Polling vs. Interrupt Driven Switch Service Routines

Haste or Waste?

By Michael H. Pelkey, Founder & CEO, LogiSwitch LLC

A Guide to the advantages and disadvantages of the polled method and the interrupt method for various switch applications.

There are two significant matters to be considered when using the standard mechanical switch in embedded microcomputer applications:

1. Dealing with Switch Contact Bounce.
2. Decision to Handle Switch Input by Interrupt Method or Polled Method.

Contact bounce is issue number one, since it must be dealt with in nearly all applications regardless of the interface method employed. It is essential to assume that the mechanical electric switch will bounce. Bouncing feeds a multiplicity of sequential high and low transitions to the host computer for a brief amount of time when a mechanical switch is actuated and released. If the bounce transitions are not removed by some method from the raw switch output, they are likely to cause problems for the switch interface in most applications.

There are numerous ways to eliminate switch bounce. Debouncing may be implemented by adding physical hardware components to your circuit, by writing debounce code in your

application, or by adding a switch debouncer IC to your circuit. This document focuses on the use of the LogiSwitch debouncer IC to eliminate switch bounce.

The decision to use the polling method or the interrupt method depends entirely on the priority of your switch service requirements with respect to your program. Every application is different, but the need for haste or waste for a particular application is usually obvious to the designer. Either method typically requires debouncing.

The polling method is also only effective if the incoming signal is debounced, but polling is a shameful waste of time under all circumstances and should be avoided like the plague under most circumstances.

Polling requires the undivided attention of the host processor to properly observe when you enter and leave one switch service cycle, so you can tell the difference when one cycle ends and another one begins. For legacy polling the host processor must abandon the task of executing its program while sitting in a loop waiting for someone to take his big fat finger off a switch. LogiSwitch has a better way: (See “The LogiSwitch Handshake for Poll-Free “Polled” Routines below”). Please note that the amount of time a switch is held active is almost never relevant to the program. The LogiSwitch handshake controlled by the program, however, is always perfectly relevant. As shown below the LogiSwitch handshake protocol provides the designer with a means to terminate the switch cycle programmatically rather than waiting for release of the switch by the user.

The LogiSwitch Handshake for Poll-Free “Polled” Routines.

All LogiSwitch IC products incorporate a two-way communication provision to make the most of the switch interface for your choice between the polled or interrupt switch interface method. The handshake uses a single wired-or host pin to communicate over a request/acknowledge link between the LogiSwitch device and the host processor.

The request/acknowledge protocol substitutes a simple handshake communication between the host computer and the LS100 Series device for the no-poll polling sequence is as follows:

- The LogiSwitch device receives an active-low signal from the switch.
- The signal is debounced by the LogiSwitch device.
- The LogiSwitch device sends a high-level output to the host processor indicating a service request from the switch in a single transition.
- The host pin, configured as an input, fields the request.
- The host reconfigures its pin to the output mode and outputs a low 5 us ACK pulse back to the LS100 device, then reconfigures the pin back to input mode.
- The LS100 series device receives the ACK and answers by latching out a low level on the common line to terminate the service request. The low level remains on the line until another switch service request is initiated to repeat the cycle.

Note that the host processor is not required to respond. If the cycle continues without an ACK from the host, it will terminate as normal when the switch is released. Also note that once the user's program responds to the REQ by the LogiSwitch device with an ACK, the cycle is complete, and no further attention is required. The host processor can go about its business and never look back. The LogiSwitch device is still working in the background to debounce the release interval of the switch cycle, but that is invisible to the program. The LS100 series device will not allow a new switch cycle to begin until the present cycle has completed and the debounce period has ended. In a typical manual switch cycle, the switch may be in the actuated state for as long as a full second. A modest 8-bit embedded processor of a moderate clock frequency of 16MHz would execute as many as four million instructions in that amount of time.

For the interrupt method, switch bounce can cause multiple interrupts and should always be eliminated with a hardware solution for proper results:

The LogiSwitch Device for Edge-Triggered Interrupt Processing

The LogiSwitch line of debouncer ICs is ideal for normal interrupt-driven switch processing applications. When used with embedded processor input pins that feature an Interrupt-on-change capability, the LS100 series device outputs a crisp, clean single-transition edge to call for a single interrupt when actuated. In all interrupt applications, the LS100 series device will

handle the interrupt-on-change switch processing perfectly. Note that for higher frequency (1 ms - 10 ms) timer interrupts, the LogiSwitch device is the recommended choice to assure that a single interrupt per switch cycle will register. The handshake may be used to assure that the switch service request will not repeat when the interrupts are re-enabled after the interrupt executes.

