# TEST HYDRAULIC ELECTROVALVES

## Proposal

The Particle Web IDE

National control devices

PCA9685 8-Channel 8W 12V FET Driver Proportional Valve Controller

8-Channel DC Current Monitor with I2C Interface

2-Channel Signal Relay 1A SPDT I2C Mini Module

1 i2C pressure sensor

1 Interface ADC....etc

Rasbery i2c communication, Particle Photon,The Particle Web IDE more info :[https://store.ncd.io/](https://www.freelancer.com.pe/users/l.php?url=https:%2F%2Fstore.ncd.io%2F&sig=085abbd7bbab6c331ecdfab2ecffd19ffe772881e0ee25e5d524706c3389df8d)2

More information:

[Requirements.pdf](https://unipe-my.sharepoint.com/personal/lsayaverdeb_uni_pe/Documents/Freelance/Projects/Test_hidraulic_electrovalves/Requirements.pdf)

### Remote access

Anydesk

User: 384892775

Pass: eral2000

## Configuration

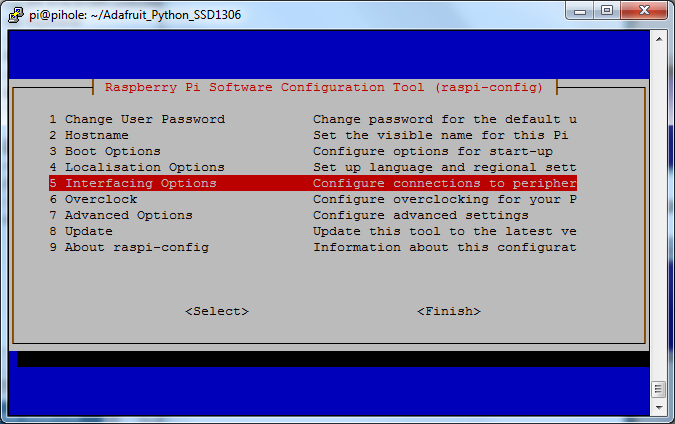
Source:

<https://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c>

### Installing Kernel Support (with Raspi-Config)

Run sudo raspi-config and follow the prompts to install i2c support for the ARM core and linux kernel

* Go to Interfacing Options



* On older versions, look under Advanced

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

* then I2C

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* Enable

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

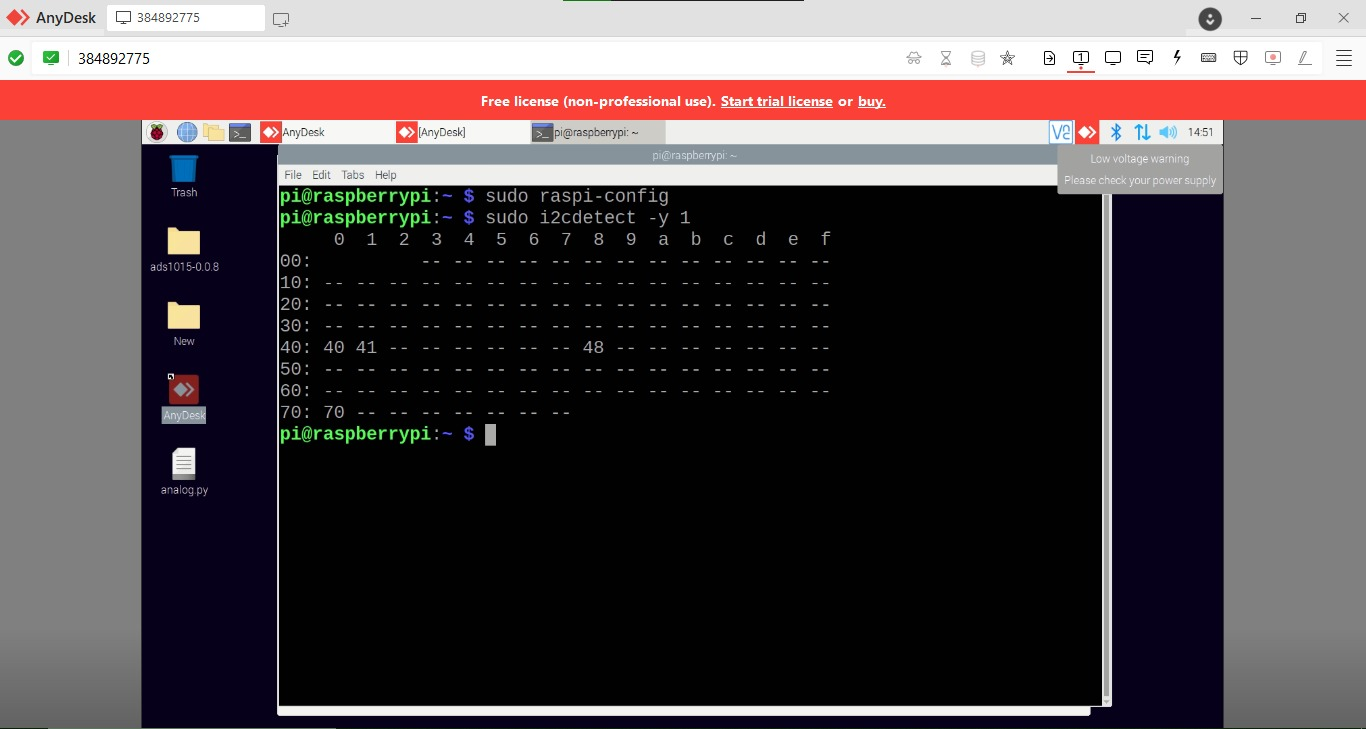
Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

