Internet of Things

Practical Week 1

Module Name: Internet of Things

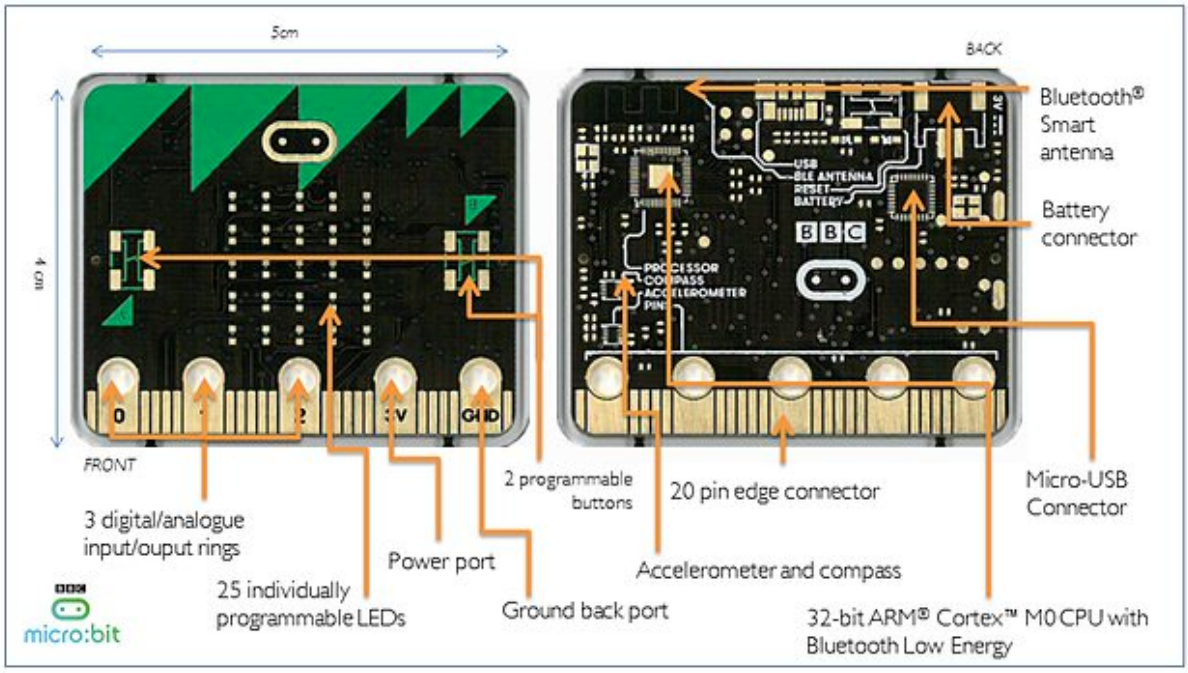
Module Code: UFCFVK-15-2

Module Leader Name: Benedict R. Gaster

UWE Bristol | University of the West of England

Introduction

Through the module practicals and corresponding assessement will be using the BBC Micro:bit, pictured below:



The micro:bit is a Internet of Things development board designed for teaching and development. It is a computer that fits in your hand and runs on batteries. As can be seen from the above image it includes a number of external interfaces, for example, buttons, I/O pins, and an accelerometer, along with Low energy Bluetooth for wireless communication and a LED matrix for display.

The microprocessor is Nordic Semiconductor nRF51822 Bluetooth chip which includes an ARM Cortex-M0. Unlike the ARM processors found in the billions of smartphones, called an application processor, the Cortex-M0 is a microcontroller, which is very small computer, that runs at only 16 Mhz, uses tiny amounts of power, and lacks many of the advance features of its bigger cousins. That being said it is still an incredibly powerful computer, capable of doing considerable computation, at a fraction of the cost of ARM's application processors, while consuming only a fraction of the power.

The Nordic chip can be found in numerous Bluetooth applications, e.g. ROLI's Block MIDI controller, Fitbit's Heartrate bands, Pebble's smartwatch. These applications all have the ring of the Internet of Things, small low-power devices, good at one particular application, often connected to other devices, e.g. the smartphone, and in many cases, connected, by extension, to the internet.

In general, we refer to devices like the BBC micro:bit as an embedded device. An embedded system is a computer system with a dedicated function, as per the

examples above, unlike general purpose computing, such as an Apple laptop or Android smartphone, which we expect to be cabable of doing many different tasks.

For many of you this will be the first time interacting and developing for an embedded device and is one of the reasons the BBC micro:bit was chosen. The device has excellent support infrastructure, including a selection of introduction programming languages, such as the Javescript Blocks, that provide a simple online programming experience:

http://microbit.org/code/

While these tools are excellent and a great way to started with programming and the Internet of Things, however, they come with limitations. By design embedded devices are resource constrained, the micro:bit runs at only 16 Mhz and only 256 KB flash memory, 16 KB static ram, and thus careful use of these resources is critial. It 16k of ram is small, it would be difficult to fit an image of any reasonable size, unlikely it could fit the trained data set of a neural network. Given limited resouces we need to consider carefully how the system is programmed, so while Javascript is an amazing language, and provided a excellent platform for programming in the web browser, it is less likely to a good choice for a resource constrained microcontroller, at least beyond teaching. To get the most out of a microcontroller, such as the micro:bit, a programming language that is close to the machine itself, allows fine grain control of the machines resouces, and so on leads to the choice of a system programming language, such as C/C++ or Rust.

