Laboratory 1: Introduction to the ESP8266, the Aston IoT hardware platform and the Arduino environment

Introduction

The Internet of Things (IoT) is set to revolutionise how the world operates. These laboratories are designed to equip you with the knowledge and practical skills required to create IoT hardware devices.

Various microcontrollers can be utilised in Internet of Things (IOT) applications. However, some are more suited to this task than others due to having inbuilt WiFi or radio connectivity. It is for this reason that hardware based upon ESP8266 microcontrollers are used throughout this module. In addition to this, there is considerable support in terms of software that allows for the rapid development of IoT applications.

Equipment and resources

Hardware:

- ESP8266 Aston IoT hardware platform.
- USB A to micro B cable.

Software & documentation:

- Arduino Integrated Development Environment (IDE). For installation and further guidance on using personally owned computers, please refer to Laboratory 0.
- Schematics and datasheets of the various modules – can be found in the resources folder on Blackboard and in the board manual.

Task 1: A reintroduction to the Arduino environment, an embedded “Hello World”

Saving and file name convention

You should keep a copy of every task that you complete on your **student network drive (H:) OR if working remotely a drive which is backed up on a regular basis. Use the folder H:\EE3IOT\labs,** where **H** is your drive**.** Sketches should be saved in the format of **surnameinitialLTX**, where **L** is the lab number and **X** is the task number.

For example, if Fred Smith has completed task 1 of Lab 1, the sketch should be saved as **smithfL1T1**.

Note that step by step instructions will only be given in this introductory laboratory as the process of using the Arduino environment is relatively intuitive.

1. Plug the Aston IOT hardware platform into the PC via the USB A to micro B lead. Note that the USB lead plugs in to the module located in the top left of Figure 1.

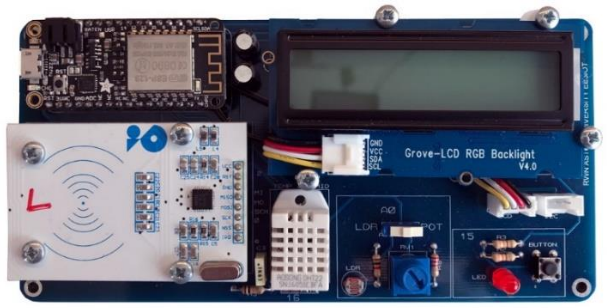
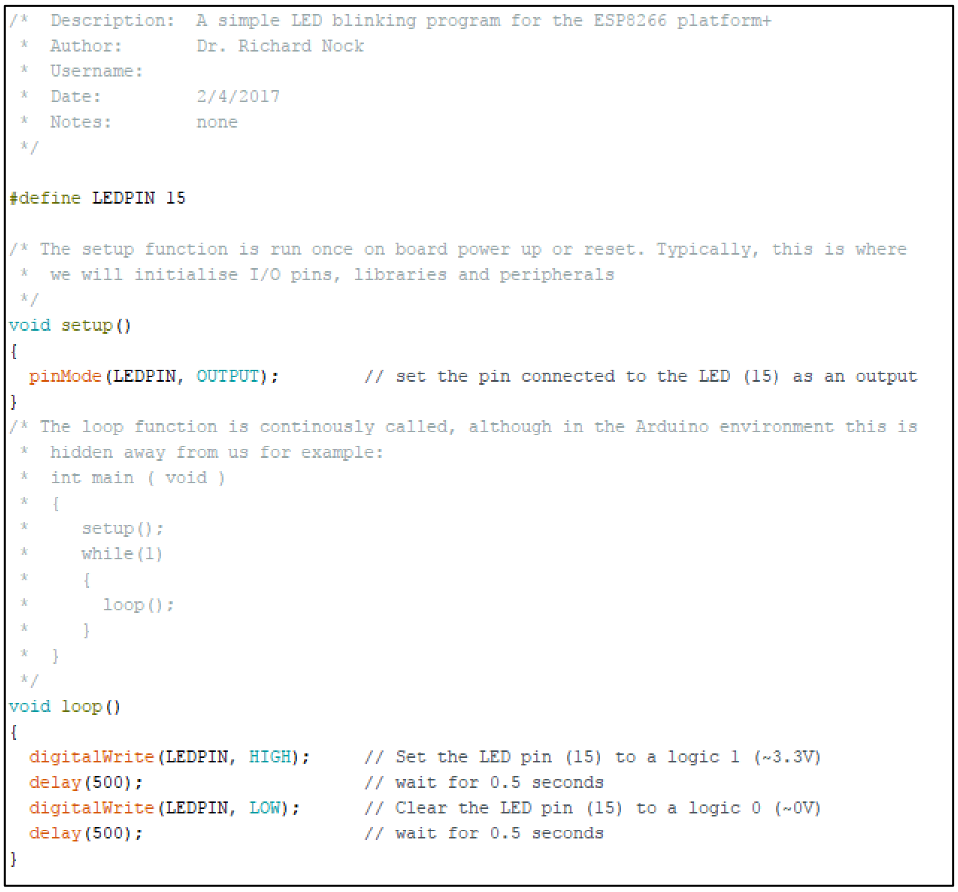


