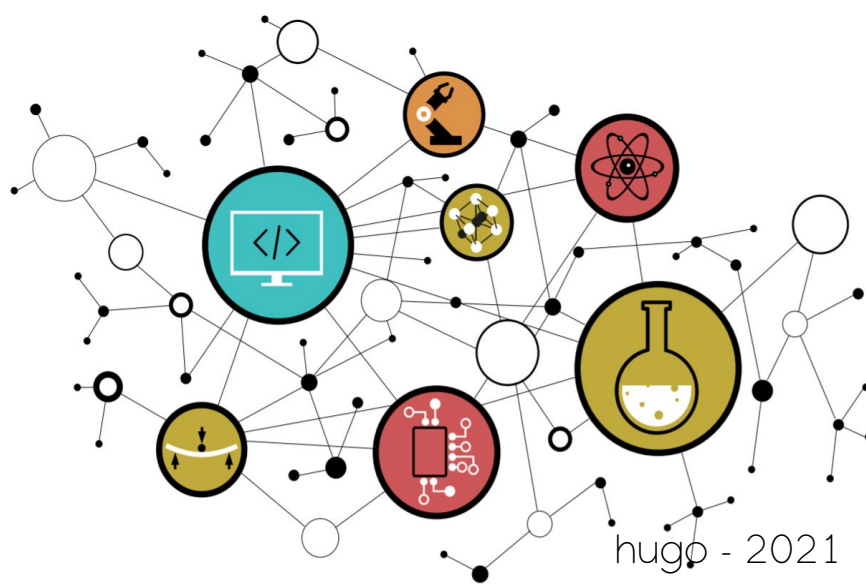


# TRAVAUX PRATIQUES

MODULE DE COMMUNICATION UART  
ET LIAISON SÉRIE

---



## SOMMAIRE

## 4. MODULE DE COMMUNICATION UART ET LIAISON SÉRIE

- 4.1. Introduction : *Protocole de communication d'une liaison série asynchrone*
- 4.2. Introduction : *Module périphérique UART*
- 4.3. Introduction : *Norme RS232*
- 4.4. Introduction : *Module périphérique UART sur PIC18*
- 4.5. Introduction : *Configuration du module UART sur PIC18*
- 4.6. Module périphérique UART1 en transmission
- 4.7. Terminal de communication série sur ordinateur
- 4.8. Transmission de chaînes de caractères
- 4.9. Module périphérique UART1 en réception
- 4.10. Buffer circulaire de réception
- 4.11. Contrôle de flux logiciel
- 4.12. Réception de chaînes de caractères
- 4.13. Module périphérique UART2
- 4.14. Bridge de communication UART1 vers UART2

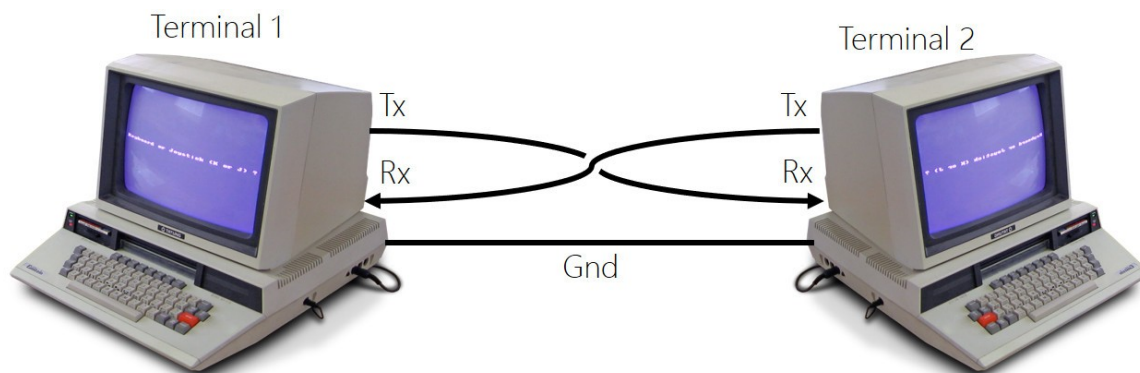
# MODULE DE COMMUNICATION

---

## UART ET LIAISON SÉRIE

---

### 4. MODULE DE COMMUNICATION UART



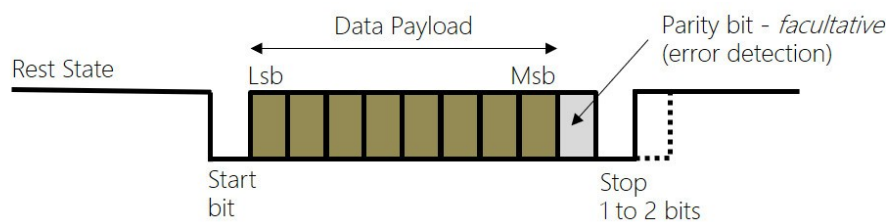
Le protocole de communication d'une liaison série asynchrone et les périphériques UART (Universal Asynchronous Receiver Transmitter) l'exploitant sont rencontrés depuis maintenant longtemps sur processeur numérique et ordinateur. A titre indicatif, la norme RS232 a été spécifiée en 1981 et n'a progressivement disparu sur ordinateur qu'à partir des années 2000 suite à l'arrivée de l'USB (norme USB1.0 Universal Serial Bus en 1996). Les modules de communications UART restent néanmoins encore très rencontrés en Systèmes Embarqués. Tout simplement car ils répondent toujours à un besoin (échanges bas débits de caractères en mode point à point. Il s'agit de communication Full Duplex (communication bidirectionnelle simultanée) utilisant 3 conducteurs physique. Les broches sont souvent nommées :

- **Rx** : Broche de réception croisée (toujours vu du récepteur). Tx vers Rx.
- **Tx** : Broche de transmission croisée (toujours vu du transmetteur). Tx vers Rx.
- **Gnd** : Fil de référence de masse (ground ou masse)

Les modules périphériques UART seront encore probablement utilisés très longtemps et sont bien connus des ingénieurs du domaine. L'UART est un périphérique spécialisé dans l'échange de caractères (données sur 8bits de façon générique) en topologie point à point entre 2 systèmes (peer to peer). Le tout avec des débits plutôt lents (de qq1KBps à qq100KBps) à notre époque (contrôle de procédé, test et prototypage, mesure, contrôle de module de communication, etc). Étant spécialisé dans l'échange de caractères, si un Homme cherche à interagir directement avec un périphérique UART depuis un ordinateur, celui-ci aura à configurer et utiliser un terminal asynchrone de communication (minicom, kermit, PuTTY etc sous GNU/Linux et TeraTerm, PuTTY, etc sous Windows). Rappelons de façon générique qu'un terminal ou une console est dédiée aux communications en mode texte par échange de phrases. Exemple ci-dessous d'interface de configuration sous terminal minicom sur système GNU/Linux.

```
+-----[configuration]-----+
| Filenames and paths          |
| File transfer protocols      |
| Serial port setup            |
| Modem and dialing            |
| Screen and keyboard          |
| Save setup as dfl             |
| Save setup as..              |
| Exit                          |
| Exit from Minicom            |
+-----+-----+-----+-----+
```

### 4.1. Protocole de communication d'une liaison série asynchrone



Ne pas confondre l'UART (périphérique matériel) et le protocole de communication (technique d'échange d'information). Une liaison série asynchrone est l'un des rares derniers protocoles asynchrones de communication rencontré sur le marché. Cela signifie que l'émetteur n'envoie aucune information concernant le débit ou vitesse de transfert, à l'instar des SPI et I2C (conducteur physique d'horloge dédié) ou encore de l'USB et l'Ethernet (champs d'horloge en en-tête des trames assurant la synchronisation d'une PLL à la réception).

Sur une liaison série asynchrone, l'état au repos de la ligne de communication est l'état logique haut (état de repos ou rest state ci-dessus). Une trame débutera toujours par 1 bit de start (état logique bas). Suivent 7 ou 8 bits de données utiles ou payload (charge utile), 1 bit de parité facultatif permettant une gestion élémentaire de détection d'erreur. Une trame se termine par 1 ou 2 bits de stop.

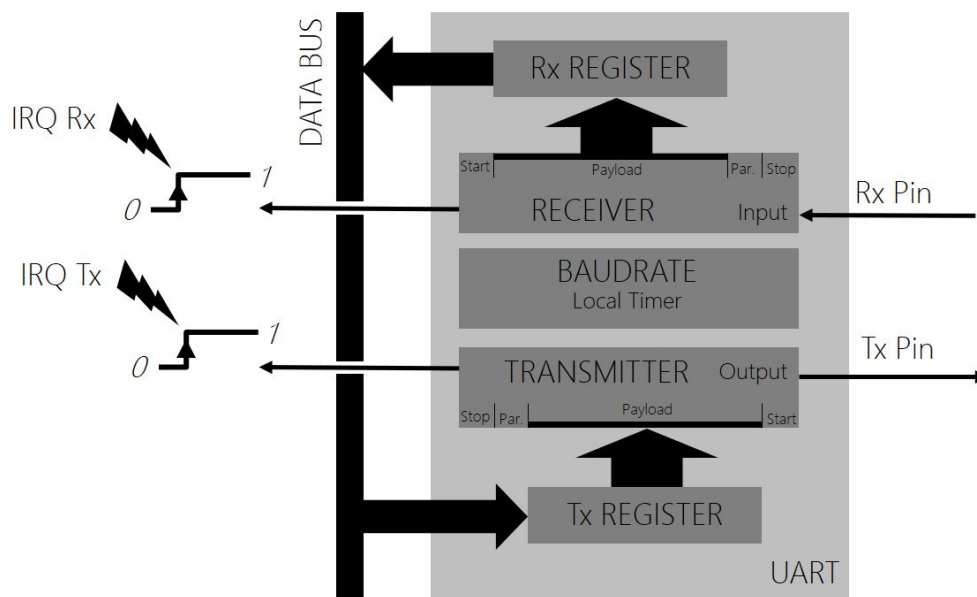
Néanmoins, la liaison série asynchrone est le plus souvent utilisée avec la configuration suivante : 1 bits de start, 8 bits de payload et 1 bit de stop (solution minimale et souvent suffisante pour du test). Une trame série complète d'informations est alors constituée de 10 bits dont 8 bits utiles (différence entre débit réel et utile). Par exemple, avec un débit configuré à 9600 Bd/s (Bps ou Baud/s ou symbole/s ou bits/s pour une liaison série asynchrone), l'envoi d'un bit dure environ 100 µs et donc l'envoi d'un caractère environ 1 ms.

Durant une communication de module UART à module UART, physique ou logique, ne jamais oublier de bien configurer récepteur et émetteur au même débit et avec le même protocole de communication (nombre de bits de donnée, nombre de bits de stop, gestion du bit de parité et contrôle de flux). Exemple ci-dessous d'interface de configuration sous terminal minicom sur système GNU/Linux :

- Débits 9600 Bps
- 8 bits de payload
- Pas de bit parité
- 1 bit de stop
- Pas de contrôle de flux matériel ni logiciel

```
+-----+
| A -   Serial Device       : /dev/ttyUSB0
| B - Lockfile Location    : /var/lock
| C -   Callin Program     :
| D -   Callout Program    :
| E -   Bps/Par/Bits       : 9600 8N1
| F - Hardware Flow Control : No
| G - Software Flow Control: No
|
| Change which setting? ☐
+-----+
```

### 4.2. Module périphérique UART



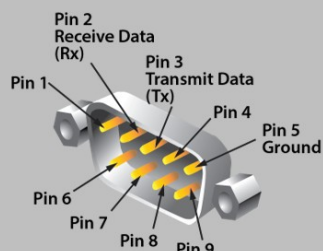
Un périphérique UART (Universal Asynchronous Receiver Transmitter) peut être vu comme deux périphériques dissociés. Un récepteur (receiver) et un transmetteur (transmitter). Néanmoins, les deux implémentent le protocole d'une liaison série asynchrone et leur configuration protocolaire est similaire (protocole et débit). Un Timer local dédié (Baurate ci-dessus) est souvent présent dans un UART afin de configurer le débit de communication.

Comme tout périphérique série de communication, le transmetteur et le récepteur sont conçus autour de registres à décalage. Le récepteur est chargé de récupérer les trames bit à bit dans son registre interne à décalage, de détecter d'éventuelles erreurs de transmission puis d'enlever l'enveloppe protocolaire de la trame pour ne récupérer que les données utiles (payload). A chaque nouvelle trame, la donnée utile est chargée dans un registre de travail (Rx register ci-dessus) afin d'être récupérée par le CPU et une requête d'interruption est envoyée au CPU (IRQ Rx) pour le prévenir de l'arrivée d'une donnée.

Le transmetteur fait quant à lui le travail inverse. L'application demande à envoyer une donnée par écriture dans le registre de transmission (Tx register ci-dessus, opération atomique de qq cycles CPU). Néanmoins, cela ne signifie pas que la donnée ait été transmise. Le transmetteur est alors responsable de charger la donnée utile dans son registre interne à décalage, de rajouter l'enveloppe protocolaire (start, stop voire parité) puis d'envoyer bit à bit les données en respectant le débit configuré. Une fois la donnée envoyée, une requête d'interruption est envoyée au CPU (IRQ Tx) pour le prévenir de la fin de transmission.

### 4.3. Norme RS232

DB-9 male to DB-9 female



DB-9 male to USB



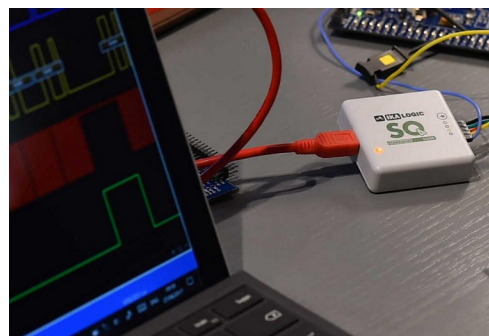
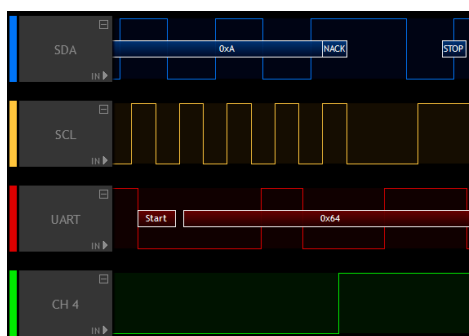
MCU bridge  
USB to UART inside

Ne pas confondre la norme RS-232 (nommée aussi EIA 232) avec le protocole d'une liaison série asynchrone et un périphérique UART. Cette norme est apparue en 1981 et a donné naissance aux interfaces port COM sur ordinateur sous Windows (remplacé par l'USB depuis les années 2000). Cette norme de communication de machine à machine utilise des UART et implémente une liaison série asynchrone, mais tend à standardiser les débits de communication, les connecteurs et câbles associés, les longueurs de câbles, les niveaux de tension sur les conducteurs physiques, etc.

Cette norme standardise donc des contraintes et des limites physiques permettant de faciliter l'interfaçage et la communication de machines entre elles. Observons quelques unes de ces limitations :

- Longueurs de câbles en fonction des débits : 60m (2,4Kb/s), 15m (9,6Kb/s) et 2,6m (56Kb/s)
- Niveaux de tensions (logique inversée) : niveau logique 0 (de +3V à +25V) niveau logique 1 (de -3V à -25V)
- Technique de codage des bits : NRZ (Non Return to Zero)
- Types de connecteurs : DB9 (9 broches), etc

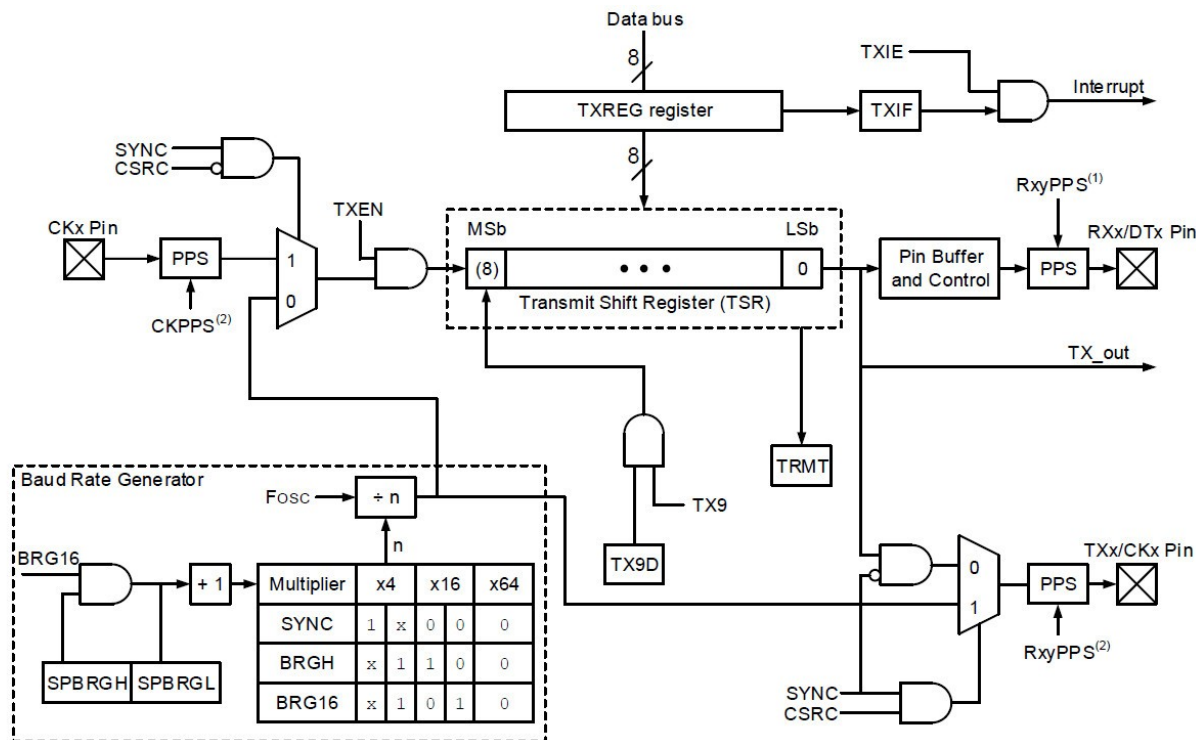
Dans certaines phases de debug et de test, nous pouvons être amenés à utiliser des outils matériels d'analyse des trames circulant sur les bus de communication externes au MCU (broches Tx et Rx pour une liaison série asynchrone). Nous utilisons par exemple à l'école des solutions conçues en France par la société IKALOGIC ainsi que des options d'analyse proposés avec les oscilloscopes.





#### 4.4. Module périphérique UART sur PIC18

## Transmetteur UART liaison série asynchrone



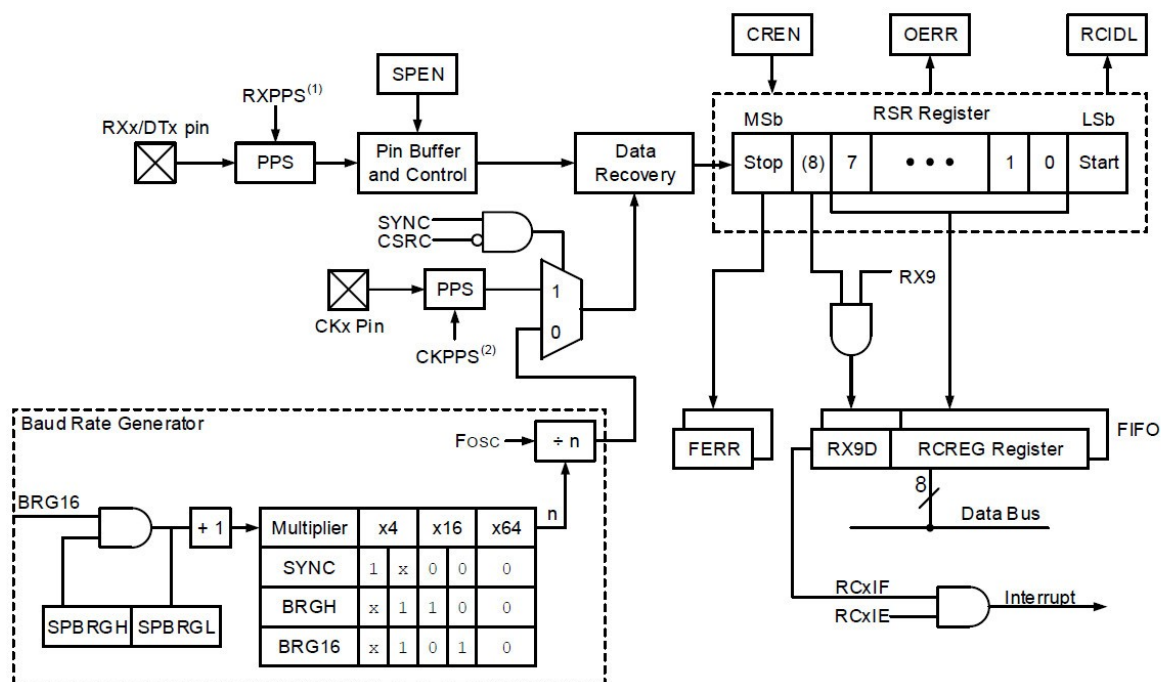
Le MCU PIC18F27K40 utilisé en TP intègre deux UART (UART1 et UART2). Les deux UART seront configurés et utilisés durant cet enseignement. Ces deux modules UART sont identiques. Seuls les noms des registres diffèrent (nommage avec indice 1 ou 2). Le schéma bloc ci-dessus présente la structure du transmetteur de ces UART. Petit exercice, entourer sur le schéma les fonctions matérielles suivantes :

- *Registre à décalage de transmission (envoi bit après bit cadencé sur l'horloge de référence)*
- *Registre de travail (pour écriture de la payload depuis l'application) permettant de charger le registre à décalage de transmission*
- *Ensemble assurant la référence d'horloge et donc le débit de la communication (Baud Rate Generator)*
- *Broche Tx de sortie*
- *Signal d'interruption IRQ envoyé au CPU (après envoi d'une information)*

Ouvrir la datasheet des processeurs PIC18F7K40 et parcourir le chapitre 27 relatif aux UART (EUSART Enhanced Universal Synchronous Asynchronous Receiver Transmitter) afin d'observer la configuration des registres. Prenons l'exemple de l'UART1, sachant que l'UART2 possède une stratégie de configuration et d'utilisation similaire. L'UART 1 possède 2 registres de configuration récepteur/transmetteur (RC1STA et TX1STA), 3 registres de configuration du débit à l'image de la configuration d'un Timer (BAUDCON1, SP1BRGH et SP1B1GL) et deux registres de travail pour la gestion des payload (TX1REG et RC1REG). Analysons une partie du registre 8bits TX1STA chargé de configurer le transmetteur. Nous pouvons observer sur le schéma ci-dessus que le champ TXEN (Transmitter Enable) permet d'appliquer la référence d'horloge au registre à décalage de transmission (TSR ou Transmit Shift Register) et autorise donc une transmission. Le champ SYNC (Synchronous) permet potentiellement d'utiliser une référence d'horloge externe, comme de sortir sur broche cette même référence et de transformer la communication asynchrone en communication synchrone.



## Récepteur UART liaison série asynchrone



Le schéma bloc ci-dessus présente la structure du récepteur UART sur PIC18. Petit exercice, entourer sur le schéma les fonctions matérielles suivantes :

- *Registre à décalage de réception (récupération bit après bit de la trame de communication)*
- *Registres en FIFO (First In First Out) de travail pour la réception (pour lecture de la payload par l'application)*
- *Ensemble assurant la référence d'horloge et donc le débit de la réception (Baud Rate Generator)*
- *Broche Rx d'entrée*
- *Signal d'interruption IRQ envoyé au CPU (après réception d'une information)*

Ouvrir la datasheet des processeurs PIC18F<sub>x</sub>7K40 et parcourir le chapitre 27 relatif aux UART (EUSART Enhanced Universal Synchronous Asynchronous Receiver Transmitter) afin d'observer la configuration des registres. Analysons une partie du registre 8bits RX1STA chargé de configurer le récepteur. Nous pouvons observer sur le schéma ci-dessus que le champ SPEN (Serial Port Enable) permet de valider ou pas la connexion physique de la broche au registre à décalage de réception. Nous pouvons également constater que le récepteur est chargé de détecter les erreurs. Si une erreur de communication est détectée, alors l'un des champs FERR (Frame Error) et OERR (Overrun Error) est activé par le récepteur. Dans tous les cas, si une erreur de communication est détectée ou constatée dans l'application, la meilleure stratégie reste de demander un renvoi à l'émetteur.

### Receive Status and Control Register

	7	6	5	4	3	2	1	0
Bit	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
Access	R/W	R/W	R/W	R/W	R/W	RO	R/HC	R/HC
Reset	0	0	0	0	0	0	0	0

### 4.5. Configuration du module UART sur PIC18

Attention, le configuration présentée ci-dessous n'est pas celle demandée dans les TP. Elle sera donc à adapter. L'exemple suivant présente une séquence assembleur de code configurant le transmetteur de l'UART1 avec un débit de 115200Bd/s pour un horloge système de 64MHz. Une fois configuré, le caractère 'D' est transmis en respectant le protocole d'une liaison série asynchrone sur la broche TX1/RC6 du processeur. Le bit ou flag TRMT (TSR Register is Empty) est mis à jour par l'UART1 et permet de savoir si des bits restent présents dans le registre à décalage de transmission (registre TSR).

```
; UART1 Transmitter configuration :
; uart1 enable, asynchronous mode
; 8bits data, no parity, high baudrate
```

```
MOVLW      0b00100100
MOVWF      TX1STA
```

```
; UART1 BaudRate Generator configuration :
; 16bits baudrate mode
; baudrate 115.200KBps (64MHz CPU clock)
```

```
MOVLW      0b00001000
MOVWF      BAUDCON1
MOVLW      0x8B
MOVWF      SP1BRG
MOVLW      0x00
MOVWF      SP1BRGH
```

```
; send 'D' ASCII code
; 'D' = 68 = 0x44
```

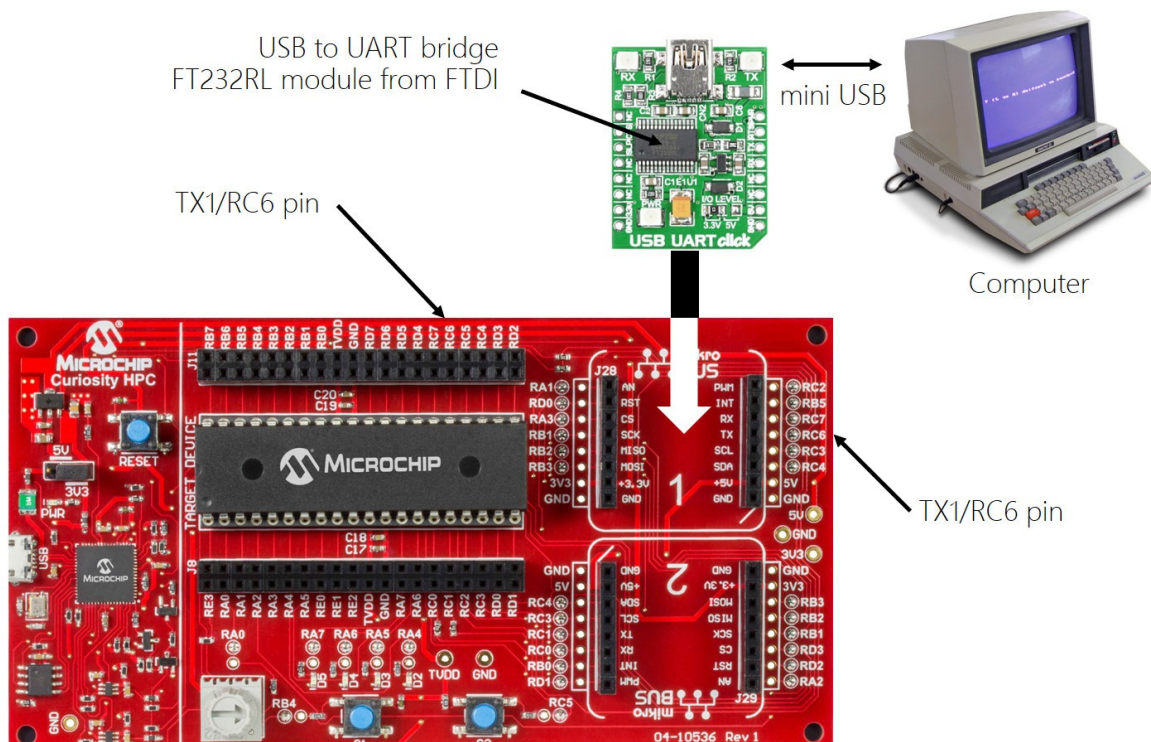
```
MOLW      0x44
MOWF      TX1REG
```

```
; wait for the end of transmission...
```

```
Label:
BTFSS     TX1STA, TRMT
GOTO      Label
```

```
; ... This program send 'D' character by UART1
; The transmission end after 86.806 us
; 115200 bit/s = 8,6806 us/bit
; Frame length : 10 bits
; 1 bit start + 8 bits payload + 1 stop
```

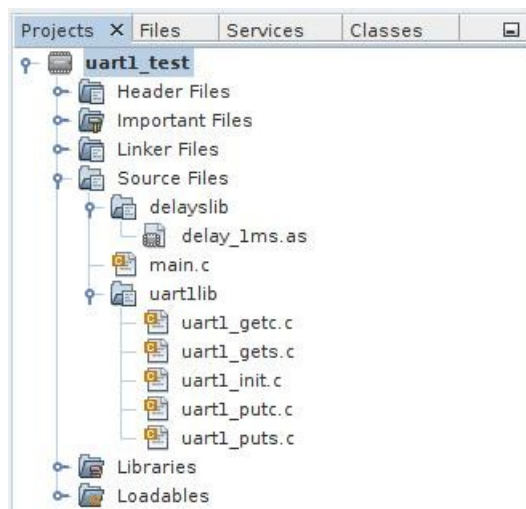
A notre époque, la plupart des ordinateurs modernes ne disposent plus de port COM avec connecteur DB-9 pour liaison série. Nous utilisons généralement des modules passerelles USB vers UART (TTL 0-5V) assurant une transposition de protocole mais ne modifiant pas la donnée échangée. Nous parlons alors de bridge ou de passerelle (exemple de votre box internet, par exemple ADSL vers WIFI). Un bridge UART1 vers UART2 sera d'ailleurs développé en TP. Sur le marché des modules "USB to UART", la société FTDI s'est spécialisée sur ce type de solution.



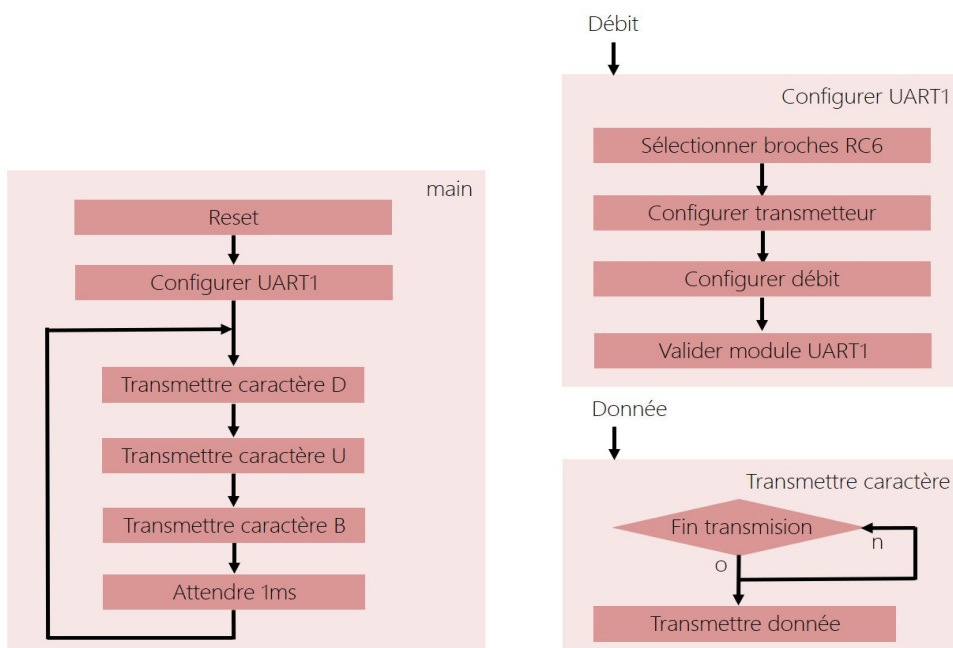
### 4.6. Module périphérique UART1 en transmission

Nous allons maintenant découvrir les techniques d'une communication série asynchrone entre un processeur MCU PIC18F27K40 et un ordinateur équipé d'un terminal asynchrone dédié (TeraTerm, PuTTY, minicom, etc).

- Créer un projet MPLABX nommé *uart1\_test* dans le répertoire *disco/bsp/uart1/test/pjct*. Inclure les fichiers *bsp/uart1/test/main.c*, *bsp/common/delay\_1ms.c* et *uart1\_init.c*, *uart1\_putc.c*, *uart1\_puts.c*, *uart1\_getc.c*, *uart1\_gets.c* présents dans le répertoire *bsp/uart1/src/*. S'assurer de la bonne compilation du projet. S'aider des tutoriels dans *mcu/tp/doc/tutos*



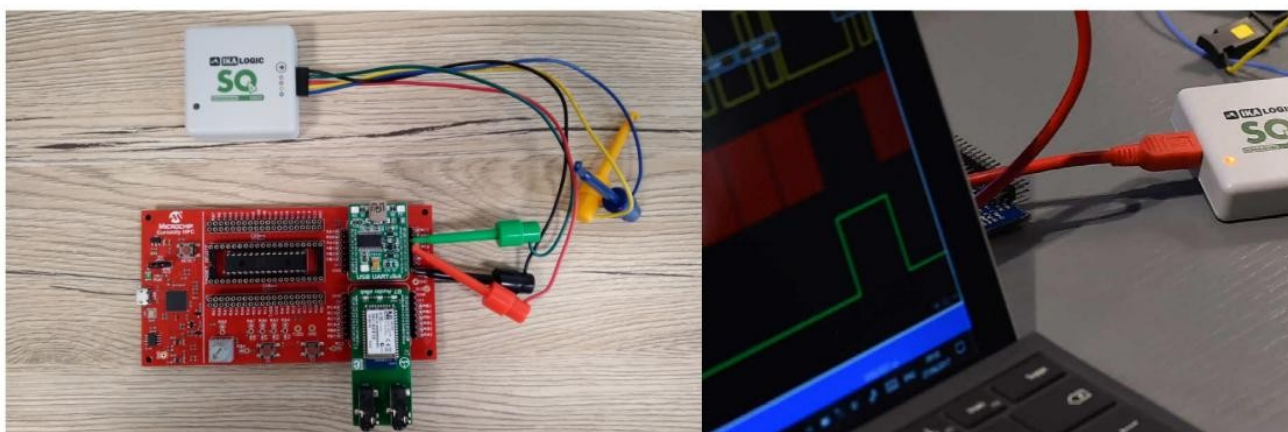
- Modifier les sources *uart1\_init.c* et *uart1\_putc.c* afin de configurer l'UART1 pour que le périphérique puisse implémenter une communication avec un débit de 9600Bd/s et respectant la configuration spécifiée dans le fichier d'en-tête *bsp/uart1/include/uart1.h*. S'aider de la documentation (cartouche doxygen) de la fonction d'initialisation dans ce même header. Dans un premier temps, nous ne configurons que le transmetteur et le générateur de débit, le tout sans gestion d'interruption.
- Développer une application de test implémentant le logigramme suivant. Cette technique d'envoi se nomme *polling* ou *scrutation* et ne nécessite aucune configuration d'interruption.



### Test sur carte physique Curiosity HPC

- Une fois validé, désélectionner le simulateur et sélectionner à nouveau la carte Curiosity pour les tests à venir sur carte physique
- Ouvrir les propriétés du projet : *fenêtre Projects > clic droit sur le nom du projet > Properties*
- Sélectionner la carte Curiosity HPC : *Hardware Tool > Microchip Kits > Curiosity > Apply > OK*
- Observer à l'oscilloscope le signal envoyé sur la broche RC6 (UART1 Tx) du MCU puis représenter ci-dessous les trames transmises par l'UART1 (caractères 'D' 'U' 'B' et attente de 1ms).

### Analyseur Logique IKALOGIC SQ50 ou SQ100



Contrairement à un oscilloscope, un analyseur logique est dédié à la capture et l'analyse de signaux logiques TOR (Tout ou Rien) le plus souvent captés lors de communications entre systèmes d'information (MCU, capteurs, actionneurs, ordinateur, etc). Nous allons ici utiliser un analyseur conçu en France par la société IKALOGIC en utilisant leur application pour ordinateur et tablette nommée SKANASTUDIO.

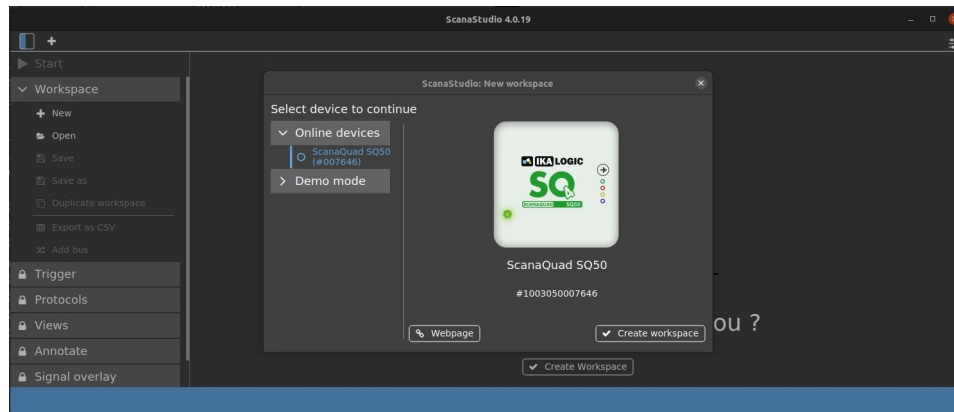


Pour rappel, ne jamais toucher un composant électronique avec les doigts pour des problème d'ESD (Electro-Static Discharge) ou de décharge électrostatique. Toujours manipuler les cartes en les tenant par les côtés/tranches. De même, lorsqu'avec une pince grippe fil vous essayer d'accrocher une broche (cf. sondes SQ50/100), toujours être prudent à ne pas réaliser de court-circuit entre la masse (GND) et l'alimentation (+5V ou +3,3V). Sinon, vous pouvez détruire le régulateur de tension +5V vers +3,3V.

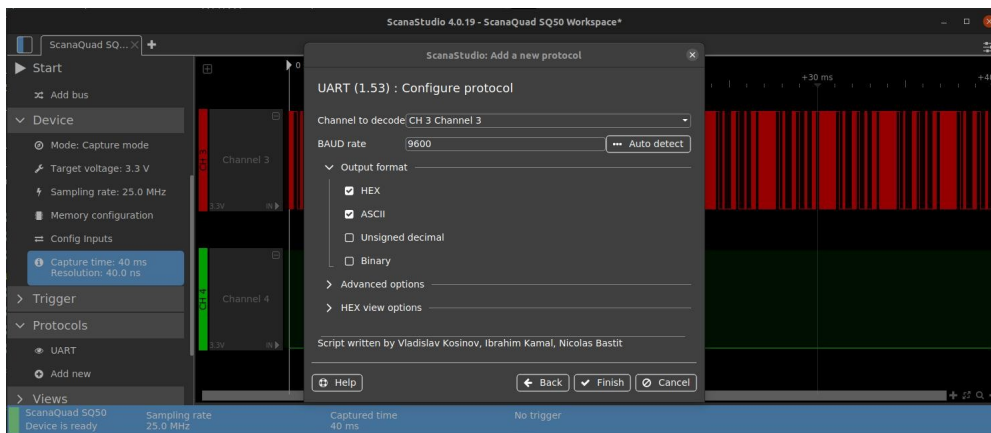
- Connecter la sonde logique IKALOGIC SQ50 ou SQ100 à la carte Curiosity HPC hors tension comme ci-dessus :
  - Fil NOIR sur GND
  - Fil ROUGE sur RC6
  - Fil VERT sur RC7



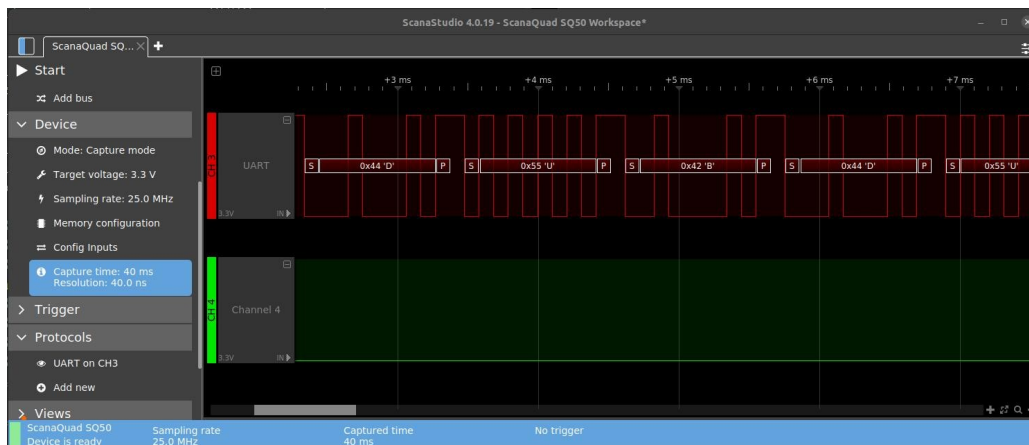
- Exécuter l'application SkanaStudio : *C:\Program Files (x86)\ScanaStudio\ScanaStudio.exe*
- Workspace > *New*
- Online devices > *ScanaQuad SQ50 ou SQ100* > *Create workspace*



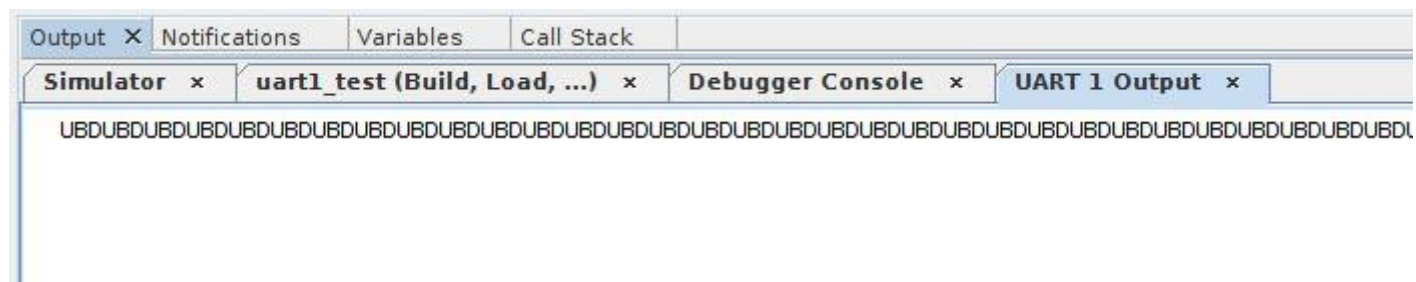
- Protocols > Add new
- Select protocol to continue > UART
  - Channel to decode : *CH 3 Channel 3*
  - BAUD rate : *9600*
  - Finish



- Observer la sortie, redimensionner l'échelle temporelle, renommer le canal, sauvegarder la configuration, etc



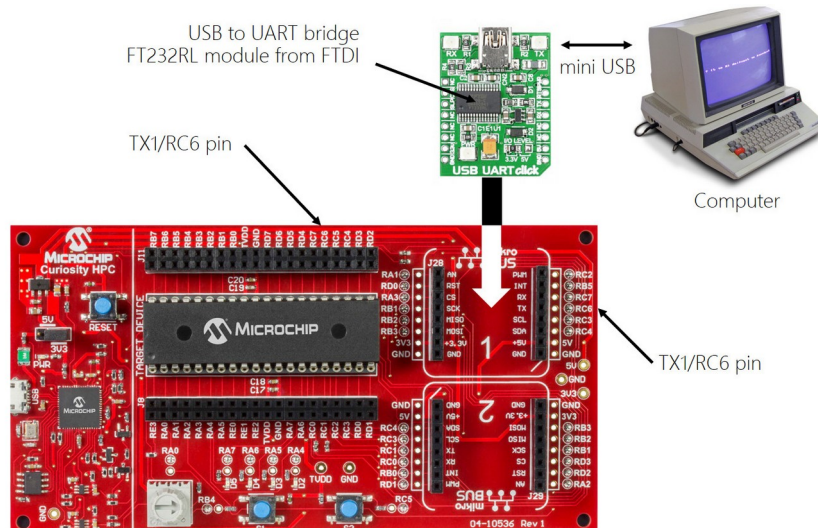
### Test sur simulateur MPLABX IDE (sans carte de développement)



