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
- Microbit breakout + QWIIC cable + random I2C device

Documentation

- nRF52833 datasheet: https://docs-be.nordicsemi.com/bundle/ps_nrf52833/attach/nRF52833_PS_v1.7.pdf
    - Online version: https://docs.nordicsemi.com/bundle/ps_nrf52833/page/keyfeatures_html5.html
- 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/1wLJlkEFv2RNPashABxo6drEbNSQXtdQU/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/XhXxqXbv

Lab 6 - I2C Sensors
- Lab 6 Checkoffs
- Lab Steps
- Part 1: Setup
  - 1. Find a partner
  - 2. Create your Github assignment repo
  - 3. Set up an additional Git remote
  - 4. Individual Setup Portions
- Part 2: I2C Accelerometer
  - 1. Find the app starter files for this lab
  - 2. Understand and double-check the I2C sensor
  - 3. Implement I2C reads
  - 4. Read temperature from the sensor
  - 5. Read from the accelerometer and magnetometer
  - 6. Convert acceleration to tilt angle
- Part 3: I2C Sensor from Scratch
  - 1. Grab a random I2C device
  - 2. Create starter files for this lab
  - 3. Find the details for your device
  - 4. Implement your device

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 Accelerometer

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.

Part 3: I2C Sensor from Scratch

a. Demonstrate your application and code
b. Return your hardware

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
    - If you can't find someone, talk to Branden

2. 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!
- That link might 404. If it does, you first have to go to https://github.com/nu-ce346-student and join the organization
- **Important: both of you should join the repo before you can do the next step**

3. Set up an additional Git remote

- Open a terminal if you haven't yet
- `cd` into your "nu-microbit-base" repo
- At the top right of your shiny new private repo on the Github website, there is a green button that says "Code". If you set up an SSH key, you can click the SSH tab to get that URL, otherwise you should get the HTTPS URL. Either way, copy the URL so you can enter it into terminal
- `git remote add lab6 <YOUR-REPO-URL-HERE>`
    - This adds a "remote" repo hosted on github as a source for this repo
    - (Both of you should still do this step too)

4. Individual Setup Portions

- **ONLY ONE OF YOU** should do the following steps
  - `git fetch lab6`
    - This gets the most recent commits from the new remote source
  - `git checkout lab6/main`
    - This changes your current commit to the remote source's main branch
  - `git switch -c lab6-code`
    - This makes a new branch for your lab code
  - `git push -u lab6 lab6-code`
    - This tells the new branch to push code to the new remote source
    - From now on, you can just pull, commit, and push as normal
- **THE OTHER STUDENT** should do this **AFTER** the first student finished the above steps:
  - `git fetch lab6`
  - `git switch lab6-code`
- **BOTH STUDENTS** should do this
  - `git submodule update --init --recursive`
    - Makes sure all git submodules are initialized and updated

Part 2: I2C Accelerometer

1. Find the app starter files for this lab

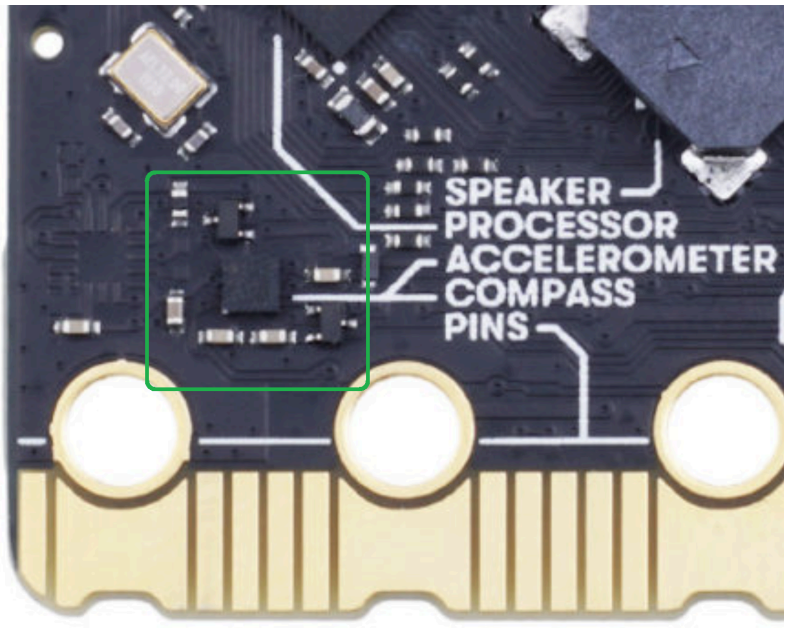
- `cd software/apps/i2c_accel_mag/`
    - This lab will use the files in this directory. Your changes will be in `main.c` and `lsm303agr.c`

- 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 Microbit 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 the little capacitor on the left moved, that's not a problem.



