UFMG

Application Note

Clock/Calendar implementation
on the STM32F10xxx and STM32F4 microcontrollers RTC

This Application Note was developed as a work in the discipline of Embedded Systems Programming at UFMG - Prof. Ricardo de Oliveira Duarte - Department of Electronic Engineering.

This application note describes the features of the RTC and how to configure it to implement several use cases such as calendar and timestamp.

In this document, the STM32 microcontroller terminology applies to the products listed in the following *Table 1*.

Table 1. Applicable products

Type	**Product series and part number**
Microcontrollers	STM32F4 Series, STM32F10x Series.

Software examples dedicated to this application note are then detailed to show how to handle the RTC feature for showing date and hour on a 128x64 OLED screen.

This dedicated software is provided through the STM32CubeMX embedded software package for setting the MCU and RTC configurations and the System Workbench for STM32 IDE for coding. It contains the source code of these examples and all the embedded software modules required to run the examples.

October 2020

Guilherme Amorim: guilherme.vini65@gmail.com
Renan Guedes: rbguedes1998@gmail.com

Contents

1 Introduction **3**

2 RTC main features **4**

3 Clock/calendar functionality features **5**

3.1 Clock/calendar basics 5

3.2 Implementing the clock function on the medium-density STM32F10xxx 5

3.3 Implementing calendar function on the medium-density STM32F10xxx 6

4 Firmware description **8**

5 Hardware setup **9**

5.1 STM32F103C8T6 9

5.2 STM32F401RE 12

6 Coding the RTC **16**

7 Results **18**

8 References **19**

1 Introduction

A real-time clock (RTC) is a computer clock that keeps track of the current time. Although the RTCs are often used in personal computers, servers and embedded systems, they are also present in almost any electronic device that requires an accurate time keeping. The microcontrollers supporting the RTC can be used for chronometers, alarm clocks, watches, small electronic agendas, and many other devices.

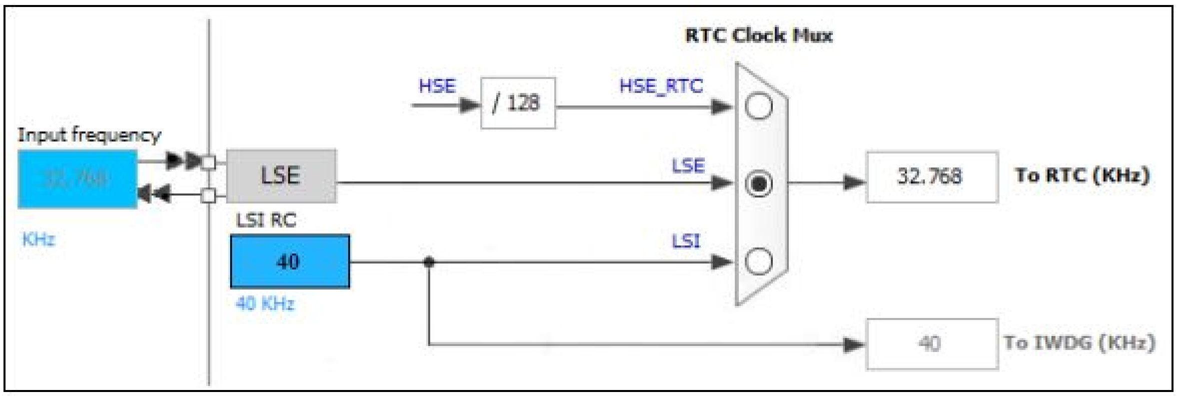
Basically a RTC is a timer-counter but unlike other timers of a MCU it is much more accurate. In most 8-bit MCUs like the regular PICs and AVRs, there are no built-in RTC modules and so we need to use dedicated RTC chips like the popular DS1302 or PCF8563 when we need an on-board precise time-keeping device. Those chips also need some additional circuitry, wiring and circuit board space.

At present, however, most modern MCUs come packed with literally every possible hardware a designer may think of. STM32 MCUs come with built-in RTC modules that require no additional hardware support. This AN covers basic features of STM32's internal RTC and how to use it for time-keeping applications.

2 RTC main features

- Programmable prescaler: division factor up to 220
- 32-bit programmable counter for long-term measurement
- Two separate clocks: PCLK1 for the APB1 interface and RTC clock (must be at least four times slower than the PCLK1 clock)
- The RTC clock source could be any of the following ones:
  - HSE clock divided by 128
  - LSE oscillator clock
  - LSI oscillator clock

Figure 1. RTC clock source options



- Two separate reset types:
  - The APB1 interface is reset by system reset
  - The RTC Core (Prescaler, Alarm, Counter and Divider) is reset only by a Backup domain reset.
- Three dedicated maskable interrupt lines:
  - Alarm interrupt, for generating a software programmable alarm interrupt.
  - Seconds interrupt, for generating a periodic interrupt signal with a programmable period length (up to 1 second).
  - Overflow interrupt, to detect when the internal programmable counter rolls over to zero.

3 Clock/calendar functionality features

3.1 Clock/calendar basics

A real-time clock keeps track of the time (hours, minutes, seconds) and date (day, month, year). It should also take into account leap years. A leap year is a year where an extra day is added to the calendar in order to synchronize it to the seasons. Since the tropical year is 365.242190 days long, a leap day must be added roughly once every 4 years (4 × 0.242190 = 0.968760). Thus, every four years, the month of February counts 29 days instead of 28.

3.2 Implementing the clock function on the medium-density STM32F10xxx

The used registers are RTC registers: prescaler register, counter register and alarm register.

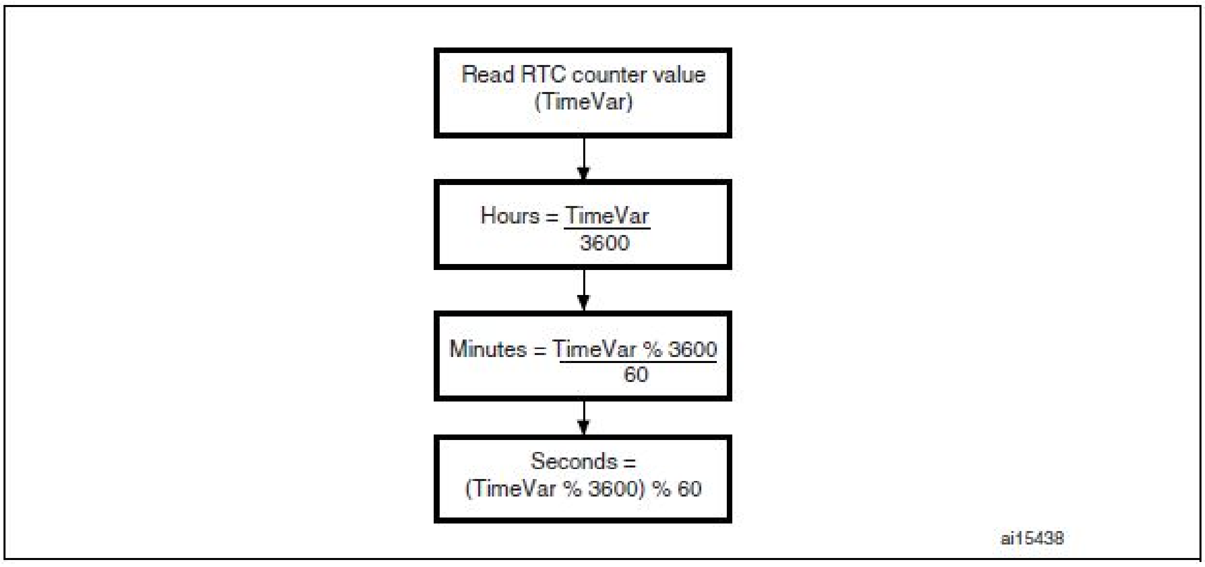
The medium-density STM32F10xxx RTC peripheral consists of a chain of programmable counters consisting of two main units:

● the RTC prescaler block generates the RTC time base. Depending on the clock period applied to the LSE input, this prescaler can be adjusted to generate a time base (RTC_CLK) of 1 second by writing to the prescaler load register.