Figure 1: The hardware setup

2. Open the Arduino development environment:
   – Typically, this is located **Programs | Arduino** or search for Arduino in **"Search programs and files"** in the Windows Start menu.
3. Start a new sketch.
   – **File | New**
4. Ensure that you have followed laboratory 0 to setup the required software and use it to also setup your sketch.
5. Enter the code given in **Figure 2**.
   – Focus on writing the code first, add comments later.
   – **#define LEDPIN 15** is known as a **macro**. After the macro, any instance that the text LEDPIN is encountered will be replaced by **15** before the code is compiled. This allows for the programmer to give meaningful names to pin numbers, constants etc. It also means that if we ever wish to change a constant, we only have to change one line rather than many!
   – The **pinMode** function allows us to set whether a digital pin is an **output** or an **input**.
   – The **delay** function can create delays in units of **milliseconds ($\times 10^{-3}$ seconds)**.
   – The **digitalWrite** function allows for us to set a digital output **high** or **low**.
6. Try "verifying" (compiling) the code.
   – **Click on the tick (verify) symbol,** .
   – **Debug the code to remove any warnings or errors generated. Do not ignore warnings.**
7. Upload the code to the board and verify that the on board led blinks at the correct rate of 1 Hz.
   – **Click on the arrow (Upload) symbol,** .
   – **Wait for the upload to complete.**
8. Verify that LED blinks at a rate of 1Hz.



```
/*  Description:  A simple LED blinking program for the ESP8266 platform+
 *  Author:       Dr. Richard Nock
 *  Username:
 *  Date:         2/4/2017
 *  Notes:        none
 */

#define LEDPIN 15

/* The setup function is run once on board power up or reset. Typically, this is where
 *  we will initialise I/O pins, libraries and peripherals
 */
void setup()
{
  pinMode(LEDPIN, OUTPUT);        // set the pin connected to the LED (15) as an output
}
/* The loop function is continously called, although in the Arduino environment this is
 *  hidden away from us for example:
 *  int main ( void )
 *  {
 *     setup();
 *     while(1)
 *     {
 *       loop();
 *     }
 *  }
 */
void loop()
{
  digitalWrite(LEDPIN, HIGH);     // Set the LED pin (15) to a logic 1 (~3.3V)
  delay(500);                     // wait for 0.5 seconds
  digitalWrite(LEDPIN, LOW);      // Clear the LED pin (15) to a logic 0 (~0V)
  delay(500);                     // wait for 0.5 seconds
}
```

Figure 2: Hello World for embedded systems

Task 2: Analogue inputs

Most sensors that measure physical phenomena (such as temperature, light level, sound etc) typically have an analogue output. Hence, most microcontrollers include an Analogue to Digital Converter (ADC) such that voltages can be converted into a digital code (number). The digital representation of the sensor's voltage can then be utilised for monitoring and control purposes. Remember to follow the commenting and indentation style guidelines given below.

