De-bouncing circuits

This article deals with a very common problem that occurs in any circuit – more precisely digital circuits – that contains input from a mechanical switch.

When a switch is operated by a human like a push button, or is operated by a machine like limit switches, spikes of low and high voltages will always occur across that switch resulting into a series of High and Low signals that can by interpreted by a digital circuit as more that one pulse instead of one clean pulse or transition from a logic state to another.

The Problem

Figure 1.A shows a basic switch configuration with a pull up resistors, which will output 0V when pressed and 5V through the resistor when released.

![Figure 1.A](image_url)

This switch configuration can be reversed to output 5V when pressed and 0V when released back, but we will study this configuration as it is a standard for all inputs of 8051 microcontrollers and any active low input.

The problem with this configuration, as described briefly before is that, due to the mechanical nature of any switch that may contains spring return action of some kind, there wont be a clean transition from a state to another, but instead there will be a series of high and low states spikes as shown in figure 1.B.
This series of spikes can be interpreted by a microcontroller (or any digital circuit) as if the button was pressed many times. It may even have happened to you before when you connected a switch to a counter of any kind, and notice that on press on the button is sometimes counted as more than one push.

**Figure 1.B** also shows approximately the ranges of voltages where a signal is considered as a HIGH signal (Logic 1) or LOW one (Logic 0).

The ‘unknown’ range can be recognized by some a digital circuit as 1 or 0, but in a completely random manner.

**The Solution**

There are two common solutions to this problem. *Analog solution,* and *digital micro-controller based solution.* Both are commonly used, and sometimes, both are used at the same time to provide a very stable design.

**The analog de-bouncing circuit**

The analog solution relies principally on a capacitor, which plays the role of resisting the voltage changes on the output. In other words, this will prevent the output to change too fast which will prevent High and Low pulses to appear on the output. A de-bouncing circuit can be tuned exactly to your requirements, it can be adjusted to choose which pulse is rejected and which one is accepted.
Figure 2.A shows how to assemble a switch de-bouncing circuit. The values of the resistor R₁ and the capacitor C₁ will determine the response speed of the switch. The more you increase R₁ and/or C₁, the more your circuit becomes immune to errors, but the more time it takes to react and give adequate output. It’s up to you to chose the values of R₁ and C₁, using trial and error (or calculations) until you get the suitable response for your application. Good starting values for a general purpose debouncing circuit would be $R₁ = 10\, \text{K}$ and $C₁ = 100\, \text{nF}$ ceramic capacitor.

It is clear on figure 2.B, that the addition of a capacitor made the voltages rise smooth and clean as compared to the spikes in figure 1.B. But even though the addition of a capacitor, you can notice that, while rising from 0 to 5 v, the voltage passes through a range where the output is unknown (shaded in yellow in the figure), which will again cause some error pulses to appear on the output.

This last minor problem can be fixed by adding a schmitt trigger. Shortly, a schmitt trigger will keep its outputs unchanged during the passage through the ‘unknown’ zone, until the input have safely reached above or below some threshold values, which define the High and Low logic states. In other words, a schmitt trigger hooked to the output will eliminate the error pulses generated when the button was released and will give straight output, clean of any errors.
Figure 2.C shows the symbol of a schmitt trigger buffer. A very similar configuration is used as a power on reset circuit for many microcontroller, to provide them with a ‘reset’ pulse some milliseconds after the device have been turned ON.

The software based de-bouncing

The approach for a software based de-bouncing circuit is totally different than what was discussed before. The idea here is not to prevent voltage spikes to occur, but rather to record them and analyze them in real-time using simple software routines.

You should now be familiar with the digital pulses in the figure 3.A. The timing $T_1$, $T_2$ and $T_3$ corresponds to the low (logic =0) pulses being sent to the microcontroller from the switch. $T_1$ and $T_2$ are the length of the reading we want to get rid off, or more scientifically, that we want to filter. $T_3$ is a valid reading that we want to take in account. It is clear that the difference between those three periods is their length, and that is how the microcontroller will be able to differentiate between valid and un-valid pulses.

There are many ways to implement digital de-bouncing or filtering using a microcontroller, one simple method is to make a counter count up as long as the signal is Low, and reset this counter when the signal is High. If the counter reach a certain fixed value, which should be 1 or 2 times bigger than $T_1$ or $T_2$ (noise pulses), this means that the current pulse is a valid pulse (corresponds to $T_3$ in our example)
This algorithm can be implemented for 8051 microcontroller using C code as in the following example:
Note that in this example, the signal to be filtered is connected to the first pin of Port 1 (P1_0)

Example C Code for 8051 micro-controllers

```c
#include <REGX52.h>
#include <math.h>

unsigned char counter; //Variable used to count
unsigned char T_valid; //Variable used as the minimum duration of a valid pulse

void main()
{
    P1 = 255; // Initialize port 1 as input port
    T_valid = 100; //Arbitrary number from 0 to 255 where the pulse if validated

    while(1){ //infinite loop
        if (counter < 255) //prevent the counter to roll back to 0
            counter++;

        if (P1_0 == 1){
            counter = 0; //reset the counter back to 0
        }

        if (counter > T_valid){
            //....
            // Code to be executed when a valid pulse is detected.
            //....
        }

        //....
        //Rest of you program goes here.
        //....
    }
}
```
Application of de-bouncing circuits

The most famous application for a de-bouncing circuit is for cleaning the output of limit switch or push buttons. However, there are other situations where similar approaches may be used.

For example, shaft encoders, that are essential in most robots are sometimes subjected to vibrations that can cause more pulses to be generated by the shaft encoder than the actual correct number of pulses corresponding to the movement of the shaft of the motor.

Also linear position encoders are prone to the same problem.

As you may have noticed before, the analog de-bouncing circuit, can be simply used as to filter the noise from a signal. but be careful, while using too big values for R and C will result in a very immune system, it will also cause very slow response, and valid pulses may as well be rejected with other noise. You have to tune your circuit with the suitable values of C and R to fit to your requirement.

Source: http://www.ikalogic.com/de-bouncing-circuits/