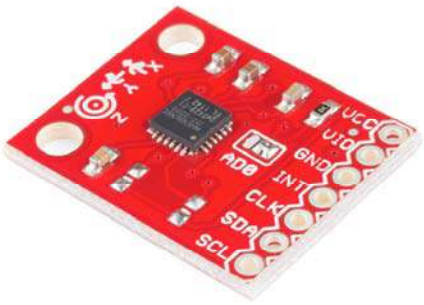
ITG-3200 Hookup Guide

Introduction

This is a breakout board for InvenSense's ITG-3200, a groundbreaking triple-axis, digital output gyroscope. The ITG-3200 features three 16-bit analog-to-digital converters (ADCs) for digitizing the gyro outputs, a user-selectable internal low-pass filter bandwidth, and a Fast-Mode I^2C (400kHz) interface. Additional features include an embedded temperature sensor and a 2% accurate internal oscillator.



This tutorial will help you get started using the ITG-3200 in your next project. We will cover the hardware, discuss the code briefly, then show you how to hook it up to a microcontroller.

Suggested Reading

This tutorial builds on some basic concepts. If you are unfamiliar with any of the topics below, go ahead and check them out. We'll be right here waiting.

- What is Arduino?
- Gyroscope Basics
- I^2C Communication
- Serial Terminal Basics
- How to Solder
- Breadboard Basics

Hardware Overview

Power

The ITG-3200 can be powered at anywhere between **2.1 and 3.6V**. For power supply flexibility, the ITG-3200 has a separate VLOGIC reference pin (labeled **VIO**), in addition to its analog supply pin (**VDD**), which sets the

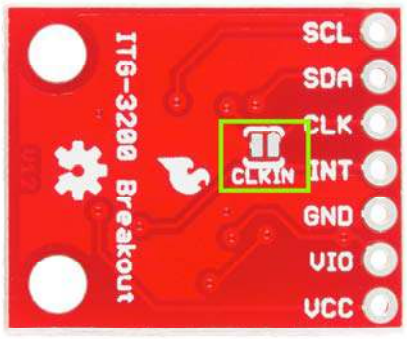
logic levels of its serial interface. The VLOGIC voltage may be anywhere from 1.71V min to VDD max. For general use, VLOGIC can be tied to VCC. The normal operating current of the sensor is just 6.5mA.

Communication

Communication with the ITG-3200 is achieved over a two-wire (I^2C) interface. The sensor also features a interrupt output and an optional clock input.

Clock Source Jumper

In the next picture, you can see a small jumper next to the pin labeled 'CLK.' The ITG-3200 has a feature that allows you to connect an external clock. Unless you plan to use an external clock, you need to **'close'** this jumper by connecting the two pads with solder. If you're following this tutorial and using the provided example code, go ahead and close the jumper.



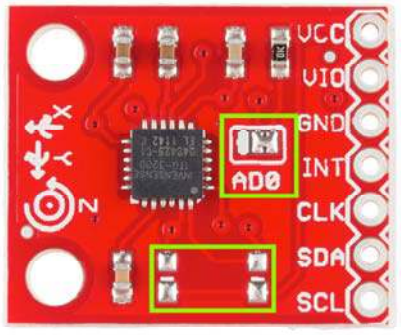
Make sure you close this jumper with solder if you're NOT using an external clock source.

I^2C Address Jumper

A jumper on the top of the board allows you to easily select the I^2C address, by pulling the AD0 pin to either VCC or GND; the board is shipped with this jumper tied to VCC.

I^2C Pull-up Resistors

Note that there are two unpopulated pull-up resistors on the I^2C lines. These can be added later by the user if desired.



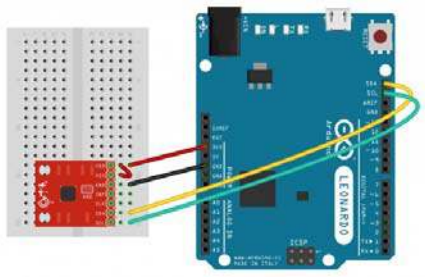
Here the I^2C address jumper (top) and the unpopulated I^2C pull-up resistors (bottom) are highlighted.