IDENTATION STYLE

Consistent indentation of code allows for the reader to easily identify the structure and behaviour of your software. **After a control/flow structure or function, place a pair of curly braces below it.**

```
if ( logical condition )
{
    printf("The logical condition is true! ");
}
else
{
    printf("The logical condition is NOT true! ");
}
```

Poor indentation will result in a loss of marks in the coursework!
The Allman coding style is highly recommended and is used in examples throughout this module.

COMMENTS

Comments are essential for assisting both yourself and others to understanding the functionality of your code. However, comments should be **meaningful and describe the functionality of the code rather than how the code works.** For example, the following comment states the obvious (from inspection):

AV = (a + b + c) / 3; // Add three variables and divide by 3

Whereas the following is more useful and describes the **intended functionality** in a concise manner:

AV = (a + b + c) / 3; // Store the average of a b and c in AV

The exception for this rule is configuring registers (setting bits as experienced in EE2EDP) and for describing what various arguments passed to a function are.

1. Investigate the analogRead(); function on the Arduino language reference website : https://www.arduino.cc/en/Reference/HomePage .
   - Note that the ESP8266 microcontroller has an ADC is a 10-bit part with reference of approximately 1V. Hence, **1V** will be equal to **1023** and **0V** will equal **0**.
2. Create an Arduino sketch that will adjust the frequency the LED's flashing dependent upon the position of the rotation sensor (potentiometer).
   - An **ADC value of 1023** should blink at a rate of approximately **0.5 Hz.**
   - An **ADC value of 0** should blink at a rate of approximately **0.5 kHz**.
   - Sketch the waveform required if you are struggling to implement this code. Is there any similarity with the value read from the ADC input and the delays required to flash at these frequencies?
   - **Hint: A copy of the code from task 1 will provide a good starting point!**
3. Upload the sketch to the IOT hardware platform and verify it works as expected.

Task 3: Serial (RS232) communication

So far, we have looked at digital outputs and analogue inputs. However, so far we do not have an easy means of communicating data to the user for monitoring and debugging purposes. To enable you to debug your applications, we will look at sending and receiving data from the ESP8266 module via the serial port.

1. Create a new Sketch and enter the code given in Figure 4.
2. Add
3. Add additional lines where shown such that the number sent to the IOT hardware platform is shown as **decimal**, **hexadecimal** and **binary** number systems.
   – Note that **Serial.print** does not add a new line character. Hence, after printing the various numbers on one line, a new line can be achieved by printing a `"\n "`.
4. Upload the sketch to the IOT hardware platform.
5. Start the Arduino’s serial monitor:
   – Click on the Serial monitor button, .
6. Configure the serial monitor:
   – In the bottom right hand corner of the monitor window there should be two drop down menus. Ensure that lines end with “Newline” and that the Baud rate is set to 9600 bits per second as shown below in Figure 3.

