الجمهورية الجزائرية الديمقراطية الشعبية

République Algérienne Démocratique et Populaire

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 : Télécommunication

Réf:.....

Mémoire de Fin d'Etudes

En vue de l'obtention du diplôme:

MASTER

Thème

CODEURS ET DÉCODEURSAMI SYNTHÈSE VHDL

Présenté par : **Proposé et dirigé par** :

BOUMEZRAG M^{ED} AMINE EDDINEM.KAHOUL NADHIR

Promotion :Juin 2012

الجمهورية الجزائرية الديمقراطية الشعبية

République Algérienne Démocratique et Populaire

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 : Télécommunication

Mémoire de Fin d'Etudes

En vue de l'obtention du diplôme:

MASTER

Thème

Codeur décodeur AMI synthèse VHDL

Présenté par : **Avis favorable de l'encadreur :**

BOUMEZRAGM[ED]AMINEEDDINE *NomPrénom* *signature*

Avis favorable du Président du Jury

Nom Prénom *Signature*

Cachet et signature

République Algérienne Démocratique et Populaire

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 : Télécommunication

Mémoire de Fin d'Etudes

En vue de l'obtention du diplôme:

MASTER

Thème

Codeur décodeur AMI synthèse VHDL

Proposé par : M.KAHOUL NADHIR

Dirigé par : M.KAHOUL NADHIR

RESUME (bilingue)

.....

.....

.....

.....

.....

DEDICACES

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 mes chères sœurs.

A toute ma famille.

A mes chers amis chaque un et son nom.

A mes ceux que j'aime.

TABLE DES MATIERES

Page

Sommaire

Liste des figures

Page

Liste des figures d'annexe:

Liste des tableaux

RÉSUMÉ

Le VHDL est un langage de description matériel, il est utilisé pour la synthèse des circuits numériques du plus simple au plus complexe. Ce travail consiste à faire la synthèse d'un ensemble de circuits appelé codeur-décodeur AMI (Alternate Mark Inversion) en utilisant le langage VHDL avec le logiciel Quartus II.7.2. Pour cela il est impératif de maîtriser ce langage, pour bien décrire les circuits en particulier la description des machines d'états avec les différentes instructions concurrentes et séquentielles. Les machines d'états ne sont que des systèmes séquentiels synchrones qu'on peut synthétiser en utilisant le langage VHDL. En plus, avec le logiciel Quartus II.7.2, on peut faire la saisie par diagramme d'états (state machine), par texte VHDL, par diagramme/schéma bloc. Ces différentes façons de faire la saisie du circuit à réaliser sont considérées comme des différentes méthodes pour la conception du codeur/décodeur AMI. Le codage AMI est utilisé dans la transmissions en bande de base, donc sans porteuse; il convertie le signal numérique (0 et 1) en un signal analogique en trois niveaux (+V, -V, 0). Le décodage AMI est l'opération inverse.

ABSTRACT

VHDL is a hardware description language; it is used for the synthesis of digital circuits from simple to complex. This work is to synthesize a set of circuits called codec AMI (Alternate Mark Inversion) using VHDL with software Quartus II.7.2. For this it is essential to master this language, to adequately describe the circuits in particular the description of state machines with various competing and sequential instructions. The state machines are only synchronous sequential systems that can be synthesized using VHDL. In addition, with the Quartus software II.7.2, can be seized by the state diagram (state machine), by text VHDL, by diagram / block diagram. These different ways of entering the circuit to be produced are considered different methods for the design of the encoder / decoder AMI. Friend coding is used in the baseband transmission, i.e. without carrier, and it converts the digital signal (0 and 1) into an analog signal into three levels (+ V,-V, 0). AMI décodage is the reverse.

ملخص:

n _t fr UTF-8

2 1

VHDL هي لغة وصف الأجهزة، ويتم استخدامها لتجميع الدوائر الرقمية من البسيط إلى المعقد. هذا العمل يرتكز على القيام بتجميع مجموعة من الدوائر ويدعى تشفير- فك التشفير AMI (البديل عكس مارك) باستخدام لغة ال VHDL و مع برمجيات Quartus II 7.2. لهذا لا بد من السيطرة على هذه اللغة، وعلى نحو كاف لوصف الدوائر في وصفها على وجه الخصوص من آلات الحالة مع مختلف تعليمات المتنافسة ومتتابعة.

أجهزة الحالة هي أنظمة متتابعة المتزامنة الوحيدة الذي يمكن تصنيعها باستخدام VHDL. وبالإضافة إلى ذلك، مع البرنامج Quartus II.7.2 ,الذي يمكن إنشاء عليه الرسم التخطيطي للحالات (جهاز الحالة),و عن طريق النص VHDL , ومن خلال رسم بياني للدارة. وتعتبر هذه الطرق المختلفة لإنشاء الدوائر طريقة للوصف بوسائل مختلفة لتصميم مشفر / فك التشفير AMI. ويستخدم الترميزAMI في نقل القاعدي، أي من دون الناقل(sans porteuse)، والذي يحول الإشارات الرقمية (0 و 1) إلى إشارة تناظرية إلى ثلاثة مستويات (+ V،-V، 0). و العملية عكسية بالنسبة لفك التشفير AMI .

Mots clés : Langage VHDL, Synthèse VHDL, Vus RTL, Codec AMI, Machine d'état, Instructions, Description structurelle, Simulation comportementale.

Résumé

INTRODUCTION GENERALE

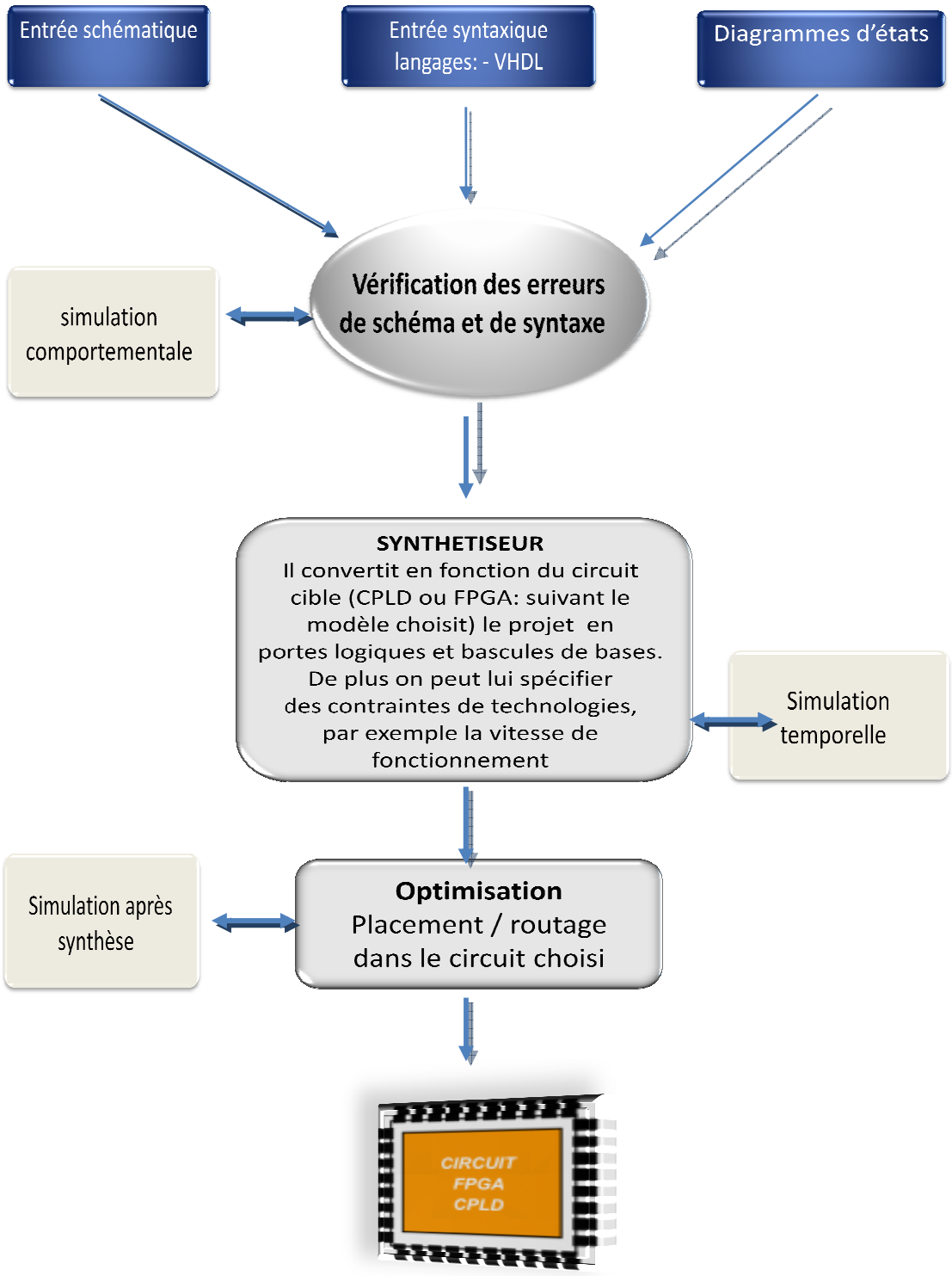
Le VHDL est un langage de description matériel, il est utilisé pour la synthèse des circuits numériques du plus simple (portes logique ET, OU, inverseur) au plus complexe(microprocesseur, mémoire,…). en utilisant le langage VHDL. L'on décrit le système par un ensemble de portes logiques et d'interconnexion « gate level » entre les deux, se trouve le niveau RTL (Register Transfer Level), pour bien décrire les circuits en particulier la description des machines d'états avec les différentes instructions concurrentes et séquentielles. En plus, avec le logiciel Quartus II.7.2, on peut faire la saisie par diagramme d'états (state machine), par texte VHDL, par diagramme/schéma bloc. Ces différentes façons de faire la saisie du circuit à réaliser sont considérées comme des différentes méthodes pour la conception du codeur/décodeur AMI qui consiste à faire dans ce travail. Cette model de codage utilisé dans la transmissions en bande de base, donc sans porteuse; il convertie le signal numérique (0 et 1) en un signal analogique en trois niveaux (+V, -V, 0). Le décodage AMI est l'opération inverse.

Le langage VHDL n'est pas absolument un langage « software » comme le C ou le Java mais un langage qui peut définir un système par une structure hiérarchique de fonctions, par une structure matériel, et encore par une modélisation temporelle (même si elle n'est pas utilisable pour faire du code synthétisable). Le Vhsic Hardware Description Language (VHSIC = Very High Speed Integrated Circuit) a été normalisé en 1987 par l'IEEE, sous la norme IEEE 1076-87. Une importante évolution est parue en 1993, sous la norme IEEE 1076-93. L'IEEE (Institute of Electrical and Electronics Engineers) est un organisme international qui définit entre autres des normes pour la conception et l'usage des systèmes électriques et électroniques. Cette version du langage qui est majoritairement supporté par les outils du marché est utilisée par Quartus, outil disponible au laboratoire. D'autres révisions de la norme sont parues , mais ce sont des évolutions « mineures », et pas forcement supporter par les logiciels. Notre premier chapitre est consacré à l'Introduction du langage VHDL.

Le circuit qu'on veut synthétiser est un circuit séquentiel synchrone. Sa caractéristique principale est la mémorisation des états, c'est pour cela qu'on parle d'états présents et d'états futurs. De même on dit état de sortie présente et état de sortie future. L'évolution du système est gérée par une horloge. Contrairement à cela, on a les systèmes asynchrones dont l'évolution se fait uniquement en fonction des entrées, sans aucune horloge. Le chapitre II est réservé pour « les systèmes séquentiels» asynchrones et synchrones. Ces derniers ne sont que des machines d'états à nombres finis. On les trouve sous l'appellation générale « Finate State Machines » qui peut être comme une machine de Mealy ou une machine de Moore.

Les machines d'états sont représentées par deux fonctions combinatoires et une fonction séquentielle qui peuvent être décrites par le langage VHDL. Plus précisément on utilise trois process pour la description de chaque fonction. Mais deux process peuvent suffire en intégrant les deux circuits combinatoires, d'entrée et de sortie, dans un même process. La description VHDL des machines d'états est donnée dans le chapitre III. L'instruction séquentielle CASE …WHEN… IF… ELSE… END IF est toujours utilisée.

Le chapitre IV est consacré à la synthèse du codeur/décodeur AMI. On commence par la présentation de cet ensemble dans une chaine de transmission en bande de base avec les 4 parties : codeur, convertisseur binaire/trois niveaux, convertisseur inverse trois niveaux/binaire et décodeur. Chaque circuit (codeur et décodeur) est synthétisé séparément. La saisie du codeur se fera de différentes manières : saisie textuelle, graphique et diagramme d'états. A chaque fois, après la compilation on a une vue RTL donnée par le logiciel Quartus, le diagramme d'état et d'autres. Après la simulation on obtient les signaux de sortie donnés par le circuit. Par la suite on a considéré le codeur/décodeur dans une description structurelle pour voir la sortie donnée par cet ensemble.


Entrée schématique
Entrée syntaxique langages: - VHDL
Diagrammes d'états
Vérification des erreurs de schéma et de syntaxe
simulation comportementale
SYNTHETISEUR
Il convertit en fonction du circuit cible (CPLD ou FPGA: suivant le modèle choisit) le projet en portes logiques et bascules de bases. De plus on peut lui spécifier des contraintes de technologies, par exemple la vitesse de fonctionnement
Simulation temporelle
Optimisation
Placement / routage dans le circuit choisi
Simulation après synthèse
CIRCUIT FPGA CPLD

CHAPITRE I:
LANGAGE VHDL POUR LA SYNTHESE

I.1 INTRODUCTION :

Le langage VHDL (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuit, **H**ardware **D**escription **L**angage) a été créé pour le développement de circuits intégrés logiques complexes. Il doit son succès, essentiellement, à sa standardisation sous la référence IEEE-1076, qui a permis d'en faire un langage unique pour la description, la modélisation, la simulation, la synthèse et la documentation.[14]

Dans ce chapitre, nous avons données une bref présentation du langage VHDL en présentant ici dans un premier temps son utilité et sa syntaxe, ensuite l'accent sera mis sur la méthodologie de développement de systèmes numérique avec le langage VHDL. La présentation des concepts et des instructions du langage VHDL est faite dans un support spécifique.

Dans un deuxième temps, nous avons présentées les caractéristiques de description les deux systèmeslogiques essentiels dans les systèmes numériques, c'est-à-dire système combinatoires et séquentiels. L'objectif principal du est de maîtriser la conception de tel systèmes. Simultanément le langage VHDL sera utilisé tout au long du projet. Toutes les fonctions de base seront expliquées puis décrites en VHDL. La description en VHDL sera pratiquée avec des exemples.

I.2 LA SYNTAXE DU LANGAGE VHDL :[5]

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

Dans la suite de ceparagraphe, nous avons expliquées les caractéristiques de chaque partie.

I.2.1 La librairie :[5]

On commence alors par définir les librairies qui seront utilisées pour décrire le circuitdans le code VHDL. La syntaxe pour la déclaration de la librairie est la suivante :

```
Library <nom_librairie> ;
Use <nom_librairie>.<nom_package>.all;
```

Généralement, c'est la librairie principale IEEE qui est utilisée. Elle contient plusieurs packages parmi lesquels on peut spécifier à l'aide du mot clé « **use** » ceux dont on a besoin.

Enfin, le ***.all*** signifie que l'on souhaite utiliser tout ce qui setrouve dans ce package. Lorsque le nom de la librairie est précédé d'IEEE., cela signifieque c'est une librairie qui est définie dans la norme IEEE, et que l'on retrouvera doncnormalement dans tout logiciel. A l'inverse, il faut se méfier des librairies qui ne sont pasIEEE, car elles sont en générale spécifiques à un logiciel.

Les packages IEEE principales sont :

- IEEE.standard
- IEEE.std_logic_1164
- IEEE.numeric_std
- IEEE.std_logic_arith

I.2.2 L'entité:

La description d'une entité correspond à celle d'un composant, d'un sous-circuit, d'un module oud'une carte. Elle comporte nécessairement une *interface* constituée d'un brochage, d'unconnecteur,… Une entité est constituée d'un entête contenant son nom suivie de la listeordonnée des différents signaux qui constituent son interface.[5]

Elle est définie de la manière suivante :

```
Entity<nom_entité>is

Port (<nom_port> : <sens><type> ;<autre_ ports>...) ;

End<nom_entité> ;
```

Un signal d'entrée/sortie est donc défini par :

- son nom
- son mode : in, out, inout, buffer (pour un signal en sortie mais utilisé comme entrée dans la description).
- son type : Std_logic, Std_Logic_Vector(x downto y). Les différents types seront présentés par la suite.

***Exemple*:**

On va définir l'entité d'une porte logique ET ayant deux entrées appelées : Input_1 et Input_2 avec une sortie appelée Output. On obtient le code VHDL de l'entité suivant :

```
Entityentité_andis
Port(Input_1: in std_logic;
Input_2: in std_logic;
 Output: out std_logic);
Endentité_and;
```

I.2.3 L'architecture:

L'architecture va décrire le fonctionnement d'une entité. On commence par sa propre déclaration. Ensuite viennent les différentes déclarations pour les signaux, les composants, les types ... [5].

```
Architecture<nom_arc> of <nom_entité>is
--liste de déclaration (déclarations des signaux, déclaration des composants, etc.)
Begin
-- corps de l'architecture
End<nom_arc> ;
```

Une architecture fait toujours référence à une entité. Elle est définie parun nom quelconque, que l'on pourra choisir pour expliciter la façon dont on code.

A la suite de la déclaration de l'architecture, on définira les déclarations préalables(Signaux internes, composants), qui seront abordés un peu plus tard.

Ensuite, vient le mot clé « ***Begin*** » après lequel vient le code qui décrit lefonctionnement de l'entité.

Dans une architecture, toutes les lignes combinatoires ou bloc de process sont exécutées en parallèle.

I.2.4 Les objets: [3]

Les objets sont des ressources internes à une unité de conception, qui n'apparaissent pas commeentrée ou sortie de l'entité. Il existe trois sortes d'objets :

A. Signal :

Le signal représente une connexion interne, déclaré dans une entité, une architectureou un paquetage. Le signal permet de pallier les limitations des directions ***in out*** et***buffer***. Il est généralement affecté à une sortie après utilisation en interne.

→ Déclarations des signaux

Toutefois les déclarations restent inchangées et sont toutes faites entre les mots clés ***is*** et ***Begin*** de l'architecture.

En voici donc la syntaxe : ***Signal***<nom_signal> : type ;

Exemple:

```
Signalport_i: std_logic;
Signalbus_signal:  Std_Logic_Vector  (15
downto 0);
Signalcount: integer range 0 to 31;
```

B. Variable :

La variable est généralement utilisée comme index dans la génération de boucle etn'a habituellement pas de correspondance avec un signal physique. Elle ne peut être déclarée quedans un process ou un sous-programme (fonction ou une procédure).

Hormis cette limitation, rien n'interdit cependant d'utiliser une variable à la place d'un signal (enmodélisation une variable utilise moins de ressources qu'un signal). Théoriquement une variablechange de valeur dès l'affectation, tandis qu'un signal ne change qu'en sortie d'un process. Les outilsde synthèse ne respectent cependant pas systématiquement cette règle.

→ Déclaration de variables Leur syntaxe est la suivante :

```
Variable<nom_variable> is : type;
```

Exemple:

```
Variablecount_v: integer range 0 to 15;
Variable data_v: Std_Logic_Vector (7 downto 0);
Variable condition_v: Boolean;
```

C. Constante:

Permet une meilleure lisibilité du programme par cecomposant définit avec une constante de N bits pourra facilement passer de 8 à 64 bits (grâce àl'instruction ***generic*** par exemple).

→ Déclaration des constantes:

```
Constant<nom_constant > : type := < valeur_initiale> ;
```

Exemple:

```
Constantetat_1 : Std_Logic_Vector := `'01" ;
Constantetat_2 : Std_Logic_Vector := `'10" ;
Constantaddr_max : integer :=1024 ;
```

I.2.5 Les types:[3], [11]

I.2.5.1 Types principaux en synthèse:

La notion de type est très importante en VHDL, chaque entrée, sortie, signal, variable ou constante,est associé à un type. Sans artifice particulier, il n'est pas question d'effectuer une opération entredeux grandeurs de type différent.

A. Type:Intègre

Nombre entier pouvant êtrelimité entre deux extremums par la déclaration :

```
« Integerrange MINI to MAXI; »
```

La déclaration d'un entier sans préciser de valeurs limites équivaut donc à créer un bus de 32 bits àl'intérieur du composant cible.

B. Type : Bit

Le type Bit peut prendre deux valeurs possibles : 0 ou 1 ; l'utilisation de la bibliothèque ***ieee1164*** permet d'ajouter une extension avec le type std_logic, lequel peut prendre en plus l'état haute impédance : **Z**.

(Attention : certains outils de synthèse exigent une majuscule pour le **Z**)

Les autres états du type ***std_logic*** définis dans la bibliothèque:

U (non initialisé), **X** (inconnus), **W** (inconnu forçage faible),**H**(NL1 forçage faible), **L**(NL0 forçage faible et quelconque) sont utilisés enmodélisation.

C. Type : Bit_Vector

Un vecteur de bits est un bus déclaré par exemple pour une largeur de ***N*** bitspar ***Bit_Vector(0 to N)*** ou bien ***Bit_Vector(N downto 0)***. Le bit de poids fort est celui

déclaré àgauche, c'est à dire le ***bit 0*** dans la première écriture et le ***bit N*** dans la seconde, ce qui explique que cette dernière est souvent préférée.

D. Type : Std_logic

Le type Std_Logic est le type utilisé dans la pratique dans l'industrie. Ilcomprend tous les états nécessaire pour modéliser le comportement dessystèmes numériques. Ce type est normalisé par IEEE. Il sera toujours utilisé pour modéliser des signaux dans un circuit. Nous donnons ci-dessousles 9 états de ce type:

'**U**'(état non initialisé), '**X**'(états inconnus forts), '**0**'(état logique 0 fort), '**1**'(état logique 1 fort). '**Z**'(état haute impédance). '**W**'(état inconnu faible).'**L**' (état logique 0 faible). '**H**'(état logique 1 faible). '**-**'(état indifférent).

E. Type : Std_Logic_Vector

L'utilisation de la bibliothèque ieee , grâce à son paquetage ***ieee. Std_logic_1164;***permet d'utiliser le type ***Std_Logic_Vector***. Avec la possibilité d'un état haute impédance.

On trouvera également dans le paquetage IEEE.std_logic_arith de la même bibliothèque la possibilitéd'utiliser les type signed (représentation de la grandeur en complément à 2) et unsigned(représentation en binaire naturel) qui permettrons de réalisée des opérations arithmétiques signéesou non.

F. Type : Booléen

ce type est utilisé pour les conditions (dans une boucle par exemple) etcomprend deux valeurs, true (vrai) ou false (faux).

I.2.5.2 Types énumération et sous-types :

Il est également possible de déclarer un type nouveau à l'intérieur d'une architecture. On parle alors detype énuméré. On peut par exemple décrire ainsi les différents états du fonctionnement d'unemachine (d'état) :

```
Type ETATS_DE_LA_MACHINEis(E1, E2, ...EN) ;
```

Il devient alors possible d'utiliser un signal (par exemple) de ce type en le déclarant par :

```
Signal ETAT_ACTUEL : ETATS_DE_LA_MACHINE ;
```

Dans la même idée le VHDL permet de définir des sous types à partir d'un type déjà existant. On pourrapar exemple créer un type octet à partir du type ***Std_Logic_Vector*** :

```
Subtype OCTET is Std_Logic_Vector(7 downto 0);
```

I.2.5.3 Type tableau :

Le type ***Std_Logic_Vector*** définit un bus de N bits qui peut être vu comme un tableau d'une colonne et Nlignes ; Il est également possible de définir des tableaux comprenant des grandeurs d'un type autre que bit etmême des tableaux de plusieurs colonnes.

Dans la pratique, les outils de synthèse ne reconnaissent souvent que les tableaux de 2 dimensions,que l'on utilise pour la synthèse de mémoire.

Voici un exemple de déclaration d'un type tableau pouvant représenter le contenu d'une mémoire de Mmots de N bits :

```
Type MEMOIRE is array (0 to M, 0 to N) of std_logic;
```

Il est possible d'accéder à un élément particulier du tableau, soit pour lui affecter une valeur, soit pouraffecter sa valeur à une autre grandeur (de même type). Imaginons que nous voulions affecter à lagrandeur s de type std_logic le deuxième bit du premier mot, nous pourrons écrire :

```
s <= MEMOIRE(2,0) ;
```

I.2.5.4 Type enregistrement:

Le VHDL permet la description de tableau comprenant des grandeurs de types différents. Dans unesynthèse cela se résume souvent à la juxtaposition de vecteurs de bits de tailles différentes. Prenonsl'exemple de la description d'un plan mémoire définit par 2^P pages de 2^M mots de ***N*** bits et créons untype MEM incluant tous ces éléments :

```
Type MEM is record
Page: std_bit_vector(P downto 0);
Adr: std_bit_vector(M downto 0);
Mot: std_bit_vector(N downto 0);
End record ;
```

En définissant un signal S de type MEM, on peut ensuite accéder à chaque élément du signal, la pagepar exemple et lui affecter la valeur hexadécimale 0 :S.page<= x"0" ;

I.2.5.5 Initialisation:

A chaque type (qu'il soit créé par l'utilisateur ou définit dans une bibliothèque normalisée) correspondune description des valeurs possibles. Le compilateur initialise automatiquement les grandeurs à lapremière valeur définie (par exemple E1 pour l'exemple du signal ETAT_DE_LA_MACHINE , 0 pour une grandeur de type bit etc...).

Si cela pose un problème,il est possible d'initialiser les grandeurs à une autre valeur lors de la déclaration par une affectation,par exemple :

```
Signal ETAT_ACTUEL : ETAT_DE_LA_MACHINE := E2;
```

Initialiser une grandeur demande cependant de gérer le circuit à la mise sous tension, ce quin'est généralement possible que par des éléments extérieurs au composant (circuit RC parexemple). Aussi l'instruction précédente, hormis associée à l'instruction ***generic***, reste sanseffet avec les outils de synthèse.

I.2.5.6 Les agrégats:

L'agrégat est un moyen d'indiquer la valeur d'un type composite. Prenons l'exemple du signal Qsuivant :

```
Signal Q: Std_Logic_Vector (5 downto 0);
```

Si on souhaite donner, par exemple, la valeur "101111", il est possible d'utiliser une affectation simple :

```
Q <= "101111";
```

Dans certains cas, par exemple lorsque le nombre de bits est important, ou pour ne modifier quecertains bits, l'utilisation d'un agrégat simplifie l'écriture. Plusieurs formes sont possibles :

```
Q <= ('1 ',' 0',' 1',' 1',' 1',' 1') ;
Q <= ( 5 =>' 1 ', 3 =>'1', 4 =>'0', 0 => '1', 2 => '1', 1 => '1') ;
Q <= (5 =>'1', 3 downto0 => '1', others=>'0');
```

Pour mettre tout les bits du bus au même état (en haute impédance par exemple), on peut écrire :

```
Q <= (others=> 'Z') ;
```

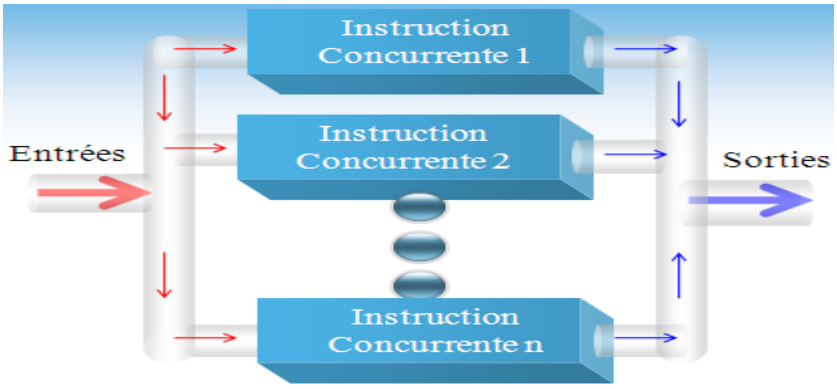
I.2.5.7 Les alias:

Ils permettent de désigner sous un autre nom une partie d'une grandeur, par exemple appeler parsigne le bit de poids fort d'un mot de 16 bits (codé en complément à 2).

```
Signal Mot: Std_Logic_Vector (15 downto0);
Alias signe:std_logic is mot(15) ;
```

I.3 LES INSTRUCTION DE BASE (MODE « CONCURRENT »):[7]

Qu'est ce que le mode « concurrent » ? Pour une description VHDL toutes les instructions sont évaluées et affectent les signaux de sortie en même temps. L'ordre dans lequel elles sont écrites n'a aucune importance. En effet la description génère des structures électroniques, c'est la grande différence entre une description VHDL et un langage informatique classique.



Avec VHDL il faut essayer de penser à la structure qui va être générée par le synthétiseur pour écrire une bonne description, cela n'est pas toujours évident.

Fig. I.1. *Instructions en mode concurrent.*

A. L'affectation simple : <=

Dans une description **VHDL**, c'est certainement l'opérateur le plus utilisé. En effet ilpermet 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 ;
```

B. Affectation conditionnelle :

Cette instruction modifie l'état d'un signal suivant le résultat d'une condition logiqueentre un ou des signaux, valeurs, constantes.

```
NOM_D'UNE_GRANDEUR <= Q1 when CONDITION1 else
Q2 when CONDITION2 else
                                    Qn ;
```

C. Affectation sélective :

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

```
With Expressionselect
 NOM_D'UNE_GRANDEUR  <= expression when valeur1_de_selection,
                                    Expressionwhen valeur2_de_selection,
                        .....
Expressionwhen others;
```

Remarque: When othersest nécessaire car il faut toujours définir les autres cas du signal de sélection pour prendre en compte toutes les valeurs possibles de celui-ci.

D. La boucle for…generate :

Une boucle est répétée plusieurs fois en fonction d'un indice.

```
For i in MIN to MAX generate
INSTRUCTIONS ;
End generate ;
```

La suite d'instruction peut comprendre des grandeurs indexées par i qui seront écrite par exemple D(i)et Q(i)

Ce type d'instruction peut induire une certaine confusion : il faut bien avoir à l'esprit que le VHDLdécrit une structure ou le comportement d'une structure implantée dans le

circuit cible, maisn'exécute pas les instructions au sens où on l'entend avec la programmation en assembleur ouen C d'un microprocesseur.

I.3.1 Les opérateurs: [11]

Le langage VHDL dispose de 6 groupes d'opérateurs. Le langage debase ne définit pas ces opérateurs pour tous les types, mais il est possiblede surcharger ceux-ci. Nous donnons la liste des opérateurs pour la norme2000 du langage VHDL.

Dans le langage VHDL, les états logiques d'un signal sont définit parun type scalaire énumé. Il s'agit des types Bit et Bit_Vector. Ceux-ci ne sontjamais utilisés en pratique. Nous utiliserons uniquement dans ce manuelles types Std_Logic et Std_Logic_Vector. Les opérateurs sont aussi définies pour les types Std_Logic et Std_Logic_Vector par une surcharge définie dans le paquetage Std_Logic_1164.

Toutes les descriptions VHDL ce paquetage Std_Logic_1164.

I.3.1.1 Opérateurs logiques:

Les opérateurs logiques sont définies pour les types Bit et Bit_Vector.Ceux-ci sont aussi définis pour les types Std_Logic et Std_Logic_Vectorpar une surcharge définie dans le paquetage Std_Logic_1164.

Opérateur	VHDL
ET	*and*
NON ET	*nand*
OU	*or*
NON OU	*nor*
OU EXCLUSIF	*xor*
NON OU EXCLUSIF	*xnor*
NON	*not*
DECALAGE A GAUCHE	*sll*
DECALAGE A DROITE	*srl*
ROTATION A GAUCHE	*rol*
ROTATION A DROITE	*ror*

Tableau I.1 : *Opérateurs logiques.*

I.3.1.2 Opérateurs relationnels.

Les opérateurs relationnels sont définis pour tous les types scalaires etles tableaux de scalaires. Mais les deux opérandes doivent être du mêmetype.

Opérateur	VHDL
Egal	=
Non égal	/=
Inférieur	<
Inférieur ou égal	<=
Supérieur	>
Supérieur ou égal	>=

***Tableau I.2** : Opérateurs relationnels*

Les opérateurs de décalages et de rotation sont définis pour tous les types scalaires et les tableaux de scalaires.

Opérateur	VHDL
DECALAGE A GAUCHE	*sll*
DECALAGE A DROITE	*srl*
ROTATION A GAUCHE	*rol*
ROTATION A DROITE	*ror*

***Tableau I.3:** Opérateurs de décalage*

Ces opérateurs demandent deux opérandes. Le premier opérande est untableau, le second est un entier qui indique le nombre de décalage ou de rotation. Le type du résultat est le même que l'opérande de gauche.

I.3.1.3 Opérateurs arithmétiques :

Les opérateurs arithmétiques sont définis pour les types numériques.L'utilisation de ces opérateurs pour les types tableaux (exemple:Std_Logic_Vector) nécessitera l'utilisation d'une bibliothèque. Il est recommandé d'utiliser uniquement le paquetage normalisé Numeric_Std.

Opérateur	VHDL
ADDITION	+
SOUSTRACTION	-
MULTIPLICATION	*
DIVISION	/

***Tableau I.4 :** Opérateurs arithmétiques.*

I.4 LES INSTRUCTIONS DU MODE SEQUENTIEL:[7]

Les instructions séquentielles doivent être utilisées uniquement dansune zone de description séquentielle. Nous utiliserons principalement ces instructions à l'intérieur de l'instruction process. Ces instructions peuventaussi être utilisées dans des procédures et des fonctions.

Fig. I.2*. Instructions en mode* séquentielles.

I.4.1 Les processus :[1]

Un **process** définit un comportement qui doit se dérouler lorsque ce **process** devientactif. Le comportement est décrit par une suite *d'instructions séquentielles* exécutées dans leprocess.

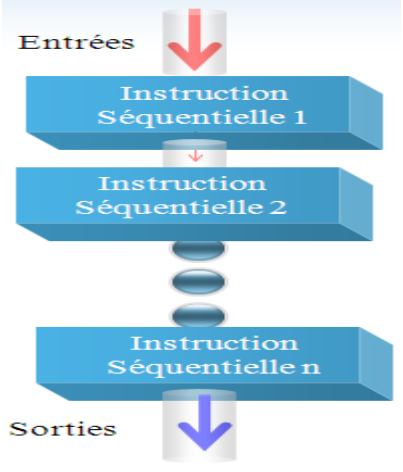
Il permet de décrirede manière algorithmique le fonctionnement d'un système numérique, et ilsera indispensable pour décrire des éléments mémoires(bascules) en VHDL comme il est utilisé pour de nombreux types de description.

La syntaxe générique de l'instruction process est la suivante:

```
[ETIQUETTE :] process(liste de « sensibilité »)
« Déclaration des objets utilisés dans le process »
Begin
« Suite d'instructions séquentielles »
End process[ETIQUETTE] ;
```

L'exécution d'un **process** est déclenchée par un ou des changements d'états de signauxlogiques. Le nom de ces signaux est défini dans **la liste de sensibilité** lors de la déclarationdu **process**.

Le nom du *process* entre crochet est facultatif, mais il peut être très utile pourrepérer un *process* parmi d'autres lors de phases de mise au point ou de simulations.



I.4.2 Règles de fonctionnement d'un process :[1]

1) L'exécution d'un ***process*** a lieu à chaque changement d'état d'un signal de la liste de sensibilité.

2) Les instructions du ***process*** s'exécutent séquentiellement.

3) Les changements d'état des signaux par les instructions du ***process*** sont pris en compte à la**fin** du ***process***.

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;
```

A. L'affectation simple :

La syntaxe générique d'une affectation simple est la suivante:

```
Signal_1 <= Signal_2 fonction_logique Signal_3;
```

L'affectation ne modifie pas la valeur actuelle du signal. Celle-ci modifieles valeurs futures. Le temps n'évolue pas lors de l'évaluation d'un process.L'affectation sera donc effective uniquement lors de l'endormissement du process. Voir "L'instruction processus".

B. L'instruction conditionnelle:

Cette instruction est très utile pour décrire à l'aide d'un algorithme le fonctionnement d'un système numérique.

La syntaxe générique de l'instructionconditionnelle est la suivante:

```
If Condition_Booléenne_1 then
--Zone pour instructions séquentielles
Elsif Condition_Booléenne_2 then
--Zone pour instructions séquentielles
   Elsif Condition_Booléenne_3 then
   ...
Else
--Zone pour instructions séquentielles
End if;
```

C. L'instruction de choix:

Ces instructions sont particulièrement adaptées à la description des machines d'états.La syntaxe générique d'une instruction de choix est la suivante:

```
Case Expression is
When ETAT _1 =>--Zone pour instructions séquentielles
When ETAT _2 =>--Zone pour instructions séquentielles
....
When others=>--Zone pour instructions séquentielles
End case;
```

Le cas ***others*** n'est jamais utilisé pour les combinaisons logiques de Sel. Mais le type ***Std_Logique*** comprend 9 états ('X', 'U', 'H', etc.).

