使用GPIO

The development board is equipped with the GPIO Python library `Hobot.GPIO`. Users can import the GPIO library with 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'
```

Setting Pin Numbering Mode

The development board has 4 pin numbering modes:

- BOARD: Physical pin numbering, corresponding to the silk screen numbering on the development board.
- BCM: GPIO naming based on the Broadcom SoC.
- CVM: Use strings instead of numbers, corresponding to the signal names of the CVM/CVB connectors.
- SOC: GPIO pin numbering corresponding to the X3M chip, matching the chip's datasheet.

This article recommends using the `BOARD` pin numbering mode. The pin numbering can be set as follows:

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

To check the current pin numbering mode:

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

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

Warning Messages

The code will produce warning log outputs, but will not affect normal functionality in the following cases:

- The GPIO being attempted to use by the user is already being used by another application.
- `GPIO.cleanup` is called to clean up the pins before setting the mode and channels.

To suppress the warning messages, you can use the following command:

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

Pin Configuration

Before using GPIO pins, they need to be configured accordingly. Below are the specific configurations:

To set as input:

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

To set as output:

```
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)
```

Input Operation

To read the value of a channel, use:

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

The command returns either 0 or 1. 0 represents GPIO.LOW, and 1 represents GPIO.HIGH.

Output Operation

To set the output value of a channel, use:

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

Where state can be GPIO.LOW or GPIO.HIGH.

Clearing Pin Usage

Before exiting the program, it is recommended to perform a channel cleanup operation, use:

```
GPIO.cleanup()
```

If you only want to clean up specific channels, use:

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

Checking Pin State

This function allows you to check the function of the corresponding GPIO channel:

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

This function returns IN or OUT.

Edge Detection and Interrupts

Edge refers to the change in the electrical signal from low to high (rising edge) or high to low (falling edge), which can be considered as the occurrence of an event. This event can be used to trigger a CPU interrupt signal.

info

On the `RDK Ultra` platform, only a specific few pins on the `40 pin` header can be used as interrupt pins. They are numbered as **13**, **16**, **18**, **22**, **27**, **28**, **32**, **33**, **37** in `BOARD` mode.

Please refer to Pin Configuration and Definitions for pin definitions.

:::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, which can be `GPIO.RISING`, `GPIO.FALLING`, or `GPIO.BOTH`. If you want to specify a timeout, you can set the timeout parameter:

```
# timeout specified 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 an event has occurred since the last call. The function can be set and called as follows:

```
# set 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`.

Running a callback function when an edge event is detected

This feature can be used to register a callback function, which runs in a separate processing thread. Here is how to use it:

```
# 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 needed, you can also add multiple callbacks by following the same method:

```
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 callback functions run on the same thread, different callbacks are executed in order, not simultaneously.

To prevent multiple invocations of the callback function by merging multiple events into one event, you can choose to set 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, you can remove it as follows:

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

Test Cases

The main test cases are provided in the `/app/40pin_samples/` directory:

Test Case	Description
simple_out.py	Testing a single pin `output`
simple_input.py	Testing a single pin `input`
button_led.py	Using one pin as a button input and another as an LED output
test_all_pins_input.py	Code for `input testing` for all pins
test_all_pins.py	Code for `output testing` for all pins
button_event.py	Capturing rising and falling edge events on a pin
button_interrupt.py	Handling rising and falling edge events on a pin using interrupts

- Set GPIO to `output mode`, toggle the output level every 1 second, which can be used to control the LED cycle on and off. Test code `simple_out.py`:

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

import Hobot.GPIO as GPIO
import time

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

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

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

- GPIO is set to `input mode`, and the pin level is read through busy polling, using the test code `simple_input.py`:

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

import Hobot.GPIO as GPIO
import time
```

```
# Define the GPIO channel as 38
input_pin = 38 # BOARD code 38

def main():
    prev_value = None

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

    print("Starting demo now! Press CTRL+C to exit")
    try:
        while True:
            # Read the 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()
```

- Set GPIO to input mode, capture rising and falling edge events on the pin, test code `button_event.py`, detect falling edge of pin 38, and control the output of pin 31:

