Debounce Explanation - by Chroma

Disclaimer + Credits

The stuff compiled here has been worked out by the community, and as such is not guaranteed 100% accurate, nor is it an exhaustive resource on the subject. There's also some placeholder terminology that may change in the future. Credit to Meat/Flesh, who did most of the research and testing for firmware debounce.

Terminology - Timings

Minimum Click Duration (**MCD**) and Minimum Time Between Clicks (**MTBC**) are the two core concepts in firmware mouse debounce (hardware debounce will be discussed later). MCD dictates the minimum amount of time a click needs to last for, and MTBC is a span of time after a click when new switch signals will not register into an input. Together, they are what is generally referred to as "debounce time".

There are also additional terms which I use to refer to more specific events - Minimum LOW State Duration (**MLSD**), and Minimum Physical Click Duration (**MPCD**).

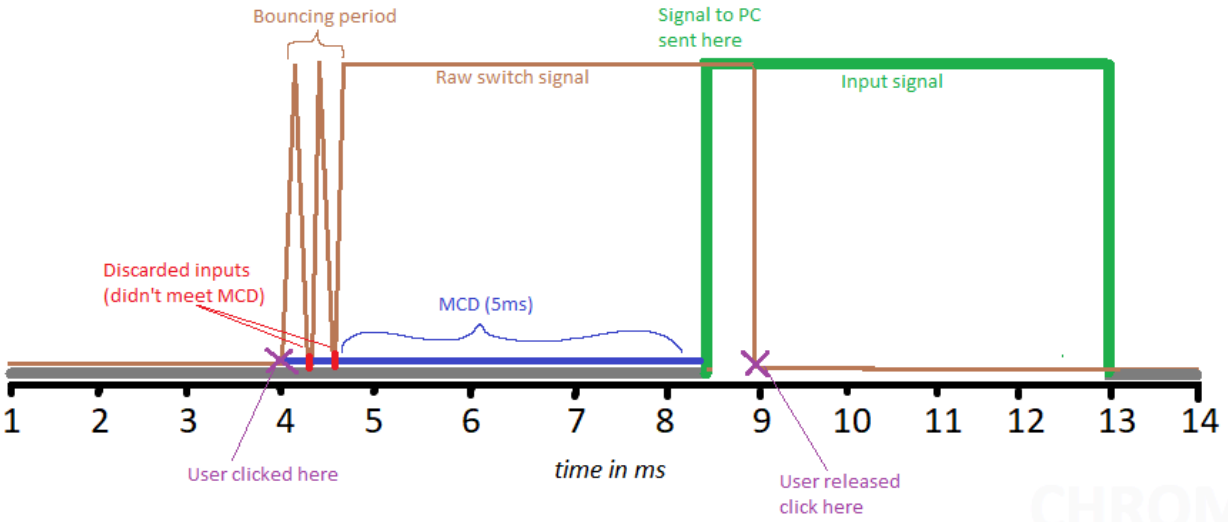
Eager and Defer - Mouse-down behaviour

The next two concepts relate to how the mouse executes firmware debounce for mouse-down events, based on the timings provided by MCD. There are two main methods for this, and the currently accepted terms for them are **eager** and **defer**. You may have also heard of them as "ignore", "hold", "mask", "buffer", etc.

DEFER-TYPE: This form of debounce, upon receiving a signal from the switch, will wait to see whether it is longer than the MCD (Minimum Click Duration) before accepting it as an input. Clicks shorter than the MCD will be discarded. This way, the multiple short "bounces" a switch contact makes when first closed are ignored, and only when the switch has "settled" does the mouse accept it as a proper click.

The main upside of defer debounce is that it's inherently effective at filtering out noise like double-clicking and slam-clicking (usually too short to reach MCD). There are two big downsides, however: firstly, because the algorithm needs to wait to see whether a click is long enough before sending it, defer-type debounce directly introduces click latency. If the MCD is 8ms long, that's 8ms of additional click latency. Secondly, if the MCD is set too high, shorter clicks may be discarded, leading to missed inputs.

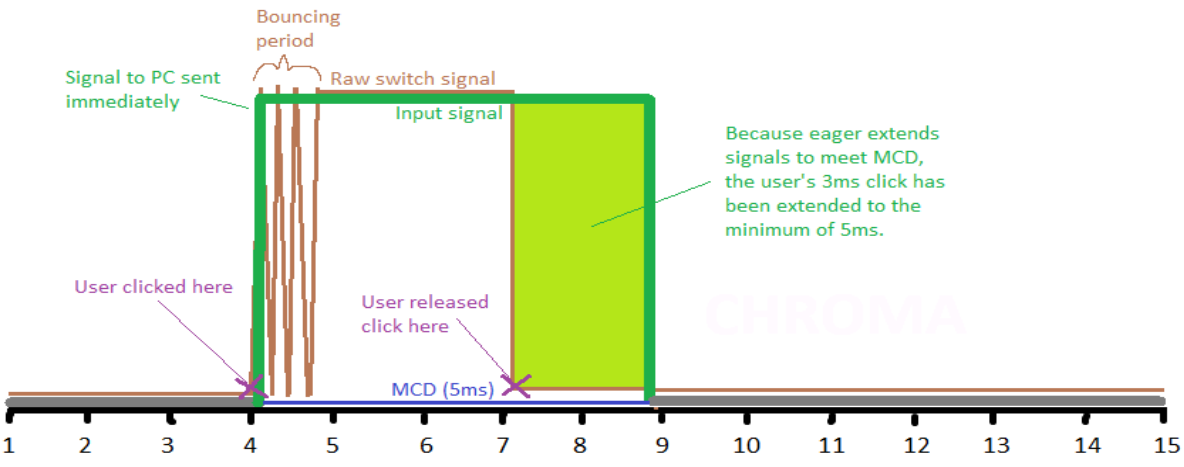
Successful Click using Defer Debounce



EAGER-TYPE: This form of debounce behaviour is so-called because it accepts the first signal from the switch as an input, even if it is shorter than MCD. To prevent repeated inputs from the following contact bounces, it will immediately extend that first signal into the set MCD length, effectively masking over the rest of them.

The upside to eager debounce is that no additional click latency is introduced - the first signal received is what is sent. The downside is that without further intervention (often in the form of a small defer period), the algorithm is prone to signal noise like slam-clicking. If the MCD is set too high, clicks can also “stick” and make tasks like tap-firing more difficult.

