Pi-16ADC

16 bit, 16 channel Analog to Digital Converter (ADC) for Raspberry Pi

User Guide

Contents

Table of Figures

Pi-16ADC key features overview

- 16 single-ended or 8 Differential Analog to Digital Conversion (ADC) ports.
- 16-bit conversion – approx. 39 μV sensitivity for a 2.5V analog signal.
- Data exchange over I^2C bus.
- Supports 27 different I^2C addresses for the device.
- Sample Python code for ADC use.
- Ideal for data logging applications.
- Built in noise filter for 50 and 60Hz noise.
- Resets and recalibrates itself when power is off completely.

Please refer to the data sheet for additional information on the features. Data sheet can be downloaded from www.alchemypower.com

Downloads

The data sheet, User Guide (this document), sample code and other related software can be downloaded from www.alchemypower.com

Functional Description

The Pi-16ADC has several functions which are described below. The Pi-16ADC follows the guidelines set by the Raspberry Pi foundation for a HAT (Hardware Accessory on Top).

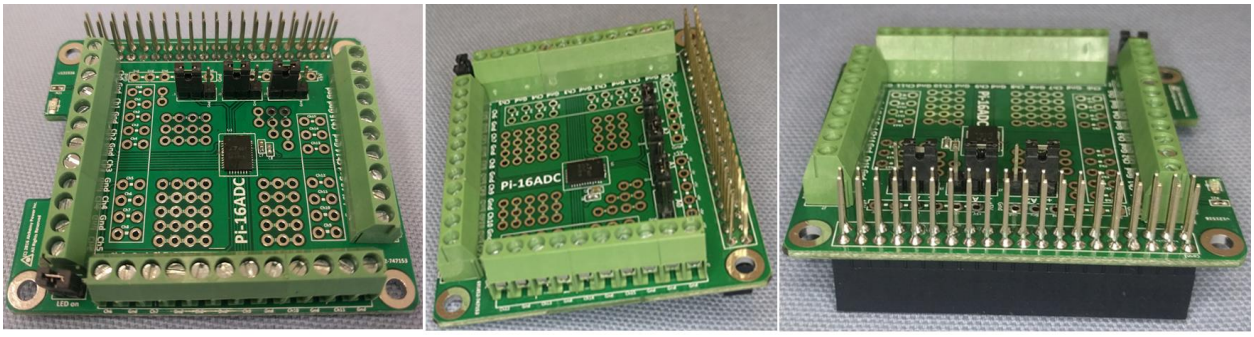

Figure 1 – Pi-16ADC showing front view, side view and rear view. The rear view shows the headers. All the signals from the Raspberry Pi headers are connected 1-on-1 allowing other boards or HATs to be stacked on top of Pi-16ADC.

The different views of the Pi-16ADC are shown above.

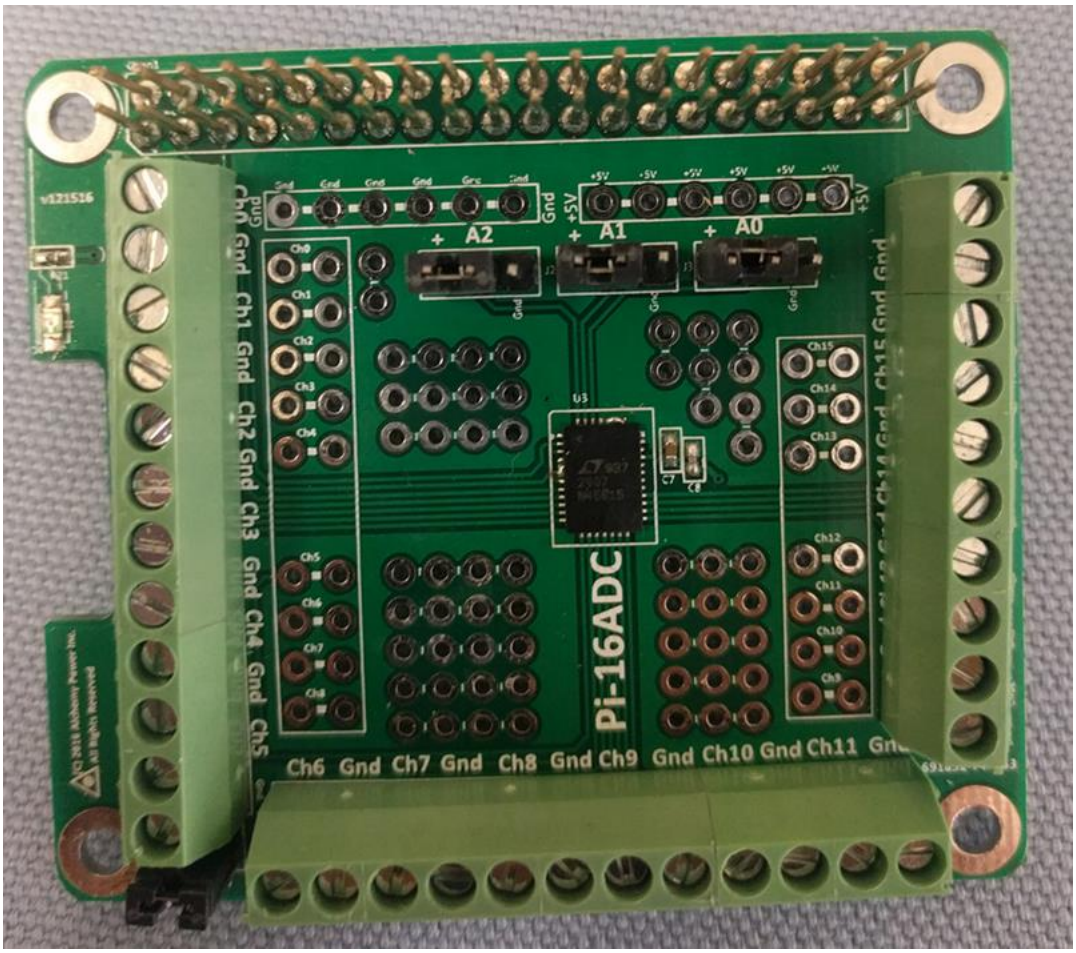

Figure 2 – Top view of Pi-16ADC showing all the connection points and address configuration jumpers. Connections to channels, +5V, GND (ground) etc. are clearly labelled. Bread board connections show a line for continuity or connections between the solder points via the board.

The top view shows the cut-out recommended by the HAT guidelines.

The solderless connections, also called terminal blocks are shown. Each terminal block connection is paired as Channel-Ground. For example, Channel-0 and Ground is in the top left corner. Channel-1 and Ground is the next pair going clockwise.

To minimize spurious readings and stray noise pickup, it is recommended to use the channel-ground pair as provided.

It is also recommended to connect unused channels to the ground with a short jumper wire when a channel is not in use.

All terminal block connections are marked on the board.

Each terminal block Channel also has a corresponding solder point next to it. These are also labelled and are shown below.

