

Traitement du signal des capteurs à transformateur variable pour le contrôleur du Beam Wire-Scanner du CERN

Travail de Bachelor

Non Confidentiel

Département : TIC

Filière : Informatique et systèmes de communication

Orientation : Systèmes informatiques embarqués

Maillard Patrick

23 mai 2025

Travail proposé par :

Jonathan Emery

CERN

Espl. des Particules 1, 1217 Genève

Supervisé par :

Prof. Yann Thoma (HEIG-VD)

Préambule

Ce travail de Bachelor (ci-après TB) est réalisé en fin de cursus d'études, en vue de l'obtention du titre de Bachelor of Science HES-SO en Ingénierie.

En tant que travail académique, son contenu, sans préjuger de sa valeur, n'engage ni la responsabilité de l'auteur, ni celles du jury du travail de Bachelor et de l'Ecole.

Toute utilisation, même partielle, de ce TB doit être faite dans le respect du droit d'auteur.

HEIG-VD

Le Chef du Département

Yverdon-les-Bains, le 23 mai 2025

Authentification

Le soussigné, Maillard Patrick, atteste par la présente avoir réalisé seul ce travail et n'avoir utilisé aucune autre source que celles expressément mentionnées.

Maillard Patrick

A handwritten signature in black ink, appearing to read "Maillard Patrick".

Yverdon-les-Bains, le 23 mai 2025

Remerciements

Je tiens à remercier mon professeur responsable, M. Yann Thoma, ainsi que M. Jonathan Emery pour le suivi et le soutien apporté durant mon projet. Je tiens à remercier également M. Raoul Herzog et Yves Chevallier pour leurs compléments d'information.

Résumé

Le groupe Instrumentation du Faisceau (Beam Instrumentation - BI) est responsable de la recherche, du développement, de la mise en service et de la maintenance des instruments de mesure des faisceaux de particules dans les accélérateurs de l'Organisation Européenne pour la Recherche Nucléaire (CERN).

Le *Wire-Scanner* (balayeur à fil) est un instrument permettant de mesurer la taille transverse des faisceaux de particules. Cet appareil mécatronique comporte un fil très fin se déplaçant à grande vitesse à travers le faisceau, sous ultra-vide. L'interaction entre le fil et le faisceau produit des particules secondaires, détectées par un scintillateur externe.

L'instrument *wire-scanner* du CERN utilise des capteurs à transformateur variable pour la mesure de déplacement rotatif, et prévoit d'utiliser des dispositifs similaires pour les déplacements linéaires. La version rotative est appelée *resolver* et permet de mesurer l'angle de l'axe moteur à l'intérieur de l'enceinte sous vide des accélérateurs. Cette configuration impose un fonctionnement du capteur à travers une membrane métallique et sur de très longs câbles, ce qui perturbe les signaux de sortie et rend difficile l'utilisation de solutions standard. Ce système bénéficierait grandement d'une solution de traitement sur mesure, intégrée de manière flexible à une FPGA. Le capteur linéaire étudié pour le nouveau dispositif est un transformateur différentiel linéaire (LVDT), qui génère des signaux similaires à ceux du *resolver*, à quelques différences près.

L'objectif de ce projet est de concevoir et de valider un IP de traitement du signal robuste dans une FPGA, afin de remplacer le circuit RDC externe. Ce core devra générer le signal d'excitation pour la bobine primaire, analyser les deux signaux de sortie (appelés *sin* et *cos* pour le *resolver*), numérisés dans le FPGA cible, puis calculer l'angle ou la position actuelle ainsi que différents états (capteur connecté, qualité du signal, etc.), en fonction de sa configuration. Plusieurs méthodes de démodulation existent et sont étudiées en début de projet afin de sélectionner la plus adaptée.

Ce projet répond à cette problématique en proposant une solution permettant de démoduler les signaux reçus du *resolver*. Plusieurs techniques de démodulation ont été comparées et c'est la démodulation en quadrature (I/Q) qui a été retenue pour son compromis entre précision, latence et simplicité, permettant d'utiliser la même chaîne de traitement pour le *resolver* (RVDT) et le futur LVDT, et donc de simplifier la maintenance.

Les algorithmes ont été implémentés en VHDL, complétés par un ensemble complet de bancs de test qui valident chaque bloc, puis le système global (démodulation, filtrage CIC, calcul d'angle, détection d'erreurs).

Le prototype est fonctionnel, respecte les exigences de résolution du RDC et s'intègre aisément dans l'architecture globale du Wire-Scanner. Il constitue une bonne base pour l'extension au mode LVDT et la future instrumentation du BWS.

Table des matières

Préambule	I
Authentification.....	III
Remerciements.....	V
Résumé	VII
Chapitre 1 Introduction	1
1.1 Contexte	1
1.2 Objectif.....	2
1.3 Travail effectué.....	2
1.4 Structure du rapport	3
Chapitre 2 Etat de l'art	4
2.1 Introduction	4
2.2 Transformée de Hilbert.....	5
2.3 Démodulation par quadrature	5
2.4 Démodulation par PLL.....	6
2.5 Démodulation par Costas Loop.....	8
Chapitre 3 Analyse comparative des algorithmes.....	9
3.1 Comparaison expérimentale des algorithmes	9
3.1.1 Paramètre réel	9
3.1.2 Démodulation par Transformée de Hilbert	11
3.1.3 Démodulation en Quadrature (I/Q)	12
3.1.4 Démodulation par PLL.....	13
3.1.5 Démodulation par Costas Loop.....	14
3.1.6 Analyse des résultats	15
3.2 Choix de l'algorithme	16
Chapitre 4 Quadrature I/Q, filtre CIC et contraintes	17
4.1 Fonctionnement Quadrature	17
4.1.1 Modulation I/Q :	17
4.1.2 Démodulation I/Q :	20
4.1.3 Signaux reçus du resolver	22
4.2 Fonctionnement CIC.....	24

4.3	Contrainte resolver (correction d'ellipcicité/ Non-idéalités)	26
4.4	Algorithmes existants pour une implémentation sur FPGA.....	27
Chapitre 5 Conception		28
5.1	Système globale	28
5.2	Démodulation par Quadrature	29
5.2.1	Filtre CIC.....	29
5.3	Détection d'erreur.....	30
5.4	Générateur de sinus	33
Chapitre 6 Implémentation		34
6.1	Filtre CIC	34
6.1.1	Formule de Hogenauer	35
6.2	Démodulation Quadrature.....	36
6.3	Calcul d'amplitude.....	39
6.4	Détection d'erreur.....	40
6.5	Système complet.....	42
6.6	Problème rencontré	44
6.7	Synthèse summary	46
Chapitre 7 Bancs de test		47
7.1	Filtre CIC	47
Lancement des simulations	47	
Synchronisation et métrique d'erreur	48	
Résultats – signal sans bruit	48	
Résultats – signal bruité.....	49	
Conclusion	49	
7.2	Démodulation en quadrature	49
Lancement des simulations	50	
Résultats – signal sans bruit	50	
Résultats – signal bruité.....	51	
Conclusion	52	
7.3	Système complet (Sans détection d'erreur)	52
Lancement des simulations	52	
Synchronisation	52	
Résultats – signal sans bruit	53	
Résultats – signal bruité.....	54	
Conclusion	55	
7.4	Détection d'erreur.....	55
Lancement des simulations	56	
Principe du test.....	56	
Résultats	56	
Conclusion	58	
Chapitre 8 Améliorations futures		59

Chapitre 9 Conclusion.....	60
Bibliographie.....	62
Annexes	64
A. Planification	64
B. Configuration Banc de tests	65
C. Illustration du fonctionnement du design à divers angles	68

Table des figures

Figure 1.1 – Schéma de fonctionnement du Beam Wire-Scanner [2].....	1
Figure 2.1 – Schéma de l'unité de traitement RVDT/LVDT sur FPGA [3]	4
Figure 2.2 - Structure interne d'un RVDT (gauche) et LVDT (droite) [3]	4
Figure 2.3 – Schéma d'un démodulateur en quadrature (I/Q) [8]	6
Figure 2.4 – Bloc fonctionnel d'une PLL (Phase-Locked Loop) [9].....	7
Figure 2.5 – Architecture de démodulation basée sur une PLL [10]	7
Figure 2.6 – Démodulateur en quadrature avec principe PLL (Costas Loop) [12]	8
Figure 3.1 – Simulation d'un signal RVDT : angle du rotor (gauche) et signaux aux secondaires (droite).....	9
Figure 3.2 – Analyse du bruit relatif sur les signaux sin et cos.....	10
Figure 3.3 – Démodulation des signaux secondaires par transformée de Hilbert (sans bruit).....	11
Figure 3.4 - Démodulation des signaux secondaires par transformée de Hilbert (avec bruit)	11
Figure 3.5 - Démodulation des signaux secondaires par Quadrature (sans bruit).....	12
Figure 3.6 - Démodulation des signaux secondaires par Quadrature (avec bruit)	12
Figure 3.7 - Démodulation des signaux secondaires par PLL (sans bruit)	13
Figure 3.8 - Démodulation des signaux secondaires par PLL (avec bruit).....	13
Figure 3.9 - Démodulation des signaux secondaires par Costas Loop (sans bruit)	14
Figure 3.10 - Démodulation des signaux secondaires par Costas Loop (avec bruit).....	14
Figure 4.1 – Comparaison entre un signal AM et un signal non modulé	17
Figure 4.2 – Représentation temporelle de deux signaux en quadrature (I/Q).....	18
Figure 4.3 - Schéma d'un modulateur en quadrature (I/Q) [8].....	18
Figure 4.4 - Signal résultant de la somme des composantes I et Q	19
Figure 4.5 – Influence relative des composantes I et Q sur la forme du signal modulé	19
Figure 4.6 – Représentation des composantes I/Q et de leur combinaison vectorielle [8].....	20
Figure 4.7 – Structure électromagnétique du resolver "brushless" [16]	22
Figure 4.8 – Sorties sinus/cosinus et enveloppe [16].....	23
Figure 4.9 – Signaux sinus et cosinus capturés en sortie du resolver	23
Figure 4.10 – Architecture d'un filtre CIC de décimation [17]	24
Figure 4.11 - Architecture complète d'un filtre CIC de décimation à 3 étages [18].....	25
Figure 4.12 – Illustration du fonctionnement d'un filtre CIC de décimation	26
Figure 4.13 – Illustration d'une ellipse [22].....	27
Figure 5.1 - Schéma du système complet.....	28
Figure 5.2 – Schéma du bloc Quadrature Demodulation.....	29
Figure 5.3 – Schéma du filtre CIC	30
Figure 5.4 – Fault register du RDC présent dans la datasheet du AD2S1210 [27]	30
Figure 5.5 – Schéma du générateur de sinus	33
Figure 6.1 – IP parameter Editor du CORDIC pour la fonction vector translate	38
Figure 6.2 - IP parameter Editor du NCO.....	39
Figure 6.3 – Tableau représentant la taille des fenêtres d'échantillon en fonction de la fréquence d'excitation [27]	42
Figure 6.4 - Réponse en amplitude d'un filtre CIC avec N = 9, R = 8 et M = 1 [31]	45
Figure 6.5 - Réponse du filtre de compensation pour un filtre CIC à 4 étages, tracée sur fS/R [31]	45

Figure 7.1 - Waveform du banc de test cic_tb.sv sans bruit	48
Figure 7.2 - Waveform du banc de test cic_tb.sv avec bruit.....	49
Figure 7.3 - Waveform du banc de test quadrature_demod_tb.sv sans bruit.....	50
Figure 7.4 - Waveform du banc de test quadrature_demod_tb.sv avec bruit	51
Figure 7.5 - Waveform du banc de test processing_tb.sv sans bruit	53
Figure 7.6 - Waveform du banc de test processing_tb.sv avec bruit	54
Figure 7.7 – Exemple vérification du moniteur	55
Figure 7.8 – Waveform montrant le test d7_clipping plus en détail	57
Figure 7.9 – Waveform du banc de test error_tb.sv en entière.....	57
Figure A.1 - Planification	64
Figure C.1 – Démonstration Angle = 20°, sans bruit.....	68
Figure C.2 - Démonstration Angle = 20°, avec bruit	69
Figure C.3 - Démonstration Angle = 100°, sans bruit	70
Figure C.4 - Démonstration Angle = 100°, avec bruit	71
Figure C.5 - Démonstration Angle = 359°, sans bruit	72
Figure C.6 - Démonstration Angle = 359°, avec bruit	73

Liste des tableaux

Tableau 3.1 - Comparaison des principales méthodes de démodulation pour RVDT/LVDT	15
Tableau 4.1 – Résultat de la démodulation par quadrature selon le signal d'entrée.....	22
Tableau 5.1 – Plan d'adressage du bloc de détection d'erreur.....	32
Tableau 5.2 – Définition des bits du signal d'erreur de sortie	32
Tableau 6.1 - Paramètres générique du filtre CIC	34
Tableau 6.2 - Paramètre d'entrée(i)/sortie(o) du filtre CIC	35
Tableau 6.3 – Paramètres générique du bloc de démodulation par quadrature	37
Tableau 6.4 – Paramètre d'entrée(i)/sortie(o) du bloc de démodulation par quadrature.....	37
Tableau 6.5 - Paramètres générique de la détection d'erreur.....	40
Tableau 6.6 - Paramètre d'entrée(i)/sortie(o) de notre système complet	41
Tableau 6.7 - Paramètres générique de notre système complet.....	43
Tableau 6.8 - Paramètre d'entrée(i)/sortie(o) de notre système complet	44
Tableau 6.9 - Ressources utilisées pour notre système après la synthèse	46

Liste des codes sources

Listing 6.1 – Déclaration de la constante B_{\max}	36
Listing 6.2 – Déclaration de la constante définie de l'IP CORDIC.....	40
Listing 6.3 – Calcul du monitor en VHDL	41
Listing 6.4 – Formule calculant le nombre de cycles d'horloge interne.....	42
Listing 7.1 – Commande pour lancer le banc de test cic_tb.sv	47
Listing 7.2 – Rapport QuestaSim pour un signal propre ($R = 10, M = 1, N = 3$).....	48
Listing 7.3 - Rapport QuestaSim pour un signal bruité ($R = 10, M = 1, N = 3$).....	49
Listing 7.4 - Commande pour lancer le banc de test quadrature_demod_tb.sv	50
Listing 7.5 - Rapport QuestaSim pour un signal propre ($R = 2000, M = 2, N = 3$)	51
Listing 7.6 - Rapport QuestaSim pour un signal bruité ($R = 2000, M = 2, N = 3$).....	52
Listing 7.7 - Commande pour lancer le banc de test processing_tb.sv.....	52
Listing 7.8 - Rapport QuestaSim pour un signal propre ($R = 1800, M = 2, N = 3$)	54
Listing 7.9 - Rapport QuestaSim pour un signal bruité ($R = 1800, M = 2, N = 3$).....	55
Listing 7.10 - Commande pour lancer le banc de test error_tb.sv.....	56
Listing 7.11 - Rapport QuestaSim pour notre détection d'erreur.....	58
Listing B.1 – Commande pour générer un script de simulation.....	65
Listing B.2 – Exemple de script pour Questasim/ModelSim	67
Listing C.1 – Rapport avec Angle = 20°, sans bruit	68
Listing C.2 - Rapport avec Angle = 20°, avec bruit.....	69
Listing C.3 - Rapport avec Angle = 100°, sans bruit.....	70
Listing C.4 - Rapport avec Angle = 100°, avec bruit.....	71
Listing C.5 - Rapport avec Angle = 359°, sans bruit.....	72
Listing C.6 - Rapport avec Angle = 359°, avec bruit.....	73

Chapitre 1

Introduction

1.1 Contexte

Ce travail a été réalisé pour l'Organisation Européenne pour la Recherche Nucléaire (CERN), plus particulièrement avec le groupe Instrumentation du Faisceau (Beam Instrumentation - BI) qui est responsable de la recherche, du développement, de la mise en service et de la maintenance des instruments de mesure des faisceaux de particules dans les accélérateurs.

Ce projet porte principalement sur le Beam Wire-Scanner (BWS), un instrument permettant de mesurer la taille transverse des faisceaux de particules. Cet appareil mécatronique comporte un fil très fin se déplaçant à grande vitesse à travers le faisceau, sous ultravide. L'interaction entre le fil et le faisceau produit des particules secondaires, détectées par un scintillateur externe. Ce fil est situé au bout d'un bras rotatif déplacé à l'aide d'un moteur brushless. Afin de positionner le fil avec précision et de garantir des mesures fiables, l'instrument s'appuie sur des capteurs de position rotatifs et linéaires, respectivement appelés RVDT et LVDT, qui nécessitent un traitement de signal performant [1]

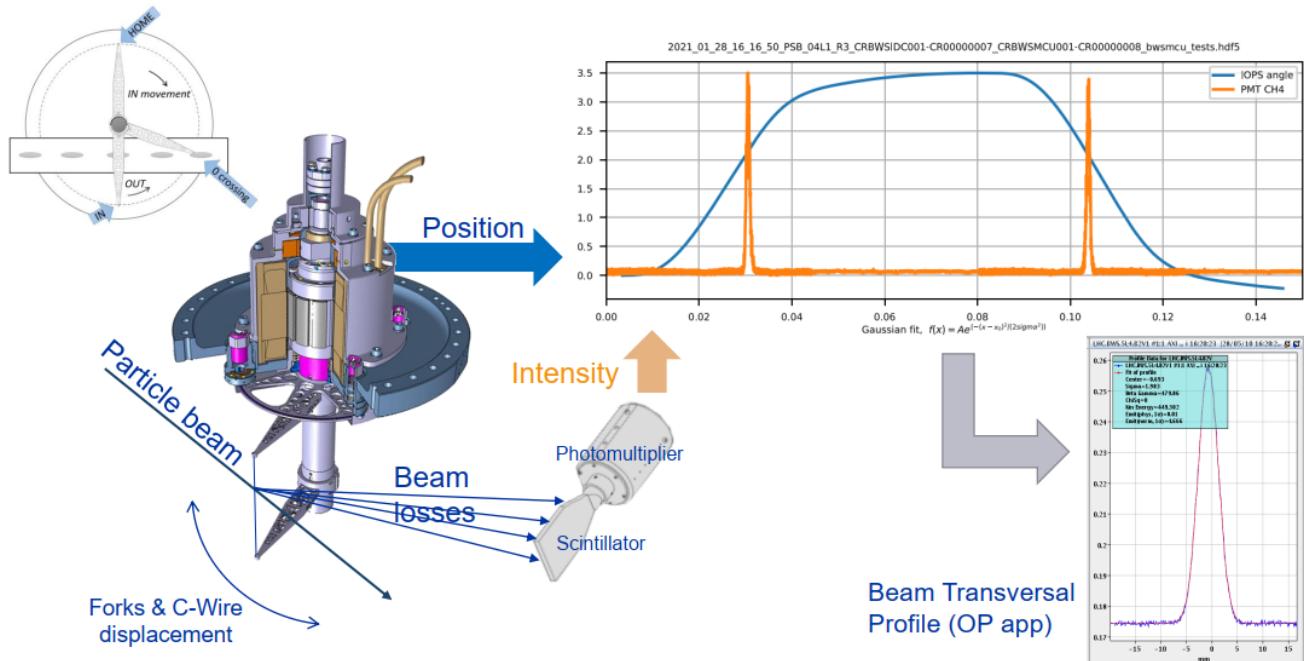


Figure 1.1 – Schéma de fonctionnement du Beam Wire-Scanner [2]

Le groupe Instrumentation du Faisceau traite ces signaux à l'aide d'un circuit intégré appelé un RDC (Resolver-to-Digital Converter), mais cette solution atteint ses limites en termes de flexibilité, de diagnostic et de robustesse. Le présent projet vise donc à rapatrier la conversion et le conditionnement de ces signaux dans un FPGA, afin de bénéficier d'un système plus évolutif et plus fiable. Cette approche se justifie d'autant plus que, dans l'architecture actuelle, les données délivrées par le RDC sont aussitôt transférées vers un FPGA pour la suite du traitement. Regrouper l'ensemble de la chaîne sur une seule plateforme simplifie donc l'électronique et réduit les sources d'erreurs.

Dans ce contexte, où la précision et la robustesse des mesures issues de capteurs rotatifs et linéaires sont cruciales, les signaux sensibles au bruit et aux déphasages induits par de longues liaisons de câbles requièrent un traitement embarqué performant. L'utilisation d'un FPGA répond donc à ces contraintes en offrant une latence minimale, une flexibilité de mise à jour et un diagnostic en temps réel, avantages inaccessibles avec les convertisseurs rigides et fermés actuellement employés.

1.2 Objectif

Ce projet vise à concevoir et valider un core IP de traitement du signal robuste, intégré dans un FPGA, afin de remplacer le circuit intégré externe de type RDC (Resolver-to-Digital Converter). Ce core devra générer le signal d'excitation destiné à la bobine primaire, analyser les deux signaux de sortie (appelés sin et cos dans le cas d'un resolver), numérisés au sein de la FPGA cible, puis calculer l'angle ou la position actuelle ainsi que divers états (connexion du capteur, qualité du signal, etc.), en fonction de la configuration du core.

Plusieurs méthodes de démodulation existent et doivent être analysées afin d'identifier celle qui convient le mieux à ce projet. À l'issue de cette étude, une description VHDL du système sera élaborée et des bancs de test seront créés pour en valider le fonctionnement.

1.3 Travail effectué

Depuis le début du projet, plusieurs étapes ont été effectuées. Tout d'abord, un état de l'art a été réalisé. Puis, une fois notre méthodologie définie, nous avons conçu et implémenté le système de traitement des signaux issus des capteurs RVDT.

Le travail se concentre sur les points suivants :

- Compréhension du système existant utilisé par le CERN, avec une analyse du fonctionnement des capteurs RVDT, du système de traitement actuel basé sur un RDC et des contraintes observées.
- Analyse des signaux réels. Extraction et traitement des données issues de l'ADC du resolver afin d'identifier les contraintes, telles que la présence de bruit, les déphasages, et les effets de transmission sur 220 mètres de câble.
- Étude et simulation de différentes techniques de démodulation (Hilbert, Quadrature, PLL, Costas Loop), en évaluant leur précision, leur robustesse face au bruit et la complexité d'implémentation sur FPGA.
- Choix porté sur la méthode par quadrature (I/Q), basée sur les résultats des simulations et les exigences du projet (complexité, utilité, etc...)
- Étude des solutions de filtrage passe-bas, avec la sélection d'un filtre CIC, mieux adapté à une implémentation FPGA.
- Conception détaillée des blocs nécessaires (NCO, mélangeurs, CIC, CORDIC, logique de diagnostic, etc.).
- Implémentation VHDL des schémas conçus.
- Développement de bancs de test automatisés pour chaque bloc et pour le système complet.
- Validation fonctionnelle des descriptions VHDL à l'aide de ces bancs de test.

1.4 Structure du rapport

Chapitre 1 – Introduction : présente le contexte du projet (1.1), les objectifs, le travail effectué ainsi que la structure du document.

Chapitre 2 - État de l'art : décrit le fonctionnement de notre système de traitement RVDT/LVDT et les principales méthodes de démodulation d'amplitude (Hilbert, Quadrature, PLL, Costas Loop)

Chapitre 3 – Analyse comparative des algorithmes : présente les simulations effectuées pour comparer les différentes méthodes de démodulation sur des signaux bruités et non bruités, en mettant en évidence les avantages et inconvénients de chaque technique.

Chapitre 4 – Quadrature I/Q, filtre CIC et contraintes : détaille la solution choisie, ainsi que son fonctionnement mathématique, les contraintes liées au résolveur, et les blocs IP qui seront utilisés pour l'implémentation sur FPGA.

Chapitre 5 – Conception : Présente les schémas fonctionnels établis en vue de l'implémentation VHDL.

Chapitre 6 – Implémentation : Décrit les modules VHDL réalisés et justifie les choix effectués lors du développement.

Chapitre 7 – Bancs de test : Explique la structure des bancs de test, leur utilisation et les résultats obtenus.

Chapitre 8 - Améliorations futures : Propose des pistes pour renforcer les performances, la robustesse et l'extensibilité du système (intégration LVDT, optimisation mémoire, etc...).

Chapitre 9 – Conclusion : Synthétise le travail accompli, rappelle le choix technologique final et ouvre sur les perspectives du projet.

Chapitre 2

Etat de l'art

2.1 Introduction

Avant de détailler les différentes techniques de démodulation utilisées pour le traitement des signaux issus des capteurs RVDT/LVDT, il est important de comprendre ce que l'on reçoit de ces derniers, ainsi que les contraintes imposées par l'environnement du CERN.

Le système étudié ici comprend donc 2 capteurs :

- Le resolver (RVDT – Rotary Variable Differential Transformer)
- Le LVDT (Linear Variable Differential Transformer)

Ces 2 capteurs fonctionnent selon des principes similaires, reposant sur l'induction électromagnétique, mais sont différents au niveau de la grandeur mesurée (angle pour un RVDT, position linéaire pour le LVDT)

Le schéma ci-dessous présente une vue d'ensemble de l'architecture du système, illustrant les signaux d'entrée générés par ces capteurs et les sorties attendues après traitement dans le FPGA.

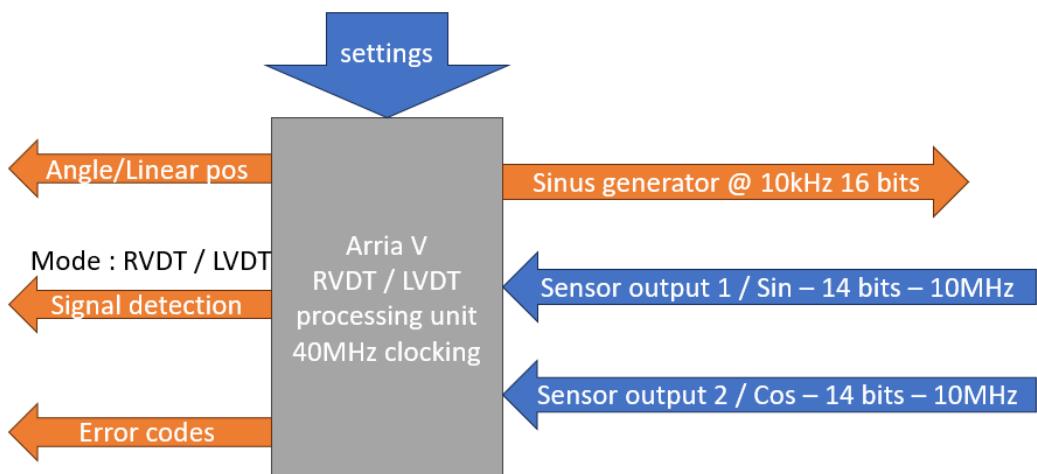


Figure 2.1 – Schéma de l'unité de traitement RVDT/LVDT sur FPGA [3]

Dans les 2 cas, le capteur est excité par un signal sinusoïdal. Les bobines secondaires génèrent 2 signaux modulés en amplitude (AM), dépendant de la position mécanique du moteur. Voici ci-dessous le schéma de chacun des capteurs.

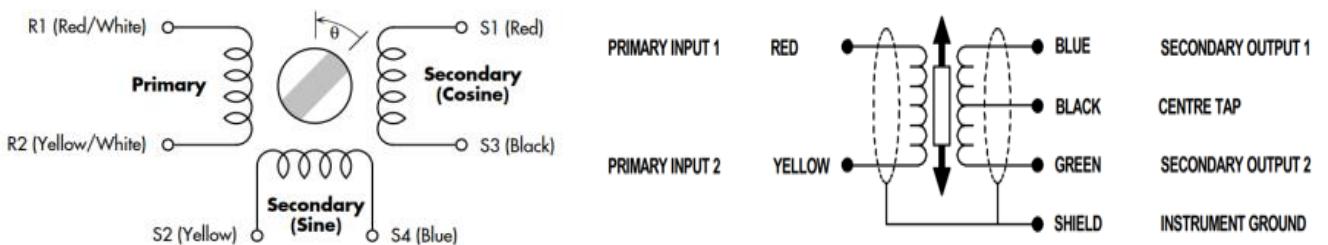


Figure 2.2 - Structure interne d'un RVDT (gauche) et LVDT (droite) [3]

Ces signaux sont ensuite numérisés par une ADC (Analog to Digital Converter) et envoyés vers le FPGA pour traitement.

Ce chapitre a pour objectif d'étudier les différentes techniques de démodulation d'amplitude existantes pour des systèmes FPGA, basées sur le document [4]. La démodulation d'amplitude est l'une des étapes les plus importantes dans le traitement des signaux provenant des capteurs RVDT et LVDT, car elle permet d'extraire les informations nécessaires aux calculs de l'angle ou de la position.

Nous passerons donc à travers différentes techniques de démodulation synchrone, telle que la transformée de Hilbert, la démodulation par PLL, la démodulation par quadrature et la démodulation par Costas Loop. L'analyse se portera principalement sur leur précision et leur robustesse au bruit.

2.2 Transformée de Hilbert

La transformée de Hilbert est un algorithme utilisé en mathématiques et en traitement de signal. Cette technique permet de construire un signal analytique complexe à partir d'un signal réel, c'est-à-dire un signal dont la partie réelle correspond au signal initial et la partie imaginaire est une version du même signal, mais décalée de ± 90 degrés. Ce décalage est obtenu à l'aide d'un filtrage compris dans la transformée de Hilbert.

Dans le traitement de signal, la partie complexe permet d'extraire 2 informations importantes :

- **L'amplitude instantanée** ou aussi appelé l'enveloppe, qui décrit l'intensité du signal à chaque instant.
- **La phase instantanée**, qui donne la position du signal dans son cycle à un moment donné. (Ne nous intéresse pas dans le contexte de notre projet)

Pour obtenir l'enveloppe du signal on vient calculer le module du signal complexe. Le module d'un signal complexe est la représentation de son amplitude instantanée et est exprimé par la racine carrée de la somme des carrés de la partie réelle et de la partie imaginaire.

Soit :

$$\text{Enveloppe} = \sqrt{\text{signal réel}^2 + \text{signal imaginaire}^2}$$

En résumé, la transformée de Hilbert s'appuie sur la transformation de Fourier, pour créer un signal réel et un signal complexe (déphasé de ± 90 degrés) de notre signal d'origine. Cette opération permet de calculer l'enveloppe de notre signal. Cette partie s'appuie notamment sur les documents [5], [6] et [7].

2.3 Démodulation par quadrature

La démodulation par quadrature est une méthode très souvent utilisée dans le traitement de signal pour récupérer l'information d'amplitude et de phase d'un signal modulé. Cette technique utilise le principe de multiplication du signal d'origine par 2 signaux sinusoïdaux, aussi appelé porteuses. Ces 2 signaux sinusoïdaux ont la même fréquence que le signal initial, mais l'un des 2 est déphasé de 90 degrés, on dit alors que ces porteuses sont en quadrature.

A l'aide de cette technique on obtient donc 2 nouveaux signaux différents :

- La **composante en phase (I)**, qui est obtenue en multipliant le signal initial par une porteuse cosinus.
- La **composante en quadrature (Q)**, qui est obtenue en multipliant le signal initial par une porteuse sinus.

Cela donnerait un schéma ressemblant à ceci :

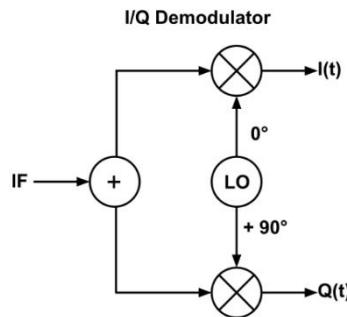


Figure 2.3 – Schéma d'un démodulateur en quadrature (I/Q) [8]

Les 2 signaux I et Q permettent ensuite d'obtenir 2 informations importantes :

- **L'amplitude instantanée** ou aussi appelé l'enveloppe, qui décrit l'intensité du signal à chaque instant.
- **La phase instantanée**, qui donne la position du signal dans son cycle à un moment donné. (Ne nous intéresse pas dans le contexte de notre projet)

Attention en multipliant le signal d'origine par nos 2 porteuses, les signaux que l'on obtient (I et Q) contiennent non seulement des composantes de basse fréquence (donc l'information que l'on recherche), mais contiennent aussi des composantes haute fréquence, ce qui représente des perturbations indésirables dans ce contexte. Pour les éliminer, il est donc nécessaire d'appliquer un filtre sur nos composantes I et Q.

Le calcul de l'enveloppe se fait de la même manière que pour la transformée de Hilbert mais cette fois-ci nous calculons le module des composantes I et Q pour obtenir :

$$\text{Enveloppe} = \sqrt{I^2 + Q^2}$$

En résumé, la démodulation par quadrature utilise 2 signaux de référence déphasés de 90 degrés, afin d'extraire l'enveloppe de notre signal d'origine. Cette partie s'appuie notamment sur le document [8].

2.4 Démodulation par PLL

La démodulation par PLL (Phase-Locked Loop) est une technique très souvent employée dans le traitement de signal afin d'extraire diverses informations d'un signal modulé. Une PLL est un système de contrôle électrique qui est capable de générer automatiquement un signal dont la fréquence et la phase s'ajustent sur celles du signal d'entrée.

Le fonctionnement d'une PLL repose sur 3 composants principaux :

- Un **comparateur de phase**, qui sert à mesurer la différence de phase entre le signal d'entrée et celui généré par le VCO
- Un **filtre passe-bas**, qui vient lisser la différence reçue du comparateur de phase afin de produire la tension de contrôle envoyée au VCO
- Un **oscillateur commandé en tension (VCO)**, qui vient générer un signal sinusoïdal dont la fréquence varie en fonction de la tension de contrôle reçue du filtre passe-bas.

