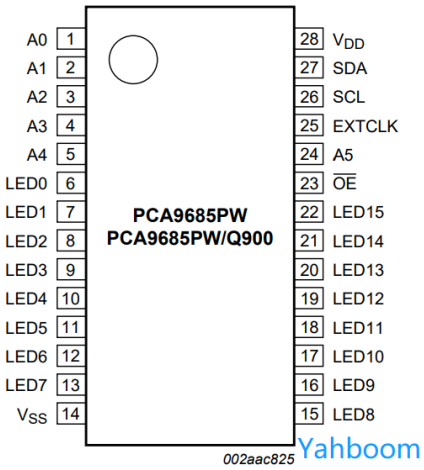
4.3 Using of PCA9685 driver

1. Introduction to PCA9685 Driver

For the Jetson.GPIO library file, there is no pin that supports PWM generation, which means that Jetson nano does not have the hardware to generate PWM capability, so we need to use other methods to generate PWM to complete the drive control, and just PCA9685 just solves this. One problem and saves its limited GPIO resources, greatly enhances its drive capability, PWM cycle and duty cycle are fully controllable, and can be done for basic control applications.



We use the PCA9685 in the TSSOP28 package shown above. The main parameters are as follows:

①I2C interface, support up to 16 PWM outputs, 12-bit resolution per channel (4096 levels)

②Built-in 25MHz crystal oscillator, can be connected to external crystal oscillator, can also be connected to external crystal oscillator, up to 50MHz

③ Support 2.3V-5.5V voltage, maximum withstand voltage 5.5V, logic level 3.3V

④With power-on reset, software reset and other functions

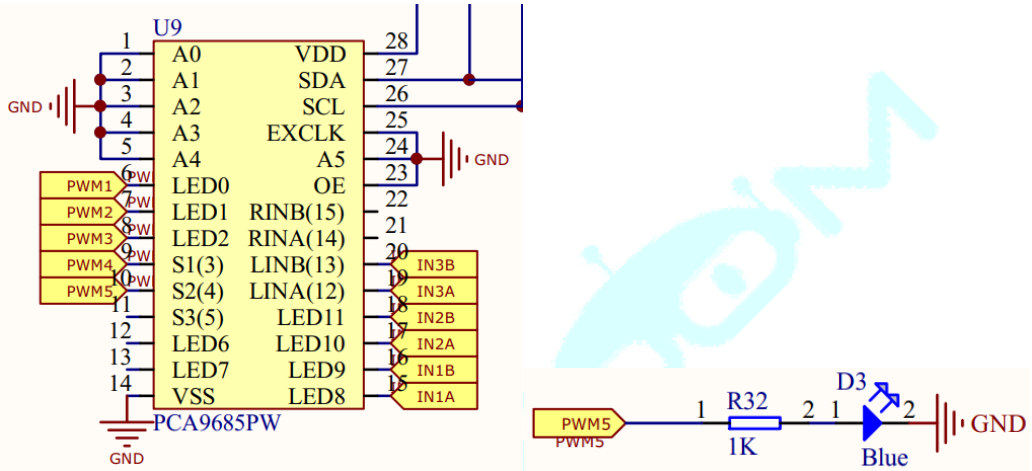
The underlying driver of the Jetbot robot car, in the initialization function in robot.py:

```
def __init__(self, *args, **kwargs):
    super(Robot, self).__init__(*args, **kwargs)
    self.motor_driver = Adafruit_MotorHAT(i2c_bus=self.i2c_bus)
    self.left_motor = Motor(self.motor_driver, channel=self.left_motor_channel,
alpha=self.left_motor_alpha)
    self.right_motor = Motor(self.motor_driver, channel=self.right_motor_chann
el, alpha=self.right_motor_alpha)
    self.vertical_motor = Motor(self.motor_driver, channel=self.vertical_motor_c
hannel, alpha=self.vertical_motor_alpha)
    self.bln = Motor(self.motor_driver, channel=self.bln_channel, alpha=self.bln
_alpha)
```

We generate an instance of controlling PCA9685 by calling the Adafruit_MotorHAT() method, which calls the Adafruit_MotorHAT.py driver.

The underlying driver for the PCA9685 is located at Adafruit_PWM_Servo_Driver.py. You can go to the underlying file for more information, including the slave address, register address, and so on. Then, Adafruit_MotorHAT.py calls Adafruit_PWM_Servo_Driver.py to complete the control.

2. Drive the onboard breathing light under the LOGO



As shown in the figure above is the drive circuit of the on board breathing lamp. We use code to control it.

