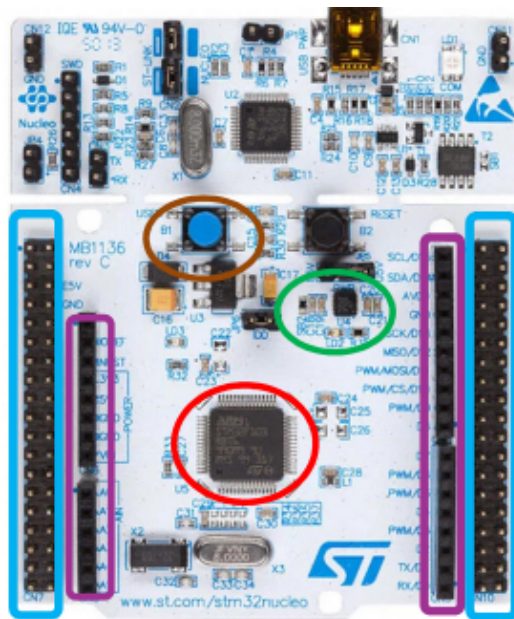


GL 3/2

# COMPTE RENDU



Réalisé Par:

**MEDIMEGH OLFA**  
**ELHAJ ASMA**  
**CHEBIL ILEF**  
**ELLINI YESSINE**

# PARTIE I :

## TRACEUR GPS



### Q1] FORMAT D'UNE TRAME GPS:

On va utiliser deux types de trame dans notre code pour extraire les paramètres demandés.

#### Trame GGA :

\$GPGGA,091709.3,4619.088593,N,00029.479943,W,1,04,2.6,64.7,M,.,.,.,0000\*27

Annotations pour la trame GGA :

- type de trame : \$GPGGA
- heure d'envoi de la trame : 09h 17min 9.3s
- Latitude : 46°19'05.3156"
- Longitude : 00°29'28.7966"
- Type de positionnement : 1 pour le GPS
- Précision horizontale : 2.6
- Altitude en mètres : 64.7
- Champs vides : ,.,.,.
- Clé de contrôle : 0000\*27
- Nombre de satellites utilisés pour le calcul : 4

#### Trame RMC :

\$GPRMC,085348.0,A,4620.066513,N,00025.763711,W,32.5,251.8,200219,,A\*37

Annotations pour la trame RMC :

- type de trame : \$GPRMC
- heure d'envoi de la trame : 08h 53min 48.0s
- Etat des données : A pour valide, V pour invalide
- Latitude : 46°20'3.991" N
- Longitude : 0°25'45.823" W
- Vitesse en noeuds : 32.5
- Angle entre la direction et le nord : 251.8
- Date d'envoi : 20/02/19
- Mode de positionnement : A pour autonome
- Clé de contrôle : A\*37

- **Trame GPGGA:** pour extraire ces informations:
  - Heure (HHMMSS) --> champ 2
  - Altitude --> champ 8
  - Longitude --> champ 4
  - Nombre de satellites --> champ 6
- **Trame GPRMC:** pour extraire ces informations:
  - Heure (HHMMSS) --> champ 2
  - Date (DDMMYY) --> champ 8
  - Position valide ou non --> champ 3
  - Longitude --> champ 5

## Q2] PÉRIPHÉRIQUES UTILISÉS:

- UART1 (pour la réception des données GPS)
- UART2 (pour l'envoi des données vers le PC)
- DMA (pour les transferts sans scrutation)
- GPIO (pour le bouton-poussoir)
- **Clock Control (RCC) :** ( configurer les horloges pour UART, DMA, GPIO, etc.)
- **NVIC (Nested Vectored Interrupt Controller) :**(pour gérer les interruptions, en particulier pour gérer l'interruption générée par le bouton-poussoir.)

