EDITED BY BRAD THOMPSON
AND FRAN GRANVILLE

# designideas

READERS SOLVE DESIGN PROBLEMS

## Contact-debouncing algorithm emulates Schmitt trigger

Elio Mazzocca, Technical Consultant, Adelaide, South Australia

Among other interface problems, contact bounce complicates the connection of mechanical contacts or any noisy digital input signal to a microcontroller. Although designers have proposed a variety of hardware and software approaches that address the problems that contact bounce poses, no one has yet claimed a definitive and predictably stable approach. (For a sampling of approaches, see **references 1** through **10**.) The usual hardware approach to eliminating contact bounce comprises an RC filter followed by a Schmitt trigger (**Figure 1**). You can extend the filter's effectiveness simply by increasing the RC time constant at the expense of increased response time.

Software-debouncing methods usually include 1-bit processing, which involves twice reading the contact's input state with a fixed delay between the two readings. You can also implement a state machine or launch an input signal through a shift register and wait for three or four register-output states that haven't changed. The low efficacy of 1-bit processing approaches stems from designers' erroneous assumptions that seemingly simple debouncing tasks can tolerate equally simple software. However, a detailed study of many types of contacts reveals a range of complex and sometimes unexpected behaviors. This Design Idea documents a more comprehensive method that can easily handle all mechanical contact interfacing to microcomputers.

The debouncing method applies full 8-bit-processing and digital-filtering techniques to digital inputs. Using as few as 20 assembly-language instructions that execute in 19 machine cycles on an ATmega8 microcontroller, the method produces a robust debouncing action (see **Listing 1** at the Web version of this Design Idea at www.edn.com/edn050707di1).

The software closely simulates the hardware circuit in **Figure 1** by using a first-order, recursive, digital lowpass filter followed by a software Schmitt trigger. In contrast to 1-bit software debouncers that generally do not apply processing to inputs, this debouncing algorithm is effective because it "remembers" past input transitions and assigns a "weight" to each transition depending on how long ago it occurred. Furthermore, you can alter the filter's settings on the fly to meet changing conditions by modifying its thresholds and hence its execution time, or time constant, from the main program. The basic recursion algorithm comprises present output value=(¼)×input value+(¾)×previous output value, or, $Y_{NEW}=(1/4)\times X_{NEW}+(3/4)\times Y_{OLD}$.

To avoid register overflow and instability, the value of $Y_{OLD}$ and $X_{NEW}$ must be less than 1, which for an 8-bit microprocessor translates to values of less than 256 for $X_{NEW}$ and $Y_{OLD}$. Consequently, the input $(1/4\times X_{NEW})$ to the filter is either 0 or 63. You then apply the output value, $Y_{NEW}$, to the software Schmitt trigger. The trigger uses the following algorithm: If $Y_{NEW}$>hi, and flag=0, then flag=1, and out=1. If ($Y_{NEW}$<lo, and flag=1, then flag=0, and out=0.

Hardware Schmitt triggers typically have fixed thresholds of one-third and two-thirds of the power-supply voltage. However, the software allows widening these thresholds and thus increasing the filter's time constant. In operation, a timer-interrupt routine should execute the debouncing program every 4 to 5 msec. Becausee one time constant equals the period of one interrupt, using thresholds of 15 and 240 causes the routine's output to "trigger" after 11 interrupts, or 44 to 55 msec, which ade-

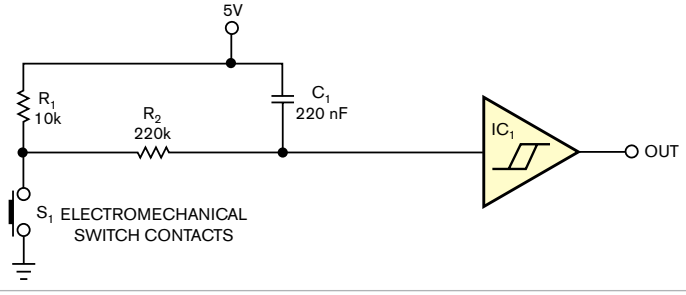


Figure 1 A basic switch-contact debouncer consists of an RC network followed by a Schmitt-trigger circuit.

### DIs Inside

quately processes most switches' contact bounce.

You can easily modify the main filter coefficient to provide different filtering-time constants. For particularly troublesome contact bounce, you can use the following recursion formula, which requires 16 time constants to trigger the software Schmitt routine. $Y_{NEW}=(1/16)\times X_{NEW}+(15/16)\times Y_{OLD}$. You can implement this algorithm with only eight assembly-language instructions, whereas the Schmitt-trigger routine requires 12 instructions. When you combine both of these routines, the software Schmitt trigger updates bit 0 of a register, which the main program loop should continuously check to ascertain the contact's status. As an alternative, you can activate a software interrupt to signal a contact's status change. To do so in the AVR architecture, you write to that port bit that functions as an external interrupt input.

Always avoid connection of mechanical contacts to interrupt inputs unless the contacts undergo hardware debouncing. Otherwise, the contacts may bounce dozens of times, unnecessarily consuming processor-machine cycles. The software routine reads the inputs only every 4 msec and thus imposes additional filtering on the inputs. Simulation and practical testing have confirmed that the debouncing algorithm behaves as expected, producing clean output transitions when enduring noisy contacts. When you program the assembly-language source code accompanying this Design Idea into an Atmel Atmega8, the code turns on an output LED connected to Port_B bit 0 when Port_D bit 0 of the microcontroller goes to ground.

