ARDUINO - A Tutorial in Connecting and Configuring a Button Switch

(RDB, Stafford (UK),
January 2021)

Contents

Copyright and Warranties

Audience

This tutorial has been written to provide a level of basic understanding relating to the connection and reliable use of simple button switches with Arduino microcontrollers. Understanding these fundamentals will ensure that beginners, hobbyists and other users will be able to choose appropriate switch circuit designs and reliable service associated button switches.

The tutorial is aimed at anyone, but particularly beginners and those new to Arduino, needing to 'get under the hood' so to speak. Having said that, beginner or expert user alike, I am sure there will be something for everyone.

To get the most out of this tutorial the reader should have some basic understanding of circuit wiring (i.e. using breadboards or otherwise), the Arduino IDE development environment, C/C++ programming language and a few common digital I/O functions native to the Arduino, but specifically, `pinMode()`, `digitalRead()` and `digitalWrite()` (see Resource links below).

Resources

There are many resources available on the internet, some good, some less so. The links provided below should prove helpful for background and further reading in relation to this tutorial.

Arduino IDE software download	see [Arduino - Software]().
Description of digital pins (I/O), including `pinMode`, `digitalRead` and `digitalWrite`	see [Arduino - DigitalPins]()
Microcontroller built in LED constant ('`LED_BUILTIN`')	see [constants – Arduino Reference]() – Defining built-ins: LED_BUILTIN.)

Introduction

In this tutorial we look at simple switches and how these can be connected to Arduino microcontrollers such that they can be used reliably.

The design aims for the tutorial are to:

- be accessible to readers with a basic level of C/C++ programming and awareness of the Arduino architecture – for example, basic digital I/O functions, specifically, `pinMode`(), `digitalRead`(), `digitalWrite`(), Arduino sketch structure, use of the Arduino IDE, etc
- provide a practical appreciation of key considerations in switch design and implementation, and to underpin these with a real world example by way of a C++ Arduino sketch that will, hopefully, demonstrate the principles laid out.

However, before we crash headlong into switches, wires, resistors, microcontrollers and coding, we will need to consider, appreciate and understand three things:

1. The type of switch to be used
2. How we wish the switch to be wired and configured (there are choices)
3. The fundamental issues inherent with imperfect switch design and manufacture.

Three Considerations

Let's look at each of these considerations in turn:

Consideration 1 - Type of switch

We will look at connecting and configuring single pole push/press switches, or button switches. This is the one of the simplest form of switch – single pole just means it has one input line and one output line which only connect when the switch is pressed/pushed. That is, when activated, the gap between the input and output is closed and so current will flow. However, there are many types of switch and each will have its own characteristics requiring specific software considerations.

Consideration 2 - Wiring Choices

A simple button switch has two terminals each of which will need to be connected to an Arduino pin if we are to detect its operation. There are several ways to connect a simple button switch, but for the purposes of this tutorial we shall look at the two most common. Let's call these two circuits C1 and C2 and consider each in turn.

Figure 1 shows an example of circuit C1 - one terminal connected to a digital I/O pin (the digital read pin), for reading switch change states, and the other terminal to +5v AND 0v (ground) pins on the Arduino, via a 10k ohm resistor.

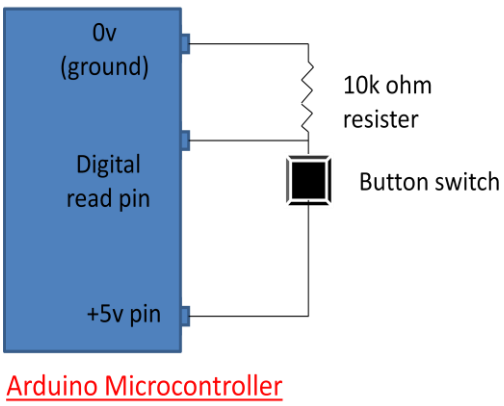


Figure 1 – Circuit C1

This is the traditional design for wiring a simple switch. The design is such that the digital read pin will be initialised as a simple input pin and will be LOW (0v) when the switch is off and HIGH (5v) when pressed (on). The reason for the 10k ohm resistor is that Arduino digital pins are very sensitive to electromagnetic fields that can generate spurious inputs. The addition of a 10k ohm resistor ensures the pin is maintained at 0v unless raised to +5v via operation of the button switch.