Voici le schéma interne d'une PLL :

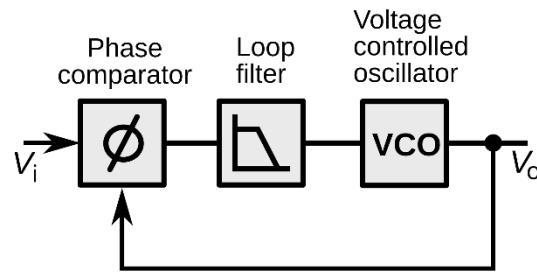


Figure 2.4 – Bloc fonctionnel d'une PLL (Phase-Locked Loop) [9]

Lorsqu'une PLL est utilisée pour démoduler un signal modulé en amplitude ou en fréquence, alors elle s'ajuste automatiquement afin de produire un signal de référence qui est aligné en phase et en fréquence avec le signal d'entrée. Ce processus nous permet d'extraire les informations suivantes :

- **L'amplitude instantanée** ou aussi appelé l'enveloppe, qui décrit l'intensité du signal à chaque instant
- La **phase instantanée**, qui donne la position du signal dans son cycle à un moment donné. (Ne nous intéresse pas dans le contexte de notre projet)
- La **fréquence instantanée**, qui correspond à la vitesse à laquelle la phase du signal évolue dans le temps. (Ne nous intéresse pas dans le contexte de notre projet)

En pratique, lorsque l'on souhaite obtenir l'amplitude instantanée (enveloppe) du signal modulé en amplitude à l'aide d'une PLL, on utilise le signal généré par la PLL (sortie du VCO) comme référence, pour multiplier le signal d'entrée. Cette multiplication permet d'effectuer une démodulation synchrone. Et comme pour la démodulation en quadrature un filtre passe-bas est nécessaire pour extraire les informations qui nous sont utiles.

Voici un schéma de notre démodulateur par PLL complet (le décalage de 90° est optionnel selon l'application) :

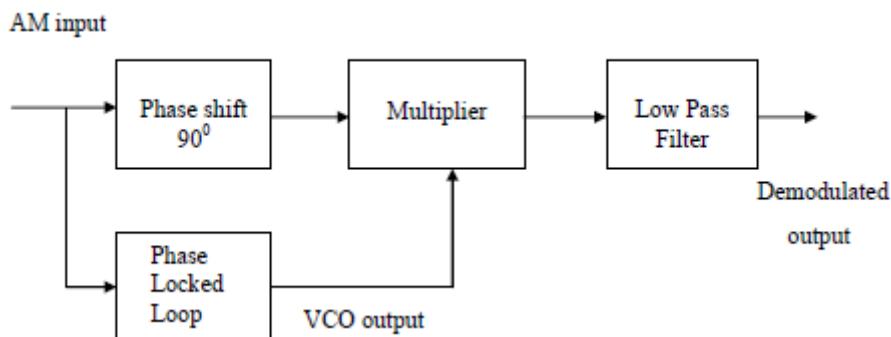


Figure 2.5 – Architecture de démodulation basée sur une PLL [10]

En résumé, la PLL s'adapte automatiquement au signal d'entrée afin de générer un signal de référence aligné en fréquence et en phase. Ce mécanisme facilite l'extraction de l'enveloppe de notre signal d'origine. Il est important de savoir qu'il existe différents types de PLL, comme la Costas Loop, qui combine les principes d'une PLL classique et ceux d'une démodulation en quadrature. Cette partie s'appuie notamment sur les documents [9] et [11].

2.5 Démodulation par Costas Loop

La Costas Loop est un circuit basé sur une PLL, c'est une technique avancée de démodulation. Cette technique est principalement utilisée pour faire de la démodulation de signaux modulés en phase ou en amplitude.

La Costas Loop fonctionne comme une PLL, mais avec un circuit de démodulation en quadrature. C'est-à-dire que le circuit prend le schéma de base d'une démodulation en quadrature et la combine dans une architecture en boucle fermée pour suivre et corriger dynamiquement la phase du signal.

Voici un schéma d'une Costas Loop :

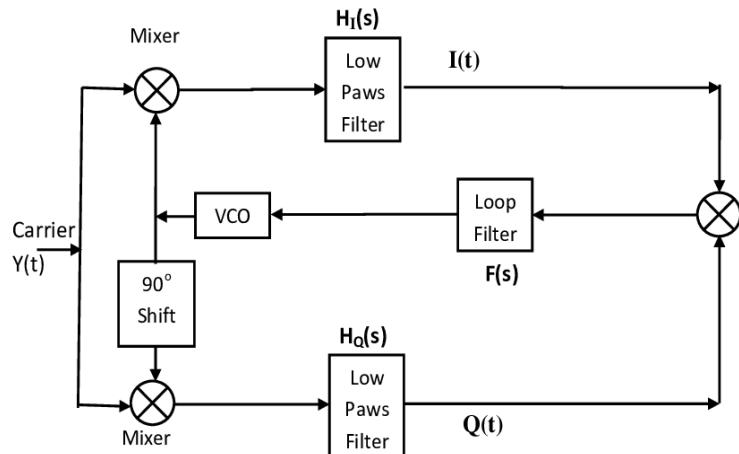


Figure 2.6 – Démodulateur en quadrature avec principe PLL (Costas Loop) [12]

Ce schéma montre clairement que l'on a bel et bien une démodulation en quadrature avec la multiplication du signal d'entrée avec 2 signaux porteuses, qui nous génèrent au travers d'un filtre passe-bas, nos sorties I et Q, puis ces signaux sont multipliés afin de produire l'erreur de phase. Comme pour une PLL classique cette erreur de phase est envoyée dans un filtre puis dans une VCO. La différence par rapport à une PLL classique est que la sortie du VCO doit générer 2 signaux porteuses. Et nous voilà avec un circuit à boucle fermée avec une partie utilisant une PLL classique et l'autre qui utilise une démodulation en quadrature, qui nous donne un circuit appelé une Costas Loop.

La Costas Loop nous permet donc d'extraire les informations suivantes :

- L'**amplitude instantanée** ou aussi appelé l'enveloppe, qui décrit l'intensité du signal à chaque instant
- La **phase instantanée**, qui donne la position du signal dans son cycle à un moment donné. (Ne nous intéresse pas dans le contexte de notre projet)
- La **fréquence instantanée**, qui correspond à la vitesse à laquelle la phase du signal évolue dans le temps

En pratique, lorsque l'on souhaite obtenir l'amplitude instantanée (enveloppe) du signal modulé en amplitude à l'aide d'une Costas Loop, on utilise les signaux I/Q générés par la Costas Loop comme référence et comme pour la démodulation en quadrature un filtre passe-bas est nécessaire pour extraire les informations qui nous sont utiles. Puis pour finir, le calcul de l'enveloppe se fait de la même manière que pour une démodulation en quadrature classique, nous calculons donc le module des composantes I et Q pour obtenir :

$$\text{Enveloppe} = \sqrt{I^2 + Q^2}$$

La Costas Loop est une technique de démodulation basée sur une boucle fermée avec démodulation en quadrature, qui permet de suivre dynamiquement la phase d'un signal modulé et d'en extraire l'amplitude. Cette partie s'appuie notamment sur les documents [13] et [14].

Chapitre 3

Analyse comparative des algorithmes

3.1 Comparaison expérimentale des algorithmes

Maintenant que nous avons étudié le fonctionnement des différents algorithmes de démodulation, nous allons les tester pour voir lequel de ces algorithmes a les meilleures performances. Pour ce faire, un code python a été réalisé où nous utilisons une version simulée d'un capteur RVDT. Nous allons donc utiliser les signaux suivants tout au long de notre simulation en y ajoutant du bruit :

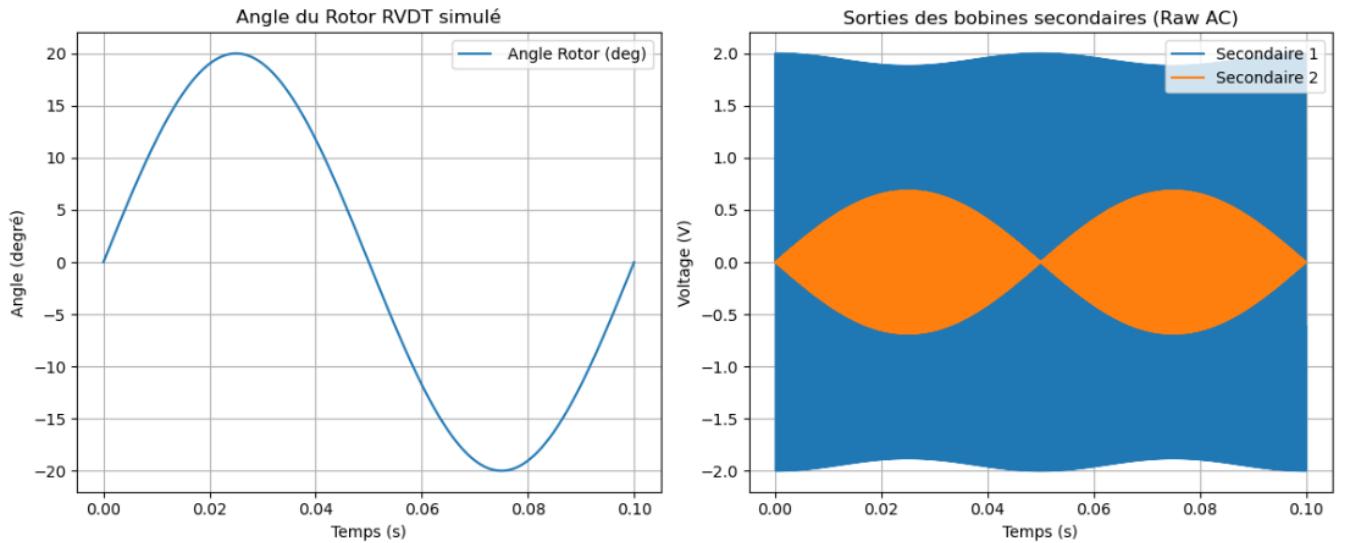


Figure 3.1 – Simulation d'un signal RVDT : angle du rotor (gauche) et signaux aux secondaires (droite)

3.1.1 Paramètre réel

Avant de s'aventurer dans les différents algorithmes, il nous faut connaître les différents paramètres réels du resolver du CERN. Pour ce faire nous avons à disposition différents fichiers contenant les signaux reçus du RVDT, plus précisément de l'ADC, ainsi que diverses autres informations, à savoir :

- Fréquence d'échantillonnage de l'ADC : 10 MHz
- Fréquence d'excitation du resolver : 10 kHz
- Donnée transmise au travers d'un câble de 220m

Dans ces informations, un point essentiel concerne la transmission des données sur un câble de 220 m. À cette distance, le signal issu du RVDT est inévitablement dégradé par le bruit. Pour évaluer ce bruit, j'ai exploité les différentes acquisitions disponibles. J'ai donc calculé, pour chaque crête, l'écart (en %) par rapport à la valeur crête-à-crête maximale du signal. Le résultat est présenté à la **figure 3.2**.

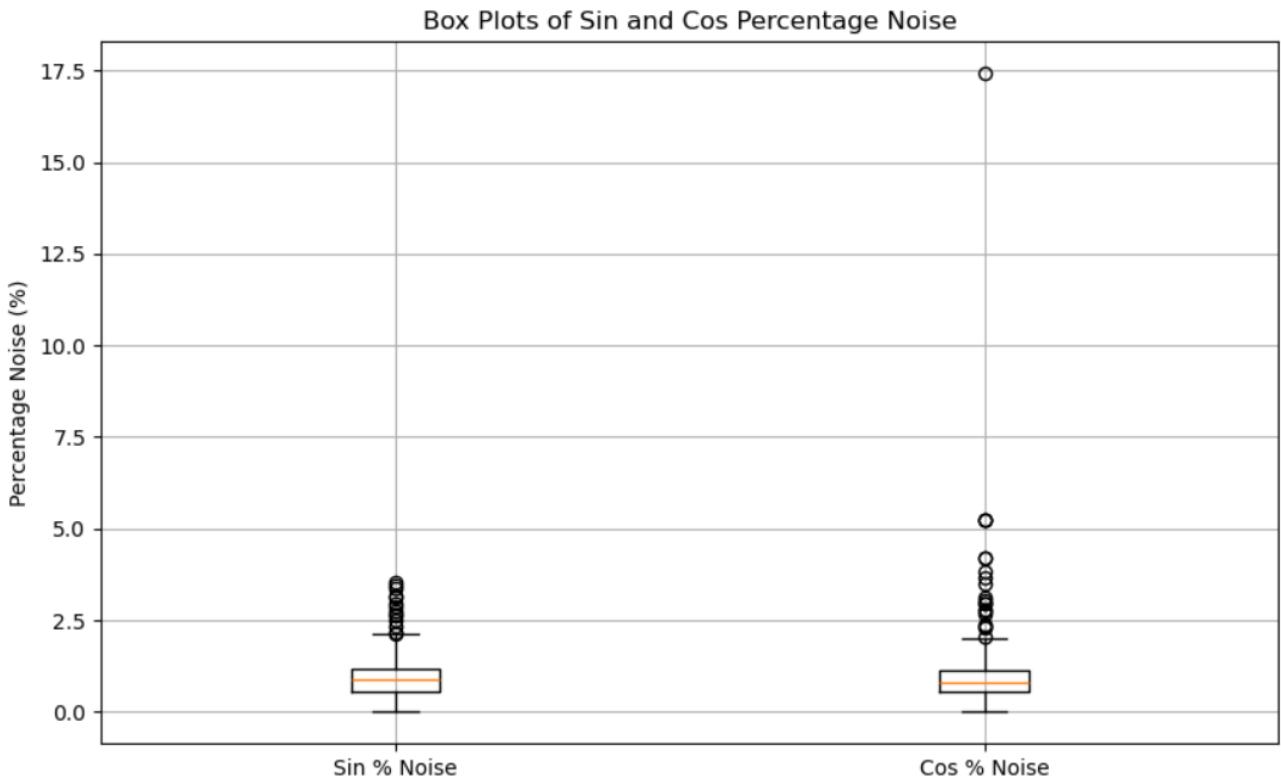


Figure 3.2 – Analyse du bruit relatif sur les signaux sin et cos

La figure ci-dessus nous présente les résultats obtenus sous forme de boîte à moustaches les pourcentages de bruit calculés pour les différents signaux reçus du RVDT.

Le bruit médian observé sur les signaux se situe aux alentours de 1%, ce qui est acceptable et même relativement faible pour un signal transmis au travers d'un câble de 220 mètres. Cependant, on aperçoit malgré tout un bruit de 17,5% sur l'un des peak du signal Cosinus, ces valeurs aberrantes peuvent parfois apparaître par exemple lorsque nous avons des perturbations électromagnétiques, des problèmes de connexion intermittents, ou d'autres facteurs environnementaux. Étant donné le cas exceptionnel de cette valeur élevée, elle sera ignorée lors des tests de simulation avec bruit continu.

Ainsi, la figure nous montre donc que notre bruit varie entre environ 0 et 5%. Pour s'assurer de disposer d'une marge suffisante et garantir la robustesse du système lors des simulations, un niveau de bruit fixé à 6 % sera appliqué à notre simulation afin de confirmer le bon fonctionnement du système dans des conditions réelles.

3.1.2 Démodulation par Transformée de Hilbert

3.1.2.1 Fonctionnement normal

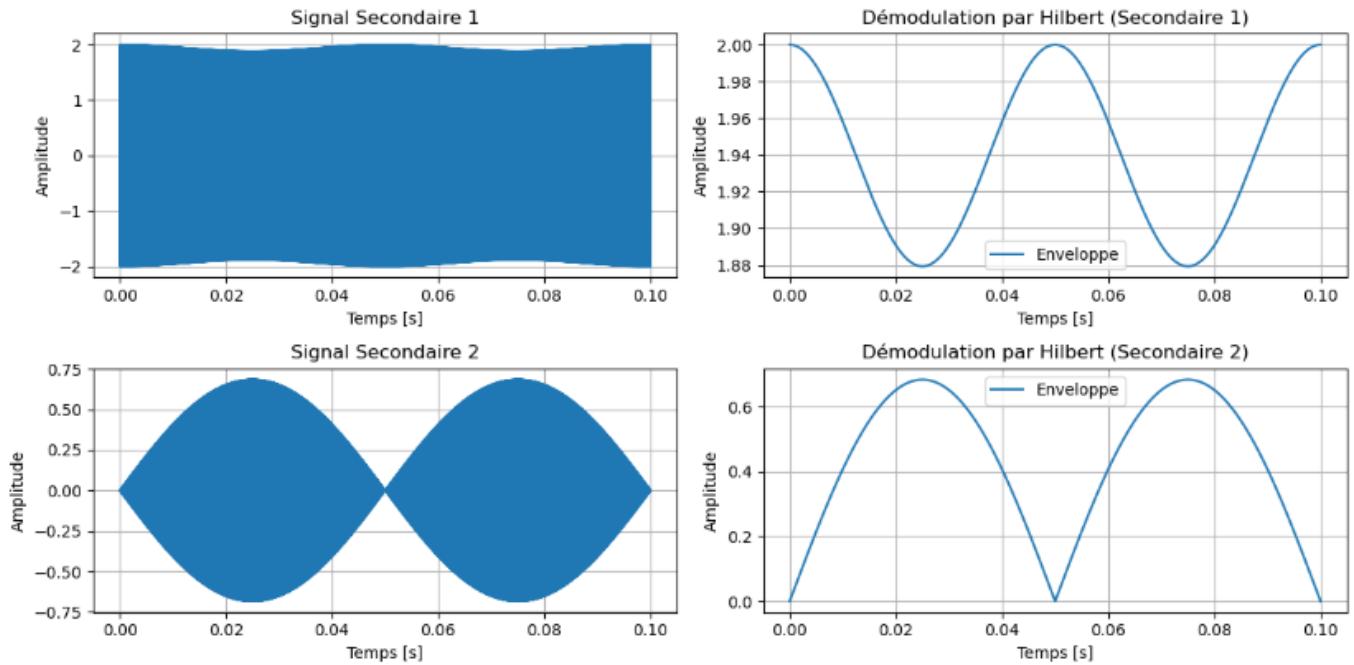


Figure 3.3 – Démodulation des signaux secondaires par transformée de Hilbert (sans bruit)

3.1.2.2 Fonctionnement avec bruit

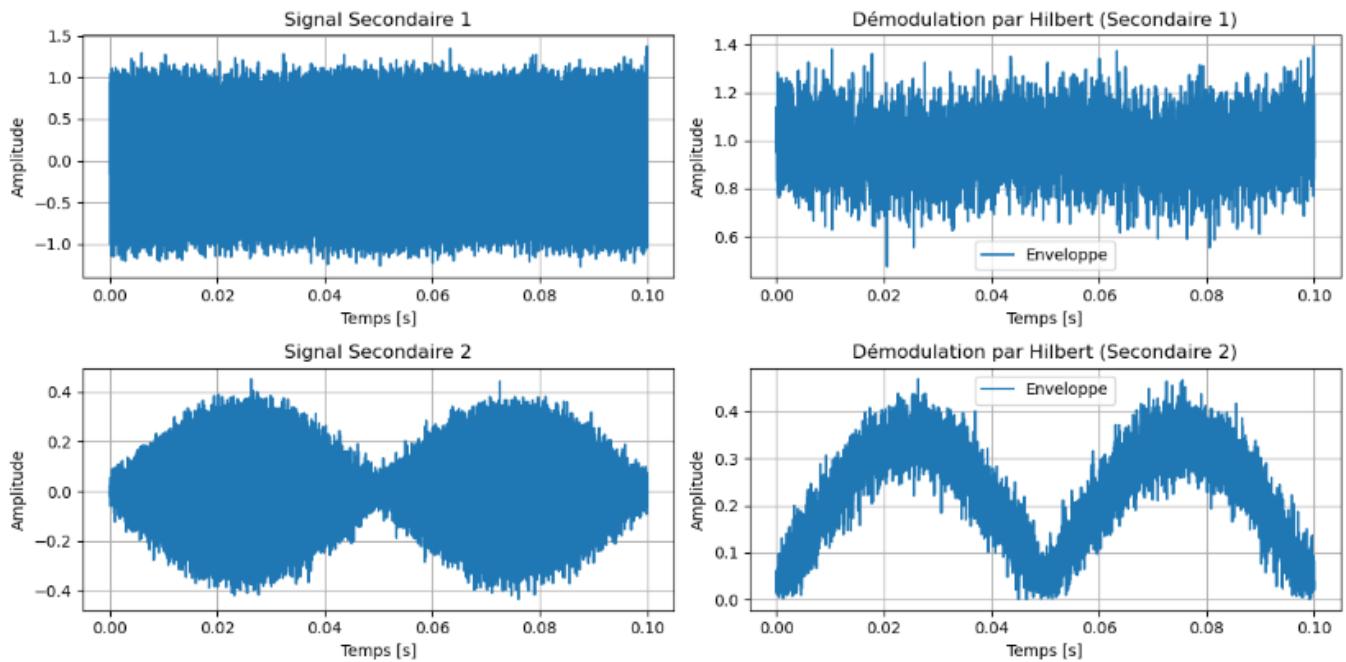


Figure 3.4 - Démodulation des signaux secondaires par transformée de Hilbert (avec bruit)

3.1.3 Démodulation en Quadrature (I/Q)

3.1.3.1 Fonctionnement normal

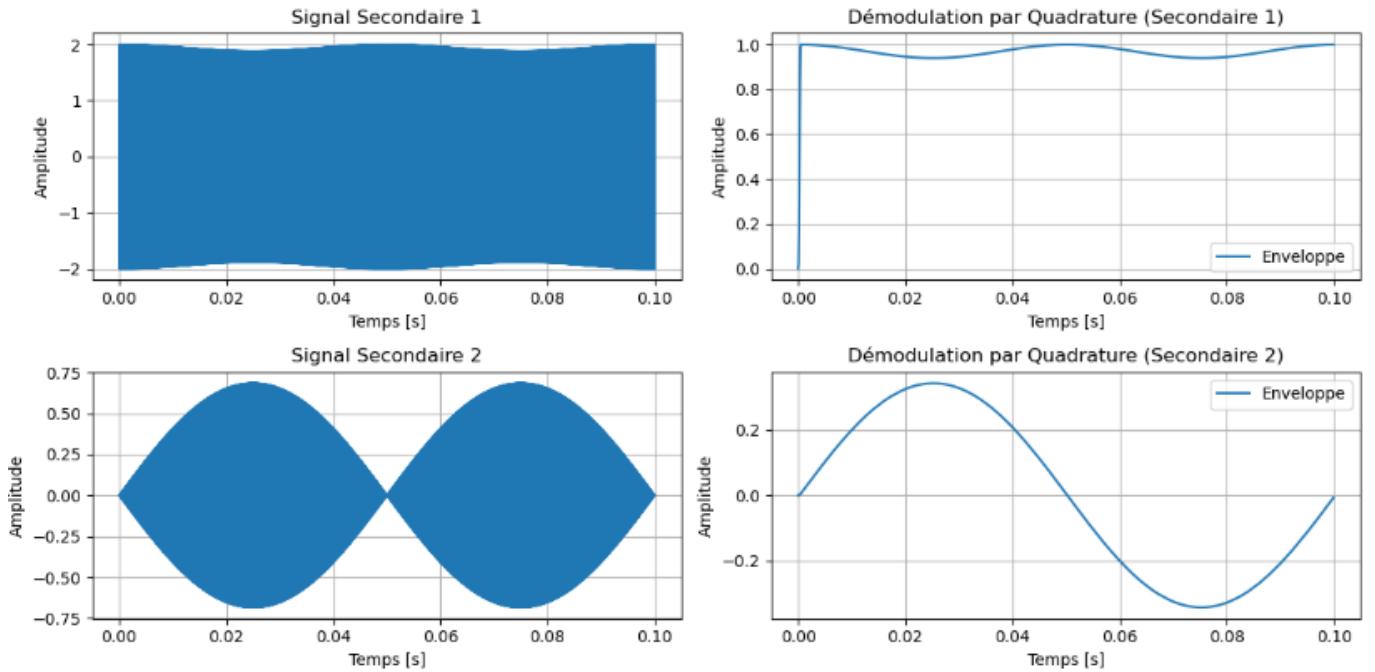


Figure 3.5 - Démodulation des signaux secondaires par Quadrature (sans bruit)

3.1.3.2 Fonctionnement avec bruit

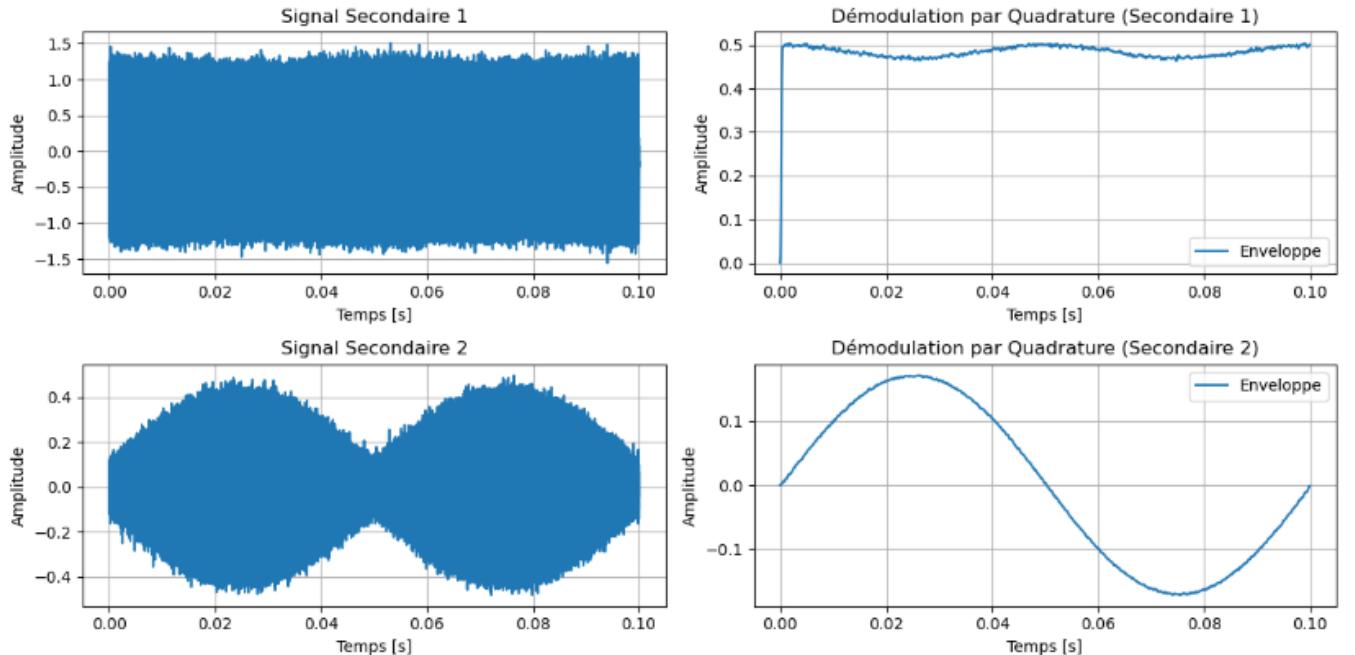


Figure 3.6 - Démodulation des signaux secondaires par Quadrature (avec bruit)

3.1.4 Démodulation par PLL

3.1.4.1 Fonctionnement normal

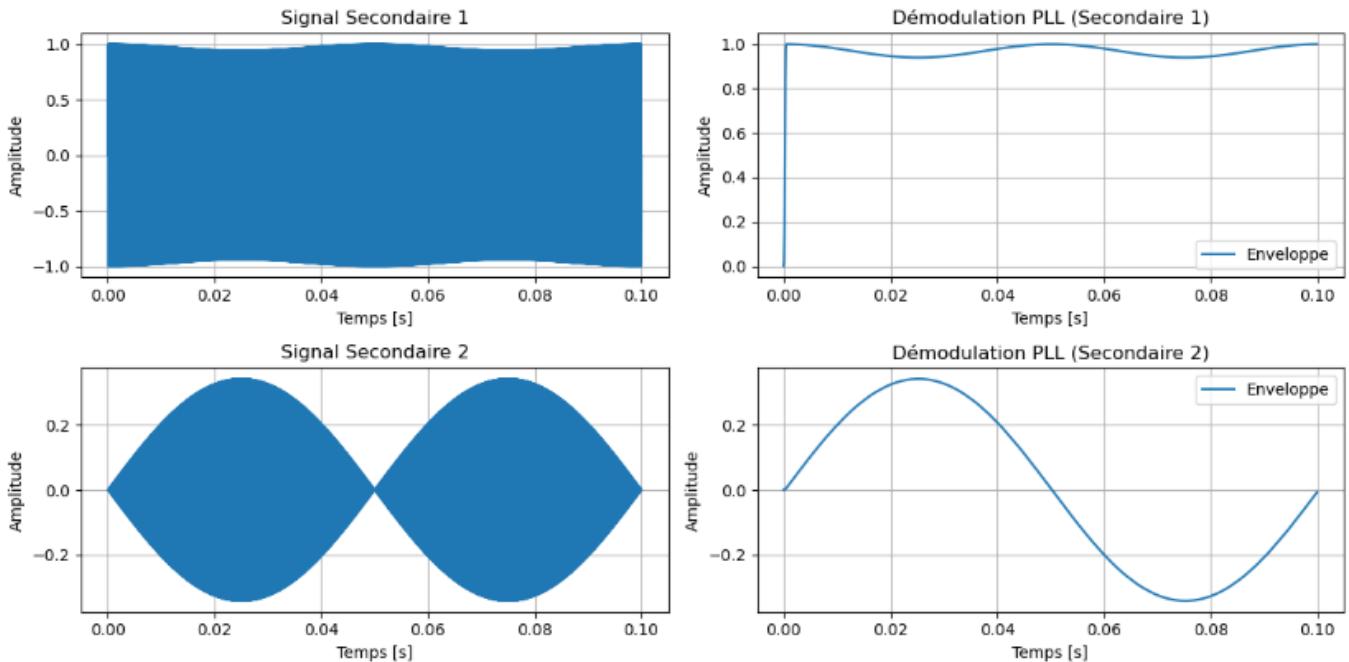


Figure 3.7 - Démodulation des signaux secondaires par PLL (sans bruit)

3.1.4.2 Fonctionnement avec bruit

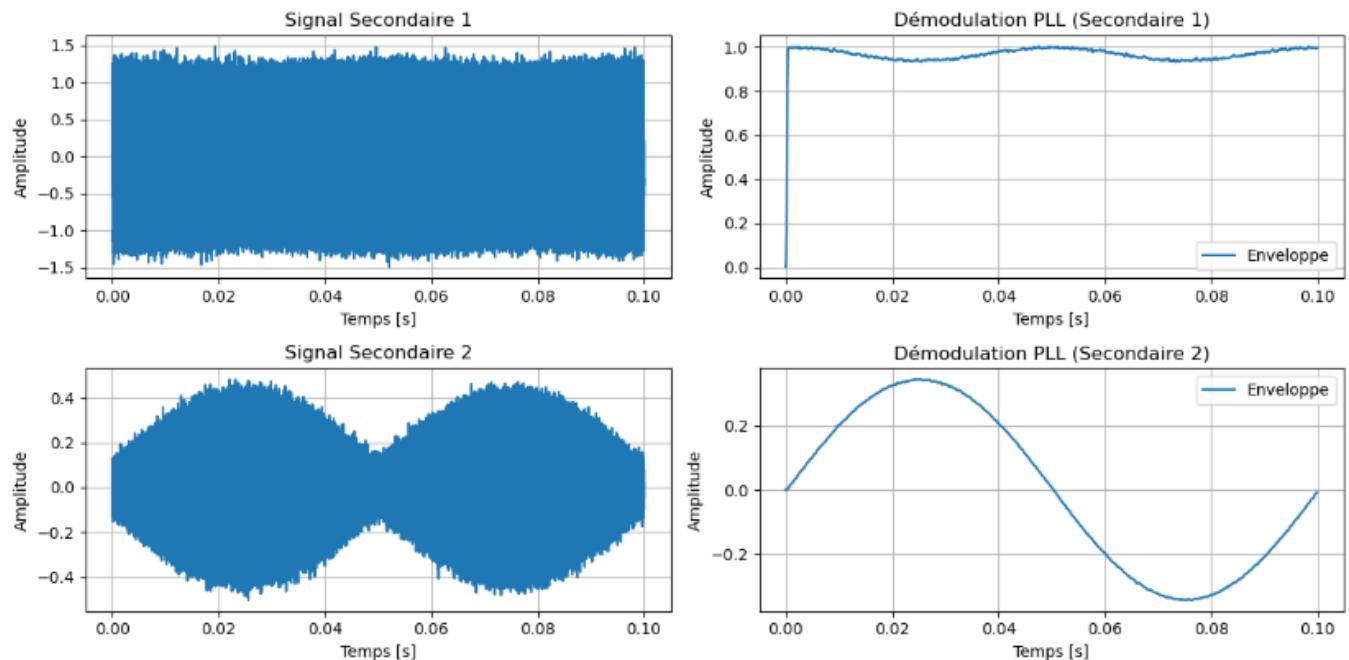


Figure 3.8 - Démodulation des signaux secondaires par PLL (avec bruit)

3.1.5 Démodulation par Costas Loop

3.1.5.1 Fonctionnement normal

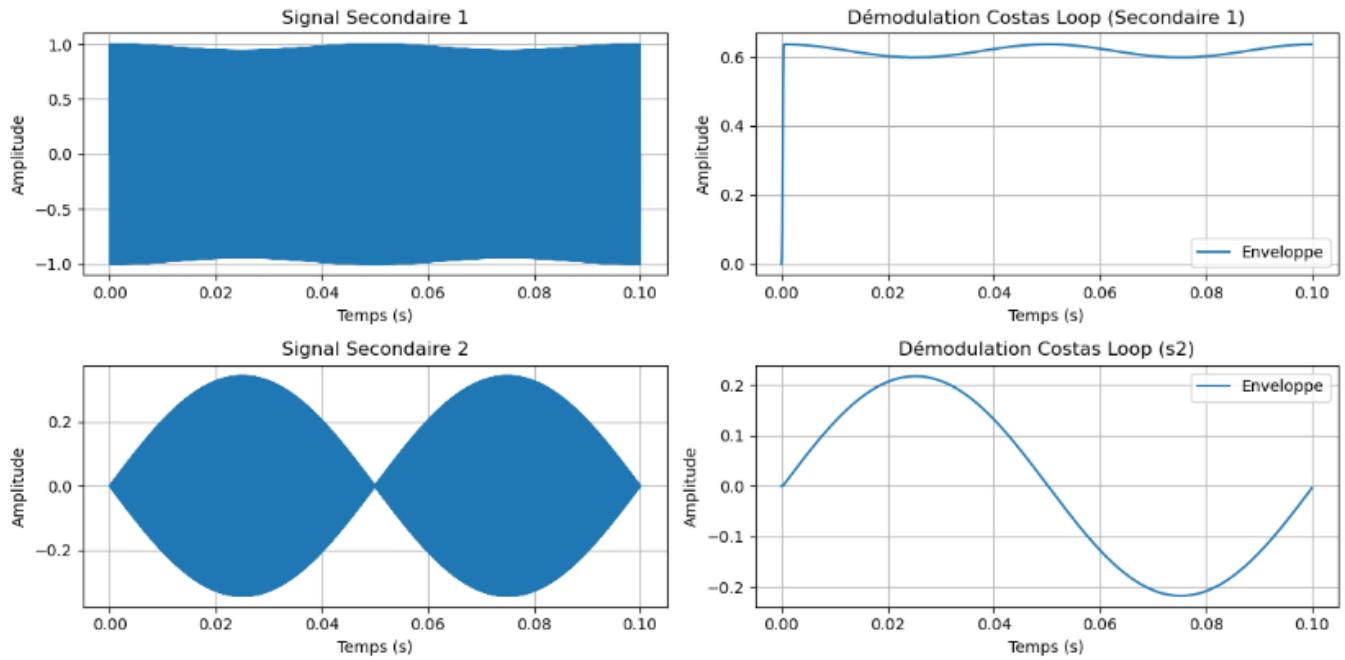


Figure 3.9 - Démodulation des signaux secondaires par Costas Loop (sans bruit)

3.1.5.2 Fonctionnement avec bruit

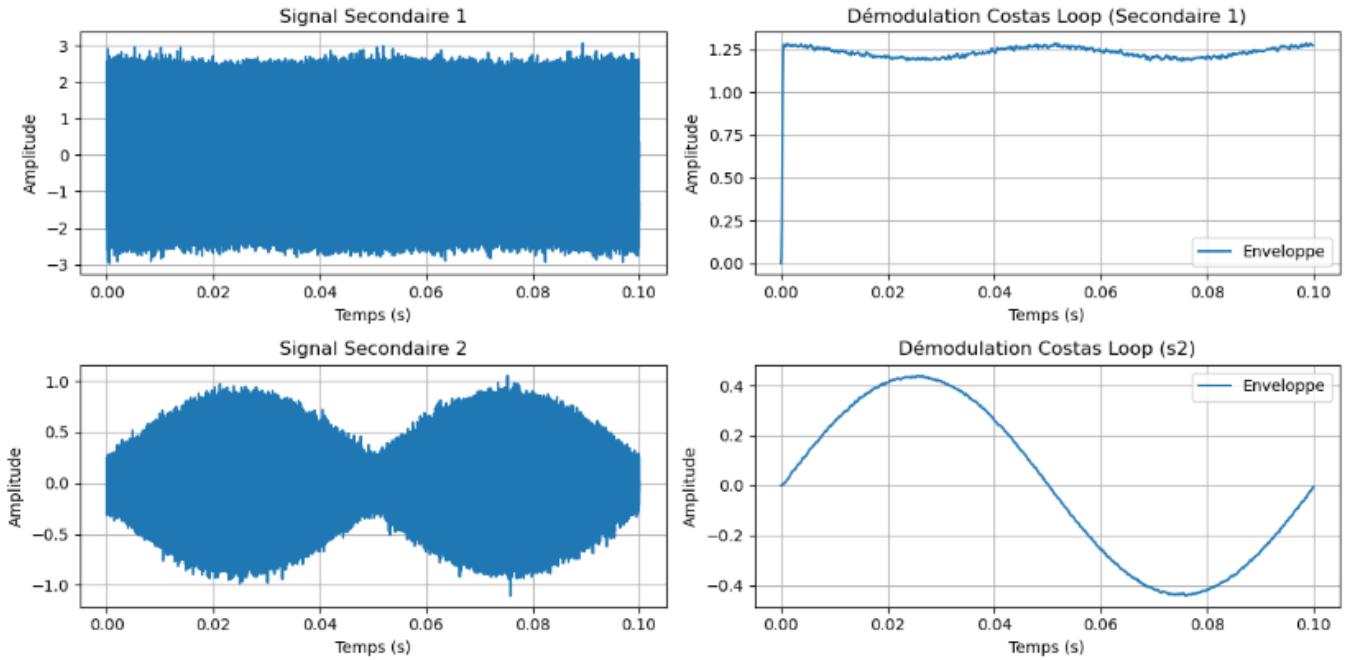


Figure 3.10 - Démodulation des signaux secondaires par Costas Loop (avec bruit)

3.1.6 Analyse des résultats

Les différentes techniques de démodulation ci-dessus ont été testées dans un environnement simulé, en prenant en compte les conditions réelles d'utilisation du resolver du CERN cité au point 3.1.1. L'analyse s'est portée sur 2 aspects à l'aide des signaux simulés, le premier est le fonctionnement idéal (sans bruit) et le second est le fonctionnement perturbé (avec un bruit de $\pm 6\%$ ajouté aux signaux). Grâce à cela, nous pouvons voir quelles sont les algorithmes de démodulation réagissant le mieux aux perturbations.