## Q3] PINS UTILISÉS:

### UART1 (pour la communication avec le module GPS) :

- RX (Réception) - GPIO Config :
  - Broche: PA10
  - Mode: Input floating
  - Alternate Function: USART1\_RX

### UART2 (pour la communication avec le PC) :

- TX (Transmission) - GPIO Config :
  - Broche: PA2
  - Mode: Alternate function push-pull
  - Vitesse: High speed
  - Alternate Function: USART2\_TX

### Bouton-Poussoir :

- Bouton-Poussoir - GPIO Config :
  - Broche: PC13
  - Mode: Input floating (avec une résistance de pull-up interne activée)
  - Configuration de l'interruption: Front descendant (pour détecter l'appui sur le bouton-poussoir)

## **Q4] INTERRUPTIONS UTILISÉES:**

- DMA1\_Channel5\_IRQHandler (pour la gestion des interruptions du canal DMA utilisé pour UART1)
- DMA1\_Channel7\_IRQHandler (pour la gestion des interruptions du canal DMA utilisé pour UART2)
- EXTI15\_10\_IRQHandler ( pour la detection de l'appui sur le bouton pc13 )

#### Q5] CANAUX DMA UTILISÉS:

- Canal 5 pour UART1 (Réception)
- Canal 7 pour UART2 (Transmission)

#### Q6] VARIABLES, ARRAYS, CONSTANTES:

- Receive\_Buffer stocker les données reçues (UART1).
- Transmit\_Buffer: pour stocker les informations demandé de la trame GPS et l'envoyer (USART2).
- Variables utilisées pour la configuration des peripheriques:

```
// Structures
GPIO_InitTypeDef GPIO_InitStructure;
NVIC_InitTypeDef NVIC_InitStructure;
EXTI_InitTypeDef EXTI_InitStructure;
USART_InitTypeDef USART_InitStructure;
DMA_InitTypeDef DMA_InitStructure;
```

## Q7] INITIALISATION DES PINS:

- Initialisation de PA10:

```
/* Configure USART1 Rx as input floating */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

- Initialisation de PA2:

```
/* Configure USART2 Tx as alternate function push-pull */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

- Initialisation de PC13:

```
// Configure PC13 as input with pull-up
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
GPIO_Init(GPIOC, &GPIO_InitStructure);
```

## Q8]INITIALISATION DES PÉRIPHÉRIQUES:

- USART1 with DMA1:

```
void Config_USART1_RX_WITH_DMA (void)
{
    /* Enable DMA1 clock */
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);

    /* Enable USART1 clocks */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);

    /* Configure USART1 Rx as input floating */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* ===== USART1 configuration ===== */
    /* - BaudRate = 9600 baud - Word Length = 8 Bits- One Stop Bit - No parity
       - Receive enabled */

    USART_InitStructure.USART_BaudRate = 9600;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No ;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx;

    USART_Init(USART1, &USART_InitStructure);

    /*Enable Rx DMA Request (s) on USART1*/
    USART_DMACmd(USART1, USART_DMAREq_Rx, ENABLE);

    /* Enable the USART1*/
    USART_Cmd(USART1, ENABLE);

    /* ===== DMA configuration ===== */

    /*DMAx channelx (UART1 RX) configuration */
    /*DMA_DeInit(DMA1_Channel5);
    DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t) &USART1->DR;
    DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)Receive_Buffer;
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
    DMA_InitStructure.DMA_BufferSize = 100;
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
    DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;

    DMA_Init(DMA1_Channel5, &DMA_InitStructure);

    /* Enable DMAx Channely Transfer Complete/ Half Transfer interrupts */
    DMA_ITConfig(DMA1_Channel5, DMA_IT_TC, ENABLE);

    /* Enable DMA1 Channelx transfer */
    DMA_Cmd(DMA1_Channel5, ENABLE);

    /* Enable and set DMAx Channel y Interrupt */
    NVIC_InitStructure.NVIC_IRQChannel = DMA1_Channel5_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 2;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```



- USART2 with DMA1:

```
void Config_USART2_TX_WITH_DMA(void)
{
    /* Enable DMA1 clock */
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);

    /* Enable USART2 clocks */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

    /* Configure USART2 Tx as alternate function push-pull */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* ===== USART2 configuration ===== */
    /* - BaudRate = 9600 baud - Word Length = 8 Bits- One Stop Bit - No parity
       - Transmit enabled */

    USART_InitStructure.USART_BaudRate = 9600;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Tx;

    USART_Init(USART2, &USART_InitStructure);

    /* Enable Tx DMA Request (s) on USART2 */
    USART_DMACmd(USART2, USART_DMAREq_Tx, ENABLE);