However, there is a second and direct way in which a simple switch can be connected to the microcontroller, one that does not suffer from detectable electromagnetic interference.

Figure 2 shows an example of circuit C2 - one terminal connected to a digital pin (the digital read pin), for reading switch change states, and the other terminal to just 0v (ground) pin on the Arduino. Note the absence of a resistor.

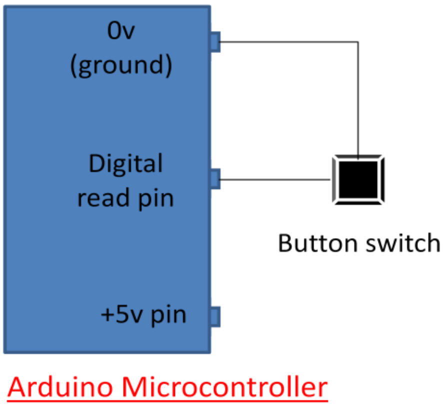


Figure 2 – Circuit C2

This approach is, perhaps, less common but does work fine. To note is that switch polarity and sensing is reversed - the design is such that the digital read pin will be initialised to be HIGH (+5v) when the switch is off and LOW (0v) when pressed (on).

In either case the assigned digital read pin to the circuit must be configured, usually in setup(), with a `pinMode()` call. Thereafter, testing for switch operation (using `digitalRead()`) will need to take note of the circuit design choice (C1 or C2) as the voltage levels for on and off are a reversal of each other. The following table shows the software parameters for each circuit choice:

Circuit	Switch initialisation in Setup()	Return value from `digitalRead(pin_no)`	
		Switch PRESSED (ON)	Switch RELEASED (OFF)
C1	`pinMode(pin_no,INPUT)`	`HIGH` (+5v)	`LOW` (0v)
C2	`pinMode(pin_no,INPUT_PULLUP)`	`LOW` (0v)	`HIGH` (+5v)

Table 1 – set up and digital reading

Understanding the circuit design and set up is critical to be able to reliably configure the sensing software solution (more of the next).

Are we there yet? Nearly there, but not quite. Let's move on to the next consideration – switch subtleties.

Consideration 3 - Switch Subtleties/ Design/Manufacture Imperfections
So far, so good - we have two circuit designs, know how each design is to be initialised and what values we will get when we read a switch (Table 1), pressed and not pressed. We can just use the `digitalRead()` function and crack on? Can't we? What more?

Well, the problem with switches, and other mechanical components, is that they do not always operate cleanly – they often generate 'noise' during the switching operation.

Switches, invariable, need a little time to 'settle down when they change their state. During this settling time, if we read the value of the switch we will likely get unreliable (spurious) results arising from the noise generated associated with the physical characteristics and mechanics of operating the switch.

If we used an oscilloscope to look at a typical button switch cycle, going from off to on and back to off, we would see something like the trace shown in figure 3.

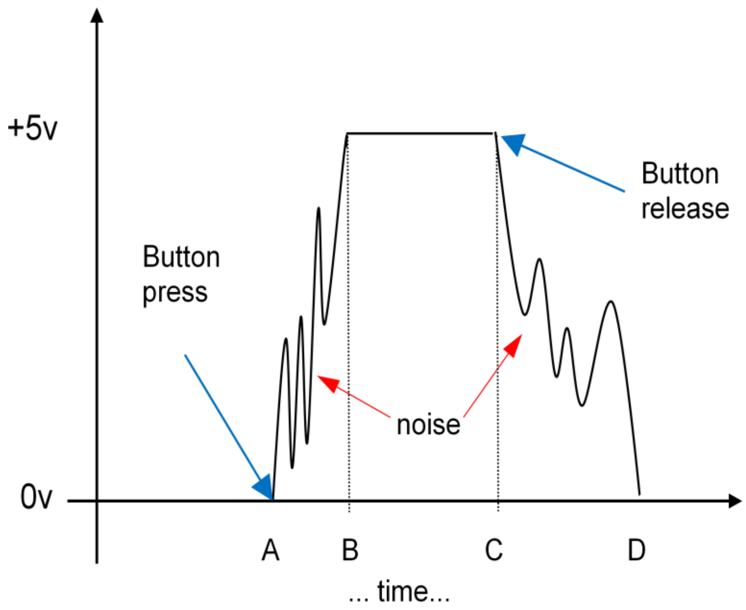


Figure 3 - Button Switch OFF/ON/OFF Cycle

The key points to note about this trace are;