● the 32-bit programmable counter can be initialized to the current system time. The system time is incremented at the RTC_CLK rate and compared with the alarm register value. When it reaches the alarm register value, an interrupt is generated.

The current system time can be inferred from the value in the counter register by following the steps shown in Figure 2 ("/" implies division, "%" implies modulus)

Figure 2. Time system flowchart


Read RTC counter value (TimeVar)
Hours = TimeVar / 3600
Minutes = TimeVar % 3600 / 60
Seconds = (TimeVar % 3600) % 60
ai15438

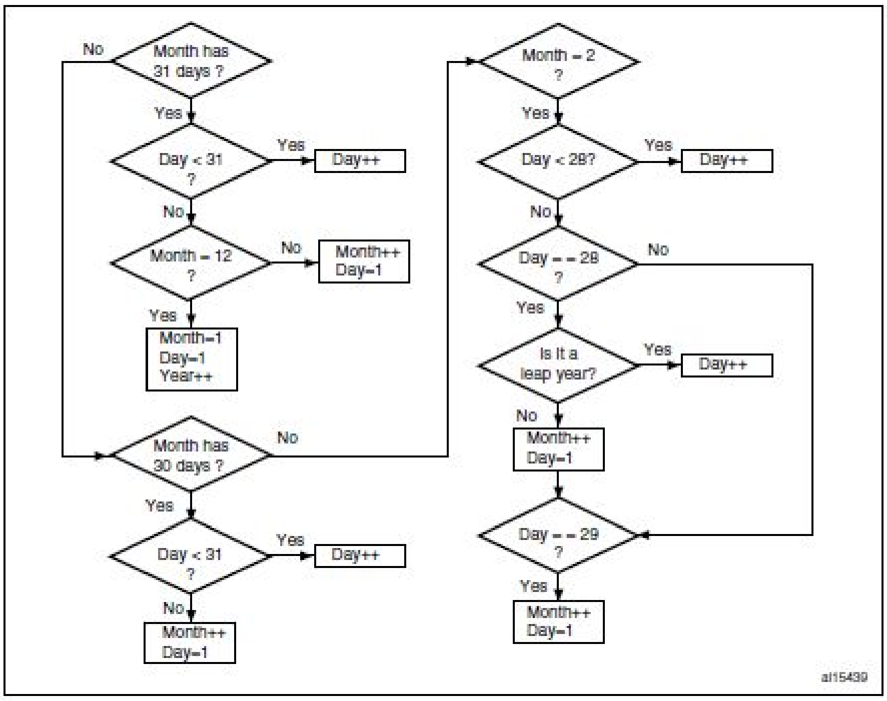
3.3 Implementing calendar function on the medium-density STM32F10xxx

The used registers are the 16-bit backup data registers.

When the 32-bit counter register value reaches 86399, this means that one day has elapsed, and that the date has to be updated. The counter value is then reset. Whenever the system date is updated, the new date is stored into the 16-bit backup data registers so that the system remains at the current date even after system reset, power reset or wake up from the Standby mode.

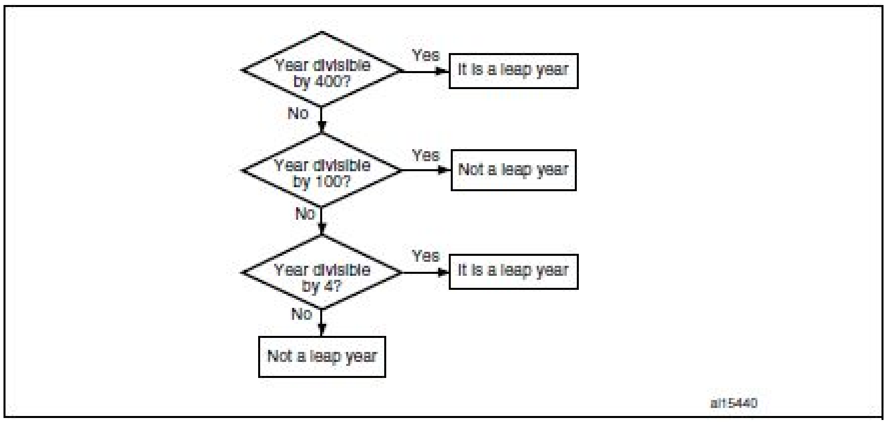
The calendar algorithm given in Figure 3 can be used to develop the date update function.

Figure 3. Calendar algorithm



Leap year correction: each time the date is updated, the possibility of a leap year also has to be considered (please refer to Figure 4).

Figure 4. Leap year flowchart



4 Firmware description

The example firmware consists of a clock/calendar driver that includes all the functions needed to realize a clock and calendar application on a 128x64 OLED screen. The source code example is based on the STM32F10xxx standard peripheral library.

User include files:

- ssd1306.h: contains the prototypes of the basic functions used to implement the ssd1306 library features functions for drawing lines, rectangles and circles. It also allows you to draw texts and characters using appropriate functions provided on it.
- fonts.h: contain some data structure definitions and FONTS_GetStringSize prototype.
- stm32fxxx_hal_rtc.h: contains the prototypes of the basic functions used to implement the internal RTC. Also includes some macros for summer-time correction, system start default time and date.

User source files:

- ssd1306.c: Contains all the ssd1306.h library functions' implementations. Examples of functions' usages are updating the screen's information by cleaning the screen with SSD1306_UpdateScreen, choosing the location of where the information will be written with SSD1306_GotoXY and writing the content on the screen with SSD1306_Puts.
- fonts.c: includes various OLED font size definitions and implements the FONTS_GetStringSize function.
- stm32fxxx_hal_rtc.c: Provides firmware functions to manage functionalities of the Real Time Clock peripheral such as: initialization and de-initialization functions, RTC time and date functions, RTC alarm functions, peripheral control functions and peripheral state functions.

