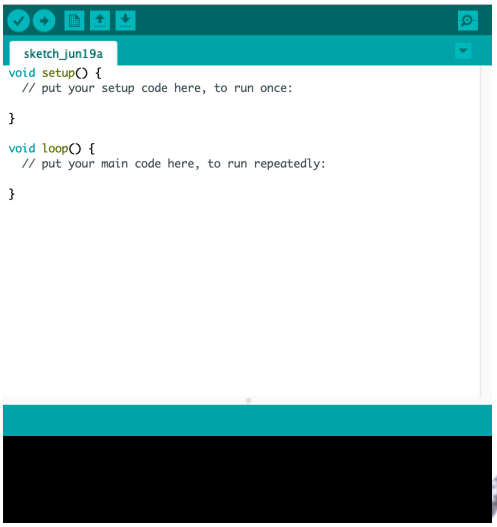
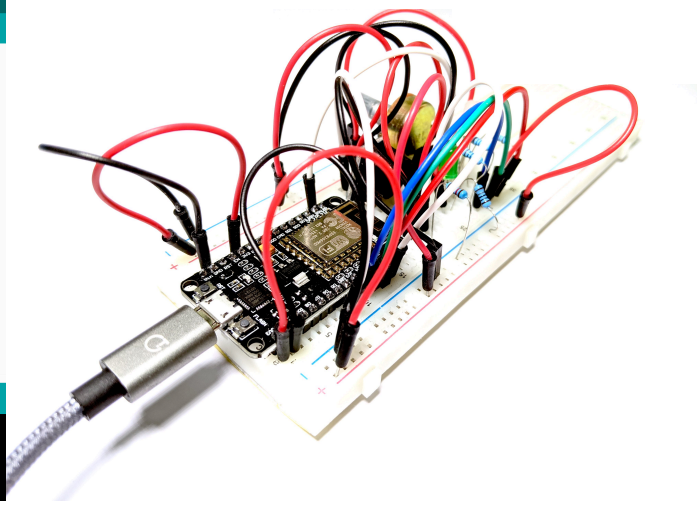
Smart Fridge

Part two: Adding more components

Project description:

In the first part of the smart fridge exercise, you started to build the circuit and the code to simulate some functionalities of a smart fridge. In this project, you will build on the previous exercise and add few more components to your smart fridge. For instance, you will use a RGB LED component to visualise the temperature of the fridge and a buzzer to signal when such temperature reaches critical values. The circuit is of course just a simulation of real smart fridge and you are not expected to create the final product. The project will get you familiar with a new digital input/output component: the buzzer.



