MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU

FACULTE DE GENIE ELECTRIQUE ET D'INFORMATIQUE

DEPARTEMENT D'AUTOMATIQUE

Mémoire de fin d'études

En vue de l'obtention du Diplôme de :

Master Académique

Spécialité : Génie Microélectronique

Thème

Système de transmission de données sur FPGA

Présenté par :

Mr. LAHLOU Hamza

Proposé et dirigé par :

Mr. Benfdila Arezki

Remerciement

Louange à dieu le miséricordieux qui m'a permis de bien accomplir ce modeste travail. Mes remerciements s'adressent tout particulièrement à mon encadreur pour l'effort fourni, et son soutien, ses conseils scientifiques, sa patience et sa persévérance dans le suivi.

Mes sincères remerciements à tous mes enseignants qui m'ont transmis les bases de la Micro-électronique tout le long de ces années de master.

Un merci particulier à ma famille pour leurs encouragements et leur soutien tout au long de ce travail.

Un grand merci à mes collègues et mes amis avec qui j'ai passé de très bons moments au sein de l'université Mouloud Mammeri.

Que ceux qui se sentent oubliés trouvent ici ma profonde gratitude et mes chaleureux remerciements pour leur concours dans l'accomplissement de ce travail.

MR.LAHLOU

Sommaire

Sommaire

Liste des figures

Liste des figures

Liste des tableaux

Liste des acronymes

CAO : Conception **A**ssisté par **O**rdinateur
CI :Circuit **I**ntégré
CPLD : Complex **P**rogrammable **L**ogic **D**evice
CLB : Configuration **L**ogic **B**loc
CMOS: **C**omplementary **M**etal **O**xide **S**emiconductor
DRAM : Dynamic **R**andom **A**ccess **M**emory **Si : Si**licium
DSP : Digital **S**ignal **P**rocessor
EEPROM : Electrically-**E**rasable **P**rogrammable **R**ead-**O**nly **M**emory
EPROM : Erasable PROM
FET : Field **E**ffect **T**ransistor
FPGA : Field **Pr**ogrammable **G**ate **A**rray
GAL : Generic **A**rray **L**ogic
HDL : Hardware **D**escription **L**anguage
IOB : Input **O**utput **B**loc
LUT : Look **U**p **T**able
MOS : Metal **O**xide **S**emiconductor
MOS FET : Metal **O**xide **S**emiconductor **F**ield **E**ffect **T**ransistor
NMOS : N-canal MOS
NRE : Non-**R**ecurring **E**ngineering
OLMC : Output **L**ogic **M**acro-**C**ell (macro-cellule logique de sortie)
OTP : One-**T**ime **P**rogrammable (circuit programmable une seule fois)
PAL : Programmable **A**rray **L**ogic
PLD : Programmable **L**ogic **D**evice
PMOS : P-canal MOS
PROM : Programmable **R**ead **O**nly **M**emory
RAM : Random **A**ccess **M**emory
RE : Recurring **E**ngineering
ROM : Read **O**nly **M**emory
RTL : Register **T**ransfer **L**evel
SDRAM : Synchronous DRAM
***SDRAM DDR : D**ouble **D**ata **R**ate SDRAM*
SI: Silicon
UART :**U**niversal **A**synchronous **R**eceiver **T**ransmitter
SRAM : Static **R**andom **A**ccess **M**emory
VHDL :VHSIC **H**ardware **D**escription **L**anguage
VHSIC : Very **H**igh **S**peed **I**ntegrated **C**ircuit

Introduction générale

Les progrès réalisés en microélectronique permettent aujourd'hui d'intégrer plusieurs centaines de millions de transistor sur un seul circuit, cette capacité d'intégration augmente exponentiellement selon la loi de Moore.

Ces évolutions technologiques ont particulièrement favorisé la famille des circuits FPGA qui sortent aujourd'hui de leur niche originale.

Les circuits FPGA sont aujourd'hui les principaux circuits reconfigurables disponibles sur le marché. Ils sont aujourd'hui en mesure de fournir une solution efficace à la réalisation matérielle d'applications dans de nombreux domaines, les FPGA offrent une surface homogène d'unités logiques universelles qui peuvent être reconfigurées à volonté de manière à implémenter n'importe quel circuit combinatoire ou séquentielles. Grace à ces avancés spectaculaire de ces circuits, de nombreuses applications complexes qui révèlent de différents domaines on put voir le jour.

L'objectif de ce travail est l'implémentation d'un système de transmission sur une carte de développement FPGA, Virtex-II (V2MB1000), de la famille Xilinx.

Dans ce contexte, ce mémoire est scindé en trois chapitres :

Au premier chapitre nous allons faire une étude générale des circuits programmables et les FPGA en particulier, et les nous allons voir aussi les différentes liaisons séries.

Au deuxième chapitre nous allons aborder le langage VHDL, on clôturera ce chapitre par donner un exemple de machine d'état de Moore.

Au troisième chapitre nous étudierons l'implémentation du système de transmission. Les étapes d'implémentations seront aussi développées.

Nous terminerons ce travail par une conclusion générale suivie par une annexe qui contient tous les programmes VHDL

Chapitre I

Système de communication sur FPGA

1 Introduction

Depuis une cinquantaine d'années, l'évolution de la complexité des circuits intégrés double tous les 18 mois (loi de Moore). Cette évolution exponentielle a permis de réaliser, de manière monolithique, des organes électroniques de plus en plus complexes qui étaient auparavant réalisés sous forme d'armoires (par exemple : des processeurs, des mémoires).

Le principal moteur de cette évolution réside dans la diminution régulière de la taille des motifs de dessin des circuits intégrés. Partis de quelques dizaines de microns dans les années 1960, ceux-ci sont maintenant de 22 nm fin 2012, et tout montre que cette évolution n'est pas terminée est certainement l'aventure technologique la plus fabuleuse de l'histoire humaine. L'ampleur des progrès réalisés dépasse de loin tout ce qui a été fait dans les autres domaines, y compris l'aviation et le spatial [1].

1.1 Circuits Intégré

Le circuit intégré (CI), aussi appelé puce électronique, est un composant électronique intégrant souvent plusieurs types de composants de base (résistances, condensateurs, diode, transistors) dans un volume réduit, reproduisant une ou plusieurs fonctions électroniques, cela va des fonctions les plus simples comme les portes logiques et les amplificateurs à des fonctions beaucoup plus complexes comme les microprocesseurs utilisés dans l'‘informatique.

La Figure I.1 montre un circuit intégré encapsulé avec les différentes parties le constituant et qui sont : le boitier, la puce, les plots d'entrée/sortie, et les fils de routage.

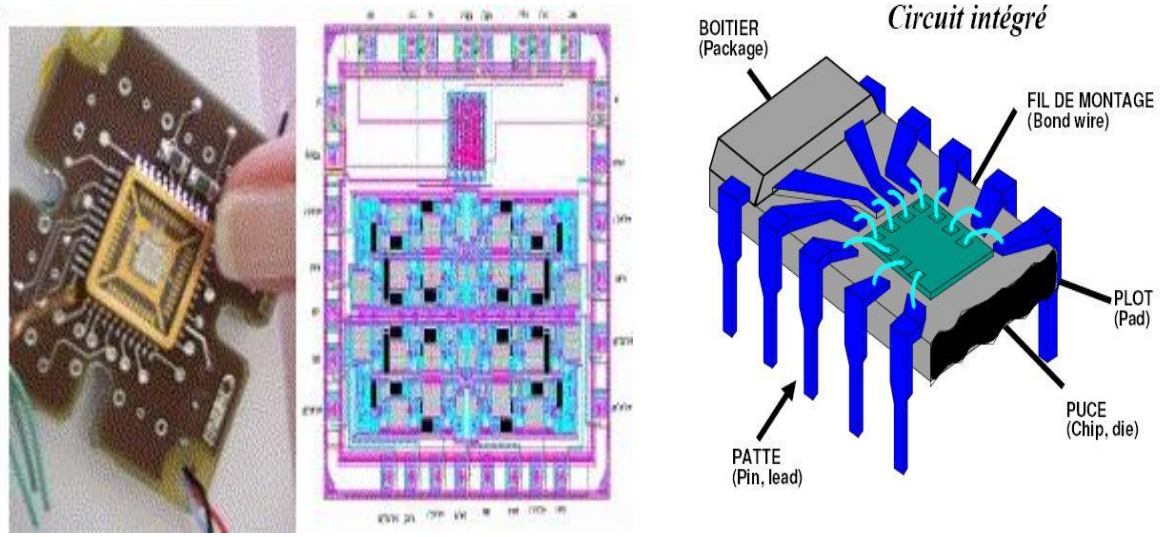


Figure I.1: Composantes des circuits intégrés.

1.2 Technologie De Fabrication

Les circuits intégrés utilisent deux types de composants actifs, appelés « transistors» :

- **Technologie CMOS :** La technologie CMOS (Complementary Metal Oxide Semiconductor) est une technologie planaire destinée au développement des systèmes à très haute échelle d'intégration (VLSI). Grâce aux propriétés des transistors CMOS, cette technologie permet de réaliser des circuits à faible coût et à basse consommation. Cet avantage lui a permis d'être reconnue comme la technologie de pointe la plus avancée et la plus maîtrisée dans le domaine de la micro-électronique.
- **Technologie Bipolaire :** La technologie bipolaire est une technologie planaire qui permet de réaliser des systèmes à très haute échelle d'intégration à partir de transistors bipolaires. Un circuit dans cette technologie peut incorporer des transistors npn, des transistors pnp, des diodes, des résistances et des éléments capacitifs. Les propriétés des transistors bipolaires font que les circuits électroniques réalisés dans cette technologie sont plus rapides comparés aux mêmes circuits réalisés dans une technologie CMOS. Cependant l'inconvénient majeur qu'ils présentent est leur forte consommation.
- **Technologie BiCMOS :** La technologie BiCMOS est apparue vers les années 1990. Comme elle utilise à la fois le procédé de fabrication d'une technologie bipolaire et le procédé d'une technologie CMOS, elle permet de réaliser sur le même circuit des transistors CMOS et des transistors bipolaires. Ainsi elle rassemble les avantages de ces deux technologies. En particulier, elle permet de réaliser des circuits rapides et à de faible consommation.

1.3 Aspect économique

L'augmentation régulière de la complexité des circuits intégrés entraine naturellement une augmentation du coût de leur conception. Le coût de revient d'un circuit intégré dépond du :

- Nombre de circuits fabriqués
- Coût de développement NRE « Non Recurring Engineering cost» résulte du :
    - Le temps de développement x Salaire x Cout outils de CAO

- Coût de fabrication RE « Recurringcost » qui dépend :
  - Surface du circuit.
  - Coût d'encapsulation.
  - Coût de test

Le cout de revient par circuit est donné par :

$$Cout\ de\ Revient = \frac{Cout\ NRE}{Nbr\ de\ Circuits} + Cout\ de\ Fabrication\ RE$$

1.4 Flot de conception

Un flot de conception est la combinaison explicite des outils de conception assistée par ordinateur (CAO) pour réaliser la conception d'un circuit intégré (voir figure I.2) :

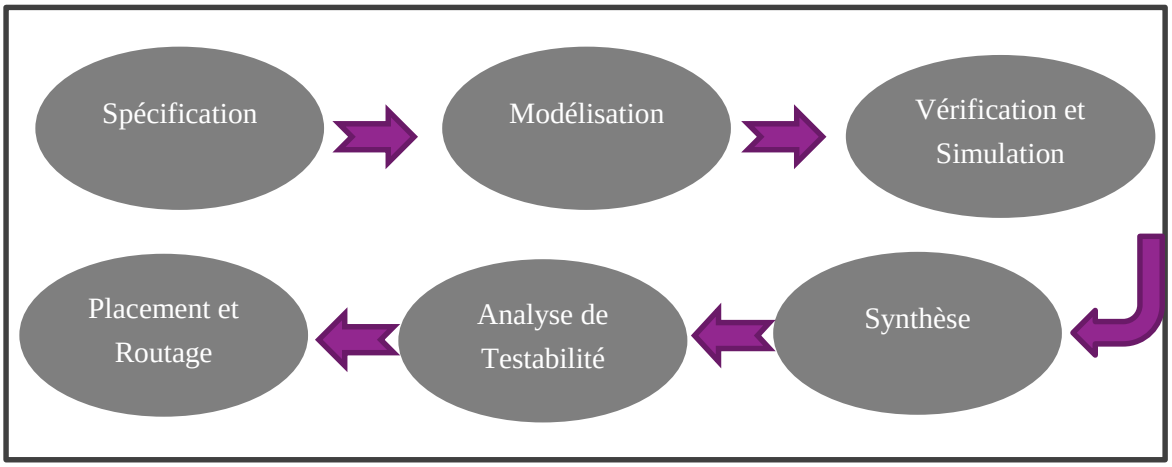


Figure I.2 : Flot de conception d'un circuit intégré.

1.4.1 Spécification

C'est une étape de création d'un modèle de référence pour le circuit à concevoir. Elle peut être présentée sous forme :

- Textuelle.
- Exécutable (Utilisation du langage VHDL ou Verilog).
- Formelle (Utilisation de modèles et de formules mathématiques pour décrire le circuit).

1.4.2 Modélisation

C'est une étape de description du circuit intégré suivant un niveau d'abstraction. Ce niveau peut être :

- Comportemental « Behaviour » (on utilise les expressions et les algorithmes.
- Structurel « structural » (on utilise les portes et les registres).
- Physique « layout » (on utilise les rectangles et les circuits).

1.4.3 Vérification et simulation

C'est une étape de validation du modèle par rapport à la spécification. Il existe plusieurs types de vérification :

- Simulation
- Vérification formelle
- Emulation

1.4.4 Synthèse

La Synthèse est le processus qui convertit un code HDL ou graphique en une netlist compatible avec la technologie cible [2]

1.4.5 Analyse de testabilité

L'analyse de testabilité est l'une des composantes de la sureté de fonctionnement et contribue à la maintenabilité et à la disponibilité d'une entité. Elle joue un rôle prépondérant dans la conception d'un système complexe et que sa participation doit se faire dans la phase aval de l'étude afin de déterminer les équipements qui satisferont aux mieux aux exigences de testabilité.

1.4.6 Placement et routage

Cette étape nous rapproche cette fois de ce que sera le circuit final. Il s'agit d'effectuer les deux opérations suivantes :

- le placement prend la liste des cellules de base qui font la fonction du circuit, et les répartit géographiquement sur l'empreinte donnée.
- le routage n'a plus alors qu'à tirer les fils pour interconnecter les entrées et les sorties des cellules.

2 Les circuits programmables

Il y a quelques années la réalisation d'un montage en électronique numérique impliquait l'utilisation d'un nombre important de circuits intégrés logiques. Ceci avait pour conséquences un prix de revient élevé, une mise en œuvre complexe et un circuit imprimé de taille importante. Le développement des mémoires utilisées en informatique fut à l'origine

des premiers circuits logiques programmables. Ce type de produit peut intégrer dans un seul circuit plusieurs fonctions logiques programmables par l'utilisateur. Sa mise en œuvre se fait très facilement à l'aide d'un programmateur, d'un micro-ordinateur et d'un logiciel adapté [3].

Un circuit logique programmable est un assemblage d'opérateurs combinatoires et de bascules dans lequel la fonction réalisée n'est pas fixée lors de la fabrication. Il contient potentiellement la possibilité de réaliser toute une classe de fonctions, plus ou moins large suivant son architecture. La programmation du circuit consiste à définir une fonction parmi toutes celles qui sont potentiellement réalisables. Comme dans toute réalisation en logique câblée, une fonction logique est définie par les interconnexions entre des opérateurs combinatoires et des séquentielles, et par les équations des opérateurs combinatoires.

