TronicsBench

Easily Learn how to use Microcontrollers in Your Projects

is

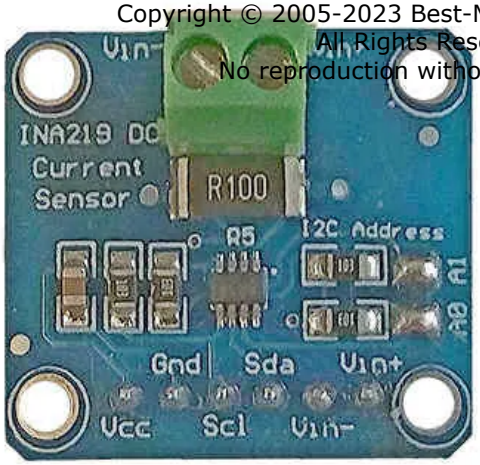
INA219: Simple Arduino current measurement up to 3.2A. How your Arduino can measure 3.2A at 100uA resolution, measuring the voltage and power consumption at the same time, using an I2C chip.

The INA219 :

- is a **shunt resistor** current measuring chip.
- uses a simple **I2C** interface.
- has a resolution of **100uA** (with a 0.1R shunt resistor).
- generates **voltage and power** values.

This chip is a current measuring chip which can also measure voltage - actually it can only measure voltage, but cheats by measuring voltage across a known resistance.

The breakout board version below can measure ~3.2A but the chip can measure 10~15A using a different sense resistor (See the datasheet).



The really useful feature of this chip is that it can measure **current and voltage** in an **external system** independently from the 5~3V0 that the chip uses. You can connect it directly the measurement inputs with voltage sources from 0~26V and it does not matter which one powers up first.

It also does not matter which way round the current measuring inputs are connected as they are protected to ±26V, however, you won't get a reading for negative input (wrong way round).

It also has an internal amplifier which means you don't need any other opamp power supplies for level conversion or amplification - just connect it up and go!

Recent Articles

Since it can measure current and voltage it can report the power used. It is an I2C device using an external shunt sense resistor which is attached to the high side voltage. The negative analogue input is used to measure the load voltage and hence, with the current measurement, the power dissipated in the load.

INA219 Specification

Parameter	INA219
Voltage Supply (Vs)	3V0 ~ 5V5

Readers Comments

"I wanted to thank

Interface	I2C
I2C rate	100,400,2560kHz
Resolution (ADC)[5]	12 bit (15 bit)
Quiescent current (typ,max)	0.7mA, 1mA
Power down mode current (typ,max)	6, 15uA
Current error % A(typ,max) B(typ,max) [2]	(**±0.2**,±0.5) (±0.2,±0.3)
Current error over full temperature (A,B) [2]	1%,0.5%
Voltage error % A(typ,max) B(typ,max) [2]	(**±0.2**,±0.5) (±0.2,±0.5)
Voltage error over full temperature (A,B) [2]	1%,1%
DNL	0.1LSB
Offset voltage (PGA =1,2,4,8)[2] (typ)	10,20,30,40uV
Offset voltage (PGA =1,2,4,8)[2] (max)[3]	100,125,150,200uV
Offset voltage drift	0.1uV/°C
Gain error	40m%
Conversion time (9bit res. ~ 12bit res., typ)	84~532us
I2C Addresses (h/w selected = 16off)	0x40 ~ 0x4F
V_{ESD} (Human body model)	±4000V
Operating temperature	-40°C ~125°C

[1] The total voltage between +Ain and -Ain must not exceed 0.3~26V.
[2] The device comes in two accuracy standards A, B with B more accurate.
[3] Offset voltage is lower for the B part - see datasheet.
[4] Inputs are reversible but you can only measure if +Ain > -Ain.
[5] 15 bit resolution is achieved by averaging for ±320mV range.

How to Wire INA219

How to connect INA219: It's simple just connect Vin+ to your supply voltage (this is the high side voltage), and connect Vin- to your circuit.

Vin- also measures the voltage appearing at the attached circuit.

INA219 Accuracy

Current Accuracy

The current measurement accuracy of the chip itself is very good but also depends on the accuracy of the sensing resistor, and typically this will be a 1% resistor.

At 25°C, if you assume a 1% resistor then the total current accuracy is 1.2% (typ) 1.5% (max) for part A .

Over the full temperature range the current accuracy is 1% so the total will be 2% (again for part A) - 1.5% for part B.

