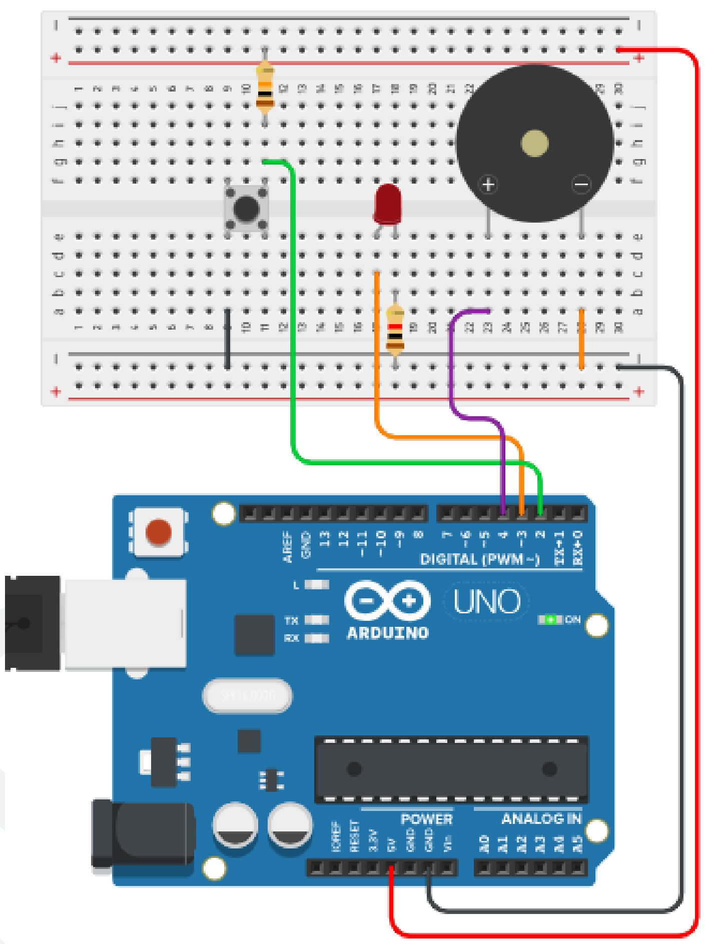
Activity: Make a Button Control Momentary and Toggle Output

Objective:



In the world of digital electronics, we don't just have Inputs and Outputs. Rather, we have different types of Inputs and Outputs. There are Active Low and Active High inputs, there are Level triggered and Edge triggered inputs, there are Momentary and Toggle inputs. And just like the inputs, we have different types of outputs as well.

We start with the simplest of them all, the Momentary and Toggle inputs. As the names suggest one of them is only active momentarily (momentary) while the other switches (toggles) between On and Off. Now, don't get confused between these technical jargons. The simplest way to remember them is by using an example of each type. Momentary means switch of a Doorbell and Toggle means switch of a Lamp. Now you immediately know what the difference is.

1. **Bell Switch:** Press to On, Release to Off.
2. **Lamp Switch:** Press once to On , Press again to Off.

So, in today's activity we will learn how to use a simple push button switch both as Momentary Input and as a Toggle Input. The same Push button will work once as a Bell switch while the other time as a Lamp switch by changing just the code.

This activity will be divided into 2 parts.

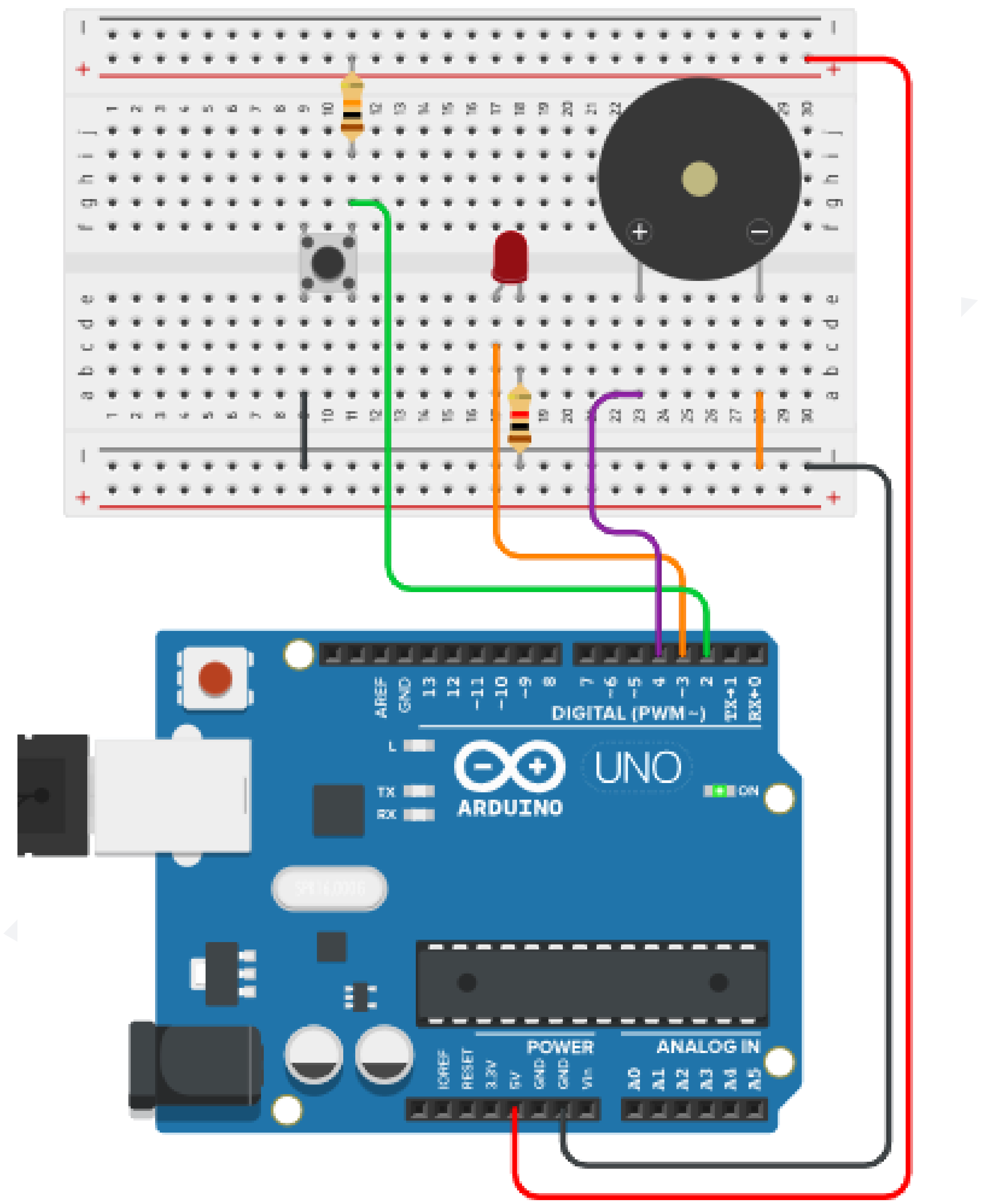
1. Controlling the Buzzer and the LED with the Push Button working as the Bell switch i.e. Press to On and Release to Off.
2. Controlling the Buzzer and the LED with the Push Button working as the Lamp switch i.e. Press once to On and Press again to Off.

Materials Required:

S.no.	Name	Qty	Image
1	Arduino (Nano or UNO)	1	
2	Breadboard	1	
3	M - M jumper wires	10	
5	USB cable for Arduino Nano	1	

6	Resistors: 10k (Brown Black Orange)	1	
7	5v Micro Buzzer	1	
8	4-pin tact switch	1	
9	5mm Red LED	1	
10	1k resistor	1	

Connection Diagram:



Explanation:

The above connection diagram shows the Red LED, the push Button and the Buzzer are all connected to the Arduino with the different values of resistors and connection wires.

Although the overall circuit appears to be visually complex, it is quite easy to understand if we take one connection at a time. Here it is.

1. The Red LED is connected such that its +ve terminal is connected to the Arduino digital pin 3 (D3) and the -ve terminal is connected to one side of the 1k resistor. The other side of the 1k resistor is connected to the -ve of the breadboard.

2. The Buzzer is connected such that its +ve terminal is connected to the Arduino digital pin 4 (D4) and the -ve terminal is connected to the -ve of the breadboard.

3. The push Button is connected such that its top-right pin is connected to the +ve of the breadboard through a 10k resistor acting as a pull-up and the same pin is also connected to arduino digital pin 2 (D2). The bottom-left pin is connected to the -ve of the breadboard.

Note: the Connection Diagram remains the same for Part 1 as well as Part 2.
Part 1: the push Button acting as Momentary input.

Arduino Code:

Using the Button as a Momentary input is something that we have already done earlier but we didn;t call it that back then. Remember we already controlled an LED and then a bunch of LEDs with a push Button. That was all Momentary behavior. The LEDs stayed On as long as the Button stays pressed and stayed Off as long as the Button stays released.

All we need to do here is use a digitalRead on the pin connected to the Button and then digitalWrite the inverse of its State to the LED and the Buzzer.

Here is the Arduino code.

```
#define Buzzer 4               // defining pin 4 as "Buzzer"
#define Button 2               // defining pin 2 as "Button"
#define Red    3               // defining pin 3 as "Red"

int Button_state;                  // state variable for button

void setup()
{
 pinMode(Buzzer, OUTPUT);     //declaring "Buzzer" as output
 pinMode(Red, OUTPUT);        //declaring "Red" as output
 pinMode(Button, INPUT);      //declaring "Button" as input
}

void loop()
{
  Button_state=digitalRead(Button); //check if button pressed

  digitalWrite(Buzzer, !Button_state); // Buzzer turned ON
  digitalWrite(Red, !Button_state);   // Red LED turned ON
}
```

Explanation:

```
#define Buzzer 4               // defining pin 4 as "Buzzer"
#define Button 2               // defining pin 2 as "Button"
#define Red    3               // defining pin 3 as "Red"
```

The first three lines are simply defining which pins are going to be used as what. This makes it easier for the coder or user to refer to the pins in a more human understandable form. So, instead of referring to them using numbers, we give them names such as Buzzer, Red (LED) and Button.

```
int Button_state;
```

In this line we are declaring that we are going to use an Integer type variable with the name “Button_state”. A variable can be of many types depending on the value we want it to hold. It can be an Integer (number without a decimal point), a

Character (a-z or A-Z), a String (set of alphabets e.g. “robot” or “ROBOT” or “Robot”) etc.

```
void setup()
{
 pinMode(Buzzer, OUTPUT);     //declaring "Buzzer" as output
 pinMode(Red, OUTPUT);        //declaring "Red" as output
 pinMode(Button, INPUT);      //declaring "Button" as input
}
```

Here we are declaring that the “Buzzer” will be used as an Output, the Red (LED) will be used as an Output and that the “Button” will be used as an Input. Note that, here, we are using “INPUT” instead of “INPUT_PULLUP”. This is because Pressing the Button makes D2 LOW and Releasing it makes D2 HIGH. This default HIGH state for pin D2 is provided here by the external 10k resistor acting as Pull-Up.

```
void loop()
{
  Button_state=digitalRead(Button); //check if button pressed

  digitalWrite(Buzzer, !Button_state); // Buzzer turned ON
  digitalWrite(Red, !Button_state);   // Red LED turned ON
}
```

Here, we are first reading the state of the Push Button and storing it in the variable “Button_state”. Pressed is stored as a 0 and Released is stored as a 1. And then we are turning the LED and the Buzzer On or Off by writing this 0 or 1 on the “Red” pin D3 and Buzzer pin D4.

Part 2: the push Button acting as Toggle input.

Arduino Code:

Now that we have already understood how a Push button works with a Buzzer, we move on to generate the Toggle effect using the same Buzzer, LED and Button setup. The Toggle effect needs to be understood mathematically first and then implemented programmatically in the Arduino code.

Toggle means that the output is inverted every time an Input is received. This simply means:

1. If the Output is initially LOW, an Input will make it HIGH.
2. If the Output is initially HIGH, an Input will make it LOW.

We first check if a button has been pressed or not and then use every detected button press to Increment a counter variable. Pressing and releasing the button once will turn the Buzzer On and pressing and releasing it again will turn the Buzzer Off. All this will be much clearer once we go through the code.