1. The figure shows a button switch configured as for circuit C1 (off = `LOW`, on = `HIGH`).
2. The button switch is pressed at time A and released at time C.
3. The total time the button switch is physically pressed is T = C - A seconds.
4. Time T can be as short or as long at the button switch is held down.
5. The state change does not instantaneously go from `LOW` to `HIGH` (pressed) or from `HIGH` to `LOW` (released) - it takes a finite time for transition to occur, as shown by the time intervals B - A (rising) and D - C (falling).
6. If we read the button switch during these transition times we will likely get unreliable results, sometimes reading `LOW`, sometimes reading `HIGH`.
7. It is not until we read the button switch within the interval C - B that we will get a reliable and consistent reading.

So how do we program our way around these imperfections of the physical world? Well, one common method is to introduce the idea of 'debounce'.

Debounce is a method that accepts the imperfections resulting from noise during state change transitions of switching. The method introduces a timing parameter short enough to allow a switch to achieve its settled state, either fully and cleanly on, or off. Unless you have an oscilloscope, there is no science to how long this period should be, but it ideally needs to as short as possible. It is normally set by trial and error or experience. The better the quality of a switch (we would hope) then we would expect the debounce (state transition) periods to be low, perhaps 10 msecs or less. However, debounce periods of 100 msecs or higher are not uncommon in many examples you may come across.

There are numerous examples of how to program switches with debounce on the internet so why have I bothered with the effort to put together this tutorial? Well, many of these examples are not particularly comprehensive. Many simply jump straight into providing some code and little more. My objective has therefore been to provide a fuller appreciation and understanding of the challenges in implementing what is, after all, a fundamental and basic interface – a button switch.

On inspection of the included example sketch, S1, it will be seen that there are two functions designed to read a simple button switch. Whilst both will read the same switch and incorporate debounce logic, they operate is completely different ways. What I have tried to present is that there is always more than one way to 'peel a banana' – the end result is the same (we get to eat the banana) but how we 'get in' may differ. See next section for an appreciation of the differences.

Putting It All Together – Example Sketch

What follows is an example of a sketch, S1, that will reliably read the state of a button switch and which is highly configurable to allow the reader to explore the ideas, concepts, constraints and limitations of using a simple button switch.

The sketch design is such that;

- It includes extensive comments throughout which will hopefully aid the reader in understanding what's going on.
- It supports either choice of switch circuit design (C1 or C2), with minimal change. By default, the sketch is coded to work with circuit C1 ('`circuit_C1`'), but if you wish to use circuit C2 make changes as required, see below.
- The choice of digital input pin used to detect switch state changes is set, by default, to pin 2, but may be changed to any suitable pin as required, see below.
- The microcontroller onboard LED is used for testing to indicate switch state changes. It is defined using a macro '`#define LED LED_BUILTIN`' so should work okay without installing an external LED/resistor circuit. '`LED_BUILTIN`' is a reserved constant and set by the Arduino IDE depending on the microcontroller board referenced in the IDE configuration data. (For information on '`LED_BUILTIN`' see constants – Arduino Reference – Defining built-ins: LED_BUILTIN.)
- It includes two methods for reading the button switch, each coded as a Boolean function:

  Function 1 - `bool read_switch_method_1()`
  This version of switch reading examines the switch once each time the function is called. It allows the code section from which it is called to continue without waiting for the switch press cycle to complete once switching is initiated – it is what is referred as a non-blocking function. The only drawback, of course, is that the design of the calling code must ensure that the switch is regularly tested to catch a change in switch status. However, this should not normally be an issue or concern.

  Function 2 - `bool read_switch_method_2()`
  This version of switch reading will wait for a switch cycle to complete once it is initiated, before control is returned to the calling code – it is what is referred as a blocking function. That is, the calling code will be held up once the switch is pressed, released

and until the debounce period has elapsed. Once the switch is pressed the function keep total processing control for a time equivalent to at least the time the switch is fully pressed plus the debounce period. This version is therefore less efficient than `read_switch_method_1()`.

