Week 3 Lab 1: *Getting started with your Embedded System*

Introduction

During your lab session today, you will work through this sheet. Don't worry if you don't get it all done during the session (especially during this first one). There are optional tasks or extensions for those who want to explore further. Optional tasks are not graded and are just for your practice.

Each lab has a small report to be written up and submitted. There are questions that are highlighted in red that you will have to answer in your lab report. Your lab report is due three days after your lab session. For example: if your lab is on a Monday, you must submit your soft copy lab report on Canvas before 23:59:59 on Thursday. Lab reports should use the Lab Report Template and not be longer than two pages.

Your tasks for today are the following:

T1. Getting Started with Arduino IDE
T2. Driving a Simple Actuator (an LED)
T3. Reading a Simple Sensor (a button)
T4. Microcontroller… Interrupted.
T5. Is it hot in here?
T6. Exploring your box of sensors

T1. Getting Started with Arduino IDE

The Arduino IDE is a free and open-source integrated development environment (IDE). It is one of the possible options for programming our ESP32 device, and it is the one we use during our first lab sessions. It is a simple and straightforward IDE which will let us get going with our hardware quickly.

The programming language we will use is a subset of the language c/c++, which contains an additional set of Arduino specific calls (https://www.arduino.cc/en/Reference/HomePage) and has a limited set of the full c/c++ libraries. Fear not though, as there is a lot of support and libraries for your final project!

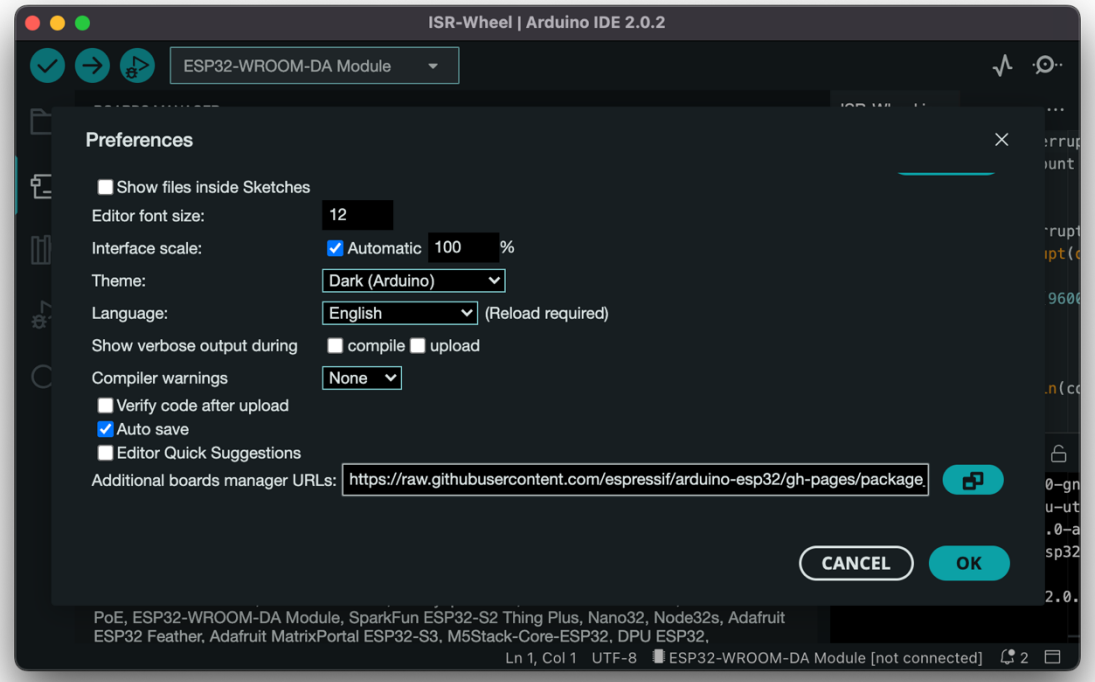
1. First download and install the latest version of Arduino IDE (https://www.arduino.cc/en/software) [For Windows] Do not use the App Store

2. Connect your ESP32 to your laptop via USB cable

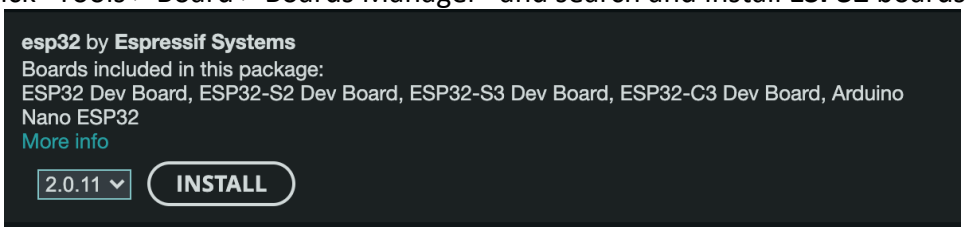
3. Start Arduino IDE on your laptop

We now need to get the board definitions for our ESP32 board.

4. Click "File>Preferences" (on Windows) or "Arduino>Preferences" (on MacOS X) and paste https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json
   to Additional Boards Manager URLs and click **Ok** as pictured on the next page.



5. Click “Tools > Board > Boards Manager” and search and install **ESP32** boards



6. Now let’s do a simple test to check whether you need to install additional drivers. Click “Tools > Board > ESP32 Boards” and select “DOIT ESP32 DEVKIT V1”

7. Click “Tools > Port” and select “COM***x***” (for Windows) or “/dev/cu.usbSerial**x**” (for MacOS X). If you don’t see anything under “Tools>Port” you might need to install a “USB to Serial” driver. For Windows, you can find the drivers here (on Canvas as well). MacOS X /Linux should have drivers already.

8. After the driver installation, replug the ESP32 and try step 7 again

9. We can now run a sample program. Select "File > Examples > 04.Communication > ASCIITable". It will open a new “Sketch” (the Arduino terminology for a program). Have a scroll through and see what it is doing. This program will print out the entire ASCII table to the serial port.

   Every Arduino Sketch minimally needs to have two main functions, the setup() and loop() functions. The setup function runs once when the device is powered on, and then the loop function runs over and over again. Additional functions and variables can be declared.

We can set the mode of the GPIO pins using the function pinMode(pin, mode), where the pin is the pin number and mode is either INPUT or OUTPUT. We can write a digital state to the pin using digitalWrite(pin, value) where the value is either 0 or 1.

10. Let's run the program now. You can compile the sketch using the tick button in the top left-hand corner and upload the program using the arrow button (it also compiles). Try uploading the program to the ESP32.

    If you're on linux, and run into problems, you might have to install pyserial:

```
$ sudo apt-get install pip
$ pip install pyserial
```

    Then you also might have to give yourself permissions to access the serial port:

```
$ sudo adduser <username> dialout
$ sudo chmod a+rw /dev/ttyUSB0
```

    **you might need to replace "ttyUSB0" with "ttyUSB<Some other number>"*
    Use the command "ls /dev/ | grep ttyUSB" to see what usb tty devices are connected*

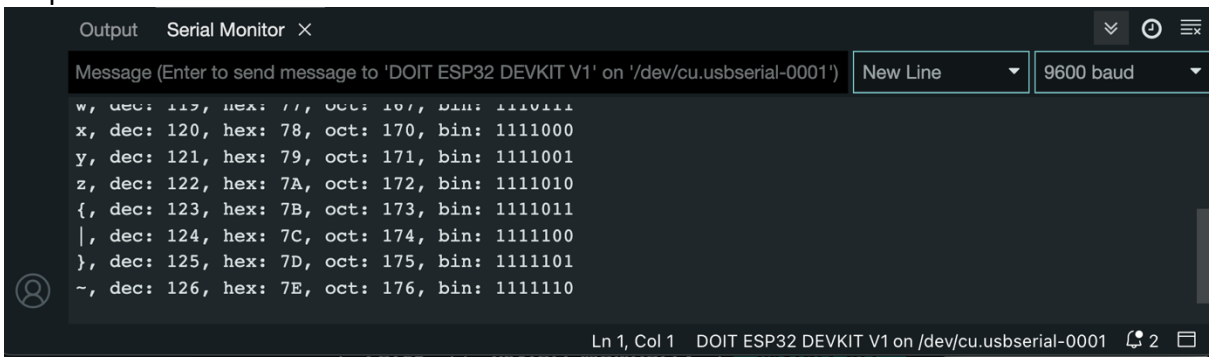
    If you are on Windows, then you might encounter the following error:

```
Serial port COM5
Connecting.....................................

A fatal error occurred: Failed to connect to ESP32: Wrong boot mode detected (0x13)! The chip needs to be in download mode.
For troubleshooting steps visit: https://docs.espressif.com/projects/esptool/en/latest/troubleshooting.html
Failed uploading: uploading error: exit status 2
```

    If you do, just PRESS and HOLD the BOOT button on the ESP32 while it is trying to connect! It should set the ESP32 into Download Mode and allow your program to be flashed.

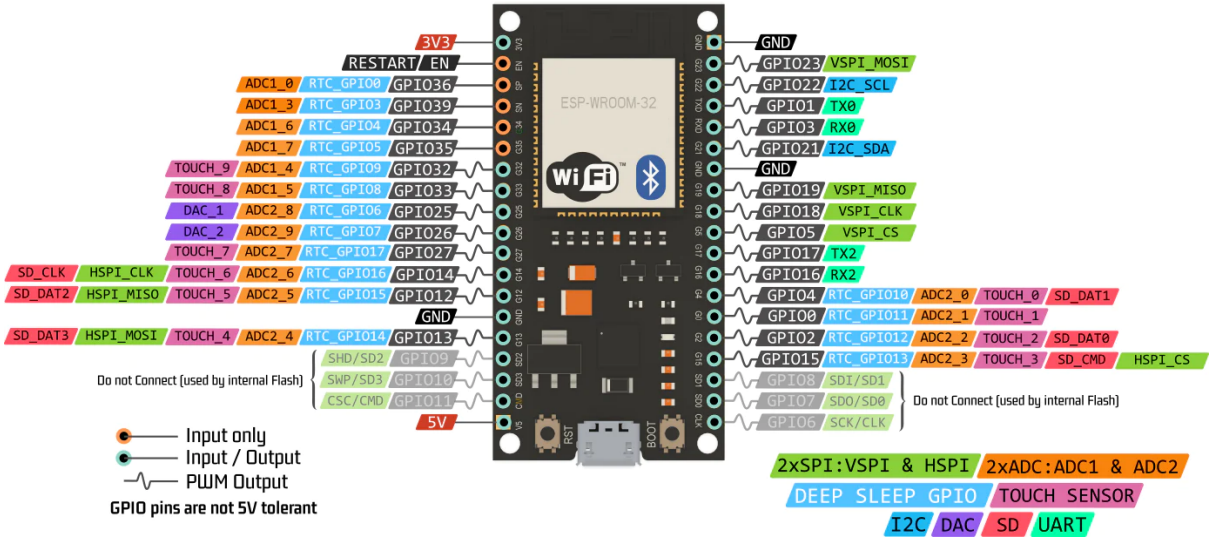
We can view the output of the program by clicking on the serial monitor button on the right-hand side of the Arduino IDE (the magnifying glass). You should see the following output:



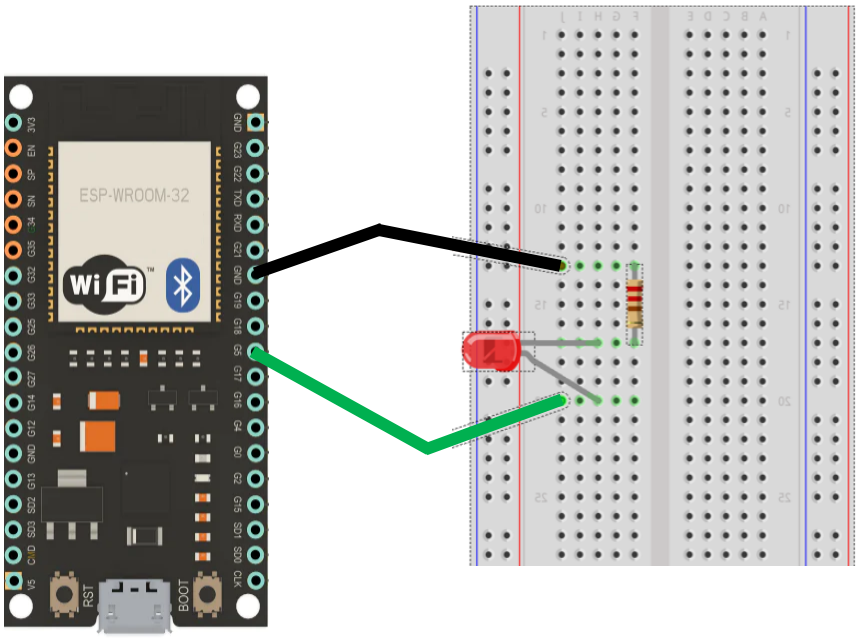
Try pressing the reset button and observe what happens. This program will print out all of the ASCII characters.
Ok great! You have run your first program on the ESP32 (possibly?).

T2. Driving a Simple LED

Our ESP32 has many pins as we have discussed in the lecture. Today we will explore some of these pins. To start with, let's drive an LED using our ESP32.



First, let's wire up our first circuit. We are going to drive an external LED using pin 5. Follow the wire diagram below. Make sure that the LED is the correct way around!



We can now run the Blink program found on Canvas. Have a scroll through and see what it is doing.

Run and upload the program and observe. You should see the LED blinking.

Lab Report Task 1: Change the Blink program to blink with ever increasing frequency. Start with four second blinks, then two second blinks, and so on, until it is permanently on. Describe your changes briefly in your lab report.

Optional Task: Can you make your ESP32 blink **SOS (** ■■■ ▬▬▬▬▬ ■■■**)**?

In Lecture 2, we also introduced the idea of Pulse Width Modulation (PWM) which would allow us to simulate an "analog" signal using our digital pins by changing the duty cycle (turning them on and off rapidly). Refer to the ESP32 figure on the previous page and note that not all the pins have PWM next to them. All your pins on the ESP32 support PWM.

The ESP32 has a PWM controller with 16 independent Channels that can output PWM. We can use an ESP32 specific Arduino function ledcAttachPin(GPIO, channel) to set the bind a pin to a PWM channel. Then you can use ledcWrite(channel, dutycycle) where dutycycle is a number between 0 (low) and 255 (high). Create a new sketch and try out the following program.

```
#define LEDPIN 5
#define LEDCHANNEL 1
#define LEDFREQ 5000
#define LEDRESOLUTION 8

// PWM DutyCycle
int brightness = 150;

void setup() {
    ledcSetup(LEDCHANNEL, LEDFREQ, LEDRESOLUTION);
    ledcAttachPin(LEDPIN, LEDCHANNEL);
    ledcWrite(LEDCHANNEL, brightness);
}

void loop() {

}
```

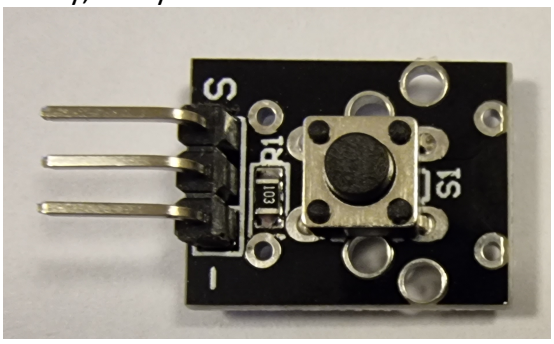
Try modifying the **Brightness** variable value to see the change in LED brightness.

Lab Report Task 2: Modify this program to make your LEDs pulse on and off. Increasing in brightness and then decreasing. Describe your changes briefly in your lab report.

Optional Task: Can you use a timer to achieve this pulsing behaviour?

T3. Reading a Simple Sensor (a switch)

We will be using a switch today, and you can borrow them from the front of the room:



Create a new sketch and copy in the program below. Notice that we are using a library called "Serial". This is for serial communication, and we will go more in depth in the week 3 lecture. For now, we will treat it like a black box. You can use the serial monitor to send messages over USB to your laptop. You can view these messages by clicking on the serial monitor button on the right-hand side of the Arduino IDE (the magnifying glass).

```
#define SWITCH 4

void setup() {
  Serial.begin(9600);
  pinMode(SWITCH, INPUT_PULLUP);
}

void loop() {
  if (digitalRead(SWITCH) == HIGH) {
    Serial.println("Switch Open");
  } else {
    Serial.println("Switch Closed");
  }
}
```

We will now wire up our circuit. Only two of the three pins on the switch module need to be connected, the **–** pin can be connected to the **GND,** and the **S** pin can be connected to **Pin 4** on the **ESP32.** Upload the program and try it out. Press the switch and see the messages. Neat!

For those interested, the pull-up means the button's logic is inverted. It goes HIGH when it's open, and LOW when it's closed.

Let's modify this program so that button behaves like a toggle. If we press the button, it should "toggle" on and when we press it again it should "toggle" off.

```
#define SWITCH 4

byte state = LOW;

void setup() {
  Serial.begin(9600);
  pinMode(SWITCH, INPUT_PULLUP);
}

void loop() {
  if (digitalRead(SWITCH) == LOW) {
    state = !state;
  }
  if (state) {
    Serial.println("Toggle On");
  } else {
    Serial.println("Toggle Off");
  }
}
```

Upload and try out this program. You might notice that sometimes clicking the button does not cause it to toggle. This is because the button needs to be debounced (have a look a Lecture 2 Slide 31). Let's fix this through software, by keeping track of when the button was last pushed, and ignoring any input that occurs in a short window after that time.

Lab Report Task 3: Modify this program to make your ESP32 debounce the switch. You might need to use a larger delay than in the lecture notes. Describe your changes briefly in your lab report.

But this is a very inefficient way of sensing a button push, and notice that our main loop() is continuously running. Our ESP32 is doing a lot of "busy waiting". Let's improve this!

T4. Microcontroller… Interrupted.

We are going to use interrupts to detect our button pushes and toggle the state as before. Recall that interrupts let us handle events without need to keep checking the state of our sensors. Have a look over the slides from Lecture 2 Slide 19 to 24.
Create a new sketch and copy the program below.

```
#define PIN 4

volatile bool pressed = false;
volatile int pressCount = 0;

void IRAM_ATTR isr() {
    pressCount++;
    pressed = true;
}

void setup() {
    Serial.begin(115200);
    pinMode(PIN, INPUT_PULLUP);
    attachInterrupt(PIN, isr, FALLING);
}

void loop() {
    if (pressed) {
        Serial.printf("Button has been
pressed %u times\n", pressCount);
        pressed = false;
    }
}
```

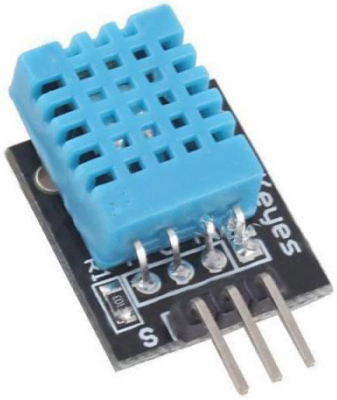
We can observe that in the setup() function, we are now attaching an ISR to the SWITCH pin. Also notice that the program has a new function called toggle(). This function is our Interrupt Service Routine or ISR.

We can now see that the main loop is spent idling (using delay) rather than checking the status of the button over and over. The state of the toggle is printed out in the main loop and not in the ISR. Note that this version is not debounced yet.

Lab Report Task 4: Improve the ISR version of our code so that it also debounces the button push. Remember that ISRs should be very short functions. Also remember that any global variable used in an ISR needs to be volatile. Copy your entire ISR, any relevant code, and describe your changes briefly in your lab report.

T5. Is it hot in here?

Now we are going to try out another sensor, namely the DHT11, the Humidity & Temperature sensor. It is a simple and low-cost temperature and humidity sensor
It uses only 1 wire to communicate with the microcontroller.



Read the label of the pin carefully. There should be three pins on the DHT11, **DATA**, **VCC**, and **GND**. Connect **GND** and **VCC** on the DHT11, to **GND** and **5V** on the ESP32. Connect the **DATA** pin to **16** on your ESP32. We will now take advantage of the libraries available for Arduino. Go to **Sketch > include Library > Manage Library.**

Search “DHT Sensor library” by AdaFruit. You will be asked to install the “AdaFruit Unified Sensor”, click "Install all”. Now let’s open some example code and test it out.

Open File>Example>DHT Sensor Library>DHT_Unified_Sensor

You will need to make some small changes:

1. DHTPIN should be defined as 16
2. DHTTYPE should be defined as DHT11

Try running the program on your ESP32. In the serial monitor, you will be able to see the temperature and humidity as shown below:

```
Temperature: 27.30°C
Humidity: 52.90%
Temperature: 27.30°C
Humidity: 52.70%
Temperature: 28.40°C
Humidity: 59.70%
Temperature: 28.20°C
Humidity: 76.70%
Temperature: 28.10°C
Humidity: 85.30%
Temperature: 29.50°C
Humidity: 89.80%
Temperature: 30.10°C
Humidity: 92.10%
Temperature: 29.90°C
Humidity: 93.20%
```

Try blowing on the sensor to increase the humidity or covering it with your hands to change the temperature. Ok, now let's combine all that we have done today together.

Lab Report Task 5: Modify the example program to make your LED turn on if the humidity or temperature goes beyond a threshold. Describe your changes briefly in your lab report.

Optional Task: Can you make the brightness of the light proportional to the humidity or temperature?

T6. Exploring your sensors and actuators

By now, you can see something of a recipe for how we can prototype and try out sensors/actuators using our ESP32 board.

1. Find and then connect the electronic components to the correct pin(s). Usually, we can find a wiring diagram online and if not look for the tech sheet
2. Write code to interact with the component. Sometimes, libraries are provided together with the components

This is a good general strategy for prototyping, so let's try it out.

Lab Report Task 6: You are to select one of the sensors/actuators in the box you have and try it out. Take a photo of your wired up system and give any relevant code excerpts. Include these and a brief description in your lab report.

Let us know if you have any questions. Your tutor and lecturer will be walking around the lab.