



---

# LSM303C 6DoF Hookup Guide

## Introduction

The LSM303C is a 6 degrees of freedom (6DOF) inertial measurement unit (IMU) in a single package. It houses a 3-axis accelerometer, and a 3-axis magnetometer. The range of each sensor is configurable: the accelerometer's scale can be set to  $\pm 2g$ ,  $\pm 4g$ ,  $\pm 6g$ , or  $\pm 8g$ , and the magnetometer has full-scale range of  $\pm 16$  gauss.



*LSM303C Breakout Board*

The LSM303C supports I<sup>2</sup>C and SPI. This tutorial focuses on using this device in I<sup>2</sup>C mode, but will briefly describe how to use SPI.

## Covered In This Tutorial

First we'll introduce you to the breakout board. Then we'll switch over to example code and show you how to interface with the board using an Arduino and our SparkFun LSM303C 6 DOF IMU Breakout Arduino Library.






The tutorial is split into the following sections:

- Breakout Board Overview – This page examines the LSM303C Breakout Board – topics like the pinout, jumpers, and schematic are covered.
- Hardware Assembly – How to assemble the hardware to run some example code.
- Installing the Arduino Library – How to install the Arduino library, and use a simple example sketch to verify that your hookup works.
- Resources & Going Further – Resources for learning and doing more with the LSM303C.

## Required Materials

This tutorial explains how to use the LSM303C Breakout Board with an

Arduino. To follow along, you'll need the following materials:

<b>SparkFun LSM303C Hookup Guide</b> <a href="#">SparkFun Wish List</a>	
	<b>SparkFun 6 Degrees of Freedom Breakout - LSM303C</b> BOB-13303 The LSM303C is a 6 Degrees of Freedom (6DOF) inertial measurem...
	<b>Jumper Wires Standard 7" M/M Pack of 30</b> PRT-11026 If you need to knock up a quick prototype there's nothing like having a...
	<b>Breadboard - Self-Adhesive (White)</b> PRT-12002 This is your tried and true white solderless breadboard. It has 2 power...
	<b>Break Away Headers - Straight</b> PRT-00116 A row of headers - break to fit. 40 pins that can be cut to any size. Us...
	<b>Arduino Pro Mini 328 - 3.3V/8MHz</b> DEV-11114 It's blue! It's thin! It's the Arduino Pro Mini! SparkFun's minimal design...

**The LSM303C is a 2.5V device!** Supplying voltages greater than 4.8V can permanently damage the IC. InvenSense recommends running from **1.9V to 3.6V**. As long as your Arduino has a 3.3V supply output, you shouldn't need any extra level shifting. See our [\[logic level tutorial\] \(tutorials/62\)](#) for more info if you aren't using a 3.3V system.

## Suggested Reading

If you're not familiar with some of the concepts below, we recommend checking out that tutorial before continuing on.

- Pull-up Resistors
- Accelerometer Basics
- Inter-Integrated Circuit Communication (I<sup>2</sup>C)

## Hardware Overview

### The Pinout

The LSM303C 6 DOF Breakout has 10 plated through hole connections.



*Top View of LSM303C Breakout Board*

The following table summarizes all of the plated through hole connections

on the breakout board:

Pin Label	Pin Function	Notes
<b>GND</b>	Ground reference	+0V
<b>VDD_IO</b>	Power supply for I/O pins	1.71V up to VDD + 0.1V
<b>SDA/ SDI/ SDO</b>	I <sup>2</sup> C serial data SPI serial data input 3-wire interface serial data output	ST calls the second serial interface SPI, but it's really a half-duplex variant that uses the same pin for MISO and for MOSI. Note that all 3 data signals are the same pin.
<b>SCL/ SCLK</b>	I <sup>2</sup> C serial clock SPI serial port clock	100 or 400 kHz I <sup>2</sup> C Up to 10 MHz SPI
<b>INT_XL</b>	Accelerometer interrupt signal	The functions, the threshold and the timing of this interrupt are configurable.
<b>DRDY</b>	Data ready	Configurable output to indicate when accelerometer or magnetometer data is ready.
<b>CS_XL</b>	Accelerometer: SPI enable I <sup>2</sup> C/SPI mode selection	1: SPI idle mode / I <sup>2</sup> C communication enabled; 0: SPI communication mode / I <sup>2</sup> C disabled
<b>VDD</b>	Power supply	1.9V to 3.6V
<b>CS_MAG</b>	Magnetometer: SPI enable I <sup>2</sup> C/SPI mode selection	1: SPI idle mode / I <sup>2</sup> C communication enabled; 0: SPI communication mode / I <sup>2</sup> C disabled
<b>INT_MAG</b>	Magnetometer interrupt signal	The functions, the threshold and the timing of this interrupt are configurable.

## Power Supply

The LSM303C breakout has three power supply plated thru-hole connections: a 0V reference (**GND**), a core supply (**VDD**), and an IO supply (**VDD\_IO**). The core of the IC can be powered from **1.9-3.6V**. The IO must be given a potential of at least 1.71V up to the core supply voltage plus 0.1V. This dual supply setup eliminates the need for external voltage level translation. A 3.3V rail can power most of the device while still being able to communicate with a 1.8V processor without drawing all of its power from that lower voltage rail.

## Communication

The LSM303C communicates over I<sup>2</sup>C or 'SPI' using the same plated thru-hole connections. The implementation of 'SPI' on the LSM303C isn't standard; it's a half-duplex variant. Standard SPI has a MOSI and a MISO signal. Both of these are found on the single SDA/SDI/SDO connection. The more common Arduino variants don't have hardware that directly supports this, so we are bit banging in our library. Your system may be compatible, so we didn't add external components to get the hardware to

work with the Atmel SPI hardware. This connection is also used as the SDA connection for I<sup>2</sup>C. Testing showed that the implementation of this IO acts like an open-drain like is common with I<sup>2</sup>C. This means that a pull-up resistor is needed for both SPI and I<sup>2</sup>C. The breakout includes this pull-up. Both communication modes share the same clock line (SCL/SCLK).

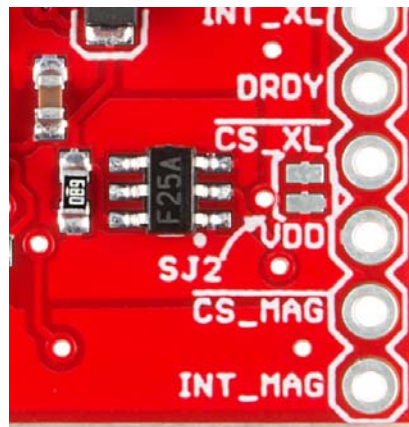
The LSM303C is implemented as two separate cores on the same die. The accelerometer and magnetometer have their own chip select lines. In I<sup>2</sup>C mode, they have their own unique addresses. The accelerometer is at **0x1D**, and the magnetometer is at **0x1E**.

## Interrupts

There are a variety of interrupts on the LSM303C. The system can be configured to generate an interrupt signal for free-fall, motion detection and magnetic field detection. The actual function of the two interrupt pins (INT\_XL & INT\_MAG) are highly configurable through either the I<sup>2</sup>C or SPI interfaces. They can be active high or low, latching or non-latching, etc. This advanced topic won't be covered in this hookup guide. Please reference the datasheet for more information.

## The Jumper

In many cases, especially Arduino related, you won't have multiple lower voltage rails. For these cases we've included SJ2. Your board comes with this jumper closed with a trace by default. This connects VDD\_IO and VDD.



