INTRODUCTION

Real physical switches bounce when opened and when closed. Figure 1 shows an example timing diagram for a digital switch circuit with a pull-down resistor. While designing hardware circuitry to debounce a switch electrically connected to a microprocessor can be costly, writing software to create a delay for a duration at least equivalent to that of any bouncing provides an easy (and almost always free) replacement.

NOTE: In this document, the term microprocessor is interchangeable with processor, central processing unit (CPU), and microcontroller.

![Figure 1: Digital switch bouncing diagram](thanks to Epec's Blog)

There are several ways to debounce a switch connected to a microprocessor with software, and only three methods are described in this document. Each of these methods has benefits and drawbacks. The first technique, debouncing with a software delay, is sometimes appropriate for simple applications, where asynchronous response to a switch is not desired, though in general, should be avoided due to a strong restriction on program flow. The second, using a timer/counter (TC) module and a hardware flag (where a hardware flag is defined here to mean either an interrupt flag native to the microprocessor, or a signal routed from an external device to an I/O port on the processor), is a viable approach for programs needing either asynchronous or synchronous response to a switch, although it should also generally be avoided due to a similar restriction on program flow. The third and final method described in this document, using a timer/counter along with a TC interrupt, is by far the most desirable, since it allows for asynchronous and synchronous response to a switch, and has little to no restriction on program flow; however, this method is also generally the most complex to implement.

Prior to continuing, it is important to note that before any switch debouncing technique is implemented, multiple measurements should first be taken to determine a likely upper-bound for the duration of the switch bouncing. Also, for any switch debouncing technique, after an appropriate delay is determined to have been made, the microprocessor should check the pin level of the relevant switch before performing any further operations; this is to remove ambiguity and verify that the switch is still in either the expected high or low state, with the assumption that all switch bouncing has ceased.

USING A SOFTWARE DELAY

The first method described here involves tasking the microprocessor with a calculated number of (meaningless) instructions, otherwise known as a software delay, to keep the processor “busy” for a duration of time at least equivalent to that of the expected switch bouncing.

Assume that, for example, it is determined that bouncing for a given switch always occurs for less than 1 ms. Then, to handle this bouncing, you could simply create a software delay of 1 ms. After the desired delay, the program would then check the appropriate pin level and perform any necessary function(s).

Generally, although software delays have the potential to be very precise, they prevent a microprocessor from executing other instructions, and ultimately cause CPU time to be wasted. Software delays are also extremely non-modular, as they cannot easily be used at other processor clock speeds, and if designed in an assembly language, cannot be directly ported to most other processors.

NOTE: Although the above is sometimes plausible when continuously polling (i.e., continuously reading) a switch in software, where responses to the switch are synchronous, software delays are abhorrent in conjunction with an interrupt service routine (ISR), i.e., within an asynchronous environment. Designing a program to explicitly delay within an ISR is a ghastly practice because this prevents a microprocessor from being able to service other interrupts during the delay, assuming that you have not allowed ISRs to nest/preempt each other, which is also normally discouraged. In general, interrupt service routines should be as short as possible.

The following two debouncing methods utilize hardware timer/counters to create an appropriate delay. In the first of these methods, it is assumed that a hardware flag used to identify when you have not allowed ISRs to nest/preempt each other, which is also normally discouraged. In general, interrupt service routines should be as short as possible.

NOTE: If debouncing a tactile switch, it is probable that upon releasing the switch, an unintended external interrupt will still occur. This is the only unintended interrupt that should occur.
USING A HARDWARE FLAG

The first of these next two methods enables a timer/counter within an external interrupt (for the purpose of creating a debounce delay) and polls an appropriate hardware flag within the main routine of the program, to determine when the relevant timer/counter has overflowed or when a compare match has occurred. A flowchart of this method is given in Figure 2, and a description is provided below.

Before or after a timer/counter has been configured within the main routine of a program, an external interrupt for the appropriate pin should be configured, but not enabled, and then interrupts should be globally configured. Following this, wherever it is desired to use the relevant switch within the main program, the external interrupt should be enabled, and the appropriate timer/counter hardware flag should be polled. (It should be recognized that this flag will be asserted after a configured debounce delay occurs.) Upon the to-be-debounced switch either opening or closing, the relevant external interrupt should trigger. Within the respective interrupt service routine, the external interrupt should be disabled (to prevent unnecessary interrupts), the chosen timer/counter module should be enabled, and if necessary, the flag for the external interrupt should be reset. Following these, the program should return to the main routine.

When the relevant TC module has counted for the designated length of time, the pertinent hardware flag should be automatically asserted, and the polling loop should terminate. Thereafter, four things should occur: the timer/counter should be turned off and have its count value reset, the hardware flag should be reset to its initial value, and the pin level of the switch should be checked.

Although this method is sensible because it does not require that another interrupt be configured, it is very non-modular; this is because the hardware flag must continuously be checked within the main program, wherever it is expected that the to-be-debounced switch will be used. In some situations, this is adequate; however, this method is generally not allowed in our course.

NOTE: There are other situations, e.g., not just when debouncing a switch, that a hardware flag should be polled. In these same contexts, a user-defined bit within a register or memory, otherwise known as a software flag, might also be sufficient.

USING A TC INTERRUPT

The final method described here involves configuring a timer/counter interrupt to handle all operations meant to occur upon a to-be-debounced switch opening or closing (i.e., upon either a rising or falling edge of the switch pin level). A flowchart of this method is given in Figure 3, and a description is provided below.

As in the previous method, a program should initially configure a timer/counter module as well as an external interrupt for a relevant pin (enabling this interrupt whenever response to the switch is desired), before configuring interrupts globally.

Figure 2: Flowchart for method of switch debouncing using a hardware flag
However, unlike the previous method, the main routine need not synchronously poll for a hardware flag, and may proceed however it is desired.

When the pertinent external interrupt is triggered and the respective ISR is executed, the external interrupt should be disabled, the appropriate TC should be enabled, and if necessary, the flag for the external interrupt should be reset. Additionally, a timer overflow (or compare match) interrupt within the TC module should be configured and enabled, before terminating the external interrupt handler.

After the TC has counted the designated length of time, the overflow (or compare match) interrupt should trigger. Within the respective interrupt, the timer and its interrupt should be disabled (also resetting the TC count value and de-asserting the respective interrupt flag if necessary), and the external interrupt should be re-enabled; additionally, the external interrupt flag should be reset to its default state, and the appropriate switch pin level should be checked, before performing any necessary function(s).

Resetting the external interrupt flag must be done here, as this is only place where you should be able to guarantee that the switch has stopped bouncing.

Unlike the previous debounce method, this method is modular, since no other thread of execution within the program is affected by the switch. The only notable downside of this approach is that it is the most complex to implement of the three mentioned in this document; however, this technique is required in our course whenever asynchronous responses to a switch are desired.

![Flowchart for method of switch debouncing using a timer/counter interrupt](image-url)

**Figure 3:** Flowchart for method of switch debouncing using a timer/counter interrupt