2.1 Les avantages et les inconvénients

En nous intéressant aux circuits programmebleson a constaté qu'ils ont beaucoup d'avantages comme ils ont aussi leurs inconvénients. On va citer quelques-uns dans le tableau suivants :

Tableau I.1 : avantages et les inconvénients des FPGA

Avantages	**Inconvénients**
➢ simulation directe par logiciel sans maquette. ➢ Gain de place sur le circuit imprimé. ➢ Consommation, fiabilité et vitesse de fonctionnement plus intéressantes, car les connexions sont réduites au minimum dans un seul boîtier. ➢ Piratage presque impossible.	➢ Disposition d'outils informatiques puissants et conviviaux. ➢ Maîtrise des outils.

2.2 Les PLD (**P**rogrammable **L**ogic **D**evice)

Un circuit logique programmable PLD est un dispositif qui peut être configuré par l'utilisateur pour réaliser une fonction logique quelconque.il comprend les PAL et les GAL

- **PAL** (**P**rogrammable **A**rray **L**ogic) : circuits logiques programmables dans lesquels seuls les fonctions ET sont programmables, les fonctions OU ne le sont pas.
- **GAL** (**G**eneric **A**rray **L**ogic) : circuits logique programmables à technologie CMOS.

2.2.1 Structure de base d'un PLD

La plupart des circuits PLD suivent la structure suivante (figure I.3) :

- Un bloc d'entrées qui permet de fournir au bloc combinatoire l'état de chaque entrée et de son complément.
- Un ensemble d'opérateur « ET » sur lesquels viennent se connecter les variables d'entrées et leurs compléments.
- Un ensemble d'opérateurs « OU » sur lesquels les sorties des opérateurs « ET » sont connectées.
- Un bloc de sorties.
- Un bloc d'entrées/sorties, qui comporte une porte à 3 états et une broche d'entrée/sortie.

Le deuxième et le troisième ensemble forment chacun ce qu'on appelle une matrice. Les interconnexions de ces matrices doivent être programmables, et ceci est réalisé par des fusibles qui sont grillé lors de la programmation. Lorsqu'un PLD est vierge, toutes les connexions sont assurées. Le bloc de sortie est souvent appelé macro-cellule que l'on nomme OLMC (signifiant macro-cellule logique de sortie). Cette macro-cellule comporte :

- Une porte OU Exclusif et une bascule D.
- Des multiplexeurs qui permettent de définir différentes configuration et un dispositif de rebouclage sur la matrice ET.
- Des fusibles de programmation.

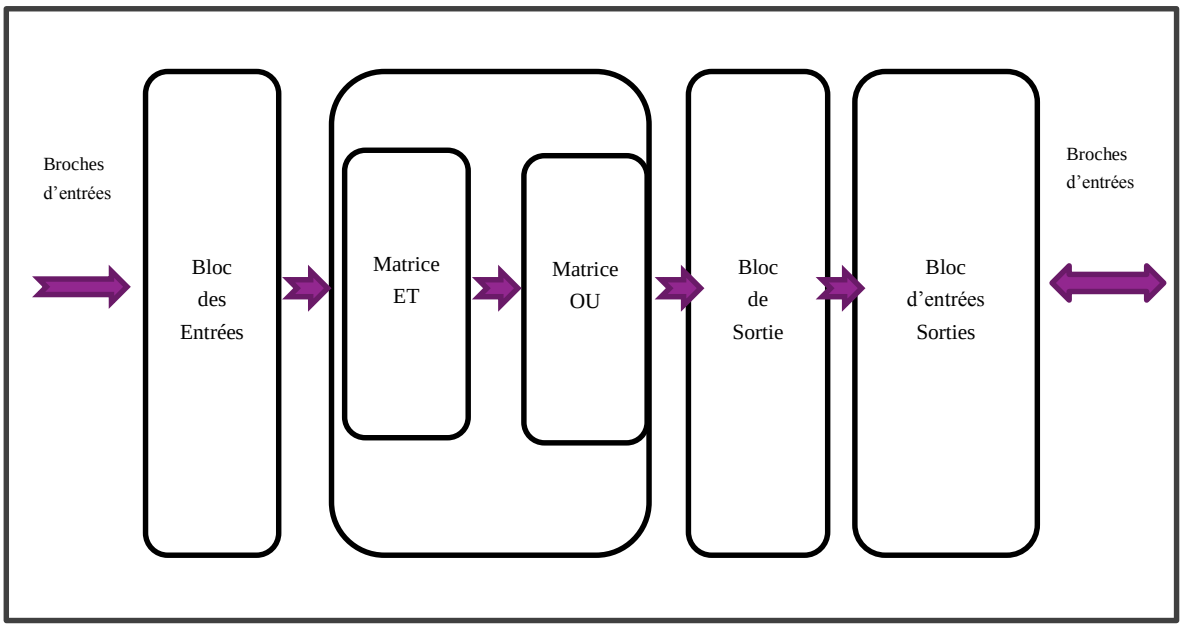


Figure I.3 : Structure d'un PLD.

2.3 Les CPLD (Complex Programmable Logic Device)

Les circuits CPLD ont une capacité en nombre de portes et en possibilités de configuration très supérieure à celle des PALs. Leurs architectures sont basées sur celles des PALs. Un CPLD c'est l'équivalent de plusieurs PALs mis dans le même circuit associé à une zone d'interconnexion. Le nombre de portes peut varier entre 100 et 100 000 portes logiques et entre 16 et 1000 bascules voir plus.Sa structure générale est illustrée dans la figure suivante [4].

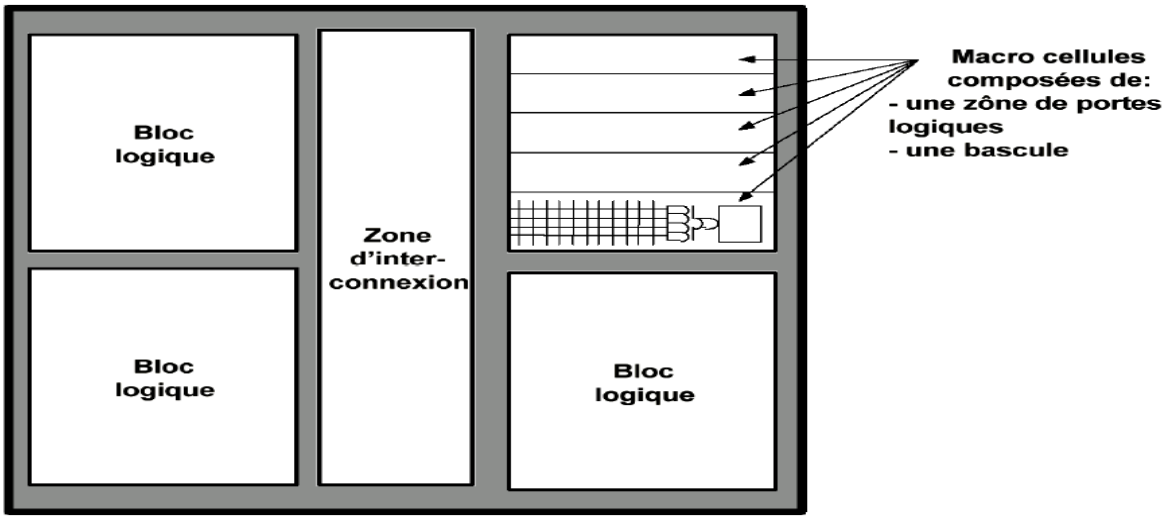


Figure I.4 : Architecture global d'un

2.4 Les FPGA (Field Programmable Gate Array)

Ces circuits sont en évolution des CPLD, Un FPGA est un composant VLSI à haute intégration (3 000 000 de ports en 2010) entièrement reconfigurable ce qui permet de le reprogrammer à volonté afin d'accélérer notablement certaines phases de calculs.

3 LES FPGA

Les FPGA sont des circuits pré-diffusés programmables. Contrairement aux circuits pré-diffusés conventionnels, on n'a pas à faire de fabrication spéciale en usine, ni système de développement coûteux.

3.1 Architecture

Un circuit FPGA contient un très grand nombre de macro cellules avec une très grande souplesse d'interconnexion entre eux. Dans le FPGA, le temps de propagation dans les couches logiques du circuit dépend de l'organisation et de la distance entre les macro-cellules interconnectées.

L'architecture, retenue par Xilinx (figure I.5) se présente sous forme de :

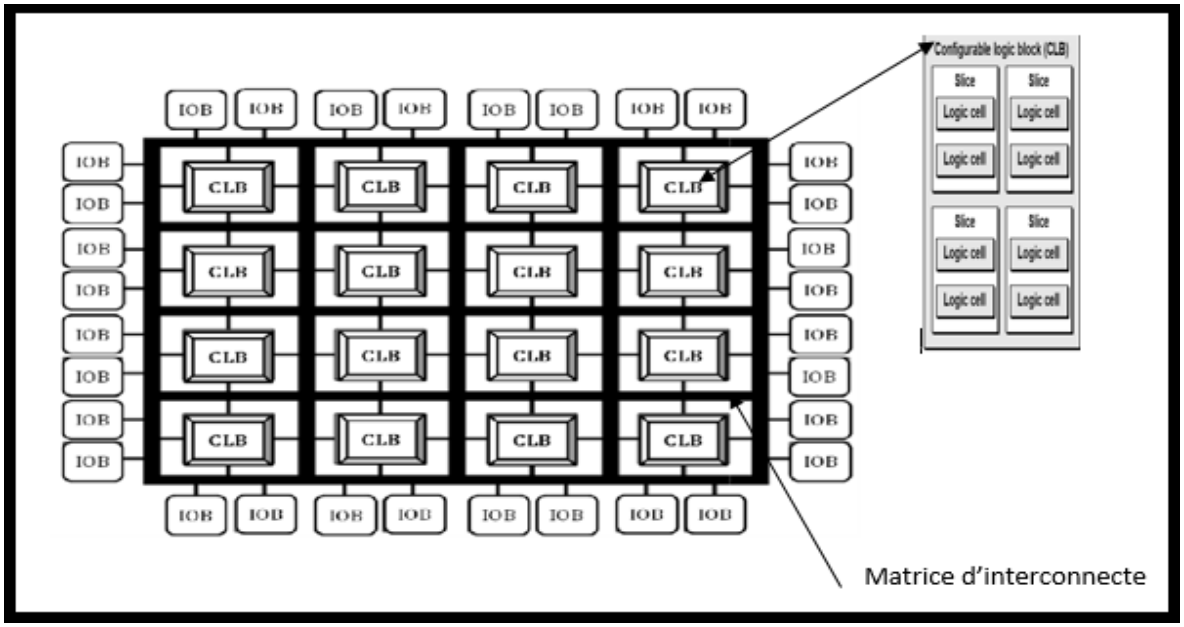


Figure I.5 : Architecture interne d'un FPGA.

- **Cellule de base :** ce sont des éléments de base d'un FPGA, avec lesquels on peut réaliser toute opération de logique. Ils contiennent des LUT qui sauvegardent la table de vérité de la fonction programmé, des générateurs de fonctions ou aussi des blocs de mémorisation.
- **Slice :** un slice est constitué de plusieurs cellules de base, qu'on va interconnecter pour avoir une fonction voulue.
- **CLB** (**C**onfiguration **L**ogic **B**loc) : c'est un ensemble de slice interconnecté pour obtenir une fonction déterminé.
- **IOB** (**I**nput **O**utput **B**loc) : les blocs IOB servent d'interface entre les broches du composant FPGA et la logique interne développée à l'intérieur du composant. Ils sont sur toute la périphérie du circuit intégré.
- **Matrice d'interconnexion :** elle est sous forme de grille de segments métalliques verticaux et horizontaux positionnée entre les lignes et les colonnes des CLB et IOB [5].

3.2 Technologie de programmation

Il existe plusieurs technologies de programmation des FPGA. Dans le but de réduire l'encombrement de la logique destinée à la programmation, et d'accélérer le fonctionnement

des circuits, on peut arriver à des circuits programmables une seule fois (One-Time Programmable ou OTP. Par contre, si la reprogrammabilité est importantes, on peut opter pour des solutions plus chère, mais reprogrammable à volonté. Il existe deux types majeurs de systèmes de programmation :

- **Les FPGA anti-fusibles :** cette technologie permet d'obtenir les circuits les plus compacts et les plus rapides. Ils sont bien entendu programmables une seule fois, ce qui les limite à une utilisation. Le point de connexion est de type ROM, c'est-à-dire que la modification des points est irréversible. Dans l'état initial, le fusible est présent et il n y a pas de contact, pour établir le contacte, il faut détruire le fusible. L'avantage de cette technologie est sa faible taille, et sa faible sensibilité aux radiations et autres perturbations environnementales, induisant une bonne fiabilité. Cependant elle présente l'inconvénient d'être configurable une seule fois [5].
- **Les FPGA à SRAM :** cette technologie permet d'avoir une reconfiguration rapide des FPGA, Les connexions sont réalisées en rendant les transistors passant. L'avantage de cette technologie est qu'elle permet une reconfiguration rapide au sein même du circuit. Le principal désavantage est la surface nécessaire pour la SRAM. Ils nécessitent l'utilisation d'une mémoire standard chargée à l'initialisation. Ils font appel à la technologie CMOS [5]-[6].

3.3 Flot de conception

Les techniques de conception CAO (Conception Assisté par Ordinateur) sont aujourd'hui largement employées afin de concevoir des circuits électroniques nécessaires à mettre en pratique les connaissances algorithmiques. Le flot de conception d'un système sur puce regroupe plusieurs niveaux d'abstraction. Dans chaque niveau le concepteur s'intéresse à la résolution d'un problème.

Le développement d'une application sur FPGA par les outils CAO, suit l'enchainement des étapes suivantes (figure I.6) : spécification du design, développement du design, la synthèse, le placement, intégration et implantation.

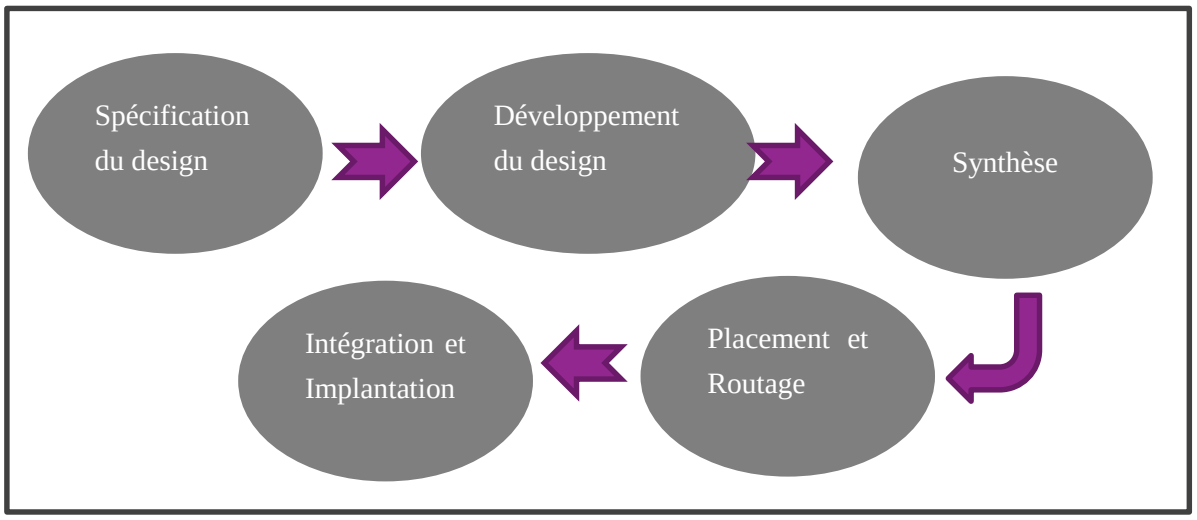


Figure I.6 : Étapes de développement d'une application sur FPGA par outils CAO.

