Rapport d’exercice n°1
Led et Bouton

Axel Jacquot

M1 SER

ynov
AÉRONAUTIQUE &
SYSTÈMES EMBARQUÉS

Table des matières

Exercice n°1

Enoncé :

Lors de l'appui du bouton User se trouvant sur la carte STM32 nous devrons allumer les leds une à une tout en arrêtant la led qui on était allumé précédemment. Nous devrons allumer les Leds dans un ordre précis Vert, Rouge, Orange puis Bleu.

Explication des fonctions fournis pour la STM32 (HAL) :

__HAL_RCC_GPIO(AouD)_CLK_ENABLE.:

Cette ligne de code permet d'activer les ports présents sur la carte STM32.

```
#define __HAL_RCC_GPIOD_CLK_ENABLE()   do { \
                                         __IO uint32_t tmpreg = 0x00U; \
                                         SET_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIODEN);\
                                         /* Delay after an RCC peripheral clock enabling */ \
                                         tmpreg = READ_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIODEN);\
                                         UNUSED(tmpreg); \
                                       } while(0U)
```

6.3.10 RCC AHB1 peripheral clock register (RCC_AHB1ENR)

Address offset: 0x30

Reset value: 0x0010 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reser-ved	OTGHS ULPIEN	OTGHSEN	ETHMACPTPEN	ETHMACRXEN	ETHMACTXEN	ETHMACEN	Res.	**DMA2DEN**	DMA2EN	DMA1EN	CCMDATARAMEN	Res.	BKPSRAMEN	Reserved	
	rw	rw	rw	rw	rw	rw		rw	rw	rw			rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CRCEN	Res.	GPIOKEN	GPIOJEN	GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
			rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Ce registre permet d'activer les ports A et D dans notre cas en mettant GPIODEN et GPIOAEN à 1.

GPIO_InitTypeDef :

```
typedef struct
{
  uint32_t Pin;
  /*!< Specifies the GPIO pins to be configured.
    This parameter can be any value of @ref GPIO_pins_define */
```

```
  uint32_t Mode;
  /*!< Specifies the operating mode for the selected pins.
    This parameter can be a value of @ref GPIO_mode_define */

  uint32_t Pull;
  /*!< Specifies the Pull-up or Pull-Down activation for the selected pins.
    This parameter can be a value of @ref GPIO_pull_define */

  uint32_t Speed;
  /*!< Specifies the speed for the selected pins.
    This parameter can be a value of @ref GPIO_speed_define */

  uint32_t Alternate;
  /*!< Peripheral to be connected to the selected pins.
    This parameter can be a value of @ref GPIO_Alternate_function_selection */
}GPIO_InitTypeDef;
```

Permet de régler les pins, le mode voulu des pins (entrée, sortie, interruption …), la vitesse, si les pins sont en pull-down ou en pull-up. Dans notre cas nous initialiserons les pins correspondant au led en sortie et le pin correspondant au bouton en entrée pour le 1er exercice et interruption pour le 2nd.

Les différents pins pour le bouton et les leds ci-dessous :

6.3 LEDs

- LD1 COM: LD1 default status is red. LD1 turns to green to indicate that communications are in progress between the PC and the ST-LINK/V2.
- LD2 PWR: red LED indicates that the board is powered.
- User LD3: orange LED is a user LED connected to the I/O PD13 of the STM32F407VGT6.
- User LD4: green LED is a user LED connected to the I/O PD12 of the STM32F407VGT6.
- User LD5: red LED is a user LED connected to the I/O PD14 of the STM32F407VGT6.
- User LD6: blue LED is a user LED connected to the I/O PD15 of the STM32F407VGT6.
- USB LD7: green LED indicates when VBUS is present on CN5 and is connected to PA9 of the STM32F407VGT6.
- USB LD8: red LED indicates an overcurrent from VBUS of CN5 and is connected to the I/O PD5 of the STM32F407VGT6.

6.4 Push buttons

- B1 USER: User and Wake-Up buttons are connected to the I/O PA0 of the STM32F407VG.
- B2 RESET: Push button connected to NRST is used to RESET the STM32F407VG.

Pour les leds il s'agit des pins 12(vert),13(orange),14(rouge),15(bleu) du port D.

Pour le bouton il s'agit de la pin 0 du port A.