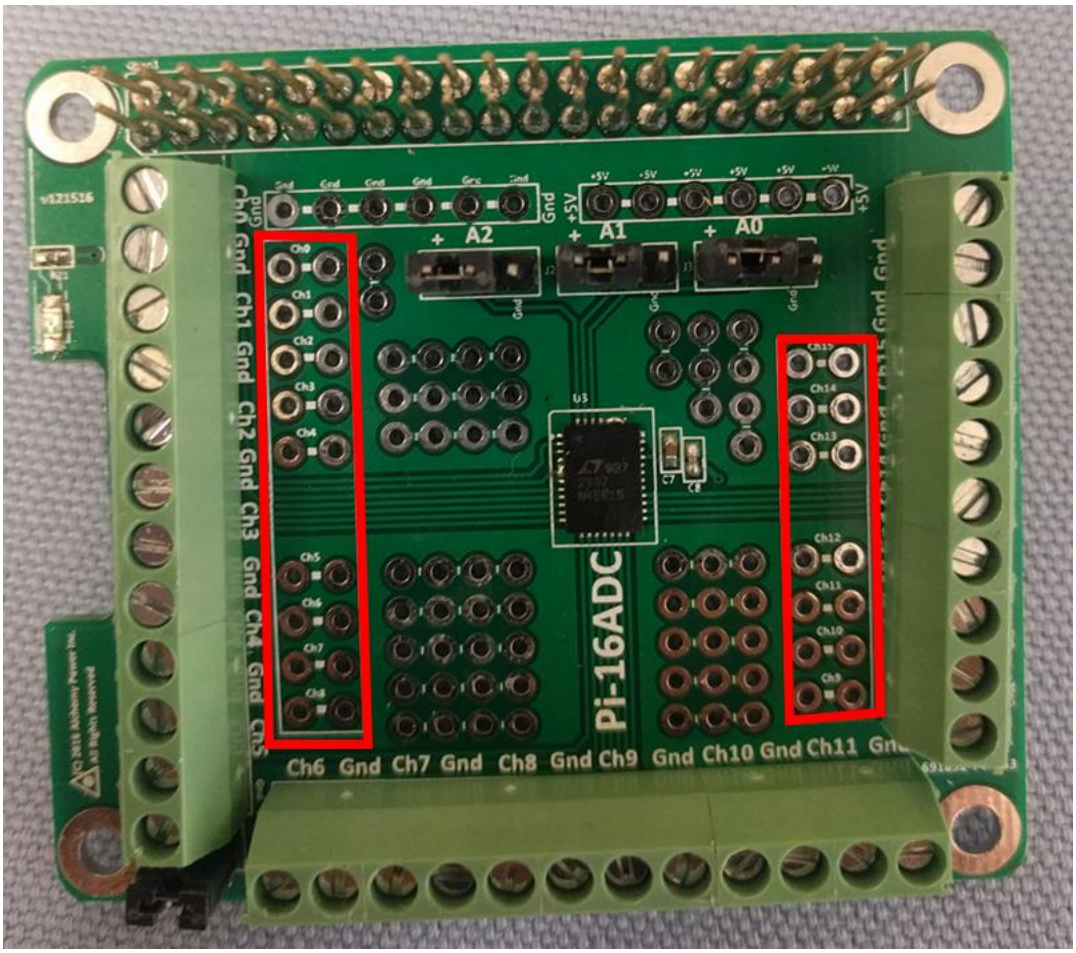

Figure 3 – solder points for each channel. Two solder points are provided to ease soldering of resistor dividers. Each solder connection is labelled, starting at channel-0 on top left side, going to channel-15, moving counter-clockwise.

The solder points as well as solderless connections make it easy to connect Pi-16ADC to various sensors.

Besides the channels, solder points are provided for +5V and Ground connections as shown below.

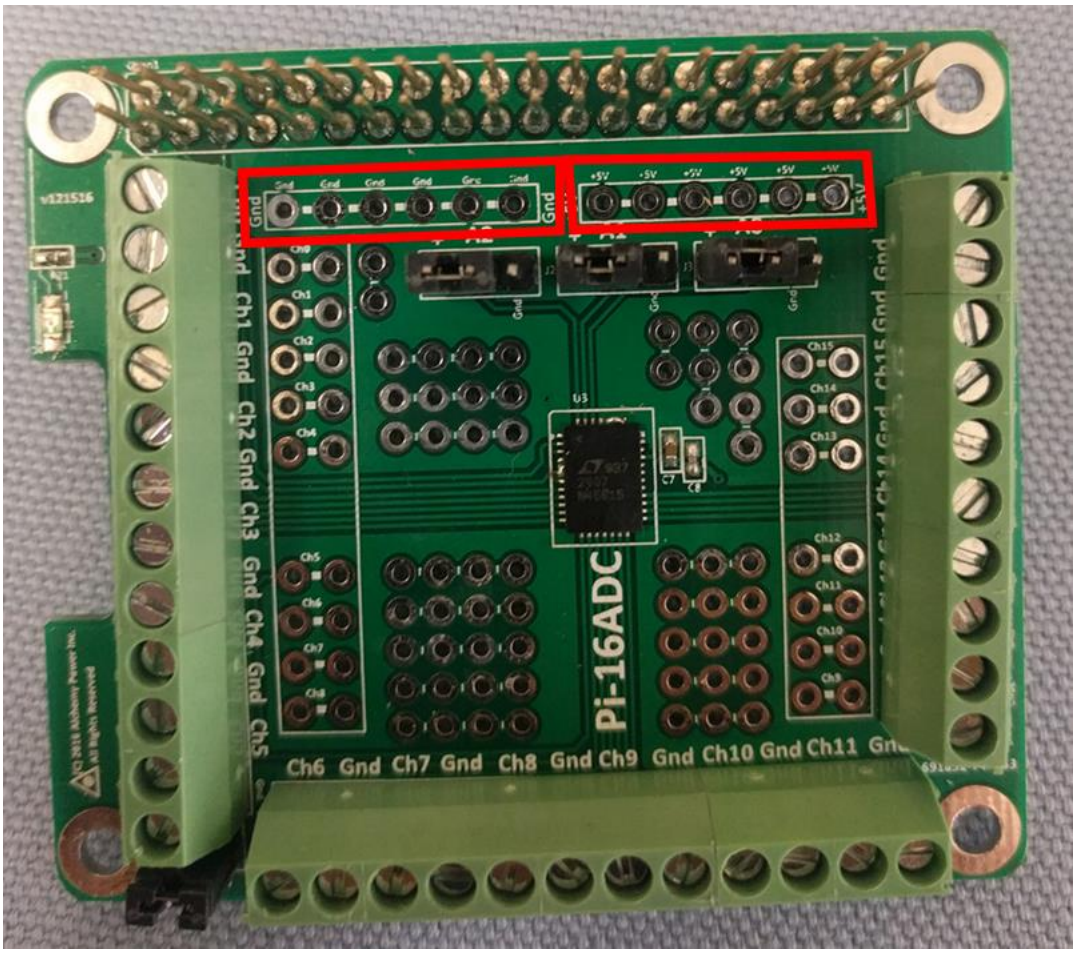

Figure 4 – Solder points for Ground (on the left) and +5V (on the right). The +5V is tapped from Pins 2 and 4 of the Raspberry Pi header.

The breadboard area which can be used with the Pi-16ADC is shown below.

