UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN GÉNIE ÉLECTRIQUE

PAR
BOUBACAR HOUSSEINI

PROTOTYPAGE RAPIDE A BASE DE FPGA D'UN ALGORITHME DE CONTROLE AVANCÉ POUR LE MOTEUR A INDUCTION

DÉCEMBRE 2010

REMERCIEMENTS

Je tiens à exprimer ma profonde gratitude et mes sincères remerciements à mon directeur de recherche, M. Rachid BEGUENANE, professeur au Collège militaire royal du Canada (RMC). Je tiens à le remercier tout particulièrement pour m'avoir fait bénéficié de son savoir, son expérience, son soutien moral et financier, mais aussi de la bonne volonté et de la patience dont il a fait preuve tout au long de la réalisation de ce travail.

Mes chaleureux remerciements vont également à l'endroit de mon co-directeur de recherche M. Adel Omar DAHMANE, de même que M. Ahmed CHERITI, professeurs à l'Université du Québec à Trois-Rivières (UQTR), pour l'aide et le soutien inestimables qu'ils m'ont apporté. J'en suis très reconnaissant.

Je remercie l'ensemble des enseignants de l'Université du Québec à Trois-Rivières (UQTR) et aussi mes collègues étudiants, M. Stéphane SIMARD et M. Jean Gabriel MAILLOUX pour m'avoir fait partager leurs expériences.

Je remercie et dédie ce travail à tous les membres de ma famille, spécialement à mes parents et ma femme pour leurs patiences, leurs encouragements incessants, et leurs supports morals durant mes longues années d'études.

Enfin, je le dédie à mes amis et à toute personne qui m'a aidé de prés ou de loin tout le long de mon cheminement.

TABLE DES MATIERES

INDEX DES FIGURES

Chapitre 2 :

Chapitre 3 :

Chapitre 4 :

Chapitre 5 :

INDEX DES TABLEAUX

Chapitre 3 :

Chapitre 4 :

Chapitre 5 :

LISTE DES SYMBOLES

Symbole	Description	Unité
A_c	Amplitude de la porteuse	(V)
A_r	Amplitude de la tension de référence	(V)
D	Nombre de diodes de bouclage	sans unité
C	Nombre de tension aux bornes des condensateurs	sans unité
C_{em}	Couple électromagnétique du moteur	(N·m)
C_r	Couple de charge	(N·m)
B	Coefficient de frottement	(Nm/rads/s)
f_n	Fréquence nominale	(Hz)
f_s	Fréquence synchronisme	(Hz)
f_v	Frottement visqueux	(Hz)
g	Glissement du moteur asynchrone	sans unité

G_k	Vecteur de commande des gâchettes	sans unité
I	Courant de sortie du moteur asynchrone	(A)
I_r	Courant rotorique	(A)
I_s	Courant statorique	(A)
J	Moment d'inertie du rotor	($Kg.m^2$)
K_i	Gain intégrateur	sans unité
K_p	Gain proportionnel	sans unité
L	Inductance	(H)
L_r	Inductance propre d'une phase du rotor	(H)
L_s	Inductance propre d'une phase du stator	(H)
L_{sr}	Inductance mutuelle	(H)
m	Nombre des niveaux de tension de sortie	sans unité
p	nombre de paires de pôles	sans unité
P	Puissance	(W)
$p_{méca}$	Puissance mécanique	(W)
R	Résistance	(Ω)

R_r	Résistance d'une phase du rotor	(Ω)
R_s	Résistance d'une phase du stator	(Ω)
$Sect$	Numéro du secteur	sans unité
S	Nombre de sources de tensions secondaires continues	sans unité
t	Temps	(s)
V	Tension	(V)
V_d	Composante horizontale de la tension de référence	(V)
V_{dc}	Tensions d'alimentation de l'onduleur	(V)
V_r	Tension rotorique	(V)
V_{ref}	Tension de référence	(V)
V_s	Tension statorique	(V)
V_{max}	Tension maximal	(V)
V_q	Composante verticale de la tension de référence	(V)
V_{tri}	Tension du signal triangulaire	(V)
φ	Flux	(Wb)

Ω_m	vitesse mécanique du moteur	(rad/s)
ω_r	Pulsation rotorique	(s^{-1})
ω_s	Pulsation statorique	(s^{-1})
θ	Angle électrique entre une phase du rotor et la phase correspondante du stator	(rad)

CHAPITRE 1

INTRODUCTION

1.1 PROBLEMATIQUE

Dans de nombreux procédés de fabrication continus, les machines électriques sont omniprésentes et sont fréquemment appelés à travailler en synchronisme, avec des tolérances souvent élevées pour assurer une qualité uniforme du produit et éviter les bris. Les courants harmoniques, définis comme la présence des courants électriques non parfaitement sinusoïdaux dans les réseaux électriques [1], sont l'une des causes principales d'arrêts et de pertes de production dans de nombreuses industries. Ils sont principalement causés par la présence d'une charge électrique non linéaire dans le réseau de transmission, de distribution ou de l'usine et par le démarrage de charges motrices importantes en industrie.

Face à ces problèmes, on utilise les onduleurs multi-niveaux. En effet dans les systèmes de commande à grande puissance, les onduleurs classiques à deux niveaux ne sont plus efficaces. Non seulement ils provoquent un niveau élevé de la dérivée dv/dt résultante de la commutation, mais aussi les interrupteurs ne supportent pas des fortes tensions inverses. Par conséquent, les onduleurs multi-niveaux ont été choisis comme le convertisseur de puissance préféré pour les applications à haute puissance [2], [3]. Parmi les algorithmes de commutation proposées dans la littérature dans ce domaine [4], [5], la modulation de largeur d'impulsions

dite space vector (vecteur spatial) ou SVPWM (Space Vector Pulse-Width Modulation), semble la plus prometteuse, du fait qu'elle offre une grande flexibilité dans l'optimisation au niveau de la conception et est également bien adaptée pour une implémentation numérique. Ainsi elle permet de maximiser la puissance disponible. Depuis plus d'une décennie [6], [7], les domaines d'applications des onduleurs de tension à modulation par largeur d'impulsion (MLI) ne cessent de se multiplier. Cependant le contrôle des onduleurs multi-niveaux, basé sur des algorithmes mathématiquement très élaborés, a souvent été implémenté en utilisant des processeurs DSP (Digital signal processing) en association avec des composants de type ASIC/FPGA (Application-Specific Integrated Circuit / Field-Programmable Gate Array) pour faire face aux contraintes de temps imposés par un contrôle temps réel [10]. Une telle association pour le traitement numérique du signal possède les inconvénients tels qu'une complexité au niveau de la conception des circuits imprimés (PCB), mais aussi une bonne maîtrise de la programmation simultanée : Software (C, Assembleur), et Hardware (VHDL, Verilog). Ces dernières années les récentes évolutions technologiques ont poussé les chercheurs à s'intéresser à un autre moyen d'implémenter des algorithmes de contrôle complexe utilisant un composant intégré unique comme FPGA qui simplifierait ainsi la structure du contrôle ainsi que sa validation[10].

Le sujet de maîtrise proposé, a pour objectif d'explorer les deux techniques de commande des onduleurs multi-niveaux PWM et SVPWM. Ce mémoire apportera une réponse sur l'efficacité de ces techniques de commande dans l'élimination des harmoniques. Quelles sont les limites de ces techniques ? Jusqu'où pourrait-on aller à propos du nombre de niveaux (3, 4, 5, 6, 7, 9, 11 ...) de l'onduleur ? Nous essayerons aussi dans ce mémoire, de remédier au double inconvénient que présente la solution hybride DSP/FPGA, en développant l'algorithme SVPWM en VHDL avec uniquement les outils académiques.

La technique de commande directe du couple (Direct Torque Control ou DTC), du fait de son efficacité et de sa simplicité de mise en œuvre, sera choisie pour le contrôle en boucle fermée de la machine à induction.

1.2 OBJECTIFS

L'objectif principal de ce travail est d'évaluer les performances de l'utilisation des onduleurs multi-niveaux (3, 5, 7, 9, … niveaux) versus les onduleurs ordinaires deux niveaux, pour la commande des machines à induction à base d'un FPGA en utilisant les outils académiques disponibles (Matlab/Simulink, Xilinx Systhem Generator, Xilinx ISE, ModelSim). Les techniques de commande PWM et SVPWM seront utilisées pour la commande des onduleurs multi-niveaux et celle de la DTC pour la machine à induction.

Ainsi les objectifs spécifiques du mémoire sont:

1. Développer les algorithmes de commande PWM, SVPWM multi-niveaux (3, 5, 7, 9 …) et DTC dans l'environnement Matlab/Simulink.
2. Évaluer les performances de l'utilisation des onduleurs multi-niveaux sur l'élimination des harmoniques;
3. Faire une étude comparative des onduleurs multi-niveaux versus les onduleurs ordinaires classiques;
4. Trouver la limite de nombre de niveaux à laquelle les onduleurs multi-niveaux ne sont plus efficaces.
5. Faire l'implémentation de l'algorithme de commande SVPWM de deux niveaux et de niveaux supérieurs sur FPGA;
6. Proposer un prototypage rapide de la commande SVPWM multi-niveaux sur FPGA en utilisant uniquement les outils académiques.

1.3 MÉTHODOLOGIE

Pour faire ce travail nous allons procéder en suivant les cinq étapes suivantes :

1. Après le 1er chapitre de la problématique, objectifs et méthodologie,
2. le chapitre 2 sera consacré à la modélisation des moteurs à induction. Les hypothèses simplificatrices nécessaires à la modélisation seront posées.
3. Le chapitre 3 présentera les onduleurs triphasés. Il sera question ici des principes des deux techniques de commande d'onduleurs PWM et SVPWM. Cette dernière est une étape nécessaire à celle du développement de l'algorithme en VHDL.
4. La commande de la machine asynchrone à boucle fermée sera présentée dans le chapitre 4. Il s'agira de la modélisation et la simulation de la commande DTC sur Simulink à l'aide des blocs de Simulink et SimPowerSystem.
5. Dans le chapitre 5, une étude sera consacrée, dans un premier temps, à la commande des onduleurs multi-niveaux, leurs limites et performances en ce qui concerne l'élimination des harmoniques du courant statorique. Ensuite dans un second temps nous aborderons l'implémentation de cet algorithme en VHDL. Nous modéliserons le système de manière générale, puis le simuler numériquement afin d'évaluer sa performance.
6. Au chapitre 6, une conclusion générale présentera les contributions de ce mémoire, les recommandations et les limites du travail.

CHAPITRE 2

MODELISATION DES MOTEURS A INDUCTION

2.1 INTRODUCTION

La machine à induction, ou machine asynchrone, est une machine électrique à courant alternatif sans connexion entre le stator et le rotor. Les machines possédant un rotor « en cage d'écureuil » sont aussi connues sous le nom de machines à cage ou machines à cage d'écureuil. Le terme asynchrone provient du fait que la vitesse de ces machines n'est pas forcément proportionnelle à la fréquence des courants qui les traversent [1].

La machine asynchrone est le moteur employé dans plus de 80% des applications. On la retrouve aujourd'hui dans de nombreuses applications, notamment dans le transport (métro, trains, propulsion des navires), dans l'industrie (machines-outils), dans l'électroménager. Le choix de son utilisation est dû à son principal avantage qui réside dans l'absence de contacts électriques glissants, ce qui conduit à une structure simple et robuste facile à construire. Elle est utilisée dans une gamme de puissance d'applications de quelques Watts à plusieurs MW.

Pour pouvoir évaluer les performances des onduleurs multi-niveaux, il est nécessaire de connaître dans un premier temps le comportement de la machine asynchrone, utilisant une source d'alimentation triphasée sinusoïdale parfaite.

Dans ce chapitre, nous présenterons le principe de modélisation et de simulation d'une machine électrique : le moteur asynchrone à cage d'écureuil.

La première partie comprendra la modélisation de la machine asynchrone à cage qui permettra d'établir le modèle mathématique de la machine étudiée.

Nous réalisons ensuite le modèle de simulation de la machine asynchrone à cage dans l'environnement MATLAB/SIMULINK. Nous verrons alors :

- la simulation du processus de démarrage à vide (le courant phase, le couple et la vitesse du moteur);
- la simulation du processus de démarrage à vide suivi de l'application d'une charge Cr=30 Nm (le courant, le couple et la vitesse du moteur).
- la simulation du processus de démarrage (jusqu'au régime permanent) suivi du freinage par contre-courant (inversion de deux phases de la tension d'alimentation).

2.2 CONSTITUTION ET PRINCIPE DE FONCTIONNEMENT

2.2.1 Constitution

Le moteur asynchrone est formé :

- **d'un stator :** anneau de tôles encoché à l'intérieur et portant un enroulement triphasé semblable à celui d'un alternateur. Cet enroulement est presque toujours relié à la source et constitue le primaire.
- **d'un rotor :** anneau de tôles rainuré à l'extérieur, concentrique au premier et séparé de lui par un entrefer étroit d'épaisseur constante. Le rotor porte un enroulement polyphasé mis en court-circuit constituant le secondaire. On distingue principalement deux types de structures de rotors :
  - **Un rotor à cage (d'écureuil) :** l'ensemble à l'aspect d'une cage cylindrique dont à chaque bout une couronne métallique est raccordée dans laquelle se trouve un empilement de tôles dont l'axe du moteur passe au centre.
  - **rotor bobiné** : comme pour le rotor à cage il est constitué de tôles empilées, mais des encoches sont pratiquées pour le passage du bobinage qui lui même est raccordé en une extrémité en bout d'arbre sur des bagues l'autre extrémité du bobinage est connecté ensemble (point milieu du couplage étoile). Sur les bagues viennent frotter les balais (aussi appelé les collecteurs) qui sont raccordés au dispositif de démarrage (résistance).

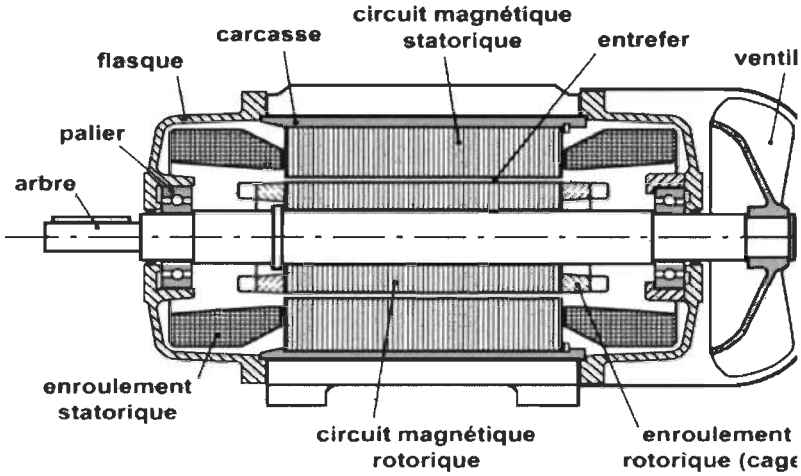


Fig. 2.1 : Morphologie d'une machine asynchrone à cage

2.2.2 Principe de fonctionnement

Les trois enroulements statoriques alimentés par un réseau triphasé équilibré créent dans l'entrefer un champ magnétique tournant à la fréquence de rotation de synchronisme n_s. Les conducteurs du rotor sont soumis à ce champ tournant. Ils sont alors traversés par des courants de Foucault induits d'après la loi de Lenz ("les courants induits s'opposent par leurs effets à la cause qui leur donnent naissance"). Les enroulements du rotor étant en court-circuit, la circulation des courants est alors possible. Les forces de Laplace qui en résultent exercent des moments sur le rotor. Le rotor tourne alors à la fréquence de rotation n. De par son principe, la fréquence de rotation du rotor est inférieure à la fréquence de synchronisme f_s : f < f_s.

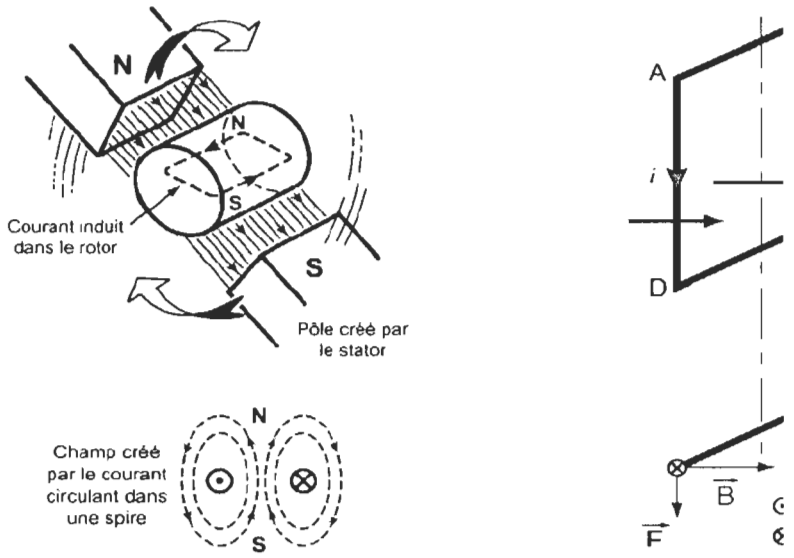


Fig. 2.2: Schéma de principe de fonctionnement

Le glissement : On définit le glissement par l'expression : $g = \frac{\Omega_s - \Omega}{\Omega_s}$

On l'exprime en %, ordre de grandeur 3 à 5 % (pour le nominal)

Le moteur asynchrone est donc caractérisé par :

- la présence d'un seul bobinage polyphasé alimenté par une source extérieure au stator ;
- la présence d'un « bobinage » massif en court-circuit, au rotor.

2.3 MODELISATION DU MOTEUR ASYNCHRONE A CAGE

2.3.1 Hypothèses

Elle est basée sur de nombreuses hypothèses, parmi ces dernières nous citons [11] :

- Entrefer parfaitement lisse ;
- Pertes fer négligeables ;
- Saturation dans le circuit magnétique négligeable ;
- Harmoniques d'espaces négligeables;

- Les enroulements du stator et du rotor sont à répartition sinusoïdale de sorte que les inductances mutuelles entre le stator et le rotor sont des fonctions sinusoïdales de la position mécanique du rotor par rapport au stator.

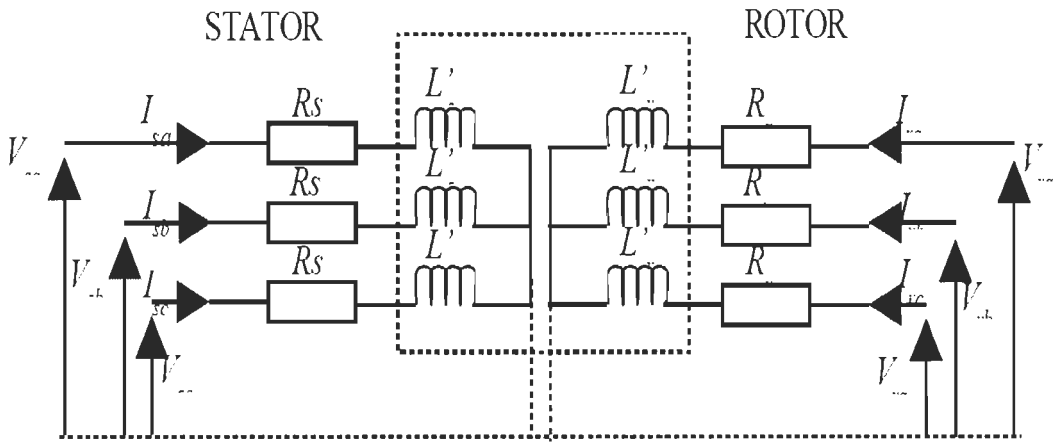


Fig. 2.3: Schéma électrique d'un moteur asynchrone

2.3.2 Equations électriques et mécaniques

A partir du schéma électrique du moteur asynchrone ci-dessus, nous pouvons déduire les équations suivantes :

$$V = R.I + \frac{d\varphi}{dt} \qquad \text{et} \qquad \varphi = \text{L.I} \tag{1.1}$$

Pour le statore :

$$V_s = \begin{bmatrix} V_{sa} \\ V_{sb} \\ V_{sc} \end{bmatrix} = \begin{bmatrix} R_s & 0 & 0 \\ 0 & R_s & 0 \\ 0 & 0 & R_s \end{bmatrix} . \begin{bmatrix} I_{sa} \\ I_{sb} \\ I_{sc} \end{bmatrix} + \frac{d}{dt} . \begin{bmatrix} \varphi_{sa} \\ \varphi_{sa} \\ \varphi_{sa} \end{bmatrix} \tag{1.2}$$

$$L_s = \begin{bmatrix} L_s & M_s & M_s \\ M_s & L_s & M_s \\ M_s & M_s & L_s \end{bmatrix} \tag{1.3}$$

Pour le rotor:

$$V_r = \begin{bmatrix} V_{ra} \\ V_{rb} \\ V_{rc} \end{bmatrix} = \begin{bmatrix} R_r & 0 & 0 \\ 0 & R_r & 0 \\ 0 & 0 & R_r \end{bmatrix} . \begin{bmatrix} I_{ra} \\ I_{rb} \\ I_{rc} \end{bmatrix} + \frac{d}{dt} . \begin{bmatrix} \varphi_{ra} \\ \varphi_{ra} \\ \varphi_{ra} \end{bmatrix} \quad (1.4)$$

$$L_r = \begin{bmatrix} L_r & M_r & M_r \\ M_r & L_r & M_r \\ M_r & M_r & L_r \end{bmatrix} \quad (1.5)$$

Couplage Stator/Rotor:

$$M_{sr}(\theta) = M_{rs}^T(\theta) = \begin{bmatrix} \cos\theta & \cos(\theta - \frac{2\pi}{3}) & \cos(\theta + \frac{2\pi}{3}) \\ \cos(\theta + \frac{2\pi}{3}) & \cos\theta & \cos(\theta - \frac{2\pi}{3}) \\ \cos(\theta - \frac{2\pi}{3}) & \cos(\theta + \frac{2\pi}{3}) & \cos\theta \end{bmatrix} \quad (1.6)$$

$$M_{rs}(\theta) = M_{sr}^T(\theta) = M_{sr}(-\theta)$$

θ désigne l'angle électrique entre une phase du rotor et la phase correspondante du stator:

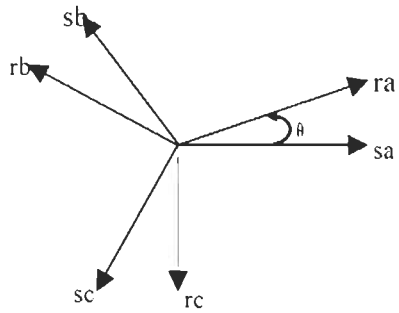


Fig. 2.4: Schéma angle électrique du stateur et roteur

Bilan de puissance :

$$P = I^T.R.I + \frac{d}{dt}(\frac{1}{2}.I^T.L.I) + \frac{1}{2}.I^T.\frac{dL}{dt}.I \quad (1.7)$$

La puissance instantanée se décompose en trois termes:

- Puissance dissipée par effet Joule.

- Variation totale d'énergie électromagnétique.

- Puissance transformée en puissance mécanique ($p_{méca}$).

Puissance mécanique :

$$p_{méca} = C_{em}.\Omega_m = \frac{1}{2}.I^T.\frac{dL}{dt}.I\ .p.\Omega_m \quad (1.8)$$

Couple moteur :

$$C_{em} = p.\frac{1}{2}.I^T.\frac{dL}{d\theta}.I \quad (1.9)$$

Equation mécanique :

$$J.\frac{d\Omega_m}{dt} + f_v.\Omega_m = C_{em} - C_\Gamma \quad (1.10)$$

2.3.3 Transformation de Concordia

La machine triphasée peut être transformée en une machine biphasée équivalente à l'aide de la transformation de Concordia :

Un vecteur x de grandeur triphasée peut être décrit par les signaux sur chacune des trois phases (abc) : $X_{abc} = [X_a\ X_b\ X_c]^T$. Un changement de repère approprié (abc) à un repère (αβh) qui permet de réduire la complexité du système sous certaines hypothèses. Supposant que l'axe h soit orienté suivant la composante homopolaire du système triphasé (c'est à dire la somme des trois signaux de phase). Lorsque le système triphasé est équilibré, cette composante est nulle, ce qui permet de réduire le système triphasé à un système biphasé (axes αβ). Sans prise en compte de la composante homopolaire (hypothèse : $X_a + X_b + X_c = 0$) :

$$X_{\alpha\beta} = K_{23}.X_{abc} \quad \text{avec} \quad K_{23} = \sqrt{\frac{2}{3}}.\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \quad (1.11)$$

Avec prise en compte de la composante homopolaire :

$$X_{\alpha\beta\,h} = K_{33}.X_{abc} \quad \text{avec} \quad K_{33} = \sqrt{\frac{2}{3}}.\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \quad (1.12)$$

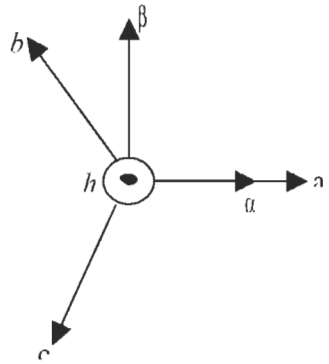


Fig. 2. 5: Transformation de Concordia

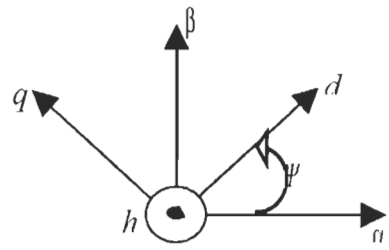


Fig. 2.6: Rotation

Un repère (dq) quelconque (ou (dqh)) s'obtient par rotation d'un angle ψ du repère (αβ) (ou (αβh)). d signifie « direct » et q signifie « quadrature ».

Sans prise en compte de la composante homopolaire :

$$X_{dq} = R(\psi).X_{\alpha\beta} \quad \text{avec} \quad R(\psi) = \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{bmatrix} \tag{1.13}$$

Avec prise en compte de la composante homopolaire :

$$X_{dqh} = \tilde{R}(\psi).X_{\alpha\beta h} \quad \text{avec} \quad \tilde{R}(\psi) = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.14}$$

Les changements de repère de ce paragraphe s'appliquent à n'importe quel vecteur de grandeur triphasée. En particulier, ils s'appliquent aussi aux grandeurs triphasées relatives au stator qu'à celles relatives au rotor d'une machine asynchrone.

2.3.4 Transformation de Park (rotor fictif équivalent fixe)

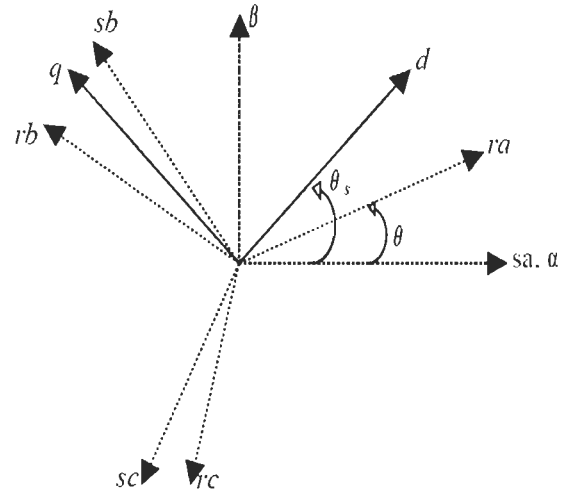


Fig. 2.7: Repère triphasé fixe par rapport au stateur (S_a, S_b, S_c), repère (dq) formant un angle θ_s quelconque par rapport au stateur

Pour simplifier d'avantage les expressions électrique et surtout pour que M ne dépende plus de q, Park a proposé de remplacer le rotor tournant par un rotor équivalent « fixe ». Cela revient à exprimer les grandeurs rotoriques dans le repère (dq) défini par rapport au stator.

Transformation de Park :

Sans prise en compte de la composante homopolaire (hypothèse : MAS équilibrée) :

$$X_{dq}=\begin{bmatrix} T(\theta_s) & 0 \\ 0 & T(\theta_s-\theta) \end{bmatrix}.X_{abc} \text{ avec : } X_{dq}=\begin{bmatrix} X_{s,dq} \\ X_{r,dq} \end{bmatrix},\ X_{abc}=\begin{bmatrix} X_{s,abc} \\ X_{r,abc} \end{bmatrix} \text{ et } T(\psi)=R(\psi).K_{23} \qquad (1.15)$$

Avec prise en compte de la composante homopolaire :

$$X_{dq}=\begin{bmatrix} \tilde{T}(\theta_s) & 0 \\ 0 & \tilde{T}(\theta_s-\theta) \end{bmatrix}.X_{abc} \text{ avec : } X_{dqh}=\begin{bmatrix} X_{s,dqh} \\ X_{r,dqh} \end{bmatrix},\ X_{abc}=\begin{bmatrix} X_{s,abc} \\ X_{r,abc} \end{bmatrix} \text{ et } \tilde{T}(\psi)=\tilde{R}(\psi).K_{33} \qquad (1.16)$$

Remarque 1 :

$\tilde{T}$ est orthonormé ($\tilde{T}$ est une matrice unité d'ordre 3 : $\tilde{T}\,\tilde{T}^{T} = I_3$) ce qui permet de conserver la puissance instantanée et d'opérer la même transformation sur les tensions que sur les courants. Les axes du repère (dqh) sont orthogonaux.

Remarque 2 :

$$T(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}. \tilde{T}(\psi) \tag{1.17}$$

$\omega_s = \frac{d\theta_s}{dt}$, représente la pulsation du repère (dq) par rapport au stator.

Expression des tensions statoriques et rotoriques dans un repère (dq) :

$$V_{ds} = R_s.I_{ds} + \frac{d\varphi_{ds}}{dt} \tag{1.18}$$

$$V_{qs} = R_s.I_{qs} + \frac{d\varphi_{qs}}{dt} \tag{1.19}$$

$$0 = R_r.I_{ds} + \frac{d\varphi_{ds}}{dt} + \omega_r.\varphi_{qr} \tag{1.20}$$

$$0 = R_r.I_{qs} + \frac{d\varphi_{ds}}{dt} + \omega_r.\varphi_{dr} \tag{1.21}$$

Le rotor du moteur asynchrone à cage étant fermé sur lui même (court-circuité), on prend V_{qs} et V_{qr} égales à zéro.

2.3.5 Expression des flux statoriques et rotoriques dans un repère (dq) :

Les flux couplés statoriques et rotoriques sont obtenus à partir des inductances propres et mutuelles :

$$\varphi_{ds} = L_s.I_{ds} + L_{sr}.I_{dr} \quad (1.22)$$

$$\varphi_{ds} = L_s.I_{qs} + L_{sr}.I_{qr}$$

$$\varphi_{dr} = L_r.I_{dr} + L_{sr}.I_{ds}$$

$$\varphi_{qr} = L_r.I_{qr} + L_{sr}.I_{qr}$$

Expression du couple moteur à partir de grandeurs électriques dans un repère (dq) :

$$C_{em} = \frac{3}{2}.p.[\varphi_{ds}.I_{qs} - \varphi_{qs}.I_{ds}] = \frac{3}{2}.p.L_{sr}.(I_{qs}.I_{dr} - I_{ds}.I_{qr}) \quad (1.23)$$

L'équation du mouvement, reliant les parties électriques et mécaniques s'écrit :

$$J.\frac{d\Omega}{dt} + f_v.\Omega_m = C_{em} - C_\Gamma \quad (1.24)$$

Les tensions biphasées dans l'équation (2) sont obtenues comme suit :

$$V_{ds} = \sqrt{\frac{2}{3}}\,(V_{as} - \frac{1}{2}.V_{bs} - \frac{1}{2}.V_{cs}) \quad (1.25)$$

$$V_{qs} = \sqrt{\frac{2}{3}}\,(\frac{\sqrt{3}}{2}.V_{bs} - \frac{\sqrt{3}}{2}.V_{cs}) \quad (1.26)$$

Les équations de transformation inverse des courants statoriques sont :

$$I_{as} = \sqrt{\frac{2}{3}}\,.I_{ds} \quad (1.27)$$

$$I_{bs} = \sqrt{\frac{2}{3}}\,(-\frac{1}{2}.I_{ds} + \frac{\sqrt{3}}{2}.I_{qs}) \quad (1.28)$$

$$I_{cs} = \sqrt{\frac{2}{3}}\,(-\frac{1}{2}.I_{ds} - \frac{\sqrt{3}}{2}.I_{qs}) \quad (1.29)$$

2.4 MODELE SIMULINK DE LA MACHINE ASYNCHRONE A CAGE

2.4.1 Modèle SIMULINK

La figure 2.8 représente le modèle SIMULINK du moteur asynchrone décrit par les équations précédentes. Il est constitué de sept blocs principaux, qui sont :

- La source d'alimentation triphasée,
- Le bloc de transformation abc/dq,
- Le bloc calculant les courants statoriques et rotoriques dans le plan dq,
- Le bloc calculant le couple électromagnétique du moteur,
- Le bloc calculant le couple résistant de la charge,
- Le bloc calculant la vitesse du moteur,
- Et le bloc calculant les trois signaux triphasés du courant statorique.

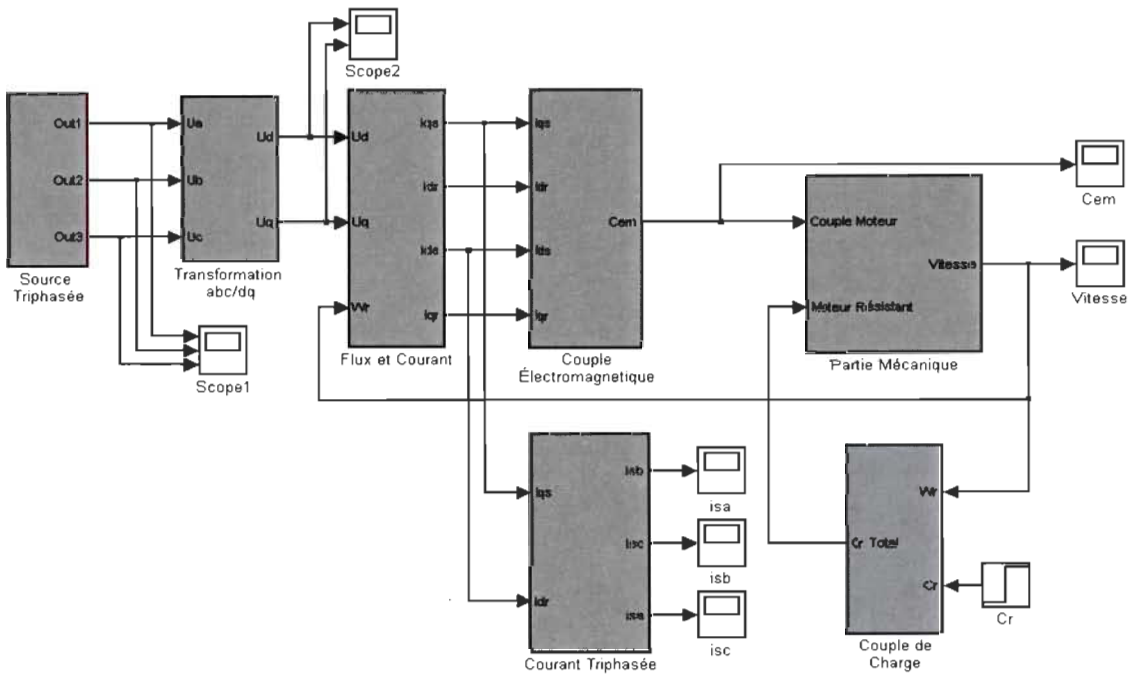
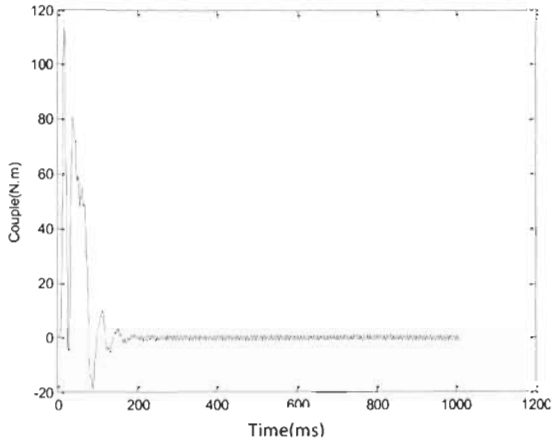


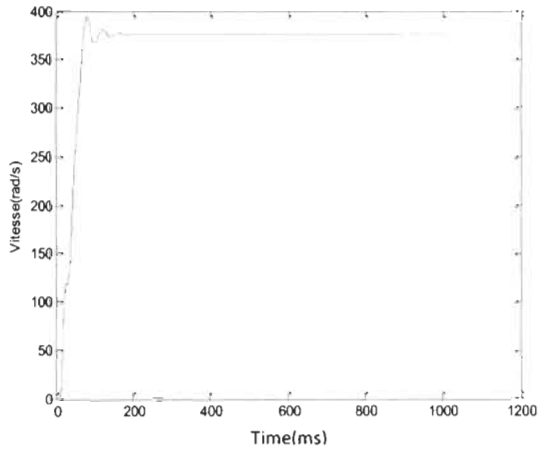
Fig. 2.8: Modèle Simulink du moteur asynchrone

2.4.2 Résultats de simulation au démarrage à vide et en charge

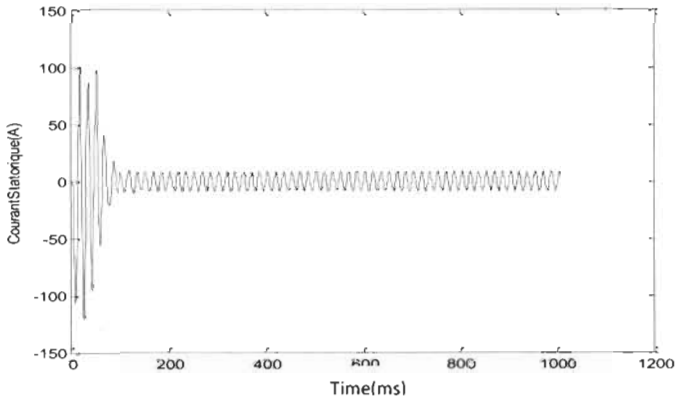
- **Démarrage à vide : courants de phase, couple et vitesse du moteur**



a) Comportement du couple



b) Comportement de la vitesse

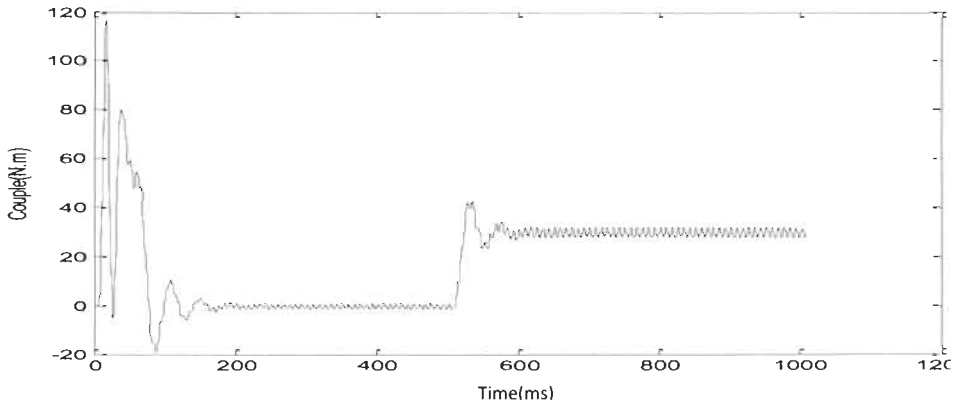


c) Comportement du courant statorique

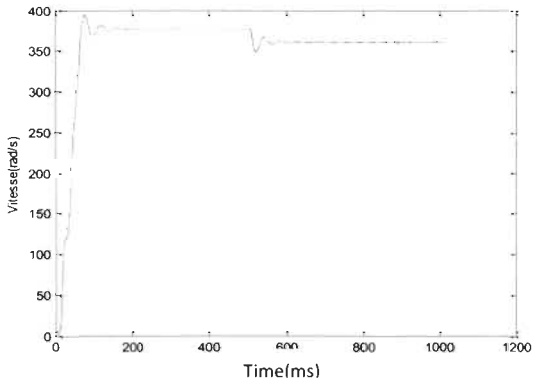
Fig. 2.9: Résultats de la simulation du processus de démarrage à vide

Nous remarquons qu'au démarrage à vide la machine demande un fort courant statorique qui dépasse 80A, Figure 2.9.c. Le moteur continu à accélérer jusqu'à ce que sa vitesse devient constante, et égale à sa valeur nominale (Figure 2.9.b).

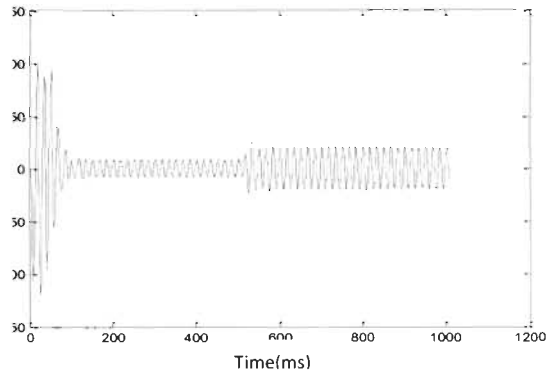
- **Démarrage à vide suivi de l'application d'une charge Cr=30 Nm : courants de phase couple et vitesse du moteur.**



a) Comportement du couple



b) Comportement de la vitesse



c) Comportement du courant statorique

Fig. 2.10: Résultats de la simulation du processus de démarrage à vide du moteur asynchrone suivi de l'application d'une charge

Les figures ci-dessus montrent les résultats de la simulation du processus de démarrage à vide du moteur asynchrone suivi de l'application d'une charge de 30 N.m à l'instant 0.5 ms.

On remarque que :

- Le courant statorique de la phase « a » (Figure 2.10.c), répond parfaitement au changement de la consigne du couple du 0 à 30 N.m à l'instant 0.5 ms. A vide I_{sa}= 10A, à charge I_{sa}= 20A.
- Aprèes l'application de la charge à l'instant t=0.5 ms, la vitesse du moteur diminue. C'est le principe du moteur asynchrone quand il fonctionne en boucle ouverte à cause du glissement (Figure 2.10.b).

2.4.3 Processus de démarrage suivi du freinage par contre-courant et rédémarrage en sens inverse

Modifions le modèle afin d'effectuer la simulation du processus de démarrage (jusqu'au régime permanent) suivi du freinage par contre-courant (inversion de deux phases de la tension d'alimentation).

Le contenu du bloc « Source triphasée » est représenté à la figure 2.11.

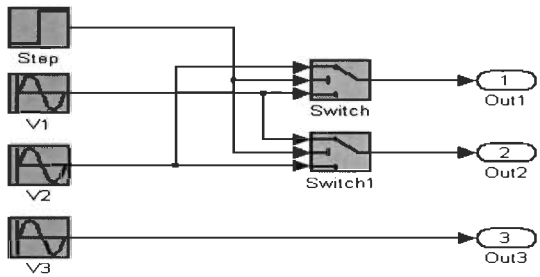
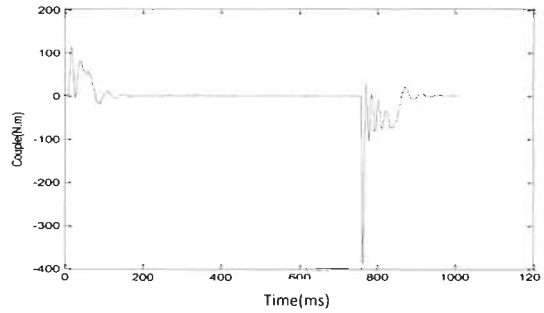


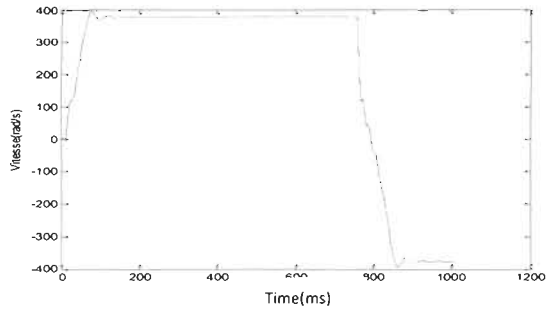
Fig. 2.11: Bloc de la source triphasée

Lors du freinage, il y a inversion de deux phases de la tension d'alimentation : le moteur est alimenté par un champ statorique inverse. Les pointes de courant sont très importantes et il est conseillé d'insérer un jeu de résistances pour limiter ce courant. Les lignes de tensions V_1, V_2et V_3 doivent être ouvertes dès l'arrêt du moteur, pour éviter un redémarrage en sens inverse : il est donc nécessaire de prévoir un capteur détectant l'absence de rotation (capteur centrifuge).

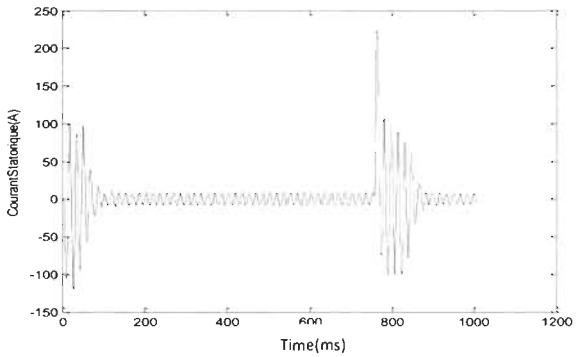
- **Démarrage à vide : courant, couple et vitesse du moteur**



a) Comportement du couple



b) Comportement de la vitesse



c) Comportement du courant statorique

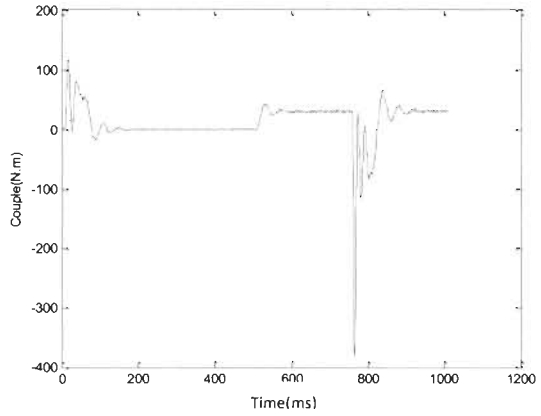
❖ **Fig. 2.12: Résultats de la simulation du processus de démarrage à vide suivi du freinage par contre-courant et rédémarrage en sens inverse.**

Nous remarquons que :

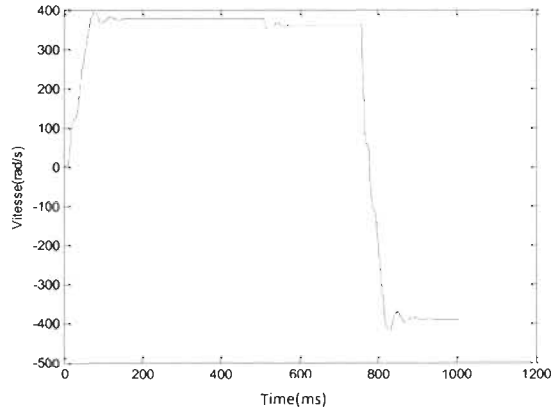
- Entre t=0.75 ms et t=1 ms, avec l'inversement de deux phases de la tension d'alimentation, à l' instant t=0.75 ms, le moteur décélère (freinage par contre-courant) figure 2.13.b.
- Une perturbation est apportée au couple et courant statorique entre t=0.75 ms et t=0.85 ms, Figure 2.13.a,c.
- Après t=0.85 ms le couple est toujours maintenu constant à sa valeur nominale. Même remarque que pour le courant.

- **Démarrage à charge : courant, couple et vitesse du moteur**

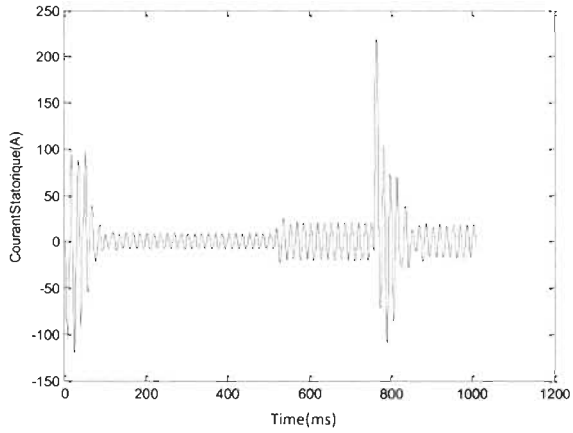
La simulation est effectuée sur une période de 1s. Elle se fait de façon à envoyer une consigne de couple Tl =30N.m pour t (0.5, 1 ms) et d'inverser les deux phases de la tension d'alimentation à l'instant t=0.75 ms. Les résultats de simulation pour la vitesse, le couple, le courant statorique sont montrés à figure 2.13.



a) Comportement du couple



b) Comportement de la vitesse



c) Comportement du courant statorique

Fig. 2.13: Résultats de la simulation du processus de démarrage à charge suivi du freinage par contre-courant et rédémarrage en sens inverse.

Nous faisons la même observation que le cas précédant sauf au niveau du changement de la consigne du couple à l'instant 0.5 ms (du 0 à 30 N.m) où on conatate que le courant statorique répond également à ce changement en passant de I_sa= 10A (à vide) à I_sa= 20A (en charge).

3.1 CONCLUSION

Dans ce chapitre, nous avons présenté la modélisation et la simulation d'un moteur asynchrone à cage. Le processus de démarrage du moteur, suivi de l'application d'une charge entraînée a été modélisé et simulé. Les résultats obtenus démontrent la justesse du modèle développé. Cette étape est nécessaire dans le cadre de ce travail car les mêmes tests réalisés ici seront faits par la suite dans le chapitre 5 en utilisant cette fois ci les onduleurs multi-niveaux au lieu d'une source d'alimentation parfaite, avec le modèle de machine asynchrone développé dans ce chapitre. Ce chapitre nous a permis de mieux voir le comportement de la machine asynchrone dans différentes situations avant son utilisation dans un système de commande à boucle fermée ou avec une alimentation constituée d'onduleur multi-niveaux.

CHAPITRE 3

COMMANDE SVPWM D'UN ONDULEUR

3.1 INTRODUCTION

Dans le domaine de la commande des machines asynchrones, de puissance inférieure à 500 kW, une structure de puissance fait maintenant l'unanimité : l'onduleur de tension associé à un redresseur non contrôlé et un filtre capacitif [12]. Cette structure s'est imposée avec le temps grâce aux progrès en coûts et en performances accomplis par les interrupteurs de puissance. La technique de commande la plus utilisée pour la commande des onduleurs de tension est la modulation de largeur d'impulsions. Elle consiste à commander les interrupteurs de manière à délivrer à la machine une suite d'impulsions d'amplitude fixe, positives ou négatives et modulées en largeur. Il existe de très nombreuses possibilités de réalisations, par exemple :

- la technique analogique utilisée sur les réalisations industrielles les plus anciennes. Elle consiste à générer :
    - une onde sinusoïdale de référence par phase dont l'amplitude et la fréquence représentent la tension de sortie

- une onde de modulation de fréquence élevée de forme triangulaire. Les interrupteurs de puissance sont commandés aux instants d'intersection de ces deux ondes, instants déterminés par des comparateurs.

- la technique numérique : l'apparition des microprocesseurs a permis de transposer le principe décrit précédemment en technique numérique.
- la modulation, entièrement réalisée par le microprocesseur, consiste à commander les interrupteurs avec un motif de base auquel on superpose une modulation à haute fréquence réalisant la variation de tension.

Dans ce chapitre, nous présenterons la partie puissance de la structure de commande des machines asynchrones qui constitue l'alimentation et nous parlerons également du principe de fonctionnement de l'onduleur triphasé à deux niveaux, avec une évaluation par simulation des performances de ce type d'onduleur associé à une machine asynchrone en boucle ouverte selon les deux techniques MLI et SVPWM. Ce qui nous amènera par la fin à une comparaison des deux techniques.

3.2 ALIMENTATION D'UNE MACHINE ASYNCHRONE

L'alimentation d'une machine sert à fournir ou à récupérer de l'énergie électrique à la machine. Comme mentionné précédemment, elle est généralement composée d'un redresseur triphasé, d'un hacheur de freinage et d'un onduleur triphasé.

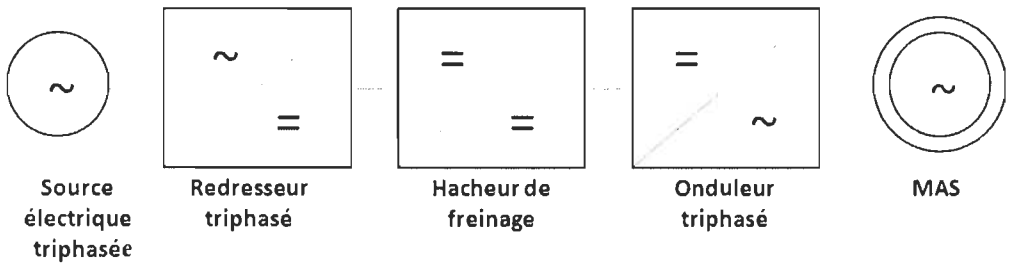


Fig. 3.1: Schéma de la structure d'alimentation

3.2.1 Redresseur triphasé

Il sert à convertir la tension triphasée sinusoïdale du réseau en tension continue pour l'alimentation de l'onduleur.

3.2.2 Hacheur de freinage

Le bloc hacheur de freinage sert à absorber l'énergie produite par la machine au moment de la décélération. Le freinage dynamique intervient lorsque la tension d'alimention atteint la limite supérieure de la bande d'hystérésis, et il s'arrête lorsqu'elle atteint la limite inférieure. La logique du hacheur hystérésis est indiquée ci-dessous:

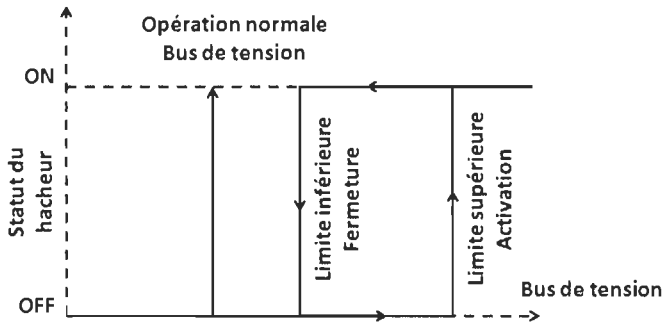


Fig. 3.2: Schéma de principe d'un hacheur hystérésis

3.2.3 Onduleur triphasé

Un onduleur est un dispositif d'électronique de puissance permettant de délivrer des tensions et des courants alternatifs à partir d'une source d'énergie électrique continue. C'est la fonction inverse d'un redresseur. L'onduleur est un convertisseur de type continu/alternatif.

Principe :

Les onduleurs sont des structures en pont constituées le plus souvent d'interrupteurs électroniques tels que les IGBT, des transistors de puissance ou thyristors. Par un jeu de commutations commandées de manière appropriée (généralement une modulation de largeur d'impulsion), on module la source afin d'obtenir un signal alternatif de fréquence désirée.

Deux types d'onduleurs sont utilisés. On retrouve l'onduleur de tension et l'onduleur de courant. On distingue habituellement :

- L'onduleur autonome qui délivre une tension avec une fréquence soit fixe, soit ajustable par l'utilisateur. Il n'a pas besoin de réseau électrique pour fonctionner. Par exemple un convertisseur de voyage que l'on branche sur la prise allume-cigare d'une voiture pour convertir le 12 V continu en 230 V alternatif, 50 Hz.
- L'onduleur non autonome : c'est le nom donné au montage redresseur tout thyristors (pont de Graëtz) qui, en commutation naturelle assistée par le réseau auquel il est raccordé, permet un fonctionnement en onduleur (par exemple par récupération de l'énergie lors des périodes de freinage dans les entraînements à moteurs électriques).

  À la base du développement des entraînements statiques à vitesse variable pour moteurs à courant continu et alternatif, cycloconvertisseurs, onduleurs de courant pour machines synchrones et asynchrones, jusqu'à des puissances de plusieurs MW, ce type de montage est progressivement supplanté, au profit de convertisseurs à IGBT ou GTO. Pour notre étude, il ne sera question que des onduleurs de tension non autonomes.

3.2.4 Topologies d'onduleurs

Par définition, l'onduleur de tension multi-niveaux possède trois ou plusieurs niveaux. L'objectif de cette partie est de donner une vue générale des trois topologies de base des onduleurs multi-niveaux : la topologie à diode de bouclage, la topologie au condensateur flotteur et la topologie en cascade. Bien que cette partie ne soit en aucun cas une vue complète sur les topologies des onduleurs multi-niveaux mais elle couvre celles qui polarisent l'attention des chercheurs.

- **Onduleur de tension à diode de bouclage**

La première topologie la plus pratique d'onduleur de tension multi-niveaux est le NPC (Neutral-Point-Clamped). Elle a été proposée, la première fois en 1981, par Nabae et Al. [7]. L'onduleur NPC à trois niveaux est donné par la figure 3.3.

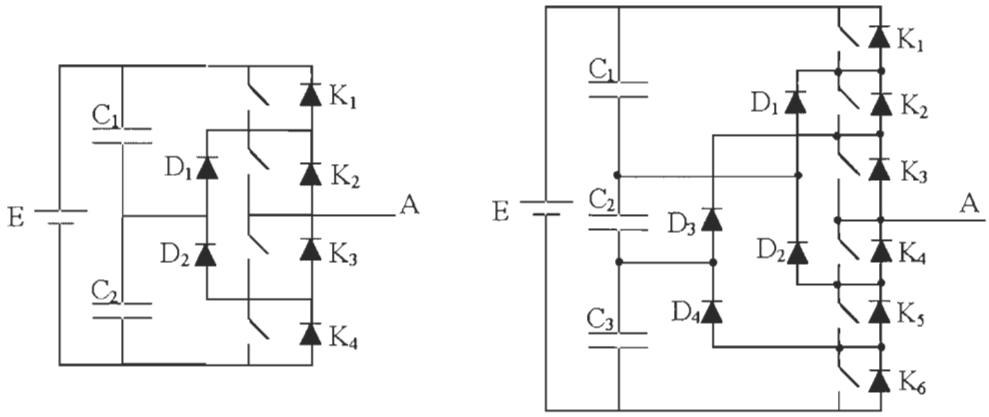


Fig. 3.3: Onduleurs à trois et à quatre niveaux (phase A)

- **Onduleur de tension à condensateur flotteur**

La topologie de l'onduleur multi-niveaux à condensateur flotteur (flying capacitor multilevel inverter), donnée par la figure 3.3, a été proposée en 1992 [13]. Elle est considérée comme l'alternative la plus sérieuse à la topologie de l'onduleur NPC. L'avantage de cette topologie est qu'elle élimine le problème des diodes de bouclage présents dans les topologies des onduleurs NPC multi-niveaux. En plus, cette topologie limite naturellement les contraintes en tension imposées aux composants de puissance (faible valeur de dv/dt aux bornes des composants) et introduit des états de commutation additionnelles qui peuvent être utilisés pour aider à maintenir l'équilibre des charges dans les condensateurs. La topologie de l'onduleur à condensateur flotteur a assez d'états de commutation pour contrôler l'équilibre des charges dans chaque bras d'onduleur ayant n'importe quel nombre de niveaux, ce qui n'est pas le cas dans l'onduleur NPC.

Actuellement il semble que cette topologie a quelques inconvénients. Néanmoins, quelques points faibles doivent toujours être explorés :

- le contrôleur de la charge du condensateur ajoute la complexité au contrôle du circuit entier ;
- la topologie de l'onduleur à condensateur flotteur à multi-niveaux peut exiger plus de condensateurs que la topologie de l'onduleur NPC. De plus, il est évident que des courants de grandes valeurs efficaces circuleront à travers ces condensateurs ;
- il y a un potentiel de résonance parasite entre les condensateurs découplés.

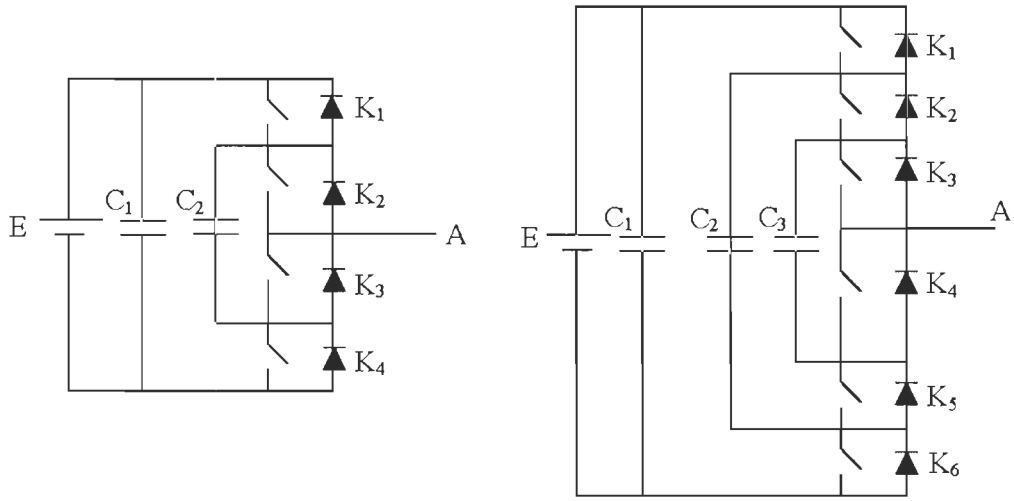


Fig. 3.4: Onduleurs à condensateurs flotteurs à trois et à quatre niveaux (phase A)

- **Onduleur de tension en cascade**

Une des premières applications des connexions en série des topologies des convertisseurs monophasés en pont était pour la stabilisation de plasma en 1988 [14]. Cette approche modulaire a été étendue pour inclure aussi les systèmes triphasés. Sans conteste, les complications et le coût des sources isolées pour chaque pont n'est pas un inconvénient sérieux parce qu'il est compensé par les avantages de la construction modulaire. L'avantage principal de cette approche est que la topologie de ce type d'onduleur facilite la maintenance

en plus elle permet de donner une façon très pratique pour augmenter le nombre de niveaux dans le système [3], [15]. La figure 3.5 représente un onduleur monophasé en cascade à cinq niveaux.

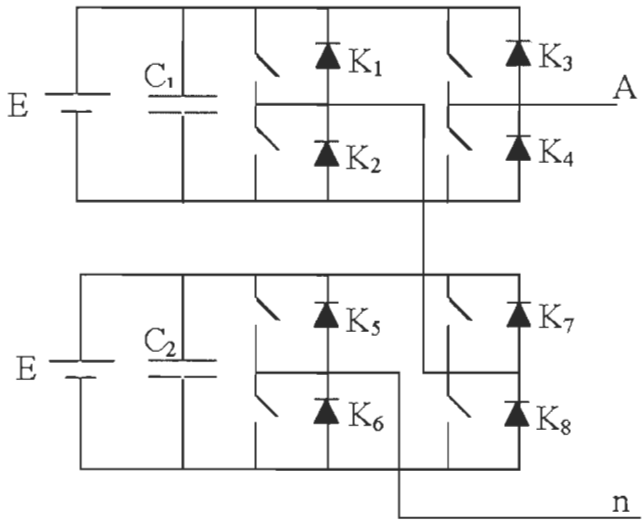


Fig. 3.5: Onduleur en cascade à 5 niveaux (phase A)

Les sorties des onduleurs en pont sont connectées en série telle que l'onde de la tension synthétisée est la somme des tensions de sortie. Le nombre des niveaux de tension de sortie dans un onduleur en cascade est définie par :

$$m = 2s + 1 \qquad (3.1)$$

où s est le nombre des sources des tensions continues.

L'avantage majeur de cette approche hybride est que le nombre de sortie peut être augmenté davantage sans aucun ajout de nouveaux composants. Il faut seulement des sources de tensions continues avec différents niveaux de tensions [16]. Probablement, le plus avantageux utilise des sources de tensions avec deux niveaux de tensions (E et 2E) comme le montre la figure 3.5. Cet arrangement peut générer une tension à sept (07) niveaux (0, +/- E, +/- 2E, +/- 3E).

3.3 COMMANDE D'UN ONDULEUR TRIPHASÉ

En électronique de puissance, les onduleurs de puissance deviennent de plus en plus incontournables. Ils sont présents dans les domaines d'application les plus variés dont le plus connu est sans doute celui de la variation de vitesse des machines à courant alternatif. Leur forte évolution est appuyée, d'une part, sur le développement de composants à semi-conducteurs entièrement commandables, puissantes, robustes et rapides, et autre part, sur l'utilisation quasi-généralisée des techniques dites de modulation de largeur d'impulsion. [17].

Il existe différentes stratégies de commande de modulation. Elles peuvent êtres classées comme suit :

- Commande en pleine onde.
- Modulation de largeur d'impulsion (MLI) ou PWM:
    - PWM Sinusoïdale
    - PWM Vectorielle ou SVPWM

Dans notre cas, dans ce chapitre, on s'intéresse à l'étude, la modélisation et la commande de l'onduleur de tension triphasé à deux niveaux de type NPC en utilisant la stratégie de la Modulation de Largeur d'Impulsion (MLI).

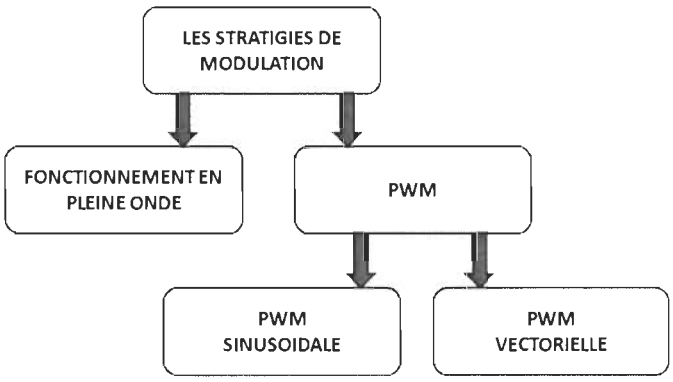


Fig. 3.6: Les différentes stratégies de modulation pour la commande des moteurs

3.3.1 Fonction d'un onduleur de tension

Un onduleur a deux fonctions principales :

- Une alimentation de secours en cas de coupure d'électricité

Dès que l'onduleur détecte une coupure brutale d'électricité, ses batteries prennent automatiquement le relais et alimentent l'appareil pendant une durée correspondante à l'autonomie (exemple de l'ordinateur).

- Une source d'électricité stable

Avoir une source d'électricité propre et stable, c'est, obtenir une onde sinusoïdale parfaite de 60Hz pour une tension de 120 V, ce qui n'est pas forcément le cas sur le réseau d'Hydro-Québec. La plage est de 106-127 V pour une alimentation à 120 V (Source : Hydro-Québec, Qualité onde).

En effet, l'électricité peut subir plusieurs types de perturbations :

- Les parasites
- Les variations de tensions (sur tensions ou sous tensions).

Elles sont provoquées par des appels de courants importants à la mise sous tension de gros équipements électriques (machine à laver, four, cumulus, ...).

- Les variations de fréquences.

Elles sont plus rares et sont provoquées par exemple par le passage sur un groupe électrogène.

- Les micro-coupures.

3.3.2 Modélisation et commande d'un onduleur de tension triphasé à deux niveaux "Structure NPC"

Un onduleur de tension triphasé dont les composants semi-conducteurs contrôlables sont des transistors ou des thyristors GTO, peut être considéré comme un amplificateur de puissance. Il est constitué de trois bras, de deux interrupteurs pour chacun. Chaque interrupteur est monté en parallèle inverse avec une diode de récupération. Pour assurer la continuité des courants alternatifs et éviter le court-circuit de la source, les interrupteurs K1 et K4, K2 et K5, K3 et K6 doivent être contrôlés de manière complémentaire.

Le schéma structurel d'un tel convertisseur statique alimentant le stator du MSAP est illustré par la figure 3.7.

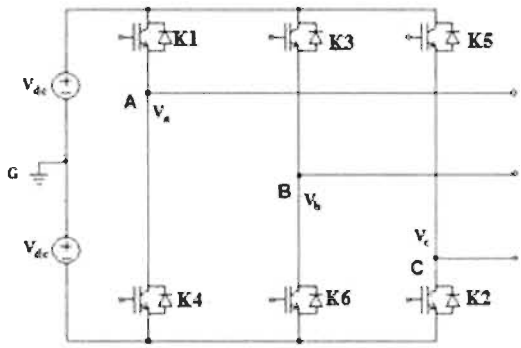


Fig. 3.7: Schéma d'un onduleur de tension triphasé

Pour simplifier l'étude, on supposera que :

- La commutation des interrupteurs est instantanée;

- La chute de tension aux bornes des interrupteurs est négligeable;

- la charge triphasée, est équilibrée, couplée en étoile avec un neutre isolé.

Les tensions simples s'écrivent en fonction des tensions composéesV_{sab}, V_{sbc} et V_{sca} sous la forme suivante:

$$\begin{cases} V_{sa} = \frac{1}{3}(V_{sab} - V_{sca}) \\ V_{sb} = \frac{1}{3}(V_{sbc} - V_{sab}) \\ V_{sc} = \frac{1}{3}(V_{sca} - V_{sbc}) \end{cases} \quad (3.2)$$

Les diodes D_i =1, 2, … 6, sont des diodes de protection des transistors assurant la roue libre ou la récupération.

Plusieurs méthodes sont utilisées pour commander les interrupteurs d'un onduleur. La stratégie la plus utilisée est la Modulation de Largeur d'Impulsion (MLI ou PWM).

- **PWM Sinusoïdale**

La technique de Modulation de Largeur d'Impulsion (MLI ou PWM : Pulse Width Modulation) ou de Modulation d'Impulsions en Durée (MID), proposé pour la première fois en 1981, par N. Akira & al [7], est, de loin, l'onduleur le plus performant.

Principe :

Une onde modulatrice sinusoïdale u, de fréquence f_u est comparée à une onde triangulaire v de fréquence f_v. La sortie du comparateur permet par l'intermédiaire de transistors de puissance le pilotage d'une phase de la machine. Les autres phases sont pilotées par des ensembles identiques, déphasés de 120°.

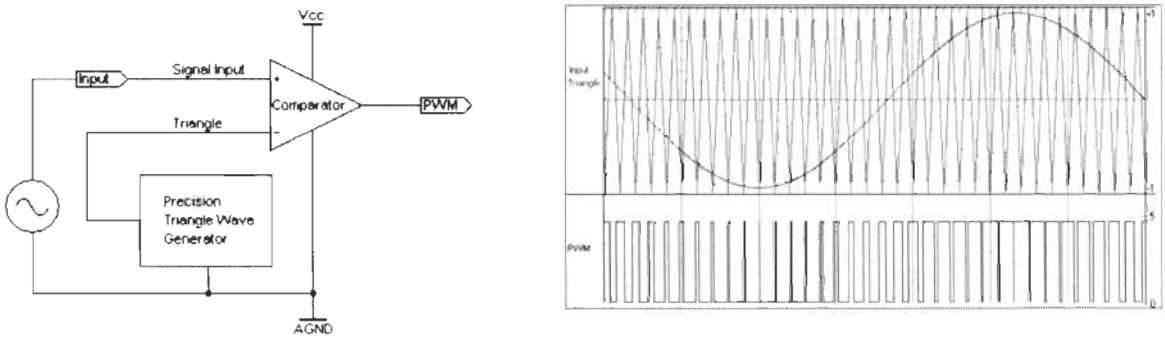


Fig. 3.8: Signal PWM modulé

Comme illustré sur la figure 3.8, la tension de sortie de l'onduleur est déterminée comme suit [8]:

- Quand $V_{control} > V_{tri}$, $V_{AO} = V_{dc}/2$
- Quand $V_{control} < V_{tri}$, $V_{AO} = - V_{dc}/2$

Aussi, la tension de sortie de l'onduleur a les caractéristiques suivantes:

- La fréquence du PWM est la même que la fréquence du V_{tri}
- L'amplitude est contrôlée par la valeur pic de $V_{control}$
- La fréquence fondamentale est contrôlée par la fréquence de $V_{control}$

Index de modulation (m) est défini comme :

$$m = \frac{V_{control}}{V_{tri}} = \frac{\text{peak of } (V_{AO})}{V_{dc}/2}, \quad (3.3)$$

Où (V_{AO}): le composant V_{AO}de la fréquence fondamentale.

- **PWM Vectorielle ou SVPWM**

Le circuit modèle d'un onduleur typique PWM triphasé est montré sur la figure 3.9.

S_1 à S_6 sont les six interrupteurs qui déterminent la sortie. Ils sont contrôlés par les variables a, a', b, b', c et c'. Lorsque un transistor de la partie supérieure est commuté sur ON, c'est-à-dire quand a, b ou c est à 1, la partie inférieure correspondante est commuté sur OFF, autrement dit a', b' ou c'est à 0. Par conséquent, les états ON et OFF des transistors supérieurs S_1, S_3 et S_5 peuvent être utilisés pour déterminer la tension de sortie.

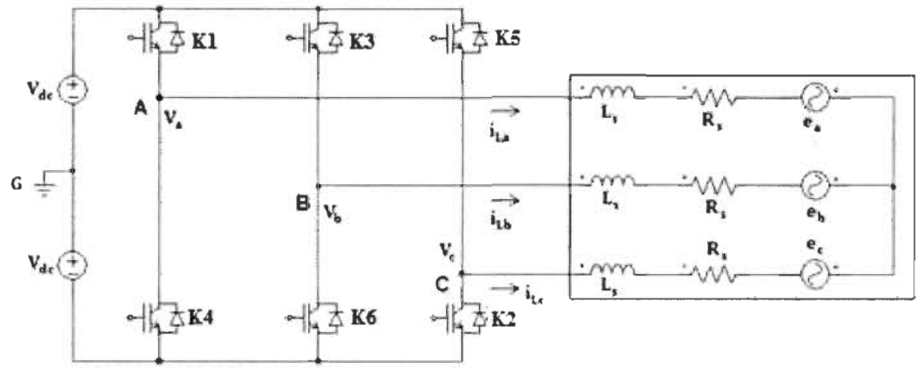


Fig. 3.9: Schéma d'un moteur alimenté par un onduleur triphasé

La relation entre la variable vecteur $[a, b, c]^t$ et le vecteur tension ligne à ligne $[V_{ab}\ V_{bc}\ V_{ca}]^t$ est donné par l'équation 2.1 comme suit :

$$\begin{bmatrix} V_{ab} \\ V_{bc} \\ V_{ca} \end{bmatrix} = V_{dc} \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{3.4}$$

Aussi, la relation entre la variable vecteur $[a, b, c]^t$ et le vecteur tension de phase $[V_a\ V_b\ V_c]^t$ peut être exprimé comme suit:

$$\begin{bmatrix} V_{an} \\ V_{bn} \\ V_{cn} \end{bmatrix} = \frac{V_{dc}}{3} \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{3.5}$$

Comme illustré au tableau 3.1, il y'a huit combinaisons possibles de ON et OFF pour les trois interrupteurs de puissance supérieurs. Les états ON et OFF des composants de puissance inférieurs sont opposés aux supérieurs et donc sont faciles à déterminer une fois les états des transistors supérieurs sont déterminés. A partir des équations (2.3) et (2.4), nous déterminons les huit vecteurs de tension de sortie ligne-neutre (tension de phase), et les tensions de sortie ligne-à-ligne. Elles sont données dans le Tableau 3.1. La figure 3.10 montre les huit vecteurs de tension (V_0 to V_7) de l'onduleur.

Tableau 3.1: Vecteurs tensions de phase et tension de sortie ligne par ligne

Vecteurs tension	Vecteurs de switch			Tension Ligne-Neutre			Tension Ligne-à-Ligne		
	a	b	c	V_{an}	V_{bn}	V_{cn}	V_{ab}	V_{bc}	V_{ca}
V_0	0	0	0	0	0	0	0	0	0
V_1	1	0	0	2/3	-1/3	-1/3	1	0	-1
V_2	1	1	0	1/3	1/3	-2/3	0	1	-1
V_3	0	1	0	-1/3	2/3	-1/3	-1	1	0
V_4	0	1	1	-2/3	1/3	1/3	-1	0	1
V_5	0	0	1	-1/3	-1/3	2/3	0	-1	1
V_6	1	0	1	1/3	-2/3	1/3	1	-1	0
V_7	1	1	1	1/3	-2/3	1/3	1	-1	0

Notons que les tensions respectives devront être multipliées par V_{dc}.

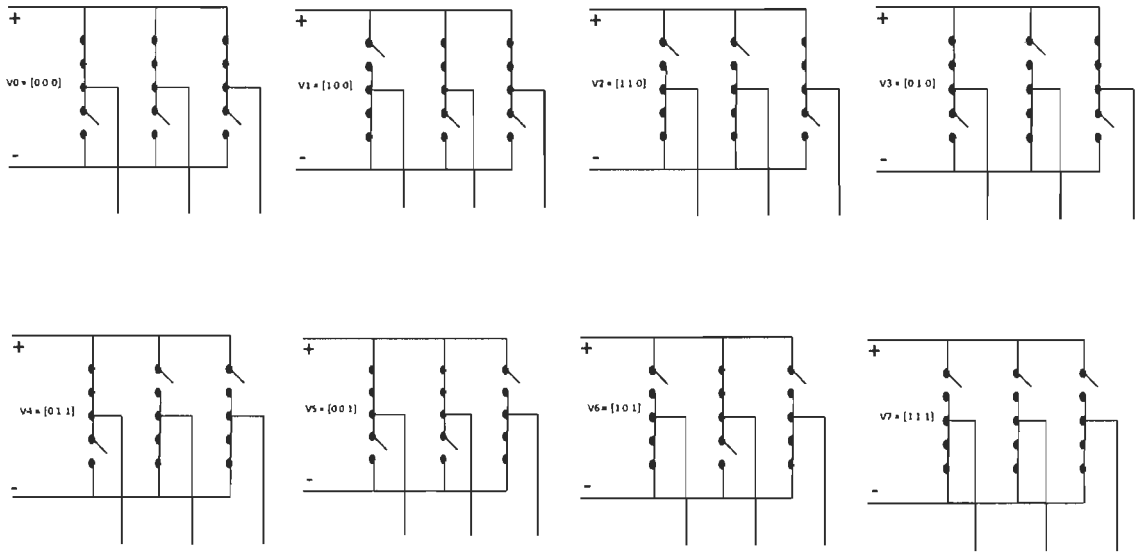


Fig. 3.10: Les huit vecteurs tensions de l'onduleur (V_0 to V_7)

Le Space vector PWM (SVPWM) fait référence à une séquence de commutation spéciale des trois transistors supérieurs de l'onduleur triphasé. Il a été élaboré de manière à générer moins de distorsion harmonique dans les tensions de sortie et/ou les courants appliqués aux phases du moteur AC et à assurer une utilisation plus efficace de la tension d'alimentation avec la technique de modulation sinusoïdale comme indiqué sur la figure 3.11.

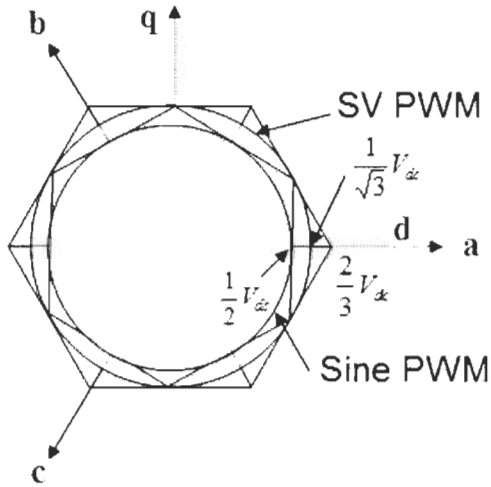


Fig. 3.11: Comparaison de la tension de contrôle linéaire maximum dans Sine PWM et SVPWM.

Pour implémenter le SVPWM, les équations de tension dans le repère *abc* peuvent être transformées dans le repère stationnaire *dq* dont l'axe des abscisses est *d* celui des ordonnées est *q* comme le montre la figure 3.12.

