Computing for Engineers (EE108)

Laboratory 1– Arduino IDE, EE108 board, and Blink LED

Objectives: This laboratory and assignment aims to introduce the environment and EE108 daughterboard for programming using Arduino C.

Learning outcomes (after completing this lab and assignment you should be able to do the following):

- Connect the Arduino board to the PC, configure the Arduino environment, and open existing example sketches
- Create new sketches for the board using Arduino C
- Write various simple Arduino sketches to blink LEDs or output text to console

Lab instructions

1. The lab is for individuals and must be completed during the lab session
2. Create a new sketch for each major section in the lab.
3. Before you leave the lab, call the lab demonstrator to check what you have done for all the sections (this is why separate sketches are important). *Anything not checked by the demonstrator during lab will have to be assigned zero marks.*
4. Create a single plain text submission file (.txt) for the lab using a plain text editor (e.g. NotePad++). Copy all the sketches you write and any other answers required for the lab into the submission text file. *Name the file "108_Lab1_firstname_surname.txt", using your actual name. Include your name and lab number at the top of the submission file also and clearly label everything in the file. Unclear submissions will have to be marked down or (in the worst case) not marked at all.*

Marking for lab

Most of the lab and assignment will be marked during lab sessions. It is essential that you get the demonstrator to confirm your progress during the lab (for the lab portion) and at the start of the next lab (for the assignment portion).

For all code sections, marks will be deducted for bad communication and style (e.g. missing or mismatching comments, poor variable names, bad indentation, inappropriate use of global variables, unnecessary code repetition, etc.) and incorrect behaviour, or failure to follow the requirements of the question.

General marks will also be lost if the submission document instructions are not followed.

1 Set up

- Create a folder for sketches on your home drive called “Z:\EE108\arduino”. We will refer to this location as <*sketchbookfolder*>.
- Open the Arduino integrated development environment (IDE), go to File > preferences and set up the sketchbook to point at the folder you just created.
- Connect the board to the PC and wait for the driver to be found if necessary.

ⓘ *You may need to check that the above settings are correct every time you start Arduino on a new computer or every time you come to lab or do an assignment on the open access computers.*

2 The Blink example

- The remaining steps are highlights from the steps that can be found under Help > Getting started in the Arduino IDE
  - In the Arduino IDE, set Tools > Board to “Arduino/Genuino Uno”, set Tools > Port to the serial port of your Arduino (should be labelled in the list)
  - Load the Blink example sketch (go to File > Examples > 01. Basics > Blink)
  - Compile and upload the example to your board
  - You should see the LED in the middle of the Arduino Uno motherboard (under the daughterboard) blinking on/off once each second

ⓘ *Ask for help if you cannot get this working.*

3 Adding a library and modifying the Blink programme —save sketch as “Lab1_3_BlinkDaughterboard”

(5 marks)

In this portion of the lab you’ll create a modified blink programme which blinks a daughterboard LED instead of an Arduino motherboard LED. You will also adjust the timing of the blinking.

- Create a new sketch. Immediately save it as a new name (e.g. Lab1_3_BlinkDaughterboard) in your sketchbook folder (that you created during setup). [You should always name and save sketches immediately after creating them as using the default named sketches inevitably leads to mistakes.]
- Download the ee108.zip library from moodle and add it to the IDE. Go to Sketch > Include library > add .zip library and browse to the ee108.zip you downloaded. It will be copied to the libraries subfolder of your sketchbook folder

ⓘ *You may need to perform the add library step every time you start Arduino on a new computer or every time you come to lab or do an assignment on the open access computers.*

- The EE108 library provides a number of "header files". Edit your sketch to add the following line just after the comments at the top of the file and before any other code in your file. This makes the constants in the header file available to your programme.

```
#include <ee108.h>
```

- Open the file ee108.h in Notepad++ so that you can see what it contains. You can find it at <*sketchbookfolder*>/libraries/ee108/ee108.h. You can see the constants for bar LED pins. You'll see that it includes a number of other .h (header) files. One of the files it includes, "ee108_constants.h" defines constants for a number of pins used by the EE108 daughterboard. Open up ee108_constants.h and take a look. It is located in the same folder as ee108.h.
- Copy the code from the Arduino blink example (see Section 2) to your sketch.
- Declare 3 new constants, using const (see notes) just after the #include "ee108.h" line you already added to your sketch file.

```
const int FIRST_CONSTANT = BAR_LED_5_PIN; // the pin to use
const int SECOND_CONSTANT = 1; // the on-time in ms during each blink
const int THIRD_CONSTANT = 99; // the off-time in ms during each blink
```

  - Don't use the example constant names above. Give the constants sensible names that make it immediately obvious to a reader what the constant is for. (Remember, use ALL_CAPS_AND_UNDERSCORES naming for constants.)
  - Change the code that actually blinks the LED, the digitalWrite lines, to use your pin constant to specify the pin. Doing this makes it easy for you to change LEDs by just modifying the declaration of your constant. (Look up digitalWrite in the arduino help to see which parameter to replace with your pin constant).
  - Change the appropriate delay(…) lines to use the on-time and off-time constants you've defined so that the LED blinking is as we want it.

