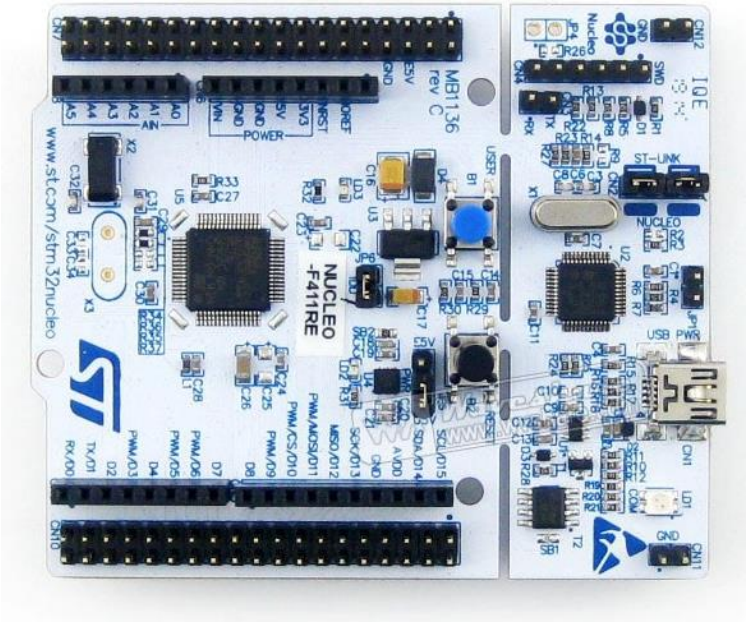
Recherche sur STM32

1- STM32CubeIDE

C’est l'interface de programmation (IDE) que nous utilisons pour programmer les microcontrôleurs de nos robots.

2- NUCLEO-F411RE Description

La carte STM32 Nucleo offre aux utilisateurs un moyen abordable et flexible d'essayer de nouvelles idées et de construire des prototypes avec n'importe quelle gamme de microcontrôleurs STM32, en choisissant parmi les différentes combinaisons de performances, de consommation d'énergie et de fonctionnalités. La prise en charge de la connectivité Arduino™ et les connecteurs ST Morpho permettent d'étendre facilement les fonctionnalités de la plateforme de développement ouverte STM32 Nucleo avec un large choix de shields spécialisés. La carte STM32 Nucleo ne nécessite pas de sonde séparée car elle intègre le débogueur/programmeur ST-LINK/V2-1. La carte STM32 Nucleo est livrée avec la bibliothèque complète de logiciels HAL STM32, ainsi qu'avec divers exemples de logiciels packagés et un accès direct aux ressources en ligne mbed. Traduit avec www.DeepL.com/Translator (version gratuite)



3- Caractéristiques principales du NUCLEO-F411RE

- Microcontrôleur STM32 avec boîtier LQFP64
  - STM32F411RET6 (512K FLASH)
- Deux types de ressources de vulgarisation
  - Connectivité Arduino Uno Revision 3
  - Connecteurs d'extension Morpho de STMicroelectronics pour un accès complet à toutes les E/S du STM32.
- mbed-enabled (mbed.org)
- Débogueur/programmeur ST-LINK/V2-1 embarqué avec connecteur SWD
  - pour utiliser le kit en tant que ST-LINK/V2-1 autonome.
- Alimentation de la carte flexible
  - USB VBUS ou source externe (3,3 V, 5 V, 7 - 12 V)
- Point d'accès à la gestion de l'énergie
- Trois LEDs
  - Communication USB (LD1), LED utilisateur (LD2), LED d'alimentation (LD3)
- Deux boutons poussoirs : USER et RESET
- Capacité de ré-énumération USB : trois interfaces différentes supportées sur USB
  - Port de communication virtuel
  - Mémoire de masse

- Port de débogage

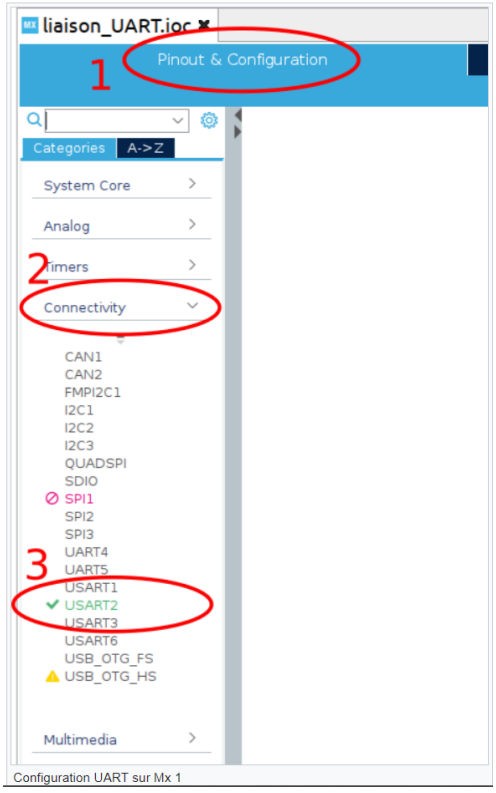
- Bibliothèque complète de logiciels libres HAL comprenant une variété d'exemples de logiciels
- Prise en charge par un large choix d'environnements de développement intégré (IDE), notamment IAR, Keil et les IDE basés sur GCC.

4- Configuration sur MX

Pour débuter, créer un nouveau projet comme décrit dans la section 2.2. (Cible STM32 NUCLEO-F446RE pour ce tuto).

