Max's Hot Beans

By Max the Magnificent

Most non-engineers and many junior engineers are under the impression that when they flip a toggle switch or press a push button, it transitions between the OFF and ON states in a smooth and seamless way. Take a light switch on the wall, for example. When we activate the switch, the light appears to turn on cleanly. It's as simple as that... or is it? In reality, the switch can bounce on and off like a ping-pong ball being dropped on a wooden table. The only reason we don't notice this is that it happens too quickly for the human eye to detect. Things are different when we are talking about microcontrollers (MCUs) with system clocks running at millions, tens of millions, or hundreds of millions of cycles per second. If we don't do something to mitigate the switch-bounce effect, our microcontroller might see a single flick of a switch as representing a large number of transitions.

Switch bouncing around

For the purpose of these discussions, let's consider an SPST (single-pole, single-throw) switch, where the number of poles refers to the number of separate circuits the switch can control, and the number of throws refers to the number of contacts (wiring paths other than 'open') to which each pole can connect. Furthermore, let's consider this in a typical configuration, with its 'main' (pole) terminal connected to GND and its 'secondary' (throw) terminal connected to the positive V_{DD} supply via a pull-up resistor (Fig.1).

When the switch is in its inactive/off position, the NO ('normally open') terminal is pulled up to a logic 1 value. When we close the switch (we use the term 'make' as in 'make the connection'), the 'spring-loaded' mechanical contact may bounce back and forth between 0 and 1 values a number of times before finally settling into a 0 state. Similarly, when we subsequently reopen the switch (we use the term 'break' as in 'break the connection'), the contact can once again bounce back and forth between 1 and 0 values before eventually settling on a 1.

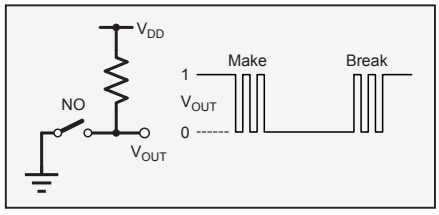

Fig.1. SPST (single pole, single throw) switch.

Have you ever pressed a button on something like a TV controller and been annoyed when the system jumped over a number of channels? What caused this? The answer is almost certainly switch bounce or – more accurately – the fact that the designer of this device didn't take appropriate precautions to mitigate against switch bounce effects. The bottom line is that, if you are in the business of creating electronic projects or products, you are going to have to deal with switch bounce by one means or another.

Debouncing an SPST switch

If you search the Internet, you will discover myriad suggestions for debouncing your switches. Unfortunately, many of these are based on anecdotal evidence ('I heard from a friend of a friend that he heard from an old engineer the tale of a switch that ...'). For example, I've often heard that no switch bounces for more than 1ms (one thousandth of a second), so if we double this to 2ms we should be safe, right? Hold your horses...

One authoritative source is embedded engineering legend Jack Ganssle, who performed (and documented) a suite of tests on a collection of switches (**https://bit.ly/2D9KueF**). Jack reported that the average switch bounce duration was 1.6ms, with a maximum of 6.2ms. (He also found one switch that bounced for a whopping 157ms, although he considered this switch to be a faulty outlier.) Meanwhile, one of my chums who was a technician in the US air force tells me that they specified 20ms, just to make sure.

When it comes to debouncing our SPST, we can opt for hardware or software solutions. You might question why we don't always do things in software, but it may be that our MCU has enough to do and we want to offload as many tasks as possible. A simple SPST debouncer circuit is shown in Fig.2.

The idea here is that charging and discharging the RC (resistor-capacitor) combo smooths (filters) out any bounces from the switch opening and closing. (Choosing the appropriate values for the resistors and capacitors is an exercise in its own right, but we don't need to go into that here because – in a little while – I'm going to make you very, very happy.) The buffer gate is used to 'sharpen' the edge on V_{OUT}. Furthermore, as we see from the symbol, this is a Schmitt buffer with hysteresis, which will stop any small ripples on the input from being seen as switching events.

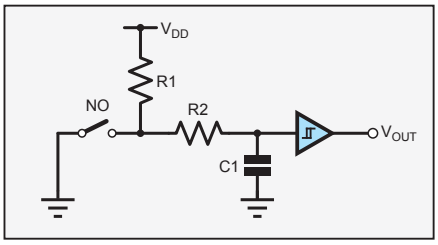

Fig.2. Simple SPST debouncer circuit.

If we decide to debounce our switch in software, then one approach is to use a counter to time how long the switch has been in its new state. Let's assume that the V_{OUT} signal in Fig.1 is connected to one of the inputs on our MCU. Let's also assume that we start off with our switch in its open state, which means $V_{OUT} = 1$. In this case, we could wait until we see V_{OUT} go to 0, indicating the switch has been closed, at which point we could reset our counter to 0 and start sampling this signal periodically (every millisecond, for example). Every time we sample, we increment the counter if V_{OUT} is still 0, or we reset the counter to 0 if V_{OUT} is 1. It's only when the counter says the signal has remained 0 for a specified amount of time – say 20ms – that we consider the switch to be truly closed, at which point we could initiate whatever task(s) we wish to perform. We could perform a similar sequence of actions to determine when the switch has been well and truly opened again.

One problem with both of the techniques – hardware and software – presented above is that they add an element of delay into the proceedings. This delay is probably not significant if the switch is activated by hand, but it may be more of a problem in the case of an automatically triggered limit switch on a piece of industrial equipment. With regard to the software approach, there is a way around this delay, but – once again – we don't need to go there, because I'm about to make you very, very happy.