Si le type utilisé pour l'expression est ordonné (exemple: Intègre, Natural, ..), il est possible de définir des plages:

```
When 0 to 7 =>--Zone pour instructions séquentielles
Ou
When 31 downto 16 =>--Zone pour instructions séquentielles
```

D. La boucle« for...in...to...loop...end loop»

La syntaxe de cette boucle***for … loop*** est la suivante :

```
For N in X to Y loop
INSTRUCTIONS_SEQUENTIELLE ;
End loop ;
```

E. La boucle « while...Loop...end loop»

La syntaxe de cette boucle ***while…Loop*** est la suivante :

```
While CONDITION loop
INSTRUCTIONS_SEQUENTIELLE;
End loop;
```

F. L'attente « *Wait until*»

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'event and CLK= '1');
```

I.4.3 Les attributs:[14]

Ils permettent d'accéder à une caractéristique particulière d'un signal, d'un tableau ou d'un type. LeVHDL laisse la possibilité à l'utilisateur de créer ses propres attributs. Nous ne nous intéresseronscependant qu'aux attributs prédéfinis.

Il existe plusieurs catégories d'attributs. Voici la liste des différents types d'attributs définit en VHDL:

- Les attributs pré-définis pour les types
- Les attributs pré-définis pour les tableaux (array)
- Les attributs pré-définis pour les signaux
- Les attributs définis par l'utilisateur

Cette dernière catégorie comprend principalement des attributs définispar les outils. Nous pouvons donner comme exemple, les attributs définissent par le synthétiseur. Ceux-ci permettent de définir les numéros des pins dansla description et de passer l'information à l'outil de placement routage.

La syntaxe d'utilisation est :

```
Nom_de_la_grandeur ' nom_de_l'attribut ;
```

Pour illustrer les principaux attributs utilisés pour les tableaux, prenons comme exemple le typetableau :

```
Type T is array (4 to 8, 7 downto 2) of std_logic;
```

Nous allons montrer le fonctionnement de ces attributs avec un vecteurqui est un tableau de ***Std_Logic***.

La syntaxe d'utilisation est :

```
Signal Data: Std_Logic_Vector (7 downto 0);
```

I.5 LES BIBLIOTHEQUES:[8]

Ce sont des répertoires au sens informatique du terme où se trouvent des fichiers contenant desunités de conception, c'est à dire :

- des entités et architectures associée,
- des spécifications de paquetage, ainsi que la description du corps associé,
- des configurations.

Ces ressources n'auront donc pas besoin d'être décrite dans le programme principal. Il suffirad'appeler la bibliothèque avec la syntaxe :

```
Library NOM_DE_LA_BIBLIOTHEQUE ;
```

Il existe des bibliothèques prédéfinies (***ieee*** ou ***std*** par exemple), mais l'utilisateur peut créer sa proprebibliothèque. Dans ce dernier cas, il est nécessaire de préciser à l'outil de synthèse où se trouvent lesbibliothèques utilisées.

Les entités et architectures permettront la description de composants. Voyons maintenant ce que sontles paquetages et configurations.

I.5.1 Les paquetages:

Un paquetage permettra d'utiliser sans avoir a les réécrire essentiellement des objets (signaux,constantes), des types et sous-types, des sous-programmes et des composants.Le paquetage est composé d'une unité de conception primaire qui en donne une vue externe (commeune entité), dont la syntaxe est :

```
Package NOM_DU_PAQUETAGE is
        Définition des types, sous-types, constantes ;
        Déclaration des fonctions, procédures, composants, signaux ;
EndNOM_DU_PAQUETAGE ;
```

L'ensemble est placé, dans un fichier VHDL au sein d'une bibliothèque.

On peut noter qu'un composant est seulement déclaré par un paquetage, mais pas décrit par le corpsdu paquetage, une unité de conception ne pouvant contenir une autre unité de conception. Ladescription se trouve dans un fichier VHDL d'une bibliothèque.

L'appel du paquetage se fait après l'appel de la bibliothèque qui le contient. On indique alors le nom dufichier contenant le paquetage par l'instruction qui suit :

```
Use NOM_DE_LA_BIBLIOTHEQUE. NOM_DU_PAQUETAGE.all;
```

Dans la pratique, il n'y a généralement qu'un paquetage par fichier, et les deux portants le même nom.Le mot réservé point ***all*** permet théoriquement d'appeler tous les paquetages du fichier (il seraitpossible d'en appeler un seul, mais cela présente peu d'intérêt).

Il existe des paquetages prédéfinis fournis avec les outils de synthèse :

✓ Le paquetage ***standard*** qui contient la déclaration de tous les types prédéfinis de base quenous avons vue. Son utilisation est implicite, il n'est pas nécessaire de l'appeler.

✓ Le paquetage ***std_logic_1164*** contenu dans la bibliothèque ieee qui contient entre autre ladéfinition des types std_logic et ***Std_Logic_Vector***.

✓ Le paquetage ***std_logic_arith*** (très semblable au paquetage ***Numeric_Std***) de la bibliothèqueieee qui permet des conversions de type (entier vers std_logic et inversement).

✓ Les paquetages ***std_logic_signed*** et ***std_logic_unsigned*** de ieee permettant desopérations arithmétiques signées ou non sur des types std_logic.

On rappelle qu'un ***Std_Logic_Vector*** sera considéré comme codé en binaire naturel si lepaquetage ***ieee. Std_logic_unsigned*** à été déclaré et comme codé en complément à 2 si lepaquetage ieee. ***Std_logic_signed*** a été déclaré.

I.6 CONCLUSION:

Dans ce chapitre nous avons connus la capacité plus précisément les descriptions les systèmes numériques en langage VHDL qui touche spécialement les systèmes combinatoires ou séquentiels. La description VHDL est utilisée donc pour des circuits simples ou complexes comme les circuits séquentiels et les machines d'états.

CHAPITRE II:
LES SYSTEMES SEQUENTIELS

II.1 INTRODUCTION :

Un système est dit séquentiel, si à un même vecteur d'entrée, il fait correspondreplusieurs vecteurs de sortie différents. Chaque vecteur de sortiedépendra alors non seulement du vecteur d'entrée à l'instant t, mais aussides précédents, ce qui introduit la notion de séquence d'entrée.

L'effet mémoire est typique des systèmes séquentiels, l'élément de mémorisationle plus important est la bascule D, constituée d'un ensemble deportes logiques. Même si, en soi, une porte logique ne retient pas de donnée,il est possible d'en raccorder quelques-unes ensemble afin d'obtenir lestockage d'une information. Il existe différentes façons de monter les portespour obtenir ces bascules. [10]

II.2 CIRCUITS LOGIQUES SEQUENTIELS:[10]

Un circuit logique séquentiel, (Fig. II.1) est un circuit logique possédant des entrées et des sorties et présentantun comportement où les sorties ne dépendent pas seulement des entrées, mais également des séquencesdes entrées passées. Pour ce faire, le circuit utilise une partie mémoire qui va lui permettre de retrouver l'**état**induit par les entrées passées. La sortie est par conséquent calculée en fonction de l'état présent et des entréesqui arrivent au système.

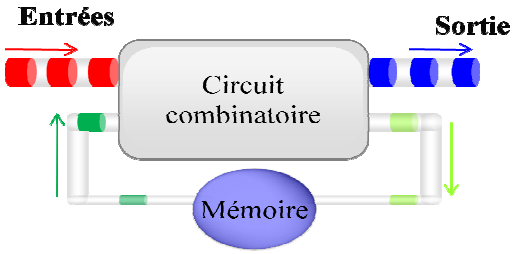


Fig. II.1.*Schéma générique d'un circuit séquentiel.*

II.3 LES OPERATEURS SEQUENTIELS:

Les opérateurs séquentiels peuvent être classés en deux grandes familles qui sedifférencient par la façon dont peuvent arriver les transitions :

1. Les opérateurs asynchrones, les plus anciens et les plus simples, sont desimples circuits combinatoires sur lesquels on introduit une réaction positive.

2. Les opérateurs synchrones, plus complexes, utilisent plusieurs opérateursasynchrones pour mémoriser leur état. L'idée qui préside à la réalisation desopérateurs synchrones est de les munir d'une entrée très particulière, l'horloge,dont un front, montant ou descendant, fixe les instants où les transitions entreétats sont effectuées.

II.4 SYSTEMES SEQUENTIELS ASYNCHRONES:[6]

Les machines asynchrones ont la particularité de changer d'état lors d'une variation sur lesentrées. Si les entrées demeurent stables, le système demeurera dans le même état. Le système est stable lorsque la valeur avant et après le délai sont identiques (l'état et l'excitation sont égaux).

Dans beaucoup de situation pratique, les entrées du système à réaliser ne sont pas gérées par une horloge etsont donc sujettes à modification à des instants quelconques. Dans ce cas il n'est pas possible de concevoir unsystème synchrone.

II.4.1 Structure des systèmes séquentiels asynchrones:[12]

Dans le mode asynchrone, la fonction de mémorisation est réalisée par de simples boucles de rétroaction.L'évolution des états ne dépend donc que des modifications intervenant sur les entrées primaires (Ei) de la machine. La représentation la plus générale d'un système asynchrone est donnée sur la figure. II.2.

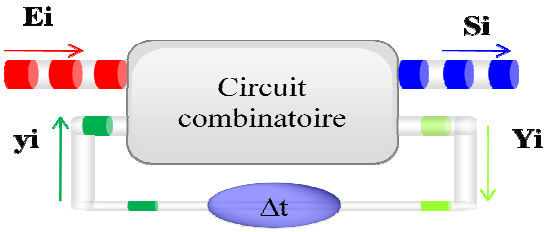


Fig. II.2.*Modèle général d'un système séquentiel asynchrone*

Le circuit est dit dans un état stable si et seulement si yi=Yi pour tout i=1, 2, ..., k. En réponse à unchangement d'entrée, la logique combinatoire produit un nouvel ensemble de valeurs sur Yi. Si ces valeurs sontdifférentes de celles de yi ce circuit entre dans un état instable. On obtiendra un nouvel état stable lorsqu'ànouveau on aura yi=Yi. Si aucun état stable n'est atteint le système oscille.

Une variation d'entrée est obligatoire pour que le système évolue d'un état stable vers un autre état stable.D'autre part, vu les délais inhérents aux structures, il est impossible de garantir le changement de deux variablessimultanément. La synthèse d'un système asynchrone doit interdire de telles situations. Cette restriction signifieen effet qu'une seule entrée peut changer à un instant donné, et que le temps écoulé entre deux variations est plusgrand que le délai interne de la structure. Lorsqu'un changement apparaît sur une entrée, nous ferons donc l'hypothèse qu'aucune variation sur toutes entrées ne peut intervenir avant que le circuit soit dans un état stable.

II.4.2 Méthode de synthèse:[7]

La méthode de synthèse de systèmes séquentiels asynchrones, dans la figure II.3, proposée a pour but de minimiser le nombre devariables secondaires (Yi). Ceci a pour effet direct de minimiser le nombre de fonctions du réseau combinatoireet par conséquent le nombre de portes du circuit.

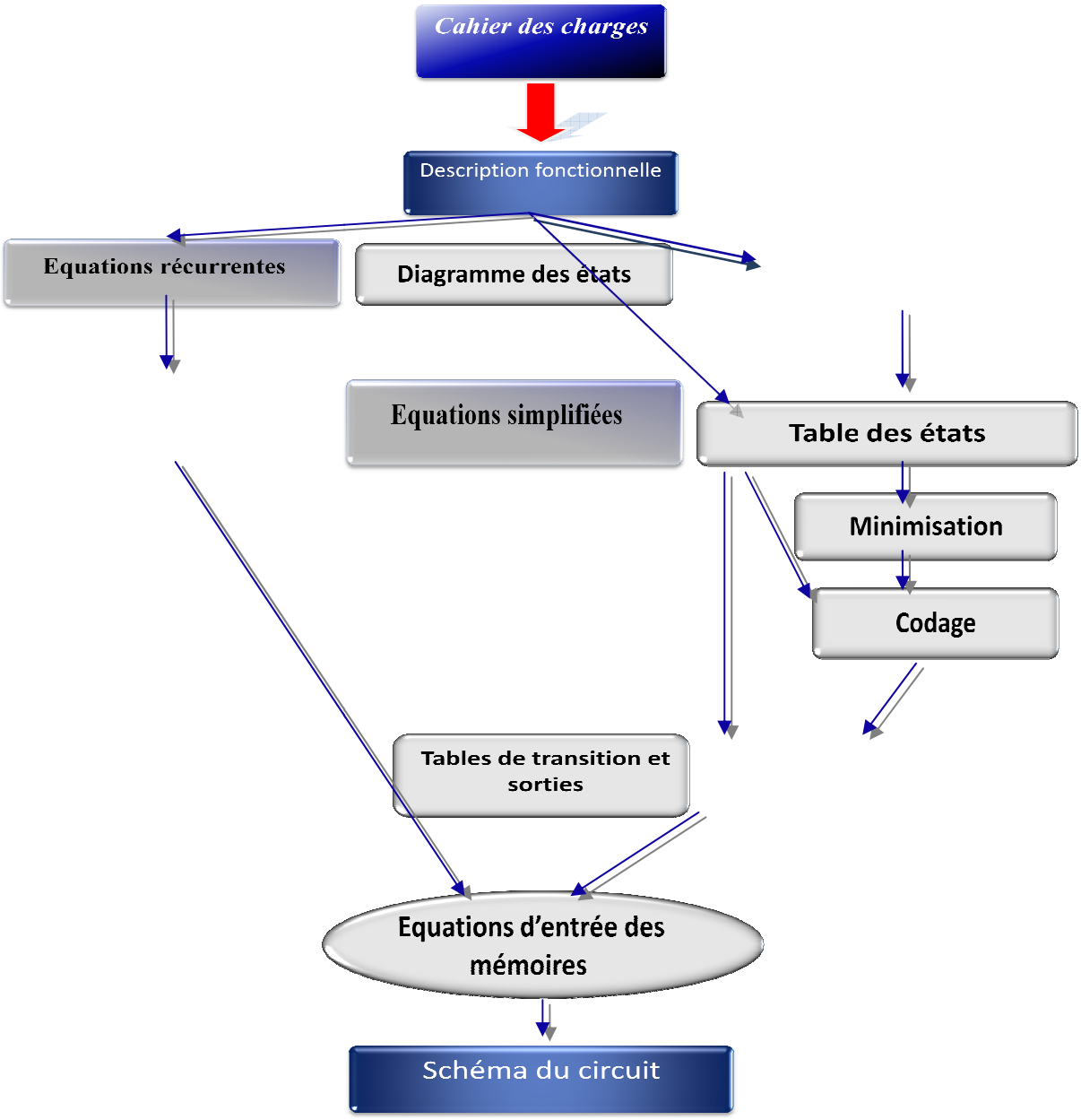


Fig.II.3. *Méthode de synthèse logique.*

II.4.2.1 Synthèse des machines séquentielles asynchrones: [4]

La méthode d'***Huffman*** permet de faire la synthèsedes machines séquentielles asynchrones à partir du principe de fonctionnement ou cahier de charge. C'est une méthode de synthèse des systèmes séquentiels qui oblige à faire une étude complète du système à réaliser, elle

fournit un moyen systématique de réalisation avec un minimum de variables internes (bascules).

Après avoir choisi les variables d'entrées et les fonctions de sortie nous avons appliqué les étapes suivantes :

- ✓ ***1. Modélisation du cahier des charges***
  - Graphe d'état(oudiagramme des transitions)
  - Matrice primitive ettableau des états de sortie
- ✓ ***2. Minimisation du nombre d'états***
  - Fusionnement d'états ou polygone des liaisons
  - Matrice réduite.
- ✓ ***3. Codage des états***
  - Graphe d'adjacence
  - Assignation des états et matrice ordonnée
- ✓ ***4. Synthèse***
  - Synthèse des variables secondaires
  - Synthèse des sorties
  - Simplifications éventuelles.
  - Schéma selon la technologie adoptée.
  - Simulation et essais

II.5 SYSTEMES SEQUENTIELSSYNCHRONES:[10]

Les systèmes séquentiels peuvent être différenciés en fonction de leur mode de fonctionnement qui peut être synchrone ou asynchrone. Dans le mode synchrone, les éléments de mémorisation sont des bascules. Les modifications d'état du système ne peuvent donc intervenir qu'à des instants très précis déterminés par des signaux d'horloge.

II.5.1 Définition et représentation*:* [4]

La caractéristique fondamentale des systèmes séquentiels synchrones est la présence du signal D'horloge. En effet ces systèmes évoluent lorsqu'un signal d'horloge leur est appliqué à leur entrée dite horloge 'H'. Ainsi les éléments de mémorisation sont des bascules synchrones qui assurent les modifications d'état du système à des instants très précis déterminés par les signaux d'horloge.(figure II.4)

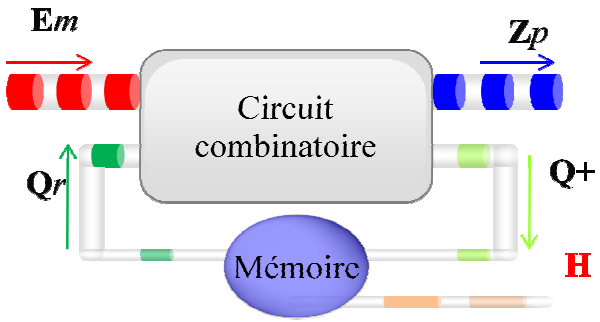


Fig. II.4. *Schéma général d'un système séquentiel synchrone.*

Un système séquentiel synchrone est constitué d'un certain nombre de variable d'entrée, de variables d'état interne (ou variables secondaires), les fonctions secondaires (ou fonctions d'excitation), et enfin de fonctions de sortie.

Il peut être représenté par le schéma de la figure II.4 avec:

→ E est le vecteur d'entrée, il représente toutes les variables principales d'entrée $E = (e_1, e_2, \dots, e_m)$. On a m entrées principales.

→ Z est un vecteur de sortie, il représente toutes les fonctions de sortie du système$Z = (z_1, z_2, \dots, z_p)$. On a p fonctions de sortie.

→ Q est un vecteur d'état, il représente les variables secondaires d'entrée ou variables d'état interne. $Q = (q_1, q_2, \dots, q_r)$. On a r variables d'état interne ou variables secondaires d'entrée.

II.5.2 Éléments de mémoire : [11]

Les bascules sont des éléments séquentiels simples qui réalisent une fonction de mémorisation. Leurintérêt réside principalement dans leur utilisation pour réaliser des systèmes complexes.

La solution couramment adoptée est de construire les fonctions complexes avec desbascules synchrones, qui disposent d'une entrée réservée et unique qui fixe les instants deséventuels changements d'états : l'horloge.

II.5.3 Signal d'horloge et bascules synchrones : [11]

Les circuits numériques peuvent fonctionner de façon synchrone ouasynchrone. Dans les systèmes asynchrones, la sortie des circuits logiquespeut changer d'état à tout moment quand une ou plusieurs entrées changent.Un système asynchrone est difficile à concevoir et à débugger.

Par contre dans un système synchrone, le moment exact où la sortiechange d'état est commandé par un signal que l'on appelle couramment signal d'horloge. Ce signal est généralement un train d'ondes rectangulairesou carrées, comme ceux de la figure. II.5.

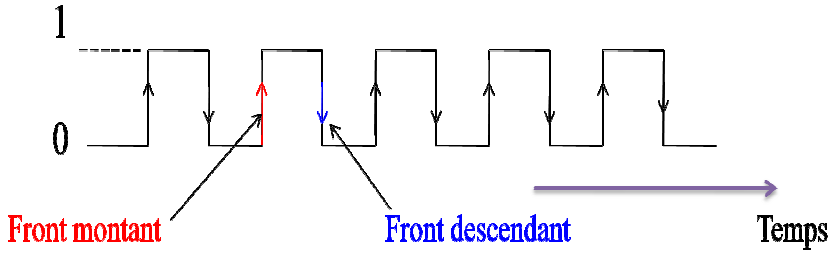


***Fig. II.5** : Signaux d'horloge.*

Le signal d'horloge est habituellement distribué à tous les étages du système, de sorte que la plupart dessorties du système, sinon toutes, changent d'état seulement quand le signald'horloge effectue une transition. Ces transitions, appelées fronts ou flancs.Quand le signal d'horloge passe de 0 à 1, on parle du front montant (transition positive), quand il passe de 1 à 0, on parle de front descendant (transition négative).

La majorité des systèmes numériques existants sont des machines synchrones du fait que les circuits synchrones sont plus simples à concevoir età dépanner. Leurs sorties ne peuvent changer qu'à des instants précis bienconnus. Autrement dit, tous les changements sont synchronisés avec lestransitions du signal d'horloge.

La synchronisation orchestrée par des signaux d'horloge est réalisée aumoyen de bascules synchrones qui ont été réalisées pour changer d'état aumoment de la transition associée à un front ou à l'autre du signal d'horloge.

II.6 LES FONCTIONS SÉQUENTIELLES : [10]

La différence essentielle entre une fonction combinatoire et une fonction séquentielleréside dans la capacité de cette dernière de «se souvenir » des événements antérieurs : unemême combinaison des entrées, à un certain instant, pourra avoir des effets différentssuivant les valeurs des combinaisons précédentes de ces mêmes entrées.

Pour traduire cet effet de mémoire on introduit la notion d'état interne de la fonction,l'action des entrées est alors de provoquer d'éventuels changements d'état, la situation quisuit le changement de l'une des entrées dépend de l'état précédent. Si le nouvel état estdifférent du précédent on dit qu'il y a eu une transition.

II.6.1 Les machines synchrones à nombre finis d'états:

Une machine à états (M.A.E.) en anglais Finite State Machine (F.S.M.) est un systèmedynamique, qui peut se trouver, à chaque instant, dans une position parmi un nombre fini depositions possibles. Elle parcourt des cycles, en changeant éventuellement d'état lors destransitions actives de l'horloge. L'architecture générale d'une machine à état est présentéeci-dessous dans la figure. II.6:

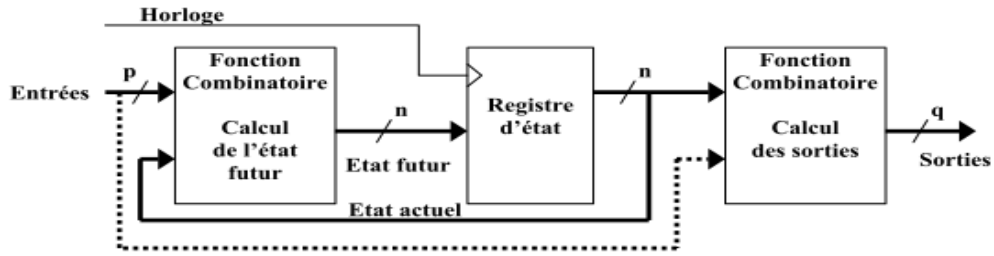


Fig. II.6*:Les machines synchrones à nombre finis d'états.*

II.6.1.1 Horloge, registre d'état et transitions:

Le registre d'état, piloté par son horloge, constitue le cœur d'une machine à états. Lesautres blocs fonctionnels sont à son service.

Il est constitué de n bascules synchrones. Son contenu représente l'état actuel de lamachine. Il s'agit d'un nombre codé en binaire sur n bits. L'entrée du registre d'état constituel'état futur, celui qui sera chargé lors de la prochaine transition active de l'horloge. Leregistre d'état est la mémoire de la machine.La taille du registre d'état fixe le nombre d'états accessibles. Si **n** est le nombre debascules, le nombre d'états$N = 2^n$.

Le rôle de l'horloge est de fixer les instants où les transitions entre états sont prises encompte. Entre deux fronts consécutifs de l'horloge, la machine est figée en position mémoire.

II.6.2 Les modèles de machine :

Il existe deux familles de machines d'état :

II.6.2.1 La machine de Moore : [14]

Le principe de ce modèle de machine (Figure II.7)réside dans le fait que les sorties ne dépendent que de l'état présent**n**.

Le circuit combinatoire d'entrée déterminel'état futur n+1 (ou l'entrée des bascules D) à partir des entrées et de l'état présent n.

Les sorties sont une combinaison logique de la sortie du registre d'état et changent de manière synchrone avec le front actif de l'horloge. L'inconvénient de cette machine, c'est son temps de réaction à un changement sur ses entrées qui est égale à une période de l'horloge dans le pire des cas.

Le modèle de Mooreest un modèle généralement représenté par les deux équations suivantes:

$$\text{Etat futur } (Q^{+}) = F(\text{entrées}(E), \text{état présent}(Q))$$
$$\text{sorties}(Z) = G(\text{état présent}(Q))$$

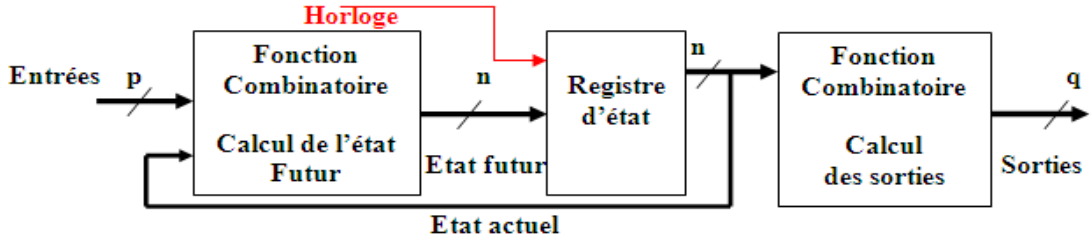


Fig. II.7*:La machinede Moore.*

II.6.2.2 la machine de Mealy : [14]

Le principe de ce modèle de machine réside dans le fait que les sorties dépendent de l'état présent **n** mais aussi de la valeur des entrées. La sortie peut donc changer de manière asynchrone en fonction de la valeur des entrées. Son temps de réaction à un changement sur $Q_i$est bien meilleur (c'est un temps de propagation combinatoire). Il existe une variante synchrone de la machine de Mealy avec un registre placé sur les sorties et activé par l'horloge. Toutefois, la machine de Moore est plus adaptée pour réaliser une machine d'état totalement synchrone.

Le modèle de Mealyest un modèle généralement représenté par les deux équations suivantes:

$$\text{Etat futur } (Q^{+}) = F(\text{entrées}(E), \text{état présent}(Q))$$
$$\text{sorties}(Z) = G(\text{entrées}(E), \text{état présent}(Q))$$

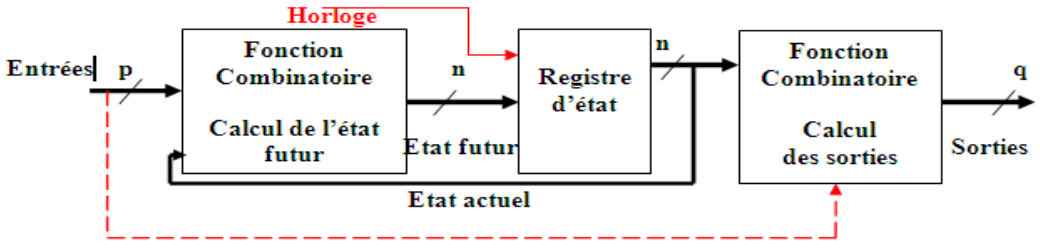


Fig. II.8*:La machine de Mealy.*

II.6.3 Outils de description:

Si l'outil d'analyse et de synthèse des fonctions combinatoires est la table de vérité, lediagramme de transition constitue l'outil privilégié pour l'analyse et la synthèse des fonctionssséquentielles.

II.6.3.1 Le diagramme de transition: [10]

On associe à chaque valeur possible du registre d'état, une case.L'évolution du système est représentée par des flèches représentant lestransitions.

Pour qu'une transition soit activée il faut que les trois conditions suivantessoient vérifiées :

1. Le système se trouve dans l'état «source » considéré
2. La condition de réalisation sur les entrées est vraie
3. Un front actif de l'horloge survient.

Pour les machines de Moore les sorties évoluent après l'activation de la transition. Les valeurs des sorties seront représentées dans les cases du diagramme.

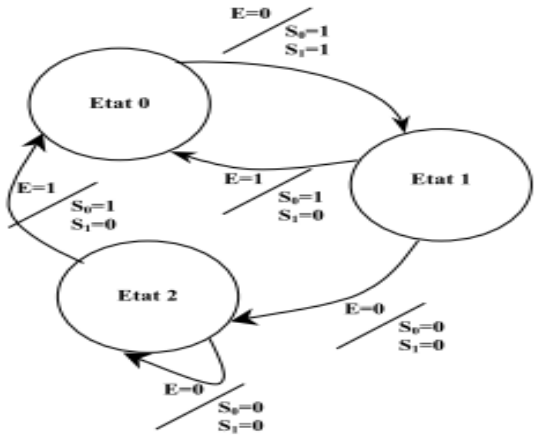


Fig. II.9*:Diagramme d'état quelconque pour La machine de Moore.*

Pour les machines de Mealy les sorties évoluent après l'évolution des entrées. Lesvaleurs des sorties seront représentées sur les flèches du diagramme.

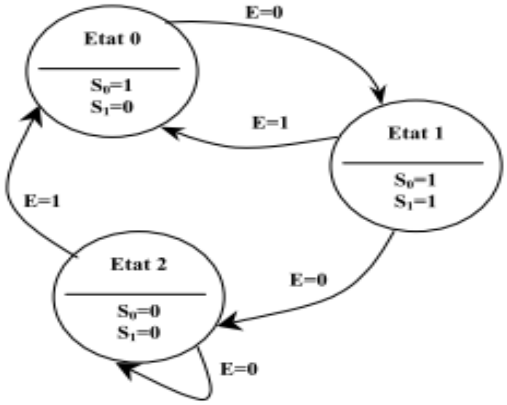


Fig. II.10*:Diagramme d'état quelconque pour La machine de Mealy.*

II.6.3.2 Passage du diagramme aux équations :

Le passage du diagramme de transition aux équations est indispensable si on veutsynthétiser la machine à états avec des circuits standards. L'outil permettant le passageexhaustif du diagramme aux équations est la table de transitions et d'états.

C'est une table de vérité constituée :

- ***En entrée***
  - ✓ de l'état actuel du registre d'état
  - ✓ des entrées de la machine à états
- ***En sortie***
  - ✓ de l'état futur du registre d'état
  - ✓ des sorties de la machine à états

Les équations des sorties du registre d'état sont ensuite adaptées au type de basculesutilisées.

Comme pour les fonctions combinatoires la complexité du problème croit de façonexponentielle avec le nombre d'états et le nombre d'entrées.

II.7 MODÈLE STRUCTUREL : MACHINE DE MEALY : [4]

La réalisation d'une fonction séquentielle quelconque repose sur une structure appelée machine de Mealy, proposée par le mathématicien du même nom. Cette structure permet de matérialiser le comportement d'une MEF.

La suite de ce paragraphe décrit la démarche permettant de passer d'unedescription comportementale d'un système séquentiel (modèle des MEF) à sa description structurelle sous forme de machine de Mealy.

L'idée de base consiste à **coder**, c'est-à-dire à matérialiser, les états de l'MEF par un vecteur de**n**variables booléennes $Q = (Q_0, \dots, Q_{n-1})$, Les variables $Q_i, i = 0 \dots n-1$sont appelées **variables internes**. Chaque état est codé par une seule combinaison de ces variables. Pour coder un MEF à **N états** il faut donc au moins **n** variables internes, si **n** est le plus petit entier vérifiant$N = 2^n$.

La tâche suivante consiste à caractériser les différentes transitions entre les états et à calculer les sorties de la machine pour chaque transition. Les deux graphes dansles figures (*Fig. II.9*, *Fig. II.10*) montrent que le franchissement d'une transition inter-état (i. e. d'un arc) est conditionné par deux entités :

- d'une part, son état courant (ou **état présent**) Q, représenté par une combinaison des variables internes Q_i .
- d'autre part, une combinaison des variables d'entrée de la machine.

L'état d'arrivée d'une transition, encore appelé **état suivant** ou **état futur**, est également fonction des variables d'entrée et des variables internes. Soit F cette fonction. Si, pour une transition donnée, Q est le vecteur représentant l'état courant, Q^+le vecteur représentant l'état futur, et E le vecteur constitué des variables d'entrée de l'automate permettant le franchissement de la transition, F est une fonction combinatoire de Q et E :

$$\text{Etat futur } (Q^+) = F(\text{entrées}(E), \text{état présent}(Q))$$

La fonction F est dite fonction **« état suivant** » ou **fonction d'excitation secondaire**. Cette fonction étant combinatoire, on sait la réaliser, quelle que soit sa complexité.

La lecture du graphe d'états montre que le vecteur Y des variables de sortie se calcule également en fonction des variables d'entrée et de l'état courant de l'automate. La fonction G permettant d'obtenir Z est donc également une fonction combinatoire de Q et E :

$$\text{sorties}(Z) = G(\text{entrées}(E), \text{état présent}(Q))$$

La fonction G est dite fonction de sortie.

Afin de réaliser les fonctions F et G, il est nécessaire de pouvoir accéder à tout instant à l'état interne courant de la machine. Il faut, pour cela, être capable de stocker les variables internes.

II.7.1 Méthode de synthèse d'Huffman-Mealy:

Dans cette partie, nous généraliserons les concepts évoqués précédemment afin de permettre le passage d'un cahier des charges quelconque au circuit correspondant. De plus nous établirons des règles de minimisation permettant d'optimiser le nombre de bascules utilisées pour la réalisation du circuit.

La méthode proposée, connue sous le nom de méthode d'Huffman-Mealy pour la synthèse des systèmes séquentiels synchrones,se décompose en plusieurs étapes. Ces étapes sont les suivantes :

1 : Modélisation du cahier des charges

- ✓ Choix et dénomination des variables primaires d'entrée et des de sorties.
- ✓ Construction du graphe de fluence à partir de l'énoncé du problème.
- ✓ Etablissement de la matrice des phases primitive, ou table d'états.

2 : Minimisation du nombre d'états

- ✓ Contraction de la matrice des phases. Cette opération se fait surtout pour éliminer les états redondants.

3 : Codage

- ✓ Choix et codage des secondaires, leur codage ne pose aucun problème puisqu'il n'y a pas de case critique à craindre.
- ✓ Etablissement de la matrice des excitations.

4 : Synthèse

- ✓ Définition des entrées et des éléments de mémoires, C'est une particularité aux synchrones. La matrice des excitations ne définit pas directement des excitations mais des variables d'entrée des éléments des mémoires de rétroaction. La détermination des tables de Karnaugh qui définissent ces variables dépend de la nature des bistables de rétroaction.
- ✓ Définition des variables de sortie, ou des fonctions de sortie.
- ✓ Le logigramme et la réalisation du circuit dans la technologie adoptée.

II.7.2 Solution selon le modèle de Moore: [4]

Nous avons appliqué la synthèse d'un circuit séquentiel synchrone destiné à reconnaitre une de trois bits par exemple '101'.On utilisera les deux méthodes d'Huffman-Mealy et Moore.

La sortie Z sera égale à 1, (Z=1), chaque fois que cette séquence apparaitra à l'entrée X du système. Puis la sortie revient à 0 avec l'arrivée du bit suivant.

On va s'intéresser uniquement à la synthèse de la machine de Moore qui sera utilisée par la suite dans le chapitre IV . On considère un circuit décrit par l'énoncé suivant :

1 : Modélisation du cahier des charges:

Choix et identification des variables primaires et des fonctions de sortie:

D'après l'énoncé, on a une variable principale d'entrée X et la fonction de sortie Z.

Graphe d'état ou de fluence selon le modèle de Moore:

On notera les états par A, B, C,… d'une façon arbitraire suite à la variation de l'entrée portée par la flèche. A l'inferieur des cercles, on portera l'état et la sortie, par exemple A/Z.

Soit A l'état initial ou l'état de répons.

Soit B l'état où il y a détection d'un début de séquence.

Soit C l'état où il y a détection de la suite de séquence.

Soit C l'état où il y a détection de séquence

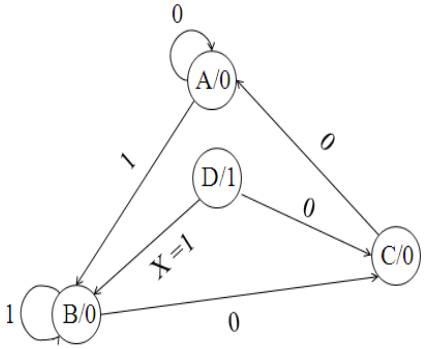


Fig. II.11*: *Diagramme d'état selon La modèle de Moore.

✓ *Etablissement de la matrice des phases primitive, ou table d'états :*

D'après le graphe d'état, on obtient la table d'état.

Etats présent	Etats futurs		Sortie
	X=0	X=1	Z
A	A	B	0
B	C	B	0
C	A	D	0
D	C	B	1

***Tableau II.1**: Table d états primitive de Moore.*

2 : Minimisation du nombre d'états

✓ *Contraction de la matrice des phases.*

Comme règle générale, deux lignes d'états présents sont équivalentes si pour chaque combinaison d'entrée, elles ont les mêmes états suivants et les mêmes sorties .Par conséquent les états présents de ces deux lignes sont équivalentes. Lorsque plusieurs états sont équivalents, on ne garde qu'un seul état qui les représente et on renomme les états suivants en conséquence.

L'objectif de la contraction (ou réduction) est de minimiser les variables internes et ainsi avoir le nombre de lignes égale à 2^n où n est le nombre de variable internes.

Si la contraction donne un nombre de lignes inferieures à 2^n, on complétera par la valeur ϕ les lignes manquantes.

Dans notre exemple, aucune contraction n'est possible. Ona table d'état avec 4 lignes.

3 : Codage

✓ *Choix et codage des états:*

On a 4 lignes d'état, ou 4 états, on peut en déduire le nombre 'n' de variables d'état (ou variables secondaires):n=2. On appelle Q_1, Q_2 ces variables d'états qui sont les sorties de deux bascules. On aura la table d'état codée ou la matrice des excitations suivante.

✓ *Etablissement de la matrice des excitations :*

Etats présents	Etats futurs		Sortie
	X=0	X=1	
	Q_1Q_2	Q_1Q_2	Z
(A): 00	00	01	0
(B): 01	11	01	0
(C): 11	00	10	0
(D): 10	11	01	1

***Tableau II.2**: Table matrice des excitations de Moore.*

4 : Synthèse

✓ *Définition des entrées des éléments mémoires:*

On peut donner les deux tableaux de Karnaugh en fonction des transitions aux excitations.

Q_1

Q_1Q_2	0	1
00	S_0	S_0
01	T_1	S_0
11	T_0	S_1
10	S_1	T_0

Q_2

Q_1Q_2	X 0	1
00	S_0	T_1
01	S_1	S_1
11	T_0	T_0
10	T_1	T_1

***Tableau II.3**:Tableaux de Karnaugh en fonction des transitions aux excitations de Moore.*

Du fait que la réalisation se sera avec des bascules D, on peut donner les tableaux de Karnaugh des entrées D_1, D_2 de ces deux bascules. Ces deuxdernières ont les expressions ci-dessous.

D_1

Q_1Q_2 \ X	0	1
00	0	0
01	1	0
11	0	1
10	1	0

$$D_1 = \overline{Q_1}Q_2\bar{X} + Q_1Q_2X + Q_1\overline{Q_2}\bar{X}$$

$$D_1 = \bar{X}[Q_1 \oplus Q_2] + Q_1Q_2X$$

D_2

Q_1Q_2 \ X	0	1
00	0	1
01	1	1
11	0	0
10	1	1

$$D_2 = \overline{Q_1}Q_2 + Q_1\overline{Q_2} + \overline{Q_2}X$$

$$D_2 = Q_1 \oplus Q_2 + \overline{Q_2}X$$

***Tableau II.4**: Tableaux de Karnaugh des entrées les bascules D_1, D_2 de Moore.*

✓ *Définition des variables de sortie, ou des fonctions de sortie :*

Z

Q_1Q_2	Z
00	0
01	0
11	0
10	1

$$Z = Q_1 + \overline{Q_2}$$

***Tableau II.5**: Tableau de Karnaugh de la fonction de sortie Z.(Moore)*

✓ *Logigramme et réalisation:*

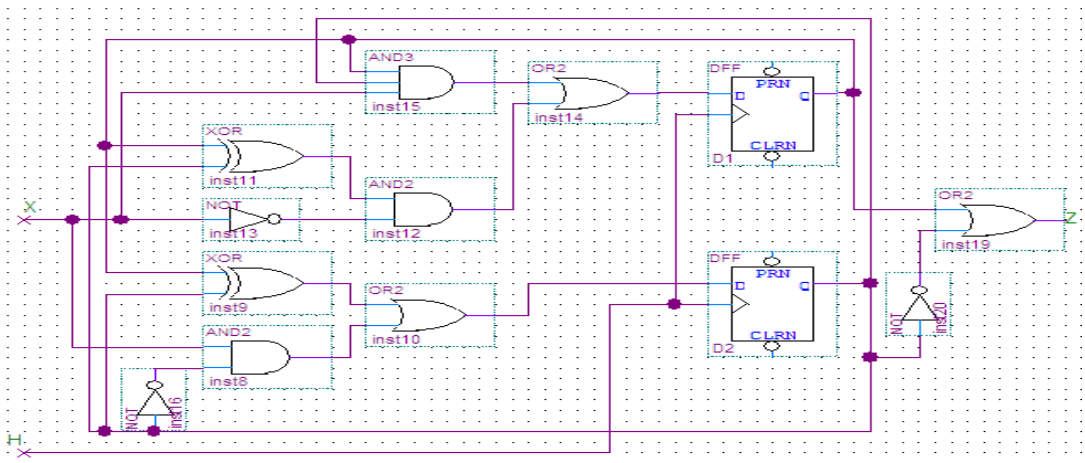


***Fig. II.12**:Logigramme selon La modèle de Moore.*

II.8 CONCLUSION:

Dans ce chapitre nous avons donnés un brefaperçu sur les systèmes séquentiels asynchrones et synchrones. Seule la synthèse selon la machine a été développée dans le but d'être appliquée par la suite pour la synthèse du codeur et du décodeur AMI.

C'est cette méthode qu'on va qualifier de manuelle comparativement aux méthodes qui utilisent le langage VHDL. Avec le logiciel Quartus on peut faire la saisi graphique du schéma trouvé, pour passer à la simulation. Mais aussi on peut faire la synthèse en faisant la saisie par le graphe de transitions de la machine d'état ou la saisie textuelle qui représente la description VHDL de la machine de Moore.

C'est l'objet du chapitre suivant.

CHAPITRE III:
DESCRIPTION VHDL POUR LES MACHINES D'ETATS

III.1 INTRODUCTION:

Nous pouvons aborder la description de systèmes séquentiels. Nous serons ensuite capable dedécrire tous les types de systèmes numériques. Nous pourrons décrire unsystème complexe en le décomposant. Nous pouvons utiliser un ou plusieurs niveaux de description structurelle. Dans ce chapitre nous montrerons des exemples de description de différents systèmes séquentiels.

Le langage VHDL ne dispose pas de déclaration explicite d'un signalde type registre. Nous devrons décrire le comportement de l'élément mémoire souhaité. Cette description sera ensuite interprétée par le synthétiseur. L'élément mémoire final obtenu peut-être différent de celui souhaitési la description est ambiguë. Nous dirons dans ce cas que la descriptionnon synthétisable. La description en VHDL d'un élément mémoire est trèssensible. Nous donnons la description type pour les principaux types d'éléments mémoires. [14]

Ces exemples respectent la norme IEEE Std 1076.6(Standard for VHDL Registres Transfer Level Synthesis).Nous allons commencer par la description d'éléments mémoires simples. Ensuite nous donnerons la structure de base pour décrire un systèmeséquentiel asynchrone.

III.2 SYNTHESE D'UNE DESCRIPTION VHDL:[11]

La synthèse d'une description VHDL représente une étape essentielle dans le processus deconception d'un composant programmable ou d'un ASIC. Le résultat de cette synthèse vadéterminer l'ampleur ou, tout au moins, les caractéristiques du composant cible. Il est doncimportant et même crucial, dans la plupart des cas, de savoir orienter une synthèse de façon à enoptimiser le résultat.

Dans les deux paragraphes qui suivent, vous trouverez quelques éléments de réflexionconcernant la synthèse de fonctions simples. En appliquant ces notions à des descriptions pluscomplexes, vous parviendrez certainement à optimiser votre description suivant vos critères.

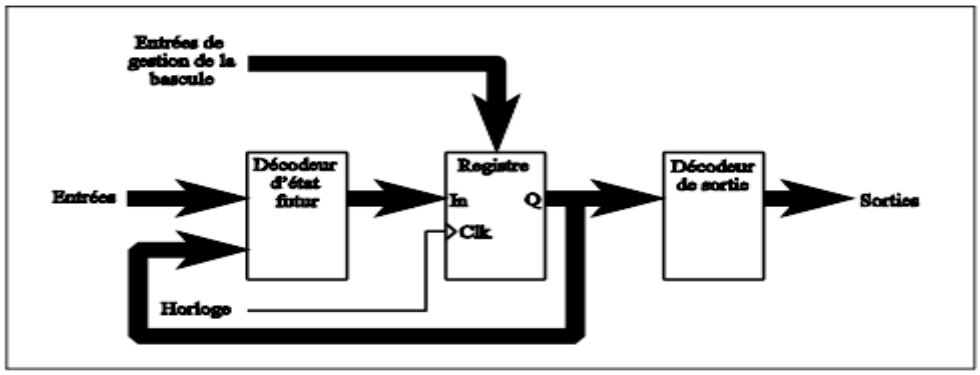


Fig. III.1 **:***Les machines synchrones à nombre finis d'états.*

III.2.1 Fonctions combinatoires: [14]

Une fonction combinatoire est une structure pour laquelle chaque combinaison d'entréefait correspondre en sortie une combinaison déterminée et indépendante des combinaisonsd'entrées et de sorties précédentes. On peut donc décrire ces structures à l'aide d'équationsbooléennes qui seront toujours vérifiées comme le sont les instructions concurrentes. En effet,l'instruction "S <= A and B or C" est toujours vérifiée et correspond à l'équation booléenne

S =A.B + C.

Mais il est aussi possible de décrire une structure combinatoire en utilisant lesinstructions séquentielles d'un process. L'équation précédente peut, par exemple, s'écrire :

