ELEC1601: Laboratory Report 1

SID: 510415022

M18 Group 11

Abstract

Weeks 2 and 3 explore the fundamentals of Arduino circuitry and programming. Following provided lab material, we assembled various electrical trinkets. We familiarised ourselves with the Arduino IDE, TinkerCad, breadboard components and hardware, and Arduino programming techniques. These tasks helped the team to introduce themselves to one another and develop a collaborative style, establishing the work dynamic for the semester. Lastly, these labs introduced programming techniques, interruptions, to allow us to achieve certain behaviour with greater specificity.

1. INTRODUCTION

The purpose of these two initial lab sessions were to introduce us to the processes involved in creating an engineering artefact as a team from start to finish, as well as for us to get to grips with basic software and hardware construction using the Arduino system.

Indeed, as professional practice would go, we first scoped out the problem statement (or the aim) for each artefact we were required to build. As a team we focused on planning our implementation for the system, but also then created a simulation of the physical product before bringing out the hardware. This allowed us to rigorously test our hardware-software interactions without risk of damaging components and without hardware-specific challenges such as calibration. Our team developed strategies to collaborate with one another, evenly distributing responsibilities. These included building circuits, prototyping with TinkerCad, writing code, testing and troubleshooting. We'd interchange these responsibilities as needed, giving everyone the opportunity to familiarise themselves with the tools we will use throughout the semester.

With our own independent experimentation, we explored how changes to our software and physical configuration altered the behaviour of our computer system. For example we explored how the use of `interrupt` achieved an immediate state change with our LED as opposed to an `if` statement within the main loop. This allowed us to gain a deeper understanding of the utility of these programming techniques.

In our final implementation we created a 'dynamic' circuit, a circuit which kept track of and responded to the user's input. The circuit *LEDs and Patterns* involved two buttons and four LEDs, which would turn on or off depending on how many times the user pressed button 1. We also implemented a reset button which when pressed would turn off all LEDs and return all sources to a LOW voltage state.

2. BACKGROUND AND MATERIALS

Several basic electronic components were used in this lab, a full list of which can be found in Table 1. The below sections briefly describe each of these.

2.1 Tools (TinkerCAD/Arduino IDE)

Through TinkerCAD's 3D modelling software developers can simulate the circuitry and programming of their systems, allowing them to test their designs before configuring their physical components. Furthermore, the simulated environment and digital cabling allows for a clear visualisation of how the system will operate. This empowers developers to revise poor designs which may have otherwise damaged hardware. Essentially, TinkerCAD saves time and money.

The Arduino IDE is software developed to provide a means to write, debug and upload code to an Arduino board. The software supports the importation of third-party libraries and provides assess to the serial monitor. This enables developers to debug uploaded code running on the micro-processor.

2.2 Computer

An Arduino Uno R3 was used to facilitate basic computation. These are cheap, easy-to-use micro-controllers that can be programmed and incorporated into circuits. Arduino Uno's have a series of pins that are used to input or output data. The state of these pins are controlled by the programming of the Arduino, allowing for the creation of intelligent circuits. The Arduino programming language is also fairly high-level (a framework built upon C++) making it relatively easy to use. All our circuits utilised an Arduino to facilitate this complex behaviour.

2.3 Sensors (inputs)

Buttons are a commonly used component in many circuits. They allow for interactions between users and an electrical system, with microcomputers interpreting the button's altered state to enact change. Pressing a button creates a conductive connection between each side of it, allowing current to flow. This project uses buttons to allow LEDs to light up when specific entry combinations are provided.

2.4 Actuators (outputs)

The circuits in this lab featured 4 LEDs (Light Emitting Diodes). LEDs are semi-conductive devices that emit light in the presence of an electrical current. LEDs are used both for illumination and indication of a circuit's configuration.

LEDs are much more efficient compared to traditional incandescent bulbs. For this reason, they're commonly used in low-powered systems. Their luminosity varies on each use-case. The brightness of an LED is dependent upon the current; changing the voltage or resistance of a circuit will change the LED's brightness.