3.3.1 Spécification du design

Cette étape se fait sur un outil de développement tel que le VHDL, VERILOG… Elle consiste à déterminer les nombres de broches et leur localisation sur le circuit FPGA, ainsi que la fréquence d'horloge du système et la mémoire requise pour l'application.

3.3.2 Développement du design

Une fois l'outil de développement choisi, il faut respecter certaines règles propres à ce dernier, pour que le code soit synthétisable. Vient alors la saisie de code et là on a deux choix, soit le graphique (machine a états) ou la saisie HDL (Hardware Description Language).Après ça, on passe à la simulation ensuite à la synthèse.

3.3.3 La synthèse

La synthèse et le processus qui convertit le code HDL ou le graphique pour avoir une représentation au niveau portes logique la mieux adaptée. Le résultat de cette étape est un fichier représentant le circuit électronique, appelé le fichier synthèse.

3.3.4 Placement et routage

A partir de fichier synthèse, l'outil de conception procède au placement et routage. Un algorithme de routage est sensé faire l'aiguillage des données qu'il reçoit vers leur destination. Ce qui est équivalent à définir les chemins qui relient l'ensemble de CLB contenus dans la fonction désirés, toute en essayant d'optimiser ces chemins au maximum.

3.3.5 Intégration et implantation

L'implémentation est la réalisation proprement dit, qui consiste à mettre en œuvre l'algorithme cible. C'est une étape de programmation physique et de tests électronique qui contrôlent la réalisation du circuit [5].

4 Principe de communication

Une liaison 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 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).

Dans ce chapitre nous avons donné une bref présentation sur la liaison série avec les deux types synchrone et asynchrone.

5 Parallèle ou série

5.1 Communication parallèle

La méthode la plus répandue, consiste à relier les deux équipements par un bus parallèle. Ce type de connexion à 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.

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.

5.2 Communication série

Pour interpréter un message binaire transmis en série, l'émetteur et le récepteur 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 regrouper en deux grandes classes : synchrones et asynchrones

5.2.1 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 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 au dispositif récepteur de se synchroniser. Un message doit commencer par un ou deux caractères spéciaux, dits « caractère de synchronisation »

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 de synchronisations 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.7.

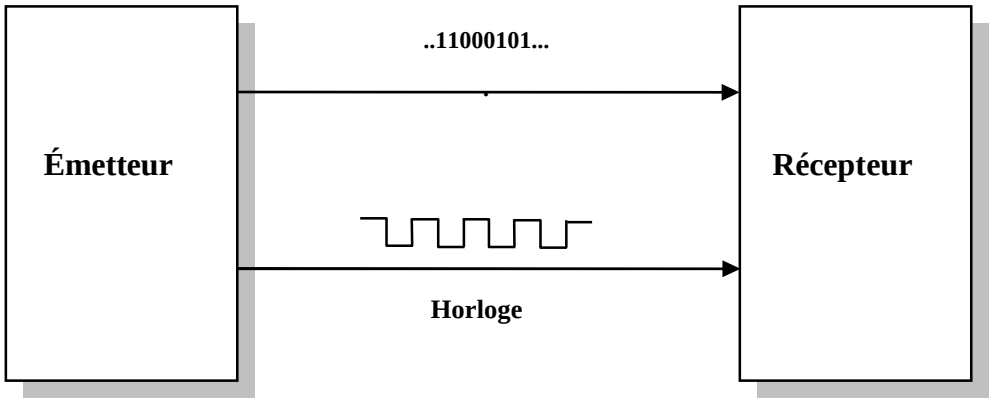


Figure I.7: Exemple d'une liaison série synchrone

5.2.2 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 se précède et se termine par caractère de synchronisation. Le schéma de la figure I.8 représente une liaison série asynchrone

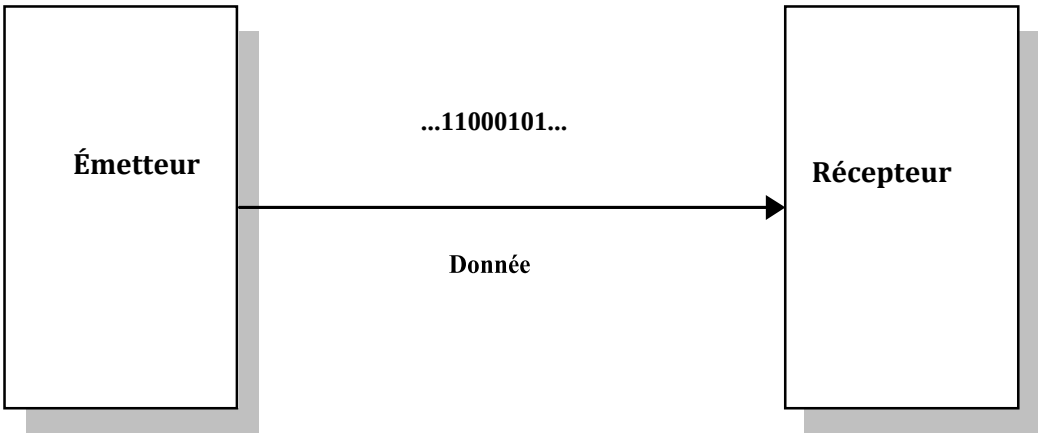


Figure I.8 : Liaison série asynchrone

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 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).

Le format d'une unité de données transmise est montré à la figure I.9

- 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 STOP.

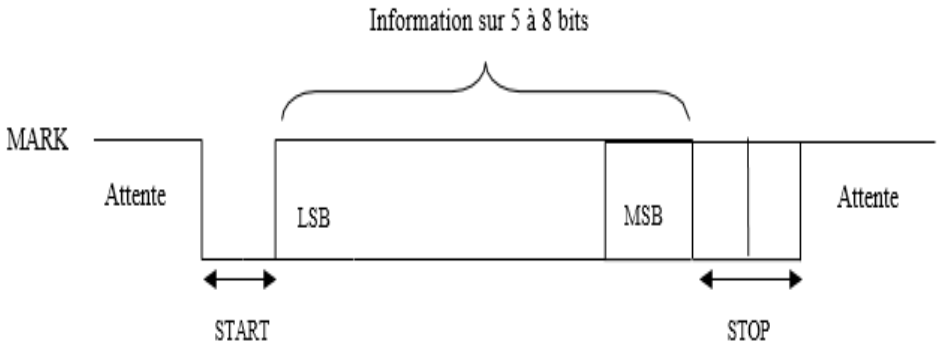


Figure I.9 : format d'une unité de données

6 Types ou modes de communication

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.

6.1 Communication 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. 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.10.

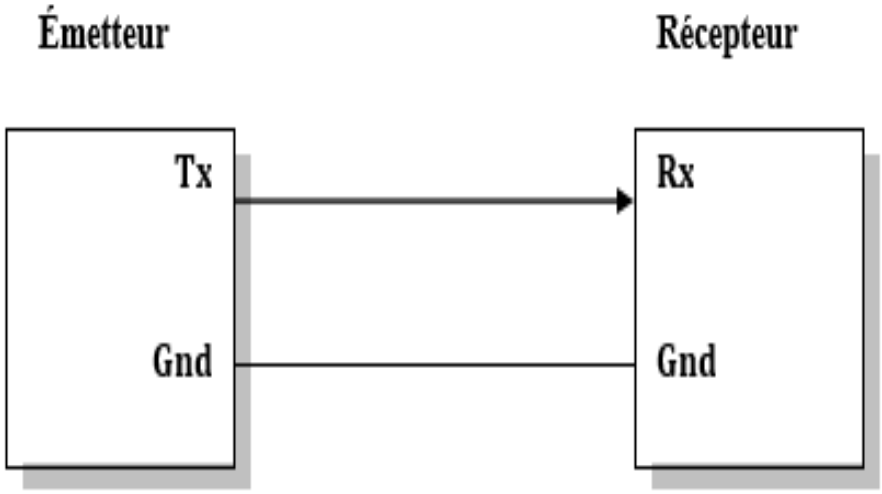


Figure I.10 : Exemple de liaison unidirectionnelle

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.

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.

Cette solution est retenue car elle simplifie la conception du système et en réduit les coûts.

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

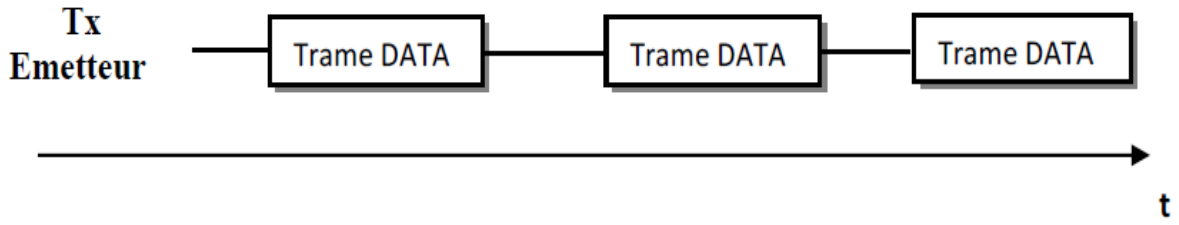


Figure I.11 : Chronogramme d'un exemple de liaison simplex

6.2 Communication 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 peu performante, fonctionnent encore uniquement dans ce mode comme le montre la figure I.12. [8]

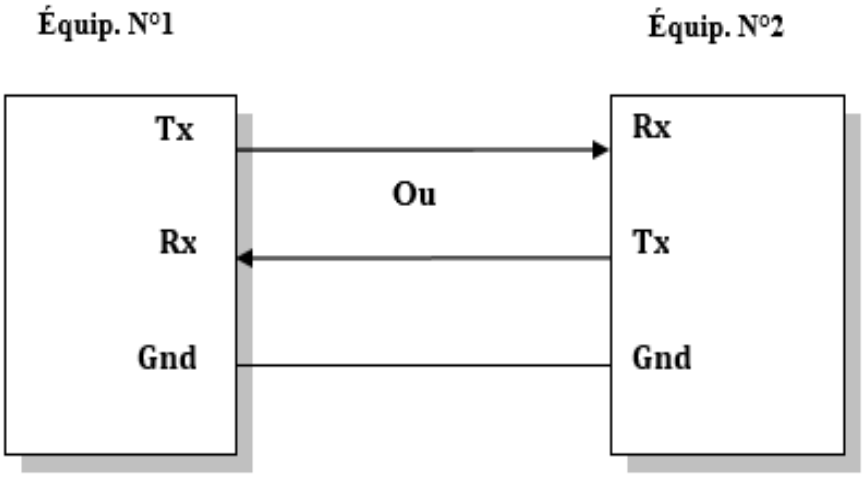


Figure I.12 : Exemple de liaison half duplex

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.

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.

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.

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.

La figure I.13 donne le chronogramme de ce mode de liaison, il montre bien la façon de fonctionnement de ce mode.

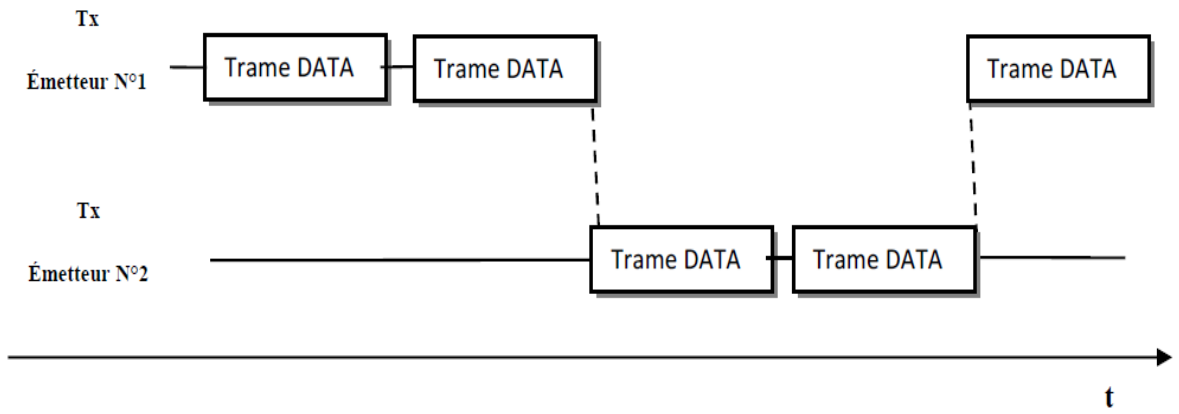


Figure I.13 : Chronogramme d'un exemple de liaison half duplex

6.3 Communication 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 [8].

Dans la figure I.14, nous avons donné un exemple de liaison full duplex tel que l'équipement N°1 représente DTE et N°2 représente DCE.

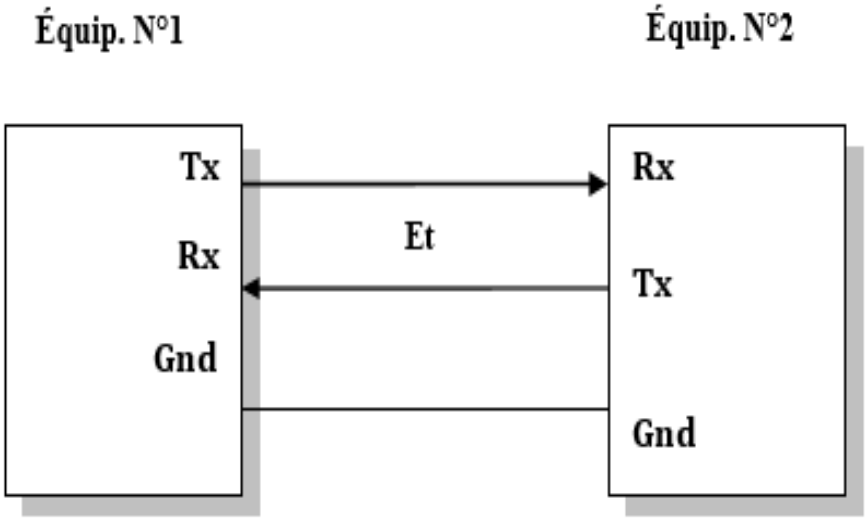


Figure I.14 : Exemple de liaison full duplex

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.

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

Dans ce cas l'émission de données indépendantes entre l'émetteur n°1 et le second est aléatoire comme le montre la figure I.15.

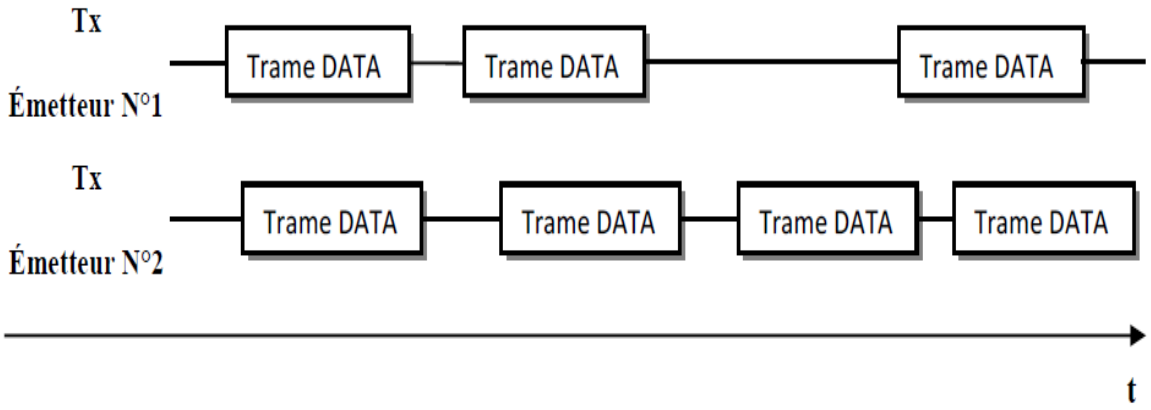


Figure I.15 : Chronogramme d'un exemple de liaison full duplex

7 CONCLUSION

Dans ce chapitre nous avons présenté les différents circuits programmables et conclure que la technologie FPGA s'inscrit au sommet de l'évolution des composants logiques programmables et ouvre de grandes perspectives en terme de coût et rapidité de conception.

