THE ASSEMBLY OF A RESPONSIVE 8-BIT BINARY COUNTER USING ARDUINO

UNIVERSITY OF THE PHILIPPINES VISAYAS
COLLEGE OF ARTS AND SCIENCES
DIVISION OF PHYSICAL SCIENCES AND MATHEMATICS

CMSC 130 - LOGIC DESIGN AND DIGITAL COMPUTER CIRCUITS
2ND SEMESTER AY 2021-2022

ASSIGNMENT 6. 8-BIT BINARY COUNTER IN ARDUINO

PREPARED BY:

DELA CRUZ , LLOYD WALLYS
LESCANO , RENMAR
ORAÑO , MYRTLLE GEM

MAY 31, 2022

Abstract

The objective of this laboratory activity was to develop a functional automated 8-bit counter device using a circuit integrated into Arduino. The device must behave in specific manner depending on the duration press and its current state. In its “LOW” state, the device was instructed to flash eight LEDs with a two-second delay after a long press; and to enter “HIGH” state upon a long press counting from 0 to 255. Conversely, in its “HIGH” state, it must reset the count after a short press; and enter “LOW” state after a long press. These categories of behaviors were implemented in code using Interrupt Service Routines allowing the program loop to be responsive and dynamic. Meanwhile, the hardware was built from components included in the Makerlab Electronics' Arduino Upgraded Learning Kit. Difficulties were experienced in building the framework and many errors were encountered. Nevertheless, the counter was implemented correctly through linking ISRs and declaring external functions.

Table of Contents

Parts **Pages**

List of Figures

Parts **Pages**

I. Design

A. Problem Statement:

The objective of this activity is to program an 8-bit binary counter. The program must follow a restriction where it used binary 1 convention to represent an "on" state of the LED (Light Emitting Diode), and binary 0 to represent an "off" state. Pins 5-12 of your Arduino Uno board were also used, as seen in the given diagram. Pin 12 is the most significant bit (MSB), whereas pin 5 is the least significant bit (LSB). This exercise aimed to (a.) measure the students' critical and mechanical skills, (b.) enhance the ability to construct a working 8-bit binary counter, and (c.) assess the students' understanding of the topic.