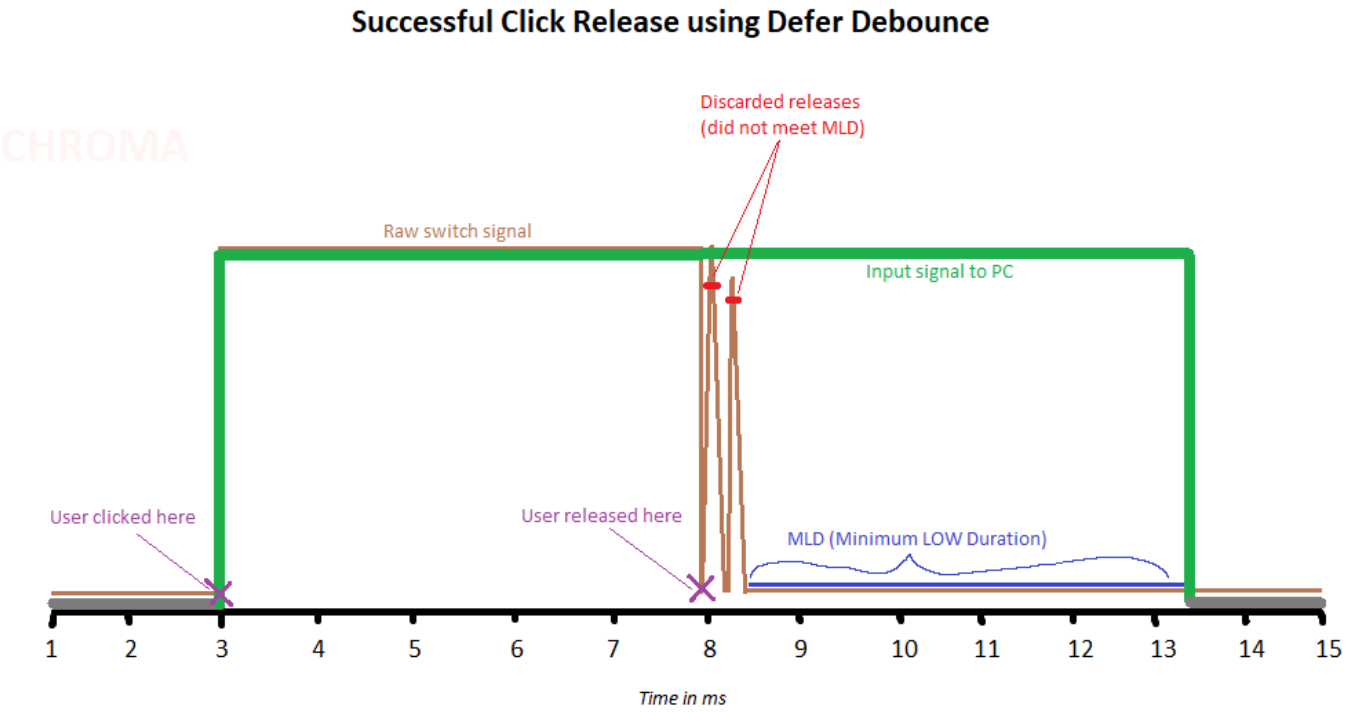
Successful Click using Eager Debounce



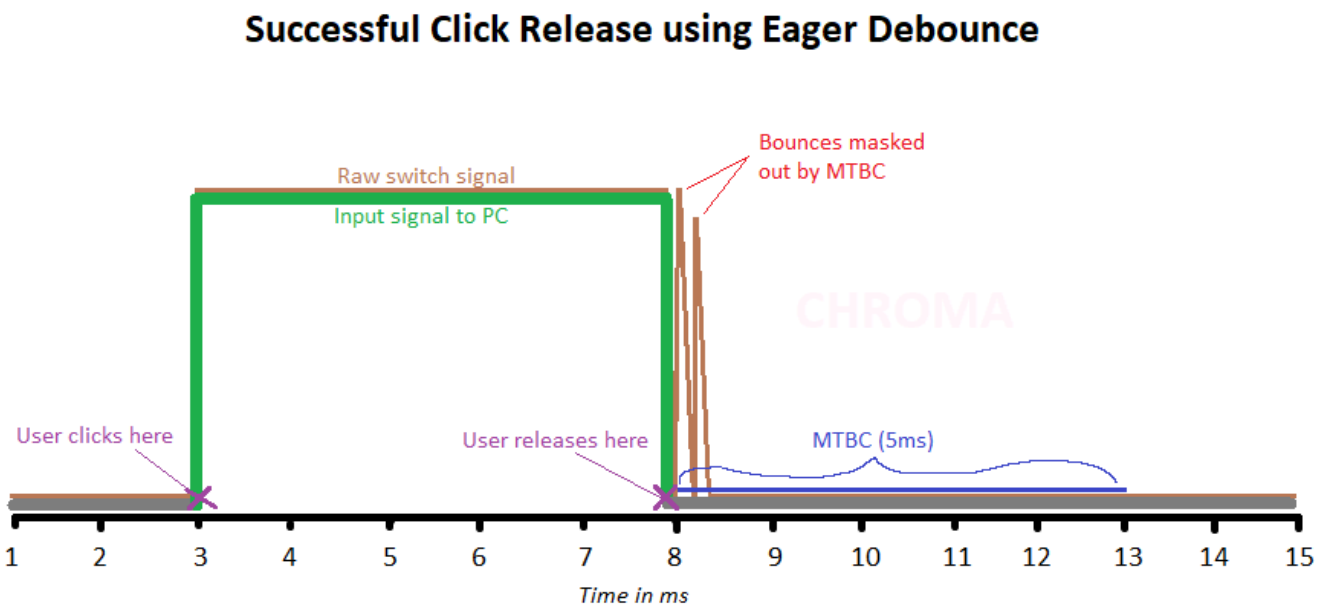
Eager and Defer - Mouse-up behaviour

Since contacts can also shift when lifted (albeit less than when they come together), debounce is also required for mouse-up events to prevent erroneous inputs there. Like mouse-down, mouse-up debounce is based on the two concepts of defer and eager, but here it defines the measured MTBC length.

DEFER TYPE: This form of mouse-up debounce waits until a set time (**MLSD**, Minimum LOW State Duration) has been reached with no “HIGH” signal from the switch. This filters out the noise from contacts lifting, but results in an extended MCD due to the mouse waiting for the “LOW” event to be long enough before allowing the click to release.



EAGER TYPE: This form of mouse-up debounce takes the first “LOW” signal from the switch and extends it into the MTBC value. This does not result in an extended MCD, but like mouse-down eager debounce, it is not noise-resistant. If the switch design has any hysteresis in it (as is common with 2-pin tact switches), eager debounce may cause unintended releases and even double-clicking.



