Rapport de TP3

Gestion des ports d’entrée/sortie

Architecture des ordinateurs

Groupe A – Mercredi 16h30/19h45

10/03/2021

RIGHI Racim

Exercice 1 :

Le but de cet exercice étant de configurer le LED2 en passant par les registres GPIO et RCC, on a besoin de voir les structures de ces registres, et modifier les bits selon nos besoins avec les opérateurs « et » et « ou ». On utilise les décalages de (numéro_pin * 2) pour atteindre les bits correspondants à notre pin (PA5), chaque pin occupant 2 bits.

Dans la fonction main il suffit d'appeler nos fonctions init, led_on et led_off, et ajouter un délai de 1000 ms entre les deux derniers avec la fonction de la librairie HAL : Delay()

```
30 // Numéro du pin du LED2
31 #define LED_PIN 5
32
33
34 /**
35   * @brief  The application entry point.
36   * @retval int
37   */
38 int main(void)
39 {
40
41   /* MCU Configuration--------------------------------------------------------*/
42
43   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
44   HAL_Init();
45
46   /* Configure the system clock */
47   SystemClock_Config();
48
49
50   /* A COMPLETER */
51     //infinite loop
52     Init_Led1();
53     while(1){
54         Led1_On();
55         HAL_Delay(1000);
56         Led1_Off();
57     }
58
59 }
```

Fonction principale 'main', et déclaration de LED_PIN

```c
void Init_Led1(void){
    /* A COMPLETER */
    //activate clock for GPIOA
    RCC->AHB1ENR |= 1;
    //Set GPIOA pin 5 output mode selection
    /*
    Etant donnée la structure des registres MODER, OTYPER, OSPEEDR, PUPDR;
    il faut faire un décalage de NUM_PIN*2 pour arriver aux bits qui lui correspondent.
    Dans notre cas: MODER5[1,0]
    */
    // Décalage de LED_PIN*2 de la valeur 01
    GPIOA->MODER  |=  (0x1 << (LED_PIN*2));
    //Set GPIOA pin 5 type
    GPIOA->OTYPER &= (1 << LED_PIN);
    //Set GPIOA pin 5 speed
    GPIOA->OSPEEDR |= (0x3 << (LED_PIN*2));
    //Set GPIOA pin 5 pull-up/pull-down
    GPIOA->PUPDR |= (0x3 << (LED_PIN*2));
}

void Led1_On(void){
    /* A COMPLETER */
    // Activate LED by setting GPIO_BSRR_BS_5 flag n°5 of BSSR register (GPIOA5-> LD1)
    // Le bit LED_PIN==5 corresponds au bit set du PIN 5 (1 = set, 0 = rien)
    GPIOA->BSRR = (1 << LED_PIN);
}


void Led1_Off(void){
    /* A COMPLETER */
    // Deactivate LED by resetting (GPIO_BSRR_BR_5) flag n°5 of BSSR register (GPIOA5-> LD1)
    // Le bit LED_PIN==21 corresponds au bit reset du PIN 5 (1 = reset, 0 = rien)
    GPIOA->BSRR = (1 << 21);
}

```

Fonctions d'initialisation du PIN5 + manipulation du LED2

Exercice 2 :

Le but de cet exercice étant le même que le premier, mais en utilisant la librairie HAL sans passer manuellement par les registres.

En lisant le fichier « stm32f4xx_hal_gpio.c », on peut trouver la structure GPIO_InitTypeDef qui contient les champs : Pin, Mode, Pull, Speed, Alternate. Chaque élément peut prendre des valeurs définies dans le fichier header « stm32f4xx_hal_gpio.h ». Ainsi que les étapes à savoir pour utiliser la librairie.

```
49                     ##### How to use this driver #####
50    ==============================================================================
51    [..]
52      (#) Enable the GPIO AHB clock using the following function: __HAL_RCC_GPIOx_CLK_ENABLE().
53
54      (#) Configure the GPIO pin(s) using HAL_GPIO_Init().
55          (++) Configure the IO mode using "Mode" member from GPIO_InitTypeDef structure
56          (++) Activate Pull-up, Pull-down resistor using "Pull" member from GPIO_InitTypeDef
57               structure.
58          (++) In case of Output or alternate function mode selection: the speed is
59               configured through "Speed" member from GPIO_InitTypeDef structure.
60          (++) In alternate mode is selection, the alternate function connected to the IO
61               is configured through "Alternate" member from GPIO_InitTypeDef structure.
62          (++) Analog mode is required when a pin is to be used as ADC channel
63               or DAC output.
64          (++) In case of external interrupt/event selection the "Mode" member from
65               GPIO_InitTypeDef structure select the type (interrupt or event) and
66               the corresponding trigger event (rising or falling or both).
67
68      (#) In case of external interrupt/event mode selection, configure NVIC IRQ priority
69          mapped to the EXTI line using HAL_NVIC_SetPriority() and enable it using
70          HAL_NVIC_EnableIRQ().
71
72      (#) To get the level of a pin configured in input mode use HAL_GPIO_ReadPin().
73
74      (#) To set/reset the level of a pin configured in output mode use
75          HAL_GPIO_WritePin()/HAL_GPIO_TogglePin().
```

