Using interrupts

Karthik Ganesan

Introduction

This lab will reacquaint you with interrupts, which you learned about in ECE243. The first part of this Lab uses interrupts to detect a button press on the board. For the second part, you will use a 4×4 keypad to input data on the board.

1. Part I: Button Interrupt

For this part, you use interrupts to check if the Blue pushbutton (labelled $B1$ on the board) has been pressed. As the button is an 'external' device (i.e, it is located outside the chip), pressing the button triggers an **external** interrupt. The board supports up to 23 external interrupt sources. Each interrupt is an extra wire going into your CPU. Thus, to save area, this board combines interrupts from several external sources into a single interrupt wire. In this case, interrupts from sources 10 to 15 are combined and an interrupt on any of these lines triggers the shared `EXTI_15_10` interrupt.
Start by downloading the provided `lab1` project into a new folder. Then, open the `stm32f4xx_it.c` file from the `Project` pane. This file contains default interrupt handlers for the various interrupts supported by this board. At the bottom of this file, you will see the `EXTI_15_10_IRQHandler`. You will see that an interrupt handler is already partially provided for you.
The first line checks if the interrupt was triggered by the pushbutton and not any of the other sources that share the same interrupt line. In our case, since the button is the only interrupt being triggered, we could skip this check. However, it is always good coding practise to check if we are responding to the right interrupt. For this, we use the `__HAL_GPIO_EXTI_GET_FLAG(USER_Btn_Pin)` function, which returns 1 if the button interrupt was triggered. After the interrupt is handled, we clear the interrupt flag using the `__HAL_GPIO_EXTI_IRQHandler(USER_Btn_Pin)` function. This will allow the handler to fire the next time the button is pressed. For part 1, you must write code to toggle the green LED each time the button is pressed.

Try it yourself: Modify the code from Lab 0 to switch the rate at which the LED blinks, each time the button is pressed. For example, you can loop between a 0.5s, 1s, and 2s delay between blinks.

2. Part II: Keypad

For this part, you will use the Grayhill Series 88 keypad available in the labs as an input device (shown in Figure 1).

2.1 Keypad Hardware

A keypad consists of a set of push buttons arranged on a grid with labels. Some manufacturers don't provide direct connections to each button, to minimize area and simplify connections. Instead, they use a circuit like shown on Figure 2[1], with individual wires for each row and column, and buttons placed at the intersections. In a keypad of N columns and N rows, the number of wires is reduced from N^2 to $2N$. However, such a keypad can only detect a single key being pressed a time.
As a trade-off, detection of individual key presses becomes more complex. One way to detect them is by connecting all columns to inputs of the board, and all rows to board outputs. By carefully controlling the rows and measuring the columns, you can identify individual key presses.
For the hardware present in our lab, if all rows are set to LOW, a network of pull down resistors causes all columns to read LOW regardless of their buttons' state. If **only one row is set to $HIGH$ at a time**, then the column board inputs are $HIGH$ only when the key in that specific row and column is pressed. By iterating over all rows and setting only one to $HIGH$ at a time, you can detect any key press from the keypad.

2.2 Keypad as an input device

Using the M-F jumper cables provided in the lab, make the connections shown in Figures 3 and 4.

[1]"4x4 Keypad Module", `https://components101.com/misc/4x4-keypad-module-pinout-configuration-features-datasheet`

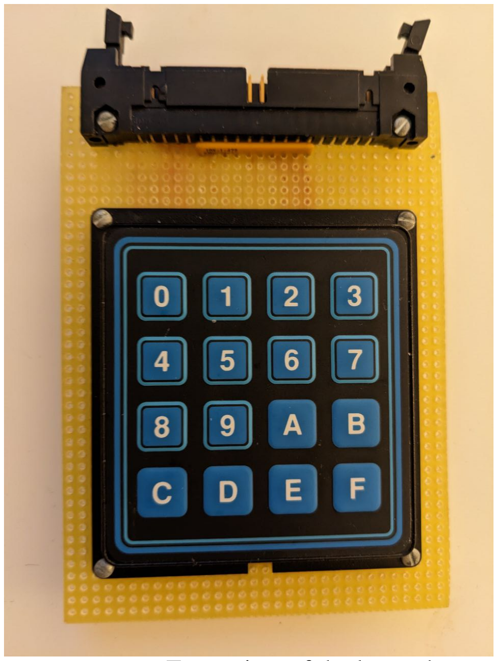

Figure 1. Front view of the keypad.