### Testing I2C

Now when you log in you can type the following command to see all the connected devices

sudo i2cdetect -y 1



This shows that four I2C addresses are in use.

|  |  |
| --- | --- |
| Device | Address |
| [PCA9685](#_I2C_PCA9685) | 0X40 |
| [PCA9536](#_I2C_PCA9536) | 0X41 |
| [ADS7828 + ACS7112](#_I2C_ADS7828+ACS712) | 0X48 |
| [PCA9685](#_I2C_PCA9685) | 0X70 |

### Support libraries

[Installing CircuitPython Libraries on Raspberry Pi | CircuitPython on Linux and Raspberry Pi | Adafruit Learning System](https://learn.adafruit.com/circuitpython-on-raspberrypi-linux/installing-circuitpython-on-raspberry-pi)

sudo apt-get update

sudo apt-get upgrade

sudo apt-get install python3-pip

sudo pip3 install --upgrade setuptools

## I2C PCA9685

### Device overview

[PCA9685 8-Channel 8W 12V FET Driver Proportional Valve Controller with IoT Interface - store.ncd.io](https://store.ncd.io/product/pca9685-8-channel-8w-12v-fet-driver-proportional-valve-controller-with-iot-interface/)

### Datasheet

<https://cdn-shop.adafruit.com/datasheets/PCA9685.pdf>

### Libraries

documentation:

[adafruit\_pca9685 — Adafruit PCA9685 Library 1.0 documentation (circuitpython.org)](https://docs.circuitpython.org/projects/pca9685/en/latest/api.html)

tutorial:

[Python & CircuitPython | Adafruit PCA9685 16-Channel Servo Driver | Adafruit Learning System](https://learn.adafruit.com/16-channel-pwm-servo-driver/python-circuitpython)

sudo pip3 install adafruit-circuitpython-pca9685

### Example

# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries

# SPDX-License-Identifier: MIT

# This simple test outputs a 50% duty cycle PWM single on the 0th channel. Connect an LED and

# resistor in series to the pin to visualize duty cycle changes and its impact on brightness.

from board import SCL, SDA

import busio

# Import the PCA9685 module.

from adafruit\_pca9685 import PCA9685

# Create the I2C bus interface.

i2c\_bus = busio.I2C(SCL, SDA)

# Create a simple PCA9685 class instance.

pca = PCA9685(i2c\_bus)

# Set the PWM frequency to 60hz.

pca.frequency = 60

# Set the PWM duty cycle for channel zero to 50%. duty\_cycle is 16 bits to match other PWM objects

# but the PCA9685 will only actually give 12 bits of resolution.

pca.channels[0].duty\_cycle = 0x7FFF

increase, decrease PWM duty cicle

# Increase brightness:

for i in range(0xffff):

led\_channel.duty\_cycle = i

# Decrease brightness:

for i in range(0xffff, 0, -1):

led\_channel.duty\_cycle = i

### I2c address

The PCA9685 is on I2C address 0x40.

