# Arduino / ESP 32 Microcontrollers

ez_switch_lib Library

Quick Start Guide

## Scope of Use

# Contents

## Purpose & Scope of this Guide

This guide has been prepared to provide an incremental approach and instructions for anyone wishing to explore and use the ez_switch_lib library for the configuration and control of switches, either of type button (momentary) or toggle.  There are many types of switch but these will largely fall into one of these two types – button or toggle.

The ez_switch_lib library is compatible with both Arduino and ESP 32 microcontrollers and is capable of supporting a great number of switches which may be individually wired in any one of three different wiring schemes.

The guide starts off small, with the configuration of a single switch (Example 1), but by the final example (Example 4) the reader will have experience of using the library for developing multi-switch applications and even how these may be linked to other digital outputs and even interrupt service routines (ISRs).

A comprehensive appreciation and understanding of the full scope of the ez_switch_lib library may be found by reference to the Arduino Project Hub ez_switch_lib article, including a User Guide incorporating many examples and a Crib Sheet.

*Note that version 2.0, or later, of the ez_switch_lib library is required for the examples used within this Quick Guide.  If you follow the download links provided then you will be assured of the correct/latest versions.*

## Introduction

This guide provides example switch designs and corresponding sketches that will provide the interested reader with a starting point for using switches with the ez_switch_lib library. The ez_switch_lib library[1] and examples contained in this guide are compatible for both Arduino and ESP 32 microcontrollers.

The Guide offers four examples - the first example configures a single switch whilst the remaining three examples configure four switches (2 x button and 2 x toggle), each wired in one of three different wiring schemes (e.g. external pull down resistors, internal pull up resistors and/or internal pull down resistors). However, the number of switches that can be configured is flexible, from a single switch to many.

The ez_switch_lib library is a non-blocking design and therefore requires a sketch's design to ensure that switches are frequently read/tested, usually within the main loop.  The examples show the easiest and, perhaps, the best way to accomplish this, by constructing a main loop around constant cycling (polling) of the 'agnostic' switch read function (`read_switch`[2]).

All examples may be used with both Arduino and ESP 32 microcontrollers, with suitable changes in pin assignments according to the specific board you wish to use. To show this in operation, the first two examples use an Arduino board and the final two an ESP 32 board. Each example has extensive comments that provide help and guidance. In summary:

- **Example 1 (Arduino)**: Configures a single button switch and provides switch processing in the main loop which flips the internal built in LED to show actuation of the switch.
- **Example 2 (Arduino)**: Configures 4 switches, 2 x button and 2 x toggle switches wired in different schemes, and provides processing in the main loop which again flips the internal built in LED to show actuation of a switch (note that the 4 switches flip the same in-built LED).
- **Example 3 (ESP 32)**: This example is largely the same as Example 2, above, but introduces linked output[3] pins for 3 of the switches - switch GPIO pins 33, 32 and 25 have linked GPIO pins 20, 21 and 22, respectively.  Use of the serial monitor will be made to show the effects of switch actuations and the status of their associated linked output pins. In addition, the in-built LED is also flipped at each switch actuation as before.
- **Example 4 (ESP 32)**: In this final example we use the same sketch as in example 3, but in this instance link (tie) three of the switches to a common linked output pin and then define this common pin as an external interrupt pin which will be processed by a common ISR.  Now, when any of the linked switches are actuated, the ISR is triggered and interrupt processing is then applied.  Note that the ISR provided in this example can be taken as a generic ISR for use by the ez_switch_lib library as it deals is capable of working with different switch types and interrupt trigger types.

The examples include no functionality other than to demonstrate their scope and purpose and should provide sufficient guidance to apply the ez_switch_lib library to your projects.  However, the library is capable of providing further and more sophisticated features than those referenced by the examples in this guide – see the Arduino/ESP Switch Library article on the Arduino Project Hub and the User Guide and Crib Sheet downloadable from github).

---

[1]  ez_switch_lib library version 2.0 or later.

[2]  The `read_switch` function will automatically determine the switch type referenced from the provided switch id parameter – you do not need to specify the switch type.

[3]  The ez_switch_lib library provides a feature that allows us to link a digital output pin to a switch such that when the switch is actuated (button or toggle) the linked output pin status is flipped.  That is, if the linked pin is `HIGH` then the actuation of the associated switch will flip it `LOW` and vice versa.  This feature is microcontroller independent.

## Switch Wiring Schemes, Pin Modes & Debounce

The switches referenced in the examples are wired and configured in one of several wiring schemes. For easy reference, the wiring schemes and pin mode configurations are shown below:

### Switch Wiring Schemes
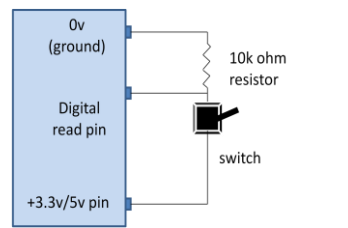
**Figure 1 – circuit_C1 (INPUT)**
Arduino + ESP Boards
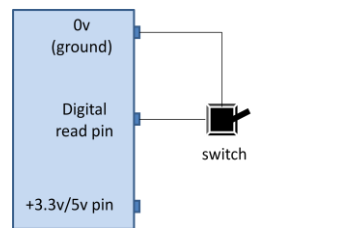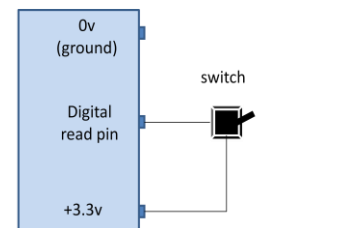pinMode parameter:
INPUT/circuit_C1
External pull down resistor

**Figure 2 – circuit_C2, (INPUT_PULLUP)**
Arduino + ESP Boards
pinMode parameter:
INPUT_PULLUP/circuit_C2
Internal pull up resistor