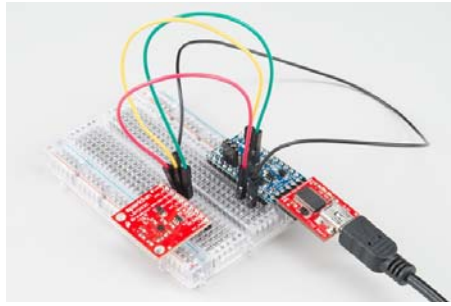
*Closeup of voltage jumper*

The intention of this jumper is to allow the end user to power use the board and begin developing right out of the box. To disable any of these jumpers, whip out your handy hobby knife, and carefully cut the small traces between the two pads. You may then connect VDD\_IO to whatever power rail you desire.

## Hardware Assembly

### I<sup>2</sup>C Example

The basic use case for the LSM303C requires 4 connections to the  $\mu$ Controller or  $\mu$ Processor; power, ground, I<sup>2</sup>C clock and data. The following images shows how we used a SparkFun FTDI Basic Breakout, and an 3.3V Arduino Pro Mini to power and interface to a LSM303C 6 DOF Breakout board.



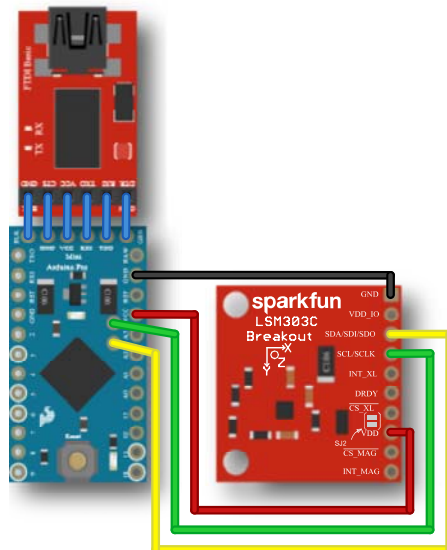
*An LSM303C wired up to an Arduino Pro Mini for the MinimalistExample (IIC)*

Make connections to the breakout anyway that makes you happy. The board in the above photo has a straight header soldered to it. We could have used a right angle header, or wire, etc. **Please note that different mounting orientations will alter the orientation of the axes.** Make sure your code matches the physical orientation for your projects.

For this demo, we made the following connections:

Arduino Pro Mini	LSM303C Breakout	Notes
VCC	VDD	+3.3V
GND	GND	+0V
SDA	SDA/SDI/SDO	Serial data @ +3.3V CMOS logic
SCL	SCL/SCLK	Serial clock @ +3.3V CMOS logic

The whole system in our testing was powered via USB through the FTDI basic.



*Electrical connections for demo*

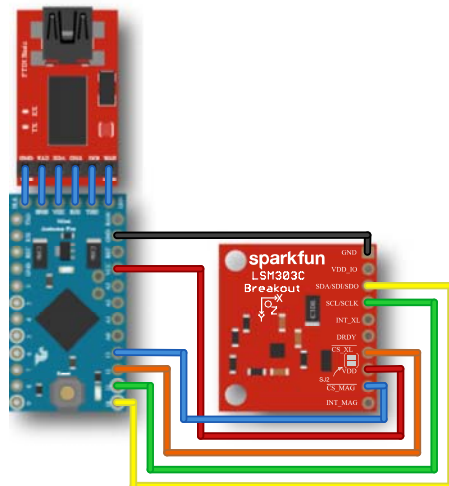
## SPI Example

Four hardware changes need to be made to interface the sensor using SPI. Move the SDA/SDI/SDO connection from SDA on the Arduino Pro Mini to digital pin 10, move the SCK/SCLK connection from SCL on the Arduino Pro Mini to digital pin 11, and add the two chip select lines.