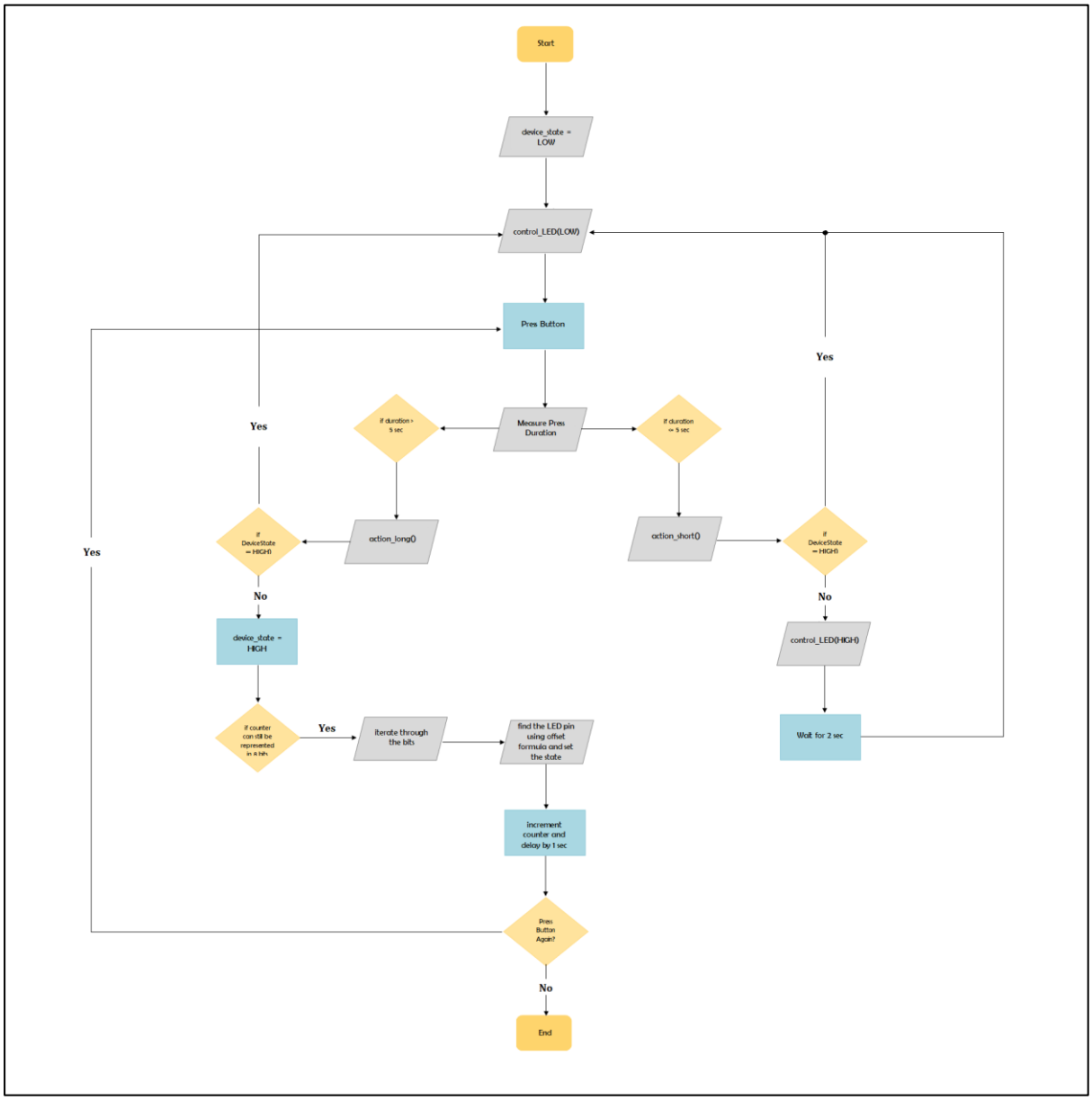


Fig. 1. The flowchart of the system's setup

B. Overview:

For a systematic procedure, the following objectives were pursued:

- Learn the Arduino functions that handle interruptions during a system operation
- Understand the limitations of the Arduino hardware and software in order to program and troubleshoot effectively
- Master how pins can be utilized for digital reading and writing

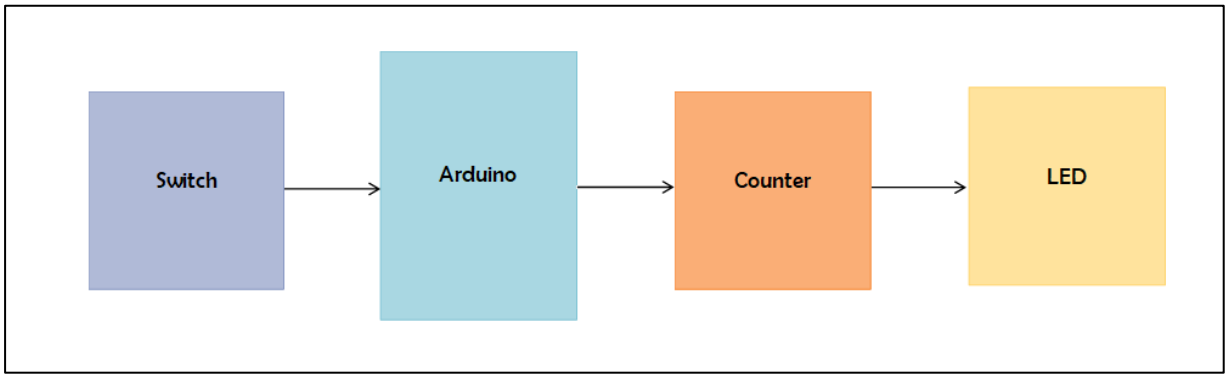


Fig. 2. The high-level block diagram of the 8-bit counter implementation

C. Hardware:

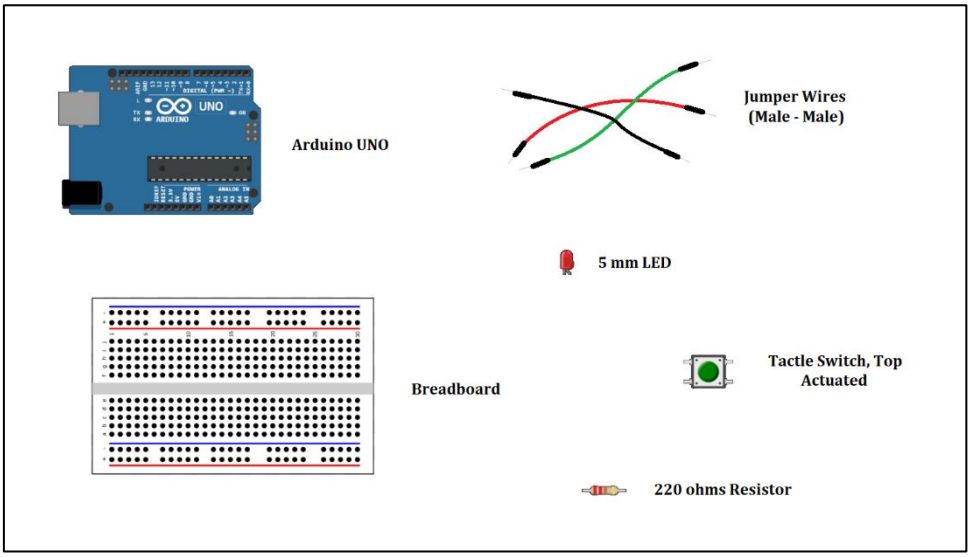


Fig. 3. The hardware required for the system

The following hardware components were used for the implementation:

- LED – produced the program output.
- 220 Ω resistor – regulated the flow of electrical current and supplied voltage to the active device acting as a transistor.
- Arduino Uno – the platform to facilitate the interaction between software and hardware components
- Jumper Wires – connected the pins of each component
- Breadboard – facilitated the flow of electricity between the hardware components.

D. Circuit:

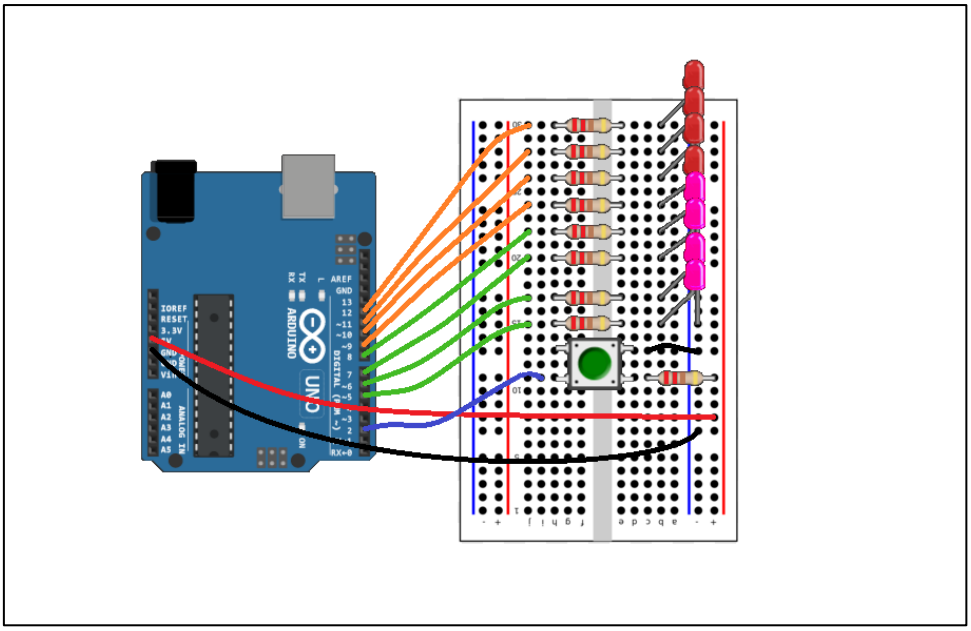


Fig. 4. The integration setup of the counter (Quluzade, 2020)

E. Schematic:

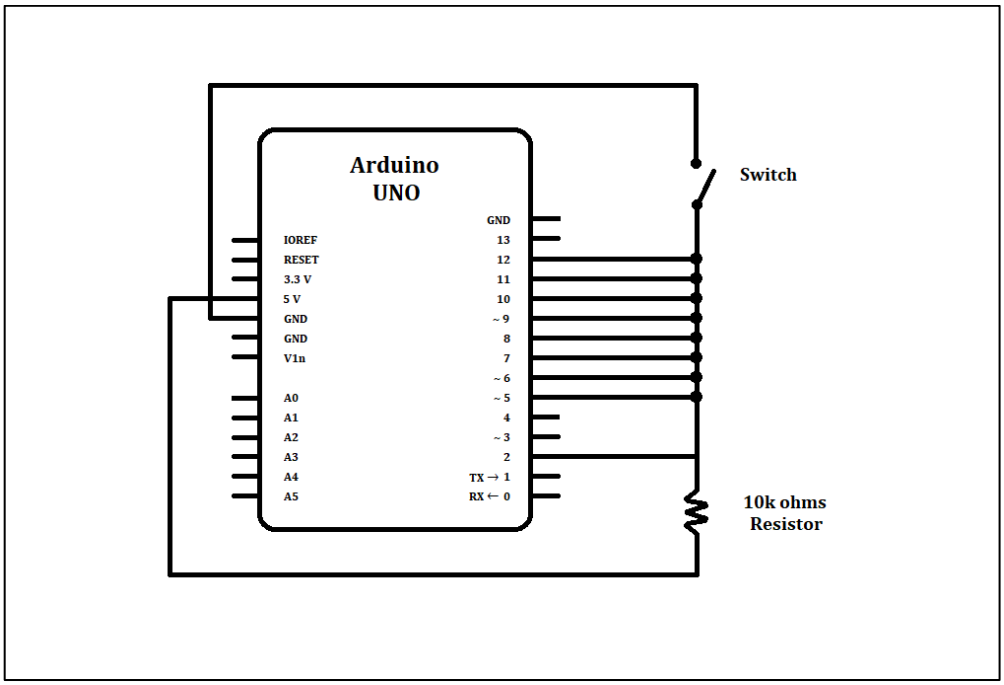


Fig. 5. The schematic for the integrated system

F. Libraries

The *ezButton* library was included into the Arduino file as part of the program. This library introduces convenient functionalities for pushbuttons and debouncing.

G. Methods

Interrupt Service Routines are powerful tools in modifying system behavior while in operation. These functions, which could be called via *attachInterrupt()*, allow programs to become flexible and modular introducing specific decision paths based on input. Inherently, this could also facilitate automations if provided with analog inputs.

Specifically in this activity, ISRs are utilized to detect button presses and record the time in between press down and release. This way, the program is allowed to skip finishing a loop iteration immediately advancing to a specified function.

H. Code:

```
#include <ezButton.h>
ezButton button(2);

unsigned long press_down = 0, press_up = 0;
const int threshold = 5000, pin_count = 8;
int state, counter = 0, device_state = LOW;

bool pressed = false;

byte pins[] = {5, 6, 7, 8, 9, 10, 11, 12};
String binary_form;

void setup() {
  Serial.begin(9600);                                   // activate serial
  button.setDebounceTime(50);                           // debounce for button
  attachInterrupt(digitalPinToInterrupt(2), down, FALLING);   // on press-down, call Interrupt Service Routine

  for(int i = 0; i < pin_count; i++) {                  // record all pins as output pins
    pinMode(pins[i], OUTPUT);
  }
}
```

Fig. 6. Declarations, assignments, and setup

```
23  void loop() {
24    if (pressed == true){                                  // if button is pressed (from ISR "up")
25      long duration = press_up-press_down;                 // calculate duration of press
26      Serial.print("PRESS DURATION: ");                    // print the duration in milliseconds
27      Serial.print(duration);
28      Serial.println();
29
30      if (duration > threshold)                            // if duration > 5 sec
31        action_long();                                     // call action for long press
32      else                                                 // if duration <= 5 sec
33        action_short();                                    // call action for short press
34
35      pressed = false;                                     // reset press
36    }
37
38    if (device_state == HIGH){                             // if device is set to ON
39      if (counter <= 255) {                                // if counter can still be represented in 8 bits
40        binary_form = String(counter, BIN);                // convert counter to bits
41        int activeLED = binary_form.length();              // number of bits = number of LEDs to use
42
43        for(int i = 0; i < activeLED; i++) {               // iterate through the bits
44          if(binary_form[i] == '1') state = HIGH;          // if bit = 1, set the LED to ON
45          else state = LOW;                                // if bit = 0, set the LED to OFF
46          digitalWrite(i + 4 + activeLED - 2*i, state);    // find the LED pin using offset formula and set the state
47        }
48        Serial.print("COUNTER VALUE: ");                   // print the counter value
49        Serial.print(counter);
50        Serial.println();
51        counter++;                                         // increment counter
52        delay(1000);                                       // delay by 1 sec
53
54        } else {                                           // if counter exceed 255
55          counter = 0;                                     // reset to 0
56        }
57    }
```

Fig. 7. The main loop of the program.

```
62  void down() {                                          // ISR for press down
63    press_down = millis();                               // record time
64    attachInterrupt(digitalPinToInterrupt(2), up, RISING);
65  }
66
67  void up() {                                            // ISR for press release
68    press_up = millis();                                 // record time
69    pressed = true;                                      // set pressed to true
70    attachInterrupt(digitalPinToInterrupt(2), down, FALLING);
71  }
```

Fig. 8. The ISR functions that record the time during presses.

```
73  void action_long(){                                   // if press > 5 sec
74    if (device_state == HIGH) {                         // if device is ON
75      device_state = LOW;                               // set device to OFF
76      control_LED(LOW);                                 // turn OFF all LED
77      Serial.println("LONG PRESS ON DEVICE ACTIVE");
78    } else {                                            // if device is OFF
79      device_state = HIGH;                              // set device to ON
80      Serial.println("LONG PRESS ON DEVICE INACTIVE");
81    }
82    counter = 0;                                        // reset counter
83  }
```

Fig. 9. The function that handles long presses.

```
85  void action_short(){                                  // if press <= 5 sec
86    if (device_state == HIGH){                          // if device is ON
87      control_LED(LOW);                                 // turn OFF all LED
88      counter = 0;                                      // reset counter
89      Serial.println("SHORT PRESS ON DEVICE ACTIVE");
90    } else {                                            // if device is OFF
91      control_LED(HIGH);                                // turn ON all LED
92      delay(2000);                                      // wait for 2 sec
93      control_LED(LOW);                                 // turn OFF all LED
94      Serial.println("SHORT PRESS ON DEVICE INACTIVE");
95    }
96  }
```

Fig. 10. The function that handles short presses.

```
98  void control_LED(int state) {                         // to control LED state
99    int pin = 0;
100   while (pin < pin_count){                            // iterate through all pins
101     digitalWrite(pin + 5, state);                     // and set to passed state, note the offset
102     pin++;
103   }
104 }
```

Fig. 11. The function that controls the eight LEDs.

II. Testing and Debugging

While the specification of the 8-bit counter is straightforward, it was challenging to implement because of the limited resources on functional interruptions within the main loop. Seemingly, interruptions could be handled by Interrupt Service Routines (ISR) or functions that are triggered by interruptions in the input. However, this is only effective for short button presses. Apparently, the duration of a press is only recorded at its beginning using the generic *attachInterupt()* function call. Therefore, long presses will be recorded by default akin to short presses.

However, by methodically chaining interrupt calls where: (a) on press-down, a time-recording ISR is called from the main loop; while (b) on press-release, another time-recording ISR is called from within the first ISR (see Fig. 14). This workaround allows the system to timestamp the beginning and end of a button press and subsequently calculate the duration. Afterwards, the algorithm and the rest of the code are simply translations of the specification.

```
62  void down() {                                          // ISR for press down
63    press_down = millis();                               // record time
64    attachInterrupt(digitalPinToInterrupt(2), up, RISING);
65  }
66
67  void up() {                                            // ISR for press release
68    press_up = millis();                                 // record time
69    pressed = true;                                      // set pressed to true
70    attachInterrupt(digitalPinToInterrupt(2), down, FALLING);
71  }
```

Fig. 12. The chained ISR for detecting and recording the time for button presses

Particularly for this project, four (4) attempts were made before the chaining of interrupt calls was correctly implemented. Prior to the working program, the following summarized challenges and issues were also experienced:

1. **The incompatibility of ezButton methods with attachInterrupt() syntax**

   As with the previous laboratory projects, a debounce mechanism was integrated using the ezButton library. However, in this activity, it was discovered that attachInterrupt() function calls are incompatible with the buttons set from the ezButton class. Specifically, it cannot detect the pin if an ezButton instance is utilized. Therefore, the pin parameter was replaced explicitly with the integer data of the input pin.

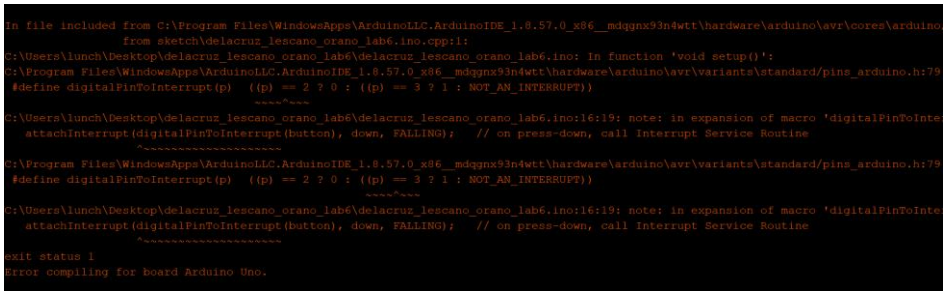


Fig. 13. The error message when an ezButton object is set as the *digitalPinToInterrupt* parameter in *attachInterrupt* method calls

2. **Interrupt Service Routines unable to return because of an infinite loop**

An instance was encountered where outputs were only coming from the chained ISR. Apparently, there was an undetected infinite loop from an external function that prohibited a return to the main loop and only allowing ISR functions to work. Because of its nature, this issue took a long time to resolve as it did not throw a warning or error message and because the program somehow functions. Nevertheless, after rewriting the code, the issue was detected and debugged accordingly.

3. **Some LEDs not turning on**

During tests, there are instances where some LEDs fail to turn on. To troubleshoot this issue, the algorithm and codes were checked together with the circuit. Occasionally, however, the issue would persist. For good measure, the LEDs were reinserted, and their positions were shifted. Fortunately, this solution was effective, and the LEDs finally behaved as expected.

