Embedded Real-Time Control, Alberto Morato

Laboratory 1, I^2C: Line sensor and keypad

Last update: 28-03-2024 Hash: 64d1555-dirty

1 Introduction

Purpose of this experience is to make you confident in interacting with an external device, connected to the microcontroller through a serial bus.

1.1 SX1509

The main device considered here is the sx1509, which provide 16 digital I/O and specific functionality like a keypad engine. Two sx1509 are connected to the microcontroller through I2C bus; both the devices are shares the same i2c line, in this case **i2c1**. For this reason, to the first sx1509, which we are going to call `sx1509_1`, is assigned the *slave address* `0x3E`, and the second one, which we are going to call `sx1509_2` is associated with the *slave address* `0x3F`.

- **sx1509_1** is used to interact with a line sensor (Pololu QTR Reflectance Sensor)
- **sx1509_2** is set as keypad engine and it is used to interact with a 16-key keypad.

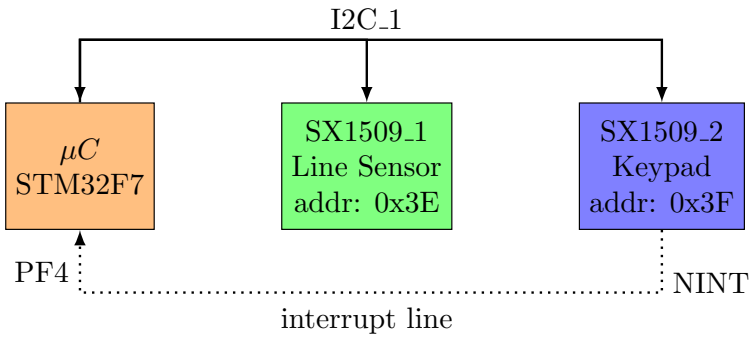

Figure 1: I2C Connection Schema

Sx1509_2 is capable of rising an interrupt if properly configured (see the datasheet for the details); the interrupt pin `NINT` is connected, respectively, to:

- PF4 (GPIO_EXTI4_KPAD_IRQ) for sx1509_2;

2 Preliminary operations

2.1 Import an existing project

You will be provided with a project that contains all the code necessary to interact with the sx1509 devices (mainly it deals with the initialization of the peripherals).

1. Download and unzip the project from Moodle.
2. Open STM32CubeIDE
3. File -> Import project from File System. In *Import source* select the folder containing the project. Click *Finish*.

2.2 Enable the interrupts

You will need to enable the interrupt source corresponding to `PF4` modify accordingly the `*.ioc` file. More information are in Laboratory 0.
Select *External Interrupt Mode with Falling edge trigger detection* and *No pull-up and no pull-down*.

Warning: Disable any interrupt for the pin `PF2`. Ensure that pin `PF2` is in **Reset** state.

2.3 HAL functions

Following are useful HAL functions for this lab. More detailed information [1].

1. `HAL_StatusTypeDef HAL_I2C_Mem_Read(I2C__HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout)`
2. `HAL_StatusTypeDef HAL_I2C_Mem_Write(I2C__HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout)`
3. `printf(const char *format, ...)`

2.4 SX1509 Registers

Following are some useful data registers of the SX1509. More detailed information in the datasheet.

- `0x10`: `REG_DATA_B` contains the data of the line sensor.
- `0x27`: `REG_KEY_DATA_1` Contains the status of the column of the keypad.
- `0x28`: `REG_KEY_DATA_2` Contains the status of the row of the keypad.

2.5 Keypad

The keypad is a 4x4 matrix, where each row is connected to a a set GPIO pins of the sx1509_2 and each column is connected to another set GPIO pin of the same device. The keypad is connected to the sx1509_2, and the status of the keys is stored in the registers `REG_KEY_DATA_1` (columns) and `REG_KEY_DATA_2` (rows).

