THE DATASHEET OF

SEN-12916

HMC6343 3-axis Compass Hookup Guide

Introduction

The HMC6343 is a fully-integrated, high-end, digital compass module that can compute and give the user a heading direction that's accurate within a couple degrees! It is tilt compensated and is calibrated to handle magnetic distortions. The IC combines 3-axis magneto-resistive sensors and 3-axis MEMS accelerometers, analog and digital support circuits, a microprocessor and algorithms in firmware required for heading computation.

The breakout board allows for easy use of the HMC6343. All that is required is power (3.3V @ 4.5mA in run mode) and I2C connections to a microcontroller so that the module can receive commands and send data back to the user.

Covered in this Tutorial

In this tutorial, we will help you learn how to use this electronic compass so you can quickly and painlessly integrate them into your project[s]. Here is what we'll cover:

- Hardware Overview – An overview of the HMC6343 breakout board as well as some details about the IC.
- Example Hookup – How to connect the compass to the ubiquitous Arduino so we can start writing code to work with it.
- Example Code – We've written example sketches and a library that demonstrate how to collect sensor data as well as make sense of it.

Required Materials

- HMC6343 Breakout Board
- Arduino Uno or any Arduino Board– We will use the Uno as the example, however you should be able to use any Arduino board you have handy including the RedBoard, Pro, Mega, etc.
- Bi-directional Logic Level Converter

You may also need a breadboard, jumper wires, and straight male headers to follow the example setup, if you don't already have these or another way of connecting the Arduino to the breakout board.

Suggested Reading

Before continuing on with this tutorial, we recommend you be somewhat familiar with the concepts in these tutorials:

- Installing an Arduino Library
- Inter-IC Communication (I^2C)
- Logic Levels
- Bi-Directional Level Shifter Hookup Guide

Hardware Overview

The Breakout Board for the HMC6343 breaks out the all the pins you'll need to send commands and collect data from the electronic compass.

As you can see, the breakout has only four pins broken out. Two are for power, 3.3V and GND. Two are for I2C communication, SCL (clock) and SDA (data).

Voltage Supply Requirements

The big alert here is that the HMC6343 has an **input voltage range of 2.7V - 3.6V** – that range applies to both the power supply and the I2C lines. You can use a 5V or 3.3V micro with this sensor as long as you power the board with 3.3V and use a logic level converter for the I2C lines if you're using a 5V micro like the Arduino Uno.

Fortunately, you don't need a lot of power to make the HMC6343 work. In the normal operating mode, the IC typically draws about 4.5mA. It draws even less current in standby (1mA) or sleep (10µA) modes.

Extra Hardware Notes

The HMC6343 can measure and compute a heading direction every 200ms (5Hz). Being able to easily acquire a tilt compensated heading direction accurate within a couple degrees without having your microprocessor spend a lot of time on computation is the highlight of this IC. However, you can read raw magnetometer and accelerometer values if needed.

The HMC6343 has two mounting holes to allow for a secure physical connection to your project. You certainly don't want the board moving/twisting around independently of your system if you're depending on it to give you a sense of direction.

Also, there is a solder jumper on the top of the board that is closed by default so that the I2C lines have 10KΩ pull-up resistors. If you want to use a different values for your I2C pull-ups, you can disconnect these default resistors by removing the solder from the jumper.

Example Hookup

Soldering

Before you can plug your compass breakout board into a breadboard and connect it to anything, you'll need to solder connectors or wires to the breakout pins. What you solder to the board depends on how you're going to use it.

If you're going to use the breakout board in a breadboard or similar 0.1"-spaced perfboard, we recommend soldering straight male headers into the pins (there are also long headers if you need).

If you're going to mount the breakout into a tight enclosure, you may want to solder wires (stranded or solid-core) directly into the pins.

Simple Hookup

This example will use an Arduino Uno to collect and interpret the sensor data from the HMC6343. Since the sensor uses I2C communication, all we need are two connections, one for a clock (SCL) and one for data (SDA) in addition to two for power (3.3V and GND).

We simply have to supply the accelerometer with power (3.3V and GND), then hookup the I2C connections. The HMC6343 IC has a max rating of 3.6V on any pin so in order to talk to the HMC6343 with the Arduino Uno or any other 5V microcontroller, we will need a bi-directional logic level converter for the I2C lines.

