

Lab 6 - I2C Sensors

Goals

- Implement an I2C driver using an nRF SDK library
- Interpret sensor data into meaningful information

Equipment

- Computer with build environment
- Micro:bit and USB cable

Documentation

- nRF52833 datasheet: https://infocenter.nordicsemi.com/pdf/nRF52833_PS_v1.5.pdf
- Microbit schematic:
https://github.com/microbit-foundation/microbit-v2-hardware/blob/main/V2/MicroBit_V2.0_0_S_schematic.PDF
- Lecture slides are posted to the Canvas homepage
- LSM303AGR datasheet:
<https://drive.google.com/file/d/1wLJkEFv2RNPashABxo6drEbNSQXtdQU/view?usp=sharing>
- Using an Accelerometer for Inclination Sensing app note:
<https://www.analog.com/media/en/technical-documentation/app-notes/an-1057.pdf>

Github classroom link: <https://classroom.github.com/a/7aismnZP>

Lab 6 Checkoffs

You must be checked off by course staff to receive credit for this lab. This can be the instructor, TA, or PM during a Friday lab session or during office hours.

Part 2: I2C Sensors

- a. Demonstrate reading the WHOAMI registers for the Accelerometer and Magnetometer
- b. Demonstrate reading temperature from the sensor over I2C
- c. Demonstrate acceleration and magnetism measurements
- d. Demonstrate your application that prints out Temperature, Acceleration, Magnetic Field Strength, and Tilt Angle. Also explain the code that does this.

Also, don't forget to answer the lab questions assignment on Canvas.

Lab Steps

Part 1: Setup

1. Find a partner

- Rule: you can pick any partner you want, but you can't pick the same partner twice
- You MUST work with a partner
 - We don't have enough computers otherwise

2. Set up your computer

- You can also use your personal computer if it's set up and you prefer
- Log into Windows
 - Password: 327-19s
- Open VMWare Workstation Player
- Open the virtual machine: CE346
 - It'll load for a minute and then ask you to log in
 - Password: microbit

3. Create your Github assignment repo

- There is a github classroom link on the first page of this document. Click it!
- Pick a team name
- Pick your partner
- Generally, do what it says
- At the end, it should create a new private repo that you have access to for your code
 - Be sure to commit your code to this repo often during class!
 - If your computer crashes, all your files WILL BE LOST unless committed and pushed to Github.
- That link might 404. If it does, you first have to go to <https://github.com/nu-ce346-student> and join the organization

4. Create a personal access token

- If you still have your personal access token from prior weeks, you can skip this part
- Go to your github profile -> Settings
 - Then Developer Settings on the left
 - Then Personal Access Tokens on the left
 - Then click the Generate New Token button on the top right
- Add a note that is the name of this token (not important, type anything)
- **IMPORTANT:** check the repo checkbox below the name of the token
- Then scroll to the bottom of the screen and click the Generate Token button
- This will create a password that allows you to clone repos
 - It will only show this once, so copy-pasting it into a google doc in your personal drive would probably be useful
 - It will be in gray at the top of the screen

5. Clone your lab repo locally

- Open a terminal
- You can clone the repo right to the home directory of the computer
 - Remember, everything in this VM will disappear when it powers down
- At the top right of your shiny new private repo, there is a green button that says code. Copy the HTTPS link to your git repo from there.
- **IMPORTANT:** run this command and don't forget the flags
`git clone <YOUR-REPO-HTTPS-LINK-HERE> --recursive --shallow-submodules`
 - Remember to include both of those flags!
 - Recursive is necessary to clone submodules
 - Shallow submodules makes it like five minutes faster to run
 - If you forgot the flags, the command to fix this is:
`git submodule update --init --recursive --depth 1`

