الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'enseignement Supérieur et de la Recherche scientifique

Université Mohamed Khider Biskra
Faculté des Sciences et de la Technologie
Département de Génie Electrique
Filière : Electronique
Option : Signaux et Communication

Réf:……

Mémoire de Fin d'Etudes
En vue de l'obtention du diplôme:

MASTER

Thème

ANALYSE ET SYNTHÈSE D'UN UART EN VHDL

Présenté par :
ZERARI HOUSSAM
Soutenu le : 2 Juin 2013

Devant le jury composé de :

Mr.HENDAOUI Moinira	**MAA**	**Président**
Mr.KAHOUL Nadhir	**MAA**	**Encadreur**
Mr.DIABI Fathi	**MAB**	**Examinateur**

Année universitaire: 2012 / 2013

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'enseignement Supérieur et de la recherche scientifique

Université Mohamed Khider Biskra
Faculté des Sciences et de la Technologie
Département de Génie Electrique
Filière : Electronique
Option : Signaux et Communication

Mémoire de Fin d'Etudes
En vue de l'obtention du diplôme:

MASTER

Thème

ANALYSE ET SYNTHÈSE D'UN UART EN VHDL

Présenté par :

ZERARI HOUSSAM

Avis favorable de l'encadreur :

Nom Prénom *signature*

Avis favorable du Président du Jury

Nom Prénom *Signature*

Cachet et signature

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'enseignement Supérieur et de la Recherche scientifique

Université Mohamed Khider Biskra
Faculté des Sciences et de la Technologie
Département de Génie Electrique
Filière : Electronique
Option : Signaux et Communication

Thème :

ANALYSE ET SYNTHÈSE D'UN UART EN VHDL

Proposé par : M.KAHOUL NADHIR

Dirigé par : M.KAHOUL NADHIR

RESUME

Le but d'un langage de description matériel tel que le VHDL est de faciliter le développement d'un circuit numérique en fournissant une méthode rigoureuse de description du fonctionnement et de l'architecture du circuit désirée, pour des circuit numériques plus simple ou plus complexe. C'est pour ca qu'on propose un circuit très intéressant dans la transmission série asynchrone. Après ce que nous définissons la transmission série et ses différentes classe, et la norme RS-232, et présenté la structure interne de l'UART, et après avoir rappelé les notions fondamentales et nécessaire au langage VHDL. Enfin on a décri en VHDL les parties essentielles de l'UART : émetteur, récepteur, générateur de vitesse.

ملخص

إن الهدف من لغة الوصف مثل VHDLهو تسهيل تطوير الدوائر الرقمية بتقديم وصف دقيق للوظيفة و البنية الكاملة للدارة المطلوبة . و ذلك من اجل الدوائر الرقمية البسيطة او المعقدة. من اجل هذا قدمنا دارة مثيرة للإهتمام في مجال النقل المتسلسل الغير متزامن. بعد تعريف النقل التسلسلي و مختلف انواعه, بالإضافة إلى المرجع RS-232 , و بعرض البنية الداخلية للمرسل المستقبل العالمي الغير متزامن UART , وبعد التعرف على المفاهيم الرئيسية و الضرورية في لغة VHDL. في النهاية نقوم بواسطة لغة VHDL بكتابة العناصر الثلاثة الرئيسية في UART و هي المرسل و المستقبل و مولد السرعة.

Mots clés : synchrone, asynchrone, RS-232, UART, HandShake

Dédicaces

De tout mon cœur je dédie ce modeste travail

A ma chère mère, la lumière qui nous a guidés vers le

Chemin de savoir.

A mon cher père, pour leur sacrifice.

A ma chère grand-mère « OMMA »

et « Marzaka »

A mes chères sœurs (djihad, achouak)

A ma belle famille

A mes chers amis chaque un et son nom

Surtout « Peluch, Aissa, Yassin, Ahmed, Mahmoud, Meyoum, Imen, OuSa»

A tous ma promotion BAC 2008 ☺

Houssam Zerari

REMERCIEMENTS

Louange à dieu le miséricordieux qui nous permis de bien accomplir ce modeste travail. Mes remerciements s'adressent tout particulièrement à mon encadreur Monsieur Kahoul Nadhir pour l'effort fourni, et son soutien, ses conseils scientifiques, sa patience et sa persévérance dans le suivi. Mes remerciements aussi à monsieur Rahmani Nassr-Eddine pour sa disponibilité, ses conseils constructifs.

Je remercie très sincèrement, les membres de jury qui m'ont fait le très grand honneur de participer à l'amélioration de mes travaux.

J'adresse également mes remerciements, à tous mes enseignants, qui m'ont donnée les bases de la science pendant les cinq ans de ma formation. Mes remerciements aussi à tous responsables dans le département de génie électrique : Mme Akila, Mlle Dalal.

Je tiens également à remercier tous mes professeurs dans lycée Ridha El-Achouri Biskra, et tous les employés qui y travaillent.

A toute personne qui a participé de près ou de loin pour l'accomplissement de ce modeste travail. Merci donc ma mère, mon père, à mes sœurs, à ma famille pour leur présence, leur confiance et leur soutien.

TABLE DES MATIERES

Liste des figures

Liste des tableaux

INTRODUCTION GÉNÉRALE

Cette dernière décennie a été témoin d'une évolution technologique incontournable dans tous les domaines, surtout dans le monde de la transmission numérique, et les systèmes de transmission numérique qui véhiculent de l'information entre une source et un destinataire en utilisant un support physique comme le câble. Pour cela on peut envisager deux modes célèbres: envoyer toutes les données en même temps sur autant de lignes de transmission, c'est le mode **parallèle**, ou bien les envoyer l'une après l'autre sur une seule ligne de transmission. C'est le mode **série**.

Les ports d'entrée-sortie sont des éléments matériels de l'ordinateur, permettant au système de communiquer avec des éléments extérieurs, c'est-à-dire d'échanger des données, d'où l'appellation d'interface d'entrée-sortie (notée parfois interface d'E/S). Ces ports communiquent avec l'extérieur en utilisant soit le mode parallèle, soit le mode série.

Ce dernier mode est le plus utilisé pour les liaisons séries à grandes distances dans le monde de communication. Parmi ses avantages, on a la réduction du nombre de fils utilisé. Généralement, on utilise un seul fil pour l'émission et un autre fil pour la réception. Cependant on peut trouver plusieurs supports de transmission, par exemple : la ligne bifilaire (signal + masse), la fibre optique, le canal infrarouge.ect.

Une difficulté majeure de ce mode de transmission est liée à l'horloge ; en effet, il est nécessaire d'employer une horloge d'émission et une horloge de réception qui doit fonctionner en synchronisme parfait. Ainsi on a deux grands types ou classes de liaison série : **asynchrone** ou bien **synchrone**.

La transmission série asynchrone : un signal de synchronisation est généré par l'émetteur (bit de Start) au début de la séquence de bits donnés (un octet par exemple), elle est utilisé plusieurs protocole (norme) par exemple RS-232.

La transmission série synchrone : l'émetteur génère un signal, au début de la trame, qui doit permettre au récepteur de se synchroniser à chaque envoi.

La liaison RS232 est une liaison série asynchrone. Ce protocole consiste à envoyer une succession de « trames » de données. Chaque trame est constituée par un

bit de Start, un caractère de plusieurs bits de données, un bit de parité, et un ou deux bits de Stop. Le cerveau d'une carte RS232 est constitué par un processeur appelé **UART** (Universal Asynchronous Receiver Transmitter) émetteur-récepteur asynchrone universel. Il permet d'établir l'interface entre le processeur et l'extérieur.

Après avoir établi l'architecture des différents organes de l'UART nous allons le décrire en VHDL dans la quatrième partie de ce mémoire. Avant cela, on va présenter les règles et les instructions essentielles du langage VHDL qui seront utilisées.

Pendant longtemps, les langages de programmation tels que FORTRAN, Pascal et C^+ ont été utilisés pour décrire des programmes informatiques. Toutefois, dans le domaine de la conception numérique, les concepteurs ont ressenti le besoin d'un langage standard pour décrire les circuits numériques ou faire la modélisation numérique. Par la suite, les Langages de Description Hardware (HDL) sont entrés dans l'existence. Aujourd'hui deux HDLs populaires sont utilisés par les ingénieurs, qui sont le **VHDL** et le Verilog. Le VHDL, qui signifie « Very high speed integrated circuit Hardware Description Language », a été développé par le département américain de la Défense et de l'IEEE dans le milieu des années 1980.

L'objet de ce travail consiste à décrire en VHDL le circuit qui gère la liaison RS-232. C'est un UART avec ses parties essentielles : émetteur, récepteur, générateur de vitesse. Pour vérifier la description VHDL et procéder à la simulation du circuit UART, on a utilisé le logiciel de la firme ALTREA Quartus II design suit 7.2.

Enfin en remarque que le circuit peut être implémenté dans un circuit logique programmable FPGA, en utilisons la carte ALTERA DE2.

Chapitre I

Généralités sur la liaison série et la norme RS232

I.1 INTRODUCTION

Une liaison série est une ligne où les bits d'information (1 ou 0) arrivent successivement, soit à intervalles réguliers (transmission synchrone), soit à des intervalles aléatoires, en groupe (transmission asynchrone).

La liaison RS232 est une liaison série asynchrone.

L'octet à transmettre est envoyé bit par bit (poids faible en premier) par l'émetteur sur la ligne Tx, vers le récepteur (ligne Rx) qui le reconstitue.

La communication peut se faire dans les deux sens (duplex), soit émission d'abord, puis réception ensuite (half-duplex), soit émission et réception simultanées (full-duplex).

La transmission étant du type asynchrone (pas d'horloge commune entre l'émetteur et le récepteur), des bits supplémentaires sont indispensables au fonctionnement: bit de début de mot (start), bit(s) de fin de mot (stop).

D'autre part, l'utilisation éventuelle d'un bit de parité, permet la détection d'erreurs dans la transmission.

Dans ce chapitre nous avons donné une bref présentation sur la liaison série avec les deux types synchrone et asynchrone et la norme RS-232 avec descriptions techniques de la norme.

I.2 PORT

Un port est le terme permettant de spécifier une connexion normalisée. Ce terme désigne implicitement le type de liaison ou de bus et la connectique qui y est rattachée. En micro-informatique, on parle communément du port RS232 ou du port parallèle. [1]

I.3 PARALLELE OU SÉRIE

I.3.1 Liaison parallèle

La méthode la plus répandue, consiste à relier les deux équipements par un bus parallèle. Ce type de connexion a l'avantage de satisfaire des débits très importants avec un nombre de bits variant suivant le matériel utilisé (4, 8, 16, 32, etc.). Elle reste limiter à de courtes distances, un à deux mètres dans le meilleur des cas. De plus, on peut être confronté à des problèmes de diaphonie dans les câbles ainsi que sur les cartes. L'encombrement peut être également un problème en terme de surface de pistes sur les cartes, d'autant plus que ce type

de liaison impose des connecteurs de grandes tailles. Pour illustrer, on peut citer les liaisons ou bus parallèle les plus connus : le port Centronics, le bus SCSI, le bus IEEE488, etc. [1]

Le schéma simplifie d'une liaison parallèle est donné par la figure I.1. [1]

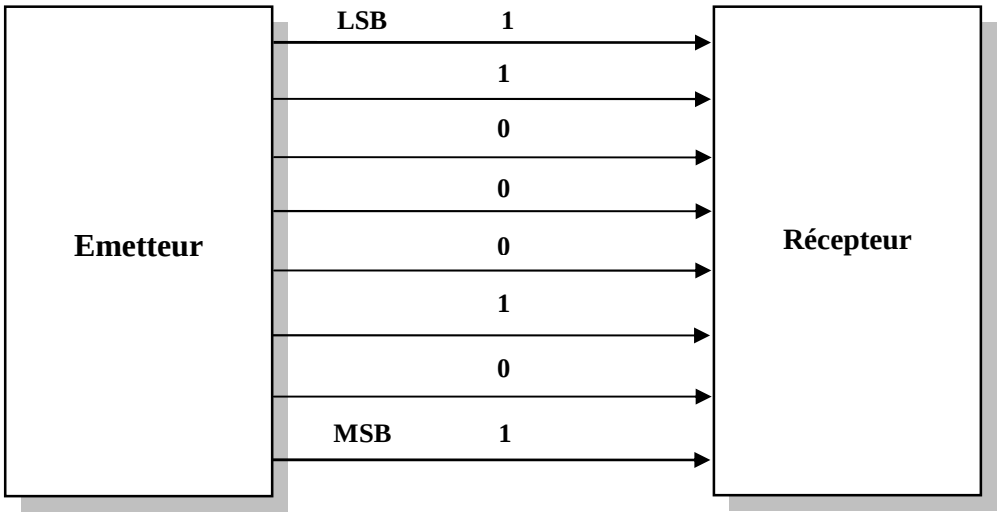


Fig.I.1. Exemple d'une liaison parallèle

Une autre méthode consiste à transmettre les données les unes après les autres sur un seul fil. Cette méthode porte le nom de liaison série.

I.3.2 Liaison série

Pour interpréter un message binaire transmis en série, le récepteur et l'émetteur doivent pouvoir identifier le début et la fin d'un caractère ou bien d'un message grâce à une procédure appelée « protocole de communication série ». Il en existe plusieurs qu'on peut regroupe en deux grandes classes : **synchrones** et **asynchrones**

I.3.2.1 Protocole synchrone

La caractéristique essentielle de transfert synchrone des données en série est l'asservissement exact des données au signal d'horloge.

Une fois que la vitesse de transmission (en bauds) a été fixée, le dispositif émetteur doit transmettre un bit à chaque impulsion d'horloge.

En plus de la vitesse de transmission, un protocole synchrone doit encore :

- Fixer la langueur d'un mot (ou caractère) de données.
- Permettre ou dispositif récepteur de se synchroniser. Un message doit commencer par un ou deux caractères spéciaux, dits « caractère de synchronisation » (mot SYNC)

Ainsi en transmission synchrone, l'émission une fois commencée est continue, l'émetteur comble les trous éventuels d'un message par l'envoi d'un ou plusieurs caractères SYNC pour permettre au récepteur de rester synchronisé.sur un seul support « fil », on a succession de 0 et 1 qui constituent la trame comme le montre la figure I.2.

Le temps qui sépare l'envoi de deux messages doit être un multiple de la durée du bit.

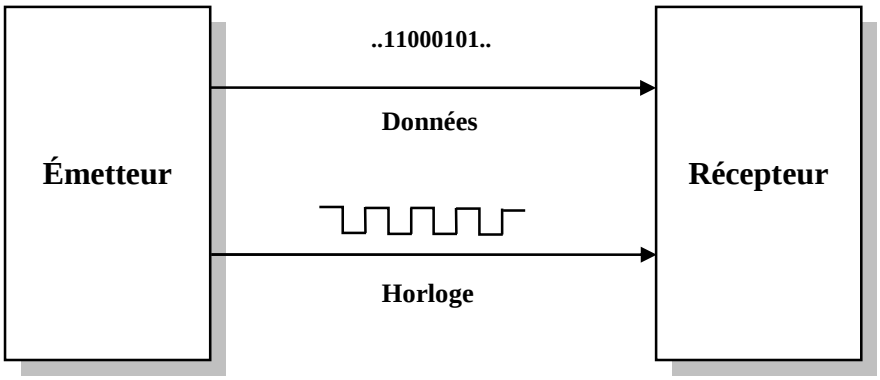


Fig.I.2 Exemple d'une liaison série synchrone

I.3.2.2 Protocoles asynchrones

Le principe de transmission est identique à la liaison série synchrone mais sans la ligne d'horloge.les deux équipements possèdent leur propre horloge indépendante l'une de l'autre. L'horloge du récepteur se synchronise uniquement sur le premier bit de la trame, puis elle continue sur la lancée.sur un seul fil, on transmet le caractère qui constitue l'information. Celle-ci est précédée du signal START et terminée par le « ou les » bit de STOP. Le schéma de la figure I.3 représente une liaison série asynchrone.

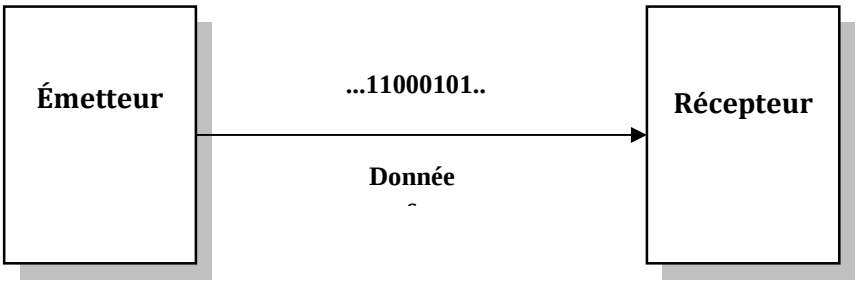


Fig.I.3. Exemple d'une liaison série asynchrone

La grande différence entre les protocoles synchrones et asynchrones est que, dans ce deuxième cas, le flot des données est DISCONTINU ; l'émission se fait caractère par caractère, au fur et à mesure que ceux-ci sont disponibles. Entre deux caractères, l'émetteur envoie un

signal de rupture consistant en un niveau de tension haut. Ce niveau haut (appelé MARK signal en technique des transmissions), pour permettre de commencer l'envoi d'un caractère par un bit de départ à l'état bas (ou zéro).

Chaque mot « donnée » en transmission asynchrone, étant séparé, a besoin, pour être identifié par le récepteur, de contenir sa propre information de synchronisation. Ceci se fait en cadrant les bits d'information utile entre un bit dit de départ (START bit) et un ou deux bits d'arrêt (STOP bit).

Dans le domaine des micro-ordinateurs, le format d'une unité de données transmise est montré à la figure I.4

- Un bit START unique, de valeur 0 (norme universelle) ;
- Un mot d'information comportant 5,6 ou le plus souvent 7 bits (code ASCII) ;
- Un bit (facultatif) de parité, paire ou impaire ;
- Un ou deux bits STOP (exceptionnellement 1,5).

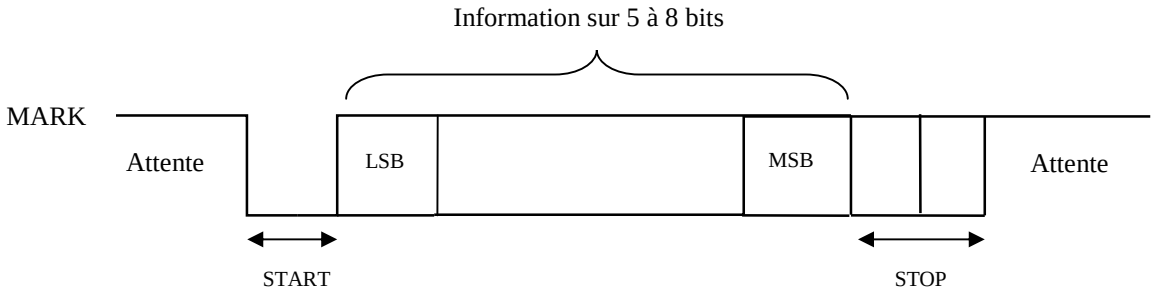


Fig.I.4. Transmission série asynchrone

I.3.3 La parité

Pour s'assurer que le caractère envoyé est bien celui qui a été reçu, on peut utiliser le contrôle de parité. Le principe est de rajouter un bit de parité directement après le caractère. Ce bit est mis à 0 ou à 1 selon que l'on veuille obtenir une parité paire (le nombre de 1 est pair) ou impaire (le nombre de 1 est impair). Si on souhaite transmettre un caractère ayant un nombre impair de bit à 1, et que l'on utilise un contrôle de parité paire, il faudra que le bit de parité soit positionné à 1 pour que le nombre total de bits à 1 soit pair. Ceci permet à l'organe de réception de vérifier si on a le même nombre de 1 dans le caractère à l'arrivée qu'il y en avait au départ. Ce contrôle n'est pas infaillible, mais la probabilité d'avoir plusieurs inversions de bits dans un même caractère est faible.

Exemple :

Caractère transmise (7 bits)	Bit de parité (parité paire)	Bit de parité (parité impaire)
1001100	1	0
0000000	0	1
1010101	0	1

Tableau I.1 : Exemple de parité paire ou impaire

I.4 BUS OU LIAISON [1]

De la même manière, pour chacune des deux natures d'interconnexions (parallèle ou série), il existe deux types d'architectures.

A. Le premier type est la liaison. C'est celle qui a été décrite ci-dessus. Il s'agit de l'interconnexion de deux équipements quelle que soit leur nature (voir la figure I.5) : émetteurs, récepteurs, ou émetteurs/récepteurs.
   Cette catégorie ne présente pas de difficulté particulière, c'est également celle qui est utilisée pour la RS232.

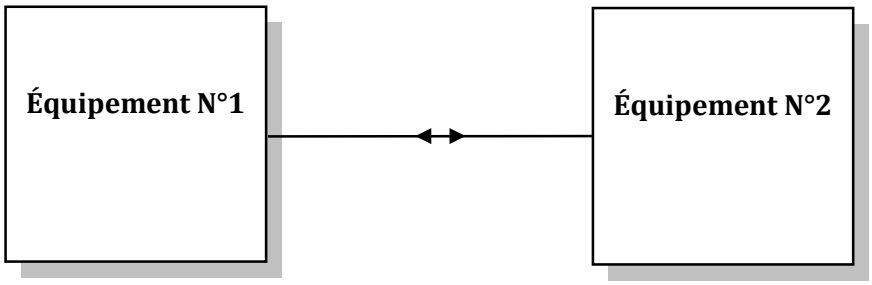


Fig.I.5. Une liaison

B. Le deuxième type d'architecture est le bus donné par la figure I.6. La structure matérielle permet ici d'interconnecter plusieurs équipements entre eux sur le même support physique. Cette architecture est beaucoup plus complexe que l'interface bus des équipements puisse supporter une interconnexion. De même, au niveau logiciel, un protocole est indispensable pour gérer les communications entre les différents équipements.

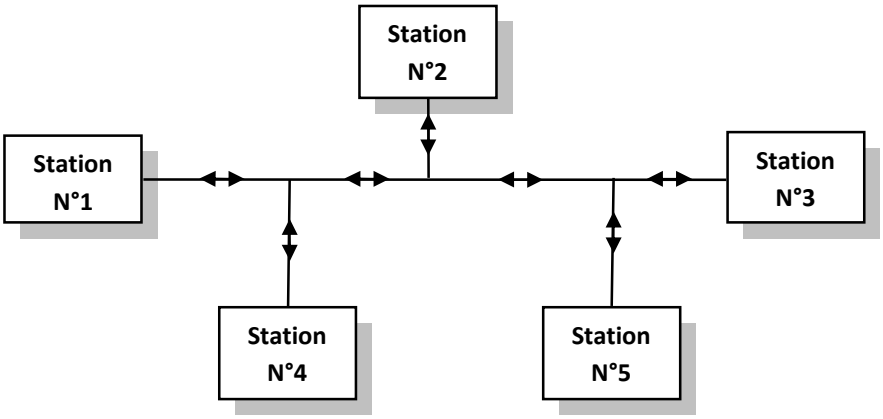


Fig.I.6. Exemple d'un bus

I.5 TYPES OU MODES DE LIAISON

Du point de vue de la capacité de transfert de la ligne de transmission, on distingue trois types de liaisons **unidirectionnelles « SIMPLEX », half duplex** et **full duplex**

I.5.1 Liaison unidirectionnelle « SIMPLEX »

Il s'agit ici vraiment de la plus simple liaison que l'on puisse trouver. La communication simplex est un mode de communication unidirectionnel, dans lequel chaque appareil est soit toujours émetteur soit toujours récepteur. Elle ne permet que la transmission dans un seul sens. [1] Ainsi l'information est transmise de l'émetteur vers le récepteur, supportée par un seul fil auquel s'ajoute la ligne de masse « Ground » comme le montre la figure I.7.

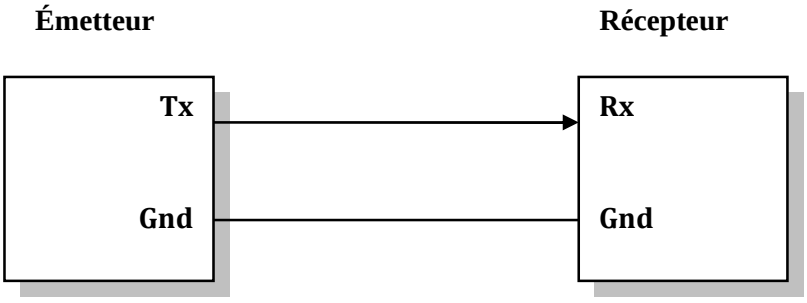


Fig.I.7. Exemple de liaison unidirectionnelle

Ce mode de communication est notamment utilisé quand il n'est pas nécessaire pour l'émetteur d'obtenir une réponse de la part du récepteur. Un circuit électronique comme un capteur qui envoie régulièrement et de manière autonome des données pourra utiliser une liaison simplex.

C'est aussi un mode de communication utilisé pour la diffusion, c'est à dire lorsqu'un même émetteur transmet simultanément à de nombreux récepteurs. Ainsi, la liaison entre un émetteur de télévision et les postes récepteurs est une liaison simplex.

Pour terminer sur un exemple encore plus simple, la télécommande de votre téléviseur communique avec ce dernier par une liaison simplex: quand vous pressez sur un bouton pour changer de chaîne, un train de signaux infrarouge est émis par la télécommande. Mais celle-ci est incapable de savoir si l'ordre a bien été reçu par le téléviseur ou pas. Cette solution est retenue car elle simplifie la conception du système et en réduit les coûts. Par ailleurs, aucun retour de type « accusé de réception » n'est nécessaire pour cette application: en effet, l'utilisateur est parfaitement en mesure de déterminer si la transmission s'est bien passée ou pas, et le cas échéant de ré-appuyer sur le bouton. [2]

D'autres exemples de liaison simplex, on peut donner la liaison de :

- L'ordinateur vers l'imprimante.
- La souris vers l'ordinateur.

La figure I .8 donne le chronogramme de l'information sous la forme de trames envoyées à des temps aléatoires sur un seul sens.

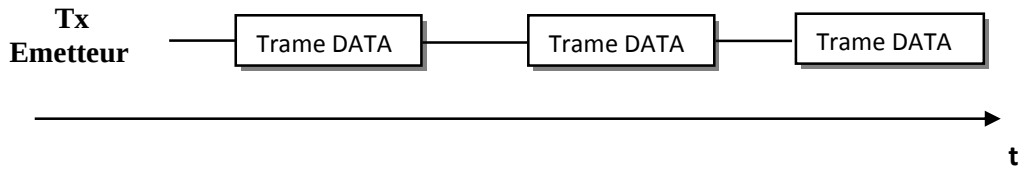


Fig.I.8. Chronogramme d'un exemple de liaison

I.5.2 Liaison bidirectionnelle (ang. **Half-Duplex)** ou **SEMI-DUPLEX**

Ce type de liaison permet le dialogue dans les deux sens mais pas simultanément (donc à chacun son tour). Elle est également peu utilisée. En effet, seuls certains systèmes, lents ou ayant une unité de contrôle RS232 peu performante, fonctionnent encore uniquement dans ce mode comme le montre la figure I.9. [1]

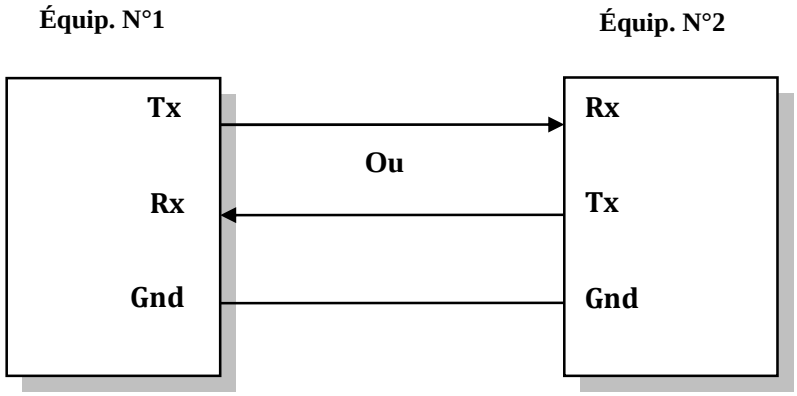


Fig.I.9. Exemple de liaison half duplex

L'avantage de ce système de communication par rapport au mode full-duplex est qu'il réduit par deux le nombre de canaux de communication nécessaires. [2]

Par contre, il impose que les deux systèmes communicants soient en mesure de déterminer qui a le droit de parler. Dans le cas contraire, on risque d'avoir une collision (quand les deux systèmes tentent de parler simultanément) ou un blocage (quand les deux systèmes se mettent à l'écoute simultanément). De plus, un délai supplémentaire peut être induit lors du basculement du sens de communication d'une direction à l'autre. [2]

Plusieurs stratégies sont possibles pour permettre aux deux systèmes de se coordonner. Par exemple, on peut envisager un multiplexage temporel dans lequel un timing précis indique à chacun le sens de la communication. Il peut également exister un canal de commande supplémentaire chargé d'indiquer à chaque périphérique s'il doit être en réception ou en transmission. Dans le même ordre d'esprit, un des deux systèmes peut être par défaut en réception et l'autre en émission, l'inversion ne se faisant qu'à la demande explicite du système en émission. Cette dernière solution s'approche des notions de maître esclave ou encore de jeton. [2]

Un exemple de ce style de communication est le télégraphe Morse: celui-ci est constitué par une ligne électrique munie à chacune de ses extrémités d'un émetteur-récepteur. Quand un opérateur tape un message sur son manipulateur, les impulsions électriques sont transmises sur la ligne et produisent un son et/ou une transcription sur papier à l'autre extrémité. Comme c'est la même ligne qui achemine les signaux dans les deux sens, il s'agit bien d'un système half-duplex dans lequel les opérateurs doivent se coordonner pour ne pas transmettre simultanément. [2]

La figure I.10 [1] donne le chronogramme de ce mode de liaison, il montre bien la façon de fonctionnement de ce mode.