**Figure 3 – circuit_C3, (INPUT_PULLDOWN)**
ESP Boards ONLY
pinMode parameter:
INPUT_PULLDOWN/circuit_C3
Internal pull down resistor

### pinMode Parameter Microcontroller Compatibility

For completeness and for your reference, the following table shows the valid pinMode parameters for defining input pins for both Arduino and ESP 32 boards:

| pinMode Parameters | | Compatibility | | Notes |
|---|---|---|---|---|
| Native Value | Synonym Value[4] | Arduino | ESP 32 | |
| INPUT | circuit_C1 | ✔ | ✔ | Circuit requires an external pull down resistor, e.g. 10k ohm |
| INPUT_PULLUP | circuit_C2 | ✔ | ✔ | No external resistor required |
| INPUT_PULLDOWN | circuit_C3 | ✘ | ✔ | No external resistor required |

### Switch Debounce

The ez_switch_lib library automatically applies a software debounce technique to all defined switches. By default, the switch debounce interval/period is 10 milliseconds but this can be set to any value required (in milliseconds). The sketch examples in this Guide reset the debounce period to 50 milliseconds within the setup() functions. However, if it is that you experience 'shaky' results when using your switches it is most likely that this interval/period is still too short – the noisier the switch the greater the period this should be set to. If you encounter this issue then change the debounce interval accordingly.
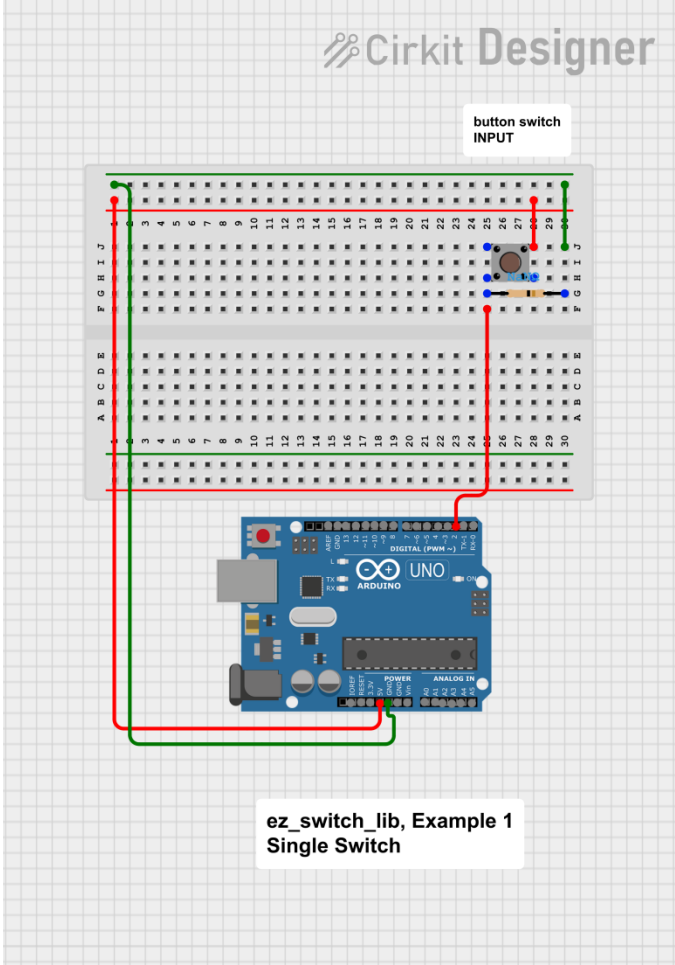
## Installing the ez_switch_lib Library

Before setting off, you will need to download the latest version of the ez_switch_lib library files. This can be done via the Arduino IDE Library Manager – Tools/Manage Libraries and enter "ez_switch_lib" in the dialogue box.

---

[4] Note that "circuit_C1", "circuit_C2" and "circuit_C3" are ez_switch_lib library reserved macros and may be used in place of "INPUT", "INPUT_PULLUP" and "INPUT_PULLDOWN", respectively.

# Arduino - Single Switch, Example 1

## Arduino - Notes & Circuit Design

| Objectives & Notes | Circuit |
|---|---|
| Objectives:<br>• To wire a simple button switch with an external 10k ohm resistor<br>• To understand the use of the ez_switch_lib library – declarations, structure and functions - declare the library, instantiate the library's class for a single switch, define switch data by adding a switch to the class, and read the switch<br>• To witness the operation of the switch by the flipping of the in-built LED<br><br>Notes:<br>• Ensure that your IDE is configured for the Arduino microcontroller you will be using<br>• Download the Example 1 sketch<br>• *For your future reference, if using a `pinMode` parameter for a switch of `INPUT` then always use an external pull down resistor as shown in this example for both Arduino and ESP 32 boards. However, if using an ESP 32 board then the resistor can be dispensed with by changing the `pinMode` parameter from `INPUT` to `INPUT_PULLDOWN`.* | Wire up the circuit shown adjacent with the components shown/below:<br><br>| Components | No. |<br>|---|---|<br>| Arduino microcontroller (e.g. UNO) | 1 |<br>| Breadboard, e.g. mini/half/full size | 1 |<br>| Button switch (momentary) | 1 |<br>| Wires | 5 |<br><br><br><br>ez_switch_lib, Example 1<br>Single Switch |