Voltage Accuracy

INA219 Resolution

The resolution of the device is a little strange as the physical ADC bit width is 12 bits, but throughout the data sheet the bit range discussed as 15bits. What is actually going on is that in some modes you can apply averaging (you can program the number of averages) and obtain better resolution.

Non averaged resolution

The resolution is set by the minimum detectable voltage and this is 1 LSB of the ADC which is fixed at 10uV. This is an internally set value controlled in the chip. The bus voltage resolution is set at 4mV.

Therefore minimum detectable current = 10uV/ Shunt resistor.

Typically RSHUNT = 0R1 so...

Minimum detectable current (12 bit res.) = 10uV/0.1R = 100uA.

However you can increase the bit resolution (lower current values than the ADC is capable of detecting) by averaging.

Normally the shunt resistor is 0.1Ohms, as these are commonly fitted to breakout boards. However, you can select any shunt resistor e.g. 2mOhm and the current resolution will change i.e. you can calculate it as above.

This is not the only way to set resolution - you can do it using the calibration register. In an example below the resolution is set to 50uA. The chip does some multiplication and gives a corresponding result.

The calculation lets you decide what the maximum expected current will be, and then sets the calibration "multiplier" to give a maximum "Full Scale Reading" for that current. So the resolution is "altered" to fit the FSR (actually the result registers are "altered").

Resolution with Averaging

When you oversample and decimate an ADC reading it is possible to increase the number of ADC bits, increasing the resolution of a signal without adding more hardware. Here oversampling and decimation is used to increase the number of bits available.

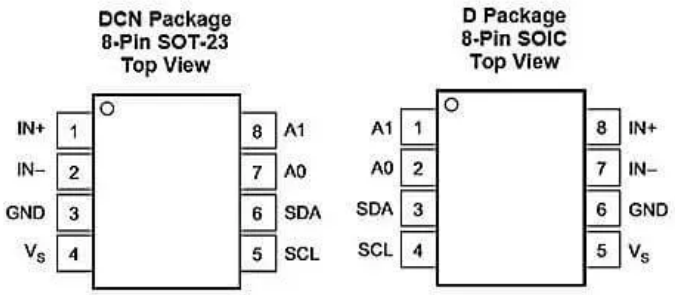
The number of bits returned from the ADC can be increased to 15 using this technique. This is why equations below reference 2^{15}.

There are 4 gain settings (1,2,4,8) for the current measurement, with 4 ranges (40mV, 80mV, 180mV and 320mV)

INA219 Datasheet

Download the INA219 datasheet here.

INA219 Pinout



[Source: Datasheet]

Default settings (Simple use)

The following explanation does not follow the process outlined in the datasheet (using equations provided in the datasheet you can find that here). It does use one statement from that datasheet:

> "The Calibration Register can also be selected to provide values in the Current Register (04h) and Power Register (03h) that either **provide direct decimal equivalents of the values being measured**, or yield a round LSB value for each corresponding register."

The idea is to use the minimal (I2C) read and write, and obtain an output that can be processed using fixed point maths - and this will save Flash memory.

If you don't program the chip then these are the defaults:

PG Programmable Gain (8)	Shunt voltage ±320mV.
Resolution	12 bit.
Bus Range	32V
Mode	Shunt and Bus continuous.

The datasheet indicates that without programming you can read the shunt voltage as

Since you know the shunt resistance value then the current is simply:

$I = V_{SHUNT}$ / Resistance

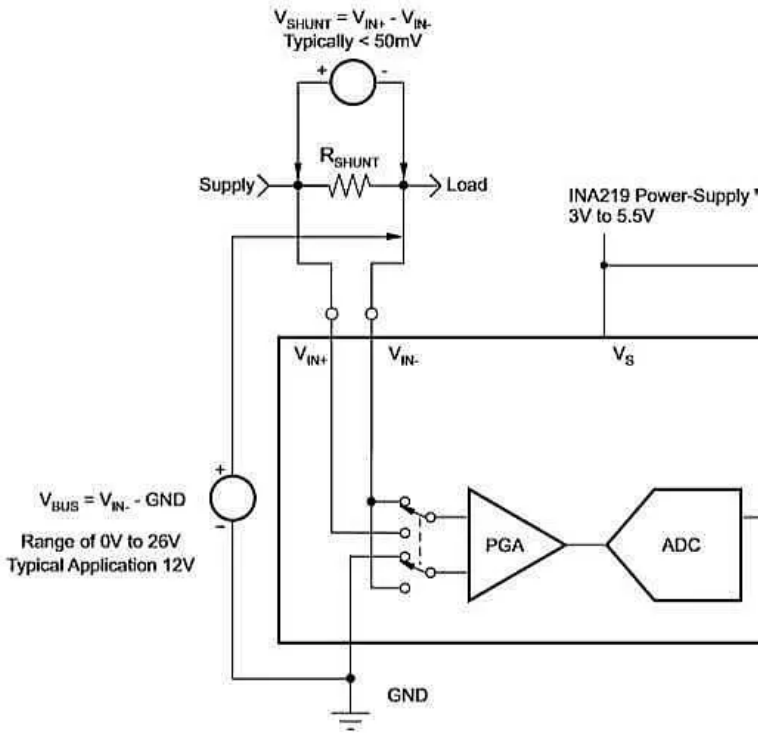
You can just make this calculation in the microcontroller and save messing around. For the power you just multiply the bus voltage reading (V_{BUS}) by the current value calculated.

