Switch (de)Bouncing

P.Bernardi

What is switch bounce?

- When you push a button, press a micro switch or flip a toggle switch, two metal parts come together.

- For the user, it might seem that the contact is made instantly.

- That is not quite correct.

Why a switch bounces

- Inside the switch there are moving parts.
- When you push the switch, it initially makes contact with the other metal part, but just in a brief split of a microsecond.
- Then it makes contact a little longer, and then again a little longer.
- At the end the switch is fully closed.
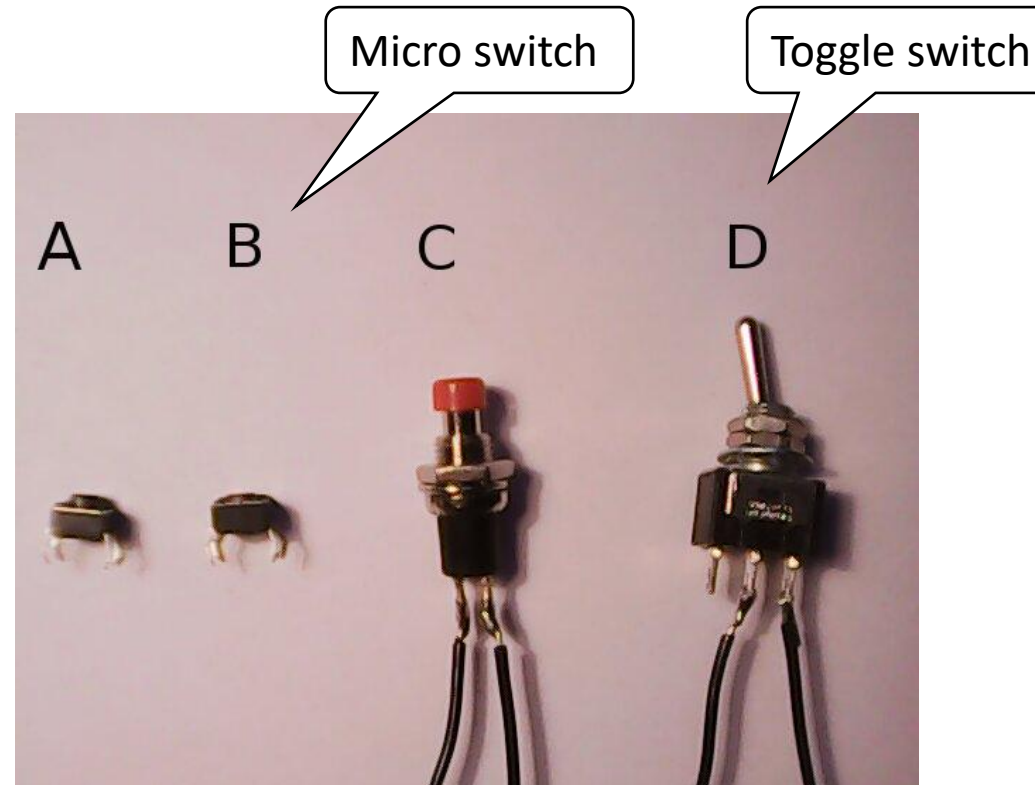- The switch is bouncing between in-contact, and not in-contact.

