LE/ESSE 2220 Algorithmic and Computational Methods

Lab 3: Planet Sounds with PWM

(Fall 2025-2026)

Z. ARJMANDI (ZARARJ@YORKU.CA)

YORK U

Review

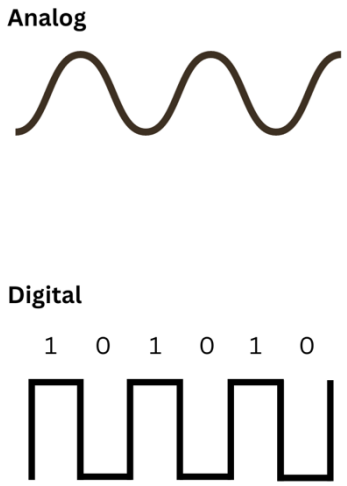
Raspberry Pi GPIO – Inputs & Outputs

Inputs (read, sense)

- **Digital Input (0 / 1, LOW / HIGH)**
  - Button, switch, motion sensor
  - Data type: **Boolean (True/False)**
- **Analog Input** (needs external ADC)
  - Temperature sensor, potentiometer
  - Data type: **Integer/Float** (e.g., 0–1023)

Outputs (write, interact)

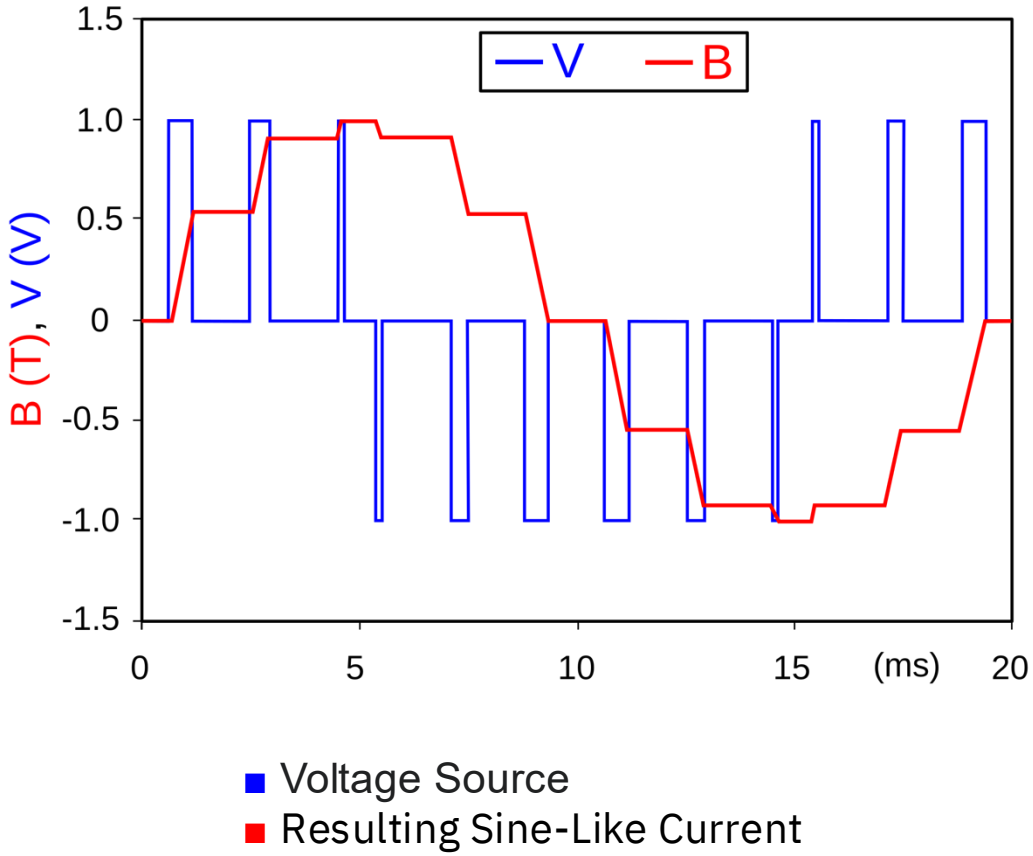
- **Digital Output (0 / 1, OFF / ON)**
  - LED, buzzer, relay
  - Data type: **Boolean (True/False)**
- **PWM Output (Pulse Width Modulation)**
  - LED dimming, motor speed control, servo angle
  - Data type: **Duty Cycle % (0–100)**



YORK U

Why Do We Need PWM?

