The Microbit Microcontroller

Aims of this experiment

- Explore the capabilities of a modern microcontroller and some peripheral devices.
- Understand how the signals at the various interfaces work with the software.
- Make a simple measuring instrument using a microcontroller.

The early parts of this experiment are fairly prescriptive, to get you started. The later parts provide scope for you to devise your own microcontroller application. There is no need to do all the later parts in equal detail, or slavishly follow the manual. Quality beats quantity.

Introduction

Microcontrollers are essentially single-chip computers. They are particularly useful for processing electrical signals, with which they can connect quite simply. They can simplify electronic circuits in many applications, because the hardware is fixed, and a new application requires only software changes.

A very small microcontroller, costing only pennies, might have the processing capacity to control a complex multi-way light switch. Capabilities and prices range upwards by perhaps 100 fold; beyond that lie smartphones and PCs. But the unique feature of microcontrollers is that they are intended for use in embedded systems, in which they behave as a sophisticated electronic component, rather than providing general-purpose computing.

The microcontroller in this experiment is a Nordic Semiconductor nRF51822 device, which is based on the ubiquitous 32-bit ARM processor. It has 16 kilobytes (K) of working memory, 256 K of permanent (flash) memory, and runs at 16 MHz. By microcontroller standards, these are mid-range numbers. The advantage of this level of device is that it can run a specialised version of the high-level language Python in real time. This is a great convenience for experimental development. [1]

The exercises in the experiment introduce some of the ways the microcontroller interacts with other electronics, and the kind of devices you can build by programming. This should be useful to you in future experimental work (e.g. Year 2 projects, group studies...)

1 The development environment

The "Micro:bit" board was developed as an educational device. We use it here because it is the least complicated hardware that supports Python, and we will assume you remember some Python programming and electronics from Yr 1. In case you need more information about the board, its functions are documented at
https://microbit-micropython.readthedocs.io/en/latest/microbit_micropython_api.html [2]

Plug the board into a USB socket on the PC, using a cable that supports data, not just power. Do not

[1] The USB interface is managed by a second ARM controller, illustrating the utility of small microcontrollers

[2] You will see a lot of schools-level material related to the Micro:bit on the web, but at a lower level than in this lab

access the board as a storage device, but instead use the Chrome web browser to access the Python editor at **`https://python.microbit.org/v/2.0`** The **`Connect`** button in the editor should allow you to select and connect to the micro:bit.

If you have a new board, click the **`Flash`** button to write the example program to the board. This will take a few seconds, and will set up the board for further use.

Use the **`Open Serial`** button to communicate with the Python command line on the board. You may have to click the **`Send CTRL-C for REPL`** button (or even the CTRL-D and CTRL-C) to get the Micropython prompt, **`>>>`**. [3] Note that these are buttons to click with your mouse on the browser page, not keys on your keyboard. Test it by typing something like **`print(2*3)`**. You could type a whole program here, but it is more convenient to edit and save it using the editor.

Switch from the command line to the editor using the **`Close Serial`** button. Delete any existing text and enter a simple Python program, for example

```
for i in range(10):
    print('{} * 5 = {}'.format(i, i*5))
```

Use the **`Flash`** button to transfer your program to the flash (permanent) memory on the board. The program will immediately run *on the microcontroller board*, while the PC just serves as a communications terminal. Use **`Open Serial`** to take a look at its output. The program remains on the board (until you re-flash it), so the **`CTRL-D`** reset option, which restarts Python on the board, will make it run again, as will pressing the physical reset button on the board, close to the USB connector.

Try saving and loading your program on the PC. Chrome treats saving as downloading a file, so **`Load/Save`**; Download Python Script; keep the file. You can give the download a sensible name using the **`Script Name`** field in the editor. To restore a saved file, locate it on the PC and drag & drop it to the **`Load`** panel.

