République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

ECOLE NATIONALE POLYTECHNIQUE
Département d'électronique

Projet de fin d'études

Pour l'obtention du diplôme
d'Ingénieur d'Etat en Electronique

THEME :

IMPLEMENTATION D'UNE COMMANDE MPPT FLOUE SUR FPGA

Présenté par :

M. DAOUD Housseyn

Membres du jury

M.M.HADDADI	**président**
M. C.LARBES	**rapporteur**
M.MS.AIT CHEIKH	**examinateur**

Promotion : juin 2006

Ecole Nationale Polytechnique
10, Avenue Hacen Badi, El-Harrach, Alger.

INTRODUCTION GENERALE

CHAPITRE I. SYSTEMES PHOTOVOLTAÏQUES

CHAPITRE II. LA COMMANDE MPPT FLOU

CHPITRE III FPGA ET LANGAGE VHDL

CHAPITRE IV DESCRIPTION ET SIMULATION

CONCLUSION GENERALE

BIBLIOGRAPHIE

ملـــخص

يهـدف مشروع نهـــاية دراســـة مهنـــدس هـــذا إلـــى تصميـــم نظـــام تحكـــم ألـــي لتتـــبع نقطـــة الطـــاقة الأعـــظمية للـــوح شمســـي .

تـــرتكز خـــوارزميـــة التحـــكم علـــى نظـــريـة المنـــطق الغـــامض ، التـــوجـه الـــذي أثبـت فـعاليـــته في مجـالات عـديدة و مختلـــفة .

لتـصميم نظـــامنا هـذا استـــعملنا ISE 7.1 لشـــركة Xilinx ، كمـــا استعمـانا أيضـــا ModelSim 5.7 لشـــركـة MentorGraphics لإخـتباره .

Résumé

Le but de ce travail est la conception d'un contrôleur pour le suivit du point de puissance maximale d'un panneau solaire et l'implémenter sur carte FPGA.

L'algorithme de la commande se base sur la théorie des ensembles flous, une approche qui a prouvé son efficacité dans plusieurs domaines.

L'outil de conception utilisé dans notre application est l'environnement de développement ISE 7.1 de Xilinx, il nous permet la description du système avec un langage connu pour simplicité et efficacité, il s'agit du langage VHDL.

Pour simuler et évaluer le comportement de notre système, nous avons utilisé le simulateur ModelSim 5.7 de Mentor Graphics.

Abstract

The goal of this work is the design of a controller for tracking the maximum power point of a solar panel and its implementation on FPGA chip.

The algorithm of the command is based on the theory of the fuzzy logic; its efficiency is proved in several fields.

To design and describe our system, we used the software environment developer ISE 7.1 of Xilinx, with a simple and efficient language, the VHD Language.

We used also the simulator ModelSim 5.7 of Mentor Graphics to simulate and evaluate the behaviour of our controller.

Remerciements

Je remercie en premier **M. C. LARBES** pour sa disponibilité, son aide et ses précieux conseils au cours de la préparation de ce projet de fin d'études.

Je remercie également **M. HADDADI** et **M. AIT CHEIKH** pour avoir accepter de juger mon modeste travail,

Je remercie le personnel de la bibliothèque et celui de l'école Nationale polytechnique en général,

J'adresse aussi mes remerciements à tous mes camarades de la promotion électronique 2006

Sans oublier **M. N.CHIKHI** et le **Centre de Développement des Technologies Avancées** pour leur aide et coopération.

Je remercie toute personne qui, d'une manière ou d'une autre a contribué à l'élaboration de ce modeste travail.

Introduction

INTRODUCTION

Depuis la généralisation de l'utilisation de l'électricité, la consommation énergétique n'a cessé d'augmenter ; des années 60 à nos jours celle ci a triplé. Actuellement, les principales sources d'énergie proviennent des combustibles fossiles (charbon, pétrole, gaz naturel) et du nucléaire. Cependant, depuis les crises successives du pétrole des années 70, le problème de la conversion et du stockage de l'énergie a conduit à la recherche et au développement de nouvelles sources d'approvisionnement. Cet intérêt s'est accru face à l'épuisement inéluctable des énergies fossiles, à leur impact sur l'environnement et aux déchets qu'elles engendrent.

Cependant et afin de décoloniser et de diversifier l'origine de la production d'électricité, de nouvelles énergies dites «renouvelables», ont émergé telles la biomasse, l'éolien, la géothermie, la marémotrice et le solaire. Elles ont toutes l'immense avantage d'être d'origine naturelle, inépuisables et non polluantes. Parmi celles-ci, l'énergie solaire est la mieux partagée par tous les habitants du globe terrestre. Son exploitation permettrait de fournir en électricité des sites isolés des réseaux électriques et éviterait la création de nouvelles lignes électriques qui demandent généralement un lourd investissement.

Les panneaux solaires, bien qu'ils soient de plus en plus performants, ont des rendements qui restent assez faibles (autour de 20%), c'est pourquoi il faut exploiter le maximum de puissance qu'ils peuvent générer en réduisant au maximum les pertes d'énergie.

Une caractéristique importante de ces panneaux est que la puissance maximale disponible est fournie seulement en un seul point de fonctionnement appelé MPP (Maximum Power Point), défini par une tension et un courant donnés, et ce point se déplace en fonction des conditions météorologiques (ensoleillement, température) ainsi que des variations de la charge. Extraire le maximum de puissance nécessite donc un mécanisme de poursuite de ce point qu'on appelle MPPT (MPP Tracker). Il existe plusieurs méthodes MPPT, nous nous intéressons à celle basée sur la logique floue.

D'autre part, les circuits FPGA (Field Programmable Gate Array), qui sont des circuits programmables adaptables à des besoins divers, deviennent incontournables dans les applications nécessitant un temps de développement rapide (time to market) et une modularité garantie. Ils sont surtout utilisés dans les systèmes embarqués (avionique, automobile, espace, …) et tendent à se généraliser dans le domaine des applications on chip.

L'objet de ce projet de fin d'études consiste en l'implémentation de l'algorithme de commande MPPT flou sur circuit FPGA. Ce travail nécessite plusieurs étapes, il commence

par la description en langage VHDL, la synthèse, le test et la simulation temporelle, et enfin le chargement du programme sur la carte FPGA.

Notre mémoire est organisé comme suit :

Dans le premier chapitre, nous introduisons les différents élément d'un système photovoltaïque : les générateurs photovoltaïques, le MPP, les batteries et les convertisseurs continu-continu.

Le deuxième chapitre introduit la théorie des ensembles flous, et explique l'algorithme de la commande MPPT flou.

Dans le troisième chapitre, nous nous intéresserons des circuits FPGA, leurs caractéristiques, les méthodologies de leurs conception, ainsi que le langage de programmation utilisé pour leur design et description qui est le VHDL.

Le dernier chapitre décrit la structure de notre contrôleur, illustre ses différents composants et simule son comportement pour, enfin, l'évaluer évaluer.

LISTE DES FIGURES

Systèmes Photovoltaïques

La Commande MPPT Floue

FPGA s & Langage VHDL

Description & Simulation

LISTE DES TABLEAUX

Chapitre

1

Systèmes Photovoltaïques

I. GENERATEURS PHOTOVOLTAÏQUES

L'électricité solaire est une importante source d'énergie renouvelable qui pourrait être une alternative aux autres sources classiques afin de satisfaire les larges besoins d'énergie dans le futur. Cette énergie trouve tout son avantage dans des applications de petite et moyenne consommation dans des régions isolées et loin des lignes de distribution électrique.

Une caractéristique importante des panneaux solaires est que la puissance maximale disponible est fournie seulement en un seul point de fonctionnement donné, localisé par une tension et un courant connu, appelé en anglais Maximum Power Point (MPP). Le problème est que la position de ce point n'est pas fixe mais se déplace en fonction de l'ensoleillement et de la température des cellules solaires ainsi que de la charge utilisée. A cause du coût relativement onéreux de ce genre d'énergie on doit extraire le maximum de puissance des panneaux solaires **[3].** Le but de ce chapitre est de montrer la manière de simuler le générateur photovoltaïque dans son ensemble.

I.1.1 CELLULE SOLAIRE

Une cellule solaire est un élément semi-conducteur qui convertit l'énergie solaire en une énergie électrique. Elle consiste en une base de silicium dopée de type P couverte d'une mince couche de type N, au-dessus de laquelle on dispose une grille métallique qui constitue avec la base arrière les électrodes de sortie de la cellule.

Lorsque la cellule est éclairée par une radiation lumineuse d'énergie $h\upsilon$ supérieure ou égale à l'énergie de la bande interdite E_g du semi-conducteur, un couple d'électron trou est créé, c'est l'effet photovoltaïque. Ces charges sont refoulées par le champ électrique interne de la jonction ***PN*** vers ses extrémités, ce qui va créer une différence de potentiel appelée tension photovoltaïque. Si une charge est appliquée aux bornes de la cellule, cette tension génère un courant I_{ph} appelé courant photovoltaïque.

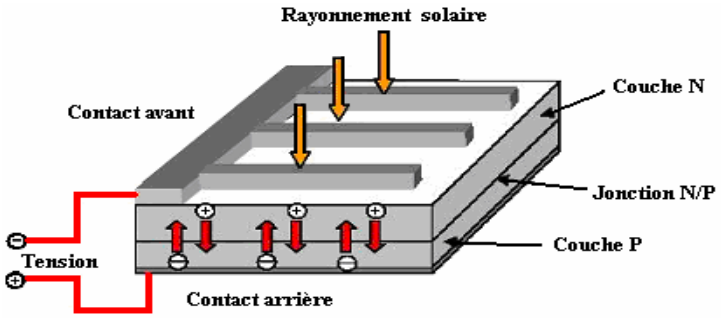


Fig.I.1 : la cellule photovoltaïque [4]

La courbe caractéristique d'une cellule PV (**Fig.I.2**), représente la variation du courant qu'elle produit en fonction de la tension à ces bornes depuis le court-circuit (tension nulle correspondant au courant maximum produit) jusqu'au circuit ouvert (courant nul pour une tension maximale aux bornes de la cellule).

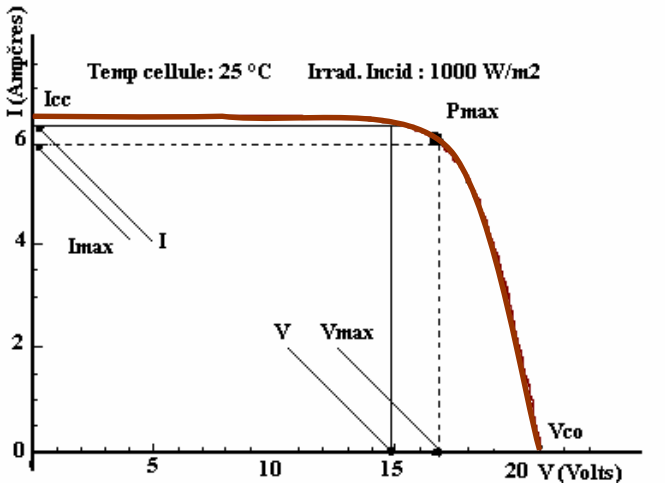


Fig.I.2 : Caractéristique I=f (V) d'une cellule photovoltaïque au silicium

Cette courbe est établie dans des conditions ambiantes de fonctionnement données (rayonnement donné, cellule PV à une température donnée, air ambiant circulant à une vitesse donnée). En effet, le fonctionnement des cellules photovoltaïques dépend des conditions d'ensoleillement et de température à la surface de la cellule. Ainsi, chaque courbe courant - tension correspond à des conditions spécifiques de fonctionnement. Si par exemple la température de la surface évolue, la courbe n'est plus la même.

Nous verrons par la suite l'influence de la température et de l'ensoleillement sur la caractéristique I(V) du module photovoltaïque.

Évidemment cette caractéristique dépend fortement de l'éclairement auquel est soumis le générateur et de sa température. On peut voir ci-dessous **Fig.I.3** et **Fig.I.4** l'allure générale des caractéristiques de la puissance en fonction de la tension et du courant en fonction de la tension pour différents éclairements et pour différentes températures.

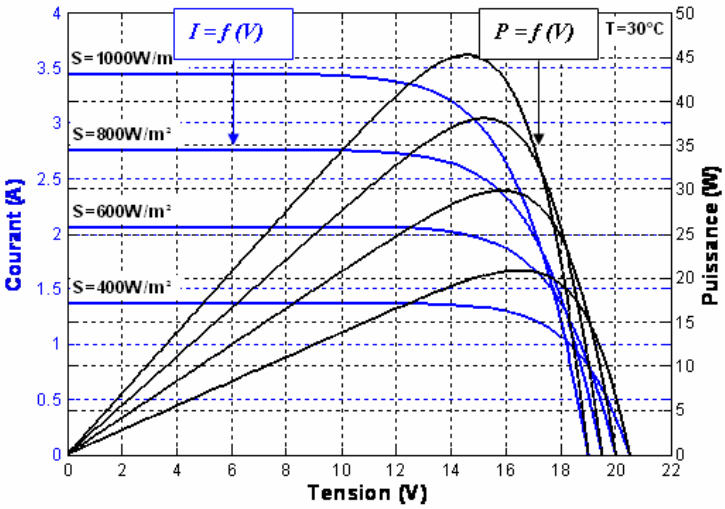


Fig.I.3 : Exemple de réseau de caractéristiques P/V-I/V d'un générateur photovoltaïque pour différents éclairement.

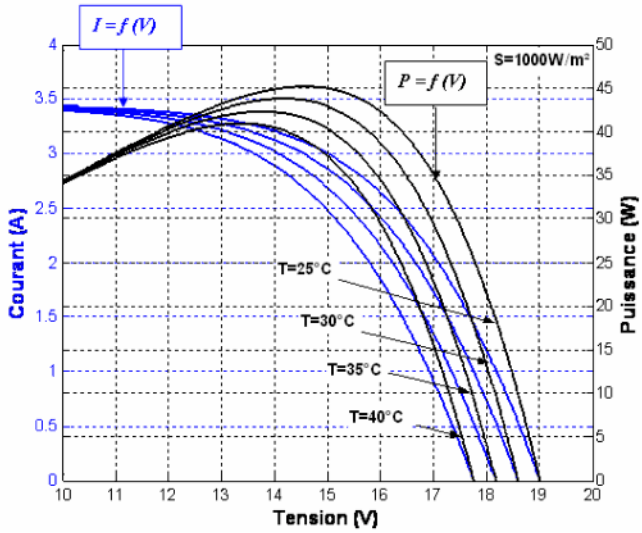


Fig.I.4 : Exemple de réseau de caractéristiques P/V-I/V d'un générateur photovoltaïque pour différentes températures

Sur chaque courbe des caractéristiques puissance / tension, le point représente le point de puissance maximale que peut fournir le panneau. On voit bien que pour un éclairement et une température donnés, il existe une valeur de la tension qui maximise la puissance produite par le générateur. Il peut donc être intéressant d'insérer un convertisseur de puissance entre le générateur photovoltaïque et sa charge pour assurer un fonctionnement à puissance maximale quelles que soient la charge et les conditions d'éclairement et de température, grâce à un convertisseur de caractéristiques I(v).

I.1.2 CIRCUIT EQUIVALENT ET MODELE MATHEMATIQUE

Pour la modélisation de la cellule solaire on a choisi le modèle fréquemment utilisé afin de décrire ses caractéristiques électriques. Ce modèle prend en compte les différentes résistances internes (**Fig.I.5**). **[8],[9]**.

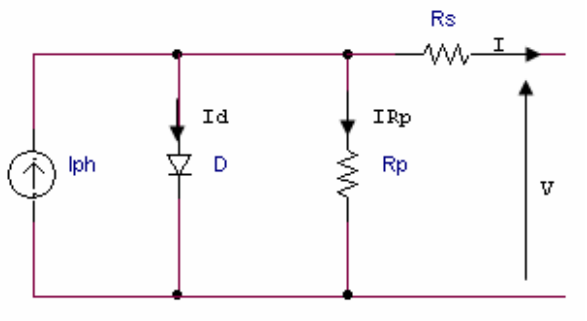


Fig.I.5 : Circuit équivalent d'une cellule solaire [2]

$\boldsymbol{R_p}$ représente la résistance shunt qui modélise les fuites par l'effet de bord autour de la cellule solaire ; $\boldsymbol{R_S}$ représente les contacts ohmiques entre le métal et le semi-conducteur ainsi que la résistance intrinsèque de silicium ; $\boldsymbol{I_{ph}}$ est le courant photovoltaïque engendré par les radiations lumineuses et ***D***, la diode, représente la jonction ***PN*** de la cellule.

A partir du modèle précédent on a :

$$I = I_{ph} - I_0\left(\exp\left(\frac{e(V+IRs)}{\alpha KT}\right)-1\right)-\frac{V+IRs}{Rp} \qquad \textbf{(1.1)}$$

α : Facteur d'idéalité.

T : la température de la cellule (K).

e *:* la charge élémentaire d'électron ($1.6021.10^{-19}$ C).

K : la constante de Boltzmann ($1.3854.10^{-23}$ JK^{-1}).

Une cellule solaire est caractérisée par les paramètres fondamentaux suivants.

- Courant de court-circuit $\boldsymbol{I_{sc}}$, c'est le courant maximal généré par la cellule ;

  $\boldsymbol{R_s << R_p}$ ce qui donne $\boldsymbol{I_{sc} \approx I_{ph}}$.

- Tension à circuit ouvert $\boldsymbol{V_o}$. Elle reflète la tension de seuil de la jonction ***PN***. En négligeant le courant qui passe dans la résistance parallèle $\boldsymbol{R_p}$, $\boldsymbol{V_{oc}}$ est exprimée comme suit :

$$V_{oc} = \frac{aKT}{e}\ln\left(\frac{I_{ph}}{I_0}\right) = V_t \ln\left(\frac{I_{ph}}{I_0}\right) \qquad \textbf{(1.2)}$$

Où $V_t = \frac{aKT}{e}$ est la tension thermique et $\boldsymbol{I_0}$ le courant de saturation de la diode ***D***.

- Point de puissance maximale, c'est le point de fonctionnement $\boldsymbol{(V_{mp}, I_{mp})}$ où la cellule solaire génère sa puissance maximale $P_{max} = V_{mp}.I_{mp}$.

- Facteur de remplissage ***FF*** (*Fill Factor* en anglais) ; il correspond au rapport de puissance maximale sur le produit de $\boldsymbol{V_{oc}}$ et I_{sc}, $FF = \frac{V_{mp}.I_{mp}}{V_{oc}.I_{sc}}$. Il reflète la qualité de la cellule par rapport à une cellule idéale (***FF*** = 1*)*.

• Le rendement η de la cellule ; c'est le rapport de conversion de l'énergie lumineuse en énergie électrique, qui est égal au rapport de la puissance maximale de sortie sur la puissance des radiations lumineuses.

$$\eta = \frac{P_{max}}{P_{in}} = \frac{V_{mp}.I_{mp}}{P_{in}} = \frac{V_{oc}.I_{sc}.FF}{P_{in}} \quad \textbf{(1.3)}$$

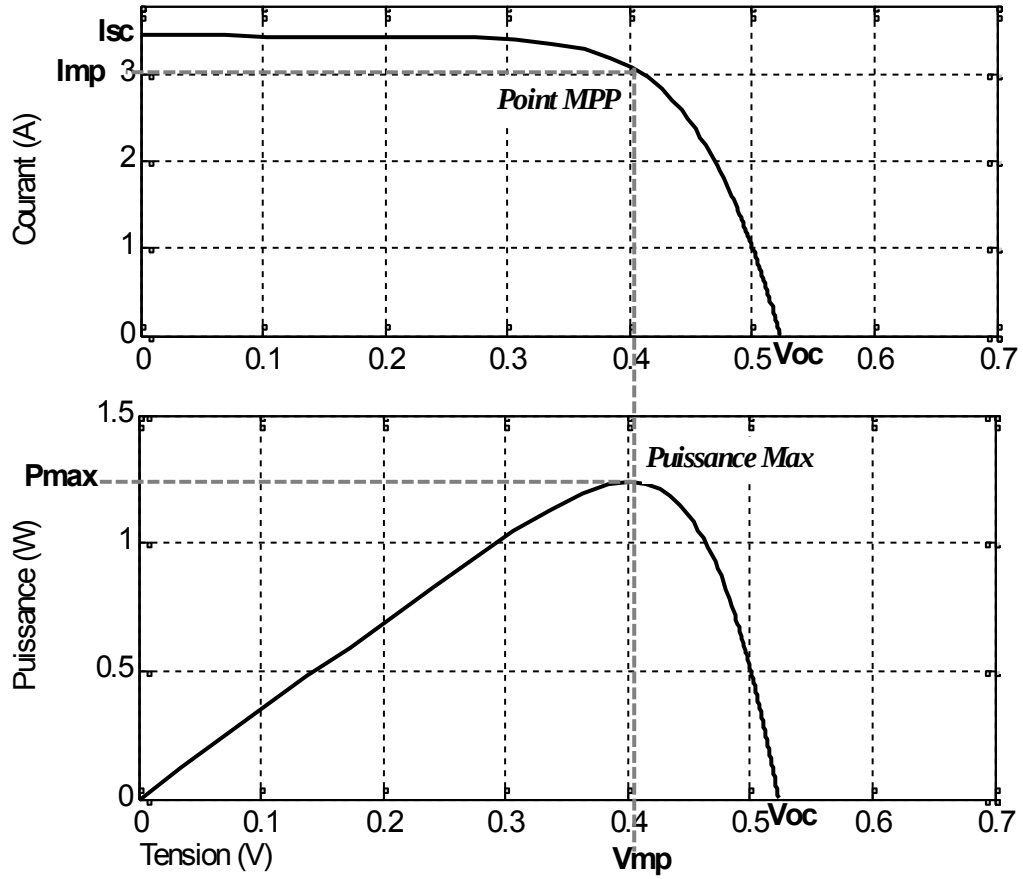


Fig.I.6 : Caractéristiques *I = f(V)* et *P = f(V)* d'une cellule solaire pour un ensoleillement et une température donnés

I.1.3 LES MODULES PHOTOVOLTAÏQUES

Une cellule photovoltaïque produit moins de 2 watts sous approximativement 0,5 Volt. L'assemblage de plusieurs cellules élémentaires forme des chaînes appelées Module PV. Le nombre de cellules connectées en parallèle et en série définit les caractéristiques du module PV.

L'association série et/ou parallèle de plusieurs modules permet de réaliser un panneau photovoltaïque. Pour n_p cellules en parallèles et de n_s cellules en séries, nous aurons la caractéristique I(V) (**Fig.I.7**). Celle-ci est homothétique à celle d'une cellule lorsque les diodes de protections n'interviennent pas et que toutes les cellules sont identiques et reçoivent le même éclairement.

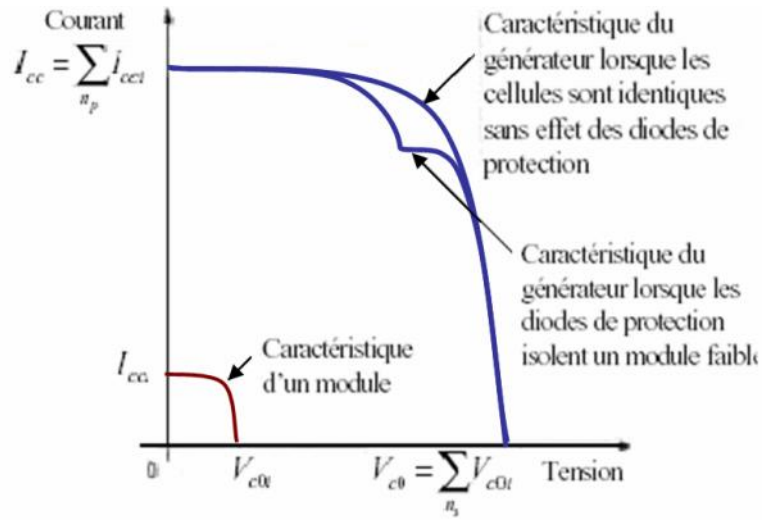


Fig.I.7 : Caractéristiques résultantes d'un générateur associant np cellules en parallèles et ns cellules en séries.

Le passage d'un module à un panneau se fait par l'ajout de diodes de protection, une en série pour éviter les courants inverses et une en parallèle, dite diode by-pass, qui n'intervient qu'en cas de déséquilibre d'un ensemble de cellules pour limiter la tension inverse aux bornes de cet ensemble et minimiser la perte de production associée.

Pour remédier aux problèmes de "Hot Spot" ou encore *point chaud*, surchauffe due à l'occultation d'une ou plusieurs cellules (ombrage ou défaillance), ces dernières deviennent des consommatrices de puissance et non des génératrices ce qui cause des pertes d'énergie, une diode by-pass, est placée dans la boîte de connexion. Elle fixée à l'arrière du panneau, (**Fig.I.8**) par le fabriquant toutes les 18, 20, 36,... n cellules (n dépendant du type de module.

Un module fonctionnant à une température ambiante élevée, par exemple dans un pays chaud, nécessiterai des diodes by-pass sur des séries courtes, 6 plutôt que 12 par exemple) **[21]**. Ces diodes éviteront que le courant passe à travers les cellules défaillantes lorsque leur tension tombe au-dessous de la tension de seuil de la diode.

Une diode série, diode de blocage, pour n modules (tous les 1, 2, 3, 4, ..., n modules) en parallèle, sera montée pour éviter le retour de courant des autres modules lorsqu'un ou plusieurs modules se retrouvent à l'ombre ou sont défaillants (**Fig.I.8**).

