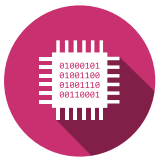



# Récepteur Série

Laboratoire Conception Numérique

# Contenu



# 1 Objectifs

Ce laboratoire vise à pratiquer la conception de circuits numériques synchrones en utilisant une Finite State Machine (FSM). Il se concentre sur la mise en œuvre d'un récepteur série de type RS232, y compris :

- Comprendre les principes de la communication série asynchrone.
- Concevoir et implémenter un registre à décalage et un compteur diviseur d'horloge.
- Créer une FSM dans HDL Designer pour contrôler le processus de réception.

À travers ce laboratoire, les étudiants acquerront une expérience pratique dans la combinaison de la logique séquentielle (FSM, compteurs) avec la gestion des données (registres à décalage) pour réaliser un module de communication complet.

La fonctionnalité du récepteur sera vérifiée en décodant un message envoyé par le $\mu$Processeur Xilinx PicoBlaze développé dans les laboratoires précédents.


Le laboratoire précédent **`$LABO_DIR/CNT`** est un prérequis. Assurez-vous de l'avoir complété, entièrement testé et importé dans votre projet.

# 2 Récepteur Série

## 2.1 Transmission série

Un récepteur série est un circuit numérique qui convertit un flux de données série asynchrone entrant en données parallèles adaptées à un traitement ultérieur. La Fig. 1 montre le timing de la transmission série d'un mot de données, où les données sont transmises bit par bit, en commençant par un bit de départ (start bit), suivi des bits de données (Least Significant Bit (LSB) en premier), et se terminant par un bit d'arrêt (stop bit).

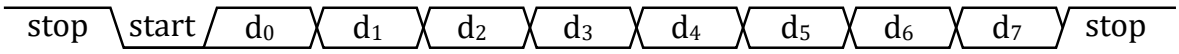



Fig. 1 - Transmission sérielle

## 2.2 Circuit

Le cœur de la conception dans Fig. 2 se compose de :

- Un registre à décalage pour collecter les bits entrants et les convertir en format parallèle.
- Un compteur pour diviser l'horloge du système et se synchroniser avec le débit en bauds des données entrantes.
- Une machine à états finis (FSM) pour contrôler le processus de réception, détecter le bit de départ, échantillonner chaque bit de données au bon moment et valider le bit d'arrêt.

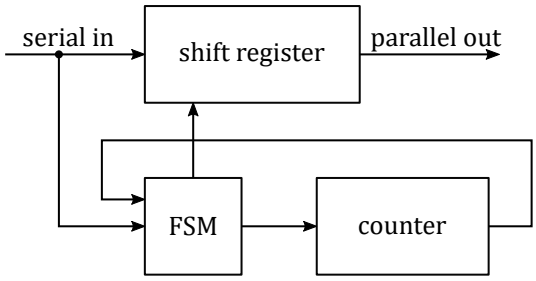



Fig. 2 - Schéma-bloc du récepteur série

La FSM surveille la ligne d'entrée série, détecte la transition indiquant le début d'une trame (start bit) et déclenche des impulsions d'échantillonnage régulières via le compteur. Chaque bit échantillonné est décalé dans le registre jusqu'à ce que l'octet complet soit reçu. La sortie est ensuite mise à disposition sous forme parallèle pour les étapes suivantes du système.

Étudier les éléments et signaux existants dans le bloc **`COM/serialPortReceiver`**.

# 3 FSM

Dans le bloc **`COM/receiverController`**, une FSM implémente la réception des données série. La FSM gère le timing de l'échantillonnage des bits entrants, garantissant que chaque bit est lu au bon moment par rapport au débit en bauds du flux de données entrant.

## 3.1 Éléments de la FSM

Plusieurs éléments sont déjà disponibles, ils sont codés en couleur dans Fig. 3:

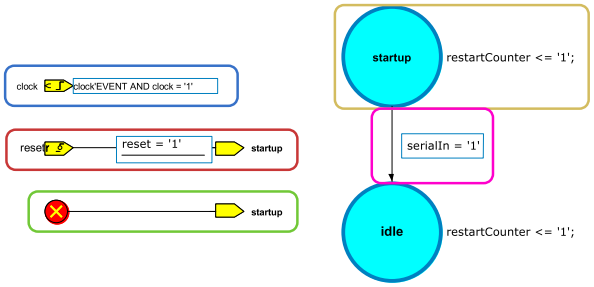



Fig. 3 - Éléments de la FSM donnée

1. Définition du signal d'horloge et de sa condition de déclenchement. Dans ce cas, l'événement d'horloge se produit sur le front montant du signal **`clock`**.
2. Définition du signal de réinitialisation, de la condition de réinitialisation et de l'état après la réinitialisation. Dans ce cas, le signal de réinitialisation est nommé **`reset`**; lorsqu'il est haut, le système se réinitialise et passe à l'état **`startup`**.
3. L'état de récupération utilisé lorsqu'il n'y a pas d'autre affectation d'état valide.
4. États de la FSM. Chaque état nécessite un nom unique. Le code à droite est exécuté pendant que vous résidez dans cet état (*le code est optionnel*). Dans ce cas :

```
restartCounter <= '1';
--   ^          ^   ^ ^
--   |          |   | +-- Symbole de fin requis d'une instruction
--   |          |   +---- Valeur à assigner au signal
--   |          +-------- Symbole d'assignation (attention <= pas =)
--   +------------------- Signal auquel attribuer une valeur
```

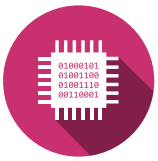
Liste 1 - Code exécuté dans l'état **`startup`**

5. Condition de transition pour relier les états entre eux (*les conditions sont optionnelles*). Dans ce cas :

```
serialIn = '1'
--  ^     ^  ^  ^
--  |     |  |  +-- Aucun symbole de fin requis puisque c'est une condition
--  |     |  +----- Condition pour prendre la transition
--  |     +-------- Comparateur égal (attention = pas ==)
--  +-------------- Le côté gauche de la condition est le signal `serialIn`
```

Liste 2 - Condition pour prendre la transition entre les états **`startup`** et **`idle`**

Cela peut être traduit par « si le signal **`serialIn`** est égal à **`'1'`**, alors l'état changera ».



## 3.2 États et Transitions

La FSM a plusieurs états, chacun représentant une phase spécifique du processus de réception. Les transitions entre ces états sont déclenchées par des événements définis dans les flèches. Si la flèche n'a pas de déclencheur, cela signifie que la transition est toujours active et qu'au prochain cycle d'horloge, elle passe dans tous les cas à l'état suivant.

En haut de l'éditeur se trouvent plusieurs barres d'outils avec toutes les fonctions nécessaires pour éditer la FSM. La première barre d'outils Fig. 4 permet de créer de nouveaux états (cercle bleu) et transitions (flèche noire).

Fig. 4 - Barre d'outils pour créer de nouveaux états ou transitions dans une FSM

## 3.3 Code d'action, déclarations de transition et constructeur d'expressions

Pour toutes les actions ou conditions de transition, vous devez écrire du code Very Highspeed Integrated Circuit Hardware Description Language (VHDL). Pour ce labo, nous n'avons besoin d'écrire que des conditions simples pour les transitions - voir Liste 2 - ou des assignations de signaux simples pour les états - voir Liste 1 -.

La deuxième barre d'outils contient le bouton pour ouvrir le constructeur d'expressions Fig. 5, il vous permet de créer de nouvelles expressions pour les conditions des transitions et le code dans les états.

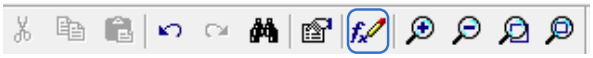


Fig. 5 - Barre d'outils pour ouvrir le constructeur d'expressions

> Pour éditer une action d'état ou une condition de transition, vous pouvez double-cliquer sur l'état ou la transition dans l'éditeur. Cela ouvre un éditeur de code où vous pourrez écrire/modifier le code VHDL pour l'action ou la condition.

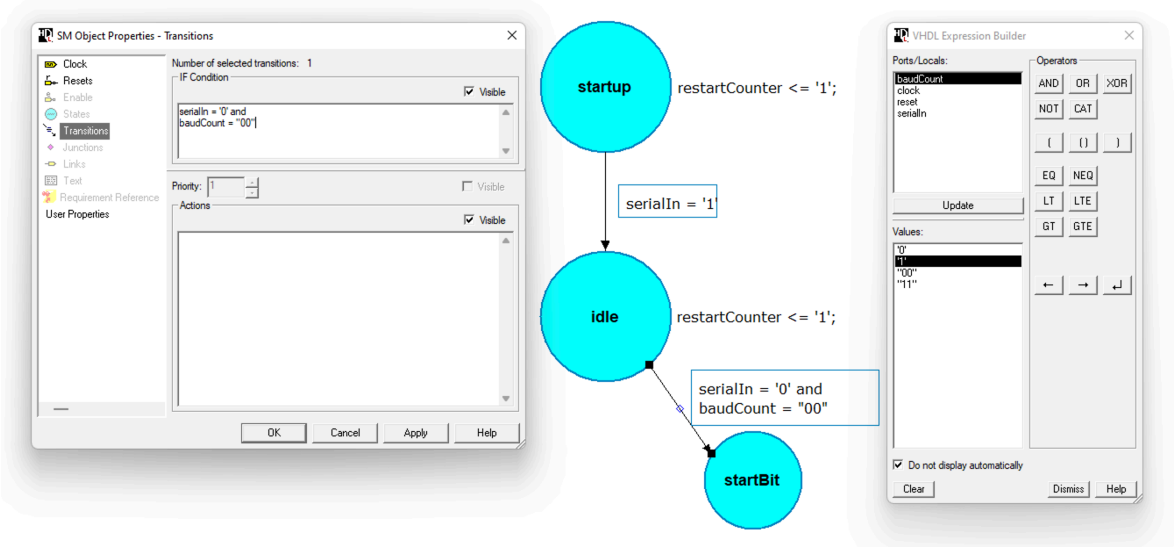


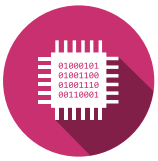

Fig. 6 - Éditeur de code pour une action d'état ou une condition de transition

Une comparaison avec un signal à un seul bit tel que **`serialIn`** ou un signal à plusieurs bits tel que **`baudCount`** est légèrement différente.

1. Les comparaisons ne se font pas avec **`==`** comme dans d'autres langages de programmation, mais seulement avec un **`=`**.
2. La comparaison du signal pour les signaux à un seul bit est entouré de guillemets simples **`'`** et pour les signaux à plusieurs bits de guillemets doubles **`"`**.
3. Vous pouvez combiner plusieurs conditions ensemble avec des opérateurs booléens tels que **`and`**, **`or`** et **`not`**.

Par exemple, pour vérifier si l'entrée série est basse et que le compteur de bauds est zéro, vous pouvez écrire :

```
  serialIn = '0' and baudCount = "00"
--  \_____________/  ^  \_____________/
--         |         |    +-- Comparaison d'un signal à plusieurs bits
--         |         +------- booléen opérateur
--         +----------------- Comparaison d'un signal à un seul bit
```




# 4 Réalisation

Les trois composants principaux du récepteur de port série à implémenter sont :

- **`COM/shiftRegister`** pour collecter les bits entrants et les convertir en format parallèle.
- **`COM/baudrateCounter`** pour diviser l'horloge du système et se synchroniser avec le débit en bauds des données entrantes.
- **`COM/receiverController`** pour contrôler le processus de réception, détecter le bit de départ, échantillonner chaque bit de données au bon moment et valider le bit d'arrêt.

## 4.1 Analyse

Afin de contrôler précisément la réception des données série, le timing des bits reçus est crucial.

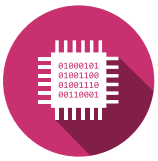
L'envoi sur le port série est réalisé par le processeur $\mu$Processeur Picoblaze développé dans les laboratoires précédents.

Le code utilisé est le code linéaire, sans boucle, tel que donné sous Liste 3:

```
LOAD        s3, FF              ; load stop bit
OUTPUT      s3                  ; output stop bit
LOAD        s3, s3              ; no operation
LOAD        s3, s3              ; no operation
LOAD        s3, s3              ; no operation
LOAD        s3, s3              ; no operation
LOAD        s0, 00              ; load start bit
OUTPUT      s0                  ; output start bit
INPUT       s1                  ; load word to send
OUTPUT      s1                  ; output word, LSB is considered
SR0         s1                  ; shift word, bit 1 -> LSB
OUTPUT      s1                  ; output bit 1
SR0         s1                  ; bit 2 -> LSB
OUTPUT      s1                  ; output bit 2
SR0         s1                  ; bit 3 -> LSB
OUTPUT      s1                  ; output bit 3
SR0         s1                  ; bit 4 -> LSB
OUTPUT      s1                  ; output bit 4
SR0         s1                  ; bit 5 -> LSB
OUTPUT      s1                  ; output bit 5
SR0         s1                  ; bit 6 -> LSB
OUTPUT      s1                  ; output bit 6
SR0         s1                  ; bit 7 -> LSB
OUTPUT      s1                  ; output bit 7
LOAD        s3, s3              ; no operation
OUTPUT      s3                  ; output stop bit
```

Liste 3 - Algorithme linéaire

En lisant le code donné Liste 3, et en regardant le bloc **`COM_test/serialPortReceiver_tb`**, déterminez quel code série est envoyé par le processeur.



Déterminez le nombre de périodes d'horloge nécessaires pour l'envoi d'un bit.

Pour ça, référez vous au code Liste 3, au calcul du baudrate effectué au laboratoire précédent **`CNT`**, ainsi qu'au signal **`serialOut`** du circuit **`COM_test/serialPortReceiver_tb`** lorsque simulé avec le fichier **`$SIMULATION_DIR/COM.do`**.

## 4.2 Développement

Avec toutes les informations recueillies dans la section précédente, vous pouvez maintenant implémenter les trois composants principaux du récepteur de port série.

### 4.2.1 Registre à décalage

Les données arrivent en série, un bit à la fois, et doivent être collectées et stockées dans un format parallèle pour un traitement ultérieur. Un registre à décalage est idéal pour cette tâche, car il peut décaler chaque bit entrant sur un signal de contrôle et fournir l'octet complet en parallèle.

Implémentez le registre à décalage **`COM/shiftRegister`**. Chaque fois que l'entrée doit être décalée, le signal **`shiftEn`** est mis à **`1`**.

### 4.2.2 Compteur de baudrate

Pour échantillonner avec précision les données série entrantes, un compteur est nécessaire pour diviser l'horloge du système jusqu'au débit en bauds des données entrantes. Ce bloc génère une valeur de comptage qui peut être utilisée pour savoir quand échantillonner les bits entrants, où nous en sommes dans la trame reçue et/ou quand réinitialiser pour la trame suivante…
Lors de l'échantillonnage d'un bit, il doit être aussi proche que possible du milieu de la période du bit.

- Déterminez le nombre de bits requis pour le compteur **`COM/baudrateCounter`** en fonction de la manière dont vous l'utiliserez.
- Saisissez le nombre de bits requis dans le système. Pour cela, dans le **`COM/serialPortReceiver`**, double-cliquez sur la définition de la constante **`baudCounterBitNb`** en haut à gauche du schéma et modifiez-la dans la nouvelle fenêtre ouverte sous **`User Declarations`** comme montré sous Fig. 7 puis cliquez sur **`Apply`** :

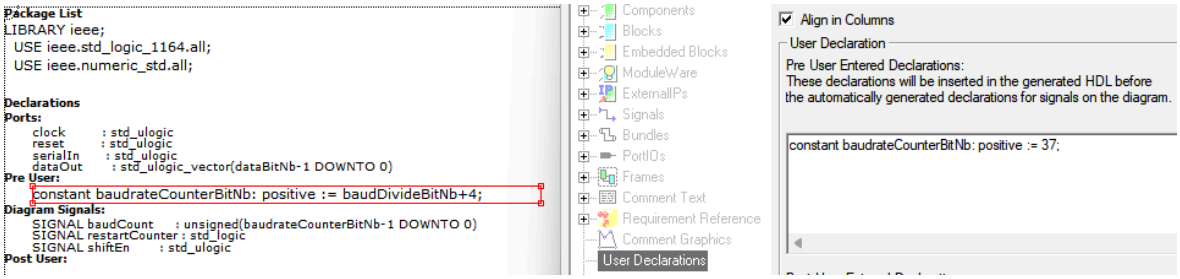



Fig. 7 - Modifier le nombre de bits du compteur système

Implémentez le compteur **`COM/baudrateCounter`**.

- N'oubliez pas de valider/changer le nombre de bits du signal **`baudCount`** sur le **`COM/serialPortReceiver`**.
- Le compteur doit prendre en charge une réinitialisation synchrone avec le signal **`restartCounter`**.

### 4.2.3 Contrôleur de réception (FSM)

La machine à états finis (FSM) est le cerveau du récepteur de port série. Elle surveille la ligne d'entrée série, détecte la transition indiquant le début d'une trame (bit de départ) et déclenche des impulsions d'échantillonnage régulières en fonction de la valeur du compteur. Chaque bit échantillonné est décalé dans le registre jusqu'à ce que l'octet complet soit reçu. La sortie est ensuite mise à disposition sous forme parallèle pour les étapes suivantes du système.

Implémentez la machine à états finis **`COM/receiverController`**. Elle doit détecter le début et la fin d'une trame et générer les signaux **`restartCounter`** ainsi que **`shiftEn`** en conséquence.

## 4.3 Simulation

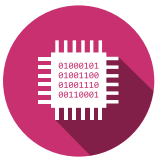
Une fois implémentés, simulez les trois composants avec le banc d'essai **`COM_test/serialPortReceiver_tb`**. Le $\mu$Processeur Xilinx Picoblaze développé dans les laboratoires précédents sera utilisé pour envoyer les données série au **`COM/serialPortReceiver`**.

Simulez le banc d'essai **`COM_test/serialPortReceiver_tb`** avec le fichier de simulation **`$SIMULATION_DIR/COM.do`**.

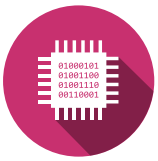
Validez que les différents octets envoyés sont correctement décodés sur le signal **`dataOut`**.

## 4.4 Analyse du système

Maintenant que le récepteur de port série est implémenté et simulé, répondez aux questions suivantes :



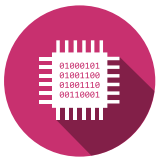

1. Combien de temps faut-il pour qu'un bit soit envoyé par le $\mu$Processeur Picoblaze en termes de temps (c'est-à-dire le débit de transmission) ?
2. Combien de temps faut-il pour recevoir deux données ?
3. Pourquoi les bits de données sont-ils échantillonnés aussi près que possible du milieu de la période du bit ?
4. Que se passe-t-il si le baudrate de l'émetteur ne correspond pas à celui du récepteur ?




# 5 Checkout

C'est la fin du laboratoire, vous avez réussi à construire un système avec plusieurs différents blocs. Avant de quitter le laboratoire, assurez-vous d'avoir accompli les tâches suivantes :

- ☐ Théorie
  - ☐ Vous comprenez comment créer des machines à états FSM dans HDL Designer.
- ☐ Conception du circuit
  - ☐ Les trois blocs **`COM/shiftRegister`**, **`COM/baudrateCounter`** et **`COM/receiverController`** ont été créés et testés.
- ☐ Simulations
  - ☐ Les données envoyées par le **`COM/nanoprocessor`** sont correctement lues et parallélisées par le **`COM/serialPortReceiver`**.
  - ☐ Les bits individuels sont lus aussi près que possible du milieu.
  - ☐ Analyse
    - ☐ Vous avez répondu aux questions d'analyse du système.
- ☐ Documentation et fichiers de projet
  - ☐ Assurez-vous que toutes les étapes (conception, simulations, comparaison) sont bien documentées dans votre rapport de laboratoire.
  - ☐ Enregistrez le projet sur une clé USB ou sur le lecteur réseau partagé (**`\\filer01.hevs.ch`**).
  - ☐ Partagez les fichiers avec votre partenaire de laboratoire pour assurer la continuité du travail.




# Glossaire

***FSM*** **– Finite State Machine** 1, 1, 1, 2, 2, 3, 3, 3, 3, 4, 4, 4, 8, 10

***LSB*** **– Least Significant Bit** 2

***PicoBlaze***: PicoBlaze is a small, 8-bit microcontroller designed by Xilinx for use in FPGAs. It is often used in educational settings to teach basic microcontroller concepts. 1

***VHDL*** **– Very Highspeed Integrated Circuit Hardware Description Language** 4, 4