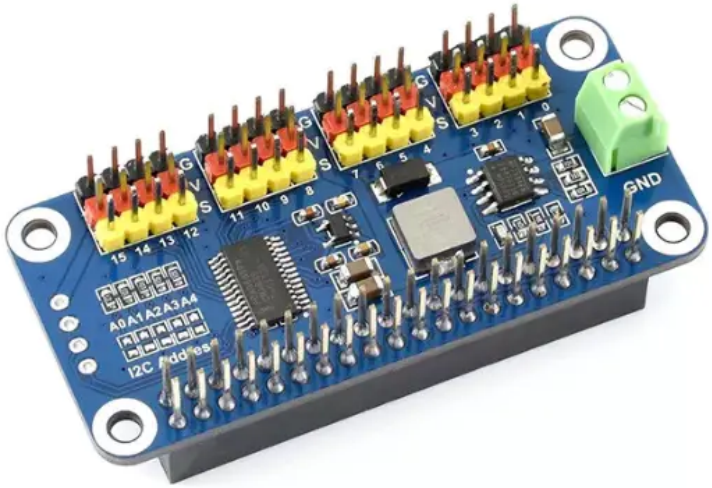
PCA9685：I2C转16路PWM，助力你的系统

1 基本介绍

1.1 该IC主要参数特征如下：

- I2C接口，支持高达16路PWM输出，每路12位分辨率(4096级)
- 内置25MHz晶振，可不连接外部晶振，也可以连接外部晶振，最大50MHz
- 支持2.3V-5.5V电压，最大耐压值5.5V,逻辑电平3.3V
- 具有上电复位，以及软件复位等功能



