TP 3 : Gestion des ports d'Entrée/Sortie

Objectifs :

-Comprendre le rôle des E/S
-Manipulation de périphérique : GPIO, Timer

Introduction

Dans le cadre de ce TP, on souhaite manipuler des périphériques du processeur, à savoir les GPIO (General Purpose Input Output) et le Timer. Tous les exercices se dérouleront par rapport à la carte Nucléo-64 STM32F446RE. L'environnement de programmation et de debug reste STM32CubeIDE.

A préparer :

Les broches d'E/S sont regroupées sur des ports et le processeur permet de mettre facilement à 0 ou 1 les broches qu'il contrôle via les registres de configuration d'un contrôleur GPIO (General Purpose Input Output).. Il existe 9 ports sur le microcontrôleurs STM32F4, donc 9 contrôleurs GPIO (GPIOA, GPIOB, …, GPIOI) et sur chaque port 16 broches, numérotées de 0 à 15.

Exercice 1 : GPIO – Programmation par registres

En Annexe 1, vous trouverez les étapes pour la création d'un projet STM32CubeIDE et d'un fichier source en C.
Rajouter dans les fichiers sources de votre projet le fichier TP3_ex1_student.c. Pour cela, il faut exclure du build le fichier main.c. Ensuite, copier le fichier TP3_ex1_student.c dans le répertoire TP3/Core/Src. Faites ensuite simplement un clic droit « Refresh » de votre projet dans le Project explorer sous STM32CubeIDE.

Durant cet exercice, on va chercher à faire clignoter une LED à travers la programmation d'un port GPIO et de la broche correspondante. En l'occurrence, il s'agit de la broche PA_5 sur laquelle est reliée la LED L1 verte utilisateur (cf. Annexe 2), disponible sur les connecteurs CN5 et CN10.

L'objectif est de concevoir 3 fonctions pour être en mesure de manipuler la broche 5 du port GPIOA.

-void Init_Led1(void) : permettant d'initialiser le port GPIOA à travers ses différents registres.

- RCC_AHB1ENR : pour activer l'horloge du port GPIOA, bit RCC_GPIOAEN

Pour contrôler le port, il faut ensuite configurer 4 registres (**cf. Annexe 3**) :

- GPIOx_MODER: pour configurer la direction (mode) de la broche [0…15] du port GPIO[A…I] ;
- GPIOx_OTYPER: pour configurer le type de sortie la broche [0…15] du port GPIO[A…I] ; La broche PA_5 sera configurée en mode output push_pull ;
- GPIOx_SPEEDR : pour configurer la vitesse de commutation de sortie. On choisira le mode haute vitesse entre 50 et 200MHz.
- GPIO_PUPDR : pour configurer la présence d'un pull-up ou pull-down. On optera pour configurer la broche 5 du port GPIOA en pull down.

-void Led1_On(void)
- GPIOx_BSRR : permet de mettre à 1 le bit de la broche x du port GPIOA..I

-void Led1_Off(void)
- GPIOx_BSRR : permet de mettre à 0 le bit de la broche x du port GPIOA..I

Pour avoir plus d'information sur les registres et leurs valeurs, cette documentation vous sera forte utile (p144-> RCC, p176 -> GPIO, …):

STMicroelectronics, 'STM32F446xx advanced Arm-based 32-bit MCUs', RM0390 Reference Manual, rev5 décembre 2020
https://www.st.com/resource/en/reference_manual/dm00135183-stm32f446xx-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf

Une fois les 3 fonctions décrites, faites clignoter la LED sur une période d'une seconde. Pour simplifier, on pourra utiliser la fonction HAL_Delay() (cf. fichier Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal.c).

-Tester sur la carte et faire valider par l'enseignant.

-Consigner tous vos développements dans votre rapport. **Commenter votre code !! Tout code non commenté et non lisible ne sera pas évalué.**

Exercice 2 : GPIO – Librairie HAL

On souhaite maintenant réaliser le même programme mais en se basant cette fois-ci sur la sur-couche logicielle HAL (Hardware Abstraction Layer) fournie par STMicroelectronics.
Pour cela, des fonctions de manipulation sont déjà présentes et documentées. Il reste simplement à les intégrer lors de la création du projet et évidemment comprendre leurs fonctionnalités et paramètres.