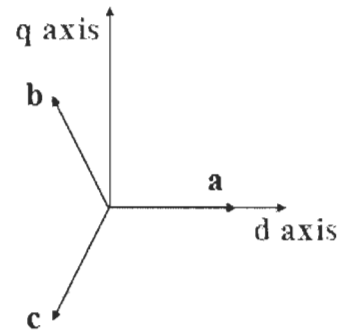


Fig. 3.12: Relation entre le repère abc et le repère stationnaire dq

D'après cette figure, la relation entre ces deux références est :

$$f_{dq0} = K_s . f_{abc} \tag{3.6}$$

Où $K_s = \frac{2}{3}\begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \\ 1/2 & 1/2 & 1/2 \end{bmatrix}$, $f_{dq0} = [f_d\ f_q\ f_0]^t$, $f_{abc} = [f_a\ f_b\ f_c]^t$ et f représente la variable de tension ou de courant. (3.7)

Comme décrit dans la figure 3.13, cette transformation est équivalent à une projection orthogonale de $[a, b, c]^t$ sur les deux dimensions perpendiculaires au vecteur $[1, 1, 1]^t$ (l'équivalent du plan d-q) dans un système de coordonnée à trois-dimension. Il en résulte, six vecteurs non nuls et deux vecteurs nuls possibles. Les six vecteurs non nuls (V_1 - V_6) forment les axes d'une forme hexagonale comme représentés à la Figure 3.13 et l'alimentation électrique de la charge. L'angle entre deux vecteurs non nuls adjacents est 60 degré. Les deux vecteurs nuls (V_0 et V_7) sont à l'origine et appliquent une tension nulle à la charge. Les huit vecteurs sont appélés vecteurs espace de base et sont représentés par V_0, V_1, V_2, V_3, V_4, V_5, V_6 et V_7. La même transformation peut être appliquée à la tension de sortie désirée pour obtenir le vecteur tension de référence désiré V_{ref} dans le plan d-q.

L'objectif de la technique SVPWM est d'avoir une approximation du vecteur tension de référence V_{ref} en utilisant les huit interrupteurs. La méthode d'approximation utilisée consiste à générer un même rendement moyen de la sortie de l'onduleur dans une petite période T comme celui de V_{ref} dans la même période.

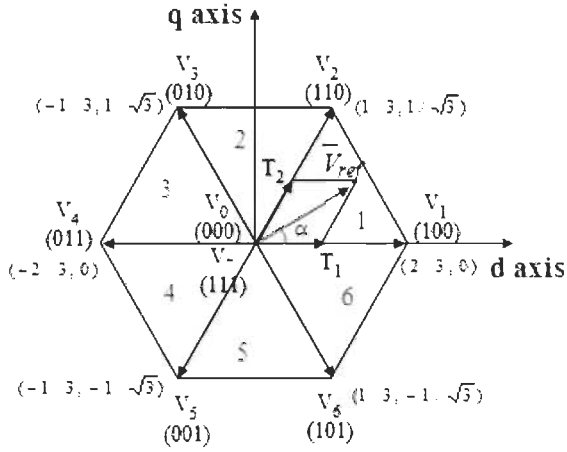


Fig. 3.13: Vecteurs de commutation de base et secteurs

Par conséquent, SVPWM peut être implémenté suivant les étapes ci-dessous :

- Étape 1 : détermination de V_d, V_q, V_{ref}, et l'angle α
- Étape 2 : détermination du temps T_1, T_2, T_0
- Étape 3 : détermination du temps de commutation de chaque transistor (S_1, S_6).

- **Étape 1 : Détermination de V_d, V_q, V_{ref}, et l'angle α**

D'après la figure 3.12, les vecteursV_d, V_q, V_{ref}, et l'angle (α) peuvent être déterminés comme suit :

$$V_d = V_{an} - V_{bn}.\cos(60) - V_{cn}.\cos(60) \quad (3.8)$$

$$= V_{an} - \frac{1}{2}V_{bn} - \frac{1}{2}V_{cn}$$

$$V_q = 0 + V_{bn}.\cos(30) - V_{cn}.\cos(30) \quad (3.9)$$

$$= V_{an} + \frac{\sqrt{3}}{2}V_{bn} - \frac{\sqrt{3}}{2}V_{cn}$$

❖ $\begin{bmatrix} V_d \\ V_q \end{bmatrix} = \frac{2}{3}\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} V_{an} \\ V_{bn} \\ V_{cn} \end{bmatrix}$ (3.10)

❖ $|\overline{V_{ref}}| = \sqrt{{V_d}^2 + {V_q}^2}$ (3.11)

❖ $\alpha = \tan^{-1}(\frac{V_q}{V_d}) = \omega t = 2\pi f t$, où f = fréquence fondamentale (3.12)

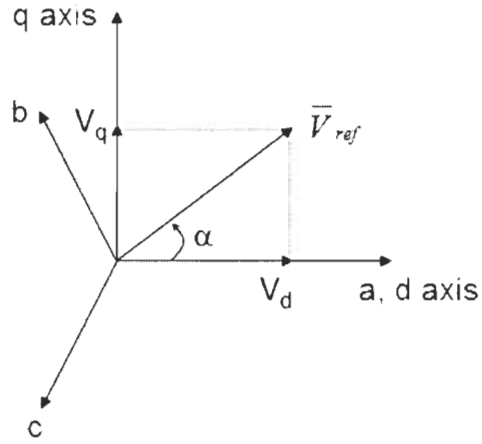


Fig. 3.14: Vecteur espace tension et ses composants dans (d, q)

- **Étape 2 : Détermination des durées T_1, T_2, T_0**

A partir de la figure 3.15, la durée du temps de commutation peut être calculée comme suit :

- La durée du temps de commutation au niveau du secteur 1

$$\int_0^{T_s} \overline{V_{ref}} = \int_0^{T_1} \overline{V_1}\, dt + \int_{T_1}^{T_1+T_s} \overline{V_2}\, dt + \int_{T_1+T_s}^{T_s} \overline{V_0} \quad (3.13)$$

❖ $T_s . |\overline{V_{ref}}| . \begin{bmatrix} \cos(\alpha) \\ \sin(\alpha) \end{bmatrix} = T_1 . \frac{2}{3} . V_{dc} . \begin{bmatrix} 1 \\ 0 \end{bmatrix} + T_2 . \frac{2}{3} . V_{dc} . \begin{bmatrix} \cos(\pi/3) \\ \sin(\pi/3) \end{bmatrix}$ (3.14)

(où $0 \leq \alpha \leq 60^o$)

❖ $T_1 = T_s . a . \frac{\sin(\pi/3 - \alpha)}{\sin(\pi/3)}$ (3.15)

❖ $T_2 = T_s . a . \frac{\sin(\alpha)}{\sin(\pi/3)}$ (3.16)

❖ $T_0 = T_s - (T_1 + T_2)$, (où $T_s = \frac{1}{f_z}$ et $a = \frac{|\overline{V_{ref}}|}{\frac{2}{3} . V_{dc}}$) (3.17)

- Durée de temps de commutation dans n'importe quel secteur

❖ $T_1 = \frac{\sqrt{3}.T_s.|\overline{V_{ref}}|}{V_{dc}}\left(\sin\left(\frac{\pi}{3} - \alpha + \frac{n-1}{3}.\pi\right)\right)$ (3.18)

$= \frac{\sqrt{3}.T_s.|\overline{V_{ref}}|}{V_{dc}}\left(\sin\left(\frac{n}{3}.\pi - \alpha\right)\right)$

$= \frac{\sqrt{3}.T_s.|\overline{V_{ref}}|}{V_{dc}}\left(\sin(\frac{n}{3}.\pi).\cos(\alpha) - \cos(\frac{n}{3}.\pi).\sin(\alpha)\right)$

❖ $T_2 = \frac{\sqrt{3}.T_s.|\overline{V_{ref}}|}{V_{dc}}\left(\sin\left(\alpha - \frac{n-1}{3}.\pi\right)\right)$ (3.19)

$= \frac{\sqrt{3}.T_s.|\overline{V_{ref}}|}{V_{dc}}\left(-\cos(\alpha).\sin(\frac{n-1}{3}.\pi) - \sin(\alpha).\cos(\frac{n-1}{3}.\pi)\right)$

❖ $T_0 = T_s - (T_1 + T_2), \left(\begin{matrix}\text{où n} = 1 \text{ à } 6 \text{ (représentant le secteur1 à 6)} \\ 0 \leq \alpha \leq 60^o\end{matrix}\right)$ (3.20)

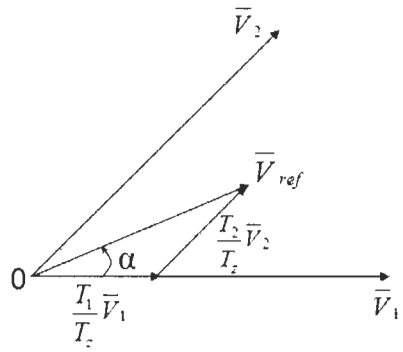


Fig. 3.15: Vecteur référence comme résultante des vecteurs adjacents du secteur 1

- **Étape 3: Détermination du temps de commutation de chaun des transistors (S_1 à S_6)**

La figure 3.16 montre le temps de commutation du SVPWM dans chaque secteur

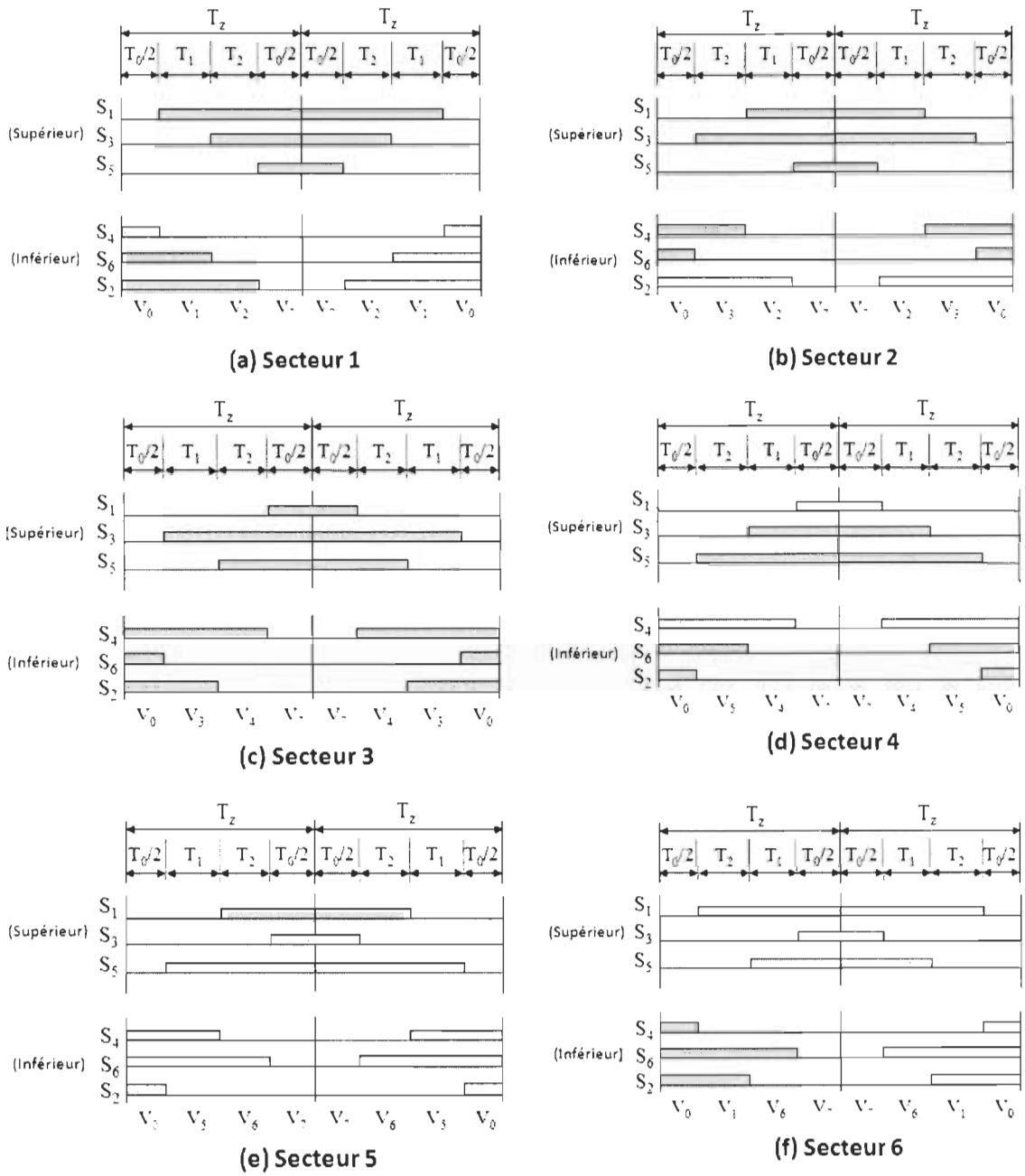


Fig. 3.16: Temps de commutation du SVPWM dans chaque secteur

En me basant sur la figure 3.16, nous pouvons calculer le temps de commutation dans chaque secteur. Ce calcul est résumé dans le tableau 3.2, et sera également modélisé dans Simulink pour l'implémentation du SVPWM.

Tableau 3. 2: Calcul de temps de commutation

Secteur	Interrupteurs supérieurs (S_1, S_3, S_5)	Interrupteurs inférieurs (S_4, S_6, S_2)
1	$S_1 = T_1 + T_2 + T_0/2$ $S_3 = T_2 + T_0/2$ $S_5 = T_0/2$	$S_4 = T_0/2$ $S_6 = T_1 + T_0/2$ $S_2 = T_1 + T_2 + T_0/2$
2	$S_1 = T_1 + T_0/2$ $S_3 = T_1 + T_2 + T_0/2$ $S_5 = T_0/2$	$S_4 = T_2 + T_0/2$ $S_6 = T_0/2$ $S_2 = T_1 + T_2 + T_0/2$
3	$S_1 = T_0/2$ $S_3 = T_1 + T_2 + T_0/2$ $S_5 = T_2 + T_0/2$	$S_4 = T_1 + T_2 + T_0/2$ $S_6 = T_0/2$ $S_2 = T_1 + T_0/2$
4	$S_1 = T_0/2$ $S_3 = T_1 + T_0/2$ $S_5 = T_1 + T_2 + T_0/2$	$S_4 = T_1 + T_2 + T_0/2$ $S_6 = T_2 + T_0/2$ $S_2 = T_0/2$
5	$S_1 = T_2 + T_0/2$ $S_3 = T_0/2$ $S_5 = T_1 + T_2 + T_0/2$	$S_4 = T_1 + T_0/2$ $S_6 = T_1 + T_2 + T_0/2$ $S_2 = T_0/2$
6	$S_1 = T_1 + T_2 + T_0/2$ $S_3 = T_0/2$ $S_5 = T_1 + T_0/2$	$S_4 = T_0/2$ $S_6 = T_1 + T_2 + T_0/2$ $S_2 = T_2 + T_0/2$

La MLI dite space vector (vecteur spatial) est surtout applicable aux variateurs de vitesse triphasés. Elle consiste à considérer globalement le système triphasé, et à lui appliquer une transformée de Concordia pour le ramener dans le plan (V_α,V_β). Le système de tensions triphasées à générer pour la durée d'échantillonnage en cours peut alors être représenté comme un unique vecteur dans ce plan. Ce vecteur n'est pas directement réalisable par les interrupteurs du variateur, mais on peut chercher les trois configurations les plus proches (situées sur les sommets et au centre de l'hexagone), et les appliquer successivement pendant une fraction adéquate de la période d'échantillonnage, de façon à obtenir en moyenne le vecteur recherché. En modulation sinusoïdale, elle donne des résultats similaires à la MLI intersective à porteuse triangulaire centrée. Néanmoins, elle peut être plus facile à implanter dans un microcontrôleur ou FPGA. Cette technique à l'avantage de maximiser la puissance disponible, ce qui justifie son usage.

3.4 MODELISATION SIMULINK DE LA TECHNIQUE SVPWM

3.4.1 Étapes de la simulation

1. Initialisation des paramètres du système avec Matlab

2. Construction du modèle Simulink

- Détermination du secteur
- Détermination de la durée T_1, T_2, T_0
- Détermination des temps de commutation (Ta, Tb, Tc) de chacun des transistors (S1 à S6)
- Génération de la tension de sortie de l'onduleur (V_{AB},V_{BC},V_{CA})
- Envoi des données au Workspace

3. Affichage des résultats de simulation avec Matlab

Paramètres du système:

- IGBTs: SEMIKRON SKM 50 GB 123D, Max ratings: VCES = 600 V, IC = 80 A
- DC- link voltage: V_{dc} = 300 V
- Fréquence fondamentale: f = 30 Hz
- Fréquence PWM (carrier) : f_z = 5 kHz
- Résistance statorique R_s: 1.115 Ω
- Résistance rotorique R_r: 1.083 Ω
- Mutual Inductance L_m: 0.2037 H

3.4.2 Modèle Simulink

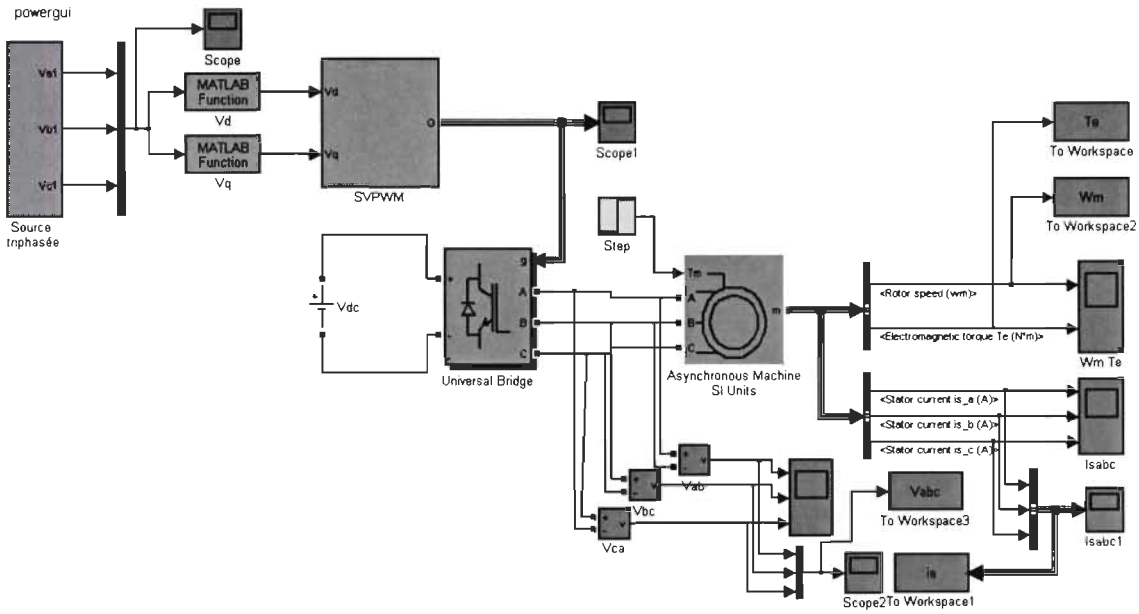


Fig. 3.17: Commande SVPWM d'une machine asynchrone

À l'intérieur du bloc SVPWM, se trouve celui qui génère les signaux PWM à partir des temps de commutation calculés (Fig. 3.18).

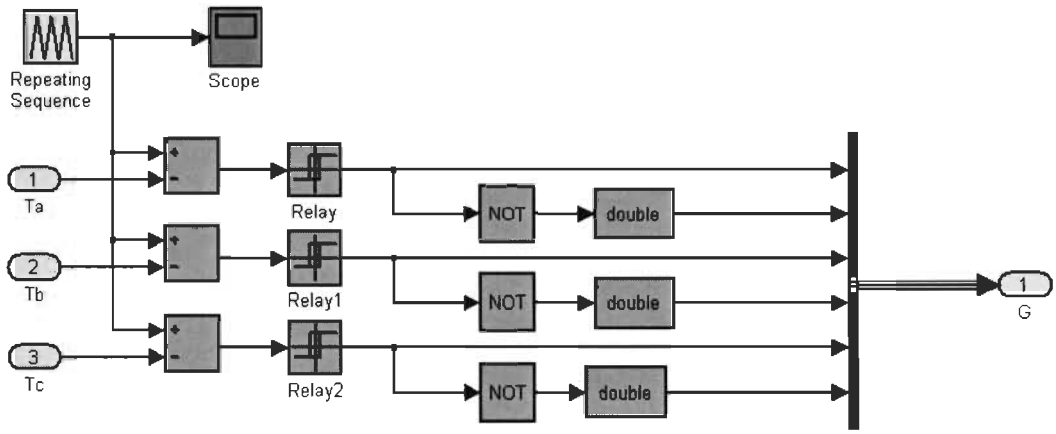
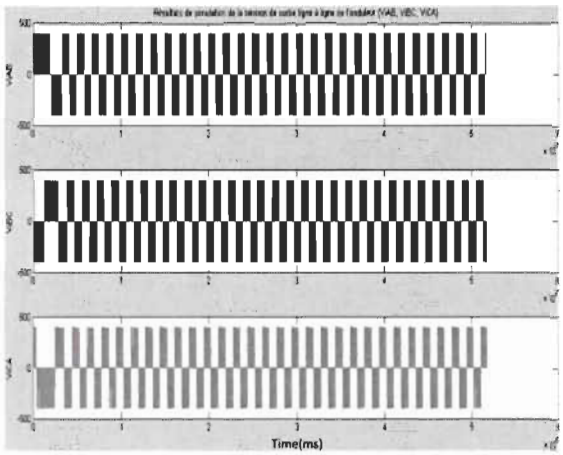


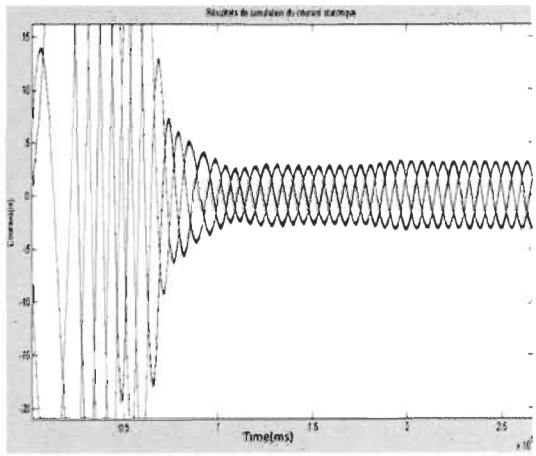
Fig. 3.18: Génération des signaux PWM

3.4.3 Résultats de simulation

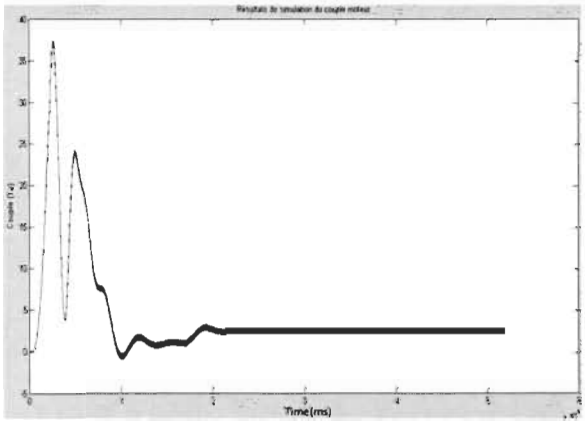
Les figures ci-dessous donnent les résultats de la simulation SVPWM à deux niveaux.



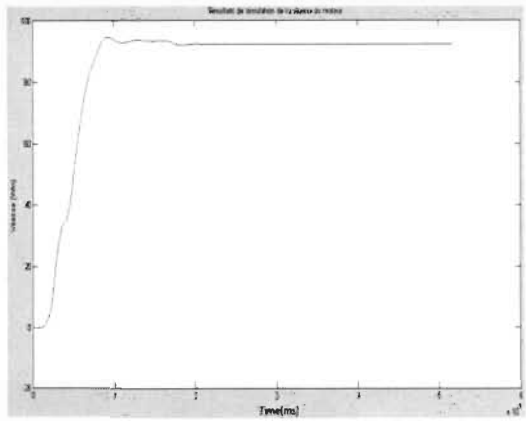
a. Tension de sortie ligne par ligne de l'onduleur (ViAB, ViBC, ViCA)



b. Courant statorique (IiAB, IiBC, IiCA)



c. Comportement du couple



d. Comportement de la vitesse

Fig. 3.19: Résultats de simulation SVPWM

On remarque que le couple (Figure 3.19.c), répond parfaitement au changement de consigne :

- Au démarrage à t = 0 ms, le couple C_{em} = 0.5 N.m
- À partir de t = 0.3 ms, C_{em} = 2 N.m

3.5 CONCLUSION

Dans ce chapitre, nous avons présenté la structure de l'alimentation d'une machine asynchrone et les principaux blocs la constituant. Une brève présentation des trois principales topologies des onduleurs a été faite ainsi que leurs caractéristiques.

Nous avons également modélisé et simulé la commande de l'onduleur triphasé avec la technique SVPWM. Les résultats obtenus démontrent la justesse du modèle développé. Dans le prochain chapitre, le module SVPWM développé, sera utilisé dans une commande à boucle fermée DTC d'une machine asynchrone.

CHAPITRE 4

CONTROLE DIRECT DU COUPLE (DTC) SIMPLIFIE D'UNE MACHINE ASYNCHRONE

4.1 INTRODUCTION

Ces dernières années, de nombreuses études ont été menées sur le contrôle des moteurs à induction afin de développer des nouvelles techniques permettant d'avoir une réponse précise et rapide du couple, et réduire la complexité des algorithmes. La technique de la Commande Direct du Couple (DTC) a été reconnue comme une solution viable qui répond à ces exigences. Cependant la DTC conventionnelle a des limites dont les principales sont : une ondulation de couple élevée, la moyenne du couple de sortie qui ne correspond pas à la demande, et dans les commandes à fréquence de commutation variable [12]. Pour pallier à ces problèmes de la DTC conventionnelle, une multitude de techniques a été proposée [18]. Parmi lesquelles nous pouvons citer: la DTC-SVPWM [19], l'utilisation d'un contrôleur de rapport cyclique pour introduire une modulation entre les vecteurs actifs choisis de la table de vérité et les vecteurs nuls [20], [21],et l'utilisation des techniques d'intelligence artificielle tels que les approches neuro-floue avec contrôleur SVPWM [22]. Dans le cadre de notre mémoire, la technique qui sera utilisée est celle de la DTC-SVPWM. Car elle a l'avantage de produire des

harmoniques plus basses et permet également de pourvoir utiliser une indice de modulation élévée.

Dans ce chapitre, la commande DTC-SVPWM sera étudiée. Nous allons dans un premier temps expliquer le principe de commande de base. Ensuite exposer comment l'algorithme est organisé. Et par après discuter des résultats des différents tests réalisés afin de valider l'algorithme. Enfin, des conclusions seront tirées. Quant aux onduleurs multi-niveaux, ils seront abordés dans le chapitre 5.

4.2 COMMANDE DTC (DIRECT TORQUE CONTROL) D'UNE MAS

4.2.1 Principe de commande

La technique DTC est un principe de commande des moteurs à courant alternatif [24]. Elle permet de calculer les grandeurs de contrôle que sont le flux statorique et le couple électromagnétique, uniquement à partir des mesures de tensions et de courants statoriques, sans disposer de mesures de vitesse, de flux ou de couple. Ces deux derniers sont entièrement estimés en utilisant des régulateurs à hystérésis qui comparent les valeurs actuelles calculées du couple et du flux aux valeurs de référence. Dépendamment du signal du contrôleur à hystérésis ou comparateur, l'onduleur est commandé par un "sélecteur d'interrupteurs optimisé". Les interrupteurs de l'onduleur imposent au moteur les tensions nécessaires qui contrôleront le couple et le flux en même temps dans une commande à boucle fermée. La technique de la DTC est ainsi représentée par la figure 4.1.

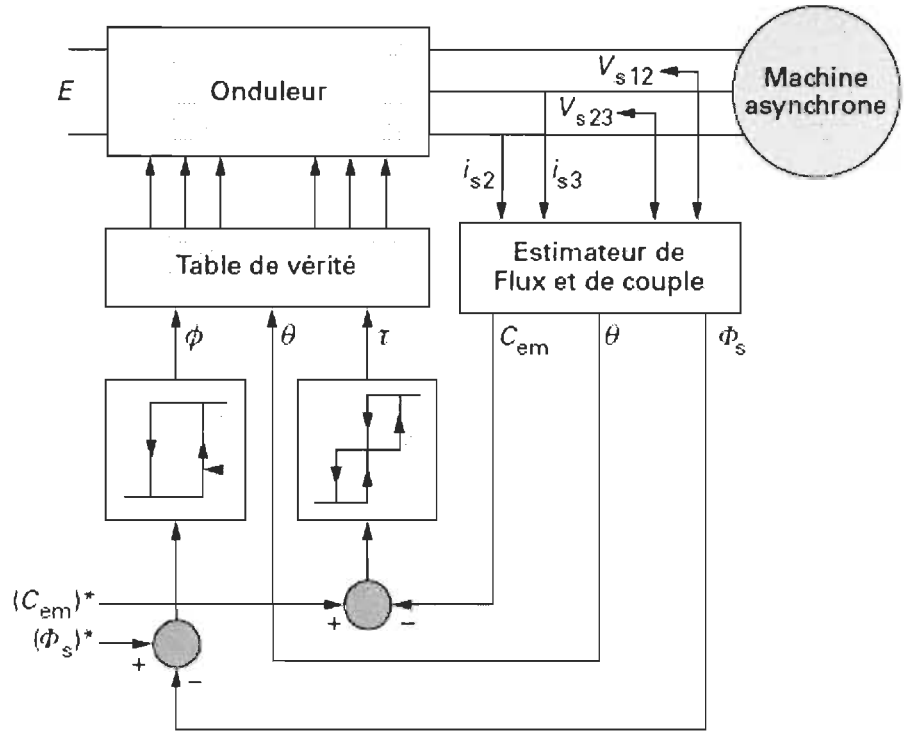


Fig. 4.1: Schéma de principe de la DTC

4.2.2 Développement de la commande DTC

Sur la figure 4.1 sont représentés les estimateurs de flux et de couple, ainsi que les régulations par hystérésis du couple et du flux. La connaissance du secteur dans lequel se trouve le vecteur flux statorique est donnée à partir des deux composantes de ce flux.

4.2.3 Estimation du flux statorique et le couple électromagnétique

Le couple électromagnétique et le flux statorique peuvent s'écrire respectivement :

$$C_{em} = \frac{3}{2}.\mathrm{P}.(\varphi_s.I_s) \tag{4.1}$$

$$\varphi_s = \int_0^t (V_s - R_s.I_s)dt \tag{4.2}$$

Dans le repère (αβ), on obtient respectivement : V_s, I_s , ω_r^*

$$C_{em} = \frac{3}{2}.\mathrm{P}.(\varphi_{s\alpha}.I_{s\beta} - \varphi_{s\beta}.I_{s\alpha}) \tag{4.3}$$

$$\begin{cases} \varphi_{s\alpha} = \int_0^t (V_{s\alpha} - R_s.I_{s\alpha})dt & (4.4) \\ \varphi_{s\beta} = \int_0^t (V_{s\beta} - R_s.I_{s\beta})dt & (4.5) \end{cases}$$

4.2.4 Comparateurs à hystérésis et table de vérité

On considère, dans un premier temps, la solution de Takahashi avec deux états pour le flux et trois pour le couple. La table de vérité définie par Takahashi est reproduite dans le tableau 4.1. Cette table est commandée par trois paramètres :

- ϕ qui dépend de la valeur du flux statorique par rapport à sa bande d'hystérésis. Si ϕ = 0, cela signifie qu'il faut réduire le flux. Si ϕ = 1, il faut augmenter le flux (figure 4.2);
- τ qui dépend de la valeur du couple électromagnétique par rapport à sa bande d'hystérésis. Si τ = + 1, le couple est inférieur à la limite inférieure de la bande et il faut donc l'augmenter. Si τ = 0, le couple est à l'intérieur de la bande et il faut l'y maintenir. Si τ = – 1, le couple est supérieur à la limite supérieure de la bande et il faut donc le diminuer (figure 4.3) ;
- θ est le numéro du secteur dans lequel se trouve le vecteur flux statorique. On considère ici six secteurs de 60° chacun (figure 4.4).

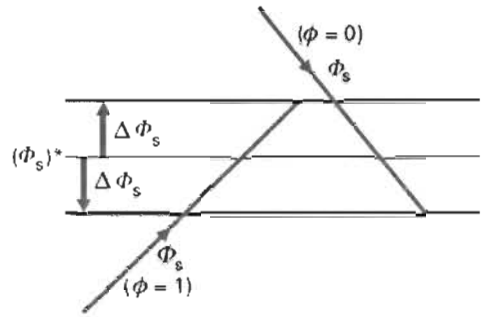


Fig. 4.2: Évolution du flux par rapport à sa bande d'hystérésis

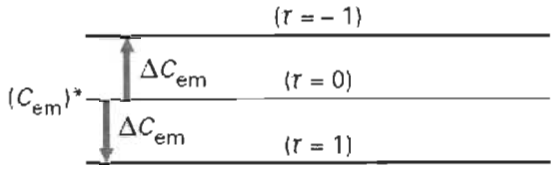


Fig. 4.3: Évolution du couple électromagnétique par rapport à sa bande d'hystérésis

Tableau 4.1 : Table de vérité de Takahashi

Tableau 1 – Table de vérité définie par Takahashi						
φ	1	1	1	0	0	0
τ	1	0	- 1	1	0	- 1
θ = 1	V_2	V_7	V_6	V_3	V_0	V_5
θ = 2	V_3	V_0	V_1	V_4	V_7	V_6
θ = 3	V_4	V_7	V_2	V_5	V_0	V_1
θ = 4	V_5	V_0	V_3	V_6	V_7	V_2
θ = 5	V_6	V_7	V_4	V_1	V_0	V_3
θ = 6	V_1	V_0	V_5	V_2	V_7	V_4

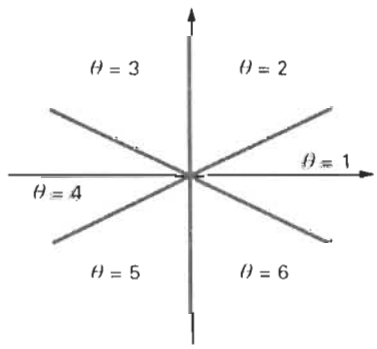


Fig. 4.4: Secteurs du plan complexe

Pour expliquer comment la table a été construite, on considère un exemple où $\phi = 1$ et $\tau = 1$ avec $\theta = 1$. Le vecteur flux est situé dans le secteur 1 et il faut augmenter le flux et le couple. On dispose des six tensions actives. Sur la figure 4.6, nous voyons que les tensions V_1, V_2 et V_6 ont tendance à augmenter l'amplitude du flux, alors que V_2, V_3 et V_4 ont tendance à accélérer le vecteur flux, donc à augmenter l'angle γ et donc le couple. On vérifie que pour cette position du vecteur flux dans le secteur 1, seule la tension V_2 est capable d'augmenter à la fois l'amplitude du flux et le couple. On peut ainsi envisager les différents cas. On remarque que Takahashi a choisi une séquence nulle toutes les fois où τ est nul, c'est-à-dire quand le couple est à l'intérieur de sa bande d'hystérésis (fig. 4.5). Le choix entre V_1 et V_7 est fait pour réduire le nombre de commutations. Pour les colonnes actives de la table, on remarque également qu'on passe d'un secteur au suivant par une permutation circulaire de l'indice de la tension.

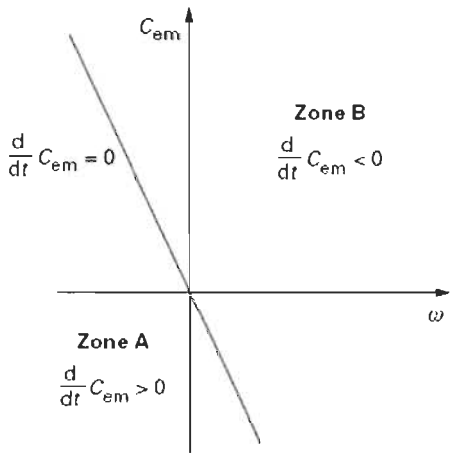


Fig. 4.5: Fonctionnement dans le plan

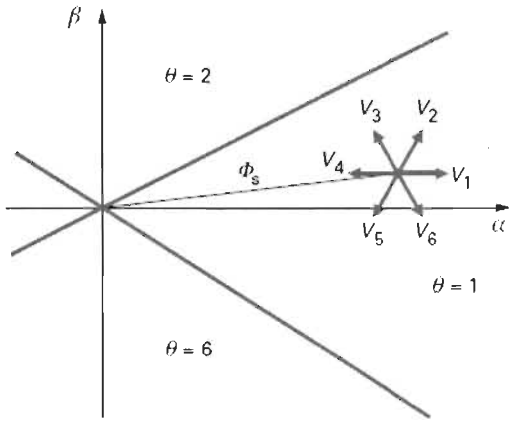


Fig. 4.6: Choix du vecteur tension

4.3 MODELISATION DE LA DTC SOUS MATLAB/SIMULINK

La figure ci-dessous, nous montre une vue générale du système de commande DTC.

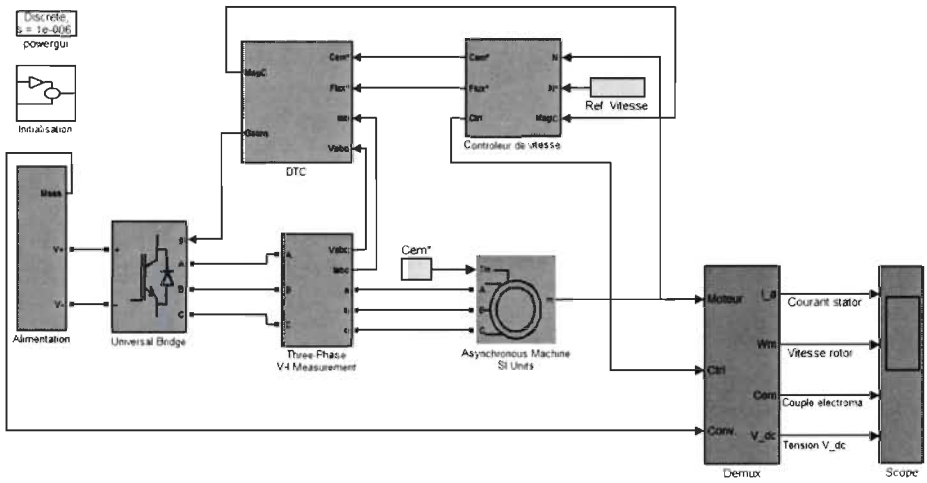


Fig. 4.7: Système de commande DTC

4.3.1 Algorithme DTC

Le bloc DTC est le bloc le plus important de la commande. Il est constitué de quatre blocs principaux : le calculateur de flux et de couple ou estimateur, les deux régulateurs de flux et de couple et enfin le bloc SVPWM (voir figure ci-dessous). Notons que dans cette partie, dans le but de faciliter la traduction en VHDL, nous avons programmé les différents blocs principaux en C sous Matlab, à l'exception du contrôleur de vitesse qui lui ne change pas selon les différentes techniques de la DTC.

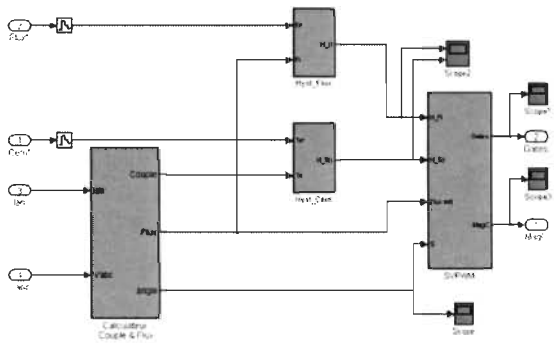


Fig. 4.8: Commande DTC sous Simulink

- **Algorithme des régulateurs de flux et de couple**

Les régulateurs de flux et de couple suivent la même logique expliquée précédemment dans la section 2.2 (comparateurs à hystérésis et table de vérité). Le programme est mis en annexe (annexe C.2.2 et C.2.3).

- **Algorithme du bloc SVPWM**

Il y'a deux blocs dans celui de la SVPWM : celui du calculateur du secteur et celui du générateur de vecteurs tension. Leurs organigrammes sont donnés respectivement sur les figures 4.9 et 4.10. Le programme est aussi donné en annexe (annexe C.2.4).

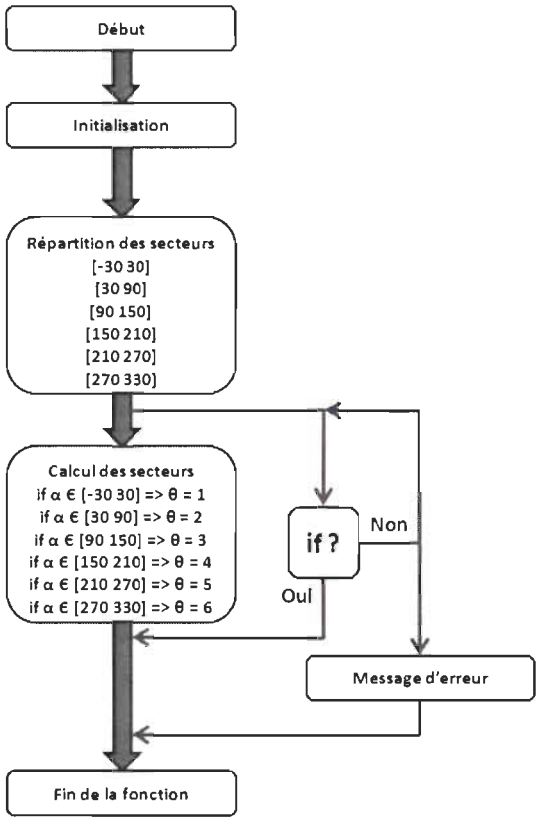


Fig. 4.9: Calcul du secteur

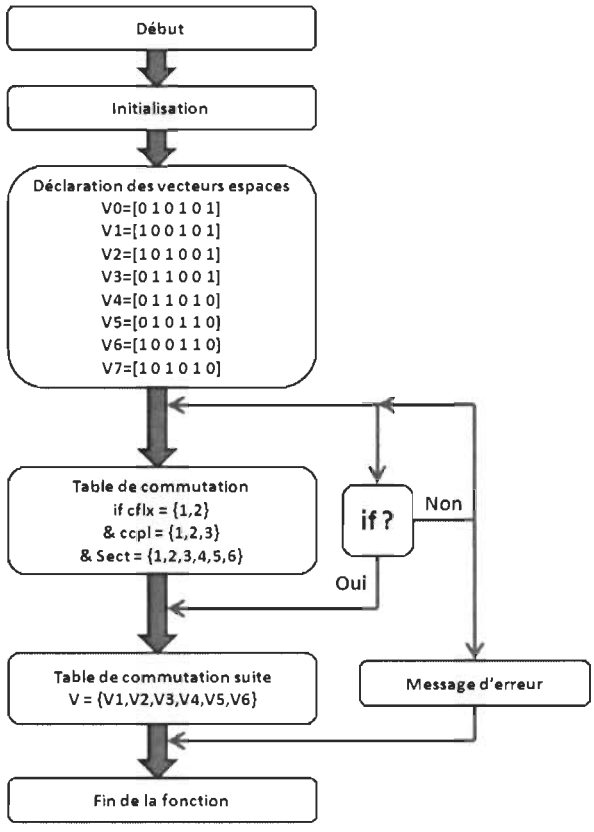


Fig. 4.10: SVPWM

Les quatre fonctions de l'algorithme sont dans la partie annexe de ce document (annexes C)

- Fonction calcul secteur,
- Fonction régulateur à hystérésis du flux,
- Fonction régulateur à hystérésis du couple,
- Fonction calcul SVPWM

4.3.2 Résultats de simulation

Les résultats de simulations ci-dessous, on été obtenus en utilisant les variables suivantes :

Paramètres

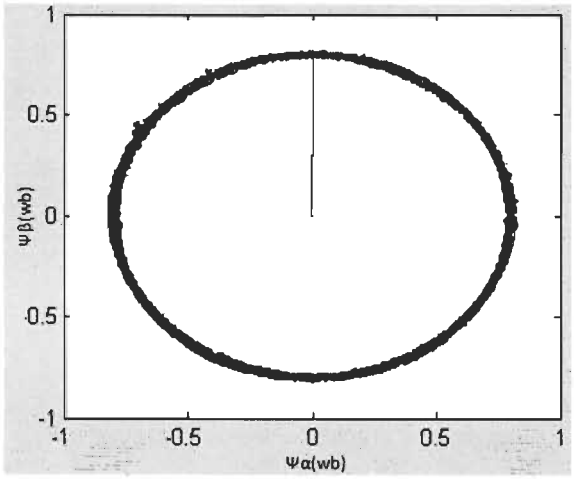
- Résistance statorique : 14.85e-3 Ω
- Inductance mutuelle : 10.46 mH
- Inductance rotorique : 0.3027 mH

Nombre de paires de pôles : 2

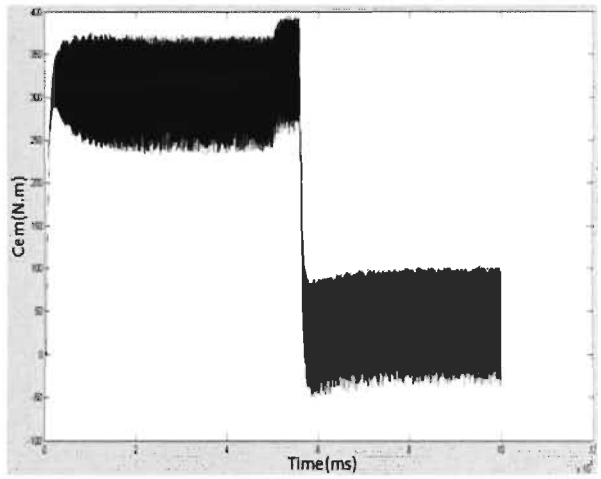
Moment d'inertie : 3.1 kg.m2

V_{dc} : 460 V

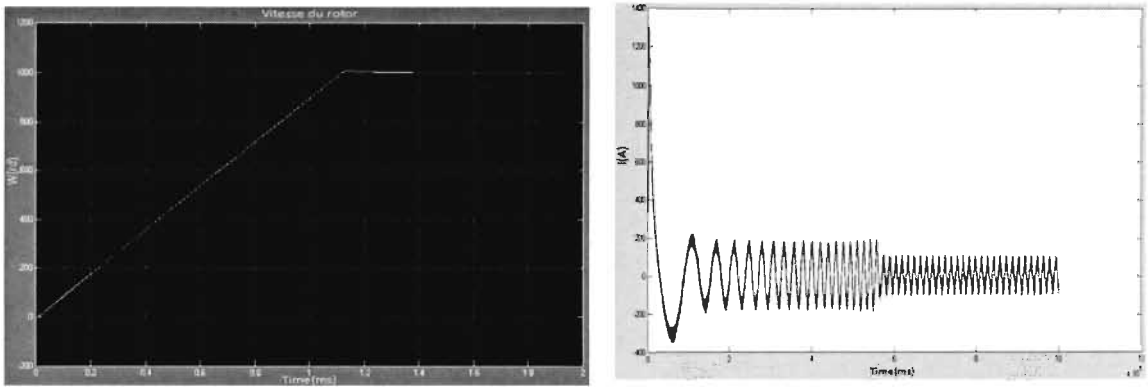
Nous pouvons constater que les variables de sortie du moteur suivent bien les références qui ont été fixé (au démarrage C_{em} = 350 N.m ; à t = 5 ms, C_{em} = 50 N.m). Sur la figure 4.11.a est représenté le cycle du flux statorique montrant la proximité de sa trajectoire avec un cercle.



a. Établissement du flux statorique dans le plan dq



b. Couple statorique



c. Vitesse rotorique

d. Courant statorique

Fig. 4.11: Résultats de simulation

4.4 CONCLUSION

Ce chapitre a été essentiellement consacré à la commande DTC. Les différents modules de la commande ont été particulièrement développés de façon à faciliter la modélisation en VHDL. Nous avons présenté la modélisation et la simulation du moteur asynchrone avec la commande DTC simplifiée en utilisant le logiciel MATLAB/SIMULINK. Dans le prochain chapitre nous traiterons de la commande des onduleurs multi-niveaux.

CHAPITRE 5

ALGORITHME DE COMMANDE GENERALE SVPWM POUR LES ONDULEURS MULTI-NIVEAUX

5.1 INTRODUCTION

Actuellement, l'industrie exige des équipements de puissance de plus en plus élevée, de l'ordre du mégawatt. L'évolution rapide des techniques de fabrication des dispositifs à semi-conducteurs et l'orientation des concepteurs vers la technologie des composants hybrides tels que l'IGBT, ont permis le développement de nouvelles structures de convertisseurs (onduleurs) d'une grande performance par rapport aux structures classiques. Ils sont mieux adaptés aux applications de grande puissance du fait qu'ils réduisent les contraintes dues aux phénomènes de commutation sollicitant les interrupteurs.

L'objectif ici est de réduire l'amplitude des harmoniques de tension et du courant, injectés par le convertisseur dans la charge. Ces harmoniques constituent l'un des plus grands problèmes concernant la qualité de puissance dans les systèmes électriques.

Dans ce chapitre, nous appliquons les onduleurs multi-niveaux pour commander une machine asynchrone, dans le but d'améliorer la qualité du signal. C'est ainsi qu'une étude des algorithmes de commande PWM et SVPWM pour les onduleurs multi-niveaux sera réalisée dans un premier temps, ensuite elle sera suivie d'une modélisation et une simulation sous Matlab/Simulink puis en VHDL. Nous analyserons également l'effet de la commande multi-niveaux sur le Taux de Distorsion Harmoniques (THD), la vitesse et le couple moteur dans un système de commande de machine asynchrone en fonction des différents niveaux. Dans ce chapitre, nous verrons également, jusqu'à où irons-nous concernant le niveau des onduleurs multi-niveaux. Aurait-il une limite de niveau afin de garder une bonne efficacité ?

Notons que les études réalisées dans ce chapitre sont basées sur les onduleurs multi-niveaux avec la topologie NPC (Neutral-Point-Clamped) du fait de ses avantages (voir chapitre 3).

5.2 INTERET DES ONDULEURS MULTI-NIVEAUX

Les onduleurs multi-niveaux sont de plus en plus utilisés dans les applications à haute puissance du fait de leur performance supérieure comparée aux onduleurs à deux niveaux.

Ainsi ils présentent plusieurs avantages [25] :

- D'une part les tensions et courants d'autre part, les structures multi-niveaux permettent de limiter les contraintes en tension subies par les interrupteurs de puissance : chaque composant, lorsqu'il est à l'état bloqué, supporte une fraction d'autant plus faible de la pleine tension de bus continu que le nombre de niveaux est élevé ;
- les sorties PWM délivrées par les onduleurs multi-niveaux présentent d'intéressantes qualités spectrales. Dans le cas plus précis d'un fonctionnement en SVPWM, le recours à ce type d'onduleur associé à une commande judicieuse des composants de puissance, permet en outre de supprimer certaines familles d'harmoniques ;

- Aussi un faible ***dv/dt*** [7] et une tension réduite sur la puissance des commutateurs.
- En utilisant les convertisseurs multi-niveaux, la fréquence de pulsation de chacun des interrupteurs est plus basse que la fréquence apparente de la tension appliquée à la charge.

Cependant, ils ne manquent pas des inconvénients : leur commande est beaucoup plus complexe et les techniques sont encore peu répandues dans l'industrie [7].

5.3 STRATEGIES DE COMMANDE ET MODELISATION DES ONDULEURS MULTI-NIVEAUX

Les stratégies de commande des onduleurs multi-niveaux sont une adaptation de celles appliquées aux onduleurs deux niveaux. Dans cette section, nous présentons celles qui sont les plus utilisées, à savoir la commande par modulation de largeur d'impulsions (PWM) et celle par modulation vectorielle (SVPWM).

5.3.1 Commande d'un onduleur multi-niveaux par la modulation PWM

5.3.1.1 Commande d'un onduleur à trois niveaux

Le principe de la commande PWM d'un onduleur NPC à deux niveaux a été déjà expliqué dans le chapitre 3. Le but est d'obtenir une tension de sortie à deux niveaux par la superposition de deux interrupteurs élémentaires alimentés chacun par une source de tension continue distincte [26]. Ce qui nous intéresse ici, c'est comment à partir de la méthode de base PWM à deux niveaux nous pouvons passer à une forme plus généralisante de plusieurs niveaux. Nous commencerons d'abord par étudier la commande d'un moteur asynchrone en utilisant la technique PWM à trois niveaux qui est la première étape des multi-niveaux ensuite j'entamerai le cas du PWM cinq niveaux, sept niveaux et enfin neuf niveaux. Dans

tous ces différents cas, nous ferons une analyse de la vitesse du moteur, du couple moteur et du Taux de Distorsion Harmonique (THD).

❖ Structure de l'onduleur de tension triphasé à trois niveaux

La structure de l'onduleur de tension en pont triphasé à trois niveaux représentée par la figure 5.1, est composée de trois demi-ponts monophasés. A partir de la source principale de tension continue E, et à l'aide d'un diviseur de tension capacitif formé par les condensateurs C1 et C2 de même capacité, on obtient deux sources secondaires de tension continue délivrant chacune un potentiel à demi tension E/2. Cette structure crée alors un point neutre O entre les condensateurs C1 et C2. Chaque demi-pont est composé de deux étages d'interrupteurs. Chaque étage comporte deux transistors IGBT en série avec un point commun relié par une diode au point neutre O. Les diodes anti-parallèles sur les transistors assurent la réversibilité des courants de la charge.

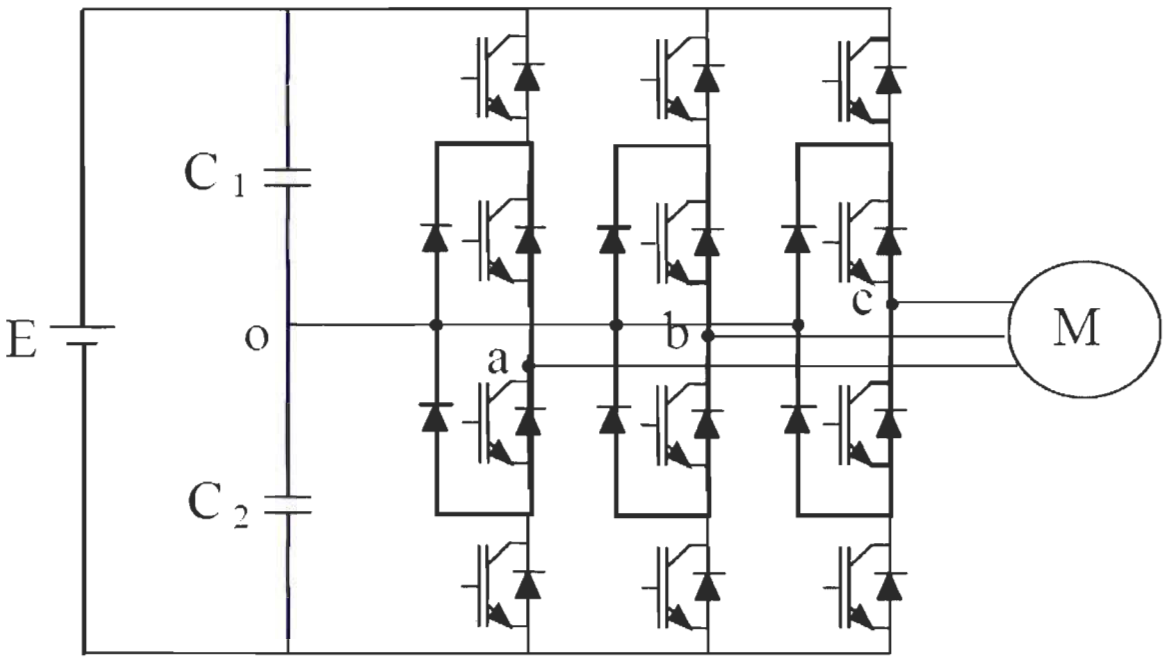


Fig. 5.1: Structure d'un onduleur à trois niveaux

❖ **Principe de fonctionnement**

Pour décrire le fonctionnement de l'onduleur triphasé de type NPC, on considère un seul bras dont la structure est représentée par la figure 5.2. Il faut déterminer les valeurs que peut prendre la tension simple V_{ao} entre la borne "a" de la charge et le point neutre O. Cette tension est entièrement définie par l'état (0 ou 1) des quatre interrupteurs K1, K2, K3 et K4 du bras. Le sens positif ou négatif des courants I_{d0}, I_{d1} et I_{d2} fixe le sens du transfert de l'énergie du convertisseur. Lorsque la source de tension est génératrice et la charge est réceptrice, le courant passe à travers les transistors. Lorsque le transfert d'énergie s'effectue de la charge vers la source d'entrée, ce sont les diodes antiparallèles qui assurent le passage du courant.

Sur les $2^4 = 16$ séquences possibles, seules trois séquences sont mises en œuvre. Toutes les autres séquences ne sont pas fonctionnelles, donc elles sont à éviter.

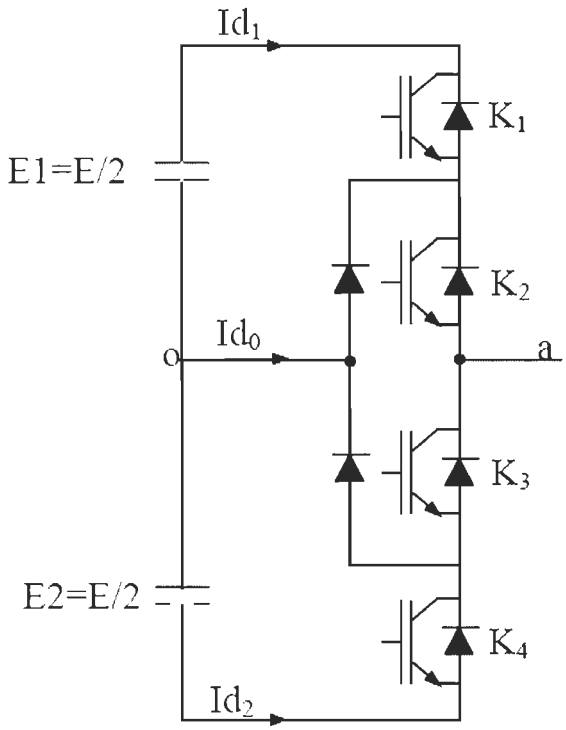


Fig. 5.2: Bras d'un onduleur triphasé à trois niveaux

En effet, elles provoquent :

Soient des court-circuits des sources de tension continue :

- court-circuit de E1 et de E2 avec les séquences [1111] et [1001],
- court-circuit de E1 avec les séquences [1110], [1000] et [1010],
- court-circuit de E2 avec les séquences [0111], [0001] et [0101] ;

Soient la déconnexion de la charge pour la séquence [0000].

Soient encore, elles ne permettent pas d'assurer la connexion de la charge au point neutre pour les séquences [0100] et [0010].

Les trois configurations du bras de l'onduleur correspondant aux trois séquences fonctionnelles sont dans le tableau suivant:

Tableau 5. 1: Séquences des vecteurs de commande du bras d'onduleur à trois niveaux

	États des interrupteurs		
Séquences	$[K_1\ K_2 K_3 K_4]$	Tension inverse des interrupteurs bloqués	Tension de sortie V_{ao}
1	[1 1 0 0]	$V_{k3}= V_{k4}= +E/2$	$V_{ao}= +E/2$
2	[0 1 1 0]	$V_{k1}= V_{k4}= +E/2$	$V_{ao}= 0$
3	[0 0 1 1]	$V_{k1}= V_{k2}= +E/2$	$V_{ao}= -E/2$

Les séquences 1, 2 et 3 vont s'enchaîner durant chaque période de la façon suivante : 1-2-3-2.

5.3.1.2 Extension de la commande PWM à des onduleurs multi-niveaux d'ordre supérieur

En se basant sur l'onduleur de tension de trois niveaux, on peut étendre l'étude à des niveaux supérieurs. Pour un onduleur à N niveaux, les nombres des différents composants et matériels nécessaires dans chaque branche sont régis par les relations suivantes: (5.1)

- Nombre de sources de tensions secondaires continues : **S = N – 1,**
- Nombre d'interrupteurs : **K = 2(N - 1),**
- Nombre de diodes de bouclage : **D = 2(N - 2)** en tenant compte des diodes des interrupteurs,
- Nombre de tension aux bornes des condensateurs : **C = E/(N - 1).**

Ainsi par exemple pour N = 5, pour avoir une tension de 220 V, on a besoin de 4 sources de tension secondaires, 8 interrupteurs et 6 diodes. Les tensions aux bornes des condensateurs sont toutes égales à 220/4. E est la tension totale du bus continu.

❖ **Algorithme de commande PWM n-niveaux**

Pour commander un onduleur à N niveaux de tension, **(N − 1)** porteuses triangulaires sont nécessaires. Les signaux triangulaires doivent toutes avoir la même fréquence f_c, et la même amplitude A_c[26]. Ils peuvent être horizontalement ou verticalement décalés. S'ils le sont horizontalement, le déphasage entre deux signaux consécutifs est donné par **2π/(N − 1)**. S'ils le sont verticalement, ils peuvent être en phase ou non et occupent une bande continue avec le même décalage vertical. Ils sont ensuite comparés au signal de référence d'amplitude A_r et de fréquence f_r. Chaque comparaison donne 1 si une porteuse est supérieure ou égale à la référence, et 0 dans le cas contraire. À la sortie du modulateur, la somme des résultats issus des comparaisons est ensuite décodée, et donne la valeur correspondant à chaque niveau de

tension. Cette modulation est aussi caractérisée par l'indice de modulation ***m*** qui est égale au rapport de l'amplitude de la tension de référence A_r à la valeur crête A_c de la porteuse : **m** = A_r/A_c .

La figure 1.3 représente ce principe d'extension.

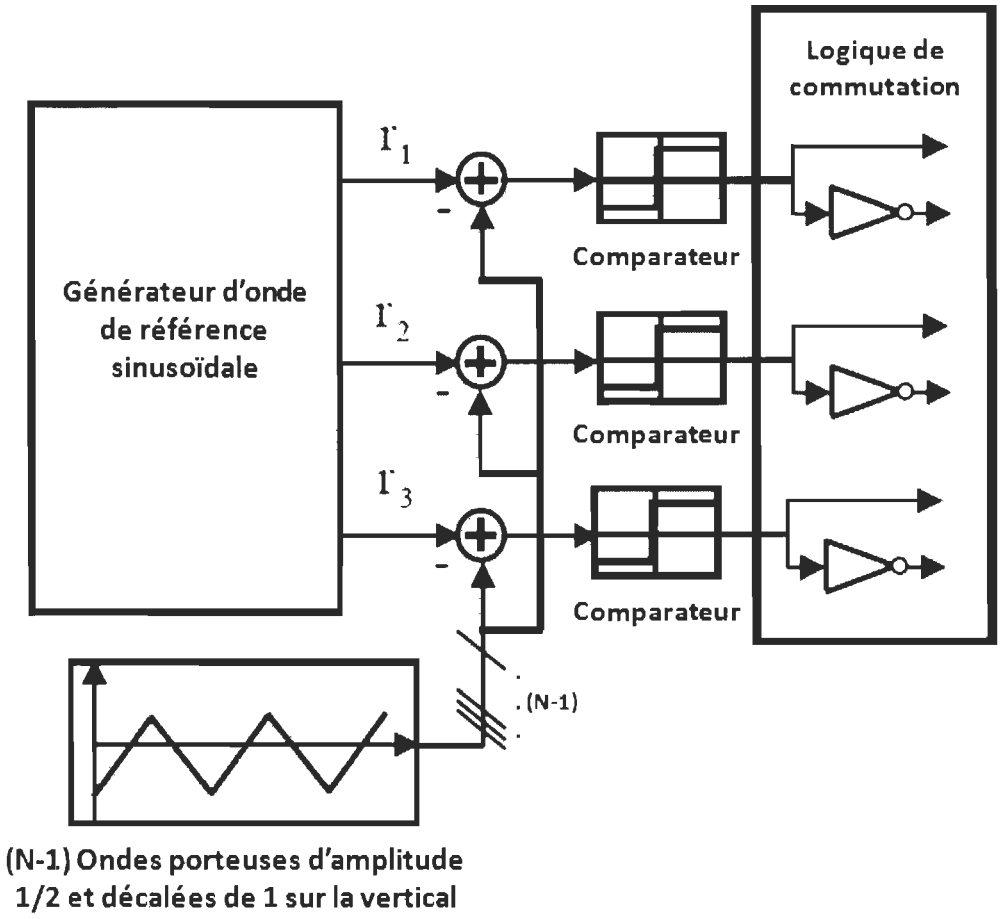


Fig. 5.3: Schéma de principe de commande PWM multi-niveaux

Par exemple pour un onduleur triphasé à sept niveaux, on utilise six porteuses en dents de scie bipolaires (V_{c11}, V_{c12}, V_{c13}, V_{c21}, V_{c22}, V_{c23}) ayant le même décalage vertical 1 l'une par rapport à l'autre.

❖ **Détermination des porteuses en dents de scie et des tensions triphasées**

➢ Les six porteuses

$$V_{c11} = A_c\left(\frac{2.t}{T_c}\right); 0 \leq t < T_c \quad (5.2)$$

$$V_{c12} = A_c\left(\frac{2.t}{T_c}\right) + 1;\ 0 \leq t < T_c$$

$$V_{c13} = A_c\left(\frac{2.t}{T_c}\right) + 2;\ 0 \leq t < T_c$$

$$V_{c21} = A_c\left(\frac{2.t}{T_c}\right) - 1;\ 0 \leq t < T_c$$

$$V_{c22} = A_c\left(\frac{2.t}{T_c}\right) - 2;\ 0 \leq t < T_c$$

$$V_{c23} = A_c\left(\frac{2.t}{T_c}\right) - 3;\ 0 \leq t < T_c$$

➢ Les trois tensions triphasées équilibrées sont :

$$V_{ref1}(t) = A_r.Sin(2\pi f_r t) \quad (5.3)$$

$$V_{ref2}(t) = A_r.Sin\left(2\pi f_r t - \frac{2\pi}{3}\right)$$

$$V_{ref3}(t) = A_r.Sin\left(2\pi f_r t - \frac{4\pi}{3}\right)$$

❖ **Calcul du comparateur**

$$\begin{cases} V_{refk} \geq V_{c11} => T_{k11} = 1 \\ V_{refk} < V_{c11} => T_{k11} = 0 \\ \quad ou \\ V_{refk} \leq V_{c21} => T_{k21} = 1 \\ V_{refk} > V_{c21} => T_{k21} = 0 \end{cases} \quad (5.4)$$

$$\begin{cases} V_{refk} \geq V_{c12} => T_{k12} = 1 \\ V_{refk} < V_{c12} => T_{k12} = 0 \\ ou \\ V_{refk} \leq V_{c22} => T_{k22} = 1 \\ V_{refk} > V_{c22} => T_{k22} = 0 \end{cases}$$

$$\begin{cases} V_{refk} \geq V_{c13} => T_{k13} = 1 \\ V_{refk} < V_{c13} => T_{k13} = 0 \\ ou \\ V_{refk} \leq V_{c23} => T_{k23} = 1 \\ V_{refk} > V_{c23} => T_{k23} = 0 \end{cases}$$

❖ **Calcul du sommateur**

$$T_k = (T_{k11} - T_{k21}) + (T_{k12} - T_{k22}) + (T_{k13} - T_{k23}) \quad (5.5)$$

❖ **Calcul des tensions et des vecteurs de commande**

$$Si\ T_k = 3 => \begin{cases} G_k = [1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0] \\ V_k = \left(\frac{1}{2}\right).V_{dc} \end{cases} \quad (5.6)$$

$$Si\ T_k = 2 => \begin{cases} G_k = [0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0] \\ V_k = \left(\frac{1}{3}\right).V_{dc} \end{cases}$$

$$Si\ T_k = 1 => \begin{cases} G_k = [0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0] \\ V_k = \left(\frac{1}{6}\right).V_{dc} \end{cases}$$

$$Si\ T_k = 0 => \begin{cases} G_k = [0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0] \\ V_k = 0 \end{cases}$$

$$Si\ T_k = -1 => \begin{cases} G_k = [0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0] \\ V_k = -\left(\frac{1}{6}\right).V_{dc} \end{cases}$$

$$Si\ T_k = -2 => \begin{cases} G_k = [0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0] \\ V_k = -\left(\frac{1}{3}\right).V_{dc} \end{cases}$$

$$Si\ T_k = -3 => \begin{cases} G_k = [0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1] \\ V_k = -\left(\frac{1}{2}\right).V_{dc} \end{cases}$$

Avec : V_{c11} ... V_{c23} : Porteuses en dents de scie,

V_{refk}: Tensions de référence triphasées,

T_k: Sortie du sommateur dans le comparateur,

G_k: Vecteur de commande des gâchettes,

V_k: Tensions de sortie de l'onduleur,

V_{dc} = E : Tensions d'alimentation de l'onduleur.

Les résultats des tensions de sortie de l'onduleur seront présentés dans la partie simulation de ce chapitre.

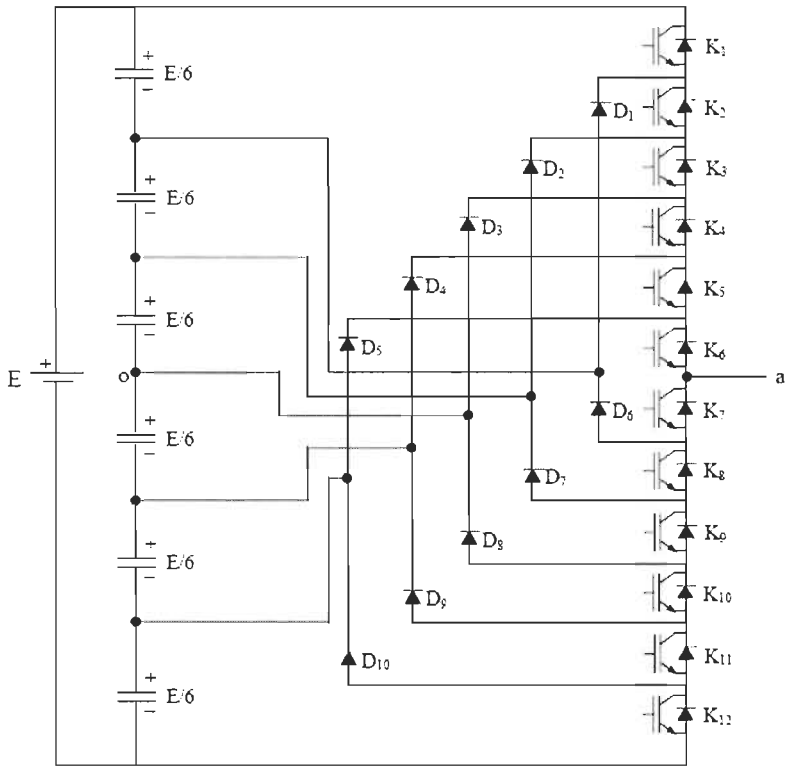


Fig. 5.4: Onduleur de type à sept niveaux

A titre d'exemple à la figure 5.4 nous avons représenté ce principe d'extension pour un onduleur à sept niveaux de tension (fig. 5.5), ainsi que les trois tensions triphasées multi-niveaux à la sortie de l'onduleur triphasé (fig. 5.6).

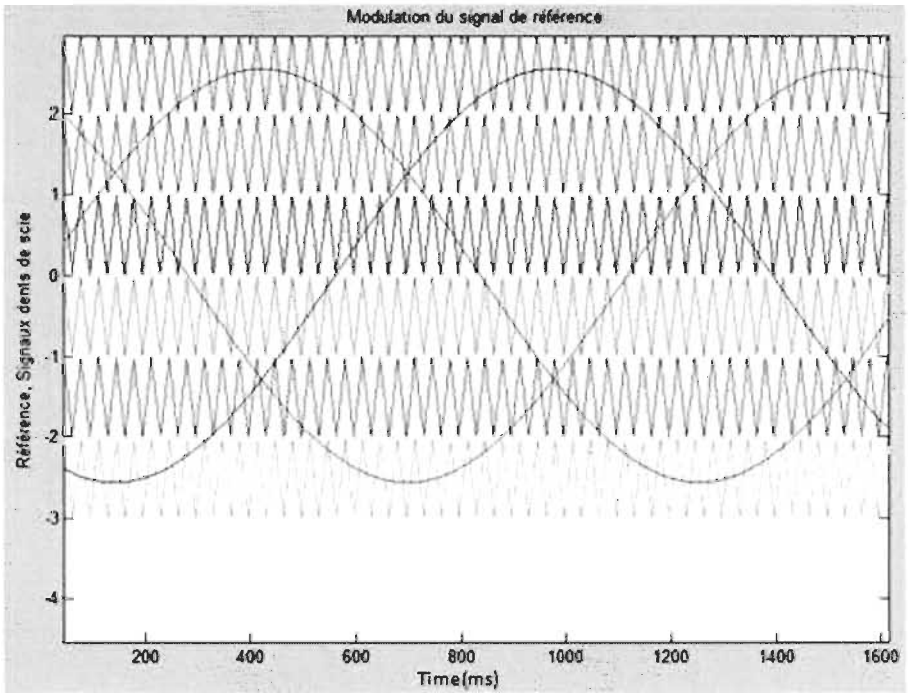


Fig. 5.5: Modélisation PWM sept niveaux

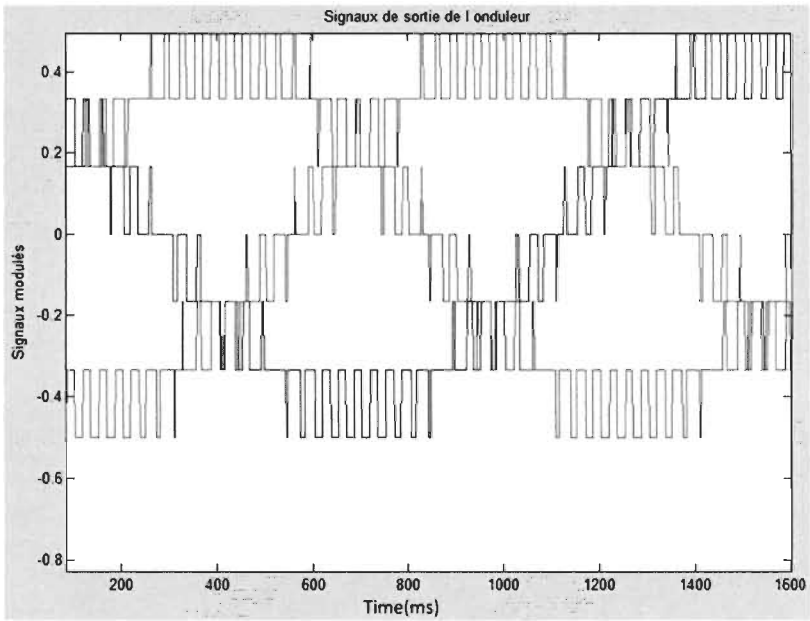


Fig. 5.6: Tension de sortie de l'onduleur à sept niveaux

Tableau 5. 2: Séquences des vecteurs commande du bras d'onduleur de 3 à 11 niveaux

Niveaux	Séquences	États des interrupteurs $[K_1K_2K_3K_4]$	Tension de sortie V_{ao}
3	1	[1 1 0 0]	V_{ao}= + (1/2).E
	2	[0 1 1 0]	V_{ao}= 0
	3	[0 0 1 1]	V_{ao}= - (1/2).E
5	1	[1 1 1 1 0 0 0 0]	V_{ao}= + (1/2).E
	2	[0 1 1 1 1 0 0 0]	V_{ao}= + (1/4).E
	3	[0 0 1 1 1 1 0 0]	V_{ao}= 0
	4	[0 0 0 1 1 1 1 0]	V_{ao}= - (1/4).E
	5	[0 0 0 0 1 1 1 1]	V_{ao}= - (1/2).E
7	1	[1 1 1 1 1 1 0 0 0 0 0 0]	V_{ao}= + (1/2).E
	2	[0 1 1 1 1 1 1 0 0 0 0 0]	V_{ao}= + (1/3).E
	3	[0 0 1 1 1 1 1 1 0 0 0 0]	V_{ao}= + (1/6).E
	4	[0 0 0 1 1 1 1 1 1 0 0 0]	V_{ao}= 0
	5	[0 0 0 0 1 1 1 1 1 1 0 0]	V_{ao}= - (1/6).E
	6	[0 0 0 0 0 1 1 1 1 1 1 0]	V_{ao}= - (1/3).E
	7	[0 0 0 0 0 0 1 1 1 1 1 1]	V_{ao}= - (1/2).E
9	1	[1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0]	V_{ao}= + (4/8).E
	2	[0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0]	V_{ao}= + (3/8).E
	3	[0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0]	V_{ao}= + (2/8).E
	4	[0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0]	V_{ao}= + (1/8).E
	5	[0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0]	V_{ao}= 0
	6	[0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0]	V_{ao}= - (1/8).E
	7	[0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0]	V_{ao}= - (2/8).E
	8	[0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0]	V_{ao}= - (3/8).E
	9	[0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1]	V_{ao}= - (4/8).E

11	1	[1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0]	V_{ao}= + (5/10).E
	2	[0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0]	V_{ao}= + (4/10).E
	3	[0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0]	V_{ao}= + (3/10).E
	4	[0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0]	V_{ao}= + (2/10).E
	5	[0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0]	V_{ao}= + (1/10).E
	6	[0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0]	V_{ao}= 0
	7	[0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0]	V_{ao}= - (1/10).E
	8	[0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0]	V_{ao}= - (2/10).E
	9	[0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0]	V_{ao}= - (3/10).E
	10	[0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0]	V_{ao}= - (4/10).E
	11	[0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1]	V_{ao}= - (5/10).E

5.3.2 Proposition de méthode générale de commande SVPWM pour onduleurs multi-niveaux

En termes de rendement, la technique SVPWM fait l'unanimité des chercheurs par rapport à celle de la PWM. Car les tensions maximales fournies par un onduleur commandé avec la technique SVPWM sont supérieures à celui commandé avec celle de la PWM. On a :

$$V_{max_{(PWM)}} = V_{dc}/2 \text{ et } V_{max_{(SVPWM)}} = V_{dc}/\sqrt{3} \quad (5.7)$$

Ce qui implique que : $V_{max_{(SVPWM)}} > V_{max_{(PWM)}}$

Cela signifie qu'avec la SVPWM, on est capable d'avoir une tension de 15% de V_{dc} de plus que la PWM. Cependant l'algorithme SVPWM est plus complexe que celui de la PWM à cause du nombre élevé des états de commutation.

Ainsi dans cette partie, nous allons aborder cette nouvelle technique SVPWM plus performante que celle de la PWM. Un algorithme général du SVPWM pour les onduleurs multi-niveaux basée sur nos lectures [27], [28], [29], [30], [31], [32] sera présenté. Cet algorithme est constitué de cinq étapes essentielles. Au niveau de chaque étape, il existe

plusieurs méthodes de calcul. Nous avons choisi celles qui nous semblent être les meilleures après plusieurs comparaisons. Les étapes sont les suivantes :

- Détermination du vecteur de tension de référence
- Calcul du secteur
- Calcul de la région
- Calcul des temps de commutation
- Calcul des séquences de commutation
- Génération des signaux PWM

Depuis que la méthode SVPWM multi-niveaux utilise la modélisation deux niveaux pour calculer les temps de commutation, l'estimation de ces derniers devient moins difficile. Une modélisation multi-niveaux est expliquée, et les résultats de simulation sont donnés pour onduleurs deux et trois-niveaux.

5.3.2.1 SVPWM à trois niveaux

❖ **Calcul du vecteur de référence**

Nous avons montré plus haut le circuit principal de l'onduleur triphasé à trois niveaux (figure 5.1). Chaque branche est composée de quatre commutateurs à trois états de commutation qui peuvent être représentés par P, O, N listé dans le tableau 5.3.

D'après ce qui précède, nous comprenons alors qu'il existe 27 vecteurs de tension de référence dans le diagramme vectoriel d'un onduleur triphasé à trois niveaux dont chacun peut être représenté sous la forme vectorielle suivante comme vecteur de référence :

$$\overrightarrow{V_{ref}} = \frac{2}{3}\left(\overrightarrow{v_{a0}}.e^{i0} + \overrightarrow{v_{b0}}.e^{i\frac{2\pi}{3}} + \overrightarrow{v_{c0}}.e^{i\frac{-2\pi}{3}}\right) \quad (5.8)$$

Avec $\overrightarrow{v_{a0}}$, $\overrightarrow{v_{b0}}$, $\overrightarrow{v_{c0}}$ sont les tensions de références des phases a, b et c du stator.

Les 27 vecteurs constituent le diagramme vecteur espace de l'onduleur à trois niveaux comme représentés à la figure 5.7. Il y'a 24 vecteurs actifs dont 12 vecteurs courts, 6 vecteurs moyens et 6 vecteurs longs, et les trois restants sont des vecteurs nuls (PPP, OOO NNN). Ils convergent tous au centre de l'hexagone.

Conformément au principe de transformation du triphasé au biphasé, les trois vecteurs de tensions de référence ($\overrightarrow{v_{a0}}$, $\overrightarrow{v_{b0}}$, $\overrightarrow{v_{c0}}$) peuvent être transformés en ($\overrightarrow{v_{\alpha}}$, $\overrightarrow{v_{\beta}}$,) comme suit :

$$\overrightarrow{V_{ref}} = \begin{bmatrix} \overrightarrow{v_{\alpha}} \\ \overrightarrow{v_{\beta}} \end{bmatrix} = \frac{2}{3} \cdot \begin{bmatrix} 1 & \frac{-1}{2} & \frac{-1}{2} \\ 0 & \frac{\sqrt{3}}{2} & \frac{-\sqrt{3}}{2} \end{bmatrix} \cdot \begin{bmatrix} \overrightarrow{v_{a0}} \\ \overrightarrow{v_{b0}} \\ \overrightarrow{v_{c0}} \end{bmatrix} \quad (5.9)$$

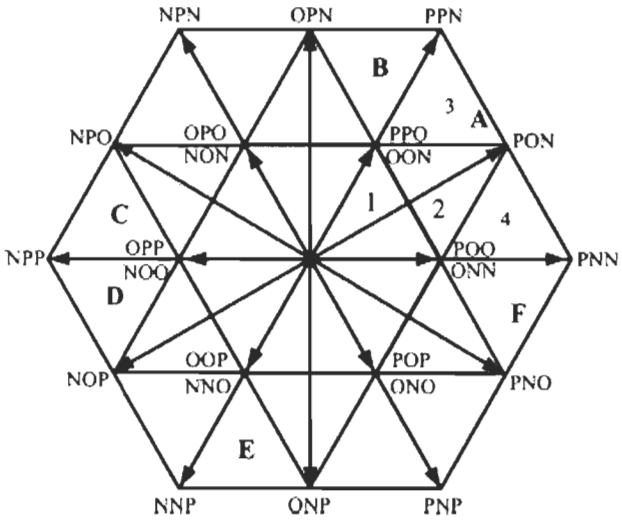


Fig. 5.7: Diagramme vectoriel d'onduleur à trois niveaux

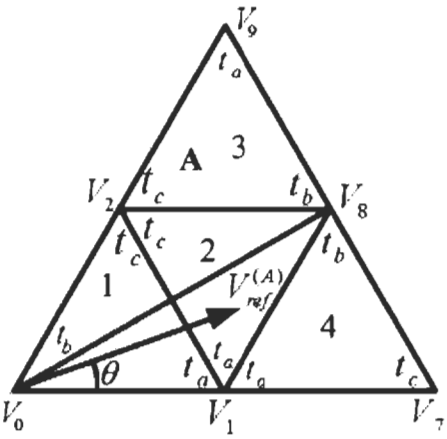


Fig. 5.8: Tension de référence dans le secteur A

$$\overrightarrow{V_{ref}} = \vec{v}.e^{i\theta} \quad (5.10)$$

$\overrightarrow{v_{\alpha}}$ et $\overrightarrow{v_{\beta}}$ sont les coordonnées cartésiennes de la tension de référence $\overrightarrow{V_{ref}}$.

θest l'angle de phase de $\overrightarrow{V_{ref}}$.

❖ **Calcul du secteur**

L'hexagone du diagramme vecteur d'espace de l'onduleur à trois niveaux peut être subdivisé en 6 secteurs (A à F) dont chacun à 4 régions (1 à 4). Ce qui donne un total de 24 régions.

Les secteurs peuvent être déterminés de la façon suivante :

$$S = \begin{cases} 1 & si\ 0 < \theta < \frac{\pi}{3} \\ 2 & si\ \frac{\pi}{3} < \theta < \frac{2\pi}{3} \\ 3 & si\ \frac{2\pi}{3} < \theta < \pi \\ 4 & si\ \pi < \theta < \frac{4\pi}{3} \\ 5 & si\ \frac{4\pi}{3} < \theta < \frac{5\pi}{3} \\ 6 & si\ \frac{5\pi}{3} < \theta < 0 \end{cases} \tag{5.11}$$

❖ **Calcul de la région**

À partir de la figure 5.9, m_2 et m_1 peuvent être calculés comme suit :

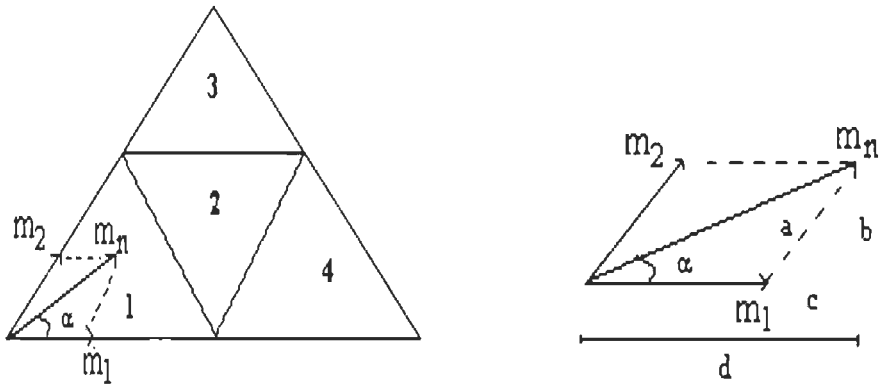


Fig. 5. 9: Diagramme vecteur espace pour m_1 et m_2 dans le secteur A

$$a = m_2 = \frac{b}{Sin(\frac{\pi}{3})} = \frac{2}{\sqrt{3}}.b = \frac{2}{\sqrt{3}}.m_n.Sin(\alpha) \tag{5.12}$$

$$m_1 = m_n.Cos(\alpha) - \left(\frac{2}{\sqrt{3}}.m_n.Sin(\alpha)\right).Cos\left(\frac{\pi}{3}\right) \tag{5.13}$$

$$m_1 = m_n.\left(Cos(\alpha) - \frac{Sin(\alpha)}{\sqrt{3}}\right)$$

Et ensuite $\overrightarrow{V_{ref}}$ est dans la région:

$$Région = \begin{cases} 1 & si\ m_1, m_2\ et\ (m_1 + m_2) < \theta.5 \\ 2 & si\ m_1 > \theta.5 \\ 3 & si\ m_2 > \theta.5 \\ 4 & si\ m_1\ et\ m_2 < \theta.5\ et\ (m_1 + m_2) > \theta.5 \end{cases} \quad (5.14)$$

❖ **Calcul des temps de commutation**

La symétrie du système triphasé, nous permet de réduire l'étude au cas général d'un secteur de $\frac{\pi}{3}$. On se place alors dans le cas où le vecteur de référence $\overrightarrow{V_{ref}}$ est situé dans le secteur A. Alors, considérant que le vecteur de tension de référence $\overrightarrow{V_{ref}}$ reste dans la région 2, dans ce cas, la tension de référence $\overrightarrow{V_{ref}}$ est reconstituée en faisant une moyenne temporelle des tensions $\overrightarrow{V_1}$, $\overrightarrow{V_2}$et $\overrightarrow{V_8}$ comme illustré à la figure 5.8.

Pour une fréquence de commutation T_s suffisamment élevée le vecteur d'espace de référence $\overrightarrow{V_{ref}}$est considéré constant pendant un cycle de commutation. Tenant compte que $\overrightarrow{V_1}$, $\overrightarrow{V_2}$et $\overrightarrow{V_8}$ sont constants, il s'en suit pour un cycle de commutation :

$$\begin{cases} \overrightarrow{V_{ref}} = \overrightarrow{V_1}.t_a + \overrightarrow{V_8}.t_b + \overrightarrow{V_2}.t_c \\ T_s = t_a + t_b + t_c \end{cases} \quad (5.15)$$

avec : t_a, t_b, t_c : temps alloués respectivement aux vecteurs $\overrightarrow{V_1}$, $\overrightarrow{V_8}$et $\overrightarrow{V_2}$.

À partir de ces expressions précédentes, les temps de commutation des vecteurs de tensions peuvent être calculés comme suit :

$$\begin{cases} t_a = T_s - 2k.\sin(\theta) \\ t_b = 2k.\sin\left(\frac{\pi}{3} + \theta\right) - T_s \\ t_a = T_s - 2k.\sin\left(\frac{\pi}{3} - \theta\right) \end{cases} \qquad \text{Où k} = \frac{4\sqrt{3}}{3}\left(\frac{V_{ref}}{E}\right).T_s \quad (5.16)$$

En suivant la même procédure, les temps de commutation dans les autres régions du secteur A peuvent être obtenus comme montré dans le Tableau 5.3.

Tableau 5. 3: Calcul temps de commutation

Espaces temps / Région	t_a	t_b	t_c
1	$2k.\sin(\frac{\pi}{3}-\theta)$	$T_s - 2k.\sin(\frac{\pi}{3}+\theta)$	$2k.\sin(\theta)$
2	$T_s - 2k.\sin(\theta)$	$2k.\sin\left(\frac{\pi}{3}+\theta\right) - T_s$	$T_s - 2k.\sin(\frac{\pi}{3}-\theta)$
3	$2k.\sin(\theta) - T_s$	$2k.\sin(\frac{\pi}{3}-\theta)$	$2T_s - 2k.\sin(\frac{\pi}{3}+\theta)$
4	$2T_s - 2k.\sin(\frac{\pi}{3}+\theta)$	$2k.\sin(\theta)$	$2k.\sin\left(\frac{\pi}{3}-\theta\right) - T_s$

❖ **Détermination des séquences des temps de commutation**

Une fois que les espaces temps (on times) ont été calculés, les séquences de commutation doivent aussi être déterminées. Cependant les onduleurs ont certains états de commutation qui sont redondants. Cela signifie que les séquences de commutation doivent être déterminées de manière intelligente, de manière à éviter les redondances afin d'optimiser le système et minimiser au maximum le taux de distorsion harmonique. Ainsi dans cette méthode de calcul du SVPWM multi-niveaux, tous les temps de commutation sont ordonnés de façon à former des séquences de commutation optimale. Les commutations se font en changeant l'état d'un seul commutateur à la fois.

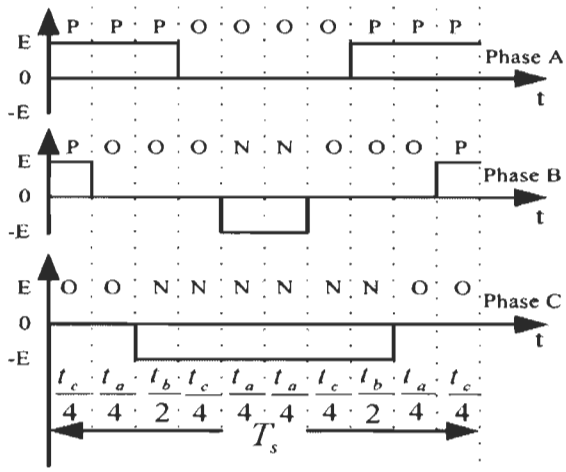


Fig. 5.10: Ordre des séquences de commutation symétrique

Nous obtenons les séquences suivantes selon la région où se trouve le vecteur de référence :

Région 1 : PPO-POO-OOO-OON-ONN et inverse

Région 2 : PPO-POO-PON-OON-ONN et inverse

Région 3 : PPO-PPN-PON-OON et inverse

Région 4 : POO-PON-PNN-ONN et inverse

❖ **Génération des signaux symétriques PWM**

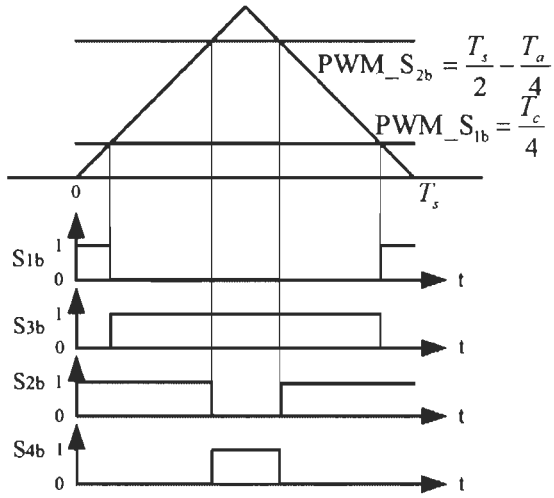


Fig. 5.11: Temps de commutation pour quatre commutateurs

Les signaux PWM sont générés de manière symétrique comme l'illustre la figure 5.10. Ils peuvent être déterminés en utilisant un générateur PWM. En prenant par exemple la phase B à la figure 5.10, l'onde PWM trois niveaux, peut être décomposée en la forme de deux ondes PWM à deux niveaux qui sont facilement générables en utilisant deux générateurs PWM comme le montre la figure 5.11. À partir des séquences de commutation symétriquement ordonnées, nous déterminons ainsi les temps de commutation alloués à chaque commutateur dans le secteur A (voir Tableau 5.4).

La même procédure du secteur A ci-dessus peut être appliquée aux autres secteurs. Et c'est le même calcul pour n'importe quel vecteur de référence qui se situe dans l'une des 24 régions.

Tableau 5. 4: Temps de commutation de la branche supérieur du secteur A

Régions / Temps	1	2	3	4
PWM_S_{1a}	$\frac{t_c}{4}+\frac{t_a}{4}$	$\frac{t_c}{4}+\frac{t_a}{4}+\frac{t_b}{2}$	$\frac{T_s}{2}-\frac{t_c}{4}$	$\frac{t_c}{4}+\frac{t_a}{4}$
PWM_S_{2a}	$\frac{T_s}{2}$	$\frac{T_s}{2}$	$\frac{T_s}{2}$	$\frac{T_s}{2}$
PWM_S_{1b}	$\frac{t_c}{4}$	$\frac{t_c}{4}$	$\frac{t_c}{4}+\frac{t_a}{4}$	0
PWM_S_{2b}	$\frac{t_c}{4}+\frac{t_a}{4}+\frac{t_b}{2}$	$\frac{t_c}{4}+\frac{t_a}{4}$	$\frac{t_c}{4}+\frac{t_a}{4}$	$\frac{t_c}{4}+\frac{t_a}{4}$
PWM_S_{1c}	0	0	0	0
PWM_S_{2c}	$\frac{T_s}{2}-\frac{t_a}{4}-\frac{t_c}{4}$	$\frac{t_c}{4}+\frac{t_a}{4}$	$\frac{t_c}{4}$	$\frac{t_a}{4}$

Organigramme général de l'algorithme SVPWM multi-niveaux

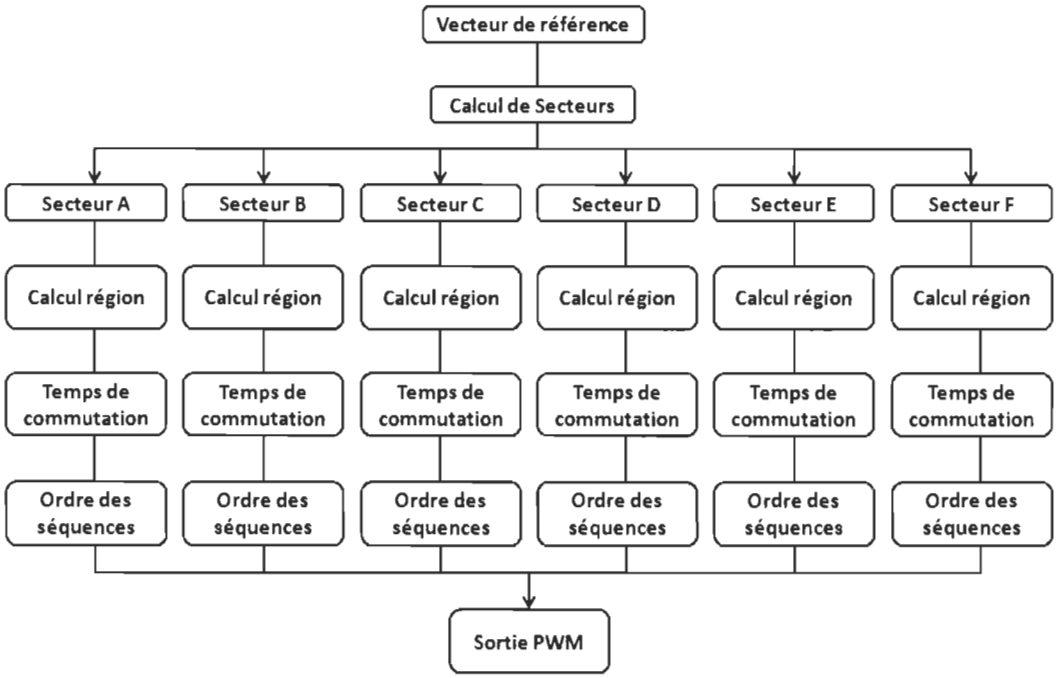


Fig. 5.12: Organigramme algorithme SVPWM

5.3.2.2 Extension de la commande SVPWM aux onduleurs multi-niveaux

Il est possible d'étendre le principe de commande par modulation vectorielle aux onduleurs à niveaux multiples en suivant la même procédure de modulation pour la commande d'onduleur à trois niveaux précédemment expliquée. Cependant un certain nombres d'éléments du diagramme de vecteur d'espace doivent être déterminés d'abord. À savoir : le nombre de vecteurs de tension, le nombre de secteurs, le nombre de triangles par secteur, le nombre de régions par secteur et le nombre de commutations.

Dans un onduleur à n niveaux, ces éléments du diagramme de vecteur d'espace pour le calcul des signaux PWM peuvent être déterminés à partir des équations suivantes :

- Nombre de vecteurs de tension : $3n^2 - 3n + 1$ (5.16)
- Nombre de secteurs : 6
- Nombre de régions par secteur : $(n-1)^2$ triangles ;
- Nombre d'états de commutation : $(n)^3$

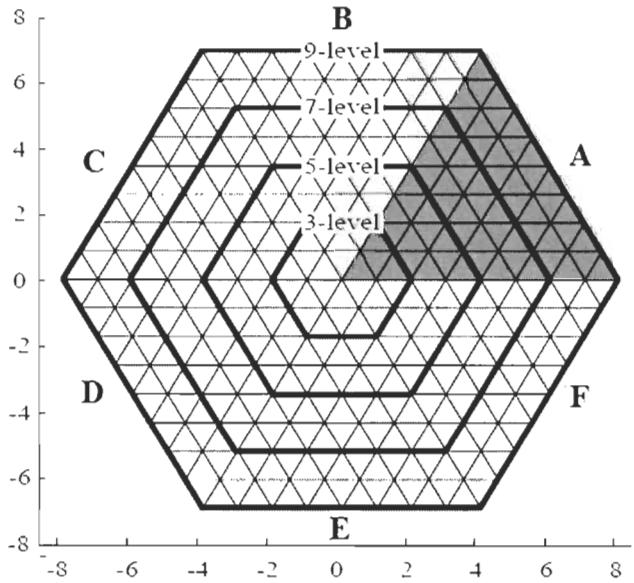


Fig. 5.13: Vecteurs de tension d'onduleurs 3, 5, 7 et 9 niveaux

La figure 5.13 représente le diagramme vectoriel SVPWM pour la commande d'un onduleur à 2, 3, 5, 7 et 9 niveaux. Par exemple pour un onduleur à 9 niveaux, on a au total 64 régions.

5.4 SIMULATION ET VALIDATION

Dans la troisième section de ce chapitre, nous avons présenté les algorithmes des deux stratégies de modulation les plus utilisées pour la commande des onduleurs multi-niveaux, à savoir la modulation PWM et celle du SVPWM. Maintenant, nous allons montrer dans cette quatrième section, l'efficacité de la modélisation multi-niveaux par rapport à la modélisation de base de deux niveaux. Ensuite étudier le comportement du système de commande multi-niveaux à vide et en charge puis au démarrage suivi de freinage par court-circuit. Enfin une comparaison du Taux de Distorsion Harmonique (THD) selon les niveaux sera effectuée et des résultats de simulation dans l'environnent Matlab/Simulink et ModelSim seront présentés.

5.4.1 Commande d'un onduleur multi-niveaux PWM

5.4.1.1 Modélisation

Compte tenu du fait que le seul onduleur multi-niveaux qui existe dans SimPower System est celui de trois niveaux, nous avons réalisé nos propres modèles d'onduleurs 3, 5, 7 et 9 niveaux à base des blocs de Matlab/Simulink. Après comparaison des résultats de simulation des deux types d'onduleur (celui de trois niveaux en Matlab/Simulink versus celui de SimPower System), nous n'avons trouvé aucun changement significatif.

La figure 5.14 représente le modèle Simulink de la commande d'onduleur à 7 niveaux. Il est constitué de 7 blocs éssentiels :

- PWM 7 niveaux : calcule les signaux PWM 7 niveaux (fig. 5.15)
- Transformateur abc= » dq : transformation les tensions triphasées à biphasées
- Flux et courants : calcule le flux et les courants statoriques
- Couple électromagnetique : calcule le couple électromagnétique
- Partie mécanique : calcule la vitesse rotorique

- Courants triphasés : transforme les courants statoriques calculés en triphasés
- Couple de la charge : calcule le couple appliqué à la charge

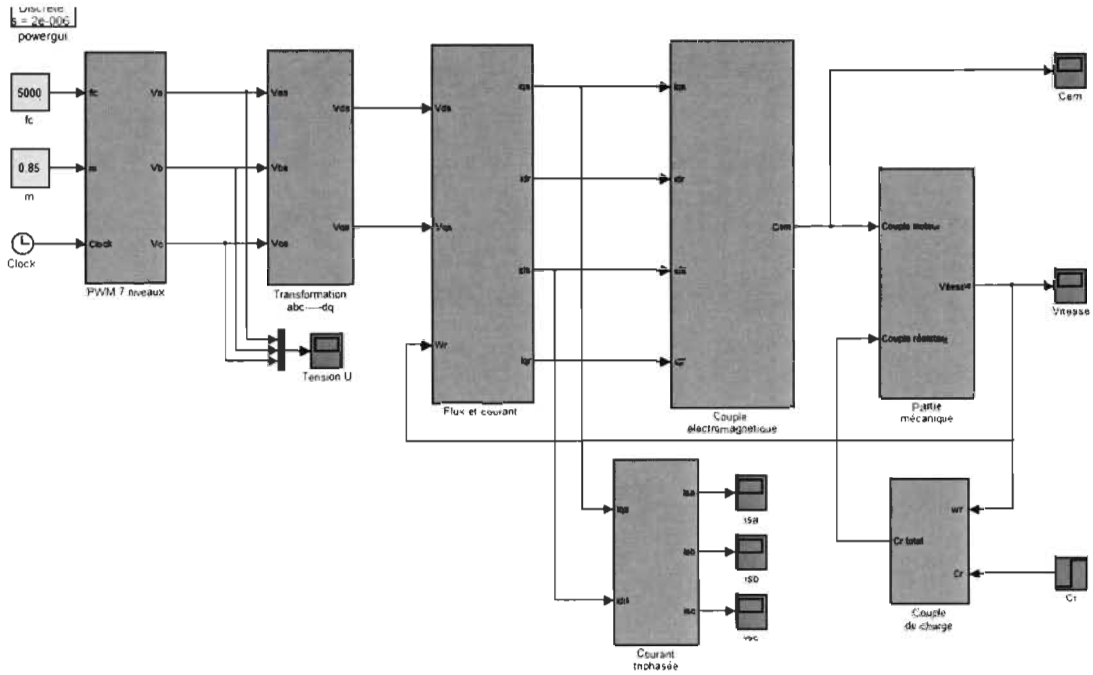


Fig. 5.14: Modélisation et commande d'onduleur à 7 niveaux

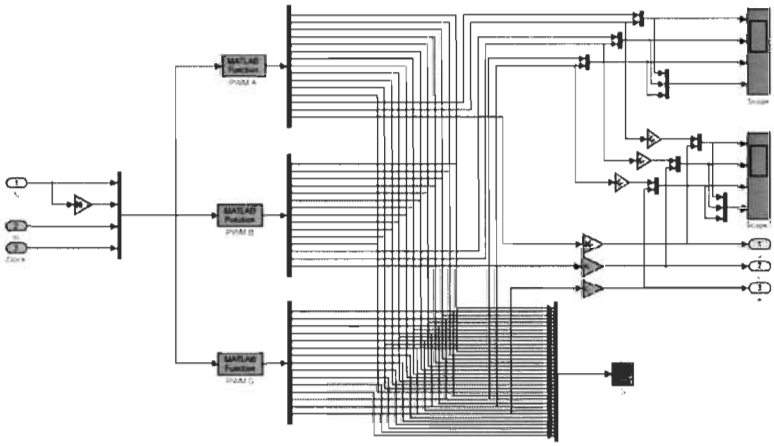
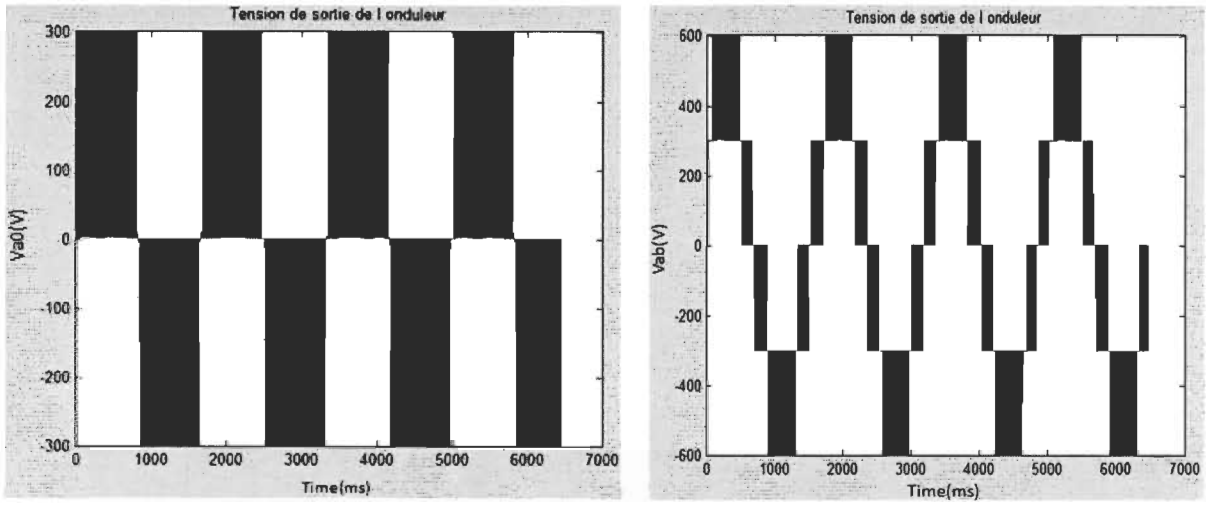


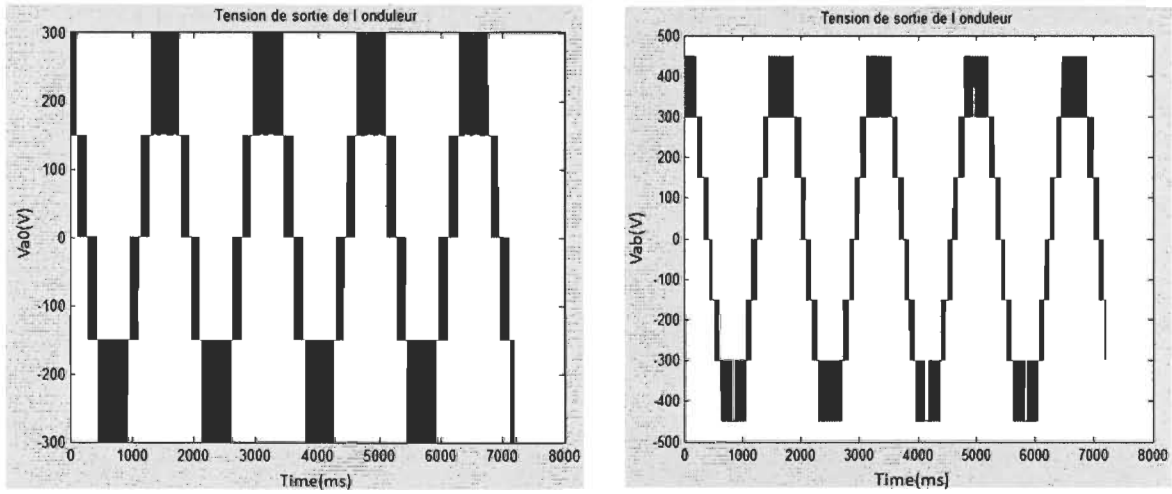
Fig. 5.15: Générateur de signaux PWM 7 niveaux

5.4.1.2 Résultats tension de sortie des onduleurs multi-niveaux

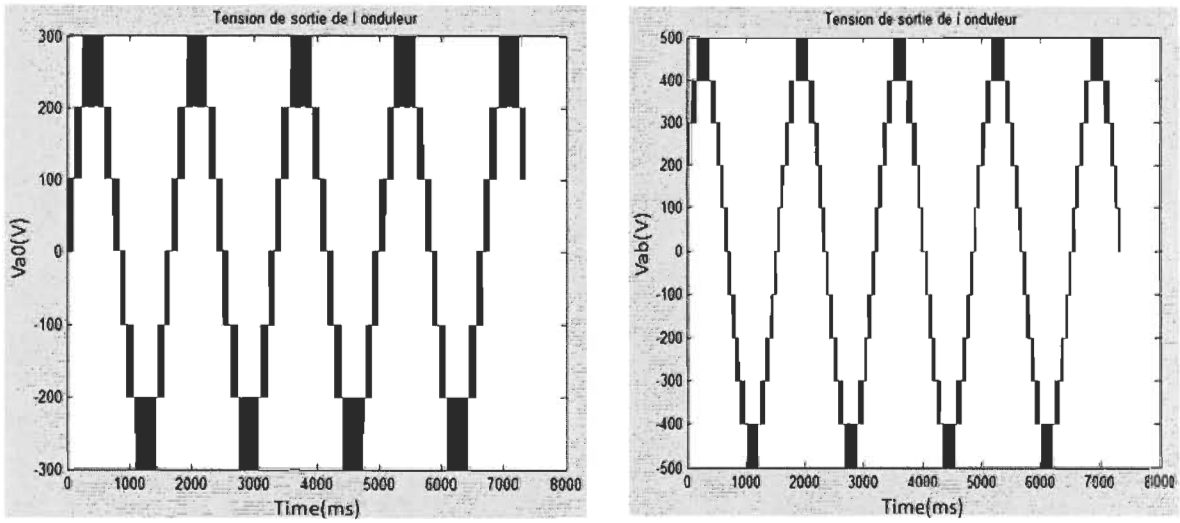
Les figures ci-dessous (Fig. 5.16) illustre les tensions entre la phase A et le point neutre 0 (V_{ao}), et les tensions de phase (V_{ab}) à la sortie des onduleurs de niveaux 3, 5, 7, 9.

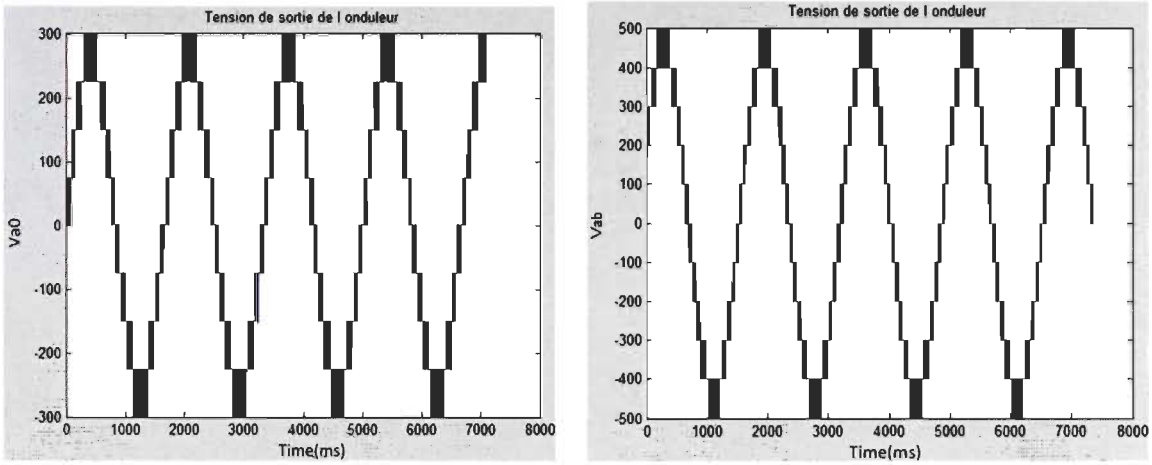


a. Tension de sortie V_{ao} et V_{ab} PWM à 3 niveaux



b. Tension de sortie V_{ao} et V_{ab} PWM à 5 niveaux

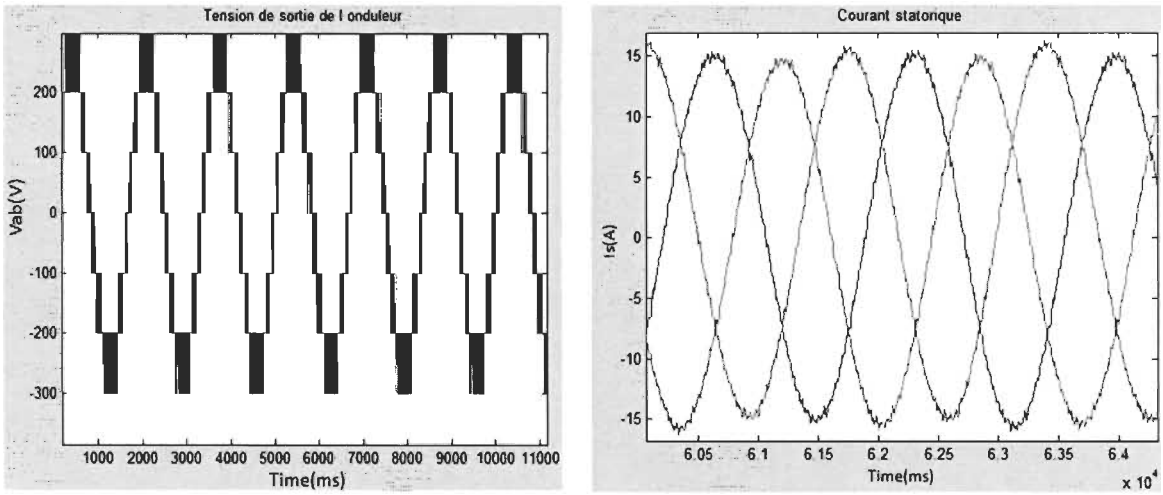




d : Tension de sortie V_{ao} et V_{ab}PWM à 9 niveaux

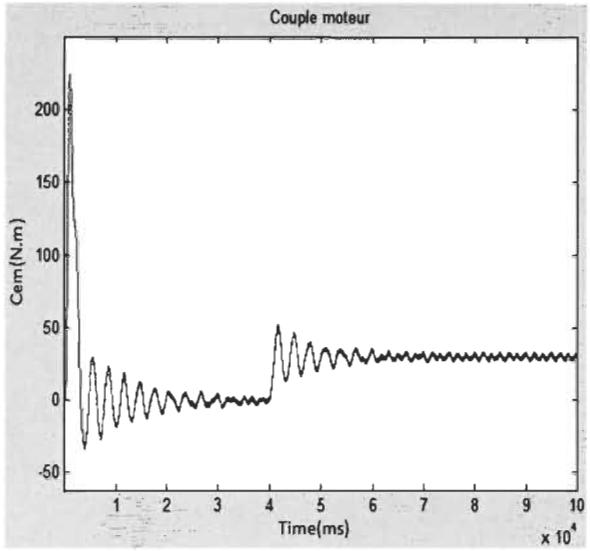
Fig. 5.16: Tension de sortie V_ab PWM 3, 5, 7, 9 niveaux

5.4.1.3 Contrôle de couple moteur : résultats au démarrage à vide et en charge

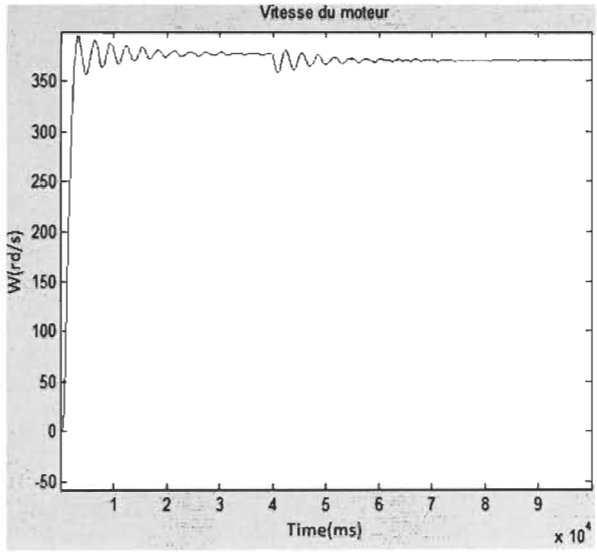


a. Tension de sortie V_ao PWM à 7 niveaux

b. Courant statorique i_s PWM à 7 niveaux



c. Couple C_{em} du moteur PWM à 7 niveaux



d. Vitesse W du roteur PWM à 7 niveaux

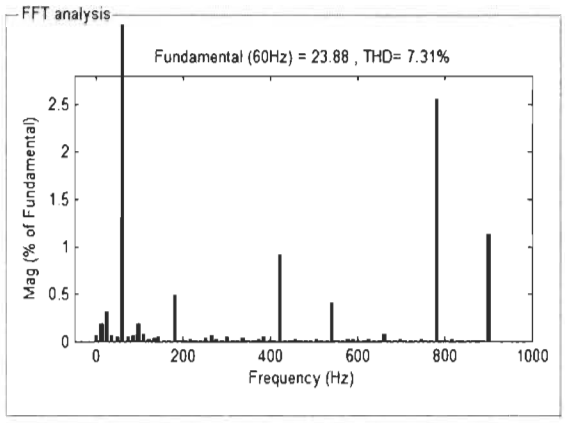
Fig. 5.17: Résultats PWM 7 niveaux au démarrage à vide et en charge

5.4.2 Études de performance des onduleurs multi-niveaux

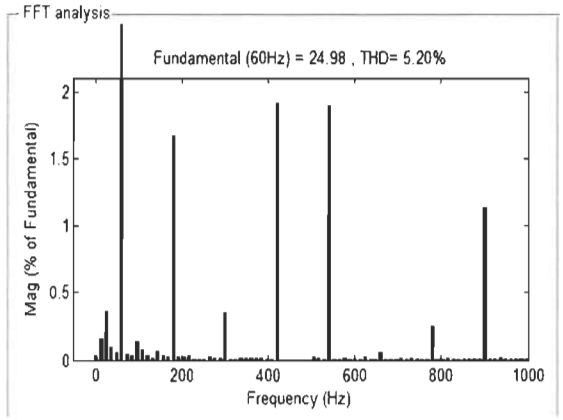
Dans cette section, nous allons comparer les performances des onduleurs multi-niveaux au cas idéal (onde sinusoïdale pure de pulsation w1) en calculant le spectre du signal généré. Le but est de diminuer le plus possible l'amplitude des harmoniques de rang faible car ce sont elles qui génèrent les courants les plus importants. S'agissant de celles de rang élevé, elles sont faciles à filtrer étant donné qu'un onduleur est toujours suivi d'un filtre passe-bas.

Cela veut dire que l'amélioration des performances des convertisseurs multi-niveaux ne peut se faire qu'en termes d'annulation ou de réduction des harmoniques de rang élevé. La qualité de l'onde de tension obtenue sera évaluée par le THD (Total Harmonic Distortion) ramené au fondamental (THD idéal = 0%).

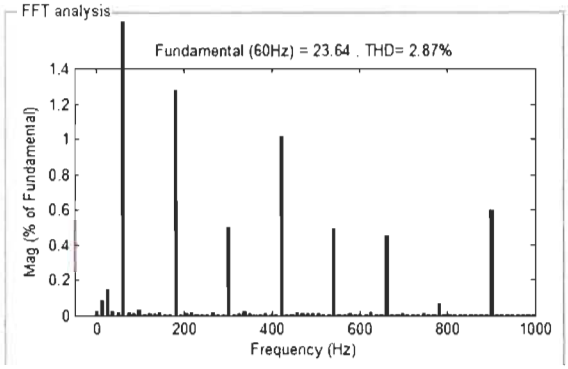
Ainsi pour une tension d'alimentation continue V_{dc}=400V, f_c=6k Hz, f_r=60 Hz, C_r=40 N.m, nous obtenons les THD suivants, présentés à la figure 5.18 en fonction du nombre de niveaux.



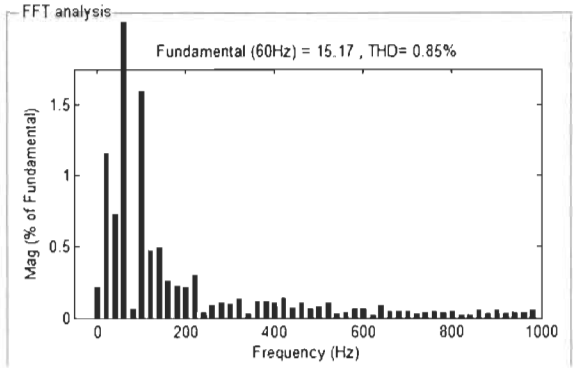
a. THD pour PWM à 3 niveaux du courant



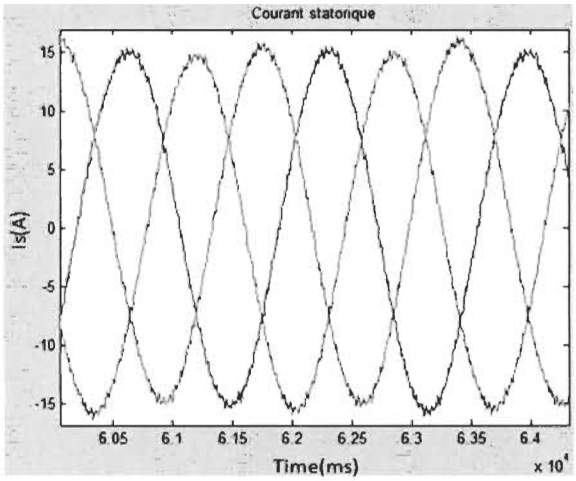
b. THD pour PWM à 5 niveaux du courant



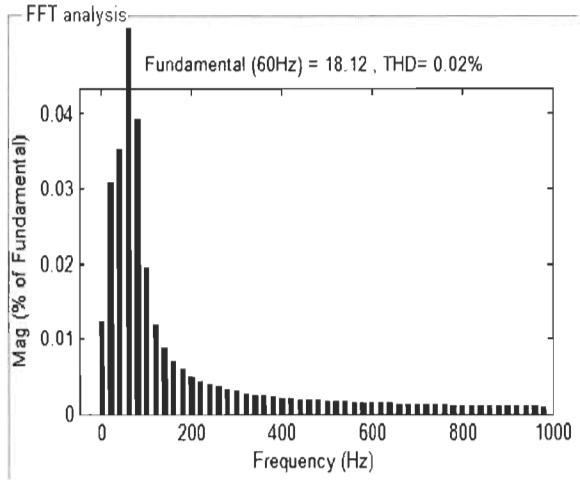
c. THD pour PWM à 7 niveaux du courant



d. THD pour PWM à 9 niveaux du courant



e. Courant moteur issu d'un onduleur idéal



f. THD d'un onduleur idéal

Fig. 5.18: Résultats de simulation THD en fonction des niveaux de tension

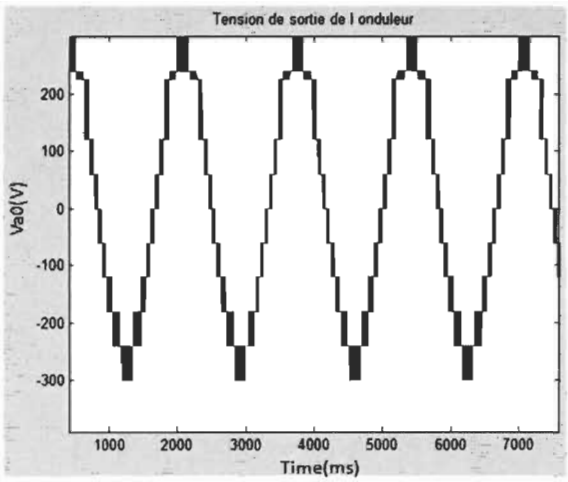
On constate bien que :

- les spectres contiennent très peu d'harmoniques HF. Pour celles qui sont de BF, il y'en existe presque pas. Elles sont très faibles. Il est facile par un simple filtre passe-bas, d'éliminer les harmoniques HF. Pour n = 5, THD = 5.20%,
- également, plus le nombre de niveaux augmente, plus les harmoniques sont de plus en plus faibles. On obtient ainsi :
- Pour n = 3, THD = 7.31%,
- Pour n = 5, THD = 5.20%,
- Pour n = 7, THD = 2.87%,
- Pour n = 9, THD = 0.85%,
- Pour un onduleur idéal, THD = 0.02%.

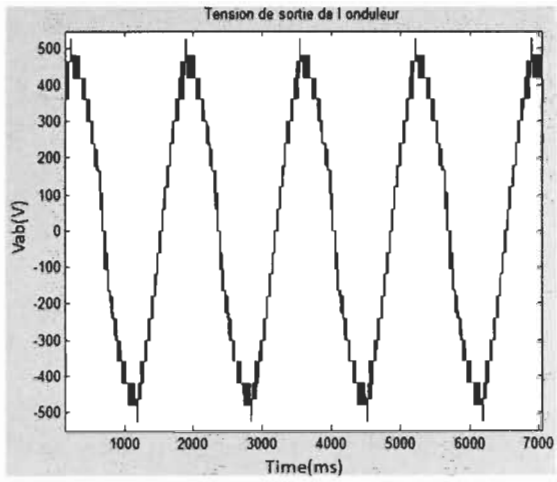
5.4.3 Limite des onduleurs multi-niveaux

Un certain nombre questions se posent toujours : les onduleurs multi-niveaux, continueraient-ils à maintenir leur performance notamment en ce qui concerne la réduction des harmoniques quelque soit leurs niveaux ? Aurait-il une limite à partir de laquelle ils ne seront plus efficaces ? Si oui, pourquoi ?

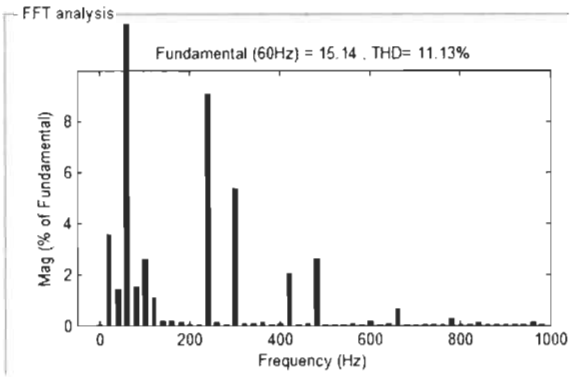
Pour répondre à ces questions nous avons poursuivi nos simulations pour voir jusqu'à où irons nous dans la réduction des harmoniques. C'est ainsi que nous avons simulé le cas de l'onduleur à 11 niveaux de tensions. Les résultats obtenus sont présentés dans les figures suivantes :



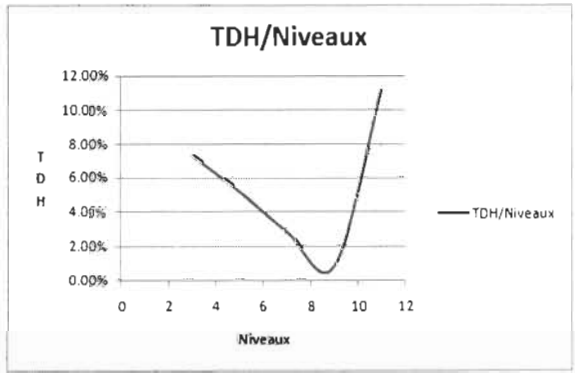
a. Tension de sortie V_{ao} PWM à 11 niveaux



b. Tension de sortie V_{ao} PWM à 11 niveaux



c. THD pour PWM à 11 niveaux du courant



d. THD pour PWM à 11 niveaux du courant

Fig. 5. 19: Tension de sortie V_ab PWM 11 niveaux

En observant les figures ci-dessous, on remarque que pour le cas des onduleurs à 11 niveaux, le taux de distorsion harmonique passe de 0.85% à 11.13% (fig.5.16.a). Ce qui est assez considérable. Cela est dû au nombre élevé d'interrupteurs sur un onduleur à 11 niveaux. Car pour réaliser ce type d'onduleur, le nombre d'interrupteurs nécessaires (d'après la formule vue plus haut : K=3(N-1) par branche), est de 90. Les pertes joules deviennent importantes et par conséquent le THD devient également plus élevé.

La figure 5.19.d, nous donne une vue globale de la performance des onduleurs multi-niveaux sur la réduction du THD. On remarque une baisse du THD du niveau 3 jusqu'à 9. A partir du niveau 11, on assiste à un changement brusque de l'allure de la courbe : THD augmente.

Ce qui nous permettra de conclure qu'à partir d'un certain niveau (11), les onduleurs multi-niveaux ne sont plus efficaces. Donc il y'a bien une limite à ne pas dépasser.

5.4.4 Commande d'un onduleur multi-niveaux SVPWM

5.4.4.1 Modélisation SVPWM trois niveaux

L'objectif ici, est de proposer une méthode de modélisation SVPWM multi-niveaux simplifiée sous Matlab/Sinulink avec moins de calculs qui faciliterait l'étape de l'implémentation en VHDL. Cette méthode a été expliquée plus haut. C'est ainsi que tout le calcul SVPWM trois niveaux a été développé dans un seul fichier .m matlab. Le reste des blocs concernent la modélisation et la commande de la machine asynchrone.

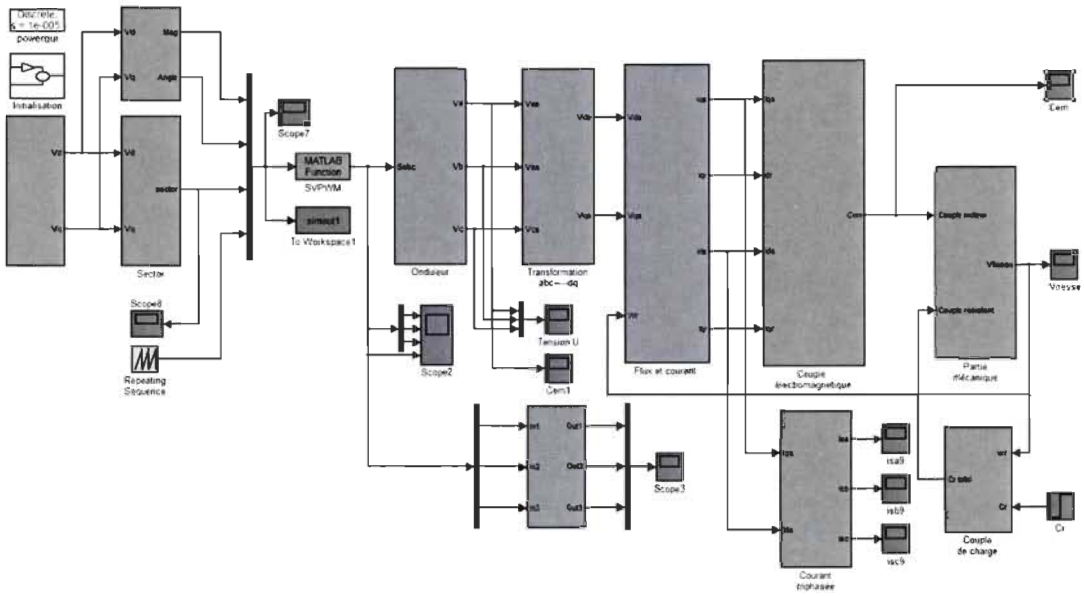


Fig. 5.20: Schéma de commande d'onduleur 3 niveaux avec SVPWM

5.4.4.2 Résultat de simulations

La figure 5.21 illustre le résultat de simulation de la commande SVPWM à trois niveaux. L'algorithme de commande est donné en annexe.

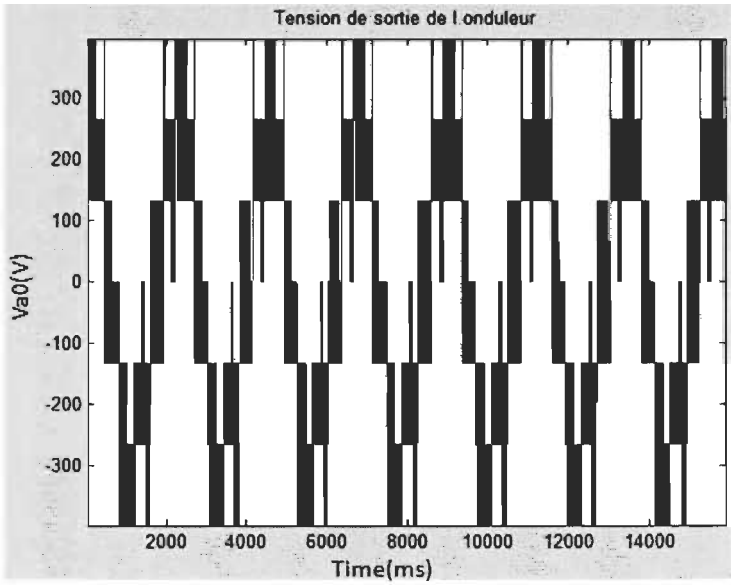


Fig. 5.21: Signaux de commande SVPWM 3 niveaux

5.4.5 SVPWM de niveaux supérieurs

Ces dix dernières années, les chercheurs s'intéressent de plus en plus à la commande SVPWM multi-niveaux. C'est ainsi que plusieurs articles ont été rédigés dans ce cadre. Cependant beaucoup de chemin reste encore à parcourir surtout quant il s'agit pour une implémentation en VHDL. À travers nos lectures et tests de simulation effectués, nous avons relevé un certain nombre de problèmes qui font encore objet d'études dont il est important de noter ici. Il s'agit de :

- La détermination exacte du numéro de la région du secteur à tous les instants t précis,
- La détermination de l'ordre des états de commutation,
- La détermination d'une formule générale permettant de calculer automatiquement les régions et générer les différents vecteurs espaces correspondants quelque soit le niveau de l'onduleur sans avoir à les écrire manuellement.

Parmi les articles basés sur la méthode simplifiée de deux niveaux, le [27] présente une façon simple et facilement implémentable en VHDL pour la détermination du secteur. Dans cette méthode l'auteur calcule le secteur en utilisant uniquement des opérations de comparaison à travers des valeurs intermédiaires. Cela permet d'éviter les opérations de multiplications et de divisions et par conséquent allègera l'algorithme en VHDL. Mais quant à la détermination des régions du secteur dans un système de commande multi-niveaux où il est question de plusieurs régions dans un même secteur, elle n'apporte pas de solution.

Quant à l'article [28], il propose une méthode pour la détermination des régions pour le calcul du SVPWM à trois et cinq niveaux, mais qui est beaucoup plus répétitive. Le diagramme vectoriel SVPWM trois niveaux est subdivisé en six hexagones dont chacun est considéré comme un diagramme vectoriel SVPWM à deux niveaux à part entière. Ce qui fait que plusieurs secteurs se retrouvent partager entre deux hexagones adjacents : d'où le problème de répétition de calcul par hexagone. Ainsi, cette méthode peut être utilisée dans un environnement Matlab/Simulink mais pas pour implémentation en VHDL.

L'article [29] présente une méthode de calcul différente de celles vues précédemment. Elle consiste à déterminer dans un premier temps les deux composantes du vecteur de référence puis décomposer chacune d'elles en deux sous-composants dont les coordonnées sont obtenues par une combinaison des valeurs arrondies supérieurs et inférieurs des coordonnées des deux premières composantes du vecteur de référence. Et ensuite, les temps de commutation sont déterminés à partir de ces quatre coordonnées. Cette méthode utilise beaucoup de fonctions d'arrondis (round, floor, ceil, fix) de Matlab. Bien que, ces dernières semblent simples en Matlab, elles restent néanmoins complexes pour une implémentation en VHDL. Aussi cette méthode est moins précise pour la détermination des temps de commutation.

La méthode présentée dans l'article [30] est celle qui semble être la plus adaptée pour une implémentation en VHDL, mais ne résout pas la question de la génération automatique des vecteurs espaces. Ce qui veut dire que tous les vecteurs doivent être saisis manuellement.
Pour illustrer l'importance d'avoir une méthode optimale qui générerait les vecteurs d'espace de façon automatique, nous allons lister ci-dessous les vecteurs espaces pour la commande d'onduleur à trois-niveaux juste :

En voyant le nombre de vecteurs nécessaires (voir tableau D.2.1 en annexe) juste pour une commande à trois niveaux, on comprend facilement qu'il est plus qu'important d'avoir une méthode qui les générerait automatiquement, surtout quand on sait qu'un seul changement de bits d'un 1 par un 0 ou la moindre erreur peut fausser les calculs.

5.5 MODELISATION DE LA COMMANDE D'ONDULEUR EN VHDL

La modélisation SVPWM pour la commande d'un onduleur multi-niveaux est considérablement plus complexe que celle de base de deux niveaux à cause du nombre important de commutateurs de l'onduleur et le problème de voltage au niveau du point neutre. Face à cette complexité de calcul, les algorithmes multi-niveaux sont souvent implémentés sur des modules plus performants et rapides comme les FPGA. Car les FPGA ont l'avantage d'exécuter les instructions de commande en parallèle au lieu de ligne par ligne comme c'est le cas des DSP. Cette caractéristique principale des FPGA permet de réduire les temps de délai et par conséquent augmenter la performance du système de contrôle.
Dans cette section nous présenterons la technique de modélisation SVPWM en VHDL pour la commande d'un onduleur à deux niveaux.

5.5.1 Modélisation SVPWM à deux niveaux en VHDL

5.5.1.1 Schéma RTL

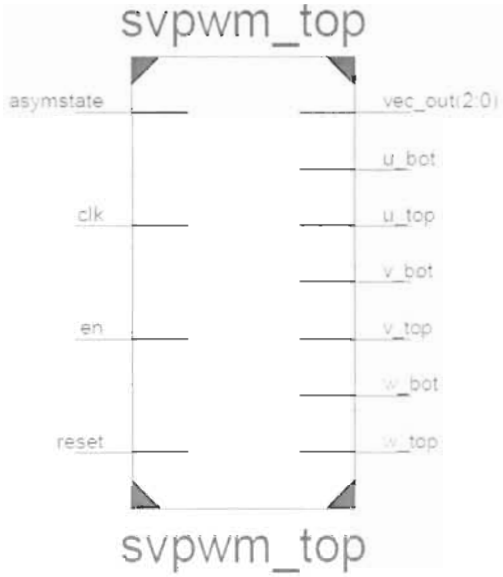


Fig. 5.22: Top schéma RTL

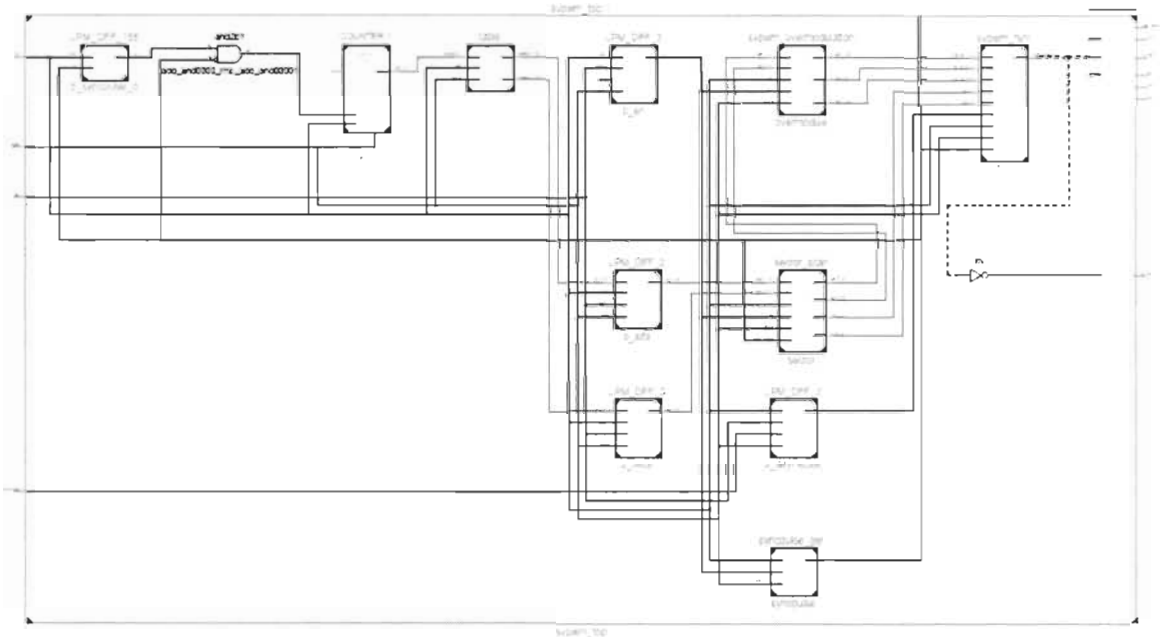


Fig. 5.23 : Schéma RTL détaillé

L'algorithme SVPWM deux niveaux a été modélisé avec le logiciel Xilin ISE et simulé avec celui de ModelSim. Après simulation, nous obtenons le schéma RTL ci-dessus (fig.5.23). Il contient sept blocs principaux :

- table : contient les tables des vecteurs espaces
- sector_scan : calcule le secteur dans lequel se trouve le vecteur de référence
- svpwm_overmodulation : calcule les temps de commutation
- svpwm_fsm : génère le vecteur espace
- syncpulse_gen : synchronise les impulsions
- deadzone : génère les signaux SVPWM

Les variables sont :

- Vx : composante x du vecteur de référence dans le secteur courant
- Vy : composante y du vecteur de référence dans le secteur courant
- s0, s1, s2, s3 : sont les états de commutation
- t0_in : temps des états "s0" et "s3"
- ta_in : temps de l'état "s1"
- tb_in : temps de l'état "s2"
- vec_out : vecteur de sortie

L'algorithme est donné dans la partie annexe (annexe D.3.2).

5.5.1.2 Ressources utilisées

L'algorithme a été développé sous 18 bits en utilisant le language de programmation VHDL qui est une language de description hardware standard de IEEE industry. Le FPGA Spartan3 a été utilisé pour la simulation grâce à son coût moins élevé. Le tableau suivant nous donne les détails :

Tableau 5. 5: FPGA utilisé

FPGA utilisé	
Compagnie	Xilinx
FPGA	FPGA Spartan®-3 Family
Détails	50K Gates 1728 Cells 630MHz 90nm Technology 1.2V 100-Pin VTQFP
Coût	$10 USD

Le tableau ci-dessous nous donne la liste des ressources utilisées dans le FPGA.

Tableau 5. 6: Ressources FPGA utilisées pour SVPWM2

Résumé de ressources utilisées (valeurs estimées)			
Utilisation Logique	**Utilisé**	**Disponible**	**Utilisation**
Nombre de Slices	562	768	73%
Nombre de Slice Flip Flops	470	1536	30%
Nombre de 4 input LUTs	852	1536	55%

5.5.1.3 Résultats de simulation

En faisant un test bench, on obtient les résultants illustrés dans la figure 5.24.

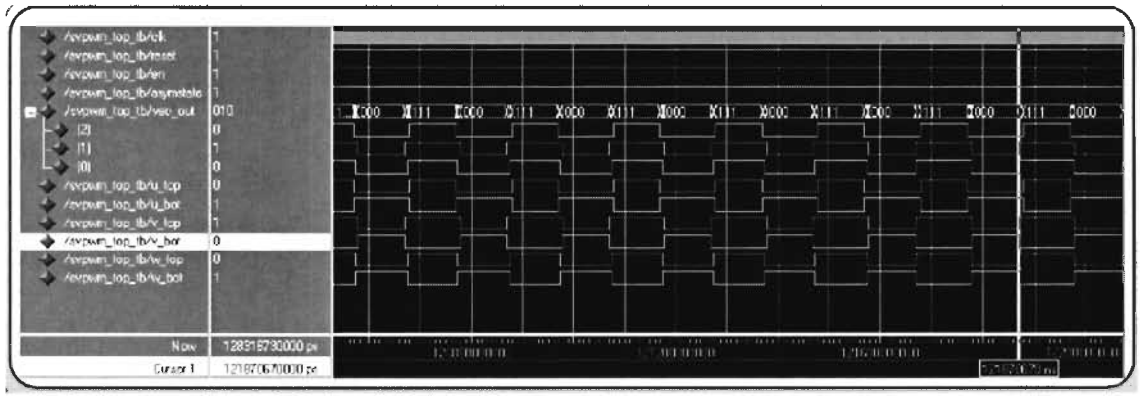


Fig. 5.24 : Signaux de commande SVPWM

Compte tenu du caractère symétrique de la technique SVPWM, on voit bien sur la figure 5.24, que pour un vecteur espace vect_out[0 1 0], on obtient les trois vecteurs de commande SVPWM u[0 1], v[1 0], w[0 1] pour les trois phases de l'onduleur. Les trois signaux de sortie PWM, ont chacun deux niveaux de signal 0 ou 1 (u[u_bot, u_top], v[v_bot, v_top], w[w_bot, w_top]). C'est le principe même de la commande PWM [33].

5.5.2 Estimation du type de FPGA nécessaire pour une implémentation de la SVPWM à sept niveaux

Dans ce qui suit, nous allons essayer de donner une estimation de ce qui serait nécessaire comme ressources pour une implémentation de la SVPWM à sept niveaux. Pour cela nous procédons de la façon suivante :

1. Etablir un tableau de comparaison de ressources pour les deux algorithmes de commande SVPWM à deux niveaux (SVPWM2) et la SVPWM à sept niveaux (SVPWM7) coté puissance

2. Etablir un tableau de comparaison de ressources des deux algorithmes de commande SVPWM à deux niveaux (SVPWM2) et la SVPWM à sept niveaux (SVPWM7) coté algorithmique.
3. Implémenter l'algorithme SVPWM2 en utilisant un FPGA différent disposant plus de ressources.

À partir des équations vues plus haut (5.3.1.2, 5.3.2.2), nous pouvons calculer les ressources matériels et algorithmiques de la SVPWM2 et SVPWM7. Elles sont présentées dans les deux tableaux ci-dessous :

Tableau 5. 7: Ressources coté puissance

Ressources coté puissance	**SVPWM2**	**SVPWM7**
Nombre de sources de tensions secondaires continues	1	6
Nombre d'interrupteurs	6	36
Nombre de diodes de bouclage	0	30

Tableau 5. 8: Ressources coté algorithmique

Ressources coté algorithmique	**SVPWM2**	**SVPWM7**
Nombre de vecteurs de tension	7	127
Nombre de secteurs	6	6
Nombre de régions par secteur	1	36
Nombre d'états de commutation	8	343

À partir de ces tableaux de comparaison ci-dessus et afin d'avoir une meilleure orientation sur le type de FPGA qu'il faut, nous avons implémenté le même algorithme SVPWM2 cette fois ci sur un FPGA Spartan-6. Et on remarque que la SVPWM2 utilise une proportion très faible des ressources disponibles comme le montre le tableau suivant :

Tableau 5. 9: Ressources FPGA utilisées

Résumé de ressources utilisées (valeurs estimées)			
Utilisation Logique	**Utilisé**	**Disponible**	**Utilisation**
Nombre de Registres de Slice	470	18224	2%
Nombre de Slice LUTs	709	9112	7%
Nombre de pairs complet LUT-FF utilisé	255	924	27%
Nombre de IOBs liés	13	232	5%
Nombre de BUFG/BUFGCTRLs	1	16	6%
Nombre de DSP48A1s	2	32	6%

Au regard de ce qui précède, nous estimons que le FPGA Spartan-6 (Tableau 5.9) serait suffisant pour une implémentation de la SVPWM7. Ce dernier dispose les ressources nécessaires pouvant répondre aux exigences d'implémentation de la SVPWM à 7 niveaux.

Tableau 5. 10: FPGA pour SVPWM7

FPGA utilisé pour SVPWM7	
Compagnie	Xilinx
FPGA	Spartan®-6 Family
Détails	14579 Cells 45nm (CMOS) Technology 1.2V
Coût	$30 USD

Il n'y a pas que le Spartan- 6 qui serait aproprié pour l'implémentation de la SVPWM7, toutes les familles Virtex le seraient aussi. Cependant ces dernières coutent plus chères (voir tableau 5.11). Ce qui fait que le Spartan-6 reste toujours le meilleur choix qualité/prix [34].

Tableau 5. 111: Prix moyens des FPGA pour SVPWM7

	Spartan-6	**Virtex-II Pro**	**Virtex-6**	**Virtex-5**	**Virtex-4**
Prix	$30 USD	$370 USD	$620 USD	$270 USD	$2,120 USD

5.6 CONCLUSION

Ce chapitre a été la partie la plus importante de notre mémoire. Dans ce dernier j'ai pu étudier la performance des onduleurs triphasés multi-niveaux à commande PWM et SVPWM en fonction des niveaux ainsi que le comportement de la machine asynchrone à vide et en charge alimenté par ces mêmes types d'onduleur. La simulation de la machine à vide ou en charge nous donne de bons résultats pour le couple et la vitesse. De même, la qualité du courant statorique en termes d'harmonique est meilleure, réduisant ainsi les effets néfastes sur la durée de vie de la machine et celle du réseau éventuel qui l'alimente. J'ai également mis en évidence dans ce chapitre, la limite de nombre de niveaux des onduleurs multi-niveaux pour une meilleure efficacité dans la réduction des harmoniques. Une implémentation de la SVPWM à deux niveaux a été réalisée, ainsi qu'une estimation du type de FGPA nécessaire pour implémentation de la SVPWM à 7 niveaux.

CHAPITRE 6

CONCLUSION GÉNÉRALE

Dans les systèmes de commande à grande puissance, l'utilisation des onduleurs classiques à deux niveaux a montré ses limites. Ils provoquent non seulement un niveau élevé de la dérivée dv/dt résultante de la commutation, mais aussi les interrupteurs ne supportent pas des fortes tensions inverses. Pour remédier à cela les onduleurs multi-niveaux ont été choisis comme le convertisseur de puissance préféré.

Dans ce travail, nous avons développé un modèle d'un système de commande des onduleurs multi-niveaux avec la technique PWM et SVPWM. Nos trois principales contributions portent sur :

- l'évaluation du taux de distorsion harmonique selon les niveaux trois, cinq, sept, neuf et onze de l'onduleur sous Matlab/Simulink;
- la détermination du niveau à partir duquel les onduleurs multi-niveaux ne sont plus efficaces.
- le développement d'un algorithme de commande simple SVPWM deux niveaux en VHDL.

Nous avons ainsi mis en évidence l'efficacité des onduleurs multi-niveaux dans la réduction des harmoniques du courant de la machine asynchrone. Nous avons pu démontrer qu'à partir du niveau onze, les onduleurs multi-niveaux ne jouent aucun rôle dans la réduction des harmoniques. Une méthode simplifiée de la SVPWM dans un système de commande DTC de la machine asynchrone a été aussi réalisée. Nous avons également développé un algorithme de la commande SVPWM implémentable sur un FPGA dont le coût est accessible à tous (10$ USA). Des propositions pour la généralisation de cet algorithme pour les onduleurs de niveaux supérieurs en VHDL ont été faites ainsi qu'une estimation des ressources matérielles pour une implémentation de la SVPWM à sept niveaux

Les résultats de simulations montrent bien une amélioration des signaux couple, vitesse et courant de la machine asynchrone alimentée par un onduleur multi-niveaux comparativement à celle alimenté par un onduleur classique.

Les points essentiels qui restent à faire sont :

1. continuer l'algorithme de commande SVPWM en VHDL que nous avons fait, pour les onduleurs de niveaux supérieur en VHDL;
2. utiliser le SVPWM multi-niveaux en VHDL dans un système de commande DTC de la machine asynchrone.

L'expérimentation sera essentielle pour valider pleinement les modèles et les résultats présentés dans ce mémoire.

Notons que tous nos modèles ont été développés, de façon à faciliter la généralisation de la méthode en VHDL. C'est ainsi que des codes Matlab simples ont été utilisés au lieu des blocs Simulink pour la plus part des modules principaux.

Cette étude nous a été très profitable, autant sur le plan technique que social. Elle nous a permis de mieux comprendre la commande des onduleurs multi-niveaux mais aussi la méthode de travail dans le domaine de la recherche.

Nous osons espérer que ce travail servira de référence de base permettant à un futur étudiant qui se pencherait sur les questions mentionnées ci-haut.

RÉFÉRENCES

[1] Wikipédia,"courants harmoniques", http://fr.wikipedia.org/wiki/Courants_harmoniques.

[2] L. Li, C. Dariusz, and Y. Liu, "Multilevel space vector PWM technique based on phase-shift harmonic suppression," Applied Power Electronics Conference and Exposition (APEC), Vol.1, 2000, pp535-541.

[3] L. M. Tolbert, "Multilevel Converters for Large Electric Drives", IEEE Trans. on Ind. Application, Vol. 35, pp. 36-44, January/February 1999.

[4] L. Yiqiao, and C.O. Nwankpa, "A new type of STATCOM based on cascading voltage source inverters with phase-shifted unipolar SPWM," IEEE Trans. on Industry Applications, Vol.35, No.5, 1999, pp1118-1123.

[5] D.G. Holmes, and P.M. Brendan, "Opportunities for harmonic cancellation with carrier based PWM for two level and multilevel cascaded inverters," IEEE Trans. on Industry Applications, Vol.37, No.2, 2001, pp574-582.

[6] T. Meynard, M. Nahrstaedt, R. Jakob, « Evolution des structures de conversion», RESELEC, 2004

[7] A. Nabae, « A Neutral – Point Clamped PWM Inverter », IEEE Transactions on Industry Applications, Vol. IA-17, N° 5, Septembre/Octobre 1981, pp 518-523.

[8] Takahashi, I., Noguchi, T., "A New Quick response and High-Efficiency Control Strategy of an induction Motor", IEEE Annual Meeting on Industry application Society, Toronto, Canada, 6-1 Octobre 1985, pp. 495-502.

[9] Depenbrock, M., Baader, U., "Direct Self Control (DSL) of Inverters-Fed Induction Machine: A basis for Speed Control without Speed Measurement", IEEE Trans. on Industry Appl., Vol. 28, May/June 1992, pp 581-588.

[10] Marcelo F. Castoldi, Manoel L. Aguiar, "Simulation of DTC Strategy in VHDL Code for Induction Motor Control", IEEE ISIE 2006, July 9-12, 2006, Montréal, Québec, Canada.

[11] M. L. Doumbia et Abdoulaye Traore, "Modélisation et estimation d'une machine asynchronen à l'aide du logiciel Matlab"

[12] Merlin Ge, "Les techniques de commande du moteur asynchrone", Intersections, Juin 1998.

[13] N. Celanovic, "Space Vector Modulation and Control of Multilevel Converters", PhD Thesis, Virginia Polytechnic Institute, 2000.

[14] Y. Shakweh, E. A. Lewis, Assessment of medium voltage PWM VSI topologies for multi-megawatt variable speed drive applications", IEEE-PESC Conference Record, vol.2, pp. 965-971, 1999.

[15] N. Celanovic, D. Boroyevich, "A Comprehensive Study of Neutral-Point Voltage Balancing Problem in Three Level Neutral-Point-Clamped Voltage Source PWM Inverters", IEEE-APEC Conference Record, 1999

[16] M. D. Manjrekar, P. Steimer, T. A. Lipo, "Hybrid Multilevel Power Conversion System: A Competitive Solution for High Power Applications", IEEE-IAS Conference, 1999.

[17] H. FOCH, F. FOREST, T. MEYNARD, "Onduleurs de tension", Techniques de l'Ingénieur, novembre 1998.

[18] B. K. Bose, 'Power Electronics And AC Drives', Edition Practice Hall, 1986.

[19] Andrzej M Trzynadlowski, 'Introduction To ModernPower Electronics', A Wiley-Interscience Publication W John & Sons, Inc USA

[20] Y. Amara "Contribution à la Conception et à la Commande des Machine Synchrone à Double Excitation Application au Véhicule Hybride", Thèse de Doctorat de l'Université de Paris XI, France, Décembre 2001.

[21] J.P. Caron, et J.P Hautier, "Modélisation et Commande de la Machine Asynchrone", Edition Technip, Paris 1995

[22] F. Labrique, G. Seguier et R. Bausier, "Les convertisseurs de l'électronique de puissance, Volume 4: La conversion continu-Alternatif", Lavoisier, 1995

[23] N. Akira, T. Isao, A. Hirofumi, "A New Neutral - Point- Clamped PWM Inverter", IEEE Tran. On Ind. Appl. Vol. IA-17, No. 5, Sep/Oct 1981. pp. 518-523

[24] Jin-Woo Jung, "Space Vector PWM Inverter" The Ohio State University, Feb. 2005

[25] R. Teodorescu , F. Beaabjerg , J. K. Pedersen , E. Cengelci , S. U. Sulistijo , B. O. Woo and P. Enjeti "Multilevel converters — A survey", Proc. EPE'99, pp. 1999

[26] Joseph Song Manguelle, "Convertisseurs multiniveaux asymétriques alimentés par transformateurs multi-secondaires basse-fréquence: réactions au réseau d'alimentation", Thèse, École Polytechnique Fédérale de Lausanne, EPFL 2004

[27] An efficient SVPWM algorithm with low computational overhead for three-phase inverters Z Shu, J Tang, Y Guo, J Lian - IEEE Transactions on Power Electronics, 2007.

[28] D. Lalili, N. Lourci, E. M. Berkouk, F. Boudjema, J. Petzold "Méthode simplifiée de la modulation vectorielle de l'onduleur à cinq niveaux", Université de Jijel, Algérie 2005.

[29] Nikola Celanovic, Dushan Boroyevich, "A fast space-vector modulation algorithm for multilevel three-phase converters", IEEE Trans. Ind. Applicat. vol. 37, pp 637 – 641, March/April 2001

[30] Amit Kumar Gupta, Ashwin M. Khambadkone, "A Space Vector PWM Scheme for Multilevel Inverters Based on Two-Level Space Vector PWM," IEEE Trans. Ind. Electron., vol.53, no.5, pp.1631-1639, Oct. 2006

[31] Zeliang Shu, Jian Tang, Yuhua Guo, Jisan Lian, "An Efficient SVPWM Algorithm With Low Computational Overhead for Three-Phase Inverters". Power Electronics, volume: 22 Issue: 5, IEEE Transactions on Sept. 2007

[32] Ayse Kocalms, and Sedat Sunter, " Simulation of a space vector PWM controller for three-level voltage -fed inverter motor drive," Firat University Turkey, IEEE 2006

[33] Hejiong, SYTU, Jiangnan University Library and Archives

[34] Xilinx inc.: http://www.xilinx.com; http://www.xilinx.com/onlinestore/index.htm

ANNEXES

ANNEXE A : MODELE SIMULINK DU MOTEUR ASYNCHRONE A CAGE

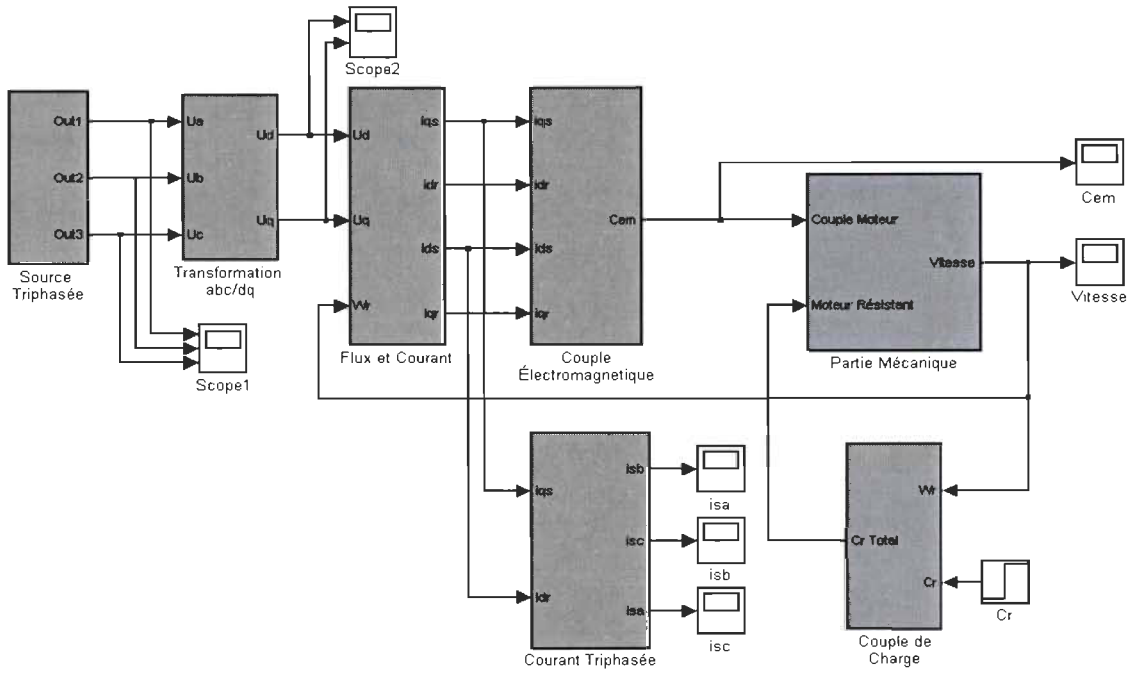


Fig. A.1: Modèle Simulink du moteur asynchrone

ANNEXE B : MODELE SIMULINK DU MOTEUR ASYNCHRONE A CAGE

B.2 Commande SVPWM

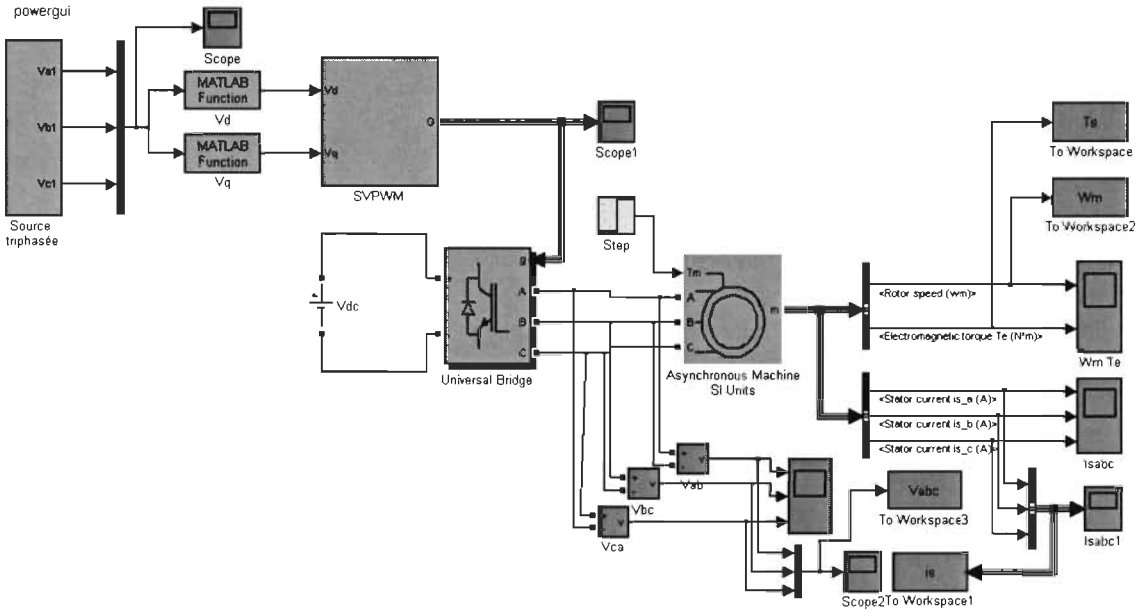


Fig. B.1: Commande SVPWM d'une machine asynchrone

B.2 Fonction SVPWM deux niveaux

```
% Fonction SVPWM 2 niveaux
% Les entres sont: magnitude mag,angle x
% et le signal triangulaire pour la comparison y

function sf    = SVPWM2(u)
ts=0.0002;
mag=(u(1)/peak_phase_max)*ts; x=u(2); y=u(3);
sa=0; sb=0; sc=0;

% Secteur 1
sa=0; sb=0; sc=0;
if(x>=0)&&(x<pi/3)
   ta = mag*sin(pi/3-x);tb= mag*sin(x);
   t0 =(ts-ta-tb);
   t1=[t0/4 ta/2 tb/2 t0/2 tb/2 ta/2 t0/4];t1=cumsum(t1);
   v1=[0 1 1 1 1 1 0];v2=[0 0 1 1 1 0 0];v3=[0 0 0 1 0 0 0];
    for j=1:7
       if(y<t1(j))
       break
       end
    end
   sa=v1(j);sb=v2(j);sc=v3(j);
end
% Secteur II
if (x>=pi/3)&&(x<2*pi/3)
    adv=x-pi/3;
    tb = mag*sin(pi/3-adv);ta= mag*sin(adv);
    t0 =(ts-ta-tb);
    t1=[t0/4 ta/2 tb/2 t0/2 tb/2 ta/2 t0/4];t1=cumsum(t1);
    v1=[0 0 1 1 1 0 0];v2=[0 1 1 1 1 1 0];v3=[0 0 0 1 0 0 0];
    for j=1:7
        if(y<t1(j))
            break
        end
    end
    sa=v1(j);sb=v2(j);sc=v3(j);
end

%Secteur III
   if(x>=2*pi/3)&&(x<pi)
      adv=x-2*pi/3;
      ta=mag*sin(pi/3-adv); tb=mag*sin(adv);
      t0 =(ts-ta-tb);
   t1=[t0/4 ta/2 tb/2 t0/2 tb/2 ta/2 t0/4];t1=cumsum(t1);
   v1=[0 0 0 1 0 0 0];v2=[0 1 1 1 1 1 0];v3=[0 0 1 1 1 0 0];
     for j=1 :7
         if(y<t1(j))
             break
         end
     end
     sa=v1(j);sb=v2(j);sc=v3(j);
   end
```

```
%Secteur IV
 if(x>=-pi)&&(x<-2*pi/3)
 adv = x + pi;
 tb=mag*sin(pi/3 - adv);ta=mag*sin(adv);
 t0 =(ts-ta-tb);
 t1=[t0/4 ta/2 tb/2 t0/2 tb/2 ta/2 t0/4];t1=cumsum(t1);
 v1=[0 0 0 1 0 0 0];v2=[0 0 1 1 1 0 0];v3=[0 1 1 1 1 1 0];
  for j=1:7
      if(y<t1(j))
          break
      end
  end
  sa=v1(j);sb=v2(j);sc=v3(j);
 end
%Secteur V
 if(x>=-2*pi/3)&&(x<-pi/3)
  adv = x + 2*pi/3;
 ta= mag*sin(pi/3 - adv);tb= mag*sin(adv);
 t0 =(ts-ta-tb);
 t1=[t0/4 ta/2 tb/2 t0/2 tb/2 ta/2 t0/4];t1=cumsum(t1);
 v1=[0 0 1 1 1 0 0];v2=[0 0 0 1 0 0 0];v3=[0 1 1 1 1 1 0];
  for j=1:7
      if(y<t1(j))
          break
      end
  end
 sa=v1(j);sb=v2(j);sc=v3(j);
 end
 %Secteur VI
 if(x>= -pi/3)&&(x<0)
  adv = x + pi/3;
 tb= mag*sin(pi/3 - adv);ta= mag*sin(adv);
 t0 =(ts-ta-tb);
 t1=[t0/4 ta/2 tb/2 t0/2 tb/2 ta/2 t0/4];t1=cumsum(t1);
 v1=[0 1 1 1 1 1 0];v2=[0 0 0 1 0 0 0];v3=[0 0 1 1 1 0 0];
   for j=1:7
      if(y<t1(j))
          break
      end
   end
  sa=v1(j);sb=v2(j);sc=v3(j);
 end
 sf = [sa, sb, sc];
```

ANNEXE C : COMMANDE DTC

C.1 Système de commande DTC

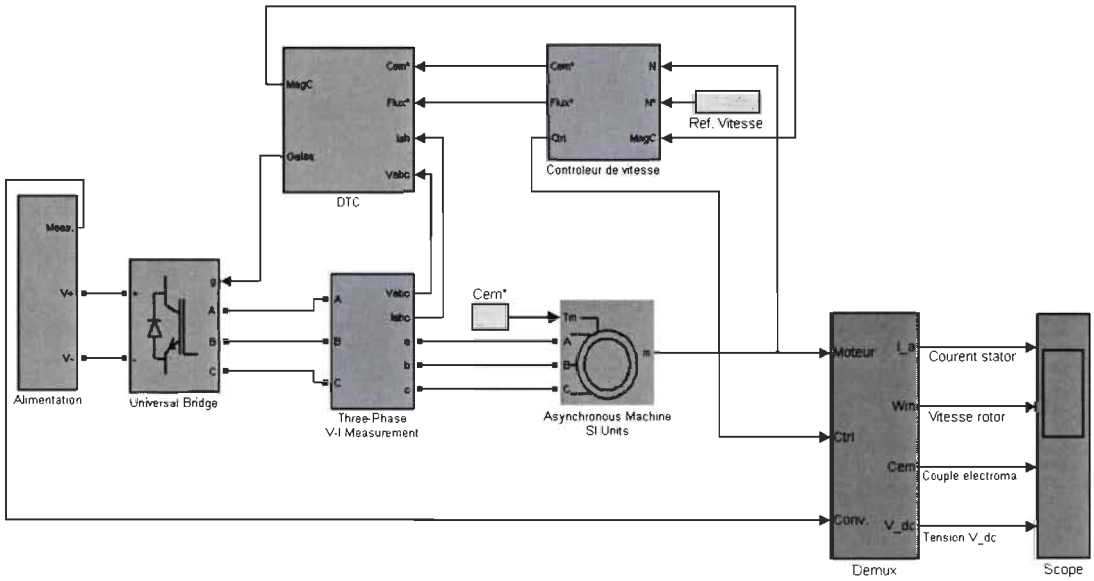


Fig 4.8 : Système de commande DTC

C.2 Fonctions DTC

C.2.1 Fonction calcul du secteur

```
function [sys,x0,str,ts]=zon(t,x,u,flag)
switch flag;

%initialisation
case 0
sizes =simsizes;
sizes.NumContStates=0;
sizes.NumDiscStates=0;
sizes.NumOutputs=1;
sizes.NumInputs=1;
sizes.DirFeedthrough=1;
sizes.NumSampleTimes=1;
sys=simsizes(sizes);
x0=[];
str=[];
ts=[-1 0];

%intervalles des secteurs
case 3
A=[-30 30];
B=[30 90];
C=[90 150];
D=[150 210];
E=[210 270];
F=[270 330];

%detection du secteur
 alpha=u;
 alpha=mod(alpha,360);
if alpha>=A(1)& alpha<A(2)
    sys=1;
elseif alpha>=B(1)& alpha<B(2)
    sys=2;
elseif alpha>=C(1)& alpha<C(2)
    sys=3;
elseif alpha>=D(1)& alpha<D(2)
    sys=4;
elseif alpha>=E(1)& alpha<E(2)
    sys=5;
elseif alpha>=F(1)& alpha<F(2)
    sys=6;
end;

case {1,2,8,9}
    sys=[];
otherwise
error(['Erreur flag=',num2str(flag)]);
end;%
```

C.2.2 Fonction du régulateur à hystérésis du flux

```
%fonction régulateur à hystérésis du flux
function [cflx,x0,str,ts]=zon(t,x,u,flag)
switch flag,

%initialisation
case 0
sizes =simsizes;
sizes.NumContStates=0;
sizes.NumDiscStates=0;
sizes.NumOutputs=1;
sizes.NumInputs=2;
sizes.DirFeedthrough=1;
sizes.NumSampleTimes=1;
cflx=simsizes (sizes);
x0=[];
str=[];
ts=[-1 0];

case 3
%variables d'entrée
flxr=u(1);
flx=u(2);
F_bw = 0.02;
%%régulation à hystérésis du flux
e_flx = flxr - flx;
if e_flx >= F_bw/2
    cflx=1;
elseif e_flx <= -F_bw/2
    cflx=2;
end;

case {1,2,4,9}
sys=[];
otherwise
error(['Erreur flag = ' ,num2str(flag)]);
end;
```

C.2.3 Fonction du régulateur à hystérésis du couple

```
%fonction régulateur à hystéresis du couple
function [ccpl,x0,str,ts]=zon(t,x,u,flag)
switch flag,

%initialisation
case 0
sizes =simsizes;
sizes.NumContStates=0;
sizes.NumDiscStates=0;
sizes.NumOutputs=1;
sizes.NumInputs=2;
sizes.DirFeedthrough=1;
sizes.NumSampleTimes=1;
ccpl=simsizes (sizes);
x0=[];
str=[];
ts=[-1 0];

case 3
%variables d'entrée
cplr=u(1);
cpl=u(2);
c_bw = 10;
%régulation à hystérésis du couple
e_cpl = cplr - cpl;
if e_cpl >= c_bw/2
    ccpl1=1;
    ccpl2=0;
elseif e_cpl >= 0 & e_cpl < c_bw/2
    ccpl1=0;
    ccpl2=0;
elseif e_cpl < 0 & e_cpl > (-c_bw)/2
    ccpl1=0;
    ccpl2=0;
elseif e_cpl <= (-c_bw)/2
    ccpl1=0;
    ccpl2=3;
end;
ccpl3 = 2*(or(ccpl1, ccpl2));
ccpl4 = not(ccpl3);
ccpl = ccpl1 + ccpl2 + ccpl4;

case {1,2,4,9}
sys=[];
otherwise
error(['Erreur flag = ' ,num2str(flag)]);
end;
```

C.2.4 Fonction du calculateur SVPWM

```
%fonction SVPW
function [sys,x0,str,ts]=zon(t,x,u,flag)
switch flag,

%initialisation
case 0
sizes =simsizes;
sizes.NumContStates=0;
sizes.NumDiscStates=0;
sizes.NumOutputs=6;
sizes.NumInputs=3;
sizes.DirFeedthrough=1;
sizes.NumSampleTimes=1;
sys=simsizes (sizes);
x0=[];
str=[];
ts=[-1 0];

case 3
%vecteurs espaces
V0=[0 1 0 1 0 1];
V1=[1 0 0 1 0 1];
V2=[1 0 1 0 0 1];
V3=[0 1 1 0 0 1];
V4=[0 1 1 0 1 0];
V5=[0 1 0 1 1 0];
V6=[1 0 0 1 1 0];
V7=[1 0 1 0 1 0];

%variables d'entrée
ccpl=u(1);
cflx=u(2);
N=u(3);

%technique SVPWM
if cflx==1
    if ccpl==1
        if N==1
            sys=V2;
        elseif N==2
            sys=V3;
        elseif N==3
            sys=V4;
        elseif N==4
            sys=V5;
        elseif N==5
            sys=V6;
        elseif N==6
            sys=V1;
        end;
    end;
end;
```

```
if cflx==1
    if ccpl==2
        if N==1
            sys=V0;
        elseif N==2
            sys=V7;
        elseif N==3
            sys=V0;
        elseif N==4
            sys=V7;
        elseif N==5
            sys=V0;
        elseif N==6
            sys=V7;
        end;
    end;
end;

if cflx==1
    if ccpl==3
        if N==1
            sys=V6;
        elseif N==2
            sys=V1;
        elseif N==3
            sys=V2;
        elseif N==4
            sys=V3;
        elseif N==5
            sys=V4;
        elseif N==6
            sys=V5;
        end;
    end;
end;

if cflx==2
    if ccpl==1
        if N==1
            sys=V3;
        elseif N==2
            sys=V4;
        elseif N==3
            sys=V5;
        elseif N==4
            sys=V6;
        elseif N==5
            sys=V1;
        elseif N==6
            sys=V2;
        end;
    end;
end;
```

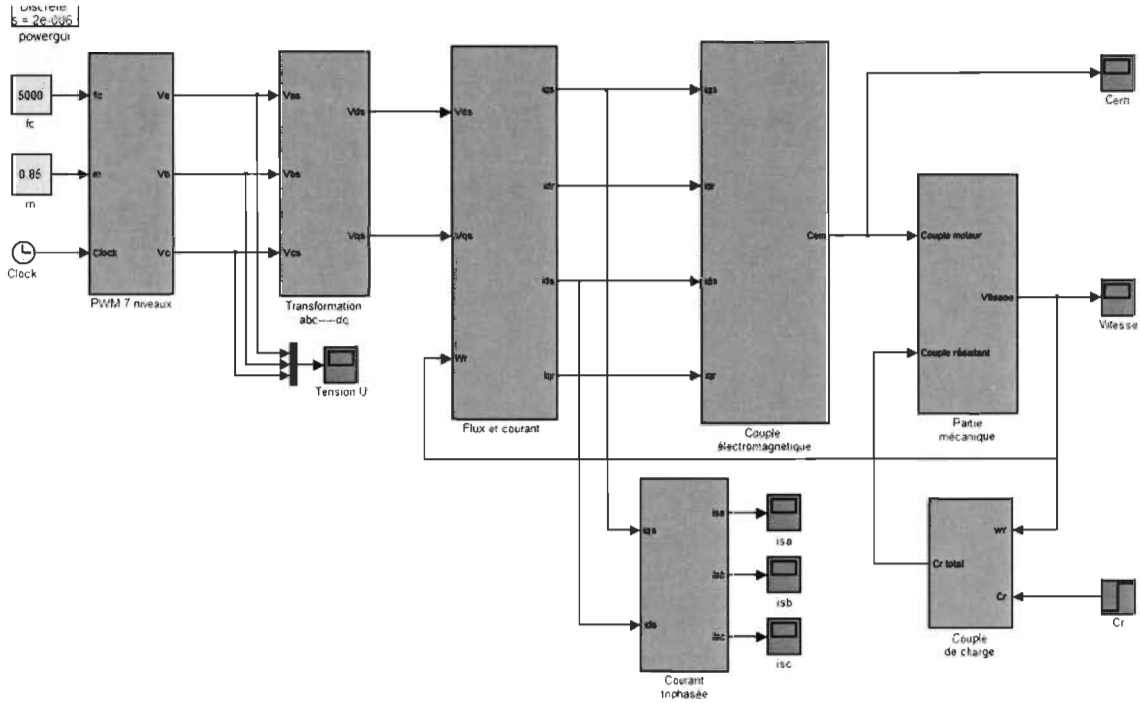
```
if cflx==2
    if ccpl==2
        if N==1
            sys=V7;
        elseif N==2
            sys=V0;
        elseif N==3
            sys=V7;
        elseif N==4
            sys=V0;
        elseif N==5
            sys=V7;
        elseif N==6
            sys=V0;
        end;
    end;
end;

if cflx==2
    if ccpl==3
        if N==1
            sys=V5;
        elseif N==2
            sys=V6;
        elseif N==3
            sys=V1;
        elseif N==4
            sys=V2;
        elseif N==5
            sys=V3;
        elseif N==6
            sys=V4;
        end;
    end;
end;

case {1,2,4,9}
sys=[];
otherwise
error(['Erreur flag = ' ,num2str(flag)]);
end;
```

ANNEXE D : COMMANDE PWM ET SVPWM MULTI-NIVEAUX

D.1 Commande PWM à 7 niveaux



D.1 Modélisation et commande d'onduleur à 7 niveaux

D.2 Commande PWM à 7 niveaux

```
%  Fonction SVPWM multi-niveaux
%  fr -> Fréquence du signal tension de référence (fr>0).
%  fc -> Fréquence de la porteuse(fc>f),fc=n*fr, ou n est un nombre entier.
%  m  -> Indice de Modulation (0<m<1),  t  -> Temps
function VUout=pwm1(fr,fc,m,t)
Ar=3; Ac=1; Vdc=1;
fc = floor(fc/fr)*fr;
if fc<fr, fc=fr; end
N=length(t);
if fc<fr, fc=fr; end
N=length(t);
Ta=zeros(N,12);
for i=1:N;
   carrier11=Ac*triangwf(fc,t(i));
   carrier12=carrier11+1;
   carrier13=carrier11+2;
   carrier21=carrier11-1;
   carrier22=carrier11-2;
   carrier23=carrier11-3;
   C11(i)=carrier11;
   C12(i)=carrier12;
   C13(i)=carrier13;
   C21(i)=carrier21;
   C22(i)=carrier22;
   C23(i)=carrier23;
   reference=Ar*m*sin(2*pi*fr*t(i));
   R(i)=reference;
   if reference >= carrier11,
      Tu11(i)=1;
   else
      Tu11(i)=0;
   end;
   if reference <= carrier21,
      Tu21(i)=1;
   else
      Tu21(i)=0;
   end;
   if reference >= carrier12,
      Tu12(i)=1;
   else
      Tu12(i)=0;
   end;
   if reference <= carrier22,
      Tu22(i)=1;
   else
      Tu22(i)=0;
   end;
   if reference >= carrier13,
      Tu13(i)=1;
   else
      Tu13(i)=0;
   end;
   if reference <= carrier23,
      Tu23(i)=1;
   else
      Tu23(i)=0;
   end;
```

```
%==============Suite PWM 7 niveaux==========================
T1(i)=Tu11(i)-Tu21(i);
T2(i)=Tu12(i)-Tu22(i);
T3(i)=Tu13(i)-Tu23(i);
T(i)=T1(i)+T2(i)+T3(i);
if T(i)==3
    Ta(i,:)=[1 1 1 1 1 1 0 0 0 0 0 0];
    V=Vdc/2;          %Vdc/2;
end;
if T(i)==2
    Ta(i,:)=[0 1 1 1 1 1 1 0 0 0 0 0];
    V=Vdc/3;          %Vdc/3;
end;
if T(i)==1
    Ta(i,:)=[0 0 1 1 1 1 1 1 0 0 0 0];
    V=Vdc/6;          %Vdc/6;
end;
if T(i)==0
    Ta(i,:)=[0 0 0 1 1 1 1 1 1 0 0 0];
    V=0;
end;
if T(i)==-1
    Ta(i,:)=[0 0 0 0 1 1 1 1 1 1 0 0];
    V=-Vdc/6;         %-Vdc/6;
end;
if T(i)==-2
    Ta(i,:)=[0 0 0 0 0 1 1 1 1 1 1 0];
    V=-Vdc/3;         %-Vdc/3;
end;
if T(i)==-3
    Ta(i,:)=[0 0 0 0 0 0 1 1 1 1 1 1];
    V=-Vdc/2;         %-Vdc/2;
end;
VUout = [Ta(i,1), Ta(i,2), Ta(i,3), Ta(i,4), Ta(i,5), Ta(i,6), Ta(i,7),
Ta(i,8), Ta(i,9), Ta(i,10), Ta(i,11), Ta(i,12), T(i), R(i), V, C11, C12,
C13, C21, C22, C23];
end;
end
```

D.2 Commande SVPWM à 3 niveaux

D.2.1 Tableau vecteurs espaces SVPWM trois niveaux

Tableau vecteurs espaces SVPWM trois niveaux			
Régions	**Phase A**	**Phase B**	**Phase C**
Secteur A			
Région1	[-1 0 0 0 1 1 1 1 1 1 0 0 0 -1]	[-1 -1 0 0 0 1 1 1 1 0 0 0 -1 -1]	[-1 -1 -1 0 0 0 1 1 0 0 0 -1 -1 -1]
Région2	[0 1 1 1 1 1 1 0]	[-1 -1 0 0 0 0 -1 -1]	[-1 -1 -1 0 0 -1 -1 -1]
Région3	[0 0 1 1 1 1 1 1 1 0 0]	[-1 0 0 0 1 1 0 0 0 -1]	[-1 -1 -1 0 0 0 0 -1 -1 -1]
Région4	[0 1 1 1 1 1 1 0];	[0 0 1 1 1 1 1 0 0];	[-1 -1 -1 0 0 -1 -1 -1]
Secteur B			
Région1	[-1 -1 0 0 0 1 1 1 1 0 0 0 -1 -1]	[-1 0 0 0 1 1 1 1 1 1 0 0 0 -1]	[-1 -1 -1 0 0 0 1 1 0 0 0 -1 -1 -1]
Région2	[0 0 1 1 1 1 1 0 0]	[0 1 1 1 1 1 1 1 0]	[-1 -1 -1 0 0 -1 -1 -1]
Région3	[-1 0 0 0 1 1 0 0 0 -1]	[0 0 1 1 1 1 1 1 1 0 0]	[-1 -1 -1 0 0 0 0 -1 -1 -1]
Région4	[-1 -1 0 0 0 0 -1 -1]	[0 1 1 1 1 1 1 1 0]	[-1 -1 -1 0 0 -1 -1 -1]
Secteur C			
Région1	[-1 -1 -1 0 0 0 1 1 0 0 0 -1 -1 -1]	[-1 0 0 0 1 1 1 1 1 1 0 0 0 -1]	[-1 -1 0 0 0 1 1 1 1 0 0 0 -1 -1]
Région2	[-1 -1 -1 0 0 -1 -1 -1]	[0 1 1 1 1 1 1 1 0]	[-1 -1 0 0 0 0 -1 -1]
Région3	[-1 -1 -1 0 0 0 0 -1 -1 -1]	[0 0 1 1 1 1 1 1 1 0 0]	[-1 0 0 0 1 1 0 0 0 -1]
Région4	[-1 -1 -1 0 0 -1 -1 -1]	[0 1 1 1 1 1 1 1 0]	[0 0 1 1 1 1 1 0 0]
Secteur D			
Région1	[-1 -1 -1 0 0 0 1 1 0 0 0 -1 -1 -1]	[-1 -1 0 0 0 1 1 1 1 0 0 0 -1 -1]	[-1 0 0 0 1 1 1 1 1 1 0 0 0 -1]
Région2	[-1 -1 -1 0 0 -1 -1 -1]	[0 0 1 1 1 1 1 0 0]	[0 1 1 1 1 1 1 1 0]
Région3	[-1 -1 -1 0 0 0 0 -1 -1 -1]	[-1 0 0 0 1 1 0 0 0 -1]	[0 0 1 1 1 1 1 1 1 0 0]

Région4	[-1 -1 -1 0 0 -1 -1 -1]	[-1 -1 0 0 0 0 -1 -1]	[0 1 1 1 1 1 1 0]
Secteur E			
Région1	[-1 -1 0 0 0 1 1 1 1 0 0 0 -1 -1]	[-1 -1 -1 0 0 0 1 1 0 0 0 -1 -1 -1]	[-1 0 0 0 1 1 1 1 1 1 0 0 0 -1]
Région2	[-1 -1 0 0 0 0 -1 -1]	[-1 -1 -1 0 0 -1 -1 -1]	[0 1 1 1 1 1 1 0]
Région3	[-1 0 0 0 1 1 0 0 0 -1]	[-1 -1 -1 0 0 0 0 -1 -1 -1]	[0 0 1 1 1 1 1 1 1 0 0]
Région4	[0 0 1 1 1 1 1 0 0]	[-1 -1 -1 0 0 -1 -1 -1]	[0 1 1 1 1 1 1 0]
Secteur F			
Région1	[-1 0 0 0 1 1 1 1 1 1 0 0 0 -1]	[-1 -1 -1 0 0 0 1 1 0 0 0 -1 -1 -1]	[-1 -1 0 0 0 1 1 1 1 0 0 0 -1 -1]
Région2	[0 1 1 1 1 1 1 0]	[-1 -1 -1 0 0 -1 -1 -1]	[0 0 1 1 1 1 0 0]
Région3	[0 0 1 1 1 1 1 1 1 0 0]	[-1 -1 -1 0 0 0 0 -1 -1 -1]	[-1 0 0 0 1 1 0 0 0 -1]
Région4	[0 1 1 1 1 1 1 0]	[-1 -1 -1 0 0 -1 -1 -1]	[-1 -1 0 0 0 0 -1 -1]

```
%Matlab Code SVPWM 3 niveaux
% Les entres sont: magnitude mag,angle x
% secteur s et le signal triangulaire pour la comparison y

function sf    = aaa(mag,x,s,y)
ts=0.0002;
h=sqrt(3)/2;
sa=0; sb=0; sc=0;
x2=rem(x,(pi/3));
V_refd=mag*cos(x2);
V_refq=mag*sin(x2);

% Calcul de paramettres de region

m1= V_refd + V_refq/sqrt(3);
m2= V_refq/h;
m11=(mag+mag/sqrt(3))/2;
m22=(mag/h)/2;
if ((m1>=m11) || (m1<=-m11))
    m1=1;
else m1=0;
end;
if ((m2>=m22) || (m2<=-m22))
    m2=1;
else m2=0;
end;

V_refdi = V_refd - m1 + 0.5*m2;
V_refqi = V_refq - m2*h;
if (V_refqi <= sqrt(3)*V_refdi);
    V_refd2 = V_refdi;
    V_refq2 = V_refqi;
    delta = fix(m1^2 + 2*m2 + 1);
else
    V_refd2 = 0.5 - V_refdi;
    V_refq2 = h - V_refqi;
    delta = fix(m1^2 + 2*m2 + 2);
end;
ta = ts*(V_refd2 - V_refq2/(2*h));
tb = ts*(V_refq2/h);
t0 = ts - (ta + tb);
```

```
%=================== Sector A ==================================
% secteur A
if (s==1)
   if (delta == 1) % region 1
       t1=[tb/4 ta/2 t0/2 tb/2 ta/2 t0/2 tb/4 tb/4 t0/2 ta/2 tb/2 t0/2 ta/2
tb/4];
       t1=cumsum(t1);
       va=[-1 0 0 0 1 1 1 1 1 1 0 0 0 -1];
       vb=[-1 -1 0 0 0 1 1 1 1 0 0 0 -1 -1];
       vc=[-1 -1 -1 0 0 0 1 1 0 0 0 -1 -1 -1];
       for j=1:14
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
   if (delta == 2)                          % region 2
       t1=[ta 2*t0 tb ta/2 ta/2 tb t0 ta/2];
       t1=cumsum(t1);
       va=[0 1 1 1 1 1 1 0];
       vb=[-1 -1 0 0 0 0 -1 -1];
       vc=[-1 -1 -1 0 0 -1 -1 -1];
       for j=1:8
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
   if (delta == 3)                          % region 3
       t1=[ta/2 t0/2 tb t0/2 ta/2 ta/2 t0/2 tb t0/2 ta/2];
       t1=cumsum(t1);
       va=[0 0 1 1 1 1 1 1 0 0];
       vb=[-1 0 0 0 1 1 0 0 0 -1];
       vc=[-1 -1 -1 0 0 0 0 -1 -1 -1];
       for j=1:10
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
   if (delta == 4)% region 4
       t1=[ta/2 tb t0 ta/2 ta/2 t0 tb ta/2];
       t1=cumsum(t1);
       va=[0 1 1 1 1 1 1 0];
       vb=[0 0 1 1 1 1 0 0];
       vc=[-1 -1 -1 0 0 -1 -1 -1];
       for j=1:8
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
end;
```

```
%================== Sector B ===================================
% secteur B
if (s==2)
   if (delta == 1) % region 1
       t1=[tb/4 ta/2 t0/2 tb/2 ta/2 t0/2 tb/4 tb/4 t0/2 ta/2 tb/2 t0/2 ta/2
tb/4];
       t1=cumsum(t1);
       va=[-1 -1 0 0 0 1 1 1 1 0 0 0 -1 -1];
       vb=[-1 0 0 0 1 1 1 1 1 1 0 0 0 -1];
       vc=[-1 -1 -1 0 0 0 1 1 0 0 0 -1 -1 -1];
       for j=1:14
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
   if (delta == 2)                          % region 2
       t1=[ta/2 t0 tb ta/2 ta/2 tb t0 ta/2];
       t1=cumsum(t1);
       va=[0 0 1 1 1 1 0 0];
       vb=[0 1 1 1 1 1 1 0];
       vc=[-1 -1 -1 0 0 -1 -1 -1];
       for j=1:8
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
   if (delta == 3)                          % region 3
       t1=[ta/2 t0/2 tb t0/2 ta/2 ta/2 t0/2 tb t0/2 ta/2];
       t1=cumsum(t1);
       va=[-1 0 0 0 1 1 0 0 0 -1];
       vb=[0 0 1 1 1 1 1 1 0 0];
       vc=[-1 -1 -1 0 0 0 0 -1 -1 -1];
       for j=1:10
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
   if (delta == 4)% region 4
       t1=[ta/2 tb t0 ta/2 ta/2 t0 tb ta/2];
       t1=cumsum(t1);
       va=[-1 -1 0 0 0 0 -1 -1];
       vb=[0 1 1 1 1 1 1 0];
       vc=[-1 -1 -1 0 0 -1 -1 -1];
       for j=1:8
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
end;
```

```
%=================== Sector C ===================================
% secteur C
if (s==3)
   if (delta == 1) % region 1
       t1=[tb/4 ta/2 t0/2 tb/2 ta/2 t0/2 tb/4 tb/4 t0/2 ta/2 tb/2 t0/2 ta/2
tb/4];
       t1=cumsum(t1);
       va=[-1 -1 -1 0 0 0 1 1 0 0 0 -1 -1 -1];
       vb=[-1 0 0 0 1 1 1 1 1 1 0 0 0 -1];
       vc=[-1 -1 0 0 0 1 1 1 1 0 0 0 -1 -1];
       for j=1:14
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
   if (delta == 2)                          % region 2
       t1=[ta/2 t0 tb ta/2 ta/2 tb t0 ta/2];
       t1=cumsum(t1);
       va=[-1 -1 -1 0 0 -1 -1 -1];
       vb=[0 1 1 1 1 1 1 0];
       vc=[-1 -1 0 0 0 0 -1 -1];
       for j=1:8
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
   if (delta == 3)                          % region 3
       t1=[ta/2 t0/2 tb t0/2 ta/2 ta/2 t0/2 tb t0/2 ta/2];
       t1=cumsum(t1);
       va=[-1 -1 -1 0 0 0 0 -1 -1 -1];
       vb=[0 0 1 1 1 1 1 1 0 0];
       vc=[-1 0 0 0 1 1 0 0 0 -1];
       for j=1:10
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
   if (delta == 4)% region 4
       t1=[ta/2 tb t0 ta/2 ta/2 t0 tb ta/2];
       t1=cumsum(t1);
       va=[-1 -1 -1 0 0 -1 -1 -1];
       vb=[0 1 1 1 1 1 1 0];
       vc=[0 0 1 1 1 1 0 0];
       for j=1:8
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
end
```

```
%=================== Sector D ===================================
% secteur D
if (s==4)
   if (delta == 1) % region 1
       t1=[tb/4 ta/2 t0/2 tb/2 ta/2 t0/2 tb/4 tb/4 t0/2 ta/2 tb/2 t0/2 ta/2
tb/4];
       t1=cumsum(t1);
       va=[-1 -1 -1 0 0 0 1 1 0 0 0 -1 -1 -1];
       vb=[-1 -1 0 0 0 1 1 1 1 0 0 0 -1 -1];
       vc=[-1 0 0 0 1 1 1 1 1 1 0 0 0 -1];
       for j=1:14
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
   if (delta == 2)                          % region 2
       t1=[ta/2 t0 tb ta/2 ta/2 tb t0 ta/2];
       t1=cumsum(t1);
       va=[-1 -1 -1 0 0 -1 -1 -1];
       vb=[0 0 1 1 1 1 0 0];
       vc=[0 1 1 1 1 1 1 0];
       for j=1:8
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
   if (delta == 3)                          % region 3
       t1=[ta/2 t0/2 tb t0/2 ta/2 ta/2 t0/2 tb t0/2 ta/2];
       t1=cumsum(t1);
       va=[-1 -1 -1 0 0 0 0 -1 -1 -1];
       vb=[-1 0 0 0 1 1 0 0 0 -1];
       vc=[0 0 1 1 1 1 1 1 0 0];
       for j=1:10
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
   if (delta == 4)% region 4
       t1=[ta/2 tb t0 ta/2 ta/2 t0 tb ta/2];
       t1=cumsum(t1);
       va=[-1 -1 -1 0 0 -1 -1 -1];
       vb=[-1 -1 0 0 0 0 -1 -1];
       vc=[0 1 1 1 1 1 1 0];
       for j=1:8
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
end;
```

```
%=================== Sector E ================================
% secteur E
if (s==5)
   if (delta == 1) % region 1
       t1=[tb/4 ta/2 t0/2 tb/2 ta/2 t0/2 tb/4 tb/4 t0/2 ta/2 tb/2 t0/2 ta/2
tb/4];
       t1=cumsum(t1);
       va=[-1 -1 0 0 0 1 1 1 1 0 0 0 -1 -1];
       vb=[-1 -1 -1 0 0 0 1 1 0 0 0 -1 -1 -1];
       vc=[-1 0 0 0 1 1 1 1 1 1 0 0 0 -1];
       for j=1:14
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
   if (delta == 2)                          % region 2
       t1=[ta/2 t0 tb ta/2 ta/2 tb t0 ta/2];
       t1=cumsum(t1);
       va=[-1 -1 0 0 0 0 -1 -1];
       vb=[-1 -1 -1 0 0 -1 -1 -1];
       vc=[0 1 1 1 1 1 1 0];
       for j=1:8
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
   if (delta == 3)                          % region 3
       t1=[ta/2 t0/2 tb t0/2 ta/2 ta/2 t0/2 tb t0/2 ta/2];
       t1=cumsum(t1);
       va=[-1 0 0 0 1 1 0 0 0 -1];
       vb=[-1 -1 -1 0 0 0 0 -1 -1 -1];
       vc=[0 0 1 1 1 1 1 1 0 0];
       for j=1:10
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
   if (delta == 4)          % region 4
       t1=[ta/2 tb t0 ta/2 ta/2 t0 tb ta/2];
       t1=cumsum(t1);
       va=[0 0 1 1 1 1 0 0];
       vb=[-1 -1 -1 0 0 -1 -1 -1];
       vc=[0 1 1 1 1 1 1 0];
       for j=1:8
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
end;
```

```
%================== Sector F ===============================
% secteur F
if (s==6)
   if (delta == 1) % region 1
       t1=[tb/4 ta/2 t0/2 tb/2 ta/2 t0/2 tb/4 tb/4 t0/2 ta/2 tb/2 t0/2 ta/2
tb/4];
       t1=cumsum(t1);
       va=[-1 0 0 0 1 1 1 1 1 1 0 0 0 -1];
       vb=[-1 -1 -1 0 0 0 1 1 0 0 0 -1 -1 -1];
       vc=[-1 -1 0 0 0 1 1 1 1 0 0 0 -1 -1];
       for j=1:14
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
   if (delta == 2)                          % region 2
       t1=[ta/2 t0 tb ta/2 ta/2 tb t0 ta/2];
       t1=cumsum(t1);
       va=[0 1 1 1 1 1 1 0];
       vb=[-1 -1 -1 0 0 -1 -1 -1];
       vc=[0 0 1 1 1 1 0 0];
       for j=1:8
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
   if (delta == 3)                          % region 3
       t1=[ta/2 t0/2 tb t0/2 ta/2 ta/2 t0/2 tb t0/2 ta/2];
       t1=cumsum(t1);
       va=[0 0 1 1 1 1 1 1 0 0];
       vb=[-1 -1 -1 0 0 0 0 -1 -1 -1];
       vc=[-1 0 0 0 1 1 0 0 0 -1];
       for j=1:10
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
   if (delta == 4)% region 4
       t1=[ta/2 tb t0 ta/2 ta/2 t0 tb ta/2];
       t1=cumsum(t1);
       va=[0 1 1 1 1 1 1 0];
       vb=[-1 -1 -1 0 0 -1 -1 -1];
       vc=[-1 -1 0 0 0 0 -1 -1];
       for j=1:8
           if(y<t1(j))
              break;
           end;
       end;
       sa=va(j);sb=vb(j);sc=vc(j);
   end;
end;
% Envoi resultats de la fonction
sf = [sa, sb, sc, ta, tb, t0, y];
```

D.3 Modélisation SVPWM à deux niveaux en VHDL

D.3.1 Schéma RTL

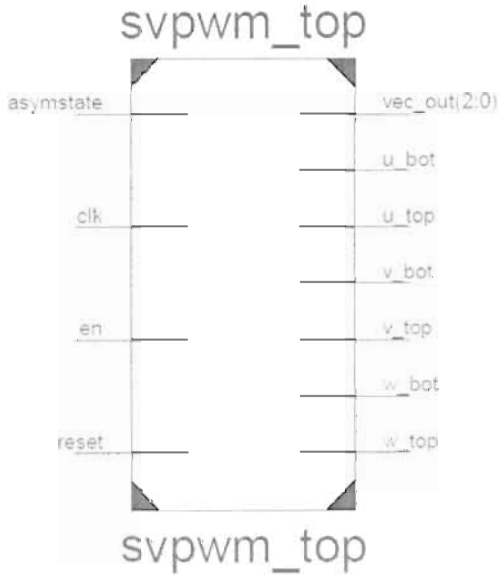


Fig. D.3.1 : Top schéma RTL

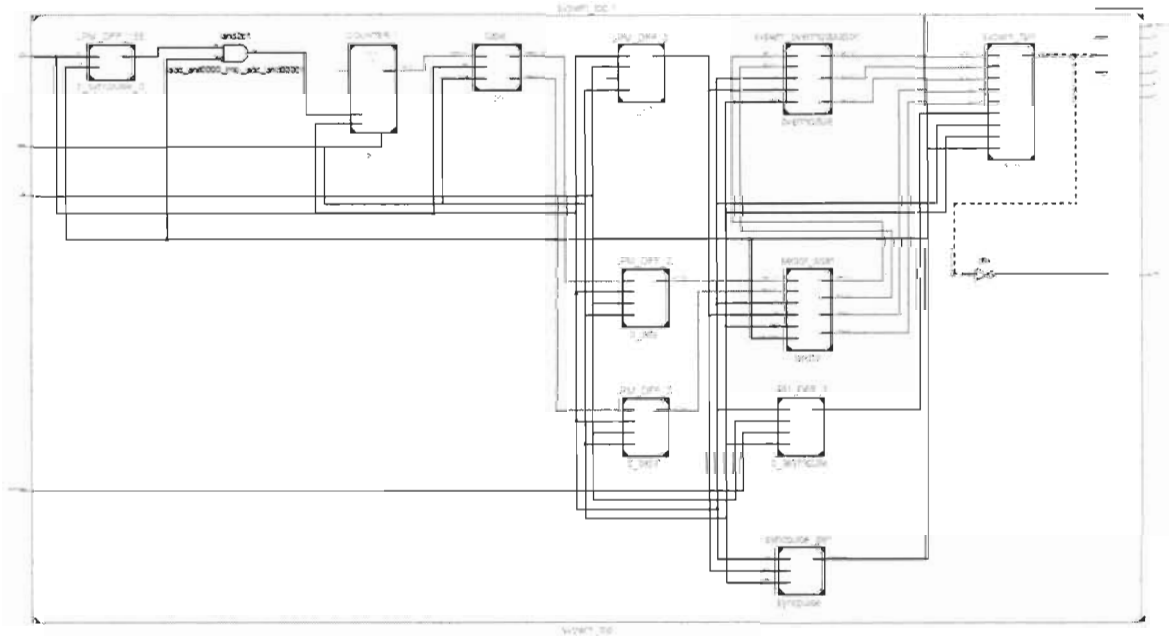


Fig. D.3.2 : Schéma RTL détaillé

D.3.2 Fonctions VHDL

D.3.2.1 Fonction SVPWM Top (svpwm_top)

```
-- Cette fonction est la fonction principale qui réunit tous les
-- sous-fonction de l'algorithme SVPWM.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity svpwm_top is
      Generic (
                alfa_beta_width : integer range 1 to 20 := 18

                );

    Port (
     clk    : in std_logic;
     reset  : in std_logic;
     en      : in std_logic;
     asymstate : in std_logic;

     vec_out : out std_logic_vector(2 downto 0);
     u_top   : out std_logic;
     u_bot   : out std_logic;
     v_top   : out std_logic;
     v_bot   : out std_logic;
     w_top   : out std_logic;
     w_bot   : out std_logic
     );
end svpwm_top;

architecture Behavioral of svpwm_top is

          signal b_en     : std_logic;
          signal b_alfa   : std_logic_vector (alfa_beta_width-1
downto 0);
          signal b_beta   : std_logic_vector (alfa_beta_width-1
downto 0);
          signal alfa   : std_logic_vector (alfa_beta_width-1
downto 0);
          signal beta     : std_logic_vector (alfa_beta_width-1
downto 0);
          signal b_asymstate: std_logic;

          signal b_vec_out : std_logic_vector (2 downto 0);
          signal b_u_top   : std_logic;
          signal b_u_bot   : std_logic;
          signal b_v_top   : std_logic;
          signal b_v_bot   : std_logic;
          signal b_w_top   : std_logic;
```

```
           signal b_w_bot   : std_logic;

           signal b_ta : std_logic_vector (alfa_beta_width-1 downto
0);
           signal b_tb : std_logic_vector (alfa_beta_width-1 downto
0);
           signal b_n_ta : std_logic_vector (alfa_beta_width-1
downto 0);
           signal b_n_tb : std_logic_vector (alfa_beta_width-1
downto 0);
           signal b_n_t0 : std_logic_vector (alfa_beta_width-1
downto 0);
           signal b_Vx : std_logic_vector (2 downto 0);
           signal b_Vy : std_logic_vector (2 downto 0);
           signal b_syncpulse : std_logic;
           signal b_syncpulse_d : std_logic;
           signal add : std_logic_vector (5 downto 0);
      --
      -- components
      --

      -- sector_scan
      component sector_scan is
        port(
                  -- input port
                  clk             :   in std_logic;
                  reset          :   in std_logic;
               en            :   in std_logic;
               ma            :   in std_logic_vector(alfa_beta_width-1
downto 0);
               mb            :   in std_logic_vector(alfa_beta_width-1
downto 0);
               syn_pulse     :   in std_logic;
               -- output port
               ta            :   out std_logic_vector(alfa_beta_width-
1 downto 0);
               tb            :   out std_logic_vector(alfa_beta_width-
1 downto 0);
               Vx            :   out std_logic_vector(2 downto 0);
               Vy            :   out std_logic_vector(2 downto 0)
               );
      end component;

      -- svpwm_overmodulation
      component svpwm_overmodulation is
            generic (
                              last_time_width : integer range 1 to 64
:= alfa_beta_width;
                              ts : std_logic_vector(alfa_beta_width-1
downto 0):= "000010011100010000"

                             );
          Port ( clk        : in   STD_LOGIC;
                 reset      : in   STD_LOGIC;
```

```
                  en          : in   STD_LOGIC;
                ta        : in   STD_LOGIC_VECTOR (last_time_width-
1 downto 0);
                tb        : in   STD_LOGIC_VECTOR (last_time_width-
1 downto 0);
                n_ta : out  STD_LOGIC_VECTOR (last_time_width-1
downto 0);
                n_tb : out  STD_LOGIC_VECTOR (last_time_width-1
downto 0);
                n_t0 : out  STD_LOGIC_VECTOR (last_time_width-1
downto 0)
                     );
    end component;

    -- svpwm_fsm
    component svpwm_fsm is
        Generic (
                V0                  : std_logic_vector (2
downto 0) := "000";
                V1                  : std_logic_vector (2
downto 0) := "111";
                asym_state_6    : integer :=6;
                asym_state_3    : integer :=3;
                cnt_width       : integer :=alfa_beta_width
        );
        Port (
                --
                -- input port
                --
                reset      : in std_logic;
                clk             : in std_logic;
                syncpulse  : in STD_LOGIC;
                asymstate   : in std_logic;
                Vx              : in std_logic_vector (2 downto
0) := "100";
                Vy              : in std_logic_vector (2 downto
0) := "110";
                t0_in      : in std_logic_vector (cnt_width-1
downto 0);
                ta_in      : in std_logic_vector (cnt_width-1
downto 0);
                tb_in      : in std_logic_vector (cnt_width-1
downto 0);
                --
                -- output port
                --
                vec_out         : out std_logic_vector (2
downto 0)
            );
    end component;

    -- table
    component table is
        Port(
```

```
          --
          -- input port
          --
          clk                  : in std_logic;
          reset           : in std_logic;
          add                  : in std_logic_vector( 5 downto 0);
          --
          -- output port
          --
          datax           : out std_logic_vector ( 17 downto 0);
          datay           : out std_logic_vector ( 17 downto 0)
     );
     end component;

     -- syncpulse_gen
     component syncpulse_gen is

     Generic (
          C_COUNTER_WIDTH : integer := alfa_beta_width;
          ts                   : integer := 5000
     );

    Port (
     --
     -- input signal
     --
          clk         : in std_logic;
          reset  : in std_logic;
          ena    : in std_logic;
          --
          -- output signal
          --
          syncpulse : out std_logic
     );

     end component;

--     deadzone
     component deadzone is
           Generic (
                     count_width : integer range 1 to 8 :=4;
                     deadtime    : integer range 1 to 128 := 10
                     );

         Port (
          --
          -- input port
          --
          clk        : in std_logic;
          reset   : in std_logic;
          pwm_in  : in std_logic;

          --
          -- output port
```

```
            --
            pwm_top  : out std_logic;
            pwm_bot  : out std_logic

            );
      end component;

begin
      process(clk) begin
            if (rising_edge(clk)) then
                  if (reset = '0') then
                        b_en <= '0';
                        b_alfa <= (others =>'0');
                        b_beta <= (others =>'0');
                        b_asymstate <= '0';
                  else
                        if (en = '1') then
                              b_en <= en;
                              b_asymstate <= asymstate;
                              b_alfa <= alfa;
                              b_beta <= beta;
                        end if;
                  end if;
            end if;
      end process;

      process (clk) begin
            if (rising_edge(clk)) then
                  b_syncpulse_d <= b_syncpulse;
            end if;
      end process;

      process (clk) begin
            if (rising_edge(clk)) then
                  if (reset = '0') then
                        add <= "000000";
                  else
                        if (b_syncpulse = '0' and b_syncpulse_d = '1')
then
                              add <= add + 1;
                        end if;
                  end if;
            end if;
      end process;

      vec_out <= b_vec_out;
      u_top  <= b_vec_out(2);
      u_bot  <= not b_vec_out(2);
      v_top  <= b_vec_out(1);
      v_bot  <= not b_vec_out(1);
      w_top  <= b_vec_out(0);
      w_bot  <= not b_vec_out(0);
      sector: sector_scan PORT MAP(
                  -- input port
```

```
            clk        => clk,
            reset       => reset,
          en            => b_en,
          ma            => b_alfa,
          mb            => b_beta,
          syn_pulse     => b_syncpulse,
          -- output port
          ta      =>b_ta,
          tb      =>b_tb,
          Vx      =>b_Vx,
          Vy      =>b_Vy
          );
fsm: svpwm_fsm port map(
            --
            -- input port
            --
            reset      => reset,
            clk             => clk,
            syncpulse  => b_syncpulse,
            asymstate   => b_asymstate,
            Vx              => b_Vx,
            Vy              => b_Vy,
            t0_in      => b_n_t0,
            ta_in      => b_n_ta,
            tb_in      => b_n_tb,
            --
            -- output port
            --
            vec_out         => b_vec_out
            );
syncpulse : syncpulse_gen port map(
            --
            -- input signal
            --
            clk         => clk,
            reset  => reset,
            ena    => b_en,
            --
            -- output signal
            --
            syncpulse => b_syncpulse
            );
overmodule : svpwm_overmodulation port map(
            clk   => clk,
            reset      => reset,
            en         => b_en,
            ta         => b_ta,
            tb         => b_tb,
            n_ta  => b_n_ta,
            n_tb  => b_n_tb,
            n_t0  => b_n_t0
            );
u_deadzone : deadzone port map(
            --
```

```
                -- input port
                --
                clk        => clk,
                reset   => reset,
                pwm_in  => b_vec_out(2),
                --
                -- output port
                --
                pwm_top  => b_u_top,
                pwm_bot  => b_u_bot
                );
    v_deadzone : deadzone port map(
                --
                -- input port
                --
                clk        => clk,
                reset   => reset,
                pwm_in  => b_vec_out(1),
                --
                -- output port
                --
                pwm_top =>  b_v_top,
                pwm_bot =>  b_v_bot
                );
    w_deadzone : deadzone port map(
                --
                -- input port
                --
                clk        => clk,
                reset   => reset,
                pwm_in  => b_vec_out(0),
                --
                -- output port
                --
                pwm_top  => b_w_top,
                pwm_bot  => b_w_bot
                );
    rom: table PORT MAP(
          --
          -- input port
          --
          clk    => clk  ,
          reset => reset,
          add    => add  ,
          --
          -- output port
          --
          datax => alfa,
          datay => beta
    );

end Behavioral;
```

D.3.2.2 Fonction Table de vecteur SVPWM (table)

```
-- Cette fonction contient la table des vecteurs espaces

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity table is
     Port(
          clk                  : in std_logic;
          reset           : in std_logic;
          add                  : in std_logic_vector( 5 downto 0);

          datax           : out std_logic_vector ( 17 downto 0);
          datay           : out std_logic_vector ( 17 downto 0)
     );
end table;

architecture Behavioral of table is

          type memory is array (0 to 63) of std_logic_vector(17
downto 0);
          constant table : memory := (
          --
"+00000000+00000000","+00000000+00000000","+00000000+00000000","+000
00000+00000000",

     "01001111000000010","01001111000010010","01001110000100001",
"01001100000110001",

     "01001001000011111","01000101100100l101","01000001100101l011",
"001111001001100111",

     "001101111001110011","001100011001111101","001010110010000110",
"001001000010001110",

     "000111010010010100","000101011010011001","000011100010011101",
"000001100010011111",

     "100000010010011111","100010010010011110","100100001010011100",
"100110001010011000",

     "100111111010010010","101001101010001011","101011011010000011",
"101100111001111001",

     "101110011001101111","101111101001100011","110000110001010110",
"110001110001001000",

     "110010100000111010","110011001000101011","110011101000011100",
"110011111000001100",
```

```
      "1100111111000000010","1100111101000010010","1100111001001000001",
"1100110001001100001",

      "1100100101001111111","1100010111010011101","1100000111010110011",
"1011110011011001111",

      "1011011111011100011","1011000111011111101","1010101101100001110",
"1010010001100011110",

      "1001110101100101000","1001010111100110001","1000111001100111101",
"1000011001100111111",

      "0000000101100111111","0000100101100111110","0001000011100111000",
"0001100011100110000",

      "0001111111100100010","0010011011100010011","0010110111100000011",
"0011001111011110001",

      "0011100111011011111","0011111011011000011","0100001101010101110",
"0100011101010010000",

      "0100101001001110010","0100110011001010011","0100111011000111000",
"0100111111000011000"
           );
           constant table1 : memory := (
           --
"+00000000+00000000","+00000000+00000000","+00000000+00000000","+000
00000+00000000",

      "0011111110000000010","0011111110000011110","0011111010000110110",
"0011110010001001111",

      "0011101010001100110","0011011110001111100","0011010010010010000",
"0011000010010100100",

      "0010110000101110000","0010011110011001000","0010001010011010110",
"0001110100011100010",

      "0001011100011101111","0001000110011110110","0000101100011111010",
"0000010100011111111",

      "1000000100011111111","1000011100011111111","1000110110011111010",
"1001001110011110010",

      "1001100110011101010","1001111100011011110","1010010000011010010",
"1010100100011000010",

      "1010111000101100000","1011001000010011110","1011010110010001010",
"1011100010001110100",

      "1011101110001010110","1011110110001000110","1011111010000101100",
"1011111110000010100",
```

```
     "101111111100000010","101111111100001110","101111101100011011",
"101111001100100111",

     "101110101100110011","101101111100111110","101101001101001000",
"101100001101010010",

     "101011000101011100","101001111101100100","101000101101101011",
"100111010101110001",

     "100101110101110111","100100011101111011","100010110101111101",
"100001010101111111",

     "000000010101111111","000001110101111111","000011011101111101",
"000100111101111001",

     "000110011101110101","000111110101101111","001001000101101001",
"001010010101100001",

     "001011100101011000","001100100101001111","001101011101000101",
"001110001100111010",

     "001110111100101110","001111011100100011","001111101100010110",
"001111111100001010"
          );


begin
     process(clk) begin
          if (rising_edge(clk)) then
               if (reset = '0') then
                    datax <= (others => '0');
                    datay <= (others => '0');
               else
                    datax <= table1(conv_integer (add))(17) &
"000000000" & table1(conv_integer (add))(16 downto 9);
                    datay <= table1(conv_integer (add))(8) &
"000000000" & table1(conv_integer (add)) (7 downto 0);

               end if;
          end if;
     end process;
end Behavioral;
```

D.3.2.3 Fonction calcul secteur (sec_find_comp)

```
-- Cette fonction permet de calculer le secteur dans lequel se
-- trouve le vecteur de référence

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
Library UNISIM;
use UNISIM.vcomponents.all;

entity sector_scan is
     port(  clk       : in std_logic;
            reset     : in std_logic;
            en        : in std_logic;
            ma        : in std_logic_vector(17 downto 0);
            mb        : in std_logic_vector(17 downto 0);
            syn_pulse : in std_logic;

            ta        : out std_logic_vector(17 downto 0);
            tb        : out std_logic_vector(17 downto 0);
             Vx        : out std_logic_vector(2 downto 0);
             Vy        : out std_logic_vector(2 downto 0)
      );
end sector_scan;

architecture Behavioral of sector_scan is

signal
data_a_in,data_b_in,data_a_3,data1,data2,data3,data4,data5:std_logic
_vector(17 downto 0);
signal data_shift_a1,data_shift_a2 :std_logic_vector(17 downto 0);
signal ta_freq,tb_freq:std_logic_vector(35 downto 0);
signal sure :std_logic;
signal index_ab:std_logic_vector(1 downto 0):="00";
signal sector_pos:std_logic_vector(2 downto 0);
signal syn_pulse_delay : std_logic;
signal d,e,ta_s,tb_s: std_logic_vector(17 downto 0);

signal ts: std_logic_vector(17 downto 0) :="000000001111101000";
signal
data_shift_b1,data_shift_b4,data_shift_b11,data_shift_b8:std_logic_v
ector(17 downto 0);
signal data_b_0433,data_b_0866,data_a_075:std_logic_vector(17 downto
0);
signal data_ma,data_mb:std_logic_vector(17 downto 0);
signal Vx_out :std_logic_vector(2 downto 0);
signal Vy_out :std_logic_vector(2 downto 0);

begin

process(clk,ma,mb,en,reset)      begin
     if (clk'event and clk='1')  then
```

```
          if (reset = '0') then
               data_ma    <= ( others => '0');
               data_mb    <= ( others => '0');
               data_a_in  <= ( others => '0');
               data_b_in  <= ( others => '0');
               index_ab   <= ( others => '0');
               syn_pulse_delay <= syn_pulse;
          else
               data_ma<=ma;
          data_mb<=mb;
          syn_pulse_delay<=syn_pulse;
          if (en='1' and syn_pulse = '1' and syn_pulse_delay = '0')
then
               data_a_in <=   '0' &  data_ma(16 downto 0);
                    data_b_in <=   '0' &  data_mb(16 downto 0);

                    index_ab  <=   data_ma(17) & data_mb(17);
          end if;
          end if;
     end if;
end process;

process(data_a_in,clk) begin
     if (clk'event and clk='1') then
          data1 <= data_a_in(16 downto 0) & '0';
          data2<= "00" & data_a_in(17 downto 2);
          data3<= "000000" & data_a_in(17 downto 6);
          data4<= "000000000" & data_a_in(17 downto 9);
          data_a_3<=data1-data2-data3-data4;
     end if;
end process;

process(data_a_3,clk) begin
     if (data_a_3>=data_b_in) then
          sure<='1';
     else
          sure<='0';
     end if;
end process;

process(sure,index_ab) begin
     case index_ab is
          when "00"  =>
                   if (sure='1')  then
                       sector_pos<="000";
                   else
                       sector_pos<="001";
                   end if;
          when "10"  =>
                   if (sure='1')  then
                       sector_pos<="011";
                   else
                       sector_pos<="010";
                   end if;
```

```
          when "11"  =>
                 if (sure='1')  then
                      sector_pos<="100";
                  else
                      sector_pos<="101";
                  end if;

          when others   =>
                 if (sure='1')  then
                      sector_pos<="111";
                    else
                      sector_pos<="110";
                  end if;
      end case;
end process;

process(data_a_in,clk) begin
     if (clk'event and clk='1') then
          if (reset = '0') then
               data_shift_a1 <= (others => '0');
               data_shift_a2 <= (others => '0');
               data_a_075    <= (others => '0');
          else
               data_shift_a1<='0'  & data_a_in(17 downto 1);

               data_shift_a2<="00" & data_a_in(17 downto 2);

               data_a_075 <= data_shift_a1 + data_shift_a2;
          end if;
     end if;
end process;

process(data_b_in,clk) begin
     if (clk'event and clk='1') then
          if (reset = '0') then
               data_shift_b1   <= (others => '0');
               data_shift_b4   <= (others => '0');
               data_shift_b8   <= (others => '0');
               data_shift_b11  <= (others => '0');
               data_b_0433     <= (others => '0');
          else
               data_shift_b1 <='0'            &    data_b_in(17
downto 1);
               data_shift_b4 <="0000"         &    data_b_in(17
downto 4);
              data_shift_b8 <="00000000"     &    data_b_in(17
downto 8);
              data_shift_b11<="00000000000"  &    data_b_in(17
downto 11);
               data_b_0433<=data_shift_b1-data_shift_b4-
data_shift_b8-data_shift_b11;
          end if;
     end if;
```

```
end process;

process(data_b_0433,clk) begin
     if (clk'event and clk='1') then
          if (reset = '0') then
               data_b_0866 <= (others => '0');
          else
               data_b_0866<=data_b_0433(16 downto 0) & '0';
          end if;
     end if;
end process;

process(sector_pos,clk)    begin
     if(rising_edge (clk)) then
          if(reset = '0') then
               ta_s <= (others => '0');
               tb_s <= (others => '0');
               Vx_out <= "000";
               Vy_out <= "000";
          else
               case sector_pos is
               when "000"  =>
                    ta_s<=data_a_075-data_b_0433;
                    tb_s<=data_b_0866;
                    Vx_out<="100";
                    Vy_out<="110";

               when "001"  =>
                    tb_s<=data_b_0433-data_a_075;
                    ta_s<=data_b_0433+data_a_075;
                    Vy_out<="110";
                    Vx_out<="010";

                when "010" =>
                    tb_s<=data_b_0433+data_a_075;
                    ta_s<=data_b_0433-data_a_075;
                    Vy_out<="110";
                    Vx_out<="010";

                when "011" =>
                    ta_s<=data_b_0866;
                    tb_s<=data_a_075-data_b_0433;
                    Vx_out<="010";
                    Vy_out<="011";

                when "100" =>
                    tb_s<=data_b_0866;
                    ta_s<=data_a_075-data_b_0433;
                    Vy_out<="011";
                    Vx_out<="001";

                when "111"  =>
                    tb_s<=data_a_075-data_b_0433;
                    ta_s<=data_b_0866;
```

```
                    Vy_out<="101";
                    Vx_out<="100";
                when "110" =>
                    ta_s<=data_b_0433-data_a_075;
                    tb_s<=data_b_0433+data_a_075;
                    Vx_out<="001";
                    Vy_out<="101";

                when others =>
                    ta_s<=data_b_0433+data_a_075;
                    tb_s<=data_b_0433-data_a_075;
                    Vx_out<="001";
                    Vy_out<="101";
              end case;
          end if;
     end if;
 Vx<=Vx_out;
 Vy<=Vy_out;

end process;

process(clk,ta_s) begin
    if (clk'event and clk='1') then
     tb<=tb_freq(25 downto 8);
     ta<=ta_freq(25 downto 8);
     end if;

end process;

MULT18X18_inst1 : MULT18X18SIO
     generic map(
                  AREG => 1,
                  BREG => 1,
                  B_INPUT => "DIRECT",
                   PREG => 1
                   )
     port map(
              BCOUT =>d ,
              P => ta_freq,
              A =>ta_s,
              B => ts,
              BCIN =>"000000000000000000",
              CEA => '1',
              CEB => '1',
              CEP => '1',
              CLK => CLK,
              RSTA => '0',
              RSTB => '0',
              RSTP => '0'
              );

MULT18X18_inst2 : MULT18X18SIO
     port map(
              BCOUT =>e ,
```

```
                P => tb_freq,
                A =>tb_s,
                B => ts,
                BCIN =>"0000000000000000000",
                CEA => '1',
                CEB => '1',
                CEP => '1',
                CLK => CLK,
                RSTA => '0',
                RSTB => '0',
                RSTP => '0'
                );
end Behavioral;
```

D.3.2.4 Fonction calcule les temps de commutation (svpwm_overmodulation)

```
-- Cette fonction permet de calculer les temps de commutation

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity svpwm_overmodulation is
      generic (
                        last_time_width : integer range 1 to 64 :=
18;
                        ts : std_logic_vector(17 downto 0):=
"000000001111101000"
                       );
    Port ( clk        : in   STD_LOGIC;
           reset      : in   STD_LOGIC;
              en           : in   STD_LOGIC;
           ta         : in   STD_LOGIC_VECTOR (last_time_width-1
downto 0);
           tb         : in   STD_LOGIC_VECTOR (last_time_width-1
downto 0);
           n_ta : out  STD_LOGIC_VECTOR (last_time_width-1 downto
0);
           n_tb : out  STD_LOGIC_VECTOR (last_time_width-1 downto
0);
           n_t0 : out  STD_LOGIC_VECTOR (last_time_width-1 downto 0)
                  );
end svpwm_overmodulation;

architecture Behavioral of svpwm_overmodulation is

signal ta_s       : std_logic_vector (last_time_width-1 downto 0)
:=(others => '0');
signal tb_s       : std_logic_vector (last_time_width-1 downto 0)
:=(others => '0');
signal n_ta_s     : std_logic_vector (last_time_width-1 downto 0)
:=(others => '0');
signal n_tb_s     : std_logic_vector (last_time_width-1 downto 0)
:=(others => '0');
signal n_t0_s     : std_logic_vector (last_time_width-1 downto 0)
:=(others => '0');
signal t_over     : std_logic_vector (last_time_width-1 downto 0)
:=(others => '0');
signal en_state   : std_logic := '0';
signal ta_and_tb  : std_logic_vector (last_time_width-1 downto 0 );

begin
process(clk, reset)
     begin
          if (reset = '0') then
               n_ta_s <= (others => '0');
```

```
                n_tb_s <= (others => '0');
                n_t0_s <= (others => '0');
                t_over <= (others => '0');
           elsif rising_edge(clk) then
                ta_s <= ta;
                tb_s <= tb;
                if (en = '1') then
                     ta_and_tb <= ta_s + tb_s;
                     if (ta_and_tb < '0' & ts(last_time_width-1
downto 1)) then
                          n_ta_s <= ta_s;
                          n_tb_s <= tb_s;
                          n_t0_s <= '0' & ts(last_time_width-1
downto 1) - ta_and_tb;
                     else
                          if (ta_and_tb = '0' & ts(last_time_width-
1 downto 1)) then
                               n_ta_s <= ta_s;
                               n_tb_s <= tb_s;
                               n_t0_s <= (others => '0');
                          else
                               t_over <= ('0' &
ta_and_tb(last_time_width-1 downto 1)) - ("00" & ts(last_time_width-
1 downto 2));
                               if (ta_s < t_over) then
                                    n_ta_s <= (others => '0');
                                    n_tb_s <= '0' &
ts(last_time_width-1 downto 1);
                                    n_t0_s <= (others => '0');
                               elsif (tb_s < t_over) then
                                    n_tb_s <= (others => '0');
                                    n_ta_s <= '0' &
ts(last_time_width-1 downto 1);
                                    n_t0_s <= (others => '0');
                               else
                                    n_ta_s <= ta_s - t_over;
                                    n_tb_s <= tb_s - t_over;
                                    n_t0_s <= (others => '0');
                               end if;
                          end if;
                     end if;
                end if;
           end if;
           n_ta <= n_ta_s;
        n_tb <= n_tb_s;
        n_t0 <= n_t0_s;
     end process;
end Behavioral;
```

D.3.2.5 Fonction qui génère les vecteurs espaces (svpwm_fsm)

```
-- Cette fonction permet de génèrer les vecteur espaces SVPWM

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity svpwm_fsm is
      Generic (
                V0         : std_logic_vector (2 downto 0) := "000";
                V1         : std_logic_vector (2 downto 0) := "111";
                asym_state_6     : integer :=6;
                asym_state_3     : integer :=3;
                cnt_width        : integer :=18
      );
      Port (

                reset      : in std_logic;
                clk              : in std_logic;
                syncpulse  : in STD_LOGIC;
                asymstate   : in std_logic;
                Vx    : in std_logic_vector (2 downto 0) := "100";
                Vy    : in std_logic_vector (2 downto 0) := "110";
                t0_in : in std_logic_vector (cnt_width-1 downto 0);
                ta_in : in std_logic_vector (cnt_width-1 downto 0);
                tb_in : in std_logic_vector (cnt_width-1 downto 0);

                vec_out    : out std_logic_vector (2 downto 0)

          );
end svpwm_fsm;

architecture Behavioral of svpwm_fsm is
type state is (
          s0,
          s1,
          s2,
          s3
     );

signal b_syncpulse              : std_logic;
signal b_syncpulse_delay   : std_logic;
signal en_signal                : std_logic;
signal sp_state            : std_logic := '0';
signal currentstate             : state := s0;
signal laststate           : state := s0;
signal asym                          : std_logic;
signal vec_cnt                  : std_logic_vector ( cnt_width-1
downto 0);
signal state_cnt           : std_logic_vector ( 3 downto 0);
```

```
signal Vx_s          : std_logic_vector (2 downto 0) := "100";
signal Vy_s          : std_logic_vector (2 downto 0) := "110";
signal t0            : std_logic_vector ( cnt_width-1 downto 0);
signal ta            : std_logic_vector ( cnt_width-1 downto 0);
signal tb            : std_logic_vector ( cnt_width-1 downto 0);

begin
     process (clk) begin
          if (rising_edge(clk)) then
               if (reset = '0') then
                    b_syncpulse <= '0';
               else
                    b_syncpulse <= syncpulse;
               end if;
          end if;
     end process;

     process (clk) begin
          if (rising_edge(clk)) then
               if (reset ='0') then
                    en_signal <= '0';
                    b_syncpulse_delay <= '0';

               else
                    b_syncpulse_delay <= b_syncpulse;
                    if (b_syncpulse = '0' and b_syncpulse_delay =
'1') then
                         en_signal <= '1';
                    end if;
               end if;
               if (en_signal = '1') then

                    en_signal <= '0';
               end if;
          end if;
     end process;

     process (clk) begin
          if (rising_edge(clk)) then
               if (reset = '0') then
                    sp_state    <= '0';
                    currentstate <= s0;

                    laststate   <= s0;
                    asym        <= '0';
                    vec_cnt          <= (others => '0');
                    state_cnt   <= "0000";
                    Vx_s        <= "000";
                    Vy_s        <= "000";
                    t0               <= (others => '0');
                    ta               <= (others => '0');
                    tb               <= (others => '0');

               else
```

```
                    if (en_signal = '1') then
                         sp_state <= '1';
                         currentstate <= s0;

                         laststate   <= s0;
                         asym        <= asymstate;
                         vec_cnt          <= (others => '0');
                         state_cnt   <= "0000";
                         Vx_s        <= Vx;
                         Vy_s        <= Vy;
                         t0               <= ('0' &
t0_in(cnt_width-1 downto 1)) + 1;
                         ta               <= ta_in + 1;
                         tb               <= tb_in + 1;
                    elsif (sp_state = '1') then
                         case currentstate is
                              when s0 =>
                                   vec_cnt <= vec_cnt+1;

                                   if (asym = '1') then
                                        if
(conv_integer(state_cnt) = asym_state_6) then
                                             sp_state <= '0';
                                             state_cnt <=(others
=> '0');
                                        end if;
                                        if (vec_cnt = (('0' &
t0)-1) ) then
                                             vec_cnt <= (others
=> '0');
                                             laststate <= s0;
                                             currentstate <= s1;
                                             state_cnt <=
state_cnt+1;
                                        end if;
                                   else
                                        if
(conv_integer(state_cnt) = asym_state_3 ) then
                                             sp_state <= '0';
                                             state_cnt <=(others
=> '0');
                                        end if;

                                        if (laststate = s0) then
                                             vec_cnt <= (others
=> '0');
                                             laststate <= s0;
                                             currentstate <= s3;
                                             state_cnt <=
state_cnt+1;
                                        end if;
                                   end if;

                              when s1 =>
```

```
                                        vec_cnt <= vec_cnt+1;
                                        if (asym = '1') then
                                             if (vec_cnt = (('0' &
ta)-1) and laststate = s0) then
                                                  vec_cnt <= (others
=> '0');
                                                  laststate <= s1;
                                                  currentstate <= s2;
                                                  state_cnt <=
state_cnt+1;
                                             end if;

                                             if (vec_cnt = (('0' &
ta)-1) and laststate = s2) then
                                                  vec_cnt <= (others
=> '0');
                                                  laststate <= s1;
                                                  currentstate <= s0;
                                                  state_cnt <=
state_cnt+1;
                                             end if;
                                        elsif (vec_cnt = ((ta & '0')-1)
and laststate = s2) then
                                                  vec_cnt <= (others
=> '0');
                                                  laststate <= s1;
                                                  currentstate <= s0;
                                                  state_cnt <=
state_cnt;
                                        end if;

                                   when s2 =>
                                        vec_cnt <= vec_cnt+1;

                                        if (asym = '1') then
                                             if (vec_cnt = (('0' &
tb)-1) and laststate = s1) then
                                                  vec_cnt <= (others
=> '0');
                                                  laststate <= s2;
                                                  currentstate <= s3;
                                                  state_cnt <=
state_cnt+1;
                                             end if;
                                             if (vec_cnt = (('0' &
tb)-1) and laststate = s3) then
                                                  vec_cnt <= (others
=> '0');
                                                  laststate <= s2;
                                                  currentstate <= s1;
                                                  state_cnt <=
state_cnt+1;
                                             end if;
```

```
                                        elsif (vec_cnt = ((tb & '0')-1)
and laststate = s3) then
                                                vec_cnt <= (others
=> '0');
                                                laststate <= s2;
                                                currentstate <= s1;
                                                state_cnt <=
state_cnt+1;
                                        end if;

                                    when s3 =>
                                        vec_cnt <= vec_cnt+1;

                                        if (asym = '1') then
                                            if (vec_cnt = ((t0 &
'0')-1)) then
                                                vec_cnt <= (others
=> '0');
                                                laststate <= s3;
                                                currentstate <= s2;
                                                state_cnt <=
state_cnt+1;
                                            end if;
                                        elsif (vec_cnt = ((t0 & '0')-
1)) then
                                                vec_cnt <= (others
=> '0');
                                                laststate <= s3;
                                                currentstate <= s2;
                                                state_cnt <=
state_cnt+1;
                                        end if;

                                    when others =>
                                                laststate <= s0;
                                                currentstate <=
s0;
                            end case;
                        end if;
                    end if;
                end if;
            end process;

            process(clk) begin
                if (rising_edge(clk)) then
                    if (reset = '0') then
                        vec_out <= "000";
                    else
                        case currentstate is
                            when s0     =>
                                vec_out <= V0;
                            when s1     =>
                                vec_out <= Vx_s;
                            when s2     =>
```

```
                                vec_out <= Vy_s;
                            when s3     =>
                                vec_out <= V1;
                            when others =>
                                vec_out <= "000";
                        end case;
                end if;
            end if;
        end process;

end Behavioral;
```

D.3.2.6 Fonction de synchronisation des impulsions (syncpulse_gen)

```
-- Cette fonction permet de synchroniser les impulsions SVPWM

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity syncpulse_gen is

     Generic (
          C_COUNTER_WIDTH : integer := 12;
          ts                    : integer := 5
     );

    Port (
          clk         : in std_logic;
          reset  : in std_logic;
          ena    : in std_logic;

          syncpulse : out std_logic
     );

end syncpulse_gen;

architecture Behavioral of syncpulse_gen is
     signal b_ena          : std_logic;
     signal b_syncpulse    : std_logic :='0';
     signal count          : std_logic_vector(C_COUNTER_WIDTH-1
downto 0 ) := (others => '0');
     signal down                : std_logic := '0';

begin
     process (clk) begin
          if (rising_edge(clk)) then
               if (reset = '0') then
                    b_ena <= '0';
               else
                    b_ena <= ena;
               end if;
          end if;
     end process;
     process (reset, clk) begin
          if (clk'event and clk = '1' ) then
               if ( reset = '0' ) then
                    count <= (others => '0' );
                    down <= '0';
               else
                    if (b_ena = '1') then
                         if (down = '0') then
                              count <= count+1;
                         else
```

```
                              count <= count-1;
                          end if;
                          if (conv_integer(count) = ts) then
                              down <= '1';
                          end if;
                          if (down = '1' and conv_integer(count) =
1) then
                              down <= '0';
                              b_syncpulse <= '1';
                          end if;
                          if (b_syncpulse = '1') then
                              b_syncpulse <= '0';
                          end if;
                      end if;
                  end if;
              end if;
      end process;
      syncpulse <= not b_syncpulse;
end Behavioral;
```

D.3.2.7 Fonction test bench de la fonction SVPWM Top (svpwm_top_tb)

```
-- Cette fonction permet de tester l'algorithme principal SVPWM
-- en introduisant des données à l'entrée afin de voir le
-- comportement du système


LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY svpwm_top_tb IS
     Generic (
               alfa_beta_width : integer range 1 to 20 := 18

               );
END svpwm_top_tb;

ARCHITECTURE behavior OF svpwm_top_tb IS

     COMPONENT svpwm_top
     PORT(
          clk    : in std_logic;
     reset  : in std_logic;
     alfa   : in std_logic_vector(alfa_beta_width-1 downto 0);
     beta   : in std_logic_vector(alfa_beta_width-1 downto 0);
     en      : in std_logic;
     asymstate : in std_logic;

          vec_out : out std_logic_vector(2 downto 0);
          u_top   : out std_logic;
     u_bot   : out std_logic;
     v_top   : out std_logic;
     v_bot   : out std_logic;
     w_top   : out std_logic;
     w_bot   : out std_logic
          );
     END COMPONENT;

          signal clk    : std_logic :='0';
     signal reset  : std_logic :='0';
     signal alfa   : std_logic_vector(alfa_beta_width-1 downto 0) :=
(others => '0');
     signal beta   : std_logic_vector(alfa_beta_width-1 downto 0) :=
(others => '0');
     signal en    : std_logic :='0';
     signal asymstate :  std_logic :='1';

          signal vec_out : std_logic_vector(2 downto 0);
          signal u_top   : std_logic;
     signal u_bot   : std_logic;
```

```
      signal v_top   : std_logic;
      signal v_bot   : std_logic;
      signal w_top   : std_logic;
      signal w_bot   : std_logic;

BEGIN

      uut: svpwm_top PORT MAP(

            clk    => clk,
            reset  => reset,
            alfa   => alfa,
            beta   => beta,
            en       => en,
            asymstate => asymstate,

            vec_out => vec_out,
            u_top => u_top,
            u_bot => u_bot,
            v_top => v_top,
            v_bot => v_bot,
            w_top => w_top,
            w_bot => w_bot

            );

      reset <= '1' after 40 ns;
      clk   <= not clk after 10 ns;
      tb : PROCESS
      BEGIN
                  wait for 100 ns;
                  en <= '1';
                  alfa <= "0000000000010011110";
                  beta <= "0000000000000010010";
            wait;
      END PROCESS;
END;
```