注：本教程侧重PCA9685的PWM输出，但PCA9685亦可用于WS2812等LED颜色控制等。

1.2 控制精度

假设舵机为50HZ的控制频率，脉宽为0.5ms~2.5ms，12位分辨率(4096级)，相关精度计算如下：

- PWM周期：

$$\frac{1}{50}s = 0.02s = 20ms = 20000us$$

- 时间分辨率：

$$\frac{20000}{2^{12}} = 4.88us$$

- 最大脉宽时间：

$$2.5ms - 0.5ms = 2ms = 2000us$$

- 最大脉宽时间可分成的份数：

$$\frac{2000us}{} = 410$$

- 0-180度的舵机，角度分辨率：

$$\frac{180^o}{410} = 0.439^o$$

2 硬件参数

2.1 封装及引脚排列

PCA9685有两种封装：TSSOP28, HVQFN28，其相应的引脚排列如下图所示：

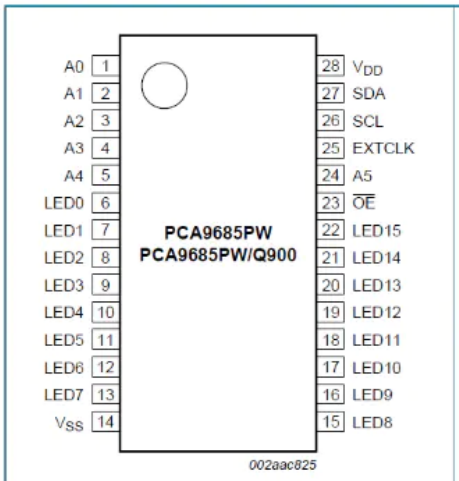


Fig 2. Pin configuration for TSSOP28

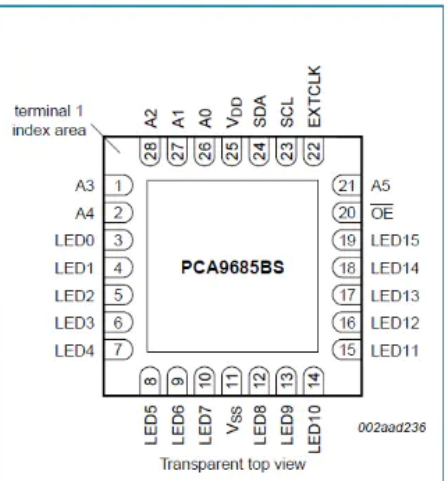


Fig 3. Pin configuration for HVQFN28

引脚功能描述如下图所示：

Table 2. Pin description

Symbol	Pin		Type	Description
	TSSOP28	HVQFN28		
A0	1	26	I	address input 0
A1	2	27	I	address input 1
A2	3	28	I	address input 2
A3	4	1	I	address input 3
A4	5	2	I	address input 4
LED0	6	3	O	LED driver 0
LED1	7	4	O	LED driver 1
LED2	8	5	O	LED driver 2
LED3	9	6	O	LED driver 3
LED4	10	7	O	LED driver 4
LED5	11	8	O	LED driver 5
LED6	12	9	O	LED driver 6
LED7	13	10	O	LED driver 7
V_{SS}	14	11[1]	power supply	supply ground
LED8	15	12	O	LED driver 8
LED9	16	13	O	LED driver 9
LED10	17	14	O	LED driver 10
LED11	18	15	O	LED driver 11

Table 2. Pin description ...continued

Symbol	Pin		Type	Description
	TSSOP28	HVQFN28		
LED12	19	16	O	LED driver 12
LED13	20	17	O	LED driver 13
LED14	21	18	O	LED driver 14
LED15	22	19	O	LED driver 15
$\overline{OE}$	23	20	I	active LOW output enable
A5	24	21	I	address input 5
EXTCLK	25	22	I	external clock input[2]
SCL	26	23	I	serial clock line

2.2 器件地址

PCA9685的器件地址是由引脚A0，A1，A2，A3，A4，A5共同决定，并且该引脚不可悬空，由于有6个引脚共同决定器件地址，因此，可以有64个器件地址，由于该IC上电便保留LED All Call address (E0h，1110 000)以及Software Reset address(06h，0000 0110)，实际仅有62个可用器件地址，因此，理论上，1个I2C接口可控制16*62=992路PWM，其引脚控制器件地址的示意图如下图所示：

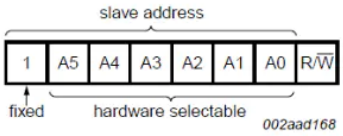


Fig 4. Slave address

默认情况下，若将A0-A5全部接地，则其器件地址为:0x40。

2.3 寄存器及其地址

默认情况下，上电复位后，寄存器地址的默认值均为0，寄存器地址及其用途见下图所示:

Table 3. Register summary

Register # (decimal)	Register # (hex)	D7	D6	D5	D4	D3	D2	D1	D0	Name	Type	Function
0	00	0	0	0	0	0	0	0	0	MODE1	read/write	Mode register 1
1	01	0	0	0	0	0	0	0	1	MODE2	read/write	Mode register 2
2	02	0	0	0	0	0	0	1	0	SUBADR1	read/write	I²C-bus subaddress 1
3	03	0	0	0	0	0	0	1	1	SUBADR2	read/write	I²C-bus subaddress 2
4	04	0	0	0	0	0	1	0	0	SUBADR3	read/write	I²C-bus subaddress 3
5	05	0	0	0	0	0	1	0	1	ALLCALLADR	read/write	LED All Call I²C-bus address
6	06	0	0	0	0	0	1	1	0	LED0_ON_L	read/write	LED0 output and brightness control byte 0
7	07	0	0	0	0	0	1	1	1	LED0_ON_H	read/write	LED0 output and brightness control byte 1
8	08	0	0	0	0	1	0	0	0	LED0_OFF_L	read/write	LED0 output and brightness control byte 2
9	09	0	0	0	0	1	0	0	1	LED0_OFF_H	read/write	LED0 output and brightness control byte 3

66	42	0	1	0	0	0	0	1	0	LED15_ON_L	read/write	LED15 output and brightness control byte 0
67	43	0	1	0	0	0	0	1	1	LED15_ON_H	read/write	LED15 output and brightness control byte 1
68	44	0	1	0	0	0	1	0	0	LED15_OFF_L	read/write	LED15 output and brightness control byte 2
69	45	0	1	0	0	0	1	0	1	LED15_OFF_H	read/write	LED15 output and brightness control byte 3
...	reserved for future use											
250	FA	1	1	1	1	1	0	1	0	ALL_LED_ON_L	write/read zero	load all the LEDn_ON registers, byte 0

