Report for the Debounce Assignment

Gioele Bernardini

July 17, 2024

Abstract

This report presents various methods to handle the debounce issue encountered when using a mechanical button in a circuit. The project was implemented using an Arduino R3, a button, and several passive components.

Contents

1 Introduction

Mechanical buttons are prone to generating multiple signals due to mechanical bouncing when pressed or released. This phenomenon, known as *bouncing*, can cause multiple unwanted state changes in digital circuits, leading to incorrect behavior. Debouncing techniques are employed to mitigate this issue and ensure that only a single state change is registered per button press. This report covers the implementation of various debouncing techniques using an Arduino R3 and evaluates their effectiveness.

2 Circuit Setup

There are endless possibilities to get a similar circuit to the one I have been using for the assignment. First off, namely the assignment involved the STM32 Nucleo, but I read that by using the `<Arduino.h>` header, it might be programmed just like a standard Arduino, so I decided to proceed this way.

That is also the reason why I had to make a circuit. The STM Nucleo has indeed a built-in GPIO button, while the Arduino R3 doesn't. The materials I have been using were:

- 220 ohm resistor
- 10k ohm resistor
- Various connecting wires
- Generic LED
- SPST button

The schema for the circuit is in the picture (Figure 1). Below is the photo of the implementation I used (Figure 2). Pay attention to the pull-down resistor regarding the button circuit part; it must be strong enough to interface ground to Vcc. Otherwise, a short circuit will be obtained and might damage the circuit overall.

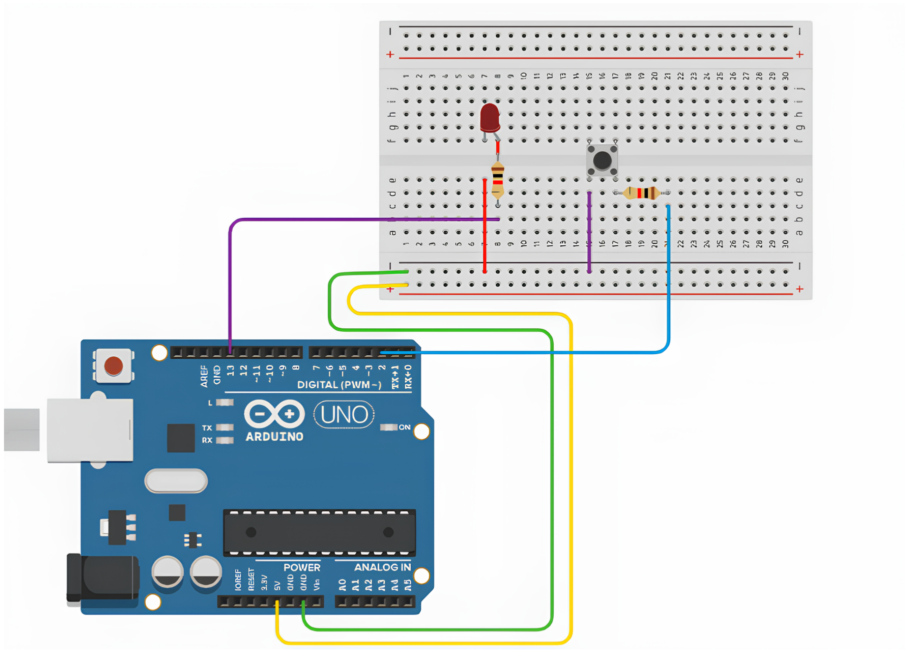

Figure 1: Circuit schematic.