LEDs are diodes, in which current can only travel one way through the component. For this reason, it is important to remember that the longer lead of the LED is the positive terminal. For this project, LED's will be used to indicate the flow of current.

2.5 Other materials

Resistors are commonly used to protect components by converting excess electrical energy into heat. They're used in most circuits, so understanding when and how to use them is an important skill. This project will use resistors in combination with buttons and LEDs so that the components will not be damaged.

Table 1. Summary of all components

Component	Number required	Other details
Arduino Uno	1	
Breadboard	1	
Wires	19	
Resistors	3	330 Ω, 1 kΩ, 4.7 kΩ
LEDs	4	
Buttons	2	

3. METHOD

In total there were 4 parts to our lab, each introducing a new hardware or programming concept.

3.1 *Part 1: Blinking an LED*

The first part of the lab was aimed at getting the most basic circuit up and running: a blinking LED. As per Listing 1, we first set up our output pin to whichever Arduino pin was connected to the LED. We then set this pin to output some non-zero voltage, which (as per Figure 1) travels through the LED and through a 330 Ω resistor before returning to ground. The resistor here is important to prevent burnout of the LED. We use some form of a `delay` function to give the effect of the LED "blinking" for 1 second before turning the LED off through setting the pin voltage to zero. This process loops infinitely.

```
assign LED PIN to OUTPUT;

forever:
  make LED PIN output non-zero voltage;
  wait 1 second;
  make LED PIN output zero voltage;
  wait 1 second;
end loop
```

Listing 1. Simple pseudo-code to blink an LED

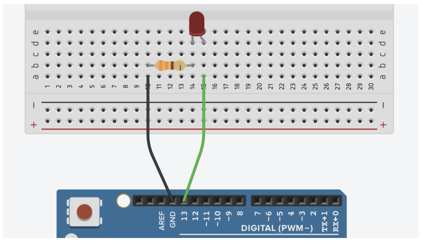


Figure 1. TinkerCAD simulation of part 1

3.2 *Part 2: Buttons, Printing and Delays*

This part of the lab was aimed to introduce the idea of inputs and micro-controllers (specifically a push-switch/button) and also programming the Arduino to respond upon an input being detected. Examining Figure 2a, we can see the circuit directly builds off the circuit from part 1. The new addition consists of an input pin which is either connected to some non-zero voltage through traversing a very high resistance, or, if the push switch is pressed, is connected straight to ground (as current takes the path of least resistance). Hence, input pin 2 takes the binary value of 0 or 1 if the button is pressed or released respectively. We then use these binary values in our output to the LED depending on whether we want the LED to be on or off when the button is not pressed (see Listing 2).

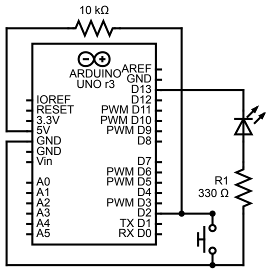
```
assign LED PIN to OUTPUT;
assign PIN 2 to INPUT;

forever:
  make LED PIN output the value of (1-PIN 2);
end loop
```

Listing 2. Pseudocode describing an elegant way of using a micro-controller's signal to control an actuator's behaviour

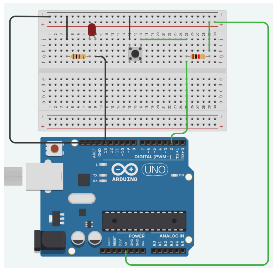
3.3 *Part 3: Interruptions*

Entirely focused on software in this part, we now introduced the idea of an `interrupt` in our program.