- The control code within the main sketch loop may incorporates either `read_switch_method_1()` or `read_switch_method_2()`. By default, `read_switch_method_1()` is coded in the main loop call, it being more efficient. . This can be changed if desired, see below. Note that either read function will work with either circuit design.

In summary, out of the box (OOTB) the sketch (S1) is configured as follows:

Configurable Parameters	Sketch Variable/Definitions	Sketch Default Values
Switch circuit design	`#define circuit_type`	`circuit_C1`
Digital input pin for switch reading	`int button_switch =`	`2`
LED output pin	`#define LED`	`LED_BUILTIN`
Debounce period (msecs)	`#define debounce`	`50`
Main loop switch read function call		`read_switch_method_1()`

Table 2 – Out Of The Box Configuration

Before the following exercises are attempted, the reader should spend a little time familiarising themselves with sketch S1. Whilst it includes lots of comment, which it is hoped the reader will find helpful, it will be seen that the sketch coding is not too lengthy and constructed so as to be as self documenting as possible. The sketch is laid out as follows:

- Introductory comments
- Declarations, variables and definitions
- Setup() function
- Digital switch read functions (x 2)
- Main loop()

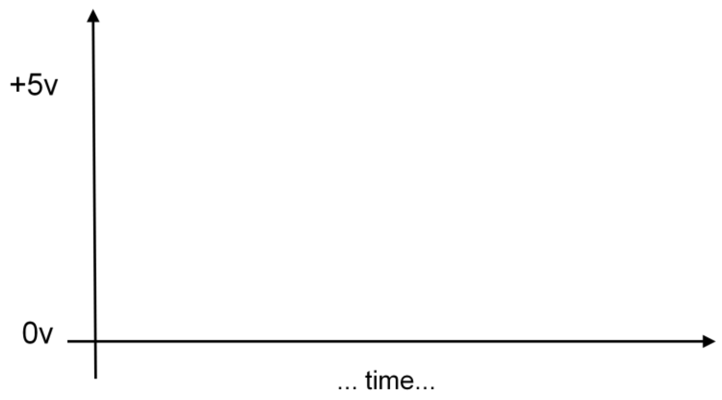
Exercises

The following exercises are suggested to help consolidate and explore understanding.

Exercise 1 - Switch Trace

Figure 3 shows an example of an oscilloscope trace for a button switch connected as for circuit C1 (`pinMode(., INPUT)`).

Draw a similar trace for a button switch designed with circuit C2 (`pinMode(., INPUT_PULLUP)`).



Exercise 2 - OOTB

Load the OOTB sketch and connect a button switch in the configuration of circuit C1 to your microcontroller. That is, with a pull down resistor, see Figure 1.

The sketch macro definition and variable values should be as in Table 2, above, but if not make the changes to reflect those in Table 2.

Compile and up load the sketch and operate the button switch in your circuit. In particular, note that the LED toggles on/off with each switch press and that the switch state change only occurs when it is released.

Exercise 3 – Alternative Switch Circuit, C2

Reconfigure the button switch design to now follow circuit C2, see Figure 2. Edit the macro definition `#define circuit_type circuit_C1` to now be `#define circuit_type circuit_C2`.

The sketch definitions and variable value should be as below:

Configurable Parameters	Sketch Variable/Definitions	Sketch Default Values
Switch circuit design	`#define circuit_type`	`circuit_C2`
Digital input pin for switch reading	`int button_switch =`	`2`
LED output pin	`#define LED`	`LED_BUILTIN`
Debounce period (msecs)	`#define debounce`	`50`
Main loop switch read function call		`read_switch_method_1()`

Compile and up load the sketch. What do you notice? The sketch should operate exactly the same as for the OOTB settings with `circuit_C1`.

Exercise 4 – Alternate Switch Read Method

Now use the alternate switch read function by editing `read_switch_method_1()` in the main code loop to be `read_switch_method_2()`.

The sketch definitions and variable value should be as below:

Configurable Parameters	Sketch Variable/Definitions	Sketch Default Values
Switch circuit design	`#define circuit_type`	`circuit_C2`
Digital input pin for switch reading	`int button_switch =`	`2`
LED output pin	`#define LED`	`LED_BUILTIN`

Configurable Parameters	Sketch Variable/Definitions	Sketch Default Values
Debounce period (msecs)	`#define debounce`	`50`
Main loop switch read function call		`read_switch_method_2()`

What do you notice? The sketch operates exactly the same as for the exercises.