Symmetrical and Asymmetrical - Mouse-down/up combined

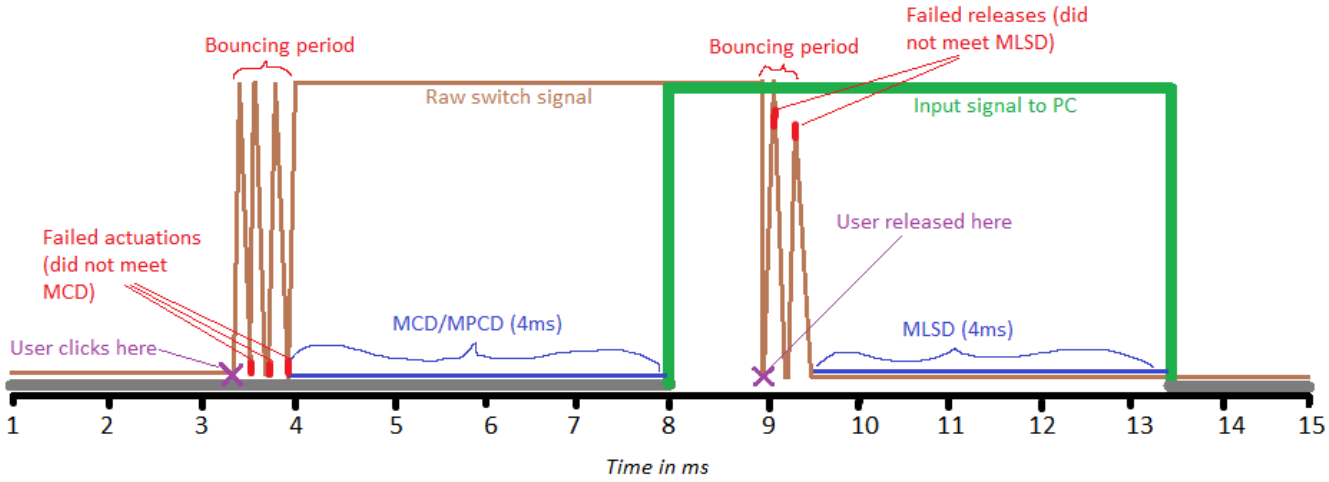
Mice need not use either defer or eager debounce exclusively, and manufacturers can choose which they want to use for both mouse-down and mouse-up (along with what MCD/MTBC values), entirely independent of each other. This results in four different possible combinations:

- **Sym_defer**: Defer debounce on both mouse-down and mouse-up
- **Sym_eager**: Eager debounce on both mouse-down and mouse-up
- **Asym_defer_eager**: Defer debounce on mouse-down, eager on mouse-up
- **Asym_eager_defer:** Eager debounce on mouse-down, defer on mouse-up

This terminology is based on the recommended conventions provided by QMK's Debounce API, since it's also applicable to mice. Mouse debounce is almost universally per-key though, so the "pk/g/pr" tags are omitted.

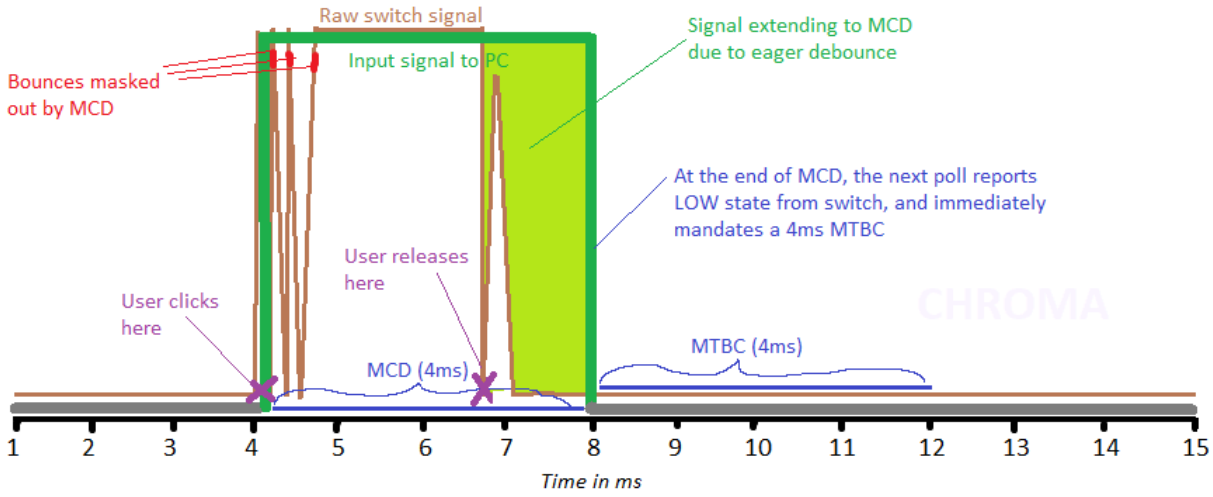
SYM_DEFER: Most common implementation of debounce, and seen on most mice with mechanical switches (a major exception being Logitech). Sym_defer results in a MCD and MTBC that are equal to one another (as the former is the result of the latter on the next click), and an MPCD that is equal to the MCD. MLSD does not technically have to be equal to MPCD/MCD, that's up to the manufacturer.

Click + Release using Sym_Defer

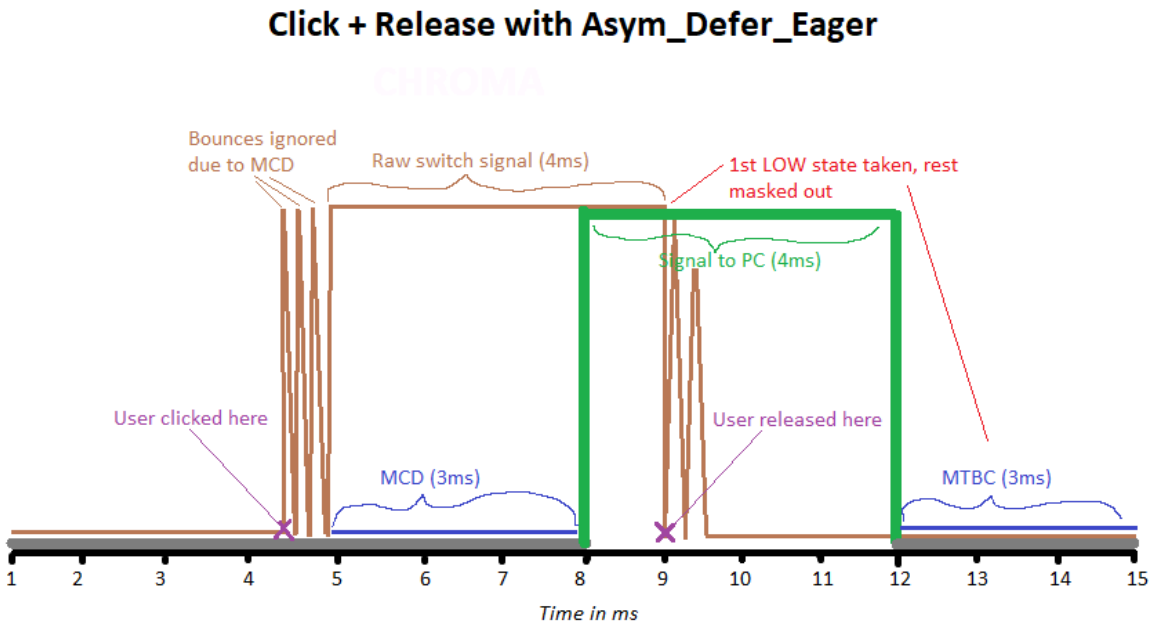


SYM_EAGER: Most common implementation for mice using eager debounce for mouse-down. Pure sym_eager results in a MPCD that can be as short as one poll event (due to eager debounce extending to meet MCD). The MCD is usually equal to the MTBC, although it doesn't have to be.

