPU interrupt signal.

The GPIO library provides three methods to detect input events.

## wait_for_edge() function

This function blocks the calling thread until the corresponding edge change is detected.

The function call is as follows.

```
GPIO.wait_for_edge(channel, GPIO.RISING)
```

The second parameter specifies the edge to detect, and the value range is GPIO.RISING, GPIO.FALLING, or GPIO.BOTH.

If you want to specify a waiting time, you can choose to set a timeout.

```
# Timeout in milliseconds
GPIO.wait_for_edge(channel, GPIO.RISING, timeout=500)
```

If the external signal changes within the timeout period, the function returns the detected channel number.

If a timeout occurs, the function returns None.

## event_detected() function

This function can be used to periodically check if any events have occurred since the last call.

This function can be set and called as follows.

```
# Set up rising edge detection on channel GPIO
GPIO.add_event_detect(channel, GPIO.RISING)
if GPIO.event_detected(channel):
    print("Rising edge event detected")
```

You can detect events of GPIO.RISING, GPIO.FALLING, or GPIO.BOTH.

## Run callback function when edge event is detected

This function can be used to register callback function, which runs in an independent processing thread.

The instructions are as follows.

```
# define callback function
def callback_fn(channel):
    print("Callback called from channel %s" % channel)

# enable rising detection
GPIO.add_event_detect(channel, GPIO.RISING, callback=callback_fn)
```

If necessary, you can also add multiple callbacks, as follows.

```
def callback_one(channel):
    print("First Callback")

def callback_two(channel):
    print("Second Callback")

GPIO.add_event_detect(channel, GPIO.RISING)
GPIO.add_event_callback(channel, callback_one)
GPIO.add_event_callback(channel, callback_two)
```

Since all callbacks run on the same thread, different callbacks are run sequentially, not simultaneously.

To prevent callbacks from being called multiple times by combining multiple events into one, you can optionally set a debounce time.

```
# bouncetime unit is ms
GPIO.add_event_detect(channel, GPIO.RISING, callback=callback_fn,
bouncetime=200)
```

### Disable interrupts

If edge detection is no longer needed, it can be removed, as follows.

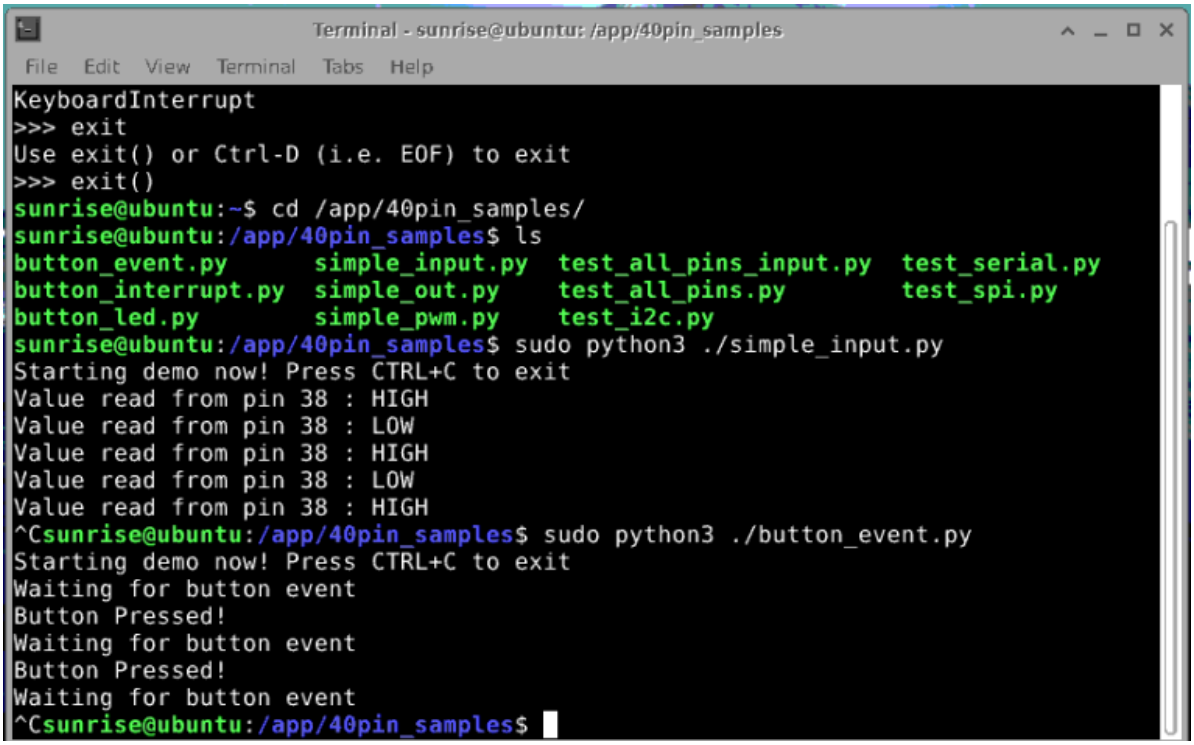
```
GPIO.remove_event_detect(channel)
```

## 10. Test routine

The main test routines are provided in the `/app/40pin_samples/` directory.

| Test routine name | Explanation |
| --- | --- |
| simple_out.py | Single pin `output` test |
| simple_input.py | Single pin `input` test |
| button_led.py | One pin is used as a key input, and one pin is used as an output to control the LED |
| test_all_pins_input.py | `Input test` code for all pins |
| test_all_pins.py | `Output test` code for all pins |
| button_event.py | Capture the rising and falling edge events of the pin |
| button_interrupt.py | Interrupt mode to handle rising and falling edge events of the pin |

As shown in the figure, take running simple_input.py and button_event.py as an example.