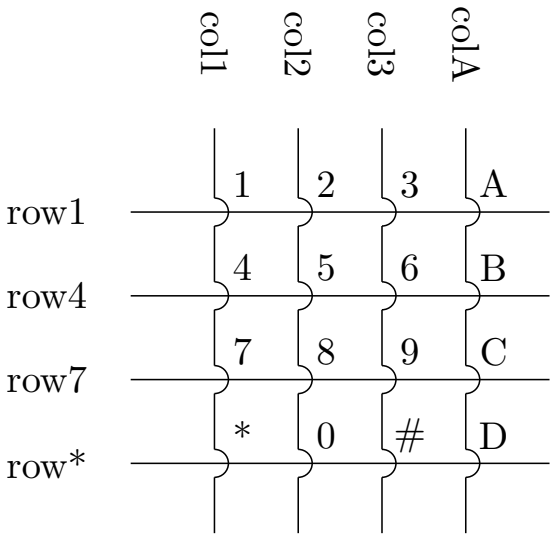

Figure 2: Keypad connection schema

Referring to Figure 2, the LSB (Least Significant Bit) of the `REG_KEY_DATA_1` corresponds to the column 1, the second bit to the column 2, and so on. The same applies to the `REG_KEY_DATA_2` register, where the LSB corresponds to the row 1, the second bit to the row 2, and so on.

Each line of the keypad is connected to a GPIO pin of the sx1509_2 using a pull–up resistor. So the default state of each bit in the GPIO registers will be `1`. In this condition. reading the two registers will give, supposing using an `uint8_t` to store the value, `REG_KEY_DATA_1 = REG_KEY_DATA_2 = 255`.

When a key is pressed, the corresponding row and column will be pulled to ground, so the value of the corresponding bit will be `0`. For example, if the key 1 is pressed, the value of the registers will be `REG_KEY_DATA_1 = 254` and `REG_KEY_DATA_2 = 254`.

Consider the example in Figure 3, where the key 8 is pressed. Than the value of the registers will be `REG_KEY_DATA_1 = 253` and `REG_KEY_DATA_2 = 251`.

[1] https://www.st.com/resource/en/user_manual/dm00189702-description-of-stm32f7-hal-and-lowlayer-drivers-stmicroelectronics.pdf

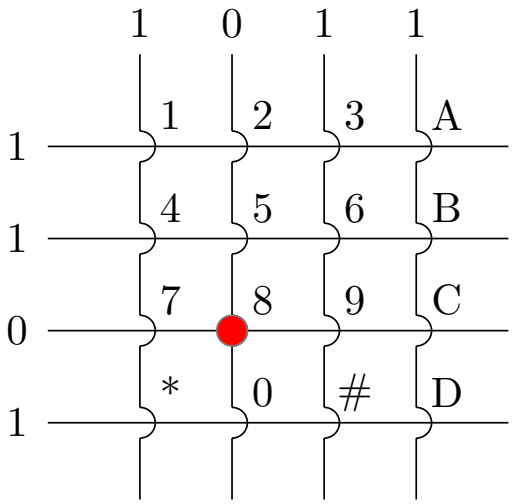

Figure 3: Keypad connection schema with key 8 pressed

3 Exercises

3.1 Exercise 1

Write an ISR that recognizes and correctly handles the keypad interrupts. You have to properly implement the `void HAL_GPIO_EXTI_Callback(uint16_t pin)` function. Then print which interrupt has been triggered, i.e., the `pin`. To be able to receive another interrupt from the keypad, have to read the registers `REG_KEY_DATA_1` and `REG_KEY_DATA_2` inside the ISR.

3.2 Exercise 2

Extend the code of exercise 1 to handle the keypad interrupt. You have to print which keypad button has been pressed.

3.3 Exercise 3

Write a routine that reads the status of the line sensor and prints it. The routine must check the status with a polling period of 100ms.

3.4 Exercise 4

Extend the code of LAB 0. Make one of the LED blink. If you use LEDs connected to `PE5` or `PE6`, check the `*.ioc` to make sure that those pins are set as `GPIO_output`. The blinking frequency should be set by the user through the keypad. There's two ways to do this:

1. Easy way: make a static mapping between the keypad buttons and the blinking frequency. For example, if the user presses the button 1, the LED should blink with a frequency of 1Hz. If the user presses the button 2, the LED should blink with a frequency of 2Hz, and so on. The mapping is up to you.

2. Hard way (Bonus): The frequency can be set "dynamically" by the user. For example, if the user presses `125#`, the LED should blink with a frequency of 125Hz. If the user presses `250#`, the LED should blink with a frequency of 250Hz, and so on.