First,we need to use our driver to create an instance of the control object:

Creating a robot instance

We use the Robot object to call our already packaged onboard breathing light library to drive the onboard breathing light.

```
from jetbot import Robot
import time
robot = Robot()
```

Then, we create a method to implement the breathing light function.
Principle: In the endless cycle, the duty ratio of PWM is 0 to 1 or 1 to 0 every 0.005s to achieve the effect of one call and one suction:

Creating a breathing light function method

```
def BLN_Onboard():
    global i , k
    i = k = 0
    while True:
        if k == 0:
            robot.set_bln(i)
            i += 0.01
            if( i >= 1 ):
                k = 1
            time.sleep(0.005)
        elif k == 1:
            robot.set_bln(i)
            i -=0.01
            if i <= 0 :
                k = 0
            time.sleep(0.005)
```

After the creation method is complete, we run the following code to complete the demo.

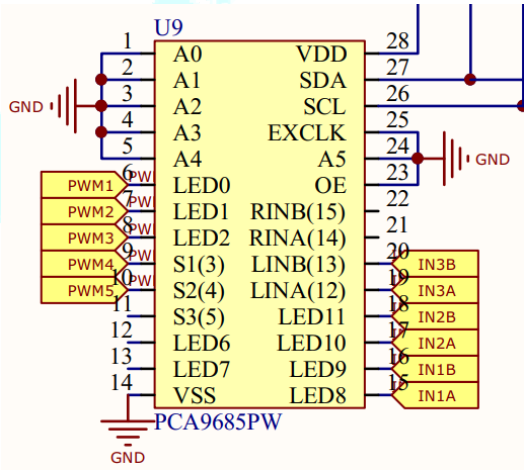
```
BLN_Onboard()
```

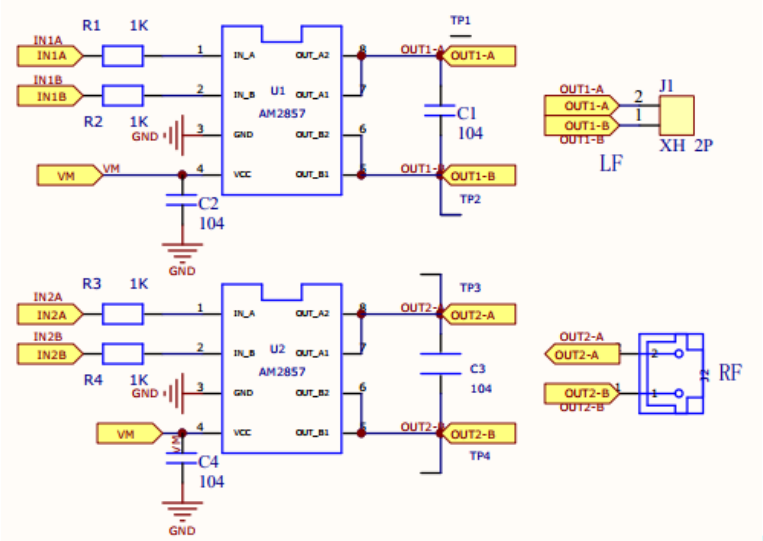
YAhboom

The corresponding complete source code is located at:
/home/jetbot/Notebook/4.Using of PCA9685 driver/1.Drive LOGO breathing light/Drive LOGO breathing light.ipynb

After the operation is complete, we can see that the blue LED breathing light under the Jetbot Logo in the middle of the expansion board is lit.

3. Drive 2-channle motor (control robot walking)





The bottom circuit that controls the two motors on the left and right sides of the Jetbot robot car, as shown in the figure above.

First, we need to create an instance of the control object, the code that controls the rotation of the left and right motors is shown below:

```
Control the left motor speed of Jetbot to 0.8 (Speed Range:0 - 1.0)
robot.left_motor.value = 0.8
Control the right motor speed of Jetbot to 0.8 (Speed Range:0 - 1.0)
robot.right_motor.value = 0.8
```

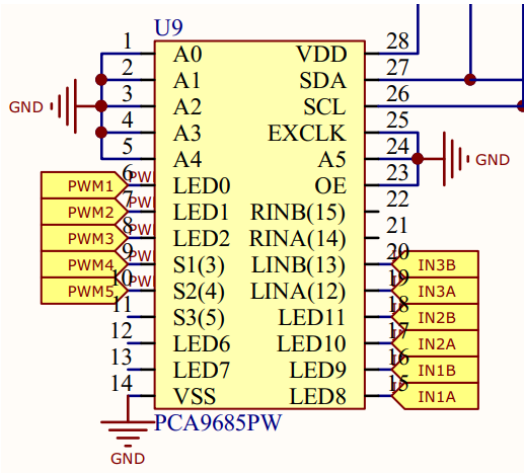
The value representing the speed of the control motor is in the range 0 – 1.0, which means that the duty cycle of the given PWM is 0 – 100%.

