ÉCOLE SUPÉRIEURE DES TECHNOLOGIES
ÉLECTRONIQUE, INFORMATIQUE, INFOGRAPHIE

ARCHITECTURE DES SYSTÈMES

# TP2 - STM32F411
# Contrôle des LEDs par PWM

*Auteur :*
Thomas LEPOIX

MASTER 2 Systèmes Embarqués

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

12 décembre 2018

# Table des matières

# Chapitre 1

# Introduction

## 1.1 Cahier des charges

Le but de ce TP est de contrôler des LEDs par un signal modulé en largeur d'impulsion (PWM) sur un microcontrôleur ARM Cortex-M4.

Pour cela un programme est à réaliser : Il devra allumer deux LEDs, la première à 75% de sa capacité, la seconde à 25%.

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

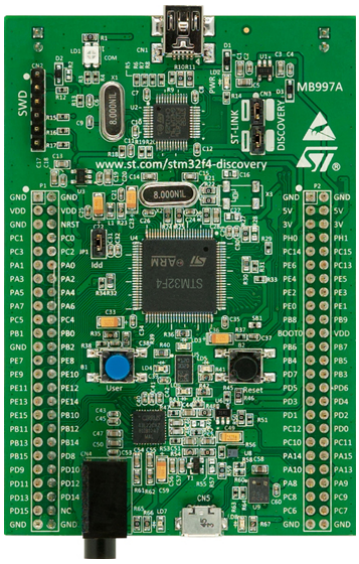



FIGURE 1.1 – Carte de développement STM32F4-Discovery

## 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 Analyse de la structure du microcontrôleur

### 2.1.1 Accès aux LEDs

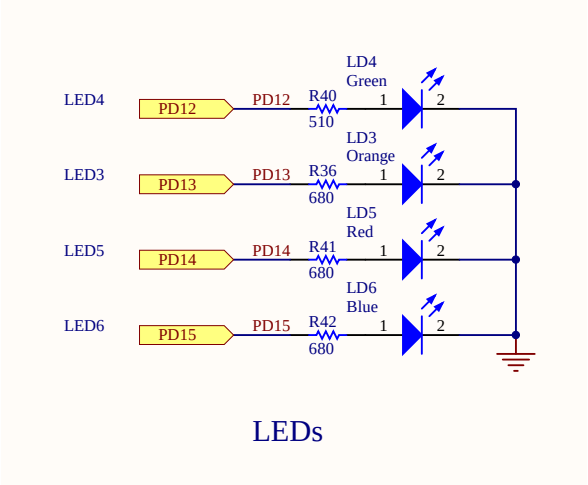



FIGURE 2.1 – Schéma de raccordement des LEDs

Les LEDs verte et orange se situent respectivement sur les entrées / sorties `12` et `13` du portD.

### 2.1.2 Choix du Timer

Le Timer est la fonction du microcontrôleur permettant, entre autres, de générer des signaux PWM.

Le choix du Timer que l'on va utiliser ne se fait pas au hasard, en effet si l'on prête attention au multiplexage des fonctions sur les pins du microcontrôleur, l'on remarque que le canal `1` du Timer `4` est physiquement raccordé à la LED verte et que le canal `2` de ce même Timer est raccordé à la LED orange.

**Table 7. STM32F40xxx pin and ball definitions**

| Pin number | | | | | | Pin name (function after reset)[1] | Pin type | I / O structure | Notes | Alternate functions | Additional functions |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LQFP64 | WLCSP90 | LQFP100 | LQFP144 | UFBGA176 | LQFP176 | | | | | | |
| - | G2 | 59 | 81 | N13 | 100 | PD12 | I/O | FT | - | FSMC_ALE/ FSMC_A17/TIM4_CH1 / USART3_RTS/ EVENTOUT | - |
| - | - | 60 | 82 | M15 | 101 | PD13 | I/O | FT | - | FSMC_A18/TIM4_CH2/ EVENTOUT | - |