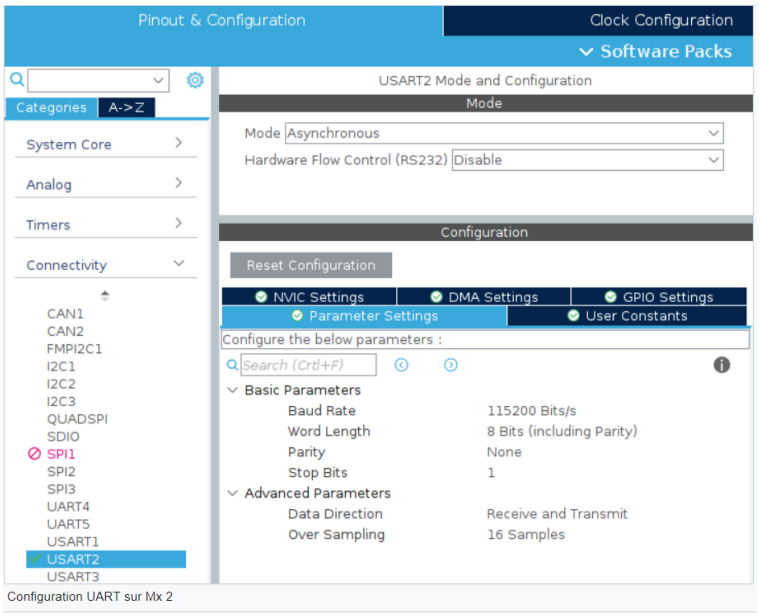
Pour plus de facilité, nous allons utiliser l'UART 2 car il s'agit de l'UART qui est aussi utilisé par le périphérique USB de votre carte.

Pour ce faire, vérifier bien que votre UART2 est enable. Pour cela cliquer sur Pinout and Configuration, puis sur l'onglet Connectivity du menu déroulant de gauche. Si l'UART 2 n'est pas enable alors vous avez soit choisi la mauvaise carte, soit choisi le processeur de la STM au lieu de sa board. Il faut relire la section 2.2 de ce tuto.



Configuration UART sur Mx 1

Ensuite cliquer sur UART2, pour le configurer.



Configuration UART sur Mx 2

Sur l'image ci-dessus, nous pouvons lire les informations suivantes:

-> baud rate correspond nombre de bits envoyer par seconde. Il s'agit de la vitesse de notre communication. Cette vitesse doit être la même sur les deux composants émetteur-récepteur de l'UART.

-> word length correspond au nombre de bits d'information envoyés par l'UART.

-> parity correspond au nombre de bit de parité, expliqué plus en haut. Il y est écrit none, cela signifie qu'il n'y a pas de bits de parité.

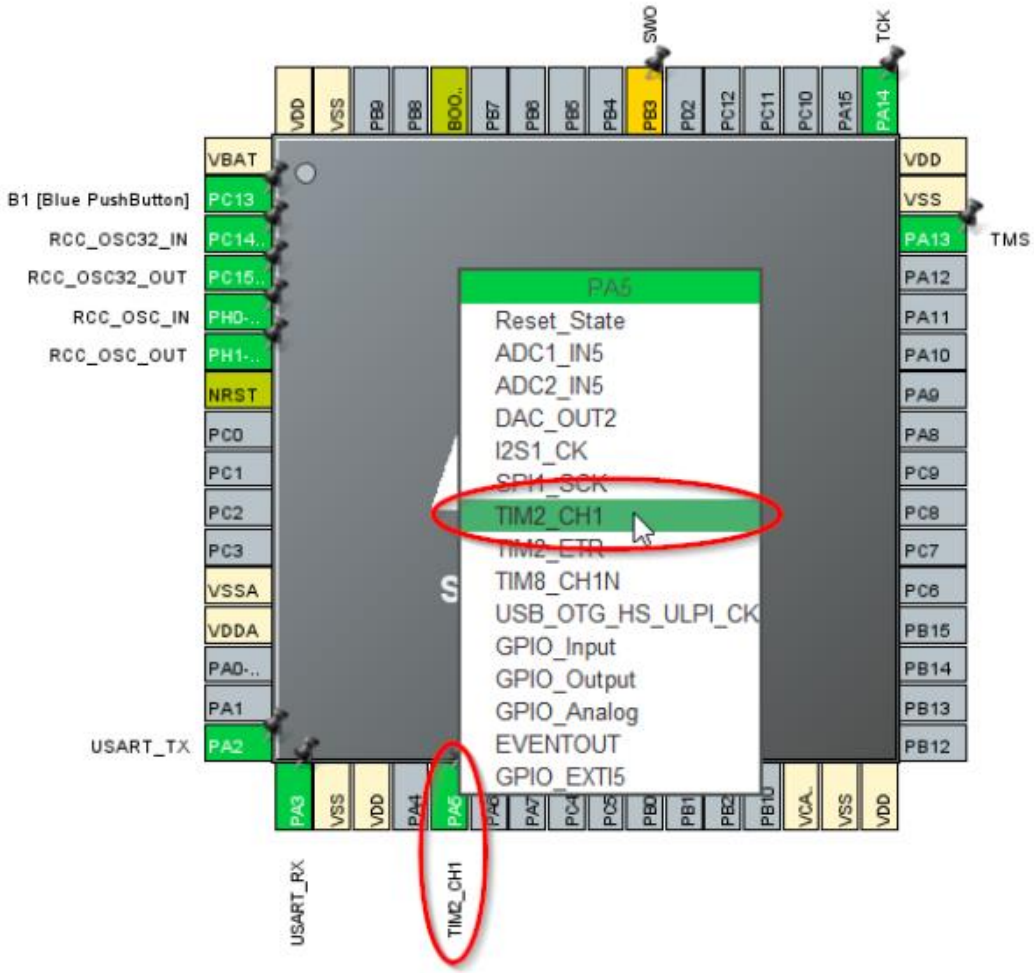
–> bit de stop correspond au nombre de bits utilisés pour déclarer la fin d'un caractère/word. Il est ici à 1. Pour l'instant, nous n'allons pas toucher ou modifier ces paramètres.