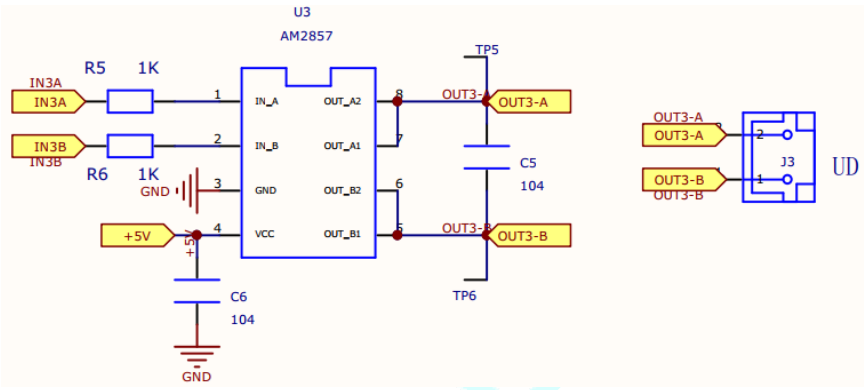
The code for turning off the motor is shown below, which sets the PWM duty cycle output to the motor to 0.

```
Stop left motor and right motor
robot.left_motor.value = 0
robot.right_motor.value = 0
```

The corresponding complete source code is located:
/home/jetbot/Notebook/Using of PCA9685 driver/2.Drive 2-channle wheel motor/Drive 2-channle wheel motor.ipynb

4.Drive 2-channle motor (control camera platform)