    /* Enable the USART2 */
    USART_Cmd(USART2, ENABLE);

    /* ===== DMA configuration ===== */

    /* DMAx channelx (UART2 TX) configuration */
    /* DMA_DeInit(DMA1_Channel7);
    DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&USART2->DR;
    DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)Transmit_Buffer;
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralDST;
    DMA_InitStructure.DMA_BufferSize = 100;
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
    DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;

    DMA_Init(DMA1_Channel7, &DMA_InitStructure);

    // /* Enable DMAx Channely Transfer Complete/ Half Transfer interrupts */
    DMA_ITConfig(DMA1_Channel7, DMA_IT_TC, ENABLE);

//disable dma
DMA_Cmd(DMA1_Channel7, DISABLE);

/* Enable and set DMAx Channel y Interrupt */
NVIC_InitStructure.NVIC_IRQChannel = DMA1_Channel7_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 2;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
}
```



- Bouton PC13 with EXTI:

```
//===== BUTTON CONFIG =====//

void Config_Button_Exti(void)
{
    // Enable GPIOC clock
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);

    // Configure PC13 as input with pull-up
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    // Enable AFIO clock
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);

    // Connect EXTI Line 13 to PC13 pin
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource13);

    // Configure EXTI Line 13 to detect falling edge (button press)
    EXTI_InitStructure.EXTI_Line = EXTI_Line13;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);

    // Enable and set EXTI Line 13 Interrupt to the lowest priority
    NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 3;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

## Q9] LA BOUCLE WHILE DU FICHIER MAIN:

```
while(1)
{
}
```

La boucle while reste vide car tous le traitement est donnés par les interruptions.

## Q10] LE CODE AU NIVEAU DE HANDLERS:

- DMA1\_Channel5\_IRQHandler:

```
void DMA1_Channel5_IRQHandler(void)
{
    /* Test on DMAx Channely Transfer Half interrupt */
    if(DMA_GetITStatus(DMA1_IT_TC5))

        DMA_ClearITPendingBit(DMA1_IT_TC5);
}
```

- DMA1\_Channel7\_IRQHandler:

```
void DMA1_Channel7_IRQHandler(void)
{
    /* Test on DMAx Channely Transfer complete interrupt */
    if(DMA_GetITStatus(DMA1_IT_TC7))

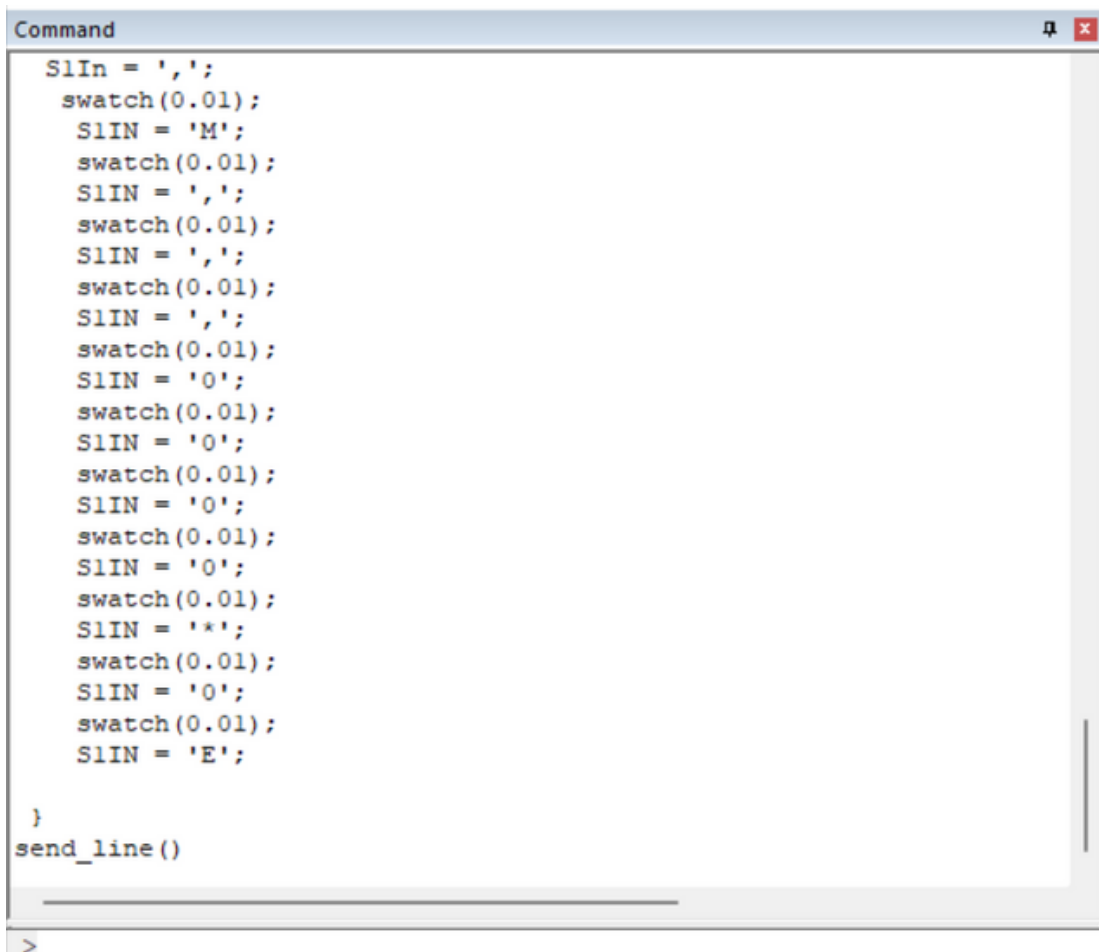
        DMA_ClearITPendingBit(DMA1_IT_TC7);
        DMA_Cmd(DMA1_Channel7, DISABLE);
}
```