-Reprenez votre projet TP3. Penser à exclure le fichier TP3_ex1_student.c du build (clic droit, **properties** > **C/C++ Build** > cocher **exclude from build**.

-Ajouter ensuite le fichier source TP3_ex2_student.c comme précédemment.

-Analyse du driver GPIO de la librairie HAL : Ouvrir le fichier stm32f4xx_hal_gpio.h. Ce fichier se trouve dans Drivers > STM32F4xx_HAL_Driver/Inc.
-Modifier ensuite le code pour réaliser la même fonctionnalité que dans l'exercice 1 mais à partir des fonctions haut-niveau de HAL. La structure GPIO_InitTypeDef doit vous rappeler des choses ☺

-Compiler et tester sur la carte
-Faire valider par l'enseignant
-Consigner tous vos développements dans votre rapport. **Commenter votre code !! Tout code non commenté et non lisible ne sera pas évalué.**

Exercice 3 : GPIO et scrutation

Dans cet exercice, nous allons utiliser le principe de scrutation. Il s'agira de monitorer l'état d'une broche et d'effectuer une action en conséquence.
La broche que l'on scrutera sera la broche 13 du port C. Cette broche est connectée au bouton poussoir bleu (B1) sur la carte Nucleo-F446RE. Dès qu'un changement d'état sur le broche est constaté (transition 1 vers 0), il faudra changer l'état de la led LD1, connectée sur la broche 5 du port GPIOA.

-Créer un nouveau project STM32CubeIDE appelé TP3_Ex3 (cf. Annexe 1)
-A partir de l'utilitaire graphique, réinitialiser la configuration par défaut des broches (Pinouts-> clear Pinouts)
-Configurer ensuite les broches suivantes :
- ➔ Configurer la broche PC_13 en entrée
- ➔ Configurer la broche PA5 en sortie

-Développer le code permettant de réaliser la fonctionnalité souhaitée ; (une petite temporisation de 500ms pourra être utile ☺)
-Consigner vos développements dans votre rapport

Q : Selon vous, quelle est la limitation de la scrutation ? Quelle solution pourrait être mise en œuvre ?

Exercice 4 : GPIO et interruption

A la suite de l'exercice 3, nous souhaitons maintenant utiliser le mécanisme fondamental d'interruption. Il s'agira de configurer la broche PC_13 comme source d'interruption externe. L'activation de cette source d'interruption au niveau du contrôleur NVIC permettra au processeur de venir exécuter ce sous-programme d'interruption à chaque événement sur le bouton poussoir et ensuite de pouvoir continuer son programme principal.

-Créer un nouveau project STM32CubeIDE appelé TP3_Ex4 (cf. Annexe 1)
-A travers la configuration graphique :
- ➔ Configurer la broche PC_13 comme source d'interruption ; il faudra que l'interruption se déclenche sur un front descendant (transition 0->1 : rising edge)
- ➔ Activer cette source d'interruption au niveau du contrôleur NVIC

-Développer le code du sous-programme d'interruption

➔ Ouvrir le fichier stm32f4xx_it.c

Q : Que contient-il ? A quoi correspond

Exercice 5 : Timer et interruption

Dans cet exercice, nous allons utiliser un timer avec les interruptions. L'objectif sera donc de se servir de ce timer pour déclencher périodiquement l'exécution d'un sous-programme spécifique. Dans ce sous-programme d'interruption, nous allons simplement changer l'état de la led LD1 (GPIOA pin 5). Par rapport à l'exemple précédent, ce ne sera donc plus l'utilisateur via l'appui sur un bouton poussoir mais bien le timer qui déclenchera périodiquement l'interruption.

