Bern University
of Applied Sciences

Exercise - 03

Edge Detection and Debouncing

Andreas Habegger | Adrian Steiner
BTS3230 | Version 1.0.0 of 17.03.2025

Please be aware that the content is subject to change at any time. For the latest version, please check the website.

In this exercise, you will learn about the side effects of electromechanical components and how to deal with them in software. You will also optimize your solution from the previous exercises and implement a simple latch for the joystick center button.

Objectives

- Electromechanical components
- Debouncing
- Latching an input (joystick center button)

Outcomes

- Understand the electromechanical side effects
- Understand a sequential logic
- Apply a digital debouncing filter
- Polling-based button event detector

Description

Your first LED application has sampled the input state of a “GPIO” pin connected to the USER button “B1” (low active) and set the inverse value on another “GPIO” pin connected to the LED “LD2” (high active). This simple task was repeated in a super loop after initialization. However, you may have updated the output pin regardless of whether the button state changed or not. Since this operation is only needed on a change, a lot of unnecessary operations were performed by the CPU. Now the output “GPIO” connected to the RGB LED should only be set or reset if the input connected to the button changes.

Note

Only the **Nucleo-64** board and **mbed Application Shield** are used in this exercise, but the temperature logger board has no effect on this exercise and can remain connected.

Tasks

Write a simple program to detect a “rising edge” event on the joystick center button and toggle a channel of the RGB LED. Wait, do you know what an “edge” actually is?

Hint

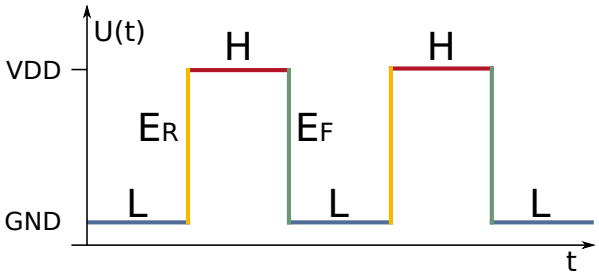
An edge describes a digital signal going from one steady state or level to another. In software we can call such a transition an event of a signal.

- **Rising Edge Event ER**

  Transition from a low state (likely GND) to a high state (likely VDD)

- **Falling Edge Event EF**

  Transition from a high state (likely VDD) to a low state (likely GND)



Implementation

Your first implementation is a pure combinatorial logic, so the output follows the value (state) of the current input. A next evolution of this program should implement a behavior that does a state transaction only on an edge (push and release sequence). So it behaves like a switch instead of a pushbutton. Each push of the button should change the state, toggling the output pin (turning the LED on/off). To implement such a behavior, “sequential logic” is needed. Sequential logic differs from combinatorial logic in that it depends on both the current and previous state(s).

Exercise

1. Configure the joystick center button pin as input and an RBG LED color as output.
2. Compare the current and the previous sampled input value in the program. Toggle the LED state only on a rising edge event.
3. Press and release the joystick center button at least 20 times. (Repeat the sequence pressing both fast and slow.) It is less stable than the Nucleo-64's blue user button. Why is this?

Question

a. Have you noticed any unusual behavior or side effects?

