Récepteur série asynchrone

Il s'agit de concevoir la partie réception d'un UART (Universal Asynchronous Receiver and Transmitter) qui permet de disposer sous forme parallèle d'une information de 8 bits transmise sous forme série asynchrone. Le composant à concevoir doit servir comme interface entre un microprocesseur et une liaison série asynchrone du type RS232 au bout de laquelle peut se trouver l'émetteur UART dont on vient d'effectuer la conception. La délimitation du composant est donnée ci-dessous :

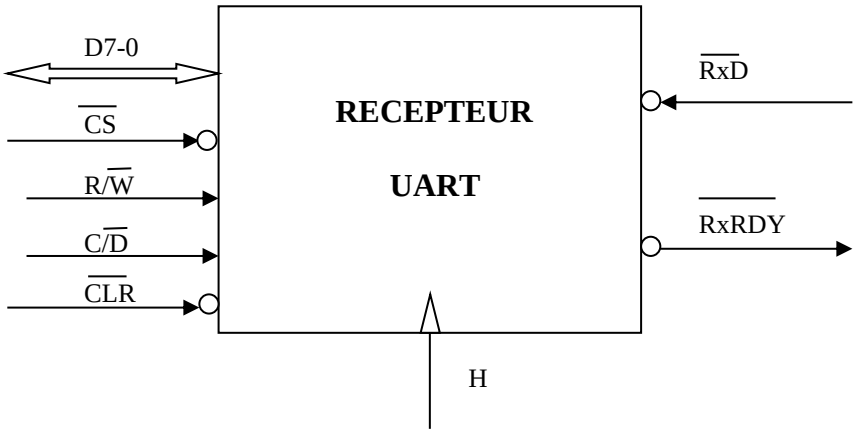


Figure 1: symbole du récepteur UART

Coté microprocesseur, le composant est programmable pour la vitesse de réception. Un registre interne de 2 bits mémorise la valeur lorsque $\overline{CS}$ = 0, $R/\overline{W}$ =0, $C/\overline{D}$ =1. La fréquence de réception est alors divisée par 4, 8 ou 16 par rapport à H. Le codage de la vitesse est au choix du concepteur. L'état du récepteur est observable lorsque $\overline{CS}$ = 0, $R/\overline{W}$ = 1, $C/\overline{D}$ = 1. Trois bits sont disponibles :

- RxRDY : Récepteur prêt à recevoir (même signification que la sortie $\overline{RxRDY}$)
- OVERRUN : écrasement d'une donnée
- PARITE : défaut de parité détecté

La donnée est lue lorsque $\overline{CS} = 0$, $R/\overline{W} = 1$ et $C/\overline{D} = 0$. L'état des indicateurs est alors mis à jour : RxRDY=vrai, OVERRUN=PARITE=faux.

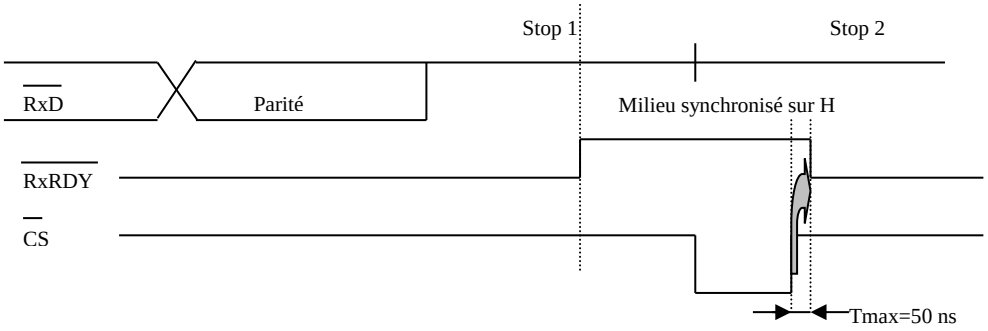
Coté liaison, le signal série $\overline{RxD}$ comprend : 1 bit start, 8 bits de donnée (bit de poids faible d'abord), 1 bit de parité parité, 1 ou 2 bits de stop. La sortie $\overline{RxRDY}$ indique la disponibilité du récepteur à recevoir une donnée. C'est cette sortie qui peut être utilisée pour assurer un asservissement du producteur en agissant sur l'entrée $\overline{CTS}$ de l'émetteur. Bien entendu, le circuit peut recevoir une donnée même lorsqu'il indique qu'il n'est pas prêt. Si la réception est achevée avant la lecture de la donnée précédente, il y a OVERRUN.

Les contraintes technologiques suivantes sont à respecter :

- Le circuit est alimenté sous 5 V
- Comme l'indique la notation, les signaux $\overline{CS}$, $\overline{RxD}$, $\overline{RxRDY}$ sont actifs au niveau bas.
- Les entrées et sorties sont compatibles TTL
- Les entrées ne doivent pas consommer plus de 0.2 mA
- Les sorties doivent pouvoir produire 2 mA au niveau bas,
- Les lignes D7-0 sont bidirectionnelles avec état tristate au repos.

On tiendra également compte des contraintes temporelles suivantes :

- L'échantillonnage de $\overline{RxD}$ doit se faire au milieu de chaque bit (au mieux)
- L'évolution de $\overline{RxRDY}$ et les temps pour les cycles de lecture et d'écriture sont donnés sur les chronogrammes de la figure ci-dessous.



(a) Contrainte pour RxRDY

Lecture *Ecriture*

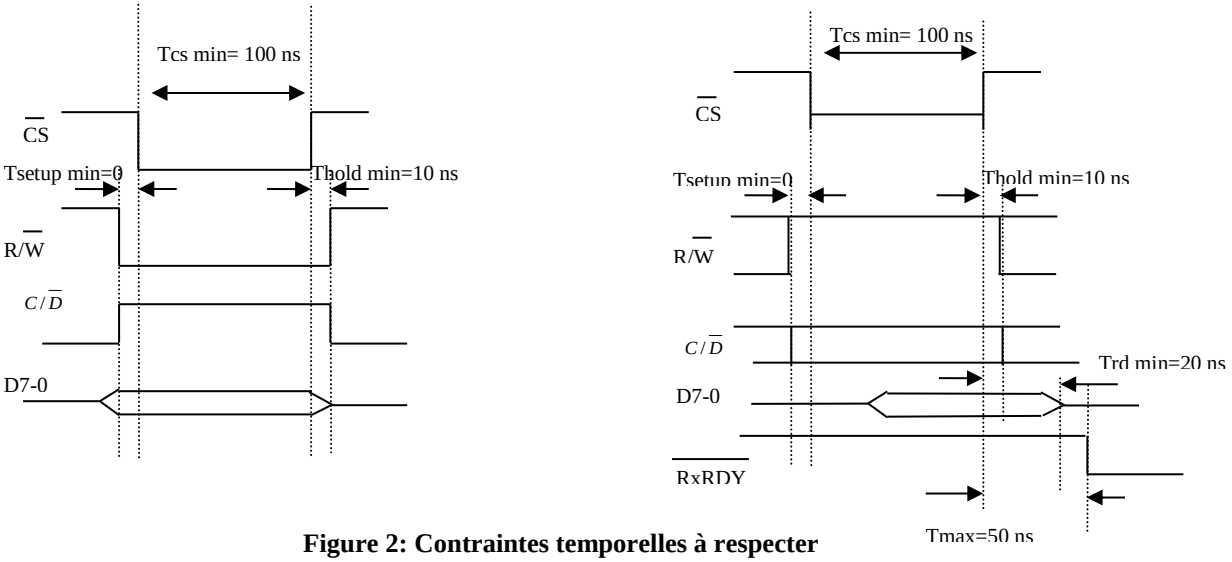


Figure 2: Contraintes temporelles à respecter

A. Analyse et modélisation de l'environnement

La description précise des entités utile pour le circuit requiert un travail d'analyse permettant de spécifier les entrées-sorties de chacune des entités. Il faut déterminer le niveau de détail suffisant dans leur description en adoptant un point de vue fonctionnel plutôt que physique. Dans le cas du récepteur, 2 entités sont à considérer :

- L'émetteur de la donnée série
- Le consommateur

L'émetteur de la donnée transmet celle-ci lorsqu'il le souhaite. En toute logique, il ne doit le faire que lorsque RxRDY est vrai, mais ce n'est pas obligatoire. Un OVERRUN peut alors apparaître dans le circuit de réception.

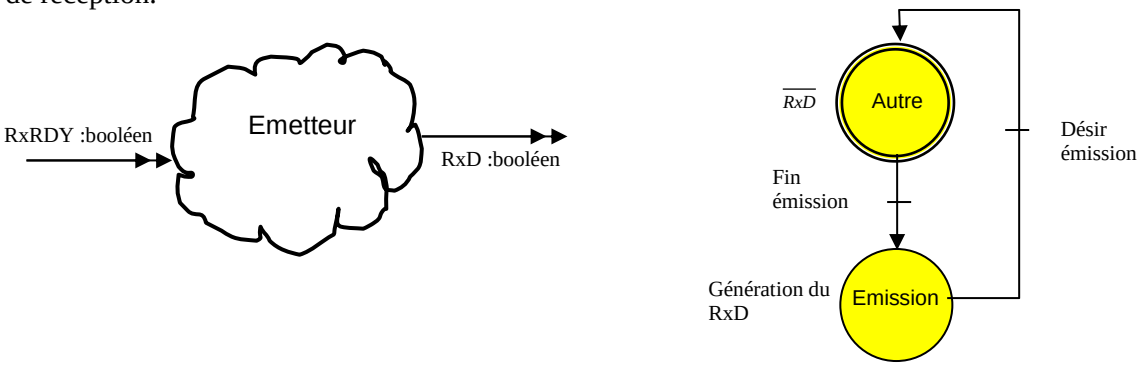


Figure 3: Comportement de l'émetteur

Le comportement de l'émetteur utile pour la conception du circuit récepteur apparaît en figure 3. On retrouve une partie des spécifications du transmetteur UART étudié précédemment mais dont on a représenté que la partie concernant la transmission utile ici. On peut noter que *Desir emission* et *fin_emission* sont des événements internes à l'émetteur (correspondent aux modifications d'état de TxRDY). RxD est une sortie du type donnée booléen (double flèche), les valeurs successives résultant de l'échantillonnage des bits en série. Le format est décrit dans le cahier des charges : 1 bit start, 8 bits de données, & bit de parité paire, 1 ou 2 bits stop. On notera que la sortie fonctionnelle est RxD alors que la sortie niveau physique est $\overline{RxD}$ (actif au niveau bas).

