

Lab 4 - Breadboarding

Goals

- Develop circuits using a breadboard
- Use the Microbit to interact with external components

Equipment

- Computer with build environment
- Micro:bit and USB cable
- [Adafruit parts pal](#)

Documentation

- nRF52833 datasheet: https://infocenter.nordicsemi.com/pdf/nRF52833_PS_v1.3.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

Lab Steps

1. Update your local repository
 - cd into the base of your repo
 - git pull <https://github.com/nu-ce346/nu-microbit-base.git>
 - git submodule update --init --recursive
 - There may be no changes
- cd software/apps/breadboard/
 - This lab will use the files in this directory. Your changes will be in main.c
 - This lab builds upon itself. So keep things on the breadboard unless told to remove them. They'll be useful for the final checkoff.

2. Power an LED

- Connect the Microbit to your breadboard

The Microbit breakout should go in column “j” of the breadboard, rows 1 and down. The Microbit plugs into the breakout with the LED Matrix side up.

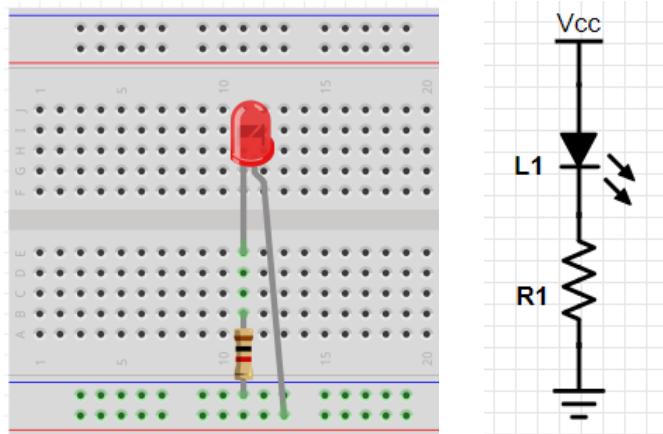
- Connect breadboard VCC and Ground rails to the Microbit power supply

Use a jumper wire to connect Ground (either of the white “GND” pins) to the Ground rail on the breadboard (the blue “-” column). Use a second jumper wire to connect VCC (the pin labeled 3V3, for 3.3 volts) to the VCC rail on the breadboard (the red “+” column).

- Power an LED

WARNING: Always ensure that there is a resistor between your LED and Ground!!! If you do not do this, you are creating a short-circuit, which will possibly damage your LED or even your Microbit.

- Connect a 1 k Ω resistor from the Ground rail to a row on the breadboard. (Color code Brown, Black, Red)
- Connect the short leg of the LED to the row with that resistor (the side of the LED with the shorter leg is also flattened on the plastic LED diffuser).
- Connect the long leg of the LED to the VCC rail on the breadboard.
- The end result should look like this and the LED should light up! If not, double-check that the Microbit is powered and that your circuit is correct.



- **No checkoff:** continue to the next step

3. Control the RGB LED

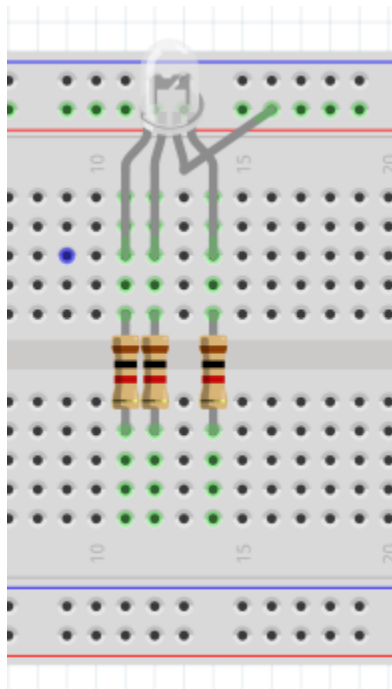
- Remove the existing LED

That was just a test to make sure that power is working and you can plug components into the breadboard properly. Everything from here on out will be relevant for the final checkoff.

- Wire the RGB LED to digital outputs

See Page 3 of the [LED datasheet](#) for details on its pins.