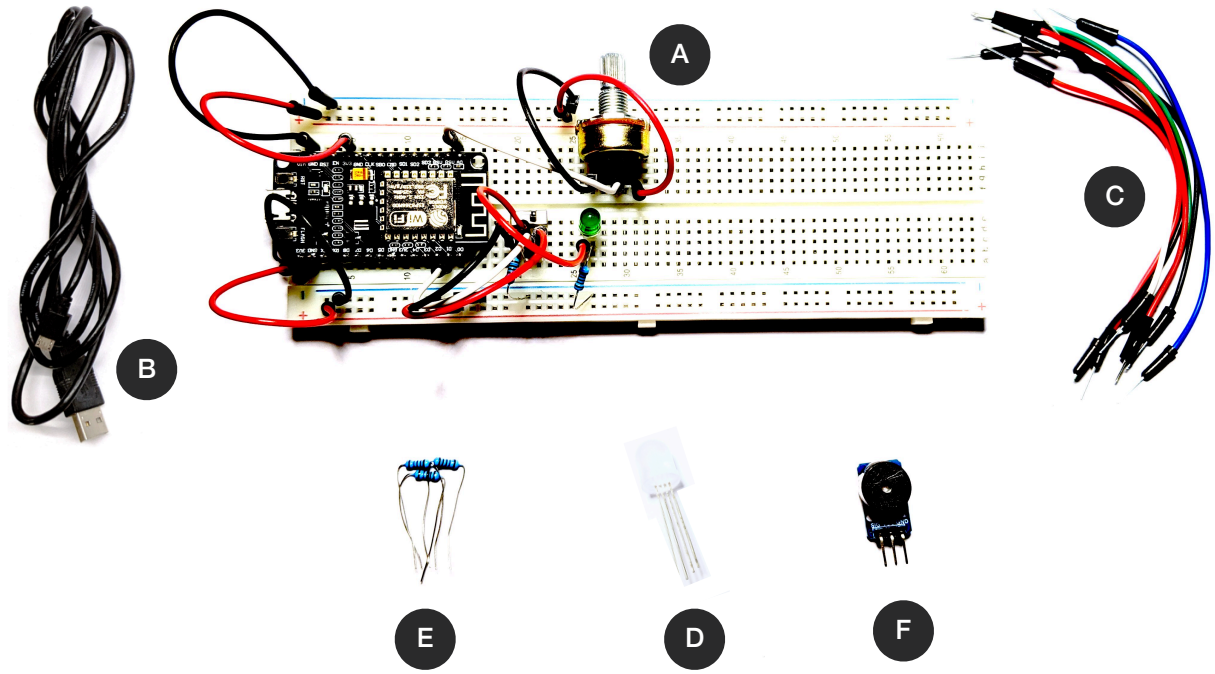


Project objectives:

- Expand your knowledge on digital and analog Input/Output pins
- Explore the buzzer component
- Use resistors on the RGB LED component

Project components:

Component Reference	Component Quantity	Component Name	Component Description
A	1	Smart Fridge Part 1	The smart fridge circuit which features the potentiometer and the switch to read and control the temperature
B	1	Micro USB Cable	A USB cable to power and upload instructions to a microcontroller
C	7	Jumper Wires	Conductive cables frequently used with a breadboard to connect two points in a circuit
D	1	RGB LED	A patch of three light sources that emits light when current flows through it
E	3	220 Ohm Resistors	A passive two-terminal electrical component that implements electrical resistance as a circuit element
F	1	Buzzer	A loudspeaker that uses the piezoelectric effect for generating sound



Step One
Introduction and Theory

In the first part of the smart fridge exercise, you encountered some interesting components such as the potentiometer and the electrical switch. This exercise introduces you to one new component, the buzzer, and re-proposes the RGB LED component that you used for your smart chair circuit.

As a process to exploring the new component, you will build on the first version of the smart fridge circuit. The idea here is to add a buzzer to signal critical temperature values and a RGB LED to visualise the temperature. The buzzer will produce a sound when you set the wrong temperature on your fridge. The RGB LED will turn full red when the fridge temperature is high and full blue when the fridge temperature is low.

You will need both the circuit and the code for the smart fridge part one to complete this exercise.

Let's briefly discuss the additional components for the circuit:

- **RGB LEDs:** An RGB LED component is a combination of three individual LEDs, respectively a red LED, a green LED, and blue LED. Combining together these three sources of light and varying their intensity allow to have one single component to produce multiple coloured lights. The component has usually four leads: the ground (GND), the red LED (R), the green LED (G), and the blue LED (B). You will use this component to visualise the fridge temperature (blue if cold and red if hot).
- **Buzzer:** The buzzer, or piezo speaker, is a simple device that can generate basic beeps and tones. It works by using a piezo crystal that changes shape when voltage is applied to it. The component has three leads: the ground (GND), the source (VCC), and the signal (SIG). You will use this component to make a sound when the fridge reaches critical temperatures.

Step Two
Wiring up the Smart Fridge Circuit

The circuit assembly for this exercise requires you to wire up 1 RGB LED, 3 resistors, and 1 buzzer.

Differently from the RGB module that you encountered for the smart chair circuit, the RGB LED that you will wire for this circuit does not have built-in resistors. This means that for each individual LEDs, you will need to add one resistor to allow the correct amount of current to flow.

The RGB LED component should be connected to individual digital input/output pins of your ESP board. This will allow you to control them individually when writing the code for the circuit. The digital pins for the RGB LED component are D2 (red), D3 (green), D4 (blue), as well as ground (GND). The longer lead of the RGB LED usually represents the ground lead and should be connected to GND of your ESP board. **Notice that sometimes the longer lead represents the VCC and should be connected to 3V (this is the case of the RGB LED in your kit).** The remaining three leads are the red, green, and blue LEDs and should be connected to pins D2, D3, and D4 respectively. Additional resistors should be connected in series with these three leads to limit the current for each LED (red, green, blue).

