ÉCOLE SUPÉRIEURE DES TECHNOLOGIES
ÉLECTRONIQUE, INFORMATIQUE, INFOGRAPHIE

ARCHITECTURE DES SYSTÈMES

# TP1 - STM32F411
# Utilisation des entrées / sorties et des interruptions

*Auteur :*
Thomas LEPOIX

MASTER 2 Systèmes Embarqués

E.S.T.E.I.
École Supérieure des Technologies Électronique, Informatique, et Infographie
Département Systèmes Embarqués

27 novembre 2018

# Table des matières

# Chapitre 1

# Introduction

## 1.1 Cahier des charges

Le but de ce TP est de manipuler les entrées / sorties (GPIO) et les interruptions matérielles sur un microcontrôleur ARM Cortex-M4.

Pour cela deux programmes sont à réaliser :

- Le premier, en mode polling, devra à chaque appui sur le bouton utilisateur allumer les LEDs dans l'ordre suivant : Vert, Rouge, Orange, Bleu.
- Le second, utilisant les interruptions, devra à chaque relâchement du bouton utilisateur allumer les LEDs dans l'ordre suivant : Rouge, Orange, Vert, Bleu.

L'environnement utilisé lors du TP est le suivant :

- Système d'exploitation : Ubuntu Mate 18.04.1 LTS 64-bit
- Carte de développement : STM32F411E-Discovery
- IDE : Eclipse System Workbench for STM32
- Librairie CMSIS

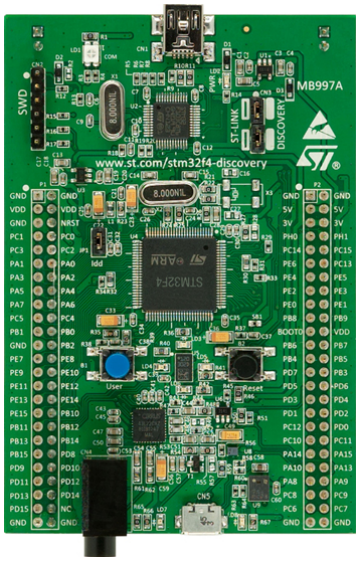


FIGURE 1.1 – Carte de développement STM32F4-Discovery

## 1.2 Librairie CMSIS

La librairie CMSIS (Cortex Microcontroller Software Interface Standard, c'est-à-dire Interface Logicielle Standard pour Microcontrôleurs Cortex) est une couche d'abstraction matérielle. Son utilisation permet de s'affranchir des adresses mémoire des registres du microcontrôleur et de fait faciliter le travail de programmation tout en augmentant la lisibilité du code.

De plus cette librairie est valable pour toute la famille de microcontrôleurs ARM Cortex-M, à la différence de la HAL (Hardware Abstraction Layer, autre couche d'abstraction matérielle de plus haut niveau que CMSIS) proposée par STMicroelectronics, qui elle n'est valable que pour les microcontrôleurs STM32. CMSIS demeure une librairie de bas niveau.

Concrètement, cette librairie propose des structures couvrant les registres mémoire et de nombreuses macro-définitions de masques et valeurs que l'on peut affecter aux registres.

Par exemple, si l'on souhaite configurer la sortie 12 du portD en mode drain ouvert, il faut mettre à `1` le douzième bit du registre `GPIOD_OTYPER`.

Pour cela on se réfère d'abord à la table d'organisation mémoire pour déterminer l'adresse du portD : `0x40020C00`

**Table 1. STM32F4xx register boundary addresses**

| Boundary address | Peripheral | Bus | Register map |
|---|---|---|---|
| 0x4002 1800 - 0x4002 1BFF | GPIOG | AHB1 | *Section 8.4.11: GPIO register map on page 287* |
| 0x4002 1400 - 0x4002 17FF | GPIOF | | |
| 0x4002 1000 - 0x4002 13FF | GPIOE | | |
| 0x4002 0C00 - 0x4002 0FFF | GPIOD | | |
| 0x4002 0800 - 0x4002 0BFF | GPIOC | | |
| 0x4002 0400 - 0x4002 07FF | GPIOB | | |
| 0x4002 0000 - 0x4002 03FF | GPIOA | | |

FIGURE 1.2 – Extrait de la table d'organisation mémoire

Puis on se réfère à la table d'organisation des registres des ports d'entrée / sortie pour déterminer le décalage du registre `OTYPER` par rapport à l'adresse du portD : `0x04`.

### 8.4.11 GPIO register map

The following table gives the GPIO register map and the reset values.

**Table 39. GPIO register map and reset values**

| Offset | Register | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | **GPIOx_MODER** (where x = C..I/J/K) | MODER15[1:0] | | MODER14[1:0] | | MODER13[1:0] | | MODER12[1:0] | | MODER11[1:0] | | MODER10[1:0] | | MODER9[1:0] | | MODER8[1:0] | | MODER7[1:0] | | MODER6[1:0] | | MODER5[1:0] | | MODER4[1:0] | | MODER3[1:0] | | MODER2[1:0] | | MODER1[1:0] | | MODER0[1:0] | |
| | Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x04 | **GPIOx_ OTYPER** (where x = A..I/J/K) | Reserved | | | | | | | | | | | | | | | | OT15 | OT14 | OT13 | OT12 | OT11 | OT10 | OT9 | OT8 | OT7 | OT6 | OT5 | OT4 | OT3 | OT2 | OT1 | OT0 |
| | Reset value | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x08 | **GPIOx_ OSPEEDR** (where x = A..I/J/K except B) | OSPEEDR15[1:0] | | OSPEEDR14[1:0] | | OSPEEDR13[1:0] | | OSPEEDR12[1:0] | | OSPEEDR11[1:0] | | OSPEEDR10[1:0] | | OSPEEDR9[1:0] | | OSPEEDR8[1:0] | | OSPEEDR7[1:0] | | OSPEEDR6[1:0] | | OSPEEDR5[1:0] | | OSPEEDR4[1:0] | | OSPEEDR3[1:0] | | OSPEEDR2[1:0] | | OSPEEDR1[1:0] | | OSPEEDR0[1:0] | |
| | Reset value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

FIGURE 1.3 – Extrait de la table d'organisation des registres des ports d'entrée / sortie

On peut alors écrire les lignes de code suivantes :

```
unsigned int volatile* portD=(unsigned int volatile*)0x40020C00;
*(portD+0x4/4) |= (0x1 << 12);
```

Remarque : Le décalage mémoire du registre est divisé par 4 car il est exprimé en octets (8 bits) dans la documentation tandis que le pointeur utilisé résonne sur des variables `int` de 32 bits.

