Use of UART Component on Microcontroller

Targeted competences: Use of UART in order to transfer message from one board to another one.

Hardware: STM32F7 Nucleo board

Framework: Keil µvision (V5.25.2.0), Attolic True Studio (V9.3.0) and CubeMX (V5.1.0) from STMircoelectronics

The aim of this document is to show how to use an UART bus in order to transmit some data from one board to another one.

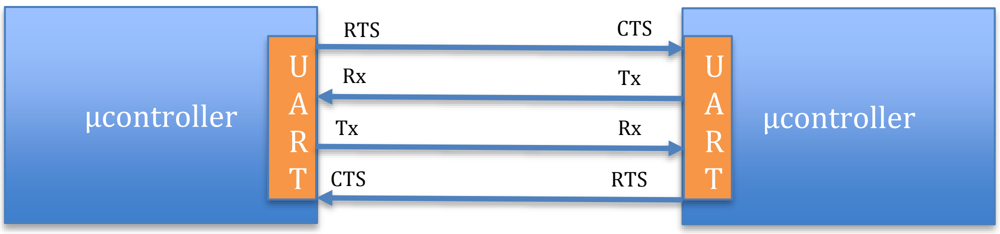


Figure 1: Connection with control flow

The RTS and CTS signals allow us to control the transmission. Indeed, in this case the transmitter performs a demand to the receiver in order to know if it can receive the data that the transmitter wants to send. It is possible to not use the CTS and RTS signals but in this case, we are not sure that the receiver is ready to store the data sent. In several case, we can realize an UART transmission without used these signal.

The algorithms used in this example are presented below.

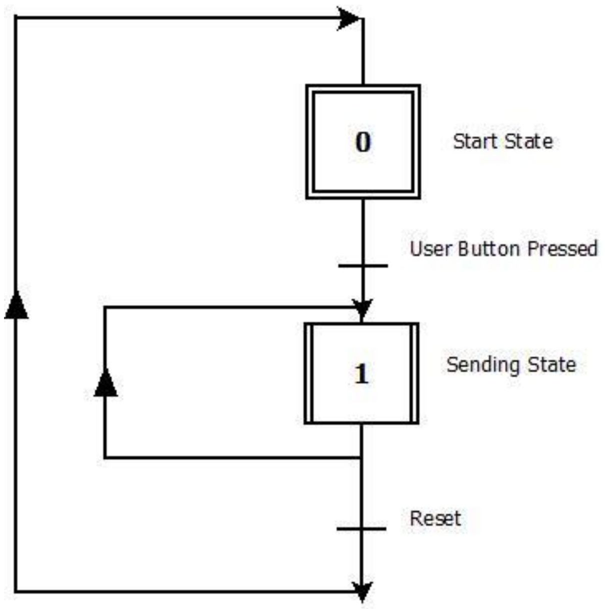


Figure 2: Transmitting algorithm

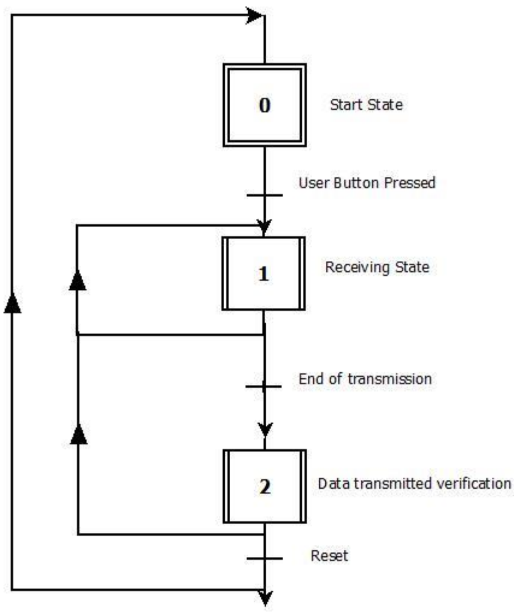
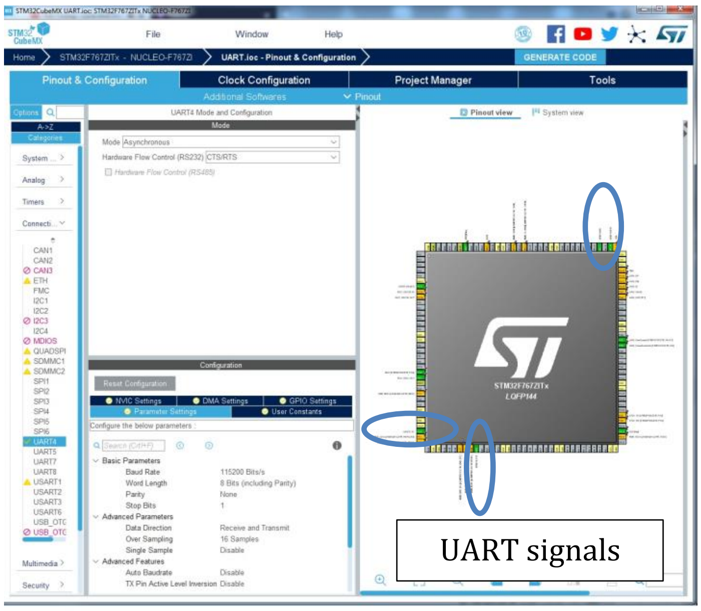