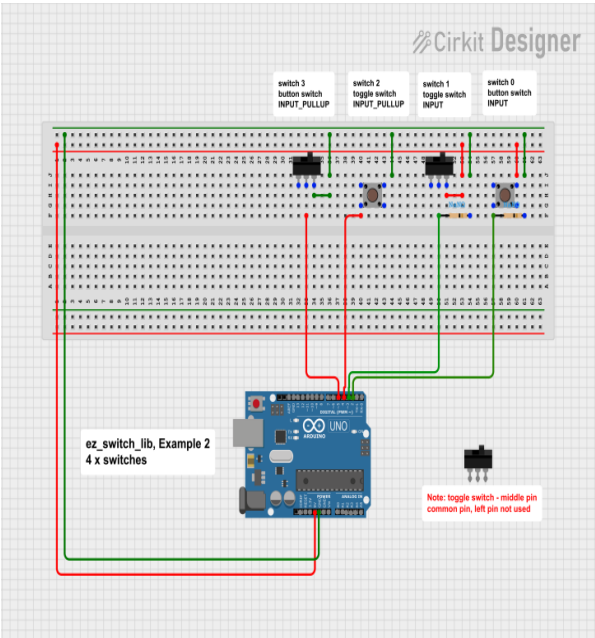## Arduino - Single Switch, Example 1 - Sketch

| Comments | Example Declarations and Code |
|---|---|
| Preamble → | ```<br>//<br>// A basic ez_switch_lib example - Quick Start, example 1<br>// A single switch sketch designed to flip the status of the in-built LED<br>// for Arduino this is on pin 13 (LED_BUILTIN) and for ESP 32 pin 2. This example is<br>// configured for an Arduino microcontroller<br>//<br>// Compatible for both Arduino and ESP 32 microcontrollers<br>//<br>// Add your own code where required, eg where you see '...'<br>//<br>// R D Bentley (Stafford UK)<br>//<br>// This example and code is in the public domain and may only be<br>// used for non-commercial purposes without restriction and<br>// without warranty.<br>``` |
| Declare the ez_switch_lib library → | ```#include <ez_switch_lib.h>``` |
| Add your own specific declarations → | ```...``` |
| We will be defining a single switch, so we will use 'switch_id; to record its id assigned by the add_switch function → | ```#define LED LED_BUILTIN // built in LED on Arduino boards```<br>```int switch_id; // use to record switch id we add to the class``` |
| Add your own specific declarations → | ```...``` |
| Instantiate the library's class for a single switch → | ```Switches my_switch(1); // create class size for 1 switch``` |
| Add your own specific declarations → | ```...``` |
| Use the setup() function to add your declared switches to the library class using the add_switch function.<br><br>Note the INPUT_PULLUP parameter of the add_switch call – this is used for simply connected switches without an external pull down resistor as per circuit diagram. We could have used "circuit_C2", also.<br><br>Note also the ability to check for success or otherwise - the add_switch function returns the switch id assigned to the switch or an error value.<br><br>Once the configuration has been verified through compilation and testing, the error checking can be removed. | ```<br>void setup(){<br>  // create switch with a 10k ohm external pull down resistor - INPUT pinMode parameter<br>  switch_id = my_switch.add_switch(button_switch,// switch type<br>                                   2,            // pin switch connected to<br>                                   INPUT);       // circuit type – pinMode<br>  if (switch_id < 0){// error reported - process error<br>   ...<br>  }<br>  my_switch.set_debounce(50); // set debounce to 50 millisecs<br>  pinMode(LED,OUTPUT); // internal built in LED<br>  ...<br>}<br>``` |

| Add any additional setup code as required. → | |
|---|---|
| Add your own specific code as required → | `...` |
| Use this construct or a similar one to ensure that the declared switch is regularly processed.<br>Note that we use the variable 'switch_id' assigned in the setup function to reference the switch.<br><br>Add your own code as required. → | ```void loop(){``` <br> ```...``` <br> ``` do {``` <br> ```   if (my_switch.read_switch(switch_id) == switched){``` <br> ```     // this switch (swith_id) has been actuated, so process it``` <br> ```     digitalWrite(LED, digitalRead(LED)^1); // flip status of in built LED to show actuation``` <br> ```     ...``` <br> ```   }``` <br> ```   ...``` <br> ``` } while (true);``` <br> ```}``` |

# Arduino - Four Switches, Example 2

## Arduino - Notes & Circuit Design

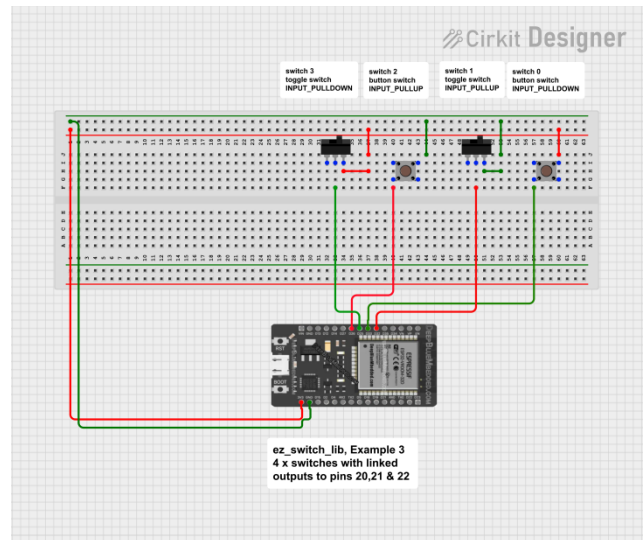| Objectives & Notes | Circuit |
|---|---|
| Objectives: <br> • To understand how we can extend Example 1 to a four switch design, using two different styles of switch (2 x button + 2 x toggle) wired and initialised in different ways (`pinMode` settings) <br> • To compare how each switch is wired with its corresponding pin initialisation `pinMode` parameter (`INPUT` or `INPUT_PULLUP`) – see circuit diagram switch labels text <br> • Witness the operation of the switches by the flipping of the in-built LED <br><br> Notes: <br> • Ensure that your IDE is configured for the Arduino microcontroller you will be using <br> • For multiple switches it is helpful to use a multi-dimensional array to define all of the data specific to each switch.  In this, and following examples, we shall use this construct.  Note that <u>each row</u> of the array will correspond to the switch's <u>id</u>, for example row 2 will relate to switch id 2, etc. <br> • The switch row data defined is: <br>   -   Switch type (`button_switch` or `toggle_switch`) <br>   -   Digital pin allocated to the switch, <br>   -   The switch circuit wiring scheme – `INPUT`/`circuit_C1`, `INPUT_PULLUP`/`circuit_C2` or `INPUT_PULLDOWN`/`circuit_C3` (ESP 32 boards only) <br> • We shall use this same construct in all subsequent examples <br> • Download the **Example 2** sketch. | Wire up the circuit shown adjacent with the components shown/below. <br><br> <table><tr><td>Components</td><td>No.</td></tr><tr><td>Arduino microcontroller (e.g. UNO)</td><td>1</td></tr><tr><td>Breadboard, e.g. mini/half/full size</td><td>1</td></tr><tr><td>Button switch (momentary)</td><td>2</td></tr><tr><td>Toggle switches (e.g. two position)</td><td>2</td></tr><tr><td>10k ohm resistors</td><td>2</td></tr><tr><td>Wires</td><td>14</td></tr></table> <br>  |