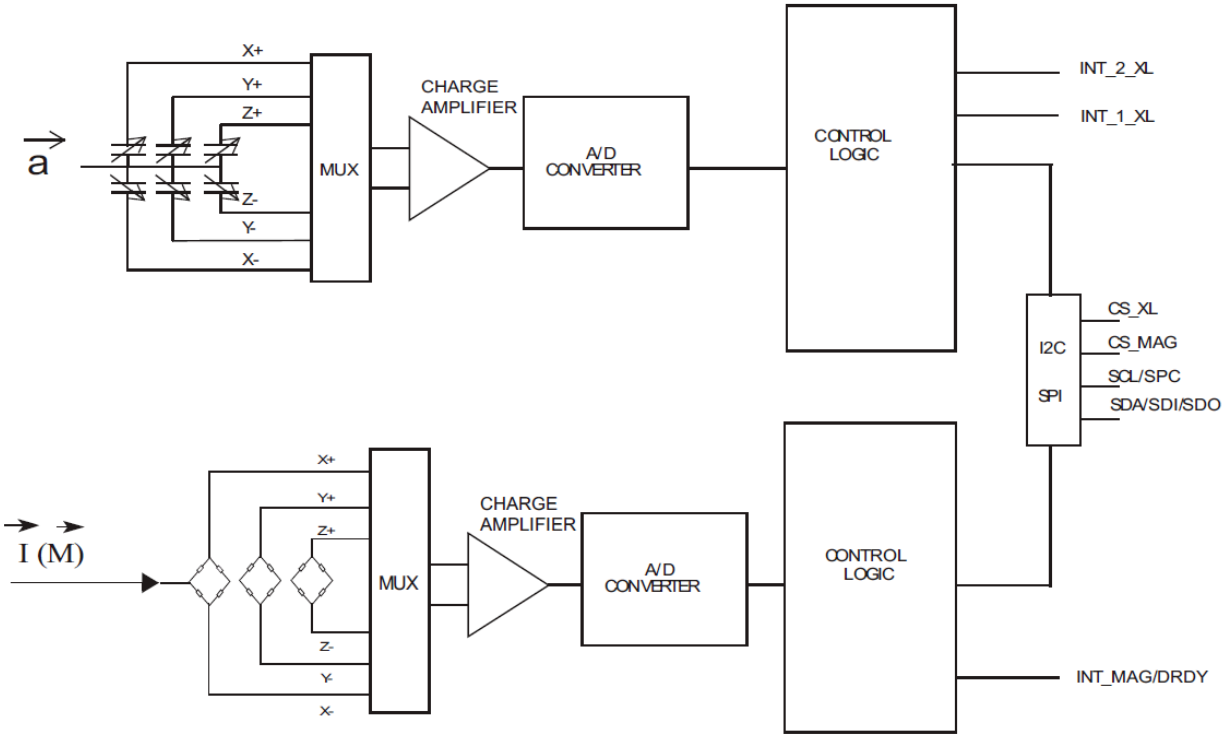
  If your board instead has an IC to the left of this, above the leftmost hole, **let the course staff know!! This lab won't work.** (I don't think this will happen to anyone)

- 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_mngr_perform()` actually runs a transaction. It expects a pointer to an array of `nrf_twi_mngr_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. You'll need to read the registers 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.

    - The register definitions can be found in `lsm303agr.h`

    - 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. Plus, it's amusing to implement yet another temperature driver (internal peripheral, analog, and now I2C).

- 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. Don't blindly copy-paste!

    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.

  - WARNING: don't just call `lsm303agr_read_temperature()` from the init function. The sensor needs a second to finish initializing before you can read it, so put stuff in `main.c` instead after a tiny delay (or repeatedly like the next step says to).

- 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. You may not just call it from the while loop in `main()`.

  The temperature reading should increase if you put your finger on the I2C sensor (which is on the bottom left of the front of the board), 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.

- 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.

- **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 (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 range of 1-100 μT in magnitude 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 aX es 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

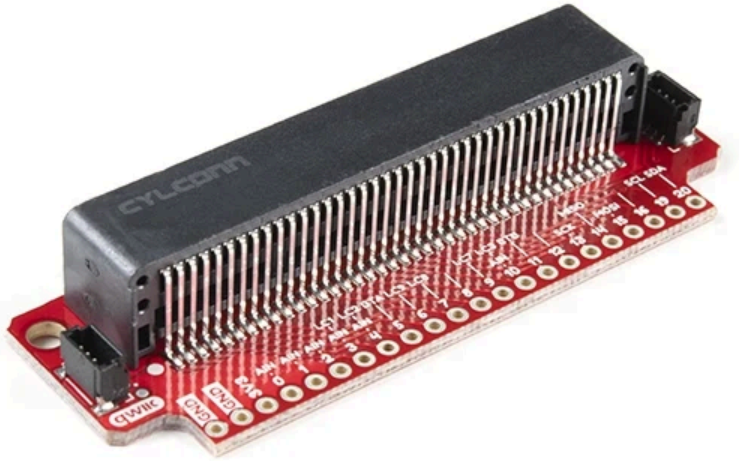
  - It should print out Temperature, Acceleration, Magnetic Field Strength, and Tilt Angle

