Hiwonder

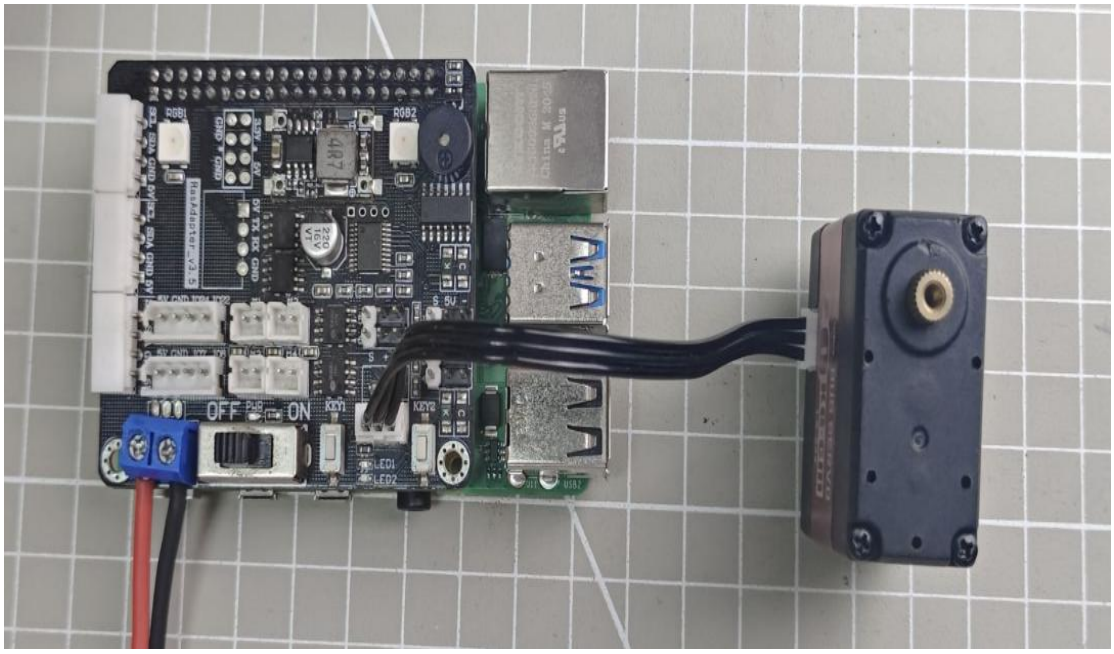
Raspberry Pi Development

V1.0

1. Getting Started

1.1 Wiring Instruction

This section employs a Raspberry Pi board and Raspberry Pi expansion board, powered by a 11.1V 6000mAh lithium battery. Connect the bus servo to any bus interface on the expansion board.



Note:

① When using our lithium battery, connect the lithium battery interface wire with red to positive (+) and black to negative (-) to the DC port.

② If the interface wire is not connected to the lithium battery, do not directly connect it to the battery interface wire to avoid a short circuit between positive and negative poles, which may cause a short circuit.

1.2 Environment Configuration

Install Nomchine software on PC. The software package is stored in "**2. Software -> Remote Connection Software**". For the detailed operations of Keil5, please refer to the relevant tutorials.

Drag the program and the SDK library files into the Raspberry Pi system image, using the desktop as an example.

Note: Ensure the library files are placed in the same directory as the program.

Open the command-line terminal, and input command to add execution permissions:

"**sudo chmod a+x Raspberry_BusServo_demo/**"。

```
ubuntu@ubuntu:~/Desktop$ sudo chmod a+x Raspberry_BusServo_demo/
ubuntu@ubuntu:~/Desktop$
```

2. Development Case

2.1 Case 1 Retrieve Bus Servo Information

In this case, the servo ID, position, temperature and other information will be displayed in the terminal monitor.

```
id:5
pos:886
dev:38
angle_range:(0, 1000)
voltage_range:(4500, 14000)
temperature_warn:85
temperature:34
vin:11905
lock:0
```

2.1.1 Run Program

1) Open a new terminal and the input the command "**cd Raspberry_BusServo_demo/**" to change to the directory where the program is located.

```
ubuntu@ubuntu:~/Desktop$ cd Raspberry_BusServo_demo/
ubuntu@ubuntu:~/Desktop/Raspberry_BusServo_demo$
```

2) Input the command to execute the program:

"**python3 BusServo_ReadStatus.py**"

```
hiwonder@hiwonder:~/Desktop/serial_servo$ python3 serial_servo_status.py
```

2.1.2 Performance

After the program runs, the servo status information will be printed in the terminal.

```
Pulse: 498
Temp:  56
Vin:   11524
id:    1
```

The specific meanings of the various pieces of information are as follows:

① **Pulse**: The current position of the servo, which is 498 here. For the conversion between position and angle, refer to "1. User Manual ->Bus Servo Instructions."