Interrupts are commonly used by hardware devices to indicate electronic or physical state changes that require time-sensitive attention. The reason we use an interrupt in part 3 is to prevent



(a) Circuit diagram



(b) TinkerCAD simulation

Figure 2. Technical diagrams for parts 2 & 3

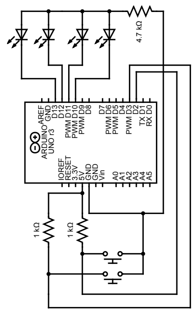
the program constantly reading from the button and writing to the LED as it did in part 2 (a process known as *polling*). This inefficiency can be solved by only reading/writing upon the detection of a change, which is what the interrupt will check for.

```
initialize INTERRUPT VARIABLE to false;

set INTERRUPT PIN to INPUT;
set LED PIN to OUTPUT;
attach INTERRUPT FUNCTION to INTERRUPT PIN;

INTERRUPT FUNCTION:
  set interrupt fired to true;
  make LED PIN output non-zero voltage;

forever:
  if interrupt fired:
    reset INTERRUPT VARIABLE;
  end if
end loop
```

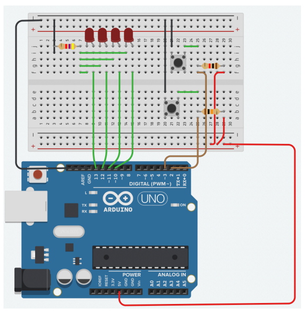
Listing 3. Pseudo-code to fire an LED upon pressing a button (using interrupts)

3.4 *Part 4: LEDs and Patterns*

Here we created a much more complex system combining elements from all 3 preceding parts. The goal of part 4 was to design a circuit which responds to the number of times a button was pressed and which implements a 'reset' button through the use of an `interrupt`.



(a) Circuit diagram



(b) TinkerCAD simulation

Figure 3. Technical diagrams for part 4

```
initialize MAIN INTERRUPT VARIABLE to false;
initialize RESET INTERRUPT VARIABLE to false;
initialize TIMES PRESSED VARIABLE to 0;

set all 4 LED PINS to OUTPUT;
set both interrupt pins to INPUT;

attach CHANGE LED FUNCTION to INTERRUPT PIN 1;
attach RESET LED FUNCTION to INTERRUPT PIN 2;

CHANGE LED FUNCTION:
  set MAIN INTERRUPT VARIABLE: true;

RESET LED FUNCTION:
  set RESET INTERRUPT VARIABLE: false;

```

```
forever:
  if increment button pressed x <= 4 times:
    // turn on the same amount of LEDs as times
    button pressed
    for each pin between 10 and 10+x:
      output a non-zero voltage;
    end for
  else if increment button x > 4 times:
    make all LEDs flash with 1 second intervals;
  end if

  if reset button pressed:
    // turn off all LEDs
    for each pin between 10 and 14:
      output zero voltage;
    end for
  end if
end loop
```

Listing 4. Pseudo-code to turn on as many LEDs as times button is pressed

4. RESULTS

We were successful in designing and assembling our hardware and software implementations. In particular, the completion of part 4 and testing of limitations highlights our understanding of the concepts and principles involved in basic circuitry design and programming.

The greatest limitation of our system is that the reset button does not take effect instantaneously – the loop that controls the flashing of the LEDs must complete its last iteration before the system actuates to the changed state. This is an inconvenience at best, but catastrophic depending on the real-world use case. If this reset button was controlling an emergency stop function for heavy machinery, there must be zero delay between pressing the button and its effect; anything otherwise could be life-threatening.

5. RELATION TO REAL-WORLD ELECTRONICS

5.1 *Part 5.1: Premise and Learning*

This lab introduced how to connect sensors and actuators to an Arduino micro-controller. We can use these techniques to create our own complex system, for example, a heat monitoring device for a reptile enclosure. A temperature sensor would be embedded within the tank, the output of which would be processed by a micro-controller. If the environment becomes inhospitably warm, say from a heat lamp, the system would provide a visual and auditory indication.

