**ESP32 asynchronous multi-button library**

**J. Lemaire**

Pierrefeu – 2023 December

# Objectives

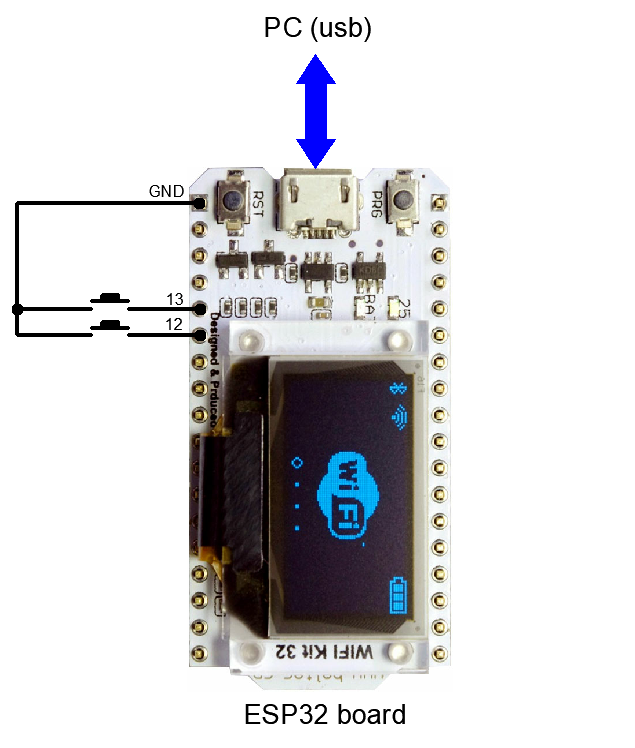
A button library is an old subject and there are many candidates on internet[[1]](#footnote-1) [[2]](#footnote-2) !

But, with the following constraints :

* Can use many push buttons, each on a different pin ;
* Short, double (short) and long press detection ;
* Debounce protection ;
* Interrupt driven with embeded callbacks in the library (asynchronous treatment) ;
* Can run on ESP32 with the Espressif Arduino Core ;
* As simple as possible !

apparently no candidates… But may be I omitted one and apologize the author(s) !

So, I choose to develop such a library, convenient for me, and tested it using this very simple circuit with an ESP32 board[[3]](#footnote-3) :

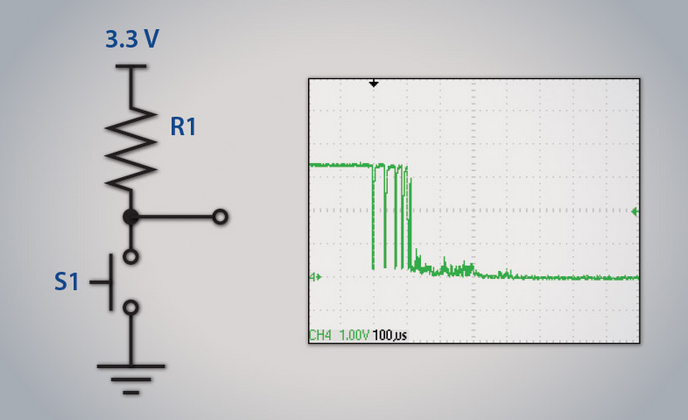


# Debounce management and detect the press action

Here, each push button is connected to the ground and to a pin which GPIO has an internal pull-up resistance[[4]](#footnote-4) ; hence, the **digitalRead** fonction will return its level :

