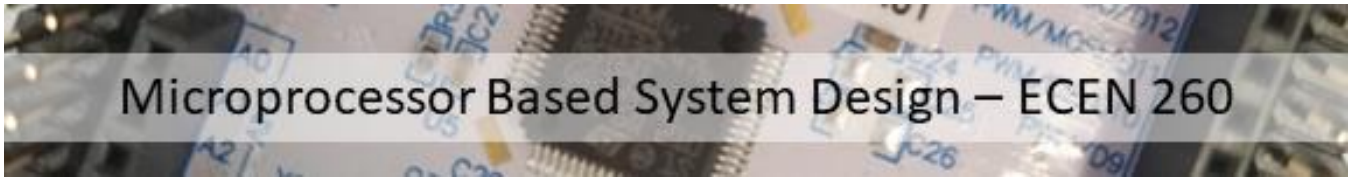

ECEN 260 Lab Report

Lab #8: Interrupts

Name: Brodric Young

Lab Partner: Karl Richards

Date: 03/01/2025

Instructor: Brother Allred

Semester: Winter 2025

Lab Overview

This lab focuses on using interrupts to handle input events efficiently rather than just polling. We do this by implementing a password security system. The objective is to implement a system where a keypad input triggers an interrupt-driven response from the microcontroller. Unlike polling where it continuously checks for user input, this approach makes sure that the microcontroller can do other things and also conserve power. The system will recognize keypad entries, validate passwords, and control LEDs for system status.

Specifications

The system allows users to set and try passwords using a 4x4 matrix keypad. The password-setting function is initiated by pressing 'A,' allowing users to enter up to 10 digits before locking the system with 'B' (blue LED). The password validation function is activated by pressing 'C,' enabling users to input their password and confirm it with 'D'. A correct password unlocks the system (green LED), while an incorrect one signals failure (yellow LED). An alarm function activates after three consecutive failed attempts, lighting a red LED ('B'-level) or blinking it ('A'-level). The system can be disarmed by pressing Button 1. The limitations are there is a maximum password length of 10 digits and limited to the available digits on the matrix excluding A, B, C and D.

Parts List:

- Nucleo-L476RG development board
- USB cable
- Breadboard
- 4 LEDs (Red, Yellow, Green, Blue)
- 4 resistors
- 4x4 matrix keypad
- 1 µF capacitor
- 10 µF capacitor
- 74C922 keypad encoder IC
- ≈24 jumper wires

