Unité de transmission/réception série asynchrone version avec bit de parité

1 Introduction

Une UART, Universal Asynchronous Receiver/Transmitter, est un contrôleur d'entrées/sorties qui gère les ports série des ordinateurs. Il en existe différents modèles : les 8250 (8 bits), les 16450 (16 bits) et la famille des 16550 (16 bits + FIFO). Cette dernière version est utilisée dans tous les PCs actuels. L'UART, présente aussi sur les modem, permet la communication entre le modem et le PC. L'objectif est l'implantation d'une UART 8 bits simplifiée avec **bit de parité**.

2 Description de l'UART

2.1 Interface

L'UART que nous allons développer possède l'interface présentée en Fig. 1.

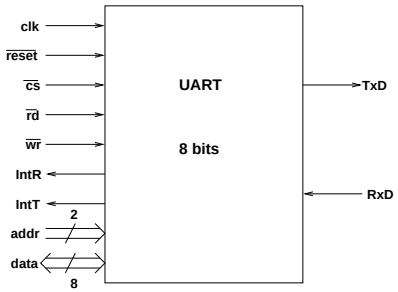

Figure 1 – Interface de l'UART

```
entity UARTunit is
  port ( clk, reset : in  std_logic;
         cs, rd, wr : in  std_logic;
         RxD        : in  std_logic;
         TxD        : out std_logic;
         IntR, IntT : out std_logic;
         addr       : in  std_logic_vector(1 downto 0);
         data_in    : in  std_logic_vector(7 downto 0);
         data_out   : out std_logic_vector(7 downto 0) );
end UARTunit;
```

2.2 Spécifications

L'UART est composée d'un registre de contrôle `ctrlReg` et de 4 unités fonctionnelles :
- `clkUnit`, cette unité décompose l'horloge de base du circuit en deux horloges nommées `enableTX` et `enableRX` qui contrôlent respectivement l'émetteur et le récepteur ;
- `ctrlUnit`, cette unité contrôle l'état de l'UART ;
- `TXUnit` est l'unité d'émission ;
- `RXUnit` est l'unité de réception.

La FIG. 2 montre la spécification interne du circuit UART.

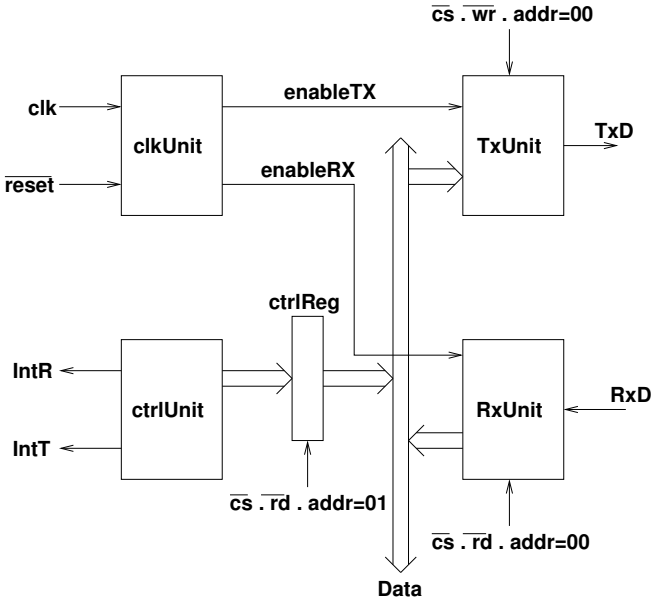

FIGURE 2 – Spécification interne de l'UART

Fonctionnement de l'UART :
- Lorsque le processeur reçoit l'interruption `IntT=0`, l'UART est prête à émettre. Le processeur positionnera `cs=0` et `wr=0` pour émettre un caractère donné à l'entrée `data_in` ;
- Lorsque le CPU reçoit l'interruption `IntR=0`, un caractère a totalement été reçu par l'UART. Pour le récupérer, le CPU positionne `cs=0` et `rd=0` ainsi que `addr=00` afin de lire le registre de réception de l'unité `RxUnit`. Le caractère reçu est placé sur `data_out`.
- À tout moment, le CPU peut connaître l'état de l'UART en lisant son registre de contrôle : `addr=01`, `cs=0` et `rd=0`. La valeur du registre de contrôle sera placée sur `data_out`.

3 L'unité d'horloge

3.1 Interface

```
entity clkUnit is
 port ( clk, reset : in  std_logic;
        enableTX   : out std_logic;
        enableRX   : out std_logic );
end clkUnit;
```

3.2 Spécifications

Cette unité a pour objectif de découper la fréquence de l'horloge de base `clk` en deux horloges :
– une pour l'émission, `enableTX`, de fréquence 9.6 kHz ;
– une pour la réception, `enableRX`, de fréquence 155 kHz.
L'horloge de réception, `enableRX`, est donc 16 fois plus rapide que l'horloge de réception `enableTX`. L'idée mise en œuvre pour obtenir ce résultat consiste à compter le nombre de fronts montants de `enableRX` et de générer un front montant de `enableTX` tous les 16 fronts montants de `enableTX`. De cette manière, il sera possible à l'unité de réception de contrôler l'asynchronisme de la communication. Ce comportement est donné par la FIG. 3.

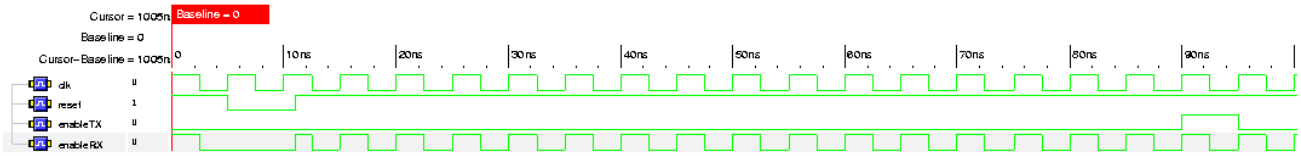

FIGURE 3 – Chronogramme de l'unité d'horloge

4 L'unité d'émission

4.1 Interface

```
entity TXUnit is
  port ( clk, reset : in std_logic;
         enable     : in std_logic;
         ld         : in std_logic;
         txd        : out std_logic;
         regE       : out std_logic;
         bufE       : out std_logic;
         data       : in std_logic_vector(7 downto 0) );
end TXUnit;
```

4.2 Spécifications

Cette unité transmet les données `data` bit à bit sur le support de communication via la sortie `txd`. Le format utilisé est de type RS232 avec 1 bit de stop et **avec** parité.

L'unité d'émission utilise 2 registres : un registre tampon et un registre d'émission (respectivement notés `BufferE` et `RegisterE`). L'état de ces registres est donné par les sorties `bufE` et `regE` :

– `bufE=1` : le tampon est vide ;
– `regE=1` : le registre d'émission est vide.

Initialement, `txd=1`, i.e. la ligne au repos est à l'état haut. L'émission est rythmée par le signal `enable` et se déroule en 6 étapes :

1. Attente d'une demande d'émission : lorsque le processeur a une donnée à transmettre, il positionne l'entrée `data` et active l'entrée `ld`. La donnée est alors chargée dans le tampon ;
2. On charge le contenu du tampon dans le registre d'émission ;
3. On lance l'émission par l'envoi du bit de start : `txd=0` ;
4. On émet les 8 bits de données contenues dans le registre d'émission du poids fort au poids faible ;
5. On émet le bit de parité (xor entre les 8 bits de données) ;
6. L'émission se termine par le bit de stop : `txd=1`.

A tout moment, si le tampon est vide, on peut demander à émettre des données. Ainsi, dès qu'une transmission est terminée, on peut recommencer à partir du point 2. A l'inverse, si le tampon est vide, on se remet en attente.

La FIG. 4 donne le contenu de l'unité d'émission ainsi que le lien entre les différents composants. Et la FIG. 5 montre un exemple d'émission d'un caractère.

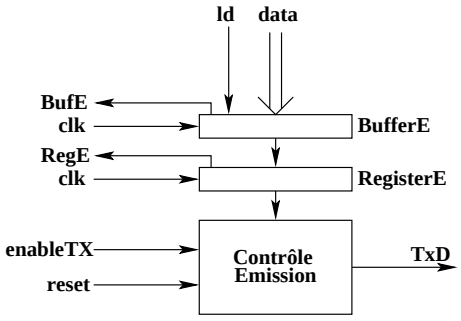

FIGURE 4 – Schéma de l'émetteur

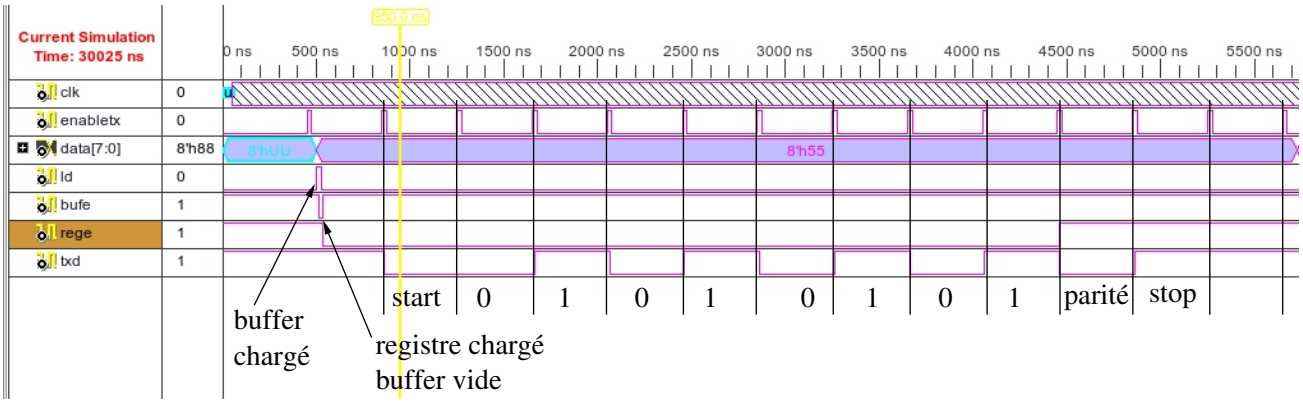

FIGURE 5 – Transmission du caractère 'U' (code ASCII : 85 (dec), Ox55 (hex))

5 L'unité de réception

5.1 Interface

```
entity RxUnit is
  port ( clk, reset        : in  std_logic;
         enable            : in  std_logic;
         rd                : in  std_logic;
         rxd               : in  std_logic;
         data              : out std_logic_vector(7 downto 0);
         FErr, OErr, DRdy  : out std_logic );
end RxUnit;
```

5.2 Spécifications

Cette unité réceptionne les données sur l'entrée `rxd` et les transmet au processeur via la sortie `data` lorsque la réception est terminée et que le processeur positionne le signal `rd`. Cette unité fournit aussi 3 signaux :

- `Drdy=1` lorsqu'une donnée est reçue. Ce signal repasse à 0 lorsque la lecture est effectuée par le processeur ;
- `FErr=1` si la trame reçue est erronée. Le bit de parité ou(et) le bit de stop est(sont) erroné(s) ;
- `OErr=1` si lorsqu'une donnée est prête, elle n'est pas lue par le processeur.

Le fonctionnement de l'unité de réception est rythmé par le signal `enable`. Il est constitué de 2 parties :

- un élément **Compteur16** qui gère les instants de réception ;
- un élément **Contrôle Réception** qui gère les états de l'unité de réception.

Compteur16 Le premier élément compte le nombre de signaux `enable` reçus et va transmettre les données récupérées sur `rxd` à la bonne cadence au second élément.

Cette cadence est donnée par `tmpClk` de la manière suivante : aucun front montant quand la ligne est au repos, un front montant après 8 fronts montants de `enable` dès qu'une réception commence puis un front montant tous les 16 fronts montants de `enable` et ceci jusqu'à réceptionner toute la trame pour se remettre ensuite en attente d'une nouvelle réception.

De cette manière, on gère l'asynchronisme entre l'émetteur et le récepteur.

Contrôle Réception Le fonctionnement du deuxième élément se découpe en 4 étapes :
– on attend tout d'abord la réception du bit de start ;
– on effectue la réception de 8 bits de données, du poids fort au poids faible ;
– on réceptionne le bit de parité ;
– on attend la réception du bit de stop.
Au cours de la dernière étape, plusieurs cas peuvent se présenter :

1. Si le bit de stop est erroné, i.e. égal à zéro, ou le bit de parité est incorrect, on avertit le processeur en positionnant `FErr=1` ;
2. Sinon, on positionne `DRdy=1` et la donnée en sortie. Au front montant suivant de `enable`, on rabaisse `DRdy` et
   (a) Si `rd` reste à zéro, il y a une erreur de transmission : `OErr=1`,
   (b) Si `rd` passe à un, tout s'est bien passé.

La FIG. 6 représente les composants de la partie récepteur.

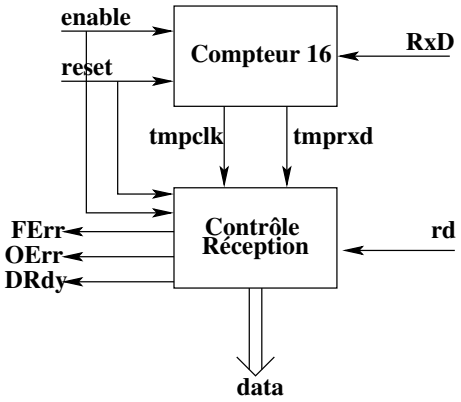

FIGURE 6 – Schéma du récepteur

La FIG. 7 montre la réception normale du caractère 'U' (Ox55), la FIG. 8 montre une erreur de transmission, la FIG. 9 montre une erreur de parité, et l'émission du signal `FErr` et la FIG. 10 un non-positionnement du signal de lecture et l'émission du signal `OErr`.

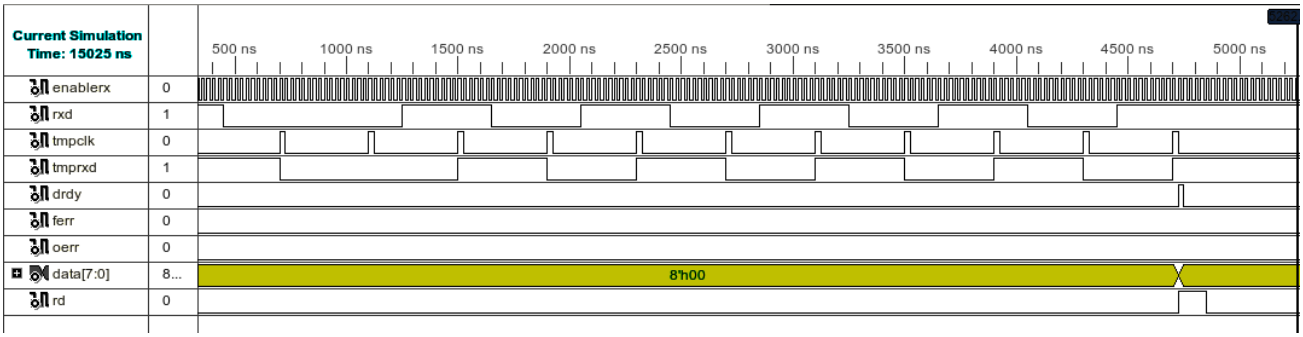

FIGURE 7 – Exemple de la réception normale du caractère 'U' (0x55)