- The board gives us **fixed voltages** (5V and 3.3V).
- But devices (LEDs, motors, buzzers) often need **variable power**.
- **PWM (Pulse Width Modulation)** solves this:
  - Switches the pin ON and OFF very fast.
  - The **duty cycle** (ON-time %) controls the *average* power.
- **Examples:**
  - LEDs → brightness
  - Motors → speed
  - Buzzers → sound tone & volume



YORK U

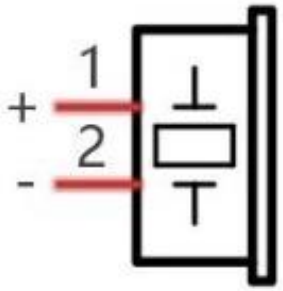
Lab 3

Lab 3

- Buzzer
  - Doorbell (6.1.1) Active Buzzer
  - Alertor (6.2.1) Passive Buzzer

Lab 3: Planet Sounds with PWM

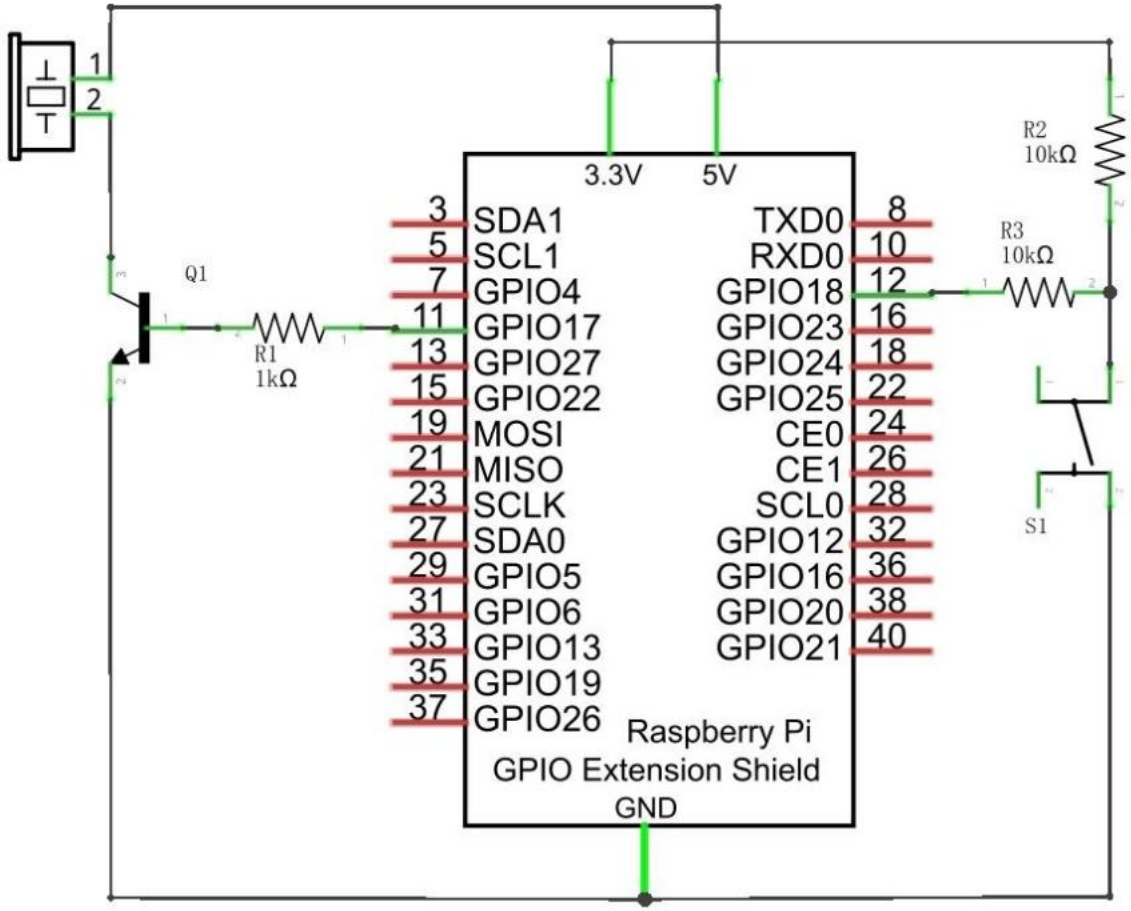
Passive and Active Buzzers



YORK U

Lab 3: Planet Sounds with PWM

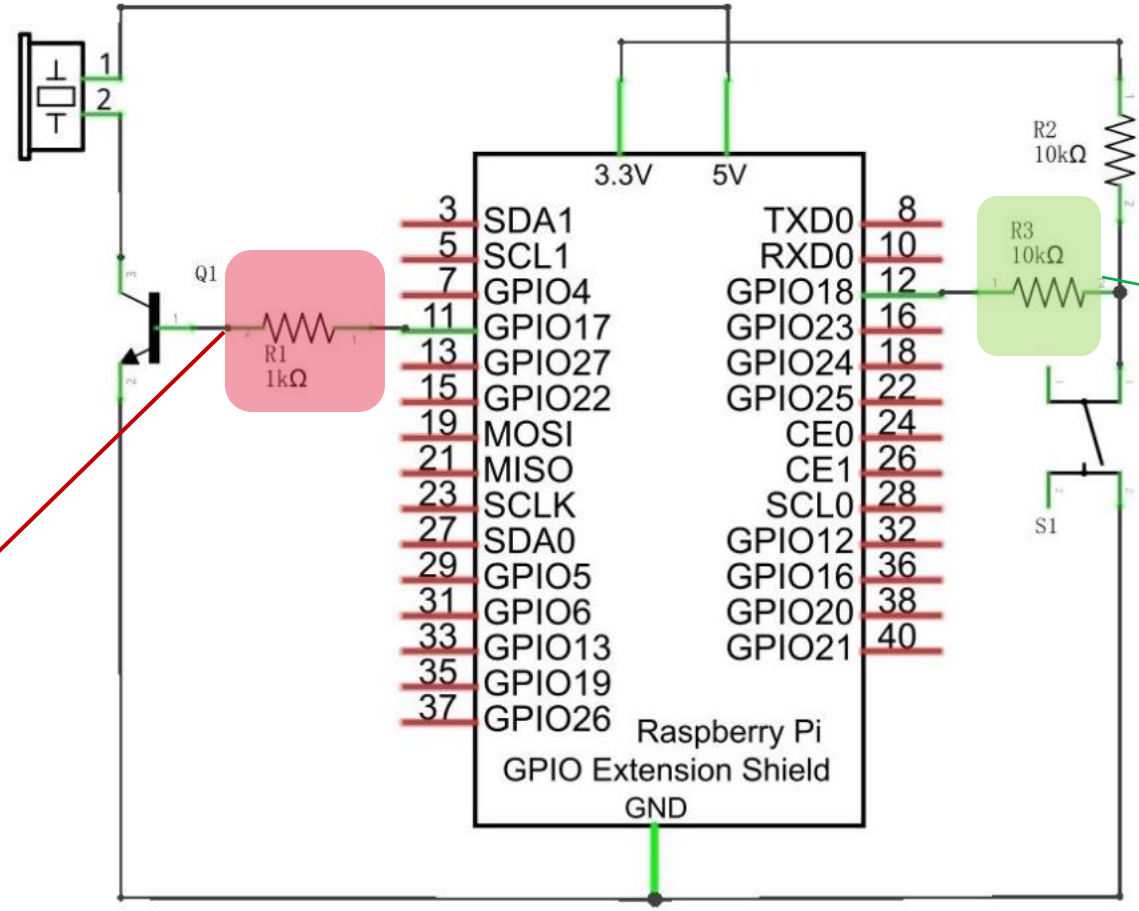
- For more details on this circuit, please refer to Tutorial Chapter 6 (available on E-Class).



