

# STM32 GPIO



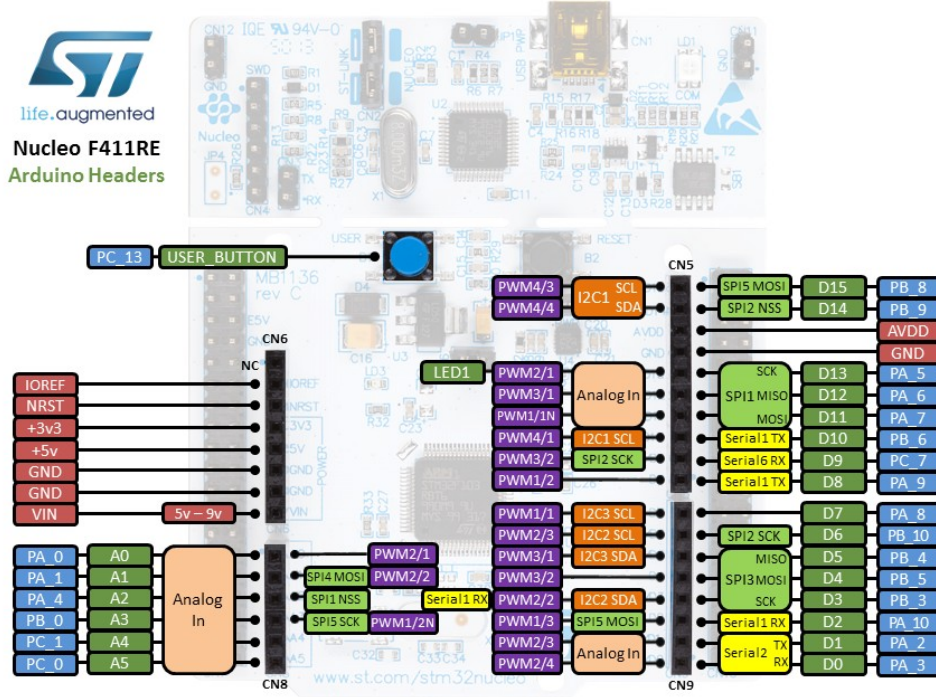
**Objectifs :** Mises en œuvre des GPIO. ( General Purpose Input Output).

- nommage
- pilotage des GPIO en entrée et en sortie

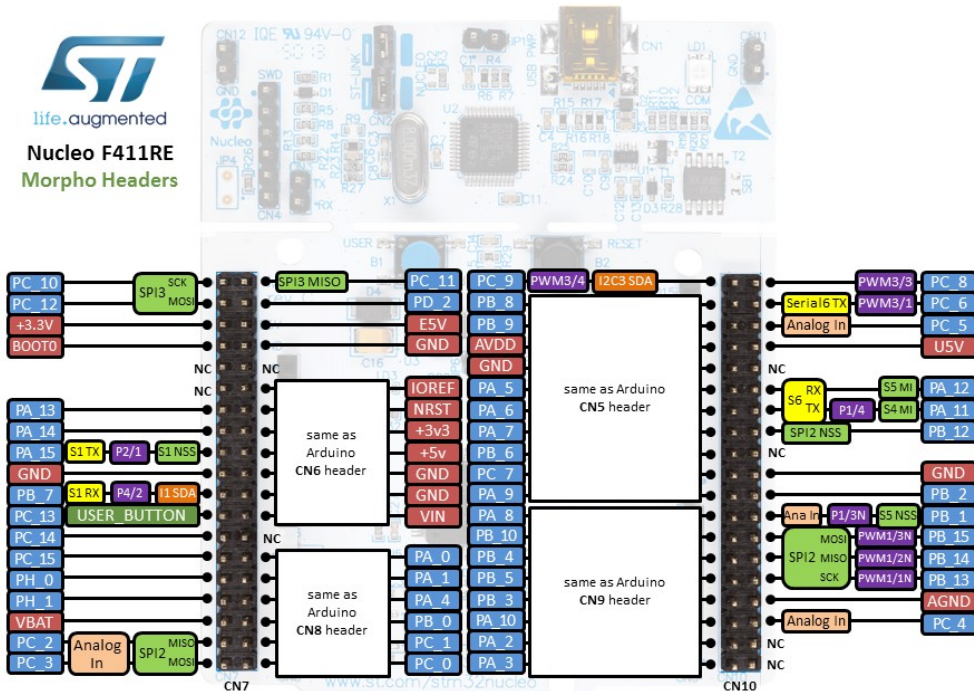
**Matériel :** Ce TP utilise une NUCLEO-F411RE, mais n'importe quelle autre carte NUCLEO convient.