```
Process (A, B, C)
Begin
If (C = '1') then S <= '1';
Elsif (A = '1' and B = '1') thenS <= '1';  <----------  S <= A and B or C
Else S <= '0';
End if;
End process;
```

L'issue de synthèse de ces deux types de description est identique, fort heureusement. Toutefois,on peut être amené à se poser la question de savoir quel type de description est à utiliser.

Enréalité, même si le résultat de la synthèse est identique, il y a une différence fondamentale entreces deux descriptions. L'une n'est que le reflet des équations logiques de la structure combinatoireà décrire, alors que l'autre se base sur l'expression de son comportement.

Pour tenter de résumer les différences existant entre ces deux méthodes de description desfonctions combinatoires, on peut lister les avantages et les inconvénients de chacune d'elles.

- ***Description de fonctions combinatoires à l'aide d'instructions concurrentes :***

Avantage :

- ✓ la description obtenue est lisible et simple,
- ✓ la description obtenue reflète bien la réalité.

Inconvénients :

- ✓ décrire une fonction combinatoire en utilisant des instructions concurrentes sous-entend que lataille de sa table de vérité le permet ou que les simplifications de cette dernière ont été faites.
- ✓

- ***Description de fonctions combinatoires à l'aide de process :***

Avantage :

- ✓ pour des fonctions compliquées dans lesquelles les équations possèdent de nombreux termes,cette méthode est parfois plus simple. Lorsque la description comportementale est plus simpleque la description algorithmique, cette méthode est avantageuse ;
- ✓ il n'est pas nécessaire de faire les simplifications de la table de vérité, elles sont faitesautomatiquement par le synthétiseur lors de la synthèse.

Inconvénients :

- ✓ dans certains cas, cette écriture est peu lisible et par conséquent très difficilement modifiablepar une personne extérieure.

Il n'y a donc pas de règle générale mais plutôt la nécessité de procéder à une démarche logiqued'observation avant la description pour choisir la méthode de description adéquate.

III.2.2 Fonctions séquentielles:

La synthèse d'une fonction logique séquentielle est beaucoup plus complexe que celled'une fonction combinatoire. Les possibilités de description offertes par le langage VHDL sontvastes et bien souvent irréalisables dans l'état actuel des technologies.

il existe des attributs qui sont très utilisés et qui représentent même unvéritable standard. La description d'un front est très simple à réaliser si l'on utilise l'attribut éventassocié à une condition de niveau pour la détection d'un front montant ou descendant. L'exempleci-dessous qui correspond à la description d'une bascule D ne posera donc aucun problème desynthèse.

III.3 DESCRIPTION D'UN SYSTEME SEQUENTIEL SYNCHRONE:

Nous allons donner la structure de base pour décrire n'importe quel système séquentiel synchrone. Cette structure est rigide afin de garantir unebonne traduction de l'élément mémoire. Il est recommandé de respectercette structure. Celle-ci garanti une synthèse correcte. Il existe d'autresstructures correctes. Notre proposition permet de limiter les risques d'erreurs et d'assurer une bonne lisibilité de la description.

Un système séquentiel se décompose en 3 blocs:

- Le décodeur d'état futur (combinatoire).
- Le registre (séquentiel).
- Le décodeur de sorties (combinatoire).

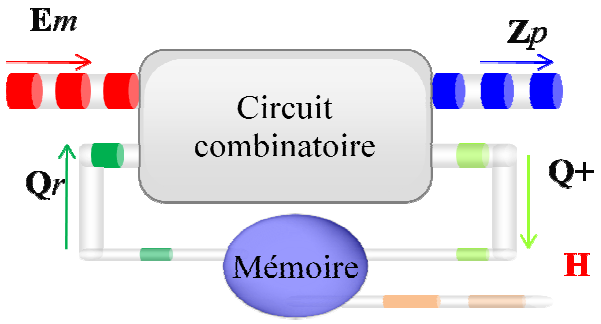


***Fig.III.2**: Schéma bloc d'un système séquentiel synchrone.*

Voici la forme de base de la description en VHDL synthétisable d'unsystème séquentiel synchrone:

```
Library IEEE;
Use IEEE.Std_Logic_1164.all;
Use IEEE. Bibliothèques_utiles.all;
Entity Nom_Du_Module is
Port (Nom_Entrée : in  Std_Logic;
....
            Nom_Sortie : out Std_Logic);
End Nom_Du_Module;
ArchitectureType_De_Description of Nom_Du_Module is
Signal Etat_Present, Etat_futur : Std_Logic_Vector (N-1 downto 0);
Begin
--------------------------------------------------------------------------
-- Description décodeur d'etat futur (combinatoire)
--------------------------------------------------------------------------
Etat_futur <=  .... When Condition_1 else
                    .... When Condition_2 else
                    ...;
-- processes decrivant la memorisation (register)
--------------------------------------------------------------------------
   Mem: process (Horloge, Actions_asynchrone)
   Begin
      If (Action_asynchrone = '1') then
      Etat_Present<= Etat_Initial;
Elsif Rising_Edge (Horloge) then
Etat_Present<= Etat_futur;
End if;
End process;
--------------------------------------------------------------------------
-- Description décodeur de sorties (combinatoire)
--------------------------------------------------------------------------
Sortie <=.... when Etat_present = Etat_i else
            .... when Etat_present = Etat_j else
            ....;
End Type_De_Description;
```

III.3.1 Description d'un 'Flip-Flop D': [9]

Le vhdl permet de décrire des fonctionnements parallèles. La 'brique' de base de ces descriptions est le process. Le process est une construction qui ne s'exécute que sur un changement détecté dans l'un des signaux auxquels il est sensible.

Nous allons commencer par la description d'un Flip-Flop D (DFF).C'est l'élément mémoire le plus utilisé. Il est à la base de tous les systèmesséquentiels synchrone. Le comportement est celui d'une bascule de type Dsensible au flanc. Celle-ci dispose d'une remise à zéro asynchrone (reset).

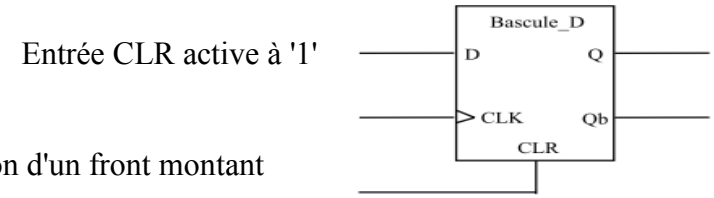


Fig.III.3*: Symbolede la bascule D flip-flop.*

La description en VHDL du flip-flop D est basée sur cette analyse.

```
Library IEEE;
Use ieee.std_logic_1164.all;
Entity bascule_Dff1is
Port (D, clk, clr: in std_logic; Q, Qb: out std_logic);
End bascule_Dff1;
Architecture bascule_Dffof bascule_Dff1is
BEGIN
Process (clk, clr)
Begin
        If (clr = '1') then Q <= '0';
                            Qb <= '1';
Elsif (clk'event and clk = '1') then Q <= D;
                                     Qb <= D;
        End if;
End process;
End bascule_Dff;
```

La description de la bascule *D* repose sur le process, Ce process est sensible à l'entrée *clk*. Il ne s'exécute que s'il y a un changement sur l'entrée *clk*.

Lors de l'exécution les instructions composant le process s'exécutent séquentiellement.

Les sorties du process ne sont affectées qu'à la fin du déroulement de celui-ci.

Nous voyons ici l'utilisation d'un attribut du signal clk. Il s'agit de l'attribut ***évent,*** celui-ci est vrai lorsqu'un changement sur le signal ***clk*** vient de se produire.

Ne pas oublier que les valeurs possibles du signal clk sont: 'U' 'X' '0' '1' 'Z' 'W' 'L' 'H' '-' si il est du type std_logic.

La condition logique clk'event and ***clk='1'*** détecte donc un front montant sur *clk*.

III.3.2 Description d'un Latch : [11]

Le comportement du latch correspond à une bascule de type sensible surun niveau. Cet élément mémoire n'est pas synchrone. Il est utilisé uniquement pour mémoriser des informations (mémoires, interface).

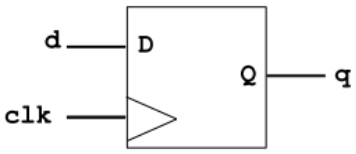


Fig.III.4*: Symbole de la bascule D latch.*

La description en VHDL synthétisable du latch D est basée sur cetteanalyse.

```
Library IEEE;
Use ieee.std_logic_1164.all;
Entitybascule_Dff1 is
    Port (d, clk: in std_logic;   q   : out std_logic);
Endbascule_Dff1;
Architecturebascule_Dffofbascule_Dff1 is
Begin
    p1: process (clk)
    Begin
        If (clk'event and clk='1') then
            q <= d;
End if;
    End process p1;
End bascule_Dff;
```

Remarque

La description du flip-flop D est simple. Il n'y a pas de décodeur d'étatfutur et de décodeur de sortie. La description comprend uniquement le processus de mémorisation.

III.3.3 Description d'une bascule T:

La bascule T (T pour Toggle, c'est à dire bascule) est un élément qui interprète sonunique entrée de commande (en plus de l'horloge, évidemment), T, non comme une donnée àmémoriser, mais comme un ordre de changement d'état :

⇒ Si T = "actif" changer d'état à la prochaine transition de l'horloge,

⇒ Si non conserver l'état initial.

D'où l'équation qui décrit son fonctionnement :

$$Q_+ = \mathrm{T} \oplus \mathrm{Q}$$

On peut éclairer l'équation précédente par le diagramme de transitions de la figure III.3,ci-dessous :

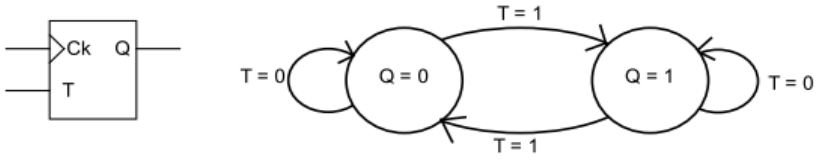


Fig.III.5*: Symbolede la bascule T avec diagramme d'états.*

La description d'une bascule T se déduit simplement du diagramme de transition :

```
Entity T_edge is
Port (T,hor: in bit; s: out bit);
End T_edge;
Architecture d_primitive of T_edge is
Signal etat : bit;
s <= etat;
Begin
Process
Begin
Wait until hor = '1';
If (T = '1') then
    Etat <= not etat;
End if;
End process;
End d_primitive;
```

III.3.4 Description d'une bascule JK:

Une bascule J-K dispose de deux commandes, J et K, outre l'horloge.Comme pour tout opérateur synchrone, l'état de la bascule ne change pas entre deuxtransitions actives du signal d'horloge.

Dans la construction du diagramme de transitions de la figure suivante, III-4, on a tenucompte de ce que chaque transition peut être obtenue par deux combinaisons différentes descommandes J et K, la transition '0' → '1', par exemple, peut être obtenue par mise à 1 explicite(JK = "10") ou par changement d'état (JK = "11") :

$$Q_+ = J.\overline{Q} + \overline{K}.Q$$

On peut éclairer l'équation précédente par le diagramme de transitions de la figure III.4,ci-dessous

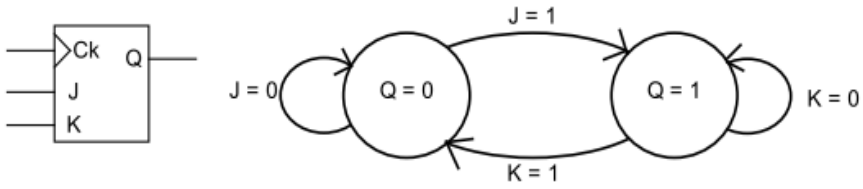


Fig.III.6*: Symbole de la bascule JK avec diagramme d'états.*

La description d'une bascule JK se déduit simplement du diagramme de transition :

```
Entity jk is
Port (j, k, clk: in bit;q: out bit);
End jk;
Architecture fsm of jk is
Signal etat: bit;
Begin
Process
Wait untilclock = '1';
If (j = '1' and k = '1') then etat <= not etat;
Elsif (j = '1' and k = '0') then etat<= '1';
Elsif (j = '0' and k = '1') then etat <= '0';
End if;
End process;
q <= etat;
End fsm;
```

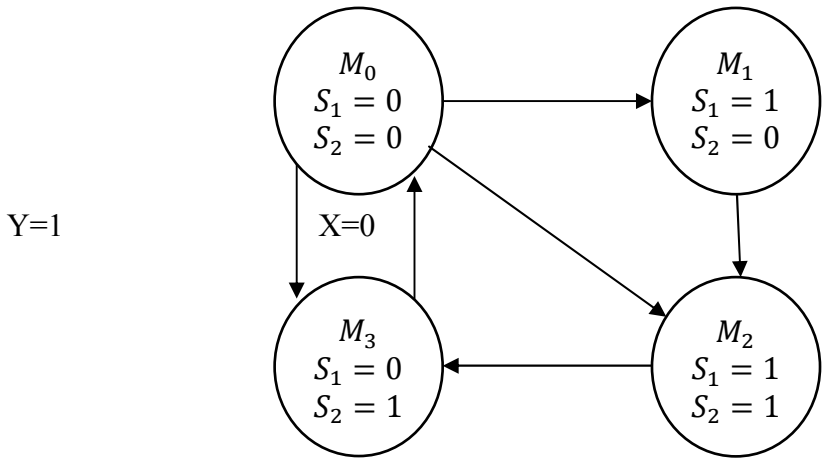
III.4 LA DESCRIPTION D'ETAT:[1]

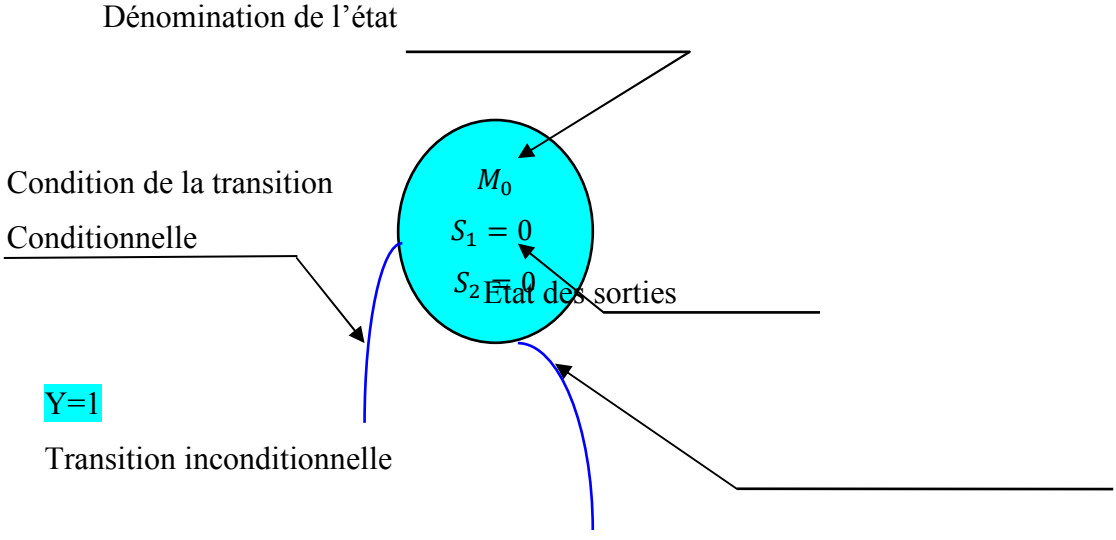
III.4.1 Description d'une machine séquentielle à partir d'un graphe d'état:

Un graphe d'états ou un organigramme se traduit tout naturellement en VHDL, en utilisantles structures CASE et IF…ELSE. La figure suivant représente un graphe d'état utilisée pour décrire des systèmes séquentiels quelconques (state machine) par exemple.

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.

III.4.2 Ecriture en langage VHDL:

On va traduire et essayer l'exemple ci-dessus. En langage VHDL une des méthodes conseillée d'écriture des machines d'état est :

* une ***entité***

* une ***architecture***de description d'état avec ***2 process*** :

Un process *asynchrone* et un process *synchrone.*

A. Définition de l'entité :

```
---- déclaration des librairies --
  Library ieee;
  Use ieee.std_logic_1164.all;

  Library SYNTH;
  Use SYNTH.vhdlsynth.all;
  ----- Définition de l'entité----
  ENTITY machine1 IS
  PORT (RAZ, horl : IN STD_LOGIC;  -- Ne pas oublier remise à 0 et horloge !
       X, Y: IN STD_LOGIC;
       S1, S2: OUT STD_LOGIC);
  END machine1;
```

B. Définition de l'architecture :

```
*LE PROCESS SYNCHRONE :
    ------ Définir L'architecture de description d'état ----------
    ARCHITECTURE diagramme OF machine1 IS
        TYPE etat_4 IS (M0, M1, M2, M3);      --définir une liste des états...
        SIGNAL etat, etat_suiv :etat_4 := M0;      --et 2 signaux: états courant et suivant, contenant la
                                                    valeur --d'un état de la liste (initialisée à M0).
    BEGIN
Definir_etat:PROCESS (raz, horl)                -- "Definir_etat":label optionnel
    BEGIN
                If raz = '1' THEN
    Etat <= M0;              --Le PACKAGE std_logic_1164
ELSIF Rising_Edge (horl) THEN        --en permet l'utilisation.
Etat <= etat_suiv ;                          --Mise à jour de la variable d'état par l'horloge.
END IF;
END PROCESS Definir_etat;
```

C. Définition des états des sorties :

```
*LE PROCESS ASYNCHRONE :
    ---------- Définir les états des sorties----------
Sorties : PROCESS (etat, x, y).
BEGIN
--Le PROCESS doit contenir une structure de CASEuniquedépendant de la variable d'état
CASE etat IS
    WHEN M0 => S1 <= '0'; S2<= '0';
IF Y='1' then etat_suiv <= M3;
Elsif X='0' thenetat_suiv <= M2;
ELSEetat_suiv <= M1;
END IF;--Le signal de l'état suivant n'est attribué que dans la structure CASE.
    WHEN M1 => S1 <= '1'; S2<= '0'; etat_suiv <= M2;
                WHEN M2 => S1 <= '1'; S2<= '1'; etat_suiv <= M3;
                WHEN M3 => S1 <= '0'; S2<= '1'; etat_suiv <= M0;
            END CASE;
    END process sorties;
    END diagramme ;          -- ne pas oublier de terminerl'architecture !
```

Dans la forme de base donnée ci-dessus, les décodeurs sont décrits avecl'instruction concurrente when ... else. Il est possible de décrire ces décodeurs avec des instructions séquentiels (dans un process) dans les cas plus complexes. Le décodeur d'état futur et le décodeur de sorties dépendant desmêmes entrées sont parfois réunis.

III.4.3 SIMULATION :

Définir les entrées :

Horl : période 200ns

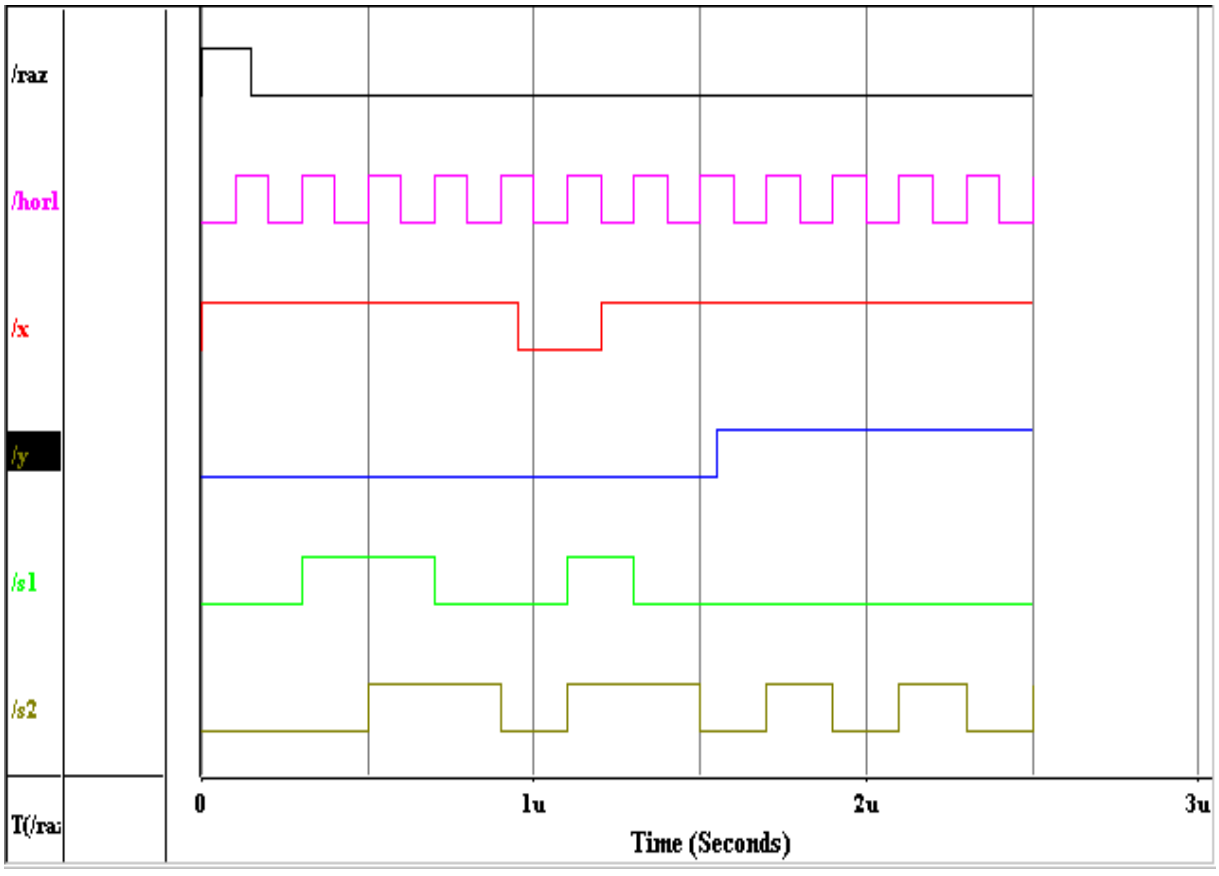
Raz à 0ns =1 et à 150ns =0

x à 0ns =1, à 950ns =0 et à 1200ns =1

Y à 0ns =0 et à 1550ns =1

Simuler pendant 2.5us

On obtient :

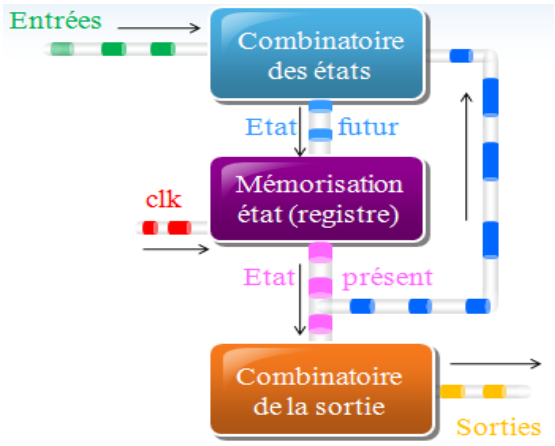


III.5 LES MODÉLES DE DESCRIPTION DES MACHINES D'ETAT:

Il existe deux familles de machines d'état :

III.5.1 Machine de Moore:

Voici la machine de Moore répondant au cahier des charges ci-dessus :

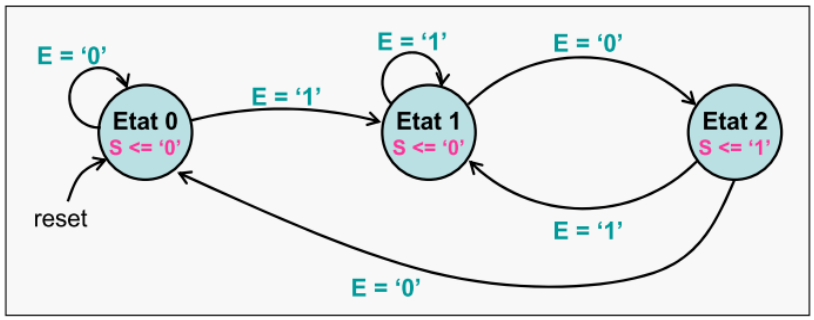


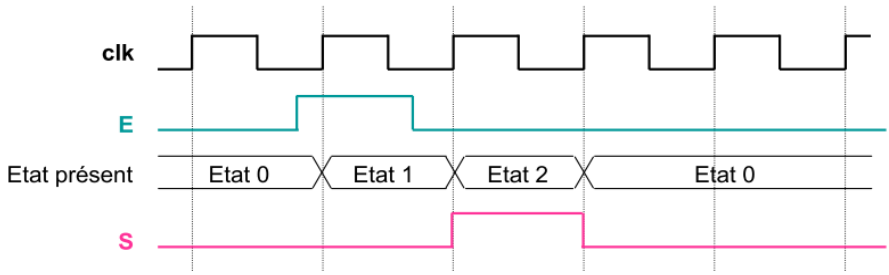
Exemple

• 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.

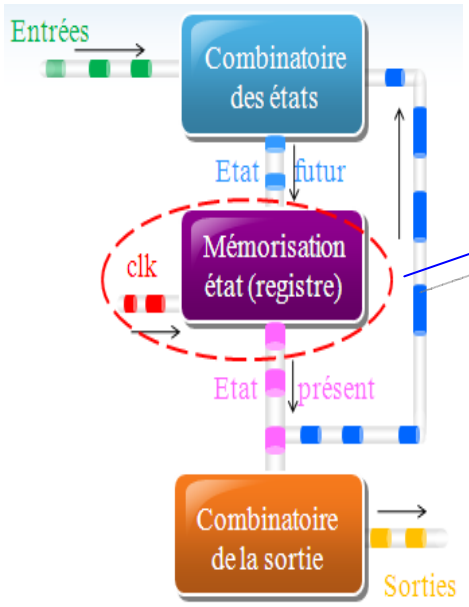
Machine de Moore reconnaissant la séquence 10 :





Et 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) :

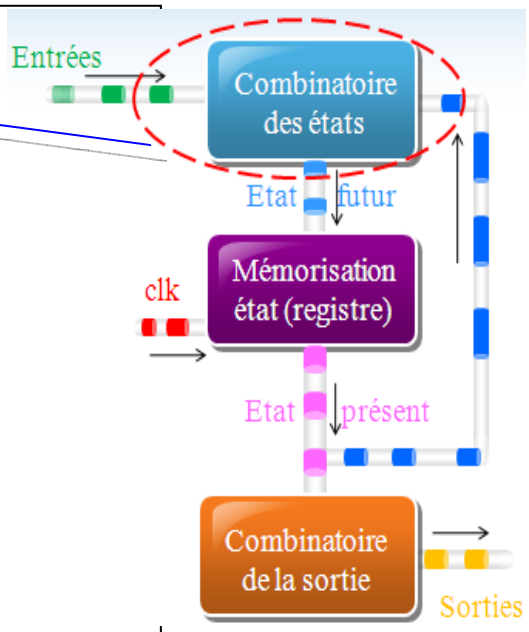



```
Type Etat is (Etat0, Etat1, Etat2);
Signal Etat_present, Etat_futur : Etat := Etat0;
```

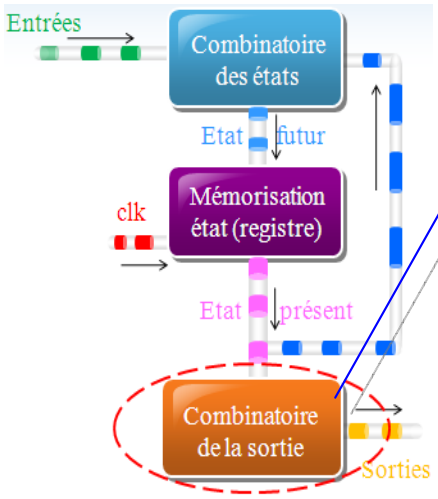
```
Sequentiel_maj_etat: process (clk, reset)
Begin
If reset = '0' then
Etat_present <= Etat0;
Elsif clk'event and clk = '1' then
Etat_present <= Etat_futur;
End if;
End process Sequentiel_maj_etat;
```

• 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 = '0' then Etat_futur <= Etat2;
ElseEtat_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 àpartir de l'état présent :




```
Combinatoire_sorties : process (Etat_present)
Begin
Case Etat_present is
When Etat0 => S <= '0';
When Etat1 => S <= '0';
When Etat2 => S <= '1';
End case;
End process Combinatoire_sorties;
```

Description avec 2 process

• Les 2 process combinatoires possèdent des listes de sensibilité « compatibles », ils peuvent donc être regroupés en un seul process afin d'abréger l'écriture.

⇒ 2 process = 1 process séquentiel + 1 process combinatoire

Description avec 1 process

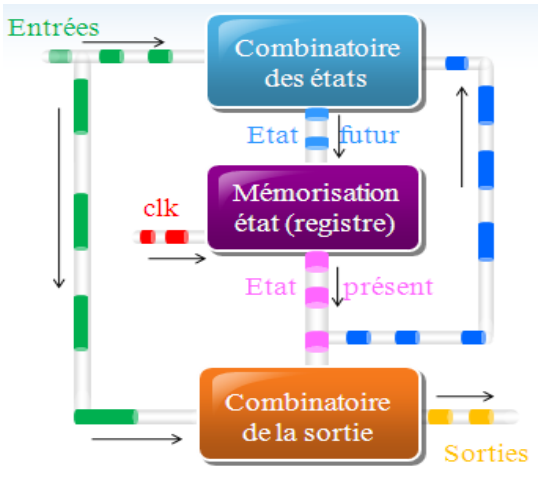
• Description la plus compacte en utilisant une variable pour l'état (en lieu et place des 2 signaux).

• Cependant perte de lisibilité lors de l'écriture. Alors que cette description n'apporte rien en termes de résultat de synthèse par rapport à une description 2 process.

⇒ À éviter (pt de vue personnel cependant …).

III.5.2 Machine de Mealy

Voici la machine de Mealy répondant au cahier des charges ci-dessus :



• 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).

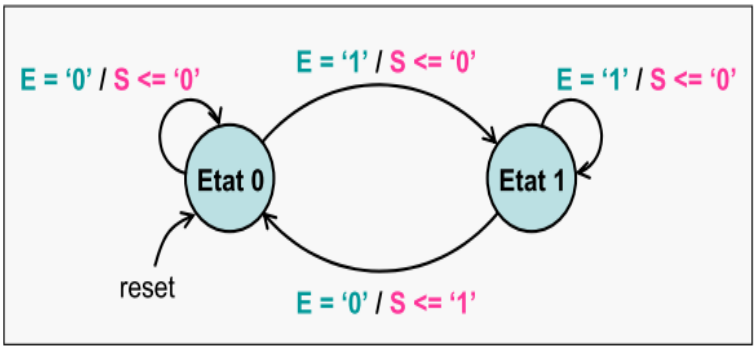
• La sortie dépend directement de l'entrée et ceci indépendamment de l'horloge (clk).

⇒ 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.

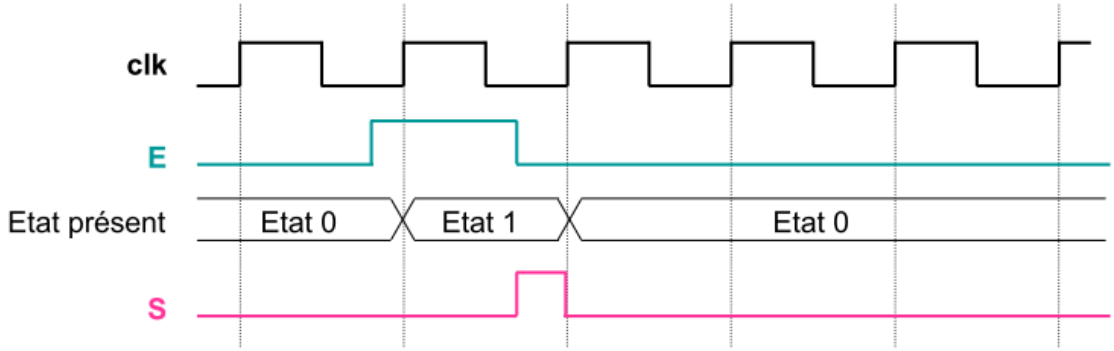
Exemple Machine de Mealy reconnaissant la séquence 10



E = '1' / S <= '0'

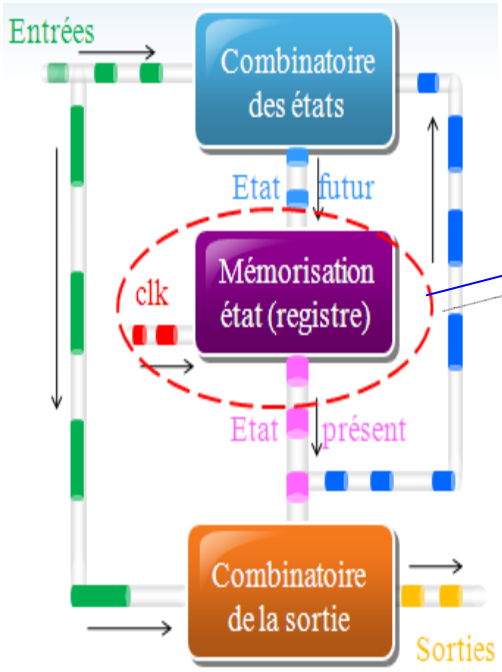
Condition de validation de la transition

Affectation de la valeur '0' à la sortie



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) :

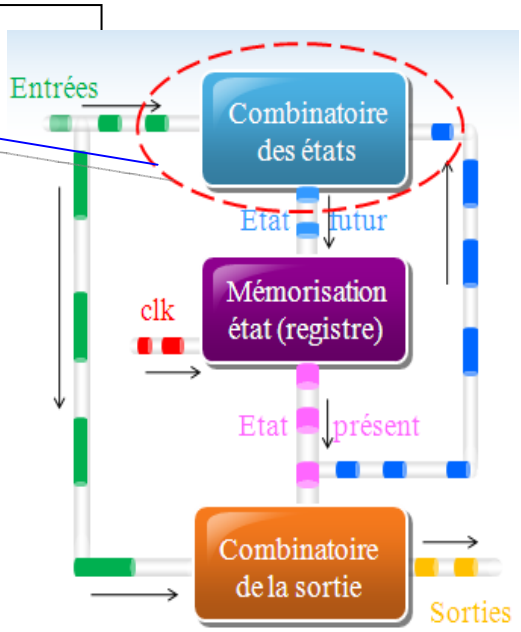



```
Type Etat is (Etat0, Etat1);
Signal Etat_present, Etat_futur : Etat := Etat0;
```

```
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;
```

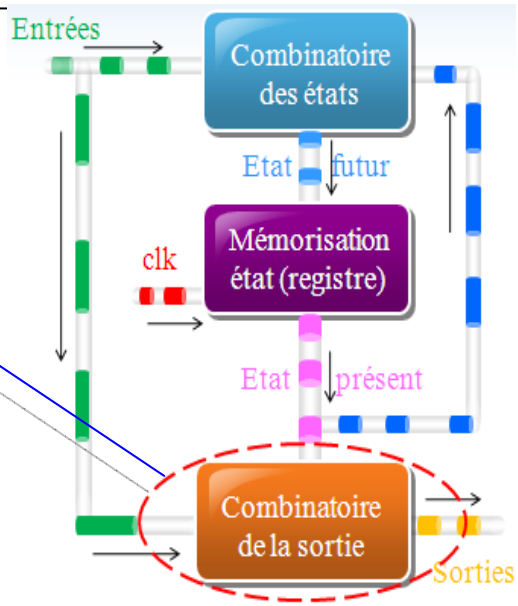
• Un process combinatoire de calcul de l'état futur à partir des entrées et de l'état présent :

```
Combinatoire_etats : process (E, Etat_present)
Begin
Case Etat_present is
When Etat0 =>if E = '1' thenEtat_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 àpartir des entrées et de l'état présent:

```
Combinatoire_sorties : process (E, Etat_present)
Begin
Case Etat_present is
When Etat0 => if E = '1' thenS <= '0';
Else
S <= '0';
End if;
When Etat1 =>if E = '0' thenS <= '1';
Else
S <= '0';
End if;
End case;
End process Combinatoire_sorties;
```



III.6 CONCLUSION :

Dans ce chapitre nous avons vu le VHDL permet d'écrire le code en deux manières distinctes ; concurrentes ou séquentielles. Le code séquentiel correspond aux process. Chaque process possède une liste sensitivité, dans laquelle on nomme tous les signaux et utilisés pour le calcul des états futur de différents signaux.