Manuel d'utilisation de la librairie HAL

```
47  typedef struct
48  {
49    uint32_t Pin;       /*!< Specifies the GPIO pins to be configured.
50                             This parameter can be any value of @ref GPIO_pins_define */
51
52    uint32_t Mode;      /*!< Specifies the operating mode for the selected pins.
53                             This parameter can be a value of @ref GPIO_mode_define */
54
55    uint32_t Pull;      /*!< Specifies the Pull-up or Pull-Down activation for the selected pins.
56                             This parameter can be a value of @ref GPIO_pull_define */
57
58    uint32_t Speed;     /*!< Specifies the speed for the selected pins.
59                             This parameter can be a value of @ref GPIO_speed_define */
60
61    uint32_t Alternate;  /*!< Peripheral to be connected to the selected pins.
62                              This parameter can be a value of @ref GPIO_Alternate_function_selection */
63  }GPIO_InitTypeDef;
```

Déclaration de la structure GPIO_InitTypeDef

```
83  /** @defgroup GPIO_pins_define GPIO pins define
84    * @{
85    */
86  #define GPIO_PIN_0                 ((uint16_t)0x0001)  /* Pin 0 selected    */
87  #define GPIO_PIN_1                 ((uint16_t)0x0002)  /* Pin 1 selected    */
88  #define GPIO_PIN_2                 ((uint16_t)0x0004)  /* Pin 2 selected    */
89  #define GPIO_PIN_3                 ((uint16_t)0x0008)  /* Pin 3 selected    */
90  #define GPIO_PIN_4                 ((uint16_t)0x0010)  /* Pin 4 selected    */
91  #define GPIO_PIN_5                 ((uint16_t)0x0020)  /* Pin 5 selected    */
92  #define GPIO_PIN_6                 ((uint16_t)0x0040)  /* Pin 6 selected    */
93  #define GPIO_PIN_7                 ((uint16_t)0x0080)  /* Pin 7 selected    */
94  #define GPIO_PIN_8                 ((uint16_t)0x0100)  /* Pin 8 selected    */
95  #define GPIO_PIN_9                 ((uint16_t)0x0200)  /* Pin 9 selected    */
96  #define GPIO_PIN_10                ((uint16_t)0x0400)  /* Pin 10 selected   */
97  #define GPIO_PIN_11                ((uint16_t)0x0800)  /* Pin 11 selected   */
98  #define GPIO_PIN_12                ((uint16_t)0x1000)  /* Pin 12 selected   */
99  #define GPIO_PIN_13                ((uint16_t)0x2000)  /* Pin 13 selected   */
100 #define GPIO_PIN_14                ((uint16_t)0x4000)  /* Pin 14 selected   */
101 #define GPIO_PIN_15                ((uint16_t)0x8000)  /* Pin 15 selected   */
102 #define GPIO_PIN_All               ((uint16_t)0xFFFF)  /* All pins selected */
103
```

Valeurs possibles de GPIO_InitTypeDef.Pin

```
#define  GPIO_MODE_INPUT                      0x00000000U   /*!< Input Floating Mode                   */
#define  GPIO_MODE_OUTPUT_PP                  0x00000001U   /*!< Output Push Pull Mode                 */
#define  GPIO_MODE_OUTPUT_OD                  0x00000011U   /*!< Output Open Drain Mode                */
#define  GPIO_MODE_AF_PP                      0x00000002U   /*!< Alternate Function Push Pull Mode     */
#define  GPIO_MODE_AF_OD                      0x00000012U   /*!< Alternate Function Open Drain Mode    */

#define  GPIO_MODE_ANALOG                     0x00000003U   /*!< Analog Mode  */

#define  GPIO_MODE_IT_RISING                  0x10110000U   /*!< External Interrupt Mode with Rising edge trigger dete
#define  GPIO_MODE_IT_FALLING                 0x10210000U   /*!< External Interrupt Mode with Falling edge trigger det
#define  GPIO_MODE_IT_RISING_FALLING          0x10310000U   /*!< External Interrupt Mode with Rising/Falling edge trig

#define  GPIO_MODE_EVT_RISING                 0x10120000U   /*!< External Event Mode with Rising edge trigger detectio
#define  GPIO_MODE_EVT_FALLING                0x10220000U   /*!< External Event Mode with Falling edge trigger detecti
#define  GPIO_MODE_EVT_RISING_FALLING         0x10320000U   /*!< External Event Mode with Rising/Falling edge trigger
/**
```

```
#define  GPIO_NOPULL        0x00000000U   /*!< No Pull-up or Pull-down activation  */
#define  GPIO_PULLUP        0x00000001U   /*!< Pull-up activation                  */
#define  GPIO_PULLDOWN      0x00000002U   /*!< Pull-down activation                */
/**
```

Valeurs possibles de GPIO_InitTypeDef.Pull/Mode

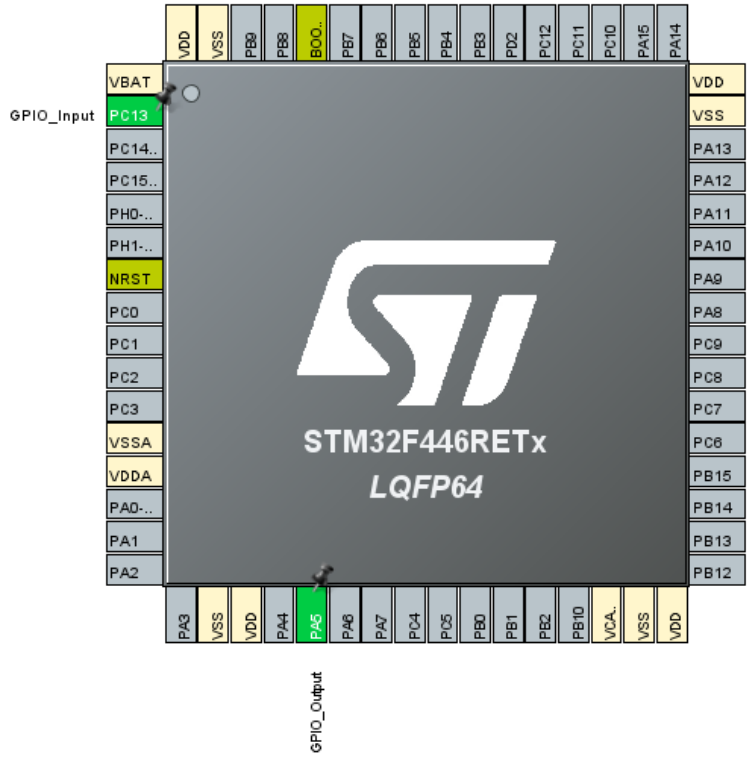
```
#define  GPIO_SPEED_FREQ_LOW         0x00000000U  /*!< IO works at 2 MHz, please refer to the product datasheet */
#define  GPIO_SPEED_FREQ_MEDIUM      0x00000001U  /*!< range 12,5 MHz to 50 MHz, please refer to the product datasheet *
#define  GPIO_SPEED_FREQ_HIGH        0x00000002U  /*!< range 25 MHz to 100 MHz, please refer to the product datasheet  *
#define  GPIO_SPEED_FREQ_VERY_HIGH   0x00000003U  /*!< range 50 MHz to 200 MHz, please refer to the product datasheet  *
/**
```

Valeurs possibles de GPIO_InitTypeDef.Speed

```
  /*A COMPLETER */
  // Déclaration d'une structure GPIO_InitTypeDef pour la configurer et l'envoyer comme paramètre à Init
  GPIO_InitTypeDef GPIO_InitStruct;
  //Activate clock for GPIOA
  __HAL_RCC_GPIOA_CLK_ENABLE();

  // Initialisation GPIOA: Pin 5
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  // Initialisation du type de sortie: Push Pull
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  // Initialisation pull down
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  // Initialisation de la vitesse de commutatation de sortie (freq_very_high = range 50 MHz to 200 MHz)
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;

  // La fonction GPIO_Init prend en paramètre le port GPIOx et la structure GPIO_InitTypeDef
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    //Set default value: RESET

    //Infinite loop
    while(1){
        // Mettre le pin 5 à 1
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
        // Délai de 1000ms = 1s
        HAL_Delay(1000);
        // Reset le pin 5 (à 0)
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
    }
}