The following tables shows the connections made for using the HMC6343 breakout with an Arduino Uno and logic level converter.

HMC6343 Breakout	Logic Level Converter	Arduino Uno
3.3V	LV	3.3V
GND	GND	GND
N/A	HV	5V
SDA	<--- LV1 HV1 --->	SDA(A4)
SCL	<--- LV2 HV2 --->	SCL(A5)

Set up this way, I2C communication will be 3.3V on the breakout board side and 5V on the Arduino Uno side. If this is your first time using a logic level converter, or the above table seems confusing, check out our tutorial on logic levels and the hookup guide for the logic level converter. That should clear up any confusion with your setup before proceeding to the code examples.

Example Code

Now that your electronic compass breakout is electrically connected to your Arduino, it's time to dive into the code. The full library (SFE_HMC6343_Library) and example sketches can be found in the github repository. If you need help with GitUb, visit our tutorial. The same goes for installing an Arduino library.

Once the library is installed, open the example labeled HMC6343_basics.ino. This sketch will teach you how to get a heading direction, pitch, and roll angles as well as accelerometer measurements.

The first lines of this example sketch include the necessary libraries, one for I2C and the other so we can declare and interact with the compass object.

```
// Libraries for I2C and the HMC6343 sensor
#include <Wire.h>
#include "SFE_HMC6343.h"

SFE_HMC6343 compass; // Declare the sensor object
```

The variable `compass` will be used for all interactions with the sensor itself.

Next we use the `setup()` function in initialize serial communication and the HMC6343. If the HMC6343 is not detected, an error message will be reported to the Serial Monitor. We also wait half a second before we try to initialize the HMC6343 since it takes that long to be responsive after being powered on.

```
void setup()
{
  // Start serial communication at 115200 baud
  Serial.begin(115200);

  // Give the HMC6343 a half second to wake up
  delay(500);

  // Initialize the HMC6343 and verify its physical presence
  if (!compass.init())
  {
    Serial.println("Sensor Initialization Failed\n\r"); // Rep
ort failure, is the sensor wiring correct?
  }
}
```

In the `loop()` function, we call two functions to read the sensor data and two more to print the data to the Serial Monitor in an easily readable format. `compass.readHeading()` is a library function that commands the sensor to collect heading direction, pitch angle, and roll angle and then store these values. We will see how to use and print these values in the `printHeadingData()` function. `compass.readAccel()` gets a reading from the IC's 3-axis accelerometer values and these are used and printed in `printAccelData()` . Here is the loop in full:

```
void loop()
{
  // Read, calculate, and print the heading, pitch, and roll f
rom the sensor
  compass.readHeading();
  printHeadingData();

  // Read, calculate, and print the acceleration on the x, y,
and z axis of the sensor
  compass.readAccel();
  printAccelData();

  // Wait for two seconds
  delay(2000); // Minimum delay of 200ms (HMC6343 has 5Hz sens
or reads/calculations)
}
```

After calling `compass.readHeading()`, the variables `compass.heading`, `compass.pitch`, and `compass.roll` carry the heading, pitch, and roll values given by the sensor in tenths of degrees. In the `printHeadingData()` function, we print these raw values and also print the value converted to degrees. Let's look at how we do this using the heading direction as an example:

```
Serial.print("Heading: ");
Serial.print(compass.heading); Serial.print("  "); // Print ra
w heading value
Serial.print((float) compass.heading/10.0);Serial.write(176);S
erial.println(); // Print heading in degrees
```

In the first line, we print the label. In the second line, we print the actual raw value returned by the sensor, `compass.heading` and a couple spaces. In the third line, we scale the heading direction by dividing it by 10 so that we can print it in degrees. `Serial.write(176)` draws a degree symbol and then we print a new line.

Within `printHeadingData()`, the same process is followed for pitch and roll. When running the program, the Serial Monitor's output of this function will look like the following:

```
Heading Data (Raw value, in degrees):
Heading: 3249  324.90°
Pitch:   28    2.80°
Roll:    24    2.40°
```

If you have the breakout board parallel to the ground, pitch and roll will be about 0 degrees and you will see the heading angle change as you slowly rotate the board. Try angling the board relative to the X and Y axis and see how that affects your pitch and roll angles.