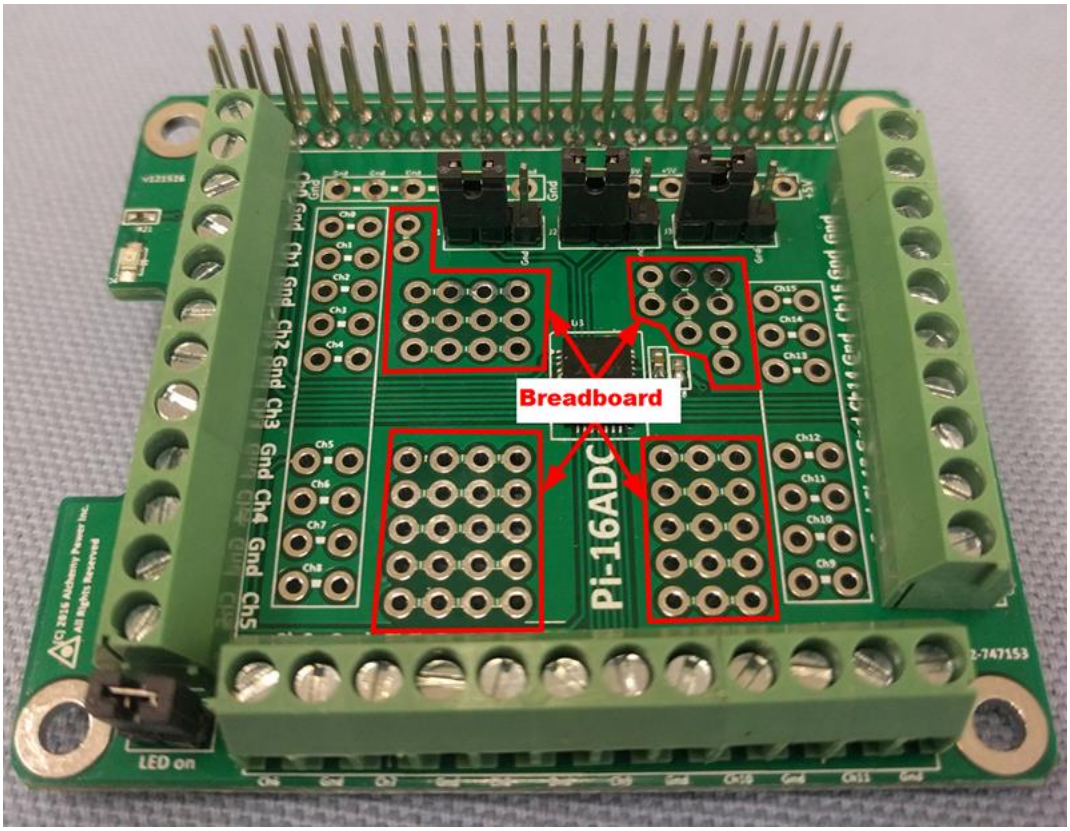

Figure 5 – Breadboard area for the Pi-16ADC. Note the lines through the solder points indicate which solder points are connected to each other.

The Pi-16ADC has an LED on board to indicate power is on and the board is active. This LED is bright and consumes some power. To reduce the power consumed by the board, the LED can be turned off and off at any time using the “LED on/off” jumper as shown below. The LED position is also shown in the picture below.

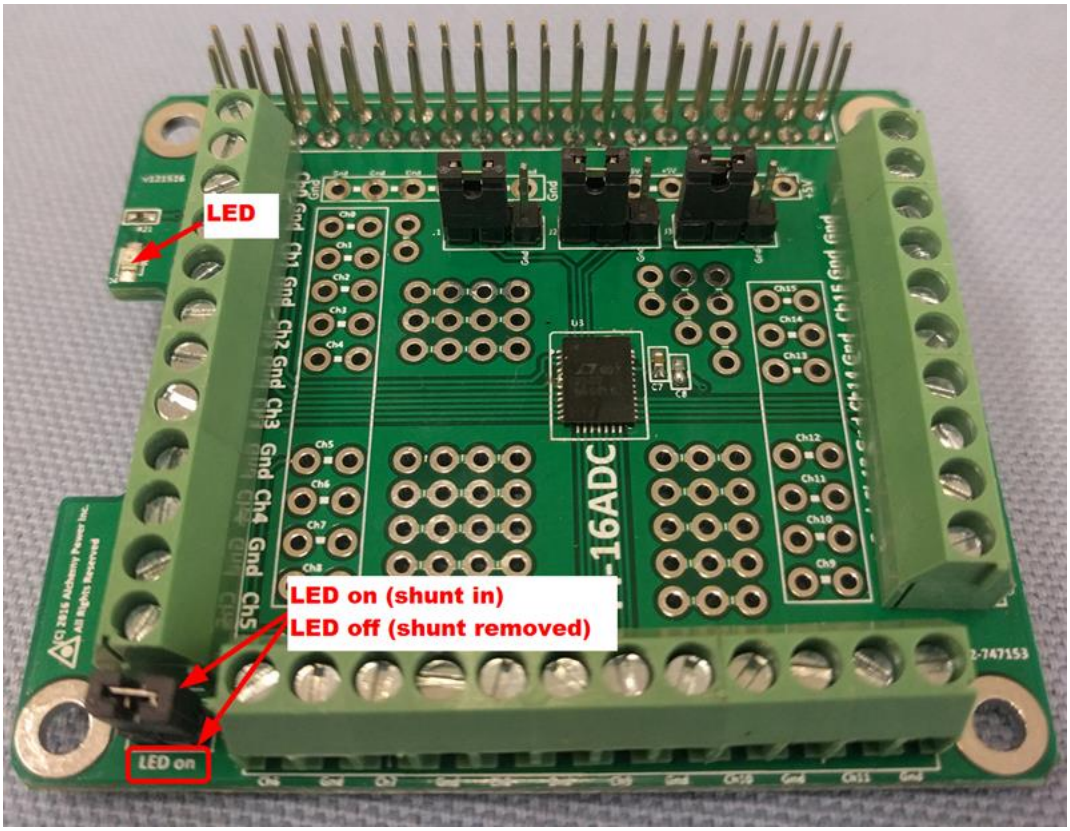

Figure 6 – LED can be turned off by removing the jumper. At any time, the jumper prongs can be shorted for a quick view if the power is on or off.

Pi-16ADC communicates with the Raspberry Pi using the I^2C bus. The I^2C bus can have several peripherals on it. Each peripheral is uniquely identified by the I^2C address. To set the address, 3 jumpers A0, A1 and A2 are provided. Each jumper can be

1) Connected to +5V – state defined as “High”.
2) Connected to Ground – stated defined as “Low”.
3) Removed – state defined as “Float”.

There are thus 27 possible addresses which can be defined using the I^2C address jumpers. See the next section on I^2C addressing.

In the picture below the jumpers are shown connected to +5V.