For our use we will use C/C++, based on ARM's mbed toolset and the The micro:bit Divice Abstraction Layer (DAL) that is s built on top the build system yotta, to enable offline development.

When using yotta to build micro:bit projects there are currently two supported toolchains:

- GCC
- ARMCC

For this course we will use GNU GCC, however, unlike the GNU GCC tools used in the Operating Systems module here we must cross compile, i.e. the GNU GCC tools will run on x86 Linux but generate binaries for ARM Cortex microprocessors. Additionally, all development will be supported using Linux, on the provided VM.

The remainder of this document describes how to get a "hello world" application running on the BBC micro:bit using the provided VM.

Linux Virtual Machine

As noted in the previous section all development will be via Linux and a C++ environment, which we have prepared a VM for VMWare. The course VM can be found here:

https://tinyurl.com/ya59un4m

There is a single account:

Userid: student

Password: programming123

Please change password on first login.

Each student is provided with an external disc drive. If you have not already been allocated one, then you can check one out from the Project room. To ease the process of picking the drive up please pre-reserve one here[1]:

https://connect2.uwe.ac.uk/Connect2_FET_BNE/

Please pickup your disk, if necessary, and also download the VM before coming to your first practical next week.

If you want to run the VM on your own machine, then you can get a free license for VMWare from the UWE software for home site (updated to zip file):

https://tinyurl.com/y75uvj74

Hello World on Micro:bit

During the practicals BBC micro:bits will be on hand, however, we do not have enough for students to take them away. In a few weeks the project room will have a number of micro:bits that can be borrowed for two days. The micro:bits are reasonably cheap and for those who would like to own one, then they can be purchased from Amazon (for example):

https://tinyurl.com/ycs96p9e

[1] You might only be able to reserve a disk for 1 day, don't worry this will be address by project room staff when you pick up the drive.

For the duration of this module all accessement will be via gitlab.uwe.ac.uk and Blackboard, and there is starting repo that contains the "hello world" program:

https://gitlab.uwe.ac.uk/br-gaster/ufcfvk-15-2-iot

Boot up your Linux VM, open a terminal, and clone the above repo:

```
git clone https://gitlab.uwe.ac.uk/br-gaster/ufcfvk-15-2-iot
```

change into the directory **ufcfvk-15-2-iot** and if you do **ls** will see a single directory **examples**[2]. This directory, currently, contains only a single "hello world" example. Change into the directory **examples**. This itself contains:

```
module.json
source
```

The later of which is a directory containing main.cpp, with the code3:

```
#include "MicroBit.h"

MicroBit uBit;

int main()
{
    uBit.init();

    uBit.display.scroll("HELLO WORLD!");

    release_fiber();
}
```

The file module.json contains a description of how yotta should build the hello world application:

```
{
  "name": "iot-example",
  "version": "1.0.0",
  "description": "IoT  micro:bit examples.",
  "license": "MIT",
  "dependencies": {
    "microbit": "lancaster-university/microbit"
  },
  "targetDependencies": {},
  "bin": "./source"
}
```

[2] Plus .gitignore is you do **ls -a**.
[3] Comments removed.

We will look at this file format in more detail later in the course for now we need only note that source code (i.e. **main.cpp**) is located in the directory **source**, as indicated by the "bin" field.

Before building the application we must set the build target, in our case the micro:bit. A yotta target contains the information required by yotta in order to build a project for a specific combination of hardware. This includes the type of compiler. To set the target run the following command:

```
yt target bbc-microbit-classic-gcc
```

Now build the project with the command:

```
yt build
```

Once the build has completed the resulting executable will be in the directory:

```
build/bbc-microbit-classic-gcc/source/
```

The executable is found in the hex file:

```
iot-example-combined.hex
```

To run the executable on the micro:bit first plug in the decive using the provided micro USB, which will mount the device at:

```
/media/student/MICROBIT
```

To upload the executable simply copy the hex file to the device:

```
cp build/bbc-microbit-classic-gcc/source/iot-example-combined.hex/media/student/MICROBIT
```

Once upload the device will reboot and display the message `"HELLO WORLD!"` scrolling across the LED matrix.

How did this work? Take a look at the code in main.cpp, you should see the standard main function, along with calls to functions `uBit.init` and `uBit.display.scroll`, which initializies the micro:bit runtime and displays a text message on the LED display, respectively.

Next steps

As can be seen from the example in the previous section the micro:bit comes with a powerful runtime. This runtime abstracts out the low-level details of the ARM Cortex. It is possible to bypass this and go directly to the machine itself, but for this course we will focus on the runtime developed by the University of Lancaster, which is documented here:

https://lancaster-university.github.io/microbit-docs/

We will be using this runtime extensively and it recommended that you familiarise yourself with the details. To begin you should read the concepts:

https://lancaster-university.github.io/microbit-docs/concepts/

Having read this try and write another example, maybe use one of the buttons for input. This time you can create your own directory, e.g. **myexample**, create a source directory and add a main.cpp, and copy over the module.json from examples to **myexample**. At this point you should be able to run the yotta target and build commands to build the executable and copy the resulting file to the micro:bit.

Once you feel happy with the basic development workflow take a look at:

https://lancaster-university.github.io/microbit-docs/advanced/