Pour simplifier la procédure, la librairie CMSIS propose dans le fichier `CMSIS/device/stm32f411xe.h` la structure et les macro-définitions suivantes (plusieurs versions de ce fichier existent, selon le modèle de la carte de développement) :

```
typedef struct
{
  __IO uint32_t MODER;    //!< GPIO port mode register,               Address offset: 0x00
  __IO uint32_t OTYPER;   //!< GPIO port output type register,        Address offset: 0x04
  __IO uint32_t OSPEEDR;  //!< GPIO port output speed register,       Address offset: 0x08
  __IO uint32_t PUPDR;    //!< GPIO port pull-up/pull-down register,  Address offset: 0x0C
  __IO uint32_t IDR;      //!< GPIO port input data register,         Address offset: 0x10
  __IO uint32_t ODR;      //!< GPIO port output data register,        Address offset: 0x14
  __IO uint32_t BSRR;     //!< GPIO port bit set/reset register,      Address offset: 0x18
  __IO uint32_t LCKR;     //!< GPIO port configuration lock register, Address offset: 0x1C
  __IO uint32_t AFR[2];   //!< GPIO alternate function registers,     Address offset: 0x20
} GPIO_TypeDef;           //                                                          -0x24


#define PERIPH_BASE        0x40000000U     //!< Peripheral base address in the alias region
#define AHB1PERIPH_BASE    (PERIPH_BASE + 0x00020000U)
#define GPIOD_BASE         (AHB1PERIPH_BASE + 0x0C00U)
#define GPIOD              ((GPIO_TypeDef *) GPIOD_BASE)


```

```
#define GPIO_OTYPER_OT12_Pos          (12U)
#define GPIO_OTYPER_OT12_Msk          (0x1U << GPIO_OTYPER_OT12_Pos)        //!< 0x00001000
#define GPIO_OTYPER_OT12              GPIO_OTYPER_OT12_Msk
```

Le code à écrire est alors celui-ci :

```
GPIOD->OTYPER |= GPIO_OTYPER_OT12;
```

# Chapitre 2

# Étude de registres

## 2.1 Configuration du bouton utilisateur

### 2.1.1 Analyse du schéma de la carte

Avant toute configuration logicielle, il est nécessaire de savoir à quelle interface du microcontrôleur est connecté le bouton utilisateur. Pour cela on observe le schéma de la carte de développement.

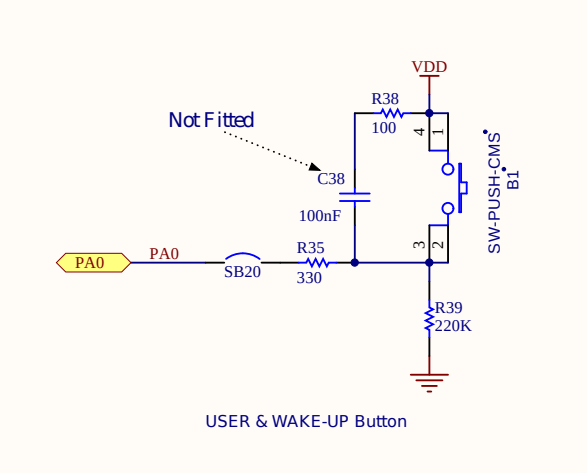


FIGURE 2.1 – Schéma de raccordement du bouton utilisateur

Le bouton se situe sur l'interface `0` du portA des entrées / sorties.

### 2.1.2 Activation de l'horloge pour le portA

Pour qu'un élément du microcontrôleur fonctionne, il faut que son horloge soit activée. Pour des raisons de consommation énergétique, la plupart des horloges sont désactivées par défaut.

#### 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 |

Bit 0 **GPIOAEN:** IO port A clock enable
This bit is set and cleared by software.
0: IO port A clock disabled
1: IO port A clock enabled

FIGURE 2.2 – Registre RCC_AHB1ENR

Le portA étant sur le bus **`AHB1`**, l'activation de son horloge se fait en mettant à **`1`** le bit **`0`** du registre **`RCC_AHB1ENR`** (RCC signifie Reset and Clock Control) :

```
1 RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
```

### 2.1.3 Mise en entrée de l'interface 0 du portA

La direction des interfaces d'un port est contrôlée par le registre **`GPIOx_MODER`**.

#### 8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MODER15[1:0] | | MODER14[1:0] | | MODER13[1:0] | | MODER12[1:0] | | MODER11[1:0] | | MODER10[1:0] | | MODER9[1:0] | | MODER8[1:0] | |
| rw | rw | rw | rw | rw | rw | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MODER7[1:0] | | MODER6[1:0] | | MODER5[1:0] | | MODER4[1:0] | | MODER3[1:0] | | MODER2[1:0] | | MODER1[1:0] | | MODER0[1:0] | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

FIGURE 2.3 – Registre GPIOx_MODER

On met donc à **`0`** les deux premiers bits du registre **`GPIOA_MODER`** :

```
GPIOA->MODER &= ~(GPIO_MODER_MODER0);
```

### 2.1.4 Définition de la logique du bouton

La logique du bouton est déterminée par la façon dont il est connecté à l'entrée du micro-contrôleur. Deux options sont possibles : par une résistance de tirage (pull-up) ou par une résistance de rappel (pull-down).

