ECEN 260 Lab 08
Interrupts

Lab Objectives

- Learn how to program I/O ports that use interrupts.
- Implement a home entrance system that requires interrupts to operate.

Required Parts (same as last lab)

- 1 Nucleo-L476RG development board
- 1 USB cable
- 1 breadboard (protoboard)
- 4 LEDs (Red, Yellow, Green, Blue))
- 4 resistors
- 4x4 matrix keypad
- 1 μF capacitor
- 10 μF capacitor
- 74c922 keypad encoder IC
  (may be in the same foam as display)
- $\approx$24 jumper wires

Overview

Microcontrollers can serve several I/O devices. There are two ways to do that: polling or interrupts. In polling, the CPU continuously monitors the status of a given device. When the status condition is met, it performs the service. In the interrupt method, whenever any device needs service, the device notifies the CPU by sending an interrupt signal. Upon receiving the interrupt signal, the CPU halts whatever it is doing and serves the device.

This laboratory will implement a home entrance system for a smart home. It will use an interrupt signal to notify the microcontroller that a key on the keypad has been pressed. The program will then read the key value and respond according to the key that is pressed.

(This laboratory is similar to the one described at the end of Chapter 6 in the *Programmable Microcontrollers* textbook.)

Note: There is **no lab partner portion** for this lab. It may be completed on your own. Working with a lab partner is allowed, but not required. If you work with a lab partner, you may help each other, but you must each write your own code, wire your own alarm system, and write your own report.

Lab Requirements

Requirement 1 – Saving a Password: At startup, the user should press the A button on the keypad to begin saving a new password. Then, he or she can enter a password (up to 10 digits) using the number buttons. Afterwards, the user can lock the system using the B button. A connected **blue** LED should turn on to indicate that the new password has been recorded and the system is *locked*.

Requirement 2 – Testing a Password: If the user wants to unlock the system, first he or she should press the C button to begin entering a password. The user then presses the number keys to enter the password. After entering the password, the user presses the D button to test the password. If the correct password was entered, the system will unlock and light up the connected **green** LED. If the wrong password was entered, the system will light up the connected **yellow** LED and remain locked.

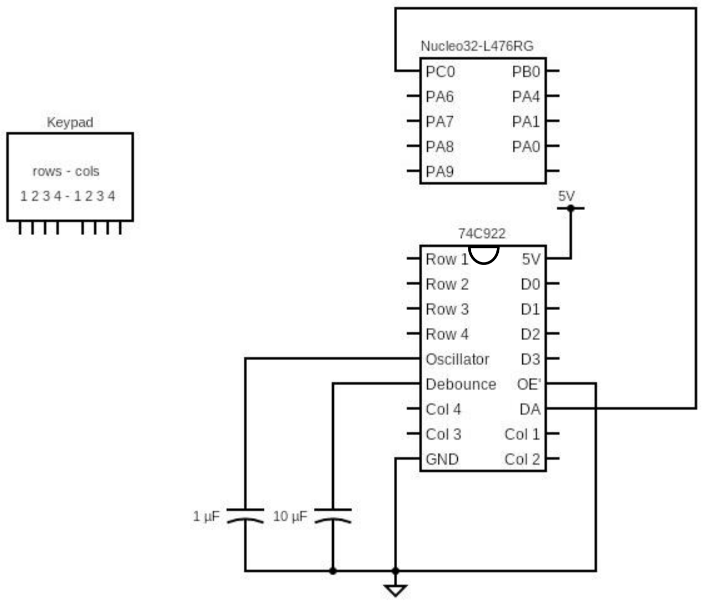
Requirement 3 – The Alarm:

'B'-level project: If the wrong password is entered three times consecutively, the connected **red** LED will turn on and stay on to indicate an alarm condition. When Button 1 (PC13) is pressed, the system will disarm and the system will unlock, shown by the red LED turning off.

'A'-level project: If the wrong password is entered three times consecutively, the connected **red** LED will begin blinking to indicate an alarm condition. When Button 1 (PC13) is pressed, the system will disarm and the system will unlock, shown when the red LED stops blinking. Do the 'B'-level first; then, once that is working, try to change it to an 'A'-level project. See the last page of these instructions for more details on an 'A'-level project.