**Logiciel :** MBED

Les GPIO sur carte NUCLEO (exemples sur F411RE)  
Connecteurs "Arduino"



Connecteurs "Morpho"



Le micro-contrôleur STM32 F411RE possède 81 ports d'entrée/sortie

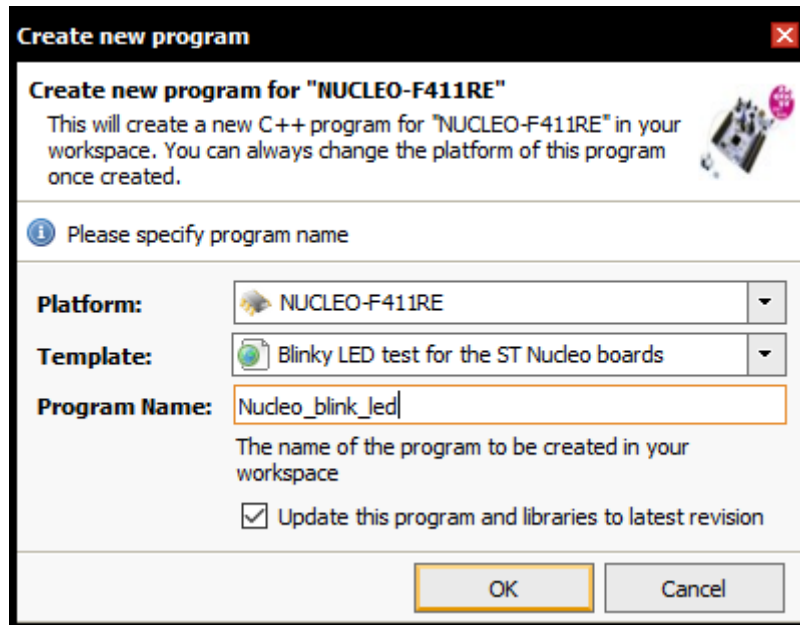
## STM32 GPIO



Ils ne sont pas tous accessibles sur les connecteurs de la carte NUCLEO (voir schémas ci-dessus). Les broches du microcontrôleur peuvent avoir plusieurs fonctions, c'est le cas par exemple des broches du port série 2 qui se superposent aux GPIO PA\_2 et PA\_3. *Ces derniers ne sont donc pas utilisables comme GPIO si le port série est activé.*

## Premier programme

Sur MBED, New Program



### Éditer main.c

```
#include "mbed.h"

DigitalOut myled(LED1);

int main() {
    while(1) {
        myled = 1; // LED is ON
        wait(0.2); // 200 ms
        myled = 0; // LED is OFF
        wait(1.0); // 1 sec
    }
}
```

Compiler, enregistrer le fichier .bin dans la carte Nucleo qui apparaît comme une clé USB. Le fichier .bin contient le code machine STM32. Après téléchargement, le code est exécuté automatiquement, la LED verte clignote suivant le programme.

### Analyse du programme :

MBED utilise un compilateur C++. C++ est un langage objet utilisant la syntaxe du C. MBED-OS est un ensemble (gigantesque) de bibliothèques (que l'on appelle classes en C++), pour en disposer il faut inclure mbed.h au programme. Pour utiliser une classe on crée un objet, instance de celle ci. Dans l'exemple ci-dessus, un objet myled, instance de la classe DigitalOut est créé sur le GPIO LED1. L'utilisation de myled est ensuite très simple. myled est un GPIO en sortie (DigitalOut).

## STM32 GPIO



Faire un clic-droit sur DigitalOut, une aide contextuelle apparaît:

The screenshot shows a context menu for the `DigitalOut` class. It lists three items:

- Function `DigitalOut` (public) with a source icon and a file icon. The class is `mbed::DigitalOut`. The signature is `DigitalOut(PinName pin)`.
- Function `DigitalOut` (public) with a source icon and a file icon. The class is `mbed::DigitalOut`. The signature is `DigitalOut(PinName pin, int value)`.
- Class `DigitalOut` with a source icon and a file icon. The namespace is `mbed`. The signature is `class DigitalOut {`.

On voit que l'on peut également initialiser le GPIO avec 1 ou 0 lors de l'instanciation.

### Équivalences de noms

Sur quel GPIO est connectée la LED1 ? Pour le savoir ....

Cette équivalence est déclarée dans la bibliothèque.

Pour connaître les équivalences (define) des GPIO de ce microcontrôleur cliquer en haut à droite sur le logo de la carte Nucleo:

NUCLEO-F411RE

Puis sur PinNames.h

The screenshot shows a pin list for the STM32 Nucleo. It includes a legend for I2C pins (XXX) and PWMOut pins (TIM n = Timer number, N = Inverted char). Below the legend, it says "You can find more details o dev library version):" and lists two links: [PeripheralPins.c](#) and [PinNames.h](#).