## Arduino - Four Switches, Example 2 - Sketch

| Comments | Example Declarations and Code |
|---|---|
| Preamble → | ```<br>//<br>// A basic ez_switch_lib example - Quick Start, example 2<br>// A sketch configured for four switches sketch to flip the status of the in-built LED<br>// for Arduino this is on pin 13 (LED_BUILTIN) and for ESP 32 pin 2. This example is<br>// configured for an Arduino microcontroller<br>//<br>// Compatible for both Arduino and ESP 32 microcontrollers<br>//<br>// Add your own code where required, eg where you see '...'<br>//<br>// R D Bentley (Stafford UK)<br>//<br>// This example and code is in the public domain and may only be<br>// used for non-commercial purposes without restriction and<br>// without warranty.<br>``` |
| Declare the ez_switch_lib library → | `#include <ez_switch_lib.h>` |
| Add your own specific declarations → | `...` |
| (For multiple switches it is helpful to use a multi-dimensional array to define the data specific to each switch.  Each row of the array will correspond to the switch's id, for example row 2 will relate to switch id 2, etc, as far as the ez_switch_lib library functions are concerned.  We shall use this same construct in all subsequent examples.)<br>Declare your specific needs for switches, e.g. number of switches, and for each switch: switch type, GPIO pin number and circuit type/pinMode setting → | ```<br>#define LED LED_BUILTIN // built in LED on Arduino boards<br><br>#define max_switches 4<br>uint8_t My_Switches[max_switches][3]{<br>button_switch, 2, INPUT,          // switch 0<br>toggle_switch, 3, INPUT,          // switch 1<br>button_switch, 4, INPUT_PULLUP,   // switch 2<br>toggle_switch, 5, INPUT_PULLUP}; // switch 3<br>``` |
| Add your own specific declarations → | `...` |
| Instantiate the library's class → | `Switches my_switches(max_switches); // instantiate the class for the required number of switches` |
| Add your own specific declarations → | `...` |
| Use the setup() function to add your declared switches to the library class using the add_switch function.  Note the ability to check for success or otherwise.<br>Once the configuration has been verified through compilation and testing, the error checking can be removed. | ```<br>void setup(){<br> int result;<br> for(uint8_t sw = 0;sw < max_switches;sw++){<br>   result = my_switches.add_switch(My_Switches[sw][0], //switch type<br>                                   My_Switches[sw][1], //pin switch connected to<br>                                   My_Switches[sw][2]);//circuit type – pinMode<br>  if (result < 0){// error reported – process error<br>    ...<br>``` |

| Add any additional setup code as required. → | ```<br>  }<br> }<br> my_switches.set_debounce(50); // set debounce to 50 millisecs<br> pinMode(LED, OUTPUT); // internal built in LED<br> ...<br>}<br>``` |
|---|---|
| Add your own specific code as required → | `...` |
| Use this construct or a similar one to ensure that all declared switches are regularly processed. Add your own code as required. → | ```<br>void loop(){<br> ...<br> do {<br>  for (uint8_t sw = 0;sw < max_switches;sw++){<br>   if (my_switches.read_switch(sw) == switched){<br>     // this switch (sw) has been actuated, so process it<br>     digitalWrite(LED, digitalRead(LED)^1); // flip status of in built LED to show actuation<br>     ...<br>   }<br>   ...<br>  }<br>  ...<br> } while (true);<br>}<br>``` |

# ESP 32 - Four Switches, Three Linked Outputs, Example 3

## ESP 32 - Notes & Circuit Design

| Objectives & Notes | Circuit |
|---|---|
| Objectives:<br>This example builds on Example 2 but uses an ESP 32 microcontroller and introduces the concept of switch linking to output pins.<br><br>• To note that the example introduces a `pinMode` parameter of `INPUT_PULLDOWN` which is not available on Arduino boards. The use of this `pinMode` initialisation parameter means that we are able to dispense with external pull down resistors making the wiring simpler – notice that the circuit diagram does not include any pull down resistors<br>• To explore how we can link switches to outputs - three switches will be defined as having a linked output using the function `link_switch_to_output`. This feature allows us to automatically invert/flip the output pin linked to a switch whenever the switch is actuated. This function is <u>in addition</u> to the normal switch actuation handling found in the main loop<br>• To extend our Example 2 switch data to reference linked outputs<br>• To witness the operation of the switches by the flipping of the in-built LED and the use of the serial monitor to show the inversions/flipping of the linked output pins associated with the switches<br><br>Notes:<br>• Ensure that your IDE is configured for the ESP 32 microcontroller you will be using<br>• Set the serial monitor to 115200 baud<br>• Note that we have 'lost' the 10k ohm resistors<br>• Note that not all ESP 32 GPIO pins have a pull_up/pull_down capability[5].<br>• Download the Example 3 sketch<br>• Why not introduce external LEDs on the breadboard linking these to each of the three linked output pins? You will see that the LEDs will operate without any code being introduced to the sketch. | Wire up the circuit shown adjacent with the components shown/below.<br><br><table><tr><td>Components</td><td>No.</td></tr><tr><td>ESP 32 microcontroller</td><td>1</td></tr><tr><td>Breadboard, e.g. mini/half/full size</td><td>1</td></tr><tr><td>Button switch (momentary)</td><td>2</td></tr><tr><td>Toggle switches</td><td>2</td></tr><tr><td>Wires</td><td>12</td></tr></table><br> |