Chapitre II

Le Langage VHDL pour la synthèse

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). [9]

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 EDA1. 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.

3 Règles d'écriture du langage VHDL

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

- **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.

- **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

[1] EDA : Electronic Design Automation

la «casse». Par exemple, on peut écrire : AND ou bien and pour l'opérateur de la fonction ET logique.

- **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.

- **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, de caractères numériques ou du caractère '_' souligné. 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.

- **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 logic 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#.

- **Les types**

Chaque entrée, sortie, signal, variable ou constante est associée à un type. Les types principaux sont : integer (de -231 à 231-1), bit, bit_ vector, std _logic, std _logic _ vector, booléen (true, false). Ainsi il n'est pas question d'effectuer une opération logique ou arithmétique entre deux grandeurs de types différents [10].

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.

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.

Toute description VHDL, utilisée pour la synthèse, a besoin de bibliothèques. L'IEEE 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 :

- 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 logic 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 : [10]

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

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 [11] :

```
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. [10]

- **Le nom du signal**

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.

- **Le sens du signal (ou le mode)**

- 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.

- **Le type**

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.

MOT_BUS : inout std_logic_vector (0 to 7); avec MOT_BUS (O) MSB et MOT_BUS (7) LSB [16].

4.3 Déclaration de l'architecture

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

Exemple : 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 :

- **FLOTS DE DONNEES**

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.

- **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
```

- **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/architecture 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;
```

Figure II.1 : Architecture Structurelle

4.4 Les instructions

4.4.1 Les instructions concurrentes

L'ordre d'écriture n'a alors pas d'importance, les opérations étant réalisées simultanément.

Comme montre la figure II.2 [12].

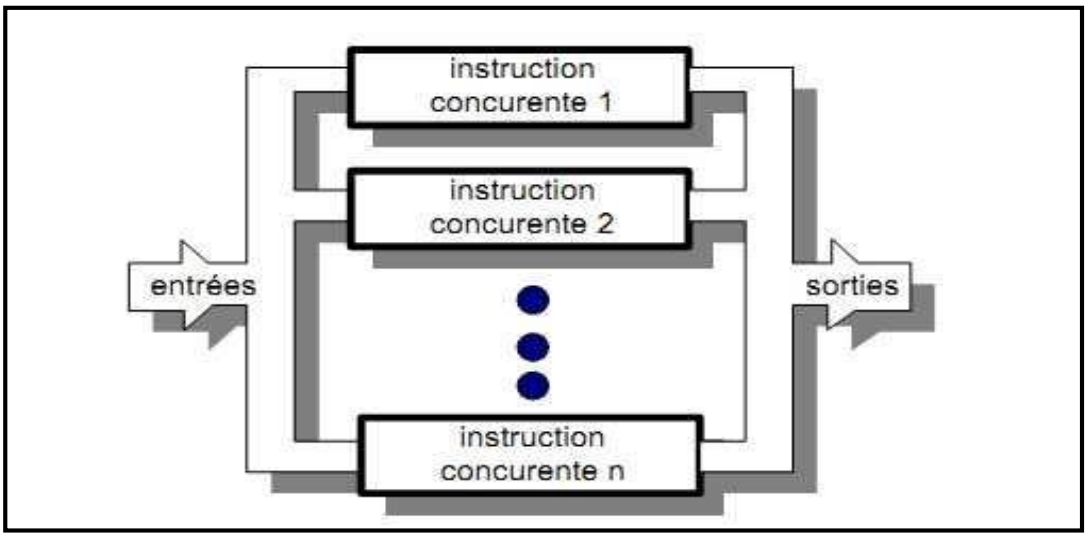


Figure II.2 : Instructions en mode concurrent

- **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 <=VALEUR _OU _NOM_D'UNE _GRANDEUR ;
```

Exemple : Compteur_de_temporisation <= "0000";

- **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.

- **Affectation sélective**

Cette instruction permet d'affecter différentes valeurs à un signal, selon les valeurs 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.

- **Le processus**

Avec les processus, une liste de sensibilité définit les signaux autorisant les actions. Ces 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 ;
```

Exemple : 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 ;
```

4.4.2 Les instructions séquentielles

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

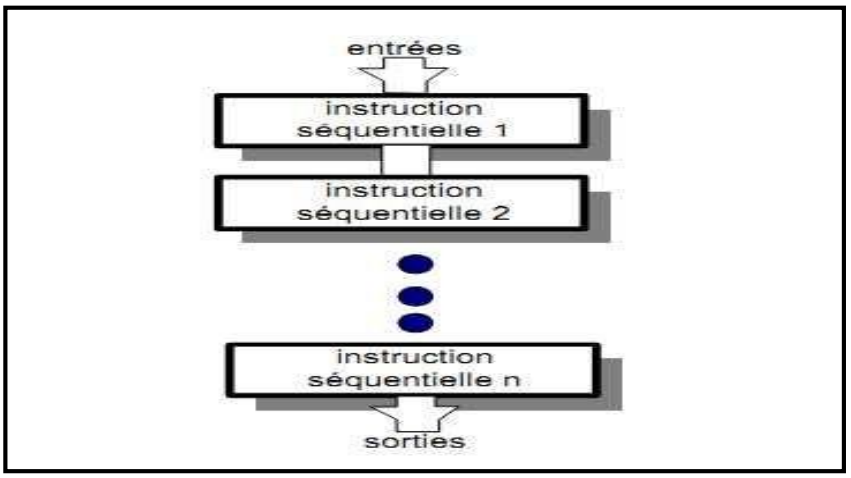


Figure II.3 : Instructions en mode séquentiel

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

- **L'affectation séquentielle**

Même syntaxe « <= » et même signification que pour une instruction concurrente ; pour les

variables, on utilise « : = ».

- **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.

- **Le test « case .. when .. end case»**

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

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

4.5 Déclaration de composant

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**

4.6 Instanciation de composant

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:

Dans la partie suivante, nous présentons les différents types des opérateurs que l'on rencontre dans le VHDL [11].

```
or_ ent_1: or_entity
Port map (
input_1 => input_1_ sig;
input_2 =>input_2_sig;
output => output_sig);
```

4.7 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 [13].

- **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.

- **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.

- **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.

- **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.

- **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.

5 Machine d’état

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

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

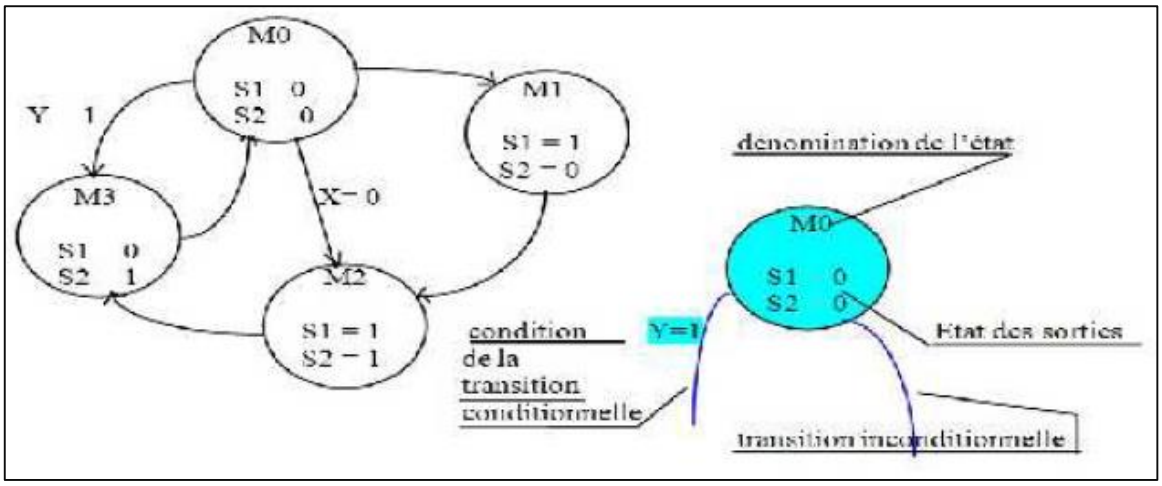


Figure II.4 : La description du système à 4 états

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.

5.2 Machine de Mealy

Voici la machine de Mealy répondant sur figure II.4

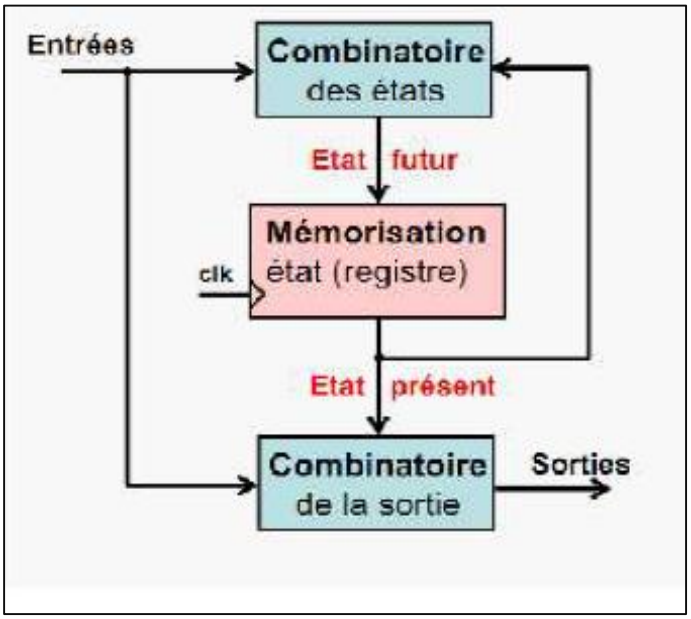


Figure II.5 : Machine d'état de Mealy

- 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 (c-à-d 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.

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 II.5 Représente la machine de Moore

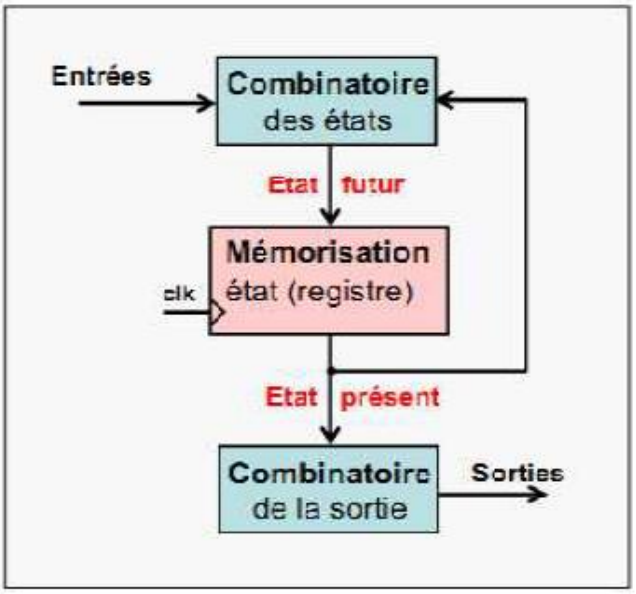


Figure II.6 : Machin d'état de Moore

Exemple : Le graphe d'état selon Moore.

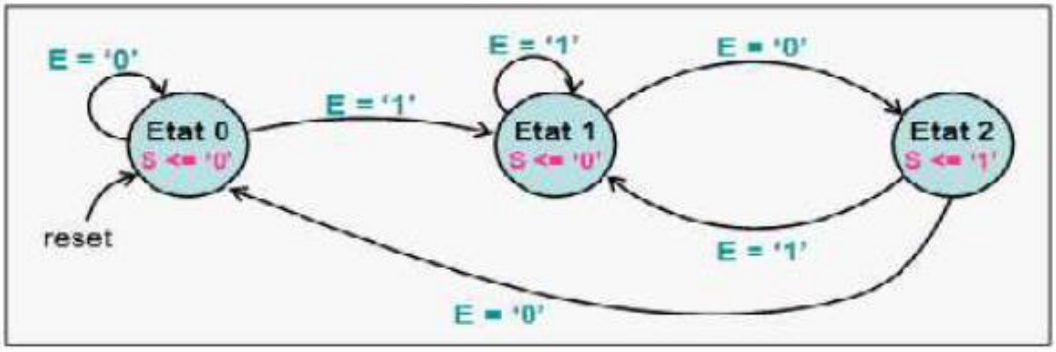
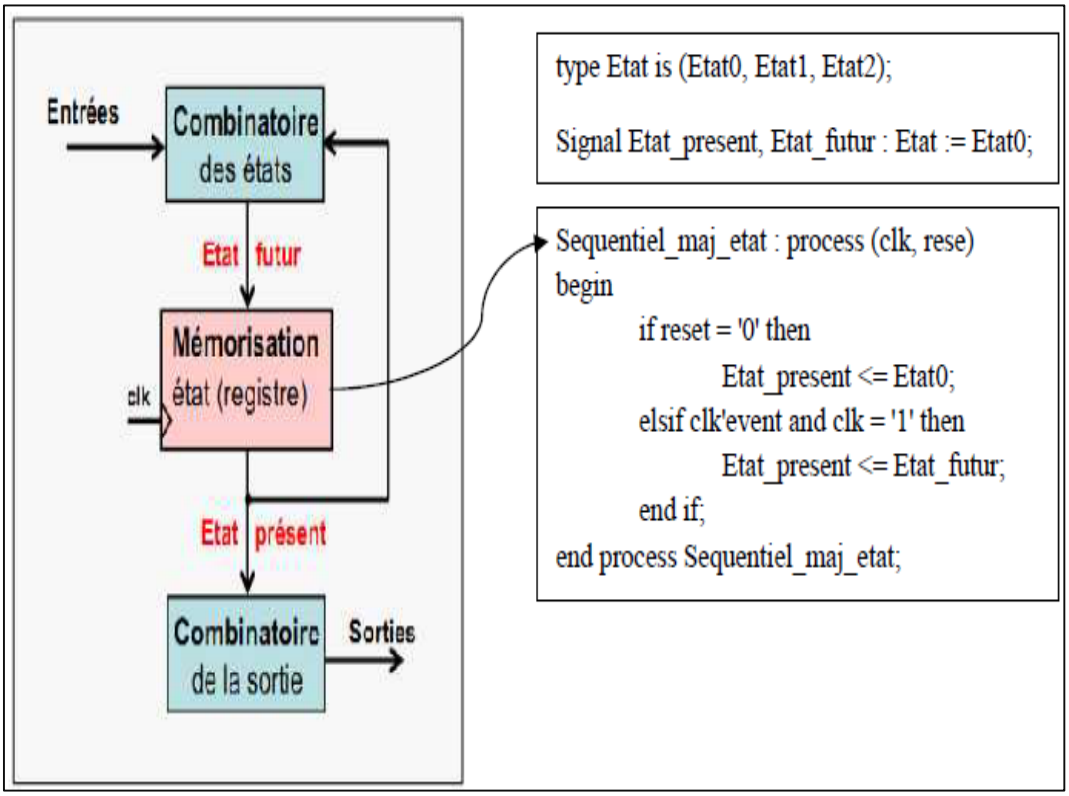