FIGURE 2.2 – Correspondance des Timers et des LEDs

## 2.2 Configuration des deux LEDs

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

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 2.3 – Registre RCC_AHB1ENR

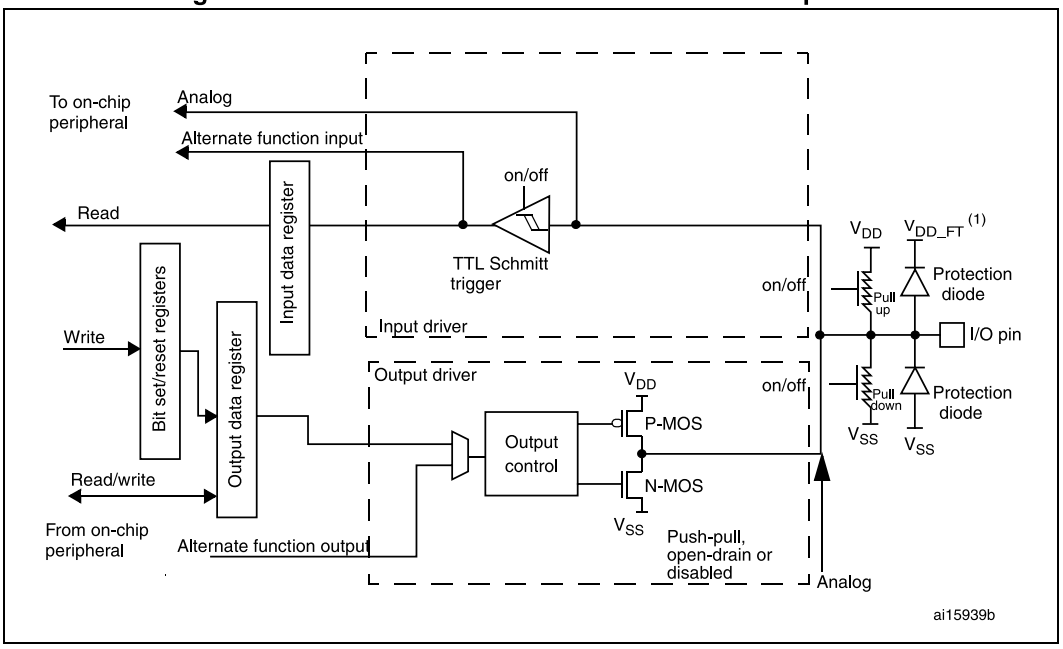
le portD étant sur le bus **AHB1**, l'activation de son horloge se fait en mettant à **1** le quatrième bit du registre **RCC_AHB1ENR**.

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

### 2.2.2 Contrôle des LEDs par les périphériques internes du microcontrôleur

Les sorties du microcontrôleur peuvent êtres contrôlées directement par l'utilisateur, il s'agit du mode sortie standard. Elles peuvent également être contrôlées par les périphériques internes du microcontrôleur, les "fonctions alternatives".

**Figure 25. Basic structure of a five-volt tolerant I/O port bit**




1. $V_{DD\_FT}$ is a potential specific to five-volt tolerant I/Os and different from $V_{DD}$.

FIGURE 2.4 – Raccordement interne des entrées / sorties

Pour soumettre les deux LEDs aux fonctions internes du microcontrôleur, on affecte la valeur `10` aux 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 2.5 – Registre GPIOx_MODER

```
GPIOD->MODER &= ~(GPIO_MODER_MODER12
                  |GPIO_MODER_MODER13);
GPIOD->MODER |= (GPIO_MODER_MODER12_1
                 |GPIO_MODER_MODER13_1);
```

### 2.2.3 Définition du type de sortie des LEDs

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.