Le timer est simplement un compteur, qui s'incrémente (décrémente, ou les 2) en fonction d'une horloge de comptage. Ils peuvent cependant être aussi configurés dans plusieurs autres modes afin de réaliser plein d'autres fonctionnalités comme :

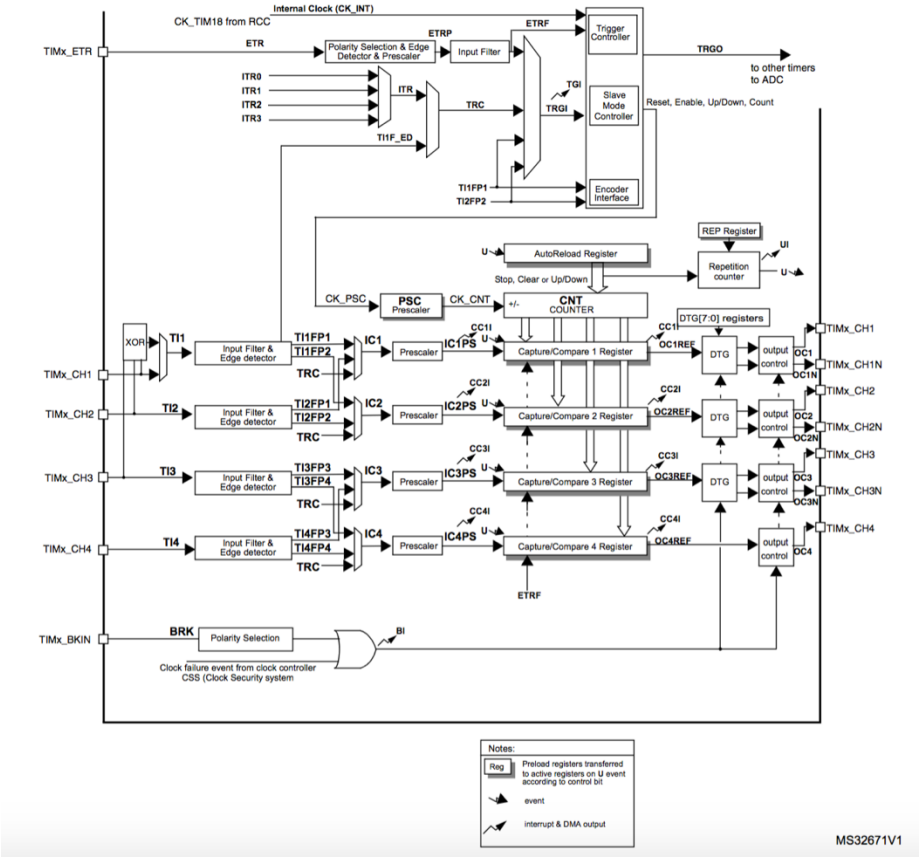
-la mesure la durée d'un pulse d'un signal en entrée (mode *input capture*)
-la génération de signaux périodiques (mode *output compare*)
-la génération de signaux PWM avec modification de largeur d'impulsion (mode PWM)
-la génération d'un pulse en sortie (mode *1 pulse*),

Un des points clés d'un timer est la configuration de l'horloge de comptage, dénotée sur la figure ci-dessous CK_CNT. Vous pouvez observer que cette horloge est générée en sortie d'un prescaler PSC, qui n'est rien d'autre qu'un diviseur. L'horloge PSC peut soit être venir d'une source d'horloge externe ou interne (CK_INT), choisie par défaut.
L'horloge CK_INT n'est rien d'autre que l'horloge délivrée par les bus APB1/APB2 (cf. p57 [1])

-APB1 : Timer 2,3,4,5,6,7,12,13
-APB2 : Timer 1,8,9,10,11

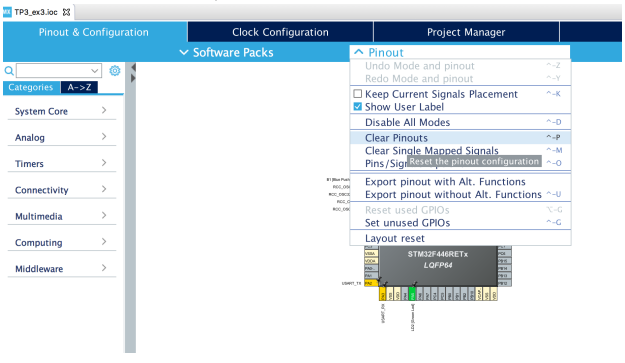
En jouant subtilement sur les valeurs de prescaler et la période de comptage, on peut alors générer des évènements très précisément dans le temps.