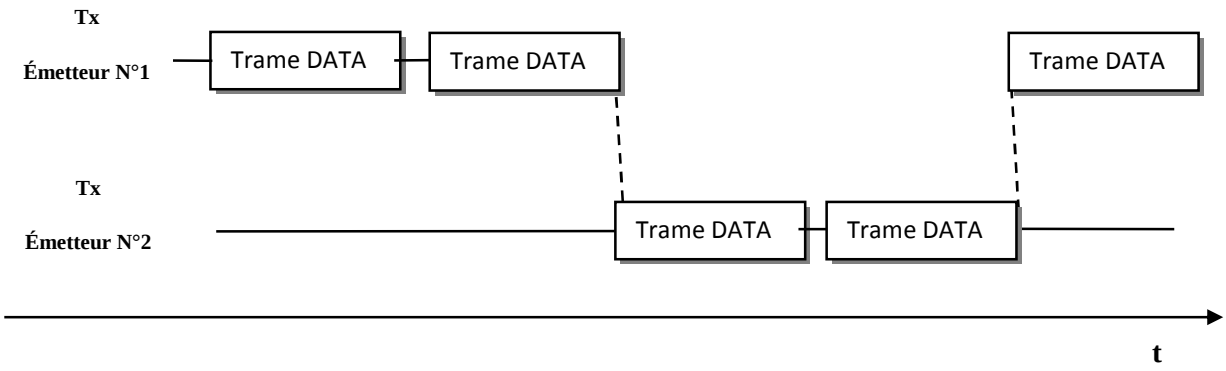


Fig.I.10. Chronogramme d'un exemple de liaison half duplex

I.5.3 Liaison directionnelle simultanée (ang. **Full-Duplex)** ou **Plein-Duplex**

Cette liaison dispose du même câblage que la liaison half duplex mais cette fois les deux interlocuteurs peuvent émettre et recevoir en même temps. Ce mode de transmission est asynchrone, les émissions et les réceptions n'ont pas à être synchronisées. [1]

Dans la figure I.11, nous avons donné un exemple de liaison full duplex tel que l'équipement N°1 représente DTE et N°2 représente DCE.

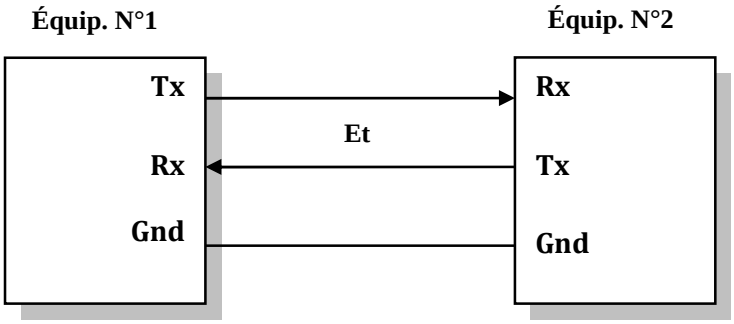


Fig.I.11. Exemple de liaison full duplex

Outre l'existence d'un canal de transmission dédié à chaque sens de communication, ce mode de communication exige aussi que chacun des deux systèmes soit capable de traiter à la fois des données entrantes et sortantes. [2]

Un exemple simple est le téléphone: en effet, lors d'un appel, il est tout à fait possible aux deux correspondants de parler simultanément et de s'entendre l'un l'autre.

De la même manière, certains disques durs permettent de simultanément lire un fichier et en écrire un autre. Cette fonctionnalité requiert un bus de communication full-duplex comme SAS (Serial attached SCSI).

Dans ce cas l'émission de données indépendantes entre l'émetteur n°1 et le second est aléatoire comme le montre la figure I.12. [1]

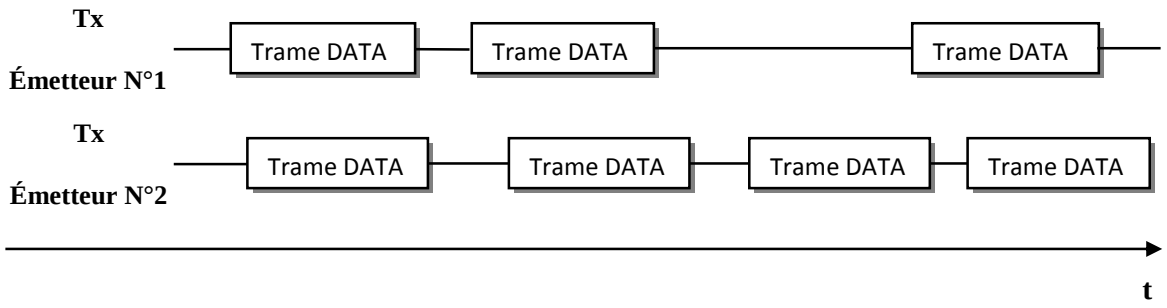


Fig.I.12. Chronogramme d'un exemple de liaison full duplex

I.6 DESCRIPTIONS TECHNIQUES DE LA NORME RS232 [1]

I.6.1 Description électrique des signaux

Nous allons nous intéresser au niveau de tension présente sur la liaison, afin de mieux comprendre son fonctionnement [1], et nous le trouvons dans le tableau I.1.

Niveau Logique	sur SUB-D du PC
«0" Low	+ 12 *V* (de + 5 à + 15 V)
«1» Hi	- 12 *V* (de - 5 à - 15 V)

Tableau I.2 : Équivalence niveaux logiques/ tensions

On peut se rendre compte que les niveaux logiques et les tensions sont inversés. Le niveau haut est représenté par une tension négative et le niveau bas par une tension positive.
On parle couramment des tensions + 12 V / - 12 V, mais la norme définit une fourchette allant de + 5 V / - 5 V à + 15 V /-15 V.

On constatera que sur les équipements, on utilise des tensions de plus en plus basses. Notamment sur les PC portables où désormais les tensions sont généralement de + 5 V / - 5 V. Attention, certaines publications indiquent un minimum à + 3 V / - 3 V pour la RS232. Ces

tensions correspondent aux seuils de comparaison à l'entrée des drivers. Par conséquent, si vous travaillez avec des tensions aussi basses, vous vous exposez à des aléas de fonctionnement causés par des parasites qui seront transformés en glinch par les drivers.

C'est pour descendre effectivement à + 3 V /- 3 V qu'a été créée la norme EIA232.

Normalement, chacune des sorties doit pouvoir fournir 20 mA, mais de nombreux constructeurs ont pris l'initiative d'abaisser la valeur du courant. Ne soyez donc pas surpris d'être limité à 10 mA sur votre PC, voire encore moins s'il s'agit d'un portable. De même, vous ne risquez pas la destruction de l'étage de sortie en cas de surcharge ou de court-circuit car le courant des signaux de sorties est limité.

Les tensions utilisées sont bipolaires, c'est-à-dire qu'elles sont référencées à la masse. Cette dernière est donc très importante pour garantir la qualité de la transmission. En général, il y a très peu de problèmes, car sur les PC, la masse est à la terre. Les choses sont un peu différentes lorsqu'il s'agit d'un PC portable sur batterie ou d'un PC n'étant pas connecté sur la même prise de terre. Il peut apparaître une différence de tension en mode commun entre les deux masses, ceci pouvant occasionner des erreurs de transmission et endommager les composants de communication.

La longueur du câble utilisé pour transmettre les données a son importance. À l'origine, la norme fixait une longueur maximale de câble de 8 mètres. Mais depuis peu, elle spécifie une impédance complexe pour ce câble. Le câble utilisé doit présenter une charge minimale de 3 kΩ et maximale de 7 kΩ, de même le câble ne doit pas posséder une charge capacitive supérieure à 2,5 nF. De par cette nouveauté, en choisissant bien le câble, on peut espérer transmettre sur une longueur d'environ 60 mètres.

De plus, la norme impose un slew rate maximal de 30 V /µs afin de limiter les effets électromagnétiques.

Le débit maximum est à présent défini à 20Kb/s (19 200 bauds), le futur débit maximal doit être de 64Kb/s (en préparation pour la norme Y.28).

Les débits que nous venons d'évoquer sont liés à la norme, le matériel quant à lui, est largement capable de dépasser le mégabit par seconde.

I.6.2 La trame RS232

Transmettre une donnée en mode série impose l'utilisation d'une structure de trame. Dans notre cas, elle est définie par la norme RS232 comme défini dans la figure I.13.

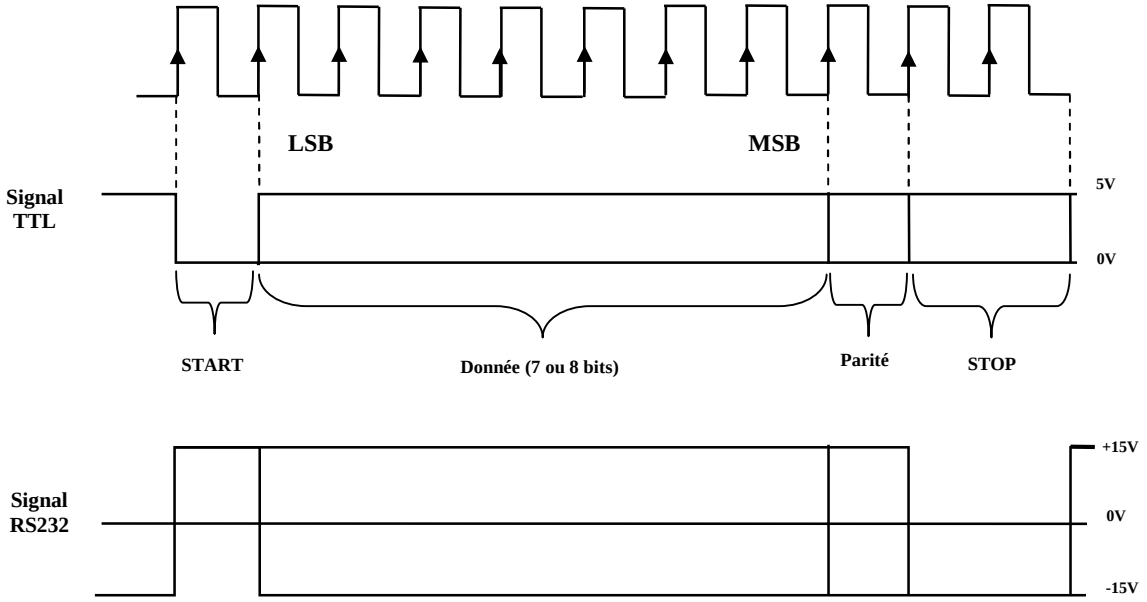


Fig.I.13. Chronogramme d'une trame de données (Trame DATA)

L'état initial vaut « 1 », c'est également l'état de repos, puis on émet le bit de START correspondant à un bit à «0».

Ensuite, on émet la donnée en commençant par le bit de poids le plus faible. Le nombre de bits peut aller, d'après la norme, de 5 à 8, mais usuellement, on utilise seulement des transmissions en 7 ou 8 bits, 7 bits pour ASCII et 8 bits pour les données. De plus, le bios du PC ne permet que ces des formats.

Nous donnons quelques exemples sur les trames de données dans la figure I.14 et la figure I.15

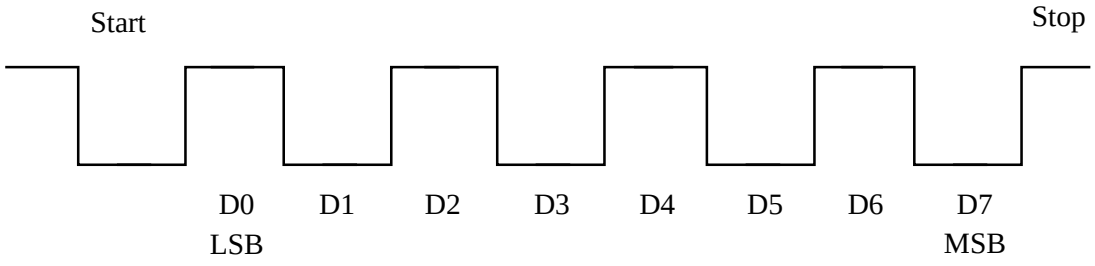


Fig.I.14. Chronogramme d'un exemple de trame de données/Valeur 55 h/85 d, pas de parité, un bit de stop

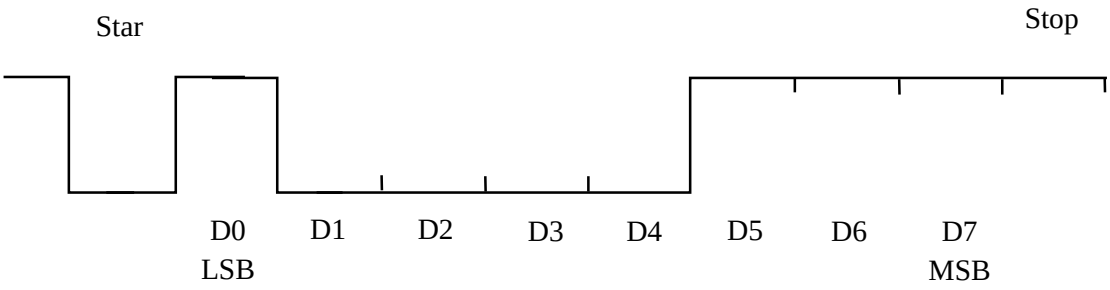


Fig.I.15. Chronogramme d'un exemple de trame de données/Valeur E1h/255 d, pas de parité, un bit de stop

La possibilité d'avoir ou non un bit de parité permet de vérifier la donnée. Son fonctionnement est simple. L'utilisateur choisit la parité paire ou la parité impaire. Ensuite, le bit de parité rend pair ou impair l'ensemble des bits (données + bit de parité).

Exemple N° 1 : 8 bits de données avec parité paire data = 55h/85d/01010101b donc un bit de parité qui vaut «0».

Exemple N° 2 : 8 bits de données avec parité impaire data = E1h/255d/11100001b donc un bit de parité qui vaut «1».

Enfin, on transmet 1 ou 1,5 ou 2 bits de stop. L'un des intérêts de faire varier la durée du bit de stop, est de laisser au récepteur d'avantage de temps pour être à nouveau prêt à recevoir une trame.

L'horloge n'est bien sûr pas la même des deux côtés de la liaison, car elle n'est pas transmise. Il s'agit en fait de deux horloges (une de chaque côté) qui sont synchronisées par le START de la trame et qui doivent avoir la même fréquence. Une tolérance de 5 % d'écart entre les deux fréquences est acceptée, au-delà, de nombreuses erreurs de réception sont à prévoir.

Cette horloge n'est pas définie en hertz par la norme, elle est indiquée en débit d'informations, c'est-à-dire en bauds.

L'utilisateur peut donc vérifier la vitesse de transmission à l'oscilloscope avec un octet facile à identifier.

La figure I.16 représenté durée d'un bit à 2400 Bauds par exemple de la transmission de l'octet : 55h / 85d / 01010101b.

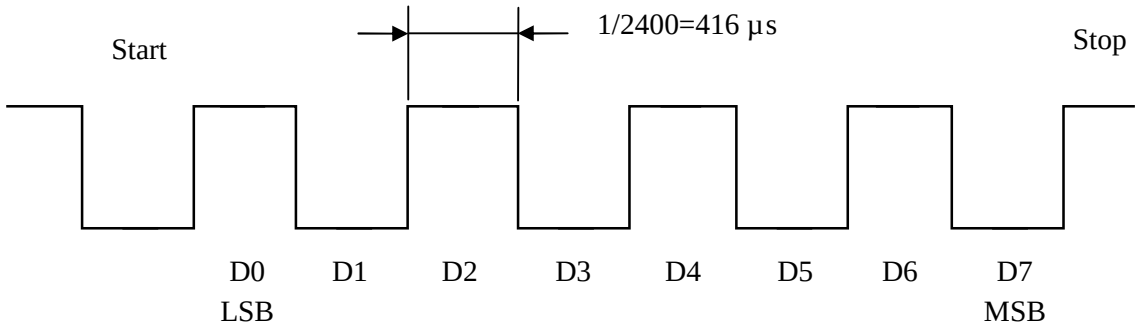


Fig.I.16. Durée d'un bit à 2400 Bauds. Exemple de la transmission de l'octet : 55h / 85d / 01010101b

Et le tableau I.2 représenté les déférentes vitesses « Débits » en Baud et le temps de maintien d'un bit.

Débits en Bauds	**Temps de maintien d'un bit**
75	**13,333 ms**
150	**6,666 ms**
300	**3,333 ms**
600	**1,666 ms**
1 200	**833 us**
2400	**416 us**
4800	**208 us**
9600	**104 us**
19200	**52 us**

Tableau I.3 Tableau de correspondance entre la valeur de débit et le temps de maintien des données.

I.6.3 Fonction des signaux

On peut rassembler les signaux de la RS232 en deux groupes de fonctions. D'une part, les signaux de communication proprement dits (Tx, Rx), d'autre part, les signaux d'HandShaking (DTR, DSR, RTS, CTS, DCD et RI). Le premier groupe est indispensable à la communication. Le second est « optionnel » et seul, il ne peut servir à rien.

Dans le tableau ci-dessous (tableau I.3), vous trouverez pour chaque signal: sa désignation définie par la norme, suivie d'une traduction interprétée, et enfin, entre parenthèse, la fonction remplie dans le protocole « HandShake »

Nom	N° Broche SUB-D9	Sens pour un DTE	Désignation
Tx	3	Sortie	Transmit Data-Donnée émise
Rx	2	Entrée	Receive Data-Donnée reçue
DTR	4	Sortie	Data Terminal Ready-Terminal de données prêt (prêt à recevoir ?)
DSR	6	Entrée	Data Set Ready-Mise à disposition des données (donnée prête ?)
RTS	7	Sortie	Request To Send-Demande de transmettre (donnée à transmettre !)
CTS	8	Entrée	Clear To Send-Ouverture de transmission (prêt à recevoir !)
DCD	1	Entrée	Data Carrier Detect-Détection de message de données (liaison établie !)
RI	9	Entrée	Ring Indicator-Indicateur de communication (demande de comm. du DCE)

Tableau I.4 : Description fonctionnelle des signaux.

I.6.4 Interconnexion des équipements

A. Liaison complète DTE/DCE

À l'origine, la liaison RS232 a été décrite uniquement pour faire communiquer un ordinateur avec un périphérique. On a même prévu des appellations particulières pour l'ordinateur (DTE) et pour le périphérique (DCE). (Tableau I.5)

Un DTE désignait, à l'origine, uniquement les ordinateurs et les terminaux. À présent, on

y retrouve quasiment la totalité des équipements pourvus d'une liaison RS232.

Nom	Signification
DTE	Data Terminal Equipement
DCE	Data Communication Equipement

Tableau I.5 : Signification DTE/DCE

Un DCE désignait auparavant tous les périphériques comme les imprimantes, les traceurs, et les modems entre autres. Désormais, ce groupe est pratiquement délaissé, seuls certains modems sont encore en mode DCE.

Nous allons donc commencer par décrire le dialogue prévu par la norme à l'origine, et la figure I.17 montre la liaison complète entre un DTE et DCE.

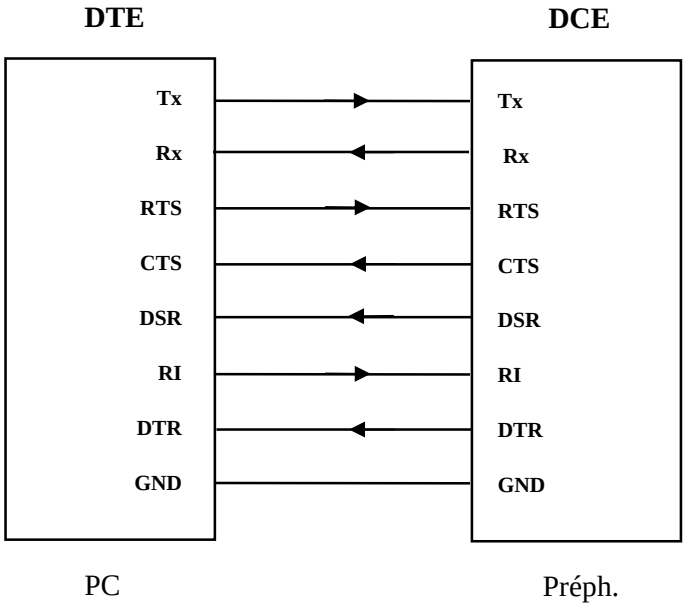


Fig.I.17. Liaison complète entre un DTE et DCE

Attention dans ce cas particulier, le sens des signaux d'un DCE est l'opposé de celui d'un DTE!

Cette manière de communiquer entre un DTE et un DCE peut être assimilée à un protocole qui est appelé « HandShake ».

Ce protocole permet de réguler le débit de transmission directement par le matériel et non par le logiciel. En effet, du point de vue du logiciel, ce protocole est transparent. Ce sont les UARTs qui gèrent le changement d'état des signaux.

B. Liaison complète DTE/DTE

De nos jours, les conditions d'utilisation ont beaucoup changé. Il ne s'agit plus de faire

communiquer un DTE avec un DCE mais désormais, dans de nombreux cas, de faire communiquer deux DTE ensembles. (La figure I.18)

En effet, on est souvent amené à connecter deux PC ensembles ou par exemple à connecter un PC avec un appareil de mesure "" intelligent répondant aux spécifications d'un DTE.

Par conséquent, il faut adapter le câblage à la situation, car c'est la seule chose modifiable dans ce cas. C'est cette modification de câblage qui rend la communication toujours possible.

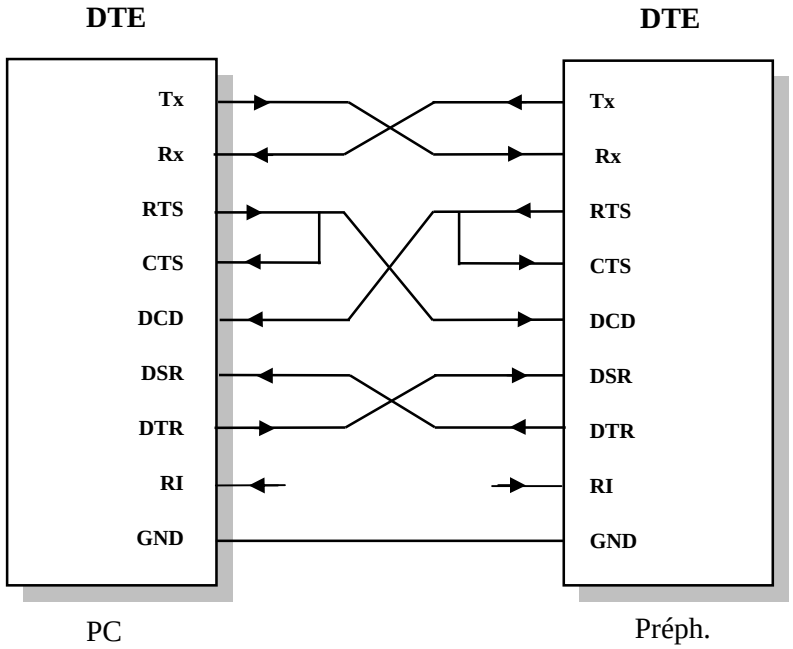


Fig.I.18. Liaison complète entre deux DTE (Null Modem).

On remarque donc le croisement des différents signaux ainsi que le rebouclage du RTS sur le CTS. Ce câblage est la seule manière de remplir le mode de fonctionnement de la liaison. De plus, le signal RI ne sera pas utilisé. Ce type de liaison porte le nom de Null Modem.

C. Liaison deux fils DTE/DTE

Il s'agit de la liaison la plus simple (voir la figure I.19). C'est une liaison unidirectionnelle appliquée à la RS232.

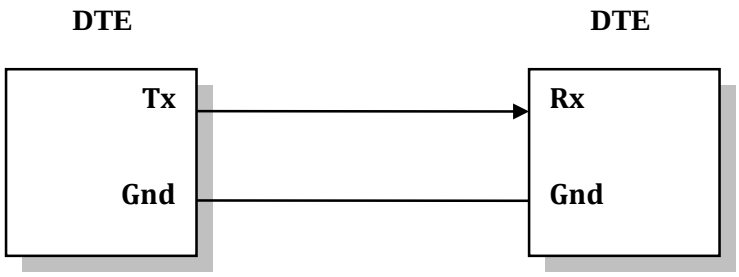


Fig.I.19. Liaison deux fils (DTE/DTE)

D. Liaison trois fils DTE/DTE

Ce type de câblage est sans doute le plus utilisé, car il répond au mieux à un besoin de facilité d'utilisation ainsi que de mise en œuvre. C'est également ce type de liaison qui est utilisé sur les microcontrôleurs.et la figure I.20 explique cette liaison.

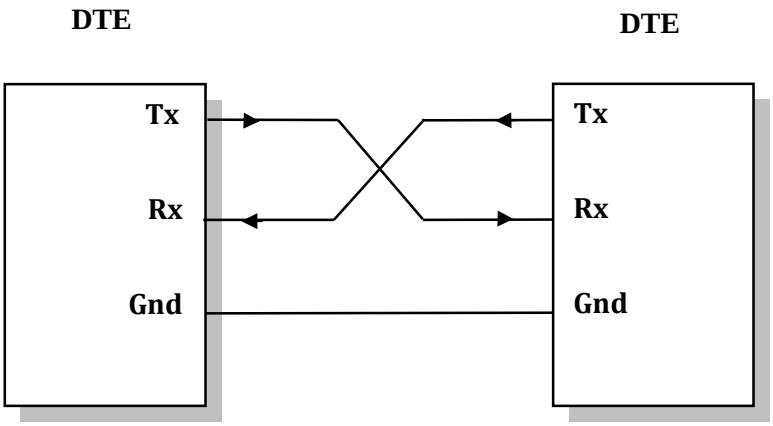


Fig.I.20. Liaison trois fils (DTE/DTE)

E. Liaison complète vers trois fils DTE/DTE

On peut être confronté au besoin d'interconnecter deux équipements différents. L'un est prévu pour fonctionner en mode liaison complète alors que l'autre ne peut communiquer qu'en trois fils.

La solution consiste à reboucler l'équipement « liaison complète » sur lui-même, la figure I.21 explique la liaison complète vert équipement trois fils et rebouclage les signaux.

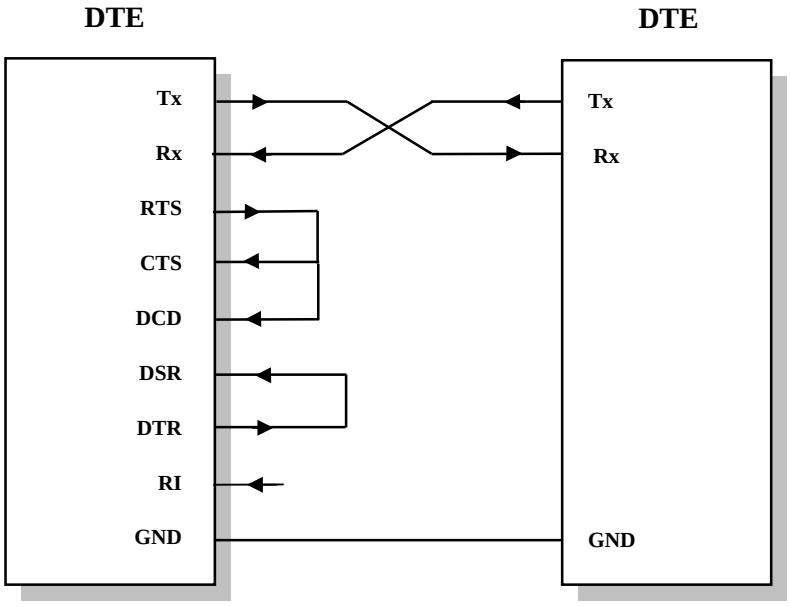


Fig.I.21. Liaison complète vert équipement trois fils (DTE/DTE)

I.7 LES PROTOCOLES

Lors de la mise en place d'une communication, nous sommes confrontés à un problème de régulation des données. Dans le premier cas de figure, la communication est parfaitement maîtrisée et aucun artifice n'est nécessaire. Dans un deuxième cas, le récepteur peut être occupé à d'autres tâches et doit pouvoir réguler lui-même le flux de données. [1]

I.7.1 Handshake [1]

La méthode la plus courante, pour réguler le flux de données, est l'HandShaking. Elle utilise une liaison full duplex complète. Ce protocole est géré directement par l'UART.

Vous trouverez sur le chrono gramme suivant (Figure I.22), le mécanisme d'une communication.et le tableau I.6 expliquer déroulement chronologique de l'HandShaking.

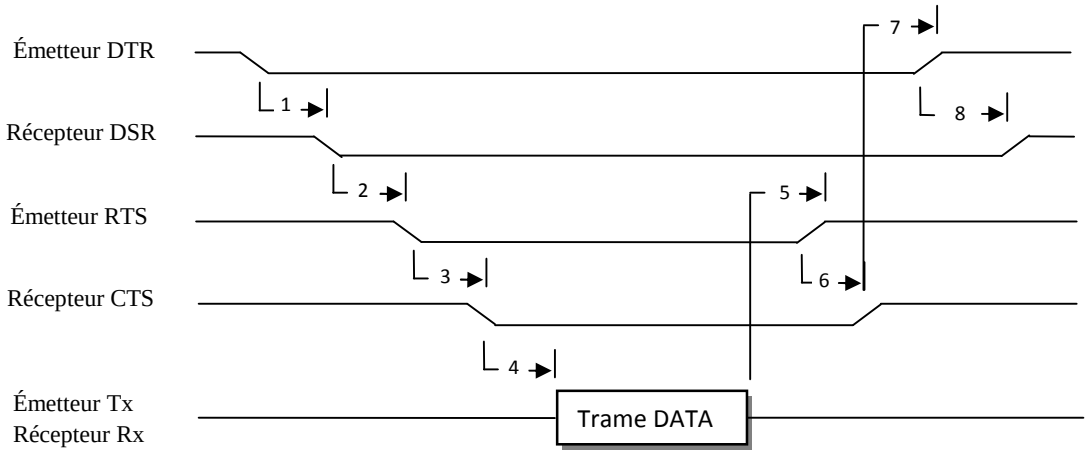


Fig.I.22. Chronogramme protocole matériel

1	L'émetteur demande si le récepteur peut recevoir des données (7)
2	Le récepteur répond, oui je peux recevoir des données (!)
3	L'émetteur demande s'il peut transmettre maintenant des données (7)
4	Le récepteur répond, oui je suis prêt maintenant (1)
Data	Transmission de la trame de données. Une fois la donnée transmise, on remet les signaux dans l'ordre initial tout en accusant réception des informations
5	Fin de transmission
6	OK fin de transmission
7	Fin de requête
8	OK Protocole terminé

Tableau I.6 Déroulement chronologique de l'HandShaking.