2 Controlling an output

2.1 Flashing LED

As with any embedded project, the first thing to get working is turning on and off a light-emitting diode (LED). Connect the LED/speaker circuit with crocodile clips to the GND and output 0 holes on the board (black=GND). Run the program flash1.py on the board. The LED should flash with a 1 second period.

flash1.py

```
import microbit as mb
from utime import sleep_ms

while True:
    mb.pin0.write_digital(0)
    sleep_ms(500)
```

[3]REPL = Read, evaluate, print loop. i.e. the software behind the command line

```
    mb.pin0.write_digital(1)
    sleep_ms(500)
```

All our programs will import from the **microbit** module, because it gives access to the hardware features of the board. **mb.pin0** is the object that represents a physical connection point, and its **write_digital** method sets the voltage appearing there: $0 \Rightarrow 0$ V and $1 \Rightarrow 3.3$ V. The **sleep_ms** function introduces a delay specified in milliseconds

You should be able to work out how the code gives rise to the flashes of the LED. Check your understanding by altering the delays. Why are two delays necessary? Document your findings. **Q**

2.2 Faster flashing

Increase the flashing rate by decreasing the delays. Eventually the LED will appear to be lit continuously. Why is this? At what flashing frequency is the flashing no longer invisible? At this kind of frequency, **Q** the clicks from the speaker will merge into an audible tone. You may want to use the CRTL-C button in the command line interface to stop it.

2.3 Fourier analysis

If you install an audio spectrum analysier on your phone or PC, you can use it to get an independent measurement of the tone being produced by the speaker. This example uses FFTWave, but many are available. Record the clearest analysis you can, see the screenshot in Fig. 1, and identify the main frequency components.

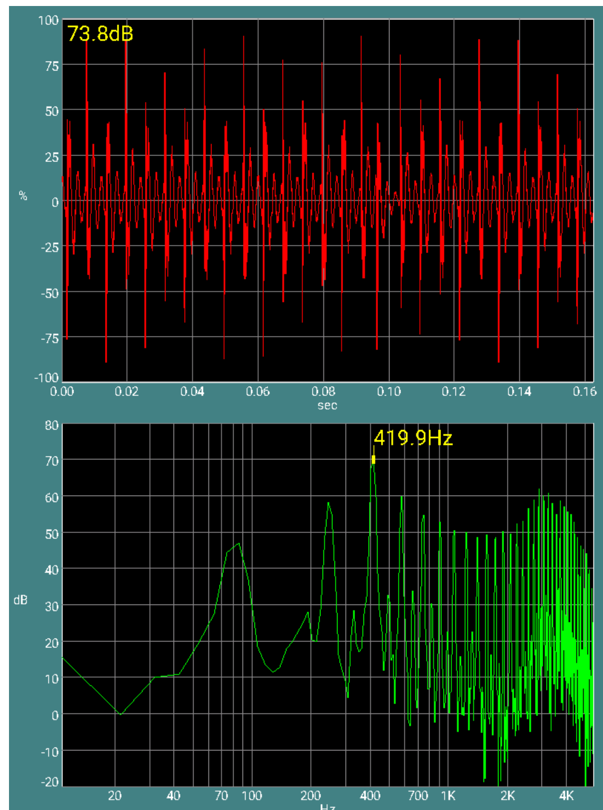

Figure 1: Fourier analysis of speaker signal. Sample rate = 11025 Hz, 1024 points, DC coupling, logarithmic axes.

The tone produced by the little speaker is very distorted (the red time trace is nothing like a sine or square wave), but this does not change the underlying frequency. The green trace is a Fourier series analysis, in which each peak shows the amplitude of the component at the corresponding frequency. The lowest peak, at about 85 Hz, is consistent with the timing of the Python code, and the many *odd* multiples show that whatever the wave shape is, the positive and negative half-cycles are distorted in the same way.

2.4 Firmware tone generation

If you want good audible tones, then some better functions are built into the Micropython firmware. They are not an essential part of this experiment, but you could use them in the later parts if necessary. e.g.

Playing with the music module on the command line

```
import music
music.pitch(440, 200)    # Hz, ms
music.play(music.ENTERTAINER)
```

3 Speed of Micropython statements

3.1 Fastest loop

To see the best speed of the flash1.py code, try removing the sleep_ms statements completely. Can you estimate the time taken by a statement in this case? **Q**

The code in flash2 is faster, and that in flash3 faster still. Try these out, try to measure their speeds as before. Can you see how time is saved?

flash2.py

```
import microbit as mb
wd = mb.pin0.write_digital
while True:
    wd(0)
    wd(1)
```

flash3.py

```
import microbit as mb
def fn():
    wd = mb.pin0.write_digital
    while True:
        wd(0)
        wd(1)
fn()
```

4 Microbit hardware

4.1 Dice

The board has a built-in LED display and pushbuttons. **`mb.display.show()`** is a function that will display digits, amongst other things, while **`mb.button_b.is_pressed()`** is a function that returns True or False, depending on the state of the right-hand button. There is a similar function for button A, on the left. The program die1.py works as a simple random number generator that displays an integer between 1 and 6 after each press of button B.

die1.py

```
import microbit as mb

i = 0

while True:
    i += 1
    if i > 6:
        i = 1
    if mb.button_b.is_pressed():
        mb.display.show(i)
```

Try this program out and explain how it works. Why are the displayed digits unpredictable?

