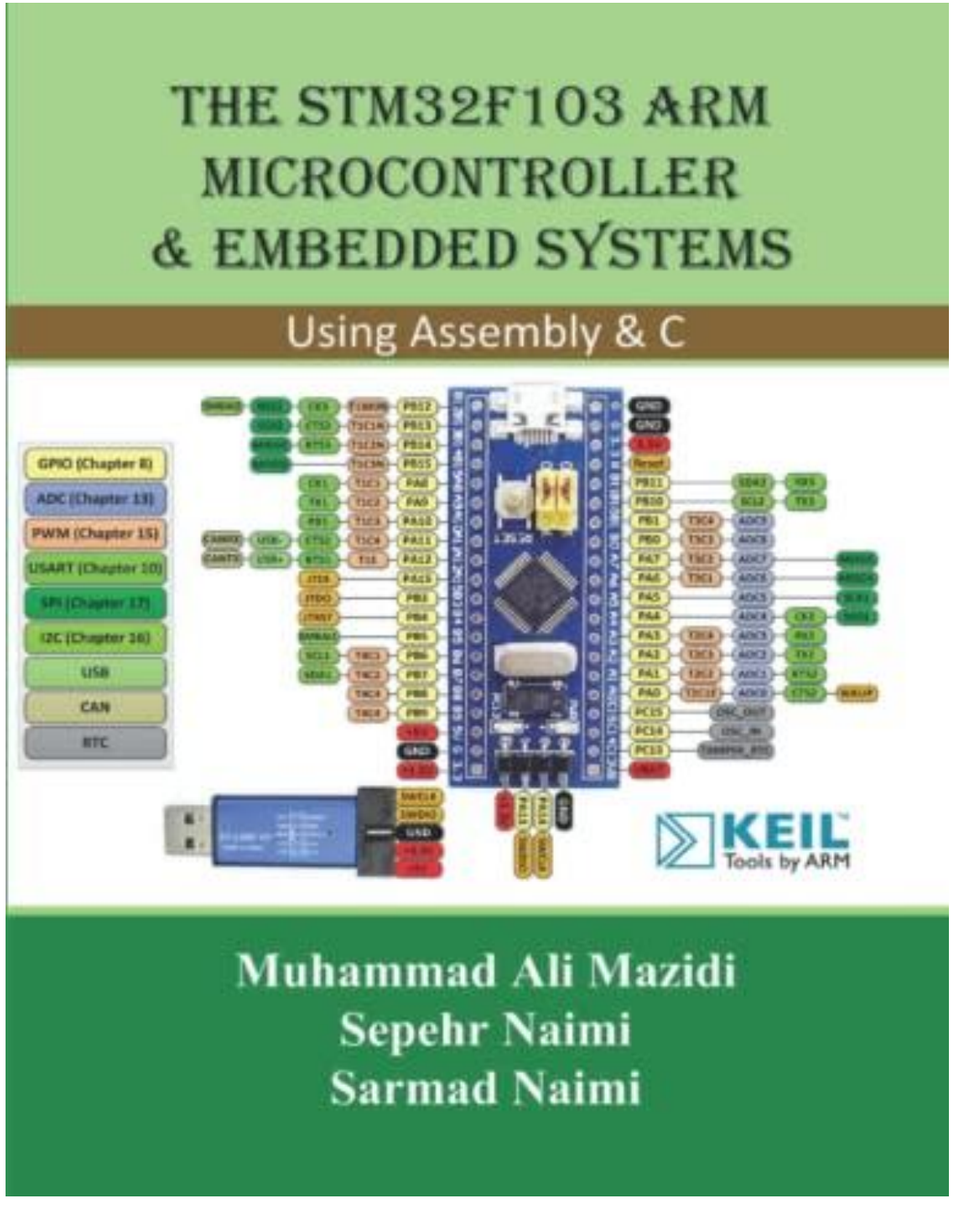


Lecture # 04M:

Switch Debouncing

Course Teacher: **Prof. Dr. Engr. Muhibul Haque Bhuyan**
Professor, Department of EEE
American International University-Bangladesh (AIUB)
Dhaka, Bangladesh

What is Debouncing?

When the state of a mechanical switch is changed from open to closed and vice versa, there is often a short period of time when the input voltage jumps between HIGH and LOW levels a couple of times before it settles. These transitions are called **bouncing** and getting rid of the effect of these transitions is called **debouncing**. To apply a manual switch signal into a digital circuit, we need to **debounce the signal,** so a single press doesn't appear as multiple presses.

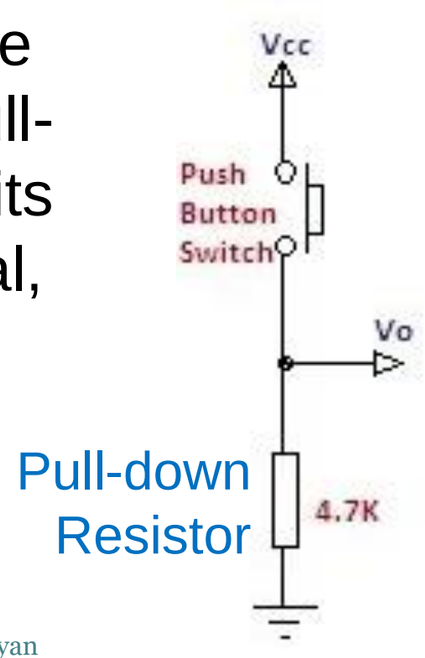
Alternatively, when a mechanical switch is flipped/pressed, the metal contacts inside may not open or close cleanly. In the microseconds before the switch achieves a **good solid connection**, the switch's contacts may "**bounce**" against each other, turning the **switch on and off** in rapid succession.

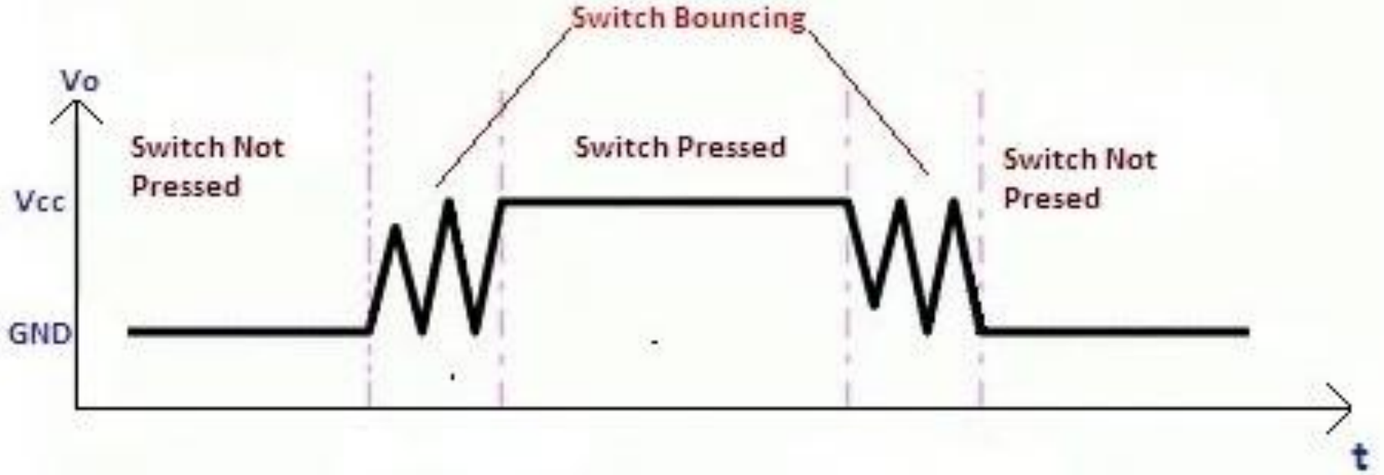
Example 1: If we connect the switch button to a pin with an external interrupt enabled, we will get several interrupts from pressing the button just once. This behavior causes multiple false button presses.
Example 2: Imagine using a button in a TV remote controller for the selection of a channel. If the button is not being debounced, one press can cause the remote to skip one or more channels.

Why does Bouncing Happen?

- Bouncing is a result of the **physical property of mechanical switches.**
- Switch and relay contacts are usually made of metallic springs so when a switch is pressed, it is essential that two metal parts come together. But, this does not happen immediately, it bounces between in-contact (close the contact) and not-in-contact (open the contact) until it finally settles down.

Figure shows a simple push switch with a pull-down resistor and at its right, the output signal, V_{out} shows the signal bounces when the switch is pressed or depressed.