Ce protocole n'a pas de contrainte de temps importante, car il s'agit d'une logique asynchrone détectant les changements d'état, non pas sur front mais sur niveau.

I.7.2 Xon-Xoff [1]

On peut également signaler, qu'une autre méthode que le HandShaking est possible pour contrôler le flux des données.

Cette méthode est communément appelée Xon-Xoff. Elle utilise comme support physique une liaison full duplex avec trois fils au minimum, et fonctionne de la manière suivante: l'émetteur transmet ses données au récepteur. Lorsque le récepteur est plein, il émet à l'émetteur un Xoff. L'émetteur par la réception de ce Xoff, arrête d'émettre les données et attend un Xon de la part du récepteur pour reprendre l'émission, et la figure I.23 montre tout.
Dans la table de caractères, ASCII Xon est égal à 17 et Xoff à 19 en décimale.

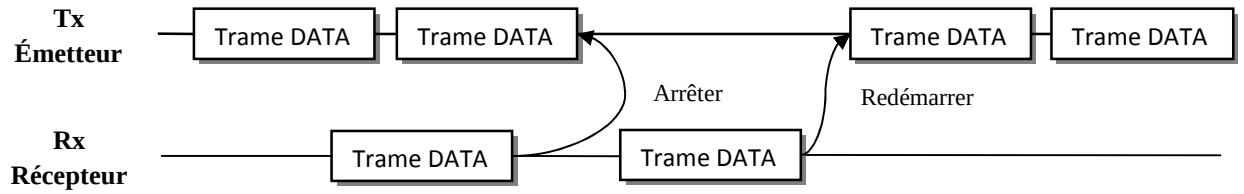


Fig.I.23. Chronogramme protocole Xon-Xoff

I.7.3 ETX/ACK [3]

Dans ce protocole, les données sont envoyées par blocs (trames). L'émetteur envoie un bloc de données auquel il ajoute le code ASCII 03h (appelé ETX pour End of Text) puis se met en attente. De son côté, le récepteur traite les données et renvoie à l'émetteur le code ASCII 06h (appelé ACK pour ACKnowledge) pour signaler qu'il a bien reçu le bloc de données précédent et qu'il est prêt à recevoir de nouveau des données. Le récepteur peut dans certains cas renvoyer le caractere ASCII 15h (appelé NACK pour Negative ACKnowledge) pour signaler qu'il a détecté une erreur dans la transmission.

I.8 CONCLUSION

La transmission des signaux numérique peut se faire en parallèle ou en série. Mais pour les longues distances c’est la transmission série qui prédomine. Parmi les deux types c’est la liaison asynchrone qui a un rapport direct avec le circuit UART à décrire en VHDL.

Chapitre II

UART (Universal Asynchronous Receiver Transmitter)

II.1 INTRODUCTION

Un UART, Universal Asynchronous Receiver/Transmitter, est un gestionnaire de la liaison RS232.pour l'émission le circuit l'UART permet de faire la conversion parallèle/série pour transmettre sur un même fil les données. Du réception, l'UART reçois une information série qu'il rend parallèle, il assure donc la conversion série parallèle. L'UART utilise des registres parallèle/série ou série/parallèle mais on ne peut pas l'assimiler à un simple registre à décalage. Il restera un circuit complexe programmable gérer par plusieurs signaux de commandes.

Parmi les blocs qui constituent l'UART, on trouve : l'émetteur et le récepteur. Ces deux derniers ont besoin d'une horloge qui détermine leur vitesse en « Baud ».

Tous ces blocs seront développés dans ce chapitre.

II.2 LE PRINCIPE D'UNE LIAISON SÉRIE

Dans une liaison série les bits d'informations sont transmis les uns derrière les autres sur un seul et même fil. Les mots transmis sont, le plus souvent de, 7 ou 8 bits.

L'échange série de données nécessite au préalable une conversion parallèle / série réalisée par un registre à décalage comme indiqué par la figure II.1. [4] A l'autre extrémité du câble, un deuxième registre à décalage reçoit les informations "série" pour les convertir en mots parallèles. La fonction de ce registre est la conversion série parallèle.

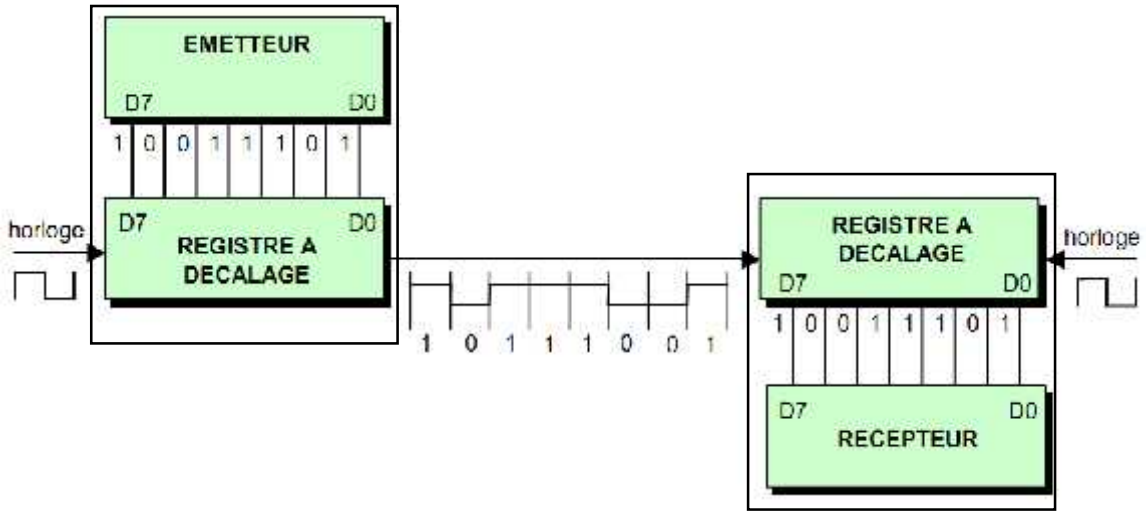


Fig.II.1. Principe d'une liaison série

II.3 L'INTERFACE SÉRIE

Une interface série permet d'échanger des données entre le microprocesseur et un périphérique bit par bit comme montré à la figure II.2.

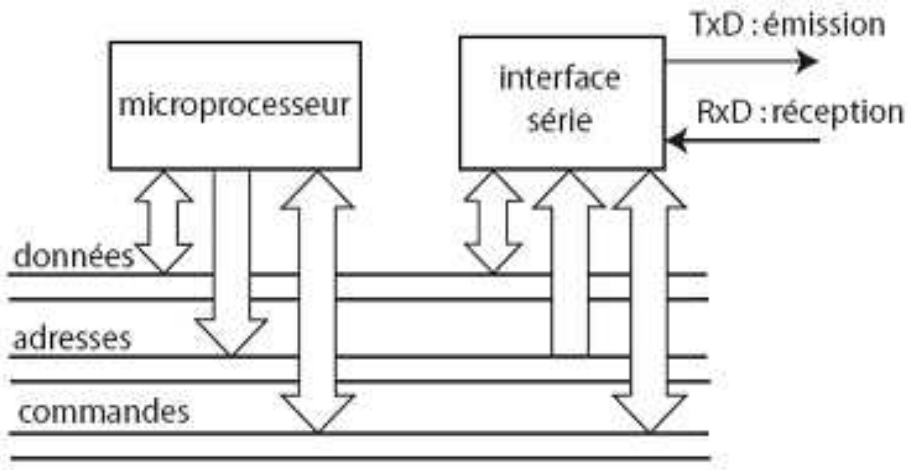


Fig.II.2. Principe de l'interface série

Avantage : diminution du nombre de connexions (1 fil pour l'´emission, 1 fil pour la réception).

Inconvénient : vitesse de transmission plus faible que pour une interface parallèle.

Parmi les plus célèbres des interfaces série, on trouve l'UART (Universal Asynchronous Receiver Transmitter).

II.4 STRUCTURE DE L'UART

Le schéma-bloc de l'UART est donné à la figure II.3. [5]

Il contient habituellement les composants fondamentaux suivants :

- Un générateur d'horloge.
- Un canal d'émission (TRANSMITTER)
- Un canal de réception (RECEIVER).
- Autres sous-fonction d'UART.

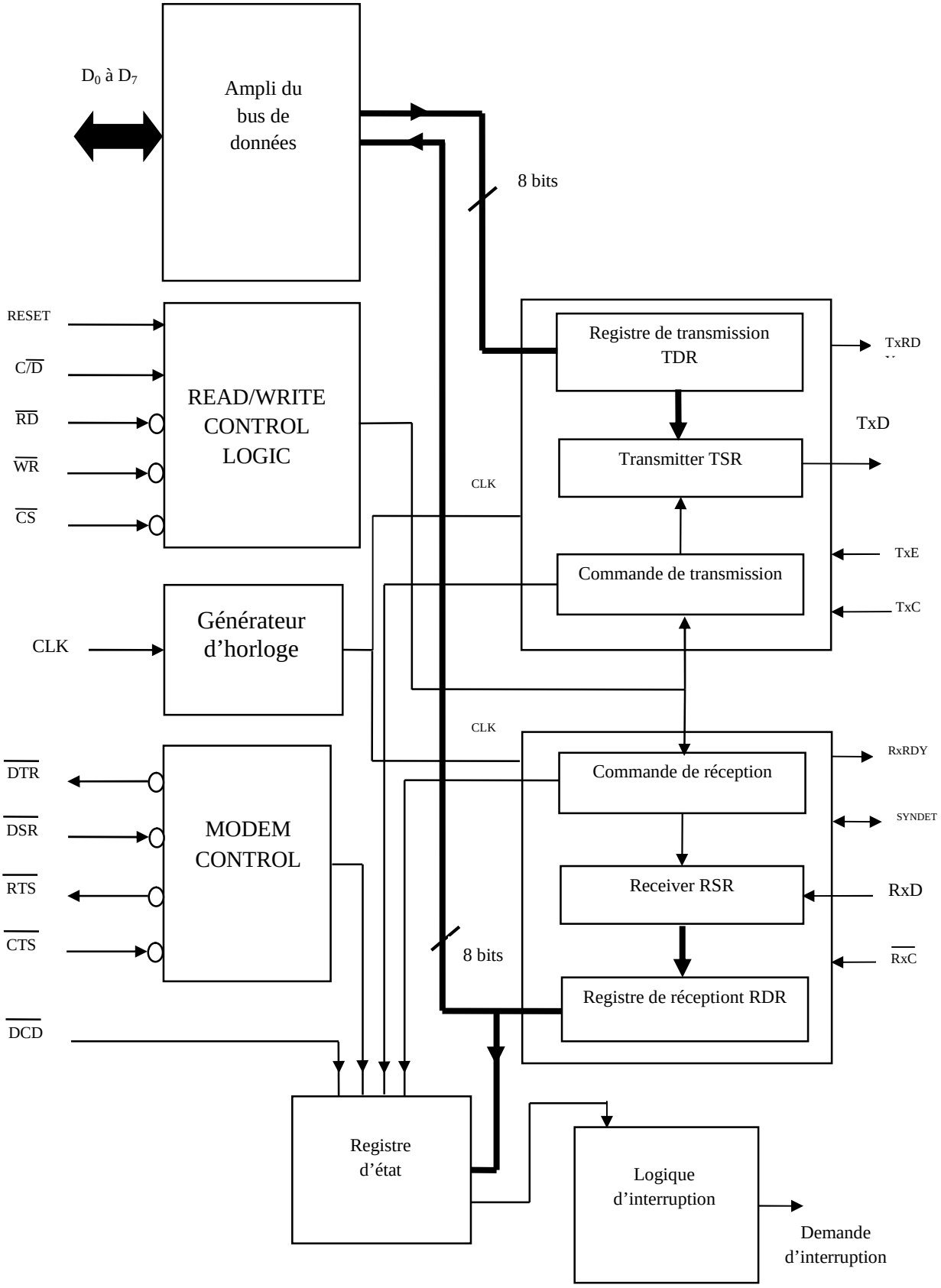


Fig.II.3. Schéma bloc et brochage d'un UART

II.4.1 Un générateur d'horloge

A partir de la fréquence de l'UART Le générateur d'horloge génère plusieurs fréquences parmi les quelles on peut choisir une seule, pour être utilisée, par l'émetteur ou le récepteur pour obtenir la vitesse désirée. Les fréquences générées obéissent à la relation suivant : on suppose que la fréquence de l'UART est f_{UART} .Pour avoir une vitesse V_{Baud}, on calcule X le nombre d'impulsions après lequel on génère une impulsion qui sera utilisée comme vitesse de l'émetteur ou le récepteur.

$$V_{Baud} = f_{UART}(Hz) / 16* X$$

II.4.2 Un canal d'émission (TRANSMITTER) [5]

Ce canal est constitué par:

- un registre de conversion parallèle-série (P-S);
- une logique de mise en forme de la séquence binaire à émettre (insertion de bits de cadrage: parité, START, STOP);
- une logique d'état et de contrôle.

Ce canal est relié en entrée au bus interne qui le met en relation avec le microprocesseur, et en sortie à la ligne d'émission série TxD (Transmitted Data). Le rythme de l'émission est commandé par l'horloge d'émission TxC (Transmitter Clock), lors d'un front descendant.

La logique de contrôle d'émission gère une bascule d'autorisation d'émission TxEN (Transmit Enable) qui doit être mise à "1" par programme avant toute opération d'émission; si cette condition est remplie, il faut encore que le signal CTS (Clear To Send) soit activé (CTS = 0) pour qu'une émission commence.

Logique de contrôle gère également deux signaux d'état: TxE (Transmitter Empty, émetteur vide) et TxRDY (Transmitter ReaDY), qui apparaissent sur les broches de même nom, ainsi que le mot d'état de l'UART.

TxE à l'état haut indique que le registre d'émission est vide; il est remis à "0" automatiquement dès la réception d'un caractère issu du microprocesseur. La valeur du bit D_2 dans le mot d'état de l'UART est l'image exacte de l'état de la broche de même nom. Le signal TxE, qui indique une fin d'émission, peut être utilisé par exemple par un processeur pour savoir quand commuter une ligne en mode semi-duplex.

TxRDY à l'état haut indique que le canal émetteur est prêt à accepter un caractère à

émettre. On peut utiliser la broche TxRDY comme ligne de demande d'interruption vers le microprocesseur; une demande est masquée si TxEN=O. TxRDY est remis à "0"' automatiquement lors d'une opération d'écriture par le microprocesseur (au front descendant de WR). Il faut noter que le bit TxRDY dans le registre d'état (D_0) N'EST PAS LA COPIE de l'état de la broche de même nom:

- bit d'état: TxRDY= Tampon de sortie vide (Empty);
- broche TxRDY = (Tampon vide) ET (CTS=O) ET (TxEN=l).

II.4.3 Un canal de réception (RECEIVER) [5]

Ce canal est constitué par:

- un registre de conversion série-parallèle (S-P);
- une logique de mise en forme de caractère (suppression des bits de cadrage: parité, START, STOP);
- une logique d'état et de contrôle.

Ce canal est relié en sortie aux 8 lignes du bus interne qui le met en relation avec le microprocesseur. Pour l'entrée, il est relié à la ligne RxD (Received Data). Le rythme de réception est fixé par l'horloge RxC (Receiver Clock), active sur son front montant.

La logique de contrôle du canal de réception assure les fonctions suivantes.

1) Elle gère une bascule d'autorisation de réception RxEN (Receive Enable), accessible par programme; RxEN=l indique: Réception autorisée.

2) Elle gère une bascule d'état permettant d'interpréter le niveau bas de la ligne RxD: ligne inutilisée ou condition de rupture (BREAK); cette dernière correspond à un intervalle entre caractères émis où RxD est à l'état SPACE au lieu de MARK. Cette bascule est associée à la ligne BRKDET (Break Detect), qui passe à l'état" 1" chaque fois que la ligne RxD reste au niveau bas pendant deux cycles complets de caractère. Dès que RxD repasse à "1", BRKDET revient à "0".

3) Elle gère une bascule qui indique si un état "1" valide a été détecté sur la ligne d'entrée RxD après un RESET général; aucune donnée ne sera acceptée venant de RxD tant que cette condition n'a pas été réalisée; par contre, il suffit qu'elle l'ait été une seule fois après un RESET.

4) Elle vérifie qu'une transition haut vers bas sur la ligne RxD correspond à

un bit START, en testant de nouveau l'état de la ligne au milieu de la cellule. S'il y a confirmation du niveau bas. Le canal récepteur interprète cette situation comme le début d'un caractère à recevoir; sinon, la transition précédente est considérée comme un parasite (ang. Transient Noise Spike) et ignorée.

5) Elle recalcule la valeur du bit de parité au fur et à mesure de la réception des bits de données et compare avec le bit de parité reçu; s'il y a discordance, une bascule d'erreur de parité est mise à "1" (ang. Parity Error).

6) Elle vérifie la présence d'un bit STOP à la fin du caractère; si ce bit est absent, une bascule d'erreur de trame (ang. Framing Error) est mise à "1". Notons qu'un seul bit STOP est vérifié par le canal, même si l'UART a été programmé pour fonctionner avec 1,5 ou 2 bits STOP.

7) Elle gère une bascule d'état RxRDY (Receiver Ready) qui indique qu'un caractère complet vient d'être reçu. Ce signal peut être utilisé pour une demande d'interruption vers le microprocesseur afin de provoquer la lecture du tampon d'entrée. Le signal de lecture RD remet automatiquement RxRDY à "0".
8) Elle signale si un nouveau caractère a été reçu avant que le précédent n'ait été lu par le microprocesseur, en mettant à "1" une bascule d'erreur d'écrasement, (ang. Overrun Error).

II.4.3 Autres sous-fonction d'UART [5]

Nous avons décrit assez longuement la structure des canaux d'émission et de réception parce qu'ils constituent les deux sous-fonctions essentielles de l'UART

Les autres sous-fonctions sont les suivantes

1) Un jeu de tampons d'E/S: un tampon d'entrée pour les commandes ou caractères à émettre; deux tampons de sortie, respectivement pour les mots d'étal pour les données reçues. Les E/S sont-relatives au l'UART.
2) Une logique de contrôle de MODEM, utilisant les signaux suivants
   - DTR : Sortie, Data Terminal Ready-Terminal de données prêt (prêt à recevoir).
   - DSR : Entrée, Data Set Ready-Mise a disposition des données (donnée prête).

- RTS : Sortie, Request To Send-Demande de transmettre (donnée à transmettre).
- CTS : Entrée, Clear To Send-Ouverture de transmission (prêt à recevoir).

3) Une logique de sélection et de lecture-écriture, apte à décoder les signaux de commande suivants:
   - RESET: réinitialisation ;
   - C/D (Command/Data): transfert d'un mot de commande (C=1) ou d'un caractère de donnée (D=O);
   - RD (Read), WR (Write): lecture, écriture;
   - CS (Chip Select): sélection de boîtier ;
   - CLK (Clock): horloge externe.

II.5 EXEMPLE L'UART 8250

Vu son importance et sa complexité, on va considérer un exemple, c'est l'UART INTEL 8250.on va voir comment il est relié au microprocesseur à travers le port COM1 (ou le port COM2).pour le microprocesseur l'UART est une suite de registres à 8 bits (un octet) ; voire la figure II.4

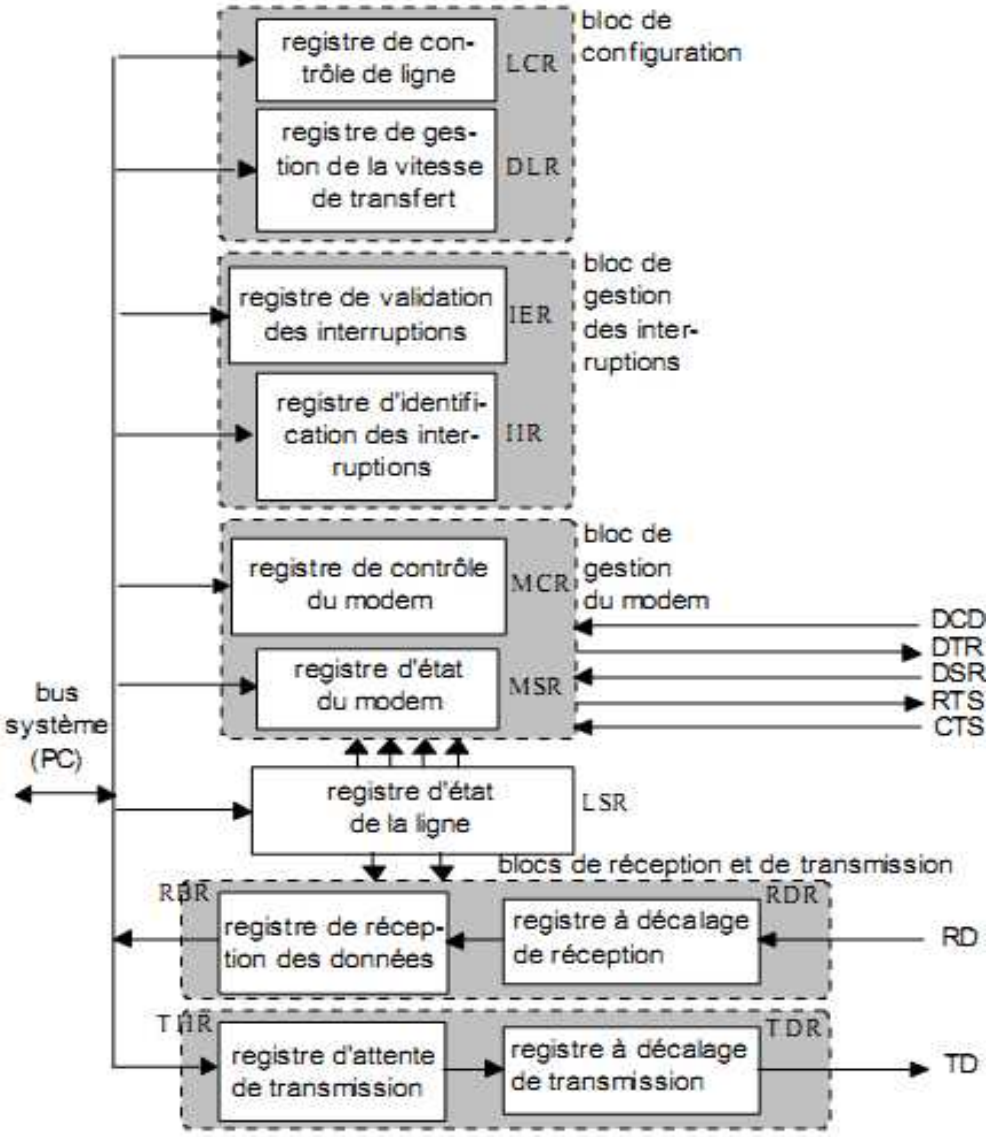


Fig.II.3. Les registres du 8250

Ces registres possèdent les adresses et les fonctions du tableau II.1 [17]

Registre	intitulé:	COMI:	COM2:
LCR	Line Control register	3FB	2FB
DLR	Divisor Latch Register	3F8	2F8
LSR	Line Status Register	3FD	2FD
IIR	Interrupt Identification Register	3FA	2FA
IER	Interrupt Enable Register	3F9	2F9
MCR	Modem Control Register	3FC	2FC

MSR	Modem Status Register	3FE	2FE
RBR	Receiver Buffer Register	3F8	2F8
THR	Transmitter Holding Register	3F8	2F8

Tableau II.1 : Les registres de l 'UART

A. Registre commande de la ligne (LCR)

Ce registre de commande de la ligne sert à configurer le port série avec les paramètres de communication: (voir tableau II.2)

Bit	Fonction
	Longueur des données
2	Nombre de bits de stop
3à5	Parité
6	Signal d'arrêt (break)
7	Accès au diviseur de fréquence

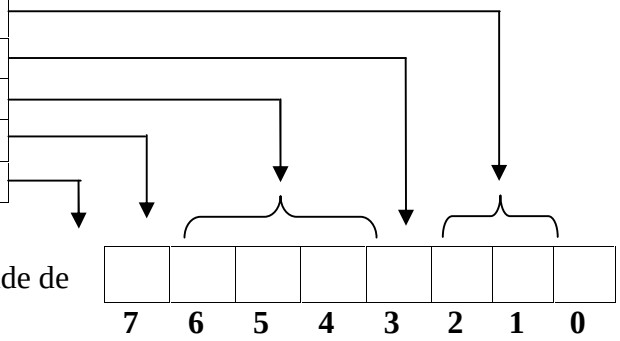


Tableau II.2 : Tableau du registre de commande de ligne

- Longueurs des données (bit 0 et 1)

Bits		Nombre de bit de données
1	0	
0	0	5
0	1	6
1	0	7
1	1	8

- Nombre de bits de stop (bit 2)

Bit 2	**Nombre de bits de stop**
0	1
1	1,5 : si le nombre de bits de données = 5 2 : si le nombre de bits de données ≠ 5

- Parité (bits 3 à 5)

Bits			**Type de parité**
5	4	3	
-	-	0	Pas de parité

0	0	1	Parité impaire
0	1	1	Parité paire
1	0	1	Parité travail (= 1)
1	1	1	Parité repos (=0)

- Signal d'arrêt (break)

Bit 6	**Ligne TD**
0	1 (ligne dans l'état de repos)
1	0 (force la ligne **TD** à 0 : break)

- Accès au diviseur de fréquence (fixer la vitesse de transmission sur le port série)

Bit 7	**Ligne TD**
0	Registre 0 = registre d'émission ou de réception Registre 1 = registre de contrôle des interruptions
1	Registre 0 et 1 : registre diviseur de fréquence de l'UART

B. Registres de sélection de la vitesse de transfert (DLR) ou (DLH & DLL)

Il s'agit de deux registres de huit bits dont les adresses sont fournies dans le tableau 2.3. Le contrôleur gère une horloge fonctionnant à f_0=1,8432 MHz, et délivre une fréquence de base f_1, égale à f_0/16. Pour parvenir à une vitesse de transfert V (en bps), on divise f_1 par V et on convertit le résultat en hexadécimal. Le tableau II.3 donne l'équivalence entre le contenu des registres (16 bits) et la valeur de la vitesse en bits par secondes.

vitesse choisie:	contenu de **DLR** :
50	0900
75	0600
110	0417
150	0300
300	0180
600	00C0
1200	0060
2400	0030
4800	0018
9600	000C

Tableau II.3: Programmation du débit en bps

C. Registre état de la ligne (LSR)

0							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**

Le registre d'état de la ligne permet de connaître à tout instant l'état de la ligne

RS232 et des erreurs associées à la transmission des données. La lecture de ce port remet à 0 tous les bits d'erreur. La signification des bits est donnée sur le tableau II.4 :

Bit	Fonction
0	Arrivée d'un caractère : donnée prête en réception
1	Ecrasement du caractère précédent : Engorgement
2	Erreur de parité
3	Configuration de trame incorrecte : protocole non respecté
4	Communication interrompue (break)
5	Registre d'émission vide
6	Registre de transfert vide
7	Non utilisé (0 permanent)

Tableau II.4 : Tableau du registre d'état de la ligne

D. Registre état des interruptions (IIR)

Une fois les interruptions activées, le registre d'état des interruptions informe sur l'état des interruptions en cours. Seuls les bits 0 à 2 de ce registre sont utilisés comme suit:

X	**X**	**X**	**X**	**X**			
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**

Le tableau II.5 explique le fonctionnement de ce registre

Bits			**Signification de l'événement**	**Action à réaliser dans la routine d'interruption**
2	**1**	**0**		
0	0	1	Aucune interruption en cours	
0	0	0	Changement d'état du modem	Lire le registre d'état du modem afin de déterminer la cause de l'interruption
0	1	0	Registre d'émission vide: fin d'émission	Ecrire dans le registre de transmission la donnée à émettre
1	0	0	Arrivée d'un caractère: fin de réception	Lire le registre de réception
1	1	0	Erreur de transmission	Lire le registre d'état de la ligne afin de déterminer la cause de l'interruption

Tableau II.5: Tableau du registre d'état des interruptions

Remarque: Lorsque le bit 0 est à 1, c'est signe qu'aucune interruption n'est en cours. Dans le cas contraire, ce bit 0 et les combinaisons des bits 1 et 2 spécifient l'événement qui a déclenché l'interruption.

E. Registre contrôle des interruptions (IER)

X	X	X	X				
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**

L'UART est capable de générer une interruption dès qu'un évènement se présente. Quatre évènements sont répertoriés dans les bits 0 à 3 du registre de contrôle des interruptions comme montre dans le tableau II.6.

Bit	Evénement
0	Arrivée d'un caractère: Fin de réception
1	Registre d'émission vide: Fin d'émission
2	Erreur en transmission
3	Changement d'état du modem
4à7	Non utilisés

Tableau II.6 : Tableau du registre de contrôle des interruptions

F. Registre commande du modem (MCR)

Ce registre de commande du modem sert à initialiser les lignes de contrôle de flux de la liaison série et donne accès à la gestion sous interruption des entrées-sorties du port RS232, le tableau II.7 représenté le fonctionnement de ce registre.

X	X	X					
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**

Bit	**ligne**	**état**	
0	DTR	**0**	0
		1	1 (connecté)
1	RTS	**0**	0
		1	1 (connecté)
2	OUT 1	Sortie utilisateur n° 1	
3	OUT2	Sortie utilisateur n02	
		0	Inhibées
		1	Autorisées
4	Bouclage de l'UART	**0**	Non bouclé
		1	TD relié à RD en interne

Tableau II.7 : Tableau du registre de commande du modem et connexion

G. Registre état du modem (MSR)

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**

Le registre d'état du modem permet de connaître à tout instant l'état des fils de contrôle du flux de la ligne ainsi que celui des signaux en provenance du modem. La signification des bits de ce registre est donnée sur le tableau II.8.

Bit	Fonction
0	Changement d'état du fil CTS
1	Changement d'état du fil DSR
2	Changement d'état du fil RI
3	Changement d'état du fil DCD
4	Etat du fil CTS
5	Etat du fil DSR
6	Etat du fil RI
7	Etat du fil DCD

Tableau II.8 : Tableau du registre d'état du modem

H. Registre de transfert THR

Lorsqu'un caractère doit être transmis à travers le canal de données, il est tout d'abord placé dans un registre appelé « Transmission Holding Register » (registre d'attente de l'émetteur). Il est ensuite envoyé dans le registre appelé « Transmission Shift Register » (registre de transfert de l'émetteur), à partir duquel il est transféré bit par bit par l'UART à travers le canal de données. Il insère le cas échéant les bits de parité et les bits de stop voulus dans le flux de données.