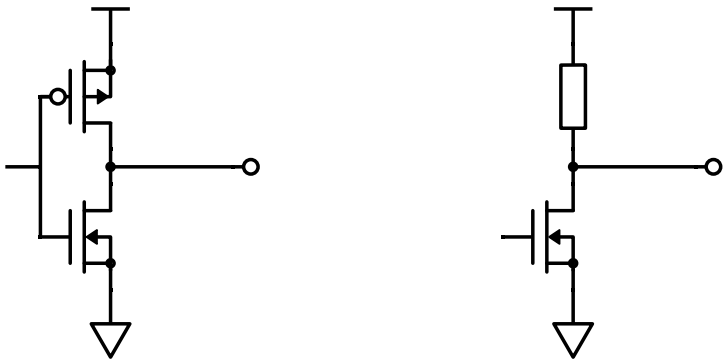


FIGURE 2.6 – Configuration push-pull à droite et open-drain à gauche

Sans contrainte particulière, ce choix est arbitraire. On préfère généralement le mode push-pull. Pour le choisir, on affecte la valeur `0` aux bits 12 et 13 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 2.7 – Registre GPIOx_OTYPER

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

## 2.2.4 Choix du périphérique interne

Les fonctions alternatives sont soumises à un multiplexage, il faut donc choisir quels périphériques auront accès aux sorties.

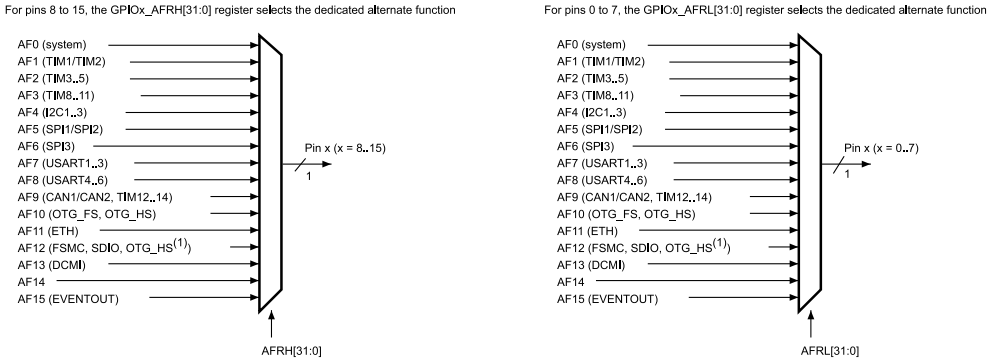



FIGURE 2.8 – Multiplexage des périphériques internes du microcontrôleur

Les deux LEDs étant contrôlées par le Timer **4**, il s'agit de la fonction alternative **AF2**. On entre la valeur **0010** dans les champs de quatre bits correspondants dans le registre **GPIOD_AFRH**.

### 8.4.10 GPIO alternate function high register (GPIOx_AFRH) (x = A..I/J)

Address offset: 0x24

Reset value: 0x0000 0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AFRH15[3:0] | | | | AFRH14[3:0] | | | | AFRH13[3:0] | | | | AFRH12[3:0] | | | |
| 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 |
| AFRH11[3:0] | | | | AFRH10[3:0] | | | | AFRH9[3:0] | | | | AFRH8[3:0] | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 31:0 **AFRHy:** Alternate function selection for port x bit y (y = 8..15)
These bits are written by software to configure alternate function I/Os

AFRHy selection:

| | |
|---|---|
| 0000: AF0 | 1000: AF8 |
| 0001: AF1 | 1001: AF9 |
| 0010: AF2 | 1010: AF10 |
| 0011: AF3 | 1011: AF11 |
| 0100: AF4 | 1100: AF12 |
| 0101: AF5 | 1101: AF13 |
| 0110: AF6 | 1110: AF14 |
| 0111: AF7 | 1111: AF15 |

FIGURE 2.9 – Registre GPIOx_AFRH

```
GPIOD->AFR[1] &= ~(GPIO_AFRH_AFSEL12
                  |GPIO_AFRH_AFSEL13);
GPIOD->AFR[1] |= (GPIO_AFRH_AFSEL12_1
                 |GPIO_AFRH_AFSEL13_1);
```

## 2.3 Configuration du Timer

Les Timers sont des fonctions complexes permettant de réaliser de nombreuses choses. La documentation offre un paragraphe expliquant la marche à suivre pour configurer un signal PWM, il suffit de respecter la procédure.