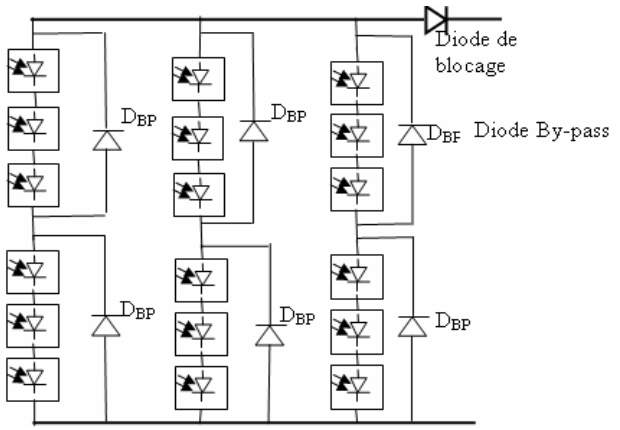


Fig.I.8 : Panneau solaire constitué de N_P branches parallèles avec *Ns* cellule séries

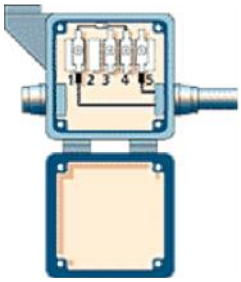


Fig.I.9: Boîte de connexion placée à l'arrière d'un module PV

Les chaînes de cellules sont encapsulées dans un plastique E.V.A. (Ethylène Vinyle Acétate) pour les protéger des agressions extérieures (rayons U.V., humidité) et les isoler électriquement (**Fig.I.10**). L'ensemble est protégé sur la surface avant par un verre trempé à haute transmission et ayant une bonne résistance mécanique, et sur la face arrière par une feuille de Tedlar ou de polyéthylène. Les différents composants sont soudés à haute température en un laminé protégé de la majorité des agressions extérieures. Le module est placé dans un cadre métallique rigide qui va lui donner une grande rigidité mécanique. Ce cadre est généralement pourvu de moyens de fixation (trous oblongs, barre profilée…).

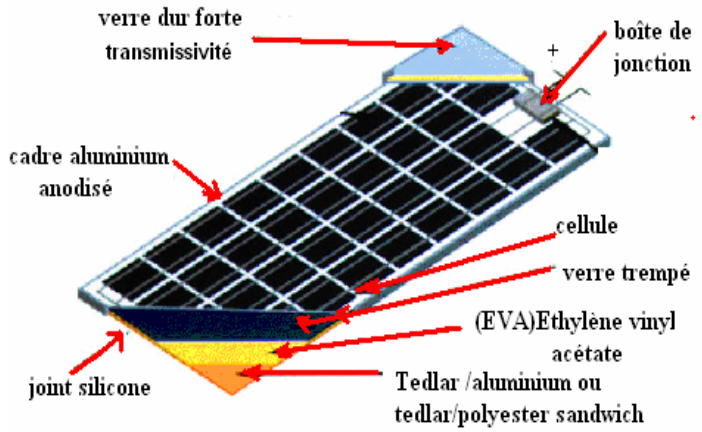


Fig.I.10: Constitution d'un module photovoltaïque [25]

Les modules photovoltaïques dépourvus de cadre et encapsulés dans une résine sont appelés « laminate ».Il subit enfin des tests mécaniques, optiques et électriques, avant d'être mis sur le marché

I.1.4 PERFORMANCES D'UN MODULE PHOTOVOLTAÏQUE

Le rendement des modules en fonctions des différentes technologies de fabrication fait apparaître des écarts importants comme la montre le tableau suivant:

Technologie	Rendement typique	Rendement maximum obtenu (laboratoire)
Monocristallin	12-15%	24%
Poly-cristallin	11-14%	18.6%
Couche mince : Amorphe	6-7%	12.7%

Tableau I.1 : Rendements des différentes technologies de modules

I.1.5 LE MODELE MATHEMATIQUE DU PANNEAU SOLAIRE

Pour éviter toute confusion entre les paramètres mathématiques de la cellule et du panneau solaire la notation suivante est utilisé : l'exposant ***'P'*** réfère aux paramètres du panneau et l'astérisque 'C'est utilisé pour la cellule. La tension globale du panneau est notée par $\boldsymbol{V^P}$ et le courant global est $\boldsymbol{I^P}$. Le modèle mathématique du panneau est obtenu en remplaçant chaque cellule par son circuit équivalent, privé de la résistance parallèle $\boldsymbol{R_p}$ car son influence sur les caractéristiques de la cellule est très faible **[11]**, **[1]**.

Le courant $\boldsymbol{I^P}$, en fonction des paramètres caractéristiques du panneau est donné par :

$$I^P = I_{sc}^P \left[1 - \exp\left(\frac{V^P - V_{OC}^P + R_S^P I^P}{N s V_t^C} \right) \right] \qquad \textbf{(1.4)}$$

Avec :

- $I_{sc}^P = N_P I_{sc}^C$ le courant de court-circuit du panneau
- $V_{oc}^P = N s V_{oc}^C$ la tension en circuit ouvert du panneau
- $R_S^P = R_S^C N_S$ la résistance série équivalente
- $V_t^C = \frac{aKT^C}{e}$ la tension thermique de la cellule

Les caractéristiques du panneau fournies par le constructeur sont de deux types :

1-Pour un ensoleillement et une température standards (Sa = 1000W/m^2, Ta = 25°C celui ci donne la puissance maximale $P^P_{\max,0}$, le courant de court circuit $I^P_{sc,0}$, et la tension en circuit ouvert $V^P_{oc,0}$.

2-Pour un ensoleillement et une température nominale (S_{ref} = 800W/m^2, T_{ref} = 20°C)

• La température de la cellule T^C_{ref}, le nombre de branches parallèles N_P et le nombre de cellules séries *Ns.*

A partir de ces données on peut calculer les caractéristiques de la cellule pour des conditions standards : $P^C_{\max,0}, V^C_{OC,0}, I^C_{sc,0}, R^C_S$. Les paramètres opérationnels suivants de la cellule sont alors calculés:

• Le courant de court circuit, à partir de la relation linéaire avec l'ensoleillement S_a :

$$I^C_{sc}=C_1 S_a \qquad \textbf{(1.5)}$$

• La température, liée à la température ambiante et l'ensoleillement

$$T^C=T_a+C_2 S_a \qquad \textbf{(1.6)}$$

Avec : $C_2=\dfrac{T^C_{ref}-T_{a,ref}}{S_{a,ref}}$.

• Lorsque T^C_{ref} est inconnue on peut faire une approximation C_2 = 0.03°Cm^2/W.

• La tension en circuit ouvert, donnée par la relation :

$$V^C_{oc}=V^C_{oc,0}+C_3(T^C-T^C_a) \qquad \textbf{(1.7)}$$

Avec C_3 = -2.3mv/°C.

Ainsi on obtient la formule générale du courant d'un panneau solaire constitué de N_P branches, chacun ayant *Ns* cellules en série :

$$I^p = N_p I^c_{sc} [1 - \exp(V^p - N_s V^c_{oc} + I^p R^c_s N_s / N_p) /(N_s V^c_t))] \qquad \textbf{(1.8)}$$

I.2. STOCKAGE DE L'ENERGIE

A cause de la non disponibilité permanente de l'énergie solaire, pour diverses raisons : conditions météorologiques, horaire de la journée, saisons etc…, l'utilisation des batteries pour le stockage d'énergie est nécessaire pour garantir une énergie constante quelles que soient les conditions atmosphériques ou l'horaire du moment.

La batterie est un ensemble de cellules électrochimiques capables de stocker de l'énergie électrique sous forme chimique (pendant la charge), puis de la restituer par la suite (pendant la décharge) grâce à la réversibilité des réactions mises en jeu. Ces réactions

consistent en des oxydations et des réductions (oxydoréductions en abrégé, soit perte ou gain d'un ou plusieurs électrons) au niveau des électrodes. Le courant circule sous forme d'ions dans l'électrolyte et sous forme d'électrons dans le circuit raccordé à la batterie **(Fig.I.11).**

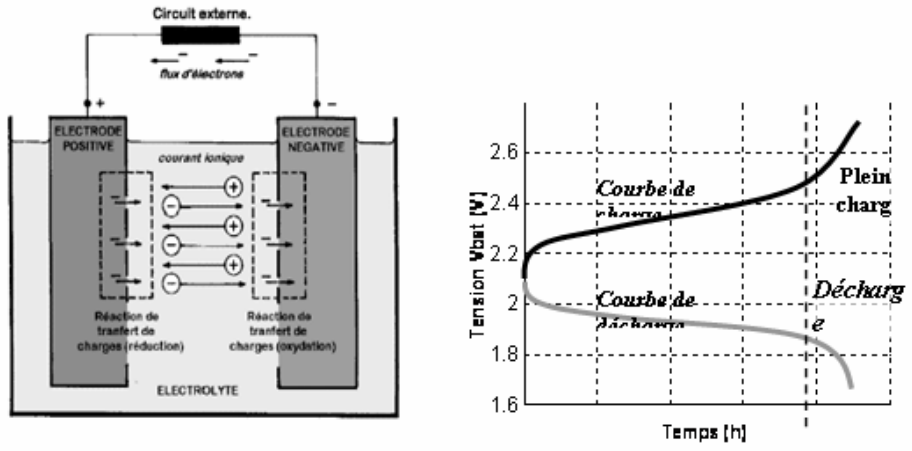


Fig.I.11 : Cellules électrochimiques de base et allure des courbes de charge et de décharge d'une cellule au plomb.

I.2.1 Modélisation mathématique de la batterie

Cette modélisation est basée sur le fait que la batterie est considérée comme un générateur de tension ***E*** variable en série avec une résistance R_0 (**Fig.I.12**).

La tension aux bornes de la batterie V_{bat} est donnée par :

$$V_{bat} = E + I_{bat} R_O \quad \textbf{(1.9)}$$

Avec :

E : tension du générateur interne, variable en fonction de l'état de charge de la batterie.

I_{bat} : Courant de batterie, positif pendant la charge et négatif pendant la décharge.

R_0 : Résistance ohmique interne de la batterie.

$$R_0 = \frac{\Delta V_{bat}}{\Delta I_{bat}} \quad \textbf{(1.10)}$$

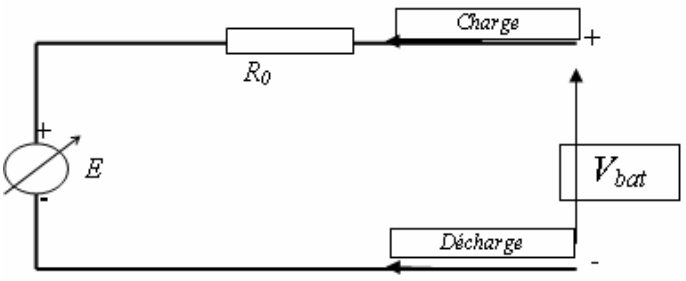


Fig.I.12 : Schéma électrique équivalent d'une batterie

Où ΔV_{bat} et ΔI_{bat} représentent une faible variation de la tension et du courant de la batterie dans la zone linéaire de charge ou de décharge dans la zone où la tension ***E*** est pratiquement constante.

La tension interne *E* est en fonction de l'état de charge de la batterie **[26]** :

$$E = E_0 + AX - \frac{BX}{C_1 - X} + \frac{B(C_2 - X)}{X} \quad \textbf{(1.11)}$$

Avec :

E_0 : La tension en circuit ouvert lorsque la batterie est complètement chargée.

A : Représente la variation linéaire de la tension de la batterie pendant la charge ou la décharge.

B, C_1 , C_2 : Ces paramètres reflètent la chute ou l'augmentation brutale de la tension pendant une décharge complète ou pendant une surcharge respectivement.

Le terme $\frac{BX}{C_1 - X}$ reflète le comportement de la batterie pendant la décharge complète alors que le terme $\frac{B(C_2 - X)}{X}$ reflète son comportement pendant une surcharge.

X : La capacité normalisée extraite/ajoutée pour un courant de décharge/charge donné, est égale :

$$X = \frac{Q_{out}}{Q_{\max}(I)} Q_{\max} \quad \textbf{(1.12)}$$

Avec :

$Q_{max}(I)$: La capacité maximale pendant la charge/décharge en fonction du courant de charge ou de décharge respectivement, Exemple pour une batterie avec $\boldsymbol{Q_{max}}$ = 45Ah et un courant de décharge de 1A, cette capacité chute de moitié, soit 22.5Ah pour un courant de décharge de 2A.

Q_{max} : La capacité maximale de la batterie en Ah.

Q_{out} : La somme des charges de la batterie extraites ou ajoutées pendant la décharge ou la charge. Elle est égale à :

$$Q_{out} = \int I_{bat} dt$$

On définit aussi l'état de charge *SOC* (*State of Charge*) de la batterie comme étant **[18]** :

$$SOC = \frac{Q_{\max} - Q_{out}}{Q_{\max}} = 1 - \frac{Q_{out}}{Q_{\max}} \quad \textbf{(1.13)}$$

Elle varie entre ***un*** (charge complète) et ***zéro*** (décharge complète).

I.3. CONVERSION DE L'ENERGIE

Les convertisseurs continu-continu ont pour fonction de fournir une tension continue variable à partir d'une tension continue fixe. La tension continue de départ peut être un réseau alternatif redressé et filtré, une batterie d'accumulateurs, une alimentation stabilisée…

Nous allons nous intéresser, dans un premier temps aux structures les plus simples des hacheurs. Il s'agit de celles qui n'assurent pas la réversibilité, ni en tension, ni en courant. L'énergie ne peut donc aller que de la source vers la charge.

I.3.1. HACHEUR DEVOLTEUR (OU SERIE).

Ce nom est lié au fait que la tension moyenne de sortie est inférieure à celle de l'entrée. Il comporte un interrupteur à amorçage et à blocage commandés (transistor bipolaire, transistor MOS ou IGBT…) et un interrupteur à blocage et amorçage spontanés (diode).

I.3.1.1 Schéma de principe.

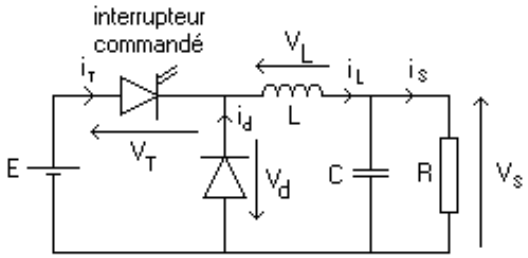


Fig.II.13 : Schéma de principe d'un hacheur dévolteur

La charge est constituée par la résistance R. Les éléments L et C forment un filtre dont le but est de limiter l'ondulation résultant du découpage sur la tension et le courant de sortie. Si ces éléments sont correctement calculés, on peut supposer que i_s et v_s sont continus (on néglige l'ondulation résiduelle).L'ensemble (filtre + charge) peut être composé différemment, mais nous raisonnerons sur cet exemple par la suite.

I.3.1.2. Fonctionnement.

Le cycle de fonctionnement, de période de hachage T (T=1/f), comporte deux étapes. Lors de la première, on rend le transistor passant et la diode, polarisée en inverse, est bloquée. Cette phase dure de 0 à α.T, avec α compris entre 0 et 1. α est appelé rapport cyclique.

Lors de la seconde, on bloque le transistor. La diode devient passante. Cette phase dure de α .T à T.

I.3.1.3. Formes d'ondes.

Nous allons être amenés à distinguer deux cas : la *conduction continue* et la *conduction discontinue.*

• Dans le premier, le courant de sortie est suffisamment fort et le courant dans l'inductance ne s'annule jamais, même avec l'ondulation due au découpage.

Dans le second, le courant de sortie moyen est bien entendu positif, mais, en raison de sa faible valeur moyenne, l'ondulation du courant dans l'inductance peut amener ce

dernier à s'annuler. Or, les interrupteurs étant unidirectionnels, le courant ne peut changer de signe et reste à 0.

Les formes d'ondes données maintenant supposent que les composants sont tous parfaits et que tension et courant de sortie, v_s et i_s, peuvent être assimilés à leur valeur moyenne (ondulations de sortie négligées).

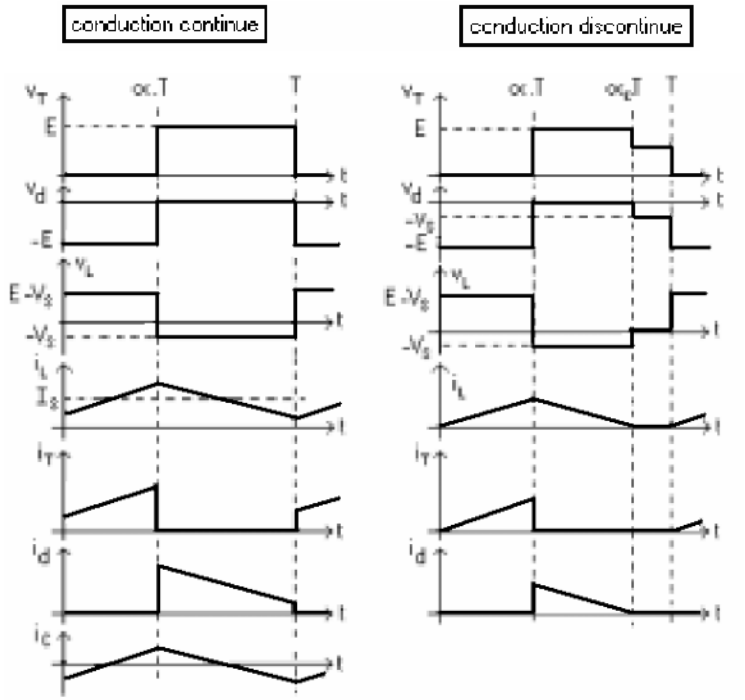


Fig.II.14 : formes d'ondes

I.3.1.4. Tension moyenne et ondulation de tension et de courant.

Nous allons désormais représenter les grandeurs par des lettres minuscules, leurs valeurs moyennes par des lettres majuscules et l'ondulation par une minuscule surmontée de

Pour une grandeur a(t) quelconque, on aura donc

$$a = A + \tilde{a}$$

valeur moyenne de la tension de sortie.

$v_s = -v_L - v_d$ Soit $\boxed{V_s = -V_d}$ car la tension moyenne aux bornes d'une inductance, en régime périodique, est nulle.

En conduction *continue*, on a $\boxed{V_s = \alpha.E}$ alors qu'en *conduction discontinue* $\boxed{V_s = \frac{\alpha}{\alpha_E}.E}$

- remarque concernant i_L.

La pente **de i_L** est $(E-V_s)/L$ de 0 à $\alpha .T$ et $(-V_s)/L$ de $\alpha .T$ à $\alpha_E.T$ (on suppose pour cela que l'ondulation de tension de sortie est négligeable) et dans le cas de la conduction continue,

α_E=1.

En effet, on a $v_L = L.\frac{di_L}{dt}$ avec v_L=E-V_s de 0 à α .T et v_L=-V_s de α .T à α_E .T.

• Calcul de l'ondulation de courant dans l'inductance (nous raisonnerons en conduction continue et nous supposerons l'ondulation de tension négligeable en sortie). Crête à crête, on obtient, compte tenu des calculs précédent

$$\Delta i_L = \frac{E-\alpha.E}{L}.\alpha.T = \frac{\alpha.(1-\alpha)}{L.f}.E \qquad (1.14)$$

On constate que l'ondulation de courant sera d'autant plus faible que l'inductance sera importante (cette inductance est appelée inductance de lissage). De plus, en augmentant la fréquence de découpage, on diminuera encore l'ondulation. Il faut cependant garder à l'esprit que les pertes par commutation dans l'interrupteur augmentent avec la fréquence (penser à adapter le radiateur à la fréquence de hachage…).

• Calcul de l'ondulation de tension de sortie (en conduction continue). Cette fois, on ne néglige plus ce phénomène. On a

$$i_c = C.\frac{dv_c}{dt} \qquad \text{et} \qquad i_c = \tilde{i}_L$$

L'ondulation crête à crête sera prise entre deux instants successifs où i_c s'annule, par exemple entre (α /2).T et α .T puis entre α .T et (α +1).T/2 puisque deux zones de fonctionnement sont à considérer. Globalement, on a donc

$$\Delta v_c = \Delta v_{c1} + \Delta v_{c2} = \frac{1}{C}\left[\int_{\frac{\alpha.T}{2}}^{\alpha.T} \tilde{i}_L.dt + \int_{\alpha.T}^{\frac{(\alpha+1).T}{2}} \tilde{i}_L.dt\right] = \frac{1}{C}.\left[\left(\frac{1}{2}.\frac{\Delta i_L}{2}.\frac{\alpha.T}{2}\right)+\left(\frac{1}{2}.\frac{\Delta i_L}{2}.\frac{(1-\alpha).T}{2}\right)\right]$$

Soit $$\Delta v_c = \frac{\Delta i_L}{8.C.f} = \frac{\alpha.(1-\alpha).E}{8.L.C.f^2} \qquad (1.15)$$

On constate donc que l'ondulation décroît plus rapidement avec la fréquence que l'ondulation de courant. De plus, cette ondulation sera d'autant plus faible qu'inductance et capacité seront élevées.

rq: les évolutions de v_c sont des portions de paraboles si le courant i_c est supposé triangulaire.

rq: on ne raisonne pas en conduction discontinue car l'ondulation sera alors moins élevée. Ce régime n'est, de toute façon, pas très intéressant pratiquement.

I.3.1.5. Caractéristique statique Vs(Is).

En conduction continue, $V_s = \alpha.E$ est indépendant de I_s. En revanche, en conduction discontinue, on a $V_s = (\alpha / \alpha_E).E$ avec α_E qui dépend de I_s. Pour trouver la relation souhaitée, on suppose que le convertisseur est parfait ce qui nous donne

$$V_s.I_s = E.I_T \tag{1.16}$$

or, on a, à la limite de la conduction discontinue

et $$I_T = \frac{i_{L\max}}{2}.\alpha \qquad i_{L\max} = \frac{E - V_s}{L}.\alpha.T$$

donc $$V_s.I_s = E.\frac{E - V_s}{2.L.f}.\alpha^2$$

soit

$$I_s = \frac{\alpha^2}{2.L.f}.\frac{E.(E - V_s)}{V_s}$$

La courbe séparant la zone de conduction continue de la zone de conduction discontinue est obtenue en associant l'équation précédente et $V_s = \alpha .E$, ce qui conduit à l'équation de parabole suivante

$$\boxed{I_s = \frac{1}{2.L.f}.\frac{V_s.(E - V_s)}{E}} \tag{1.17}$$

Cette courbe est appelée courbe de conduction critique.

Graphiquement, la caractéristique $V_s(I_s)$, paramétrée par α, pour une fréquence fixée, se présente sous la forme suivante

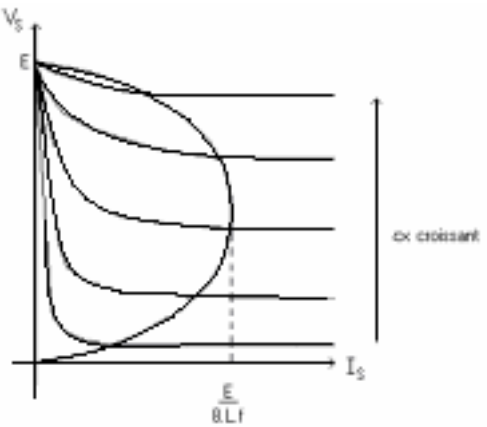


Fig.II.15 : La caractéristique V_S (I_S)

rq : dans la zone de conduction continue, on obtient une source de tension parfaite, dont la valeur est commandée par α.

rq: en fait, les évolutions de V_s ne sont pas rigoureusement horizontales, mais légèrement décroissantes, en raison des pertes ohmiques dans le montage, et

notamment dans les interrupteurs.

I.3.2 HACHEUR SURVOLTEUR (OU PARALLELE)

Dans ce hacheur, la tension moyenne de sortie est supérieure à la tension d'entrée, d'où son nom. Cette structure demande un interrupteur commandé à l'amorçage et au blocage (bipolaire, MOS, IGBT…) et une diode (amorçage et blocage spontanés).

I.3.2.1. Schéma de principe.

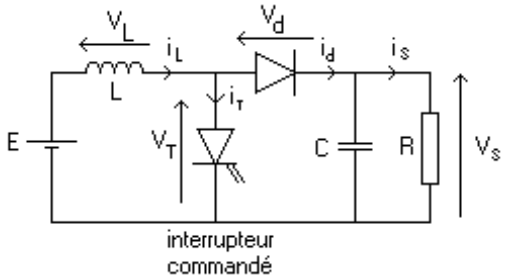


Fig.II.16 : Schéma de principe d'un hacheur survolteur

L'inductance permet de lisser le courant appelé sur la source. La capacité C permet de limiter l'ondulation de tension en sortie.

I.3.2.2. Fonctionnement.

Lors de la première partie du cycle de fonctionnement, de 0 à α .T, l'interrupteur commandé est fermé (passant). Cette fois, la source et la charge ne sont pas en contact durant cette phase. La diode est alors bloquée.

Lors de la seconde partie du cycle, de α .T à T, on ouvre l'interrupteur commandé et la diode devient passante. C'est alors que la source et la charge sont reliées.

I.3.2.3. Formes d'ondes.

Les formes d'ondes sont de la forme suivante (en supposant la tension et le courant de sortie continus).

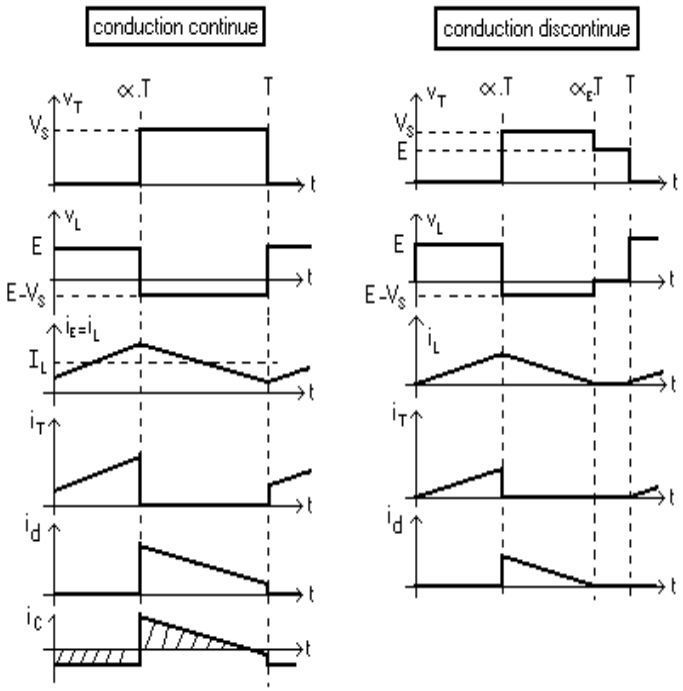


Fig.II.17 : formes d'ondes

I.3.2.4. Calcul de la tension moyenne de sortie et des ondulations.

- valeur moyenne de la tension de sortie.

On sait que la tension moyenne aux bornes de l'inductance est nulle donc on a, en conduction continue

$$E.\alpha.T = (-E + V_s).(1-\alpha).T \qquad (1.18)$$

soit

$$V_s = \frac{E}{1-\alpha}$$

Or comme α est inférieur à 1, la tension moyenne de sortie est bien supérieure à la tension d'entrée.

- Relation entre le courant moyen de sortie et le courant moyen dans la diode.

$I_s = I_d$ car le courant moyen dans la capacité est nul.

Donc $I_s = (1 - \alpha).I_L$. De plus, connaissant la tension moyenne de sortie et la résistance de charge, on en déduit facilement I_s ce qui permet de connaître I_L.

- Calcul de l'ondulation de courant crête à crête dans l'inductance L.

$$\Delta i_L = \frac{\alpha.E}{L.f} \qquad (1.19)$$

I.3.3 HACHEUR A STOCKAGE INDUCTIF.

I.3.3.1. Structure.

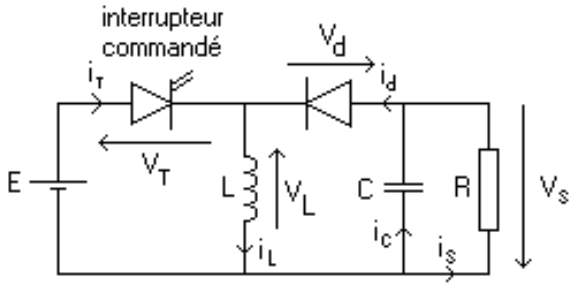


Fig.II.18 : structure d'un hacheur à stockage inductif

I.3.3.2. Fonctionnement.

Lors de la première partie du cycle de fonctionnement, de 0 à α .T, l'interrupteur commandé est fermé (passant). La diode est ouverte et l'inductance stocke l'énergie fournie par le générateur d'entrée.

Lors de la seconde partie du cycle, de α .T à T, on ouvre l'interrupteur commandé et la diode devient passante. L'inductance restitue son énergie à la charge.

rq: On note que le sens de la tension de sortie est inversé par rapport aux deux cas précédents.

I.3.3.3. Formes d'ondes.

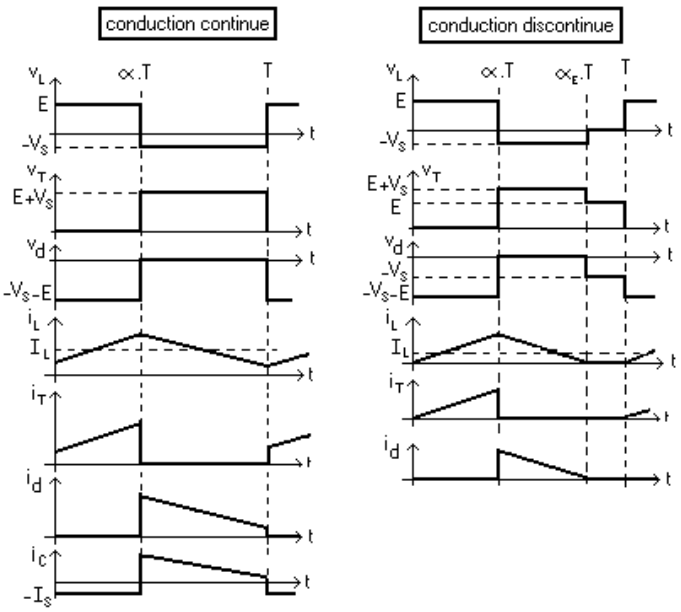


Fig.II.19 : formes d'ondes

I.3.3.4. Calcul de quelques grandeurs.

En régime de conduction continue on peut calculer les relations suivantes.

- valeur moyenne de la tension de sortie.

On sait que la tension moyenne aux bornes de l'inductance est nulle donc on a, en conduction continue

$$E.\alpha.T = V_s.(1-\alpha).T \qquad (1.20)$$

soit

$$V_s = \frac{\alpha.E}{1-\alpha}$$

Suivant la valeur de α, la tension moyenne de sortie peut être supérieure ou inférieure à la tension d'entrée, d'où le nom de hacheur survolteur-dévolteur parfois donné à ce montage.

- Calcul de l'ondulation de courant crête à crête dans l'inductance L.

$$\Delta i_L = \frac{\alpha.E}{L.f} \qquad (1.21)$$

Relation entre le courant moyen de sortie et le courant moyen dans l'inductance.

$I_s = I_d$ car le courant moyen dans la capacité est nul.

Donc $I_s = (1-\alpha).I_L$. De plus, connaissant la tension moyenne de sortie et la résistance de charge, on en déduit facilement I_s ce qui permet de connaître I_L.

rq: on sait que v_s décroissant de 0 à α .T si i_c est alors négatif. On calcule la valeur de i_c sur cet intervalle (ce courant vaut $-I_s$) et on calcule l'intégrale, comme pour le hacheur parallèle.

rq: en fait, rien n'indique que i_c n'est négatif que sur l'intervalle de temps [0, α .T], mais il y aura d'autant plus de chance que ce soit le cas si l'ondulation de courant est faible, donc si l'inductance est forte.

rq : prise en compte de la résistance de la bobine de stockage d'énergie (on remplace L par

L en série avec r) dans le cas de la conduction

continue. Cette fois, on peut écrire que

$$\langle i_T \rangle = \alpha.\langle i_L \rangle$$

$$\langle i_s \rangle = (1-\alpha).\langle i_L \rangle = \frac{V_s}{R}$$

$$r.\langle i_L \rangle = \alpha.E - (1-\alpha).V_s$$

On en déduit que

$$V_s = \frac{\frac{R}{r}.\alpha.(1-\alpha)}{1+\frac{R}{r}.(1-\alpha)^2}.E \qquad (1.22)$$

Cette relation traduit un comportement très différent du cas du système parfait. Quand α tend vers 1, la tension V_s tend vers 0 au lieu de tendre vers l'infini.

On trouve un rendement **η** inférieur à 1 pour α = 0 et tendant vers 0 quand α tend vers 1

et dont la formule est donnée par

$$\eta = \frac{\frac{V_s^2}{R}}{E.\langle i_T \rangle} = \frac{\frac{R}{r}.(1-\alpha)^2}{1+\frac{R}{r}.(1-\alpha)^2} \qquad (1.22)$$

CONCLUSION

La technologie photovoltaïque présente un grand nombre d'avantages.

- D'abord, une haute fiabilité elle ne comporte pas de pièces mobiles qui la rendent particulièrement appropriée aux régions isolées. C'est la raison de son utilisation sur les engins spatiaux.
- Ensuite, le caractère modulaire des panneaux photovoltaïques permet un montage simple et adaptable à des besoins énergétiques divers. Les systèmes peuvent être dimensionnés pour des applications de puissance allant du Milliwatt au Mégawatt.
- Leurs coûts de fonctionnement sont très faibles.
- Enfin, la technologie photovoltaïque présente des qualités sur le plan écologique car le produit fini est non polluant, silencieux et n'entraîne aucune perturbation du milieu.

La puissance de sortie ne dépend pas seulement de l'ensoleillement et de la température mais aussi de la tension de fonctionnement *V* du module. La puissance de sortie est maximale pour une certaine tension *V*. C'est en ce point de fonctionnement qu'on doit faire fonctionner le module pour qu'il travaille avec un rendement maximal. Ce point est appelé *MPP* (Maximum Power Point).

- Enfin, le stockage de l'énergie électrique sous forme chimique (batterie) est nécessaire.
- la fiabilité et les performances du système dépendent de la batterie, le convertisseur et le régulateur associés.

Chapitre

2

La Commande MPPT Floue

II.1. INTRODUCTION A LA LOGIQUE FLOUE

Depuis quelques années déjà, on trouve sur le marché des appareils de grande consommation (appareils de photos, vidéo, ...) qui sont présentés comme faisant intervenir un réglage par logique floue ou « fuzzy-logic » ou encore « fuzzy-control ». Au delà de l'argument publicitaire évident, il est intéressant de comprendre ce nouveau concept de réglage et de l'appliquer à certains types de problèmes de réglage rencontrés par l'ingénieur.

Le principe du réglage par logique floue part du constat suivant: dans les problèmes de régulation auxquels il est confronté, l'homme ne suit pas, à l'image de ses inventions, un modèle mathématique fait de valeurs numériques et d'équations. Au contraire il utilise des termes tel que « un peu trop chaud, aller beaucoup plus vite, freiner à fond, etc... » ainsi que ses propres connaissances qu'il a dans le domaine. Ces connaissances sont, le plus souvent, acquises de façon empirique. Le principe du réglage par la logique floue s'approche de la démarche humaine dans le sens que les variables traitées ne sont pas des variables logiques (au sens de la logique binaire par exemple) mais des variables linguistiques, proches du langage humain de tous les jours. De plus, ces variables linguistiques sont traitées à l'aide de règles qui font référence à une certaine connaissance du comportement du système à régler. Sur la base de ce principe, différentes réalisations ont vu le jour et, actuellement, on trouve deux types d'approche pour le réglage par logique floue. Dans l'une de ces approches, les règles sont appliquées aux variables à l'aide d'une approche numérique par le biais d'un microprocesseur spécialisé ou non ou d'un ordinateur. Dans l'autre approche, les règles sont appliquées aux variables de façon analogique. Ces deux approches permettent de développer des organes de contrôle par le flou que l'on désigne par processeur digital flou ou par processeur analogique flou.

II.1.1 BREF HISTORIQUE

Les quelques points de repères historiques suivants permettent de situer dans le temps le développement de la logique floue et ses applications au réglage:

1965 Le Prof. L. A. Zadeh de l'Université de Berkeley (Californie) pose les bases théoriques de la logique floue.

1973 L. A. Zadeh propose d'appliquer la logique floue aux problèmes de réglage.

1974 Première application du réglage par la logique floue appliquée à une turbine à vapeur. Suivie en 1980 par une application sur un four à ciment et en 1983 sur un épurateur d'eau.

1985 Premiers produits industriels (Japon) utilisant le principe de la logique floue appliqué à des problèmes de réglage et de commande.

Développement de processeurs dédiés à des applications de réglage par la logique floue.

II.1.2 PRINCIPES DE LA LOGIQUE FLOUE

La logique floue est une branche des mathématiques et, à ce titre, toute une série de notions fondamentales sont développées. Ces notions permettent de justifier et de démontrer certains principes de base. Dans ce qui suit, on ne retiendra que les éléments indispensables à la compréhension du principe du réglage par la logique floue. Ces éléments sont :

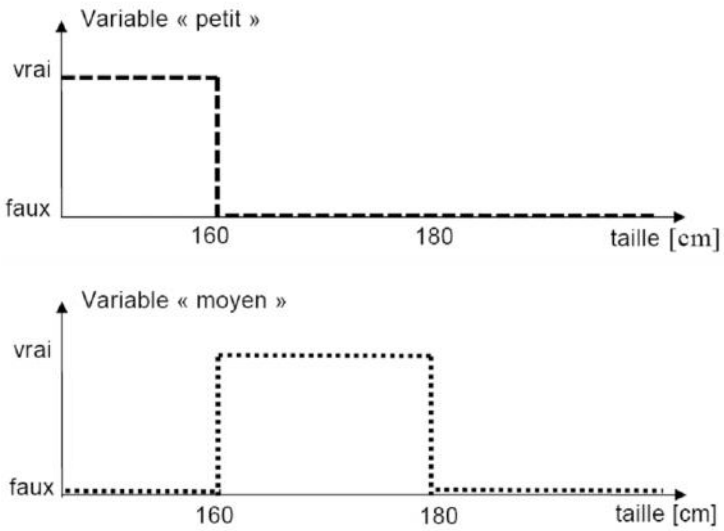
- les variables floues
- les règles d'inférences

II.1.2.1. Variables floues

Contrairement aux variables binaires qui sont définies par les deux états « vrai » ou « faux », les variables floues présentent toute une gradation entre la valeur « vrai » et la valeur « faux ».

L'exemple qui suit permet de mieux saisir la distinction qui existe entre les variables binaires et les variables floues :

Si l'on désire classer un groupe d'individu par leur taille en définissant la catégorie des petits par une taille en dessous de 160 cm, la catégorie des moyens par une taille comprise entre 160 cm et 180 cm et la catégorie des grand par une taille supérieure à 180 cm, la logique binaire donne la représentation de la **Fig.II.1** pour les trois variables « petit », « moyen » et « grand »



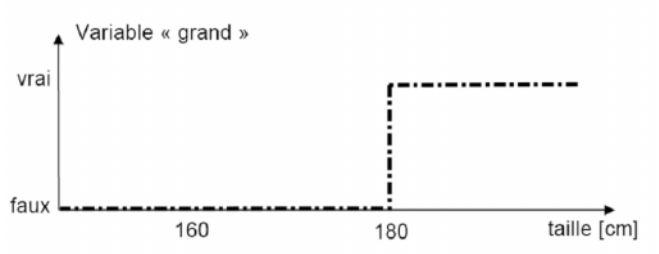


Fig.II.1 : la logique binaire

Deux remarques s'imposent au sujet de cette représentation :

• D'une part, on préfère représenter l'état de la variable à l'aide de son degré de vérité en associant la valeur 1 (degré de vérité de 100%) à la valeur « vrai » et le degré de vérité nul à la valeur « faux ».

• D'autre part, on constate que cette façon de faire est très éloignée de ce que fait l'être humain lorsqu'il résout ce genre de problème. En effet, l'homme ne fait pas naturellement une distinction franche entre « petit » et « moyen » par exemple. Il utilise des expressions du genre « plutôt petit » pour qualifier une taille légèrement inférieure à 160 cm et « plutôt moyen » pour une taille légèrement supérieure à cette valeur.

En conclusion, la logique binaire présente l'avantage de la simplicité mais est assez éloignée de la logique utilisée naturellement par l'être humain.

Si l'on représente le même problème à l'aide de la logique floue, les variables ne sont plus binaires mais présentent une infinité de valeurs possible entre le « vrai » et le « faux » (**Fig.II.**2).

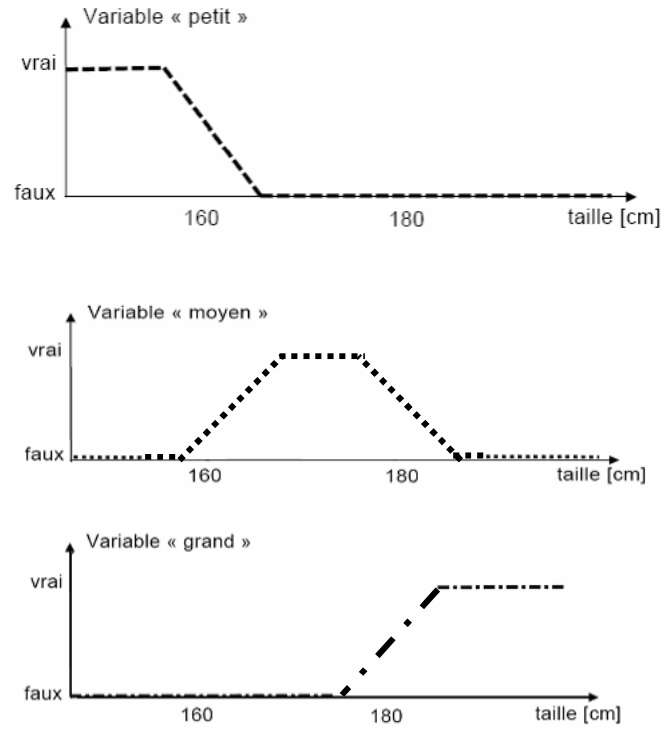


Fig.II.2 : la logique floue

On constate que cette représentation est beaucoup plus proche de la façon dont l'être humain raisonne puisqu'elle permet de faire intervenir des notions telles que « plutôt petit », « assez grand »... Cet avantage se fait, évidemment, au détriment de la simplicité de la représentation.

II.1.2.2 Règles d'inférence

On appelle règles d'inférence l'ensemble des différentes règles reliant les variables floues d'entrée d'un système aux variables floues de sortie de ce système. Ces règles se présentent sous la forme :

Si condition 1 et/ou condition 2 (et/ou...) alors action sur les sorties.

L'exemple suivant, tiré de la vie quotidienne, permet d'illustrer ceci :

Lorsque l'on prend une douche, un des problèmes qui se présente est de régler la température de l'eau. La variable d'entrée du système homme-douche est la température de l'eau mesurée à l'aide de nos capteurs de température. Les variables de sorties sont les deux robinets eau chaude et eau froide. Dans la pratique, le réglage de la température se fait en utilisant notre expérience, expérience qui recouvre à la fois nos préférences et notre connaissance de l'installation sanitaire utilisée. Ce réglage se fait en utilisant des règles du genre :

Si la température est très froide alors ouvrir à fond l'eau chaude

Si la température est un peu trop froide alors fermer un peu l'eau froide

Si la température est bonne alors laisser les deux robinets dans leur état

Si la température est trop chaude alors fermer un peu l'eau chaude et ouvrir un peu l'eau froide. etc.…

En termes d'intelligence artificielle, ces règles résument en fait l'expérience de l'expert et elles ne sont en général pas définissables de façon unique puisque chaque individu crée ses propres règles.

II.1.3 VARIABLES FLOUES

Les grandeurs utilisées dans un système de réglages sont généralement générées par des capteurs. Il est nécessaire de convertir ces grandeurs en variables floues. Pour ce faire on définit les deux notions suivantes :

Les fonctions d'appartenances qui permettent de définir le degré de vérité de la variable floue en fonction de la grandeur d'entrée

Les intervalles flous qui déterminent le nombre de variables floues

Dans l'exemple de la **Fig.II.2**, on fait intervenir trois intervalles flous: « petit », « moyen » et « grand ». En outre chaque intervalle fait référence à une fonction d'appartenance qui permet de définir le degré de vérité de la variable floue correspondante en fonction de la taille.

II.1.3.1 Fonctions d'appartenance

Il s'agit d'établir une relation entre le degré de vérité de la variable floue et la grandeur d'entrée correspondante (**Fig.II.3**). On parle de fuzzification :

Fig.II.3 : fonction d'appartenance

On peut évidemment choisir n'importe quelle forme pour les fonctions d'appartenance. Cependant, en pratique, on utilise les formes trapézoïdales (cas de la **Fig.II.2**) et les formes en cloche (gaussiennes).

II.1.3.2. Intervalles flous

Ces intervalles définissent le nombre de variables floues associées à une grandeur d'entrée. Dans le cas du réglage, trois à cinq intervalles s'avèrent suffisants. De façon générale ils sont caractérisés à l'aide de symboles tels que ceux présentés dans le tableau 1.

Symbole	Signification
NG	négatif grand
NM	négatif moyen
EZ	environ zéro
PM	positif moyen
PG	positif grand

Tableau II.1

II.1.3.3 Cas particulier : grandeur de sortie

La grandeur de sortie peut être définie à l'aide d'un certain nombre d'intervalles flous et diverses fonctions d'appartenance. Toutefois, en pratique, cette définition peut sembler assez

lourde et le concepteur (l'expert) peut choisir d'associer une seule valeur à chaque intervalle flou. Par exemple, pour une grandeur à cinq intervalles flous, on peut définir les valeurs suivantes (**Tableau II.2**) :

Intervalle	Valeur en % du maximum
très petit	0
petit	25
moyen	50
grand	75
très grand	100

Tableau II.2

Ce qui définit des fonctions d'appartenances en forme de raies comme illustré à la **Fig.II.4**

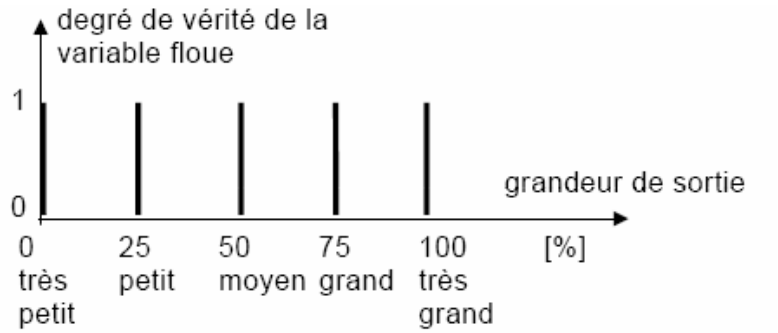


Fig.II.4

II.1.4 INFERENCES ET OPERATEURS

II.1.4.1 Règles d'inférences

Ces règles permettent de relier les variables floues d'entrée aux variables floues de sortie à l'aide de différents opérateurs. Elles doivent être définies par le concepteur de système de réglage en fonction de son expérience (rôle d'expert) et mémorisées dans l'organe de commande.

II.1.4.2 Opérateurs

Les règles d'inférences font appel aux opérateurs et, ou et non, qui s'appliquent aux variables floues. Dans le cas de la logique binaire ces opérateurs sont définis de façon simple

et univoque. Dans le cas de la logique floue, la définition de ces opérateurs n'est plus univoque et on utilise le plus souvent les relations présentées dans le **Tableau II.3.**

Opérateur	Opération sur le degré de vérité des variables
ET	minimum
	produit
OU	maximum
	valeur moyenne
NON	complément à 1

Tableau II.3

Les opérations minimum et maximum présentent l'avantage de la simplicité lors du calcul, par contre, elles privilégient l'une des deux variables. Les opérations de produit et valeurs moyennes sont plus complexes à calculer mais elles produisent un résultat qui tient compte des valeurs des deux variables.

II.1.5 COMBINAISON DE REGLES ET DEFUZZICATION

Les différentes règles d'inférences produisent chacune une valeur. Ces différentes valeurs doivent être combinées afin d'obtenir la (éventuellement les) variable(s) de sortie. Ensuite la (ou les) variable(s) floue(s) de sortie doivent être converties en une grandeur de commande (tension, couple...) afin d'être appliquée au système à régler. On appelle cette dernière étape la défuzzification.

II.1.5.1 Combinaison des règles

L'ensemble des règles se présente sous la forme d'une énumération du type :

Si condition 1 et/ou condition 2 (et/ou...) alors action sur les sorties

Si condition 3 et/ou condition 4 (et/ou...) alors action sur les sorties

Si condition 5 et/ou condition 6 (et/ou...) alors action sur les sorties

...

La combinaison de ces différentes règles se fait à l'aide de l'opérateur ou. La justification du choix de l'opérateur se fonde sur la pratique du langage courant : en effet, une telle énumération est comprise dans le sens

Si... alors...

ou

Si... alors...

ou ...

bien que l'opérateur ou ne soit pas explicitement mentionné.

II.1.5.2 Défuzzification

Les valeurs obtenues lors de la combinaison des règles appliquées aux intervalles flous de la variable de sortie défini une fonction d'appartenance.

Il s'agit de convertir cette information en une grandeur physique. Plusieurs façons de faire peuvent être envisagées mais, en pratique, on utilise surtout les deux méthodes suivantes:

• défuzzification par calcul du centre de gravité

• défuzzification par calcul du maximum

II.1.5.2.1 Défuzzification par calcul du centre de gravité

Il s'agit de calculer le centre de gravité de la fonction d'appartenance de la variable de sortie (**Fig.II.5**) :

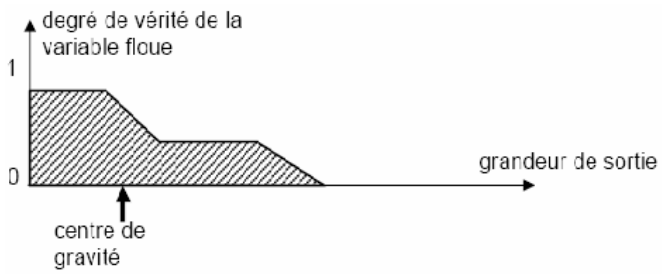


Fig.II.5

Le calcul du centre de gravité permet bien d'obtenir une seule valeur pour la grandeur de sortie. Son calcul est cependant relativement complexe puisqu'il nécessite le calcul d'une intégrale, ou dans le cas simple de fonctions d'appartenance en raies, d'une somme pondérée.

II.1.5.2.2 Défuzzification par calcul du maximum

Il s'agit de la façon la plus simple, au point de vue du volume de calcul, pour effectuer la défuzzification. La façon de procéder diffère cependant fondamentalement du cas général exposé ci-dessus. Tout d'abord, la grandeur de sortie doit être normalisée (en pour-cent par exemple) et la définition des intervalles flous doit se résumer à une valeur : par exemple « petit » correspond à 0 et « moyen » à 0,5 (fonctions d'appartenance en forme de raies). L'opération de défuzzification consiste à prendre d'abord le minimum entre la valeur produite par la règle concernée et la valeur de la variable floue de sortie. La valeur de sortie est définie par la valeur maximale des variables floues de sortie.

L'exemple suivant permet d'en illustrer le principe :

Soit un système avec trois règles :

La règle 1 donne une sortie de type « petit » d'un degré de vérité de 0,8

La règle 2 : sortie de type « moyen » d'un degré de vérité de 0,3

La règle 3 : sortie de type « grand » d'un degré de vérité de 0,1

La valeur normalisée de l'intervalle « petit » vaut 0

La valeur normalisée de l'intervalle « moyen » vaut 0,5

La valeur normalisée de l'intervalle « grand » vaut 1

La règle 1 donne une valeur de sortie de 0 (minimum entre 0,8 et 0)

La règle 2 donne une valeur de sortie de 0,3 (minimum entre 0,3 et 0,5)

La règle 3 donne une valeur de sortie de 0,1 (minimum entre 0,1 et 1)

La grandeur de sortie est déterminée par le maximum des valeurs obtenues et vaut 0,3 ce qui correspond à une valeur « plutôt petite ».

On constate que cette méthode est simple à appliquer mais, étant basée sur l'opérateur maximum, elle privilégie une seule règle parmi celles présentes.

II.2 LA POURSUITE DU POINT DE PUISSANCE MAXIMALE

Le générateur photovoltaïque transforme directement l'énergie solaire en énergie électrique. Cette énergie varie en fonction de l'éclairement et de la température. La fonction du courant fournie par le générateur en fonction de sa tension, la caractéristique I-V, du générateur passe par un point appelée Point de Puissance Maximale ou MPP (Maximum Power Point) qui est le point optimum où la puissance du générateur est maximale (I_m et V_m). Il est donc nécessaire de placer entre le générateur photovoltaïque et la charge, un convertisseur continu–continu afin d'adapter l'impédance de la charge au champ photovoltaïque **[13],** dont le contrôle du rapport cyclique permet de se maintenir au point de puissance maximale, quelque soit les variations du flux et de la température.**[14]**, **[15]**. Les performances de ce convertisseur dépendent essentiellement de sa topologie et de la loi de commande.

Pour un transfert de puissance optimale, la puissance fournie à la charge est maximale quand l'impédance interne du générateur est bien adaptée à la charge. Aussi pour en extraire le maximum de puissance électrique, il suffit de réajuster constamment le rapport cyclique qui est donné par l'équation suivante :

$$d = \frac{t_{on}}{T} \quad (1.10)$$

Avec $\mathbf{t_{on}}$: durée de l'impulsion et T : période du signal

La littérature nous offre de nombreuses techniques et méthode de recherche et d'ajustement du MPP **[15]**, **[17]**, nous avons choisi l'algorithme MPPT flou :

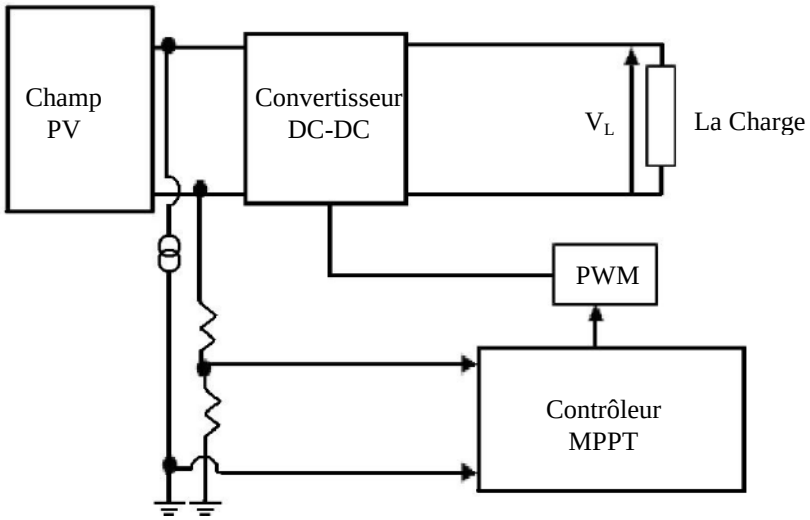


Fig.I1.6.a : Schéma bloc d'un système PV à base de contrôleur MPPT

II.2.1 LE CONTROLEUR FLOU APPLIQUE POUR LA POURSUITE DU MPP

La commande a pour but de poursuivre le point de puissance maximale qui correspond au point de fonctionnement optimale d'un générateur photovoltaïque pour différentes insolations et différentes valeurs de température.

Le contrôleur comporte les trois blocs suivants :

La Fuzzification des variables d'entrée par l'utilisation des fonctions trapèze ou triangulaire.

L'inférence où ces variables fuzzifiées sont comparées avec des ensembles prédéfinis pour déterminer la réponse appropriée.

La défuzzifiation pour convertir les sous-ensembles fuzzifiés.

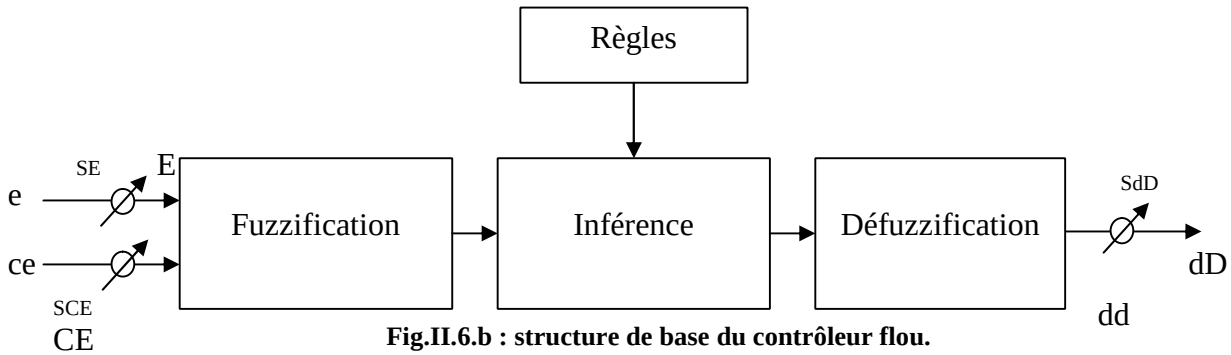


Fig.II.6.b : structure de base du contrôleur flou.

Cette Fig. montre la configuration du contrôleur flou qui se compose des :

Entrée-sortie d'échelle, fuzzification, l'inférence, et la défuzzification.

Les facteurs d'échelle : Se, SCE et SdD font changer les valeurs d'entrées et de sorties du contrôleur proportionnellement. La sortie dD est déduite par les deux variables E et CE, où cette dernière set dérivée du signal actuel par la division par le facteur d'échelle correspondant.

II.2.1.1 La Fuzzification

La tension et le courant actuel du générateur photovoltaïque sont mesurés instantanément puis échantillonnés et convertis avec un convertisseur A/N, et la puissance peut être ainsi calculée : $p(k) = i(k).v(k)$ (2.1)

On suppose que le contrôle se fait par la satisfaction des deux critères relative à deux variables d'entrée du contrôleur flou proposé, qui sont:

L'erreur (E) et le changement de l'erreur (CE) à des instants échantillonnés (*k*). les variables *E* et *CE* sont exprimées comme suit :

$$E(k) = \frac{p_{ph}(k) - p_{ph}(k-1)}{V_{ph}(k) - V_{ph}(k-1)} \quad (2.2)$$

$$CE(k) = E_{ph}(k) - E_{ph}(k-1) \quad (2.3)$$

Où P_{ph} *(k)* et V_{ph} *(k)* sont respectivement : la puissance et la tension du générateur photovoltaïque.

D'après l'entrée *E (k)* on peut savoir si le point de fonctionnement de la charge est situé à gauche ou à droite du point de puissance maximale de la courbe puissance-tension.

Donc :

Si *E (k)* est positif, le point de fonctionnement est à gauche du point de puissance maximale.

Si *E (k)* est négatif, le point de fonctionnement est à droite du point de puissance maximale.

Si *E (k)* est Zéro au du point de puissance maximale.

D'après l'entrée *CE (k)* on peut déterminer la valeur de la variation l'erreur de l'entrée du contrôleur flou qui représente la direction du point de fonctionnement.

Dans ce cas, pour atteindre le point de puissance maximale on doit :

Pousser vers la droite le point de fonctionnement qui se situe à gauche du point de puissance maximale.

Pousser vers la gauche le point de fonctionnement qui se situe à droite du point de puissance maximale.

En utilisant les bases des sous ensembles flou on peut exprimer les variable d'entrées et de sorties en terme de variable linguistiques, ces dernières sont écrites en étiquettes :

PG : positif grand

PP : positif petit

ZE : zéro

NP : négatif petit

NG : négatif grand

Le choix de cette classification est basé sur le raisonnement suivant :

On travail sur plusieurs phases de poursuite, a première est rude où on utilise un pas de recherche important pour diminuer le temps de réponse.

Une fois le point est proche du MPP, on utilise un pas plus faible pour diminuer les amplitudes des ondulations, c'est la phase fine.

Donc on choisi des variables linguistiques pour savoir si le point de fonctionnement est loin ou proche du MPP, afin d'arriver le plus vite possible à ce point.

La Fig.II.7 montre les fonctions d'appartenances des cinq sous ensembles flou des variables d'entrées et de la sortie.

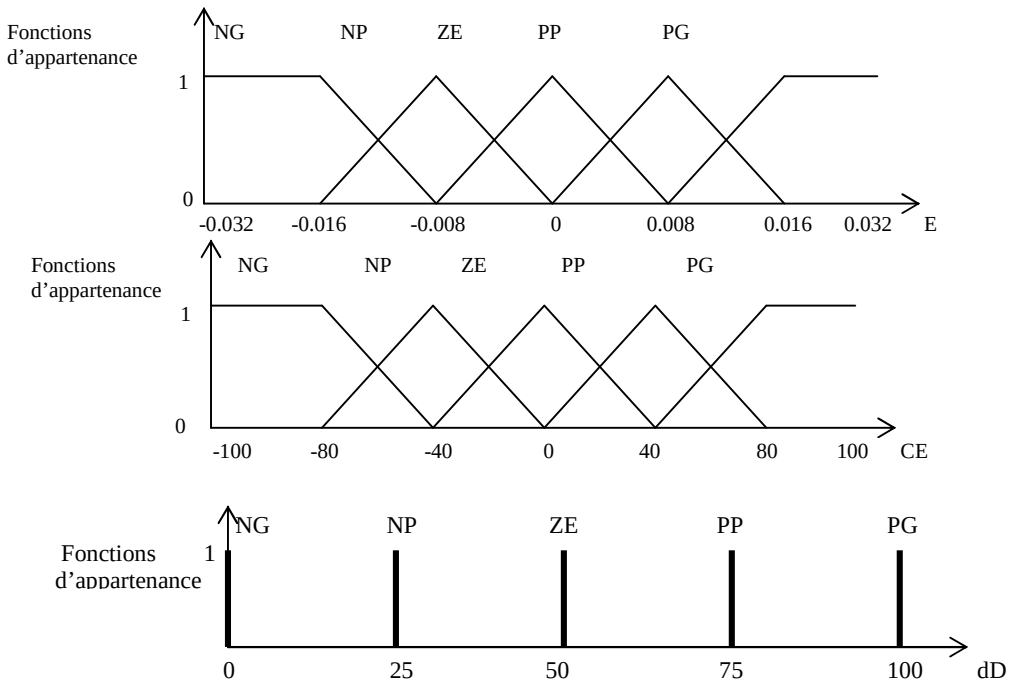


Fig.II.7 : fonction d'appartenances des variables d'entrées et de sortie.

II.2.1.2 La méthode d'inférence

La table II.4 montre les règles d'inférence du contrôleur flou où toutes les entrées de la matrice sont :

Les ensembles flous d'erreur (E) et le changement de l'erreur (CE).la sortie de cette table est le changement du rapport cyclique (dD).

Dans le cas de notre contrôleur flou, les règles de contrôle doivent être désignées de sorte que les variables d'entrée (E) soient toujours Zéro.

CE \ E	NG	NP	ZE	PP	PG
NG	ZE	ZE	PG	PG	PG
NP	ZE	ZE	PP	PP	PP
ZE	PP	ZE	ZE	ZE	NP
PP	NP	NP	NP	ZE	ZE
PG	NG	NG	NG	ZE	ZE

Table II.4 : Tale de règles floues.

On prend comme exemple de règle de commande de cette table :

Si E est NP et CE est PG alors dD est PP

Ce qui veut dire que : « si le point de fonctionnement est prés du point de puissance maximale (MPP) vers la droite,et la courbe caractéristique P-V est croissante avec un grande changement de la pente, alors diminuer le rapport cyclique (dD) légèrement ».

Nous avons choisi la méthode de Mamdani pour l'inférence floue avec opération MAX-MIN, qui consiste à utiliser l'opérateur MI N pur le ET et l'opérateur MAX pour le OU.

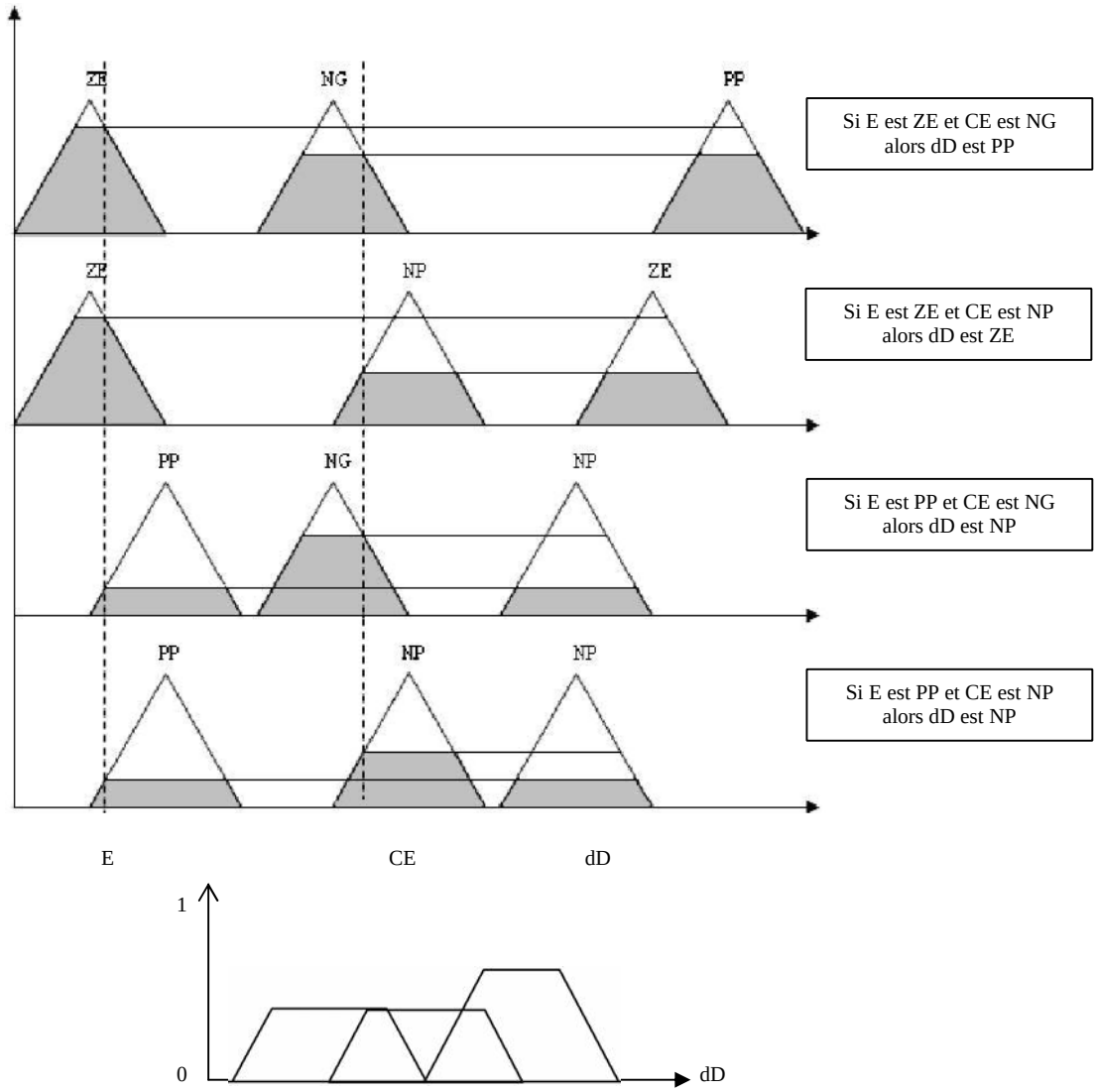


Fig.II.8 : L'inférence avec la loi de Composition MAX-MIN

Nous remarquons que pour une valeur de E et CE, il y'a quatre règles à vérifier (Fig.II.8) pour aboutir à la valeur de la sortie dD.

II.2.1.3 La Défuzzification

Les sorties floues des règles floues précédentes sont combinées en utilisant la fonction MAX pour former un seul ensemble flou, cet ensemble est défuzzifié pour générer une seule valeur de commande, il existe plusieurs méthodes de défuzzification, nous avons choisi celle des moyennes des maximums pour la simplicité de son implémentation.

Pour une présentation des données échantillonnées, la valeur de la commande est calculée par la formule :

$$dD = \frac{0 \times \mu_{NG} + 25 \times \mu_{NP} + 50 \times \mu_{ZE} + 75 \times \mu_{PP} + 100 \times \mu_{PG}}{\mu_{NG} + \mu_{NP} + \mu_{ZE} + \mu_{PP} + \mu_{PG}} \quad (2.3)$$

Pour l'exemple de la Fig.II.8, la sortie est calculée à partir de la figure suivante :

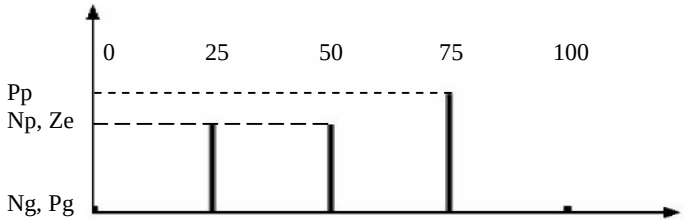


Fig.II.9 : La défuzzification

Les valeurs de sortie sont défuzzifiées et multipliées par le facteur d'échelle convenable pour construire le signal de contrôle.

II.3 CONCLUSION

On peut dire que la logique floue a les avantages de ses inconvénients. Elle n'est pas tellement rigoureuse mais très souple.

En effet, la logique floue a déjà fait ses preuves dans les domaines d'application suivants : la commande prédictive d'un plancher chauffant, le contrôle de la glycémie chez les diabétiques, la commande d'un pont roulant, la reconnaissance de caractères, ou encore le traitement d'images.

Les trois avantages principaux de la logique floue sont :

La souplesse de la maintenance

Adaptabilité à des conditions changeantes

Respect des contraintes en temps réel

Nous avons profité des avantages de la logique floue pour réaliser une commande MPPT, reste à implémenter cette commande, pour cela nous allons utiliser un outil et un support qui serons étudiés dans le chapitre suivant.

Chapitre

3

FPGA s & Langage VHDL

III.1 LES FPGA

L'électronique moderne se tourne de plus en plus vers le numérique qui présente de nombreux avantages sur l'analogique : grande insensibilité aux parasites et aux dérives diverses, modularité et (re)configurabilité, facilité de stockage de l'information etc. Les circuits numériques nécessitent par contre une architecture plus lourde, et leur mode de traitement de l'information met en œuvre plus de fonctions élémentaires que l'analogique. Il en découle des temps de traitement plus long. Aussi les fabricants de circuits intégrés numériques s'attachent à fournir des circuits présentant des densités d'intégration toujours plus élevée, pour des vitesses de fonctionnement de plus en plus grandes.

D'abord réalisées avec des circuits SSI (**S**mall **S**cale **I**ntegration), les fonctions logiques intégrées se sont développées avec la mise au point du transistor MOS (**M**etal **O**xyde **S**emiconductor) dont la facilité d'intégration a permis la réalisation de circuits MSI (Medium Scale Integration), puis LSI (**L**arge **S**cale **I**ntegration), puis VLSI (**V**ery **L**arge **S**cale **I**ntegration). Ces deux dernières générations ont vu l'avènement des microprocesseurs et microcontrôleurs. Ces derniers permettent de résoudre la majeure partie des problèmes logiques, par une solution programmée, mais se sont heurtés à plusieurs butoirs :

- D'une part, la gamme des produits standard offerts aux utilisateurs, ne répond pas tout le temps à leurs exigences en terme de performances.
- D'autre part, les outils de conception tardent à venir.

Entre ces circuits une autre gamme s'est ouverte : l'utilisation des ASIC (**A**pplication **S**pécifique **I**ntégration par **O**rdinateur), ou des circuits sur mesure (Full Custom), cette gamme a pu voir le jour, grâce aux progrès des outils CAO (**C**onception **A**ssistée par **O**rdinateur) qui permit aux ingénieurs de concevoir des circuits intégrés de manière automatique et sûre. Ces circuits regroupent tous les circuits dont la fonction peut être spécialisée et d'une application spécifique, avec un temps de développement très long et de très coûteux en investissement. D'où l'utilisation des circuits FPGAs (**F**ield **P**rogrammable **G**ate **A**rrays), qui émergent comme solution ultime aux problèmes de risques financiers dans le développement, aussi, afin de minimiser le temps de conception (prototypes rapides).

Les circuits FPGAs sont bien distincts des autres familles de circuits programmables tout en offrant le plus haut niveau d'intégration logique. La **Fig. III.1** montre la classification des différents circuits numériques, nous distinguons les circuits logiques standard (74HC…), les circuits à fonctionnement programmable (Microprocesseurs, Microcontrôleurs), les

circuits à architecture programmable et faible temps de développement (FPGA, PLD, CPLD) et les circuits à temps de développement important nous trouvons les ASIC.

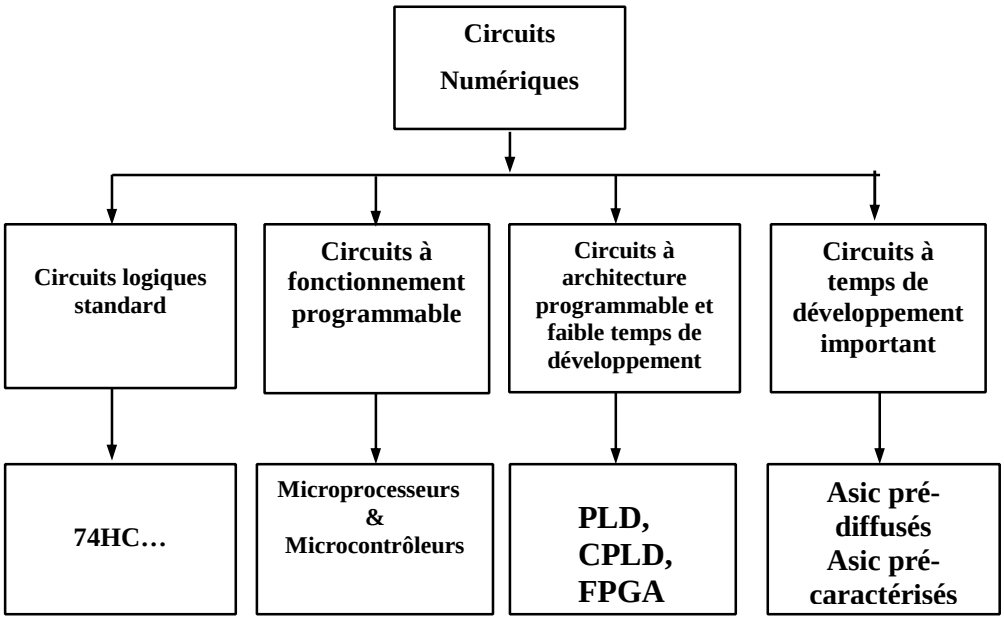


Figure III.1 : Classification des circuits numériques

Nous présenterons dans ce chapitre un aperçu surfacique sur les circuits FPGAs, nous nous intéresserons aux derniers circuits présents sur le marché à savoir le circuit FPGA de la famille Virtex-II. Cette dernière présente une densité d'intégration de [4 à 10 millions de portes], avec une vitesse dépassant les 420 Mhz, par la suite nous présenterons le langage de développement VHDL et l'outil Mentor graphique que nous avons utilisé dans notre implémentation.

III.1.1 LES CIRCUITS LOGIQUES PROGRAMMABLES

Ce sont des circuits programmables, rapides flexibles à très haute densité d'intégration. Ils peuvent être aussi de technologie PROM et EPROM, implémentant n'importe quelle table de vérité d'une fonction logique. Ces composants ne nécessitant aucune étape technologique supplémentaire pour être personnalisé. Nous y trouvons les circuits logiques programmables incluant un grand nombre de solutions, toutes basées sur des variantes de l'architecture des portes ET, OU, nous distinguons parmi ces circuits :

- Les **PLDs** "**P**rogrammable **L**ogic **D**evice" : Comprend une matrice de portes AND connectée à une matrice de portes OR il est alors représenté sous forme de sommes de produits, nous trouvons aussi des **EPLD** qui sont des circuits effaçables par rayons ultraviolets, ils peuvent être reprogrammé.

- Les **PALs** "**P**rogrammable **A**rray **L**ogic" : c'est la partie essentielle des PLD, elle est constituée d'un plan de portes AND programmables suivi d'un plan de portes OR fixé ou figée.

• Les **PLAs** "**P**rogrammable **L**ogic **A**rray" : Sont constitués par un plan de portes AND suivi d'un plan de portes OR. Mais dans ce cas les connexions dans les deux plans sont programmables.

• Les **EEPLDs** "**E**lectrically **E**rasable **PLD**" programmables et effaçables électriquement, ils peuvent être reprogrammés sur site. Les limites de l'architecture du PLD résident dans le nombre de bascules, le nombre de signaux d'entrées/sorties, la rigidité du plan logique ET/OU et des interconnexions. Précisons que ces composants très souples d'emploi sont limités à des fonctions numériques et adaptées à des productions de petites séries et ne présentent aucune garantie quant à la confidentialité.

• Les FPGAs **(F**ield P**rogrammable** G**ate** A**rrays)** ou "réseaux logiques programmables" sont des composants VLSI entièrement reconfigurables ce qui permet de les reprogrammer à volonté afin d'accélérer notablement certaines phases de calculs. Les circuits FPGAs sont constitués d'une matrice de blocs logiques programmables entourés de blocs d'entrée sortie programmable. L'ensemble est relié par un réseau d'interconnexions programmable.

Les FPGA, sont aujourd'hui utilisés dans des systèmes électroniques complexes de plus en plus variés. Pour répondre aux nouveaux marchés de ces composants, leurs fabricants proposent des gammes de choix très étendues, dans lesquelles on trouve des architectures, des tailles, des fonctionnalités bien différentes. Le développeur se trouve alors face à un large choix dans l'espace de conception d'où la disposition des différents outils capable de comparer entre elles les différentes architectures de composants reconfigurables.

Grâce aux évolutions de la technologie microélectronique les FPGA, deviennent de plus en plus performants avec des capacités sans cesse d'augmenter (jusqu'à 10 millions de portes pour le circuit Virtex II de Xilinx). Longtemps réalisés autour de blocs de logique configurable à base de LUT (Look Up Table), les FPGA peuvent aujourd'hui comporter de larges mémoires RAM configurables des opérateurs arithmétiques complexes (comme les Block Select RAM et les blocs multiplieur du Virtex II de Xilinx) et des cœurs de microprocesseurs (tel que le cœur ARM 9 intégré sur la puce des composants Excalibur d'Altera). Leur complexité et leurs possibilités en terme de réalisation de systèmes électroniques sur puce (SOC, System On Chip), ont permis aux FPGA de ne plus être seulement utilisés pour des applications de types prototypages rapides ou Glue Logic, prenant ainsi de la place dans l'espace de conception proposé aux développeurs, longtemps contraints

d'utiliser des circuits du type ASIC (Application Specific Integrated Circuit), chers et de fabrication délicate.

III.1.2 ARCHITECTURE INTERNE DES FPGAS

Nombreux autres fabricants, de moindre envergure, proposent également leurs propres produits. Les circuits FPGA possèdent une structure matricielle de deux types de blocs (ou cellules). Des blocs d'entrée sortie et des blocs logiques programmables. Le passage d'un bloc logique à un autre se fait par un routage programmable.

Certain circuit FPGA intègrent également des mémoires RAM, des multiplieurs et même des noyaux de processeur. Actuellement deux fabricants mondiaux se disputent le marché des FPGA : **Xilinx** et **Altera**. De nombreux autres fabricants, de moindre envergure, proposent également leurs propres produits. Nous ferons une description de l'architecture utilisée par Xilinx, qui se présente sous forme de deux blocs :

- Un bloc appelé circuit configurable.
- Un bloc appelé réseau mémoire SRAM. La couche dite "circuit configurable" est constituée d'une matrice de blocs logiques configurables CLB permettant de réaliser des fonctions combinatoires et des fonctions séquentielles. Tout autour de ces blocs logiques configurables, nous trouvons des blocs entrés/sorties IOB dont le rôle est de gérer les entrées-sorties réalisant l'interface avec les modules extérieurs **Fig. III.2**. La programmation du circuit FPGA appelé aussi LCA (logic cells arrays) consistera par le biais de l'application d'un potentiel adéquat sur la grille de certains transistors à effet de champ à interconnecter les éléments des CLB et des IOB afin de réaliser les fonctions souhaitées et d'assurer la propagation des signaux. Ces potentiels sont tout simplement mémorisés dans le réseau mémoire SRAM.

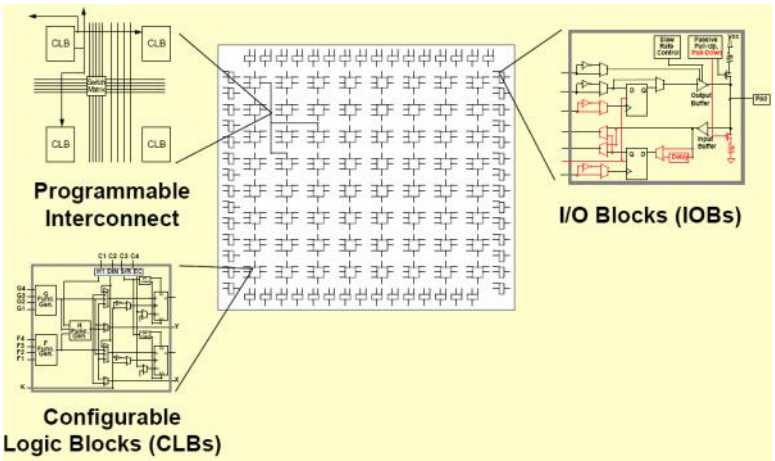


Fig. III.2 : Architecture interne du FPGA

Il existe 4 principales catégories disponibles commercialement : comme la montre la **Fig. III-3**.

- Tableau symétrique (Symmetrical Array).
- En colonne (Row Based).
- Mers de portes (Sea Of Gates).
- Les PLD hiérarchiques (Hierachical PLD).

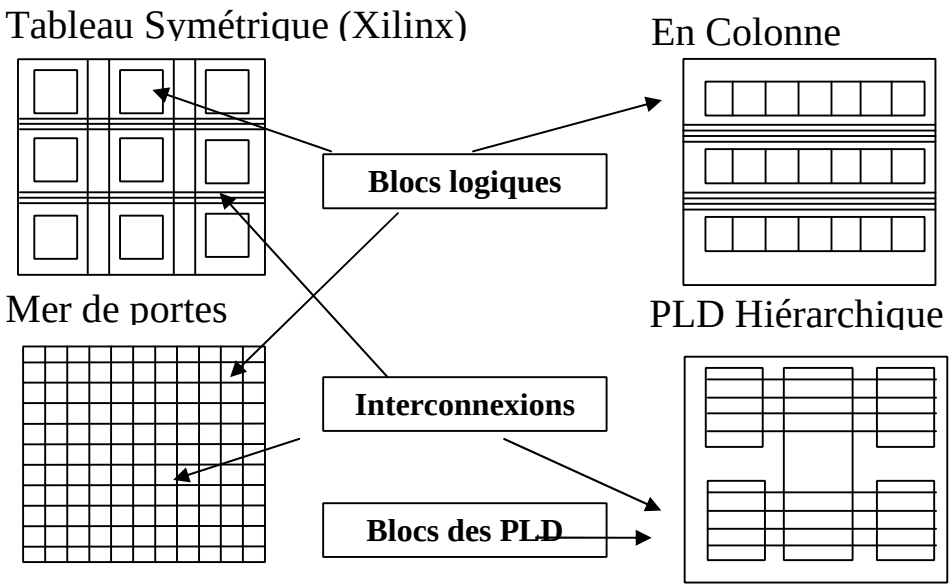


Fig. III.3 : Les différentes classes des FPGA

III.1.2.1. Types d'architectures des FPGAs

Classiquement on rencontre dans les architectures de FPGAs trois topologies différentes, voir **Fig. III.3**.

III.1.2.1.1. Architecture de type mer de portes

Elle est composée hiérarchiquement et le routage est de type logarithmique. Ce type de topologie fut utilisé par Xilinx pour sa série 6000. Mais ces composants n'ont pas eu de succès (commercialement parlant) par manque d'outils de CAO capables de les exploiter correctement. Celle-ci n'est plus utilisée.

III.1.2.1.2. Architecture de type îlots de calcul

Ce type d'architecture est celui choisi dés le départ par Xilinx. Dans ce cas, le FPGA est constitué d'une matrice plane d'éléments Ces éléments (que l'on détaillera dans la suite) constituent les ressources logiques et de routages programmables des FPGAs.

III.1.2.1.3. Architecture de type hiérarchique

Cette fois il existe plusieurs plans dans le FPGA, mais ces plans ne sont pas physiques, ils correspondent aux niveaux de hiérarchie logique. C'est à dire qu'un élément d'un niveau logique peut contenir des éléments de niveau logique inférieur, d'où la notion de hiérarchie.

Chaque niveau logique reprend la topologie d'une architecture du type îlots de calcul avec un routage dédié pour chaque niveau.

III.1.2.2. Les éléments de différentes architectures

Quelle que soit l'architecture choisie, les éléments constitutifs d'un FPGA sont toujours à peu près les mêmes. Chaque fabricant ayant ses variantes par rapport à un autre. Nous pouvons citer un certain nombre de ces éléments :

III.1.2.2.1. Les éléments logiques

Ce sont les briques de bases de tout FPGA. Grâce à leur configuration on peut réaliser dans ces blocs toutes les opérations de logique combinatoire (dans la limite d'un nombre d'entrées). Ces blocs ont souvent la même constitution et cela malgré la différence de fabricants et d'architectures. Ils sont généralement constitués d'une ou plusieurs LUTs (Look Up Table) qui contiennent, après configuration, la table de vérité de la fonction logique qu'elles doivent réaliser ou alors un ensemble de valeurs qui sont mémorisées comme dans une ROM. La taille des LUTs est généralement de 4 entrées. Les LUTs sont généralement suivies d'un registre de sortie, ce qui permet de synchroniser, si nécessaire, la sortie sur une horloge. La plupart des blocs logiques de bases sont munis d'une chaîne de propagation rapide de retenue, afin de former de petits additionneurs rapides.

III.1.2.2.2. Les éléments de mémorisation

Pour des applications plus importantes les FPGAs demandent souvent des capacités de stockage (citons en exemple les applications du traitement d'images). La nécessité d'intégrer des blocs de mémoires directement dans l'architecture des FPGAs est vite devenue cruciale. De cette façon les temps d'accès à la mémoire sont diminués puisqu'il n'est plus nécessaire de communiquer avec des éléments extérieurs au circuit.

III.1.2.2.3. Les éléments de routages

Les éléments de routages sont les plus importants dans un FPGA. En effet les ressources de routages représentent la plus grosse partie de silicium consommée sur la puce réalisant le circuit. Ces ressources sont composées de segments (de longueurs différentes) qui permettent de relier entre eux les autres éléments via des matrices de connexions. Le routage de ces ressources est un point critique du développement d'une application sur un FPGA, et les

méthodes utilisées pour les ASICs ne sont plus tout à fait valables (du fait de la segmentation des ressources). Si ces éléments sont importants c'est parce qu'ils vont déterminer la vitesse et la densité logique du système.

III.1.2.2.4. Les éléments d'entrées/sorties

Sur une carte, le circuit FPGA appartient à un système d'ensemble pouvant contenir des parties micro programmées comme dans le cas du *"Co-Design"*. Le circuit doit donc avoir un lien avec son environnement, c'est le but des éléments d'entrées/sorties. Ceux-ci peuvent bénéficier de protections, de buffer ou d'autres éléments permettant la gestion des entrées et des sorties. En particulier, il est à noter que les circuits actuels proposent différentes normes pour les niveaux d'entrées et de sorties (par exemple : LVTTL 2 - 54mA, PCI 5V, PCI 3.3V, HSTL…) qui par configuration peuvent êtres choisis afin de s'adapter à l'environnement.

III.1.2.2.5. Les éléments de contrôle et d'acheminement des horloges

Il paraît évident que dans tout système électronique relativement important, il faut disposer d'horloges et qu'elles sont souvent d'une importance capitale pour le bon fonctionnement du système. De ce fait, les FPGAs sont prévus pour recevoir une ou plusieurs horloges. Des entrées peuvent être spécialement réservées à ce type de signaux, ainsi que des ressources de routages spécialement adaptées au transport d'horloges sur de longues distances (bufferisation des lignes).

III.1.3 LES FAMILLES DES FPGAS DE XILINX

La tendance des dernières générations est de cibler certains créneaux porteurs du marché, comme la solution "bas coût" ou, à l'opposé la solution "haute performance" : la taille, le type et le nombre de cellules qui varient suivant les familles de composants. Ainsi, la panoplie présentée par le fabricant montre sa volonté de couvrir tous les segments de marché. Xilinx en est à sa septième génération de composants FPGA, on site :

XC2000 ; XC3000 ; XC4000 ; XC5200 ; XC6200 ; XC8100 ; SPARTAN ; VIRTEX ; VIRTEX-E ; VIRTEX-II ; VIRTEX-II Pro.

III.1.4 ARCHITECTURE DE LA FAMILLE VIRTEX-II

La famille Virtex II, encore plus performant, leur capacité est de 4 à 10 millions de portes dans une technologie de pointe : 0.15µm à 8 niveaux de métallisation. Leur architecture reste très près de celle du Virtex (des blocs multiplieurs câblés font leur apparition). Cette famille contient une architecture optimisée pour une grande vitesse (horloge système interne 420 MHz, cadence externe de 33 MHz à 133 MHz) et une consommation d'énergie minimale.

Elle combine entre la flexibilité dans l'intégration et la densité d'intégration. Elle comporte deux modules de base : les cores générateurs (fonctions DSP, fonctions mathématique, microprocesseur) et les modules personnalisés. Elle apporte des solutions pour les télécommunications, les réseaux, vidéo et les applications DSP. C'est la première famille à avoir intégré des multiplieurs et des blocs Select RAM dans sa structure interne.

Les technologies de pointe 0.15 µm / 0.12 µm CMOS à 8 couches de métallisation, le processus et l'architecture Virtex-II sont optimisées pour une grande vitesse d'horloge avec une faible consommation d'énergie. Elle combine entre la flexibilité dans l'intégration et la densité d'intégration plus de 10 millions de portes logiques, son architecture est divisée en deux blocs principaux, les blocs d'entrée/sortie programmable et bloc logique interne programmable comme le montre la fig. présentée si dessus **Fig. III.4**.

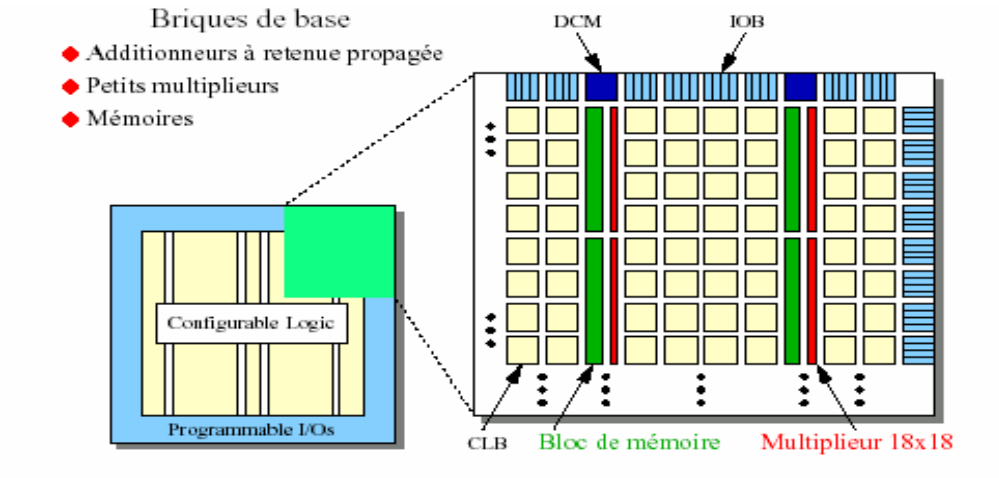


Fig.III.4 : Architecture interne de la famille VIRTEX-II

III.1.4.1 les blocs d'entrée/sortie programmable (IOB ou Input/Output Block)

Ils constituent l'interface entre les bornes du circuit et les CLB. Le dispositif de routage VersaRing offre les ressources nécessaires à l'interconnexion des CLB aux IOB. Nous pouvons ainsi modifier le système implanté sur le FPGA sans interférer avec l'attribution des bornes. Cette caractéristique s'avère importante si nous souhaitons développer des nouvelles versions d'un produit tout en conservant la compatibilité au niveau du boîtier. (Fig.III.5)

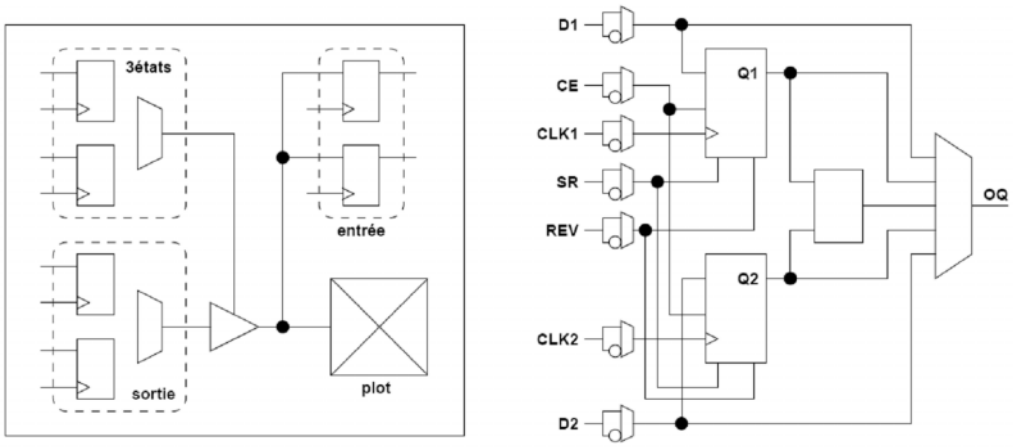


Fig. III.5 : Les blocs d'entrée/sortie programmable (IOB)

III.1.4.2 Les blocs logiques (CLB ou Configurable Logic Block)

Composés de quatre cellules logiques (LC ou Logic Cell), **Fig. III.6,** réparties en deux tranches identiques, ils servent à construire les circuits numériques implantés sur le FPGA. Chaque LC contient essentiellement :

III.1.4.2.1 Les blocs logiques configurables (CLBs)

C'est l'unité fondamentale du bloc logique qui fournit des éléments utilitaires pour la logique combinatoire et la logique synchrone, y compris les éléments du stockage de base : buffers à 3 états à associer avec chaque CLB. **Fig. III.6.** Les CLBs incluent quatre parties identiques appelées SLICE et deux buffers à 3 états.

Chaque SILICE est équivalente et contient :

- Deux générateurs de la fonction (F & G).
- Deux éléments du stockage.
- Des portes de la logique arithmétiques.
- Grands multiplexeurs.
- Une large fonctionnalité.
- Une logique de retenue rapide.
- Une chaîne de cascade Horizontale (porte OU).

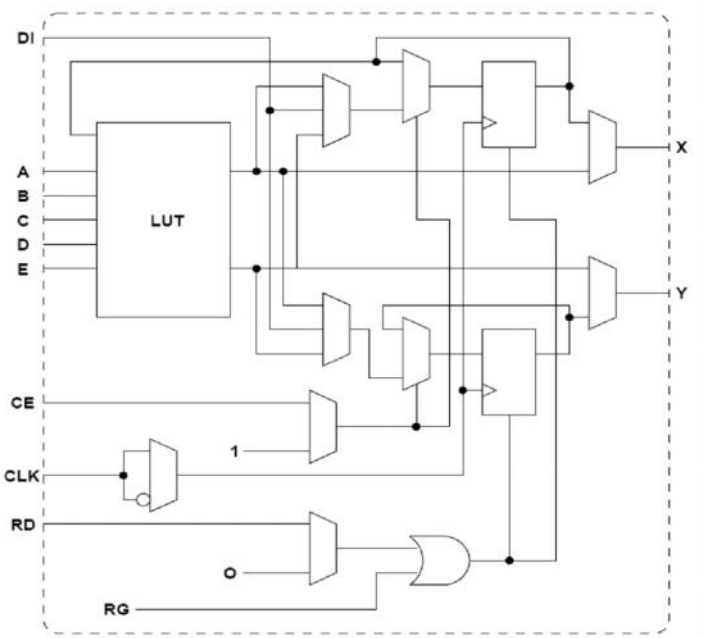


Fig. III.6 : Un bloc logique configurable (CLB)

III.1.4.2.2 Les blocs mémoires (Select RAM)

Ils possèdent des signaux d'initialisation (set et reset) synchrones ou asynchrones. Le bloc Select RAM produit de large élément de stockage (18 Kbit), programmable de 16K x 1 bit à 512 x 36 bits et peut être en deux blocs RAM (dual-port RAM). **Fig. III.7**.

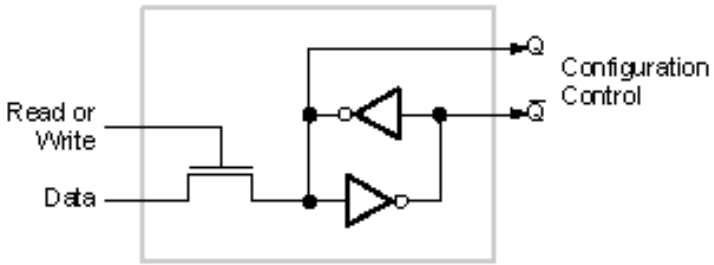


Fig. III.7 : Une cellule SRAM

III.1.4.2.3 Les blocs Multiplieurs

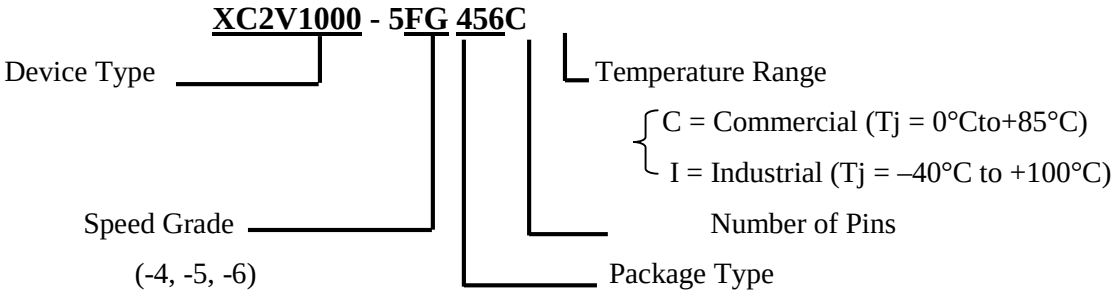
Les blocks multiplieurs sont des multiplieurs de18x18 bits. Le circuit VIRTEX-II incorpore une grande densité de blocs multiplieurs. Chaque multiplieur peut être associé au bloc Select RAM ou peut être utilisé indépendamment **Fig. III.4**.

III.1.4.2.4 Les blocs DCM (Digital Clock Manager)

Le DCM produit le nouveau système d'horloges (soit intérieurement ou extérieurement au FPGA), il produit une large gamme de fréquences de l'horloge de multiplication et même la division, le bloc logique programmable contient plus de 12 DCM voir **Fig. III.4**.

III.1.4.3 Nomenclature des circuits FPGA

Les circuits FPGA suivent la nomenclature suivante :



Device type : le type de la famille qui est dans notre exemple VIRTEX-II.

Speed grade : la vitesse du composons selon la technologie.

III.1.5 PROGRAMMATION ET CONFIGURATION DES CIRCUITS FPGAS

La configuration est le processus de charger des données spécifiques à un design dans un ou plusieurs FPGAs pour définir l'opération fonctionnelle des blocs internes ainsi que leur interconnexion, et leurs temps de configuration dépendent du mode de configuration sélectionnée.

Dans la littérature on peut distinguer 4 modes de configuration et programmation des circuits FPGAs comme le montre la **Fig. III.8** :

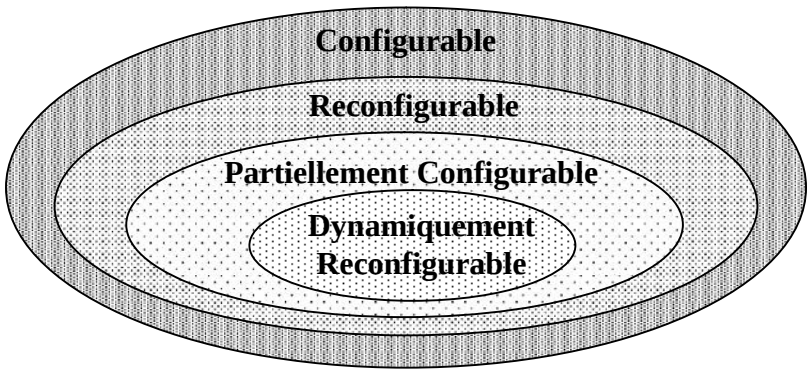


Fig. III.8 : Classification des circuits FPGAs selon leurs configurations

III.1.5.1 Circuit configurable

Un circuit Configurable est un circuit programmé et chargé par différentes données, où les interconnections d'un FPGA sont programmées afin de donner un fonctionnement spécifique pour un tel circuit.

III.1.5.2 Circuit reconfigurable

C'est le même principe de configuration, sauf que cette fois en reconfigurant le circuit FPGA une deuxième fois pour l'utilité dans une autre fonction comme on peut garder la dernière configuration du circuit, et même on peut effacer cette configuration et on config. le circuit une nouvelle fois.

III.1.5.3 Circuit partiellement reconfigurable

Un dispositif ou un circuit est défini comme partiellement reconfigurable (dans la littérature en trouve aussi la terminologie Run Time Reconfiguration RTR globale) s'il est possible de le reconfig.r sélectivement, tandis que l'état de repos du reste du dispositif est inactif, mais il conserve son information configurée. Encore, il ne semble pas y avoir n'importe quel dispositif sur le marché qui soit partiellement reconfigurable, mais non aussi dynamiquement reconfigurable, La reconfiguration partielle permet de rendre un FPGA effectif, multiple fonctions, et change des fonctions pendant le fonctionnement du système.

III.1.5.4 Circuit dynamiquement reconfigurable

Un circuit FPGA est reconfigurable dynamiquement (dans la littérature on trouve la terminologie Run Time Reconfiguration RTR local) s'il peut être partiellement reconfiguré durant son fonctionnement, c.-à-d. une partie du circuit correspondant à certaines fonctions logiques et leur interconnections peut être changé sans affecter le fonctionnement de la logique restante.

On peut aussi parler de reconfiguration dynamique dans le cas ou plusieurs circuits FPGAs sont connectés entre eux et il s'agit de reconfig.r un seul composant FPGA tout en maintenant les autres circuits en fonctionnement.

III.2 LE LANGAGE VHDL

III.2.1 HISTORIQUE ET PRESENTATION DU LANGAGE VHDL

L'apparition du langage VHDL remonte aux années 80. À cette époque, le département de la défense des États-Unis motivé par le besoin de développer un langage normalisé de design et de documentation parraina un projet de développement du VHDL (de l'anglais,Very High Speed Integrated Circuits Hardware Description Language). Quelques années plus tard, en 1987, VHDL fut adopté par IEEE comme une norme de langage de description pour le matériel. Depuis il a été largement accepté par l'industrie.

Langage VHDL est conçu de manière à nous permettre de modéliser des systèmes complexes décrits à des niveaux d'abstractions très différents. En d'autres mots, VHDL intègre adéquatement les méthodologies de conception de systèmes dites "Top/Down". De plus, VHDL est un langage modulaire et hiérarchique. Un système complexe peut être divisé en plusieurs blocs, chaque bloc peut à son tour être divisé en plusieurs sous blocs et ainsi e suite. L'interface entre les différents blocs se fait par des "liens de communication". De la même manière, un modèle VHDL peut être composé de

plusieurs sous-modules, chaque sous-module peut être divisé à son tour en d'autres sous-modules et ainsi de suite. VHDL a d'abord été proposé comme un langage de modélisation et il a graduellement été utilisé comme un langage de synthèse. Cependant, il faut noter que les possibilités de modélisation avec VHDL dépassent de loin les besoins de la synthèse. Certaines caractéristiques abstraites de VHDL n'ont pas d'équivalent dans la logique digitale. Autrement dit, la synthèse ne supporte qu'un sous ensemble des possibilités de VHDL.

Parmi les avantages du langage VHDL nous insistons sur les aspects suivants :

- Cycle de design plus court.
- Description plus compacte, moins d'erreurs.
- Exploration de l'espace de design, re-design.
- Maîtrise de la complexité.
- Description portable (indépendante de la technologie).

III.2.2 Description d'un module en VHDL

Un module VHDL élémentaire est appelé entité. Il est principalement composé de deux éléments de base : l'énoncé entity et l'énoncé architecture. L'énoncé entity décrit l'interface entre l'entité et le monde extérieur (i.e. les signaux d'entrée/sortie, les broches de la puce). L'énoncé architecture décrit quant à lui son contenu interne, son comportement. Notons ici qu'une entité VHDL peut avoir plusieurs architectures différentes. Ceci nous permet de générer des vues différentes de la même entité à des niveaux d'abstraction différents.

III.2.2.1 L'énoncé ENTITY

L'énoncé entity définit le nom de l'entité, les noms des ports d'entrées/sorties, la direction des ports (entrée, sortie...) ainsi que leurs types logiques (un port de 1 bit, un bus de 8 lignes, etc.). La direction du port nous indique s'il s'agit d'un signal d'entrée, de sortie ou "inout". De plus, le type logique nous indique la nature logique du port. La **Fig. III.9** montre une vue générale de la syntaxe d'une déclaration d'entité.

```
entity <nom de l'entité> is
port(
        <Nom du port1> : <Direction> <Type Logique>;
        <Nom du port1> : <Direction> <Type Logique>;
                                                    ...;
                                                ...; );
end <nom de l'entité>;
```

Fig. III.9 : Vue générale de la syntaxe d'une déclaration d'entité

```
library ieee;
use ieee.std_logic_1164.all;
-- Description de l'entité incrementeur
entity incrementeur is
port (
        A: in   std_logic_vector (11 downto 0);
        D: out std_logic_vector (11 downto 0);
        UP: in std_logic                        );
end incrementeur;
```

Fig. III.10 : Exemple d'entité avec déclaration de bibliothèque

Un exemple de code VHDL d'un énoncé entity est présenté ci-dessus à la **Fig. III.10.** Voyons en détail chaque ligne de cet exemple:

library ieee;

use ieee.std_logic_1164.all;

Pour des raisons de flexibilité les types logiques des ports sont souvent définis au préalable dans des bibliothèques. En VHDL, les bibliothèques sont spécifiées par le mot clé library. Pour avoir accès à une partie de la bibliothèque on utilise l'énoncé use. On a donc accès dans l'entité incrementeur à tous les types définis dans le package std_logic_1164 de la libraire IEEE. Cela nous permet entre autres d'utiliser le type std_logic.

-- Description de l'entité incrementeur

Tout comme dans la plupart des langages de programmation, le VHDL permet d'introduire des commentaires. Les commentaires sont introduits en plaçant deux tirets consécutifs "--" et se terminent dans tous les cas à la fin de la ligne. Évidemment les commentaires sont ignorés par le compilateur VHDL.

entity incrementeur **is**

On spécifie ici le nom de l'entité créée. Ce nom est choisis par le programmeur et reflète habituellement l'utilité du nouveau design. Les mots entity et is sont des mots clés du langage VHDL. Ils indiquent le début de la description d'un nouveau design en VHDL.

```
port(   A: in    std_logic_vector (11 downto 0);
D: out std_logic_vector (11 downto 0);
UP: in std_logic                        );
```

On déclare ici chacun des ports du module que l'on veut faire. Chacune des lignes comprend le nom donné au port, sa direction (in, out, inout) ainsi que le type de ce port. Le type indique le genre de données qui vont circuler par cette entrée/sortie. Par exemple le port UP est un port d'entrée qui peut traiter avec des données du type std_logic.

end incrementeur;

La fin de la déclaration de l'entité est indiquée avec le mot réservé end. Il est facultatif de placer ensuite le nom de l'entité, mais c'est une bonne habitude de le faire pour que les programmes soient plus clairs et faciles à lire.

III.2.2.2 L'énoncé architecture

L'énoncé architecture contient un identificateur architecture ainsi que le nom de l'énoncé entity qui définit ses entrées/sorties. Il contient aussi la déclaration des signaux internes et des sous- éléments de ce circuit. La **Fig. III.11** donne une vue générale de la syntaxe VHDL d'une architecture.

```
architecture <nom de l'architecture> of <nom de l<entité> is
--déclaration des signaux internes;
begin
        <instructions>…
        …
end <nom de l'architecture>
```

Fig. III.11.a : Vue générale de la syntaxe d'une déclaration d"architecture

Tout comme pour la partie entity, nous allons voir ligne par ligne un exemple d'architecture très simple. Le schéma logique de cet exemple est présenté à la **Fig. III.11** et le code VHDL associé à ce circuit sur la **Fig. III.12** . Comme vous pouvez le constater, il s'agit d'un circuit logique qui effectue l'opération suivante:

$$F = \overline{(A * B) + (C * D)}$$

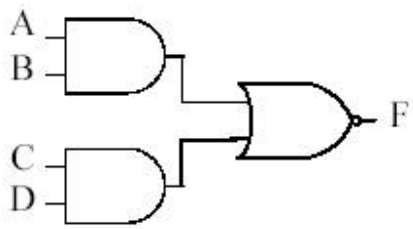


Fig. III.11.b : Schéma logique d'un ET-OU-INVERSEUR

```
library ieee;
use ieee.std_logic_1164.all;
-- Description de l'entité
entity et_ou_inv is
port(
        A, B, C, D : in std_logic;
        F : out std_logic);
end et_ou_inv;
-- Description de l'architecture reliée à l'entité
architecture behav of et_ou_inv is
begin
        F <= not ((A and B) or (C and D));
end behav;
```

Fig. III. 12 : Code VHDL du circuit de la Fig. III.11

architecture behav **of** et_ou_inv **is**

C'est ici que vous donnez un nom à votre architecture. Ce nom est arbitraire, mais il est important d'en donner un, car pour une même entité plusieurs modélisations sont possibles. Ainsi, à une entité peuvent appartenir plusieurs architectures. achitecture , of et is sont des mots clés du langage VHDL. Noter aussi qu'il est nécessaire de spécifier à qu'elle entité réfère notre architecture, dans notre cas nous implémentons le comportement de l'entité et_ou_inv.

begin

Le mot clé begin indique la fin de la partie déclaration de l'architecture. La partie déclaration permet d'introduire dans le design des signaux internes. Dans cet exemple simple nous n'avons pas besoin de signaux et cette partie est vide. C'est donc après le begin que nous commençons la partie instruction du programme.

F <= **not** ((A **and** B) **or** (C **and** D));

La partie instruction de cette architecture est plutôt simple, la sortie F est une fonction logique des entrées A, B, C et D. not, and et or sont tous des opérateurs du langage VHDL.

end behav;

Tout comme pour l'énoncé entity, end signale la fin de la description de l'architecture. Il est optionnel de placer à la suite du end le nom de l'architecture, mais on le fait pour avoir un code plus facile à maintenir.

Cet exemple vous a permis de voir comment on décrit un module simple en VHDL. Ce langage permet de décrire de différentes façons le même module ou la même architecture d'une entité donnée. En générale on distingue deux approches ou type de description : description structurelle et description comportementale. Les prochaines sous -section traitent des ces différentes approches.

III.2.2.2.1 Description structurelle

L'approche structurelle permet de décrire le fonctionnement d'un circuit en fonction de ses composants physiques. La description du fonctionnement des composants physiques eux-mêmes doit alors exister dans une bibliothèque prédéfinie ou définie par le programmeur. Pour illustrer en quoi consiste la description structurelle d'un circuit, considérons le sommateur complet dont le circuit logique est donné par la **Fig. III.13.**

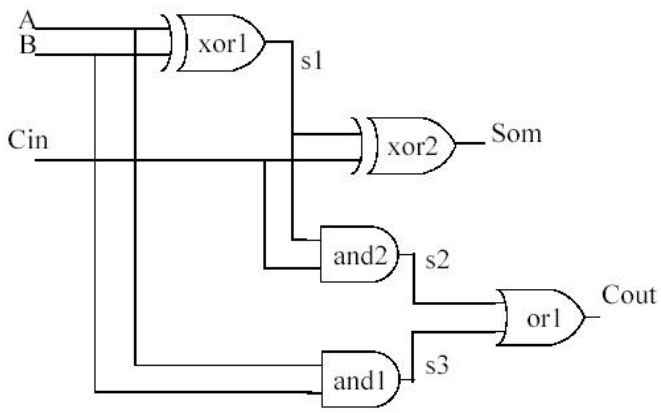


Fig. III.13 : Schéma logique d'un sommateur

La description structurelle de ce sommateur est présentée à la **Fig. III.14**.Les mots clés réservés sont en gras tandis que les étiquettes ou variables définies par le programmeur sont en lettres minuscules. Tel qu'il a été mentionné, toute modélisation doit commencer par une définition d'entité. Cette définition débute par le mot clé entity suivie du nom du circuit modélisé ("sommateur") et contient la définition de ses signaux d'entrées et de sorties. Une initialisation de ces signaux est également possible à ce stade-ci, comme c'est le cas pour le signal "cin" qui est initialisé à "0". La définition d'architecture suit celle de l'entité et commence avec le mot-clé architecture, suivi du nom de l'architecture ("struct1"). Entre la définition de l'architecture et sa description au niveau des portes logiques se situent la déclaration des signaux internes (mot-clé signal) et des composants (mot-clé component) faisant partie du circuit du sommateur.

Évidemment, la déclaration des ports de ces composantes doit correspondre à celle qui a été faite dans le fichier de description de la composante. On peut se représenter les composants comme les portes du circuit de la **Fig. III.13** et les signaux comme les fils qui relient chacune des portes.

définition de l'entité

```
entity sommateur is
port(
    A,B : in qsim-state;
    C,in : in qsim_state:='0';
    Som : out qsim_state   );
end sommateur;
```

définition de l'architecture
signaux internes

```
architecture struct1 of sommateur is
signal s1, s2, s3: qsim_state;
```

définition des composantes utilisées dans la modélisation structurelle du sommateur

```
component xor
port(x1, x2 :in qsim_state; xo1: out qsim_state);
end component;
 component and
        port(x1, x2 :in qsim_state; xo1: out qsim_state);
end component;
component or
port(x1, x2 :in qsim_state; xo1: out qsim_state);
end component;
```

description structurelle du sommateur

```
begin
xor1 : xor port map(A, B, s1);
        xor2 : xor port map(s1, Cin, Som);
        and1 : and port map(A, B, s3);
and2: and port map (s1, Cin, s2);
or1: or port map (s2, s3, Cout);
end struct1;
```

Fig. III.14 : Code VHDL structurel du sommateur de la Fig. III.13

De plus, les composants déclarés (xor, and et or) sont "liés" à des entités existantes. La description VHDL de ces composants pourrait se trouver dans une bibliothèque ou avoir été faite avant le début de la modélisation du sommateur par le programmeur dans un autre fichier (dans ce cas les fichiers VHDL doivent se trouver dans le même répertoire de travail). À chaque fois que l'on écrira xor? (ou ? est un nombre), on utilise en fait une composante xor dont le comportement est défini par son architecture. Il en va de même pour les autres composantes (and et or).

Connaissant le circuit simple du sommateur à 1 bit, sa description structurelle était relativement simple à réaliser. Pour des circuits composés de milliers de portes logiques, cette façon de procéder pourrait être très longue. Une description de type comportementale serait alors préférable, tout au moins pour une première simulation.

On peut distinguer trois grandes étapes générales lors de la réalisation d'une architecture structurelle:

1. identifier tous les signaux qui permettrons de relier les composantes entre elles (les fils sur le circuit) et les déclarer (signal);

2. recenser tous les composants utilisés (les puces) et les définir par la syntaxe :

component <nom de la composante>
port (...)
end component;

3. Affectation des liaisons du schéma aux broches des composantes (les fils entre chaque portes) avec le mot réservé port map (…).

III.2.2.2.2 Description comportementale de type algorithmique

Le type de description comportementale se subdivise, d'après le style de programmation, selon deux sous catégories: algorithmique et flux de données. Ces deux styles ne sont toutefois pas mutuellement exclusifs et une combinaison des deux est parfois appelée style mixte.

La description de la **Fig. III.16** est un exemple classique de description comportementale de type algorithmique alors que le programme de la **Fig. III.18** est du type flux de données. L'approche comportementale de type algorithmique est différente des deux autres méthodes. Lorsqu'on fait la description VHDL en style algorithmique, on ne se base pas sur un schéma logique, c'est une approche "boîte noire". Le programmeur décrit le comportement des entrées et des sorties de la composante (son fonctionnement) sans avoir à se soucier du design intérieur de celle-ci.

L'intérieur de la puce créée sera déterminé lors de la synthèse, par l'outil de synthèse. Le modèle de la **Fig. III.16** est un multiplexeur à deux entrées implémenté selon le style comportemental algorithmique. Comme toujours la description commence par la définition de l'entité "multiplex". Ensuite on décrit le comportement de notre multiplexeur de façon algorithmique. Le style comportemental algorithmique est en fait un programme VHDL qui utilise des instructions de contrôle comme if...then...else, case...when, etc..., comme dans la plupart des langages de programmation.

Une structure importante dans une architecture de type comportemental et qui est utilisée à la **Fig. III.16** est la boucle process. Cette boucle peut avoir une liste de sensibilité ou non. Ainsi, l'instruction process (sel,a,b) indique que les instructions contenues entre le début et la fin de la boucle process ne sont à exécuter que dans le cas où l'un des trois signaux "sel", "a" et "b" subit une transition d'état. Une boucle process sans liste de sensibilité est en fait une boucle sans fin.

L'énoncé process sera traité à la section III.2.2.3.

```
use ieee.std_logic_1164.all;
entity multiplex is
port(a,b,sel : in std_logic;
        y : out std_logic);
end multiplex;

architecture comportemental of multiplex is
                begin
```

```
process(sel,a,b) -- la liste de sensibilité
                        begin
    if sel = '0' then -- si sel vaut 0 alors
        y <= a;
    elsif sel = '1' then -- si sel vaut 1 alors
        y<= b;
    else
        y <= 'x'; -- si sel n'est pas à 0 ou à 1, y vaut X (état indéterminé)
    end if;
end process;
end comportemental;
```

Fig. III.16 : Exemple de description comportementale algorithmique

III.2.2.2.3 Description comportementale de type flux de données

Le type de description flux de données est illustré à la Fig. 18 où le fonctionnement d'un comparateur simple est décrit. Ce comparateur est schématisé ci-dessous à la **Fig. III.17**. Le flux de données est caractérisé par une assignation séquentielle ou concurrentielle de valeurs aux signaux de sortie du circuit. On décrit donc en fait la relation logique entre les différents nœuds du circuit (entrées, sorties et noeuds internes). Le programme VHDL que l'on fait montre la façon dont les données ou signaux se propagent dans le circuit. L'exemple de la **Fig. III.13** est également du type comportemental flux de données.

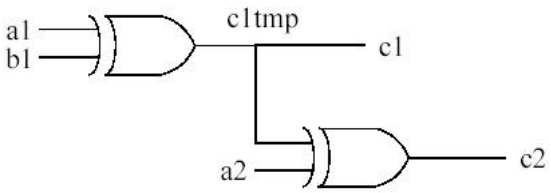


Fig. III.17 : Schéma logique du design de la Fig. III.18

Comme vous pouvez le voir dans cet exemple (**Fig. III.18**), on assigne aux signaux de sorties la combinaison des entrées qui produisent le résultat. La sortie c1 est donc obtenue par les signaux d'entrée a1 et b1 qui passent par une porte XOR (xor est un opérateur du langage VHDL). Le même principe est appliqué pour la sortie c2, on calcule le résultat de l'expression (c1tmp nor a2) et on assigne la réponse à la sortie c2. On peut remarquer aussi qu'on utilise le signal c1tmp comme intermédiaire. Ce signal est nécessaire car on ne peut pas placer un port de sortie dans la partie de droite d'une assignation de signal.

```
library ieee;
use ieee.std_logic_1164.all;
entity compar2 is
    port (a1, b1, a2 : in   std_logic := '0';
                            c1 : out std_logic;
                c2 : out std_logic       );
    end compar2;
```

```
architecture behavior of compar2 is
   signal c1tmp : std_logic;
begin
    c1tmp <= a1 XOR b1;
    c2 <= c1tmp XOR a2;
    c1 <= c1tmp;
end behavior;
```

Fig. III.18 : Exemple de description comportementale « Flux de données »

III.2.2.3 L'énoncé process

L'énoncé process constitue un élément puissant de la description VHDL. Il permet d'écrire des instructions parallèles qui utilisent les mêmes signaux (voir **Fig. III.19**). Un process peut être vu comme un sous-programme en boucle infinie qui ne s'arrête qu'à l'instruction de synchronisation wait. On accompagne habituellement l'énoncé process d'une liste de sensibilité, et le process est exécuté une fois au complet lorsqu'un des signaux de la liste change d'état. La liste de sensibilité du process est donc interprétée comme une instruction wait à la fin du process. La liste de sensibilité du process doit comprendre tous les signaux qui sont lus durant l'exécution de la boucle (c'est-à-dire tous les signaux se trouvant dans la partie de droite d'une assignation) afin de tenir compte des changements d'état sur ces signaux. Durant une simulation, un process ne peut être que dans une des situations suivantes:

- en exécution comme un sous-programme;
- ou en attente d'une transition d'un ou de plusieurs signaux, à l'aide de l'instruction wait.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity sr_latch is
          port ( s_n : in  std_logic ;
                 r_n : in  std_logic ;
                   q : out std_logic ;
                 q_n : out std_logic);
end sr_latch;
architecture struct of sr_latch is
              signal s_qtmp : std_logic;
      signal s_qtmp_n : std_logic;
begin
      process (s_n, s_qtmp_n)
      begin
            s_qtmp <= s_n nand s_qtmp_n;
      end process;
      process (r_n, s_qtmp)
      begin
            s_qtmp_n <= r_n nand s_qtmp;
      end process;
```

```
 q <= s_qtmp;
q_n <= s_qtmp_n;
end;
```

Fig. III.19 : Bascule SR, exemple de parallélisme

III.2.2.4 Les signaux et les variables

On peut déclarer des variables dans un process, une procedure ou une function. Il est aussi possible de leur donner une valeur initiale qui sera attribuée à chaque appel de la procédure ou de la fonction. Noter cependant qu'il faut être prudent avec les valeurs initiales, elles facilitent le travail lors des simulations fonctionnelles, mais elles sont ignorées après la synthèse. Votre circuit ne fonctionnera donc pas correctement dans la réalité. Il est préférable d'initialiser les variables lors d'un reset par exemple et non à leur déclaration. Comme dans les langages de programmation, la valeur de la variable est modifiée aussitôt qu'elle est assignée. En général, une variable ne génère aucune porte supplémentaire lors de la synthèse. En fait, les variables servent surtout à manipuler de la valeur intermédiaire ou temporaire qui vont faciliter le développement d'un algorithme séquentiel. Une variable peut générer un élément de délai si elle est lue avant d'être assignée. Voici comment on utilise les variables

Déclaration:

variable <nom_de_variable> : type;

Assignation:

var_destination **:=** var_source;
Examples:

```
variable i: integer range 0 to 3;
variable tmp: std_logic:= '0';
i := 2 ;
```

Dans les circuits intégrés, l'information se propage sur les fils. En VHDL, ces fils sont appelés des signaux (**signal**). Les signaux doivent être déclarés dans la partie déclarative de l'architecture. Lors de la déclaration d'un signal, on doit spécifier son type et il est aussi possible de lui attribuer une valeur par défaut.

Déclaration:

signal <nom_du_signal> : type;
Assignation:

<nom_du_signal> **<=** expression;
Exemples:

```
signal etat_courant, etat_suivant : bit;
etat_suivant <= '0';
A <= B;
```

Noter bien que A <= B ne signifie pas que la valeur du signal A est B, mais signifie plutôt que la valeur de B est enregistrée dans un tampon de A comme valeur projetée. L'assignation (ou le lien physique) ne se fait qu'à la prochaine instruction de synchronisation (l'instruction **wait**). Il faut aussi souligner que dans la partie déclarative d'une architecture, on ne peut pas déclarer desvariables, alors que dans la partie déclarative d'un **process**, on ne peut déclarer des signaux.

III.2.2.5 Les types de données (data types)

Le langage VHDL supporte plusieurs types de données. Chaque constante, signal, variable ou fonction est déclarée avec un type donné. Voici les principaux types supportés par le langage VHDL:

1.Le type énuméré:

```
type COLOR is (RED, GREEN, YELLOW, BLUE, VIOLET);
type MY_LOGIC is ('0', '1', 'U', 'Z');
variable COL: COLOR;
signal SIG: MY_LOGIC;
. . .
COL:=GREEN;
SIG <= 'Z';
```

2. Le type numérique:

```
type entier is range 0 to 255; -- entier qui peut prendre des valeurs situés entre 0 et 255
```

3. Le type vecteur:

```
type BYTE is array (7 downto 0) of BIT; -- tableau de 8 bit : BYTE(7) à BYTE(0)
type VECTOR is array (3 downto 0) of BYTE; -- tableau à deux dimensions
variable tmp: BYTE
. . .
tmp (3 downto 0):="0010"
tmp: ="10111000"
```

III.2.2.3 Applications du langage VHDL

Le VHDL est un langage permettant de faire :

1. La spécification : le langage VHDL est très bien adapté à la modélisation des systèmes numériques complexes grâce à son niveau élevé d'abstraction. Le partitionnement en plusieurs sous ensembles permet de subdiviser un modèle complexe en plusieurs éléments prêts à être développés séparément.

2. La simulation : la notion du temps, présente dans le langage, permet son utilisation pour décrire des fichiers de simulation (Test-Bench). Le modèle comportemental avec les fichiers de simulation peut constituer, ensemble, un cahier de charges. Les fichiers de simulation peuvent également être utilisés avec un banc de tests de production.

3. La synthèse logique : les logiciels de synthèse permettent de traduire la description VHDL en logique. Il est ainsi possible d'intégrer la description dans un composant programmable (CPLD, FPGA) ou dans un circuit ASIC.

4. La preuve formelle : le langage permet de prouver formellement que 2 descriptions sont parfaitement identiques au niveau de leur fonctionnalité.

III.2.3 Utilisation du langage VHDL pour la synthèse

Les débuts de son utilisation en synthèse furent assez difficiles. Chaque société ayant adapté le langage VHDL à sa manière. De 1993 à 1997, différentes adaptations de la norme IEEE définissent sa conception numérique : description VHDL et synthèse, utilisation pour la synthèse. Il faut attendre la fin des années 90 pour que tous les outils intègrent ces nouvelles modifications. Parallèlement, la forte évolution des circuits logiques programmables dans les années 1990, nécessite de disposer d'un langage de haut niveau afin de maîtriser la complexité toujours plus importante. En Europe, le VHDL s'est imposé comme un standard reconnu par tous les principaux vendeurs d'outils de développement.

Simulation et synthèse : Le développement en VHDL nécessite l'utilisation de deux outils : le simulateur et le synthétiseur. Le premier va nous permettre de simuler notre description VHDL avec un fichier de simulation appelé "Test-Bench". Cet outil interprète directement le langage VHDL. Le simulateur comprend l'ensemble du langage. L'objectif du synthétiseur est très différent. Il doit traduire le comportement décrit en VHDL en fonctions logiques de bases. Celles-ci dépendent de la technologie choisie. Cette étape est nommée : "synthèse". Le langage VHDL permet d'écrire des descriptions d'un niveau comportemental élevé. La question est de savoir si n'importe quelle description comportementale peut être traduite en logique ? L'intégration finale dans le circuit ciblé est réalisée par l'outil de placement et routage. Celui-ci est fourni par le fabricant de la technologie choisie. Avec les outils actuels, il est possible de disposer des fichiers VDHL à chaque étape. Le même fichier de simulation "Test-Bench" est ainsi utilisable pour vérifier le fonctionnement de la description à chaque étape.

La **Fig**. III.20 montre le déroulement des différentes étapes de développement d'un projet en VHDL.

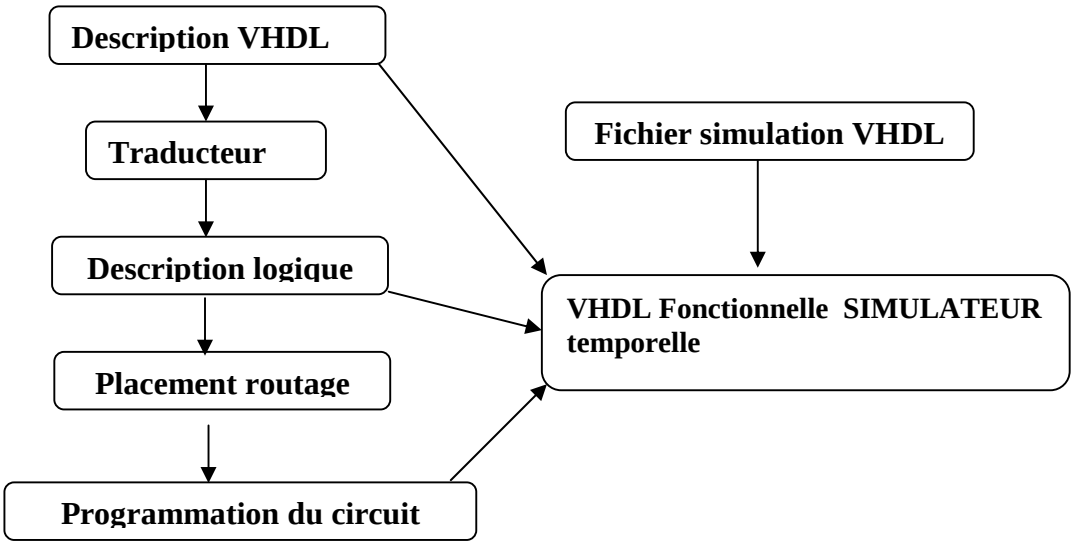


Fig. III.20 : les différentes étapes de développement d'un projet en VHDL

III.2.4 L'outil de conception Mentor Graphics

Nous avons utilisé pour notre application l'outil Mentor Graphics qui contient 4 modules de dessin et de vérification des circuits, comme le montre la **Fig. III.21** :

1. Module « Design Architect IC ».
2. Module « FPGA Advantage ».
3. Module « Design for Test ».
4. Module « IC Station »

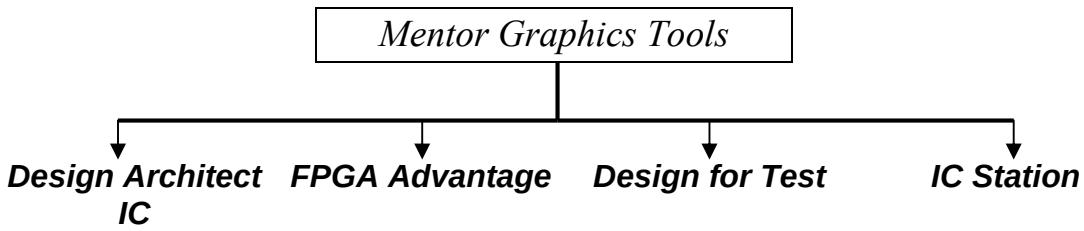


Fig. III.21 : Les différentes outils de conception

Dans notre cas nous avons utilisé le module « FPGA Advantage »pour l'implémentation de l'algorithme MPPT Floue sur le circuit Virtex-II, l'outil FPGA Advantage contient 3 environnements essentiels de conception.

1. Environnement de modélisation en langage VHDL : Cet environnement permet la description de l'architecture ou d'un design en VHDL, soit en utilisant un éditeur de schéma ou bien en utilisant les blocs digrammes ou bien les diagrammes d'états.

2. Environnement de simulation et vérification de circuits intégrés (Model Sim) : la simulation se fait en parallèle après chaque bloc pour vérifier si le design est opérationnel.

- Une simulation fonctionnelle après la description en code VHDL.

• Une simulation temporelle ou statique après l'implémentation de l'algorithme sur le circuit désiré.

3. Un environnement de placement et routage: Dans cet environnement on réalise l'implémentation et le routage sur un circuit FPGA (Virtex-II dans notre cas).

CONCLUSION

A travers ce chapitre, nous avons essayé de donner en premier lieu les descriptions et caractéristiques essentielles des FPGA en abordant leur architecture et configurations ; en second lieu nous nous sommes intéressé à l'un des langage de développement de ces derniers qu'est le VHDL.

A partir de cette étude nous avons jugé les circuits FPGA performants vu les nombreux avantages qu'ils présentent. Le premier avantage est la souplesse de programmation qui permet l'emploi conjoint d'outils de schématique aussi bien que l'exploitation d'un langage de haut niveau tel VHDL. Ce qui permet de multiplier les essais, d'optimiser de diverses manières l'architecture développée, de vérifier à divers niveaux de simulation la fonctionnalité de cette architecture. Le second avantage est évidemment la nouvelle possibilité de reconfiguration dynamique partielle ou totale d'un circuit ce qui permet d'une part, une meilleure exploitation du composant, une réduction de surface de silicium employé et donc du coût, et d'autre part, une évolutivité assurant la possibilité de couvrir à terme des besoins nouveaux sans nécessairement repenser l'architecture dans sa totalité. L'un des points forts de la reconfiguration dynamique est effectivement de permettre de reconfigurer en temps réel en quelques microsecondes tout ou partie du circuit, c'est à dire de permettre de modifier la fonctionnalité d'un circuit en temps quasi réel. Ainsi le même CLB pourra à un instant donné être intégré dans un processus de filtrage numérique d'un signal et l'instant d'après être utilisé pour gérer une alarme. On dispose donc quasiment de la souplesse d'un système informatique qui peut exploiter successivement des programmes différents, mais avec la différence fondamentale qu'ici il ne s'agit pas de logiciel mais de configuration matérielle, ce qui est infiniment plus puissant.

Chapitre

4

Description & Simulation

IV.1 DESCRIPTION DU SYSTEME

Nous allons essayer d'expliquer et illustrer les différents composants du contrôleur flou implémenté sur l'FPGA Virtex II de Xilinx. Nous avons utilisé l'environnement de développement ISE 7.1 de Xilinx, quant à la simulation nous avons utilisé ModelSim 5.7 de Mentor Graphiques.

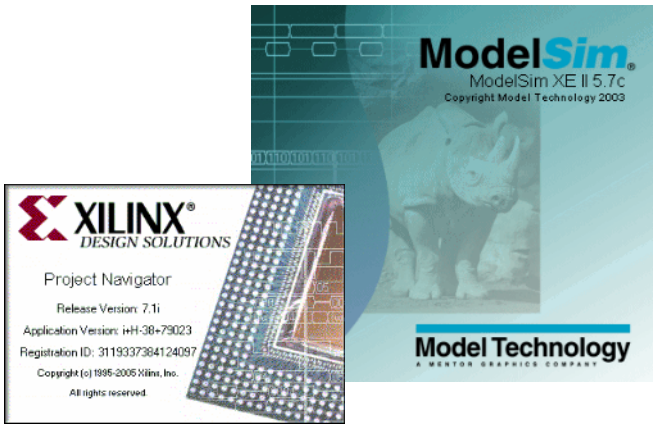


Fig.IV.1 : Les outils de développement ISE 7.1 & ModelSim

Le langage VHDL est conçu de manière à nous permettre de modéliser des systèmes complexes décrits à des niveaux d'abstractions très différents. De plus, il est modulaire et hiérarchique. Un système complexe peut être divisé en plusieurs blocs, chaque bloc peut à son tour être divisé en plusieurs sous blocs et ainsi de suite.

Pour notre cas, le système possède l'hiérarchie suivante (Fig.IV.2) :

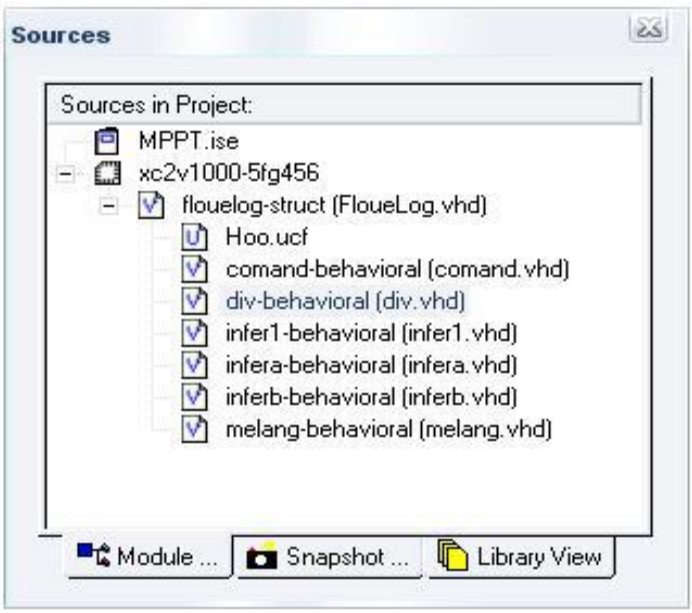


Fig.IV.2 : la hiérarchie du système

Le module principal dans cette hiérarchie, décrit par l'entité «floulog » contient les sous modules suivants :

- Hoo.ufc
- Infera.vhd et infrb.vhd
- melang.vhd
- Infer1.vhd
- Comand.vhd
- Div.vhd

Chacun de ces composant sera détaillé dans la section suivante.

IV.1.1 Le Module principal

Afin de poursuivre le point de puissance maximale, nous devons, à chaque instant, adapter la tension de la charge à celle du module solaire, cette adaptation par variation de la tension est assurée par un convertisseur continu-continu piloté par notre contrôleur MPPT flou qui mesure pour chaque échantillon du courant et de la tension, l'erreur st sa variation (section II.2). Le contrôleur flou, calcule à partir de ces entrées, le rapport cyclique qui attaque un dispositif de modulation en largeur d'impulsion, pour que ce dernier génère le signal de commande du convertisseur.

Pour cela, le module principal, vu comme une boite noire, a pour entrées, évidemment après échantillonnage, numérisation et adaptation (facteurs d'échelles), les deux valeurs E et CE, de l'erreur la variation de l'erreur respectivement, codées sur huit bits signés.

Et comme la majorité des systèmes numériques, notre entité globale à une entrée d'horloge pour la synchronisation de toutes les opérations.

La sortie dD, nous donne la valeur du rapport cyclique codée sur huit bits.

La figure IV.2 illustre ce qui préecde.

Fig.IV.3: Floulog

IV.1.2 les sous modules ou composants du système

IV.1.2.1 l'entité Hoo.ufc

Ce fichier contient les contraintes d'implémentation, comme l'attribution des pines de la

carte aux entrées et sorties du système décrit dans l'entité principale, les caractéristiques du signal d'horloge (période, rapport cyclique, pine spécifique selon la fréquence).

Le fichier des contraintes peut être édité comme un texte ou en utilisant le outils de « User Constraints » dans la fenêtre des processus.

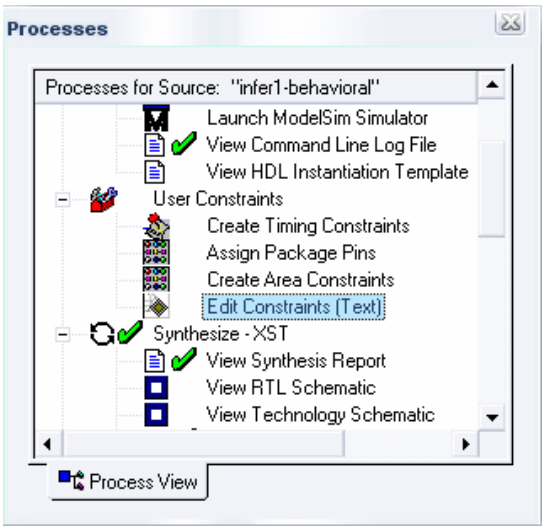


Fig.IV.4 : outil d'édition de contraintes de l'utilisateur

IV.1.2.2 l'entité Infera.vhd et infrb.vhd

Ces deux entités ont pour rôle de *fuzzifier* les deus variables E et CE d'entrée. Grâce aux intervalles *uniformes* des fonctions d'appartenance, nous avons considéré chacun comme une *zone* ayant un code sur trois bits (trois bits pour pouvoir coder les six zones)

Sachant le code de la zone d'appartenance, par une simple comparaison de la valeur d'entrée avec les valeurs limites des zones, il suffit de calculer une valeur pour déduire la seconde, car nous savons que chaque valeur d'entrée engendre deux règles.

La fuzzification d'une variable est facile vu l'allure des fonctions d'appartenance dans chaque zone qui sont des segments de droites, donc, il suffit d'avoir la pente des segments et la largeur des intervalles pour calculer tout simplement la valeur fuzzifiée puis déduire la deuxième par la complémentation à 1.

Nous avons choisi les limites des zones d'appartenance comme des puissance de deux (2^n) en gardant les même proportionnalités pour faciliter les calcules (la division par une puissance de deux devient de simples décalages vers la droite). (Fig.IV.5)

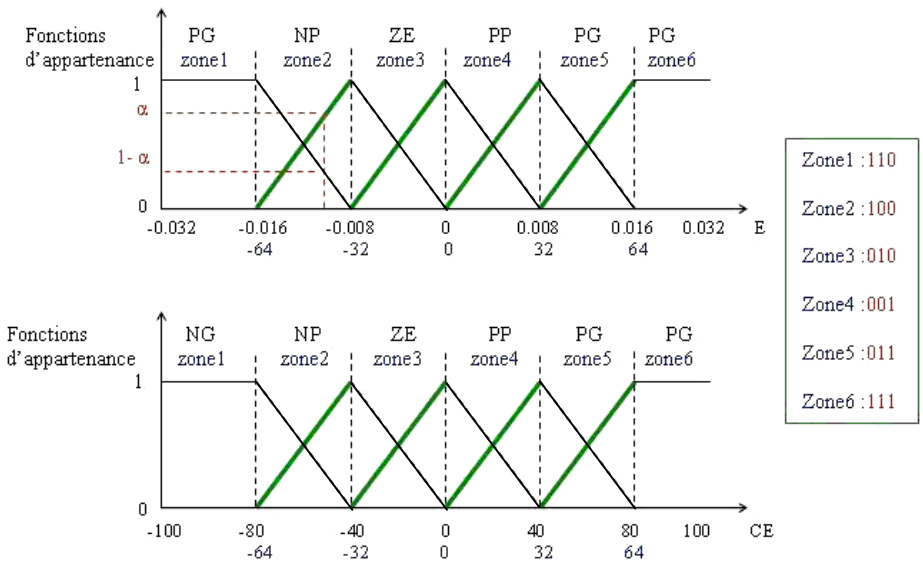


Fig.IV.5:illustration de l'algorithme de fuzzification

Les deux entités ont pour entrées un bus de huit bits signé et une horloge,quant à la sortie,elles ont un bus de huit bits pour transmettre la valeur fuzzifiée et un bus de trois bits pour donner la zone où nous travaillons.(Fig.IV.6)

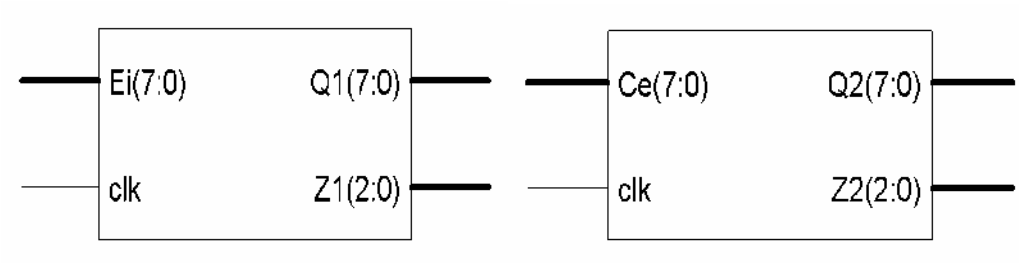


Fig.IV.6: les entités infera et inferb.

IV.1.2.3 l'entité melang.vhd

Le rôle de ce composant est de déduire les règles correspondantes aux valeurs E et CE à partir leurs valeurs fuzzifiées puis effectuer l'opération MIN pour obtenir l'ensemble d'éléments à défuzzsifier.

Cette entité a pour entrées deux bus de huit bits correspondant à E et CE *fuzzifiées* respectivement et une horloge. Pour les sorties, elle possède quatre bus de huit bits qui correspondent aux éléments d'entrée de la phase de défuzzification. (Fig.IV.7)

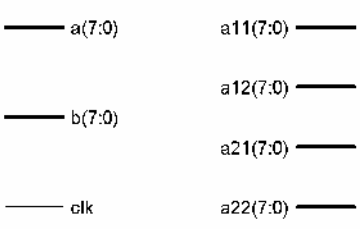


Fig.IV.7: l'entité melang

IV.1.2.4 l'entité Infer1.vhd

C'est la première phase de défuzzification, il s'agit de combiner les règles en se basant sur les valeurs des éléments de l'entité précédente et la connaissance des zones de travail, cette entité effectuer l'opération MAX pour obtenir le sous ensemble flou de la sortie dD.

Cette entité a pour entrées quatre bus de huit bits pour les éléments précédents, un bus de six bits résultant de la concaténation des deux bus Z1 et Z2 provenant des entités infera et inferb respectivement et une horloge. Les sorties sont les cinq bus de huit bits qui correspondent aux éléments de sous ensemble flous de dD. (Fig.IV.8.a)

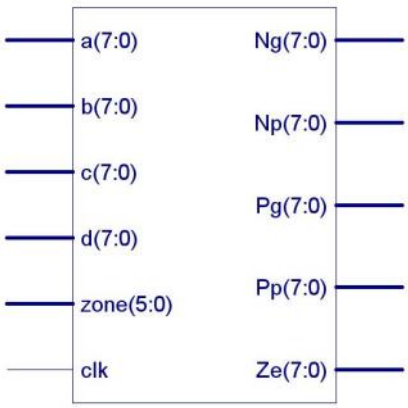


Fig.IV.8.a : l'entité infer1

IV.1.2.5 l'entité Comand.vhd

C'est le cœur le la phase de défuzzification, ce composant reçoit les éléments du sous ensemble flou de dD pour calculer le numérateur et le dénominateur de la formule de défuzzification (section II.2.13, formule 2.3)

$$dD = \frac{0 \times \mu_{NG} + 25 \times \mu_{NP} + 50 \times \mu_{ZE} + 75 \times \mu_{PP} + 100 \times \mu_{PG}}{\mu_{NG} + \mu_{NP} + \mu_{ZE} + \mu_{PP} + \mu_{PG}}$$

Donc les entrées de cette entité sont les cinq bus de huit bits et l'horloge, alors que les sorties sont deux bus, l'un est de seize bits pour le numérateur, il est plus large car c'est le résultat d'une opération de multiplication, ce qui double sa taille, tandis que l'autre est huit bits pour le dénominateur. (Fig.IV.8.b)

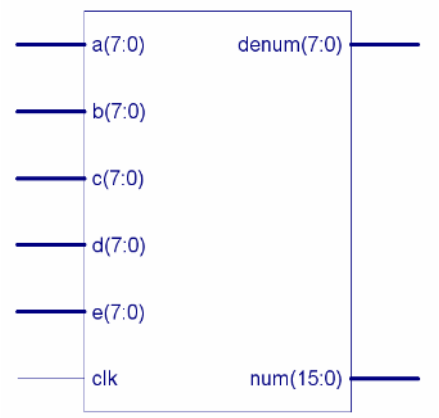


Fig.IV.8.b : l'entité comand

IV.1.2.6 l'entité Div.vhd

La dernière phase de défuzzification, et le dernier composant du régulateur, cette entité effectue la division des deux membres de la formule (2.3).

Elle reçoit les deux bus de seize et huit bits avec un signal d'horloge, pour enfin, donner en sortie, sur un bus de huit bits, la valeur du rapport cyclique dD.

Fig.IV.9 : l'entité div

IV.2 SIMULATION DU SYSTEME

Après la spécification et la description du système on procède à sa simulation temporelle et fonctionnelle grâce au logiciel ModelSim. Cela est illustré dans ce qui suit.

Nous allons simuler chaque composant à part pour s'assurer de son bon fonctionnement, puis nous passons à la simulation du système globale.

IV.2.1 SIMULATION DES DIFFERENTS COMPOSANTS DU SYSTEME

La simulation consiste à forcer les entrées d'une entité et voir si les sorties sont en accord avec les résultats voulus, ce qui justifie le comportement du système y décrit. De plus le simulateur nous permet de vérifier chaque signal interne du système pour déceler et corriger d'une façon efficace une éventuelle anomalie.

IV.2.1.1 simulation de l'entité infera ou inferb

Toujours, Nous commençons d'abord par le réglage du signal d'horloge (période, rapport cyclique, front et valeur logique ...) (Fig.IV.10)

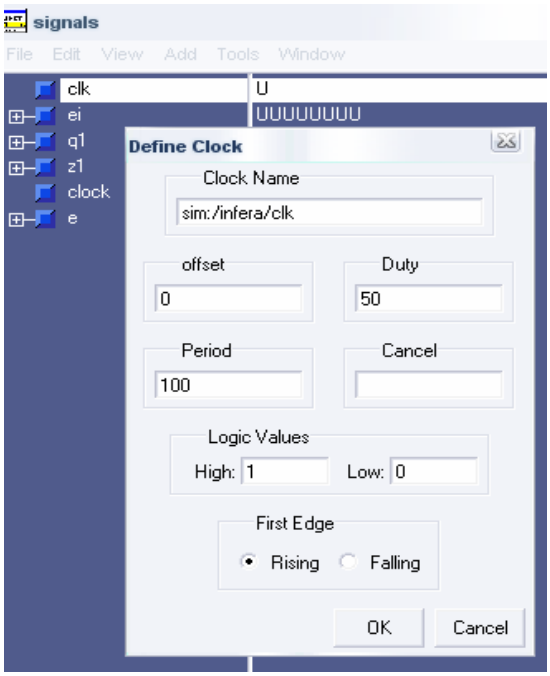


Fig.IV.10. spécification des paramètres de l'horloge

Nous choisissons ensuite le type du signal à afficher (clique droit de la souris) (Fig.IV.11)

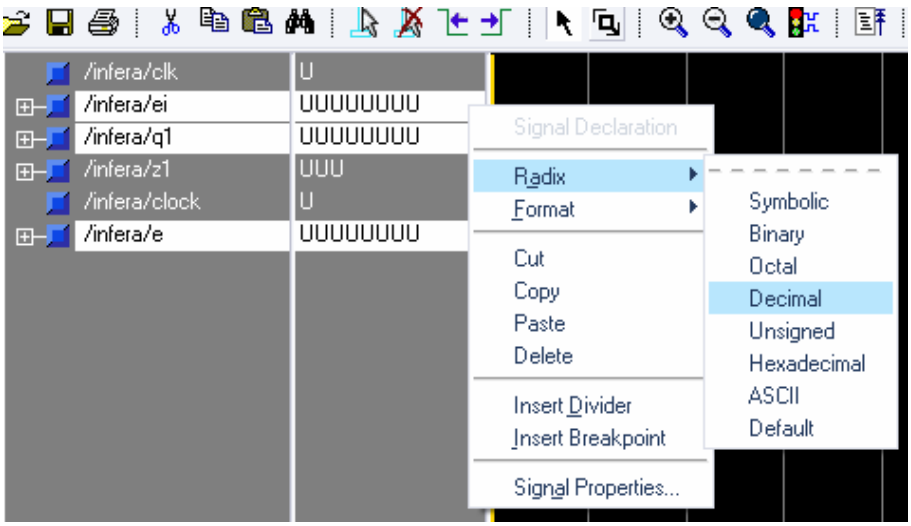


Fig.IV.11. choix du type d'affichage des signaux

Puis nous forçons les entrées à partir de la fenêtre des signaux (Fig.IV.12), il s'agit, pour l'entité infera, d'un seul signal à forcer « Ei ».

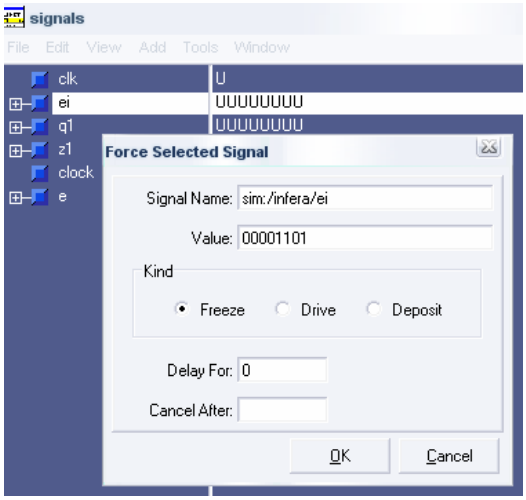


Fig.IV.12. donner des valeurs aux variables d'entrées

La figure ci dessous illustre les résultats de la simulation, pour chaque valeur de Ei nous avons en sortie les deux valeurs, la zone de travail « Z1 » et celle de l'entrée fuzzifiée « Q1 ».

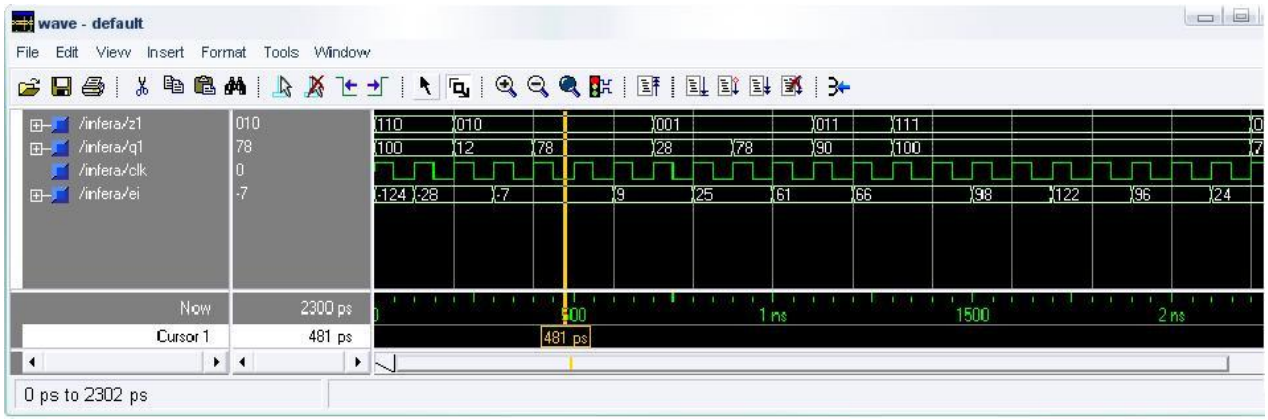


Fig.IV.13. infera -simulation temporelle et fonctionnelle

Pour vérifier le bon fonctionnement de ce composant, nous pouvons se référer à la figure IV.5 et au tableau II.4 de la section II.2.1.2.

Après la simulation temporelle et fonctionnelle, l'outil utilisé nous permet d'avoir un aperçu sur l'architecture interne et l'organisation des différents éléments constituant le système, cela nous pouvons le voir sur la figure suivante : (Fig.IV.14, Fig.IV.15)

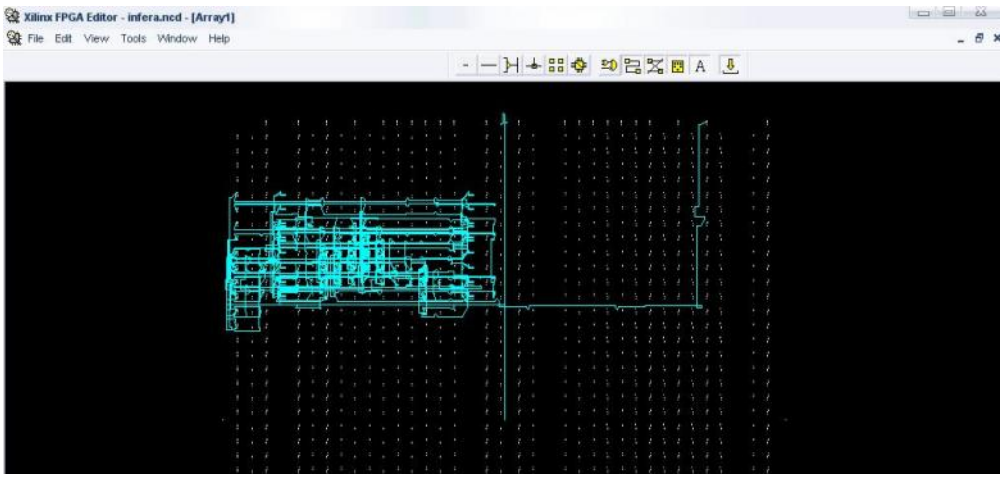


Fig.IV.14. infera- résultat du placement et routage

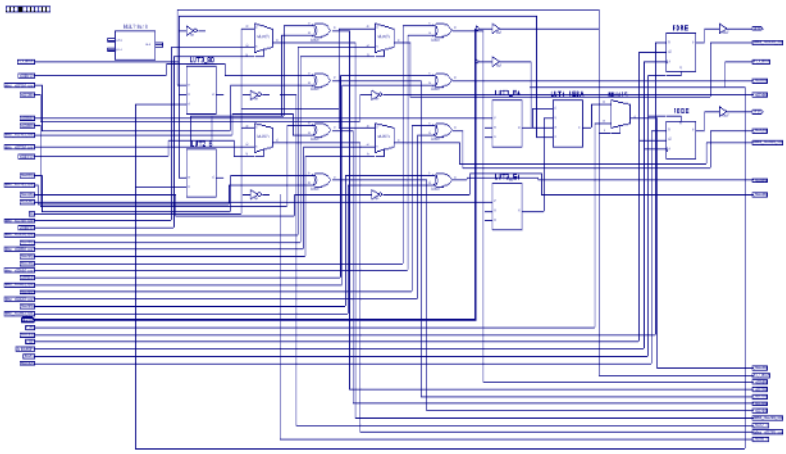


Fig.IV.15. infera- résultats de la synthèse

Toutes ces étapes, étant détaillées pour l'entité précédente, elles sont les même pour les autres entités, dont les résultats sont les suivants :

IV.2.1.2 simulation de l'entité melang

Fig.IV.16: (a) melang- simulation temporelle et fonctionnelle

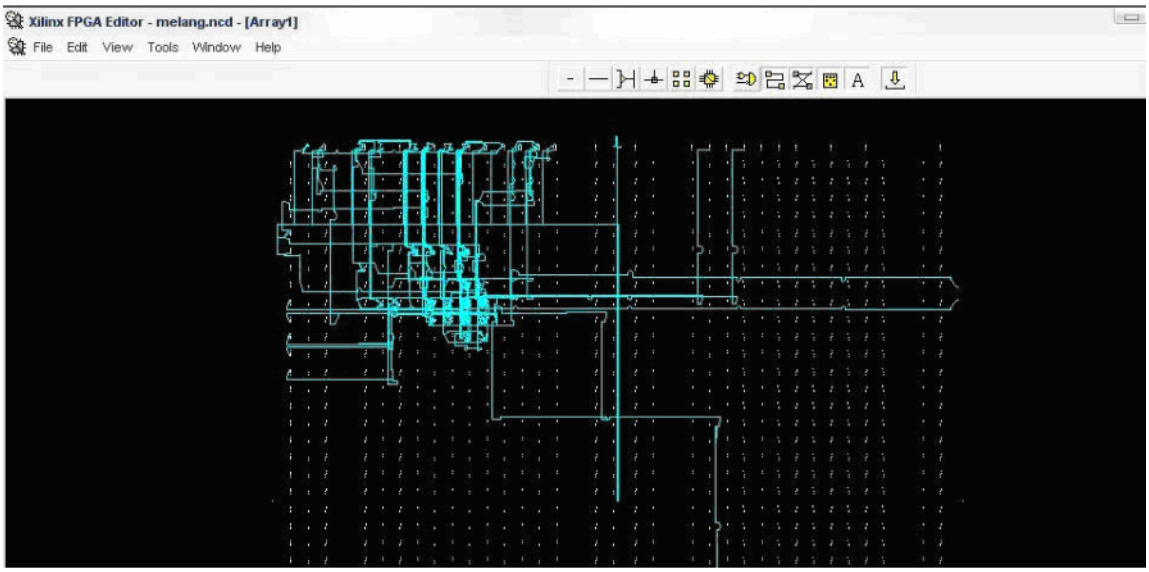


Fig.IV.16: (b) melang- résultat du placement et routage

IV.2.1.3 simulation de l'entité infer1

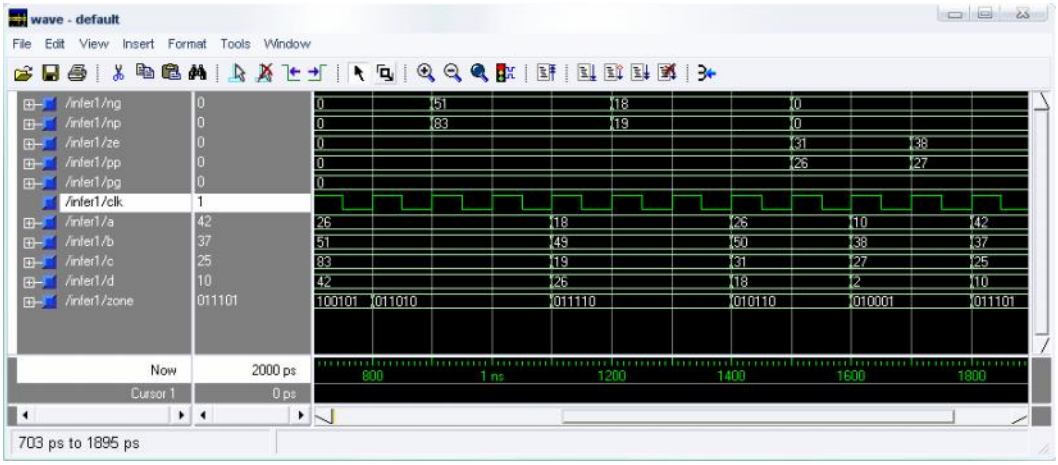


Fig.IV.17 :(a) infer1-simulation temporelle et fonctionnelle

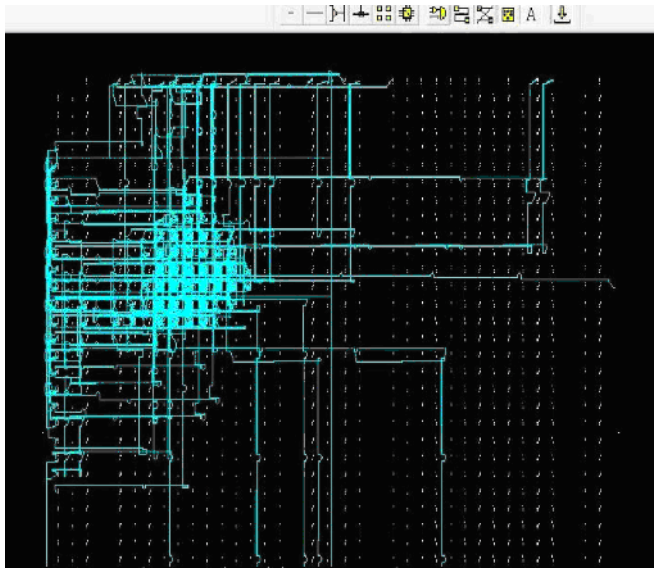


Fig.IV.17 :(b) infer1- résultat du placement et routage

IV.2.1.4 simulation de l'entité comand

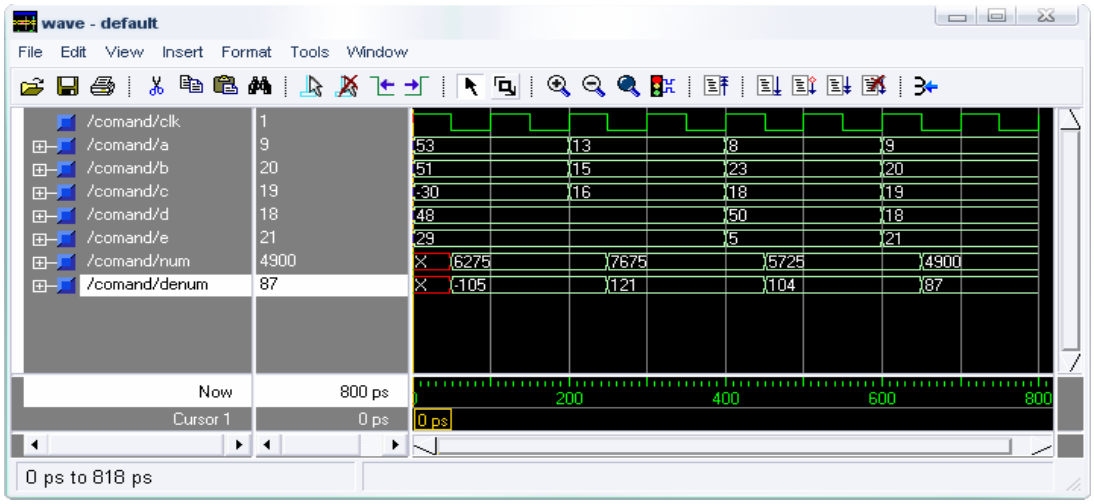


Fig.IV.18 : (a) Comand- simulation temporelle et fonctionnelle

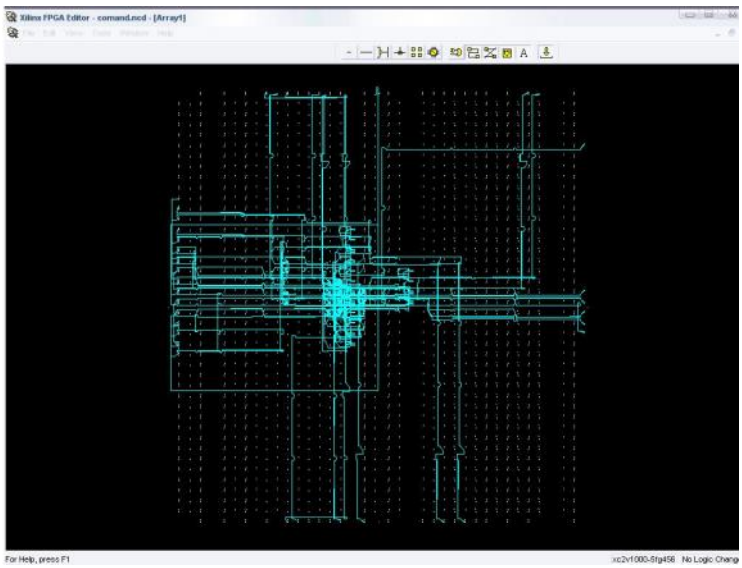


Fig.IV.18: (b) Comand- résultat du placement et routage

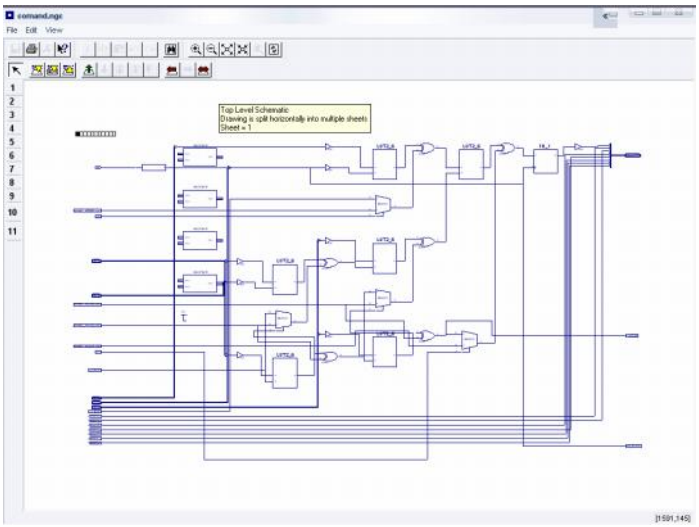


Fig.IV.18 : (c) Comand- résultat de la synthèse

IV.2.1.5 simulation de l'entité div

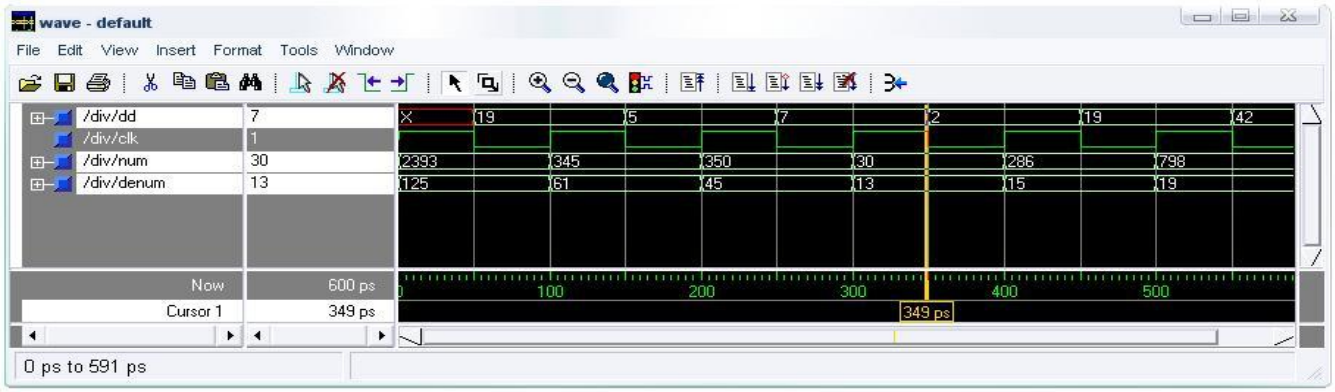


Fig.IV.19 : (a) div -simulation temporelle et fonctionnelle

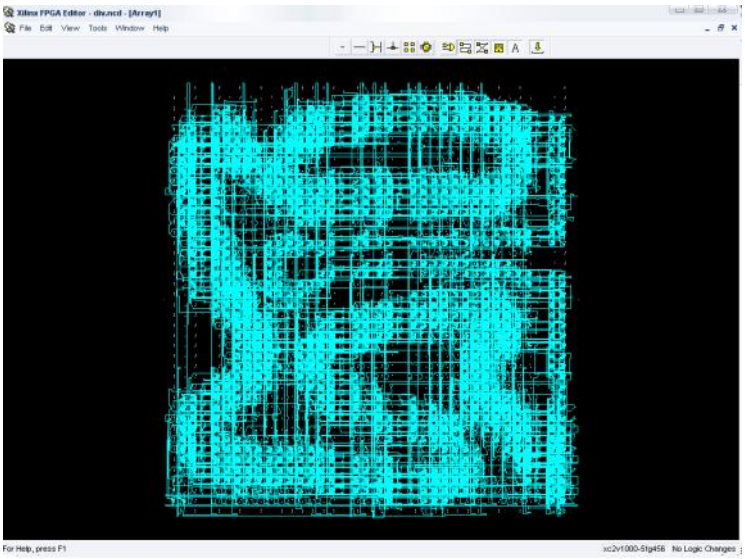


Fig.IV.19 : (b) div - résultat du placement et routage

IV.2.1.6 simulation de l'entité floulog

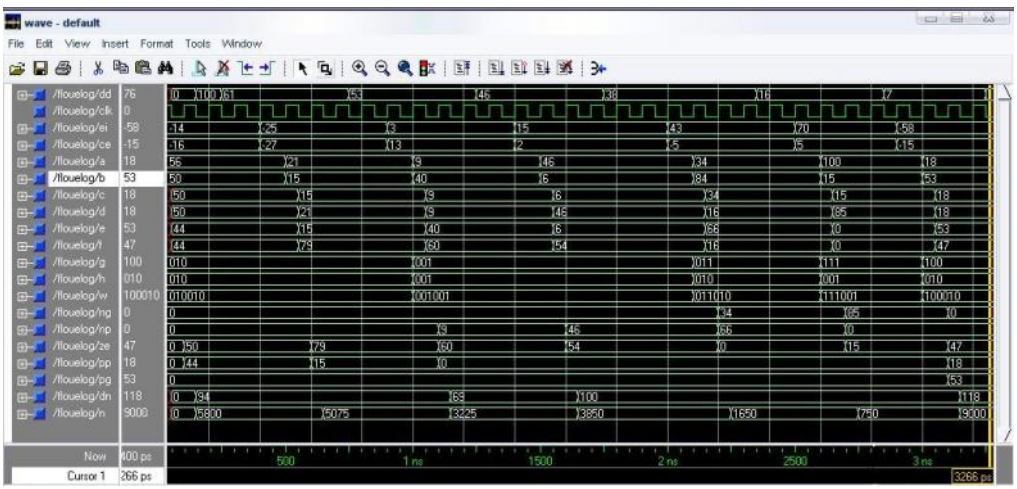


Fig.IV.20 : (a) Floulog -simulation temporelle et fonctionnelle

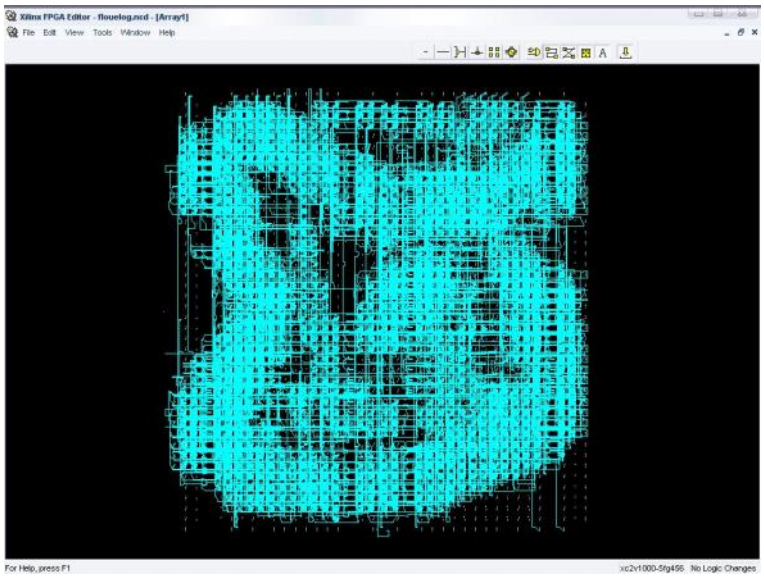


Fig.IV.20 : (b) Floulog- résultat du placement et routage

CONCLUSION

L'industrie moderne a des besoins de plus en plus importants en énergie. Les sources classiques d'énergie, qui sont les sources fossiles tel que le charbon et les hydrocarbures, laissent progressivement la place aux énergies renouvelables. L'augmentation fulgurante du prix du pétrole ces dernières années a en effet contraint les pays développés à investir dans ce type d'énergies telles que l'énergie solaire, éolienne, marémotrice ou géothermique. Ces énergies, en plus d'être inépuisables, représentent un secteur porteur permettant un développement durable tout en préservant l'environnement.

Nous nous sommes axés sur l'énergie solaire qui en plus d'être renouvelable est aussi d'une flexibilité utile, cette énergie est fournie par des générateurs photovoltaïques caractérisés par un point où la puissance est maximale. Ce point se déplace en fonction des conditions atmosphériques, un mécanisme de poursuite s'avère indispensable pour une efficacité meilleure du générateur.

Ce mécanisme de poursuite est le contrôleur MPPT qui recueille des informations du panneau et génère en conséquence un signal MLI pour commander un hacheur intercalé entre le générateur photovoltaïque et la batterie.

Notre travail a consisté en l'implémentation d'un contrôleur MPPT flou. Après avoir introduit les systèmes photovoltaïques, donné un aperçu sur la commande et logique flou, nous avons présenté l'outil, le support et le langage de description.

La commande floue est une technique intelligente et robuste mais trop chère car utilisant des circuits spécialisés, donc nous nous sommes penché sur les circuits FPGA qui offrent les meilleurs performances à des coût très raisonnables.

Concernant les FPGA qui sont des circuits numériques programmables, dynamiquement reconfigurables, caractérisés par une architecture très flexible, développés par des outils de conception utilisant des langages de haut niveau tel que VHDL, que nous avons choisi pour sa simplicité et modularité.

Enfin, nous avons essayé de combiner dans ce travail l'efficacité de la commande floue, la flexibilité des FPGA et la puissance du langage VHDL pour concevoir un contrôleur MPPT flou performent, économiquement et techniquement.

Bibliographie

[1] A.Labouret , P.Cumunel , J.Paul Braun et B.Faraggi , *Cellules solaires, les bases de l'énergie photovoltaïque,* Dunod ,Paris 2001

[2] A. Mermoud , *PVSYST a user friendly software for pv systems simulation,*13th european photovoltaic solair energy ,1991

[3] M.A.S. Masoum, M. Sarvi , S*imulation and construction of a new fuzzy-based maximum power point tracker for photovoltaic applications. AUPEC 02*-10-2002

[4] " Recherche photovoltaïque " .Yves Mafraing. Système Solaire n° 136, mars-avril 2000.

[5] " Crystec Technology Trading GmbH. Matériel pour l'industrie du semi-conducteur". Koyo Thermo Systems.

[6] " Procédés propres et rapides de fabrication des photopiles " par Jean-Claude Muler. Groupe photovoltaïque du laboratoire PHASE - CNRS. 31 novembre 1996.

[7] M. Ishengoma, E. Norum, D*esign and implementation of digitally controlled stand-alone photovoltaic power supply*. Nordic Workshop on power and Industrial Electronics (norpie 2002), Stockholm, Sweden. 12-14 aug. 2002.

[8] A. Hoque and K.A. Wahid, *New mathematical model of a photovoltaic generator (pvg).* Journal of Electrical Engineering vol. ee 28, no. 1, june 2000

[9] A .Hansen, , P. Sørensen., L. Hansen, H. Bindner, , *Models for stand-alone pv system 2000*, Riso National laboratory, Roskilde, Denmark, isbn 87-550-2774-1

[10] T. Robbins, J.Hawkins, B*attery model for over-current protection simulation of dc distribution systems.*

[11] M.A.S. Masoum, M. Sarvi , S*imulation and construction of a new fuzzy-based maximum power point tracker for photovoltaic applications.* ilee papers, 13-jul-2002

[12] Partein, L.D. Solar Cells and Their Applications, Wiley, 1995.

[13] A.Adane and T.Tacfticht" An optimum regulator device for stand-alone photovoltaic systems". World renewable Energy Congress, 2. 231, Reading, U.K

[14] Z.Salameh and D.Taylor "Step-up maximum power point tracker for photovoltaic arrays". Solar Energy, 44(1), 57-61 (1990)

[15] D.B.Snyman and J.H.R.Enslin "An experimental evaluation of MPPT converter topology for PV installations. Renewable Energy, 3, (8), 841-848 (1993)

[16] L.Van de Merwe,"Maximum Power Point Tracking Implementation Strategies" Renewable Energy, Vol.2, n°6, pp.214-217, 1998

[17] H.Sugimob “A New Scheme for Maximum Photovoltaic Power Point Tracking Control “IEEE pp.691-696, 1999.

[18] M.Firas Sharaif, C.Alonso, A.Martinez, A *simple and robust maximum power point control (mppc) for ground photovoltaic generators* , LAAS report 00621, April 2000

[19] F. Delfosse, D*etermination de l’état de charge des batteries d’un véhicule électrique*. PFE, Université de Liège, Année académique 1997-1998

[22] C. TAVERNIER, *Applications Industrielles des PIC, Paris, DUNOD, 2001.*

[23] BIGONOFF, La Programmation des PIC, site Web : ww.abcelectronique.com/bigonoff.

[24] G. CAILLON, *Accumulateurs portables, Techniques de l’ingénieur Traité Electronique*, Volume E2 140.

[25] Bechtel national,*Handbook for battery energy storage photovoltaic power systems*. Inc. San Francisco , California.116p ,1979.

[26] N.Achaibou. Thèse de magistère , *Introduction à l’étude du système de stockage dans un système photovoltaïque* ,Université ,Blida ,2001.

[27] M.A.Dasoyan , I.A.Aguf. *current theory of acid batteries* ,Teckno copy limited, England, 1979

[28] M.Jaquier , *Accumulateur, Teckniques de l’ingenieur* 1981.

Sitographie

WWW.Pvsyst.com

WWW. Photowatt.com

WWW.Firstsolar.com

WWW. Afme.com

WWW.Devloper .com

WWW.Solarix.com

كلمات مفتاحيه:

لوح شمسي, كهر وضوئي, محول مستمر- مستمر, تعقب نقطة الاستطاعة القصوى, تعقب, مراقب المنطق اللاواضح, لغة البرمجة السريعة للدارات المدمجة, الدارات المدمجة القابلة البرمجة,برمجة...

Key words :

Solar panel, photovoltaic, DC-DC converter, MPPT, tracking, fuzzy controller, VHDL, FPGA, implementation...

Mots clés :

Panneau solaire, convertisseur continu- continu, poursuite du point de puissance maximale, poursuite, contrôleur flou, VHDL, circuits programmables FPGA, implémentation…