Poll, No-Poll, or Interrupt?

Pushbuttons, relays, limit switches, etc., represent the very simplest of computer peripherals, typically signaling the on or off state to your program. They are used in many ways for many different purposes and each application requires its own level of attention. Below is a rough outline of a variety of switch applications and suggested methods of implementation:

Emergency Stop.

The E-Stop must always be implemented at the highest possible priority. It should always be implemented as an interrupt subroutine (ISR). If non-maskable interrupts (NMI) are available this is the place for it. The responsibility of the E-Stop is to immediately stop any motors, slides or machine members that may cause harm to the operator without regard as to whether it harms the machine or not. The E-Stop interrupt must always be enabled, and execution must be completely contained in the ISR itself with all other interrupts disabled and its highest priority task must be the shut-down of all moving parts. Note that the E-Stop is not truly a safety feature. It is commonly implemented as a response to a harmful action that has already taken place such as a jammed machine or operational error.

- Use the Interrupt method, preferably a non-maskable interrupt (NMI) if one is available. Contain all the code for the emergency stop in the ISR to the end without deviation. If the E-Stop ISR must be a shared function in a common interrupt, it must be at the top of the list.

Time-Critical Functions.

The operator may be watching a machine member that requires an instantaneous manual pushbutton response, or something similar. While not at the critical level of importance of the E-Stop routine, a function like this does require a snappy response by the software. The implementation must be weighed against other operations which are required to be running simultaneously. Coding as an interrupt is the best choice in this case. The question is whether to code the entire routine beginning to end in the ISR or setting a flag in the ISR and returning to the main loop. This of course depends on the priority of the other routines. If the time-critical portion of the task can be completed in the ISR and the less time-critical portion can be completed in the main loop, that is usually the best way to go. Interrupts should (almost) always be short and sweet, except for the E-Stop ISR.

- Initiate the time-critical routine with an interrupt. If the time spent in the ISR will be short, contain the entire routine in the ISR. Otherwise execute the time-critical portion in the ISR and the rest of it in the main loop.

Timer Interrupts.

A timer interrupt usually contains a checklist of several events that must be handled within certain time constraints. Each device is normally read in the ISR to determine its readiness for any action to be performed. As a rule, if the frequency of the interrupt is of a short duration (5 ms to microseconds) a single switch transition may be read several times in contiguous interrupts. To insure against this happening, the switch routine in the ISR uses the LogiSwitch handshake to remove the switch service request, precluding the possibility of it showing up in the next timer interrupt. Note that this only works properly in conjunction with the single-transition hardware debounce.

Timer interrupts are actually a hybrid of the interrupt and polling method in one. They initiate quickly as an interrupt but the ISR code must sample the switch device similar to the polling method. Timer interrupts implemented specifically for a single purpose like a switch service routine should be avoided if possible. Edge-triggered interrupts when available, are much more efficient and are triggered immediately when the switch is activated. Note that the handshake is

not necessary in an edge-triggered interrupt since a subsequent switch service ISR will not be invoked until after the initial switch closure has been released.

Normal Switch Input.

Under most conditions switch input has no importunate urgency. It is said that a delay of 100 ms or more is perceptible to the human eye and generally agreed that 50 ms or less is perceived as immediate. A typical quick pushbutton closure and release normally eats up approximately 200 ms - 300 ms. This is one of the areas operators find the software to appear to be “sluggish”. Removing the polling for release of the switch will dramatically improve the responsiveness of the switch service routine. This is where the handshake adds responsiveness to your routine.

- Use the Polled method for non-time critical switch input. For responsiveness use the handshake. For a little better response in initiating these routines, sample the switch at multiple points in your main loop. Terminating the request with a handshake ACK pulse will result in no time spent in a repetitive loop.

Waiting for Input.

