Embedded Systems Design

Semester 2 academic year 2017/18

**Lab C: Using STM32CubeMX for operating RTC and USB comms**

Lab C consists of three assignments

- operating the real-time clock;

- operating a USB CDC device;

- operating a USB MSC host.

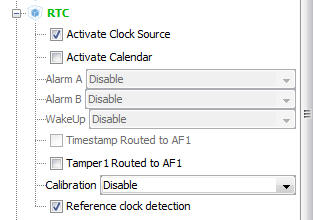
***Assignment 1*** - operating the real-time clock

The real time clock (RTC) peripheral supplies time stamps when the MCU requests these and is capable of waking the CPU up from the deep sleep modes when the right time comes. The RTC needs to be enabled in the STM32CubeMX project, and uses a specific HAL structure to set and read the RTC values.

***1.1. Enabling the RTC in the STM32CubeMX project and checking its clock***

***Make LabC folder and clone your A2 project into C1 in LabC folder first.***

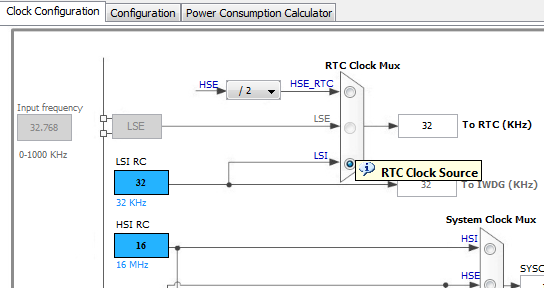
On the ***Pins*** tab, left pane, find the entry for the RTC and enable the peripheral by enabling the ***Reference clock detection*** as shown below:



In principle the RTC can be clocked from either Low Speed External (LSE) or Low Speed Internal (LSI) oscillators; the latter option being the default one when it is enabled. The essential components for the Low Speed External (LSE) oscillator (two capacitors C16, C27 and crystal X3) are not fitted to the STM32F4Discovery board by the manufacturer as shown below:



Therefore one can only use the Low Speed Internal (LSI) oscillator; this default option is enabled by the STM32CubeMX itself which can be checked from the Clock configuration tab as shown below:



The default configuration of the RTC (**Configuration** tab) should work just fine.

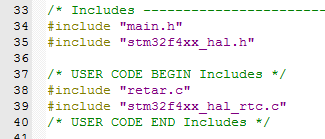
Generate the source code and open it in Keil uVision.

***1.2. Preparing the code to operate the RTC***

Copy the file ***stm32f4xx\_hal\_rtc.c*** from BB into the **C1\Src** folder and remove this file from the MDK ARM project (the leftmost pane, find the file in the folder ; make sure you do not confuse the a.m. file with similarly named ***stm32f4xx\_hal\_rcc.c***). This is required to mitigate a bug present in the stock STM32CubeMX file (sometimes it happens, tinyurl.com/h9rtvr9 for details if you want).

Edit file ***main.c*** as detailed below.

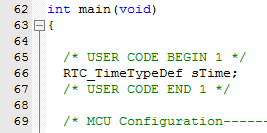
Add the line for this file to the placeholder for the user includes as shown:



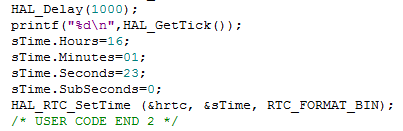
Check that the source code compiles successfully out of the box.

Enable ***printf*** function as it is discussed in the Appendix C of the Reference Manual. You need to do it only once for every project generated by the STM32CubeMX.

Declare the following variable (best place to do it within the user code placeholder straight after the ***main()*** as shown below)



Add the following code to the initialisation section (make sure that you put the code between valid placeholders)



The above code sets the RTC's initial values. Change the **sTime.Hours, sTime.Minutes** and **sTime.Seconds** values shown above to the present hours, minutes and seconds.

Please remove the code for flashing your LEDs form the superloop.

At the end of the superloop you will also need two more lines of code   
- use function **HAL\_RTC\_GetTime** (with the same parameters as shown above for **HAL\_RTC\_SetTime** ) and   
- ***printf*** the obtained values of **sTime.Hours, sTime.Minutes** and **sTime.Seconds**.