Figure II.7 : Le graphe d'état selon Moore

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

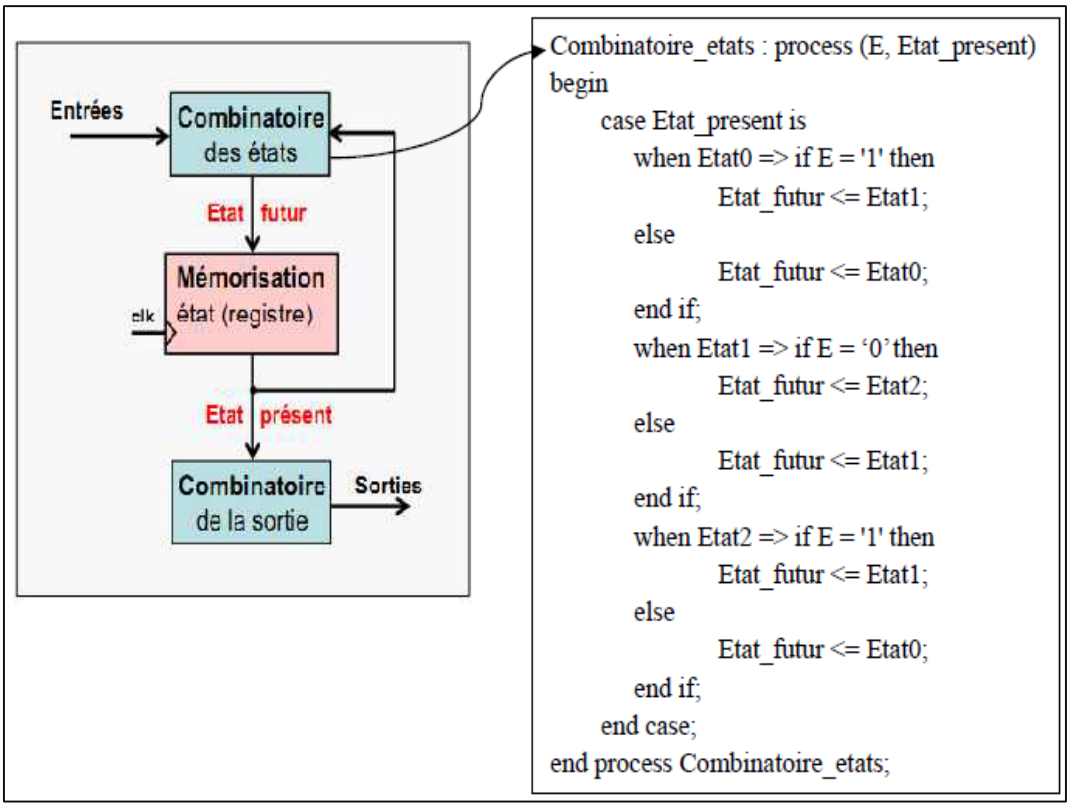



```
type Etat is (Etat0, Etat1, Etat2);

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

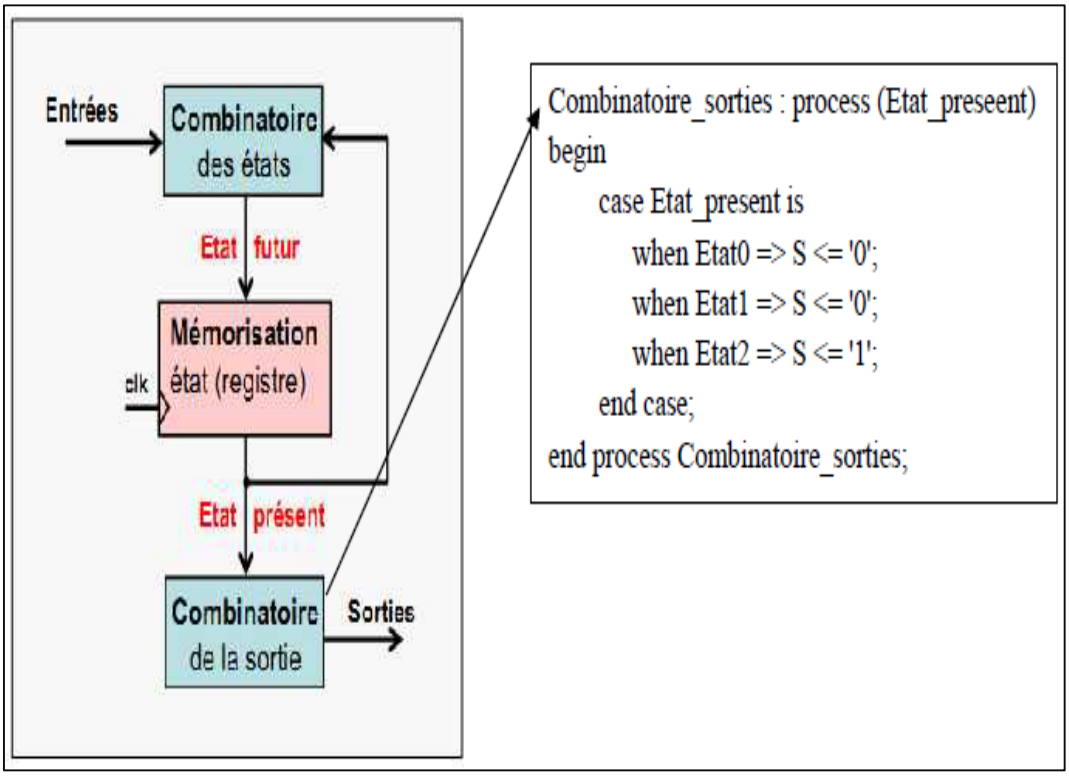
```
Sequentiel_maj_etat : process (clk, rese)
begin
        if reset = '0' then
                Etat_present <= Etat0;
        elsif clk'event and clk = '1' then
                Etat_present <= Etat_futur;
        end if;
end process Sequentiel_maj_etat;
```

- 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 :



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 système de transmission qui fait l'objet de ce travail

Chapitre III

Système de transmission

1 Introduction

Dans le chapitre I nous avons décrit les circuits programmables et es différente liaisons, En se basant sur ces derniers on va d'abord débuter par la présentation de notre protocole, Par la suite on va programmer notre système en utilisant le langage VHDL cité dans le chapitre II et faire la simulation et on passera à l'environnement de développement ISE afin de l'implémenter et de le tester sur la carte de développement Virtex II.

2 Description de la carte de développement Virtex II

La figure III.1 montre la carte de développement FPGA Virtex II avec les principaux composants le constituant qui sont :

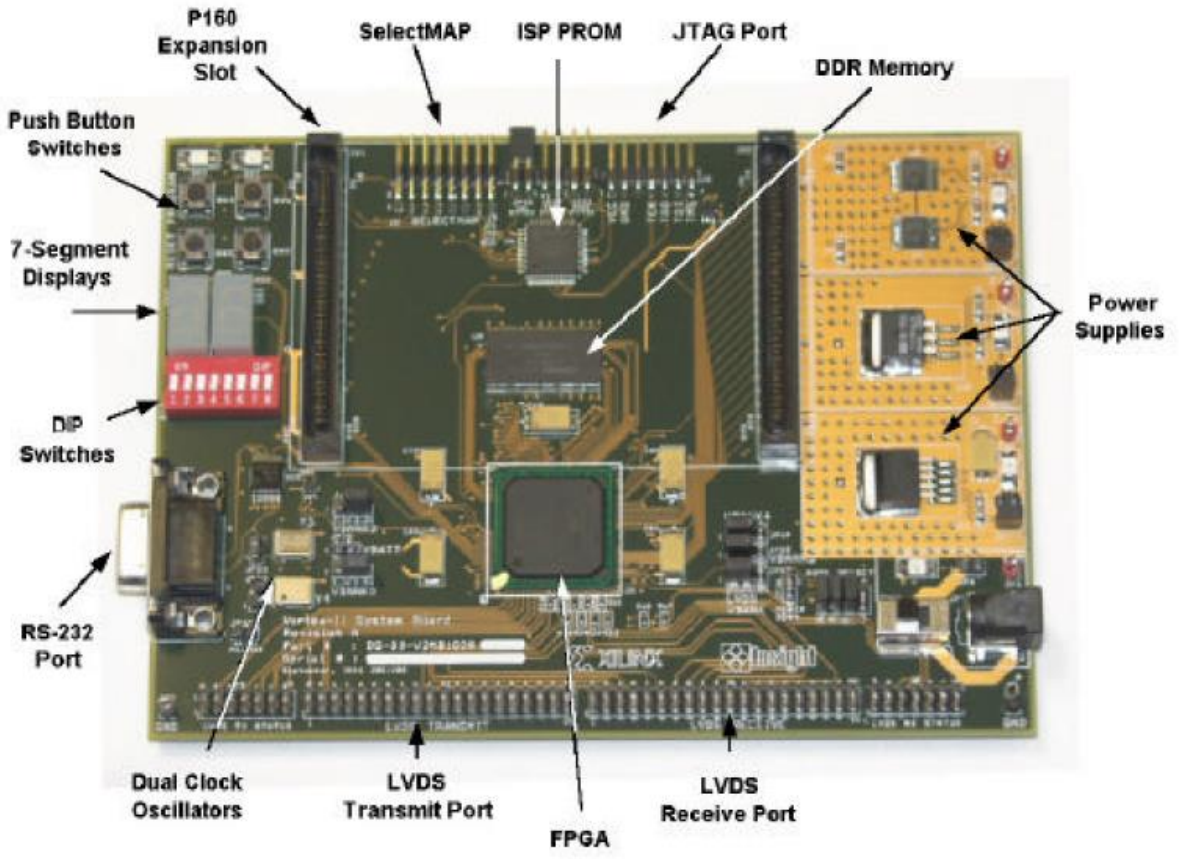


Figure III.1 : Carte de développement Virtex II.

- Un circuit FPGA XC2V1000-4FG456C d'une intégration d'un million de porte logiques.

- La carte de développement Virtex-II fournit 8 entrées de commutation pour l'utilisateur. Ces commutateurs sont statiques, mis à un niveau bas ou niveau haut logique.
- La carte de développement Virtex-II dispose de deux entrées de commutation avec le FPGA Virtex-II qui sont des boutons poussoirs. Chaque commutateur à bouton poussoir peut être utilisé pour générer un signal actif au niveau bas.
- 01 Port de connexion RS-232.
- La carte de développement Virtex-II fournit 32 Mo de mémoire DDR. Cette mémoire est mise en œuvre à l'aide du dispositif DDR Micron MT46V16M16TG-75 16Mx16.
- La carte de développement Virtex-II dispose de deux oscillateurs fonctionnant à 100 Mhz (CLK.CAN2) et 24 Mhz (CLK.CAN1). L'oscillateur fonctionnant à 100 Mhz est activé lorsque le cavalier JP24 est ouvert et désactivé lorsque JP24 est fermé. JP23 commande l'oscillateur 24 Mhz. Une troisième prise de l'horloge est fournie pour l'utilisateur.
- La carte de développement Virtex-II utilise deux afficheur 7 segments LED, à cathode commune qui peuvent être utilisés pendant l'essai et la phase de mise au point d'un dessin. L'utilisateur peut activer un segment donné par l'envoi d'un signal haut.
- La carte de développement Virtex-II fournit une seule LED pour l'utilisateur qui est active pour un signal haut. Elle est reliée au Pin A9 du FPGA.
- La carte de développement Virtex-II fournit un connecteur JTAG qui peut être utilisé pour programmer la PROM de la carte et configurer le FPGA Virtex-II. Deux options de connexion sont fournies, J2 est utilisé pour la connexion standard JTAG, et JP29 est utilisé pour la connexion Xilinx parallèle IV avec câble JTAG.
- Port d'expansion pour connecter une carte esclave si on veut augmenter les ressources.
- Un bloc d'alimentation pour les différents composants.
- Des dizaines de jumpers pour commander les différents composants. En suivant la documentation fournie par le constructeur on modifie les jumpers selon nos besoins.
- Des outils arithmétiques intégrés comme les multiplieurs, les multiplexeurs et les registres [12].

❖ Utilisation de la carte

La carte de développement Virtex-II prend en charge plusieurs méthodes de configuration du FPGA Virtex-II. Le port JTAG sur la carte de développement FPGA Virtex-II peut être utilisé pour configurer directement le FPGA Virtex-II, ou de programmer le XC18V04 FAI PROM.

Une fois la PROM programmée, elle peut être utilisée pour configurer le FPGA Virtex-II. Le port série SelectMap /esclave sur cette carte de développement peut également être utilisé pour configurer le FPGA Virtex-II.

La figure III.2 ci-après montre le branchement pour tous les modes de configuration FPGA Virtex-II qui sont pris en charge sur La carte de développement Virtex-II.

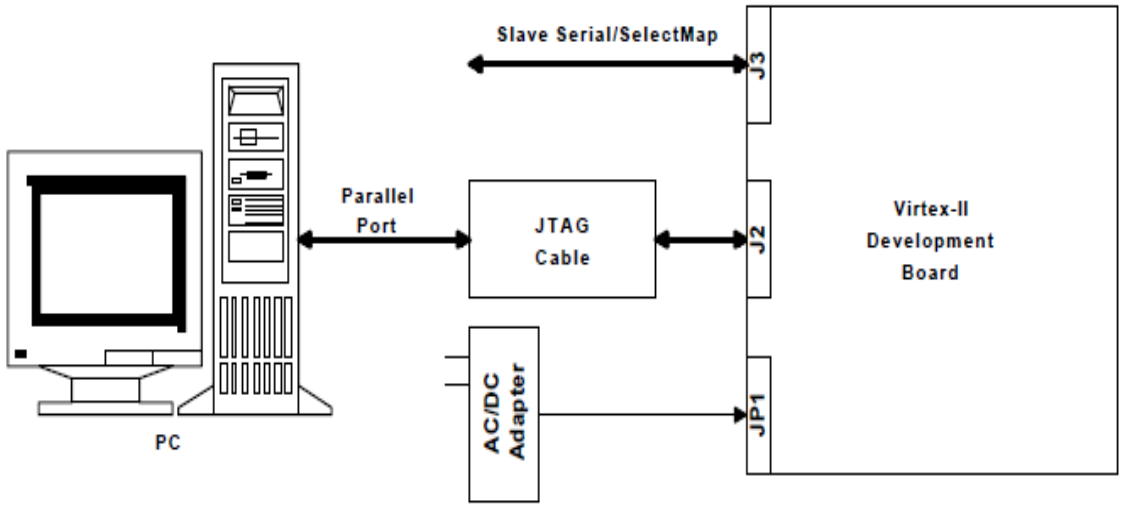


Figure III.2 : Branchement des modes de configuration

Le connecteur J2 JTAG sur la carte de développement FPGA Virtex-II peut être utilisé pour configurer le Virtex-II ou de programmer le XC18V04 FAI PROM. Le câble JTAG est connecté à la carte de développement Virtex-II via J2 à une extrémité et au port parallèle d'un PC à l'autre extrémité.

Lorsque le port JTAG est utilisé pour programmer le FAI PROM, effectuer les étapes suivantes :

- En utilisant le tableau (III.1) régler le mode de configuration du FPGA Virtex-II en Mode Master Serial ou Master SelectMap.
- Utiliser l'utilitaire de programmation JTAG Xilinx (IMPACT) pour charger le fichier de conception sur la PROM, associer le FPGA soit à un fichier fictif .bits ou à un fichier .bsd. permettant au logiciel de programmation JTAG de transmettre les données à l'FPGA [5].

Tableau III.1 : configuration du mode sélectionné

Mode	PC Pull-up	J1			
		1-2 (M0)	3-4 (M1)	5-6 (M2)	7-8 (M3)
Master Serial	No	Closed	Closed	Closed	Closed
Master Serial	Yes	Closed	Closed	Closed	Open
Slave Serial	No	Open	Open	Open	Closed