---

[5]    ESP 32 GPIO pins 34-39 – these can only be set as **input** mode and **do not** have software enabled pullup or pulldown functions. (Source: esspressif)

## ESP 32 - Four Switches, Three Linked Outputs Example 3 - Sketch

| Comments | Example Declarations and Code |
|---|---|
| Preamble → | ```<br>//<br>// A basic ez_switch_lib example - Quick Start, example 3<br>// A sketch configured for four switches sketch.  The sketch will:<br>// 1. flip the status of the in-built LED for all switch actuations<br>// 2. flip the linked output pins defined for 3 of the switches. The serial<br>//    monitor is used to show that the linked pins are flipped each time<br>//    its associated switch is actuated.<br>//<br>// For Arduino this is on pin 13 (LED_BUILTIN) and for ESP 32 pin 2. This example is<br>// configured for an ESP 32 microcontroller<br>//<br>// Compatible for both Arduino and ESP 32 microcontrollers<br>//<br>// Add your own code where required, eg where you see '...'<br>//<br>// R D Bentley (Stafford UK)<br>//<br>// This example and code is in the public domain and may only be<br>// used for non-commercial purposes without restriction and<br>// without warranty.<br>``` |
| Declare the ez_switch_lib library → | `#include <ez_switch_lib.h>` |
| Add your own specific declarations → | `...` |
| Declare your specific needs for switches, e.g. number of switches, and for each switch:<br>   - switch type,<br>   - GPIO pin number,<br>   - circuit type/pinMode setting.<br>If switch linked:<br>   - GPIO output pin to be linked to <u>and</u> the linked pin's initial setting (LOW or HIGH)<br>Otherwise, if not linked:<br>   - 0, 0<br><br>                                    → | ```<br>#define LED 2 // this is the pin of the internal built LED on ESP 32<br><br>#define max_switches 4<br>uint8_t My_Switches[max_switches][5] {<br>  button_switch, 23, INPUT_PULLDOWN, 21,  LOW, // switch 0, start with linked pin LOW<br>  toggle_switch, 22, INPUT_PULLUP,   19, HIGH, // switch 1, start with linked pin HIGH<br>  button_switch, 25, INPUT_PULLUP,    0,    0, // switch 2, no linked pin on this switch<br>  toggle_switch, 26, INPUT_PULLDOWN, 18,  LOW  // switch 3, start with linked pin LOW<br>};<br>``` |
| Add your own specific declarations → | `...` |
| Instantiate the library's class → | `Switches my_switches(max_switches); // instantiate the class for the required number of switches` |
| Add your own specific declarations → | `...` |
| Use the setup() function to add your declared switches to the library class using the add_switch function.  Note the ability to check for success or otherwise. | ```<br>void setup() {<br>  int result;<br>  for (uint8_t sw = 0; sw < max_switches; sw++) {<br>    result = my_switches.add_switch(My_Switches[sw][0], //switch type<br>``` |

| | |
|---|---|
| Once the configuration has been verified through compilation and testing, the error checking can be removed.<br><br>Add any additional setup code as required. → | ```cpp<br>                              My_Switches[sw][1], //GPIO pin switch connected to<br>                              My_Switches[sw][2]);//circuit type – pinMode<br>      if (result < 0) { // error reported – process error<br>        //...<br>      }<br>      if (My_Switches[sw][3] != 0) { // there is an output pin to be linked<br>        result = my_switches.link_switch_to_output(sw, // id of switch to be linked<br>                My_Switches[sw][3], // GPIO link<br>                My_Switches[sw][4]);// initial setting of link<br>        if (result < 0) { // error reported – process error<br>          //...<br>        }<br>      }<br>    }<br>  }<br>  my_switches.set_debounce(50); // set debounce to 50 millisecs<br>  pinMode(LED, OUTPUT); // internal built in LED<br>  Serial.begin(115200); // open serial monitor<br>  //...<br>}<br>``` |
| Add your own specific code as required → | ```cpp<br>...<br>``` |
| Use this construct or a similar one to ensure that all declared switches are regularly processed. Add your own code as required. → | ```cpp<br>void loop() {<br>  //...<br>  do {// keep reading switches each loop<br>    for (uint8_t sw = 0; sw < max_switches; sw++) {<br>      if (my_switches.read_switch(sw) == switched) {<br>        // this switch (sw) has been actuated, so process it<br>        // flip status of in built LED to show actuation for all switches<br>        // but report switch status to the serial monitor also<br>        digitalWrite(LED, digitalRead(LED) ^ 1); // flip in-built LED<br>        // now report status of this sw(itch) which has been actuated<br>        Serial.print("switch ");<br>        Serial.print(sw);<br>        Serial.print(" (");<br>        if (My_Switches[sw][0] == button_switch)Serial.print("button switch)"); else<br>        Serial.print("toggle switch)");<br>        Serial.print(" actuated on pin ");<br>        Serial.print(My_Switches[sw][1]);        // the pin this sw(itch) is connected to<br>        uint8_t linked_pin = My_Switches[sw][3]; // the pin this sw(itch) is linked to<br>        if (linked_pin != 0) {<br>          // this switch has a linked output so read that pins current status<br>          // and report<br>          bool pin_status = digitalRead(linked_pin);<br>          Serial.print(", linked pin is ");<br>          Serial.print(linked_pin);<br>          Serial.print(", current status is ");<br>``` |

```
                          if (pin_status == HIGH)Serial.println("HIGH"); else Serial.println("LOW");
                        } else {
                          Serial.println(", no linked pin");
                        }
                        Serial.flush();
                        //...
                      }
                      //...
                    }
                    //...
                  } while (true);
                }
```
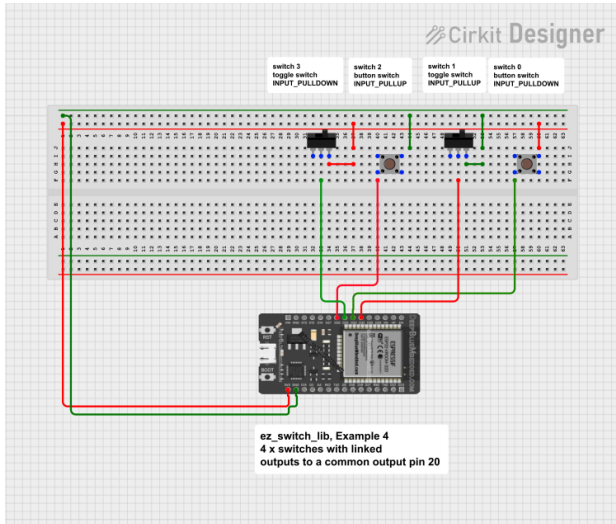
## Serial Monitor Output from Example 3