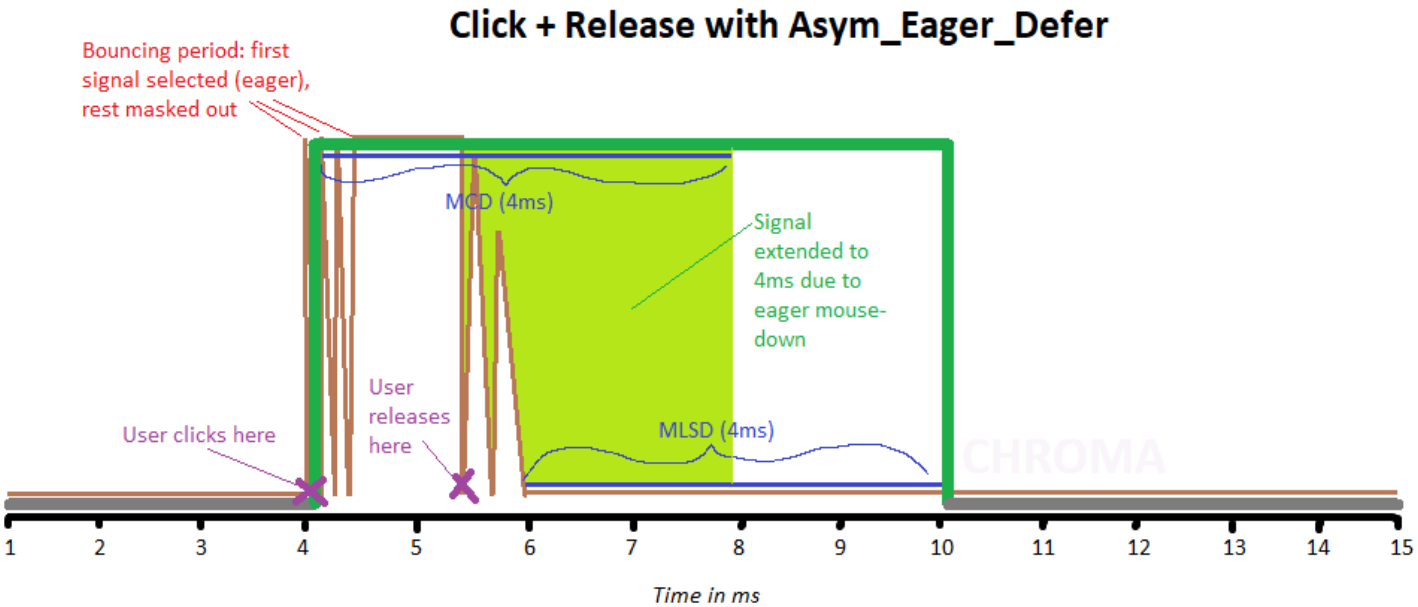
Click + Release using Sym_Eager



ASYM_DEFER_EAGER: The rarest form of debounce that results in an MTBC at least double the length of MCD, due to the combination of an eager mouse-up and a deferred mouse-down. Useful for if clicks of extremely short duration are required without double-clicking, at the cost of not being able to follow up with another input quickly.



ASYM_EAGER_DEFER: Another rare form of debounce. This behaviour allows for relatively long MCDs with extremely short MTBC (basically the polling interval time), due to the eager mouse-down and deferred mouse-up.



Testing Debounce - Mouse-specific Implementation

To figure out how debouncing has been implemented on a particular mouse, there are several ways to test it. The main program used to do such testing is “L/R.exe” (available on this OCN thread), which shows the duration of clicks. This is used to establish the length of timings such as MCD/MPCD, and can also be used to check for eager/defer behaviour.

To use the program, open the .exe for the mouse button you intend to use (l.exe for LMB, r.exe for RMB). When clicking, **do not** click in the command prompt window, this messes up the results. If done correctly, with each click you should see a readout of click duration in milliseconds:

To establish the **MCD**, press as lightly and quickly as possible on the mouse button. This should be less of a click than a sharp tap, with the goal of getting as low of a number in the program as possible. For most mice you will quickly reach a consistent minimum (~22ms in the above image), which is your Minimum Click Duration.

Testing for **defer/eager** behaviour on mouse-down can also be done in LR by hand, though it requires a bit more practice. To do so, start by clicking in the same way as you would for MCD testing.

If the mouse is utilizing **defer-type** debounce with sufficiently high timings, you’ll begin to notice the shorter clicks not registering on screen. This is due to the switch’s HIGH signal not being

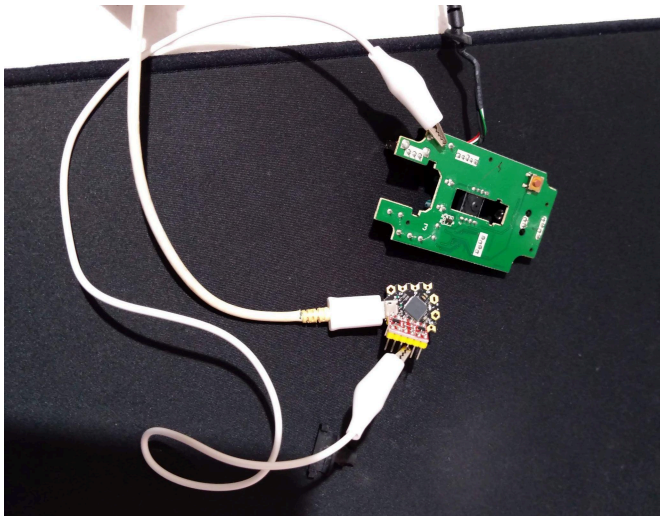
long enough to reach the MCD. If you're still not sure at this point, results from a latency test (bump test or wiring middle pins) can be evidence for defer-style debounce, as the debouncing will add measurable click latency (greater than MCD).

