

# TD n°4

**GPIO STM32**

Romain POSSÉMÉ  
Théo MAINGUENÉ  
Nicolas LE GUERROUÉ

18 décembre 2020

# Table des matières

|       |  |   |
|-------|--|---|
| 0.1   | Question 1 . . . . .                           | 2 |
| 0.1.1 | 1.a . . . . .                                  | 2 |
| 0.1.2 | 1.b . . . . .                                  | 2 |
| 0.1.3 | 1.c . . . . .                                  | 3 |
| 0.2   | Question 2 . . . . .                           | 3 |
| 0.2.1 | 2.a) . . . . .                                 | 3 |
|       | Instruction . . . . .                          | 3 |
|       | Justification de l'instruction . . . . .       | 4 |
|       | Exemple . . . . .                              | 4 |
| 0.2.2 | 2.b) . . . . .                                 | 4 |
|       | Instructions . . . . .                         | 5 |
|       | Justification de l'instruction . . . . .       | 5 |
|       | Exemple . . . . .                              | 6 |
| 0.2.3 | 2.c) . . . . .                                 | 6 |
|       | Instructions . . . . .                         | 7 |
|       | Présentation . . . . .                         | 7 |
| 0.3   | Question 3 . . . . .                           | 7 |
| 0.3.1 | a) . . . . .                                   | 7 |
| 0.3.2 | Instructions BSRR . . . . .                    | 7 |
| 0.3.3 | Justification de l'instruction BSRR . . . . .  | 7 |
| 0.3.4 | Instructions ODR . . . . .                     | 8 |
| 0.3.5 | Justification de l'instruction ODR . . . . .   | 8 |
| 0.3.6 | b) . . . . .                                   | 8 |
| 0.3.7 | Justification de l'instruction input . . . . . | 9 |
| 0.3.8 | c) Code final . . . . .                        | 9 |

## 0.1 Question 1

### 0.1.1 1.a

Les broches LEDs

- LED\_RED : **OUT OUTPUT** sur la broche **PIN PB4**
- LED\_BLUE : **OUT OUTPUT** sur la broche **PIN PA9**
- LED\_GREEN : **OUT OUTPUT** sur la broche **PIN PC7**

Les broche des interrupteurs

- SW\_UP : **IN INPUT** sur la broche **PIN PA4**
- SW\_DOWN : **IN INPUT** sur la broche **PIN PB0**
- SW\_LEFT : **IN INPUT** sur la broche **PIN PC1**
- SW\_RIGHT : **IN INPUT** sur la broche **PIN PC0**
- SW\_CENTER : **IN INPUT** sur la broche **PIN PB5**

### 0.1.2 1.b

Étant donné que les LEDs sont configurées en mode **anode commune**, il convient de mettre un niveau logique bas pour activer ces dernières.

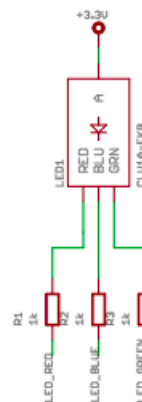


FIGURE 1 – Schéma des LEDs

Un niveau logique haut ne crée pas de différence de potentiel aux bornes des LEDs, de ce fait, aucun courant ne circule.

### 0.1.3 1.c

Lors d'un appui sur un interrupteur, le niveau logique associé est un niveau haut (3.3V).

Ce niveau se justifie par le type de montage. En effet, nous distinguons un montage en mode **Pull-down** avec la résistance à la masse, ce qui implique que lors d'un appui, le courant circule dans la résistance et toute la tension est au borne de la résistance.

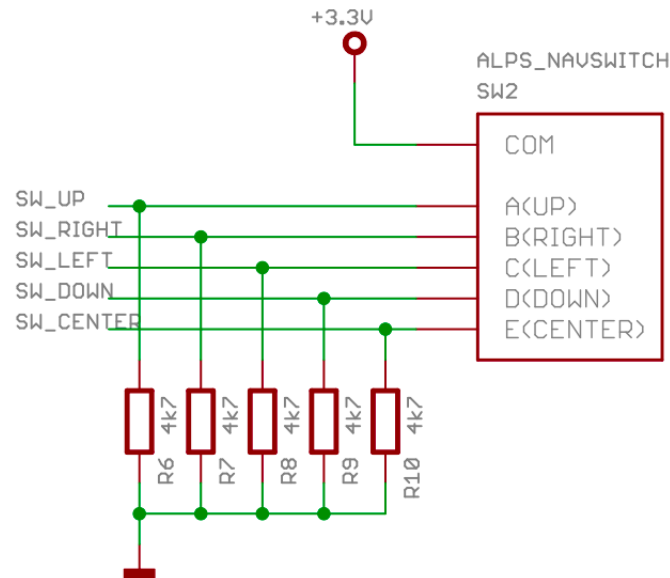


FIGURE 2 – Schéma des interrupteurs

## 0.2 Question 2

### 0.2.1 2.a)

#### Instruction

L'adresse du registre `REG` `RCC_AHB1ENR` est `ADR` `0x40023830`

#### Instruction

L'instruction est la suivante :

```
#define PORT_C 2

RCC->AHB1ENR |= 1u << PORT_C;
```

Affectation de l'horloge au port C

### Justification de l'instruction

Le registre **RCC\_AHB1ENR** est de la forme suivante (Avec GPIOAEN le bit de poids faible) :

|                 |                |                |                |                |
|-----------------|----------------|----------------|----------------|----------------|
| <b>RESERVED</b> | <b>GPIODEN</b> | <b>GPIOCEN</b> | <b>GPIOBEN</b> | <b>GPIOAEN</b> |
| X X X X X X     | X              | X              | X              | X              |

Pour synchroniser l'horloge au port voulue, il suffit de mettre à 1 le bit du port C.

Pour cela, on commence par placer un '1' devant le bon bit, dans notre cas sur **GPIOCEN**. On utilise l'opérateur de décalage à gauche.

$1 \ll 2$  revient à écrire **100** puis en faisant ou **OU** logique, cela permet de passer le bit à 1 si ce dernier est à 0 et de ne rien changer si il est déjà à 1.

Le fait de faire un **OU** permet de ne pas affecter les autres bits du registre, ce qui est souhaité.

### Exemple

Prenons en considération les 4 premiers bits de notre registre **RCC\_AHB1ENR** et observons le résultat avec notre opération.  
Ici, le port B et A sont déjà activés (LSB).

| Opérateur | Données | Information           |
|-----------|---------|-----------------------|
|           | 0011    | (RCC_AHB1ENR tronqué) |
|           | 0100    | ( $1 \ll pin$ )       |
| =         | 0111    | (RCC_AHB1ENR tronqué) |

On constate que le port C est bien relié à l'horloge et que les autres ports n'ont pas été affectés.

### 0.2.2 2.b)

- **GPIOC\_MODER** : Ce registre permet de définir le mode de la broche. Il existe 4 modes.
  - 00 : entrée (mode par défaut)
  - 01 : sortie
  - 10 : broches alternatives (SPI, I2C, UART...)
  - 11 : analogique
- **GPIOC\_PUPDR** : Ce registre permet de définir les éventuelles résistances de rappel. Il existe 4 modes.
  - 00 : aucune résistance de rappel

- 01 : mode Pull-Up
- 10 : mode Pull-Down
- 11 : mode réservé

Dans notre cas, les broches  PC0 et  PC1 sont en entrée **sans résistance de pull-up/pull-down**.

## Instructions

```
int pc0 = 0;           //Broche associée au port
int pc1 = 1;           //Broche associée au port


int upDown = 0b00;     //Aucune résistance de rappel
int input = 0b00       //broche en entrée

//Etat de la broche
GPIOC->MODER=GPIOC->MODER & ~(0b11 << (pc0*2) ) | input << (pc0*2);
GPIOC->MODER=GPIOC->MODER & ~(0b11 << (pc1*2) ) | input << (pc1*2);

//pull-up/pull-down ?
GPIOC->PUPDR=GPIOC->PUPDR & ~(0b11 << (pc0*2) ) | upDown << (pc0*2);
GPIOC->PUPDR=GPIOC->PUPDR & ~(0b11 << (pc1*2) ) | upDown << (pc1*2);
```

Configurations des broches en entrée

## Justification de l'instruction

Le registre  GPIOC\_MODER est de la forme suivante, de manière tronquée (Avec MODER0 les 2 bits de poids les plus faibles) :

|        |        |        |        |
|--------|--------|--------|--------|
| MODER3 | MODER2 | MODER1 | MODER0 |
| XX     | XX     | XX     | XX     |

Où **XX** représente l'état de la broche.

On souhaite mettre nos deux broches en entrée. Pour cela, on va utiliser l'opérateur de décalage pour sélectionner le duo de bits voulus (en fonction du numéro de la broche).

Cependant, on souhaite écraser la paire de bits car si on utilise la technique précédente, si on veut passer du mode **11** au mode **00**<sup>1</sup>, le bit de droite ne sera pas affecté.

1. Ce mode ne sera pas utilisé mais cela permet de rendre l'opération générique pour tous les modes

On va donc réinitialiser les deux bits en faisant un complément de la valeur **0x3** avec un décalage de 2 fois le numéro de la broche.

Ce coefficient est justifié par le fait que l'état de chaque broche est défini sur 2 bits.

Ensuite, il ne nous reste plus qu'à faire un **ET** avec le registre pour ne pas modifier les autres broches puis faire un **OU** avec notre mode souhaité. Il est impératif que le mode soit exprimé en valeur hexadécimal.

### Exemple

Prenons en considération les 8 premiers bits de poids faibles (4 premières broches du port) de notre registre `REG GPIOC_MODER` et observons le résultat avec notre opération.

On souhaite mettre la broche 4 au mode **00**

| Opérateur | Données  | Information         |
|-----------|----------|---------------------|
|           | 10110011 | (GPIOC_MODER)       |
| &         | 00111111 | not(0b11 << 2* pin) |
| =         | 00110011 |                     |
|           | 00000000 | (0b00 << 2* pin)    |
| =         | 00110011 | (GPIOC_MODER)       |

Le registre `REG GPIOC_PUPDR` est modifié de la même façon que le registre `REG GPIOC_MODER`.

### 0.2.3 2.c)

- `REG GPIOC_MODER` : Voir question précédente
- `REG GPIOC_OTYPER` : Ce registre permet de définir le type de configuration de sortie. Il existe 2 modes.
  - 0 : sortie push pull
  - 1 : sortie à drain ouvert
- `REG GPIOC_OSPEEDR` : Ce registre permet de définir la vitesse des broches de sortie. Il existe 4 modes.
  - 00 : Vitesse faible
  - 01 : Vitesse intermédiaire
  - 10 : Vitesse élevée
  - 11 : Vitesse maximale

## Instructions

```
int pc7 = 0;    //Broche associée au port

int output_type = 0x0;    //sortie push-pull
int speed = 0b01;        //vitesse medium
int upDown = 0b00;        //Aucune résistance de rappel

//Type de sortie
GPIOC->OTYPER = GPIOC->OTYPER & ~(0b1 << pc7 ) | output_type << pc7;
//Vitesse
GPIOC->OSPEED=GPIOC->OSPEED & ~(0b11 << (pc7*2) ) | speed << (pc7*2);
//Pullup-down ?
GPIOC->PUPDR=GPIOC->PUPDR & ~(0b11 << (pc7*2) ) | upDown << (pc7*2);
```

Configurations des paramètres des sorties

## Présentation

Les registres `REG GPIOC_OTYPE` et `REG GPIOC_SPEEDR` sont modifiés de la même façon que le registre `REG GPIOC_MODER` si ce n'est que le registre `REG GPIOC_OTYPER` ne nécessite pas de décaler de deux fois le numéro de la broche car chaque information sur le type de sortie est stockée dans un bit et non deux.

## 0.3 Question 3

### 0.3.1 a)

### 0.3.2 Instructions BSRR

```
#define pin 7 //Broche de la led sur le port C
void green_led(uint32_t on) {

    GPIOC->BSRR = (on)? 1u << pin : 0b1 << (16u)<<pin;

}
```

Instruction BSRR

### 0.3.3 Justification de l'instruction BSRR

Ce registre utilise les 16 premiers bits de poids les plus faibles pour mettre la sortie à 1 et les 16 bits suivants pour mettre la sortie à 0.



