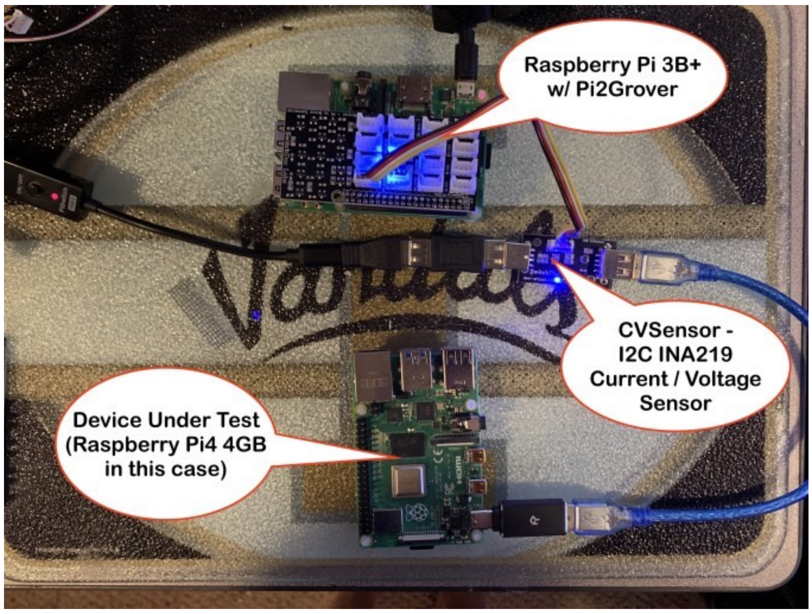
INA219 I2C USB/USB I & V Sensor

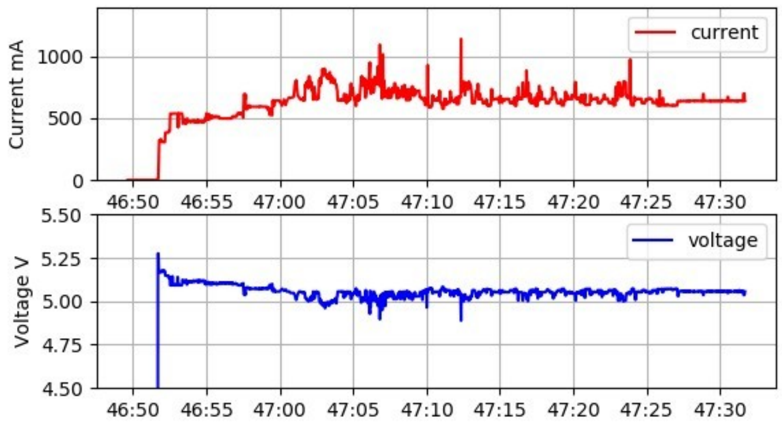
2020F Jun 09 (Ina219_08.odt)

by: chris.borrill@gmail.com (edits by D@CC)

Project description : pi-ina219 1.3.0

The USB CVSensor is an USB to USB current and voltage measuring device. It uses an INA219 to accurately (and dynamically) measured currents and voltages through a USB plug. It is available from SwitchDoc Labs.


Raspberry Pi 3B+ w/ Pi2Grover
CVSensor - I2C INA219 Current / Voltage Sensor
Device Under Test (Raspberry Pi4 4GB in this case)



Pi4 Turn On Current
Average Current 604.12mA
Peak Current 1140.98mA
Minimum Voltage 4.89V
2019-09-25 15:46:49.644000 UTC

USB CVSensor Specification (0047-USBCVSNSR-DSBT with Ina219)