Arduino Pro Mini	LSM303C Breakout	Notes
VCC	VDD	+3.3V
GND	GND	+0V
Digital 10	SDA/SDI/SDO	Serial data @ +3.3V CMOS logic
Digital 11	SCL/SCLK	Serial clock @ +3.3V CMOS logic
Digital 12	CS_XL	Accelerometer chip select @ +3.3V CMOS logic
Digital 13	CS_MAG	Magnetometer chip select @ +3.3V CMOS logic



*Example of a Pro Mini wired up for SPI*



*Connecting for SPI interface*

## Installing the Arduino Library

## Download and Install the Library

Visit the GitHub repository to download the most recent version of the libraries, or click the link below:

[DOWNLOAD THE LSM303C ARDUINO LIBRARIES](#)

For help installing the library, check out our Installing an Arduino Library tutorial. You might find it easier to use the Arduino IDE library manager if you are running a modern release, but feel free to use any method in the tutorial.

The example Arduino code allows you to do things like read the magnetometer in all 3 axis, read the accelerometer in all 3 axis, and read the temperature of the die in Fahrenheit and Celsius.

## Running the Minimalist Example

Now, you can now run the example sketches. Open File ⇒ Examples ⇒ SparkFun LSM303C 6 DOF IMU Breakout ⇒ MinimalistExample. This sketch is as simple as possible other than a little error checking.

The setup function configures the Arduino's serial port to 115200 baud and configures the LSM303C to some reasonable defaults.

```
LSM303C myIMU;

void setup()
{
  Serial.begin(115200);
  if (myIMU.begin() != IMU_SUCCESS)
  {
    Serial.println("Failed setup.");
    while(1);
  }
}
```

The loop function sequentially prints out the x, y, and z vales measured by the accelerometer, then the gyroscope, and then the magnetometer. This is followed up by the temperature of the die in degrees Celsius and Fahrenheit. All values are rounded to 4 digits past the decimal point.

```
void loop()
{
  //Get all parameters
  Serial.print("\nAccelerometer:\n");
  Serial.print(" X = ");
  Serial.println(myIMU.readAccelX(), 4);
  Serial.print(" Y = ");
  Serial.println(myIMU.readAccelY(), 4);
  Serial.print(" Z = ");
  Serial.println(myIMU.readAccelZ(), 4);
}
```

Here is some sample output:

```

Accelerometer:
X = 56.7017
Y = 42.7856
Z = 946.2891

Gyroscope:
X = nan
Y = nan
Z = nan

Magnetometer:
X = -0.2051
Y = 0.0527
Z = 0.0742

Thermometer:
Degrees C = 24.5000
Degrees F = 76.1000

```

You may have noticed that gyro data is printed despite there not being a gyro on this board. If a value is not available, the library functions will return nan (not a number). In this case the LSM303C doesn't have a gyroscope, but it still returns a value for a consistent IMU interface. Any IMU library that implements the SparkFunIMU abstract class can be swapped out without having to change the code that uses the library. If at some point in the future it is determined that a project needs more or less degrees of freedom (and is coded with error checking) the code change is trivial. Change `#include "SparkFunLSM303C.h"` to `#include "SparkFun<some other sensor>.h"`, and `LSM303C myIMU;` to `<some other sensor> myIMU`. If for whatever reason the sensor doesn't have valid data ready for a sensor that does exist, it will also return nan. This indicates that the value is undefined.

## Running the Configure Example

By default, the easy to configure example is configured exactly the same as the minimalist example. Here is the code that differentiates the two examples, the setup:

```

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

  if (myIMU.begin(
    ///// Interface mode options
    //MODE_SPI,
    MODE_I2C,

    ///// Magnetometer output data rate options
    //MAG_DO_0_625_Hz,
    //MAG_DO_1_25_Hz,

```

This setup function exposes all of the configuration options necessary to read the magnetometer and accelerometer. See the datasheet for more advanced configuration. To change a configuration option, uncomment the desired option in that section, and remove or comment out all other options in that section. For example, if you wanted to use the SPI interface, you would change that section to look like the following.



```

///// Interface mode options
MODE_SPI,
//MODE_I2C,

```

After uploading the sketch, open the serial monitor or your favorite terminal emulator and you should start seeing something similar to the following repeating over and over once per second.

```

Accelerometer:
X = 64.5752
Y = 31.4941
Z = 943.1152

Magnetometer:
X = -0.2085
Y = 0.0425
Z = 0.0972

Thermometer:
Degrees C = 25.3750
Degrees F = 77.6750

```

In the setup that this data capture came from, the breakout board was sitting roughly flat on a desk. This orients the z-axis parallel to the earth's gravitational field of 1,000 mg. This is seen by the z-axis value of around 943. If the LSM303C were in free fall, the z-axis component would be 0 g, because it wouldn't be accelerating. Since the sensor isn't in free fall, it is measuring an effective acceleration of 1 g in the positive z direction up out of the table.