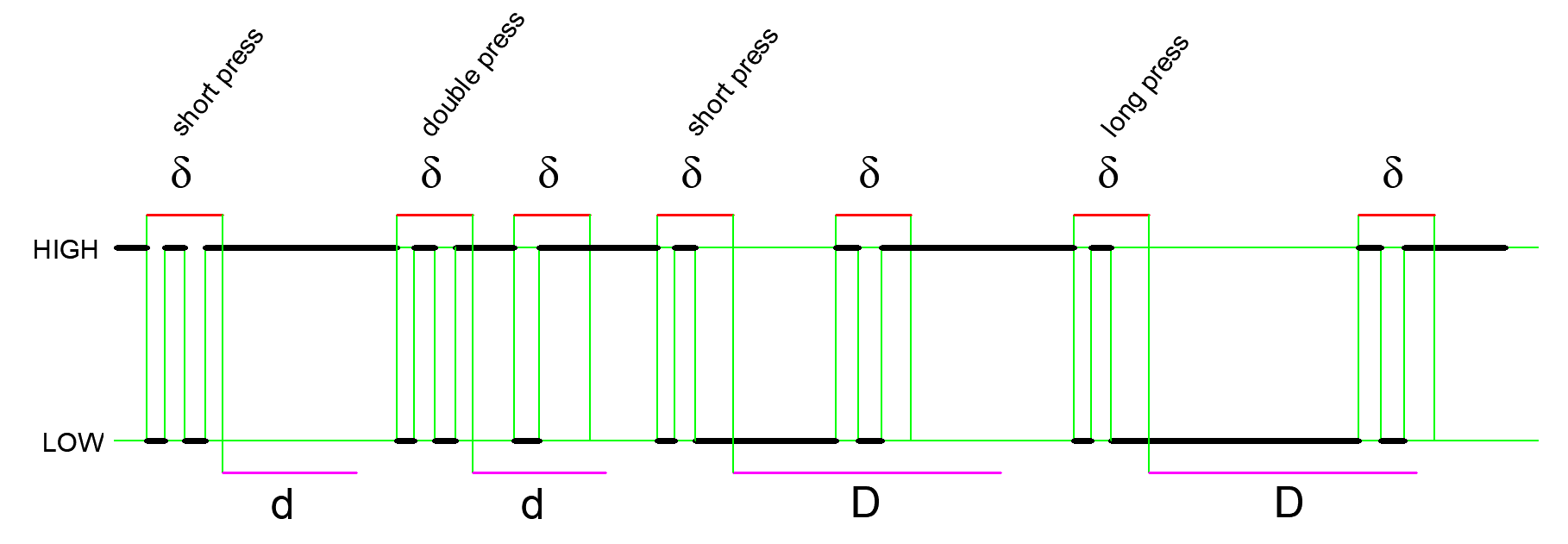
* LOW if the button is pushed ;
* HIGH if the button is released.

Unfortunately and due to the contact and its imperfections, an action on such a button doesn’t produce immediatly the expected change in the GPIO level, but in fact a serie of alternances during a short period ; e.g. :

 [[5]](#footnote-5)

Knowing that these changes can be detected using an external interrupt and an attached callback procedure (ISR[[6]](#footnote-6)), we just need here to do nothing during a laps **δ** (0.25s e.g.) of time, so that only one change can be registred.

Otherwise, it’s also important to evaluate the GPIO level after this delay δ of stabilisation and to measure the delay between 2 successive evaluations, with a view to make the difference between short, double (short ) and long press actions on a button :



More precisely, considering a little d (0.4s e.g.) and big D (1s e.g.) delay :

|  |  |  |  |
| --- | --- | --- | --- |
| **From** | **To** | **Delay from**  **previous eval.** | **Action** |
| HIGH | HIGH | > d | short press |
| HIGH | HIGH | <= d | double press |
| LOW | HIGH | < D | short press (maintained) |
| LOW | HIGH | >= D | long press |

Practically, in the ISR procedure, an alarm timer is enabled which calls, only one time after the delay **δ**, a second ISR procedure with the following tasks :

* Evaluate the new level, using **digitalRead ;**
* Discard the case where different buttons are pressed at the same time ;
* If necessary, enable another timer which calls, only one time after the delay d or D, a third or fourth ISR procedure ;
* Determine the action on the button as previously explained and store it ;
* Reactive the treatment of the change interrupt on the GPIO.

# The MButton Class

A simple library declares the **MButton** class in the the file **MButton.h**:

// MButton.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ESP32 Library to detect actions on several push buttons

Functionalities :

- Acceptable actions : short, double (short) and long press button.

- Works asynchonously, using change and timer interrupts.

- Debounce protection

- Calls the Ticker library.

Restrictions :

- Each button is connected to the ground and a different pin.

- Their gpios need to have a pull-up resistance (e.g. pin number 12, 13, etc.).

- Only one button can be pressed at a time (no buffer to register the actions).

- Need to wait that an action has been processed before a new action.

- 2\*DELAY\_DEBOUNCE > DELAY\_DOUBLE\_PRESS so that maintained press is not possible in a double press

Usage :

- Declare an instance of the class.

- In the setup function call begin(<no Buttons>, <num1>, <num2>, ...) to precise the pins and attach their callbacks.

- In the loop function call successively toProccess(), getNum(), getAction(), processed().

- Can call end() to detach callbacks.