HAL_GPIO_Init() :

Cette ligne permet d'initialiser les pins que nous avons régler avant sur le port que nous voulons. En paramètre de la fonction nous devons mettre le port et ensuite la configuration des différents pins.

HAL_GPIO_WritePin() :

Change l'état des pins se trouvant sur le port qui lui sont mis en paramètre ainsi que l'état dans lequel nous voulons configurer le ou les pins.

```
void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
{
  //GPIOx prend la valeur d'un Port soit Port A ou D dans notre cas
  //GPIO_PIN correspond à ou aux pin avec lesquels nous voulons changer leurs états
  //PinState est l'état dans lequel nous voulons mettre le pin
  /* Check the parameters */
  assert_param(IS_GPIO_PIN(GPIO_Pin));
  assert_param(IS_GPIO_PIN_ACTION(PinState));

  if(PinState != GPIO_PIN_RESET)  //Vérifie si le pin doit être mis a 1
  {
    GPIOx->BSRR = GPIO_Pin;    //Le pin est mis à 1. BSRR permet d'agir sur l'état du pin
  }
  else
  {
    GPIOx->BSRR = (uint32_t)GPIO_Pin << 16U;  //Le pin est mis à 0
  }
}
```

8.4.7 GPIO port bit set/reset register (GPIOx_BSRR) (x = A..I/J/K)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BRy:** Port x reset bit y (y = 0..15)
These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.
0: No action on the corresponding ODRx bit
1: Resets the corresponding ODRx bit
Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BSy:** Port x set bit y (y= 0..15)
These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.
0: No action on the corresponding ODRx bit
1: Sets the corresponding ODRx bit

HAL_GPIO_ReadPin() :

Permet de lire l'état d'un pin se trouvant sur un port, nous devons mettre ceux-ci en paramètre de la fonction.

```
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
  //GPIOx prend la valeur d'un Port soit Port A ou D dans notre cas
  //GPIO_PIN correspond a ou aux pin avec lesquels nous voulons changer leurs états
  GPIO_PinState bitstatus;    //Déclaration d'un varibale qui récuperera l'état de lapin

  /* Check the parameters */
  assert_param(IS_GPIO_PIN(GPIO_Pin));

  if((GPIOx->IDR & GPIO_Pin) != (uint32_t)GPIO_PIN_RESET)
  //Vérifie si le pin n'est pas 0. IDR permet de lire les états des pins du port choisit
  {
    bitstatus = GPIO_PIN_SET;         //variable de récupération mis à 1
  }
  else
  {
    bitstatus = GPIO_PIN_RESET;       //variable de récupération mis à 0
  }
  return bitstatus;  //Sortie de la fonction avec en retour la variable de récupération
}
```

8.4.5 GPIO port input data register (GPIOx_IDR) (x = A..I/J/K)

Address offset: 0x10

Reset value: 0x0000 XXXX (where X means undefined)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