图中节选的部分寄存器地址中，主要关心以下寄存器：

- 模式设置寄存器：MODE1，MODE2。
- 脉宽(占空比)设置寄存器：LED0_ON_L,LED0_ON_H,LED0_OFF_L,LED0_OFF_H.....LED15.....每一路PWM通道占用4个寄存器。
- 周期(频率)设置寄存器：PRE_SCALE。

  接下来介绍以上寄存器的使用及其注意事项。

2.4 模式设置寄存器

2.4.1 MODE1寄存器

首先介绍MODE1寄存器,如下图:

Table 6. MODE1 - Mode register 1 (address 00h) bit description

*Legend: * default value.*

Bit	Symbol	Access	Value	Description
7	RESTART	R		Shows state of RESTART logic. See Section 7.3.1.1 for detail.
		W		User writes logic 1 to this bit to clear it to logic 0. A user write of logic 0 will have no effect. See Section 7.3.1.1 for detail.
			0*	Restart disabled.
			1	Restart enabled.
6	EXTCLK	R/W		To use the EXTCLK pin, this bit must be set by the following sequence: 1. Set the SLEEP bit in MODE1. This turns off the internal oscillator. 2. Write logic 1s to both the SLEEP and EXTCLK bits in MODE1. The switch is now made. The external clock can be active during the switch because the SLEEP bit is set. This bit is a 'sticky bit', that is, it cannot be cleared by writing a logic 0 to it. The EXTCLK bit can **only** be cleared by a power cycle or software reset. EXTCLK range is DC to 50 MHz. $refresh\_rate = \frac{EXTCLK}{4096 \times (prescale + 1)}$
			0*	Use internal clock.
			1	Use EXTCLK pin clock.
5	AI	R/W	0*	Register Auto-Increment disabled[1].
			1	Register Auto-Increment enabled.
4	SLEEP	R/W	0	Normal mode[2].
			1*	Low power mode. Oscillator off[3][4].
3	SUB1	R/W	0*	PCA9685 does not respond to I²C-bus subaddress 1.
			1	PCA9685 responds to I²C-bus subaddress 1.
2	SUB2	R/W	0*	PCA9685 does not respond to I²C-bus subaddress 2.
			1	PCA9685 responds to I²C-bus subaddress 2.
1	SUB3	R/W	0*	PCA9685 does not respond to I²C-bus subaddress 3.
			1	PCA9685 responds to I²C-bus subaddress 3.
0	ALLCALL	R/W	0	PCA9685 does not respond to LED All Call I²C-bus address.
			1*	PCA9685 responds to LED All Call I²C-bus address.

在使用该寄存器的时候要注意：

- 如果未停止所有PWM输出就将其进入到睡眠模式，那么，所有输出通道在下一轮都将输出高电平。
- 睡眠后重新启动PWM的操作为:

To restart all of the previously active PWM channels with a few I^2C-bus cycles do the following steps:

1. Read MODE1 register.
2. Check that bit 7 (RESTART) is a logic 1. If it is, clear bit 4 (SLEEP). Allow time for oscillator to stabilize (500 μs).
3. Write logic 1 to bit 7 of MODE1 register. All PWM channels will restart and the RESTART bit will clear.