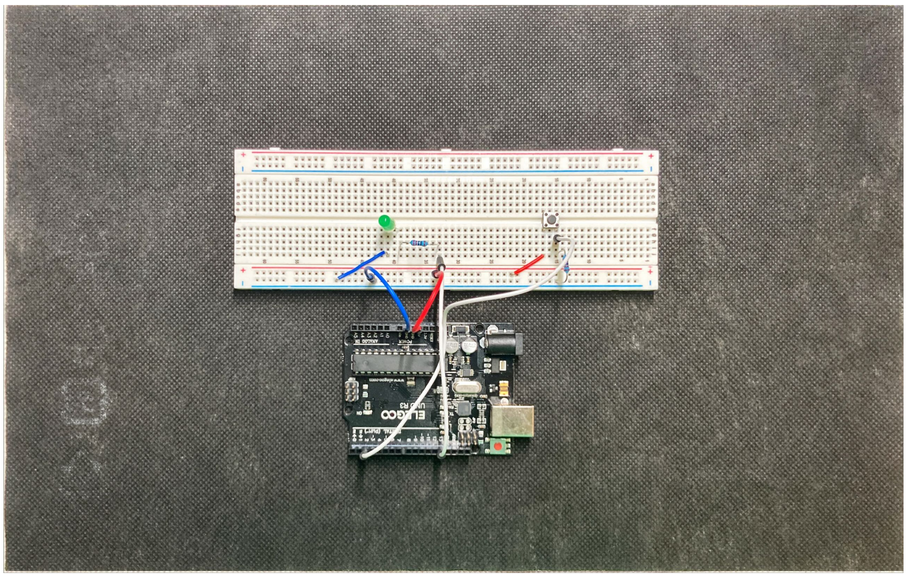

Figure 2: Photo of the circuit implementation.

3 Basic Program

`ButtonBounce.ino` contains the first program I wrote, basically a counter for the presses of the button which does **not** address debounce issues. A simple counter is set to 0 at the start (or restart) of the Arduino and is incremented once a transition from low signal to high signal is recognized. In addition to it, three LED implementations are added:

- The first makes the LED toggle on and off, turning the button into a virtual switch. This is the default mode I used for the solutions since it is easy to notice that, in the case of debounce, the LED will either not change state or change state only after an unexpected blink.
- The second is the most classic behavior: the LED remains on only while the button is pressed.
- The third makes the LED blink rapidly when the button is pressed, regardless of how long the button is held down.

Regardless of the LED behavior, opening the serial monitor (`Ctrl + Shift + M`) reveals debounce problems. The circuit detects that the button is pressed many more times than it actually is. I estimated that without any debounce control, the program detects **62% more presses than what actually occurs**. I implemented a serial plotter via the terminal since the plotter in the Arduino IDE was not functioning properly. More on this in a later section.

It's pretty straightforward that a debounce problem is present, especially since the button is not built-in and is of lower quality (essentially a button from a larger set that takes hits back and forth when moved in the box).

4 Debouncing Solutions

Four debounce methods were implemented and evaluated: stable readings, pre-change debounce, post-change debounce, and debounce estimation.

4.1 Debounce with Stable Readings

This method relies on achieving a certain number of consecutive stable readings before considering the button state as changed.

4.1.1 Algorithm Description

The algorithm for this method involves continuously reading the button state and comparing it to the previous reading. If the current reading matches the previous reading, a stability counter is incremented. Once the stability counter exceeds a predefined threshold, the button state is considered stable, and any actions (e.g., toggling an LED) are performed. If the readings do not match, the stability counter is reset.

4.1.2 Pros and Cons

- **Pros:**
  - Effective in filtering mechanical bounces.
  - Relatively simple to implement.
- **Cons:**
  - Introduces a slight delay in updating the button state.
  - May not be suitable for applications requiring immediate response.

4.2 Pre-change Debounce

This method initiates a debounce timer before considering a button state change as stable.

4.2.1 Algorithm Description

In this method, when a change in the button state is detected, a timer is reset. If the button state remains unchanged for the duration of the debounce period (measured by the timer), the new button state is considered valid and any corresponding actions (e.g., toggling an LED) are performed. If another change is detected before the timer expires, the timer is reset again.

4.2.2 Pros and Cons

- **Pros:**
  - Effectively filters mechanical bounces.
  - Introduces minimal delay in detecting state changes.
- **Cons:**
  - Requires precise timer management to avoid false positives.
  - Introduces a slight delay in updating the button state.

4.3 Post-change Debounce

This method initiates a debounce timer immediately after detecting a button state change, and further state changes are ignored during the debounce period.

4.3.1 Algorithm Description

In this method, as soon as a change in the button state is detected, the button state is immediately updated, and a debounce timer is started. Any further changes in the button state are ignored until the debounce timer expires. This ensures that only the initial state change is registered, and any subsequent bounces are filtered out.