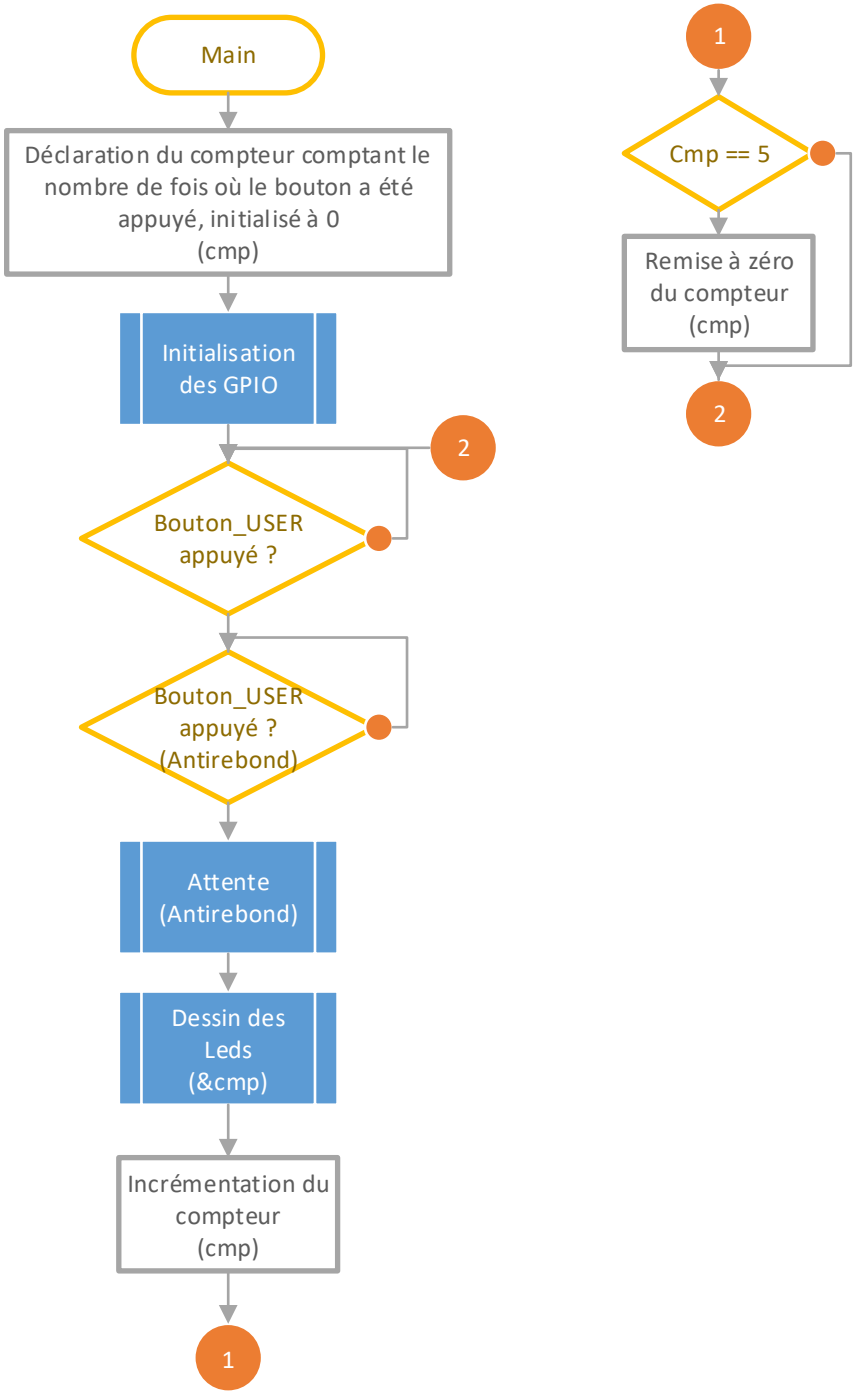
Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDRy**: Port input data (y = 0..15)

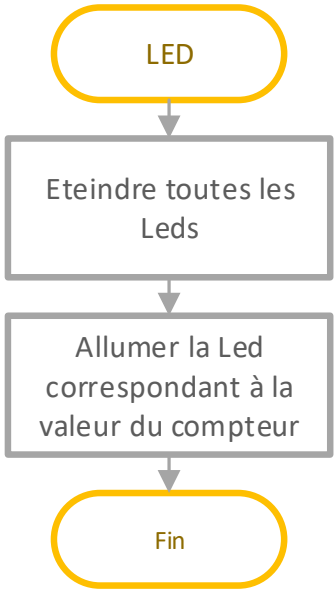
These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.

Algorigramme :

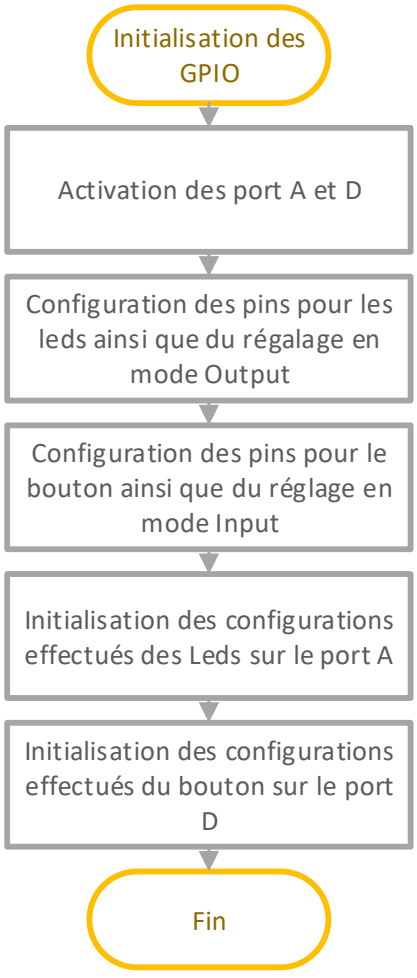
Programme Principal :



Programme des Leds :



Configuration des ports:



Exercice n°2

Enoncé :

Lors de l'appui du bouton User se trouvant sur la carte STM32 nous devrons déclencher une interruption qui aura pour but d'allumer les leds une à une tout en arrêtant la led qui on était allumé précédemment. Nous devrons allumer les Leds dans un ordre précis Rouge, Orange, Vert puis Bleu.

