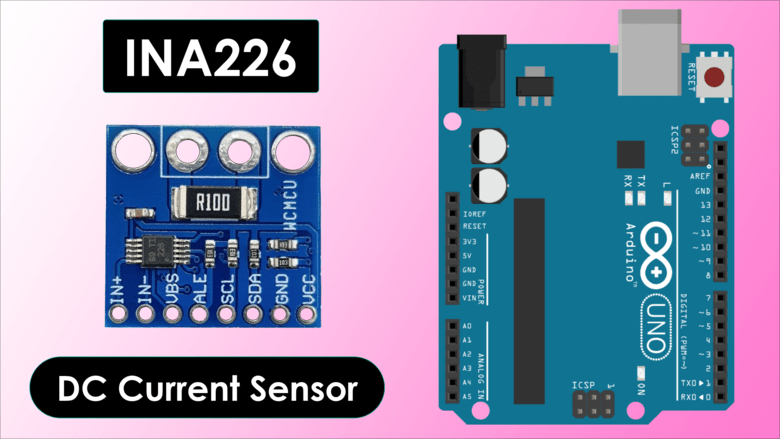
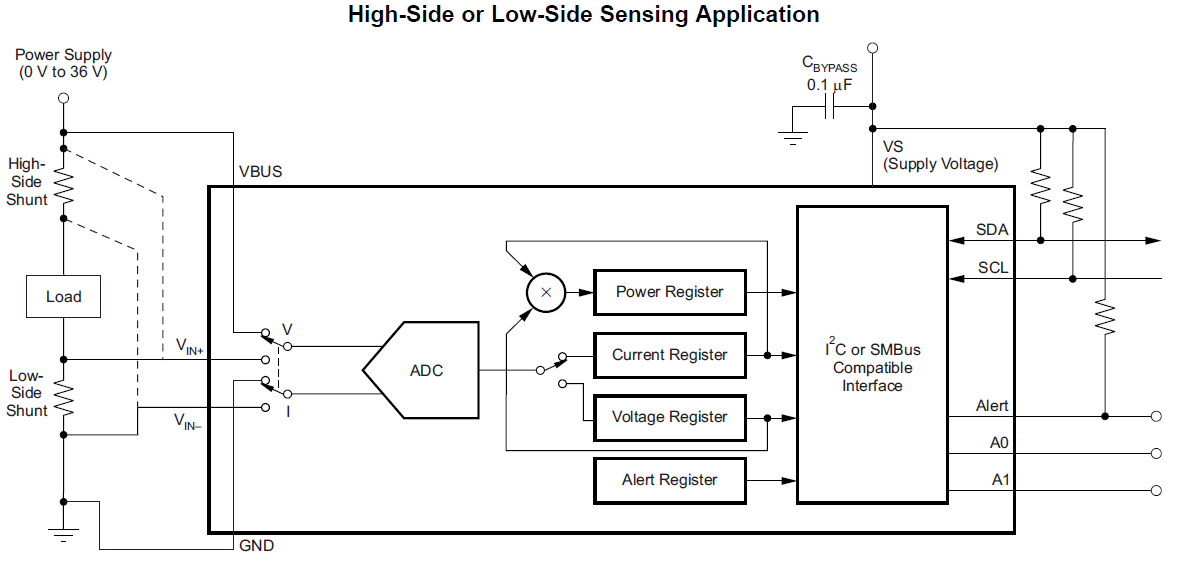
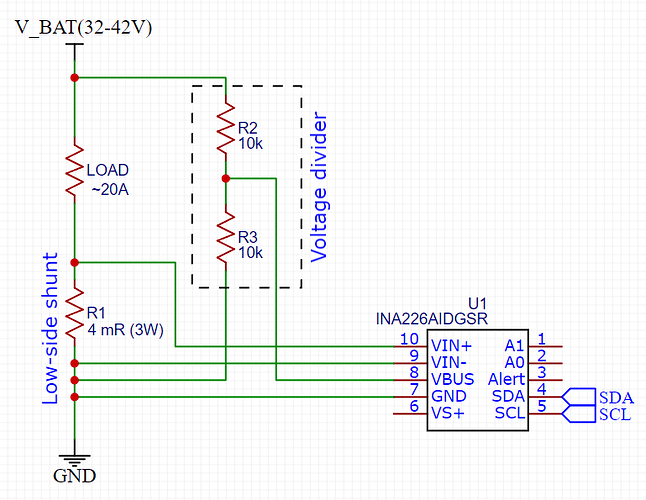
**How to use INA226 DC Current Sensor with Arduino**



**SHARE**







**Overview**

In this tutorial, we will learn how to interface the **INA226 DC Current Sensor** Module with an **Arduino**. The INA226, like its predecessor the [**INA219**](https://how2electronics.com/how-to-use-ina219-dc-current-sensor-module-with-arduino/), is a device that can measure **current**, **voltage**, and **power**. However, it has an extended range, capable of measuring voltages up to **36V** and current up to **20A**. Thus you can make [**DC Energy Meter**](https://how2electronics.com/iot-dc-energy-meter-using-esp8266-blynk-36v-30a/) project using this sensor or a [**12V Battery Monitoring System**](https://how2electronics.com/iot-based-12v-battery-monitoring-system-with-esp8266/) project and also the **[Solar Panel Monitoring](https://how2electronics.com/iot-solar-panel-monitoring-system-with-esp8266-mqtt/" \t "_blank)** System project.

The INA226 is a shunt and **bus voltage monitor** introduced by **Texas Instruments**. It offers an integrated, zero-drift, bi-directional interface that monitors shunt voltage, bus voltage, current, and power. The INA226 communicates with microcontrollers through an **I2C interface**, allowing for easy data transfer and interpretation.

One of the unique features of the INA226 is its **configurable averaging** and **conversion times** for both the **shunt voltage** and **bus voltage measurements**. This provides a means to optimize the device for various applications and system requirements. Furthermore, it also includes a **programmable threshold** and **alert functionality** for these measurements which can be used for system-level optimizations such as **power saving** and event-driven interrupt programming.

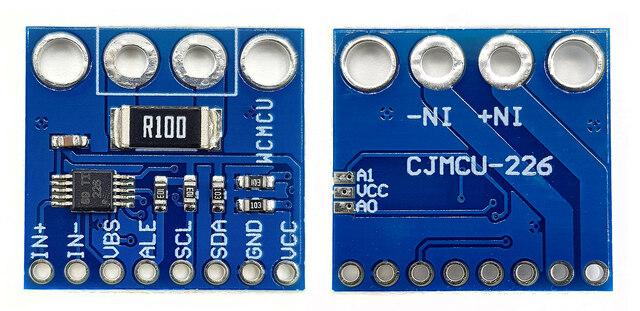
The **INA226 Current Sensor** Module is versatile, finding its utility in server **power management**, telecommunications, power supply equipment, and **battery management systems**. Essentially, any system requiring high-accuracy, cost-effective power and current monitoring can significantly benefit from this module.

**Bill of Materials**

To interface INA226 DC Current Sensor Module with Arduino, we need following components. You can purchase all of them from the given links:

| **S.N.** | **Components** | **Quantity** | **Purchase Link** |
| --- | --- | --- | --- |
| 1 | Arduino Nano | 1 | **[Amazon](https://amzn.to/3Y4YsX2" \t "_blank)** | **[AliExpress](https://s.click.aliexpress.com/e/_DFdXwN9" \t "_blank)** | **[SunFounder](https://www.sunfounder.com/products/arduino-nano-board?ref=how2electronics" \t "_blank)** |
| 2 | INA226 Current Sensor | 1 | **[Amazon](https://amzn.to/3rfzr08" \t "_blank)** | **[AliExpress](https://s.click.aliexpress.com/e/_DeMvXu9" \t "_blank)** |
| 3 | Battery 3.7V | 1 | **[Amazon](https://amzn.to/3I5GpLG" \t "_blank)** | **[AliExpress](https://s.click.aliexpress.com/e/_DDpe3dd" \t "_blank)** |
| 4 | DC Motor | 1 | **[Amazon](https://amzn.to/3FPnO3o" \t "_blank)** | **[AliExpress](https://s.click.aliexpress.com/e/_Dk7taTN" \t "_blank)** | **[SunFounder](https://www.sunfounder.com/products/5pcs-1-5v-6v-type-miniature-dc-motors?ref=how2electronics" \t "_blank)** |
| 5 | Jumper Wires | 10 | [**Amazon**](https://amzn.to/3F8fLhW) | **[AliExpress](https://s.click.aliexpress.com/e/_DnqCpEJ" \t "_blank)** | **[SunFounder](https://www.sunfounder.com/products/breadboard-jumper-wires?ref=how2electronics" \t "_blank)** |
| 6 | Breadboard | 1 | **[Amazon](https://amzn.to/3Bg68wE" \t "_blank)** | **[AliExpress](https://s.click.aliexpress.com/e/_DnDCZk7" \t "_blank)** | **[SunFounder](https://www.sunfounder.com/products/rab-holder-kit?ref=how2electronics" \t "_blank)** |

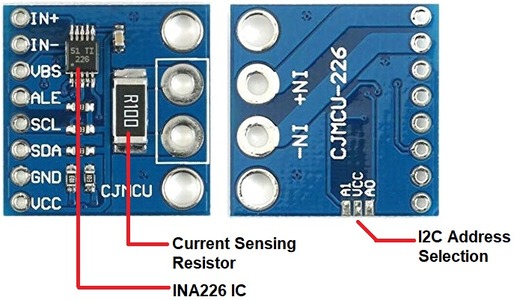
**INA226 DC Current Sensor Module**

[](https://how2electronics.com/wp-content/uploads/2023/07/INA226-.jpg)

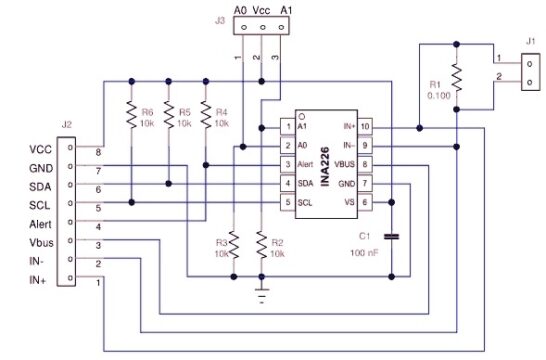
The INA226 is an advanced, high-side current shunt and power monitor IC developed by Texas Instruments. It operates over an I2C-compatible or SMBus-compatible interface, offering direct digital communication with a microcontroller. It provides more extensive capabilities than the INA219, including better accuracy, more extensive voltage and current range, as well as integrated math functions that directly report power in watts.

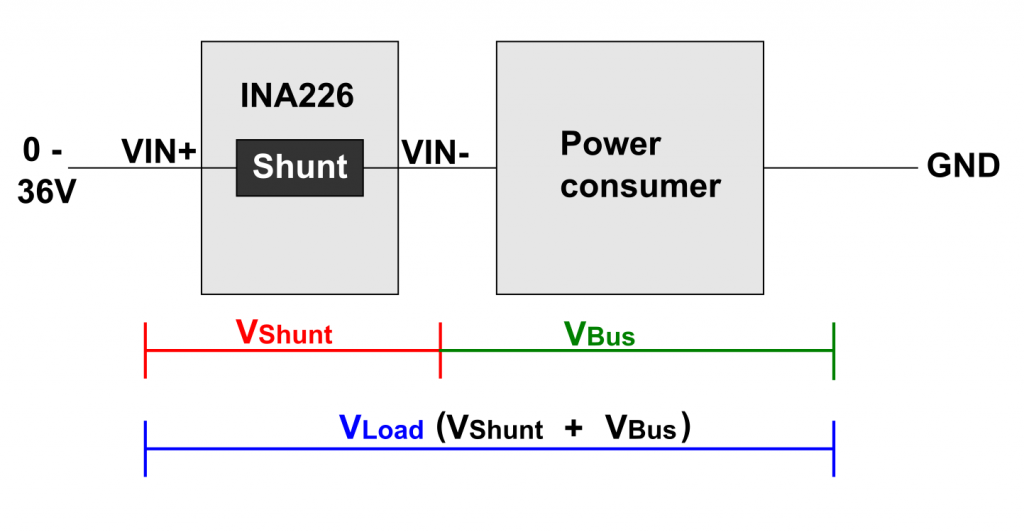
Download INA226 Datasheet: [**INA226 Datasheet**](https://www.ti.com/lit/ds/symlink/ina226.pdf)

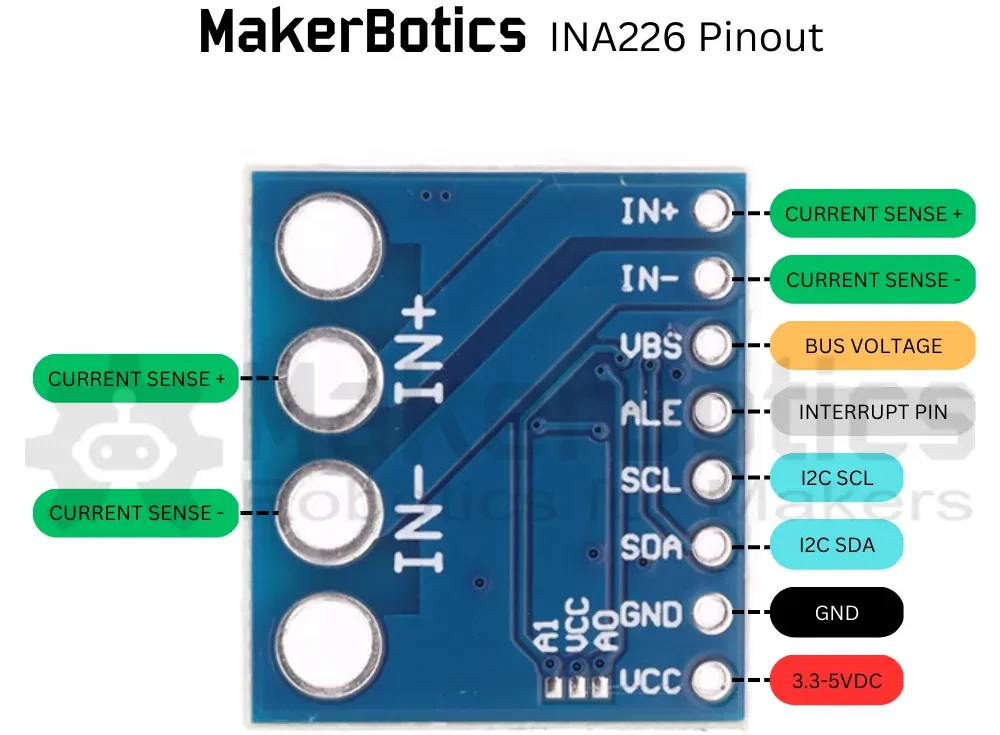
**INA226 Board Circuit & Schematic**

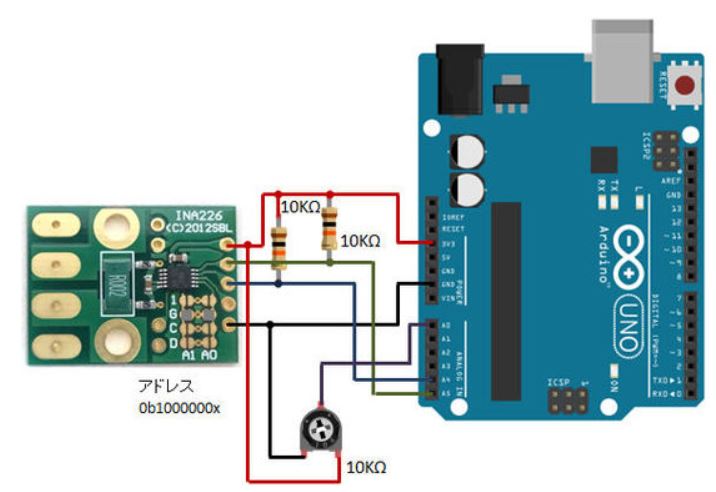
[](https://how2electronics.com/wp-content/uploads/2023/07/INA226-board.jpg)