Hooking it Up

There are only two considerations for assembling the ITG-3200 breakout: what to do for the headers, and whether or not you're going to use an external clock source for the ITG-3200.

For the headers you have a couple options, you can solder in male or female 0.1" headers, or you can just solder wires directly to the holes in the breakout board. In this example, male headers are soldered to the breakout board to make it easy to attach to a breadboard. Then, we'll be hooking the ITG-3200 up to an Arduino Leonardo using some male-to-male jumper wires.

Second, since we will not be using an external clock source in this setup, make sure the CLKIN jumper on the bottom of the ITG-3200 is closed with a blob of solder.



Here's everything all hooked up. Make sure you add the small jumper from VDD to VIO to ensure they are both connected to 3.3V.

The SDA and SCL pins should be present on most Arduinos. Older, pre-rev3 Arduinos might not have SCL and SDA pins. In that case, connect SDA to **A4** and SCL to **A5**.

The ITG3200 sensor is a 3.3V device. This means that the sensor should be powered by 3.3V and the communication signals should be between 0V and 3.3V. The Arduino Leonardo (and other similar boards) are 5V devices. Even though we power the board with the 3.3V output from the Arduino, the communication signals are still going to be 5V. Technically this should be avoided as it can cause damage to the sensor in the long run. When implementing this gyro in a final project, it's in your best interest to use something like a Logic Level Converter to change the voltages of the communication signals. You could also use an Arduino Pro (3.3V/8 MHz). However, for the purposes of testing out your gyro, using a 5V device should work fine.

That's all there is to it! Now, let's look at some code to get this gyro up and running.

Firmware

We're finally ready to start looking at the firmware. We've written an example Arduino sketch to help you get started. You can download firmware from the ITG-3200 GitHub page.

The sample sketch reads the gyroscope data for the X, Y, and Z axes and prints it to the serial port. This is raw gyroscope data, and it has not been converted to degrees per second yet. Bigger numbers mean the device is rotating faster. Positive numbers indicate one direction of rotation while negative numbers indicate the opposite rotation direction. Since this is a triple-axis gyroscope, we can measure the rotational rate of the board no matter which way the board is rotating. Rotation is usually measured in degrees per second. If the board spins around an axis exactly one time in a second, the gyroscope would measure 360 degrees per second.

Now, let's break up the code in to sections to go over what's happening a little more in depth.

```
//The Wire library is used for I2C communication
#include <Wire.h>

//This is a list of registers in the ITG-3200. Registers are p
arameters that determine how the sensor will behave, or they c
an hold data that represent the
//sensors current status.
//To learn more about the registers on the ITG-3200, download
and read the datasheet.
char WHO_AM_I = 0x00;
char SMPLRT_DIV= 0x15;
char DLPF_FS = 0x16;
char GYRO_XOUT_H = 0x1D;
char GYRO_XOUT_L = 0x1E;
char GYRO_YOUT_H = 0x1F;
char GYRO_YOUT_L = 0x20;
char GYRO_ZOUT_H = 0x21;
char GYRO_ZOUT_L = 0x22;

//This is a list of settings that can be loaded into the regis
ters.
//DLPF, Full Scale Register Bits
//FS_SEL must be set to 3 for proper operation
//Set DLPF_CFG to 3 for 1kHz Fint and 42 Hz Low Pass Filter
char DLPF_CFG_0 = 1<<0;
char DLPF_CFG_1 = 1<<1;
char DLPF_CFG_2 = 1<<2;
char DLPF_FS_SEL_0 = 1<<3;
char DLPF_FS_SEL_1 = 1<<4;

//I2C devices each have an address. The address is defined in
the datasheet for the device. The ITG-3200 breakout board can
have different address depending on how
//the jumper on top of the board is configured. By default, th
e jumper is connected to the VDD pin. When the jumper is conne
cted to the VDD pin the I2C address
//is 0x69.
char itgAddress = 0x69;
```

This is the configuration section of the sketch. It looks more complicated than it is! First we include the **"Wire.h"** library, which comes standard with the Arduino IDE. This library is used for I^2C communication, which is the communication protocol used by the ITG-3200.