Figure 3: Receiving algorithm

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.



UART4 Mode and Configuration

Mode

Mode: Asynchronous

Hardware Flow Control (RS232): CTS/RTS

Hardware Flow Control (RS485)

Configuration

Reset Configuration

NVIC Settings | DMA Settings | GPIO Settings | Parameter Settings | User Constants

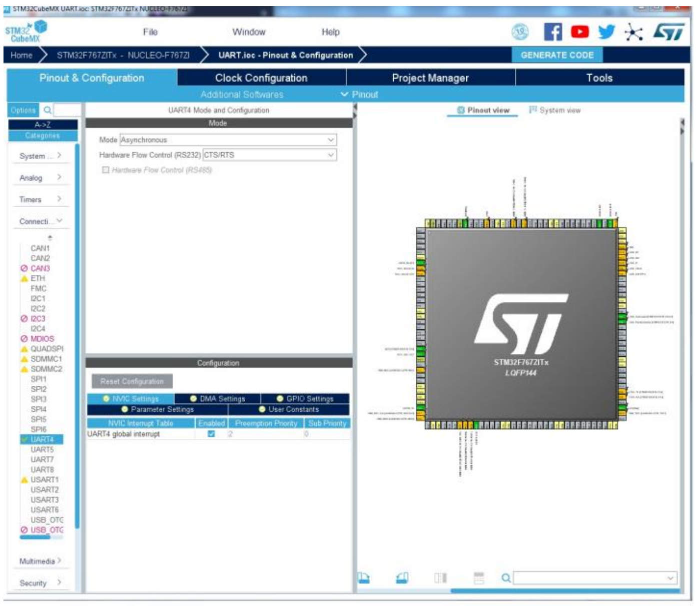
Configure the below parameters :

Basic Parameters	
Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1
Advanced Parameters	
Data Direction	Receive and Transmit
Over Sampling	16 Samples
Single Sample	Disable
Advanced Features	
Auto Baudrate	Disable
TX Pin Active Level Inversion	Disable

Figure 4: CubeMx configuration for UART

Here, I chose to use the UART bus with the following configuration: 115200bit/s, word length 8 bit, no parity and 1 stop bit. The UART uses 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.



UART4 Mode and Configuration

Mode

Mode Asynchronous

Hardware Flow Control (RS232) CTS/RTS

☐ Hardware Flow Control (RS485)

Configuration

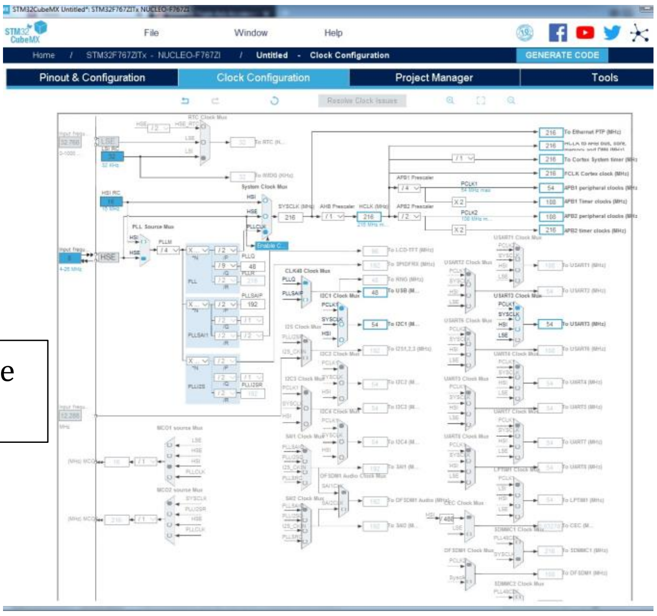
Reset Configuration

NVIC Settings | DMA Settings | GPIO Settings | Parameter Settings | User Constants

