Use of UART Component on Microcontroller

Targeted competences: Use of UART in order to retrieve GPS data at each PPS generation

Hardware: STM32F7 Nucleo board

Framework: Keil µvision (V5.25.2.0) and CubeMX (V5.1.0) from STMircoelectronics

The aim of this document is to show how to use an UART bus in order to read the data from GPS sensor.

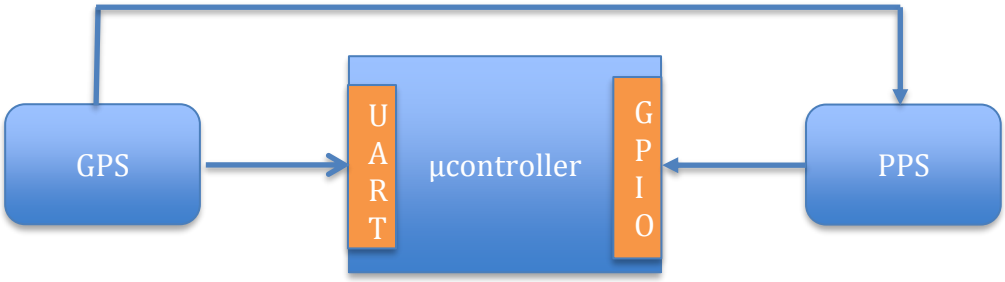

Figure 1: Acquisition Stream

In order to sample the read of the GPS data, we use the PPS (Pulse Per Second) signal comes from the GPS sensor.

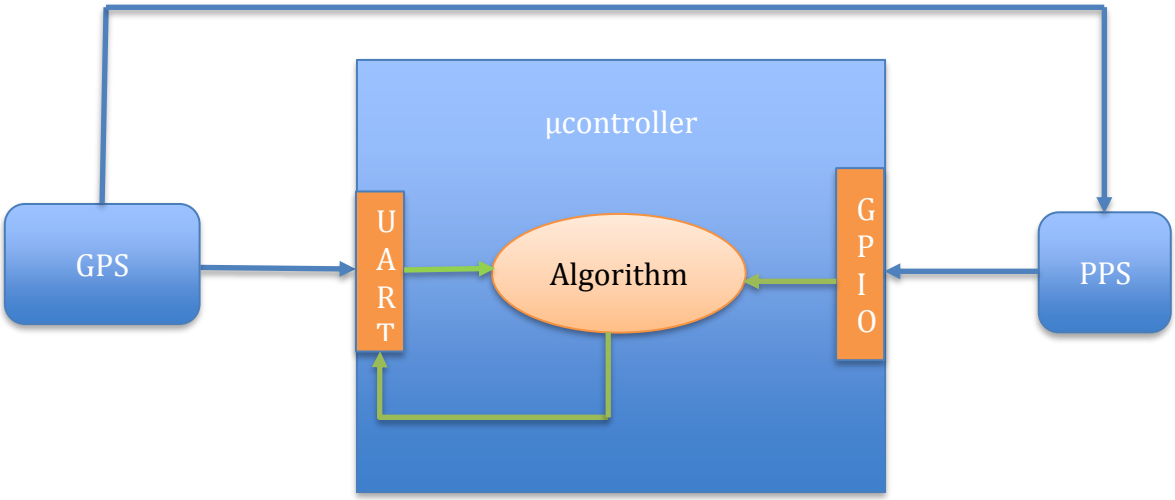

Figure 2: Global Scheme

1. Microcontroller configuration

The first step is to configure the microcontroller. In our case, we use the NUCLEO-F767ZI platform based on a STM32F7 architecture. In order to configure this board we will use the CubeMx software. After the choice of the board (attention the configuration can depend on the board used) you obtain the figure below.

