Switches Bounce!

A big problem with using mechanical switches is that they physically bounce and they are subject to picking up noise form the environment. Just can't seem to shield switches from environmental noise and t is very expensive to make a switch that is bounce-less.

Take the switch in Figure 1. A normally open , single pole, single throw pushbutton switch (NO SPST) that uses a pull-up resister so it's compatible with standard TTL voltage levels. If the pushbutton switch is pressed, the output becomes 0V (logic 0) and if the pushbutton is not pressed, the output becomes 5.0V (logic 1). The value of the pull-up resister is system dependent, but for normal system loading (TTL) it ranges in value from 1K ? to 27 K ? .

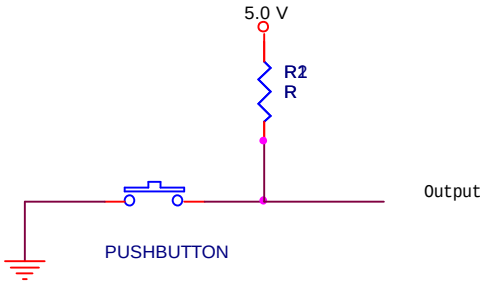

Figure 1

When a pushbutton or any switch's position is changed noise is generated. Some noise (contact) occurs because the switch contact is metal and it has elasticity. When the switch is moved to a new position it strikes a metal contact and physically bounces a few times. We call this contact bounce. Contact bounce can be eliminated (or significantly reduced) by using mercury wetted contacts or a solid state device such as a Hall-Effect device. Problem is that these are expensive. Figure 2 illustrates the electrical signal produced by contact bounce and other noise

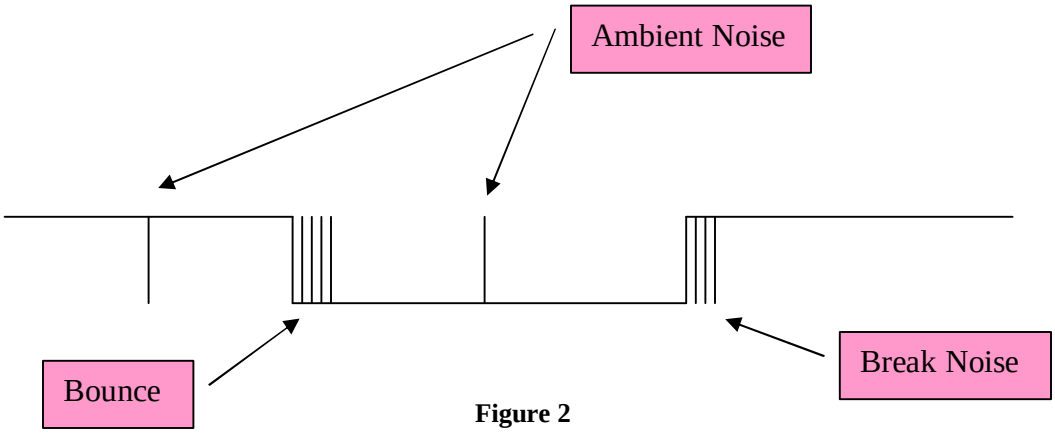

Figure 2

Removing bounces with a circuit.

These noise problems can be easily removed by using a few inverters as illustrated in Figure 3.

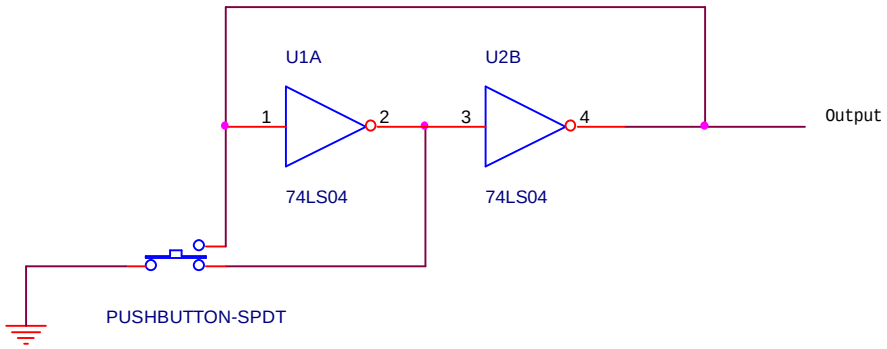

Figure 3

As you can see, the only problem with this method is that we need a SPDT pushbutton switch, which costs more than a SPST pushbutton switch. Another method uses a SPST switch, but still requires a Schmidt Trigger inverter and a resister as illustrated in Figure 4.