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
UART4 global interrupt	✓	2	0

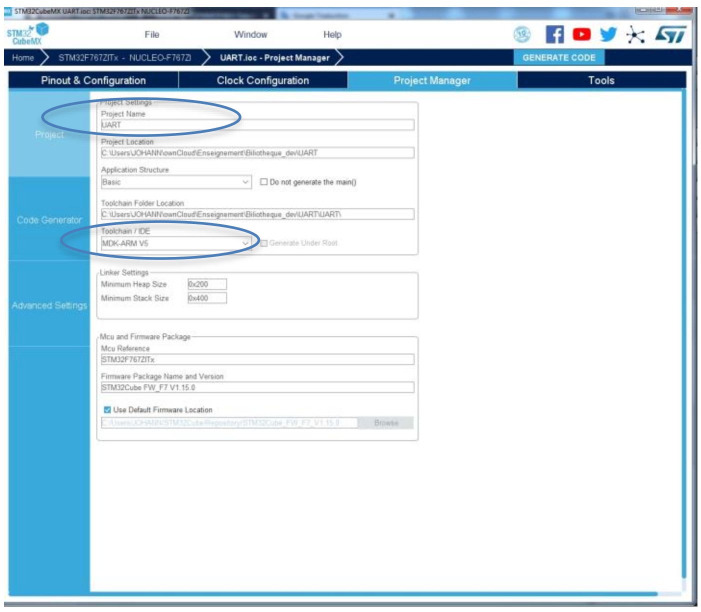
Figure 5:UART NVIC settings

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 6).



Clock source choice

Figure 6: Clock configuration for the board



Project Settings
Project Name: UART
Project Location: C:\Users\JOHANN\ownCloud\Enseignement\Biliotheque_dev\UART
Application Structure: Basic ☐ Do not generate the main()
Toolchain Folder Location: C:\Users\JOHANN\ownCloud\Enseignement\Biliotheque_dev\UART\UART\
Toolchain / IDE: MDK-ARM V5 ☐ Generate Under Root

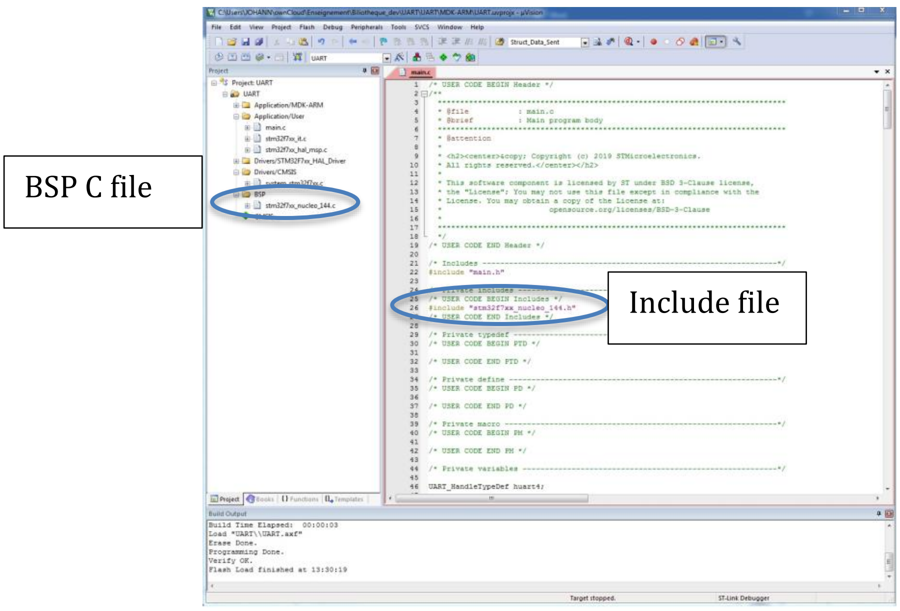
Linker Settings
Minimum Heap Size: 0x200
Minimum Stack Size: 0x400

Mcu and Firmware Package
Mcu Reference: STM32F767ZITx
Firmware Package Name and Version: STM32Cube FW_F7 V1.15.0
☑ Use Default Firmware Location
C:/Users/JOHANN/STM32Cube/Repository/STM32Cube_FW_F7_V1.15.0 Browse

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 for the receiving board project and the Attolic TrueStudio for the transmitting board.