With **eager-type** debounce, you'll notice the shortest clicks registering on-screen with durations much longer than what you clicked for, indicating that they are being extended to meet MCD. Mice using eager-type debounce will typically have low click latency numbers as well due to the lack of a wait buffer on each click.

From here, testing gets more difficult. In order to establish **MTBC** timings, the program "old_lr.exe" is used, which prints click duration (0002 for LMB) as well as the time from the last mouse-up event to the current mouse-down event (0001 for LMB).

```
C:\Users\ \Desktop\lr\old_lr.exe
id:10044        x:0     y:0     btn:0002        time:21.961800
id:10044        x:1     y:0     btn:0001        time:373.854900
id:10044        x:1     y:0     btn:0002        time:22.221100
id:10044        x:1     y:0     btn:0001        time:167.804700
id:10044        x:1     y:0     btn:0002        time:21.943700
id:10044        x:0     y:0     btn:0001        time:177.152500
id:10044        x:0     y:0     btn:0002        time:21.968800
id:10044        x:0     y:0     btn:0001        time:175.062700
id:10044        x:0     y:0     btn:0002        time:22.029100
id:10044        x:0     y:0     btn:0001        time:169.025200
id:10044        x:1     y:0     btn:0002        time:21.772600
id:10044        x:1     y:0     btn:0001        time:178.031900
id:10044        x:1     y:-1    btn:0002        time:21.992900
id:10044        x:0     y:0     btn:0001        time:197.131500
id:10044        x:0     y:0     btn:0002        time:21.973400
id:10044        x:0     y:0     btn:0001        time:174.019200
id:10044        x:0     y:0     btn:0002        time:21.988000
id:10044        x:0     y:-1    btn:0001        time:162.921600
id:10044        x:1     y:-1    btn:0002        time:22.063800
id:10044        x:0     y:0     btn:0001        time:188.024100
id:10044        x:0     y:0     btn:0002        time:21.977700
```

Input spamming is necessary for this part - "bolt clicking" seems to be the easiest way to achieve this by hand. Do this until you get the lowest number possible more or less consistently on 0002 (or 0004 if you're using RMB) - that should be your MTBC. On most mice that value is identical to MCD.

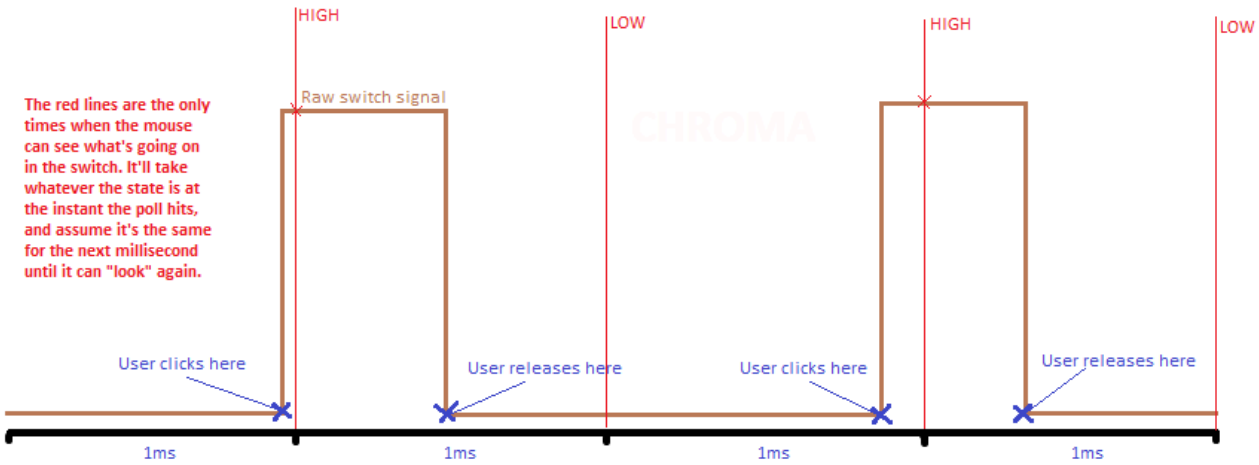


Further testing (determining asymmetrical/symmetrical debouncing, finding length of defer period in front of eager mouse-down, etc.) becomes difficult by hand. I personally use an Arduino for this, since it's able to send signals of precise length and interval. If you do go down this route, keep in mind that most modern mice use 3.3V logic, and you'll need a voltage converter to step down from the 5V that Arduinos typically output at.

Firmware Debounce FAQ - Miscellaneous Stuff

“But can it drag click???” - Maximum CPS depends on the debounce timings. Once you know MCD and MTBC, add them together and divide 1000 (number of ms in a second) by the sum. That is the maximum CPS you’ll be able to achieve.

What’s the minimum debounce value? - Debounce time is affected by polling rate. Even without debouncing, the lowest MCD/MTBC value will be 1/1 (at 1000hz). This is due to the polls defining whether a click is in HIGH or LOW state.. Once polled, the signal is locked to HIGH until the next poll hits 1ms later, then it will be locked to LOW until the poll after that. The lower your polling rate, the longer these timings are going to be - at 125hz the MCD/MTBC become 8/8, for a maximum CPS of 62.5, whereas if you go up to 8000hz, it’s 0.125/0.125 (theoretical maximum of 4000 CPS).



Polling intervals at 1000hz

Can I change the debounce time on my mouse? - There are programs that exist to increase debounce time, mostly to filter out double-clicks from failing switches. These work somewhat like the *asym_*_eager* method in that the MTBC is extended to mask out additional clicks. If you’re looking to decrease debounce time, it’s rarely possible if the manufacturer doesn’t provide the option (there are a few exceptions).

What debounce value should I use? - For mice that do offer a debounce time setting in software, the rule of thumb is the lowest setting that doesn’t double-click. This testing page is handy for figuring out what that value is.

Do optical switches use debounce? - Usually, yes. While there are exceptions (like Bloody’s A-series), mice with optical switches typically use the *sym_eager* algorithm for debouncing. A longer defer period is sometimes added after liftoff as well, to prevent slam clicks.

Alternate Debouncing Method #1 - Hardware Debouncing

Everything written above has been in regard to firmware debouncing, where time-based processing is done by the MCU to remove the undesirable parts of a switch's signal (ie. the bouncing period). This is the technique used by basically all mice on the market, but there is another method - hardware debouncing. Hardware debouncing techniques use special circuits and components to achieve the same result as firmware debouncing, while also removing the disadvantages such as additional click latency and CPS limitation.