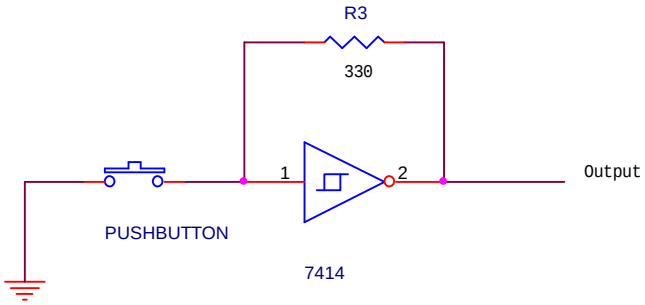

Figure 4

Both circuits function and remove all noise and bounces, but they cost money. The ideal circuit is the one illustrated in Figure 1 — no circuit besides the pull-up resister.

Removing bounces with software.

The key behind removing bounces with software is a time delay. We can check the switch, determine when it changes and start a time delay at the time we first detect a change. If we check again after the time delay has elapsed and find that the switch has not changed we have detected noise or bouncing. If the state of the switch has changed then we have discovered either a make or a break. Again the key to this problem is the time delay. The question is how long a time delay? I have researched switch bounce times and have discovered that the maximum bounce time seems to be 8 ms or less for most switches. Thus is we use a time delay of from 10 to 15 ms we should be able to de-bounce a switch.

So to de-bounce a switch with software we need to (1) check the state of the switch, (2) wait for 10 ms, and (3) check the state of the switch again. If the states are different in Step 1 and 3 we have found that the switch has changed states (not noise or bounces) and we can use the state detected in step 3 for the output of the switch.

In practice de-bouncing a switch is done in two parts. One part waits for the user to press the pushbutton and the other waits for the user to release the pushbutton. This can be accomplished as indicated in the program snippet illustrated in Example 1.

Example 1

```
.REPEAT
      .REPEAT
            IN    AL,BUTTON    ;check button
            SHR   AL,1
      .UNTIL  CARRY?
      CALL  TIME_DELAY         ;wait 10 ms
      IN    AL,BUTTON          ;check button again
      SHR   AL,1
.UNTIL CARRY?
```

In example 1 we assume that that pushbutton is connected to input port BUTTON at the least significant bit position. To check the pushbutton we input it to AL and shift AL right 1 place so the rightmost bit moves into the carry flag. This allows us to check the state of the pushbutton by looking at the carry flag. In Example 1 we check the pushbutton for a logic 1 (CARRY), then next we execute the time delay, finally we check the button again and check for a logic 1. If it passes through this software we are guaranteed that the switch is at its logic 1 position. You could call this section wait for a release. To modify the software so it waits for the pushbutton to be pressed, change the CARRY? to a !CARRY? in both places.

A procedure (see Example 2), called KEY, is now written that completely de-bounces the pushbutton. If KEY is called in a program the system will stay in KEY until the pushbutton is pressed. The pushbuttin is processed after the return to the calling program.

Example 2

```
KEY   PROC  NEAR

      .REPEAT                        ;wait for release
            .REPEAT
                  IN    AL,BUTTON    ;check button
                  SHR   AL,1
            .UNTIL  CARRY?
            CALL  TIME_DELAY         ;wait 10 ms
            IN    AL,BUTTON          ;check button again
            SHR   AL,1
      .UNTIL CARRY?

      .REPEAT                        ;wait for press
            .REPEAT
                  IN    AL,BUTTON    ;check button
                  SHR   AL,1
            .UNTIL  !CARRY?
            CALL  TIME_DELAY         ;wait 10 ms
            IN    AL,BUTTON          ;check button again
            SHR   AL,1
      .UNTIL !CARRY?

      RET

KEY   ENDP
```