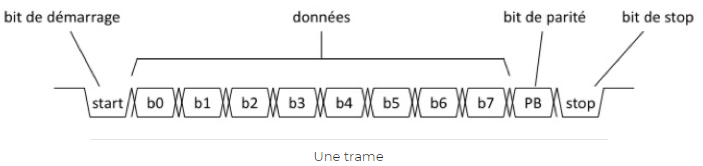
**UART sur stm32** Universal asynchronous receiver transmitter (UART)

Trame :



Le bit de parité peut être utilisé pour détecter des erreurs.

La parité est générée par l’émetteur et vérifiée par le receveur.

Si l’erreur est faible et n’entraîne pas de conséquence grave ne pas utiliser de parité est à privilégier.

Liaison Série

Pas d’horloge communes, pour identifier les différents bits d’une trame, le récepteur se base sur la durée de ses bits.

Cette durée doit être fixée et identique pour l’émetteur et le récepteur.

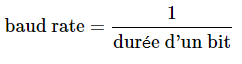
La durée d’un bit est le temps entre chaque bit.

Ainsi :

L’émetteur émet un bit, attend la durée d’un bit puis émet un nouveau bit.

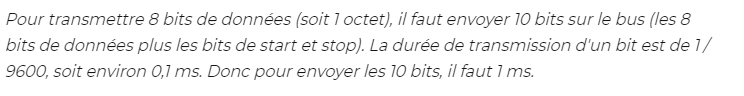
Le receveur attend jusqu’à ce qu’il détecte un bit de démarrage puis attend la moitié d’une durée d’un bit pour se caler sur la lecture des bits qu’il reçoit.

La durée d’un bit décrit en termes de baud rate (unité bps), le nombre de bits par seconde qui est transmis par la communication série



La bande passante (bandwith) d’une communication quantifie l’information transmise sans prendre en considération les surcoûts induits par les bits de démarrage, de parité et de stop soit :





Une autre caractéristique du système est full-duplex, half-duplex (ici un bus doit assurer la retransmission en cas de collision)

Une communication UART est asynchrone il ne faut pas non plus que les horloges de dérives pas trop car une synchronisation à lieu à chaque bit de démarrage, par contre cela limite la bande passante.

Pour USART les bauds rate sont élevé il y a une horloge commune permet de réduire les surcouts dus au bit de démarrage et de stop, car les données peuvent être plus longues.

STM32 dispose d’un périphérique USART full-duplex permettant un baud rate jusqu’à 4.5Mbps.

Avec une broche RX (recevoir) et une TX (transmettre).

Quand l’émetteur est désactivé la broche prend l’état imposée par la configuration du port alors que si l’émetteur est activé et que rien n’est à transmettre la broche TX est dans l’état haut.

Configuration d’un USART en émetteur avec 1 bit de démarrage, 8 bits données, 1 bit stop (pas de parité) et un baud à 9600bps.

* Activer l’USART en mettant à 1 le bit UE du registre CR1,
* Choisir la taille des données (8 ou 9 bits) à l’aide du bit M du registre CR1,
* Indiquer le nombre de bits stop dans le champ de bits STOP du registre CR2,
* Sélectionner le baud rate à l’aide du registre BRR,
* Mettre le bit TE de CR1 à 1 pour envoyer la première trame d’attente.

Pour régler le baud rate la documentation, nous montre qu’il faut diviser la fréquence d’entrée du périphérique 80MHZ par une valeur représentée par un nombre à virgule fixe.

A saisir dans le registre BRR avec les 4 premiers bits pour la partie fractionner et les 12 suivants la partie entière.

Envoyer

1. Programmez les bits M dans USART\_CR1 pour définir la longueur du mot.

2. Sélectionnez le débit en bauds souhaité à l'aide du registre USART\_BRR.

3. Programmez le nombre de bits d'arrêt dans USART\_CR2.

4. Activez l'USART en écrivant le bit UE dans le registre USART\_CR1 à 1.

5. Sélectionnez l'activation DMA (DMAT) dans USART\_CR3 si la communication multi-tampon doit prendre endroit.

Configurez le registre DMA comme expliqué dans la communication multi-tampon.

6. Réglez le bit TE dans USART\_CR1 pour envoyer une trame inactive comme première transmission.

7. Ecrire les données à envoyer dans le registre USART\_TDR (ceci efface le bit TXE).

Répétez ceci pour chaque donnée à transmettre en cas de mémoire tampon unique.

8. Après avoir écrit les dernières données dans le registre USART\_TDR, attendez que TC=1.

Cela indique que la transmission de la dernière trame est terminée.

Ceci est requis par exemple lorsque l'USART est désactivé ou passe en mode Halt pour éviter de corrompre la derniere

Transmission.

Procédure de réception de trame

1. Programmez les bits M dans USART\_CR1 pour définir la longueur du mot.

2. Sélectionnez le débit en bauds souhaité à l'aide du registre de débit en bauds USART\_BRR

3. Programmez le nombre de bits d'arrêt dans USART\_CR2.

4. Activez l'USART en écrivant le bit UE dans le registre USART\_CR1 à 1.

5. Sélectionnez l'activation DMA (DMAR) dans USART\_CR3 si la communication multi-tampon doit prendre endroit.

Configurez le registre DMA comme expliqué dans la communication multi-tampon.

6. Activez le bit RE USART\_CR1. Cela permet au récepteur qui commence à rechercher un peu de démarrage.

Lorsqu'une trame est reçu :

• Le bit RXNE est défini pour indiquer que le contenu du registre à décalage est transféré au RDR. En d'autres termes, les

données ont été reçues et peuvent être lues (ainsi que leurs indicateurs d'erreur associés).

• Une interruption est générée si le bit RXNEIE est défini.

• Les drapeaux d'erreur peuvent être définis si une erreur de trame, du bruit ou une erreur de dépassement a été détecté

lors de la réception. Le drapeau PE peut également être défini avec RXNE.

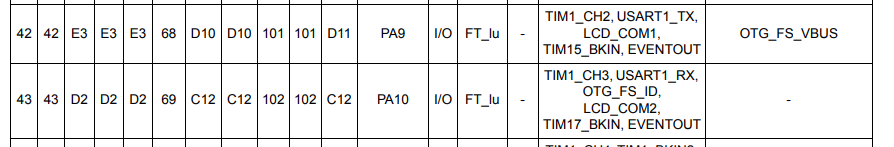
• En multi-tampon, RXNE est défini après chaque octet reçu et est effacé par la lecture DMA de

le Registre des données de réception.

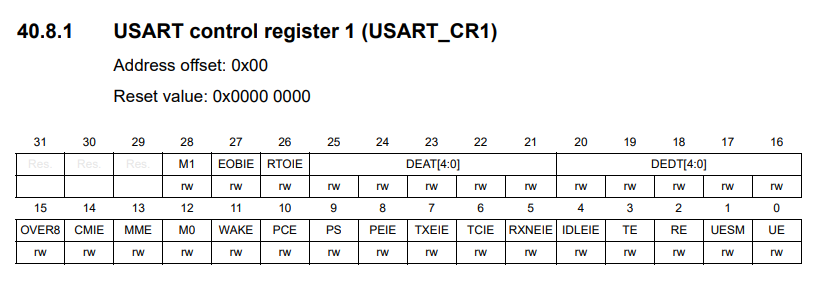
• En mode tampon unique, l'effacement du bit RXNE est effectué par un logiciel lu sur le Registre USART\_RDR. Le drapeau RXNE peut également être effacé en écrivant 1 dans le RXFRQ dans le registre USART\_RQR. Le bit RXNE doit être effacé

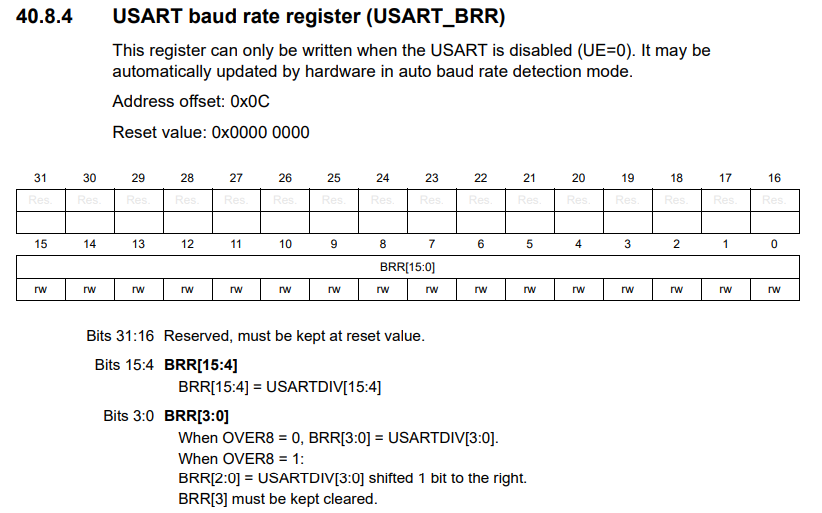
Pratique :

Comme l’exemple d’openclassRoom



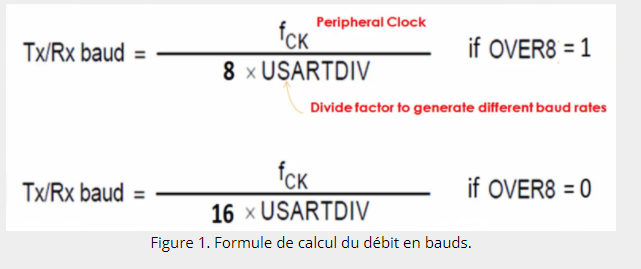
3 USART, 2 UART, 1 Low-power UART





Calcul du baud rate

Si USART->CR1 => OVER 8 à 0 pour nous (on peut le mettre à 1 mais le calcul change)



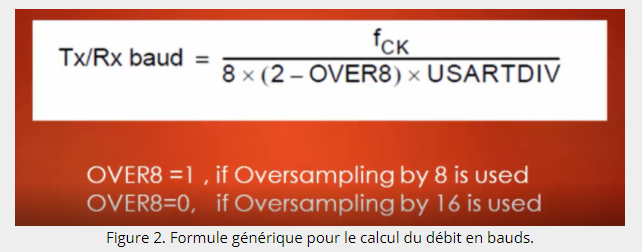
Mode usart nous parlons de UART asynchrone

Over8=1 cela signifie nous utilisons un suréchantillonnage par 8, et le bloc récepteur du périphérique prend 8 échantillons pour comprendre un peu.

Over=8=0 on utilise un suréchantillons par 16.

USARTDIV est le facteur de division pour générer différents débits en bauds (la valeur minimale est 1)

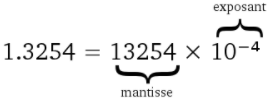
Formule générique :



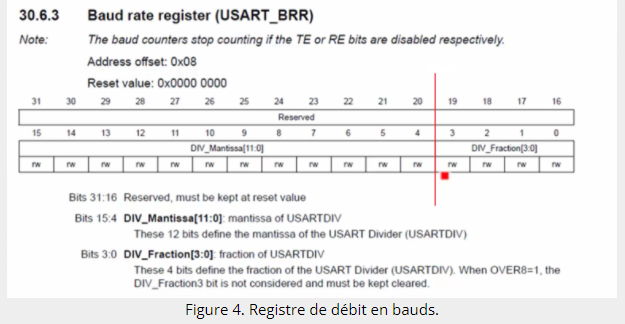
Générer un baud de 9600 bits par seconde

Si horloge générique est de 16MHZ et le suréchantillonnage par 16 est utilisé :

= 16000000/(8\*2\*9600) = 104.16 arrondi à 17 dans le BRR ce registre est divisé en 2 sections [15 :4] et [3 :0]



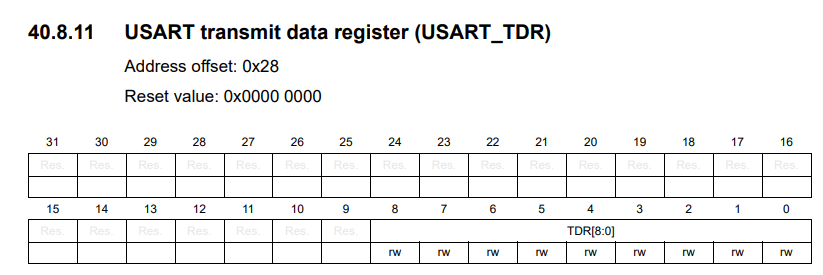
Pour exemple 104.1978 (fractionner 1875 et 104 est la mantisse) : 0000 0000 0000 0000 mantisse de 11 bits



Donc 0,1876 mutipliez par 16 (suréchantillonnage) = 3 partie fractionnée

Donc 104 en hexa 0x68

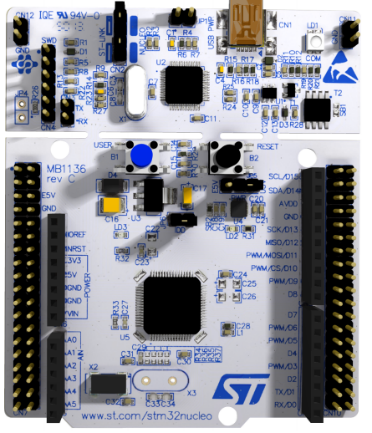
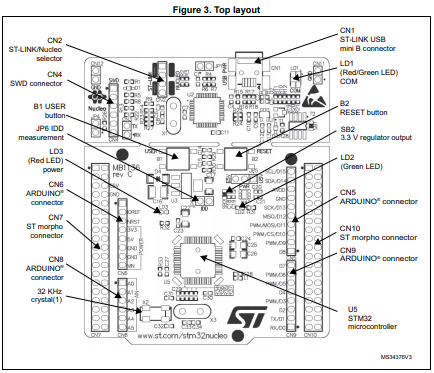
Donc USARTDIV sera 0x683 pour 9600bps

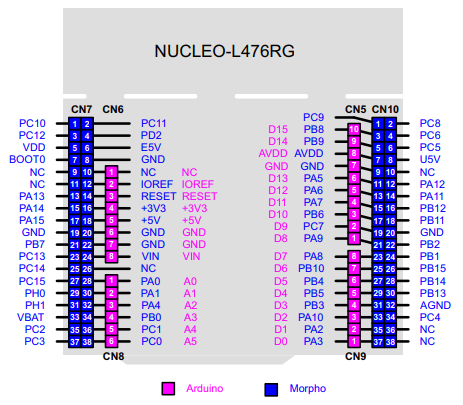


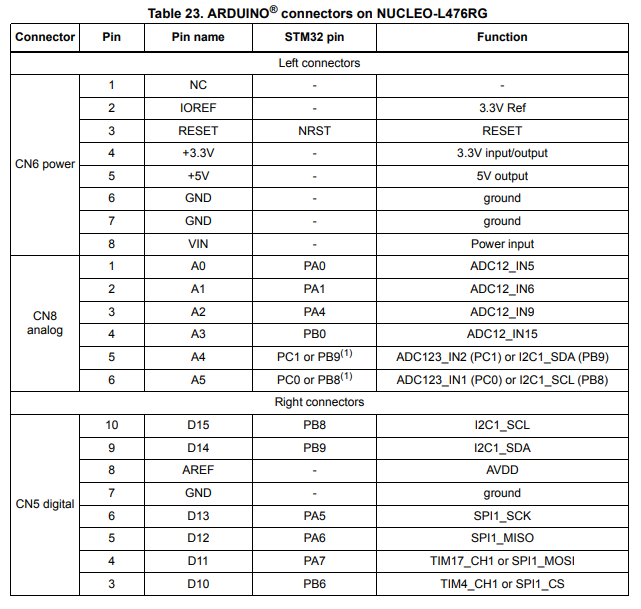
Pour nous 72Mhz en 9600bps over8=0

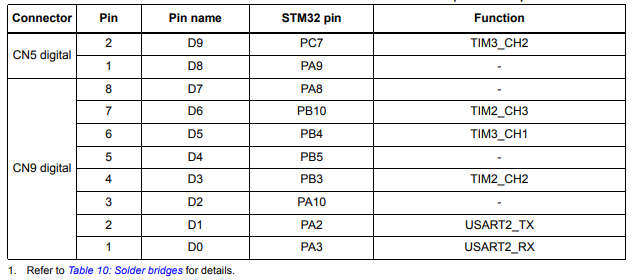
= 72000000/(8\*2\*9600) = 468.75 on retrouve bien le bon résultat

Pour nous on est à 80MHZ : = 80000000/(8\*2\*9600)=520,83

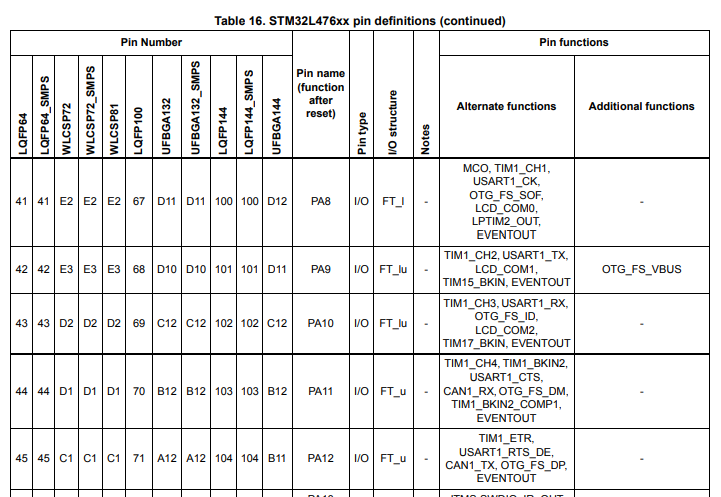








PA9 USART1\_TX même pas dans la doc Nucleo mais dans [datasheet]



Liaison

RaspberryPi Nucleo

GND-----------------------CN7(20)

TXD-----------------------CN10(23) PA10 USART1\_RX

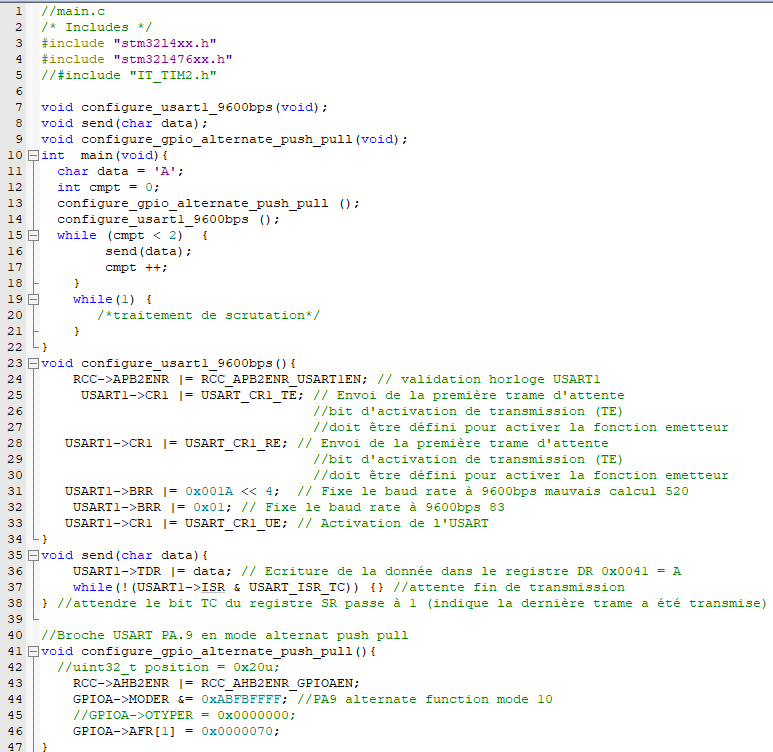
RXD-----------------------CN10(21) PA9 USART1\_TX

--------------------------------------------------------------------------------------------------------------------------------

Programme keil (Envoyer les données depuis stm32)

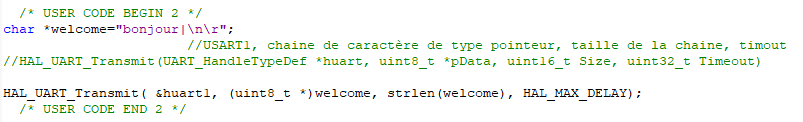
GND -----------------------CN7(20)

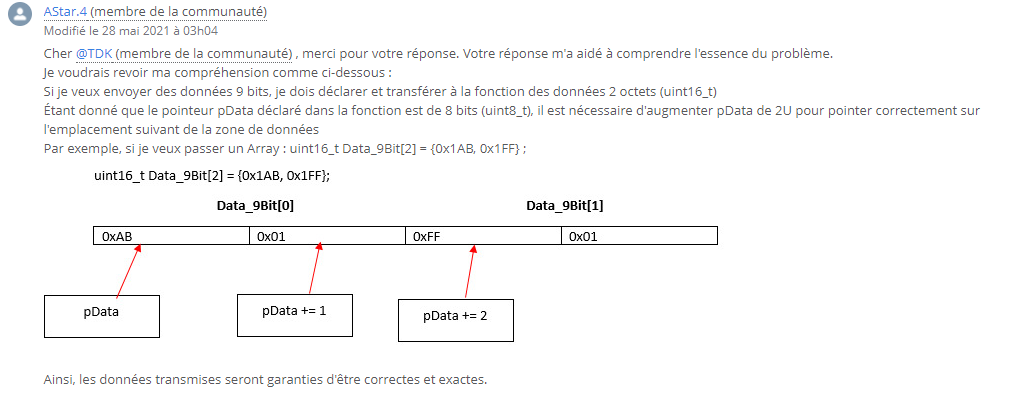
RXD <-----------------------CN10(21) PA9 USART1\_TX



Après CubMX la compréhension est ça marche (code qui fonctionne ci-dessus)

MXCUBE config PA.9 USART1 (9600,8bits, 1stop)





ET L’INCROYABLE et mis en évidence

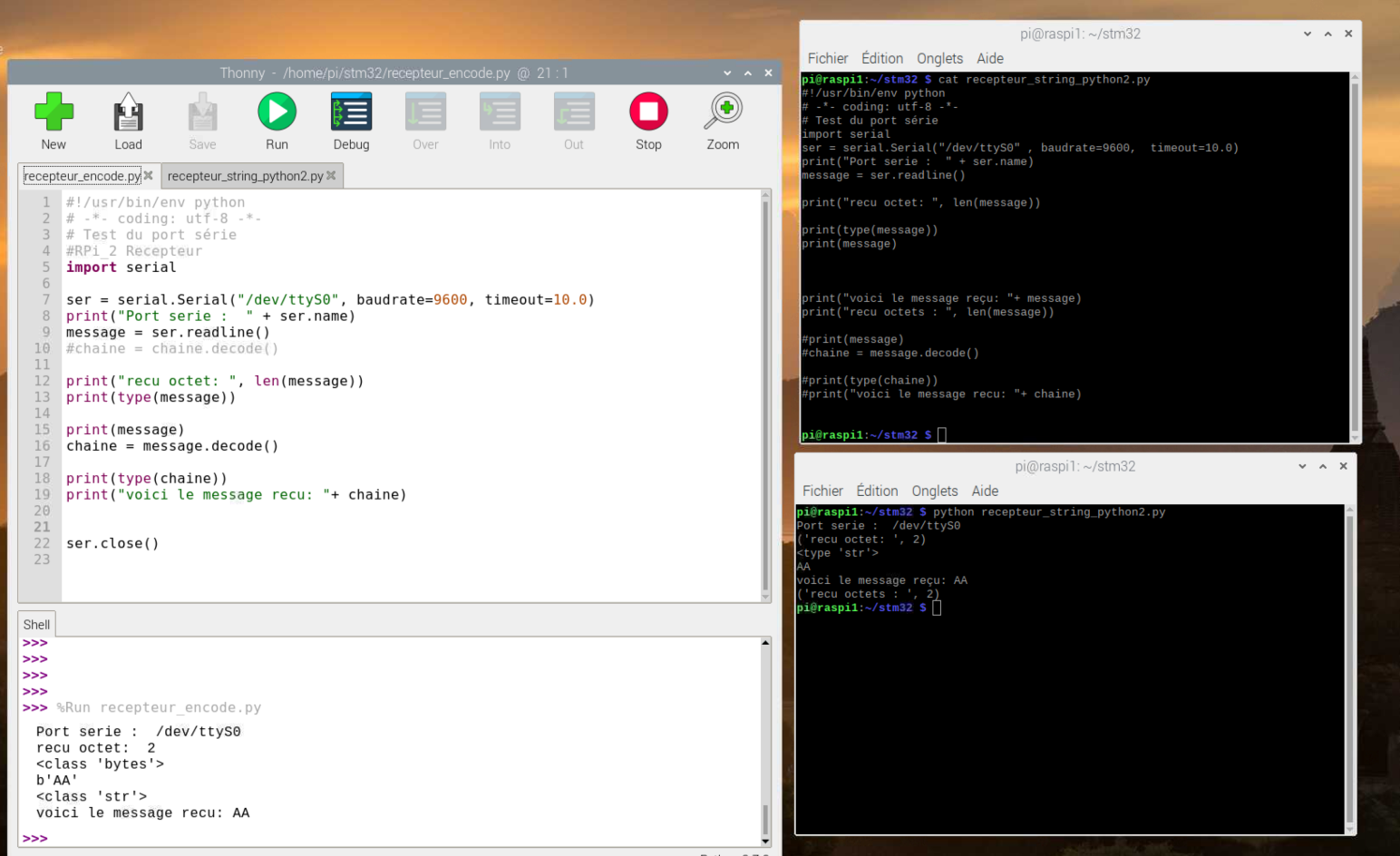
Python2 msg=ser.read() le type <c\_str> Python3 msg=ser.read() le type <bytes>

Pourtant j’envoie bien je pense en byte. Même constat si on discute uart avec un pc et le raspberry

Effectivement la seul fois que j’envoie un string c’est avec python2 emetteur

Mais si on exécute le script python2 avec python3 il nous dit bytes

Le code envoie un seul caractère et je n’ai pas réussit à envoyer une chaine de caractère



Recevoir des données sur le port UART en mode interruption

Une information va arriver sur le port uart, un évènement qui va être pris en charge par le NVIC qui va appeler l’ISR avec la priorité la plus haute (pour que le NVIC soit disponible pour la prise en charge d’autres évènements)

L’ISR va appeler une fonction de callback (Interrupt service routine) et l’ISR doit être aussi libérer pour d’autres instructions. Une fois les instructions du callback exécutés, l’ISR est rappelé pour remettre le flag d’interruptions à 0.

Ensuite la CPU reprend là où elle en était au moment de l’interruption.

Aller plus loin dans le registre car les callbacks sont directement appelés dans l’ISR.

Initialiser et démarrer le service d’interruption (Gérer par deux fonctions générer par cubemx)

Une autre fonction non générer par cubemx la fonction receive (soit dans le void setup ou le while).

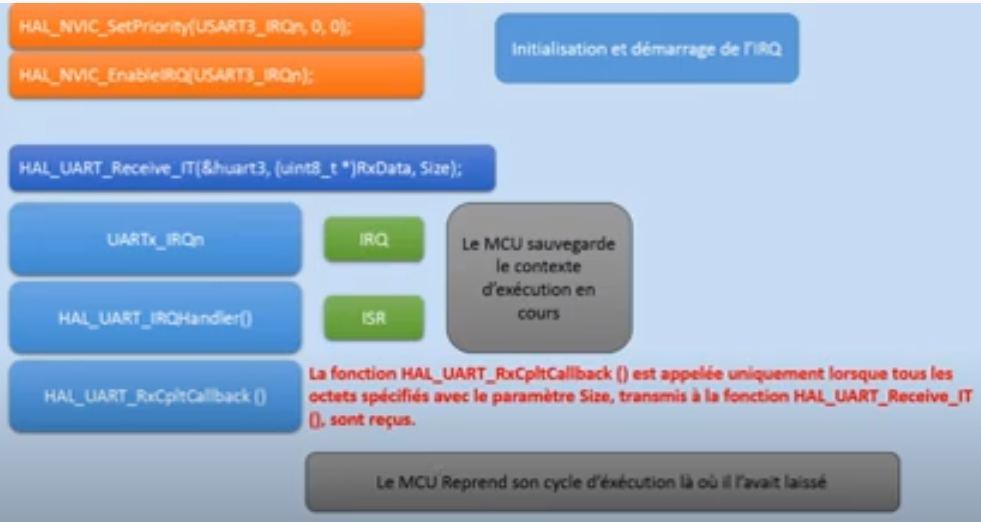
On a le prototype ensuite l’appelle, quand on reçoit une donnée sur le port uart, cela déclenche une requête d’interruption et va appeler une fonction IRQn, elle-même va appeler IRQHandler.

Parallèlement à l’appel sur une des deux fonctions, ka cpu sauvegarde le contexte d’exécution en cours pour reprendre où il en était quand on aura exécuté les instructions.

Donc la fonction IRQHandler appelle RXCompletCallback seulement si le nombre d’octet reçu correspond au paramètre size de la fonction UART\_Receive\_IT (pas de fonction rxcomplete callback)

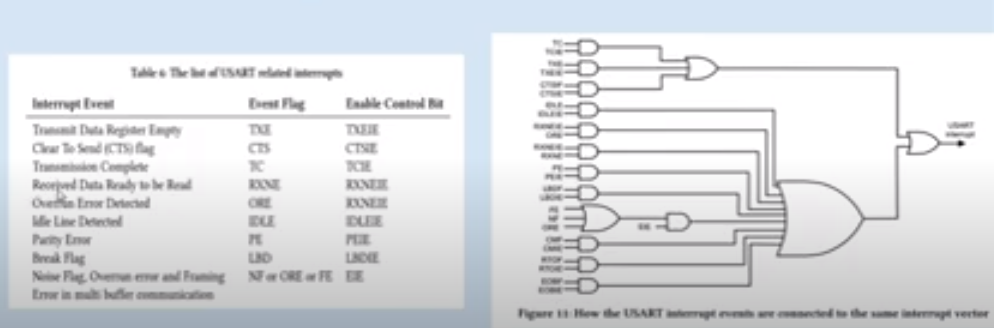
(déconseille de mettre nos instruction dans la fonction IRQHandler car il va avoir un problème avec les drapeau d’interruption) Alors utiliser le RXCompletCallback (c’est même encore galère).

Un fois terminer le cpu reprend où il en était.



Si la fonction UART\_receive\_IT n’est pas dans la boucle il faudra la réactiver (la boucle while(1) a IRQn, IRQHandler,RxCpltCallback). Après RxCpltCallback ou remettre dans le while(1).

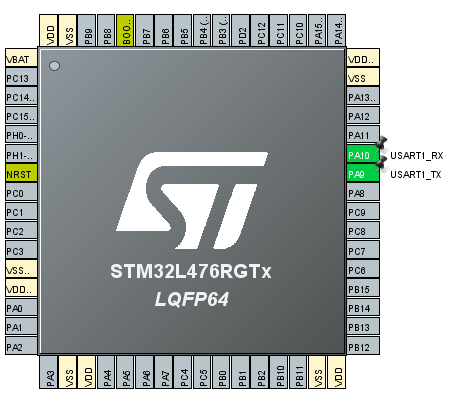
Dans le manual usart



Programme Stm32CubeMX

Téléchargement : <https://www.st.com/en/development-tools/stm32cubemx.html>

Téléchargement : <https://www.st.com/en/development-tools/stm32cubeide.html>



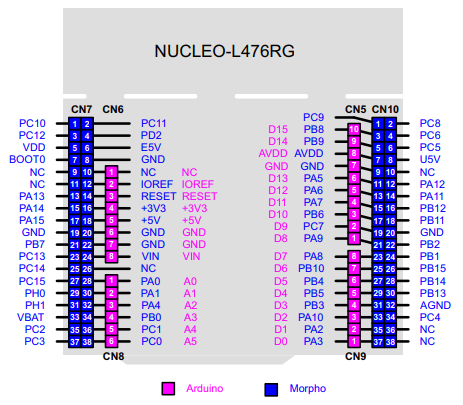
USART1 9600Baud voir en fonction du clock

NVIC enable USART1 global iterrupt

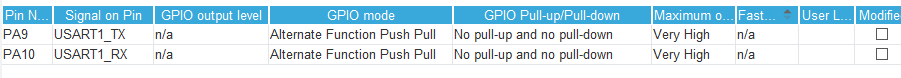
GND-----------------------CN7(20)

TXD-----------------------CN10(33) PA10 USART1\_RX

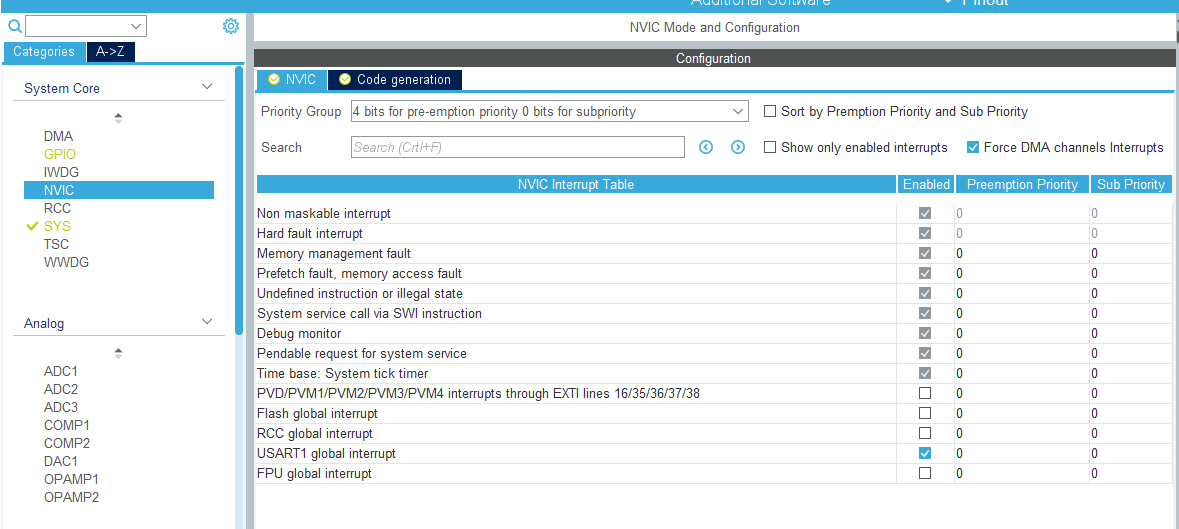
RXD-----------------------CN10(21) PA9 USART1\_TX



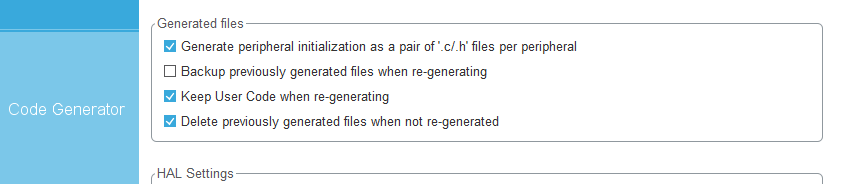
Connectivity: USART1 /Asynchrone/9600bauds

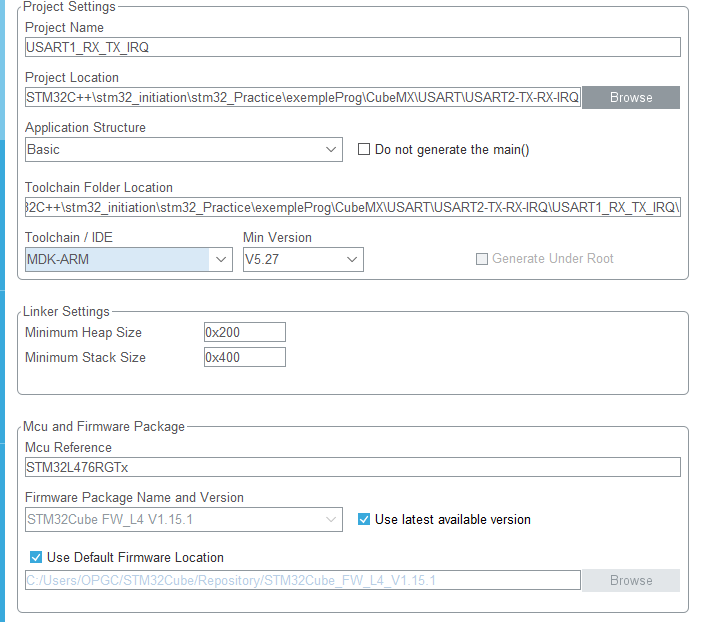
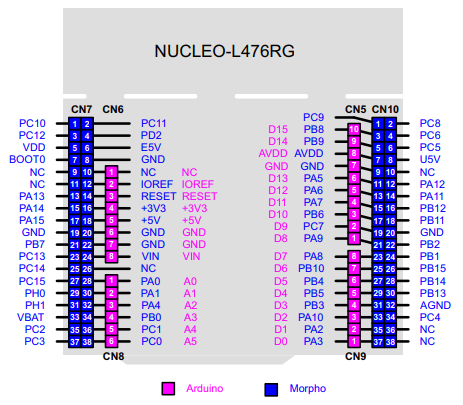


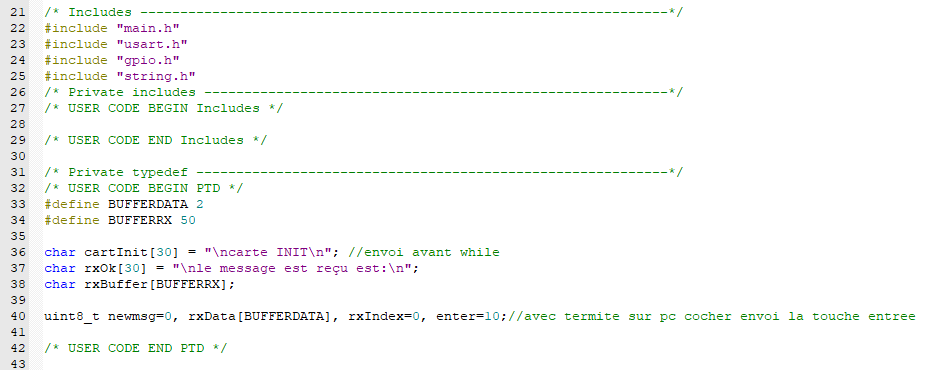
On peut changer la priorité mais à 0 ça marche

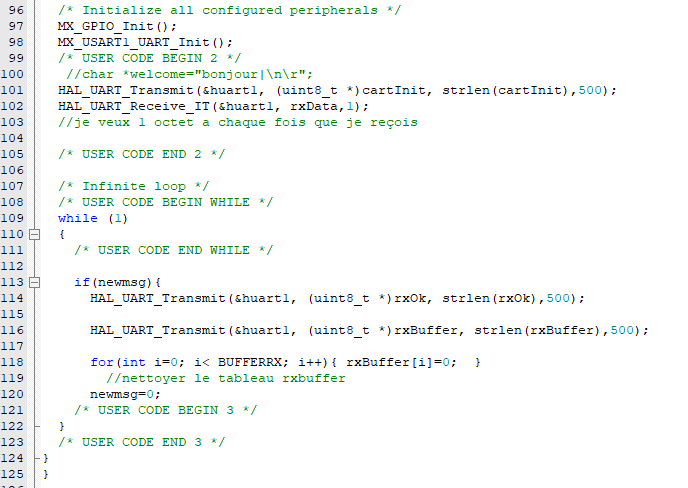


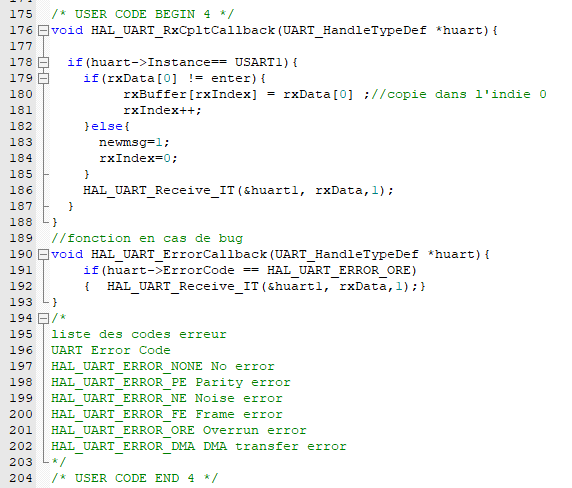
Créer le projet pour avoir les codes usart.h et gpio.h





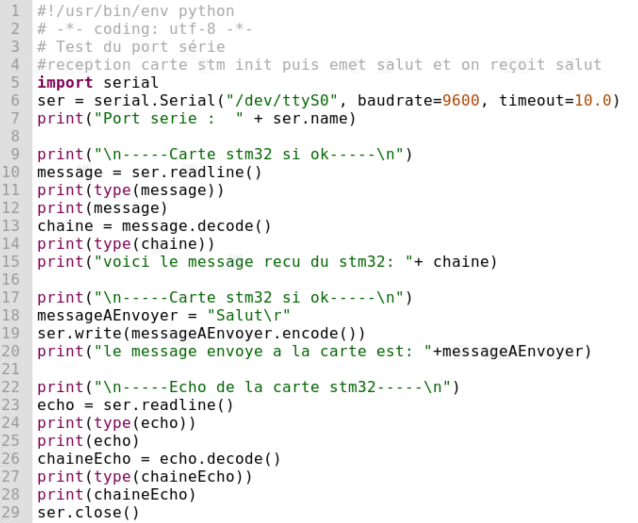


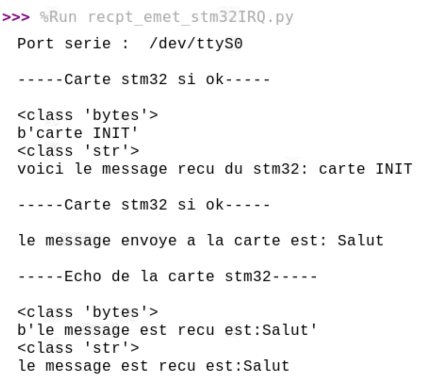




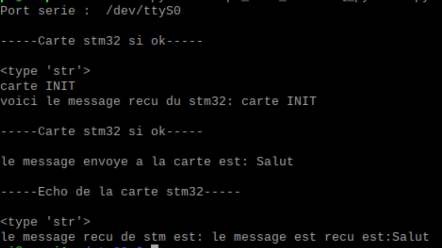
Code Python3 raspberryPi

mais changer le enter=13 ce qui équivaut à CR carriage return = ascii : 13 = \r





Python2



Recevoir des données transmises par l’uart ordinateur Vers STM32 en mode polling et sous forme de question réponse.

Le pc envoie une question le micro-contrôleur envoie un message de confirmation et la réponse sera le même message.

3 modes pour recevoir avec l’uart :

Polling : (opération bloquante la cpu est bloqué) si l’opération est appelée la cpu ne charge plus rien tant que la réception est en cours. Et riense fait, si la bien reçu l’information d’être en réception car la cpu peut être occupée à autre chose.

Interupt : la tâche du microcontrôleur est interrompue, une interruption se lève et la cpu procède à l’écoute du port et de la réception de la trame et ensuite la cpu reprend ses activités là où elle est à laisser. Aucune réception ne sera manquée.

Il faut gérer l’interruption et la fonction de callback:

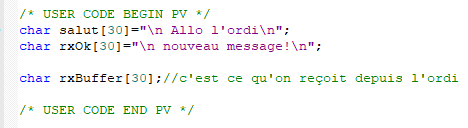
DMA : le mode DMA c’est ce qui a de mieux pour l’approche d’une application temps réel.

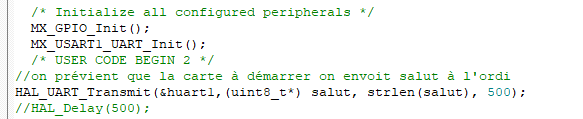
Prise en compte :

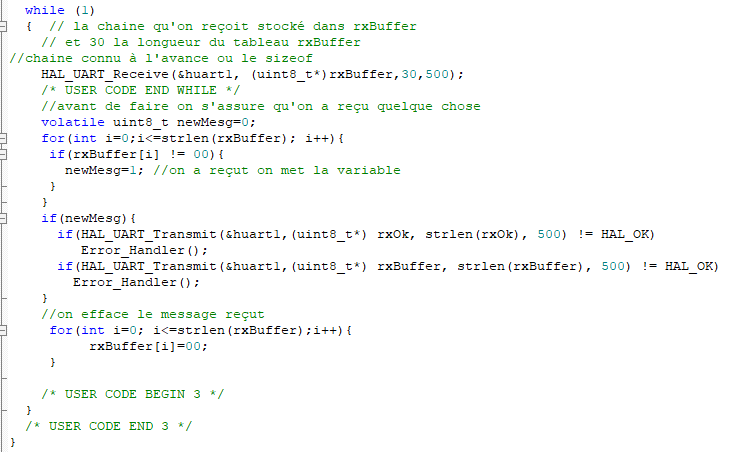
Par voie d’interruption la cpu est mobilisé tout le temps de l’interruption donc tant que des trames arrivent la cpu ne fait pas autres choses elle est bloquée.

Alors ici la DMA à chaque réception de trame elle lève une interruption lorsque la trame n’est pas réussite, mais complète ce qui permet de gagner un peu de temps.

MODE POLLING







Attention au modèle HAL le Error\_Handler à une boucle while(1) on peut rester coincé donc on peut modifier et traiter l’interruption le code erreur sinon on ne l’utilise pas.

Tuto super :

<https://www.carnetdumaker.net/forum/topics/140-les-stm32-en-videos/#partie-12-recevoir-des-donnees-depuis-le-pc-par-uart-interrupt-mode>