- EXTI15\_10\_IRQHandler:

```
void EXTI15_10_IRQHandler(void)
{
    // Check if the interrupt was generated by EXTI Line 13
    if (EXTI_GetITStatus(EXTI_Line13) != RESET)
    {
        // Button (PC13) is pressed, handle the button press
        TransmitData();
        //enable DMA
        DMA_Cmd(DMA1_Channel7, ENABLE);
        // Clear the interrupt flag
        EXTI_ClearITPendingBit(EXTI_Line13);
    }
}
```

## Q10] CAPTURES D'ECRAN [MODE DEBOGAGE]:

Avant de lancer l'exécution du programme, on ajoute une fonction qui permet de simuler l'envoi de données vers l'USART1 que notre application va recevoir et placer dans la chaîne Receive\_buffer.  
on la compile.



```
Command
SlIn = ',';
swatch(0.01);
SlIN = 'M';
swatch(0.01);
SlIN = ',';
swatch(0.01);
SlIN = ',';
swatch(0.01);
SlIN = ',';
swatch(0.01);
SlIN = '0';
swatch(0.01);
SlIN = '0';
swatch(0.01);
SlIN = '0';
swatch(0.01);
SlIN = '0';
swatch(0.01);
SlIN = '*';
swatch(0.01);
SlIN = '0';
swatch(0.01);
SlIN = 'E';

}
send_line()
>
```

Au niveau de la ligne de commande dans la fenêtre « Command », on saisie le nom de la fonction `send_line()` + enter . puis on observe le contenu de la variable `receive_Buffer`.

Name	Value	Type
Receive_Buffer	0x20000030 Receive_B...	uchar[100]
[0]	0x24 'S'	uchar
[1]	0x47 'G'	uchar
[2]	0x50 'P'	uchar
[3]	0x47 'G'	uchar
[4]	0x47 'G'	uchar
[5]	0x41 'A'	uchar
[6]	0x2C ','	uchar
[7]	0x30 '0'	uchar
[8]	0x36 '6'	uchar
[9]	0x34 '4'	uchar
[10]	0x30 '0'	uchar
[11]	0x33 '3'	uchar
[12]	0x36 '6'	uchar
[13]	0x2E '.'	uchar
[14]	0x32 '2'	uchar
[15]	0x38 '8'	uchar
[16]	0x39 '9'	uchar
[17]	0x2C ','	uchar
[18]	0x34 '4'	uchar
[19]	0x38 '8'	uchar
[20]	0x33 '3'	uchar
[21]	0x36 '6'	uchar

Ensuite on navigue Peripherals -> General Purpose I/O -> GPIO C

The screenshot shows the "General Purpose I/O C (GPIOC)" configuration window. It contains several sections:

- Pin Configuration Table:** A table listing pins PC.8 through PC.15 and their current configurations.
- Selected Port Pin Configuration:** Two dropdown menus for "MODE:" (set to "0: Input") and "CNF:" (set to "1: Floating Input").
- Configuration & Mode Settings:** Fields for "GPIOC\_CRH:" (0x44844444) and "GPIOC\_CRL:" (0x44444444).
- GPIO Section:** Fields for "GPIOC\_IDR:", "GPIOC\_ODR:", "GPIOC\_LCKR:", and "Pins:". To the right are bit fields for bits 15-8 and 7-0, with a "LCKK" checkbox.
- Settings:** A field for "Clock Enabled" which is checked.

Pin	CNF
PC.8	Floating Input
PC.9	Floating Input
PC.10	Floating Input
PC.11	Floating Input
PC.12	Floating Input
PC.13	Input with pull-up/down
PC.14	Floating Input
PC.15	Floating Input

Selected Port Pin Configuration  
 MODE: 0: Input CNF: 1: Floating Input

Configuration & Mode Settings  
 GPIOC\_CRH: 0x44844444 GPIOC\_CRL: 0x44444444

GPIO  
 GPIOC\_IDR: 0x00000000  
 GPIOC\_ODR: 0x00002000  
 GPIOC\_LCKR: 0x00000000  
 Pins: 0x00000000

15 Bits 8 7 Bits 0  
 LCKK [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]  
 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]

Settings: Clock Enabled

L'appui sur le Pin 13 permet de remplir le tableau Transmit\_Buffer avec les données filtrées et son envoi a travers USART2

General Purpose I/O C (GPIOC)

Pin	CNF
PC.8	Floating Input
PC.9	Floating Input
PC.10	Floating Input
PC.11	Floating Input
PC.12	Floating Input
PC.13	Input with pull-up/down
PC.14	Floating Input
PC.15	Floating Input

Selected Port Pin Configuration  
MODE: 0: Input CNF: 1: Floating Input

Configuration & Mode Settings  
GPIOC\_CRH: 0x44844444 GPIOC\_CRL: 0x44444444

GPIOC

GPIOC_IDR	GPIOC_ODR	GPIOC_LCKR	Pins
0x00002000	0x00002000	0x00000000	0x00002000

Settings: Clock Enabled

Watch 2

Name	Value	type
Transmit_Buffer	0x20000094 Transmit_...	uchar[100]
[0]	0x30 '0'	uchar
[1]	0x36 '6'	uchar
[2]	0x34 '4'	uchar
[3]	0x30 '0'	uchar
[4]	0x33 '3'	uchar
[5]	0x36 '6'	uchar
[6]	0x2E '.'	uch
[7]	0x32 '2'	uch
[8]	0x38 '8'	uch
[9]	0x39 '9'	uch
[10]	0x2C ','	uch
[11]	0x30 '0'	uch
[12]	0x30 '0'	uch
[13]	0x37 '7'	uch
[14]	0x34 '4'	uch
[15]	0x30 '0'	uch
[16]	0x2E '.'	uch
[17]	0x39 '9'	uch
[18]	0x33 '3'	uch
[19]	0x37 '7'	uch

UART #2

064036.289,00740.9373,E,04,200.2,M,

UART #1 | Call Sta... | UART #2 | Watch 2 | Memory 1

# PARTIE II :

Pour pouvoir utiliser le code donné dans l'énoncé, il faut développer une classe Serial qui permet la communication série via l'interface UART, il faut aussi définir les méthodes pour l'initialisation, l'envoi et la réception des caractères en utilisant UART2 (TxPin = PA2 et RxPin = PA3).

- **Fichiers à Implémenter:**

**1. Serial.h:** Ce fichier d'en-tête déclarerait la classe Serial et définirait son interface publique, y compris les constructeurs et les méthodes Printchar et Getchar.

**2. Serial.cpp:** Ce fichier source contiendrait les définitions des méthodes de la classe Serial.

- **Rôles des méthodes et des classes:**

La **classe Serial** est conçue pour gérer la communication série via UART2 (TxPin = PA2, RxPin = PA3)

=> Serial représente une abstraction de la communication série

Le constructeur **Serial(PinName txPin, PinName rxPin)** initialise la communication série pour la transmission (Tx) et la réception (Rx).