Résumé des observations :

Méthode	Précision (sans bruit)	Robustesse au bruit	Complexité d'implémentation FPGA
Hilbert	Bonne	Mauvaise	Élevée (FFT)
Quadrature	Excellente	Bonne	Faible
PLL	Excellente	Bonne	Moyenne à élevée (paramétrage complexe)
Costas Loop	Excellente	Bonne	Élevée (Mixe quadrature et PLL)

Tableau 3.1 - Comparaison des principales méthodes de démodulation pour RVDT/LVDT

Analyse détaillée :

- **Transformée de Hilbert** : Cette méthode donne de très bons résultats en condition idéale, mais elle présente plusieurs problèmes. La première est qu'elle fournit uniquement l'enveloppe absolue du signal, ce qui nous limite dans son usage dans les prochaines étapes. Deuxièmement, elle est aussi énormément plus sensible aux bruits que les autres techniques. De plus, la transformée de Hilbert utilise une FFT (Fast-Fourier Transform) qui est un algorithme complexe à implémenter dans un FPGA car elle demande énormément de ressources.
- **Quadrature (I/Q)** : Comme le montrent les figures, la démodulation par quadrature est une technique robuste. En effet, cette méthode montre de bonnes performances en conditions idéales, mais réagit aussi à merveille lorsque nous avons du bruit. La démodulation par quadrature est aussi un algorithme très simple à implémenter sur FPGA puisqu'elle ne fait principalement que des additions et des multiplications. On peut tout de même remarquer un défaut aux débuts de chacune des enveloppes, nous avons en effet un pic qui est causé par le temps de convergence de nos filtres. Ce comportement n'a cependant aucun impact dans notre cas, car notre système est conçu pour fonctionner en continu.
- **PLL** : Les résultats observés montrent que l'on obtient de très bons résultats en condition réelle ainsi que lorsque le signal est perturbé. Nous pouvons aussi remarquer le même problème que la quadrature où le début de notre signal met du temps à converger au démarrage, mais comme pour la méthode précédente ce phénomène est négligeable. Quant à l'implémentation sur FPGA, elle reste tout de même complexe notamment en raison de la complexité du réglage de ses composants internes, qui demande d'être précis afin de garantir un système stable et performant.
- **Costas Loop** : La démodulation par Costas Loop qui combine les principes d'une PLL et d'une démodulation par quadrature, nous donne des résultats similaires à ceux de la PLL et de la démodulation par quadrature. Étant donc un système hybride, il en va de soi que ce système hérite de la complexité des 2 techniques précédentes et est par conséquent assez compliqué à implémenter sur FPGA.

3.2 Choix de l'algorithme

À la suite de cette simulation des différents algorithmes de démodulation, ainsi qu'après une discussion avec monsieur Herzog Raoul, Professeur à la HEIG-VD spécialisé en traitement du signal numérique (DSP), au sujet de ces algorithmes, le choix s'est porté sur la démodulation par quadrature. Ce choix repose sur plusieurs critères essentiels liés à notre système :

- La fréquence de notre signal est connue et fixe (10 kHz), ce qui rend donc inutile l'utilisation d'algorithmes comme la PLL ou la Costas Loop, dont l'avantage principal est de retrouver dynamiquement la fréquence du signal.
- La démodulation par transformée d'Hilbert a une très faible robustesse face au bruit par rapport aux autres méthodes, ce qui est un point crucial.
- La démodulation par quadrature est simple d'implémentation en FPGA, car elle n'utilise que de simples additions et multiplications.
- Et pour finir les performances observées lors des simulations, tant en condition idéale qu'en présence de bruit, nous montrent que la méthode par quadrature est très robuste et précise.

Chapitre 4

Quadrature I/Q, filtre CIC et contraintes

4.1 Fonctionnement Quadrature

Cette section détaille le principe de la démodulation par quadrature, également appelée démodulation I/Q. Avant d'y entrer en détail, il est préférable de commencer par expliquer ce qu'est la modulation I/Q.

4.1.1 Modulation I/Q :

La modulation I/Q est une technique utilisée en transmission numérique et traitement du signal. Elle nous est très utile pour moduler simultanément 2 signaux indépendants sur une seule porteuse dite "sinusoïdale". Considérons un signal modulé ou non modulé, nous avons par exemple cette formule :

$$A(t) \cdot \sin(2\pi f t + \phi(t))$$

Où :

- $A(t)$ = l'amplitude de notre signal dans le temps, donc elle peut varier (cf. Figure 4.1 gauche) ou être constante (cf. Figure 4.1 droite)
- f = fréquence du signal
- $\phi(t)$ = déphasage de phase

Ce qui représente des signaux de ce style :

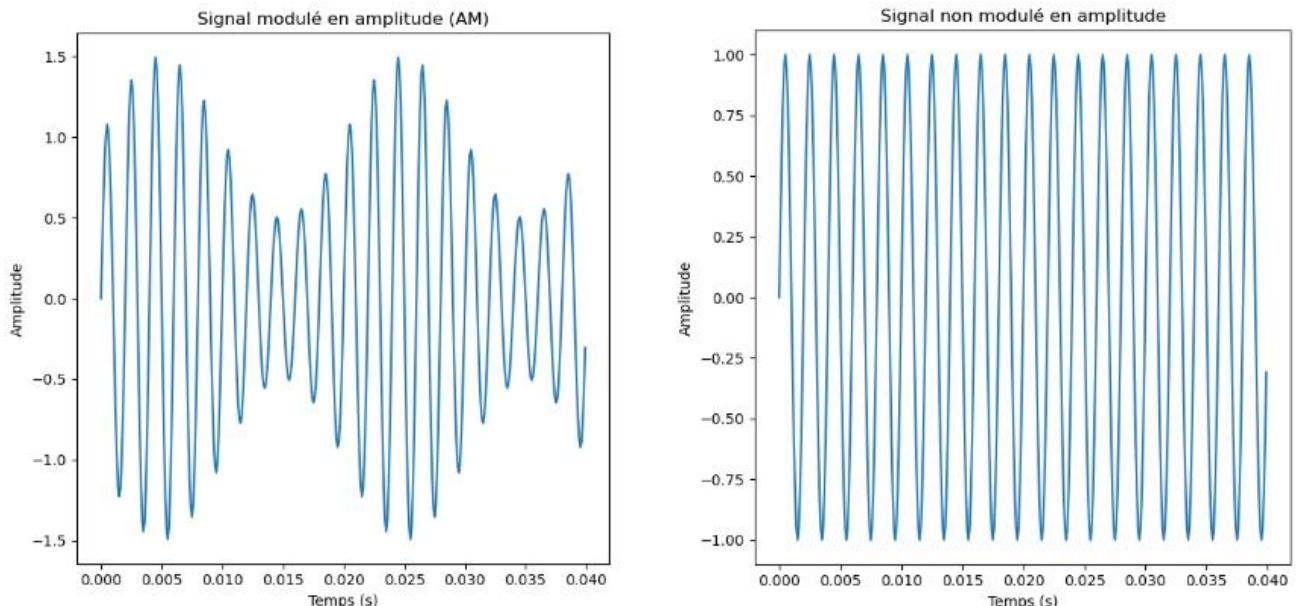


Figure 4.1 – Comparaison entre un signal AM et un signal non modulé

Comme cité plus tôt, afin d'avoir une modulation I/Q, il nous faut 2 signaux indépendants qui ont la caractéristique d'être orthogonaux (aussi dits en quadrature de phase), qui sont donc déphasés de $\pm 90^\circ$. Prenons

donc comme exemple un signal sinusoïdal et un signal cosinus :

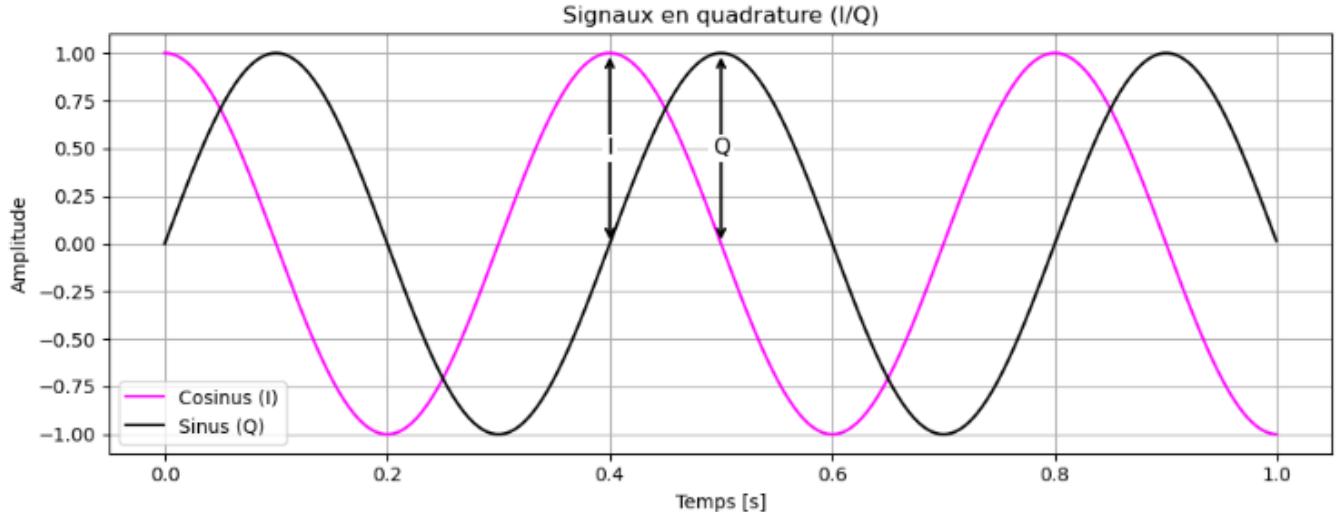


Figure 4.2 – Représentation temporelle de deux signaux en quadrature (I/Q)

Conventionnellement, nous avons donc ceci :

- L'Amplitude du signal "In-Phase" est représenté par I
 $I(t) \cdot \cos(2\pi ft + \varphi(t))$
- L'Amplitude du signal "shifté de 90°" est représenté par Q
 $Q(t) \cdot \sin(2\pi ft + \varphi(t))$

Maintenant que nous avons nos 2 signaux en quadrature, nous pouvons générer notre signal modulé. Un signal modulé par quadrature est représenté par le schéma suivant :

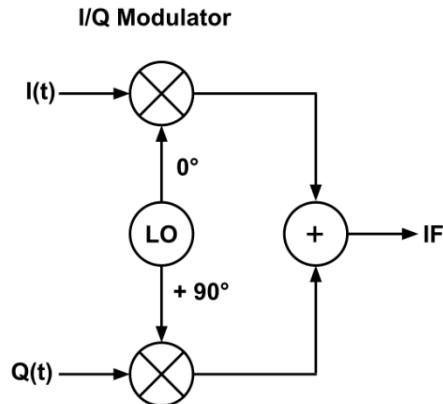


Figure 4.3 - Schéma d'un modulateur en quadrature (I/Q) [8]

La partie multiplication étant faite il ne nous reste plus qu'à additionner les 2 signaux ensemble. Dans le cadre de l'addition des 2 signaux présentés à la figure 4.2 nous obtenons le résultat suivant :

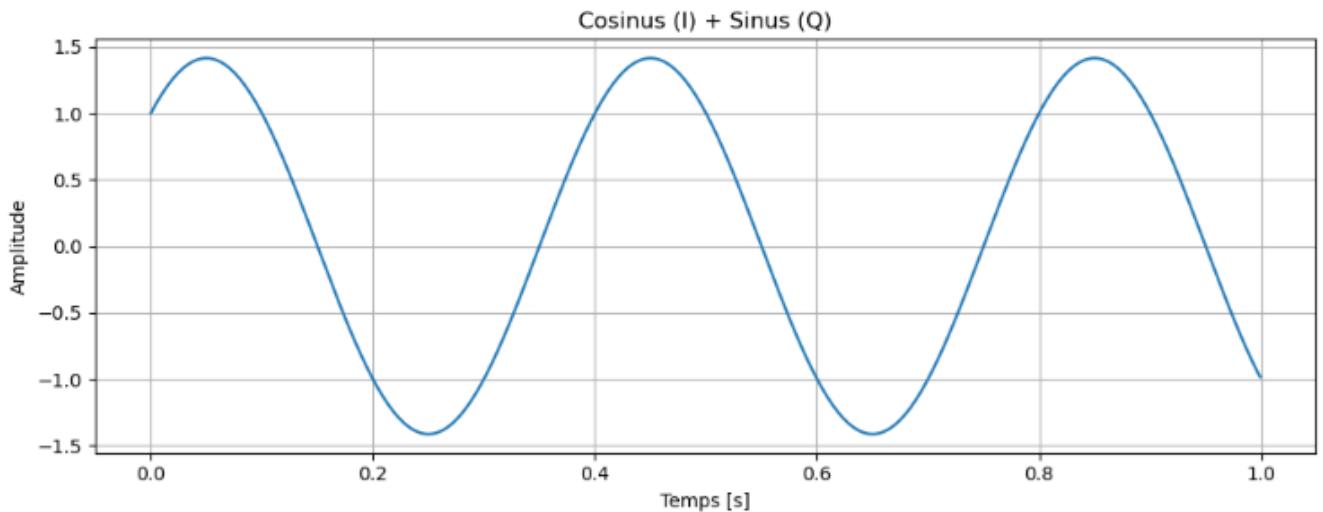


Figure 4.4 - Signal résultant de la somme des composantes I et Q

Les points importants à noter sont les suivants :

- Lorsque I et Q varient de manière similaire, alors l'amplitude de la somme varie.
- Lorsque I et Q varient de manière différente, alors la phase et l'amplitude de la somme varie.

Nous avons donc une modulation où une variation de I et Q résulte en une modulation d'amplitude et de phase.

Voici ci-dessous 2 exemples où I et Q varient différemment :

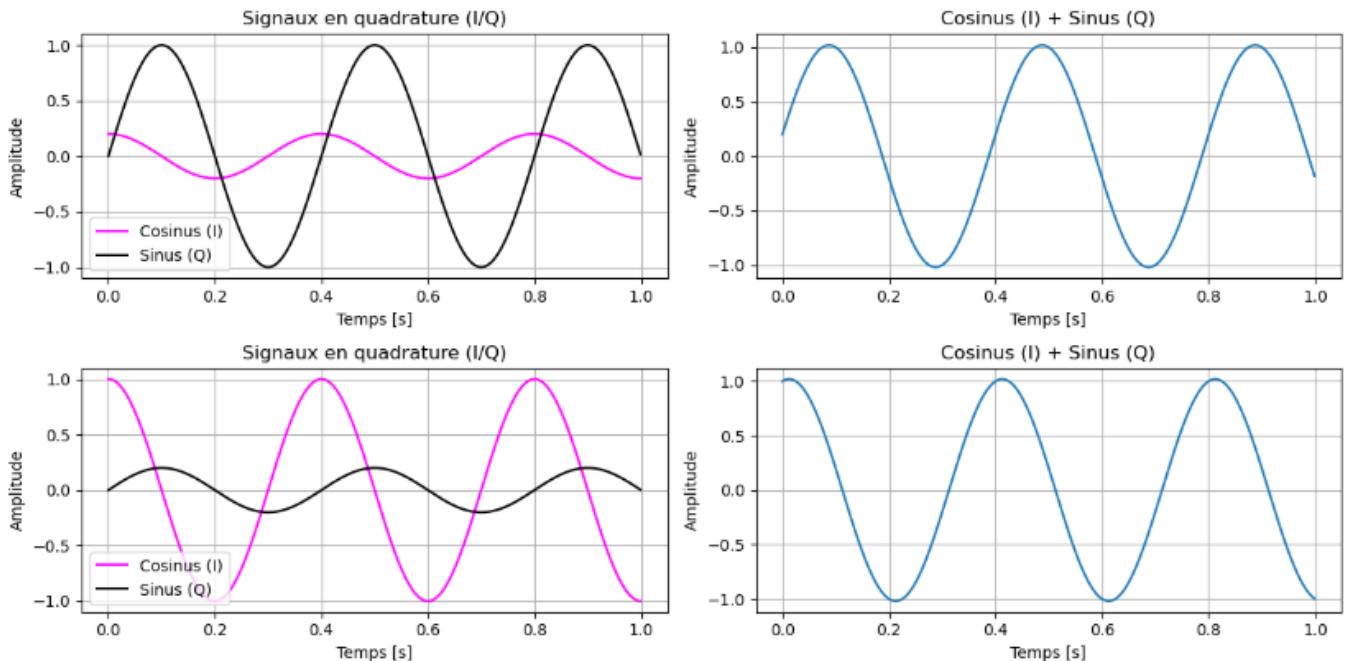


Figure 4.5 – Influence relative des composantes I et Q sur la forme du signal modulé

2 observations peuvent être faites dans ces 2 exemples par rapport à l'exemple où $I = Q$ (cf. Figure 4.4) qui affirme les 2 conventions concernant la variation de I et Q :

- L'amplitude a diminué à la suite de la diminution de I ou Q
- Lorsque l'on diminue I, notre sortie est de plus en plus en phase avec le sinus d'entrée Q
- Lorsque l'on diminue Q, notre sortie est de plus en plus en phase avec le cosinus d'entrée I

On peut donc en conclure qu'avec la modulation I/Q chaque composante (I et Q) pilote un axe orthogonal du vecteur complexe. La figure 4.6 nous montre un résumé des explications précédentes.

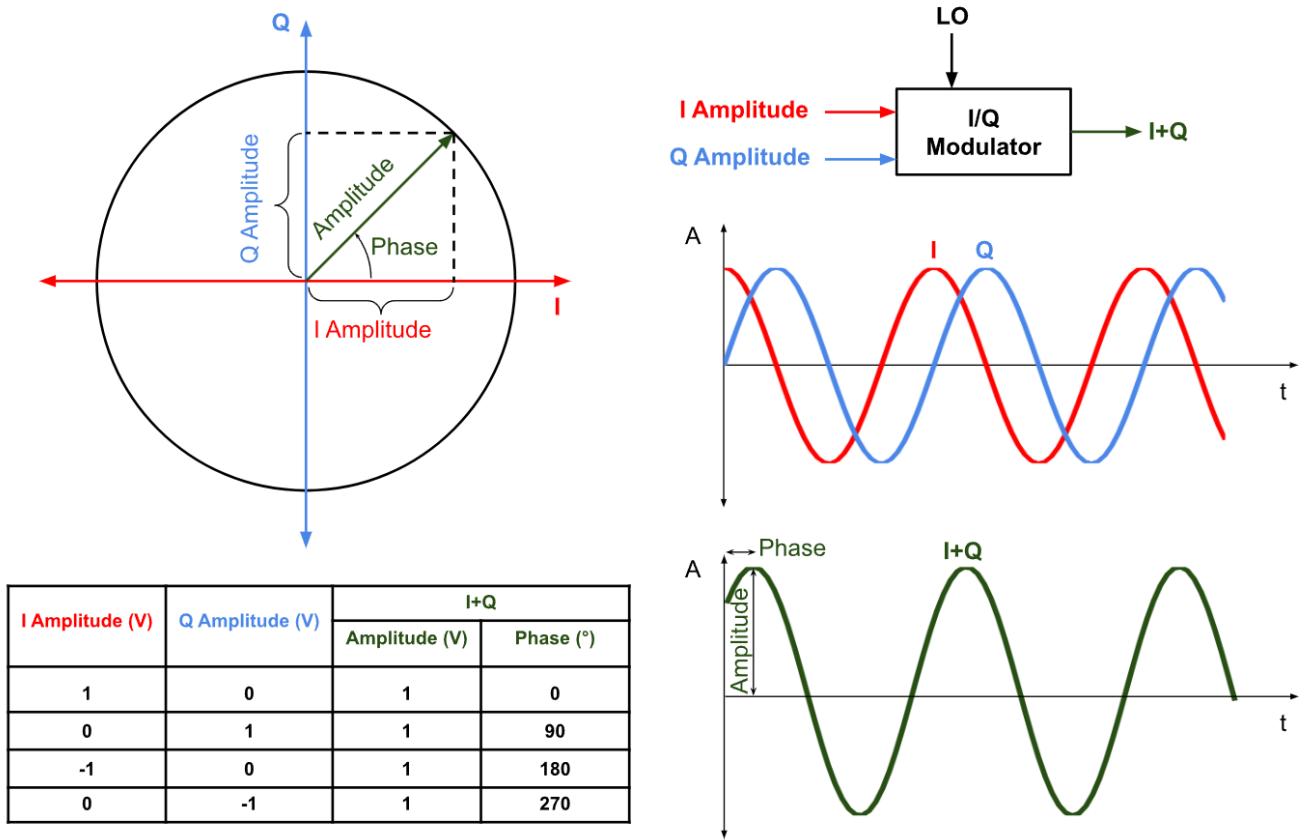


Figure 4.6 – Représentation des composantes I/Q et de leur combinaison vectorielle [8]

Cela signifie que notre signal modulé en quadrature, contient 2 informations qui sont un sinus (Q) et un cosinus (I), cette information est importante pour la suite. Pour un complément d'explication, on pourra se référer à la vidéo [15].

4.1.2 Démodulation I/Q :

La démodulation I/Q est une méthode de traitement du signal numérique nous permettant d'extraire l'amplitude et la phase d'un signal modulé. Comme expliqué au point 2.3, cette méthode utilise 2 porteuses, un sinus et un cosinus (aussi appelés des signaux en quadrature de phase ou orthogonal), ayant tous 2 la même fréquence que le signal d'excitation du capteur (qui est de 10 kHz dans notre configuration)

Principe de base :

Le signal d'entrée, que nous allons nommer $x(t)$, est multiplié simultanément par nos 2 signaux orthogonaux, soit :

- Un signal sinus : $\sin(2\pi ft)$
- Un signal cosinus : $\cos(2\pi ft)$

Où, comme expliqué plutôt, f est la fréquence d'excitation de notre capteur RVDT, qui est connue et fixe. Avec cela on obtient donc deux composantes qui sont les suivantes :

- La composante en phase (I) : $I(t) = x(t) \cdot \cos(2\pi ft)$
- La composante en quadrature (Q) : $Q(t) = x(t) \cdot \sin(2\pi ft)$

À l'aide de ces 2 composantes nous pouvons maintenant retrouver l'amplitude et la phase du signal initial.

Avant de passer à la suite, voyons à quoi devrait ressembler le signal modulé dont on souhaite extraire les informations. Un signal modulé simple à la formule suivante :

$$x(t) = A(t) \cdot \sin(2\pi ft + \varphi(t)) \text{ ou } A(t) \cdot \cos(2\pi ft + \varphi(t))$$

Où A est l'amplitude, f la fréquence d'excitation et φ la phase instantanée. Prenons donc les 2 exemples, voici ce que l'on obtient lorsque l'on multiplie notre $x(t)$ avec nos 2 porteuses.

Exemple avec $x(t) = A(t) \cdot \sin(2\pi ft + \varphi(t))$:

$$I(t) = A(t) \cdot \sin(2\pi ft + \varphi(t)) \cdot \cos(2\pi ft)$$

$$I(t) = \frac{A(t)}{2} [\sin((2\pi ft + \varphi(t)) + 2\pi ft) + \sin((2\pi ft + \varphi(t)) - 2\pi ft)]$$

$$I(t) = \frac{A(t)}{2} [\sin(4\pi ft + \varphi(t)) + \sin(\varphi(t))]$$

Filtrage (Généralement passe-bas) :

$$I(t) = \frac{A(t)}{2} \sin(\varphi(t))$$

$$Q(t) = A(t) \cdot \sin(2\pi ft + \varphi(t)) \cdot \sin(2\pi ft)$$

$$Q(t) = \frac{A(t)}{2} [\cos((2\pi ft + \varphi(t)) - 2\pi ft) - \cos((2\pi ft + \varphi(t)) + 2\pi ft)]$$

$$Q(t) = \frac{A(t)}{2} [\cos(\varphi(t)) - \cos(4\pi ft + \varphi(t))]$$

Filtrage (Généralement passe-bas) :

$$Q(t) = \frac{A(t)}{2} \cos(\varphi(t))$$

Exemple avec $x(t) = A(t) \cdot \cos(2\pi ft + \varphi(t))$:

$$I(t) = A(t) \cdot \cos(2\pi ft + \varphi(t)) \cdot \cos(2\pi ft)$$

$$I(t) = \frac{A(t)}{2} [\cos((2\pi ft + \varphi(t)) - 2\pi ft) + \cos((2\pi ft + \varphi(t)) + 2\pi ft)]$$

$$I(t) = \frac{A(t)}{2} [\cos(4\pi ft + \varphi(t)) + \cos(\varphi(t))]$$

Filtrage (Généralement passe-bas) :

$$I(t) = \frac{A(t)}{2} \cos(\varphi(t))$$

$$Q(t) = A(t) \cdot \cos(2\pi ft + \varphi(t)) \cdot \sin(2\pi ft)$$

$$Q(t) = \frac{A(t)}{2} [\sin((2\pi ft + \varphi(t)) + 2\pi ft) - \sin((2\pi ft + \varphi(t)) - 2\pi ft)]$$

$$Q(t) = \frac{A(t)}{2} [\sin(\varphi(t)) - \sin(4\pi ft + \varphi(t))]$$

Filtrage (Généralement passe-bas) :

$$Q(t) = \frac{A(t)}{2} \sin(\varphi(t))$$

Nous avons donc les résultats finaux suivants :

Signal d'entrée	$I_{filtré}(t)$	$Q_{filtré}(t)$
$A(t) \cdot \sin(2\pi ft + \varphi(t))$	$\frac{A(t)}{2} \sin(\varphi(t))$	$\frac{A(t)}{2} \cos(\varphi(t))$
$A(t) \cdot \cos(2\pi ft + \varphi(t))$	$\frac{A(t)}{2} \cos(\varphi(t))$	$\frac{A(t)}{2} \sin(\varphi(t))$

Tableau 4.1 – Résultat de la démodulation par quadrature selon le signal d'entrée

4.1.3 Signaux reçus du resolver

Le système que nous avons en place utilise un resolver, dans notre cas nous stimulons ce dispositif avec un signal sinusoïdal de 10 kHz (signal d'excitation), ce qui nous donne, aux bornes de notre RVDT, 2 secondaires qui nous génèrent des signaux AM étant tous 2 en phase avec notre signal d'excitation. Comme nous l'avons cité aux explications concernant la modulation I/Q, si notre signal modulé a une forme telle que " $A(t) \cdot \sin(2\pi ft + \varphi(t))$ " alors cela signifie que notre composante I est négligeable. Dans ce cas, seule la composante Q contient l'information utile.

Malheureusement, cela n'est pas aussi simple. En effet, notre système RVDT ressemble à la figure 4.7, ci-dessous.

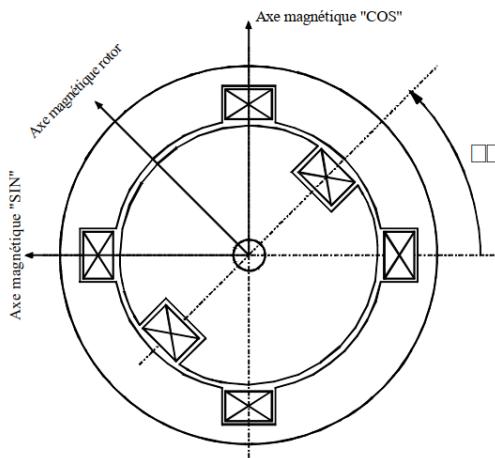


Figure 4.7 – Structure électromagnétique du resolver "brushless" [16]

Avec au milieu le rotor et tout autour le resolver. Lorsqu'un axe magnétique du rotor est aligné avec un axe du stator, la polarité de l'induction magnétique s'inverse, ce qui provoque une inversion de phase de 180° dans le signal mesuré. Ce phénomène est naturel et se répète à chaque demi-tour (π radians). Si l'on représente cela sous la forme d'un signal nous obtenons les signaux de la figure 4.8.

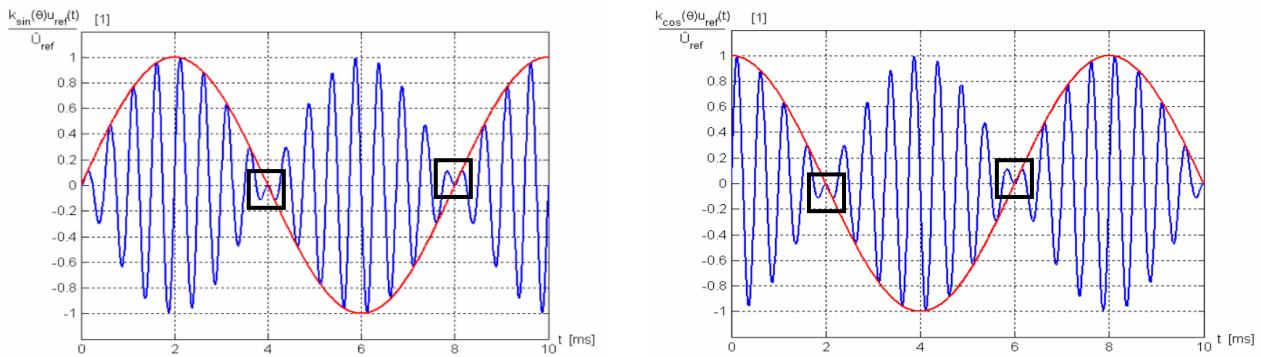


Figure 4.8 – Sorties sinus/cosinus et enveloppe [16]

Les signaux bleus de la figure 4.8 représentent les sorties des deux enroulements secondaires. On remarque que chaque fois que l'amplitude de notre signal atteint 0 (carré noir), donc quand l'axe magnétique du rotor est aligné aux axes du resolver, notre signal s'inverse. Donc approximativement tous les demi-tours ce phénomène se produit, ce qui est tout à fait normal et logique puisque les pôles magnétiques de notre rotor sont inversés.

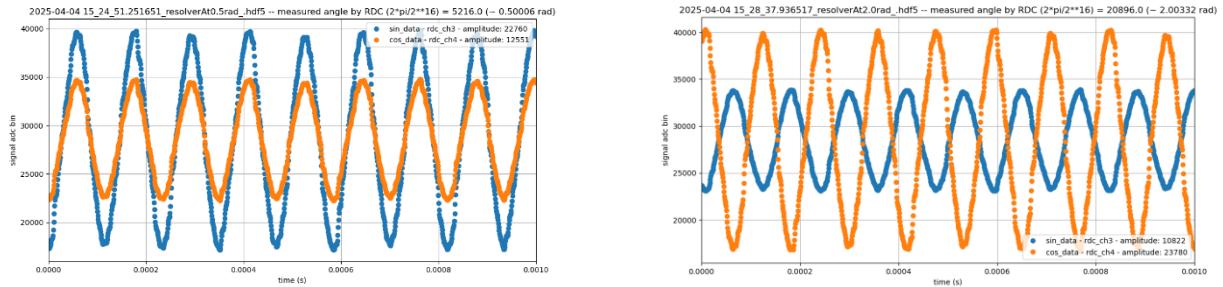


Figure 4.9 – Signaux sinus et cosinus capturés en sortie du resolver

C'est donc pour cette raison que dans les signaux fournis par le CERN (les signaux ont été pris à des angles fixes), comme ceux de la figure 4.9, ont la même phase à 0.5 radians (1^{er} quadrant) mais que le signal cosinus est inversé à 2.0 radians (2^{ème} quadrant) car nous avons passé l'axe cosinus. Cette alternance d'inversion s'explique par la disposition spatiale des enroulements du resolver, mécaniquement déphasés de 90°. À chaque transition de quadrant, l'un des deux signaux change de signe en raison du passage d'un pôle magnétique opposé, ce qui nous donne :

- Quadrant 1 à 2 : signal cosinus s'inverse
- Quadrant 2 à 3 : signal sinus s'inverse
- Quadrant 3 à 4 : signal cosinus s'inverse (phase original)
- Quadrant 4 à 1 : signal sinus s'inverse (phase original)

Cette propriété rend l'analyse directe des signaux I et Q difficile, car leur signe n'est pas constant. Une solution efficace consiste à calculer l'enveloppe du signal modulé à l'aide de la magnitude :

$$\text{Enveloppe} = \sqrt{I^2 + Q^2}$$

En effet effectuer une racine sur un système FPGA n'est pas une chose recommandée mais nous y reviendrons plus tard.

4.2 Fonctionnement CIC

Comme l'ont mis en évidence les calculs de démodulation I/Q, il est indispensable d'atténuer les hautes fréquences pour extraire correctement l'enveloppe du signal. Cette opération repose sur l'utilisation d'un filtre passe-bas, lequel peut être implanté dans un FPGA selon trois approches :

- FIR filter : Implémentation simple, mais devient complexe s'il nécessite une grande précision
- IIR Filter : Ne demande pas beaucoup de ressources, mais est plus complexe à mettre en œuvre sur FPGA
- CIC filter : Très léger (ne nécessite pas de multiplication), a de la peine pour les bruits à fréquence très élevée.

Compte tenu de la simplicité d'intégration recherchée, c'est le filtre CIC qui a été retenu.

Pourquoi un CIC :

Dans notre système FPGA, l'utilisation des ressources et la latence sont très importantes, ainsi les filtres CIC proposent 2 choses :

- Filtrer les bruits haute fréquence
- Réduire la fréquence d'échantillonnage par décimation

Le filtre CIC permet donc de faire 2 opérations simultanément, sans utiliser de multiplication, mais tout simplement des additions et des soustractions, ce qui est idéal pour une implémentation sur FPGA.

Structure d'un filtre CIC :

Un filtre CIC est composé de 2 blocs principaux :

- Des **intégrateurs**, généralement en cascade, qui agissent comme un filtre passe-bas, par accumulation des échantillons successifs.
- Des **combs** (peigne), aussi en cascade, après la décimation, jouant le rôle de filtres différentiateurs.

Voici un schéma de la structure d'un filtre CIC :

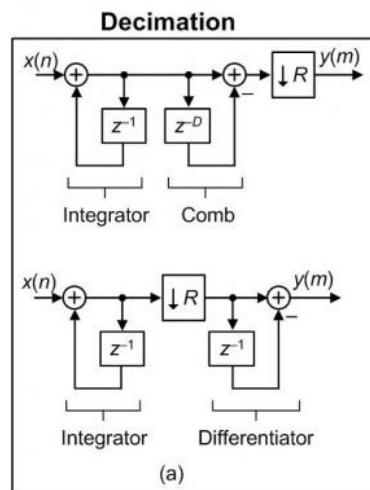


Figure 4.10 – Architecture d'un filtre CIC de décimation [17]

Et voici le schéma d'un CIC à plusieurs étages (ici 3 étages) :
 $\text{clock} = f_{\text{sample}}$

$$\text{clock} = f_{\text{sample}} / n$$

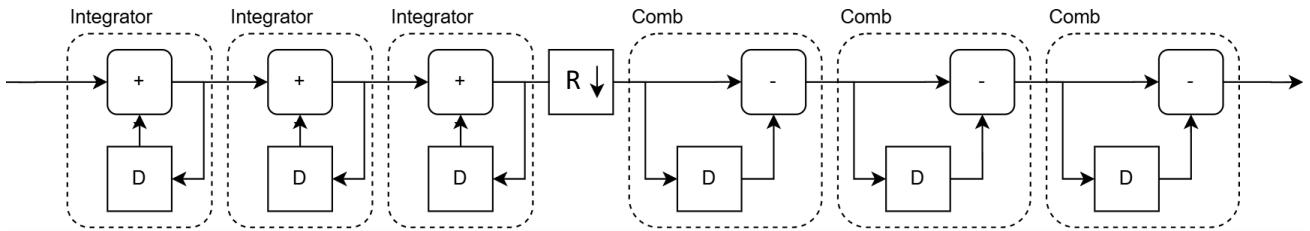


Figure 4.11 - Architecture complète d'un filtre CIC de décimation à 3 étages [18]

Décimation :

La décimation est l'opération qui consiste à réduire la fréquence d'échantillonnage du signal. Si le signal initial est échantillonné à une fréquence de $f_{\text{échantillonnage}}$, alors la fréquence après décimation devient :

$$f_D = \frac{f_{\text{échantillonnage}}}{R}$$

Cela permet d'alléger la bande passante et de réduire le volume de données à traiter après filtrage.

Fonction de transfert du filtre CIC :

La fonction de transfert en z (domaine pour représenter les signaux discrets), pour un filtre CIC avec :

- R : Le facteur de décimation
- N : Le nombre d'étages (aussi appelé ordre du filtre)
- M : Délai différentiel (souvent 1)

Est donnée par :

$$H_{\text{CIC}}(z) = \left(\frac{1-z^{-RM}}{1-z^{-1}}\right)^N = \left(\frac{1}{1-z^{-1}}\right)^N \cdot (1-z^{-RM})^N$$

Cette équation peut être interprétée comme :

- La partie $\left(\frac{1}{1-z^{-1}}\right)^N$ représente les N intégrateurs en cascade
- La partie $(1-z^{-RM})^N$ représente les N combs différentiateurs, espacés de RM échantillons.

Chaque partie a donc son rôle bien spécifique, notre intégrateur accumule chaque valeur de notre signal donc plus on avance dans notre signal plus on aura une grande valeur.

Nous aurions ensuite le déimateur qui réduit le nombre d'échantillons donc nous avons toujours la même forme de signal qu'après les intégrateurs mais avec moins d'échantillon.

Puis les étages de comb viendraient, quant à eux, faire l'exact opposé de nos intégrateurs en annulant la dérive puis restaurer notre signal de départ tout en supprimant le bruit.

Voici donc les résultats obtenus en prenant par exemple la sortie Q de notre secondaire 2 dans notre démodulation par quadrature :

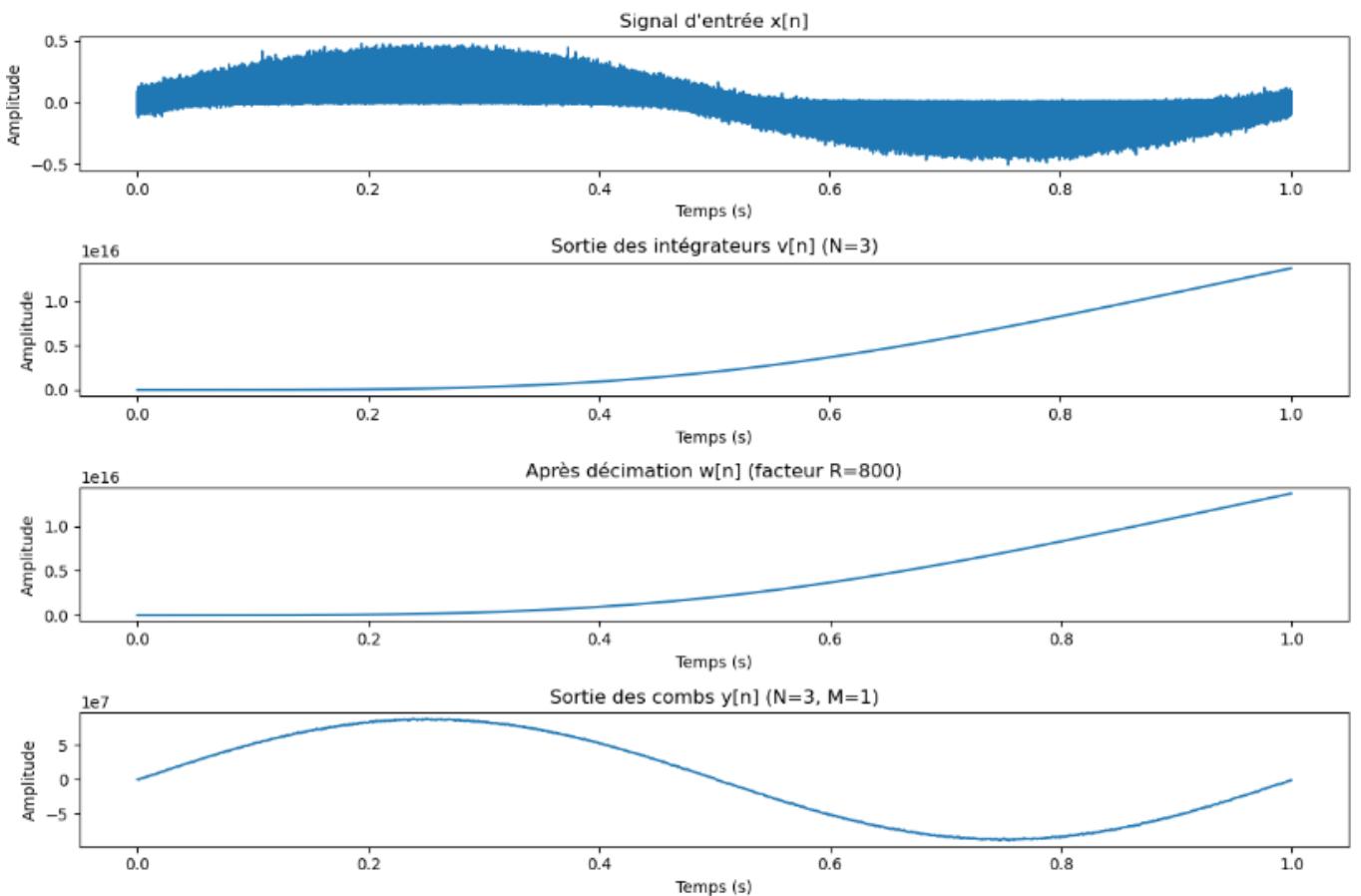


Figure 4.12 – Illustration du fonctionnement d'un filtre CIC de décimation

Le filtre CIC est donc une solution particulièrement adaptée pour des systèmes FPGA afin d'effectuer un filtrage passe-bas et une décimation de manière efficace, simple et économique. Cette partie s'appuie notamment sur les documents [17], [18], [19] et [20].

4.3 Contrainte resolver (correction d'ellipcicité/ Non-idéalités)

Un resolver se compose d'une bobine primaire excitée par un signal sinusoïdal et de 2 bobines secondaires déphasées mécaniquement de 90°. Lorsque le rotor bouge cela modifie la répartition du flux magnétique couplé aux secondaires, selon l'angle mécanique du rotor.

Ce qui fait donc que nous avons 2 signaux modulés en amplitude en fonction de l'angle mécanique du rotor et idéalement en phase fixe par rapport au signal d'excitation.

Mais, en pratique, plusieurs phénomènes peuvent provoquer des variations de phase. On appelle ces phénomènes les non-idéalités du resolver.

Non-Idéalité :

Malgré le fait que les resolver sont très performants, la construction physique du resolver implique nécessairement certaines imperfections mécaniques, telles que (voir [21]):

- **Excentricité mécanique du rotor :** Si le rotor est mal mis au centre, cela crée des déformations du couplage magnétique, selon la position mécanique du rotor, ce qui nous donne des offsets dans les signaux de sorties
- **Déséquilibre d'amplitude :** des inductances légèrement différentes dans les deux bobines secondaires entraînent un écart d'amplitude entre les canaux sin / cos.

- **Erreur de quadrature** : les bobines secondaires ne sont pas exactement déphasées mécaniquement de 90°, on obtient donc une erreur de phase constante.

Ces problèmes peuvent être réglés de différentes manières, soit en faisant en sorte que le montage mécanique soit fait de manière rigoureuse, soit en appliquant un algorithme appelé la correction d'ellipticité.

Correction d'ellipticité :

Lorsque les signaux sin et cos du resolver, censé former un cercle trigonométrique parfait (I, Q), sont affectés par des non-idealités, ils ne forment plus un cercle mais une ellipse.

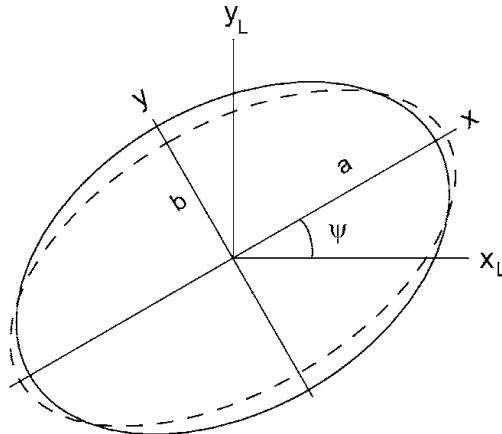


Figure 4.13 – Illustration d'une ellipse [22]

Comme l'illustre la figure 4.13, l'ellipse obtenue met en évidence les distorsions introduites par les imperfections du resolver. L'objectif idéal serait de ramener cette ellipse à un cercle parfait au moyen d'un algorithme de correction. Toutefois, la correction d'ellipticité ne sera pas implémentée dans le cadre de ce travail. Le sujet est complexe et dépasse le périmètre du projet. Il s'agit ici avant tout de signaler l'existence de ce problème.

Par ailleurs, ces défauts étant propres au resolver réellement installé, je ne dispose pas d'éléments permettant de confirmer qu'ils se manifestent effectivement sur notre capteur.

4.4 Algorithmes existants pour une implémentation sur FPGA

Le développement et la synthèse de l'ensemble du système seront réalisés avec l'outil Quartus Prime (Intel/Altera), tandis que la simulation fonctionnelle et temporelle sera effectuée sous QuestaSim. La cible visée impose des contraintes claires : l'horloge principale du FPGA doit fonctionner à 40 MHz et l'ADC fournit un taux d'échantillonnage de 10 MHz. Tous les blocs décrits ci-dessous (NCO, CIC, CORDIC, etc.) devront donc respecter cette marge de timing et de bande passante, ce qui influence directement les profondeurs de pipeline, la taille des registres internes et le choix éventuel d'options d'optimisation dans Quartus Prime.

Dans le cadre de ce projet, certaines fonctions critiques/complexes du traitement numérique peuvent être confiées à des blocs IP standards déjà disponibles dans les bibliothèques d'Intel/Altera. L'objectif de cette section est donc de présenter des blocs existants pour notre système.

- **NCO – Numerical Controlled Oscillator** (IP Intel/Altera) [23] : Générateur numérique de signaux sinusoïdaux (et cosinus) avec fréquence et phase paramétrable. Idéal pour générer localement des porteuses pour la démodulation I/Q. Ce bloc IP évite l'utilisation de tables LUT manuelles.
- **Filtre CIC – Cascaded Integrator-Comb** (IP Intel/Altera) [24]: Permet de filtrer les composantes I et Q après la démodulation, en supprimant les hautes fréquences issues de la multiplication avec les porteuses grâce à une structure basée uniquement sur des additions et des soustractions. Ces blocs sont hautement paramétrables (facteur de décimation, nombre d'étages, etc...)
- **Cordic – Coordinate Rotation Digital Computer** (IP Intel/Altera) [25]: Utilisé pour le calcul de l'angle à partir des composantes I/Q. Il permet un calcul efficace de l'angle du même style que atan2, sans avoir recours à des fonctions trigonométriques complexes.

Chapitre 5

Conception

Ce chapitre présente les schémas des différents blocs du système afin de calculer l'angle, mesuré par le resolver. Ces schémas ne sont pas exhaustifs de l'implémentation finale, mais permettent de comprendre le fonctionnement global et comment les signaux sont traités.

5.1 Système globale

La figure 5.1 présente le schéma global du système de traitement de notre resolver, afin d'en obtenir un angle. Les blocs de *Quadrature Demodulation* sont le cœur de notre projet, puisque ce sont eux qui nous permettront de récupérer l'information utile du signal à amplitude modulée en démodulant notre signal de la même manière que cela a été expliqué au [Chapitre 4](#). Par la suite nous allons utiliser un bloc IP *CORDIC* de chez Intel/Altera afin de calculer l'angle à l'aide de nos 2 signaux démodulé en utilisant la fonction *Atan2* du *CORDIC*, ce bloc sera expliqué plus en détail dans le chapitre suivant.

Dans notre système nous avons aussi à disposition un générateur de sinus, ce dernier est utilisé afin de générer notre signal d'excitation afin d'alimenter notre resolver. Ce bloc a été mis dans notre unité de traitement car nous devons connaître la fréquence d'excitation qui alimente notre resolver afin d'effectuer notre démodulation par quadrature. Le signal *phi_inc_i* est ce qui permet de définir la fréquence de notre signal d'excitation.

Et pour finir, un bloc de détection d'erreurs reproduisant un certain nombre d'erreurs disponibles dans le composant RDC (AD2S1210) qui est actuellement utilisé pour faire le calcul d'angle de notre resolver est mis en place. Ce bloc donne accès à des registres en lecture et/ou écriture afin de configurer les différents seuils et paramètres par défaut, nous permettant ainsi de garantir que les signaux traités n'ont pas de défaut. Ce bloc sera expliqué plus en détail dans la section correspondante.

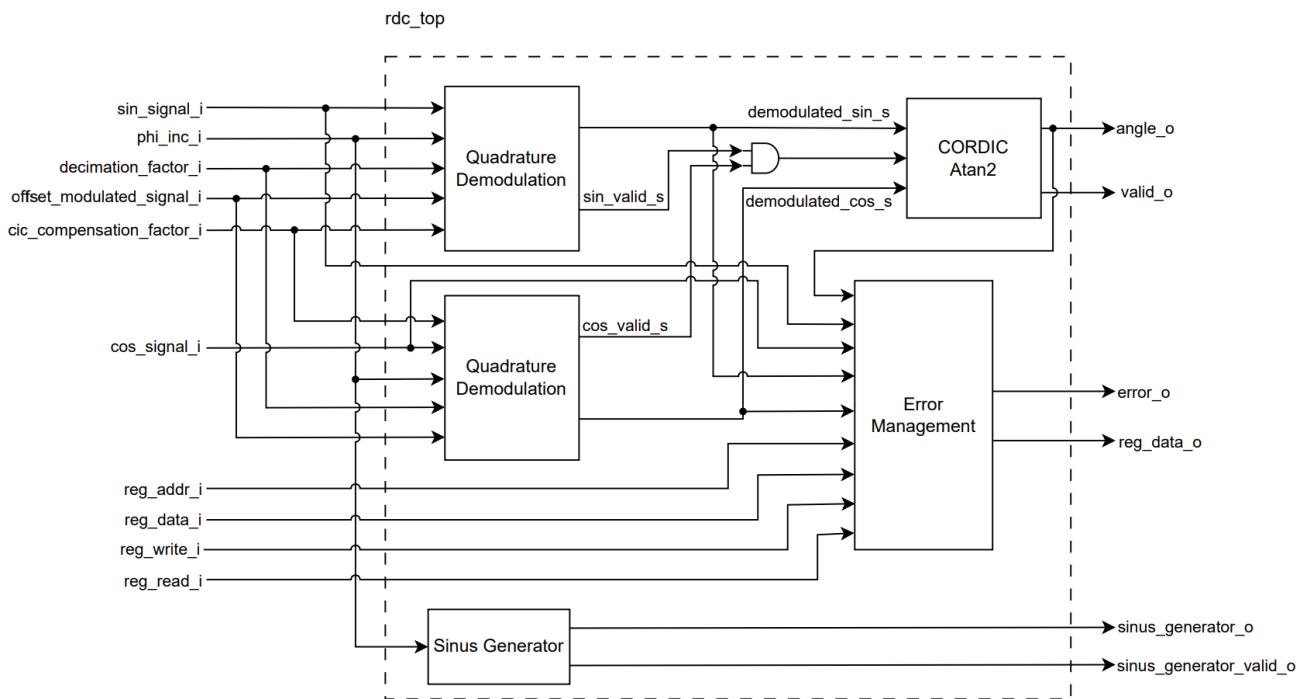


Figure 5.1 - Schéma du système complet

Il est important de noter que pour le schéma de la figure 5.1, ainsi que tous ceux qui vont suivre, chaque entrée/sortie ayant le même nom représente le même signal. Les noms de ces différents signaux sont ceux étant utilisés dans l'implémentation VHDL.

5.2 Démodulation par Quadrature

La démodulation en quadrature reprend l'essentiel des concepts présentés au chapitre 4. Comme le montre la figure 5.2, le système intègre un bloc Offset Correction, sa fonction est de recentrer le signal modulé d'entrée autour de zéro, afin de compenser le décalage continu introduit soit par les traitements effectués en amont, soit par les caractéristiques propres de l'ADC. Il est important de faire osciller notre signal autour de zéro, car sans cela notre démodulation par quadrature ne fonctionnera pas correctement, ceci est un pré-requis de la quadrature [26]. Nous pouvons aussi observer dans ce système que nous avons un bloc IP NCO (Numerically Controlled Oscillator) et CIC cité dans les chapitres précédents permettant de faire fonctionner notre démodulation par quadrature. Pour garantir un traitement fiable de tous les signaux, un indicateur de validité est véhiculé tout au long de la chaîne, jusqu'à la sortie du système global. Et pour finir nous avons en sortie notre bloc IP CORDIC utilisant la fonction vector translate, afin d'appliquer la formule finale à la démodulation par quadrature :

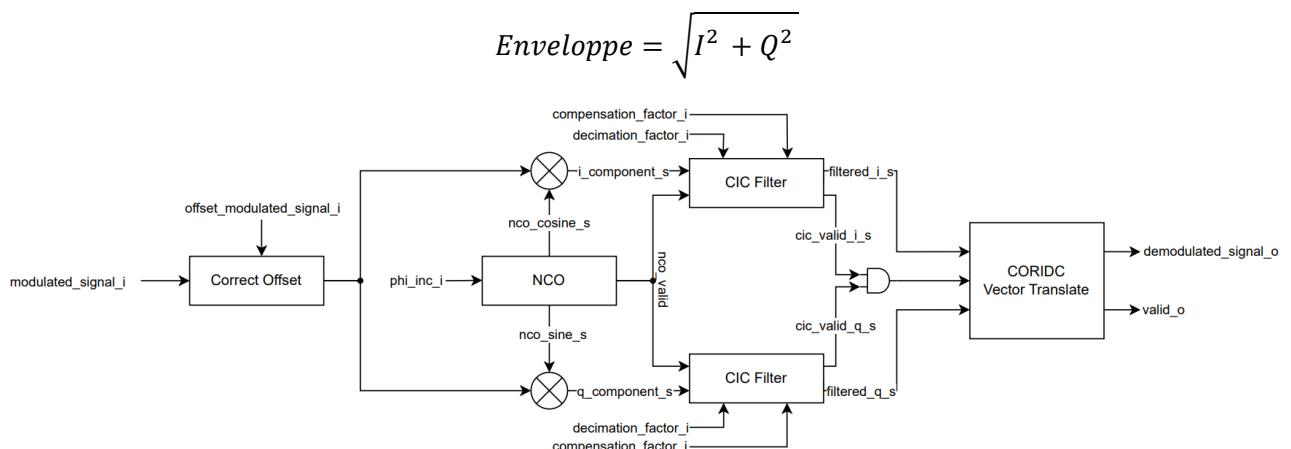


Figure 5.2 – Schéma du bloc Quadrature Demodulation

5.2.1 Filtre CIC

Lors du chapitre précédent, nous envisagions d'utiliser l'IP CIC d'Intel/Altera. Malheureusement n'ayant pas la licence pour utiliser ce bloc IP, notre propre filtre CIC a donc été développé. Pour ce faire, nous sommes partis du schéma de la figure 4.11 puis adaptés ce dernier pour correspondre à nos besoins. La figure 5.3 en présente l'architecture détaillée. Les blocs figurant en pointillé représentent les parties répétitives du filtre, leur nombre dépend des deux paramètres suivants du CIC :

- N : Le nombre d'étages (aussi appelé ordre du filtre)
- M : Délai différentiel (souvent 1)

Ces deux valeurs sont fixées à la synthèse (génériques VHDL), et déterminent directement la profondeur des accumulateurs ainsi que la quantité de logique utilisée. Le facteur R de décimation, lui, reste programmable à l'exécution via le port `decimation_factor_i`, d'où la présence du bloc "R ↓". Les pointillés indiquent donc qu'il faut répéter le motif autant de fois que le dictent N et M.

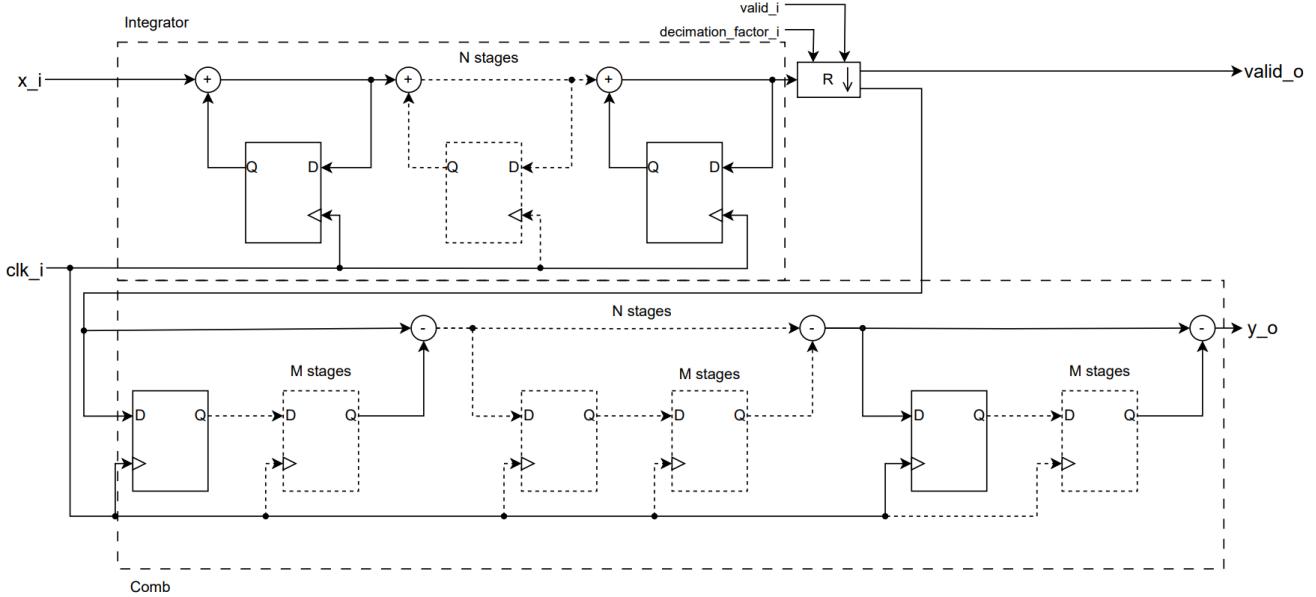


Figure 5.3 – Schéma du filtre CIC

5.3 Détection d'erreur

La détection d'erreur est une partie où la principale tâche est de faire des choix, car puisque l'on reprend la détection d'erreur du RDC actuellement en place dans le BWS, il n'utilise pas forcément les mêmes techniques de traitement que nous. Voici donc dans la figure 5.4, les différentes erreurs du RDC étant représentées par un "Fault Register".

Table 26. Fault Register Bit Descriptions

Bit	Description
D7	Sine/cosine inputs clipped
D6	Sine/cosine inputs below LOS threshold
D5	Sine/cosine inputs exceed DOS overrange threshold
D4	Sine/cosine inputs exceed DOS mismatch threshold
D3	Tracking error exceeds LOT threshold
D2	Velocity exceeds maximum tracking rate
D1	Phase error exceeds phase lock range
D0	Configuration parity error

Figure 5.4 – Fault register du RDC présent dans la datasheet du AD2S1210 [27]

Chacune des erreurs contrôle l'intégrité du système, voici donc de brèves explications expliquant le rôle de chacun dans le RDC [28]:

- D7 : Le convertisseur surveille en continu la tension instantanée des quatre broches $SIN\pm/COS\pm$, si l'une d'elles s'approche des rails d'alimentation ($\approx 0,15$ V ou AVDD – 0,2 V pendant ≥ 4 μ s), le bit est activé, pour signaler un écrêtage pouvant endommager l'entrée analogique.
- D6 : Lorsque la valeur maximale du moniteur passe sous la limite **LOS** (Loss Of Signal) programmée, le RDC considère qu'il y a une perte de signal.
- D5 : Lorsque la valeur maximale du moniteur dépasse la valeur maximale programmée, le RDC considère qu'il y a un **DOS** (Degradation Of Signal).
- D4 : Le RDC calcule en permanence la différence entre la valeur maximale et minimale du moniteur. Si cette différence dépasse le seuil **DOS mismatch**, alors nous avons un **DOS**.

- D3 : Le convertisseur suit la position à l'aide d'une tracking loop de type II, si l'erreur interne dépasse la limite haute **LOT** (Loss Of Tracking) ou si la vitesse ou l'accélération instantanée débordent la capacité de la boucle, le système déclare une **LOT**.
- D2 : Chaque résolution possède une vitesse de poursuite maximale. Si la vitesse mesurée franchit cette barrière, D2 passe à 1, cela protège la précision de l'angle en cas d'accélération brusque ou de sous-échantillonnage.
- D1 : Le RDC tolère par défaut $\pm 44^\circ$ de décalage entre l'excitation et les retours SIN/COS. Si la phase instantanée sort de cette fenêtre, le bit D1 s'active.
- D0 : Après chaque écriture, le périphérique stocke un bit de parité sur les 7 LSB du registre config. Si une lecture ultérieure révèle un désaccord, D0 est mis à 1 pour alerter d'une corruption de configuration.

Pour certaines de ces erreurs nous parlons d'un moniteur, ce dernier est une formule propre au RDC consistant à mesurer l'amplitude utile des signaux SIN/COS. Si cette valeur reste constante, cela signifie généralement que tout se passe bien. Voici la formule ci-dessous :

$$\text{Monitor} = A_1 \cdot \sin\theta \cdot \sin\phi + A_2 \cdot \cos\theta \cdot \cos\phi$$

Où :

A_1 est l'amplitude du signal sinus entrant ($A_1 \cdot \sin\theta$)

A_2 est l'amplitude du signal cosinus entrant ($A_2 \cdot \cos\theta$)

θ est l'angle du resolver

ϕ est l'angle mesuré par le RDC

Il est important de noter que la formule du moniteur est affichée après démodulation, avec le signal porteur $\sin\omega t$ supprimé, cela signifie que nos $A_1 \cdot \sin\theta$ et $A_2 \cdot \cos\theta$ représentent nos signaux démodulés.

En résumé, le moniteur reflète l'amplitude instantanée des signaux SIN/COS.

Maintenant que les fonctions associées à chaque code d'erreur ont été identifiées, nous pouvons déterminer lesquels ne seront pas utilisés dans notre système. Les indicateurs D3, D1 et D0 sont propres à l'implémentation du convertisseur AD2S1210 :

- D3 et D1 vérifient le bon verrouillage de la tracking loop interne. Comme notre architecture ne comporte pas de boucle de poursuite, ces deux indicateurs sont donc ignorés.
- D0 signale une erreur dans la mémoire interne du composant à l'aide de pin physique, il n'a pas d'équivalent dans notre conception.

L'erreur D2 indique si le système parvient à suivre la vitesse de rotation du moteur. Un code de test a été prévu, mais sa validation expérimentale étant complexe, ce bloc a été temporairement désactivé.

Le choix des erreurs à détecter étant désormais défini, la prochaine étape consiste à définir un plan d'adressage permettant de configurer les seuils et les paramètres par défaut. Le tableau 5.1 en récapitule la structure.

Register Name	Register Address	Register Data	Read/Write Register
Constant ID	0x0	D31 – D0	Read only
LOS Threshold	0x1	D13 – D0 (ADC Resolution)	Read/Write
DOS Overrange Threshold	0x2	D13 – D0 (ADC Resolution)	Read/Write
DOS Mismatch Threshold	0x3	D13 – D0 (ADC Resolution)	Read/Write
DOS Reset Max Threshold	0x4	D14 – D0 (ADC signed)	Read/Write
DOS Reset Min Threshold	0x5	D14 – D0 (ADC signed)	Read/Write
Error Enable Mask	0x6	D7 – D0	Read/Write
Error Clear	0x7	D7 – D0	Write only

Tableau 5.1 – Plan d'adressage du bloc de détection d'erreur

Voici une brève description de chacun des registres mis en place :

- **Constant ID** : Contient une valeur constante d'identification du module. Sert à vérifier l'intégrité de la communication ou à détecter la présence du bloc.
- **LOS Threshold** : Définit le seuil en dessous duquel le système considère que le signal d'entrée a une perte (Loss Of Signal).
- **DOS Overrange Threshold** : Seuil supérieur déclenchant une alerte de signal dégradé (Degradation Of Signal) en cas de dépassement.
- **DOS Mismatch Threshold** : Seuil de différence entre les pics max et min du signal, utilisé pour détecter une anomalie de forme d'onde.
- **DOS Reset Max Threshold** : Valeur maximale permettant de réinitialiser l'état du pics max, calculé, lorsqu'un clear de l'erreur est effectué
- **DOS Reset Min Threshold** : Valeur minimale permettant de réinitialiser l'état du pics min, calculé, lorsqu'un clear de l'erreur est effectué
- **Error Enable Mask** : Permet d'activer ou désactiver individuellement chaque détection d'erreur via un masque binaire.
- **Error Clear** : Permet de réinitialiser les bits d'erreur à 0. L'écriture d'un '1' sur une position efface l'erreur correspondante.

Le signal de sortie définissant quel est le bit d'erreur est défini dans un registre de 8 bits, où chaque bit correspond à une erreur représentée dans le tableau 5.2.

Bit	Description
D7	Réservé pour des utilisations future
D6	Réservé pour des utilisations future
D5	Réservé pour des utilisations future
D4	Réservé pour des utilisations future
D3	Les entrées SIN/COS dépassent le DOS mismatch threshold
D2	Les entrées SIN/COS dépassent le DOS overrange threshold
D1	Les entrées SIN/COS sont inférieurs au LOS threshold
D0	Les entrées SIN/COS sont saturées

Tableau 5.2 – Définition des bits du signal d'erreur de sortie

5.4 Générateur de sinus

La génération de sinus constitue la partie la plus simple. Il suffit d'instancier un NCO identique à celui utilisé pour la démodulation quadrature et de n'en exploiter que la sortie sinusoïdale, comme l'illustre la figure 5.5.

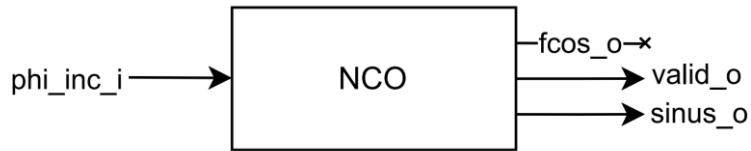


Figure 5.5 – Schéma du générateur de sinus

Une autre option aurait été de réutiliser la sortie sinus de notre générateur de sinusoïdes comme référence dans le bloc de démodulation quadrature. Cela aurait permis de réduire la logique du design en évitant un NCO supplémentaire. Cependant, cette approche rendrait le bloc quadrature moins modulable, car afin d'utiliser ce dernier nous devrions aussi instancier un bloc NCO, créant ainsi une forte dépendance à une configuration correcte du NCO pour assurer le bon fonctionnement du système. Le choix final s'est donc porté sur une instantiation NCO séparée, assurant ainsi une conception plus robuste et évolutive.

Chapitre 6

Implémentation

Ce chapitre présente en détail les différents choix d'implémentation qui ont été effectués tout au long du développement du projet, en particulier lors de l'écriture des descriptions matérielles en VHDL. Il ne s'agit pas uniquement de décrire les blocs fonctionnels, mais aussi d'expliquer les raisons techniques et pratiques qui ont motivé leur conception, ainsi que les compromis réalisés entre performance, précision et flexibilité.

Le système final se concentre autour de quatre grands composants principaux :

1. **La démodulation quadrature**, où les signaux sin et cos sont multipliés par les signaux issus du NCO, puis filtrés pour en extraire les composantes I et Q.
2. **Le filtre CIC**, qui agit comme un filtre passe-bas sans multiplication, est utilisé pour supprimer les composantes hautes fréquences issues du mélangeur, tout en réduisant la fréquence d'échantillonnage via la décimation.
3. **Le système de détection d'erreur**, chargé d'évaluer en temps réel l'intégrité du signal résolveur. Il surveille des indicateurs tels que la perte de signal (LOS), l'écrêtage (clipping), les écarts d'amplitude (DOS), et s'appuie notamment sur un moniteur calculé à partir des voies I et Q.
4. **Le système complet**, dans lequel les composants précédents sont interconnectés pour former une architecture fonctionnelle complète pour le traitement du signal. Cette architecture intègre également la conversion des signaux I/Q en angle via un bloc CORDIC, la gestion de la validité des données, ainsi que l'accès aux différents registres internes par une interface d'adressage simple.

Chaque section de ce chapitre reviendra sur les blocs VHDL développés, en détaillant la structure interne, les signaux utilisés, les génériques de paramétrage, les mécanismes de synchronisation et de validation, ainsi que l'intégration de chaque composant au sein du système global.

L'objectif de cette partie est également de mettre en évidence la cohérence de l'architecture retenue avec les exigences initiales du projet, traitement temps réel à haute fréquence (10 MSPS et système fonctionnant à 40 MHz), détection d'erreurs fiables, et modularité permettant une future adaptation à un capteur LVDT.

Enfin, quelques choix de conception alternatifs envisagés au cours du développement seront discutés, afin de justifier les solutions retenues et d'illustrer les compromis retenus pour atteindre les objectifs du projet.

6.1 Filtre CIC

Comme expliqué précédemment, malheureusement nous n'avons pas pu utiliser l'IP pré-fait d'Intel/Altera du filtre CIC car nous n'avions pas la licence. La réalisation du filtre CIC est très simple dans l'ensemble, car il suffit d'appliquer le schéma réalisé au chapitre 5, mais ce filtre CIC contient quelques subtilités qui sont expliquées en détail plus loin.

Nom	Description
DIVISION_RESOLUTION	Largeurs de bits de l'IP LPM_DIVIDE présent dans le CIC
DIVISION_LATENCY	Latence de l'IP LPM_DIVIDE
N	Nombre d'étages du filtre CIC, aussi appelé ordre du filtre
M	Délai différentiel du filtre CIC
R_MAX	Valeur maximale du facteur de décimation R du filtre CIC
WIDTH	Taille des valeurs d'entrée et de sortie souhaité du filtre

Tableau 6.1 - Paramètres générique du filtre CIC

Nom	Description
cic_compensation_factor_i	Facteur de compensation pour le filtre CIC (remplace le FIR)
decimation_factor_i	Facteur R défini pour le filtre CIC
valid_i	Entrée indiquant que x_i est valide
x_i	Signal à filtré
y_o	Signal filtré
valid_o	Sortie indiquant que y_o est valide

Tableau 6.2 - Paramètre d'entrée(i)/sortie(o) du filtre CIC

Afin de simplifier le code de notre filtre CIC chaque partie a sa propre entité, c'est-à-dire que nous avons une entité integrator et une entité comb. Cela permet d'utiliser l'instruction VHDL *generate* afin de générer autant de ces entités que le facteur N défini. Le facteur M quant à lui n'est utilisé que dans la partie comb afin de générer autant de flip-flop que l'on souhaite avoir de délai différentiel. Cela permet de déplacer les zéros (fréquence à laquelle la réponse du filtre est nulle) de la réponse en fréquence, ce qui peut être utile pour mieux rejeter certaines fréquences de repli (aliasing) après la décimation.

Notre filtre contient 2 facteurs de décimation, soit R_MAX et decimation_factor_i, la raison à cela est due au fait que notre système nécessite de connaître la taille que peuvent prendre les différents signaux du filtre CIC. Malheureusement la taille des signaux ne peut être définie que lors de la synthèse et ne peut pas être appliquée au run time, il est donc nécessaire de définir la valeur max que peut prendre notre facteur de décimation afin de pouvoir définir le facteur R souhaité au run time. Le même principe est appliqué dans l'IP CIC d'Intel/Altera.

6.1.1 Formule de Hogenauer

Dans un filtre CIC utilisé en décimation, les signaux peuvent atteindre des valeurs très élevées, en particulier dans la partie intégrateur, qui agit comme un accumulateur (moyenne glissante ou "moving average"). Cette accumulation peut rapidement provoquer une croissance importante de la largeur des données, surtout si le facteur de décimation R, le nombre d'étages N ou le délai différentiel M sont élevés.

Afin que le filtre CIC fonctionne correctement sans provoquer de débordement (overflow) ou d'utilisation excessive de ressources logiques, il existe plusieurs techniques pour gérer la croissance dynamique du signal. La plus répandue est la technique de Hogenauer [29].

Cette méthode consiste à réduire intelligemment la taille des registres internes du filtre en éliminant les bits qui ne contribuent pas de manière significative au résultat final, en particulier ceux en trop dus à l'accumulation. En s'appuyant sur le gain maximal théorique du filtre CIC, on peut déterminer combien de bits sont réellement nécessaires à chaque étage. Cela permet d'optimiser l'utilisation des ressources logiques (registres, multiplexeurs, connexions) tout en maintenant une précision suffisante pour garantir le bon fonctionnement du système.

Pour un filtre décimation, le gain maximum entre le premier étage et le dernier étage est :

$$G = (RM)^N$$

Si le nombre de bits du flux de données d'entrée est B_{in} , la croissance du registre peut être utilisée pour calculer B_{max} , le bit de poids fort à la sortie du filtre.

$$B_{max} = \lceil N \times \log_2(RM) + B_{in} - 1 \rceil$$

Comme mentionné précédemment, il est nécessaire de déterminer la taille maximale que peuvent atteindre nos signaux afin d'éviter tout dépassement de capacité dans le filtre CIC. Pour cela, le paramètre générique R_MAX a été introduit. Il garantit que, tant que la valeur de decimation_factor_i reste inférieure ou égale à R_MAX, le système fonctionnera correctement.

Étant donné que le calcul de Bmax (nombre de bits requis à la sortie du filtre) doit être connu à la compilation, cette valeur est définie comme une constante. Le Listing 6.1 présente l'implémentation de cette constante en VHDL, laquelle reflète directement la formule théorique du Bmax.

```
constant Bmax : integer := Bmax_calc(WIDTH, R_MAX, N, M);
```

Listing 6.1 – Déclaration de la constante B_{max}

La formule du gain est utilisée pour corriger l'amplification introduite par le filtre CIC. Pour compenser ce gain, il est nécessaire de diviser la sortie du filtre par la valeur du gain théorique. Deux approches peuvent être envisagées :

1. Utiliser un décalage logique (shift right) en calculant le \log_2 du gain. Cette méthode est très économique en ressources, mais elle introduit une perte de précision lorsque le gain n'est pas une puissance de deux.
2. Effectuer une division réelle afin d'obtenir un résultat plus précis, au prix d'une consommation plus importante de ressources logiques et d'une latence accrue.

Dans un premier temps, la solution basée sur le décalage logique avait été retenue, car elle offrait un bon compromis pour le calcul de l'angle corrigé, avec des performances satisfaisantes. Cependant, cette approche a dû être abandonnée lors de la mise en œuvre de la détection d'erreur. En effet, le moniteur de signal requiert une très grande précision pour détecter efficacement les défauts de signal et garantir que les signaux SIN/COS d'entrée sont valides.

C'est pourquoi un diviseur complet a été finalement implémenté. Cette solution assure une détection d'erreur fiable, sans imposer de traitement supplémentaire côté utilisateur, ni de prendre en compte les imprécisions liées au décalage binaire, lors des réglages des différents registres.

6.2 Démodulation Quadrature

La démodulation par quadrature constitue une brique essentielle du traitement des signaux issus d'un capteur RVDT ou LVDT. Son objectif est d'extraire les composantes I (in-phase) et Q (quadrature) du signal d'entrée modulé, afin de permettre la reconstitution de l'amplitude via une transformation vectorielle (CORDIC).

Le bloc de démodulation repose sur quatre éléments clés :

1. Un oscillateur numérique contrôlé en phase (NCO), générant les signaux de référence sinusoïdaux (sin et cos) à fréquence configurable.
2. Un multiplicateur, multipliant le signal d'entrée corrigé de son offset avec les références NCO.
3. Un filtrage passe-bas appliqué aux signaux I et Q via un filtre CIC.
4. Un module CORDIC permettant d'obtenir l'enveloppe finale de notre signal d'entrée

Puisque nous avons plusieurs éléments qui ne sont configurables qu'à la synthèse plusieurs valeurs sont définies en tant que génériques, voici donc 2 tableaux présentant les différents paramètres de notre système (le clock et le reset étant des paramètres évident ils ne seront pas explicités).

Nom	Description
<i>ADC_RESOLUTION</i>	Largeurs de bits de l'ADC fournissant les signaux du resolver
<i>NCO_RESOLUTION</i>	Largeurs de bits de l'IP NCO (Intel/Altera)
<i>VECTOR_TRANSLATE_RESOLUTION</i>	Largeurs de bits de l'IP CORDIC (Intel/Altera)
<i>DIVISION_RESOLUTION</i>	Largeurs de bits de l'IP LPM_DIVIDE (Intel/Altera) présent dans le CIC
<i>VECTOR_TRANSLATE_LATENCY</i>	Latence de l'IP CORDIC avec la fonction vector translate
<i>DIVISION_LATENCY</i>	Latence de l'IP LPM_DIVIDE
<i>R_MAX</i>	Valeur maximale du facteur de décimation R du filtre CIC
<i>N</i>	Nombre d'étages du filtre CIC
<i>M</i>	Délai différentiel du filtre CIC

Tableau 6.3 – Paramètres générique du bloc de démodulation par quadrature

Nom	Description
<i>offset_modulated_signal_i</i>	Permet de recentrer le signal modulated_signal_i autour de 0
<i>cic_compensation_factor_i</i>	Facteur de compensation pour le filtre CIC (remplace le FIR)
<i>decimation_factor_i</i>	Facteur R défini pour le filtre CIC
<i>phi_inc_i</i>	Paramètre définissant la fréquence du NCO
<i>modulated_signal_i</i>	Signal modulé à traiter
<i>demodulated_signal_o</i>	Signal d'entrée démodulé
<i>valid_o</i>	Sortie indiquant si demodulated_signal_o est valide

Tableau 6.4 – Paramètre d'entrée(i)/sortie(o) du bloc de démodulation par quadrature

Parmi ces différents paramètres, divers choix ont dû être effectués, commençons par définir ceux qui ont influencé les paramètres génériques. En effet, cela aurait pu être possible de définir plusieurs de ces paramètres comme constantes, mais ce choix fut écarté, car je souhaitais avoir un bloc le plus modulable possible et permettre de régler plus facilement le système lors de son intégration dans le code complet. Le paramètre générique VECTOR_TRANSLATE_LATENCY n'est pas non plus réglé comme constante car ce dernier est influencé par différents paramètres du système, tel que la fréquence d'horloge du système ou encore la latence souhaitée. La valeur du VECTOR_TRANSLATE_LATENCY peut être définie en se servant du rapport fourni par l'IP CORDIC, tel qu'on peut le voir dans la figure 6.1.

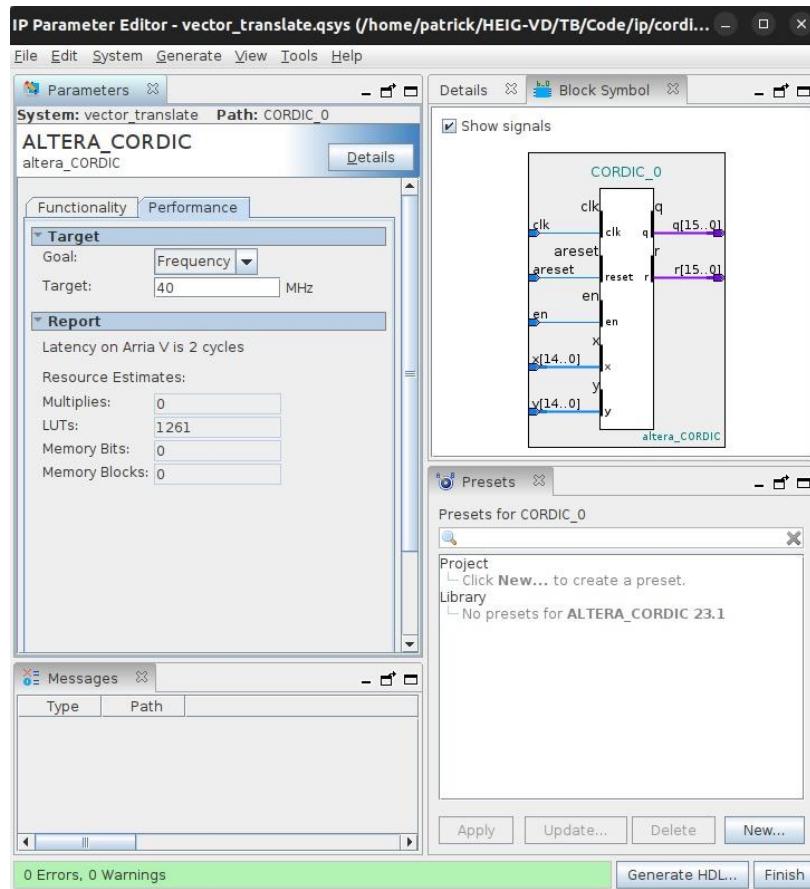


Figure 6.1 – IP parameter Editor du CORDIC pour la fonction vector translate

La fréquence de sortie du NCO (Numerically Controlled Oscillator) est définie par le signal `phi_inc_i`. En effet, ce signal représente l'incrément de phase appliquée à chaque cycle d'horloge au sein de l'accumulateur de phase du NCO. Plus la valeur de `phi_inc_i` est élevée, plus la phase progresse rapidement, ce qui se traduit par une fréquence de sortie plus élevée. La relation entre `phi_inc_i` et la fréquence de sortie est donnée par la formule :

$$f_{\text{out}} = \frac{\phi_{\text{inc_i}} * f_{\text{clk}}}{2^{\text{NCO_RESOLUTION}}} \text{ ou } \phi_{\text{inc_i}} = \frac{f_{\text{out}} * 2^{\text{NCO_RESOLUTION}}}{f_{\text{clk}}}$$

Où f_{clk} est la fréquence d'horloge du système et NCO_RESOLUTION est la résolution de l'IP NCO (16 bits dans notre cas). Cette approche permet de générer des signaux sinusoïdaux avec une fréquence programmable, simplement en ajustant la valeur de `phi_inc_i`, sans recourir à des oscillateurs analogiques externes. Ce principe est au cœur de la modulation et démodulation numérique dans notre système, notamment pour synchroniser le NCO avec le signal modulé d'entrée. Il est important de noter que malheureusement `phi_inc_i` ne peut pas être une valeur à virgule fixe ou flottante car l'IP ne prend pas cela en charge, si la valeur de la formule est donc 16.384 alors ce dernier sera arrondi à 16, ce qui nous donne en réalité une fréquence d'excitation de 9.766 kHz. L'IP NCO d'Intel/Altera fournit un calculateur automatique nous permettant de définir la valeur nécessaire de notre `phi_inc_i` en fonction de ces différents paramètres comme on peut le voir dans la figure 6.2, dans la section Generated Output Frequency Parameters, qui n'est qu'un simple calculateur intégré à l'IP.

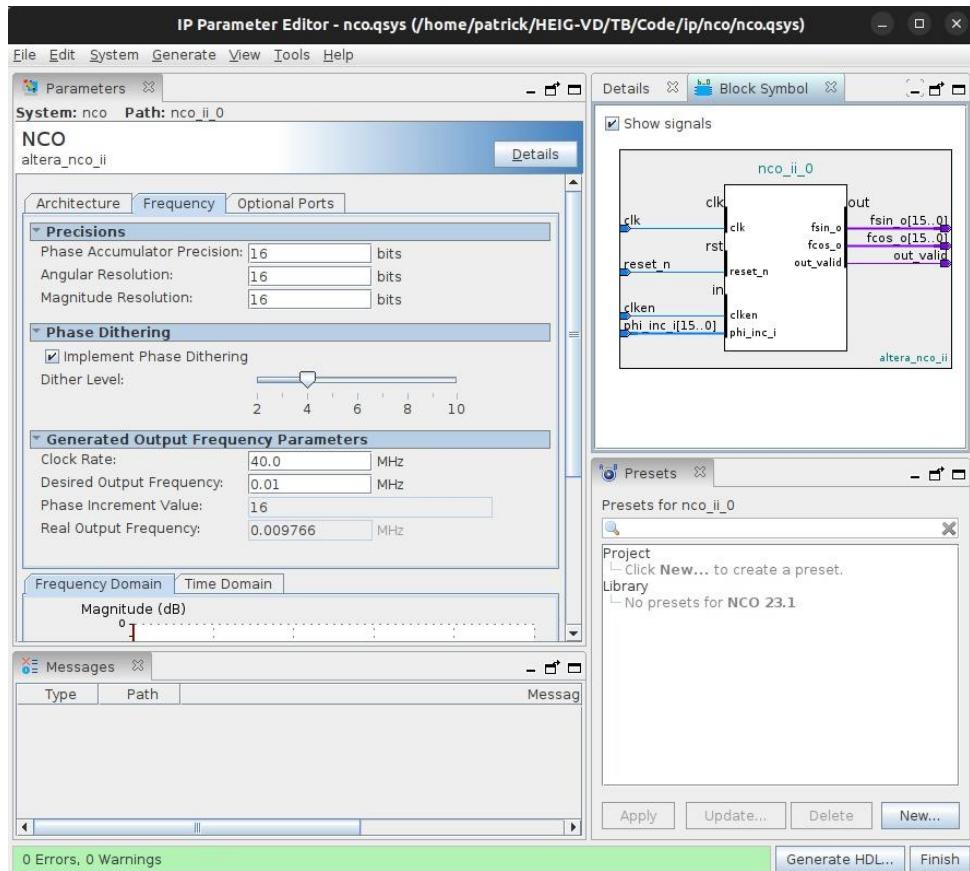


Figure 6.2 - IP parameter Editor du NCO

6.3 Calcul d'amplitude

L'implémentation du calcul de l'amplitude en fonction des composantes I et Q fut très simple à réaliser, car il s'agit ni plus ni moins que de configurer l'IP CORDIC correctement afin de réaliser l'opération suivante :

$$\text{Enveloppe} = \sqrt{I^2 + Q^2}$$

Cette opération est réalisée avec la fonction vector translate présente dans l'IP, et nous fournit en sortie l'amplitude et l'angle des 2 vecteurs en sortie. Dans notre cas l'angle ne nous intéresse pas et est donc une sortie ignorée.

Le calcul de l'amplitude de l'IP CORDIC a tout de même une petite spécialité. En réalité, la formule du calcul de l'amplitude de l'IP a la formule suivante :

$$\text{Enveloppe} = K(x^2 + y^2)^{0.5}$$

Où K représente une constante spécifique au gain cumulatif causé par les itérations du CORDIC, cette constante a pour valeur 1.646760258121. Afin d'éviter toute division, cette valeur a été inversée afin d'avoir la valeur $1/K \approx 0.607252935$, ce qui nous permet de faire une simple multiplication de cette constante inversée sur la valeur de sortie de notre IP CORDIC. En effet, on pourra observer dans le code VHDL que cette constante a été définie de telle sorte à avoir le code du Listing 6.2. Cette constante est donc multipliée par notre amplitude de sortie, puis shift à droite de 16 bits afin de corriger cette multiplication. Cela aurait pu être simplifié en utilisant les paquets ieee.fixed_pkg (QuestaSim/ModelSim) et ieee_proposed.fixed_pkg (Quartus), qui auraient pu faciliter de nombreux calculs dans les différents blocs. Malheureusement ayant découvert ce paquet un peu tard dans le projet, le code n'utilise pas les virgule fixe comme type de variables directement.

```
constant CORDIC_SCALE_FACTOR : unsigned(15 downto 0) := to_unsigned(39797, 16); --
0.6072529350088 * 2^16
```

Listing 6.2 – Déclaration de la constante définie de l'IP CORDIC

Un défaut de la méthode de calcul de l'amplitude est qu'elle élimine toutes les composantes négatives du signal, ce qui peut poser des problèmes lors du calcul ultérieur de l'angle. Pour pallier ce problème, nous exploitons une caractéristique spécifique de notre système.

Dans le chapitre 4, nous avons vu que la composante Q est en phase avec un signal sinusoïdal lorsque la composante I est faible, et inversement (voir page 18). Comme notre signal d'excitation est purement sinusoïdal, seule la composante Q porte réellement l'information, la composante I restant proche de zéro. En tirant parti de cette particularité, il serait possible de se passer entièrement du calcul de la composante I, du CORDIC, de la multiplication et du filtrage CIC. Toutefois, cela rendrait notre démodulation très spécifique à ce système, moins modulable et donc moins évolutive.

Ainsi, pour récupérer les informations de phase manquantes, nous utilisons le bit de poids fort (signe) de la composante Q pour signer l'amplitude calculée. Concrètement, si ce bit indique une valeur négative, l'amplitude est multipliée par -1 pour reconstituer la version signée du signal. Un autre moyen de réaliser cela pour notre système serait d'utiliser la sortie Atan2 du CORDIC pour identifier le quadrant. Si l'angle se situe dans les quadrants III ou IV du cercle trigonométrique, nous considérons l'amplitude comme négative.

Si l'excitation avait été un cosinus au lieu d'un sinus, le même principe s'appliquerait, mais nous aurions récupéré le signe de I, ou nous aurions défini les quadrants II et III comme négatifs.

Malheureusement cette correction de signe fonctionne si l'excitation et le mode de démodulation sont connus à l'avance. Pour un système plus flexible pouvant recevoir tout type d'excitation (sinus, cosinus, déphasé...), il serait nécessaire d'introduire :

- Un paramètre d'entrée précisant la nature de l'excitation (sinus ou cosinus).
- Une entrée de phase de référence, pour suivre dynamiquement son déphasage.
- Une LUT de correction des quadrants paramétrable, ajustable à chaque configuration.

Cela permettrait de généraliser la reconstruction du signe en fonction des conditions réelles du système, sans perte de robustesse ni de précision.

6.4 Détection d'erreur

Comme expliqué au chapitre 5 la majorité de la logique implémentée dans la détection d'erreur est basée sur les mécanismes du convertisseur AD2S1210, notamment à travers l'utilisation du signal monitor pour détecter plusieurs catégories de défauts.

Nom	Description
<i>CLK_FREQUENCY</i>	Fréquence de l'horloge système
<i>ADC_RESOLUTION</i>	Largeurs de bits de l'ADC fournissant les signaux du resolver
<i>VECTOR_TRANSLATE_RESOLUTION</i>	Largeurs de bits de l'IP CORDIC (Intel/Altera) (vector translate function)
<i>SINCOS_RESOLUTION</i>	Largeurs de bits de l'IP CORDIC (Intel/Altera) (sincos function)
<i>ATAN2_RESOLUTION</i>	Largeurs de bits de l'IP CORDIC (Intel/Altera) (Atan2 function)
<i>ADDR_WIDTH</i>	Largeur de bits pour le bus d'adresse
<i>DATA_WIDTH</i>	Largeur de bits pour le bus de donnée

Tableau 6.5 - Paramètres générique de la détection d'erreur

Nom	Description
<i>sin_signal_i</i>	Signal sinus à évaluer
<i>cos_signal_i</i>	Signal cosinus à évaluer
<i>angle_i</i>	Angle calculé du résolveur
<i>valid_i</i>	Entrée indiquant qu' <i>angle_i</i> est valide
<i>delayed_sin_i</i>	Signal sin démodulé retardé
<i>delayed_cos_i</i>	Signal cos démodulé retardé
<i>first_data_received_i</i>	Première donnée valide reçue de notre système (angle)
<i>cic_settled_i</i>	Flag indiquant que le filtre CIC est stable
<i>reg_addr_i</i>	Adresse du registre auquel on souhaite accéder
<i>reg_data_i</i>	Donnée à écrire dans le registre
<i>reg_data_o</i>	Donnée lue du registre
<i>reg_write_i</i>	Enable l'écriture
<i>reg_read_i</i>	Enable la lecture
<i>error_o</i>	Sortie de la détection d'erreur

Tableau 6.6 - Paramètre d'entrée(i)/sortie(o) de notre système complet

Plusieurs points essentiels ont été pris en compte lors de l'implémentation de notre détection d'erreur. La première concerne notre algorithme pour calculer le monitor, ci-dessous :

$$\text{Monitor} = A_1 \times \sin\theta \times \sin\phi + A_2 \times \cos\theta \times \cos\phi$$

Les blocs CORDIC génèrent une certaine latence pipeline (le nombre d'itérations ou l'architecture en pipeline des blocs atan2 et sincos). Si l'on multipliait directement les sorties actuelles de nos signaux démodulée avec les sorties sin/cos, le produit serait décalé dans le temps et incorrect.

Pour compenser cette latence, un signal retardé des sorties démodulé est introduit. Un processus spécifique synchronise les données démodulées avec leur cosinus/sinus respectif. Cela garantit que notre monitor est correctement calculé à un instant donné, correspondant à la même étape de pipeline des CORDIC. Cela nous donne donc une formule du moniteur correspondant au Listing 6.3.

```
product_sin := signed(delayed_demodulated_sin_i) * signed(sin_angle_s);
product_cos := signed(delayed_demodulated_cos_i) * signed(cos_angle_s);
scaled_product_sin := resize(shift_right(product_sin, 13), MONITOR_WIDTH);
scaled_product_cos := resize(shift_right(product_cos, 13), MONITOR_WIDTH);
```

Listing 6.3 – Calcul du monitor en VHDL

Le deuxième point concerne le moment où les données deviennent réellement exploitables. Les premières valeurs valides de l'angle n'apparaissent jamais à l'instant $t = 0$, il faut attendre que toute la chaîne du signal (CORDIC, CIC, etc.) soit opérationnelle. En particulier, un filtre CIC nécessite un certain temps de stabilisation après la mise sous tension pour éliminer les transitoires et délivrer des valeurs fiables.

Pour garantir que les erreurs relevées correspondent à un état du système entièrement opérationnel et stable, nous avons introduit :

- Un signal *first_data_received*, activé dès que la sortie de l'angle est devenue valide pour la première fois.
- Un flag *cic_settled*, qui s'active uniquement après que le filtre CIC a franchi la durée de stabilisation (calculée en fonction du facteur de décimation $\cdot R$, du nombre d'étages $\cdot N$, etc...)

- Une condition logique, issue de la combinaison de ces deux indicateurs, garantit que seuls les échantillons provenant d'une chaîne synchronisée (premier angle valide et filtre stabilisé) peuvent déclencher les mécanismes de détection d'erreur.

Cette stratégie permet d'éviter les faux positifs lors de la phase de démarrage et garantit que seules des conditions de fonctionnement valides déclenchent les indicateurs d'erreur.

Le dernier point concerne la fenêtre d'observation utilisée pour mesurer les valeurs minimale et maximale du signal "monitor". Dans l'AD2S1210, la détection d'erreur (LOS/DOS) ne s'effectue que sur une durée correspondant à la moitié de la période du signal d'excitation, afin de garantir une observation représentative du comportement du système.

La datasheet (Tableau 5) donne les plages de fréquences d'excitation (2 kHz–20 kHz) avec le nombre de cycles internes et la fenêtre correspondante pour un CLKIN = 8,192 MHz.

Table 5. Window Counter Period vs. Excitation Frequency Range, CLKIN = 8.192 MHz

Excitation Frequency Range	Number of Internal Clock Cycles	Window Counter Period (μs) ¹
2 kHz ≤ Exc Freq < 4 kHz	1065	260
4 kHz ≤ Exc Freq < 8 kHz	554	135.25
8 kHz ≤ Exc Freq ≤ 20 kHz	256	62.5

¹ CLKIN = 8.192 MHz. The window counter period scales with clock frequency and can be calculated by multiplying the number of internal clock cycles by the period of the internal clock frequency, that is, CLKIN/2.

Figure 6.3 – Tableau représentant la taille des fenêtres d'échantillon en fonction de la fréquence d'excitation [27]

Cette période garantit que la mesure couvre au moins une demi-période d'excitation. Pour une fréquence d'excitation de 10 kHz, la fenêtre vaut donc 62,5 μs. Dans notre système, l'horloge fonctionne à 40 MHz et non à 8,192 MHz. Il faut donc adapter le nombre de cycles internes à observer :

$$nb_{cycle} = \frac{WINDOW_PERIOD}{T_{horloge}} = \frac{62.5\mu s}{25ns} = 2500 \text{ cycles}$$

C'est l'objectif du Listing 6.4 : il recalcule dynamiquement le nombre de cycles requis pour refléter correctement la fenêtre de 62,5 μs, quel que soit le taux d'horloge utilisé.

```
constant WINDOW_SAMPLES_PERIOD      : real := 62.5e-6;
constant WINDOW_SAMPLES            : integer := integer(real(CLK_FREQUENCY) * WIN-
DOW_SAMPLES_PERIOD);
```

Listing 6.4 – Formule calculant le nombre de cycles d'horloge interne

En résumé, cette méthode garantit une détection d'erreur fiable, rapide et cohérente avec la logique du convertisseur AD2S1210, mais adaptée à notre environnement FPGA à 40 MHz.

6.5 Système complet

L'entité rdc_top représente l'intégration finale de tous les blocs fonctionnels précédemment conçus. Ce bloc centralise la réception des signaux sinusoïdaux issus du resolver, leur démodulation, le calcul d'angle, la détection d'erreurs ainsi que la génération d'un signal d'excitation sinusoïdal. Il constitue ainsi l'élément principal du traitement du signal dans notre système.

Le système prend en entrée deux signaux modulés (`sin_signal_i` et `cos_signal_i`) représentant la sortie analogique numérisée du resolver. Ces deux signaux sont traités indépendamment par deux blocs `quadrature_de-mod`. Ces modules réalisent la démodulation quadrature pour extraire l'amplitude de nos signaux modulés.

Un mécanisme de suivi de validité est mis en place pour prendre en compte la latence du pipeline du CORDIC. Cela permet de garantir que les valeurs traitées sont cohérentes et stables, évitant ainsi toute erreur liée à une désynchronisation.

Les sorties démodulées sont ensuite utilisées dans un bloc `atan2` (implémenté avec un CORDIC) afin de calculer l'angle instantané du rotor. L'IP `atan2` retourne une valeur dans la plage $[-\pi, +\pi]$. Pour l'étendre à $[0, 2\pi]$, nous appliquons un phase unwrapping inspiré d'un algorithme communément utilisé [30]. Si la différence entre deux angles successifs dépasse π , on soustrait 2π . Inversément, si elle est inférieure à $-\pi$, on ajoute 2π .

Le tout est surveillé par notre bloc d'erreur qui regroupe quant à lui toutes les fonctions de surveillance telles que la perte de signal (LOS), les detections de survaleurs, ou encore la discordance entre les signaux démodulé. Il utilise pour cela les signaux démodulés et l'angle calculé. Un système de registre mémoire (accessible en lecture/écriture) permet de configurer dynamiquement les seuils de détection d'erreur et de lire l'état des fautes détectées.

Enfin, un générateur de signal sinusoïdal (NCO) indépendant est également intégré, permettant de générer un signal d'excitation à la fréquence désirée. Ce générateur est séparé de la partie démodulation afin d'assurer une meilleure modularité du système.

Nom	Description
<code>CLK_FREQUENCY</code>	Fréquence de l'horloge système
<code>ADC_RESOLUTION</code>	Largeurs de bits de l'ADC fournissant les signaux du resolver
<code>NCO_RESOLUTION</code>	Largeurs de bits de l'IP NCO (Intel/Altera)
<code>VECTOR_TRANSLATE_RESOLUTION</code>	Largeurs de bits de l'IP CORDIC (Intel/Altera) (vector translate function)
<code>SINCOS_RESOLUTION</code>	Largeurs de bits de l'IP CORDIC (Intel/Altera) (sincos function)
<code>ATAN2_RESOLUTION</code>	Largeurs de bits de l'IP CORDIC (Intel/Altera) (Atan2 function)
<code>DIVISION_RESOLUTION</code>	Largeurs de bits de l'IP LPM_DIVIDE (Intel/Altera) présent dans le CIC
<code>ATAN2_LATENCY</code>	Latence de l'IP CORDIC avec la fonction sincos
<code>SINCOS_LATENCY</code>	Latence de l'IP CORDIC avec la fonction Atan2
<code>VECTOR_TRANSLATE_LATENCY</code>	Latence de l'IP CORDIC avec la fonction vector translate
<code>DIVISION_LATENCY</code>	Latence de l'IP LPM_DIVIDE
<code>OUTPUT_WIDTH</code>	Largeurs de bits de sortie souhaitée (doit être supérieur à <code>ATAN2_RESOLUTION</code>)
<code>R_MAX</code>	Valeur maximale du facteur de décimation R du filtre CIC
<code>N</code>	Nombre d'étages du filtre CIC
<code>M</code>	Délai différentiel du filtre CIC
<code>ADDR_WIDTH</code>	Largeur de bits pour le bus d'adresse
<code>DATA_WIDTH</code>	Largeur de bits pour le bus de donnée

Tableau 6.7 - Paramètres générique de notre système complet

Nom	Description
<i>offset_modulated_signal_i</i>	Permet de recentrer le signal modulated_signal_i autour de 0
<i>cic_compensation_factor_i</i>	Facteur de compensation pour le filtre CIC (remplace le FIR)
<i>decimation_factor_i</i>	Facteur R défini pour le filtre CIC
<i>phi_inc_i</i>	Paramètre définissant la fréquence du NCO
<i>sin_signal_i</i>	Signal sinus à traiter
<i>cos_signal_i</i>	Signal cosinus à traiter
<i>angle_o</i>	Angle calculé du resolver
<i>error_o</i>	Sortie de la détection d'erreur
<i>valid_o</i>	Sortie indiquant si angle_o est valide
<i>reg_addr_i</i>	Adresse du registre auquel on souhaite accéder
<i>reg_data_i</i>	Donnée à écrire dans le registre
<i>reg_data_o</i>	Donnée lue du registre
<i>reg_write_i</i>	Enable l'écriture
<i>reg_read_i</i>	Enable la lecture
<i>sinus_generator_o</i>	Signal d'excitation pour le resolver
<i>sinus_generator_valid_o</i>	Sortie indiquant si sinus_generator_o est valide

Tableau 6.8 - Paramètre d'entrée(i)/sortie(o) de notre système complet

6.6 Problème rencontré

Les filtres CIC sont très efficaces en tant que filtre passe-bas grâce à leur structure basée uniquement sur des additions et des délais. Toutefois, ils présentent deux inconvénients majeurs :

- Une amplification décroissante (droop) dans la bande utile.
- Une large zone de transition, ce qui rend difficile l'obtention d'une bande passante nette et plate

Pour pallier ces défauts, on ajoute un filtre FIR de compensation en cascade avec le CIC. Ce filtre, parfois appelé inverse-sinc, compense la chute de gain du CIC dans la bande passante et resserre la zone de transition.

En l'absence de compensation, la sortie du CIC ne conserve pas l'amplitude exacte du signal d'entrée. Ce n'est pas un défaut de calcul (du gain), la structure même du CIC crée une atténuation progressive dans sa bande passante. Par exemple, avec $N = 9$, $R = 8$, $M = 1$, la réponse en fréquence (figure 6.4) montre une atténuation dans la bande utile, caractéristique du "droop" associé au filtre CIC.

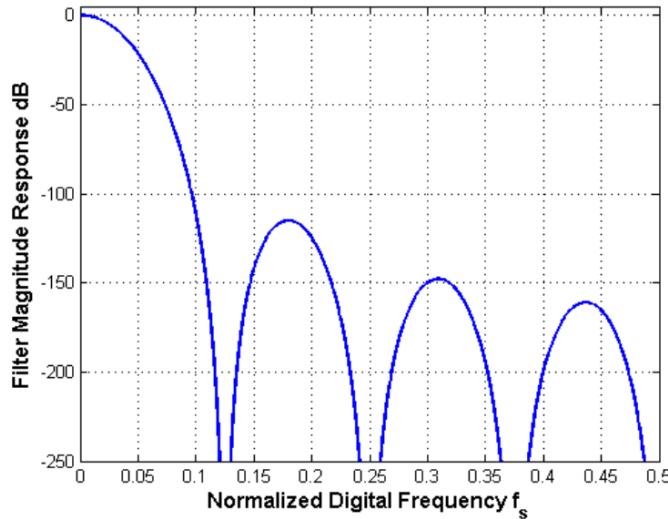


Figure 6.4 - Réponse en amplitude d'un filtre CIC avec $N = 9$, $R = 8$ et $M = 1$ [31]

Le rôle du filtre FIR est conçu pour avoir une réponse en fréquence représentant l'inverse de celle du CIC dans la bande utile, ce qui permet de :

- Corriger le droop afin que l'amplitude de sortie retrouve celle du signal d'entrée.
- Réduire la largeur de la zone de transition entre bande passante et bande d'arrêt, ce qui limite les aliasings et améliore l'intégrité du signal.

La figure 6.5 montre que la courbe verte (FIR de compensation) compense la chute de gain du CIC (courbe bleue), et la courbe rouge, représentant la réponse totale, retrouve une bande passante bien plus plate.

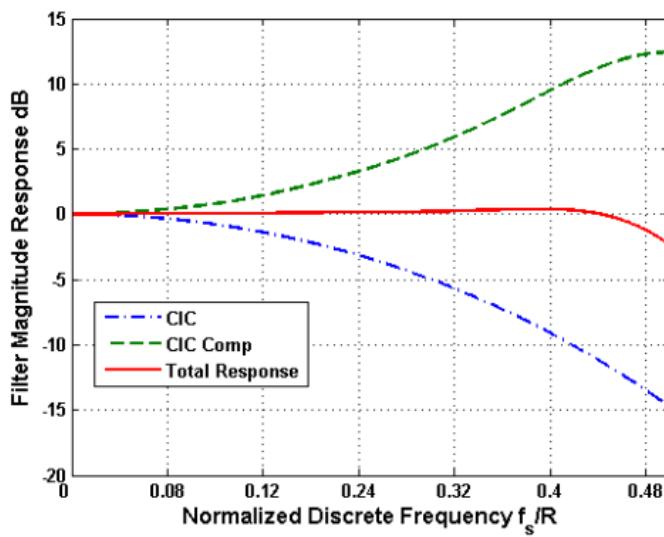


Figure 6.5 - Réponse du filtre de compensation pour un filtre CIC à 4 étages, tracée sur f_S/R [31]

Une mauvaise configuration du filtre FIR (mauvais choix de coefficients, de bande passante ou d'ordre) pourrait toutefois fausser la compensation, créant une surcompensation ou une atténuation résiduelle. Il est donc essentiel de bien dimensionner ce filtre. Des ressources comme la note d'application Intel AN455 [31] ou l'outil interactif gibbard.me [20] permettent de comprendre plus en détail la nécessité de ce filtre de compensation.

Malheureusement, je n'ai pas réussi à intégrer le filtre FIR de compensation à temps dans notre système, car sa configuration correcte s'est révélée plus complexe que prévu. Malgré de multiples essais basés sur la documentation de conception MATLAB (outil recommandé par Intel pour générer les coefficients) [32], [33], [34], la compensation n'a pas fonctionné comme prévu :

- La polarité du signal corrigé était inversée.
- L'amplitude contient toujours une perte similaire à celle créée par le CIC.

Pour pallier provisoirement l'absence du filtre FIR de compensation, un paramètre d'entrée (appelé cic_compensation_factor) a été ajouté, ce dernier multiplie la sortie du CIC par un facteur correcteur.

Aucune formule simple ne permet de calculer ce facteur, la seule méthode fiable consiste à :

1. Fixer les paramètres R, N et M.
2. Lancer une simulation (ModelSim, QuestaSim, etc...)
3. Mesurer l'amplitude maximale en entrée du CIC, puis celle en sortie.
4. Déterminer le cic_compensation_factor = Amplitude entrée / Amplitude sortie.

Malheureusement, je n'ai pas trouvé d'équation suffisamment précise pour éviter cette étape de simulation.

6.7 Synthèse summary

Pour valider les descriptions VHDL, une synthèse est réalisée sous Quartus Prime. Afin d'obtenir une analyse temporelle fiable, une entité "wrapper" a été créée qui instancie l'entité principale tout en reproduisant exactement ses ports d'entrée et de sortie. Chaque signal externe est d'abord échantillonné dans un registre d'entrée, puis renvoyé vers un registre de sortie après traitement. Ainsi, tous les chemins critiques considérés par l'outil se font de registre à registre, et non entre une broche d'E/S et la logique interne, ce qui garantit une estimation temporelle représentative du fonctionnement réel du circuit lors d'une intégration dans un autre design.

La synthèse a été réalisée pour un FPGA de la famille Arria V, modèle 5AGXMB1G4F40C4. Le tableau 6.9 présente les ressources occupées après compilation. La dernière colonne indique, pour chaque catégorie, le pourcentage d'utilisation globale actuel du projet dans lequel le générateur devra être intégré.

Type	Nombre	Pourcentage	Actuellement utilisé
Modules logique (en ALMS)	13'324	12 %	36 %
Bit mémoire	2'100'416	14 %	87 %
Bloc DSP	20	2 %	15 %

Tableau 6.9 - Ressources utilisées pour notre système après la synthèse

Les ressources utilisées sont suffisamment faibles pour permettre l'intégration du générateur dans le projet. La fréquence maximale de fonctionnement obtenue après synthèse est de 41.28 MHz, ce qui est supérieur à la fréquence d'horloge visée de 40 MHz. Cela confirme que le générateur peut être intégré sans contrainte immédiate de timing.

Cependant, les signaux d'entrée et de sortie de l'entité principale ne sont pas systématiquement issus ou destinés à des registres. Ainsi, lors de l'intégration finale, certains chemins de propagation pourraient limiter la fréquence maximale atteignable. Dans ce cas, il conviendrait d'ajouter des registres intermédiaires aux endroits critiques afin de garantir le respect des contraintes temporelles.

Les ressources consommées par le système proviennent essentiellement du diviseur intégré au filtre CIC, son ajout a fait grimper l'occupation logique d'environ 4 %. En ce qui concerne la mémoire, la plus grande part est allouée au NCO, celui-ci fonctionne en mode Large ROM, une option qui utilise davantage de blocs RAM pour générer une forme d'onde de meilleure qualité.

Chapitre 7

Bancs de test

Ce chapitre décrit les bancs de test mis en place pour valider chaque bloc de notre système. Tous les bancs de test sont rédigés en SystemVerilog et exécutés sous QuestaSim. Des scripts .do automatisent la compilation des bibliothèques et le lancement des simulations, ce qui évite toute manipulation manuelle fastidieuse.

Avant d'exécuter le moindre test, il faut suivre la procédure de préparation détaillée en annexe B. Comme nous utilisons plusieurs IP Intel/Altera, leurs bibliothèques doivent être compilées au préalable. Pour cela, un script msim_setup.tcl est généré à l'aide de la commande ip-setup-simulation [35] fournie avec Quartus, le script compile automatiquement toutes les librairies requises (d'autres scripts sont aussi générés pour fonctionner avec des logiciels d'Aldec, Synopsys et Xcelium), à l'exception de celle du bloc "diviseur", traitée comme notre code VHDL interne.

Quatre bancs de test ont été développés :

- cic_tb : vérifie le fonctionnement du filtre CIC.
- quadrature_demod_tb : valide la démodulation I/Q.
- error_tb : teste la logique de détection d'erreurs.
- processing_tb : contrôle l'intégration de l'ensemble du design.

Les sections suivantes détaillent pour chacun d'eux les stimuli appliqués, les critères de succès et les résultats obtenus.

Lorsque l'on compare la sortie simulée d'un bloc VHDL avec une référence, il nous faut un indicateur nous indiquant si la sortie est correcte ou non. Plusieurs métriques existent, tel que le MSE, la corrélation, le PSNR, etc... Mais le RMSE a été retenu car ce dernier permet d'amplifier l'erreur, utilise la même unité que le signal et est simple à implémenter.

7.1 Filtre CIC

Pour valider le filtre CIC décrit en VHDL, un modèle de référence a été réalisé en SystemVerilog reproduisant exactement l'architecture de la figure 5.3. Le banc de test compare la sortie du device under test (DUT) à cette référence lorsqu'on applique un sinus (propre ou bruité à 10 %) (sinus_wave_generator.sv).

Lancement des simulations

Le script cic.do automatise la compilation des différents codes et l'exécution des scénarios, voici les commandes permettant d'exécuter ce banc de test :

```
cd Code/sim

# Run the tests for each signal
vsim -do "do ../script/cic.do all"

# test a clean signal
vsim -do "do ../script/cic.do clean"

# test a noisy signal
vsim -do "do ../script/cic.do noisy"
```

Listing 7.1 – Commande pour lancer le banc de test cic_tb.sv

Synchronisation et métrique d'erreur

Comme le filtre CIC introduit une latence, les sorties du DUT et du modèle de référence sont réalignées à l'aide de FIFO internes.

Tous les 1 000 échantillons (paramètre rmse_sample modifiable dans cic_tb.sv), on calcule la RMSE, segmenter ainsi le signal permet de repérer facilement la zone où le filtre pourrait dériver (pentes ascendantes, crêtes, etc...).

Résultats – signal sans bruit

La figure 7.1 montre notre sinus d'entrée (x_i) et les sorties filtrées (y_o , ref_out_signal). Le gain théorique (RM) N et la largeur maximale de données Bmax (formule de Hogenauer) sont ajoutés, car ce furent deux sources d'erreur fréquentes pendant le développement.

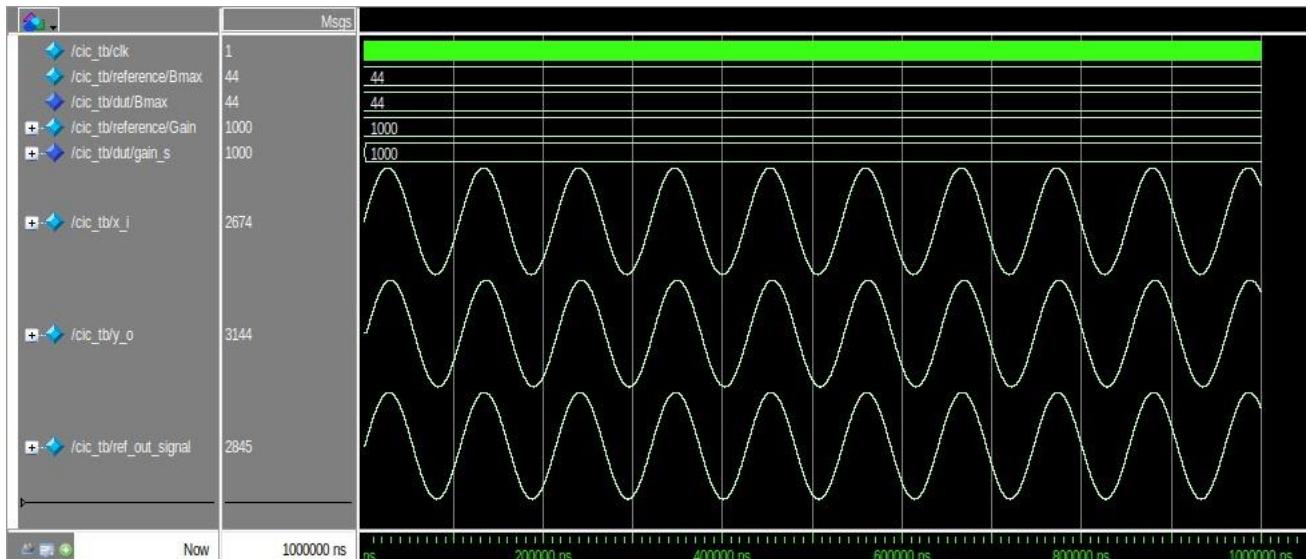


Figure 7.1 - Waveform du banc de test cic_tb.sv sans bruit

```
# ---- TESTCASE 0: Clean Signal Test ----
# Noise level: 0%
# PASS: signal test - RMSE = 0.640312 (threshold = 1.000000)
# PASS: signal test - RMSE = 0.752330 (threshold = 1.000000)
# PASS: signal test - RMSE = 0.676018 (threshold = 1.000000)
#
# ---- FINAL TEST RESULTS ----
# Test Case: 0 (CLEAN_SIGNAL)
# Noise Level: 0%
# Total Samples: 3835
# Final RMSE: 0.676018
# Threshold: 1.000000
# OVERALL RESULT: PASS
# ---- TEST COMPLETED ----
```

Listing 7.2 – Rapport QuestaSim pour un signal propre ($R = 10$, $M = 1$, $N = 3$)

L'erreur reste inférieure à ± 1 LSB, critère très souvent présent dans la documentation de l'AD2S1210, le filtre répond donc aux attentes.

Résultats – signal bruité

La figure 7.2 illustre le même essai avec un bruit appliqué au signal sinusoïdal (écart-type = 10 % de la pleine échelle). Le CIC restitue le sinus et atténue efficacement le bruit, la RMSE reste plus ou moins du même ordre qu'en conditions idéales.

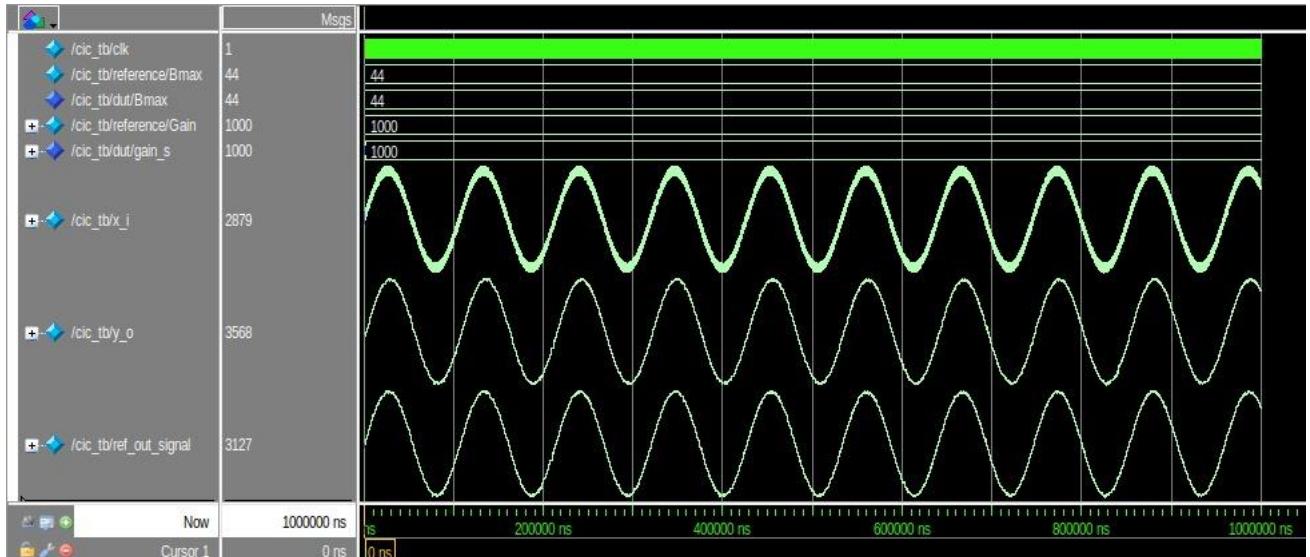


Figure 7.2 - Waveform du banc de test cic_tb.sv avec bruit

```
# ---- TESTCASE 1: Noisy Signal Test ----
# Noise level: 10%
# PASS: signal test - RMSE = 0.618870 (threshold = 1.000000)
# PASS: signal test - RMSE = 0.728011 (threshold = 1.000000)
# PASS: signal test - RMSE = 0.652687 (threshold = 1.000000)
#
# ---- FINAL TEST RESULTS ----
# Test Case: 1 (NOISY_SIGNAL)
# Noise Level: 10%
# Total Samples: 3836
# Final RMSE: 0.652687
# Threshold: 1.000000
# OVERALL RESULT: PASS
# ---- TEST COMPLETED ----
```

Listing 7.3 - Rapport QuestaSim pour un signal bruité ($R = 10, M = 1, N = 3$)

Conclusion

Ces deux scénarios confirment que :

- Le calcul du gain et de Bmax est correct.
- L'implémentation VHDL suit fidèlement le modèle de référence.
- Le filtre CIC amortit à merveille un bruit blanc de 10 % sans dépasser ± 1 LSB d'erreur RMS.

Les bancs de test qui suivent appliquent la même méthodologie (FIFO d'alignement + RMSE) aux blocs de démodulation et au système complet.

7.2 Démodulation en quadrature

Après la validation du filtre CIC, la même méthodologie a été appliquée au bloc de démodulation par quadrature. Un modèle de référence en SystemVerilog reproduit point par point la démodulation décrite en VHDL. Il

calcule la sortie attendue à partir d'un stimulus RVDT simulé (RVDT_simulator.sv), inspiré du générateur Python développé durant l'état de l'art.

Lancement des simulations

Le script demod.do offre les mêmes options que pour le CIC : signal "all", "clean" ou "noisy".

Deux paramètres supplémentaires sont toutefois requis :

- <angle> : position du rotor (0 – 360 °) pour le simulateur RVDT.
- <compile_IP> : 1 pour re-compiler les bibliothèques Intel/Altera (nécessaire après toute modification d'IP, hors diviseur), 0 sinon.

```
cd Code/sim

# Run the tests for each signal
vsim -do "do ../script/demod.do all <angle> <compile IP>"

# test a clean signal
vsim -do "do ../script/demod.do clean <angle> <compile IP>"

# test a noisy signal
vsim -do "do ../script/demod.do noisy <angle> <compile IP>"
```

Listing 7.4 - Commande pour lancer le banc de test quadrature_demod_tb.sv

Résultats – signal sans bruit

La figure 7.3 affiche :

- s1_in : cosinus du RVDT simulé (ici angle variant entre $\pm 80^\circ$).
- s1_out_ref : amplitude attendue (modèle SystemVerilog)
- s1_out : sortie du DUT.

Avec les paramètres exigeants $R = 2000$, $M = 2$, $N = 3$, la chaîne présente un gain CIC théorique très élevé, en raison des caractéristiques du filtre CIC, on observe donc une amplitude divisée par deux. Le banc de test applique donc un facteur cic_compensation_factor = 2. Les amplitudes mesurées (entrées : ± 2047 , sorties : $0 \rightarrow 2044$) montrent que la démodulation reproduit la partie positive du signal, comme attendu lorsque la composante n'a pas de transition significative entre les parties positive et négative.

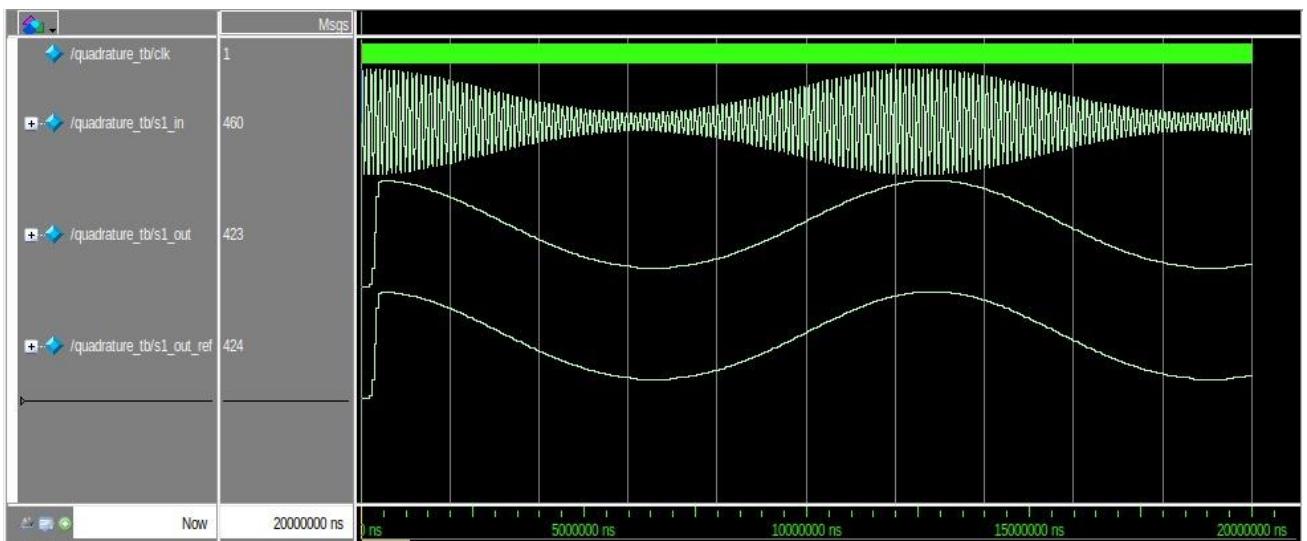


Figure 7.3 - Waveform du banc de test quadrature_demod_tb.sv sans bruit

```

# ---- TESTCASE 0: Clean Signal Test ----
# Noise level: 0%
# PASS: Sample      100 - RMSE = 0.848528 (threshold = 1.000000)
# PASS: Sample      200 - RMSE = 0.670820 (threshold = 1.000000)
# PASS: Sample      300 - RMSE = 0.781025 (threshold = 1.000000)
#
# ---- FINAL TEST RESULTS ----
# Test Case: 0 (CLEAN_SIGNAL)
# Noise Level: 0%
# Total Samples: 395
# Final RMSE: 0.781025
# Threshold: 1.000000
# OVERALL RESULT: PASS
# ---- TEST COMPLETED ----

```

Listing 7.5 - Rapport QuestaSim pour un signal propre ($R = 2000$, $M = 2$, $N = 3$)

L'erreur reste inférieure à ± 1 LSB, la démodulation par quadrature répond donc aux attentes.

Résultats – signal bruité

La figure 7.4 reprend la même simulation avec un bruit (10 % de la pleine échelle). Grâce au filtre CIC, le bruit est fortement atténué :

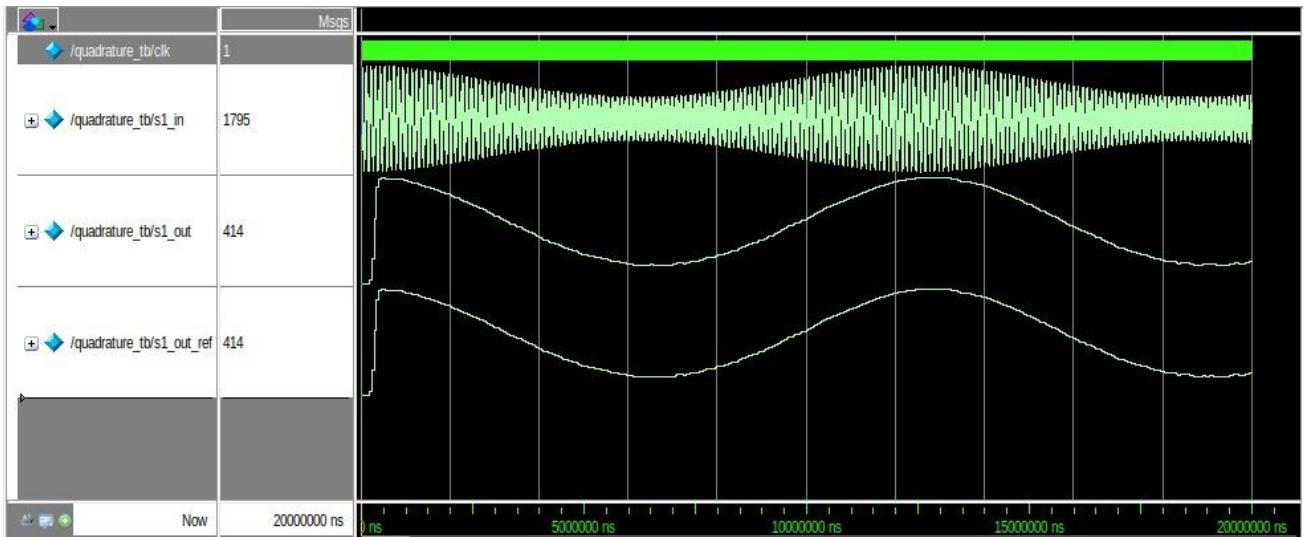


Figure 7.4 - Waveform du banc de test quadrature_demod_tb.sv avec bruit

```

# ---- TESTCASE 1: Noisy Signal Test ----
# Noise level: 10%
# PASS: Sample      100 - RMSE = 0.768115 (threshold = 1.000000)
# PASS: Sample      200 - RMSE = 0.714143 (threshold = 1.000000)
# PASS: Sample      300 - RMSE = 0.714143 (threshold = 1.000000)
#
# ---- FINAL TEST RESULTS ----
# Test Case: 1 (NOISY_SIGNAL)
# Noise Level: 10%
# Total Samples: 395
# Final RMSE: 0.714143
# Threshold: 1.000000
# OVERALL RESULT: PASS
# ---- TEST COMPLETED ----

```

Listing 7.6 - Rapport QuestaSim pour un signal bruité ($R = 2000$, $M = 2$, $N = 3$)

Conclusion

- Le modèle de référence et le DUT concordent à ± 1 LSB sur signal propre et bruité.
- Le facteur de compensation scalaire $\times 2$ suffit ici, la bande utile se situant dans une zone où le droop du CIC reste quasi constant.
- Ces résultats confirment la bonne intégration de la quadrature_demod avec le filtre CIC, base nécessaire avant de passer au banc de test système complet.

7.3 Système complet (Sans détection d'erreur)

Après avoir validé séparément le filtre CIC et la démodulation I/Q, la dernière étape consiste à vérifier l'assemblage intégral. Le banc de test rdc_tb.sv compare l'angle calculé par le DUT à l'angle imposé au simulateur RVDT (RVDT_simulator.sv). Le principe FIFO + RMSE, déjà utilisé pour les blocs unitaires, est réutilisé dans ce banc de test, l'unité utilisée pour le RMSE est en radians.

Lancement des simulations

Le script proc.do accepte les mêmes options que demod.do :

- `<angle>` : position du rotor ($0 - 360^\circ$) pour le simulateur RVDT.
- `<compile_IP>` : 1 pour re-compiler les bibliothèques Intel/Altera (nécessaire après toute modification d'IP, hors diviseur), 0 sinon.

```

cd Code/sim

# Run the tests for each signal
vsim -do "do ../script/rdc.do all <angle> <compile IP>"

# test a clean signal
vsim -do "do ../script/rdc.do clean <angle> <compile IP>"

# test a noisy signal
vsim -do "do ../script/rdc.do noisy <angle> <compile IP>"

```

Listing 7.7 - Commande pour lancer le banc de test processing_tb.sv

Synchronisation

La référence d'angle n'est soumise ni au CIC ni à la démodulation, elle ne présente donc pas la latence pipeline du DUT. Pour recaler correctement les deux flux, le banc de test :

- Attend la fin du temps de stabilisation du CIC

$$T_{settle} = (R \cdot M \cdot N) \cdot T_{clk}$$

- Ignore un nombre d'échantillons sur l'angle de référence grâce à la constante ANGLE_REF_DELAY (multipliée implicitement par R), afin d'écartier les valeurs utilisées pendant le temps de stabilisation du CIC.

Résultats – signal sans bruit

La figure 7.5 affiche :

- s1_in / s2_in : signaux cosinus / sinus générés par le simulateur RVDT (ici angle variant $\pm 200^\circ$).
- demodulated_cos_signal_s / demodulated_sin_signal_s : sorties démodulées
- angle_rad : angle calculé du DUT.
- rotor_angle_rad : angle de référence.

Comme pour le banc de test de la démodulation I/Q, les signaux démodulés issus du DUT présentent une amplitude divisée par deux en raison des paramètres élevés du CIC ($R = 1800$, $M = 2$, $N = 3$). Le banc de test applique donc un facteur cic_compensation_factor = 2. Ce coefficient n'influence pas la valeur d'angle calculée, il sert uniquement à garantir que la détection d'erreurs fonctionne correctement.

Mesuré sur Questasim, le cosinus d'entrée s1_in varie entre ± 2047 tandis que la sortie corrigée s'étend de -2045 à +2048, ce qui confirme que la démodulation reproduit fidèlement les parties positives comme négatives du signal d'origine. Pour l'angle, la référence oscille de -3,49066 radians à +3,49066 radians, alors que le DUT délivre -3,48926 radians à +3,49109 radians. Cette plage n'est pas une preuve absolue du bon fonctionnement du système, mais constitue un bon indicateur.

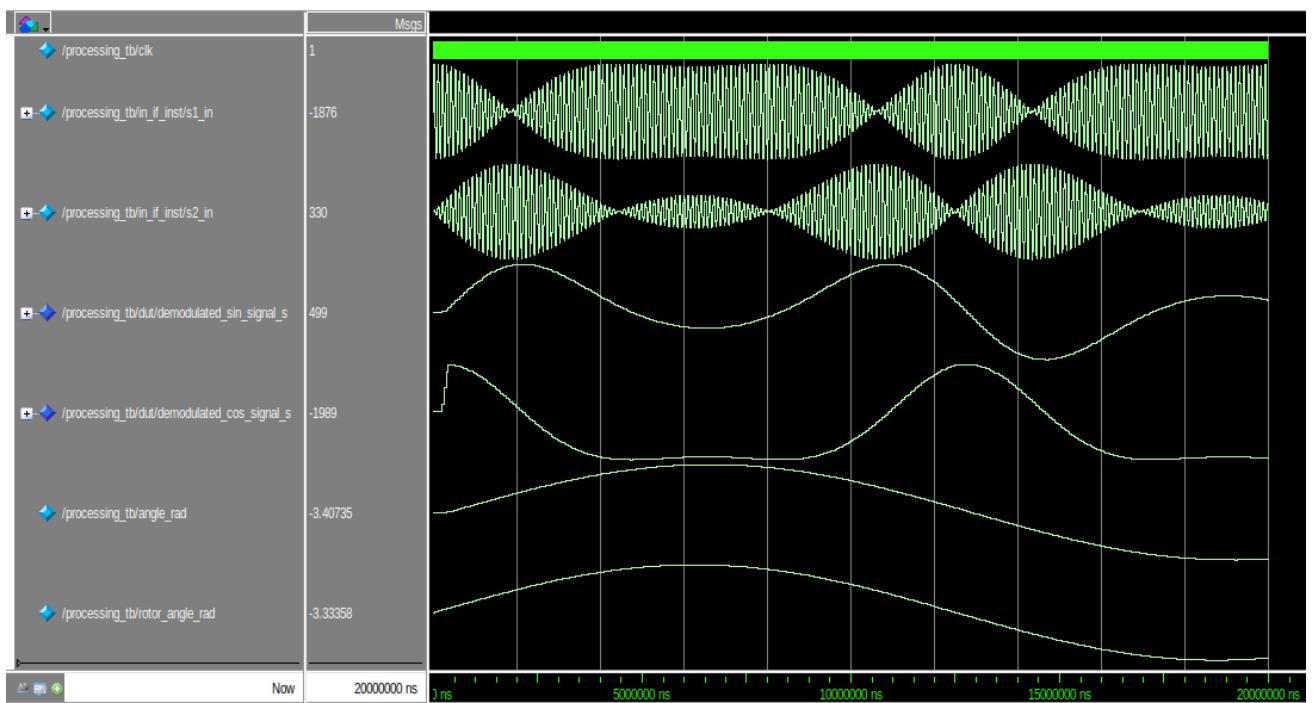


Figure 7.5 - Waveform du banc de test processing_tb.sv sans bruit

```

# ---- TESTCASE 0: Clean Signal Test ----
# Noise level: 0%
# PASS: Sample      100 - RMSE = 0.005302 (threshold = 0.010000)
# PASS: Sample      200 - RMSE = 0.004905 (threshold = 0.010000)
# PASS: Sample      300 - RMSE = 0.004911 (threshold = 0.010000)
# PASS: Sample      400 - RMSE = 0.005008 (threshold = 0.010000)
#
# ---- FINAL TEST RESULTS ----
# Test Case: 0 (CLEAN_SIGNAL)
# Noise Level: 0%
# Total Samples: 433
# Final RMSE: 0.005008
# Threshold: 0.010000
# OVERALL RESULT: PASS
# ---- TEST COMPLETED ----

```

Listing 7.8 - Rapport QuestaSim pour un signal propre ($R = 1800$, $M = 2$, $N = 3$)

L'erreur absolue reste inférieure à 0,01 rad ($\approx 0,57^\circ$), ce qui est acceptable pour notre système.

Résultats – signal bruité

La figure 7.6 nous montre un même scénario, mais avec un bruit de 10 % appliqué aux sorties sinus/cosinus du RVDT simulé. Le CIC supprime efficacement le bruit, la précision angulaire reste très bonne :

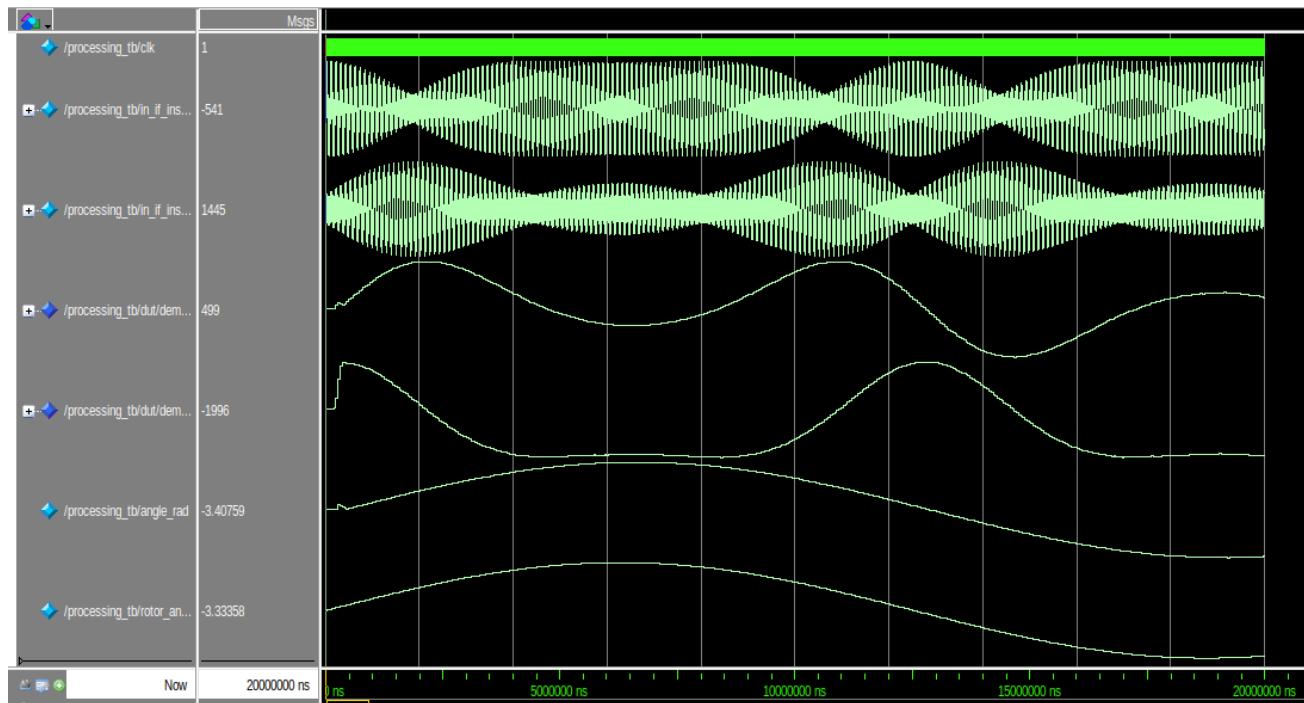


Figure 7.6 - Waveform du banc de test processing_tb.sv avec bruit

```

# ---- TESTCASE 1: Noisy Signal Test ----
# Noise level: 10%
# PASS: Sample      100 - RMSE = 0.007966 (threshold = 0.010000)
# PASS: Sample      200 - RMSE = 0.006152 (threshold = 0.010000)
# PASS: Sample      300 - RMSE = 0.006507 (threshold = 0.010000)
# PASS: Sample      400 - RMSE = 0.006956 (threshold = 0.010000)
#
# ---- FINAL TEST RESULTS ----
# Test Case: 1 (NOISY_SIGNAL)
# Noise Level: 10%
# Total Samples: 434
# Final RMSE: 0.006956
# Threshold: 0.010000
# OVERALL RESULT: PASS
# ---- TEST COMPLETED ----

```

Listing 7.9 - Rapport QuestaSim pour un signal bruité ($R = 1800$, $M = 2$, $N = 3$)

Conclusion

- Le banc de test confirme la cohérence temporelle entre référence et DUT, malgré la latence du CIC.
- L'angle calculé reste inférieur à 0,01 rad d'erreur RMS en conditions propres et bruitées (10 %)
- Ces résultats confirment la parfaite intégration et la bonne coopération de tous les blocs du système.

7.4 Détection d'erreur

La validation de la logique d'erreur diffère sensiblement des bancs de test précédents. Il ne s'agit plus de comparer une sortie numérique à un modèle de référence, mais de vérifier que chaque faute se déclenche (ou non) lorsque les seuils programmés le requièrent.

En revanche, la formule du moniteur n'est pas testée de bout en bout par simulation aléatoire, elle a été vérifiée manuellement sur plusieurs points, par exemple, pour un angle fixe de 45 ° (0,785 rad) comme l'illustre la figure 7.7

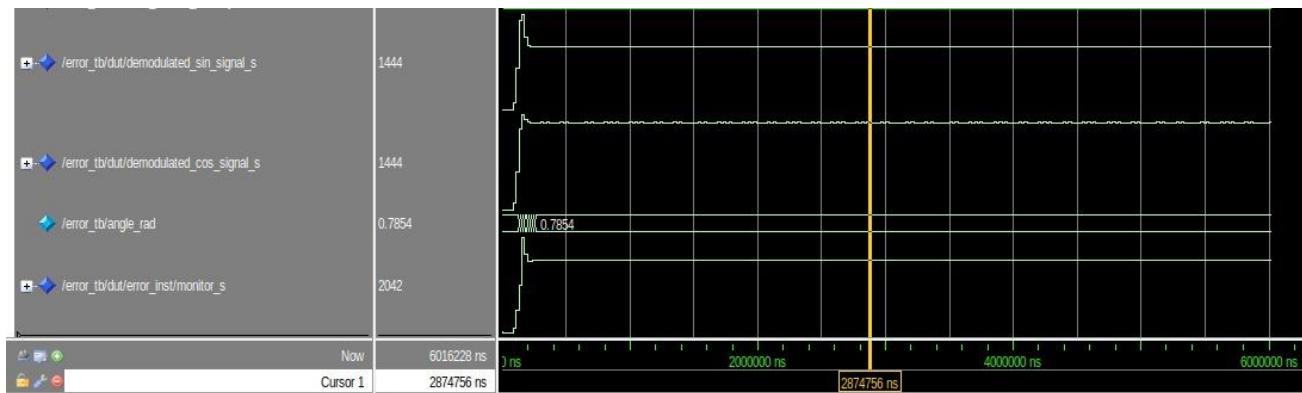


Figure 7.7 – Exemple vérification du moniteur

Pour rappel, la formule du moniteur est la suivante :

$$\text{Monitor} = A_1 \cdot \sin\theta \cdot \sin\phi + A_2 \cdot \cos\theta \cdot \cos\phi$$

Cela représente avec les valeurs de la figure 7.7 :

$$\text{monitor_s} = \text{demodulated_sin_signal_s} \cdot \sin(\text{angle_rad}) + \text{demodulated_cos_signal_s} \cdot \cos(\text{angle_rad})$$

En appliquant cette formule avec les valeurs du curseur nous obtenons 2042.12, ce qui correspond à monitor_s dans notre figure 7.7.

Lancement des simulations

Le script error.do ne prend qu'un seul argument :

- <compile_IP> : 1 pour re-compiler les bibliothèques Intel/Altera (nécessaire après toute modification d'IP, hors diviseur), 0 sinon.

```
cd Code/sim
```

```
# Run all the tests
vsim -do "do ../script/error.do <compile IP>"
```

Listing 7.10 - Commande pour lancer le banc de test error_tb.sv

Principe du test

Cinq scénarios sont enchaînés, chacun modifie les registres pour placer le seuil juste en-dessous puis juste au-dessus de la valeur mesurée, afin de vérifier l'activation correcte du bit d'erreur. Pour les tests 2 à 4 le simulateur RVDT génère sin et cos à 45 ° :

Test-case	Description du test
0	Lecture de l'ID
1	Clip : saturation en amplitude
2	Perte de signal (LOS)
3	Dépassement d'amplitude (DOS overrange)
4	Discordance sin/cos (DOS mismatch)

Test-case	Description du test
0	Lecture de l'ID
1	Clip : saturation en amplitude
2	Perte de signal (LOS)
3	Dépassement d'amplitude (DOS overrange)
4	Discordance sin/cos (DOS mismatch)

Résultats

Les figures 7.8 et 7.9 illustrent les signaux observés durant ces tests. Leur lecture à l'écran reste peu parlante, faute de détails suffisamment lisibles sur les captures. Elles sont incluses dans ce rapport à titre de référence. Cela permet, pour toute personne souhaitant rejouer les tests dans un autre simulateur (par exemple ModelSim, Riviera-Pro ou Vivado), de comparer les chronogrammes obtenus avec ceux issus de la simulation de référence sous QuestaSim. Ces figures affichent :

- sin_signal / cos_signal : signaux cosinus / sinus générés par le simulateur RVDT (ici angle fixe à 45°).
- test_case : test en cours.
- error_out : registre d'erreur accessible par l'utilisateur.
- err_clipped / err_los / err_mismatch : angle de référence.

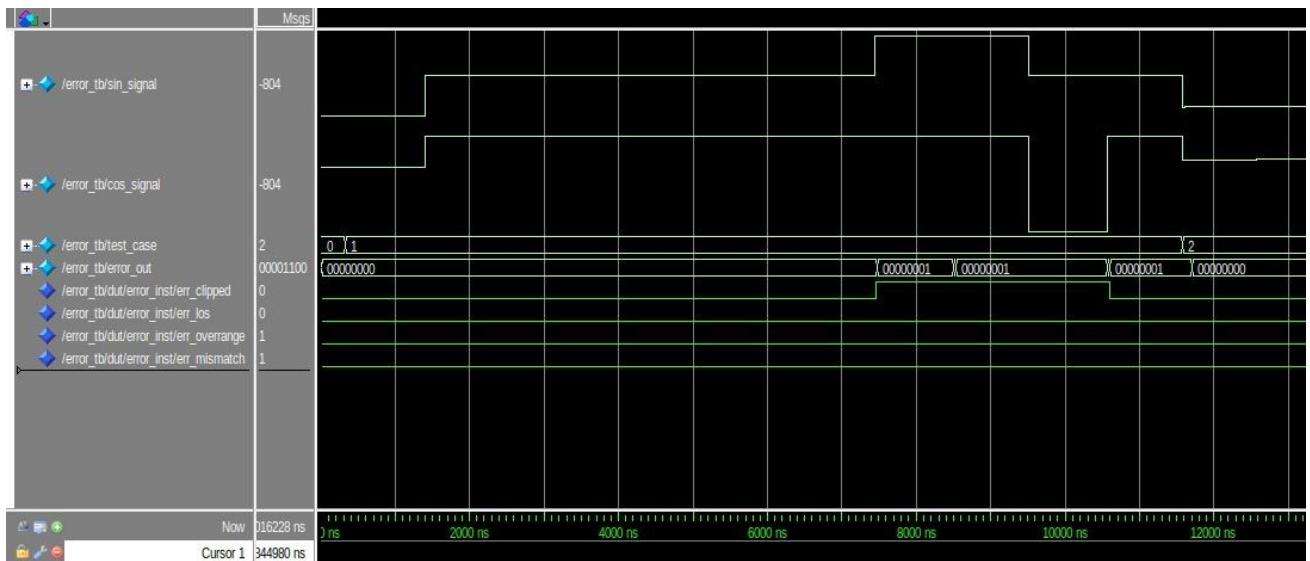


Figure 7.8 – Waveform montrant le test d7_clipping plus en détail

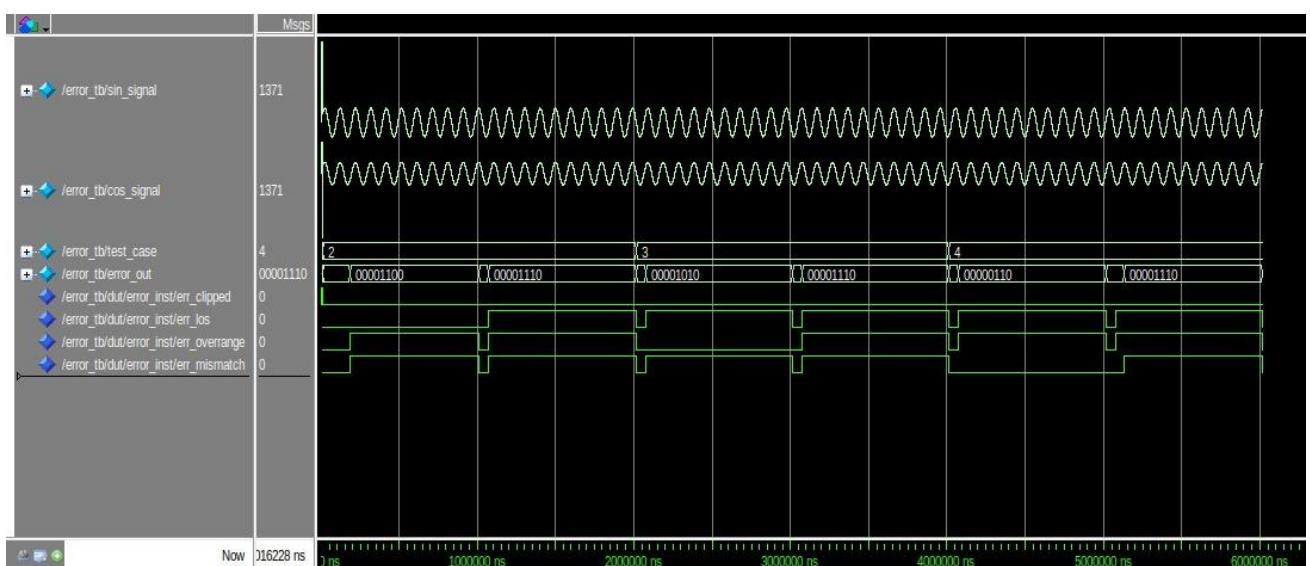


Figure 7.9 – Waveform du banc de test error_tb.sv en entier

```

# ---- ID Verification ----
# ID Verification PASSED: ID = 12345678
# ---- Testing D7: Clipping Detection ----
# PASS: No clipping detected for normal signals
# PASS: Clipping detected for positive sin saturation
# PASS: Clipping detected for negative cos saturation
# ---- Testing D6: Loss of Signal Detection ----
# Test 1: should not trigger LOS
# PASS: No LOS detected
# Test 2: Increasing LOS threshold to trigger LOS
# PASS: LOS detected
# ---- Testing D5: Overrange Detection ----
# Test 1: should not trigger DOS
# PASS: No DOS detected
# Test 2: Increasing DOS threshold to trigger DOS
# PASS: DOS detected after increasing DOS threshold
# ---- Testing D4: Mismatch Detection ----
# Test 1: should not trigger MISMATCH
# PASS: No MISMATCH detected
# Test 2: Decreasing MISMATCH threshold to trigger MISMATCH
# PASS: MISMATCH detected after decreasing MISMATCH threshold
# ---- ERROR DETECTION TESTS COMPLETED ----

```

Listing 7.11 - Rapport QuestaSim pour notre détection d'erreur

Tous les cas déclenchent (ou non) le bit correspondant exactement comme attendu.

Conclusion

Le banc de test confirme que :

- La lecture/écriture des registres fonctionne (ID et CSR).
- Chaque faute s'active exactement lorsque le seuil l'exige.
- L'intégration du bloc error avec les données démodulées est correcte.

Ces résultats, ajoutés à ceux des bancs CIC, quadrature et système complet, valident l'ensemble de l'architecture.

Chapitre 8

Améliorations futures

Comme tout projet, celui-ci peut encore gagner en robustesse, en performances et en nouvelle fonctionnalité. Les pistes suivantes ont été identifiées :

- **Optimiser le calcul du gain** : remplacer la division directe par un pré-calcul de l'inverse du gain ($1/gain_s$) suivi d'une simple multiplication avec la sortie du CIC. Cette astuce réduit fortement la latence et l'occupation logique.
- **Ajouter le FIR de compensation** : insérer le filtre de compensation après le CIC éliminerait le droop et affinerait l'amplitude, améliorant la précision globale de la démodulation.
- **Étoffer le banc de test du bloc error** : Actuellement, seules les registres de seuil sont vérifiés. Un banc de test capable d'introduire des défauts réalistes (déséquilibre sin/cos, perte de signal, amplitude transitoire, etc.) dans RVDT_simulator.sv permettrait de valider la formule complète du moniteur.
- **Passer aux paquets fixed-point IEEE** : le code actuel repose sur des entiers signés. Migrer vers les packages fixed_pkg/fixed_generic_pkg offrirait davantage de précision et simplifierait la maintenance, tant pour le design que pour les bancs de test.
- **Affiner la logique de détection d'erreurs** : La logique de détection d'erreurs pourrait être affinée. Par exemple, le système pourrait lever une erreur spécifique lorsque le gain devient nul, évitant ainsi tout risque de division invalide. De même, si un FIR de compensation est implémenté, les erreurs fournies par l'IP devraient être capturées et relayées dans le registre d'erreurs global.
- **Réduire la mémoire des IP** : le NCO et d'autres blocs proposent des options d'optimisation qui peuvent libérer des ressources en cas de limitations dans les ressources FPGA.
- **Envisager l'IP CIC** : si une licence Intel/Altera est disponible, l'IP "CIC Compiler" simplifie la génération des coefficients FIR de compensation et garantit une implémentation plus robuste.
- **Error Enable Mask** : actuellement, le registre error_enable_mask permet simplement d'activer ou de désactiver les bits de faute à l'exécution. Une évolution intéressante serait de transformer ce réglage en paramètre générique. Dès la synthèse, le compilateur pourrait alors inclure ou exclure les blocs de détection correspondants. Par exemple, si le moniteur n'est pas utilisé dans une configuration donnée, toute la logique qui lui est liée pourrait être "supprimée", réduisant ainsi la surface matérielle et la consommation de ressources.
- **LVDT** : Le système démodule déjà sans difficulté les signaux issus d'un LVDT, mais il lui manque encore la partie qui transforme cette amplitude en position linéaire. Il conviendra donc d'implémenter l'algorithme de calcul de position linéaire, ainsi qu'un mécanisme de sélection de mode (RVDT ou LVDT) permettant de choisir dynamiquement la chaîne de calcul appropriée.

La mise en œuvre de ces améliorations renforcerait à la fois la fiabilité, la précision et l'efficacité matérielle du système.

Chapitre 9

Conclusion

Au terme de ce travail de Bachelor, nous avons démontré la faisabilité et la pertinence d'un traitement entièrement numérique des capteurs RVDT/LVDT du Beam Wire-Scanner (BWS) du CERN. La problématique initiale, remplacer le circuit Resolver-to-Digital Converter (RDC) externe, devenu rigide et limité en diagnostic, répond à la nécessité d'obtenir un système plus flexible, plus robuste au bruit de ligne (220 m de câble) et disposant de fonctions de diagnostic avancées.

L'objectif principal est de concevoir, dans un FPGA, un cœur IP capable de générer le signal d'excitation du capteur, de démoduler les voies sin/cos numérisées, de calculer l'angle ou translation et de fournir des indicateurs d'état du capteur. Cet objectif inclue l'analyse comparative des méthodes de démodulation, la sélection des filtres adaptés et la mise en place d'une chaîne complète de vérification.

Dans cette optique, plusieurs étapes ont été réalisées :

- Une analyse approfondie des signaux réels issus du resolver du CERN, nous permettant de caractériser le bruit et les contraintes spécifiques du système (longueur de câble, non-idealités).
- Une étude comparative de différentes techniques de démodulation (Hilbert, quadrature, PLL, Costas Loop) a été effectuée par simulation en Python, avec et sans bruit, afin de déterminer la méthode la plus adaptée.
- À la suite de cette analyse. La méthode de démodulation en quadrature (I/Q) a été choisie, particulièrement pour sa simplicité d'implémentation sur FPGA et pour sa robustesse face au bruit.
- Le choix du filtre CIC pour le filtrage des composantes I et Q a également été validé, spécifiquement pour sa légèreté en ressources et sa capacité de décimation dans le contexte FPGA.
- Implémentation VHDL de blocs modulaires (NCO, mélangeurs, CIC, CORDIC, registre d'erreurs) et élaboration de bancs de test exhaustif.

Les simulations ont confirmé que la démodulation par quadrature offre le meilleur compromis, précision élevée en conditions idéales et bruitées, latence minimale et coût matériel réduit, contrairement aux solutions PLL/Costas plus complexes à configurer.

L'architecture finale s'organise autour de quatre blocs principaux, soit démodulation I/Q, filtre CIC à décimation, système de détection d'erreurs inspiré du RDC AD2S1210 et générateur NCO indépendant. Le calcul de l'angle est assuré par un CORDIC (atan2) tandis que la logique d'erreur surveille clipping, perte ou dégradation de signal.

Les simulations QuestaSim ont prouvé la conformité fonctionnelle, chaque faute (clipping, LOS, DOS, Mismatch) est détectée au seuil prévu, et la chaîne restitue l'angle sans dérive. Les performances atteignent la résolution du RDC tout en offrant une flexibilité supérieure. L'objectif initial de remplacement de l'IC externe est ainsi atteint.

Bien que fonctionnelle, l'implémentation actuelle mérite plusieurs améliorations :

- Affinement de la logique d'erreurs : lever une alerte spécifique si le gain devient nul ou si le futur filtre FIR de compensation signale une saturation.
- Optimisation mémoire des IP (tables NCO, CIC) et usage d'un compilateur CIC licencié pour réduire la surface logique
- Paramétrage synthèse/masquage des registres d'erreurs pour éliminer automatiquement la logique inactive
- Extension au mode LVDT : ajout du calcul de position linéaire et d'un sélecteur dynamique RVDT/LVDT

- Optimisation du gain via pré-calcul d'inverses et ajout d'un filtre FIR de compensation pour corriger le droop résiduel après CIC

Ce projet démontre qu'un traitement sur FPGA permet d'outrepasser les limites des RDC commerciaux. Il facilite la maintenance, autorise l'intégration de diagnostics avancés et prépare l'instrumentation du BWS à de futures évolutions (capteurs linéaires, vitesses plus élevées, auto-calibrations). Les changements proposés garantissent ainsi une mesure plus précise, plus fiable et plus évolutive.

La planification détaillée en annexe A n'a pas pu être entièrement respectée, l'analyse approfondie de l'algorithme de démodulation et la rédaction du rapport ont exigé davantage de temps que prévu. Par ailleurs, le développement imprévu d'un filtre CIC "maison" a mobilisé près de deux semaines avant d'aboutir à une solution fonctionnelle et fiable, impactant de façon non négligeable le planning.

Sur le plan personnel, ce projet s'est avéré particulièrement enrichissant. Il m'a permis de consolider mes compétences en VHDL et SystemVerilog, ainsi que de me familiariser avec l'environnement Quartus pour la génération d'IP Cores. J'ai également découvert les bases de MATLAB, même si cette prise en main est restée brève. Je reste toutefois légèrement frustré de ne pas avoir pu mener à bien la partie dédiée au LVDT, et de livrer un système encore améliorable en termes de robustesse et de précision. Les contraintes de temps liées à ce travail n'ont malheureusement pas permis de traiter l'ensemble de ces aspects. Malgré tout, réaliser un projet pour le CERN a été une expérience extrêmement motivante, et j'espère que les résultats obtenus pourront être intégrés à l'instrumentation finale.

Maillard Patrick

Bibliographie

- [1] C. Dieperink, « Génération de trajectoires à la demande pour le contrôleur du Beam Wire-Scanner du CERN ».
- [2] J. Emery, « CERN Beam Wire-Scanners »,
- [3] J. Emery, « Basic entity interfaces ».
- [4] K. Bouallaga, L. Idkhajine, A. Prata, et E. Monmasson, « Demodulation methods on fully FPGA-based system for resolver signals treatment », in *2007 European Conference on Power Electronics and Applications*, Aalborg, Denmark: IEEE, 2007, p. 1-6. doi: 10.1109/EPE.2007.4417560.
- [5] « Transformation de Hilbert », *Wikipédia*. 6 mai 2025. Consulté le: 20 mai 2025. [En ligne]. Disponible sur: https://fr.wikipedia.org/w/index.php?title=Transformation_de_Hilbert&oldid=225438925
- [6] « Hilbert transform », *Wikipedia*. 14 avril 2025. Consulté le: 20 mai 2025. [En ligne]. Disponible sur: https://en.wikipedia.org/w/index.php?title=Hilbert_transform&oldid=1285600529
- [7] T. Ulrich, « Envelope calculation from the Hilbert transform ».
- [8] « In-phase and quadrature components », *Wikipedia*. 17 avril 2025. Consulté le: 20 mai 2025. [En ligne]. Disponible sur: https://en.wikipedia.org/w/index.php?title=In-phase_and_quadrature_components&oldid=1286067732
- [9] « Phase-locked loop », *Wikipedia*. 25 février 2025. Consulté le: 20 mai 2025. [En ligne]. Disponible sur: https://en.wikipedia.org/w/index.php?title=Phase-locked_loop&oldid=1277626667
- [10] F. Küçük, « Answer to “Demodulating AM signal from carrier signal on Matlab” », Signal Processing Stack Exchange. Consulté le: 20 mai 2025. [En ligne]. Disponible sur: <https://dsp.stackexchange.com/a/44436>
- [11] Q. Chaudhari, « Phase Locked Loop (PLL) in a Software Defined Radio (SDR) », Wireless Pi. Consulté le: 20 mai 2025. [En ligne]. Disponible sur: <https://wirelesspi.com/phase-locked-loop-pll-in-a-software-defined-radio-sdr/>
- [12] « Fig 12: Block diagram of Costas loop Fig.12 shows the basic block... », ResearchGate. Consulté le: 20 mai 2025. [En ligne]. Disponible sur: https://www.researchgate.net/figure/Block-diagram-of-Costas-loop-Fig12-shows-the-basic-block-diagram-of-Costas-loop-the_fig2_345419696
- [13] « Costas loop », *Wikipedia*. 10 octobre 2024. Consulté le: 20 mai 2025. [En ligne]. Disponible sur: https://en.wikipedia.org/w/index.php?title=Costas_loop&oldid=1250427526
- [14] K. Sung-mi, S. Yeon-ho, J. Yu-rin, L. Min-woong, C. Seong-ik, et L. Jong-yeol, « FPGA Implementation of RVDT Digital Signal Conditioner with Phase AutoCorrection based on DSP », p. 8, 2017. [En ligne]. Disponible sur: <https://doi.org/10.6109/jkiice.2017.21.6.1061>
- [15] w2aew, #170: *Basics of IQ Signals and IQ modulation & demodulation - A tutorial*, (3 septembre 2014). Consulté le: 20 mai 2025. [En ligne Vidéo]. Disponible sur: https://www.youtube.com/watch?v=h_7d-m1ehoY
- [16] M. Correvon, « Capteurs inductifs de position ».
- [17] « A Beginner’s Guide To Cascaded Integrator-Comb (CIC) Filters - Rick Lyons ». Consulté le: 20 mai 2025. [En ligne]. Disponible sur: <https://www.dsprelated.com/showarticle/1337.php>
- [18] « An Intuitive Look at Moving Average and CIC Filters », Electronics etc.... Consulté le: 20 mai 2025. [En ligne]. Disponible sur: <https://tomverbeure.github.io/2020/09/30/Moving-Average-and-CIC-Filters.html>
- [19] « Cascaded integrator-comb filter », *Wikipedia*. 12 janvier 2025. Consulté le: 20 mai 2025. [En ligne]. Disponible sur: https://en.wikipedia.org/w/index.php?title=Cascaded_integrator%E2%80%93comb_filter&oldid=1269032789
- [20] « CIC Filters ». Consulté le: 20 mai 2025. [En ligne]. Disponible sur: https://www.gibbard.me/cic_filters/cic_filters_ipython.html
- [21] S. Ayesha, « Impact of Resolver Error in Electric Vehicle Traction Systems »,
- [22] « Vladimir MIRNOV | University of Wisconsin–Madison, Madison | UW | Department of Physics | Research profile », ResearchGate. Consulté le: 23 juillet 2025. [En ligne]. Disponible sur: <https://www.researchgate.net/profile/Vladimir-Mirnov>
- [23] « NCO IP Core: User Guide », *intel*.
- [24] « CIC IP Core User Guide ».
- [25] « ALTERA_CORDIC IP Core User Guide », *intel*.
- [26] « AN-1039: Correcting Imperfections in IQ Modulators to Improve RF Signal Fidelity | Analog Devices ». Consulté le: 23 juillet 2025. [En ligne]. Disponible sur: <https://www.analog.com/en/resources/app-notes/an-1039.html>

- [27] « AD2S1210 ».
- [28] « AD2S1210 Resolver to digital. Configuration - Q&A - Precision ADCs - EngineerZone ». Consulté le: 23 juillet 2025. [En ligne]. Disponible sur: https://ez.analog.com/data_converters/precision_adcs/f/q-a/25314/ad2s1210-resolver-to-digital-configuration
- [29] E. Hogenauer, « An economical class of digital filters for decimation and interpolation », *IEEE Trans. Acoust. Speech Signal Process.*, vol. 29, n° 2, p. 155-162, avr. 1981, doi: 10.1109/tassp.1981.1163535.
- [30] Akusho, « Phase unwrapping on FPGA », r/DSP. Consulté le: 23 juillet 2025. [En ligne]. Disponible sur: https://www.reddit.com/r/DSP/comments/15cmxqv/phase_unwrapping_on_fpga/
- [31] « Understanding CIC Compensation Filters ». Altera Corporation, 2007.
- [32] « FIR Compiler User Guide », 2011.
- [33] « CIC Filter Analysis script ». Consulté le: 23 juillet 2025. [En ligne]. Disponible sur: <https://www.dsprelated.com/showcode/8.php>
- [34] « dsp.CICCompensationDecimator - Compensate for CIC decimation filter using FIR decimator - MATLAB ». Consulté le: 23 juillet 2025. [En ligne]. Disponible sur: <https://ch.mathworks.com/help/dsp/ref/dsp.ciccompensationdecimator-system-object.html>
- [35] « 1.9.4.1. Generating a Combined Simulator Setup Script », Intel. Consulté le: 23 juillet 2025. [En ligne]. Disponible sur: <https://www.intel.com/content/www/us/en/docs/programmable/683102/21-3/generating-a-combined-simulator-setup-script.html>

Annexes

A. Planification

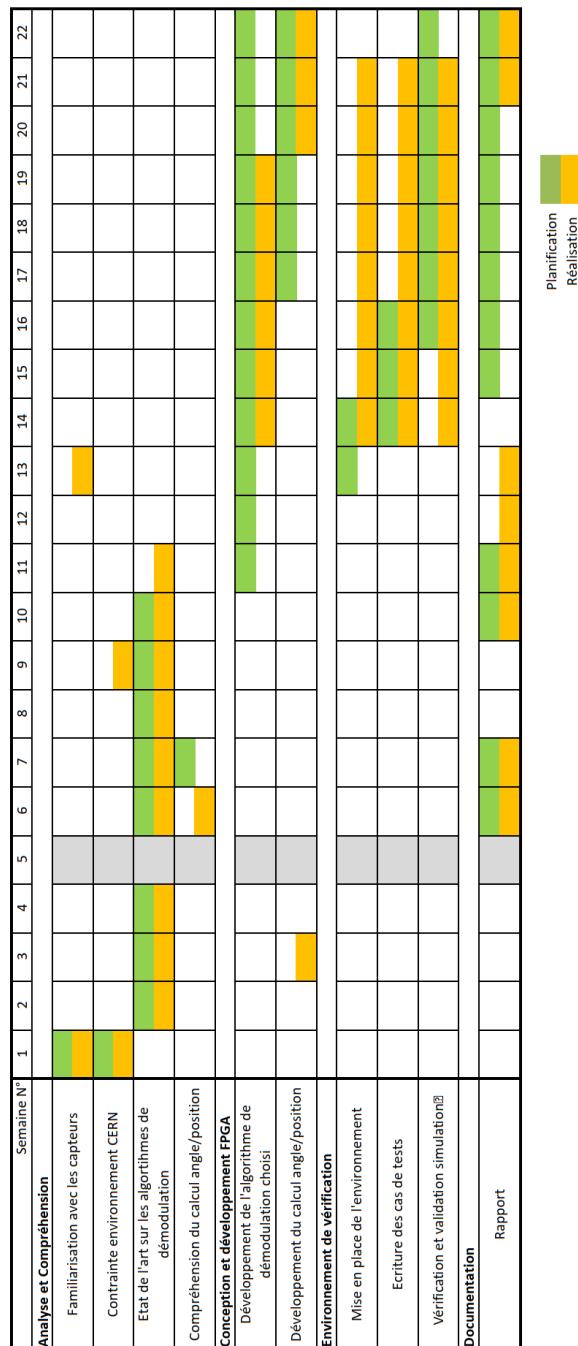


Figure A.1 - Planification

B. Configuration Banc de tests

Imaginons que votre projet Quartus contienne déjà tous les IP cores configurés et générés, vos descriptions VHDL et une synthèse passée sans erreur. Voici ci-dessous la liste des étapes afin de pouvoir tester votre design (sur QuestaSim/ModelSim). Voici, d'après [35], la procédure pour lancer la simulation dans QuestaSim/ModelSim:

1. Ouvrez un terminal et placez-vous dans le répertoire où se trouve le fichier <my proj>.qpf généré par Quartus.
2. Exécutez la commande du Listing B.1, en remplaçant <my proj> par votre fichier qpf, puis remplacer <my dir> par le répertoire où vous souhaitez créer le script.
3. Si le terminal affiche un avertissement du type "2025.07.23.16:23:14 Warning: QIP file/ip/nco/nco/synthesis/nco.qip has no associated spd file (expected:/ip/nco/nco/synthesis/nco.spd). You may need to generate simulation file sets.", cela signifie que le fichier spd n'est pas généré dans le bon répertoire, en effet dans mon cas le fichier spd se trouve dans/ip/nco/nco/nco.spd, vous devrez donc déplacer ce dernier dans/ip/nco/nco/synthesis/. Si le fichier spd n'existe pas pour une IP cela signifie soit qu'il y a eu une erreur pendant la génération HDL de l'IP, soit que l'IP n'a pas de fichier spd généré, ce qui ne nécessite que d'un vcom simple du fichier vhd (comme pour l'IP du diviseur montré au Listing B.2). Et réessayer l'étape 2.
4. Dans <my dir>, identifiez le script de configuration généré pour Mentor généralement *msim_setup.tcl* qui est compatible avec QuestaSim/ModelSim.
5. Préparez votre script personnel (par ex. un fichier .do) pour lancer la simulation. Un exemple figure au Listing B.2.
6. Exécutez votre script.
7. Si un message d'erreur indique qu'un fichier ou répertoire est introuvable, corrigez manuellement les chemins erronés dans *msim_setup.tcl* ou dans votre *script .do*, puis relancez l'étape 6.
8. Lorsque tout est correctement configuré, la fenêtre de QuestaSim s'ouvre et la simulation démarre. Vous pouvez alors visualiser les signaux et procéder au débogage.

Il est important de noter que toute modification d'un IP oblige à reprendre l'ensemble des étapes décrites précédemment.

```
ip-setup-simulation --quartus-project=<my proj> \
--output-directory=<my_dir> \
--use-relative-paths \
--compile-to-work
```

Listing B.1 – Commande pour générer un script de simulation

```

#!/usr/bin/tclsh

# Main proc at the end #

puts -nonewline "  Path_VHDL => "
set Path_VHDL      "../src"
set Path_TB        "../src_tb"
set Path_IP        "../ip"
set QUARTUS_ROOTDIR "/home/patrick/intelFPGA/23.1std/quartus"
set TOP_LEVEL_NAME error_tb

global Path_VHDL
global Path_TB
global Path_IP
global QUARTUS_ROOTDIR
global TOP_LEVEL_NAME

# Adjust the path as needed
source ../rdc_proj/simulation/mentor/mentor/msim_setup.tcl

#-----
proc vhdl_compil {} {
    global Path_VHDL
    global Path_TB
    global Path_IP
    global QUARTUS_ROOTDIR
    global TOP_LEVEL_NAME

    puts "\ncompilation :"

    vcom -2008 $Path_IP/divide/division.vhd
    vcom -2008 $Path_VHDL/error.vhd
    vcom -2008 $Path_VHDL/cic/cic_pkg.vhd
    vcom -2008 $Path_VHDL/cic/integrator.vhd
    vcom -2008 $Path_VHDL/cic/comb.vhd
    vcom -2008 $Path_VHDL/cic/cic_decimator.vhd
    vcom -2008 $Path_VHDL/quadrature_demod.vhd
    vcom -2008 $Path_VHDL/rdc_top.vhd
    vlog $Path_TB/error_tb.sv
}

proc do_all {} {
    vhdl_compil
    sim_start
}

#-----
proc sim_start {} {
    global TOP_LEVEL_NAME
    global USER_DEFINED_ELAB_OPTIONS
}

```

```

set TOP_LEVEL_NAME error_tb
set ELAB_OPTIONS ""
set USER_DEFINED_ELAB_OPTIONS "-voptargs=\"+acc +access+r\""
elab;
if {[file exists wave.do] == 0} {
    add wave -position insertpoint -radix decimal sim:/error_tb/sin_signal
    add wave -position insertpoint -radix decimal sim:/error_tb/cos_signal
    add wave -position insertpoint -radix decimal sim:/error_tb/test_case
    add wave -position insertpoint sim:/error_tb/error_out
    add wave -position insertpoint sim:/error_tb/dut/error_inst/err_clipped
    add wave -position insertpoint sim:/error_tb/dut/error_inst/err_los
    add wave -position insertpoint sim:/error_tb/dut/error_inst/err_overrange
    add wave -position insertpoint sim:/error_tb/dut/error_inst/err_mismatch
}

} else {
    do wave.do
}
run -all
wave zoom full
wave refresh
}

## MAIN #####
# Compile folder -----
if {[file exists work] == 0} {
    vlib work
}

# start of sequence -----
if {$argc > 1} {
    puts "Command is: vsim -do ../scripts/error.do <compile_IP>"
    quit -sim
}

} else {
    if {$1 == 1} {
        puts "Compiling IP cores..."
        dev_com
        com
    }
    puts "Error sim"
    do_all
}

```

Listing B.2 – Exemple de script pour Questasim/ModelSim

C. Illustration du fonctionnement du design à divers angles

Cette annexe présente des exemples montrant le fonctionnement du design pour d'autres angles, comme illustré au chapitre 7.

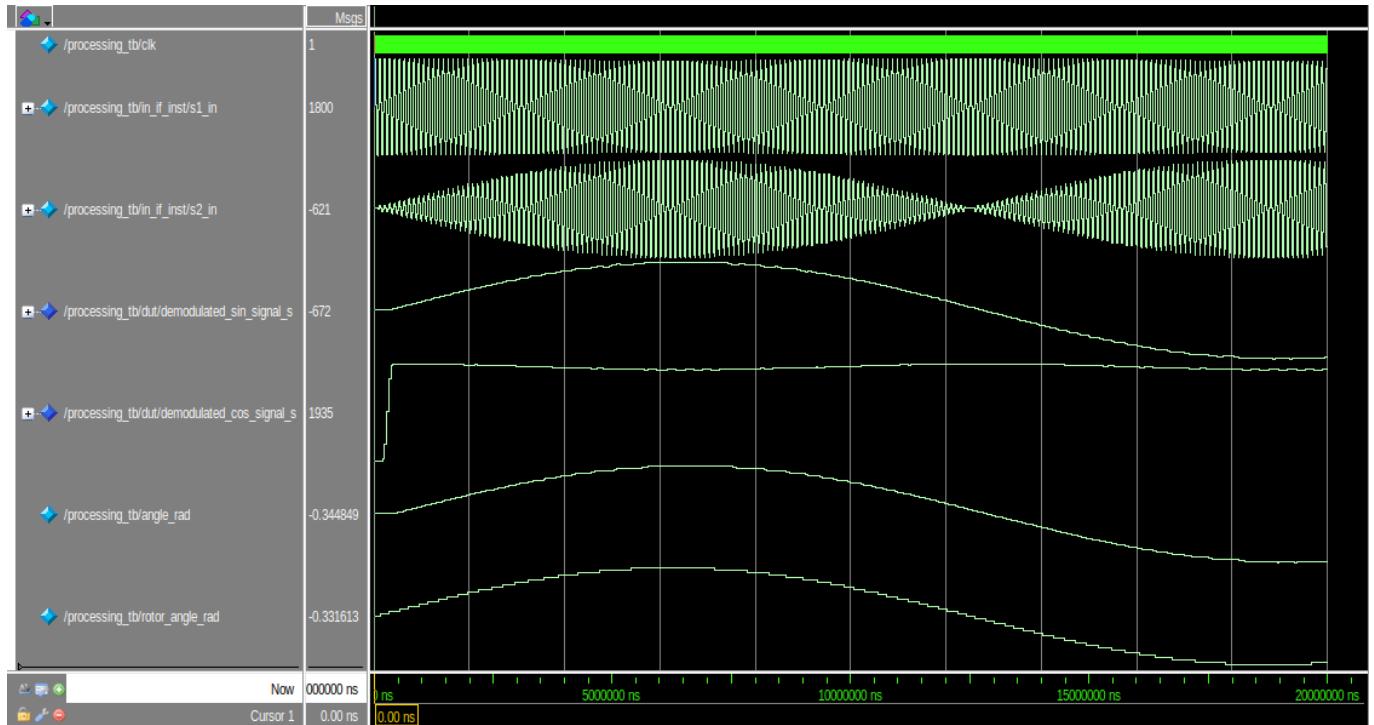


Figure C.1 – Démonstration Angle = 20°, sans bruit

```
# ---- TESTCASE 0: Clean Signal Test ----
# Noise level: 0%
# PASS: Sample      100 - RMSE = 0.004904 (threshold = 0.010000)
# PASS: Sample      200 - RMSE = 0.003728 (threshold = 0.010000)
# PASS: Sample      300 - RMSE = 0.006853 (threshold = 0.010000)
# PASS: Sample      400 - RMSE = 0.005328 (threshold = 0.010000)
#
# ---- FINAL TEST RESULTS ----
# Test Case: 0 (CLEAN_SIGNAL)
# Noise Level: 0%
# Total Samples: 431
# Final RMSE: 0.005328
# Threshold: 0.010000
# OVERALL RESULT: PASS
# ---- TEST COMPLETED ----
```

Listing C.1 – Rapport avec Angle = 20°, sans bruit

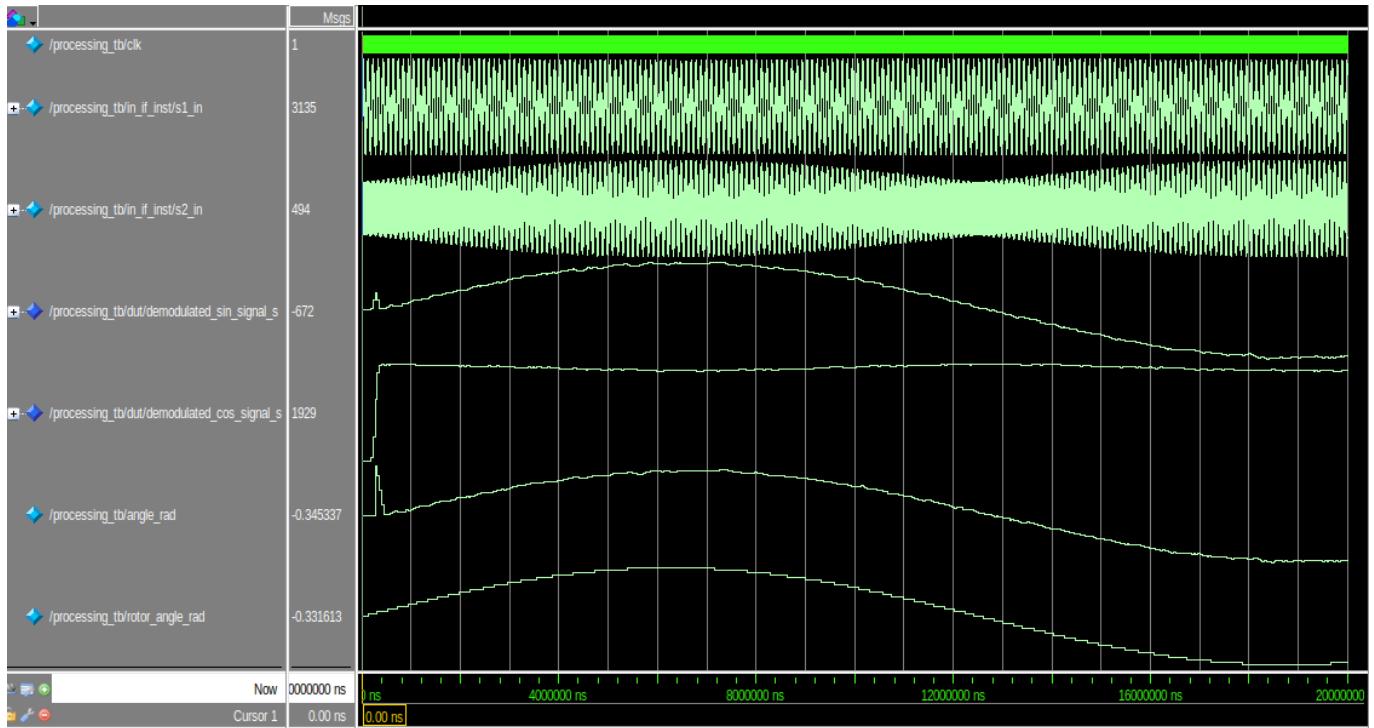


Figure C.2 - Démonstration Angle = 20°, avec bruit

```
# ---- TESTCASE 1: Noisy Signal Test ----
# Noise level: 10%
# PASS: Sample      100 - RMSE = 0.009896 (threshold = 0.010000)
# PASS: Sample      200 - RMSE = 0.006920 (threshold = 0.010000)
# PASS: Sample      300 - RMSE = 0.009486 (threshold = 0.010000)
# PASS: Sample      400 - RMSE = 0.008022 (threshold = 0.010000)
#
# ---- FINAL TEST RESULTS ----
# Test Case: 1 (NOISY_SIGNAL)
# Noise Level: 10%
# Total Samples: 434
# Final RMSE: 0.008022
# Threshold: 0.010000
# OVERALL RESULT: PASS
# ---- TEST COMPLETED ----
```

Listing C.2 - Rapport avec Angle = 20°, avec bruit

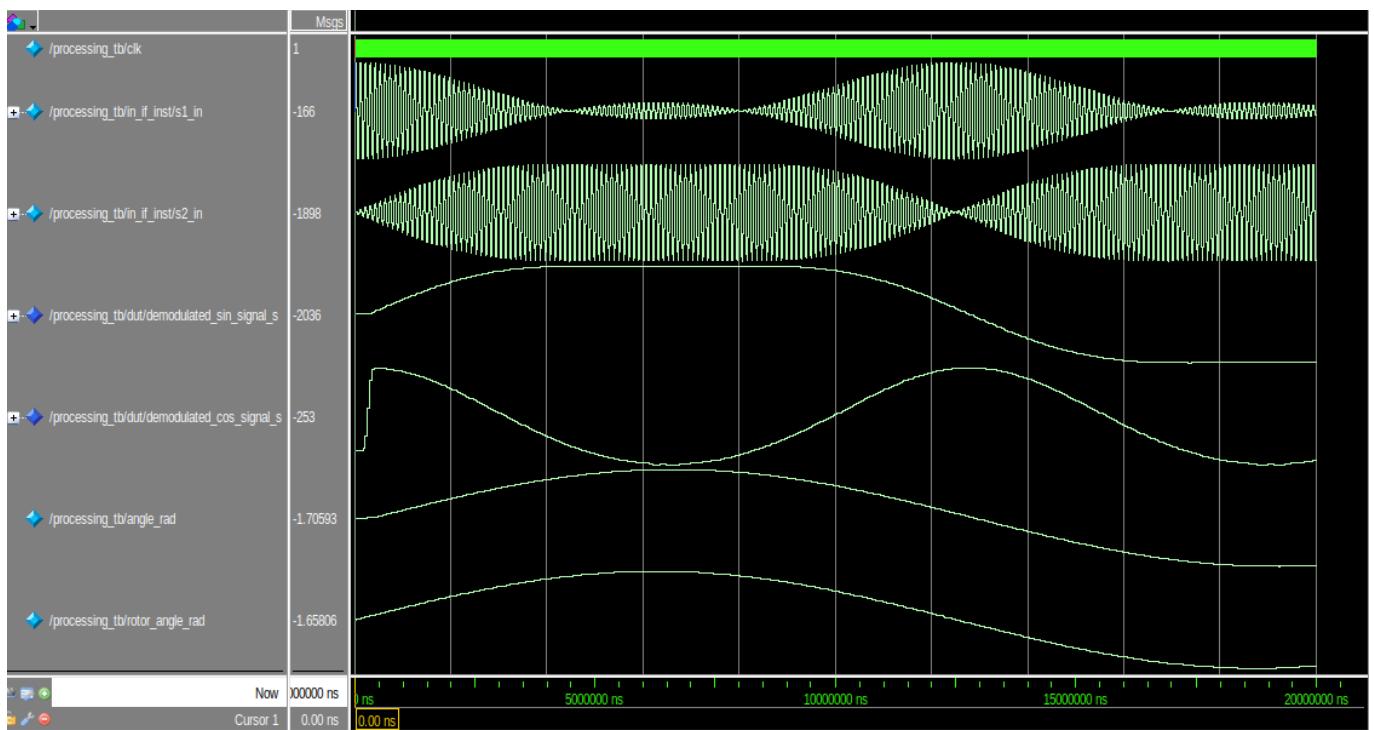


Figure C.3 - Démonstration Angle = 100°, sans bruit

```
# ---- TESTCASE 0: Clean Signal Test ----
# Noise level: 0%
# PASS: Sample      100 - RMSE = 0.005582 (threshold = 0.010000)
# PASS: Sample      200 - RMSE = 0.004421 (threshold = 0.010000)
# PASS: Sample      300 - RMSE = 0.005515 (threshold = 0.010000)
# PASS: Sample      400 - RMSE = 0.004998 (threshold = 0.010000)
#
# ---- FINAL TEST RESULTS ----
# Test Case: 0 (CLEAN_SIGNAL)
# Noise Level: 0%
# Total Samples: 434
# Final RMSE: 0.004998
# Threshold: 0.010000
# OVERALL RESULT: PASS
# ---- TEST COMPLETED ----
```

Listing C.3 - Rapport avec Angle = 100°, sans bruit

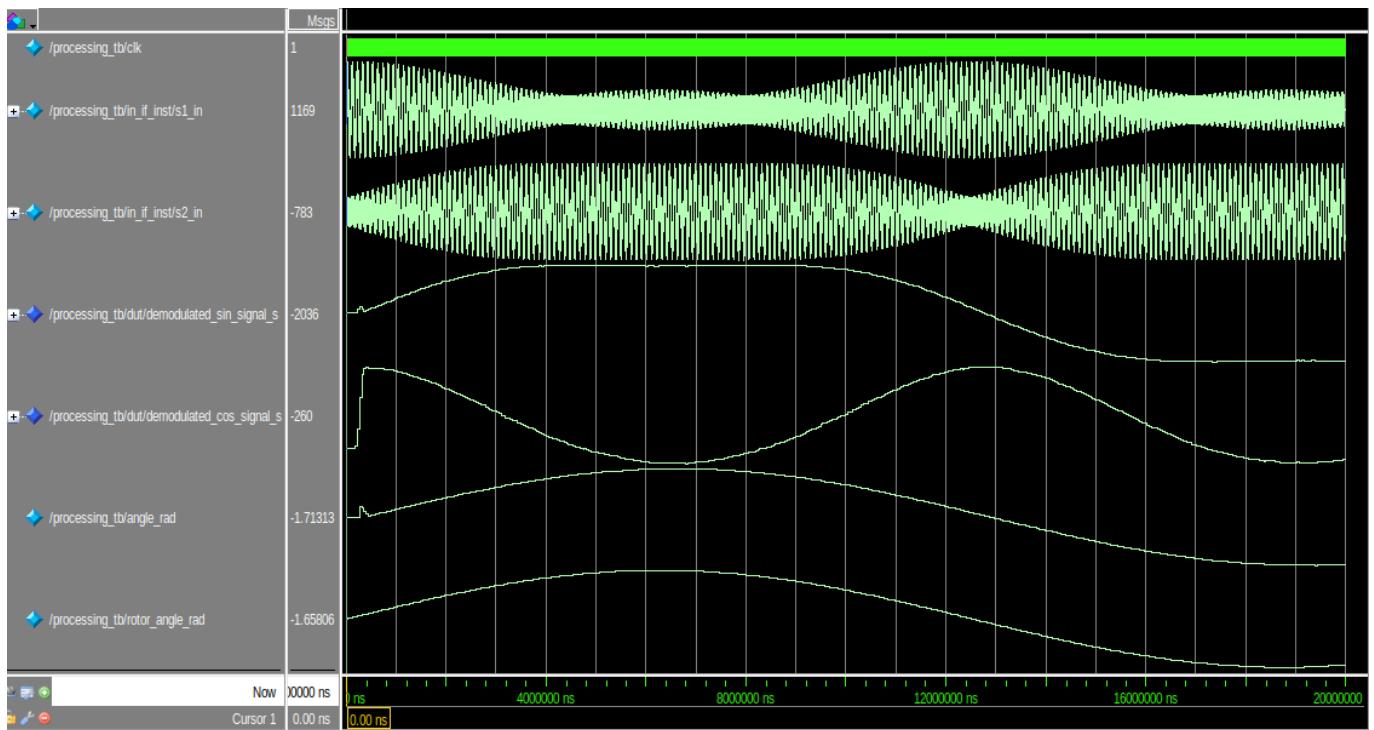


Figure C.4 - Démonstration Angle = 100°, avec bruit

```
# ----- TESTCASE 1: Noisy Signal Test -----
# Noise level: 10%
# PASS: Sample      100 - RMSE = 0.006616 (threshold = 0.010000)
# PASS: Sample      200 - RMSE = 0.006243 (threshold = 0.010000)
# PASS: Sample      300 - RMSE = 0.006772 (threshold = 0.010000)
# PASS: Sample      400 - RMSE = 0.006529 (threshold = 0.010000)
#
# ----- FINAL TEST RESULTS -----
# Test Case: 1 (NOISY_SIGNAL)
# Noise Level: 10%
# Total Samples: 434
# Final RMSE: 0.006529
# Threshold: 0.010000
# OVERALL RESULT: PASS
# ----- TEST COMPLETED -----
```

Listing C.4 - Rapport avec Angle = 100°, avec bruit

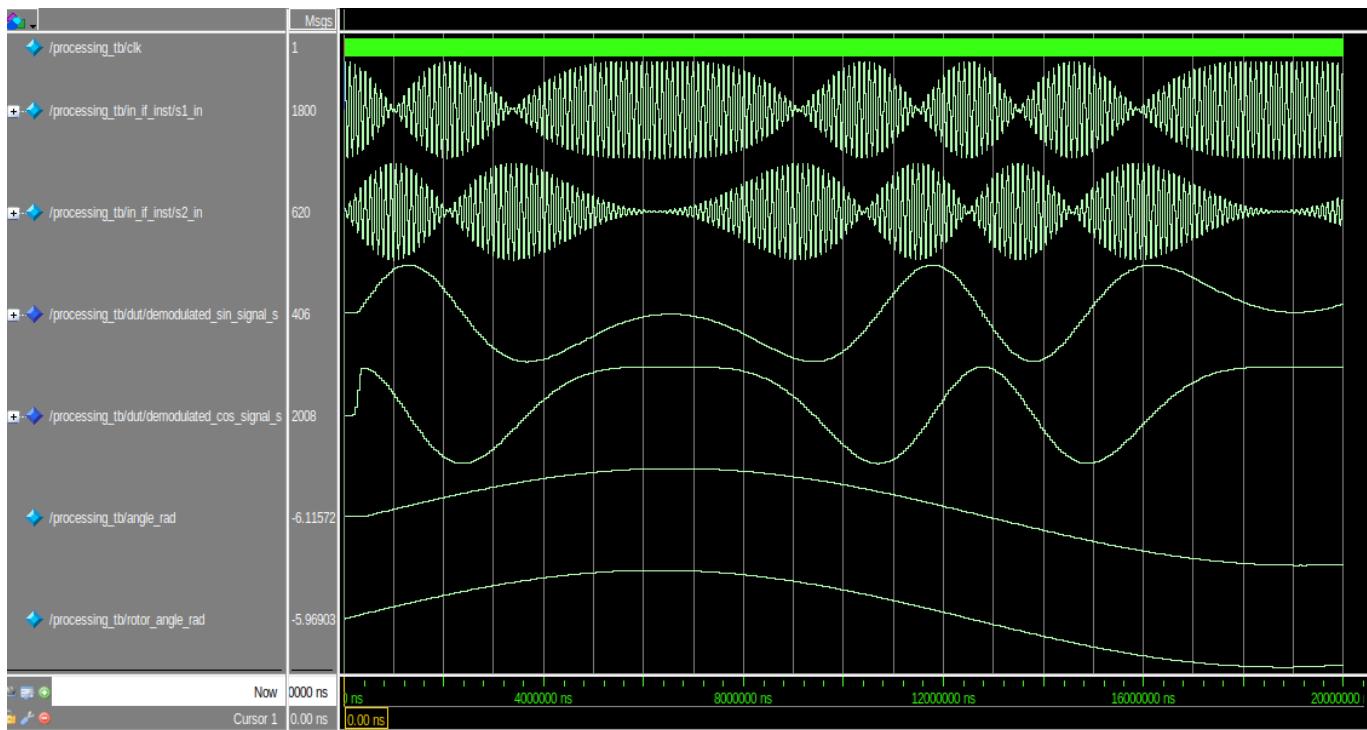


Figure C.5 - Démonstration Angle = 359°, sans bruit

```
# ---- TESTCASE 0: Clean Signal Test ----
# Noise level: 0%
# PASS: Sample      100 - RMSE = 0.005776 (threshold = 0.010000)
# PASS: Sample      200 - RMSE = 0.004717 (threshold = 0.010000)
# PASS: Sample      300 - RMSE = 0.005810 (threshold = 0.010000)
# PASS: Sample      400 - RMSE = 0.004926 (threshold = 0.010000)
#
# ---- FINAL TEST RESULTS ----
# Test Case: 0 (CLEAN_SIGNAL)
# Noise Level: 0%
# Total Samples: 433
# Final RMSE: 0.004926
# Threshold: 0.010000
# OVERALL RESULT: PASS
# ---- TEST COMPLETED ----
```

Listing C.5 - Rapport avec Angle = 359°, sans bruit

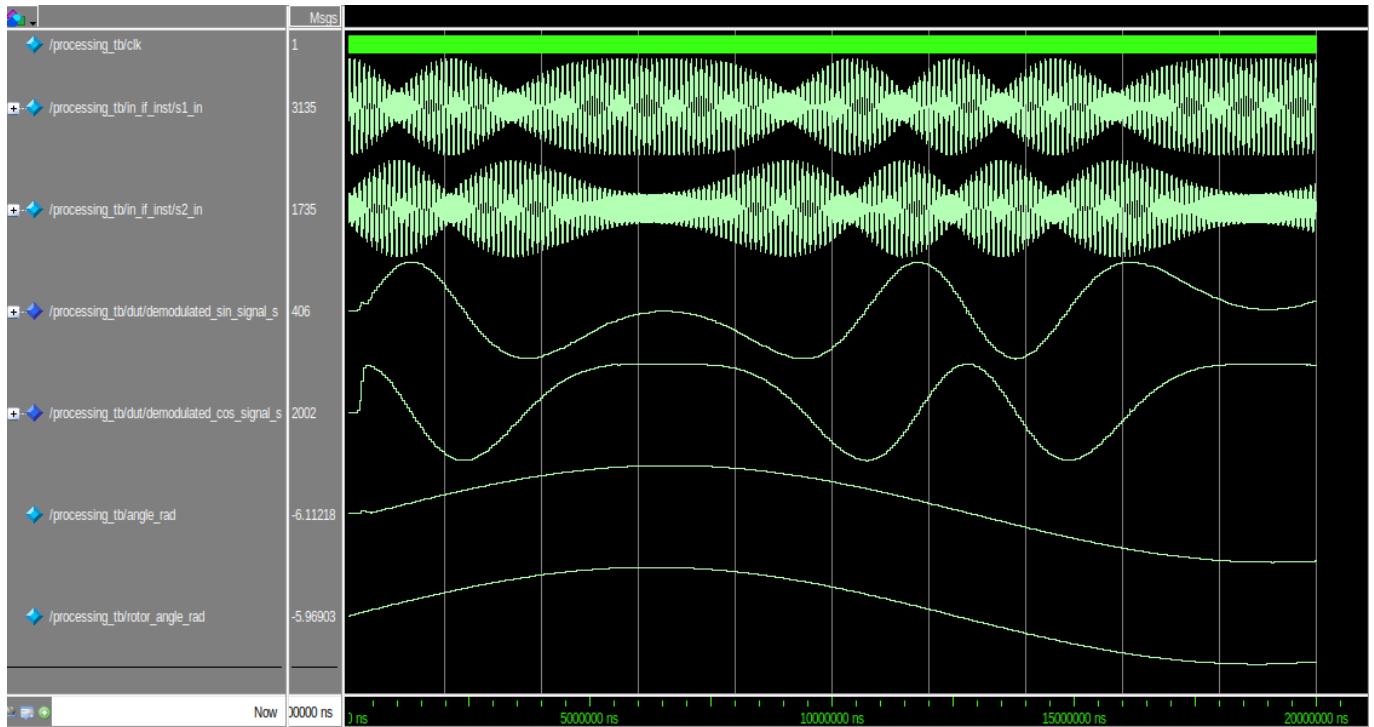


Figure C.6 - Démonstration Angle = 359°, avec bruit

```
# ---- TESTCASE 1: Noisy Signal Test ----
# Noise level: 10%
# PASS: Sample      100 - RMSE = 0.008389 (threshold = 0.010000)
# PASS: Sample      200 - RMSE = 0.006790 (threshold = 0.010000)
# PASS: Sample      300 - RMSE = 0.007256 (threshold = 0.010000)
# PASS: Sample      400 - RMSE = 0.006065 (threshold = 0.010000)
#
# ---- FINAL TEST RESULTS ----
# Test Case: 1 (NOISY_SIGNAL)
# Noise Level: 10%
# Total Samples: 434
# Final RMSE: 0.006065
# Threshold: 0.010000
# OVERALL RESULT: PASS
# ---- TEST COMPLETED ----
```

Listing C.6 - Rapport avec Angle = 359°, avec bruit

Glossaire

ADC (Analog to Digital Converter) : Composant électronique permettant de numériser un signal analogique

BI (Beam Instrumentation) : groupe Instrumentation du Faisceau du CERN

CIC (Cascaded Integrator Comb) : Filtre numérique sans multiplicateur, utilisé pour le filtrage et la décimation de signaux.

BWS (Beam Wire Scanner) : Balayeur à fil du CERN

CORDIC (Coordinate Rotation Digital Computer) : Algorithme itératif efficace permettant de calculer des fonctions trigonométriques pour des systèmes FPGA

FPGA (Field-Programmable Gate Array) : Circuit Logique programmable permettant de créer des architectures matérielles personnalisées.

I/Q (In-phase / Quadrature) : Méthode de décomposition d'un signal en 2 composantes orthogonales.

IP core (Intellectual Property core) : Bloc fonctionnel préconçu et réutilisable dans un projet FPGA.

LVDT (Linear Variable Differential Transformer) : Capteur permettant de mesurer une position linéaire à partir d'un signal différentiel.

RVDT (Rotary Variable Differential Transformer) : Capteur permettant de mesurer une position angulaire.

NCO (Numerically Controlled Oscillator) : Oscillateur numérique utilisé pour générer des signaux périodiques (exemple : sinus, cosinus)

PLL (Phase-Locked Loop) : Utilisé pour synchroniser la fréquence et la phase d'un oscillateur sur un signal d'entrée.

RDC (Resolver-to-Digital Converter) : Circuit dédié à la conversion des signaux analogiques d'un resolver en position numérique.

DSP (Digital Signal Processing) : Bloc de traitement de signal numérique

DOS (Degradation Of Signal) : indique une dégradation du signal

FIR (Finite Impulse Response) : filtre à réponse impulsionnelle finie

LOS (Loss Of Signal) : indique une perte de signal