I. Registre de réception RBR

Lors de la réception des données, celles-ci sont tout d'abord chargées dans le « Receiver Shift Register » (registre de transfert du récepteur), d'où elles sont ensuite envoyées dans le « Receiver Data Register » (Registre de données du récepteur). Les bits de parité et les bits de stop sont éliminés par l'UART à cette occasion. Si un caractère reçu précédemment figurait encore dans le registre de données, et s'il vient d'être effacé par l'opération, alors le bit 1 de l'état de la ligne est fixé à 1. Le bit 0 indique qu'un caractère a été reçu. Si l'UART détecte une erreur de transmission (erreur de parité par exemple) lors du traitement du caractère reçu, il fixe le bit 2 de l'état de la ligne. Si le protocole convenu

(nombre de bits de parité et de bits de stop) n'a pas été respecté, c'est le bit 3 qui est fixé. Le bit 4 est fixé par l'UART chaque fois que la ligne est utilisée plus longtemps que nécessaire pour la transmission d'un caractère. Alors, le bit 7 signale une erreur "time out", c'est-à-dire un dépassement du temps imparti. Cette erreur apparaît parfois lorsqu'on veut transférer des données à travers une ligne téléphonique et que la communication entre la carte RS232 et le modem ne peut être établi correctement.

II.6 FORMAT DES DONNÉES

A chaque extrémité de la ligne de communication, les ordinateurs sont généralement dans ce que l'on appelle un état de repos vis-à-vis de la transmission ou de la réception de caractères. Dans cet état de repos, on transmet une tension continue équivalente à un "1" logique et encore appelée «mark» en anglais. C'est un procédé de sécurité. En effet un niveau bas continu "0" n'est alors possible que dans une configuration où l'ordinateur est inopérant. Un "0" logique encore appelé «space» en anglais, se prolongeant pendant une bonne fraction de seconde ou plus est utilisé comme signal d'arrêt par de nombreux systèmes et correspond à des interruptions ou à des initialisations. La transmission d'un nouveau caractère est toujours signalée par un bit de départ qui est toujours une interruption de la transmission de tension (d'où un «0» logique). Ce bit de départ est suivi de 5 à 8 bits de données, le nombre exact dépendant du codage utilisé pour la transmission. Le séquencement de la transmission est organisé de telle façon que le bit de poids le plus faible (LSB) soit transmis le premier, suivi des bits de poids croissant. Il est possible qu'un bit de parité succède aux bits de données. Ce dernier sert à s'assurer de la validité de la transmission, sa valeur est 0 ou 1 selon la convention de parité utilisée par les ordinateurs, et il est fonction des bits composant la donnée à transmettre. Par exemple, si le signal est transmis avec une convention de parité impaire alors le nombre de bits (de données et de parité) à «1» doit être impair. Par exemple, si l'on transmet le mot 11001011 avec une parité impaire, alors la valeur du bit de parité est 0 car la donnée contient un nombre impair de 1. Inversement si le même mot 11001011 était transmis avec une parité paire, le bit de parité prendrait alors la valeur 1 de façon à rendre pair le nombre de 1 de données plus le bit de parité.

Le bit de parité, s'il existe, est suivi d'un ou de plusieurs bits d'arrêt qui sont toujours des «h. Jusqu'à présent trois options ont été offertes: 1, 1,5 et 2 bits d'arrêt. Cependant, la plupart des UART n'offrent que deux de ces trois options: 1 ou 2 bits d'arrêt car le 1,5 bit (au sens de l'horloge de l'UART) n'est utilisé que dans des configurations de

systèmes Télétype à faible vitesse qui utilisent généralement des méthodes électromécaniques de génération et de réception de signaux. Dans tous les cas cependant, le nombre choisi de bits d'arrêt dépend du protocole de transmission choisi. Une fois cette séquence de bits transmise, le système retourne à l'état de repos jusqu'à ce qu'un prochain caractère soit transmis.

II.7 CONCLUSION

Après avoir donné l'architecture interne et expliqué le fonctionnement de chaque composant de l'UART et les différents registres, Maintenant, nous abordons à expliquer la langage VHDL pour faire la description de chaque composant afin de faire la conception du notre UART par regroupement des différentes descriptions des composants.

Chapitre III

Le langage VHDL pour la synthèse

III.1 INTRODUCTION

Le VHDL est un langage de description de matériel qui est utilisé pour la spécification (description du fonctionnement), la simulation et la preuve formelle d'équivalence de circuits. Ensuite il a aussi été utilisé pour la synthèse automatique. L'abréviation VHDL signifie VHSIC (Very High Speed Integrated Circuit) Hardware Description Language (langage de description de matériel pour circuits à très haute vitesse d'intégration). [6]

III.2 Qu'est ce que le VHDL [16]

III.2.1 Historique

Au début des années 80, le département de la défense américaine (DOD) désire standardiser un langage de description et de documentation des systèmes matériels ainsi qu'un langage logiciel afin d'avoir une indépendance vis-à-vis de leurs fournisseurs. C'est pourquoi, le DOD a décidé de définir un langage de spécification. Il a ainsi mandaté des sociétés pour établir un langage. Parmi les langages proposés, le DOD a retenu le langage VHDL qui fut ensuite normalisé par IEEE. Le langage ADA est très proche, car celui-ci a servit de base pour l'établissement du langage VHDL.

La standardisation du VHDL s'effectuera jusqu'en 1987, époque à laquelle elle sera normalisée par l'IEEE (lnstitute of Electical and Electroniecs Engineers). Cette première normalisation a comme objectif:

- La spécification par la description de circuits et de systèmes.
- La simulation afin de vérifier la fonctionnalité du système.
- La conception afin de tester une fonctionnalité identique mais décrite avec des solutions d'implémentations de différents niveaux d'abstraction.

En 1993, une nouvelle normalisation par l'IEEE du VHDL a permis d'étendre le domaine d'utilisation du VHDL vers:

- La synthèse automatique de circuit à partir des descriptions.
- La vérification des contraintes temporelles.
- La preuve formelle d'équivalence de circuits.

III.2.2 Pourquoi utiliser le VHDL

Le VHDL est un langage normalisé, cela lui assure une pérennité. Il est indépendant d'un fournisseur d'outils. Il devenu un standard reconnu par tous les vendeurs d'outils EDA. Cela permet aux industriels d'investir sur un outil qui n'est pas qu'une mode éphémère, c'est un produit commercialement inévitable. Techniquement, il est incontournable car c'est un langage puissant, moderne et qui permet une excellente lisibilité, une haute modularité et une meilleure productivité des descriptions. Il permet de mettre en œuvre les nouvelles méthodes de conception.

Il est à noter toutefois un inconvénient qui est la complexité du langage.

En effet, ce langage s'adresse à des concepteurs de systèmes électroniques, qui n'ont pas forcément de grandes connaissances en langages de programmation.

III.3 RÈGLES D'ÉCRITURE DU LANGAGE VHDL:[7]

L'écriture d'une description en VHDL doit obéir aux règles suivantes:

A. Les commentaires:

Généralement tout programme VHDL, débute par des informations sur l'auteur, le titre, la date de l'écriture, date de modification: de ce programme. Elles constituent des commentaires.

Tout commentaire débute par un double tiret '--' et se prolonge jusqu'à la fin de la ligne. Si le commentaire doit se poursuivre jusqu'à la deuxième ligne, un double tiret doit être utilisé au début de cette ligne.

B. Majuscules et minuscules:

Le langage VHDL ne distingue pas les majuscules et les minuscules. Pour lui, les écritures, suivantes représentent le même mot: is, IS ou Is. On dit que, le VHDL n'est pas sensible à la «casse». Par exemple, on peut écrire: AND ou bien and pour l'opérateur de la fonction ET logique.

C. La fin d'instruction :

Lorsqu'on termine une instruction VHDL, on utilise le point virgule (;).Il exprime la fin de l'instruction.

Lorsqu'on oublie le point virgule en fin d'instruction, c'est une erreur qu'il faut corriger. Une instruction peut être écrite sur plusieurs lignes comme on peut écrire plusieurs instructions sur une même ligne.

D. Les identificateurs:

Un identificateur est un nom qui désigne un objet pouvant être: une entité, une architecture, un signal, un processus, etc. ... Ce nom est constitué d'une suite de caractères alphabétiques (les 26 lettres de l'alphabet uniquement), de caractères numériques (les 10 chiffres décimaux), ou du caractère '_' souligné: Les caractères ASCII spéciaux sont exclus. Le premier caractère doit être toujours une lettre, le caractère souligné ne doit pas terminer le nom, ni figurer deux fois consécutives. Ce nom ne doit pas être un mot réservé de VHDL et sa longueur ne doit pas dépasser une ligne.

E. Les mots réservés:

On ne peut pas utiliser les noms réservés comme identificateurs. Les noms réservés sont donnés ci-dessous:

ABS ACCESS AFTER ALIAS ALL AND ARCHITECTURE ARRA Y ASSERT A TIRIBUTE. BEGIN BLOCK BODY BUFFER BUS CASE COMPONENT CONFIGURATION CONSTANT DISCONNECT DOWNTO ELSE ELSIF END ENTITY EXIT FILE FOR FUNCTION GENERATE GENERIC GUARDED IF IN INOUT IS LABEL LIBRARY LINKAGE LOOP MAP MOD NAND NEW NEXT NOR NOT NULL OF ON OPEN OR OTHERS OUT PACKAGE PORT PROCEDURE PROCESS RANGE RECORD REGISTER REM REPORT RETURN SELECT SEVERITY SIGNAL SUBTYPE THEN TO TRANSPORT TYPE UNITS UNTIL USE VARIABLE W AIT WHEN WHILE WITH XOR.

F. Les littéraux:

Les littéraux sont des valeurs explicites attribuées à différents objets: constantes, variables, attributs, entrées, sorties, etc ...

Les littéraux se notent différemment selon le type d'objets auxquels ils s'appliquent.

- **les entiers décimaux:** Ils utilisent la notation décimale habituelle: 576 ou 7533, cependant on peut inclure le symbole souligné '_', pour améliorer la lisibilité sans aucune autre conséquence: 1_564 700 est identique à 1564700.

- **les caractères:** Ils s'écrivent avec apostrophes simples: 'A', 'm', ‘ ‘ (espace). Les caractères acceptés sont les caractères ASCII imprimables. Les majuscules et les minuscules sont différenciées lorsqu'il s'agit de valeurs littérales: 'B' est différent de 'b'.

- **les chaînes de caractères:** Elles s'écrivent entre des guillemets comme: "valeur", "1001 ", "compteur". Une chaîne de caractères peut être obtenu par concaténant plusieurs chaînes au moyen de l'opérateur &. "la valeur" &" obtenue" & "est 187" est équivalente à "la valeur obtenue est 187". L'espace est contenu dans la chaîne de caractères.
- **les bits:** Les bits utilisent la notation -des caractères et ont pour seules valeurs '0' et '1'_ Le type de bit généralisé std Jogic utilise en outre les valeurs 'U', 'X', 'H', ’L’, ’W’, ’Z’ et ‘_’.
- **Les vecteurs de bits :** Ils utilisent la notation des chaînes de caractères constituées des symboles 0 et 1 par exemple: "1001011 0" .Par défaut, la base de représentation est binaire. On trouve aussi 2# 1 00 1 #. Mais on peut utiliser la notation octale: 0"734" et 8#734#, ou hexadécimale: X"A 1 OF53'.' et 16#AI0F53#.

G. Les espaces et les sauts de lignes:

Les espaces ne sont pas significatifs sauf dans les valeurs littérales et pour la séparation des identificateurs. Une instruction peut s'étendre sur une ligne ou sur plusieurs lignes. On doit veiller à ce que le saut de ligne ne s'effectue pas au milieu d'un identificateur ou d'un littéral.

H. Les types:

Chaque entrée, sortie, signal, variable ou constante est associée à un type. Les types principaux sont: integer (de -2^{31} à 2^{31}-1), bit, bit_ vector, std _logic, std _logic _ vector, boolean (true, false). Ainsi il n'est pas question d'effectuer une opération logique ou arithmétique entre deux grandeurs de types différents.

On remarque qu'il y a aussi les types énumérations et les sous-types, les types tableaux,…

III.4 LA SYNTAXE DU LANGAGE VHDL

Pour décrire en VHDL un circuit, on aura à spécifier trois parties essentielles qui sont: la librairie, l'entité et l'architecture.

- Une déclaration des librairies, contenants les fonctions et les types pour former le modèle.

- La déclaration de l'entité, c'est l'interface du modèle. Elle peut contenir un ou plusieurs composants, chaque composant étant défini par les mêmes parties que l'entité de conception principale.
- L'architecture de l'entité, elle décrit le comportement de l'entité.

Dans la suite de ce paragraphe, nous allons expliquer les caractéristiques de chaque partie.

III.4.1 Les librairies

Les librairies contiennent les fonctions et les types dont nous avons besoin pour compléter le modèle défini ci-dessous. Ces outils sont constitués en paquets (packages) et on y accède par le mot clé **use** [8].

Toute description VHDL, utilisée pour la synthèse, a besoin de bibliothèques. L'IEEE (Institut of Electrical and Electronics Engineers) les a normalisées et plus particulièrement la bibliothèque IEEE 1164. Elles contiennent les définitions des types de signaux électroniques, des fonctions et des sous- programmes permettant de réaliser des opérations arithmétiques et logiques. La library ieee contient plusieurs paquetages dont: [7]

- std_logic_1164: pour le type std_logic et std_logic_ vector,
- std _logic _ arith: pour les fonctions de conversion de type (entier vers std _logic et inversement),
- std Jogic unsigned: pour les opérations non signées, std _logic _ signed: pour les opérations signées numeric std.

La directive USE permet de sélectionner les bibliothèques à utiliser. On écrit alors pour déclarer les bibliothèques: [7]

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
```

III.4.2 La déclaration d'entité

Une entité doit définir l'interface pour un module. La déclaration d'une entité commence par : **entity <nom_entité>** et se termine bien sûr par le end de la déclaration. Ensuite c'est la déclaration des ports en précisant au cas échéant les ports d'entrées et de sorties de notre module. La syntaxe est donnée ci-dessous [8]:

```
entity NOM_DE_L'ENTITE  is
Port (description des signaux d'entrées / sortie….) ;
end NOM_DE_L'ENTITE;
```

Remarque: avant la fermeture de la parenthèse, il n'y a pas de point virgule.

L'instruction PORT:

La syntaxe: NOM_DU_SIGNAL: SENS TYPE;

Exemple: EN: in std_Iogic;

BUS: out std_logic_ vector (7 downto 0);

Pour chaque signal, on doit préciser: le **nom_du_signal**, le sens et le type. [7]

A. Le nom du signal : [7]

Il doit répondre à la règle des identificateurs. Il est composée de caractères, le premier doit être une lettre et non un chiffre, sa longueur est quelconque, mais ne doit pas dépasser une ligne de code sachant que le VHDL n'est pas sensible à la casse.

B. Le sens du signal (ou le mode): [7]

IN: pour un signal en entrée.

OUT: pour un signal en sortie.

INOUT: pour un signal en entrée sortie.

BUFFER: pour un signal de sortie mais utilisé comme entrée dans la description.

C. Le type: [7]

Le type, utilisé pour les signaux d'entrée/sortie, est:

- std_logic ou bien le type bit, pour un signal,
- std _logic _ vector ou bien le type bit_ vector, pour un bus composé de plusieurs signaux.

Exemple: un bus bidirectionnel de 8 bits s'écrira comme ci-dessous.

```
MOT BUS: inout std_logic_vector (7 downto 0);
```

Où MOT_BUS(7) correspond au MSB et MOT_BUS(O) correspond au LSB. Les valeurs que peut prendre un signal de type std_Iogic sont:

- '0' ou 'L' : pour un niveau bas.
- 'l' ou 'H' : pour un niveau haut.
- 'Z' : pour un état Z de haute impédance (Z en majuscule).
- '_' : quelconque, c'est-à-dire n'importe quelle valeur.
- 'U' : Uninitialized, c'est à dire non initialisé, valeur par défaut déposée au début de la

simulation. .

- 'X' : inconnu.

- 'W' : Weak conflict, c'est-à-dire faible conflit.

U,X,W: ne servent qu'au simulateur.

MOT_BUS: inout std_logic_vector (0 to 7); avec MOT_BUS(O) MSB et MOT_BUS(7) LSB.

III.4.3 Déclaration de l'architecture [7]

Architecture est le mot-clé pour définir le début de la description interne de l'entité. La syntaxe est la suivante:

```
ARCHITECTURE  nom_de_l'architecture OF nom de l'entité IS
        Zone de déclaration (facultative ou optionnelle)
BEGIN
        Description de la structure logique.
END nom_de_l'architecture;
```

L'architecture décrit le fonctionnement de l'entité. Elle établit, à travers les instructions, les relations entre les entrées et les sorties. II est possible de créer plusieurs architectures pour une même entité. Chacune de ces architectures décrira l'entité de façon différente.

L'architecture représente la description interne de la fonction. Cette description n'est pas unique. On dira qu'elle est de nature:

- flot de données: lorsqu'elle décrit la fonction par des équations.
- Comportementale (behavioral) lorsqu'elle décrit la fonction sans référence à la structure ou aux équations.
- Structurelle (structural): lorsqu'elle décrit la fonction à partir de fonction prédéfinies.

III.4.4 Les Descriptions de l'architecture [7]

III.4.4.1 Flots de données:

On décrit le fonctionnement de la fonction ET logique à deux entrées A, B et une sortie S en écrivant l'équation booléenne dans la partie architecture.

```
--Si on définit la sortie d'un ET (A, B, S) par:
S<=A and B;
-- la description est de nature flot de données, elle utilise une équation.
```

Le fonctionnement est défini par des équations booléennes.

III.4.4.2 Comportementale:

La description comportementale, c'est une description algorithmique. Pour le même exemple ET, on peut dire que la sortie du ET est égale à 1 si toutes les deux entrées sont à 1, sinon, dans les autres cas la sortie vaut 0. De même on peut dire que la sortie du ET est égale à l'entrée B quand l'entrée A est égale à l, sinon, on lui affecte la valeur 0. C'est cette dernière description qui est donnée ci-dessous.

```
-- la fonction ET(A,B,S) peut être décrite à travers son comportement
S<=B when (A='l') else '0' ;
-- on a utilisé l'affectation conditionnelle
```

III.4.4.3 Structurelle:

La description Structurelle est une association de structures déjà définies qu'on utilise comme boîtiers ou composants (component en anglais).

Ainsi dans ce type de description, on relie entre eux des composants préalablement décrits en VHDL.

Supposant que la fonction OU_EX2 (entrées: El, E2 et sortie: X) est décrite en VHDL. Pour décrire, la fonction parité à 3 entrées, on utilise la fonction OU _ EX2 déjà réalisé en tant que composant. On aura besoin de deux composants (ouex à 2 entrées). Alors on commence par décrite entité/architectute de OU _EX2 puis on donne l'entité de la fonction parité.

```
Entity ou_ex is
Port (El ,E2: in sdtIogic;
        X : out std _logic); End ou_ex;
Architecture Arch ouex of ou ex 1S Begin
     X <= El XOR E2 ; End Arch_ouex;
--cette entité sera utilisée pour avoir un xor à 3 entrées
--on déclare la nouvelle entité Parity3
Entity Parity3 is
Port ( A, B, C : in std_logic;
        S       : out std logic j..
End Parity3;

-- on déclare la nouvelle architecture selon une description structurelle
Architecture Arch_paire_structural of Parity3 is
--la zone de déclaration sera utilisée pour déclarer un signal
--interne R et le composant utilisé dans la description.
```

```
Signal R: std_logic; Component ou_ex
        Port (EI, E2: in std_logic; X: out std_logic);
End component
 Begin
        Ul: ou ex
            Port map (El =>A, E2=>B, X=>R);
        U2: ou ex
            Port map (R, C, S) ; .
-- on peut écrire (R,C,S) ou bien (El =>R, E2=>C, X=>S)
 End Arch -paire structural;
```

III.4.5 Les instructions

III.4.5.1 Les instructions concurrentes

L'ordre d'écriture n'a alors pas d'importance, les opérations étant réalisées simultanément. Comme montre la figure III.1 [11].

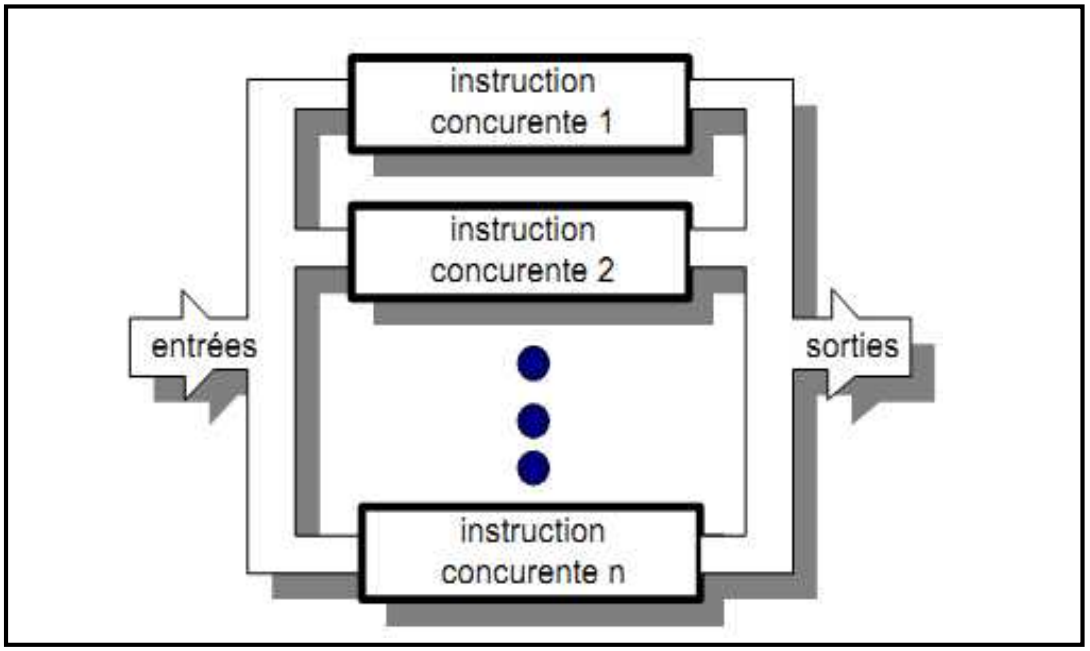


Fig.III.1. Instructions en mode concurrent

A. Affectation simple

Dans une description VHDL, c'est certainement l'opérateur le plus utilisé. En effet il permet de modifier l'état d'un signal en fonction d'autres signaux et/ou d'autres opérateurs.

```
NOM_ D'UNE_GRANDEUR <=V ALEUR _OU _NOM_D'UNE _GRANDEUR;
```

B. L'affectation conditionnelle

```
L'interconnexion est cette fois soumise à une ou plusieurs conditions.
NOM_D'UNE_GRANDEUR <= Q1 when CONDITIONI  else
                      Q2 when CONDITION2 else
                      •••
                      Qn ;
```

On note l'absence de ponctuation à la fin des lignes intermédiaires.

C. Affectation sélective

Cette instruction permet d'affecter différentes valeurs à un signal, selon les vàleurs prises par un signal dit de sélection.

```
With EXPRESSION select
     NOM_D'UNE_GRANDEUR <= Q1 when valeur1,
                           Q2 when valeur2,
                           •••          ,
                           Qn when others ;
```

On note la présence d'une virgule (et non un point virgule) à la fin des lignes intermédiaires.

D. L'instruction concurrente generate

Cette instruction a deux formes:

La forme itérative, exemple:

```
LABEL_BOUCLE: for 1 in 0 to N generate
--suite d'instructions concurrentes
end generate LABEL_BOUCLE; --le label est facultatif ici.
```

La forme conditionnelle, exemple:

```
LABEL_BOUCLE: if CONDITION_BOOLEENNE generate
suite d'instruction concurrentes
end generate LABEL_BOUCLE; -- le label et facultatif ici
```

E. Le processus

Avec les processus, une liste de sensibilité définit les signaux autorisant les actions. Deux syntaxes sont couramment utilisées. Dans la première les signaux à surveiller sont énumérés dans une liste de sensibilité juste après le mot **process** :

```
Process(LISTE _ D E _ SENSIBILITE)
        NOM_DES _OBJETS_INTERNES: « type» ; -- zone facultative
        begin
                INSTRUCTIONS_SEQUENTIELLES;
end process ;
```

Le déclenchement sur un front montant d'un élément de la liste de sensibilité (CLK par exemple) se fait par les instructions suivantes:

```
If (CLK ‘event’and CLK = ‘1’) then
                INSTRUCTIONS;
end if;
```

Avec la seconde syntaxe on utilise l'instruction wait pour énoncer la liste de sensibilité:

```
Process
                NOM_DES_OBJETS_INTERNES : « type» ; -- zone facultative
        begin
                wait on (LISTE_DE_SENSIBILITE) ;
                INSTRUCTIONS_ SEQUENTIELLES;
end process
```

Le déclenchement sur un front montant d'un élément de la liste de sensibilité (CLK par exemple) se fait en replaçant l'instruction wait par la forme suivante:

```
Wait until (CLK=’even’ and CLK=’1’)
```

L'instruction process peut être précédée d'une étiquette afin de faciliter la lecture du programme:

```
LABEL: Process(LISTE _ DE_SENSIBILITE)
                NOM_DES_OBJETS_INTERNES : « type»; -- zone facultative
           Begin
```

```
        INSTRUCTIONS _SEQUENTIELLES;
    end process LABEL;
```

III.4.5.2 Les instructions séquentielles

Un ou plusieurs événements prédéfinis déclenchent l'exécution des instructions dans l'ordre où elles sont écrites. Comme indique la figure III.2 [11].

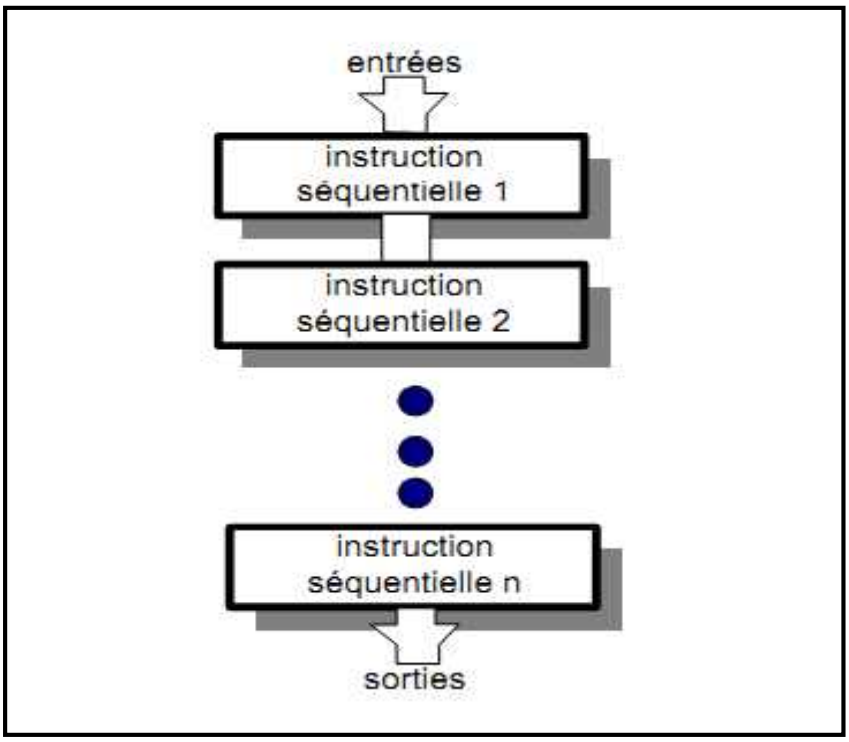


Fig.III.2. Instructions en mode séquentiel

Elles s'utilisent uniquement à l'intérieur d'un « process» et sont examinées dans l'ordre d'écriture.

A. L'affectation séquentielle

Même syntaxe « <= » et même signification que pour une instruction concurrente; pour les variables, on utilise « **:** = ».

B. Le test «if .. then .. elsif .. else .. end if»

La syntaxe de cette instruction est la suivante:

```
If CONDITIONI then
        INSTRUCTION 1 ;
        elsif CONDITION 2 then
        INSTRUCTION 2 ;
```

```
        elsif CONDITION3 then
        INSTRUCTION 3 ;
        •••
        else INSTRUCTIONX;
    end if ;
```

Les conditions sont examinées les unes après les autres dans l'ordre d'écriture; dès que l'une d'elle est vérifiée, on sort de la boucle et les conditions suivantes ne sont pas examinées.

C. Le test « case .. when .. end case»

```
Case EXPRESSION is
        when ETAT l => INSTRUCTION 1;
        when ETAT2 => INSTRUCTION 2;
              •••
        when others => INSTRUCTION n ;
end case ;
```

Ces instructions sont particulièrement adaptées à la description des machines d'états.

III.4.6 Déclaration de composant [8]

Nous aurons à les utiliser assez fréquemment puisqu'elle est une clé pour la description structurale (la dernière partie) de nos modèles. Les gammes de syntaxe marcheront si cette déclaration correspond à son usage, mais cela synthétisera uniquement si la déclaration correspond à un composant existant. Très souvent, il est possible de simplement recopier la déclaration de l'entité du composant et de changer le mot clé « entity » par « component ».

En voici donc la syntaxe de la déclaration d'un composant:

```
        component NOM_COMPONENT is
        Port (les déclarations de ports ayant aussi été faites dans la déclaration « entity » ;
        End component
```

Exemple :

```
component or_entit is
        port (input_l : in
std Jogic ;
        input_2: in std
```

```
_logic ;
        output:  out  std
Jogic) ;
        end component;
```

III.4.7 Instanciation de composant [8]

L'instanciation de composant est la clé des caractéristiques du code structural.

Voici sa syntaxe :

```
<identificateur_composant> : <nom_composant>
port map(
        <nom _port> => <signal_assigne>, ...) ;
```

Exemple:

```
or_ ent_l : or_entity
port map(
input_l => input_l_ sig ;
input_2 =>input_2_sig ;
output => output_sig);
```

Dans la partie suivante, nous présentons les différents types des opérateurs que l'on rencontre dans le VHDL.

III.4.8 Les opérateurs de base

Le langage VHDL comporte six classes d'opérateurs avec un niveau de priorité défini pour chaque classe. Lors de l'évaluation d'une expression, l'opération dont la classe a la plus haute priorité est effectuée en premier. Etudions, l'une après l'autre, chacune de ces six classes en les donnant par ordre de priorité croissante, les opérations logiques ayant donc le niveau de priorité le plus faible [9].

A. Les opérations logiques

Ce sont les opérations and, or, nand, nor, xor, soit les fonctions logiques: ET, OU, NON¬ET, NON-OU, OU exclusif. Les cinq opérateurs logiques ont la même priorité, d'où l'intérêt et quelquefois la nécessité de parenthèses. Ainsi, l'opération

A or B and C, est impossible à interpréter sans parenthèses.

B. Les opérations relationnelles

Ce sont les relations qui expriment la relation entre deux grandeurs: égal, inégal, plus petit, plus petit ou égal, plus grand, plus grand ou égal (=, /=, <, <=, », >=). Le résultat de la comparaison est de type booléen, la relation concernée ne pouvant qu'être vrai ou fausse. Pour rendre l'écriture plus lisible, il est utile et conseillé de mettre entre parenthèses l'opération relationnelle.

C. Les opérations d'addition

Elles sont au nombre de trois: l'addition, la soustraction et la concaténation de symboles respectifs : +, -, et &. La concaténation est définie pour des chaînes de bits et des chaînes de caractères. C'est une simple juxtaposition des valeurs.

D. Les opérations de signe

Ce sont les signes" + " ou "-". Ces opérations sont définies, c'est-à-dire valides, pour des objets de type entier ou flottant.

E. Les opérations de multiplication

Ce sont les opérations suivantes:

- La multiplication, de symbole "*".
- La division symbole ‘’ / ‘’.
- Le modulo, de symbole ‘’mod’’.
- Le calcul de reste, (remainder) de symbole Il ‘’rem’’. Elles sont définies sur les types entier et flottant.

**F. Les opérations NOT, ABS et **

Ce sont les trois opérations de plus haute priorité. L'opération NOT prend le complément de la valeur d'un objet de type bit, booléen, bit_vector et également d’une chaîne d’éléments de type booléen.

Exemples

C < = not A ; __ C = complément de A

C < = not (A or B) ; __ C = complément (A+B), la parenthèse est nécessaire ici.

L'opération ABS rend la valeur absolue. Elle est définie sur les entiers et les flottants.

L'opération ** est l'exponentiation, c'est-à-dire l'élévation de l'opération de gauche à la

puissance définie par l'opérande de droite. Ce dernier doit être un nombre entier tandis que l'opérande de gauche doit être de type entier ou flottant. Exemple:
A ** B est égal à A^B.

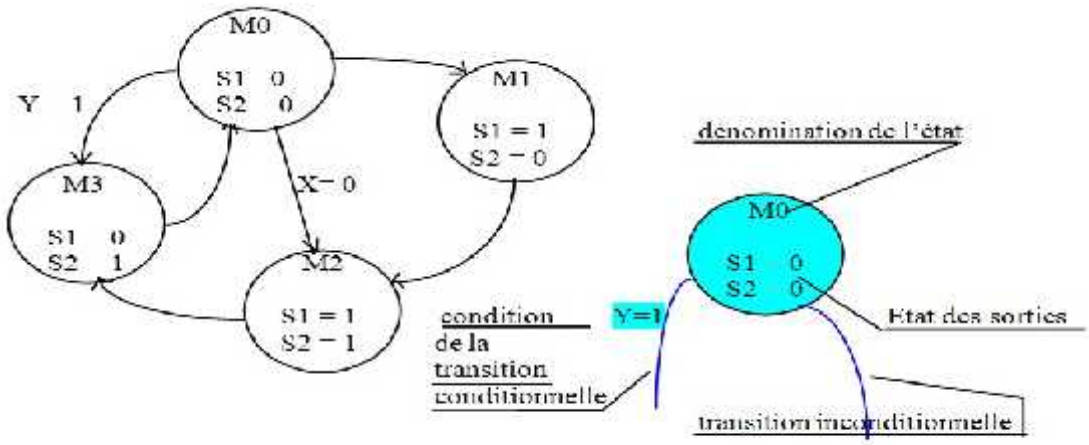
Après avoir présenté la syntaxe des déclarations, un paquetage permet de grouper ces déclarations et de les stocker dans une bibliothèque.

III.5 MACHINE D'ÉTAT

Utilisée pour décrire des systèmes séquentiels quelconques (state machine).

III.5.1 Principe

La description du système se fait par un nombre fini d'états. Ci-dessous la représentation schématique d'un système à 4 états (M0 à M3), 2 sorties (S1 et S2), 2 entrées X et Y, sans oublier l'entrée d'horloge qui fait avancer le processus, et celle de remise à zéro qui permet de l'initialiser



L'état initial est M0. Les 2 sorties sont à 0. Au coup d'horloge on passe inconditionnellement à l'état M1 sauf si la condition Y=1 a été vérifiée, ce qui mène à l'état M3 ou si X=0 a été validé ce qui mène à M2.

De M3 on revient au coup d'horloge à M0. De M1 on passe à M2, et de M2 on passe à M 3...

Dans chaque état on définit les niveaux des sorties.

Il existe deux famille de machines d'état : machine de Moore et machine de Mealy qu'on va expliquer ci-après.

III.5.2 Machine de Mealy

Voici la machine de Mealy répondant sur figure III.3

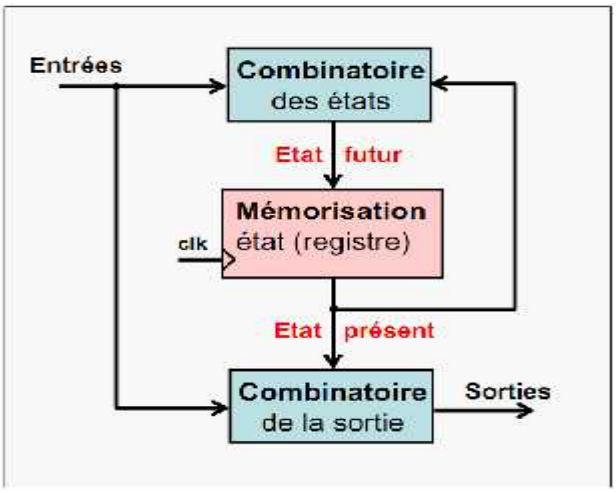


Fig.III.3. Machine de Mealy

- L'état futur est calculé à partir des entrées et de l'état présent.
- Les sorties d'une machine de Mealy dépendent de l'état présent et des entrées.
- Mémorisation synchrone des états (càd sur un front d'horloge).
- La sortie dépend directement de l'entrée et ceci indépendamment de l'horloge (clk). => Sortie asynchrone.
- Nombre d'états plus réduit que pour une machine de Moore.
- Il est possible de resynchroniser la sortie au besoin en ajoutant des bascules D.

Exemple : Machine de Mealy reconnaissant la séquence 10 donne le graphe d'état de la figure III.4, et leur diagramme dans la figure III.5

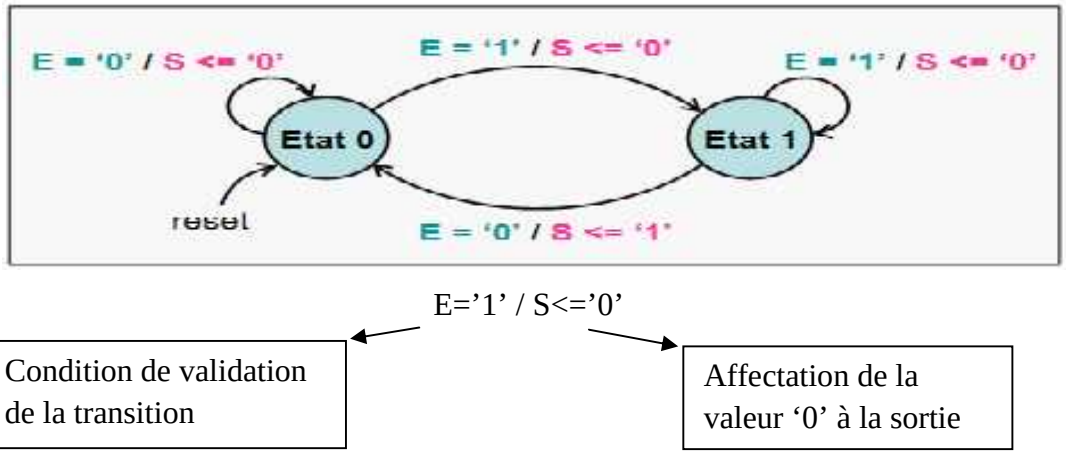
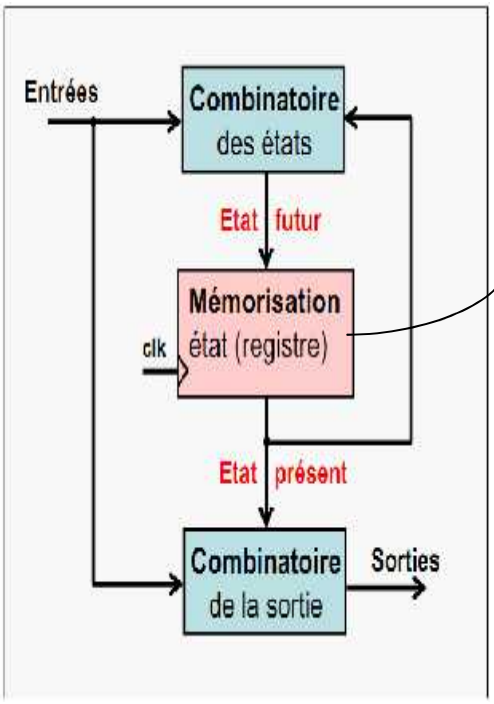


Fig.III.4. le graphe d'état de la Machine de Mealy reconnaissant la séquence 10

Fig.III.5. Diagramme machine de Mealy reconnaissant la séquence 10

La description VHDL correspondante Description avec **3 process**

- Un process séquentiel de mise à jour de l'état présent par l'état futur sur les fronts montant d'horloge (reset asynchrone inclus)

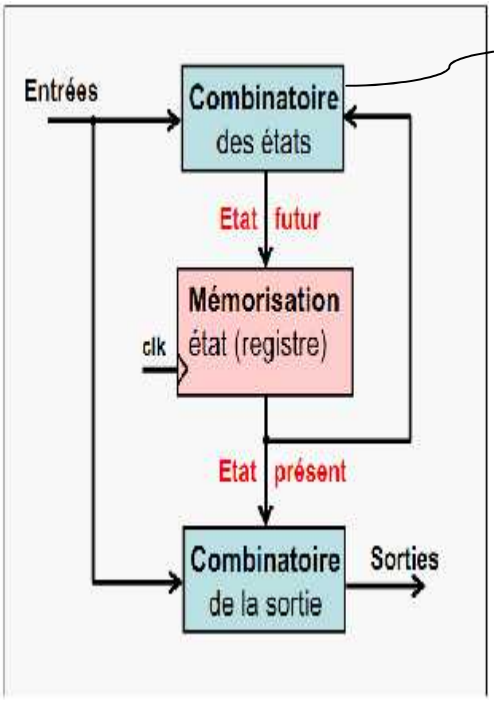



```
type Etat is (Etat0, Etat1);

Signal Etat_present, Etat_futur : Etat := Etat0;
```

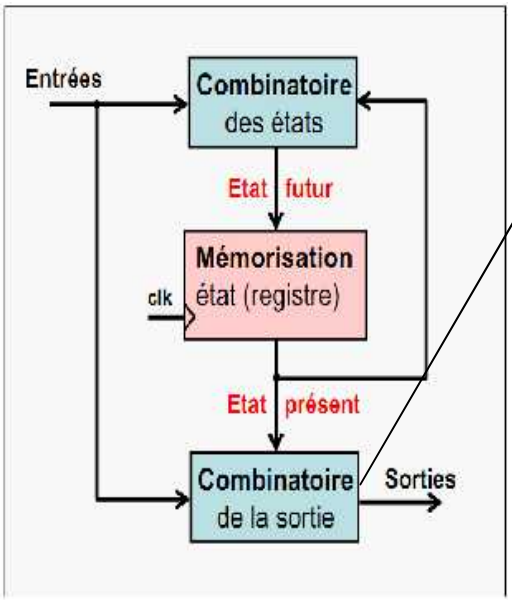
```
Sequentiel_maj_etat : process (clk, reset)
begin
        if reset = '0' then
                Etat_present <= Etat0;
        elsif clk'event and clk = '1' then
                Etat_present <= Etat_futur;
        end if;
end process Sequentiel_maj_etat;
```

- Un process combinatoire de calcul de l'état futur à partir des entrées et de l'état présent :




```
Combinatoire_etats : process (E, Etat_present)
begin
      case Etat_present is
          when Etat0 => if E = '1' then
                        Etat_futur <= Etat1;
                        else
                        Etat_futur <= Etat0;
                        end if;
          when Etat1 => if E = '1' then
                        Etat_futur <= Etat1;
                        else
                        Etat_futur <= Etat0;
                        end if;
      end case;
end process Combinatoire_etats;
```

- Un process combinatoire de calcul des sorties à partir des entrées et de l'état présent :




```
Combinatoire_sorties : process (E, Etat_presen
begin
      case Etat_present is
          when Etat0 => if E = '1' then
                        S <= '0';
          else
                        S <= '0';
          end if;
          when Etat1 => if E = '0' then
                        S <= '1';
          else
                        S <= '0';
          end if;
      end case;
end process Combinatoire_sorties;
```

III.5.3 Machine de Moore

- Les sorties d'une machine de Moore dépendent de l'état présent (synchrones, elles changent sur un front d'horloge).
- L'état futur est calculé à partir des entrées et de l'état présent.

La figure III.5 Représente la machine de Moore

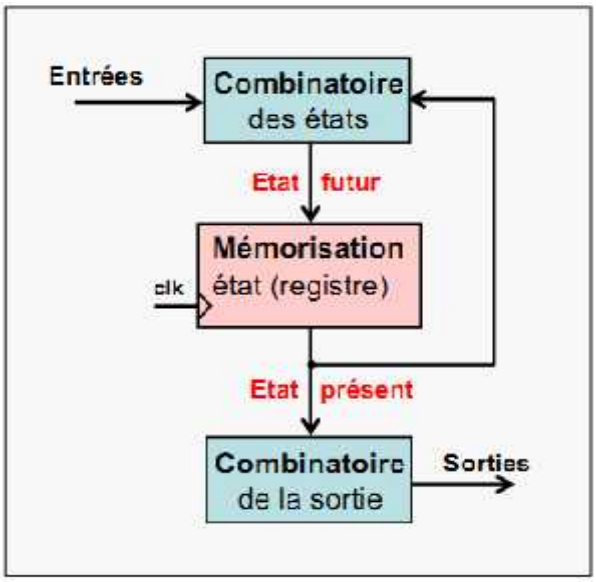


Fig.III.6. Machin de Moore

Exemple : Machine de Moore reconnaissant la séquence 10 donne le graphe d'état de la figure III.7, et leur diagramme dans la figure III.8

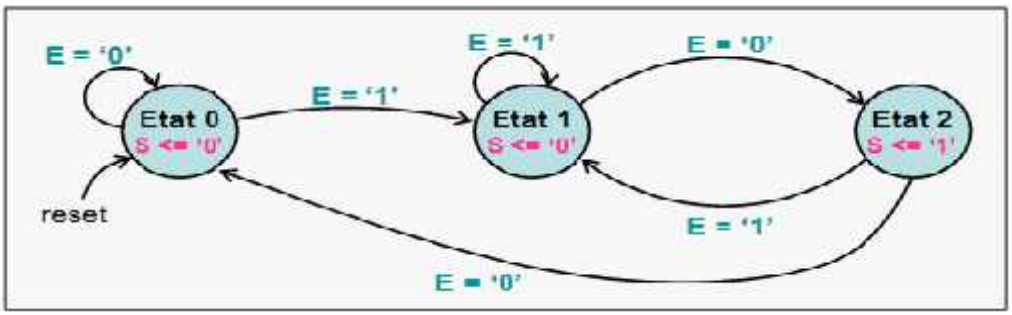


Fig.III.7. Le graphe d'état du détecteur de la séquence 10 selon Moore

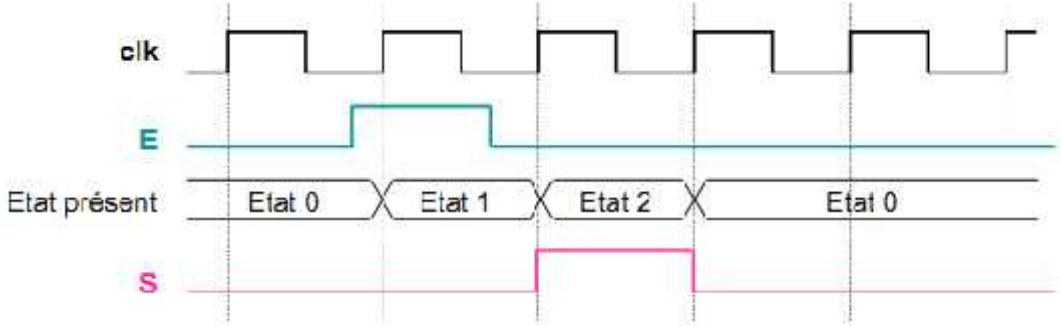
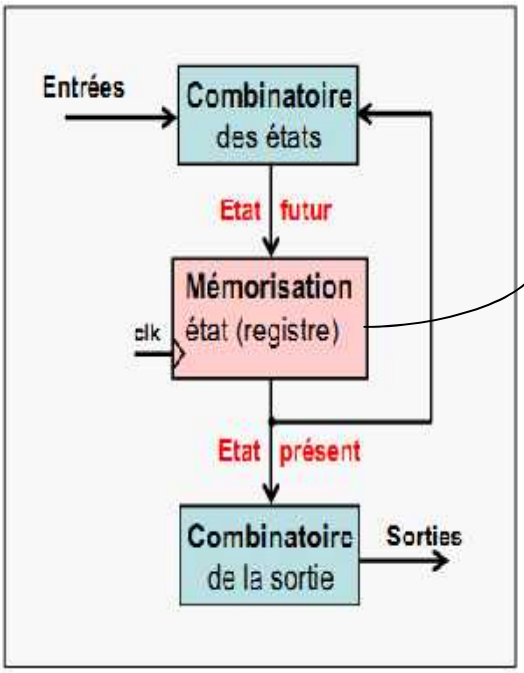


Fig.III.8. Diagramme machine de Moore

- Un process séquentiel de mise à jour de l'état présent par l'état futur sur les fronts montant d'horloge (reset asynchrone inclus) :

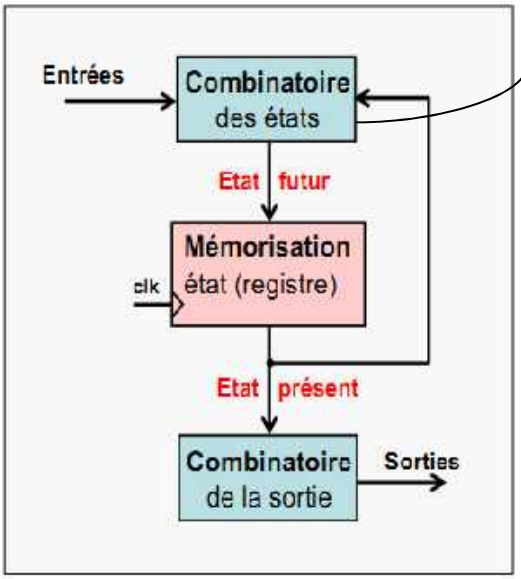



```
type Etat is (Etat0, Etat1, Etat2);

Signal Etat_present, Etat_futur : Etat := Etat0;
```

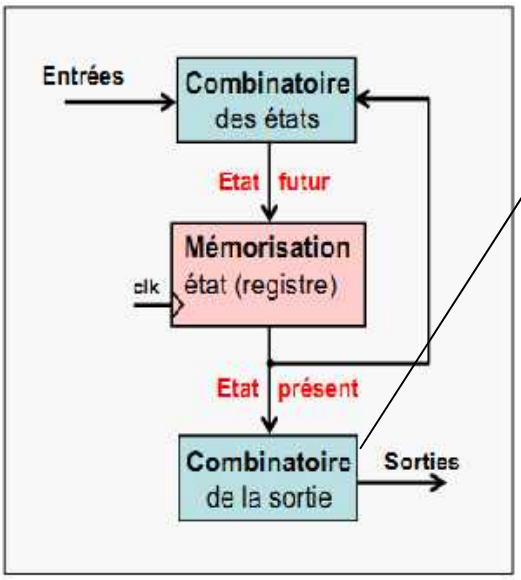
```
Sequentiel_maj_etat : process (clk, rese)
begin
        if reset = '0' then
                Etat_present <= Etat0;
        elsif clk'event and clk = '1' then
                Etat_present <= Etat_futur;
        end if;
end process Sequentiel_maj_etat;
```

- Un process combinatoire de calcul de l'état futur à partir des entrées et de l'état présent :




```
Combinatoire_etats : process (E, Etat_present)
begin
      case Etat_present is
         when Etat0 => if E = '1' then
                        Etat_futur <= Etat1;
         else
                        Etat_futur <= Etat0;
         end if;
         when Etat1 => if E = '0' then
                        Etat_futur <= Etat2;
         else
                        Etat_futur <= Etat1;
         end if;
         when Etat2 => if E = '1' then
                        Etat_futur <= Etat1;
         else
                        Etat_futur <= Etat0;
         end if;
      end case;
end process Combinatoire_etats;
```

- Un process combinatoire de calcul des sorties à partir de l'état présent :




```
Combinatoire_sorties : process (Etat_preseent)
begin
      case Etat_present is
          when Etat0 => S <= '0';
          when Etat1 => S <= '0';
          when Etat2 => S <= '1';
      end case;
end process Combinatoire_sorties;
```

III.6 CONCLUSION

Avec l'arrivée des nouveaux outils de description, tel que le langage VHDL qui offre de nombreux avantages pour la conception des circuits et des systèmes, cette matière redevient en soi un domaine qui conduit à l'étude et au développement des systèmes numériques modernes. C'est ce langage qui sera utilisé pour d'écrire le circuit de l'UART qui fait l'objet de ce travail.

Chapitre IV

La description VHDL et la simulation

IV.1 INTRODUCTION

Cette partie portera sur la description en VHDL d'un UART avec chaque composant à partir du générateur de Baud avec leur description VHDL et la simulation avec quelques exemples de vitesse, ensuite la description VHDL du deuxième composant l'émetteur/récepteur et la simulation, enfin le circuit générale.

IV.2 DESCRIPTION DES COMPOSANTS

On va détailler bien l'architecture interne de l'UART, et dans cette partie on va le décrire en VHDL un UART avec 3 paries fondamentales qui sont : (figure IV.1)

- Générateur de Baud
- Un émetteur « TRANSMITTER »
- Un récepteur « RECEIVER »

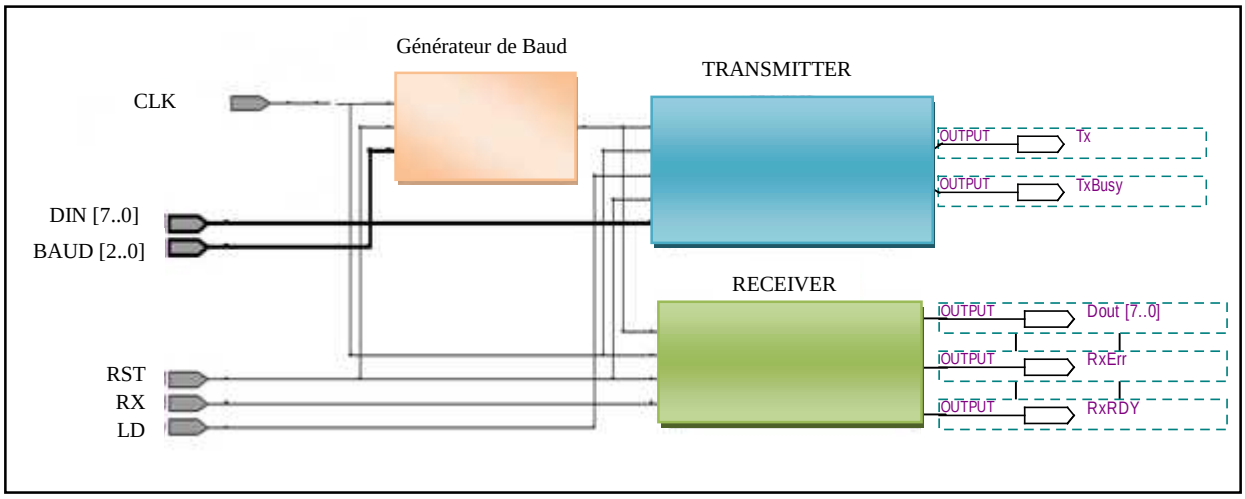


Fig.IV.1. Les différents blocs d'UART

Chaque bloc sera détaillé par la description VHDL suivi de la simulation qui permet d'avoir les différentes valeurs en sortie.

IV.2.1 Générateur de Baud

C'est le circuit qui détermine la vitesse de transmission. On a considéré la fréquence CLK du système égale à 14.7456 MHz. Cette dernière est divisée pour avoir la fréquence de transmission TopTx, cette fréquence n'est autre que la vitesse de transmission en Baud.

On va utiliser une succession de division de la fréquence. On commence par définir le nombre X par lequel on doit diviser la fréquence CLK. Ce nombre X est sélectionné par l'entrée « Baud» à 3 bits en fonction de la vitesse V_{UART} à choisir. Il est donner par la relation ($14.7456 * 10^6/16 * X) = V_{UART}$.

Dans notre cas on a 8 vitesses donc 8 valeurs de X (8, 16, 24, 48, 96, 192, 384, 768) qui sont données dans les relations ci-dessous :

$14.7456 * 10^6/16 * 8=115200$
$14.7456 * 10^6/16 * 16=57600$
$14.7456 * 10^6/16 * 24=38400$
$14.7456 * 10^6/16 * 48=19200$
$14.7456 * 10^6/16 * 96=9600$
$14.7456 * 10^6/16 * 192=4800$
$14.7456 * 10^6/16 * 384=2400$
$14.7456 * 10^6/16 * 768=1200$

D'après la division de « Top16 » ce qui nous donnera deux vitesses la 1ère pour l'émission « TopTx » et la 2ème pour la réception « TopRx », tel qu'une impulsion pour TopTx est égale 16 impulsions de Top16, et une impulsion pour TopRx est égale 8 impulsion de Top16.

Le tableau IV.1 explique comment commander et sélectionner les vitesses à travers 3 bits de sélections.

N°	Bits de sélections (3bits)	Compteur mod-X	La vitesse sélectionnée
1	000	8	115200
2	001	16	57600
3	010	24	38400
4	011	48	19200
5	100	96	9600
6	101	192	4800
7	110	384	2400
8	111	768	1200

Tableau IV.1 : Les déférentes vitesses et les bits de sélection

IV.2.1.1 La description VHDL d'un générateur de Baud

On va procéder à la description du fonctionnement de ce circuit en langage VHDL par une entité « genClock » et une architecture « arch_genClock »

L'entité est composée avec les entrées et les sorties suivantes :

- Les entrées : Baud, CLK, RST.
- Les sorties : TopTx, TopRx.

Dans l'architecture, on a :

- 6 signaux internes : Divisor, Div16, ClkDiv, RxDiv, Top16, clrDiv
- 4 **process** pour générer les signaux internes et les signaux de sortie.

Cette architecture est résumée par le schéma de la figure IV.2.

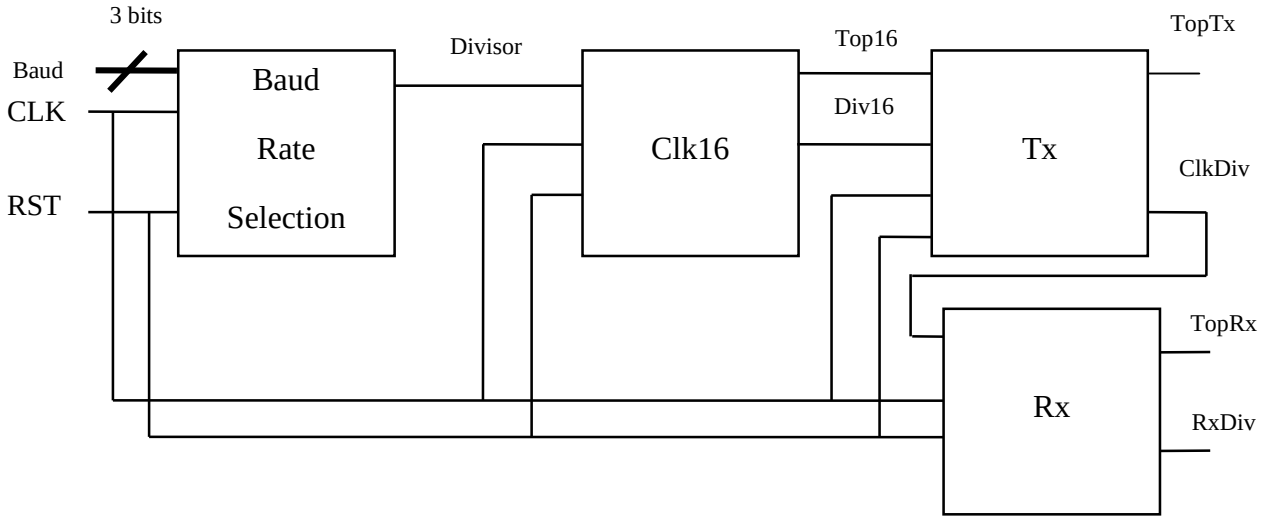


Fig.IV.2. Schéma de l'architecture du générateur de Baud

La description VHDL est donnée par le programme suivant :

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
-- ----------------------------------------------------
Architecture arch_genClock of genClock is
begin
process (RST, CLK)
begin
        if RST='1' then
                Divisor <= 0;
        elsif rising_edge(CLK) then
                case Baud is
                        when "000" => Divisor <= 7; -- 115.200
                        when "001" => Divisor <= 15; -- 57.600
                        when "010" => Divisor <=23; -- 38.400
                        when "011" => Divisor <= 47; -- 19.200
                        when "100" => Divisor <= 95; -- 9.600
```

```
                        when "101" => Divisor <= 191; -- 4.800
                        when "110" => Divisor <= 383; -- 2.400
                        when "111" => Divisor <= 767; -- 1.200
                        when others => Divisor <= 7; -- n.u.
                end case;
        end if;
end process;
process (RST, CLK)
begin
        if RST='1' then
                Top16 <= '0';
                Div16 <= 0;
        elsif rising_edge(CLK) then
                        Div16 <= Div16 + 1;
                end if;
        end if;
end process;
process (RST, CLK)
begin
        if RST='1' then
                TopTx <= '0';
                ClkDiv <= 0;
        elsif rising_edge(CLK) then
                TopTx <= '0';
                if Top16='1' then
                        ClkDiv <= ClkDiv + 1;
                        if ClkDiv = 15 then
                                TopTx <= '1';
                        end if;
                end if;
        end if;
end process;

process (RST, CLK)
```

```
begin
        if RST='1' then
                TopRx <= '0';
                RxDiv <= 0;
        elsif rising_edge(CLK) then
                TopRx <= '0';
                if ClrDiv='1' then
                        RxDiv <= 0;
                elsif Top16='1' then
                        if RxDiv = 7 then
                                RxDiv <= 0;
                                TopRx <= '1';
                        else
                                RxDiv <= RxDiv + 1;
                        end if;
                end if;
        end if;
end process;
end arch_genClock;
```

IV.2.1.2 La compilation

Après la création du projet avec l'entité «genClock » et création du fichier « genClock.vhdl » on procède à la saisie de la description VHDL du circuit. Après cela on procède à la compilation qui réalise 4 étapes suivantes :

- L'analyse et la synthèse logique (Analysis and synthesis) ;
- La synthèse physique (Fitter) ;
- L'assemblage (Assembler) ;
- L'analyse temporelle (Classic Timing Analyzer).