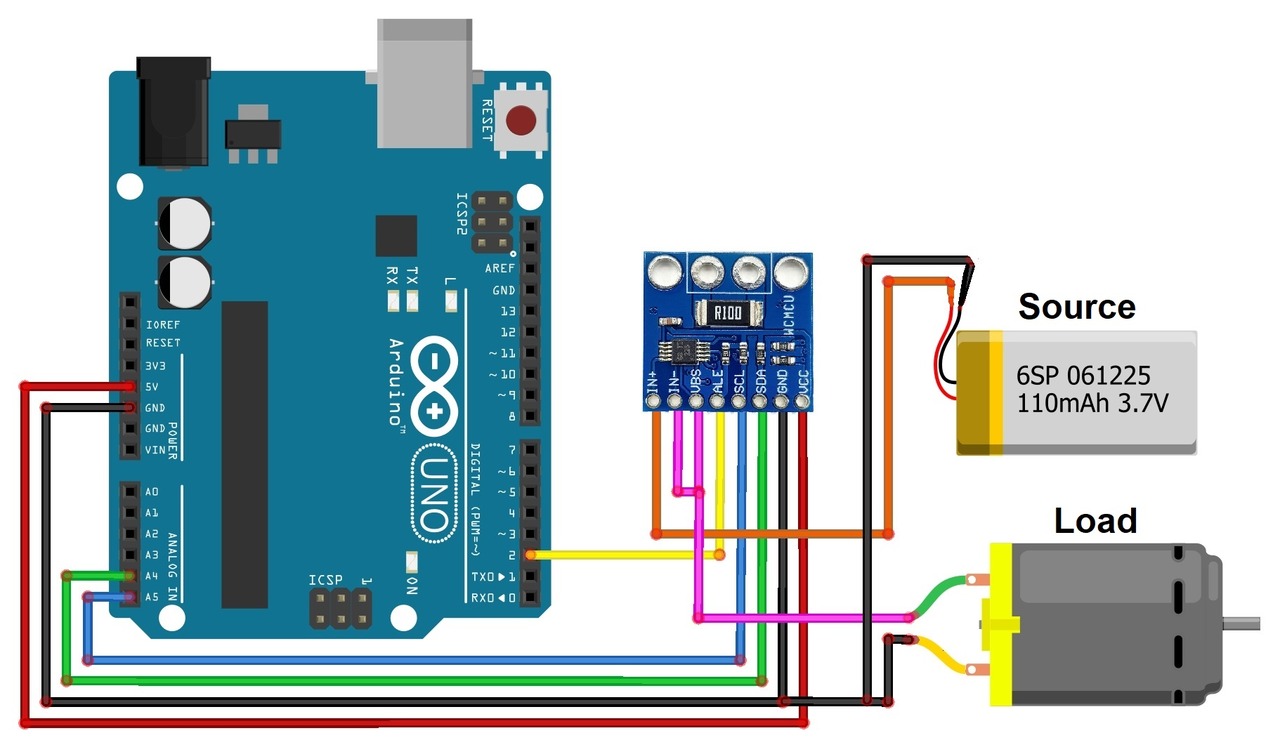
The INA226 module is built with an INA226 chip, a few resistors, and a capacitor that helps reduce noise, or unwanted electrical signals.

[](https://how2electronics.com/wp-content/uploads/2023/07/INA226-board-sch.jpg)







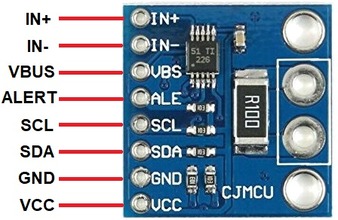


**Features & Specifications of INA226**

1. **Operational Voltage (2.7 – 5.5 Volts)**: The INA226 operates between 2.7 to 5.5 volts. This widens its compatibility with systems that run at different voltage levels. This trait is beneficial in accommodating an extensive range of circuit designs.
2. **Bus Voltage Range (0 – 36 Volts)**: The INA226 can monitor power supplies up to 36 volts, expanding the device’s applicability to various power supplies.
3. **Current Sensing Range (± 500mA to ± 50A)**: Depending on the value of the shunt resistor, INA226 can monitor a wide range of currents, thus catering to a myriad of applications including power management, battery chargers, and DC motor control.
4. **Power consumption**: Continuous mode: 0.35 mA; Power-Down Mode: 2.3.µA
5. **Measurement modes:**: continuous or on-demand (“triggered”)
6. Averaging of 1, 4, 64, 128, 256, 512 or 1024 individual measurements
7. A/D conversion time adjustable in eight levels: 0.14 to 8.2 ms
8. **Higher Precision**: The INA226 provides increased precision through its 16-bit ADC, resulting in more accurate measurements.
9. Programmable alarm pin for limit violations and available data

**Pinout of INA226**

The INA226 sensor module typically has 8 pins, which are as follows:

[](https://how2electronics.com/wp-content/uploads/2023/07/INA226-board-1.jpg)

1. **VCC**:  It accepts an input voltage from 2.7V to 5.5V.
2. **GND**: Ground pin, connected to the ground of the power supply.
3. **SDA**: Serial Data line for the I2C interface. It’s used for the bidirectional transfer of data.
4. **SCL**: Serial Clock line for the I2C interface. It’s used for synchronization during data transfer.
5. **ALE**: This is the Alert pin. It’s an open-drain output that requires a pull-up resistor. This pin can be used for various alerts or as a conversion-ready signal.
6. **VBUS**: This pin is used to measure the supply voltage. It can measure the supply voltage up to 36V.
7. **IN-**: This pin connects to the Load. This is where the shunt resistor is placed for current sensing.
8. **IN+**: This pin connects to the Power Source.

**Current Sensing Resistor Selection**

The **maximum current** that the INA226 can measure is not determined directly by the chip itself, but rather by the value of the **shunt resistor** used in conjunction with it.

The INA226 measures the **voltage drop** across a **shunt resistor**, and from that calculates the current using **Ohm’s law (I = V/R)**. Since the INA226 can measure a maximum shunt voltage of **81.92mV** (with a resolution of 2.5µV), the maximum current is dependent on the **resistance** of the shunt.

Below is a table of suggested shunts.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Shunt Resistance** | **Max Current (A)** | **Part Size** | **Part Number** | **Power Rating (W)** | **Max Power (W)** |
| 0.003 | 27.31 | 2512 | ERJMS4SF3M0U | 3 | 2.24 |
| 0.004 | 20.48 | 2512 | ERJMS4SF4M0U | 3 | 1.68 |
| 0.005 | 16.38 | 2512 | ERJMS4HF5M0U | 3 | 1.34 |
| 0.006 | 13.65 | 2512 | ERJMS4HF6M0U | 3 | 1.12 |
| 0.007 | 11.70 | 2512 | ERJMS4HF7M0U | 2 | 0.96 |
| 0.008 | 10.24 | 2512 | ERJMS4HF8M0U | 2 | 0.84 |
| 0.009 | 9.10 | 2512 | ERJMS4HF9M0U | 2 | 0.75 |
| 0.010 | 8.19 | 2512 | ERJMS4HF10MU | 2 | 0.67 |

**ADC Resolution**

The bus voltage register of the INA226, a **16-bit signed** entity, features a **least significant bit (LSB)**, or resolution, of **1.25mV**. This permits a full-scale voltage measurement range of **+/-40.96V**.

The INA226’s analog input designed for the **shunt resistor** is capable of accepting a maximum (full scale) voltage of **81.92mV**. When paired with a shunt resistor of **4mOhm**, it enables the measurement of a current up to **+/-20.48A** with a resolution as precise as **625µA**.

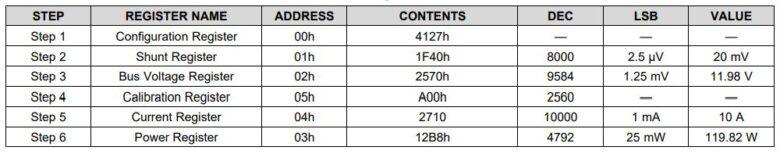
**I2C Address**

Similar to INA219, the INA226 uses the I2C protocol to communicate with microcontrollers. The default I2C address for the INA226 is 0x40. This address can be changed to facilitate the use of multiple INA226 sensors on the same I2C bus. The INA226 allows up to 16 different I2C addresses, from 0x40 to 0x4F. You can change the I2C address by modifying the connections of the A0 and A1 pins.

[](https://how2electronics.com/wp-content/uploads/2023/07/I2C-Address-Selection.jpg)

**Register Address & Values Calculations**

The INA226 also has internal registers which are accessed via the I2C interface. Some of the essential ones are:

[](https://how2electronics.com/wp-content/uploads/2023/07/Register-Address.jpg)

1. **Configuration Register (Address = 00h)**: Similar to [**INA219**](https://how2electronics.com/how-to-use-ina219-dc-current-sensor-module-with-arduino/), this register is used to control various aspects of the INA226.
2. **Shunt Voltage Register (Address = 01h)**: This register holds the raw measurement for the shunt voltage.

*Shunt Voltage (V) = Shunt Voltage Register Value \* 2.5 µV*

1. **Bus Voltage Register (Address = 02h)**: This register holds the raw measurement for the bus voltage.

*Bus Voltage (V) = Bus Voltage Register Value \* 1.25 mV*

1. **Power Register (Address = 03h)**: This register holds the calculated power value.

*Power (W) = Power Register Value \* Power\_LSB*

1. **Current Register (Address = 04h)**: This register holds the calculated current value.

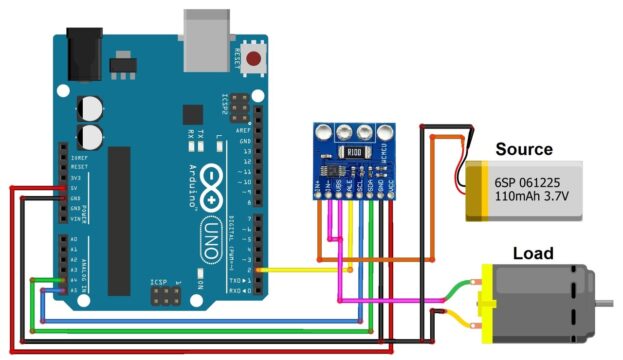
*Current (A) = Current Register Value \* Current\_LSB*

1. **Calibration Register (Address = 05h)**: This register is used to set the calibration value for the current and power calculations.

**How to use INA226 DC Current Sensor with Arduino**

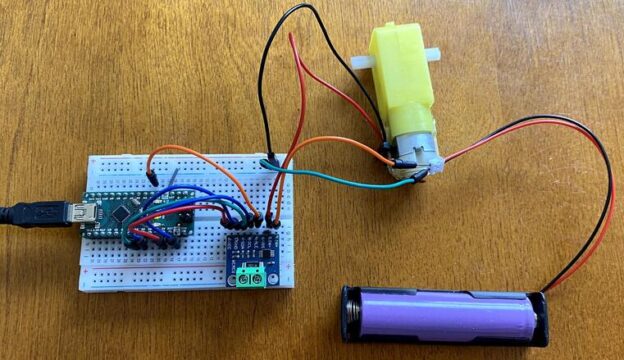
Now let us interface the INA226 DC Current Sensor Module with Arduino. The hardware interfacing is very simple.

Here is a simple connection diagram that can be used to interface INA226 Current sensor with Arduino along with external load and power source.

[](https://how2electronics.com/wp-content/uploads/2023/07/INA226-DC-Current-Sensor-with-Arduino.jpg)

Connect the VCC, GND, SCL & SDA pin of INA219 to 5V, GND, A5 & A4 of Arduino respectively. The Vin+ pin should be connected to power source and the Vin- to the load as shown in the image above. The INA226 Sensor has VBus Pin, which is used to measure the source Voltage. Connect the VBus pin to Vin- pin.

For test, I used 3.7V and 9V Battery as a Power Source. As a load, I used a DC Motor for this circuit.

[](https://how2electronics.com/wp-content/uploads/2023/07/INA226-Current-Sensor-Arduino-Connection.jpg)

The Vin+ pin can accept input voltage upto 36V. The device is rated to 36V (40V absolute maximum) on the input pins. Therefore, do not supply anything above 36V.

You may need to desolder the shunt resistor and replace it with above mentioned resistor in case, you want to measure higher current.

**INA226 Arduino Library**

There are many versions of INA226 library available for use. I tested most of them. I found the library from **[Wollewald](https://github.com/wollewald" \t "_blank)** as accurate one.

You can download the library from GitHub or install it directly with the Library Manager of the Arduino IDE.

**Download**: [INA226 Library](https://github.com/wollewald/INA226_WE)

There is a total of seven example sketches used to present the functions of the library. I have used the example of continuous mode. Many of the functions are used in all sketches and therefore only need to be explained once.

**Source Code/Program**

Copy the following code and paste it on your Arduino IDE editor window. Before you upload the code, you may need to make some modifications.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98 | #include <Wire.h>  #include <INA226\_WE.h>  #define I2C\_ADDRESS 0x40    INA226\_WE ina226 = INA226\_WE(I2C\_ADDRESS);    void setup()  {    Serial.begin(9600);    while (!Serial); // wait until serial comes up on Arduino Leonardo or MKR WiFi 1010    Wire.begin();    ina226.init();      /\* Set Number of measurements for shunt and bus voltage which shall be averaged      Mode \*     \* Number of samples      AVERAGE\_1            1 (default)      AVERAGE\_4            4      AVERAGE\_16          16      AVERAGE\_64          64      AVERAGE\_128        128      AVERAGE\_256        256      AVERAGE\_512        512      AVERAGE\_1024      1024\*/      //ina226.setAverage(AVERAGE\_16); // choose mode and uncomment for change of default      /\* Set conversion time in microseconds       One set of shunt and bus voltage conversion will take:       number of samples to be averaged x conversion time x 2           Mode \*         \* conversion time       CONV\_TIME\_140          140 µs       CONV\_TIME\_204          204 µs       CONV\_TIME\_332          332 µs       CONV\_TIME\_588          588 µs       CONV\_TIME\_1100         1.1 ms (default)       CONV\_TIME\_2116       2.116 ms       CONV\_TIME\_4156       4.156 ms       CONV\_TIME\_8244       8.244 ms  \*/      //ina226.setConversionTime(CONV\_TIME\_1100); //choose conversion time and uncomment for change of default      /\* Set measure mode      POWER\_DOWN - INA226 switched off      TRIGGERED  - measurement on demand      CONTINUOUS  - continuous measurements (default)\*/      //ina226.setMeasureMode(CONTINUOUS); // choose mode and uncomment for change of default      /\* Set Resistor and Current Range       if resistor is 5.0 mOhm, current range is up to 10.0 A       default is 100 mOhm and about 1.3 A\*/      ina226.setResistorRange(0.1, 1.3); // choose resistor 0.1 Ohm and gain range up to 1.3A      /\* If the current values delivered by the INA226 differ by a constant factor       from values obtained with calibrated equipment you can define a correction factor.       Correction factor = current delivered from calibrated equipment / current delivered by INA226\*/      ina226.setCorrectionFactor(0.93);      Serial.println("INA226 Current Sensor Example Sketch - Continuous");      ina226.waitUntilConversionCompleted(); //if you comment this line the first data might be zero  }    void loop()  {    float shuntVoltage\_mV = 0.0;    float loadVoltage\_V = 0.0;    float busVoltage\_V = 0.0;    float current\_mA = 0.0;    float power\_mW = 0.0;      ina226.readAndClearFlags();    shuntVoltage\_mV = ina226.getShuntVoltage\_mV();    busVoltage\_V = ina226.getBusVoltage\_V();    current\_mA = ina226.getCurrent\_mA();    power\_mW = ina226.getBusPower();    loadVoltage\_V  = busVoltage\_V + (shuntVoltage\_mV / 1000);      Serial.print("Shunt Voltage [mV]: "); Serial.println(shuntVoltage\_mV);    Serial.print("Bus Voltage [V]: "); Serial.println(busVoltage\_V);    Serial.print("Load Voltage [V]: "); Serial.println(loadVoltage\_V);    Serial.print("Current[mA]: "); Serial.println(current\_mA);    Serial.print("Bus Power [mW]: "); Serial.println(power\_mW);    if (!ina226.overflow)    {      Serial.println("Values OK - no overflow");    }    else    {      Serial.println("Overflow! Choose higher current range");    }    Serial.println();      delay(3000);  } |

**Code Modifications & Setup**

From this code, you need make some modifications in order to get the result accurately.

|  |  |
| --- | --- |
| 1 | ina226.setResistorRange(0.1, 1.3); |

This line set Resistor and Current Range. If resistor is 5.0 mOhm, current range is up to 10A. By deefault is 100 mOhm and about 1.3A.

|  |  |
| --- | --- |
| 1 | ina226.setCorrectionFactor(0.93); |

If the current values delivered by the INA226 differ by a constant factor from values obtained with calibrated equipment you can define a correction factor.