Part 3: I2C Sensor from Scratch

1. Grab a random I2C device

- We have a box of I2C sensors available, give it a (gentle) shake, reach a hand in, and pull out your prize: the gift of learning how to implement a brand new sensor.
    - There are at least three different sensors available, so not every group will be implementing the same sensor.
    - You'll also need a Microbit breakout and a QWIIC cable

- You will need to connect this to your Microbit. The breakout has little black headers on the left and right sides that are used for QWIIC cables.
    - First, visually inspect those headers. There should be 4 little pins inside. If the pins look all bent or messed up use the other header.
    - Second, be gentle when plugging in the QWIIC cable so you don't mess up the pins.



2. Create starter files for this lab

- In the `software/apps/` directory, make a new directory for your app. You can name it whatever you like.
    - Copy the `Makefile` from `software/apps/i2c_accel_mag/` into your new application.
    - You may also want to copy `main.c`.
    - You will want a driver for your sensor, both a `.c` and `.h` file, named appropriately for what you're working on

- **WARNING:** in `main.c` you'll need to update the I2C pins to use `I2C_QWIIC_SCL` and `I2C_QWIIC_SDA` in order to communicate with an external sensor

3. Find the details for your device

Here is the list of sensors that you might have. For each, we'll provide a link to the part we purchased, a link to the datasheet for the chip, and a link to some Arduino example code implemented for the chip.

In general, this is the same process you should apply when implementing drivers for new hardware as a part of the final project. You won't be able to copy-paste the code from the example apps as they're in the Arduino framework rather than our lab system, but hopefully the datasheet and example code together can help fill in your understanding of how the device works so you can write your own code.

- Non-Volatile Memory 24LC32
  - 4 kilobyte EEPROM memory chip (not technically a sensor, but works pretty similarly)
  - Part details: https://www.adafruit.com/product/5146
  - Datasheet: https://ww1.microchip.com/downloads/en/DeviceDoc/21072G.pdf
  - Example code: https://github.com/gsampallo/EEPROM_24LC32A_I2C/blob/master/EEPROM_24LC32A_I2C.cpp

- Light Sensor BH1750
  - Light sensor, measures lux
  - Part details: https://www.adafruit.com/product/4681
  - Datasheet: https://www.mouser.com/datasheet/2/348/bh1750fvi-e-186247.pdf?srsltid=AfmBOoqtY5DA2YliIRQrUehYdrWkYQmZ_nDEcdaIJ8jPfa-X86-ePchI
  - Example code: https://github.com/claws/BH1750/blob/master/src/BH1750.cpp

- Temperature & Humidity Sensor AHT20
  - Temperature and Humidity sensor, measures relative humidity and degrees Celsius
  - Part details: https://www.adafruit.com/product/4566
  - Datasheet: https://files.seeedstudio.com/wiki/Grove-AHT20_I2C_Industrial_Grade_Temperature_and_Humidity_Sensor/AHT20-datasheet-2020-4-16.pdf
  - Example code: https://github.com/adafruit/Adafruit_AHTX0/blob/master/Adafruit_AHTX0.cpp

4. Implement your device

- First you'll need to implement a driver that can initialize the device and read data from it. What exactly this looks like varies between devices. Use the datasheets and example code to figure out how your device should work.

  Definitely copy things from your previous implementation of the Microbit accelerometer/magnetometer, but be careful to think as you do so. Many bugs in this lab are "copy-and-paste" bugs where students forget to update some field when pasting.

  Here are some hints:
  - Non-Volatile Memory
    - No particular initialization for this device
    - You will have to implement reading and writing though, but just a single-byte write and single-byte read is sufficient.
  - Light Sensor
    - You will definitely need to initialize this correctly to power it on.
    - Reading lux while in continuous high-resolution mode is sufficient.
  - Temperature & Humidity Sensor
    - This has initialization. The datasheet is very specific about what you need to do (although it really poorly explains why you need to do those things or what the bytes mean) and it works well if you follow it precisely.
    - You can ignore the CRC, but you definitely need to translate sensor data into meaningful units.

- Second, you'll need to implement a demo application just like we did with the Microbit accelerometer/magnetometer. Just calling a timer once per second and printing out some data seems pretty reasonable.

  Here are some requirements:
  - Non-Volatile Memory
    - Once per second, first read from some address and print out the value, then increment the value and write back to that same address.
    - When reset or even unplugged/replugged, this app should continue where it left off (because the value was written to non-volatile memory)
  - Light Sensor
    - Just read and print lux once per second.
  - Temperature & Humidity Sensor
    - Just read and print relative humidity and temperature in degrees Celsius once per second.

- **CHECKOFF**: demonstrate your code and application to course staff

- **CHECKOFF:** return your I2C device, Microbit breakout, and QWIIC cable