Exercise 5 – Playing Around With Debounce Values

Using the switch circuit and sketch as for exercise 4, vary the debounce value (`#define debounce`) and see how decreasing/increasing this affects the switch's operation. Note the results for your future use.

The sketch definitions and variable value should be as below:

Configurable Parameters	Sketch Variable/Definitions	Sketch Default Values
Switch circuit design	`#define circuit_type`	`circuit_C2`
Digital input pin for switch reading	`int button_switch =`	`2`
LED output pin	`#define LED`	`LED_BUILTIN`
Debounce period (msecs)	`#define debounce`	Suggested values to try: `0, 25, 100, 150, 200, 250`
Main loop switch read function call		`read_switch_method_2()`

And Finally

I hope that, through this tutorial, the reader has reached a fuller appreciation and understanding of connecting and using simple button switches with Arduino microcontrollers.

The simple button switch is commonly used but there are many choices and examples of switch. Whilst the majority are of a digital nature (i.e. either off or on), we should not dismiss analogue components that can also be used as inputs for switching, e.g. potentiometers, photoresistors, etc. Each will invariably have its own peculiarities which will need to be understood and suitably programmed for. The good news is that the coding does not get necessarily more complex.

Enjoy!

Example OOTB Sketch S1

```
//  This example and code is in the public domain and may be used without restriction and
//  without warranty.

/*
  READING SIMPLE SWITCHES RELIABLY, WITH OR WITHOUT A SWITCH PULL DOWN RESISTOR
  '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
  In this example sketch we look at configuring a simple button switch such that when pressed
  it will toggle a LED on and off.  The sketch automatically allows for one of two switch
  circuits to be configured, either with a pull down switch resistor or without.  This is configured
  and controlled with a simple macro parameter - see points of note below.

  Additionally, this sketch offers two methods to read a simple button switch, each offered as a
  specific function.  Only one function method should be configured (..._method_1/..._method_2),
  the choice left to preference of the user. The differences between each method is highlighted and
  explained below:
    Method 1 -  Function 'bool read_switch_method_1()' - this version of switch reading examines the switch
                input once each time the function is called.
                This allows the code section from which it is called to continue without waiting
                for the switch press cycle to complete once switching is initiated.
                Only drawback, of course, is that the design of the calling code must ensure that the
                switch is regularly tested to catch a change in switch status.

    Method 2 -  Function 'bool read_switch_method_2()' - this version of switch reading will wait for a
                switch cycle to complete, once it is initiated, before control is returned to the calling code.
                Once the switch is pressed, the code will fully consume the debounce period AFTER switch
                release until control is returned back to the calling code.
                That is, the calling code will be held up once the switch is pressed and until
                the debounce period has elapsed.

  Points of note:
    1.  The digital pin chosen as the input is initialised according to the 'circuit_type' macro
        parameter, either INPUT_PULLUP or INPUT using a call to pinMode(button_switch,circuit_type)
        in the setup() function.
        As the conditions for detecting switch on/off are different for each 'circuit_type' (they are
        reversed) two variables are used ('switch_low' and 'switch_high') to provide a reference
```

```
        indicating low and high (on/off) conditions.  However, this is transparent to the functioning
        of the code.  The only requirement is to define the 'circuit_type', everything else is
        automatically taken care of.

    2.  The design for each reading method incorporates code to debounce spurious inputs when the switch
        is pressed which, if not accounted for, would produce unexpected/spurious results.

    3.  The wiring designs for each type of switch circuit are:

        With switch pull down resistor configured and 'circuit_type' of INPUT

              10K ohm     button      (onboard)
              resistor    switch         LED
          ____|<><><>|_______ \___        O
         |              |         |       |
         0v          pin 2     +5v     pin =
                                    LED_BUILTIN
         ___________________________________
         |            ARDUINO              |

        With NO switch pull down resistor configured and 'circuit_type' of INPUT_PULLUP

             button     (onboard)
             switch        LED
            ___ \___        O
           |        |       |
         pin 2     0v     pin =
                       LED_BUILTIN
         _______________________
         |       ARDUINO       |

    5.  The on board LED (MEGA 2560) is utilised for testing to keep circuit design to a minimum.
        If not using the onboard LED then configure one in the traditional way using a 220ohm resistor (for
        a red LED) and a red LED on a breadboard, or otherwise.
*/
```

```
// define type of switch circuit and associated LOW and HIGH variables:
//    - circuit C1, INPUT with switch resistor           -> switch_high = HIGH, switch_low = LOW
//    - circuit C2, INPUT_PULLUP with no switch resistor -> switch_high = LOW,  switch_low = HIGH

#define circuit_C1    INPUT
#define circuit_C2    INPUT_PULLUP

#define circuit_type  circuit_C1   // circuit type configured, see circuit design specs

int     switch_high, switch_low;

#define button_switch        2     // digital pin connected to button switch
#define debounce            50     // number of milliseconds to wait for switch to settle once pressed
#define switched          true     // signifies switch has been pressed

#define LED        LED_BUILTIN     // digital pin connected to LED, for testing of switch code only
bool    led_status      =  LOW;    // start with LED off, for testing of switch code only

