

Interfacing with Peripherals

Fabián Torres Álvarez, Karthik Ganesan, Prof. Bruno Korst - bkf@comm.utoronto.ca

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 can be used to store data even in the absence of power. For example, the memory in USB Flash drives is a type of EEPROM. To interface with the DFR0151, you will be using the I²C 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.

I²C Communication Bus

Inter-Integrated Circuit (I²C) is a commonly used communication protocol in embedded systems. I²C is a 2-wire protocol, with a Serial Clock Line (SCL) and a Serial Data Line (SDL). I²C is a multi-device bus, which means that any number of devices can share the bus. Each device is identified with a unique 7-bit address. Using I²C, you can read or write to multiple addresses on each device using a single I²C message.

You will learn about the timing of the I²C protocol in the lectures. For this lab, you will be using the I²C functions provided by the Nucleo library, which abstract away a lot of the low-level complexity of the bus. In this lab, you will learn how to use these functions to read and write to the DFR0151 module using I²C.

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. The DFR0151 module, according to the manufacturer, can run on a single CR1220 battery for 3–5 years.

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 I²C 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 I²C.

First, take a look at the provided `dfr0151.c` file, which provides some functions for you to get started with. `rtc_read()` and `rtc_write()` provide examples of using the built-in `HAL_I2C_Mem_*` functions to communicate over the I²C bus.

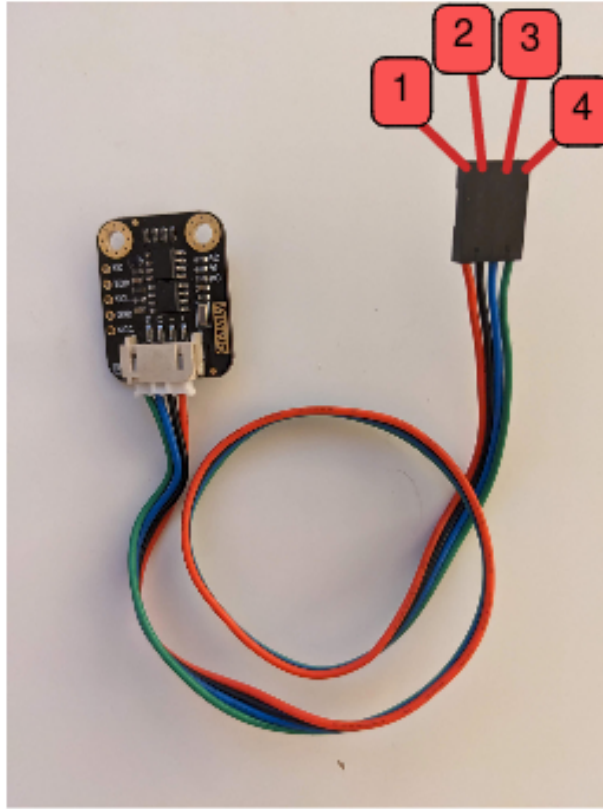


Figure 1

Both `HAL_I2C_Mem_Read` and `HAL_I2C_Mem_Write` functions accept the same parameters to perform read and write operations, respectively. 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 I²C controller if there is no response from the target device.

If you are using Keil μ Vision 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.

All the values in the RTC registers are stored in Binary-Coded Decimal (BCD) format instead of simple binary. 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 I²C. Complete the functions `rtc_get_date()` and `rtc_get_time()` in file `dfr0151.c` to perform those operations. You should be able to solve this with one call to `HAL_I2C_Mem_Read()` per function.

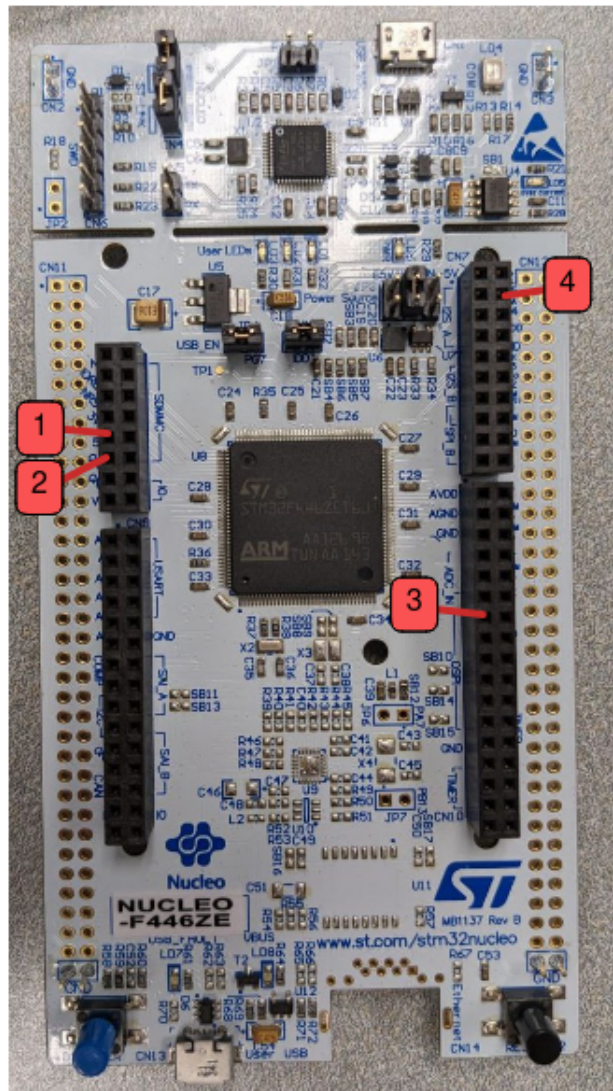


Figure 2

1.3 Setting RTC Date/Time

Next, you should modify the existing 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 exposes an EEPROM to store non-volatile application data**. **Non-volatile means that the data remains even when the device loses power**. We will use it in this section to store the current time and date so they can be restored in the event of a power failure. The EEPROM also exposes several memory-mapped registers which can be accessed via I²C. However, unlike the RTC, these registers are all simple memory locations, which can read and write to like you would with any other memory.

Complete the functions `eeeprom_write()` and `eeeprom_read()` in `dfr0151.c`. You must take into account the following differences:

- The EEPROM uses the same I²C 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 based on a time and date stored in EEPROM. When the alarm goes off, we will flash all three LEDs on the board.

Modify `main()` so that when button B1 is pressed, set an alarm time that is 1 minute later then the current time in addresses 7 – 13 of the EEPROM. You should be able to store both the current date/time plus the alarm with a single call to `HAL_I2C_Mem_Write()`. When the program starts it should 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 2023 and time to 10:00:00 AM.
 - (b) An alarm for July 1st 2023 at 10:01:00 AM.
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 2023 at 10:00:00 AM, and the alarm is set for 10:01:00.

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