5- Configuration via CubeMX

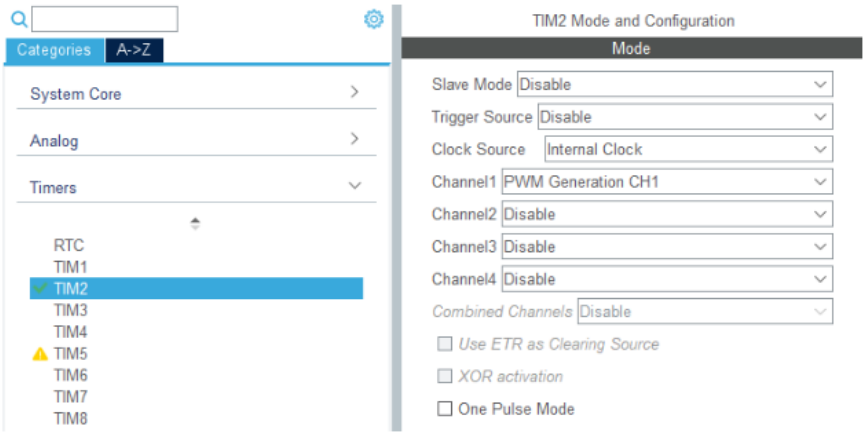
Avant de sauvegarder le fichier.ioc (fichier de config STM32CubeMX) et générer le code du projet, nous devons faire quelques configurations supplémentaires.

1- Nous choisissons le pin PA5 pour générer le PWM (pin connecté à la LED intégrée LD2). Le signal est généré par un timer et donc nous devons “connecter” le pin à ce périphérique. Pour ce faire, veuillez cliquer sur la pâte PA5 (dans la vue Pinout view). D'après le menu qui apparaît, ce pin peut être connecté au canal 1 du timer 2

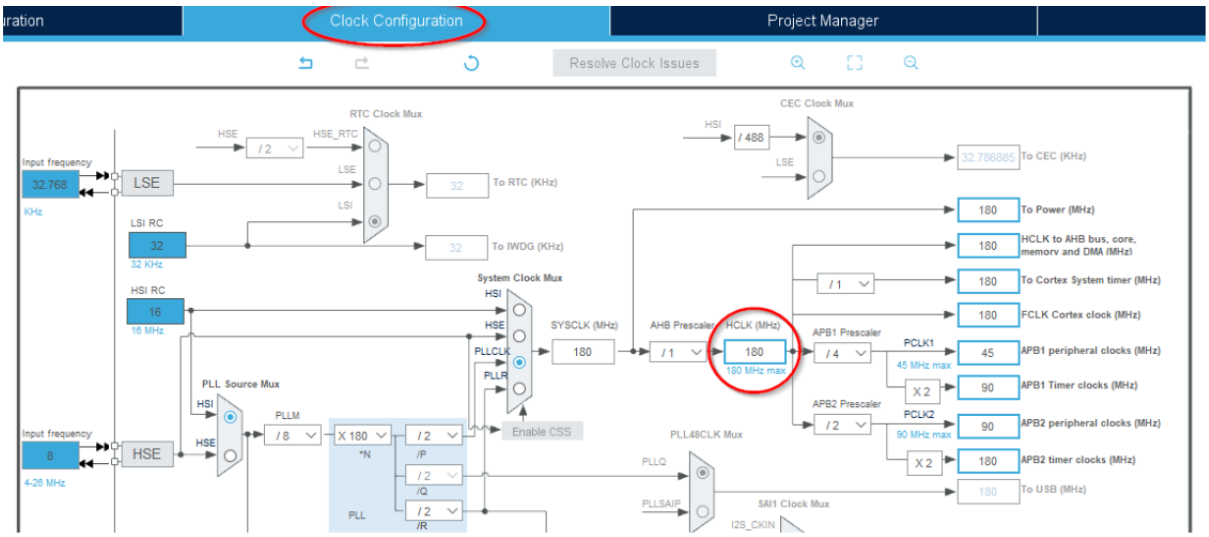
(TIM2_CH1) et au canal 1 du timer 8 (TIM8_CH1N). Nous choisissons TIM2_CH1 (cliquer pour activer)



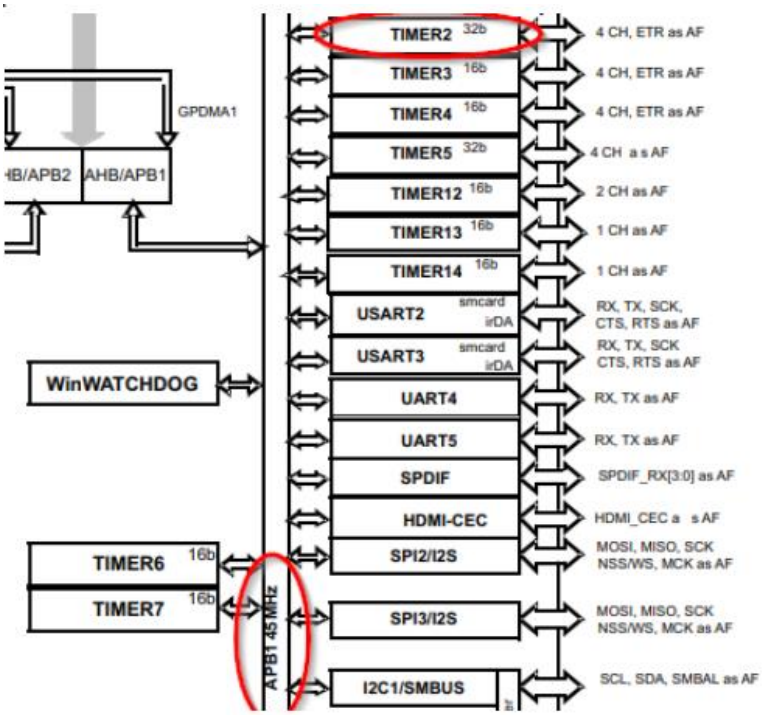
2- Ensuite, accéder à la catégorie Timers (menu à gauche) et sélectionner le TIM2. Activer le canal 1 en mode PWM Generation et choisir Internal Clock comme Clock Source.