There are times when the program has come to a point where it has nothing else to do until a pushbutton is pressed. In those circumstances it is perfectly proper to poll the switch in a tight loop for pressed status. Again, to make your program snappy, the handshake should be used to terminate the switch service routine, rather than waiting to proceed for the irrelevant reason that someone is still pressing on a button.

- Use the Polled method as shown above for normal input. If the software has no other tasks in the queue, stop and poll the switch for pressed status. Before executing the switch service routine always use the handshake to avoid having to go back and wait for release of the switch.

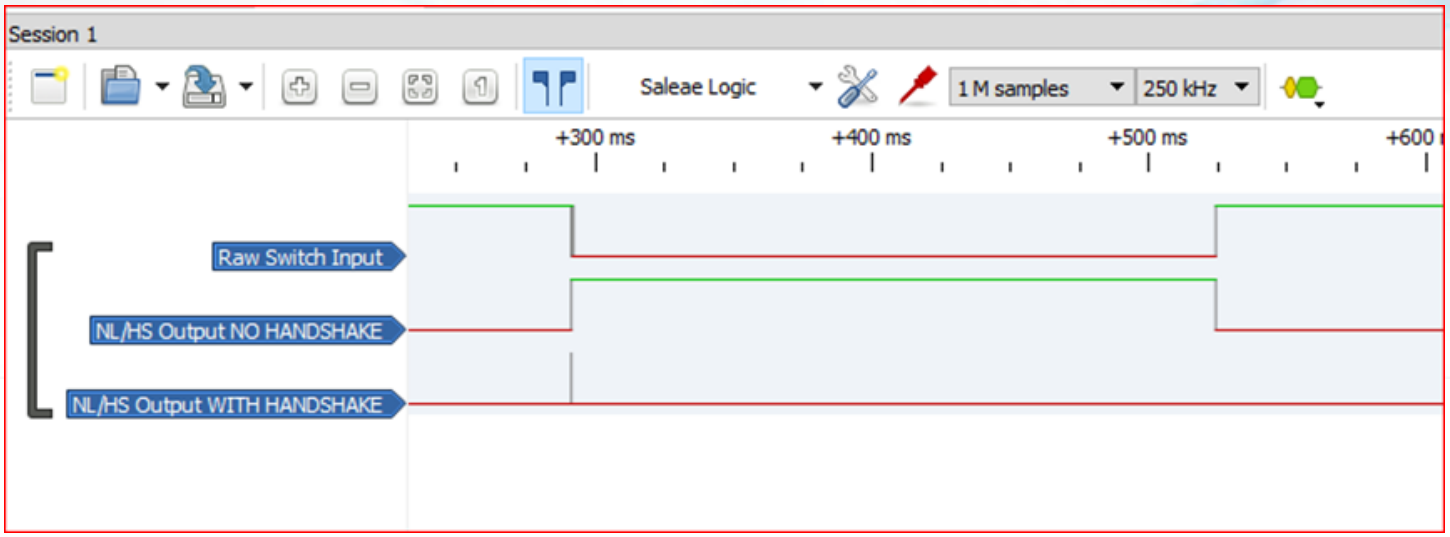

Figure 1. Handshake Timing.

Figure 1 shows the input/output timing of an NL/HS output of a full switch cycle without the handshake vs. the 5 µs - 15 µs response of the LogiSwitch handshake feature.

Note the immediate response of the output in the “no handshake” cycle timing shown in this capture. The switch is held pressed for approximately 240 ms in this example compared to the approximate 15 µs cycle time of the same cycle when the handshake is utilized.