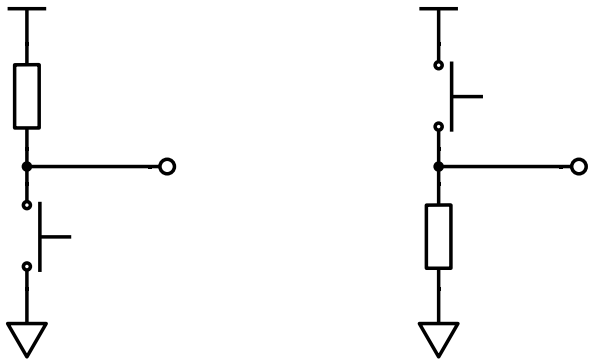

FIGURE 2.4 – Configuration pull-up à droite et pull-down à gauche

Sans contrainte particulière, le choix est arbitraire. On choisit le mode pull-down c'est-à-dire que l'entrée sera à `0` au repos du bouton et à `1` à l'appui sur celui-ci.

Pour configurer cela, il faut appliquer la valeur `10` aux deux premiers bits du registre `GPIOA_PUPDR`.

### 8.4.4 GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A..I/J/K)

Address offset: 0x0C

Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 0x0000 0000 for other ports

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PUPDR15[1:0] | | PUPDR14[1:0] | | PUPDR13[1:0] | | PUPDR12[1:0] | | PUPDR11[1:0] | | PUPDR10[1:0] | | PUPDR9[1:0] | | PUPDR8[1:0] | |
| rw | rw | rw | rw | rw | rw | 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 |
| PUPDR7[1:0] | | PUPDR6[1:0] | | PUPDR5[1:0] | | PUPDR4[1:0] | | PUPDR3[1:0] | | PUPDR2[1:0] | | PUPDR1[1:0] | | PUPDR0[1:0] | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 2y:2y+1 **PUPDRy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O pull-up or pull-down
00: No pull-up, pull-down
01: Pull-up
10: Pull-down
11: Reserved

FIGURE 2.5 – Registre GPIOx_PUPDR

```
GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR0);
GPIOA->PUPDR |= GPIO_PUPDR_PUPDR0_1;
```

## 2.2 Configuration des quatre LEDs

### 2.2.1 Analyse du schéma de la carte

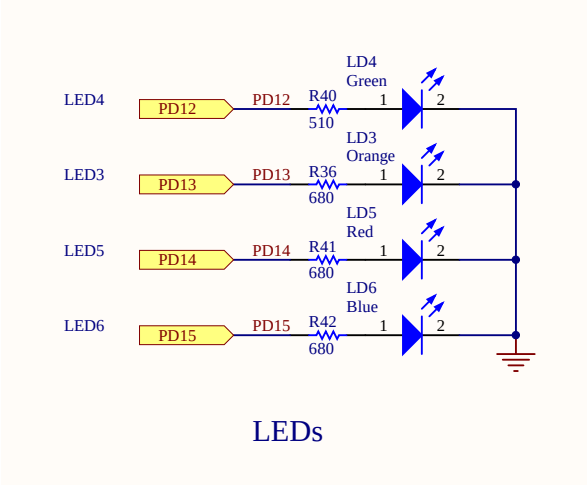


FIGURE 2.6 – Schéma de raccordement des LEDs

Les LEDs verte, orange, rouge et bleue se situent respectivement sur les entrées / sorties `12`, `13`, `14` et `15` du portD.

### 2.2.2 Activation de l'horloge pour le portD

De même que pour le portA, il est nécessaire d'activer l'horloge sur le portD en mettant à `1` le quatrième bit du registre **`RCC_AHB1ENR`**.

#### 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 |

Bit 0 **GPIOAEN:** IO port A clock enable
This bit is set and cleared by software.
0: IO port A clock disabled
1: IO port A clock enabled

FIGURE 2.7 – Registre RCC_AHB1ENR

```
RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN;
```

### 2.2.3 Mise en sortie des interfaces 12, 13, 14 et 15 du portD

Pour configurer les quatre interfaces en sortie, on affecte la valeur `01` aux quatre couples de bits concernés.

#### 8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MODER15[1:0] | | MODER14[1:0] | | MODER13[1:0] | | MODER12[1:0] | | MODER11[1:0] | | MODER10[1:0] | | MODER9[1:0] | | MODER8[1:0] | |
| rw | rw | rw | rw | rw | rw | 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 |
| MODER7[1:0] | | MODER6[1:0] | | MODER5[1:0] | | MODER4[1:0] | | MODER3[1:0] | | MODER2[1:0] | | MODER1[1:0] | | MODER0[1:0] | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

FIGURE 2.8 – Registre GPIOx_MODER

```
GPIOD->MODER &= ~(GPIO_MODER_MODER12
                 |GPIO_MODER_MODER13
                 |GPIO_MODER_MODER14
                 |GPIO_MODER_MODER15);
GPIOD->MODER |= (GPIO_MODER_MODER12_0
                |GPIO_MODER_MODER13_0
                |GPIO_MODER_MODER14_0
                |GPIO_MODER_MODER15_0);
```

### 2.2.4 Définition du type de sortie des LEDs

Comme pour une entrée, il faut définir la façon dont est raccordée une sortie à l'intérieur du microcontrôleur. Nous avons le choix entre deux options : la configuration push-pull et la configuration drain-ouvert (open-drain), version CMOS de la configuration collecteur-ouvert.

