



Ecole Nationale  
Supérieure  
de l'Electronique  
et de ses Applications LAB\_LM

## HAPPY PLOTTER

V03

LAB (or semi-guided project)  
of microprocessors  
(S7: International group & ENSEA)  
2023-2024  
4 x 4-hour LAB sessions

Before arriving at the first session:  
READ the first 4 paragraphs of this text (5 pages)  
DO the preparations for the first session:  
**INSTALL STM32CubeIDE!**

We will continue the exploration of the STM32 microcontroller you started to study last year. In first year, we focused on Microprocessor; now we are going to focus on Microcontroller.

At the same time as studying microcontrollers, we are going to meet a practical need for many electronics engineers: a signal plotter to obtain a simple oscilloscope!

*This text covers the 4 microprocessor LAB sessions. Each session lasts 4 hours.*

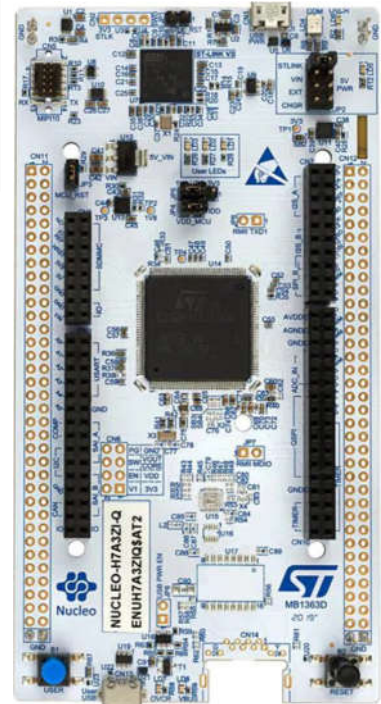
**You are asked to progress your work between each session in the form of a report and a preparation where you will record the writing of the following questions in code, flow chart or pseudocode form. The priority is to prepare the questions preceded by the pictograms:** 🐛 ⚡ 🔧

*By default, a report is requested at the end of each session a few hours or a few days later: it depends on each professor. In your report, number the questions as in this text.*

*You may also be asked to send your project as is at the end of the session. Feel free to ask us to clarify any ambiguities.*

The essential goals of this LAB are:

- ✓ To understand the mechanisms of **exceptions/interrupts** through their implementation.
- ✓ To discover and use some **peripherals of a microcontroller**: Timers, USART, EXTI.



### INSTRUCTIONS FOR USE of this document.



First you have a description of the work to be done with cross-references to the various Developer's 📖 guides for "technical details". You should read the document up to the requested work marked with a different font:


#### or **Required Work -4: Stop reading and act**

The work required of you is then briefly summarized and refers to the previous descriptions. Do not start a job until you see the text that explicitly asks you to do so; you risk starting a useless job. Do not forget to keep a record of each piece of work done (screen capture, photos), so that you can comment on it and include it in your final report. Do not write your final report during the practical sessions (unless the teacher explicitly asks you to do so) as you may not be able to move fast enough, but update your report between sessions. Note that your report should also include analyses, i.e. links or explanations of what you see with the course. At the end of each session, you will be asked to send your code, in particular the file(s) that have been modified. Continue reading.

#### **Required work -3: Stop reading, start thinking and writing.**

When the icon representing a hand appears, you are invited to answer the questions.

The paragraphs noted  correspond to preparations to be made before coming to the LAB session. Then  paragraphs are not to be done during the sessions.

**IMPORTANT**: respect the names of functions and files, even if they may seem strange the first time. Case (upper or lower case) is also important when defining symbols. Respecting names will have several advantages. It will make it easier for the professor to find out where you are at and therefore to be able to dialogue with you more easily about the stages of your project. It will also allow you to insert files into your work. The naming "method" is relatively easy to understand: a "function1" function in the "MyFile.c" file is named "MYFILE\_Function1".

# 1 Goals and means

## 1.1 The mission

We want to view analog (and digital) signals thanks to a software running on your Laptop: "Arduino Serial Plotter". Your Nucleo Board will be helpful to do it by acquiring a signal, and sending the data to your laptop.

Your mission will consist in programming your Nucleo Board to acquire data and send them to your laptop to obtain the best "Serial Plotter": so you will be able to use your board as a very basic oscilloscope to view signals on its pins.

## 1.2 The Hardware

Your Nucleo Board (STM32H7A3). This board has been given to all the students last year (in first year). If you are a new student this year, ask M. Laurent MONCHAL by email or at the end of a course session: you can get or borrow (it will depend on the stock) one for the LABs period.

A micro-USB cable (to be plugged on USB PWR).

## 1.3 The software

### **STM32CubeIDE**

This is the software, to be installed on your laptop (Windows, Linux or Mac), recommended by ST to program the Nucleo board in C. You already used it last year.

### **Terminal**

This is a software to watch the data sent by your Nucleo Board. It will be also used to send data to your board. I recommend "Tera Term" on Windows. It is also possible to use the Console in STM32CubeIDE (but it is not as user friendly interface to do it).

### **Plotter**

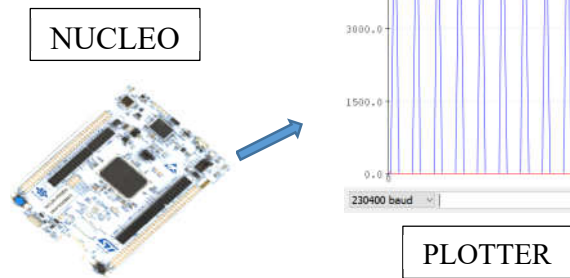
This is a software, first dedicated for Arduino, but it is possible to display data sent by your board in serial communication. The Arduino plotter along with a Terminal is available in Arduino IDE. It is possible to use at the same time, the plotter to view signal and send characters to your board (but not with Tera Term opened). If you can't install Arduino IDE, you can use the version in the labs and communicate with the Nucleo using TeraTerm or the console in STM32CubeIDE.