YORK U

Lab 3: Planet Sounds with PWM

- For more details on this circuit, please refer to Tutorial Chapter 6 (available on E-Class).



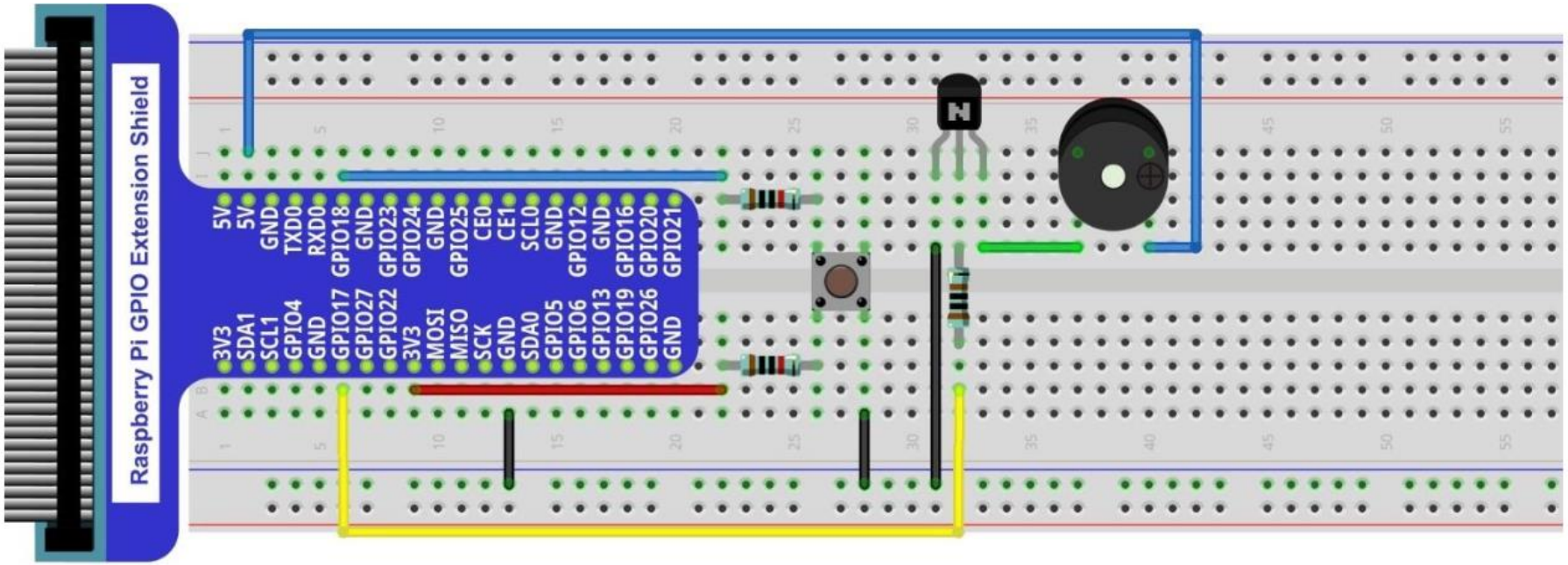
GPIO Output
Works like a **switch**
(turns devices ON/OFF).

GPIO Input
Works like a **voltmeter**
(detects voltage levels).