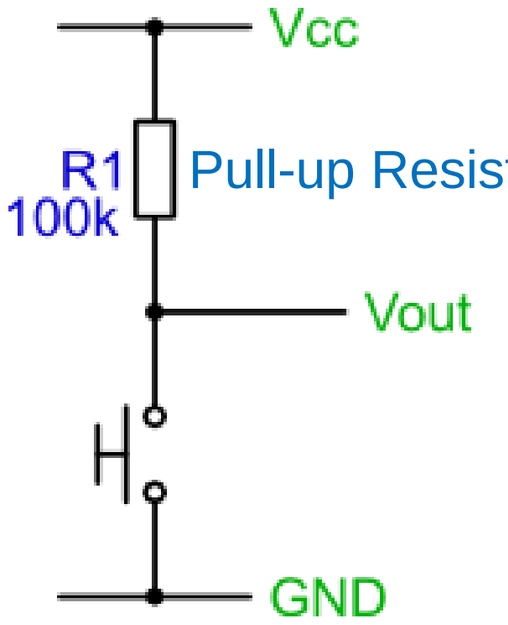


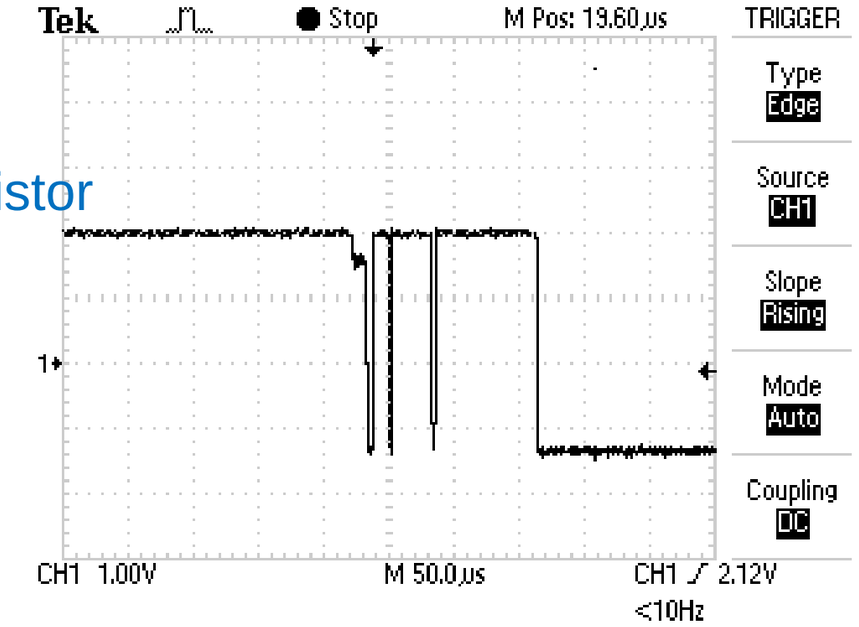


Why does Bouncing Happen?

- Figure shows a simple push switch with a pull-up resistor and at its right side, the trace shows the output signal, V_{out}, when the switch is pressed. As can be seen, pressing the switch does not provide a clean edge. If this signal is used as an input to a digital counter, for example, you will get multiple counts rather than the expected single count.

- The same can also occur on the release of a switch.
- The problem is that the contacts within the switch don't make contact cleanly, but slightly 'bounce'. The bounce is quite slow, so you can recreate the trace, and the problem quite easily.





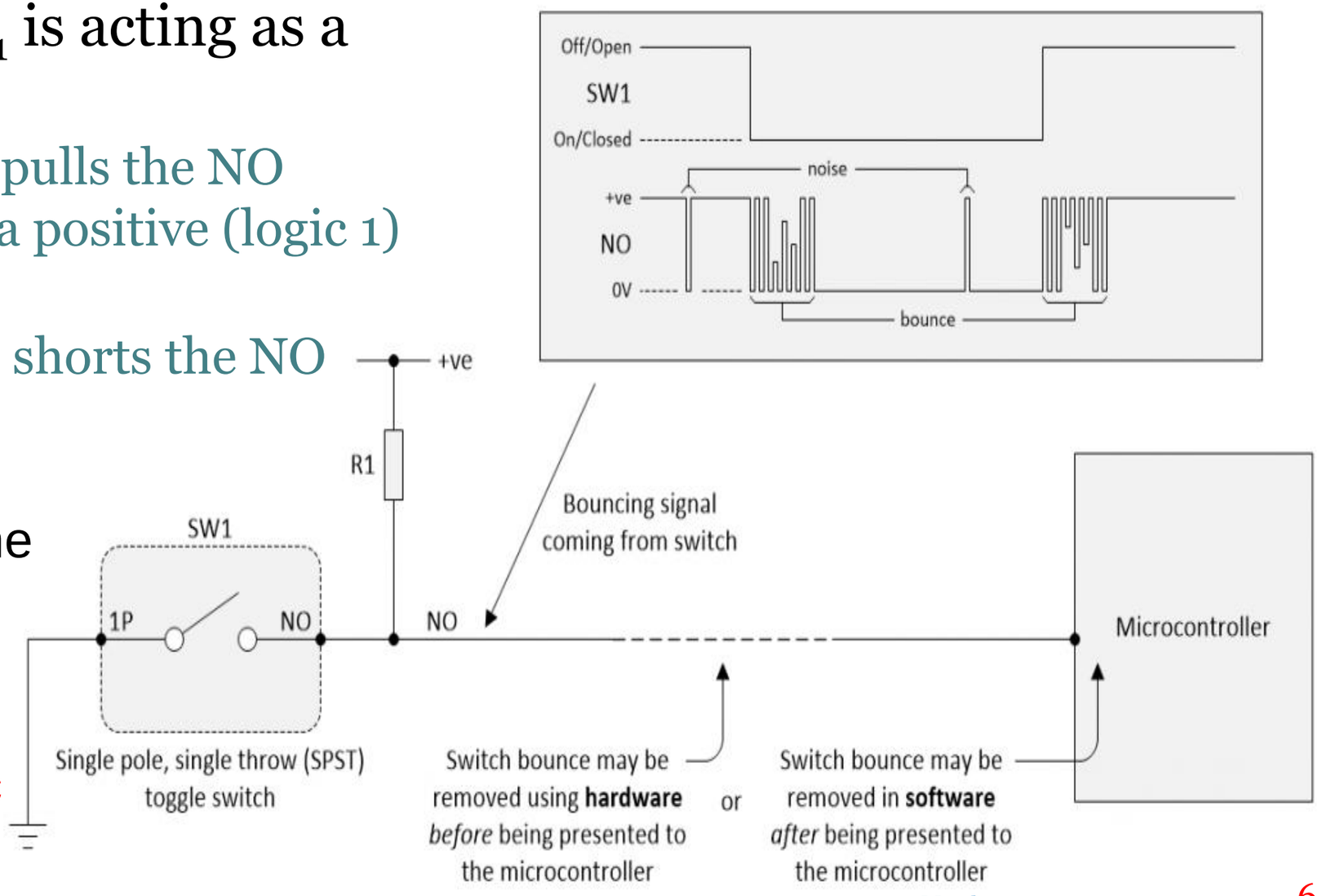
Why does Bouncing Happen?

- The bouncing phenomenon is associated with the **Single Pole Single Throw (SPST)**, **Single Pole Double Throw (SPDT)**, and similar other types of toggle switches. The **bouncing case data may be analyzed** to see **how long** the switch **bounce persists**, **how wide** the individual **bounce pulses** are, and **how many bounces** are observed.
- It is essential that the technique is not fooled by the occasional noise "glitch" or "spike" caused by other electrical, electronic, or electromagnetic signals, such as
  - Crosstalk,
  - EMI (electromagnetic interference)
  - RFI (radio frequency interference)
  - ESD (electrostatic discharge).

How does Bouncing Happen?

- In this diagram, resistor R_1 is acting as a pull-up resistor.
  - √When the switch is open, R_1 pulls the NO (Normally Open) contact to a positive (logic 1) value, usually +5 V.
  - √When the switch is closed, it shorts the NO contact to 0 V (logic 0).

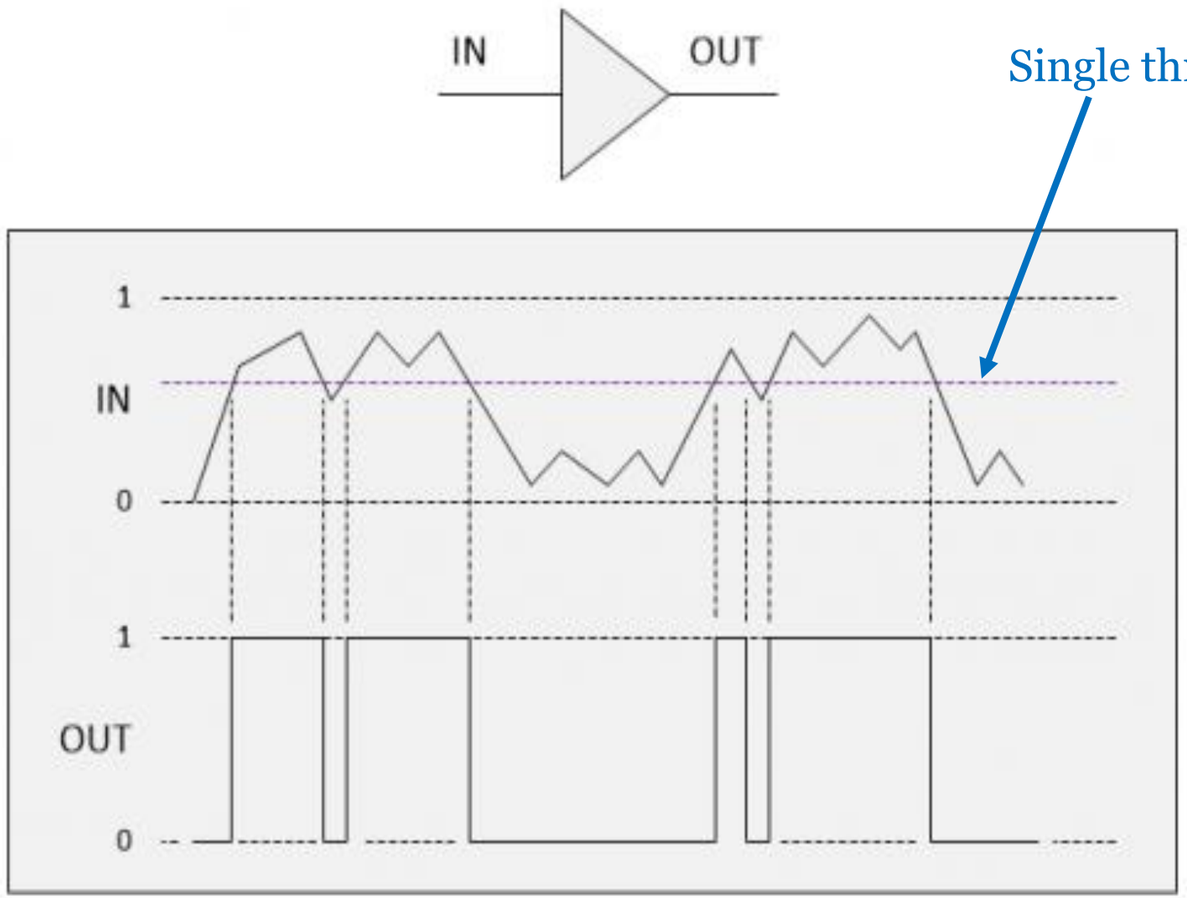
We intend to get an output voltage of 5 V (logic 1) when the switch is not pressed, and 0 V (logic 0) when it is pressed. But before the signal settles, it bounces between the two logic states several times.