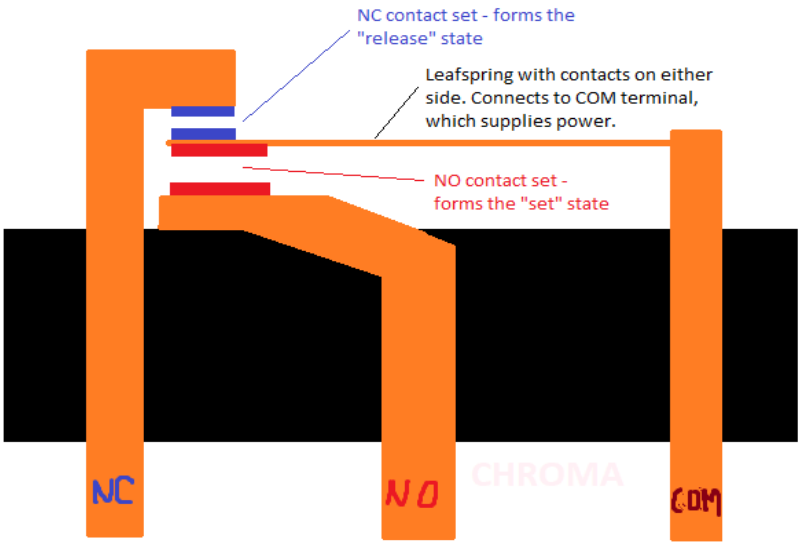
So if it works so well, why don't more mice use it? For starters, hardware debounce adds additional cost through extra components and complexity. Firmware-only debounce, when implemented properly, works basically just as well, and few manufacturers have seen the need to use anything else in their designs. There are just a few exceptions - the Mad Catz Mojo M1, Zaunkoenig M1K/M2K and the Atompalm Hydrogen all utilize the **firmware set/release latch** method of hardware debouncing.

Firmware S/R Latch - How it works

Out of all the various hardware debouncing methods, the firmware-emulated set/release latch method is best suited for this application. When implemented correctly it adds no additional latency due to being time-independent (unlike RC filtering and traditional firmware debounce), and makes the switch basically impervious to double-clicking until total mechanical failure (also unlike firmware debounce, where double-clicking can begin to occur when the bouncing period exceeds the set timings). Its only real downside is that it requires a second GPIO pin per switch.

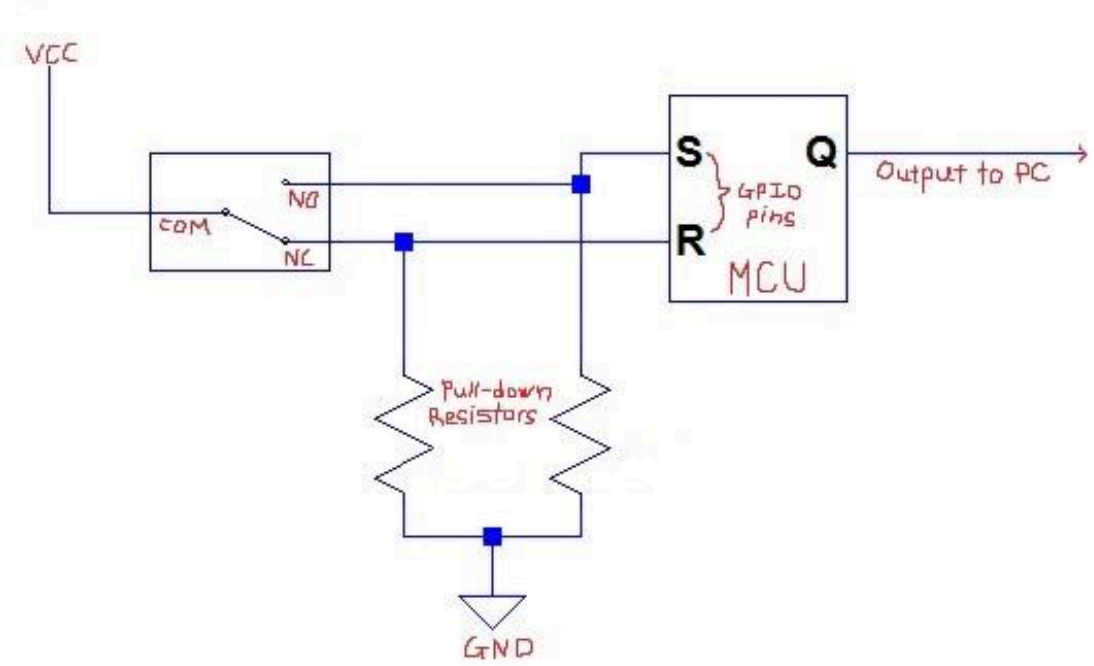
It's worth noting that there is a fully hardware-based way to do SR latch debounce, with a transistor circuit. However, this method can potentially introduce latency due to propagation delay, so emulating the latch in firmware (at the cost of needing that second GPIO pin) is generally preferred.

In essence, SR debouncing is based on the switch having three distinct states - set, release, and neither. This is possible due to SPDT switches having two sets of contacts, with NC being the "release" and NO being the "set". SPST switches (like Omron D2FC and Kailh) either cannot be used, or perform suboptimally since they lack a proper NC contact.



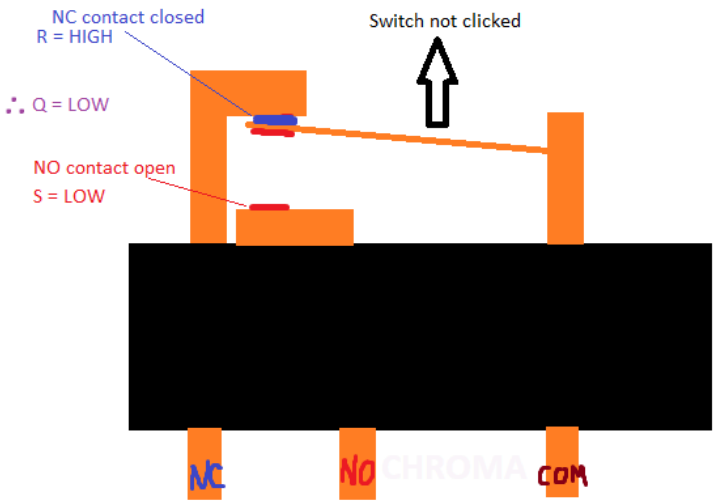
Anatomy of SPDT Microswitch

As with a standard SPST switch, the COM terminal is connected to VCC, but NC and NO each get their own GPIO pin (a firmware debounce implementation would require only the NO pin to be connected). A pull-down resistor is attached to each pin to make sure the output is LOW when the contacts are not closed. This is a diagram of what it all ends up looking like (the resistors are integrated into the MCU):

