Interfacing with Peripherals

Karthik Ganesan

Introduction

In this lab, you will be interfacing to an external Real Time Clock (RTC) module using the Nucleo platform. You will be using the DFR0151 module, which consists an RTC chip and an Electrically Erasable Programmable Read-Only Memory (EEPROM). EEPROM is a commonly used type of non-volatile ROM in embedded systems. For example, the memory in USB Flash drives is a type of EEPROM. To interface with the DFR0151, you will be using the I^2C communication protocol. At the end of this lab document, you will find a list of steps that you must show your TA when being marked during your in-lab demo.

I^2C Communication Bus

Inter-Integrated Circuit (I^2C) is a commonly used communication protocol in embedded systems. The I^2C protocol uses just two signals (or wires): a Serial Clock Line (SCL) and a Serial Data Line (SDL). I^2C supports multiple devices sharing the same bus. Each device is identified with a unique 7-bit address. Using I^2C, you can read or write to multiple addresses on each device using a single I^2C message.
You will learn more about the protocol and how it works in lectures. For this lab however, you will be using the I^2C functions provided by the STM Nucleo library, which abstract away a lot of the low-level complexity of the bus. You will use these functions to read and write to the DFR0151 module using I^2C.

Real-Time Clock

A real-time clock is an electronic device used to keep track of time on human scales in applications that rely on the current date and time. RTCs modules typically use independent power sources like batteries so they can continue running even when the main microprocessor is off. For example, an RTC (with a separate battery) is how your laptop still has the correct date and time, even if the battery drains to 0%. Internally, an RTC has a very accurate crystal oscillator which counts up the required time period (e.g., one second) and adds it to the stored value. For example, the DFR0151 module you will use in this lab can run on a single CR1220 battery for 3–5 years, according to the manufacturer.

1. Hardware Set Up

First, connect the module to the Nucleo board. Using jumper cables, make the connections shown in Figures 1 and 2.

1.1 Understanding the RTC module

The DFR0151 module stores the date and time using several memory-mapped registers which can be accessed using the I^2C bus. Table 1 shows the different registers, their address and what range of values they can hold. Each register is 8-bits in size.

Memory Address	Content	Value Range
0	Seconds	00-59
1	Minutes	00-59
2	Hours	1-12 / 0-23
3	Weekday	01-07
4	Date	01-31
5	Month	01-12
6	Year	00-99

Table 1. Registers for the RTC exposed via I^2C.

You must write all the functions to communicate with the RTC module in the provided `dfr0151.c` file. To get you started, some functions are already provided for you. For example, `rtc_read()` and `rtc_write()` use the built-in `HAL_I2C_Mem` functions to communicate over the I^2C bus.