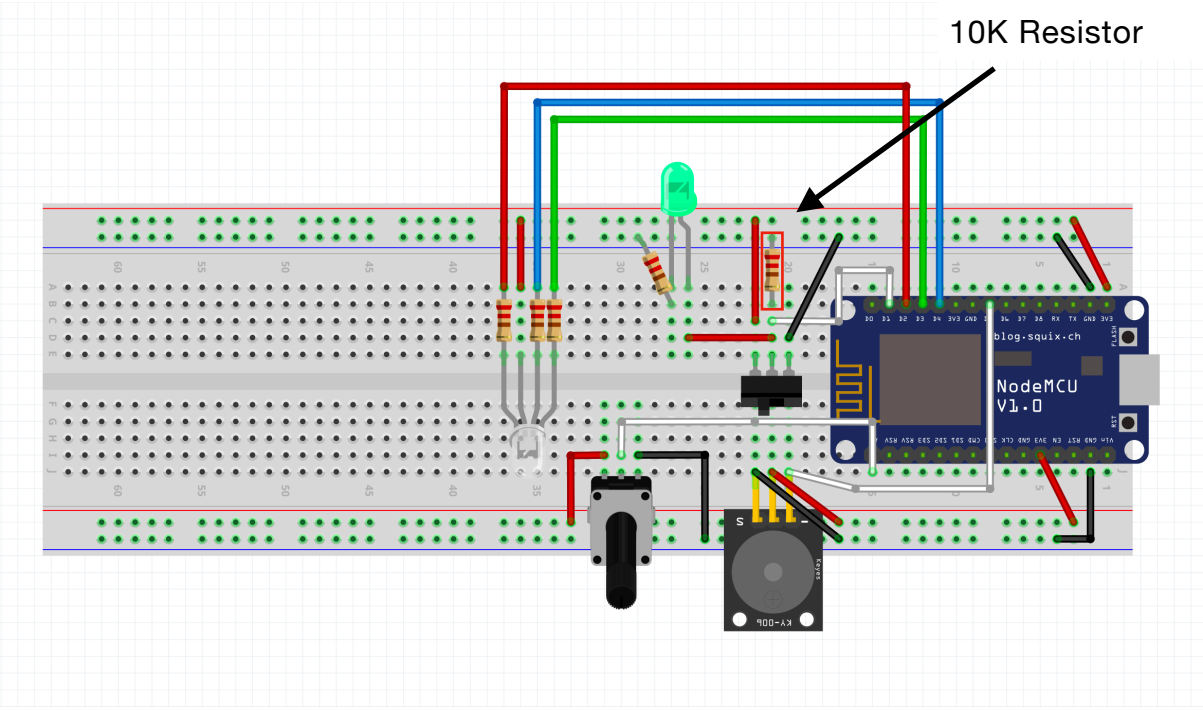
Finally, the buzzer should be connected to digital pin D5 of your ESP board. This component has three leads. You should connect the VCC lead to the source (3.3V) and the GND lead to ground. The middle lead is the signal and it should be connected to digital pin D5.

Here a quick recap of the circuit wiring:

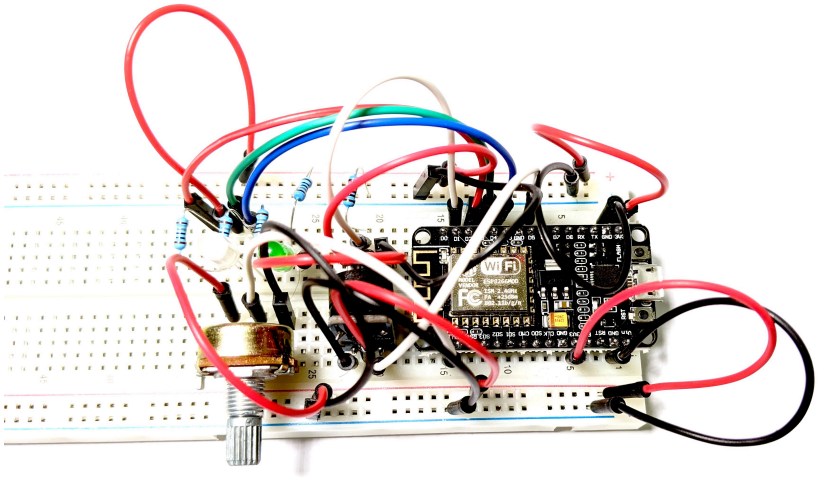
- **RGB LED:** It should be wired up to pins D3, D4, D5. The longer lead goes to VCC (3V). Additional 100ohm resistors should be connected in series with the red, green, and blue LED leads.
- **Buzzer:** The middle lead of the buzzer (SIG) should be connected to digital pin D5. The GND lead goes to ground and the VCC lead goes to 3.3V.