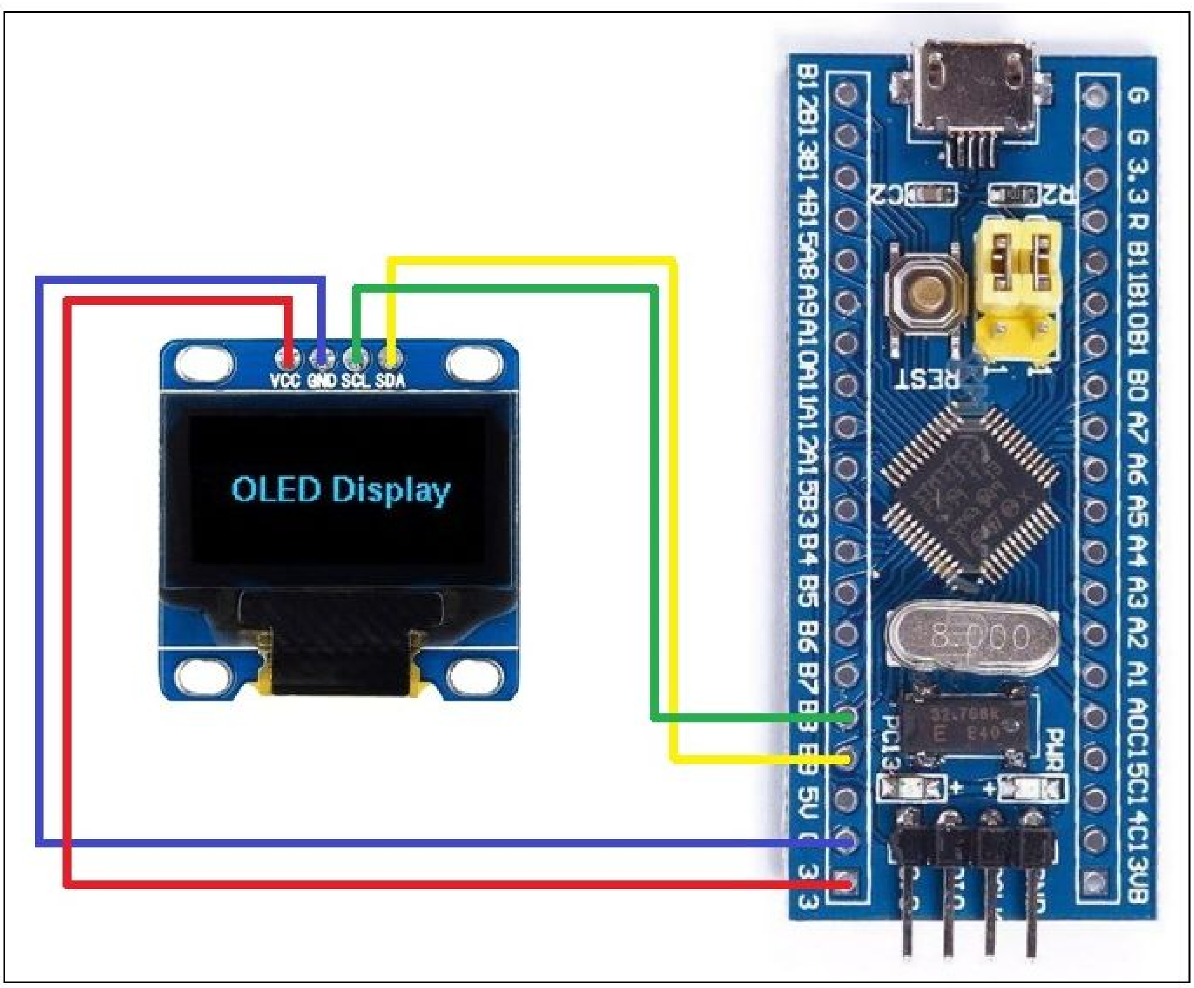
You can download ssd1306.h, fonts.h, ssd1306.c and fonts.c on the following link: https://github.com/guiguitz/STM32-Display-OLED-128x64-API/

5 Hardware setup

5.1 STM32F103C8T6

In order to use the application example project, the user needs a STM32F103C8T6 board (Blue Pill).

Figure 5. STM32F103C8T6 (Blue Pill board) connected to a OLED 128x64

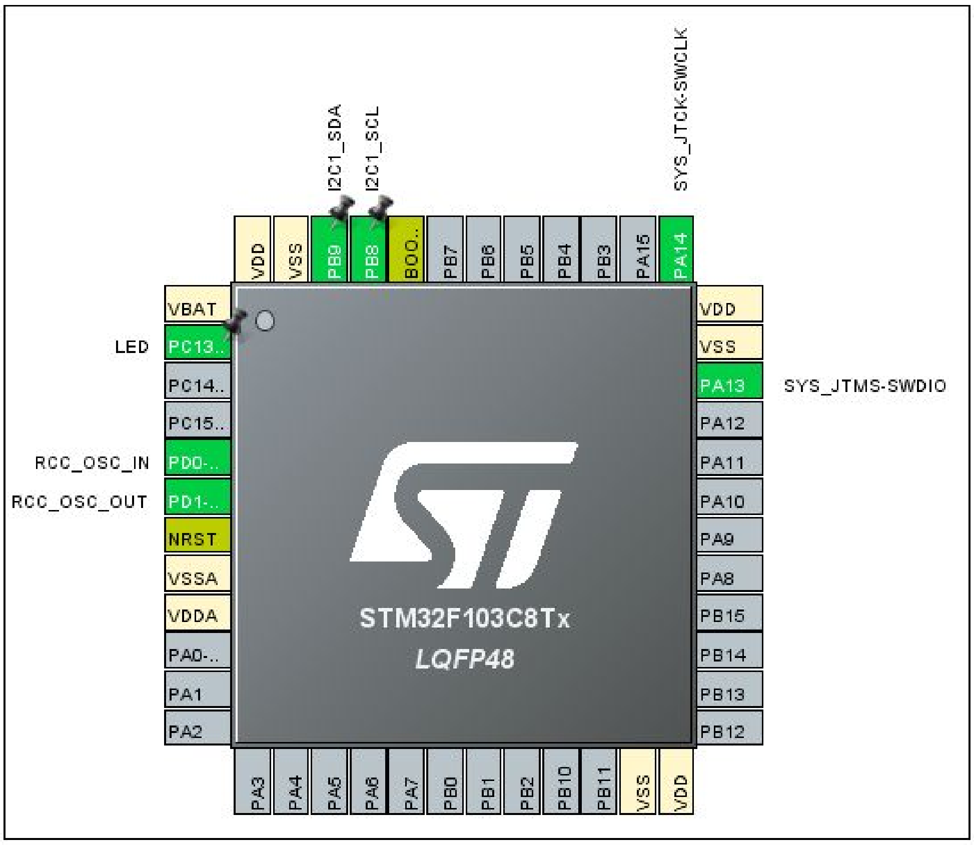

VCC GND SCL SDA
OLED Display
B12 B13 B14 B15 A8 A9 A10 A11 A12 A15 B3 B4 B5 B6 B7 B8 B9 5V G 3.3
G G 3.3 R B11 B10 B1 B0 A7 A6 A5 A4 A3 A2 A1 A0 C15 C14 C13 VB
RESET
8.000
PC13
PWR

Supply the board using the USB cable or 3,3V supply. For this example, the user needs to:

- Connect the OLED Vcc pin on Blue Pill 3.3V pin;
- Connect the OLED GND pin on Blue Pill GND pin;
- Connect the OLED SCL pin on Blue Pill B8 pin;
- Connect the OLED SDA pin on Blue Pill B9 pin;