**18.3.9 PWM mode**

Pulse width modulation mode allows generating a signal with a frequency determined by the value of the TIMx_ARR register and a duty cycle determined by the value of the TIMx_CCRx register.

The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing 110 (PWM mode 1) or '111 (PWM mode 2) in the OCxM bits in the TIMx_CCMRx register. The user must enable the corresponding preload register by setting the OCxPE bit in the TIMx_CCMRx register, and eventually the auto-reload preload register by setting the ARPE bit in the TIMx_CR1 register.

As the preload registers are transferred to the shadow registers only when an update event occurs, before starting the counter, the user has to initialize all the registers by setting the UG bit in the TIMx_EGR register.

OCx polarity is software programmable using the CCxP bit in the TIMx_CCER register. It can be programmed as active high or active low. OCx output is enabled by the CCxE bit in the TIMx_CCER register. Refer to the TIMx_CCERx register description for more details.

In PWM mode (1 or 2), TIMx_CNT and TIMx_CCRx are always compared to determine whether TIMx_CCRx≤TIMx_CNT or TIMx_CNT≤TIMx_CCRx (depending on the direction of the counter). However, to comply with the ETRF (OCREF can be cleared by an external event through the ETR signal until the next PWM period), the OCREF signal is asserted only:

- When the result of the comparison changes, or
- When the output compare mode (OCxM bits in TIMx_CCMRx register) switches from the "frozen" configuration (no comparison, OCxM='000) to one of the PWM modes (OCxM='110 or '111).

This forces the PWM by software while the timer is running.

The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIMx_CR1 register.

FIGURE 2.10 – Configuration d'un signal PWM

### 2.3.1 Configuration de la fréquence du Timer

La fréquence du Timer est déterminée par le registre **TIM4_ARR**. Ce registre contient le nombre de périodes d'horloge que constituent une période du signal. Ce registre est de 16 bits pour le Timer **4**.

Dans notre cas où le rapport cyclique est au pourcent près, il convient d'avoir une période de 100 fronts d'horloge. Il est inutile d'augmenter davantage la période car si la fréquence du signal diminue trop, il devient possible de percevoir à l'œil la fréquence de rafraîchissement des LEDs, ce qui n'est pas souhaitable.

Il est également possible de ralentir l'horloge selon le facteur du registre **TIM4_PSC**, de 32 bits, mais cela n'a aucun intérêt dans notre cas pour la raison citée précédemment.

#### 18.4.12 TIMx auto-reload register (TIMx_ARR)

Address offset: 0x2C

Reset value: 0xFFFF FFFF

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ARR[31:16] (depending on timers) | | | | | | | | | | | | | | | |
| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ARR[15:0] | | | | | | | | | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 31:16 **ARR[31:16]**: High auto-reload value (on TIM2 and TIM5).

Bits 15:0 **ARR[15:0]**: Auto-reload value

FIGURE 2.11 – Registre TIMx_ARR

```
TIM4->ARR = 100;
```

### 2.3.2 Configuration du rapport cyclique des signaux

Le rapport cyclique des signaux, et plus précisément le nombre de périodes d'horloge à l'état haut (ou bas selon la polarité) par période de signal, est déterminé par les registres **TIM4_CCR1** et **TIM4_CCR2** pour chacun des canaux.

#### 18.4.13 TIMx capture/compare register 1 (TIMx_CCR1)

Address offset: 0x34

Reset value: 0x0000 0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CCR1[31:16] (depending on timers) | | | | | | | | | | | | | | | |
| rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CCR1[15:0] | | | | | | | | | | | | | | | |
| rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro | rw/ro |

Bits 31:16 **CCR1[31:16]**: High Capture/Compare 1 value (on TIM2 and TIM5).