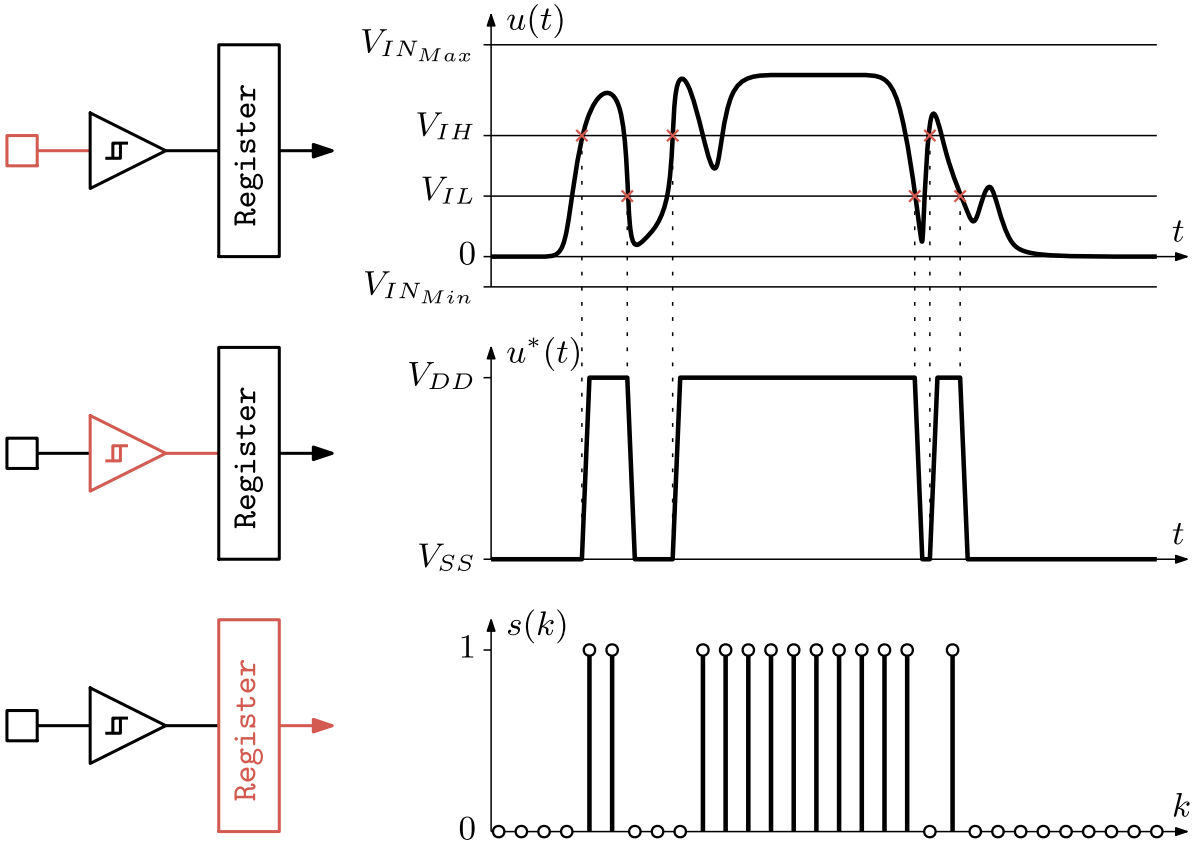
b. If so, what might be the cause?

c. Measure with the saleae logic analyzer the voltage flow of the switch center button in analog and digital with the fastest sample rate. Can you detect/measure this problem with logic analyzer?

Debouncing

While testing your implementation, you may have encountered some unexpected behavior. Pressing the button once may have toggled the LED twice or more. This is a side effect of the button. Because the button is a mechanical device, it tends to oscillate when it changes state.

Hint



An example of an input signal sampling in a GPIO port.

However, the duration of the oscillation is usually quite short. A simple solution to this problem would be to accept a new input level only after a certain period of time. Thus, only if N samples after a transition have the same value, the edge is adopted. Such an implementation, which is a kind of digital filter, is called software debouncing. It is also possible to have a hardware debouncing circuit. See the NULCEO-64 MB1136 schematic for an example of a hardware debouncing circuit (USER button B1). For low-cost electronic designs, hardware debouncing may not be an option, so software debouncing solves the problem of side effects due to mechanical bouncing.

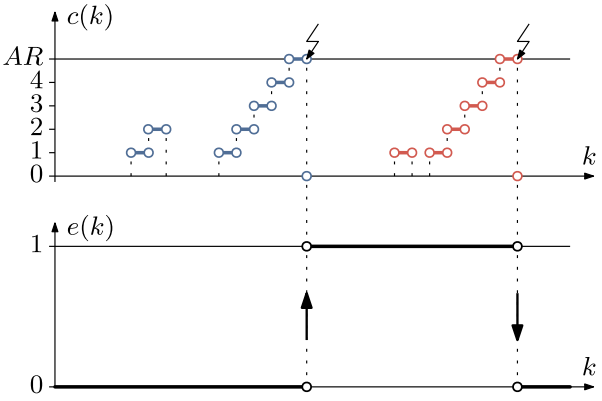
Example Software Debouncing

```
if (sample != currentState){
   ++counter;
} else {
   counter = 0;
}

if (counter == AR){
   counter = 0;
   currentState != currentState;
}

if (currentState != formerState){
  if (currentState) {
      risingEdgeCallback();
  } else {
      fallingEdgeCallback();
  }
}

formerState = currentState;
```



An example of software debouncing after the input signal has been sampled.

Exercise

4. Implement a software debounce functionality on the joystick center button.
5. Test your implementation with different values of AR (auto reload). What is the effect of this change?

See also

For more information about the bouncing effect, read the article Button Debouncing.