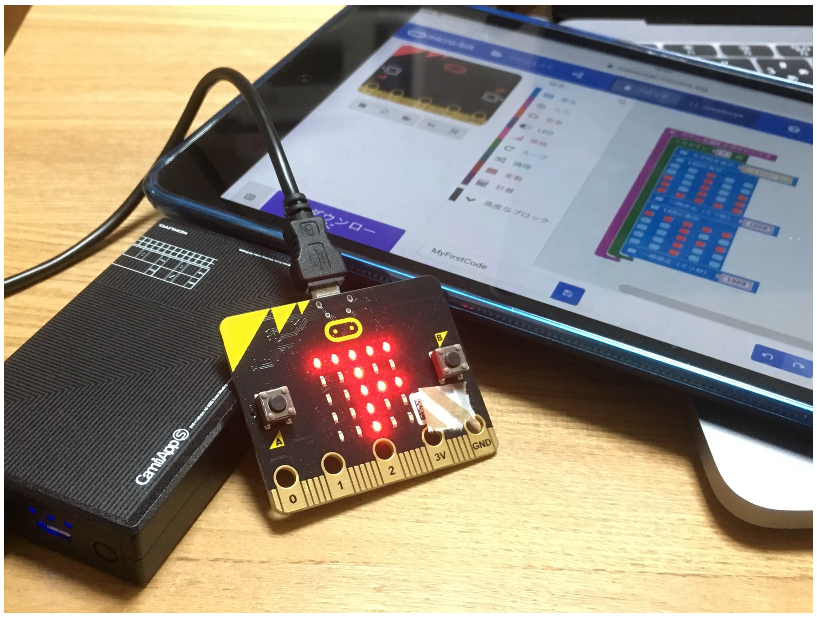
Laboratory 9: Computer Hardware

Bryan Ruddy

20 May 2022

In this lab, you will explore writing a simple device driver for the micro:bit. Your device driver should allow the user to turn on any one pixel, based on its row and column location, with all other pixels turned off.


The micro:bit display. (Image by Tatsuo Yamashita, from `https://www.flickr.com/photos/yto/39026283615`.)

Note: This lab is expected to take most pairs 2-3 hours to complete. You will therefore be likely to need to do some of the tasks on your own, outside the scheduled lab session. Thus, please make sure you try all of the exercises during the lab session, so that you have the TAs and instructors available to help. You will be best served by waiting to work on answering the questions until you have finished all of the exercises. Also consider skipping asterisk-marked steps and questions that are potentially time-consuming, and doing them on your own time instead. **Remember to return to and complete these tasks after the lab session.**

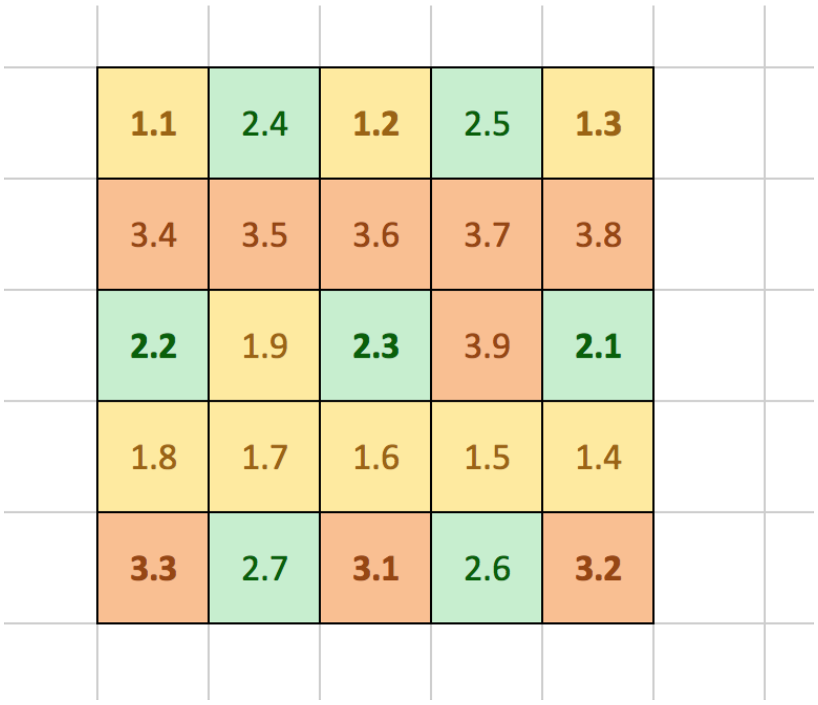
Getting started

There are some aspects of this lab that are slightly different if you are using a simulator rather than a physical micro:bit. We have provided different template code for each case, in the `on_hardware` folder for those using physical devices and in the `simulator` folder for those using the simulator. **Please note that code written for the simulator will not run on the physical device, and vice versa.**

If you are writing code to run on the physical micro:bit, we recommend the official online editor at `https://python.microbit.org/v/2`. If you are using the simulator, you can use any text editor of your choice (such as PyCharm) to write your code. When you are ready to simulate, open a terminal or command prompt (such as the Anaconda Prompt) in the `simulator` folder, then run this command: `python -m http.server --bind localhost` Leave the terminal open, then point your web browser at `http://localhost:8000/`. If you want to save any edits you have made directly in the simulator, there is a "Save" button you can use. Be warned that the file is immediately downloaded with a random filename, so you will need to move and rename it as desired.

Exercise 1: Understanding the hardware

Your micro:bit has a display made up of 25 red LEDs, arranged in a 5 x 5 square grid. If each LED was individually controlled, this would require 25 pins on the microcontroller; however, the microcontroller only has 16! To manage this, the micro:bit uses a technique called *multiplexing*: each LED is connected to two pins, and can only be lit if one pin outputs a high voltage ("power") and the other outputs a low voltage ("ground"). Each pin is then connected to multiple LEDs. The micro:bit uses 3 pins for power and 9 pins for ground, with the pixels assigned as shown below.



1.1	2.4	1.2	2.5	1.3
3.4	3.5	3.6	3.7	3.8
2.2	1.9	2.3	3.9	2.1
1.8	1.7	1.6	1.5	1.4
3.3	2.7	3.1	2.6	3.2

The mapping between pins and display pixels. (Image from `https://lancaster-university.github.io/microbit-docs/ubit/display/`.)

Multiplexing allows each LED to be chosen individually; however, problems arise if you wish to turn on multiple LEDs. To have full control, only one power pin may be turned on at a time. Otherwise, each activated ground pin would turn on two or more LEDs. To show the whole display, the micro:bit must rapidly cycle through the three power pins to create the illusion of continuous illumination.

The following procedure need only be followed if you are working on a physical micro:bit. If you don't have access to one, a video will be provided on Canvas to help you answer the questions.

1. Plug in your micro:bit, load the online editor, and use the *Connect* button to connect the editor to your device.

2. Use the *Open Serial* button to gain access to the REPL, which behaves similarly to Jupyter Notebook and allows you to enter and run code one line at a time.

3. Use the command `import microbit` in the REPL, and then display a character on the LEDs using `microbit.display.show()`, preferably one that uses a number of pixels across the display.

4. Shake the micro:bit back and forth very quickly while looking at the display. Pay attention to what you see each illuminted pixel do as it moves. What do you see? Are the LEDs always on?

Questions

1. What do you see when you shake the micro:bit? Is this consistent with what you would expect to see?

2. *What is the minimum number of pins needed to control a 5 x 5 grid of LEDs?

3. *Why do you think the designers instead used 12 pins?

Exercise 2: Memory-mapped I/O

As discussed in this week's lectures, one common way input and output devices are accessed in a computer system is through *memory mapping*. A memory-mapped device is associated with one or more special addresses in memory, called registers; instead of storing information, these registers cause an output action when data is written, and/or provide input data from the world when read.

Using a memory-mapped device requires understanding the hardware system through reading its datasheets. Those of you in BME will gain experience doing this duirng BIOMENG 241. However, rather than ask you to read a 115-page datasheet and a 174-page reference manual, we provide the key memory addresses for controlling the LEDs to the right.

Address	Function
0x50000504	OUT
0x50000508	OUT_SET
0x5000050C	OUT_CLEAR

At each memory address, a 16-bit number is used to represent the pins of the microcontroller, with each bit corresponding to a pin. Each of these 3 registers controls the output in a different way, however. Setting the OUT register equal to a particular value changes the state of every pin to match the bits in that value. Setting the OUT_SET register equal to a value causes the pins in the "1" locations to turn on, but doesn't change the others. Setting the OUT_CLEAR register causes the pins in the "1" locations to turn off, but doesn't change the others.

The pins are used as follows: pins 13-15 are used for power, while pins 4-12 are used for ground. The other pins (0-3) have different functions, and **should not be changed** unless directed. The following table shows how the bits in these registers correspond to physical pins on the microcontroller, and how these correspond to the pin functions described in the previous exercise.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	b
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Pin number
P3	P2	P1	G9	G8	G7	G6	G5	G4	G3	G2	G1	XX	XX	XX	XX	Pin function

Top: a 16-bit number.
Middle: Physical pin number.
Bottom: Pin function, P = power pin, G = ground pin, XX = not part of display.

Ordinarily, Python does not provide easy access to memory addresses. However, because the version on the micro:bit was designed specifically for embedded systems, it has a special set of arrays in the 'machine' module that represent the memory. The command `machine.mem16[0xDEADBEEF] = 0xBEAD` sets the contents of memory at address `0xDEADBEEF` to the 16-bit value `0xBEAD`. (There are corresponding `mem8` and `mem32` commands for 8-bit and 32-bit values, but you don't need them for this lab.)

With this introductory material out of the way, please proceed with the following steps to explore memory-mapped I/O on the micro:bit. On the physical device, you can execute each line of code one at a time on the REPL. On the simulator, we recommend you build a program one line at a time, running after each is added to get a similar effect.

1. Import the `machine` module.
2. Disable the normal display driver with `microbit.display.off()`.
3. Turn on all the pixels by setting the OUT register equal to `0xE000`.
4. Shake the device back and forth again, and compare to what you saw previously. (Skip this step if simulating.)
5. Turn off power pin 2, by writing `0x4000` to the OUT_CLEAR register.
6. Turn off ground pins 1-3 and 9, by writing `0x1070` to the OUT_SET register. What do you see?
7. *From this state, choose values to write to the OUT_SET and OUT_CLEAR registers to give a display with a grid of 9 lit LEDs, like this:

   Note: To return to this point quickly if you make a mistake, just set the OUT register to `0xB070`.

```
X.X.X
.....
X.X.X
.....
X.X.X
```

Questions

1. What did you observe when you shook the device this time? Why was it different?
2. What shape was created in step 6?

3. *What values did you need in step 7 to change the display? If you instead wished to directly set the OUT register, what would you need instead?
4. *Why might you prefer to use OUT_SET and OUT_CLEAR instead of just writing to the OUT register?
5. *Can you make any arbitrary pattern of lit pixels this way? Why or why not?

Exercise 3: Writing a driver

Now it's time to take what you have learned from your experiments and implement a device driver. You will need to implement three functions for your display device: turn one specific pixel on with all others off, turn all pixels off, and turn all pixels on. The code template `hwlab_driver.py` contains a test for the first function, but you will need to implement a test for the other two. Be sure to write good code comments, as they will form part of your assessment.

In addition to the template files tailored to the physical micro:bit and to the simulator, we have also provided a testing suite in the `testing` folder, suitable for use with one of the most common Python testing tools, pytest (`https://pytest.org`). It is not mandatory that you use this, but this suite is very similar to what we will use to assess the function of your code for grading purposes.

1. Implement the function `display_pixel`. Note that you will need to convert from the physical row and column on the 5x5 display (use row and column numbers ranging from 0 to 4, with (0,0) in the top left corner) to the power and ground pins needed to drive the pixel. You may use either OUT or OUT_SET and OUT_CLEAR; whichever method you use, make sure you **do not change the values of pins 0 to 3**. (In exercise 2, we did not worry about this constraint.) You can test your code by flashing it to the micro:bit or by running it on the simulator; you should see a pixel move from the centre towards the top left corner.
2. *Implement the functions `clear_display` and `illuminate_display`.
3. *Implement a basic test for these two functions in the `main` function. (Not a formal testing function, just a bit of code to demonstrate your driver works.) You should have the device begin with a cleared screen, then turn on the centre pixel, then turn on all pixels, then go blank.

Exercise 4: Using a driver

Your driver is most useful if it forms a module that can be imported by other code. In Python, this is done by placing the file in an appropriate location, and then `import`-ing it from other programs. Here, we will put it in the right location on the micro:bit, and use it in a simple user program.

1. Complete the code in `hwlab_program.py`, so that it imports your driver and calls the appropriate function. Make sure you youse the correct version of the template.
2. If you are running the simulator, ensure your completed `hwlab_driver.py` is in the `simulator` folder, at the top level with `index.html`. Load your code in the simulator and it should work.
3. If you are using the physical micro:bit, try flashing your program code to the device. The code should fail with an error, because it cannot find the file being imported.
4. Use the *Load/Save* button and click "Show Files" to view the file storage system on the micro:bit. Click the *Add file* button to load your completed driver onto the device.
5. Reset the micro:bit; your program should now work.

Note: If you re-flash the code, the driver file will be erased from the micro:bit, and you will need to repeat step 4.

Questions

1. What does this program do on the display?
2. *Comment on the relative difficulty of writing this program compared to writing the driver code. How much work could you save future engineers by sharing your driver code?
3. *The micro:bit foundation has recently released a new version of the device, with a different memory mapping and different numbers and connections of pins controlling the display. Discuss the effect this would have on application software developers, and the effect it would have on device driver developers.

Submission Instructions

For this lab, you should submit the following:

- `hwlab_driver.py`
- `hwlab_program.py`
- A single PDF file containing your answers to the questions in exercises 1, 2, and 4.

You do **not** need to submit any other files. Do **not** change the file names for your code from what was provided; however, don't worry if Canvas appends a -1 or -2 to the end.

Remember, all submissions are compared against each other and those from previous years. Copying someone else's code and changing the variable names constitutes academic misconduct by *both* parties. Because you are working in pairs, it is okay if your submission is very similar to your partner's.