Slave Serial	Yes	Open	Open	Open	Open
Master SelectMap	No	Closed	Open	Open	Closed
Master SelectMap	Yes	Closed	Open	Open	Open
Slave SelectMap	No	Open	Open	Closed	Closed
Slave SelectMap	Yes	Open	Open	Closed	Open
JTAG	No	Open	Closed	Open	Closed
JTAG	Yes	Open	Closed	Open	Open

3 Principe de la liaison

Dans la liaison série les bits d'informations sont transmis les uns derrières les autres sur un seul et même fils. Les mots transmis sont les plus souvent de 5 à 8 bits.

3.1 Format de la donnée

A chaque extrémité de la ligne de communication, les deux équipements sont généralement dans ce que l'on appelle un état de repos vis-à-vis de la transmission ou de la réception de caractère.

Dans cet état de repos, on transmet une tension continue équivalente à un « 1 » logique, c'est un procédé de sécurité. La transmission d'un caractère est toujours signalée par un bit de départ et un autre bit de fin de transmission.

Le séquencement de la transmission est organisé de telle façon que le bit de poids le plus faible (LSB) est transmis le premier, suivi des bits de poids croissants.

4 Présentation de notre protocole

Notre protocole consiste à réaliser un système de transmission qui est composé de deux machines d'états (FSM) ; une pour la transmission et une autre pour la réception, afin de concevoir un système qui nous permet d'envoyer une donnée parallèle (de la machine d'état de transmission) sous forme série (bit par bit) et la recevoir (machine d'état de réception) sous sa forme originale.

4.1 Structure du système de transmission

Le schéma bloc de système est donné par la figure III.3. Il contient habituellement les composants fondamentaux suivant :

- Une horloge
- Un module d'émission (transmitter)
- Un module de réception (Receiver)

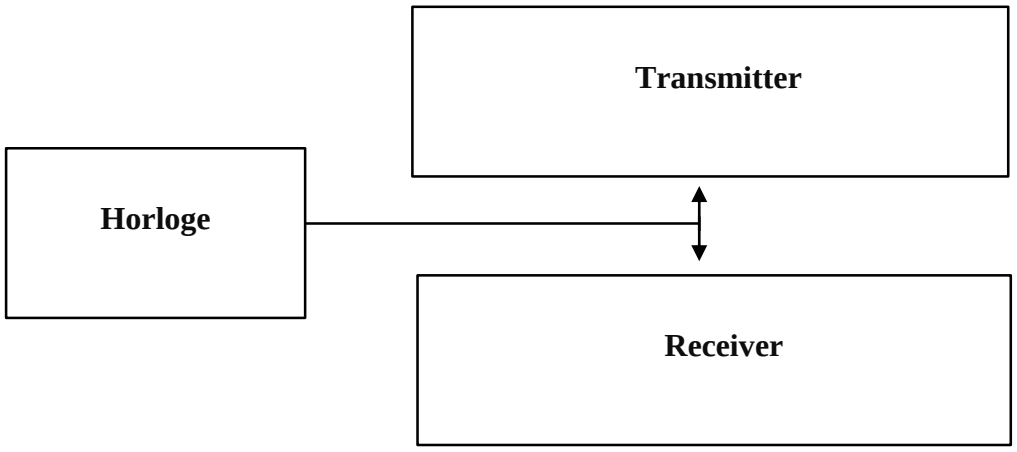


Figure III.3 : schéma d'un système de transmission de données

➢ L'horloge

Un signal d'horloge est un signal électrique oscillant qui rythme les actions d'un circuit. Sa période est appelée cycle d'horloge

À chaque cycle d'horloge, des calculs peuvent être effectués et les données seront validées au cycle d'horloge suivant

➢ Module d'émission (Transmitter)

Le module possède un bus d'entrée « DATA_IN », sur lequel le bus de donnée de l'application est connecté et une sortie « BIT_OUT » qui assure la transmission série de la donnée.

Le module en question exige un certain contrôle de la part de l'application pour indiquer la présence d'un résultat. Pour ce faire, la trame doit commencer par un bit de START (début) et se termine par un bit de STOP (fin) afin de respecter le protocole de transmission.

Le schéma synoptique de ce module est montré sur la figue III.4.

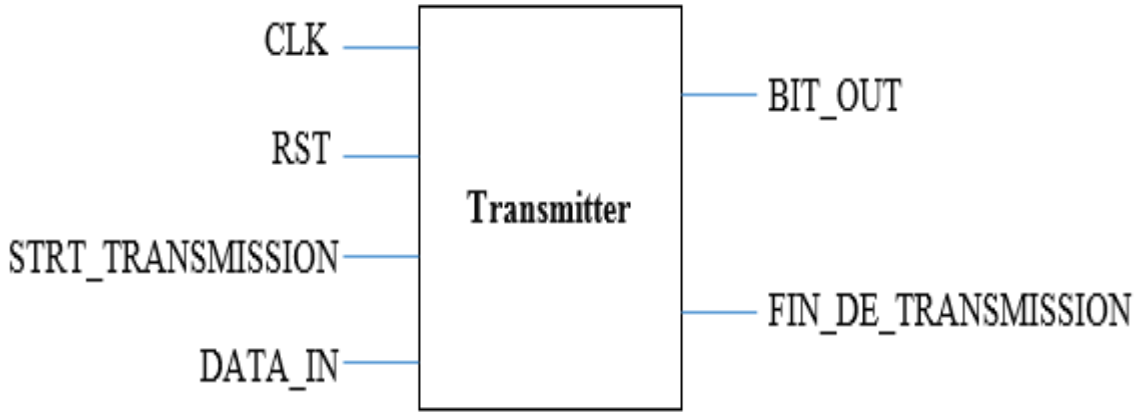


Figure III.4 : schéma du module de transmission de données

- **Module de Réception (Receiver)**

Ce module de Réception possède un bus de sortie « DATA_OUT » sur lequel le bus de donnée est connecté sur l'entrée « BIT_IN » qui assure la réception de cette dernière fournit sur la sortie « BIT_OUT » du module de transmission.

Le schéma synoptique de ce module est montré sur la figue III.5.

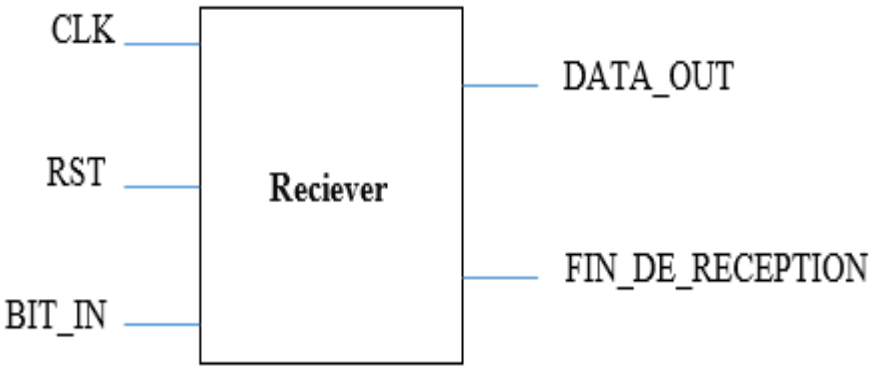


Figure III.5 : schéma du module de réception de données

5 Description de notre protocole

Pour mieux comprendre le fonctionnement de ce système on a utilisé les diagrammes d'états de Moore cités en chapitre II.

La description de ces diagrammes se fait par un nombre fini d'état comme la montre les figures III.6 et III.7 .

5.1 Machine d'état de transmission

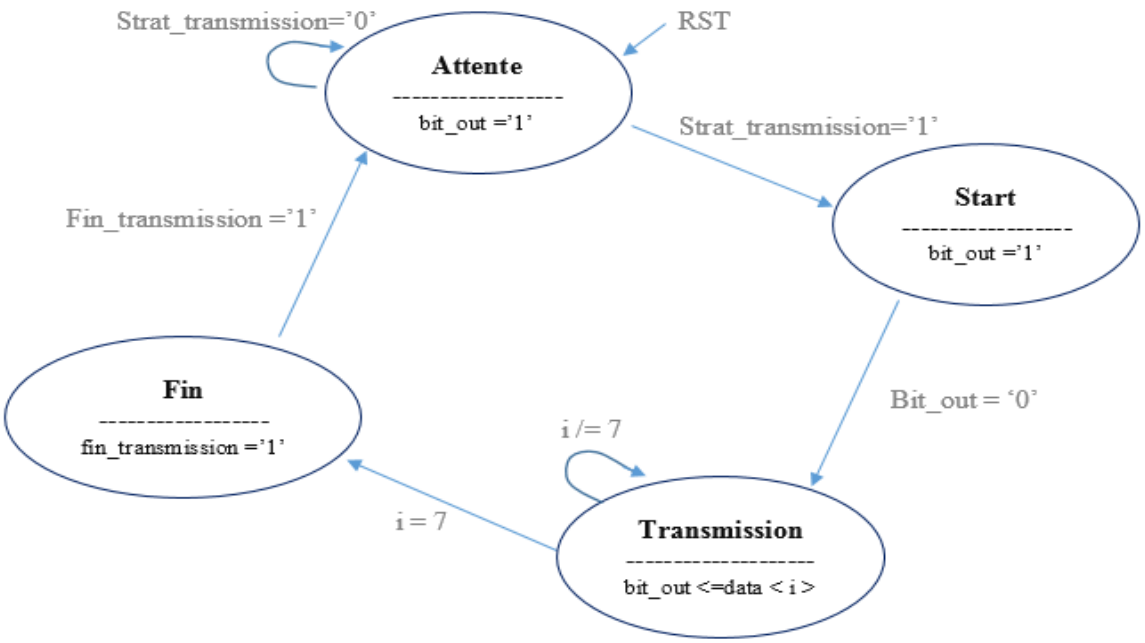


Figure III.6 : Machine d'état de transmission

Au début, tous les signaux sont initialisés. Si le signal « *STRAT_TRANSMISSION* » est à « *0* » la machine est à l'état d'attente (repos). Si ce dernier est remis à « *1* » la machine passe à l'état *START*, elle attend la remise à « 0 » du bit « *BIT_OUT* » pour commencer le transfert de la donnée.

Une fois ce bit est à « *0* », la machine est à l'état de *TRANSMISSION*, le transfert de la donnée commence.

Dans cet état le bit « *BIT_OUT* » reçois les bits un par un du bus « *DATA* » et les envois au « *BIT_IN* »se trouvant à l'autre côté de la machine de réception.

Arrivé à « *i = 7* » ce qui fait que la donnée est complétement transférée, la machine passe à l'état *FIN* et le signal « *FIN_TRANSMISSION* » est détecté.la machine de transmission revient son état initial (repos).

5.2 Machine d'état de réception

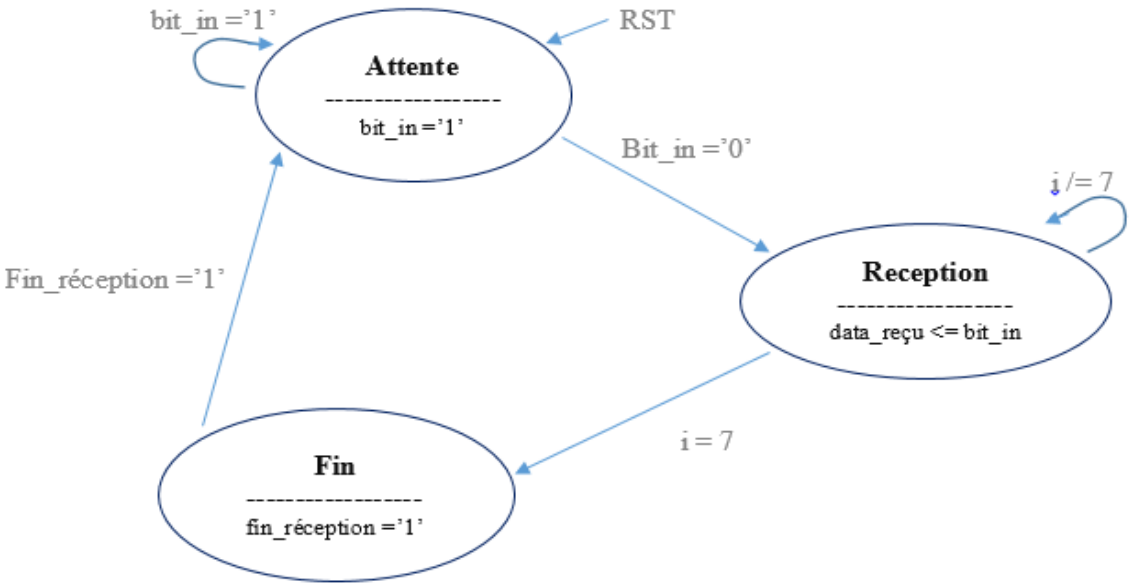


Figure III.7 : Machine d'état de réception

Au début la machine de réception est à l'état d'attente (repos).

Une fois le signal « *BIT_IN* » reçois un « *0* » de la part du signal « *BIT_OUT* » ce qui veut dire que la machine va commencer la réception de la donnée.

Dans cet état de réception que le bus « *DATA_REÇU* » reçoit la donnée via « *BIT_IN* » provenant du « *BIT_OUT* » de la machine d'état de transmission.

Arrivé à « *i = 7* » la machine d'état de réception à totalement reçu la donnée, c'est dans cette étape qu'elle passe à l'état de fin de réception.

Une fois la machine est à l'état *FIN*, le signal « *FIN_DE_RECEPTION* » est activé et la machine revient à son état d'attente (initial).

6 Description du navigateur de projet ISE

Dans ce qui suit nous allons présenter l'implémentation de notre protocole matériel décrit avec le langage VHDL en utilisant le navigateur de projet ISE.