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

import RPi.GPIO as GPIO
import time

# Define the GPIO channels:
# Pin 31 as output to light up an LED
# Pin 38 as input for a button
led_pin = 31  # BOARD coding 31
but_pin = 38  # BOARD coding 38

# Disable warning messages
GPIO.setwarnings(False)

def main():
    # Set the pin coding mode to 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 pins to input mode, enable GPIO interrupt function, respond to rising and falling edge events on the pins, test code `button_interrupt.py`, detect falling edge on pin 38, and then control pin 36 to switch between high and low levels quickly 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 GPIO channels:
# Pin 12 as output, can light up an LED
# Pin 13 as output, can light up an LED
# Pin 38 as input, can connect a button
led_pin_1 = 12 # BOARD code 12
led_pin_2 = 13 # BOARD code 13
but_pin = 38   # BOARD code 38

# Disable warning messages
GPIO.setwarnings(False)

# Blink LED 2 fast for 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 blink function as the interrupt handler for falling edge events on
the button pin.GPIO.add_event_detect(but_pin, GPIO.FALLING, callback=blink,
bouncetime=10)
    # Start testing, Led1 slowly blink
    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()
```

Introduction to `hb_gpioinfo` Tool

The `hb_gpioinfo` tool is a GPIO helper tool adapted for the X5 platform. It is used to view the mapping relationship between `PinName` and `PinNum` on the current development board.

Components of `hb_gpioinfo`

The `hb_gpioinfo` tool consists of two parts:

1. **Driver**: Parses the `pinmux-gpio.dtsi` file and exports `PinNode` and `PinName` information to the `debugfs` system.
2. **Application**: Parses and displays the information in the terminal.

- **Driver Code Path:** `kernel/drivers/gpio/hobot_gpio_debug.c`

Example Usage of `hb_gpioinfo`

- **PinName:** Refers to the pin name on the SoC, consistent with the X5 SoC pin names in the schematic.
- **PinNode:** Refers to the `PinNode` information in the device tree.
- **PinNum:** Refers to the actual GPIO number corresponding to the pin on the X5.

```
root@ubuntu:~# hb_gpioinfo
```

```
gpiochip0 - 8 lines: @platform/31000000.gpio: @GPIOs 498-505
        [Number]                [Mode]  [Status]  [GpioName]       [PinName]
         [PinNode]           [PinNum]
         line  0:        unnamed input                            AON_GPIO0_PIN0
       aon_gpio_0              498
         line  1:        unnamed input                            AON_GPIO0_PIN1
       aon_gpio_1              499
         line  2:        unnamed input  active-low  GPIO Key Power AON_GPIO0_PIN2
       aon_gpio_2              500
         line  3:        unnamed input              interrupt      AON_GPIO0_PIN3
       aon_gpio_3              501
         line  4:        unnamed input                            AON_GPIO0_PIN4
       aon_gpio_4              502
         line  5:        unnamed input              id             AON_ENV_VDD
         aon_gpio_5              503
         line  6:        unnamed input              id             AON_ENV_CNN0
       aon_gpio_6              504
         line  7:        unnamed input                            AON_ENV_CNN1
       aon_gpio_7              505
gpiochip1 - 31 lines: @platform/35060000.gpio: @GPIOs 466-496
        [Number]                [Mode]  [Status]  [GpioName]       [PinName]
         [PinNode]           [PinNum]
         line  0:        unnamed input                            HSIO_ENET_MDC
         hsio_gpio0_0            466
         line  1:        unnamed input                            HSIO_ENET_MDIO
       hsio_gpio0_1            467
         line  2:        unnamed input
HSIO_ENET_TXD_0       hsio_gpio0_2            468
         line  3:        unnamed input
HSIO_ENET_TXD_1       hsio_gpio0_3            469
         line  4:        unnamed input
HSIO_ENET_TXD_2       hsio_gpio0_4            470
         line  5:        unnamed input
HSIO_ENET_TXD_3       hsio_gpio0_5            471
         line  6:        unnamed input                            HSIO_ENET_TXEN
       hsio_gpio0_6            472
         line  7:        unnamed input
HSIO_ENET_TX_CLK      hsio_gpio0_7            473
         line  8:        unnamed input
HSIO_ENET_RX_CLK      hsio_gpio0_8            474
         ....
```