Next, there is list of variables which are assigned to different registers on the ITG-3200. Registers are used mostly to do two things: configure parameters for a sensor, or hold data that the sensor has collected. When we want to interact with a sensor's register, we must tell the sensor which register address we want to work with. After the list of registers is a short list of register parameters. This is only a list for the parameters used by this sketch. There are many more parameters listed in the ITG-3200 datasheet that aren't used in this example.

Finally, after the list of parameters, is the `itgAddress` variable. This is the I^2C address of the ITG-3200. The I^2C address for the sensor is also listed in the datasheet. Remember, this address is directly impacted by the configuration of the solder jumper on the top of the PCB.

```
//In the setup section of the sketch the serial port will be c
onfigured, the i2c communication will be initialized, and the
itg-3200 will be configured.
void setup()
{
  //Create a serial connection using a 9600bps baud rate.
  Serial.begin(9600);

  //Initialize the I2C communication. This will set the Arduin
o up as the 'Master' device.
  Wire.begin();

  //Read the WHO_AM_I register and print the result
  char id=0;
  id = itgRead(itgAddress, 0x00);
  Serial.print("ID: ");
  Serial.println(id, HEX);

  //Configure the gyroscope
  //Set the gyroscope scale for the outputs to +/-2000 degree
s per second
  itgWrite(itgAddress, DLPF_FS, (DLPF_FS_SEL_0|DLPF_FS_SEL_1|D
LPF_CFG_0));
  //Set the sample rate to 100 hz
  itgWrite(itgAddress, SMPLRT_DIV, 9);
}
```

The Setup section of the code is pretty short. First, we create a Serial connection so that we can print data to a terminal window. Then we initialize the I²C communication protocol. Now the Arduino is ready to start interacting with the ITG-3200. Most sensors have some kind of identification register. A good way to verify that the communication is working properly is to read the identification register and ensure the result is valid. After reading the identification register a couple of values are written to some registers on the ITG-3200 to configure the gyroscope to read data at 100hz and measure rotation rates up to 2000 degrees per second. The itgRead and itgWrite functions will be explained a little later. Once the device has been configured the actual gyroscope data can be read.

```
//The loop section of the sketch will read the X,Y and Z outpu
t rates from the gyroscope and output them in the Serial Termi
nal
void loop()
{
  //Create variables to hold the output rates.
  int xRate, yRate, zRate;
  //Read the x,y and z output rates from the gyroscope.
  xRate = readX();
  yRate = readY();
  zRate = readZ();
  //Print the output rates to the terminal, seperated by a TA
B character.
  Serial.print(xRate);
  Serial.print('\t');
  Serial.print(yRate);
  Serial.print('\t');
  Serial.println(zRate);

  //Wait 10ms before reading the values again. (Remember, the
output rate was set to 100hz and 1reading per 10ms = 100hz.)
  delay(10);
}
```