Part 2: I2C Sensors

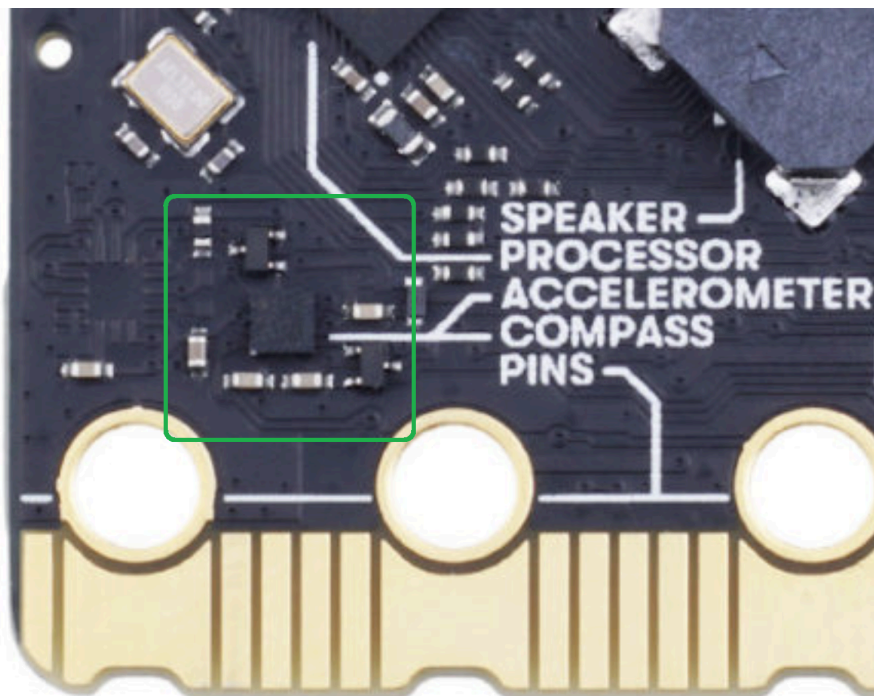
1. Find the app starter files for this lab

- `cd software/apps/i2c_sensors/`
 - This lab will use the files in this directory. Your changes will be in `main.c` and `lsm303agr.c`
- You will be interacting with the LSM303AGR today. I recommend you download the datasheet for it so that you can have a local copy open while working on the lab.
[LSM303AGR Datasheet](#)

2. Understand and double-check the I2C sensor

- Confirm that you have the correct I2C sensor

Your board should look like this, with the black IC for the accelerometer between the two leftmost holes. There is also a 3-pin transistor to the right of it and above it.



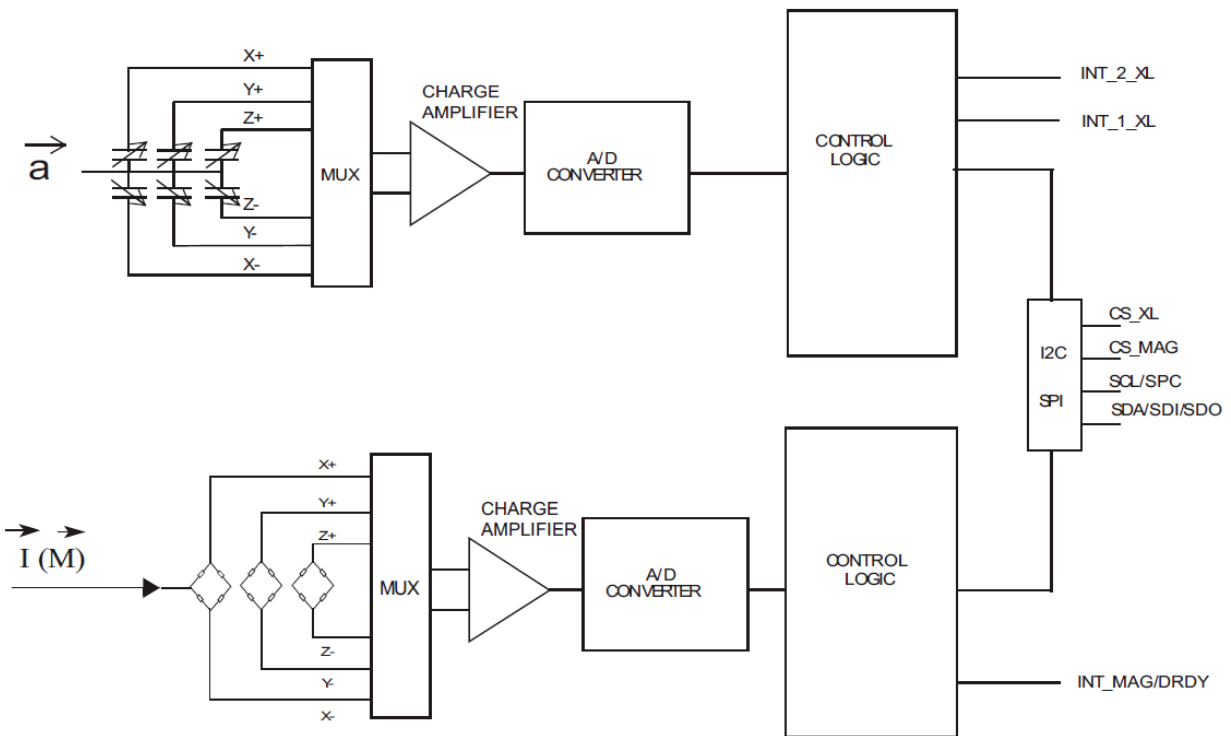
If your board instead has an IC to the left of this, above the leftmost hole, **let the course staff know immediately!!** (I don't think this will happen to anyone)

- Today we'll be interacting with the LSM303AGR accelerometer and magnetometer. Actually, this chip functions as two independent sensors both connected to the same I2C bus. The two sensors have different I2C addresses.