•USB Type A Plugs

•Data+, Data-, Ground directly passed through

•INA219 Highside DC Current and Voltage Sensor

•Up to 2000 conversions per second

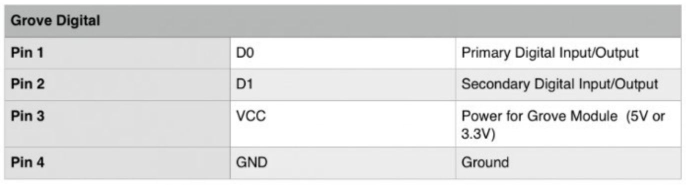
•0.1 ohm 1% 2W current sense resistor

•Up to ±3.2A current measurement, with ±0.8mA resolution

•Standard Grove I2C Connector to INA219

•Default I2C Address 0x45

RaspPi Pins		**Grove Pins**	
GPIO 3 SCL	**5**	**Yellow**	**Pin 1**
GPIO 2 SDA	**3**	**White**	**Pin 2**
GPIO 3V3	**1**	**Red**	**Pin 3**
GPIO Gnd	**9**	**Black**	**Pin 4**



Grove Digital		
Pin 1	D0	Primary Digital Input/Output
Pin 2	D1	Secondary Digital Input/Output
Pin 3	VCC	Power for Grove Module (5V or 3.3V)
Pin 4	GND	Ground

Build Status

codecov

PyPI version

This Python library supports the INA219 voltage, current and power monitor sensor from Texas Instruments on both Python 2 and 3. The intent of the library is to make it easy to use the quite complex functionality of this sensor.

The library currently only supports *continuous* reads of voltage and power, but not *triggered* reads.

The library supports the detection of *overflow* in the current/power calculations which results in meaningless values for these readings.

The low power mode of the INA219 is supported, so if only occasional reads are being made in a battery based system, current consumption can be minimised.

The library has been tested with the Adafruit INA219 Breakout.

Installation and Upgrade

This library and its dependency (Adafruit GPIO library) can be installed from PyPI by executing:

```
sudo pip3 install pi-ina219
```

To upgrade from a previous version installed direct from Github execute:

```
sudo pip3 uninstall pi-ina219

sudo pip3 install pi-ina219
```

The Adafruit library supports the I2C protocol on all versions of the Raspberry Pi. Remember to enable the I2C bus under the *Advanced Options* of *raspi-config*.

Usage

The address of the sensor unless otherwise specified is the default of *0x40*.

Note that the bus voltage is that on the load side of the shunt resistor, if you want the voltage on the supply side then you should add the bus voltage and shunt voltage together, or use the *supply_voltage()* function.

Simple - Auto Gain

This mode is great for getting started, as it will provide valid readings until the device current capability is exceeded for the value of the shunt resistor connected (3.2A for 0.1Ω shunt resistor). It does this by automatically adjusting the gain as required until the maximum is reached, when a *DeviceRangeError* exception is thrown to avoid invalid readings being taken.

The downside of this approach is reduced current and power resolution.

```
#!/usr/bin/env python

from ina219 import INA219

from ina219 import DeviceRangeError



SHUNT_OHMS = 0.1





def read():

    ina = INA219(SHUNT_OHMS)

    ina.configure()



    print("Bus Voltage: %.3f V" % ina.voltage())

    try:

        print("Bus Current: %.3f mA" % ina.current())
        print("Power: %.3f mW" % ina.power())

        print("Shunt voltage: %.3f mV" % ina.shunt_voltage())

    except DeviceRangeError as e:

        # Current out of device range with specified shunt
resistor

        print(e)



if __name__ == "__main__":

    read()
```

Advanced - Auto Gain, High Resolution

In this mode by understanding the maximum current expected in your system and specifying this in the script you can achieve the best possible current and power resolution. The library will calculate the best gain to achieve the highest resolution based on the maximum expected current.

In this mode if the current exceeds the maximum specified, the gain will be automatically increased, so a valid reading will still result, but at a lower resolution.

As above when the maximum gain is reached, an exception is thrown to avoid invalid readings being taken.

```python
#!/usr/bin/env python

from ina219 import INA219

from ina219 import DeviceRangeError



SHUNT_OHMS = 0.1

MAX_EXPECTED_AMPS = 0.2




def read():

    ina = INA219(SHUNT_OHMS, MAX_EXPECTED_AMPS)

    ina.configure(ina.RANGE_16V)



    print("Bus Voltage: %.3f V" % ina.voltage())

    try:

        print("Bus Current: %.3f mA" % ina.current())
        print("Power: %.3f mW" % ina.power())

        print("Shunt voltage: %.3f mV" % ina.shunt_voltage())

    except DeviceRangeError as e:

        # Current out of device range with specified shunt
resistor

        print(e)




if __name__ == "__main__":

    read()
```

Advanced - Manual Gain, High Resolution

In this mode by understanding the maximum current expected in your system and specifying this and the gain in the script you can always achieve the best possible current and power resolution, at the price of missing current and power values if a current overflow occurs.

```
#!/usr/bin/env python

from ina219 import INA219
from ina219 import DeviceRangeError


SHUNT_OHMS = 0.1

MAX_EXPECTED_AMPS = 0.2



def read():

    ina = INA219(SHUNT_OHMS, MAX_EXPECTED_AMPS)

    ina.configure(ina.RANGE_16V, ina.GAIN_1_40MV)


    print("Bus Voltage: %.3f V" % ina.voltage())

    try:

        print("Bus Current: %.3f mA" % ina.current())

        print("Power: %.3f mW" % ina.power())

        print("Shunt voltage: %.3f mV" % ina.shunt_voltage())
    except DeviceRangeError as e:

        print("Current overflow")




if __name__ == "__main__":

    read()
```

Sensor Address

The sensor address may be altered as follows:

```
ina = INA219(SHUNT_OHMS, MAX_EXPECTED_AMPS, address=0x41)
```

Low Power Mode

The sensor may be put in low power mode between reads as follows:

```
ina.configure(ina.RANGE_16V)

while True:

    print("Voltage : %.3f V" % ina.voltage())
    ina.sleep()

    time.sleep(60)

    ina.wake()
```

Note that if you do not wake the device after sleeping, the value returned from a read will be the previous value taken before sleeping.

Functions

- `INA219()` constructs the class. The arguments, are: _ shunt_ohms: The value of the shunt resistor in Ohms (mandatory). _ max*expected_amps: The maximum expected current in Amps (optional).
  - busnum: The I2C bus number for the device platform, defaults to *auto detects 0 or 1 for Raspberry Pi or Beaglebone Black* (optional).
  - address: The I2C address of the INA219, defaults to *0x40* (optional). * log*level: Set to _logging.INFO* to see the detailed calibration calculations and _logging.DEBUG* to see register operations (optional).

- `configure()` configures and calibrates how the INA219 will take measurements. The arguments, which are all optional, are: _ voltage_range: The full scale voltage range, this is either 16V or 32V, represented by one of the following constants (optional). _ RANGE*16V: Range zero to 16 volts
  - RANGE*32V: Range zero to 32 volts (**default**). **Device only supports up to 26V.**
  - gain: The gain, which controls the maximum range of the shunt voltage, represented by one of the following constants (optional). _ GAIN_1_40MV: Maximum shunt voltage 40mV _ GAIN*2_80MV: Maximum shunt voltage 80mV
  - GAIN*4_160MV: Maximum shunt voltage 160mV
  - GAIN*8_320MV: Maximum shunt voltage 320mV
  - GAIN*AUTO: Automatically calculate the gain (**default**)