For this action use the simplest code (using library INA219_WE) to read the registers (see the code and explanation to do this here).

Note: The code in the previous code link also sets the sample number to 128 for increased accuracy and resolution.

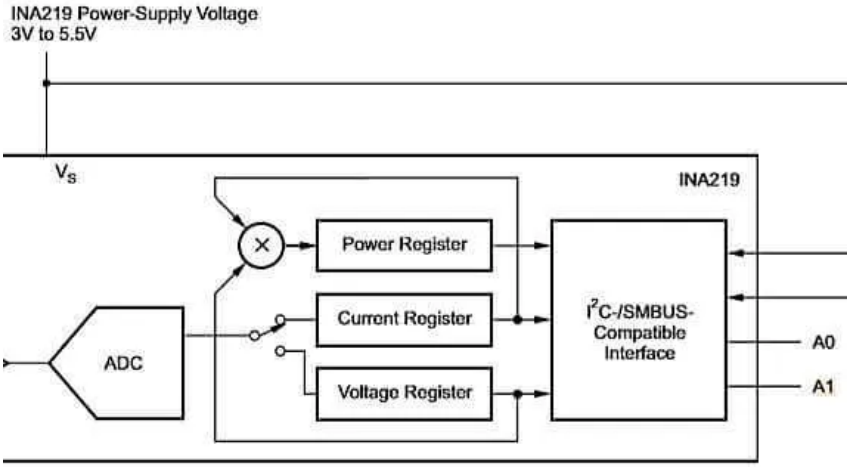
How the INA219 works

Inside the INA219 there is an ADC which can only measure voltage; Specifically the differential input V_{IN+} - V_{IN-}, and using internal switches it also measures V_{IN-} referenced to ground (using the same amplifier).



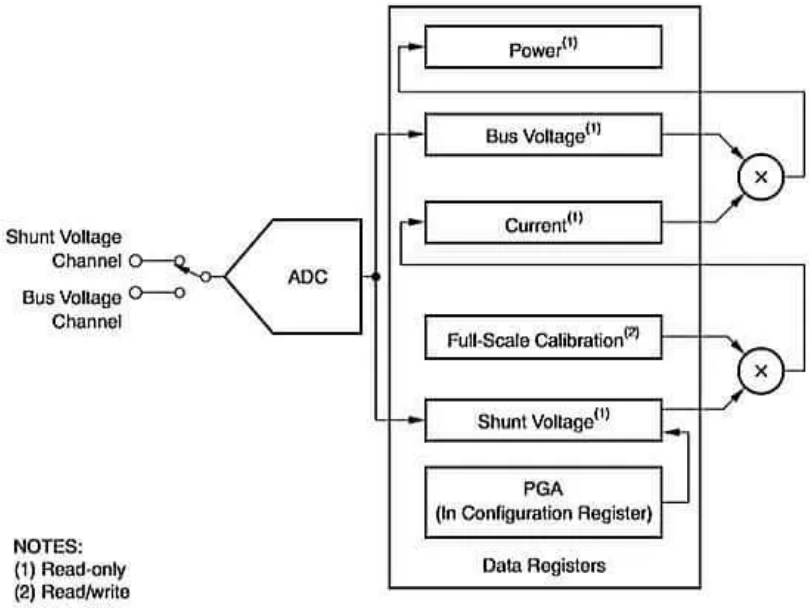
[Source: datasheet]

The INA219 cannot measure current unless it knows the value of the external shunt resistance. This why there is are programmable registers for configuration and calibration (See Table 2 in the datasheet "Summary of Register Set").



[Source: datasheet]

These registers and their operation are shown in the diagram below:



[Source: datasheet]

Programming Current and Power

To get the chip to return current and power values, you have to program in the value of the shunt resistance (unless you follow this method). You do this by programming the calibration register.

