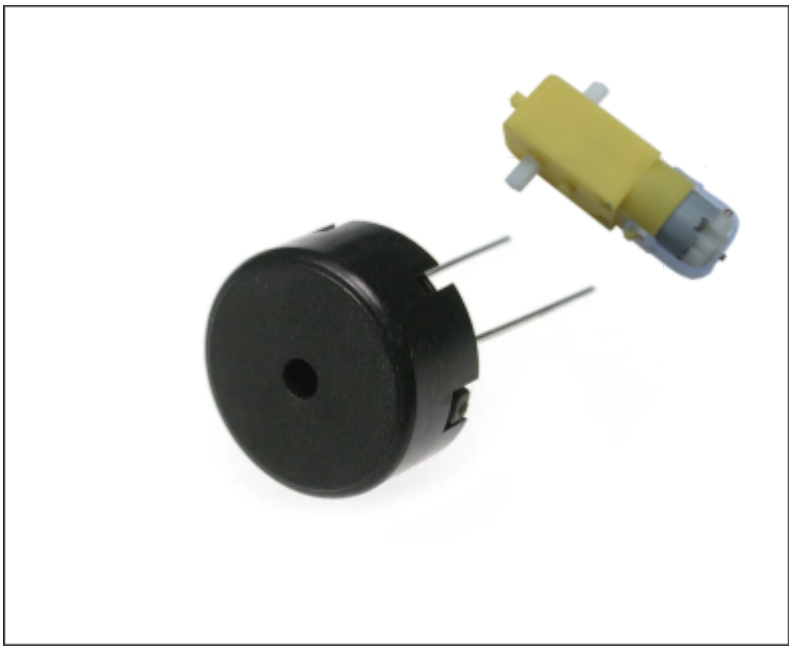
EnVision Tool for Imagination: Buzz Buzz



Team Members Gordon Ling, Mico Guinto, Sophia Lam

Last Updated 12-01-2023

Table of Contents

Introduction

In this tutorial, we will be programming with a customized printed circuit board (PCB) called the Imagination Board. This tutorial will delve into the practical aspects of programming a microcontroller board, connecting theoretical and real-world applications such as embedded systems, programming, and electrical engineering. You will learn how to manipulate a motor and buzzer using CircuitPython and Arduino C. The objective of this tutorial is to demonstrate how to govern peripheral devices through software-based input to integrate programming and hardware control. By the end of this tutorial, you will be able to understand the basic principles of software control over hardware components.

Learning Objectives

- Peripheral Interface
- Python Programming
- C Programming
- Circuit Building

Background Information

A piezo buzzer is an electrical component that generates sound with multiple functions. These buzzers are used due to their simplicity and versatility which makes them beginner-friendly. It is commonly used for alarms and simple melody generation but can be applied in various situations. However, the piezo buzzer offers limited sound quality and limited range. Another alternative is the application of a speaker. However, piezo buzzers are known to use less power. The point of the tutorial is to be able to manipulate how the buzzer operates based on some input. Key concepts to understand through this tutorial include microcontroller programming, embedded systems, and hardware control. Of course, with this tutorial being simple there are not many layers to it. However, the hardest part tends to be combining the software and the hardware and this tutorial is meant to show the tip of the iceberg on software controlling peripherals.

Getting Started

For any software prerequisites, write a simple excerpt on each technology the participant will be expected to download and install. Aim to demystify the technologies being used and explain any design decisions that were taken. Walk through the installation processes in detail. Be aware of any operating system differences.

For hardware prerequisites, list all the necessary components that the participant will receive. A table showing component names and quantities should suffice. Link any reference sheets or guides that the participant may need.

The following are stylistic examples of possible prerequisites, customize these for each workshop.

Required Downloads and Installations

VSCode: https://code.visualstudio.com/docs/setup/windows

Python: https://code.visualstudio.com/docs/python/python-tutorial

Arduino IDE: https://www.circuitbasics.com/arduino-basics-installing-software/

Required Components

List your required hardware components and the quantities here.

Component Name	Quantity
Imagination Board	1
DC Motor	1
9V Battery Holder	1
Buzzer	1

Required Tools and Equipment

Hardware: Computer, VSCode, Arduino IDE, 9V Battery, Soldering Station, Breadboard, Buzzer and Wires,
Software: Python Serial Library, Python Tkinter, Python Thread, Python Lock

Part 01: Creating a GUI

Introduction

With an imagination board, we will drive a DC Motor and buzzer with varying speed and volume using a slider on a graphical user interface (GUI). The GUI allows users to interact with electronic devices, in this case, the DC motors and buzzer, through graphical icons. In this part, we will set up the GUI with a slider to control the speed of a DC motor and the frequency of the buzzer.