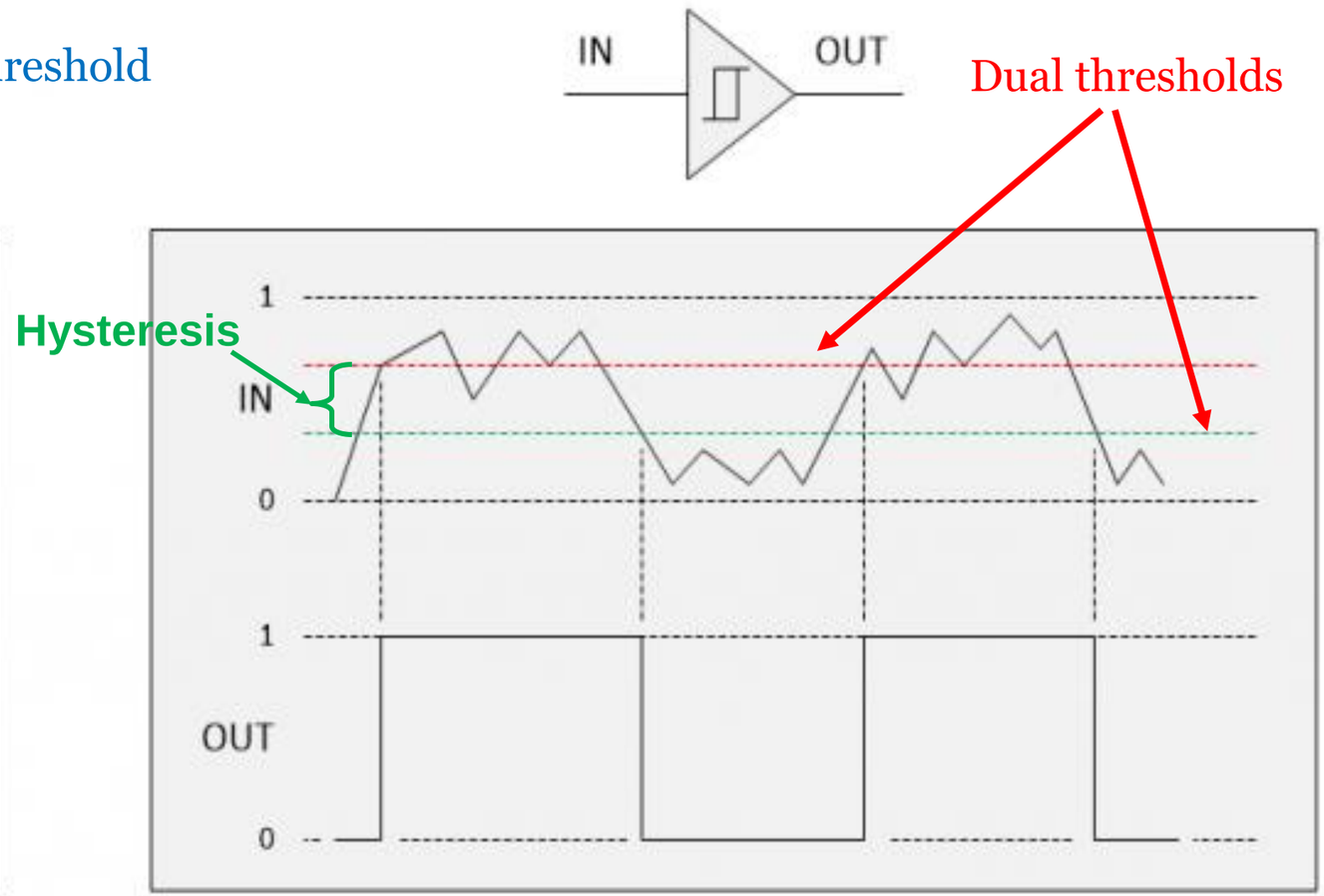
How to Avoid Bouncing?

- The digital circuits and microcontrollers don't like its input signals wandering around the **undefined region** between a "good" logic 0 and a "good" logic 1.
- A standard buffer has **single switching threshold**, so a noisy signal or a signal with a bit of ripples on it, can result in multiple transitions at the output.
- In 1934, a young graduate student named **Otto Herbert Schmitt** invented a circuit known as the **Schmitt trigger** (this invention was a result of Otto's study into the propagation of neural impulses in the nerves of squids- a kind of sea fish) that has **two thresholds**. The output changes only when the input crosses the upper threshold (during upward transition) or the lower threshold (during downward transition) . This **dual-threshold action** is called **hysteresis** (the dependence of the state of a system on its history), which implies that the trigger possesses **memory**.

How to Avoid Bouncing?