YORK U

Lab 3: Planet Sounds with PWM

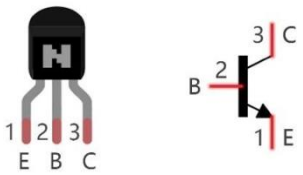
- Build the Circuit

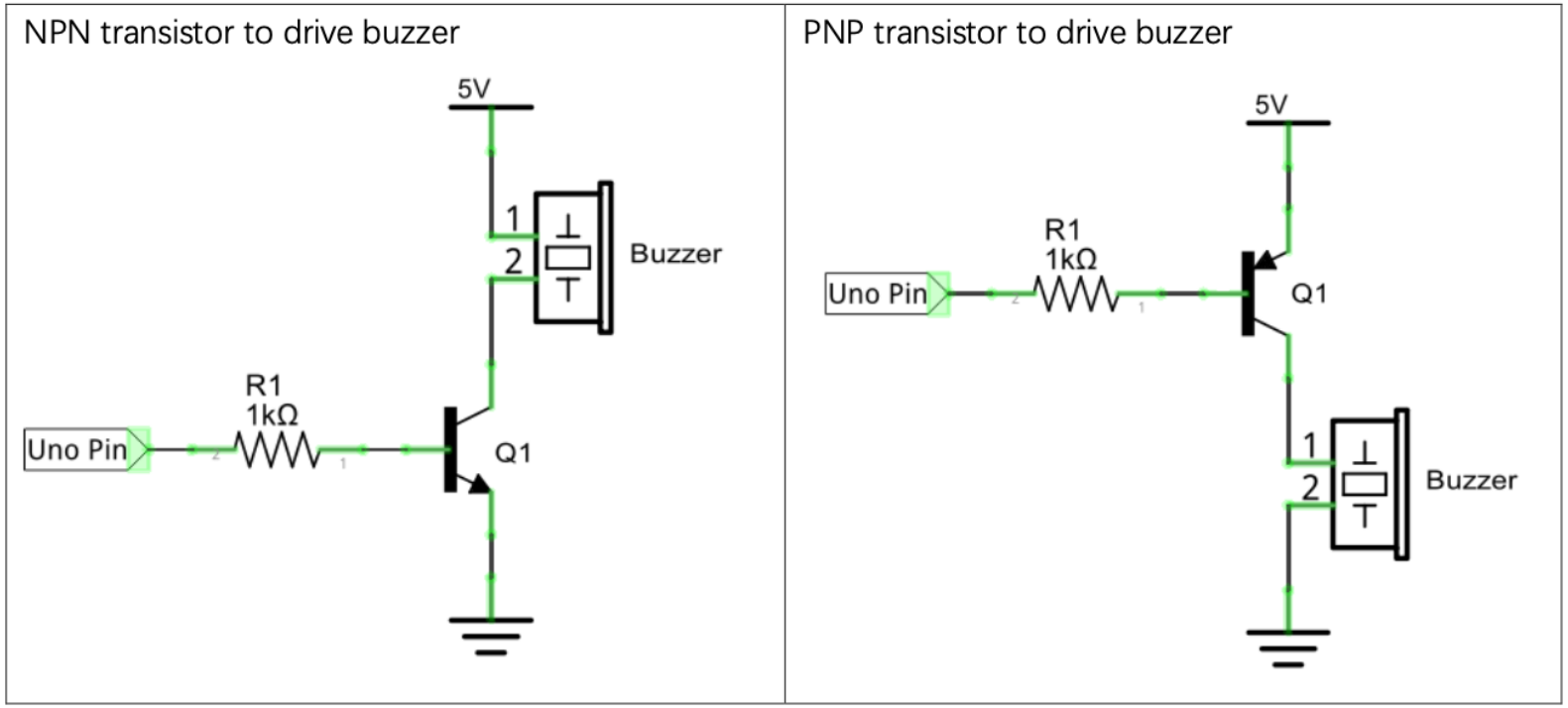


Component knowledge: Transistors

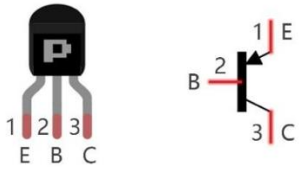
- Transistors: Amplify weak signals, or to work as a switch.
- **We use the GPIO pin to send a control signal to the transistor, which then allows current to flow through the buzzer.**
- (PINs): base (b), collector (c) and emitter (e).

NPN transistor





PNP transistor



Current flows when a small current enters the base (B).

Current flows when a small current leaves the base (B).

YORK U

Component knowledge: Buzzers

Active buzzer bottom

Passive buzzer bottom

Has a built-in oscillator.
Just give it a **DC signal** (HIGH/LOW) → it makes a fixed tone.
Easy to use, but **tone cannot be changed**.

No built-in oscillator.
Needs an **AC signal** (sound frequency).
We use **PWM (Pulse Width Modulation)** to generate different tones.