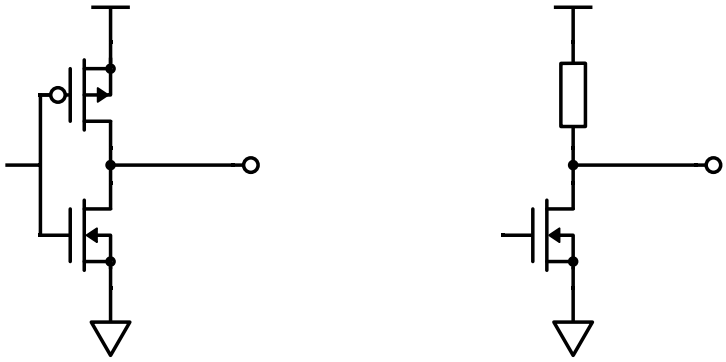

FIGURE 2.9 – Configuration push-pull à droite et open-drain à gauche

Sans contrainte particulière, ce choix est également arbitraire. On préfère généralement le mode push-pull. Pour le choirir, on affecte la valeur `0` aux bits 12, 13, 14 et 15 du registre **`GPIOx_OTYPER`**.

### 8.4.2 GPIO port output type register (GPIOx_OTYPER) (x = A..I/J/K)

Address offset: 0x04

Reset value: 0x0000 0000

| 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 |
| OT15 | OT14 | OT13 | OT12 | OT11 | OT10 | OT9 | OT8 | OT7 | OT6 | OT5 | OT4 | OT3 | OT2 | OT1 | OT0 |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

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

Bits 15:0 **OTy**: Port x configuration bits (y = 0..15)
These bits are written by software to configure the output type of the I/O port.
0: Output push-pull (reset state)
1: Output open-drain

FIGURE 2.10 – Registre GPIOx_OTYPER

```
GPIOD->OTYPER &= ~(GPIO_OTYPER_OT12
                  |GPIO_OTYPER_OT13
                  |GPIO_OTYPER_OT14
                  |GPIO_OTYPER_OT15);
```

## 2.3 Configuration de l'interruption

Cette section est à ignorer dans le cas du programme en mode polling.

### 2.3.1 Activation de l'horloge pour le contrôleur de configuration système

De la même façon que pour les entrées / sorties, pour que le contrôleur de configuration système fonctionne, il est nécessaire d'activer son horloge. Celui-ci étant sur le bus **`APB2`**, on active son horloge en mettant à `1` le bit `14` du registre **`RCC_APB2ENR`**.

### 7.3.14 RCC APB2 peripheral clock enable register (RCC_APB2ENR)

Address offset: 0x44

Reset value: 0x0000 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | TIM11 EN | TIM10 EN | TIM9 EN |
| | | | | | | | | | | | | | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reser-ved | SYSCFG EN | Reser-ved | SPI1 EN | SDIO EN | ADC3 EN | ADC2 EN | ADC1 EN | Reserved | | USART 6 EN | USART 1 EN | Reserved | | TIM8 EN | TIM1 EN |
| | rw | | rw | rw | rw | rw | rw | | | rw | rw | | | rw | rw |

Bit 14 **SYSCFGEN:** System configuration controller clock enable
Set and cleared by software.
0: System configuration controller clock disabled
1: System configuration controller clock enabled

FIGURE 2.11 – Registre RCC_APB2ENR

```
RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
```

## 2.3.2 Configuration du multiplexeur d'interruptions matérielles

Pour déclencher une interruption, les entrées sont multiplexées en 15 lignes nommées **EXTIx**. Il n'est ainsi pas possible de configurer simultanément deux interruptions sur deux entrées faisant partie de la même ligne multiplexée. Par exemple il est possible de configurer simultanément une interruption sur **PA0** et sur **PB1** mais pas sur **PA0** et **PB0**. Il existe d'autres lignes d'interruption matérielle associées à d'autres fonctions du microcontrôleur que les entrées / sorties standard.

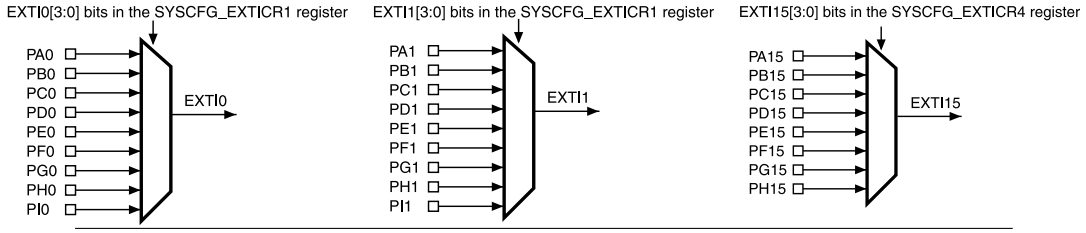


FIGURE 2.12 – Multiplexage des lignes d'entrée d'interruption

Le bouton utilisateur étant sur **PA0**, on cherche a associer la ligne **EXTI0** à cette entrée. Cette configuration se fait en affectant la valeur **0000** au premier groupe de 4 bits du registre **SYSCFG_EXTICR1**.

### 9.3.3 SYSCFG external interrupt configuration register 1 (SYSCFG_EXTICR1)

Address offset: 0x08

Reset value: 0x0000 0000

| 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 |
| EXTI3[3:0] | | | | EXTI2[3:0] | | | | EXTI1[3:0] | | | | EXTI0[3:0] | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

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

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration (x = 0 to 3)

These bits are written by software to select the source input for the EXTIx external interrupt.

0000: PA[x] pin
0001: PB[x] pin
0010: PC[x] pin
0011: PD[x] pin
0100: PE[x] pin
0101: PF[x] pin
0110: PG[x] pin
0111: PH[x] pin
1000: PI[x] pin
1001: PJ[x] pin
1010: PK[x] pin

FIGURE 2.13 – Registre SYSCFG_EXTICR1

```
SYSCFG->EXTICR[0] &= ~(SYSCFG_EXTICR1_EXTI0);
SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI0_PA;
```

Remarque : La valeur par défaut de ce registre étant la même que celle désignant l'entrée du bouton, **PA0**, en l'absence d'horloge sur le contrôleur de configuration système il est tout de même possible de faire fonctionner une interruption sur le bouton utilisateur. Cependant la configuration du multiplexeur par le registre **SYSCFG_EXTICR1** ne fonctionne pas, c'est à dire qu'il est impossible de déclencher une interruption sur l'entrée **PB0** par exemple.