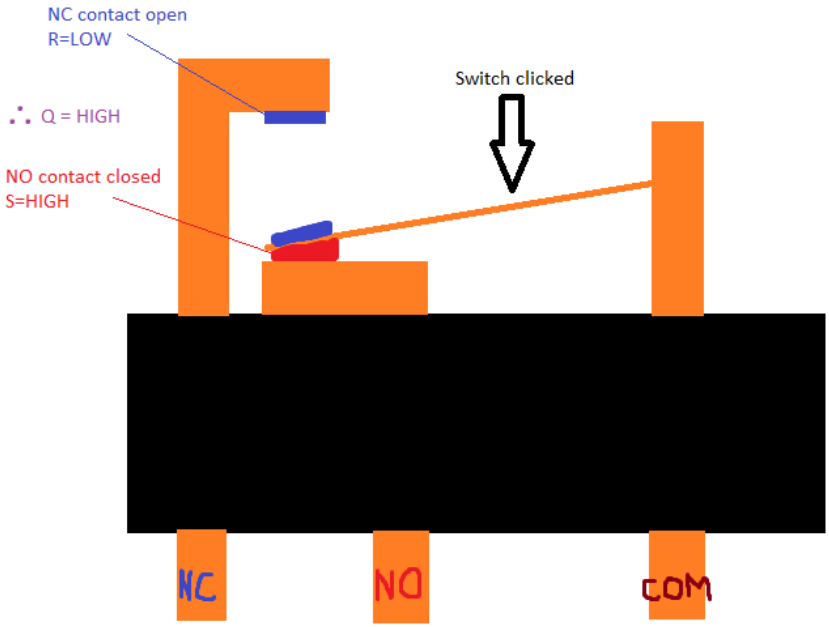


Both the set and release terminals are hooked up to the MCU, and so both of their states (HIGH or LOW) can be read independently. With this information, the firmware will emulate the hardware SR latch by creating a **Q-value**.

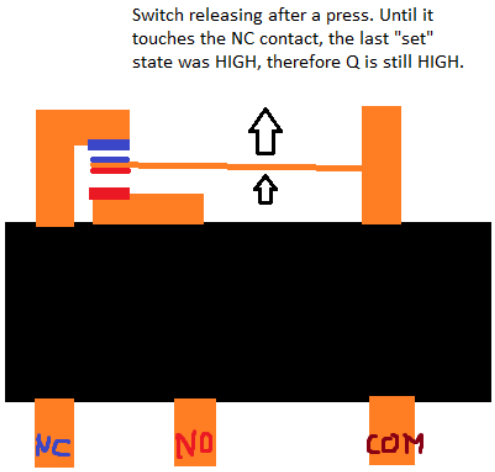
The Q-value is based on the most recent HIGH state for either set or release. When the user lets go of the switch and the NC contacts close, R (release) will go HIGH, and the Q-value gets set to LOW. This Q value is then what the MCU reports to the PC as the input.

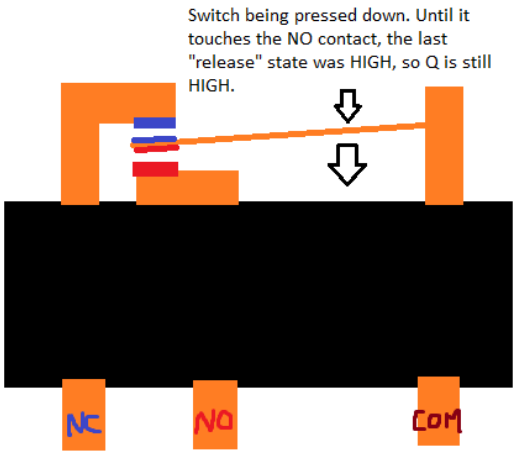


The Q-value then stays LOW until the user clicks, and the NO contacts close. S (set) then goes HIGH, at which point Q gets set to HIGH, and the MCU reports a HIGH input to the PC on the next poll.



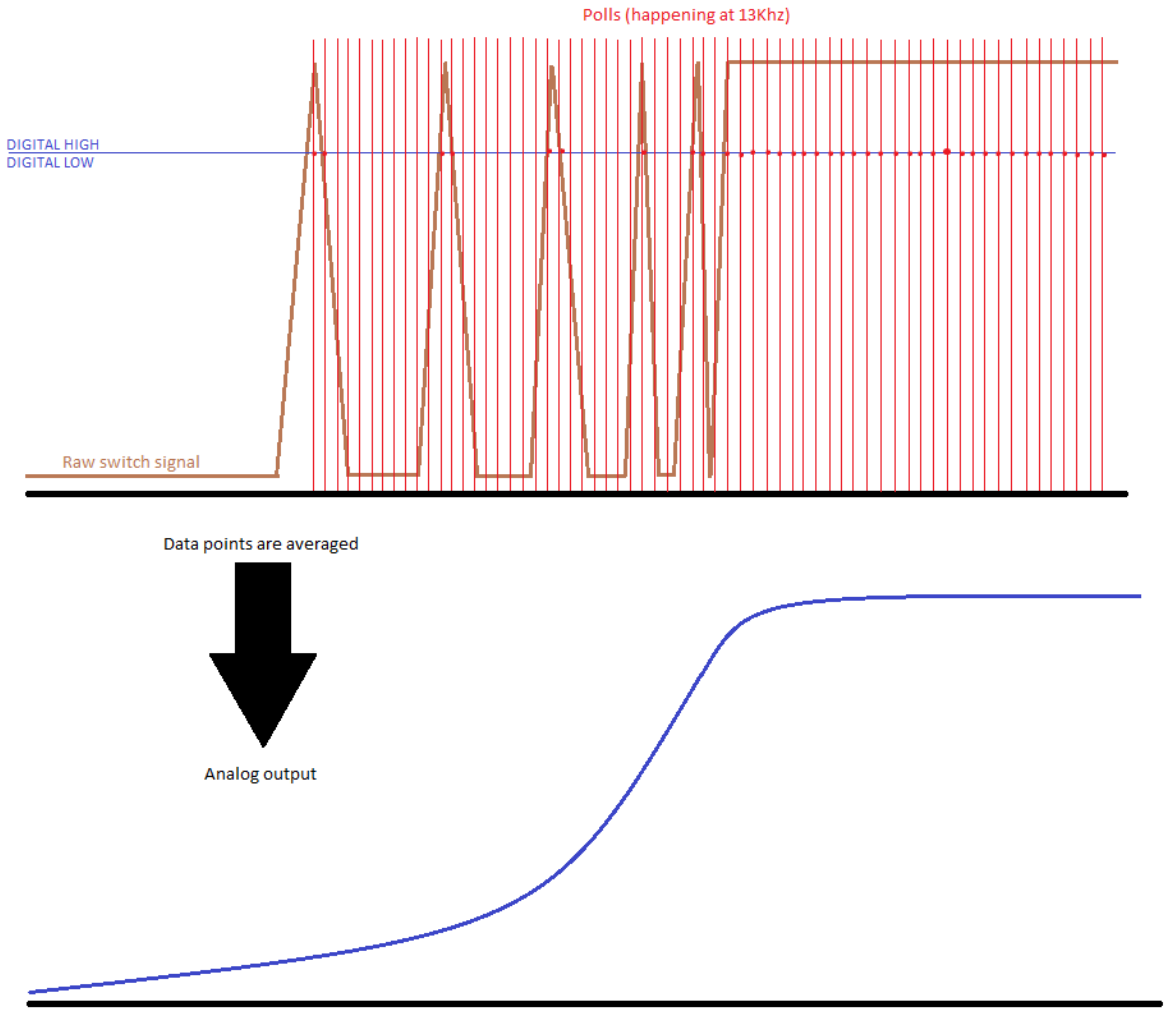
When the switch is in the middle of its travel and *both* S and R are LOW, Q will hold its last known value. This means that during the bouncing period, the first time NO goes HIGH the Q-value is set to HIGH, and when the contacts bounce apart, the Q-value (and thus the input to the PC) stays HIGH, completely unaffected by the bouncing of the switch. It takes a complete mechanical change of the switch (closing the NC contacts) to “release” the Q-value back to LOW, and the same is true for the opposite (“set” Q-value to HIGH). This ensures that double-clicks will never occur even as the switch wears out and its bouncing time gets longer.





