Debouncing is a solution to the problem of contact bouncing that occurs when two metals come into contact (almost every mechanical switches).

When a button is pressed, the button's contacts "bounce off" before they fully contact each other, which in the case of microcontrollers can be interpreted as pressing the button several times.

Hardware solution for bouncing is simple circuit with resistors and capacitor, as shown in figure 1.

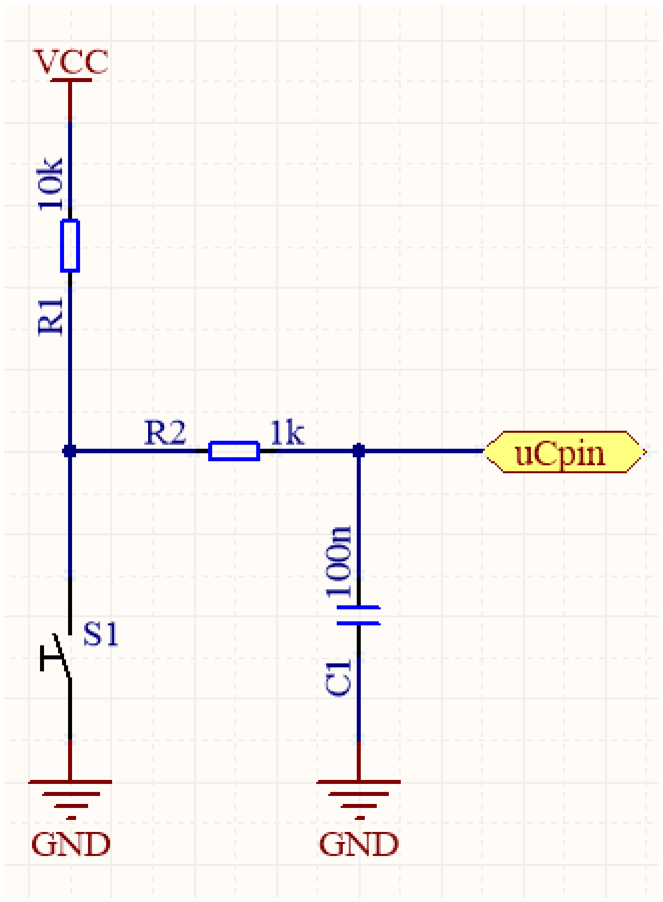
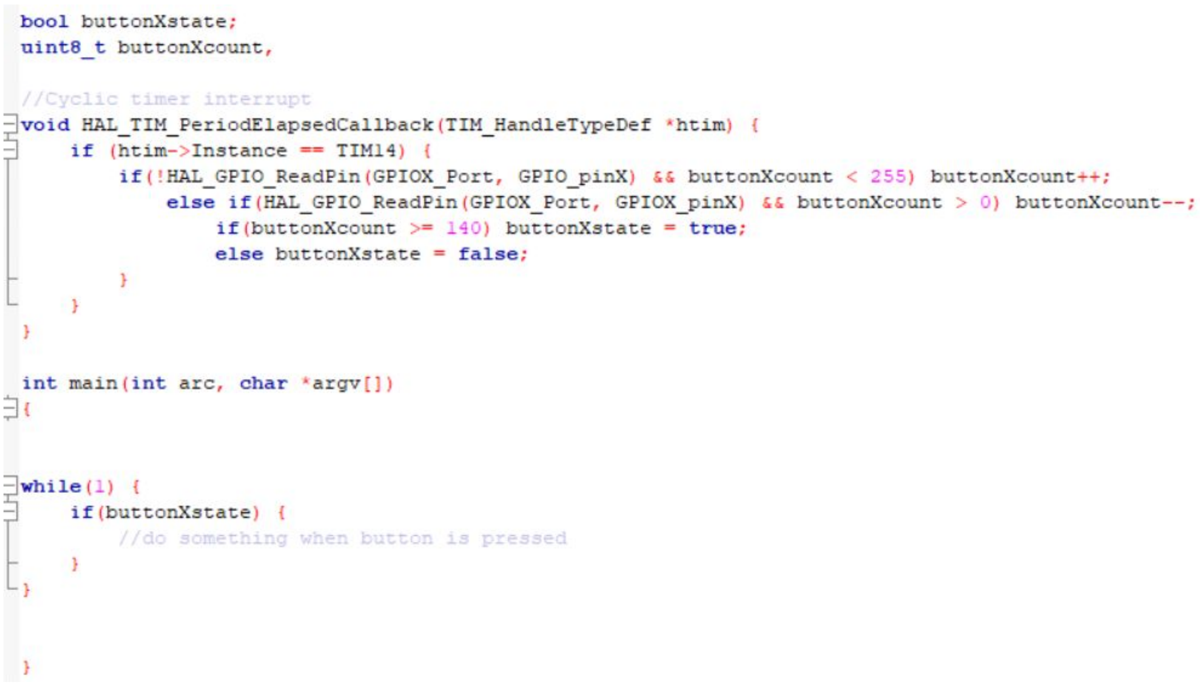


Fig. 1. Simple hardware debouncing

Initially, the pin is pulled-up (when button is pressed there is low level logic state). When the button is pressed, the C1 discharges, eliminating the voltage spikes, likewise when disconnecting the button, the filter capacitor slowly charges.

The simplest **software** solution is, after an edge (falling when pulled-up) occurs (button pressed), to wait a certain time (bounce time) and check the state of the button again.

I realize software debouncing by adding two extra variables for each button and implement simple code:



```
bool buttonXstate;
uint8_t buttonXcount,

//Cyclic timer interrupt
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM14) {
        if(!HAL_GPIO_ReadPin(GPIOX_Port, GPIO_pinX) && buttonXcount < 255) buttonXcount++;
            else if(HAL_GPIO_ReadPin(GPIOX_Port, GPIOX_pinX) && buttonXcount > 0) buttonXcount--;
                if(buttonXcount >= 140) buttonXstate = true;
                else buttonXstate = false;
        }
    }
}

int main(int arc, char *argv[])
{


while(1) {
    if(buttonXstate) {
        //do something when button is pressed
    }
}



}
```

For example, if the timer is set to cyclically interrupt every 1ms, the finally state of button will be assigned 140 ms (bouncing time) after the button is pressed. This is only example, when there is more buttons, better solution is to move code from interrupt to while loop.