En d'autres termes sans activer l'horloge sur le contrôleur de configuration système, le programme tombe en marche.

### 2.3.3 Sélection du front de déclenchement

Une interruption matérielle se déclenche sur un changement d'état d'une entrée. Il est possible de déclencher une interruption sur un front montant, un front descendant ou les deux. La configuration de l'interruption se fait dans le registre **EXTI_RTSR** pour un front montant et dans le registre **EXTI_FTSR** pour un front descendant.

Dans le cas du programme avec interruption, l'on souhaite faire changer l'état des LEDs au relâchement du bouton. L'entrée du bouton étant de type pull-down, son état de repos est `0`. L'appui provoque un front montant vers l'état `1` et le relâchement un front descendant vers l'état `0`. Il faut donc déclencher l'interruption sur un front descendant.

Il faut pour cela mettre à `1` le premier bit du registre **EXTI_FTSR**.

### 12.3.4 Falling trigger selection register (EXTI_FTSR)

Address offset: 0x0C
Reset value: 0x0000 0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | TR22 | TR21 | TR20 | TR19 | TR18 | TR17 | TR16 |
| | | | | | | | | | rw | rw | rw | rw | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TR15 | TR14 | TR13 | TR12 | TR11 | TR10 | TR9 | TR8 | TR7 | TR6 | TR5 | TR4 | TR3 | TR2 | TR1 | TR0 |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 **TRx:** Falling trigger event configuration bit of line x
0: Falling trigger disabled (for Event and Interrupt) for input line
1: Falling trigger enabled (for Event and Interrupt) for input line.

FIGURE 2.14 – Registre EXTI_FTSR

```
EXTI->FTSR |= EXTI_FTSR_TR0;
```

## 2.3.4 Démasquage de l'interruption

### 12.2.2 EXTI block diagram

*Figure 41* shows the block diagram.

**Figure 41. External interrupt/event controller block diagram**

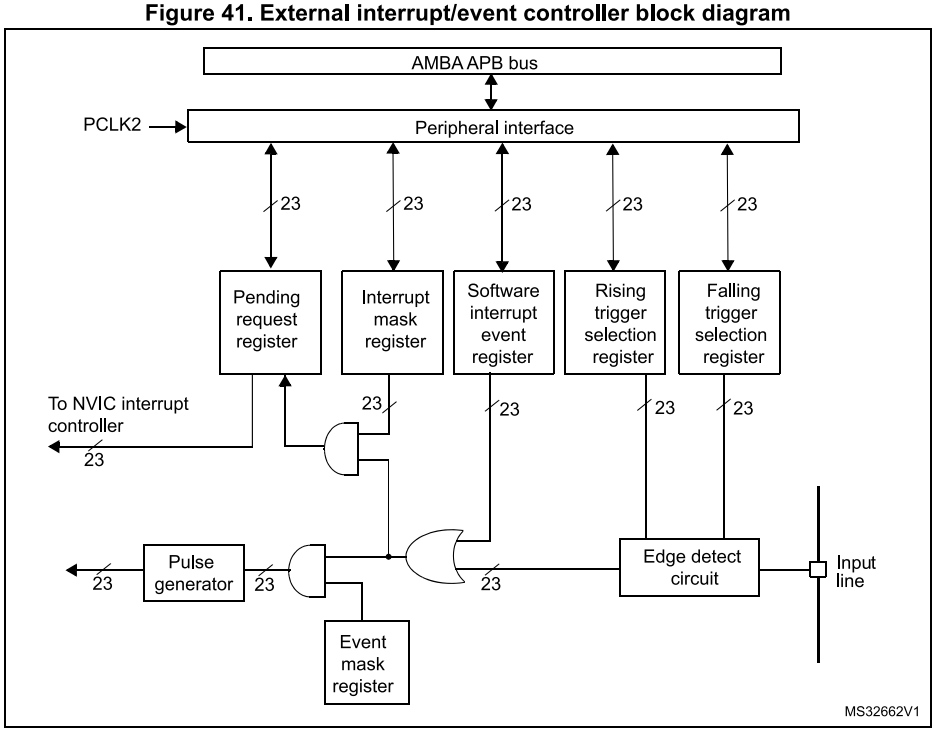


FIGURE 2.15 – Diagramme du contrôleur d'interruptions matérielles

Le masque d'interruption intervient comme un filtre permettant de prendre en compte ou non certaines interruptions matérielles. On le configure à travers le registre **`EXTI_IMR`**. On souhaite ici démasquer la ligne **`0`** des interruptions matérielles.

### 12.3.1 Interrupt mask register (EXTI_IMR)

Address offset: 0x00

Reset value: 0x0000 0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | MR22 | MR21 | MR20 | MR19 | MR18 | MR17 | MR16 |
| | | | | | | | | | rw | rw | rw | rw | rw | rw | rw |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| MR15 | MR14 | MR13 | MR12 | MR11 | MR10 | MR9 | MR8 | MR7 | MR6 | MR5 | MR4 | MR3 | MR2 | MR1 | MR0 |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 **MRx:** Interrupt mask on line x
0: Interrupt request from line x is masked
1: Interrupt request from line x is not masked

FIGURE 2.16 – Registre EXTI_IMR

```
EXTI->IMR |= EXTI_IMR_MR0;
```

### 2.3.5 Configuration de la priorité

La priorité des interruptions est gérée par un autre organe du microcontrôleur, le contrôleur d'interruptions vectorielles imbriquées (NVIC). Celui-ci associe à chaque interruption :

- Un numéro la décrivant, sa position dans la table vectorielle.
- Un numéro de priorité. Certains sont configurables, d'autres non.
- Un pointeur (vecteur) vers une fonction à exécuter lorsque l'interruption a lieu.