## More Library Details

### Common IMU Interface

This library does the C++ 'equivalent' of implementing a common interface or template. It does this by implementing the pure virtual methods of the **SparkFunIMU** class. This strays from the pure definition of an abstract because not all of the methods are purely virtual. We've strayed from this to give unimplemented methods a default behavior. *In less technical terms*, we've provided a common set of basic functions that any IMU should have.

```

virtual float readGyroX() { return NAN; }
virtual float readGyroY() { return NAN; }
virtual float readGyroZ() { return NAN; }
virtual float readAccelX() { return NAN; }
virtual float readAccelY() { return NAN; }
virtual float readAccelZ() { return NAN; }
virtual float readMagX() { return NAN; }
virtual float readMagY() { return NAN; }
virtual float readMagZ() { return NAN; }
virtual float readTempC() { return NAN; }
virtual float readTempF() { return NAN; }

```

The LSM303C provides useful definitions for all of these methods except for the ones that read the gyroscope, since the LSM303C doesn't have a gyroscope. If you were to call `readGyroX()`, the default definition would be used, and it would return **Not A Number** (NAN). This is useful because, if

you write your code to use these functions and later decide to use a different IMU that also implements this interface, you only have to change a few words, and all of the code will work with the new sensor.

## LSM303C Types

This library is written a little to the computer science object-oriented, encapsulation-type safety side, and a little less to the code size or speed optimized side. To provide type safety and improve readability, **LSM303CTypes.h** was written. In this header file, many types are defined, all registers are defined to types with descriptive names. As are all of the values you might want to write to the registers. Here is a simple example:

```
typedef enum
{
  ACC_I2C_ADDR = 0x1D,
  MAG_I2C_ADDR = 0x1E
} I2C_ADDR_t
```

The first *keyword* there, `typedef`, is used in C and C++ to define more complex types out of existing types. In this case a type named `I2C_ADDR_t` is defined to be an enumeration with the `enum` keyword. An enumeration is a list of explicitly named integral type constants. They are guaranteed to be a variable large enough to hold an `int` type, but what they really are depends on the compiler. For **avr-gcc** there are compiler switches that can change the actual value used. In this case there are two valid values for a variable of type `I2C_ADDR_t`; `ACC_I2C_ADDR` and `MAG_I2C_ADDR`. `ACC_I2C_ADDR` is short for "accelerometer (Inter-Integrated Circuit (I<sup>2</sup>C) address". Arduino will interpret that value to be `0x1D`.

Consider the following two function prototypes:

```
uint8_t I2C_ByteWrite(I2C_ADDR_t, uint8_t, uint8_t);
uint8_t I2C_ByteWrite(int, uint8_t, uint8_t);
```

Both versions of that function are capable of accepting the values `0x1D` and `0x1E`. The main difference is that the first prototype is type safe. The first parameter has to be of type `I2C_ADDR_t`. This type only has two valid values, `ACC_I2C_ADDR` and `MAG_I2C_ADDR`. If you try and pass any other value without explicitly casting it your code won't compile. You cannot make a mistake that can be loaded onto your Arduino. The `avr-gcc` toolchain that comes with the Arduino IDE will not let you make that mistake.