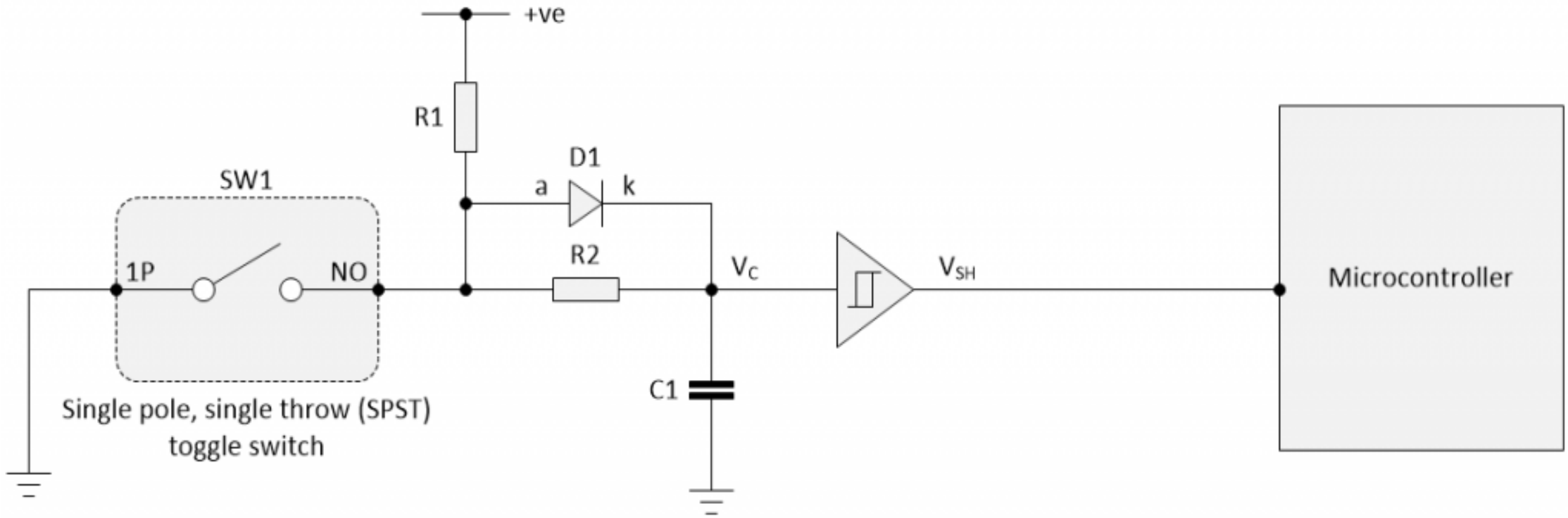
Standard buffer with single threshold



Schmitt trigger buffer with upper and lower thresholds

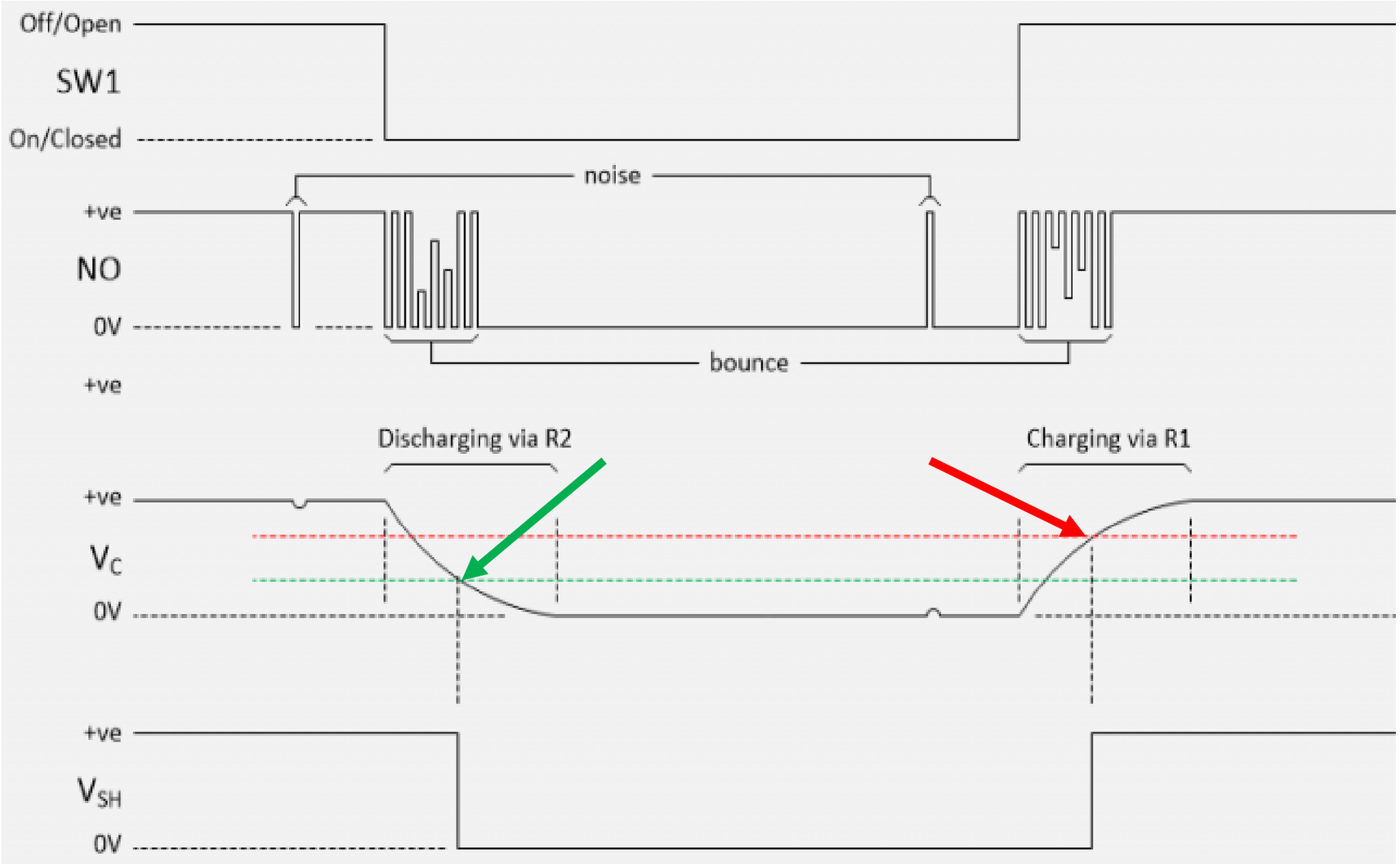
How to Avoid Bouncing?

- In the RC network, we use a simple SPST switch debounce circuit, we would add a **Schmitt trigger buffer** between the RC network and the microcontroller as illustrated below.



How to Avoid Bouncing?

- We have shown a **non-inverting Schmitt trigger buffer** to keep things simple.
- It is common to use an **inverting Schmitt trigger buffer** because inverting functions are faster than their non-inverting counterparts.



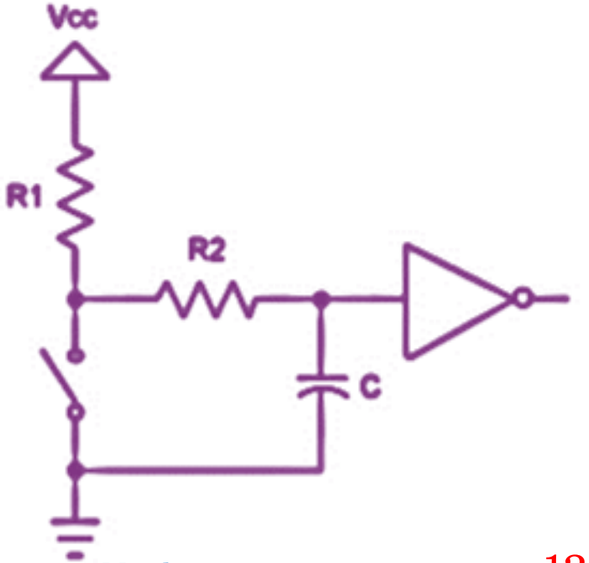
Types of Debouncing

- We can remedy this problem by debouncing the switch. There are many ways to do this. There are **two general ways** of getting rid of the bounces:
  - **Hardware debouncing:** adding circuitry that gets rid of the unwanted transitions.
  - **Software debouncing:** adding code/program that causes the application to ignore the bounces.
- Besides, we can debounce the signals using the following techniques as well
  - **Digital Switch Debouncing:** achieved in the same way as the software approaches
  - **Switch Debouncer using VHDL:** The VHDL entity description is used.

Hardware Debouncing

- There are various implementations of circuits that can be used for eliminating the effect of switch debouncing right at the **hardware level**.
- One of the most popular solutions using hardware is to employ a **resistor/capacitor network** with a **time constant (τ)** longer than the time it takes the bouncing contacts to stop, i.e., $\tau > t_{bounce}$.
- Often it is also followed by a Schmitt trigger to get rid of the capacitor voltage values midway between **HIGH** and **LOW** logic.

- If we wish to **preserve program execution cycles**, it is the **best way** to use the **hardware debouncing** approach.
- In hardware debouncing, we may use a flip-flop to 'latch' the signal, using a capacitor to **absorb the bouncing periods**.

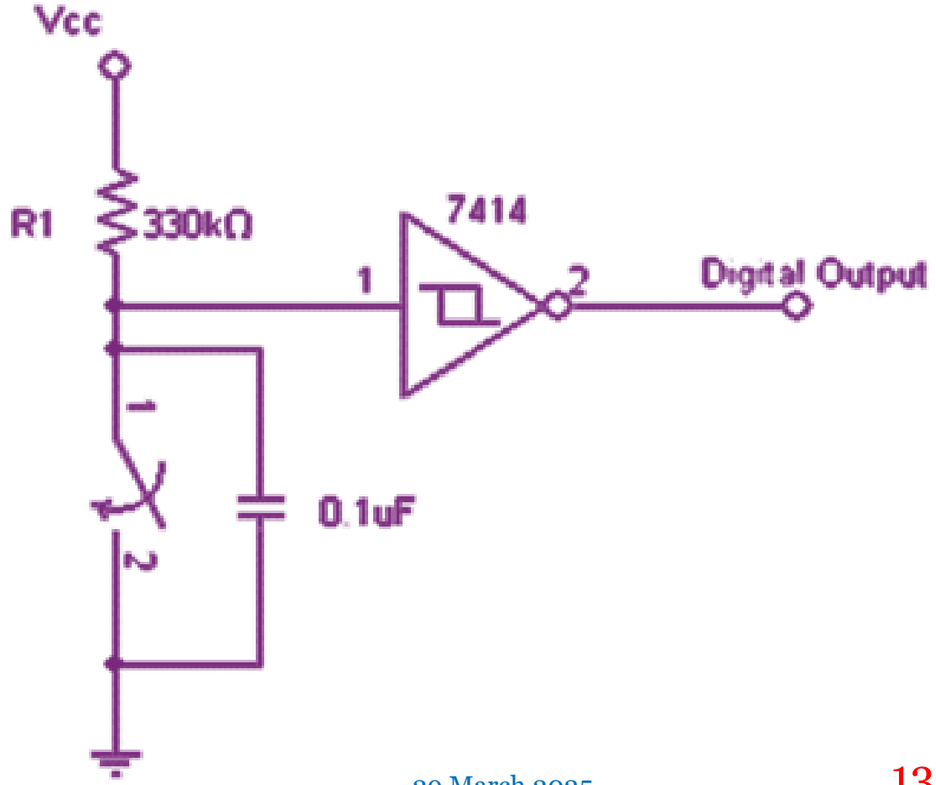


Hardware Debouncing Circuit

- The following circuit is a simple hardware debouncing circuit with a switch for digital logic. When the switch is closed, the voltage is pulled down LOW through the switch, and when the switch is open, the output voltage is pulled up HIGH through the resistors *R*1 and *R*2. The resistor *R*2 prevents a short circuit when the switch is closed.

