De-bounce switches using software defined Schmitt triggers

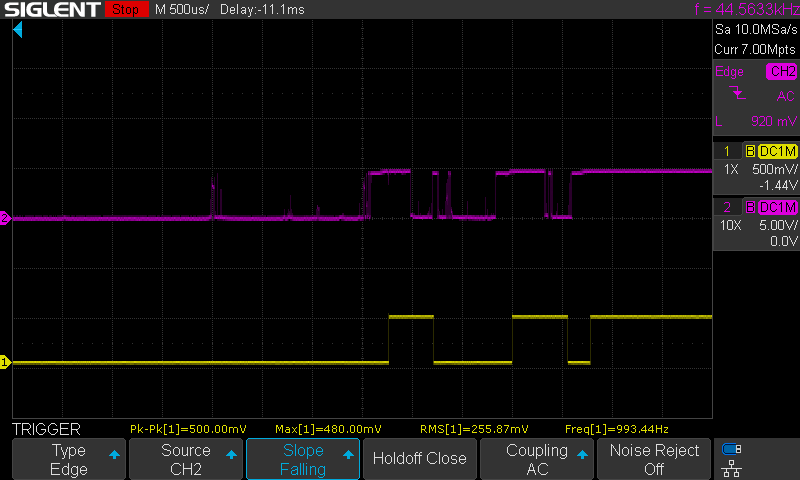
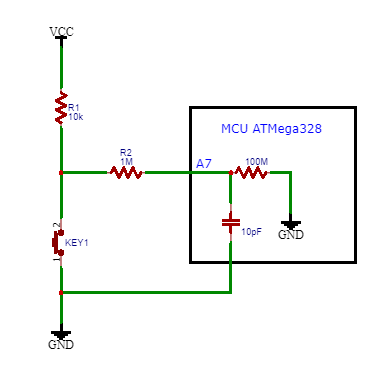
Mechanical switches, sensors and rotary encoders do not normally produce clean and noise free signals that can be used reliably by discrete logic, or microcontroller, circuits. The state of a mechanical device can be incorrectly captured if software driven polling is used as the state can be indeterminate at the instant of polling. Worse, if the mechanical device is connected to an integrated circuit, or software, counter the noise produced can be interpreted as multiple events.

Image 1

The purple trace shown in image 1 is the signal produced by a push-button switch connected to +5v via a 10K resistor, as shown in the circuit below, and the noise generated by the metal contacts rubbing against each other is clear.



A common microcontroller (MCU) such as the Microchip ATMega328 is used on many maker boards such as the Arduino UNO and Nano. Both digital and analog I/O pins are featured on these microcontrollers and it is the digital inputs that are normally used to interface with mechanical switch type devices. It is easy to find Arduino code on the Internet that de-bounces switches using time delays to allow switch induced noise to be ignored – just ask ChatGPT! But it can be difficult to configure the timing and sometimes a trade-off has to be made between reliability and speed. If digital de-bounce software is not adequate, or reliable enough, often the designer will resort to a conventional analog solution using passive RC networks and Schmitt trigger logic ICs such as a 74HC14 hex Schmitt device, using more passive and active components and adding to the size and cost.

I wanted to create a software solution that would emulate an RC/Schmitt circuit with the minimum of passive components.

First I developed a software implementation of a Schmitt trigger with high and low voltage switching points and therefore an element of hysteresis.

An analog input on the MCU is used to create a software implementation of a Schmitt trigger and the code is shown in fig 1.

bool Schmitt = false;

const int threshold\_high = 600;  // in counts

const int threshold\_low = 400;   // in counts

const int LED\_PIN = 9;

void setup() {

  pinMode(LED\_PIN, OUTPUT);

}

void loop() {

  // Better code suggested by Google Bard

  // Read input from switch (assumed to be connected to A7 pin)

  int A7\_input = analogRead(A7);  // assuming 10-bit ADC on 5V reference

  // Check if the switch state has changed

  if (A7\_input > threshold\_high && !Schmitt) {

    Schmitt = true;

  } else if (A7\_input < threshold\_low && Schmitt) {

    Schmitt = false;

  }

  // Turn on or off LED based on boolean variable

  digitalWrite(LED\_PIN, Schmitt);

}

Fig 1

A Boolean variable called Schmitt is defined, which can have one of two binary values TRUE (1) or FALSE (0). Also two analog thresholds are defined in terms of count values that can be compared with the number produced by the MCUs 10-bit Analog to Digital Convertor (ADC) with a range of 0 to 1023 equating to 0 to 5vDC. The defined thresholds equate to approximately 2.9vDC and 1.9vDC i.e. above and below the hallway voltage of 2.5vDC and with 1vDC of hysteresis. The code listed in Fig 1 just drives an LED on digital pin 9 of the MCU but could be used with the built-in timers and counters in the MCU or various software routines e.g. to assess the steps and rotational direction of a rotary encoder.

Next how to implement a low pass RC filter to remove high frequency switching noise. The obvious solution would be a resistor and a capacitor, but as a devout Yorkshire man I wondered if the cost of either could be eliminated? And yes – looking at the data sheet for the ATMega328 the analog input pins are connected to a CMOS comparator with an input resistance of 100M and capacitance of about 10pF. So the ADC input pin can be fed from a high value resistor without significantly affecting the voltages from the mechanical switch i.e. 0v or 5v. Using a high value resistor e.g. 1M creates an effective low pass filter just using the 10pF capacitance of the analog input pin resulting in a time constant of 10pF x 1M = 10uS or in the frequency domain a cut-off frequency of 15.9 KHz.

The combination of a passive RC filter together with a software defined Schmitt trigger is very effective at eliminating noise and de-bouncing switches as is shown by the yellow trace of image 1.

If a longer time constant is needed I have used a 15M resistor on the analog input pin, which has a time constant of 150uS and a cut-off frequency of 1 KHz.

Space and cost are reduced as one resistor; one capacitor and logic IC have been replaced by a single resistor and a few lines of code. The saving in components and the flexibility offered by a software Schmitt trigger is even more apparent if interfacing to a rotary encoder with a built in push-button function as three inputs would be required.

Finally makers can vary the value of the input resistor and the software defined Schmitt thresholds to optimise this deign to their specific needs.

Project materials are available for download at: <https://github.com/BenthamITServices/debounce_switches_using_software_schmitt_triggers>