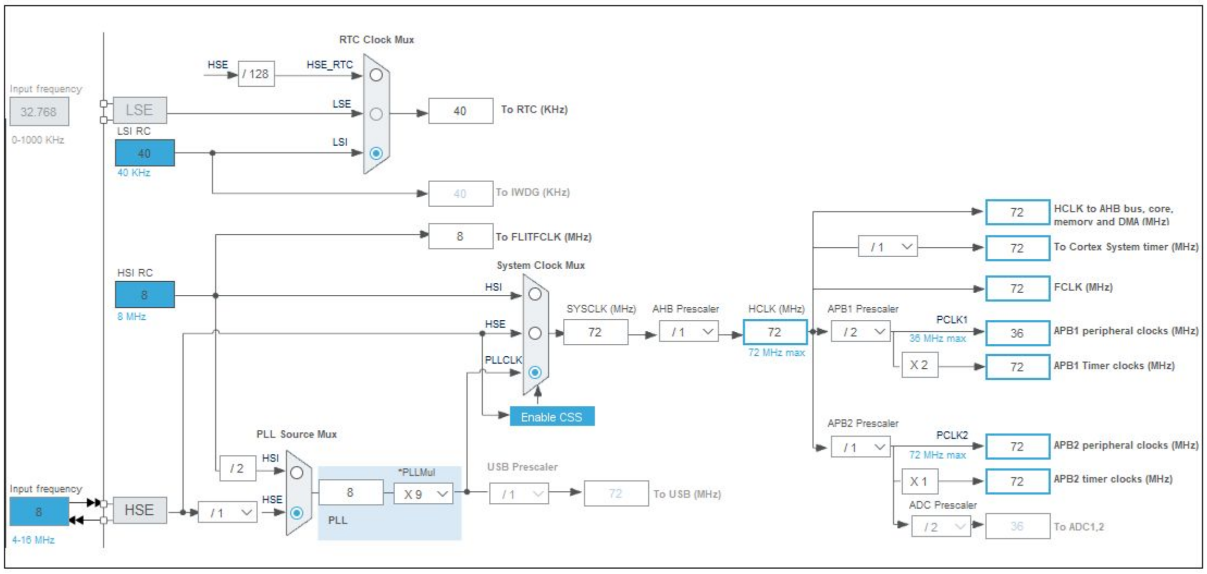
The CubeMX Pinout setup is as follows:

Figure 6. CubeMX Pinout setup



The CubeMX Clock Configuration setup setup is as follows:

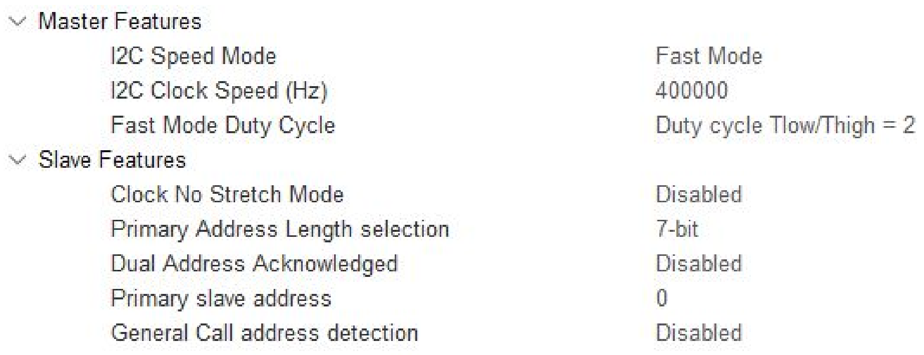
Figura 7. CubeMX Clock Configuration setup



The Blue Pill board has an 8MHz external crystal for RCC. Using PLL, the clock speed is multiplied to generate a 72 MHz system clock.

The I2C (OLED communication protocol) speed used is 400kHz and was setup as follows:

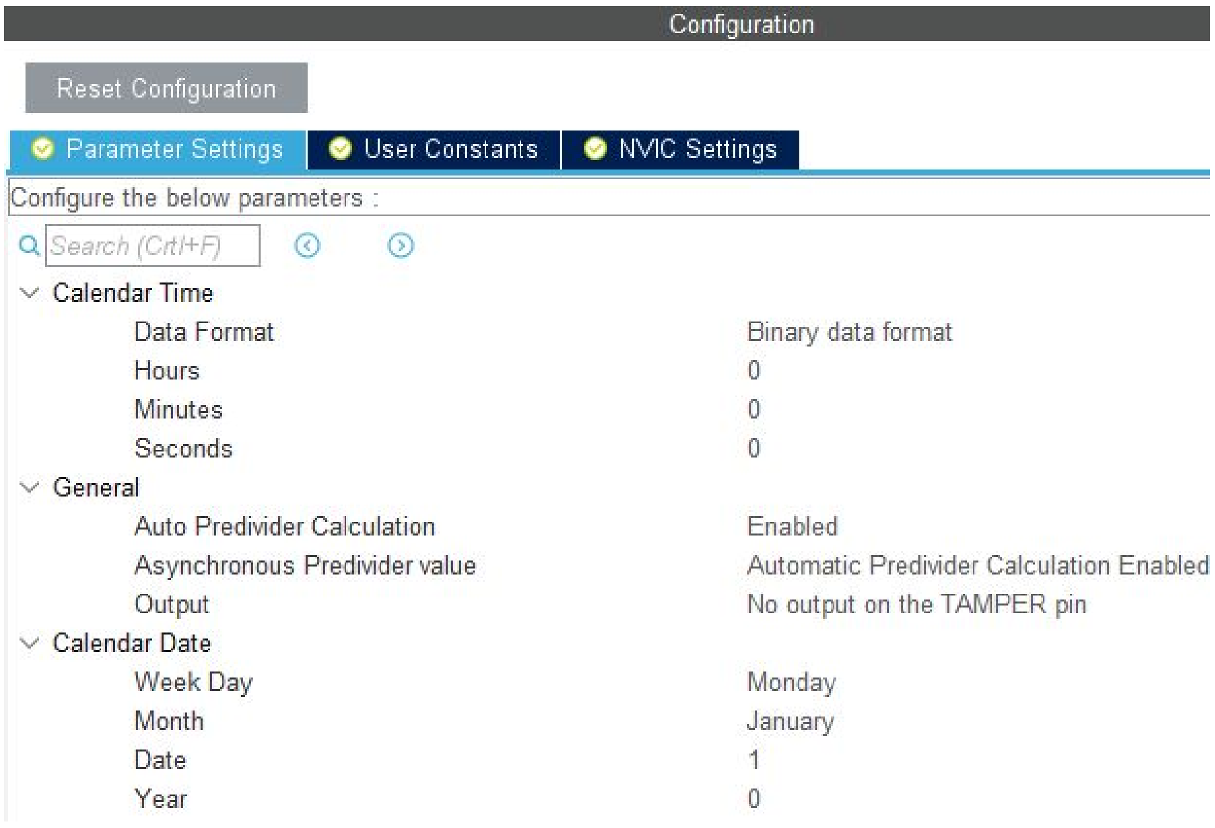
Figure 8. CubeMX I2C Configuration Setup



Master Features	
I2C Speed Mode	Fast Mode
I2C Clock Speed (Hz)	400000
Fast Mode Duty Cycle	Duty cycle Tlow/Thigh = 2
Slave Features	
Clock No Stretch Mode	Disabled
Primary Address Length selection	7-bit
Dual Address Acknowledged	Disabled
Primary slave address	0
General Call address detection	Disabled

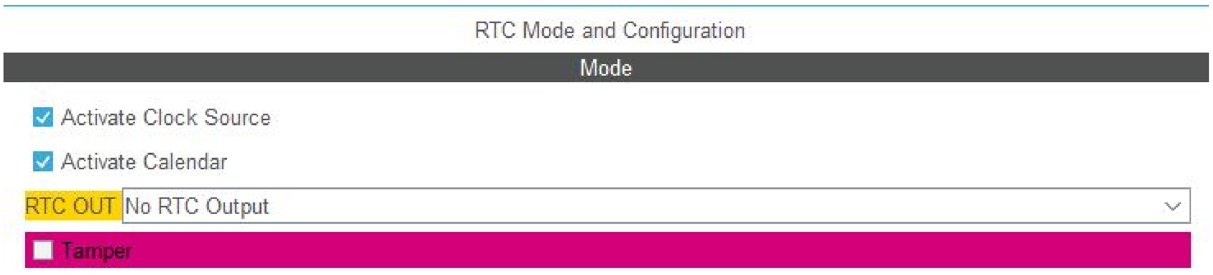
The CubeMX RTC setup is as follows:

Figure 9. CubeMX RTC configuration



Calendar Time	
Data Format	Binary data format
Hours	0
Minutes	0
Seconds	0
General	
Auto Predivider Calculation	Enabled
Asynchronous Predivider value	Automatic Predivider Calculation Enabled
Output	No output on the TAMPER pin
Calendar Date	
Week Day	Monday
Month	January
Date	1
Year	0

Figure 10. CubeMX RTC mode configuration


RTC Mode and Configuration

Mode

✓ Activate Clock Source

✓ Activate Calendar

RTC OUT: No RTC Output

☐ Tamper

Note:

To keep tracking time and calendar when the MCU is turned off, the user can add a power supply backup. This backup can be achieved in various ways. Two common backup sources are:

- Battery – small NiMH, NiCd, Li ion or Lipo cells.
- Supercapacitor – 0.22F to 1F.

To have more information, access the following link: STM32'S INTERNAL RTC.

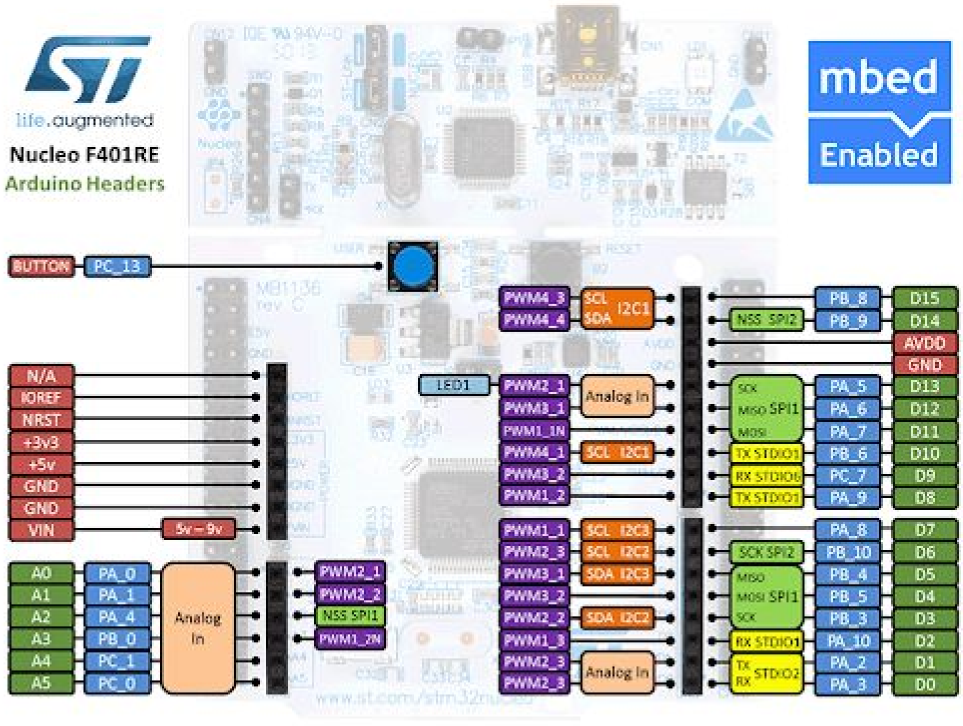
5.2 STM32F401RE

In case of using a STM32F401RE board, a NUCLEO-F401RE is needed and the following connection must be made:

- Connect the OLED Vcc pin on NUCLEO-F401RE 3.3V pin;
- Connect the OLED GND pin on NUCLEO-F401RE GND pin;
- Connect the OLED SCL pin on NUCLEO-F401RE PB8 pin;
- Connect the OLED SDA pin on NUCLEO-F401RE PB9 pin.

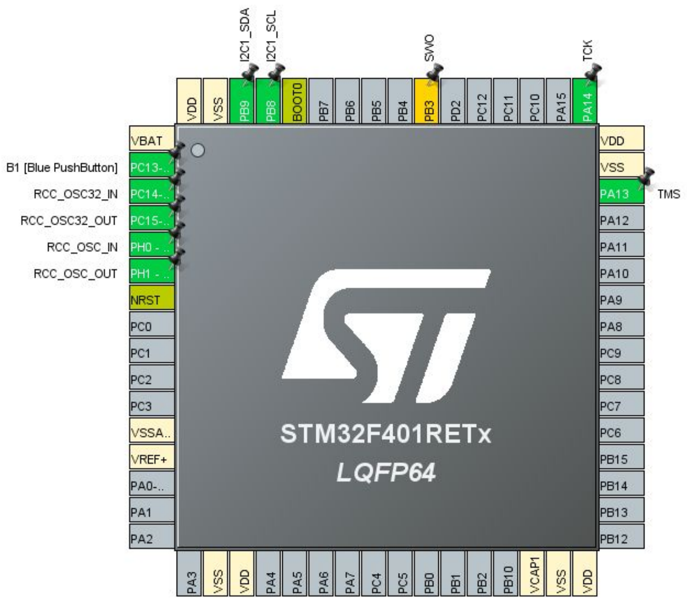
The NUCLEO-F401RE pins' physical locations and its functions are shown in Figure 11:

Figure 11. NUCLEO-F401RE pin configuration



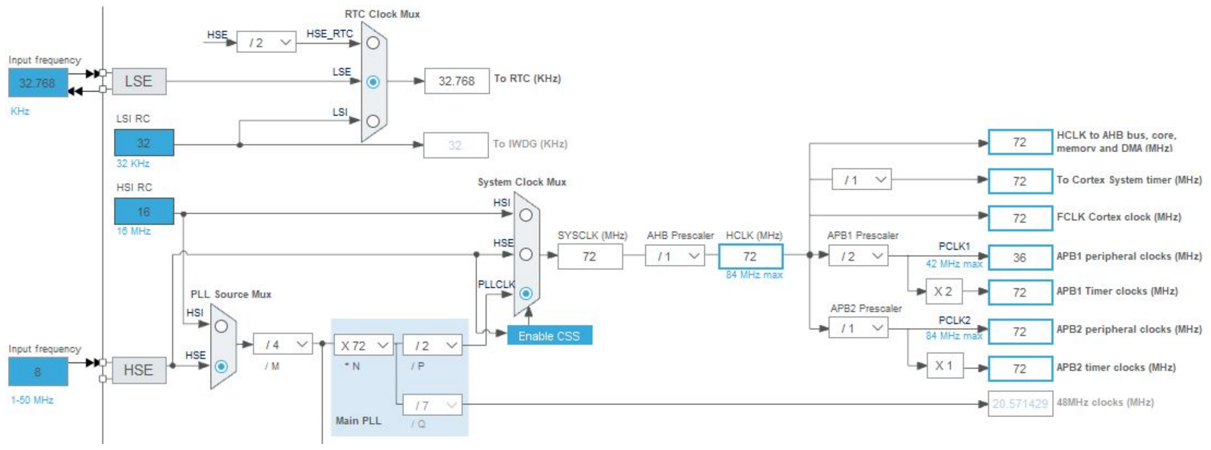
For setting the pinouts' configuration, the STM32CubeMX software is used and its setup is as shown on Figure 12.