After this provide blocking delay determined by the last two digits of your student ID number (bXXABCDE) using the following formula

delay = (D+E)\* 100 ms + 500 ms

For example, the student with the ID number b1234567 will calculate put the delay of (6+7)\*100 ms + 500 ms = 1800 ms and call the following HAL function

**HAL\_Delay(1800);**

**Demonstrate your working code to the lecturer**.

***Assignment 2*** - operating a USB CDC device

Communication Data Class (CDC) devices enable sending data to a USB host that can be used in a consistent manner by any host application which operates serial (COM) ports.

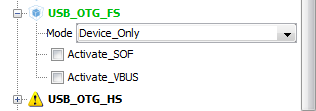
STMicroelectronics, in addition to USB transceivers built into the STM32F4 MCUs, provides a middleware (set of subroutines) that ease up communicating data between the application code and the transceivers.

Both the hardware and middleware needs to be enabled in the STMCubeMX project, and application code has to use the correct HAL calls.

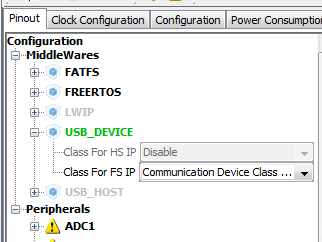
***2.1. Enabling the USB transceiver and middleware in the STMCubeMX***

***Clone C1 into C2. Remove file stm32f4xx\_hal\_rtc.c from the MDK project tree. Enable printf, compile and check that the code operates correctly.***

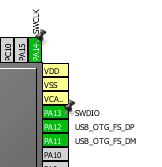
On the Pinout tab, left pane, find the USB\_FS (full speed) peripheral and enable it in the device only mode as it is shown below:



Then scroll up to the very top of the pane and enable the USB\_DEVICE middleware:



Check that the two USB MCU pins became enabled on the pinout diagram:



Switch to the **Clock configuration** tab and enter the HCLK (MHz) of 168 in the relevant box. This is necessary to get the proper 48 MHz clock required for the USB hardware.

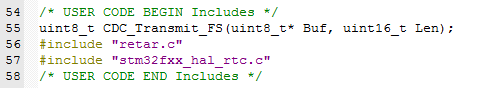
The default configuration of the USB\_DEVICE and USB\_FS (**Configuration** tab) is fine.

Generate code from Cube, open the project and remove ***stm32f4xx\_hal\_rtc.c*** from the **Project** tree (the leftmost pane in the uVision; make sure you do not confuse the a.m. file with similarly named ***stm32f4xx\_hal\_rcc.c***). Please remember that you need to repeat this procedure every time you re-generate the C code from the CubeMX.

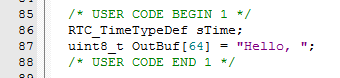
***2.2. Preparing the code to operate the CDC USB device (main.c)***

Enable ***printf*** function as it is discussed in the Appendix C of the Reference Manual. You need to do it once for every project generated by the STM32CubeMX.

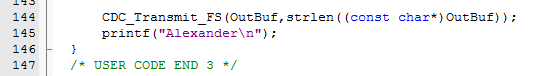
Add the following line to the include section of the code as shown below:



Add declaration of the variable OutBuf between the Cube placeholders shown:



Add the two following lines of code at the end of the superloop as shown below



Please put your own name instead of mine.

Install the virtual COM port driver at your computer (only if you are working from home) as it is described in the Appendix A. (***The driver was installed at the virtual machine already.***)

Connect the micro USB port on the STM32F4Discovery board to a USB port of your computer using a male micro USB to a male A cable (not provided, this is a standard cable that comes with any mobile phone. Some of the micro USB cables are charging cables only; these are not suitable for data communication). ***Make sure that you put the connector in the right way; it is very easy to put it incorrectly then break the on board socket when forcing it in.***

To get data out of the USB virtual COM port, one needs to use a terminal program. Two options are presented in the appendix G of the Reference manual - TeraTerm (available on the host computer only) and CoolTerm (available in the virtual machine). I can assure you that the second option will save you lots of debugging time as you can easily re-scan the USB ports without restarting the CollTerm. In contrast it is necessary to restart TeraTerm every time you make changes to the code.