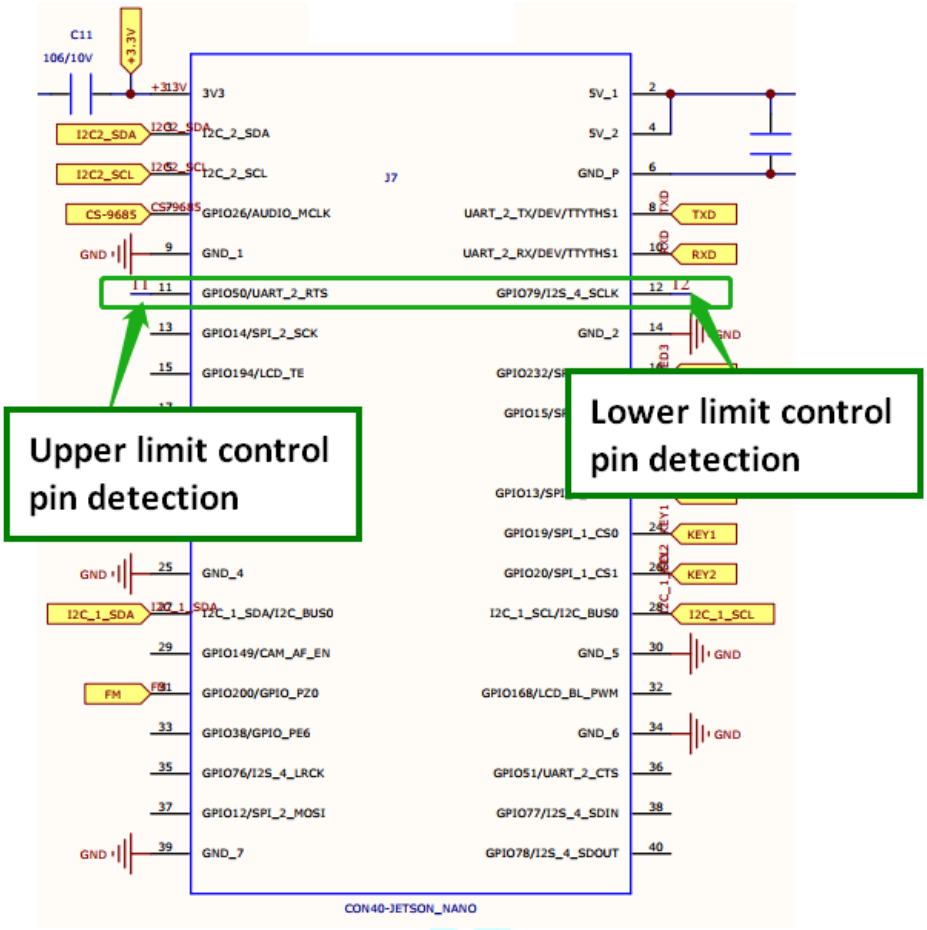


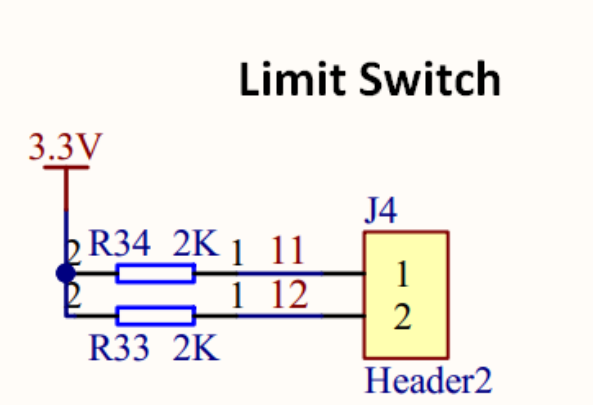
The circuit that controls camera platform, as shown in the figure above.

```
Yuntai rises 2S at a speed of 1

robot.up(1)
time.sleep(2.0)
robot.vertical_motors_stop()

Yuntai decline 2S at a speed of 1

robot.down(1)
time.sleep(2.0)
robot.vertical_motors_stop()
```

The corresponding complete source code is located:
/home/jetbot/Notebook/Using of PCA9685 driver/3.Drive 2-channle camera platform motor/Drive 2-channle camera platform motor.ipynb

5. Use the limit switch to control the upper and lower limits of the camera platform motor





As shown in the figure above, it is the circuit diagram of the limit switch.

We can see that there is also a pull-up circuit on the outside. When the limit switch is not turned on, it is high. When the limit switch is turned on, the potential of the point is low.

Because we need to detect whether the current upper and lower limits are reached in real time when we are operating the gimbal, we can judge whether the motor is running.

If the motor is running for a long time after reaching the upper and lower limits, it may cause heat to damage the motor, so we added the thread operation here.

Re-open a thread to monitor the switch status of the upper and lower limits of

the current PTZ, so that the motor can be turned off in time to protect.

Import python thread library, initialize PTZ lift variable

Create a vertical_motors_action pan/tilt global variable and import the thread package to monitor the pan/tilt status

```
import threading
global vertical_motors_action
vertical_motors_action = 0
```

PTZ lifting upper and lower limit detection method

```
def limit_detect():
    global vertical_motors_action
    while 1:
        if vertical_motors_action == 1:
            if GPIO.input(up_limit_pin) == 0:
                robot.vertical_motors_stop()
                vertical_motors_action = 0
                print('云台到顶')
        elif vertical_motors_action == 2:
            if GPIO.input(down_limit_pin) == 0:
                robot.vertical_motors_stop()
                vertical_motors_action = 0
                print('云台到底')
        time.sleep(0.5)
```

Create and open the thread that monitors the upper and lower limits of the PTZ

```
thread1 = threading.Thread(target=limit_detect)
# thread1.setDaemon(True)
thread1.start()
```

Yahboom

As shown in the code above. First, we need to create a new code that monitor the state of the upper and lower limits of the camera platform, and vertical_motors_action is the state of the current camera platform.
“0” : The current state of the camera platform is stopped.
“1” : The current state of the camera platform is rising state.
“2” : The current state of the camera platform is in a descending state.

When it is the motion state "1", we will detect the state of the upper limit switch to control whether the motion of the motor stops;
When it is the motion state "2", we will detect the state of the lower limit switch to control whether the motion of the motor stops.
When the state of motion is “0”, we do not process the control state of the camera platform;

Then, turn this thread on for detection by running the third cell code in the above figure.

Method of PTZ rise

```
vertical_motors_action = 1
if(GPIO.input(up_limit_pin)):
    robot.up(1)
    print('cameraup')
else:
    print('Top')
    robot.vertical_motors_stop()
    vertical_motors_action = 0
```

Method of PTZ decline

```
vertical_motors_action = 2
if(GPIO.input(down_limit_pin)):
    robot.down(1)
    print('cameradown')
else:
    print('Bottom')
    robot.vertical_motors_stop()
    vertical_motors_action = 0
```

Method of PTZ stop

```
robot.vertical_motors_stop()
vertical_motors_action = 0
print('camerastop')
```

The control of the ascending and descending motions of the gimbal is shown in the cell codes 1 and 2 above.
First, the current upper and lower limit states are identified by judging the state of the limit switch. Then, decide whether to move up or down.
If the pan/tilt does not reach the upper and lower limits and wants to maintain the height of the current pan/tilt camera, we can also run the third cell code to stop the pan/tilt at the current height position.

The corresponding complete source code is located at:
/home/jetbot/Notebook/4.PCA9685 driver use /4.Control camera U-L by limit switch/Control camera U-L by limit switch.ipynb