Figure 12. NUCLEO-F401RE CubeMX pinout setup



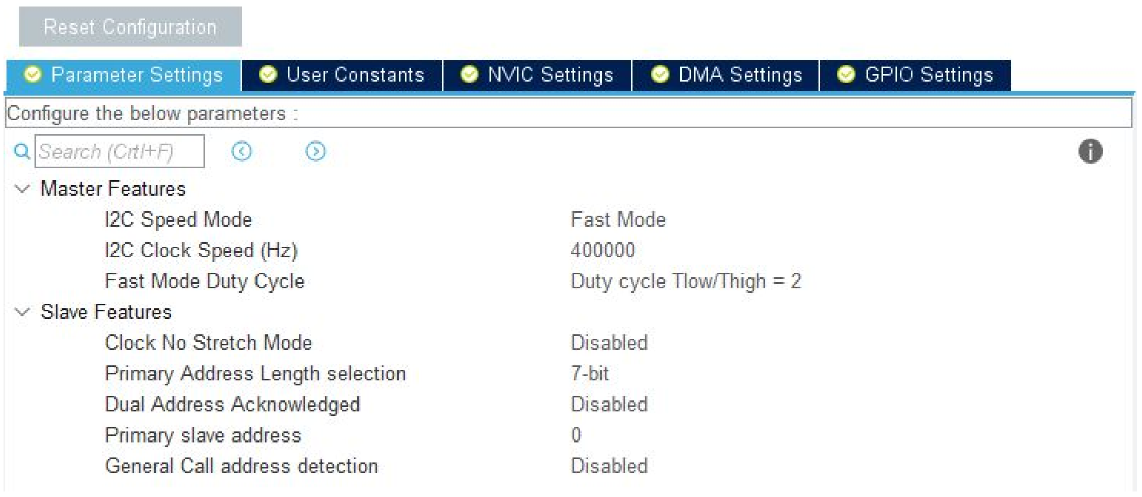
On Figure 13, the NUCLEO-F401RE clock tree configuration is shown.

Figure 13. NUCLEO-F401RE CubeMX clock tree configuration



The 128x64 OLED display has its own controller chip SSD1306 and it has an I²C communication interface. Thus, for making the application run correctly, it is necessary to set up the I²C configuration on CubeMX as shown on Figure 14.

Figure 14. NUCLEO-F401RE CubeMX I²C configuration

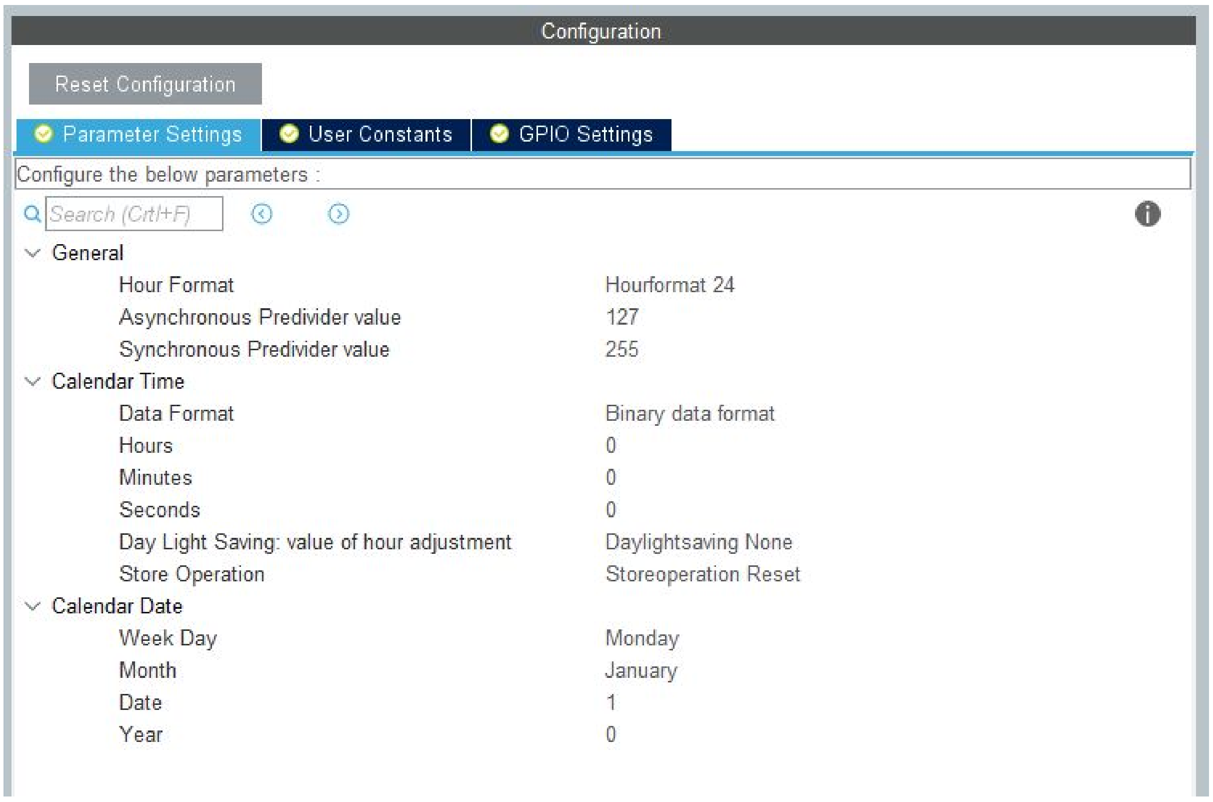


Configure the below parameters :

Master Features	
I2C Speed Mode	Fast Mode
I2C Clock Speed (Hz)	400000
Fast Mode Duty Cycle	Duty cycle Tlow/Thigh = 2
Slave Features	
Clock No Stretch Mode	Disabled
Primary Address Length selection	7-bit
Dual Address Acknowledged	Disabled
Primary slave address	0
General Call address detection	Disabled

Finally, the RTC is configured as follows on Figure 15.

Figure 15. NUCLEO-F401RE CubeMX RTC configuration



General	
Hour Format	Hourformat 24
Asynchronous Predivider value	127
Synchronous Predivider value	255
Calendar Time	
Data Format	Binary data format
Hours	0
Minutes	0
Seconds	0
Day Light Saving: value of hour adjustment	Daylightsaving None
Store Operation	Storeoperation Reset
Calendar Date	
Week Day	Monday
Month	January
Date	1
Year	0

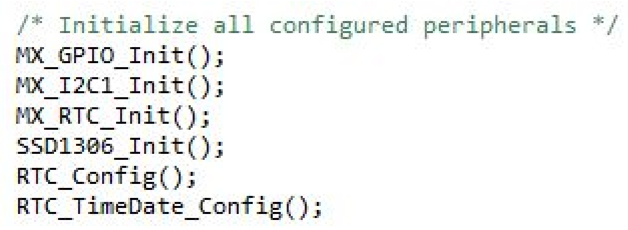
6 Coding the RTC

The Blue Pill and NUCLEO-F401RE Application projects can be downloaded on the following link: https://github.com/guiguitz/RTC-OLED-Application.