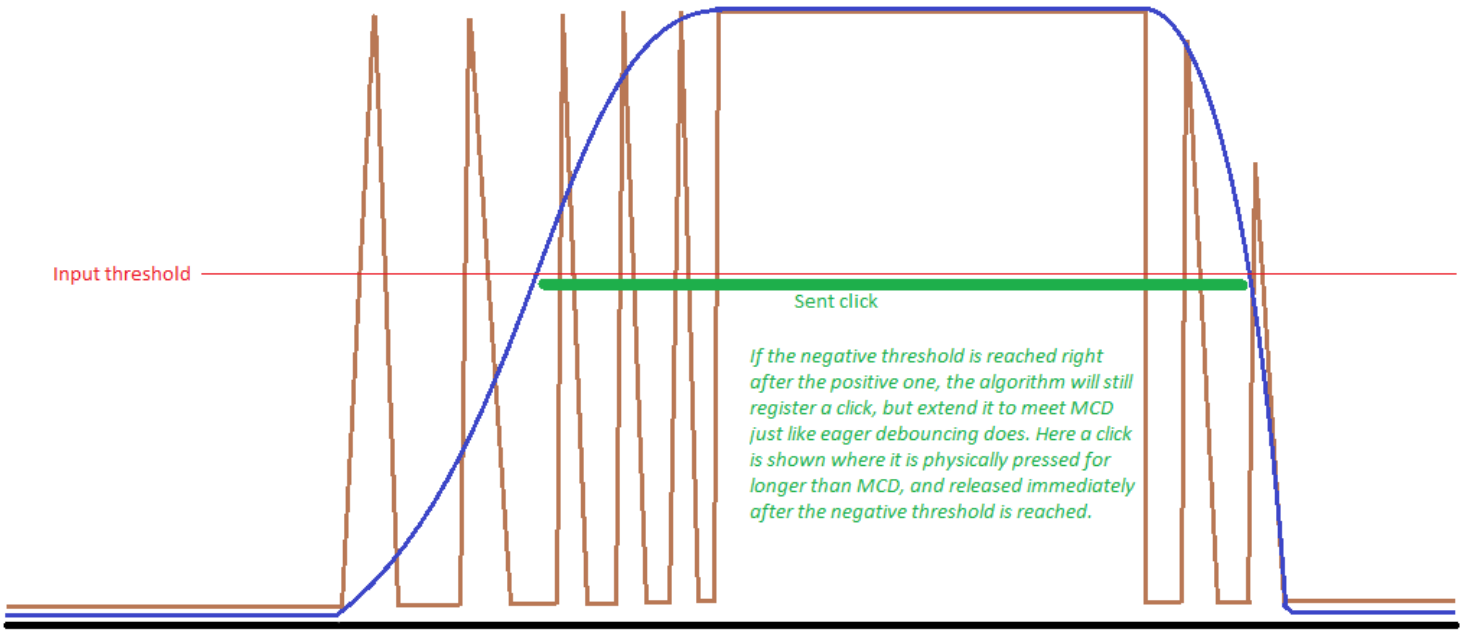
Alternate Debouncing Method #2 - Digital-Analog Conversion

This is the debouncing method that Endgame Gear employs for their XM1 lineup. It's technically also a type of firmware debouncing, but since it's pretty unique it gets its own section. Unlike the previously discussed debouncing methods, EGG's implementation doesn't directly use the switch's inherent HIGH/LOW states, but instead polls the NO pin internally at 13,000 hz starting whenever it detects a HIGH value. The mixture of HIGHs (1) and LOWs (0) the polls "see" as the switch contacts bounce are averaged to produce analog values - it's unknown how incremental these values are since EGG hasn't specified the wordlength. The whole thing is somewhat similar to the principle behind PWM output, but in reverse.



With the analog output, firmware engineers can set a threshold at which they believe the click should be sent as an input. The lower the threshold, the quicker the input will register since it'll happen earlier in the bouncing period, but double-clicking can become an issue as the switch wears out and its behaviour becomes more inconsistent.

Since the XM1 has a clearly defined MCD (especially for certain firmware versions where it caused issues), it appears that the debouncing algorithm has something in common with eager debouncing - once the threshold is reached the input to the PC is held for a certain time period. After the time period expires, the click is held if the negative threshold hasn't been reached, and ended if it has.



The upsides of this debouncing behaviour over traditional is that click latency can be decreased, while still being noise-resistant (slam clicks are an issue in traditional eager debounce, whereas with analog conversion short peaks can be filtered by the input threshold). It also has some ability to adapt to longer bouncing periods as the switch wears out, and by setting parameters such as input threshold and MCD, firmware engineers can easily tailor debouncing algorithms that are optimal for a particular switch or shell design. Unlike latch debouncing it can be done entirely in firmware (although an RC filter can be used to smooth the inputs), which saves pins on the MCU.

Overall, in order of effectiveness digital-analog conversion sits between traditional firmware debouncing and latch debouncing. In click latency and noise resistance it's superior to the former, but still inferior to the latter. It's worth noting that EGG owns the patent to this technology, so we're quite unlikely to see this form of debouncing adopted widely beyond their own products.