Run the code; if everything was setup correctly it will now print <YourName> (and timestamps) to uVision and "Hello," to TeraTerm.

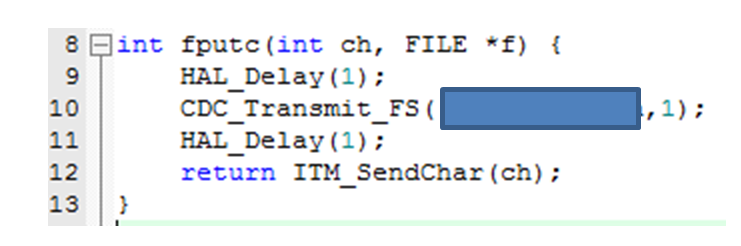
***2.3. Retargeting printf to both ITM and CDC***

We have used the ITM to pass the ***printf*** output to the uVision for debugging and data reporting which helped a lot as you could develop your code from home.

(It was necessary to enable the ITM and set the correct clock frequency for the Cortex core in the Keil MDK project when the source code was generated by the STM32CubeMX at the first time.)

This facility would only work for a PC with installed Keil MDK - not an Android smartphone, for example.

It is possible to direct the ***printf*** output to both ITM and CDC by adding a line of code to the file ***retar.c*** that will direct the received characters not only to the ITM but also to the CDC USB device. It will contain a correct call to function ***CDC\_Transmit\_FS*** in order to output a single byte received by the ***putchar*** subroutine to CDC:



***CDC\_Transmit\_FS*** function accepts two parameters: the first one is the address of the character array to output, the second is the number of characters in the array.   
Please note that the input parameter to ***fputc*** is treated as ***int***; you need to cast data types to avoid compiler error and warning messages when calling the ***CDC\_Transmit\_FS*** function. NOT AN EASY TASK, TRY YOUR BEST.

Please also add lines **HAL\_Delay(1);** before and after the ***CDC\_...*** call as shown above.

After this amendment the code will output "<YourName>!" and timestamps to uVision and "Hello,<YourName>!" and timestamps to TeraTerm.

**Demonstrate your working code to the lecturer**.

***Assignment 3*** - operating a USB MSC host

Mass Storage Class (MSC) embedded hosts enable connecting standard external USB flash or hard drives to embedded systems.

STMicroelectronics, in addition to USB transceivers built into the STM32F4 MCUs, provides a middleware (set of subroutines) that ease up communicating data between the application code and the transceivers.

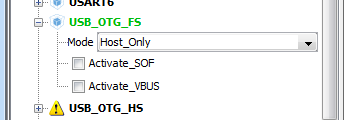
Both the hardware and middleware needs to be enabled in an STM32CubeMX project, and application code has to use the correct HAL calls.

In order to complete this lab you will need to connect a mass storage device to the micro USB female connector on the STM32F4Discovery board. Several options to accomplish this are discussed in Appendix C.

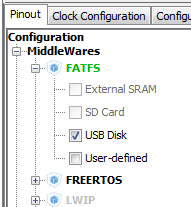
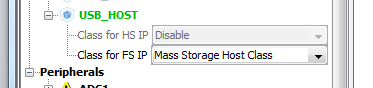
***3.1. Enabling the USB transceiver and middleware in the STMCubeMX***

***Clone C1 into C3***

On the Pinout tab, left pane, find the USB\_FS (full speed) peripheral and enable it in the host only mode as it is shown below:



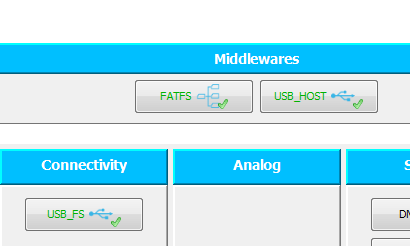
Then scroll up to the very top of the pane and enable the relevant middleware:  
USB\_HOST for Mass Storage Host Class then FATFS for USB disk



Check that the two USB MCU pins became enabled on the pinout diagram as it was shown for the C2.