- The capacitor absorbs the bounces when the switch changes states. Keep in mind that this circuit will cause a short delay in switching because the capacitor takes time to charge and discharge through the resistors.

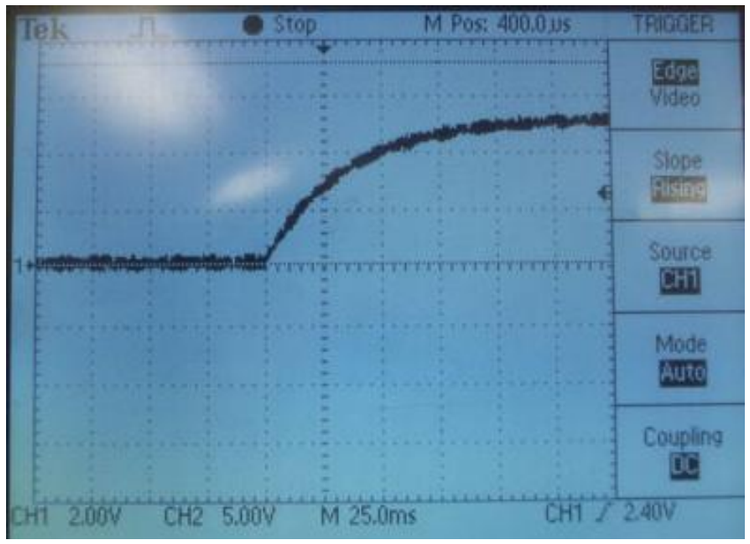
- The delay can be minimized by varying the values of the capacitor and resistor, but if the resistor is too small, it may not be sufficient to absorb the bounces.



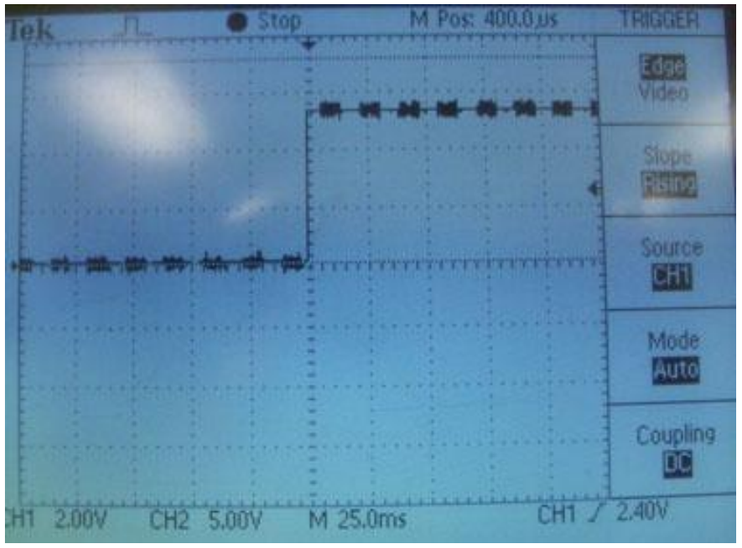
Hardware Debouncing Circuit

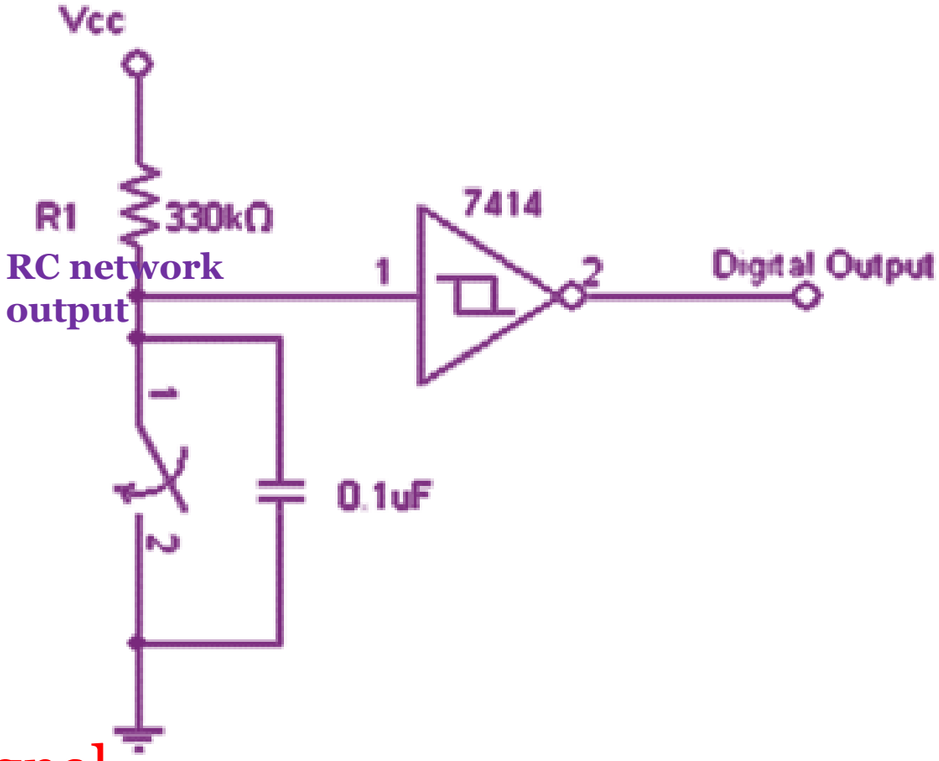
It is to be noted that the capacitor exhibits a curve when charging, and parts of this curve fall in the region where the logic circuit is undecided whether to treat it as a HIGH or LOW signals. To fix this problem, a **Schmitt trigger** is added to the output of the RC network.

Without Schmitt trigger RC network output



With Schmitt trigger digital output





Remember that the 7414 Schmitt trigger will invert the signal.

Hardware Debouncing

Circuit Design using the IC 7414 is a **Schmitt trigger (inverting type buffer)** with the **input hysteresis.**

A CMOS device like the 74AHCT14 dribbles about a μA from the inputs.

Resistances R_1 and R_2 control the capacitor's charging-discharging time and thus set the **debounce period** for the **switch open state**.

The equation for charging is:

$$v_c = V_{final}\left(1 - e^{-t/RC}\right)$$

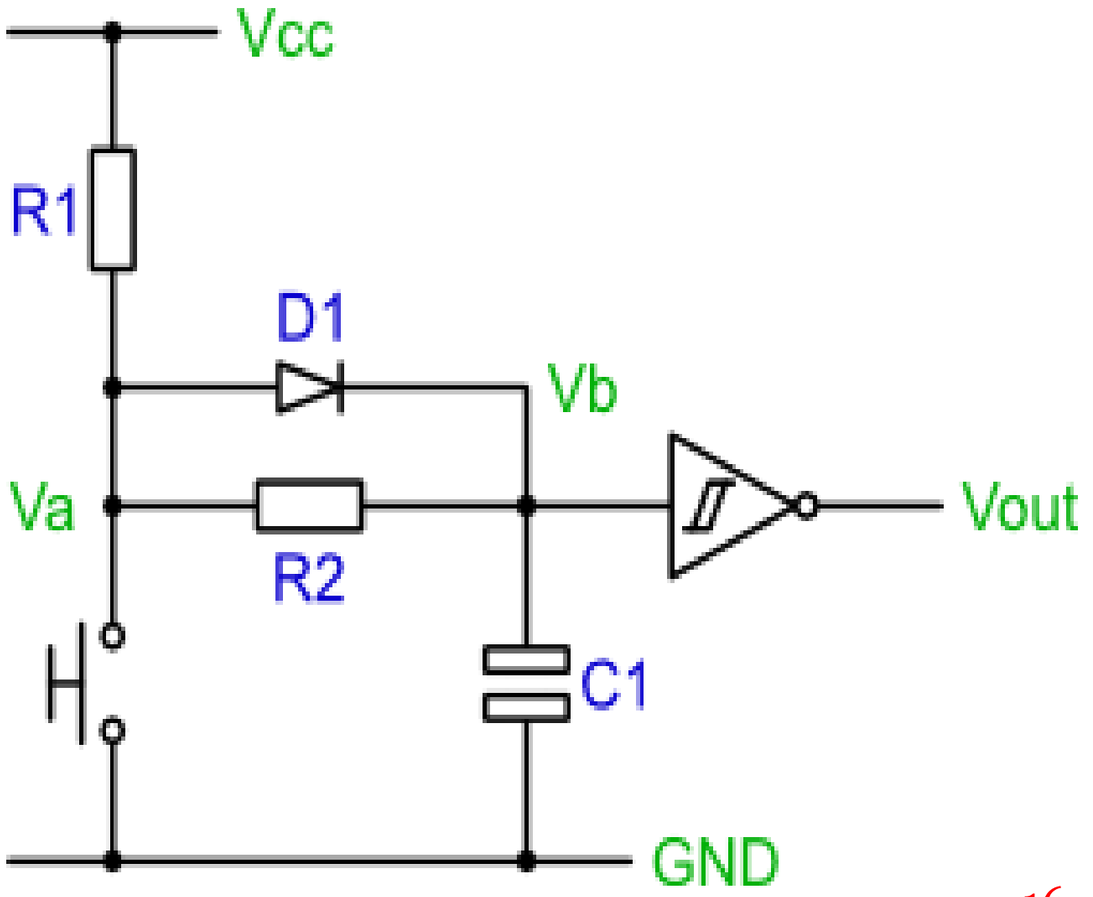
V_{final} is the voltage towards where the capacitor will reach after the capacitor is completely charged. This is the supply voltage, V_{CC} in this case.