A simulated input waveform (pind0) and its corresponding output log file (portb0.log, both available at the Web version of this Design Idea at www.edn.com/050707di1,) illustrate the filter's excellent debouncing capabilities. Beginning with a key closure at 10 msec, the stimulus loads into the AVR Studio integrated development environment. After multiple input transitions, the output-log file shows a single output transition occurring at 55.333 msec. The software effectively filters out the three input pulses starting at 56.1 msec (**figures 2** and **3**).EDN

## REFERENCES


1. Hills, Paul, "A Mercury switch filter," http://homepages.which.net/~paul.hills/Circuits/MercurySwitchFilter/MercurySwitchFilter.html, August 2001.
2. Baker, Bonnie, "The debounce debacle," *EDN*, Oct 28, 2004, pg 26, www.edn.com/article/CA472833.
3. Ganssle, Jack, "My favorite hard-


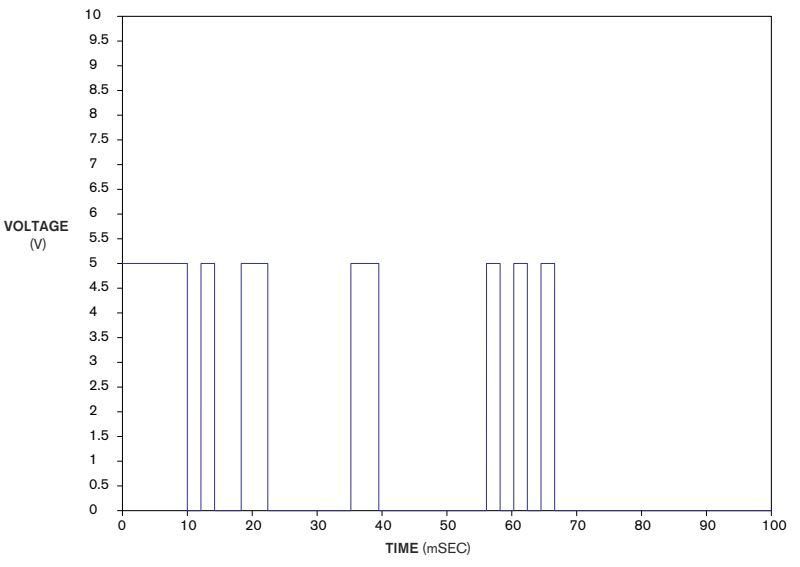


Figure 2 Switch contacts first close at 10 msec and then bounce erratically for more than 50 msec before reaching a stable closed condition.

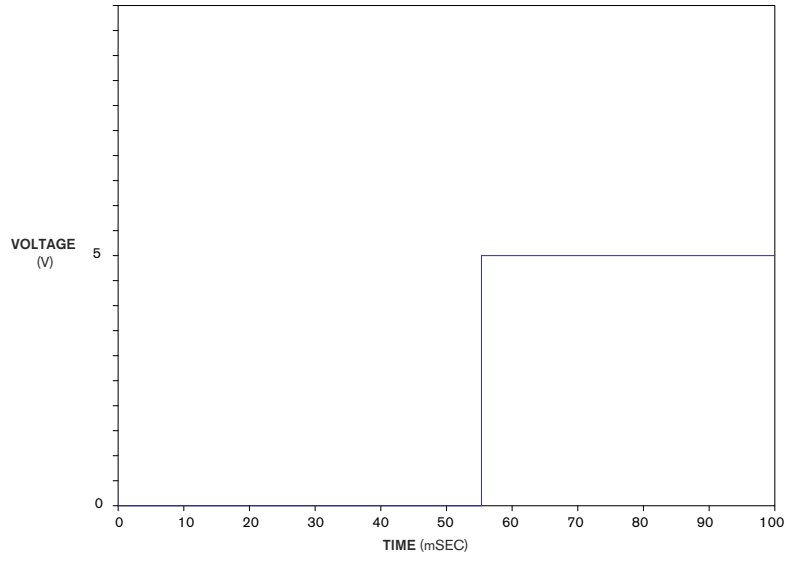


Figure 3 The debouncing-software routine signals that the contacts have closed only after bouncing ceases.

ware debouncers," *Embedded Systems Programming*, June 16, 2004, www.embedded.com/showArticle.jhtml?articleID=22100235.

4 Ganssle, Jack, "The secret life of switches," *Embedded Systems Programming*, March 18, 2004, http://embedded.com/showArticle.jhtml?articleID=18400810.

5 Smewing, Alan, "Icc-avr keypad-debounce code," http://dragonsgate.net/pipermail/icc-avr/2004-March/003376.html, March 4, 2004.

6 "Switch debouncing," www.miteduc.freeserve.co.uk/Design/debounce.htm.

7 Matic, Nebojsa, *The PIC Microcontroller*, www.mikroelektronika.co.yu/english/product/books/PICbook/7_03chapter.htm.

8 Ganssle, Jack, "Smoothing digital inputs," *Embedded Systems Programming*, October 1992, www.ganssle.com/articles/adbounce.htm.

9 Ganssle, Jack, "My favorite software debouncers," *Embedded Systems Programming*, June 16, 2004, www.embedded.com/shared/printableArticle.jhtml?articleID=22100235.

10 "Switch bounce and other dirty little secrets," Maxim, www.maxim-ic.com/an287, September 2000.