Q : -Par quel facteur peut-on diviser la fréquence d'horloge à l'aide d'un prescaler sur 16 bits ? [1 à].



On souhaite alors configurer le Timer 1 afin de générer une interruption toutes les secondes. Dans le sous-programme d'interruption du Timer, on cherchera alors à commuter l'état de la broche PA_5 afin de la faire clignoter toutes les secondes. Si elle était à 0, alors son état basculera à 1, et inversement.

- **Configuration graphique sous STM32CubeIDE**
  -Créer un nouveau projet STM32. Regarder l'annexe 1 si besoin.
  -Sous l'utilitaire graphique, commencer par ré-initialiser complètement la configuration des broches (Pinout > Clear Pinouts) comme illustré ci-dessous :



- Ajouter ensuite le Timer TIM1. Dans la fenêtre supérieure de mode, on choisira une source d'horloge interne.
  Q : Quelle est l'horloge source arrivant en entrée du timer 1 ?

- Dans la configuration du timer TIM1, dans l'onglet *Parameter settings*, configurer les paramètres suivants afin d'avoir de générer un événement toutes les secondes :
  - ➔ Prescaler (PSC – 16 bits value) = ??
  - ➔ Counter Period (AutoReload Register) = ??
  - ➔ Trigger Output (TRG0) Parameters > Trigger Event Selection = Update event

- Toujours pour le timer TIM1, aller dans la fenêtre NVIC settings. Activer le le mode d'interruption permettant le déclenchement à chaque événement de fin de comptage.

La configuration du timer et l'activation de la source d'interruption associée est maintenant terminée. On peut passer à la configuration de la broche PA5 connectée à la led utilisateur.

- Dans la fenêtre Pinout, trouver et configurer la broche PA5.

La configuration est maintenant finie. En sauvegardant le projet, la génération du code s'effectuera.

- **Modification du code source**

La configuration est maintenant finie. En sauvegardant le projet, la génération du code s'effectuera.

-Analyser le fichier main.c
Q : Que réaliser les fonctions MX_GPIO_Init() et MX_TIM1_Init ()?

-Ouvrir le fichier stm32f4xx_it.c
Q : A quoi correspond cette fonction : void TIM1_UP_TIM10_IRQHandler(void)

-Modifier les fichiers main.c et stm32f4xx_it.c afin de réaliser la fonctionnalité souhaitée. Prenez le temps de regarder les drivers des périphériques, c'est-à-dire les fonctions HAL dans les fichiers suivants : stm32f4xx_hal_tim.c, stm32f4xx_hal_gpio.c).

-Tester sur la carte (ne pas hésiter à faire du debug)
-Faire valider par l'enseignant
-Consigner vos développements dans votre rapport. Commenter votre code !!

Q : Bilan : Selon vous, quel est l'intérêt des interruptions par rapport la scrutation ?

Exercice 6 (option) : Pour aller plus loin…

A partir de vos développements en exercice 5, nous souhaitons pouvoir modifier la fréquence de clignotement de la led LD1 (PA_5), à chaque fois que l'on détectera un appui sur le bouton poussoir utilisateur (PC_13).
Par exemple, par défaut, la led LD1 clignote toutes les 1s, puis lors d'un appui, on pourra accélérer la vitesse de clignotement par 2, jusqu'à une limite d'un facteur 16.

Avant les développements :

-Identifier quels sont les périphériques utilisés
-Comment modifier la vitesse de clignotement simplement

-Créer un nouveau projet STM32CubeIDE
-Réaliser les développements après avoir configuré graphiquement le processeur et les périphériques identifiés,
-Consigner vos développements dans votre rapport – Commenter votre code !!

Conclusion

A travers ce TP, vous avez pu découvrir la programmation des périphériques d'un microprocesseur, les GPIOs et le Timer. Vous avez aussi mis en œuvre le mécanisme fondamental d'interruption, permettant l'exécution multitâche.
Les librairies comme HAL ont permis de travailler à haut niveau et ont facilité le développement logiciel.