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



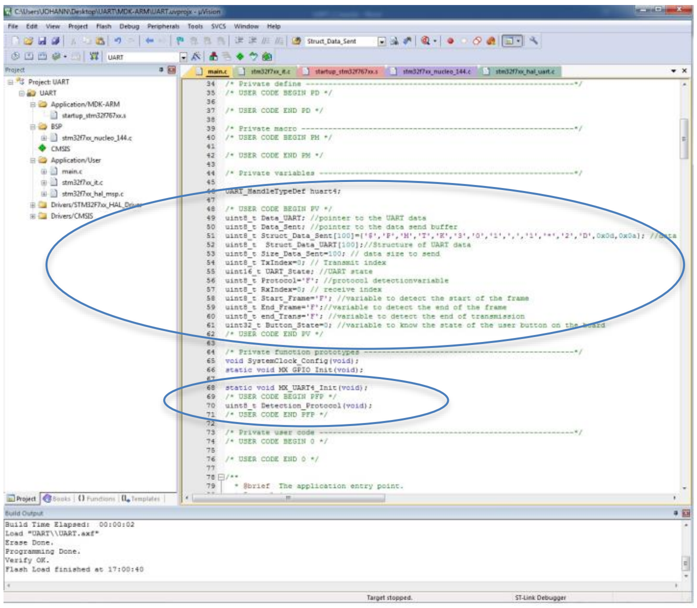
```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2019 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "stm32f7xx_nucleo_144.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

UART_HandleTypeDef huart4;
```

```
Build Time Elapsed:  00:00:03
Load "UART\\UART.axf"
Erase Done.
Programming Done.
Verify OK.
Flash Load finished at 13:30:19
```

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.

I defined a function named Detection_Protocol allows us to verify if the data received is similar to the data sent.



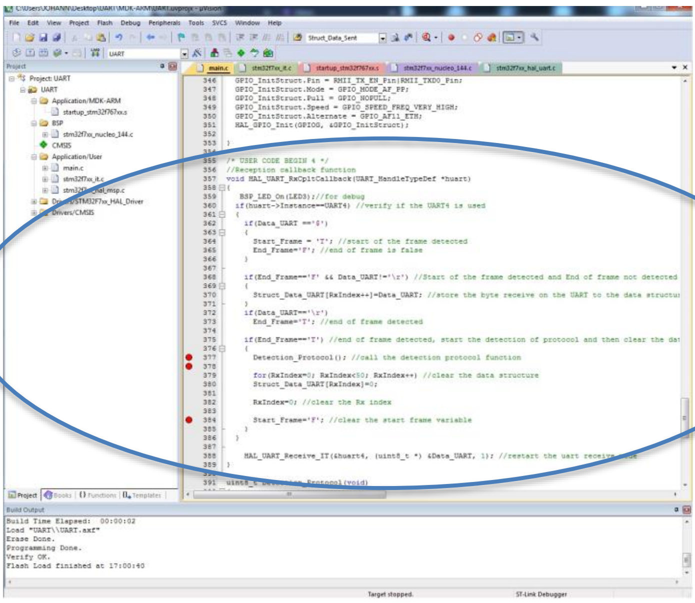
```
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

UART_HandleTypeDef huart4;

/* USER CODE BEGIN PV */
uint8_t Data_UART; //pointer to the UART data
uint8_t Data_Sent; //pointer to the data send buffer
uint8_t Struct_Data_Sent[100]={'$','P','M','T','K','3','0','1',',','1','*','2','D',0x0d,0x0a}; //data
uint8_t  Struct_Data_UART[100];//Structure of UART data
uint8_t Size_Data_Sent=100; // data size to send
uint8_t TxIndex=0; // Transmit index
uint16_t UART_State; //UART state
uint8_t Protocol='F'; //protocol detectionvariable
uint8_t RxIndex=0; // receive index
uint8_t Start_Frame='F'; //variable to detect the start of the frame
uint8_t End_Frame='F';//variable to detect the end of the frame
uint8_t end_Trans='F'; //variable to detect the end of transmission
uint32_t Button_State=0; //variable to know the state of the user button on the board
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);

static void MX_UART4_Init(void);
/* USER CODE BEGIN PFP */
uint8_t Detection_Protocol(void);
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
```

```
Build Time Elapsed:  00:00:02
Load "UART\\UART.axf"
Erase Done.
Programming Done.
Verify OK.
Flash Load finished at 17:00:40
```

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 transmitting board starts its code then this waiting state is left since an interrupt was generated by the UART.

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 (I use this uart in this example).