Objective

- Create a GUI with a slider to drive the sound of the buzzer and speed of DC Motor

Background Information

In this challenge, you will learn how to design and implement a GUI with a slider and retrieve the value from the slider.

Components

- PC/Laptop
- Software: Python, VSCode, Tkinter

Instructional

1) Follow the installation guide for VSCode and Python. To install Tkinter, run the command 'pip install tk' in your VSCode terminal. Then, all you need to do is include the following imports:

```
import tkinter as tk
import tkinter.ttk as ttk
```

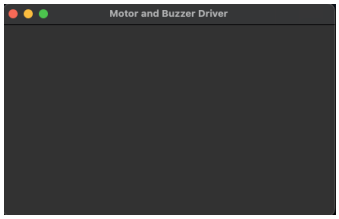
2) Now we will import the necessary libraries, create a class that inherits from tk.TK, set window title, and run an event loop to display the window:

```
class App(tk.Tk):
    def __init__(self):
        super().__init__()

        self.title("Motor and Buzzer Driver")

if __name__ == '__main__':
    app = App()
    app.mainloop()
```

What you should see after clicking the run icon on the top right of VS Code is a blank window that pops up:



3) Now we want to include a slider in our GUI so we can later on drive the motor's speed and buzzer's frequency. In Tkinter, the scale widget will provide a graphical object of a scale. Let's include the following two lines that will create a slider:

```
1   import tkinter as tk
2   import tkinter.ttk as ttk
3
4   class App(tk.Tk):
5       def __init__(self):
6           super().__init__()
7
8           self.title("Motor and Buzzer Driver")
9
10
11          self.scale = ttk.Scale(self)
12          self.scale.pack()
13
14
15  if __name__ == '__main__':
16      app = App()
17      app.mainloop()
18
```

If you rerun the new code, you should see a slider that can slide back and forth but we are not storing the value of this slider anywhere. On line 11, we created an instance of the scale variable and on line 12, we displayed our scale widget within our tkinter window.

4) Now let's make sure we can store the value of the slider in a variable so we can communicate this variable to the Imagination board we will use later on for the DC motor and buzzer.
    a) First we create a variable to hold the scale's value. We will also set the minimum of the scale to 0 and maximum to 100 for now:

```
self.scale_var = tk.DoubleVar()

self.scale = ttk.Scale(self, variable=self.scale_var, from_=0, to=100)
self.scale.pack()
```

    b) Now we will display the value of the slider on our GUI as slider changes. We want a label associated with our slider so we can display the value of the slider as text on our GUI. But we also need a callback function to actually get the slider's value when the slider changes. We define our

callback function within class App(tk.TK) as:

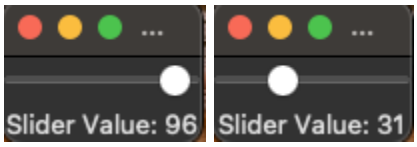
```
20      def update_slider_value(self, event):
21          value = self.scale_var.get()
22          self.value_label.config(text=f"Slider Value: {value}")
```

We then make the call to this function upon an event, which is to say when the slider changes. We will associate this function with our scale widget and initialize a label to display the slider value:

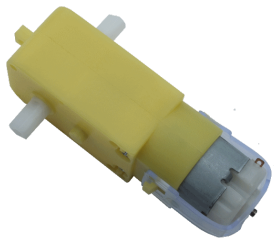
```
self.scale = ttk.Scale(self, variable=self.scale_var, from_=0, to=100, command=self.update_slider_value)
self.scale.pack()

self.value_label = ttk.Label(self, text="Slider Value: 0")
self.value_label.pack()
```

When you run the code and move the slider around, slider value should change:



Part 02: Driving DC Motor



Introduction

With an imagination board, we will drive a DC Motor with the GUI slider we just created.