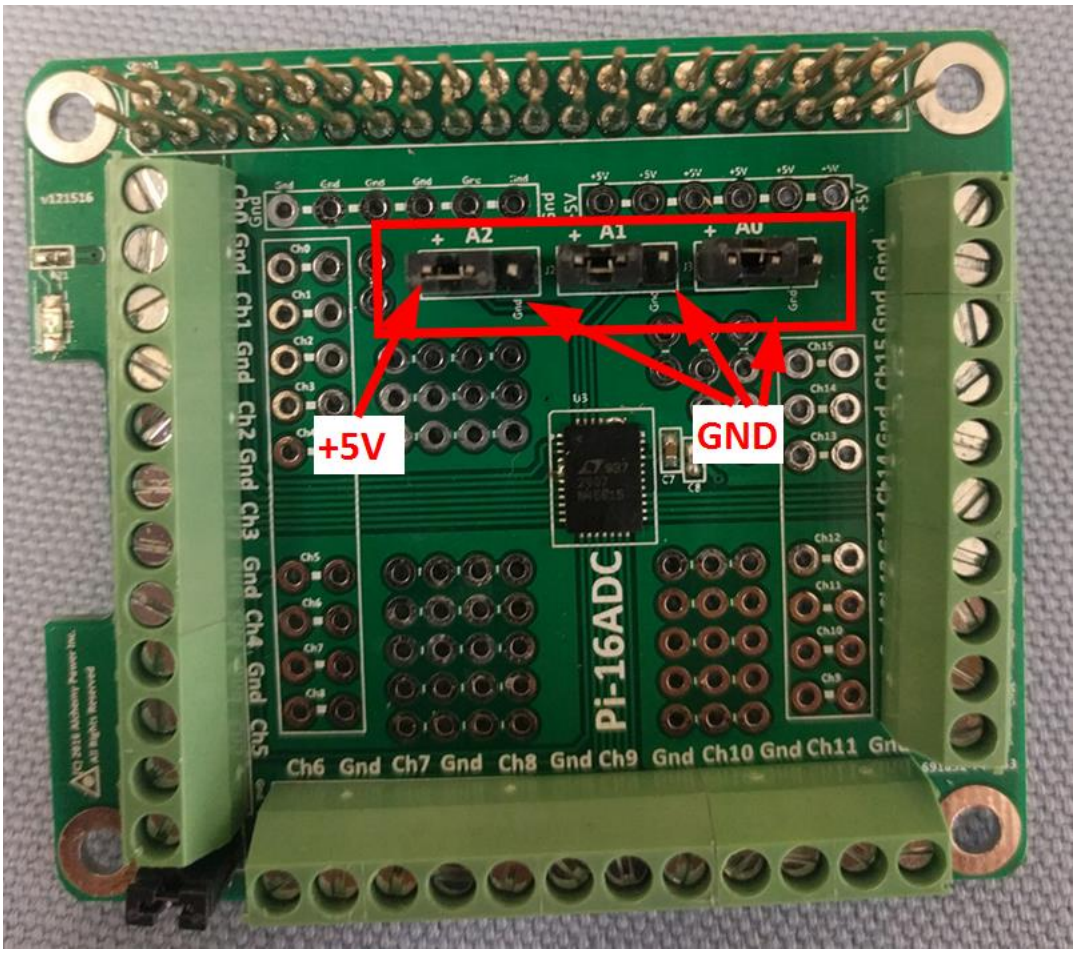

Figure 7 – I²C address jumpers. The jumpers can be connected to +5V (High), Ground (Low) or removed (Float). See the section on I²C address setup for more details.

Pi-Models Supported

Any Raspberry Pi with a 40 pin header is supported. This includes Pi-2, Pi-3, Pi-Zero, Pi-Zero W, A+, B+….

The headers (male pins as well as female mating pins) are included with Pi-16ADC.

Pi-16ADC I²C address setup

Every peripheral attached to the I²C bus requires a unique address. You have to ensure:

1) I²C is enabled.
2) Proper Python (or other programming languages you plan to use e.g. C or Java) libraries are loaded.

To enable I²C please use the Raspberry Pi Configuration utility provided. See picture below which shows how I²C can be configured.

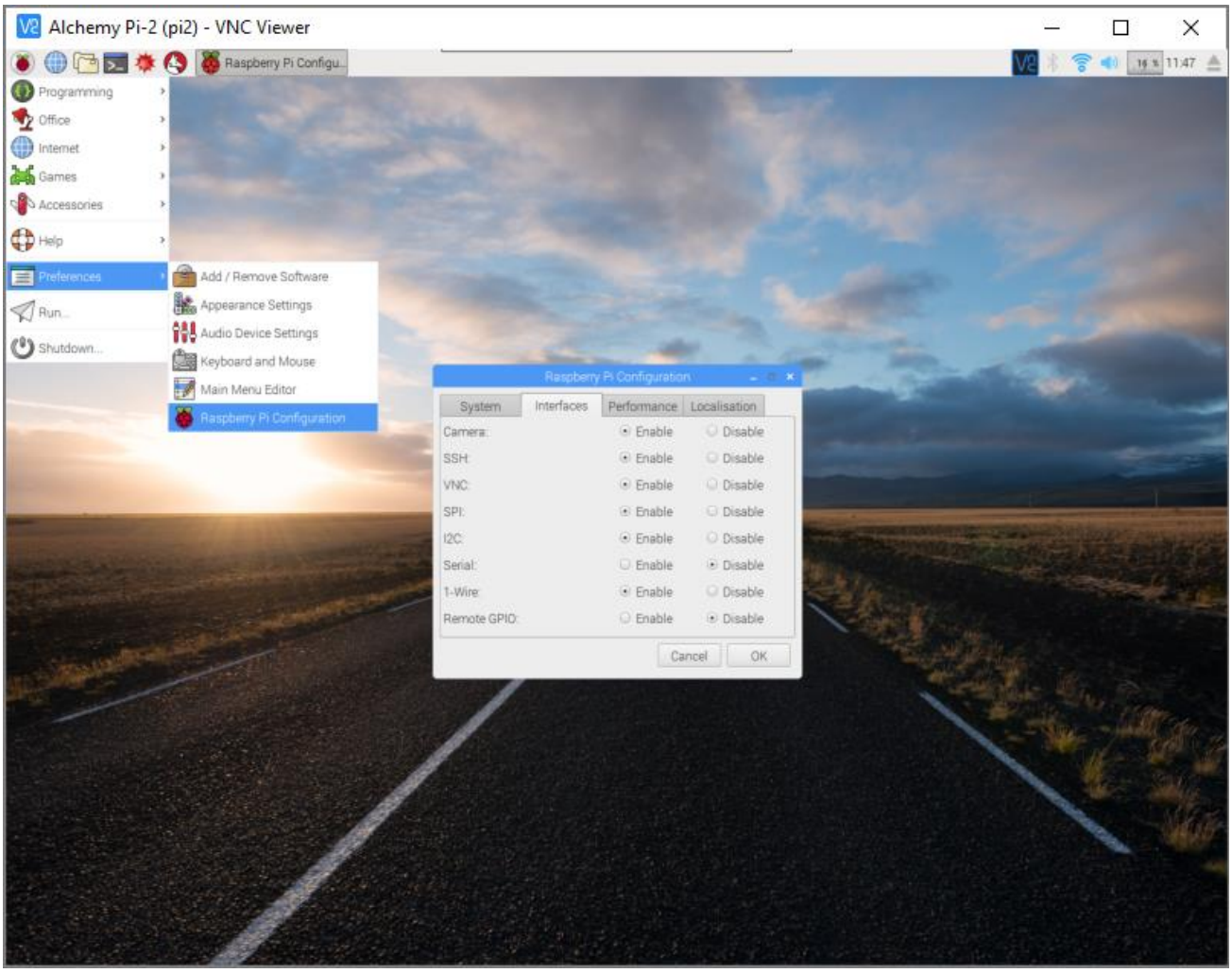

Figure 8 – using the Raspberry Pi Configuration utility to enable I²C. Use the Interfaces tab to enable I²C. This setup usually requires a reboot to ensure the changes made are enabled. The above image was captured using RealVNC on a Raspberry Pi-2 platform, running Pixel and a Real VNC viewer on Windows 10.

