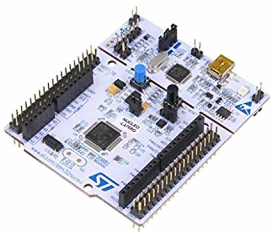

# Microprocesseurs

**Principes**

Nicolas LE GUERROUE

21 mars 2021

# Table des matières

# Conversion des nombres

## Coder un nombre dans la norme IEEE754

A travers un exemple, nous allons détailler la conversion de deux nombre dans la norme IEEE754.
Ces deux nombres seront 75.375 et $-75.375$. Nous allons voir que la technique pour coder les nombres positifs ou négatifs est identique.

### Signe du nombre

Le premier bit de la trame IEEE754 correspond au signe du nombre à coder.

- 0 si le nombre est **positif**
- 1 si le nombre est **négatif**

Dans notre cas (75.375), le premier bit vaut BIN 0

### La mantisse

La mantisse correspond à la valeur significative du nombre.
Nous allons normaliser la partie entière de la mantisse afin d'avoir un nombre de la forme $n = 1.M \cdot 2^E$

avec

- $E$ l'exposant
- $M$ la partie décimale de la mantisse.

#### La partie entière

On convertit la **valeur absolue** de la partie entière en binaire. Pour cela, on peut utiliser la technique des puissances de 2.
Dans 75, il y a $1 \cdot 2^6, 1 \cdot 2^3, 1 \cdot 2^1 et 1 \cdot 2^0$

On a donc $(75)_{10} =$ BIN 1001011

#### La partie décimale

On effectue la technique des multiplications par deux.

- On prend notre partie décimale que l'on multiplie par deux.

  $0.375 \cdot 2 = \boxed{0}.75$

- Ensuite, on prend la partie entière du résultat que l'on met de coté.

  La partie décimale est mise à la ligne et est de nouveau multipliée par deux.

  $0.75 \cdot 2 = \boxed{1}.5$

  *On exécute cet algorithme tant que le résultat est différent de 1.*

$0.5 \cdot 2 = \boxed{1}$

Ici, le résultat vaut 1, la conversion est terminée.
Les parties entières obtenues sont classées par pondérations décroissantes, c'est à dire que la première partie entière représente le bit de poids fort de la partie décimale.

Ainsi, $(0.375)_{10} = \boxed{\text{BIN}\ 011}$

## Normalisation de la mantisse

Le nombre 75.375 codé de façon **non normalisée** vaut $\boxed{\text{BIN}\ 1001011{,}011}$
Nous allons normaliser ce nombre.

Pour cela, on décale la virgule de $n$ emplacement(s) vers la gauche afin d'obtenir un seul **'1'** à gauche de la virgule.

Pour un décalage de $n$ emplacement(s), on multiplie le nombre binaire par $2^n$

Retenir l'exposant $\boxed{n}$ pour la suite.

$\boxed{\text{BIN}\ 1001011{,}011}$ normalisée vaut donc $\boxed{1}, 001011011 \cdot 2^6$

## Gestion de l'exposant

### Biais de l'exposant

Pour un format simple précision 32 bits, il convient d'ajouter à l'exposant la valeur $127(2^7 - 1)$
Notre exposant normalisé vaut donc $6 + 127 = (133)_{10} = \boxed{\text{BIN}\ 10000101}$

## Mise en place des composantes de la trame

Nous avons nos trois éléments : **le signe, la partie décimale de la mantisse et l'exposant normalisés**

On place d'abord le bit de signe puis l'exposant sur 8 bit et enfin la partie décimale de la mantisse sur 23 bits.

Notre partie décimale de la mantisse vaut donc BIN 001011011. Pour mettre sur 23 bit (protocole), il suffit de compléter avec autant de 0 nécéssaires en bit de poids faibles.

La partie décimale de la mantisse sur 23 bits vaut donc BIN 00101101100000000000000

Le nombre binaire vaut donc :

$N_{IEEE} = 0\,1\,0\,0\,0\,0\,1\,0\,1\,0\,0\,1\,0\,1\,1\,0\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0$

Avec

Le signe

L'exposant

La partie décimale de la mantisse

## Conversion en hexadécimal

On regroupe les bits par paquets de 4 et on convertit les paquets en hexadécimal.

$N_{IEEE} = 0\,1\,0\,0\;0\,0\,1\,0\;1\,0\,0\,1\;0\,1\,1\,0\;1\,1\,0\,0\;0\,0\,0\,0\;0\,0\,0\,0\;0\,0\,0\,0$

On obtient au final :

$(+75.375)_{10} =$ HEX 0x4296C000

## Cas des valeurs négatives

Pour les nombres négatifs, seuls le bit de signe change dans la trame sur 32 bits.

$N_{IEEE} = 1\,1\,0\,0\;0\,0\,1\,0\;1\,0\,0\,1\;0\,1\,1\,0\;1\,1\,0\,0\;0\,0\,0\,0\;0\,0\,0\,0\;0\,0\,0\,0$

D'où :

$(-75.375)_{10} =$ HEX 0xC296C000

# Question 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

## 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.

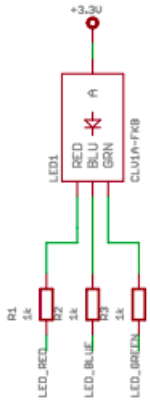


Figure 2.1 – Schéma des LEDs

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

### 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.

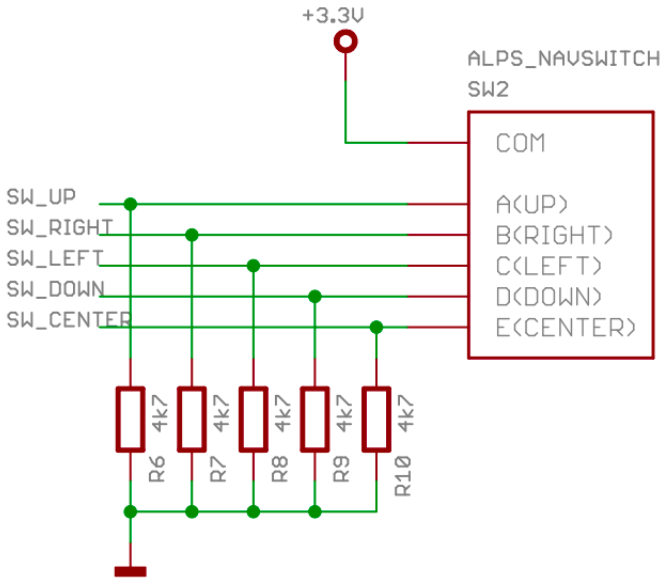


FIGURE 2.2 – Schéma des interrupteurs

## Question 2

### 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 REG RCC_AHB1ENR est de la forme suivante (Avec GPIOAEN le bit de poids faible) :

| RESERVED | GPIODEN | GPIOCEN | GPIOBEN | GPIOAEN |
|---|---|---|---|---|
| XXXXXX | 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 << 2 revient à écrire BIN 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 déja 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 REG RCC_AHB1ENR et observons le résultat avec notre opération.
Ici, le port B et A sont déja activés (LSB).

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

## 2.b)

- REG 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
- REG GPIOC_PUPDR : Ce registre permet de définir les éventulles 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 PIN PC0 et PIN 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 << (pc0*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 REG 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 represente 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** [1], le bit de droite ne sera pas affecté.

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.

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

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.

## 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 elevée
  - 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 nécéssite 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.

# Question 3

## a)

## Instructions BSRR

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

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

}
```

Instruction BSRR

## 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 **state** est à 0, on veut donc écrire dans le bit BR7 qui force la broche à 0. D'ou 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 **state** est à 1, on veut donc écrire dans le bit BS7 qui force la broche à 1. D'ou le décalage de la pin uniquement.

## Instructions ODR

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

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

}
```

Instruction ODR

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

Lorsque **state** 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.

## b)

```
uint32_t input() {

    uint32_t output;

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

    return output;
}
```

Instruction input

## 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 résulat 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.

## c) Code final

```
#define LED 7

void green_led(uint32_t state){

  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