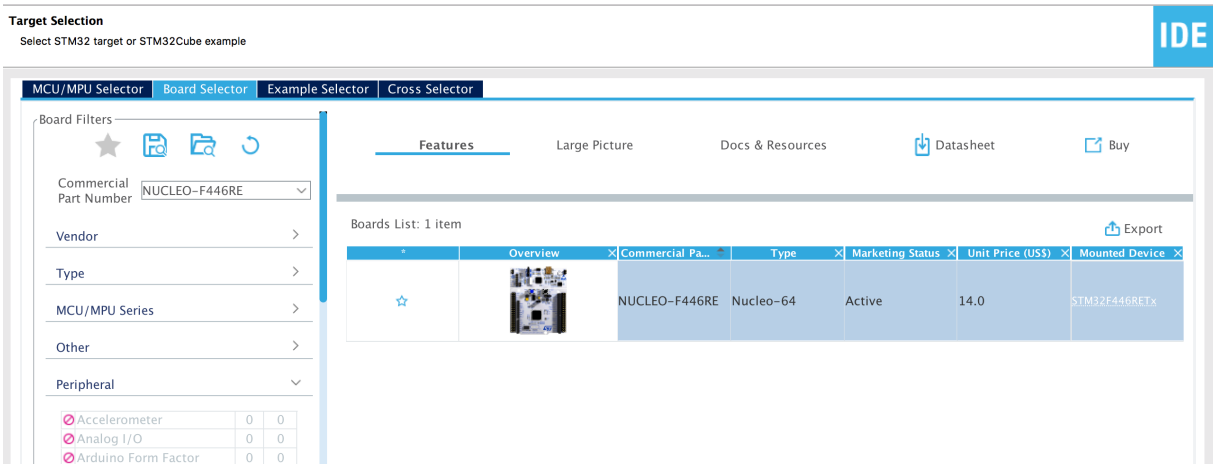
Références

[1]STMicroelectronics, '*STM32F446xx advanced ARM –based 32-bit MCUs*', RM0390 Reference Manual revision 4 February 2018, https://www.st.com/resource/en/reference_manual/dm00135183-stm32f446xx-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf

[2] ARM, Cortex-M4 Instructions, ARM Developer, https://developer.arm.com/documentation/ddi0439/b/Programmers-Model/Instruction-set-summary/Cortex-M4-instructions

ANNEXE 1 : Création d'un projet STM32CubeIDE (sans configuration graphique) en C

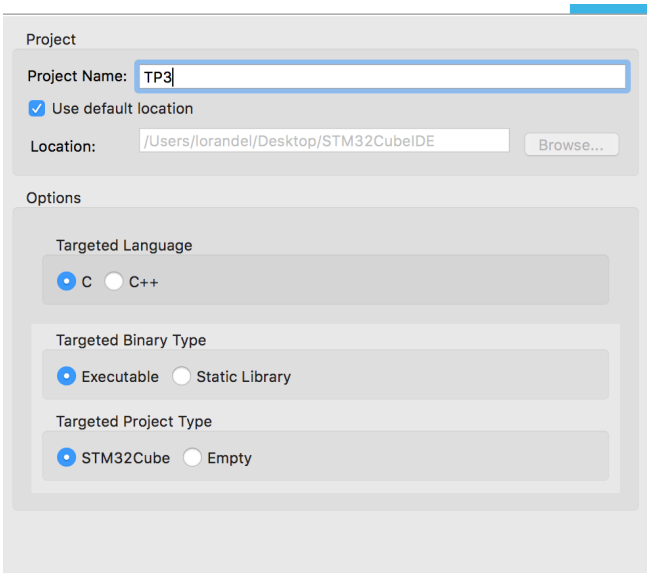
Après avoir lancé STM32CubeIDE, dans l'onglet **File** > **New** > **STM32 Project**, sélectionner la bonne plateforme de développement, via l'onglet **Board Selector.** Pour cela, à l'aide du filtre, tapez **NUCLEO-F446RE** puis **Next**.