Once I²C is enabled, the address can be checked using the command "i2cdetect" in a terminal window. See the screen which shows that. First open a terminal window. Next type in the command "i2cdetect -y 1" as shown.

In the screen capture below, there is only one device connected – a Pi-16ADC at address 0x76.

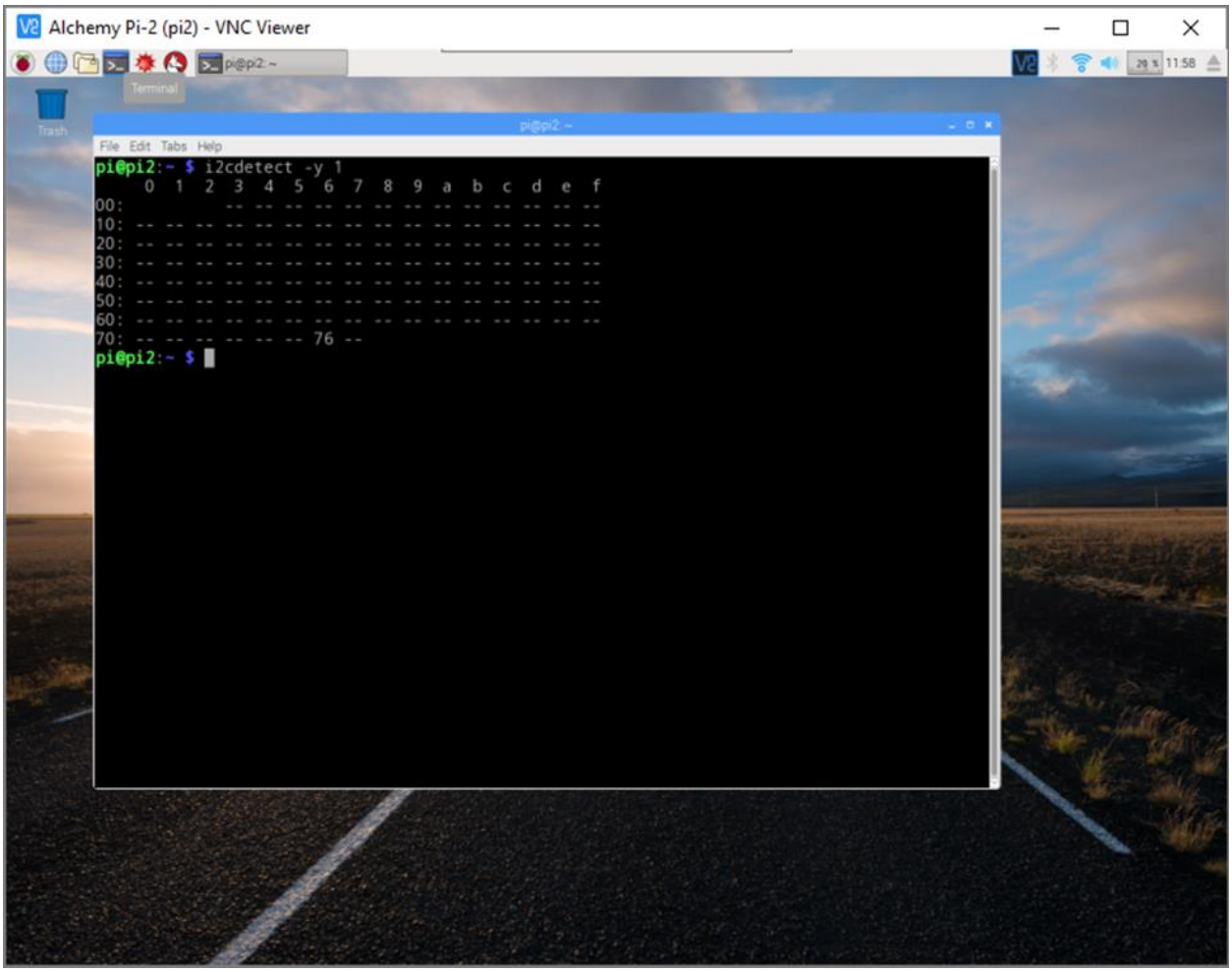

Figure 9 – Listing I^2C peripherals connected to the Raspberry Pi. In this example, the Pi-16ADC is the only peripheral connected and has the address of 0x76. The above image was captured using RealVNC on a Raspberry Pi-2 platform, running Pixel and a Real VNC viewer on Windows 10.

As discussed earlier, there are 27 unique address which can be set for the Pi-16ADC using jumpers A0, A1, A2. Each jumper can be connected to +5V (high), Ground (low) or removed (float).

The table below indicates the addresses.

This information is also duplicated in the Python script file to ease the coding effort once the address is changed. Please make sure to uncomment the line in the Python script to indicate the proper address in use. The default setting 0x76 is already uncommented in the sample Python code.

A2	A1	A0	Address (Hex)	Address (binary)
Low	Low	Low	0x14	0b0010100
Low	Low	High	0x16	0b0010110
Low	Low	Float	0x15	0b0010101
Low	High	Low	0x26	0b0100110
Low	High	High	0x34	0b0110100
Low	High	Float	0x27	0b0110100
Low	Float	Low	0x17	0b0010111
Low	Float	High	0x25	0b0100101
Low	Float	Float	0x24	0b0100100
High	Low	Low	0x56	0b1010110
High	Low	High	0x64	0b1100100
High	Low	Float	0x57	0b1010111
High	High	Low	0x74	0b1110100
High	High	High	0x76	0b1110110
High	High	Float	0x75	0b1110101
High	Float	Low	0x65	0b1100101
High	Float	High	0x67	0b1100111
High	Float	Float	0x66	0b1100110
Float	Low	Low	0x35	0b0110101
Float	Low	High	0x37	0b0110111
Float	Low	Float	0x36	0b0110110
Float	High	Low	0x47	0b1000111
Float	High	High	0x55	0b1010101
Float	High	Float	0x54	0b1010100
Float	Float	Low	0x44	0b1000100
Float	Float	High	0x46	0b1000110
Float	Float	Float	0x45	0b1000101

Table 1 – Addresses Pi-16ADC can assume for the I^2C bus. The default address is highlighted.

The default address (as shipped from the factory) is 0x76 or 0b1110110 (all jumper pins connected to +5V). This is highlighted in the table.

Please make sure that the python code reflects the address shown by the command “i2cdetect -y 1” command as shown earlier. In this document when we refer to address 0x76, we are referring to this device on the I^2C bus.

Python setup for UPS and ADC

You can skip this section if you already are using other I²C peripherals on the Pi.

The Python code for the UPS uses the I²C channel to collect data from the ADC chip. This guide assumes the user is familiar with edit (e.g. nano, vi, vim etc) and other utilities on a Linux (Pixel) system.

To setup the Python code, please make sure the Python libraries for I²C are installed. The latest version of Pixel OS does not need any modifications to /boot/config.text or to /etc/modules. For older versions of the Operating systems, please follow the following steps:

1) Use rpi-setup command or from the menu configure the Raspberry Pi to ensure I²C is enabled. Skip the rest of this step if the menu interface is used to enable I²C. Note a reboot maybe needed after the configuration. If desired, this can be done manually. Edit /etc/modules and add the following lines to the file. If the file is blank, ignore this step and move onto the next step. (Note you need to be root or use sudo privileges to edit this file.)

   i2c-bcm2708
   i2c-dev
   rtc-pcf2127Xxx

2) This step is needed for older versions of the Raspbian OS. Look at the file /etc/modprobe.d/raspi-blacklist.conf – if the file is blank skip this step. If there is content in this file, comment the lines as shown below by editing /etc/modprobe.d/raspi-blacklist.conf
   Comment the lines (by adding a # symbol at the beginning of the line) which blacklist I²C capabilities, as shown below:
   # blacklist i2c-bcm2708Xxx

3) The next step loads the necessary Python libraries. Please make sure you are connected to the network.

   sudo apt-get install python-smbus → Install the Python SMB routines
   sudo apt-get install i2c-tools → Install the I²C Python tools

4) A reboot maybe needed. It is recommended to do a reboot after installation is complete.
5) After this step, you should be able to use i2cdetect to check if the ADC board is recognized. The command for newer Raspberry Pi the I²C bus is designated number 1.
   a. I2cdetect -y 1 (note you may have to use sudo with this command.) You should see “76” when the board is connected properly.

b. If there is already another device on the I²C bus with the address 0x76, please follow instructions for changing the address in this document.

```
pi@raspberrypi~$ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 --
```

Figure 10 – i2cdetect output to show device address.

Essential step for continued use of Python with Pi-16ADC

Note – without this step being followed, the Pi-16ADC will not work or you will get an IO error. Please make sure this step is implemented before contacting Alchemy Power Inc.

The Pi-16ADC uses capabilities not yet supported on Pixel (or Raspbian) by default. To overcome that and to make sure the feature is supported on each reboot, please update /et/rc.local file to add lines shown below. Note to edit /etc/rc.local, you must use sudo privileges.

A typical /etc/rc.local file looks as follows.

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
```

```
fi

exit 0
```

Figure 11 – default /etc/rc.local file as distributed by Pixel or Raspbian.

Please add the following lines to /etc/rc.local and then reboot the Pi. The reboot is needed to activate the enhanced capabilities of I2C for Pi-16ADC.

```
# For Pi-16ADC
echo -n 1 > /sys/module/i2c_bcm2708/parameters/combined
# the above adds successive reads for I2C.
```

Figure 12 – Lines added to /etc/rc.local at the bottom of the file

A complete and ready to use /etc/rc.local will look as follows:

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi

# For Pi-16ADC
echo -n 1 > /sys/module/i2c_bcm2708/parameters/combined
# the above adds successive reads for I2C.

exit 0
```

Figure 13 A complete and ready to use /etc/rc.local file.

Analog to Digital Converter (ADC) background

The chip used for ADC conversion is the LTC2497 from Linear Technologies Inc. Please refer to the data sheet for the chip for detailed information on electrical characteristics as well as other relevant information. The data sheet is available at http://www.linear.com/product/LTC2497 as a PDF file.

The chip provides 16 ADC channels. Three of these channels are used to monitor input V, battery Voltage and the total current consumed. The fourth channel is reserved for future use.

ADC information is communicated to the Raspberry Pi using the I^2C bus. The bus operates at the default speed of 100kHz.

One solderless connection-point is for the Analog signal. The other solderless connection-point is for ground.

The maximum analog voltage supported is 2.5V peak-to-peak or from -2.5V to +2.5V for differential channels. Signals over (or under) that will be converted as an overflow and will be indicated as such. The sample python code checks for this situation and reports that.

The analog channels should also be connected to ground when not in use. If the analog channel is open or not connected properly (loose connection) the conversion process will indicate and over flow or an underflow situation.

Please connect the channels to the sensors before use. To prevent ESD discharge and damage to the board (or the Pi) please make sure that proper grounding protection is taken.

ADC Communications

Pi-16ADC uses the I^2C bus for communicating information from the ADC chip to Raspberry Pi. The I^2C bus operate at a default speed of 100kHz. This speed is used for communication. Higher speed is supported by the ADC chip, however higher speed has not been tested nor supported by Alchemy Power Inc. The LTC2497 communications are described in detail in the LTC2497 data sheet. Table 3 in the data sheet lists this information. The single-ended channel information is replicated here from the data sheets.

For single-ended channels, the following addresses are used to communicate with the channels.

Channel Number	Address
Channel 0	0xB0
Channel 1	0xB8
Channel 2	0xB1
Channel 3	0xB9
Channel 4	0xB2
Channel 5	0xBA
Channel 6	0xB3
Channel 7	0xBB
Channel 8	0xB4
Channel 9	0xBC
Channel 10	0xB5
Channel 11	0xBD

Table 2 – single-ended channel addresses.

For differential channels, please refer to the data sheet and use the appropriate channel information. For differential channels and single-ended channels, the information from the LTC2497 data sheet is replicated here for simplicity.

SDA column represents the byte to be written on to the SDA bus. This is the address to use to collect the data from a specific channel.

SDA (hex)	Ch 0	Ch 1	Ch 2	Ch 3	Ch 4	Ch 5	Ch 6	Ch 7	Ch 8	Ch 9	Ch 10	Ch 11	Ch 12	Ch 13	Ch 14	Ch 15
A0	In+	In-														
A1			In+	In-												
A2					In+	In-										
A3							In+	In-								
A4									In+	In-						
A5											In+	In-				
A6													In+	In-		
A7															In+	In-
A8	In-	In+														
A9			In-	In+												
AA					In-	In+										
AB							In-	In+								
AC									In-	In+						
AD											In-	In+				
AE													In-	In+		
AF															In-	In+
B0	In+															
B1			In+													
B2					In+											
B3							In+									
B4									In+							
B5											In+					
B6													In+			
B7															In+	
B8		In+														
B9				In+												
BA						In+										
BB								In+								
BC										In+						
BD												In+				
BE														In+		
BF																In+

Table 3 – All possible address combinations as well as command to be written to the I^2C bus.

Note – for addresses B0-BF, the common is connected to the ground.

Please use the ground pin next to the ADC channel pin. Other ground connections could pick up stray noise.

Also recommended practice is to use shielded twisted pair cable (from source of Analog signal to Pi-16ADC) to minimize stray noise.

The simplest form of communications using I^2C bus for single-ended channel is as follows:

1) Address the device at address 0x76 and send a write command to the required channel to initiate a conversion. So for example, if a conversion from the analog data to digital data is required on channel 1, this is done by the python directive “bus.write_byte(0x76, 0xB8)”

2) Provide sufficient time for the conversion to complete – usually around 0.2 seconds. This is done using the python directive "time.sleep(tiempo)" – where the variable tiempo is defined as number of seconds to sleep between successive readings.
3) Read the information from the channel. Usually multiple bytes of information are returned on the read. In the case of the LTC2497, a minimum of 3 bytes are returned. If there is a bus read error or other fault, the channel responded by a "NAK". If a NAK is returned, sleep again for another 0.2 seconds and repeat that till it is determined that there is something wrong (a time out.) Reading of the information is done by the directive "bus.read_i2c_block_data(0x76, channel#, # bytes to be read in)
4) Eliminate the first two bits returned which provide the voltage status. This information is provided in Table 1 in the LTC2497 data sheet.
5) Shift the bits appropriately to represent the value collected. Divide the value by 2^{16} or 65,535 to calculate the relative voltage. Multiply the result by Vref/2 or 2.5 to get the final voltage value.
6) Repeat the process with other channels. If the channels are used as a differential pair, use the appropriate address code to address the channel(s).

The same principle is used for the differential bus using the appropriate address of the signal pins used. All of the above is summarized in more details in the LTC2497 data sheet.

Sample Python code explained

To run the Python code, follow these steps.

1) Download the code from www.alchemypower.com and unzip the files.
2) Please make sure the i2c-tools and python-smbus libraries are installed as discussed earlier.
3) Define the variables to be used:
    a. address – this is the address of the device on the I^2C bus. To determine the address, use the command "i2cdetect -y 1" (or "y 0" for older Pi's.) You may have to precede the command with "sudo".
    b. Select the channels to collect data on. All 16 channels are available for the Pi-16ADC board.
        i. Note the channel can be a differential channel by selecting the proper pins and address as discussed earlier as well as in the LTC2497 data sheet.
    c. Define how often a reading should be taken. This is defined by the "zeit" (German – time) variable. This defines how long the software sleeps between each loop execution.
    d. Define the sleep time between each consecutive data channel. The minimum value for this is 0.2. This is defined by the "tiempo" (Spanish – time – noun) variable.
    e. Define the multiplier to use – normally this is "1". You can use a different multiplier to magnify a value to diminish a value.
4) In a loop (or a sequence), write the command to the I^2C bus initiate the conversion and then read the data collected. LTC2497 provides a means to read and get back the previous read in value all in one command. For example, a read command will respond back with the prior data in the register and trigger the next channel conversion. Please refer to application note "Easy

Drive Delta-Sigma ADCs Deliver Powerful Features and Reduce Design Effort" by Mark Thoren on the Linear Technology web site. See Bibliography.

5) When the three bytes are read in, eliminate the top two bits which show the status of the conversion. See Table 1 for the status of the conversion.
    a. Eliminate the top two bits by and operation with 0x3F in the first byte.
    b. Convert the remaining bits of information to a 16-bit integer by proper shifting and adding. This information is represented in "valor" (Spanish – value).
    c. Convert the integer to a voltage level by representing the proper voltage level by multiplying by "vref" and dividing by maximum value possible.
6) Repeat the process for other channels.
7) Run the code using "python <filename.py>".

Python Code

The code is documented. Should you have questions or suggestions you can send email to support@alchemypower.com