*Correction factor = current delivered from calibrated equipment / current delivered by INA226*

You may connect a multimeter in Series with Vin- and measure the current and verify with the value in Serial Monitor.

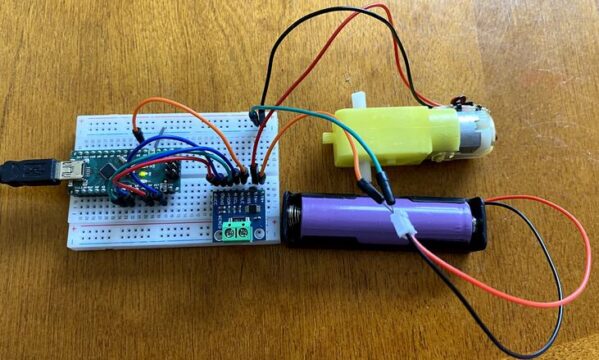
Some of the other fucntionalities and setup required are as follows:

1. Set the number of individual measurements for the shunt and bus voltage conversion with setAverage()
   * 1, 4, 16, 64, 128, 256, 512, or 1024 individual measurements are averaged
2. Set the A/D conversion time for the shunt and bus voltage with setConvTime()
   * 8 levels adjustable between 140 µs and 8.244 ms
   * Note: obtaining a data set of shunt and bus voltage takes twice the time
3. Set the measurement mode with setMeasureMode()
   * CONTINUOUS – continuous measurement
   * TRIGGERED – “on request”: I explain in the next example.
   * POWER\_DOWN – turns off the INA226. But better use the more comfortable powerDown() function, which is explained below.
   * The INA226 actually allows determining shunt or bus voltages – but I did not implement that. Using my library, the measurements are only available in a double pack.
4. Set the current range with setCurrentRange()
   * You can set 400 or 800 mA as the maximum current. The smaller the current range, the higher the resolution for the current and the power.

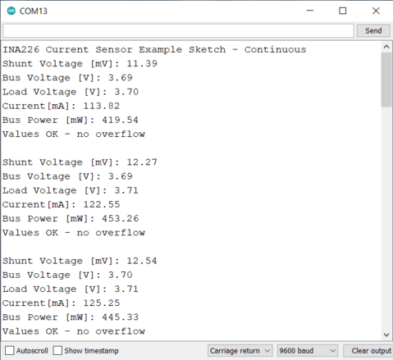
**Test Results**

After you upload the code to the Arduino board, you can start testing the setup.

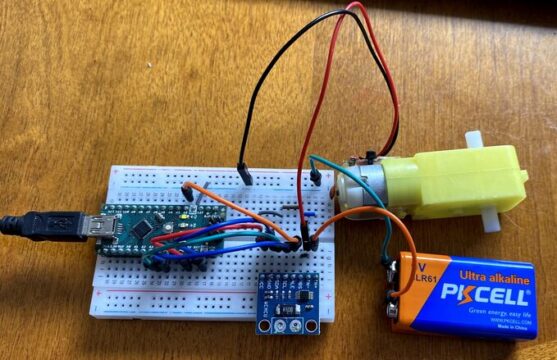
First connect 3.7V Lithium-ion Battery to the Vin+ Pin and a DC motor to Vin- Pin. The DC motor will start rotating.

[](https://how2electronics.com/wp-content/uploads/2023/07/INA226-Current-Sensor-Load-Test.jpg)

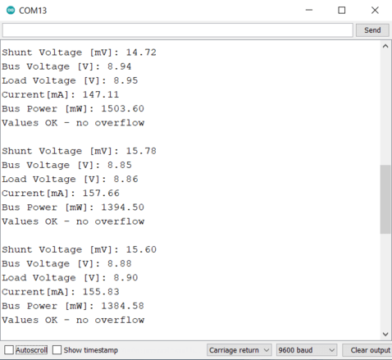
Now open the Serial Monitor and you will see the following results for 3.7V Source.

[](https://how2electronics.com/wp-content/uploads/2023/07/3.7V-Bat.png)

Remove the 3.7V Battery and connect the 9V Battery.

[](https://how2electronics.com/wp-content/uploads/2023/07/9V-DC-Voltage-Test.jpg)

The same circuit when tested with 9V DC source gives following results.

[](https://how2electronics.com/wp-content/uploads/2023/07/9V-Bat.png)

**Conclusion**

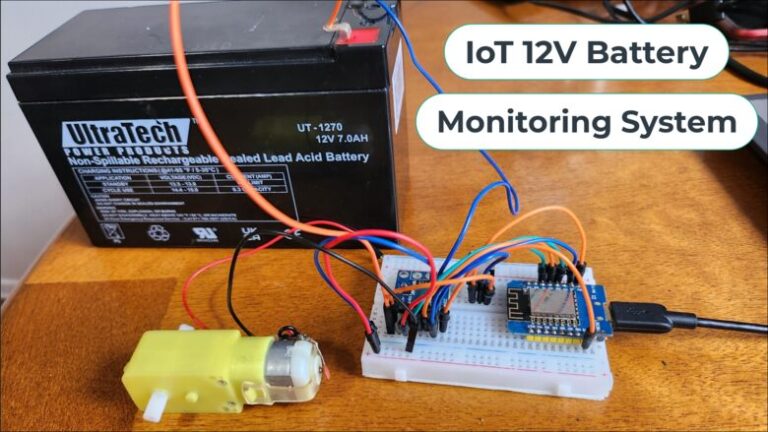
In conclusion, the INA226 DC Current Sensor Module is a highly versatile and capable device, perfect for a wide variety of applications where precise power, current, and voltage monitoring is needed. It has an extended range compared to its predecessor, the INA219, enabling the measurement of up to 36V in voltage and 20A in current.

Interfacing this INA226 Current Sensor module with an Arduino allows us to leverage its advanced capabilities in a straightforward manner, making it an invaluable tool for those seeking high-accuracy and cost-effective monitoring solutions. Thus, by mastering the interfacing of the INA226 with an Arduino, we can significantly enhance our capacity to monitor and manage power within electronic systems.

**IoT Based 12V Battery Monitoring System with ESP8266**

AdminBy [Admin](https://how2electronics.com/author/alex-newton/" \o "Posts by Admin)Updated:August 10, 2023[3 Comments](https://how2electronics.com/iot-based-12v-battery-monitoring-system-with-esp8266/#comments)4 Mins Read

**Share**



**SHARE**

[](https://www.hqonline.com/)

**Overview:**

In this project, we will build an **IoT-based 12V Battery Monitoring System** using **ESP8266** and **INA226** DC Current Sensor. This system is specifically designed for monitoring **lead-acid batteries**, which are widely used in automotive, solar, and other high-capacity applications. The primary goal of this system is to ensure the **optimal performance** and **longevity** of the battery by preventing **overcharging** or excessive **discharging**, which can lead to battery damage or system failure.

In this setup, we are using the [**INA226 sensor**](https://how2electronics.com/how-to-use-ina226-dc-current-sensor-with-arduino/), a high-precision current and power monitor, to provide accurate readings of the **battery’s voltage**, load voltage, **current**, and **power**. These readings are crucial for maintaining the **health** and **efficiency** of the battery. The ESP8266, a low-cost Wi-Fi microchip with full TCP/IP stack and **microcontroller** capability, is used to send this data to the **ThingSpeak server**.

The ThingSpeak server, an open-source **Internet of Things (IoT)** application and **API**, is used to collect and store sensor data in the cloud and develop IoT applications. This allows users to monitor the **battery status** remotely from anywhere in the world via their smartphones or computer dashboards. The server displays the battery voltage, load voltage, current, and power, providing a comprehensive overview of the **battery’s condition** in both charging and discharging states.

Check our previous similar project for 3.7V Lithium-Ion/Lithium-Polymer Batteries:

1. [**IoT Based Battery Status Monitoring System using ESP8266**](https://how2electronics.com/iot-based-battery-status-monitoring-system-using-esp8266/)
2. [**IoT Battery Monitoring System with DIY LiPo Charger**](https://how2electronics.com/iot-battery-monitoring-system-with-diy-lipo-charger/)

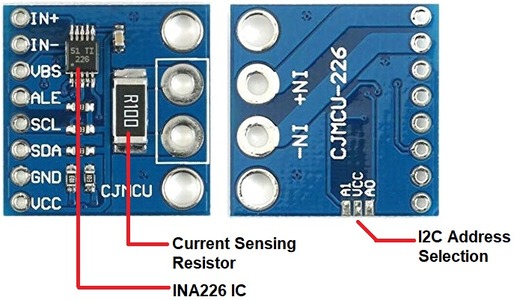
**Bill of Materials:**

The following are the components required for building the IoT-Based 12V Battery Monitoring System Project.

| **S.N.** | **Components** | **Quantity** | **Purchase Links** |
| --- | --- | --- | --- |
| 2 | NodeMCU ESP8266 or Wemos D1 Mini Board | 1 | **[Amazon](https://amzn.to/3BVv395" \t "_blank)** | **[AliExpress](https://s.click.aliexpress.com/e/_DCd1DML" \t "_blank)** | **[SunFounder](https://www.sunfounder.com/products/esp8266-nodemcu-cp2102-esp-12e-development-board-open-source-serial-module-works-great-for-arduino-ide-micropython?ref=how2electronics" \t "_blank)** |
| 3 | INA226 Current Sensor | 1 | **[Amazon](https://amzn.to/3rfzr08" \t "_blank)** | **[AliExpress](https://s.click.aliexpress.com/e/_DeMvXu9" \t "_blank)** |
| 4 | Jumper Wires | 10 | [**Amazon**](https://amzn.to/3F8fLhW) | **[AliExpress](https://s.click.aliexpress.com/e/_DnqCpEJ" \t "_blank)** | **[SunFounder](https://www.sunfounder.com/products/breadboard-jumper-wires?ref=how2electronics" \t "_blank)** |
| 5 | Breadboard | 1 | **[Amazon](https://amzn.to/3Bg68wE" \t "_blank)** | **[AliExpress](https://s.click.aliexpress.com/e/_DnDCZk7" \t "_blank)** | **[SunFounder](https://www.sunfounder.com/products/sunfounder-breadboard-kit?ref=how2electronics" \t "_blank)** |
| 6 | Micro-USB Cable | 1 | **[Amazon](https://amzn.to/3UFocqS" \t "_blank)** | **[AliExpress](https://s.click.aliexpress.com/e/_DdCccnH" \t "_blank)** |
| 7 | 12V Lead-Acid Battery | 1 | **[Amazon](https://amzn.to/3QrH123" \t "_blank)** | **[AliExpress](https://s.click.aliexpress.com/e/_DEkLboB" \t "_blank)** |

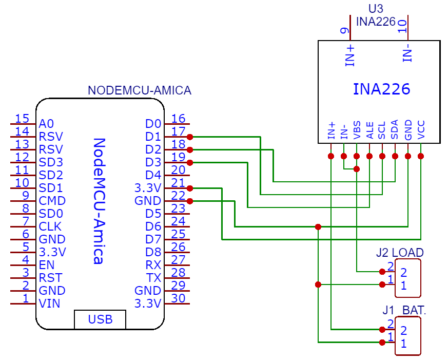
**Circuit Design & Schematic:**

Let us move to the project part. We could have used [**INA219 Current Sensor**](https://how2electronics.com/how-to-use-ina219-dc-current-sensor-module-with-arduino/) for this project, but INA226 has voltage limitations of 26V and the maximum current it can measure is ±3.2A.

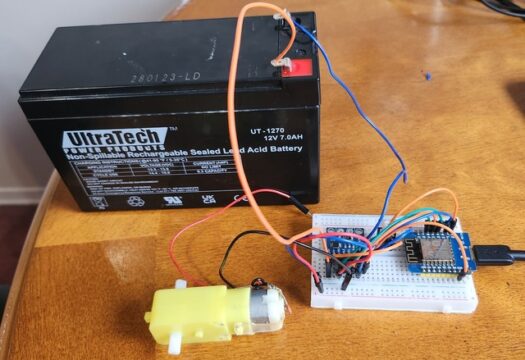
[](https://how2electronics.com/wp-content/uploads/2023/07/INA226-board.jpg)

The **INA226** can measure the voltage up to **36V** and the current up to **30A**. The current setting is based on the shunt resistance that needs to be changed based on current requirements. Follow the [**INA226 Interfacing Guide**](https://how2electronics.com/how-to-use-ina226-dc-current-sensor-with-arduino/) for learning more about current settings.

Let us take a look at the schamtic of IoT Based 12V Battery Monitoring System with ESP8266.

[](https://how2electronics.com/wp-content/uploads/2023/08/IoT-Based-12V-Battery-Monitoring-System-with-ESP8266.png)

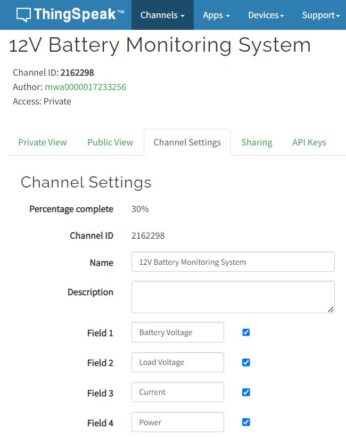
The **SDA** and **SCL** pins of INA226 are connected to the D2 & D1 pins of the ESP8266 Board. The **Vin+** pin should be connected to a power source and the **Vin-** to the load as shown in the design schematic. The INA226 Sensor has VBus Pin, which is used to measure the Load Voltage. Therefore we need to connect the **VBus** pin to Vin- pin.

[](https://how2electronics.com/wp-content/uploads/2023/08/12V-Battery-Monitoring-ESP8266.jpg)

You may use a breadboard for connection or design your own custom PCB.

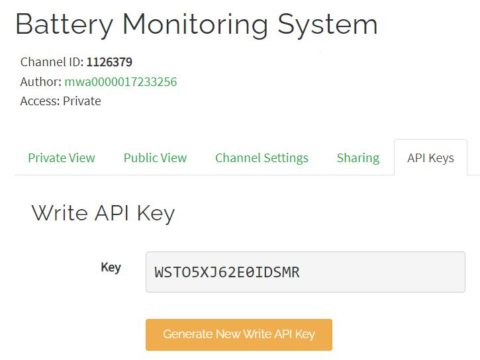
**Setting up Thingspeak**

In order to Monitor the **Battery Data** on Thingspeak Server, you first need to Setup the **Thingspeak**. To set up the Thingspeak Server, visit <https://thingspeak.com/>. Create an account or simply sign in if you created the account earlier. Then create a new channel with following details.

[](https://how2electronics.com/wp-content/uploads/2023/08/Setup.jpg)

The parameters that we are gonna measure is Battery Voltage, Load Voltage, Current and Power. Therefore, we need to create a 4 parameters.

Then go to the API section of the dashboard and copy the **API Key**. This API key is needed in the code part.

[](https://how2electronics.com/wp-content/uploads/2021/04/api-key-part-2.jpg)

Now your Thingspeak account setup is complete.

**Source Code/Program**

Let us move to the programming part of IoT Based 12V Battery Monitoring System with ESP8266. The code requires [**INA226 Library**](https://github.com/wollewald/INA226_WE) for compilation. First add the library to the Arduino IDE.

From the following lines, change the WiFi SSID, Password & Thingspeak API Key.

|  |  |
| --- | --- |
| 1  2  3 | String apiKey = "\*\*\*\*\*\*\*\*\*\*";  const char\* ssid = "\*\*\*\*\*\*\*\*\*\*";          // Enter your WiFi Network's SSID  const char\* pass = "\*\*\*\*\*\*\*\*\*\*";  // Enter your WiFi Network's Password |

Here is a complete code for this project.

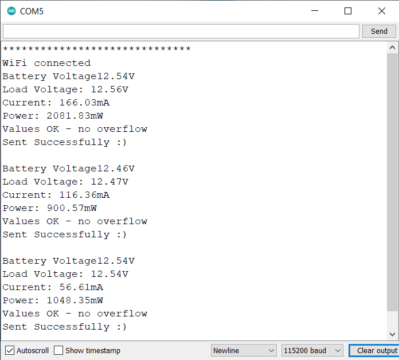
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128 | #include <Wire.h>  #include <ESP8266WiFi.h>  #include <INA226\_WE.h>  #define I2C\_ADDRESS 0x40    String apiKey = "\*\*\*\*\*\*\*\*\*\*";  const char\* ssid = "\*\*\*\*\*\*\*\*\*\*";          // Enter your WiFi Network's SSID  const char\* pass = "\*\*\*\*\*\*\*\*\*\*";  // Enter your WiFi Network's Password  const char\* server = "api.thingspeak.com";    INA226\_WE ina226 = INA226\_WE(I2C\_ADDRESS);    WiFiClient client;    void setup()  {    Serial.begin(115200);    while (!Serial); // wait until serial comes up on Arduino Leonardo or MKR WiFi 1010    Wire.begin();    ina226.init();      /\* Set Resistor and Current Range       if resistor is 5.0 mOhm, current range is up to 10.0 A       default is 100 mOhm and about 1.3 A\*/      ina226.setResistorRange(0.1, 1.3); // choose resistor 0.1 Ohm and gain range up to 1.3A      /\* If the current values delivered by the INA226 differ by a constant factor       from values obtained with calibrated equipment you can define a correction factor.       Correction factor = current delivered from calibrated equipment / current delivered by INA226\*/      ina226.setCorrectionFactor(0.93);      Serial.println("INA226 Current Sensor Example Sketch - Continuous");      ina226.waitUntilConversionCompleted(); //if you comment this line the first data might be zero      Serial.println("Connecting to ");    Serial.println(ssid);    WiFi.begin(ssid, pass);      while (WiFi.status() != WL\_CONNECTED)    {      delay(100);      Serial.print("\*");    }    Serial.println("");    Serial.println("WiFi connected");  }    void loop()  {    float shuntVoltage\_mV = 0.0;    float loadVoltage\_V = 0.0;    float batteryVoltage\_V = 0.0;    float current\_mA = 0.0;    float power\_mW = 0.0;      ina226.readAndClearFlags();    shuntVoltage\_mV = ina226.getShuntVoltage\_mV();    batteryVoltage\_V = ina226.getBusVoltage\_V();    current\_mA = ina226.getCurrent\_mA();    power\_mW = ina226.getBusPower();    loadVoltage\_V  = batteryVoltage\_V + (shuntVoltage\_mV / 1000);        Serial.print("Battery Voltage");    Serial.print(batteryVoltage\_V);    Serial.println("V");      Serial.print("Load Voltage: ");    Serial.print(loadVoltage\_V);    Serial.println("V");      Serial.print("Current: ");    Serial.print(current\_mA);    Serial.println("mA");      Serial.print("Power: ");    Serial.print(power\_mW);    Serial.println("mW");      if (!ina226.overflow)    {      Serial.println("Values OK - no overflow");    }    else    {      Serial.println("Overflow! Choose higher current range");    }      if (client.connect(server, 80)) {        String postStr = apiKey;      postStr += "&field1=";      postStr += String(batteryVoltage\_V);      postStr += "&field2=";      postStr += String(loadVoltage\_V);      postStr += "&field3=";      postStr += String(current\_mA);      postStr += "&field4=";      postStr += String(power\_mW);      postStr += "\r\n\r\n\r\n\r\n";        client.print("POST /update HTTP/1.1\n");      delay(100);      client.print("Host: api.thingspeak.com\n");      delay(100);      client.print("Connection: close\n");      delay(100);      client.print("X-THINGSPEAKAPIKEY: " + apiKey + "\n");      delay(100);      client.print("Content-Type: application/x-www-form-urlencoded\n");      delay(100);      client.print("Content-Length: ");      delay(100);      client.print(postStr.length());      delay(100);      client.print("\n\n\n\n");      delay(100);      client.print(postStr);      delay(100);    }    client.stop();    Serial.println("Sent Successfully :)");    Serial.println();    delay(3000);  } |

From the Board Manager, select the NodeMCU 1.0 Board and the COM port. Then hit the upload button to upload the code to the ESP8266 Board.

### ****Monitoring 12V Lead-Acid Battery on Thingspeak:****

Open the **Serial Monitor** after uploading the code. The ESP8266 will try connecting to the **WiFi Network**. Once it connects to the WiFi Network, it will display the Battery Voltage, Load Voltage, Current and Power.

If nothing is connected at the load, it will only display the Battery Voltage and else everything will appear zero.

[](https://how2electronics.com/wp-content/uploads/2023/08/Serial.png)

In order to test the current and power consumption, connect a Load like Motor, 12V LED Lights or anything else at the Load Terminal.

Now go to the private view of Thingspeak Dashboard. The Dashboard will shows the values of Battery Voltage, Load Voltage, Current and Power in graphical format as per time.

[](https://how2electronics.com/wp-content/uploads/2023/08/Graph-Thingspeak.png)

You may start charging the Battery using 12V Battery Charger and observe the change in Current and Voltage on the graph.

### ****Conclusion:****

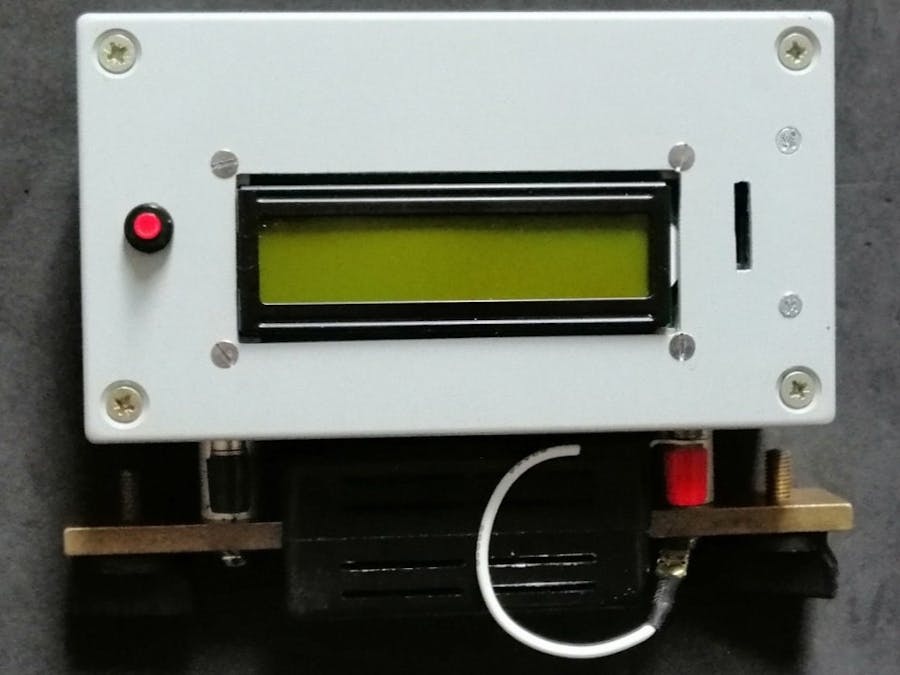
In **conclusion**, we successfully designed and built an IoT-based **12V Battery Monitoring** System that leverages the **ESP8266** and **INA226 DC Current Sensor** for optimal monitoring of lead-acid batteries. This sophisticated system safeguards **battery performance** and longevity by preventing overcharging and excessive discharging, which are common culprits of battery damage and system failure. By accurately measuring vital parameters such as **battery voltage, load voltage, current**, and **power**, our system promotes effective battery health management.

Furthermore, with the integration of the **ThingSpeak server**, users can effortlessly monitor their **battery status** from anywhere around the globe, providing enhanced convenience and flexibility. This comprehensive solution therefore, not only enhances the understanding of the **battery’s condition** in various states but also opens the door to potential advancements in **remote battery management** and other I**oT applications** in the future.

# The ArduINA226 Power Monitor

How to build a current, voltage and power datalogger with Arduino and the INA226 module.

[Intermediate](https://www.hackster.io/projects?difficulty=intermediate)Full instructions provided10 hours10,116



## Things used in this project

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Hardware components | | | | | |
|  | |  | | --- | | • INA226 breakout board | |  | | × | 1 |  |  |
|  | |  | | --- | | • Arduino Nano board | |  | | × | 1 |  |  |
|  | |  | | --- | | • SD board with 5V I/O levels | |  | | × | 1 |  |  |
|  | |  | | --- | | • 2 x 16 LCD display | |  | | × | 1 |  |  |
|  | |  | | --- | | • 20 A (or other current value) shunt | |  | | × | 1 |  |  |
| Software apps and online services | | | | | |
| Arduino IDE | |  | | --- | | [Arduino IDE](https://www.hackster.io/arduino/products/arduino-ide?ref=project-80d641) | |  | |  | |  |  |
|  | |  | | --- | | • https://github.com/peterus/INA226Lib | |  | |  | |  |  |
| Hand tools and fabrication machines | | | | | |
| Soldering iron (generic) | |  | | --- | | Soldering iron (generic) | |  | |  | |  |  |

## Story

With the powerful INA226 chip and Arduino, you can create a very accurate instrument capable of monitoring the electrical consumption of an electrical DC device. The current, voltage and power measurements are displayed on an LCD display and saved on a micro SD card.

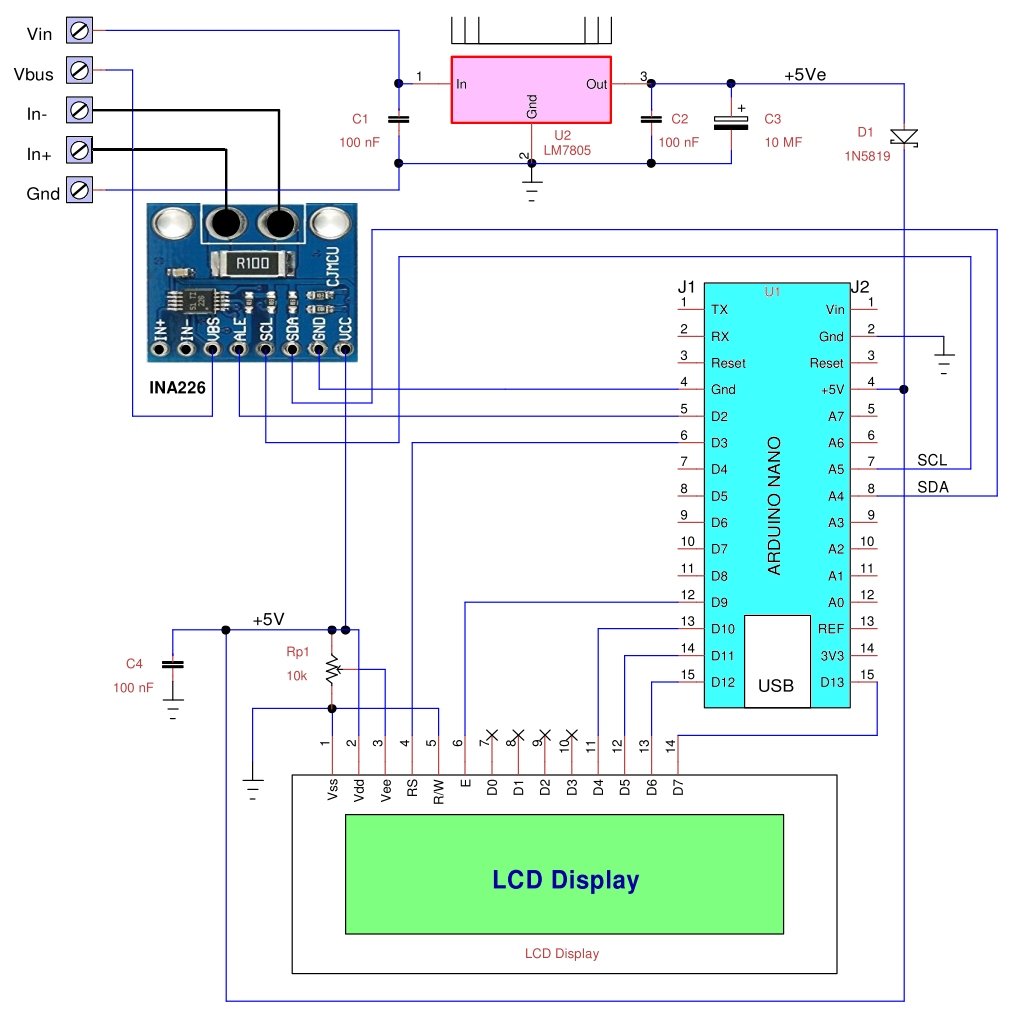
There are numerous possible applications for this monitoring tool: battery-powered devices such as scooters or pedal-assisted bicycles, photovoltaic panels, etc.

There are some libraries for the INA226 chip, I used the Korneliusz Jarzebski library which seems to me quite complete although I had to modify a couple of functions because they led to wrong results.

A complete and detailed description of this project and modified library functions can be found on my blog: [http://ardupiclab.blogspot.it/.](http://ardupiclab.blogspot.it/)

## Schematics

### arduina226\_natguBmVWY.jpg



## Code

### ArduINA226.ino

Arduino

The program requires only two parameters which are: rShunt and iMaxExpected. If you want to use an alert, you need to decide which signal is of interest. In my case I chose the maximum current, therefore, the maximum shunt voltage and use the enableShuntOverLimitAlert and setShuntVoltageLimit functions. In the case of a system powered by lead or lithium ion batteries, it would be more useful to check the minimum voltage of VBUS with the enableBusUnderLimitAlert and setBusVoltageLimit functions.

/\* Program ArduINA226 to current, voltage and power

with Arduino Nano and INA226 module

uses Korneliusz Jarzebski Library for INA226 (modified)

save measurements on SD if present

Giovanni Carrera, 14/03/2020

\*/

#include <SPI.h>

#include <LiquidCrystal.h>

#include <Wire.h>

#include <SD.h>

#include <INA226.h>

// LCD pins

#define rs 7

#define en 6

#define d4 5

#define d5 4

#define d6 A2

#define d7 A3

#define SSbutton 2

#define SD\_CS 10

// initialize the library by associating any needed LCD interface pin

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

INA226 ina;

char bline[17] = " ";// blank line

const int deltat= 500;// sampling period in milliseconds

unsigned long cmilli, pmilli;

boolean SDOk = true, FileHeader = true, ACQ = false;

unsigned int ns=0;

void setup() {

// set up the LCD's number of columns and rows:

lcd.begin(16, 2);

pinMode(SSbutton, INPUT);// Start/Stop button

pinMode(SD\_CS, OUTPUT);// SD chip select

// Default INA226 address is 0x40

ina.begin(0x40);

lcd.print("ArduINA226");

lcd.setCursor(0, 1);// print on the second row

lcd.print("Power Monitor");

// Configure INA226

ina.configure(INA226\_AVERAGES\_1, INA226\_BUS\_CONV\_TIME\_1100US, INA226\_SHUNT\_CONV\_TIME\_1100US, INA226\_MODE\_SHUNT\_BUS\_CONT);

// Calibrate INA226. Rshunt = 0.004 ohm, Max expected current = 20.48 A

ina.calibrate(0.004, 20.48);

ina.enableShuntOverLimitAlert();// enable Shunt Over-Voltage Alert, current over the limit

ina.setShuntVoltageLimit(0.08); // current limit = 20 A for 0.004 ohms

ina.setAlertLatch(true);

if (!SD.begin(SD\_CS)) {

LCDprintLine("SD not present!", 1);

delay(5000);

SDOk = false;

}

LCDprintLine("Push to Start", 1);

while (digitalRead(SSbutton) == HIGH) {};// wait for start

if (SDOk) ACQ = true;

}

void loop(){

cmilli = millis();

if (cmilli - pmilli > deltat) {

pmilli = cmilli;

float volts= ina.readBusVoltage();

float current = ina.readShuntCurrent();

LCDprintLine("V=", 0);// print bus voltage

lcd.print(volts,3);

float power = ina.readBusPower();// INA calculate power

// float power = volts \* current;// Arduino calculate power

lcd.print(" W=");// print power

lcd.print(power,4);

float Vshunt= ina.readShuntVoltage();

String dataString = String(volts,3)+','+String(current,4)+','+String(power,4)+','+String(Vshunt,6);

if (ACQ){

File dataFile = SD.open("powerlog.csv", FILE\_WRITE);

if (dataFile) {

if (FileHeader){// print file header

dataFile.print("Deltat [ms] = ");

dataFile.println(deltat);

dataFile.println("Vbus[V], Ishu [A], P [W], Vshu [V]");

FileHeader = false;

}

dataFile.println(dataString);

ns++;// number of acquired samples

dataFile.close();

if (digitalRead(SSbutton) == LOW && ns>=10){ //stop after at least 10 samples

ACQ = false;// stop acquisition

LCDprintLine(String(ns), 1);

lcd.print(" samples");

delay(5000);

}

} else {

LCDprintLine("Can't open file!", 1);

ACQ = false;

delay(5000);

}

}

if (ina.isAlert()) {

LCDprintLine("Shunt V=", 0);

lcd.print(Vshunt,5);

LCDprintLine("SOL ALERT", 1);

}

else {

LCDprintLine("I= ", 1);

lcd.print(current,3);

if (ACQ){

dataString = " N=" + String(ns);

lcd.print(dataString);// print number of acquired sample

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void LCDprintLine(String text, byte line){

lcd.setCursor(0, line);

lcd.print(bline);// clear the second row

lcd.setCursor(0, line);

lcd.print(text);// print text

}

This blog is dedicated to electronic projects and software due to the author’s enthusiasm for these activities. The main aim is to share to other enthusiasts the experience in electronic design of the author who disclaims all responsibility. All presented projects are realized and fully tested by the author who intends to preserve the intellectual property of the projects or information, whose utilization is intended only for non-professional purposes.



Powered by [**[Google Translate](https://translate.google.com/)Translate**](https://translate.google.com/)

## Sunday, March 22, 2020

### The ArduINA226 power monitor

**Introduction**

In the past I have developed various projects of ammeters based on Hall effect current sensors such as the ACS712, or on High-Side Current-Sense Amplifiers such as the MAX4080SASA or made with operational amplifiers. All these systems have an analog output which must then be digitized. The INA226 sensor has a digital output and incorporates a 16-bit ADC for which a high accuracy and precision is obtained.

It measures current and voltage and calculates power while Arduino communicates with the chip, presents the measurements on an LCD display and stores them on a micro SD card. This chip operates with a maximum voltage of 36 volts while the current is limited only by the shunt used.

There are some libraries for the INA226 chip, I used the Korneliusz Jarzebski library which seems to me quite complete even if I had to make some changes to two functions.

There are numerous possible applications for this monitoring tool: battery-powered devices such as scooters or pedal-assisted bicycles, photovoltaic panels, etc.

**The INA226 sensor**

In current measurements with the shunt there are two ways to insert it:

1)      To ground (low-side): the shunt is connected between the load and the ground.

2)      Towards the power supply (high-side): the shunt is connected between the power supply and the load.

The INA226 integrated circuit, by Texas Instruments, is a digital device that measures the current with a high-side or low-side shunt and also measures the voltage, calculates the power and provides a multifunctional alarm. The functional scheme is visible in figure 1.

|  |
| --- |
|  |
| Figure 1 |

The resolution of the shunt voltage is 2.5 mV with a full scale of 32768x2.5 = 81.92mV. For the VBUS voltage the resolution is 1.25 mV with a theoretical full-scale of 40.96 V even if the 36 V must not be exceeded. The resolution of the power is 25 times that of the current, with a full-scale that depends on the shunt used. So the system has a remarkable measurement accuracy.

The internal ADC is based on a 16-bit delta-sigma converter (ΔΣ) with a typical sampling frequency of 500 kHz (± 30%), so it is also suitable for currents that change rapidly over time.

The chip has 10 pins and very small dimensions, with a DGS case (VSSOP).

The INA226 is able to provide a hardware or software alert if a variable, selected by the user, has exceeded a limit. The user can select one of the five functions available to monitor and/or set the Conversion Ready bit. The five alert functions that can be monitored are:

•       Shunt Voltage Over-Limit (SOL): exceeding the maximum current threshold;

•       Shunt Voltage Under-Limit (SUL): exceeding the minimum current threshold;

•       Bus Voltage Over-Limit (BOL): exceeding the maximum voltage threshold;

•       Bus Voltage Under-Limit (BUL): exceeding the minimum voltage threshold;

•       Power Over-Limit (POL): exceeding the maximum power threshold;

The Alert output is open-drain and can be easily connected to a blocking device. The Alert output is open-drain and can be easily connected to a blocking circuit. For simplicity, I preferred to read the Shunt Voltage Over-Limit (SOL) type alert via software and present it on the display.

**The INA226 module**

On the market there are small breakout boards such as the one I used, visible in figure 2, of which I obtained the diagram in figure 3.

|  |
| --- |
|  |
| Figure 2 |

|  |
| --- |
|  |
| Figure 3 |

The device has two address pins, A0 and A1. The following table lists the pin connections for each of the 16 possible addresses.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A1** | **A0** | **Slave Address** | **A1** | **A0** | **Slave Address** |
| GND | GND | 1000000 | SDA | GND | 1001000 |
| GND | VS | 1000001 | SDA | VS | 1001001 |
| GND | SDA | 1000010 | SDA | SDA | 1001010 |
| GND | SCL | 1000011 | SDA | SCL | 1001011 |
| VS | GND | 1000100 | SCL | GND | 1001100 |
| VS | VS | 1000101 | SCL | VS | 1001101 |
| VS | SDA | 1000110 | SCL | SDA | 1001110 |
| VS | SCL | 1000111 | SCL | SCL | 1001111 |

The module mounts two pull-down resistors (R2 and R3), therefore the address is 0x40 if we don’t connect the jumpers placed on the opposite side to that of the components (see figure 2 on the right).

With the shunt resistor of 0.1 W (R100), mounted on the module, there is a current resolution of 2.5 mV / 0.1 = 0.025 mA, a full scale of 81.92mV/0.1W = 819.2 mA and a power resolution of 0.025 mA \*25 = 0.625 mW. I preferred to mount an external shunt for high currents, unsolder the internal one, and insert an RC filter, as shown in the diagram in figure 6.

**The LCD display**

I used a common two-line 16-character LCD display with Hitachi HD44780 compatible controller and a backlit equipped with high efficiency LED diodes that consumed about 20 mA, which I reduced to 10 mA with an external resistor in series with the internal one. The connections with Arduino Nano are as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **LCD**  **pin** | **function** | **Arduino**  **pin** | **LCD**  **pin** | **function** | **Arduino**  **pin** |
| 14 | D7  Data Bus Line | A3 | 5 | R/W - Read/Write | Gnd |
| 13 | D6  Data Bus Line | A2 | 4 | RS - Register Select | D7 |
| 12 | D5  Data Bus Line | D4 | 3 | VEE | - |
| 11 | D4  Data Bus Line | D5 | 2 | VCC (+5V) | VCC |
| 6 | E - Enable Signal | D6 | 1 | VSS (GND) | Gnd |

**The micro SD module**

In the latest version I added an SD module suitable for Arduino, that is, with 5 V power supply and logic levels. The appearance of the module used is visible in figure 4.

|  |
| --- |
|  |
| Figure 4 |

The following table shows the connections and pins dedicated to the microSD:

|  |  |  |
| --- | --- | --- |
| **J2**  **pin** | **SD module**  **name** | **Arduino**  **pin** |
| 6 | CS | D10 |
| 5 | SCK | D13 |
| 4 | MOSI | D11 |
| 3 | MISO | D12 |
| 2 | VCC | +5 V |
| 1 | GND | GND |

**Calculations for the shunt used**

I used a shunt I had at home, visible in figure 5, it was an accessory of the glorious ICE 680R analog tester.

|  |
| --- |
|  |
| Figure 5 |

As can be seen from the figure, it has an output of 100 mV with a current of 25 A, so Rs = 0.1/25 = 0.004 Ω.

I unsoldered the 0.1 Ω shunt, mounted on the INA226 module, and I soldered a 1 MF capacitor and two filter resistors. This expedient allows RC filtering, as suggested by the chip manufacturer.

With the full-scale shunt voltage in mind, the maximum measurable current now becomes:

*Ifs = 81.92mV/4 mΩ = 20.48 A*

The current resolution will be: 20.48 / 32768 = 0.625 mA.

The calibrate function has as input parameters the shunt resistance and the maximum current, in our case: rShunt = 0.004 Ω and iMaxExpected = 20.48 A.

It calculates several variables:

•       currentLSB = iMaxExpected / 32768 = 20.48 / 32768 = 0.625 [mA]

•       calibrationValue = 0.00512 / currentLSB / rShunt = 0.00512 / 0.000625 / 0.004 = 2048

•       powerLSB = currentLSB \* 25 = 0.000625 \* 25 = 15.625 [mW]

Another data to pass to the program is the current threshold of the setShuntVoltageLimit function.

If we set the upper limit of the instrument at 20 A, we can calculate:

setShuntVoltageLimit = 0.004 Ω \* 20 A = 0.08 V

**The prototipe**

I used an Arduino Nano, of course you can use other Arduino boards, such as Arduino Uno or Arduino Pro but this board is very compact, complete with USB adapter, and can be mounted on a prototype pre-drilled pcb.

Arduino Nano has its own 5V regulator, a Low DropOut type AMS1117, but I preferred to power the system with a common LM7805, this is because the former accepts a maximum input voltage of 15V against the 35V of the second. If I have to power ArduINA226 with the voltage I have to monitor, it is better to have compatible values. Another reason is the backlight of the display. The consumption of the whole system is around 45 mA. The Schottky D1 diode, which has a drop of only 0.2V, is used to avoid conflicts between external power supply and USB power supply.

Figure 6 shows the wiring diagram of my realization.

|  |
| --- |
|  |
| Figure 6 |

**List of components used**

|  |  |  |  |
| --- | --- | --- | --- |
| **component** | **description** | **component** | **description** |
| C1,C2,C4 | 100 nF 50V, ceramic capacitor | R4 | 100 ohm, ±5%, 0.5 W resistor |
| C3 | 10 MF 50V, electrolytic capacitor | U1 | Arduino Nano board |
| C4 | 1 MF 25V, ceramic capacitor | U2 | LM7805, 5V regulator |
| D1 | 1N5819, Schottky diode | U3 | INA226 module |
| Rp1 | 22kW resistive trimmer | display | 2 rows x 16 columns LCD |
| R1, R2 | 10 ohm, ±5%, 0.25 W resistor | modulo SD | micro SD board with 5V levels |
| R3 | 4.7 kohm, ±5%, 0.25 W resistor | Sw1 | Normally Open (NO) push button |

**Note**: capacity prefix 'M' stands for microfarad (1e-6 F)

Figures 7 shows the appearance of my prototype. I screwed the shunt directly on the instrument case, note also the slot for the micro SD card.

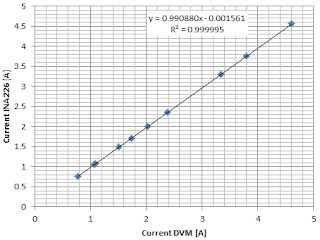
|  |
| --- |
|  |
| Figure 7 |

**The tests**

Figure 8 shows the instrument running during the tests.

|  |
| --- |
|  |
| Figure 8 |

With a ten measuring points, compared with a precision multimeter, the results have been very good, as can be seen from the graph in figure 9.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEgoz3A3vJZhC6eVe7TkmLCzvkae7hCxMZqPL1tHPmmLqTfX1fBRGFBMl0xFONiCA-hlm0-Pe0LDdzsVRkwx1k1vCtwOCHFhJvItC5ZwZN8Mo9UQHtT5QvWXwtIRu5wjliWPuCddE1s6xxiQ/s1600/grafico.jpg)

In my case, the VBUS voltage was very accurate and required no correction. The current is slightly lower than the expected value (correction of 1.0092) and the linearity was excellent with R = 0.999995.

**Changes to the INA226.cpp library**

There are some libraries for the INA226 chip, I used the Korneliusz Jarzebski library which seems to me quite complete. I changed the calibrate function because the shunt current was incorrect. The setShuntVoltageLimit function was also wrong. Then I realized that the latter, in version 1.1 of the library, had been corrected. Here are the modified functions, to replace the original ones:

bool INA226::calibrate(float rShunt, float iMaxExpected){ // MODIFIED by GCAR

    uint16\_t calibrationValue;

    float iMaxPossible;

    iMaxPossible = vShuntMax / rShunt;

    currentLSB = iMaxExpected / 32768;// calculate current resolution

    powerLSB = currentLSB \* 25;// power resolution

    calibrationValue = (uint16\_t)((0.00512) / (currentLSB \* rShunt));

    writeRegister16(INA226\_REG\_CALIBRATION, calibrationValue);

    return true;

}

void INA226::setShuntVoltageLimit(float voltage){// MODIFIED by GCAR

    uint16\_t value = voltage/2.5e-6;

    writeRegister16(INA226\_REG\_ALERTLIMIT, value);

}

**The program**

The program requires only two parameters which are: rShunt and iMaxExpected. If you want to use an alert, you need to decide which signal is of interest. In my case I chose the maximum current, therefore, the maximum shunt voltage and use the enableShuntOverLimitAlert and setShuntVoltageLimit functions. In the case of a system powered by lead or lithium ion batteries, it would be more useful to check the minimum voltage of VBUS with the enableBusUnderLimitAlert and setBusVoltageLimit functions.

If we have not inserted the SD card or if it is not valid, the message "SD not present!" is printed.

At the end of the setup () function, the program prints "Push to Start" and waits for the SS button to be pressed, if the SD is present, begins to acquire the measurements on file, otherwise it prints them only on the display.

As for the LCD display, two rows and 16 columns are just enough, so you need to manage your prints well. Taking into account the resolution of the variables to be printed, these are their formats:

•       Bus voltage: V = xx.xxx (8 characters)

•       Shunt current: I = xx.xxx (8 characters)

•       Bus power: W = xxx.xx (8 characters printed)

So, in the first line I enter the voltage and power, separated by a space, in the second the current and the number of samples saved on SD (if inserted), in this way:

V=xx.xxx W=xxx.xx

I=xx.xxx N=xxxxx

If the alert that I set as exceeding the maximum current limit (SOL, Shunt Voltage Over-Limit) occurs, the "SOL ALERT" print will appear on the second line, instead of the current value and on the first the shunt voltage:

Shunt V=xx.xxxxx

SOL ALERT

**The code**

/\* Program ArduINA226 to current, voltage and power

 with Arduino Nano and INA226 module

uses Korneliusz Jarzebski Library for INA226 (modified)

save measurements on SD if present

Giovanni Carrera, 14/03/2020

 \*/

#include <SPI.h>

#include <LiquidCrystal.h>

#include <Wire.h>

#include <SD.h>

#include <INA226.h>

// LCD pins

#define rs 7

#define en 6

#define d4 5

#define d5 4

#define d6 A2

#define d7 A3

#define SSbutton 2

#define SD\_CS 10

// initialize the library by associating any needed LCD interface pin

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

INA226 ina;

char bline[17] = "                ";// blank line

const int deltat= 500;// sampling period in milliseconds

unsigned long cmilli, pmilli;

boolean SDOk = true, FileHeader = true, ACQ = false;

unsigned int ns=0;

void setup() {

  // set up the LCD's number of columns and rows:

  lcd.begin(16, 2);

  pinMode(SSbutton, INPUT);// Start/Stop button

  pinMode(SD\_CS, OUTPUT);// SD chip select

  // Default INA226 address is 0x40

  ina.begin(0x40);

  lcd.print("ArduINA226");

  lcd.setCursor(0, 1);// print on the second row

  lcd.print("Power Monitor");

  // Configure INA226

  ina.configure(INA226\_AVERAGES\_1, INA226\_BUS\_CONV\_TIME\_1100US, INA226\_SHUNT\_CONV\_TIME\_1100US, INA226\_MODE\_SHUNT\_BUS\_CONT);

  // Calibrate INA226. Rshunt = 0.004  ohm, Max expected current = 20.48 A

  ina.calibrate(0.004, 20.48);

  ina.enableShuntOverLimitAlert();// enable Shunt Over-Voltage Alert, current over the limit

  ina.setShuntVoltageLimit(0.08); // current limit = 20 A for 0.004 ohms

  ina.setAlertLatch(true);

  if (!SD.begin(SD\_CS)) {

    LCDprintLine("SD not present!", 1);

    delay(5000);

    SDOk = false;

  }

  LCDprintLine("Push to Start", 1);

  while (digitalRead(SSbutton) == HIGH) {};// wait for start

  if (SDOk) ACQ = true;

}

void loop(){

  cmilli = millis();

  if (cmilli - pmilli > deltat) {

    pmilli = cmilli;

    float volts= ina.readBusVoltage();

    float current = ina.readShuntCurrent();

    LCDprintLine("V=", 0);// print bus voltage

    lcd.print(volts,3);

    float power = ina.readBusPower();// INA calculate power

//    float power = volts \* current;// Arduino calculate power

    lcd.print(" W=");// print power

    lcd.print(power,4);

    float Vshunt= ina.readShuntVoltage();

    String dataString = String(volts,3)+','+String(current,4)+','+String(power,4)+','+String(Vshunt,6);

    if (ACQ){

      File dataFile = SD.open("powerlog.csv", FILE\_WRITE);

      if (dataFile) {

        if (FileHeader){// print file header

          dataFile.print("Deltat [ms] = ");

          dataFile.println(deltat);

          dataFile.println("Vbus[V], Ishu [A], P [W], Vshu [V]");

          FileHeader = false;

        }

        dataFile.println(dataString);

        ns++;// number of acquired samples

        dataFile.close();

        if (digitalRead(SSbutton) == LOW && ns>=10){ //stop after at least 10 samples

          ACQ = false;// stop acquisition

          LCDprintLine(String(ns), 1);

          lcd.print(" samples");

          delay(5000);

        }

      } else {

        LCDprintLine("Can't open file!", 1);

        ACQ = false;

        delay(5000);

      }

    }

    if (ina.isAlert()) {

      LCDprintLine("Shunt V=", 0);

      lcd.print(Vshunt,5);

      LCDprintLine("SOL ALERT", 1);

    }

    else {

      LCDprintLine("I= ", 1);

      lcd.print(current,3);

      if (ACQ){

        dataString = " N=" + String(ns);

        lcd.print(dataString);// print number of acquired sample

      }

    }

  }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void LCDprintLine(String text, byte line){

   lcd.setCursor(0, line);

   lcd.print(bline);// clear the second row

   lcd.setCursor(0, line);

   lcd.print(text);// print text

}

The system can operate with even very small deltat periods, but the display, printouts and writing to file take some time: if we want to reduce it, for example to 100 ms, the measurements must be shown every 5 samples. Some hardware circuits can also be implemented, such as using the alert output to disconnect the power supply to the load via a static or electromechanical relay or to make a buzzer sound via Arduino.

