Project 2: Digital Inputs and Interrupts Solution

Braidan Duffy*

May 25, 2022

Overview

Students will be considered to have completed the project when they demonstrate the following tasks:

1. integrated three buttons inputs and three LED outputs with their Arduino
   - one button must have an unfiltered input
   - one button must have a software-regulated input
   - one button must have a hardware RC low-pass filter circuit
2. implemented three appropriate Interrupt Service Routines (ISRs) into their program
   - each ISR must increment a counter for each button pressed
   - each ISR must send the counter through the Serial port with an appropriate label
3. showed that the LED state switches and the counter is printed to the serial monitor with every button press

Grading

Category	**No Credit**	**Half Credit**	**Full Credit**
Efficacy	Student did not demonstrate their working project	student demonstrated a working project, but it did not meet all of the requirements specified in the project handout	student demonstrated a feature-complete project that accomplished all of the goals specified in the project handout
3 types of input filtering	Student did not implement enough unique filters		Student implemented enough filters
Printout of Serial monitor	Student did not provide a printout of the Serial monitor	Student provided a printout, but it was not appropriately labeled or incoherrent	Student provided an acceptable printout
Schematic neatness	Student did not provide a schematic, or it is illegible	Student provided a schematic, but it is difficult to read or understand	Student provided a schematic that is easy to read and understand
Mystery extra credit	Student did not request extra credit or reach the necessary thresholds		Student successfully implemented a button hold feature

*B.S. Ocean Engineering 2021
M.S. Ocean Engineering 2023

Extra Credit

Students will be granted extra credit if they demonstrated that they record a button press *after* a certain time period and follow the rest of the requirements following the detected press.

Guide

Wiring the Breadboard

Students should have three buttons and three LEDs on their breadboard connected to the Arduino's GPIO pins. The GND pin of the Arduino should be connected to the negative (blue) rail of the breadboard. Each LED should have a current-limiting resistor in series with the anode (long wire) and the Arduino output pin with the cathode (short pin) connected to the negative rail. Two buttons should have a 10k-ohm pulldown resistor on one contact end to the negative rail and the other contact end conected directly to an interrupt-capable input pin. The third button should have an RC circuit in series with the Arduino input wire as shown in Figure 1.

Arduino Code

Students should have a well-organized and thought out program for this project. Firstly, they must declare and initialize the input and output pins being used as well as the counters for the number of button presses. Then, they must have created ISRs to perform the counter increments and printing for each button as well as change the LED state from on to off or vice versa. These interrupts must be attached to the appropriate input pins and, ideally, be triggered on the FALLING edge of the signal - though RISING could also work. The main loop of this function should just be setting the LED state for each LED since most of the logic will be handled asynchronously by the ISRs.

For the sample code provided below, the pins are defined using the `#define` pre-compiler instruction and the LED states and counters are initialized. In `setup()` The Serial port is opened and the modes for the inputs and outputs are declared. Afterwards, the ISRs defined at the bottom of the code are attached to the input pins and set to trigger on the FALLING edge of the pin's signal (i.e. when the button is pressed and the voltage on the pin go towards 0).

In `loop()`, the various LED states are written to the LED pins based on the global variable. The ISR code executes asynchronously, so nothing else is reuqired in this section

Both `unregBtnISR()` and `hwregBtnISR()` do the same thing because the debouncing is either not handled or done so externally to the program. The software trick comes inside `swregBtnISR()`. Here, a static variable is initialized that will retain its value after the ISR is finished executing. This value holds the last known millisecond reading of the ress detection logic. Then, the current millisecond reading is captured and used to compare with the previous value. If 100 ms have passed since the ISR was last executed, then the code block assumes it was a valid button press and increments the counter, changes the LED state, and prints out the counter to the Serial port. If an appropriate time has not passed, then the ISR assumes the detection was a bounce and resets the last millisecond reading before exiting.

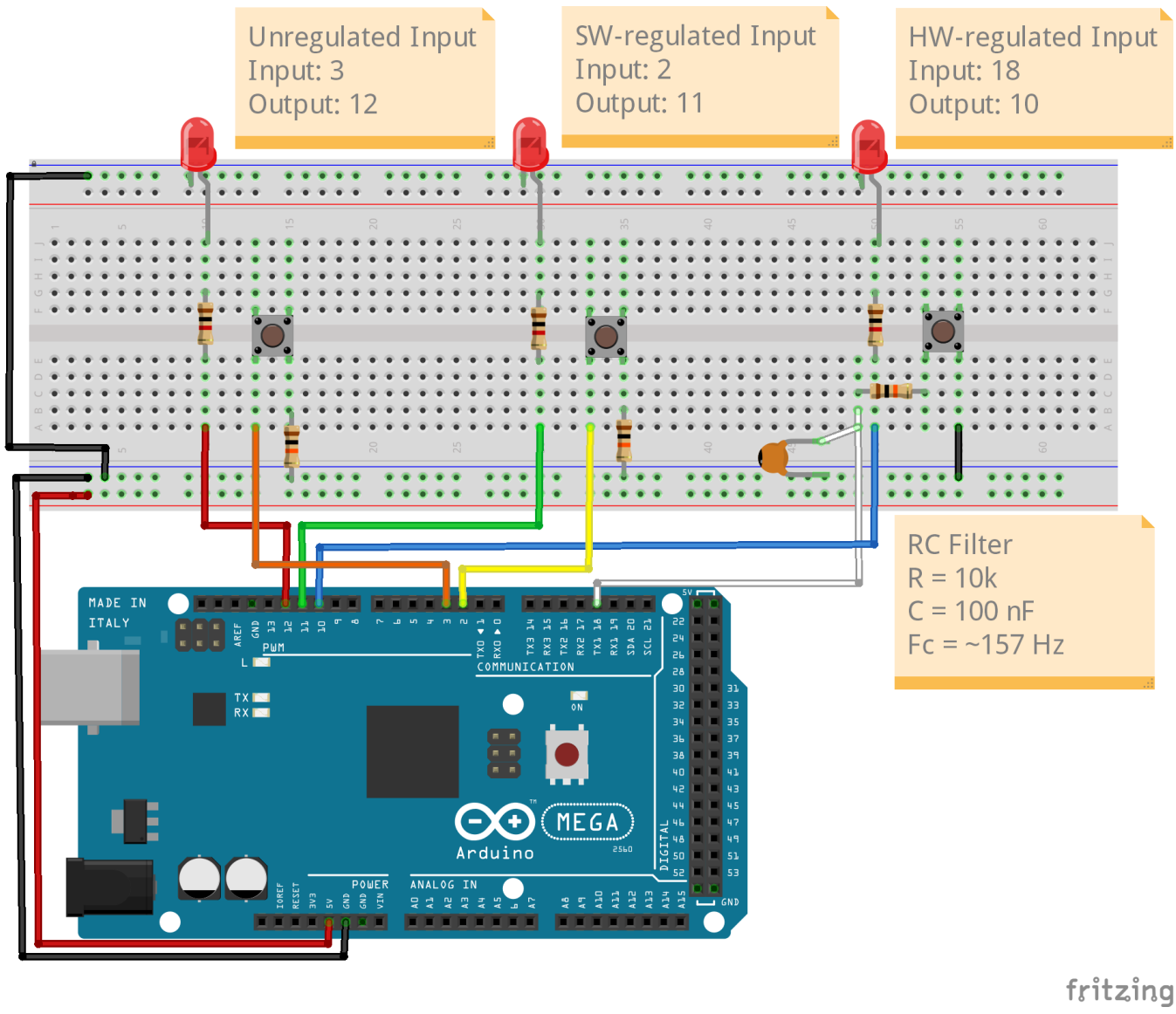

Figure 1: The breadboard layout for Project 2

```
/**
 * OCE4531 Project 2: Digital Inputs and Interrupts Solution
 * @brief This is the solution for OCE4531's second project, Digital Inputs and Interrupts.
 * This should not be considered the absolute correct answer that all of the students must have,
 * but it is the ball park they should be within.
 * See handout for more information
 *
 * @author Braidan Duffy
 *
 * @date May 25, 2022
 *
 * @version v1.0
 */

// Define button pins according to schematic
#define HWREG_BUTTON_PIN 18 // Button that is regulated by a RC low-pass filter
#define SWREG_BUTTON_PIN 3 // Button that is regulated by a software debounce
#define UNREG_BUTTON_PIN 2 // Button that has an unregulated input

// Define LED pins according to schematic
#define HWREG_LED_PIN 10
#define SWREG_LED_PIN 11
#define UNREG_LED_PIN 12

// Initialize LED states
bool hwregLEDState = false;
bool swregLEDState = false;
bool unregLEDState = false;

// Initialize counters
long hwregBtnPresses = 0;
long swregBtnPresses = 0;
long unregBtnPresses = 0;

void setup() {
    Serial.begin(115200);
    while(!Serial); // Wait for Serial connection

    // Set button pins to inputs with internal pullup resistors
    pinMode(HWREG_BUTTON_PIN, INPUT_PULLUP);
    pinMode(SWREG_BUTTON_PIN, INPUT_PULLUP);
    pinMode(UNREG_BUTTON_PIN, INPUT_PULLUP);

    // Attach Interrupt Service Routines (ISRs) to pins
    attachInterrupt(digitalPinToInterrupt(HWREG_BUTTON_PIN), hwregBtnISR, FALLING);
    attachInterrupt(digitalPinToInterrupt(SWREG_BUTTON_PIN), swregBtnISR, FALLING);
    attachInterrupt(digitalPinToInterrupt(UNREG_BUTTON_PIN), unregBtnISR, FALLING);

    // Set LED pins to outputs
    pinMode(HWREG_LED_PIN, OUTPUT);
    pinMode(SWREG_LED_PIN, OUTPUT);