5.2 *Part 5.2: System Description and Challenges*

Our primary sensor is a temperature diode, the conductivity of which is correlated to it's thermal environment. This conductivity is interpreted by the Arduino to estimate the surrounding temperature.

Ideally, this sensor would be positioned within an enclosure connected via male to female extension leads. However, the sensor and its leads would need to be covered in a protective sheath to prevent damage incurred by the reptilian inhabitants.

Our primary actuator is a piso (speaker) which can emit sound at specified frequencies. However, its limited volume may not guarantee users notice the alarm. Although, a larger speaker may require voltage beyond the Arduino's capacity. Additionally, some users may want the ability to temporarily silence the device. This would be achieved using a pull-down resister, as previously demonstrated. Lastly, the visual indication of an active alarm is achieved using a simple LED.

5.3 *Part 5.3: Pseudocode and Circuit*

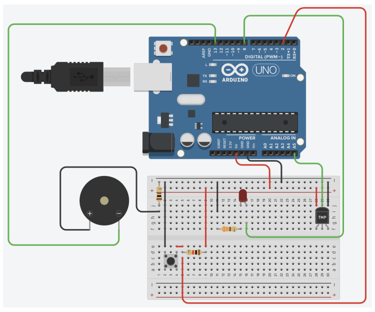


Figure 4. TinkerCAD simulation, Piso and Temp Sensor

```
1  assign LED PIN to OUTPUT;
2  assign PIN 2 to INPUT;
3
4  initialize BaselineTemp to 40;
5  initialize Celsius to 0;
6  initialize Silent to false;
7
8  set PIN_A5 as TEMP_INPUT;
9  set PIN_13 as PISO_OUTPUT;
10 set PIN_8 as LIGHT_OUTPUT;
11
12 attach TOGGLE SILENT FUNCTION to INTERRUPT PIN 2;
13
14 TOGGLE SILENT FUNCTION:
15   set Silent to true;
16
17 forever:
18   set Celsius to TEMP_PIN;
19
20   if Celsius > BaselineTemp and !Silent:
21     PISO_PIN set HIGH;
22     make all LEDs flash with 500ms intervals;
23
24   else if Celsius < BaselineTemp and Silent:
25     set Silent to false;
26   end if
27 end loop
```

Listing 5. Pseudocode describing the programming of our heat maintenance system

5.4 *Part 5.4: Explanation of Pseudocode*

Line 18 controls the activation of the piso speaker. However, when writing actual code you utilise the inbuilt function `tone(pin, frequency, duration)`. We can also take advantage of interrupts to silence our system with greater immediacy. Take note, that when an alarm is disabled, we don't want it to remain so indefinitely. Therefore, lines 24-25 disable 'silent mode' once the temperature returns within the pre-defined threshold.

6. CONCLUSION

Our group was successful in attaining the learning outcomes from these laboratory exercises. We all engaged in team-based design, drawing on our creativity and each of our respective domains (electrical engineering/computer science) to deliver solutions to each part of the lab. Particularly demonstrative of this was part 4 which simply provided us a problem brief; we were then required to apply concepts, principles and techniques from prior parts to scope, build and test our solution to this problem.

In particular, we've learnt how to connect sensors and actuators to an Arduino micro-controller. We've also became familiar with simulation tools, such as TinkerCAD, to plan and test our designs before implementation. We've also gained experience in C++ Arduino programming, being able to associate pins as inputs, process their state (using traditional control structures and interrupts) and actuate change in the environment.