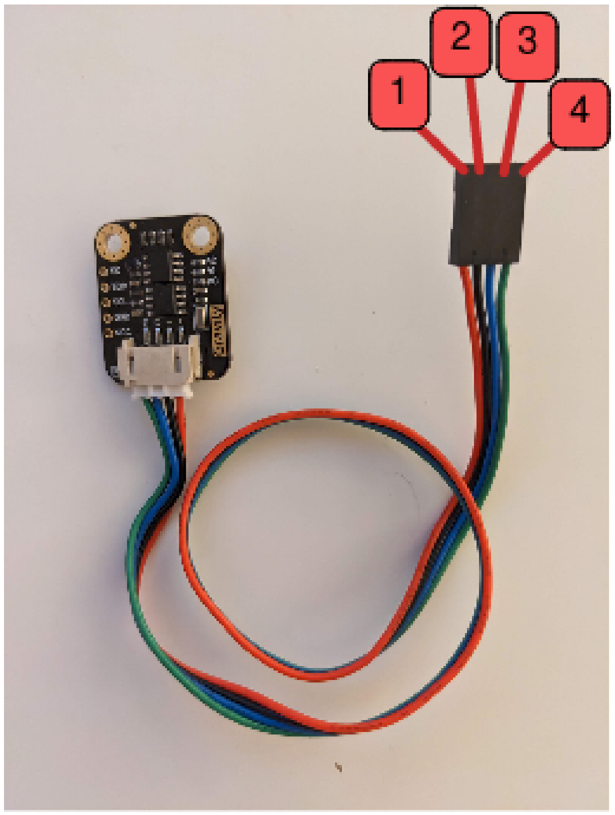

Figure 1

Both `HAL_I2C_Mem_Read` and `HAL_I2C_Mem_Write` functions accept the same parameters to perform read and write operations, respectively. As the names would suggest, you can use these to read and write to memory-mapped registers on an I^2C device. The parameters for both these functions, in sequence are:

1. **hi2c**: Pointer to a `I2C_HandleTypeDef` structure that contains the configuration information for the specified I2C.
2. **DevAddress**: The 7-bit address of the device.
3. **MemAddress**: The register on the device to be accessed.
4. **MemAddSize**: The size of the register to be accessed
5. **pData**: Pointer to data buffer for this operation.
6. **Size**: Amount of data to be sent/received.
7. **Timeout**: A timeout for the I^2C controller if there is no response from the target device.

If you are using Keil μVision or STMCubeIDE, you can also find this information by right-clicking these functions and clicking on 'Go to Definition'. You are also provided an `rtc_init()` function, which initializes the RTC module by writing to some specific register values. You do not need to modify this `rtc_init()` function.
All the values in the RTC registers are stored in Binary-Coded Decimal (BCD) format instead of simple binary. In the BCD format, the decimal values 0 – 9 are represented as 0b000 – 0b1010, while the binary numbers above this are not used. This makes it easy to convert a 4-bit binary number into decimal digits. You may use the provided `bcd2bin()` and `bin2bcd()` functions to translate values from BCD to regular binary.

1.2 Reading RTC Date/Time

As a first step, add code in `main()` to read and print the current date and time from the RTC using I^2C . Complete the functions `rtc_get_date()` and `rtc_get_time()` in file `dfr0151.c` to perform those operations. Note the parameters passed