  - bus*adc: The bus ADC resolution (9, 10, 11, or 12-bit), or set the number of samples used when averaging results, represented by one of the following constants (optional).
  - ADC*9BIT: 9 bit, conversion time 84us.
  - ADC*10BIT: 10 bit, conversion time 148us.
  - ADC*11BIT: 11 bit, conversion time 276us.
  - ADC*12BIT: 12 bit, conversion time 532us (**default**).
  - ADC*2SAMP: 2 samples at 12 bit, conversion time 1.06ms.
  - ADC*4SAMP: 4 samples at 12 bit, conversion time 2.13ms.
  - ADC*8SAMP: 8 samples at 12 bit, conversion time 4.26ms.
  - ADC*16SAMP: 16 samples at 12 bit, conversion time 8.51ms
  - ADC*32SAMP: 32 samples at 12 bit, conversion time 17.02ms.
  - ADC*64SAMP: 64 samples at 12 bit, conversion time 34.05ms.
  - ADC*128SAMP: 128 samples at 12 bit, conversion time 68.10ms.
  - shunt*adc: The shunt ADC resolution (9, 10, 11, or 12-bit), or set the number of samples used when averaging results, represented by one of the following constants (optional).
  - ADC*9BIT: 9 bit, conversion time 84us.
  - ADC*10BIT: 10 bit, conversion time 148us.
  - ADC*11BIT: 11 bit, conversion time 276us.
  - ADC*12BIT: 12 bit, conversion time 532us (**default**).
  - ADC*2SAMP: 2 samples at 12 bit, conversion time 1.06ms.
  - ADC*4SAMP: 4 samples at 12 bit, conversion time 2.13ms.
  - ADC*8SAMP: 8 samples at 12 bit, conversion time 4.26ms.
  - ADC*16SAMP: 16 samples at 12 bit, conversion time 8.51ms
  - ADC*32SAMP: 32 samples at 12 bit, conversion time 17.02ms.
  - ADC_64SAMP: 64 samples at 12 bit, conversion time 34.05ms. * ADC_128SAMP: 128 samples at 12 bit, conversion time 68.10ms.

- `voltage()` Returns the bus voltage in volts (V).

- `supply_voltage()` Returns the bus supply voltage in volts (V). This is the sum of the bus voltage and shunt voltage. A *DeviceRangeError* exception is thrown if current overflow occurs.
- `current()` Returns the bus current in milliamps (mA). A *DeviceRangeError* exception is thrown if current overflow occurs.
- `power()` Returns the bus power consumption in milliwatts (mW). A *DeviceRangeError* exception is thrown if current overflow occurs.
- `shunt_voltage()` Returns the shunt voltage in millivolts (mV). A *DeviceRangeError* exception is thrown if current overflow occurs.
- `current_overflow()` Returns 'True' if an overflow has occured. Alternatively handle the *DeviceRangeError* exception as shown in the examples above.
- `sleep()` Put the INA219 into power down mode.
- `wake()` Wake the INA219 from power down mode.
- `reset()` Reset the INA219 to its default configuration.

Performance

On a Raspberry Pi 2 Model B running Raspbian Jesse and reading a 12-bit voltage in a loop, a read occurred approximately every 10 milliseconds.

Debugging

To understand the calibration calculation results and automatic gain increases, informational output can be enabled with:

```
ina = INA219(SHUNT_OHMS, log_level=logging.INFO)
```

Detailed logging of device register operations can be enabled with:

```
ina = INA219(SHUNT_OHMS, log_level=logging.DEBUG)
```

Testing

Install the library as described above, this will install all the dependencies required for the unit tests, as well as the library itself. Clone the library source from Github then execute the test suite from the top level directory with:

```
python3 -m unittest discover -s tests -p 'test_*.py'
```

A single unit test class may be run as follows:

```
python3 -m unittest tests.test_configuration.TestConfiguration
```

Code coverage metrics may be generated and viewed with:

```
coverage run --branch --source=ina219 -m unittest discover -s
tests -p 'test_*.py'

coverage report -m
```

Coding Standard

This library adheres to the *PEP8* standard and follows the *idiomatic* style described in the book *Writing Idiomatic Python* by *Jeff Knupp*.

WRITING TO/READING FROM THE INA219
(pg 10 of the ti ina219.pdf data sheet. . .)
Accessing a particular register on the INA219 is accomplished by writing the appropriate value to the register pointer. Refer to Table 4 for a complete list of registers and corresponding addresses. The value for the register pointer as shown in Figure 17 is the first byte transferred after the salve address byte with the R/W bit LOW. Every write operation to the INA219 requires a value for the register pointer.

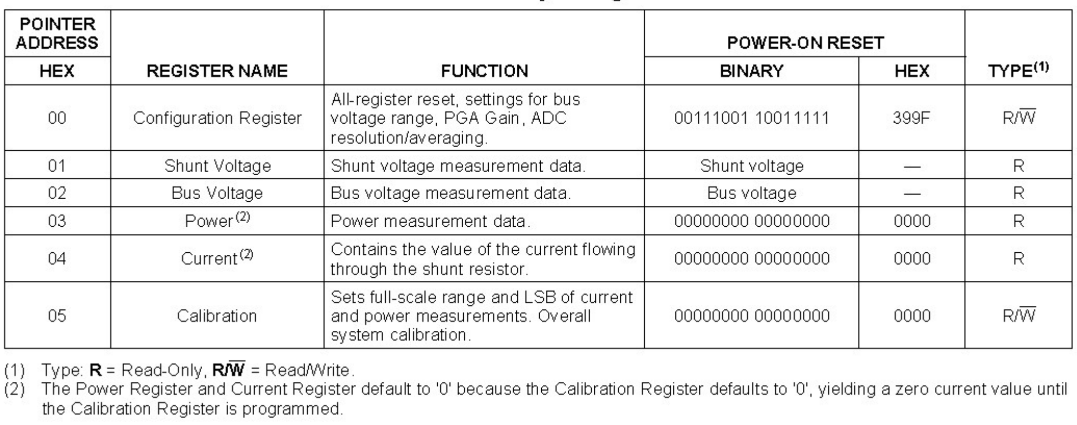


Table 4. Summary of Register Set

POINTER ADDRESS			POWER-ON RESET		
HEX	REGISTER NAME	FUNCTION	BINARY	HEX	TYPE[1]
00	Configuration Register	All-register reset, settings for bus voltage range, PGA Gain, ADC resolution/averaging.	00111001 10011111	399F	R/$\overline{W}$
01	Shunt Voltage	Shunt voltage measurement data.	Shunt voltage	—	R
02	Bus Voltage	Bus voltage measurement data.	Bus voltage	—	R
03	Power[2]	Power measurement data.	00000000 00000000	0000	R
04	Current[2]	Contains the value of the current flowing through the shunt resistor.	00000000 00000000	0000	R
05	Calibration	Sets full-scale range and LSB of current and power measurements. Overall system calibration.	00000000 00000000	0000	R/$\overline{W}$

(1) Type: **R** = Read-Only, **R/$\overline{W}$** = Read/Write.
(2) The Power Register and Current Register default to '0' because the Calibration Register defaults to '0', yielding a zero current value until the Calibration Register is programmed.

Simple Current Shunt Monitor Usage
(No Programming Necessary)   (p16 of the ti data sheet)

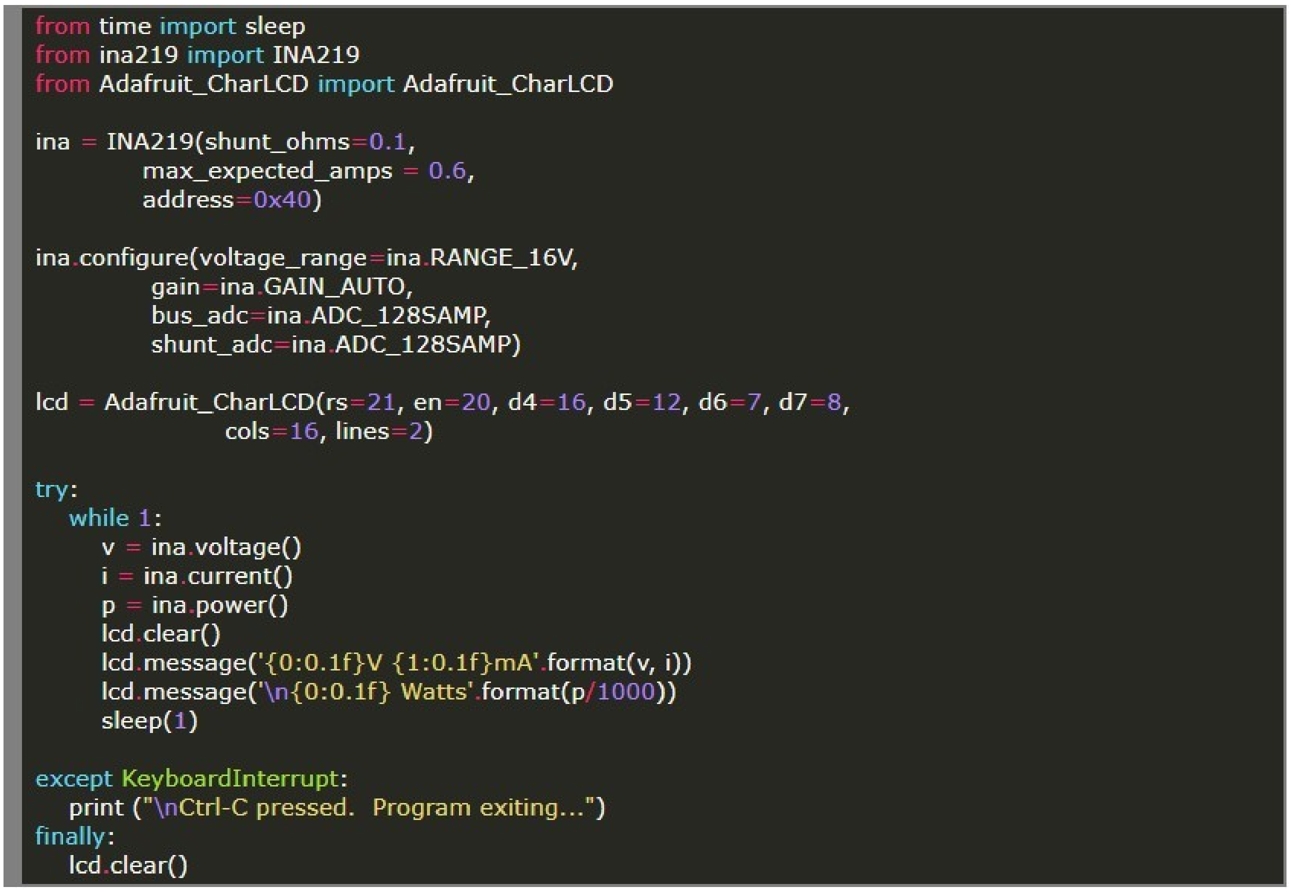
The INA219 can be used without any programming [Editor: see “Simple Code” below] if it is only necessary to read a shunt voltage drop and bus voltage with the default 12-bit resolution, 320mV shunt full-scale range (PGA=÷8), 32V bus full-scale range, and continuous conversion of shunt and bus voltage.

Without programming, current is measured by reading the shunt voltage. The Current Register and Power Register are only available if the Calibration Register contains a programmed value.

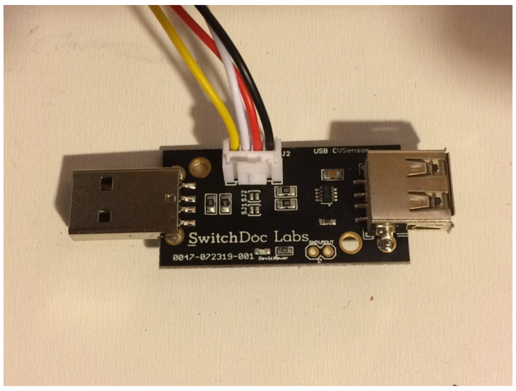
-end of text from the ti ina219 data sheet-

RASPBERRY PI INA219 TUTORIAL by rdagger68

Source 09 by rdagger68 describes in detail how to use a very simple program to read the bus voltage and shunt current from a ina219. This ultra-simple code is shown below.


```
from time import sleep
from ina219 import INA219
from Adafruit_CharLCD import Adafruit_CharLCD

ina = INA219(shunt_ohms=0.1,
             max_expected_amps = 0.6,
             address=0x40)

ina.configure(voltage_range=ina.RANGE_16V,
              gain=ina.GAIN_AUTO,
              bus_adc=ina.ADC_128SAMP,
              shunt_adc=ina.ADC_128SAMP)

lcd = Adafruit_CharLCD(rs=21, en=20, d4=16, d5=12, d6=7, d7=8,
                       cols=16, lines=2)

try:
    while 1:
        v = ina.voltage()
        i = ina.current()
        p = ina.power()
        lcd.clear()
        lcd.message('{0:0.1f}V {1:0.1f}mA'.format(v, i))
        lcd.message('\n{0:0.1f} Watts'.format(p/1000))
        sleep(1)

except KeyboardInterrupt:
    print ("\nCtrl-C pressed.  Program exiting...")
finally:
    lcd.clear()
```

The above code displays the matching voltage and current readings from the ina219 on an LCD. But it is very simple to remove all lines of code beginning with “lcd” and replace some of them with the equivalent python “print()” statements. This code will function well with either the Adafruit INA219 board or the similar board from SwitchDoc Labs.   This is because they both contain an INA219 with a shunt resistor of .1 ohms. Our thanks to rdagger68 for this code. He acknowledges the outstanding work that was done by ChrisB2 to create the ina219 library referenced in Source 01. The article by rdagger68 describes how to set up and use the INA219 in the simplest way possible. This approach shows how to operate the UBA219 effectively, not needing the “programming mode” as described in detail in the “ti” data sheet.



Compared to the Adafruit INA219 board, I prefer the ina219 board (shown above) from SwitchDoc Labs (Source 02) because it is equipped with 2 USB Connectors; one to accept the supply voltage and the other to feed the device whose current is to be monitored. The third connector (white plastic called a Grove connector) provides the 4 I2C signals leading to the Raspberry Pi that is taking the measurements. A matching 4 wire Grove cable can also be purchased to link it to the Raspberry Pi. The articles in Source 03 and Source 04 describe how to connect other devices that are also available from Grove. The color-coding used in the Grove wires and connectors is described at the very beginning of this article (on page 2).

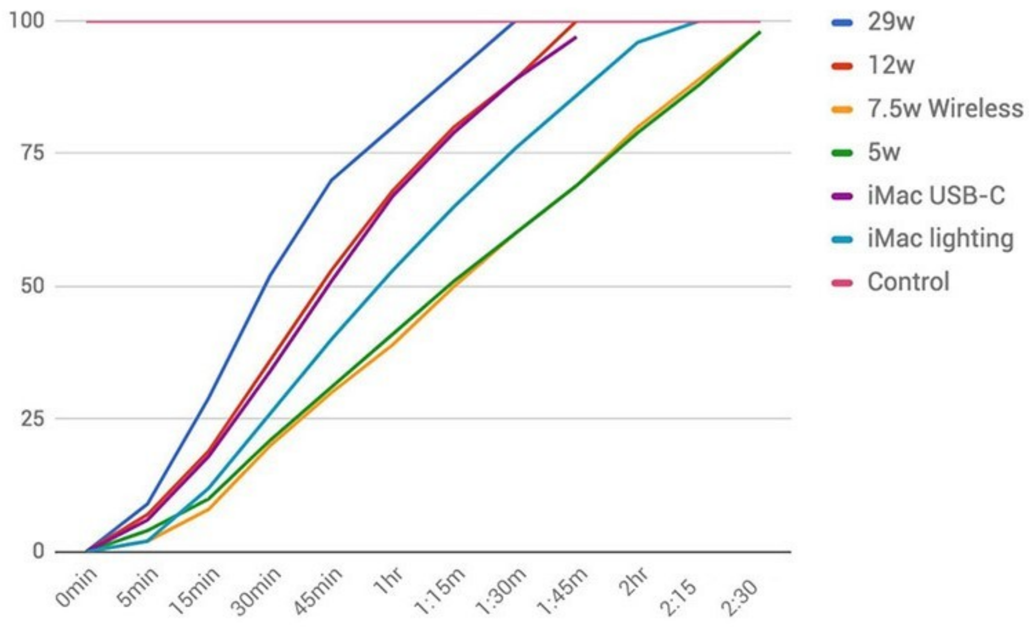
Beware of the Unsuccessful Software described in Source 05. Apparently, no example of functioning code was ever published.

Source 06 describes a Bachelor's thesis written in Spain that uses an INA219. Sources 10 and 11 refer to a website and article(s) written by David@ColeCanada.com (D@CC).

Measurements using the ina219
by D@CC on 2020F Jun 09

Device	**Volts**	**mAmps**	**Watts**	**Equiv. Resistance**
no device	5.1V	1.2mA	0.0 Watts	4,250. ohms
iPhone (100% charged)	5.1V	78.8mA	0.9 Watts	65.38 ohms
iPhone (89% charged)	4.9V	358.6mA	1.8 Watts	13.6 ohms

Compare these numbers with the charging curve below (as published in Source 12).



iphone-8-charging-speeds (See Source 12)

WEB SOURCES

Source 01 : https://pypi.org/project/pi-ina219/ at pypi.org by Chris Borrill 2019K Nov 09
Source 02 : http://www.SwitchDoc.com (SwitchDoc Labs)
Source 03: https://www.seeedstudio.com/blog/2016/03/09/tutorial-intro-to-grove-connectors-for-arduinoraspberry-pi-projects/ Seeedstudio.com
Source 04: http://www.seeedstudio.com/blog/2016/03/10/tutorial-grove-connector-project-examples-for-raspberry-pi-arduino/ Grove Fan Connection
Source 05: https://www.raspberrypi.org/forums/viewtopic.php?t=212387 Unsuccessful Software
Source 06: https://upcommons.upc.edu/bitstream/handle/2117/180533/tfg-report-pol-planas.pdf
Power consumption datalogger based on Python and Raspberry Pi
(Bachelor's Thesis by Pol J. Planas Pulido Fall 2019 for Director: Manuel Moreno Eguílaz Enginyeria Industrial de Barcelona)
Source 07: https://hexdocs.pm/ina219/INA219.html#content Unknown programming language
Source 08: https://github.com/chrisb2/pi_ina219 pi-ina219 1.3.0 by ChrisB2
Source 09: https://www.rototron.info/raspberry-pi-ina219-tutorial/ INA219 Simple Code by
rdagger68 on 2017B Feb 10
Source 10: http://www.mehincharge.com/ www.MehInCharge.com web site by D@CC to log many
various Acquisition/Control readings from devices such as the INA219.
Source 11: http://ephotocaption.com/a/130/130.html Electronics for the Raspberry Pi Computer
by D@CC listing various electronic sensing devices such as the INA219.
Source 12: https://appleinsider.com/articles/17/10/03/comparing-iphone-8-charging-speeds-with-fast-charge-wireless-and-more iphone-8-charging-speeds by Max Yuryev in 2018.

/Documents/2020/SwitchDocLabs/Ina219_08.odt