References

ArduinoDocs. 2022a, Digital Read Serial, https://www.arduino.cc/en/Tutorial/BuiltInExamples/DigitalReadSerial

—. 2022b, tone(), https://www.arduino.cc/reference/en/language/functions/advanced-io/tone/

Autodesk. 2011–2022, TinkerCAD, https://www.tinkercad.com/

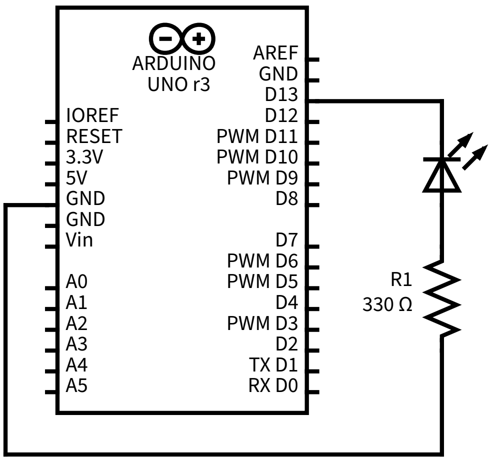
ELEC1601 Team. 2022, ELEC1601 Course Notes, Tech. rep., The University of Sydney

Sparkfun. 2022, Processor Interrupts with Arduino, https://learn.sparkfun.com/tutorials/processor-interrupts-with-arduino/all

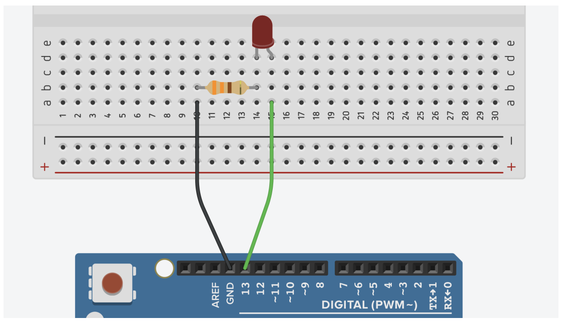
Studio, S. 2022, Pull up resistor vs pull down differences, https://www.seeedstudio.com/blog/2020/02/21/pull-up-resistor-vs-pull-down-differences-arduino-guide/

Appendix 1. Technical diagrams

View the full-size technical diagrams below.