Bouncing measurement

- "When the switch is closed, the two contacts actually separate and reconnect, typically 10 to 100 times over a period of about 1ms." ["The Art of electronics", Horowitz & Hill, Second edition, pg 506.]
- Usually, the SoC works faster than the bouncing, which results in that the hardware thinks you are pressing the switch several times.
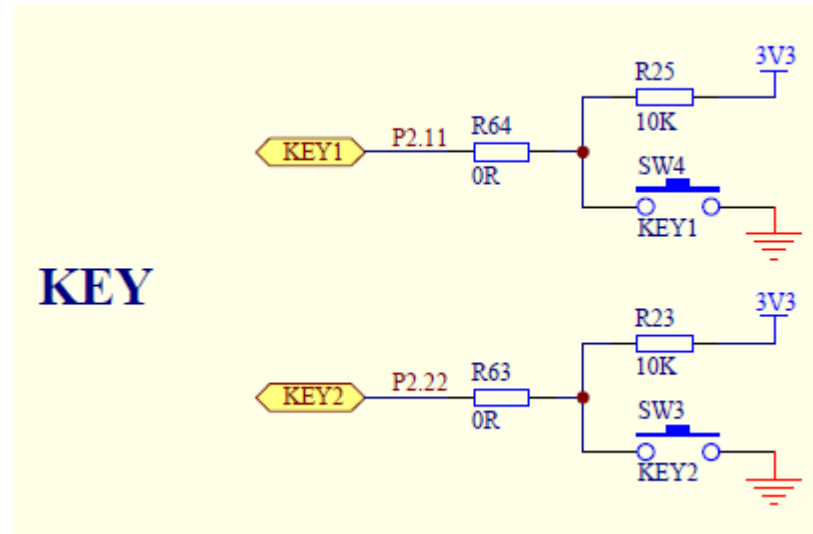
https://www.allaboutcircuits.com/technical-articles/switch-bounce-how-to-deal-with-it/
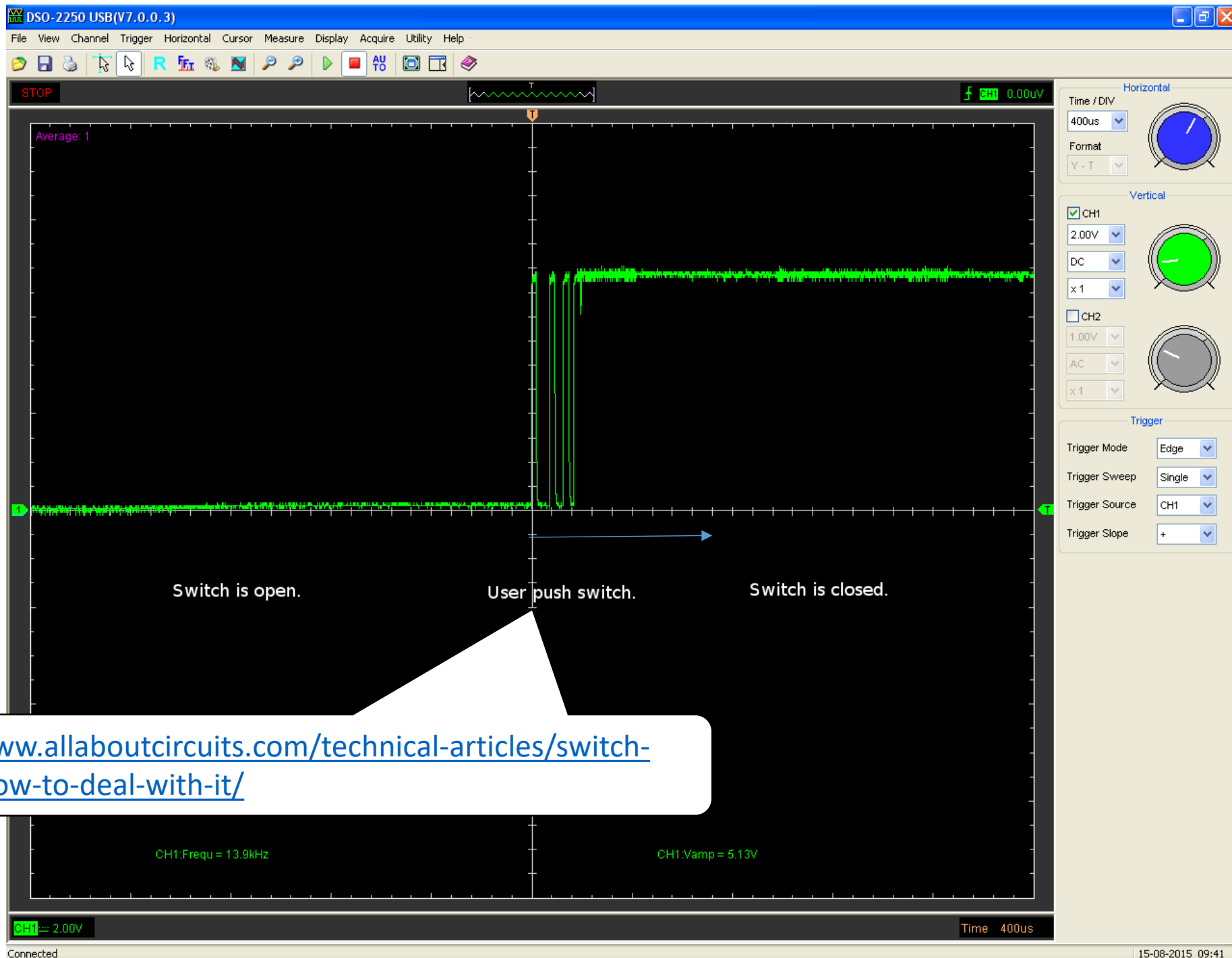
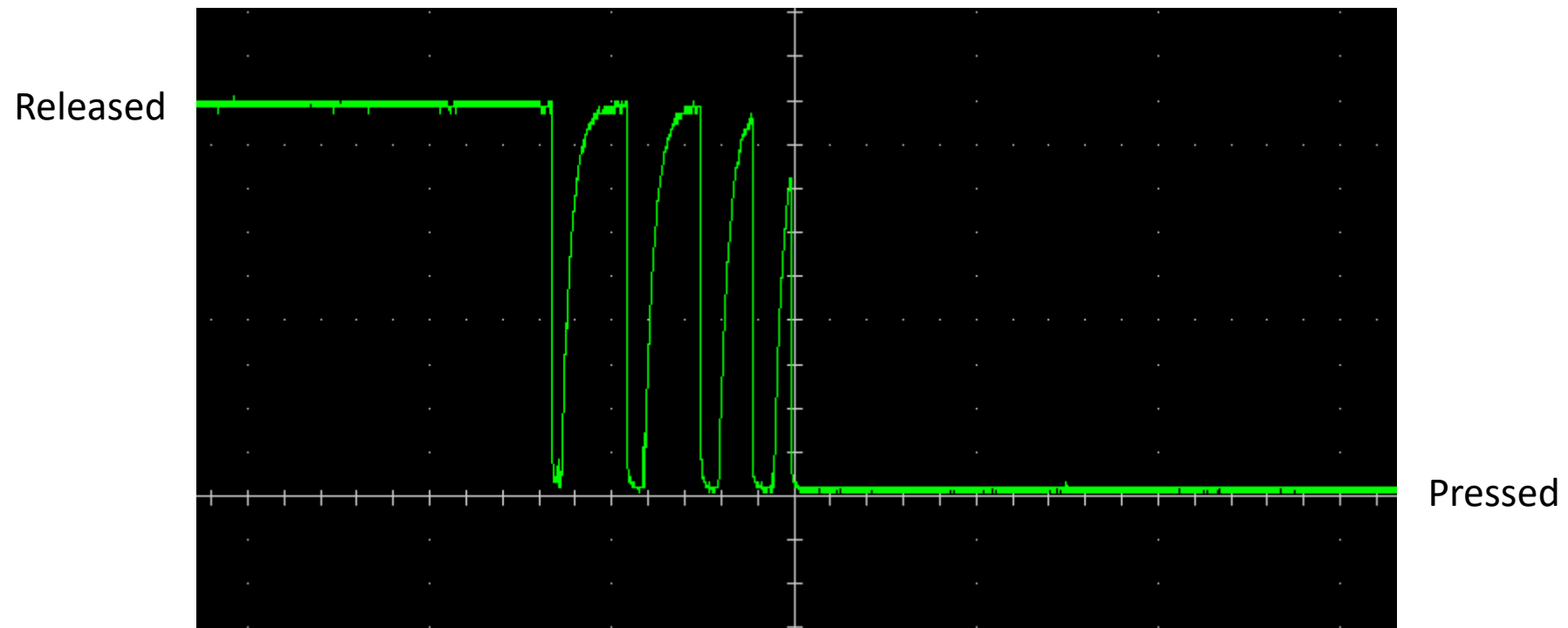- A nice experiment about 4 switch components is shown at the above URL

