ICM20948 : Getting Started

M.B.

08.08.2025

Contents

1 About the ICM20948

The ICM20948 chip is a 9-axis MEMS motion-tracking device providing:

- a 3 axis accelerometer
- a 3 axis gyroscope
- a 3 axis compass

The datasheet can be found here:

`https://invensense.tdk.com/download-pdf/icm-20948-datasheet`

Additional information about how to connect the sensor to a Raspberry Pi can be found here: `https://cdn-learn.adafruit.com/downloads/pdf/adafruit-tdk-invensense-icm-20948-9-do` `pdf`

According to this document the ICM20948 (communication via I2C) must be connected like this:

ICM20948 connector	**Raspberry Pi pin**
Vin (red wire)	Pin#1
GND (black wire)	Pin#6
SCL (yellow wire)	Pin#5
SDA (blue wire)	Pin#3

Images of the ICM20948 and its I2C connection are shown here:

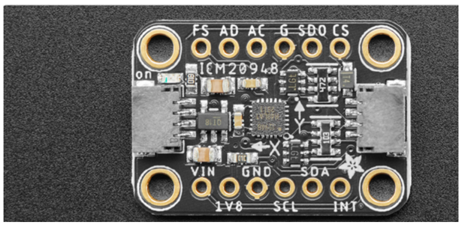

Figure 1: breakout board / ICM20948

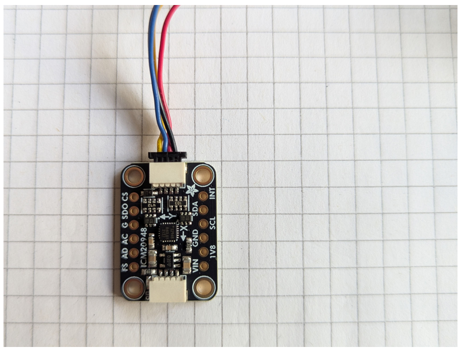

Figure 2: ICM20948 and its connections

And here is a resource from the internet showing the Pins available on a Raspberry Pi 3/4.

Accessing the ICM20948 is done via I2C. So some preliminary tests should give us enough confidence that communication via I2C does work in principle. More advanced

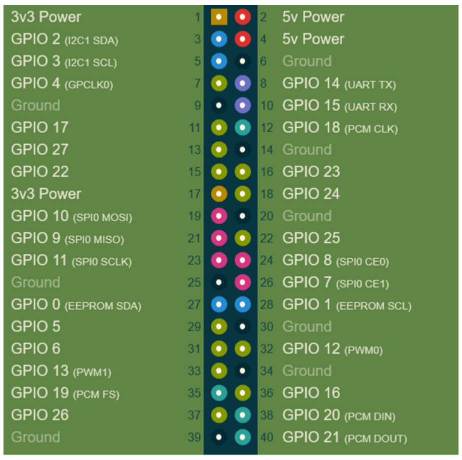
3v3 Power 1 2 5v Power
GPIO 2 (I2C1 SDA) 3 4 5v Power
GPIO 3 (I2C1 SCL) 5 6 Ground
GPIO 4 (GPCLK0) 7 8 GPIO 14 (UART TX)
Ground 9 10 GPIO 15 (UART RX)
GPIO 17 11 12 GPIO 18 (PCM CLK)
GPIO 27 13 14 Ground
GPIO 22 15 16 GPIO 23
3v3 Power 17 18 GPIO 24
GPIO 10 (SPI0 MOSI) 19 20 Ground
GPIO 9 (SPI0 MISO) 21 22 GPIO 25
GPIO 11 (SPI0 SCLK) 23 24 GPIO 8 (SPI0 CE0)
Ground 25 26 GPIO 7 (SPI0 CE1)
GPIO 0 (EEPROM SDA) 27 28 GPIO 1 (EEPROM SCL)
GPIO 5 29 30 Ground
GPIO 6 31 32 GPIO 12 (PWM0)
GPIO 13 (PWM1) 33 34 Ground
GPIO 19 (PCM FS) 35 36 GPIO 16
GPIO 26 37 38 GPIO 20 (PCM DIN)
Ground 39 40 GPIO 21 (PCM DOUT)

Figure 3: GPIO pins / Raspberry Pi 3

interactions with the ICM20948 will usually be accomplished via Python programs.

A very readable book on how to interface with the Raspberry Pi is

`Raspberry PI, interfacing to the real world with embedded Linux`; autor: Derek Molloy.

This book has a chapter `Interfacing to the Raspberry Pi Buses` which also covers some Linux I2C tools:

- i2cdetect
- i2cdump
- i2cget
- i2cset

```
mbi1955@raspberrypi:~ $ i2cdetect -y 1
0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- 69 -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

and reading out `WHO_AM_I` yields this output:

```
mbi1955@raspberrypi:~ $ i2cget -y 1 0x69 0x00
0xea
```

which is ok according to the datasheet. Below is a figure showing the ICM20948 sensor connected to a Raspberry Pi.

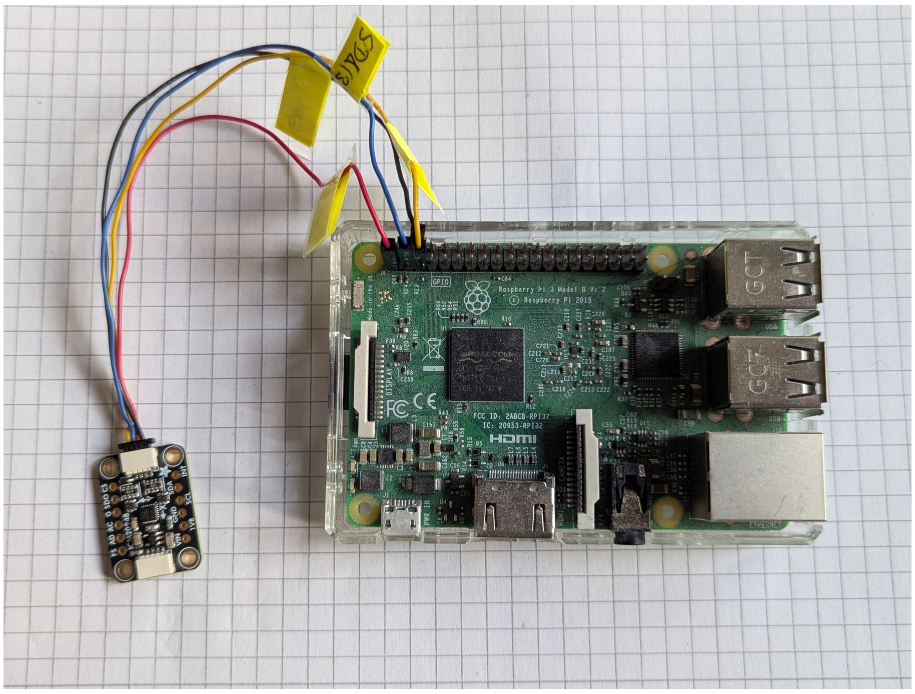

Figure 4: Raspberry Pi 3 with ICM20948

2 Programming the ICM20948 with Python

To access the I2C bus programmatically from Python at least the `smbus2` library needs to be installed. The library defines a class `SMBus` with methods for reading / writing to a I2C device.

The documentation of the library is available at `https://smbus2.readthedocs.io/en/latest`.

To see whether the ICM20948 sensor can be accessed a test is done directly in the Python shell of the Raspberry Pi. Here is an interactive session which reads out the chip id of the ICM20948.

```
mbi1955@raspberrypi:~ $ python3
Python 3.9.2 (default, Mar 20 2025, 22:21:41)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from smbus2 import SMBus
>>> ic2bus = SMBus(1)
>>> I2C_ADDR = 0x69
>>> WHO_AM_I = 0x00
>>> BANK_SEL = 0x7f
>>> bank = 0
>>> ic2bus.write_byte_data(I2C_ADDR, BANK_SEL, bank)
>>> chip_id = ic2bus.read_byte_data(I2C_ADDR, WHO_AM_I)
>>> print(f"who_am_i: {hex(chip_id)}")
who_am_i: 0xea
```

Not surprisingly the output `0xea` is identical to what we got using Linux command `i2cget`.

For a more convenient working environment another library seems to be useful.

Library `https://pypi.org/project/icm20948/` defines some methods which make working with the ICM20948 a bit easier. It defines some constants and addresses along with a Python class `ICM20948` which has methods for initialisation and reading/writing .

Before trying to understand some details how the sensor is initially configured, the usage of the library shall be tested with a Python program.

```
# icm20948_test.py

from icm20948 import *
import time

if __name__ == "__main__":
imu = ICM20948(i2c_addr=I2C_ADDR_ALT, i2c_bus=None)

for k in range(3):
x, y, z = imu.read_magnetometer_data()
ax, ay, az, gx, gy, gz = imu.read_accelerometer_gyro_data()

print(f"""
Accel: {ax:05.2f} {ay:05.2f} {az:05.2f}
Gyro:  {gx:05.2f} {gy:05.2f} {gz:05.2f}
Mag:   {x:05.2f} {y:05.2f} {z:05.2f}""")

time.sleep(0.25)
```

And the output is:

```
mbi1955@raspberrypi:~/Projects/icm20948 $ /usr/bin/python3 /home/mbi1955/Projects/icm2094

Accel: -0.13 -0.33 00.96
Gyro:  -1.34 00.27 00.02
Mag:   -38.10 -67.05 74.10

Accel: -0.13 -0.33 00.95
Gyro:  -1.19 00.25 00.05
Mag:   -38.10 -67.05 74.40

Accel: -0.14 -0.33 00.95
Gyro:  -1.32 00.14 00.15
Mag:   -37.20 -67.20 73.20
```

The output shows that reading out the 3-axis values of

- accelerometer
- gyroscope
- magnetometer

works in principle. Three readings have been obtained with a time interval between readings of 0.25 seconds. Between the measurements the sensor has not been moved. Accordingly the values are more or less constant (apart from measurement noise).

Now that we have confirmed the functionality of library `icm20948` we shall look into the details of configuring the accelerometer, gyroscope and magnetometer. The library provides several methods to accomplish initial configuration tasks.

2.1 resetting the ICM20948

The MSB (Bit 7) of register `PWR_MGMT_1` is set.

Bits 2:0 of register `PWR_MGMT_1` are set to 1. The data sheet states: 1-5: Auto selects the best available clock source – PLL if ready, else use the Internal oscillator.

Register PWR_MGMT_2 is set to 0x00. All axes of accelerometer and gyroscope are turned on.

2.2 accelerometer / initial configuration

The initial configuration comprises settings of the sampling rate, low pass filter, the scale/range of accelerometer.

2.2.1 samplerate

The samplerate can be configured in the range 4.5 to 1125 Hz. The actual sampling rate f_s is computed using this equation:

$$f_s = \frac{1125\ Hz}{1 + ratio}$$

where $ratio$ is a 12 bit number configured via registers ACCEL_SMPLRT_DIV_1 and ACCEL_SMPLRT_DIV_2.

Example: to configure a sampling rate of $f_s = 125\ Hz$ we must choose $ratio = 8$.

2.2.2 low pass filter

The low pass filter is enabled setting the LSB of register ACCEL_CONFIG to 1.

The bandwidth of the low pass filter is then configured via bits 5:3 of the same register.

2.2.3 full scale

Bits 2:1 of ACCEL_CONFIG configure the scale (min/max range) of the accelerometer.

2.3 gyroscope / initial configuration

2.3.1 samplerate

The samplerate can be configured in the range 4.5 to 1125 Hz. The actual sampling rate f_s is computed using this equation:

$$f_s = \frac{1125\ Hz}{1 + ratio}$$

where *ratio* is a 8 bit number configured via registers `GYRO_SMPLRT_DIV`.

Example: to configure a sampling rate of $f_s = 125\ Hz$ we must choose $ratio = 8$.

2.3.2 low pass filter

The low pass filter is enabled setting the LSB of register `GYRO_CONFIG_1` to 1.

The bandwidth of the low pass filter is then configured via bits 5:3 of the same register.

2.3.3 full scale

Bits 2:1 of `GYRO_CONFIG_1` configure the scale (min/max range) of the accelerometer.

2.4 magnetometer / initial configuration

The magnetometer is reset

3 Exploring the accelerometer and the gyroscope

A sample Python program has been written which records a user defined number of samples of the (x,y,z)-data of the accelerometer and the gyroscope. The data are stored in as compressed Numpy file.

The ICM20948 sensor is oriented in various static positions. For each position the recorded data are stored in a file.

A `Jupyter` notebook retrieves data from these files and produces some graphics to illustrate how accelerometer and gyroscope data behave under static, arbitrary but fixed orientation.

The static recordings indicate that for any meaningful application the accelerometer must be calibrated to compensate for a number of non-ideal properties:

- misalignment of x,y,z sensors (not being exactly mutually orthogonal)
- sensitivity error; scaling factors in x,y,z direction are not identical
- additive offsets

Another test for the gyroscope shows that readings will benefit from calibration as well. The sensors of the gyroscope have significant offsets. An example in the notebook shows, that angle data obtained from postprocessing raw gyroscope data exhibit a large drift over time if these offsets are not compensated for.