Nous commençons par une brève description de navigateur de projet ISE, par la suite, nous présentons l'algorithme de notre protocole sous VHDL et testeront sa fonctionnalité par la simulation sur la carte de développement Virtex II

L'ISE est un environnement intégré de développement de système numérique ayant pour le but une implémentation matériel sur FPGA ou CPLD de la compagnie XILINX.

ISE intégré dans différents outils permettant de passer à travers tous les de conception d'un système numérique :

- Un éditeur de texte, de schémas et diagramme d'états.
- Un compilateur VHDL / Verilog.
- Un outil de simulation.
- Des outils pour la synthèse.
- Des outils pour l'implémentation sur PFGA.

7 Les étapes de simulation

7.1 Compilation

L'implémentation se fait à l'aide de l'environnement de développement du projet ISE. Dans ce dernier on va introduire le code de notre protocole sous VHDL comme le montre la figure suivante.

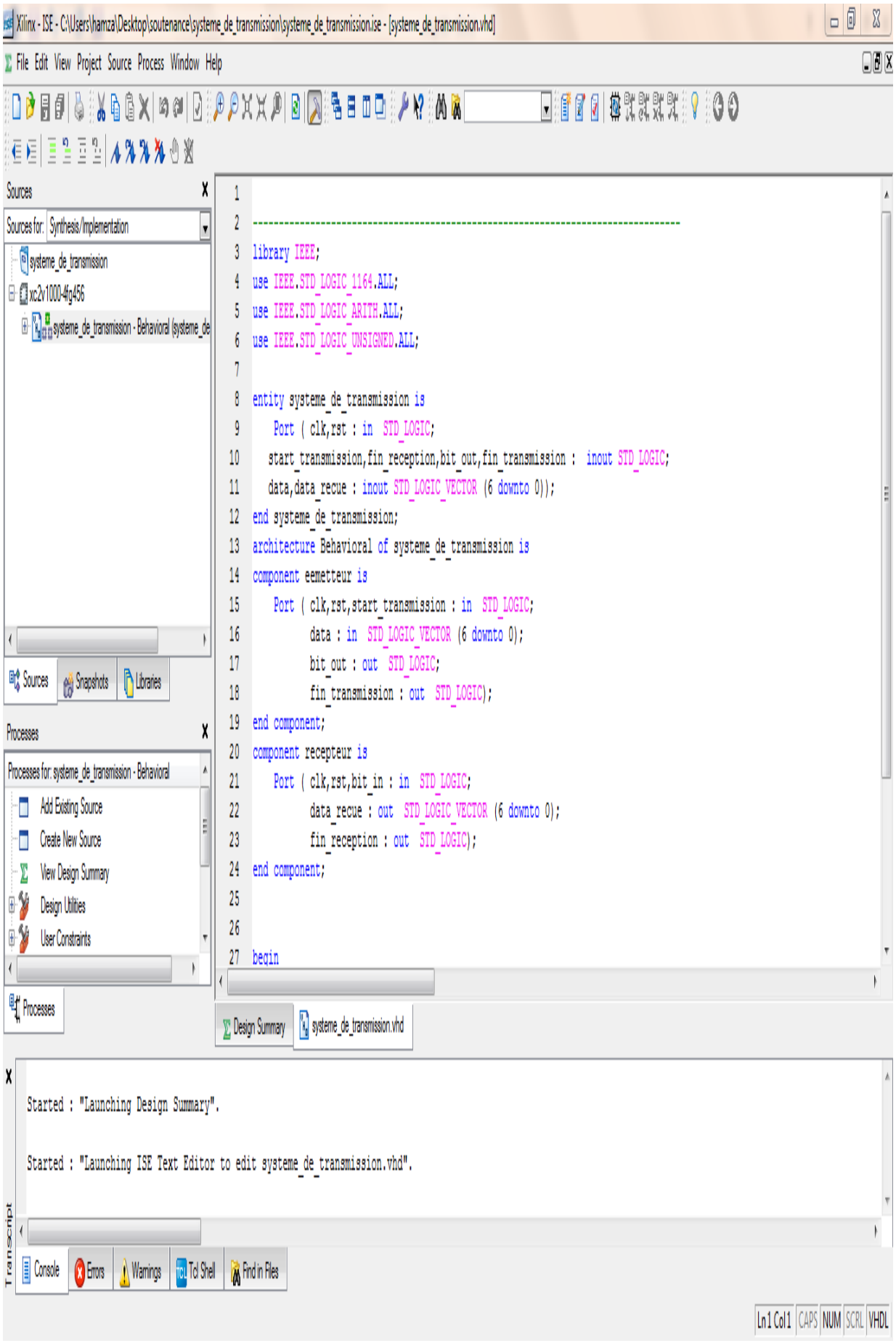


Figure III.8 : Notre protocole sous ISE

Une fois que le fichier VHDL édité, il est impératif de vérifier la syntaxe du design afin de corriger les erreurs éventuelles.

Pour cela on double clique sur le process check Syntax. Si une erreur est détectée une croix rouge apparait à côté de l'onglet check syntax et un message en bas de l'écran indique la source de l'erreur si non, c'est un croché vert qui apparait (voir la figure suivante)

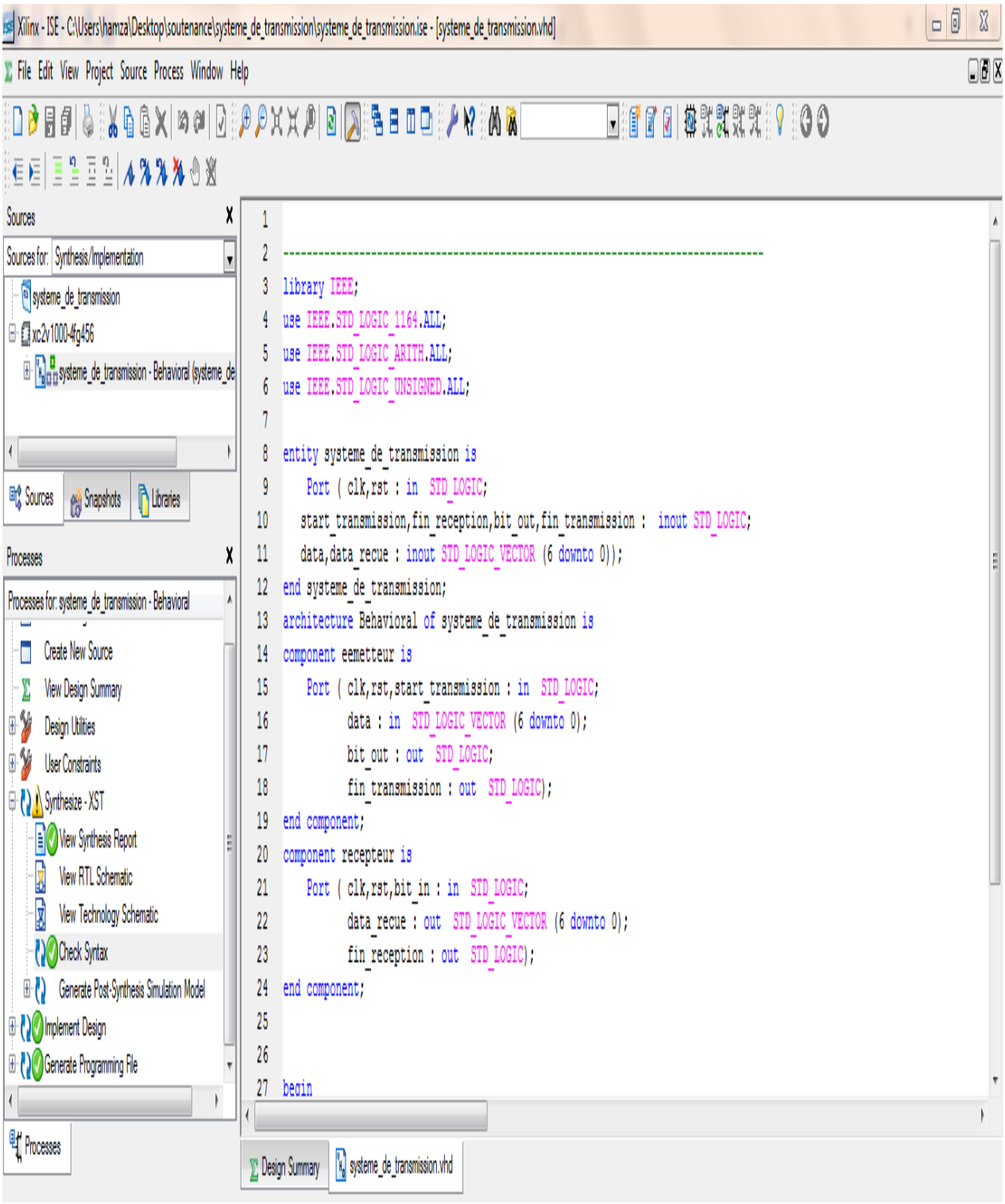


Figure III.9 : Vérification de la syntaxe du protocole

7.2 Simulation

La simulation du design permet de savoir s'il fonctionne de la façon prévue par les spécifications.

L'algorithme présenté en annexe permettra la transmission et la réception de données.la figure qui suit illustre notre simulation.

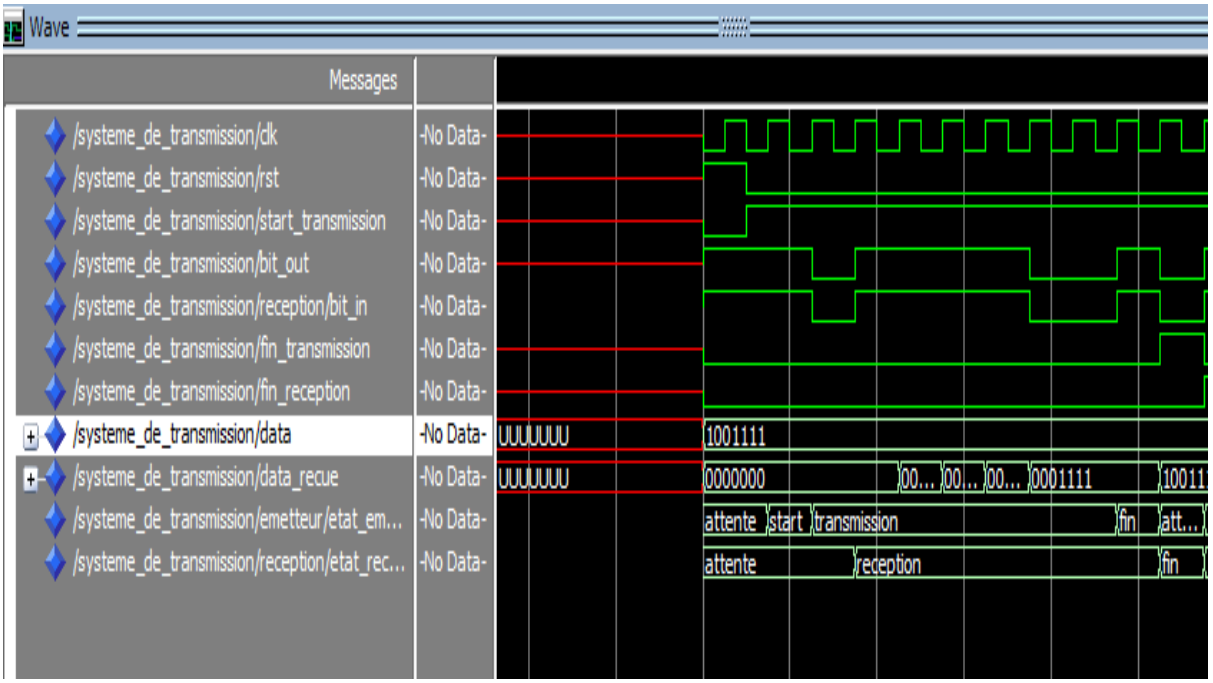


Figure III.10 : Simulation du protocole

8 Implémentation du projet

Arrivé à ce niveau notre protocole induit dans l'ISE est correcte.

8.1 Assignation des broches

Cette étape consiste à affecter les entrées et les sorties aux pins du l'FPGA de la carte

Pour se faire :

- Dans la fenêtre process, on déroule le menu User Contraints, on double clique ensuite sur Assign Package Pins.
- Une fenêtre d'attribution de pins va suivre, dans la section Loc. saisir le numéro des broches du FPGA qu'on souhaite connecter aux E/S de désigne.
- On sauvegarde les allocations avec la sélection de File > Save.

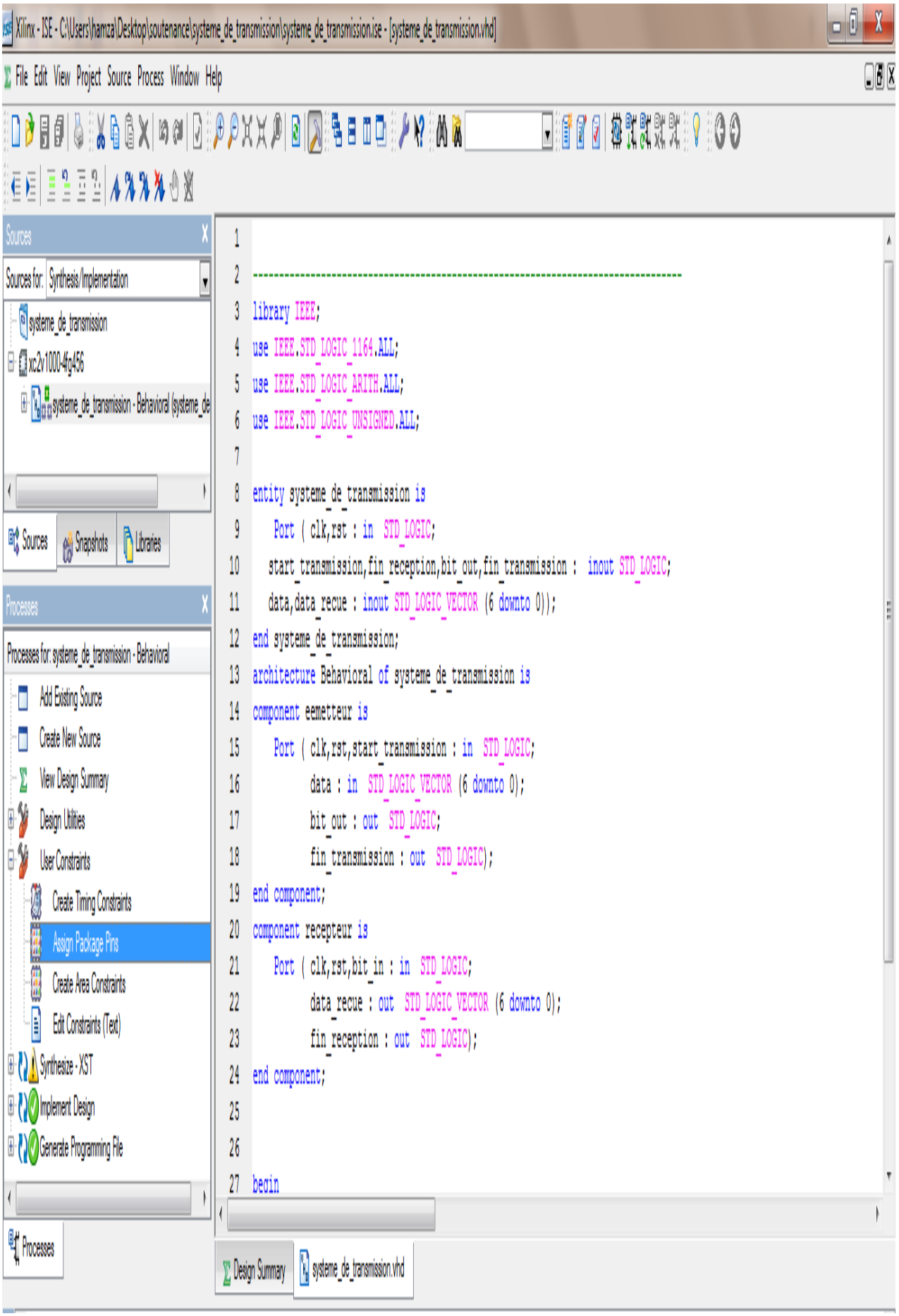


Figure III.11 : Création du fichier contrait

La figure suivante montre les affectations des broches sur la carte FPGA

Signal Name	Virtex-II Pin #	Description
DISPLAY.1A	D9	7-Segment LED Display1, Segment A
DISPLAY.1B	C9	7-Segment LED Display1, Segment B
DISPLAY.1C	F11	7-Segment LED Display1, Segment C
DISPLAY.1D	F9	7-Segment LED Display1, Segment D
DISPLAY.1E	F10	7-Segment LED Display1, Segment E
DISPLAY.1F	D10	7-Segment LED Display1, Segment F
DISPLAY.1G	C10	7-Segment LED Display1, Segment G
DISPLAY.2A	B9	7-Segment LED Display2, Segment A
DISPLAY.2B	A8	7-Segment LED Display2, Segment B
DISPLAY.2C	B8	7-Segment LED Display2, Segment C
DISPLAY.2D	E7	7-Segment LED Display2, Segment D
DISPLAY.2E	E8	7-Segment LED Display2, Segment E
DISPLAY.2F	E10	7-Segment LED Display2, Segment F
DISPLAY.2G	E9	7-Segment LED Display2, Segment G

-Affectation des broches de l'afficheur 7 segments-

Signal Name	Virtex-II Pin #	Description
DIP8	C6	User Switch Input 8
DIP7	D6	User Switch Input 7
DIP6	A5	User Switch Input 6
DIP5	B5	User Switch Input 5
DIP4	C5	User Switch Input 4
DIP3	C4	User Switch Input 3
DIP2	A4	User Switch Input 2
DIP1	B4	User Switch Input 1

-Affectation des broches de boutons switch -

Signal Name	Virtex-II Pin #	Direction	Description
CLK.CAN2	B11	Input	On-board 100 MHz Oscillator
CLK.CAN1	A11	Input	On-board 24 MHz Oscillator
CLK.CAN3	F12	Input	User clock socket (2.5V supply)

-Affectation des broches pour les horloges -

Signal Name	Virtex-II Pin #	Description
FPGA.PUSH1	D7	User Push Button Switch Input 1 (SW5)
FPGA.PUSH2	A6	User Push Button Switch Input 2 (SW6)

-Affectation des broches des boutons poussoirs-

Figure III.12 : Affectation des broches

Après avoir consulté le guide de la carte de développement de la carte FPGA Virtex II on affecte les pins selon leur distributions .la figure suivante illustre leur affectations.

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Name	Local Clock
clk	Input	B11	BANK								Unknown		☑
data<0>	InOut	C6	BANK								Unknown		☐
data<1>	InOut	D6	BANK								Unknown		☐
data<2>	InOut	A5	BANK								Unknown		☐
data<3>	InOut	B5	BANK								Unknown		☐
data<4>	InOut	C5	BANK								Unknown		☐
data<5>	InOut	C4	BANK								Unknown		☐
data<6>	InOut	A4	BANK								Unknown		☐
data_recue<0>	InOut	D9	BANK								Unknown		☐
data_recue<1>	InOut	C9	BANK								Unknown		☐
data_recue<2>	InOut	F11	BANK								Unknown		☐
data_recue<3>	InOut	F9	BANK								Unknown		☐
data_recue<4>	InOut	F10	BANK								Unknown		☐
data_recue<5>	InOut	D10	BANK								Unknown		☐
data_recue<6>	InOut	C10	BANK								Unknown		☐
rst	Input	D7	BANK								Unknown		☐
start_transmission	InOut	B4	BANK								Unknown		☐

Figure III.13 : Affectation des pins au FPGA

On a connecté

- les entrées du bus « *DATA*» au bouton switch.
- Les sorties du bus « *DATA_RECU* » à l'afficheur 7 segments.
- L'horloge « *CLK* » à *CLK.CAN 2*.
- *RST* au bouton poussoir *SW6*.
- « *START_TRANSMISSION* » au bouton switch.

Une fois les étapes sont terminées avec succès, on va générer notre programme.