Explication du code fourni pour la STM32 :

HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin):

Cette fonction est l'emplacement ou nous allons mettre le code que nous souhaitons exécuter lors de l'interruption.

EXTI0_IRQHandler :

Fonction contenant la fonction permettant de vérifier si l'interruption doit être activé. EXTIO correspond à la première interruption GPIO qui est disponible.

HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0):

Fonction qui permet de lancer le code se trouvant dans l'interruption si celle-ci est demandé.

Le pin 0 est en paramètre de cette fonction pour spécifier à celle-ci sur quel pin du port réglé en interruption elle doit vérifier son état.

```
void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
{
   //GPIO_PIN est le pin que nous souhaitons tester sur un port défini à l'avance dans notre cas le port A
  /* EXTI line interrupt detected */
  if(__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != RESET) //Verifie si un pin du port A est à 1 pour pouvoir déckencher une interruption si c'est le cas
  {
    __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);        //Nettoie l'interruption
    HAL_GPIO_EXTI_Callback(GPIO_Pin);          //Fonction permettant de lancer le code se trouvant dans l'interruption
  }
}
```

HAL_NVIC_SetPriority(EXTI0_IRQn, 5, 5):

```
void HAL_NVIC_SetPriority(IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)
//Dans STM32f4xx_hal_cortex.c

{
   //IRQn_Type IRQn Permet de choisir le type de l'interruption (GPIO,UART,I2C,SPI)
```

```
  //PreemptPriority Permet de déterminer quand une interruption peut préempter une autre

  //SubPriority permet de déterminer laquelle des interruptions de même priorité de préemption en attente s'exécutera en premier
  uint32_t prioritygroup = 0x00U;

  /* Check the parameters */
  assert_param(IS_NVIC_SUB_PRIORITY(SubPriority));
  assert_param(IS_NVIC_PREEMPTION_PRIORITY(PreemptPriority));

  prioritygroup = NVIC_GetPriorityGrouping();

  NVIC_SetPriority(IRQn, NVIC_EncodePriority(prioritygroup, PreemptPriority, SubPriority));
  //NVIC_SetPriority permey de régler la priorité sur le type souhaité
}
```

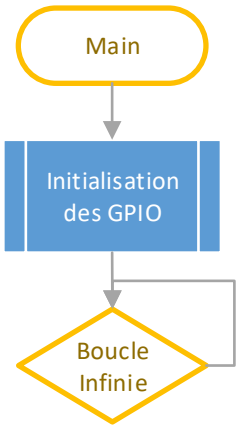
HAL_NVIC_EnableIRQ(EXTI0_IRQn):

Active l'interruption.