Feel free to use/distribute/change/modify the code as needed. Simply leave the header section in place for other users and the Alchemy Copyright notice as well as the right to change the code at any time.

```
pi@pi3 $ cat Pi-16ADC.py
#!/usr/bin/python

################################################################################
#
# IF YOU GET IO-READ ERROR - YOU ARE MISSING THE SUCCESSIVE READ
# CAPABILITY ENABLED ON THE PI.
# For I2C and SMBus Documentation see
# http://wiki.erazor-zone.de/wiki:linux:python:smbus:doc
#
# See repeated start discussion or successive read information at
# https://www.raspberrypi.org/forums/viewtopic.php?f=44&t=15840&start=25
#
# sudo su -
# echo -n 1 > /sys/module/i2c_bcm2708/parameters/combined
# exit
# sudo reboot
# File /sys/module/i2cbcm2708/parameters/combined changed from N to Y
#
# Add line
# echo -n 1 > /sys/module/i2c_bcm2708/parameters/combined
# in /etc/rc.local before the last line. After adding the line, reboot
# the pi.
#
#
################################################################################
# (C) ALCHEMY POWER INC 2016, 2017 etc. - ALL RIGHT RESERVED.
# CODE SUBJECT TO CHANGE AT ANY TIME.
# YOU CAN COPY/DISTRIBUTE THE CODE AS NEEDED AS LONG AS YOU MAINTAIN
# THE HEADERS (THIS PORTION) OF THE TEXT IN THE FILE.
################################################################################

import time
import smbus
import sys
import os, commands
import subprocess

from smbus import SMBus
from sys import exit

# for older PI's (version 1) use bus = SMBus(0) in statement below.
bus = SMBus(1)
#
# address is the address of the ADC chip.
# Use i2cdetect -y 1 to find the address. Use "y 0" for older Pi's.
#
```

```
# Depending on the jumper settings A0, A1 and A2, all possible combinations are shown
below.
# Please refer to LTC 2495 data sheet, available at wwww.linear.com
# Page 17 of the data sheet has this information.
# Also see the Pi-16ADC User Guide for more information.
#
#
# Uncomment the line to match jumper settings for your board.
#

#  I2C Addresses for 8/16 channels (LTC2495/7/9) - note the other chips are
# pin compatible replacements from Linear Technology.
#
#             ADDRESS     A2    A1     A0
# address =  0b0010100 #  LOW   LOW    LOW     0x14
# address =  0b0010110 #  LOW   LOW    HIGH    0x16
# address =  0b0010101 #  LOW   LOW    FLOAT   0x15
# address =  0b0100110 #  LOW   HIGH   LOW     0x26
# address =  0b0110100 #  LOW   HIGH   HIGH    0x34
# address =  0b0100111 #  LOW   HIGH   FLOAT   0x27
# address =  0b0010111 #  LOW   FLOAT  LOW     0x17
# address =  0b0100101 #  LOW   FLOAT  HIGH    0x25
# address =  0b0100100 #  LOW   FLOAT  FLOAT   0x24
# address =  0b1010110 #  HIGH  LOW    LOW     0x56
# address =  0b1100100 #  HIGH  LOW    HIGH    0x64
# address =  0b1010111 #  HIGH  LOW    FLOAT   0x57
# address =  0b1110100 #  HIGH  HIGH   LOW     0x74
#
address =  0b1110110 #  HIGH  HIGH   HIGH    0x76
#
# The above address - 0x76 is the default address for Pi-16ADC.
#
# address =  0b1110101 #  HIGH  HIGH   FLOAT   0x75
# address =  0b1100101 #  HIGH  FLOAT  LOW     0x65
# address =  0b1100111 #  HIGH  FLOAT  HIGH    0x67
# address =  0b1100110 #  HIGH  FLOAT  FLOAT   0x66
# address =  0b0110101 #  FLOAT LOW   LOW     0x35
# address =  0b0110111 #  FLOAT LOW   HIGH    0x37
# address =  0b0110110 #  FLOAT LOW   FLOAT   0x36
# address =  0b1000111 #  FLOAT HIGH  LOW     0x47
# address =  0b1010101 #  FLOAT HIGH  HIGH    0x55
# address =  0b1010100 #  FLOAT HIGH  FLOAT   0x54
# address =  0b1000100 #  FLOAT FLOAT LOW     0x44
# address =  0b1000110 #  FLOAT FLOAT HIGH    0x46
# address =  0b1000101 #  FLOAT FLOAT FLOAT   0x45
#
#

# Channel Address - Single channel use
# See LTC2497 data sheet, Table 3, Channel Selection.
# All channels are uncommented - comment out the channels you do not plan to use.
#
#
channel0        =     0xB0
channel1        =     0xB8
```

```
channel2        =     0xB1
channel3        =     0xB9
channel4        =     0xB2
channel5        =     0xBA
channel6        =     0xB3
channel7        =     0xBB
channel8        =     0xB4
channel9        =     0xBC
channel10       =     0xB5
channel11       =     0xBD
channel12       =     0xB6
channel13       =     0xBE
channel14       =     0xB7
channel15       =     0xBF
#
# Differential Channels below. Note if a differential channel is used, uncomment 2
channels above
# two channels make up a differential channel.
#
#
# The "+" and "-" signs show which channel has the positive voltage and the negative
voltage.
#
#
#channel0       =       0xA0  # Differential pair Channel 0 + and Channel 1 -
#channel2       =       0xA1  # Differential pair Channel 2 + and Channel 3 -
#channel4       =       0xA2  # Differential pair Channel 4 + and Channel 5 -
#channel6       =       0xA3  # Differential pair Channel 6 + and Channel 7 -
#channel8       =       0xA4  # Differential pair Channel 8 + and Channel 9 -
#channelA       =       0xA5  # Differential pair Channel A + and Channel B -
#channelC       =       0xA6  # Differential pair Channel C + and Channel D -
#channelE       =       0xA7  # Differential pair Channel E + and Channel F -
#
#channel0       =       0xA8  # Differential pair Channel 0 - and Channel 1 +
#channel2       =       0xA9  # Differential pair Channel 2 - and Channel 3 +
#channel4       =       0xAA  # Differential pair Channel 4 - and Channel 5 +
#channel6       =       0xAB  # Differential pair Channel 6 - and Channel 7 +
#channel8       =       0xAC  # Differential pair Channel 8 - and Channel 9 +
#channelA       =       0xAD  # Differential pair Channel A - and Channel B +
#channelC       =       0xAE  # Differential pair Channel C - and Channel D +
#channelE       =       0xAF  # Differential pair Channel E - and Channel F +
#
#
#

##############################################################################################
#
# Determine the reference voltage
#
##############################################################################################
vref = 2.5
##############################################################################################

# To calculate the voltage, the number read in is 3 bytes. The first bit is ignored.
# Max reading is 2^23 or 8,388,608
```

```
#

max_reading = 8388608.0

# Now we determine the operating parameters.
# lange = number of bytes to read. A minimum of 3 bytes are read in. In this sample
we read in 6 bytes,
# ignoring the last 3 bytes.
# zeit (German for time) - tells how frequently you want the readings to be read from
the ADC. Define the
# time to sleep between the readings.
# tiempo (Spanish - noun - for time) shows how frequently each channel is read in
over the I2C bus. Best to use
# timepo between each successive readings.
#
#

lange = 0x06 # number of bytes to read in the block
zeit = 5     # number of seconds to sleep between each measurement
tiempo = 0.4 # number of seconds to sleep between each channel reading

# tiempo - has to be more than 0.2 (seconds).


#=============================================================================
# This is a subroutine which is called from the main routine.
# The variables passed are:
# adc_address - I2C address for the device.
#         This is set using the jumpers A0, A1 and A2. Default is 0x76
# adc_channel - the analog channel you want to read in.
#
#=============================================================================
def getreading(adc_address,adc_channel):
        bus.write_byte(adc_address, adc_channel)
        time.sleep(tiempo)
        reading  = bus.read_i2c_block_data(adc_address, adc_channel, lange)
#----------- Start conversion for the Channel Data ----------
        valor = ((((reading[0]&0x3F))<<16))+((reading[1]<<8))+(((reading[2]&0xE0)))
# Debug statements provide additional prinout information for debuggin purposes. You
can leave those commented out.
# Debug
#       print("Valor is 0x%x" % valor)
#----------- End of conversion of the Channel ----------
        volts = valor*vref/max_reading

#######
# See table 1 of the LTC2947 data sheet. If the voltage > Vref/2, then it gives an
error condition....
#
# So lets check the first byte
#
#######

        if( (reading[0]& 0b11000000) == 0b11000000): # we found the error
                print "**************"
```

```
                print ("Input voltage to channel 0x%x is either open or more than
%5.2f. The reading may not be correct. Value read in is %12.8f Volts." %
((adc_channel), vref, volts))
                print "*************"
#       else:
                print (">>>Voltage read in on channel 0x%x is %12.8f Volts" %
((adc_channel),volts))
# Note - print is on stdout - so you can redirect that to a file if needed. You
# can also comment out the verbiage to suite your needs.

#       time.sleep(tiempo)
# If needed pause the reading..
#
# Note - the pause is usually put in the main routine and not subroutine.
# NAK conditions are not checked for in this sample code. You can add that if needed.
#
        return volts
#==================================================================================

time.sleep(tiempo)
# Initial startup - a recommendation is to sleep tiempo seconds to give ADC time to
stabilize.
# Found that it was worth it. Somehow Python and Pi get into a race situation without
it. You can
# expriment omitting the initial sleep.
#
ch0_mult = 1 # This is the multiplier value to read the Current used by the Pi.
ch1_mult = 1 # Multiplier for Channel 1

# Add more multiplier for each channel as needed.

# Main routine - shows an endless loop. If used with a cron script or a
# trigger via GPIO, an endless loop is not recommended.
#
# Here we show only channel 0 and channel 1 - you can all 16 in a loop or a
# simple waterfall code as below.
#
while (True):
        #   Read Channel 0
        Ch0Value = ch0_mult*getreading(address, channel0)
        print ("Channel 0 at %s is %12.2f" % (time.ctime(), Ch0Value))
        # Sleep between each reading.
        time.sleep(tiempo)
        ############## End of Channel 0 block ###################
        #   Read Channel 1
        Ch1Value = ch1_mult*getreading(address, channel1)
        print ("Channel 1 at %s is %12.2f" % (time.ctime(), Ch1Value))
        # Sleep between each reading.
        time.sleep(tiempo)
        ############## End of Channel 1 block ###################
        # Write the values read should there be an interrupt. Since output is to
stdout, it needs to be flushed
        # on exit or interrupt. Without that, readings could be lost.
        sys.stdout.flush()
        # Sleep till next reading.
```

```
        time.sleep(zeit)
# End of main loop.

pi@pi3$
```

Bibliography

1) Easy Drive Delta-Sigma ADCs Deliver Powerful Features and Reduce Design Effort – Mark Thoren, Linear Technology, http://cds.linear.com/docs/en/lt-journal/LTMag-V17N01-01-LTC24xx-Thoren.pdf
2) LTC2497 data sheet, Linear Technology Inc., Product information page found at http://www.linear.com/product/LTC2497

Product Video

Product Capabilities: https://youtu.be/gApMLcfn0A4

Contact

Address: 2098 Walsh Avenue, Suite A, Santa Clara, CA 95050.

Email Support: support@alchemypower.com

Email Sales: sales@alchemypower.com

Web site: www.alchemypower.com