It is now your turn to add the additional two components to the smart fridge circuit. Use the same half of the longer breadboard to build this circuit. We will reserve the other half for upcoming projects.

The diagram below shows you how the circuit should be assembled. It combines the circuit components for the first part of the smart fridge circuit with the additional RGB LED and the buzzer components. **Notice that some RGB LEDs might have a different configuration and require an alternative connection (the longer lead to GND instead of VCC). Also, the colour leads order might be different on some RGB LEDs:**



Furthermore, see below a picture of the circuit assembled:



Step Three
Writing the Code

Now that you have assembled the circuit, it is time to add the code for the buzzer and the RGB LED components. Go ahead and create a new empty sketch from the Arduino IDE.

You should see an empty sketch like the following:

Feel free to save the sketch and rename it to something sensible: **smart_fridge_part2** for instance.

Now copy and paste the code that you wrote for the first part of the smart fridge exercise. You should have the code with the **fridgeTemperature()** and the **fridgeOn()** utility functions that allow you to turn ON and OFF the fridge and also read the temperature value from the potentiometer.

There are two core functionalities that we want to add to the program here:

- Use the buzzer to produce a sound when the fridge reaches a critical temperature.
- Use the RGB LED component to visualise the fridge temperature.

Let's begin by adding other useful variables just before the setup function:

```
// Initialise the critical minimum and maximum temperature of the fridge
int criticalMinTemp = 2;
int criticalMaxTemp = 4;

// Initialise the buzzer pin
const int buzzer_pin = D5;

// Initialise the RGB pins
const int red_led_pin = D2;
const int green_led_pin = D3;
const int blue_led_pin = D4;
```

Type the above code just before the setup() function.

Here we initialise all the variables needed to replicate the additional smart fridge behaviour. **criticalMinTemp** and **criticalMaxTemp** store the ideal temperature range for your fridge. **buzzer_pin** is a reference to the digital pin D5. **red_led_pin**, **green_led_pin,** and **blue_led_pin** are references to the digital pins D2, D3, and D4.

Next we want to add some code to the setup() function:

```
// Set the buzzer to OUTPUT
// you want to set the state here
pinMode(buzzer_pin, OUTPUT);
// No sound initially
digitalWrite(buzzer_pin, LOW);

 // LEDs as OUTPUT
pinMode(red_led_pin, OUTPUT);
pinMode(green_led_pin, OUTPUT);
pinMode(blue_led_pin, OUTPUT);
```

Here we we want to set the RGB LED pins mode to OUTPUT (pinMode) as we want to send out a signal to either turn the red, green, or blue LEDs ON or OFF. Similarly, we want to set the buzzer pin mode to OUTPUT as we want to turn it ON or OFF. We also want to make sure that the buzzer does not make any sound at the beginning (digitalWrite LOW).

pinMode: Set the pin mode (<pin number>, <mode>). The first argument is the pin reference and the second argument is the mode, either INPUT (read signal from digital pin) or OUTPUT (send signal to digital pin).

At this point, we have set the pin mode for our components. We can now use some additional utility functions to code the buzzer and RGB LED fridge functionalities.

First we want to create a function which allows us to determine when we the buzzer should alert for critical temperatures.

The **trigBuzzer()** function:

```
// Utility function to trigger the buzzer
void trigBuzzer(){

  // Check if your fridge temperature is acceptable
    if(fridgeTemperature() < criticalMinTemp || fridgeTemperature() > criticalMaxTemp){
       // Critical temperature, make a sound
       tone(buzzer_pin, 1000);
    }else{
       // Good temperature, no sound
       noTone(buzzer_pin);
    }

}
```

Type the above code after the fridgeTemperature() function.

The **trigBuzzer()** utility function checks if the temperature level is outside the critical levels. If the temperature is below the minimum temperature that we set or the temperature is above the maximum temperature that we set, the buzzer produces a sound. Otherwise, our fridge is within the recommended temperature and the buzzer should make no sound.

tone: Signal the buzzer (<pin number>, <frequency>). The first argument is the pin reference and the second argument is the frequency in hertz.

noTone: Stops the buzzer (<pin number>). The only argument is the pin reference.

Next, we want to use the fridge temperature reading to change the light of the RGB LED. The RGB LED should turn blue between the fridge minimum temperature (-2) and 0, and red between 0 and the maximum temperature (6).