Remark: The SLEEP bit **must** be logic 0 for at least 500 μs before a logic 1 is written into

- 注意，在设置PWM频率(写PRESCALE寄存器)的时候，要先设置为Sleep模式，请参考后面源码部分。

2.4.2 MODE2寄存器

该寄存器的各位功能如下图所示:

Table 6. MODE2 - Mode register 2 (address 01h) bit description
*Legend: * default value.*

Bit	Symbol	Access	Value	Description
7 to 5	-	read only	000*	reserved
4	INVRT[1]	R/W	0*	Output logic state not inverted. Value to use when external driver used. Applicable when $\overline{OE}$ = 0.
			1	Output logic state inverted. Value to use when no external driver used. Applicable when $\overline{OE}$ = 0.
3	OCH	R/W	0*	Outputs change on STOP command[2].
			1	Outputs change on ACK[3].
2	OUTDRV[1]	R/W	0	The 16 LEDn outputs are configured with an open-drain structure.
			1*	The 16 LEDn outputs are configured with a totem pole structure.
1 to 0	OUTNE[1:0][4]	R/W	00*	When $\overline{OE}$ = 1 (output drivers not enabled), LEDn = 0.
			01	When $\overline{OE}$ = 1 (output drivers not enabled): LEDn = 1 when OUTDRV = 1 LEDn = high-impedance when OUTDRV = 0 (same as OUTNE[1:0] = 10)
			1X	When $\overline{OE}$ = 1 (output drivers not enabled), LEDn = high-impedance.

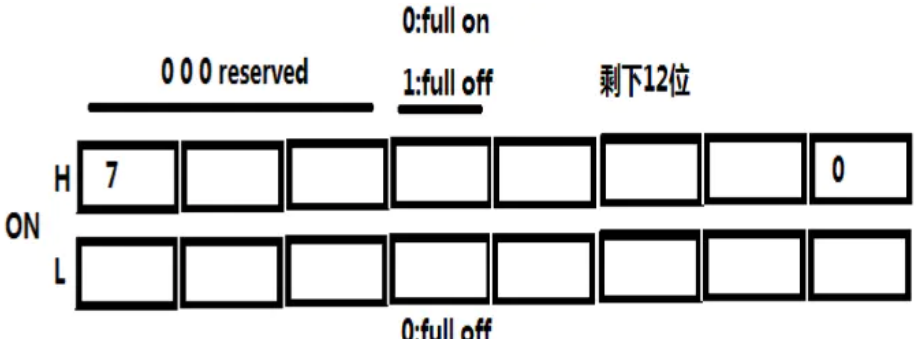
2.5 PWM通道寄存器及其占空比设置

PWM通道寄存器如下图：

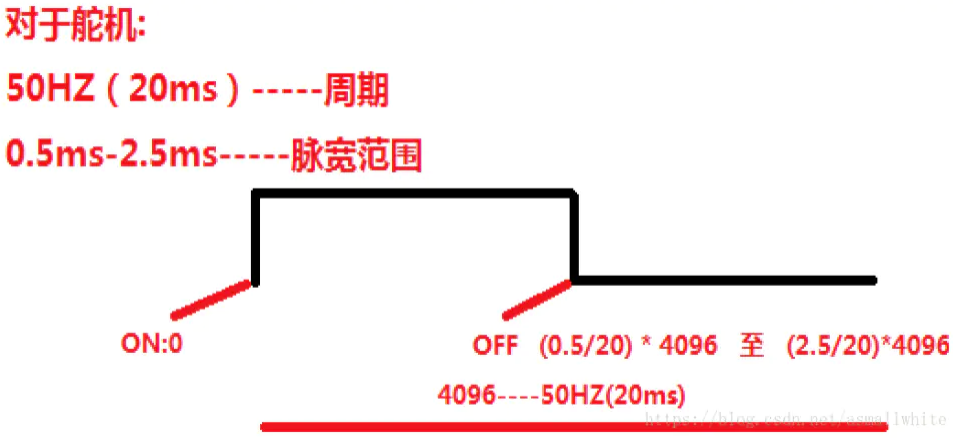
Table 7. LED_ON, LED_OFF control registers (address 06h to 45h) bit description
*Legend: * default value.*

Address	Register	Bit	Symbol	Access	Value	Description
06h	LED0_ON_L	7:0	LED0_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED0, 8 LSBs
07h	LED0_ON_H	7:5	reserved	R	000*	non-writable
		4	LED0_ON_H[4]	R/W	0*	LED0 full ON
		3:0	LED0_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED0, 4 MSBs
08h	LED0_OFF_L	7:0	LED0_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED0, 8 LSBs
09h	LED0_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED0_OFF_H[4]	R/W	1*	LED0 full OFF
		3:0	LED0_OFF_H[3:0]	R/W	0000*	