Video Example: See this video for an example of how your alarm should work (shows the 'B'-level).

Starting Schematic (you must complete it)



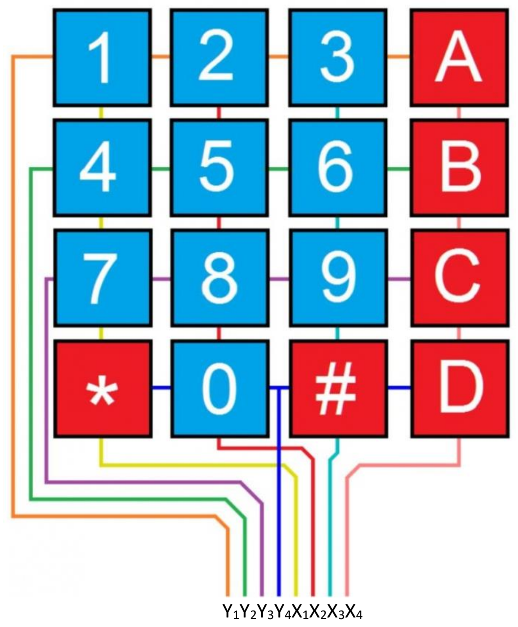
Here is the starting schematic file for use on circuit-diagram.org: link. **Note:** From now on, your schematics do not need to show the components that are already on the board, such as B1 (PC13) and L2 (PA5), nor do your schematics need to show the internal pull resistors – just the items you connect.

The Key Encoder Chip (74C922)

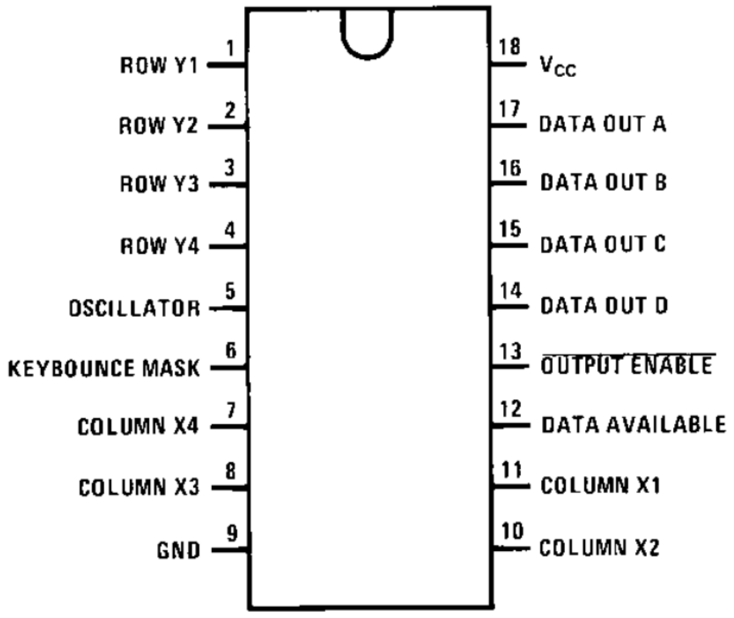
This lab uses a keypad with 16 keys arranged in a 4x4 matrix (shown below on the left). There is a chip in your lab kit called the 74C922 key encoder chip. (Its pin diagram is shown below on the right.) The encoder chip works by scanning each row and column of the keypad to see which row and column are connected to each other. When it detects that a key has been pressed, it encodes the key value as a 4-bit binary number and places that value on the DATA OUT pins. Finally, it sends a rising edge on the DATA AVAILABLE pin to indicate that a key was pressed.

When the interrupt controller in the microcontroller detects that rising edge from the DATA AVAILABLE pin, it will read the 4-bit value from the DATA OUT pins.

Pin Diagrams



Keypad Pins



Key Encoder IC Pins

Note: The encoder chip doesn't know how the buttons are labeled on the keypad. It just uses the first 2 bits to indicate the row and the last 2 bits to indicate the column (shown below). In your code, you will edit the `keypad_decode()` function to remap the data value to the printed keypad labels.

0000 (0x0)	0001 (0x1)	0010 (0x2)	0011 (0x3)
0100 (0x4)	0101 (0x5)	0110 (0x6)	0111 (0x7)
1000 (0x8)	1001 (0x9)	1010 (0xA)	1011 (0xB)
1100 (0xC)	1101 (0xD)	1110 (0xE)	1111 (0xF)

How the encoder encodes the key data

$\rightarrow$

0001 (0x1)	0010 (0x2)	0011 (0x3)	1010 (0xA)
0100 (0x4)	0101 (0x5)	0110 (0x6)	1011 (0xB)
0111 (0x7)	1000 (0x8)	1001 (0x9)	1100 (0xC)
1110 (0xE)	0000 (0x0)	1111 (0xF)	1101 (0xD)

How your `keypad_decode()` function remaps it.

Connections Shown in the Starting Schematic

The capacitors

Capacitors are devices that can hold charges over a small period of time. The capacitor on the Debounce (or “Keybounce mask”) pin is for smoothing the keypad signals to avoid switch bouncing making the CPU think the button was pressed multiple times. (Note: If you are unfamiliar with switch bouncing, read this article.) The capacitor on the Oscillator pin is for creating the timing for signal on the Data Available pin. The curved side of the capacitors is the negative side. On your capacitors, the negative side is often labeled with a black stripe, with minus sign(s), or with zero(s).

Data Available pin

The Data Available pin goes high when a button has been pressed, then returns low. So, we will have the microcontroller look for a rising edge on that pin. This signal will be connected to PC0, which we will attach to External Interrupt Line 0.

$\overline{\text{OUTPUT ENABLE}}$ pin

The *NOT* notation on this pin indicates that putting a *LOW* voltage on this pin will enable the output. A *HIGH* voltage will disable the output. Connect it to ground to leave it enabled.

Additional Connections You Will Add

Connect your status LEDs

- Connect a **red** LED to PA6.
- Connect a **yellow** LED to PA7.
- Connect a **green** LED to PA8.
- Connect a **blue** LED to PA9.

Note: Each LED should have a current limiting resistor between its positive lead (the long leg) and the microcontroller pin. Each LED should have its negative lead (the short leg) connected to ground.

Connect the keypad to the encoder chip

- Using the pinout images and the starting schematic from earlier, connect the four row pins and four column pins of the keypad to their respective pins on the encoder.

Connect the data pins

The encoder has 4 data pins (D0, D1, D2, D3; also called Data Out A, B, C, and D). The 4-bit value sent on these 4 pins ($D_3D_2D_1D_0$) represents the code for the key that was pressed.

- Connect D0 to PA0.
- Connect D1 to PA1.
- Connect D2 to PA4.
- Connect D3 to PB0.

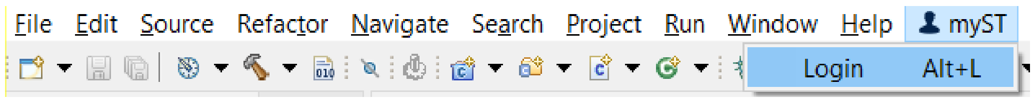
Power pins

- Connect the 5V pin of the Nucleo board to 5V pin of the encoder chip.
- There are several things that need to be connected to ground. You may connect a GND pin of the Nucleo board to a ground bus on your breadboard. Then, use the bus for the other ground connections.

Setting up the IOC to Generate the Configuration Code

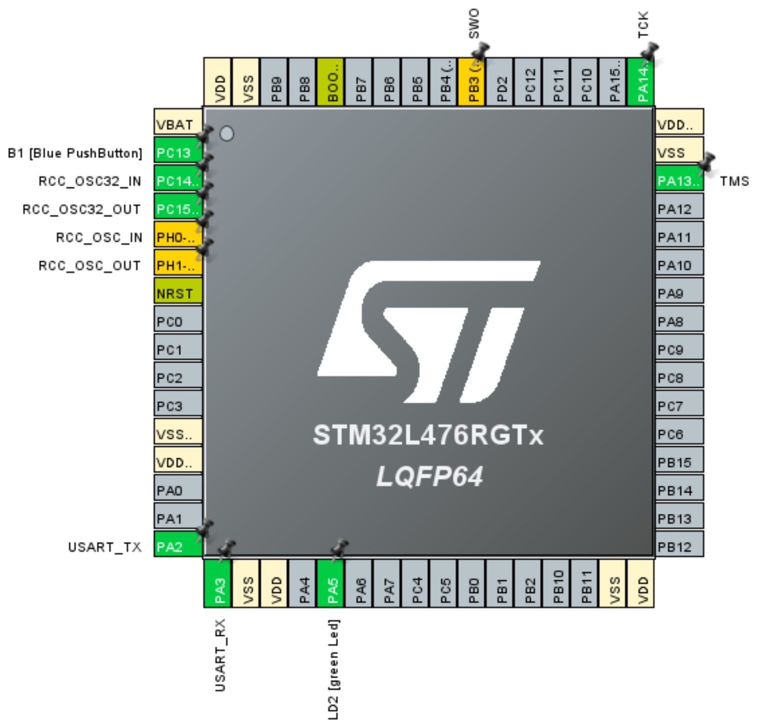
After you wire up the microcontroller, keypad, encoder chip, capacitors, LEDs, and resistors, you will use the IOC (I/O Configuration) tool in the STM32Cube IDE to generate code to configure your general-purpose inputs and outputs as well as your interrupts.

1. If you haven't already, you must make an ST account and login. In the IDE, go to mtST -> Login.



2. After you are logged in, create a new project:
    a. File -> New -> STM32 Project.
    b. In the "Board Selector" tab, select the "NUCLEO-L476RG" board.
    c. Click "Next."
    d. Name the project "Lab8"
    e. Under "Targeted Project Type," do **not** select "Empty" – leave it on "STM32Cube."
    f. Click "Finish."
    g. When asked "Initialize all peripherals with their default Mode?" select "Yes."
    h. If asked to switch perspectives, click "Yes."

The *Device Configuration Tool* (aka the IOC) should now be open, showing a pinout image of the chip (see below). Notice that PA5 is already configured for LED 2 (LD2) and PC13 is already configured for Button 1 (B1). Other pins also have a default configuration, such as PA2 as the transmit (Tx) pin and PA3 as the receiver (Rx) pin. The Tx/Rx pins will be relevant later in the semester.



You will use this tool to name and configure the pins. First, let's set up the interrupt signals: one for the blue button on PC13 (which will be the disarm button), and one for the DATA AVAILABLE signal on PC0.

3. **DISARM_BUTTON:** Click on **PC13**. Notice that it is set to "GPIO_EXTI13" (External Interrupt – Line 13). *Leave it* with that selection, but rename it by right-clicking on the pin and selecting "Enter User Label." Give it the name DISARM_BUTTON.
4. **DATA_AVAILABLE:** Click on **PC0**. Select "GPIO_EXTI0" (External Interrupt – Line 0). Then right click on the pin -> "Enter User Label," and give it the name DATA_AVAILABLE.

Choose which event (which "edge") you want the microcontroller to look for.

5. In the left pane, under "Categories," expand "System Core" and select "GPIO."
6. In the middle pane called "GPIO Mode and Configuration," select the "GPIO" tab under the "Configuration" header.
7. Select the PC0 pin. Underneath, verify that "GPIO mode" is set to "External Interrupt Mode with **Rising** edge trigger detection."
8. Select the PC13 pin. Underneath, verify that its "GPIO mode" is set to "External Interrupt Mode with **Falling** edge trigger detection."

Note: We don't need pull resistors in this lab. But if we did, you would also do that here in this pane.

We next need to enable the NVIC (Nested Vectored Interrupt Controller) for Line 0 and Line 13.

9. Still in the "GPIO Mode and Configuration" pane, switch from the "GPIO" tab to the "NVIC" tab.
10. Check the "Enabled" box for EXTI line0 interrupt.
11. Lines 10-15 are grouped together. Since line 13 is in that group, click the "Enabled" box for EXTI line[15:10] interrupts.

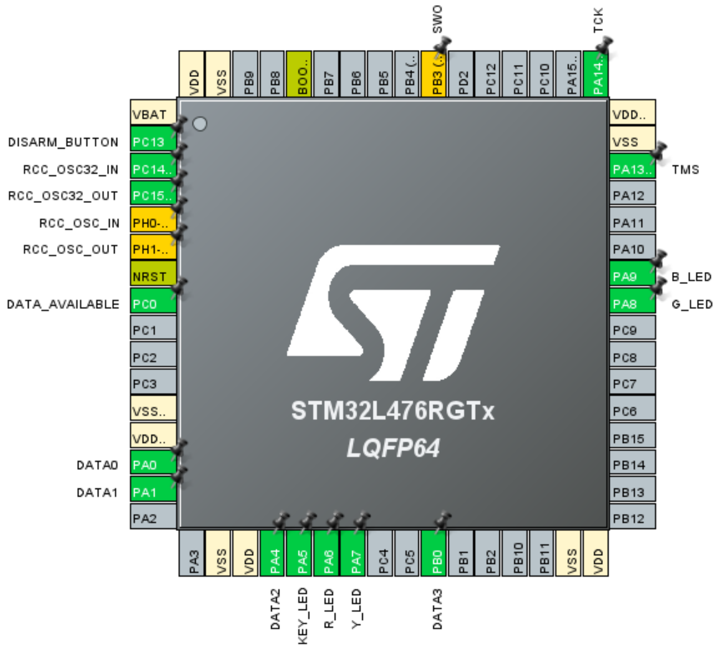
Next, we will set up the other input and output pins.

12. Back in the pinout diagram, select the following pins, set them each as GPIO_Input, and rename them with the following names:
    a. **PA0**: set as GPIO_Input and rename as **DATA0**
    b. **PA1**: set as GPIO_Input and rename as **DATA1**
    c. **PA4**: set as GPIO_Input and rename as **DATA2**
    d. **PB0**: set as GPIO_Input and rename as **DATA3**
13. In the pinout diagram, select the following pins, set them each as GPIO_Output, and rename them with the following names:
    a. **PA5**: rename as **KEY_LED**. (This is the onboard LED – LD2. We will use it for testing key presses. Note that it should already be a GPIO_Output, so just rename it.)
    b. **PA6**: set as GPIO_Output and rename as **R_LED**. (This is connected to your red LED.)
    c. **PA7**: set as GPIO_Output and rename as **Y_LED**. (This is connected to your yellow LED.)
    d. **PA8**: set as GPIO_Output and rename as **G_LED**. (This is connected to your green LED.)
    e. **PA9**: set as GPIO_Output and rename as **B_LED**. (This is connected to your blue LED.)

Since you won't be using PA2 and PA3 for their default communication configuration, go ahead and reset them:

14. Click on **PA2** and select "RESET_STATE."
15. Click on **PA3** and select "RESET_STATE."

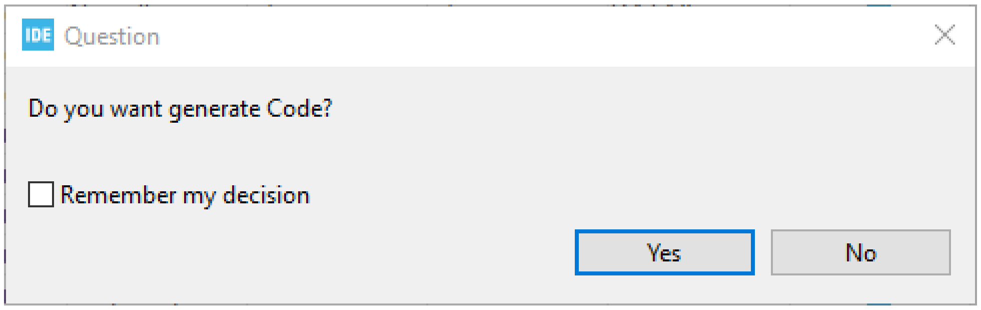
Your IOC pin diagram should now look like this:



Now, all the I/O pins are configured, and you can generate code from this IOC file that will use the HAL functions to correctly set up all the special function registers.

16. Save the file.

It should ask you:



Question

Do you want generate Code?

☐ Remember my decision

Yes No

17. Click "Yes." If you don't see this window, you can manually tell it to generate code by going to Project -> Generate Code. (If asked to switch perspectives, always select "Yes.")

After it generates the code, it should open the main.c file. In main.c, make the following changes:

18. In the “USER CODE BEGIN Includes” section, type the following “include” line:

```
#include <stdbool.h>
```

This will allow us to use the bool data type.

19. In the “USER CODE BEGIN PV” section, type the following global Private Variables (PV):

```
int password_index = 0, num_attempts = 0;
int current_password[10], temporary_password[10];
bool saving_password = false, entering_password = false, locked = false, alarm = false;
```

These global variables will allow the ISR to check and change the current state of the program.

20. In the “USER CODE BEGIN PFP” section, type the following Private Function Prototype (PFP):

```
uint8_t keypad_decode();
```

21. Last, in the “USER CODE BEGIN 4” section after main, add the following function definitions:

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {

    // This is where you will put the code that gets called when an interrupt occurs.

}

unsigned char keypad_decode() {

    // Place code here to read the key value, decode it, and return it.

}
```

You will finish writing these two functions to complete the lab. See the scaffolding code below to get started. You will use the `HAL_GPIO_WritePin` and `HAL_GPIO_ReadPin` functions we learned about last week.

Scaffolding Code

This link will take you to the scaffolding code for this week. Use this code as a starting point for the function definitions you just added above.

Debugging

This lab can be a difficult lab to debug when things aren't working correctly. To help, the code is set up to have the onboard LED (LD2 on PA5) light up whenever the microcontroller successfully receives an interrupt signal from the Data Available pin of the keypad encoder chip on PC0 (EXTI Line 0). It will stay on until you release the key – which can help you know if a key got stuck, which is very common. To unstick the keys, just bend the keypad.

Hardware or Software Issue? When the system doesn't work how you expect, you need to first identify if the issue is with the hardware or the software (or both!). To do so, you can "mimic" the Data Available signal from the encoder chip by giving the PC0 pin a "rising edge" – connect it to ground, then to a high voltage. If your interrupt is configured correctly, that should trigger an interrupt on EXTI Line 0, which then calls your `HAL_GPIO_EXTI_Callback` function and turns on LED2. You can set a breakpoint in the `HAL_GPIO_EXTI_Callback` function to see if that function is ever reached. If it is, you know your interrupt is being detected by the software. Then, you can step through the rest of that function to see what it is doing after receiving the interrupt signal. It may be helpful to pause *after* it reads the key with the `keypad_decode` function and see what value the `key` variable holds after calling that function. If the `key` variable shows the right value for each key press, you know your hardware is working correctly.

'A'-Level Project vs 'B'-Level Project

To turn your project from a 'B'-level project to an 'A'-level project, you must have the red LED **blink** when there is an alarm condition. You may not have it blink inside the Callback function or else the code will get stuck in the Callback function forever. Instead, you must have the Callback function change some global variable that indicates whether the LED should be blinking. Then, in the main while loop, you can check that global variable to see if you should call the `HAL_GPIO_TogglePin` function. You will also need a delay, for which you may use the `HAL_Delay(ms)` function, where `ms` is the number of milliseconds to delay.

Demonstration of Correct Operation

As soon as your system is working correctly, take a video demonstrating correct operation. Upload (1) the video and (2) your `main.c` file to the "Lab 08 Demo & Code" assignment in I-Learn. Don't forget to update the comments in your code before submitting. Please also add a comment to your submission indicating if your project is a 'B'-level or 'A'-level project.

Lab Report Instructions - Specifications

Starting with this week's lab report, your lab reports should now always include a "Specifications" section. Specifications are meant to be "specific" and should detail exactly what the system was designed to do and how it operates. So, use this section of your lab report to describe what your system does and how to use it. Be *specific*. In addition, include a parts list as a part of this section. Your specifications section should also disclose any known limitations (for example, what is the maximum password length?). Assume the reader understands microcontrollers but isn't aware of this specific alarm system.

Submit your report as a pdf to the "Lab 08 Report" assignment in I-Learn.