**Table 61. Vector table for STM32F405xx/07xx and STM32F415xx/17xx**

| Position | Priority | Type of priority | Acronym | Description | Address |
|---|---|---|---|---|---|
| | - | - | - | Reserved | 0x0000 0000 |
| | -3 | fixed | Reset | Reset | 0x0000 0004 |
| | | | | | |
| 4 | 11 | settable | FLASH | Flash global interrupt | 0x0000 0050 |
| 5 | 12 | settable | RCC | RCC global interrupt | 0x0000 0054 |
| 6 | 13 | settable | EXTI0 | EXTI Line0 interrupt | 0x0000 0058 |
| 7 | 14 | settable | EXTI1 | EXTI Line1 interrupt | 0x0000 005C |

FIGURE 2.17 – Extrait de la table vectorielle du NVIC

La position de l'interruption du bouton utilisateur **`EXTI0`** est **`6`**, son numéro de priorité par défaut est **`13`**. Les numéros de priorité par défaut ne se configurent pas par défaut, il faut le faire soi même à travers les registres **`NVIC_IPRx`**.

### 4.3.7 Interrupt priority registers (NVIC_IPRx)

Address offset: 0x00- 0x0B

Reset value: 0x0000 0000

Required privilege: Privileged

The NVIC_IPR0-IPR80 registers provide an 8-bit priority field for each interrupt. These registers are byte-accessible. Each register holds four priority fields, that map to four elements in the CMSIS interrupt priority array IP[0] to IP[67], as shown in *Figure 19*.

**Figure 19. NVIC_IPRx register mapping**

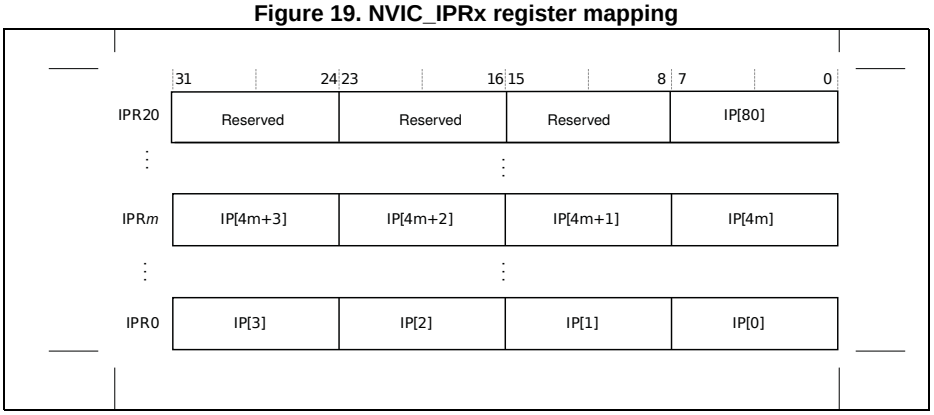


**Table 46. IPR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:24] | Priority, byte offset 3 | Each priority field holds a priority value, 0-255. The lower the value, the greater the priority of the corresponding interrupt. The processor implements only bits[7:4] of each field, bits[3:0] read as zero and ignore writes. |
| [23:16] | Priority, byte offset 2 | |
| [15:8] | Priority, byte offset 1 | |
| [7:0] | Priority, byte offset 0 | |

FIGURE 2.18 – Registre NVIC_IPRx

Chaque registre **`NVIC_IPRx`** contient quatre blocs **`IP[x]`**. De plus chaque bloc **`IP[x]`** de 8 bits est composés de quatre bits de poids fort déterminant la priorité et de quatre bits de poids faible figés à **`0`**.

Ici nous souhaitons mettre à **`13`** la priorité du vecteur d'interruption numéro **`6`**. Il faut donc mettre à **`13`** les quatre bits de poids fort du bloc **`IP[6]`** situé dans le registre **`NVIC_IPR1`**.

CMSIS propose une syntaxe relative aux blocs **`IP[x]`** plutôt qu'aux registres **`NVIC_IPRx`** ainsi qu'une fonction à l'effet équivalent mais à la syntaxe plus intuitive.

```
NVIC->IP[EXTI0_IRQn] = 13<<4;
```

```
NVIC_SetPriority(EXTI0_IRQn, 13);
```

### 2.3.6 Activation de l'interruption

La dernière étape est d'activer l'interruption au niveau du contrôleur NVIC. Pour cela il faut mettre à `1` le bit `6` du registre `NVIC_ISER0`.

#### 4.3.2 Interrupt set-enable registers (NVIC_ISERx)

Address offset: 0x00 - 0x0B

Reset value: 0x0000 0000

Required privilege: Privileged

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SETENA[31:16] | | | | | | | | | | | | | | | |
| rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SETENA[15:0] | | | | | | | | | | | | | | | |
| rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs |

Bits 31:0 **SETENA**: Interrupt set-enable bits.

**Write**:

0: No effect

1: Enable interrupt

**Read**:

0: Interrupt disabled

1: Interrupt enabled.

If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt, regardless of its priority.

FIGURE 2.19 – Registre NVIC_ISERx

Ici aussi CMSIS propose une fonction équivalente à la syntaxe plus commode.

```
NVIC->ISER[0] |= (1<<EXTI0_IRQn);
```

```
NVIC_EnableIRQ(EXTI0_IRQn);
```

## 2.4 Usage du bouton utilisateur

### 2.4.1 Lecture de l'état du bouton

La lecture de l'état du bouton utilisateur se fait à travers le premier bit du registre `GPIOA_IDR`. L'entrée étant en mode pull-down, un `0` signifie que le bouton est au repos et un `1` signifie qu'il est enfoncé.