Hardware Debouncing Circuit Design

The equation for discharging a capacitor is given by

$$v_c = v_{th} = V_{final} e^{-t/RC}$$

V_{final} is the voltage from where the capacitor starts to discharge after the capacitor is completely charged.



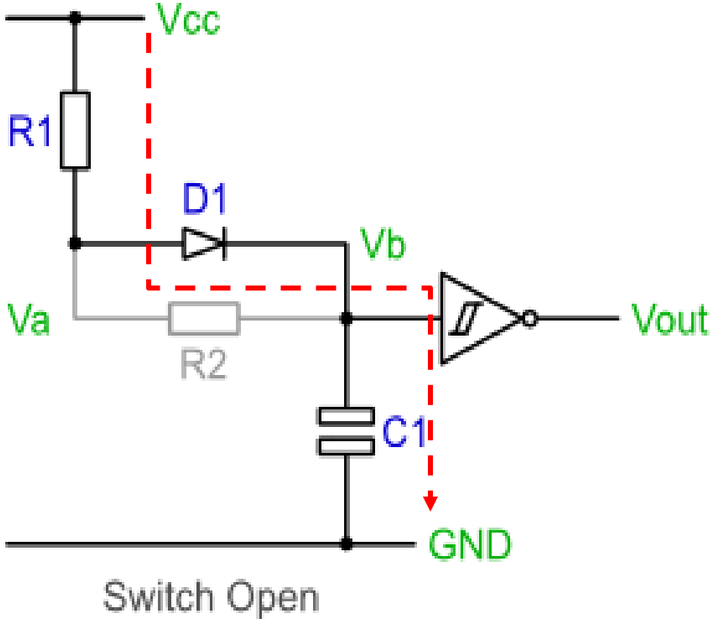
Debouncing circuit design using the IC 7414 Schmitt trigger (inverting type buffer)

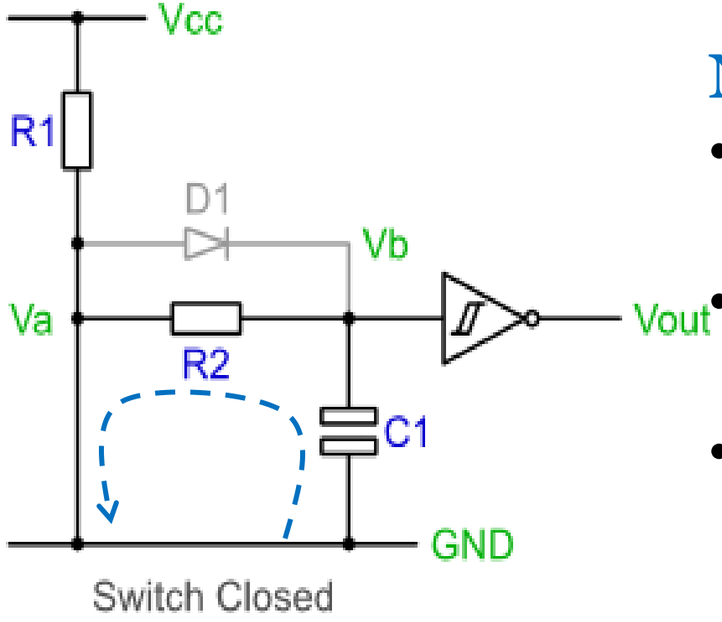
Hardware Debouncing Circuit Operation

The circuit's operation can be explained by looking at the equivalent circuits formed in the two switching states, e.g., open and closed.

Starting with the switch **OPEN**:

- The capacitor C_1 will **charge** via R_1 and D_1.
- In time, C_1 will charge and V_b will reach within 0. 7 V of V_{cc}.
- Therefore, the output of the inverting Schmitt trigger will be a logic 0.





Now, **CLOSE** the switch:

- The capacitor will **discharge** via R_2.
- In time, C_1 will discharge and V_b will reach 0 V.
- Therefore, the output of the inverting Schmitt trigger will be a logic 1.

Hardware Debouncing Circuit Design

What happens when the bounce occurs?

If bounce occurs and there are short periods of switch closure or opening, the capacitor will stop the voltage at V_b immediately reaching V_{cc} or GND. Although bouncing causes slight charging and discharging of the capacitor, the hysteresis of the Schmitt trigger input stops the output from switching.

Why is a diode used?

The resistor R_2 is required as a discharge path for the capacitor. Without it, C_1 will be shorted when the switch is closed. Without the diode, D_1, both R_1 and R_2 would form the capacitor's charging path when the switch is open. The combination of R_1 and R_2 increases the capacitor's charging time and thus slows down the circuit. With diode, when the switch is closed, R_1 limits the current flow and the diode bypasses R2 as it is forward biased.

Hardware Debouncing Circuit Design

The goal of the debouncing circuit design is to select R-C values that ensure the capacitor's voltage stays above the threshold voltage, V_{th} around which the signal switches till the switch stops bouncing.

Most of the switches exhibit **bounce times well under 10 ms**, so the use of this value would be a **conservative choice**.

To calculate the R-C values considering this bounce time, we rearrange the **capacitor's charging-discharging formula** as follows to solve for R (the cost and size of capacitors vary widely so it is the best option to select a value for C arbitrarily and then compute R)

$$v_c = v_{th,L} = V_{final}\left(1 - e^{-t/RC}\right)$$

$$R = -\frac{t}{C\ln\left(1 - \frac{V_{th,L}}{V_{final}}\right)}$$

$$v_c = v_{th,H} = V_{final}e^{-t/RC}$$

$$R = -\frac{t}{C\ln\frac{V_{th,H}}{V_{final}}}$$

Hardware Debouncing Circuit Design

Numerical Example:

The 74AHCT14 version has a **worst-case V_{th}** for a signal going low of 1.7 V, the higher threshold voltage, $V_{th,H}$. This is used for discharging time to remain in the **safe side** while computation of the resistance value.

Let us try with a 0.1 μF capacitor, since this is small, cheap, and solve for the conditions in which the switch is closed.

We start our computation using the capacitor discharge formula. Since the capacitor discharges through R_2, from this equation we can get its value. We consider, the power supply as 5 V (i.e., V_{final} = 5 V at the start of discharging).

$$R_2 = -\frac{t}{C ln \frac{V_{th,H}}{V_{final}}} = -\frac{20 \times 10^{-3}}{0.1 \times 10^{-6} ln \frac{1.7}{5}} = 185\ \text{k}\Omega$$

Considering bounce times as 20 ms

Since a resistor with this value is not available, so we use 180 kΩ.

Hardware Debouncing Circuit Design

Numerical Example (continuing):

But the analysis ignores the gate's input leakage current when the signal is **LOW** at its input. A CMOS device like 74AHCT14 dribbles about 1 µA from the inputs.

As such, a 180 kΩ resistor may bias the input to 0.18 V (since as per Ohm's Law, $V = IR = 1 \times 10^{-6}\text{A} \times 180 \times 10^{3}\text{V} = 0.18\,\text{V}$), uncomfortably close to the gate's best-case switching point of 0.5 V.

Therefore, the capacitor's capacitance is increased to 1 µF from 0.1 µF and hence the resistor value decreases 10 times. So, we can use 18 kΩ for R_2.

Hardware Debouncing

Numerical Example (continuing):

$R_1 + R_2$ controls the capacitor's charging time and so sets the debounce period for the condition when the switch opens. The equation for charging is:

$$v_c = V_{final}\left(1 - e^{-t/RC}\right)$$

Solving for R:

$$R = R_1 + R_2 = -\frac{t}{C ln\left(1 - \frac{V_{th,L}}{V_{final}}\right)}$$

V_{final} is the final charged value of the 5 V power supply. V_{th} is the worst-case transition point for a high-going signal, which for our 74AHCT14 is 0.9 V. So,

$R_1 + R_2 = -\frac{20\times10^{-3}}{1\times10^{-6} ln\left(1-\frac{0.9}{5}\right)} = 100.78\text{k}\Omega$ (if no diode is used R2 won't be shorted during charging). Since, we have calculated R_2 = 18 kΩ, hence, R_1 = 82 kΩ.

Hardware Debouncing

Numerical Example (continuing):

The diode is an optional part. If we use a diode across $R2$ then it removes R_2 from the charging circuit. All the charging current flows through the R_1.

Equations will be the same except that we must take the voltage drop across the diode into account. Change V_{final} to 4.3 V (5 minus the 0.7 V of diode's forward voltage drop, V_D) and solve for R_1 and R_2.

