How to Make One Button Act Like Two or More with Arduino

Do you have an application where you want multiple buttons for different user inputs? Maybe you have a timer and you want one button for minutes and another for hours.

But there is a problem – you only have room for one button!

In this tutorial, we are going to use Arduino to explore how to make one button have the functionality of two or more.

You Will Need:

(1) Momentary push button
(5) Jumper wires
(1) Solderless breadboard
(2) LEDs
(2) 220 Ohm resistors

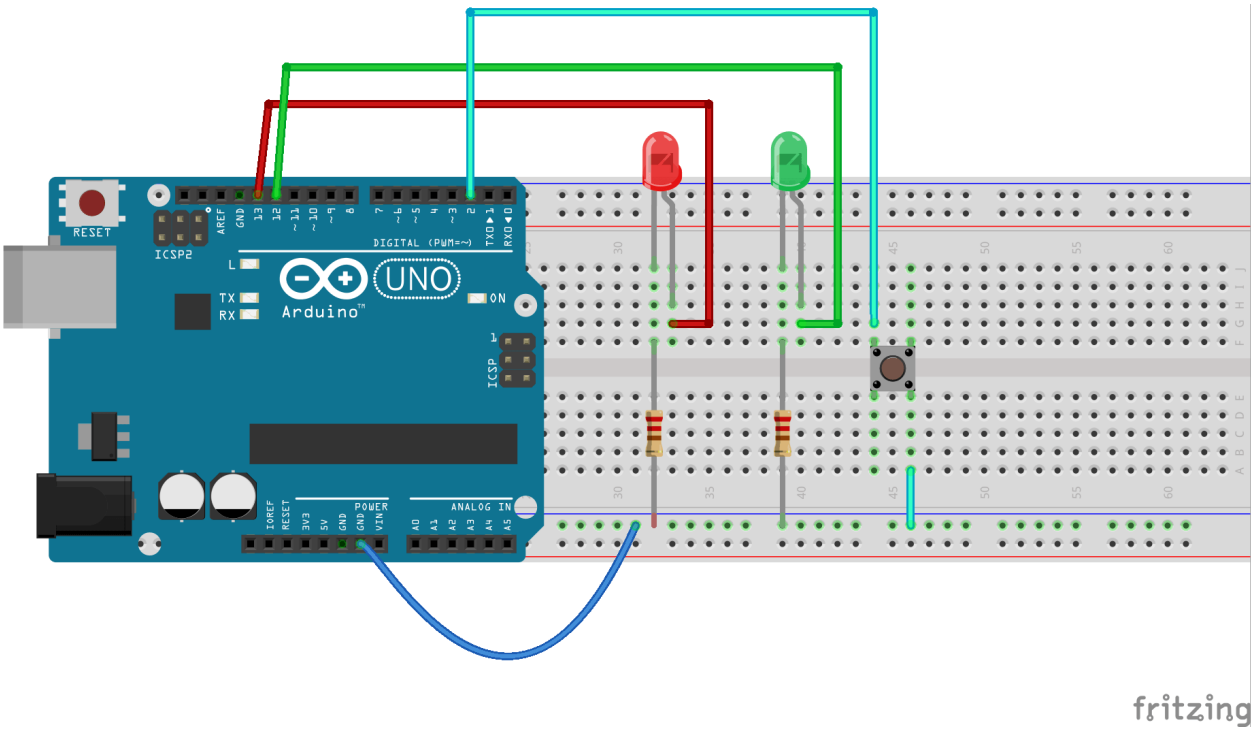
Set Up The Circuit:

To demonstrate using a single button for multiple functions, we will set up a simple circuit with 2 LEDs and a button. Based on how we press the button, different LEDs will illuminate.

Follow the instruction and schematic below to get the circuit setup before we dive into the mechanics of the Arduino code.

1. Using a jumper wire, connect any GND pin from the Arduino, to the ground rail on your breadboard.
2. Place an LED on your breadboard, make sure to note which way the long leg is facing.
3. Using a jumper wire, connect pin 13 from your Arduino to the breadboard in the same channel where you have the long leg of the LED attached.
4. Now connect one side of the 220 Ohm resistor to the short leg of the LED, and the other leg connect to the ground rail on the breadboard. The orientation of the resistor doesn't matter.
5. Now repeat this using pin 12, and another LED and resistor.
6. Finally, place your push button on the breadboard. Depending on the style of your pushbutton, they often fit well straddling the long trench that goes through the breadboard.
7. Connect a jumper wire form one side of the button to pin 2 on the Arduino.
8. Connect a jumper wire from the other side of the button to the ground rail on the breadboard.

That is it for the circuit setup. **Now, when you press the push button (which will electrically connect both sides of the button), pin 2 to will have ground voltage applied.** We will use this ground voltage input to trigger our different functions.



Examine the Sketch:

There are couple ways to implement the multi-function button press using Arduino. One way is to have the number of presses determine the output. For example, a single click might highlight the "hour" field of an LCD timer and a double click might highlight the "minute" field of the display.

Another way we can implement multiple functions with one button is for the user to hold down the button for different lengths of time, the length of the hold determining the output.

For example, if the user holds the button for half a second and releases, something happens. If she holds it for 2 seconds, something different happens.

This latter method of using button hold length time to determine separate functions is the strategy we will learn here.

Before I go any further though, I would like to thank Steve for creating the base Arduino code that we will be using. Steve is a member of the Premium Arduino course (a couple months ago he was new to Arduino).

While creating a home automation project he was in need of using a single button to do multiple things, and came up with a very simple way to make it happen. Thanks Steve!

Here is the complete sketch, I recommend looking it over first, and then we will discuss it piece by piece below.

```
/*Using a Single Button, create multiple options based on how long the button is pressed

 The circuit:
* LED attached from pin 13 to ground through a 220 ohm resistor
* LED attached from pin 12 to ground through a 220 ohm resistor
* one side of momentary pushbutton attached to pin 2
* other side of momentary pushbutton attached to Ground

* Note 1: on most Arduinos there is already an LED on the board
attached to pin 13.
* Note 2: In this circuit, when the button is pressed, Ground Volatage is what will be

 Created DEC 2014 by Scuba Steve
 Modified JAN 2015 by Michael James
 Both members of https://OpenSourceHardwareGroup.com

 This code is in the public domain
 */


/////////Declare and Initialize Variables//////////////////////////

//We need to track how long the momentary pushbutton is held for in order to exectute
different commands
//This value will be recorded in seconds
float pressLength_milliSeconds = 0;

// Define the *minimum* length of time, in milli-seconds, that the button must be
pressed for a particular option to occur
int optionOne_milliSeconds = 100;
int optionTwo_milliSeconds = 2000;

//The Pin your button is attached to
int buttonPin = 2;

//Pin your LEDs are attached to
int ledPin_Option_1 = 13;
int ledPin_Option_2 = 12;
```

```
void setup(){

  // Initialize the pushbutton pin as an input pullup
  // Keep in mind, when pin 2 has ground voltage applied, we know the button is being
pressed
  pinMode(buttonPin, INPUT_PULLUP);

  //set the LEDs pins as outputs
  pinMode(ledPin_Option_1, OUTPUT);
  pinMode(ledPin_Option_2, OUTPUT);

  //Start serial communication - for debugging purposes only
  Serial.begin(9600);

} // close setup


void loop() {

  //Record *roughly* the of tenths of seconds the button in being held down
  while (digitalRead(buttonPin) == LOW ){

    delay(100);  //if you want more resolution, lower this number
    pressLength_milliSeconds = pressLength_milliSeconds + 100;

    //display how long button is has been held
    Serial.print("ms = ");
    Serial.println(pressLength_milliSeconds);

  }//close while


  //Different if-else conditions are triggered based on the length of the button press
  //Start with the longest time option first

  //Option 2 - Execute the second option if the button is held for the correct amount of
time
  if (pressLength_milliSeconds >= optionTwo_milliSeconds){

    digitalWrite(ledPin_Option_2, HIGH);

  }
```

```
//option 1 - Execute the first option if the button is held for the correct amount of time
  else if(pressLength_milliSeconds >= optionOne_milliSeconds){

    digitalWrite(ledPin_Option_1, HIGH);

  }//close if options


  //every time through the loop, we need to reset the pressLength_Seconds counter
  pressLength_milliSeconds = 0;

} // close void loop
```

Comments:

At the top of the sketch we find the comments. You should make it a habit to read the comments in a sketch before jumping into the mechanics of the code. The comments should lay the groundwork for what is going to happen in the program and will help you interpret the intent of the code as you begin to analyze it.

Declare and Initialize Variables:

After the comments we start initializing and declaring variables. Since we are going to be tracking time, we need to have a variable to record the length of time a button is being help. We do that with the *pressLength_milliSeconds* variable:

```
//We need to track how long the momentary pushbutton is held for in order to exectute
different commands
//This value will be recorded in seconds
float pressLength_Seconds = 0;
```

Now you might think that the variable name is really long and annoying. And I wouldn't particularly argue with you – I mean, why would I include milliSeconds in the name of the variable!

The reason I do this, is because I think including the unit of measurement in the variable name is helpful when other people are trying to read your code. Writing code that other people can read is not only good for other people, but also future versions of yourself who forget what the heck you were thinking when you wrote the code! [End Rant]

The next thing we need to set up are the parameters for when options will get executed. In this example I have two variables for two options:

```
// Define the *minimum* length of time, in milli-seconds, that the button must be
pressed for a particular option to occur
int optionOne_milliSeconds = 100;
int optionTwo_milliSeconds = 2000;
```

Each option is defined by the number of milliseconds that the button must be held for that specific option to get executed. In order to get my first option to happen, I have to hold the button for at least 100 milliseconds, that is pretty much a short tap on the button.