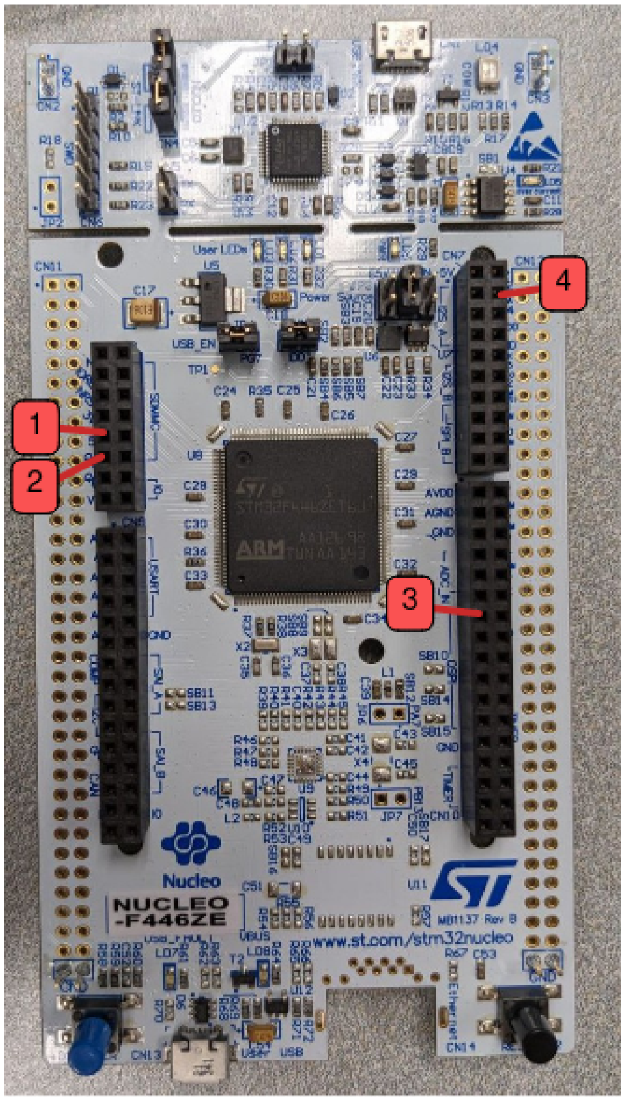

Figure 2

into each of these functions; these are the values you must read and return. Refer to the memory map provided earlier to see which registers you have to read for each of these functions. Then, using a single call to `HAL_I2C_Mem_Read()` read the set of registers required for `rtc_get_date()` and `rtc_get_time()`.

1.3 Setting RTC Date/Time

When you read the date and time, you will see that it is set to some default value by the manufacturer. Next, you should add functionality to set the date and time by completing the `rtc_set_date()` and `rtc_set_time()` in file `dfr0151.c`. Once again, you should only call `HAL_I2C_Mem_Write()` once per function. Then, modify `main()` to set an arbitrary date and time when button `B1` is pressed.

Try it yourself: What happens if you try to set an impossible date or time? Examples: 2023-01-32, 2023-02-29, etc.

2. Interacting with EEPROM

In addition to the RTC, the DFR0151 module also contains an EEPROM to store non-volatile application data. You will use the EEPROM to save the current time and date so they can be restored in the event of a power failure. The EEPROM is actually a separate I^{2}C device located on the same module, which shares the I^{2}C bus with the RTC chip.

Unlike the memory-mapped registers available for the RTC, the EEPROM addresses just correspond to memory locations. You can read and write to them just like you would with any other memory. Complete the functions `eeprom_write()` and `eeprom_read()` in `dfr0151.c`. You must take into account the following differences:

- The EEPROM uses the same I^{2}C as the RTC, but at a different device address (DevAddress). Use macro `ADDR_EEPROM` instead.
- The EEPROM uses 16 bit addresses instead of 8 bits like the RTC module.

To make use of the EEPROM, make two changes to your `main()` function:

1. When button `B1` is pressed, store the current date and time in addresses 0 – 6 of the EEPROM.
2. When the program starts, read this stored date and time and set this to be the RTC's current date and time.

> **Try it yourself:** Above, you are only saving the time to the EEPROM when the button is pressed. Instead, if you wanted to always save the current date and time, how often would you have to do that? What are the trade-offs with doing so instead of using the button?

3. Setting an Alarm

Finally, we will use the RTC to trigger an alarm. The time for the alarm should be stored ahead of time in EEPROM. When the alarm goes off, we will flash all three LEDs on the board.
Modify `main()` so that pressing button `B1` writes a time 1 minute later then the current time in addresses 7 – 13 of the EEPROM. This will be the time for the alarm to go off; when `B1` is pressed, you are setting an alarm for 1 minute later. You should be able to store both the current date/time plus the alarm with a single call to `HAL_I2C_Mem_Write()`. When your program starts, it should first read the alarm time from EEPROM and print it in a loop along with the current time. When the alarm time matches the current RTC time, the program must print the text `"ALARM"` and flash all three LEDs on for 1 second and off for 1 second for 10 seconds.

> **Try it yourself:** Instead of setting an arbitrary time, use the keypad from Lab 1 as an input device to control and set the current time and alarms.

4. In-lab Demo

During your lab session, you must show your TA the following:

1. Display the current RTC time and the current alarm through the serial port in a loop.
2. When `B1` is pressed, set the following:
   (a) RTC date to July 1st 2025 and time to 01:00:00 PM.
   (b) An alarm for July 1st 2025 at 01:01:00 PM.
3. After one minute, the alarm must be triggered and all three LEDs on the board must flash.
4. Disconnect and reconnect power from the RTC board (cable 1 from Figure 1) to emulate a power cycle. Restart the board with the reset button.
5. Show that the time is automatically reset to July 1st 2025 at 01:00:00 PM, and the alarm is set for 01:01:00 PM.

Your TA may ask questions about your solution. Be prepared to explain your design decisions and your code.