Using INA219 Equations

The following paragraphs from the datasheet talk a lot but are not easy to understand. If you just apply them then you get the result they suggest (they do work but require floating point multiplication to process the results).

The exception is if you are designing a bespoke system where you need a specific current resolution and/or range - then you will have to understand internal chip operation.

One issue is: To process the register results from the INA219 chip you have to use floating point calculations to output a meaningful result.

The INA219 does not output useful data directly. So, if you have to use floating point anyway, then the internal calculation processing within the chip seems a little redundant.

In fact, for space constrained chips, you can save 3kbytes of memory by not using floating point and instead use fixed point as explored later.

There seems to be no point in letting the chip do half the work and the microcontroller do the rest; other than the fact that it relieves the microcontroller from a little processing. The other reason could be that the chip operates in the background so doing a minimal calculation may be quicker when retrieving results.

Chip equations

Note: The text (below) "...**provide direct decimal equivalents of the values being measured**" is not easy to figure out. That text means you can directly read the current register for a measurement in amps. This is not usually done since the equations are complex to follow.

Datasheet paragraphs:

> "The Calibration Register enables the user to scale the Current Register (04h) and Power Register (03h) to the most useful value for a given application. For example, set the Calibration Register such that the largest possible number is generated in the Current Register (04h) or Power Register (03h) at the expected full-scale point"
>
> "This approach yields the highest resolution using the previously calculated minimum Current_LSB in the equation for the Calibration Register."
>
> "The Calibration Register can also be selected to provide values in the Current Register (04h) and Power Register (03h) that either **provide direct decimal equivalents of the values being measured**, or yield a round LSB value for each corresponding register."

Then the equations follow:

$$cal = trunc\left[\frac{0.04096}{CurrentLSB * Rshunt}\right]$$

$$CurrentLSB = \frac{Maxiumexpectedcurrent}{2^{15}}$$

$$PowerLSB = 20 * CurrentLSB$$

$$CurrentRegister = \frac{Shuntvoltageregister * calibrationregister}{4096}$$

$$PowerRegister = \frac{Currentregister * Busvoltageregister}{5000}$$

$$CorrectedFullScaleCal = \left[\frac{Cal * Measuredshuntcurrent}{INA219Current}\right]$$

Other simpler equations from text in the datasheet

RealShuntVoltage = ShuntVoltageRegister * 10e-6

RealBusVoltage = BusVoltageRegister * 4e-3

From these datasheet statements:

> "Shunt voltage is calculated by multiplying the Shunt Voltage Register contents with the shunt Voltage LSB of 10uV."
>
> "The bus voltage bus voltage register (shifted right 3 bits) is multiplied by the Bus Voltage LSB of 4mV to compute the bus voltage.".

So you can calculate the real voltages using the following equations:

```
Vshunt = Shunt_register * 10uV
Vbus   = (Bus_register>>3) * 4mV
```

Using Standard Equations

The data sheet goes through use of the equations to give an optimal resolution for the LSB of the ADC when you use the maximum expected current. When you set this arbitrary limit the resolution of the result is optimised.

Here's the process for choosing a 16V maximum and 400mA limit (the same as one example in the Adafruit library code).

```
Rshunt = 0.1
VshuntMax = 40mV (Choose this by setting the PGA gain).
VbusMax = 16V (Select voltage range 23V or 16V)

MaxPossibleCurrent = VshuntMax/Rshunt = 0.4A
MaxExpectedCurrent = 400mA
```

The following gives example current for maximum resolution (averaging enabled) and 12 bit operation. Setup for 12bit or averaging are selected by programming the registers.

Adafruit chose 50uA.

CurrentLSB = 50uA

Now you can figure out the cal register contents using the provided equation:

CalReg = trunc(0.04096 / (CurrentLSB * Rshunt))
CalReg = trunc(0.04096 / (50e-6 * 0.1)) = 8192.0

At this point the device will return values that you can read. The problem is that they need to be processed by the microcontroller to give actual values of current and power. (This is why the above process seems a little pointless i.e. why build a chip that does all the processing in the background but then requires you to do floating point operations to get the "real" values out. See fixed point operation on how to do it using the raw values. Note the only real reason is that you get optimal resolution - so maybe that is a good reason!).

Real Current Computation

Calculate register correction factors for real value output:

The full scale Current output is the current register value multiplied by the CurrentLSB (chosen earlier) i.e. multiply by 50e-6 giving a result in Amps.

realCurrent = CurrentReg * 50e-6

A more convenient result is mA so multiply CurrentLSB by 1000 to get the current output in mA:

realCurrent = CurrentReg * 50e-6* 1000.