```

```
void setup() {
  // define the switch circuit type
  pinMode(button_switch, circuit_type); // circuit_type == INPUT or INPUT_PULLUP
  // establish meanings for switch on/off depending on circuit_type
  if (circuit_type == INPUT_PULLUP) {
    // switch is NOT configured with a pull down switch resistor
    switch_high = LOW;  // switch pin goes LOW when switch pressed, i.e. on
    switch_low  = HIGH; // switch pin goes HIGH when switch released, i.e. off
  } else {
    // circuit_type == INPUT, so switch IS configured with a pull down switch resistor
    switch_high = HIGH; // switch pin goes HIGH when switch pressed, i.e. on
    switch_low  = LOW;  // switch pin goes LOW when switch released, i.e. off
  }
  // set LED pin for output, for testing purposes only
  pinMode(LED, OUTPUT);
}
```

```
//
//  Button switch reading, method 1
//  This version of switch reading examines switch once each time the function is called.
//  This allows the code section from which it is called to continue without waiting
//  for the switch press cycle to complete once switching is initiated.
//  Only drawback, of course, is that the design of the calling code must ensure that the
//  switch is regularly tested to catch a change in switch status.
//
bool read_switch_method_1() {
  int             switch_pin_reading;
  // static variables because we need to retain old values between function calls
  static bool     switch_pending = false;
  static long int elapse_timer;
  switch_pin_reading = digitalRead(button_switch);
  if (switch_pin_reading == switch_high) {
    // switch is pressed, so start/restart debounce process
    switch_pending = true;
    elapse_timer = millis();    // start elapse timing
    return !switched;           // now waiting for debounce to conclude
  }
  if (switch_pending && switch_pin_reading == switch_low) {
    // switch was pressed, now released, so check if debounce time elapsed
    if (millis() - elapse_timer > debounce) {
      // dounce time elapsed, so switch press cycle complete
      switch_pending = false;
      return switched;
    }
  }
  return !switched;
}
```

```
//
//  Button switch reading, method 2.
//  This version of switch reading will wait for a switch cycle to complete once it
//  initiated, before control is returned to the calling code.
//  That is, the calling code will be held up once the switch is pressed and until
//  the debounce period has elapsed.
//
bool read_switch_method_2() {
  int      switch_pin_reading;
  bool     switch_status;
  long int elapse_timer;
  switch_status = !switched;                            // assume switch not pressed
  //  read the given switch_pin, if pressed will be HIGH, if not pressed will be LOW
  switch_pin_reading = digitalRead(button_switch);
  do {
    if (switch_pin_reading  == switch_high) {
      // switch has been pressed, now debounce
      switch_status = switched;                         // flag that switch was pressed
      elapse_timer  = millis();
      do {} while (millis() - elapse_timer < debounce); // wait for debounce period to lapse
      switch_pin_reading = digitalRead(button_switch);  // see if switch is still being pressed
    }
  } while (switch_pin_reading == switch_high);          // keep debouncing until switch no longer pressed
  return switch_status;                                 // result is either 'switched' or '!switched'
}

```

```
void loop() {
  do {
    if (read_switch_method_1() == switched) {
      led_status = HIGH - led_status;                   // flip between HIGH and LOW
      digitalWrite(LED, led_status);
    }
  } while (true);
}
```