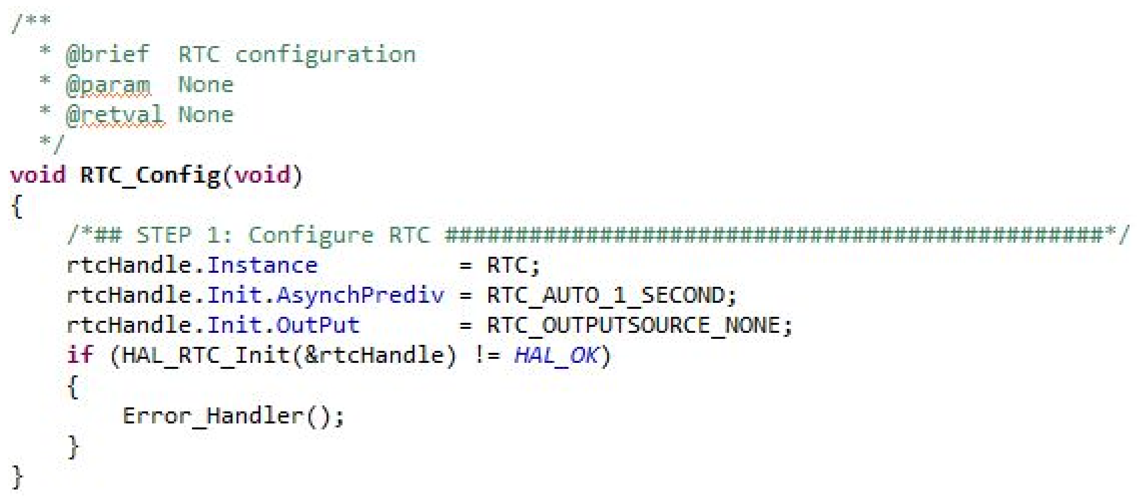
The application is based on the following four steps, which the steps 3 and four are in the infinite loop:

1. Initializes peripherals;
2. Setting the RTC;
3. Read the current time and date;
4. Puts current time and date in OLED 128x64.

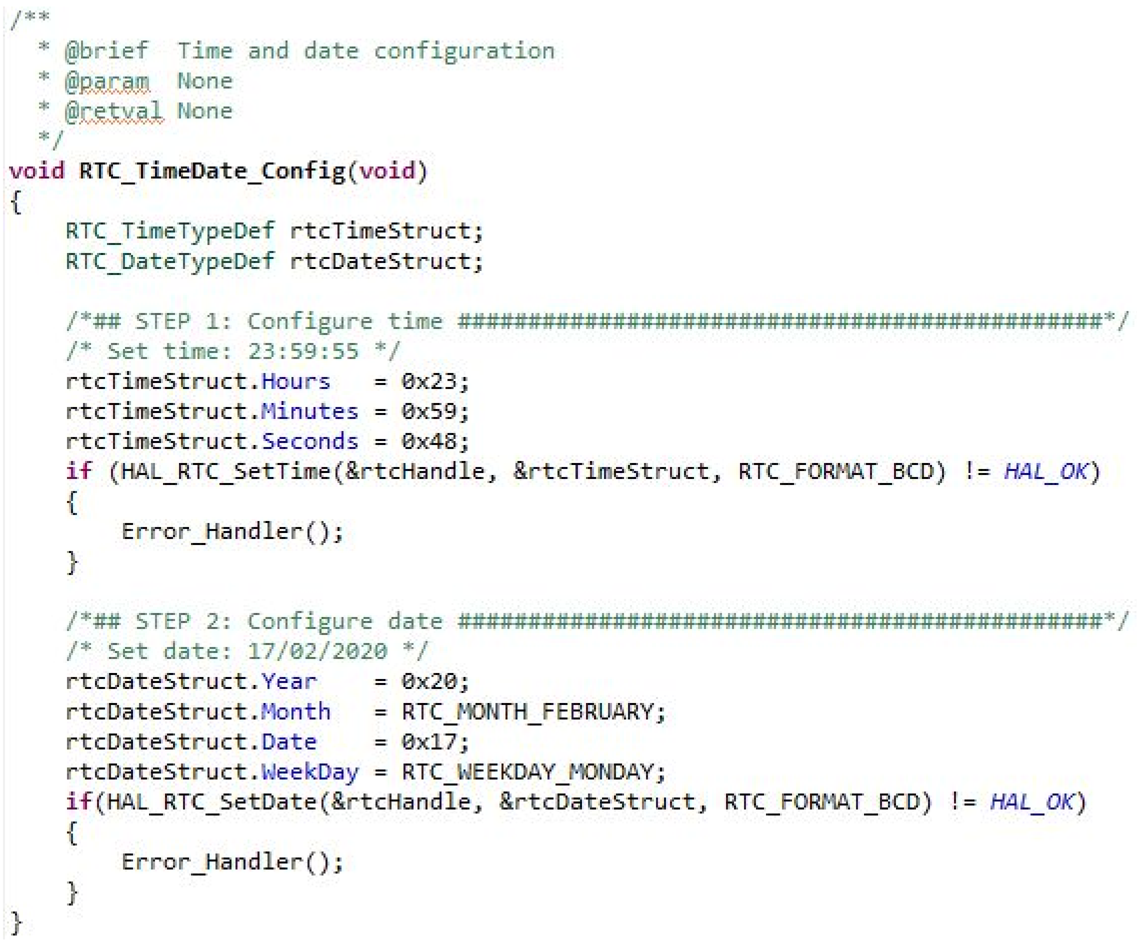
Figure 16. Code of first and second step



```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_I2C1_Init();
MX_RTC_Init();
SSD1306_Init();
RTC_Config();
RTC_TimeDate_Config();
```



```
/**
  * @brief  RTC configuration
  * @param  None
  * @retval None
  */
void RTC_Config(void)
{
    /*## STEP 1: Configure RTC ###############################################*/
    rtcHandle.Instance          = RTC;
    rtcHandle.Init.AsynchPrediv = RTC_AUTO_1_SECOND;
    rtcHandle.Init.OutPut       = RTC_OUTPUTSOURCE_NONE;
    if (HAL_RTC_Init(&rtcHandle) != HAL_OK)
    {
        Error_Handler();
    }
}
```

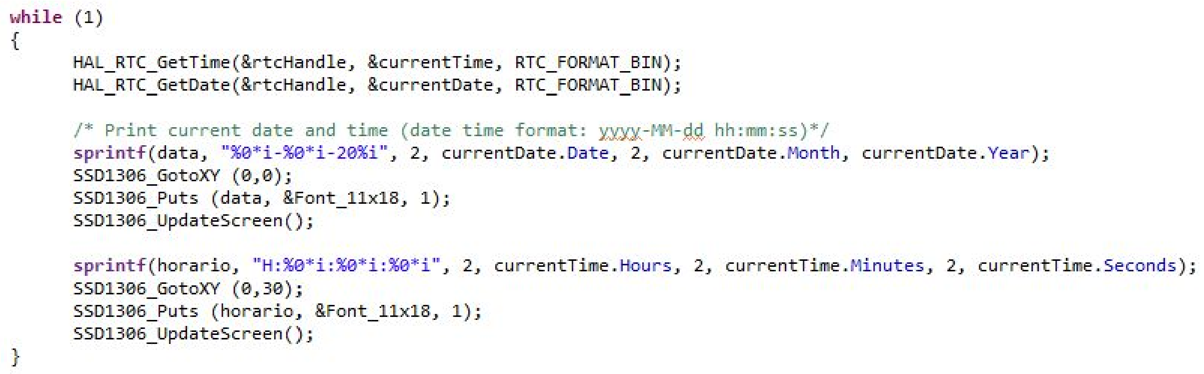


```c
/**
  * @brief  Time and date configuration
  * @param  None
  * @retval None
  */
void RTC_TimeDate_Config(void)
{
    RTC_TimeTypeDef rtcTimeStruct;
    RTC_DateTypeDef rtcDateStruct;

    /*## STEP 1: Configure time ###############################################*/
    /* Set time: 23:59:55 */
    rtcTimeStruct.Hours   = 0x23;
    rtcTimeStruct.Minutes = 0x59;
    rtcTimeStruct.Seconds = 0x48;
    if (HAL_RTC_SetTime(&rtcHandle, &rtcTimeStruct, RTC_FORMAT_BCD) != HAL_OK)
    {
        Error_Handler();
    }

    /*## STEP 2: Configure date ###############################################*/
    /* Set date: 17/02/2020 */
    rtcDateStruct.Year    = 0x20;
    rtcDateStruct.Month   = RTC_MONTH_FEBRUARY;
    rtcDateStruct.Date    = 0x17;
    rtcDateStruct.WeekDay = RTC_WEEKDAY_MONDAY;
    if(HAL_RTC_SetDate(&rtcHandle, &rtcDateStruct, RTC_FORMAT_BCD) != HAL_OK)
    {
        Error_Handler();
    }
}
```