The accelerometer's address is where all accelerometer and temperature sensor commands and reads go.

The magnetometer's address is where all magnetometer commands and reads go.

See the picture below for a conceptual diagram of the sensor.



3. Implement I2C reads

- Create the proper transfer in the helper function

In the file `lsm303agr.c` there is a helper function `i2c_reg_read()` which performs a 1-byte read operation of a given I2C register for a given I2C device. You'll need to fill in the transfer that needs to occur.

- We are using the [nRF TWI Manager library](#) to interact with I2C devices. Initialization of the library has been done for you in `main()`. Calling `nrf_twi_mgr_perform()` actually runs a transaction. It expects a pointer to an array of `nrf_twi_mgr_transfer_t` structs, which are the actions it will perform. The easiest way to create these structs is with the [NRF_TWI_MNGR_WRITE\(\)](#) and [NRF_TWI_MNGR_READ\(\)](#) macros.
- I2C registers are addresses within the device where data is located. A list of I2C registers for the LSM303AGR can be found in Chapters 7 and 8 (starting on Page 43) of the datasheet.
- The transfer consists of a Write (to write the register address), a repeated start condition, and then a Read (to read the register value) as described in Table 22 of the datasheet (Page 38).
- You'll need to add two things to the array:
 - First, a write operation where the data value is the register address you want to read. That write operation should include a flag to ensure that it doesn't stop communication (which will create the repeated start condition).
 - Second, a read operation where the data pointer should be a pointer to the `rx_buf`. This operation should end with a normal stop condition (flags can be set to zero).
- Test that I2C reads work with the WHOAMI registers

A WHOAMI register always returns a constant value. You can read the value and confirm it has the expected value in order to confirm that I2C reads are working. We read the registers for you in `lsm303agr_init()`.

- The list of I2C registers for the LSM303AGR can be found in Chapters 7 and 8 (starting on Page 43) of the datasheet. There you'll find `WHO_AM_I_A`, which is the LSM303AGR Accelerometer's WHOAMI as well as `WHO_AM_I_M`, which is the LSM303AGR Magnetometer's WHOAMI.

- Check both of the values in `lsm303agr_init()` and print a message about whether they are correct or not. Note that there is a format directive for `printf()` that displays values in hex. You should use it!
 - If things aren't working, try physically unplugging and replugging your Microbit. A reset is enough to reset state on your Microcontroller, but it doesn't affect the sensor chip. So if that gets in a bad state, the only way to fix it is with a power cycle.
- **Checkoff:** demonstrate that you can read the WHOAMI registers

4. Read temperature from the sensor

- The I2C sensor has an internal temperature sensor so it can adjust its measurements based on temperature. We can read it as a test of our library.
- First, we need to get I2C writes working as well in order to configure the temperature sensing capability.

`i2c_reg_write()` is a helper function that should perform a 1-byte I2C write to a given device address and register. Implement this function so it performs the proper transaction.

- This is very similar to `i2c_reg_read()`. **One key difference is that the transaction array only needs to perform a single action now:** a two-byte write where the first byte is the register address and the second byte is the value to write to it. Before, we needed to do two transactions (a write then a read), whereas here we can just do a single two-byte write.

Note that you will need to change the arguments to `nrf_twi_mngr_perform()`.

- The best test of whether I2C writes are working is to implement the `lsm303agr_read_temperature()` function. It only uses calls to `i2c_reg_read()`, but the resulting data will always have a value of zero (25° C) if the write to configure the temperature sensor at the end of `lsm303agr_init()` has not occurred correctly.
 - The registers to read from are `OUT_TEMP_L_A` and `OUT_TEMP_H_A`, the least significant 8 bits and most significant 8 bits of a **signed** temperature value. You can find definitions of those register addresses in `lsm303agr.h`
 - You'll need to perform two I2C read operations, one for each register value. Then combine them together into a single `int16_t`
 - To convert to a temperature reading, first cast the value into a float, then multiply by the sensitivity, $\frac{1.0}{256.0}$, and add the bias, 25.0. The end result is a temperature in degrees Celsius.
- If your code isn't working correctly:
 - The debugging first step is to print out the values read from the accelerometer and see if they change with temperature.
 - If they are always zero, the problem is with your read/write functions. You may need to unplug/replug your Microbit to reset the accelerometer IC.