② **Temp**: The current temperature of the servo, which is 56°C here.

③ **Vin**: The current voltage of the servo, which is 11.524V here.

④ **id**: The current ID of the servo, which is 1 here.

2.1.3 Case Program Analysis

- **Import the Necessary Module**

```
import time
import Board
```

First, import the **Board** and **time** modules. By importing these modules,

we can use the functions and classes defined within the modules to control the servo.

- **Obtain and Print Servo Status Information**

```
Pulse = Board.getBusServoPulse(servo_id) # obtain position information of the servo
Temp = Board.getBusServoTemp(servo_id) # obtain temperature information of the servo
Vin = Board.getBusServoVin(servo_id) # obtain voltage information of the servo
```

After entering the loop function, various status information of the servo can be obtained by calling methods of the Board object. This status information includes the servo's position, temperature, voltage, and power-on state.

Once obtained, these parameters can be printed out.

```
print('Pulse: {}\nTemp:  {}\nVin:   {}\nid:   {}\n'.format(Pulse, Temp, Vin, servo_id))
time.sleep(0.5) # delay for easier check
```

2.2 Case 2 Set Servo ID

2.2.1 Run Program

```
ubuntu@ubuntu:~/Desktop/Raspberry_BusServo_demo$ sudo python3 set_BusServo_status.py
```

1) Open a new terminal and the input the command "**cd Raspberry_BusServo_demo/**" to change to the directory where the program is lcoated.

```
ubuntu@ubuntu:~/Desktop$ cd Raspberry_BusServo_demo/
ubuntu@ubuntu:~/Desktop/Raspberry_BusServo_demo$
```

2) Input the command to execute the program:

"sudo python3 set_BusServo_status.py"

```
hiwonder@hiwonder:~/Desktop/serial_servo$ python3 serial_servo_status.py
```

2.2.2 Performance

```
---------------------------
1 id
2 dev
3 save_dev
4 help
5 voltage_range
6 temperature_warn
7 lock
8 exit
---------------------------
*******current status**********
id:1
dev:-56
angle_range:(0, 1000)
voltage_range:(4500, 14000)
temperature_warn:85
lock:1
**********************************
input mode:
```

In the prompt section:

1. **id**: Set ID
2. **dev**: Set deviation
3. **save_dev**: Save deviation
4. **help**: Operation tips
5. **voltage_range**: Set voltage range
6. **temperature_warn**: Set temperature warning value
7. **lock**: Enter 1 for power-on, enter 0 for power-off
8. **exit**: Exit

In "input mode", enter 1, then enter the ID you want to set and press Enter.

Note: The ID range should be 0-253.

2.2.3 Case Program Analysis

- **Import the Necessary Module**

```
import time
import Board
```

First, import the **Board** and **time** modules. By importing these modules, we can use the functions and classes defined within the modules to control the servo.

- **Obtain and Print Servo Information**

```
def get_status():
    servo_id = Board.getBusServoID()
    dev = Board.getBusServoDeviation(servo_id)
    if dev > 125:
        dev = -(0xff - (dev - 1))
    angle_range = Board.getBusServoAngleLimit(servo_id)
    vin_range = Board.getBusServoVin(servo_id)
    temperature_warn = Board.getBusServoTempLimit(servo_id)
    load_state = Board.getBusServoLoadStatus(servo_id)
    print('*******current status**********')
    print('id:%s'%(str(servo_id).ljust(3)))
    print('dev:%s'%(str(dev).ljust(4)))
    print('angle_range:%s'%(str(angle_range).ljust(4)))
    print('voltage_range:%s'%(str(vin_range).ljust(5)))
    print('temperature_warn:%s'%(str(temperature_warn).ljust(4)))
    print('lock:%s'%(str(load_state).ljust(4)))
    print('********************************')
```

Print out the obtained servo status information. Use string formatting and the ljust() method for each print statement to align the output results.