### 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.

FIGURE 2.20 – Registre GPIOx_IDR

```
if(GPIOA->IDR & GPIO_IDR_ID0) {
    // do something
    }
```

## 2.5 Usage des LEDs

### 2.5.1 Contrôle de l'état des LEDs

L'état des lignes connectées aux LEDs est image de la valeur des bits `12`, `13`, `14` et `15` du registre `GPIOD_ODR`. Pour allumer toutes les LEDs, il faut mettre tous ces bits à `1`.

### 8.4.6 GPIO port output data register (GPIOx_ODR) (x = A..I/J/K)

Address offset: 0x14

Reset value: 0x0000 0000

| 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 |
| ODR15 | ODR14 | ODR13 | ODR12 | ODR11 | ODR10 | ODR9 | ODR8 | ODR7 | ODR6 | ODR5 | ODR4 | ODR3 | ODR2 | ODR1 | ODR0 |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

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

Bits 15:0 **ODRy**: Port output data (y = 0..15)

These bits can be read and written by software.

*Note: For atomic bit set/reset, the ODR bits can be individually set and reset by writing to the GPIOx_BSRR register (x = A..I/J/K).*

FIGURE 2.21 – Registre GPIOx_ODR

```
GPIOD->ODR |= (GPIO_ODR_OD12
              |GPIO_ODR_OD13
              |GPIO_ODR_OD14
              |GPIO_ODR_OD15);
```

## 2.6 Usage des interruptions

### 2.6.1 Écriture de l'interruption

L'écriture du code à exécuter lorsqu'une interruption a lieu se fait dans une fonction au nom prédéterminé. Son nom est composé du nom de l'interruption, ici **`EXTI0`** suivi de **`_IRQHandler`**. Ces fonctions ne prennent pas d'arguments et ne renvoient rien.

```
1 void EXTI0_IRQHandler(void) {
2     // do something
3     }
```

### 2.6.2 Réinitialisation de l'état d'attente

Lorsque une interruption est déclenchée des bits passent à `1` dans certains registres pour indiquer qu'elle est en attente de traitement. Lorsque tout le code de l'interruption a été exécuté, il faut réinitialiser ces bits pour indiquer que l'interruption a été traitée et que des interruptions moins prioritaires puissent être traitées si elles venaient à survenir.

Dans notre cas, deux registres indiquent l'état d'attente, le registre **`EXTI_PR`** au niveau du contrôleur d'interruptions externes et le registre **`NVIC_ICPR0`** au niveau du contrôleur d'interruptions vectorielles imbriquées.

Remarque : Ces registres ont tous deux la particularité de se remettre à `0` lorsque l'on y écrit `1`.

#### 12.3.6 Pending register (EXTI_PR)

Address offset: 0x14
Reset value: undefined

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | PR22 | PR21 | PR20 | PR19 | PR18 | PR17 | PR16 |
| | | | | | | | | | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PR15 | PR14 | PR13 | PR12 | PR11 | PR10 | PR9 | PR8 | PR7 | PR6 | PR5 | PR4 | PR3 | PR2 | PR1 | PR0 |
| rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 |

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 **PRx:** Pending bit
0: No trigger request occurred
1: selected trigger request occurred
This bit is set when the selected edge event arrives on the external interrupt line.
This bit is cleared by programming it to '1'.

FIGURE 2.22 – Registre EXTI_PR

```
1 EXTI->PR |= EXTI_PR_PR0;
```

### 4.3.5 Interrupt clear-pending registers (NVIC_ICPRx)

Address offset: 0x00 - 0x0B

Reset value: 0x0000 0000

Required privilege: Privileged

The ICPR0-ICPR2 registers remove the pending state from interrupts, and show which interrupts are pending.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CLRPEND[31:16] | | | | | | | | | | | | | | | |
| rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CLRPEND[15:0] | | | | | | | | | | | | | | | |
| rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 |

Bits 31:0 **CLRPEND**: Interrupt clear-pending bits

**Write**:

0: No effect

1: Removes the pending state of an interrupt

**Read**:

0: Interrupt is not pending

1: Interrupt is pending

Writing 1 to an ICPR bit does not affect the active state of the corresponding interrupt.

FIGURE 2.23 – Registre NVIC_ICPRx

Comme pour les autres utilisations du contrôleur NVIC, CMSIS propose une fonction alternative à la syntaxe plus digeste.

```
NVIC->ICPR[0] |= (1 << EXTI0_IRQn);
```

```
NVIC_ClearPendingIRQ(EXTI0_IRQn);
```

# Chapitre 3

# Programmes

## 3.1 Programme en mode polling

Dans le programme en mode polling, la surveillance du bouton utilisateur se fait dans une boucle perpétuelle faisant partie de la fonction `main()`. Une fonction `delay()` est ajoutée pour corriger l'effet de rebond propre aux interrupteurs.

De plus, afin d'interdire le défilement des LEDs en maintenant le bouton enfoncé, l'on ajoute une boucle vide durant tant que le bouton est enfoncé. Cela dans le but que les deux programmes se comportent de la même façon, les interruption n'ayant lieu qu'à l'appui sur le bouton.