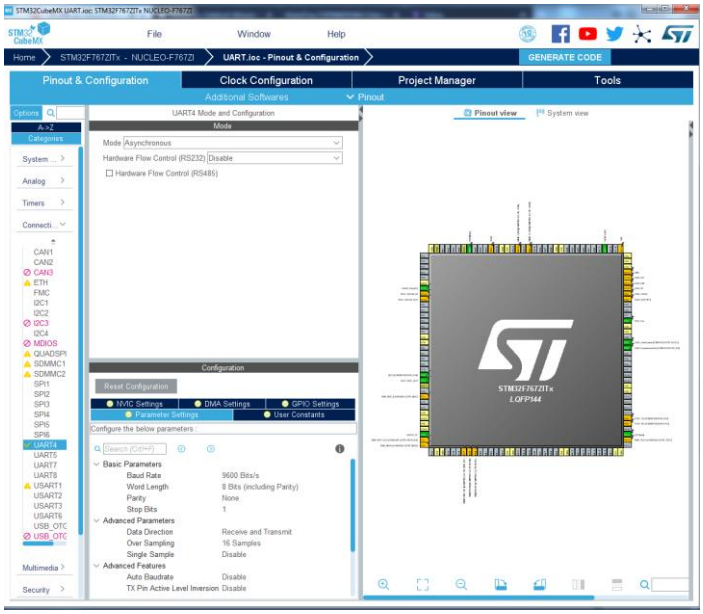

Figure 3: CubeMx configuration for UART

Here, I chose to use the UART bus with the following configuration: 9600bit/s, word length 8 bit, no parity and 1 stop bit. The UART uses no hardware flow control

I configure also the NVIC (enabled the interruption in NVIC settings) in order to interruptions generated when an error or an event appears on the bus.

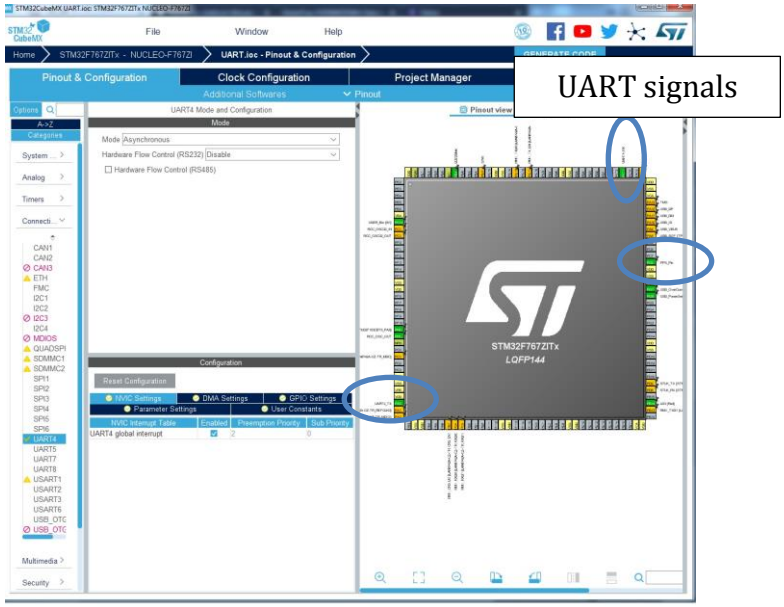

Figure 4:UART signals and NVIC settings

Now we have to configure the clock for the whole board; to do this click on clock configuration item. With this configuration panel, we can choose the frequency of different µcontroller components as CPU frequency, AHB, APB1 buses and so on. I decide to use the maximum frequency of the SYSCLK that is 216MHz. In this case, the frequency of the APB1 and APB2 timer clocks are respectively 108MHz and 216MHz (see Figure 5)..

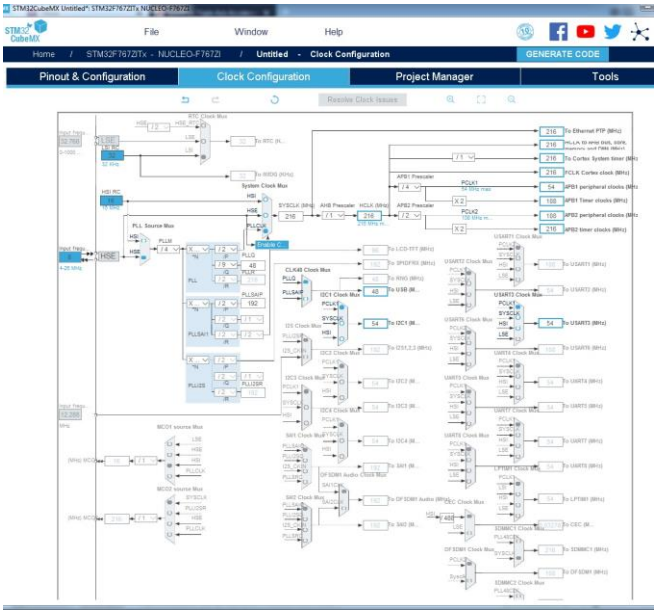

Figure 5: Clock configuration for the board

I have configured the PC6 pin as an external interrupt with a rising edge detection. This GPIO is connected to the PPS signal comes from the GPS board. This external interrupt will be used to start the read of the UART_Rx pin in order to retrieve the NMEA frame.

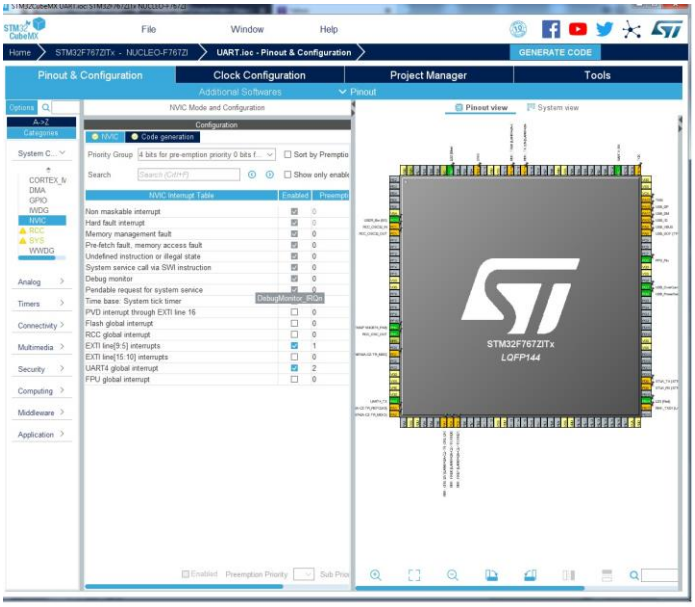

Figure 6: Timer interrupt configuration

We can see in Figure 6 that the EXTI line is enabled and its preemption priority is equal to 1. I have also enabled the UART4 global interrupt with the 2 level priority; this means that the PPS signal (PC6 pin) has a higher interrupt priority level than the uart. Using the UART interrupt avoids us to not realize a blocking pooling on the UART bus in order to know if a data is ready to be read.

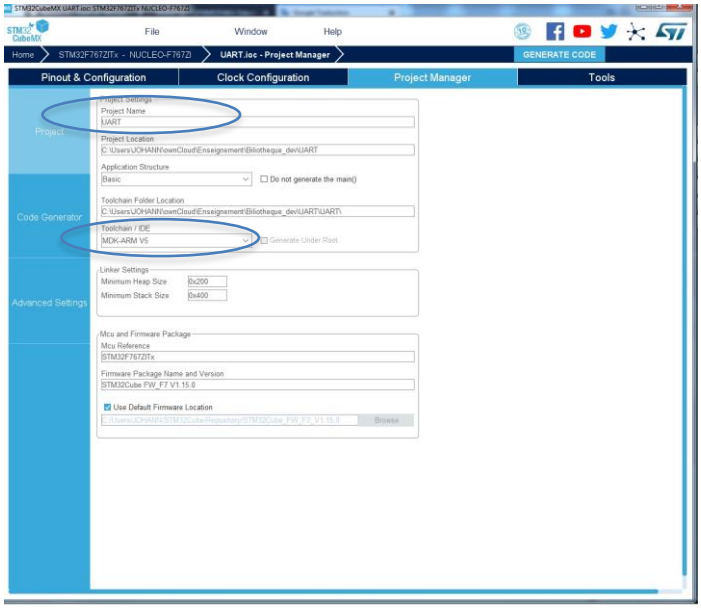

Figure 7: Project manager configuration

You must give a name for your project and choose the toolchain that you want to use; here I choose to use MDK-ARM toolchain.

After clicking on Generate Code, CubeMx generates the application code and creates the MDK-ARM project as shown in the figure below.

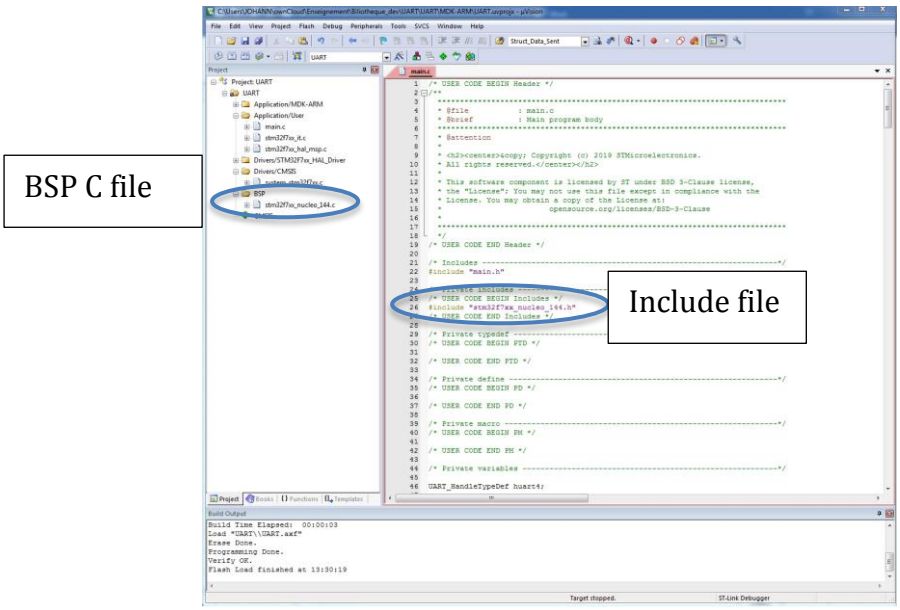

Figure 8: Code generation

As I want to add the BSP function in this project, I include the following header file: stm32f7xx_nucleo_144.h and I add also the associated C file in the project.

In the figure below, you can see all the variable that I have defined to realize the algorithm. Some of them are not used in this code but I created a generic code that you can use if you want to change the configuration of the GPS. The GPS board can receive commands from the µcontroller in order to change for instance the baud rate of the transmission, realize a cold start and so on (see the command documentation here: https://cdn-shop.adafruit.com/datasheets/PMTK_A11.pdf.
The datasheet of the GPS module can be found here: https://cdn-shop.adafruit.com/datasheets/GlobalTop-FGPMMOPA6H-Datasheet-V0A.pdf.
I defined a function named Detection_Protocol allows us to detect the NMEA frame that we want to store.

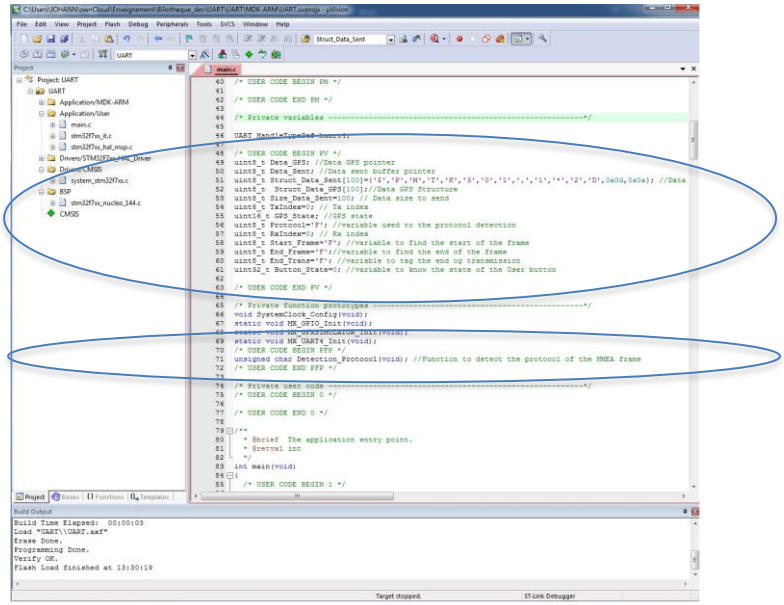

Figure 9: Declaration of all the variables and of the function Detection Protocol

In the code below, I write a part that show you how realize a waiting state driven by a press on the User_Button of the board. This step is not necessary here but it is useful to know how realize it for other application. In order to verify that this step works I switch on a board led.
Note that if the GPS board fixes the satellites before you press the button then the application will begin then even.

When a PPS comes from the GPS board, an external interrupt is generated and the µcontroller jumps at the interrupt routine that you can see in the figure below.

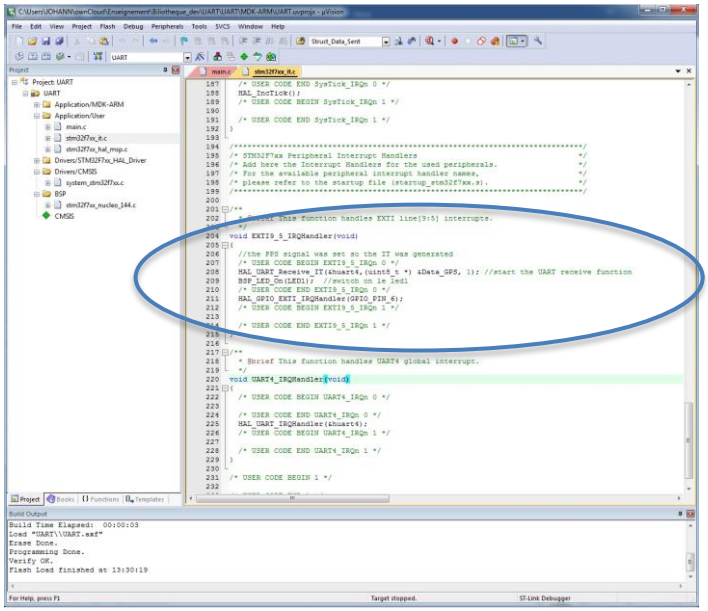

Figure 10: Interrupt routine executed when PPS signal occurs

In this function, I start the UART to receive the GPS data and I switch on the led1 of the board (switch on the led allows knowing if the interrupt routine has been executed).

When a data is received in the UART, an interrupt is generated and then Callback function is called. This callback function first verifies if the UART that generated the interrupt is the uart4 uses by the GPS.

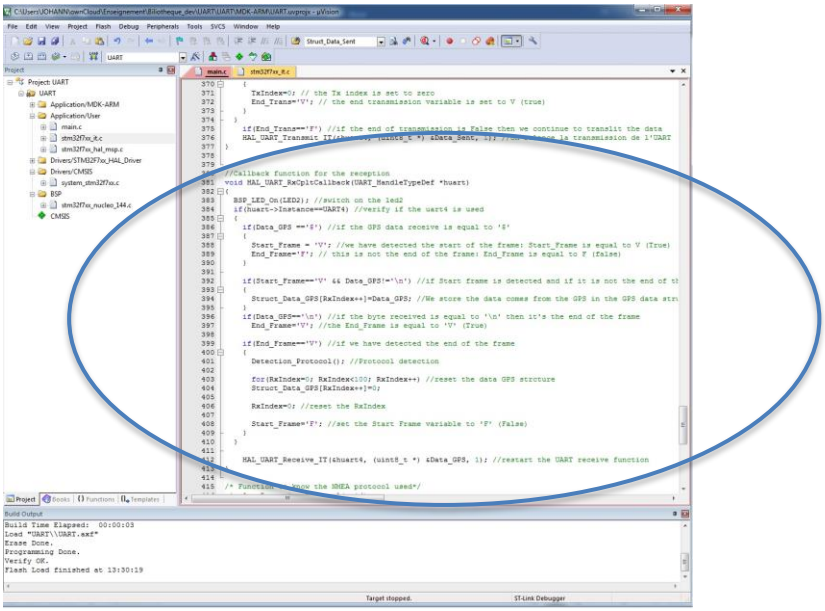

Figure 11: Reception callback function of the UART

Then this function tries to find the start of the frame; for that we test if the first element is a '$' (you can see also that I switch on the led2; it's just for debugging). If we find this tag then we know that it is the start of the frame and we can store all the GPS frame elements until we find the '\n' element. This one specifies that this is the end of the frame (see the datasheet module). When the end of the frame is found (line 399 in the main.c file) then we call the Detection_Protocol function in order to know if the frame

store is the one that we want to decode. After that, we clear all the elements of the data GPS structure in order to store the next valid frame from the GPS. This step is not essential but we are sure that the data structure has no old value inside. At the end we restart the UART receive function; if you do not do this you will receive only one byte from the UART.

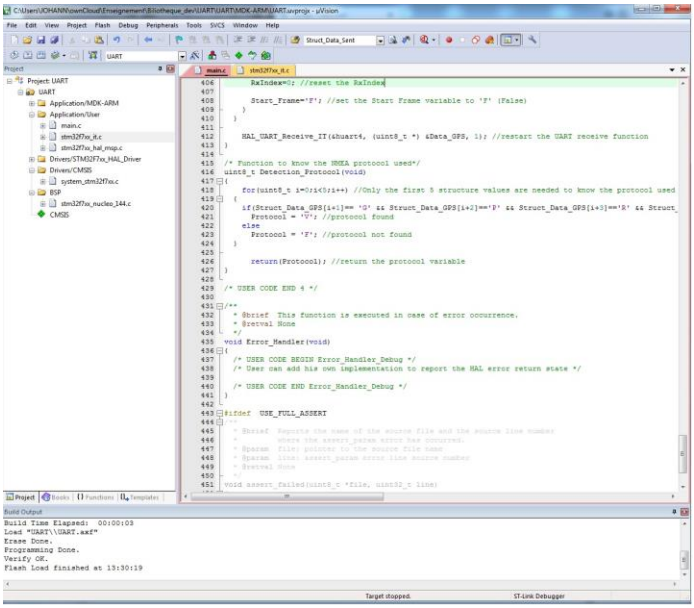

Figure 12: Detection protocol function

In order to analyze only one NMEA frame between all of them, we must determine what is the type of the frame stored. To do this, we just read the first 5 element of the GPS data structure since all the frame type can be decoded from these elements. The first one is always the '$' that tags the start of the frame. Here I directly code the frame type that I want to analyze but you can define this parameter in an include file (this would facilitate the use of this function). You can see that I return only a variable to know if the protocol is the good one (Protocol variable is true 'V' or false 'F'). I have no use this information after to realize some actions but you can do it if you want to complete this code example.

At the end we have to compile the code in order to obtain the executable file for the STM32F7. When the compilation is done without error, we can transfer the code into the board. A demo video can be found here: https://youtu.be/R5l7x_rHsQc