However it is easier to turn it upside-down (instead of multiplying by a fraction divide by an integer):

realCurrent = CurrentReg / (1000 * 50e-6) = CurrentReg / 20

Assume you have the full current flowing and 40mV across Rshunt then:

ShuntVoltageReg = ShuntVoltage / 10e-6 = 40e-3/10e-6 = 4000

Current reg = ShuntVoltageReg *calreg / 4096 = 4000 * 8192 / 4096 = 8000

Then

realCurrent = 8000 / 20 = 400 = 400mA

To get the real current you divide the "Current register value" by 20 (in the microcontroller). Note that this value of 20 is a direct result of choosing a round number for CurrentLSB.

Real Power computation

PowerLSB = 20 *CurrentLSB

PowerRegister = CurrentReg * BusVoltageReg / 5000

Assume 400mA flows, (power will be 0.4*10 = 4W)

RealBusVoltage = 10V

busVoltageRegister = RealBusVoltage/ 4e-3 = 10 / 4e-3 = 2500
(Since the datasheet says 4mV per bus voltage LSB).
CurrentReg = 8000 (as before)

so

PowerRegister = (8000 * 2500) / 5000 = 4000

PowerLSB = 20 * 50e-6 = 0.001W per LSB

RealPower = 0.001 * 4000 = 4 = 4W

To get the real power you multiply by 0.001 (in the microcontroller) or you can forget the 0.001 multiplication leaving the result in mW.

RealPower = 4000mW

Using the simplest equations

This section shows how to use the equations to derive voltage and current with minimal programming and without using floating point. This can save a lot of memory - for instance if you use a ATTiny85 saving memory is paramount.

Simple Current

The easiest way to get the current reading is to use the shunt voltage register directly. The equation is:

RealShuntVoltage = ShuntVoltageRegister * 10e-6

Each LSB of the shunt voltage register is worth 10uV . What you need is the current so (when Rshunt = 0.1Ohm):

Current = V/R = (ShuntVoltageRegister * 10e-6) / Rshunt
Current = (ShuntVoltageRegister * 1e-4)

The 1e-4 value is saying the decimal point is four places to the left (see below for examples).

If the full maximum value of the shunt voltage is dropped across the shunt resistor (320mV) then it means that 3.2Amps is flowing (0.32V/0.1Ohms=3.2A). The value in the current register will be 32000 (Datasheet: Table 7).

Taking this value as a fixed point value means that the decimal point is 4 places to the left so:

For a current of 3.2A, the shunt register will contain 32000:

32000 -> 3.2000A

Some more examples:

For a current of 10mA, the shunt register will contain 100:

00100 -> 0.0100A

For a current of 1mA, the shunt register will contain 10:

00010 -> 0.0010A

Warning: This technique only works easily for Shunt resistances with a power of 10. e.g. 0.1Ohm, 10mOhm, 1.0 Ohm etc.

These register values can easily be processed without floating point. For instance you could make a limit detect at 300mA just by detecting if the value is greater than 3000.

You can also display these values by converting them to an ASCII representation e.g. using itoa, and then displaying leading zeros as necessary. One example is shown here: ina219-compare.ino.

Bus Voltage

The bus voltage conversion has an LSB value of 4mV. The bus register is shifted left by 3 bits. You could shift it right by 3 bits then multiply by four. However, conveniently, two shifts to the left are equivalent to a x4 operation so all you need to do is shift the bus register right by 1 bit!

This is because the register value is already left shifted by 3 bits due to how the hardware works. Don't forget to 'AND' the result with 0xFFF8 to zero the lower three bits before using it.

Getting the values

Code Warning: Both libraries do not allow you to read or write registers directly - this is a problem in C++ as the designer cannot foresee the usage of all code so it gets hacked to give the required functionality making the encapsulation fall apart (even if done properly you end up with multiple classes just to do a simple action making code maintenance an absolute pain) - lets just get the code done, test it and the encapsulate it later (for all the 100 other employees).

Library Code Warning

Warning: Neither library allows you to read or write registers directly.

So we'll have to get read access to the chip register in question.