Le code concurrent décrit aussi un comportement ou une logique hors d'un process. Il existe sur cette manière les machines d'états que l'utilisateur n'est plus contraint alors d'écrire du code VHDL, il lui suffit de dessiner les différents états et les relations pour passer d'un état à l'autre.

Dans le prochain chapitre nous allons voir comment le programme génère automatiquement le code VHDL associé au diagramme dessiné et peut ensuite être compilé pour la simulation, puis synthétisé et appliqué dans le circuit codec AMI qui est proposé dans cette mémoire.

CHAPITRE IV:
DESCRIPTION CODEUR ET DECODEUR AMI AVEC SYNTHESE VHDL

IV.1 INTRODUCTION :

Dans ce chapitre, nous allons étudier un code utilisé dans la transmission en bande de base. Ce code est adapté aux câbles métalliques où chaque élément binaire est transmis sous forme d'un niveau de tension ou d'une variation simple de niveau de tension.

Cette opération s'appelle le *codage en ligne*. Dans la figure IV.1, nous donnons le schéma bloc de la chaine de transmission qui utilise le codeur et le décodeur AMI qui sont l'objet de l'étude de ce projet :

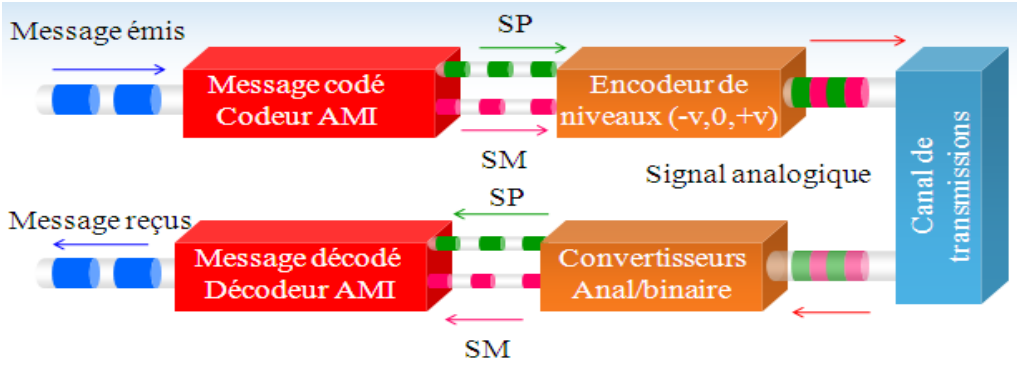


Fig. IV.1. *Schéma bloc de la chaine de transmission en bande de base dans le code AMI.*

On trouve ce principe dans les transmissions téléphoniques à grande distance où les informations, sous la forme numérique, sont transmises en série (un bit à la fois), au rythme d'une horloge. Le code binaire, avec ses deux valeurs 0 et 1, est transformé en un code à 3 niveaux de tension (+V, -V, 0) sur la ligne (câble coaxial, par exemple) :

⇒ Un **ZERO** logique correspond toujours à une tension nulle,

⇒ les niveaux logiques **UN** sont représentés par des impulsions, qui durent une période de l'horloge de transmission, alternativement positives et négatives, d'où le nom du code AMI (Alternate Mark Inversion). On notera que le système doit se « souvenir » de la polarité de la dernière impulsion transmise pour fonctionner correctement [13].

IV.1.1 Transmission en bande de base:

Un signal électrique, ou optique est caractérisé par sa densité spectrale de puissance (encombrement spectral). On peut le transporter dans la bande

d'origine avec un éventuel transcodage (bande de base) ou effectuer une transposition dans une autre bande de fréquence (modulation avec porteuse).

Un signal est dit "bande de base" s'il n'a pas subit de transposition en fréquence. La transmission en bande de base s'avère particulièrement simple et économique pour les signaux synchrones et rapides.

IV.1.2 Le codage :

La transmission directe du message n'est généralement possible que sur de très courtes distances. Un signal à transmettre subira donc un codage plus ou moins élaboré afin d'adapter son spectre au support utilisé : réduction ou suppression de la composante continue, transmission de l'horloge en synchrone ...[13].

Les codages utilisés peuvent être classés selon le nombre de niveaux électriques. On peut citer les codes à:

- 2 niveaux : NRZ, biphasés, Miller...
- 3 niveaux : bipolaires AMI, bipolaires haute densité...
- multi-niveaux : 2B1Q...

La figure IV.2 donne la représentation de quelques codes utilisés

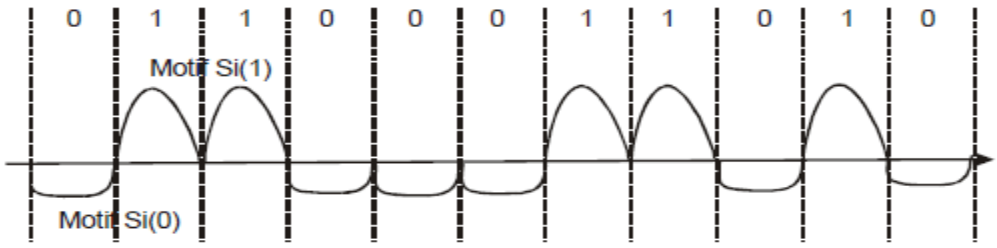


Fig. IV.2. *Signal numérique en bande de base.*

IV.2 DEFINITION DU CODE AMI ET SIMULATION :

C'est un code très répandu appelé aussi : code bipolaire (ou **AMI : A**lternate **M**ark **I**nversion).Un bit 0 est représenté par un niveau zéro pendant une période d'horloge , un 1 par un niveau alternativement +V ou –V. Il ne possède ni composante continue ni raie à la fréquence bit. Il est parfois remplacé par un code bipolaire RZ pour lequel la durée d'un niveau 1 est réduite à une demi période horloge. Dans ce cas, la restitution de l'horloge bit peut être obtenue par un simple redressement.

La création des impulsions alternées passe par un changement de code : le circuit du codeur aura comme entrées : l'horloge d'émission «CLK», le signal de mise à zéro « RESET» et l'entrée des données à transmettre« DIN».

En sortie, on a deux signaux binaires que nous nommerons « SP » pour l'état V_ plus et « SM » pour l'état V_ moins. Ainsi le circuit fonctionne de la manière suivante:

⇒ Si « RESET=1 » les sorties sont remises à zéro indépendamment de l'horloge « SP = '0' », « SM = '0' ».

⇒ Si l'entrée « DIN = '0' », les sorties restent à zéro « SP='0' », « SM = '0' ».

⇒ Si l'entrée est «DIN = '1' », les sorties deviennent « SP = '1' », « SM = '0' »

⇒ Si l'entrée donne un deuxième «DIN = '1' » les sorties changent pour devenir « SP = '0' », « SM = '1' » au front montant de l'horloge.

Les sorties changent pour chaque valeur « DIN=1 ». Ce fonctionnement est donné dans la figure IV.3:

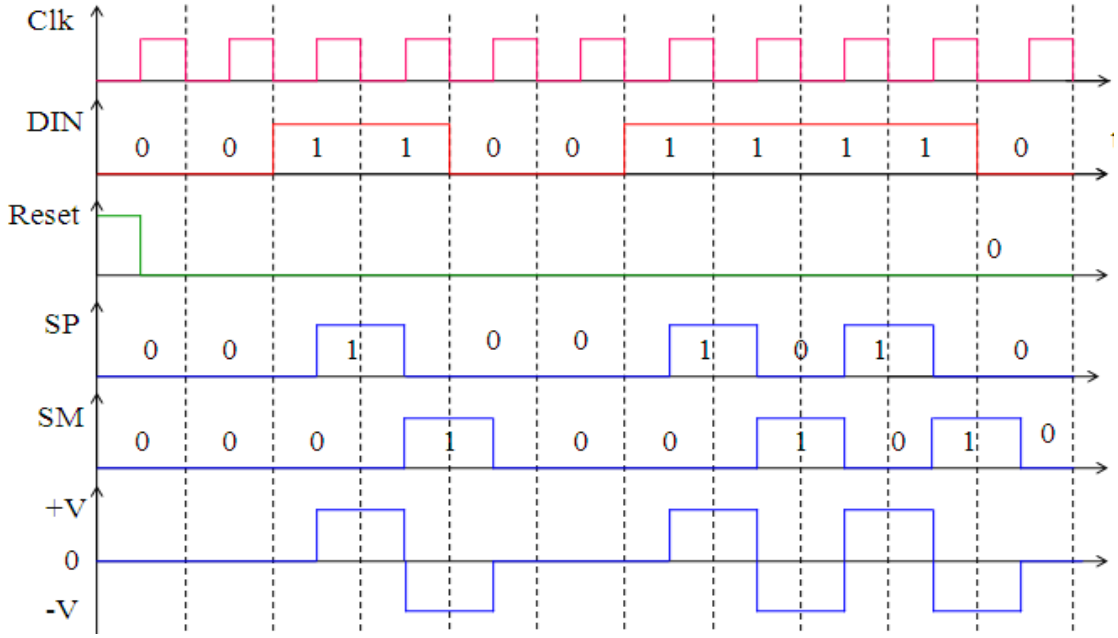


Fig. IV.3. *Le signal alterné du code AMI.*

IV.2.1 DIAGRAMME D'ETAT DU CODE AMI :

Le cahier de charge décrivant le fonctionnement de ce code permet d'obtenir le digramme d'état selon la figure IV.4.

Ce diagramme d'état est constitué de 4 états dénommés : Zero1, V_plus, V_moins, Zero2. Les valeurs prises par les sorties « SP » et « SM » correspondent au cas d'une

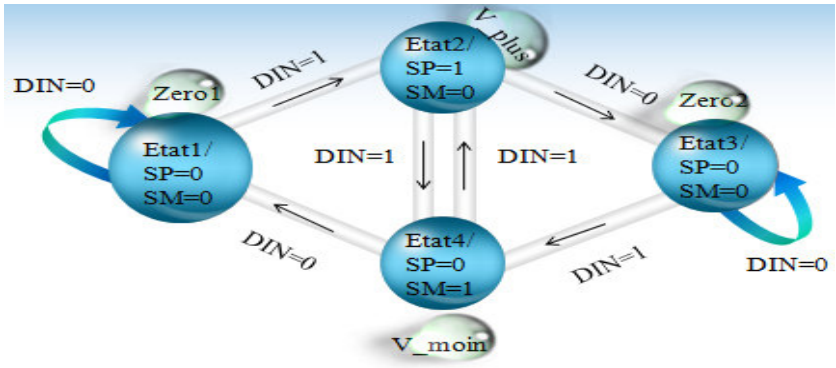


Fig. IV.4. *Diagramme d'état du code AMI.*

Machine de Moore où les valeurs dépendent des états. Ce sont les valeurs prises par les sorties après chaque impulsion d'horloge.

Le diagramme d'état ci-dessus qui a été proposé sera la base dans la synthèse du circuit du codeur AMI. Ainsi en utilisant le logiciel Quartus II.7.2 on peut faire la synthèse de différentes manières à savoir:

- ✓ La synthèse par la saisie textuelle en langage VHDL,
- ✓ La synthèse par la méthode classique et manuelle, suivie par une saisie graphique.
- ✓ La synthèse par la saisie du graphe d'état avec ses deux possibilités (voir annexe).

IV.2.2 DESCRIPTION DU CODEUR AMI AVEC SYNTHESE VHDL :

Ayant résumé le fonctionnement du code AMI par le diagramme d'état, on va procéder à la description du fonctionnement de ce circuit en langage VHDL par une entité « amicod » et une architecture «diagramme ».

L'entité est donc une boîte ayant :

- Les entrées : RAZ, DIN, clk
- Les sorties : SP, SM.

La description en VHDL est donnée par le programme de la page suivante.

L'une des méthodes conseillée pour décrire des machines d'état est basée sur:

* une ***entité***

* une ***architecture*** de description d'état avec ***2 process*** :

L'entité nommée par « amicod » regroupe les entrées «RAZ», « DIN », « clk » et les sorties « SP », « SM ». Dans l'architecture, on commence par déclarer les types

```
library ieee ;
Use ieee.std_logic_1164.all ;
Entity amicod is
Port ( RAZ, DIN, clk: in std_logic;
        SP, SM: out std_logic);
end amicod ;
Architecture  diagramme  of  amicod  is
Type etat is (Zero1, V_moin, Zero2, V_plus) ;
Signal etat_present, etat_futur : etat :=Zero1 ;
begin
Description_etat: process (clk, RAZ)
begin
If RAZ='1' then
    etat_present<= Zero1 ;

elsif
    clk ‘event and clk='1' then
    Etat_present <= etat_futur ;
end if ;
end process Description_etat;
Sortie: process (DIN, etat_present)
Begin
  Case etat_present is
        When  Zero1 =>  Sp <= '0';  SM <= '0';
                if  DIN= '1'  then
                      etat_futur  <=  V_plus ;
else etat_futur  <=  Zero1 ;
                end if;
        When V_plus  => Sp <= '1';  SM <= '0';
                if DIN = '1'  then
      etat_futur <= V_moin ;
                else etat_futur <=  Zero2 ;
                end if ;
        When  Zero2 =>  Sp <= '0'; SM <= '0';
                if DIN = '1' then
                      etat_futur  <=  V_moin ;
                else etat_futur <=  Zero2 ;
                end if ;
        When V_moin =>  Sp <= '0'; SM <= '1';
                if DIN = '0' then
                      etat_futur <=  Zero1;
                else etat_futur  <=  V_plus ;
                end if ;
 when others => etat_futur<=Zero1;
end case;
end process sortie;
end diagramme;
```

énumérés Zero1, V_plus, V_moins, Zero2 qui sont adaptés pour les machines d’états qu’on a identifiés par « etat ». Aussi on a déclaré les signaux internes type « etat »qui est « etat_futur » et « etat_present ».

Pour la description du fonctionnement du codeur AMI, il possible d’utiliser deux process pour la partie séquentiel et la partie combinatoire. Dans chaque process on utilise les structures suivantes :

- ***Pocess1***=> if…elsif…endif, permet d’initialiser la machine par le signal RAZ et d’assurer les transitions à chaque front montant de l’impulsion de l’horloge.
- ***Process2***=> Case…if…when…then…else…elsif…endif, permet d’exposer tous les cas possibles pour passer d’un état à un autre. Pour rester dans la description d’une machine de Moore, les sorties sont associées aux différentes états, indépendamment de l’entrée DIN.

Par exemple, on commence par l'état initial « Zero1 » qui est considéré comme un « état présent ». Les sorties « SP » et « SM » sont à 0. Dans le cas où l’entrée « DIN »

prend la valeur '1', au front montant de l'horloge on passe à l'état futur qui est « V_plus » et les sorties seront alors : «SP=1» et «SM=0». Mais Si «DIN=0», au front montant de l'horloge, on ne peut que rester dans le même état « Zéro1 » comme état futur.

Après avoir pris en compte les 4 états possibles, on termine par « when others» puis par « end Case ». Il ne faut pas oublier le « end process» et le « end diagramme».

La description VHDL du circuit du codeur AMI fera l'objet de la création d'un nouveau projet en utilisant le logiciel Quartus II avec une saisie textuelle pour ensuite être compilé et permettre la simulation.
Les étapes de cette opération sont développées dans l'annexe C pour plus de clarté.

IV.2.2.1 La compilation :

Après la création du projet avec l'entité « amicod » et la création du fichier « amicod. Vhdl » on procède à la saisie de la description vhdl du circuit. Après cela, on procède à la compilation qui réalise 4 étapes suivant ;

- ✓ L'analyse et la synthèse logique (Analysis and synthesis),
- ✓ la synthèse physique (Fitter),
- ✓ L'assemblage (Assemble),
- ✓ L'analyse temporelle (Classic Timing Anlyzer).

En fin de compilation on a reçu un message qui confirme le succès de l'opération à 100% comme le montre la fenêtre de la figure IV.5.

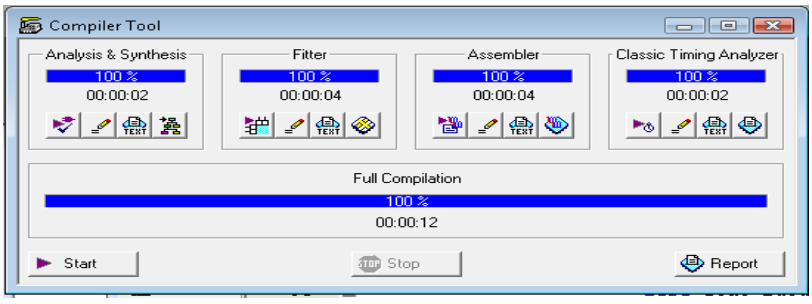


Fig. IV.5. *Fenêtre processus de compilation code AMI. (Textuelle)*

- ➢ **Résultat de compilation :**

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

confirmer le fonctionnement exact de ce circuit.

✓ La synthèse logique est la transformation des descriptions textuelles en une structure électronique à base de portes et de registres.

✓ Ces derniers peuvent être placés en fonction des ressources matérielles du circuit cible : C'est la synthèse physique.

✓ L'étape de l'assemblage consiste à produire les fichiers qui permettent la programmation du circuit.

✓ Enfin, les temps de propagation entre portes et le long des chemins choisis sont étudiés : C'est l'analyse temporelle.

IV.2.2.2 La vue RTL (*Register Transfer Level*) donnée:

L'utilisation de cette commande permet de voir la synthèse initiale des résultats pour déterminer le circuit de codeur AMI comme le montre la figure IV.6 :

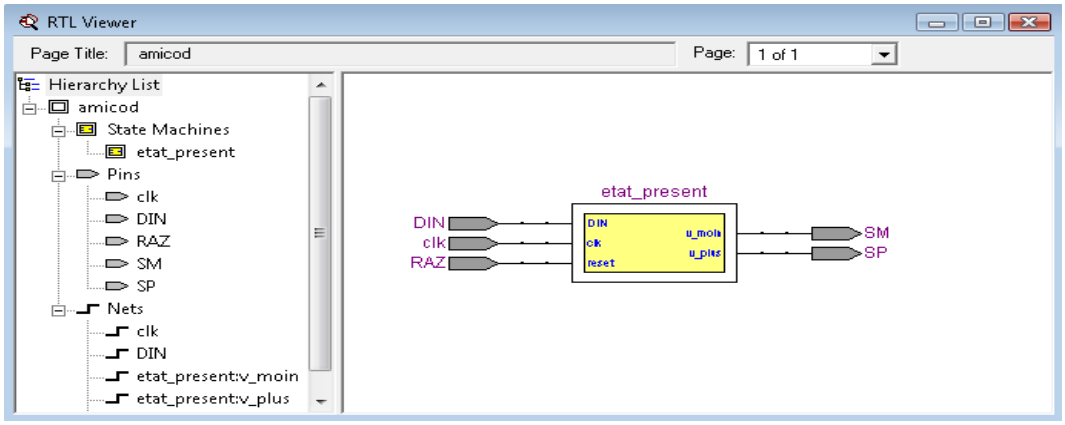


Fig. IV.6. *Fenêtre de présentation le codeur AMI. (Textuelle)*

➢ **Résultat de la vue RTL :**

D'après la fenêtre de la figure .IV.6 qui présente le circuit du codeur AMI avec la liste hiérarchique « Hierarchy List » qui définit les pins des entrées et des sorties et les signaux.

De même, nous pouvons visualiser la synthèse physique en utilisant la commande ***Tools —>Technology Map Viewer.*** On donner la produit entre les 4 bascules D et 4 sélecteurs placées dans le circuit et repérées par leurs références comme suivant:

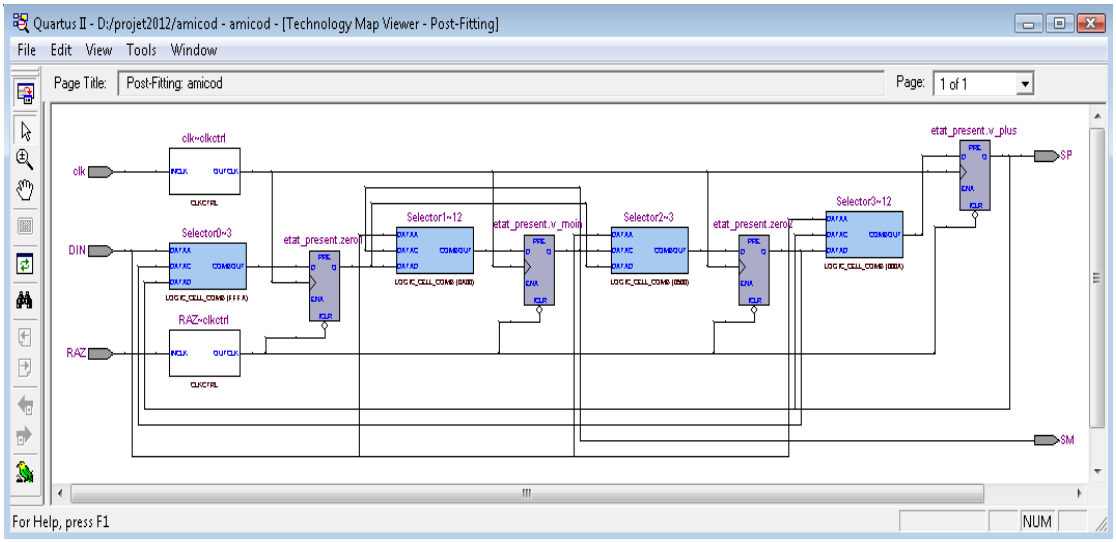


Fig. IV.7. *Fenêtre de visualiser la synthèse physique du codeur AMI. (Textuelle)*

IV.3.2.3 Le graphe d'état :(State Machine)

L'utilisation de cette commande permet d'avoir le diagramme d'état du codeur AMI où sont représentés les différents états avec leurs transitions. C'est ce que donne la figure IV.8.

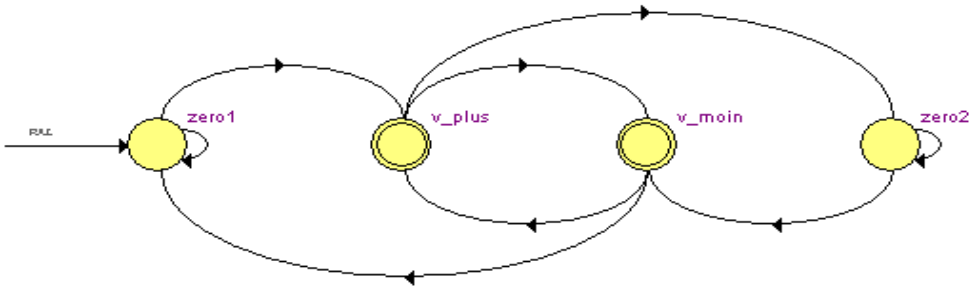


Fig. IV.8. *Fenêtre de présentation de graphe d'état pour le codeur AMI. (Textuelle)*

On remarque qu'on a les 4 états : Zero1, V_plus, V_moin, Zero2. Le changement d'états se fait en fonction de l'entrée DIN.

IV.2.2.4 La simulation :

Elle permet de voir le fonctionnement exact du circuit, en donnant les états des sorties SP et SM en fonction des différentes entrées clk, DIN, RAZ.

Le stimulus du « vector Waveform File » est donné par la figure IV.9 avant la simulation où les états de l'horloge, du RAZ et de DIN sont précisés.

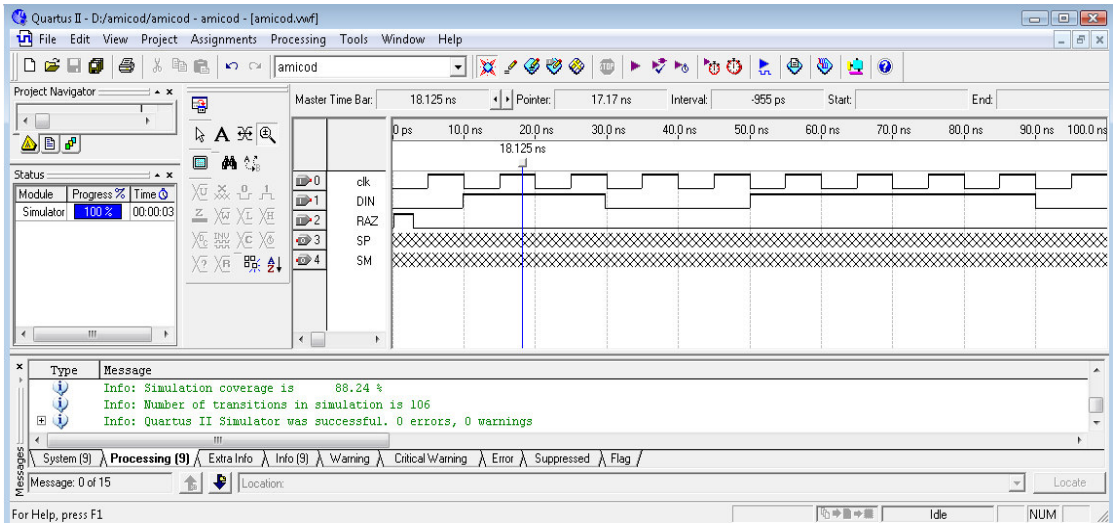


Fig. IV.9. *Les états des entrées de codeur AMI pour la simulation. (Textuelle)*

En choisissant la simulation fonctionnelle, on lance la simulation, pour obtenir le résultat de cette dernière qui est donné par la figure IV.10 où on voit en particulier les sorties SP et SM du codeur AMI.

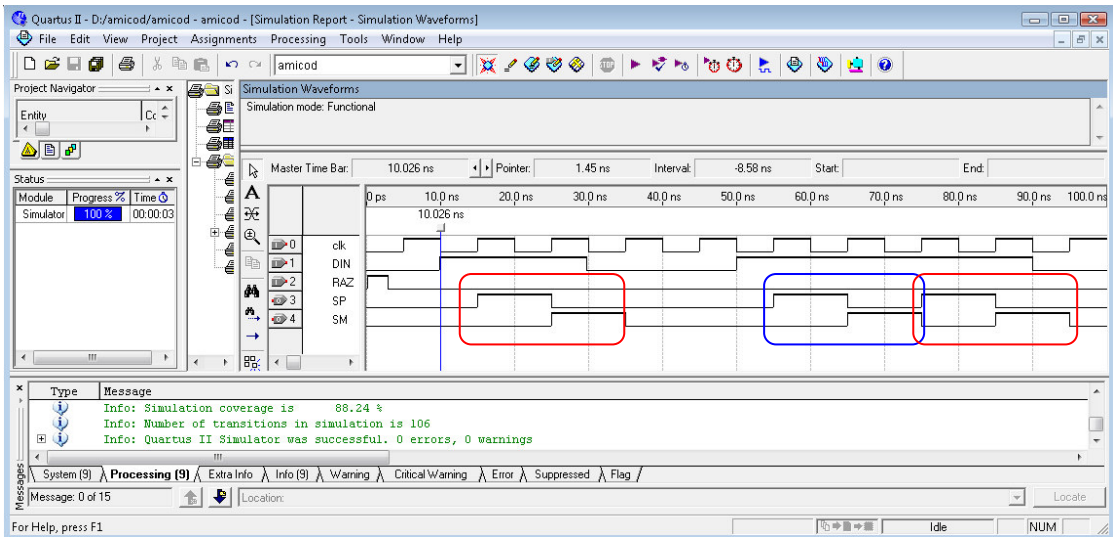


Fig. IV.10. *Le résultat de simulation défini les sorties pour le codeur AMI. (Textuelle)*

A partir de la fenêtre du résultat de la simulation on peut voir les valeurs des deux sorties SP et SM qui sont conformes au fonctionnement désiré.

A chaque front montant de l'horloge « clk » il y a lecture de la valeur de l'entrée DIN pour donner la valeur correspondante des deux sorties.

Par la suite la sortie du codeur AMI sera transformé en un code à 3 niveaux de tension : +V, -V, 0. La figure IV.11 représente cette transformation.

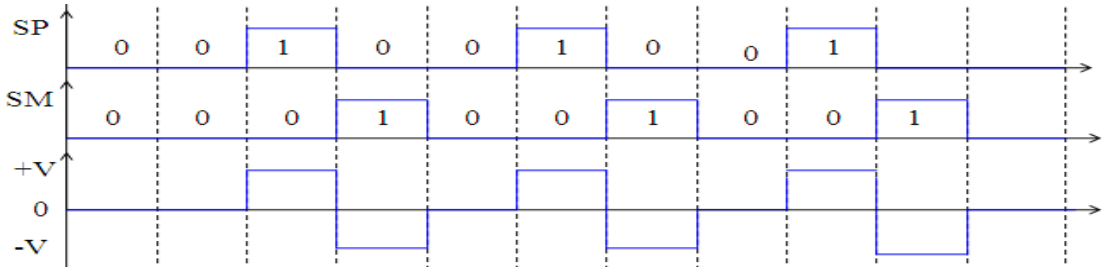


Fig. IV.11 *Transformation le codeur AMI à 3niveaux +V, 0,-V.*

IV.2.3 SYNTHESE MANUELLE ET SCHEMA OBTENU :

Maintenant on fait la synthèse du codeur AMI en utilisant la méthode d'Huffman, dans le cas d'une machine de Moore, qu'on qualifie de manuelle comparativement à la synthèse VHDL. Cette méthode est basée sur les étapes suivantes :

A. Table des états :

B. Le diagramme d'état du codeur AMI permet d'obtenir la table des états du tableau IV.1.

Etats présent	Etats futurs		Sortie	
	DIN=0	DIN=1	SP	SM
Zero_1	Zero_1	V_plus	0	0
V_plus	Zero_2	V_moin	1	0
Zero_2	Zero_2	V_moin	0	1
V_moin	Zero_1	V_plus	1	0

Tableau IV.1: *Etablissement de la matrice des phases primitive du codeur AMI*

C. Le codage des états :

On procède par la suite au codage des états pour obtenir le tableau IV.2. Après cela on voit qu'on a un tableau de Karnaugh avec 3 variables Q1, Q2 et DIN. Les 4 lignes ont données lieu à un codage sur deux bits Q1 et Q2.

DIN / Q1Q2	0	1	Sortie	
			SP	SM
00	00	01	0	0
01	11	10	1	0
11	11	10	0	0
10	00	01	0	1

Tableau. IV.2: *Etablissement de la matrice des excitations du codeur AMI.*

Le nombre 'n' de variables d'état est (ou variables secondaires): n=2. On appelle Q_1, Q_2 ces variables d'états qui sont les sorties de deux bascules. On aura la table d'état codée ou la matrice des excitations.

D. Définition des entrées des éléments mémoires:

On peut donner les deux tableaux de Karnaugh en fonction des transitions aux excitations.

Notation :

D	Q	Q^+	Les transitions
0	0	0	S_0
0	1	0	T_1
1	0	1	T_0
1	1	1	S_1

Q / DIN Q1Q2	0	1
00	S_0	S_0
01	T_1	T_1
11	S_1	S_1
10	S_0	T_0

Q / DIN Q1Q2	0	1
00	S_0	T_1
01	S_1	T_0
11	S_1	S_0
10	S_0	T_1

***Tableau. IV.3**: tableaux de Karnaugh en fonction des transitions aux excitations de code AMI.*

E. Les équations des états présents :

On peut donner les deux tableaux de Karnaugh en fonction des transitions aux excitations. Du fait que la réalisation se sera avec des bascules D, on peut donner les tableaux de Karnaugh des entrées D_1, D_2 de ces deux bascules. Ces deux dernières ont les expressions des tableaux IV.4 et IV.5 ci-dessous.

D1

DIN Q1Q2	0	1
00	0	0
01	1	1
11	1	1
10	0	0

$$D_1 = Q_2$$

$$D_2 = Q_2.\overline{DIN} + \overline{Q_2}.DIN$$

$$D_2 = Q_2 \oplus \text{DIN}$$

D2

DIN Q1Q2	0	1
00	0	1
01	1	0
11	1	0
10	0	1

***Tableau. IV.4:** Définition des entrées de la bascule D1.*

***Tableau. IV.5** Définition des entrées de la bascule D2.*

F. Les équations de sortie pour SP et SM :

SP, SM DIN Q1Q2	SP	SM
00	0	0
01	1	0
11	0	0
10	0	1

$SP = \overline{Q_1}.Q_2$

$SM = Q_1.\overline{Q_2}$

Tableau. IV.6: *Définition des variables de sortie, ou des fonctions de sortie du codeur AMI.*

G. Le schéma :

Ayant obtenu les équations des deux sorties SP et SM, on peut donner le schéma de la figure IV.12 en tenant compte des équations de D1 et de D2 entrées des bascules.

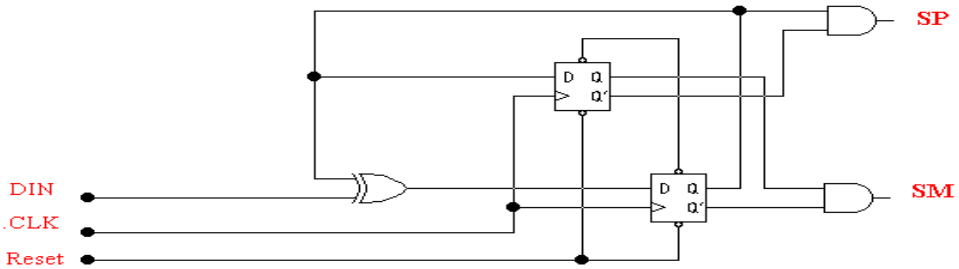


Fig. IV.12:*Logigramme du codeur AMI d'après la synthèse manuelle.*

Partant du schéma de la synthèse manuelle, on va faire sa saisie graphique donnée par la figure IV.13 dans un nouveau projet pour faire sa simulation.(voire l'annexe B)

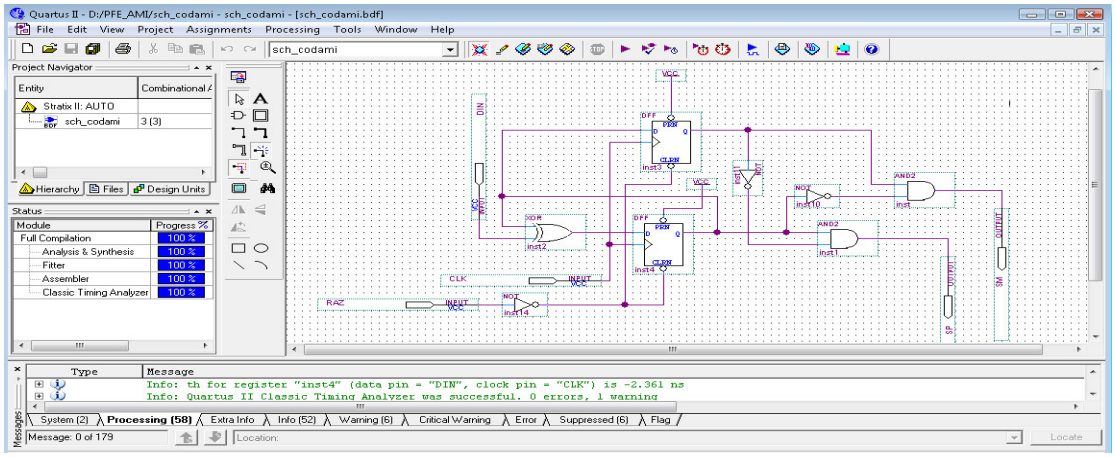


Fig. IV.13. *Saisie graphique du circuit du codeur AMI.*

IV.2.3.1 La compilation et la vue RTL pour la saisie graphique :

La compilation a été faite aussi avec succès dans ses 4 étapes. Puis utilisée la commande « **vue RTL** » qui permettre de voir la synthèse initiale des résultats pour déterminer le circuit de codeur AMI comme voir dans la fenêtre .IV.14.