4. **Inconsistent calculation of press duration**

The variability on how a button can be physically interacted with could result to unexpected outputs. Specifically, during testing of the counter, many instances were observed where the calculation for the duration of the press is equal to zero (0). Yet, the program functions normally as it would given a short press. While persistent, this issue rarely occurs and as stated, is not disruptive with the functions of the counter.

```
14:50:22.585 -> PRESS DURATION: 206
14:50:24.605 -> SHORT PRESS ON DEVICE INACTIVE
14:50:25.924 -> PRESS DURATION: 166
14:50:27.912 -> SHORT PRESS ON DEVICE INACTIVE
14:50:33.620 -> PRESS DURATION: 5103
14:50:33.620 -> LONG PRESS ON DEVICE INACTIVE
14:50:33.667 -> COUNTER VALUE: 0
14:50:34.602 -> COUNTER VALUE: 1
14:50:35.633 -> COUNTER VALUE: 2
14:50:36.600 -> COUNTER VALUE: 3
14:50:37.629 -> COUNTER VALUE: 4
14:50:38.614 -> COUNTER VALUE: 5
14:50:39.593 -> COUNTER VALUE: 6
14:50:40.594 -> COUNTER VALUE: 7
14:50:41.626 -> COUNTER VALUE: 8
14:50:42.610 -> COUNTER VALUE: 9
14:50:43.595 -> PRESS DURATION: 6109
14:50:43.642 -> LONG PRESS ON DEVICE ACTIVE
14:50:45.470 -> PRESS DURATION: 157
14:50:47.457 -> SHORT PRESS ON DEVICE INACTIVE
14:50:48.760 -> PRESS DURATION: 128
14:50:50.753 -> SHORT PRESS ON DEVICE INACTIVE
14:50:52.160 -> PRESS DURATION: 147
14:50:54.129 -> SHORT PRESS ON DEVICE INACTIVE
```

Fig. 14. Sample outputs of the press duration in the Serial Monitor.

No other notable concerns were observed aside from these software and hardware issues. Ultimately, with the working code, the succeeding tests provided expected outputs both in the Serial Monitor and in the circuit. Precisely, the 8-bit counter was able to reliably perform the following operations:

TABLE 1. THE SUMMARY OF DEVICE OPERATIONS DEPENDING ON PRESS DURATION

Press Duration	Initial Device State	Operation
Less than or equal to 5 seconds	inactive	Turn on all LEDs. After 2 seconds, turn off all LEDs.
	active	Reset the counter to 0.
Greater than 5 seconds	inactive	Activate the device. Set the counter to 0 and increment by 1 after 1 second until counter is at 255. Then, repeat.
	active	Deactivate the device.

III. Conclusion

In summary, an 8-bit counter with specific button interactions was implemented in this activity. Particularly, there are dedicated actions for short (less than or equal to 5 seconds) and long (more than 5 seconds) button presses. Moreover, depending on the device state, the resulting behavior also varies. Thus, it is integral to develop a comprehensive and flexible program structure to facilitate all these specific interactions. Consequently, amidst initial difficulties, a program framework based on Interrupt Service Routines was developed. Moreover, in contrast to the previous activities, the program for the 8-bit counter employs multiple external functions to allow organized processing of inputs. Altogether, these improved and more organized approach in programming Arduino resulted to a system that is easier to describe, test, troubleshoot, and verify. And as implied, the 8-bit counter, with all its specification, was correctly and efficiently implemented.

Evidently, this exercise serves as a culmination activity for all the concepts discussed in the course. All the recommendations and insights from previous laboratory activities were strictly taken into consideration allowing the students to hypothesize and formulate workarounds. Arduino's appeal as limited system both in software and hardware. However, with resourcefulness and careful planning, unexpected complex systems could be designed.

References

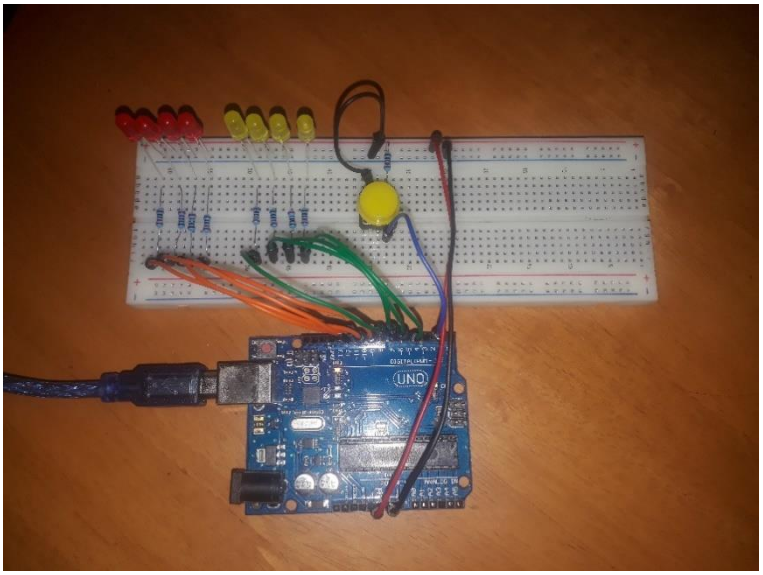
Arduino, "Attachinterrupt()," *attachInterrupt() - Arduino Reference*, 2022. [Online]. Available: https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/. [Accessed: 31-May-2022].