## 1.4 The documentation

Guides have been elaborated and gathered on the Moodle page: <https://moodle.ensea.fr/course/view.php?id=652> . You should find there all the pieces of information for your work.

## 2 The main steps of your work

Our final goal is to display signals from your NUCLEO board on a PLOTTER (the one in Arduino IDE is great) running on your laptop. You will follow these steps during the 4 lab sessions:



- Creation of the project
- Sending messages with the USART
- Study of the boot and Clock configuration.
- Timers: analysis and choice
- Green LED blinking with TIMA at 1Hz (Still Alive LED: SALED).
- TIMB to count the  $\mu s$  (test with serial message in LED blinking) (absolute Time)
- TIMC to send samples to the plotter, one by one
- Blue Button to trigger an acquisition and display
- Analysis of time to send data (limits of the performances)
- Improvement by speeding up the serial communication to 230 400 bauds
- Improvement by implementing a new strategy: sending samples in batch.
- Yellow LED: active on certain activities of your code (Busy LED: BLED)
- Configuration and use of the ADC (test with GND and VCC)
- Generation of a test signal thanks to PWM
- First the red LED is used to test PWM code
- This signal PWM is set by sending commands: USART interpreter
- Generation of a PWM signal to test your plotter
- Use of DMA to improve the use of the CPU...

## 3 Home works to do before coming at the first lab session



### **Requested preparation -2: Software installation**

If this is not already the case, install the STM32CubeIDE software on your laptop: it is explained in <https://moodle.ensea.fr/mod/resource/view.php?id=35626>.

You will have to use a Terminal to communicate with your board. The best Solution (for Windows) is "Tera Term". Install it or find another solution (Putty for example).

You need the plotter in Arduino IDE, so install it (available for Windows, Linux and Mac).

Your kit is necessary for testing. It is referenced "NUCLEO-H7A3" (distributed last year). It is described in the "NUCLEO 64 user manual" in Nucleo Board section: <https://moodle.ensea.fr/mod/resource/view.php?id=48749>. The schematic of the Nucleo Board is available here: <https://moodle.ensea.fr/mod/resource/view.php?id=49702>.

The microcontroller is described in the Reference Manual: <https://moodle.ensea.fr/mod/resource/view.php?id=46183>.

VCP is for Virtual COM Port. One USART is selected in your project to communicate on this link: this the serial link you are going to use in the "SERIAL.c" functions.



### **Required preparation -1: Knowledge of the hardware**

Open the “user manual” and “schematic” of your board and the Reference Manual.

How many pins are on the STM32 microcontroller of your Nucleo board (here the question is about the microcontroller itself, not the Nucleo) (count all the pins on the  $\mu$ C)?

Where are located the ST morpho connectors on your board?

Why don't we use ST Morpho?

Where are located the “Zio” connectors on your board?

How many pins are available on the Zio connectors?

On which USART is connected the VCP signal?



### **Requested preparation 0: Creation of the project**

Please define your project in the correct location so that it can be easily found in subsequent sessions. To do so, create a directory dedicated to your project (and only for it) in your personal workspace on your account. Preferably give it a simple name without spaces, accents,  $\mu$  or other special symbols.

Details on how to create your project are provided in the developer's guide for STM32Cube IDE. One detail must be strictly followed: at the beginning, select “board”.

For this project, select “Default Initialization” then you just have to unselect USB\_OTG\_FS.

Next you will add your own files but not yet. The easiest way is to create the C files in “Core/Src” and the H files in “Core/Inc”. You will find in the guide how to make these additions or file creations in other directories (if you wish).

The first goal is to compile your project without errors.

The test consists in generating the code, compiling it, entering debug mode and just execute one step (maybe you could have to add a breakpoint). So you should arrive for the first lab session with a project you can work with. Maybe, there will be trouble shootings with the installation, we'll try to fix them first.

This work will be evaluated at the beginning of the first session.

## **4 Virtual Com Port and Clocks**

### **4.1 Communication between your laptop and your Nucleo**

Your first goal is to send a first simple message with the UART communication line to your laptop.

The message is sent from your Nucleo board using an USART module to your laptop. This module, correctly configured, will send characters.

On your laptop, you need a “terminal” correctly configured to receive Serial Com characters. Install Tera-Term or use the terminal in STM32CubeIDE.



### **WORK 1: Sending a first message to your laptop**

Read the code for initializations in “main” function (in main.c). Which USART is initialized? Give its number n. Write it down.

Check the code in MX\_USARTn\_UART\_Init: What is the baud rate? How many data bits are sent at a time by the serial link? How many stop bits? Is there a parity bit? You must know all these features because there will be asked in console or terminal to communicate in the right way with your board.

Use the “ioc” view (CubeMX) to check again the USART.

You can first add your code in main function of “main.c”. Next time, you will add code in handlers of “stm32xxx\_it.c”.

Add a private variable: `char UART_buf[100];`

Add the includes you need:

```
/* USER CODE BEGIN Includes */
#include <string.h>
```

```
#include <stdio.h>
Add the code to send a message with "HAL_UART_Transmit":
/* USER CODE BEGIN WHILE */
sprintf(UART_buf, "LAB:STM32H7A3>>");
HAL_UART_Transmit(&huart3, (unsigned char *)UART_buf, strlen(UART_buf),
1);
```

You must have in mind that adding code in your STM32CubeIDE must be done in given areas (between "USER\_CODE\_BEGIN" and "USER\_CODE\_END"). Outside these areas, the code is erased as you generate again code. Is it the case for the code you have inserted? Yes! So let's generate the code.