Build and upload the programme to ensure that it works correctly. The LED should be blinking visibly, and quickly.

- Now, change the value of the off time (by changing the value of the constant) so that the LED blinks so quickly that it appears to be on constantly and does not flicker. The off time will be something between 15 and 30 ms.

Build and upload the programme to ensure that it works correctly. The LED should now appear to be constantly on, but lighting a bit more dimly than before.

To light the LED even more dimly, we must ensure that it is switched on for even less time in each duty cycle (the repetition of on and off times).

- To get even more control over the on-time declare a new constant to represent the microseconds part of the LED on-time. Change the value of the ms part of the LED on-time to 0 and the microseconds part to 500.
- Add a line of code to delay for the microseconds part of the on-time. The function you need is called delayMicroseconds and you should call it with the microseconds constant you defined. See Help > Reference, delayMicroseconds for more information.
- Reduce the microseconds part of the on-time until the LED is very dim but still visible and not flickering.

Build and upload the programme once more to ensure that it works correctly.

✏ Copy the sketch into your answer document (ensure to put a clear label/heading above it) and demonstrate it to a lab demonstrator.

4 Using serial output and variable scope

(5 marks)

In this final portion of the lab, you'll develop a programme to print the time in milliseconds (since the Arduino board was last reset) and the value of a global, static, and local variable to serial output, once every 2 seconds.

You will use `Serial.print` and `Serial.println` (see Help > Reference > Serial). Basically each call to `Serial.print` or `Serial.println` can only print a single number, character, or text string. `Serial.println` includes a new line where `Serial.print` does not, so normally a bunch of `Serial.print` calls followed by a `Serial.println` are used to print a single line of text.

- Create a new sketch, rename it appropriately and save to your sketchbook. The following example sketch may be a helpful starting point. Ensure you add in/modify the comment at the top of the file to explain the purpose of the sketch.

```
/**
 * TODO: explanation of this file goes here...
 */

const unsigned long SUPERLOOP_MS = 2000;

void setup() {
    Serial.begin(9600);
}


void loop() {
    Serial.println("hello");

    delay(SUPERLOOP_MS);
}
```

- You will be using a combination of Serial.print and Serial.println statements to provide output similar to the following, once every 2000 ms:

```
24388 ms: gYourGlobalName = 12, someStaticLocal = 12, someLocal = 1
```

- First focus on the the time in milliseconds. You can call the millis function to get this time:

```
24013 ms
```

- Use a global variable to keep track of the loop number.
  - Declare a new global variable to hold the loop number and initialize it to zero. (What data type should it be? Can it ever be negative? Give it a self-explanatory name. Prefix the name of the global variable with a 'g' to indicate that it is global and remember our lowerMixedCase style.)
  - Each time the loop function runs, add 1 to the global variable. Modify/extend your existing print statements to display text matching the following format (but not the example values or names):

```
24013 ms: gYourGlobalName = 12
```

- Now add a static local variable inside the loop function to keep track of the loop number also. You should find that the static local can remember its value between executions of the loop function, just like the global variable, so both variables should end up with the same values.
  - Declare a new static local variable to hold the loop number and initialize it to zero. (What data type should it be? Can it ever be negative? Give it a self-explanatory name and remember our lowerMixedCase style.)
  - Each time the loop function runs, add 1 to the static local variable also. Modify/extend your print statements to display text matching the following format (but not these example values or names):

```
24047 ms: gYourGlobalName = 12, someStatic = 12
```

- Now add an automatic (ordinary) local variable inside the loop function to try and keep track of the loop number. You should find that the ordinary local variable cannot remember its value between executions of the loop function.
  - Declare a new automatic local variable to hold the loop number and initialize it to zero.
  - Each time the loop function runs, add 1 to the local variable. Modify/extend your print statements to display text matching the following format (but not these example values or names):

```
24388 ms: gYourGlobalName = 12, someStaticLocal = 12, someLocal = 1
```

✏ Copy the sketch into your answer document (ensure to put a clear label/heading above it) and demonstrate it to a lab demonstrator.