En fin de compilation on a reçu un message qui confirme le succès de l'opération à 100% comme montre la figure IV.3.

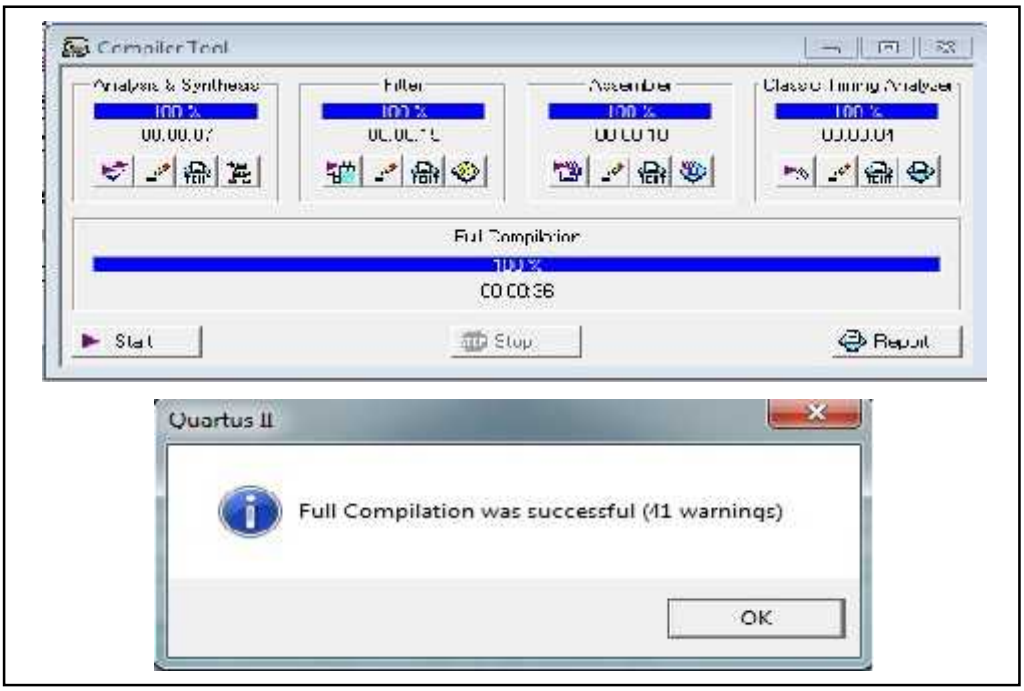


Fig.IV.3. Fenêtre de compilation projet « genClock »

Le succès de la compilation permet de savoir qu'il n'y a pas d'erreurs dans la description VHDL mais le fonctionnement doit être certifié par la simulation pour conformer le fonctionnement exact de circuit.

IV.2.1.3 La vue RTL (**R**egister **T**ransfer **L**evel)

Tool → Netlist Viewer → RTL Viewer

L'utilisation de cette commande permet la meilleure visibilité des connections de nous pouvons également voir comme montre la figure IV.4.

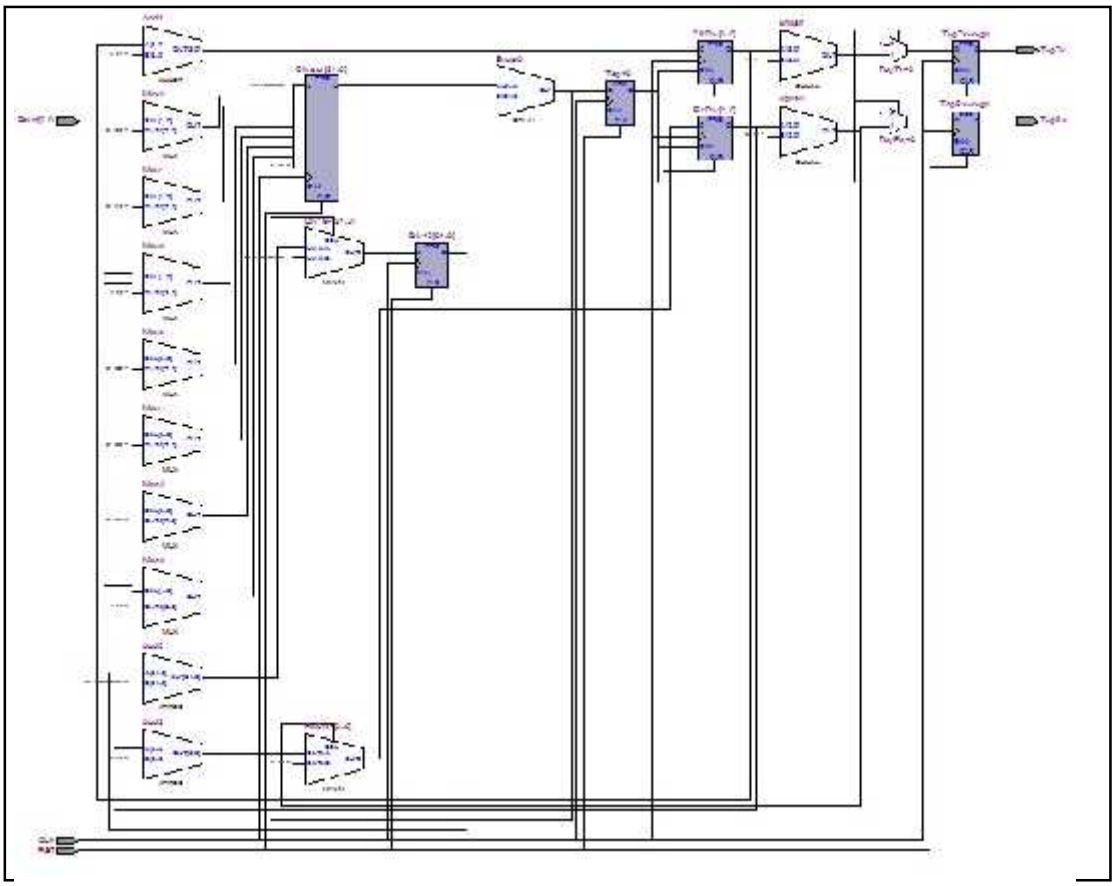


Fig.IV.4. Vue RTL du générateur de baud

IV.2.1.4 La simulation

La simulation c'est l'étape la plus importante pour voir le fonctionnement exact du générateur de Baud, et dans cette simulation nous allons prendre certaines vitesses à partir tableau IV.1 par exemple la 1ère, la 4ème, et la 7ème, comme représente le tableau IV.2.

Bits de sélections (3bits)	La vitesse sélectionnée
000	115200
011	19200
110	2400

Tableau IV.2 : Les vitesses choisies pour la simulation

Et pour les trois vitesses, le signal RST nous avons seulement à 1 au début parce que le signal pour retourner à zéro, et le signal CLK comme nous l'a mentionné 14.7456 Mhz, et juste échange les trois bit de sélection dans chaque cas.

- Pour la vitesse 115200 Baud les 3 bits de sélections doit être « 000 » dans entrées « Baud », nous avons obtenir les trois sorties comme la figure IV.5 montre.

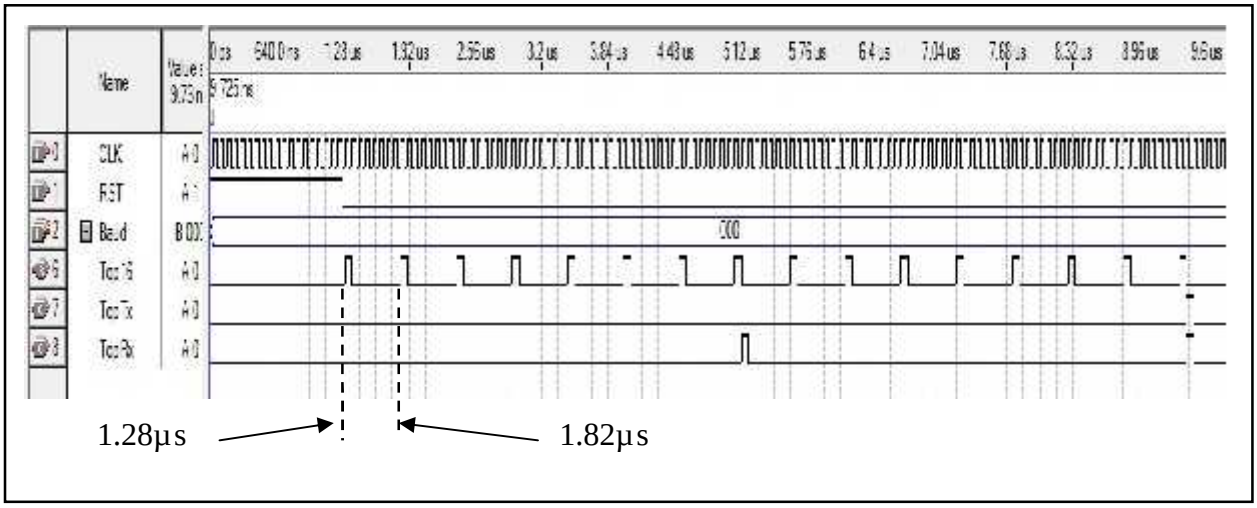


Fig.IV.5. La vitesse 115200 Baud

D'après la figure IV.5 La période de la vitesse 115200 est 1.82-1.28=0.54 µs.

Nous calculons la période de cette vitesse :

La période = 1/ (115200*16) = 0.543 * 10^{-6} donc 0.543 µs.

- Pour la vitesse 19200 Baud les 3 bits de sélections sont « 011 » dans entrées « Baud », nous avons obtenir les trois sorties comme la figure IV.6 montre.

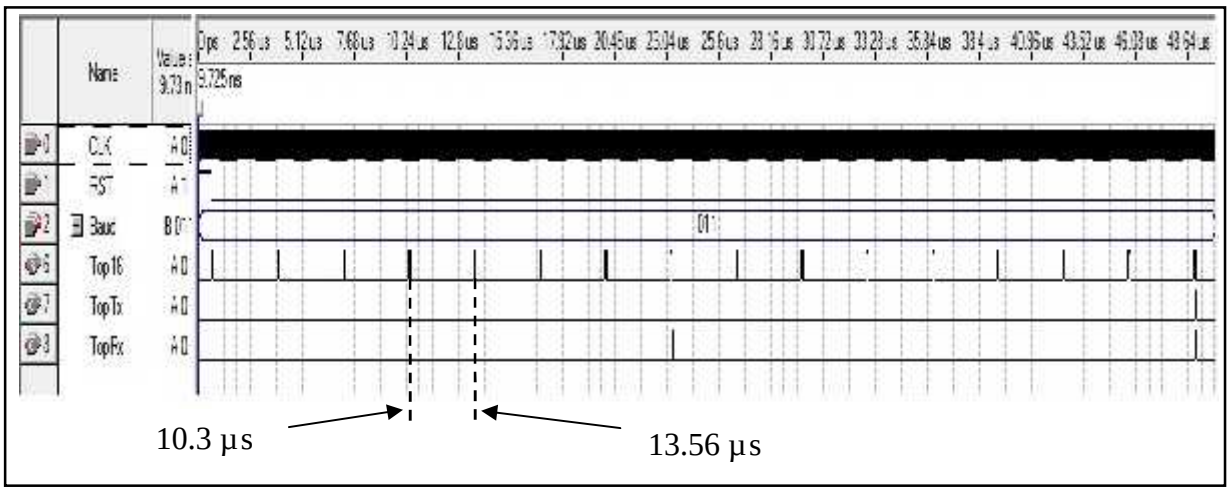


Fig.IV.6. La vitesse 19200 Baud

D'après la figure IV.6 La période de la vitesse 19200 est 13.56 -10.3=3.26 µs.

Nous calculons la période de cette vitesse :

La période = 1/ (19200*16) = 3.26*10^{-6} donc 3.26 µs.

- Et le dernier exemple pour la vitesse 2400 Baud les 3 bits de sélection, sont « 110 » dans entrées « Baud », nous allons obtenir les trois sorties comme la figure IV.7 montre.

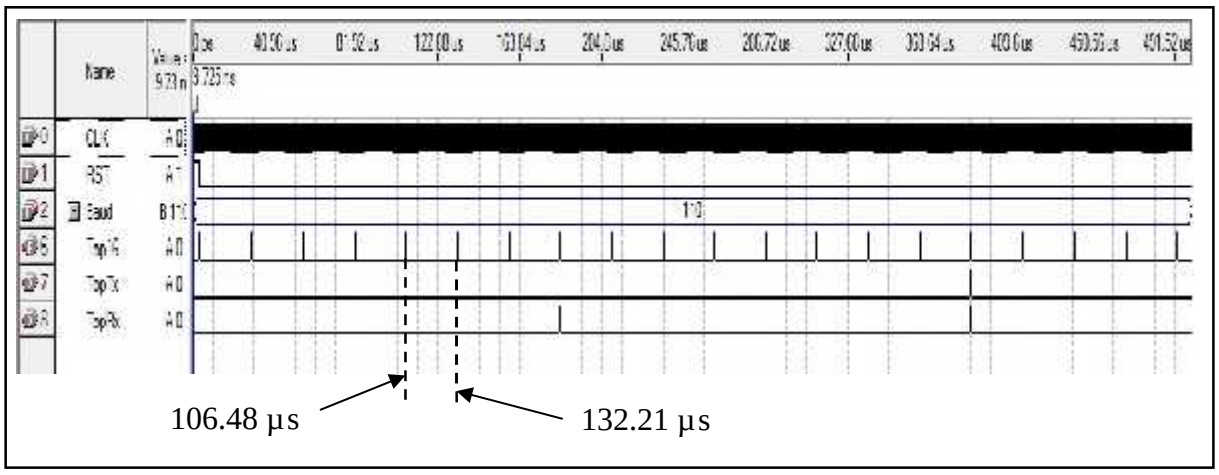


Fig.IV.7. La vitesse 2400 Baud

D'après la figure IV.7 La période de la vitesse 2400 est

132.21-106.48=25.73 µs.

Nous calculons la période de cette vitesse :

La période = 1/ (2400*16) = 26.04 µs

IV.2.2 L'émetteur « TRANSMITTER »

C'est le circuit qui va faire la transformation parallèle/série et donc le décalage des données à la vitesse en baud.

A la valeur du registre donnée il va ajouter le bit Start et le bit parité et le bit de Stop.

C'est un circuit séquentiel asynchrone à machine d'état qui peut être représenté par un graphe d'état comme montre la figure IV.8.

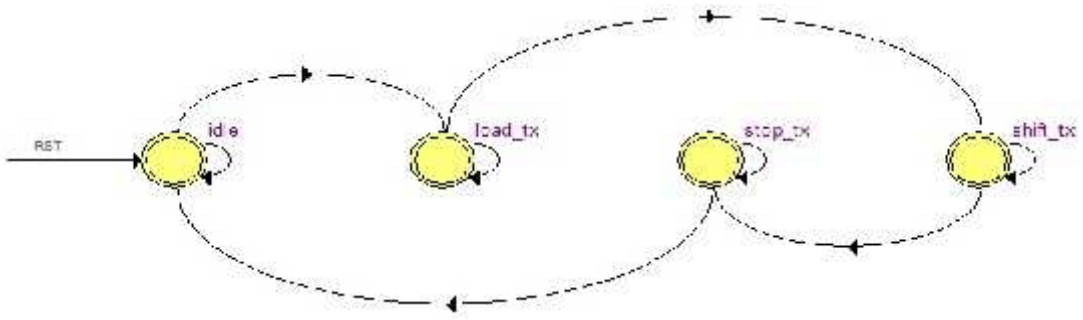


Fig.IV.8. La graphe d'état de l'émetteur

On va procéder à la description du fonctionnement de ce circuit en langage VHDL par une entité « transmitter » et une architecture « arch_transmitter ».

L'entité est composée avec des entrées et des sorties comme suit :

- Les entrées : DIN, LD, Baud, CLK, RST ;
- Les sorties : Tx, TxBusy

Et l'architecture est composée avec des signaux internes, et des états de transmission suivants :

- Les signaux : Tx_Reg ; TxBitCnt ; TxFSM ; RegDin ; parity ; NDBits ; Divisor ; Top16 ; Div16 ; TopTx ; ClkDiv.
- Les états : Idle, Stop_Tx, Shift_Tx, Load_Tx.

La figure IV.9 représente le schéma interne d'un émetteur

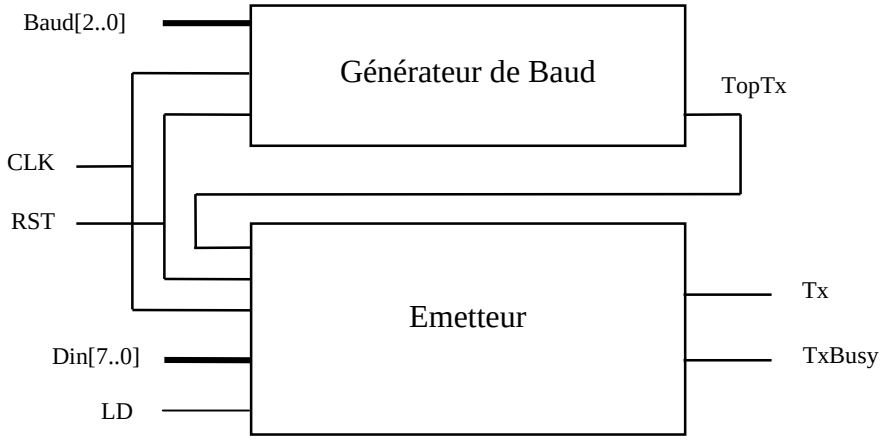


Fig.IV.9.Schéma interne d'un émetteur

IV.2.2.1 La description VHDL de l'émetteur

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
-- ----------------------------------------------------
Architecture arch_transmitter of transmitter is
begin
process (RST, CLK)
begin
   if RST='1' then
         Divisor <= 0;
   elsif rising_edge(CLK) then
         case Baud is
            when "000" => Divisor <= 7; -- 115.200
            when "001" => Divisor <= 15; -- 57.600
            when "010" => Divisor <= 23; -- 38.400
            when "011" => Divisor <= 47; -- 19.200
            when "100" => Divisor <= 95; -- 9.600
            when "101" => Divisor <= 191; -- 4.800
            when "110" => Divisor <= 383; -- 2.400
```

```
            when "111" => Divisor <= 767; -- 1.200
            when others => Divisor <= 7; -- n.u.
        end case;
    end if;
end
    elsif rising_edge(CLK) then
        Top16 <= '0';
          if Div16 = Divisor then
            Div16 <= 0;
            Top16 <= '1';
        else
            Div16 <= Div16 + 1;
        end
    if RST='1' then
        TopTx <= '0';
        ClkDiv <= 0;
    elsif rising_edge(CLK) then
        TopTx <= '0';
          if Top16='1' then
              ClkDiv <= ClkDiv + 1;
              if ClkDiv = 15 then
                 TopTx <= '1';
              end if;
          end if;
    end if;
end process;
TX <= Tx_Reg(0);
Tx_FSM: process (RST, CLK)
begin
   if RST='1' then
        Tx_Reg <= (others => '1');
        TxBitCnt <= 0;
        TxFSM <= idle;
        TxBusy <= '0';
```

```
        RegDin <= (others=>'0');
    elsif rising_edge(CLK) then
        TxBusy <= '1'; -- except when explicitly '0'
        case TxFSM is
        when Idle =>
            if LD='1' then
                -- latch the input data immediately.
                RegDin <= Din;
                TxBusy <= '1';
                TxFSM <= Load_Tx;
            else
                TxBusy <= '0';
            end if;
        when Load_Tx =>
            if TopTx='1' then
                TxFSM <= Shift_Tx;
                if parity then
                  -- start + data + parity




        TxBitCnt <= (NDBits + 2);
                  Tx_Reg <= '1'& Din & '0';
              else
                  TxBitCnt <= (NDBits + 1); -- start + data
                  Tx_reg <= '1' & RegDin & '0';
              end if;
            end if;
        when Shift_Tx =
                    TxFSM <= Stop_Tx;
                end if;
            end if;
        when Stop_Tx =>
```

```
            if TopTx='1' then
                TxFSM <= Idle;
            end if;
        when others =>
            TxFSM <= Idle;
        end case;
    end if;
end process;
end arch_transmitter;
```

Apres la description VHDL de l'émetteur, maintenant nous pouvons obtenir le schéma de l'émetteur et leur entries et les sorties comme montre dans la figure IV.10.

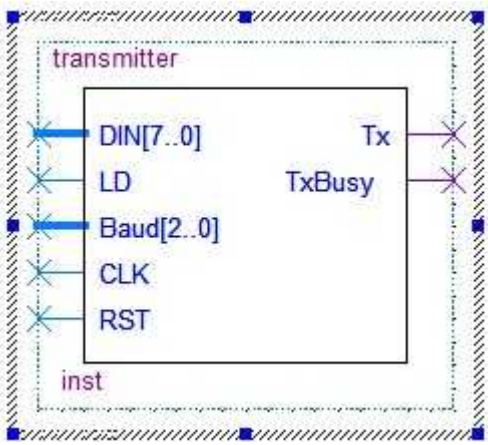


Fig.IV.10. L'émetteur d'un UART

Et leur description VHDL donne ci après avec les deux descriptions VHDL de générateur de baud et le récepteur « UART complet ».

IV.2.3 Le récepteur « RECEIVER »

C'est le circuit qui va faire la transformation série/ parallèle.

C'est un circuit séquentiel asynchrone à machine d'état qui peut être représenté par un graphe d'état comme montre la figure IV.11.

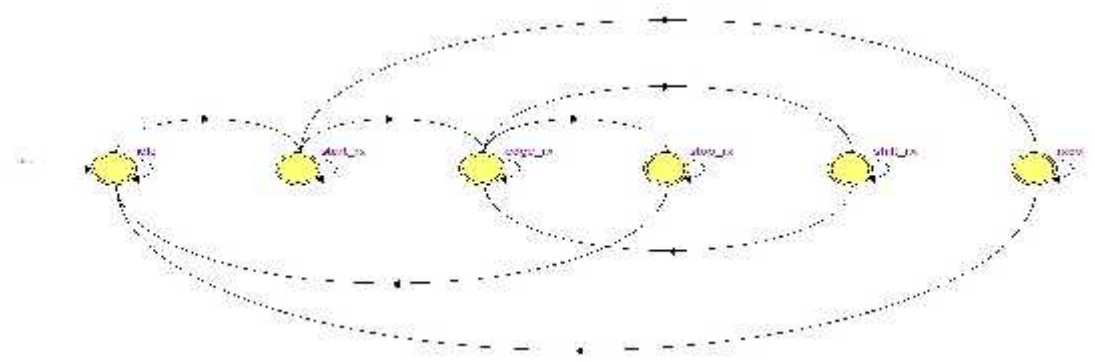


Fig.IV.11. La graphe d'état du récepteur

On va procéder à la description du fonctionnement de ce circuit en langage VHDL par une entité « recepteur » et une architecture « arch_recepteur »

L'entité est composée avec des entrées et des sorties comme suit :

- Les entrées : Rx, Baud, CLK, RST.
- Les sorties : RxRDY, RxErr, Dout.

Et l'architecture est composée avec des signaux internes, et des états de réception suivants :

- Les signaux : Divisor ; Top16 ; Div16 ; TopRx ; RxDiv ; ClrDiv ; Rx_Reg ; RxRdyi ; RxFSM ; RxBitCnt ; NDBits ; debug.
- Les états de réception : Idle ; Stop_Rx ; Edge_Rx ; Shift_Rx ; Start_Rx,RxOvf.

On à obtenir le schéma interne du récepteur dans la figure IV.12.

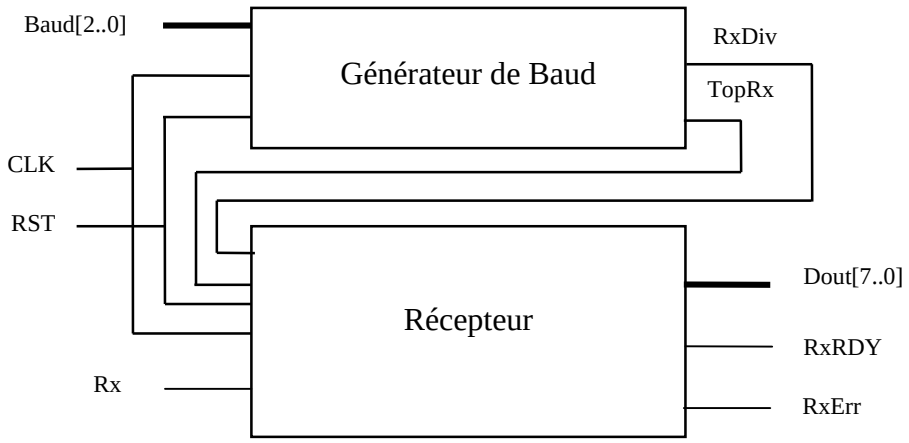


Fig.IV.12.Schéma interne d'un récepteur

IV.2.3.1 La description VHDL du récepteur

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
```

```
Entity recepteur is
Port ( Rx : In std_logic;
       Baud : In std_logic_vector (2 downto 0)  ;
       CLK : In std_logic;
       RST : in std_logic;
       Dout : Out std_logic_vector(7 downto 0);
       RxRDY : Out std_logic;
       RxErr : Out std_logic
       );
end recepteur;
Architecture arch_recepteur of recepteur is
begin
process (RST, CLK)
begin
    if RST='1' then
         Divisor <= 0;
    elsif rising_edge(CLK) then
   case Baud is
        when "000" => Divisor <= 7; -- 115.200
        when "001" => Divisor <= 15; -- 57.600
        when "010" => Divisor <= 23; -- 38.400
        when "011" => Divisor <= 47; -- 19.200
        when "100" => Divisor <= 95; -- 9.600
        when "101" => Divisor <= 191; -- 4.800
        when "110" => Divisor <= 383; -- 2.400
        when "111" => Divisor <= 767; -- 1.200
        when others => Divisor <= 7; -- n.u.
   end case;
end if;
end process;
process (RST, CLK)
begin
    if RST='1' then
        Top16 <= '0';
```

```
        Div16 <= 0;
    elsif rising_edge(CLK) then
        Top16 <= '0';
          if Div16 = Divisor then
              Div16 <= 0;









              Top16 <= '1';
          else
               Div16 <= Div16 + 1;
           end if;
    end if;
end process;
process (RST, CLK)
begin
    if RST='1' then
        TopRx <= '0';
        RxDiv <= 0;
    elsif rising_edge(CLK) then
        TopRx <= '0';
        if ClrDiv='1' then
            RxDiv <= 0;
        elsif Top16='1' then
            if RxDiv = 7 then
                  RxDiv <= 0;
                  TopRx <= '1';
            else
                  RxDiv <= RxDiv + 1;
            end if;
         end if;
    end if;
end process;
```

```
Rx_FSM: process (RST, CLK)
begin
   if RST='1' then
       Rx_Reg <= (others => '0');
       Dout <= (others => '0');
       RxBitCnt <= 0;
       RxFSM <= Idle;
       RxRdyi <= '0';
       ClrDiv <= '0';
       RxErr <= '0';
   elsif rising_edge(CLK) then
          ClrDiv <= '0'; -- default value
          -- reset error when a word has been received Ok:
          if RxRdyi='1' then
               RxErr <= '0';
              RxRdyi <= '0';
          end if;
    case RxFSM is
          when Idle => -- wait on start bit
              RxBitCnt <= 0;
              if Top16='1' then
                  if Rx='0' then
                      RxFSM <= Start_Rx;
                      ClrDiv <='1'; -- Synchronize the divisor
                  end if; -- else false start, stay in Idle
              end if;
          when Start_Rx => -- wait on first data bit
              if TopRx = '1' then
                   if Rx='1' then -- framing error
                        RxF
```

```
SM <= RxOVF;
                    report "Start bit error." severity note;
                else
                    RxFSM <= Edge_Rx;
                end if;
            end if;

<= Stop_Rx;
                else
                    RxFSM <= Shift_Rx;
                end if;
            end if;
        when Shift_Rx => -- Sample data !
            if TopRx = '1' then
                RxBitCnt <= RxBitCnt + 1;
                -- shift right :
                Rx_Reg <= Rx & Rx_Reg (Rx_Reg'high downto 1);
                RxFSM <= Edge_Rx;
            end if
Idle;
                assert (debug < 1)
                report "Character received in decimal is : "
                & integer'image(to_integer(unsigned(Rx_Reg)))
                severity note;
            end if;
        when RxOVF => -- Overflow / Error
            RxErr <= '1';
            if Rx='1' then
                RxFSM <= Idle;
            end if;
    end case;
    end if;
end process;
end arch_recepteur;
```

Apres la description VHDL du récepteur, on à obtenir le schéma de récepteur avec leur entrées et sorties comme montre dans la figure IV.13.

