Momentary buttons create a temporary connection between a set of contacts when the button is pressed. It takes a finite force to overcome a spring inside the button so that electrical contact is made. We hear an audible “click” as the mechanism inside button makes the contact and then springs back to its neutral position.

Although it is undetectable to our finger when the button is pressed, the internal mechanism bounces as the energy stored in the spring is dissipated. Thus, instead of a single and temporary electrical contact, the button may have several secondary contacts after the initial click. When a button is tied to a digital input circuit, the unwanted secondary bounces have to be either electrically damped, or ignored in the software. These notes demonstrate how a software “de-bounce” can be implemented. In this example the button input is handled by polling during the `loop` function. A more robust solution using interrupts is discussed in other course notes.

Figure 1 shows a circuit used to demonstrate software debouncing. The button is configured with a pull-down resistor and is tied to digital pin 4. When the button is not pressed, the blue wire is connected to ground through a 10k resistor, which causes the digital input to be LOW. When the button is pressed, the digital input is connected directly to the 5V rail, which causes the digital input to be HIGH. The red LED is connected directly to the button, and as such it indicates whether the button is pressed or not. The blue LED in Figure 1 indicates the effect of button presses on the Arduino program. Pressing the button causes the blue LED to toggle between on and off states.

An Arduino sketch that uses the circuit in Figure 1 is included at the end of this document in Listing 1a and Listing 1b. The sketch does not fit onto one page. Listing 1a contains the `setup` and `loop` functions. Listing 1b contains a user-defined function called `debounce_button` that handles the button input with debouncing. The code contains extensive comment statements. Only a few features of the code will be discussed in this narrative text.

Two variables, `last_click` and `LED_state` are declared with the `static` modifier. A `static` variable retains its value between function calls. Therefore, the value of l`ast_click` is retained between subsequent calls to `debounce_button`. Similarly, the value of `LED_state` is retained between subsequent calls to the `loop` function.

The debouncing operation is implemented by keeping track of the time since the last legitimate button click. By legitimate, we the electrical contact initiated by pressing the button, not the secondary contacts made when the internal button mechanism bounces. Each time the `debounce_button` function is called, the current system clock is stored in `time_now`. The difference between `time_now` and `last_click` is compared to `ignore_bounce`, which is set to a constant value of 200. The units of `time_now`, `last_click` and `ignore bounce` are milliseconds, so the program ignores any clicks that are closer in time that `ignore_bounce`, or 200 milliseconds.

The `debounce_button` function provides a convenient way to process a single momentary button. Unfortunately, the same code cannot be used for more than one button in a single sketch. You could create near-identical copies of the `debounce_button` function for each button in your project. Each button would require its own value of `last_click`.

An alternative to cloning the debounce_button function when multiple buttons are present, users should consider one of the available button debouncing libraries with an object-oriented interface. The libraries listed in Table 1 support multiple buttons. However, none of those libraries appear to support hardware interrupts.

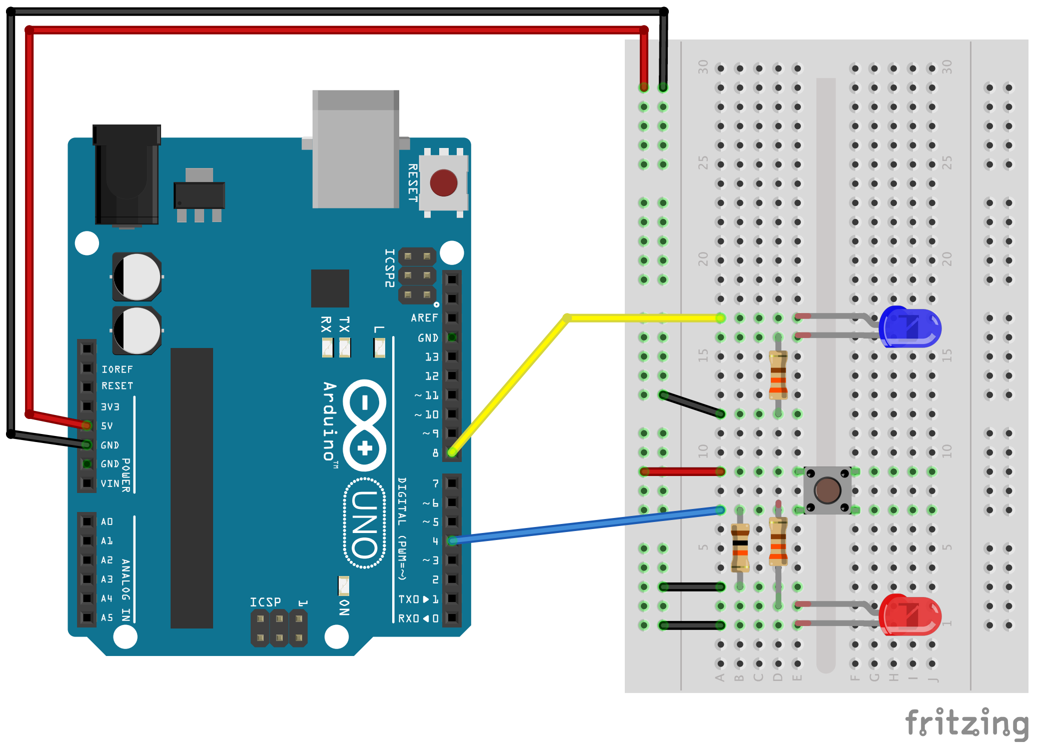

Figure 1 Arduino circuit for button input. The button is configured with a pull-down resistor. Two LEDs are used as indicators.

