CHAPITRE 3

Modèle System Generator

3.1 Brève description

Dans le cadre de cette recherche, le développement du prototype s'effectue avec le logiciel XSG. Il s'agit d'un toolbox développé par Xilinx pour être intégré dans l'environnement Matlab-Simulink et qui laisse l'usager créer des systèmes hautement parallèles pour FPGA. Les modèles créés sont affichés sous forme de blocs, et peuvent être raccordés aux autres blocs et autres toolbox de Matlab-Simulink comme SPS. Une fois le système complété, le code VHDL généré par l'outil *XSG* reproduit exactement le comportement observé dans Matlab. Pour le prototypage rapide, le choix de cet outil est facilement explicable. Le système de contrôle devant être vérifié et simulé souvent et rapidement pendant tout le développement, il est beaucoup plus simple d'analyser les résultats avec Matlab qu'avec les outils habituellement associés au VHDL, tel que Modelsim. Aussi, le modèle peut ensuite être couplé à des moteurs virtuels (à l'aide du toolbox SPS) et des simulations en boucle fermée sont réalisables. Quand le prototype fonctionne, le passage vers la plateforme matérielle pour des tests sur le terrain est rapide, ce qui rend la validation du prototype un projet réalisable à court terme.

3.2 Méthode de développement

Avec une banque de résultats provenant de la référence établie, le développement des blocs du prototype peut se faire. L'outil XSG est utilisé afin de produire un modèle qui va tout de suite fonctionner sur le matériel une fois achevé et validé.

3.2.1 Développement de bloc individuel et validation de premier niveau

L'avantage d'utiliser XSG pour le prototypage rapide devient plus évident lorsqu'il est nécessaire de tester un bloc achevé. Il suffit de brancher aux entrées les données intermédiaires obtenues de la référence. En simulant, on recueille les données à la sortie pour ensuite les comparer avec les données de la référence. Notons qu'il peut y avoir une certaine erreur, car une précision arbitraire selon le nombre de bits des opérandes est employée. L'outil GAPPA va plus tard permettre d'ajuster cette précision (voir section 4.2). Pour l'instant, il faut seulement vérifier le comportement du bloc.

Lorsqu'un bloc se comporte bien, ce test s'effectue assez rapidement, car le temps de simulation est plus court. Nous avons déjà mentionné que les signaux intermédiaires ne sont pas emmagasinés en trop grand nombre, de façon à rendre le temps de simulation acceptable. Cependant, lorsqu'un bloc ne répond pas comme il se doit, le travail devient plus important. Dans le cas de la commande vectorielle, plusieurs blocs sont d'une complexité importante comme le bloc découplage ou le bloc estimateur de flux du rotor. La provenance de l'erreur à l'intérieur du bloc est souvent difficile à identifier. Il faut alors reprendre notre méthode en traitant le bloc comme un algorithme séparé. On retourne vers la référence afin d'identifier les signaux que l'on retrouve à l'intérieur du bloc problématique, et une nouvelle collecte de données est effectuée. Ensuite, de nouvelles sorties temporaires sont instanciées à l'intérieur du bloc fautif pour tenter d'identifier la source du problème.

Afin de rapidement identifier si le comportement du modèle XSG correspond au modèle théorique Simulink, les résultats des deux modèles sont comparés. Cependant, il s'agit de tests qui ne serviront pas seulement lors du développement initial, mais qui seront sollicités lors de toutes les phases de débogage. Ainsi, il est important d'établir certaines règles afin d'accéder rapidement à cet environnement lorsqu'un problème survient. Deux méthodes sont ici proposées : la méthode simultanée et la sainte croyance.

Voyons premièrement les conditions communes aux deux méthodes.

- *Le bloc XSG utilise le même fichier d'initialisation que le modèle final*

  Il est important d'utiliser le même fichier d'initialisation que le modèle XSG final afin de ne pas avoir à créer d'autres copies de ce fichier. Chaque fichier supplémentaire d'initialisation doit être modifié si un changement de constante, de précision, etc. est effectué. Lorsqu'un modèle est composé de nombreux blocs, maintenir un nombre égal de scripts d'initialisation devient une source d'erreurs trop importante.

- *Les données (entrées et sorties) sont toutes dans une même source*

  Dans les deux méthodes proposées, les entrées proviennent de l'enregistrement des signaux lors d'un test en boucle fermée du système de référence (Simulink et SPS). Comme tous ces signaux sont enregistrés au même endroit (ex. : fichier MAT), il est important de toujours utiliser cette source, même si une seule portion des signaux est utilisée. Encore une fois, la maintenance de multiples fichiers doit être évitée pour minimiser les causes d'erreur et faciliter la réutilisation de l'environnement de bloc individuel lors de débogage avancé.

- *Les entrées respectent la nomenclature des signaux du modèle de référence Simulink*

  Dans un système complexe, plusieurs signaux sont insérés dans des blocs différents afin de subir plusieurs traitements. Il convient donc d'utiliser le même nom pour un signal injecté dans plusieurs blocs différents. Dans l'environnement de bloc individuel, les entrées sont souvent des signaux provenant du *workspace* ou d'un fichier. Comme la deuxième règle spécifie que l'on doit utiliser un seul et même fichier de source, les noms des entrées vont devoir correspondre dans chaque environnement de test. Les sorties ne vont pas se plier exactement à cette règle selon la méthode choisie.

- *Les blocs sont encapsulés exactement comme dans le modèle final ou de référence*

  Il est important d'utiliser le même routage de fils afin de pouvoir supprimer et remplacer un bloc rapidement lors de modifications à vérifier.

- *L'analyse des résultats se trouve dans un fichier Matlab unique au bloc, exécutable en tout temps*

  Chaque bloc doit avoir son fichier Matlab d'analyse qui va générer les résultats (et souvent, les graphiques) permettant de valider le bloc XSG. Un seul fichier par bloc assure que l'utilisateur n'a pas besoin de faire fonctionner les autres blocs (ou de commenter des lignes du fichier Matlab) pour faire une analyse. Le fichier doit également pouvoir produire une analyse en tout temps, afin que l'usager puisse faire une pause dans sa simulation et analyser les résultats obtenus jusqu'à cette pause, sans devoir compléter un temps donné de simulation.

3.2.1.1 *La méthode simultanée*

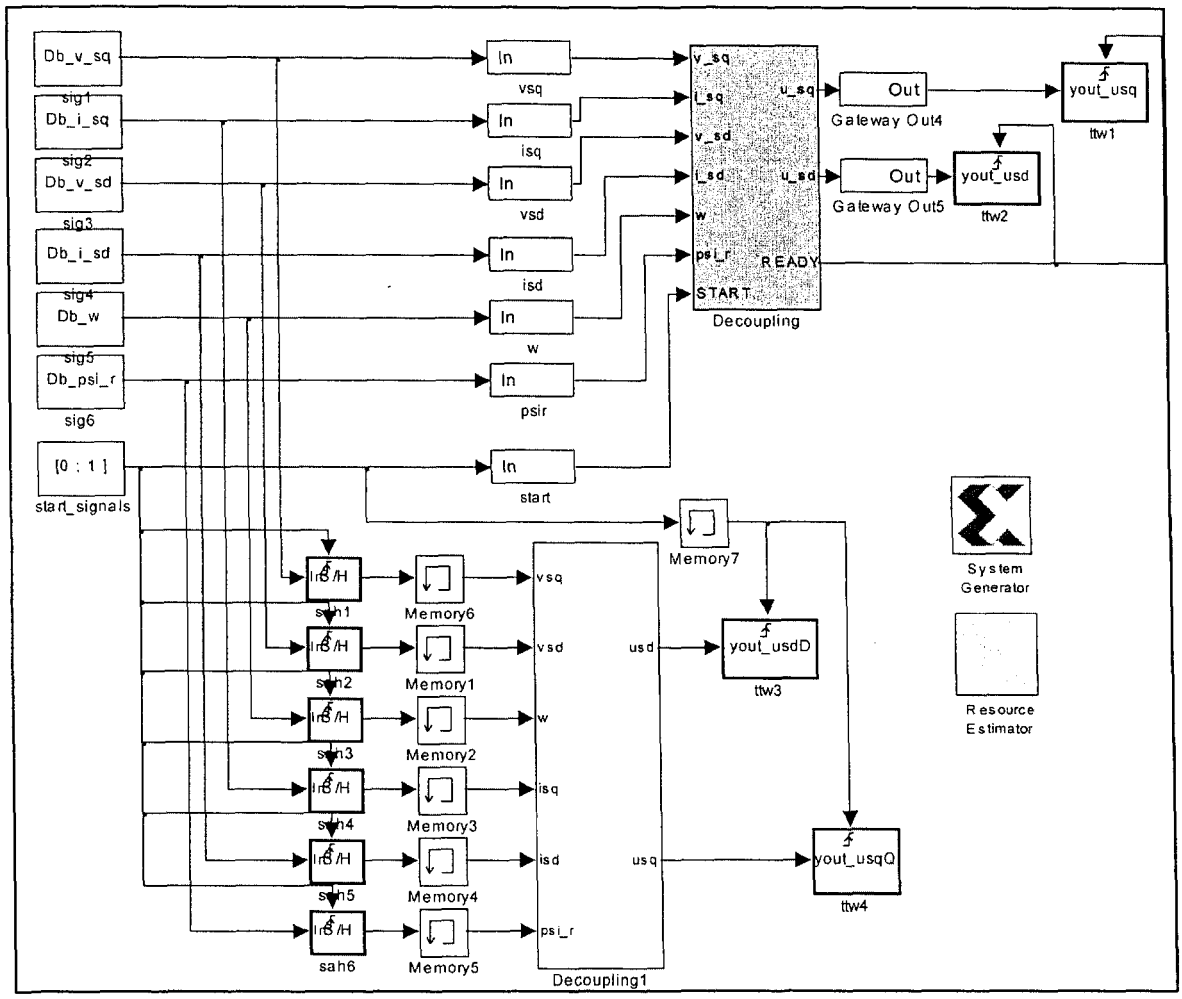

figure 9 - méthode simultanée (bloc individuel)

L'environnement contient une source d'entrée, deux blocs et deux ensembles de sorties. La source d'entrée commune aux deux blocs provient du modèle de référence Simulink. Ces signaux sont ensuite injectés dans le bloc XSG ainsi que son bloc Simulink correspondant, en parallèle. Les deux sorties (XSG et référence) sont ensuite enregistrées et comparées à l'aide du fichier d'analyse Matlab créé pour le bloc. Cette méthode est la plus efficace pour tester la correspondance des

deux blocs, car ceux-ci sont tous les deux isolés et doivent traiter les mêmes entrées. Les connexions et délais entre les blocs du système ne sont donc pas une cause d'erreur possible dans ce scénario.

Cependant, la comparaison des deux systèmes occasionne souvent des problèmes de « timing ». Le nombre de pas nécessaire pour le traitement n'est pas forcément le même pour les deux blocs, ce qui engendre l'insertion de blocs supplémentaires pour synchroniser les résultats à comparer et introduit conséquemment une nouvelle source d'erreur. Mais le plus gros désavantage de cette méthode est que, advenant un changement dans le bloc XSG, les contraintes temporelles peuvent être appelées à changer, ce qui signifie que l'usager doit modifier le mécanisme de synchronisation entre les deux blocs. Un tel ouvrage ralentit l'usager lors de tests de débogage et détériore la facilité de réutilisation de l'environnement.

3.2.1.2 La sainte croyance

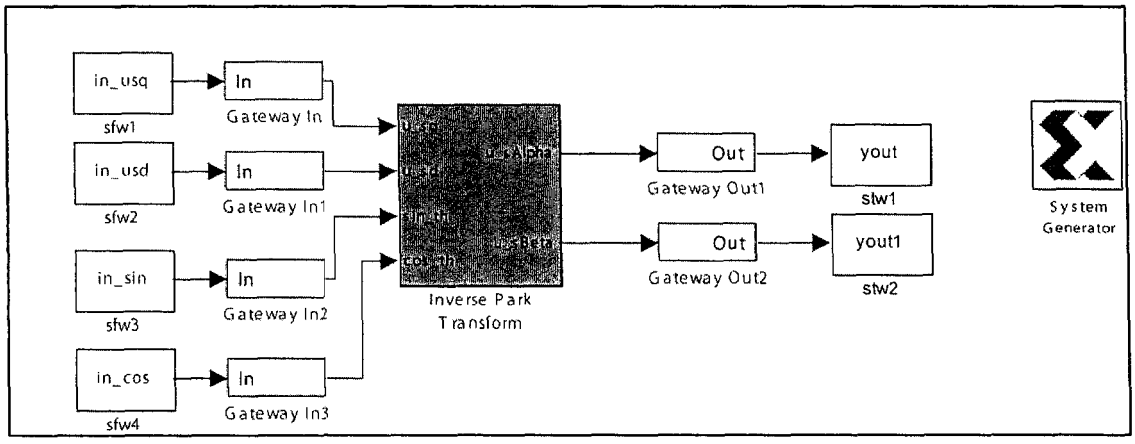

figure 10 - méthode de la sainte croyance

Cette méthode ne contient qu'un seul bloc par environnement, soit le bloc XSG. Pour des entrées qui proviennent de la simulation du modèle Simulink théorique entier, le bloc XSG traite ces données et produit des résultats qui sont ensuite comparés aux résultats enregistrés aux

sorties du même bloc dans le modèle théorique entier. On suppose ici que le bloc du modèle théorique est juste, et que les sorties théoriques utilisées sont fiables pour l'analyse.

L'avantage d'une telle méthode est la rapidité d'utilisation. Un seul bloc doit être simulé par l'ordinateur, un seul ensemble de sorties doit être enregistré, et l'environnement ne change jamais, peu importe les modifications apportées au bloc XSG. La réutilisation de cet environnement est donc très grande, et le débogage s'effectue rapidement.

Cependant, un changement des constantes et paramètres de n'importe quel bloc du système affecte les résultats du système de référence Simulink. Donc, pour chaque changement de constante, la simulation du modèle de référence en entier doit à nouveau être effectuée. Cela n'est pas toujours nécessaire en utilisant la méthode simultanée, car pour une même entrée, les deux traitements sont simulés. Pour un modèle complexe (figure 11), ce désavantage peut coûter du temps plutôt qu'en sauver. Le plus grand désavantage est au niveau de la précision d'identification, car cette méthode ne permet pas de détecter un problème qui vient des interconnexions entre les blocs. Par exemple, un usager pourrait chercher longtemps le problème de correspondance entre deux sorties, alors qu'avec la méthode simultanée, l'erreur serait la même à la sortie des deux traitements, ce qui confirme que le bloc XSG démontre le même comportement que son bloc de référence. Bien que plus rare, ce cas peut se produire et ralentir l'identification d'un problème surtout dans un système qui comporte de nombreux délais, registres et bascules d'entrée et de sortie.

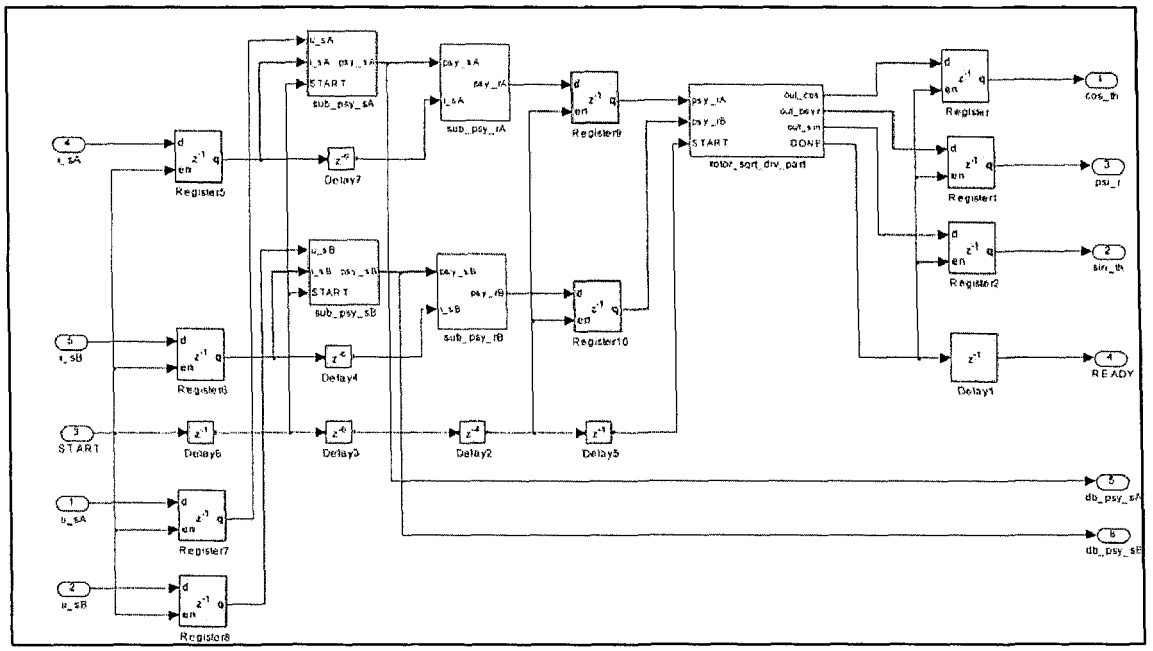

figure 11 - Estimateur de flux rotorique (bloc complexe)

3.2.2 Développement d'ensemble, liaisons et validation de deuxième niveau

Le test de chaque bloc individuel peut sembler laborieux, mais le temps investi dans cette étape se rentabilise lors du débogage du système complet. En effet, il serait laborieux de chercher le bloc fautif qui rend le système complet inopérant. Plus le test des blocs individuels est rigoureux, moins le test du système complet est laborieux. Souvent, comme dans le cas de la commande vectorielle, le défi est alors de bien synchroniser les blocs entre eux. C'est ici que des signaux « START » et « READY » peuvent être implémentés afin de s'assurer que le traitement des données se fait dans le bon ordre. Il ne s'agit plus ici d'un problème d'ordre mathématique, mais plutôt d'ordre logistique.

3.3 Contrôle vectoriel

Le contrôle vectoriel XSG est divisé selon les mêmes blocs schématisés lors de la prise de connaissance de l'algorithme de contrôle. De cette façon, il est plus facile de comparer et de tester les blocs de manière individuelle durant le développement. Le résultat d'assemblage de tous les composants est illustré à la figure 12. Des schémas internes de chaque sous-système composant le contrôle vectoriel se trouvent à l'annexe E.

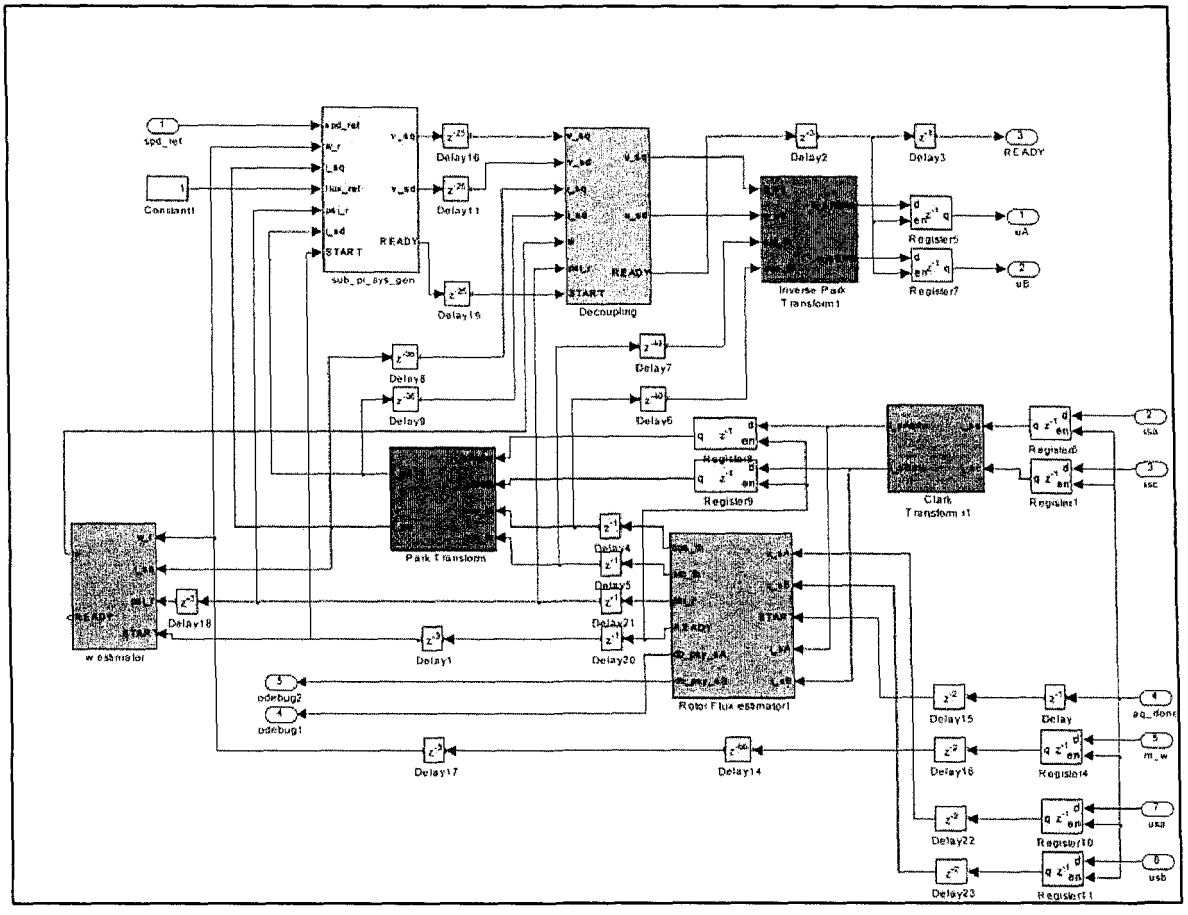

figure 12 - Contrôle vectoriel complet XSG

En incluant le calcul des délais d'activation (*firing*), le contrôle demande 122 pas d'horloge avec une précision adéquate pour donner les résultats observés à la section 4.10. Comme illustré dans la section 2.1.2, le contrôle ne s'effectue que deux fois pour une période de commutation, ce qui signifie qu'à une vitesse d'horloge de 62.5 MHz (environs 2 µs), il a amplement le temps de se compléter. Advenant le cas où il serait désirable d'effectuer des contrôles en série, il faut considérer qu'on doit attendre 70 pas d'horloge entre chaque échantillonnage. Ceci est dû au fait que l'opération de division et celle de racine carrée présentes dans le système doivent produire un résultat avant de traiter une nouvelle valeur (voir section 3.4).

La figure 13 illustre les propriétés d'un bloc d'opération XSG où la précision peut être changée par une valeur ou une variable. La section 4.2 explique l'importance d'utiliser des variables et l'outil GAPPA pour déterminer ces précisions. La précision des signaux d'entrée de tout le système (courants, tensions, vitesse) est représentée sur 16 bits, car il s'agit de la précision offerte par la carte AIO utilisée au laboratoire.

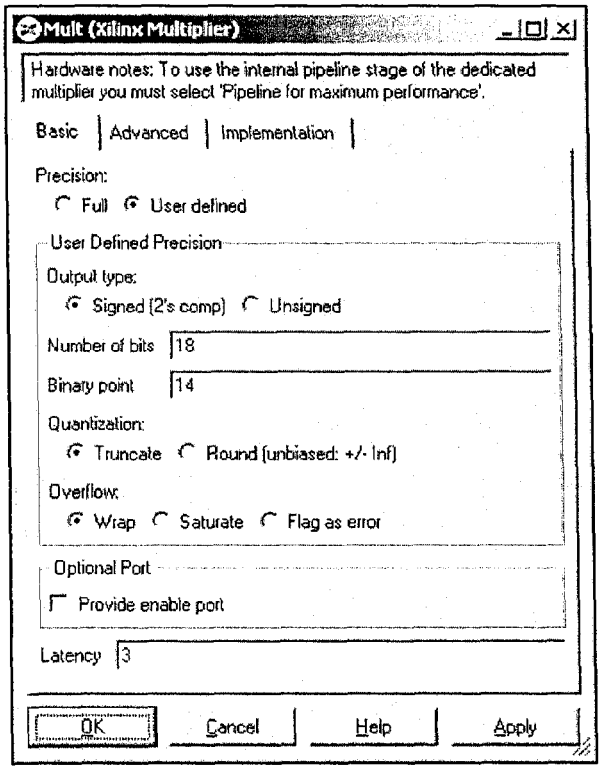

figure 13 - propriétés d'un bloc XSG

3.4 Division et Racine carrée

Il est possible d'utiliser une version non restaurante de la racine carrée et de la division afin de garder petite la taille du système et d'obtenir une meilleure précision. En effet, les opérations de division et de racine carrée fournies par XSG sont des implémentations CORDIC lourdes et peu précises. Après de nombreux tests en utilisant les versions 7, 8.1 et 8.2 de XSG (les résultats n'ont pas été vérifiés avec la nouvelle version 9), l'opération de racine carrée CORDIC de XSG ne donne qu'une grossière approximation du résultat lorsqu'une grande précision fractionnaire est demandée. De plus, la lourde implémentation de cette opération nécessite un long temps de synthèse lors de la génération d'un bloc co-simulé ou du fichier BIN pour des tests sur le terrain. Dans le cas du contrôle vectoriel, la précision offerte par l'opérateur racine carrée n'est pas suffisante, et le contrôle diverge. La division CORDIC offre pour sa part une précision très satisfaisante et permet le bon fonctionnement d'un algorithme comme le contrôle vectoriel. Cependant, sa grande taille et sa vitesse d'exécution moins grande peut la rendre moins intéressante pour une application très complexe qui nécessite déjà beaucoup de ressources du FPGA.

Les versions non restaurantes décrites dans l'annexe I de ces deux opérations offrent la précision désirée, mais elles ne peuvent pas recevoir plusieurs valeurs à traiter avant l'accomplissement de l'opération. Pour une valeur X à l'entrée de la racine carrée, il n'est pas possible de changer X pour une valeur Y tant que le résultat de la racine de X n'est pas obtenu. Heureusement, comme le contrôle vectoriel s'effectue très rapidement et s'insère confortablement entre deux échantillonnages, les versions non restaurantes peuvent être utilisées sans effet négatif. Les opérations sont complétées longtemps avant qu'un nouvel échantillonnage doive être traité. De plus, le nombre de pas nécessaire pour effectuer une racine carrée est coupé de moitié avec cette nouvelle implémentation, tandis que la division nécessite le même nombre de pas d'horloge.

Les tailles et performances des opérateurs sont comparées au tableau 2. Il s'agit de versions qui n'ont jamais été optimisées pour les FPGAs auparavant. Les schémas XSG des versions non restaurantes crées dans XSG pour cette recherche se trouvent à l'annexe F.

32 bits (14 binary point)						
	Sqrt NON CORDIC	Sqrt CORDIC	Diff.	Division NON CORDIC	Division CORDIC	Diff.
Slices	60	1377	2295%	124	1806	1456%
FFs	82	2289	2791%	96	3132	3263%
BRAMs	0	0	n.a.	0	0	n.a.
LUTs	103	2425	2354%	218	2981	1367%
IOBs	0	0	n.a.	0	0	n.a.
Emb. Mults	0	6	600%	0	3	300%
TBUFs	0	0	n.a.	0	0	n.a.
Nbr. de pas	24	51	213%	48	47	98%
Vitesse	238.112 MHz	58.666 MHz	25%	173.408 MHz	76.963 MHz	25%

tableau 2 - Tailles des opérateurs division et racine carrée

3.5 Portillonnage (délai d'activation et temps de maintien)

Les blocs de portillonnage précédemment implémentés à l'aide du bloc de fonction embarquée (voir section 2.2.2) dans Simulink sont maintenant créés avec XSG (figure 14 et figure 15).

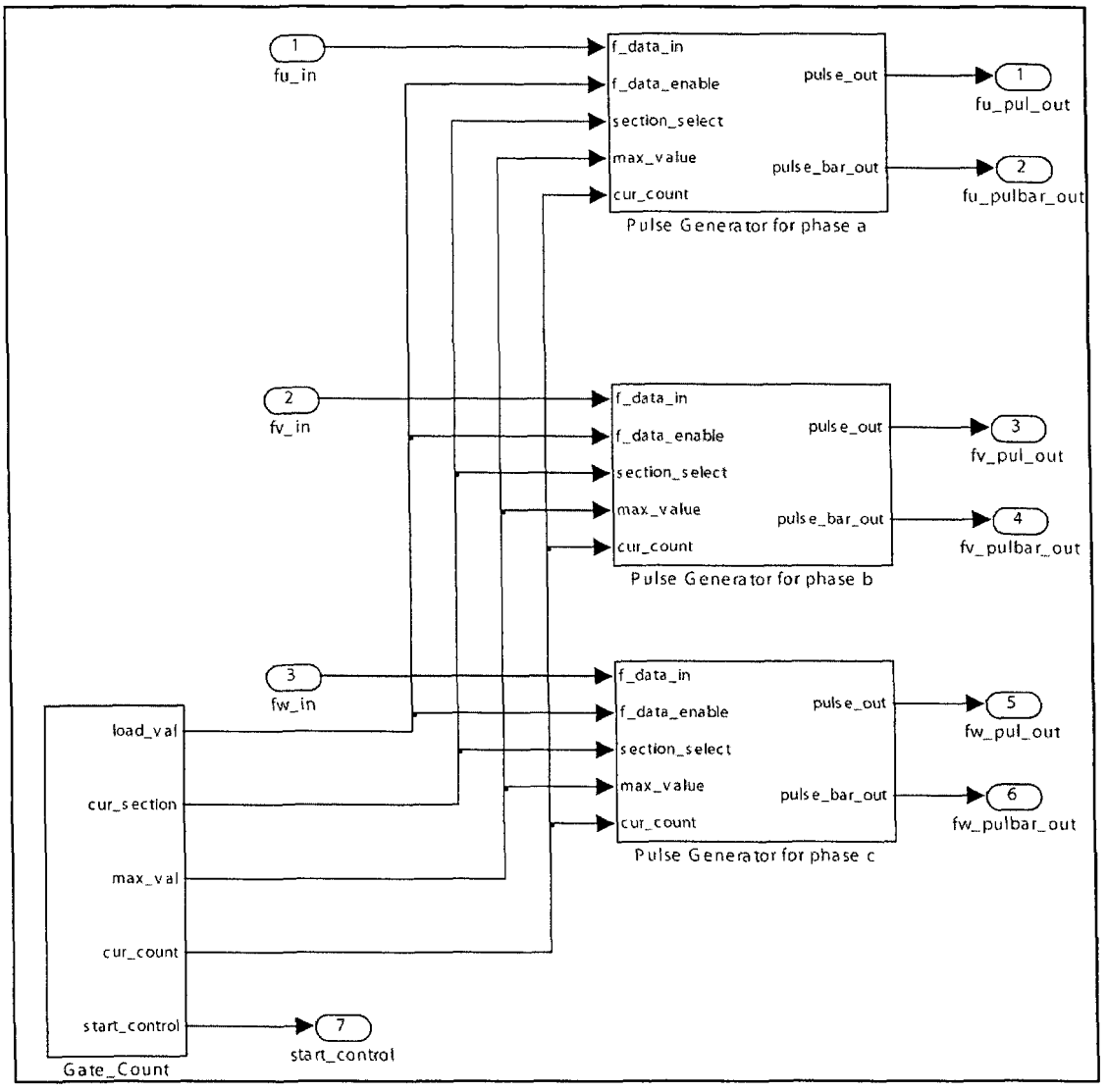

figure 14 - Portillonnage

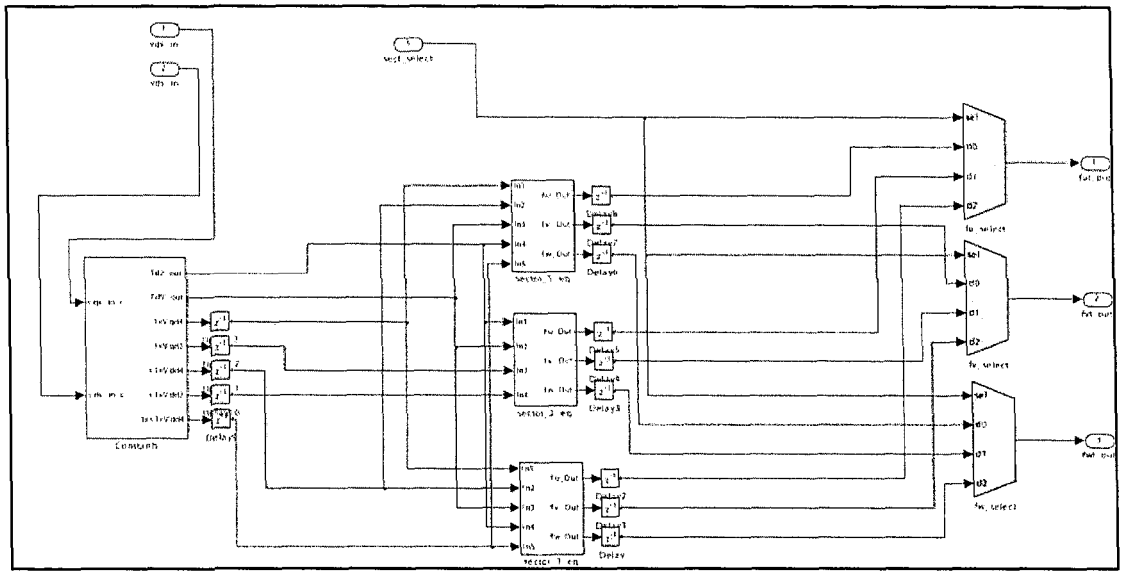

figure 15 - Calcul des délais d'activation

Le calcul du temps mort (figure 16) qui s'effectue à l'intérieur du bloc de portillonnage est basé sur un algorithme développé par OPAL-RT dans [9] et adapté à la situation de la recherche. Le fichier d'initialisation Matlab dicte le temps mort désiré avec la constante `dead_time`.

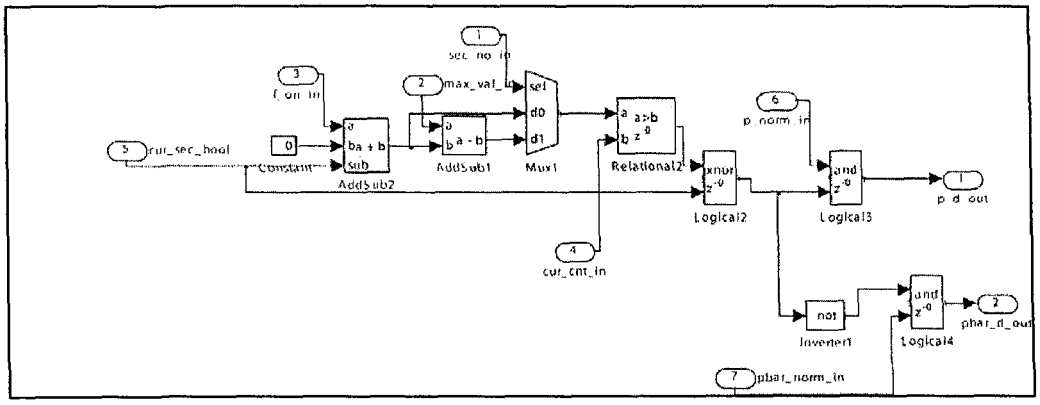

figure 16 - Calcul du temps mort

Afin de tester le bon fonctionnement du portillonnage avant son intégration au sein du projet entier, celui-ci est inséré dans un tutoriel fournit par SPS. Il remplace alors le portillonnage de base

s'y trouvant, et le débogage peut s'effectuer jusqu'à l'obtention d'une précision adéquate. Les résultats sont observables à la figure 17.

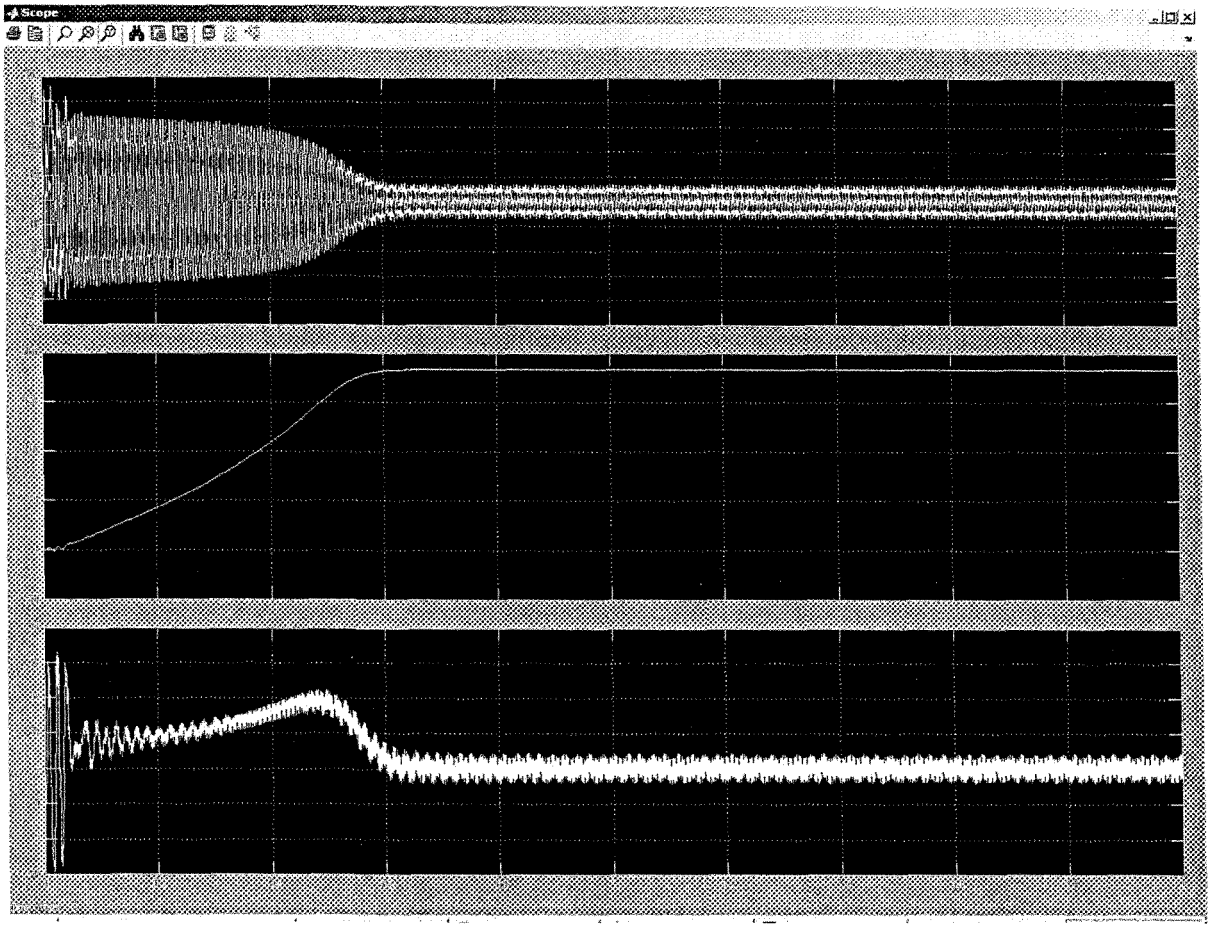

figure 17 - Courants, vitesse et torque du moteur avec simulation du portillonnage XSG

Le profil suivant donne les résultats de la figure 19.

```
freq=10;
Ua=0;Ub=0;
for k=Ts:Ts:3

    Wref=2*pi*(freq/p);


    if k > (0.3)
         freq=30;
    end
```

```
if k > (0.8)
     freq=10;
end

    if k > (1.7)
     freq=50;
end

if k > (2.5)
     freq=5;
end
```

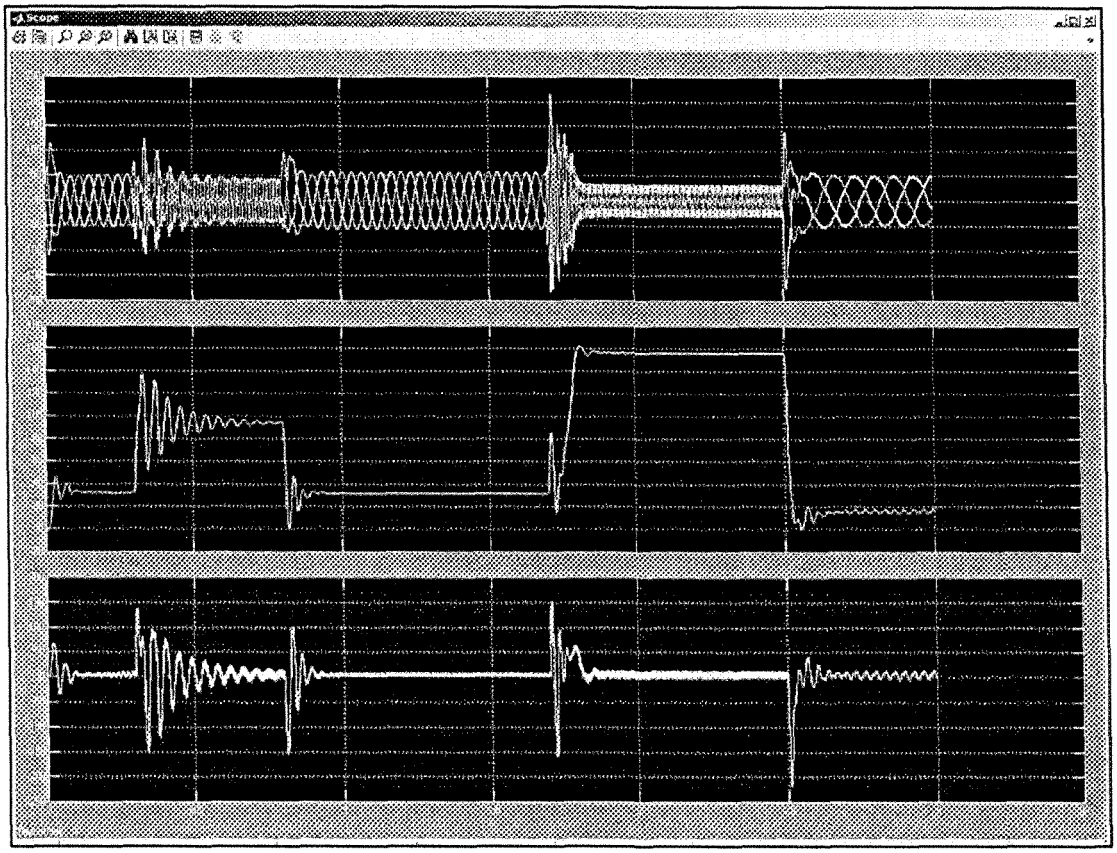

figure 18 - Courants, vitesse et torque du moteur avec simulation du portillonnage XSG

3.6 Lecture de vitesse provenant d'un encodeur optique

L'implémentation de la lecture de vitesse simple se trouve à la figure 19. Il s'agit d'un filtre simple qui calcule la vitesse moyenne pour une plage de temps spécifiée. Une implémentation beaucoup plus efficace décrite dans [12] sera éventuellement développée et testée avant d'effectuer des tests plus poussés en laboratoire. Pour des tests préliminaires et la vérification des connexions, cette méthode est suffisante.

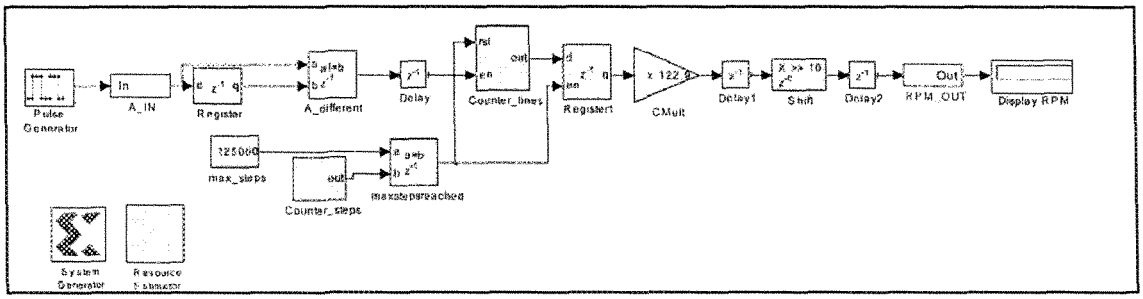

figure 19 - Lecture de vitesse d'un encodeur optique (modèle simple)