The Loop section of the code is usually the ‘meat’ of the sketch, in this case the loop is very straightforward. The sketch reads the X, Y, and Z gyroscope values using the readX(), readY() and readZ() functions. After storing these values, they are printed to the serial terminal. We delay 10 ms at the end of the loop so that we don’t try reading information from the sensor faster than it can be provided. For the Setup and Loop sections of the code to look so simple we had to use a couple of functions, let’s see how the functions work.

```
//This function will write a value to a register on the itg-32
00.
//Parameters:
// char address: The I2C address of the sensor. For the ITG-32
00 breakout the address is 0x69.
// char registerAddress: The address of the register on the se
nsor that should be written to.
// char data: The value to be written to the specified registe
r.
void itgWrite(char address, char registerAddress, char data)
{
  //Initiate a communication sequence with the desired i2c dev
ice
  Wire.beginTransmission(address);
  //Tell the I2C address which register we are writing to
  Wire.write(registerAddress);
  //Send the value to write to the specified register
  Wire.write(data);
  //End the communication sequence
  Wire.endTransmission();
}

//This function will read the data from a specified register o
n the ITG-3200 and return the value.
//Parameters:
// char address: The I2C address of the sensor. For the ITG-32
00 breakout the address is 0x69.
// char registerAddress: The address of the register on the se
nsor that should be read
//Return:
// unsigned char: The value currently residing in the specifie
d register
unsigned char itgRead(char address, char registerAddress)
{
  //This variable will hold the contents read from the i2c dev
ice.
  unsigned char data=0;

  //Send the register address to be read.
  Wire.beginTransmission(address);
  //Send the Register Address
  Wire.write(registerAddress);
  //End the communication sequence.
  Wire.endTransmission();

  //Ask the I2C device for data
  Wire.beginTransmission(address);
  Wire.requestFrom(address, 1);

  //Wait for a response from the I2C device
  if(Wire.available()){
    //Save the data sent from the I2C device
    data = Wire.read();
  }

  //End the communication sequence.
  Wire.endTransmission();

  //Return the data read during the operation
  return data;
}

//This function is used to read the X-Axis rate of the gyrosco
pe. The function returns the ADC value from the Gyroscope
```

```
//NOTE: This value is NOT in degrees per second.
//Usage: int xRate = readX();
int readX(void)
{
  int data=0;
  data = itgRead(itgAddress, GYRO_XOUT_H)<<8;
  data |= itgRead(itgAddress, GYRO_XOUT_L);

  return data;
}

//This function is used to read the Y-Axis rate of the gyrosco
pe. The function returns the ADC value from the Gyroscope
//NOTE: This value is NOT in degrees per second.
//Usage: int yRate = readY();
int readY(void)
{
  int data=0;
  data = itgRead(itgAddress, GYRO_YOUT_H)<<8;
  data |= itgRead(itgAddress, GYRO_YOUT_L);

  return data;
}

//This function is used to read the Z-Axis rate of the gyrosco
pe. The function returns the ADC value from the Gyroscope
//NOTE: This value is NOT in degrees per second.
//Usage: int zRate = readZ();
int readZ(void)
{
  int data=0;
  data = itgRead(itgAddress, GYRO_ZOUT_H)<<8;
  data |= itgRead(itgAddress, GYRO_ZOUT_L);

  return data;
}
```

There are five functions in this sketch, but three of them are very similar. The first function, `itgWrite()`, is used to write a value to a register on the ITG-3200. To use this function three parameters must be provided: the address, the registerAddress, and the data. The address is the I²C address of the sensor. As it turns out, more than one sensor can be connected to theI²C pins at the same time. In order for the sensors to know who is supposed to be getting the data, they each have a unique address. That's what we're providing with the 'address' parameter. The second parameter is the registerAddress. Like we discussed earlier, most sensors have a set of registers, and each register has it's own address. The last parameter is the data to be written to the address. We can configure a parameter on a sensor by writing data to a register address.

The next function is the `itgRead()` function. This function allows us to read the data stored in the register of a sensor. The itgRead function requires two parameters, and it returns a character value. The parameters are similar to those in the `itgWrite()` function; the address is the I²C address of the sensor we want to read from, and the registerAddress is the address of the register we want to read. The function will send the contents of the register back.

Running the Sketch

Once you've connected the ITG-3200 breakout board to the Arduino you can upload the ITG3200 Basic Arduino sketch. To see the data from the gyroscope, just open the serial terminal with a baud rate setting of 9600. You'll see values start streaming through the terminal window almost

immediately. On each line of the terminal, there are three values: x, y, and z rotation values. Remember, we didn't convert this data to degrees per second so the values that are being streamed are the ADC values from the ITG-3200. You may also notice that even if the gyroscope is sitting still (not rotating in any direction) the values aren't reporting 0. This is because there is an inherent bias in the gyroscope. To get accurate measurements you'll need to calibrate the readings. You can do this in the sketch by reading the values output from the sensor while it is sitting still and storing them into some variables. Then later, when the sensor values are being read, just offset the readings by the calibration values.

Resources and Going Further

You should now have a good understanding of how the ITG-3200 works. Now get out there and make some cool projects! If you need more info on the ITG-3200, make sure you check out the datasheet.

Want to learn more about gyroscopes? Check out our buying guide to learn about all the variteies SparkFun carries.