4.3.2 Pros and Cons

- **Pros:**
  - Immediate feedback on button state change.
  - Simpler logic and fewer variables compared to other methods.
- **Cons:**
  - May still register brief bounces as state changes if the debounce period is too short.
  - Ignores all changes during the debounce period, which may miss legitimate rapid presses.

4.4 Debounce Estimation

This method not only handles debouncing but also provides an estimate of the number of debounce events.

4.4.1 Algorithm Description

In this method, each time a change in the button state is detected, the debounce timer is reset. If the button state remains consistent for the duration of the debounce period, the button state is updated. The method also counts the number of debounce events that occur within the debounce period. This can help in manually determining the optimal debounce period.

4.4.2 Determining the Debounce Period

To manually determine the optimal debounce period using this method, follow these steps:

1. **Monitor the serial output** for the number of debounce events.
2. **Adjust the debounce delay** value and observe how the number of debounces changes.
3. **The optimal debounce period** is reached when the number of debounce events is minimized while ensuring reliable button press detection.

4.4.3 Pros and Cons

- **Pros:**
  - Provides insight into the number of debounce events, helping to fine-tune the debounce period.
  - Ensures reliable detection of button presses.
- **Cons:**
  - May require manual adjustment and monitoring to find the optimal debounce period.
  - Slightly
    more complex implementation compared to basic debounce methods.

5 Serial Plotter Simulation

Due to issues with the Arduino IDE's built-in serial plotter, a custom solution was developed to visualize the button press states via a terminal window.

5.1 Program Description

The program reads button press states from the Arduino via a serial port and visualizes these states in the terminal. It displays the current state of the button, transitions between states, and counts the number of button presses. The program continuously updates the display, providing *real-time feedback* on the button's behavior.

5.1.1 Algorithm Description

The program operates as follows:

1. **Initialization:** The serial port is initialized with non-blocking reads, and the plot array is set up to store the states.
2. **Reading Data:** The function `process_serial_char(char value)` reads and processes each character from the serial port, updating the button state and counting button presses on state transitions.
3. **Updating the Plot:** The function `update_plot()` shifts the values in the plot array to the left and inserts the new state at the end.
4. **Displaying the Plot:** The terminal screen is cleared, and the updated plot is printed, showing the states and transitions.
5. **Displaying Additional Information:** The elapsed time and button press count are displayed at the bottom of the terminal.

5.1.2 Key Functions

- `process_serial_char(char value)`: Processes the character read from the serial port and updates the button state.
- `update_plot()`: Shifts the values in the plot array and inserts the new state.
- `initialize_plot()`: Initializes the plot array.

6 Evaluation and Conclusion

Among the three methods, the timer-based debounce was found to be the most effective for general applications due to its **simplicity and reliability**. The post-change debounce method is more efficient in terms of memory usage but can be slightly more complex to implement. The hardware-based debounce was the least effective due to the limited component values available.

In conclusion, the choice of debounce method depends on the specific application requirements and available resources. For most scenarios, a software-based solution offers the best balance of simplicity and effectiveness.

7 Considerations and Improvements

Each debounce method has its own advantages and disadvantages, making them suitable for different scenarios.

7.1 Use Cases

- **Debounce with Stable Readings:**
  - Best suited for applications where a small delay in response is acceptable.
  - Ideal for systems with limited processing power since it is simple to implement.
- **Pre-change Debounce:**
  - Suitable for applications requiring minimal delay in button response.
  - Effective in scenarios where precise timing is crucial, such as in user interfaces with quick inputs.
- **Post-change Debounce:**
  - Ideal for applications where immediate feedback is important.
  - Suitable for environments where the button is expected to be pressed rapidly in succession.
- **Debounce Estimation:**
  - Useful for applications where the optimal debounce period needs to be determined experimentally.
  - Provides additional insights into the debounce behavior, helping to fine-tune the system.

7.2 Potential Improvements

Several potential improvements could enhance the debounce methods discussed. Introducing adjustable debounce periods would allow for fine-tuning based on the specific characteristics of the button and application. Developing adaptive algorithms that dynamically adjust the debounce period based on the frequency and pattern of button presses could further improve performance. Additionally, combining multiple debounce methods, such as integrating pre-change debounce with stable readings, could offer more robust performance across varying conditions, leveraging the strengths of different approaches for a comprehensive solution.