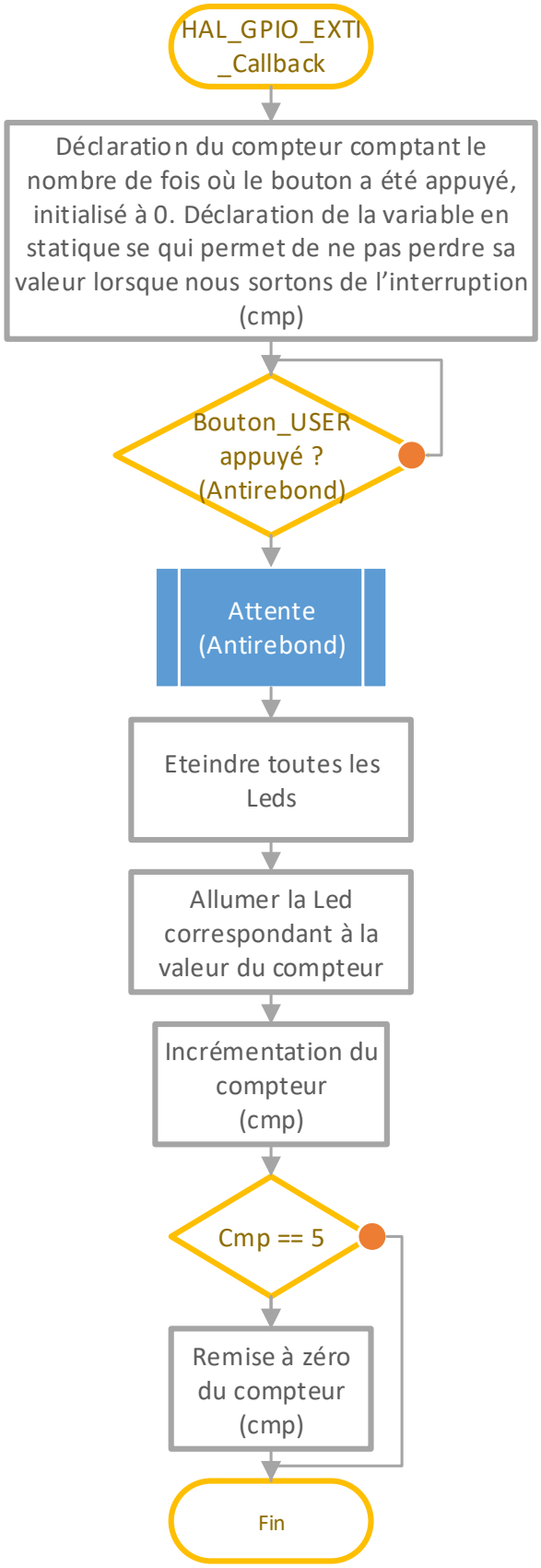
```
void HAL_NVIC_EnableIRQ(IRQn_Type IRQn)
{
  /* Check the parameters */
  assert_param(IS_NVIC_DEVICE_IRQ(IRQn));

  /* Enable interrupt */
  NVIC_EnableIRQ(IRQn);
}
```

Algorigramme:

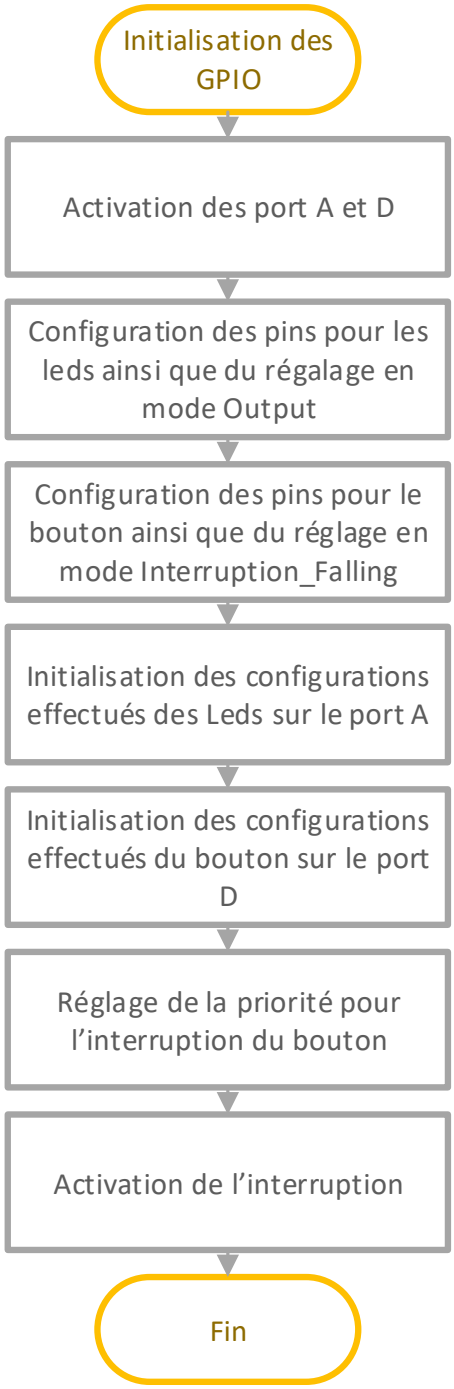
Programme Principal:



Interruption Led:



Configuration des ports:



Conclusion

Ces exercices nous on permit de plus approfondir les conséquences que nous avons acquises pendant les cours d'ARM et aussi de connaitre la déclaration des interruptions ainsi que leurs fonctions sur une carte STM32.