Table 1 Object-oriented libraries for button input without hardware interrupts.

Name	Author/Source, URL and Description
Bounce2	Thomas Friedricks https://github.com/thomasfredericks/Bounce2 Accessed 14 November 2018 Allows specification of button press duration and use of Arduino pins with internal pull-up resistors. Allows specification of time interval for ignoring button bounces.
EasyButton	Evert Arias https://evert-arias.github.io/EasyButton/ Accessed 14 November 2018 Allows specification of button press duration and use of Arduino pins with internal pull-up resistors. Allows specification of time interval for ignoring button bounces. Allows specification of callback functions to be associated with pre-defined button behaviors.
JC_Button	Jack Christensen https://github.com/JChristensen/JC_Button accessed 14 November 2018 Allows specification of button press duration and use of Arduino pins with internal pull-up resistors. Allows specification of time interval for ignoring button bounces.

```
//   File:  button_debounce_function_loop.ino
//
//   Demonstrate how to implement software debouncing.  In this example,
//   the button input is determined manually in a function that is called
//   from the loop function, i.e., this example does not use an interrupt.
//
//   The button circuit uses a momentary button with a pull-down resistor.
//   Clicking the button changes the output from LOW to HIGH

// -- Identify digital I/O pins for button input and LED output.
//    The LED is toggled on and off with each (non-bounced) button click
int  button_pin = 4;
int  LED_pin = 8;

// ---------------------------------------------------------------------------
// -- Nothing fancy in setup
void setup() {
  pinMode(button_pin, INPUT);   // Connect a momentary button
  pinMode(LED_pin, OUTPUT);
  Serial.begin(9600);
}

// ---------------------------------------------------------------------------
// -- In this example, loop uses a utility to determine button state, toggles
//    the LED state based on button input, and prints diagnostics
void loop() {

  static int LED_state = false;  // status of the LED:  either on or off (could be boolean)
  int  button_click;             // True button click, no bounce effects (could be boolean)

  button_click = debounce_button(button_pin);   // Get the debounced button state.

  // -- If a button_click occured, toggle the state of the LED:  on becomes off; off become on
  if ( button_click ) {
    LED_state = !LED_state;
  }

  // -- Regardless of whether a button event occured, use LED_state to turn the LED on or off
  //    Note that LED_state is a static variable, which means that it retains its value after
  //    the loop function is terminated
  if ( LED_state ) {
    digitalWrite(LED_pin, HIGH);
  } else {
    digitalWrite(LED_pin, LOW);
  }

  // -- Diagnostic printing.
  Serial.print(button_click);   Serial.print("  ");  Serial.println(LED_state);
}
```

Listing 1a The `setup` and `loop` functions for an Arduino code to demonstrate software debouncing.

```
// ------------------------------------------------------------------------------------
// -- debounce_button function reads a digital input pin and returns TRUE only
//    if the input is separated in time from the last time that pin was TRUE
//    NOTE: This will only work for one button.
int debounce_button(int button_pin) {

  static long last_click=0; // time when the last legit button click occurred
  long time_now;            // current clock; is compared to last_click
  int  test_click;          // Result of digitalRead; Can't distinguish bounces from legit clicks
  int  button_click;        // True button click, with bounces ignored
  int  ignore_bounce=200;   // Time interval (milliseconds) use as threshold for bouncing
                            // Ignore any click (or bounce) that happened less than ignore_bounce
                                 // milliseconds from the last click

  // -- Begin button reading with debouncing
  time_now = millis();                      //  Read the clock
  test_click = digitalRead(button_pin);     //  Read the button pin

  // -- If test_click is true (HIGH), then something happened on the digital input channel.
  //    But, test_click cannot distinguish a legit click from a bounce.  To separate a bounce
  //    from a true click, check that enough time has passed since the last click event.
  //    The (time_now-last_click > ignore_bounce) expression is true if this click event
  //    happened more than ignore_bounce milliseconds from the last click event.
  //
  //    In other words, test_click is true if a true click occurred OR if the button bounced
  //    button_click is true ONLY IF the click event was caused by a button click, not a bounce.
  //
  if ( test_click & (time_now-last_click > ignore_bounce) ) {
    button_click = true;     //  A true click: it's been long enough since the last click event
    last_click = time_now;   //  IMPORTANT:  update last_click to be the time of last true click
  } else {
    button_click = false;    //  No true click:  button_click is false if either test_click is
  }                          //  false (no button event) or if not enough time has passed since
                             //  the last button event
  // -- End button reading with debouncing

  return(button_click);
}
```

Listing 1b The `debounce_button` function implements a software debouncing of a momentary input button. Note that the button is assumed to be configured with a pull-down resistor.