4.2 Encoding an image

`mb.display.show()` will also display user-defined images which are constructed from five rows of five digits representing the brightness (0-9) of each LED, e.g.
`mb.Image('00000:00000:00900:00000:00000')` lights the centre LED, and
`mb.Image('90000:00000:00000:00000:00009')` lights two diagonal corners.

The program die2.py uses an array to translate the dice result into the canonical dice pattern. Only the results 1 and 2 have been encoded, while the rest are displayed as question marks. Run this program and complete it so that all the dice patterns are shown correctly. Verify it works, and explain how the translation from integer to pattern is done.

die2.py

```
import microbit as mb

i = 0
images = [
    '?',
    mb.Image('00000:00000:00900:00000:00000'),
    mb.Image('90000:00000:00000:00000:00009'),
    '?',
    '?',
    '?',
    '?'
]

while True:
    i += 1
    if i > 6:
        i = 1
    if mb.button_b.is_pressed():
        mb.display.show(images[i])
```

5 Reaction timer

react1.py is a simple reaction timer that makes the user wait for a short random time while a diamond is displayed, then it starts progressively lighting LEDs. The user has to stop the LEDs lighting as soon as possible by pressing button B. The number of LEDs that get illuminated gives the user's reaction time.

react1.py

```
import microbit as mb
import utime
from random import randint

while True:
    mb.display.show(mb.Image.TARGET)
    delaytime = randint(1000, 6000)
    utime.sleep_ms(delaytime)         # Keep the user waiting...

    mb.display.clear()
```

```
    for i in range(25):                # Start the clock...
        mb.display.set_pixel(i%5, i//5, 9)
        utime.sleep_ms(20)
        if mb.button_b.is_pressed():
            print('reaction time = {} ms'.format(i*20))
            break
    utime.sleep_ms(3000)
```

Look at the code in **react1.py** and answer the following questions, in terms of the code used: **Q**

- How long is the diamond (the pre-defined TARGET image) displayed for?
- What does the break statement do here?
- What happens if the switch is never pressed?
- set_pixel(x,y,9) turns on the LED at location (x,y). What do the arguments (i%5, i//5) do?

Try measuring your reaction time. What is the resolution of the measurement (from the program)?

You might notice that you can cheat and get a reaction time of one LED. Improve the program so that this is not possible, and explain how your fix works.

6 Sensors

The board contains some simple sensors, accessed via the microbit module: temperature(), accelerometer.xxx, display.read_light_level() and compass.xxx (magnetic field). In this part of the experiment, you should develop your own device or instrument based on one of these. What follows are some suggestions.

6.1 Accelerometer

accel1.py sends the reading of one axis of the accelerometer over the serial connection. Run this program and use the Open Serial editor button to see its output.

accel1.py

```
import microbit as mb
import utime

while True:
    ax, ay, az = mb.accelerometer.get_values()
    print(ax)
    utime.sleep_ms(100)
```

The sensor units are 10^{-3} g [g is the acceleration due to gravity], so by rotating about the correct axis, you should get readings varying roughly from +1000 to -1000.

Modify the code to make your own accelerometer instrument. One example would be an inclinometer, which measures the angle at which the device is held with respect to gravity. You will need to experiment to find which directions on the board correspond to the x, y and z components of acceleration, then pick two and use the Python arctangent function, **math.atan2()**.

You could alternatively make a digital spirit level, in which the position of the illuminated LED depends on the angle that the board is held at.

6.2 Magnetometer

You can make a magnetic object detector using the output of `mb.compass.get_field_strength()` to control the display. Take care to not make short-circuits on the board with metal test objects. Fridge magnets and scissors work well.

7 Checklist

Your lab book should contain:

- Diagrams of circuits you built
- Programs (or parts of programs) that you wrote or understood, with comments showing how they work – but there is no need for the full text of programs that are supplied to you.
- Observations and explanations as prompted by this manual.
- Enough detail to enable someone to quickly pick up the thread of what you did and what you found out.

MSC 2020-10-05