By default in line pca = PCA9685(i2c) the address is 0x40 and ref frec is 25000000

To full customizable line look like:

PCA9685(i2c, address, reference\_clock\_speed)

Address: 0x00 to 0x77, by default 0x40

reference\_clock\_speed: in Hz, by default 25 000 000Hz

### With library smbus

smbus: [Adafruit\_Python\_PureIO/smbus.py at main · adafruit/Adafruit\_Python\_PureIO · GitHub](https://github.com/adafruit/Adafruit_Python_PureIO/blob/main/Adafruit_PureIO/smbus.py)

[Adafruit\_PureIO.smbus — Adafruit PureIO Library 1.0 documentation (adafruit-pureio.readthedocs.io)](https://adafruit-pureio.readthedocs.io/en/latest/api.html)

GitHub: [PIPPY/PCA9685.py at master · waveshare/PIPPY · GitHub](https://github.com/waveshare/PIPPY/blob/master/PCA9685.py)

#### Example

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, int(pulse))

if \_\_name\_\_=='\_\_main\_\_':

#I2C channel, where the PCA9685 is connected, is chosen, 0x40

pwm = PCA9685(0x40, debug=False)

#frecuency of PWM is set

pwm.setPWMFreq(50)

while True:

#setServoPulse(channel , microseconds to off)

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)

## I2C ADS7828+ACS712

### Device overview

[8-Channel DC Current Monitor with I2C Interface - store.ncd.io](https://store.ncd.io/product/8-channel-dc-current-monitor-with-i2c-interface/)

### Datasheet

ADS7828:

<https://media.ncd.io/sites/2/20170721134916/ADS7828-2.pdf?_ga=2.226438362.1942944568.1675366777-1088693194.1675366777>

ACS712:

<https://www.sparkfun.com/datasheets/BreakoutBoards/0712.pdf>

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Senbility for current<5A 🡪 0.185 v/a

### Libraries

smbus: [Adafruit\_Python\_PureIO/smbus.py at main · adafruit/Adafruit\_Python\_PureIO · GitHub](https://github.com/adafruit/Adafruit_Python_PureIO/blob/main/Adafruit_PureIO/smbus.py)

[Adafruit\_PureIO.smbus — Adafruit PureIO Library 1.0 documentation (adafruit-pureio.readthedocs.io)](https://adafruit-pureio.readthedocs.io/en/latest/api.html)

GitHub: [Raspberry-Pi-Shild/ads7828.py at master · mocona05/Raspberry-Pi-Shild · GitHub](https://github.com/mocona05/Raspberry-Pi-Shild/blob/master/Python/ads7828.py)

### Example

import smbus

import time

DEVICE\_ADDRESS = 0x48 #7 bit address (will be left shifted to add the read write bit)