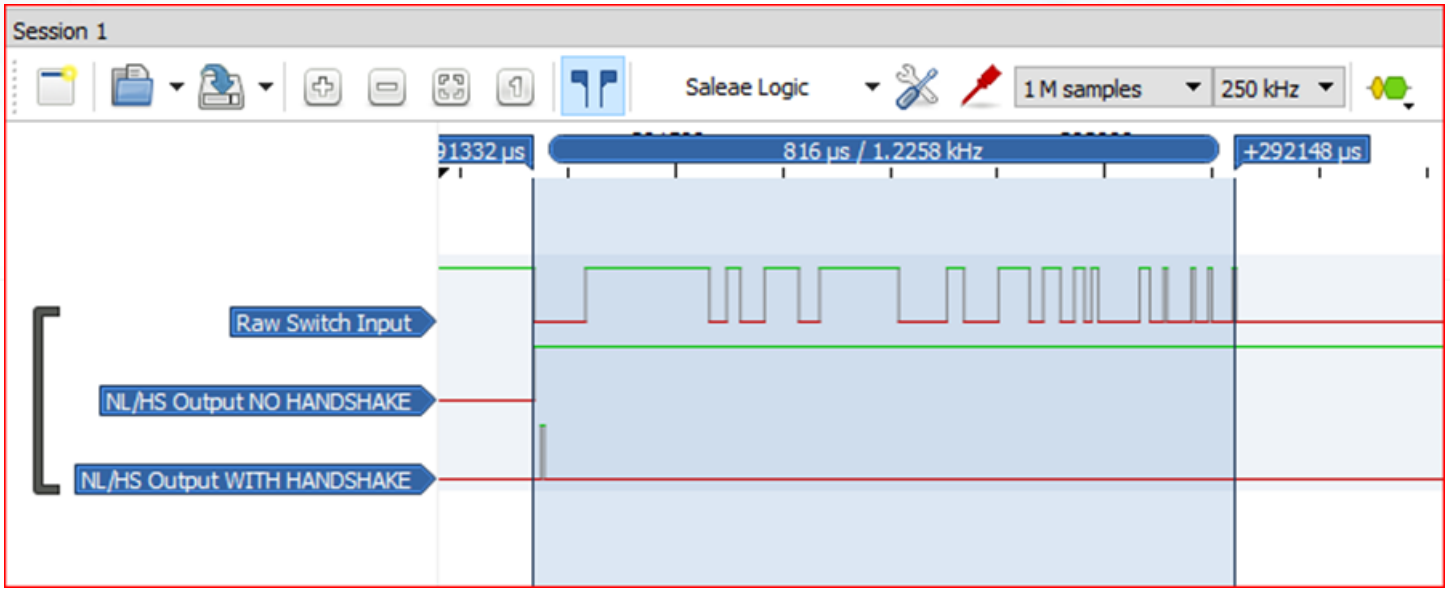

Figure 2. Handshake timing with respect to 816µs “Make” bounce.

Figure 2 is a zoomed-in logic analyzer view of the Figure 1 capture showing the response of a cycle taking advantage of the powerful handshake feature of the LogiSwitch LS100 Series devices compared to a bounce duration of 816 us. Typical response time from LS100 series device is approximately 8 µs after the host computer responds.

LogiSwitch for Interrupt Driven Applications.

Interrupt implementation for switch input should always use hardware debouncing. There are great advantages to triggering interrupts on a rising or falling edge, only if the triggering signal from the switch comes in the form of a clean, single transition. Depending on the priority level of the interrupt and the frequency in the case of the timer interrupts, it may not be necessary to utilize the handshake.

LogiSwitch for (Poll-Free) Polling Applications.

Most polled switch routines require the programmer to be aware of both the pressed and the released state of a switch to keep track of each individual switch cycle when one ends and the

next one begins. The lion's share of time wasted is in polling for release of the switch. Under all circumstances your program will execute with greater response using the LogiSwitch handshake to eliminate repetitive release polling. Note that the time spent with the switch held closed is almost never relevant. Only the program knows when it no longer needs the request to be active.

About the Author

Michael H. Pelkey, Founder and CEO, LogiSwitch LLC

Mike is a serial inventor and serial entrepreneur who has a broad background in designing, using, and manufacturing electronic systems and equipment. In his younger years, Mike pioneered the sport of BASE Jumping.

Prior to founding LogiSwitch, Mike was an Automation Engineer at Jaxx Manufacturing where he designed and built assembly and metalworking machines to increase production rates, such as automatic screwdrivers, optical cut to length machines, and a variety of machines, jigs and fixtures to automate printed circuit assembly operations.

Mike's 40+ year career goes back to the early days of the microprocessor where he was the lead engineer for the first microprocessor-based product in the numerical control industry a Z-axis controller called the Micro-Z. Mike also developed the world's first networked cash register in the mid-1970s, and he founded Macrotech International Corporation, which was a major manufacturer of board-level computer products in the late-1970s and early-1980s.