ArduINa226, if a micro SD card has been inserted, transfers the measurements to the "powerlog.csv" file.

The acquisition starts by pressing the 'SS' (Start / Stop) button which also serves to end the recording after at least ten samples.

The first two lines contain the deltat sampling period in milliseconds and the names of the signals, as seen in the following example:

Deltat [ms] = 500

Vbus[V], Ishu [A], P [W], Vshu [V]

10.021,0.0950,0.9531,0.000375

10.023,0.0944,0.9531,0.000377

10.023,0.0944,0.9531,0.000375

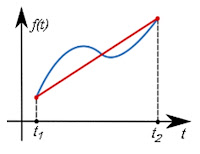
……

The data are appended on the file which is not overwritten.

**If we need to calculate the energy**

The energy supplied by the source is the integral of the power over time. The simplest system for making the definite integral of a function, although roughly enough, is that of trapezoids. The power curve is approximated in many trapezoids, of which it is easy to calculate the area, for two points we have:

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEgOYZnl26UFOWnNn4atodeq_go_581VpEb-yAUjKeuKROJEaoqpIRXWtgWDUliremow6Nivesixi8kmVJ_w94HIwRA7X0clhxyuBnf0dz0hH08LAxuINr2YC8iMVe40X_g6wznex0gDZneq/s1600/int+trapezi+2.jpg)

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEgShplucs_ipcWV8FhrHkgY8tHmqATDpNrkA6mzPvgYIBAGQkXVeN0h2srIWhh8tZNkuzoMYUspI66bM80SvlnpXzoUSHAHStNMhMApPOxHFUU0IBPaDQRG4okAn1gfHf0nMQtl-9J9oSLA/s1600/int+trapezi+1.jpg)

This method will give better results for smaller time intervals.

The integral will be equal to the sum of the individual areas. In our case: (t2-t1) = deltat and f(t) is the power P(t), so:

*E2 = deltat \* (P1 + P2) / 2 + E1 [J]*

Where P1 and P2 are the powers in two successive instants t1, t2 and E1, E2 the respective energies expressed in *W×s = joules*, which can be easily expressed in watt hours [Wh] dividing the joules by 3600.

Here's how to calculate energy with Arduino:

watthour = watthour + (power + ppower)/2.0\*deltath; // integer for trapezoids

ppower = power; // then updates the power

Where power is the current power, ppower that of the previous instant and

deltath = deltat / 3.6e6; // sampling period in hours (deltat in [ms])

At the beginning you have to reset the watthour variable.

I didn't add this function because I had no more space on the display and I preferred to calculate the energy on the PC with more sophisticated integration algorithms.

**References**

1.      “INA226 Bi-directional Current/Power Monitor Arduino Library”, Korneliusz Jarzebski, [https://github.com/jarzebski/Arduino-INA226](https://github.com/jarzebski/Arduino-INA226-)

2.      “INA226 High-Side or Low-Side Measurement, Bi-Directional Current and Power Monitor with I2C Compatible Interface”, Texas Instruments, SBOS547A –June 2011–revised august 2015

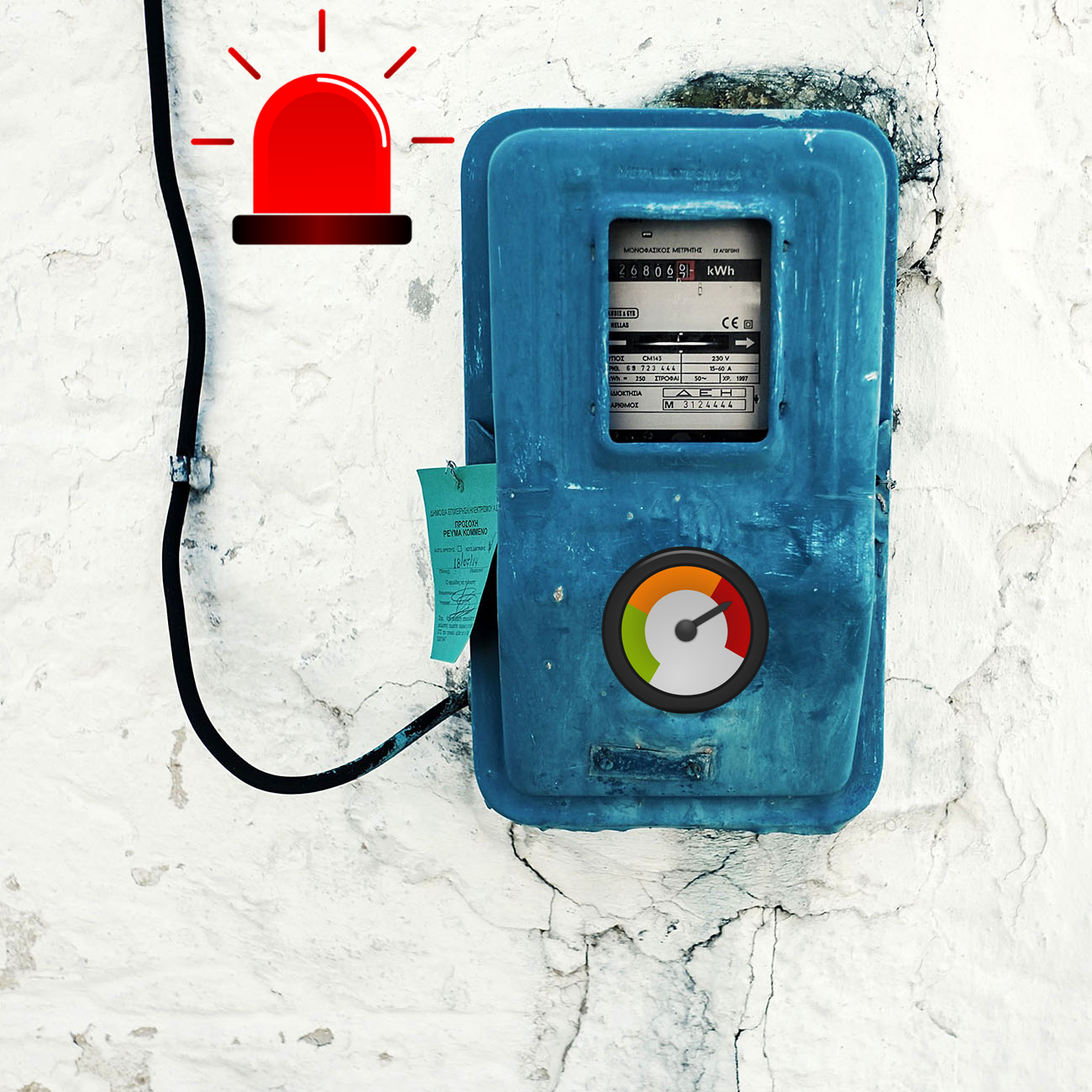
3.      “VARDULOG - Data logger dei consumi di un apparato elettrico”, Giovanni Carrera, rivista Fare Elettronica n. 361-362, Novembre-Dicembre 2015.

4.      “Progetto ArduWattmeter”, Giovanni Carrera, rivista Fare Elettronica n. 367/368 ¬ Giugno/Luglio 2016.

5.      “ArduAmmeter, a simple circuit for measuring electrical current with Arduino”, Giovanni Carrera, <https://www.hackster.io/ArduPic/arduammeter-b59d40>, July 16, 2019.

# NA226 Current and Power Sensor

POSTED ON [JANUARY 3, 2021](https://wolles-elektronikkiste.de/en/ina226-current-and-power-sensor) BY [WOLFGANG EWALD](https://wolles-elektronikkiste.de/en/author/w0ellelle_ewood)



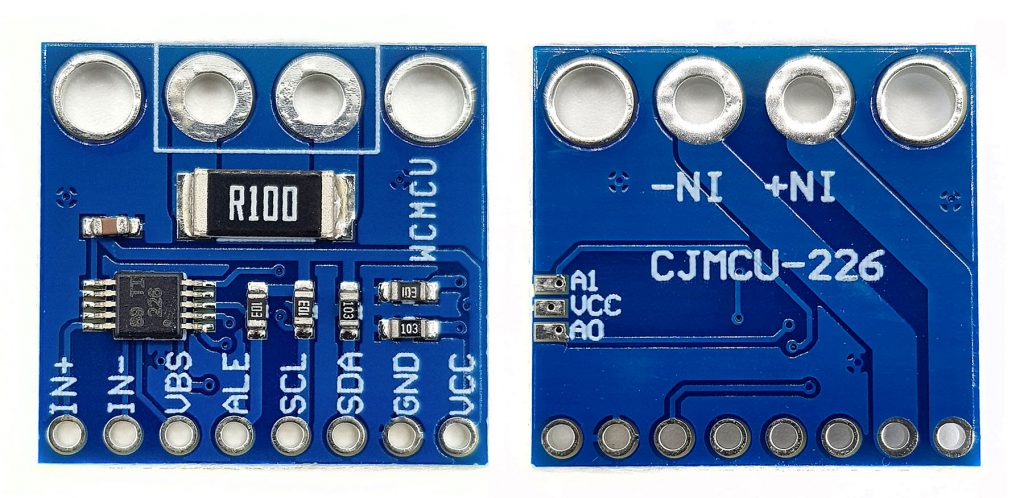
## About this post

After reporting on the INA219 in [my last post](https://wolles-elektronikkiste.de/en/ina219-current-and-power-sensor" \t "_blank), I would like to introduce the INA226 and my associated library [INA226\_WE](https://github.com/wollewald/INA226_WE) in this article.

In a first approximation, the INA226 is an INA219 with alarm function, which is particularly well suited for monitoring currents. In addition, the INA226 can be used on both the high-side and the low-side. I will come back to further differences to the INA219 during the article.

First of all, I will deal with the measuring principle and the technical data. Then I present the library with its numerous example sketches. Finally, the last part is for those who want to go deeper. It deals with the inner details of the INA226 and the library.

## The measuring principle

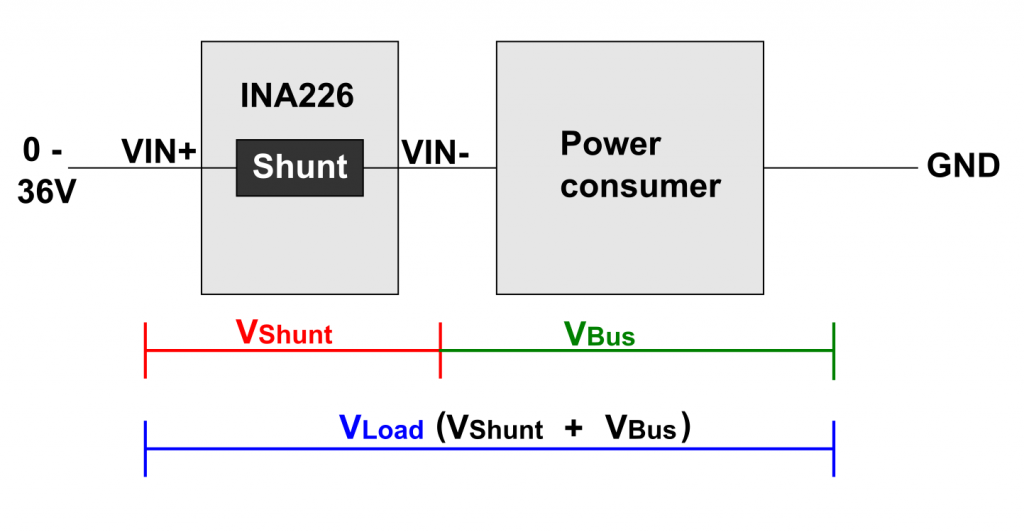
[](https://wolles-elektronikkiste.de/wp-content/uploads/2020/06/INA226_Modul-1024x498.jpg)An INA226 module, front and back

Basically, the INA226 works the same way as the INA219. You conduct the current to be measured via the terminals IN+ and IN- through a [shunt](https://de.wikipedia.org/wiki/Shunt_(Elektrotechnik)" \t "_blank) (current resistor). An A/D converter measures the voltage drop across the shunt and the INA226 calculates the current from this.

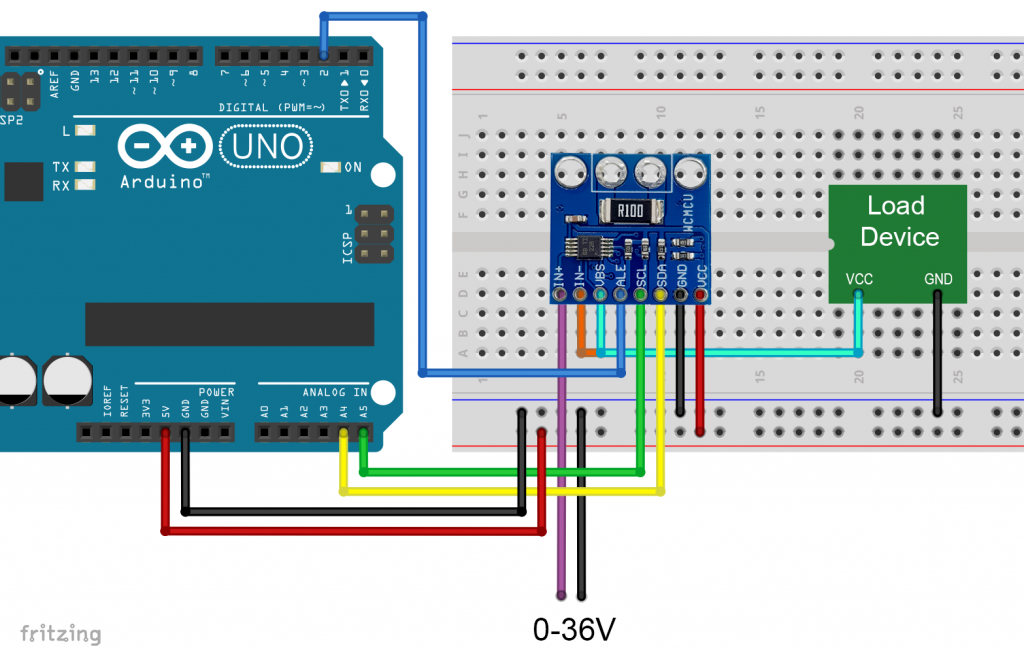
If you use the bare INA226 (the ten-pin IC on the module), then you are free to choose the size of the shunt. The modules have a shunt of 0.1 ohms. In any case, this applies to all models that I have dealt with.