I2C\_CHANNEL = 1 #raspberry i2c channel

PULL\_UP\_RESIST =41.2

PULL\_DOWN\_RESIST =10.0

ADC\_REF\_VOLT =5.0

DEFAULT\_RATIO =1.0

ADC\_RESOLUTION =4095.0

#TC1047 temperature sensor

TC1047\_ZERO\_DEGC =0.5

TC1047\_DEGC\_FOR\_VOLT =0.01

#LM61B temperatur sensor

LM61B\_ZERO\_DEGC =0.6

LM61B\_DEGC\_FOR\_VOLT =0.01

# temperature sensor set

ZERO\_DEG = TC1047\_ZERO\_DEGC

DEGC\_FOR\_VOLT = TC1047\_DEGC\_FOR\_VOLT

#ADC7828 CONTROL REGISTER

ADS7828\_CONFIG\_SD\_DIFFERENTIAL = 0x00

ADS7828\_CONFIG\_SD\_SINGLE = 0x80

ADS7828\_CONFIG\_CS\_CH0 = 0x00

ADS7828\_CONFIG\_CS\_CH1 = 0x40

ADS7828\_CONFIG\_CS\_CH2 = 0x10

ADS7828\_CONFIG\_CS\_CH3 = 0x50

ADS7828\_CONFIG\_CS\_CH4 = 0x20

ADS7828\_CONFIG\_CS\_CH5 = 0x60

ADS7828\_CONFIG\_CS\_CH6 = 0x30

ADS7828\_CONFIG\_CS\_CH7 = 0x70

ADS7828\_CONFIG\_PD\_OFF = 0x00

ADS7828\_CONFIG\_PD\_REFOFF\_ADON = 0x04

ADS7828\_CONFIG\_PD\_REFON\_ADOFF = 0x08

ADS7828\_CONFIG\_PD\_REFON\_ADON = 0x0c

#ADS 7828 I2C CONTROL CLASS

class Ads7828:

def \_\_init\_\_(self, address=DEVICE\_ADDRESS, bus\_id=I2C\_CHANNEL, debug=False):

self.i2c = smbus.SMBus(bus\_id)

self.address = address

self.debug = debug

def read\_raw\_adc(self, ch):

config = 0

config |= ADS7828\_CONFIG\_SD\_SINGLE

config |= ADS7828\_CONFIG\_PD\_REFOFF\_ADON

if ch == 0:

config |= ADS7828\_CONFIG\_CS\_CH0

elif ch == 1:

config |= ADS7828\_CONFIG\_CS\_CH1

elif ch == 2:

config |= ADS7828\_CONFIG\_CS\_CH2

elif ch == 3:

config |= ADS7828\_CONFIG\_CS\_CH3

elif ch == 4:

config |= ADS7828\_CONFIG\_CS\_CH4

elif ch == 5:

config |= ADS7828\_CONFIG\_CS\_CH5

elif ch == 6:

config |= ADS7828\_CONFIG\_CS\_CH6

elif ch == 7:

config |= ADS7828\_CONFIG\_CS\_CH7

time.sleep(0.01)#adc convertion time waiting

data =[0,0]

data = self.i2c.read\_i2c\_block\_data(self.address,config,2)

time.sleep(0.01)

return ((data[0] << 8) + data[1])

def all\_ch\_raw\_adc\_display(self):

data= [0]\*8

for i in range (8) :

data[i] = adc.read\_raw\_adc(i)

print ("ch0=%d, ch1=%d, ch2=%d, ch3=%d, ch4=%d, ch5=%d, ch6=%d, ch7=%d" %(data[0],data[1],data[2],data[3],data[4],data[5],data[6],data[7]))

#ADC CONVERTION CALCULATOR CLASS

class Voltage\_cal(Ads7828):

def \_\_init\_\_(self, address=DEVICE\_ADDRESS, bus\_id=I2C\_CHANNEL, debug=False, vref=ADC\_REF\_VOLT,default\_ratio=DEFAULT\_RATIO, offset=0):

Ads7828.\_\_init\_\_(self, address, bus\_id, debug)

self.adc\_resol =ADC\_RESOLUTION

self.vref =vref

self.ratio =[self.adc\_resol/(default\_ratio\*self.vref)]\*8

self.offset =[offset]\*8

self.temp\_offset =[0.0]\*8

self.temp\_devider = [1.0]\*8

self.adc\_ratio\_int()

Ads7828()

def adc\_ratio\_int(self):

self.temp\_ratio(0) # ADC temperature sensor setting

self.ratio\_set(1,0,1) #direct input adc(none pull-up & pull-down resist)

self.ratio\_set(2,0,1) #direct input adc(none pull-up & pull-down resist)

self.ratio\_set(3,0,1) #direct input adc(none pull-up & pull-down resist)

self.ratio\_set(7,0,1) #direct input adc(none pull-up & pull-down resist)

for i in range(4,7):

self.ratio\_set(i) #adc pull-up & pull-down resist ratio set

def ratio\_set(self, ch, PULL\_UP\_RESIST=PULL\_UP\_RESIST, PULL\_DOWN\_RESIST =PULL\_DOWN\_RESIST): #channel pull-up/pull-down ratio set

regSum = PULL\_UP\_RESIST+PULL\_DOWN\_RESIST

self.ratio[ch] = self.adc\_resol/((regSum/PULL\_DOWN\_RESIST)\*self.vref)

def temp\_ratio(self, ch, v\_for\_deg=DEGC\_FOR\_VOLT, zero\_for\_volt=ZERO\_DEG):

self.temp\_offset[ch] = zero\_for\_volt / self.vref \* self.adc\_resol

self.temp\_devider[ch] = v\_for\_deg / self.vref \* self.adc\_resol

def read\_voltage(self, ch):

ch\_value = self.read\_raw\_adc(ch)/self.ratio[ch]-self.offset[ch]

return round(ch\_value,2)

def read\_temp(self,ch):

ch\_value = (self.read\_raw\_adc(ch)-self.temp\_offset[ch]) / self.temp\_devider[ch]

return round(ch\_value,2)

def all\_ch\_value\_display(self):

data= [0]\*8

data[0] = self.read\_temp(0)

for i in range (1, 8) :

#data that is a voltage values is scaled to a represent a current value

#an sensibility of 0.185V/A is used because the current is less than 5A

data[i] = (self.read\_voltage(i) -2.5)/0.185

print ("ch0=%2.2fA, ch1=%2.2fA, ch2=%2.2fA, ch3=%2.2fA, ch4=%2.2fA, ch5=%2.2fA, In ch6=%2.2fA, ch7=%2.2fA"

%(data[0],data[1],data[2],data[3],data[4],data[5],data[6],data[7]))

if \_\_name\_\_ =="\_\_main\_\_":

# adc = Ads7828()

adc\_measu = Voltage\_cal()

while True:

# adc.all\_ch\_raw\_adc\_display()

adc\_measu.all\_ch\_value\_display()

## I2C PCA9536

### Device overview

[2-Channel Signal Relay 1A SPDT I2C Mini Module - store.ncd.io](https://store.ncd.io/product/2-channel-signal-relay-1a-spdt-i2c-mini-module/)

### Datasheet

[PCA9536 Remote 4-Bit I2C and SMBus I/O Expander with Configuration Registers datasheet (Rev. H)](https://www.ti.com/lit/ds/scps125h/scps125h.pdf?ts=1675463898501&ref_url=https%253A%252F%252Fwww.google.com%252F)

### Libraries

smbus: [Adafruit\_Python\_PureIO/smbus.py at main · adafruit/Adafruit\_Python\_PureIO · GitHub](https://github.com/adafruit/Adafruit_Python_PureIO/blob/main/Adafruit_PureIO/smbus.py)

[Adafruit\_PureIO.smbus — Adafruit PureIO Library 1.0 documentation (adafruit-pureio.readthedocs.io)](https://adafruit-pureio.readthedocs.io/en/latest/api.html)

GitHub: [GitHub - ControlEverythingCommunity/PCA9536: 4-Channel Input Output I2C Mini Module](https://github.com/ControlEverythingCommunity/PCA9536)

### Example

import smbus

import time

# Get I2C bus, i specify the I2C bus to use -> SMBus(i)

bus = smbus.SMBus(1)

# write\_data\_byte(address, command, value)

#Address: PCA9536 address, 0x41(65)

#Command register: Select configuration register, 0x03(03)

#Value : 0xFF(255) All pins configured as inputs

bus.write\_byte\_data(0x41, 0x03, 0xFF)

# Output to screen

print "All Pins State are HIGH"

time.sleep(0.5)

# Address: PCA9536 address, 0x41(65)

# Command register: Read data back from 0x00(00), 1 byte

data = bus.read\_byte\_data(0x41, 0x00)

# Convert the data to 4-bits 🡪 0000xxxx

data = (data & 0x0F)

# Compare data and find which switch is on and off 🡪 0001, 0010, 0100, 1000

for i in range(0, 4) :

if (data & (2 \*\* i)) == 0 :

print "I/O Pin %d State is LOW" %i

else :

print "I/O Pin %d State is HIGH" %i

time.sleep(0.5)

## Temperature sensor

### Device overview

### Datasheet

### Libraries

### Example

## Pressure sensor

### Device overview

### Datasheet

### Libraries

### Example

## Dashboard

Source:

[node-red-dashboard (node) - Node-RED (nodered.org)](https://flows.nodered.org/node/node-red-dashboard)