The first parameter of the second function prototype can be any `int`, including the values `0x1D` and `0x1E`, but not limited to those valid values. Sure your code could handle unexpected values, but what should it do about it? Strobe out an error message in Morse code on an LED? Lock up? This type unsafe code can allow runtime errors to get onto your microcontroller and cause strange bugs. The configure example was designed to show all of the common options, so referencing this header isn't typically needed. The extra complexity is 'hidden' away where you only see it if you go looking for it.

## Debug Macros

Also in the library is a header file named **DebugMacros.h**. As the name suggests this file contains the definitions of 4 macro functions used for debugging. This is a very simple tool hacked together for the use in developing this library, but is useful none the less. They don't follow the GNU coding style (case), so they blend in more like standard functions in an attempt to hide the complexity from the beginner programmer.

Prototype	Description
<code>debug_print</code> (msg, ...)	Prints a labeled debug message to the serial monitor. This macro function prepends the name of function & ':' to what <code>Serial.print</code> would do. E.g. <code>loop::Debug message</code>
<code>debug_prints</code> (msg, ...)	Very similar to the above function except it's shorter. No function label here. Basically the same function as <code>Serial.print()</code> .
<code>debug_println</code> (msg, ...)	Very similar to the first macro function except it appends a newline character.
<code>debug_printlns</code> (msg, ...)	Basically <code>Serial.println()</code> . No function label here.

These macro functions have a few advantages over the built in serial printing functions. The first is that all you have to do is change a single character to turn all of your debug statements on or off. They will be completely removed from the compiled code if turned off. This is done by defining `DEBUG` to be 1 (or non-zero) to enable the debug output. If `DEBUG` is defined to be 0, all of the `debug_print<options>` statements will be removed from the code by the preprocessor before compilation. The place to define the `DEBUG` macro is at the top of **SparkFunLSM303C.cpp**.

Another useful trick they can be used for is generating something similar to a stack trace. To do this, simply add `debug_print(EMPTY);` to each function. Pretend that there are a bunch of functions defined. Here is a stripped down code example:

```
void loop()
{
  debug_print(EMPTY); // This is kind of unnecessary
  level_1_func();
}

void level_1_func(void)
{
  debug_print(EMPTY);
  level_2_func();
}

void level_2_func(void)
{
  debug_println("Example error message");
}
```

Running the sketch containing this code would produce the following output:

```
loop::level_1_func::level_2_func::Example error message
```

Along with the message saying what went wrong the code provides an in order list of all of the functions that called it. This is helpful for tracing back where the error occurred. It tells all of the recent functions involved. This isn't as useful as a real stack trace, but it's worth the 20 lines of code.

## Resources & Going Further

Hopefully that info dump was enough to get you rolling with the LSM303C.

If you need any more information, here are some more resources:

- [LSM303C Product GitHub Repository](#) – Your revision-controlled source for all things LSM303C. Here you'll find our most up-to-date hardware layouts and code.
- [LSM303C Datasheet](#) – This datasheet covers everything from the hardware and pinout of the IC, to the register mapping of the accelerometer, and magnetometer.
- [LSM303C Breakout Schematic](#)
- [LSM303C Breakout EAGLE Files](#)

## Going Further

Now that you've got the LSM303C up-and-running, what project are you going to incorporate motion-sensing into? Need a little inspiration? Check out some of these tutorials!

- [Dungeons and Dragons Dice Gauntlet](#) – This project uses an accelerometer to sense a “rolling the dice” motion. You could swap in the LSM303C to add more functionality – like compass-based damage multipliers!
- [Are You Okay? Widget](#) – Use an Electric Imp and accelerometer to create an “Are You OK” widget. A cozy piece of technology your friend or loved one can nudge to let you know they're OK from half-a-world away.
- [Leap Motion Teardown](#) – An IMU sensor is cool, but image-based motion sensing is the future. Check out this teardown of the miniature-Kinect-like Leap Motion!
- [Pushing Data to Data.SparkFun.com](#) – Need an online place to store your IMU data? Check out [data.sparkfun.com](#)! This tutorial demonstrates how to use a handful of Arduino shields to post your data online.