Compile the code and enter the debug mode and run the code.

You have many options for the Terminal to be used. It may be "Tera Term" (my favourite), the terminal under debug session of STM32CubeIDE or any other Terminal with Serial Com that works. Whatever the terminal you use, configure the Serial Communication with: 115200 bauds, 8 bits, no parity, 1 stop bit, no hardware control (same configuration as MX\_USARTn\_UART\_Init).

TEST: check that you can display a simple message and display it.

We want to send messages to our laptop in order to display in a terminal the values (so they must be in ASCII code) and to plot the values using "Arduino Plotter" in the Arduino IDE (to be installed) (it is in ASCII mode too!). That's why, in the next work, your goal is to send more complex messages with numbers (integers or floating-point values) by using SERIAL functions in "serial.c" (SERIAL\_SendCharBuf, SERIAL\_SendInt, SERIAL\_SendFloat).

The functions are:

- "SERIAL\_Init(UART\_HandleTypeDef \*phuartInit)" is to be used once to send the handler corresponding to the UARTn to be used.
- "SERIAL\_SendCharBuf(char \*buf)" to send the content of an ASCII buffer ending with a null character.
- "SERIAL\_SendTAB (void)" to send the ASCII code of tabulation (9).
- "SERIAL\_SendNL (void)" to send the ASCII codes of CR (13) and LF (10) as a new line.
- "SERIAL\_SendInt(int n)" to display the value "n" of an integer passed as an argument.
- "SERIAL\_SendFloat(float v)" to display the value "v" of a floating value passed as an argument. You have to check "Use float with printf from newlib-nano" in "C/C++ Build", "Settings", "MCU Settings".
- "SERIAL\_SendToPlot(int \* dataA, int \*dataB, int nb)" to send nb integers in data and dataB arrays.



## **WORK 2: Sending more complex messages to your laptop**

Add the serial.c and serial.h files to your project (the files are available on Moodle).

You have 2 options to add this file.

The easiest way consists in adding the source file (\*.c) in "Core/Src" and the header file (\*.h) in "Core/Inc" (Drag and drop or copy). It is easy because you can compile this code without modifying parameters. The main drawback comes from the fact that you must not update the library used for the project (because serial.c and serial.h files will be erased). For the lab session, I think that it is the quickest way but there is a main drawback to this method: don't update the libraries for H7xx before the end of the last session.

The second way consists in adding the source file (\*.c) in a new source folder and the header file (\*.h) in an include folder. You have to create these folders at the same level as "Core/Src" and "Core/Inc". It could be: "Plotter/Src" and "Plotter/Inc". The main drawback comes from the fact that it does not compile at first because a C file can be compiled if you give its exact location to STM32Cube IDE and an header



file can be included if you include its path in the project. That's why, you have to add "Plotter/Src" as a source location and "Plotter/Inc" as an include path (it is explained in the documentation). It is explained in "Developer's guide for STM32CubeIDE" with more details. If you don't find this information, ask the teacher during the lab session.


The code to send messages is in serial.c. First and just one time, use SERIAL\_Init and then other SERIAL functions to test messages with numbers. Your first test TESTW02, will consist in displaying messages with integers and floats (declared as variables in the code).

#### 4.2 A first vector

This third work is about analysis of your project and the way it is implemented in the memory.



#### **WORK 3: Finding the very beginning of your code (spoil: it's not "main")**

Now that your project is compiled and runs perfectly, you can enter Debug mode . To do so, have a look at the guide. There are also videos about it (in Moodle to download in Ubcast).

The value of SCB\_VTOR contains the address of the beginning of the table of vectors. You can access this value in "debug" mode in SFRs window ("Window"->"Show View"->"SFRs" to make it appear). VTOR register is "hidden" amongst STM32 registers in Control\_VTOR. Give the value of SCB\_VTOR (in Control\_VTOR).

According to the ARM documentation, the address (32 bits) of the first instruction (executed after the Reset signal) is located at SCB\_VTOR+4. To open a memory window: select "Window"->"Show View"->"Memory". The memory window is not opened in the right mode! You have to modify the rendering. To observe the memory select "Hex Integer" rendering.

Give the value of the address of the first instruction by erasing the LSB of the value at SCB\_VTOR+4.

Open Disassembly view (an address in hexadecimal format starts with "0x"). To view the code in disassembly mode: "Window"->"Show View"->"Disassembly". Remember the spoil: the first instruction executed is not marked by the label "main". The label for the Reset code is "Reset\_Handler" and it is in "startup\_stm32....s". To confirm your answer, give the screenshots and the comments to demonstrate that it is consistent. Use screenshots to show it.

#### 4.3 Clock control

We are going to analyze the frequencies of the different clocks:

- On the one hand, the frequencies present at the start, therefore in the absence of initializations (so at Reset\_Handler or at least before SystemClock\_Config).
- Frequencies after initializations (SystemClock\_Config).

We won't study the details of the initialization: just 2 fields in 2 registers. A field is a sequence of 1 to 16 bits in a 32-bit register that defines an initialization.



#### **Required preparation 4: RCC registers for clock control**

Read the Reference manual about the SW field of RCC\_CFGR and the CDCPRE field of RCC\_CDCFG1. What do they define?



#### **Required work 5: clock after Reset**

Use SFRs window interface to observe and read the contents of RCC\_CFGR and RCC\_CDCFG1 **before executing any code**. Deduce the CPU clock frequency thus defined before any initialization.



### Required work 6: Clocks after SystemClock Config()

ReUse SFRs window interface to observe and read the contents of the 2 previous fields in RCC registers (RCC\_CFGR and RCC\_CDCRGR1): what is the clock that has been chosen as SYS CLOCK?

Have a look at the “clock configuration” tab in the “ioc” view. Then give the different frequencies thus defined after Initializations for AHBx and APBx clocks. What is the main advantage of such a choice?

Don't try to find the complete configuration of the bits: it's too much complex.

#### 4.4 A review of this first part

You didn't write a lot of code but there are full of notions to explain.

Now, the Reset vector is no secret for you and you can explain how the microprocessor starts.

You scanned 2 fields from 2 registers of the RCC module to find a link with the various frequencies (after Reset and then after an initialization). It's a way to make a link between 0s and 1s and initialization of the clocks.

Then you studied a device (USARTn) and used it.

Last year you used a method very close to the **hardware**: it consisted in writing in memory using pointers that you defined yourself. It is important to understand the general basic principle for initializing a device: it is done by writing data to a location in the memory area (assigned to the device in the Peripheral memory). It is an elementary and general method called “Bare metal Programming”.

That is why we use ioc view to generate the code to activate and configure modules with HAL functions. At the end, the HAL functions write in the peripheral memory.

Always keep that in mind! Let's continue then.

### Assignment 7: Make your own picture sheet of this assessment.

Provide a one-page summary of what you feel as essential to the principles that have been seen in this first part. Relate this to the basic principles of a microcontroller. Try to mention the various levels for the code. Have some words about the different memories in the microcontroller: RAM, ROM, PERIPHERAL.



## 5 Green LED blinking with Timer Interrupt

### 5.1 Principles of a Timer

Today, we are going to start by making an LED blink. It will help you next, to write another code in the same way to send 100 digital samples of a sine signal to a plotter on your laptop as you push the blue button.



#### **Required preparation 8: Timers**

To achieve this goal, we need to use Timers from your Nucleo Board. That's why you have to find the Timers available on your Nucleo. Have a look at the datasheet: Give the list of all the Timers on your microcontroller and for each Timer give the frequency of the clock, the maximum values for ARR and PSC.

Remember how a TIMER works. It is explained in the course.

What are the Timers that can count using 32 bits?

What are the values for ARR and PSC to obtain an End of Count 2 times per second with 96MHz (or the frequency of the Timers if they have been modified) as an input? Give the way to obtain it.

Which bit, in which register of the RCC module enables the Timer peripherals?

Which registers and at which addresses are located the registers that allow us to configure the Timers? Give at least 5 of them.



#### **Required preparation 9: Timers, LEDs and PWM**

We will use certain Timers to control LEDs: the green and/or the red one.

What is the pin for the green LED (use Nucleo schematic or ioc view)?

What is the pin for the red LED?

Each Timer owns (up to) 6 channels to generate a PWM signal. For example, for Timer 2 there are 4 channels: TIM2\_CH1, TIM2\_CH2, TIM2\_CH3 and TIM2\_CH4. These channels can be connected to certain pins. For example, for TIM2\_CH4, it can be connected to PA3 and PB11. Check that TIM2\_CH4 is connected to PA3 and PB11, by searching TIM2\_CH4 in the datasheet. Give the screenshots of your search.

In the other way, if you search for PA3 you should find that it is connected to TIM2\_CH4 (and other Timers).

Give the Timers that can be used to generate a PWM signal for the Green LED.

Give the Timers that can be used to generate a PWM signal for the Red LED.



#### **Required preparation 10: Timers and Handler code**

Most of the Timers can generate an interrupt signal as an end of count (EOC) occurs (when CNT is equal to ARR).

It is possible to trigger a code from this event. We will see how.

The event for the end of count of a timer is a "global interrupt" event.

In the Reference Manual, in the NVIC chapter, a table gives the vectors that are available. You have to open this Manual and search in the NVIC chapter for "global interrupt". At each time, it is associated with a Timer, write down the "acronym": it gives the name of the interrupt handler. In STM32CubeIDE the name of the Handler code is the acronym followed by "\_IRQHandler". For example, for the TIM2 acronym the name of the Handler code is TIM2\_IRQHandler. Sometimes it is less obvious because the code is shared (but you can use it). For example: TIM8\_BRK\_TIM12, so TIM8\_BRK\_TIM12\_IRQHandler for TIM12 (if you don't use TIM8 as an interrupt signal, you can use TIM12).

Give the names of the Timers, alone, for their "interrupt" Handler code. I advise you to use these Timers first.

Give the names of the Timers, sharing their “interrupt” Handler code. They can be used if necessary.

### **Work required 11: check available Timers with ioc view**

In the datasheet of your microcontroller (H7A3Zi) the architecture is given with all the Timers you can use. To ease the configuration of all these Timers, with STM32CubeIDE, you have an “\*.ioc” file in your project (use “Project Explorer” window on the left). Open “\*.ioc” file in STM32CubeIDE by double clicking on it. It may take time to open but it is worth waiting for it.

Click the “Pinout & Configuration” tab and select “Timers”. You can select the timers one after the other in order to check their configuration. Compare the ioc view for the Timers with the information you get from the datasheet.

## 5.2 Control of the SALED

Here we want to be sure that your code is still running (or not) by having a brief glance at the green LED. We call it SALED for “Still Alive LED”. Your goal is now to make this LED blink at 1 Hz using Interrupt Handler triggered by a Timer each 0,5 s (ON/OFF so  $0,5 \times 2 = 1\text{s}$ ).

### **Work required 12: Green LED control**

The green LED is mentioned as LD1 on the STM32H7A3-NUCLEO board and in STM32CubeIDE software. In the default configuration of your project, it should be already configured (the work is done!). Check it in the “\*.ioc” view in STM32CubeIDE (select “Pinout and Configuration”-“Categories”- “System Core”- “GPIO”). On which pin is connected this LED and how is it configured (mode, pull-up...) (you can switch the column of the array)?

Generate the code, compile it and launch it in Debug mode.

The configuration code for GPIO (so for the LED) is in function MX\_GPIO\_Init in “main.c”. The configuration is done by writing in Peripheral memory with HAL\_GPIO\_Init function. Show the registers that are modified when configuring the Green LED with HAL\_GPIO\_Init. Use breakpoint and SFR view before and after modification of the memory and so the registers. Give the name of the registers and their addresses modified by HAL\_GPIO\_Init for the Green LED (and only for it).

In the SFR view, show that you can toggle the bit by controlling the green LED in ODR register and thus switching on and off the green LED. In the next work it will be possible to toggle the LED by software, using HAL\_GPIO\_TogglePin function.

You are going to make the LED blink by triggering (not calling!) a handler code each 0,5s to toggle the LED. The main steps are:

1. Configuring a Timer
2. Configuring the NVIC to enable the interrupt from the Timer
3. Writing the handler code to toggle the LED
4. Testing the code

## 5.3 Using a Timer (TIMA) for the Green LED

We are continuing our work by setting an interrupt on a periodic counter (we call it TIMA but it can be TIM1 to TIM17 depending on your choice) which will be useful to make the green LED blink. We will periodically count down (or count up) a counter and at the end of each count we will generate an event to trigger an interrupt handler. The interrupt handler code will toggle the LED at each time.

### **Required work 13: configuration of TIMERA**

Use CubeMX view (the ioc file) to activate TIMERA. You must select "Internal Clock" as "Clock Source" in CubeMX view (double click on \*.ioc file to obtain it).

Enter the values of PSC (Prescaler) and ARR (Counter Period) in the CubeMX interface to obtain an "end of Count" event two times a second.

Save and generate the code. This step may take time because software package may be downloaded and then installed.

Compile it and execute it in debug mode to show that the configuration is the one you want for the values of PSC and ARR.

Give the code in "main.c" that configures TIMERA. Which call (HAL function) configures TIMERA?

Does the counter count? In other words: does CNT register evolve? What did you expect? (as Penelope Cruz asked). How to make it evolve? Or which bit in CR1 register enables the counting?

Your Timer is configured but not yet activated. You need:

1. To launch your Timer in your code with `HAL_TIM_Base_Start_IT`
2. To add your code in the corresponding handler to toggle the LED
3. To test the LED blinking at the right speed

### **Required work 14: LED blinking with TIMA interrupt**

Add the code to launch the Timer.

Use again CubeMX for initializations. In "NVIC Settings" of TIMA enable "TIMA global interrupt". For more details have a look at the guide for STM32CubeIDE. Thank to these choices, a `TIMA_IRQHandler` handler function is then generated in "stm32xxx\_it.c". This is where you are going to add your own code.

Change the preemption priority to a value other than 0.

Which registers (and which bits) in the NVIC indicate that the previous initialization is correct? What exactly are the NVIC registers affected by Timer A.

Which bit of which register allows us to say that Timer A interrupts are accepted by NVIC? Take a screenshot.

Which bits and in which register gives the priority level of Timer A? Take a screenshot.

However, if you start the program, the interrupt processing will not be started because the TIMER lacks the code to start the TIMER, which sends interrupt requests to the NVIC. The TIMER A is launched after the initializations using the following code:

```
HAL_TIM_Base_Start_IT(&htimA);
```

This code has to be called once: it starts the counting of TimerA as well as the generation of the interrupt requests associated to the end of counting of each cycle (EOC= End Of Count). So don't forget to call it somewhere in your initializations: without this call the Timer counter remains (desperately) disabled and your LED won't blink even if everything else in the code is correct. To use this code, it is necessary that `htimA` is set as a global variable. This must be the case in only one of your project files. If you want to use `htimA` in another file, you must take the precaution of declaring the `htimA` data structure on the timer2 as an external global variable beforehand:

```
extern TIM_HandleTypeDef htimA;
```

Compile the set so that you can observe the blinking of the green LED. Check the consistency between the blinking rhythm and the values of ARR, PSC and the clock sent to the Timer. Show that it is cohesive.

Show the result to the professor.

## 6 Plotting signals

### 6.1 Generation of sine and cosine signals: samples one after the other (1kHz sample rate)

As we want to send samples of an analog signal to the plotter, we need to know precisely the time elapsed on your board. That's why you are going to choose a Timer (TIMB) to count  $\mu s$  in the register TIMB\_CNT. Then we need to check precisely this time counting, that is why you will use a special "tool" inside the debug to check the time: ITM\_Port32.



#### Required preparation 15: Very basic DSP

Our goal is to know what we must expect as a display.

The signal to be generated is in this form:  $A \cdot \sin(2 \cdot \pi \cdot F \cdot t)$ . A sample of this signal is computed and sent to be plotted at a sampling rate of 1kHz. So it means that, each 1 ms (the sampling period), a sample is computed and displayed. That's the goal but maybe this goal won't be reached (for various reasons) that's why we want to know what to expect and to be able to say OK or not OK for our implementation.

In our test we will send 50 samples at a time (because only 50 samples can be displayed by the Plotter in the last version).

If F is 20Hz, how many periods of the signal must we see with 50 samples?

If F is 50Hz, how many periods of the signal must we see with 50 samples?



#### Required work 16: Timer with $\mu s$ accuracy

Our goal is to generate a time reference for t with  $\mu s$  accuracy to generate:  $A \cdot \sin(2 \cdot \pi \cdot F \cdot t)$ .

Choose a Timer to count the  $\mu s$  in its CNT register (tip: choose it to count as many  $\mu s$  as possible) (so a 32-bit Timer). We call this Timer TIMB.

Configure this Timer in ioc view.

In your code launch TIMB with HAL\_TIM\_Base\_Start.

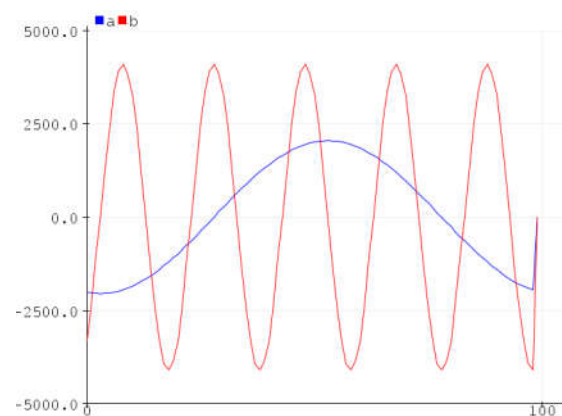
Add in your code a Time stamp (ITM\_Port32 described in "Developer's guide") and use SERIAL\_SendInt to display TIMB\_CNT (TIMx->CNT in C, with x a value from 1 to 16) in TIMA handler code.

Show the consistency of your results. Now TIMB\_CNT should contain the number of  $\mu s$  elapsed since you launched TIMB. This will be our true time reference. Next, we will use it to generate a sine or cosine signal.

```
a:1218,b:-4090
a:1311,b:-3868
a:1399,b:-3328
a:1496,b:-2357
a:1586,b:-1122
a:1658,b:40
a:1724,b:1200
a:1804,b:2578
a:1856,b:3374
a:1903,b:3894
a:1949,b:4094
a:1986,b:3837
a:2011,b:3274
a:2030,b:2442
```

In the next step we are going to send samples to the Arduino Plotter in order to obtain the drawing of sine and/or cosine signals.

Such a drawing (on the right) is generated by sending values as the one (on the left) using the serial link.



The plotter can be found in "Arduino IDE" – "Outils" – "traceur série".

On this graph the x-axis is from 0 to 100 because there are 100 samples but in the last version there are only 50 samples that can be displayed.

We advise you to distinguish 2 states:

- **SAMPLING:** the samples are computed at each time (and sent). It is done 50 times.
- **IDLE:** no samples are computed nor sent.

### **Required work 17: sending samples one after the other (1kHz sampling)**

Choose a new Timer to trigger the sending of a sample for each signal each ms. Our aim is to generate one 20Hz sine (amplitude 1000) and one 50 Hz cosine (amplitude 2000). We call this Timer TIMC. You must use the same strategy as the one for making the LED blink, but instead of toggling the LED in the interrupt handler, you “send” 2 samples:  $a = (A \cdot \sin(2 \cdot \pi \cdot Fa \cdot t))$  and  $b = B \cdot \cos(2 \cdot \pi \cdot Fb \cdot t)$  with the serial functions. The time “t” is given by your reference TIMB. Give 2 different amplitudes to the signals, so you can differentiate them more easily.

The samples must be computed first. You must include math.h to use cos and sin.

Then the samples must be sent with the SERIAL functions. For example, if you want to send the 2 samples a and b with value 1218 for a and -4090 for b, the following string must be sent: “a : 1218 , b : -4090”. Do not forget that you can send variables thanks to the SERIAL functions.

In a first test, trigger the interrupt each 1 ms. Test with “ITM\_Port32” that it is well triggered each ms (you can export the data). In your interrupt handler, use a global variable to count the samples and stop after 50 samples (in order to not saturate the plotter and be able to easily analyze what you see).

To use the Arduino Serial Plotter, you must select a board though our Nucleo board is not an Arduino one (it works with the first one).

How many periods do you expect for your sine (20Hz) and cosine (50 Hz) signals with 50 samples?

Give the screenshots of your results and analyze them. Use ITM\_Port32 markers to answer the following questions.

How long does it take between 2 calls of TIMC interrupt when sending data?

How long does it take between 2 calls of TIMC interrupt when no data is sent?

## 6.2 A Blue button for a functional improvement

We’d like to use the blue button to send 50 more samples without relaunching the code. This improvement doesn’t speed up the process.

### **Required work 18: re-triggering a sending when pushing blue button**

The goal: when debugging, testing or using your kit, it is useful to launch a new plotting of a sine signal as you push on the blue button.

We invite you to add the code to obtain this new behavior. Use EXTI peripheral to trigger EXTI handler. Write in the EXTI handler a code to enable plotting again: it can be as easy as to reset the global value counting the sent samples.

Test your code: a new plot with 50 more samples must appear, each time you push the blue button.

Tip 1: select “External Interrupt Mode...” in GPIO configuration.

Tip 2: configure EXTI Line in NVIC.

Tip 3: insert your code in EXTIxxxxx\_IRQHandler.

Tip 4: more details in your favorite guide.

## 6.3 Improvement of the sample rate

We’d like to send as much samples as possible. As a first improvement, we aim to sample data at 10 kHz (10 times better than the previous version).

A limit comes from the fact that the data are sent at a default rate of the serial link: here 115 200 bauds (bits per second). So this first improvement consists in speeding up the serial communication to 230 400 bauds.

**Required work 19: Improvement ver1 with faster serial communication**

Improve your code by speeding up the serial communication to 230 400 bauds (modify the value in the ioc view, generate and compile).

Show the improvement with "ITM\_Port32" time stamps.

The trouble comes from the fact that for some board, 230 400 may be the highest speed that could be used and the speed must be compliant with the plotter.

Optional: what is the highest speed you can apply to your board as a baud rate?

**Required preparation 20: Very basic DSP at 10kHz sampling**

Our goal is to know what we must expect as a display.

The signal to be generated is in this form:  $A \cdot \sin(2 \cdot \pi \cdot F \cdot t)$ . The sampling rate is 10kHz.

In our test we send 50 samples at a time.

If F is 200Hz, how many periods of the signal should we see with 50 samples?

If F is 500Hz, how many periods of the signal should we see with 50 samples?

**Required work 21: plotting sine signals (10kHz sampling)**

Modify TIMC in order to obtain a 10kHz sampling.

Our aim is now to generate one 200Hz sine and one 500 Hz cosine.

Give the screenshots of your results and analyze them. Use "ITM\_Port32" to time stamp and analyze. Does it work as expected? What is the time to send the signal and so what is the real sampling frequency? Compare with the one expected. Explain the signal that is actually plotted.

What are the limits to get a better sampling time?

**6.4 A new and better strategy to send more samples**

The "bottleneck" of our previous strategy comes from the fact that it takes time to compute (generate) the samples (but we have to do it at each interrupt) and it takes more time to send the samples (this action can be delayed).

So there are two phases during the « sampling time »: first the samples are generated (later there will also be acquired) and then the samples are sent with the serial link. It is compulsory to generate (or acquire next) the data at each sampling time but there is no need to send the data at once. So we propose to save the generated (or acquired) data in an array (it does not take time) and to send then as it is over (we'll have plenty of time). So another way to improve sampling time is to generate and save the samples as fast as possible (at the rythm of the sampling time) and then to send the send the samples in a second phase.

In this strategy TIMC Handler can do one of the 3 main actions:

1. "Do nothing" because there is no need to generate or send samples.
2. "Generate" samples (or "acquire" data in the complete version) and save them in an array
3. "Send the data" as the array is full.

Remark: at each triggering you can generate or acquire one sample in each array but sending the data can be done in one time (send all the data in the arrays).

We advise you to distinguish 3 states:

- SAMPLING: the samples are computed at each time (and sent). It is done 50 times.
- SENDING: the samples are sent.
- IDLE: no samples are computed nor sent.

We advise you to distinguish 2 strategies:



- ONEBYONE: the samples are computed at each time (and sent). It is done 50 times.
- BATCH: the samples are computed at each time (but not sent). They are sent after the computation of the 50 samples.

**Required work 22: Improvement ver2 with better implementation strategy**

Modify your code in order to implement this 3-state version (save your previous code before).

**Required work 23: Limits of ver2**

If you want to send more than 50 samples, it is possible to do it by using a larger array. What is the limit?

What is the best sampling time you could implement now? Justify what you say by measurements from ITM\_Port32.

## 7 Acquisition of an analog signal

You have generated 2 signals. The goal of such a generation was to test the plotter with signals we know the form: sine and cosine at given frequencies.

Now we want to acquire signals on 2 different analog inputs of the Nucleo board. That's why we are going to use the ADC to acquire the signals.

### Required preparation 24: ADC pins

Choose one Zio connector: CN7, CN8, CN0 or CN10. What are the pins that can be used on your chosen connector to acquire analog data?

Tip: open the Nucleo 144 User Manual for the pins on the connector and the datasheet for the use of the pin.

In analog use, a STM32 pin is associated with one or more ADC and one channel. For example, for the STM32H7A3, the STM32 pin PC0 is on channel 10 of ADC1 and ADC2. The name of the function is therefore: ADC12\_INP10.

Chose 2 of the available pins for your use. What are the corresponding ADC and channels? If you want to launch 2 acquisitions, you must use 2 different ADCs.

The first way to use the ADC is the « Software mode ». In this mode, the acquisition is launched by using: "HAL\_ADC\_Start" and result is given after waiting with "HAL\_ADC\_PollForConversion".

We advise you to add 2 modes:

- SINE: the samples are computed sine and cosine signals.
- ANALOG: the samples are acquired by ADC.

### Required work 25: ADC acquisition

Use the ioc view to configure ADC (12 bits) to acquire analog data on the 2 pins you have chosen with 2 ADCs (ADC1 and ADC2).

You should have to set the frequency of the ADC: follow the constraints given in the ioc file. Do not try to generate code with errors in the frequencies. Set properly the divider factors DIVM2, DIVN2 and DIVP2 in order to obtain a right frequency for the ADC (less or equal to 50 MHz).

**Add the following code to read 0 for a 0V signal:**

```
HAL_ADCEx_Calibration_Start(&hadc1, ADC_CALIB_OFFSET, ADC_SINGLE_ENDED) ;  
HAL_Delay(10) ;
```

**It has to be called once, after ADC initializations but before use of ADC in your code.**

Insert a new code to acquire analog data in TIMC Handler. I advise you to keep the generation of the sine and cosine signals: you can switch by testing a variable for the mode (SINE or ANALOG). In this version the mode will be set on ANALOG. We will use this variable, next, to switch mode.

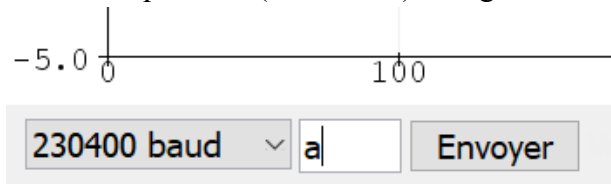
Show that you can acquire signals on the pins. You can wire these pins (one to VCC and the other one to GND) for a first test.

Add the code to toggle the yellow LED as signals are being acquired. So you should see the LED blink as you launch a new acquisition.

## 8 USART communication : configuration from a terminal

It will be useful for the tests to switch between acquisition (ANALOG) and generation (SINE) mode. One way to communicate with your board is to send characters through the USART from a terminal (such as Tera Term) or the interface of the plotter.

The characters are received through your USART. To get them, it is possible to wait for them by polling. A better solution consists in triggering an interrupt Handler code as a character is received. This is done by enabling the global interrupt of your USART (to be configured with ioc view). In this configuration the USARTX\_IRQHandler is triggered as you receive a character. It is possible to read the character by using HAL\_UART\_Receive\_IT (it must be used first once in the main function and then in the Handler).



### Required work 26: mode switching with USART

Use the ioc view to configure your USART to trigger the Handler code of the USART as a character is received. Save and generate code.

Insert a call of HAL\_UART\_Receive\_IT before the loop but after the initialization of the USART (to trigger the handler code as a character will be received).

Write the code in USARTX\_IRQHandler in order to set in ANALOG mode if you receive 'a' or 'A', or to set in SINE mode if you receive 's' or 'S' and start a 50-sample sending. If another letter is received, nothing is done. To read a character re-use HAL\_UART\_Receive\_IT to read the character you want.

Test it and give screenshots of your experiments in your report.

## 9 Generation of a test signal

Now we want to test the limits of the acquisition system. If you have not modified the sampling time, it should be  $T_{sample} = \frac{1}{10kHz} = 100\mu s$ . We would like to improve this value, so it is necessary to test the value with signals.

It is possible to create high speed PWM signals from the board. So we will wire the PWM signals to the analog inputs and analyze the results. The generation of PWM signal requires a Timer and a pin also called a Channel.

First, you are going to generate a pwm signal to make the red LED blink (LD3).



### Required work 27: generation of a PWM signal on the Red LED

It is possible to generate a PWM signal on the Red LED pin.

Thanks to the user manual of the Nucleo Board or the schematic, find the STM32 pin where is connected the red LED.

Thanks to the datasheet, find the timer with a PWM signal connected to it (it must be something like TIMX\_CHx). Give the channel you must use.

Use the ioc view to configure the PWM with the previous parameters: timer, channel.

Tip: to use the LD3, it is necessary to unpin the LD3 pin in the pinout view.

Configure it, in order to obtain a pulse of 25% (the signal is high 25% of the period) and a period you choose.

Save and generate the code.

In your main code, in your initializations, do not forget to add HAL\_TIM\_PWM\_Start to launch the generation of the PWM.

View the result on the LED.

For the tests we would like to modify the frequency (F) of the PWM or the percentage (or ratio) of the pulse (R) by typing a string of characters in the terminal. So the commands will be like this:

- “F 20” to set the frequency of the PWM at 20 kHz
- “R 250” to set the pulse duration of the PWM at 25 %

So the code of your Nucleo has to interpret the characters you send.

You have already begun to write an interpreter of the characters. It was rather easy compared with this new need: in the first version, you had only to interpret the letters “a”, “A”, “s” and “S”.

One way to transform your USART Handler code in a more complex interpreter is to view this code as a state machine. The difference of this state machine with the state machines in digital electronics comes from the fact that the event to switch state is not the rising edge of a clock but the reception of a new character. So our state machine programming is triggered each time we receive a new character (that is the case of USART Handler). The new state is defined by the value of the character.

At first the code is in a WAITLETTER state. It will remain in this state as no character corresponding to a right letter (A F G R) is received.

1. If the letter that is received is a A or G you act as before (by modifying the mode) and you remain in this state to wait another letter.
2. If the letter is F or R the new state is WAITSPACE

In the WAITSPACE state

1. If the character is “ ” (space) the next state is WAITDECIMALDIGIT
2. If the character is different from space the next state is WAITLETTER

In the WAITDECIMALDIGIT state

1. If the character is a decimal symbol ("0" to "9") remain in WAITDECIMALDIGIT and save or compute the value.
2. If the character is different from space the next state is WAITLETTER. The command is entire, execute it.

This state machine can be used to start but it can be improved.

### Required work 28: command interpreter for modifying the PWM signal with the terminal

Write the code in the USART Handler in order to be able to modify the frequency and pulse of the PWM for the LED. Test it with various values.

### Required work 29: generation of a PWM signal (for the test)

Chose a pin on which it is possible to generate a PWM signal.

Use the ioc view to configure a new PWM.

In your main code, in your initializations, do not forget to add HAL\_TIM\_PWM\_Start to launch the generation of the PWM.

Send the signal to one of the analog input. View the result with the plotter.

It is time to test highest frequency limits.

### Required work 30: command interpreter

Make it possible to modify the frequency and period of your second PWM.

Test various frequencies and period with the plotter.

### Required work 31: sampling Time limit

Test other sampling time (less than 100  $\mu$ s).

What is the highest sampling time you can use?

### Required work 32: number of samples limit

What is the limit for the samples if you have two channels to acquire?

Implement it.

### Optional work 33: DMA strategy

Implement a new version of your code in which you use the DMA.

Show the improvements.

### Optional work 34: Improvements

Implement code improvements you choose.

## Table of Contents

HAPPY Plotter .....	1
1 Goals and means.....	3
1.1 The mission .....	3
1.2 The Hardware .....	3
1.3 The software .....	3
STM32CubeIDE.....	3
Terminal .....	3
Plotter .....	3
1.4 The documentation .....	3
2 The main steps of your work.....	4
3 Home works to do before coming at the first lab session .....	4
4 Virtual Com Port and Clocks .....	5
4.1 Communication between your laptop and your Nucleo.....	5
4.2 A first vector.....	7
4.3 Clock control .....	7
4.4 A review of this first part .....	8
5 Green LED blinking with Timer Interrupt .....	9
5.1 Principles of a Timer .....	9
5.2 Control of the SALED.....	10
5.3 Using a Timer (TIMA) for the Green LED.....	10
6 Plotting signals .....	12
6.1 Generation of sine and cosine signals: samples one after the other (1kHz sample rate) 12	
6.2 A Blue button for a functional improvement .....	13
6.3 Improvement of the sample rate.....	13
6.4 A new and better strategy to send more samples .....	14
7 Acquisition of an analog signal .....	16
8 USART communication : configuration from a terminal .....	17
9 Generation of a test signal.....	18