Fig.IV.13. Récepteur d'un UART

IV.3 DESCRIPTION DE L'UART COMPLET

Apres la description de chaque composant, on les regroupe pour avoir l'UART complet « générateur d'horloge + l'émetteur+Récepteur »

IV.3.1 La description VHDL de l'UART

La description VHDL de l'UART est basée sur les trois descriptions VHDL précédentes.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
-- ----------------------------------------------
Entity UART is
-- ----------------------------------------------
Port ( DIN : In std_logic_vector(7 downto 0);









end UART;
-- ----------------------------------------------
Architecture arch_UART of uart is
  type state_Tx is (Idle, Stop_Tx, Shift_Tx, Load_Tx);
  type state_Rx is (Idle, Stop_Rx, Edge_Rx, Shift_Rx, Start_Rx,RxOvf);
```

```
begin
process (RST, CLK)
begin
        if RST='1' then
                Divisor <= 0;
        elsif rising_edge(CLK) then
                case Baud is
                        when "000" => Divisor <= 7; -- 115.200
                        when "001" => Divisor <= 15; -- 57.600
                        when "010" => Divisor <=23; -- 38.400
                        when "011" => Divisor <= 47; -- 19.200
                        when "100" => Divisor <= 95; -- 9.600
                        when "101" => Divisor <= 191; -- 4.800
                        when "110" => Divisor <= 383; -- 2.400
                        when "111" => Divisor <= 767; -- 1.200
                        when others => Divisor <= 7; -- n.u.
                end case;
        end if;
end process;
process (RST, CLK)
begin
        if RST='1' then
                Top16 <= '0';
                Div16 <= 0;
        elsif rising_edge(CLK) then
                Top16 <= '0';
                if Div16 = Divisor then
                        Div16 <= 0;
                        Top16 <= '1';
                else
                        Div16 <= Div16 + 1;
                end if;
        end if;
end process;
process (RST, CLK)
begin
        if RST='1' then
                TopTx <= '0';
                ClkDiv <= 0;
        elsif rising_edge(CLK) then
                TopTx <= '0';
                if Top16='1' then
                        ClkDiv <= ClkDiv + 1;

;
                if ClrDiv='1' then
                        RxDiv <= 0;
```

```
                elsif Top16='1' then
                        if RxDiv = 7 then
                                RxDiv <= 0;
                                TopRx <= '1';
                        else
                                RxDiv <= RxDiv + 1;
                        end if;
                end if;
        end if;
end process;









TX <= Tx_Reg(0);
Tx_FSM: process (RST, CLK)
begin
if RST='1' then
        Tx_Reg <= (others => '1');
        TxBitCnt <= 0;
        TxFSM <= idle;
        TxBusy <= '0';
        RegDin <= (others=>'0');
elsif rising_edge(CLK) then
        TxBusy <= '1'; -- except when explicitly '0'
        case TxFSM is
                when Idle =>
                        if LD='1' then
                                -- latch the input data immediately.
                                RegDin <= Din;
                                TxBusy <= '1';
                                TxFSM <= Load_Tx;
                        else
                                TxBusy <= '0';
                        end if;
                when Load_Tx =>
                        if TopTx='1' then
                                TxFSM <= Shift_Tx;
                                if parity then
                                        -- start + data + parity
```

```
                                    TxBitCnt <= (NDBits + 2);
                                    Tx_Reg <= '1'& Din & '0';
                                else
                                    TxBitCnt <= (NDBits + 1); -- start +
                                data
                                    Tx_reg <= '1' & RegDin & '0';
                                end if;
                        end if;
                when Shift_Tx =>
                        if TopTx='1' then
                                TxBitCnt <= TxBitCnt - 1;
                                Tx_reg <= '1' & Tx_reg (Tx_reg'high
                        downto 1);
                                if TxBitCnt=1 then
                                        TxFSM <= Stop_Tx;
                                end if;
                        end if;
                when Stop_Tx =>
                        if TopTx='1' then
                                TxFSM <= Idle;
                        end if;
                when others =>
                        TxFSM <= Idle;
        end case;
end if;
end process;


Rx_FSM: process (RST, CLK)
begin
if RST='1' then
        Rx_Reg <= (others => '0');
        Dout <= (others => '0');
        RxBitCnt <= 0;
        RxFSM <= Idle;
        RxRdyi <= '0';
        ClrDiv <= '0';
        RxErr
```

```
                <= 0;
                if Top16='1' then
                        if Rx='0' then
                                RxFSM <= Start_Rx;
                                ClrDiv <='1'; -- Synchronize the
                        divisor
                        end if; -- else false start, stay in Idle
                end if;

        when Start_Rx => -- wait on first data bit
                if TopRx = '1' then
                        if Rx='1' then -- framing error
                                RxFSM <= RxOVF;
                                report "Start bit error." severity note;
                        else

                                RxFSM <= Edge_Rx;
                        end if;
                end if;
        when Edge_Rx => -- should be near Rx edge
                if TopRx = '1' then
                        RxFSM <= Shift_Rx;
                        if RxBitCnt = NDbits then
                                RxFSM <= Stop_Rx;
                        else
                                RxFSM <= Shift_Rx;
                        end if;
                end if;
        when Shift_Rx => -- Sample data !
                if TopRx = '1' then
                        RxBitCnt <= RxBitCnt + 1;
                        -- shift right :
                        Rx_Reg <= Rx & Rx_Reg (Rx_Reg'high
                downto 1);
                        RxFSM <= Edge_Rx;
                end if;
        when Stop_Rx => -- during Stop bit
                if TopRx = '1' then
```