Schematic

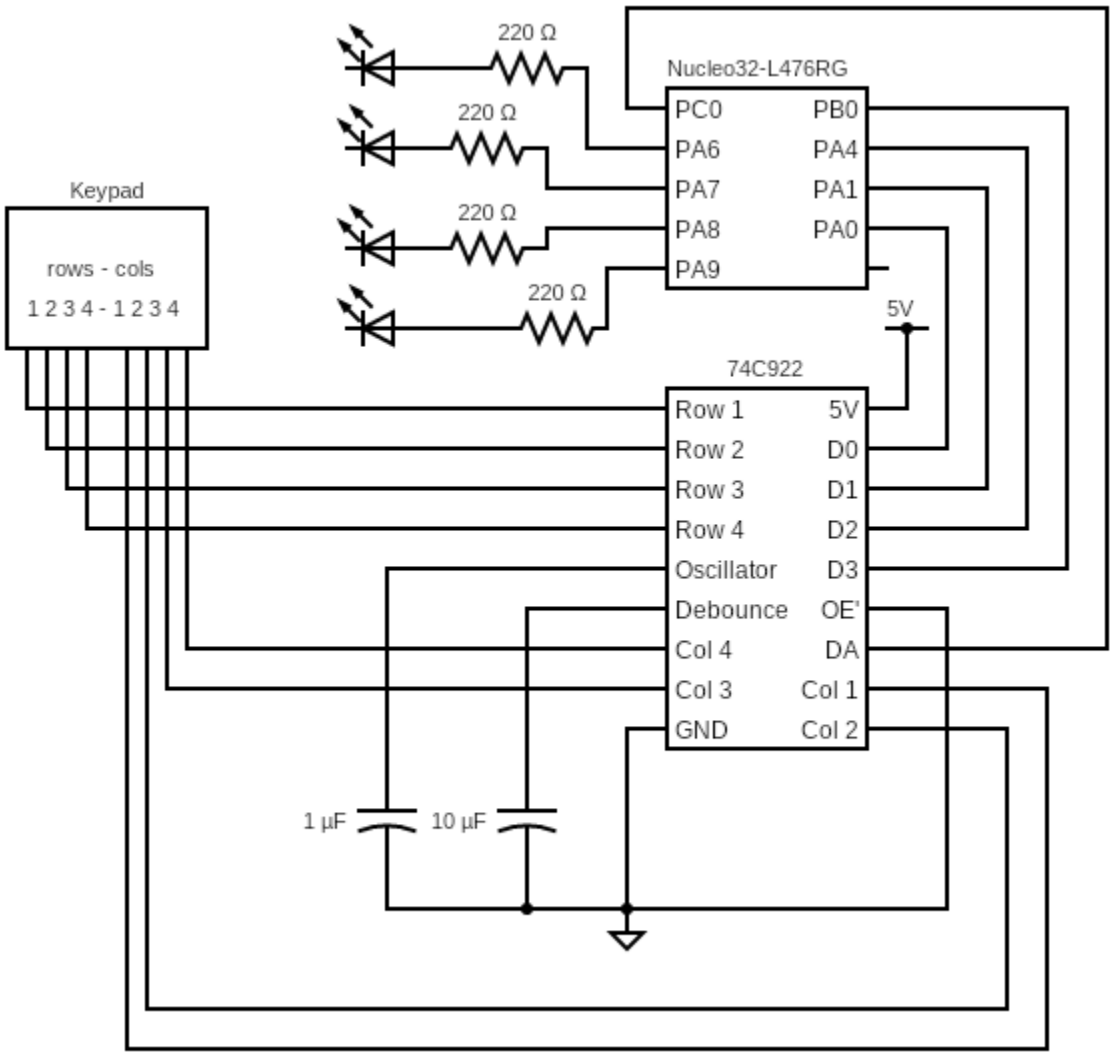

Figure 1: A schematic of the final system.

Test Plan and Results

Fill out the table below with your test plan. The test plan should include several tests, each with precise steps to complete the test. The set of tests should include common cases as well as edge cases / error cases as appropriate. Each test should also accurately indicate *specific* expected results. Then, the actual results should be recorded after completing each test. Delete this paragraph. (i.e., Don't leave the template instructions in your report.)

#	Test Scenario	Expected Result	Observed Result
1	Step 1: Press 'A' Step 2: Press '123' Step 3: Press 'B'	Blue LED should turn on	Blue LED turns on
2	After test scenario 1, Step 1: Press 'C' Step 2: Press '123' Step 3: Press 'D'	Green LED should turn on, blue should turn off	Green LED turns on, blue turns off
3	After test scenario 1, Step 1: Press 'C' Step 2: Press '5' Step 3: Press 'D'	Yellow LED should turn on	Yellow LED turns on
4	Repeat test scenario 3, 3 times	Red LED should blink	Red LED blinks
5	Press Button 1	All LEDs should turn off	All LEDS turned off
6	After test scenario 5, Enter previous password	Green LED should turn on	Green LED turns on
7	After test scenario 1, Step 1: Press 'C' Step 2: Enter 11 digits Step 3: Press 'D'	Yellow LED should turn on	Yellow LED turns on
8	Step 1: Press 'A' Step 2: Enter 10 digits	Blue LED should turn on	Blue LED turns on
9	After test scenario 4, Press any buttons other than disarm button	Nothing should change	Nothing changes
10	Step 1: Press 'A' Step 2: Enter correct password Step 3: Press 'B'	Yellow and blue LED should turn off, green LED should turn on	Yellow and blue LED turn off, green LED turns on
11	Step 1: Press 'A' Step 2: Press '#*7' Step 3: Press 'B'	Blue LED should turn on	Blue LED turns on
12	After test scenario 11, Step 1: Press 'C' Step 2: Press '#*7' Step 3: Press 'D'	Green LED should turn on, blue LED turns off	Green LED turns on, blue LED turns off

Table 1: Test plan with expected and observed results.

Code

Decode Key Press:

```
unsigned char keypad_decode() {
	unsigned char key = 0x0;
	unsigned char data = 0b0000;

	// read the data pins and combine into the 4-bit value: D3_D2_D1_D0
	unsigned int D3 = HAL_GPIO_ReadPin(DATA3_GPIO_Port, DATA3_Pin);
	unsigned int D2 = HAL_GPIO_ReadPin(DATA2_GPIO_Port, DATA2_Pin);
	unsigned int D1 = HAL_GPIO_ReadPin(DATA1_GPIO_Port, DATA1_Pin);
	unsigned int D0 = HAL_GPIO_ReadPin(DATA0_GPIO_Port, DATA0_Pin);
	data = ((D3 << 3) | (D2 << 2) | (D1 << 1) | D0);
	// switch statement to decode the data to the correct hex digit
```

What to do on a key press:

```
// Check if the interrupt was for Data Available Pin
	if (HAL_GPIO_ReadPin(DATA_AVAILABLE_GPIO_Port, DATA_AVAILABLE_Pin)) {

		// Turn on LED2 to show key press detected
		HAL_GPIO_WritePin(KEY_LED_GPIO_Port, KEY_LED_Pin, 1); // turn on
the KEY_LED (LED2)

		// Read data
		uint8_t key = keypad_decode();  // determine which key was pressed

		// if alarm is not going off, receive inputs
		if (!alarm) {…}
		// if alarm != false, set to true so the main while loop knows to
	blink R_LED
		else {
			alarm = true;
		}
```

What to check over and over:

```
while (1)
  {
        // if 3 incorrect passwords are entered, blink R_LED
        if(alarm) {
              HAL_GPIO_TogglePin(R_LED_GPIO_Port, R_LED_Pin);
              HAL_Delay(250);
        }
  }
```

Conclusion

This lab successfully implemented an interrupt-driven password security system using a microcontroller, a keypad, and LED. By using interrupts instead of polling, the system efficiently detects and responds to keypad inputs. It demonstrated the practical application of interrupts, input handling, and state management.

This lab reinforced what we learned in class about interrupts. It expanded on previous labs involving GPIO configurations and using the HAL drivers. The skills developed here will be useful for future projects when we need to do anything based off a state change. Additionally, understanding how to handle interrupts lays a strong foundation for working with more complex systems in the future.