由图可知，对于每一个通道，有4个寄存器，每个寄存器图解如下图所示:



在设置PWM占空比的时候，首先，配置舵机的示例如下图所示(ON < OFF的情况):



当特殊情况下，PWM周期大于定时器一次计数时,如下图所示(ON>OFF的情况):

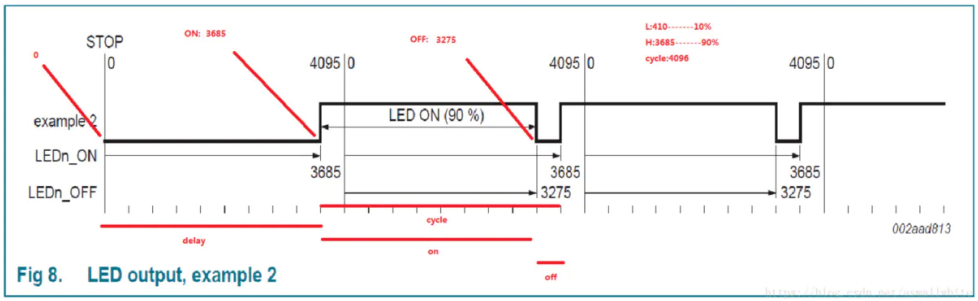


Fig 8. LED output, example 2

2.6 PWM周期(频率)寄存器及其周期(频率)设置

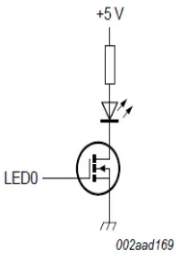
接下来介绍配置PWM频率的寄存器:
一般情况下，在用内置晶振，为25MHZ，通过配置PRE_SCALE寄存器进行配置，配置的PRE_SCALE寄存器的值与PWM频率的关系如下图所示:

$$prescale\ value = round\left(\frac{osc\_clock}{4096 \times update\_rate}\right) - 1$$

如果在舵机控制中，采用内置晶振，取osc_clock=25000000，update_rate=50(舵机控制频率50Hz)

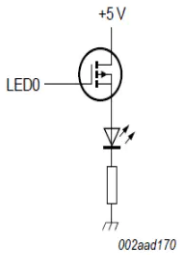
2.7 推荐硬件设计

首先，OE引脚须接低电平以确保IC使能，如果连接LED灯，则推荐以下几种连接方式，如下图所示:



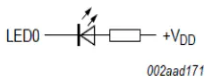
INVRT = 0
OUTDRV = 1

Fig 13. External N-type driver



INVRT = 1
OUTDRV = 1

Fig 14. External P-type driver



INVRT = 1
OUTDRV = 0

Fig 15. Direct LED connection

3 软件设计

3.1 Micro:bit平台TypeScript版

接下来进行软件设计部分讲解，由于本次开发采用Micro:bit底层开发，采用的是TypeScript(JavaScript的超类),所以暂提供该语言，提供基本操作方法及其思路，日后再更新C,C++及其它平台(STM32,Linux树莓派，Arduino等)操作方法，源码如下，可结合DataSheet及以上教程理解：

```
/**
 * 使用此文件来定义自定义函数和图形块。
 * 想了解更详细的信息，请前往 https://makecode.microbit.org/blocks/custom
 */

/**
 * 自定义图形块
 */
//% weight=5 color=#0fbc11 icon="\uf113"
namespace Servo {
    const PCA9685_ADDRESS = 0x40
    const MODE1 = 0x00
    const MODE2 = 0x01
    const SUBADR1 = 0x02
    const SUBADR2 = 0x03
    const SUBADR3 = 0x04
    const PRESCALE = 0xFE
    const LED0_ON_L = 0x06
    const LED0_ON_H = 0x07
    const LED0_OFF_L = 0x08
    const LED0_OFF_H = 0x09
    const ALL_LED_ON_L = 0xFA
    const ALL_LED_ON_H = 0xFB
    const ALL_LED_OFF_L = 0xFC
    const ALL_LED_OFF_H = 0xFD

    const STP_CHA_L = 2047
    const STP_CHA_H = 4095

    const STP_CHB_L = 1
    const STP_CHB_H = 2047

    const STP_CHC_L = 1023
    const STP_CHC_H = 3071

    const STP_CHD_L = 3071
    const STP_CHD_H = 1023

    let initialized = false

    function i2cwrite(addr: number, reg: number, value: number) {
        let buf = pins.createBuffer(2)
        buf[0] = reg
        buf[1] = value
        pins.i2cWriteBuffer(addr, buf)
    }

    function i2cread(addr: number, reg: number) {
        pins.i2cWriteNumber(addr, reg, NumberFormat.UInt8BE);
        let val = pins.i2cReadNumber(addr, NumberFormat.UInt8BE);
        return val;
```

```
    }

    function initPCA9685(): void {
        i2cwrite(PCA9685_ADDRESS, MODE1, 0x00)
        setFreq(50);
        setPwm(0, 0, 4095);
        for (let idx = 1; idx < 16; idx++) {
            setPwm(idx, 0, 0);
        }
        initialized = true
    }

    function setFreq(freq: number): void {
        // Constrain the frequency
        let prescaleval = 25000000;
        prescaleval /= 4096;
        prescaleval /= freq;
        prescaleval -= 1;
        let prescale = prescaleval; //Math.Floor(prescaleval + 0.5);
        let oldmode = i2cread(PCA9685_ADDRESS, MODE1);
        let newmode = (oldmode & 0x7F) | 0x10; // sleep
        i2cwrite(PCA9685_ADDRESS, MODE1, newmode); // go to sleep
        i2cwrite(PCA9685_ADDRESS, PRESCALE, prescale); // set the prescaler
        i2cwrite(PCA9685_ADDRESS, MODE1, oldmode);
        control.waitMicros(5000);
        i2cwrite(PCA9685_ADDRESS, MODE1, oldmode | 0xa1);
    }

    function setPwm(channel: number, on: number, off: number): void {
        if (channel < 0 || channel > 15)
            return;

        let buf = pins.createBuffer(5);
        buf[0] = LED0_ON_L + 4 * channel;
        buf[1] = on & 0xff;
        buf[2] = (on >> 8) & 0xff;
        buf[3] = off & 0xff;
        buf[4] = (off >> 8) & 0xff;
        pins.i2cWriteBuffer(PCA9685_ADDRESS, buf);
    }

    /**
     * Servo Execute
     * @param degree [0-180] degree of servo; eg: 90, 0, 180
    */
    //% blockId=setServo block="Servo channel|%channel|degree %degree"
    //% weight=85
    //% degree.min=0 degree.max=180
    export function Servo(channel: number,degree: number): void {
        if (!initialized) {
            initPCA9685();
        }
        // 50hz: 20,000 us
        let v_us = (degree * 1800 / 180 + 600); // 0.6 ~ 2.4
        let value = v_us * 4096 / 20000;
        setPwm(channel, 0, value);
    }

    /**
     * Servo Execute
     * @param pulse [500-2500] pulse of servo; eg: 1500, 500, 2500
    */
    //% blockId=setServoPulse block="Servo channel|%channel|pulse %pulse"
    //% weight=85
    //% pulse.min=500 pulse.max=2500
    export function ServoPulse(channel: number,pulse: number): void {
        if (!initialized) {
            initPCA9685();
        }
        // 50hz: 20,000 us
        let value = pulse * 4096 / 20000;
        setPwm(channel, 0, value);
    }
}
```

以上便是Micro:bit驱动PCA9685的源代码，注意源代码中的时间为us，而教程中的时间为ms。

3.2 树莓派平台Python版

要运行该程序，首先选装python，安装好Python后，还需要安装树莓派平台的smbus库：

```
sudo apt-get install python-smbus