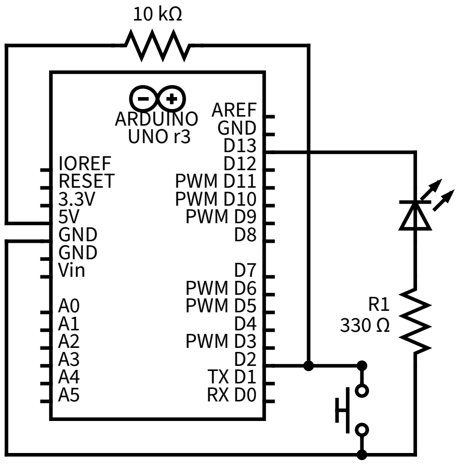


(a) Electrical circuit diagram

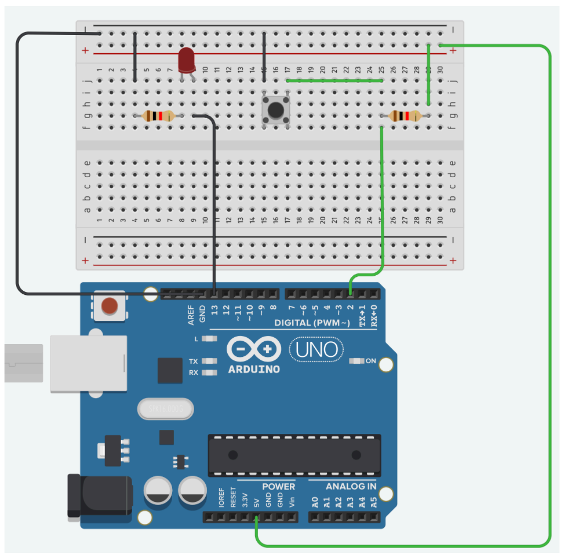


(b) TinkerCAD simulation

Figure 1. Full-size technical diagrams for part 1

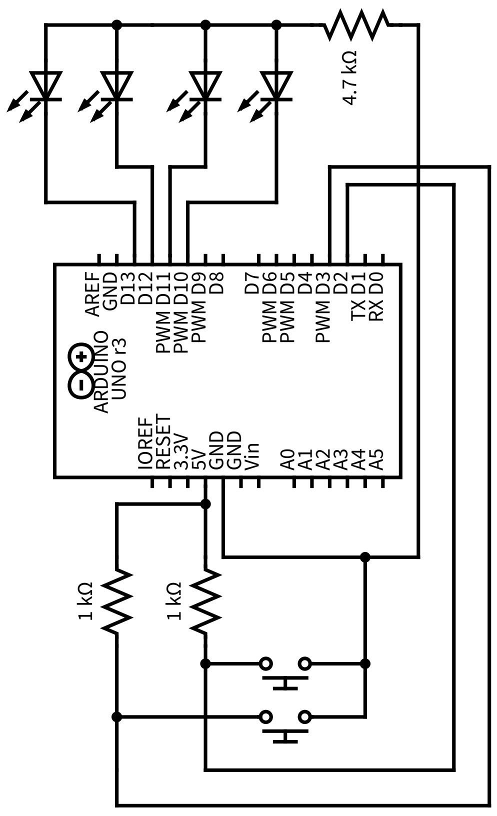


(a) Electrical circuit diagram

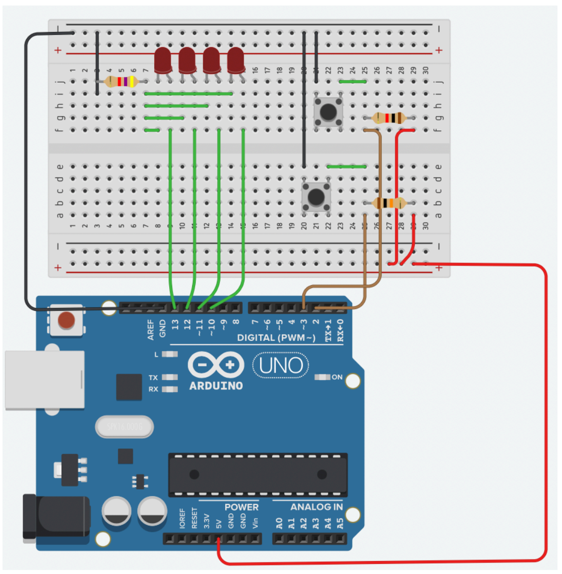


(b) TinkerCAD simulation

Figure 2. Full-size technical diagrams for parts 2 & 3



(a) Electrical circuit diagram



(b) TinkerCAD simulation

Figure 3. Full-size technical diagrams for part 4

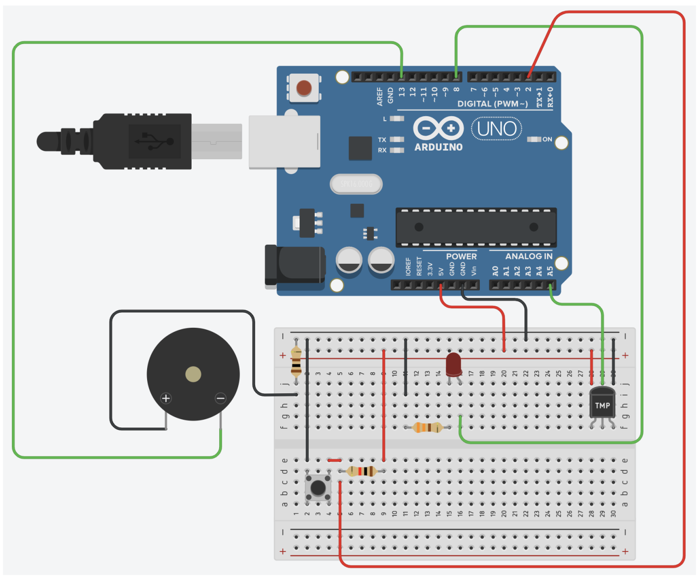


Figure 4. TinkerCAD simulation of our lab’s real-world application