

Rapport de TP4

UART

Architecture des ordinateurs

Groupe A – Mercredi 14h/17h15

24/03/2021

RIGHI Racim

Exercice 1 :

Q : A quoi correspond le paramètre baud rate ?

Il correspond à la vitesse à laquelle l'information est transférée dans un canal de communication.

Qu'est-ce que cela implique au niveau du récepteur (ici le PC) ?

Le pc doit être configuré pour avoir exactement la même vitesse de transmission (115200 baud/s dans mon cas)

A quoi sert l'activation de l'option 'Parity' ?

Elle sert à vérifier l'exactitude des informations transmises.

Expliquer les paramètres de la structure UART_HandleTypeDef associée à huart2 au niveau de la configuration de l'USART2 (MX_USART2_UART_Init ())

```
/* USER CODE END USART2_Init 1 */
huart2.Instance = USART2; Adresse de base des registres de l'USART (on y trouve les registres de: status, data, baud rate, controle et prescaler
huart2.Init.BaudRate = 115200; Vitesse de transmission de l'UART
huart2.Init.WordLength = UART_WORDLENGTH_8B; Nombre de bits par transfert (ici on choisit 8)
huart2.Init.StopBits = UART_STOPBITS_1; Nombre de bits d'arrêt et de parité à transférer, dans ce cas 1 et 0
huart2.Init.Parity = UART_PARITY_NONE; respectivement
huart2.Init.Mode = UART_MODE_TX_RX; Mode d'utilisation, TX_RX veut dire envoi et reception
huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE; Activer/désactiver la synchronisation via les signaux CTS/RTS
huart2.Init.OverSampling = UART_OVERSAMPLING_16; Activer/désactiver la fonction x16 oversampling qui
if (HAL_UART_Init(&huart2) != HAL_OK) permet d'avoir une meilleure tolérance aux erreurs de
{ timers
Error_Handler();
}
```

Paramètres de la fonction d'initialisation

Q : Ouvrir le fichier stm32f4xx_hal_msp.c et analyser la fonction HAL_UART_MspInit()

Qu'est-ce qui est réalisé ?

Cette fonction configure les broches utilisées (PA2 et PA3) pour pouvoir utiliser les fonctions alternatives et l'USART2

MSP = MCU Support Package, qui correspond à un fichier qui définit toutes les fonctions d'initialisation selon la configuration de l'utilisateur.

En quoi est-ce important ?

L'USART2 étant connectée avec les broches PA2 et PA3 au port COM, leur configuration est nécessaire pour pouvoir communiquer avec l'ordinateur. De plus, étant donné que la vitesse doit être identique pour l'émetteur et le récepteur, on aura besoin des différentes horloges pour synchroniser les transmissions.

Expliquer chaque ligne de code

```

88 void HAL_UART_MspInit(UART_HandleTypeDef* huart)
89 {
90     GPIO_InitTypeDef GPIO_InitStruct = {0};
91     if(huart->Instance==USART2) Vérifier que le paramètre est bien USART2, qu'on a configuré dans STM32CubeMX
92     {
93         /* USER CODE BEGIN USART2_MspInit 0 */
94
95         /* USER CODE END USART2_MspInit 0 */
96         /* Peripheral clock enable */
97         __HAL_RCC_USART2_CLK_ENABLE(); Activer l'horloge de l'USART2 et de GPIOA pour synchroniser les
98                                         transmissions
99         __HAL_RCC_GPIOA_CLK_ENABLE();
100     /**USART2 GPIO Configuration
101         PA2 -----> USART2_TX
102         PA3 -----> USART2_RX
103     */
104     GPIO_InitStruct.Pin = GPIO_PIN_2|GPIO_PIN_3; Selectionner les ports PA2 et PA3
105     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP; Les configurer en mode fonction alternative push pull
106     GPIO_InitStruct.Pull = GPIO_NOPULL; Configurer la fréquence et l'absence de pull
107     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
108     GPIO_InitStruct.Alternate = GPIO_AF7_USART2; Selectionner la fonction alternative correspondant à l'USART2
109     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct); Appeler la fonction d'initialisation de GPIO
110
111     /* USER CODE BEGIN USART2_MspInit 1 */
112
113     /* USER CODE END USART2_MspInit 1 */
114 }