Reference :

https://github.com/jlemaire06/Esp32-async-multi-button-library

https://github.com/espressif/arduino-esp32/tree/master/libraries/Ticker

This program is free software: you can redistribute it and/or modify

it under the terms of the GNU General Public License as published by

the Free Software Foundation, either version 3 of the License, or

(at your option) any later version.

This program is distributed in the hope that it will be useful,

but WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

GNU General Public License for more details.

You should have received a copy of the GNU General Public License

along with this program. If not, see http://www.gnu.org/licenses/.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef \_MBUTTON\_H

#define \_MBUTTON\_H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Libraries and types

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "Arduino.h"

#include <Ticker.h>

enum ButtonAction

**{**

BA\_SHORT\_PRESS**,**

BA\_DOUBLE\_PRESS**,**

BA\_LONG\_PRESS

**};**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Constants

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define DELAY\_DEBOUNCE 250 // Delay for debounce (ms)

#define DELAY\_DOUBLE\_PRESS 500 // Delay for double press (ms)

#define DELAY\_LONG\_PRESS 1000 // Delay for long press (ms)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MButton class

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class MButton

**{**

public**:**

MButton**();** // Constructor

void begin**(**int nbButtons**,** **...);** // To define the pin numbers and attach callbacks

void end**();** // To detach the callbacks

bool toProcessM**()** **{return** toProcess**;}** // True if there is a pending action on the button

int getNum**()** **{return** num**;}** // Pin number

ButtonAction getAction**()** **{return** action**;}** // Action on the button

void processed**();** // To call after action treatment (needed !)

private**:**

static volatile int num**;** // Pin number

static volatile bool toProcess**;** // True if action to process

static volatile ButtonAction action**;** // Action on the button

static bool okButton**;** // To avoid debounce

static int oldNum**;** // To save pin number so that mixed action are forbidden

static int oldLevel**;** // To save pin level for detecting actions (LOW or HIGH)

static Ticker timerDebounce**;** // Alarm timer for debounce protection

static Ticker timerDoublePress**;** // Alarm timer for double press detection

static Ticker timerLongPress**;** // Alarm timer for long press detection

static bool timerDoublePressActive**;** // State of the double press timer

static bool timerLongPressActive**;** // State of the long press timer

static void IRAM\_ATTR buttonInterrupt12**();**

static void IRAM\_ATTR buttonInterrupt13**();**

// Here can add ISR for other pins

static void IRAM\_ATTR timerDebounceInterrupt**();**

static void IRAM\_ATTR timerDoublePressInterrupt**();**

static void IRAM\_ATTR timerLongPressInterrupt**();**

**};**

#endif

The implementation is in the file **MButton.cpp**:

**// MButton.cpp**

#include "MButton.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MButton class