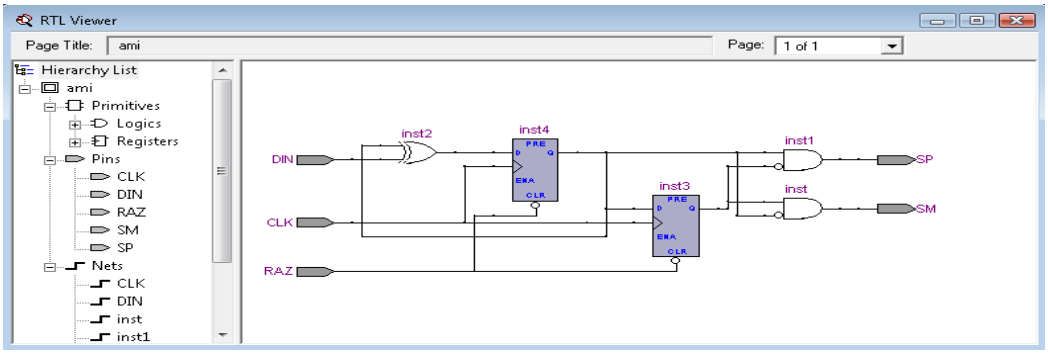


Fig. IV.14. *Fenêtre de la vue RTL de la saisie graphique du codeur AMI.*

De même, nous pouvons visualiser la synthèse physique en utilisant la commande ***Tools —>Technology Map Viewer.***

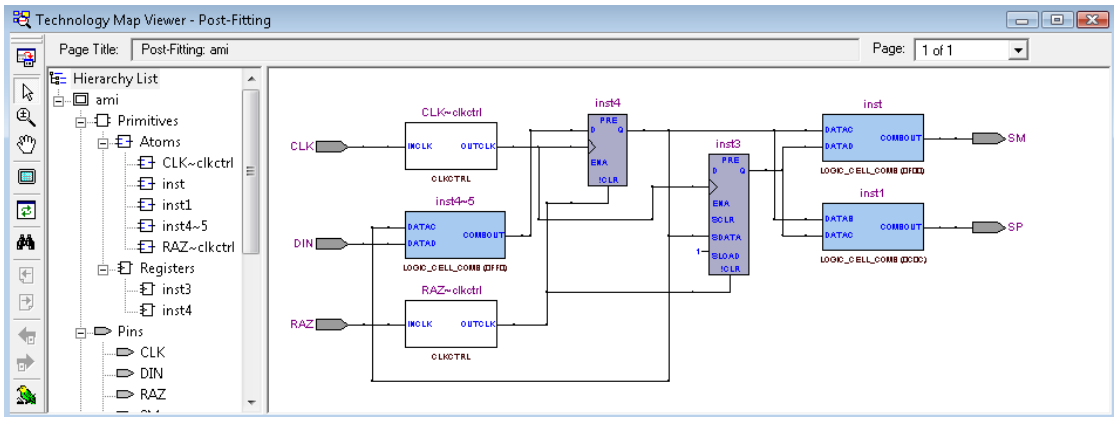


Fig. IV.15. *Fenêtre de visualiser la synthèse physique du codeur AMI. (Graphique)*

IV.2.3.2 La simulation :

Elle permet de voir le fonctionnement exact du circuit, en donnant les états des sorties SP et SM en fonction des différentes entrées clk, DIN, RAZ.

En choisissant la simulation fonctionnelle, on lance la simulation, pour obtenir le résultat de cette dernière qui est donné par la figure IV.16 où on voit en particulier les sorties SP et SM du codeur AMI.

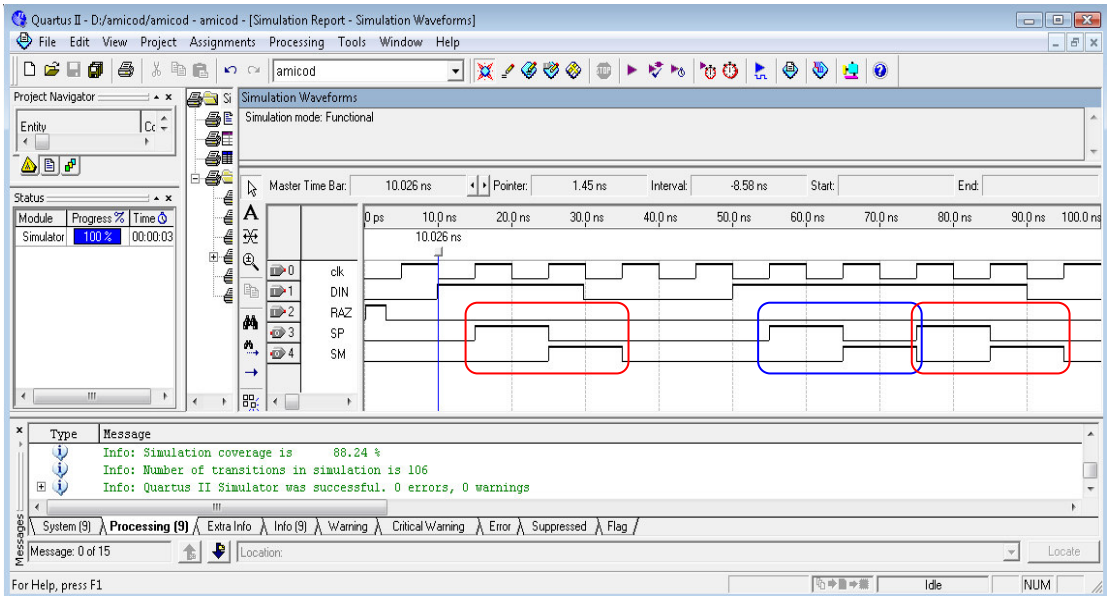


Fig. IV.16. *Le résultat de simulation défini les sorties pour le codeur AMI.*

A partir de la fenêtre du résultat de la simulation on peut voir les valeurs des deux sorties SP et SM qui sont conformes au fonctionnement désiré.

IV.2.4 SYNTHESE AVEC SAISIE DU DIAGRAMME D'ETAT :

Une autre méthode pour la synthèse des machines d'états consiste à créer un fichier « State Machine » par le logiciel Quartus. Une fois achevé, ce diagramme peut donner naissance à une description VHDL en utilisant la commande adéquate. Ensuite cette description est compilée et bien sûr simulée.

Ainsi donc le diagramme d'états de la figure IV.4 est saisi avec le logiciel Quartus II pour avoir un fichier avec l'extension « .smf » comme le montre la figure IV.17.

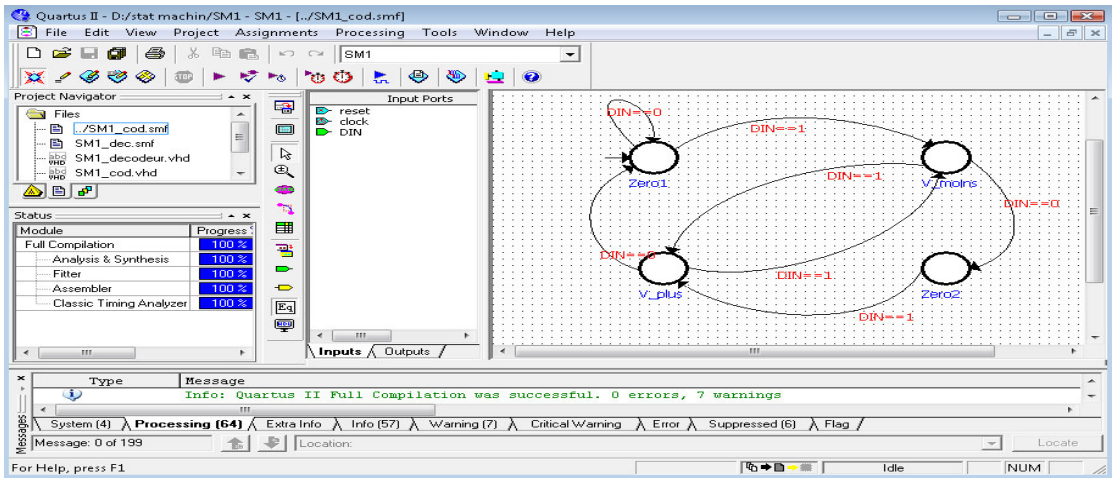


Fig. IV.17. *Diagramme d'états du fichier smf AMI.*

Pour plus de détail, on peut revenir à l'annexe D.

IV.2.4.1 Description VHDL du codeur AMI à partir du diagramme d'état :

C'est en utilisant la commande « ***Generate HDL File*** » qu'on obtient le fichier de la description VHDL à partir de la saisie du diagramme d'état. Le fichier obtenu est donné ci-dessous. Le fonctionnement correct sera confirmé par la simulation.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY SMCODAMI IS
   PORT (       reset  : IN STD_LOGIC := '0';
                clock : IN STD_LOGIC;
                 DIN   : IN STD_LOGIC := '0';
                 SP    : OUT STD_LOGIC;
                 SM    : OUT STD_LOGIC );
END SMCODAMI;
ARCHITECTURE BEHAVIOR OF SMCODAMI IS
   TYPE type_fstate IS(V_plus, Zero1,V_moins, Zero2);
   SIGNAL fstate : type_fstate;
   SIGNAL reg_fstate : type_fstate;
BEGIN
   PROCESS (clock,reg_fstate)
   BEGIN
      IF (clock='1' AND clock'event) THEN
         fstate <= reg_fstate;
      END IF;
   END PROCESS;
   PROCESS (fstate,reset,DIN)
   BEGIN
      IF (reset='1') THEN
         reg_fstate <= Zero1;
         SP <= '0';  SM <= '0';
      ELSE
         CASE fstate IS
            WHEN V_plus =>
               IF ((DIN = '0')) THEN
                  reg_fstate <= Zero1;
               ELSIF ((DIN = '1')) THEN
                  reg_fstate <= V_moins;
 -- Having else block to avoid latch inference
               ELSE
                  reg_fstate <= V_plus;
               END IF;
               SP <= '0';  SM <= '1';
            WHEN Zero1 =>
               IF ((DIN = '0')) THEN
                  reg_fstate <= Zero1;
               ELSIF ((DIN = '1')) THEN
                  reg_fstate <= V_moins;
 -- Having else block to avoid latch inference
               ELSE
                  reg_fstate <= Zero1;
               END IF;
               SP <= '0'; SM <= '0';
            WHEN V_moins =>
               IF ((DIN = '1')) THEN
                  reg_fstate <= V_plus;
               ELSIF ((DIN = '0')) THEN
                  reg_fstate <= Zero2;
 -- Having else block to avoid latch inference
               ELSE
                  reg_fstate <= V_moins;
               END IF;
               SP <= '1'; SM <= '0';
            WHEN Zero2 =>
               IF ((DIN = '1')) THEN
                  reg_fstate <= V_plus;
 -- Having else block to avoid latch inference
               ELSE
                  reg_fstate <= Zero2;
               END IF;
               SP <= '0'; SM <= '0';
            WHEN OTHERS =>
               SP <= 'X';                SM <= 'X';
              report "Reach undefined state";
         END CASE;
      END IF;
   END PROCESS;
END BEHAVIOR;
```

IV.2.4.2 La compilation et la vue RTL correspondante:

De la même manière, on crée un nouveau projet avec la description VHDL obtenue. La compilation est faite avec succès. L'utilisation de commande « vue RTL » permet d'obtenir la vue RTL de la figure IV.18. Correspondant à une saisie par le diagramme d'état du codeur AMI.

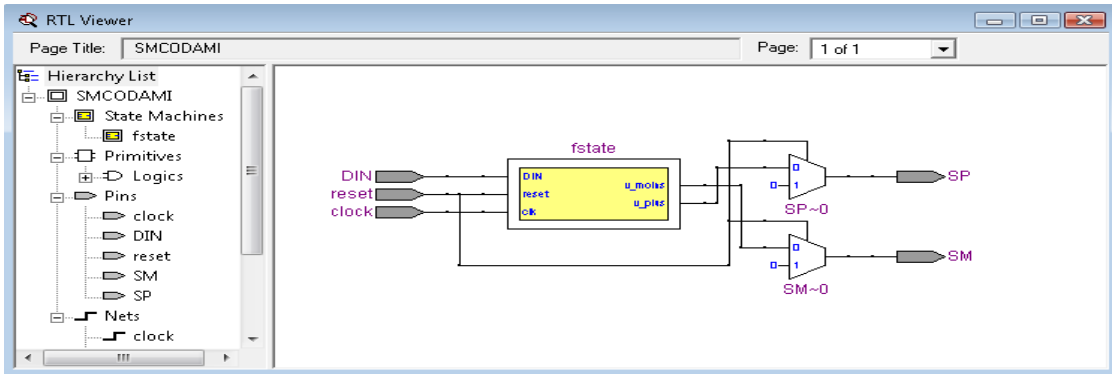


Fig. IV.18. *Fenêtre de présentation le codeur AMI. (Diagramme d'état)*

De même, nous pouvons visualiser la synthèse physique, figure IV.19, en utilisant la commande ***Tools —>Technology Map Viewer.***

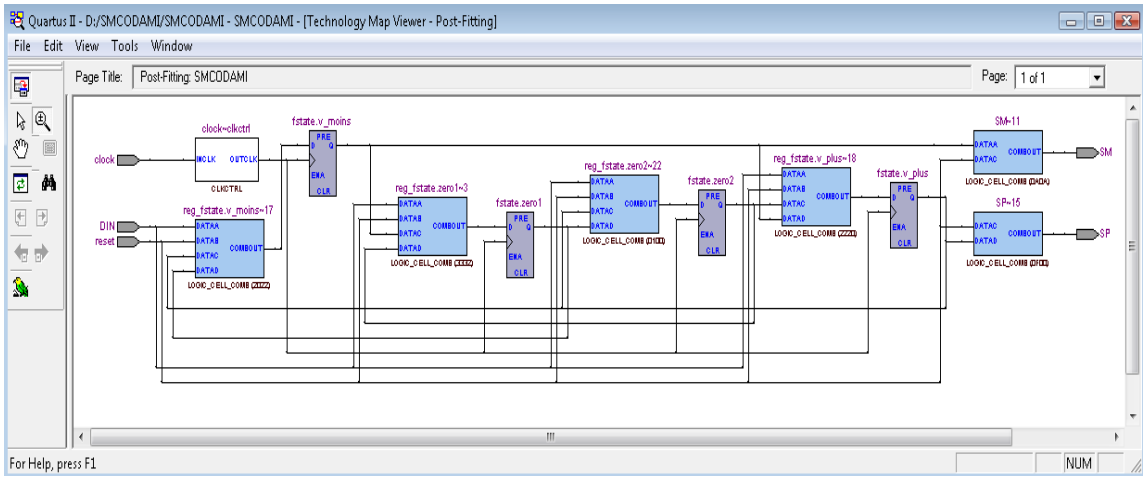


Fig. IV.19. *La synthèse physique du codeur AMI par diagramme d'état.*

IV.2.4.3 La simulation :

La simulation donne un fonctionnement conforme au cahier de charge dans la figure IV.20.

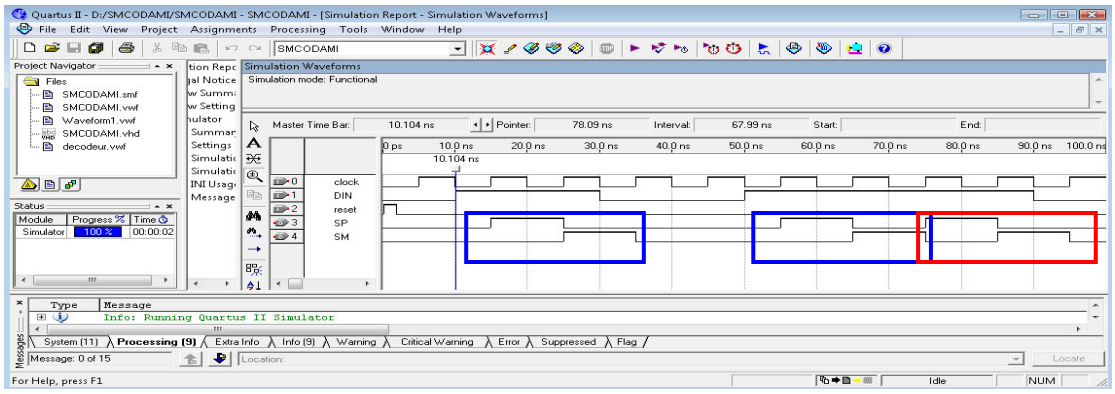


Fig. IV.20. *Le résultat de simulation défini les sorties du codeur AMI.*

IV.2.5 CIRCUIT DU CODEUR AMI PAR " Electronic WORKBENCH 5.1":

Le logiciel "WORKBENCH 5.1" nous a permis de réaliser le circuit pour tester son fonctionnement. Dans la figure IV.21 on donne le circuit complet avec les différentes parties qui lui sont associées :

1. On a besoin d'un circuit qui génère le signal d'entrée DIN qui est une suite de signaux logiques 0 et1. Cela peut se faire de deux manières : soit manuellement avec l'interrupteur D ou bien avec le générateur constitué par l'ensemble des trois bascules D constituant l'entrée de la porte OU-EX. Les trois bascules D (A, B, C) constituent un registre à décalage à droite avec une rétroaction formée par un OU-EX dont les entrées sont Q_C et $\overline{Q_B}$. La sortie A⊕B du deuxième OU-EX dont les entrées sont Q_A et Q_B, donne le signal DIN.

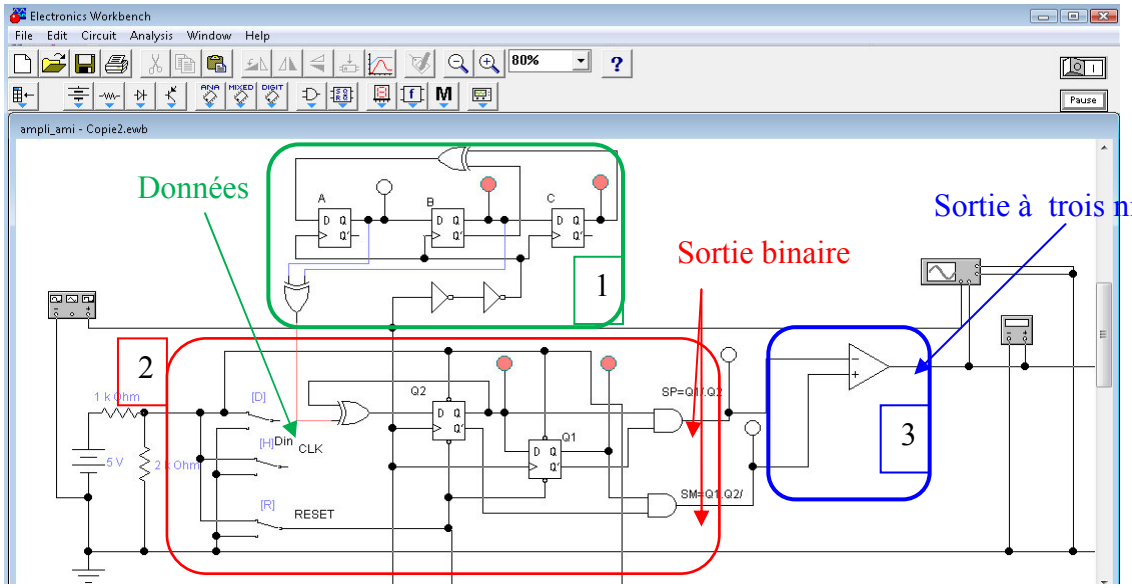


Fig. IV.21:*Réalisation du circuit codeur AMI par "WORKBENCH 5.1"*

On trouve la fonction d'entrée pour la bascule A $D_A = B \oplus \bar{C}$. Le tableau IV.7 résume le fonctionnement du générateur de fonction avec sa sortie A⊕B.

A	B	C	$D_A = B \oplus \bar{C}$	DIN=A⊕B
0	0	0	1	0
1	0	0	1	1
1	1	0	0	0
0	1	1	1	1
1	0	1	0	1
0	1	0	0	1
0	0	1	0	0
0	0	0	1	0

Tableau IV.7 : *Table du générateur de fonction.*

La séquence de la sortie est donc DIN=01011100

2. La deuxième partie représente le codeur avec ses deux sorties SP et SM vu précédemment.

3. C'est la conversion du signal logique (SP, SM) en un signal analogique à trois niveaux +V, -V, 0, dans la partie 3.

Le signal à la sortie de l'amplificateur opérationnel peut être observé à l'oscilloscope comme le montre la figure IV.22.

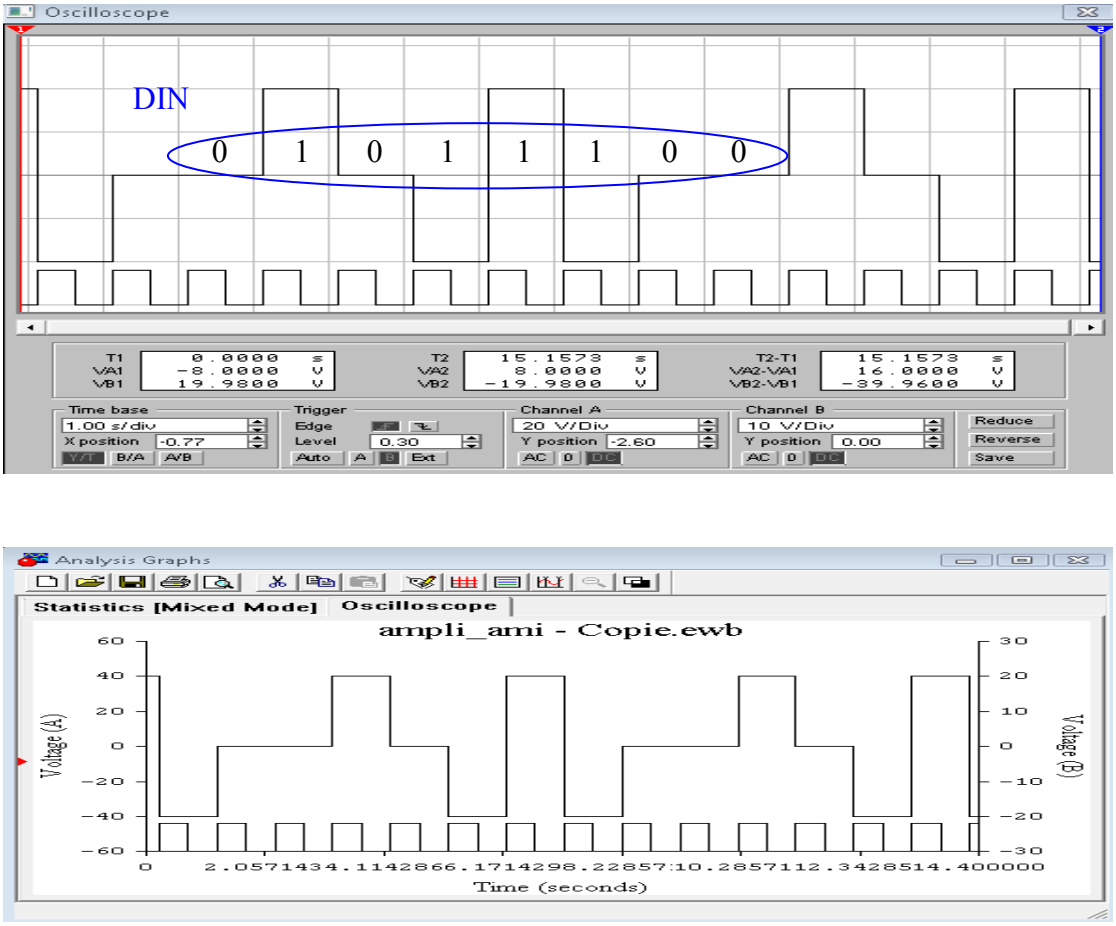


Fig. IV.22: *Le signal de sortie à trois états du circuit codeur AMI.*

IV.3 DEFINITION DU DECODEUR AMI ET SIMULATION :

On peut définir la fonction du décodeur comme étant un circuit qui, à partir des signaux SP et SM du codeur AMI, va nous restituer la séquence d'entrée DIN initiale. C'est donc l'opération inverse pour le codeur AMI.

Pour le moment on essaie d'ignorer La conversion du signal « SP, SM » en un signal à trois niveaux à l'émission et sa reconversion en un signal « SP, SM » à la réception.

On est amené à faire la conception d'un circuit dont les entrées sont SP et SM et la sortie est « Dout » et dont le fonctionnement est comme sui :

- Si l'entrée « SP = '0' », « SM = '0' », la sortie « Dout = '0' ».
- Si l'entrée « SP = '0' », « SM = '1' », la sortie « Dout = '1' ».
- Si l'entrée « SP = '1' », « SM = '0' », la sortie « Dout = '1' ».
- Si l'entrée « SP = '1' », « SM = '1' », la sortie « Dout = '0' ».

Ce dernier état ne peut pas se présenter normalement et doit normalement indiquer qu'il y a une erreur dans la transmission.

IV.3.1 DIAGRAMME D'ETAT DU DECODEUR AMI :

D'après le fonctionnement du décodeur AMI, on peut proposer le digramme d'état d la figure IV.23.

Fig. IV.23. *Diagramme d'état du décodeur AMI.*

C'est un diagramme avec trois états : Zero, UN, DEUX. La sortie Dout est obtenue après chaque impulsion d'horloge, comme dans la machine de Moore, avec une sortie synchrone. Pour la synthèse de ce circuit on utilisera les deux méthodes à savoir : - La saisie textuelle par langage VHDL,

- La saisie par diagramme d'état.

IV.3.2 DESCRIPTION DECODEUR AMI AVEC SYNTHESE VHDL :

Le programme suivant donne la description VHDL du décodeur AMI pris comme une machine d'état.

Dans le diagramme d'états du décodeur AMI, on utilise aussi deux process dans l'architecture l'un pour la partie séquentiel et l'autre pour la partie combinatoire.

Un nouveau projet a été créé pour la saisie textuelle de ce décodeur avec le logiciel Quartus II pour ensuite compiler et la simuler.

```
library ieee;
use ieee.std_logic_1164.all;
--décodeur AMI avec trois etat
Entity DECAMI2 is
port ( RAZ, SP,SM,CLK: in std_logic;
       DOUT: out std_logic );
end DECAMI2;
-- architecture
architecture DECAMI of DECAMI2 is
type ETAT_DECO is (zero, un, deux);
signal
etat_present, etat_futur: ETAT_DECO   :=zero;
begin
     REGETAT : process (clk, RAZ, etat_futur )
      begin
          if RAZ='1'then
          etat_present <= zero;
         elsif Clk'event and clk='1' then
               etat_present<=etat_futur;
         end if;
         end process REGETAT;
```

```
COMBETATOUT: process (SP,SM, etat_present)
   begin
       case etat_present is
         when zero =>  DOUT <= '0' ;
            if (SP = '1' and SM = '0') then
               etat_futur <=  deux  ;
            elsif (SP = '0' and SM = '1') then
               etat_futur <= un ;
            else etat_futur <= zero ;
            end if ;
         when un => DOUT <= '1';
            if (SP = '1' and SM = '0') then
               etat_futur <=deux ;
              else etat_futur <= zero ;
            end if ;
         when deux => DOUT <= '1';
            if (SP= '0' and SM = '1')  then
                etat_futur <= un;
               else etat_futur <=zero ;
            end if ;
         when others => etat_futur <= zero;
         end case ;
    end process COMBETATOUT ;
end DECAMI;
```

IV.3.2.1 La compilation et la vue RTL:

La description VHDL a été compilée avec succès. On utilise la commande « vue RTL » qui permettre de voir la synthèse initiale des résultats pour déterminer le circuit de décodeur AMI et donnée par la figure IV.24.

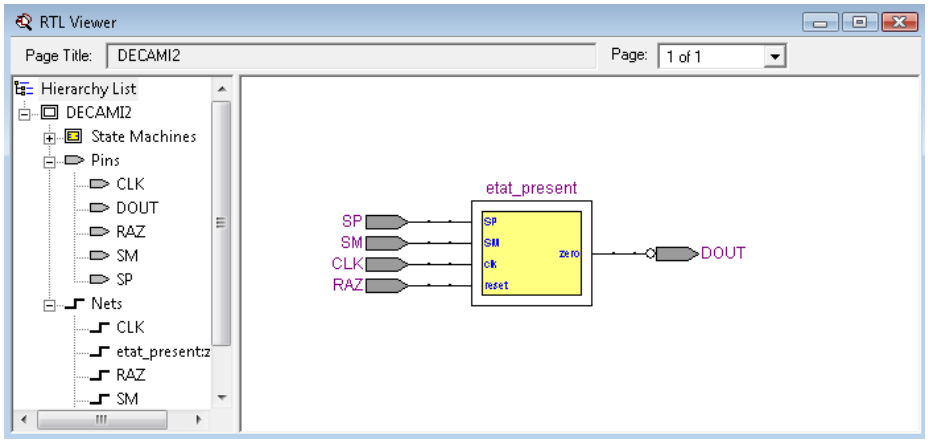


Fig. IV.24. *Fenêtre de présentation le décodeur AMI. (Textuelle).*

De même, nous pouvons visualiser la synthèse physique en utilisant la commande ***Tools —>Technology Map Viewer.***

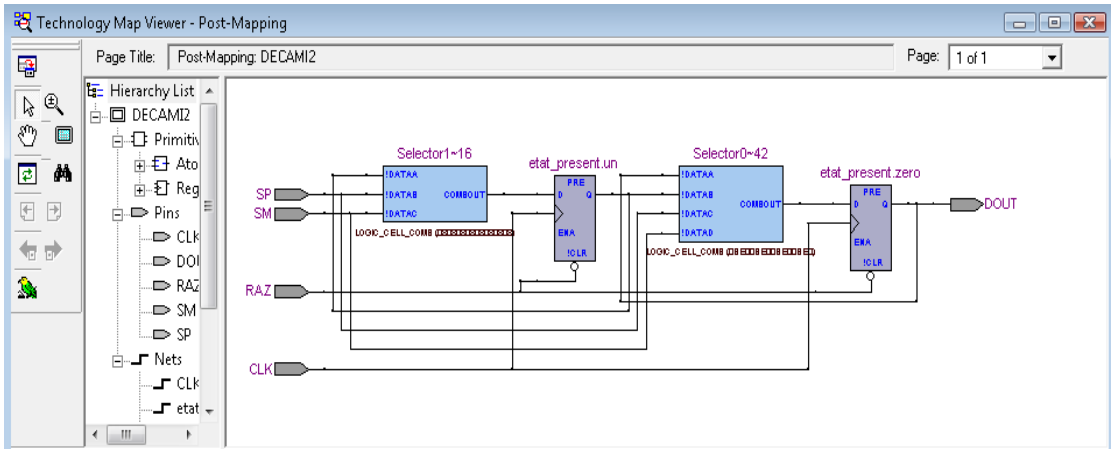


Fig. IV.25. *Fenêtre de visualiser la synthèse physique du décodeur AMI. (Textuelle).*

IV.3.2.2 Le graphe d'état :(State Machine)

L'utilisation de cette commande permet de voir la machine d'état du décodeur AMI où on retrouve les trois états avec les différentes transitions dans la figure. IV.26.

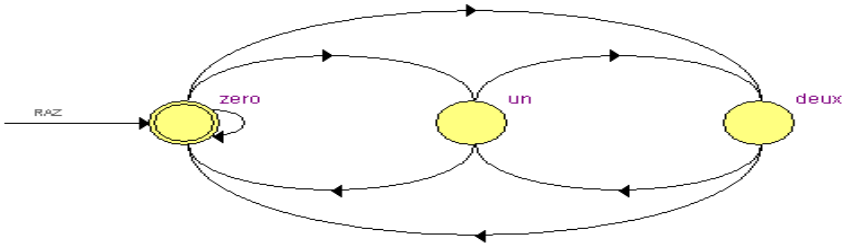


Fig. IV.26. *Graphe d'état pour le décodeur AMI.*

IV.3.2.3 La simulation :

On a créé un fichier « DECAMI.vwf» pour la simulation avec les entrées RST, clk, SP, SM et la sortie DOUT comme le montre la figure. IV.27.

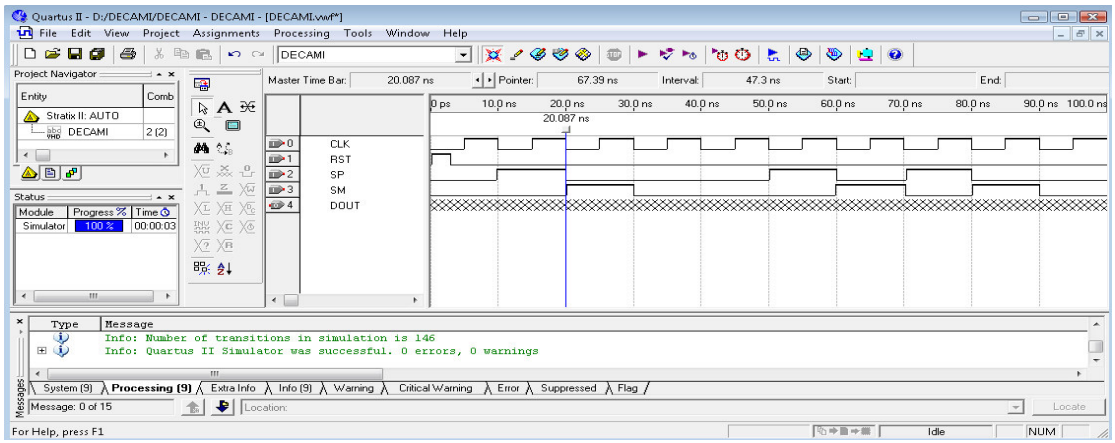


Fig. IV.27. *Fenêtre du fichier de la simulation du décodeur AMI.*

Après la simulation, on trouve dans la figure IV.28 les valeurs de la sortie DOUT qui sont conformes au fonctionnement du circuit du décodeur AMI.

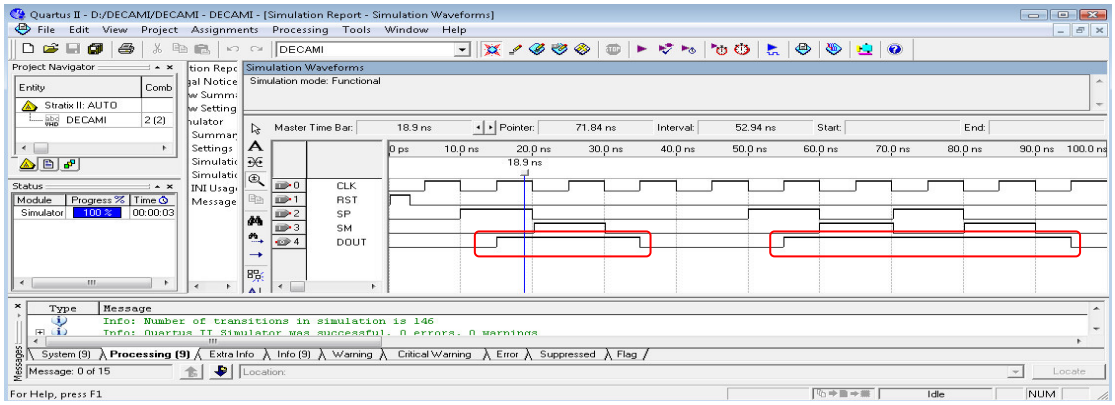


Fig. IV.28. *Valeurs de la sortie DOUT après la simulation du décodeur AMI*

IV.3.3 LA SYNTHESE AVEC SAISIE PAR DIAGRAMME D'ETAT:

Comme précédemment on va faire la synthèse du décodeur AMI en utilisant le diagramme d'état pour obtenir la description VHDL générée par Quartus.

Pour lancer cette opération on commence par la saisie du diagramme d'état du décodeur AMI comme il a été fait avec le décodeur. On a créé un fichier « SMDECAMI.smf » dont la fenêtre est donnée dans la figure IV.29.

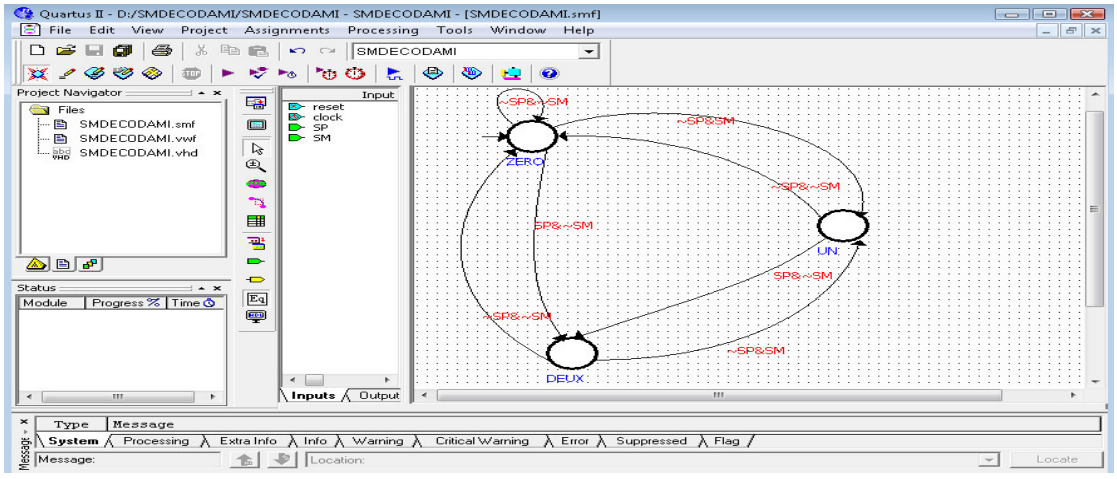


Fig. IV.29. *Fenêtre du fichier SMDECAMI.smf du décodeur AMI.*

Ayant terminé le diagramme, on le convertit en une description VHDL qui à son tour doit être compilée et simulée.

IV.3.3.1 Description VHDL du décodeur AMI générée à partir du diagramme d'état :

La description VHDL générée par le logiciel Quartus II en utilisant la commande ***Generate HDL File*** est donnée ci-dessous :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY  DECAMI_D  IS
   PORT (reset     : IN STD_LOGIC := '0';
         clock     : IN STD_LOGIC;
         SP        : IN STD_LOGIC := '0';
         SM        : IN STD_LOGIC := '0';
         DOUT   : OUT STD_LOGIC       );
END DECAMI_D;
ARCHITECTURE  BEHAVIOR OF DECAMI_D IS
   TYPE type_fstate IS (ZERO, DEUX, UN);
   SIGNAL fstate : type_fstate;
   SIGNAL reg_fstate : type_fstate;
BEGIN
   PROCESS (clock, reset, reg_fstate)
   BEGIN
      IF (reset='1') THEN
         fstate <= ZERO;
      ELSIF (clock='1' AND clock'event) THEN
         fstate <= reg_fstate;
      END IF;
   END PROCESS;
PROCESS (fstate, SP,SM)
BEGIN
  CASE fstate IS
    WHEN ZERO =>
      IF ((NOT((SP = '1')) AND NOT((SM = '1')))) THEN
            reg_fstate <= ZERO;
        ELSIF ((NOT((SP = '1')) AND (SM = '1'))) THEN
            reg_fstate <= UN;
        ELSIF (((SP = '1') AND NOT((SM = '1')))) THEN
            reg_fstate <= DEUX;
    -- Having else block to avoid latch inference
          ELSE
            reg_fstate <= ZERO;
         END IF;
         DOUT <= '0';
WHEN DEUX =>
  IF ((NOT((SP = '1')) AND NOT((SM = '1')))) THEN
            reg_fstate <= ZERO;
     ELSIF ((NOT((SP = '1')) AND (SM = '1'))) THEN
            reg_fstate <= UN;
 -- Having else block to avoid latch inference
          ELSE
            reg_fstate <= DEUX;
  END IF;
         DOUT <= '1';
WHEN UN =>
  IF ((NOT((SP = '1')) AND NOT((SM = '1')))) THEN
            reg_fstate <= ZERO;
     ELSIF (((SP = '1') AND NOT((SM = '1')))) THEN
            reg_fstate <= DEUX;
 -- Having else block to avoid latch inference
          ELSE
            reg_fstate <= UN;
     END IF;
         DOUT <= '1';
  WHEN OTHERS =>
         DOUT <= 'X';
         report "Reach undefined state";
      END CASE;
   END PROCESS;
END BEHAVIOR;
```

IV.3.3.2 La compilation et la vue RTL donnée :

De la même manière, La description VHDL a été compilée avec succès. On utilise la commande « vue RTL » qui permettre de voir la synthèse initiale des résultats pour déterminer le circuit de décodeur AMI et donnée par la figure IV.30.