- **Enter the Main Loop Function**

```
# 4 angle_range
while True:
    try:
        get_status()
        mode = int(input('input mode:'))
        if mode == 1:
            oldid = int(input('current id:'))
            newid = int(input('new id(0-253):'))
            Board.setBusServoID(oldid, newid)
        elif mode == 2:
            servo_id = int(input('servo id:'))
            dev = int(input('deviation(-125~125):'))
            if dev < 0:
                dev = 0xff + dev + 1
            Board.setBusServoDeviation(servo_id, dev)
        elif mode == 3:
            servo_id = int(input('servo id:'))
            Board.saveBusServoDeviation(servo_id)
        # elif mode == 4:
        #     servo_id = int(input('servo id:'))
        #     min_pos = int(input('min pos(0-1000):'))
        #     max_pos = int(input('max pos(0-1000):'))
        #     Board.setBusServoAngleLimit(servo_id, min_pos, max_pos)
        elif mode == 5:
            servo_id = int(input('servo id:'))
            min_vin = int(input('min vin(4500-14000):'))
            max_vin = int(input('max vin(4500-14000):'))
            Board.setBusServoVinLimit(servo_id, min_vin, max_vin)
        elif mode == 6:
            servo_id = int(input('servo id:'))
            max_temp = int(input('temperature(0-85):'))
            Board.setBusServoMaxTemp(servo_id, max_temp)
        elif mode == 7:
            servo_id = int(input('servo id:'))
            status = int(input('status 0 or 1:'))
            Board.unloadBusServo(servo_id, status)
```

```
        elif mode == 4:
            print('''
--------------------------
1 id
2 dev
3 save_dev
4 help
5 voltage_range
6 temperature_warn
7 lock
8 exit
--------------------------''')
        elif mode == 8:
            break
        else:
            print('error mode')
    except KeyboardInterrupt:
        break
```

In the main loop, the program first calls the **get_status()** function to print the current status of the servo. Then, through input mode, the user selects the operation they want to perform. Depending on the mode chosen, the program executes the corresponding code block to achieve servo settings and control.

- **Print Menu Options**

To print the menu options, different operation modes are provided for the user to choose from.

- **Setting Interrupt Flag**

```
except KeyboardInterrupt:
    break
```

Finally, set up the key binding so that when the terminal interface is printing, pressing Ctrl+C will catch the KeyboardInterrupt exception, break out of the loop, and terminate the program execution.

2.3 Case 3 Control Servo to Rotate

2.3.1 Run Program

1) Open a new terminal and the input the command "**cd Raspberry_BusServo_demo/**" to change to the directory where the program is located.

```
ubuntu@ubuntu:~/Desktop$ cd Raspberry_BusServo_demo/
ubuntu@ubuntu:~/Desktop/Raspberry_BusServo_demo$
```

2) Input the command to execute the program:

"**sudo python3 BusServo_Move.py**"

```
ubuntu@ubuntu:~/Desktop/Raspberry_BusServo_demo$ sudo python3 BusServo_Move.py
```

2.3.2 Performance

After the program runs, the servo will rotate back and forth between positions 200 and 800 with a 1-second interval.

2.3.3 Case Program Analysis

- **Import the Necessary Module**

```
import time
import Board
```

First, import the **Board** and **time** modules. By importing these modules, we can use the functions and classes defined within the modules to control the servo.

- **Enter the Main Loop Function**

```
while True:
    # paramete: parameter 1: servo ID; parameter 2: position; parameter 3: runtime
    # The range of rotation for the servo is 0-240 degrees, and the corresponding pulse width is 0-1000.

    Board.setBusServoPulse(1, 800, 1000) # Rotate servo 1 to position 800 in 1000ms
    time.sleep(0.5) # delay for 0.5s

    Board.setBusServoPulse(1, 200, 1000) # Rotate servo 1 to position 200 in 1000ms
    time.sleep(0.5) # delay for 0.5s
```

In the main loop, the program uses the **Board.setBusServoPulse()** method to set the position and running time of the servo. First, it sets the servo ID to 1, position to 800, and running time to 1000ms. Then, it calls **Board.setBusServoPulse()** to control the servo to move to the specified position.

Next, it uses the **time.sleep()** method to wait for the servo to complete its movement. To ensure the servo reaches the target position completely, an additional delay of 100ms is added. Then, it sets the servo position to 1000 and again calls **Board.setBusServoPulse()** to control the servo to move to the new position.

2.4 Case 4 Adjust Servo Speed

2.4.1 Run Program

1) Open a new terminal and the input the command "**cd Raspberry_BusServo_demo/**" to change to the directory where the program is located.

```
ubuntu@ubuntu:~/Desktop$ cd Raspberry_BusServo_demo/
ubuntu@ubuntu:~/Desktop/Raspberry_BusServo_demo$
```

2) Input the command to execute the program:

"sudo python3 BusServo_Speed.py"

```
ubuntu@ubuntu:~/Desktop/Raspberry_BusServo_demo$ sudo python3 BusServo_Speed.py
```

2.4.2 Performance

After the program runs, the servo behaves as follows:

① The servo starts from position 500.

② It rotates to position 1000 over 0.5 seconds.

③ It rotates to position 500 over 1.5 seconds.

④ It rotates to position 0 over 2.5 seconds.

⑤ It rotates to position 500 over 3.5 seconds.

2.4.3 Case Program Analysis

- **Import the Necessary Module**

```
import time
import Board
```

First, import the **Board** and **time** modules. By importing these modules, we can use the functions and classes defined within the modules to control the servo.