Introducing LogiSwitch ICs

One of my friends, Mike Pelkey, is credited with being one of the grandfathers of base jumping (parachuting or wing-

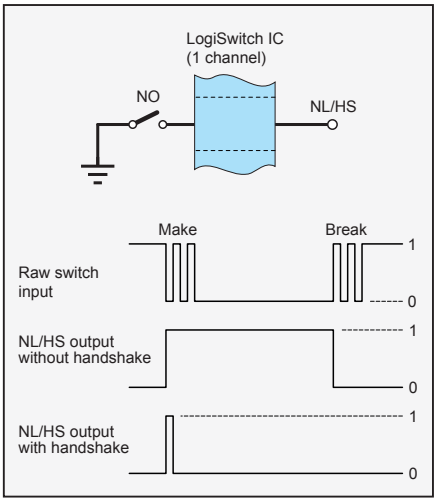

Fig.3. Single channel of a LogiSwitch IC.

suit flying from a fixed structure like a building, bridge, antenna, or cliff). Mike is also an engineer of renown; whose experience goes all the way back to the dawn of the microprocessor.

Like almost every engineer I know, Mike wrestled with switch bounce throughout his career, which encompassed more than 40 years of designing digital systems. Having employed almost every version of traditional hardware and software switch bounce mitigation techniques, Mike decided there had to be a better way. The result of Mike's cogitations and ruminations was the LogiSwitch concept (**https://bit.ly/2GnufxA**). Boasting adaptive bounce detection algorithms and a unique 1-wire handshaking protocol, LogiSwitch ICs reflect the state-of-the-art in switch bounce mitigation technology.

Low-cost LogiSwitch ICs (they start at $3.40; volume pricing is available) are presented in 3-, 6-, and 9-channel variants, where each channel can accommodate a single SPST switch. All variants are available in both surface-mount technology (SMT) and lead through-hole (LTH) packages, the latter making them perfect for hobbyists. The best way to illustrate how these devices work is to consider a single channel, as illustrated in Fig.3.

The LogiSwitch IC is powered via a pair of V_{DD} and GND pins. Each channel on the LogiSwitch IC has a single input pin (equipped with an internal pull-up resistor), which is connected to the raw switch input signal, and a single NL/HS (normally low with handshake capability) output, which is connected to the MCU.

First consider the NL/HS signal without the handshake protocol being employed. In this case, the MCU pin to which it is connected always acts like an input. The main thing to note here is that the NL/HS signal responds almost instantaneously to the initial make or break transition on the incoming signal, so the MCU sees the switch event as soon as it occurs without any artificial delays.

Now let's consider an example scenario in which we have a program loop that waits for the switch to be activated and then performs some action like incrementing a counter; something like: `if (pinNLHS == 1) counter++;` The problem is that the switch could be active for quite some time, so – if we aren't careful – we will end up incrementing our counter multiple times.

There are two ways in which we typically handle something like this. The first is to keep track of the switch's state ('open' or 'closed'), and to only increment the counter when the switch is first closed. The second is to wait for the switch to be closed, then increment the counter, and then wait for the switch to be opened again before restarting the sequence.

LogiSwitch ICs offer another possibility. As soon as the MCU sees that the NL/HS input has transitioned to 1, in addition to performing any tasks like incrementing the counter, we can change the mode of the pin to OUTPUT, pull the NL/HS signal to 0 for 50µS (that's 50 microseconds), then return the mode of the pin to INPUT.

When the LogiSwitch IC sees that the MCU is pulling its NL/HS signal low (this pin is driven by an open-collector output with internal pull-up), it will itself start to drive it to 0. This means that the next time the MCU reads this pin, it will see a 0. In turn, this means that the MCU no longer has to keep track of the switch's state.

I have to say that when I first saw this, I thought, 'Wow! That's clever! Why did I not think of this myself?' Suffice it to say that I'm now using LogiSwitch ICs in all of my projects (for example, I have a 3-channel device in my *Audio-Reactive Artifact* and three 6-channel devices in my *Inamorata Prognostication Engine*).

There's so much more to say about the switch bounce topic, but no time to say it here. Maybe in a future column…

Somewhere over the rainbow

I've been receiving a lot of interest from last month's column regarding my *Audio-Reactive Artifact* project, so I thought I'd provide a brief update. Once I'd finished wiring the LEDs and making sure they all lit up as planned, I decided to run a quick rainbow test pattern to see how the defunct vacuum tubes looked when illuminated from below with different colours. The way I did this was to divide the tubes into thirteen vertical groups (columns) in software, and then cycle a rainbow of colours across these groups. I must admit that I was a little worried that the structures in the tubes would block too much of the light, so I was very pleasantly surprised with the result (Fig.4.)

You can see a short video of this on YouTube (**https://bit.ly/2RAmC9g**), but I have to apologise for the quality; this was the best I could obtain with an iPhone and it really doesn't do the artifact justice. You can also peruse and ponder my Arduino code for this effect if you wish (**https://bit.ly/2BkmKEe**).

Fig.4. A rainbow of colours.

Hot bean Max Maxfield (Hawaiian shirt, on the right) is editor-in-chief at **EEWeb.com** – the go-to site for users of electronic design tools and askers of electronic questions.

Comments or questions? Email Max at: **max@CliveMaxfield.com**