```

Fonction d'initialisation de l'UART

Modification du programme pour envoyer « Hello CY » et les chiffres de 0 à 9 avec un délai de 1 seconde :

On utilise la fonction de HAL : « HAL_UART_Transmit »

```

91 /* USER CODE BEGIN 2 */
92
93 char *str = "Hello CY";
94 char new_line[2] = "\r\n";
95 char *n = "";
96
97 HAL_UART_Transmit(&huart2, (uint8_t *) str, 8, 0);
98 HAL_UART_Transmit(&huart2, (uint8_t *) new_line, 2, 0);
99
100 for(int i=0 ; i<10 ; i++)
101 {
102     HAL_UART_Transmit(&huart2, (uint8_t *) new_line, 2, 0);
103     n = ""+i;
104     HAL_UART_Transmit(&huart2, (uint8_t *) n, 1, 1000);
105 }
106
107 /* USER CODE END 2 */

```

Exercice 2 :

Partie 1 : en utilisant la scrutation

En utilisant les fonctions HAL_UART_Transmit et receive dans la boucle while()

Partie 2 : en utilisant les interruptions

Tout d'abord, il faut activer les interruptions globales dans NVIC settings de l'USART2, ensuite tout ce qu'on a à faire est d'implémenter les fonctions de callback et faire en sorte qu'elles transmettent et reçoivent les données. Ces fonctions sont « HAL_UART_RxCpltCallback() » pour la réception et « HAL_UART_TxCpltCallback » pour l'envoi.

```

230 /* USER CODE BEGIN 4 */
231 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
232 {
233     HAL_UART_Transmit_IT(&huart2, (uint8_t *)rx_buff, 8);
234 }
235 }
236
237 void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
238 {
239     HAL_UART_Receive_IT(&huart2, (uint8_t *)rx_buff, 8);
240 }

```

Fonctions de callback de l'uart

Exercice 3 :

En partant du résultat de l'exercice 2 avec les interruptions, on peut facilement créer le jeu du plus ou moins en modifiant les fonctions de callback comme suit :

```

205 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
206 {
207     if(atoi(rx_buff) < nombre_mystere) // convertir le caractère en int et le comparer au nombre mystère
208     {
209         strcpy(tx_buff, "Plus!\r\n");
210         nb_tentative++;
211     }
212     else if(atoi(rx_buff) > nombre_mystere)
213     {
214         strcpy(tx_buff, "Moins!\r\n");
215         nb_tentative++;
216     }
217     else
218     {
219         strcpy(tx_buff, "Bingo!\r\n");
220         HAL_UART_Transmit_IT(&huart2, (uint8_t *)tx_buff, 8);
221
222         strcpy(tx_buff1, "Nombre de tentatives: "+nb_tentative);
223         HAL_UART_Transmit(&huart2, (uint8_t *)tx_buff1, 24, 100);
224     }
225     // Envoyer le message selon l'entrée de l'utilisateur
226     HAL_UART_Transmit_IT(&huart2, (uint8_t *)tx_buff, 8);
227 }
228