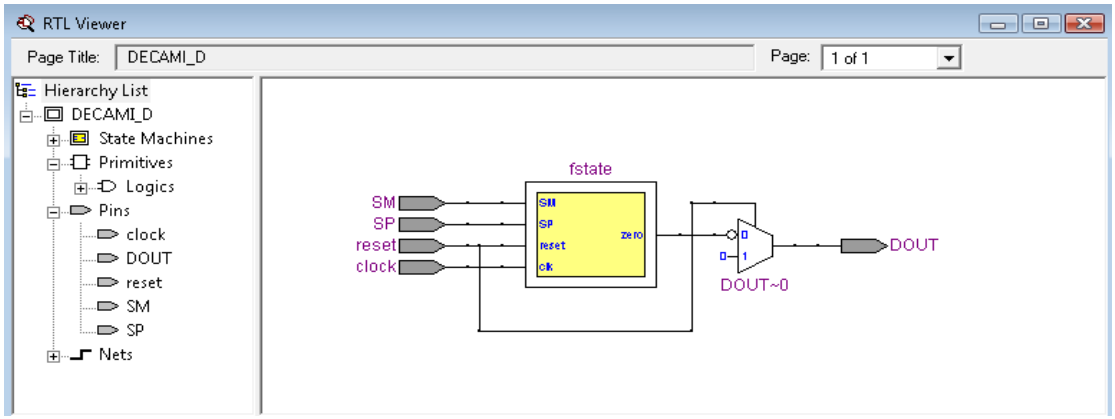


Fig. IV.30. *Fenêtre de la vue RTL du décodeur AMI par diagramme d'état.*

De même, nous pouvons visualiser la synthèse physique en utilisant la commande ***Tools —>Technology Map Viewer.*** On obtient le circuit du schéma IV.36.

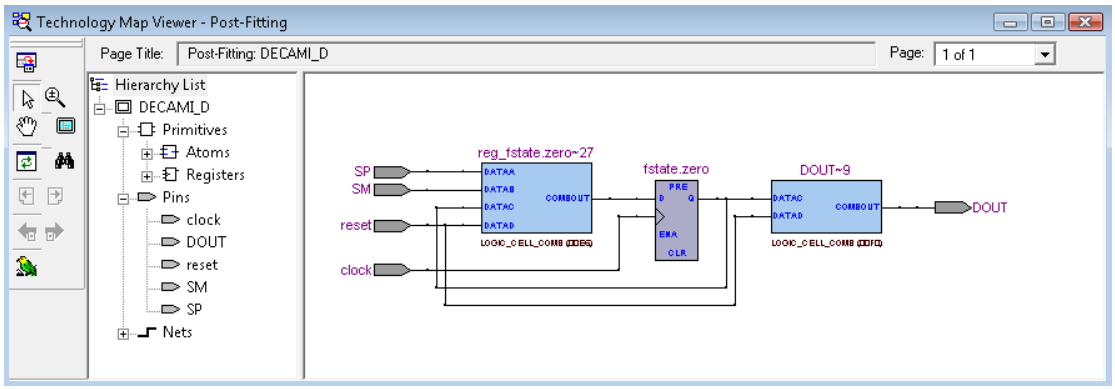


Fig. IV.31. *Fenêtre de la synthèse physique du décodeur AMI.*

IV.3.3.3 La simulation :

Lorsqu'on a simulé la description on a obtenu la sortie Dout similaire comme on peut le voir dans le schéma IV.32.

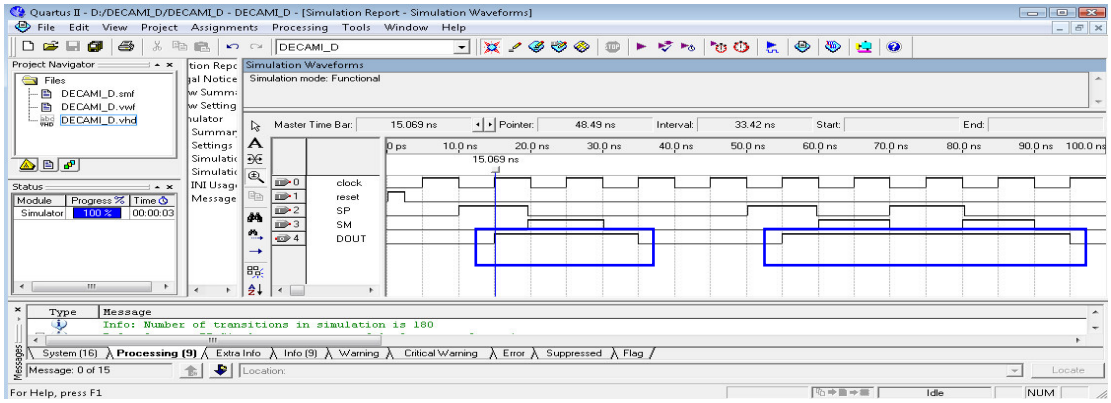


Fig. IV.32. *Valeurs de la sorties DOUT après la simulation du décodeur AMI par diagramme d'état*

IV.4 SYNTHESE DU CODEUR_DECODEUR AMI EN VHDL:

Maintenant on va synthétiser le codeur-décodeur AMI après avoir synthétisé séparément chacun d'eux. Pour cela on va utiliser la description structurelle où le « amicod » et le « DECAMI » seront des « components ».

IV.4.1 Description structurelle du codeur-décodeur AMI :

La description structurelle de cet ensemble est composée de l'entité « CODEC» et de l'architecture « description» qui est donne en annexe AB.

Cette description nécessite être compilée et simulée pour certifier son fonctionnement correct.

IV.4.1.1 La vue RTL

Cette commande nous permet de voir le schéma de la figure IV.33. Où on les deux composants « amicod » et « DECAMI ».

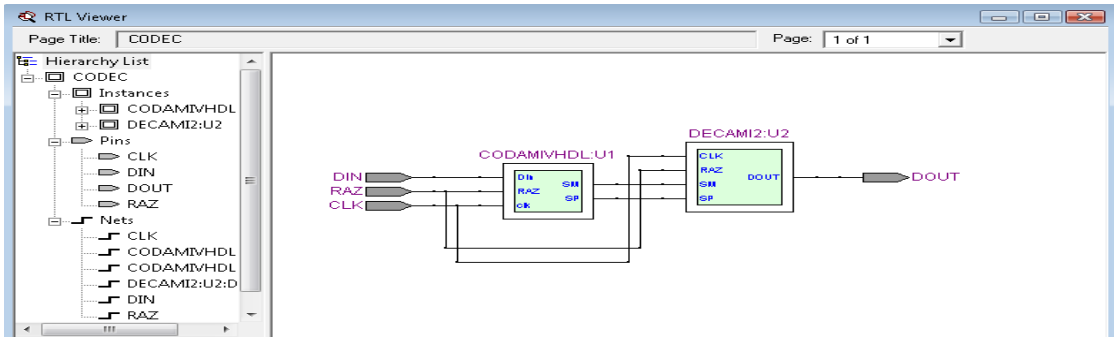


Fig. IV.33. *Fenêtre de la vue RTL du codec AMI.*

De même, nous pouvons visualiser la synthèse physique en utilisant la commande ***Tools —>Technology Map Viewer.*** Comme le montre la figure IV.34.

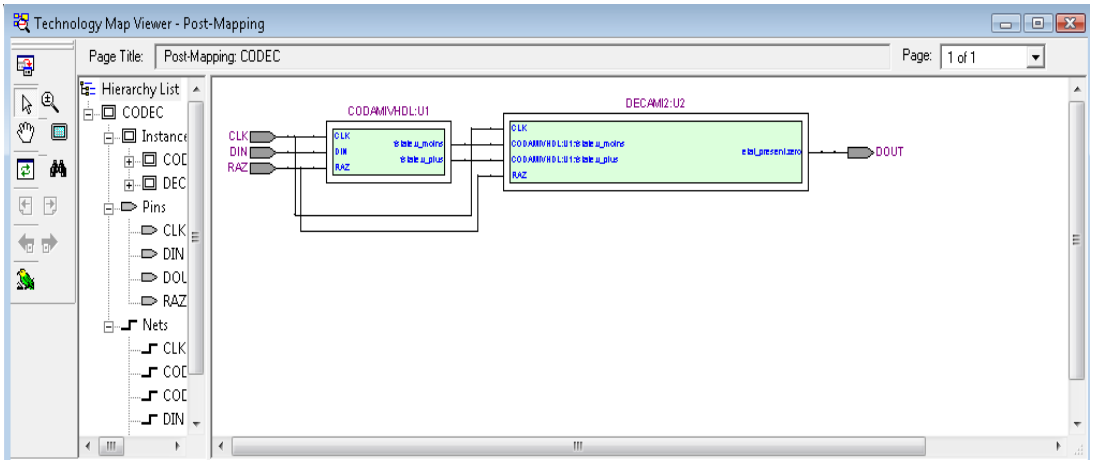


Fig. IV.34. *Fenêtre de la synthèse physique du codec AMI.*

IV.4.1.2 Le graphe d'état :(State Machine)

L'utilisation de cette commande permet de voir la machine d'état de codeur et du décodeur AMI sur deux fenêtres différentes données par la figure IV.35.

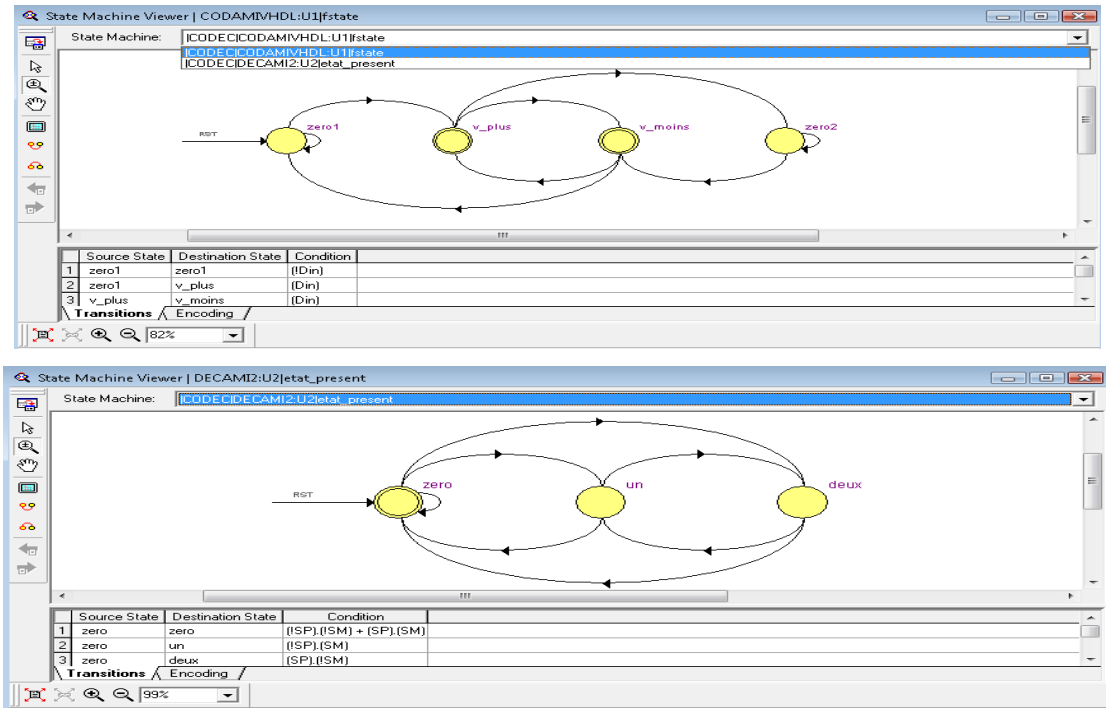


Fig. IV.35. *Fenêtre de d'état pour le codeur et décodeur AMI.*

IV.4.1.3 La simulation :

Dans la simulation du circuit du CODEC, on utilise les entrées clk, RAZ, DIN comme dans la figure IV.36 et on observera la variation des sorties correspondantes. Pour plus d'information, on a aussi intégré les sorties intermédiaires SP et SM.

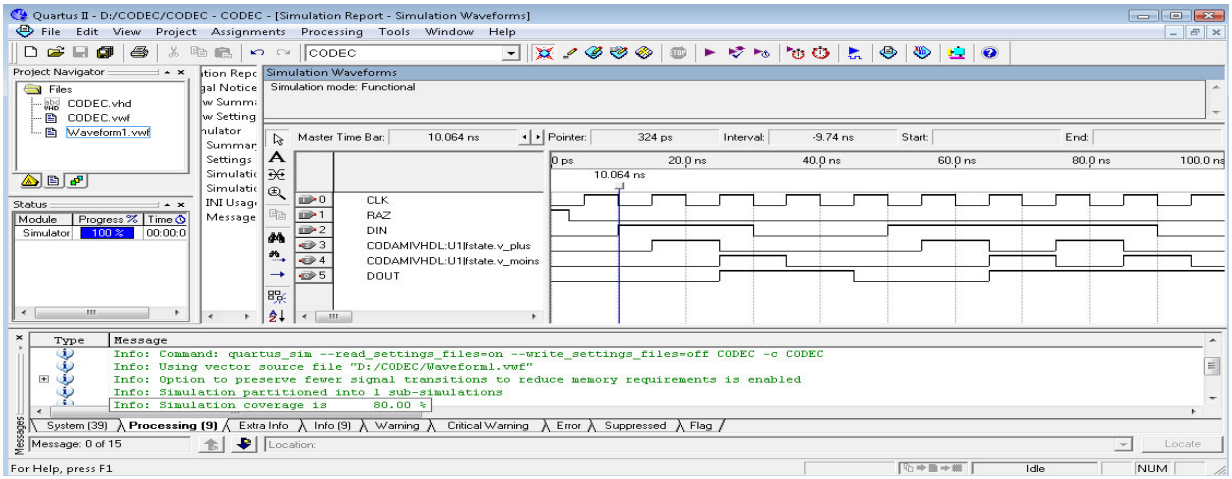


Fig. IV.36. *Le résultat de simulation défini les sorties pour le codec AMI.*

IV.4.2 REALISATION DU CIRCUIT CODEC AMI PAR " WORKBENCH 5.1":

Enfin on va présenter une structure plus complète du codeur-décodeur en introduisant la conversion du signal « SP, SM » en trois états +V, -V, 0 et l'opération inverse qui à partir des trois états redonner le signal « SP, SM ».

Cette partie est du domaine de l'électronique analogique et qu'on n'a pas synthétisé en VHDL.

La figure IV.37 représente le schéma complet du codeur-décodeur AMI où les signaux peuvent être observés à l'oscilloscope.

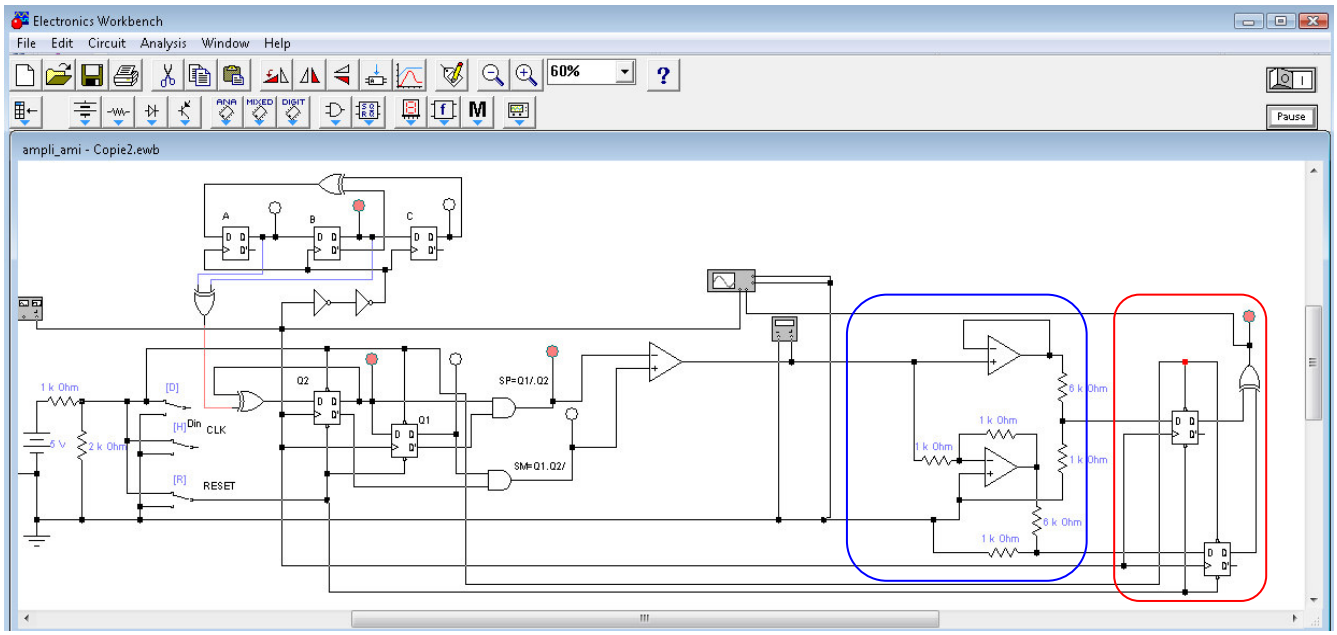


Fig. IV.37: *Le circuit codec AMI par "WORKBENCH 5.1".*

La sortie DOUT nous donne l'entrée Din après un décalage de deux impulsions d'horloge. Dans la zone en bleu on a la conversion trois niveau vers deux sorties binaires SP' et SM'. Ces deux signaux sont convertis en un signal de sortie binaire DOUT dans la zone rouge.

Dans notre exemple la séquence générée est : DIN= 01011100

En sortie on a la même séquence : Dout=01011100.

La figure IV.38 représente DOUT en fonction de l'entrée DIN.

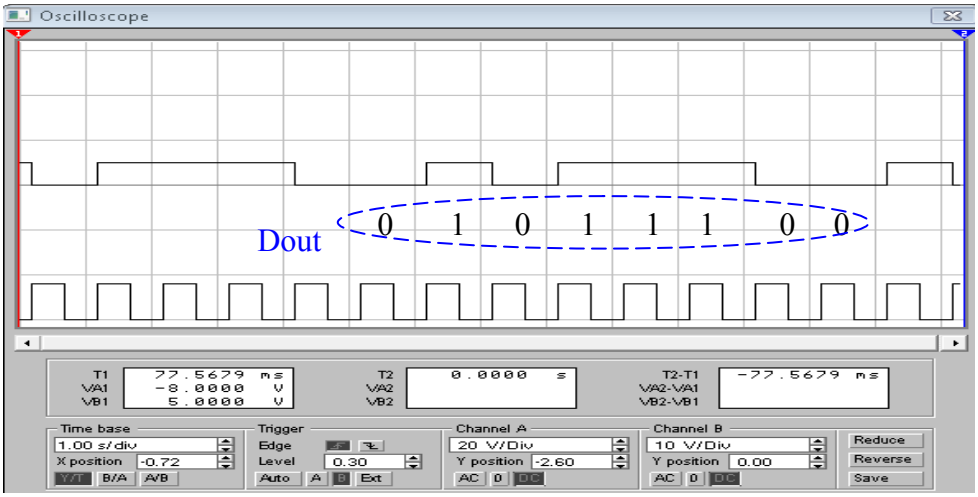


Fig. IV.38:*La sortie du circuit décodeur AMI par "WORKBENCH 5.1".*

IV.5 CONCLUSION :

La synthèse du codeur-décodeur a été faite par différentes méthodes :

- la saisie textuelle en langage VHDL
- la méthode classique et manuelle, suivie par une saisie graphique
- la saisie du graphe d'état

Aussi on a donné une description structurelle pour l'ensemble du circuit. La simulation a confirmé le résultat attendu de ce circuit.

Le décalage de la sortie résulte du fait que les deux machines de Moore conçues sont gérées par l'horloge.

CONCLUSION ET PERSPECTIVES

C'est autour de la synthèse des machines d'états avec le langage VHDL que ce travail a été effectué. Il a été nécessaire d'introduire les systèmes séquentiels asynchrones et synchrones pour faire une nette distinction entre les deux. Mais pour plus d'efficacité, on a jugé utile de développer uniquement les systèmes séquentiels synchrones. De ces derniers, on peut développer la synthèse selon les deux machines : Mealy et Moore. Mais puisque les circuits à synthétiser se font selon le modèle de Moore on s'est limité à cette option.

A travers le circuit du codeur et décodeur AMI, on a exposé une transmission en bande de base. Ces deux circuits ont été étudiés en tant que systèmes séquentiels synchrones ou machines d'états dont les sorties sont synchrones avec le signal d'horloge et ne pouvant pas être autrement. C'est pourquoi le modèle de Moore s'est imposé.

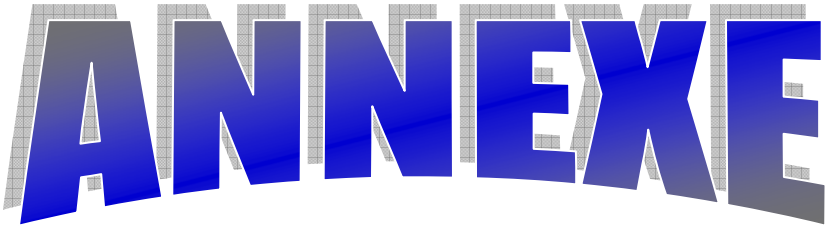
On a expliqué que la synthèse du circuit (codeur ou décodeur) peut être faite de trois manières :

- La méthode manuelle en utilisant la méthode Huffman-Mealy et faire une saisie graphique avec Quartus pour ensuite simuler.
- La saisie textuelle d'après la description VHDL des machines d'états, en utilisant l'instruction CASE.
- La saisie par diagramme d'état ou Finite State Machine pour obtenir la description VHDL.

L'ensemble codeur /décodeur a été synthétisé avec succès puis simulé pour avoir un fonctionnement correct. De même une partie analogique a été intégré pour avoir les trois états du codeurs +V, -V, 0 et leur reconversion en signaux logiques pour être l'entrée du décodeur. Aussi le codeur/décodeur et la partie analogique ont été simulés avec le logiciel Workbench pour vérifier le résultat.

Vu l'inconvénient du code AMI dans le cas d'une suite de zéro, il faut passer au codage HDBn (Haute Densité Binaire d'ordre n) qui est un code bipolaire dans lequel pour éviter de longue suite de niveaux bas, on remplace le bit de rang n+1 par un bit particulier caractérisé par le viol de parité pour être reconnu par le système. C'est-à-dire que sa polarité n'est pas inversée par rapport au bit codé précédemment.

On ajoute enfin que la génération de l'horloge au niveau du décodeur n'a pas été abordée dans notre réalisation. On s'est contenté d'utiliser l'horloge du codeur uniquement.



Notice de prise en main du logiciel Quartus II 7.2.

ANNEXE A : Création d'un projet dans le logiciel Quartus II

Introduction: [15]

Dans cette partie d'annexe, vous apprendrez comment programmer la carte DE2 pour y implémenter un circuit logique. Vous apprendrez la base de l'environnement graphique du programme « Quartus II », qui est le compilateur du FPGA soudé sur la carte. Ce programme vous permet de faire la conception de votre circuit de façon schématique et de le télécharger ensuite sur la carte.

1. Présentation

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 :

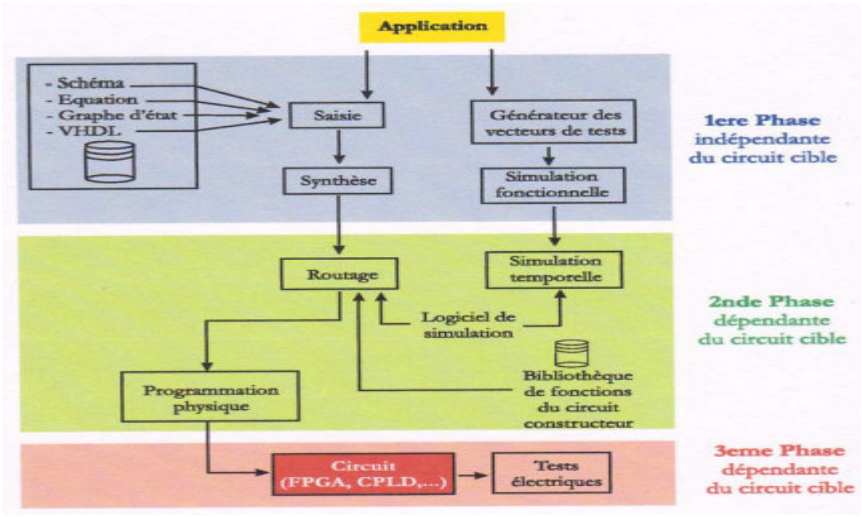


Fig. 1. *La manière de conception circuit programmables*

1.1 Démarrage du logiciel et configuration de base:

Chaque circuit dans Quartus doit faire partie d'un projet pour pouvoir être compilé. Le projet décrit quel matériel est utilisé, et quels fichiers contiennent les circuits et sous-circuits à considérer.

Nous allons ici créer un projet, et dire à Quartus quelle puce nous utilisons.

1.2 Ouverture du logiciel :

- Démarrez le programme Quartus II

Dans le menu démarrer de l'environnement Windows choisissez:

Programmes -> Altera -> Quartus II 7.2

Lors de l'ouverture du logiciel, l'espace de travail illustré à la Figure 2 est présenté à l'utilisateur espace de travail contient:

1. La barre des menus.
2. La barre d'outils.
3. Le navigateur de projet permet de voir les fichiers, les projets et leur hiérarchie.
4. La fenêtre de statut vous permet de connaître l'état d'avancement de la compilation de votre circuit.
5. La fenêtre de message vous avertit des erreurs et autres avertissements durant la compilation ou la programmation

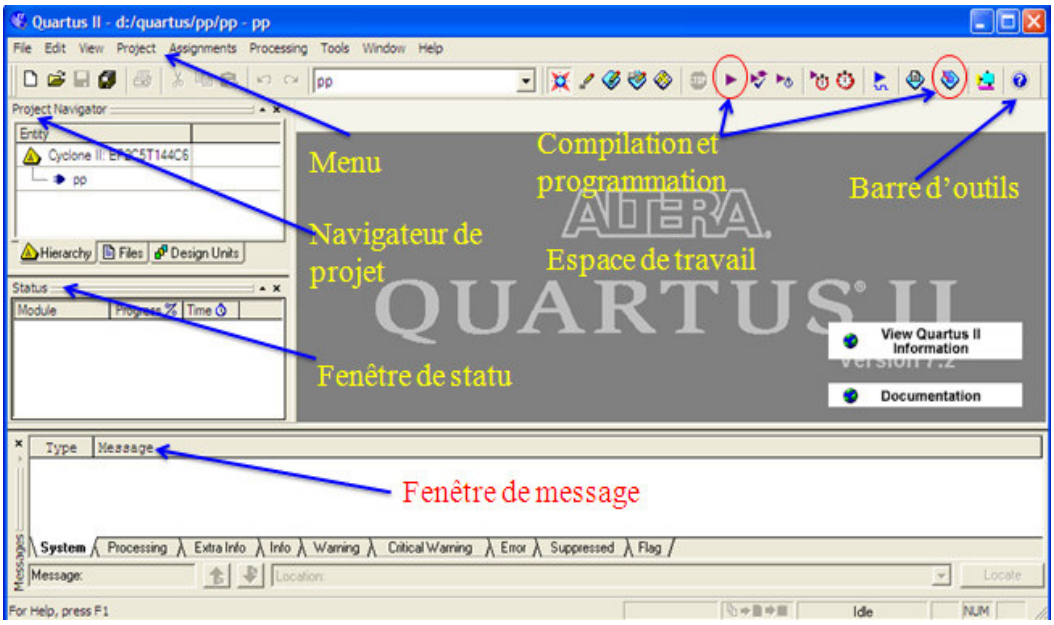


***Fig. 2**. Présentation de l'environnement de conception Quartus II*

2. Création d'un projet : [17]

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).

Afin de créer un nouveau projet aller dans le menu :

File —>New Project Wizard

➢ Dans la figure (3) présenté la fenêtre **"New Project Wizard: introduction"** qui s'ouvre pour fournir de drèves explication, cliquer sur: « **Next** »

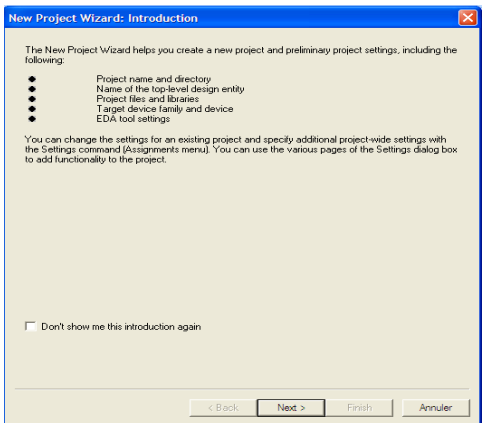


***Fig.3**. Fenêtre d'explication de logiciel.*

Puis se laisser guider. La figure (4) présenté une nouvelle fenêtre **"New Project Wizard: Directory, Name, Top-level Entity [page 1 of 5]"** permettant de configurer le projet apparaît. Dans cette dernière trois champs sont à renseigner :

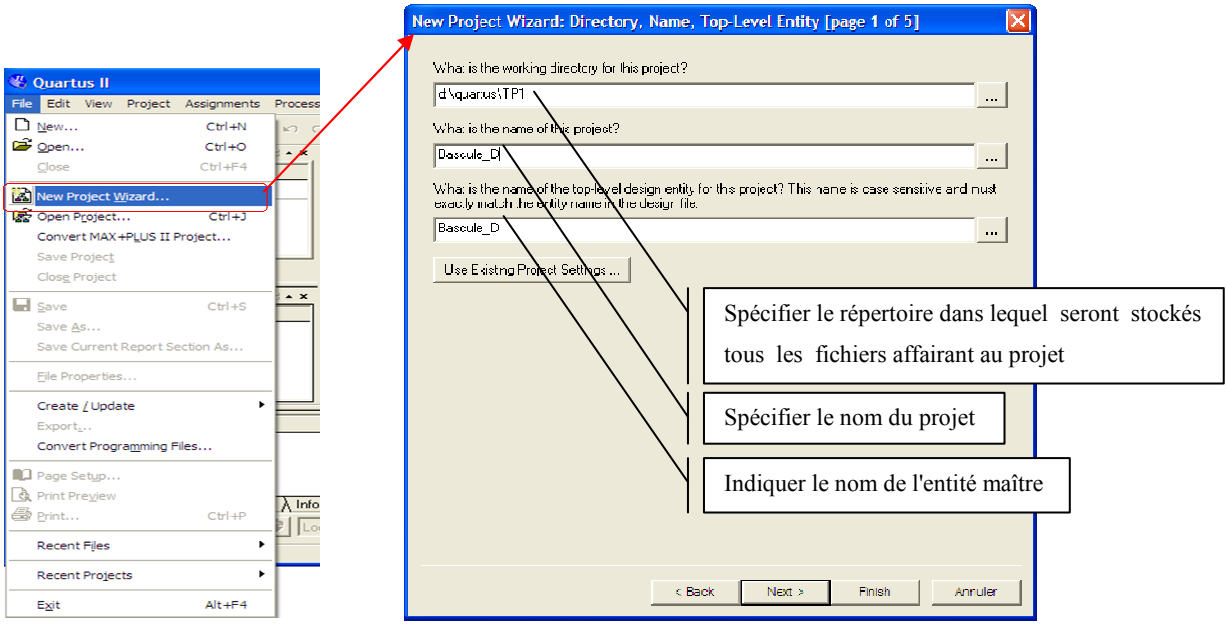


***Fig.4**. Fenêtre de nommé le projet et l'entité et enregistrement.*

Dans la seconde indiquez dans quel dossier vous voulez créer le projet. Spécifiez le dossier **TP1** (D:\...\TP1) créé plus tôt comme répertoire de projet.

Écrivez ensuite le nom de votre projet et mettre le même nom pour l'entité qui englobe les autres (la plus haute du projet). Pour ce projet, nous avons utilisés par exemple le nom « **TP1** ».

Ensuit il faut cliquer sur « **Next** » s'ouvre une fenêtre pour demander que l'emplacement (**directory**) que vous avez choisi n'existe pas. Voulez-vous le crée ? Cliquer sur « **Oui** » pour le créer ou « **Non** » pour annuler.

➢ Dans la figure (5) présenté la fenêtre **"New Project Wizard: Add Files [page 2 of 5]";**

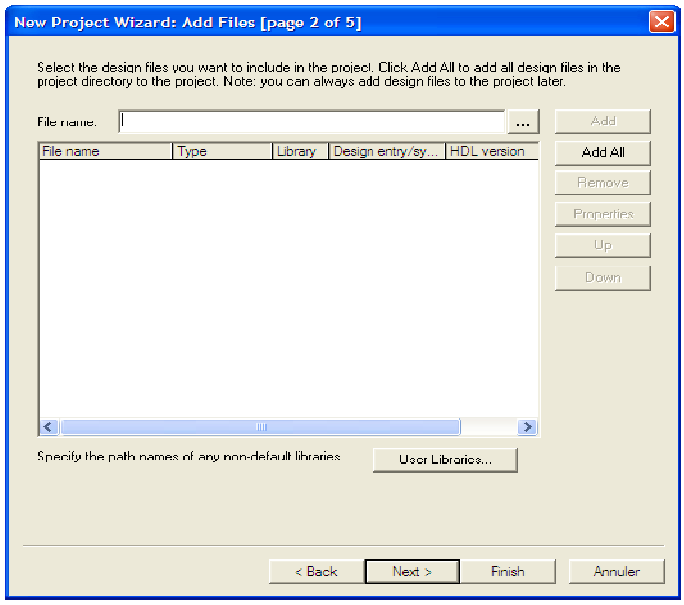


Fig.5. *Fenêtre d'ajouter des fichiers au projet*

Dans cette troisième page, il vous est possible d'ajouter des fichiers au projet, si vous voulez en réutiliser. Pour l'instant, nous n'utiliserons pas cette fonctionnalité, on reclique sur « **Next** ».

➢ La page **"New Project Wizard: Family & Device Setting [page 3 of 5]";**

Dans la quatrième page, vous devrez spécifier le FPGA utilisé. Dans le menu déroulant « **Family** », vous choisissez « **Cyclone II** ».

Par la suite, dans la liste « **Available Devices** » sélectionnez « **EP2C35F672C6** », qui est le FPGA que nous utilisons. Dans la figure (6) présenté comment faire cette étape.

Si on a une carte (DE0, DE1, DE2) de TP pour l'implémentation, on doit choisir le circuit logique programmable de cette carte.

Ainsi déjà on a besoin de spécifier le circuit logique programmable bien qu'il peut être modifié par la suite en cas de nécessité. Après cela on peut cliquer sur « **Next** ».

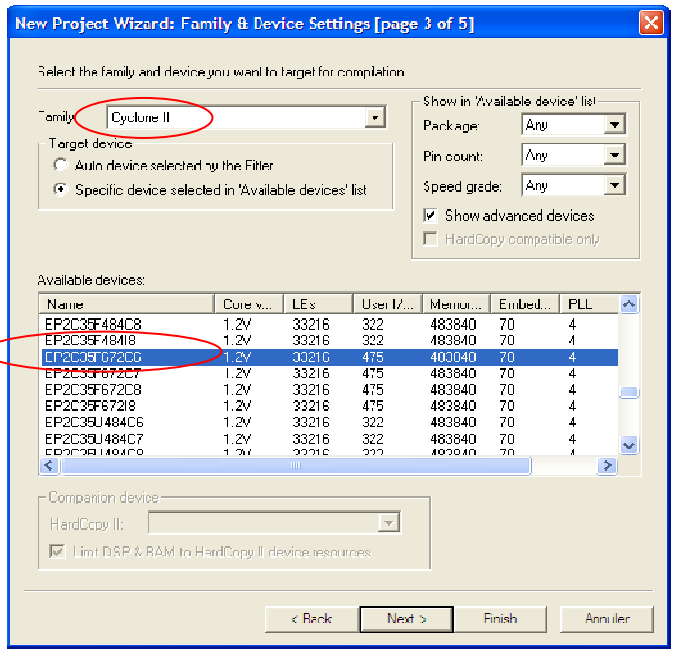


***Fig.6**. Fenêtre de choisir la famille et devise pour la compilation*

➢ Dans la figure (7) présenté la fenêtre **"New Project Wizard: EDA Tool Setting [page 4 of 5]";**

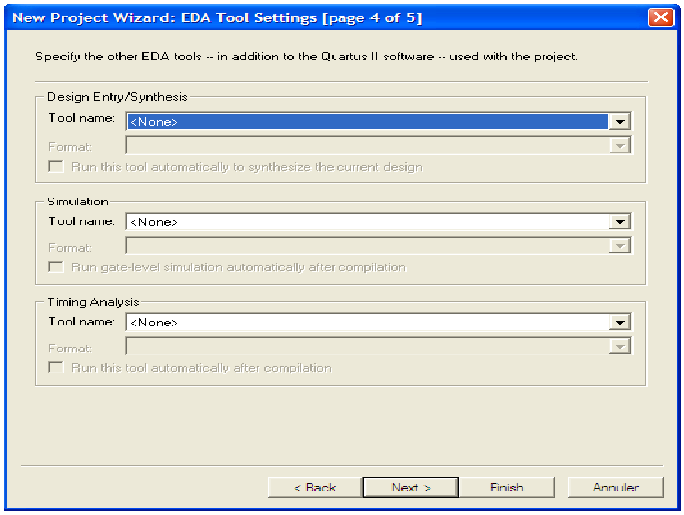


***Fig.7.** Fenêtre de EDA Tool Setting*

Dans la cinquième page, Vous pouvez cliquer sur « **Next** », car il n'y a plus rien à rajouter pour le moment.

➢ La page **"New Project Wizard: Summary [page 5 of 5]";**

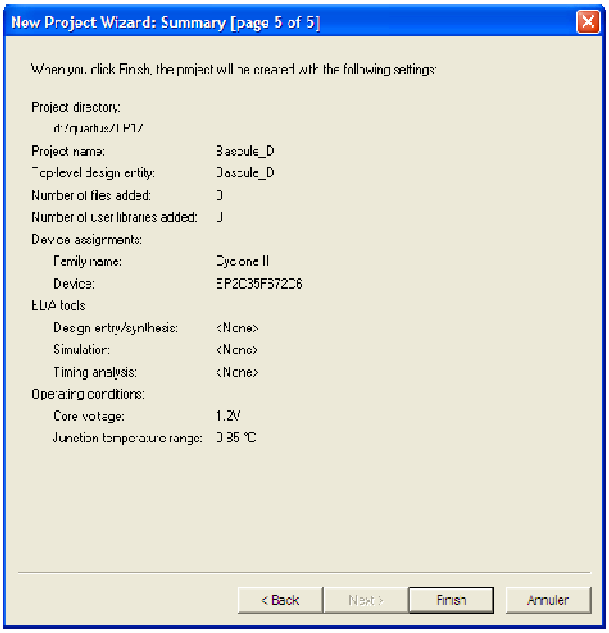


***Fig.8**. Fenêtre pour voir le résumé de ce projet*

Dans cette dernière page, permet de voir le résumé de ce projet comme la figure(8) à savoir: Project Directory, Project Name, Top-level Entity, Device assignments: Family Name, Device…

Si on clique sur « **Finish** », dans le navigateur du projet à gauche la fenêtre on aura:

Project navigator

Entity

Cyclone II: EP2C35F672C6

└── Bascule_D

ANNEXE B : Saisie graphique en assemblant des symboles.

