Accessing the AK09916 Magnetometer via I2C Passthrough on the ICM-20948

1. Introduction

This guide explains how to access and read raw magnetometer data from the AK09916, an embedded 3-axis magnetometer within the ICM-20948 IMU, using I2C passthrough mode on a Raspberry Pi Pico W (Pico 2W).

The AK09916 is not directly connected to the host microcontroller — instead, it sits behind the ICM-20948 and requires proper configuration to communicate over I2C.

This step-by-step guide walks through:

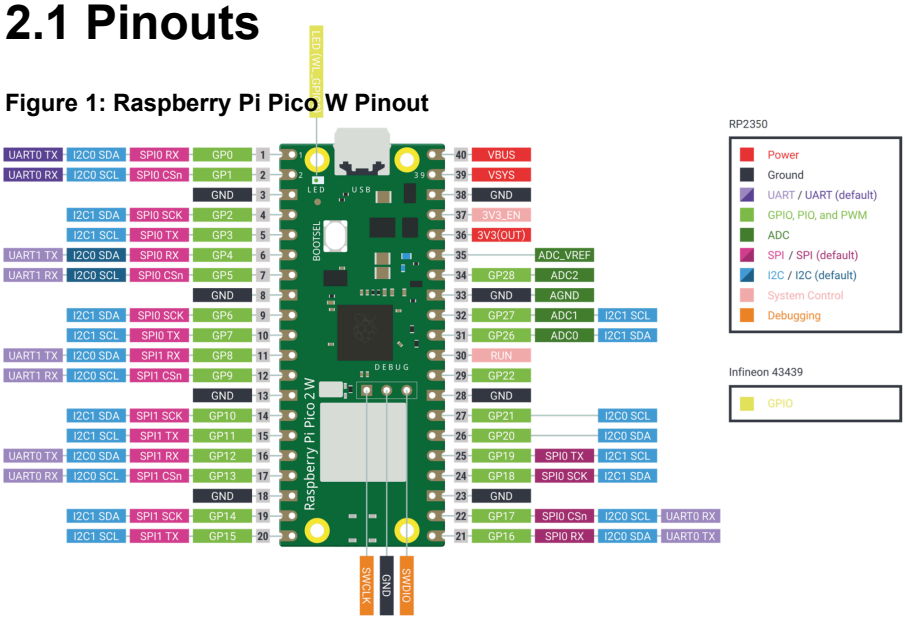
- Wiring the ICM-20948 to the Pico 2W
- Enabling passthrough mode via register configuration
- Verifying the connection by reading the **WHO_AM_I** register
- Reading and converting raw magnetometer data
- Common troubleshooting issues and fixes

All steps assume a basic familiarity with I2C, C/C++ development with the Pico SDK, and serial debugging via USB.

2. Hardware Setup

2.1 Pinouts

Figure 1: Raspberry Pi Pico W Pinout



For an I2C connection, you can use either `i2c0` or `i2c1` on the Raspberry Pi Pico. Use female-to-female jumper wires to connect the **SDA** (Serial Data) and **SCL** (Serial Clock) pins on the Pico to the corresponding **SDA** and **SCL** pins on the ICM-20948.

Additionally, connect **3.3V (3V3)** from the Pico to **VDD** on the ICM-20948, and connect **GND** to **GND**.

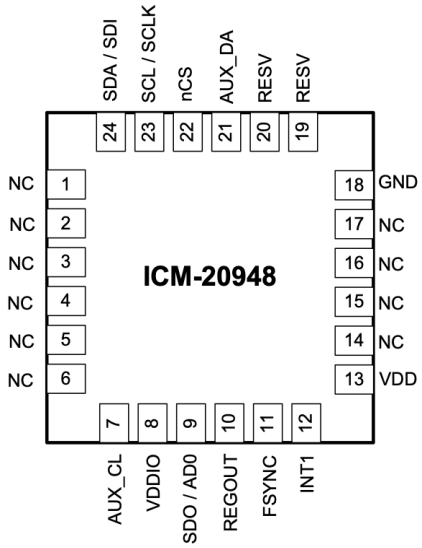


Figure 2: ICM-20948 Pinout

2.2 Initializing I2C and Read/Write from a Register

Once you have the physical pins connected, the next step is to create a C++ project using the Pico SDK. You'll need to define which GPIOs are used for I2C communication, based on the Pico pinout. You must also select the I2C port (either `i2c0` or `i2c1`) and be aware of the device address on the bus. For the ICM-20948, the I2C address is usually `0x68`, but it may be `0x69` depending on your breakout board's AD0 pin.

```
constexpr uint8_t SDA_PIN = 18;

constexpr uint8_t SCL_PIN = 19;

constexpr auto I2C_PORT = i2c1;
```

This init function:

- Initializes i2c1 at 400kHz (standard fast mode)
- Sets the GPIOs to I2C function mode
- Enables internal pull-ups (required for I2C communication)

```
void ICM20948::init() const {

    i2c_init(i2c_, 400'000);

    gpio_set_function(SDA_PIN, GPIO_FUNC_I2C);

    gpio_set_function(SCL_PIN, GPIO_FUNC_I2C);

    gpio_pull_up(SDA_PIN);

    gpio_pull_up(SCL_PIN);

}
```

The `write_reg()` function sends a single byte of data to a specific register on the I2C device. This is a fundamental operation for configuring the ICM-20948 or any device on the I2C bus.

```
void ICM20948::write_reg(uint8_t i2c_addr, uint8_t reg, uint8_t
value) const {

    uint8_t data[2] = {reg, value};

    i2c_write_blocking(i2c_, i2c_addr, data, 2, false);

}
```