Key behavior

Button pulse-low pressure (our case)

Button bouncing example

- Also the release of the button may produce bouncing

Released

Pressed
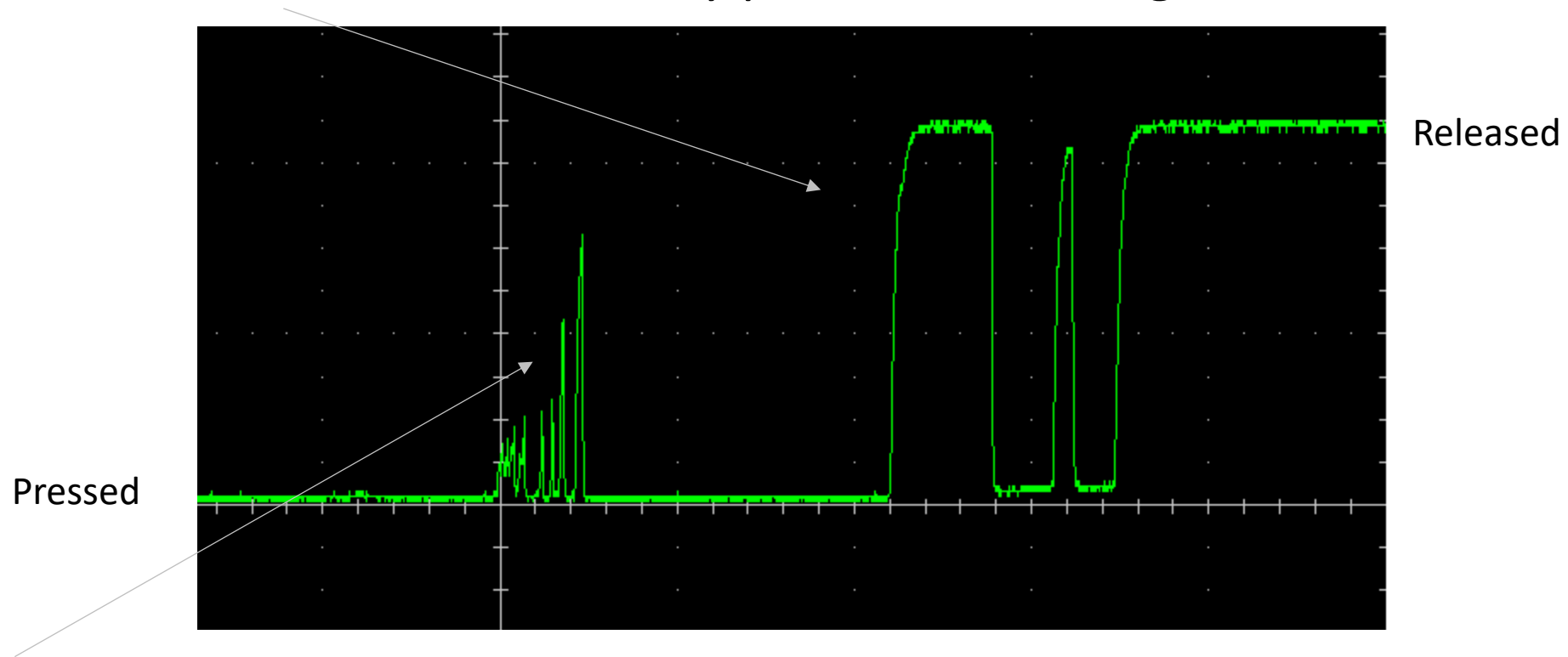
- Spikes may also show up (and be triggered during pressure of the button).

A first conclusion

- When working with microcontrollers, we can deal with switch bouncing in a different way that will save both hardware space and <u>money</u>.

SW implementation

- A common way to deal with switch bouncing is to re-read the value of the pin after 50ms delay from the first bounce.

Assumptions

- We prefer using interrupt mode for buttons (this is power efficient as we can enter power down mode since the button is pressed)

- If interrupt is not available, "polling" the value of the button related pins is the solution
  - A timer can be used to wake up the system at regular time

- Blocking delay implementations are not desirable
  - SW delay by using "for/while/do-while" empty constructs is considered a very "dirty" technique and deprecated.
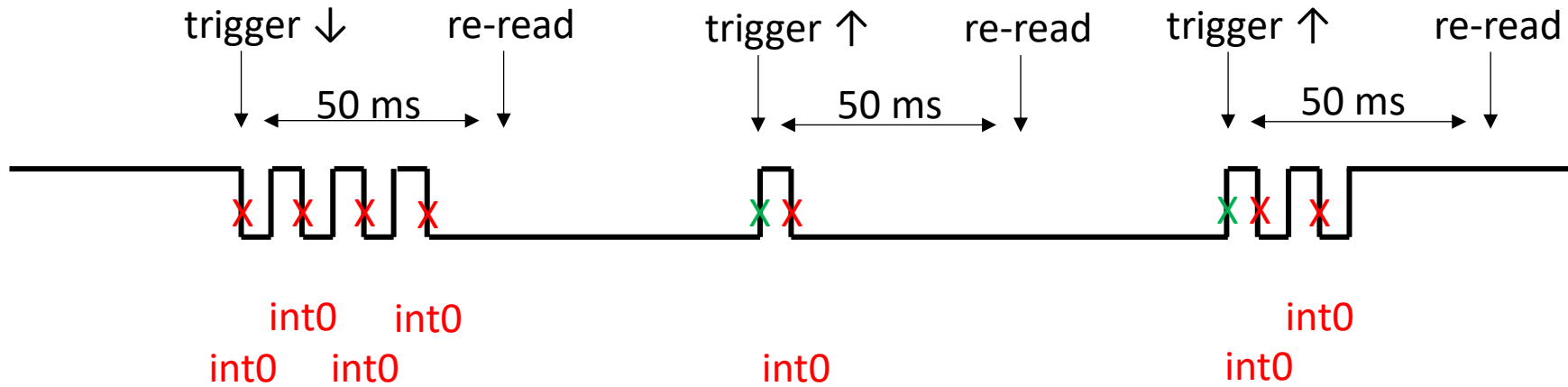
Complications