Actuating each of the switches produces the following output, note the automatic changing of each of the linked output pins. Compare the output pin status with the value initially assigned :

```
switch 0 (button switch) actuated on pin 23, linked pin is 21, current status is HIGH
switch 1 (toggle switch) actuated on pin 22, linked pin is 19, current status is LOW
switch 1 (toggle switch) actuated on pin 22, linked pin is 19, current status is HIGH
switch 2 (button switch) actuated on pin 25, no linked pin
switch 3 (toggle switch) actuated on pin 26, linked pin is 18, current status is HIGH
switch 3 (toggle switch) actuated on pin 26, linked pin is 18, current status is LOW
switch 0 (button switch) actuated on pin 23, linked pin is 21, current status is LOW
switch 1 (toggle switch) actuated on pin 22, linked pin is 19, current status is LOW
switch 1 (toggle switch) actuated on pin 22, linked pin is 19, current status is HIGH
switch 2 (button switch) actuated on pin 25, no linked pin
switch 3 (toggle switch) actuated on pin 26, linked pin is 18, current status is HIGH
switch 3 (toggle switch) actuated on pin 26, linked pin is 18, current status is LOW
```

# ESP 32 - Four Switches, Three Linked Outputs, One ISR, Example 4

## ESP 32 - Notes & Circuit Design

| Objectives & Notes | Circuit |
|---|---|
| Objectives:<br><br>• This example builds on Example 3, again using an ESP 32 microcontroller, and introduces the concept of switch linking to a <u>common</u> output pin to which we shall attach an external interrupt handling routine (ISR).<br>• To link out switches to a common output pin - three switches will be defined as having a linked output using the function `link_switch_to_output`.<br>• To understand the constraints in using multiple common output pins with a common ISR - switch data will be declared as for the previous examples but take careful note of the sketch's comments relating to the initial settings of the linked output pin with respect to the interrupt trigger type<br>• To understand how we are able to define an ISR within the setup process - the sketch will define an ISR, using `attachInterrupt` to handle interrupts on the common output pin which may be triggered by <u>any</u> of the three switches<br>• To see that whenever one of the three linked switches is actuated, the associated common ISR is triggered <u>before</u> the switch can be handled by the main loop (the calling switch read function). <u>Post</u> ISR, processing of the actuating switch may then be applied as required in the main loop<br>• The ISR included within the sketch can be taken as a standard/generic ISR for this library as it deals with button switches, toggle switches switched to 'on' and toggle switches switched to 'off'<br>• Like Example 3, to witness switch actuations and ISR triggering by the use of the in-built LED, an ISR counter and the serial monitor<br><br>Notes:<br>1. Ensure that your IDE is configured for the ESP 32 microcontroller you will be using<br>2. Note that we are using the <u>same</u> circuit design as for Example 3. The difference here is that the ESP 32 linked pin outputs are configured for the <u>same</u> pin<br>3. Download the Example 4 sketch. | Wire up the circuit shown adjacent with the components shown/below.<br><br>| Components | No. |<br>|---|---|<br>| ESP 32 microcontroller | 1 |<br>| Breadboard, e.g. mini/half/full size | 1 |<br>| Button switch (momentary) | 2 |<br>| Toggle switches | 2 |<br>| Wires | 12 |<br><br> |