Bits 15:0 **CCR1[15:0]**: Low Capture/Compare 1 value

**If channel CC1 is configured as output**:
CCR1 is the value to be loaded in the actual capture/compare 1 register (preload value).
It is loaded permanently if the preload feature is not selected in the TIMx_CCMR1 register (bit OC1PE). Else the preload value is copied in the active capture/compare 1 register when an update event occurs.
The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signaled on OC1 output.

FIGURE 2.12 – Registre TIMx_CCR1

```
//duty cycle in percent [0-100]
#define    DUTY_CYCLE_1    75
#define    DUTY_CYCLE_2    25

TIM4->CCR1 = (TIM4->ARR*DUTY_CYCLE_1/100);
TIM4->CCR2 = (TIM4->ARR*DUTY_CYCLE_2/100);
```

### 2.3.3 Mise en mode PWM des canaux 1 et 2 du Timer

Les trios de bits **`OCxM`** permettent pour chaque canal de définir le mode du Timer. Deux d'entre eux sont consacrés aux signaux PWM, le premier en logique positive, le rapport cyclique décrit le temps à l'état haut et le second en logique négative, le rapport cyclyque décrit le temps à l'état bas.

Il faut également mettre à **`1`** les bits **`OCxPE`**. Cela a pour effet d'empêcher une mise-à-jour sauvage des registres dédiés au rapport cyclique **`TIM4_CCRx`**. La nouvelle valeur est stockée le temps que le prochain événement de mise-à-jour se déclenche. Ceux-ci surviennent par exemple lorsque le compteur atteint la valeur du registre **`TIM4_ARR`**, c'est-à-dire à la fin d'une période. Ils peuvent également survenir lorsque le signal change d'état durant une période.

Cette fonctionnalité sert à protéger certains équipements sensibles notamment. Elle n'est pas vraiment utile dans notre cas où le rapport cyclique de chaque signal ne sera pas amené à changer durant le programme, toutefois la documentation précise que ne pas mettre à **`1`** ces bits en mode PWM donne lieu à des comportements imprévisibles des Timers.

#### 18.4.7 TIMx capture/compare mode register 1 (TIMx_CCMR1)

Address offset: 0x18

Reset value: 0x0000

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OC2CE | OC2M[2:0] | | | OC2PE | OC2FE | CC2S[1:0] | | OC1CE | OC1M[2:0] | | | OC1PE | OC1FE | CC1S[1:0] | |
| IC2F[3:0] | | | | IC2PSC[1:0] | | | | IC1F[3:0] | | | | IC1PSC[1:0] | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

**Output compare mode**

Bits 6:4 **OC1M**: Output compare 1 mode

These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.

110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx_CNT<TIMx_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF='0) as long as TIMx_CNT>TIMx_CCR1 else active (OC1REF=1).

111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx_CNT<TIMx_CCR1 else active. In downcounting, channel 1 is active as long as TIMx_CNT>TIMx_CCR1 else inactive.

Bit 3 **OC1PE**: Output compare 1 preload enable

0: Preload register on TIMx_CCR1 disabled. TIMx_CCR1 can be written at anytime, the new value is taken in account immediately.

1: Preload register on TIMx_CCR1 enabled. Read/Write operations access the preload register. TIMx_CCR1 preload value is loaded in the active register at each update event.

*Note: **2:** The PWM mode can be used without validating the preload register only in one-pulse mode (OPM bit set in TIMx_CR1 register). Else the behavior is not guaranteed.*

FIGURE 2.13 – Registre TIMx_CCMR1

```
TIM4->CCMR1 |= (TIM_CCMR1_OC1M_2
               |TIM_CCMR1_OC1M_1
               |TIM_CCMR1_OC1PE
               |TIM_CCMR1_OC2M_2
               |TIM_CCMR1_OC2M_1
               |TIM_CCMR1_OC2PE);
```

### 2.3.4 Activation du Timer en mode edge-aligned

Mettre à `0` le couple de bits `CMS` du registre `TIM4_CR1` sélectionne le mode edge-aligned par opposition au mode center-aligned. Il s'agit de deux types de signaux PWM. Le mode standard edge-aligned convient pour contrôler des LEDs, il est donc superflu d'utiliser le mode center-aligned. Celui-ci est généralement utilisé pour contrôler des moteurs.