Here is the code for generating the Toggle effect with a Buzzer and a Push Button.

```
#define Buzzer 4             // defining pin 4 as "Buzzer"
#define Button 2             // defining pin 2 as "Button"
#define Red    3             // defining pin 3 as "Red"

int Button_state;                // state variable for button
int Counter = 0;             // Counter variable for Toggle


void setup()
{
 pinMode(Buzzer, OUTPUT);     //declaring "Buzzer" as output
 pinMode(Red, OUTPUT);        //declaring "Red" as output
 pinMode(Button, INPUT);      //declaring "Button" as input
}

void loop()
{
  Button_state=digitalRead(Button); //check if button pressed

  if(Button_state == 0)            // if button is pressed
  {
    Counter = Counter + 1;         // increment Counter by 1
    delay(500);                   // wait for half a second
  }

  if(Counter == 1)                 // if button pressed once
  {
    digitalWrite(Buzzer, HIGH);    // Buzzer turned On
    digitalWrite(Red, HIGH);       // Red LED turned On
  }

  if(Counter == 2)                 // if button pressed again
  {
    digitalWrite(Buzzer, LOW);     // Buzzer turned OFF
    digitalWrite(Red, LOW);        // Red LED turned OFF
    Counter = 0;                   // reset Counter value
  }
}
```

Explanation:

The above code does the following:

1. Check whether the push button is pressed or not.
2. If the button is pressed, increment a Counter value by 1.
3. If the Counter value is 1, turn the Buzzer and the Red LED On.
4. If the Counter value is 2, turn the Buzzer and the Red LED Off and then reset the Counter value back to 0.

```
#define Buzzer 4             // defining pin 4 as "Buzzer"
#define Button 2             // defining pin 2 as "Button"
#define Red    3             // defining pin 3 as "Red"
```

These lines define that the Buzzer, the Button and the Red LED are connected to Arduino digital pins 4, 2 and 3 respectively.

```
int Button_state;            // variable to store Button status
int Counter=0;          // variable to store Counter value
```

Here, we have taken 2 variables, both integer types, one for storing the button status and the other for storing the Counter value. The Counter value has been initialized by 0.

```
 Button_state=digitalRead(Button);    // check if button is pressed
```

This is the line which actually checks the state of the button whether it is in the pressed state or released state and then stores that state in the “Button_state” variable. The way we have connected our push button, “digitalRead” of a pressed state will be 0 (LOW) while “digitalRead” of a released state will be 1 (HIGH).

```
  if(Button_state == 0)      // if button is in pressed state
  {
    Counter = Counter + 1;   // increment Counter by 1
    delay(500);              // wait for half a second
  }
```

These lines are first comparing if the “Button_state” variable has detected a “LOW” (a pressed state). And if this is true, then increment the Counter variable by 1 (which was initially 0) and then wait for half a second (500ms).

```
    digitalWrite(Buzzer, HIGH);    // Buzzer turned On
    digitalWrite(Red, HIGH);       // Red LED turned On
```

These above two lines are turning the Red LED and the Buzzer On.

```
    digitalWrite(Buzzer, LOW);     // Buzzer turned OFF
    digitalWrite(Red, LOW);        // Red LED turned OFF
```

These above two lines are turning the Buzzer and the Red LED Off.

Note that throughout the program, we have used “=” for assigning a value to a variable while “==” has been used for comparison and the comparison is always inside the braces of an “if” statement such as:

```
if(Counter == 1)
```

Here, we are comparing if the value of the variable “Counter” is equal to 1 or not. If it is 1, this “if” gives output as True. If, however, it is not 1, this “if” gives output as False.

```
Counter = 0;
```

Here, we are assigning the variable “Counter” the value 0.

Outcome and Observations:

1. For Part 1, once you are done compiling and uploading the code to the Arduino, you will observe the following:

    a. When you press the Button and keep it pressed, the Buzzer turns On and Stays On as long as the Button remains pressed.

    b. When you release the Button and it stays released, the Buzzer turns Off and Stays Off as long as the Button remains released.

    c. No matter how fast or slow you press the Button, the Buzzer stays On when the Button is pressed and stays Off when the Button is released.

    This type of behavior is called “Momentary” because the state of the output (the Buzzer) entirely depends on the state of the Input (the push button). Button On makes Buzzer On and Button Off makes Buzzer Off.

2. For Part 2, once you are done compiling and uploading the code to the Arduino, you will observe the following:

    a. Initially, the Buzzer remains Off.

    b. When the push button is pressed and released for the first time, the Counter value increments from 0 to 1 and plays the 1st beeping pattern on the Buzzer.

    c. When the push button is pressed and released for the second time, the Counter value increments from 1 to 2 and the Counter value is re-initialized to 0. This operation readies the entire setup to start over again.

    This type of behavior is called “Toggle” because the state of the output (the Buzzer) changes every time an Input (the push button) is received. Button On makes beeping On and Button Off makes beeping Off.