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Implement static members

volatile int MButton**::**num**;**

volatile bool MButton**::**toProcess**;**

volatile ButtonAction MButton**::**action**;**

bool MButton**::**okButton**;**

int MButton**::**oldNum**;**

int MButton**::**oldLevel**;**

Ticker MButton**::**timerDebounce**;**

Ticker MButton**::**timerDoublePress**;**

Ticker MButton**::**timerLongPress**;**

bool MButton**::**timerDoublePressActive**;**

bool MButton**::**timerLongPressActive**;**

MButton**::**MButton**()**

**{**

okButton **=** **true;**

oldLevel **=** HIGH**;**

toProcess **=** **false;**

timerDoublePressActive **=** **false;**

timerLongPressActive **=** **false;**

**}**

void MButton**::**begin**(**int nbMButtons**,** **...)**

**{**

va\_list arg**;**

va\_start**(**arg**,** nbMButtons**);**

**for** **(**uint8\_t i **=** 0**;** i **<** nbMButtons**;** i**++)**

**{**

uint8\_t num **=** va\_arg**(**arg**,** int**);**

pinMode**(**num**,** INPUT\_PULLUP**);**

**switch** **(**num**)**

**{**

**case** 13**:**

attachInterrupt**(**13**,** buttonInterrupt13**,** CHANGE**);**

**break;**

**case** 12**:**

attachInterrupt**(**12**,** buttonInterrupt12**,** CHANGE**);**

**break;**

// Here can add attachments for other pins

**}**

**}**

**}**

void MButton**::**end**()**

**{**

detachInterrupt**(**12**);**

detachInterrupt**(**13**);**

// Here can add detachments for other pins

timerDebounce**.**detach**();**

timerDoublePress**.**detach**();**

timerLongPress**.**detach**();**

**}**

void MButton**::**processed**()**

**{**

toProcess **=** **false;**

**}**

void IRAM\_ATTR MButton**::**buttonInterrupt12**()**

**{**

**if** **(**okButton **&&** **!**toProcess**)**

**{**

num **=** 12**;**

okButton **=** **false;**

timerDebounce**.**once\_ms**(**DELAY\_DEBOUNCE**,** timerDebounceInterrupt**);**

**}**

**}**

void IRAM\_ATTR MButton**::**buttonInterrupt13**()**

**{**

**if** **(**okButton **&&** **!**toProcess**)**

**{**

num **=** 13**;**

okButton **=** **false;**

timerDebounce**.**once\_ms**(**DELAY\_DEBOUNCE**,** timerDebounceInterrupt**);**

**}**

**}**

void MButton**::**timerDebounceInterrupt**()**

**{**

int level **=** digitalRead**(**num**);**

**if** **(!**toProcess**)**

**{**

**if** **(**oldLevel **==** HIGH**)**

**{**

**if** **(**level **==** HIGH**)**

**{**

**if** **(**timerDoublePressActive**)**

**{**

**if** **(**num **!=** oldNum**)** num **=** oldNum**;** // Forbidden mixed case

**else**

**{**

action **=** BA\_DOUBLE\_PRESS**;**

timerDoublePress**.**detach**();** // Stop timer double press

timerDoublePressActive **=** **false;**

toProcess **=** **true;**

**}**

**}**

**else** // First short press

**{**

timerDoublePress**.**once\_ms**(**DELAY\_DOUBLE\_PRESS**,** timerDoublePressInterrupt**);** // Activate timer double press

timerDoublePressActive **=** **true;**

oldNum **=** num**;**

**}**

**}**

**else** // Level == LOW

**{**

timerLongPress**.**once\_ms**(**DELAY\_LONG\_PRESS**,** timerLongPressInterrupt**);** // Activate timer long press

timerLongPressActive **=** **true;**

oldNum **=** num**;**

oldLevel **=** LOW**;**

**}**

**}**

**else** // oldLevel = LOW

**{**

**if** **((**level **==** HIGH**))**

**{**

**if** **(**num **!=** oldNum**)** num **=** oldNum**;** // Forbidden mixed case

**else**

**{**

**if** **(!**timerLongPressActive**)** action **=** BA\_LONG\_PRESS**;**

**else** // Maintained Short press

**{**

timerLongPress**.**detach**();** // Stop timer long press

timerLongPressActive **=** **false;**

action **=** BA\_SHORT\_PRESS**;**

**}**

toProcess **=** **true;**

oldLevel **=** HIGH**;**

**}**

**}**

**}**

**}**

okButton **=** **true;** // Enable button interrupts

**}**

void MButton**::**timerDoublePressInterrupt**()**

**{**

timerDoublePressActive **=** **false;**

action **=** BA\_SHORT\_PRESS**;**

toProcess **=** **true;**

**}**

void MButton**::**timerLongPressInterrupt**()**

**{**

timerLongPressActive **=** **false;**

**}**

It’s not difficult to understand what is realized in this library and why.

* As announced, an **MButton** class instance can manage several push buttons. They are precised on calling the **begin** method which accepts a variable number of parameters ; e.g. **begin(2, 12, 13)** in our case.
* The crucial point, as previously introduced, is the asynchronous treatment using interrupts and the definition of their attached ISR functions. They need to be static[[7]](#footnote-7) and then only use static variables. For that reason, we have opted to declare static variables members ; we can do it because only one button can be pressed at a time and because the actions are not bufferized.
* Each pin, connected to a push button, has its own ISR. Observe that it can do something only when **okButton** is true (debounce condition) and **toProcess** is false (cannot register a new action until the previous action is processed). This ISR stores in the variable **num** the number of the pin, sets **okButton** at false and starts the alarm debounce timer. Obviously, if other pins are considered, the library need to be upgraded a little.
* Here the alarm timer is an hardware one[[8]](#footnote-8) and we use a static instance **timerDebounce** of the **Ticker** class in the **Ticker** library added with the Arduino Core for ESP32[[9]](#footnote-9) ; we just have to call :

timerDebounce**.**once\_ms**(**DELAY\_DEBOUNCE**,** timerDebounceInterrupt**);**

to start the timer so that the ISR **timerDebounceInterrupt** should be called only one time, after **DELAY\_DEBOUNCE** ms.