Printing accelerometer values is very similar. Under the `printAccelData()` function, we print the acceleration measured on the x axis as follows:

```
Serial.print("X: ");
Serial.print(compass.accelX); Serial.print("  "); // Print ra
w acceleration measurement on x axis
Serial.print((float) compass.accelX/1024.0);Serial.println
("g"); // Print x axis acceleration measurement in g forces
```

Again, in the first line we print the label. On the second line, we print the raw acceleration measurement felt on the x axis using the variable `compass.accelX`. Acceleration measurements for y and z are accessed with the variables `compass.accelY` and `compass.accelZ` respectively. On

the third line above, we scale the raw value by dividing it by 1024 to get the accleration in g forces. We print that scaled value as well as the unit symbol, g.

Here an example of what you'll see in the Serial Monitor after calling the `printAccelData()` function:

```
Accelerometer Data (Raw value, in g forces):
X: -52    -0.05g
Y: -44    -0.04g
Z: -1047  -1.02g
```

At the end of `loop()` there is a delay of two seconds before new sensor data is read and printed. The fastest you can consecutively acquire new sensor readings is 5Hz or every 200ms.

Advanced Features

All the additional features of the HMC6343 device library are covered in the `HMC6343_advanced` sketch. However, I'll overview some of the more appealing features here.

If you want to save power, you'll want to be able to use the sleep or standby modes. The IC in standby draws 1.0mA as opposed to 4.5mA under normal run mode operation. Under sleep mode, the IC only uses 10µA however it takes the device longer to wake up after entering that low power state. Here are examples of how the two modes can be entered and exited:

```
// Enter and exit standby mode (4.5mA draw in run mode, 1.0mA
in standby)
// HMC6343 requires 1 ms before it can receive commands after
switching modes
compass.enterStandby();
delay(1);
compass.exitStandby(); // Exit standby, enter run mode (defaul
t)
delay(1);

// Enter and exit sleep mode (4.5mA draw in run mode, 10uA in
sleep)
// HMC6343 requires 1ms after entering sleep and 20ms after ex
iting before it's able to receive new commmands
compass.enterSleep();
delay(1);
compass.exitSleep();
delay(20);
```

If you don't want to mount the HMC6343 breakout board parallel to the ground, then you'll need to know how to set the new orientation of the board in order to get correct heading, pitch, and roll values. Here is a list of the three possible orientations and the command to configure the HMC6343 to use a new orientation:

```
// Possible orientations:
// LEVEL      X = forward, +Z = up (default)
// SIDEWAYS   X = forward, +Y = up
// FLATFRONT  Z = forward, -X = up
compass.setOrientation(LEVEL);
delay(1);  // 1 ms before sensor can receive commands after se
tting orientation
```

Those are the most important additional features. However, the advanced example covers reading/printing raw 3-axis magnetometer values and raw 'tilt' values (pitch, roll, and temperature). It also covers reading the primary

status register (OPMode1) as well as reading and writing to the EEPROM registers of the sensor. There is also a command to reset the processor of the HMC6343 as well as a way to enter and exit user calibration mode.

While the library provides a convenient way to do all these things (every HMC6343 command in the datasheet is used by the library), if you want to know the details of each EEPROM register or how to properly calibrate the sensor, consulting the datasheet is a must.

My hope is that the library allows you to read sensor data and interact with the HMC6343 as easily as possible and allows you to quickly integrate the breakout board into your project.

Resources & Going Further

Thanks for reading! By now you've become familiar with both the hardware and software to use the HMC6343, the high end electronic compass. We're excited to see what you build with this. The following resources may be helpful for you when building your related projects:

- HMC6343 Datasheet
- HMC6343 Github Repository
- Using Github
- You could make a robot like the HUB-ee Buggy or the RedBot, and give it a sense of direction with the HMC6343.
- You could make a multitude of autonomous vehicles, and enter it in the SparkFun Autonomous Vehicle Competition (AVC).

Looking for pricing, stock, or lifecycle information?

Click below to explore more details on WIN SOURCE:

- View SEN-12916 on WIN SOURCE
- SparkFun Electronics Information

Optimize Your Supply Chain with WIN SOURCE Solutions

- Global Sourcing Solution
- Obsolete Management
- Cost Control Management
- Shortage Management
- Alternative Solution
- Excess Inventory Management