For INA219_WE its quite easy - just make the read and write I2C routines public (or if feeling squeamish re-write the two read/write functions in c) and use the #defines to access the register values - use at least some of the hard work in the library!)

```
public:
        void writeRegister(uint8_t reg, uint16_t val);
        uint16_t readRegister(uint8_t reg);

private:
        INA219_ADC_MODE deviceADCMode;
        INA219_MEASURE_MODE deviceMeasureMode;
        INA219_PGAIN devicePGain;
        INA219_BUS_RANGE deviceBusRange;
        int i2cAddress;
        uint16_t calVal;
        uint16_t calValCorrected;
        uint16_t confRegCopy;
        float currentDivider_mA;
        float pwrMultiplier_mW;
        bool calc_overflow;
//      void writeRegister(uint8_t reg, uint16_t val);
//      uint16_t readRegister(uint8_t reg);
```

To get the current measurement in fixed point form (See above) we only need:

```
        ina219.readRegister(INA219_SHUNT_REG);
```

To output a useful value for display requires a little bit of formatting (and knowing where to place the fixed point). See ina219-compare.ino for how to do this.

Total (simple) code for this operation is :

```
#include <Wire.h>
#include <INA219_WE.h>

INA219_WE ina219(0x45);

void setup(void) {
   Wire.begin();
   Serial.begin(119200);

   ina219.init();
   ina219.setADCMode(SAMPLE_MODE_128); // Maximum averaging, ~78ms.
}

void loop(void) {

   int num = ina219.readRegister(INA219_SHUNT_REG);

   Serial.print("raw VS "); Serial.println(num);

   delay(1000);
}
```

[ina219.ino]

Code size:

Flash: 4182 Bytes (13%) , SRAM 431 Bytes (20%).

Comparison Sketch Example

Just to illustrate the readings you get using fixed point the following sketch shows "normal" use of the library compared to, "raw register values", and "fixed point" formatting:

Copy Sketch

```
#include <Wire.h>
#include <INA219_WE.h>

INA219_WE ina219(0x45);

void setup(void) {
   Wire.begin();
   Serial.begin(119200);

   ina219.init();
   ina219.setADCMode(SAMPLE_MODE_128);
}

// Print fixed point with dp decimal places to left.
void printFixedPointdp(long v, int dp) {

   long dpdiv = 1;
   for(int i=0;i<dp;i++,dpdiv*=10);

   long left = v / dpdiv;
   long rght = v % dpdiv;
   if (left==0) Serial.print('0');  else Serial.print(left);
   Serial.print('.');
```

[ina219-compare.ino]

This is the typical output from the sketch above:

```
float--------
I  15.5000mA
VS 1.5500mV
VB 4.7880V
P  76.0000mW
RAW--------
raw VS 155
raw  I 1550
    BV 4788
     P 7421400
Fixed------
 I 15.50mA
VS 1.55mV
BV 4.788V
 P 74.21400mW
```

Resolution Bits and Sampling Trade Off

By averaging any ADC reading you can increase the number of bits available. This chip can do that for you and you can have an average of up to 128 samples.

You can select a lower resolution result to give a faster read rate or you can allow the chip to oversample which results in a more accurate reading - but the read rate is far slower. The table below shows the rate for the lower bit numbers:

Bits/Samples	Conversion Time (typ)	Conversion Time (max)
9 bit	84us	93us
10 bit	148us	163us
11 bit	276us	304us
12 bit	532us	586us
2 samples	1.06ms	1.17ms

You can also use sample rates 4, 8, 16, 32, and 64. To find the conversion time simply multiply the sample rate chosen by the 12bit sample time e.g. 128 samples gives 128 * 532e-6 = 0.068096.

Although it does not say so in the datasheet the sampled results are sampled at 12bit resolution - it is implied by the timings.

The number of bits obtained does not directly map to increased sample rate but you can reduce the noise using higher sample rates.

Bit Range and PGA Gain

The total number of bits collected depends on the PGA gain selected:

PGA Gain	ADC Bits (n)	FSR (mV)	LSB calc mV/pow(2,n)
8	15	320	9.765uV
4	14	160	9.765uV
2	13	80	9.765uV
1	12	40	9.765uV

[Source: Datasheet 8.6.3.1 Shunt Voltage Register]

FSR is the Full Scale Range and is measured across the shunt resistor.

As the gain is increased so the number of ADC bits increases keeping the LSB value the same. Increasing the number of bits by oversampling means the time to gather the result increases.

Size and Speed comparisons

Since there is no function to set the acquisition time to 128 Samples in the Adafruit_INA219 library (it uses the default 12bit, 532us resolution time) all tests are done with the default resolution/sampling setting.