```

树莓派平台采用Python驱动PCA9685的Python代码如下所示:

```
#!/usr/bin/python

import time
import math
import smbus

# ============================================================================
# Raspi PCA9685 16-Channel PWM Servo Driver
# ============================================================================

class PCA9685:

  # Registers/etc.
  __SUBADR1            = 0x02
  __SUBADR2            = 0x03
  __SUBADR3            = 0x04
  __MODE1              = 0x00
  __PRESCALE           = 0xFE
  __LED0_ON_L          = 0x06
  __LED0_ON_H          = 0x07
  __LED0_OFF_L         = 0x08
  __LED0_OFF_H         = 0x09
  __ALLLED_ON_L        = 0xFA
  __ALLLED_ON_H        = 0xFB
  __ALLLED_OFF_L       = 0xFC
  __ALLLED_OFF_H       = 0xFD

  def __init__(self, address=0x40, debug=False):
    self.bus = smbus.SMBus(1)
    self.address = address
    self.debug = debug
    if (self.debug):
      print("Reseting PCA9685")
    self.write(self.__MODE1, 0x00)

  def write(self, reg, value):
    "Writes an 8-bit value to the specified register/address"
    self.bus.write_byte_data(self.address, reg, value)
    if (self.debug):
      print("I2C: Write 0x%02X to register 0x%02X" % (value, reg))

  def read(self, reg):
    "Read an unsigned byte from the I2C device"
    result = self.bus.read_byte_data(self.address, reg)
    if (self.debug):
      print("I2C: Device 0x%02X returned 0x%02X from reg 0x%02X" % (self.address, result & 0xFF, reg))
    return result

  def setPWMFreq(self, freq):
    "Sets the PWM frequency"
    prescaleval = 25000000.0    # 25MHz
    prescaleval /= 4096.0       # 12-bit
    prescaleval /= float(freq)
    prescaleval -= 1.0
    if (self.debug):
      print("Setting PWM frequency to %d Hz" % freq)
      print("Estimated pre-scale: %d" % prescaleval)
    prescale = math.floor(prescaleval + 0.5)
    if (self.debug):
      print("Final pre-scale: %d" % prescale)

    oldmode = self.read(self.__MODE1);
    newmode = (oldmode & 0x7F) | 0x10        # sleep
    self.write(self.__MODE1, newmode)        # go to sleep
    self.write(self.__PRESCALE, int(math.floor(prescale)))
    self.write(self.__MODE1, oldmode)
    time.sleep(0.005)
    self.write(self.__MODE1, oldmode | 0x80)

  def setPWM(self, channel, on, off):
    "Sets a single PWM channel"
    self.write(self.__LED0_ON_L+4*channel, on & 0xFF)
```

```
    self.write(self.__LED0_ON_H+4*channel, on >> 8)
    self.write(self.__LED0_OFF_L+4*channel, off & 0xFF)
    self.write(self.__LED0_OFF_H+4*channel, off >> 8)
    if (self.debug):
      print("channel: %d  LED_ON: %d LED_OFF: %d" % (channel,on,off))

  def setServoPulse(self, channel, pulse):
    "Sets the Servo Pulse,The PWM frequency must be 50HZ"
    pulse = pulse*4096/20000        #PWM frequency is 50HZ,the period is 20000us
    self.setPWM(channel, 0, pulse)

if __name__=='__main__':

  pwm = PCA9685(0x40, debug=True)
  pwm.setPWMFreq(50)
  while True:
   # setServoPulse(2,2500)
    for i in range(500,2500,10):
      pwm.setServoPulse(0,i)
      time.sleep(0.02)

    for i in range(2500,500,-10):
      pwm.setServoPulse(0,i)
      time.sleep(0.02)
```

保存文件命名为: pca9685.py,命令行进入该文件所在的路径，运行该Python脚本:

```
sudo python pca9685.py
```

执行该命令后，便可控制舵机从0度转到180度，再从180度转到0度。