```

Déclaration et initialisation des structures + manipulation du LED avec HAL

Exercice 3 :



Configuration graphique des ports PA5 et PC13 en entrée/sortie

```
87  /* Init GPIO A P5
88      Initialisation GPIOA: Pin 5
89      Initialisation du type de sortie: Push Pull
90      Initialisation pull down
91      Initialisation de la vitesse de commutatation de sortie (freq_very_high = range 50 MHz to 200 MHz)
92  */
93  GPIO_InitStructA5.Pin = GPIO_PIN_5;
94  GPIO_InitStructA5.Mode = GPIO_MODE_OUTPUT_PP;
95  GPIO_InitStructA5.Pull = GPIO_PULLDOWN;
96  GPIO_InitStructA5.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
97  HAL_GPIO_Init(GPIOA, &GPIO_InitStructA5);
98
```

Configuration du port PA5 en sortie

```
100 /* Init GPIO C P13
101       Initialisation GPIOC: Pin 13
102       Initialisation du type de sortie: Push Pull
103       Initialisation pull down
104       Initialisation de la vitesse de commutatation de sortie (freq_very_high = range 50 MHz to 200 MHz)
105 */
106   GPIO_InitStructC13.Pin = GPIO_PIN_13;
107   GPIO_InitStructC13.Mode = GPIO_MODE_INPUT;
108   GPIO_InitStructC13.Pull = GPIO_PULLUP;
109   GPIO_InitStructC13.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
110   HAL_GPIO_Init(GPIOC, &GPIO_InitStructC13);
111
```

Configuration du port PC13 en entée

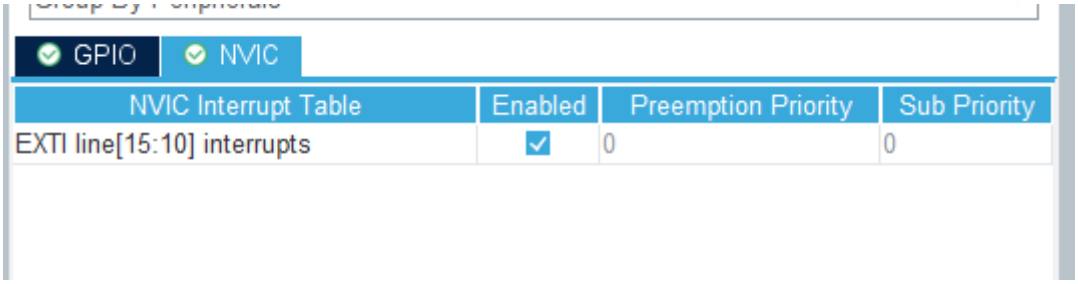
```
89    /* Infinite loop */
90    while (1)
91    {
92      if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) ==  GPIO_PIN_SET)
93      {
94          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
95  //      HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
96      }
97      else
98      {
99          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
100     }
101     HAL_Delay(500);
102   }
```

Lecture de l'état de PC13 et allumage/extinction du LED selon l'état

Exercice 4 :

Pin Name	Signal on Pin	GPIO outpu...	GPIO mode	GPIO Pull-u...	Maximum o...	User Label	Modified
PC13	n/a	n/a	External Int...	No pull-up a...	n/a		☐
PA5	n/a	Low	Output Pus...	Pull-down	High		☑

Configuration de PC13 et PA5 dans CubeMX



Activation de la source d'interruption dans NVIC

```c
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

  /*Configure GPIO pin : PC13 */
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pin : PA5 */
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

}
```

```
154 {
155   GPIO_InitTypeDef GPIO_InitStruct = {0};
156
157   /* GPIO Ports Clock Enable */
158   __HAL_RCC_GPIOC_CLK_ENABLE();
159   __HAL_RCC_GPIOA_CLK_ENABLE();
160
161   /*Configure GPIO pin : PC13 */
162   GPIO_InitStruct.Pin = GPIO_PIN_13;
163   GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
164   GPIO_InitStruct.Pull = GPIO_NOPULL;
165   HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
166
167   /*Configure GPIO pin : PA5 */
168   GPIO_InitStruct.Pin = GPIO_PIN_5;
169   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
170   GPIO_InitStruct.Pull = GPIO_PULLDOWN;
171   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
172
173   /* EXTI interrupt init*/
174   HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
175   HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
176
177 }
178
```