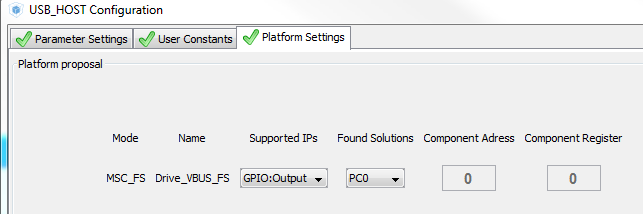
Configure pin PC0 for GPIO output to provide power to the attached USB device via the on board power FET   
(you may want to have a look at the schematic diagram of the board in the board's manual; the switch is labelled U6 on page 39; the part name is STMPS2141STR; it connects the 5V power line from the mini USB connector to the same line of the micro USB connector as memory sticks do not have power supplies).

Proceed to the ***Configuration*** tab



and open configuration for the ***USB\_HOST*** middleware.

In the platform setting tab select the options below for the ***Platform Settings***



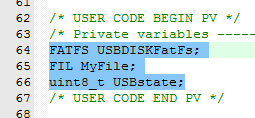
All the other default configuration settings of the USB\_HOST and USB\_FS are fine.

Generate code from Cube, open the project and remove ***stm32f4xx\_hal\_rtc.c*** from the **Project** tree (the leftmost pane in the uVision; make sure you do not confuse the a.m. file with similarly named ***stm32f4xx\_hal\_rcc.c***). Please remember that you need to repeat this procedure every time you re-generate the C code from the CubeMX.

***3.2. Preparing the code to operate the USB host***

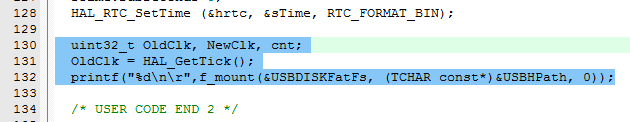
Enable ***printf*** function as it is discussed in the Appendix C of the Reference Manual. You need to do it only once for every project generated by the STM32CubeMX.

Copy and paste from file ***C3\_superloop.c*** highlighted declaration of the following private global variables as shown below

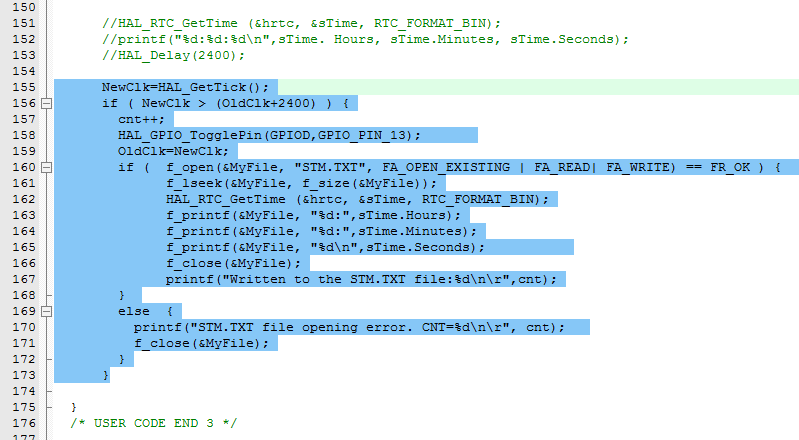


The first variable holds all the data related to a USB drive itself (560 bytes); the second one does the data related to a file (556 bytes). If you need to operate more than one file at a time, you will need to add more FIL variables. The third variable is used for storing then checking the USB connection state where required.

Copy and paste from file ***C3\_superloop.c*** the following highlighted code at the end of the user initialisation code as it is shown below (one needs to mount the USB drive before starting using it):



Comment out the old code for delay and printing the time values then copy and paste from file ***C3\_superloop.c*** the following highlighted user code as shown below



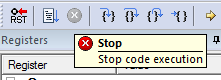
There is a sequencer in the code which attempts to open the file ***STM.TXT*** every 2400 ms. If the existing file was successfully opened for read and write, the counter value ***cnt*** is incremented and printed to the uVision. If there is no file or it will not open (or no valid USB drive attached at all), an error message is printed to uVision.

***NOTE regarding blocking delays (HAL\_Delay) in the superloop: if these are more than roughly 50 ms, the USB host will not work as it needs checking status of the attached device. That is why the sequencer was required in the above code.***

***NOTE regarding f\_printf function: it can only accept one piece of data to print to file and cannot print floating point data. That is why printing time stamp was split into three lines of code.***

If you want to find information regarding ***FATFS*** operation (you can do pretty much everything that you can do at a desktop computer), please consult the online documentation on <http://elm-chan.org/fsw/ff/00index_e.html> .

***NOTE: please remember that it is safe to connect/disconnect/reconnect an MSC device when the MCU is not writing to it only. To make sure this is the case please either press and hold the RESET (black) pushbutton on board or press STOP button shown below in the IDE***



***In order to enable disconnection of the drive, the file is opened before and closed after every write operation. Please use the same arrangement in your coursework but do remember to back up your data as there is still a (little) chance to break the file system when the drive is being disconnected.***

**Demonstrate your working code to the lecturer**.

BLANK PAGE

Coursework:

Variants: selection of the LEDs; clock frequency for the core (table P={2,4,6,8} and Q={3,5,7,9}); temperature sampling time; period and duty cycle for the heartbeat LED.

- ADC to convert VBAT, Vtemp and Vref; calculate and output the temperature after set intervals; keep an LED on whilst these actions are executed;

- provide heartbeat indication - an LED to become on every X seconds for Y milliseconds;

- if the button is pressed turn on an LED and take an additional temperature reading;

- log the measured temperature values to either a PC or flash drive;

- utilise low power mode(s) and estimate the power consumption using the STM32CubeMX ***Power consumption calculator***;

- to achieve the full grade you should develop an On-The-Go USB application that will detect whether it is connected as a device or as a host and log the temperatures accordingly;

- take temperature measurements for at least 12 hours using correct time stamps, plot the recorded data and comment on the observed temperature behaviour.

NOTES to myself:

to do - format SD card and flash drive differently and check whether they are recognised by the library

***extra references on FAT FS***

supports FAT, exFAT (no NTFS)

up to 32 GB only !

no directories on the card !

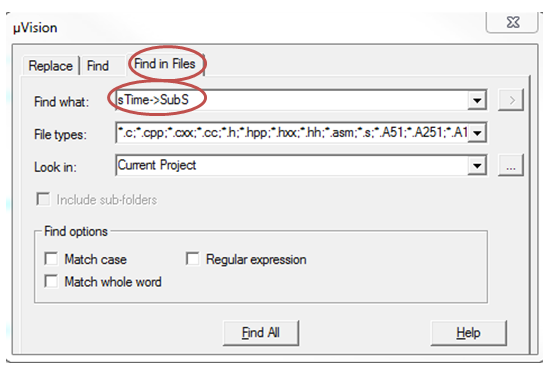
project's website - <http://elm-chan.org/fsw/ff/00index_e.html>

! requires low level drivers that are to be provided (STM does)

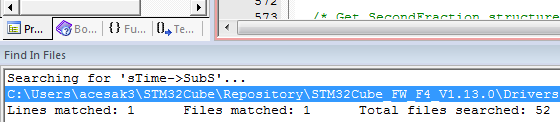
comparison among various file systems <https://www.howtogeek.com/235596/whats-the-difference-between-fat32-exfat-and-ntfs/>

***~~1.3. Fix a bug in the STM32CubeMx generated code (yes, this does happen)~~***

~~Search for the line below (Ctr+F) in the~~ **~~complete project~~** ~~(not in the open file) as shown below~~

~~~~

~~The required line can be accessed by double clicking on the highlighted line shown in the Find in Files window~~

~~~~

~~The following line should be shown then~~

**~~sTime->SubSeconds = (uint32\_t)(hrtc->Instance->SSR);~~**

~~in function~~ **~~HAL\_RTC\_GetTime of the~~** ~~file~~ **~~stm32f4xx\_hal\_rtc.c .~~**

***~~Comment it out, save the file, compile and run the project.~~***

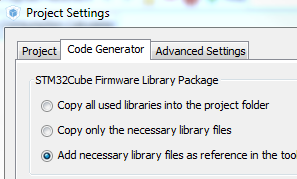
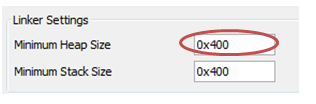
~~(This bug was reported by user dolezal.ivan on page~~ **~~tinyurl.com/h9rtvr9~~** ~~which took me quite a bit of time to find. Unfortunately you will need to comment this line out every time you generate the source code involving the RTC.~~

~~If this line is not commented out, the RTC always returns the set values without ever incrementing them.)~~

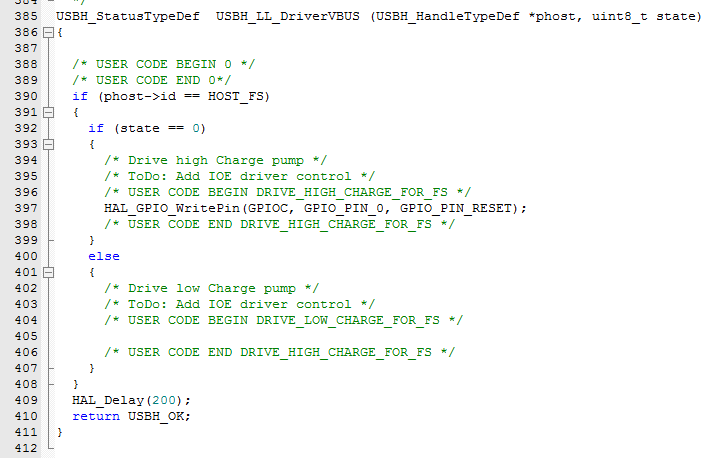
(This bug is now permanently fixed by adding the corrected file ***stm32f4xx\_hal\_rtc.c*** to your project as it is discussed in the second line of the LabCa1 description; please ignore the stricken through text BUT remember to include this file where required and remove the original CubeMX file from the project.)

REMOVED FROM 2.1:

Make sure that you have the following setting in your Cube options BEFORE generating the code (not the default top one) then generate the source code.

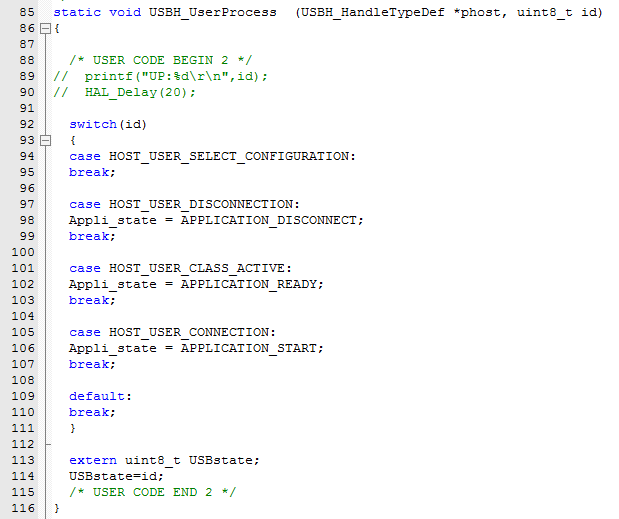
 

if the USB\_HOST platform settings are not configured in the STM32CubeMX one needs to modify ***usb\_conf.c*** as below



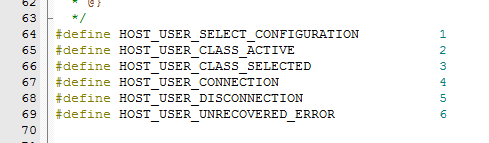
If the code is added to the else part the host stops working though.

***USBH\_UserProcess*** subroutine from ***usb\_host.c*** can analyse the state of the host as it gets the ***id*** parameter

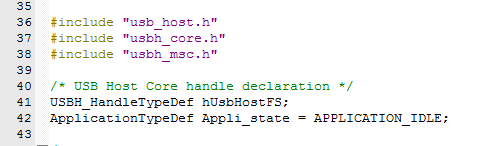


defined in the ***usbh\_core.h***

0 - USB\_IDLE



In turn, the variable ***Appli\_state*** is defined in ***usb\_host.c***



and can take values defined in ***usb\_host.h***

