Hacettepe University

BBM434 Experiment 1

Name and Surname: ZÜBEYDE CİVELEK
Identity Number: 2200356814
Experiment Subject: Switch Debouncing
E-Mail: b2200356814@cs.hacettepe.edu.tr

Switch Debouncing

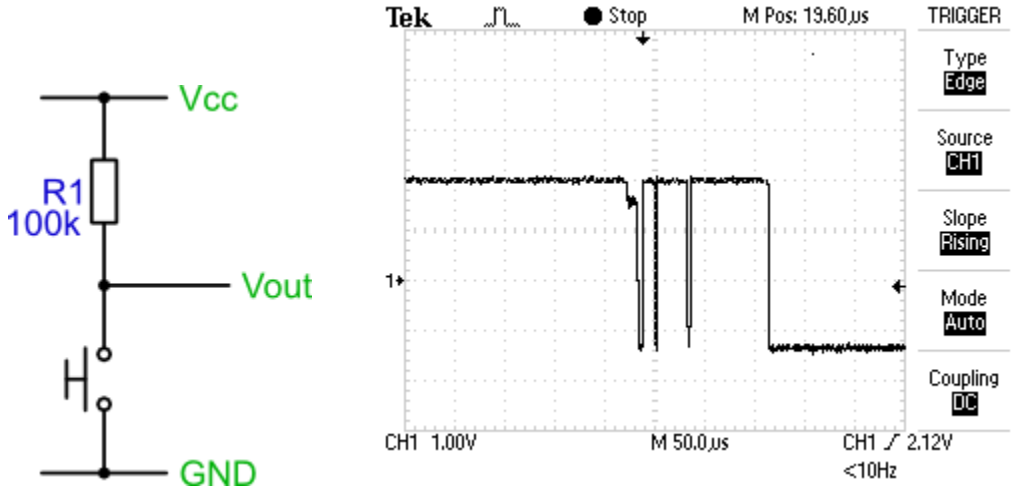
When working with switches and digital circuits, switch debouncing is something you frequently have to deal with. Devise a manual switch signal so that it appears to be pressed more than once if you want to input it into a digital circuit.

What is Switch Bounce?

A pull-up resistor and a basic push switch are seen in the image on the left below. When the switch is pressed, the trace at the output terminal, Vout, is displayed in the image on the right. It is evident that turning on the switch does not produce a clean edge. For instance, you would receive numerous counts instead of expected single count if this signal were the input for a digital counter.

Keep in mind that this might also happen when a switch is released.

The issue is that the switch's contacts don't make clean contact; instead, they 'bounce' a little. Because the bounce is so slow, it's easy to recreate the trace and the issue.



Approach 1 - Timer-Based Approach

In this approach, the time interval between consecutive switch state changes is measured using a timer. The switch is said to be debounced if it stays in that condition for a predetermined amount of time.

```
const int switchPin = 2;
int switchState = 0; // 0 for released, 1 for pressed
int lastSwitchState = 0; // Last recorded switch state
unsigned long lastDebounceTime = 0; // Timestamp of the last state change
unsigned long debounceDelay = 50; // Debounce time in milliseconds

void setup() {
pinMode(switchPin, INPUT);
Serial.begin(9600); // Serial connection for debugging
}

void loop() {
int currentSwitchState = digitalRead(switchPin);
if (currentSwitchState != lastSwitchState) {
lastDebounceTime = millis();
}
if ((millis() - lastDebounceTime) > debounceDelay) {
if (currentSwitchState != switchState) {
switchState = currentSwitchState;

Serial.print("Switch State: ");
Serial.println(switchState == HIGH ? "Pressed" : "Released");
}
}
lastSwitchState = currentSwitchState;
}
```

This code uses a timer to ensure that the switch state remains stable for the specified debounce delay before considering it as a valid button press or release.

Approach 2 - Finite State Machine Approach

This approach monitors the switch's state and identifies a stable press or release state using a finite state machine.

```
const int switchPin = 2;
int switchState = LOW; // Current switch state
int lastSwitchState = LOW; // Previous switch state
int debounceState = 0; // Debounce state: 0 - Idle, 1 - Possibly pressed, 2 - Pressed,
3 - Possibly released
```

```
void setup() {
pinMode(switchPin, INPUT);
Serial.begin(9600); // Serial connection for debugging
}

void loop() {
int currentSwitchState = digitalRead(switchPin);
if (currentSwitchState != lastSwitchState) {
debounceState = 1;
} else {
if (debounceState == 1) {
debounceState = 2;
} else if (debounceState == 2) {
switchState = currentSwitchState;
debounceState = 0;

Serial.print("Switch State: ");
Serial.println(switchState == HIGH ? "Pressed" : "Released");
}
}
lastSwitchState = currentSwitchState;
}
```

Experiment Results

In both experiments, the software-based switch debouncing methods effectively filter out switch bounce, ensuring that only stable transitions between 'Pressed' and 'Released' states are detected. When the button is pressed and held, the code consistently maintains the 'Pressed' state without experiencing debouncing issues. Upon release, the state cleanly transitions to 'Released,' and when pressed again, the switch reliably registers as 'Pressed,' demonstrating the successful mitigation of false readings and the enhancement of switch input accuracy in embedded systems.