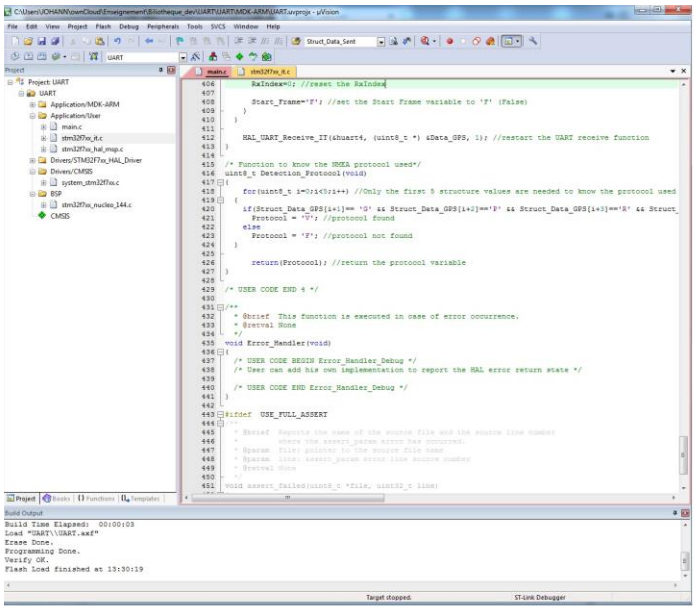
```c
  GPIO_InitStruct.Pin = RMII_TX_EN_Pin|RMII_TXD0_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF11_ETH;
  HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */
//Reception callback function
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
   BSP_LED_On(LED3);//for debug
  if(huart->Instance==UART4) //verify if the UART4 is used
  {
    if(Data_UART =='$')
    {
      Start_Frame = 'T'; //start of the frame detected
      End_Frame='F'; //end of frame is false
    }

    if(End_Frame=='F' && Data_UART!='\r') //Start of the frame detected and End of frame not detected
    {
      Struct_Data_UART[RxIndex++]=Data_UART; //store the byte receive on the UART to the data structu
    }
    if(Data_UART=='\r')
      End_Frame='T'; //end of frame detected

    if(End_Frame=='T') //end of frame detected, start the detection of protocol and then clear the dat
    {
      Detection_Protocol(); //call the detection protocol function

      for(RxIndex=0; RxIndex<50; RxIndex++) //clear the data structure
      Struct_Data_UART[RxIndex]=0;

      RxIndex=0; //clear the Rx index

      Start_Frame='F'; //clear the start frame variable
    }
  }

   HAL_UART_Receive_IT(&huart4, (uint8_t *) &Data_UART, 1); //restart the uart receive mode
}

uint8_t Detection_Protocol(void)
```

```
Build Time Elapsed:  00:00:02
Load "UART\\UART.axf"
Erase Done.
Programming Done.
Verify OK.
Flash Load finished at 17:00:40
```

Figure 11: Reception callback function of the UART

In this Callback function, I start by switching on a led in order to debug my code (I know with this led that the callback function has been called). Then I try to know if the byte received is the start of frame element ('$' in this case). If it is the start of frame I store the data comes from the transmitter board into a data structure until I find the end element ('\r' in this case). At the end of the frame, I call the protocol_detection function to verify if the values sent by the transmitter are the same that those I have store in the data structure. In the last step of the callback function, I clear all the element of the data structure; it is not necessary but as I send always the same values in this example this allows me to verify that the transmission always works.

At the end we restart the UART receive function; if you do not do this, you will receive only one byte from the UART.



```c
      RxIndex=0; //reset the RxIndex

      Start_Frame='F'; //set the Start Frame variable to 'F' (False)
    }
  }

   HAL_UART_Receive_IT(&huart4, (uint8_t *) &Data_GPS, 1); //restart the UART receive function
}

/* Function to know the NMEA protocol used*/
uint8_t Detection_Protocol(void)
{
   for(uint8_t i=0;i<5;i++) //Only the first 5 structure values are needed to know the protocol used
  {
   if(Struct_Data_GPS[i+1]== 'G' && Struct_Data_GPS[i+2]=='P' && Struct_Data_GPS[i+3]=='R' && Struct_
     Protocol = 'V'; //protocol found
   else
     Protocol = 'F'; //protocol not found
  }

     return(Protocol); //return the protocol variable
}

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */

  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
```

```
Build Time Elapsed:  00:00:03
Load "UART\\UART.axf"
Erase Done.
Programming Done.
Verify OK.
Flash Load finished at 13:30:19
```

Figure 12: Detection protocol function

In order to analyze the data receive you can see that I just read the first 5 element of the UART data structure but to be sure you could have to analyze all the data receive. You can add this yourself and verify if all the data are received. You can see that I return only a variable to know if the data is the good one (Protocol variable is true 'T' 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/lw8MnpwkYzk