YORK U

Lab 3: Planet Sounds with PWM

- **Run Doorbell**
  - Use the **active buzzer** in your kit.
  - Make sure it works as a simple doorbell (ON/OFF sound).
- **Switch to Passive Buzzer**
  - Replace the active buzzer with the **passive buzzer** from the box.
  - Run the provided alertor code.
  - Verify that you hear the **siren-like sweep** when pressing the button.
- **Planet Siren Upgrade**
  - Create a new file planet_tones.py with a **class** (__init__ + method).
  - In your main program:
  - Ask for a planet name.
  - Use the class to store that planet's **base** and **depth** values (from the table).
  - Inside the alertor() loop, call your class method to generate the tone.
  - Each planet must produce a **different siren voice**.

- **Bonus Challenge: Planet Sound Patterns:**
  - Experiment with different **sine patterns for different planets**:
    - Multiply the angle → sin(2 * angle) (faster wobble).
    - Use cos(angle) instead of sin(angle).
    - Try adding both: sin(angle) + cos(angle).

Hints to Get Started (Optional)

What to keep

- The for x in range(0, 361): loop must stay.
- The loop still calculates a sine value (sinVal).
- The buzzer frequency is still updated with p.ChangeFrequency(...).

What to change

- **Remove this line in alertor():**

  toneVal = 2000 + sinVal * 500

- **Replace it with a call to your class**

```
def loop():
    while True:
        if GPIO.input(buttonPin) == GPIO.LOW:
            alertor()
            print('alertor turned on >>>')
        else:
            stopAlertor()
            print('alertor turned off <<<')

def alertor():
    p.start(50)
    for x in range(0, 361):  # sweep through a sine wave
        sinVal = math.sin(x * (math.pi / 180.0))
        toneVal = 2000 + sinVal * 500  # frequency swings up and down
        p.ChangeFrequency(toneVal)
        time.sleep(0.001)
```

YORK U

Planet Table for Siren Experiment

- Use these values for your class.
- Each planet has a **base frequency** and a **depth** (range of the sweep).

```
planet_table = {
   "MERCURY": (1500, 300),
   "VENUS":   (1800, 400),
   "EARTH":   (2000, 500),
   "MARS":    (2200, 600),
   "JUPITER": (2500, 700),
   "SATURN":  (2700, 800),
   "URANUS":  (2900, 900),
   "NEPTUNE": (3100, 1000),
}
```

Part 3 Report Format (short, personal, verifiable)

Setup
- Attach a clear **photo of your circuit** (showing the button and buzzer connected).
- Copy and paste your final code (main program and planet_tones.py).
- Include **meaningful comments** in your code explaining each part.

Demo
- Ask your TA or instructor to come and check your code running on the hardware.
- Show how the buzzer changes sound depending on the planet you choose.

Observations
- Describe how the **base frequency** and **depth** made each planet's "voice" sound different.
- Did the siren feel higher, lower, faster, or slower for different planets?

Analysis
- Explain your **class** (PlanetTone) in your own words:
- What values does it receive (__init__)?
- What does the method return when given an angle?
- Why was using a **class** useful here instead of hardcoding the numbers in the main file?
- How does the alertor() loop use your class to make the buzzer sound different?

Bonus (Optional)
- If you experimented with different sine patterns (sin(2x), cos(x), etc.), explain what effect it had on the sound.

Rubric

- **Planet Siren (10 pts)**
- **Circuit setup**
  - (clear photo of wiring, pins explained, safe connections — no risk of blowing up components) – 2 pts
- **Code and Demo**
  - (runs correctly + meaningful comments included) – 1 pts
  - (show the buzzer working on hardware in front of TA/instructor) – 3 pts
- **Observations** – 2 pts
- **Analysis** (class explained: what it stores, what the method does, how it's used in the loop) – 2 pts
- **Bonus (optional, up to +2 pts)**
  - Extra planet added, or new sine pattern tried (sin(2x), cos(x), etc.) – +2 pts
- **Due Date: (Lab2 and Lab3)**
  - Monday, 11:59 PM
- **Quiz 3**
  - Release: Monday, 11:59 PM
  - Deadline: Friday, 12:00 PM (before labs)