The **temperatureStatus()** function:

```
// Utility function to signal the temperature status
void temperatureStatus(int fridgeTemp, bool fridgeOn){

   // Turn off the RGB led (inverse mapping)
   if(!fridgeOn){
     analogWrite(red_led_pin, 255);
     analogWrite(green_led_pin, 255);
     analogWrite(blue_led_pin, 255);
     return;
   }

   // Turn on the red value of the RGB led (inverse mapping)
   if (fridgeTemp >= 0){
     int red_value = map(fridgeTemp,0,maxTemp,255,0); // mapping the red value 0 - maxTemp to 255 - 0
     analogWrite(red_led_pin,red_value );
     analogWrite(green_led_pin, 255);
     analogWrite(blue_led_pin, 255);
   }else{
     int blue_value =map(fridgeTemp,minTemp,0,0,255);  // mapping the blue value minTemp - 0 to 0 - 255
     analogWrite(red_led_pin, 255);
     analogWrite(green_led_pin, 255);
     analogWrite(blue_led_pin,blue_value );
   }

}
```

Type the above code after the trigBuzzer() function.

The **temperatureStatus()** utility function takes two parameters: the fridge temperature and whether the fridge is ON or OFF.

First, we check if the fridge is OFF. If it is, we do not want to show any lights on the RGB LED and we set all individual red, green, and blue components to 255 (the analogWrite value is inverted on RGB LEDs with the longer lead connected to VCC). A value of 0 means full light and a value of 255 means OFF. We also return from the function as no further action is needed.

Next we address two cases when the fridge in ON. The first one is when the temperature is above or equal to 0. Here, we map the temperature to a range of 255 - 0 and we use it to light up the red component of the RGB LED. The second one is when the temperature is below 0. Here, we map the temperature to a range of 255 - 0 and we use it to light up the blue component of the RGB LED. This results in a transition from full blue, at the fridge minimum temperature, to full red at the fridge maximum temperature.

Now that we have all the utility functions in place, it is time to put the code together in the **loop()** function to simulate the complete smart fridge functionalities. Remember, we want to first check if the fridge is ON. If it is, we want to use the serial monitor to read the temperature value, light up the RGB LED component according to the temperature, and activate the buzzer on critical temperatures. If the fridge is OFF, we want to turn OFF the RGB LED and stop the buzzer from making any sound.

The **loop()** function:

```
// put your main code here, to run repeatedly:
void loop() {

  // Only execute if the fridge is ON
  if (fridgeOn()){

    // Print the fridge temperature on the serial monitor
    Serial.println(fridgeTemperature());

    // Signal temperature status
    temperatureStatus(fridgeTemperature(), fridgeOn());

    // Signal critical temperatures
    trigBuzzer()

  }else{
    // Signal temperature status (OFF)
    temperatureStatus(fridgeTemperature(),fridgeOn());
    // No sound, fridge is off
    noTone(buzzer_pin);
  }
}
```

The **loop()** function uses all our utility functions to replicate the smart fridge behaviours. First, it checks if the fridge in ON (the switch component). If the fridge is ON, the mapped temperature is printed on the serial monitor. Additionally, the RGB LED is set to the correct colour according to the fridge temperature and the buzzer is triggered if the temperature reaches critical values. On the other hand, if the fridge is OFF, the RGB LED turns OFF and the buzzer stops from making any sound.

Hurray, you have successfully completed the code section of the circuit.
Now you have a more advanced smart fridge circuit.
Go ahead, compile your code, and upload it to your ESP board.

Now open the serial monitor.

When you turn on the switch, you should see the green LED light up. Furthermore, the serial monitor should output the fridge temperature in the range of -2 and 6. Rotating the potentiometer pole should output a different temperature inside the serial monitor.

Rotating the potentiometer should also change the RGB LED colour from full red to full blue depending on the temperature value.

If you turn the potentiometer and reach critical temperature values, the buzzer will make a sound and alert you.

Step Five
Additional Tasks

In upcoming exercises, you will add more components to the smart fridge circuit.

Meantime, try the following:

- Can you use the serial monitor to print “Too Cold” when the temperature reaches a low critical value and ‘Too Hot’ when the temperature reaches a high critical value?
- Check the buzzer documentation and change the sound that it makes when the temperature reaches critical values.
- Play around with the temperature critical values. Change the criticalMinTemp and the criticalMaxTemp values to trigger the buzzer at different ranges.