1. Saisie d'un projet: [16]

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

- une saisie graphique en assemblant des symboles.
- une saisie textuelle à l'aide de différents langages (VHDL, Verilog, AHDL,...).

1.1 Création d'un fichier de description schématique :

Pour pouvoir créer un circuit pour dessiner votre circuit, il faut aller dans le menu et créez le fichier d'après cliquer sur :

File → New

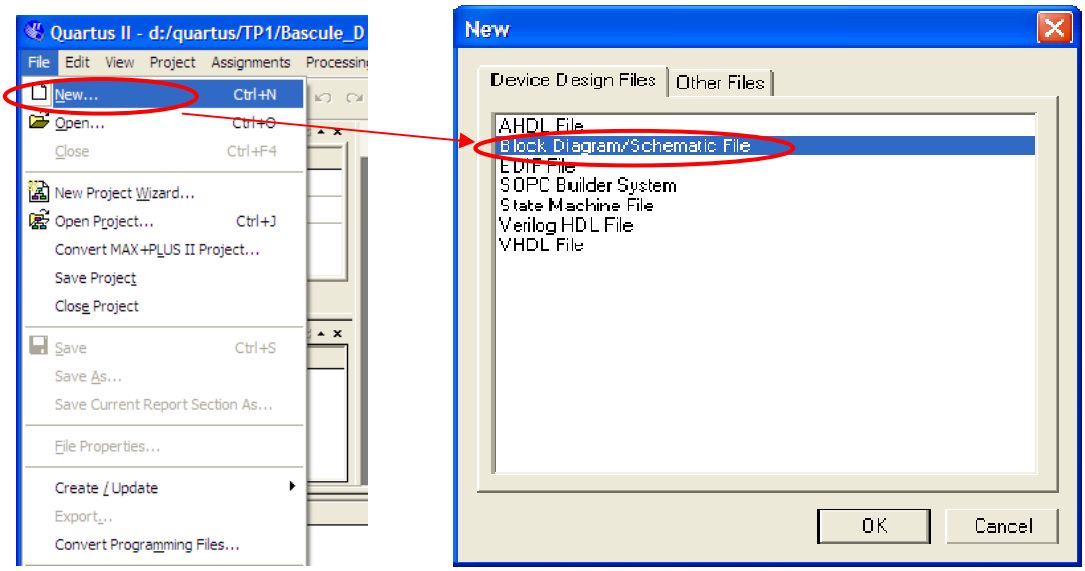


Fig.9. *Fenêtre de création d'un fichier de description schématique.*

Dans l'onglet « **Device Design 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 assurez-vous de sauvegarder le fichier dans votre répertoire de projet.

File → Save as...

Maintenant, il sera possible de dessiner un circuit donné.

1.2 Description schématique d'un circuit:

Dans la barre d'outils schématique figure (10), 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).

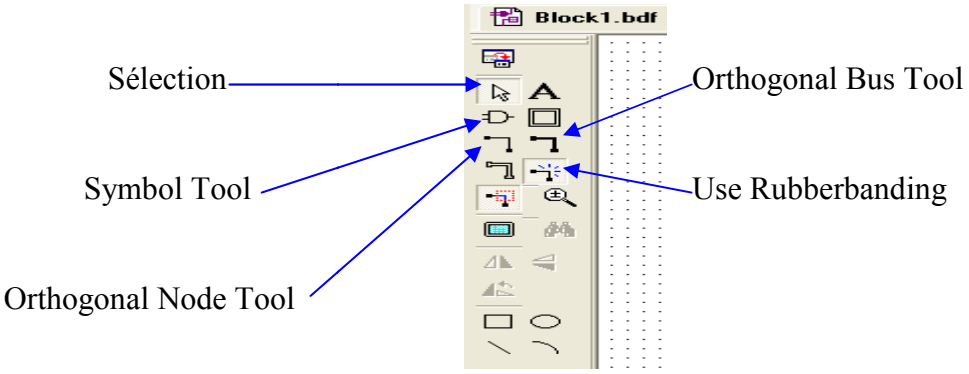


Fig. 10. *Barre d'outils de la fenêtre schématique*

En choisissant « **Symbol Tool** », vous verrez une fenêtre apparaître. Cliquez sur le dossier « **d:/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. 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.

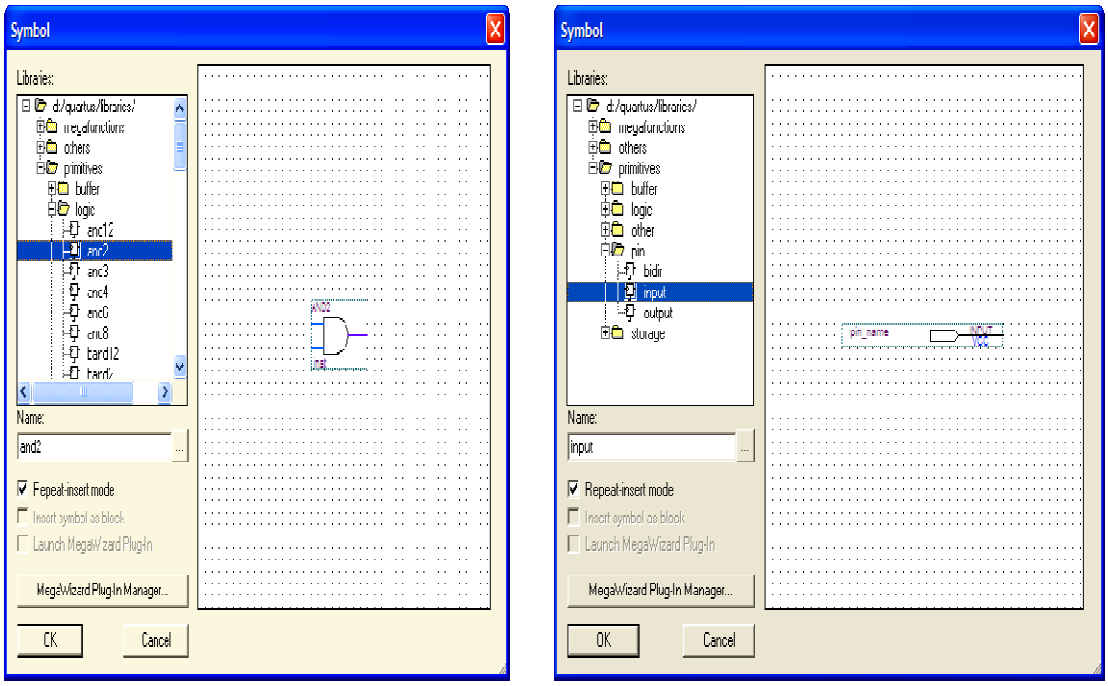


***Fig. 11**. Libraires d'outils de la fenêtre schématique*

Dans libraires d'outils de la fenêtre schématique figure (11), vous avez sélectionnés une porte ET à 2 entrées par exemple (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 ». Dans la figure (12) suivant on réalise circuit OU_EX par exemple:

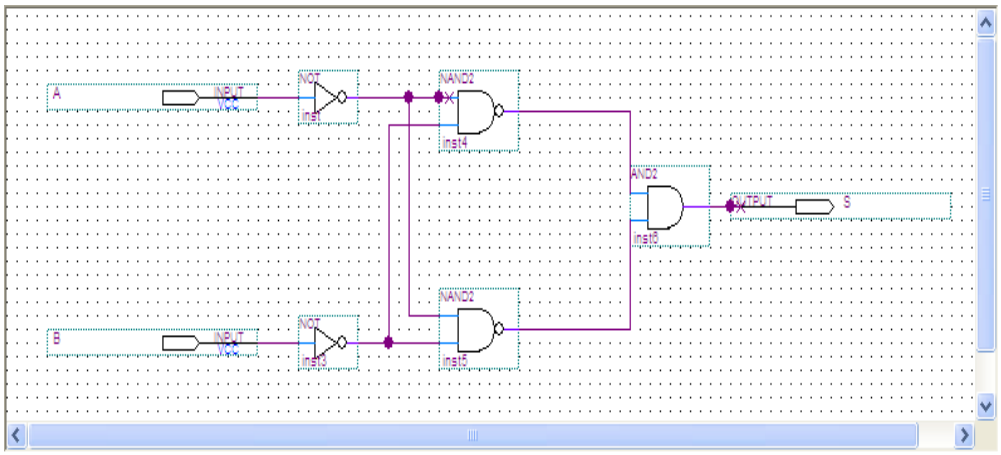


***Fig.12.** Réalisation le circuit OU_EX*

le repérage des signaux entrées/sorties, il suffit d'aller à la librairie et cliquer sur:

+primitive→ +Pin

Puis choisir pin input, pin output ou pin bidirectionnel et l'insérer dans la feuille comme on a fait avec les composants.

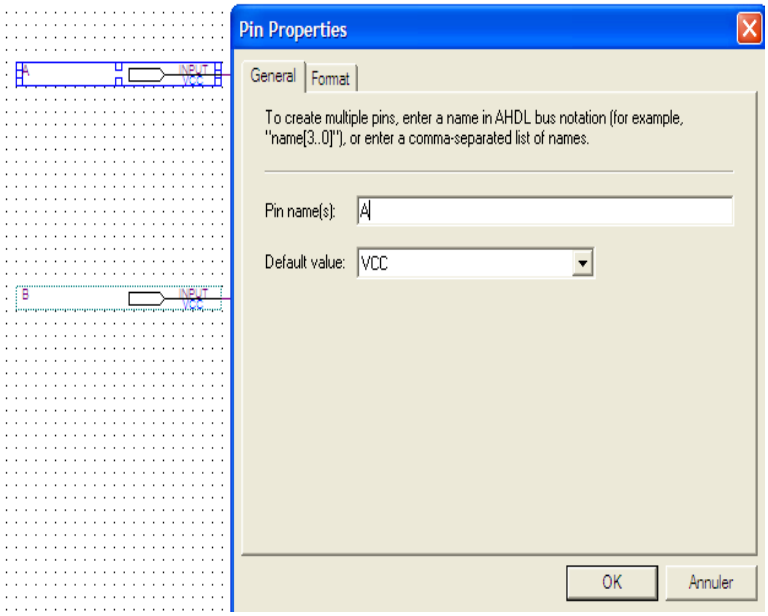


***Fig. 13**. Changement du nom d'une broche dans le circuit.*

Ensuite, il faut nommer les entrées/sorties comme voir dans la figure (13) en utilisant les noms définis dans le fichier utilisé pour l'assignation des broches (DE2_pin.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 sorties.

ANNEXE C : Saisie textuelle à l'aide de langages VHDL

1. *Création d'un fichier de description textuelle :*[15]

La saisie d'un composant décrit en VHDL se fait de la même manière. Pour cela, il faut aller dans le menu et cliquer sur:

File → New

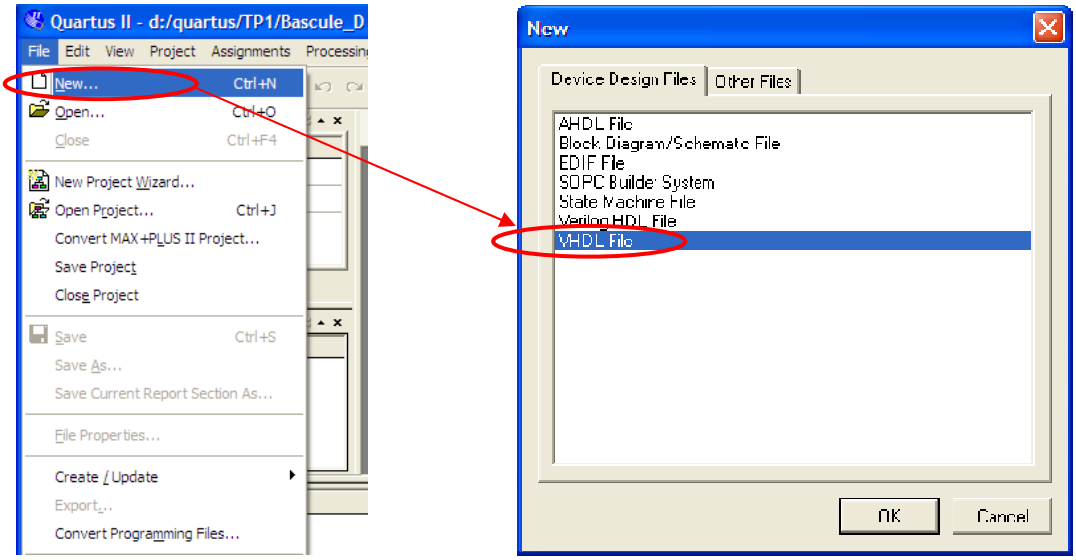


Fig.14.*Fenêtre de création d'un fichier de description textuelle.*

Lorsque la fenêtre « **New**» s'ouvre choisir: "**VHDL File**" Comme voir dans la figure 14. La fenêtre de l'éditeur VHDL apparait. C'est la où le programme VHDL est écrit.

Par exemple, nous avons écrire le programme VHDL d'un OU_EX comme suivant:

```
Library  ieee;
Use ieee.std_Logic_1164.all;
Entity OU_EX is
Port (a, b: in bit;    S: out bit);
End OU_EX;
Architecture Description of OU_EX is
Begin
S<= a xor b ;
End Description;
```

On sauvegarde ce fichier. Pour cela on fait: ***File → Save as***

Dans la fenêtre, ***"Save as"*** on donne un nom au ficher avec l'extension **.vhd** par exemple " **OU_EX.vhd**".

Il ne faut pas oublier de cocher "**Add file to current project**" et d'enregistrer.

On peut utiliser "**VHDL templates**" comme la figure (15) pour cela, on sélectionne:

Edit→ Insert Template → VHDL

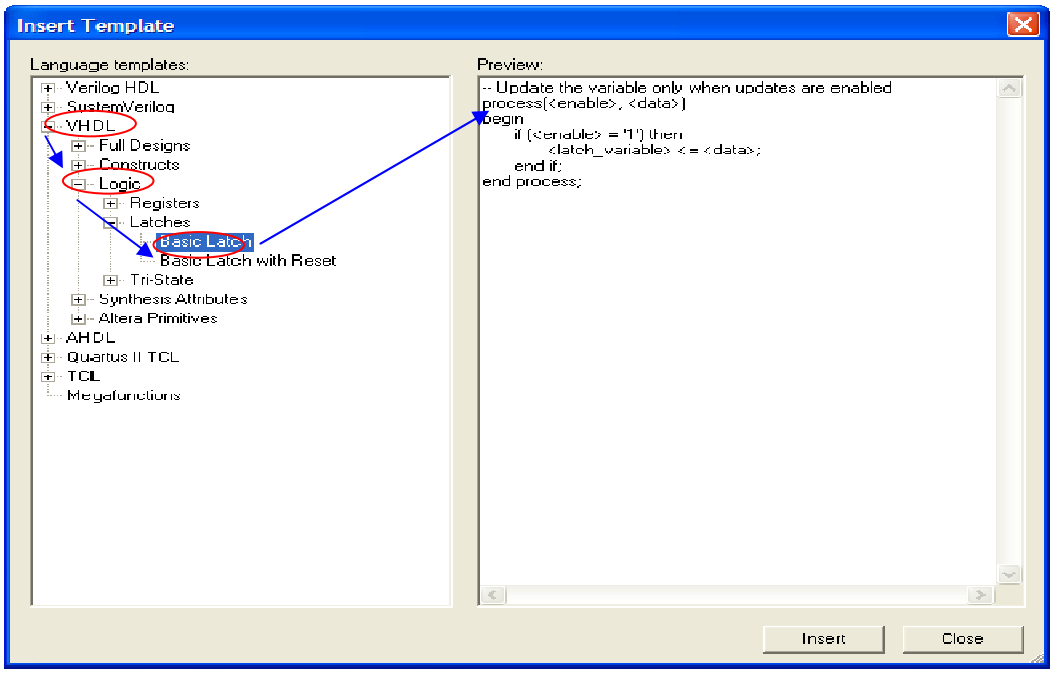


***Fig.15**. Fenêtré de texte circuit programmable.*

→ Pour ajouter des fichiers au projet et voir son contenu:

De manières, à partir du menu pour obtenir la fenêtre "**settings**" ;

- Choisir: "**project**"→ "**Add/Remove Files in Project**"
- Ou bien: "**Assignement**"→"**settings**"

A partir de la fenêtre "**settings**", on sélectionne "**File**" pour voir la liste des fichiers inclus dans le projet. Ainsi on peut ajouter ou supprimer des fichiers.

Une fois que le fichier VHDL est OK, on peut alors créer un symbole graphique qui nous permettra d'insérer le composant dans la feuille graphique initiale. A partir du menu Quartus, choisir: "**File**"→ "**Create/Update**"→"**Create Symbol File from Current File**"

La fenêtre Quartus nous confirme la création du fichier symbole d'après VHDL par le message " **Create Symbol was Successful ok**" et se trouve dans la bibliothèque pour être utilisé éventuellement.

2. *Liste des mots réserves à la syntaxe du langage vhdl* [3]

abs
access
after
alias
all
and
architecture
array
assert
attribute

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

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

wait
when
while
with

xor

ANNEXE D : Conception des circuits synchrones par diagramme d'état : [BMAE /05/2012]

La séquence de comptage est le point de départ dans la conception de circuit synchrone. Elle peut être spécifiée à l'aide d'une table de transition ou d'un diagramme d'état. Dans l'exemple suivant dans la figure 16 nous allons illustrer le processus habituel pour la conception d'un circuit synchrone dont la séquence de comptage est donnée par le diagramme d'état comme suivant :

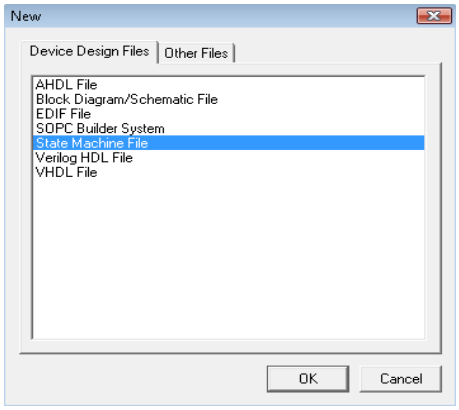


Fig.16.*Fenêtre de choisit la zone de création un graphe d'état.*

On choisir à partir du menu: ***File → New***

Lorsque la fenêtre "**New**" s'ouvre, on sélectionne "**State Machine File**" et on valide par "**Ok**".

D'après s'ouvert cette zone, maintenant, il sera possible de dessiner un graphe d'état donné. Dans la barre d'outil figure (17), vous pouvez voir la «Flèche» qui sert à sélectionner les objets.

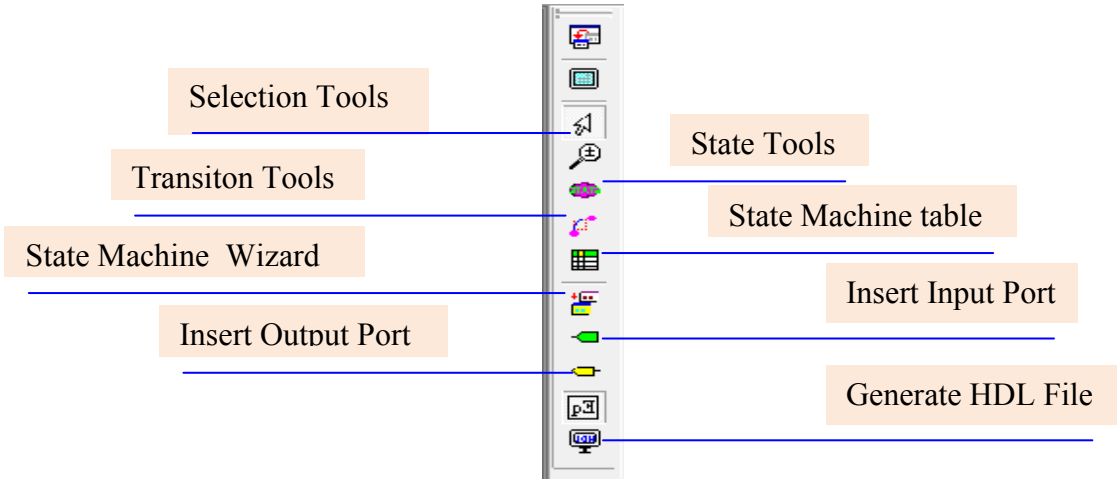


Fig.17*. Barre d'outils de la fenêtre machine d'état.*

✓ Le bouton « **Stat Tool** » vous permet de créer l'état que vous voulez insérer dans la zone de cette fenêtre. Pour dénommé les états il faut faire double clique sur l'état et lorsque la fenetre « **State Properties** » s'ouvre : on donné le nom dans la zone « **State name** » dans le cas « **General** ».

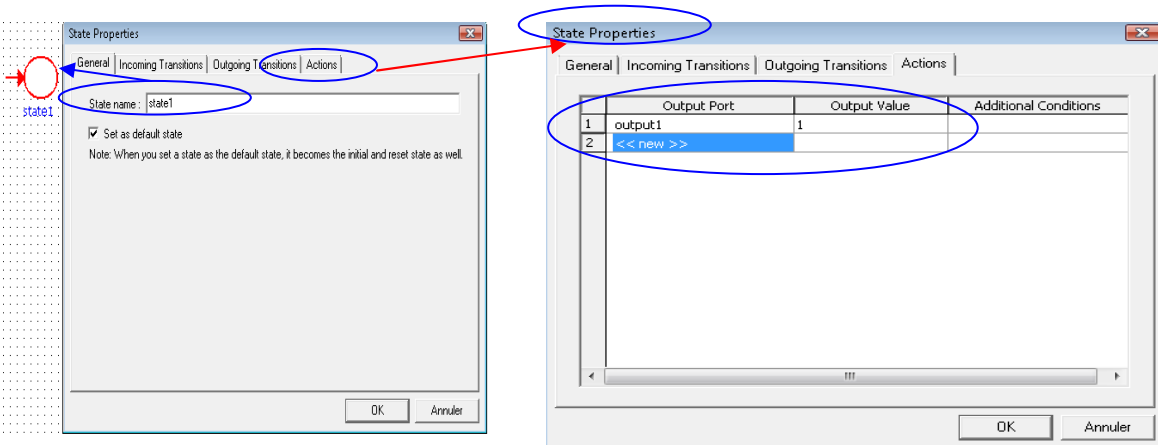


Fig.18. *Fenêtres dénommé et donné les cas d'états.*

Pour le cas « **Action** » nous avons données les valeurs '0' ou '1' dans le cas de sortie (output) et on valide par "**Ok**".

✓ Les deux boutons (1) «**Insert Input Port**» et «**Insert Output Port**» permet d'insérer les entrées et les sorties qui présenté les différents de transitions de la machine d'état.

✓ Le bouton «**Transition Tools**» vous permet de dessiner les transitions conditionnelles et les transitions inconditionnelles d'entrée et de sortie entre vos d'états.

Il possible afficher les cas d'entrées de transition sur cette zone à partir de faire une double clique(2) sur la ligne de transition entre les états et afin s'ouvre la fenêtre «**Transition Properties**»(3) qui indique les différentes équations de transitions.

Dans le cas de transition défini par un sel entrée, on peut exprimer par exemple "**DIN==0**". Dans notre cas nous avons utilisés "**~**" avant chaque entrée qui exprime la négation c'est-à-dire pour "**input1=0**" et "**input2=0**" l'équation défini par

"**input1&˜input2**" et on valide par "**Ok**"(4). La figure 25 explique les étapes précédentes comme suivant:

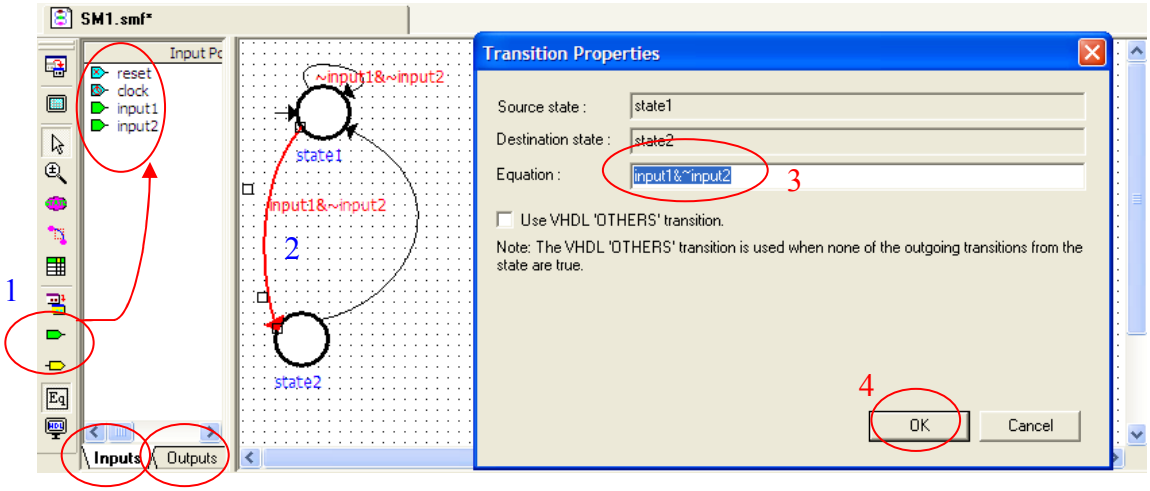


Fig.19. *Fenêtre propriété de transition.*

✓ Le bouton «**Generate HDL File**» permettre donnée le diagramme d'état comme un texte VHDL et ensuit on peut visualiser la synthèse physique et le design (ou le schéma) après utilise les commandes de compilation et de simulation qui proposé dans cette annexe.

Remarque

On peut résumer cette opération d'après utilise le bouton "**State Machine Wizard**" qui donné quatre pages pour indiquer tous les donner de cette machine et l'opération passe comme suit:

➢ **La première page:** "***State Machine Wizard: General [page 1 of 4]***";

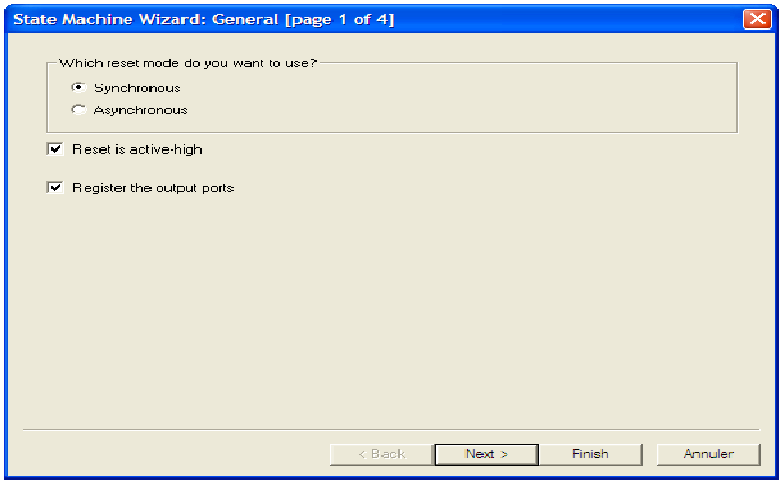


Fig.20*. Fenêtre de choix le mode et l'activation de système séquentiel.*

Dans cette fenêtre on peut choisir la nature ou le mode votre système séquentiel, par "**Synchrone**" ou "**Asynchrone**". Il y a aussi l'activation par "**High**" ou "**Löw**" et comment présenté la sortie de circuit physique par des registre ou non. « **Next** ».

- **La deuxième page:** "*State Machine Wizard: Transition [page 2 of 4]*";

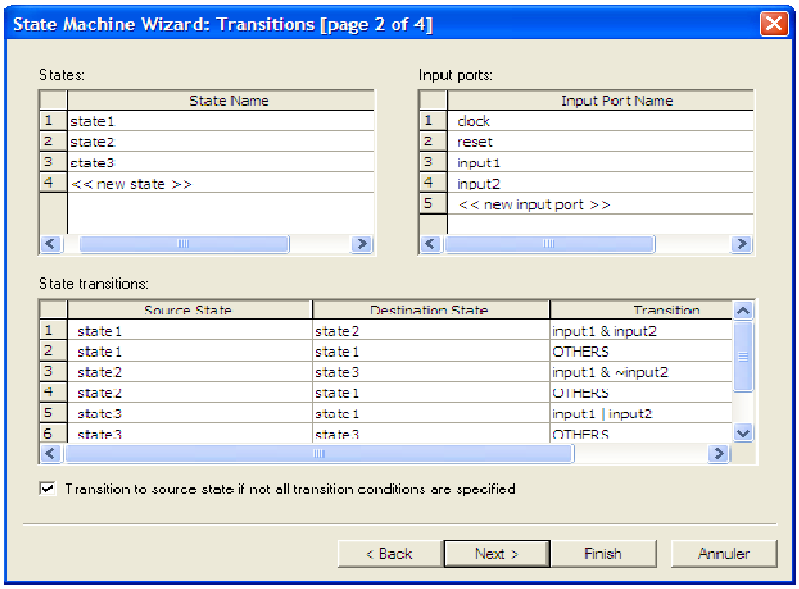


Fig.21. *Fenêtre de création les caractéristiques d'un diagramme d'état.*

Dans cette fenêtre il y a trois case permettre remplirai par les caractéristiques importances pour créer un diagramme d'état. Ainsi la case première, on peut donne les nombres d'états avec les noms et ensuit dans case deuxième on peut présenter les entrées et dernière case nous avons donné tous les différentes de transitions entre les états puis clique sur "**Next**".

- **La troisième page:** "*State Machine Wizard: Action [page 3 of 4]*";

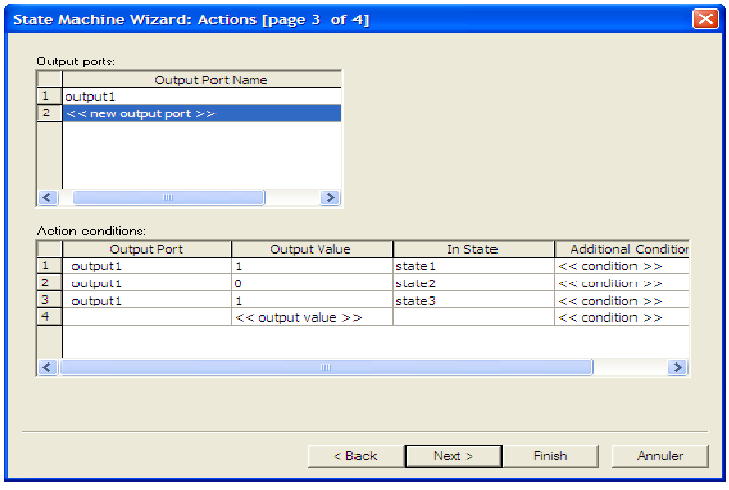


Fig.22*. Fenêtre de création les sorties d'un diagramme d'état.*

Dans cette fenêtre; on peut donner les actions de chaque sorties pour les machine d'états et on reclique sur "**Next**".

- **La troisième page:** "***State Machine Wizard: Summary [page 4 of 4]***";

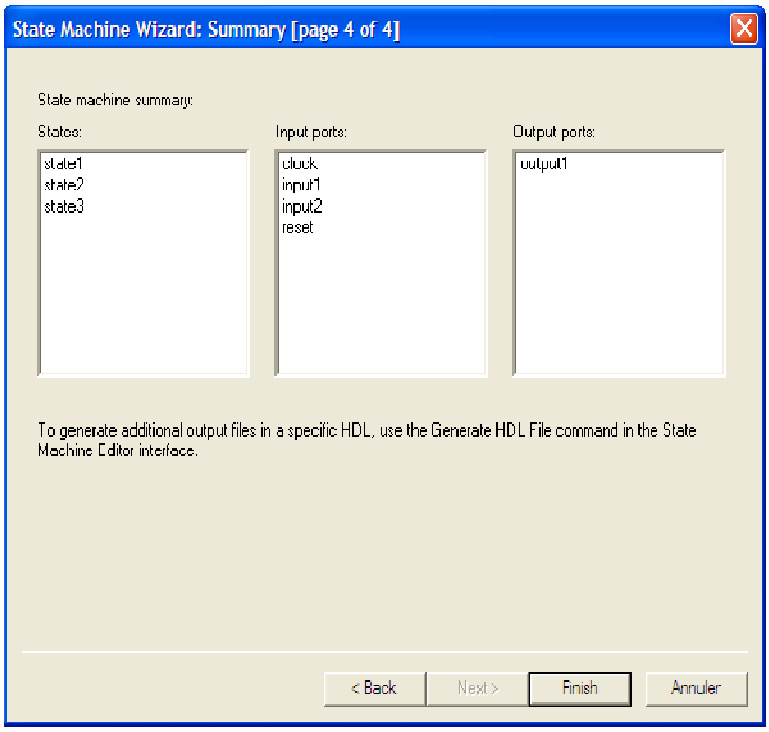


Fig.23. *Fenêtre de sommaire les caractéristiques d'un diagramme d'état.*

Dans cette fenêtre on peut voire un sommaire qui présenté tous les caractéristiques d'un diagramme d'état qui proposer dans les pages précédent. Afin on peut voire le diagramme à partir reclique sur "**Finish**".

ANNEXE E : La compilation [15]

La compilation est un processus permettant de transformer la description d'un circuit (schématique dans notre cas) en un fichier de programmation permettant de programmer le FPGA de sorte à implémenter le circuit décrit. Pour un petit projet comme celui-ci, le processus devrait prendre moins de 30 secondes, mais pour des circuits plus costauds, le processus de compilation peut requérir plus de 30 minutes.

Durant la compilation, Quartus va réaliser 4 étapes :

✓ La transformation des descriptions graphiques et textuelles en une structure électronique à base de portes et de registres : C'est la synthèse logique.

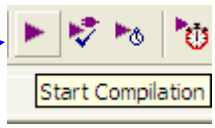
✓ L'étape de Fitting (ajustement) consiste à voir comment les différentes portes et registres (produit par la synthèse logique) peuvent être placés en fonction des ressources matérielles du circuit cible (FLEX 10K20) : C'est la synthèse physique.

✓ L'assemblage consiste à produire les fichiers permettant la programmation du circuit. Ce sont des fichiers au format Programmer Object Files (.pof), SRAM Object Files (.sof), Hexadecimal (Intel-Format) Output Files (.hexout), Tabular Text Files (.ttf), and Raw Binary Files (.rbf).

✓ L'analyse temporelle permet d'évaluer les temps de propagation entre portes et le long des chemins choisis lors de l'étape de « fitting ». .....

Pour lancer le processus de compilation:

Processing→ Start Compilation (Ctrl-L)



Appuyez le bouton "flèche mauve" de la barre d'outils.

Ou cliquer sur:

Processing→ Compilation Tool

Dans la fenêtre " ***Compilation Tool***", il faut cliquer sur «**Start** » et l'opération faire comme voir la figure (24);

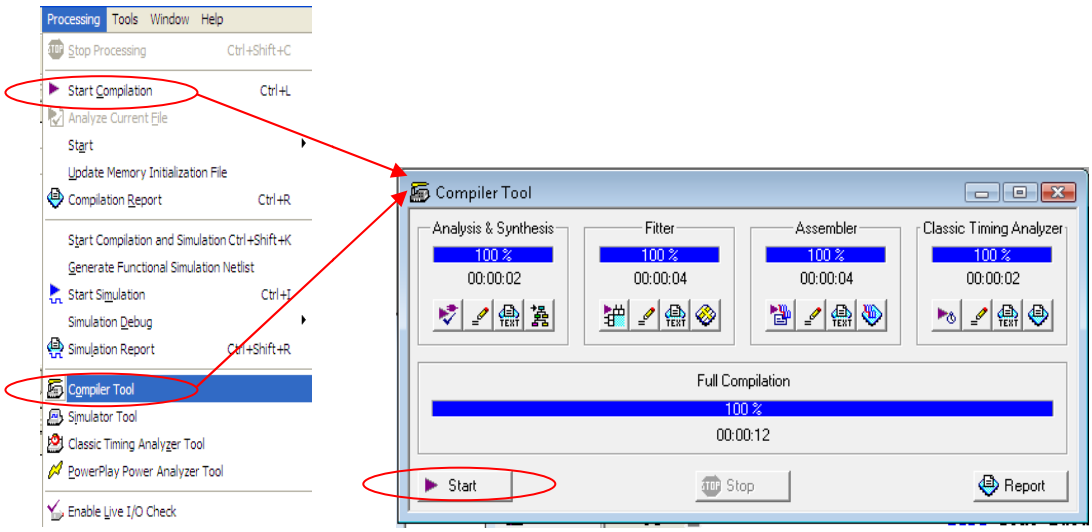


Fig.24. *Fenêtre processus de compilation.*

À la fin du processus de compilation, une fenêtre vous indiquera s'il y a des erreurs dans votre circuit. Si tel est le cas, vous pouvez vous référer aux alertes dans la fenêtre des messages en bas de l'écran. Vous pouvez double-cliquer sur le premier message d'erreur (rouge) pour que l'outil vous pointe la source du problème rencontré. Sinon, si vous avez seulement des avertissements (warnings), n'en tenez pas compte.

La fenêtre "Compilation report Flow Summry" constitue le rapport de la compilation avec "flow Summry" ou sommaire des données.

Pour voir comment le design (ou le schéma) a été transformé en portes et registres, on utilise la commende suivant;

"Tool"→ "Netlist-Viewers"→ "RTL Viewer"

Pour voir le graphe d'état il faut choisir la commende "**State Machine Viewer**".De même, nous pouvons visualiser la synthèse physique en utilisant la commande :

Tools —>Technology Map Viewer

ANNEXE F : La simulation: [16]

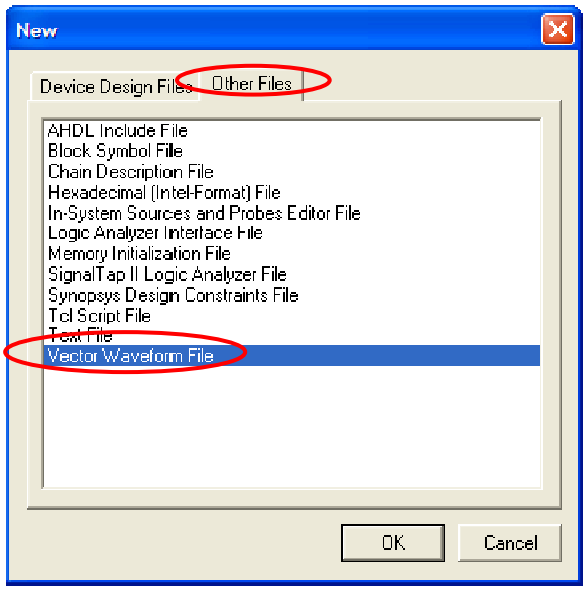
La simulation d'un circuit logique reproduit le comportement de ce dernier au sein d'un environnement contrôlé, fréquemment dénoté banc d'essai. Typiquement, il est plus facile de vérifier le fonctionnement d'un circuit dans un environnement contrôlé, et d'autre simulation requiert typiquement moins de temps que celle requise pour réaliser un circuit physique. Elle est accompagnée par la définition des vecteurs de simulation.