In addition, the INA226 measures the bus voltage, i.e. the voltage drop across consumer. This happens between VBUS and GND. The INA219, on the other hand, measures the bus voltage between IN and GND. That’s why you have to place the INA219 before the consumer (high-side). With the INA226, you are more flexible, you can use it on both the high side and the low-side.

The INA226 calculates the power from the current and the voltage drop across the consumer. It saves the measured values in its data registers, where you can read them by I2C.

[](https://wolles-elektronikkiste.de/wp-content/uploads/2021/01/INA226_scheme-1024x532.png)INA226 in high-side configuration

### Typical circuit

[](https://wolles-elektronikkiste.de/wp-content/uploads/2020/06/INA226_Wiring_HiSide-1024x655.png)Typical INA226 circuit (used for examples)

I used the above (high-side) circuit for all example sketches. It is important that the INA226 and the consumer have a common GND, otherwise the measurement of the bus voltage will not work. If you swap IN+ and IN- you will get negative values for the shunt voltage and current.

## Some technical data of the INA226 module

* Bus voltage: 0 – 36 volts
* Maximum bus current: 800 milliampere
* Supply voltage: 3 – 5.5 volts
* Power consumption (self-determined):
  + Continuous mode: 0.35 mA
  + Power-Down Mode: 2.3.µA
* Measurement modes: continuous or on-demand (“triggered”);
* Averaging of 1, 4, 64, 128, 256, 512 or 1024 individual measurements
* A/D conversion time adjustable in eight levels: 0.14 to 8.2 ms
* Data registers:
  + Shunt voltage register
  + Bus voltage register
  + Current register
  + Power register
* Communication via I2C, 4 addresses adjustable (module back):
  + 0x40: A0, A1 open
  + 0x41: A0 closed, A1 open
  + 0x44: A0 open, A1 closed
  + 0x45: A0, A1 closed
* Programmable alarm pin for limit violations and available data

Further technical data can be found in the manufacturer’s [data sheet.](https://www.ti.com/lit/ds/symlink/ina226.pdf?ts=1592146667758&ref_url=https%253A%252F%252Fwww.google.de%252F" \t "_blank)

Most INA226 modules have the 0.1 Ω shunt. However, there are also models with 0.01 or 0.02 Ω, for example. With the 0.1 Ω shunt, the maximum current is 0.819175 A. If you use the bare module, you are correspondingly more flexible. I have implemented a function that allows you to use different shunts.

## Use of the INA226 library

You can download the library INA226\_WE [here](https://github.com/wollewald/INA226_WE" \t "_blank) from GitHub or install it directly with the Library Manager of the Arduino IDE.

I’ve created a total of seven example sketches that I use to present the functions of the library. I will focus most on the example of continuous mode. Many of the functions are used in all sketches and therefore only need to be explained once.

### Example 1: Continuous Mode

After you have installed the library and wired your INA226, upload the sketch “Continuous.ino”.

**Continuous.ino**

#include <Wire.h>

#include <INA226\_WE.h>

#define I2C\_ADDRESS 0x40

/\* There are several ways to create your INA226 object:

\* INA226\_WE ina226 = INA226\_WE() -> uses Wire / I2C Address = 0x40

\* INA226\_WE ina226 = INA226\_WE(ICM20948\_ADDR) -> uses Wire / I2C\_ADDRESS

\* INA226\_WE ina226 = INA226\_WE(&wire2) -> uses the TwoWire object wire2 / I2C\_ADDRESS

\* INA226\_WE ina226 = INA226\_WE(&wire2, I2C\_ADDRESS) -> all together

\* Successfully tested with two I2C busses on an ESP32

\*/

INA226\_WE ina226 = INA226\_WE(I2C\_ADDRESS);

**void** setup() {

Serial.begin(9600);

Wire.begin();

ina226.init();

/\* Set Number of measurements for shunt and bus voltage which shall be averaged

\* Mode \* \* Number of samples \*

AVERAGE\_1 1 (default)

AVERAGE\_4 4

AVERAGE\_16 16

AVERAGE\_64 64

AVERAGE\_128 128

AVERAGE\_256 256

AVERAGE\_512 512

AVERAGE\_1024 1024

\*/

//ina226.setAverage(AVERAGE\_16); // choose mode and uncomment for change of default

/\* Set conversion time in microseconds

One set of shunt and bus voltage conversion will take:

number of samples to be averaged x conversion time x 2

\* Mode \* \* conversion time \*

CONV\_TIME\_140 140 µs

CONV\_TIME\_204 204 µs

CONV\_TIME\_332 332 µs

CONV\_TIME\_588 588 µs

CONV\_TIME\_1100 1.1 ms (default)

CONV\_TIME\_2116 2.116 ms

CONV\_TIME\_4156 4.156 ms

CONV\_TIME\_8244 8.244 ms

\*/

//ina226.setConversionTime(CONV\_TIME\_1100); //choose conversion time and uncomment for change of default

/\* Set measure mode

POWER\_DOWN - INA226 switched off

TRIGGERED - measurement on demand

CONTINUOUS - continuous measurements (default)

\*/

//ina226.setMeasureMode(CONTINUOUS); // choose mode and uncomment for change of default

/\* If the current values delivered by the INA226 differ by a constant factor

from values obtained with calibrated equipment you can define a correction factor.

Correction factor = current delivered from calibrated equipment / current delivered by INA226

\*/

// ina226.setCorrectionFactor(0.95);

Serial.println("INA226 Current Sensor Example Sketch - Continuous");

ina226.waitUntilConversionCompleted(); //if you comment this line the first data might be zero

}

**void** loop() {

float shuntVoltage\_mV = 0.0;

float loadVoltage\_V = 0.0;

float busVoltage\_V = 0.0;

float current\_mA = 0.0;

float power\_mW = 0.0;

ina226.readAndClearFlags();

shuntVoltage\_mV = ina226.getShuntVoltage\_mV();

busVoltage\_V = ina226.getBusVoltage\_V();

current\_mA = ina226.getCurrent\_mA();

power\_mW = ina226.getBusPower();

loadVoltage\_V = busVoltage\_V + (shuntVoltage\_mV/1000);

Serial.print("Shunt Voltage [mV]: "); Serial.println(shuntVoltage\_mV);

Serial.print("Bus Voltage [V]: "); Serial.println(busVoltage\_V);

Serial.print("Load Voltage [V]: "); Serial.println(loadVoltage\_V);

Serial.print("Current[mA]: "); Serial.println(current\_mA);

Serial.print("Bus Power [mW]: "); Serial.println(power\_mW);

**if**(!ina226.overflow){

Serial.println("Values OK - no overflow");

}

**else**{

Serial.println("Overflow! Choose higher current range");

}

Serial.println();

delay(3000);

}

#### Parameter setting using the example of continuous mode

INA226\_WE ina226 = INA226\_WE() creates the IN226 object. You can pass the I2C address and/or a Wire object. The latter allows you to use e.g. both I2C busses of an ESP32.

The function init() activates the INA226 with the default values. To change these basic settings, you can modify the following parameters in the setup:

1. Set the number of individual measurements for the shunt and bus voltage conversion with setAverage()
   * 1, 4, 16, 64, 128, 256, 512 or 1024 individual measurements are averaged
2. Set the A/D conversion time for the shunt and bus voltage with setConvTime()
   * 8 levels adjustable between 140 µs and 8.244 ms
   * Note: obtaining a data set of shunt and bus voltage takes twice the time
3. Set the measurement mode with setMeasureMode()
   * CONTINUOUS – continuous measurement
   * TRIGGERED – “on request”: I explain in the next example.
   * POWER\_DOWN – turns off the INA226. But better use the more comfortable powerDown() function, which is explained below.
   * The INA226 actually allows determining shunt or bus voltages – but I did not implement that. Using my library, the measurements are only available in a double pack.

With setCorrectionFactor() you can introduce a correction factor if the current values determined with the INA226 should differ from those determined by you, for example, with calibrated measuring instruments. The factor is the quotient of the exact and INA226 value.

#### Other functions used in the example

You can query the data registers of the INA226 at any time. They contain the last measured value. Before the first measurement is completed, all values are zero. With waitUntilConversionCompleted() you can wait until the current measurement is complete.

Using readAndClearFlags() the overflow and alarm flags are read. In this example sketch, we need this call only to update the state of the variable overflow, which – if true – signals the overflow of a register.

The functions for reading the data registers, such as getShuntVoltage\_mV(), should be self-explanatory.

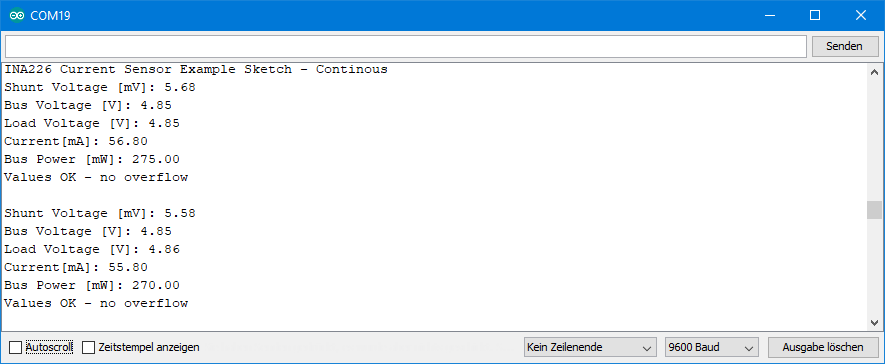
#### Calculation of the measurement time

The duration of a measurement is as follows:

𝐷𝑢𝑟𝑎𝑡𝑖𝑜𝑛=𝑁𝑢𝑚𝑏𝑒𝑟𝑚𝑒𝑎𝑠𝑢𝑟𝑒𝑚𝑒𝑛𝑡𝑠(𝐴𝑣𝑒𝑟𝑎𝑔𝑒𝑠)⋅𝐶𝑜𝑛𝑣𝑒𝑟𝑠𝑖𝑜𝑛 𝑇𝑖𝑚𝑒⋅2*Duration*=*Numbermeasurements*​(*Averages*)⋅*Conversion* *Time*⋅2

#### Output

And this is what the output of the sketch looks like on the serial monitor:

[](https://wolles-elektronikkiste.de/wp-content/uploads/2020/06/Ausgabe_Cont_INA226.png)Output of the continuous sketch

### Example 2: On-Demand (Triggered) Mode

You set the triggered mode with setMeasureMode(TRIGGERED) (line 47 in Triggered.ino).   Each measurement is started manually with startSingleMeasurement() (line 74). I have programmed the function to automatically wait until the current readings are available. So you **don’t** need to call waitUntilConversionCompleted() in triggered mode.

Otherwise, the sketch is identical to Continuous.ino.

**Triggered.ino**

#include <Wire.h>

#include <INA226\_WE.h>

#define I2C\_ADDRESS 0x40

INA226\_WE ina226(I2C\_ADDRESS);

// INA226\_WE ina226 = INA226\_WE(); // Alternative: sets default address 0x40

**void** setup() {

Serial.begin(9600);

Wire.begin();

ina226.init();

/\* Set Number of measurements for shunt and bus voltage which shall be averaged

\* Mode \* \* Number of samples \*

AVERAGE\_1 1 (default)

AVERAGE\_4 4

AVERAGE\_16 8

AVERAGE\_64 64

AVERAGE\_128 128

AVERAGE\_256 256

AVERAGE\_512 512

AVERAGE\_1024 1024

\*/

//ina226.setAverage(AVERAGE\_1); // choose mode and uncomment for change of default

/\* Set conversion time in microseconds

One set of shunt and bus voltage conversion will take:

number of samples to be averaged x conversion time x 2

\* Mode \* \* conversion time \*

CONV\_TIME\_140 140 µs

CONV\_TIME\_204 204 µs

CONV\_TIME\_332 332 µs

CONV\_TIME\_588 588 µs

CONV\_TIME\_1100 1.1 ms (default)

CONV\_TIME\_2116 2.116 ms

CONV\_TIME\_4156 4.156 ms

CONV\_TIME\_8244 8.244 ms

\*/

//ina226.setConversionTime(CONV\_TIME\_1100); //choose conversion time and uncomment for change of default

/\* Set measure mode

POWER\_DOWN - INA219 switched off

TRIGGERED - measurement on demand

CONTINUOUS - Continuous measurements (default)

\*/

ina226.setMeasureMode(TRIGGERED); // choose mode and uncomment for change of default

/\* If the current values delivered by the INA226 differ by a constant factor

from values obtained with calibrated equipment you can define a correction factor.

Correction factor = current delivered from calibrated equipment / current delivered by INA226

\*/

// ina226.setCorrectionFactor(0.95);

Serial.println("INA226 Current Sensor Example Sketch - Triggered");

// ina226.waitUntilConversionCompleted(); //makes no sense - in triggered mode we wait anyway for completed conversion

}

**void** loop() {

float shuntVoltage\_mV = 0.0;

float loadVoltage\_V = 0.0;

float busVoltage\_V = 0.0;

float current\_mA = 0.0;

float power\_mW = 0.0;

ina226.startSingleMeasurement();

ina226.readAndClearFlags();

shuntVoltage\_mV = ina226.getShuntVoltage\_mV();

busVoltage\_V = ina226.getBusVoltage\_V();

current\_mA = ina226.getCurrent\_mA();

power\_mW = ina226.getBusPower();

loadVoltage\_V = busVoltage\_V + (shuntVoltage\_mV/1000);

Serial.print("Shunt Voltage [mV]: "); Serial.println(shuntVoltage\_mV);

Serial.print("Bus Voltage [V]: "); Serial.println(busVoltage\_V);

Serial.print("Load Voltage [V]: "); Serial.println(loadVoltage\_V);

Serial.print("Current[mA]: "); Serial.println(current\_mA);

Serial.print("Bus Power [mW]: "); Serial.println(power\_mW);

**if**(!ina226.overflow){

Serial.println("Values OK - no overflow");

}

**else**{

Serial.println("Overflow! Choose higher current range");

}

Serial.println();

delay(3000);

}

### Example 3: Power-Down Mode

In power-down mode, you bring down the power consumption of the INA226 from approx. 0.35 mA to approx. 2.3 µA (own measurements).

The example sketch PowerDown.ino shows the power-down mode in action. The sketch initializes the INA226 with the default parameters. Five sets of measurements are output every three seconds. The function powerDown() then saves the contents of the configuration register and disables the INA226. The function powerUp() writes back the copy of the configuration register. On the one hand, this writing process awakens the INA226, on the other hand it ensures that the INA226 returns to the previously selected mode (here: continuous).

**PowerDown.ino**

#include <Wire.h>

#include <INA226\_WE.h>

#define I2C\_ADDRESS 0x40

INA226\_WE ina226(I2C\_ADDRESS);

// INA226\_WE ina226 = INA226\_WE(); // Alternative: sets default address 0x40

**void** setup() {

Serial.begin(9600);

Wire.begin();

// default parameters are set - for change check the other examples

ina226.init();

Serial.println("INA226 Current Sensor Example Sketch - PowerDown");

Serial.println("Continuous Sampling starts");

Serial.println();

}

**void** loop() {

**for**(int i=0; i<5; i++){

continuousSampling();

delay(3000);

}

Serial.println("Power down for 10s");

ina226.powerDown();

**for**(int i=0; i<10; i++){

Serial.print(".");

delay(1000);

}

Serial.println("Power up!");

Serial.println("");

ina226.powerUp();

}

**void** continuousSampling(){

float shuntVoltage\_mV = 0.0;

float loadVoltage\_V = 0.0;

float busVoltage\_V = 0.0;

float current\_mA = 0.0;

float power\_mW = 0.0;

ina226.readAndClearFlags();

shuntVoltage\_mV = ina226.getShuntVoltage\_mV();

busVoltage\_V = ina226.getBusVoltage\_V();

current\_mA = ina226.getCurrent\_mA();

power\_mW = ina226.getBusPower();

loadVoltage\_V = busVoltage\_V + (shuntVoltage\_mV/1000);

Serial.print("Shunt Voltage [mV]: "); Serial.println(shuntVoltage\_mV);

Serial.print("Bus Voltage [V]: "); Serial.println(busVoltage\_V);

Serial.print("Load Voltage [V]: "); Serial.println(loadVoltage\_V);

Serial.print("Current[mA]: "); Serial.println(current\_mA);

Serial.print("Bus Power [mW]: "); Serial.println(power\_mW);

**if**(!ina226.overflow){

Serial.println("Values OK - no overflow");

}

**else**{

Serial.println("Overflow! Choose higher current range");

}

Serial.println();

}

### Example 4: Conversion Ready Alarm

Now upload the sketch “Continuous\_Alert\_Controlled.ino”. With this example, you get to know the alert pin. With 1024 the maximum number of measurements to be averaged is used. Then we go to the upper limit of the conversion time, namely 8.244 milliseconds. This means that the combination of shunt and bus voltage measurement takes approximately 16.9 seconds. As measurement mode we select CONTINUOUS. The function enableConvReadyAlert() activates the alert pin, which is active-low by default. The alert pin is connected to the Arduino Pin 2, for which we set up an interrupt.