- **Obtain Servo ID**

```
servo_id = Board.getBusServoID()
```

Use the **Board.getBusServoID()** function to retrieve the servo's ID information and assign it to **servo_id**.

- **Control Servo Speed**

```
while True:
    # Parameters: parameter 1: servo ID; parameter 2: position; parameter 3: running time
    # The range of rotation for the servo is 0-240 degrees, and the corresponding pulse width is 0-1000.

    Board.setBusServoPulse(servo_id, 500, 1000)
    time.sleep(2)

    Board.setBusServoPulse(servo_id, 1000, 500)
    time.sleep(1)

    Board.setBusServoPulse(servo_id, 500, 1500)
    time.sleep(2)

    Board.setBusServoPulse(servo_id, 0, 2500)
    time.sleep(3)

    Board.setBusServoPulse(servo_id, 500, 3500)
    time.sleep(4)
```

First, move the servo to position 500. Set the running time to 1000ms, then call the Board.getBusServoPluse() method to control the servo to move to the specified position.

Next, wait for the servo to complete its movement using the time.sleep() method. Then, set the servo position to 1000 and again call the Board.getBusServoPluse() method to control the servo to move to the new

position. Similarly, wait for the servo to complete its movement using the time.sleep() method. Then, set the servo position to 500 and again call the Board.getBusServoPluse() method to control the servo to move to the new position.

Finally, set the servo position to 0 and again call the Board.getBusServoPluse() method to control the servo to move to the new position. After each servo movement, wait for the servo to complete its movement using the time.sleep() method.

- **Setting interrupt flag.**

```
except KeyboardInterrupt:
    break
```

Finally, set up the key press event so that when the terminal interface is printing, pressing Ctrl+C will catch the KeyboardInterrupt exception, break out of the loop, and terminate the program execution.

2.5 Case 5 Teaching Record Operations

2.5.1 Run Program

1) Open a new terminal and the input the command "**cd Raspberry_BusServo_demo/**" to change to the directory where the program is located.

```
ubuntu@ubuntu:~/Desktop$ cd Raspberry_BusServo_demo/
ubuntu@ubuntu:~/Desktop/Raspberry_BusServo_demo$
```

2) Input the command to execute the program:

"**sudo python3 BusServo_Record.py**"

```
ubuntu@ubuntu:~/Desktop/Raspberry_BusServo_demo$ sudo python3 BusServo_Record.py
```

2.5.2 Performance

After the program runs, return to the central position, power off the servo, and prompt: "**Start turning the servo**." At this point, move the servo arm to any angle and record the current position. Press key 0; the servo powers on,

returns to the neutral position, and then rotates to the angle recorded as pos.

2.5.3 Case Program Analysis

- **Import the Necessary Module**

```
import time
import Board
```

First, import the **Board** and **time** modules. By importing these modules, we can use the functions and classes defined within the modules to control the servo.

- **Obtain and Print Servo ID**

```
servo_id = Board.getBusServoID()
print('id:',servo_id)
```

Call **Board.getBusServoID()** to retrieve the servo's ID and print it out.

- **Control Servo Back to Central Position**

```
Board.setBusServoPulse(servo_id, 500, 1000) # control servo 1 to rotate to position 500 in 1000ms
time.sleep(1)
```

Call **Board.setBusServoPulse()** to move the servo back to position 500 and set the running time to 1000ms.

- **Obtain and Print the Current Position**

```
pos1 = Board.setBusServoPulse(servo_id, 500, 1000)
print('pos1:',pos1)
```

Call **Board.setBusServoPulse()** function to retrieve the servo position and assign it to pos1.

- **Setting the servo to power-off mode.**

```
# servo_control.unload_servo(servo_id,False)
Board.unloadBusServo(servo_id)

print('Can start turning the servo motor')
time.sleep(4)
```

Call **Board.unloadBusServo()** to power off the servo, and print the message "Start rotating the servo". At this point, you can manually move the servo by hand. Use **time.sleep()** to delay for 4 seconds.

- **Obtain Input Mode**

```
mode = int(input('input mode:'))
```

Use the **input()** function to obtain the mode input from the user and convert it to an integer type.

- **Perform operations based on the obtained mode**

```
if mode == 0:
    Board.getBusServoPulse(servo_id)
    pos = Board.getBusServoPulse(servo_id)
    print('pos:',pos)
    Board.setBusServoPulse(servo_id, 0, 1000)
    time.sleep(3)
    Board.setBusServoPulse(servo_id, pos, 1000)
    time.sleep(3)
```

If the mode is 0, first call get_servo_position() to retrieve the current position of the servo. Then set the servo position to 0 and wait for 3 seconds. Finally, restore the servo position to its previous position.