* The ISR **timerDebounceInterrupt** performs the tasks described in the previous paragraph. In its implementation, 2 other timers are enabled :

timerDoublePress**.**once\_ms**(**DELAY\_DOUBLE\_PRESS**,** timerDoublePressInterrupt**);**

timerLongPress**.**once\_ms**(**DELAY\_LONG\_PRESS**,** timerLongPressInterrupt**);**

and the 2 callbacks procedures are implemented according the detection process.

* And finally, some public methods are provided, **constructor**, **begin**, **toProcessM**, **getNum**, **getAction**, **processed** and **end**, to access to the library as described in the following example.

The Arduino sketch :

// MButton\_test.ino

// Multi button (on pins 12 and 13 here) library test

#include "MButton.h"

#include <elapsedMillis.h>

#define DELAY\_TEST 60000 // Delay for the test (ms)

MButton mButton**;**

elapsedMillis timerTest**;**

bool active**;**

void setup**()**

**{**

Serial**.**begin**(**115200**);**

**while(!**Serial**);**

mButton**.**begin**(**2**,** 12**,** 13**);**

timerTest **=** 0**;**

active **=** **true;**

Serial**.**println**(**"Test started"**);**

**}**

void loop**()**

**{**

**if** **(**timerTest **<** DELAY\_TEST**)**

**{**

**if** **(**mButton**.**toProcessM**())**

**{**

Serial**.**print**(**mButton**.**getNum**());**

**switch** **(**mButton**.**getAction**())**

**{**

**case** BA\_SHORT\_PRESS**:**

Serial**.**println**(**" short pressed"**);**

**break;**

**case** BA\_DOUBLE\_PRESS**:**

Serial**.**println**(**" double pressed"**);**

**break;**

**case** BA\_LONG\_PRESS**:**

Serial**.**println**(**" long pressed"**);**

**break;**

**}**

mButton**.**processed**();**

**}**

**}**

**else** **if** **(**active**)**

**{**

mButton**.**end**();**

active **=** **false;**

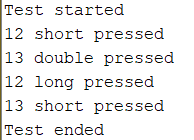
Serial**.**println**(**"Test ended"**);**

**}**

delay**(**500**);**

**}**

And finally an output example :



**Remarks**

* Here, no critical section seems to be needed if we respect the constraints on the actions. But, often they are needed and in this situation a classical solution consist in using **portMUX\_TYPE** variables[[10]](#footnote-10), because the Arduino core for ESP32 doesn’t implement the classical **cli()** and **sti()** functions :
* Declaration :

portMUX\_TYPE timerMux **=** portMUX\_INITIALIZER\_UNLOCKED**;**

* Usage :

portENTER\_CRITICAL\_ISR**(&**timerMux**);**

// Here critical section

portEXIT\_CRITICAL\_ISR**(&**timerMux**);**

* We opted to declare an ISR for each pin connected to a push button : it’s simple and not really difficult to consider other pins. However, there is a solution to avoid this constraint and use only one ISR for all the pins : store the state of all the pins and detect what pin has changed, as proposed in the Alarm-Siren library[[11]](#footnote-11). May be there exist simpler solutions but I don’t found !

1. E.g. <https://github.com/LennartHennigs/Button2> but with no interrupts. [↑](#footnote-ref-1)
2. E.g. <https://github.com/Alarm-Siren/arduino-buttons> but without managing long press action… and with empty arduino sketches ! [↑](#footnote-ref-2)
3. HELTEC Wifi kit 32 : <https://heltec.org/project/wifi-kit-32/> . [↑](#footnote-ref-3)
4. E.g. gpio number 34 to 39 does’nt have internal resistance. [↑](#footnote-ref-4)
5. From <https://www.nuvation.com/resources/article/switch-debouncing-electronic-product-designs> . [↑](#footnote-ref-5)
6. ISR = Interrupt Service Routine. [↑](#footnote-ref-6)
7. E.g. <https://randomnerdtutorials.com/esp32-pir-motion-sensor-interrupts-timers/> [↑](#footnote-ref-7)
8. <https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/timer.html> [↑](#footnote-ref-8)
9. <https://github.com/espressif/arduino-esp32/tree/master/libraries/Ticker> [↑](#footnote-ref-9)
10. E.g. <https://techtutorialsx.com/2017/09/30/esp32-arduino-external-interrupts/> [↑](#footnote-ref-10)
11. <https://github.com/Alarm-Siren/arduino-buttons> [↑](#footnote-ref-11)