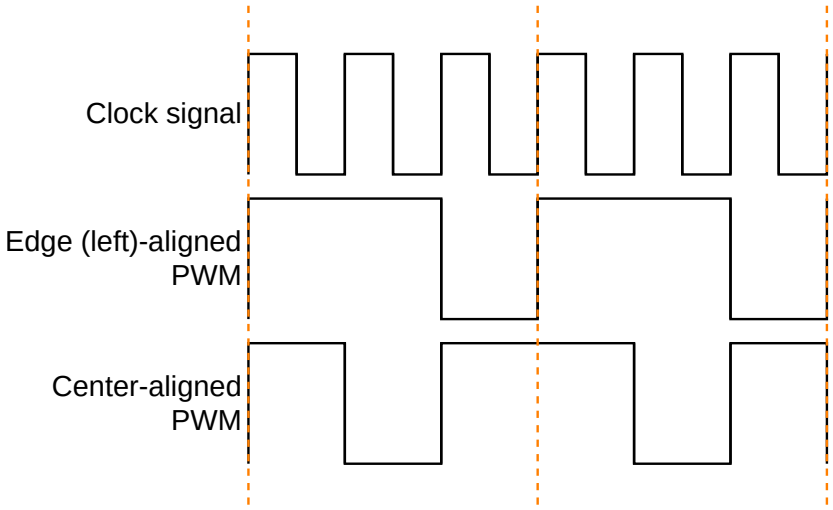



FIGURE 2.14 – Signaux PWM de rapport cyclique 2/3 alignés à gauche et au centre

De la même façon que les bits `OCxPE` étudiés ci-dessus protègent d'une mise-à-jour intempestive les registres du rapport cyclique `TIM4_CCRx`, Le bit `ARPE` protège le registre de la fréquence `TIM4_ARR`. Il ne peut alors être mis à jour qu'à la fin d'une période, la nouvelle valeur étant stockée dans un buffer le temps que l'événement se produise.

Activer cette protection n'est en revanche pas nécessaire pour le bon fonctionnement du Timer.

Enfin, le bit `CEN` permet d'activer le compteur `TIM4_CNT` et donc le Timer lui même.

### 18.4.1 TIMx control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x0000

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | CKD[1:0] | | ARPE | CMS | | DIR | OPM | URS | UDIS | CEN |
| | | | | | | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bit 7 **ARPE**: Auto-reload preload enable
0: TIMx_ARR register is not buffered
1: TIMx_ARR register is buffered

Bits 6:5 **CMS**: Center-aligned mode selection
00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).
01: Center-aligned mode 1.
10: Center-aligned mode 2.
11: Center-aligned mode 3.

*Note: It is not allowed to switch from edge-aligned mode to center-aligned mode as long as the counter is enabled (CEN=1)*

Bit 4 **DIR**: Direction
0: Counter used as upcounter
1: Counter used as downcounter

Bit 0 **CEN**: Counter enable
0: Counter disabled
1: Counter enabled

FIGURE 2.15 – Registre TIMx_CR1

```
TIM4->CR1 &= ~(TIM_CR1_CMS);
TIM4->CR1 |= (TIM_CR1_ARPE
             |TIM_CR1_CEN);
```

### 2.3.5 Démarrage des canaux 1 et 2 du Timer en polarité positive

La dernière étape est de déclencher les signaux pour chacun des canaux du Timer. Le registre permettant cela dispose également de bits régissant la polarité des signaux. L'on conserve une polarité positive.