```

Fonctions de callback pour le jeu du plus ou moins



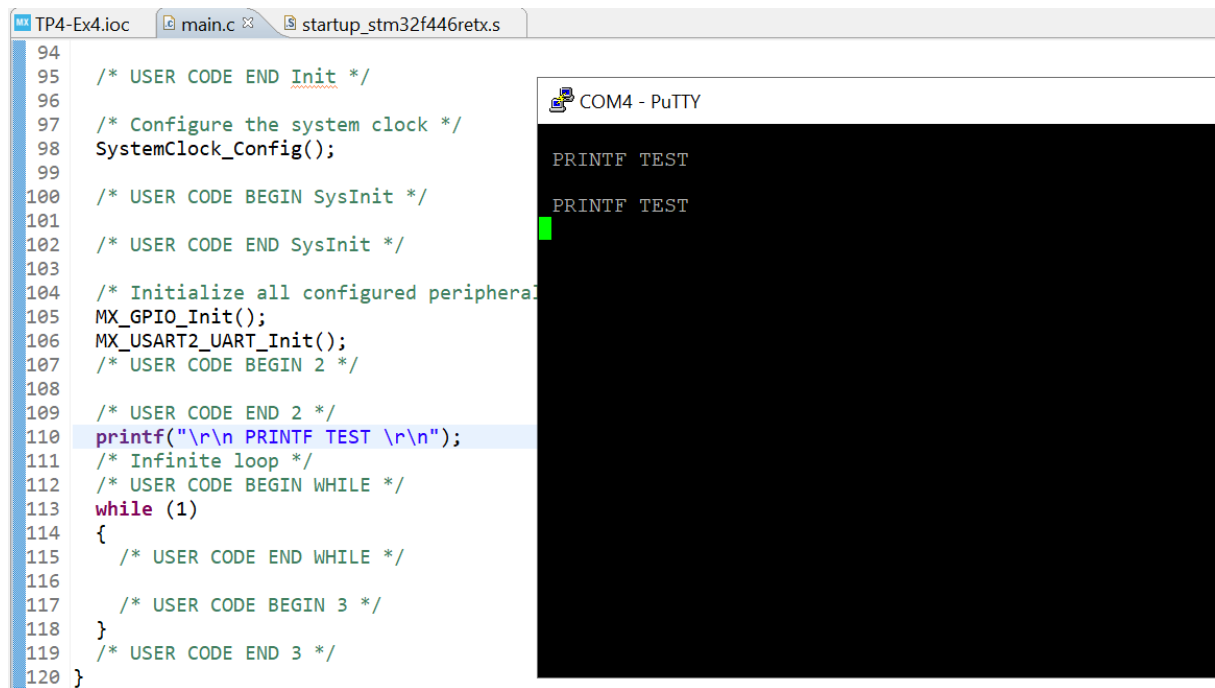
COM4 - PuTTY

```

Jeu du plus ou moins!
Jeu du plus ou moins!
Plus!
Plus!
Moins
Moins
Moins
Bingo

```

Exercise 4 :



The image shows a screenshot of a development environment. On the left, a code editor displays a C program for an STM32 microcontroller. The code includes initialization functions like `SystemClock_Config()`, `MX_GPIO_Init()`, and `MX_USART2_UART_Init()`. It features a `while (1)` loop containing a `printf` statement: `printf("\r\n PRINTF TEST \r\n");`. On the right, a terminal window titled 'COM4 - PuTTY' shows the output of the program, displaying 'PRINTF TEST' on two separate lines, indicating that the `printf` function is working correctly.

```
94
95  /* USER CODE END Init */
96
97  /* Configure the system clock */
98  SystemClock_Config();
99
100 /* USER CODE BEGIN SysInit */
101
102 /* USER CODE END SysInit */
103
104 /* Initialize all configured peripherals */
105 MX_GPIO_Init();
106 MX_USART2_UART_Init();
107 /* USER CODE BEGIN 2 */
108
109 /* USER CODE END 2 */
110 printf("\r\n PRINTF TEST \r\n");
111 /* Infinite loop */
112 /* USER CODE BEGIN WHILE */
113 while (1)
114 {
115     /* USER CODE END WHILE */
116
117     /* USER CODE BEGIN 3 */
118 }
119 /* USER CODE END 3 */
120 }
```

COM4 - PuTTY

```
PRINTF TEST
PRINTF TEST
```

Test de la fonction printf