    pinMode(UNREG_LED_PIN, OUTPUT);
}

void loop() {
    // Hardware-regulated button LED code
    digitalWrite(HWREG_LED_PIN, hwregLEDState);

    // Software-regulated button LED code
    digitalWrite(SWREG_LED_PIN, swregLEDState);

    // Un-regulated button LED code
    digitalWrite(UNREG_LED_PIN, unregLEDState);
}

// ================================
// === INTERRUPT SERVICE ROUTINES ===
```

```
68 // ==================================
69
70 void hwregBtnISR() {
71     hwregBtnPresses++; // Increment button presses
72     Serial.println("Hardware-regulated button pressed!");
73     Serial.print("It has been pressed "); Serial.print(hwregBtnPresses); Serial.println("
       times!");
74     Serial.println();
75     hwregLEDState = !hwregLEDState; // Change LED state
76 }
77
78 void swregBtnISR() {
79     static unsigned long _lastInterruptTime = 0;
80     unsigned long _curInterruptTime = millis();
81     if (_curInterruptTime - _lastInterruptTime > 100) { // If interrupts come faster than
       100ms, assume it's a bounce and ignore
82         swregBtnPresses++; // Increment button presses
83         Serial.println("Software-regulated button pressed!");
84         Serial.print("It has been pressed "); Serial.print(swregBtnPresses); Serial.println(
       " times!");
85         Serial.println();
86         swregLEDState = !swregLEDState; // Change LED state
87     }
88     _lastInterruptTime = _curInterruptTime;
89 }
90
91 void unregBtnISR() {
92     unregBtnPresses++; // Increment button presses
93     Serial.println("Un-regulated button pressed!");
94     Serial.print("It has been pressed "); Serial.print(unregBtnPresses); Serial.println("
       times!");
95     Serial.println();
96     unregLEDState = !unregLEDState; // Change LED state
97 }
```