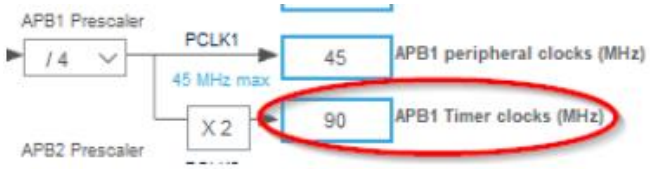
3- Maintenant, nous devons configurer les horloges du microcontrôleur. Accéder à l'onglet Clock Configuration. Puis enter 180 MHz comme fréquence de processeur (maximale pour ce microcontrôleur) et appuyer sur entrée. CubeMX règle automatiquement la fréquence des périphériques.



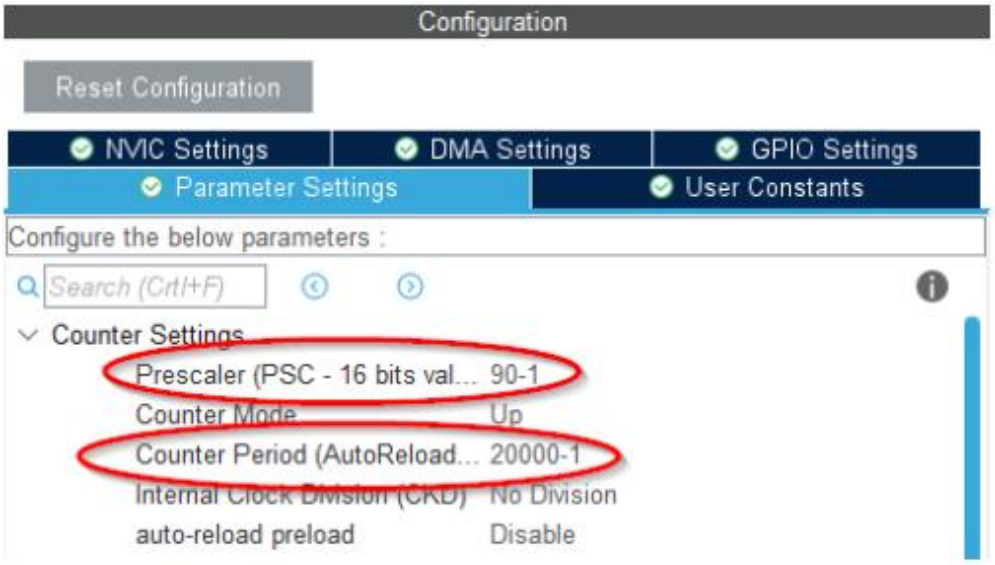
4- Le timer 2 est connecté au bus APB1.



Et d'après la vue précédente, les timers de APB1 tournent à 90 MHz (x2 de APB1 peripheral clocks).

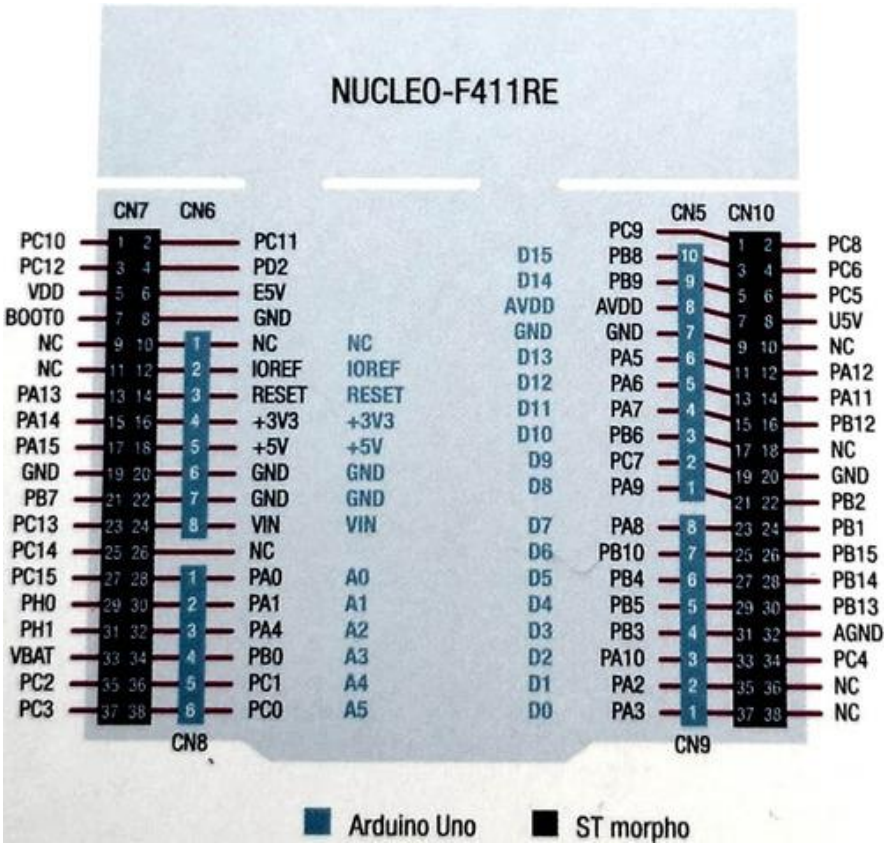


5- Revenant sur la configuration du TIM2, nous devons configurer le registres Prescaler et Counter Period pour régler la fréquence du signal généré d'après la formule suivante : f_PWM = f_APBx_timer/((PSC+1)*(CounterPeriod+1)). Pour arriver à une fréquence de 50 Hz, nécessaire pour piloter un servomoteur analogique, nous pouvons entrer les valeurs suivantes:



6- Pourquoi utiliser CubeIDE

Les MCU étant des ordinateurs minimalistes, ils ne possèdent pas de système d'exploitation (linux etc…). Il n'y en a aussi pas l'intérêt. De ce fait, les programmes au sein de MCU sont souvent codés, compilés puis téléversés par un ordinateur extérieur. Plusieurs solutions existent comme les systèmes arduino, le compilateur en ligne MBED ou bien Atollic. MBED et Arduino ont l'avantage de permettre une prise en main rapide. Cependant pour les prendre en main vous trouverez de nombreux tutos sur internet. Et ils ne sont pas utilisés à Polybot ! À Polybot, nous utilisons CubeIDE pour avoir une architecture logicielle commune. De plus CubeIDE permet d'utiliser des OS temps réel qui nous permette d'avoir un contrôle précis de ce que nous faisons. Enfin CubeIDE est une solution professionnelle utilisée en entreprise, ainsi prendre en main cette outil est un véritable atout pour votre CV.



7- Brochage de la carte

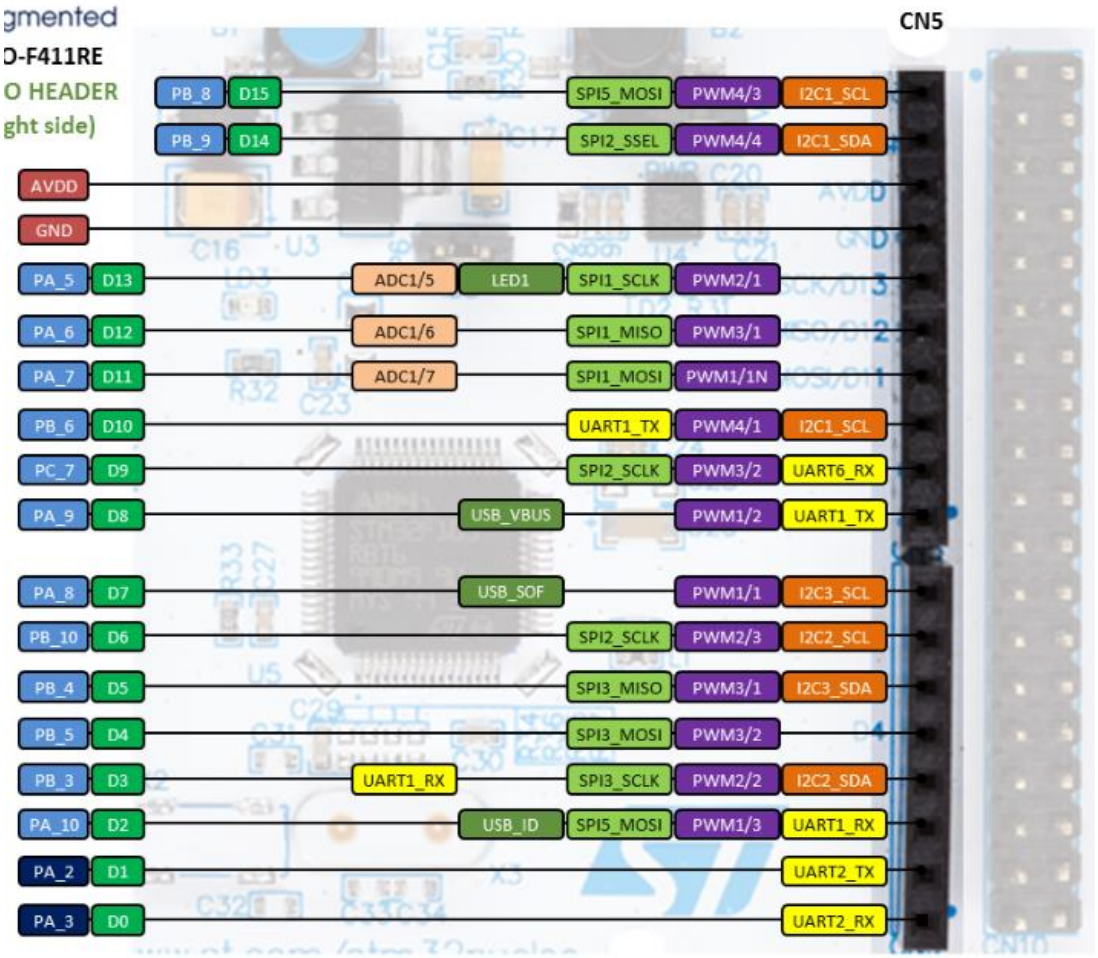
Code de couleur

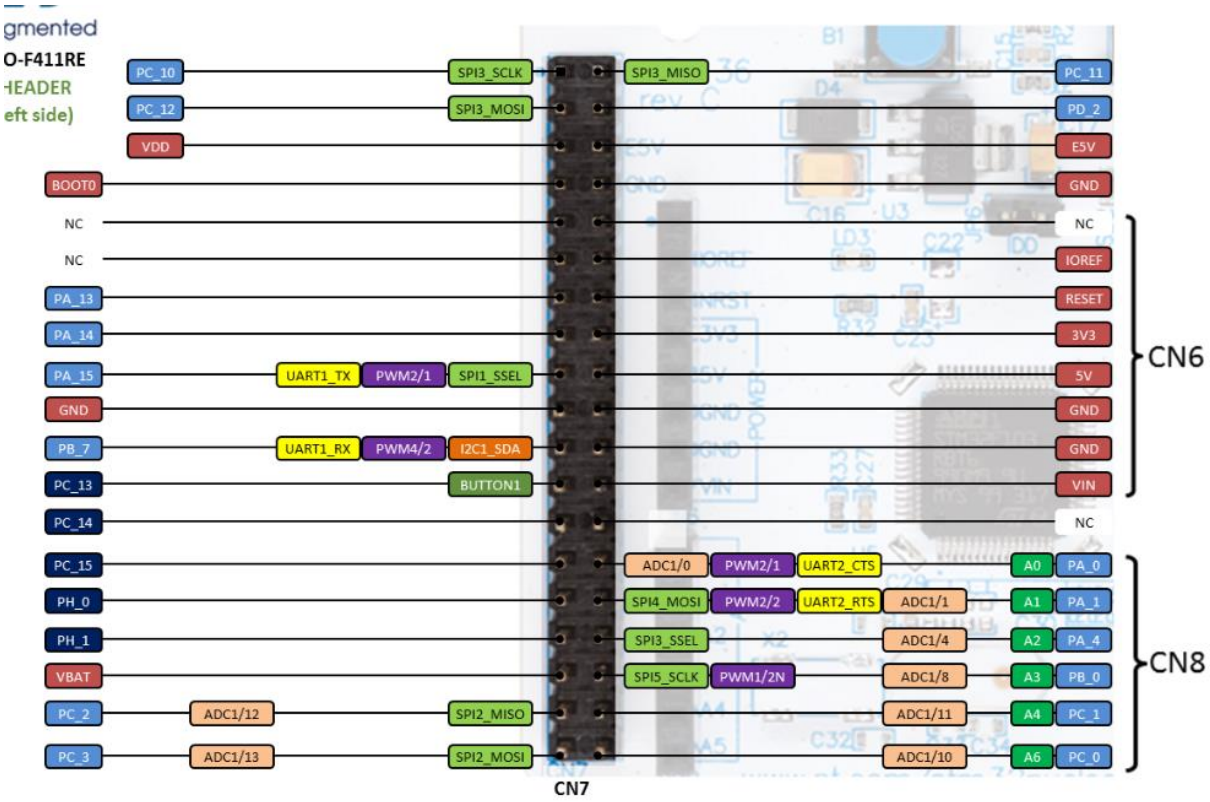
Labels usable in code

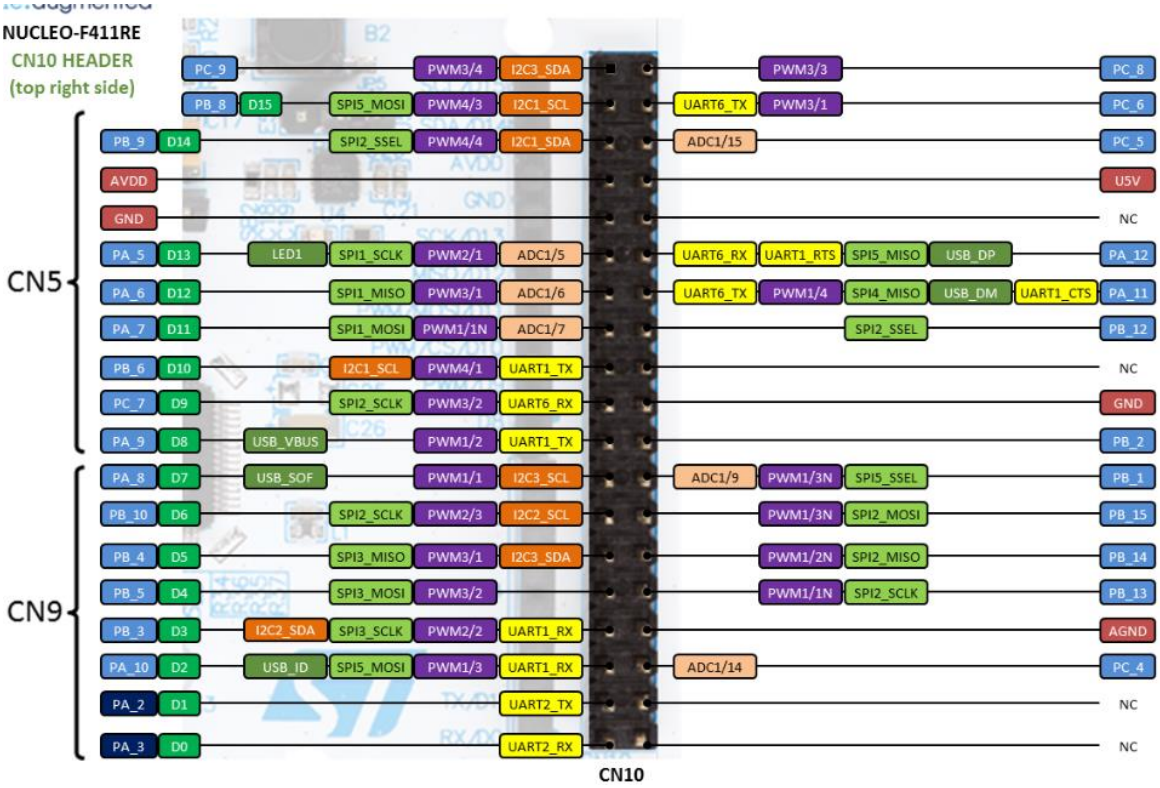
PX_Y	MCU pin without conflict	XXX	Arduino connector names (A0, D1, ...)
PX_Y	MCU pin connected to other components See **PeripheralPins.c** (link below) for more information	XXX	LEDs and Buttons (LED_1, USER_BUTTON, ...)