8.2 Génération du fichier de programmation

- On sélectionne le fichier VHDL dans la fenêtre source et on clique sur Impliment Desgin

  C'est dans cette phase que le désigne est routé.
- Dans la fenêtre processs on clique sur Generat Programming file, un symbole vert indique que les étapes de la synthèse sont bien déroulées.

8.3 Programmation de l'FPGA avec le logiciel iMPACT

- On connecte notre machine de développement au PC.
- Dans la fenêtre process on double-clique sur Configure Device (iMPACT).
- On sélectionne le fichier de programmation(NOM_DE_FICHIER.bit).
- On clique sur OK dans la fenêtre Programming properties.
- L'écran suivant indique que la programmation s'est bien déroulée.

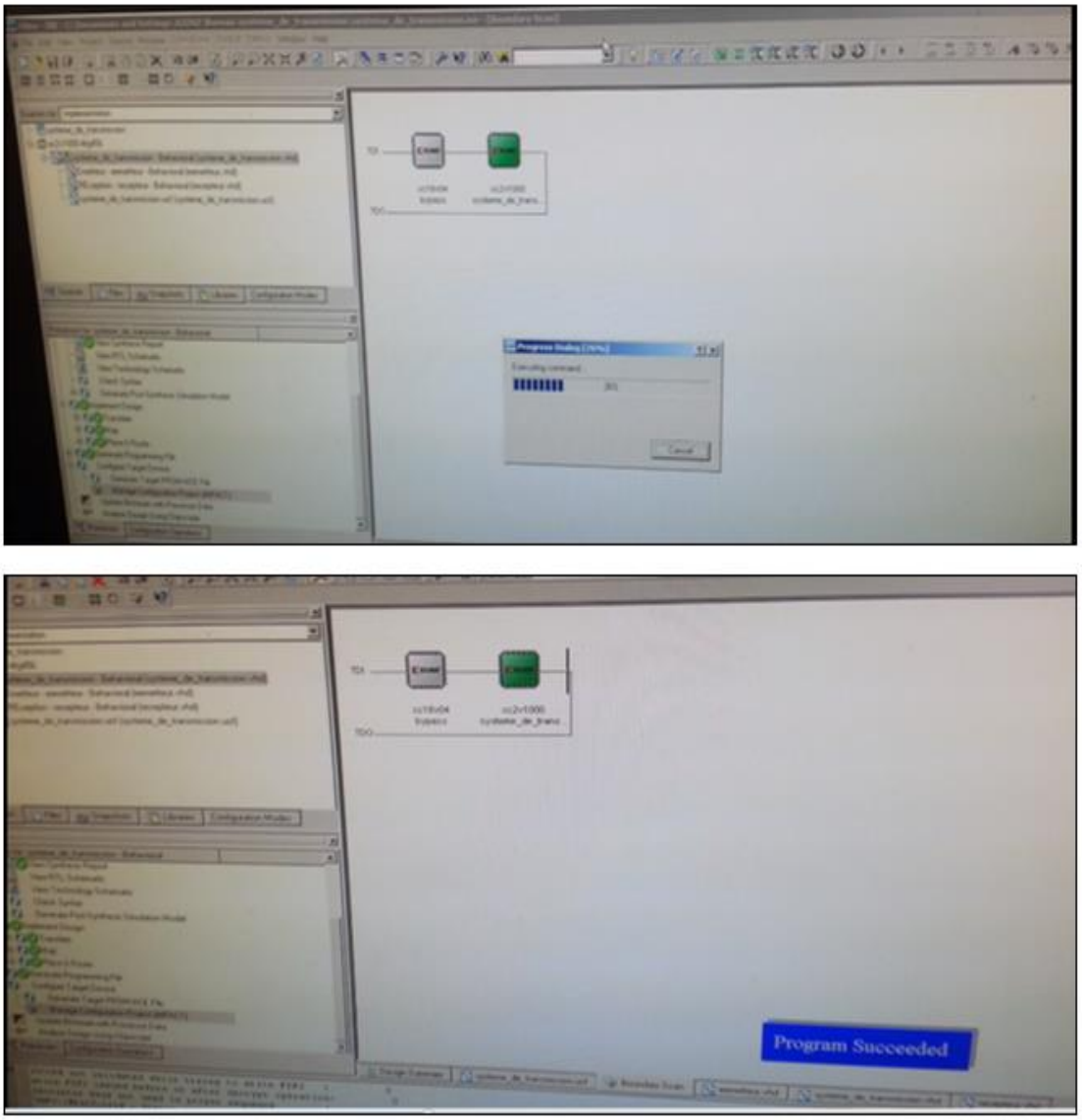


Figure III.14: changement du programme

Arrivé à cette étape, on retrouve les résultats qui s'affichent sur l'afficheur 7 segments de notre carte comme la montre la figure ci-dessous.

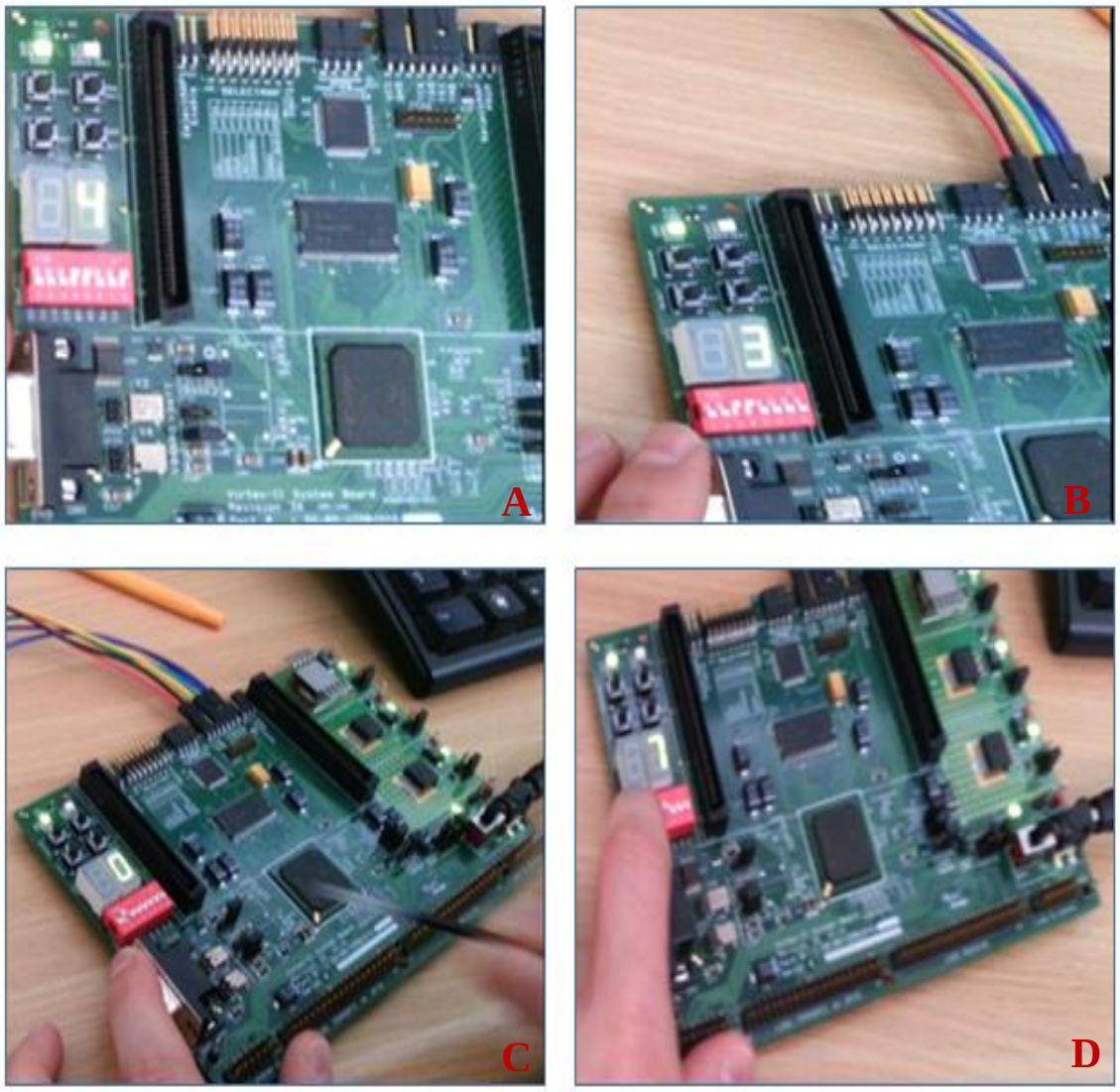


Figure III.15 : Affichage du résultat sur la carte Virtex-II

A/ envoie d'une trame de "1100110". B/ envoie d'une trame de "1001111".

C/ envoie d'une trame de "0000111". D/ envoie d'une trame de "0111111".

9 Conclusion

Dans ce chapitre on a présenté notre protocole de transmission et les différentes étapes qu'on a suivies pour la compilation, simulation et l'implémentation du projet et on a sortie avec un résultat ci-dessus

Conclusion Générale

L'implémentation du Système de transmission ne pose plus de problème avec l'évènement des FPGAs. En fait, la nano technologie et la microélectronique ont mis au point de nombreux composants pour la programmation.

Dans ce travail, j'ai implémenté un système d'acquisition de données sur la carte de développement FPGA Virtex II dans le principale avantage réside dans sa rapidité ce qui permit l'acquisition en temps réel des données.

Ce travail m'a permis d'enrichir mes connaissances théoriques et pratiques acquises au cours de mon cursus d'étude, en dépit des difficultés que j'ai rencontrées durant la concrétisation de ce projet, j'ai acquis une grande expérience en terme réalisation pratique des circuits électronique et de programmation des circuits FPGA.

J'espère que cette thèse servira de base d'étude pour des systèmes plus performants d'une part et servira de moyen didactique pour les promotions à venir.

Bibliographie

Bibliographie :

[1] F.Anceau and Y. “Bonnassieux, introduction “ in “ *conception des circuits VLSI* “. Paris, Dunod, 2007,p.1-4.

[2] A.KHOUAS. (2008). ÉLECTRONIQUE NUMÉRIQUE AVANCÉE. Available : http// :http://infotroniquedz.ble.fr/Files/77_sm6_ena_ch1_introduction.pdf

[3] Philippe. (2007).les circuits programmables, Système numérique embarqué. lien : http// : http://philippe.petitpa.perso.sfr.fr/composantsprog.pdf

[4] T. NACHEF, “IMPLANTATION D’UNE INSTRUMENTATION SUR UN FPGA,” Ph.D. dissertation, Dept. Elect, UMMTO Univ, Tizi-Ouzou, Alg, 2011

[5] M.N.HOUADJ, “Synthèse d’un module de cryptage RSA sur un circuit de type FPGA “.M.S. thèse, Dept. Automat. Univ UMMTO, Tizi-Ouzou, Alg, 2014.

[6] T.Boualem. N.Bouksil, “étude et réalisation d’une transmission sécurisée de donnée par le chaos sur une carte FPGA“ .M.S. thèse, Dept. Automat. UMMTO Univ, Tizi-Ouzou, Alg, 2013

[7] virtex-II V2MB1000 Developpent board User’s Guide ‘’, Xilinx, Inc., Décembre 2002.

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

[9] 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 », mémoire de thèse DOCTORAT, L’Université de TOULOUSE II 01 décembre 2005.

[10] KAHOUL NADHIR « COURS VHDL PLD ».

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

[12] Denis Rabasté, IUFM d’Aix-Marseille « programmation des CPLD et FPGA en VHDL avec Quartus II ».

[13] MICHEL AUMIAUX « Initiation au langage VHDL » 2ème édition, Dunod, Paris, 1999.

Annexe

Annexe : Le Code VHDL du Système de transmission

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity systeme_de_transmission is
    Port ( clk,rst , start_transmission : in  STD_LOGIC;
           data : in STD_LOGIC_VECTOR (6 downto 0);
           data_recue :  out STD_LOGIC_VECTOR (6 downto 0));
end systeme_de_transmission;
architecture Behavioral of systeme_de_transmission is
component eemetteur is
    Port ( clk,rst,start_transmission : in  STD_LOGIC;
        data : in  STD_LOGIC_VECTOR (6 downto 0);
        bit_out : out  STD_LOGIC;
        fin_transmission : out  STD_LOGIC);
end component;
component recepteur is
    Port ( clk,rst,bit_in : in  STD_LOGIC;
        data_recue : out  STD_LOGIC_VECTOR (6 downto 0);
        fin_reception : out  STD_LOGIC);
end component;
signal fin_reception,bit_out,fin_transmission :  STD_LOGIC;
begin
Emetteur: eemetteur port map(clk,rst,start_transmission,data,bit_out,fin_transmission);
REception: recepteur port map(clk,rst,bit_out,data_recue,fin_reception);
end Behavioral;
```

```
when start =>
                bit_out<='0';
                 etat_emetteur<=transmission;


when fin =>
                fin_transmission<='1';
                bit_out<='0';
                  etat_emetteur<=attente;
end case;
end if;
end process;
end Behavioral;
```

```
            data_recue(i)<=bit_in;
            i:=i+1;
            if i=7 then
               i:=0;
                  etat_recepteur<=fin;
            end if;
when fin =>
            fin_reception<='1';
            etat_recepteur<=attente;
end case;
end Behavioral;
```