- The longest pin should connect directly to the VCC rail.
- The shorter pins should each connect to separate rows in the breadboard.
- Connected to each row should be a resistor (1 k Ω or 10 k Ω are fine). The other side of the resistor should connect to the row for the digital pin that controls that color channel of the LED. Use pins 13-15, as specified for each color in `main.c`
- When you are finished, the layout should look something like this, but with the resistors connecting to the same rows as the Microbit breakout pins 13-15 respectively.



- Configure the pins and write to them to control the LED

Use the [nRF gpio library](#) to configure and set the pins.

As you've done in previous labs, each pin needs to be configured as a digital output. There are `#defines` at the top of `main.c` that you can use as the GPIO pin number. They are renames of pins from `boards/microbit_v2/microbit_v2.h`

Based on how you wired the LED channels, they will be active low. Clearing the GPIO pin should light up the respective color of the RGB LED.

- Test that you can control each color channel of the LED.

A common mistake is that a GPIO pin controls a different color channel than expected. To overcome this, you can either re-wire the circuit, or simply change the `#defines` at the top of `main.c` to swap pins.

If it doesn't work, make that the correct LED pin is connected to the VCC rail.

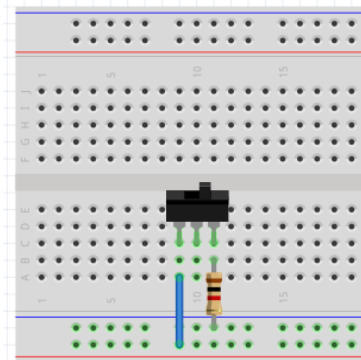
- **No checkoff:** continue to the next step

4. Read a switch

- Wire up the switch

We'll connect the [slide switch](#) in your kit to the Microbit as a digital input.

- The switch should be installed in the breadboard so that each pin is in a different row. It takes a little bit of force to push into the breadboard. Don't worry about it.
- Connect one edge pin of the slide switch to the VCC rail with a jumper wire.
- Connect the other edge pin of the slide switch to the Ground rail with a 1 k Ω resistor. **IMPORTANT:** use a resistor to connect to ground. Our switches seem to be crappy and are briefly shorting their outer pins when switching, causing the Microbit to reset if no resistor is used.
- Connect the middle pin of the slide switch to pin 16 of the Microbit breakout.
- The slide switch on the breadboard should look like this, except with the middle pin connected to the Microbit breakout.



- Configure and read in a digital input

The pin will need to be configured as an input. No pulling resistor is needed, since the input signal will always be either VCC or Ground.

Afterwards, reading the pin should give you the state of the switch and should change as you slide the switch.

- Test that you can properly read in the state of the switch

If it doesn't work, check that you've connected to the edge pins correctly. The three pins of the switch occupy three consecutive rows when installed.

- **No checkoff:** continue to the next step

5. Read an analog input

- Configure and read the ADC

You'll be using the [nRF SAADC library](#). All the code has been provided for you in `main.c`

- `adc_init()` configures the ADC to read in 12-bit samples. It also initializes each channel with the proper pin and with a range of 0 to 3.6 volts. (It connects the internal 0.6 Volt reference and applies a gain of 1/6 to the signal.)
- `adc_sample_blocking()` allows you to read an analog sample. Samples are read in as a 16-bit signed number, but will typically range from 0-4095. (Occasionally you'll get a small negative number, for example -6. I don't know why or how that is working...)

- Convert ADC counts to volts

The units of each sample in `adc_sample_blocking()` right now are "ADC counts". You will need to convert those into volts in order to properly interpret the data. The 4096 values represent voltages from 0 to 3.6 volts.

- Test your conversion code by connecting the analog input to VCC and Ground

Briefly connect Microbit breakout pin 2 to the VCC rail or the Ground rail. Sample the ADC voltage for channel 1 and print it out. It should be either 3.3 volts or 0 volts respectively. Try the other rail as well to make sure things are working.

If you aren't reading what you expect, make sure you're connecting to breakout pin 2 and reading channel 1. Then double check your conversion code. You can also print out the raw ADC counts, which should change between about 3700 and 0 with respect to the VCC and Ground rails.