Avant de lancer une simulation, il faut définir les vecteurs de test, c'est-à-dire définir les signaux à appliquer aux entrées. Afin de simuler le design réalisé, il convient de lui injecter des stimuli. Lorsque ces stimuli sont générés à partir d'un fichier de **Bench**.

1. *Le stimulus à partir de "Wave Editor" :*

On choisir à partir du menu: ***File → New***

Lorsque la fenêtre "**New**" s'ouvre, on sélectionne "**Vector Waveform File**" et on valide par "**Ok**".



La fenêtre " **Wavefrom1.Vwf** " apparait et permet de sauvegarder un fichier de vecteur de test, puis enregistrement avec "**Save as**" du menu "File et l'ajouter au projet.

→ Pour régler la durée de la simulation, on utilise la commende "**Edit Time**" du menu "**Edit**" Lorsque la fenêtre "**Edit Time**" s'ouvre comme voir la figure 25, régler

→ le temps "**Time**" en précisant la durée en chiffre et l'unité ps, ns,…Pour valider, il faut cliquer sur "**Ok**".

→ Pour régler la largeur de la grille d'affichage dans la fenêtre de simulation, on choisir la commende "**Grid size…**".

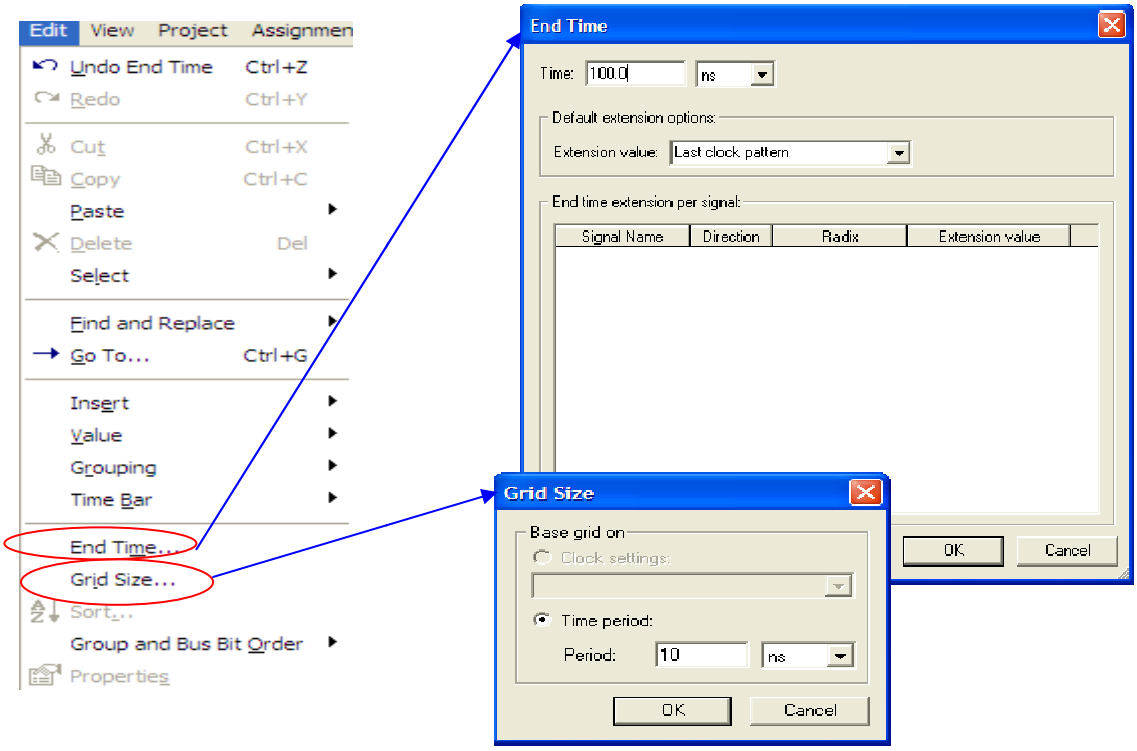


Fig.25.*Fenêtres des réglages la durée et largeur de la grille d'affichage de la simulation.*

→ Pour insérer les différents signaux de simulation, c'est-à-dire les différents signaux d'entrée et de sortie (Input, Clk, Output,…), à partir du menu, on utilise la commande:

Edit→ Insert→ Insert Node or Bus

Dans la fenêtre "**Node Finder**", cela se déroule en 5 étapes :

Cliquer sur "**Node Finder**", ce qui permet de lancer le navigateur de signaux

1. Cliquer sur **"List"** afin de faire apparaître les signaux du design,
2. Sélectionner les signaux voulus,
3. Cliquer sur la flèche correspondante

4. Vérifier que tous les signaux que vous voulez visualiser sont dans "**Selected Nodes**".

5. Cliquer sur **OK** dans la fenêtre "**Insert Node or Bus".**

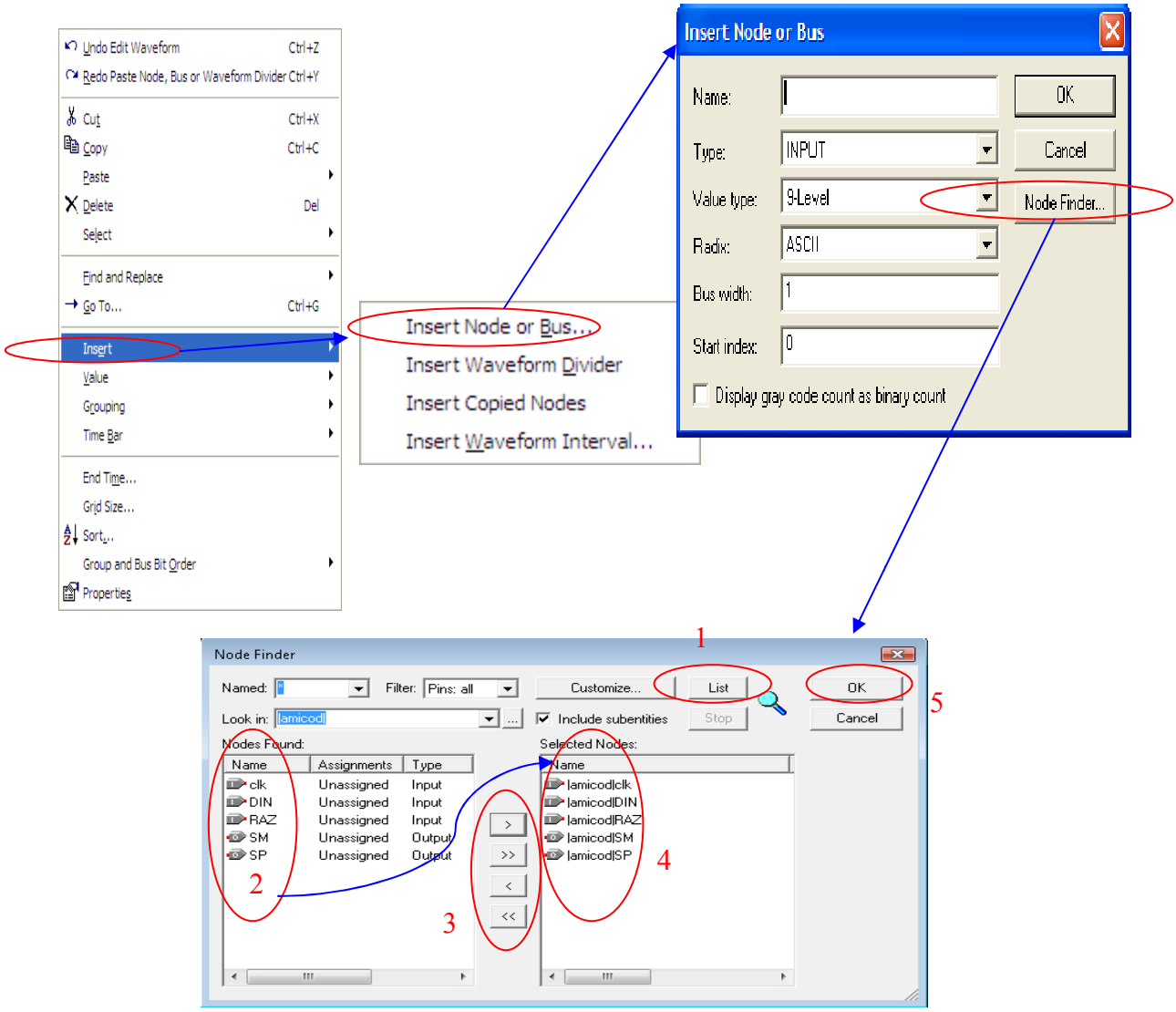


Fig.26. *Fenêtres d'insérer les différents signaux de simulation.*

Les signaux apparaissent maintenant dans le visualiser de signal. Si des bus ont été sélectionnés, il est possible de les "déplier" en bit à bit en cliquant sur le Zone. Dans la figure 27, donnez une fenêtre de la simulation on aura les signaux de simulation sous forme de chronogramme si on fait varier les valeurs des signaux d'entrée.

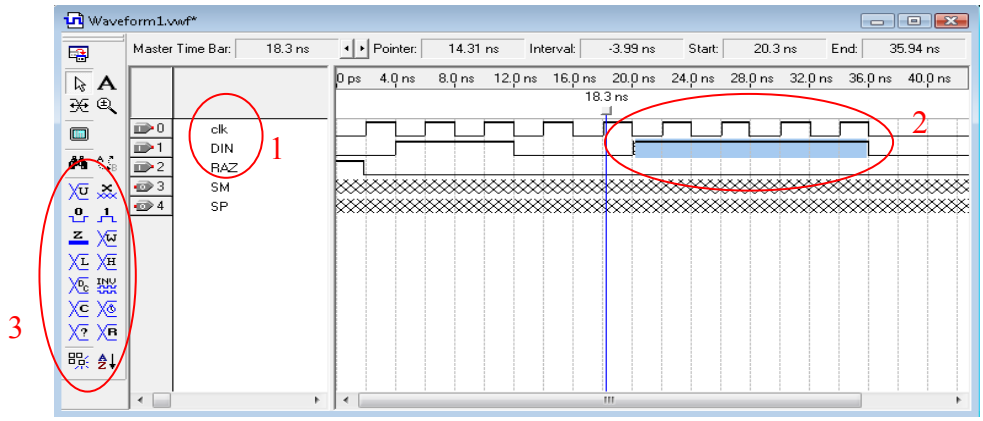


***Fig.27**. Fenêtre d'insérer les différents valeurs pour signaux de simulation.*

Afin de donner des valeurs de stimuli, on sélectionne à la souris une partie du signal (2) et avec le menu (3) on lui attribue une valeur. On notera la possibilité d'insérer automatiquement une horloge (clk) ou un compteur.

En dernier lieu, il ne reste plus qu'à lancer la simulation par la commande :

Tools →Simulator Tool.

3.1. Type de simulation :

Le logiciel Quartus propose deux types de simulation :

A. La simulation fonctionnelle :

Pour la simulation fonctionnelle, il faut sélectionner à partir du menu : "**Assignements**"→"**Settings**" pour ouvrir la fenêtre "**Simulator Settings**". (Voir la figure 28).

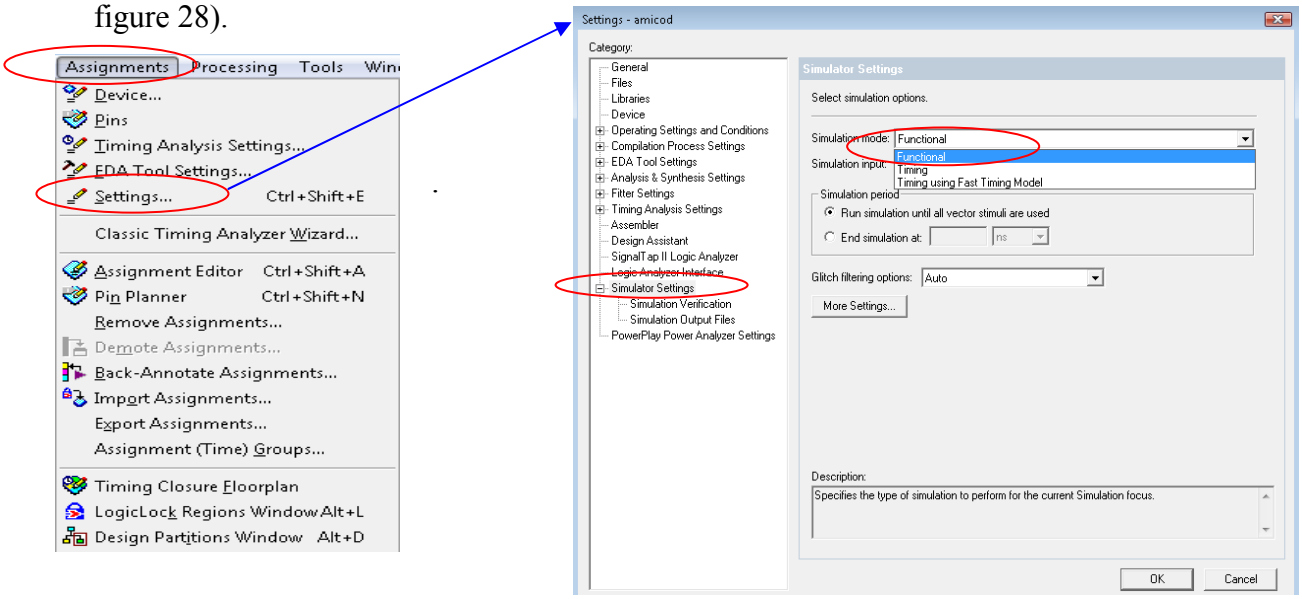


***Fig.28.** Fenêtre de types de simulation fonctionnelle ou temporelle.*

Dans cette dernière, on choisit le mode " **Fonctional** " dans "**Simulation Mode**" puis valider par "**Ok**".

Afin pour lancer la simulation et voir le résultat de cette dernière à partir de la fenêtre " **Simulation Report-Wavefrom1**"(où Waveform1 représente le nom du fichier vwf) nous avons trouvé cette fenêtre toujours dans le menu, sélectionner : "**Processing**"→"**Génératif Functional Simulataion Netlist**" et d'après on revient au menu pour la commende : "**Processing**"→"**Simulation**" comme voir dans la figure29.

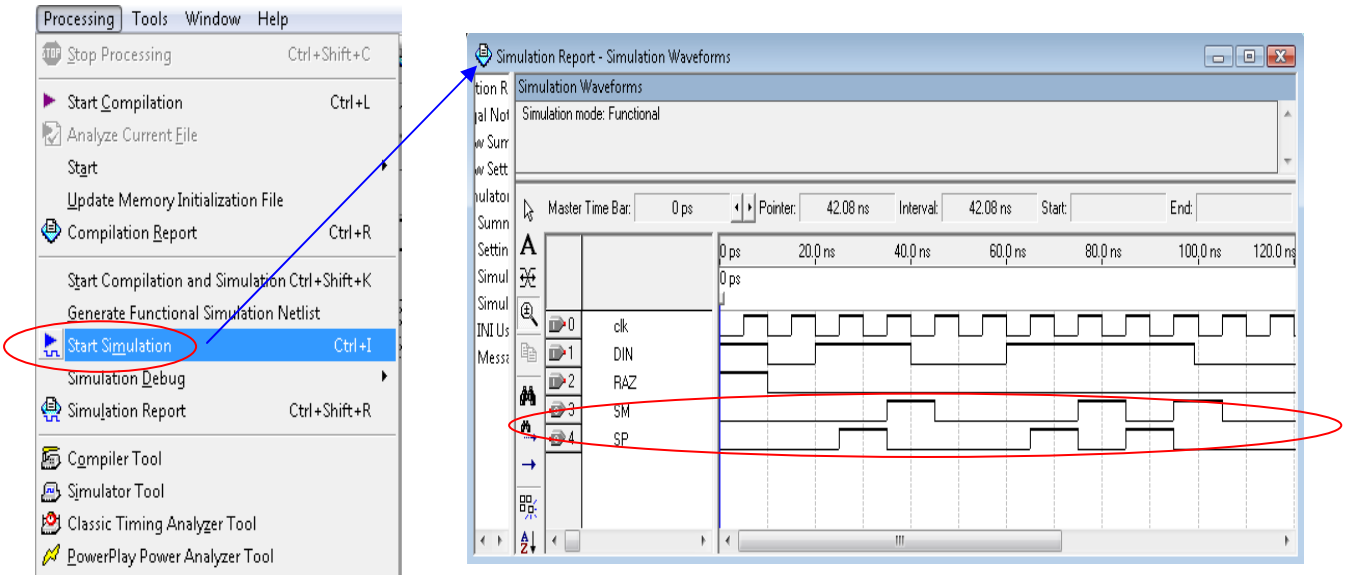


***Fig.29**. Fenêtre du résultat de simulation fonctionnelle.*

B. La simulation temporelle :

Pour la simulation temporelle, il faut sélectionner à partir du menu :

" **Assignement**" →"**Settings**" pour ouvrir la fenêtre "**Simulation Settings**". Dans cette dernière, on choisit le mode "**Timing**" dans "**Simulation Mode**" puis valider par "**Ok**". Puis lancer la simulation de la même façon que précédemment.

Le retard de quelques dizaines de ns peut être observé dans la simulation temporelle.

ANNEXE G : Programmation de la carte DE2 [17]

Le circuit (ou design) une fois compilé et simulé avec succès, peut être chargé dans le composant programmable FPGA qui se trouve sur l'une des cartes DE0, DE1, DE2 d'Altera disponible au laboratoire. Ainsi FPGA peut être programmé et configuré pour recevoir le circuit à réaliser.

La configuration se fait selon deux modes connus comme :

✓ **JTAG mode** (**J**oint **T**est **A**ction **G**roupe) :

Dans le mode JTAG, les données de configuration sont chargées directement dans le FPGA device. Il est nécessaire d'avoir le pilote «**USB Blaster Driver** » installé. Les informations de la configuration sont perdues lorsque l'alimentation est coupée.

✓ **AS mode** (Active **S**erial) :

Dans le mode AS, la configuration du circuit fait appel à une mémoire flash qui stocke les données de configurations.

Le choix entre ces deux modes est rendu possible sur la carte DE1, DE1 grâce au commutateur « **RUN/PROG** ». La position RUN sélectionne le mode JTAG tandis que la position PROG sélectionne le mode AS.

A. Affectation des pins:

Afin de choisir quelle broche physique du circuit doit être connectée, lancer l'outil d'assignement de pins. Dans le menu, on clique sur "**Assignement**" pour disposer de trois choix possible à savoir:

1) Si on clique sur "**Assignements Editor**": lorsque la fenêtre s'ouvre, dans "**catégorie**", il faut choisir "**Pins**".

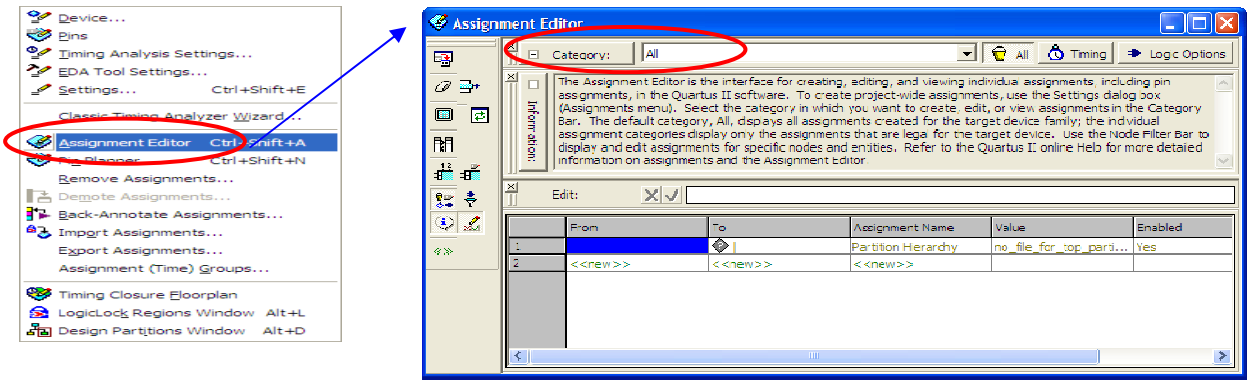


Fig.30. *Fenêtre d'Assignation d'une location physique aux pins.*

D'après cliquer deux fois sur "**New**" au dessous de "**To**", la liste de toutes les entrées et toutes les sorties du design (input te output) apparait, il faut choisir l'entrée ou la sortie à laquelle on veut affecter le N^0 de pin.

2) Si on clique sur "**pin**": lorsque la fenêtre s'ouvre, il faut aller à "**Filter: Pins**" et on clique sur la flèche comme voir dans la figure (31):

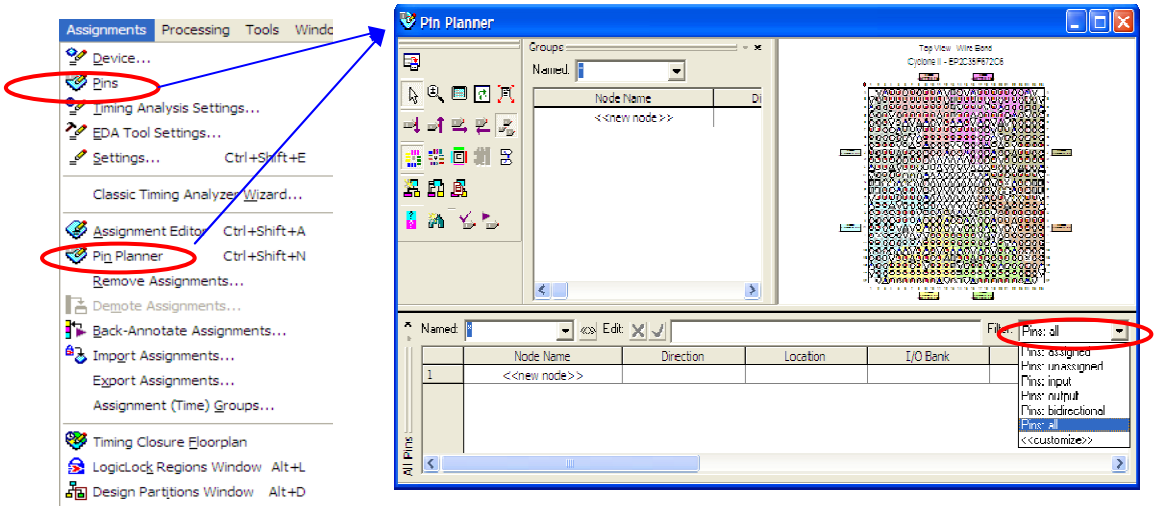


Fig.31. *Fenêtre de choix le plane de pin dans la carte.*

Lorsque la liste des pins du circuit logique programmable s'affiche, il faut choisir celle qui convient. A la fin on sauvegarde colle précédemment.

3) Dans la fenêtre de la même figure, si on clique sur "**Pin Planner**": cette dernière s'ouvre pour l'affectation des pins, la colonne "**Node Name**" représente la liste des entrées et des sorties. La "**Direction**" est représentée par la seconde colonne (input et output). Dans la troisième colonne, "**Location**" on clique deux fois pour afficher les pins du circuit logique programmable et choisir celle qui convient. La "Location" doit être remplir pour chaque entrée et sortie. On sauvegarde à la fin de la même manière.

B. Programmation du circuit :

La programmation du circuit se fait via le protocole JTAG (Joint Test Action Group). Pour cela vérifier que les connections entre le PC (port parallèle) et la carte via le module **Byte-Blaster** sont opérationnelles. Si tout est bon et que la carte Altera est sous tension et en place le commutateur « **RUN/PROM** » dans la position

« **RUN** », Puis aller le menu et clique sur **Tools →Programmer**, puis cliquer sur le bouton **Autodetect** (1).

Si le PC ne détecte pas la carte, une erreur doit apparaître du type « **Unable to scan device Chain. Hardware is not connected** ».Vérifier dans la partie 2 que le fichier « **.sof** » est bien là et que la case **Program/Configure** est cochée (3).

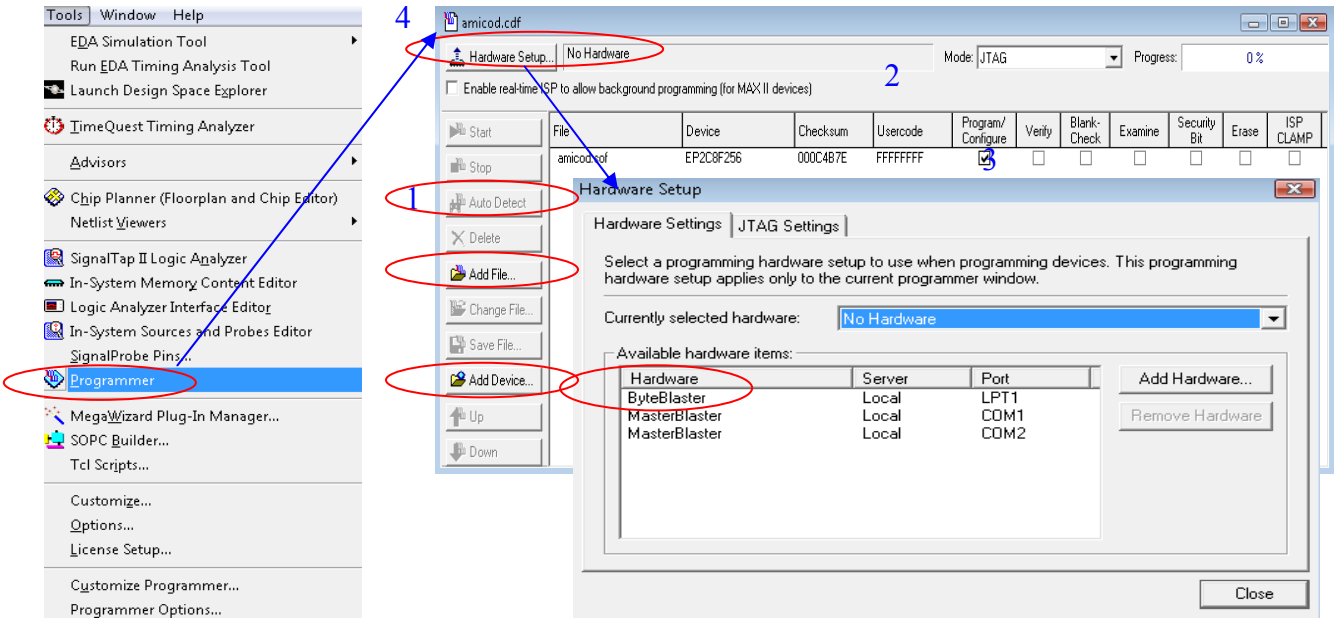


Fig. 32. *Fenêtre de programmer et configurer dans la carte DE2.*

Aussi, si le « **USB-Blaster** » n'est pas choisir par défaut cliquer sur puis cliquer sur « **Hardware Steup** » et dans la fenêtre correspondante, choisir « **USB-Blaster** » (4). La figure 31 données la méthode pour faire cette dernière opération.

Si tout est correct, on peut lancer la programmation en utilisant la commande « **Start »** de cette fenêtre où on peut observer sa progression.

Pour s'allume une LED sue la carte, dire que la configuration est chargée complètement avec succès.

C. L'essai du circuit implanté dans le FPGA :

Une fois que la programmation et la configuration sont achevées on peut procéder à l'essai et la vérification du fonctionnement de ce circuit implémenté dans le FPGA. Dans la carte Altera (DE0, DE1, DE2) sachant que les entrées du circuit correspondent aux entrées précisées dans le FPGA.

Par exemple si le circuit a deux entrées a et b correspondant aux deux commutateurs ou switchers SW0 et SW1, on peut vérifier son fonctionnement en attribut les 4 combinaisons à l'entrée pour observer la sortie sue LEDG0 ce qui revient à relever la table de vérité.

SW1	SW0	LEDG0
0	0	!
0	1	!
1	0	!
1	1	!

D. Présentation de la carte DE-2 :

Quand vous voudrez utiliser la carte, il faudra que vous vérifiiez si l'alimentation est branchée et si la carte s'allume en pesant le bouton de démarrage. Lorsque la carte s'allume, vous verrez toutes les DEL clignoter et l'affichage hexadécimal incrémenter de 0 jusqu'à F en même temps. Il y aura aussi le LCD qui affichera « Welcome to the Altera DE2 Board ».

Dans le bas de la carte, il y a 18 boutons de type « Switch » et 4 boutons poussoir. Ce sont les boutons que nous allons utiliser tout au long de la session. Quand vous voudrez programmer la carte, vous devrez brancher le port USB dans la prise « USB Blaster Port » en haut à gauche à côté de la prise d'alimentation La documentation venant de la carte peut être trouvée dans le répertoire suivant (Q :) → DE2 → DE2_user_manual → DE2_user_manual.pdf

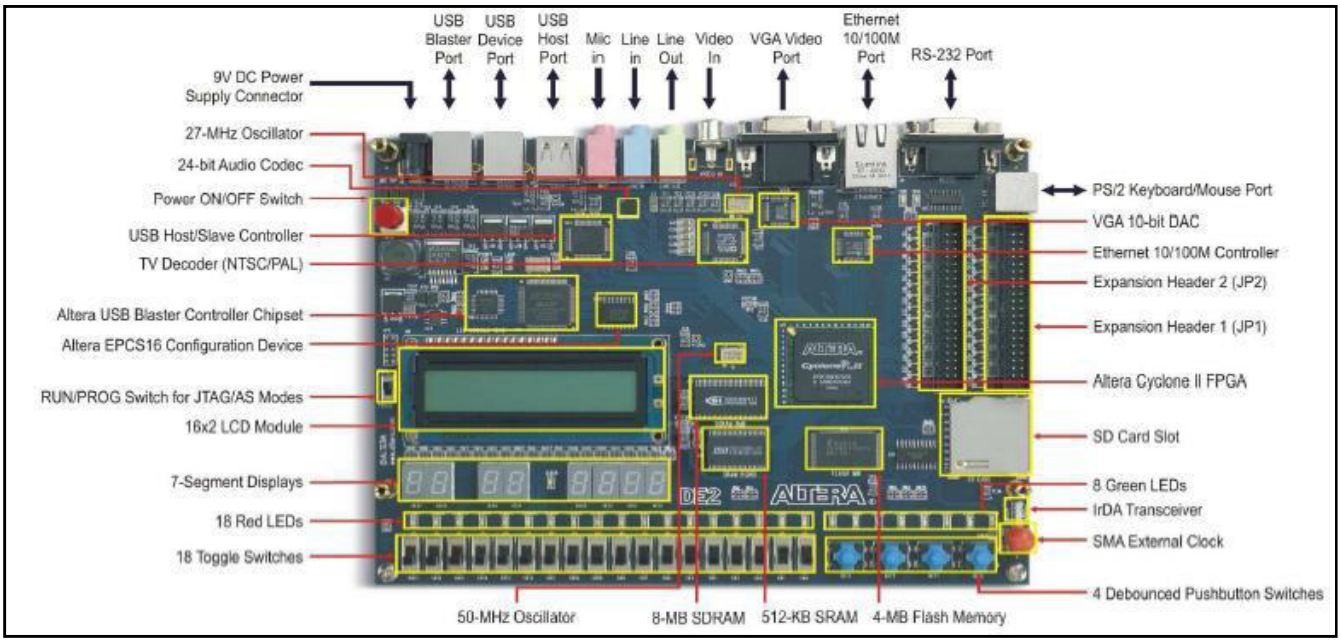


Fig.33:*La carte de développement DE2.*

ANEXE H : CODE VHDL POUR "CODEC AMI" [BMAE/05/2012]

```
--Codeur AMI décrit en VHDL
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY CODEC IS
PORT (       RAZ: IN STD_LOGIC:= '0';
             CLK: IN STD_LOGIC;
             DIN: IN STD_LOGIC:= '0';
            DOUT: OUT STD_LOGIC);
END CODEC;
ARCHITECTURE DESCRIPTION OF CODEC IS
SIGNAL SP, SM: STD_LOGIC;
Component CODAMIVHDL
 PORT (RAZ: IN STD_LOGIC:= '0';
          CLK: IN STD_LOGIC;
          DIN: IN STD_LOGIC:= '0';
           SP: OUT STD_LOGIC;
           SM: OUT STD_LOGIC);
END Component;
Component DECAMIVHDL
PORT (RAZ, SP, SM, CLK: IN STD_LOGIC;
                  DOUT: OUT STD_LOGIC);
END component;
BEGIN
U1: CODAMIVHDL
  Port map (RAZ, CLK, DIN, SP, SM);
 U2: DECAMI2
   Port map (RAZ, SP, SM, CLK, DOUT) ;
END DESCRIPTION;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY CODAMIVHDL IS
   PORT (RAZ: IN STD_LOGIC:= '0';
            CLK: IN STD_LOGIC;
            DIN: IN STD_LOGIC:= '0';
             SP: OUT STD_LOGIC;
             SM: OUT STD_LOGIC);
END CODAMIVHDL;
ARCHITECTURE BEHAVIOR OF
CODAMIVHDL IS
   TYPE type_fstate IS (ZERO1, V_PLUS,
V_MOINS, ZERO2);
   SIGNAL fstate: type_fstate;
   SIGNAL reg_fstate: type_fstate;
BEGIN
   PROCESS (CLK, RAZ, reg_fstate)
   BEGIN
      IF (RAZ='1') THEN
         fstate <= ZERO1;
      ELSIF (CLK='1' AND CLK' event) THEN
         fstate <= reg_fstate;
      END IF;
   END PROCESS;
   PROCESS (fstate, DIN)
```

1

```
BEGIN
   CASE fstate IS
      WHEN ZERO1 =>
         IF ((DIN = '0')) THEN
            reg_fstate <= ZERO1;
         ELSIF ((DIN = '1')) THEN
            reg_fstate <= V_PLUS;
         -- Having else block to avoid latch inference
         ELSE
            reg_fstate <= ZERO1;
         END IF;
         SP <= '0'; SM <= '0';
      WHEN V_PLUS =>
         IF ((DIN = '1')) THEN
            reg_fstate <= V_MOINS;
         ELSIF ((Din = '0')) THEN
            reg_fstate <= ZERO2;
         -- Having else block to avoid latch inference
         ELSE
            reg_fstate <= V_PLUS;
         END IF;
         SP <= '1';   SM <= '0';
      WHEN V_MOINS =>
         IF ((DIN = '0')) THEN
            reg_fstate <= ZERO1;
         ELSIF ((DIN = '1')) THEN
            reg_fstate <= V_PLUS;
         -- Having else block to avoid latch inference
         ELSE
            reg_fstate <= V_MOINS;
         END IF;
         SP <= '0'; SM <= '1';
      WHEN ZERO2 =>
         IF ((DIN = '0')) THEN
            reg_fstate <= ZERO2;
         ELSIF ((DIN = '1')) THEN
            reg_fstate <= V_MOINS;
         -- Having else block to avoid latch inference
         ELSE
            reg_fstate <= ZERO2;
         END IF;
         SP <= '0';    SM <= '0';
      WHEN OTHERS =>
         SP <= 'X'; SM <= 'X';
         Report "Reach undefined state";
   END CASE;
  END PROCESS;
END BEHAVIOR;
--décodeur AMI avec trois etat
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY DECAMI2 IS
PORT (RAZ, SP, SM, CLK: IN STD_LOGIC;
                  DOUT: OUT STD_LOGIC);
END DECAMI2;
```

2

```
-- Architecture
ARCHITECTURE DECAMI OF DECAMI2 IS
TYPE ETAT_DECO IS (ZERO, UN, DEUX);
SIGNAL etat_present, etat_futur: ETAT_DECO
:=ZERO;
BEGIN
     REGETAT: PROCESS (CLK, RAZ, etat_futur)
      BEGIN
          IF RAZ='1'THEN etat_present <= ZERO;
ELSIF CLK'EVENT and CLK='1' THEN
               etat_present<=etat_futur;
          END IF;
          END PROCESS REGETAT;
     COMBETATOUT: PROCESS (SP, SM,
etat_present)
      BEGIN
          CASE etat_present IS
           WHEN ZERO => DOUT <='0';
              IF (SP='1' and SM='0') THEN
                 etat_futur <= DEUX;
              ELSIF (SP='0' and SM='1') THEN
                 etat_futur <= UN ;
              ELSE etat_futur<= ZERO ;
              END IF;
           WHEN UN => DOUT <= '1';
              IF (SP='1' and SM='0') THEN
                etat_futur<=DEUX ;
              ELSE etat_futur<= ZERO;
               END IF;
            WHEN DEUX => DOUT <= '1';
               IF (SP='0' and SM='1') THEN
                  etat_futur<= UN;
                ELSE etat_futur<=ZERO;
              END IF;
            WHEN others => etat_futur <= ZERO;
            END CASE;
            END PROCESS COMBETATOUT;
 END DECAMI;
                                3
```

BIBLIOGRAPHIE

[1] Philippe Larcher "VHDL, Introduction à la synthèse logique", © Eyrolles, Paris, 1997, ISBN 2-212-09584-8

[2] Thierry Schneider "VHDL: méthodologie de design et techniques avancées", © DUNOD, Paris, 2001, ISBN 2-10-005371-X

[3] J. Weber, S. Moutault, M. Meaudre "Cours; Le langage VHDL, du langage au circuit, du circuit au langage", © DUNOD, Paris, 2007, ISBN 978-2-10-050191-5.

[4] Kahoul Nadhir "Cours; le module TEC 480", département d'électronique, , Univ. M^{ed} Kheider, Biskra, Année universitaire 2004/2005.

[5] genelaix.free.fr/IMG/pdf /aide_VHDL_quartus.pdf

[6] www2.toulouse.iufm.fr/iufm/cours/vhdl/vhdl.pdf

[7] Philippe Lecardonnel, Philippe Letenneur "Introduction à la Synthèse logique V.H.D.L", Lycée Julliot de la Morandière – GRANVILLE – 2001.pdf

[8] Claude Guex "Introduction au VHDL"/ E-mail : guex@einev.ch /eivd - Ecole D'ingénieurs Du Canton de Vaud, rte de Cheseaux 1, 1400 Yverdon-les-Bains, Suisse, 1997 Date: 15/04/2012.pdf

[9] D.Giacona "VHDL – Logique programmable/ Partie 6 - Logique combinatoire- Logique Séquentielle", École Nationale Supérieure d'Ingénieur Sud Alsace, FRANCE .pdf

[10] www.telecom-bretagne.eu/ Techniques d'intégration/ Logique séquentielle.pdf

[11] Version initiale de ce manuel: Manuel VHDL pour la synthèse automatique, Yves Sonnay, Diplomant de l'EIVD, octobre 1998.pdf

[12] J. AUVRAY "Systèmes électroniques ", Université Pierre et Marie Curie, 2000-2001.pdf

[13] Arnaud BOURNEL « Electronique Systèmes de Télécommunications » Partie I, , Université Paris XI 2001-2002.pdf

[14] T. BLOTIN "le langage de description VHDL", , Lycée Paul-Eluard 93206 SAINT-DENIS. pdf

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

[16] Boniface Mbouzao "Construction et Fonctionnement des Systèmes Informatiques". Université canadienne 2010 /uOttaw.pdf

[17] ELE1300 - Circuits Logiques I Laboratoire 1 : Utilisation du logiciel Quartus II D'Altera.pdf