Objective

- Interface a DC Motor with an Imagination board
- Use Serial Communication to drive motor

Background Information

Briefly explain the technical skills learned/needed in this challenge. There is no need to go into detail as a separation document should be prepared to explain more in-depth about the technical skills

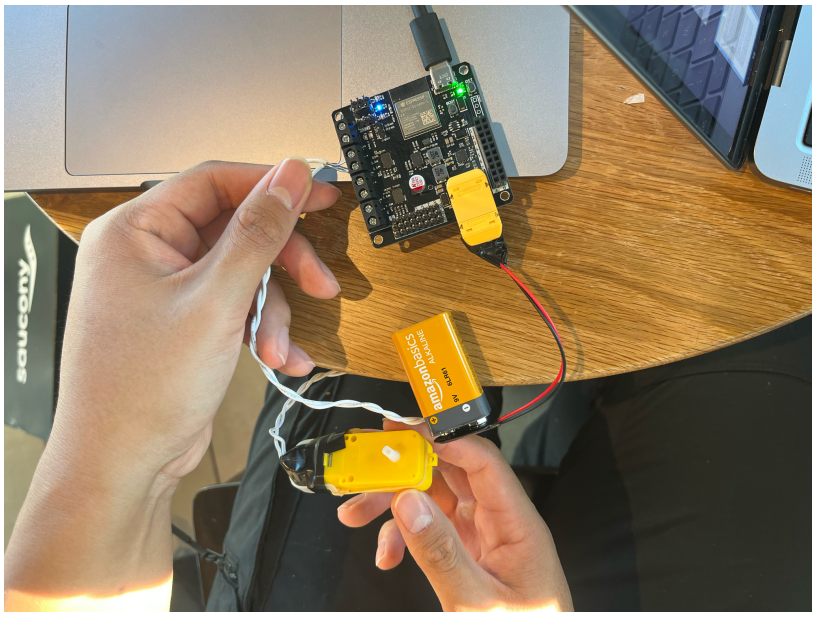
In this challenge, you will learn how to drive a DC motor with the GUI on your laptop. In addition, you will learn how to interact with peripherals for microcontrollers and communicate with microcontrollers with Python's Serial library.

Components

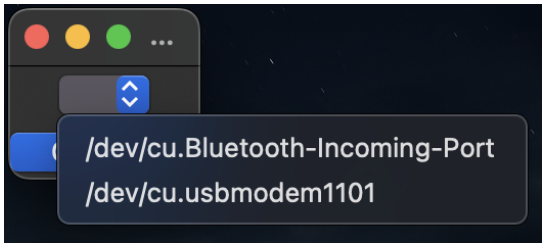
- DC Motor
- Imagination Board
- 9V battery
- PC/Laptop

Instructional

1) Connect an external battery to the imagination board and a DC motor to a pair of screw terminals. Then connect the imagination board to the laptop with a usb-c cable.



2) Then we make the option to connect to the imagination board for our python GUI. We would like a menu that allows us to choose the serial port associated with the imagination board instead of hardcoding a port:



Make the following changes to your code:

```
1  import tkinter as tk
2  import tkinter.ttk as ttk
3  from serial import Serial   # add this
4  from serial.tools.list_ports import comports    # add this
5
6  class App(tk.Tk):
7      ser: Serial
8
9      def __init__(self):
10
11         super().__init__()
12
13         self.title("Motor and Buzzer Driver")
14
15         self.port = tk.StringVar() # add this
16         self.scale_var = tk.IntVar()
17
18         self.scale = ttk.Scale(self, variable=self.scale_var, from_=0, to=100, command=self.update_slider_value)
19         self.scale.pack()
20
21         self.value_label = ttk.Label(self, text="Slider Value: 0")
22         self.value_label.pack()
23
24         SerialPortal(self) # add this
25
26     # add this
27     def connect(self):
28         self.ser = Serial(self.port.get())
29
30     def update_slider_value(self, event):
31         value = self.scale_var.get()
32         self.value_label.config(text=f"Slider Value: {value}")
33
```

On lines 3 and 4, we import the necessary modules from the serial and comports library.

On line 15, we define a tkinter string variable instance to hold the user-inputted serial port information.

On line 24, we invoke serial port functionality which we define as:

```
34 # Define this
35 class SerialPortal(tk.Toplevel):
36     def __init__(self, parent: App):
37         super().__init__(parent)
38
39         self.parent = parent
40         self.parent.withdraw() # hide App until connected
41
42         ttk.OptionMenu(self, parent.port, '', *[d.device for d in comports()]).pack()
43         ttk.Button(self, text='Connect', command=self.connect, default='active').pack()
44
45     def connect(self):
46         self.parent.connect()
47         self.destroy()
48         self.parent.deiconify() # reveal App
49
```

The SerialPortal class creates a pop-up window that allows the user to select a serial port from available options retrieved using comports(). It hides the main App window until a port is connected via the "Connect" button which establishes the connection and closes the pop-up to reveal the main window with the slider for use. When running the code, you should now see our desired functionality of a menu allowing us to choose the correct port before proceeding to the slider.