Note: If you edit the Adafruit library code then you can setup 128 sampling mode but you need to create a new member function following the existing ones e.g. follow the format of setCalibration_32V_2A();

Library INA219_WE for Fixed point

Copy Sketch

```
#include <Wire.h>
#include <INA219_WE.h>

INA219_WE ina219(0x45);

void setup(void) {
   Wire.begin();
   Serial.begin(115200);

   ina219.init();
   ina219.setADCMode(BIT_MODE_12);
}

// Print fixed point with dp decimal places to left.
void printFixedPointdp(long v, int dp) {

   long dpdiv = 1;
```

X

```
   if (left==0) Serial.print('0');  else Serial.print(left);
```

[Code: ina219-test-fixed.ino]

Fixed Point Results

```
--------
 I 15.50mA
VS 1.55mV
BV 4.748V
 P 73.59400mW
 Time 2596us
```

Fixed Point Results : Flash: 4840 Bytes, SRAM 479 Bytes, Time: 2596us

Library INA219_WE

Copy Sketch

```
#include <Wire.h>
#include <INA219_WE.h>

INA219_WE ina219(0x45);

void setup(void) {
   Wire.begin();
   Serial.begin(115200);

   ina219.init();
   ina219.setADCMode(BIT_MODE_12);
}

void loop(void) {
static uint32_t time_start,time_end;

   Serial.println("WE--------");
   time_start = micros();
   float fi  = ina219.getCurrent_mA();
   float fv  = ina219.getShuntVoltage_mV();
   float fbv = ina219.getBusVoltage_V();
   float fp  = ina219.getBusPower();
   time_end=micros();
```

[Code: ina219-test-we.ino]

INA219_WE Results

```
WE--------
I  15.1000mA
VS 1.5100mV
VB 4.7120V
P  76.0000mW
 Time 2256us
```

Library INA219_WE : Flash: 6108 Bytes, SRAM 489 Bytes, Time: 2256us.

Adafruit_INA219 Library

Note the Adafruit _INA219 Library requires you to also install the Adafruit_BusIO library on which it depends.

Even though it probably makes the code size larger for this one example using a sub library for I2C may pay off in the end, as most individual libraries re-write the common I2CC read and write functions as duplicates. This sub library, Adafruit_BusIO, will allow memory savings as more devices use it (as long as they do use this common library).

X

```
#include <Wire.h>
#include <Adafruit_INA219.h>

Adafruit_INA219 ina219(0x45);

void setup(void) {
   Wire.begin();
   Serial.begin(115200);

   ina219.begin();
   ina219.setCalibration_32V_2A();
}

void loop(void) {
static uint32_t time_start,time_end;

   Serial.println("Adafruit--------");
   time_start = micros();
   float fi  = ina219.getCurrent_mA();
   float fv  = ina219.getShuntVoltage_mV();
   float fbv = ina219.getBusVoltage_V();
   float fp  = ina219.getPower_mW();
```

[Code: ina219-test-adafruit.ino]

Adafruit_INA219 Results

```
Adafruit--------
I  15.9000mA
VS 1.5900mV
VB 4.7040V
P  76.0000mW
 Time 3180us
```

Adafruit_INA219 Results : Flash: 8304 Bytes, SRAM 496 Bytes.

Results Summary

These are for an Arduino Nano/Uno.

Fixed Point Results : Flash: 4840 Bytes, SRAM 479 Bytes, Time: 2596us

Library INA219_WE : Flash: 6108 Bytes, SRAM 489 Bytes, Time: 2256us.

Adafruit_INA219 Results : Flash: 8304 Bytes, SRAM 496 Bytes, Time: 3180us.

You can see that:

1. The amount of SRAM used does not change much.
2. The time taken is not vastly different but is fastest for library INA219_WE.
3. You can save 3464 Bytes with Fixed point c.f. library Adafruit_INA219.
4. You can save 1268 Bytes with Fixed point c.f. library INA219_WE.
5. The difference between INA219_WE and Adafruit_INA219 is 2196 Bytes.

TIP: Use the fixed point method if you want to save Flash memory.

The Flash size difference between using Adafruit_INA219 and INA219_WE is due to the library code size [5] (since in both cases floating point operations were used to calculate the results).

Hardware

INA219 I2C Address's

The lower two bits of the address consist of the two digital inputs A1, A0 but they are multiplexed in the same way as the ADS1115. This means inputs can be set to GND, VCC, SCL or SDA and gives an address range of 16 values (not 4 as you would normally expect using just A0 and A1). The the last bit(LSB ' 'L) is ignored as it is the read write bit (R/Wn). Therefore the addresses available are:

0x40, 0x41, 0x42, 0x43,.... 0x4C, 0x4D, 0x4E, 0x4F.

See Datasheet: 8.5.5.1 Serial Bus Address.