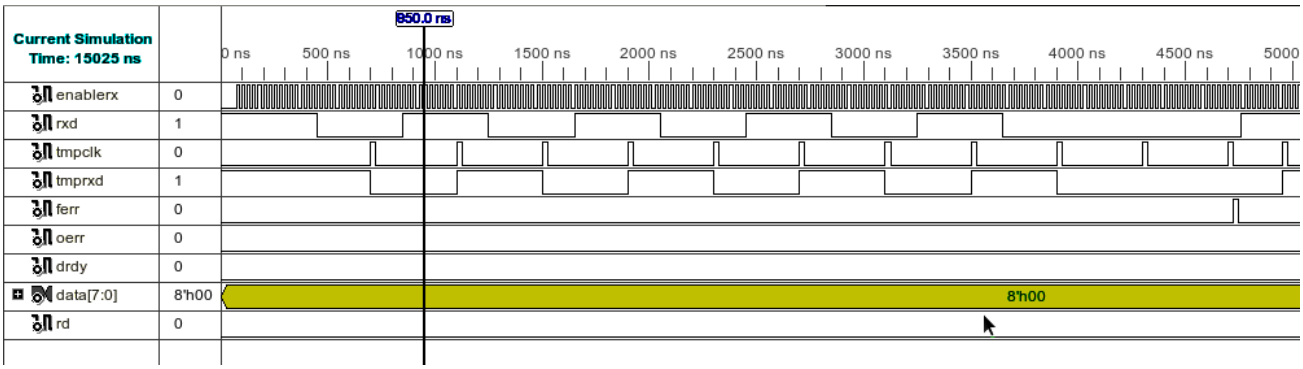

FIGURE 8 – Exemple de réception erronée : bit de stop faux

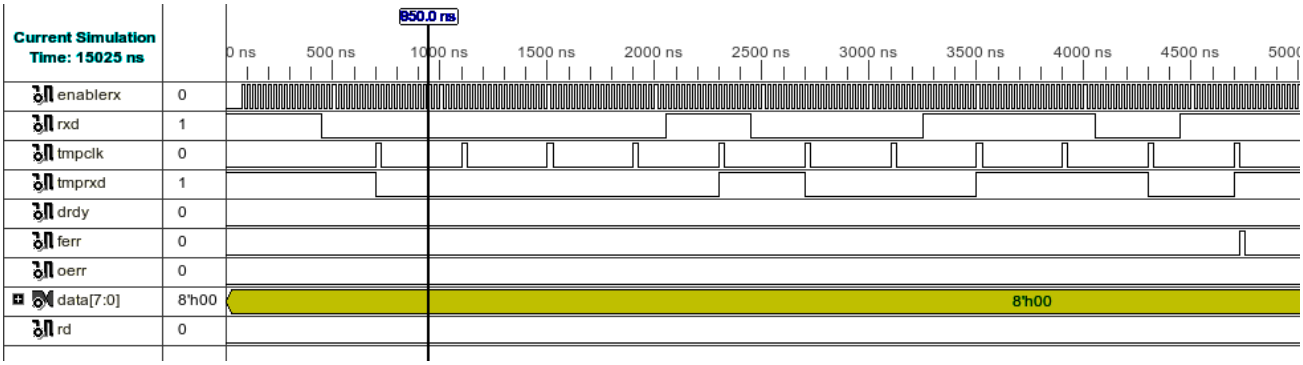

FIGURE 9 – Exemple de réception erronée : bit de parité faux

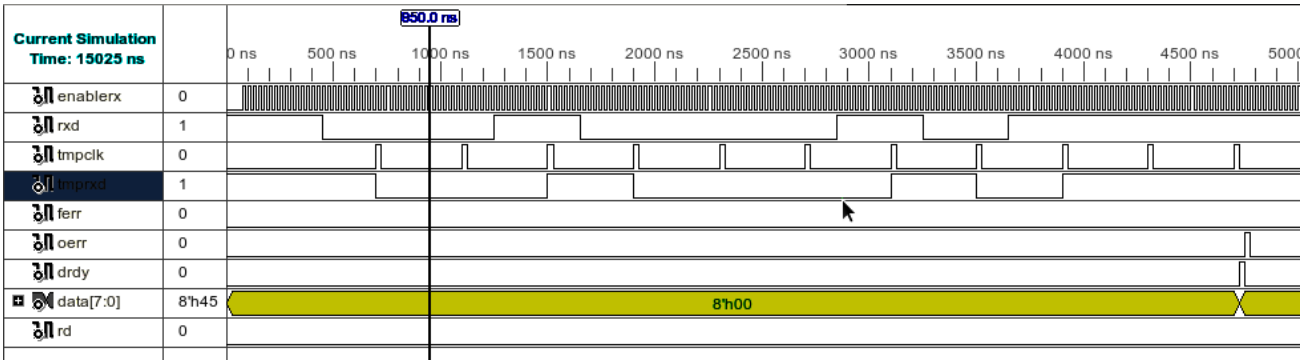

Figure 10 – Exemple de réception erronée : pas de positionnement de rd

6 L'unité de contrôle

6.1 Interface

```
entity ctrlUnit is
  port ( clk, reset        : in  std_logic;
         rd, wr            : in  std_logic;
         DRdy, FErr, OErr  : in  std_logic;
         BufE, RegE        : in  std_logic;
         IntR              : out std_logic;
         IntT              : out std_logic;
         ctrlReg           : out std_logic_vector(7 downto 0) );
end ctrlUnit;
```

6.2 Spécifications

L'unité de contrôle fournit les signaux d'interruption `IntR` et `IntT` indiquant respectivement si une donnée a été reçue ou non et s'il est possible de transmettre une donnée. Elle gère aussi les données contenues dans le registre ctrlReg donné FIG. 11.

Si une donnée est disponible, `Drdy=1`, `IntR=0`. `IntR` sera remis à 1 lorsque la lecture de la donnée est effectuée.

Lorsque le buffer ou le registre d'émission est libre (`BufE=1` ou `RegE=1`), il est possible de transmettre une donnée : `IntT=0`. Sinon, l'écriture d'une donnée dans le buffer ou le registre d'émission désactive l'interruption : `IntT=1`.

Gestion du registre de contrôle ctrlReg :
- le champs TxRdy indique que l'émetteur est prêt à émettre une donnée ;
- le champs RxRdy indique que le récepteur a reçu une donnée ;
- les champs FErr et OErr correspondent aux erreurs de transmission obtenues par l'unité de réception ;
- les champs réservés, notés `Res`, sont toujours à 1.

Res	Res	Res	Res	TxRdy	RxRdy	FErr	OErr

FIGURE 11 – Le registre de contrôle regCtrl.

7 Travail à faire

1. Coder l'unité d'horloge,
2. Etudier le comportement des unités de réception et d'émission sous la forme d'automates,
3. Construire et valider les unités de réception et d'émission,
4. Assemblage du tout : l'UART.