- If they are not always zero, you may have a problem with your combination and conversion logic. Try printing out the values in hexadecimal as well as the combined value in hexadecimal. Make sure the bit patterns look right.
- Test your temperature sensor reading by printing the value periodically in an app timer callback.

You should go look at earlier lab code to remember how to use the app timer.

It should increase if you put your finger on the I2C sensor, and generally is expected to be a little bit above room temperature as the entire sensor will heat up a little while running. If you always read a value of 25.0, that means the I2C write to configure the sensor did not work properly and you'll need to go back and look at that.

- **Checkoff:** demonstrate that you can read temperature values correctly

5. Read from the accelerometer and magnetometer

- You should fill in the functions `lsm303agr_read_accelerometer()` and `lsm303agr_read_magnetometer()`. The sensors have already been configured for you in `lsm303agr_init()`.
 - You can find the proper registers to read from in the datasheet. Like with temperature you'll need to read two different registers for each axis and combine them to create a measurement.
 - The combined measurement is a 10-bit signed value.
 - For the accelerometer, this 10-bit value is left-aligned, which means you must first store it, and then shift it right 6 bits to get the proper value.
 - For the magnetometer, this 10-bit value is right-aligned, which means that just storing it is sufficient. No shifting needed.
 - There is no bias, but there is a scaling factor for each sensor value. They can be found in Table 3 on page 13 of the datasheet. They are listed in values of UNIT/LSB, which is to say "scaling per least significant bit". You can multiply your value by them to get mg and mgauss respectively.
 - For the accelerometer, it is configured at +/- 2g in normal mode
 - For the magnetometer, the sensitivity is one number regardless of configuration
 - Further convert the sensor values into meaningful units.
 - The accelerometer readings are expected to be returned as g (rather than mg)
 - The magnetometer readings are expected to be returned as μT (microteslas). You can divide by 10 to convert from milligauss to microtesla
 - The values should be returned in the `lsm303agr_measurement_t` struct, which is defined in `lsm303agr.h`
- Test your sensor reading implementations by printing the values periodically in an app timer callback
 - The accelerometer measures the force of gravity while static. Gravity measures as "1 g" on the sensor. By default, you should read 1 g on the z-axis with the speaker side of the Microbit up and -1 g on the z-axis with the LED matrix side of the Microbit up. Other axes measure values when the Microbit is rotated.

- The magnetometer readings are hard to put into perspective. If you have a magnet handy (or most headphones have magnets in them), you can hold it close to the sensor to show a change in readings. Generally, you should be getting values in the 1-100 μT range with no magnet nearby. A nearby magnet will record values of thousands of μT .
- **Checkoff:** demonstrate the measured acceleration and magnetism readings

6. Convert acceleration to tilt angle

- Knowing that the force of gravity creates a constant downwards 1 g acceleration, we can use the readings on the axes of the Microbit to determine its tilt angle.
- Implement a function to determine tilt angle from acceleration

[This application note](#) explains how to convert accelerometer measurements into tilt angle. We want to implement triple-axis tilt measurements, using the technique described in the left column of page 7 of the application note.

- The angles Theta (θ), Psi (Ψ), and Phi (Φ) are angle offsets from the X, Y, and Z axes respectively and correspond to [Pitch, Roll, and Yaw](#) respectively. Assuming the Microbit is usually held face-up or face-down, the most important of these from our perspective is Phi (Φ), which measures tilt from the Z axis.
- You should implement this as a helper function in `lsm303agr.c` that either takes in accelerometer data and returns an angle measurement or reads the accelerometer itself and then returns an angle measurement
- Use equation 13 to calculate Phi. The [atan\(\)](#) and [sqrt\(\)](#) functions in `math.h` will be useful here.
 - **Warning:** be sure to `#include <math.h>` at the top of your file, or it might use a built-in version of `sqrt()` that gives very different results... (I *think* what's happening is that the built-in works on integers, not floats.)
- The resulting value will be in radians, so convert it into degrees before returning it because nobody actually speaks radians.
- Test your conversion by printing the tilt value of the Microbit periodically in an app timer callback
 - The Phi value should be close to zero if the Microbit is face up or face down, and close to +/- 90 degrees if the Microbit is held vertically (in any orientation). Sign tells you which side is up (positive for the speaker or negative for the LED matrix).
- **Checkoff:** demonstrate your code and application to course staff
 - It should print out Temperature, Acceleration, Magnetic Field Strength, and Tilt Angle