**Printchar(char c)** envoie un caractère via l'UART2.

**Getchar()** lit un caractère de l'UART2.

=>PrintChar et GetChar facilitent l'envoi et la réception de données.

**GPIO\_InitStruct\_RX** et **GPIO\_InitStruct\_TX** initialisent les configurations des pins Rx et Tx.

**Usart\_INIT** initialise les paramètres de l'USART.



- **Contenu des Méthodes et Fonctions:**

**PrintChar** utilise USART\_SendData.

**GetChar** utilise USART\_ReceiveData.

**GPIO\_InitStruct\_RX** et **GPIO\_InitStruct\_TX** initialisent les GPIO selon les spécifications.

**Usart\_INIT** configure les paramètres de l'USART.

- **Utilisation de Fonctions de la Librairie ST:**

Les méthodes Printchar et Getchar font appel aux fonctions de la **bibliothèque ST** existante pour manipuler les registres UART et envoyer/recevoir des données.

*Pour mieux expliquer la démarche nous avons fait une petite implémentation:*

## 1) Serial.h :

```
#include "stm32f10x.h"
#include "Pin_Mode_Names.h"

#ifndef _STM32_WRAPPER_UART_H
#define _STM32_WRAPPER_UART_H

#ifdef __cplusplus
extern "C" {
#endif

class Serial {
public:
    Serial(PinName txPin, PinName rxPin);
    void PrintChar(char c);
    uint8_t GetChar();
    void GPIO_InitStruct_TX(PinName TxPin);
    void GPIO_InitStruct_RX(PinName RxPin);
    void Usart_INIT();
    static GPIO_TypeDef* Get_gpioport (PinName pin)
private:
    GPIO_TypeDef* txPort;
    uint16_t txPin;
    GPIO_TypeDef* rxPort;
    uint16_t rxPin;
};

#ifdef __cplusplus
}
#endif
#endif
```

## 2) Serial.cpp :

```
#include "Serial.h"
#include "stm32f10x.h"
#include "stm32f10x_usart.h"
#include "stm32_wrapper_gpio.h"
```

```
GPIO_InitTypeDef GPIO_InitStructure;
USART_InitTypeDef USART_InitStructure;
```

**//Construteur**

**Serial(PinName txPin, PinName rxPin)**

```
{
    GPIO_InitStruct_RX(rxPin);
    GPIO_InitStruct_TX (txPin);
    Usart_INIT();
}
```

**// Fonction pour envoyer un caractère**

**void Printchar(uint8\_t data){**  
 USART SendData(USART2, data);  
**}**

**// Fonction pour lire un caractère**

**uint8\_t Getchar(){**  
 return USART\_ReceiveData(USART2) & 0xFF;  
**}**

**// Initialisation du pin de réception (Rx)**

**void GPIO\_InitStruct\_RX(PinName RxPin)**

```
{
    int gpioID = (RxPin & 0xF0) >> 4;
    RCC_APB2PeriphClockCmd(1<<(gpioID+2), ENABLE);
    GPIO_TypeDef* GPIOX = Get_gpioport (RxPin);
```

```
GPIO_InitStructure.GPIO_Pin = 1<<(RxPin & 0x0F);
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOx, &GPIO_InitStructure);
}
```

**// Initialisation du pin de transmission (Tx)**

**void GPIO\_InitStruct\_TX (PinName TxPin)**

```
{
    int gpioID = (TxPin & 0xF0) >> 4;
    RCC_APB2PeriphClockCmd(1<<(gpioID+2), ENABLE);
    GPIO_TypeDef* GPIOx = Get_gpioport (TxPin);
    GPIO_InitStructure.GPIO_Pin = 1<<(TxPin & 0x0F);
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    .GPIO_Init(GPIOx, &GPIO_InitStructure);
}
```

**// Initialisation de l'USART**

**void Usart\_INIT()**

```
{
    USART_InitStructure.USART_BaudRate = 9600;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity= USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl=
        USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Tx;
    USART_Init(USART2, &USART_InitStructure);
    USART_Cmd(USART2, ENABLE);
}
```