When a measurement is finished, the alert pin goes to LOW and an interrupt is triggered. The variable “event” becomes true and the if block is processed in the main loop. First, readAndClearFlags() is executed. This will delete the conversion ready flag and also read the overflow flag. The measurement data is read and displayed. The interrupt on pin 2 was deactivated after triggering and is switched on again after the values have been output.

**Continous\_Alert\_Controlled.ino**

#include <Wire.h>

#include <INA226\_WE.h>

#define I2C\_ADDRESS 0x40

int interruptPin = 2;

volatile bool event = **false**;

INA226\_WE ina226(I2C\_ADDRESS);

// INA226\_WE ina226 = INA226\_WE(); // Alternative: sets default address 0x40

**void** setup() {

Serial.begin(9600);

Wire.begin();

ina226.init();

/\* Set Number of measurements for shunt and bus voltage which shall be averaged

\* Mode \* \* Number of samples \*

AVERAGE\_1 1 (default)

AVERAGE\_4 4

AVERAGE\_16 16

AVERAGE\_64 64

AVERAGE\_128 128

AVERAGE\_256 256

AVERAGE\_512 512

AVERAGE\_1024 1024

\*/

ina226.setAverage(AVERAGE\_1024);

/\* Set conversion time in microseconds

One set of shunt and bus voltage conversion will take:

number of samples to be averaged x conversion time x 2

\* Mode \* \* conversion time \*

CONV\_TIME\_140 140 µs

CONV\_TIME\_204 204 µs

CONV\_TIME\_332 332 µs

CONV\_TIME\_588 588 µs

CONV\_TIME\_1100 1.1 ms (default)

CONV\_TIME\_2116 2.116 ms

CONV\_TIME\_4156 4.156 ms

CONV\_TIME\_8244 8.244 ms

\*/

ina226.setConversionTime(CONV\_TIME\_8244); // Conversion ready after conversion time x number of averages x 2

/\* Set measure mode

POWER\_DOWN - INA219 switched off

TRIGGERED - measurement on demand

CONTINUOUS - Continuous measurements (default)

\*/

//ina226.setMeasureMode(CONTINUOUS); // choose mode and uncomment for change of default

/\* If the current values delivered by the INA226 differ by a constant factor

from values obtained with calibrated equipment you can define a correction factor.

Correction factor = current delivered from calibrated equipment / current delivered by INA226

\*/

// ina226.setCorrectionFactor(0.95);

Serial.println("INA226 Current Sensor Example Sketch - Continuous\_Alert\_Controlled");

attachInterrupt(digitalPinToInterrupt(interruptPin), alert, FALLING);

ina226.enableConvReadyAlert(); // an interrupt will occur on interrupt pin when conversion is ready

}

**void** loop() {

**if**(event){

ina226.readAndClearFlags(); // reads interrupt and overflow flags and deletes them

displayResults();

attachInterrupt(digitalPinToInterrupt(interruptPin), alert, FALLING);

event = **false**;

}

delay(100);

}

**void** displayResults(){

float shuntVoltage\_mV = 0.0;

float loadVoltage\_V = 0.0;

float busVoltage\_V = 0.0;

float current\_mA = 0.0;

float power\_mW = 0.0;

shuntVoltage\_mV = ina226.getShuntVoltage\_mV();

busVoltage\_V = ina226.getBusVoltage\_V();

current\_mA = ina226.getCurrent\_mA();

power\_mW = ina226.getBusPower();

loadVoltage\_V = busVoltage\_V + (shuntVoltage\_mV/1000);

Serial.print("Shunt Voltage [mV]: "); Serial.println(shuntVoltage\_mV);

Serial.print("Bus Voltage [V]: "); Serial.println(busVoltage\_V);

Serial.print("Load Voltage [V]: "); Serial.println(loadVoltage\_V);

Serial.print("Current[mA]: "); Serial.println(current\_mA);

Serial.print("Bus Power [mW]: "); Serial.println(power\_mW);

**if**(!ina226.overflow){

Serial.println("Values OK - no overflow");

}

**else**{

Serial.println("Overflow! Choose higher current range");

}

Serial.println();

}

**void** alert(){

event = **true**;

detachInterrupt(2);

}

#### Practical application of the Conversion Ready Alert

During the long 17 seconds between measurements, the Arduino or any other microcontroller you use in your project has nothing to do. This consumes valuable electricity in battery-powered projects. So just send your microcontroller to sleep and let it wake up through the interrupt. If you don’t know how to do it, look [here](https://wolles-elektronikkiste.de/en/sleep-modes-and-power-management" \t "_blank) at my post on this subject.

### Example 5: Limit Alert

I would like to introduce you to the Limit Alarm using the sketch Limit\_Alert.ino. The INA226 is in continuous mode. On the Arduino side, an interrupt on pin 2 is set up again.

The function enableAlertLatch() sets the alarm pin so that it is active when an alarm is raised until it is manually readAndClearFlags() set inactive again. Without this setup, the pin would be automatically reset if the next measured value is within the limits.

With setAlertType() you determine the limit and which of the measured values is observed. You can specify a min or max limit for the shunt voltage, bus voltage, or current. For the power you can only set a max limit. Actually, the INA226 has no alarm function for the current, I implemented this via a detour.

And that’s it. When the set limit is exceeded, the alarm pin becomes active, the interrupt is triggered and the measured values are read. Again, you will read the current values. At fast measuring frequency, therefore, you do not necessarily read exactly the measured value that triggered the alarm. If you wanted to, you could set the INA226 into power-down mode when the alarm is triggered. Then you can take your time to read the measured data.

You have to be a bit careful with the readAndClearFlags() function. When you read the flags to evaluate them, you delete them. If the alarm condition still exists, the alarm pin becomes active again immediately. If this happens before the interrupt is reactivated, everything gets confused. The alarm pin would already be low while the interrupt pin was waiting for a falling edge. It could wait for a very long time! Therefore, readAndClearFlags() is called again after the interrupt has been reactivated. Separating the reading and deletion of the flags would be a little easier to control, but this is not implemented in INA226.

**Limit\_Alert.ino**

#include <Wire.h>

#include <INA226\_WE.h>

#define I2C\_ADDRESS 0x40

int interruptPin = 2;

volatile bool event = **false**;

INA226\_WE ina226(I2C\_ADDRESS);

**void** setup() {

Serial.begin(9600);

Wire.begin();

ina226.init();

/\* Set Number of measurements for shunt and bus voltage which shall be averaged

\* Mode \* \* Number of samples \*

AVERAGE\_1 1 (default)

AVERAGE\_4 4

AVERAGE\_16 16

AVERAGE\_64 64

AVERAGE\_128 128

AVERAGE\_256 256

AVERAGE\_512 512

AVERAGE\_1024 1024

\*/

// ina226.setAverage(AVERAGE\_1024);

/\* Set conversion time in microseconds

One set of shunt and bus voltage conversion will take:

number of samples to be averaged x conversion time x 2

\* Mode \* \* conversion time \*

CONV\_TIME\_140 140 µs

CONV\_TIME\_204 204 µs

CONV\_TIME\_332 332 µs

CONV\_TIME\_588 588 µs

CONV\_TIME\_1100 1.1 ms (default)

CONV\_TIME\_2116 2.116 ms

CONV\_TIME\_4156 4.156 ms

CONV\_TIME\_8244 8.244 ms

\*/

// ina226.setConversionTime(CONV\_TIME\_8244);

/\* Set measure mode

POWER\_DOWN - INA219 switched off

TRIGGERED - measurement on demand

CONTINUOUS - Continuous measurements (default)

\*/

//ina226.setMeasureMode(CONTINUOUS); // choose mode and uncomment for change of default

/\* If the current values delivered by the INA226 differ by a constant factor

from values obtained with calibrated equipment you can define a correction factor.

Correction factor = current delivered from calibrated equipment / current delivered by INA226

\*/

// ina226.setCorrectionFactor(0.95);

Serial.println("INA226 Current Sensor Example Sketch - Limit\_Alert");

/\* In the default mode the limit interrupt flag will be deleted after the next measurement within limits.

With enableAltertLatch(), the flag will have to be deleted with readAndClearFlags().

\*/

ina226.enableAlertLatch();

/\* Set the alert type and the limit

\* Mode \* \* Description \* \* limit unit \*

SHUNT\_OVER Shunt Voltage over limit mV

SHUNT\_UNDER Shunt Voltage under limit mV

CURRENT\_OVER Current over limit mA

CURRENT\_UNDER Current under limit mA

BUS\_OVER Bus Voltage over limit V

BUS\_UNDER Bus Voltage under limit V

POWER\_OVER Power over limit mW

\*/

ina226.setAlertType(POWER\_OVER, 230.0);

attachInterrupt(digitalPinToInterrupt(interruptPin), alert, FALLING);

}

**void** loop() {

**if**(event){

ina226.readAndClearFlags(); // reads interrupt and overflow flags and deletes them

displayResults();

attachInterrupt(digitalPinToInterrupt(interruptPin), alert, FALLING);

event = **false**;

ina226.readAndClearFlags();

}

delay(1000);

}

**void** displayResults(){

float shuntVoltage\_mV = 0.0;

float loadVoltage\_V = 0.0;

float busVoltage\_V = 0.0;

float current\_mA = 0.0;

float power\_mW = 0.0;

shuntVoltage\_mV = ina226.getShuntVoltage\_mV();

busVoltage\_V = ina226.getBusVoltage\_V();

current\_mA = ina226.getCurrent\_mA();

power\_mW = ina226.getBusPower();

loadVoltage\_V = busVoltage\_V + (shuntVoltage\_mV/1000);

Serial.print("Shunt Voltage [mV]: "); Serial.println(shuntVoltage\_mV);

Serial.print("Bus Voltage [V]: "); Serial.println(busVoltage\_V);

Serial.print("Load Voltage [V]: "); Serial.println(loadVoltage\_V);

Serial.print("Current[mA]: "); Serial.println(current\_mA);

Serial.print("Bus Power [mW]: "); Serial.println(power\_mW);

**if**(!ina226.overflow){

Serial.println("Values OK - no overflow");

}

**else**{

Serial.println("Overflow! Choose higher current range");

}

Serial.println();

}

**void** alert(){

event = **true**;

detachInterrupt(2);

}

### Example 6: Limit and Conversion Alert

I hope you are still motivated – that is the last example. With the sketch Limit\_And\_Conversion\_Alert.ino I want to show how you can use the limit and the conversion ready alarm side by side. Both alarms are activated as in the previous sketches.

In the event of an alarm, you now want to be able to distinguish which condition triggered the alarm. In case of alarm, read the flags via readAndClearFlags(). This will update the variables limitAlert and convAltert and allow you to query them.

#include <Wire.h>

#include <INA226\_WE.h>

#define I2C\_ADDRESS 0x40

int interruptPin = 2;

volatile bool event = **false**;

INA226\_WE ina226(I2C\_ADDRESS);

**void** setup() {

Serial.begin(9600);

Wire.begin();

ina226.init();

// Conversion will be ready after conversion time x number of averages x 2

ina226.setAverage(AVERAGE\_512);

ina226.setConversionTime(CONV\_TIME\_8244);

// ina226.setCorrectionFactor(0.95);

Serial.println("INA226 Current Sensor Example Sketch - Limit\_And\_Conversion\_Alert");

/\* In the default mode the limit interrupt flag will be deleted after the next measurement within limits.

With enableAltertLatch(), the flag will have to be deleted with readAndClearFlags().

\*/

ina226.enableAlertLatch();

/\* Set the alert type and the limit

\* Mode \* \* Description \* \* limit unit \*

SHUNT\_OVER Shunt Voltage over limit mV

SHUNT\_UNDER Shunt Voltage under limit mV

CURRENT\_OVER Current over limit mA

CURRENT\_UNDER Current under limit mA

BUS\_OVER Bus Voltage over limit V

BUS\_UNDER Bus Voltage under limit V

POWER\_OVER Power over limit mW

\*/

ina226.setAlertType(CURRENT\_UNDER, 45.0);

ina226.enableConvReadyAlert(); // In this example we also enable the conversion ready alert interrupt

attachInterrupt(digitalPinToInterrupt(interruptPin), alert, FALLING);

}

**void** loop() {

**if**(event){

ina226.readAndClearFlags();

displayResults();

attachInterrupt(digitalPinToInterrupt(interruptPin), alert, FALLING);

ina226.readAndClearFlags();

event = **false**;

}

delay(1000);

}

**void** displayResults(){

float shuntVoltage\_mV = 0.0;

float loadVoltage\_V = 0.0;

float busVoltage\_V = 0.0;

float current\_mA = 0.0;

float power\_mW = 0.0;

shuntVoltage\_mV = ina226.getShuntVoltage\_mV();

busVoltage\_V = ina226.getBusVoltage\_V();

current\_mA = ina226.getCurrent\_mA();

power\_mW = ina226.getBusPower();

loadVoltage\_V = busVoltage\_V + (shuntVoltage\_mV/1000);

**if**(ina226.limitAlert){

Serial.println("Limit Alert !!!!");

}

**if**(ina226.convAlert){

Serial.println("Conversion Alert!!!!");

}

Serial.print("Shunt Voltage [mV]: "); Serial.println(shuntVoltage\_mV);

Serial.print("Bus Voltage [V]: "); Serial.println(busVoltage\_V);

Serial.print("Load Voltage [V]: "); Serial.println(loadVoltage\_V);

Serial.print("Current[mA]: "); Serial.println(current\_mA);

Serial.print("Bus Power [mW]: "); Serial.println(power\_mW);

**if**(!ina226.overflow){

Serial.println("Values OK - no overflow");

}

**else**{

Serial.println("Overflow! Choose higher current range");

}

Serial.println();

}

**void** alert(){

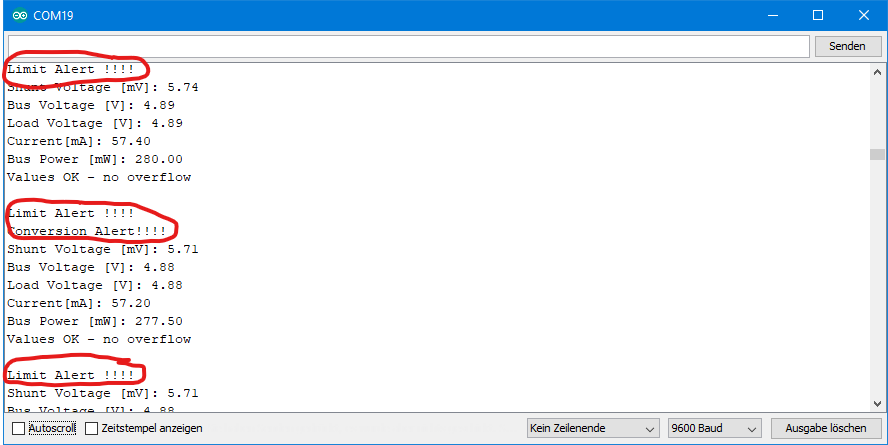
event = **true**;

detachInterrupt(2);

}

#### Output of Limit\_And\_Conversion\_Alert.ino

For the following output, I have permanently exceeded the set limit. Accordingly, the INA226 reports limit violations every second. With the conditions set above, the INA226 also issues a conversion ready alert approximately every 8 seconds:

[](https://wolles-elektronikkiste.de/wp-content/uploads/2020/06/Ausg_Limit_And_Conv.png)Output of the Limit\_And\_Conversion\_Alert.ino sketch

### Beispiel 7: Continuous mit alternative resistor

A kind contributor has added a function to my library that allows you to change the shunt:

* setResistorRange(0.005, 10.0) sets the resistor in ohms and the range in amperes.

<p