Lorsque **on** est à 0, on veut donc écrire dans le bit BR7 qui force la broche à 0. D'où le décalage de la valeur 1 de 16 bits puis du décalage de la pin.

On peut écrire la valeur brute dans le registre sans faire de masque dans la mesure où les changements d'états se font dès qu'un **1** est présent. Au tour d'horloge suivant, les bits sont réinitialisés à 0.

Lorsque **on** est à 1, on veut donc écrire dans le bit BS7 qui force la broche à 1. D'où le décalage de la pin uniquement.

### 0.3.4 Instructions ODR

```
#define pin 7 //Broche de la led sur le port C
void green_led(uint32_t on) {

    GPIOC->ODR = (on) ? GPIOC->ODR | (0b1 << pin) : GPIOC->ODR & ~(0b1
    << pin);

}
```

Instruction ODR

### 0.3.5 Justification de l'instruction ODR

Ce registre utilise les 16 premiers bits de poids les plus faibles pour mettre la sortie à 1 ou 0.

Lorsque **on** est à 1, on veut donc écrire dans le bit ODR7 qui force la broche à 0. D'où le décalage de la valeur 1 de la pin puis le **OU** pour ne pas affecter les autres sorties.

Lorsque **on** est à 0, on veut donc écrire dans le bit ODR7 qui force la broche à 0.

Il suffit de réinitialiser le bit en le mettant à 0 avec un décalage du port d'un bit complémenté et en faisant un **ET** avec la valeur actuelle du registre.

### 0.3.6 b)

```
uint32_t input() {

    uint32_t output;

    output = GPIOC->IDR & (0b1 << 0); //Right
    output |= _GPIOC->IDR & (0b1 << 1); //Left

    return output;

}
```

}

Instruction input

### 0.3.7 Justification de l'instruction input

Le registre REG `GPIOC->IDR` est accessible en lecture seule.

Il suffit de faire un **ET** avec la valeur 1 décalé de la valeur de la broche et de mettre le résultat dans une variable.

On effectue la même opération si ce n'est que l'on fait un **OU** avec la variable **output** afin de ne pas écraser le bit lu précédemment.

### 0.3.8 c) Code final

```
#define LED 7

void green_led(uint32_t on){

    GPIOC->BSRR = (state)? 0b1 << pin : 0b1 << (16u) << 7;

} //End green_led

#define SW_RIGHT (1u)
#define SW_LEFT (1u<<1)

uint32_t input(){

    uint32_t output;
    output = _GPIOC->IDR & (0b1)); //Right
    output |= _GPIOC->IDR & (0b1 << 1); //Left

    return output;

} //End input

int main(){

    _RCC->AHB1ENR |= RCC_AHB1ENR_GPIOCEN; //Clock

    //Settings LED
    //Output
    GPIOC->MODER=GPIOC->MODER & ~(0b11 << (LED*2) ) | 0b01 << (LED*2);
    //Type of output
```

```
GPIOC->OTYPER = GPIOC->OTYPER & ~(0b1 << LED ) | 0b0 << LED;
//Medium speed
GPIOC->OSPEED=GPIOC->OSPEED & ~(0b11 << (LED*2) ) | 0b01 << (LED*2)
;
//no Pullup-down
GPIOC->PUPDR=GPIOC->PUPDR & ~(0b11 << (LED*2) ) | 0b00 << (LED*2);

//Settings buttons SW_LEFT (PC1) and SW_RIGHT (PC0)
//Right button
GPIOC->MODER=GPIOC->MODER & ~(0b11 << (0*2) ) | 0b00 << (0*2);
GPIOC->PUPDR=GPIOC->PUPDR & ~(0b11 << (0*2) ) | 0b00 << (0*2);
//Left buttons
GPIOC->MODER=GPIOC->MODER & ~(0b11 << (1*2) ) | 0b00 << (1*2);
GPIOC->PUPDR=GPIOC->PUPDR & ~(0b11 << (1*2) ) | 0b00 << (1*2);

while (true){

    uint32_t button_value = input();

    if(button_value & (1u)){

        green_led(0);

    }//End if

    else if(button_value & (1u << 1) ){

        green_led(1u);

    }//End else if

} //End while

return 0;
} //End main
```

Code final