Figure 17. Code of second and third step



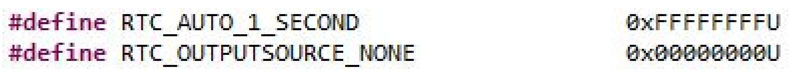
```c
while (1)
{
        HAL_RTC_GetTime(&rtcHandle, &currentTime, RTC_FORMAT_BIN);
        HAL_RTC_GetDate(&rtcHandle, &currentDate, RTC_FORMAT_BIN);

        /* Print current date and time (date time format: yyyy-MM-dd hh:mm:ss)*/
        sprintf(data, "%0*i-%0*i-20%i", 2, currentDate.Date, 2, currentDate.Month, currentDate.Year);
        SSD1306_GotoXY (0,0);
        SSD1306_Puts (data, &Font_11x18, 1);
        SSD1306_UpdateScreen();

        sprintf(horario, "H:%0*i:%0*i:%0*i", 2, currentTime.Hours, 2, currentTime.Minutes, 2, currentTime.Seconds);
        SSD1306_GotoXY (0,30);
        SSD1306_Puts (horario, &Font_11x18, 1);
        SSD1306_UpdateScreen();
}
```

For NUCLEO-F401RE, the code development follows the same steps as the STM32F103C8T6, except in two parts of the code. First, the stm32f4xx_hal_rtc.h does not have the RTC_AUTO_1_SECOND and RTC_OUTPUTSOURCE_NONE defines like stm32f1xx_hal_rtc.h has. So, it is needed to define these two values, be it in the main.c file (like shown in this example) or be in the main.h file. This part of the code is shown in Figure 18.

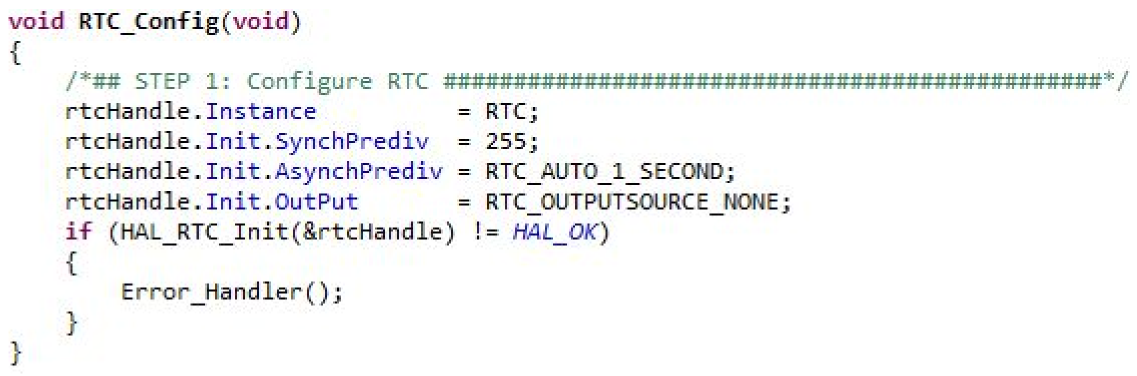
Figure 18. Adding defines on NUCLEO-F401RE main.c



```
#define RTC_AUTO_1_SECOND                   0xFFFFFFFFU
#define RTC_OUTPUTSOURCE_NONE               0x00000000U
```

Second, in the RTC_Config function, it is necessary to set up another parameter not considered in the STM32F103C8T6's implementation. This parameter is the synchronous predivide value or the SynchPrediv field in rtcHandle's struct that should be initialized with the value of 255 as shown on Figure 19.

Figure 19. Setting up synchronous predivide value

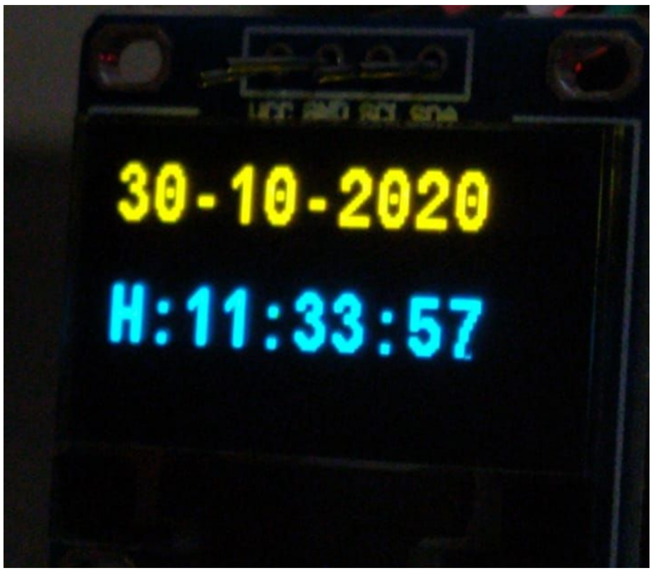


```
void RTC_Config(void)
{
    /*## STEP 1: Configure RTC ###############################################*/
    rtcHandle.Instance          = RTC;
    rtcHandle.Init.SynchPrediv  = 255;
    rtcHandle.Init.AsynchPrediv = RTC_AUTO_1_SECOND;
    rtcHandle.Init.OutPut       = RTC_OUTPUTSOURCE_NONE;
    if (HAL_RTC_Init(&rtcHandle) != HAL_OK)
    {
        Error_Handler();
    }
}
```

7 Results

The result is shown in Figure 20.

Figure 20. Calendar/Time result on OLED 128x64.



8 References

- RM0008

https://www.st.com/resource/en/reference_manual/CD00171190-.pdf

- AN475

https://www.st.com/resource/en/application_note/dm00226326-using-the-hardware-realtime-clock-rtc-and-the-tamper-management-unit-tamp-with-stm32-microcontrollers-stmicroelectronics.pdf

- UM1850 - Description of STM32F1 HAL and Low-layer drivers

https://www.st.com/resource/en/user_manual/dm00154093-description-of-stm32f1-hal-and-lowlayer-drivers-stmicroelectronics.pdf

- AN2821

https://www.st.com/resource/en/application_note/cd00207941-clock-calendar-implementation-on-the-stm32f10xxx-microcontroller-rtc-stmicroelectronics.pdf

- STM32'S INTERNAL RTC

http://embedded-lab.com/blog/stm32s-internal-rtc/