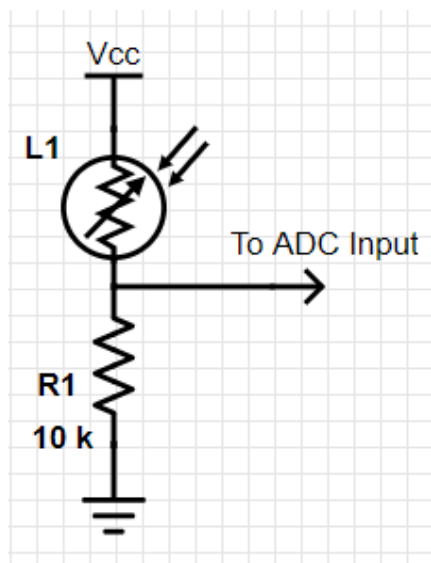
- **No checkoff:** continue to the next step

6. Read the light sensor

- Wire up the [light sensor](#) in a resistor divider

It should form a resistor divider with a 10 k Ω resistor.

- Connect one end of the light sensor to the VCC rail on the breadboard. Doesn't matter which end, the component is reversible.
- Connect the other end of the light sensor to a row on the breadboard.
- Connect the 10 k Ω resistor from that same row to the Ground rail.
- Connect a jumper wire from that same row to pin 2 on the Microbit breakout.
- The schematic for what you are connecting looks like this:



- Create a helper function to convert measured voltage into light states

Unfortunately, an exact formula to convert between resistance and lux seems to be difficult to calibrate. Instead, you should determine voltage thresholds to distinguish between at least three states: dark, medium-bright, and very-bright. You'll have to do so manually with a little experimentation. You can decide how to define each state.

- Test that you can distinguish between states on the light sensor
- **No checkoff:** continue to the next step

7. Read the temperature sensor

- Wire up the [temperature sensor](#) in a resistor divider

The schematic is the same as with the light sensor. Pairing the temperature sensor with a 10 kΩ resistor for the lower half of the divider seems reasonable. Connect this to pin 1 on the breakout (since pin 2 is in use for the light sensor).

The temperature sensor looks like this:



- Create a helper function to convert measured voltage into temperature

This will take two steps: converting voltage to resistance (based on your resistor divider) and then converting resistance to temperature.

- Convert voltage to resistance. You can use a traditional voltage divider equation to do so. [Details on resistive voltage dividers](#).
- Convert resistance to temperature. While Adafruit has a [table of calibrated temperatures and resistances](#), encoding that in software would be lame.

Instead, we can use a conversion equation for [an NTC thermistor](#).

$$T = \frac{B}{\ln(R/r_{\infty})}. \quad \text{where } r_{\infty} = R_0 e^{-B/T_0}.$$

For our thermistor:

B = 3950.0

R = measured resistance

R₀ = 10000.0

T₀ = 298.15

T = resulting temperature in degrees Kelvin

You can use the [log\(\)](#) and [exp\(\)](#) functions from `math.h` to calculate the natural log and exponential respectively. Make sure all the numbers you are using are

floats so the math doesn't truncate. Include a decimal point in each to be sure. Be sure to convert back into Celcius. Note that R_{∞} could be calculated in advance if you prefer.

- Test that you can accurately measure temperature.

Room temperature is usually around 25 C. You can hold the thermistor between your fingers to increase the temperature.

- **No checkoff:** continue to the next step

8. Control the RGB LED with various inputs

- The red channel of the RGB LED should be controlled by the light sensor

The microcontroller should read the analog value of the light sensor, and should use that value to determine whether the LED should be on or off. You can determine whatever light state you want for switching the LED on and off, as long as it is straightforward to demonstrate switching between them.

You can sample and change LED states at whatever rate you want (once per second, ten times per second, etc.).

- The green channel of the RGB LED should be controlled by the temperature sensor

Same as the light sensor. You can choose whatever temperature value you want for switching. Be careful to choose something that's easy to demonstrate with the heat from your fingers. It's okay if it doesn't change back to the original state right away.

- The blue channel of the RGB LED should be controlled by a switch

The switch should be read by the microcontroller as a digital input. Then based on that input the LED state should be changed. The switch does not need to immediately change the LED's state. Sampling it at whatever rate you sample the light and temperature sensors is sufficient.

- Once per second, print out the measurements from each sensor as well as the state of the switch

Temperature measurements should be printed in units of and degrees C. Light states should be printed as human-understandable strings (something like "Bright", "Dark", etc.)

- **Checkoff:** demonstrate this application and your code to course staff

Lab 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.

- Explain your application code to course staff
- Show your breadboard to the course staff
- Demonstrate your application

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