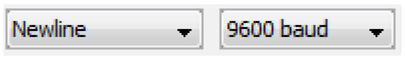
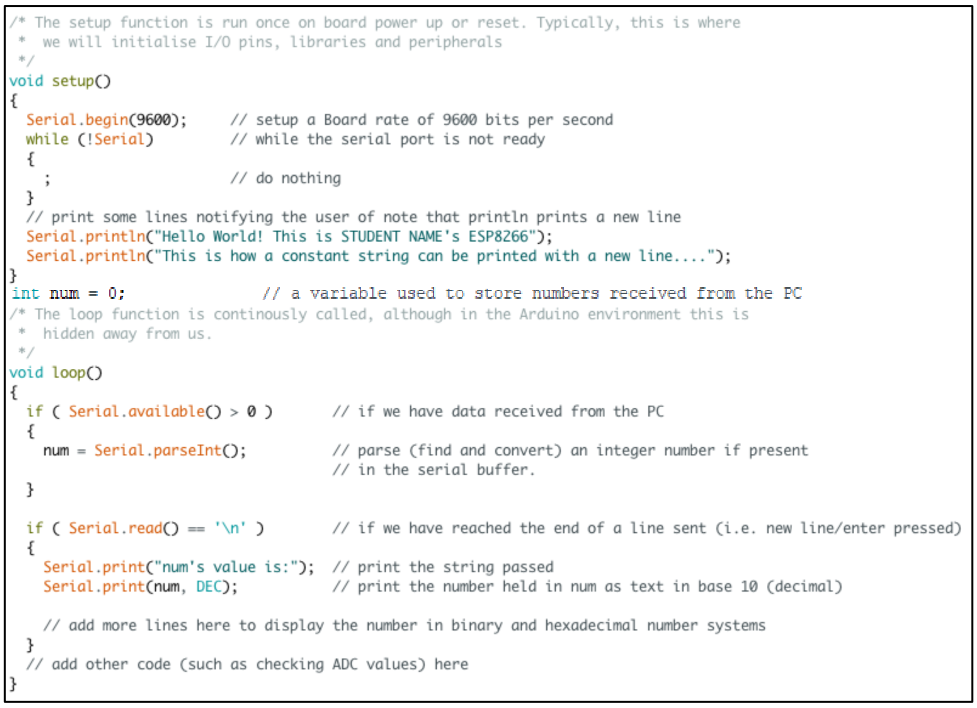

Newline
9600 baud

Figure 3: Selecting a line ending for sending and the correct Baud rate for serial communications

7. Try sending various numbers to the ESP8266 module. Does it return the correct values?
8. Modify the sketch from task 4 such that it reports the value of the potentiometer connected to A0 to the user **every second**.
9. Modify the sketch such that it will turn the LED on if the value of **ADC** is greater than a value passed to the microcontroller via **serial** (this number will be held in **num**).



```
/* The setup function is run once on board power up or reset. Typically, this is where
 *  we will initialise I/O pins, libraries and peripherals
 */
void setup()
{
  Serial.begin(9600);     // setup a Board rate of 9600 bits per second
  while (!Serial)         // while the serial port is not ready
  {
    ;                     // do nothing
  }
  // print some lines notifying the user of note that println prints a new line
  Serial.println("Hello World! This is STUDENT NAME's ESP8266");
  Serial.println("This is how a constant string can be printed with a new line....");
}
int num = 0;                  // a variable used to store numbers received from the PC
/* The loop function is continously called, although in the Arduino environment this is
 *  hidden away from us.
 */
void loop()
{
  if ( Serial.available() > 0 )       // if we have data received from the PC
  {
    num = Serial.parseInt();          // parse (find and convert) an integer number if present
                                      // in the serial buffer.
  }

  if ( Serial.read() == '\n' )        // if we have reached the end of a line sent (i.e. new line/enter pressed)
  {
    Serial.print("num's value is:");  // print the string passed
    Serial.print(num, DEC);           // print the number held in num as text in base 10 (decimal)

    // add more lines here to display the number in binary and hexadecimal number systems
  }
  // add other code (such as checking ADC values) here
}
```

Figure 4: Code template to receive a number and to return the number to the user in decimal/base 10.

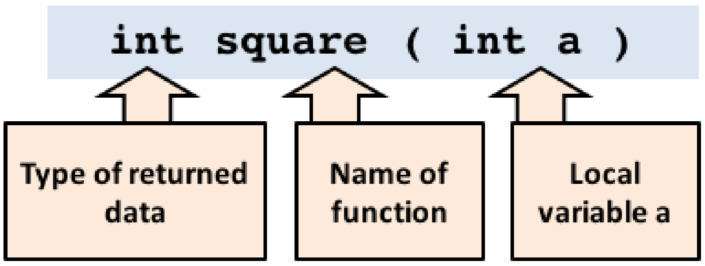
Task 4: Digital inputs and conditional code

So far, only digital outputs, analogue inputs and the serial peripheral have been used. However, being able to read digital inputs and make decisions based upon such values is at the core of any embedded system application.

FUNCTIONS

Functions play a vital role in any programming language as they allow for code to be developed in a modular manner. This not only improves our ability to test code, but it also increases portability and reusability. The labs are designed such that you will create various functions, of which you re-use later on in the course.

The figure below gives an example of a function header that returns the square of an integer variable. Note that the function is typically placed **after** any functions that utilise it and although we can pass many numbers/variables to a function, a function can only return **one** variable (pointers can be used to circumvent this limitation). Variables passed as shown are actually **copied** into local variables within the function.



The return and argument variables can be replaced with **void** , if no data is to be passed or returned from the function.

Example function header and code:

```
/* note that the function square is placed BEFORE the loop function where it is called. This is necessary as otherwise,
 * the compiler would not know of the existance of the function as it has not encountered it yet. This limitation can
 * be circumvented via the use of function prototypes.
 */
int square ( int b )  // function header. This function takes an integer argument and returns an integer value.
{
  int temp;        // a local variable to store the result of the square
  temp = b*b;      // calculate the square of a and store the result in result (note a is a local variable to this function!)
  return temp;     // return an integer value stored in result
}
```

Example use of the function in the loop function:

```
void loop( )
{
    int a = 5;    // this variable a is local to the loop function only.
    int result;   // this variable is also only local to loop.

    result = square(a); // call the function. The number variable a gets copied into the variable b in square.

    Serial.println(result, DEC);  // output the result over serial on a new line in decimal format

}
```

Examples of using functions are located in the supporting material folder on blackboard.

1. This task will guide you through the process to create a sketch based on Task 1 and task 3 such that when a button is pressed, the ESP8266 sends **ONE** message out to the Serial Monitor.
   - **Note, due to a lack of input/output (IO) pins on the ESP8266, the LED and the button share the same pin**. Refer to the manual for further information.
   - Hence, we must periodically check the button state to determine if a message is to be sent.
   - Simple approaches may send multiple messages due to switch bounce.
2. Create a button read function based upon the flow chart given in Figure 5.
   - Hint: the function header should be **int checkbutton (void)**.

- Refer to https://www.arduino.cc/en/Reference/HomePage for information on **pinMode** and **delay** functions.
- Note that a yellow diamond signifies a decision (typically an if structure or loop condition), white square boxes an action (such as calling a function, performing a calculation etc.) and green rounded rectangles represent the start and stop of a function/algorithm.
- Variables current and last must either be global or a local static variable.

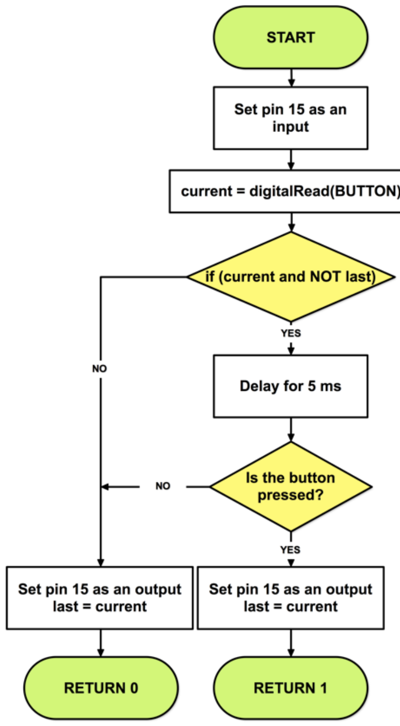

START
Set pin 15 as an input
current = digitalRead(BUTTON)
if (current and NOT last)
YES
NO
Delay for 5 ms
Is the button pressed?
NO
YES
Set pin 15 as an output last = current
Set pin 15 as an output last = current
RETURN 0
RETURN 1

Figure 5: A simple switch debounce algorithm to detect a button press

3. In the loop function, add code such that if the button has been pressed (i.e. the **checkbutton** function returns 1), a message is sent to the serial monitor with your username.
4. Upload the sketch to the IOT hardware platform and verify it works as expected.