```
                                    Dout <= Rx_reg;
                                    RxRdyi <='1';
                                    RxFSM <= Idle;
                                    assert (debug < 1)
                                    report "Character received in decimal is : "
                                    &
                            integer'image(to_integer(unsigned(Rx_Reg)))
                                    severity note;
                            end if;
                    when RxOVF => -- Overflow / Error
                            RxErr <= '1';
                            if Rx='1' then
                                    RxFSM <= Idle;
                            end if;
            end case;
    end if;
    end process;
    end arch_UART ;
```

IV.3.2 Le schéma bloc utilisée

Pour tester le fonctionnement de cet UART par la simulation, on a créé un fichier qui contient ce symbole de ce circuit à partir de la description VHDL précédente. On a utilisé le projet « UART » et « UART.vhd » au moyen de la commande de Quarus II :

File ⟶ Create/Update ⟶ Create Symbol Files For Current File

On obtient, dans la librairie du projet, le schéma bloc suivant (figure IV.14)

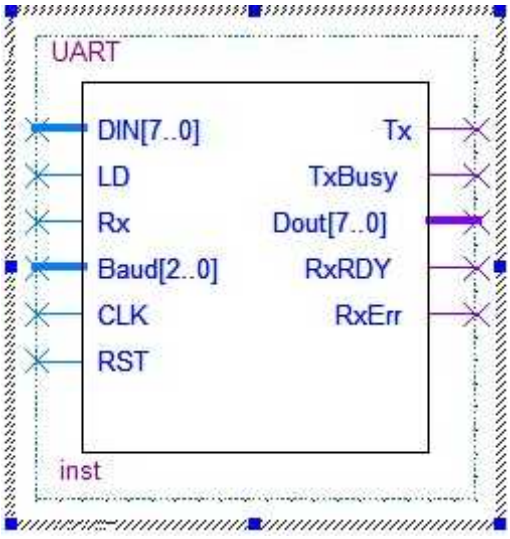


Fig.IV.14. Schéma de circuit complet UART

On va ouvrir un nouveau fichier « Bloc Diagram/schematic File » pour utiliser l'UART dans un schéma qui sera soumis à la simulation.

Pour faire la simulation nous devons simple relation suivante, donner la valeur pour être transféré à l'entré « DIN », et relié l'entrée Rx et la sortie Tx (voire la figure IV.15).

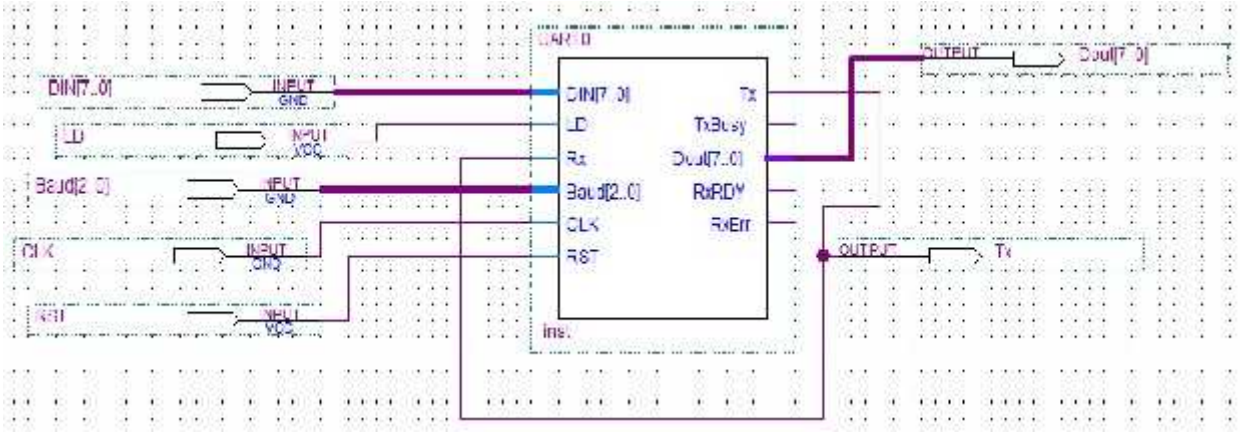


Fig.IV.15. Schéma utilisé dans la simulation de l'UART

Ce schéma nous permet de voir le fonctionnement des deux parties émission et réception. L'octet à transmettre est à l'entrée DIN la sortie de l'émission Tx est considérée comme l'entrée Rx de la partie réception qu'on doit avoir à sa sortie 'Dout' (un octet).

La valeur 'DIN' doit être retrouvée à la sortie 'Dout'

IV.3.3 La simulation

Cette étape permet de certifier le fonctionnement exact de l'UART, on a obtenu le rapport de simulation de quelques exemples

- 1er exemple on a envoyé la valeur « A9 » avec la vitesse de transmission 115200 baud (l'entrée Baud à '000'), le rapport de simulation donné par la figure IV.16

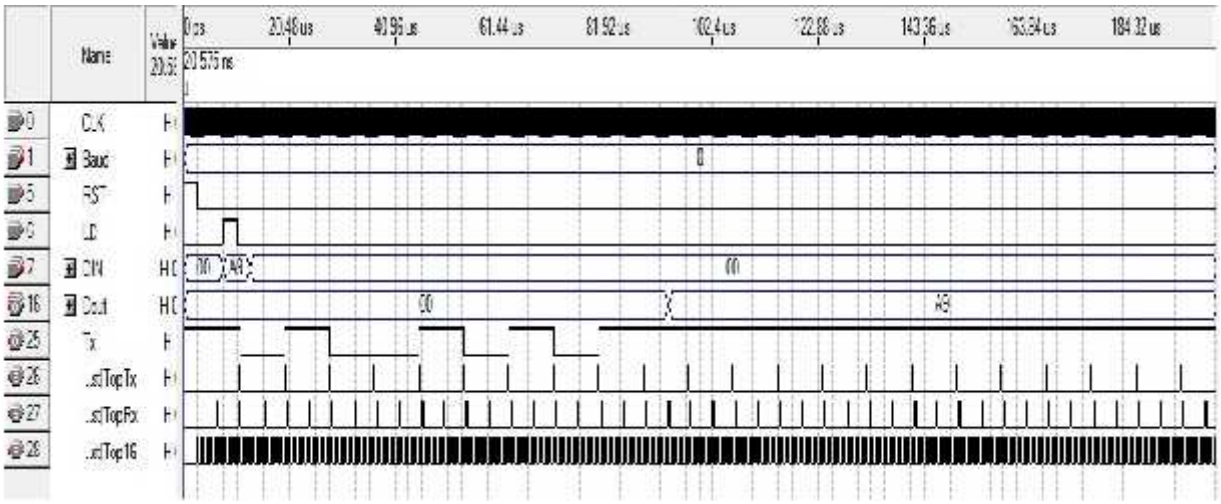


Fig.IV.16. Le rapport de simulation de l'exemple « A9 »

« A9 » en hexadécimale, et en binaire est égale « 10101001 ».

Le signale Tx nous donne **STAR_TBIT 10010101 STOP_BIT**

Dans la réception on a trouvé la valeur qui est transmise « **A9** »

- 2ème exemple, on a utilisé la valeur «B7 » avec vitesse 19200 baud (l'entré Baud à $(011)_b$), le rapport de simulation donné par la figure IV.17.

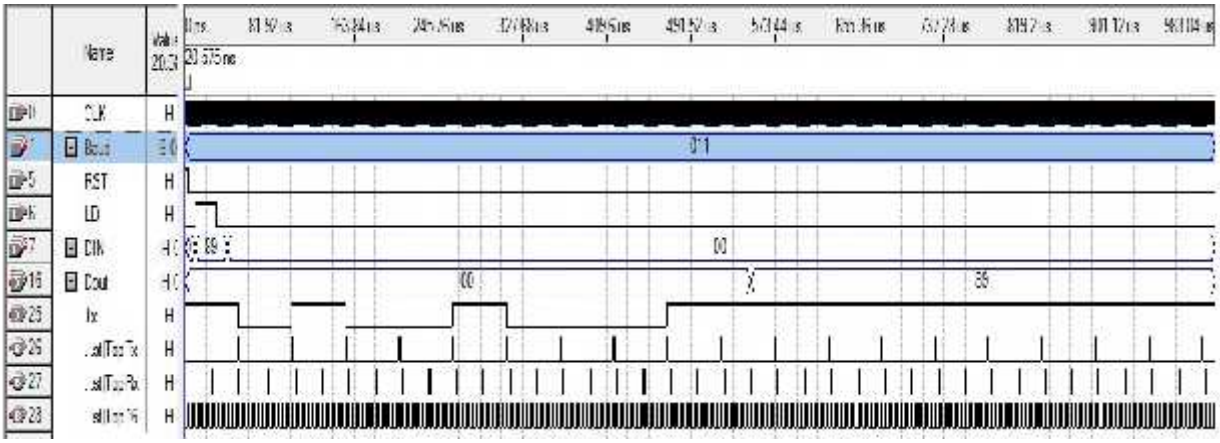


Fig.IV.17. Le rapport de simulation de l'exemple « 89 »

« 89 » en hexadécimale, et en binaire est égale « 10001001 ».

Le signale Tx nous donne **STAR_TBIT 10010001 STOP_BIT**

Dans la réception on a trouvé la valeur qui est transmise « **89** »

IV.4 CONCLUSION

Après la description VHDL du l'UART et la vérification de son fonctionnement à partir la simulation avec câblage entre l'émetteur et le récepteur pour un même UART, nous avons trouvé bons résultats qui nous permettent implémenté dans un circuit logique programmable FPGA.

CONCLUSION ET PERSPECTIVES

Comme on vient de le voir, la transmission des données dans la norme RS-232, est gérée par l'UART, qui délivre la trame à transmettre. Cette dernière est caractérisée par la durée de transmission de chaque bit qui correspond à un cycle de l'horloge de transmission Tx. C'est le rôle du générateur de baud.

Le format de la trame (bit de Start, bits de données, bit de parité, bit de Stop), est formé au niveau de l'état 'load_Tx' de la machine d'état de l'émetteur.

La machine d'état qui constitue le récepteur, se charge de l'extraction de la donnée à partir du format de la trame de l'émetteur.

Lors de la simulation on a fait un câblage entre l'émetteur et le récepteur pour un même UART pour se rendre compte que le format de la trame existe réellement et que c'est bien à partir de cette même trame que les bits de données seront obtenus.

La simulation témoigne que la réalisation par description VHDL de l'UART en été atteinte cependant des améliorations peuvent être apportées principalement :

- Sur la longueur des bits donnés, c'est-à-dire permettre la programmation de la longueur des mots.
- Avec ou sans bit de parité paire/impaire.
- Faire sortir les signaux de contrôle de flux des échanges de données, selon le protocole RTS/CTS ou handshaking matériel.

GLOSSAIRE

UART : Universal Asynchronous Receiver/Transmitter.

VHSIC : Very High Speed Integrated Circuit.

VHDL : VHSIC Hardware Description Languag.

FPGA : Field-Programmable Gate Array.

IEEE Institute of Electrical and Electronics Engineers.

PC : Program Cunter.

RS : The Recommended Standard.

SCSI : Small Computer System Interface.

SYNC : Synchronization.

DTE Data Terminal Equipement.

DCE Data Communication Equipement.

SAS : Serial attached SCSI).

EIA : The Electronic Industries Association.

FSM : Finite State Machine.

EDA : Electronic Design Automation.

BIBLIOGRAPHIE

[1] la liaison RS232 Description technique et mise œuvre 2ème édition PHILIPPE ANDRÉ, Dunod, Paris, 2002.

[2] http://www.chicoree.fr/w/Full-duplex,_half-duplex_et_simplex , 2011

[3] http://fr.wikipedia.org/wiki/Duplex_%28canal_de_communication%29 , avril 2013.

[4] STI GE « LES LIAISONS SERIE » Lycee Felix Le Dantec, pdf, 2010.

[5] Léon CLEMENT « Microprocesseurs et circuits associés» 1987 2ème édition.

[6] Madame Céline GUILLEMINOT « ÉTUDE ET INTÉGRATION NUMÉRIQUE D'UN SYSTÈME MULTICAPTEURS AMRC DE TÉLÉCOMMUNICATION BASÉ SUR UN PROTOTYPE VIRTUEL UTILISANT LE LANGAGE DE HAUT NIVEAU VHDL-AMS » MEMOIRE de THESE DOCTORAT de L'UNIVERSITE de TOULOUSE II 01 décembre 2005.

[7] KAHOUL NADHIR « COURS VHDL PLD » (introduction au langage de description VHDL avec les circuits logiques programmables et le logiciel de développement QuartusII). Université Med KHIDER BISKRA, Département de Génie Electrique, Filière Electronique, 2012.

[8] JERRY TEKOBEN « FPGA and Traffic Network Analysis » Ingénieur de conception en Génie Electrique et Télécommunication, Ecole Nationale Supérieure Polytechnique de Yaounde Cameroun 2007.

[9] MICHEL AUMIAUX « Initiation au langage VHDL » 2ème édition, Docteur ès sciences Professeur à l'école supérieure d'électronique de l'ouest (Angers), Dunod, Paris, 1999.

[10] http://www2.toulouse.iufm.fr/iufm/cours/vhdl/vhdl.pdf , T. BLOTIN, Lycée Paul-Eluard, 93206 SAINT-DENIS.

[11] Denis Rabasté,IUFM d'Aix-Marseille « programmation des CPLD et FPGA en VHDL avec Quartus II » http://genelaix.free.tf/IMG/pdf/aide_VHDL_quartus.pdf

[12] AZIZI ABDALLAH « ANALYSE ET CONCEPTION EN VHDL DE L'ARCHITECTURE D'UN MICROPROCESSEUR » Mémoire fin d'étude 2ème Master Promotion 2010/2011, Université Med KHIDER BISKRA, Département de Génie Electrique, Filière Electronique.

[13] D.DUBOIS « Programmation du contrôleur série UART Intel 8250 » pdf, 2012.

[14] Altera « Notice de prise en main du logiciel Quartus II »,2010 pdf.

[15] ELE1300 - Circuits Logiques Laboratoire 1 Utilisation du logiciel Quartus II d'Altera, 2011.

[16] Haute Ecole d'Ingénierie et de Gestion du Canton de VHDL « Manuel VHDL synthèse et simulation » septembre 2007.

[17] Le Port Série du PC « Universal Asynchronous Receiver Transmitter », TP5 2007 pdf.

Annexe

A. Création d'un projet dans logiciel Quartus II

INTRODUCTION [14]

Quartus est un logiciel développé par la société Altera, permettant la gestion complète d'un flot de conception CPLD ou FPGA. Ce logiciel permet de faire une saisie graphique ou une description HDL (VHDL ou verilog) d'architecture numérique, d'en réaliser une simulation, une synthèse et une implémentation sur cible reprogrammable.

Il comprend une suite de fonctions de conception au niveau système, permettant d'accéder à la large bibliothèque d'IP d'Altera et un moteur de placement-routage intégrant la technologie d'optimisation de la synthèse physique et des solutions de vérification. De manière générale, un flot de conception ayant pour but la configuration de composants programmables se déroulent de la manière suivante :

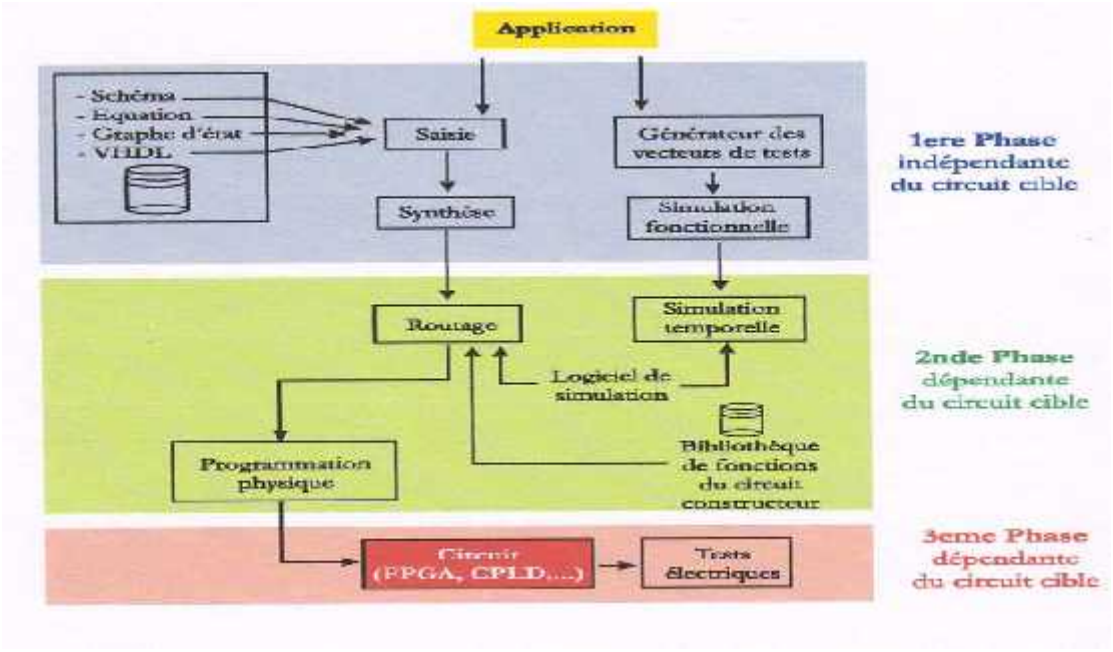


Fig.1. La manière de conception circuit programmable

2. Création d'un projet

Quartus est un logiciel qui travaille sous forme de projets c'est à dire qu'il gère un design sous forme d'entités hiérarchiques. Un projet est l'ensemble des fichiers d'un design que ce soit des saisies graphiques, des fichiers VHDL ou bien encore des configurations de composants (affectation de pins par exemple).

Pour lancer le logiciel, on cliquera sur :

Démarrer ⟶ Programmes ⟶ Altera ⟶ Quartus II 7.2

La fenêtre suivante s'ouvre : (voir la figure 2)

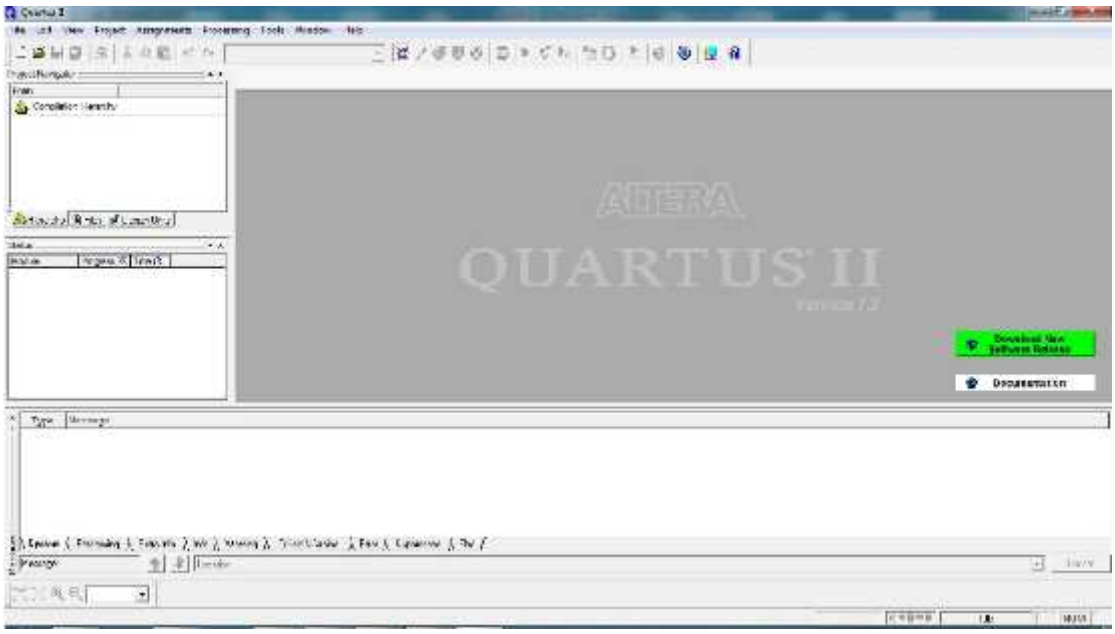


Fig.2. Fenêtre présentation Quartus II

Afin de créer un nouveau projet aller dans le menu :

File ⟶ New Project Wizard

- La figure 3 présentation de la fenêtre **« New Project Wizard : Introduction »** qui s'ouvre pour fournir de drèves explication, cliquer sur **Next**

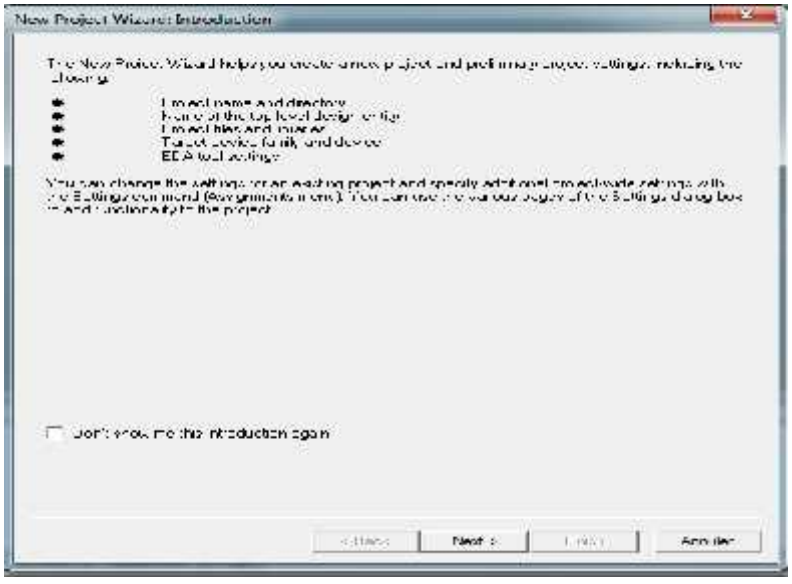


Fig.3. Fenêtre d'exploitation de logiciel

- Puis se laisser guider.la figure 4 présenté une nouvelle fenêtre **« New Project Wizard : Directory,Name,Top-Level Entity [page1 of 5] »** permettant de configurer le projet apparaît. Dans cette dernière trois champs sont à renseigner :

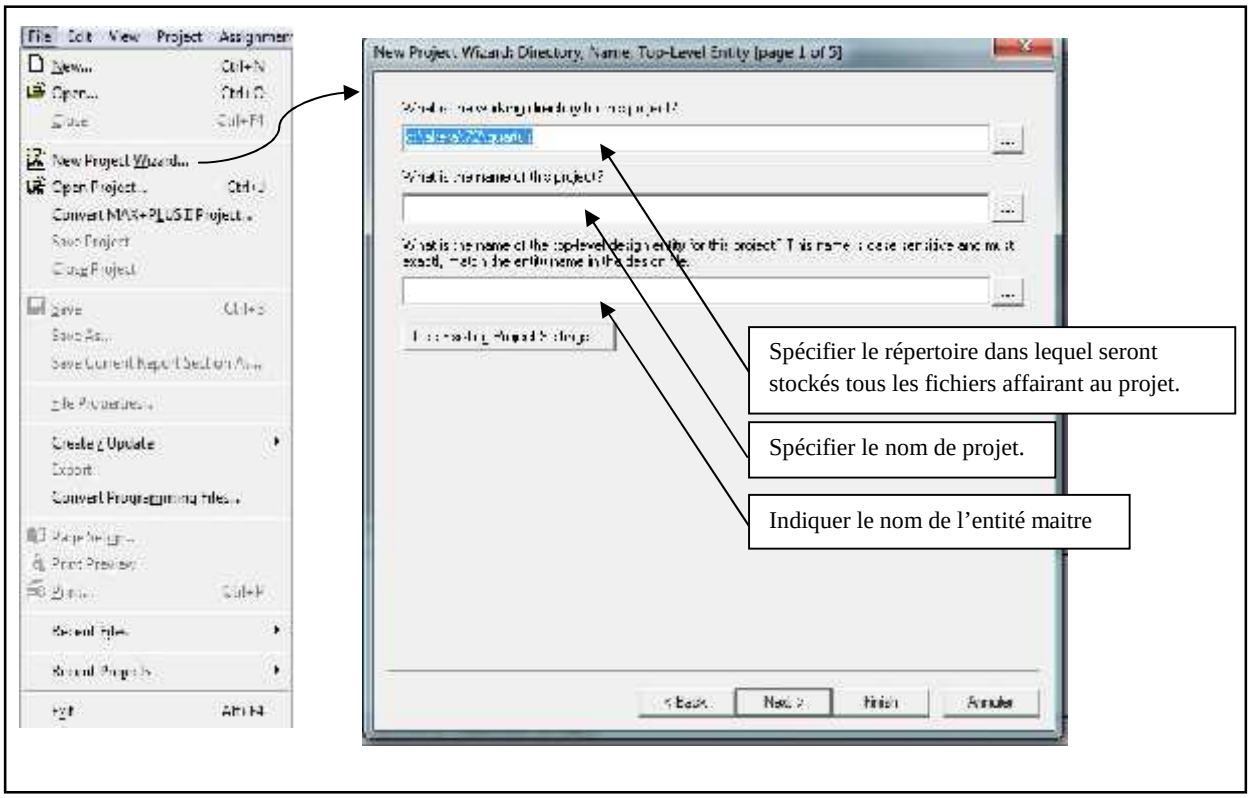


Fig.4. L'enregistrement et nommé le projet et l'entité

- L'emplacement du répertoire où seront stockés tous les fichiers. Il est conseillé de créer un répertoire propre afin d'assurer la sauvegarde des données d'une séance surl'autre,
- Le nom du projet,
- Le nom de l'entité maître du projet. On appelle entité maître, le design qui correspond à la couche hiérarchique la plus haute. Par exemple s'il on conçoit un schéma nommé additionneur dans lequel il y aura plusieurs composants graphiques ou décrit en VHDL, alors additionneur sera l'entité maître. En d'autres termes, additionneur ne dépend pas d'autres fichiers, c'est le niveau le plus haut dans le design.

- La figure 5 présenté la fenêtre «**New Project Wizard :Add Files[page 2 of 5]**

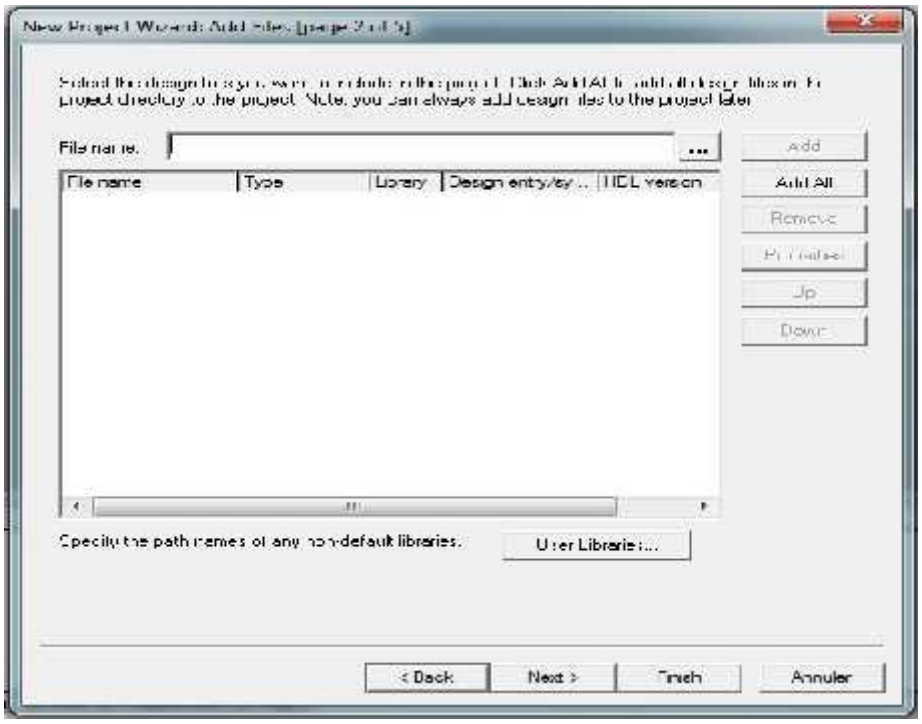


Fig.5. Fenetre d'ajouter des fichiers au projet

- Cliquer sur **Next**, puis quand la fenêtre **Add Files** apparaît recliquer sur **Next**. Dans la fenêtre suivante intitulée « **New Project Wizard :Family & Device Settings[page 3 of 5] »**, choisir le circuit logique programmable que l'on souhaite utiliser. Comme définie dans la figure 6.

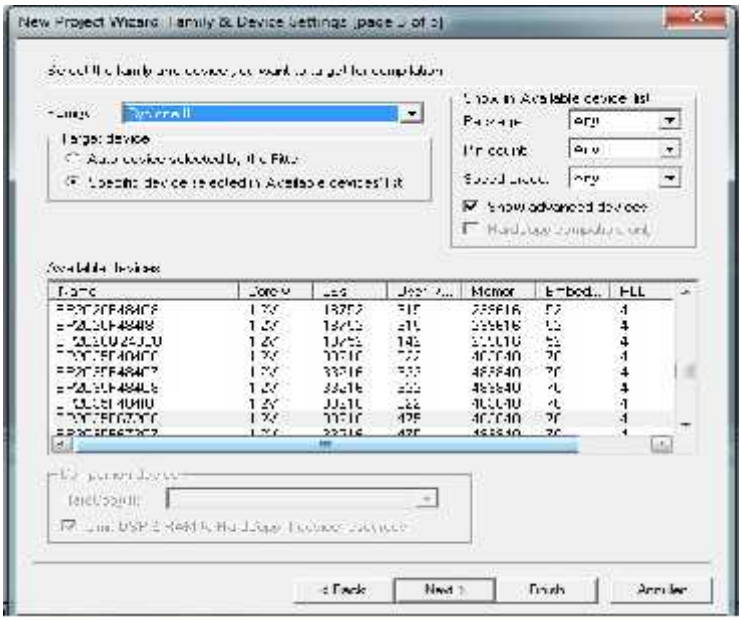


Fig.6. Fenêtre de choisir la famille et devise pour la compilation

- La figure 7 présenté la fentetre « **New Project Wizard : EDA Tool Setting [page 4 of 5]**

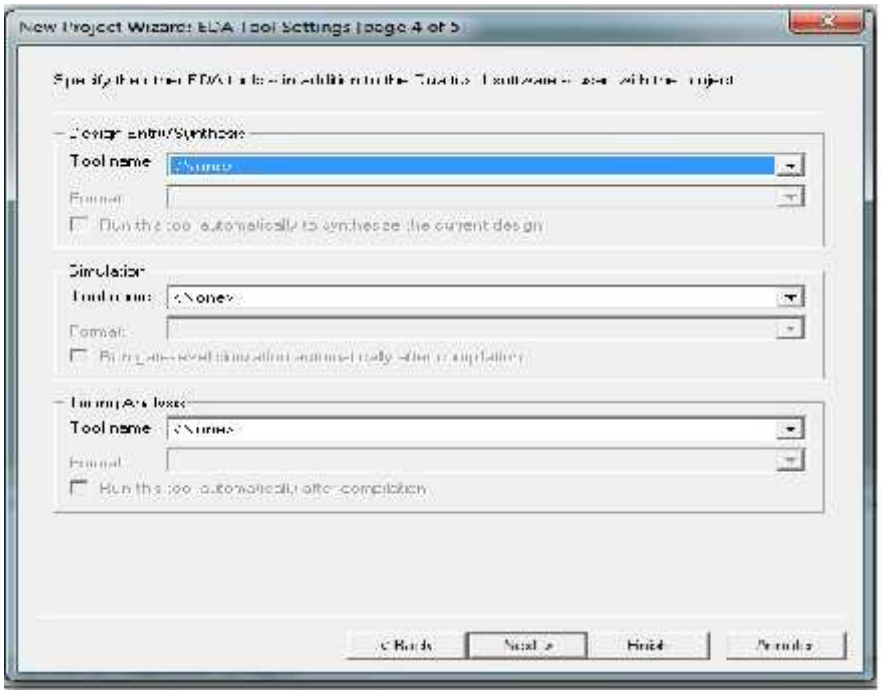


Fig.7. Fenêtre de ETD Tool Setting

- Quand la fenêtre EDA Tool Settings apparaît cliquer sur **Next**. Une fenêtre récapitulative apparaît, comme montre la figure 8.

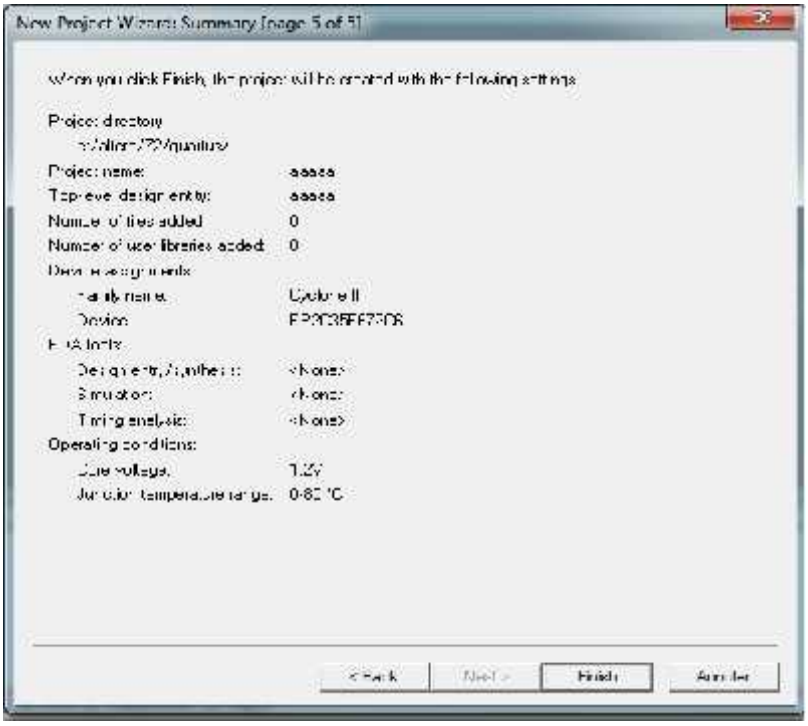


Fig.8. Le résumé pour le projet

Valider les choix par **Finish** ou bien faire **Back** pour des modifications éventuelles. Dans le navigateur de Projet, un onglet avec le type composant et l'entité maître apparaît (la figure 9)

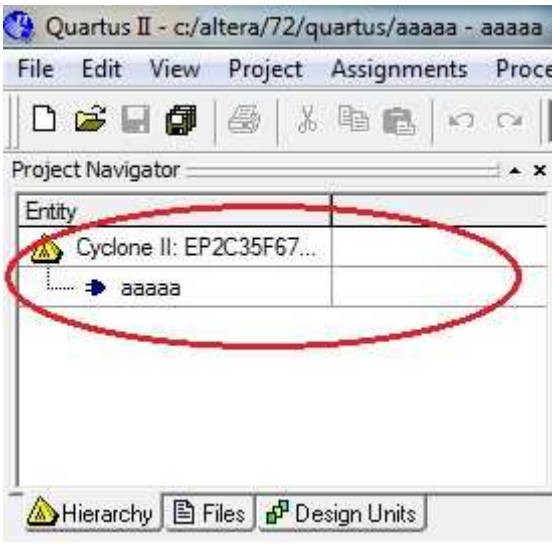


Fig.9. L'ouverture de projet

B. Saisie graphique en assemblant des symboles.

1 Saisie d'un projet : [14]

Cette étape permet de définir et configurer les différentes parties du projet. Quartus accepte plusieurs types de saisie à savoir :

- Un saisie graphique en assemblant des symboles.
- Une saisie texuelle à l'aide de différents langage VHDL,Verilog,AHDL,…).

1.1 Création d'un schéma

File ⟶ New

On a trouvé la figure 10

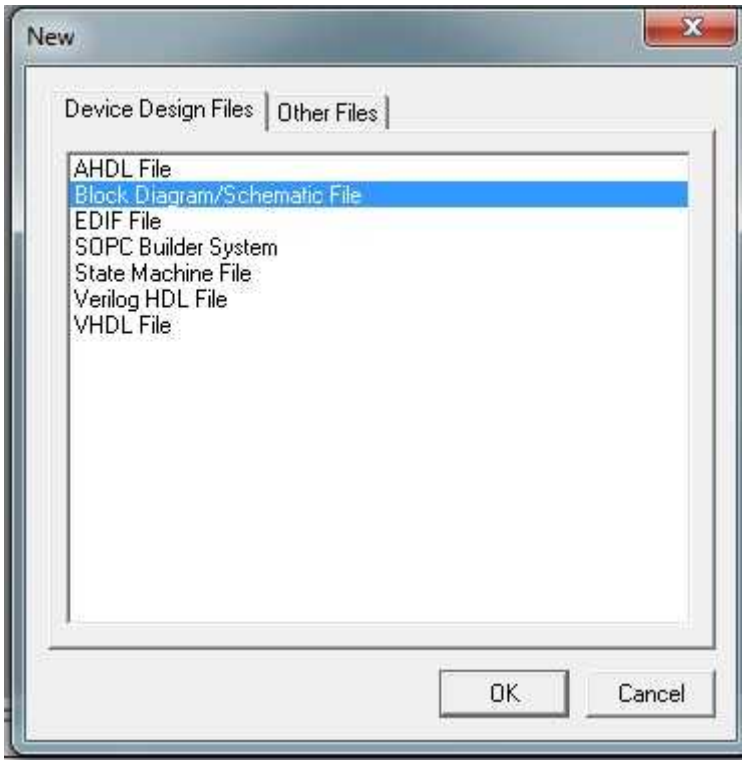


Fig.10. Fenêtre de création d'un fichier de description schématique

Dans l'onglet **« Device Desing Files »,** choisissez « **Block Diagram/Schematic File »** et on fait **« OK ».**

Sauvegarder votre fichier sous le nom Block1 **« Block1.bdf »**,qui sera ainsi reconnu comme entité de haut-niveau et assure-vous de sauvegarder le fichier dans votre répertoire de projet.

File ⟶ Save as

Maintenant, il sera possible de dessiner un circuit donné.

Dans la barre d'outils schématique (Figure 11), vous pouvez voir la «Flèche» qui sert à sélectionner les objets. Le bouton « Symbol Tool » vous permet de choisir l'élément que vous voulez insérer dans le circuit tel que : porte logique, entrée/sortie ou une macro-fonction. Le bouton « Orthogonal Node Tool » vous permet de dessiner les connexions entre vos portes logiques. Le bouton « Orthogonal Bus Tool » vous permet de dessiner un bus (un bus peut être vu comme un câble contenant plusieurs fils). Ensuite, lorsque l'option « Use Rubberbanding » est activée, les fils sont "soudés" lorsque mis en contact et ils restent reliés (aux portes notamment) ensembles lorsque des déplacements sont appliqués aux composants du circuit (évitez de la désactiver).

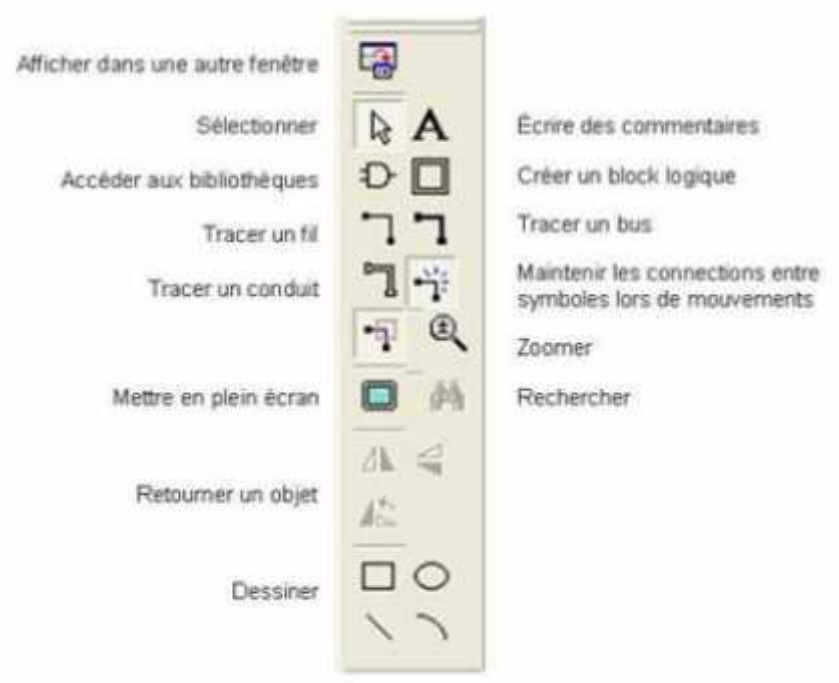


Fig.11. Barre d'outils de la fenêtre schéma

En choisissant **« Symbol Tool »,** vous verrez une fenêtre apparaître. Cliquez sur le dossier **« Q:/altera/quartus…/librairies/ »** dans la section librairies, à droite. Choisissez **« primitive »** et vous avez accès à tous les éléments logiques de base. Le dossier **« Logic »** contient toutes les portes logiques **« and »**, **« or »**, **« nand »** etc. utiles pour le cours. Le dossier **« pin »** contient les entrées et sorties que vous pouvez utiliser conformément au fichier d'assignation que nous avons ajouté au projet. Le dossier **« storage »** contient les différentes bascules que vous verrez dans le cours.

Sélectionnez une porte ET à 2 entrées (and2) et cliquez sur OK. Vous verrez que l'objet sélectionné est attaché à votre souris. À chaque fois que vous cliquez, il placera un objet à l'endroit où est situé le curseur. Pour ne plus placer d'objet, cliquez sur la flèche dans la barre d'outils à gauche de la fenêtre schématique ou pesez sur **« Escape ».**

- Notez que l'inverseur est dénommé "not".

  Dans un premier temps, il vous est simplement demandé de reproduire le circuit illustré à la Figure 4. Ce circuit produit simplement à ces 3 sorties le résultat des fonctions logiques ET, OU, et OUX à deux entrées. Sur la carte, nous allons relier les 2 entrées aux interrupteurs SW[0] et SW[1], et les 3 sorties aux DELs rouges LEDR[0], LEDR[1], et LEDR[2].(voir la figure 12)

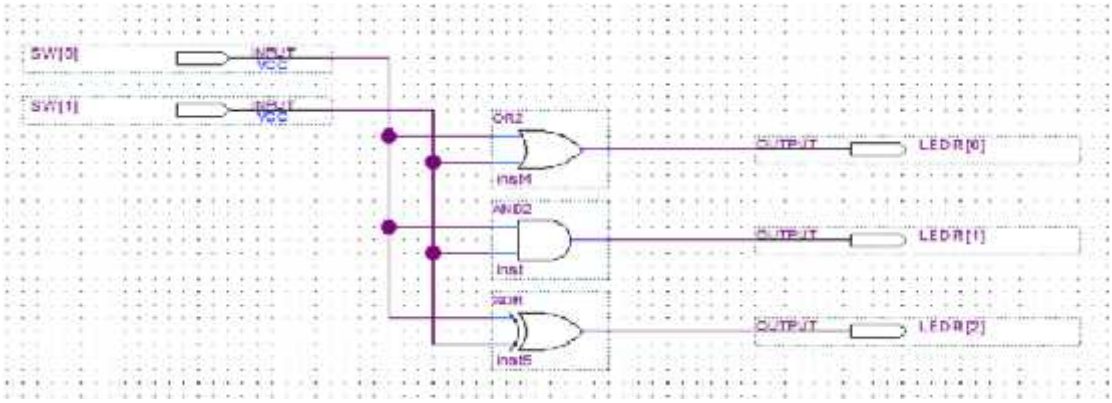


Fig.12. Réalisation le circuit OU_EX

Commencez par aller chercher les portes logiques dont vous avez besoin ainsi que vos entrées et sorties en appuyant sur le bouton **Symbol Tool**.

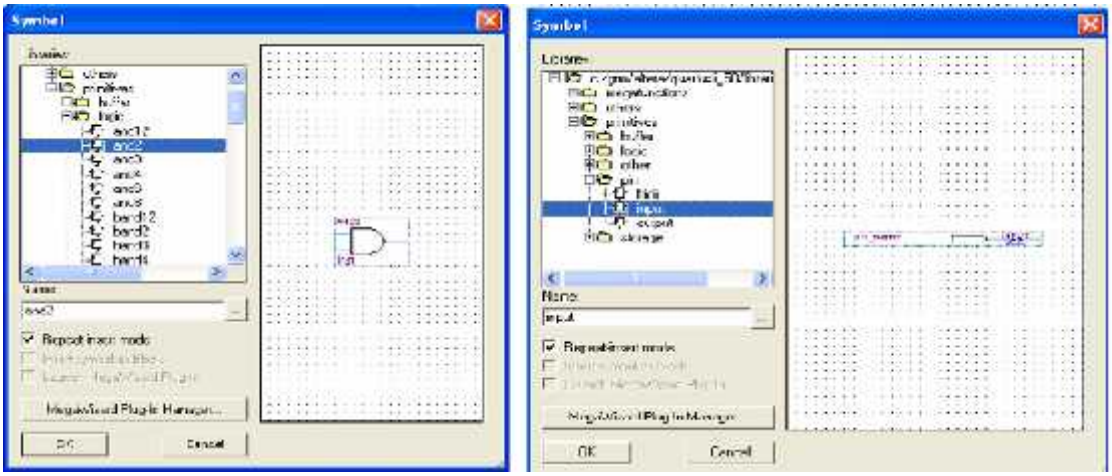


Fig.13. Libraires d'outils de la fenêtre schématique

Ensuite, il faut nommer les entrées/sorties en utilisant les noms définis dans le fichier utilisé pour l'assignation des broches (DE2_pins.csv). De cette façon, Quartus II fera la juste correspondance entre le schéma et les connexions physiques sur la carte. Vous changerez les noms en double-cliquant sur l'entrée ou la sortie (figure 14).

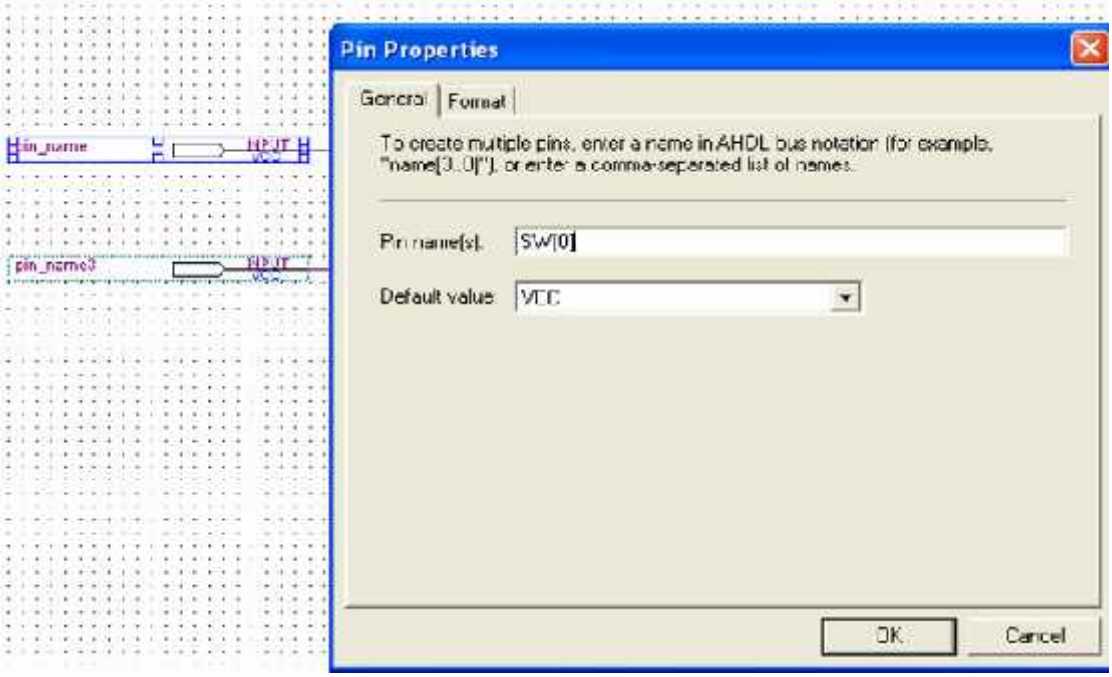


Fig.14. Changement du nom d'une broche dans le circuit

C. Saisie textuelle en VHDL [14]

La saisie d'un composant VHDL se fait de la même manière que précédemment. Pour cela, aller dans le menu :

File ⟶ New

La fenêtre suivante apparaît : (voire la figure 15)

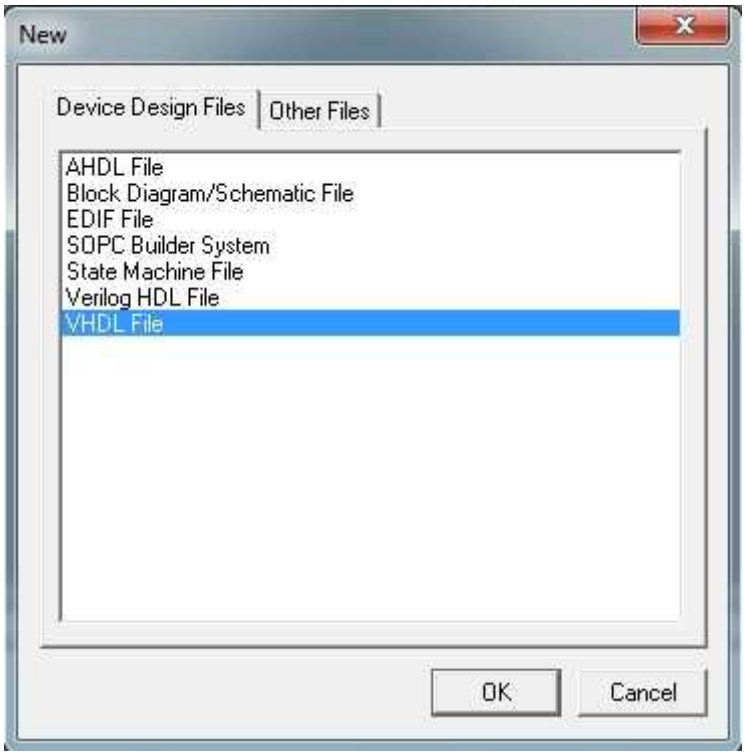


Fig.15. Fenêtre de création d'un fichier de description textuelle

Choisir « **VHDL File »** et faire « **OK** ». Un petit éditeur de texte apparaît. Afin de fixer les idées, nous allons créer un petit composant réalisant un ET logique sur 4 bits par le programme suivant :

```
library ieee;
use ieee.std_logic_1164.all;
ENTITY ET4 IS
        PORT(
                a, b : IN bit_vector(3 downto 0);
                s : OUT bit_vector(3 downto 0));
        END ET4 ;
ARCHITECTURE data_flow OF ET4 IS
BEGIN
                s<= a AND b;
END data_flow ;
```

Une fois le code VHDL saisie, il convient de le sauver 1 (**File —>Save** sous le nom ET4 par exemple) puis d'en vérifier la syntaxe. Pour cela, on cliquera sur l'icône entouré noté 1 et l'on suivra l'évolution de la vérification en 2 et les différentes informations s'afficheront en 3. (Figure 16).

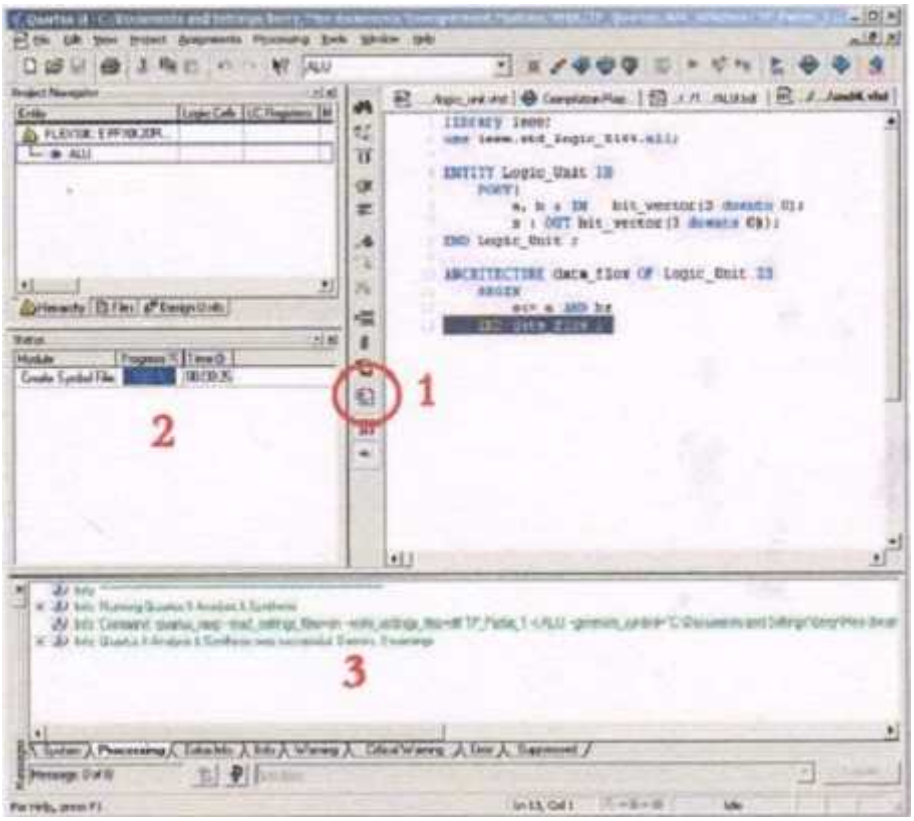


Fig.16. Fenêtre de description VHDL d'un AND

D. Normes IEEE [16]

Voici la liste des normes IEEE pour le langage VHDL. Nous les avons classé selon leur date d'édition.

- 1980 Début du projet financé par le DoD
- 1987 Standard IEEE Std 1076-1987 (VHDL 87)
- 1993 Standard IEEE Std 1164-1993 (Std_Logic_1164)
- 1993 Standard IEEE Std 1076-1993 (VHDL 93)
- 1994 Approbation ANSI (ANSIIIEEE Std 1076-1993)
- 1995 Standard IEEE Std 1076.4 (Vital Primitive et Vital Timing)
- 1996 Standard IEEE Std 1076.2 (Mathematical Packages)
- 1997 Standard IEEE Std 1076.3 (Numeric_Bit et Numeric_Std)
- 1999 Standard IEEE Std 1076.6
  (Standard for VHDL Register Transfer Level Synthesis)
- 2000: Standard IEEE Std 1076-2000 (VHDL 2000)
- 2002: Standard IEEE Std 1076-2002 (VHDL 2002)