ERTC - Laboratory 1

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 connected to the same microcontroller peripheral, 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 6 proximity sensors (GP2Y0D810Z0F) and 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.

Both sx1509 are capable of rising an interrupt if properly configured (see the datasheet for the details); the interrupt pin `NINT` are connected, respectively, to:

- PF2 (GPIO_EXTI2_PROXY_TOF_SENS_IRQ) for sx1509_1;
- 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 `PF2` and `PF4` modify accordingly the *.ioc file. More information are in Laboratory 0.

2.3 HAL functions

Following are useful HAL functions for this lab. More detailed information here.

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)`
    - Read an amount of data in blocking mode from a specific memory address
    - **hi2c** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
    - **DevAddress** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
    - **MemAddress** Internal memory address
    - **MemAddSize** Size of internal memory address
    - **pData** Pointer to data buffer
    - **Size** Amount of data to be sent
    - **Timeout** Timeout duration
    - **retval** HAL status
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)`
    - Write an amount of data in blocking mode to a specific memory address
    - **hi2c** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
    - **DevAddress** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
    - **MemAddress** Internal memory address
    - **MemAddSize** Size of internal memory address
    - **pData** Pointer to data buffer
    - **Size** Amount of data to be sent
    - **Timeout** Timeout duration
    - **retval** HAL status
3. `printf(const char *format, ...)`
    - Print a string on the standard output
    - **format** Format string
    - **...** Variable arguments
    - More informations about the printf function here

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.
- **0x19**: `REG_INTERRUPT_SOURCE_A` Interrupt source register. Reading this register you can check which proximity sensor has triggered the interrupt. Remember that you have to clear manually this register in the ISR by writing `0xFF` in it.
- **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.

3 Exercises

3.1 Exercise 1

Write an ISR that recognize and correctly handles the keypad interrupts. You have to properly implement the `void HAL_GPIO_EXTI_Callback(uint16_t pin)` function. Than print which interrupt has been triggered i.e. the `pin`. **To be able to receive another interrupt from the keypad, have to read thye 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. The blinking frequency should be set by the user through the keypad. There's two way to do this:

1. Easy way: make a static mapping between the keypad buttons and the blinking frequency. For example, if the user press the button 1, the led should blink with a frequency of 1Hz. If the user

press 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 press `125#` the LED should blink with a frequency of 125Hz. If the user press `250#` the LED should blink with a frequency of 250Hz, and so on.