## ESP 32 - Four Switches, Three Linked Outputs, One ISR Example 4 - Sketch

| Comments | Example Declarations and Code |
|---|---|
| Preamble → | ```<br>//<br>// A basic ez_switch_lib example - Quick Start, example<br>// A sketch configured for four switches sketch.  The sketch will<br>// demonstrate how we are able to use external interrupt routines (ISRs)<br>// by linking switches to output pins such that when a switch is actuated<br>// the associated ISR is triggered.<br>// The in-built LED is used to indicate a switch actuation, plus the serial<br>// monitor is also used to confirm that the ISR is entered for any switch<br>// for which a linked output is defined.<br>//<br>// For Arduino this is on pin 13 (LED_BUILTIN) and for ESP 32 pin 2. This example is<br>// configured for an ESP 32 microcontroller<br>//<br>// Compatible for both Arduino and ESP 32 microcontrollers<br>//<br>// Add your own code where required, eg where you see '...'<br>//<br>// R D Bentley (Stafford UK)<br>//<br>// This example and code is in the public domain and may only be<br>// used for non-commercial purposes without restriction and<br>// without warranty.<br>``` |
| Declare the ez_switch_lib library → | `#include <ez_switch_lib.h>` |
| Add your own specific declarations → | `...` |
| Declare:<br>• The GPO pin to be used for the external interrupt pin – `common_interrupt_pin`<br><br>• Your specific needs for switches, e.g. number of switches, and for each switch:<br><br>- switch type,<br>- GPIO pin number,<br>- circuit type/pinMode setting.<br>If switch linked:<br>- GPIO output pin to be linked to and the linked pin's initial setting (LOW or HIGH)<br>Otherwise, if not linked:<br>- 0, 0 | ```<br>#define LED 2 // this is the pin of the internal built LED on ESP 32<br><br>#define common_interrupt_pin 21 // external (common) interrupt GPIO pin that all linked<br>                                // switches will use<br><br>#define max_switches 4<br><br>// This array (My_Switches)includes all of the configuration data for the switch(es)<br>// to be defined. Switch types and wiring schemes may be mixed.  Each row<br>// of the array references the configuration data for each switch to be defined.<br>// My_Switches array - rows are defined as:<br>// [row][0] - switch type, either 'button_switch' or 'toggle_switch', these are<br>//            reserved library macros and may be used without declaration<br>// [row][1] - digital pin assigned to the switch<br>// [row][2] - circuit type switch is wired as and initialised for via a pinMode call:<br>//            This parameter must take one of three values:<br>//            1. INPUT, or circuit_C1. This requires an external 10k ohm pull down<br>resistor<br>``` |

| | |
|---|---|
| • The interrupt trigger type, either RISING, FALLING or CHANGE. Note that LOW and HIGH are not supported) → | ```
//             to be wired in the switch circuit
//          2. INPUT_PULLUP, or circuit_C2. NO external pull up resistor is required
//          3. INPUT_PULLDOWN, or circuit_C3. NO external pull down resistor is required
//
//          Note: INPUT, INPUT_PULLUP, INPUT_PULLDOWN, circuit_C1, circuit_C2 &
circuit_C3
//          are all library reserved macros and may be used without any declaration
//
// [row][3] - digital pin number of any linked output pin to this switch
// [row][4] - If a switch has a linked output pin, this is the value for the pin
//          to be initialised with (HIGH or LOW) when linked via the function
//          link_switch_to_output.
//          NOTE:
//          1. this parameter must be the SAME for ALL switches
//             linked to the SAME common interrupt pin
//          2. if the interrupt is triggered on
//             a. RISING, then this value must be set LOW
//             b. FALLING, then this value must be set HIGH
//             c. CHANGE, then this parameter can be either LOW or HIGH

volatile uint8_t My_Switches[max_switches][5] {
  button_switch, 23, INPUT_PULLDOWN, common_interrupt_pin, HIGH,// switch 0
  toggle_switch, 22, INPUT_PULLUP,   common_interrupt_pin, HIGH,// switch 1
  button_switch, 25, INPUT_PULLUP,                      0,    0,// switch 2, no linked pin
for switch
  toggle_switch, 26, INPUT_PULLDOWN, common_interrupt_pin, HIGH // switch 3
};

volatile const uint8_t interrupt_trigger_type = CHANGE;   // can be RISING, FALLING or
CHANGE

volatile uint16_t ISR_count = 0;    // used to show the ISR is being entered
``` |
| Add your own specific declarations → | ... |
| Instantiate the library's class → | **Switches** my_switches(max_switches); // instantiate the class for the required number of switches |
| Add your own specific declarations → | ... |
| Use the setup() function to add your declared switches to the library class using the add_switch function and make the defined linkages to the declared GPIO output pins using link_switch_to_output function. Note the ability to check for success or otherwise.<br><br>Once the configuration has been verified through | ```
void setup() {
  int result;
  for (uint8_t sw = 0; sw < max_switches; sw++) {
    result = my_switches.add_switch(My_Switches[sw][0], //switch type
                                    My_Switches[sw][1], //GPIO pin switch connected to
                                    My_Switches[sw][2]);//circuit type - pinMode
    if (result < 0) { // error reported - process error
      //...
``` |

| | |
|---|---|
| compilation and testing, the error checking can be removed.<br><br>Add any additional setup code as required. → | ```cpp<br>      }<br>    if (My_Switches[sw][3] != 0) { // there is an output pin to be linked<br>      result = my_switches.link_switch_to_output(sw, // id of switch to be linked<br>              My_Switches[sw][3], // GPIO link<br>              My_Switches[sw][4]);// initial setting of link<br>      if (result < 0) { // error reported – process error<br>        //...<br>      }<br>    }<br>  }<br>    attachInterrupt(digitalPinToInterrupt(common_interrupt_pin),<br>                switch_ISR,// name of the sketch's ISR handler for switch interrupts<br>                interrupt_trigger_type);// how the interrupt will be triggered<br>  }<br>  my_switches.set_debounce(50); // set debounce to 50 millisecs<br>  pinMode(LED, OUTPUT); // internal built in LED<br>  Serial.begin(115200);<br>  //...<br>}<br>//...<br>``` |
| Add your own specific code as required → | ```cpp<br>...<br>``` |
| Use this construct or a similar one to ensure that all declared switches are regularly processed. Add your own code as required. → | ```cpp<br>//...<br>void loop() {<br>  //...<br>  do {// keep reading switches each loop<br>    for (uint8_t sw = 0; sw < max_switches; sw++) {<br>      if (my_switches.read_switch(sw) == switched) {<br>        // this switch (sw) has been actuated, so process it<br>        // flip status of in built LED to show actuation for all switches<br>        // but report switch status to the serial monitor also<br>        digitalWrite(LED, digitalRead(LED) ^ 1); // flip in-built LED<br>        // now report status of this sw(itch) which has been actuated<br>        Serial.print("switch ");<br>        Serial.print(sw);<br>        Serial.print(" (");<br>        if (My_Switches[sw][0] == button_switch)Serial.print("button switch)");<br>        else Serial.print("toggle switch)");<br>        Serial.print(" actuated on pin ");<br>        Serial.print(My_Switches[sw][1]);        // the pin this sw(itch) is connected to<br>        uint8_t linked_pin = My_Switches[sw][3]; // the pin this sw(itch) is linked to<br>        if (linked_pin != 0) {<br>          // this switch has a linked output that triggered its ISR<br>          // so display the ISR count<br>          Serial.print(", ISR linked pin is ");<br>          Serial.print(linked_pin);<br>``` |

```
              Serial.print(", ISR counter = ");
              Serial.println(ISR_count);
          } else Serial.println();
          Serial.flush();
          //...
        }
        //...
      }
      //...
    } while (true);
}
```