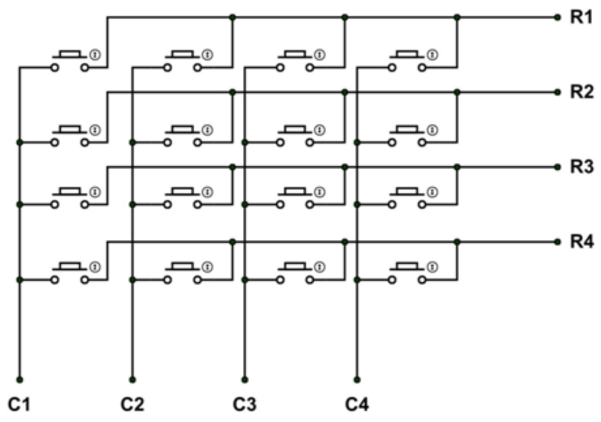

Figure 2. Internal structure of a 4×4 keypad

Write the loop in `main.c` to iterate over all rows, setting only one of them to *HIGH* at a time. You may benefit from a small delay between iterations, to leave room for transient voltage switches.
The project is preconfigured to use the labels `ROW0-3`, and `COL0-3` for inputs and outputs in the same way you used `LD2` in Lab 0.
Finally, write your interrupt handler code in `stm32f4xx_it.c`, `COL0` is handled in `EXTI9_5_IRQHandler()` and `COL1-3` are handled in `EXTI9_5_IRQHandler()`. These handlers should detect any individual key press and inform the main loop which column was triggered. Finally, you should output the name of the pressed key via serial port (UART).

Try it yourself: Sometimes when pressing keys you may notice the board detects as if it were pressed multiple times. This is a phenomenon known as *bouncing*, caused by mechanical and physical issues in push buttons. But this settles after a short time (typically a few tens of milliseconds). Based on this, how can you make the code robust to *bouncing*?

Try it yourself: Implement a simple calculator with the keypad as input and the serial port as the screen. Use the A-F keys for operations ($+ - * / =$).

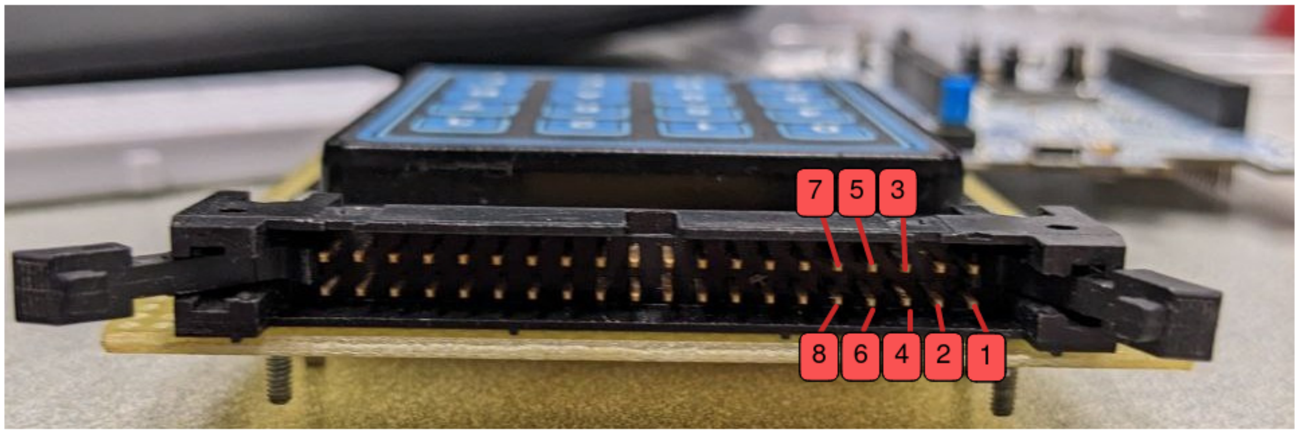

Figure 3

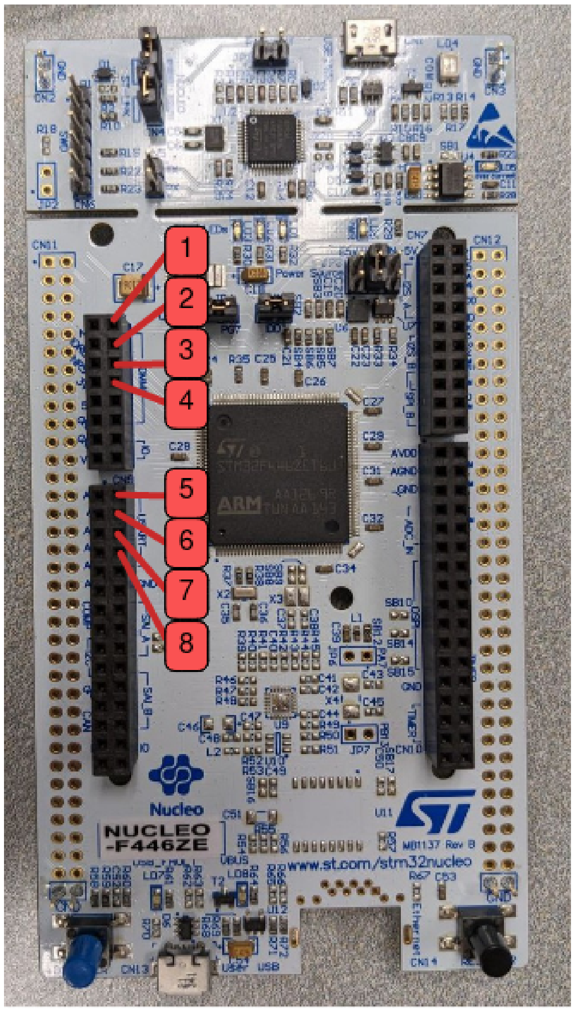

Figure 4