### 3.1.1 Code source

```
#include "stm32f4xx.h"
#include "stm32f411e_discovery.h"

void init(void) {
    //enable clock for portA & portD
    RCC->AHB1ENR |= (RCC_AHB1ENR_GPIOAEN
                    |RCC_AHB1ENR_GPIODEN);

    //user button in input
    GPIOA->MODER &= ~(GPIO_MODER_MODER0);
    //user button in pull-down
    GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR0);
    GPIOA->PUPDR |= GPIO_PUPDR_PUPDR0_1;

    //4 LEDs in output
    GPIOD->MODER &= ~(GPIO_MODER_MODER12
                     |GPIO_MODER_MODER13
                     |GPIO_MODER_MODER14
                     |GPIO_MODER_MODER15);
    GPIOD->MODER |= (GPIO_MODER_MODER12_0
                    |GPIO_MODER_MODER13_0
                    |GPIO_MODER_MODER14_0
                    |GPIO_MODER_MODER15_0);
    //4 LEDs in push-pull
    GPIOD->OTYPER &= ~(GPIO_OTYPER_OT12
                      |GPIO_OTYPER_OT13
                      |GPIO_OTYPER_OT14
                      |GPIO_OTYPER_OT15);
    }

void delay(unsigned int time) {
    for(int i=0;i<time*4000;i++);
    }

int main(void) {
    init();
    while(1) {
        if(GPIOA->IDR & GPIO_IDR_ID0) {
            if(GPIOD->ODR & GPIO_ODR_OD12) {
```

```
                GPIOD->ODR &= ~(GPIO_ODR_OD12);
                GPIOD->ODR |= GPIO_ODR_OD14;
            } else if(GPIOD->ODR & GPIO_ODR_OD14) {
                GPIOD->ODR &= ~(GPIO_ODR_OD14);
                GPIOD->ODR |= GPIO_ODR_OD13;
            } else if(GPIOD->ODR & GPIO_ODR_OD13) {
                GPIOD->ODR &= ~(GPIO_ODR_OD13);
                GPIOD->ODR |= GPIO_ODR_OD15;
            } else if(GPIOD->ODR & GPIO_ODR_OD15) {
                GPIOD->ODR &= ~(GPIO_ODR_OD15);
                GPIOD->ODR |= GPIO_ODR_OD12;
            } else {
                GPIOD->ODR &= ~(GPIO_ODR_OD12
                                |GPIO_ODR_OD13
                                |GPIO_ODR_OD14
                                |GPIO_ODR_OD15);
                GPIOD->ODR |= GPIO_ODR_OD12;
                }
            while(GPIOA->IDR & GPIO_IDR_ID0);
            delay(50);
            }
        }
    }
```

## 3.2 Programme avec interruption

Cette fois ci, le code se trouve dans la fonction **`EXTI0_IRQHandler()`**, le **`main()`** est comblé avec une boucle vide. On utilise également dans ce programme un délai anti-rebond.

### 3.2.1 Code source

```
#include "stm32f4xx.h"
#include "stm32f411e_discovery.h"

void init(void) {
    //enable clock for portA & portD & system configuration controller
    RCC->AHB1ENR |= (RCC_AHB1ENR_GPIOAEN
                    |RCC_AHB1ENR_GPIODEN);
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

    //user button in input
    GPIOA->MODER &= ~(GPIO_MODER_MODER0);
    //user button in pull-down
    GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR0);
    GPIOA->PUPDR |= GPIO_PUPDR_PUPDR0_1;

    //4 LEDs in output
    GPIOD->MODER &= ~(GPIO_MODER_MODER12
                     |GPIO_MODER_MODER13
                     |GPIO_MODER_MODER14
                     |GPIO_MODER_MODER15);
    GPIOD->MODER |= (GPIO_MODER_MODER12_0
                    |GPIO_MODER_MODER13_0
                    |GPIO_MODER_MODER14_0
                    |GPIO_MODER_MODER15_0);
    //4 LEDs in push-pull
    GPIOD->OTYPER &= ~(GPIO_OTYPER_OT12
                      |GPIO_OTYPER_OT13
                      |GPIO_OTYPER_OT14
                      |GPIO_OTYPER_OT15);

    //external interrupt line 0 is user button
    SYSCFG->EXTICR[0] &= ~(SYSCFG_EXTICR1_EXTI0);
    SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI0_PA;
    //trigger on falling edge
```

```
35      EXTI->FTSR |= EXTI_FTSR_TR0;
36      //enable interrupt for line 0
37      EXTI->IMR |= EXTI_IMR_MR0;
38      //interrupt priority 13
39  //  NVIC_SetPriority(EXTI0_IRQn, 13);
40      NVIC->IP[EXTI0_IRQn] = 13<<4;
41      //enable interrupt for line 0
42  //  NVIC_EnableIRQ(EXTI0_IRQn);
43      NVIC->ISER[0] |= (1<<EXTI0_IRQn);
44      }
45
46  void delay(unsigned int time) {
47      for(int i=0;i<time*4000;i++);
48      }
49
50  void EXTI0_IRQHandler(void) {
51      if(GPIOD->ODR & GPIO_ODR_OD15) {
52          GPIOD->ODR &= ~(GPIO_ODR_OD15);
53          GPIOD->ODR |= GPIO_ODR_OD14;
54      } else if(GPIOD->ODR & GPIO_ODR_OD14) {
55          GPIOD->ODR &= ~(GPIO_ODR_OD14);
56          GPIOD->ODR |= GPIO_ODR_OD13;
57      } else if(GPIOD->ODR & GPIO_ODR_OD13) {
58          GPIOD->ODR &= ~(GPIO_ODR_OD13);
59          GPIOD->ODR |= GPIO_ODR_OD12;
60      } else if(GPIOD->ODR & GPIO_ODR_OD12) {
61          GPIOD->ODR &= ~(GPIO_ODR_OD12);
62          GPIOD->ODR |= GPIO_ODR_OD15;
63      } else {
64          GPIOD->ODR &= ~(GPIO_ODR_OD12
65                          |GPIO_ODR_OD13
66                          |GPIO_ODR_OD14
67                          |GPIO_ODR_OD15);
68          GPIOD->ODR |= GPIO_ODR_OD14;
69          }
70      delay(50);
71      //interrupt is done -> clear its pending state
72  //  NVIC_ClearPendingIRQ(EXTI0_IRQn);
73      NVIC->ICPR[0] |= (1 << EXTI0_IRQn);
74      EXTI->PR |= EXTI_PR_PR0;
75      }
76
77  int main(void) {
78      init();
79      while(1);
80      }
```