3) In order to make sure we correctly close our ports when we exit the app, we need to implement functionality to disconnect from our board and exit the app safely.
    a) Add these functions to our App

```
42      def __enter__(self):
43          return self
44
45      def __exit__(self, *_):
46          self.disconnect()
47
```

And the following change to guarantee resource release:

```
65  if __name__ == '__main__':
66      with App() as app:
67          app.mainloop()
68
```

By adding these changes, we construct App in a managed context

4) Now we launch Arduino IDE and write base code (just high and low) for the motor. Upload the code to the correct Serial port and make sure the board selected on Arduino IDE is ESP32 Dev Module.

```
// Pins for Motor 1 and 2
const int MOTOR_1 = 2;  // Pin 1  (or 1)
const int MOTOR_2 = 1;  //pin 2 (or 2 for value)

const int S_OK  = 0xaa;
const int S_ERR = 0xff;

void setup() {

  // Set pins as output
  pinMode(MOTOR_1, OUTPUT);
  pinMode(MOTOR_2, OUTPUT);

  USBSerial.begin(9600);

void loop() {
  // Set Motor 1 to HIGH and Motor 2 to LOW
  digitalWrite(MOTOR_1, HIGH)
  digitalWrite(MOTOR_2, LOW)
  delay(1000)
}
```

5) With the DC motor being driven by our loop, we now want to change it so we run the motor based on the GUI's slider variable instead of a loop. In other words, we want to receive serial commands from our Python GUI on the Arduino side.

```
void on_receive(void* event_handler_arg, esp_event_base_t event_base, int32_t event_id, void* event_data) {

    // Reads in 1 byte
    int motorSpeed = USBSerial.read();

    // Ensure byte is valid
    if (motorSpeed>255 || motorSpeed<0) {

      // Invalid byte received
      USBSerial.write(S_ERR);
      return;
    }

    // Update motor to GUI Speed
    analogWrite(MOTOR_1, 0);
    analogWrite(MOTOR_2, motorSpeed);

    USBSerial.write(S_OK);
}

void setup() {

  // Set pins as output
  pinMode(MOTOR_1, OUTPUT);
  pinMode(MOTOR_2, OUTPUT);

  // Upon event execute on_receive
  USBSerial.onEvent(ARDUINO_HW_CDC_RX_EVENT, on_receive);
  USBSerial.begin(9600);
}

void loop() {

}
```

By using analogWrite, we can vary the speed of the motor unlike the DC Motor, which can only turn it on or off.

6) Now we implement functionality for serial communication on the GUI side of things using Serial library.
    a) Before we continue, we need to make sure we spawn detached threads when our callback functions execute. If we don't, then our UI will freeze if our serial code gets stuck since we use the main thread.

    First, we import the necessary libraries. Include this at the top of your program to import the necessary libraries: '`from threading import Thread, Lock`'

    Then include this function outside the classes, which allows us to spawn detached threads:

```
65  def detached_callback(f):
66      return lambda *args, **kwargs: Thread(target=f, args=args, kwargs=kwargs).start()
67
```

    b) We can now implement our Locked Serial Class to wrap serial in a :

```
65  class LockedSerial(Serial):
66      _lock: Lock = Lock()
67
68      def __init__(self, *args, **kwargs):
69          super().__init__(*args, **kwargs)
70
71      def read(self, size=1) -> bytes:
72          with self._lock:
73              return super().read(size)
74
75      def write(self, b: bytes, /) -> int | None:
76          with self._lock:
77              super().write(b)
78
79      def close(self):
80          with self._lock:
81              super().close()
```

    Now we need to change every instance of Serial to LockedSerial.

With these changes implemented, dragging the slider should drive the DC motor.

Part 03: Adding a Buzzer

Introduction

With an imagination board, we will drive a DC Motor and buzzer with varying speed and volume using a slider on a GUI.

