EE 1301: IoT

Debouncing Input Switches

“something profound is typed here”

# This quick lesson is a work-in-progress

You have been warned!

# Introduction

Often times Human-Interface elements are complex because the physical buttons, when pressed, physically “bounce” several times before settling on the desired output (see oscilloscope output on the right.) Usually, this bouncing is very rapid, lasts less than 10ms (we have trouble perceiving anything less than 30ms). This string of rapid presses is not noticeable to a human but can cause havoc with software programs.

A **debounce** function converts a rapid string of button presses into a single event that can be handled cleanly by software.

One method to solving this problem adds a delay before allowing the button to transition from “not pressed” to “pressed” and vice versa. Basically, we watch for the button to change state, tell the software the button changed, then wait to allow the button to settle down before accepting further input.

# Required knowledge

Implementing a debounce function is not trivial and requires knowledge of “switch” statements (sometimes called “case” statements), a rough idea of what a “state-machine” is, knowledge of using software timers, and user-defined functions. Please complete the following reading assignments before digging into this quick lesson.

|  |
| --- |
| Pre-Reading Checklist   * Switch Case Statements - <https://docs.particle.io/reference/firmware/core/#switch-case> * Example of a state machine - <https://en.wikipedia.org/wiki/Finite-state_machine#Example:_coin-operated_turnstile> * Quick Lesson - Programming Constructs |

# The debounce state machine

As stated in the Wikipedia article, yet not well emphasized, the key element of a state machine is “*Identical stimuli trigger different actions depending on the current state.*” This means that the state machine responds differently depending on the situation.

Looking at the abbreviated state diagram on the right. We see that there are times when Button=HIGH does not trigger a software event. In fact, the only time Button=HIGH triggers a new software event is when we are in “State:1 - Button not pressed” and has been that way for a while.

Implementing a state machine in c-code is fairly straightforward with the switch statement (also referred to as a “case” statement in other programming languages). An example is shown below:

|  |
| --- |
| **int SwitchState = 1;**  <snip>  **switch(SwitchState) {**  **case 1: //Button Not Pressed**  **// Watch for a button press**  **break;**    **case 2: //Button has just been pressed**  **// Ignore input for ~10ms**  **break;**    **case 3: //Button Pressed**  **// Watch for button release**  **break;**    **case 4: //Button has just been released**  **// Ignore input for ~10ms**  **break;**    **default: //This should never occur**  **//It is good practice to put error trapping code here.**  **break;**  **} // End switch case statement** |

# Human Interface Device - Debounce Code

|  |
| --- |
| **int SwitchState = 1;**  **int ButtonPIN = D2;**  **int PotPIN = A2;**  **int PotOut = 0;**  **bool ButtonOut = FALSE;**  **int ButtonCount = 0;**  **bool ButtonLast = FALSE;**  **int SwitchState = 1;**  **unsigned ButtonTimer = millis();**  **void setup() {**  **pinMode(ButtonPIN, INPUT\_PULLDOWN);**  **pinMode(PotPIN, INPUT);**  **Serial.begin(9600);**  **}**  **void loop() {**    **switch(SwitchState) {**  **case 1: //Button Not Pressed**  **// Wait for a button press**  **if(digitalRead(ButtonPIN)) {**  **ButtonOut = TRUE;**  **ButtonTimer = millis()+10;**  **SwitchState = 2;**  **}**  **break;**    **case 2: //Button has just been pressed**  **// Ignore input for ~10ms**  **if(ButtonTimer < millis()) {**  **SwitchState = 3;**  **}**  **break;**    **case 3: //Button Pressed**  **if(!digitalRead(ButtonPIN)) {**  **ButtonOut = FALSE;**  **ButtonTimer = millis()+10;**  **SwitchState = 4;**  **}**  **break;**    **case 4: //Button has just been released**  **// Ignore Input for ~10ms**  **if(ButtonTimer < millis()) {**  **SwitchState = 1;**  **}**  **break;**    **default: //This should never occur**  **//It is good practice to put error trapping code here.**  **Serial.print("Major Code Bug Detected!");**  **SwitchState = 99;**  **delay(1000);**  **break;**  **} // End switch case statement**  **PotOut = analogRead(PotPIN);**  **if(ButtonOut == HIGH && ButtonLast == LOW) {**  **ButtonCount = ButtonCount + 1;**    **Serial.print("Button Count = ");**  **Serial.print(ButtonCount);**  **Serial.print(" , Level = ");**  **Serial.println(PotOut);**    **ButtonLast = HIGH;**  **} else if (ButtonOut == LOW) {**  **ButtonLast = LOW;**  **}**    **}** |

# 