If I want the second option to happen, then I have to hold the button for at least 2000 milliseconds aka 2 seconds.

If you wanted more options, you would add more variables here with their corresponding hold times.

Our final initializations will be to specify pin numbers for our button and LEDs.

```
//The Pin your button is attached to
int buttonPin = 2;

//Pin your LEDs are attached to
int ledPin_Option_1 = 13;
int ledPin_Option_2 = 12;
```

Setup() the Sketch

The setup() for this sketch is pretty straight forward (if it's not straight forward to you, make sure to check out our free 12-part Arduino Course, after which this setup will be very familiar to you).
We want to make sure that the pin our push button is connected to is set as an INPUT_PULLUP:

```
// Initialize the pushbutton pin as an input pullup
// Keep in mind, when pin 2 has ground voltage applied, we know the button is being
pressed
  pinMode(buttonPin, INPUT_PULLUP);
```

We do this to make sure that the button pin is not floating (if you are wondering what the heck that means, you can read more on that here – but if you just roll with me until we get through this tutorial and you should be fine).

We also want to specify the pins that our LEDs are attached to as OUTPUTs, because we will be applying voltages to these pins in order to illuminate them:

```
//set the LEDs pins as outputs
pinMode(ledPin_Option_1, OUTPUT);
pinMode(ledPin_Option_2, OUTPUT);
```

Finally, it never hurts to start serial communications for debugging purposes.

```
//Start serial communication - for debugging purposes only
Serial.begin(9600);
```

With setup() complete, now we can jump into the main loop of our sketch…

The Main Loop()

We know we are going to have to measure the length of time the button is pressed and then record it.

To do this, we use a while statement whose condition requires the button pin to be in a LOW state (remember, when we push the button, pin 2 will have a ground voltage applied).

```
//Record *roughly* the of tenths of seconds the button in being held down
 while (digitalRead(buttonPin) == LOW ){
```

Once the button is pressed and held then the while statement starts executing. The first thing we do in the while statement is to delay 100 milliseconds, and then record that into our time tracking variable:

```
  delay(100);  //if you want more resolution, lower this number
  pressLength_milliSeconds = pressLength_milliSeconds + 100;
```

Keep in mind the first time through the loop, pressLength_milliSeconds will be equal to 0, so we are just adding 100 to the variable.

It can be handy to know how long the button has been pressed as you add options, to make this easy we want to print the current value of the pressLength_milliSeconds variable to the serial monitor window:

```
//display how long button is has been held
  Serial.print("ms = ");
  Serial.println(pressLength_milliSeconds);
```

Let's ignore the rest of the code for a second, and imagine what happens if we keep holding the button.

The first time through the while loop, we add 100 milliseconds to the time tracking variable and we print that value to the serial port. The next time through loop, we add another 100 milliseconds to the timer counter variable, and print this new value to the serial monitor.

As long as the button is being held down, then we keep adding time to the pressLength_milliSeconds variable – this is the crux of the program.

When we release the button, then the while statement stops, because the condition is no longer met, and we stop adding time to pressLength_milliSeconds.

So let's pretend we held the button for three seconds and then let go - what happens?

Well, as we discussed the while statement ends, and the next line of code we encounter is an *if statement*.

```
  //Option 2 - Execute the second option if the button is held for the correct amount of
time
  if (pressLength_milliSeconds >= optionTwo_milliSeconds){

    digitalWrite(ledPin_Option_2, HIGH);

  }
```

The condition of the if statement requires that the time we held the button be longer then or equal to the time we set for option number two.

If you recall, option number two was set to occur with at least 2 seconds of button press time – since we have held the button for three seconds, then this if statement will get executed.

And all we do is write HIGH voltage to our "option 2" LED, making it illuminate.

What if we had only held the button for one second – then what would happen?

If this were case, then the first if statement condition would not have been met, but a subsequent else-if statement only requires the button hold time be 100 milliseconds or more – so the second else-if statement would get executed, which turns on the "option 1" LED.

```
//option 1 - Execute the first option if the button is held for the correct amount of time
else if(pressLength_milliSeconds >= optionOne_milliSeconds){

  digitalWrite(ledPin_Option_1, HIGH);

}//close if options
```

Basically, if we hold the button a long time, the second option gets executed. If we hold the button a short time, the first option gets executed.

If we wanted to add more options, we add the longer hold options at the top, and the shorter hold options at the bottom.

I wouldn't try to squeeze to many options in a small span of time or might drive the end user crazy trying figure out the timing.

Nor would I try to add more than three options for a single button within a given context, or else you chance making your potential end user want to beat you up.

To finish up the sketch, we reset our button press timing variable to zero. This ensures the next time the button is pressed and held that we start from time zero again.

Try On You Own Challenge:

1. Add another option, which turns off both LEDs. Try adding it before the first option (you will have to adjust the timing) and then after each option.

2. How tight can you squeeze the option time together? Experiment and determine what a good rule of thumb is.