Code généré par l'IDE

Après avoir généré le code, on doit modifier la fonction qui est appelée à chaque interruption, on la trouve dans le fichier « stm32f4xx_it.c » nommée « EXTI15_10_IRQHandler(void) »

Il suffit d'y modifier l'état de PA5 pour allumer/éteindre le LED2, en utilisant la fonction « HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5) »

➔ **Ouvrir le fichier stm32f4xx_it.c**

Q : Que contient-il ? A quoi il correspond ?

Ce fichier contient les routines du service d'interruption

C'est l'ensemble des fonctions appelées suite à certains types d'interruptions, y compris celles initialisées par l'utilisateur, dans notre cas la fonction « EXTI15_10_IRQHandler(void) »

Une autre manière de faire cet exercice est d'utiliser la fonction HAL_GPIO_EXTI_CALLBACK, on l'implémentant dans notre main de la manière suivante :

```
57 /* USER CODE BEGIN 0 */
58 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
59 {
60     HAL_GPIO_TogglePin(PORT, PIN);
61 }
62 /* USER CODE END 0 */
```

Appel à la fonction callback d'interruption

Ce qui nous permet d'utiliser les variables locales à notre fonction main.c

```
34 /* USER CODE BEGIN PD */
35 #define PORT GPIOA
36 #define PIN GPIO_PIN_5
37 /* USER CODE END PD */
```

Constantes définies dans main.c

Exercice 5 :

Q : -Par quel facteur peut-on diviser la fréquence d'horloge à l'aide d'un prescaler sur 16 bits ?

De [1 à 2^16-1=65535].

Q : Quelle est l'horloge source arrivant en entrée du timer 1 ?

C'est l'horloge système.

Dans notre cas l'horloge système est à 84 mhz, ce qui nous permet avec un timer 16 bits d'arriver jusqu'à 65535 / 84 microsecondes = 780 microsedondes

Donc on aura besoin d'un prescaler de 8400 (84 / 8400 = 0.01mhz = 10 = 1 cycle / 0.0001 seconde) = 1000 cycles / seconde

Donc 10000 cycles pour atteindre exactement 1 seconde.

D'où :

- Prescaler (PSC – 16 bits value) = 8400 - 1
- Counter Period (AutoReload Register) = 10000 – 1 (car le compteur commence à 0)

Q : Que réaliser les fonctions MX_GPIO_Init() et MX_TIM1_Init ()?

Ces fonctions initialisent les ports GPIO et le timer1 qu'on a configuré depuis l'interface graphique STM32CubeMX.

Q : A quoi correspond cette fonction : void TIM1_UP_TIM10_IRQHandler(void) ?

Cette fonction gère les interruptions de mise à jour du timer1 et les interruptions globales du timer10

Après consultation des fichiers de la librairie HAL, on trouve qu'on doit utiliser la fonction de HAL « HAL_TIM_Base_Start_IT » pour les interruptions et implémenter « HAL_TIM_PeriodElapsedCallback » pour le code à exécuter lors des interruptions

Donc on implémente la fonction de callback, et y vérifier si c'est bien le timer1 qui a lancé l'interruption, à la fin du temps imparti, le led 2 sera activé.

```
227 void HAL_TIM_PeriodElapsedCallback(TIM_HandlerTypeDef *htim)
228 {
229     // Vérifier que c'est le bon timer qui a lancé l'interruption
230     if(htim == &htim1)
231     {
232         HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
233     }
234 }
```

Fonction de callback d'interruption du timer

Q : Bilan : Selon vous, quel est l'intérêt des interruptions par rapport la scrutation ?

L'objectif principal et de pouvoir réduire la charge sur le CPU, en effet, sans interruptions on serait obligé d'utiliser des boucles actives et la fonction « HAL_Delay() », qui utilisent constamment le CPU.

Exercice 6 :

Pour modifier la vitesse de clignotement à chaque appui de bouton, on pourrait modifier les valeurs du timer et le réinitialiser à chaque interruption, étant donné que les interruptions sont configurés en mode update, donc la nouvelle valeur du timer sera effective au prochain cycle.

Pour modifier les valeurs du timer on utilise la variable de type « TIM_HandleTypeDef htim1 », et on modifie les prescaler de la manière suivante : TIM3->RCC