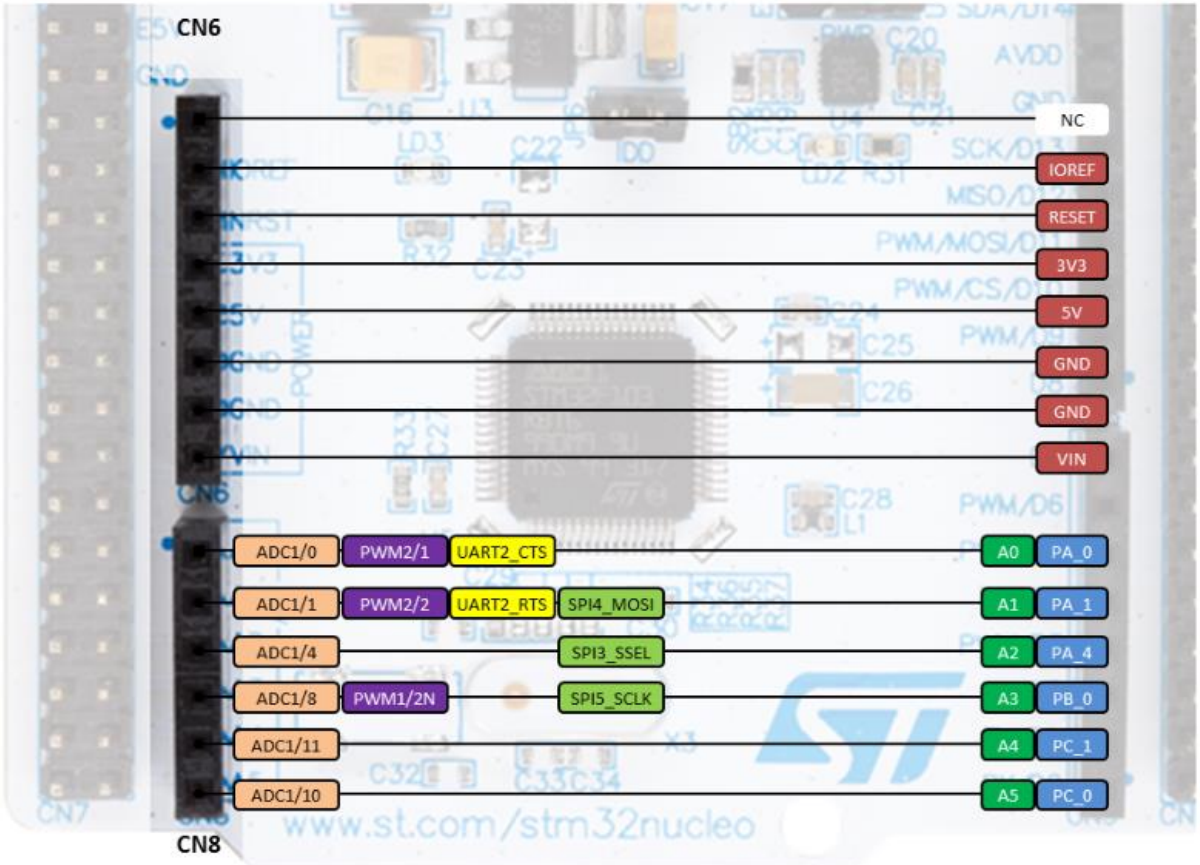
Labels not usable in code (for information only)

XXX	Serial pins (USART/UART)	XXX	AnalogIn (ADC) and AnalogOut pins (DAC)
XXX	SPI pins	XXX	CAN pins
XXX	I2C pins		
XXX	PWMOut pins (TIMER n/c[N]) n = Timer number c = Channel N = Inverted channel	XXX	Power and control pins (3V3, GND, RESET, ...)


gmented
O-F411RE
O HEADER
ght side)
CN5
PB_8 D15
PB_9 D14
AVDD
GND
PA_5 D13
PA_6 D12
PA_7 D11
PB_6 D10
PC_7 D9
PA_9 D8
PA_8 D7
PB_10 D6
PB_4 D5
PB_5 D4
PB_3 D3
PA_10 D2
PA_2 D1
PA_3 D0
ADC1/5
LED1
ADC1/6
ADC1/7
USB_VBUS
USB_SOF
USB_ID
SPI5_MOSI
SPI2_SSEL
SPI1_SCLK
SPI1_MISO
SPI1_MOSI
UART1_TX
SPI2_SCLK
SPI3_MISO
SPI3_MOSI
SPI3_SCLK
UART1_RX
PWM4/3
PWM4/4
PWM2/1
PWM3/1
PWM1/1N
PWM4/1
PWM3/2
PWM1/2
PWM1/1
PWM2/3
PWM3/1
PWM2/2
PWM1/3
I2C1_SCL
I2C1_SDA
UART6_RX
UART1_TX
I2C3_SCL
I2C2_SCL
I2C3_SDA
I2C2_SDA
UART2_TX
UART2_RX


gmented
O-F411RE
HEADER
eft side)
PC_10
PC_12
VDD
BOOT0
NC
NC
PA_13
PA_14
PA_15
GND
PB_7
PC_13
PC_14
PC_15
PH_0
PH_1
VBAT
PC_2
PC_3
SPI3_SCLK
SPI3_MOSI
UART1_TX
PWM2/1
SPI1_SSEL
UART1_RX
PWM4/2
I2C1_SDA
BUTTON1
ADC1/12
ADC1/13
SPI2_MISO
SPI2_MOSI
SPI3_MISO
ADC1/0
PWM2/1
UART2_CTS
SPI4_MOSI
PWM2/2
UART2_RTS
SPI3_SSEL
SPI5_SCLK
PWM1/2N
ADC1/1
ADC1/4
ADC1/8
ADC1/11
ADC1/10
PC_11
PD_2
E5V
GND
NC
IOREF
RESET
3V3
5V
GND
GND
VIN
NC
A0 PA_0
A1 PA_1
A2 PA_4
A3 PB_0
A4 PC_1
A6 PC_0
CN6
CN8
CN7


NUCLEO-F411RE
CN10 HEADER
(top right side)
CN5
CN9
CN10
PC_9 PWM3/4 I2C3_SDA PWM3/3 PC_8
PB_8 D15 SPI5_MOSI PWM4/3 I2C1_SCL UART6_TX PWM3/1 PC_6
PB_9 D14 SPI2_SSEL PWM4/4 I2C1_SDA ADC1/15 PC_5
AVDD 5V
GND NC
PA_5 D13 LED1 SPI1_SCLK PWM2/1 ADC1/5 UART6_RX UART1_RTS SPI5_MISO USB_DP PA_12
PA_6 D12 SPI1_MISO PWM3/1 ADC1/6 UART6_TX PWM1/4 SPI4_MISO USB_DM UART1_CTS PA_11
PA_7 D11 SPI1_MOSI PWM1/1N ADC1/7 SPI2_SSEL PB_12
PB_6 D10 I2C1_SCL PWM4/1 UART1_TX NC
PC_7 D9 SPI2_SCLK PWM3/2 UART6_RX GND
PA_9 D8 USB_VBUS PWM1/2 UART1_TX PB_2
PA_8 D7 USB_SOF PWM1/1 I2C3_SCL ADC1/9 PWM1/3N SPI5_SSEL PB_1
PB_10 D6 SPI2_SCLK PWM2/3 I2C2_SCL PWM1/3N SPI2_MOSI PB_15
PB_4 D5 SPI3_MISO PWM3/1 I2C3_SDA PWM1/2N SPI2_MISO PB_14
PB_5 D4 SPI3_MOSI PWM3/2 PWM1/1N SPI2_SCLK PB_13
PB_3 D3 I2C2_SDA SPI3_SCLK PWM2/2 UART1_RX AGND
PA_10 D2 USB_ID SPI5_MOSI PWM1/3 UART1_RX ADC1/14 PC_4
PA_2 D1 UART2_TX NC
PA_3 D0 UART2_RX NC


CN6
CN8
NC
IOREF
RESET
3V3
5V
GND
GND
VIN
ADC1/0 PWM2/1 UART2_CTS A0 PA_0
ADC1/1 PWM2/2 UART2_RTS SPI4_MOSI A1 PA_1
ADC1/4 SPI3_SSEL A2 PA_4
ADC1/8 PWM1/2N SPI5_SCLK A3 PB_0
ADC1/11 A4 PC_1
ADC1/10 A5 PC_0
www.st.com/stm32nucleo

Le code du « main.c ».

```
#include "main.h"
```

Le main.h regroupe d'autres inclusions, afin d' éviter de surcharger le main.c. Il contient aussi une macro de vérification d'inclusion.

```
static GPIO_InitTypeDef  GPIO_InitStruct;
```

En gros, on renomme la structure **GPIO_InitTypeDef** en **GPIO_InitStruct**, « static » afin que la structure ne soit modifiable que dans ce fichier (main.c).

```
static void SystemClock_Config(void) ;
```

Prototype de la fonction de configuration de l'horloge système. Observons maintenant ce que contient la function

```
HAL_Init();
```

Il sert à initialiser la librairie HAL.

```
/* -1- Enable GPIO Clock (to be able to program the configuration
registers) */
  LED3_GPIO_CLK_ENABLE();
```

Cela semble être un délai d'attente après l'initialisation de RCC (Reset and Clock Control).

```
  /* -2- Configure IO in output push-pull mode to drive external LEDs
*/
  GPIO_InitStruct.Mode  = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull  = GPIO_PULLUP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  GPIO_InitStruct.Pin = LED3_PIN;

  HAL_GPIO_Init(LED3_GPIO_PORT, &GPIO_InitStruct);
```

Ce block est très intéressant, on configure chaque éléments de la structure GPIO_InitTypeDef, puis on l'envoie en paramètre dans la fonction HAL_GPIO_Init();. GPIO_initTypeDef s' appel maintenant GPIO_InitStruct.

```
  while (1)
  {
    HAL_GPIO_TogglePin(LED3_GPIO_PORT, LED3_PIN);
    /* Insert delay 100 ms */
    HAL_Delay(100);
  }
}
```

La structure de contrôle **while (1){}** est l' équivalent de la fonction **void loop(){}** pour Arduino, c' est ici qu' on cuisine.

Commande de base pour les GPIOs avec HAL.

```
 HAL_Delay();
```

Quand à elle appartient à la librairie « hal » , dans le même répertoire . C'est l'équivalent de la fonction delay() ; d' Arduino. Donc des millisecondes.

```
HAL_GPIO_TogglePin () ;
```

C’est une fonction de la librairie « hal_gpio.h » que l'on trouve dans le répertoire /votre_carte/Drivers/Votre_Carte_HAL_Driver. Cette fonction inverse l' état de la sortie à chaque fois qu' elle est appelée.

```
HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
```

Lecture des entrées. GPIOx est le PORT, une lettre et GPIO_Pin le numéro de la broche concerné. La fonction renvoie la valeur « bitstatus » pouvant être GPIO_PIN_SETou GPIO_PIN_RESET.

```
HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState
PinState);
```

Voici les deux valeurs de PinState : GPIO_PIN_SET et GPIO_PIN_RESET.