#### 18.4.9 TIMx capture/compare enable register (TIMx_CCER)

Address offset: 0x20

Reset value: 0x0000

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CC4NP | Res. | CC4P | CC4E | CC3NP | Res. | CC3P | CC3E | CC2NP | Res. | CC2P | CC2E | CC1NP | Res. | CC1P | CC1E |
| rw | | rw | rw | rw | | rw | rw | rw | | rw | rw | rw | | rw | rw |

Bit 3 **CC1NP**: *Capture/Compare 1 output Polarity.*
CC1 channel configured as output:
CC1NP must be kept cleared in this case.

Bit 2 Reserved, must be kept at reset value.

Bit 1 **CC1P**: *Capture/Compare 1 output Polarity.*
**CC1 channel configured as output:**
0: OC1 active high
1: OC1 active low

Bit 0 **CC1E**: *Capture/Compare 1 output enable.*
**CC1 channel configured as output:**
0: Off - OC1 is not active
1: On - OC1 signal is output on the corresponding output pin

FIGURE 2.16 – Registre TIMx_CCER

```
TIM4->CCER &= ~(TIM_CCER_CC1P
               |TIM_CCER_CC2P);
TIM4->CCER |= (TIM_CCER_CC1E
              |TIM_CCER_CC2E);
```

# Chapitre 3

# Programme

L'utilisation des Timers pour le contrôle des LEDs a le gros avantage de ne pas utiliser la ressource du processeur, les Timers étant un autre organe du microcontrôleur.

## 3.1 Code source

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

//Duty cycles in percent [0-100]
#define    DUTY_CYCLE_1    75
#define    DUTY_CYCLE_2    25

void init(void) {
    //enable clock for portD & timer4
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN;
    RCC->APB1ENR |= RCC_APB1ENR_TIM4EN;

    //alternate function mode
    GPIOD->MODER &= ~(GPIO_MODER_MODER12
                      |GPIO_MODER_MODER13);
    GPIOD->MODER |= (GPIO_MODER_MODER12_1
                     |GPIO_MODER_MODER13_1);
    //push-pull
    GPIOD->OTYPER &= ~(GPIO_OTYPER_OT12
                       |GPIO_OTYPER_OT13);
    //alternate function 2 (timer 4)
    GPIOD->AFR[1] &= ~(GPIO_AFRH_AFSEL12
                       |GPIO_AFRH_AFSEL13);
    GPIOD->AFR[1] |= (GPIO_AFRH_AFSEL12_1
                      |GPIO_AFRH_AFSEL13_1);

    //divide frequency
//  TIM4->PSC = 0xFFFFFFFF;
    //frequency 100*systick
    TIM4->ARR = 100;
    //duty cycle
    TIM4->CCR1 = (TIM4->ARR*DUTY_CYCLE_1/100);
    TIM4->CCR2 = (TIM4->ARR*DUTY_CYCLE_2/100);
    //channels 1 & 2 in PWM positive mode, protect untimely duty cycle update
    TIM4->CCMR1 |= (TIM_CCMR1_OC1M_2
                    |TIM_CCMR1_OC1M_1
                    |TIM_CCMR1_OC1PE
                    |TIM_CCMR1_OC2M_2
                    |TIM_CCMR1_OC2M_1
                    |TIM_CCMR1_OC2PE);
    //edge aligned
    TIM4->CR1 &= ~(TIM_CR1_CMS);
    //enable timer4, protect untimely frequency update
    TIM4->CR1 |= (TIM_CR1_ARPE
                  |TIM_CR1_CEN);
    //start channels 1 & 2 in high polarity
    TIM4->CCER &= ~(TIM_CCER_CC1P
                    |TIM_CCER_CC2P);
```

```
49      TIM4->CCER |= (TIM_CCER_CC1E
50                    |TIM_CCER_CC2E);
51      }
52
53  int main(void) {
54      init();
55      while(1);
56      }
```