Breakout Board Addresses

On the breakout boards A0 and A1 are pulled low by the resistors and the two solder blobs allow either to be connected to V_{CC}. So the I2C address of the board, with no blobs, will be 0x40. For this board you can set the following addresses:

Solder blob A1	**Solder blob A0**	**INA219 Address**
not	not	0x40
not	soldered	0x41
soldered	not	0x44
soldered	soldered	0x45

Connections

For testing use an Arduino Uno/Nano and connect it as follows:

Arduino	**INA219**
5V	VDD
GND	GND
A5	SCL
A4	SDA
To your circuit	Vin-
To your circuit v+ supply	Vin+

You could connect Vin- to a simple circuit e.g. an LED with a 320R resistor in series.

Software

Arduino IDE Version used: 1.8.8

Arduino Libraries

Implicit assumption

The assumption built into the libraries is that a 0.1R resistor is used as the shunt resistor. If you want a different value you will need to re-write the library code.

This is not a bad assumption as it is the resistor most commonly used but you can get lower ones if you want less voltage drop e.g. 10mOhms etc.

Adafruit_INA219

This library has useful comments that go through the design process in setting up the INA219.

Install the library named **Adafruit_INA219**. Version used 1.0.9.
Also install the library named **Adafruit_BUSIO**. Version used 1.3.2.

Use the library manager, searching for "**ina219"**, "**BUSIO**".

INA219_WE

Install the library named **INA219_WE,** Version used 1.1.1.

Use the library manager, searching for "**ina219"**.

(On github https://github.com/wollewald/INA219_WE)

INA219_WE Library warning

IDE output error message

The Arduino IDE complains that Sensor is the wrong category - you can ignore this or change the text "Sensor" in the file library.properties (within the INA219_WE directory) to "Sensors" to fix it.

INA219_WE Library Functions

Available INA219_WE functions are:

```
INA219_WE(int addr);
INA219_WE();                    //sets default I2C Address 0x40
void init();
void reset_INA219();
void setCorrectionFactor(float corr);
void setADCMode(INA219_ADC_MODE mode);
void setMeasureMode(INA219_MEASURE_MODE mode);
void setPGain(INA219_PGAIN gain);
void setBusRange(INA219_BUS_RANGE range);
float getShuntVoltage_mV();
float getBusVoltage_V();
float getCurrent_mA();
float getBusPower();
bool getOverflow();
```

X

voltage and power quite easily.

Adafruit_INA219 Library functions

Available

```
Adafruit_INA219(uint8_t addr = INA219_ADDRESS);
bool begin(TwoWire *theWire = &Wire);
void setCalibration_32V_2A();
void setCalibration_32V_1A();
void setCalibration_16V_400mA();
float getBusVoltage_V();
float getShuntVoltage_mV();
float getCurrent_mA();
float getPower_mW();
void powerSave(bool on);
```

The Adafruit_INA219 library takes a different approach - pre-supplied calibrations.

It has three convenient calibrations already worked out for you**:**

1. setCalibration_32V_2A();
2. setCalibration_32V_1A();
3. setCalibration_16V_400mA();

If you don't want to mess around figuring out stuff then these functions provide quick and easy setup:

For 1. above, the maximum expected current is 2A,
and a range of 32V (actually 26V maximum input).

For 2. above, the maximum expected current is 1A,
and a range of 32V (actually 26V maximum input).

For 3. above, the maximum expected current is 400mA,
and a range of 16V (this in actually is 16V maximum input although it withstands 26V).

The expected current is just that - it is the maximum current you expect to see and only changes the resolution of the resultant reading.

For quick use you can probably see a range setting you want to use.

INA219 Conclusions

The INA219 is a very easy chip to use and it is really plug in and go - as long as your measurement voltage is within 26V. You can use the standard libraries to easily obtain readings for current, bus voltage and power.

Do all the libraries tested here work - giving the same results? : Yes.

You can also save over 3k Bytes of flash if you don't use floating point operations.and instead, use fixed point here.

Of the two libraries studied here Adafruit_INA219 offers the easiest use, providing you with pre-calculated calibration options. However, it does not give you control so you can't set the number of samples etc. INA219_WE does offer this control.

TIP: Use a low shunt resistance to measure higher current.

If you change the shunt resistor to 0.02 Ohms you can measure 10~15A. (See the datasheet for an example). You have to re-calculate the calibration register settings for this resistance.

Show Index

Comments

Have your say about what you just read! Leave me a comment in the box below.

Don’t see the comments box? Log in to your Facebook account, give Facebook consent, then return to this page and refresh it.

X