Objective

- Create a GUI with a slider to drive the sound of the buzzer and speed of the DC Motor
- Interface a DC Motor and buzzer with an Imagination board

Background Information

Briefly explain the technical skills learned/needed in this challenge. There is no need to go into detail as a separation document should be prepared to explain more in depth about the technical skills

In this challenge, you will learn how to add a buzzer that responds to the slider you made in Part 1. Additionally, the frequency of the buzzer will correspond to values from slider.

Components

- Imagination Board
- Breadboard
- Piezzo Buzzer
- PC/Laptop

Instructional

Hardware:

1) Connect a wire to IO46 on the Imagination board (see the back of the Imagination board to locate this pin)
2) Connect this same wire to the breadboard

3) Connect the positive side of the buzzer to the wire used in the previous two steps on the breadboard (the long pin is positive)
4) Connect a wire to GND (ground) on the Imagination Board (B10 or A1)
5) Connect the same wire on the ground of the buzzer (short pin) on the breadboard

Software:

Add a constant for the port you've connected the buzzer to

```
const int BUZZER = 46;
```

The constant value depends on the port so the value may not be 46

After we get the motorSpeed, we create a variable to handle the frequency the buzzer will emit in the on_receive() function.

```
int buzzerFreq;
```

We will use the map() function in Arduino to map motorSpeed values to a range of frequencies (we will use the frequency range of 2000-5000Hz) and assign the result of this function to buzzerFreq.

```
int buzzerFreq = map(motorSpeed, 0, 255, 2000, 5000);
```

The reason we do this is because the buzzer operates on the frequency range while the dc motor only functions in 256 values; 0 to 255 (8 bits of information if you will). As such, we normalize motor values to the frequency range of the buzzer so that the buzzer will emit a sound based on the value of motorSpeed (this value is the value of the slider). Consequently, the higher the motorSpeed value, the higher the pitch the buzzer's sound will be.

Next, we use the tone() function to send buzzerFreq to the pin the buzzer is connected to.

```
tone(BUZZER,buzzerFreq);
```

In setup(), we set the buzzer pin as output

```
pinMode(BUZZER, OUTPUT);
```

Example

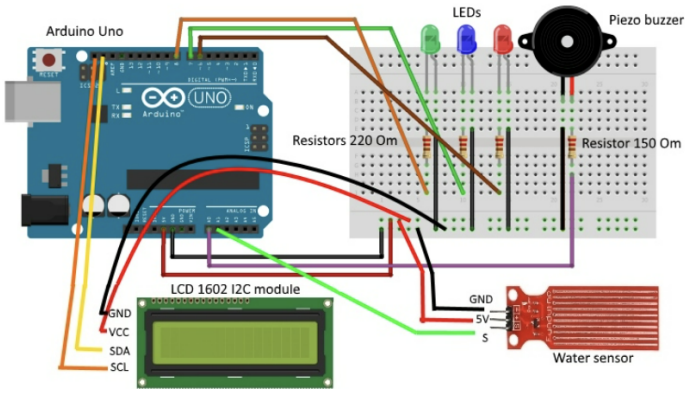
Introduction

[Water Level Indicator]

[https://acoptex.com/wp/arduino-guide-for-water-level-sensor-leds-lcd1602-i2c-module-and-piezo-buzzer/]

With the incorporation of a Piezzo Buzzer, one simple project to implement the skills learned in this tutorial is a water level indicator. This is an example of a practical application of the Piezo Buzzer, which reinforces the understanding of the integration of programming with electronic components to create useful. A water level indicator aids in providing a clear indication of the water level, which is prevalent for tanks or reservoirs. This aids the user to practice water conservation

Example



This figure depicts the circuit setup required to fabricate a water level indicator.

Analysis

Explain how the example used your tutorial topic. Give in-depth analysis of each part and show your understanding of the tutorial topic

Additional Resources

Useful links

List any sources you used, documentation, helpful examples, similar projects etc.

The Arduino reference is useful for understanding the syntax used in this tutorial as well as other projects that use Arduino

https://www.arduino.cc/reference/en/

This slider tutorial provides an overview of how a slider can be made. Of course, this slider is not exactly the same slider as in this tutorial but the idea is the same.

https://www.pythontutorial.net/tkinter/tkinter-slider/

https://randomnerdtutorials.com/esp32-dc-motor-l298n-motor-driver-control-speed-direction/

https://acoptex.com/wp/arduino-guide-for-water-level-sensor-leds-lcd1602-i2c-module-and-piezo-buzzer/