New computations are as follows:

$$R_2 = -\frac{t}{Cln\frac{V_{th,H}}{V_{final}}} = -\frac{20\times10^{-3}}{1\times10^{-6}ln\frac{1.7}{4.3}} = 21.55\,\text{k}\Omega \cong 22\text{k}\Omega$$

Considering bounce times as 20 ms and capacitance value as 1 μF

$$R_1 == -\frac{t}{Cln\left(1-\frac{V_{th,L}}{V_{final}}\right)} = -\frac{20\times10^{-3}}{1\times10^{-6}ln\left(1-\frac{0.9}{4.3}\right)} = 85.165\text{k}\Omega \cong 85\text{k}\Omega$$

Hardware Debouncing

Numerical Example (continuing):

Be wary of the components' tolerances!

Standard resistors are usually ±5 (based on silver band as the 4th band on the resistor) to ±10% (based on gold band as the 4th band on the resistor) and the capacitors are ±20% (for electrolytic) to ±30% (for small ceramic capacitors) are error prone from their nominal rating.

For example, if a resistor has nominal resistance value of 180 kΩ, and its 4th color band is silver color then the resistance value may vary between (180kΩ ± 5% of 180kΩ) i.e., from (180kΩ - 5% of 180kΩ) to (180kΩ + 5% of 180kΩ) or from (180 kΩ - 9kΩ) to (180 kΩ + 9kΩ) or from 171kΩ to 189kΩ.

Example 2 on Debouncing

A TV remote controller is to be designed with the channel selection buttons to be debounced. It was observed that the switches exhibit bounce times well under 20 ms. **Design** a circuit using the 74HC14 Schmitt trigger IC along with the resistance and capacitance. The worst-case V_{th} of this IC for a signal going low is 2.5 V and that of when going high is 0.8 V. Also, consider that the CMOS device leakage current is 10 μA and the gate's best-case switching point is of the order of 0.4 V. **Compute** the hysteresis voltage.

Answer:

While falling the signal, we choose a value of capacitance, C = 1 μF and use $v_c = v_{th,H} = V_{final}e^{-t/RC}$.

After rearranging the equation, we get $R = -\frac{t}{C ln\frac{V_{th}}{V_{final}}}$, and then putting the values, we find

$$R_2 = \frac{-20 \times 10^{-3}}{1 \times 10^{-6} ln\frac{2.5}{5}} = 28.85\ \text{k}\Omega\ \cong 27\ \text{k}\Omega$$

For TTL IC family, we usually choose supply voltage, V_{CC} = 5 V.

Example 2 on Debouncing

If we choose this value, therefore, during discharging, due to leakage current of 10 μA, a voltage drops of $V = IR = 10 \times 10^{-6} \times 27 \times 10^{3} = 0.27$ V would occur, and it is comfortably well over the gate's best-case switching point of 0.4 V. Therefore, we can increase this capacitor's capacitance to 10 μF so that the resistance of discharging path becomes 2.7 kΩ and as such the voltage drop becomes 0.027 V.

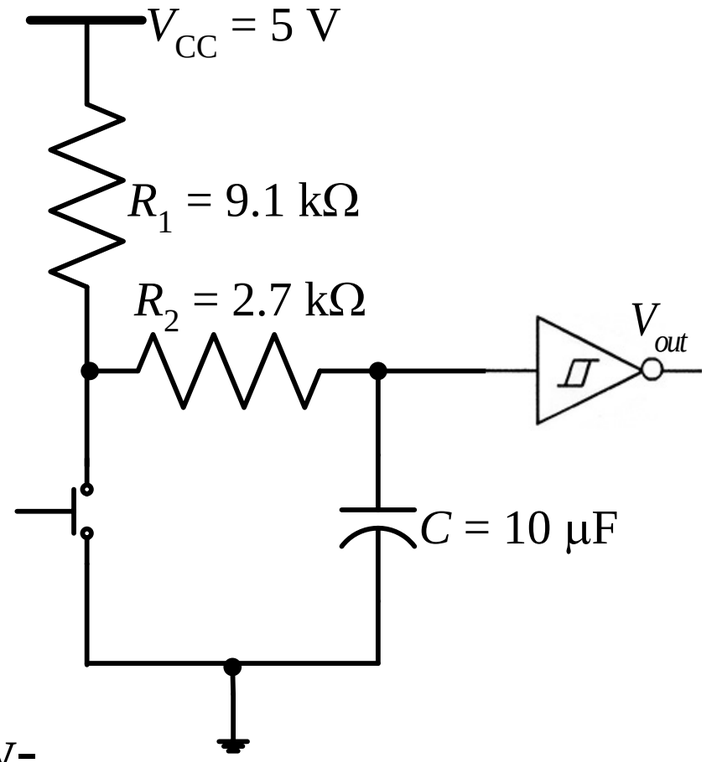
Again, during rising the signal, we choose this capacitance value of C = 10 μF and use capacitor's charging equation, $V_c = v_{th,L} = V_{final}\left(1 - e^{-t/RC}\right)$.

After rearranging the equation, we get $R = -\dfrac{t}{Cln\left(1-\frac{V_{th,L}}{V_{final}}\right)}$, and then putting the values, we find-

$$R_1 + R_2 = \frac{-20 \times 10^{-3}}{10 \times 10^{-6} ln\left(1 - \frac{0.8}{5}\right)} = 11.471\ \text{k}\Omega\ \cong 12\ \text{k}\Omega$$

Example 2 on Debouncing

Since, already we have calculated R_2 = 2.7 kΩ, hence, R_1 = 12 – 2.7 = 9.3 kΩ. But these values are not available, therefore we will use 9.1 kΩ for R_1 and 2.7 kΩ for R_2. Therefore, the final designed circuit is given as follows:



The hysteresis voltage is given by-

$$V_H = V_{th,H} - V_{th,L} = 2.5 - 0.8 = 1.7\ V$$

Example 3 on Debouncing

A TV remote controller is to be designed with the channel selection buttons to be debounced. It was observed that the switches exhibit bounce times well under 10 ms. A **Si diode** is used across the second resistor to make the duty cycle 50%. **Design** a circuit using the 74HC14 Schmitt trigger IC along with the resistance and capacitance. The worst-case V_{th} of 74HC14 for a signal going low is 2.3 V and that of when going high is 0.95 V. Consider that the CMOS device leakage current is 5 μA and the gate's best-case switching point is of the order of 0.5 V. **Compute** the hysteresis voltage.

Answer:

While falling the signal, we choose a value of capacitance, C = 1 μF and use $v_c = v_{th,H} = V_{final}e^{-t/RC}$.

After rearranging the equation, we get $R = -\frac{t}{Cln\frac{V_{th}}{V_{final}}}$, and then putting the values, we find-

$$R_2 = \frac{-10 \times 10^{-3}}{1 \times 10^{-6} ln\frac{2.3}{5-0.7}} = 15.98\ \text{k}\Omega\ \cong 15\ \text{k}\Omega$$

For TTL IC family, we usually choose supply voltage, V_{CC} = 5 V.

Assuming Si diode's forward voltage drop, V_D = 0.7 V.

Example 3 on Debouncing

If we choose this value, therefore, during discharging, due to leakage current of 5 μA, a voltage drops of $V = IR = 5 \times 10^{-6} \times 15 \times 10^{3} = 0.075$ V would occur, and it is comfortably well below the gate's best-case switching point of 0.5 V. Therefore, we use this capacitor's capacitance of 1 μF so that the resistance of discharging path becomes 15 kΩ and as such the voltage drop becomes 0.075 V.

Again, during rising the signal, we choose this capacitance value of C = 1 μF and use this in capacitor's charging equation, $V_c = v_{th,L} = V_{final}\left(1 - e^{-t/RC}\right)$.

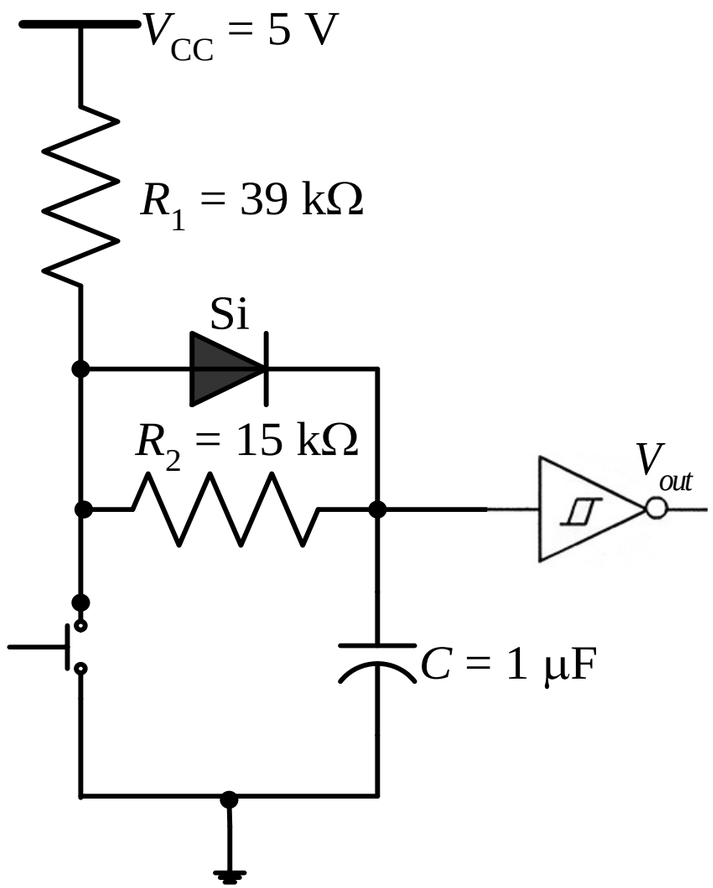
After rearranging the equation, we get $R = -\dfrac{t}{C ln\left(1 - \frac{V_{th,L}}{V_{final}}\right)}$, and then putting the values, we find-

$$R_1 = \frac{-10 \times 10^{-3}}{1 \times 10^{-6} ln\left(1 - \frac{0.95}{5 - 0.7}\right)} = 40.05\ \text{k}\Omega\ \cong 39\ \text{k}\Omega$$

Neglecting a very small diode's forward resistance.

Example 3 on Debouncing

So, our calculated values for the designed circuit are R_1 = 39 kΩ, R_2 = 15 kΩ, and C = 1 μF. Therefore, the final designed circuit is given as follows:



The hysteresis voltage is given by-

$$V_H = V_{th,H} - V_{th,L} = 2.3 - 0.95 = 1.35\ V$$

Software Debouncing

- Debouncing in hardware may give rise to additional cost, and it is more difficult to determine a good debouncing for all the push button switches that will be used. So, it may be preferable to debounce the switch in software.
- Debouncing a switch in software is very simple. The basic idea is to sample the switch signal at a regular interval and filter out any glitches. There are a couple of approaches to achieving this. These approaches assume a switch circuit like that shown in the explanation of switch bounce- a simple push switch with a pull-up resistor.
- While numerous algorithms exist to perform the debouncing function, we are going to limit ourselves to implementing two during the lab. Both approaches use a timer.

Software Debouncing

Approach 1

The first approach uses a counter to time how long the switch signal has been low. If the signal has been low continuously for a set amount of time, then it is considered pressed and stable.

```
1 Setup a counter variable, initialized to zero.
2 Setup a regular sampling event, perhaps using a timer. Use a period of about 1 ms.
3 On a sample event:
4   if the switch signal is HIGH then
5     RESET the counter variable to ZERO
6     SET internal switch state to released
7   else
8     Increment the counter variable to a maximum of 10
9   end if
10   if counter=10 then
11    SET internal switch state to pressed
12   end if
```

Software Debouncing

Approach 2

The second approach is like the first but uses a shift register instead of a counter. The algorithm assumes an **unsigned 8-bit register value**, such as that found in 8-bit microcontrollers.

```
1 Set up a variable to act as a shift register and initialize it to 0xFF.
2 Set up a regular sampling event, perhaps using a timer. Use a period of about 1 ms.
3 On a sample event:
4   SHIFT the variable towards the most significant bit
5   SET the least significant bit to the current switch value
6   if shift register value = 0 then
7     SET internal switch state to pressed
8   else
9     SET internal switch state to released
10  end if
```

Software Debouncing

- There are **two more approaches to software debouncing**.
- In the first technique, wait for a switch closure, then test the switch again after a **short delay** (**15 ms** or so). If it is still closed, detect that the switch has changed state.
- In the second technique, **check the switch state periodically** to see if it **has changed its state**.
- When working with microcontrollers, we can deal with switch bounce in a different way that may save both hardware, space, and money. Some programmers do not care much about bouncing switches and just add a 50 ms delay after the first bounce. This forces the microcontroller to wait 50 ms for the bouncing to stop, and then continue with the program. This is not a good practice, as it keeps the microcontroller waiting.
- Another way is to use an interrupt for handling the switch bounce. Be aware that the interrupt might be fired on both the rising and falling edge, and some microcontrollers might stack up one waiting interrupt.

Software Debouncing

The following is a simple software debounce code for Arduino written in IDE

```
int inPin = 7;         // the pin number of the input pin
int outPin = 13;       // the pin number of the output pin

int counter = 0;       // how many times we have seen new value
int reading;           // the current value read from the input pin
int current_state = LOW;    // the debounced input value

// the following variable is a long because the time, measured in milliseconds,
// will quickly become a bigger number than can be stored in an int.
long time = 0;         // the last time the output pin was sampled
int debounce_count = 10; // number of millisecond/samples to consider before declaring a debounced input

void setup()
{
  pinMode(inPin, INPUT);
  pinMode(outPin, OUTPUT);
  digitalWrite(outPin, current_state); // setup the Output LED for initial state
}
```

```
void loop()
{
 // If we have gone on to the next millisecond
 if(millis() != time)
 {
  reading = digitalRead(inPin);

  if(reading == current_state && counter > 0)
  {
   counter--;
  }
  if(reading != current_state)
  {
    counter++;
  }
  // If the Input has shown the same value for long enough let's switch it
  if(counter >= debounce_count)
  {
   counter = 0;
   current_state = reading;
   digitalWrite(outPin, current_state);
  }
  time = millis();
 }
}
```

millis () function:

This function is used to return the number of milliseconds at the time, the Arduino board begins running the current program, that is, this Arduino function returns the present time in milliseconds from the moment the Arduino board is powered on or reset. The return value of millis is the number of milliseconds through an unsigned long variable since the program in Arduino started.

This number overflows i.e., goes back to zero after approximately 50 days.
millis() function Syntax. millis(); This function returns milliseconds from the start of the program.

Software Debouncing

- The following program toggles two LEDs connected to a PIC microcontroller.

```
// INCLUDES
#include <stdio.h>
#include <stdlib.h>
#include <xc.h>

// CONFIG
#pragma config FOSC = INTOSCIO  // Oscillator Selection bits (INTOSC oscillator: I/O function on RA6/OSC2/CLKOUT pin, I/O function on RA7/OSC1/CLKIN)
#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled)
#pragma config PWRTE = OFF      // Power-up Timer Enable bit (PWRT disabled)
#pragma config MCLRE = ON       // RA5/MCLR/VPP Pin Function Select bit (RA5/MCLR/VPP pin function is MCLR)
#pragma config BOREN = ON       // Brown-out Detect Enable bit (BOD enabled)
#pragma config LVP = ON         // Low-Voltage Programming Enable bit (RB4/PGM pin has PGM function, low-voltage programming enabled)
#pragma config CPD = OFF        // Data EE Memory Code Protection bit (Data memory code protection off)
#pragma config CP = OFF         // Flash Program Memory Code Protection bit (Code protection off)

// DEFINITIONS
#define _XTAL_FREQ 4000000
#define LED1 PORTBbits.RB3
#define LED2 PORTBbits.RB2
#define BTN PORTBbits.RB5
```

```
// VARIABLES
char BTN_pressed = 0;
char BTN_press = 0;
char BTN_release = 0;
char BounceValue = 500;

// MAIN PROGRAM
int main(int argc, char** argv) {
    // Comparators off
    CMCON = 0x07;

    // Port directions, RB5 input, the rest is output
    TRISA = 0b00000000;
    TRISB = 0b00100000;

    // Port state, all low
    PORTA = 0b00000000;
    PORTB = 0b00000000;

    // Starting with LED1 high and LED2 low
    LED1 = 1;
    LED2 = 0;
```

```
while (1)
  {
    // If BTN is pressed
    if (BTN == 1)
    {
      // Bouncing has started so increment BTN_press with 1, for each "high" bounce
      BTN_press++;
      // "reset" BTN_release
      BTN_release = 0;
      // If it bounces so much that BTN_press is greater than Bounce value then the button must be pressed
      if (BTN_press > Bouncevalue)
      {
        // This is initial value of BTN_pressed.
        // If program gets here, button must be pressed
        if (BTN_pressed == 0)
        {
          // Toggle the LEDs
          LED1 ^= 1;
          LED2 ^= 1;
          // Setting BTN_pressed to 1, ensuring that we will not enter this code block again
          BTN_pressed = 1;
        }
```

```
BTN_press = 0;
          }
       }
       else
       {
          // Increment the "low" in the bouncing
          BTN_release++;
          BTN_press = 0;
          // If BTN_release is greater than Bouncevalue, we do not have a pressed button
          if (BTN_release > Bouncevalue)
          {
             BTN_pressed = 0;
             BTN_release = 0;
          }
       }

    }
    return (EXIT_SUCCESS);
}
```

References

- ATMega328 manual
- http://www.ganssle.com/debouncing-pt2.htm
- https://mansfield-devine.com/speculatrix/2018/04/debouncing-fun-with-schmitt-triggers-and-capacitors/

Resistances in kilo Ohm range:

1kΩ	10kΩ	100kΩ
1.1kΩ	11kΩ	110kΩ
1.2kΩ	12kΩ	120kΩ
1.5kΩ	15kΩ	150kΩ
1.8kΩ	18kΩ	180kΩ
2kΩ	20kΩ	200kΩ
2.2kΩ	22kΩ	220kΩ
2.7kΩ	27kΩ	270kΩ
3.3kΩ	33kΩ	330kΩ
3.6kΩ	36kΩ	360kΩ
3.9kΩ	39kΩ	390kΩ
4.7kΩ	47kΩ	470kΩ
5.6kΩ	56kΩ	560kΩ
6.8kΩ	68kΩ	680kΩ
7.5kΩ	75kΩ	750kΩ
8.2kΩ	82kΩ	820kΩ
9.1kΩ	91kΩ	910kΩ

Thanks for Attending….