Un documentation complète est accessible en cliquant « more infos »

More Info

On trouve entre autre dans ce fichier :

```
// Generic signals namings
LED1      = PA_5,
LED2      = PA_5,
LED3      = PA_5,
LED4      = PA_5,
LED_RED   = LED1,
USER_BUTTON = PC_13,
// Standardized button names
BUTTON1 = USER_BUTTON,
SERIAL_TX  = PA_2,
SERIAL_RX  = PA_3,
USBTX     = PA_2,
USBRX     = PA_3,
I2C_SCL   = PB_8,
I2C_SDA   = PB_9,
SPI_MOSI  = PA_7,
SPI_MISO  = PA_6,
SPI_SCK   = PA_5,
SPI_CS    = PB_6,
PWM_OUT   = PB_3,
```

**LED1 est sur le GPIO PA\_5**  
**LED2, LED3, LED4, LED\_RED sont d'autres nom pour LED1.** (il peut y avoir plusieurs LED sur d'autres cartes Nucleo)

La fonction wait du programme réalise une temporisation, pour en savoir plus, **cliquez sur le mot wait** un menu contextuel s'ouvre, découvrez, wait\_ms, wait\_us, bonne lecture.



**Exercice 1** : Réaliser un programme faisant clignoter la LED1, l'état bas durera toujours une seconde, l'état haut sera variable. **Pour cela** :

Créer une variable réelle double  $t = 0.1$  qui servira pour la temporisation.

Dans la boucle sans fin, après chaque clignotement ajouter 100mS à la temporisation de l'état haut jusqu'à atteindre une seconde puis recommencer.

## GPIO en entrée


Rechercher l'aide contextuelle sur DigitalIn

Function **DigitalIn** (public)  

Class: [mbed::DigitalIn](#)

`DigitalIn(PinName pin)`


---

Function **DigitalIn** (public)  

Class: [mbed::DigitalIn](#)

`DigitalIn(PinName pin, PinMode mode)`


---

Class **DigitalIn**  

Namespace: [mbed](#)

`class DigitalIn {`


---

Function **DigitalInOut** (public)  

Class: [mbed::DigitalInOut](#)

`DigitalInOut(PinName pin)`



---

Function **DigitalInOut** (public)  

Class: [mbed::DigitalInOut](#)

`DigitalInOut(PinName pin, PinDirection direction, PinMode mode, int value)`

---

Class **DigitalInOut**  


Namespace: [mbed](#)

`class DigitalInOut {`

Cette classe permet de créer un objet GPIO en entrée.

La classe DigitalInOut regroupe les deux précédentes, nous ne l'utiliserons pas.

Un clic sur 'A' donne le code source de la classe, indisponible ici, par défaut seule la bibliothèque MBED compilée est dans le projet (cela est généralement suffisant)

Un clic sur le logo à côté du 'A'  donne le détail des paramètres et des exemples d'utilisation de la classe.



## Constructor & Destructor Documentation

### **DigitalIn** ( *PinName* *pin* )

Create a **DigitalIn** connected to the specified pin.

**Parameters:**

**pin** **DigitalIn** pin to connect to

Definition at line 58 of file [DigitalIn.h](#).

### **DigitalIn** ( *PinName* *pin*, *PinMode* *mode* )

Create a **DigitalIn** connected to the specified pin.

**Parameters:**

**pin** **DigitalIn** pin to connect to

**mode** the initial mode of the pin

Definition at line 68 of file [DigitalIn.h](#).

## Member Function Documentation

### **int** *is\_connected* ( )

Return the output setting, represented as 0 or 1 (int)

**Returns:**

Non zero value if pin is connected to uc GPIO 0 if gpio object was initialized with NC

Definition at line 99 of file [DigitalIn.h](#).

### **void** *mode* ( *PinMode* *pull* )

Set the input pin mode.

**Parameters:**

**pull** PullUp, PullDown, PullNone, OpenDrain

Definition at line 87 of file [DigitalIn.h](#).

On peut lire que lors de l'instanciation, il est possible de définir (mode) le type d'entrée.

**Exercice 2 :** Rechercher la signification de pullup, pulldown, pullnone, opendrain

**Exercice 3 :** A partir du programme précédent :

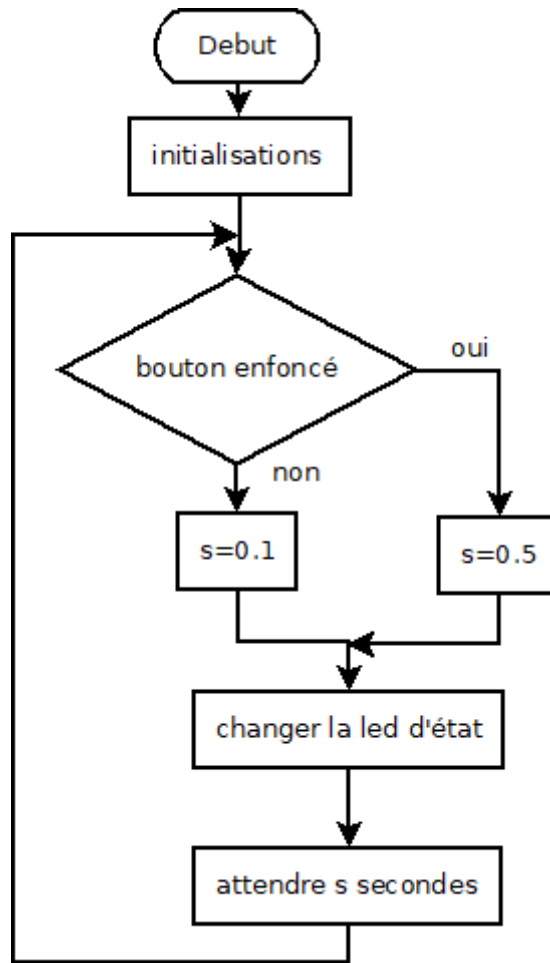
Créer un objet bouton sur le GPIO correspondant au bouton bleu de la carte Nucleo

Le programme allumera la LED1 lors de l'appui sur ce bouton.

Modifier ensuite le programme pour changer l'état de la LED à chaque relâchement du bouton.



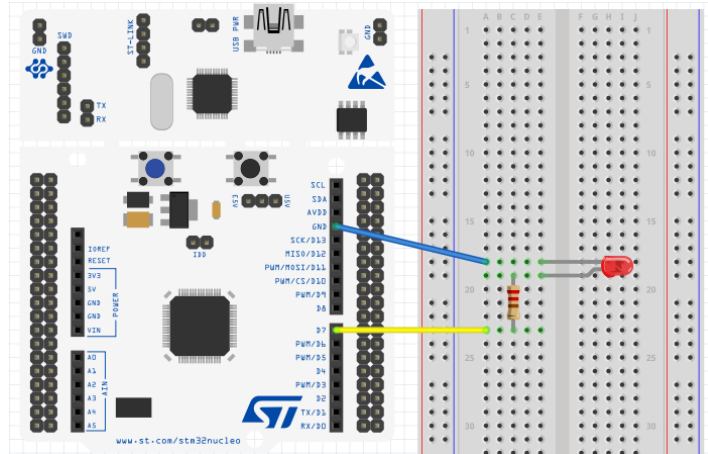
Exercice 4 : Réaliser le programme correspondant à l'algorithme ci dessous



# STM32 GPIO



**Exercice 5 :** Réaliser le montage ci-dessous et faire clignoter la LED sur le GPIO PA\_8.



## Exercice 6

Mbed offre la possibilité de créer un bus 16 bits à partir de bits de différents ports physiques. Dans la documentation Mbed rechercher « reference », « APIs », « Drivers » « BusOut ». A partir de la documentation de la classe BusOut, créer un bus 8bits à partir de GPIO disponibles sur les connecteurs Arduino. Câbler 8 Leds sur ce bus. Chaque LED sera placée en série avec une résistance de 220Ω à 270Ω Réaliser le chenillard ci-contre, avec une temporisation de 200mS entre deux allumages (utiliser <<)

```
00000001
00000010
00000100
00001000
00010000
00100000
01000000
10000000
```

## Exercice 7

Réaliser le chenillard suivant (utiliser << et une addition ou un "ou")

```
00000001
00000011
00000111
00001111
00011111
00111111
01111111
11111111
01111111
00111111
00011111
00001111
00000111
00000011
00000001
00000000
```

## Exercice 8

Reprendre l'exercice précédent, l'avancement d'un pas du chenillard s'effectuera lors de l'appui sur B1 (bouton bleu).

## Exercice 9

A l'aide d'un tableau de données, réaliser l'animation K2000 (ou n'importe laquelle)

```
00000000
10000001
11000011
11100111
11111111
01111110
00111100
00011000
```

## Exercice 10

Remplacer les LEDs par un afficheur 7 segments et réaliser un compteur hexadécimal des appuis sur B1

Le brochage est disponible sur le datasheet de l'afficheur fourni.