- GPIO is set to `output mode`, and the output level is switched every 1 second. It can be used to control the LED light to turn on and off in a cycle.

  Test code `simple_out.py`

```
#!/usr/bin/env python3

import Hobot.GPIO as GPIO
import time

# Define the GPIO channel used as 38
output_pin = 38 # BOARD Coding 38

def main():
    # Set the pin encoding mode to hardware encoding BOARD
    GPIO.setmode(GPIO.BOARD)
    # Set to output mode and initialize to high level
    GPIO.setup(output_pin, GPIO.OUT, initial=GPIO.HIGH)
    # Record the current pin status
    curr_value = GPIO.HIGH
    print("Starting demo now! Press CTRL+C to exit")
    try:
        # 1 second interval, cycle control LED light on and off
        while True:
            time.sleep(1)
            GPIO.output(output_pin, curr_value)
            curr_value ^= GPIO.HIGH
    finally:
        GPIO.cleanup()

if __name__=='__main__':
    main()
```

- Set GPIO to `input mode` and read the pin level value through busy polling. Test code `simple_input.py`.

```
#!/usr/bin/env python3

import Hobot.GPIO as GPIO
import time

# Define the GPIO channel used as 38
input_pin = 38 # BOARD coding 38

def main():
    prev_value = None

    # Set the pin encoding mode to hardware encoding BOARD
    GPIO.setmode(GPIO.BOARD)
    # Set as input mode
    GPIO.setup(input_pin, GPIO.IN)

    print("Starting demo now! Press CTRL+C to exit")
    try:
        while True:
            # Read pin level
            value = GPIO.input(input_pin)
            if value != prev_value:
                if value == GPIO.HIGH:
                    value_str = "HIGH"
                else:
                    value_str = "LOW"
                print("Value read from pin {} : {}".format(input_pin,
value_str))
                prev_value = value
            time.sleep(1)
    finally:
        GPIO.cleanup()

if __name__=='__main__':
    main()
```

- GPIO is set to input mode to capture the rising and falling edge events of pin 38. Test code 'button_ event. py' to detect the falling edge of pin 38 and control the output of pin 36.

```
#!/usr/bin/env python3

import Hobot.GPIO as GPIO
import time

#Define the GPIO channels used:
#Number 36 as output, can light up an LED
#Number 38 as input, can be connected to a button
led_pin = 36 # BOARD coding 36
but_pin = 38 # BOARD coding 38

# Disable warning messages
GPIO.setwarnings(False)

def main():
```

```
    # Set pin encoding mode to hardware encoding BOARD
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(led_pin, GPIO.OUT)  # LED pin set as output
    GPIO.setup(but_pin, GPIO.IN)  # button pin set as input

    # Initial state for LEDs:
    GPIO.output(led_pin, GPIO.LOW)

    print("Starting demo now! Press CTRL+C to exit")
    try:
        while True:
            print("Waiting for button event")
            GPIO.wait_for_edge(but_pin, GPIO.FALLING)

            # event received when button pressed
            print("Button Pressed!")
            GPIO.output(led_pin, GPIO.HIGH)
            time.sleep(1)
            GPIO.output(led_pin, GPIO.LOW)
    finally:
        GPIO.cleanup()  # cleanup all GPIOs

if __name__ == '__main__':
    main()
```

- Set GPIO to input mode, activate gpio interrupt function, respond to rising edge and falling edge events of pin, test code 'button_interrupt. py' to detect the falling edge of pin 38, and then control pin 36 to quickly switch high and low levels for 5 seconds:

```
#!/usr/bin/env python3
import sys
import signal
import Hobot.GPIO as GPIO
import time

def signal_handler(signal, frame):
    sys.exit(0)

#Define the GPIO channels used:
#As output on the 12th, it can light up an LED
#As output on the 13th, it can light up an LED
#Number 38 as input, can be connected to a button
led_pin_1 = 12 # BOARD encoding 12
led_pin_2 = 13 # BOARD encoding 13
but_pin = 38   # BOARD encoding 38

# Disable warning messages
GPIO.setwarnings(False)

#LED 2 flashes rapidly 5 times when the button is pressed
def blink(channel):
    print("Blink LED 2")
    for i in range(5):
        GPIO.output(led_pin_2, GPIO.HIGH)
        time.sleep(0.5)
```

```
        GPIO.output(led_pin_2, GPIO.LOW)
        time.sleep(0.5)

def main():
    # Pin Setup:
    GPIO.setmode(GPIO.BOARD)  # BOARD pin-numbering scheme
    GPIO.setup([led_pin_1, led_pin_2], GPIO.OUT)  # LED pins set as output
    GPIO.setup(but_pin, GPIO.IN)  # button pin set as input

    # Initial state for LEDs:
    GPIO.output(led_pin_1, GPIO.LOW)
    GPIO.output(led_pin_2, GPIO.LOW)

    # Register the blink function as an interrupt handling function for button
falling edge events
    GPIO.add_event_detect(but_pin, GPIO.FALLING, callback=blink, bouncetime=10)
    # Start testing, Led1 flashes slowly
    print("Starting demo now! Press CTRL+C to exit")
    try:
        while True:
            # blink LED 1 slowly
            GPIO.output(led_pin_1, GPIO.HIGH)
            time.sleep(2)
            GPIO.output(led_pin_1, GPIO.LOW)
            time.sleep(2)
    finally:
        GPIO.cleanup()  # cleanup all GPIOs

if __name__ == '__main__':
    signal.signal(signal.SIGINT, signal_handler)
    main()
```