Arduino Binary (8 bit) LED's counter. YouTube, 2020.

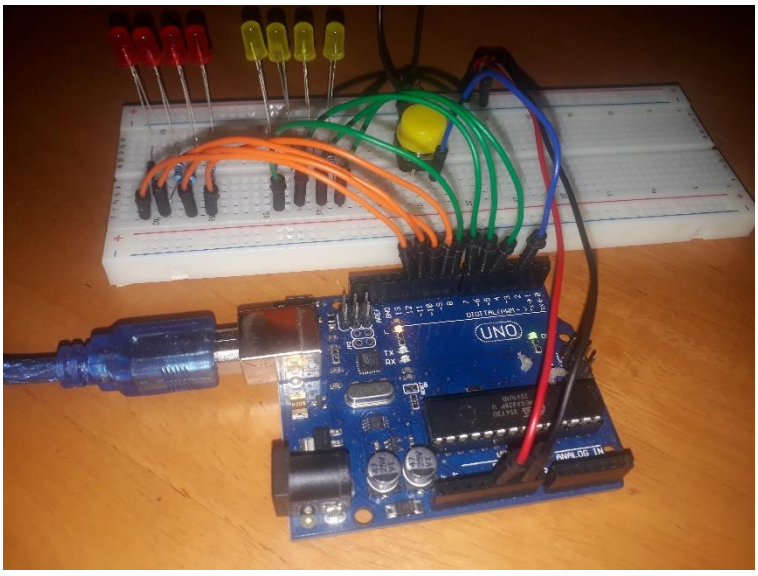
SwengX, "Pushbutton short/long press with interrupt," *Arduino Forum*, 27-Jul-2020. [Online]. Available: https://forum.arduino.cc/t/pushbutton-short-long-press-with-interrupt/667457/4. [Accessed: 31-May-2022].

Appendix

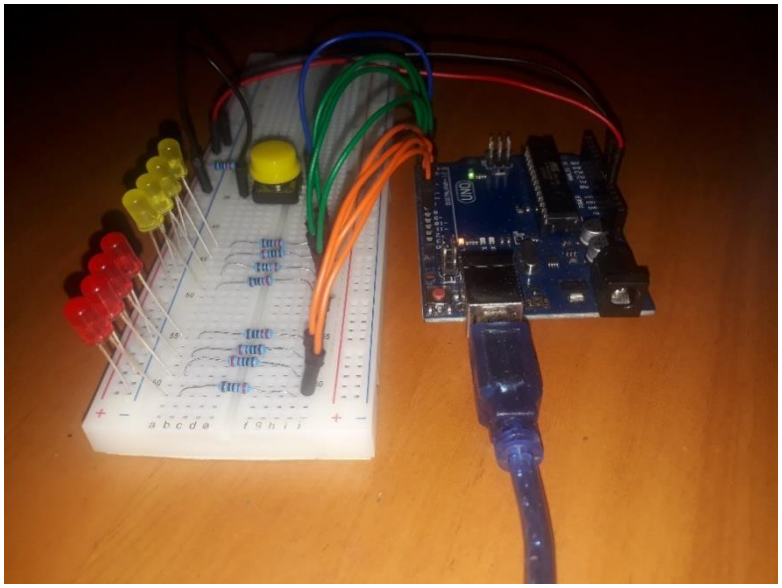
Additional documentary images of the Arduino Hardware Setup for the 8-bit Counter.



The top-view of the 8-bit counter device.



The side-view of the 8-bit counter device highlighting the pins in use.



The side-view of the 8-bit counter device highlights the connections.