- If interrupt pin mode selection is selected, the pin may not be directly readable.
  - So, in order to read the button value, it is necessary to disable the interrupt functionality and accept to read an input value.

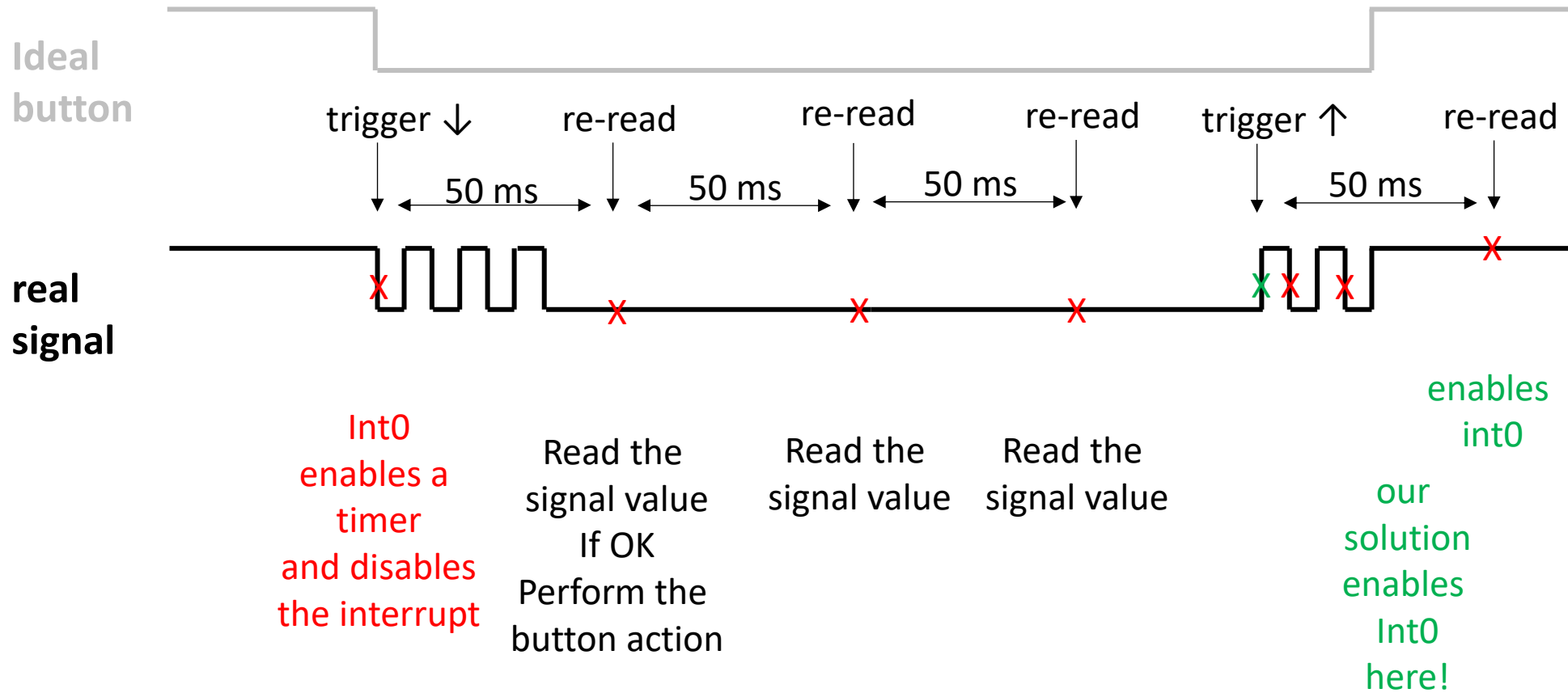# Visual solution

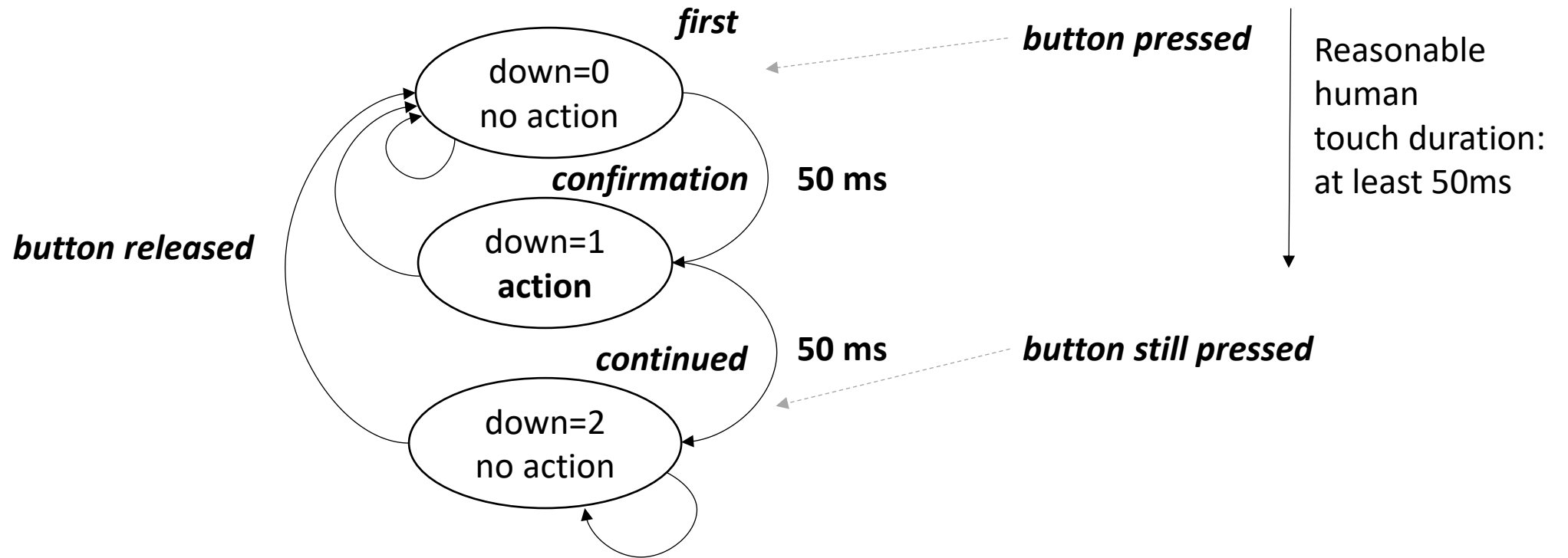# Visual solution - 2

Button de-bounce - finite state machine

Repetitive Interrupt Timer

- You may take advantage of the sample project

sample_BUTTON_DE-BOUNCING

- that includes the Repetitive Interrupt Timer library.

RIT

- Repetitive Interrupt Timer

The Repetitive Interrupt Timer provides a versatile means of generating interrupts at specified time intervals, without using a standard timer.

- Configured to interrupt every 50ms:

Table 41. PCLKSEL1

27:26	PCLK_RIT	Peripheral clock selection for Repetitive Interrupt Timer.	00

Table 42.   Peripheral Clock Selection register bit values

PCLKSEL0 and PCLKSEL1 individual peripheral's clock select options	Function	Reset value
00	PCLK_peripheral = CCLK/4	00
01	PCLK_peripheral = CCLK	
10	PCLK_peripheral = CCLK/2	
11	PCLK_peripheral = CCLK/8, except for CAN1, CAN2, and CAN filtering when "11" selects = CCLK/6.	

lib_RIT.c

```c
uint32_t init_RIT ( uint32_t RITInterval )
{
    LPC_SC->PCLKSEL1  &= ~(3<<26);
    LPC_SC->PCLKSEL1  |=  (1<<26); //RIT Clock = CCLK
…
```

RIT_cnt = 50ms * 100MHz
RIT_cnt = 5.000.000 = 0x4C4B40

Exercise

- Experiment switch bouncing with your board and try to mitigate Key bouncing: they must use the external interrupt functionalities

Advanced -> Joystick: implement a «timer controlled polling strategy» also able to mitigate debouncing

Quite Advanced -> can you manage the pressur of many buttons or the contemporary use of buttons and Joystick?

Super-Advanced -> implement button and joystick debouncing by using the RIT only.