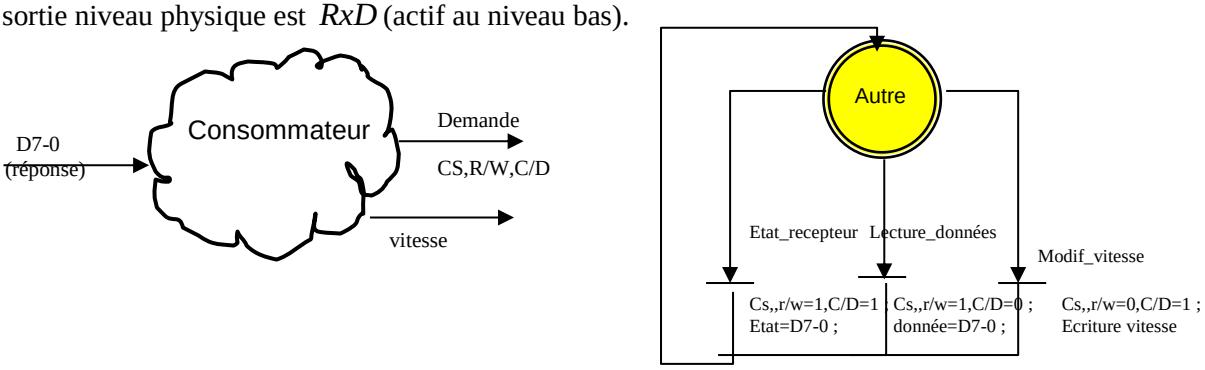


Figure 4: Comportement du consommateur

Le consommateur (un programme sur microprocesseur) peut souhaiter à tout moment :

- Modifier la vitesse de réception
- Connaître l'état du circuit
- Lire une donnée reçue

Son comportement peut être formalisé comme sur la figure 4. *Etat_recepteur, lecture_donnée,modif_vitesse* sont des événements internes. Le type de demande est matérialisé par CS (chip select), R/W (read/write), C/D (adresse du registre control/data)

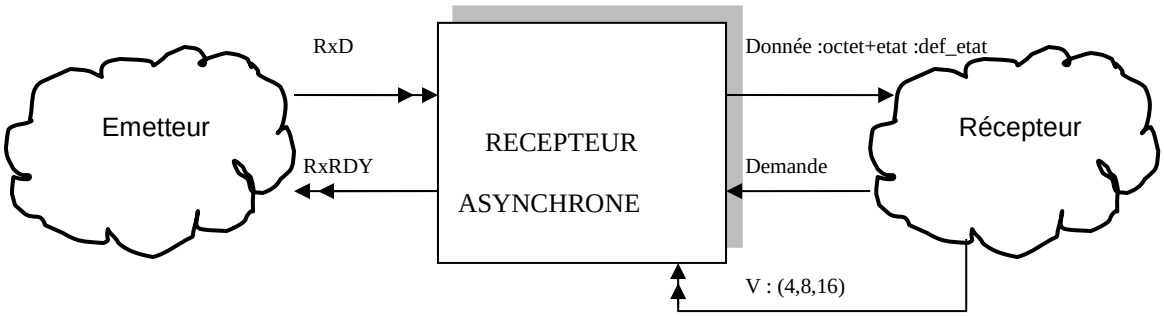


Figure 5: Délimitation des entrées/sorties du circuit.

Une délimitation du circuit vous est fournie ci-dessus. Pour simplifier, les différentes demandes issues du consommateur ont été regroupées sous le lien événement *demande*. La réponse est une information (simple flèche) comprenant la valeur de la dernière donnée reçue et l'état du récepteur. La vitesse est, quant à elle, considérée comme un paramètre pour le récepteur. Pour cela, V est considérée comme une grandeur (double flèche) permanente dont la valeur est mise à jour du coté consommation.

Question : On vous propose de compléter les spécifications fonctionnelles ci-dessous en déterminant les conditions, actions nécessaires au fonctionnement du récepteur série. 3 automates doivent être définis :

- Un pour décrire l'évolution de la sortie *RxRdy (*typiquement connecté sur *cts* du transmetteur). On spécifie les 2 valeurs possibles de la sortie
- Un automate pour décrire le comportement sur réception de l'événement demande et générer les sorties Données+etat
- Un automate pour décrire le comportement sur l'entrée RxD

Pour ce dernier automate, la spécification est proche de celle effectuée pour l'évolution de Txd dans le cas de l'émetteur, mais on procède par les entrées. Ne pas oublier que l'échantillonnage du bit doit s'effectuer le plus proche possible du centre du bit reçu.