Une fenêtre « STM32 Project s'ouvre. Nommer simplement votre projet, et pour l'option « **Targeted Project Type** » choisir soit :

-Empty -> Projet simple sans configuration graphique via STM32CubeMX

-STM32Cube -> si génération d'un projet avec configuration graphique du microcontrôleur ; <u>Choisir cette option par défaut.</u>

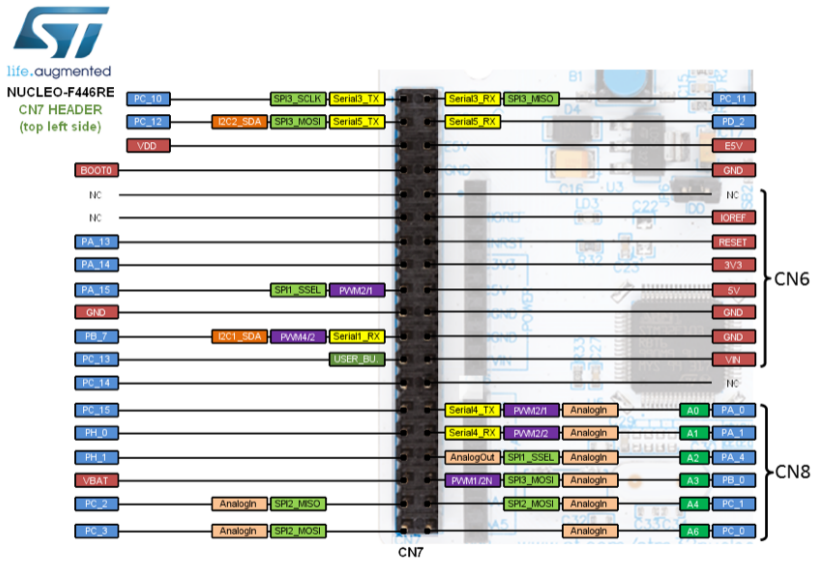


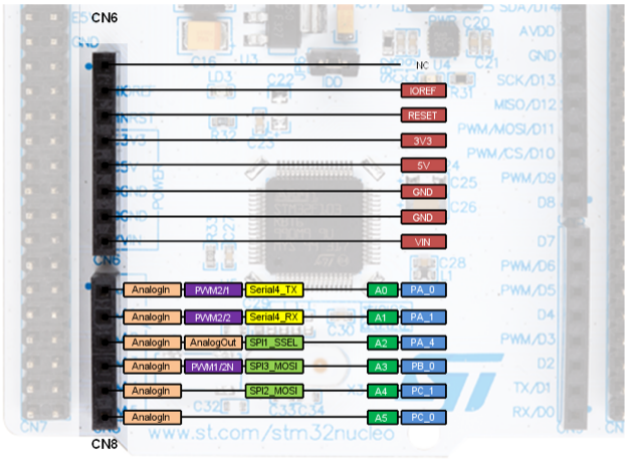
Puis **Finish**.

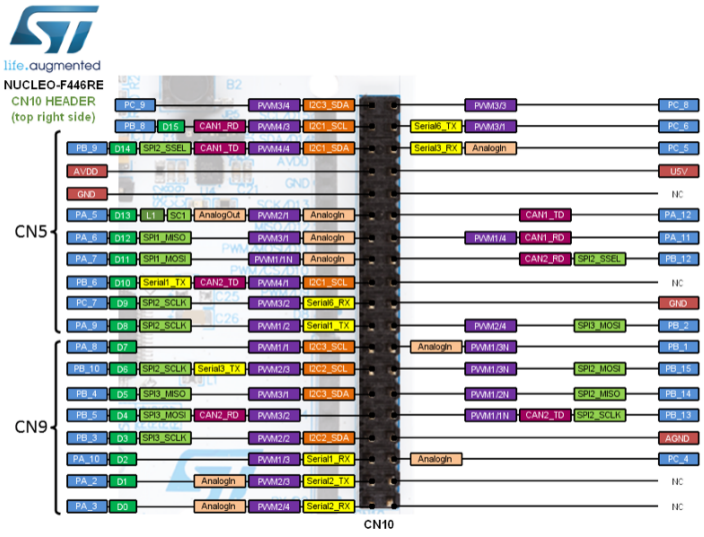
L'utilitaire de création vous demandera si vous souhaitez initialiser les périphériques à leur mode par défaut, choisir **non**.

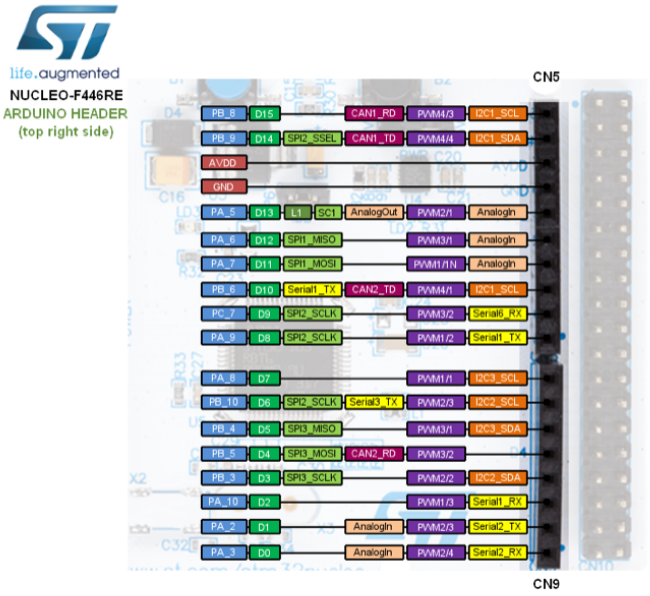
ANNEXE 2 : Pinout du STM32F446RE

Images tirées de Mbed : https://os.mbed.com/platforms/ST-Nucleo-F446RE/








life.augmented
NUCLEO-F446RE
ARDUINO HEADER
(top right side)
CN5
CN9
PB_8 D15 CAN1_RD PWM4/3 I2C1_SCL
PB_9 D14 SPI2_SSEL CAN1_TD PWM4/4 I2C1_SDA
AVDD
GND
PA_5 D13 LI SCI AnalogOut PWM2/1 AnalogIn
PA_6 D12 SPI1_MISO PWM3/1 AnalogIn
PA_7 D11 SPI1_MOSI PWM1/1N AnalogIn
PB_6 D10 Serial1_TX CAN2_TD PWM4/1 I2C1_SCL
PC_7 D9 SPI2_SCLK PWM3/2 Serial6_RX
PA_9 D8 SPI2_SCLK PWM1/2 Serial1_TX
PA_8 D7 PWM1/1 I2C3_SCL
PB_10 D6 SPI2_SCLK Serial3_TX PWM2/3 I2C2_SCL
PB_4 D5 SPI3_MISO PWM3/1 I2C3_SDA
PB_5 D4 SPI3_MOSI CAN2_RD PWM3/2
PB_3 D3 SPI3_SCLK PWM2/2 I2C2_SDA
PA_10 D2 PWM1/3 Serial1_RX
PA_2 D1 AnalogIn PWM2/3 Serial2_TX
PA_3 D0 AnalogIn PWM2/4 Serial2_RX

ANNEXE 3 : Registre de configuration des GPIOs

- GPIOx_MODER : sélection du mode
  - 4 valeurs possibles pour chaque broche
    -‘00’ Input (reset state)
    -‘01’ General purpose output mode
    -‘10’ Alternate Function mode
    -‘11’ Analog mode

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

- Registre GPIOx_OTYPER : sélection du type de sortie
  - 2 valeurs possibles
    -‘0’ sortie en push-pull (reset state)
    -‘1’ sortie en drain ouvert

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

- Registre GPIOx_OSPEEDR :
  - sélection de la vitesse de commutation des sorties
  - 4 valeurs possibles (Fmax dépendant de VDD)
    -‘00’ Low speed (max [2-8]MHz)
    -‘01’ Medium speed (max [12,5-50]MHz)
    -‘10’ Fast speed (max [25-100]MHz)
    -‘11’ High speed (max [50-180]MHz)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [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
OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1 [1:0]		OSPEEDR0 1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Registre GPIOx_PUPDR : mise en œuvre du pull-up/pull-down
  - 4 valeurs possibles :
    -‘00’ no pull-up, no pull-down
    -‘01’ pull-up
    -‘10’ pull-down

-‘11’ reserved

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[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
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw