N7-TP

Ghayad Tala, Sabra Karim

22 septembre 2025

Table des matières

1 Etude de la métastabilité

La métastabilité dans les circuits numériques est une problématique critique, particulièrement dans les systèmes où des signaux asynchrones interfèrent avec des horloges synchrones. Ce phénomène peut entraîner des comportements imprévisibles et compromettre la fiabilité du système.

Le cours *Métastabilité et Synchronisation* introduit les bases théoriques de la métastabilité, des bascules (flip-flops), et des synchroniseurs. Ce TP, en complément, vise à démontrer ces concepts à travers des simulations et des expériences matérielles sur FPGA.

1.1 Procédure Expérimentale

1.1.1 Étape 1 : Simulation de la Métastabilité

Nous avons conçu une bascule D basique avec une entrée asynchrone et une horloge générée par un IP *Clock Wizard* (fréquence de 5 MHz). Dans un premier temps, nous avons testé la fonctionnalité de base de cette bascule pour vérifier le bon fonctionnement du bloc. À noter que cette bascule D est dépourvue de signal de remise à zéro (*reset*).

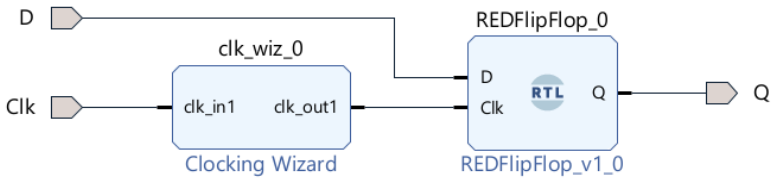

FIGURE 1 – Schéma de la bascule D sans synchronisation

Une fois le fonctionnement de la bascule vérifié, nous avons cherché à mettre en évidence le phénomène de métastabilité avant d'aborder les solutions permettant de le gérer. Pour cela, nous avons consulté la feuille de données (*datasheet*) afin de déterminer les valeurs de T_{setup} et T_{hold}. Ces valeurs varient légèrement selon les sources, mais elles se situent généralement dans le même ordre de grandeur : $T_{setup} + T_{hold} \approx 0.3 - 0.5\,\text{ns}$.

Pour illustrer la métastabilité, nous avons configuré un front montant sur le signal d'entrée D approximativement $0.1 - 0.2\,\text{ns}$ avant le front actif de l'horloge. En simulation comportementale (*behavioral*), la bascule échantillonne correctement le signal D, conformément aux attentes. Cependant, en simulation post-implémentation avec contraintes temporelles, nous avons observé que la bascule reste à 0 pendant deux cycles d'horloge. Cela illustre clairement un cas de métastabilité où l'état de sortie Q n'est pas déterminé.

Pour résoudre ce problème, nous avons introduit la solution classique consistant à utiliser des synchroniseurs, comme détaillé dans les sections suivantes.

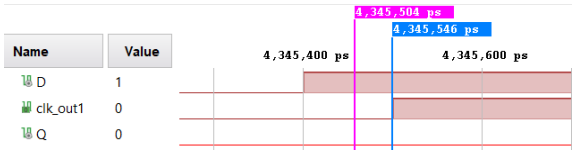

FIGURE 2 – Simulation de la métastabilité (vue générale)

1.1.2 Étape 2 : Implémentation des Synchroniseurs

Pour résoudre le problème de métastabilité, nous avons implémenté un synchroniseur à deux étages, composé de deux bascules D en cascade. Cette structure permet de réduire considérablement la probabilité que l'état métastable initial se propage à la sortie.

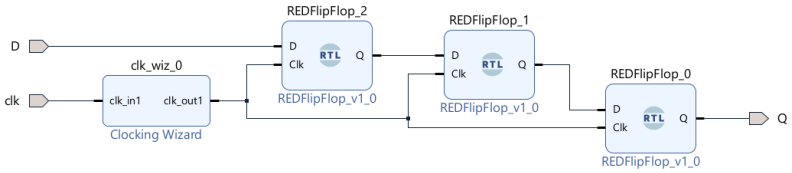

FIGURE 3 – Diagramme bloc du synchroniseur à deux ètages

Le fonctionnement du synchroniseur repose sur les principes suivants :

— La première bascule peut entrer dans un état métastable si l'entrée asynchrone viole les contraintes de synchronisation (T_{setup} et T_{hold}).

— La seconde bascule reçoit la sortie de la première bascule après un cycle d'horloge, ce qui réduit drastiquement la probabilité que la métastabilité persiste.

Ce mécanisme repose sur un compromis :

— **Stabilité accrue :** En augmentant le nombre d'étages dans le synchroniseur, la probabilité que la métastabilité persiste est réduite de manière exponentielle.

— **Délai introduit :** Chaque étage ajoute un cycle d'horloge au délai total de propagation. Il est donc essentiel de trouver un équilibre entre la stabilité requise et les contraintes temporelles du système.

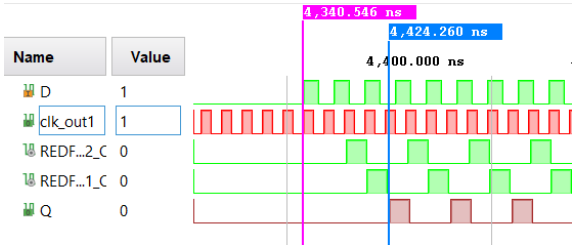

FIGURE 4 – Simulation du synchroniseur montrant la stabilisation de la sortie

Comme le montre la simulation (Figure 4), le synchroniseur garantit une sortie stable Q après deux cycles d'horloge, même en présence de violations des contraintes temporelles à l'entrée. Ce compromis est essentiel pour assurer la fiabilité des systèmes numériques modernes.

2 Moteur Pas à Pas

Les moteurs pas à pas sont largement utilisés dans les applications de contrôle de précision telles que la robotique, les imprimantes 3D et les machines CNC. Ce TP se concentre sur la conception et la mise en œuvre d'un pilote pour moteur pas à pas à l'aide de machines à états finis FSM et de VHDL. Le pilotage sera implémenté sur un FPGA, avec des tests et validations réalisés à l'aide d'un petit moteur pas à pas 28BYJ-48 et d'un module pilote ULN2003.

2.1 Stepper Motor Driver IP

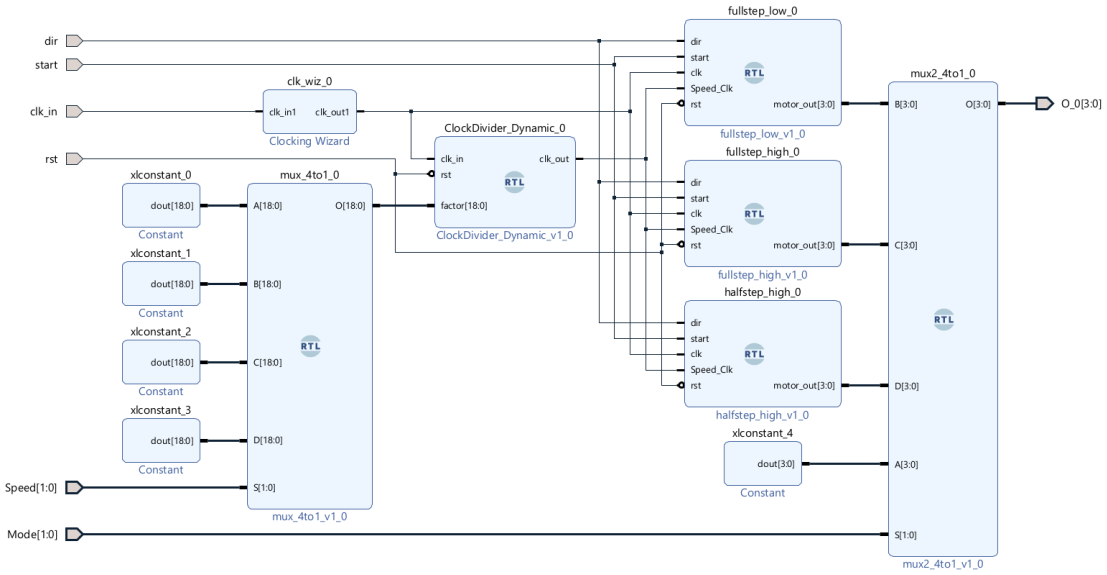

FIGURE 5 – Diagramme des blocs fonctionnels

2.1.1 FSM Design

Cette partie traite la conception d'un pilote IP pour moteur pas à pas en utilisant des machines à états :

- **Pas complet avec faible couple** : Active une bobine à la fois pour un couple minimal mais avec une consommation d'énergie réduite.
- **Pas complet avec couple élevé** : Active deux bobines à la fois pour un couple plus élevé.
- **Demi-pas avec haute précision** : Alterne entre l'activation d'une et de deux bobines pour un contrôle plus précis.

Ce code implémente une FSM pour contrôler un moteur pas à pas en mode Full Step High Torque.

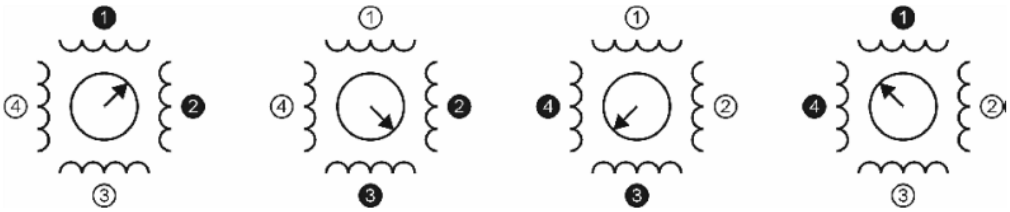

FIGURE 6 – Full Step High Torque Functional States

Nom	Type	Description
`clk`	Entrée	Horloge principale.
`rst`	Entrée	Réinitialisation.
`Speed_Clk`	Entrée	Horloge pour ajuster la vitesse.
`dir`	Entrée	Contrôle de la direction.
`start`	Entrée	Active le moteur.
`motor_out`	Sortie	Contrôle des bobines (4 bits).

TABLE 1 – Description des entrées et sorties.

- **États :**
  - STATE_0 : Bobines 1 et 2 activées (`1100`).
  - STATE_1 : Bobines 2 et 3 activées (`0110`).
  - STATE_2 : Bobines 3 et 4 activées (`0011`).
  - STATE_3 : Bobines 4 et 1 activées (`1001`).
- **Transitions d'états :**
  - Horaire (**dir = 0**) :
    `STATE_0` → `STATE_1` → `STATE_2` → `STATE_3` → `STATE_0`.
  - Antihoraire (**dir = 1**) :
    `STATE_0` → `STATE_3` → `STATE_2` → `STATE_1` → `STATE_0`.

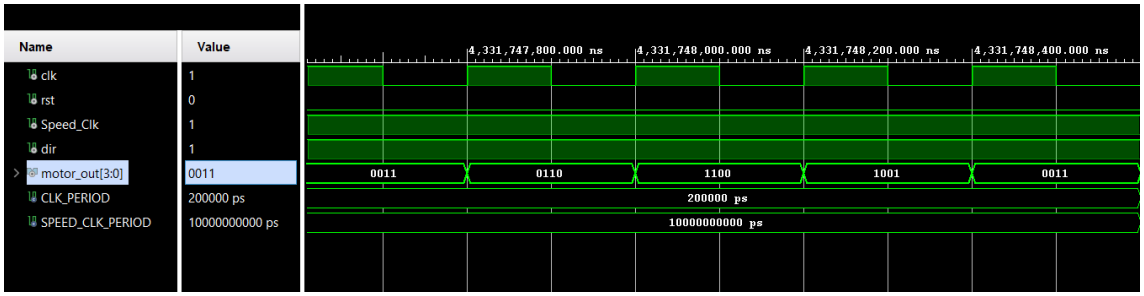

FIGURE 7 – Simulation d'une FSM

Les résultats montrent que la FSM pour le mode **Full Step High Torque** fonctionne correctement. Les transitions d'états sont bien synchronisées avec le signal

`Speed_Clk`, et la sortie `motor_out` suit la séquence attendue pour une rotation horaire (`1100`, `0110`, `0011`, `1001`, puis retour à `1100`). Le signal de direction (`dir = 0`) influence correctement la séquence, confirmant un fonctionnement conforme aux spécifications validant ainsi le comportement de la FSM.

Full Step Low Torque et Half Step High Precision :
Le même schéma de conception a été appliqué pour les autres modes :

- **Full Step Low Torque** : une seule bobine activée à la fois.
- **Half Step High Precision** : alternance entre une bobine et deux bobines activées, pour augmenter la précision angulaire.

Les **FSM** correspondantes ont été développées et vérifiées par simulation, confirmant des transitions d'états correctes et la production de signaux `motor_out` adéquats. Par exemple, la figure ci-dessous illustre la simulation de la FSM en mode *Half Step* :

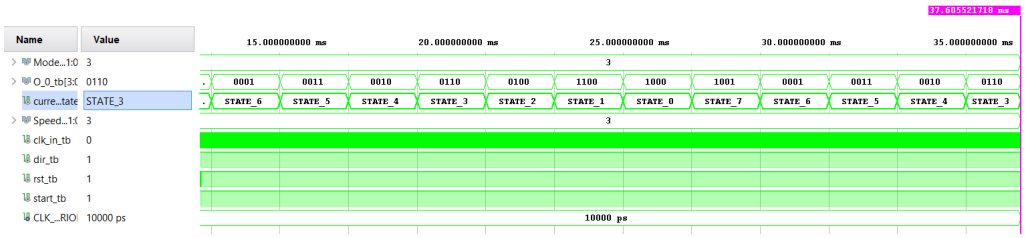

FIGURE 8 – Simulation de la FSM en mode Half Step High Precision

2.2 Clock Divider Dynamique

Dans cette partie, nous avons conçu un diviseur d'horloge dynamique, jouant un rôle critique en tant que "horloge de tic" (*tick clock*), pour maintenir tous les modules synchronisés avec l'horloge principale de 5 MHz. De plus, ce diviseur ajuste la fréquence de sortie pour contrôler la vitesse du moteur pas à pas.

2.2.1 Fonctionnement du Diviseur d'Horloge

Le fonctionnement du diviseur repose sur trois concepts clés :

Division de la Fréquence : Le diviseur d'horloge utilise un compteur pour réduire la fréquence de l'horloge principale `clk_in` influençant directement la vitesse des transitions des FSM. Par conséquent, la vitesse de rotation du moteur pas à pas est contrôlée avec précision en jouant sur cette fréquence divisée. Ce compteur incrémente à chaque front montant de `clk_in` et compare sa valeur à un facteur défini (`factor`). Lorsqu'il atteint - 1, le compteur est réinitialisé et génère un "tic" en activant brièvement le signal `clk_out`. Ce "tic" représente une période de l'horloge divisée, dont la fréquence est déterminée par la valeur de `factor`.

Durée de l'État Haut de `clk_out` : Pour éviter des comportements inattendus dans les machines à états finies (FSM) synchronisées, le signal `clk_out` reste à l'état haut (`'1'`) pendant exactement un cycle d'horloge de `clk_in`. Cette durée contrôlée garantit que chaque FSM ne détecte qu'un seul front montant par période de `clk_out`, éliminant ainsi tout risque d'activations multiples.

2.2.2 Entrées, Sorties et Signaux Internes

Le tableau ci-dessous résume les différents signaux du diviseur d'horloge, leurs types et leurs rôles dans le fonctionnement global.

Nom	Type	Description
`clk_in`	Entrée	Horloge du système (5 MHz)
`rst`	Entrée	Réinitialisation asynchrone.
`factor`	Entrée	Facteur de division dynamique (19 bits)
`clk_out`	Sortie	Signal de sortie synchronisé, agissant comme un "tic"
`counter`	Signal Interne	Compteur pour diviser la fréquence de `clk_in`
`clk_toggle`	Signal Interne	Signal intermédiaire pour générer `clk_out`

TABLE 2 – Description des signaux du diviseur d'horloge

2.2.3 Illustrations et Analyse des Simulations

Durée de l'État Haut de `clk_out` La simulation de la Figure 9 illustre que l'état haut du signal `clk_out` dure exactement un cycle d'horloge de `clk_in` (5 MHz). Cela garantit que les FSM activées par ce signal détectent un seul front montant par période et évitent tout comportement inattendu.

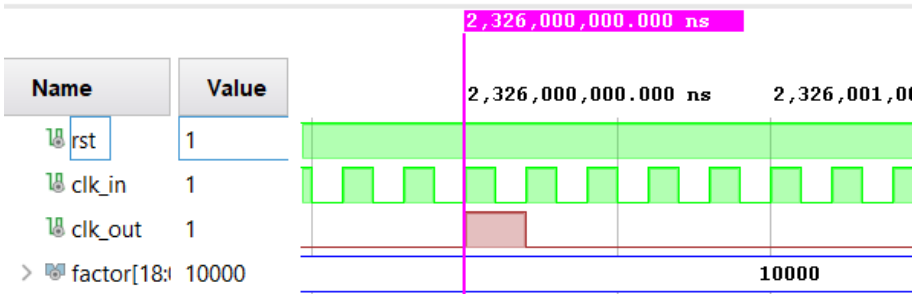

FIGURE 9 – Simulation de la durée de l'état haut de `clk_out`, correspondant à un cycle d'horloge de `clk_in`.

Division de Fréquence Basée sur `factor` La simulation de la Figure 10 montre que la fréquence de `clk_out` est divisée par le facteur `factor`. Avec `factor = 10,000`, deux "ticks" successifs de `clk_out` sont séparés par 10,000 cycles d'horloge de `clk_in`, correspondant à une période de 200 μs.

Ces simulations démontrent la capacité à générer un signal de sortie avec un état haut limité à un cycle de `clk_in` et une fréquence ajustable garantit une synchroni-

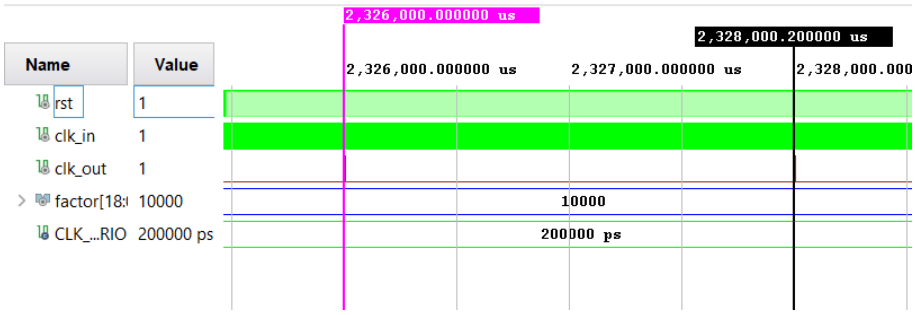

FIGURE 10 – Simulation montrant la division de fréquence basée sur le facteur `factor` = 10,000. Les deux "ticks" de `clk_out` sont espacés par 10,000 cycles de `clk_in`.

sation fiable et un contrôle précis de la vitesse dans les FSM et le moteur pas à pas. Le diviseur d'horloge constitue donc un composant clé du système.

2.3 Fichier de contraintes

Ce fichier est utilisé mapper correctement les signaux logiques aux ressources physiques du FPGA. Chaque signal utilisé est relié à une broche spécifique de la carte FPGA.
Sans ce fichier, le projet ne pourra pas être correctement synthétisé et implémenté sur la carte.

```
1  set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [
       get_ports { clk_in }];
2  create_clock -add -name sys_clk_pin -period 10.00 -waveform {0
       5} [get_ports {clk_in}];
3
4  set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [
       get_ports { rst }];
5  set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [
       get_ports { start }];
6  set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [
       get_ports { dir }];
7  set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [
       get_ports { Speed[0] }];
8  set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [
       get_ports { Speed[1] }];
9  set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [
       get_ports { Mode[0] }];
10 set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [
       get_ports { Mode[1] }];
11
12 set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [
       get_ports { O_0[0] }];
13 set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [
       get_ports { O_0[1] }];
14 set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 } [
       get_ports { O_0[2] }];
15 set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 } [
       get_ports { O_0[3] }];
```

3 Communication UART

La communication UART (Universal Asynchronous Receiver Transmitter) est une technologie de communication série largement utilisée dans les systèmes embarqués pour transmettre des données entre différents dispositifs. Ce rapport présente les principes de fonctionnement de la communication UART en mode 8N1, ainsi que les concepts d'oversampling et de baud rate, dans le cadre d'une implémentation en VHDL.

3.1 Concept Général de la Communication UART 8N1

Le protocole UART 8N1 est caractérisé par les éléments suivants :

- **8 bits de données :** Chaque trame contient 8 bits d'information utile.
- **Pas de parité :** Aucun bit de parité n'est utilisé pour la vérification des erreurs.
- **1 bit de stop :** Indique la fin de la trame.
- **1 bit de start :** Signale le début de la transmission.

Une trame UART typique commence par un bit de start (niveau bas), suivi des 8 bits de données, et se termine par un bit de stop (niveau haut). La communication est asynchrone, ce qui signifie que l'horloge n'est pas transmise avec les données ; le récepteur doit synchroniser son horloge interne avec le flux de données entrant.

3.2 Le Baud Rate et l'Oversampling

3.3 Baud Rate

Le baud rate représente la vitesse de transmission, exprimée en bits par seconde (bps). Par exemple, un baud rate de 9600 signifie que 9600 bits sont transmis par seconde. Ce paramètre doit être identique entre l'émetteur et le récepteur pour garantir une communication correcte.

3.3.1 Oversampling

L'oversampling est une technique utilisée pour améliorer la fiabilité de la détection des bits dans les systèmes UART. Au lieu de prélever un seul échantillon par bit, le récepteur en prélève plusieurs (généralement 16) et utilise ces échantillons pour déterminer avec précision la valeur du bit. Cela permet de compenser les dérives ou les bruits dans le signal.

3.4 Description des Modules VHDL

Cette section est divisée en deux parties distinctes pour analyser séparément les modules RX et TX du UART.

3.4.1 Générateur de Baud Rate

Le générateur de baud rate produit un signal d'horloge divisé (tick) correspondant au baud rate souhaité. Le diviseur est calculé à partir de la fréquence de l'horloge d'entrée et du baud rate configuré. Le fonctionnement est illustré dans les simulations suivantes :

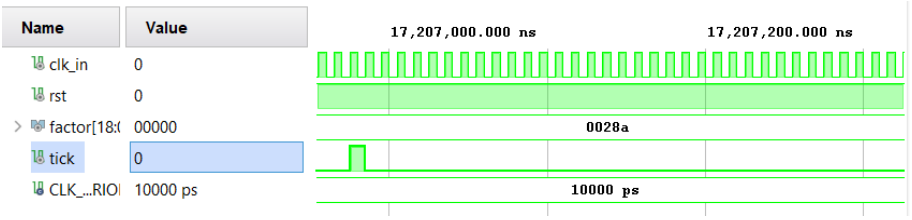

FIGURE 11 – Simulation du générateur de Baud Rate.

3.5 Analyse du Transmetteur UART (TX)

3.5.1 Description du Fonctionnement

Le transmetteur UART convertit les données parallèles en un flux série. Il passe par les états suivants :

- **IDLE :** Attente du signal `tx_start`.
- **START :** Transmission du bit de start.
- **DATA :** Transmission des 8 bits de données.
- **STOP :** Transmission du bit de stop.

Une fois tous les bits transmis, le signal `tx_done` est activé.

3.5.2 FSM du Transmetteur UART

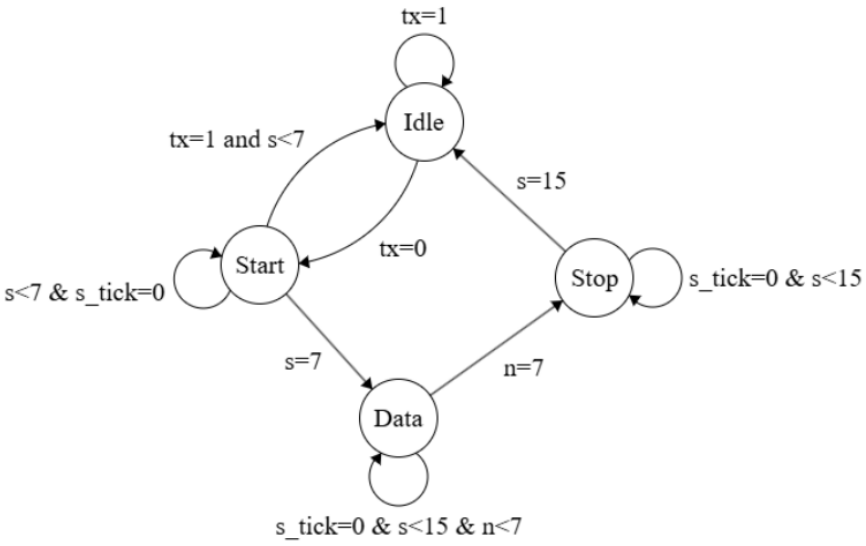

FIGURE 12 – FSM du Transmetteur UART (TX).

3.5.3 Simulations du Transmetteur

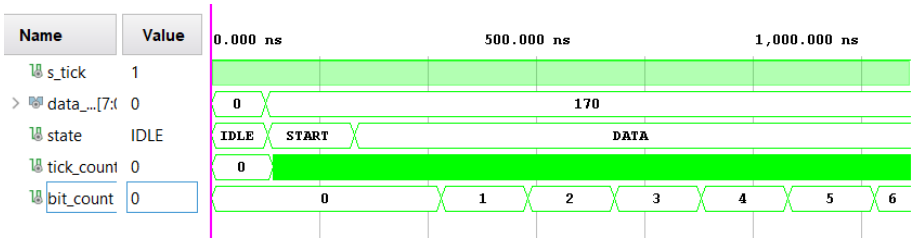

FIGURE 13 – Simulation des états du Transmetteur UART.

3.6 Analyse du Récepteur UART (RX)

3.7 Description du Fonctionnement

Le récepteur UART décode le flux série pour reconstruire les données parallèles. Deux compteurs principaux sont utilisés pour assurer la précision de la réception :

- **Compteur d'Échantillons (`s_counter`) :** Incrémenté pour déterminer le milieu de chaque bit.
- **Compteur de Bits (`bit_counter`) :** Incrémenté pour traiter chaque bit des données.

3.7.1 FSM du Récepteur UART

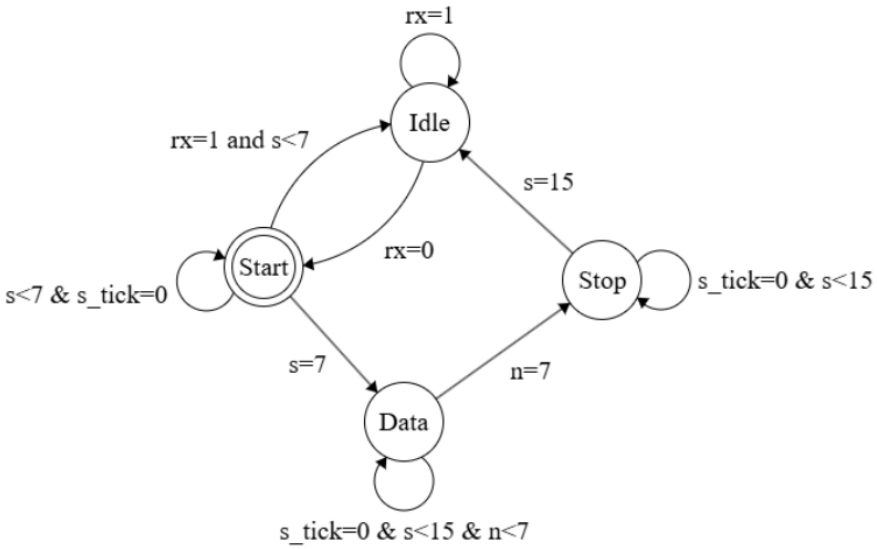

FIGURE 14 – FSM du Récepteur UART (RX).

3.7.2 Simulations du Récepteur

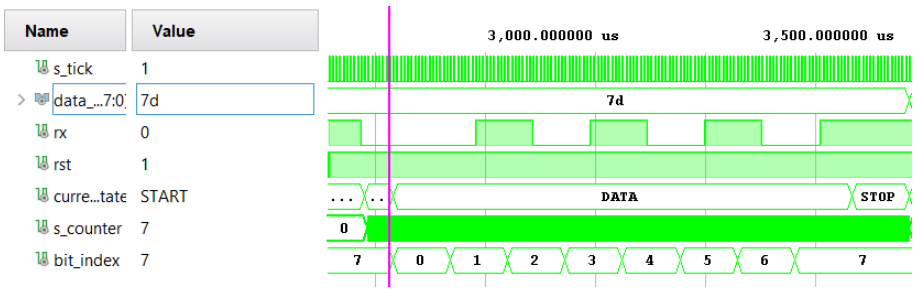

FIGURE 15 – Simulation des états du Récepteur UART.

3.8 Synthèse UART

La communication UART est un protocole essentiel pour les systèmes embarqués, offrant une solution simple et efficace pour la transmission de données. L'implémentation en VHDL des différents modules permet une personnalisation et une intégration adaptées aux besoins spécifiques. Ce projet a été conçu pour explorer ces concepts et fournir une base solide pour des applications futures.