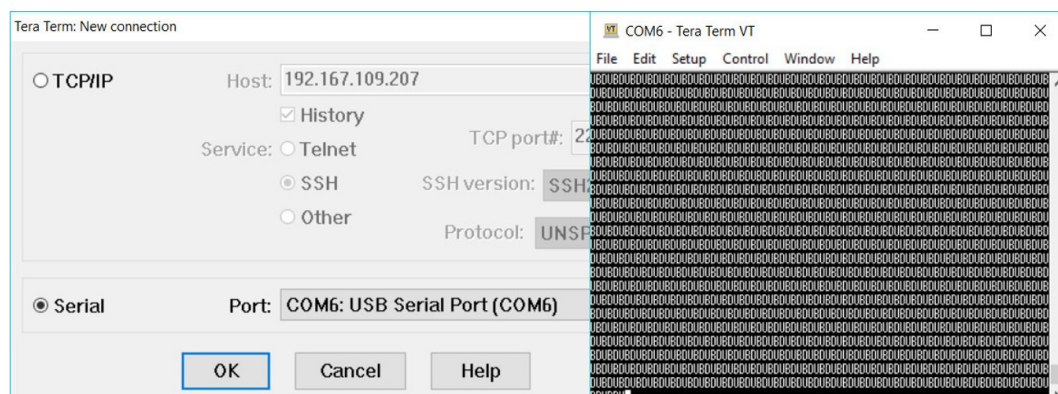
- Ouvrir les propriétés du projet : *fenêtre Projects* → *Clic droit sur le nom du projet* → *Properties*
- Sélectionner le simulateur : *Hardware Tool* → *Simulator* → *Apply*
- Conf : *[default]* → *Simulator*
- Activer l'UART1 sur le simulateur de MPLABX : *Option categories* → *UART1 IO Options* → *Enable UART1 IO* → *Apply* → *OK*
- Compiler et exécuter le programme : *fenêtre Projects* → *clic droit sur le nom du projet* → *Debug*
- Arrêter la simulation en cliquant sur le bouton carré rouge STOP (ou [Shift] + [F5] )
- Observer la fenêtre de sortie UART 1 Output

### 4.7. Terminal de communication série sur ordinateur

Nous allons maintenant valider nos développements en déployant une communication série avec un ordinateur. Vérifier que le module *click board* de *Mikroelektronika USB to UART* soit bien physiquement connecté au socket *mikro BUS 1* dédié sur la carte Curiosity HPC. Vérifier également la connexion en USB avec l'ordinateur.



- Ouvrir un terminal de communication série sur ordinateur et valider le bon fonctionnement du programme. Par exemple, utiliser TeraTerm sous Windows :
  - Exécuter l'application : `C:\Program Files (x86)\teraterm\termpro.exe`
  - Sélectionner le terminal série : *Serial* → *sélectionner votre interface de communication* → OK
  - Configurer le port série : *Setup* → *Serial Port...* → *valider la configuration de la communication série* → OK
  - Il ne reste plus qu'à observer les retours dans la console. De même, tout caractère saisi sur cette même console sera envoyé par le module série virtuel depuis l'ordinateur (port COM sous Windows) vers notre carte de développement*



### 4.8. Transmission de chaînes de caractères

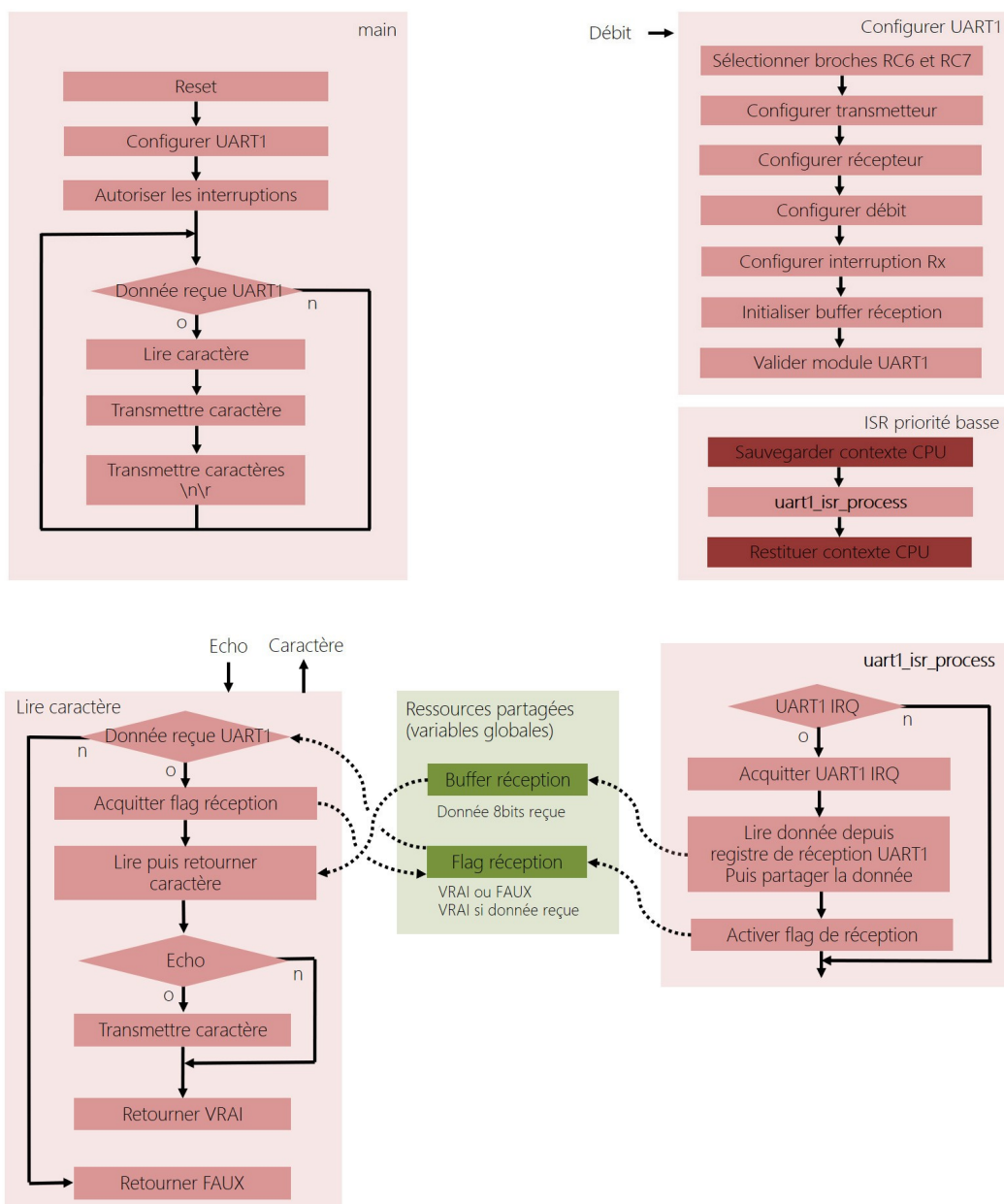
- Modifier le fichier source `uart1_puts.c` puis de valider l'envoi de chaînes de caractères par UART1. Modifier la fonction `main` du programme en conséquent

```
uart1_puts("DUB");
```



### 4.9. Module périphérique UART1 en réception

- Modifier les sources `uart1_init.c` et `uart1_getc.c` afin de configurer l'UART1 pour que le périphérique puisse implémenter la réception de données en respectant la configuration spécifiée dans le fichier d'en-tête `bsp/uart1/include/uart1.h`. S'aider de la documentation (cartouche doxygen) de la fonction d'initialisation dans ce même header. Nous aurons à configurer l'interruption en réception en priorité basse ainsi que le récepteur de l'UART1. Pour information, en général sur MCU pour acquitter une interruption en réception pour un UART il nous suffit de lire et donc vider le registre de réception. C'est le cas sur MCU PIC18. Attention, les fonctions du fichier `uart1_getc.c` doivent probablement être les fonctions les plus complexes à écrire de la trame de TP. Être donc attentif à cette partie de la trame !
- Quelle est la broche de réception Rx utilisée par défaut par l'UART1 sur carte Curiosity HPC ?
- Développer une application de test implémentant le logigramme suivant. Lire les conseils et informations sur la page suivante



L'utilisation de fonctions d'interruption ou ISR dans une application amène un nouveau paradigme, celui de la programmation événementielle. En effet, nous ne pouvons pas prédire le moment de réveil d'une ISR. Une routine d'interruption est donc entièrement indépendante de la fonction main. Pour la première fois à l'école, nous allons devoir impérativement utiliser des variables globales (ressources partagées) afin d'assurer l'échange d'information entre ISR et les fonctions appelées depuis le main. Nous aurons besoin d'échanger deux informations :

- Variable globale statique mémorisant la donnée reçue sur 8bits (cf. logigramme - Buffer réception)
- Variable globale statique mémorisant l'indicateur booléen (flag tout ou rien) spécifiant si une donnée est reçue (cf. logigramme - flag réception)

En langage C, toujours déclarer une variable globale dans un fichier source, le plus souvent dans le fichier source où est présent l'écrivain. De même, constater que le code de la fonction d'interruption n'est pas directement placé dans l'ISR. Ceci est lié à notre technologie processeur PIC18 et au fait qu'elle ne propose que deux fonctions d'interruptions de priorité haute ou basse pour tous les périphériques du MCU. Par exemple, dans notre ISR de priorité basse, nous n'appelons pour le moment qu'une fonction de traitement pour l'UART1 (uart1\_isr\_process). Mais à l'avenir, nous implémenterons plusieurs fonctions de traitement (UAR1, UART2, etc tout autre périphérique nécessaire à l'application). Cette technique nous permettra de garder visible nos ISR (priorités haute et basse) à côté de la fonction main au sein de nos applications. Notre programme gagne alors en simplicité, clarté et lisibilité.

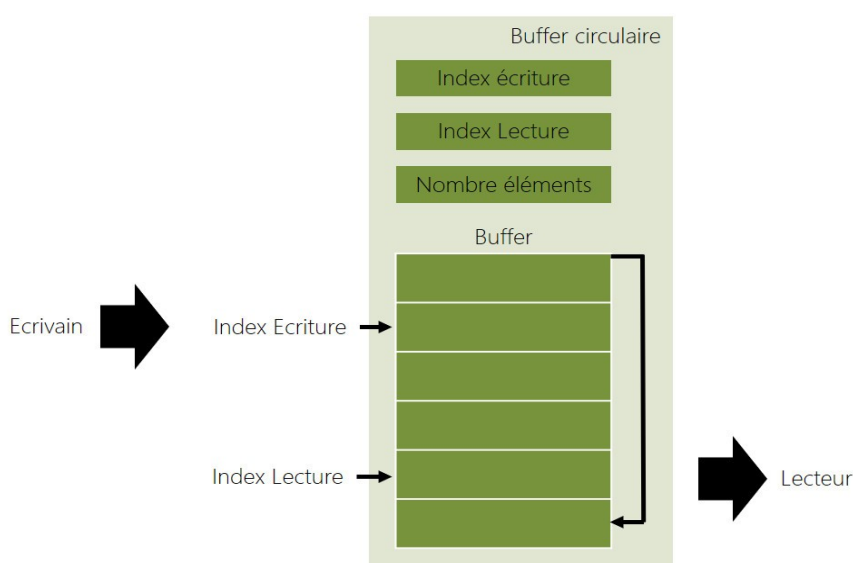


- Valider le bon fonctionnement de votre programme en utilisant une console série sur ordinateur.
- Après validation, envoyer maintenant le contenu d'un fichier texte depuis la console. Voici la procédure sous TeraTerm :
  - *Se placer sur la console TeraTerm*
  - *File → Send file...*
  - *Sélectionner le fichier texte suivant disco/bsp/uart1/test/rx\_test\_file1.txt-> Ouvrir*
- Pourquoi n'observe-t-on qu'un caractère sur trois en retour sur la console ?
- Proposer des solutions à ce problème

### 4.10. Buffer circulaire de réception

Actuellement, le buffer d'échange entre l'ISR et la fonction de lecture de caractère ne fait qu'un octet. Nous allons le remplacer par un buffer de taille configurable (à la compilation) géré circulairement. Nous pouvons observer une définition de structure représentant un objet buffer circulaire dans le fichier d'en-tête *uart1.h*.

Dans notre cas, l'écrivain sera l'ISR et le lecteur la fonction *getc*. Lorsque l'écrivain écrit dans le buffer, il incrémente l'index d'écriture ainsi que le nombre d'éléments présents dans le buffer. Si le buffer est plein, l'écrivain stoppe les écritures. Les données reçues sont alors perdues. Lorsque que l'index d'écriture pointe la fin du buffer, il bascule au début de buffer en position initiale. Il est alors géré circulairement. Il en sera de même pour l'index de lecture. A chaque récupération de donnée correspondant à une lecture dans le buffer circulaire, le lecteur incrémente l'index de lecture et décrémente le nombre d'éléments présents dans le buffer. Si le buffer est vide, le lecteur passe son tour et attend une nouvelle écriture.



- Implémenter un buffer circulaire de 20 éléments entre l'ISR et fonction *getc*. Valider le fonctionnement du programme par envoi du fichier texte *rx-test-file1.txt* depuis le terminal de communication série.
  - Se placer sur la console *TeraTerm*
  - *File* → *Send file...*
  - Sélectionner le fichier texte suivant *disco/bsp/uart1/test/rx\_test\_file1.txt* → Ouvrir
- Après validation, envoyer maintenant le contenu du fichier texte *rx-test-file3.txt* depuis la console. Pourquoi certains caractères sont-ils perdus durant la transmission ?
- Proposer des solutions à ce problème.

### 4.1.1. Contrôle de flux logiciel

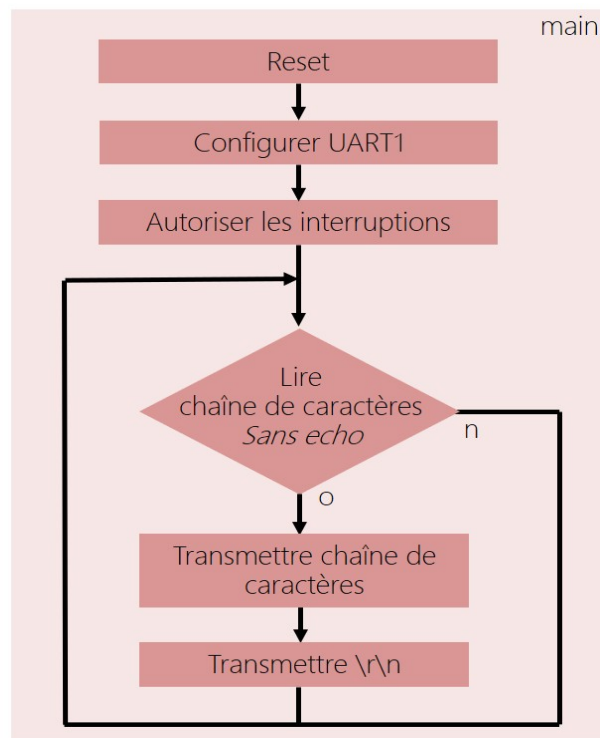
A ce niveau des développements, deux solutions s'offrent à nous si nous ne souhaitons plus perdre de données à la réception. La première consisterait à étendre la taille du buffer circulaire. Néanmoins cette solution ne résout pas le problème mais repousse seulement les limites. Pour information, selon l'utilisation de l'UART dans une application, ces limitations peuvent être amplement suffisantes. Néanmoins, une solution optimale serait d'implémenter un contrôle de flux matériel ou logiciel.

Le concept de contrôle de flux est simple. Si le buffer de réception est plein, le récepteur demande explicitement à l'émetteur d'arrêter de transmettre. Dès que suffisamment de place a été libérée dans son buffer circulaire de réception, le récepteur prévient l'émetteur qu'il peut reprendre sa transmission. Ce contrôle des flux de communication peut être matériel ou logiciel :

- *Contrôle de flux matériel* : Utilisation de deux broches et conducteurs physiques complémentaires (broches CTS et RTS). Les deux signaux physiques sont croisés et permettent par simple logique booléenne aux récepteurs de chaque côté de la liaison série de stopper ou de valider les communications.
- *Contrôle de flux logiciel* : Sans utiliser de broches ni conducteurs physiques complémentaires, le contrôle de flux se fait par envoi de caractères spéciaux (XON ou 0x11 et XOFF ou 0x13). Si l'émetteur reçoit le caractère XOFF, il stoppe les transferts. Dès qu'il recevra le caractère XON, il reprendra les échanges.
- Implémenter et valider un contrôle de flux logiciel. La solution est simple, moins de 10 lignes de codes en tout. Ne pas se perdre donc dans l'implémentation. Les limites seront les suivantes :
  - *Envoi du caractère XOFF depuis l'ISR (écrivain) dès que le buffer circulaire de réception s'approche d'être plein à 4 éléments de sa capacité totale. La taille minimale du buffer circulaire sera fixée à 6 éléments.*
  - *Envoi du caractère XON depuis la fonction getc (lecteur) dès que la moitié du buffer a été vidée par l'application.*
- Répéter l'envoi et la réception du fichier texte *rx-test-file3.txt*
  - *Se placer sur la console TeraTerm*
  - *File → Send file...*
  - *Sélectionner le fichier texte suivant disco/bsp/uart1/test/rx\_test\_file3.txt-> Ouvrir*
  - *Normalement, la magie devrait opérer, félicitations !*

### 4.12. Réception de chaînes de caractères

- Modifier le fichier source `uart1_gets.c` afin d'assurer la réception de chaînes de caractères. Depuis la console, une action sur la touche Entrée (Enter ou '\r') signifiera la fin d'une saisie de chaîne de caractères. A la réception, la fonction devra remplacer le caractère '\r' par un '\0' afin de construire une chaîne de caractères. Si la chaîne de caractères capturée dépasse une taille fixée en entrée de fonction, l'écriture du tableau de destination se stoppera et se clôturera également par l'écriture d'un '\0'.
- Développer une application de test implémentant le logigramme suivant :

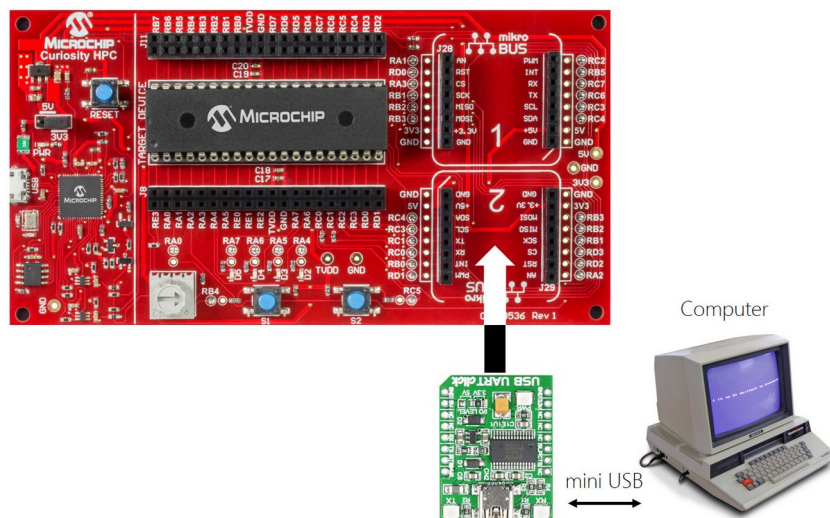


- Valider le bon fonctionnement de votre programme en utilisant une console série sur ordinateur.
- Envoyer maintenant le contenu du fichier texte `rx-test-file3.txt` depuis la console. *Si le contrôle de flux logiciel est implémenté ainsi qu'un buffer circulaire de taille suffisante, alors la magie devrait opérer ! Vous voilà apte à communiquer avec un ordinateur distant, bravo !*

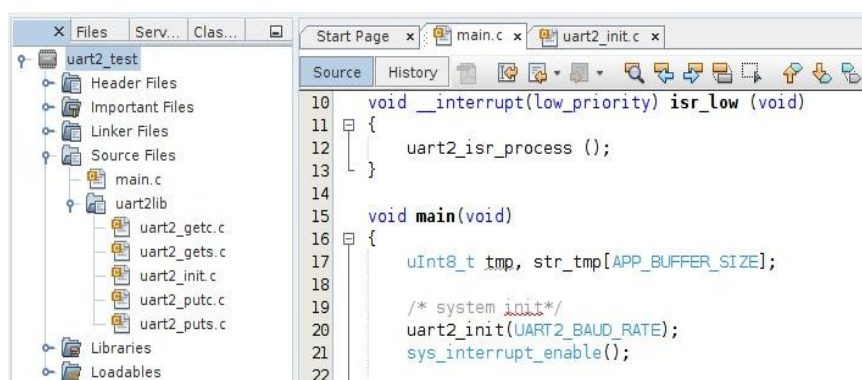
Nous venons de clôturer l'une des parties les plus techniques de la trame de TP. Synthétisons les développements, tests unitaires et validations réalisés durant les derniers exercices autour du périphérique de communication UART :

- Fonction de configuration du transmetteur et récepteur UART avec interruption à la réception
- Fonction d'envoi de caractères par polling (scrutation par boucle d'attente)
- Fonction d'envoi de chaînes de caractères par polling (scrutation par boucle d'attente)
- Fonction de réception de caractères par interruption
- Fonction de réception de chaînes de caractères par interruption
- Implémentation d'un buffer circulaire de réception
- Implémentation d'un contrôle de flux logiciel à la réception

### 4.13. Module périphérique UART2



- Vérifier que le module *click board* de *Mikroelektronika USB to UART* soit physiquement connecté au socket *mikro BUS 2* dédié sur la carte Curiosity HPC
- Créer un projet MPLABX nommé *uart2\_test* dans le répertoire *disco/bsp/uart2/test/pjct*



- Rétiter l'ensemble des configurations précédentes. Il s'agit quasiment que d'un exercice de recopie avec changement d'indice de 1 en 2. Rétiter les tests et valider la bibliothèque de fonctions pilotes pour l'UART2. Hormis pour la sélection des broches à adapter au module UART 2 (RX2 connecté à RC0 et TX2 connecté à RC1), il s'agit donc essentiellement d'un exercice de renommage de registres et de variables de l'UART1 vers l'UART2. Sous MPLABX (il y en a pour moins de 10mn) :
  - Copier les contenus (définitions) un à un des 5 fonctions pour l'UART1 vers les sources vides des 5 fonctions pour l'UART2 (~5mn), Puis réitérer les opérations qui suivent depuis le projet MPLABX pour l'UART2 ...
  - Edit
  - Replace in Projects
  - Scope → Main Project (*choisir le projet principal affiché en gras sous MPLABX*)
  - Containing Text → *choisir le nom d'un registre UART1 à modifier*
  - Replace With → *nouveau nom du registre pour l'UART2*
  - Continue (*la modification est appliquée à tous le fichiers du projet principal*)
  - Rétiter l'opération pour tous les registres et variables à renommer (~5mn) !

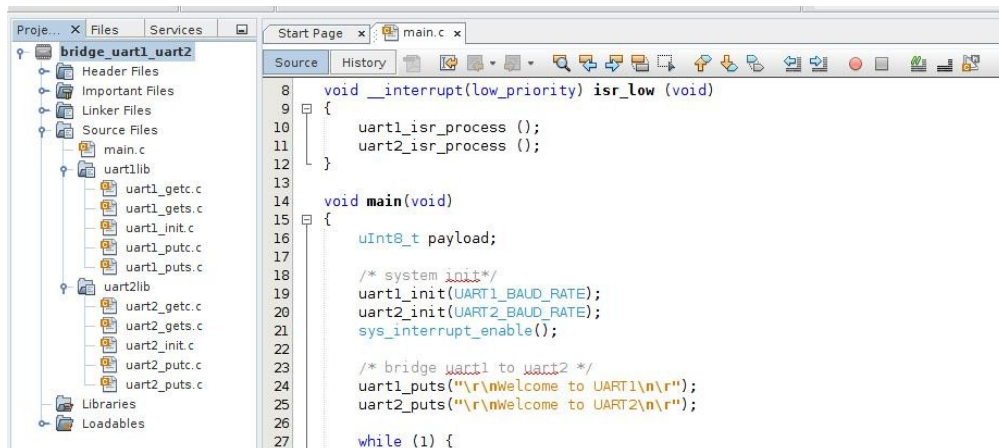


### 4.14. Bridge de communication UART1 vers UART2



Un bridge ou passerelle de communication garantit la non-altération des données (payload) échangées, tout en offrant une transposition de protocole de communication à un autre. Prenons les exemples de votre box internet ADSL ou câble ou fibre (extérieur à l'habitat depuis l'opérateur internet) vers WIFI ou Ethernet (intérieur à l'habitat pour l'utilisateur), des adaptateurs vidéo pour ordinateur VGA vers HDMI (ou DVI ou DisplayPort, etc), etc les applications sont multiples.

- Créer un projet MPLABX nommé *bridge\_uart1\_uart2* dans le répertoire applicatif *disco/apps/bridge\_uart1\_uart2/test/pjct* et ajouter au projet le fichier de test *disco/apps/bridge\_uart1\_uart2/test/main.c*. Ajouter également les sources de vos bibliothèques UART1 et UART2 précédemment développées puis compiler le projet.

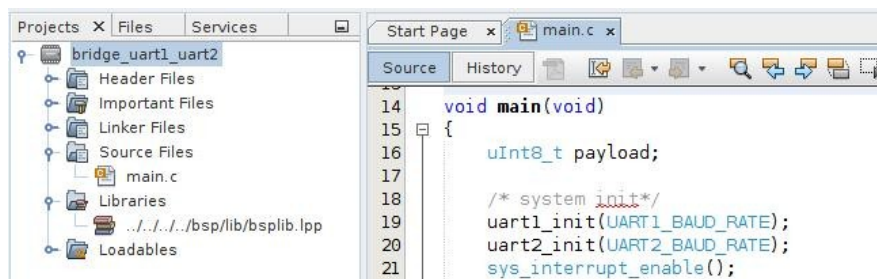


```

8 void __interrupt(low_priority) isr_low (void)
9 {
10     uart1_isr_process ();
11     uart2_isr_process ();
12 }
13
14 void main(void)
15 {
16     uInt8_t payload;
17
18     /* system init */
19     uart1_init(UART1_BAUD_RATE);
20     uart2_init(UART2_BAUD_RATE);
21     sys_interrupt_enable();
22
23     /* bridge uart1 to uart2 */
24     uart1_puts("\r\nWelcome to UART1\r\n");
25     uart2_puts("\r\nWelcome to UART2\r\n");
26
27     while (1) {

```

- Analyser puis tester le programme fourni. Il s'agit d'une application simple de test assurant une passerelle de communication de périphérique UART1/2 à périphérique UART2/1.
- Créer une bibliothèque statique du BSP et réitérer le test (cf. *mcu/tp/doc/tutos*)



```

14 void main(void)
15 {
16     uInt8_t payload;
17
18     /* system init */
19     uart1_init(UART1_BAUD_RATE);
20     uart2_init(UART2_BAUD_RATE);
21     sys_interrupt_enable();

```

Cet exercice conclut bien des concepts en systèmes embarqués et dans le domaine des communications numériques. Nous venons d'atteindre ici même les objectifs strictement minimaux de la trame d'enseignement, félicitations ! Bien entendu, beaucoup reste encore à découvrir dans le domaine de l'embarqué. Comme tout domaine des sciences, l'apprentissage de l'embarqué est sans frontière ni limite ...

Si certains concepts ou points techniques appréhendés jusqu'à présent ne sont pas pleinement assimilés, nous vous invitons à repasser sur la compétence. Car tout ce qui sera vu jusqu'en 3<sup>ème</sup> année en Systèmes Embarqués dépend des bases assises jusqu'à présent. Il en est de même en langage C.



