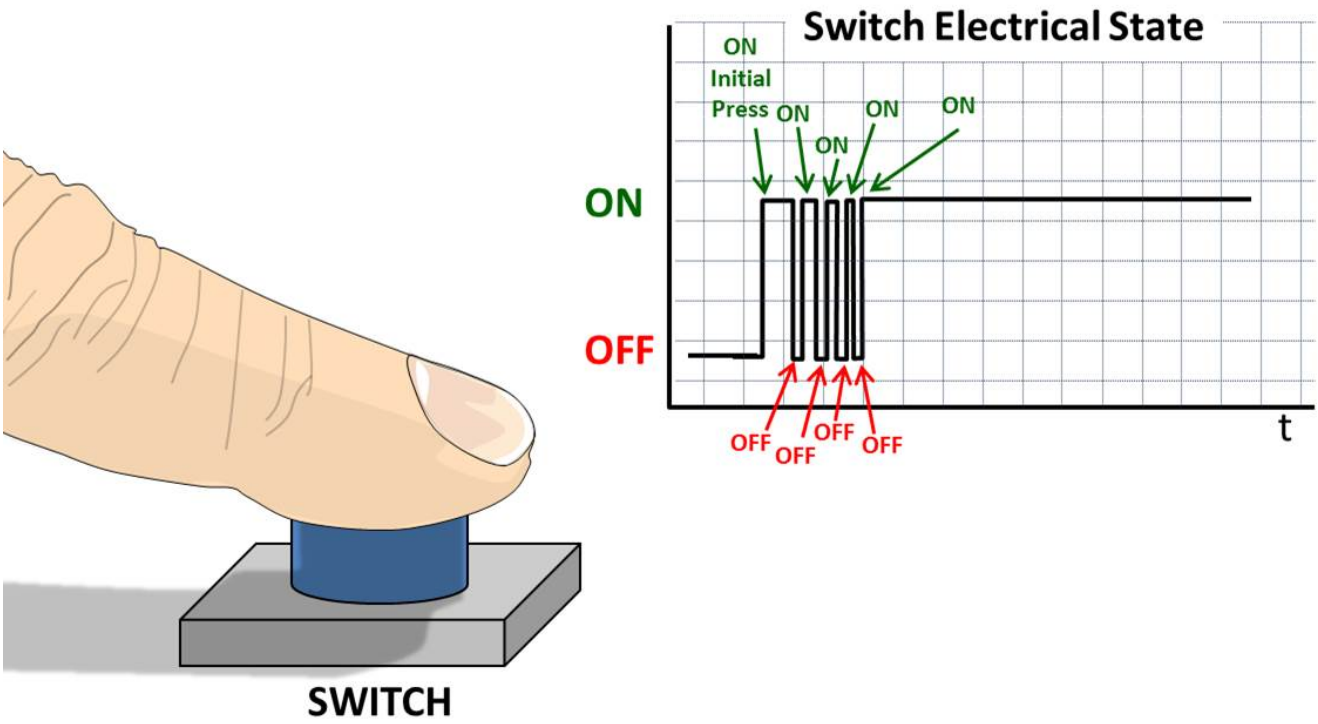
Debouncing considerations when reading input from switch

Mechanical switches and buttons can have bouncing or chatter when they are pressed or released. During this time, the electrical contacts inside the switch make and break contact rapidly before settling into a stable state. This bouncing can generate multiple pulses or transitions in a short period.

Debouncing is an essential consideration when reading digital input from switches, buttons, or other mechanical contacts in embedded systems or digital circuits. Failing to debounce inputs properly can lead to erratic behavior and false readings.



In order to reduce or eliminate the effect of bouncing, we can do Software or Hardware Debouncing:

- Hardware Debouncing: Hardware debouncing involves using external components like resistors and capacitors (RC networks) or specialized ICs (debouncers) to filter out the noise. This method is less CPU-intensive and can be more reliable for high-speed or noisy switches.
- Software Debouncing: In software debouncing, you use code to filter out noise and transitions that occur within the debounce time. This approach is flexible and can work with various switches, but it relies on the processing power of your microcontroller.

We can implement different debouncing Algorithms like:

- Simple Delay-Based Debouncing: In this approach, you can implement a simple delay-based debouncing algorithm using a timer or loop delay.
- State Machine Debouncing: In this approach, you maintain a state machine to track the current state of the switch over time. The state machine transitions only after a stable state has been detected for a

specified duration.

- Interrupt-Based Debouncing: You can use interrupts to detect switch transitions and then check the state of the switch after a debounce delay in an interrupt service routine (ISR).

For our example, we will use Simple Delay-Based Debouncing.

The debounce time is the duration for which you should ignore any changes in the input state after detecting a transition. The debounce time depends on the characteristics of your switch but typically ranges from a few milliseconds to tens of milliseconds. You should choose a debounce time that is long enough to handle the worst-case bounce for your specific switch.

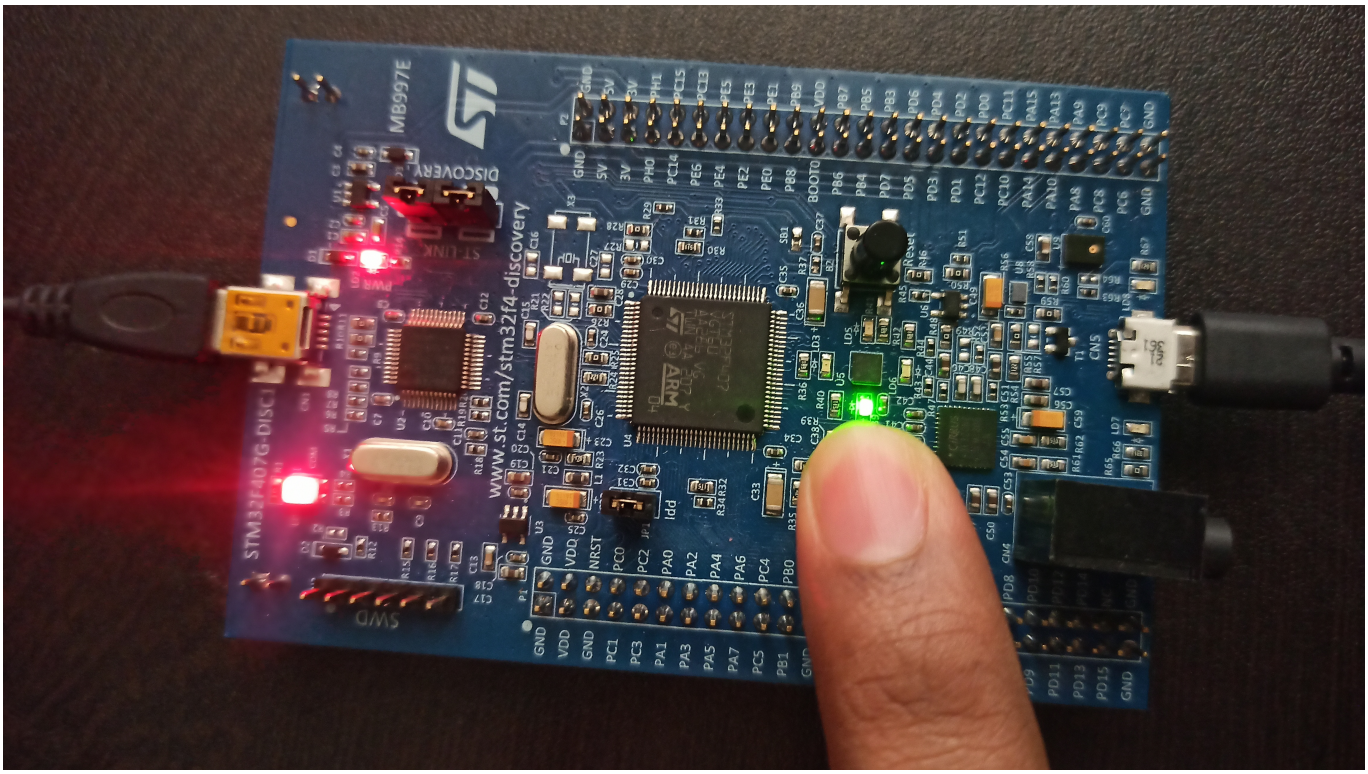
Lets is improve the project 1.2 code by adding debouncing logic. Create "switch_read_led_write_debounce.c" and add the below code.

```
#include "switch_read_led_write.h"

/*
 * Function to Read from GPIOA PIN 0 and Write to GPIOD PIN 12.
 */
void Read_GPIOA_PIN_0_Write_GPIOD_PIN_12(void)
{
    uint32_t status; /* variable to store the GPIO pin status */
    uint32_t lastState = 0; /* variable to store last the GPIO pin status */
    uint16_t debounceDelay = 500; /* Adjust this value based on your needs */

    /*
     * Read the pin status of GPIO port A and Pin 0
     * and store the same in status variable.
     */
    status = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);

    /* introduce the delay to let the switch status to settle */
    HAL_Delay(debounceDelay);

    lastState = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);
    /* Check if the switch state is consistent */
    if(status == lastState)
    {
        /* Check if pin status is SET or REST */
        if(status == PIN_SET)
        {
            /* Write the 1 (SET) to GPIOD pin 12 */
            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, PIN_SET);
        }
        else
        {
            /* Write the 0 (RESET) to GPIOD pin 12 */
            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, PIN_RESET);
        }
    }
}
```

We added debounce delay of "debounceDelay" HAL_Delay() function between two successive reads of switch status.

We should see Green LED is OFF when you push-button is not pressed and LED is ON when you push the the push-button. But this time you need to hold the button little long.

Feel free to change the value and experience how much time you need to keep holding the pushbutton to to reflect the its status on LED.



Source code is available at :
https://github.com/embeddedforallefa/embedded_sys_dev_with_stm32_code/tree/main/projects/EfA_STM32_dev/Core