| | |
|---|---|
| Use this ISR function as a framework to add your own specific to purpose code.  Note that it deals with both button (momentary) switches and toggle switches, so you will be able to deal explicitly with which switch has triggered the interrupt and, if it is a toggle, if it is in the on or off state.<br><br>The last part of the ISR is housekeeping and should not be modified. | <pre>//<br>// A common ISR associated with defined switches with linked outputs (standard<br>// feature of the ez_switch_ib library).<br>// Note that this ISR is a common framework that can be used for all uses<br>// of the ez_switch_lib library where switches are linked to output(s) that are<br>// configured as external interrupts.  The ISR indicates where end user code<br>// should be inserted, depending on the switch type triggering the ISR.<br>//<br>void switch_ISR() {<br>  ISR_count++; //increment ISR counter, for demo only<br>  uint8_t switch_id = my_switches.last_switched_id; // switch id of switch currently switched<br>  if (my_switches.switches[switch_id].switch_type == button_switch) {<br>    // this was a button switch triggering the interrupt<br>    // *** Button switch processing<br>    // *** Put end user code here to deal with this event<br>    //...<br>  } else {<br>    // this was a toggle switch triggering the interrupt<br>    // determine if the switch transitioned to 'on' or 'off'<br>    // and process accordingly<br>    if (my_switches.switches[switch_id].switch_status == on) {<br>      // *** Toggle switch processing for switch being 'on'<br>      // *** Put end user code here to deal with this event<br>      //...<br>    } else {<br>      // *** Toggle switch processing for switch being 'off'<br>      // *** Put end user code here to deal with this event<br>      //...<br>    }<br>  }<br>  // Finish up, with housekeeping tasks...<br>  if (interrupt_trigger_type != CHANGE) {</pre> |

```
        // not CHANGE, so we need to reset the interrupt pin
        bool level;
        // Deal with the switch control structure data first:
        //   reset interrupt pin (linked output pin) status back to previous value
        //   (i.e. invert it), ready for next read if interrupt trigger type is
        //   RISING or FALLING. If it is CHANGE, then we leave it as is.
        if (interrupt_trigger_type == RISING)level = LOW; else level = HIGH;
        my_switches.switches[switch_id].switch_out_pin_status = level;

        // Now deal with the physical interrupt pin (linked output pin) level:
        //   if the attachInterrupt function trigger parameter is:
        //      "RISING"  then the common interrupt pin must be reset to LOW
        //      "FALLING" then the common interrupt pin must be reset to HIGH
        //      "CHANGE"  then we DO NOT reset the common interrupt pin
        digitalWrite(my_switches.switches[switch_id].switch_out_pin, level);

    } else {
        // trigger type is CHANGE, which the code here is a bit more involved!
        uint8_t ISR_pin = My_Switches[switch_id][3]; // pin defined for this interrupt
        bool level = my_switches.switches[switch_id].switch_out_pin_status;
        for (uint8_t sw = 0; sw < max_switches; sw++) {
          if (my_switches.switches[sw].switch_out_pin != 0) {
            // linked output pin associated with this switch, but only
            // perform update if it is the same pin number as the one
            // that triggered this interrupt
            if (my_switches.switches[sw].switch_out_pin == ISR_pin) {
              // make this switch (sw) same level as the actual switch
              // that triggered the interrupt
              my_switches.switches[sw].switch_out_pin_status = level;
            }
          }
        }
    }
} // End of switch_ISR
```

## Serial Monitor Output from Example 4

Actuating each of the switches produces the following output, note the automatic ISR count generated by the ISR:

```
switch 0 (button switch) actuated on pin 23, ISR linked pin is 21, ISR counter = 1
switch 1 (toggle switch) actuated on pin 22, ISR linked pin is 21, ISR counter = 2
switch 1 (toggle switch) actuated on pin 22, ISR linked pin is 21, ISR counter = 3
switch 2 (button switch) actuated on pin 25
switch 3 (toggle switch) actuated on pin 26, ISR linked pin is 21, ISR counter = 4
switch 3 (toggle switch) actuated on pin 26, ISR linked pin is 21, ISR counter = 5
switch 0 (button switch) actuated on pin 23, ISR linked pin is 21, ISR counter = 6
switch 1 (toggle switch) actuated on pin 22, ISR linked pin is 21, ISR counter = 7
switch 1 (toggle switch) actuated on pin 22, ISR linked pin is 21, ISR counter = 8
switch 2 (button switch) actuated on pin 25
switch 3 (toggle switch) actuated on pin 26, ISR linked pin is 21, ISR counter = 9
switch 3 (toggle switch) actuated on pin 26, ISR linked pin is 21, ISR counter = 10
```

## Conclusion

Hopefully, you have enjoyed working with and through this Quick Start Guide. If you have not already done so and would like to explore further examples and capabilities of the ez_switch_lib library then have a look at the Arduino Project Hub ez_switch_lib article, User Guides, associated examples and Crib Sheet (links) above.