The following helper function allows you to read the value of an 8-bit register from a given I2C address. It's useful for verifying I2C communication by checking known

registers — such as `WHO_AM_I` — which can confirm that your setup is working properly.

```
uint8_t ICM20948::read_reg(uint8_t i2c_addr, uint8_t reg) const
{
    uint8_t value = 0;
    i2c_write_blocking(i2c_, i2c_addr, &reg, 1, true);
    i2c_read_blocking(i2c_, i2c_addr, &value, 1, false);
    return value;
}
```

This function reads multiple consecutive bytes starting from a given register on an I2C device.

```
void ICM20948::read_reg_multi(uint8_t i2c_addr, uint8_t reg,
uint8_t* buf, size_t len) const {
    i2c_write_blocking(i2c_, i2c_addr, &reg, 1, true);
    i2c_read_blocking(i2c_, i2c_addr, buf, len, false);
}
```

3. Enabling I2C Passthrough Mode

Wake up the IMU and set it to the correct power mode:
Make sure the device is out of sleep mode by writing `0x01` to `PWR_MGMT_1` (register `0x06` in the user bank 0):

```
write_reg(icm_i2c_adr, 0x06, 0x01);  // Wake device and select auto clock
```

Enable the I2C Master:
The ICM-20948 contains an internal I2C master to communicate with the magnetometer. To enable passthrough, first disable the I2C master by clearing the relevant bit in the `USER_CTRL` register.

```
write_reg(icm_i2c_adr, 0x03, 0x00);  // USER_CTRL: disable I2C master
```

Enable passthrough mode in INT_PIN_CFG:
Write `0x02` to the `INT_PIN_CFG` register (still in user bank 0). This sets the `BY_PASS_EN` bit, which allows the host microcontroller to directly access the AK09916.

```
write_reg(icm_i2c_adr, 0x0F, 0x02);  // INT_PIN_CFG: enable I2C bypass
```

Verify:
After enabling bypass mode, you should be able to scan I2C addresses from the Pico and see the AK09916 at address `0x0C`. To confirm communication, read the magnetometer's `WHO_AM_I` register (0x01) and check if it returns `0x09`.

4. Reading Magnetometer Data

Once the magnetometer has been configured and I2C passthrough mode is enabled, you can begin reading raw magnetic field data from the AK09916. Each reading consists of three 16-bit signed integers representing the X, Y, and Z magnetic field components.

The standard sequence for reading data is as follows:

1. **Poll the ST1 (Status 1) register**
   Wait until the Data Ready (DRDY) bit is set, indicating new data is available.
2. **Read six data bytes from HXL to HZH**
   These bytes represent the X, Y, and Z axis readings, each in little-endian format (low byte first).

3. **Read ST2 (Status 2) register**
   This step finalizes the read and clears the DRDY flag. It also contains an overflow indicator (HOFL bit).
4. **Check for I2C errors**
   If any register returns `0xFF`, it typically indicates a failed I2C communication (e.g., disconnected or misconfigured device).

The function `read_magnetometer_raw()` in the code block below implements this full sequence and populates the output parameters with the latest raw data values.

```
bool ICM20948::read_magnetometer_raw(int16_t& x, int16_t& y,
int16_t& z) {

   // Wait for data ready

   while (!(read_reg(Magnetometer::ADDR, Magnetometer::ST1) &
0x01));



   // Read 6 bytes (X, Y, Z)

   uint8_t data[6];

   read_reg_multi(Magnetometer::ADDR, Magnetometer::HXL, data,
6);

   x = (data[1] << 8) | data[0];

   y = (data[3] << 8) | data[2];

   z = (data[5] << 8) | data[4];



   // Complete transaction by reading ST2

   read_reg(Magnetometer::ADDR, Magnetometer::ST2);
```

```
    return true;

}
```

5. Troubleshooting

If the magnetometer is not functioning as expected, refer to the following common issues and solutions:

I2C Device Not Detected

- Ensure that SDA and SCL are connected to the correct GPIO pins on the Pico.
- Verify that pull-up resistors are in place or that internal pull-ups are enabled via `gpio_pull_up()`.
- Confirm that I2C passthrough mode is enabled (`BYPASS_EN` bit in `INT_PIN_CFG` is set).
- Use an I2C scanner to verify the AK09916 appears at address `0x0C`.

WHO_AM_I Returns 0xFF or Incorrect Value

- Check that the ICM-20948 is out of sleep mode by writing `0x01` to `PWR_MGMT_1`.
- Make sure the correct user bank is selected before reading or writing registers.
- Verify that the I2C bus is not busy or held low by another device.

No Magnetometer Readings or All Zeros

- Ensure that the AK09916 is placed in continuous measurement mode (write `0x08` to `CNTL2`).
- Poll the `ST1` register until the `DRDY` (Data Ready) bit is set before reading data.
- Read the `ST2` register after every data read to complete the transaction and clear `DRDY`.

Unstable or Noisy Magnetometer Data

- Average multiple readings to reduce high-frequency noise.
- Check the `HOFL` (overflow) bit in `ST2` to ensure the sensor is operating within range.
- Verify that the ICM-20948 and AK09916 are properly grounded and powered.

- Ensure that the wiring is short and shielded from interference if possible.

Code Hangs or Unexpected Behavior

- Avoid infinite polling loops; consider adding a timeout when waiting for `DRDY`.
- Check the return value of `read_reg()` calls; `0xFF` may indicate a failed I2C read.
- Insert debug print statements to identify where the code may be stuck.

References

- ICM-20948 | InvenSense
- Raspberry Pi Pico 2 W Pinout
- Hardware APIs - Raspberry Pi Documentation
- AK09916 English Datasheet
- GitHub - raspberrypi/pico-sdk: