ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE ÉLECTRIQUE
M. Ing.

PAR
BERNARD DIONNE

# ÉTUDE ET ANALYSE TEMPS RÉEL D'UN CANAL DE RÉCEPTION GPS

MONTRÉAL, LE 22 DÉCEMBRE 2006

CE MÉMOIRE A ÉTÉ ÉVALUÉ
PAR UN JURY COMPOSÉ DE :

M. René Jr Landry, directeur de mémoire
Département de génie électrique à l'École de technologie supérieure

M. Marcel Gabrea, président du jury
Département de génie électrique à l'École de technologie supérieure

M. Adrian Hiliuta, membre du jury
CMC Electronique Inc.

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC
LE 28 NOVEMBRE 2006
À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

# ANALYSE ET ÉTUDE EN TEMPS RÉEL D'UN CANAL DE RÉCEPTION GPS

Bernard Dionne

## SOMMAIRE

Dans le cadre de ce travail, nous avons analysé les performances d'un récepteur GPS numérique reprogrammable. Avec le développement de plus en plus rapide des systèmes GNSS, le besoin en récepteur numérique reprogrammable est devenu obligatoire pour toutes entreprises voulant concurrencer dans le domaine du positionnement satellitaire. À l'instar de ce que le SDR a fait pour l'évolution du cellulaire, le récepteur numérique reprogrammable se veut une première analyse des possibilités dans ce domaine.

La démarche utilisée durant cette recherche nous a permis d'identifier une plate-forme de développement de base pour l'implémentation du récepteur. L'analyse des variables d'environnements des signaux GPS a établi les balises pour la recherche des processeurs dans lesquels le récepteur sera implémenté. Il a été identifié que le récepteur sera divisé entre deux processeurs, c'est-à-dire un FPGA et un DSP, en raison des différents algorithmes compris à l'intérieur des boucles qui composent le modèle du récepteur GPS.

Afin d'arriver a implémenter un récepteur dans les processeurs, nous avons modélisé, à l'aide de Matlab/Simulink, une chaîne de communication GPS, de la source au récepteur. Après la modification de ce modèle pour générer un code binaire, la première étude du récepteur numérique a clairement montré que, pour un rapport signal à bruit plus faible que dans le cas d'un récepteur analogique, les performances du récepteur numérique sont nettement supérieur à ceux des récepteurs analogiques existants.

Pour réaliser un modèle plus réaliste, les perturbations appliquées sur le signal GPS ont pu démontrer que le récepteur numérique répondait plus rapidement à un stress dynamique important. De plus, la conception d'un tel récepteur pourrait être fastidieuse mais la méthodologie implémentée simplifie la programmation a un minimum, permettant ainsi à tout développeur mathématique de pouvoir réaliser son propre récepteur.

Les résultats de ce travail tout comme l'expertise acquise sont utilisés actuellement pour le développement d'un récepteur hybride complet, basé sur le concept du software defined navigator (SDN).

**ANALYSIS OF A REAL TIME ONE-CHANNEL GPS NUMRICAL RECEIVER**

Bernard Dionne

## ABSTRACT

In this work, we analysed the performance of a reconfigurable GPS digital receiver. With the satellite networks that are developing faster and faster, the need for a reconfigurable digital receiver will be an issue for companies involved in the satellite positioning market. As SDR did for the first phase of the evolution of cellular, the digital receiver is a first step to the many possibilities in that field.

The methodology used in this research was to; first, identify the materials needed to develop a platform to implement the receiver. In order to find it, the environment variables of the GPS signal lead us in the search of the right processors for that platform. Secondly, we divided our receiver in smaller process and we identified that the digital receiver was bound to be implemented in 2 differents types of processor, namely a DSP and a FPGA. The criteria of the division of the processes resided in the algorithm of each process, such as complexity and resources needed in the processor.

To achieve the implementation in the two processors, we generated a model, with Matlab/Simulink, a GPS communication link, from the satellite to the receiver. The code generated on the model, once implemented in the processor, clearly demonstrated that, for a given signal-to-noise ratio, the performances of the digital receiver are superior to the actual analogue receiver.

The realism of the model was kept by applying perturbations of the simulated source of the GPS satellite. The perturbation applied showed that the digital receiver had a faster time-to-respond for a higher Doppler effect. But the conception of a digital receiver could be a hard task but the methodology develop simplifies the programmation to a minimum, allowing developers to concentrate on the mathematical aspect of the design.

The results of this work, and the expertise acquired through it, are actually being used to develop a hybrid receiver based on the concept of software defined navigator (SDN).

## REMERCIEMENTS

Le développement d'un tel projet ne se fait pas sans l'apport d'une équipe multidisciplinaire. À la base, ce projet se devait d'analyser l'étude des discriminateurs pour un récepteur GPS mais j'ai accepté de modifier le but de ce travail afin de pouvoir contribuer au développement des récepteurs numériques qui, avec l'effervescence technologique actuelle, seront des outils à la fine pointe de la technologie.

J'aimerais remercier énormément pour son support, sa compréhension, sa guidance et sa patience, le professeur René Jr. Landry, qui, tout au long de cette maîtrise, a su m'aiguiller et me faire confiance. J'aimerais aussi remercier tous les collègues du LACIME et plus particulièrement, les stagiaires, chercheurs du groupe de travail SDN (Aurélian et Iurie particulièrement) qui ont aidé grandement au développement de cette maîtrise. J'aimerais aussi remercier tous mes amis, et particulièrement Marie-Ève et Léon, pour leur support et leurs encouragements.

Enfin, je dédie ce mémoire à mes parents, Gilbert et Françoise, pour leur indéniable support et pour leur extraordinaire patience. Sans eux, ce travail n'aurait jamais existé. Je leur en serai infiniment reconnaissant.

## TABLE DES MATIÈRES

Page

# LISTE DES TABLEAUX

# LISTE DES FIGURES

# LISTE DES ABRÉVIATIONS ET SIGLES

| | |
|---|---|
| AMR | Avance moins retard (type du discriminateur DLL) |
| BOC | *Binary Offset Carrier* |
| AltBOC | *Alternate BOC Modulation* |
| Bps | Bribe par seconde |
| BPSK | *Binary Phase Shift Keying* |
| $C/N_0$ | *Signal to Noise Ratio* |
| Code C/A | *Coarse Acquisition Code* |
| Code P | *Precision Code* |
| CDMA | *Code Division Multiple Access* |
| DLL | *Delay Locked Loop* |
| DSP | *Digital Signal Processor* |
| EAMR | Enveloppe Avance Moins Retard (type du discriminateur DLL) |
| EAMRN | Enveloppe Avance Moins Retard Normalisé (type du discriminateur DLL) |
| ECEF | *Earth-centered Earth-fixed* |
| ESA | *European Space Agency* |
| FPGA | *Field Programmable Gate Array* |
| GNSS | *Global Navigation Satellite System* |
| GPS | *Global Positioning System* |
| GUI | *Graphical User Interface* |
| IF | *Intermediate Frequency* |
| ITU | *International Telecommunication Union* |
| LACIME | *Laboratoire de Communication et d'Intégration de Micro-Électronique* |
| PLL | *Phase Locked Loop* |
| PRN | *Pseudo Random Noise* |
| PAMR | Puissance Avance Moins Retard (type du discriminateur DLL) |
| PC | Produit Croisé (type du discriminateur DLL) |

## LISTE DES ABRÉVIATIONS ET SIGLES (SUITE)

| | |
|---|---|
| PA | *Pseudo aléatoire (code)* |
| QPSK | *Quadrature Phase Shift Keying* |
| SDN | *Software Defined Navigator* |
| SDR | *Software Defined Radio* |
| SNR | *Signal to Noise Ratio* |
| SA | *Selective Availability* |
| VHDL | *VHSIC Hardware Description Language* |

# INTRODUCTION

La radionavigation est un domaine en pleine expansion. Durant ces dernières années, plusieurs gouvernements ont investi des millions de dollars dans différents systèmes de positionnement par satellite. L'intérêt des pays envers la radionavigation vient d'une demande grandissante par leurs citoyens ou les utilisateurs de connaître leur position exacte sur la surface du globe. Des notions de sécurité ont été invoquées pour le développement de ces systèmes. Depuis les années 70, seuls les Américains possédaient un tel système, soit le *Global Positionning System*, communément appelé GPS. Développé par l'armée américaine, ce système était surtout utilisé en temps de guerre afin de déterminer la position exacte de ses propres troupes et des troupes adverses. Cependant, on a rapidement vu le potentiel énorme de ce système.

## Problématique

L'exigence de la couverture mondiale d'un système de positionnement, ainsi que les besoins futurs de la radionavigation ne peuvent être satisfaits que par un seul système, le GPS. Le système GPS, quoiqu'efficace, possède quelques failles. Le niveau de précision (pouvant atteindre plusieurs dizaines de mètres) est nettement en deçà des besoins des utilisateurs. De plus, la fiabilité du système n'est pas optimale. Il faut se rappeler que le système GPS a été développé pendant la Guerre Froide (entre les Etats-Unis d'Amérique et l'ex-Union Soviétique) et est principalement dédié aux applications militaires. Les Russes ont aussi développé un système de positionnement, Glonass, mais ce dernier n'a aucune application civile. Ces caractéristiques des deux systèmes empêchent une bonne couverture mondiale pour les applications civiles. À la lumière de ces informations, il est nécessaire de développer d'autres systèmes de positionnement pour satisfaire les exigences des utilisateurs.

Avec l'augmentation du nombre de constellations de satellite donc de sources de signaux de positionnement, il est tout aussi essentiel de développer des récepteurs

hybrides pouvant récupérer ces signaux et les traiter adéquatement afin d'en sortir l'information pertinente. Avec cette évolution, la demande envers les récepteurs numériques sera en croissance pour les prochaines années. Selon les prévisions de l'Agence Spatiale Européenne, le marché des puces GNSS (récepteur numérique) atteindra 3,6 milliards de puces vendues en 2010. De plus, sur une période de 20 ans, la firme PriceWaterhouseCoopers estime un ratio entre les avantages et les coûts de ce marché à 4,6**[29]**. Avec un tel ratio, il n'est pas étonnant que plusieurs pays et compagnies investissent dans le domaine. L'ESA a donc lancé son programme de positionnement par satellite, nommé Galileo. Par exemple, le coût du système Galileo est estimé à 3,4 millions d'Euro. Imaginez les retombées financières d'un tel investissement. La nécessité de nouveaux récepteurs numériques n'est pas seulement un besoin technologique, mais aussi une opportunité financière importante.

**Objectifs**

Voyant l'importance et l'opportunité de développer un récepteur numérique, et devant l'effervescence de la technologie en ce moment, le laboratoire LACIME (Laboratoire de Communication et d'Intégration de Micro-Électronique) a décidé de créer son propre récepteur numérique. Pour effectuer le design de notre récepteur, nous avons basé nos démarches sur un concept reconnu, le « *Software Defined Radio* » (SDR). En ayant une plate-forme reprogrammable, il nous sera possible de diminuer les coûts et d'optimiser les ressources disponibles. Grâce à l'évolution rapide des signaux radio et à ses conséquences, la nécessité de développer une plate-forme reprogrammable est devenue apparente pour le domaine de la radionavigation. En accord avec le projet de recherche, les objectifs de ce mémoire sont :

1. Développer un environnement de travail
2. Conceptualiser un modèle temps réel d'une chaîne de communication GPS

a. Analyser les performances du modèle en cosimulation en présence de perturbations
b. Analyser les performances du modèle en temps réel en présence de perturbations

Afin d'en arriver à un environnement semblable au SDR[9], plusieurs étapes doivent être effectuées et ce mémoire s'inscrit dans le cadre de cette démarche. Après l'analyse des caractéristiques du domaine de recherche (dans notre cas, la radionavigation et ses composants), on arrive à édifier les bases d'un environnement de développement.

Cette base servira comme point de départ pour définir un ensemble que l'on pourrait qualifier de *Software Defined Navigator* (SDN), à l'image du SDR.

En respectant le réalisme de notre modèle, plusieurs modifications et analyses, dont la performance en présence de perturbations, devront être faites pour que ce modèle représente bien la problématique et devienne une solution viable aux problèmes actuels.

**Structure du mémoire**

Ce document est séparé en 3 parties distinctes.

Premièrement, dans les trois premiers chapitres, les bases de notre projet seront expliquées. Tout d'abord, la problématique en radionavigation sera abordée, ainsi que la définition des différents signaux utilisés dans le positionnement. Ensuite, nous expliquerons notre méthodologie en retraçant l'historique du SDR et l'environnement choisi pour effectuer le développement. Puis, une fois l'implémentation expliquée, l'architecture du récepteur numérique sous Simulink sera présentée.

Dans la deuxième partie de ce mémoire, nous aborderons le passage en temps réel de notre récepteur et ses effets sur notre modèle. Certains paramètres de base doivent être mis à contribution pour le fonctionnement correct du récepteur.

Finalement, dans la dernière partie, les performances, en cosimulation et en temps réel, de notre récepteur seront présentées et analysées. Et les perspectives d'optimisation de notre système seront aperçues par une brève analyse.

**Contribution**

Le développement d'un récepteur numérique fonctionnel demande la contribution de plusieurs intervenants. Le projet fut élaboré il y a quelques années par le prof. Landry et plusieurs personnes y ont participé. De l'étude de discriminateurs à la conception en temps réel, ce projet couvre un large éventail de connaissance.

Après l'étude de la robustesse et des performances sous Simulink du récepteur numérique par M.Ilie, le modèle pouvait alors être testé en temps réel. Plusieurs changements s'opéreront autour du modèle de base, afin de pouvoir réussir à implémenter le récepteur numérique en temps réel. Ma contribution à ce projet constitue justement cette transformation. Construire un projet à partir de zéro est une chose, mais en transformer un pour le passer à l'étape suivante peut s'avérer un défi tout aussi grand. Il faut tenir compte des éléments existants, et redéfinir les paramètres de l'environnement dans lequel on travaille. L'énorme bagage de connaissance acquis durant ce projet sera démontré tout au long de ce mémoire.

# CHAPITRE 1

# STATUT ACTUEL ET ÉVOLUTION DU GNSS

## 1.1 Principes de base en radionavigation et ses problématiques

Depuis le début des temps, l'homme a toujours été un être dont les déplacements et le transport furent une préoccupation primordiale. Ainsi, si l'on regarde la définition de la navigation, cela ne veut pas simplement dire voyager sur les mers, comme nous l'entendons souvent. La navigation, c'est l'art de se rendre à une destination donnée par la détermination de la position. Ainsi, on peut appliquer cette définition sur mer, dans l'air, sur terre et dans l'espace.

Au fil du temps, l'humain a développé de nombreux outils afin de s'aider à se positionner dans l'espace. Les Grecs, les Égyptiens et les Mésopotamiens faisaient appel aux astres dans le ciel tandis que, durant la Renaissance, les Européens utilisaient des sextants. Jusqu'à quelques décennies encore, la boussole était de rigueur pour tout déplacement en zone peu connue.

Les besoins s'étant développés, la technologie a suivi elle aussi. La nécessité d'augmenter la précision des outils, de diminuer le temps d'attente des coordonnées, a engendré des développements importants dans le monde de la navigation. D'outils simples, comme la boussole, la complexité des nouveaux outils implique une plus grande recherche et une plus grande connaissance des techniques. Et c'est dans cette sphère qu'est arrivé la radionavigation par satellites, tel le système américain nommé GPS pour Global Positioning System

Comme son nom l'indique, la radionavigation est le calcul de la position d'un mobile par des ondes radio. Connaissant la position de la source, la position à déterminer est calculée en rapport avec le temps de propagation des ondes électromagnétiques.

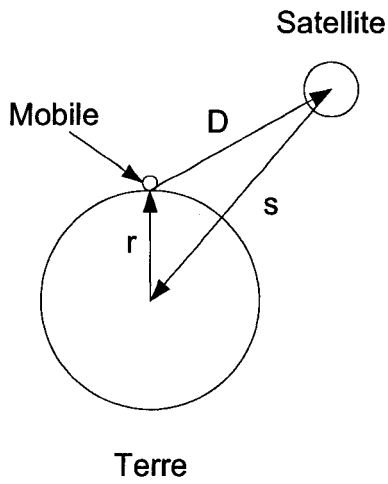



Figure 1 Schéma vectoriel du positionnement

Sachant la vitesse de propagation $c$ d'une onde électromagnétiques, la différence entre les temps $t_1$ et $t_2$ nous permet de savoir quelle distance l'onde a parcouru entre les points 1 et 2. Dans la situation qui nous concerne, le point 1 est le point d'émission du signal alors que le point 2 est la réception de ce même signal. La distance est calculée comme suit :

$$D = c * (t_1 - t_2) \tag{1.1}$$

Cette distance $D$ entre la source émettrice et le récepteur permet, avec l'aide d'autres sources émettrices, de calculer la position du récepteur grâce à la méthode de triangulation.

En général, nous n'avons besoin que de trois repères (dans notre cas, des signaux) afin de déterminer la position exacte d'un mobile. Cependant, dans la réalité, comme les distances sont grandes, nous devons tenir compte des valeurs du temps d'émission du signal de la source. Comme les horloges ne sont pas toutes synchronisées, il en résulte une erreur sur la distance mesurée. Apparaît alors la nécessité d'avoir une quatrième source émettrice, diminuant ainsi l'ambiguïté sur la valeur du positionnement calculé. De plus, la **Figure 1** nous montre un calcul de position sur un seul plan, or, dans la radionavigation, nous avons plutôt des sphères (puisque nous sommes dans le cas tridimensionnel) et non un plan comme trajectoire, que ce soit pour le satellite ou pour le mobile.

Dans les constellations satellitaires, la position de l'émetteur, c'est-à-dire le satellite, est donnée en fonction du centre de la Terre en coordonnées ECEF (Earth-centered Earth-Fixed). De façon vectorielle, la distance entre le satellite et le mobile se calcule par la relation suivante, tel que décrit par Kaplan **[1]** :

$$D = \|s - r\| \tag{1.2}$$

où **s** est la distance du satellite (émetteur) par rapport au centre de la Terre et **r**, la position du mobile (récepteur) par rapport au centre de la Terre. Tel que mentionné plus haut, le temps du satellite n'est pas synchronisé avec le temps du mobile, ce qui occasionne un délai tel que démontré à la **Figure 2**.

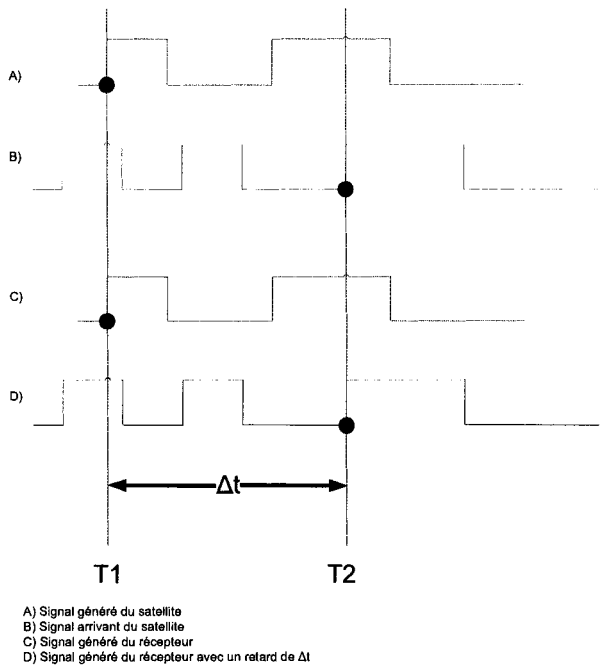



Figure 2 Introduction du délai dans la transmission des données

Le terme calculé par la dernière relation est donc la pseudo-distance, ρ.

$$\rho = c * \left( t_{2MOBILE} - t_{1MOBILE} \right) = c * \Delta t_{MOBILE} \tag{1.3}$$

La différence d'horloge entre le satellite et le mobile s'écrit comme $\Delta t_{ERR} = t_{1SATELLITE} - t_{1MOBILE}$, et la distance $d$ entre le satellite et le mobile devient alors :

$$d = c * \Delta t_{MOBILE} + c * \Delta t_{ERR} = \rho + c * \Delta t_{ERR} \tag{1.4}$$

La nécessité d'avoir une quatrième source apparaît dans cette équation car nous avons une variable additionnelle qui s'ajoute, demandant ainsi une équation de plus pour la résolution du système. Nous avons donc le système d'équations suivant :

$$
\begin{aligned}
D_1 &= \sqrt{(x-x_1)^2+(y-y_1)^2+(z-z_1)^2}+c*\Delta t \\
D_2 &= \sqrt{(x-x_2)^2+(y-y_2)^2+(z-z_2)^2}+c*\Delta t \\
D_3 &= \sqrt{(x-x_3)^2+(y-y_3)^2+(z-z_3)^2}+c*\Delta t \\
D_4 &= \sqrt{(x-x_4)^2+(y-y_4)^2+(z-z_4)^2}+c*\Delta t
\end{aligned}
\qquad \textbf{(1.5)}
$$

Ce système est, bien évidemment, en cordonnées cartésiennes, et les positions des satellites sont données par $x_i$, $y_i$ et $z_i$, où *i* indique le numéro du satellite dans sa constellation respective. Il y a plusieurs méthodes afin de résoudre ce système d'équation, mais dans les récepteurs GPS, on privilégie la linéarisation des équations de navigation ou le filtre de Kalman afin de calculer la position du mobile. Dans un système idéal, nous n'aurions de besoin que des trois équations pour résoudre notre problème, i.e trouver *x,y* et *z*. Cependant, l'arrivée d'une erreur au niveau de l'horloge nous impose une quatrième source émettrice, pour enlever l'ambiguïté sur le temps.

Ainsi, pour avoir une quatrième source afin de calculer le positionnement d'un mobile, les systèmes de positionnement par satellites doivent avoir un nombre minimum de quatre satellites visibles et ce, en tout point sur la surface de la terre et en tout moment. De plus, quatre étant un nombre minimal, il est possible que l'erreur calculée dans la position soit élevée. Ainsi, pour en obtenir une meilleure précision, le récepteur doit voir un nombre de satellites supérieur à 4 afin de valider ses calculs de position. La **Figure 3**, obtenue avec un simulateur de constellation, nous montre la visibilité des satellites GPS, à un moment donné, sur la surface de la Terre avec un masque d'élévation de 10 degrés (les satellites avec une élévation inférieures à 10 degrés ne sont pas visibles au récepteur). Les zones bleues correspondent au minimum de satellites, soit 4, alors que les zones rougeâtres indiquent le maximum de satellites visibles, soit 10.

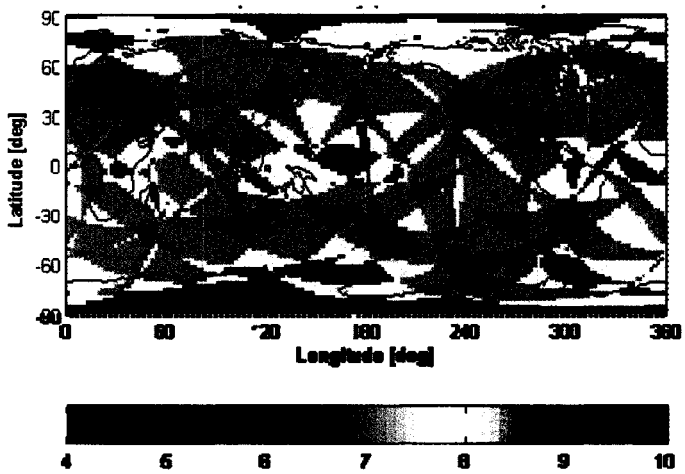



Figure 3 Visibilité spatiale des satellites GPS (masque d'élévation de 10°)

De plus en plus d'applications demandent l'utilisation d'un positionnement exact. Que ce soit au niveau des cellulaires ou de la localisation des avions, par exemple, le marché exige de plus en plus un positionnement précis. Pour palier à cette demande, puisque le système GPS a des limites de précision, plusieurs pays ont décidé de créer leur propre constellation de satellites de positionnement. L'augmentation du nombre de constellations (GPS, Galileo, Glonass, Beidou), entraîne nécessairement une augmentation du nombre de signaux de positionnement disponible. Les récepteurs de positionnement actuels n'ont pas été conçus pour utiliser tous ces signaux, d'où la nécessité de développer de nouveaux récepteurs. Différentes architectures ont été proposées à travers les années sur des récepteurs logiciels performants, qui peuvent s'ajuster très rapidement. Les récepteurs de forme logicielle permettront un ajustement rapide et efficace par rapport à l'ensemble des récepteurs GPS sur le marché. Combiné aux nombres grandissant de signaux, la complexité des calculs grandissante, le temps de développement des récepteurs et ses coûts suivent la courbe et tendent à augmenter, ce qui justifie l'apparition d'un développement logiciel au niveau de la radionavigation.

## 1.2 Activités internationales de modernisation et évolution du GNSS

Pour répondre à cette demande grandissante, la Chine ainsi que la Communauté Européenne ont décidé de produire leur propre système de positionnement. Les Américains, quant à eux, ont décidé de moderniser leur système GPS afin d'introduire plus de signaux pour les applications civiles et militaires.

Placés en orbite en Octobre 1982, les trois satellites de Glonass (GLObal Navigation Satellite System) de départ font toujours parti de la constellation russe, qui aujourd'hui en comporte 24, mais seulement 17 fonctionnels. Complétée en 1985, la constellation de Glonass se situe à une orbite de 19 100 km du centre de la Terre et les satellites ont une révolution de 11 heures et 15 minutes. La constellation est séparée en 3 orbites comprenant 8 satellites**[22]**. Deux signaux sont émis par les satellites : un de haute précision L2 (pour les applications militaires) et un autre de précision standard L1 (pour les applications civiles). Le signal de précision standard utilise le mode FDMA (*Frequency Division Multiple Access*) pour générer ces signaux ayant comme fréquence minimum 1602 MHz et un différentiel de fréquence de 0,5625 MHz pour les canaux subséquents **[23]**. Selon le plan proposé par les Russes, les 24 satellites deviendront opérationnels en 2010 et le développement est assuré autant par les Russes que par les Indiens**[22]**.

Nommé La Grande Ourse, ou Beidou en chinois, en l'honneur de la constellation d'étoiles du même nom, le système de positionnement de la Chine a été lancé le 30 octobre 2000 dans sa première version. Constitué de seulement 3 satellites, le système chinois est géostationnaire, à l'inverse des autres systèmes similaires, ce qui fait en sorte qu'il est limité au niveau de son espace d'opération**[24]**. Ces satellites (Beidou 1A, 1B et 1C) sont placés à, respectivement, 140° de longitude à l'est de la Chine, 80° de longitude à l'est et, pour le 1C à 110,5° de longitude à l'est**[25]**. Voyant la communauté

internationale embarquée dans l'expérience Galileo, la Chine a fait pareil en 2003 en investissant près de 300 millions de dollars américains dans le projet.

Devant cet investissement massif des Chinois, plusieurs autres pays ont décidé d'embarquer dans le développement de Galileo, dont la Russie. À la base, Galileo est la contrepartie européenne du système GPS américain. Devant l'augmentation de la demande pour un tel système, la France, l'Italie, l'Allemagne et la Grande-Bretagne ont unis leurs efforts pour développer un système de positionnement propre à l'Europe. À ce jour, plus d'une vingtaine de pays participent au développement de Galileo. Après un départ difficile, l'accord pour le lancement officiel de Galileo fut signé en mars 2002. Il fut décidé que la constellation serait à 23 222 km séparée en 3 plans orbitaux de 56° d'inclinaison de 9 satellites opérationnels par orbite**[26]**. De ce système, quatre(4) services seront offerts : Open Service (OS), Commercial Service (CS), Public Regulated Service (PRS) et Safety of Life (SoL). Chacun a ses fréquences et caractéristiques propres tel que montré au **Tableau I[26]**.

Tableau I

Caractéristiques des services Galileo

| | **Fréquence d'opération** | **Précision** | **Disponibilité** |
|---|---|---|---|
| **Open service (OS)** | 1164 – 1212 MHz<br>1563 – 1591 MHz | Verticale : <35m<br>Horizontale : <15m | Gratuit à tous |
| **Commercial (CS)** | 1260 – 1300 MHz | <1m | Payant à tous |
| **Public Regulated Service (PRS)** | 1164 – 1214 MHz | Idem que l'OS | Réservé (armée, police,…) |
| **Safety of Life (SoL)** | 1164 – 1214 MHz | Idem que l'OS | Réservé (armée, police,…) |

De 24 satellites pour la constellation GPS, ce nombre augmentera à 51 avec la constellation de Galileo. Et le marché technologique doit s'adapter aux demandes

grandissantes des consommateurs. La prochaine figure nous montre les futurs signaux de positionnement satellitaire GPS, tel que présenté par Shaw**[19]**.

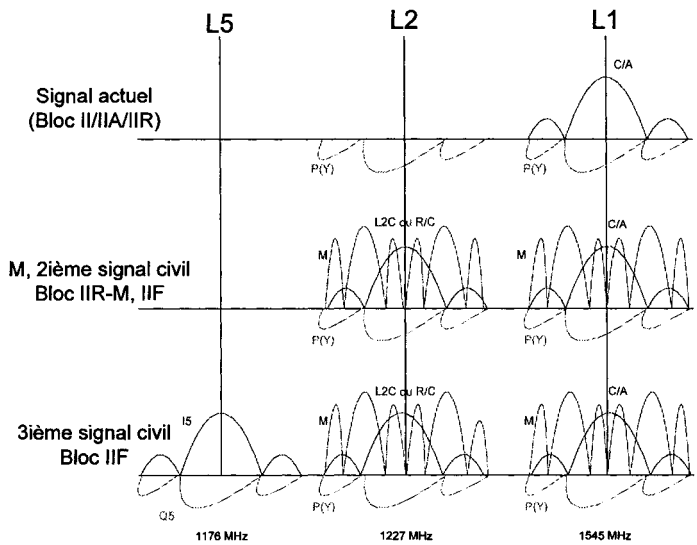



Figure 4 Modernisation des signaux GPS

Ces signaux, schématisés à la **Figure 4**, permettront de modéliser l'architecture du récepteur GPS et serviront de paramètres dans la conception du récepteur de base. Lors du passage en temps réel de notre modèle, les caractéristiques des signaux satellitaires devront être respectées afin que notre récepteur puisse les démoduler. Dans notre modèle, l'incorporation en simulation de plusieurs signaux de positionnement des différents satellites, canaux de transmission et récepteur nous permettra d'augmenter son efficacité et son réalisme.

Augmentant la disponibilité des signaux, la précision des récepteurs deviendra de plus en plus grande, encore faut-il que les récepteurs puissent suivre cette évolution. Ainsi, le récepteur doit faire un recouvrement au niveau de la porteuse (en phase) et au niveau du délai pour récupérer les données envoyées par les satellites. Pour les applications

actuelles, les récepteurs devront être plus précis lors de la démodulation, que ce soit au niveau de la phase ou du délai.

Afin de respecter les normes d'émission des signaux radio-électriques et dans le but de réduire les interférences entre les signaux de télécommunication actuels et les signaux de radio-navigation, l'ITU (*International Telecommunication Union*) a déterminé la densité spectrale du signal GPS. Elle ne doit pas dépasser la valeur de -154 $dBW/m^2$ dans une bande de 4kHz pour une fréquence comprise entre 1.525 et 2.500 GHz. Cette imposition nous oblige à étaler le plus possible notre signal, et utiliser la technique d'accès multiple à répartition par code (CDMA). En utilisant ce principe, la démodulation de notre signal doit reposer sur la corrélation du signal reçu avec la réplique du signal transmis pour récupérer l'information utile.

Les trois constellations satellitaires (GPS, Galileo et Glonass) opéreront parallèlement avec des structures et des codes différents, augmentant la quantité de signaux de positionnement disponible pour les usagers. La **Figure 5** nous montre ce que sera le nombre de satellites visibles lorsque les trois constellations seront opérationnelles. On peut remarquer la différence avec la **Figure 3** quant au nombre minimum et maximum de satellites visibles passant respectivement de 4 à 16 et de 10 à 29. Cependant, cette augmentation du nombre de satellites et de signaux demande aux récepteurs de se moderniser en proposant des architectures hybrides flexibles dans les futurs récepteurs numériques.

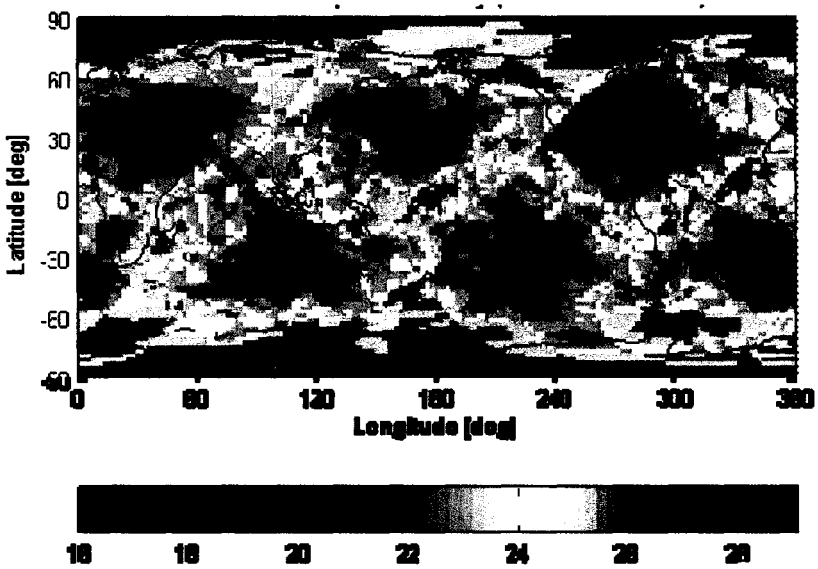



Figure 5 Visibilité des satellites GPS, Galileo et Glonass (masque d'élévation de 10°)

## 1.3 Présentation des signaux GNSS et leur évolution

À la base, le GPS fut conçu par le Département de la Défense des États-Unis dans les années 70 pour des applications militaires seulement. Voyant l'efficacité au niveau du positionnement du système militaire, le gouvernement américain a déposé un plan de développement pour les applications civiles sur la radionavigation basé sur ce système à l'aide du Département de la Défense et du Département du Transport**[28]**. Le département du Transport du gouvernement américain était intéressé par ce système pour le positionnement de ses véhicules, tel que présenté à une conférence en 2002 par Karner**[18]**. C'est à partir de cette date que le système GPS est entré dans sa phase de modernisation. Voici quelques données empiriques sur la constellation satellitaire GPS :

- Nombre de satellites : 24
- Nombre d'orbites : 6
- Inclinaison : 55°

- Altitude : 20 200 km

Les paramètres de la constellation GPS ont été calculés en fonction d'une visibilité optimale. Ces paramètres permettent d'avoir un minimum de 4 satellites visibles en tout temps au sol, répondant ainsi aux critères minima exprimés par la relation 1.5 de la section précédente. Chaque satellite est équipé de trois(3) horloges atomiques précises permettant un calcul précis de la pseudo-distance, élément important dans le calcul du positionnement tel que discuté à la section 1.1.

À ses débuts, le GPS utilisait 3 types de signaux, un signal à une fréquence L2, utilisé par les militaires seulement, avec un code P de codage (réservé aux applications militaires), et 2 autres signaux à une fréquence L1, soit un avec le code P militaire et un deuxième avec un code C/A pour les applications civiles. Ce système entraînait beaucoup d'erreurs dues au délai ionosphérique. Un programme de modernisation du système GPS a été présenté afin de diminuer ces erreurs. Chaque satellite émet ces trois signaux. Le code C/A (*Coarse Acquisition Code*), d'usage civil, a un taux de 1.023 Mbps (mégabits par seconde) et une période de 1 ms. Le code P (*Precision Code*), d'usage militaire, a un taux 10 fois plus élevé que le code C/A et une durée de 7 jours. Ces codes furent calculés afin d'obtenir le minimum d'intercorrélation entre les satellites, chacun ayant son propre code. Grâce à cette caractéristique, le CDMA (*Code Division Multiple Access*) est tout désigné pour récupérer le bon signal, comme le souligne Kaplan**[1]**.

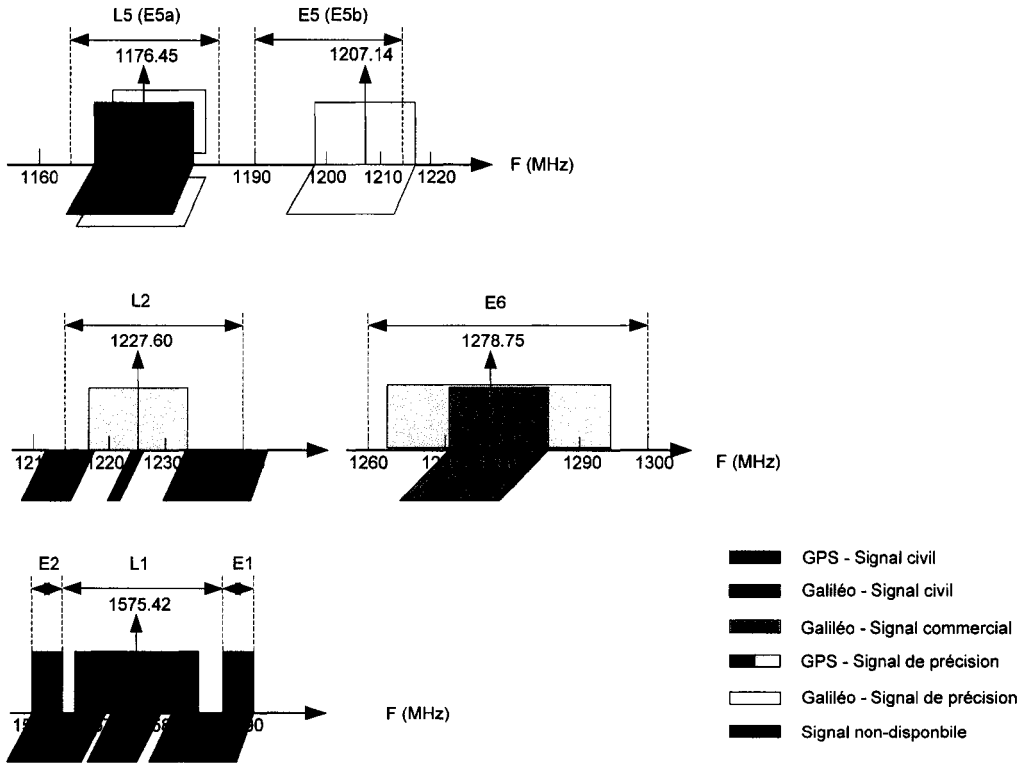



Figure 6 Spectre des signaux GPS et Galileo

Chacun de ces signaux a une fonction précise et l'objectif est surtout d'augmenter la robustesse ainsi que la précision du positionnement. Par la figure ci-dessus, on peut remarquer que les signaux agiront dans une bande de fréquence assez large (entre 1160 Mhz et 1590 MHz). Heureusement, certaines bandes, comme E5a/L5 et E2-L1-E1/L1, utiliseront la même fréquence porteuse. La génération des ces signaux sera décrite dans le CHAPITRE 3 pour le GPS, mais, à la **Figure 6**, on peut en avoir un aperçu du spectre qu'ils utiliseront dans le domaine fréquentiel.

Cette caractéristique des signaux facilite la conception des récepteurs. Cependant, elle entraîne une augmentation des interférences mutuelles, demandant ainsi une meilleure gestion et une meilleure précision des signaux démodulant au niveau du récepteur. Les récepteurs devant satisfaire ces normes se devront d'être rapides, précis, efficaces et

robustes. Le développement de ces récepteurs hybrides doit, lui aussi, être rapide et évolutif.

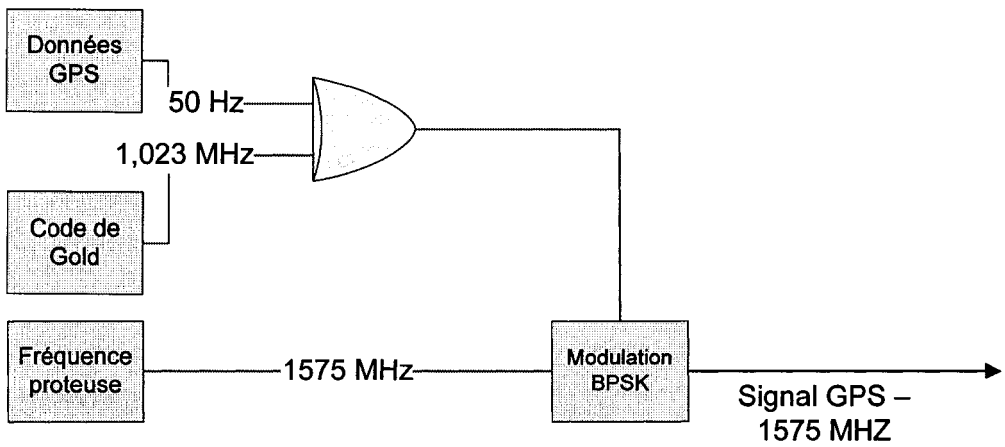



Figure 7 Génération du signal GPS L1

La **Figure 7** montre la génération d'un signal GPS L1 ainsi que les fréquences respectives des différents signaux (code, données, porteuse) y intervenant, qui seront les signaux satellitaires que nous considérerons dans notre projet. Les concepteurs du signal GPS ont ajouté un aspect important au signal, c'est-à-dire la synchronisation des signaux imbriqués. Chaque signal, soit le code de Gold, les données et la fréquence porteuse ont un dénominateur commun (ce sont donc des signaux cohérents), ce qui simplifie les calculs de modulation et permet de détecter rapidement des erreurs lors des tests effectués.

Les récepteurs actuels n'ont pas été conçus pour utiliser tous les nouveaux signaux, d'où la nécessité de développer de nouveaux récepteurs. Différentes architectures ont été proposées à travers les années sur des récepteurs logiciels performants qui peuvent s'ajuster très rapidement. Les récepteurs de forme logicielle permettront un ajustement rapide et efficace par rapport à l'ensemble des récepteurs GPS sur le marché. Combiné au nombre grandissant de signaux, la complexité des calculs grandissant, le temps de

développement des récepteurs et ses coûts suivent la courbe et tendent à augmenter, ce qui justifie l'apparition d'un développement logiciel de la radionavigation.

## 1.4 Besoin d'interopérabilité des systèmes GNSS

Avec le nombre grandissant de constellations satellitaires de positionnement, toutes indépendants les unes des autres, chacun doit tenir compte des autres afin de ne pas entrer en interférence avec les autres signaux. En regard de cet aspect, Galileo a pris comme objectif l'optimisation, au niveau de l'usager, des services Galileo avec les autres systèmes GNSS. Cet objectif se définit clairement dans le choix des fréquences, du référentiel temporel et de la structure des signaux (au niveau du codage entre autre) afin non seulement d'éviter les interférences mais aussi que le signal Galileo vienne bonifier, pour l'usager, le signal GPS.

Ainsi, comme les applications demanderont de plus en plus de précision, les récepteurs devront traiter différentes structures de signaux pour atteindre cet objectif. Les architectures des récepteurs devront s'adapter à l'infrastructure changeante des signaux de navigation. Actuellement, les récepteurs ne sont pas adaptés à ces changements, et leurs architectures doivent être modifiés. C'est pourquoi les récepteurs numériques deviendront de plus en plus intéressants puisqu'ils sont flexibles et rapidement reprogrammable, contrairement aux récepteurs analogiques.

## 1.5 Conclusion

Le marché de la radionavigation arrive à un tournant dans son histoire. L'arrivée de Galileo, la modernisation des signaux GPS et l'interopérabilité des systèmes demandent à ce que le développement des nouveaux récepteurs soit plus complexe et flexible. Plusieurs chercheurs dans le domaine analysent les signaux émis et la façon de les récupérer mais peu prétendent que leurs résultats sont finaux.

Dans cette optique, en se basant sur le succès du SDR, il devient indispensable de développer un environnement logiciel et matériel adapté à la radionavigation afin de créer les nouveaux récepteurs et de répondre aux besoins du marché. Une analyse des différents paramètres et des ressources matérielles utilisées est la première étape pour en arriver à un premier prototype, s'inscrivant dans la démarche d'un récepteur conçu de façon purement logiciel. À partir d'un simulateur, plusieurs étapes et analyses seront nécessaires afin d'en arriver à un récepteur rapide, robuste et facilement reconfigurable, un récepteur créé à partir du concept de « Software Defined Navigator »

# CHAPITRE 2

# PRÉSENTATION DU PROJET SDN « Software Defined Navigator »

## 2.1 Historique du concept du SDR, « Software Defined Radio »

L'évolution de la radionavigation demandant beaucoup de ressources, l'optimisation des ressources, autant matérielle que financière demeure un enjeu important dans le développement des nouveaux produits et services. Le domaine de la radionavigation n'est pas le premier domaine où son évolution rapide exige une nouvelle méthodologie de travail. Le domaine des radios télécommunications a dû s'adapter à la nouvelle réalité dans les dernières décennies.

Les origines de la communication radio se situent au 19$^{ième}$ siècle. Tout d'abord, les équations de Maxwell qui ont permis de comprendre et d'utiliser les ondes électromagnétiques. En 1876, Alexander Graham Bell invente le téléphone, outil qui a percé près de 99 % des foyers canadiens. En 1887, Heinrich Hertz conclut ses premières expériences et découvre ce que l'on appelle maintenant les ondes radio, les ondes « hertziennes ». Enfin, Marconi, en 1896, réalise la première communication radio. Dans les années 1960 et 1970, la demande en ondes radio est tellement grande, qu'on invente de nouveau protocole de communication tel l'allocation dynamique des canaux de transmission. C'est en 1978 que Bell Telephone (US) présente le premier réseau de communication cellulaire, devenu un standard en 1982**[20]**.

Vers la fin des années 80, le domaine des ondes radio a connu un essor comparable à celui de la radionavigation en ce moment. De cette effervescence est né un concept, celui du « Software Defined Radio » communément appelé SDR.

La définition du SDR est un système de radio communication qui utilise une partie logiciel pour la modulation et démodulation d'un signal radio **[20]**. Le système performe donc une majorité d'opérations sur le signal radio dans un microprocesseur, qu'il soit à l'intérieur d'un ordinateur ou sur un processeur reprogrammable. L'objectif de ce système est de produire un nouveau protocole de modulation/démodulation simplement par la compilation d'un nouveau logiciel.

Le concept idéal d'un tel système serait de raccorder un convertisseur analogique numérique à une antenne radio, tel que montré à la **Figure 8**. Les données numériques ainsi recueillies sont envoyées à un ordinateur qui, à l'aide d'un logiciel, transforme ces dernières en informations utiles à l'usager. Cette partie traite de la démodulation seulement. On pourrait facilement construire une modulation radio à partir du même principe, mais en attachant un convertisseur numérique-analogique à l'antenne pour l'émission des ondes radio.

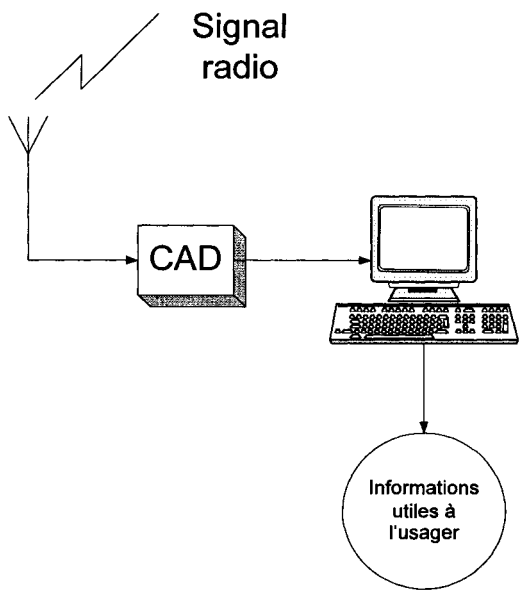



Figure 8 Concept du SDR

L'armée américaine est la première à avoir développé le concept de SDR. Ce projet s'appelait SpeakEasy[**20**]. Le but premier de ce projet est d'utiliser des processeurs reprogrammable afin d'émuler plus de 10 fréquences radio militaires opérant dans les bandes de fréquences entre 2 et 200 MHz. Un aspect important de ce concept est l'incorporation de nouveaux types de codages et de techniques de modulation dans le futur, afin de suivre le développement dans ces domaines.

Au début des années 90, le projet SpeakEasy entrait dans sa première phase. L'armée américaine voulait avoir une radio pouvant œuvrer dans le spectre de 2 MHz à 2 GHz afin de pouvoir communiquer avec toutes sortes de radios : les radios navales, les radios de l'aviation, les radios de l'armée de terre ainsi que les satellites. D'autres objectifs plus particuliers étaient la conception d'un autre format de signal radio en 2 semaines, et de construire une radio en connectant différents modules adaptables.

Ce projet a atteint ces objectifs, mais il a démontré quelques failles au niveau du traitement des signaux reçus, ne pouvant supporter qu'une seule communication à la fois. Le récepteur était composé d'une antenne, alimentant un amplificateur et un mélangeur. Le signal ainsi créé était amené à un convertisseur analogique numérique relié via un bus à plusieurs processeurs Texas Instrument de la série C40. Du coté de la source du signal, un convertisseur numérique analogique relié à un bus PCI était branché à une antenne radio et un amplificateur. Ce design est devenu, avec le temps, un modèle standard pour un projet de type SDR.

La deuxième phase de ce projet se voyait un peu plus audacieuse par les objectifs fixés. Le premier objectif était de construire une architecture reprogrammable plus rapide, dans une architecture ouverte, avec plusieurs canaux à protocoles différents. De plus, on constata que ce genre de système pouvait réduire les coûts et ils ont voulu faire un prototype plus petit et moins coûteux. Dans un projet basé sur un plan triennal, on obtint des résultats dans les premiers 15 mois du programme.

L'architecture avait été séparée en différents modules :

- Contrôle du modem : module gérant la modulation et démodulation des signaux
- Processeur du signal : module servant à moduler le signal
- Processeur de cryptographie : module servant à coder le message
- Multimédia : module pour analyser la voix humaine
- Interface : module servant de GUI (*Graphical User Interface*)
- Routage : module de réseau afin de gérer les communications entre les modules
- Contrôle : module de contrôle du modèle

Ces différents modules communiquaient sur un bus PCI entre eux, d'où la nécessité d'avoir des modules de contrôle et de routage. Pour la première fois dans l'histoire du concept de SDR, on utilisait des FPGA *(Field Programmable Gate Array)* dans le modèle. À partir de ce moment, le temps de programmation du FPGA devint alors un facteur à prendre en considération.

Dès lors, on commence de plus en plus à se regrouper afin de produire un système ou une architecture commune. Cette architecture est le *Software Defined Radio* (SDR). Le principe du SDR est simple : utiliser un logiciel afin de moduler et démoduler les signaux radio avec un but ultime d'avoir un signal reçu ou transmis par le simple fonctionnement d'un logiciel. Une partie importante du traitement du signal est effectuée à l'intérieur d'un processeur (qu'il soit à l'intérieur d'un ordinateur ou monté sur une plate-forme à part), et l'analyse peut être faite autant par des instruments de mesure que par l'étude des données logicielles. La composition matérielle d'une radio « *software defined* » se définit par les outils suivants :

- Modulateur/Démodulateur superhétérodyne RF
- Convertisseur analogique numérique et convertisseur numérique analogique

Cette simplicité de montage fait en sorte que les plateformes de développement sont moins coûteuses. Et comme le traitement se fait par voie logicielle, il est plus facile de l'implémenter à travers d'autres plate-formes matérielles. Le projet GNU Radio a tiré rapidement avantage de cette fonctionnalité en distribuant à travers le monde une SDR gratuit avec tous les outils pour construire et déployer une SDR. Son aspect le plus pratique, son coté reprogrammable, permet à toute personne pouvant modifier un code informatique de créer sa propre SDR, prouvant encore une fois la flexibilité du projet.

## 2.2 Origine et approche utilisé dans le projet SDN

En 1998, le gouvernement américain a développé un plan de développement sur la radionavigation**[28]**. Ce rapport à été entendu et révisé par le département de la défense nationale et le département des transports. Ce rapport recommandait que le système GPS devienne le seul système de positionnement utilisé par le département des transports du gouvernement américain. Le système était perçu comme étant fiable et invulnérable, le développement d'un autre moyen de positionnement était purement inutile. Les systèmes de positionnement existant déjà, tel que LORAN *(Long RAnge Navigation)*, ILS *(Instrument Landing System)* et NDB *(Non-Directional Beacon)* par exemple, se devaient d'être mis sur les tablettes. Les économies engendrées par cette décision devaient être importantes pour le gouvernement américain. Cependant, la vulnérabilité du système GPS a été mise en évidence et toutes les conclusions de ces rapports ont dû être révisées.

La possibilité de perdre le signal GPS implique qu'une solution alternative doit être trouvée afin de permettre à un mobile de connaître sa position en tout temps. Dans le cas d'un avion, par exemple, le positionnement est important afin d'assurer un atterrissage sécuritaire et ce, peu importe les conditions climatiques et/ou le trafic. Mais les systèmes

existants, n'ayant pas été développés pour les raisons mentionnées plus haut, ne sont pas capables de fournir l'aide requise au niveau du GPS.

Le gouvernement américain et le département de la défense se devaient donc de trouver une solution conjointe avec le département des transports afin de palier à cette mauvaise interprétation. Soit ils modernisaient le système GPS afin de le rendre moins vulnérable, soit ils investissaient dans le développement d'autres systèmes**[28]**. La deuxième solution n'était pas réaliste à cause de l'investissement déjà fait à travers les années sur le système GPS. La modernisation du système GPS, tel que décrite précédemment dans le texte, était donc nécessaire afin de combler ces lacunes.

Les utilisateurs civils, dont le département des transports, ont seulement accès au code C/A du signal GPS, empêchant, par le fait même, de faire une correction double fréquence du retard ionosphérique. Les signaux L1 et L2 ne fournissent pas un spectre de protection totale et, étant noyés dans le bruit, ont des puissances faibles. Ceci engendre une vulnérabilité aux niveaux des brouilleurs et aux interférences. De plus, en région urbaine, les structures de béton et d'acier provoquent des interférences et des multi-trajets. Les brouilleurs spécialisés ainsi que les multi-trajets provoquent des interférences qui empêchent la démodulation correcte du signal par le récepteur, problème qui devra être traité par notre récepteur numérique.

Deux catégories de besoins ont été considérées pour la modernisation du système GPS, et de ces catégories des pistes de solutions ont été envisagées. Pour les applications militaires, de nouveaux signaux avec séparation spectrale et une puissance augmentée pour améliorer la protection, la prévention et la préservation du système sont les besoins les plus appréciés. Dans les applications civiles, les demandes étaient beaucoup plus nombreuses :

- Augmenter la disponibilité du signal
- Augmenter la précision du signal

- Augmenter la redondance du signal
- Ajouter 2 fréquences civiles :
    - Une pour la correction ionosphérique
    - Une pour les applications « Safety-Of-Life (SOL) »

Plusieurs options ont été envisagées dont celles-ci :

- ✓ Augmentation de la puissance
- ✓ Addition d'un code militaire aux bandes de fréquences L1 et L2
- ✓ Stations au sol supplémentaires
- ✓ Augmentation du nombre de satellites

Outre l'autorisation d'un nouveau signal civil à 10.23 MHz, les signaux L1 et L2 seront disponibles aux applications civiles. Les demandes de l'aviation exigent des caractéristiques très précises et il est décidé de créer un nouveau signal pour cette application, dans la bande L5, à 1176,45 MHz. De plus, on a crée un comité gérant le système GPS, l'Interagency GPS Executive Board (IGEB), duquel les membres serait issus des différents départements (Commerce, Intérieur, Justice, Agriculture, Etat, Transport, Défense et de la NASA) utilisant le GPS afin que les besoins de chacun soit respectés.

Le programme GPS III offrira un service encore plus complet. Son objectif est de satisfaire les besoins autant civils que militaires. Une architecture nouvelle et une meilleure valeur d'acquisition entreront en jeu et répondront aux besoins civils pour le positionnement spatial, la navigation et le système de référence temporel jusqu'en 2030. L'armée de l'air américaine désire que GPS III offre un potentiel de meilleure robustesse aux brouilleurs en ajoutant 2 bandes pour le code M sur les canaux L1 et L2.

Tous ces changements impliquent une évolution technologique relativement importante, autant au niveau matériel qu'au niveau logiciel. En fait, au niveau matériel, la demande est sans cesse grandissante. Pensons seulement aux largeurs de fréquence sur lesquelles on travaille. À des fréquences dépassant le gigahertz, on utilise des bandes de fréquences d'une largeur de 10 MHz.

En comparant l'évolution de la radionavigation et de la radio communication, on se rend compte de plusieurs similitudes :

- ✓ Demande grandissante de plusieurs domaines
- ✓ Évolution rapide de la technologie
- ✓ Incorporation du système dans plusieurs technologies existantes
- ✓ Demande de nouveaux services
- ✓ Demande d'augmentation de la qualité et de la robustesse du signal
- ✓ Création d'un organisme mondial pour la gestion des technologies

À l'intérieur du message transmis par les satellites, il y aura des différences au niveau du codage et de la distribution des données. Par exemple, dans le cas du GPS, il y a le code C/A et le code M tandis qu'avec Galileo, le code BOC (« Binary Offset Carrier ») sera privilégié. Donc, comme on le verra plus tard à la section 3.4, dans l'architecture des nouveaux récepteurs, chaque composante sera dédiée à une tâche en particulier puisque les signaux sont si spécifiques.

Pour l'intégration d'un nouveau service sur une technologie existante, deux options s'offrent aux chercheurs et/ou programmeurs, tel que montré à la **Figure 9** :

1. Acheter tout le matériel requis afin de développer un prototype,
2. Trouver une plate-forme reprogrammable, alliant logiciel et matériel pour continuer le développement.

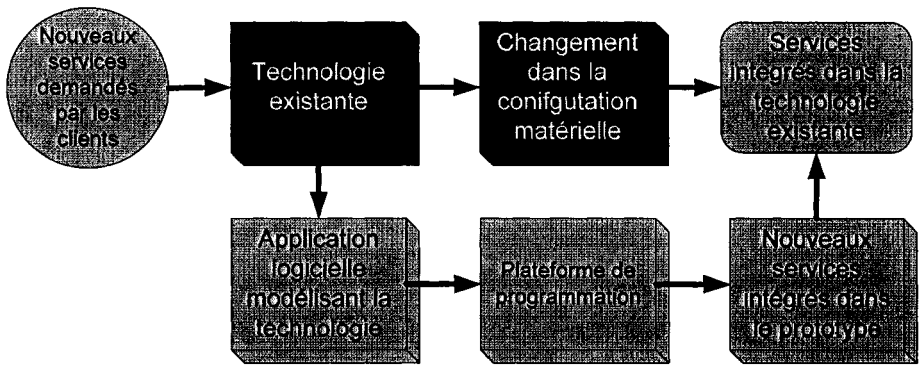



Figure 9 Schéma bloc de l'intégration d'un nouveau service

La première solution engendre des coûts assez élevés. Comme il y a peu de redondances dans la partie démodulation, le récepteur devra être composé de plusieurs parties indépendantes. De plus, les systèmes étant en évolution constante, il est difficile de prédire exactement le fonctionnement des systèmes de positionnement. Par exemple, pour le signal de Galileo, la Communauté Européenne hésite encore entre quelle modulation BOC (Binary Offset Carier) sera utilisée pour moduler le signal. Cependant, en mai 2006, une décision a été prise sur le type de modulation que supportera Galileo. Mais comme nos travaux étaient déjà effectués, nous ne tiendrons pas compte de cette décision dans ce travail. Il n'est pas impossible aussi qu'il y ait des modifications au niveau des services offerts. Chaque changement ou modifications apportées entraîne un changement dans le modèle et, par conséquent, dans les composantes requises pour la conception du prototype. Avec la valeur des pièces actuellement sur le marché et l'assemblage, le coût de cette option est assez élevé et peut satisfaire certaines compagnies mais comporte plusieurs points à éviter, tel la limite de développement et la difficulté d'adaptation.

La deuxième option, favorisée par les compagnies de développement en radiocommunication, constitue l'approche « software defined ». La combinaison

matérielle et logicielle engendre certains problèmes mais ceux-ci sont contournables par l'élaboration de protocoles de communication. Déjà, plusieurs fabricants de matériel investissent dans le développement d'outils de communication entre leurs composantes et un ordinateur. L'utilisation d'une plateforme de programmation unique permet aux développeurs de tester plusieurs aspects sans avoir à apprendre de nouveau les différents protocoles de communication. La plateforme doit cependant répondre à des caractéristiques précises et doit satisfaire aux normes les plus exigeantes de la technologie afin d'assurer un développement adéquat.

Dans le cadre de ce mémoire, l'approche « software defined » comporte plusieurs avantages. Au niveau des coûts, nous devons travailler avec un budget limité dans un cadre de laboratoire de recherche universitaire. L'achat d'une seule plateforme reprogrammable nous permet de réduire les coûts de recherche et développement. De plus, elle permet un prototypage rapide de notre modèle de communication satellite. Dans la prochaine section, nous vous présenterons l'environnement de travail et la plateforme de développement.

## 2.3 Environnement de travail et outils utilisés dans le projet SDN

L'environnement de travail est la clé du succès dans ce type de projet. La première étape est l'analyse du système. Cette étape permettra de déterminer les outils qui seront les plus adéquats et les plus performants pour la réalisation de notre prototype. Cette étape sera abordée plus en détail dans le CHAPITRE 4 mais, pour cette section, nous en garderons les grandes lignes. Tel que présenté dans le CHAPITRE 1, le signal GPS est en constante évolution mais les bases restent inchangées. Premièrement, la fréquence de la porteuse se situe à exactement à 1575.42 MHz. Une condition imposée par cette caractéristique est la fréquence d'opération (ou fréquence d'échantillonnage) des composantes électroniques contenues sur la plate-forme. Comme ce signal est porteur du code et des données, on doit, à son tour, l'échantillonner adéquatement afin de respecter

le théorème de Nyquist et recouvrir les données. Cette facette engendre une fréquence minimum au niveau des opérations.

Deuxièmement, dans notre modèle de récepteur GPS, nous utilisons plusieurs calculs arithmétiques, dont certains sont plutôt complexes. Ce facteur deviendra limitatif lorsque nous travaillerons en temps réel et devra être tenu en compte pour les différents délais que nous obtiendrons dans nos résultats.

Troisièmement, la rapidité des communications entre les différentes cartes devient importante. Puisque nous devons échantillonner à très haute fréquence et utiliser des algorithmes complexes, les protocoles de communication se doivent d'être rapides, efficaces et simples.

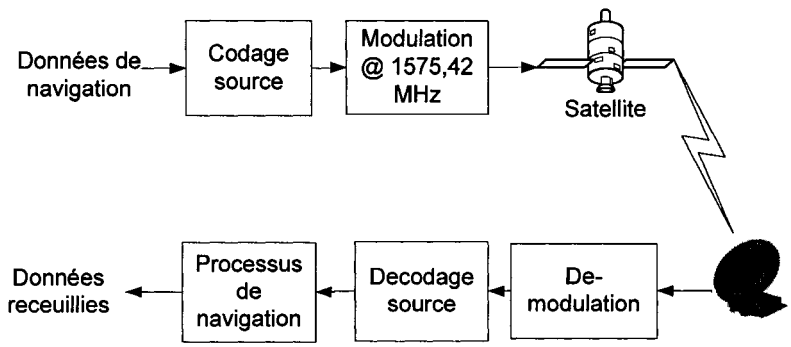



Figure 10 Chaîne de communication GPS

À la **Figure 10**, nous pouvons voir le système complet que nous voulons modéliser, des données de navigation envoyées jusqu'au décodage de ces données. Les buts poursuivis par notre projet sont les suivants :

1. Simuler et traiter en cosimulation (tel que montré à la **Figure 14** plus loin dans cette section) et en temps réel la source

2. Simuler et traiter en cosimulation et en temps réel le récepteur

On remarque à la **Figure 10** que nous avons de l'interaction entre plusieurs signaux (par exemple, codage, modulation, décodage,...). Ces processus devront être tenus en compte lors de l'analyse de notre modèle. Plusieurs processus ne requièrent pas les mêmes caractéristiques, certains étant plus complexe que d'autres. Par exemple, le taux d'échantillonnage devra être élevé dans certain cas (la démodulation), alors que la complexité des calculs sera la limitation d'un autre processus (le processus de navigation).

À travers la multitude d'offre de processeurs qui se trouve sur le marché, le choix d'un processeur demande une analyse plus poussée. Cette analyse, et le choix, seront basés sur quatre attributs de notre produit final : complexité, vitesse, prise de décision et algorithmes mathématiques. La complexité de notre système se mesure par deux caractéristiques : la diversité et le nombre de tâches que l'on doit exécuter. Pour un récepteur GPS, non seulement il y a beaucoup de tâches à exécuter mais elles vont de la modulation à l'algorithme complexe des discriminateurs. La vitesse aura aussi un rôle important à jouer, puisque nous allons traiter des signaux à fréquence élevée. Un processeur rapide est essentiel comme cible à notre méthodologie. De plus, les algorithmes mathématiques complexes contenus à l'intérieur de notre modèle seront nombreux, le processeur doit être capable de supporter des calculs mathématiques. De la démodulation d'un signal, jusqu'au calcul de l'erreur de phase par la méthode de la puissance avance moins retard normalisée, la liste des algorithmes mathématiques utilisés est longue. Ces algorithmes seront présentés au CHAPITRE 3.

Les valeurs recherchées dans les processeurs sont donc les suivants en ordre d'importance :

1. Vitesse
2. Complexité

3. Algorithmes mathématiques
4. Prise de décision

La partie prise de décision est une option secondaire sur laquelle on ne s'attardera pas. Les valeurs des résultats des calculs mathématiques effectués à l'intérieur du modèle serviront à guider les boucles au niveau des décisions à prendre. Cependant, un autre critère peut être ajouté à ce modèle, soit la mémoire disponible.

Basé sur ces critères de sélections, 2 choix possibles s'imposent : un FPGA (pour la vitesse et les algorithmes mathématiques) et un DSP (pour la complexité). Les avantages de chacun de ces processeurs pour notre modèle sont présentés brièvement dans les sections 4.4.1 et 4.4.2.

Au niveau de la vitesse, le problème peut se scinder en deux : la vitesse de l'horloge et les performances des fonctions. Le FPGA, sur ce point, l'emporte sur le DSP. Les fonctions peuvent être optimisées rapidement avec un minimum de coups d'horloge. Les DSP peuvent combler ce problème avec des bus haute vitesse mais ils n'atteignent pas la vitesse du FPGA. Les algorithmes mathématiques peuvent être optimisés sur les FPGA pour obtenir une rapidité des plus efficace. Le problème sur ce point réside dans la tête du programmeur puisque l'optimisation doit être faite par programmation, augmentant ainsi le temps de développement du processus. Sur l'autre point, les performances des fonctions, le DSP est mieux équipé que le FPGA car ses performances dépendent moins de la programmation mais plutôt des attributs matériels du processeurs. Au niveau de la complexité, le DSP possède des *pipelines* aidant la distribution des données, ce qui permet de pouvoir utiliser les bonnes ressources au bon moment. Et le DSP est rempli d'applications embarquées qui permettent l'introduction d'algorithmes complexes à l'intérieur des calculs.

Puisque les avantages sont intéressants dans le cas des deux différents processeurs, la combinaison des deux devient une avenue à envisager. Pour notre récepteur GPS, la vitesse de fonctionnement et la complexité sont des critères équivalents. Avec des protocoles de communications efficaces et une gestion optimale du transport des données optimale, la combinaison des deux processeurs répondrait exactement aux besoins identifiés. L'échange de données entre les processeurs pourrait avoir un impact minimal sur les performances du système et permettrait à notre plate-forme de tirer profit des caractéristiques des processeurs en implémentant des algorithmes complexes et rapides. La décision prise dans le cadre du projet est donc d'allier DSP et FPGA pour maximiser et optimiser les performances du récepteur GPS. De plus, en pensant à l'avenir du projet, la conciliation des deux processeurs permet d'envisager le développement de nouveaux systèmes combinés, tel que Galileo ou Glonass, sur cette même plate-forme, optimisant l'investissement monétaire effectué sur cette carte.

### 2.3.1 Matériel utilisée dans le projet

D'après les caractéristiques et objectifs énumérés plus haut, une recherche de plateforme de développement au niveau matériel fut entreprise. Il existe, sur le marché des cartes de développement, plusieurs possibilités qui peuvent répondre à nos besoins compte tenu de notre budget. Voici un résumé de nos besoins :

- Fréquence d'échantillonnage élevée
- Convertisseur analogique-numérique à large bande
- Processeur à algorithme complexe
- Entrée analogique
- Grande capacité d'espace mémoire
- Processeur à haute vitesse
- Port série (pour communiquer avec l'ordinateur)

Le choix d'une plateforme est crucial dans la démarche du projet et plusieurs options et compagnies ont été étudiées. La plateforme retenue est produite par Sundance Inc[14]. La plateforme se compose de quatre (4) parties :

1- Carte d'acquisition de données
2- Carte modulaire avec FPGA
3- Carte modulaire avec un DSP
4- Carte-mère

Le fait que la plateforme se compose de quatre (4) cartes nous donne une option de plus que prévue. En effet, puisque nous avons trois (3) modules distincts, une flexibilité au niveau de l'adaptation de la plateforme de développement offre la possibilité de mettre à jour le système, tel qu'observé par Avrunin et ass **[6]**. En effet, avec l'évolution grandissante des signaux et les changements continuels, certaines cartes choisies pourraient s'avérer rapidement désuètes. Cependant, des cartes modulaires mieux adaptées s'interchangeraient facilement avec la carte actuelle. Le fait de changer seulement une carte modulaire et non la plateforme entière est un atout majeur pour l'évolution du projet car il diminue grandement les coûts de développement.

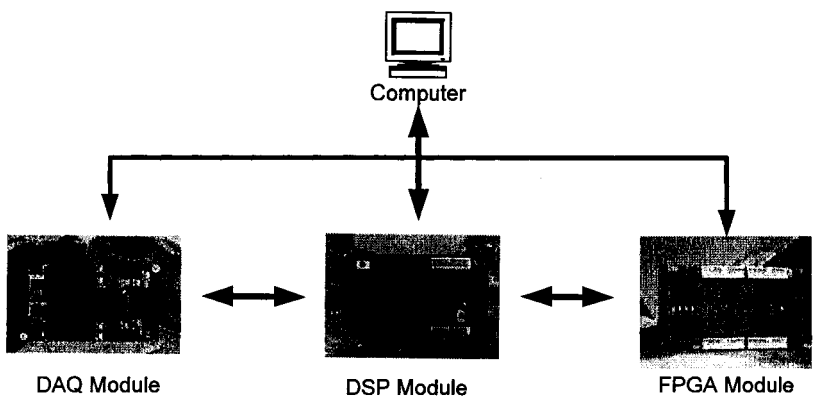



Figure 11 Interaction entre les cartes modulaires

La Figure 11 nous montre l'interaction entre les différentes cartes modulaires et l'ordinateur. Ces communications se font par l'intermédiaire d'une carte-mère sur laquelle sont déposées et branchées les cartes modulaires. De plus, une fois les modules fixés sur la carte mère, cette dernière est fixée sur le bus PCI de l'ordinateur afin de communiquer avec lui. Cette carte a une horloge interne de 33 MHz sur son bus PCI, une possibilité d'y installer quatre cartes modulaires et de la mémoire à partager entre ces différents modules pour la configuration.

### 2.3.2 Logiciels requis

Suite à la sélection du support matériel requis, les logiciels de programmation, de génération de code et de communication deviennent donc indispensables. Ces différents logiciels sont utilisés afin de convertir notre modèle en langage machine compréhensible pour les processeurs dans lesquels les programmes iront. Avant de déterminer les différents logiciels que nous utiliserons, nous devons entrevoir les différentes étapes par lesquelles notre modèle passera afin de terminer en langage machine.

1. Définition de notre modèle de base.
2. Changement du modèle pour le passage en temps réel : au niveau du processeur digital et au niveau du FPGA.
3. Génération des codes adaptés au DSP et au FPGA.
4. Encapsulation des codes à l'intérieur des protocoles de communication pour les processeurs.
5. Activation des codes et retour d'information.

Il n'existe pas, pour notre projet, un logiciel ayant la capacité de faire toutes ces étapes. Il nous faudra analyser notre modèle et les caractéristiques des logiciels afin de faire correspondre les deux. Ainsi, nous devons diviser notre travail en différents processus et sous processus. Cependant, le fait de faire intervenir plusieurs logiciels peut entraîner

des problèmes, au niveau de la compatibilité entre eux. Il n'est pas acquis que les fichiers résultant d'un logiciel de niveau hiérarchique supérieur soient compatibles avec celui des niveaux inférieurs. L'interaction entre 2 logiciels entraîne systématiquement des erreurs de communications ou d'échange de données. La réduction au minimum du nombre d'interaction favorisera un développement plus rapide et une diminution du nombre d'erreurs dans l'échange de données.

Pour la simulation de notre récepteur numérique GPS, la suite Simulink de Matlab, créée par MathWorks, possède des librairies détaillées qui nous permettent de diviser les processus de notre modèle en de simples opérations. Cette division des processus en tâches simples facilitera notre tâche au niveau de l'implémentation puisque les algorithmes utilisés seront simples à leur tour. De plus, l'environnement Matlab/Simulink offre rapidité, précision et flexibilité. Rapidité puisqu'il permet de simuler rapidement notre modèle, facilitant la validation de celui-ci. De plus, nous pouvons utiliser des fréquences d'échantillonnage élevées ce qui augmentera la précision de notre modèle. Comme plusieurs compagnies travaillent en collaboration avec MathWorks, plusieurs options s'offrent afin de travailler en temps réel. Que ce soit avec Real-Time Workshop, ou d'autres librairies, une panoplie d'options est incluse dans la suite de Matlab, offrant ainsi une flexibilité dans notre choix de matériel.

Maintenant, en analysant notre modèle, nous en arrivons à la conclusion que nous allons devoir séparer notre modèle en deux. Les diverses raisons seront présentées dans la section 4.3. Pour des raisons de complexité d'algorithme et de fréquence d'horloge, nous aurons deux cibles telles que nous avons vu plus haut, un FPGA et un DSP. La **Figure 12** montre l'interaction des logiciels utilisés à travers les étapes d'implémentations.

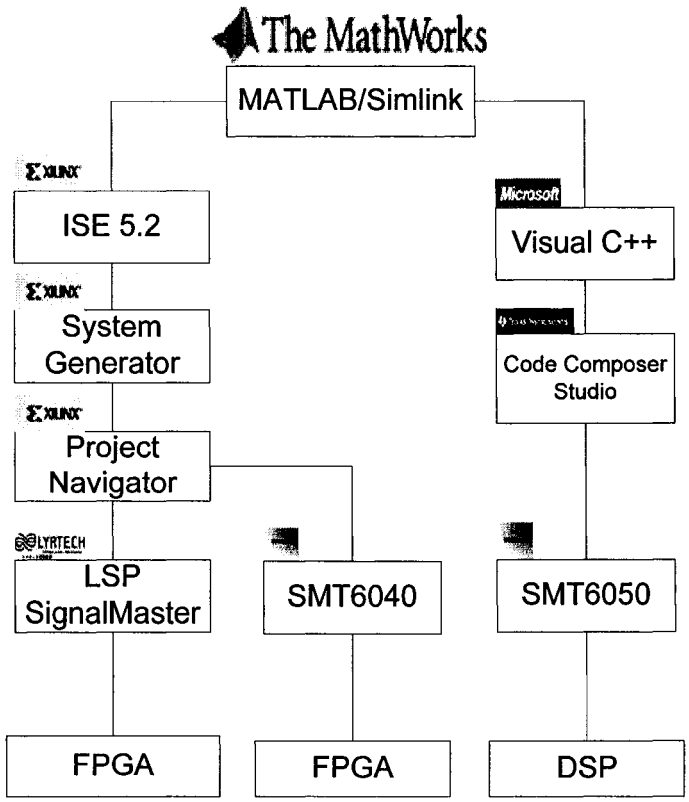



Figure 12 Interaction entre les différents logiciels

De toutes les compagnies qui font la production des FPGA, Xilinx Inc. a développé une série d'outils logiciels faciles à utiliser et adaptés au FPGA qu'ils produisent. La suite de Xilinx, appelé ISE, comporte plusieurs éléments aidant à la génération, à la validation et à la compréhension des codes binaires qui seront générés pour l'implémentation du FPGA.

Xilinx incopore à l'intérieur des librairies de Matlab/Simulink, une librairie de blocs adaptés et préparés à être transformés en code binaire. Cette librairie nous servira de base afin de préparer et de modifier le modèle Simulink de notre chaîne de

communication GPS. Les blocs inclus offrent les éléments de conceptions nécessaires afin d'implanter notre modèle.

La partie System Generator de la suite Xilinx génère les codes VHDL. À partir de ces codes, le logiciel Project Navigator, de la suite Xilinx ISE, permet de transformer les codes VHDL en code binaire. Avant de créer le code binaire, le logiciel nous permet aussi de visualiser les emplacements et les différents blocs utilisés directement sur une carte, représentant le FPGA. Ainsi, le concepteur peut facilement voir, non seulement l'espace requis par son système, mais aussi les différents branchements à travers le processeur et le programmeur peut ainsi optimiser, manuellement, l'emplacement du code à l'intérieur du matériel.

### 2.3.3 Concepts fondamentaux et raffinements

#### 2.3.3.1 Démarche d'implémentation pas-à-pas

Tel que vu précédemment, par la chaîne d'interaction logicielle, nous avons certaines étapes à suivre avant de réussir à implémenter le système dans un processeur. Tout d'abord, nous devons bien paramétrer notre modèle. Certains de ces paramètres auront des répercussions importantes, non seulement dans la performance du système au niveau de la simulation, mais aussi au niveau des performances du système en temps réel puisque ces derniers interviendront dans la génération des codes.

La **Figure 13** nous montre l'interaction, autant logicielle que matérielle, de l'implémentation de notre système.

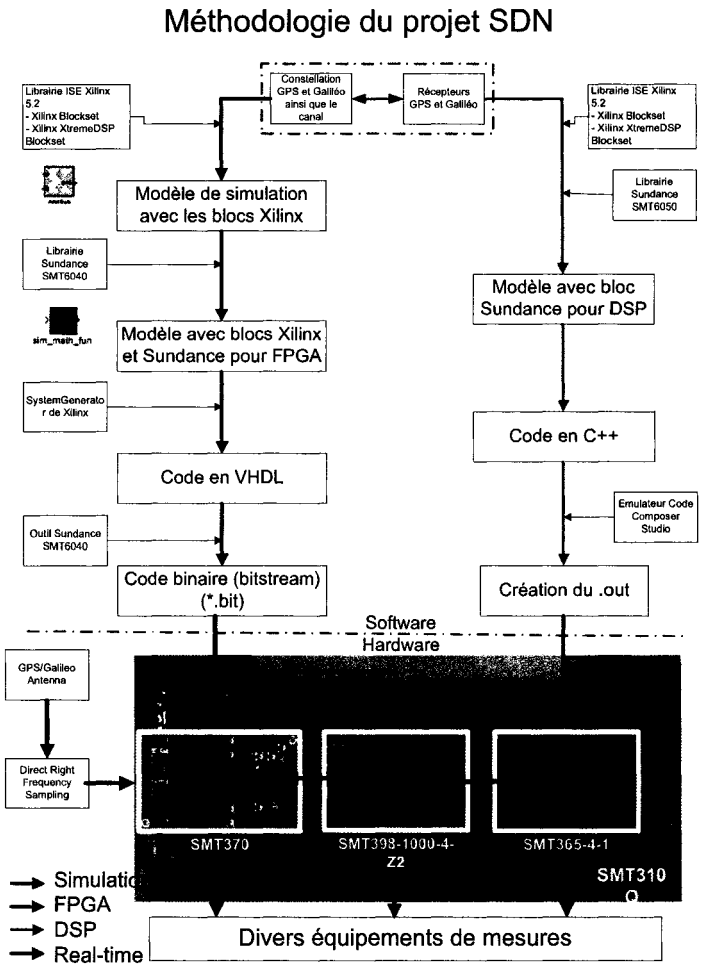



Figure 13 Méthodologie de travail pour l'implémentation d'un récepteur temps réel

La démarche exposée dans la **Figure 13** peut non seulement s'appliquer à notre projet, mais aussi à tout autre projet de réalisation de modèle en temps réel. Il est à noter que cette démarche est utilisée dans des entreprises oeuvrant dans le domaine. Ceci est une marque que notre méthodologie a fait ses preuves et qu'elle est adaptée à nos besoins.

La première partie, en vert sur la figure, représente la partie modélisation du système avec Matlab/Simulink. Elle est la base du développement, ce qui fait qu'on la retrouve la première sur le graphique. Une erreur à ce niveau, tel qu'on le verra plus tard, se

répercute dans tout le reste du processus. Cependant les simulations et l'analyse des performances du modèle en simulation, les erreurs peuvent être diagnostiquées à ce niveau et être corrigées avant l'implémentation. Ainsi, nous nous attarderons plus longuement sur les explications et les paramètres concernant cette partie. Le chapitre 3 explorera et expliquera le système du projet, c'est-à-dire la chaîne de communication GPS considérée. Pour cette section, seule la démarche sera expliquée. Les chapitres suivants montreront les différentes étapes une par une utilisées dans ce projet afin de compléter l'implémentation.

Une fois le système modélisé avec Matlab/Simulink et fonctionnant de manière réaliste, il est important de bien connaître les différentes librairies de Xilinx et de Sundance. On peut remarquer que la **Figure 12** et la **Figure 13** se doivent d'être superposées puisqu'elles sont complémentaires, la démarche utilisant les logiciels afin de réaliser les différentes étapes. Afin de compléter le passage en temps réel du système, certains blocs sont particuliers et importants. Plusieurs paramètres doivent être respectés sur chacun de ces blocs, que ce soit le nombre de bits après la virgule, le nombre de bits total ou la fréquence d'échantillonnage. Évidemment, le début de cette étape se situe au niveau de l'étude théorique du système à implémenter. On en extirpe les grandes caractéristiques et les limitations, ce qui aidera à modéliser le système. Dans le chapitre 3 du présent ouvrage, l'étude théorique d'une communication GPS est présentée, et jumelée avec les caractéristiques énumérées précédemment complète l'étude théorique.

Après la simulation validée avec des résultats réels et probants, pour suivre la méthodologie, la séparation de notre modèle est exigée afin de déterminer la marche à suivre. Évidemment, tel qu'expliqué plutôt, les processeurs ciblés étant différents, ils impliquent une marche à suivre non seulement qui diffère mais qui fait intervenir différents logiciels. Les caractéristiques de notre modèle nous imposent souvent le processeur ciblé et réduit les choix du programmeur. Tel qu'il sera expliqué plus loin, certains processus requièrent une attention particulière lors de cette étape. Ensuite, selon

la cible choisie, divers logiciels ou partie de logiciels interviennent. Ainsi, pour la génération d'un code binaire, nous devons faire appel à différentes librairies de Simulink que Xilinx a développé. Ces libraires possèdent des outils adaptés afin de faciliter le passage en temps réel de notre modèle. À l'instar de Xilinx, Texas Instruments, un des leaders mondiaux dans le domaine des processeurs numériques, utilise des logiciels spécifiques pour convertir notre modèle en un code C++. Cependant, du coté de Texas Instruments, aucun changement n'est requis afin de passer le modèle en C++. Par contre, Sundance a développé des outils pour faciliter la communication une fois que le modèle est en assembleur. Ces blocs doivent donc apparaître au niveau du modèle, ce qui nous force à modifier notre modèle exactement comme dans le cas où le processeur ciblé est un FPGA.

Une fois les changements faits, il nous est donc possible de déterminer sur quelle branche de la marche à suivre nous nous trouvons. Du coté DSP, par la génération automatique du code C++ relié à notre modèle, le logiciel Code Composer Studio prend ce code et le traduit en langage machine, soit l'assembleur, pour que le DSP puisse l'exécuter. De plus, ce logiciel prend vraiment le contrôle des communications avec le DSP car c'est à partir de son écran de contrôle que nous pouvons commander l'exécution de notre code. Ceci est une différence majeure entre le DSP et le FPGA, puisque pour ce dernier, une fois le code implémenté dans le processeur, il s'exécute automatiquement. Cette particularité fera de notre DSP le système de démarrage de notre modèle en temps réel.

Une des difficultés de cette démarche reste la communication entre la partie logicielle et la partie matérielle. Il faut s'assurer que les protocoles de communication soit parfaits puisqu'une erreur dans l'implémentation peut rendre notre modèle tout à fait inopérant Il faut donc respecter la composition des processeurs afin d'établir un protocole qui nous permettra de les configurer. Puisque nous avons deux différents processeurs (DSP et FPGA), les protocoles diffèrent pour chacun d'entre eux. Tel que mentionné plus haut,

Texas Instruments a inclus ce protocole avec le logiciel Code Composer Studio. Par contre, pour le FPGA, ce n'est pas le fabricant qui a conçu le système de communication mais bien Sundance, le développeur de la plate-forme. Ceci implique une interaction de plus entre notre modèle et le processeur ciblé, augmentant ainsi la complexité de la tâche. Il est important de noter que plus nous avons d'étapes, plus il est probable que nous introduisions des erreurs.

Une fois notre modèle dans les processeurs, nous pouvons donc tester le système en co-simulation. Cette étape nous permet de valider l'implémentation à l'intérieur des processeurs. La **Figure 14** illustre le principe de co-simulation et les relations qu'il y a entre le matériel intervenant.

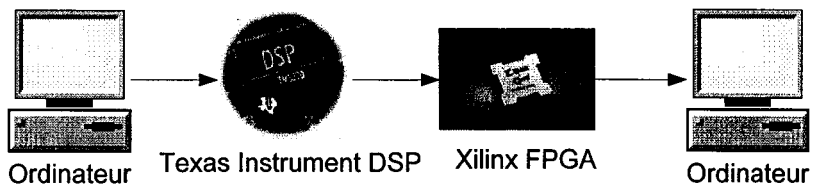



Figure 14 Principe de co-simulation

Par la co-simulation, les données sont générées par l'ordinateur, via un logiciel tel Matlab/Simulink, et le traitement est fait à l'intérieur des processeurs. Les données traitées sont ensuite redirigées vers le logiciel qui nous permet d'analyser les performances du système. Ainsi, l'analyse ainsi faite sert à dépister plus rapidement les erreurs pouvant avoir lieu, dans un environnement rapide et avec un minimum d'interaction. Il est plus facile de cette façon d'adapter le système et le modèle aux caractéristiques des processeurs.

Le système implémenté dans les deux processeurs est alors prêt à être testé en temps réel. Mais pour ce faire, il faut être capable de récupérer les signaux générés par les processeurs. Il nous faut donc des convertisseurs analogique-numérique rapides. La

troisième carte de notre plate-forme, nommée SMT370 sur la **Figure 13**, sert à acquérir ces signaux afin de pouvoir les analyser en temps réel. L'analyse du fonctionnement en temps réel, toujours à partir de signaux générés à partir du logiciel Matlab, permet d'établir les limites du système. Ainsi, lors de la prochaine étape, il sera plus facile de déterminer l'environnement de notre système.

La dernière étape de notre méthodologie permute les signaux générés avec les signaux réels de la constellation satellitaire GPS. L'antenne GPS sera branchée sur une tête RF, qui servira à moduler le signal dans la bande passante active des convertisseurs analogique-numériques. Le signal IF sera donc traité par nos processeurs et les données comparées aux éphémérides disponibles sur le web.

## 2.4 Conclusion

Le principe de radionavigation étant assez récent dans le monde moderne, son développement est encore à l'état embryonnaire. À l'image du développement radio (qui engendra l'éclosion des cellulaires), le principe de « Software Defined Navigator » est essentiel au développement des récepteurs de positionnement. Mais pour y arriver, une méthodologie doit être développée pour identifier les éléments nécessaires à la conception d'un récepteur de position purement logiciel. L'avenir de cette technologie est important si l'on se réfère à ce qui s'est produit au niveau des ondes radio. Au cours de ce mémoire, nous avons développé et testé cette méthodologie et ce principe. Les chapitres suivants font des conclusions et des projets futurs de ce projet du « Software Defined Navigator ».

# CHAPITRE 3

# ARCHITECTURE D'UNE CHAÎNE DE COMMUNICATION GPS

## 3.1 Études des perturbations agissant sur le signal GPS

Nous avons pu voir les signaux utilisés dans une chaîne de communication GPS au CHAPITRE 1. Ces signaux définissent les variables d'environnement de notre modèle. Dans ce chapitre, on présente le système dans son entièreté et sa fonctionnalité globale. L'interaction entre les blocs et les boucles est capitale et les réponses se devront être probantes et réalistes. L'approche ainsi privilégiée se divise en ces deux sections :

- Représentation pyramidale
- Utilisation de l'architecture planaire

La représentation pyramidale de notre système nous permet d'économiser de l'espace en mémoire quant à l'utilisation des paramètres. Pour ce faire, le modèle est décomposé en différents systèmes qui, à leur tour, sont décomposés en sous-système. Les paramètres de chaque sous-système sont particulier à lui, et ne sont pas transférer en tant que variables générales. Ce système nous permettra de sauver des ressources de point de vue temps et espace ressources pour simuler notre modèle que ce soit en simulation pure ou en co-simulation.

Tel qu'exprimé plus tôt, la modélisation d'une chaîne de communication GPS se divise en deux partie :

1. Source du signal GPS avec perturbations
2. Récepteur GPS avec récupération sur la phase et le code

La première partie génère le signal GPS en simulant le signal satellitaire, en y incorporant les perturbations subites par ce dernier lors de son déplacement sur sa trajectoire dans l'espace entre le satellite et le récepteur. Les signaux sont simulés tel que présentés dans la section 1.3. L'ajout de l'effet Doppler subit par cette onde peut être une tâche ardue et pour bien comprendre son implantation dans le modèle, une analyse théorique de l'effet Doppler est essentielle.

L'effet Doppler se produit lors du déplacement relatif entre une source et un récepteur, que l'on parle de son, qui est plus facile à sentir, ou tout autre forme d'onde. Le signal reçu par le récepteur n'a pas toujours la même fréquence que le signal émis par la source, tel que montré à la **Figure 15**.

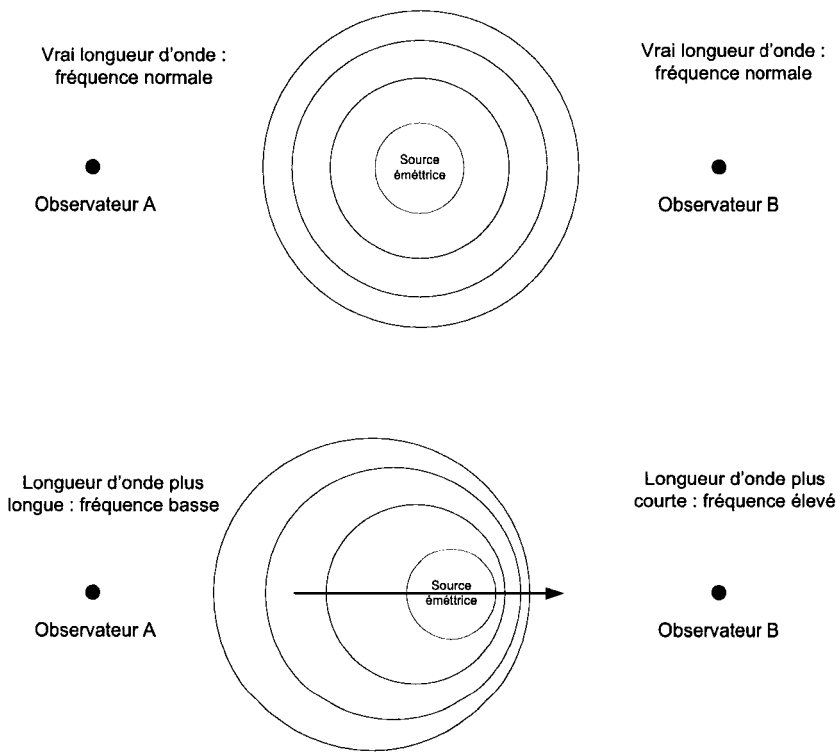



Figure 15 Principe de l'effet Doppler

La **Figure 15** nous montre l'effet Doppler vu par 2 observateurs ou récepteurs. Dans la première partie de la figure, les 3 composantes (observateurs A et B ainsi que la source) sont immobiles. Les longueurs d'ondes observées sont les même pour les 2 récepteurs. Par contre, dans la partie inférieure du schéma, la source se déplace vers la droite. Les fréquences vont se modifier selon les relations suivantes :

$$f_A = f_s - f_d = f_s - \frac{v_s f_s}{c} \tag{3.1}$$

$$f_B = f_s + f_d = f_s + \frac{v_s f_s}{c} \tag{3.2}$$

où $f_A$, $f_B$ et $f_s$ sont les fréquences perçues respectivement par A, B et la source, $c$, la vitesse de la lumière et $v_s$ est la vitesse de la source dans l'axe directionnel qui relie la source au récepteur.

Nous pouvons donc constater que la fréquence augmente si la source se rapproche de l'observateur tandis qu'elle diminue si la source s'en éloigne. Le changement en fréquence perçu pour l'observateur a des conséquences, non seulement sur la récupération de la porteuse mais aussi sur le code C/A. La période du code changeant, à cause de l'effet Doppler, la position des raies spectrales de ce signal est donc modifiée. Cet aspect change donc la distribution de la puissance du signal, élément important pour la récupération des données car l'autocorrélation est la base des calculs pour récupérer le code dans la boucle DLL (cet aspect sera démontré dans la section 3.4. Pour aider nos boucles PLL et DLL à générer les bonnes répliques, il peut être utile de connaître les limites de l'effet Doppler sur le signal GPS. L'effet Doppler sur les satellites est dû à 2 mouvements : celui du satellite autour de la Terre et celui du récepteur dû à la rotation de la Terre et à sa propre dynamique dans le repère terrestre.

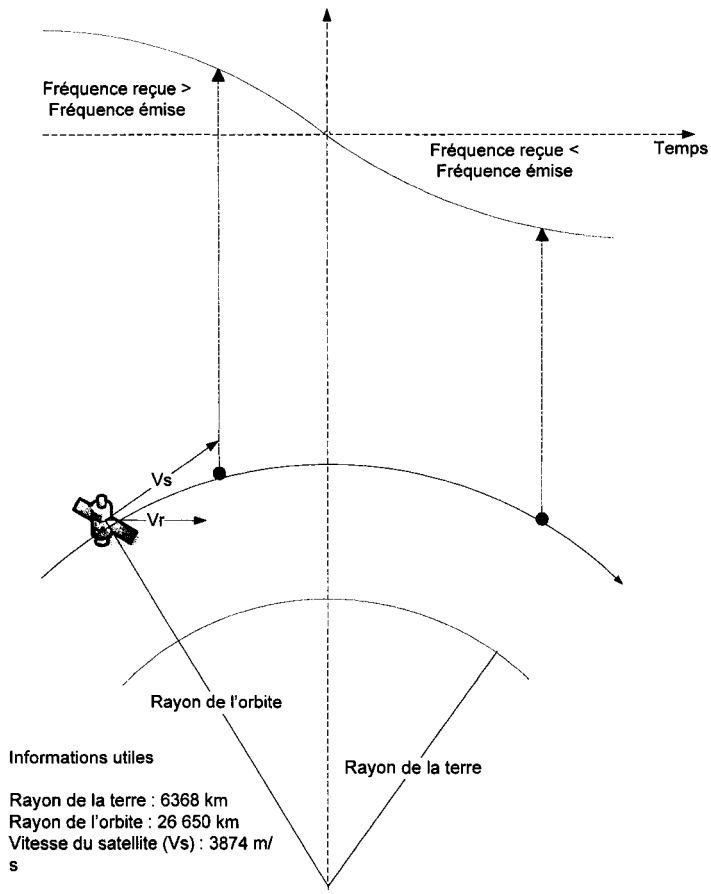



Figure 16 Effet Doppler sur le satellite

La **Figure 16** nous montre l'impact d'un mouvement satellitaire sur la fréquence émise. Tel que montré dans les équations, la vitesse du mobile est un élément essentiel de l'effet Doppler et, pour la constellation GPS, les vitesses ($V_s$) des satellites sont considérées comme constantes et facilement calculables. Cependant, il en est tout autrement pour la vitesse radiale $V_r$. Cette dernière peut s'exprimer de la façon suivante :

$$V_r = \frac{V_S R_T \cos\theta}{\sqrt{R_T^2 + R_S^2 - 2R_T R_S \sin\theta}} \qquad \textbf{(3.3)}$$

où θ est l'angle d'élévation du satellite par rapport au centre de la Terre. Cette vitesse n'est pas constante et varie justement en fonction de l'angle d'élévation. Il nous est donc possible de déterminer l'accélération radiale du satellite avec l'expression suivante :

$$a_R = \frac{\partial V_R}{\partial t} = \frac{V_S R_T \left[ V_S R_T \sin^2\theta - (R_T^2 + R_S^2)\sin\theta + V_S R_T \right]}{\sqrt{(R_T^2 + R_S^2 - 2R_T R_S \sin\theta)^3}} \cdot \frac{\partial \theta}{\partial t} \qquad (3.4)$$

Les limites de l'effet Doppler dépendent des limites des vitesses et accélérations radiales du satellite. Par le théorème des valeurs maximales et minimales, nous pouvons calculer ces bornes en égalant la dérivée des équations 3.3 et 3.4 égale à 0.

$$\frac{\partial V_R}{\partial \theta} = \frac{V_S R_T \left[ V_S R_T \sin^2\theta - (R_T^2 + R_S^2)\sin\theta + V_S R_T \right]}{\sqrt{(R_T^2 + R_S^2 - 2R_T R_S \sin\theta)^3}} = 0 \qquad (3.5)$$

Avec cette équation, la valeur maximale de la vitesse radiale trouvée est de 929 m/s. La valeur de l'angle d'élévation correspondant à cette vitesse est de 13,9 degrés, toujours par rapport au centre de la Terre. Tel qu'illustré à la **Figure 16**, cette vitesse est atteinte lorsque le satellite est 13,9 degrés ou 166,1 degrés par rapport au récepteur, donc à l'horizon. Et l'effet Doppler sera donc maximum lorsque le satellite sera à ces positions. En se référant à la **Figure 16**, nous pouvons déjà déduire que l'effet Doppler augmentera la fréquence quand le satellite se rapprochera du récepteur et diminuera la fréquence lorsqu'il s'éloignera. Ainsi, le calcul de l'effet Doppler maximal se traduit par l'expression suivante :

$$f_{Doppler\,\max} = \frac{f_s V_{R\max}}{c} = \frac{1.57542\times10^9 \times 929}{3\times10^8} = 4.8786\times10^3 \ \text{Hz} \qquad (3.6)$$

Cet effet Doppler se calcule sur le signal GPS dans son ensemble. Mais il faut considérer que cet effet se répercutera sur les données encodées à l'intérieur du signal. L'effet Doppler maximal sur le code et sur les données est de respectivement 3 Hz et 0,15 mHz, valeur négligeable par rapport au Doppler subi par la porteuse.

Pour générer l'effet Doppler dans notre modèle, il est important de rappeler que la fréquence de notre porteuse ne sera pas la même que le signal GPS réel. Les logiciels que nous utilisons (Matlab et Simulink) ne peuvent échantillonner à fréquence élevée sans utiliser une partie trop importante de la mémoire, ralentissant énormément le temps de simulation. Dans le cas du GPS, l'échantillonnage minimum nécessaire serait de 3,15084 GHz. Ainsi, nous devons simuler à une fréquence porteuse plus faible que la fréquence du signal GPS. Cependant, pour choisir une fréquence que nous appellerons intermédiaire, nous devons respecter les normes de base en génération du signal GPS. La synchronisation des signaux à l'intérieur est une importante caractéristique pour la génération du signal. Or, un changement dans l'une de ces fréquences diminuerait le réalisme de notre modèle. Par contre, si l'on respecte dans un certain rapport les fréquences des signaux, il nous sera possible de garder le réalisme de notre modèle.

$$k = \frac{f_p}{f_c} = \frac{f_p + f_{doppler\_porteuse}}{f_c + f_{doppler\_code}} = \frac{1.57542\ GHz}{1.023\ MHz} = 1540 \qquad (3.7)$$

où $f_c$ est la fréquence du code et $f_p$ est la fréquence de la porteuse.

La relation 3.7 nous indique que le rapport entre la porteuse GPS et le code C/A est de 1540. En utilisant une fréquence intermédiaire, nous devons respecter le rapport que l'effet Doppler aura sur le code. Ainsi, si on choisit une fréquence porteuse autre que 1,57542 GHz, l'effet Doppler sur le code se devra de respecter ce rapport. Cette façon de calculer l'effet Doppler est plus réaliste mais, pour la simulation, peu convaincante sur les performances de nos boucles. Pour respecter ce rapport, l'effet Doppler sur le code

ne devrait pas dépasser 3 Hz, tel que nous avions calculés plus tôt. Mais un changement de 3 Hz sur une fréquence de 1,023 MHz ne représente que 0.00029 % de variation, ce qui est négligeable par rapport à la valeur absolue de l'effet Doppler sur la porteuse. Donc, pour tester nos boucles et leurs performances nous allons séparer ces perturbations en les ajoutant sur chaque générateur de fréquence individuellement. Même si cela diminue le réalisme de notre modèle, il nous sera possible d'augmenter le stress dynamique appliqué sur nos boucles et d'étudier leur comportement.

La génération de l'effet Doppler sera appliquée de façon individuelle dans notre modèle, et la méthode vous sera expliquée dans le détail plus loin dans le texte. Les autres perturbations, outre l'effet Doppler, considérées dans notre modèle sont :

- Le saut de phase
- Le saut en fréquence

Le saut de phase correspond à une erreur due au retard du signal par rapport à son émission. Par exemple, la réflexion du signal GPS sur un édifice dans un milieu urbain donne lieu à des multi-trajets. Cependant, nous nous devons de traiter ce signal puisqu'il s'agit peut-être du seul signal dont le récepteur à accès. Cette perturbation sera générée directement à l'aide d'un bloc incorporant un délai sur le signal.

La forme générale d'une onde sinusoïdale s'exprime de la forme suivante :

$$y(t) = A * \sin(2\pi(F_c + F_d) * t + \theta_i) \qquad \textbf{(3.8)}$$

où A est l'amplitude de l'onde, $F_c$ est la fréquence centrale, $F_d$ est l'effet Doppler, et $\theta_i$, la phase initiale de l'onde.

Tel qu'expliqué plus haut, le saut de phase s'obtient en changeant tout simplement $\theta_i$, ou en retardant le signal GPS. Par contre, pour le saut de phase, nous devons incorporer un autre paramètre qui influence la fréquence centrale de notre système, soit $F_c$. Selon l'expression 3.10, on peut le faire via la composante $F_d$ de l'équation. Il faut se rappeler que, l'effet Doppler, n'est qu'une particularité du saut de fréquence. L'effet Doppler est, dans le fond, un saut de fréquence qui dépend de la vitesse relative entre la source et le récepteur. La vitesse du mobile influence la fréquence que l'on reçoit, et la vitesse est la dérivée du déplacement par rapport au temps. Nous pouvons donc affirmer que l'effet Doppler dépend du temps, et peux s'exprimer ainsi :

$$F_d = \frac{F_c}{c} * (\frac{dy}{dt}) \tag{3.9}$$

Ainsi, si l'on veut avoir un saut de fréquence constant, il suffit de poser que la dérivé du déplacement en t soit égale à 0.

$$y(t) = A * \sin(2\pi(F_c + F_d) * t + F_b + \theta_i) \tag{3.10}$$

où $F_b$ est la composante à dérivé nulle de l'effet Doppler. Donc, pour simuler un saut de phase, nous incorporerons un paramètre dans notre définition de l'argument de la sinusoïde.

## 3.2 Architecture de la source des signaux GPS

Pour tout signal de communication, la première étape est la génération du signal de base ou d'une horloge. Dans le cas du GPS, la génération du signal complet se fait en 2 étapes. Tout d'abord, on effectue un étalement spectral qui module une porteuse à une fréquence, puisque l'on fonctionne en bande de base, de 2,046 MHz. On utilise une

modulation BPSK qui fait un changement de phase de 180° selon si la donnée change ou conserve la même valeur. La **Figure 17** détaille la source du signal GPS.

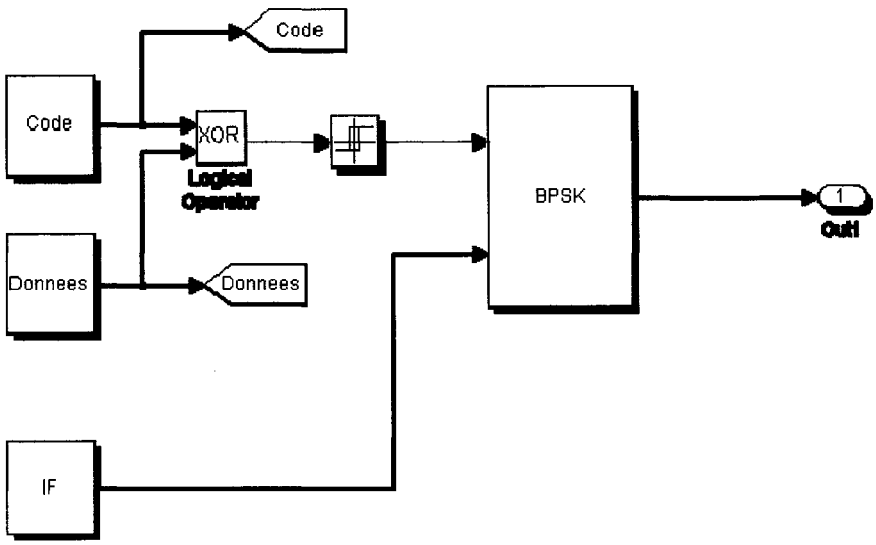



Figure 17 Schéma de la source du signal GPS

### 3.2.1 Le bloc Étage IF

L'intérêt de l'étage IF *(Intermediate Frequency)* réside principalement dans le fait qu'il génère la plupart des perturbations. En effet, dans ce bloc, les paramètres caractéristiques des perturbations sur notre porteuse, prendront place dans la génération de l'étage IF. Ainsi, on remarquera que pour les sauts de fréquence ou de phase, on change directement la donnée dans l'oscillateur local tandis que dans le cas d'une rampe de fréquence, on impose une rampe à l'entrée en ajustant la sensibilité de l'oscillateur à 1 Hz/V. La **Figure 18** nous montre ces caractéristiques.

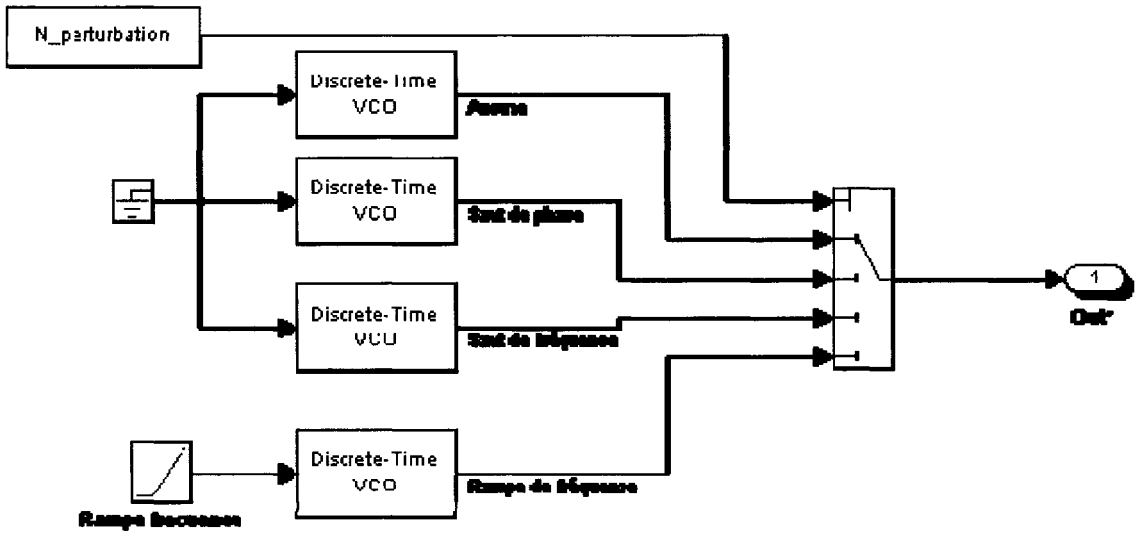



Figure 18 Schéma de la génération de la porteuse et des perturbations

Les VCOs (*Voltage Controlled Oscillator)* utilisés sont commandés par l'expression suivante :

$$y(t) = A_c * \cos(2\pi F_C * t + k * \theta + P_h) \qquad \textbf{(3.11)}$$

où $A_c$ est l'amplitude, $F_c$, la fréquence centrale (2,046 MHz dans notre cas), $P_h$ la phase initiale, $k$ la sensibilité du VCO (en Hz/V) et $\theta$ l'erreur de phase introduite comme entrée. Les variables $k, P_h$ *et* $F_c$ sont paramétrables par l'usager. Il est donc possible de générer tous les types de perturbations possibles à partir de ces trois paramètres.

Tableau II

Variables paramétrables de perturbation

| Perturbation | Paramètre |
|---|---|
| Saut de phase | $P_h$ |
| Saut de fréquence | $F_c$ |
| Effet Doppler | k |

Le **Tableau II** nous montre les variables paramétrables pour le fonctionnement du modèle et la génération des perturbations. En fait, dans le cas du saut de fréquence, on ajoute une valeur à la fréquence centrale afin de simuler cette perturbation. L'expression 3.11 devient donc :

$$y(t) = A_c * \cos(2\pi(F_C + F_p) * t + k * \theta + P_h) \qquad \textbf{(3.12)}$$

où $F_p$ est la valeur du saut de fréquence désiré.

Dans le cas de l'effet Doppler, le paramètre *k* (en Hz/V) sera l'amplitude de l'effet Doppler. Bien évidemment, l'effet Doppler n'a pas d'amplitude en tant que tel, il est donc normal que nous allons le laisser à la valeur unitaire. Cependant, ce qui est critique pour l'effet Doppler, c'est d'avoir une erreur de phase qui n'est pas constante sur 6 heures, temps maximal pendant lequel un récepteur peut voir le même satellite. Comme nous l'avons vu plutôt, l'effet Doppler varie avec la vitesse radiale du satellite, qui dépend, à son tour, de l'angle d'élévation du satellite. L'angle calculé pour un effet Doppler maximal, autour de 13 degrés, correspond à l'horizon. Lors de sa conception, la constellation satellitaire GPS a été calculé pour que chaque satellite fasse une rotation de

la terre en près de 12 heures. Donc, d'un horizon à l'autre, pour un récepteur immobile, un temps de 6 heures doit être pris en considération pour passer de +4kHz à -4kHz d'effet Doppler sur la porteuse. Un signal qui simule cette trajectoire doit être conçu pour commander le VCO. Même si, dans la réalité, le changement de fréquence n'est pas linéaire, lors de la simulation, nous allons le considérer comme linéaire pour notre modèle. Ce signal aura donc les caractéristiques suivantes :

- Période : 12h
- Amplitude max : 4000
- Amplitude min : -4000

Avec les définitions des systèmes d'ordre 1 et de la pente, nous arrivons à l'équation qui pilotera la rampe de fréquence, c'est-à-dire :

$$y = mx + b = \frac{(-4000 - 4000)}{12 * 60 * 60} x + b = -0.1852x \qquad (3.13)$$

La pente, 0.1852, devient donc la sensitivité $k$ de notre VCO. La valeur de $b$ est nulle car, tel que montré à la **Figure 16**, au zénith entre le satellite et le récepteur, la valeur de l'effet Doppler est 0.

### 3.2.2 Le bloc Code

Le générateur pseudo-aléatoire du code de Gold permet de moduler les données avec un code spécifique et unique pour chacun des satellites. Le code de Gold est un code pseudo aléatoire spécifique à chaque satellite, appelé PRN (*Pseudo Random Noise*), dont la fréquence est de 1,023 MHz. La particularité de ce code est qu'il permet une autocorrélation maximum et un minimum d'intercorrélation, ce qui diminue le risque d'erreur dans la réplique au récepteur. La **Figure 19** nous montre la génération du code

de Gold C/A. Basé sur deux registres, G1 et G2, le code de Gold est un code bloc cyclique à 2 registres de 10 cellules dont les registres représentent les équations de codage. Les équations des deux registres sont semblables pour tous les satellites, cependant, au niveau du registre G2, un sélecteur de phase à configuration unique pour chaque satellite confère la particularité d'autocorrélation. La réponse de ce sélecteur est ajoutée avec une porte logique OU EXCLUSIF avec la réponse du registre G1 afin de produire le code de Gold.

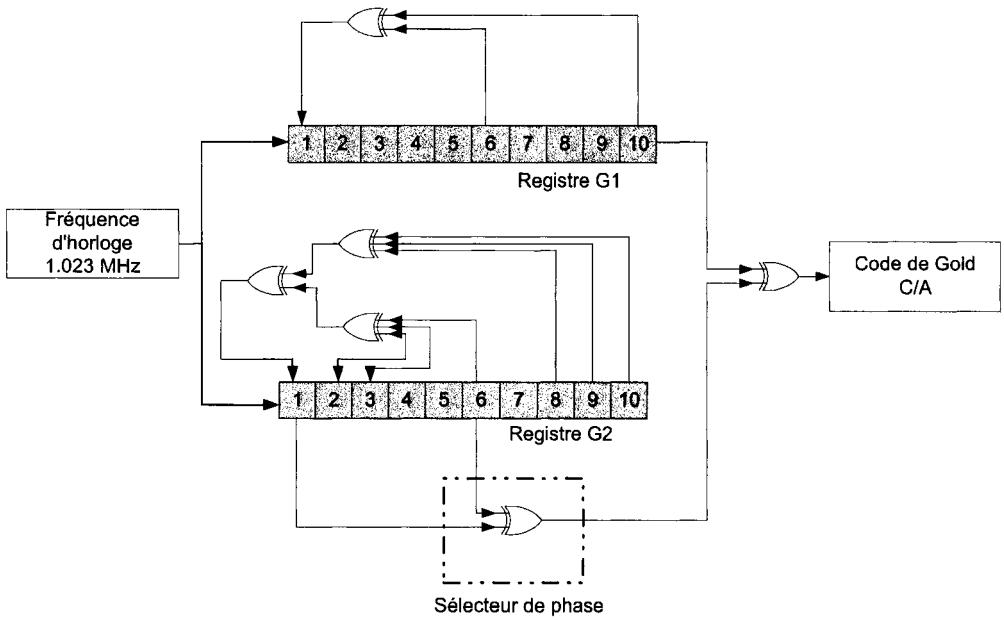



Figure 19 Principe de génération du code C/A

Dans le contexte de ce projet, nous n'utilisons qu'un seul code comme modèle de base. Une horloge donne la fréquence à laquelle le code est généré. L'ajout d'un paramètre pour choisir le PRN peut être envisagé mais n'est pas nécessaire pour l'analyse des performances du modèle. Cependant, dans une phase ultérieure du projet, pour l'analyse de quatre canaux, le PRN devra être différent pour chacune des sources. L'ajout de cette caractéristique sera abordé dans la section 5.3.

La génération des perturbations sur le code se fait de la même façon que sur la porteuse. Ayant un VCO comme générateur, nous utilisons exactement les mêmes paramètres, avec des valeurs différentes évidemment, que dans le cas de l'étage IF. Au niveau du code, cependant, on ne peut introduire une erreur de phase de plus de $2\pi$, puisque, comme nous le verrons plus tard, les discriminateurs n'en tiennent pas compte. Cette erreur revient à une erreur initiale de plus d'une bribe et les discriminateurs utilisés ne fonctionnent qu'à l'intérieur d'une seule bribe.

Cependant, le signal sinusoïdal de commande doit être modifié pour générer le code C/A. En effet, si l'on se réfère à la **Figure 19**, les registres G1 et G2 sont composés eux aussi de 10 cellules chacun. Or, pour simuler ces cellules, nous utilisons des registres réagissant à front montant. Notre signal de commande, alors sinusoïde, doit donc être modifié en signal carré ayant la même fréquence, tel que montré à la **Figure 20**. Pour ce faire, nous utilisons tout simplement un bloc relais (avec comme paramètres 1 et -1). Les registres seront actifs seulement sur un front montant.

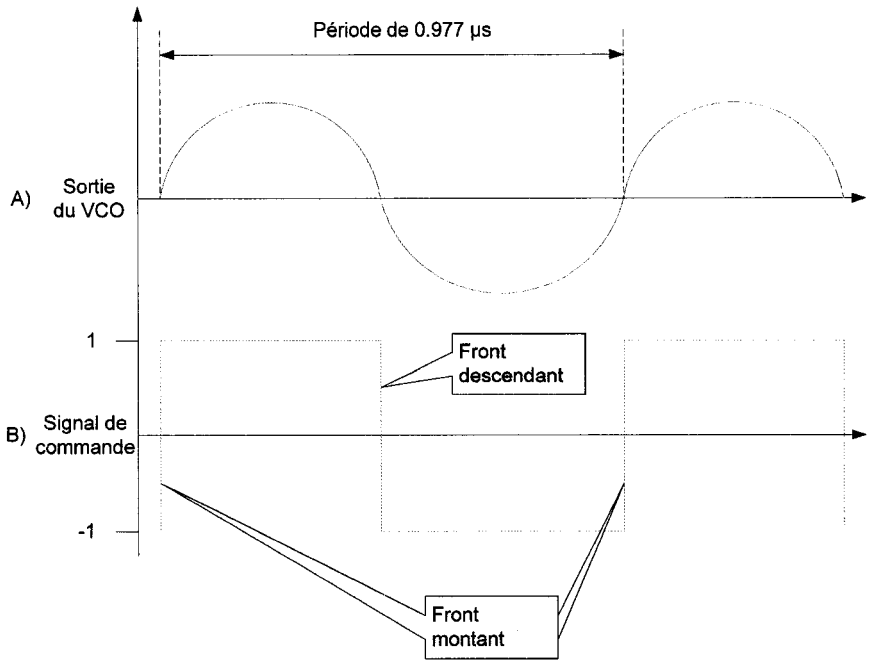



Figure 20 Génération du signal de commande des registres

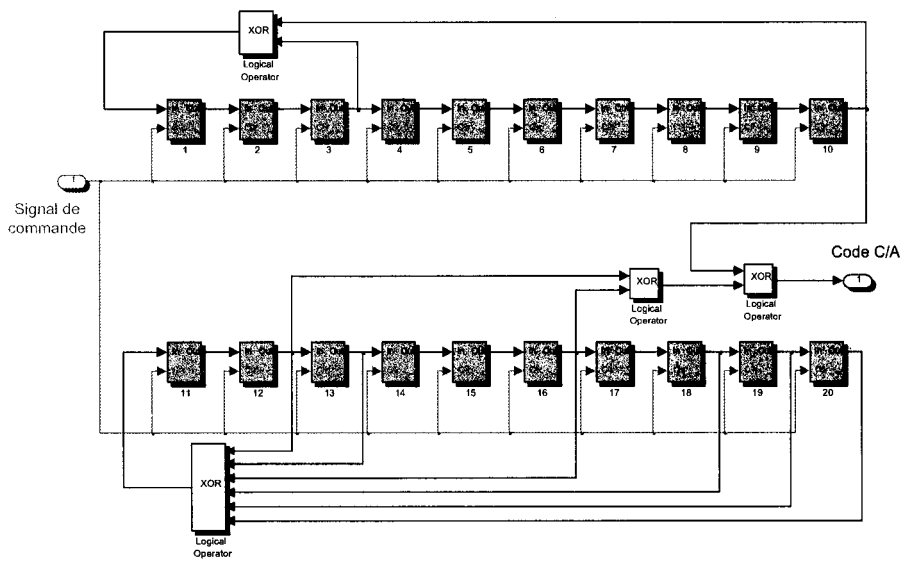



Figure 21 Génération du code C/A en Simulink

La **Figure 21** montre la méthode utilisée pour générer le code C/A en Simulink. Cette figure montre l'exemple pour un satellite. Le signal de commande doit comporter toutes les perturbations que nous voulons appliquer sur le code (i.e l'effet Doppler). La composition d'une cellule à l'intérieur d'un registre est montrée à la **Figure 22**

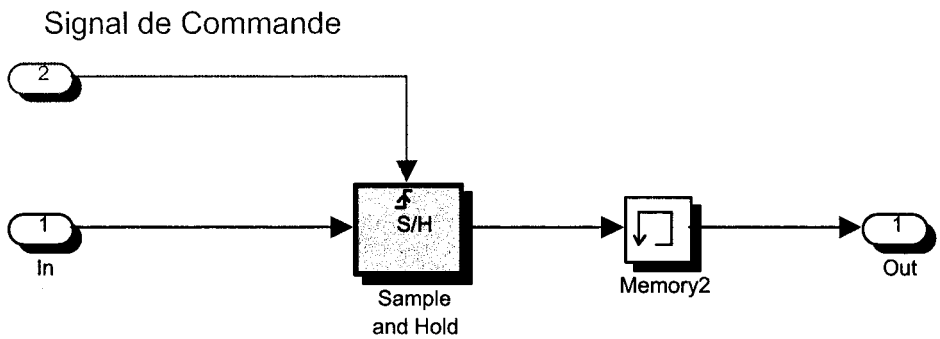



Figure 22 Cellule d'un registre du générateur de Code

Le bloc *Sample and Hold* peut, au choix du concepteur réagir sur un front montant, un front descendant ou les deux. Dans notre cas, que ce soit sur le front montant ou descendant, le résultat est le même. La caractéristique importante est que l'écart entre 2 fronts soit exactement de 0.977 μs, ce qui donne un signal de 1.023 MHz.

### 3.2.3 Signal en sortie de la source

Le système GPS utilise une modulation sur la porteuse de type BPSK (*Binary Phase Shift Keying*). L'étalement spectral est obtenu en multipliant (opération logique XOR) les données à envoyer au code pseudo aléatoire. Le gain de codage est défini par le rapport entre la fréquence du code pseudo aléatoire et la fréquence des données. On obtient donc un gain de codage de $\frac{1.023MHz}{50Hz} = 20460$. Plus le gain de codage est élevé, meilleure est la résistance du signal envoyé au brouillage.

A l'intérieur du cadre du projet, nous utilisons, à la base, seulement le code C/A. La séquence pseudo-aléatoire est générée de façon à maximiser l'auto-corrélation d'un canal et minimiser l'intercorrélation avec les autres canaux. Le code se répète à chaque 1023 bits pour une période totale de 1 ms.

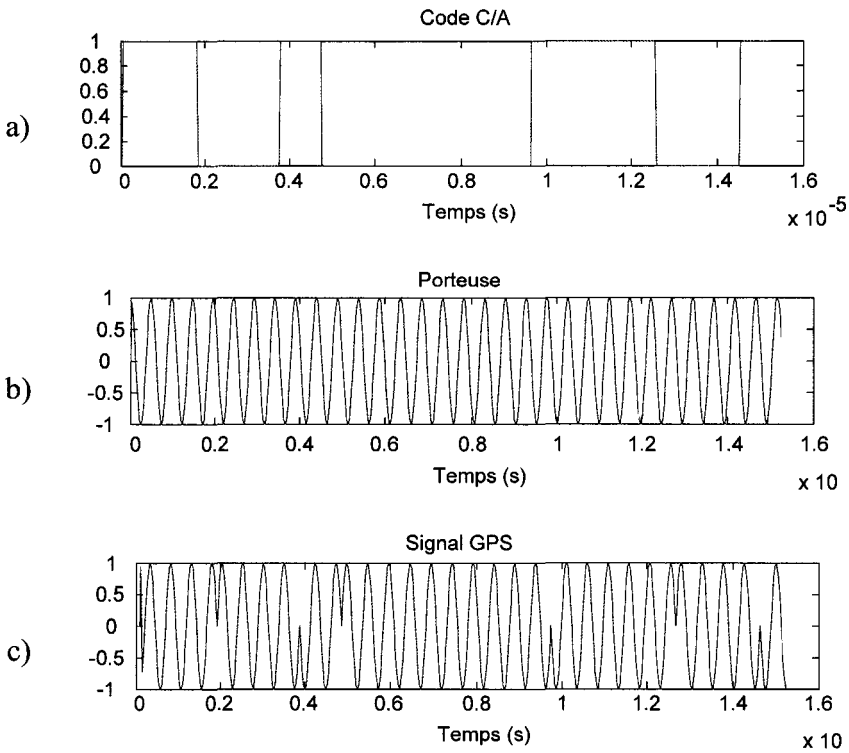



Figure 23 Génération du signal GPS

On remarque, sur la **Figure 23** qu'à chaque transition du code pseudo-aléatoire, la sortie (le signal GPS) est déphasée de 180°. La durée de chaque bit du code est de 1 usec. La fréquence de la porteuse, la forme d'onde centrale sur la figure précédente, est de 2.046 MHz. Cette fréquence intermédiaire se base sur la fréquence d'échantillonnage de notre modèle. Les limites de mémoire et de simulation du logiciel Matlab nous obligent à limiter cette fréquence d'échantillonnage pour avoir des résultats probants. Le choix de cette fréquence doit tenir compte aussi d'un nombre minimum permettant une reproduction réelle des signaux. Comme la fréquence du code est de 1,023 MHz, notre fréquence porteuse doit être supérieure à cette valeur, tout en respectant aussi la synchronisation des signaux. Une valeur élevée de notre fréquence d'échantillonnage,

quoique intéressante, exige un espace mémoire trop grand et le logiciel ne peut fonctionner correctement. La fréquence d'échantillonnage a donc comme limite inférieure le respect du théorème de Nyquist et comme limite supérieure, le respect de l'espace mémoire disponible.

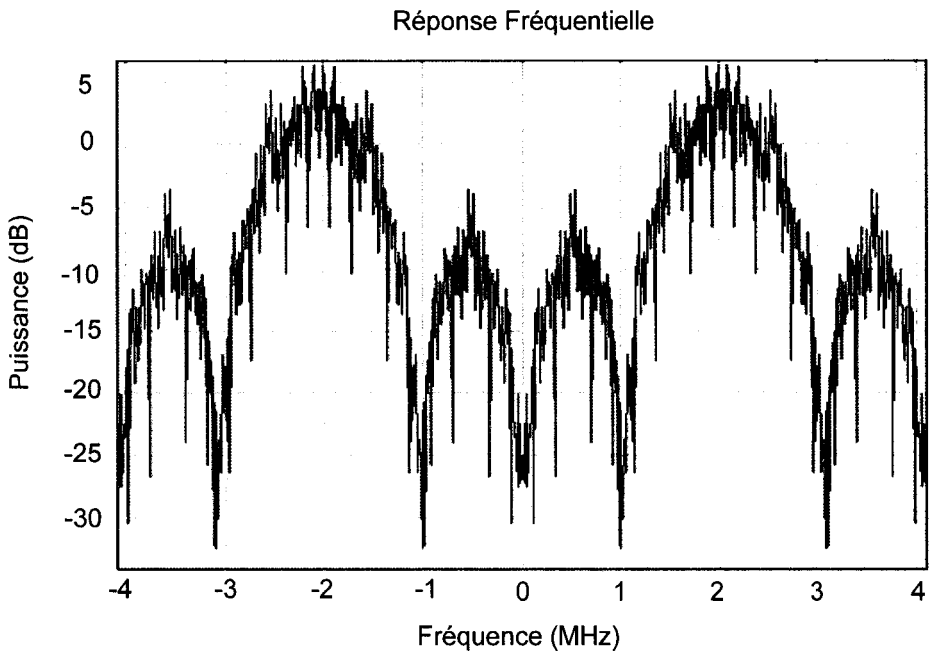



Figure 24 Réponse fréquentielle de la sortie

On remarque sur la **Figure 24** que la porteuse est bien à 2.046 MHz. Le lobe principal a un largeur de bande de 2,046 MHz et les lobes secondaires ont une largeur de bande de 1.023 MHz, ce qui correspond à la fréquence du signal modulant. La fréquence d'échantillonnage utilisée pour générer la **Figure 24** est de 8*1.023MHz. Plus la fréquence d'échantillonnage est élevée, plus on peut observer de lobes secondaires. La fréquence de la porteuse est fixée à $\frac{Fs}{4}$ pour maximiser l'utilisation de la bande de fréquence et éviter qu'il y ait recouvrement (aliasing) du spectre. Cet aspect sera analysé dans la section 3.5.

## 3.3 Principe de fonctionnement d'une boucle de Costas

Les boucles de recouvrement, autant de code que de la porteuse, d'un récepteur GPS sont basées sur le principe des boucles de Costas. Dans cette section, nous analyserons seulement les composantes utiles à nos boucles, c'est-à-dire les boucles d'ordre 3 appliquées à la PLL. Cette boucle a un filtre d'ordre 2 et, en théorie, devrait être la plus robuste. Cependant, en ANNEXE 1, l'analyse complète de la boucle de Costas est effectuée.

Le principe d'une boucle PLL est simple : on multiplie le signal reçu par une référence variable générée à l'interne et on filtre le produit. On utilise le résultat (qui correspond à l'erreur de phase entre le signal d'entrée et la référence) pour ajuster la fréquence de la référence variable (le VCO).

La configuration de la boucle de Costas est illustrée à la **Figure 25**:

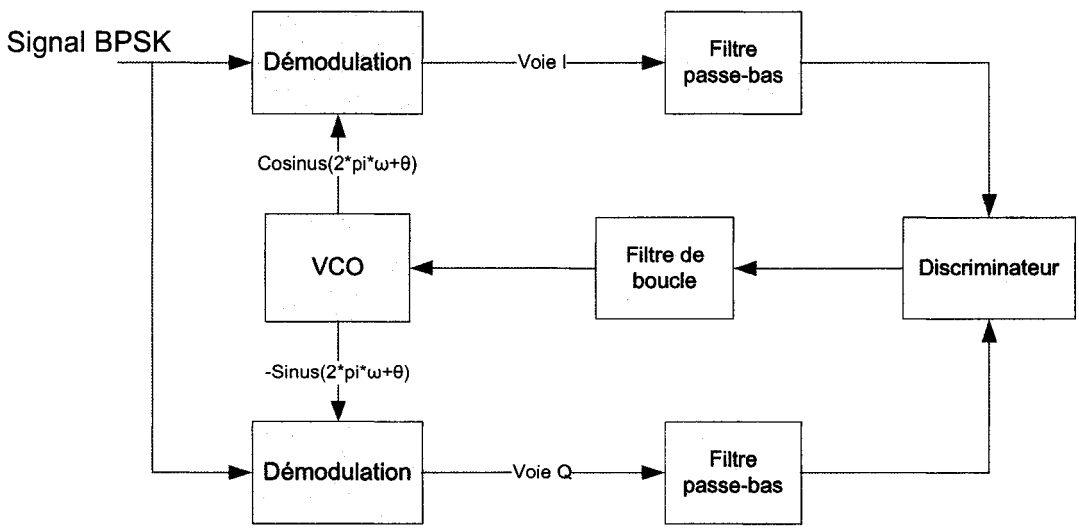



Figure 25 Exemple de boucle de Costas

La boucle de Costas génère un signal en phase (I) et en quadrature (Q) pour démoduler le signal BPSK. Le signal en quadrature doit être en avance de 90° sur le signal en phase. Dans le cas de la modulation BPSK, le signal en phase est un cosinus. Le signal en quadrature correspond donc à –sinus.

Le signal BPSK est défini par : $BPSK=\cos\left(\omega_c t+DATA(t)\frac{\pi}{2}\right)$ où *DATA* représente le message binaire (code et données) et correspond à ±1.

Un changement de phase de 180° équivaut à multiplier la porteuse par « -1 ». L'expression du signal BPSK peut donc être simplifiée :

$BPSK=DATA(t)\cdot\cos(\omega_c t)$

Pour analyser le comportement de la boucle, deux propriétés trigonométriques sont utilisées :

1. $\cos(a)\cos(b)=\frac{1}{2}[\cos(a+b)+\cos(a-b)]$
2. $\sin(a)\cos(b)=\frac{1}{2}[\sin(a-b)+\sin(a+b)]$

On analyse d'abord le produit du signal BPSK par la composante en phase:

$$I = BPSK * \cos(\omega_c t) = \left[ DATA(t) \cdot \cos\left(\omega_c t\right)\cos\left(\omega_c t\right)\right] \qquad \textbf{(3.14)}$$

Selon la propriété trigonométrique, on obtient :

$$I = \frac{1}{2}\left[ DATA(t) \cdot \cos(0) + DATA(t) \cdot \cos\left(2\omega_c t\right)\right] \qquad \textbf{(3.15)}$$

Une fois filtré par le filtre passe-bas, on obtient la composante continue:

$$I\_filtré = \frac{1}{2} \cdot DATA(t) \tag{3.16}$$

On analyse ensuite le produit du signal BPSK par la composante en quadrature :

$$Q = BPSK \cdot \sin(\omega_c t) = DATA(t) \cdot \cos(\omega_c t) \cdot \sin(\omega_c t) \tag{3.17}$$

Selon la propriété trigonométrique 2, on obtient :

$$Q = \frac{1}{2}\left[DATA(t) \cdot \sin(0) + DATA(t) \cdot \sin(2\omega_c t)\right] \tag{3.18}$$

Une fois filtré par le filtre passe-bas, on obtient :

$$Q\_filtré = 0 \tag{3.19}$$

Cette analyse montre que pour une porteuse en phase, l'extraction des données se fait après le filtre passe-bas du signal I. De plus, l'analyse montre que lorsque la boucle PLL est accrochée, le signal *Q_filtré* est nul.

Le signal BPSK est alors définit par : $BPSK=DATA(t)\cos(\omega_c t+\theta)$ où θ est l'erreur de phase.

Le produit du signal BPSK avec erreur de phase par la composante en phase correspond à :

$$I = \pm\frac{1}{2}\left[\cdot\cos\left(\left(\omega_{VCO} - \omega_{BPSK}\right)t + \theta\right) + \cos\left(\left(\omega_{VCO} + \omega_{BPSK}\right)t + \vartheta\right)\right] \tag{3.20}$$

où *DATA(t)* est remplacé par ±1.
Une fois filtré, on obtient :

$$I\_filtré = \pm\frac{1}{2}\cdot\cos\left(\left(\omega_{VCO}-\omega_{BPSK}\right)t+\theta\right) \tag{3.21}$$

La composante en quadrature correspond à :

$$Q = \pm\frac{1}{2}\left[\sin\left(\left(\omega_{VCO}-\omega_{BPSK}\right)t+\theta\right)+\sin\left(\left(\omega_{VCO}+\omega_{BPSK}\right)t+\vartheta\right)\right] \tag{3.22}$$

Une fois filtré, on obtient :

$$Q\_filtré = \pm\frac{1}{2}\cdot\sin\left(\left(\omega_{VCO}-\omega_{BPSK}\right)t+\theta\right) \tag{3.23}$$

En utilisant un discriminateur qui fait la multiplication des signaux *I_filtré* et *Q_filtré* on obtient en sortie du discriminateur :

$$sortie\_discr = -\frac{1}{8}\sin\left(2\left(\left(\omega_{VCO}-\omega_{BPSK}\right)t+\theta\right)\right) \tag{3.24}$$

On constate donc que la boucle de Costas nous permet d'ajuster simultanément l'erreur de fréquence $\omega_{VCO}$ - $\omega_{BPSK}$ et l'erreur de phase, θ. Le schéma équivalent de la boucle PLL linéarisée est présenté à la **Figure 26**:

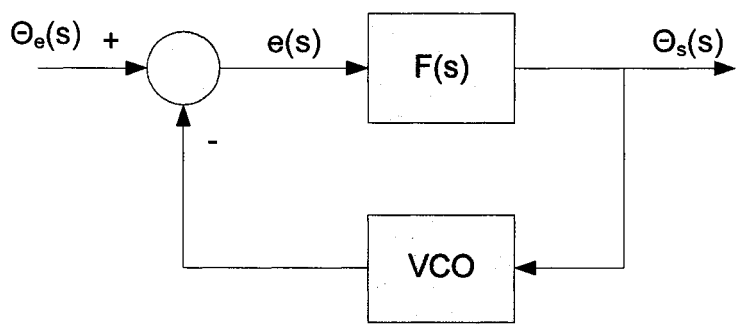



Figure 26 Boucle PLL dans le domaine fréquentiel

La fonction de transfert du filtre de boucle est F(s). Le VCO peut être vu comme un intégrateur de phase. Sa fonction de transfert dans le domaine Laplace est donc $VCO(s)=\frac{1}{s}$.

Le tableau suivant montre l'erreur en régime permanent des boucles d'ordre 1,2 et 3 en fonction de différentes perturbations en entrée : saut de phase, rampe de phase, rampe de fréquence, ainsi qu'un jerk.

Tableau III

Erreur en fonction de l'ordre de la boucle et du type de perturbation

| | **Erreur Ordre 1** | **Erreur Ordre 2** | **Erreur Ordre 3** |
|---|---|---|---|
| **Saut phase** | 0 | 0 | 0 |
| **Saut fréquence** | $\frac{\Delta\omega_e}{\omega_0}$ | 0 | 0 |
| **Rampe de fréquence** | $\infty$ | $\frac{R}{\omega_0^2}$ | 0 |
| **« Jerk »** | $\infty$ | $\infty$ | $\frac{J}{\omega_0^3}$ |

Pour la boucle d'ordre 3, on a :

1. La fonction de transfert du système :

$$G(s)=\frac{\theta_v(s)}{\theta_e(s)}=\frac{\omega_0\left(b_3 s^2+a_3\omega_0 s+\omega_0^2\right)}{s^3+b_3\omega_0 s^2+a_3\omega_0^2 s+\omega_0^3} \qquad \textbf{(3.25)}$$

2. La fonction de transfert de l'erreur :

$$E(s) = \frac{e(s)}{\theta_e(s)} = \frac{s^3}{s^3 + b_3\omega_0 s^2 + a_3\omega_0^2 s + \omega_0^3} \tag{3.26}$$

### 3.3.1 Étude graphique de G(s) et de E(s)

Les valeurs suggérées pour la boucle d'ordre 1 par Kaplan[**1**] sont $\omega_0 = 4*B_n = 4*10 = 40$ Hz. Les valeurs suggérées pour la boucle d'ordre 2 sont : $\omega_0 = B_n/.53 = 10/.53 = 18.868$ et $a_2 = 1.414*\omega_0 = 26.679$. Les valeurs suggérées pour la boucle d'ordre 3 sont : $\omega_0 = B_n/0.7845 = 10/.0.7845 = 12.747$, $a_3 = 1.1$ et $b_3 = 2.4$. On utilise ces valeurs pour générer les **Figure 27** et **Figure 28**.

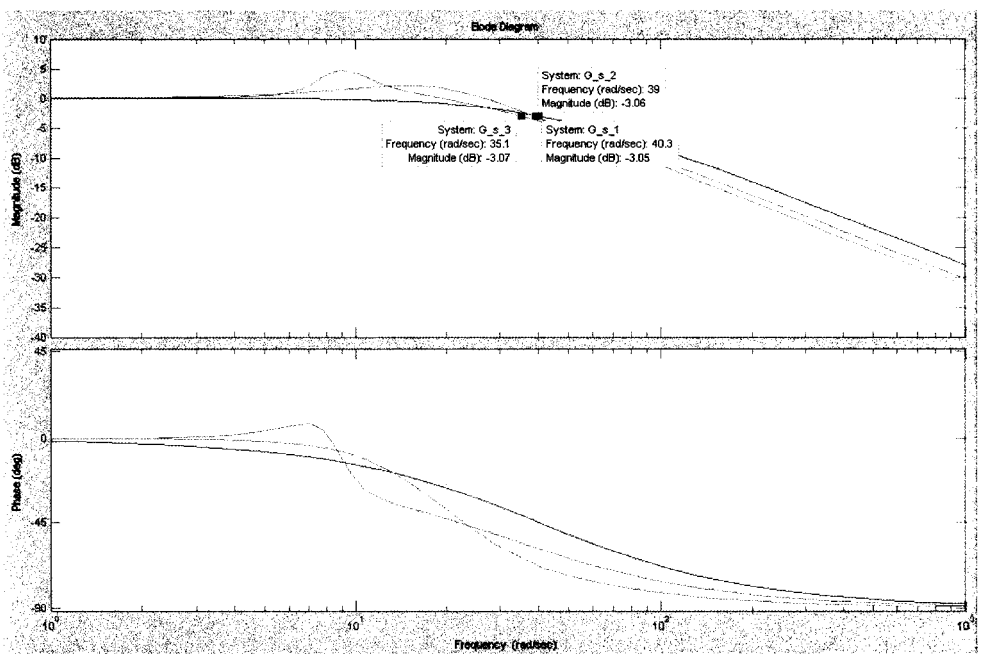



Figure 27 Réponse en fréquence de G(s), boucles d'ordre 1,2 et 3

Sur la **Figure 27**, il est difficile de différencier l'ordre des fonctions de transfert car l'échelle en dB n'est pas assez grande. On voit par contre que la fréquence de coupure des systèmes d'ordre 1,2 et 3 sont très proches (40.3, 39 et 35.1 rad/s).

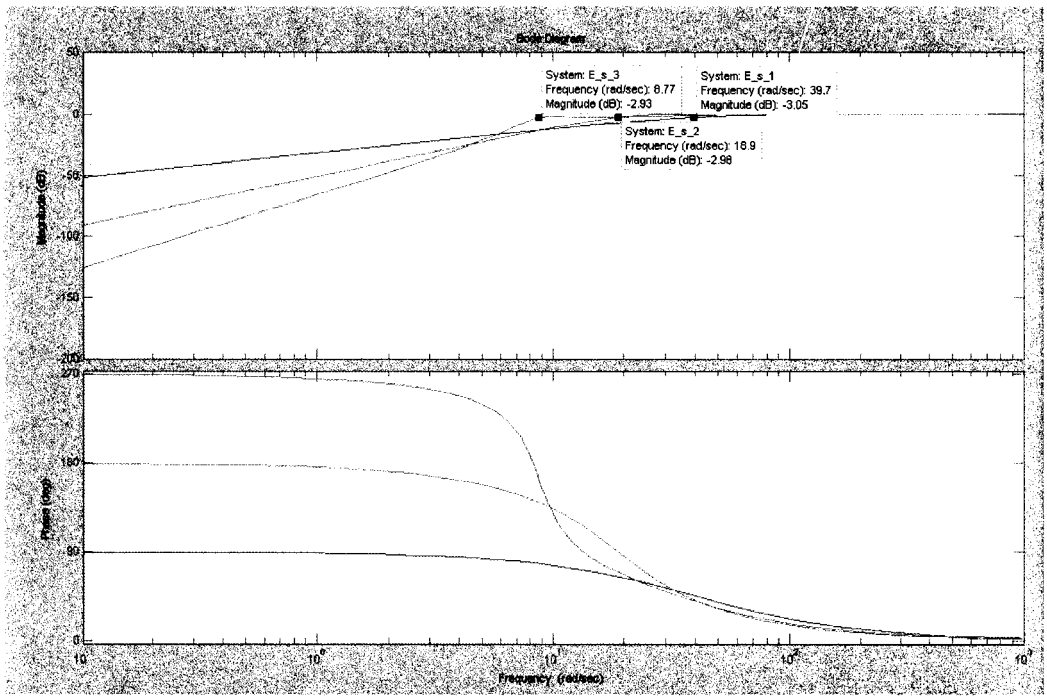



Figure 28 Réponse en fréquence de E(s), boucles d'ordre 1,2 et 3

Sur la **Figure 28**, on distingue plus facilement l'ordre de la fonction de transfert E(s). On voit que pour l'ordre 1, la pente d'atténuation est de 20 dB/octave, pour l'ordre 2, elle est de 40 dB/octave et pour l'ordre 3, on a une pente de 60 dB/octave. Les fréquences de coupure pour les ordres 1, 2 et 3 sont respectivement de 39.7, 18.9 et 8.77 rad/s.

## 3.4 Architecture des boucles du récepteur GPS numérique

Basée sur les boucles de Costas, les architectures possibles d'un récepteur GPS sont limitées. Les concepteurs du système GPS l'ont conçu afin que le système soit performant en présence de bruit et l'architecture du récepteur démontre cette facette avec plusieurs étapes dans le processus. La **Figure 29** montre le schéma bloc d'un récepteur GPS.

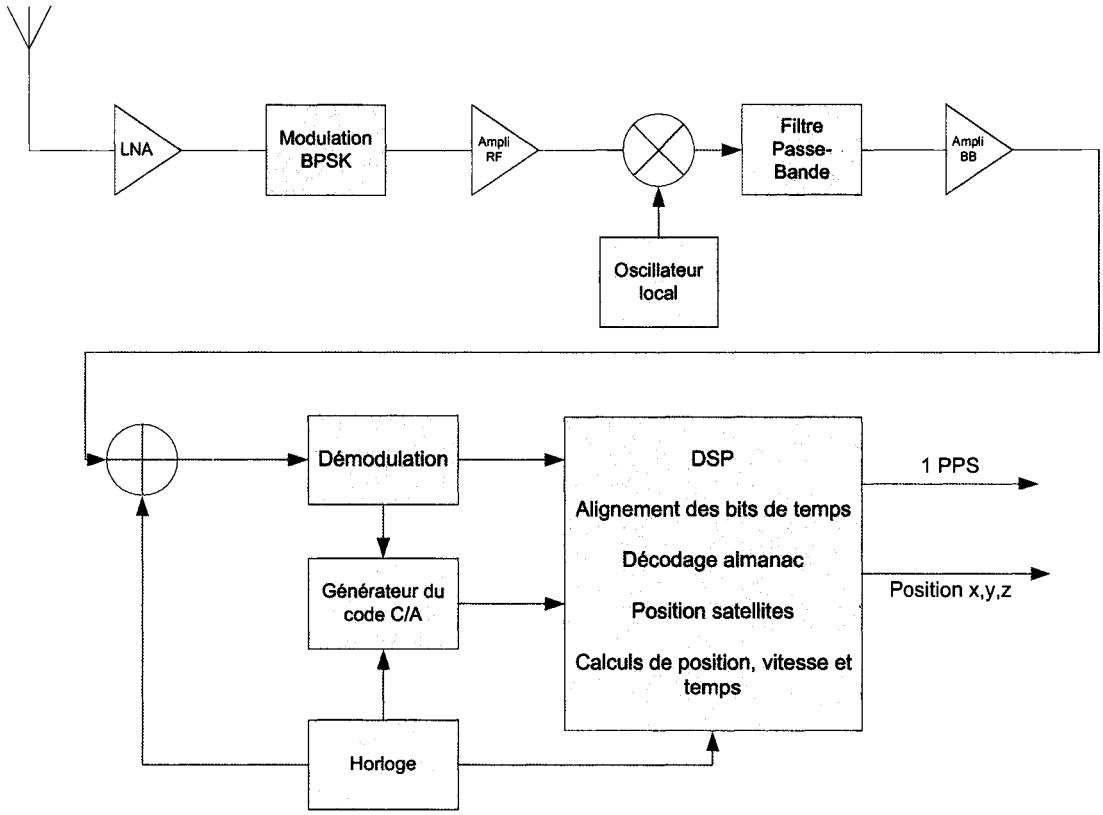



Figure 29 Schéma bloc d'un récepteur GPS

### 3.4.1 Principe de fonctionnement

Le récepteur GPS reçoit la fréquence L1 à un niveau de signal de -130 dBm et la largeur de sa bande est de 2,046 MHz. Un premier étage constitué d'un amplificateur faible bruit LNA (*Low Noise* Amplifier) amplifie le signal.

À la suite de cet amplificateur, un filtre passe-bande isole les fréquences utiles du signal (fréquences radio du satellite) et un amplificateur RF ajoute un gain supplémentaire. À la sortie du mélangeur, le signal est de nouveau filtré à l'aide d'un filtre bande de base.

Enfin, un amplificateur bande de base, qui permet de passer de la fréquence RF en bande de base, amplifie le signal contenu dans la bande de fréquence désirée.

Le DSP génère le code pseudo-aléatoire et le compare au signal reçu. De plus, il analyse l'almanach et apporte les corrections appropriées. Basé sur ce principe, le récepteur conçu respecte les normes de la réalité, mais l'analyse des almanachs n'est pas considérée pour ce projet. Cependant, cet aspect pourra être ajouté au projet afin d'améliorer les performances du modèle.

### 3.4.2 Architecture détaillée du récepteur GPS

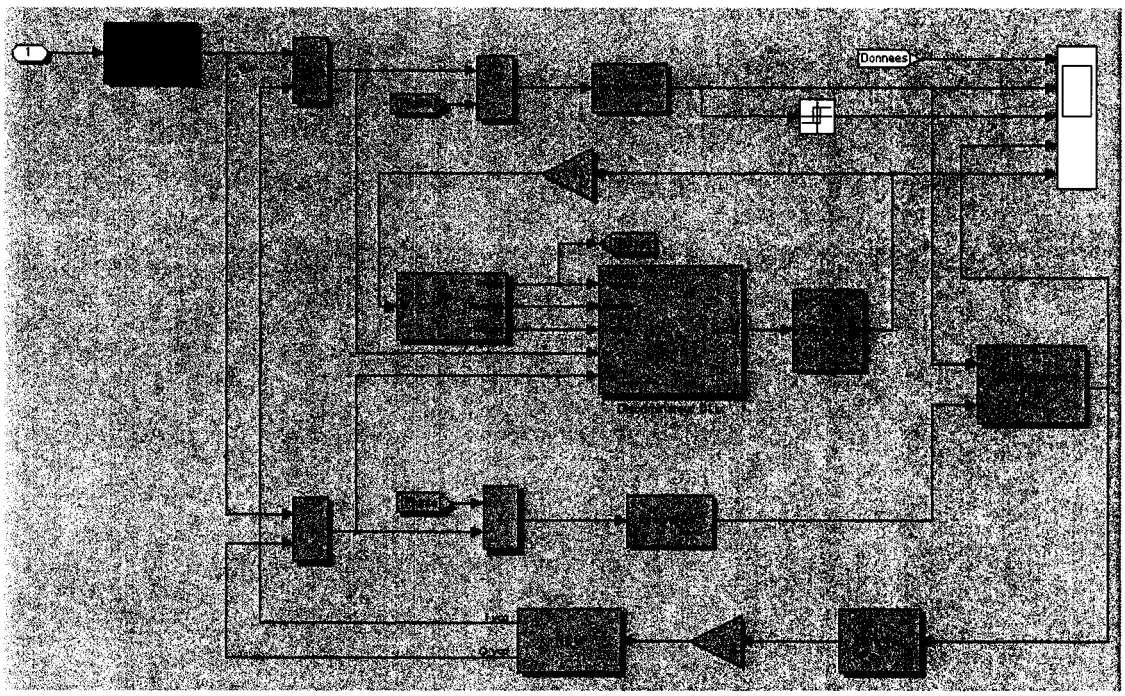



Figure 30 Schéma d'un récepteur GPS

La **Figure 30** montre l'architecture d'un canal de réception GPS. On y distingue deux types de boucles : la boucle de phase et la boucle de code. La boucle de phase permet

d'extraire le code pseudo aléatoire et la boucle de code rend possible l'extraction des données envoyées par le satellite GPS, appelé aussi désétalement spectral. Sur le schéma Simulink, les blocs de la partie du haut et du bas correspondent à la boucle de maintien de la porteuse (boucle de phase), tandis que la partie centrale est la boucle de code. Les données sont extraites à la sortie du deuxième multiplicateur (voie en phase). Un bloc « relais » est utilisé pour extraire les données. On remarque à l'entrée du récepteur le filtre anti-brouilleur qui accroît la robustesse du système face aux brouilleurs de toutes sortes.

La synchronisation se fait simultanément :

1- En fréquence sur la porteuse.
2- En temps sur le code.

Les sections suivantes exposent plus en détails les différentes composantes du récepteur, les boucles PLL et DLL, les filtres et les discriminateurs.

### 3.4.3 Boucle PLL

La boucle PLL se base sur les boucles de Costas, expliquées plus tôt dans le texte (section 3.3).

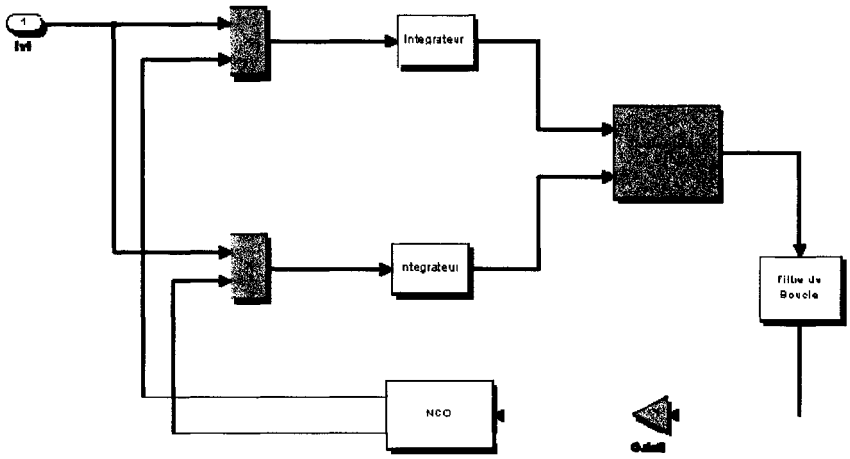



Figure 31 Boucle PLL de la porteuse

À l'entrée, notre signal GPS est multiplié avec un signal en phase (en haut), aussi appelé la voie I, et un autre en quadrature (en bas), appelé la voie Q. Ces signaux passent à travers un filtre passe-bas afin d'éliminer les produits d'inter-modulation de haute fréquence.

### 3.4.3.1 Le bloc intégrateur

Dans notre récepteur, le filtre passe-bas se traduit par un intégrateur d'une période de 1 ms. La période d'intégration doit être relativement petite pour que la boucle puisse réagir rapidement aux fluctuations d'entrée et aux perturbations subies par le signal. La **Figure 32** représente les intégrateurs que nous utilisons dans notre système autant au niveau de la boucle PLL que DLL. La seule différence entre la boucle PLL et DLL est le temps d'intégration, qui se doit d'être plus long pour la récupération du code (1 ms dans le cas de la boucle de phase et 10 ms, pour la boucle de code).

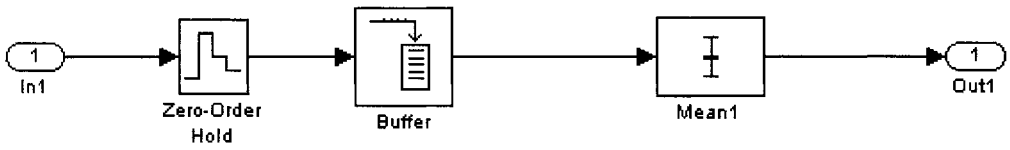



Figure 32 Schéma des intégrateurs

Pour représenter un intégrateur, nous avons utilisé la définition théorique de l'intégration. On sait que l'intégration est la moyenne de notre fonction évaluée entre deux temps, $t_1$ et $t_2$. Donc, au lieu d'utiliser le bloc *integrate and dump* déjà inclus dans la librairie de Simulink, on utilise un tampon qui emmagasine les valeurs de la fonction pendant un temps déterminé par l'usager et on en calcule la moyenne. Il faut maintenant établir un lien entre le nombre de données à emmagasiner et le temps d'intégration. Nous savons que nous avons 1023 échantillons par seconde ce qui nous donne la relation suivante :

$$No\_échantillons = 1023 * \frac{F_s}{F_c} * T\_PLL \tag{3.27}$$

où T_PLL est le temps d'intégration choisi par l'usager (en milliseconde). Le temps d'intégration pour la boucle PLL doit être court afin que celle-ci réagisse rapidement. Pour que le système soit efficace, la valeur du temps d'intégration ne doit pas être trop court afin que l'ajustement apporté au NCO soit plausible. Comme il s'agit d'une récupération sur la porteuse, il nous faut prendre une décision en utilisant les deux autres signaux portés par cette fréquence. La période des données est de 0,02 seconde (50 Hz)tandis que celle d'un code de Gold est de près de 1 ms (1,023 MHz). Cette dernière valeur permet à la boucle de prendre une décision efficace sur la valeur de sortie de l'intégrateur. En intégrant sur 1 ms, nous accumulons assez d'échantillons pour calculer l'erreur, et de plus, nous intégrons sur un code de Gold au complet ce qui limite les erreurs dues à la boucle DLL.

### 3.4.3.2 Le bloc discriminateur

Le discriminateur permet de déterminer l'erreur de phase entre le signal en phase et le signal en quadrature au niveau de la porteuse. Notre simulateur comporte plusieurs types de discriminateurs. Un interrupteur placé à la sortie des discriminateurs nous permet de choisir entre les différents discriminateurs disponibles. La variable se situe dans le menu principal et est choisie par l'usager. Le **Tableau IV** montre les discriminateurs implémentés dans le récepteur.

Tableau IV

Discriminateur de la boucle PLL et l'erreur de phase

| **Discriminateur** | **Erreur de phase** |
|---|---|
| Sign(I)*Q | Sin $\phi$ |
| $I_{PL}$*$Q_{PL}$ | $Sin^2$ $\phi$ |
| $Q_{PL}/I_{PL}$ | Tan $\phi$ |
| Arctan($Q_{PL}/I_{PL}$) | $\phi$ |

Dans le **Tableau IV**, quatre opérations peuvent être choisies pour extraire l'erreur de phase. En analysant le tableau, on constate que, pour une erreur de phase plus précise, nous devons utiliser la fonction arctan(Q/I). Cette opération nous permet de récupérer plus rapidement la porteuse car on obtient l'erreur de phase $\phi$ exacte contrairement à l'approximation donnée par les trois autres opérations. De plus, dans le cas d'un déphasage très grand, le théorème de la proximité de $\phi$ et de sin($\phi$) devient non valide. Donc, la réaction pour retrouver la porteuse est plus courte, ce qui est important pour des perturbations crées soit par des brouilleurs, l'effet Doppler ou les multi-trajets. La

**Figure 33** montre la réponse théorique des différents discriminateurs de la boucle PLL en fonction de l'erreur de l'entrée.

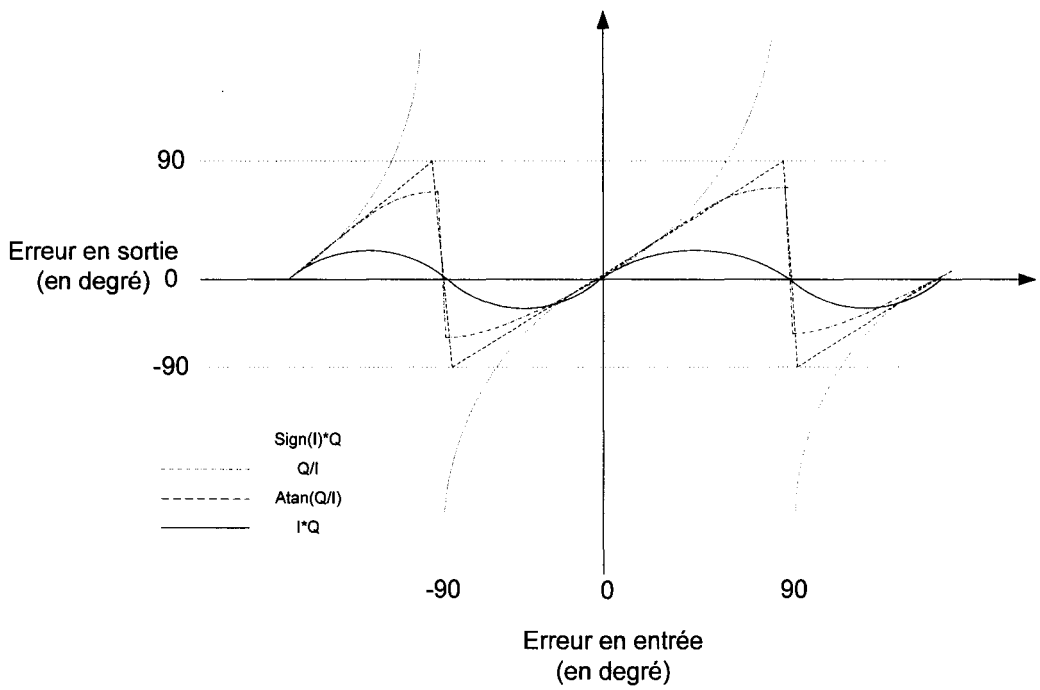



Figure 33 Réponse des discriminateurs de la boucle de phase

### 3.4.3.3 Le bloc filtre

Les filtres de boucle servent à réduire le bruit afin d'augmenter la précision de l'erreur de phase donné par les discriminateurs. Avec leurs coefficients typiques, les filtres proposent un gain à l'erreur de phase dont il faut tenir compte dans la conception de ces filtres. L'ordre des filtres vient en réponse au stresse dynamique du signal et aide la récupération du signal original (dans le cas de la PLL, ce signal est la porteuse). La **Figure 34** montre le détail du bloc filtre.

Dans la boucle, un filtre est nécessaire, avant le NCO, pour nous permettre d'ajuster les pôles de la fonction de transferts de la boucle. On peut ainsi ajuster le dépassement, le temps de réponse et l'erreur en régime permanent. On note aussi que le filtre utilisé pour la boucle de code aura la même architecture que celui utilisé pour la boucle de phase. Ces filtres sont paramétrables par l'usager au lancement de la simulation en correspondance avec les valeurs du **Tableau V**. Le coefficient T des filtres correspond au temps d'intégration de l'intégrateur situé avant le discriminateur.

Tableau V

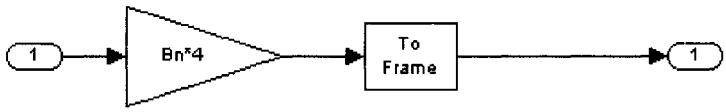
Caractéristiques des filtres des boucles PLL et DLL

| Ordre de la boucle | $B_n$(Hz) | Coefficients typiques des filtres | Erreur en régime permanent | Fonction de transfert du filtre |
|---|---|---|---|---|
| 1 | $\frac{\omega_0}{4}$ | $\omega_0$<br>$B_n = 0.25\omega_0$ | $\frac{(dR/dt)}{\omega_0}$ | $H_1(s) = \frac{\omega_0}{s}$ |
| 2 | $\frac{\omega_0(1+a_2^2)}{4a_2}$ | $\omega_0^2$<br>$a_2\omega_0 = 1.414\omega_0$<br>$B_n = 0.53\omega_0$ | $\frac{(dR^2/dt^2)}{\omega_0^2}$ | $H_2(s) = \frac{\omega_0}{s^2}(\omega_0 + a_2 \times s)$ |
| 3 | $\frac{\omega_0(a_3b_3^3 + a_3 - b_3)}{4(a_3b_3 - 1)}$ | $\omega_0^3$<br>$a_3\omega_0^2 = 1.1\omega_0^2$<br>$b_3\omega_0 = 2.4\omega_0$<br>$B_n = 0.7845\omega_0$ | $\frac{(dR^3/dt^3)}{\omega_0^3}$ | $H_3(s) = \frac{\omega_0}{s^3}(\omega_0^2 + a_1 * s * \omega_0 + b_1 * s^2)$ |

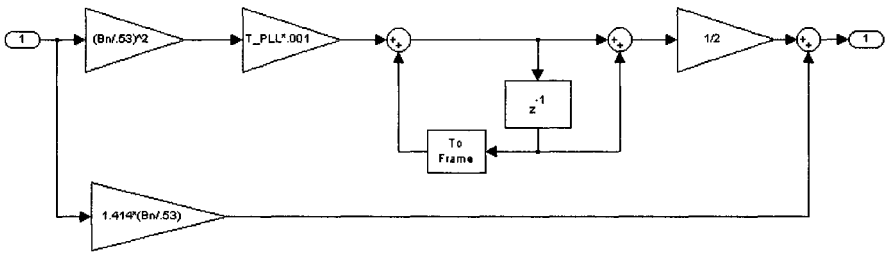
Le tableau ci-dessus nous permet de calculer les coefficients des filtres selon l'ordre de la boucle. Ainsi, si nous avons un filtre de 1$^{er}$ ordre et que nous voulons une bande passante de 18 Hz, la valeur de $\omega_0$ se calcule comme suit :

$$B_n = 0.25 \times \varpi_0$$
$$\varpi_0 = \frac{B_n}{0.25} = \frac{18}{0.25} = 72\ Hz \qquad \textbf{(3.28)}$$

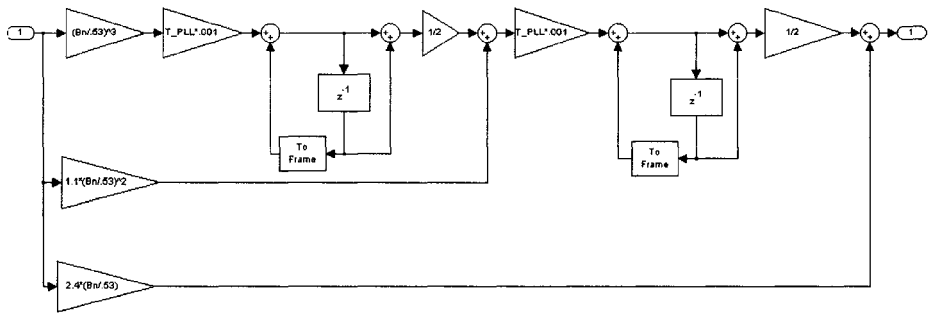
La valeur de la largeur de bande $B_n$, décidé par l'utilisateur au niveau du modèle simulé, ne pourra dépasser 18 Hz. $B_n$ est la bande équivalente de bruit de la boucle PLL. C'est comme si l'on concentrait tout le bruit dans cette bande passante et cette valeur est en fonction du rapport signal sur bruit. La valeur de R est la distance entre le satellite et le récepteur. Les trois cas présentés à la **Figure 34** montrent les filtres d'ordre 0, 1 et 2.




A) Filtre d'ordre 0




B) Filtre d'ordre 1




C) Filtre d'ordre 2

Figure 34 Schéma des filtres numériques d'ordre 0, 1 et 2

Les graphiques suivants (**Figure 35** à **Figure 38**) nous montrent l'erreur de phase à la sortie du discriminateur pour les 3 ordres de filtres utilisés. Le discriminateur choisi est *arctan*. Les graphiques donnent l'erreur de phase pour les 3 perturbations suivantes : un saut de phase, un saut de fréquence et une rampe de fréquence. Nous montrerons que les courbes obtenues en simulation correspondent à l'étude théorique effectuée.

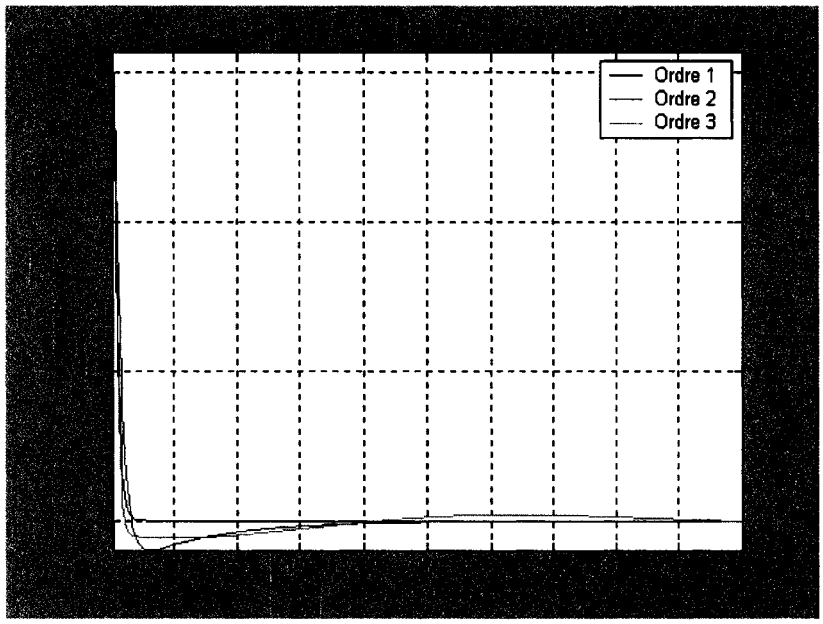



Figure 35 Réponse du discriminateur PLL à un saut de phase

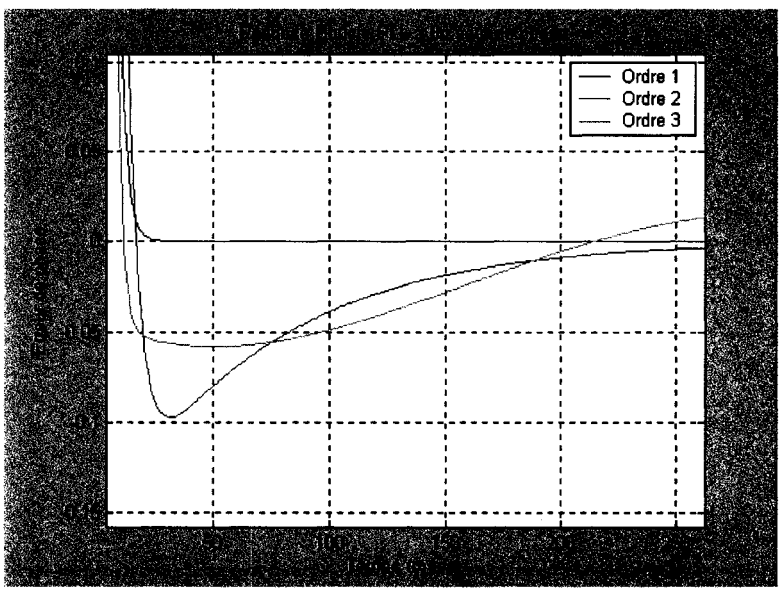



Figure 36 Réponse à un saut de phase entre 0 et 250 msec

Tel que prévu par l'étude théorique, la **Figure 36** montre qu'on obtient une erreur nulle pour la boucle d'ordre 1,2 et 3 en réponse à un saut de phase.

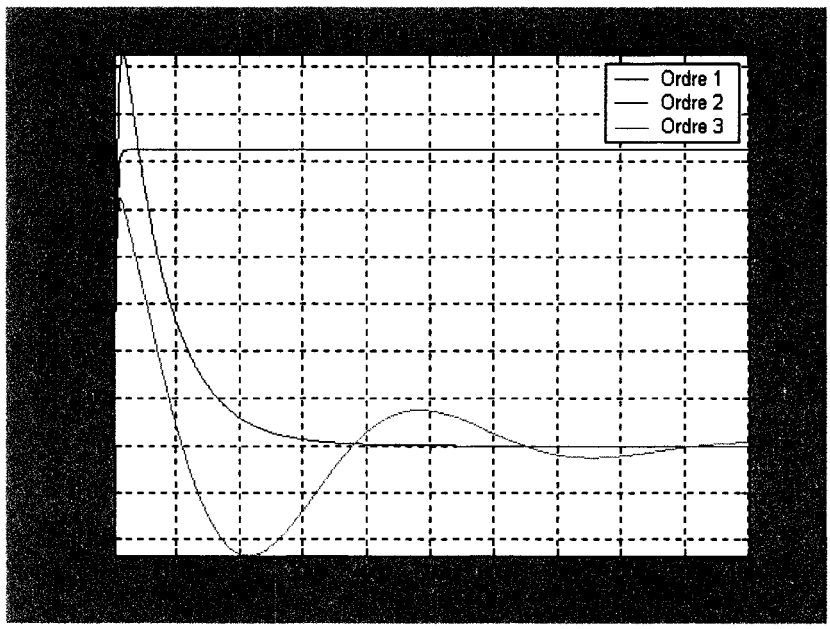



Figure 37 Erreur à la sortie du PLL pour un saut de fréquence de 5 Hz

La **Figure 37** montre que pour un saut de fréquence, la boucle PLL d'ordre 1 produit une erreur constante. On peut donc affirmer qu'une boucle d'ordre 1 ne suivra jamais le signal et sera toujours en retard. Cette situation est non-désirée car il nous sera impossible de retrouver nos données si notre boucle ne peut se « barrer » sur la porteuse. On constate, en accord avec l'analyse théorique, que l'erreur en régime permanent pour une boucle d'ordre 1 est de $\frac{\Delta\omega_e}{\omega_0}=\frac{5}{40}=0.125$. Pour la boucle d'ordre 2 et 3, l'erreur en régime permanent est de 0.

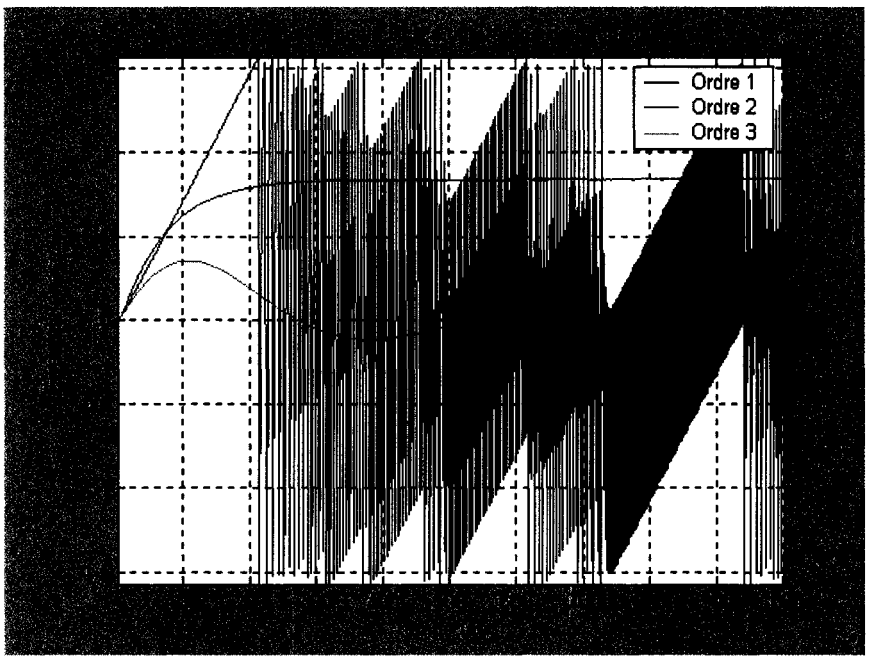



Figure 38 Erreur à la sortie du PLL pour une rampe de fréquence

La **Figure 38** montre la réponse de la boucle PLL à une rampe de fréquence. On constate, en accord avec l'analyse théorique, que l'erreur en régime permanent pour une boucle d'ordre 1 est infinie. On remarque que la boucle « décroche » lorsque l'erreur de phase atteint $\pi/2$.

L'étude théorique prédit pour la boucle d'ordre 2 une erreur en régime permanent de $\frac{R}{\omega_0^2}=\frac{300}{\left(\frac{10}{0.53}\right)^2}=0.8247$. On obtient en simulation une erreur de 0.82.

Il est bien évident que, dans la réalité, ces perturbations sont regroupées et ne sont pas nécessairement subit de façon individuelle. Il faut donc prendre en note que la perturbation la plus globale, l'effet Doppler, est beaucoup plus proche de la réalité que les deux autres et ses résultats caractérisent mieux le comportement de notre boucle.

Comme le prédit l'analyse théorique, on obtient pour la boucle d'ordre 3 une erreur nulle. Ces situations permettent d'affirmer que notre modèle est conforme à la théorie et à la réalité. La boucle de phase donne les valeurs théoriques et est capable de contrer les perturbations appliquées sur la porteuse.

### 3.4.4 Boucle DLL

La boucle de code permet d'effectuer le désétalement spectral. Pour ce faire, elle doit générer un code pseudo-aléatoire en phase avec celui qui est reçu. En séparant les deux boucles (PLL et DLL), le schéma de boucle DLL isolé devient (voir **Figure 39**) :

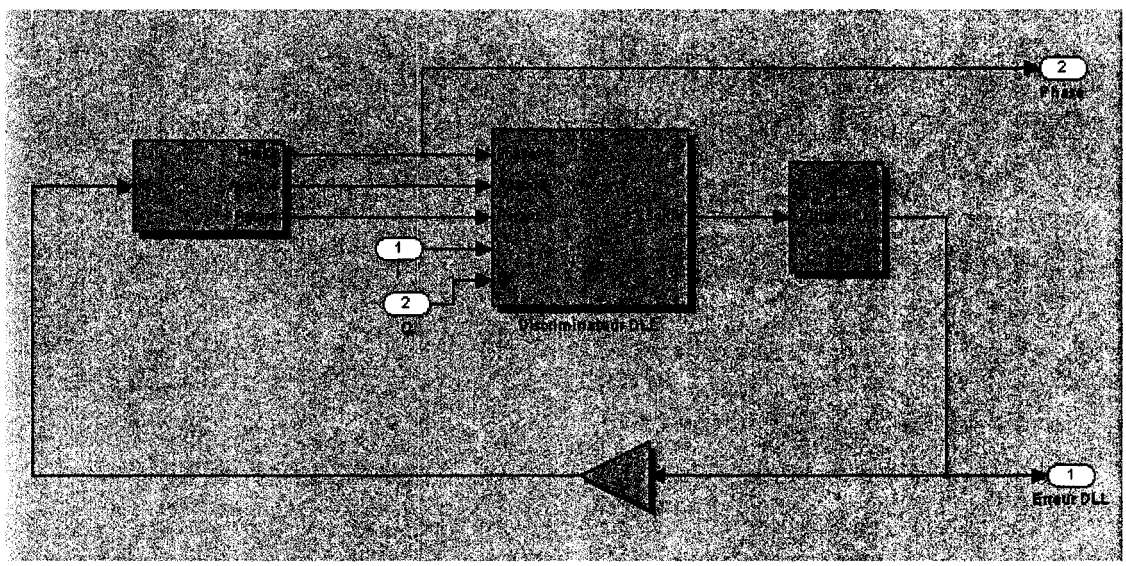



Figure 39 Schéma de la boucle de code (DLL)

#### 3.4.4.1 Le bloc discriminateur

Le discriminateur DLL permet de trouver l'erreur de phase dans le code. En plus des signaux avance, phase et retard provenant du NCO de la boucle DLL, nous utilisons les

signaux I et Q provenant de la boucle PLL. L'usager a le choix entre quatre discriminateurs. Le **Tableau VI** montre les différents discriminateurs.

Tableau VI

Discriminateur de code

| **Discriminateur** | **Équation** |
|---|---|
| *Produit croisé* | $\sum(I_a - I_r)\times I_p + \sum(Q_a - Q_r)\times Q_p$ |
| *Puissance avance moins retard (AMR)* | $\sum\left(I_a^{\,2} - Q_a^{\,2}\right) - \sum\left(I_r^{\,2} - Q_r^{\,2}\right)$ |
| *Enveloppe AMR* | $\sum\sqrt{\left(I_a^{\,2} - Q_a^{\,2}\right)} - \sum\sqrt{\left(I_r^{\,2} - Q_r^{\,2}\right)}$ |
| *Enveloppe AMR normalisée* | $\dfrac{\sum\sqrt{\left(I_a^{\,2} - Q_a^{\,2}\right)} - \sum\sqrt{\left(I_r^{\,2} - Q_r^{\,2}\right)}}{\sum\sqrt{\left(I_a^{\,2} - Q_a^{\,2}\right)} + \sum\sqrt{\left(I_r^{\,2} - Q_r^{\,2}\right)}}$ |

Les discriminateurs de la boucle DLL ont la réponse théorique suivante (à l'ajout de perturbations au code) ;

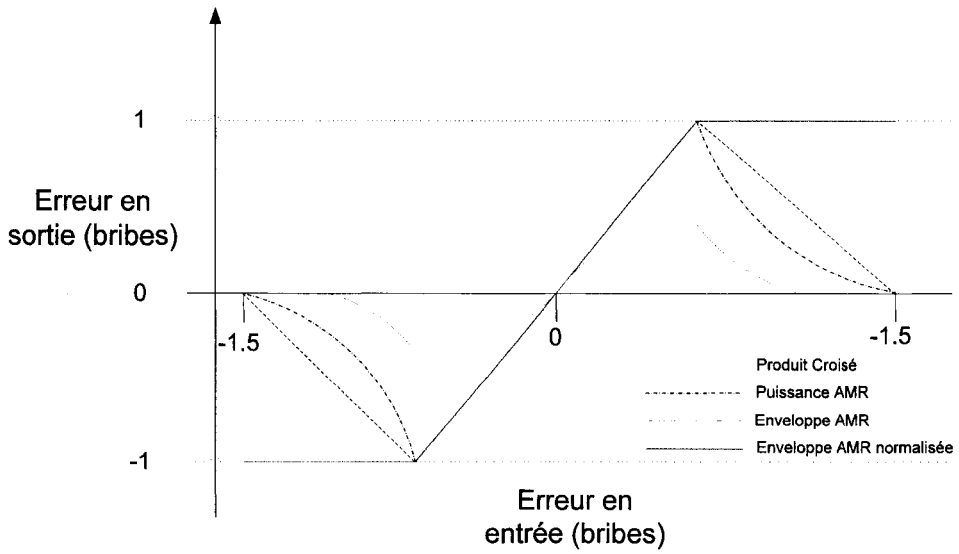



Figure 40 Réponse des discriminateurs de la boucle DLL

La **Figure 40** nous prouve ce que l'on affirmait plus tôt, c'est-à-dire que les discriminateurs ne tiennent pas compte d'une erreur plus grande d'une bribe.

La **Figure 41** montre le schéma du discriminateur DLL.

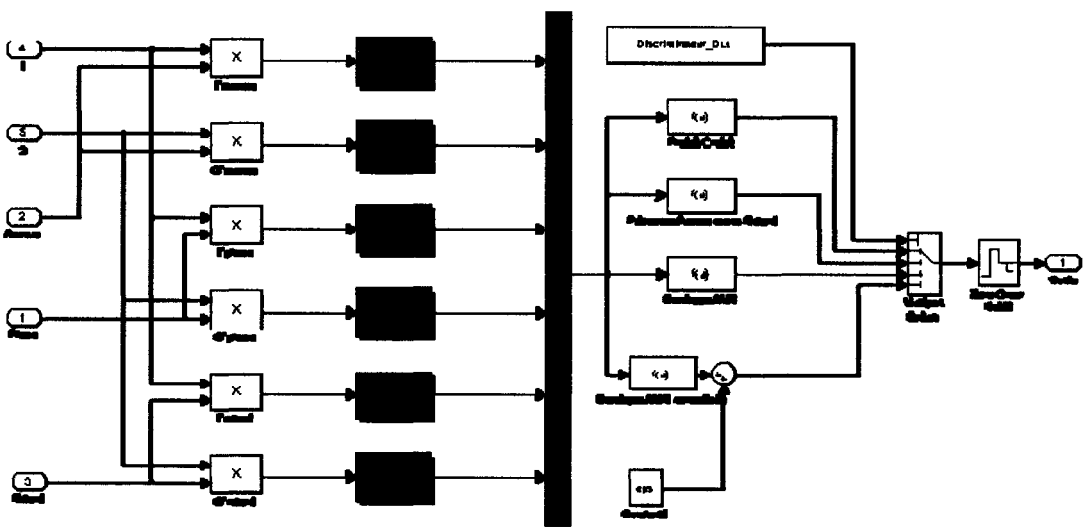


Figure 41 Schéma du discriminateur DLL

Suite aux intégrateurs, nous avons placé un multiplexeur qui a comme avantage d'identifier les signaux à la sortie par son numéro d'entrée. Par exemple, le premier signal au début du multiplexeur, $I_{avance}$, s'appelle u(1), $Q_{avance}$ s'appelle u(2), etc. De cette façon, il est plus facile de traduire mathématiquement les fonctions représentant l'action de chaque discriminateur. Dans le cas du discriminateur « Puissance Avance moins Retard », la fonction devient :

$$erreur = (u(1)^2 - u(2)^2) - (u(5)^2 - u(6)^2) \quad \textbf{(3.29)}$$

Le bloc intégrateur est le même bloc que celui utilisé pour la boucle de phase mais le temps d'intégration est un peu plus long. Dans ce cas, le temps d'intégration est de 20 ms car il faut un nombre minimum d'échantillon avant de calculer l'erreur. De plus, ce temps d'intégration plus long fait de la boucle DLL la plus robuste au bruit. Le nombre d'échantillons étant plus élevé, les erreurs introduites par le bruit ont une influence minime.

On utilise les mêmes filtres que dans le cas de la boucle de phase mais avec des coefficients différents car le temps d'intégration et la bande de bruit équivalente sont différents. Pour la boucle de code, la bande $B_n$ se limite à 1 Hz.

Pour la génération des codes en phase, en retard et en avance, nous utilisons le même générateur de code Gold que la source. Par contre, l'entrée du NCO déterminant la fréquence du code est commandée par l'erreur calculée par le discriminateur DLL. Pour le code en avance, on n'utilise aucun délai. Pour le code en phase, on utilise un délai de $\frac{Fs}{Fc} * \frac{1}{2}$.

Les figures suivantes (**Figure 42** à la **Figure 43**) montrent la réponse de la boucle de code. Dans un premier cas, nous avons appliqué un saut de phase à notre code. Le temps

de réaction est assez long, ce qui est normal puisque le temps d'intégration est de 20 ms. Le signal supérieur (en haut sur le graphique) est la donnée envoyée, les signaux d'en bas sont les erreurs à la sortie du discriminateurs DLL (en bas complètement) et PLL (troisième graphe), le signal entre la sortie du discriminateur et la donnée est la sortie de l'intégrateur de la boucle PLL, endroit où l'on doit recouvrer nos données. Il est à noter que dans cette simulation l'ordre du filtre considéré est 0.

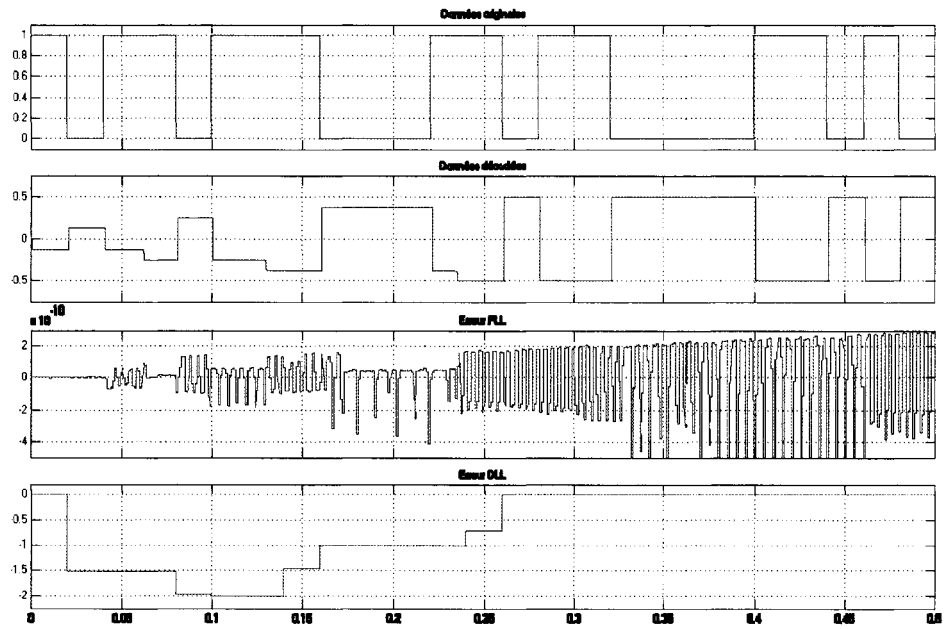


Figure 42 Réponse de la boucle de code à un saut de phase

On peut conclure que notre système fonctionne bien car nous retrouvons en simulation des résultats similaires qu'en théorie (à un détail près, c'est-à-dire que les points maximum et minimum sont inversés). En effet, en théorie, le point minimum arrive avant le point maximum, mais le résultat ne change en rien la position du zéro, qui est notre point de détection.

Enfin, la dernière courbe étudiée pour comprendre le fonctionnement de notre système est l'effet de l'ordre du filtre sur une perturbation. Nous pouvons donc tester l'effet du

filtre sur une rampe de fréquence, qui est l'équivalent de l'effet Doppler. La courbe obtenue est montrée à la **Figure 43**.

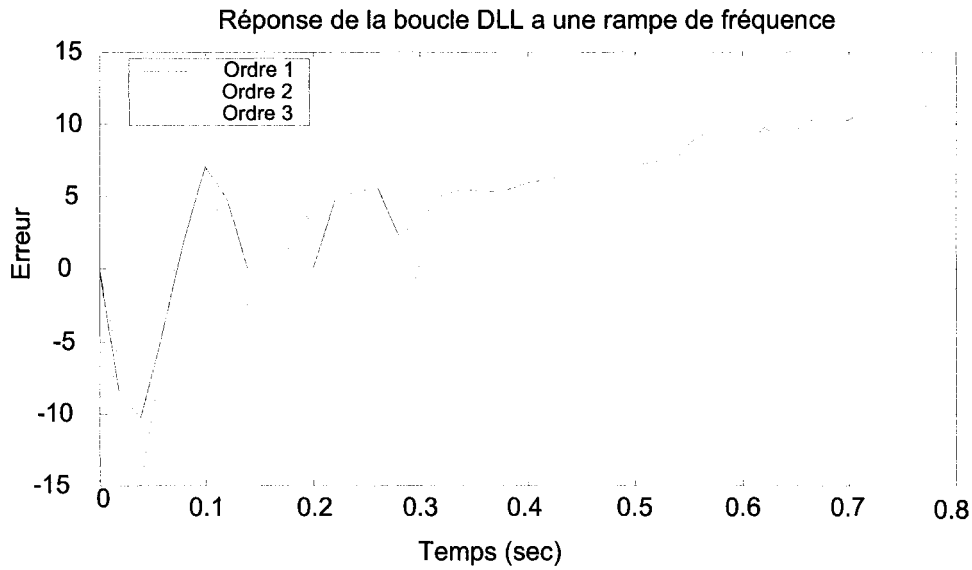



Figure 43 Réponse des filtres de la boucle DLL avec une perturbation

On remarque qu'après un certain temps, les boucles d'ordre 1 et 2 décrochent mais par contre, la boucle d'ordre 3 réussi à compenser la déviation en fréquence. On sait que la fréquence se déplace à cause du mouvement du satellite. Donc, cette particularité de la boucle d'ordre 3 nous permet de limiter le Doppler causé sur le code.

## 3.5 Impact des différents paramètres du modèle sur les performances

Parmi les paramètres du modèle, certains sont importants pour le fonctionnement des boucles PLL et DLL ainsi que pour la simulation des perturbations sur la source du signal GPS. Cependant, dans la simulation d'une chaîne de communication GPS, il existe différents processus de numérisation. L'échantillonnage direct se fait directement à l'entrée du récepteur à la fréquence L1 du signal GPS. Évidemment, à cause de la

fréquence élevée de ce signal, les convertisseurs doivent être extrêmement performants lors de cet échantillonnage. Afin de contourner cette difficulté, le signal peut subir un premier échantillonnage à une fréquence intermédiaire. De cet étage IF, il est donc alors plus facile de numériser le signal. Cependant, en multipliant notre signal GPS par un oscillateur local, certaines erreurs sur la fréquence peuvent être introduites. Il est donc important de bien choisir notre fréquence d'échantillonnage afin d'introduire le moins d'erreur et de perturbation possible. Un choix possible comme fréquence d'échantillonnage, est d'être relié avec le taux du code pseudo-aléatoire (1.023Mbps pour le code C/A). Mais il est important que la fréquence d'échantillonnage ne soit pas un multiple de ce taux car cela empêcherait une bonne résolution de la position recherchée. Pour en accroître la précision, il existe 2 solutions : soit d'augmenter la fréquence d'échantillonnage, soit d'utiliser un échantillonnage asynchrone.

En plus de ce taux sur le code C/A, il faut tenir compte des différentes perturbations sur le signal, comme l'effet Doppler. La fréquence d'échantillonnage ne doit pas être un multiple de la somme du taux de code et de la fréquence Doppler pour les mêmes raisons expliquées plus haut. Dans le cas du signal GPS, cette dernière n'excède pas la valeur de 4kHz.

Nous savons que la bande d'analyse pour une fréquence d'échantillonnage $f_s$ se situe entre 0 et $\frac{f_s}{2}$. La loi de Nyquist nous rappelle que pour échantillonner un signal et de le récupérer sans distorsion, nous devons avoir une fréquence d'échantillonnage supérieur à la moitié de la fréquence du signal. Puisque nous avons un étage IF en bande de base, nous nous devons d'avoir au moins le double de la fréquence d'entrée.

$$f_0 = f_i - n\frac{f_s}{2} \tag{3.30}$$

$$f_0 < \frac{f_i}{2} \tag{3.31}$$

Les deux relations précédentes nous indiquent la valeur de $f_0$, la fréquence de sortie, en fonction de $f_s$, la fréquence d'échantillonnage, et de $f_i$, la fréquence IF. La valeur de $n$ est entière et permet de situer la bande d'analyse sur l'échelle des fréquences. Si la fréquence en entrée, donc de l'étage IF, se situe entre $nf_s$ et $(2n+1)\cdot f_s/2$, la fréquence de sortie en bande de base est repliée en mode direct. Et c'est dans cette zone qu'il est préférable d'avoir notre signal à traiter. Au-delà de cette zone, on assiste à une partie superposée du signal dans la bande de sortie. Cette situation est à éviter car nous aurons une perte d'information de notre signal, donc une diminution de la précision. Afin d'éviter cette situation, les deux expressions suivantes se doivent d'être respectés :

$$f_0 = f_i - n\frac{f_s}{2} \approx \frac{f_s}{4} \tag{3.32}$$

$$f_s > 2\Delta f \tag{3.33}$$

La relation 3.32 fait en sorte que nous respectons la loi de Nyquist tandis que la relation 3.33 nous permet de centrer le signal de sortie autour de la fréquence de sortie. Cette expression, 3.53, nous permet en même temps, de choisir notre fréquence d'échantillonnage lors de la conception du récepteur. Dans un récepteur GPS, il est connu que la fréquence d'échantillonnage est de 5 MHz, ce qui, une fois dans l'expression 3.34, nous donne :

$$f_{IF} = 2.5n + 1.25MHz \tag{3.34}$$

où $n$ est un entier au choix de l'usager. Cependant, certains choix de $n$ sont plus judicieux que d'autres, car l'efficacité et la précision en dépendent. Par exemple pour

$n$=1, la valeur de la fréquence de sortie est de 0.25 MHz à 2.25 MHz, ce qui couvre plusieurs octaves. Dans la pratique, on se doit de limiter le nombre d'octaves franchis afin de diminuer l'impact des harmoniques générées par le mélangeur.

Évidemment, l'impact du choix de la fréquence d'échantillonnage a des répercussions sur le design du système mais aussi sur l'équipement que nous allons choisir pour la partie temps réel. Mais cette fréquence, tel que décrite dans les expressions précédentes, a une précision limitée dû à son instabilité. En effet, une erreur de 100 Hz sur la précision de la fréquence d'échantillonnage, se répercute en une erreur de 400 Hz, sans oublier l'erreur provoquée par la fréquence Doppler. Donc, en utilisant un étage à fréquence IF, on introduit une imprécision due à l'échantillonnage. En admettant que l'on décide d'utiliser un échantillonnage direct, sans étage IF, tout en conservant une fréquence d'échantillonnage de 5 MHz, puisque $n$ est plus élevé en échantillonnage direct, l'imprécision s'amplifie et devient 41 800 Hz. C'est une très grande valeur et non acceptable au niveau de la précision. C'est pourquoi, lors du design du modèle, la méthode à conversion descendante est retenue. Cette méthode préconise l'utilisation d'une fréquence intermédiaire (IF) pour l'utilisation de notre modèle, justifiant ainsi la sélection de notre choix d'une fréquence porteuse à 2,046 MHz.

## 3.6 Conclusion

Dans ce chapitre, nous avons étudié l'aspect théorique de notre récepteur GPS numérique. Il est bien important de comprendre les processus et les algorithmes inclus à l'intérieur de la chaîne de communication GPS. Ce sont ces processus que nous devons modifier afin de passer à l'implémentation en temps réel de notre récepteur. De plus, l'étude doit être faite afin de comprendre les impacts des paramètres puisque ces derniers auront un rôle crucial à jouer dans la prochaine partie du projet. Nous avons pu aussi voir les différentes limites, que ce soit au niveau des filtres ou des discriminateurs par exemple, du modèle, ce qui nous permettra de choisir les processus à implémenter

dans le matériel de notre plate-forme de développement. La validation du modèle est une partie importante de la méthodologie et permet de repérer les erreurs avant la suite du processus d'implémentation.

Maintenant que la validation par simulation logicielle de notre système est effectuée, il nous est possible de suivre la méthodologie et de passer aux étapes suivantes, présentées dans le prochain chapitre.

# CHAPITRE 4

# ANALYSE DE LA QUANTIFICATION SOUS SIMULINK

De tous les éléments autour de lui, le seul élément que l'humain ne peut modifier à sa guise, est le temps. À travers les différentes évolutions technologiques, le défi pour appliquer les technologies a toujours été l'aspect temporel. Dans les simulations, l'aspect temps, au départ, est plutôt mis de coté. Les travaux portent plus sur les performances et l'efficacité du modèle. Mais, pour mettre en marché un produit, il est important de tenir compte de l'influence du temps sur le processus. Le temps réel, par définition, c'est l'étude d'un matériel ou d'un logiciel lorsque ce dernier est soumis à une contrainte de temps, c'est-à-dire que l'opération a des limites de temps pour ses sous-opérations ou processus**[21]**. Même si l'on désire souvent une réponse rapide ou un système performant, il n'est pas essentiel dans certains cas.

Il y a deux catégories de système en temps réel : des systèmes dits critiques et des systèmes dits non critiques. Un système non critique a peu de contraintes temporelles ou ces dernières sont relativement larges. Par exemple, un système de gestion des places dans un avion peut réagir à l'intérieur de quelques secondes sans toutefois causer un problème critique dans le système de réservation des places. Dans le cas des systèmes critiques, il est impératif que le processus réponde rapidement à la donnée entrante. Dans cette catégorie, on peut inclure, par exemple, le cas des contrôleurs d'une chaîne de montage dans une usine ou un électrocardiogramme.

Dans notre cas, il est essentiel que notre récepteur réponde dans les plus brefs délais à cause du mouvement des satellites et du mobile. Le récepteur en mouvement doit connaître sa position de manière instantanée, sinon l'information s'avère peu utile et même désuète, ce qui peut entraîner des erreurs graves (dans le cas, par exemple, de l'atterrissage d'un avion). Il s'agit donc d'un système critique.

Pour assurer le passage efficace entre la partie logicielle et la partie matérielle, plusieurs méthodes ont été développées. La toute première, et la plus connue, est la technique MASCOT. Les concepteurs de cette technique ont été à l'encontre des méthodologies existantes à cette époque. Les méthodologies existantes se développaient autour des aspects fonctionnels du projet, en développant un langage rigoureux et structuré. À l'inverse, le principe MASCOT met l'emphase sur l'aspect architectural du logiciel. Ainsi, les efforts se concentraient sur le contrôle en temps réel des processus et des interfaces entre les processus du modèle.

## 4.1 Principes de base pour l'implémentation en temps réel avec Matlab/Simulink

La méthodologie développée dans le cadre de notre projet se base, non seulement sur les méthodes développées pour le passage en temps réel tel MASCOT, mais aussi sur le concept SDR tel qu'expliqué à la section 2.1. Au CHAPITRE 3, l'architecture du récepteur numérique a été présentée, tel que la méthodologie MASCOT le propose. Pour le passage en temps réel, nous devons tenir compte de certains paramètres, tout comme dans le cas des simulations. La fréquence d'échantillonnage reste le paramètre de base qui contrôle la majorité des fonctions. Dans les librairies de Xilinx, à l'intérieur de Simulink, les blocs définis par les programmeurs ont tous des paramètres communs. Ces paramètres sont :

1. Entrée de remise à zéro
2. Port de contrôle
3. Utilisation des variables « *double* »
4. Fréquence d'échantillonnage

Les deux premiers paramètres sont utilisés pour la commande des blocs, à savoir quand on veut les mettre actifs ou inactifs et avec quelle valeur de départ. L'utilisation des variables de type *double* peut se faire seulement lors de simulation à l'intérieur de Simulink seulement.

À l'inverse des blocs Simulink de base, les blocs des librairies Xilinx discrétisent automatiquement les variables arrivant dans leurs ports d'entrées. Dans le modèle Simulink, des bloqueurs d'ordre 0 avaient dû être incorporés dans le modèle afin d'assurer la discrétisation des variables. Ainsi, la fréquence d'échantillonnage devient un paramètre global. Tous les processus dépendront de cette fréquence, et il nous sera impossible de faire un échantillonnage asynchrone. Aussi, puisque tout dépend de ce paramètre, nos variables paramétrables tels le temps d'intégration des boucles, la commande pour la fréquence centrale des NCO et les filtres numériques sont tous en fonction de cette variable.

## 4.2 Analyse de l'effet de quantification entre les modèles (Simulink et quantifié)

Le passage en temps réel implique de nombreux changements. Un des changements les plus importants est la numérisation des signaux utilisés. Puisque les signaux sont analogiques et que les signaux à traiter seront numérisés par les convertisseurs numériques analogiques, notre modèle doit être capable de traiter des signaux numériques. Ainsi, dans notre modèle, les données traitées doivent être définies en binaire. Lors de la simulation en Simulink, les variables étaient de type *double*, c'est-à-dire des nombres réels. Cependant, dû au nombre limité de bits pour une variable, une erreur d'approximation intervient dans la quantification de notre signal. La **Figure 44** nous montre un exemple de quantification et les paramètres s'y rattachant.

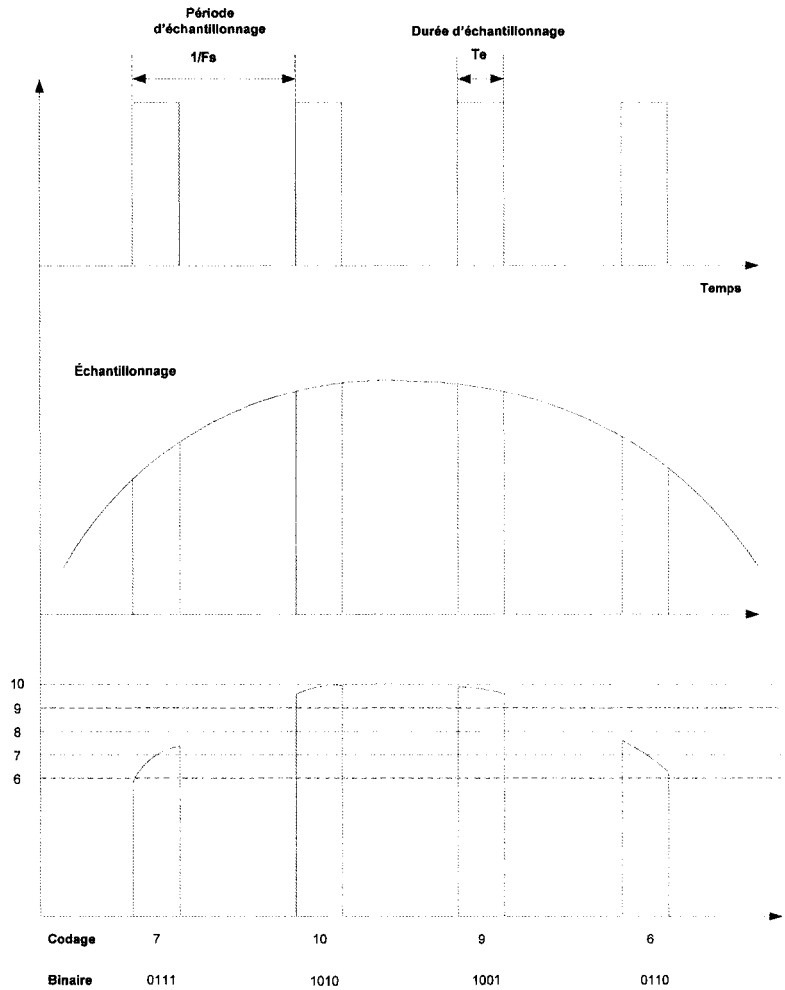



Figure 44 Exemple de quantification d'un signal sinusoïdal

Le fait de quantifier notre signal introduit une erreur d'approximation, appelé erreur de quantification. Lorsqu'on quantifie un signal, on doit prendre un certain nombre d'échantillons, et si la durée de l'échantillonnage ($T_e$ sur la **Figure 44**) n'est pas assez courte, il se peut que le signal change de valeur fréquemment à l'intérieur de cet intervalle. Ce temps de quantification est défini dans les blocs de conversion des variables (*Gateway In* de la libraire de Xilinx) et il est important de choisir ce temps le plus court possible. La période qui correspond au temps le plus court de notre modèle est la période d'échantillonnage, soit l'inverse de la fréquence $F_s$. De cette façon, nous

arrivons à avoir seulement une approximation du signal. Cependant, il est possible de diminuer cette erreur en introduisant un nombre plus élevé de bits. Le tableau suivant nous montre un exemple de quantification et l'erreur engendrée en fonction du nombre de bits de quantification.

Tableau VII

Exemple d'erreur de quantification

| **Nombre de bits de quantification** | **Valeur à convertir** | **Valeur en binaire** | **Valeur réelle de la quantification** | **Erreur de quantification** |
|---|---|---|---|---|
| 3 bits | 1.65 | 1.11 | 1.75 | 0.10 |
| 4 bits | 1.65 | 1.101 | 1.62 | 0.03 |

Cette forme d'erreur de quantification survient sur les nombres dont la partie fractionnaire est différente de 0. En analysant les différentes valeurs possibles de nos signaux à l'intérieur de notre récepteur numérique, seule la donnée du satellite GPS a une composante fractionnaire nulle. Les sinusoïdes, les réponses des discriminateurs et les sorties des filtres auront une composante fractionnaire non nulle, impliquant obligatoirement une erreur de quantification.

Pour déterminer le nombre minimum de bits nécessaire, nous devons regarder les limites de nos variables utilisées dans les calculs du récepteur numérique.

Tableau VIII

Valeurs limites des signaux de commandes

| Signal | Limite inférieure | Limite supérieure |
|---|---|---|
| Récupération de la porteuse | -1 | 1 |
| Récupération du code | -1 | 1 |
| Intégrateur | -1 | 1 |
| Discriminateur PLL (arctan) | $-\pi/2$ | $\pi/2$ |
| Discriminateur DLL (AMR) | -1 | 1 |

Les valeurs limites des discriminateurs sont démontrées sur les **Figure 33** et **Figure 40**, selon les discriminateurs. Malgré leurs différences d'algorithmes, les valeurs limites des discriminateurs sont calculables. À la sortie des filtres, les valeurs limites dépendent directement de la valeur de la bande de bruit équivalente choisie, tel que montré au **Tableau V**. Cependant, à l'intérieur des processus, certaines valeurs peuvent dépasser ces limites, mais comme elles sont des variables locales, il sera toujours possible de minimiser l'impact.

Le nombre de bits est aussi important dans un autre aspect de notre récepteur numérique, la précision. Le dernier bit de quantification constitue le plus petit pas de discrétisation possible pour notre modèle, déterminant la résolution de notre système. Comme nous avons des nombres fractionnaires, notre résolution est de $2^{-m}$ où m est le nombre de bits après la virgule. La détermination de $m$ permettra non seulement de réduire les erreurs

de quantification mais aussi de déterminer la plus petite erreur de phase que le modèle pourra calculer. L'importance de ce paramètre est démontrée dans les **Figure 45** et 47.

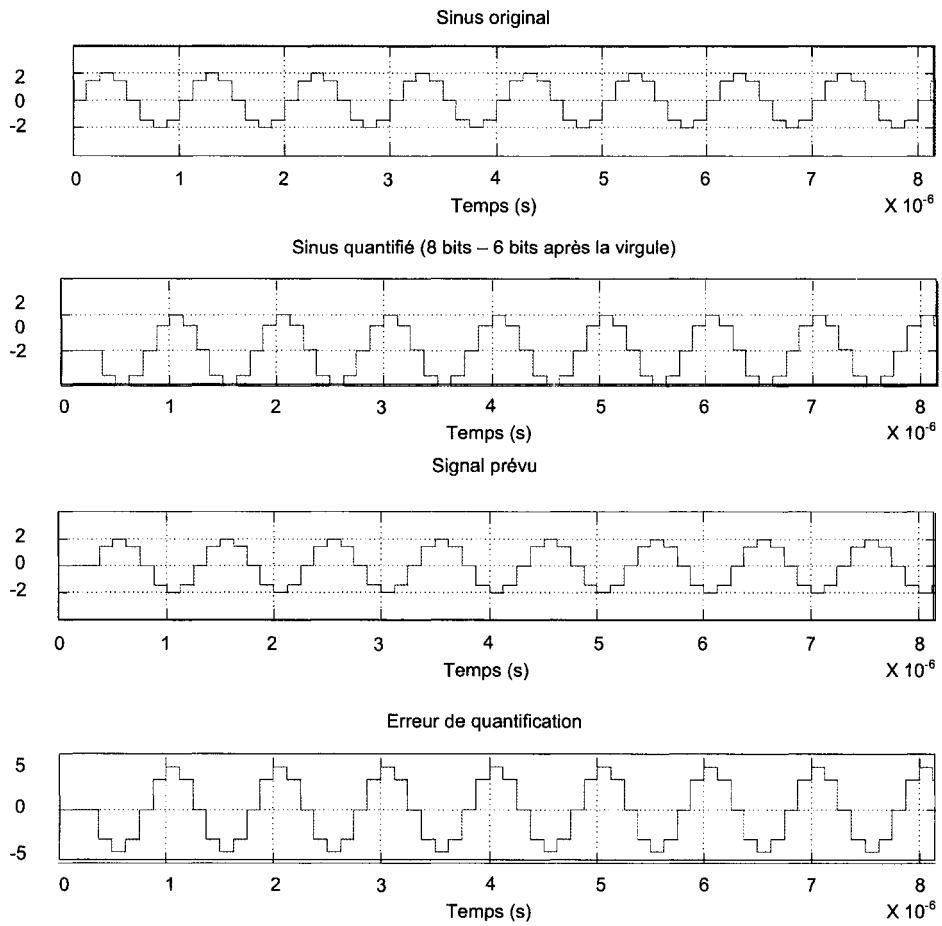



Figure 45 Erreur de quantification avec une quantification de 6 bits

La **Figure 45** permet de montrer l'erreur de quantification lorsque le nombre de bit après la virgule est trop grand par rapport au nombre de bits total, soit 8 bits. Comme nous n'avons que 2 bits avant la virgule dont un est le bit de signe, la précision de notre système est de 0,015625. Cependant, la valeur maximale que nous pouvons atteindre est 1,984375, et non 2. Cette petite erreur, jumelée avec le déphasage engendrée par le bit de signe, fait en sorte que nous avons une erreur de quantification qui s'approche de 5

par moment. Une telle erreur est trop importante pour notre récepteur numérique. Pour diminuer l'erreur de quantification, le nombre de bits après la virgule a été réduit en gardant le même nombre de bits total. Les résultats sont présentés à la **Figure 46**.

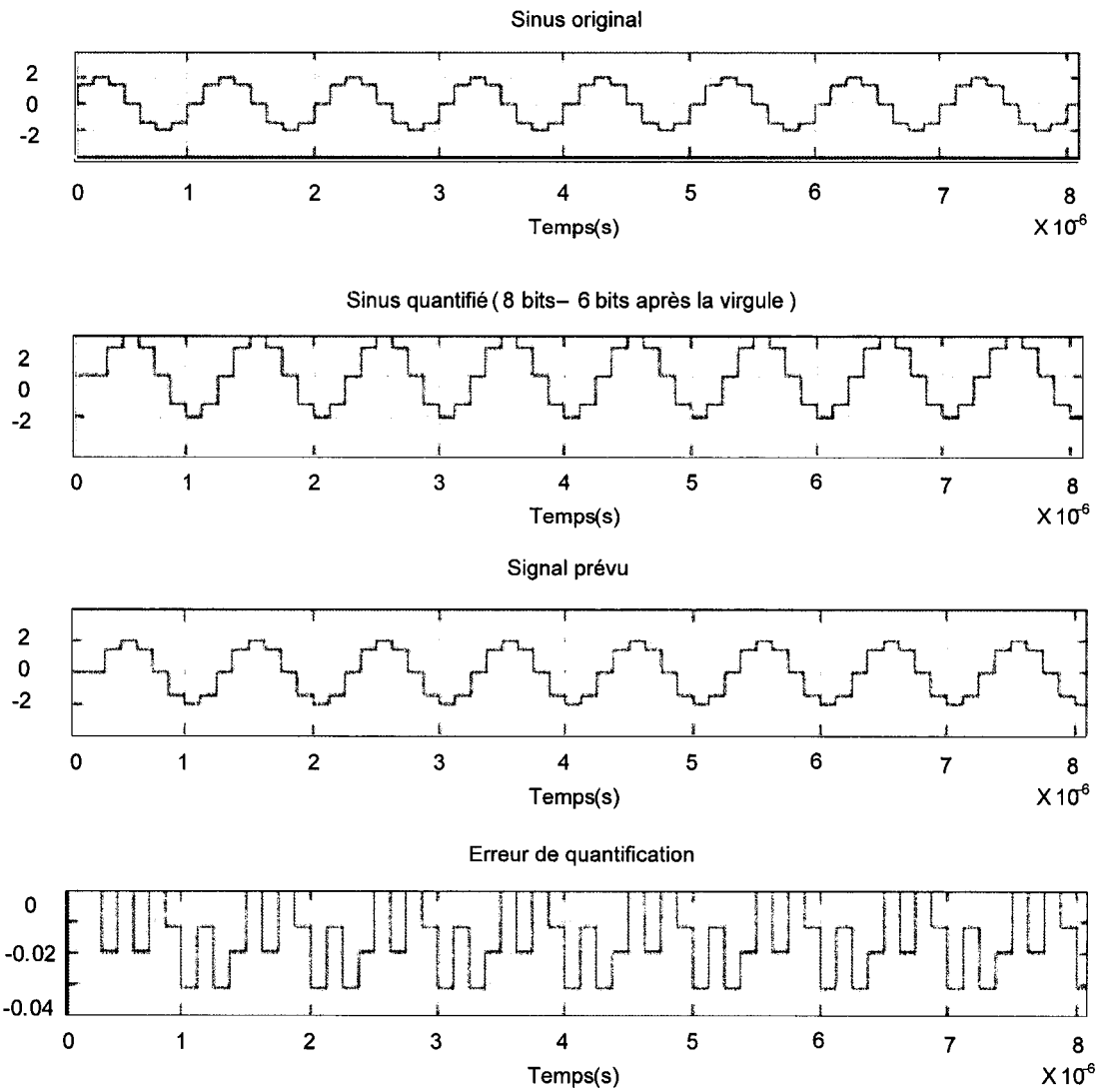



Figure 46 Exemple de quantification avec 4 bits fractionnaires

Avec seulement 4 bits de précision, la résolution est de 0,0625. Par contre, l'erreur de quantification a diminué à 0.03, (au maximum). Une amélioration grandement importante pour la précision de notre système. Si l'on se réfère au **Tableau VIII**, la

valeur maximale est de $\pi/2$, ce qui demande un minimum de 3 bits pour définir la partie entière de notre signal.

Au niveau de la résolution, le Tableau IX

**Niveau de précision en fonction du nombre de bits** montre le niveau de précision selon le nombre de bits de la partie fractionnaire.

Tableau IX

Niveau de précision en fonction du nombre de bits

| **Nombre de bits** | **Précision minimale** |
|---|---|
| 1 | 0,5 |
| 2 | 0,25 |
| 3 | 0,125 |
| 4 | 0,0625 |
| 5 | 0,03125 |
| 6 | 0,015625 |

De par la conception du NCO et des tables de correspondance de génération des sinusoïdes, le nombre total de bits requis est de 9. En étant déjà assuré d'avoir au moins 3 bits pour la partie entière de nos variables, nous pouvons aller jusqu'à 6 bits de précision au niveau de la partie fractionnaire. Cependant, en analysant le **Tableau IX**, le gain de précision après le quatrième bit n'est pas significatif. Une précision de plus du centième n'est pas nécessaire, par contre, celle du dixième est essentielle.

Ces conditions d'utilisations (nombre de bits et précision) de notre récepteur indiquent que notre récepteur doit avoir, au minimum 3 bits pour la partie réelle et 4 pour la partie fractionnaire. Ainsi, pour optimiser l'utilisation des ressources à notre disposition dans le FPGA, le choix de 8 bits (dont quatre pour la partie fractionnaire) s'impose de lui-même.

## 4.3 Modification du modèle Simulink pour le passage en temps réel

Au CHAPITRE 3, nous avons démontré et expliqué le fonctionnement du simulateur d'une chaîne de communication GPS. Les différentes caractéristiques du modèle doivent maintenant être adaptées afin de le modifier pour effectuer le passage en temps réel. Si l'on reprend notre cheminement montré à la **Figure 13**, le chapitre antérieur établit la base de notre graphique. En comparant le tout à une figure géométrique simple, nous avons établi les fondations de la pyramide. Maintenant, il faut modifier notre modèle Simulink afin de pouvoir générer les bons codes. Les outils utilisés ont été décrits dans le chapitre 2 du présent ouvrage, mais seront expliqués dans le détail à la section 4.4 .

En tenant compte des différentes caractéristiques du modèle ainsi que du matériel utilisé, la première étape est de séparer notre modèle en divers processus et d'analyser ces processus afin de déterminer par quel matériel ils seront exécutés. Il est à noter, qu'à partir de cette section, la partie représentant le canal n'est pas étudié. Le projet a pour but de faire un récepteur numérique, et, dans cette situation, la modélisation du canal n'est pas nécessaire. Le canal simulé génère des interférences et nous permet de tester la robustesse de notre récepteur. Cette partie pourra être faite à l'intérieur de l'ordinateur, ce qui nous permettra de réserver le maximum de ressources disponibles pour nos processus.

Pour revenir à notre modèle, nous pouvons le séparer de la façon qui suit :

Tableau X

Processus à l'intérieur du récepteur numérique

| Partie du modèle | Processus |
|---|---|
| ***Source*** | Générateur de fréquence |
| | Générateur de code |
| | Générateur de donnée |
| | Modulation en fréquence |
| ***Récepteur*** | Boucle PLL |
| | Boucle DLL |
| | NCO |
| | Filtre |
| | Filtre passe-bas/Intégrateur |
| | Discriminateur PLL |
| | Discriminateur DLL |

Au niveau de la source, on remarque, outre la multiplication en fréquence, que nous n'avons que des oscillateurs. Une caractéristique de ces oscillateurs est qu'ils ne sont pas tous indépendants. Premièrement, la fréquence des données est de 50 Hz, qui est la fréquence de base. Les données sont ensuite modulées par le code qui est à 1,023 MHz. Le message codé est enfin modulé par la porteuse de 1575,42 MHz. Tous ces signaux (données, code et porteuse) sont synchronisés, ce qui nous permet de limiter au minimum les oscillateurs. En utilisant un générateur de fréquence de 1575,42 MHz, nous pouvons, à l'aide d'un diviseur de fréquence, générer la fréquence du code. Comme ils sont synchronisés, la cadence du code est un dénominateur de la fréquence du code. Ainsi, le diviseur de fréquence aura comme valeur 1575,42/1,023 donc 1540. Pour notre modèle, nous n'utiliserons pas la fréquence réelle de la porteuse, tel qu'expliqué à la section 3.5, mais une valeur intermédiaire qui respecte la synchronisation des signaux.

Cette particularité du signal GPS nous permet donc de générer un seul oscillateur, ce qui, nous le verrons plus tard, diminue la quantité de ressources utilisées dans notre matériel, laissant ainsi plus de ressources disponibles au traitement du signal.

Au niveau du récepteur, contrairement à la source, les processus ont des caractéristiques relativement différentes.

Tableau XI

Caractéristiques des processus du récepteur

| **Processus** | **Caractéristiques** |
|---|---|
| Boucle PLL | Sensible au bruit |
| Boucle DLL | Résistante au bruit |
| NCO | Grande résolution, oscillateur local |
| Filtre | Calcul mathématique, fréquence faible |
| Filtre passe-bas/Intégrateur | Calcul mathématique, temps d'intégration court |
| Discriminateur PLL | Algorithme mathématique complexe |
| Discriminateur DLL | Algorithme mathématique complexe |

Le **Tableau XI** nous résume les caractéristiques des processus du récepteur GPS. La grande diversité des processus fait en sorte que nous ne pouvons, comme dans le cas de la source, réutiliser un processus. Pour ajouter à cette difficulté, à cause des caractéristiques différentes, on ne peut même pas utiliser le même matériel pour tout notre récepteur, ce qui augmente la complexité du travail. Si l'on se réfère au CHAPITRE 2, les caractéristiques des DSP et FPGA nous mettent sur la piste pour séparer notre modèle. Ainsi, les processus qui demandent un calcul mathématique complexe seront modifiés pour être traités dans le DSP. Dans le même ordre d'idée, les

algorithmes demandant une fréquence élevée de traitement seront transformés pour le FPGA. L'optimisation des ressources étant un élément important, nous allons devoir surveiller l'implémentation afin de ne pas surcharger un des processeurs.

Cette séparation de notre modèle implique aussi un changement et une difficulté accrue au niveau des communications. L'échange de données devra être rapide afin d'arriver à récupérer le signal GPS. Cet élément est important d'autant plus qu'il s'agit d'interaction à l'intérieur même d'une boucle. Le bus de données choisi devra fonctionner à une vitesse élevée. Comme le récepteur se compose de deux boucles, le temps alloué à l'échange de données augmentera le temps de récupération de la porteuse ou du code, diminuant l'efficacité de nos boucles.

Tableau XII

Répartition des processus du récepteur numérique

| **Processeur** | **Processus** |
|---|---|
| FPGA | Boucle PLL |
| | Boucle DLL |
| | NCO |
| | Filtre passe-bas/Intégrateur |
| DSP | Filtre |
| | Discriminateur PLL |
| | Discriminateur DLL |

Tel qu'indiqué au **Tableau XII**, le FPGA prendra en charge les processus dit rapides, tandis que le DSP prendra en charge les processus dit complexes. Les filtres sont laissés au DSP en raison du minimum d'interaction entre les différents processeurs. En raison

de sa fréquence d'horloge rapide, le FPGA est en mesure d'effectuer des opérations non complexes beaucoup plus rapidement que le DSP. Cet avantage est capital pour les récupérations de porteuse et de code. Par exemple, le désétalement spectral doit être fait rapidement afin que les échantillons de données soient envoyés au filtre passe-bas, remplacé ici par un intégrateur.

Les processus du modèle étant repartis à travers les processeurs cibles (DSP ou FPGA), nous pouvons poursuivre à travers notre implémentation pas-à-pas. Après la simulation de notre modèle, il faut le modifier avec les librairies correspondantes incluses dans les logiciels de Xilinx et de Sundance. Le développement des librairies de transition à l'intérieur de Matlab/Simulink par ces entreprises facilite le passage en temps réel car elle respecte les caractéristiques des processeurs.

Pour le DSP, Sundance en collaboration avec Texas Instruments, a développé une implémentation contenant une librairie (SMT6050) et un logiciel (Code Composer Studio). Code Composer studio permet non seulement de transformer un fichier en code C++ mais aussi de se connecter et de configurer le processeur. De plus, Mathworks offre un compilateur C++ avec Matlab directement. Ces deux aspects combinés ensemble font que les changements pour les processus allant dans le DSP sont minimes. La **Figure 47** nous montre, comme exemple, la source de notre modèle.

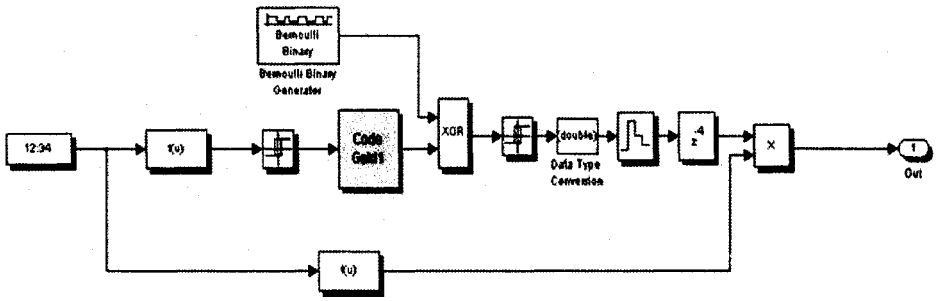



Figure 47 Source GPS en Simulink

On remarque certaines modifications par rapport à la source présentée au chapitre 3. Tout d'abord, il n'y a qu'un seul oscillateur pour générer les fréquences de la porteuse et du code de Gold. À l'aide d'une horloge simple et d'un algorithme mathématique, la génération des fréquences se fait de façon plus simple et utilise moins de ressources. Les expressions mathématiques utilisées pour générer les fréquences sont :

1. Pour la porteuse :

$$\frac{\sqrt{2}}{2} * sin(2 * Fs * u(1)) \qquad \textbf{(4.1)}$$

où fs est la fréquence de la porteuse, et u(1) la valeur de l'horloge d'entrée.

2. Pour le Code de gold :

$$\frac{\sqrt{2}}{2} * sin(2\pi * Fc * u(1)) \qquad \textbf{(4.2)}$$

où fc est la fréquence du générateur de code de Gold, et u(1) la valeur de l'horloge d'entrée.

Comparativement au système de base, qui comprenait au moins 4 oscillateurs différents afin de simuler des perturbations, ce système permet une utilisation maximale des ressources avec des algorithmes mathématiques simples. Ainsi pour simuler l'effet Doppler, nous n'avons qu'à rajouter une composante de phase dans la génération des signaux. Nous pouvons simuler ainsi deux perturbations : perturbation sur la fréquence et perturbation sur la phase. En se basant sur l'expression générale d'une onde radio, nous avons :

$$\cos(\omega * t + \theta) \tag{4.3}$$

où θ est la phase du signal et ω, la fréquence. Ainsi, l'expression mathématique de la porteuse devient :

$$\frac{\sqrt{2}}{2} * sin((2\pi * Fs + \omega_p) * u(1) + \theta_p) \tag{4.4}$$

où $\omega_p$ est la déviation en fréquence de la porteuse et $\theta_p$ est la déviation en phase de la porteuse. Comme la fréquence $F_s$ est fixe, les paramètres variables à définir sont les déviations du signal. Donc, au lieu de générer 3 fréquences, ce qui occupe beaucoup de place dans le processeur, nous n'avons qu'à passer les valeurs des perturbations.

À travers les libraires de Sundance, la SMT6050 est celle qui s'assure du lien entre Code Composer Studio, le DSP ainsi que le modèle Simulink. En effet, cette librairie est divisée en quelques parties, dont communication, mathématique, etc... Comme Matlab possède un convertisseur en C++ d'un modèle Simulink, les blocs internes Simulink n'ont pas besoin d'être modifiés ou changés. Nous pouvons générer notre code à partir du schéma du modèle lui-même.

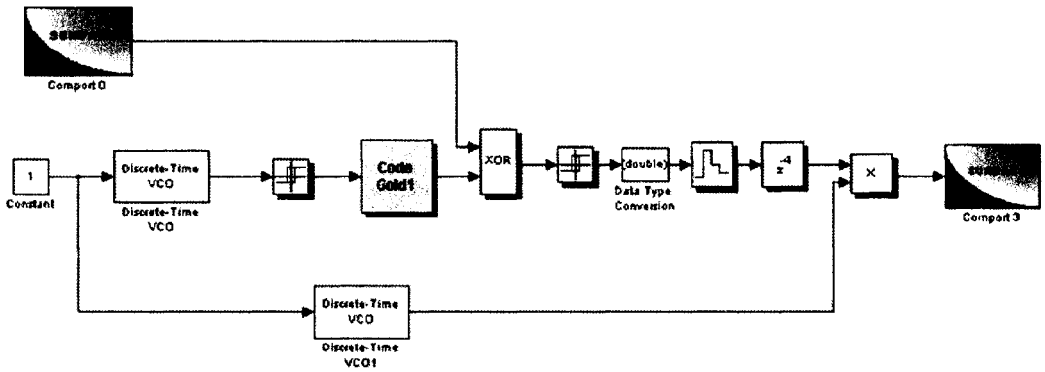



Figure 48 Source GPS modifié pour le DSP

La **Figure 48** nous montre les changements apportés à la source du signal GPS. Puisque nous simulions une perturbation, nous avons décidé de laisser deux(2) oscillateurs (VCO) locaux pour générer les fréquences de Gold et de la porteuse. À la différence du modèle de base, les données de navigation, simulées normalement par un générateur de Bernoulli, sont un paramètre que nous simulerons à l'intérieur de l'ordinateur. Pour ce faire, nous devons indiquer au processeur TMS320C6416 qu'il doit vérifier ses ports de communication afin d'y recevoir une donnée. De plus, avec les libraires Sundance, nous avons le choix du port sur lequel la donnée sera envoyée. Il en est de même pour le signal de la source GPS. Le bloc *Comport* nous permet d'effectuer ses tâches en précisant la fréquence d'échantillonnage que nous voulons. Dans notre modèle, nous avons soumis l'entrée sur le port 0 du processeur et la sortie sur le port 3. Les résultats de cette implémentation seront discutés à la section 5.2.

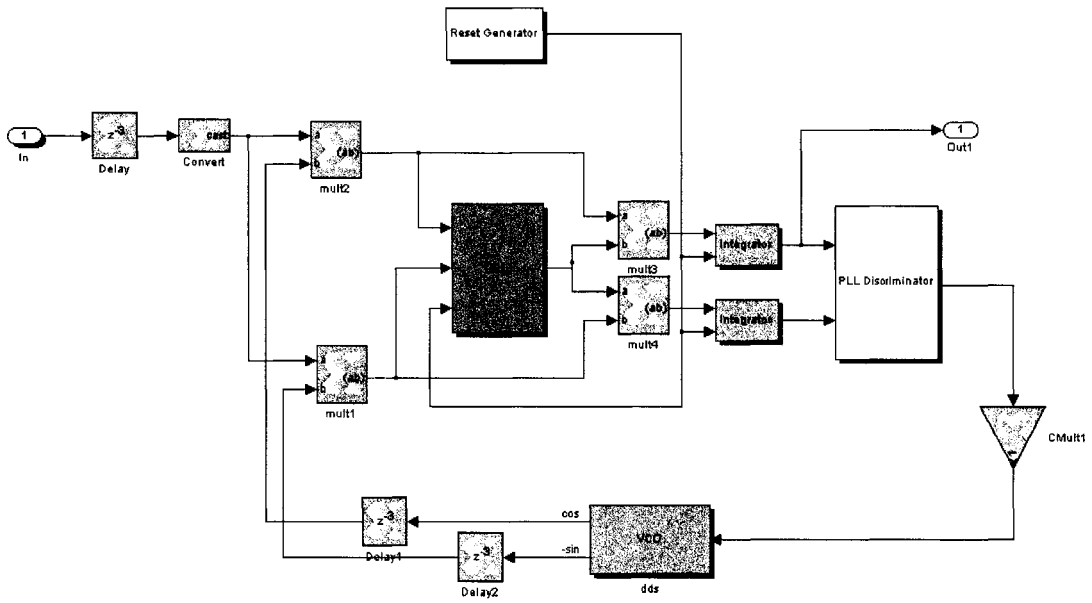



Figure 49 Schéma du récepteur préliminaire pour l'implémentation dans le FPGA

Au niveau du récepteur, comme il est séparé en plusieurs processus, nous avons plusieurs changements à faire. Si l'on se réfère au **Tableau X**, les processus les plus simples à modifier seront les multiplicateurs en bande de base. La multiplication de deux signaux se fait relativement bien en binaire. La **Figure 50** nous montre le désétalement spectral de notre signal en Simulink. Il s'agit tout simplement d'une multiplication entre le code de Gold et le signal démodulé avant de transmettre la voie I et la voie Q dans le filtre passe-bas.

Cette partie de notre récepteur se trouvera dans le FPGA tel que montré au **Tableau XII**. Dans la libraire que XIlinx ont conçu pour les FPGA, il existe plusieurs blocs pouvant faire le désétalement spectral. Par contre, comme nous sommes soucieux de l'optimisation des ressources, le choix d'un multiplicateur simple avec un délai de 2 échantillons reste un choix logique. Ainsi, la démodulation sous Simulink, montré à la

**Figure 50** devient le schéma de la **Figure 51** une fois modifié pour le temps réel. L'ajout d'un bloc *Gateway In* et de deux blocs *Gateway Out* est nécessaire pour la numérisation du signal. Ces deux types de blocs sont importants pour la réalisation du modèle en temps réel puisqu'il détermine le code binaire dont nous voulons paramétrer le modèle. Ce sujet, ainsi que ses effets, sera abordé plus loin dans le présent chapitre.

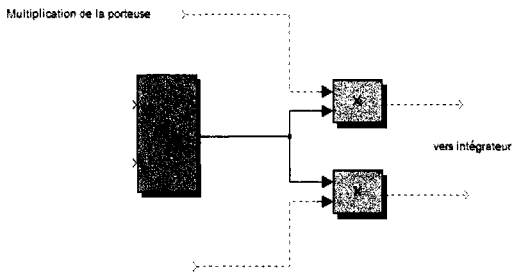



Figure 50 Désétalement spectral du signal en Simulink

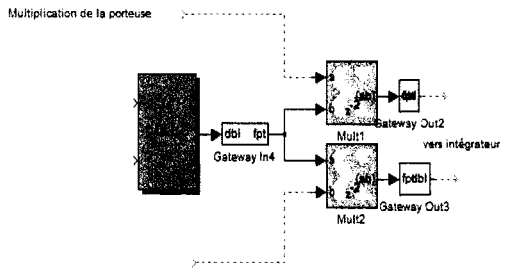



Figure 51 Désétalement spectral modifié pour le temps réel

Ce premier changement nous montre déjà un des effets de la modification pour le passage en temps réel sur notre modèle, la quantité de bloc nécessaire. Si l'on compare les deux figures (**Figure 50** et **Figure 51**), on remarque que pour une opération avec deux blocs Simulink, nous devons mettre cinq blocs des librairies de Xilinx pour le FPGA. Cette particularité nous demande de simplifier au maximum notre système afin d'utiliser le moins de bloc possible, donc le moins de ressources matérielles. Il est bien important de rappeler que nous travaillons avec un nombre limité d'outils, que ce soit de portes logiques, de tables de correspondance, etc. Aussi, un autre aspect démontré par ce changement est le délai qui sera incorporé à cause du bloc multiplicateur. Lorsque nous avons un signal GPS à 1575,42 MHz et deux boucles de Costas pour récupérer la porteuse et le code, l'ajout d'un délai peut causer une désynchronisation importante, ce qui pourrait entraîner un temps plus long dans la récupération des donnés, puisque la boucle mettra plus de temps à s'arrimer sur le signal GPS. Un bloc de délai devra donc

être ajouté au signal pour que la porteuse et le code soient récupérés plus rapidement. Ce bloc devra donc tenir compte du nombre de délais ajoutés lors de la modification de notre système. Considérons la génération de la réplique de la porteuse, il est important qu'elle soit en phase, sur la voie I, par rapport au signal GPS. Or, en incorporant des délais, la réplique sera automatiquement retardée, c'est-à-dire déphasée. Le système prendra plus de temps avant de récupérer les données GPS, diminuant ainsi les performances de notre modèle. Cependant, comme la réplique est un signal sinusoïdal, nous pouvons récupérer notre signal à chaque période, ce qui fait que pour une valeur de délai $x$, nous avons la possibilité de récupérer notre signal en phase, ainsi que pour $nx$, où $n$ est un entier. La détermination de $x$ sera expliquée plus loin dans le texte.

Ce petit changement, sur le désétalement spectral, sert à illustrer l'implication du nombre de modifications à apporter afin de bien arrimer le passage entre le temps réel et la simulation. De plus, l'importance de bien optimiser notre modèle est facilement visible par le dernier exemple, puisque nous avons plus que doubler les ressources nécessaires lors de la modification.

En poursuivant la modification de notre récepteur, un cas particulier, le NCO, nous force à effectuer des changements importants dans notre modèle. Le NCO se situe dans les deux boucles, de phase et de code, de notre récepteur et génère la réplique que l'on module avec notre signal pour récupérer, soit la porteuse, soit le code. La théorie, tel que montré dans Kaplan**[1]**, nous annonçait cette modification qui a trait à la sensibilité du NCO. Le générateur de fréquence doit se baser sur une fréquence d'horloge afin de créer le signal qui servira au recouvrement. Mais ce signal doit être numérisé, ce qui en fait, se traduit par la relation suivante :

$$F_{out} = \frac{F_{clk}}{2^L} * S \qquad \textbf{(4.5)}$$

où $F_{out}$ est la fréquence de sortie du NCO, $F_{clk}$ la fréquence de l'horloge, $L$ , le nombre de bits sur la phase et $S$, valeur de l'ajustement de la phase. On remarque que le nombre de bits a une influence importante sur la fréquence de sortie. Ce nombre détermine en fait le pas avec lequel les changements peuvent être effectués. Par exemple, avec une valeur L de 2, le pas sera du quart de la fréquence d'horloge. Par contre, pour une valeur de L plus grande, soit 8, le pas seront de 1/256ième de la fréquence d'horloge, ce qui rend le NCO beaucoup plus précis. Mais l'ajout du nombre de bits augmente l'imprécision au niveau de la valeur de $S$, nous devons donc faire un compromis entre ces deux sources d'erreur. Plusieurs possibilités peuvent être envisagées pour améliorer la sensibilité du NCO, mais deux approches retiennent notre attention : soit l'ajustement fin de la phase et le processeur d'arrondi**[30]**.

L'ajustement fin de la phase se veut d'utiliser le nombre maximum de bits sur $S$. Cette valeur contient l'erreur sur la phase de la boucle. Ainsi, en augmentant la précision de $S$, sans changer $L$, nous diminuons l'erreur dû à $L$ sur la valeur d'erreur. Pour contourner cette problématique, nous pouvons séparer l'équation 4.5 de cette façon :

$$F_{out} = \frac{F_{clk}}{2^L} * \left( S' + \frac{B}{A} \right) \tag{4.6}$$

Les valeurs de A et de B doivent être l'erreur en arrondi. Comme nous voulons une fréquence de sortie, en simulation, pour la porteuse de 2,046 MHz, en se basant sur cette relation, nous devons avoir un nombre de bit égal à 11. Cependant, notre signal $S$ a seulement 8 bits. Nous devons donc le séparer afin de diminuer l'erreur.

Une première hypothèse posée dans notre modèle est que nous avons 8 bits, soit 4 bits après la virgule. Pour les nombres binaires, la valeur des chiffres après la virgule $\frac{1}{2^n}$ où $n$ est la position du chiffre. Donc, avec 4 bits comme partie fractionnaire, nous avons

une précision de $\frac{1}{16}$ sur la valeur de *S.*, impliquant un pas minimal de 0,5435 MHz. Le dénominateur, la valeur de A, sera toujours de 16. Le bloc *slice* nous permet d'isoler les bits d'un nombre binaire. Ainsi nous pouvons donc raffiner la valeur de *S* en sachant que le dénominateur aura une valeur constante à 4 bits. Le schéma de la **Figure 63** nous montre l'arrangement du modèle avec les blocs *slice* et des opérateurs logiques utilisés.

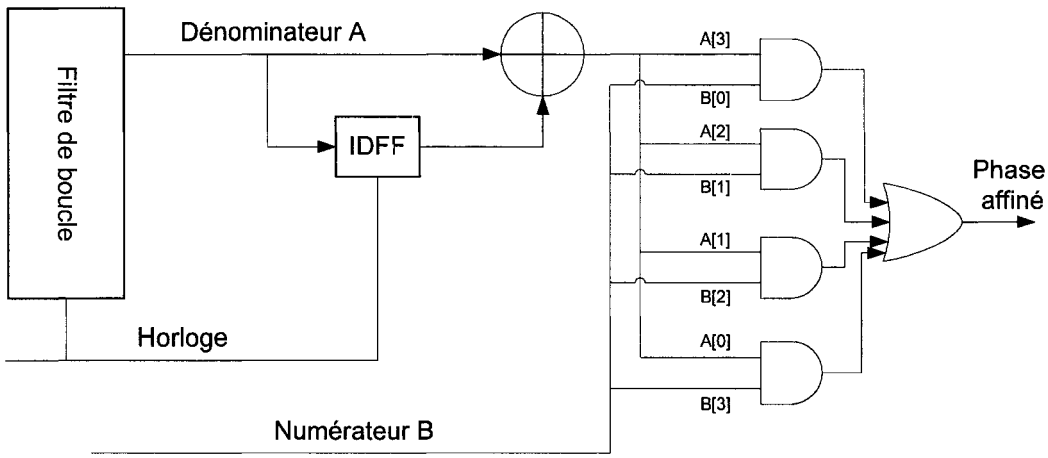



Figure 52 Principe de l'ajustement fin de la phase

Le schéma de la **Figure 53** nous montre le processus qui nous permettra d'arrondir correctement la valeur de *S.* Ce principe permet de limiter la perte des bits moins significatifs sur la précision du NCO. Le principe d'arrondissement utilisé dans notre système est assez simple. La première étape est de séparer le nombre en 2, soit la partie entière et la partie décimale. La partie décimale est alors envoyée dans un registre. La prochaine valeur décimale est additionnée à la précédente. Cette valeur est donc ajoutée à la partie entière. L'équation suivante nous démontre ce calcul :

$$y_M(n) = \left[ x_L(n) + \sum_{k=0}^{n} x_{L-M}(k) \right] \qquad (4.7)$$

Cette formule générale peut être adapté à notre système avec les termes suivant, basé sur les équations 4.5 et 4.6.

$$y_{sortie}(n) = \left[ S(n) + \sum_{k=0}^{n} LSB_{S(n)}(k) \right] \tag{4.8}$$

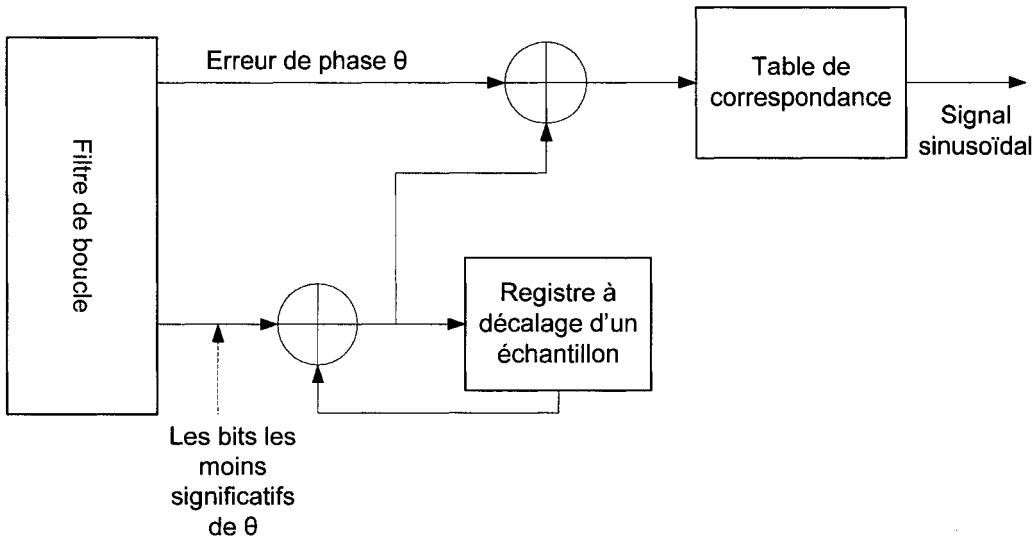



Figure 53 Principe d'arrondissement

Le résultat est donc une valeur beaucoup plus près de la réalité, donc une réponse plus rapide de nos boucles de phases et de code. À l'intérieur du FPGA, le signal sinusoïdal est calculé à partir de LUT (*look-up tables*) ou tables de correspondance, ce qui signifie qu'il y a un pas de discrétisation à respecter. Ce facteur a une influence considérable sur le temps de réaction de notre boucle. À la sortie des discriminateurs, le calcul indique l'erreur de la fréquence de recouvrement, et non la valeur de la fréquence, le NCO doit donc compenser cette erreur avec une augmentation ou diminution de la fréquence de sortie. Donc, par rapport aux tables de correspondance, l'erreur doit être additionnée à la valeur centrale de notre NCO. Le pas de discrétisation intervient ici dans notre modèle. Plus il est petit, plus le raffinement de la fréquence de sortie sera ajusté. Cependant, le nombre limité de ressource fait en sorte qu'un compromis devra être fait entre la

précision du NCO et le temps de réponse de nos boucles. Les tables de correspondance prennent un espace fixe à l'intérieur du FPGA et le travail doit être fait à l'intérieur de ces paramètres.

## 4.4 Méthodologie de génération des codes sources temps réel

Les modifications et l'analyse de la quantification permettent de prédire la réponse de notre récepteur numérique à l'intérieur des parties matérielles. Avant l'implémentation de notre récepteur numérique dans le DSP et FPGA, l'application doit être encodé pour les exécuter dans les processeurs.

### 4.4.1 Génération du code C++ pour le DSP

Les différentes ententes entre les compagnies font en sorte que la génération d'un code pour l'implémentation à l'intérieur d'un processeur soit fait de façon le plus conviviale possible. Le DSP choisi, dont les caractéristiques sont à l'ANNEXE 2, a été conçu par Texas Instrument et doit être programmé en assembleur. Cependant, avant de l'implémenter en assembleur, nous devons générer le modèle en C++ et Texas Instruments a développé le logiciel Code Composer Studio qui permet de programmer, à partir d'un code C++, le DSP.

Tel qu'expliqué à la section 4.3, les modifications pour le DSP sont minimes. Cependant, il reste des paramètres à imposer qui auront une influence sur le code. Matlab et Microsoft ont travaillé ensemble afin d'inclure un simulateur à l'intérieur de Matlab, ce qui permet de générer un code C++ directement à l'intérieur de Matlab. Le DSP doit avoir une fréquence d'opération plus élevée que notre fréquence d'échantillonnage. Le fonctionnement du DSP, à l'inverse du FPGA, ne dépend pas d'une horloge interne, mais plutôt de la longueur des instructions. Le langage assembleur, qui sera le langage avec lequel sera programmé le DSP, est construit

d'instruction simple et à chaque pas d'opération, une instruction est faite. Ainsi, c'est le nombre d'instruction pour traiter une donnée qui détermine le temps de traitement de cette donnée, représenté par la relation suivante :

$$t_{traitement} = nb_{opération} * F_{opération} \tag{4.9}$$

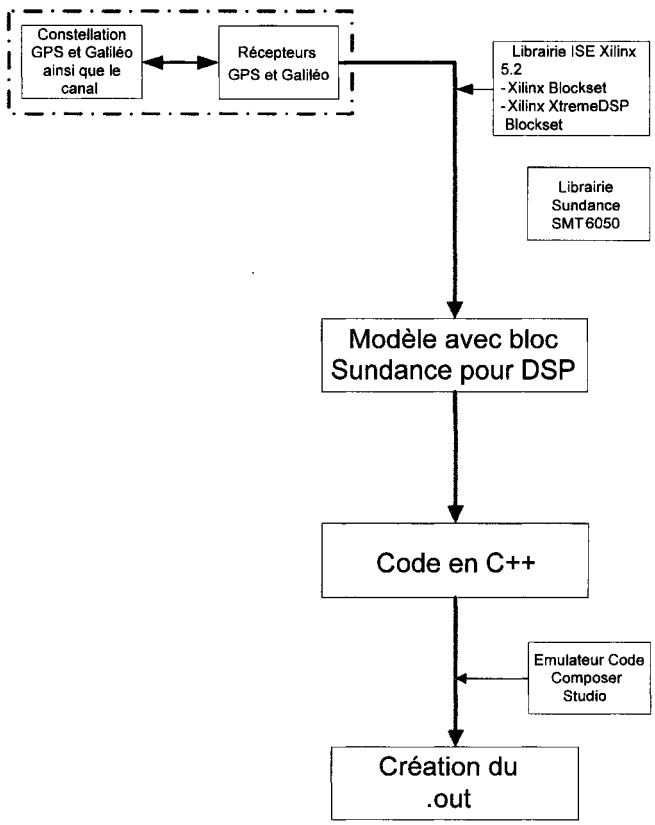



Figure 54 Méthodologie de génération du code pour le DSP

La **Figure 54** montre les étapes pour arriver à générer le code C++ pour le logiciel Code Composer Studio. Tout d'abord, tel que montré précédemment (**Figure 48** et **Figure 49**), des blocs de communication doivent être introduits pour recevoir ou envoyer les données. Ensuite les générateurs de code C compatible avec Mathworks génèrent le code

C++ et automatiquement le logiciel Code Composer Studio nous affiche le code généré. Il est alors possible d'analyser le code afin de l'optimiser.

### 4.4.2 Génération du code VHDL pour le FPGA

Le VHDL a été conçu pour programmer les différents processeurs utilisés en industrie. Il est la relève du langage assembleur. L'avantage d'utiliser le langage VHDL est qu'il facilite l'incorporation des vecteurs dans les algorithmes. Selon la déclaration des variables faites, le code peut inclure des calculs sur les nombres binaires. En fait, les nombres binaires sont déclarés comme des vecteurs et le langage VHDL permet d'effectué des opérations binaires (tel le déplacement d'un bits vers la gauche ou la multiplication en complément à 2). En considérant que nous utilisons des variables binaires dans notre modèle, le langage VHDL est tout désigné pour l'implémentation dans le FPGA.

Le passage entre le modèle Simulink de notre récepteur GPS numérique et le code VHDL est représenté à la **Figure 55**, qui est un agrandissement d'une partie de la **Figure 13**.

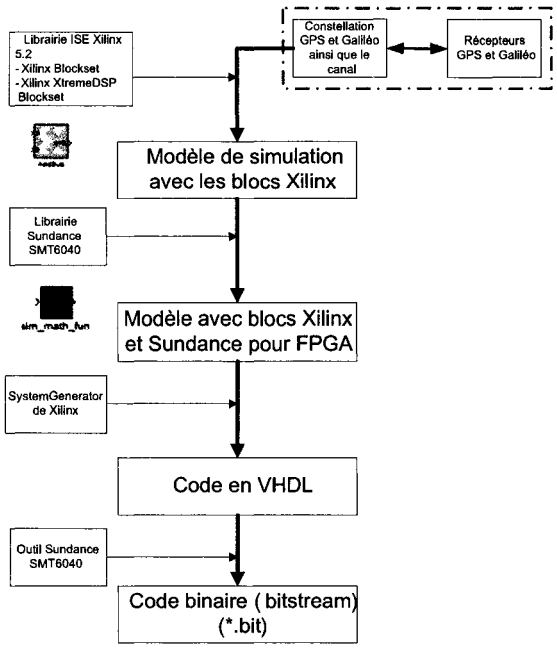



Figure 55 Méthodologie de génération du code binaire

Les changements au récepteur sont nettement plus nombreux que dans le cas du DSP. En effet, Mathworks a développé sont logiciel avec le langage Visual Basic, langage qui s'apparente plus au C++ qu'au VHDL, ce qui explique le nombre plus important de modifications apportées au modèle Simulink.

Tout d'abord, le changement de type de variable entraîne plusieurs modifications. Le fait de travailler en binaire, au lieu des variables de type *double* ou *real*, limite les opérations que nous pouvons effectuer sur les variables, tel qu'expliqué aux sections 4.3 et 4.2. Une fois les modifications de variables et d'algorithmes effectués, les blocs de communications sont ajoutés au modèle pour assurer l'échange des données entre les processeurs. Ces blocs sont inclus dans les librairies des processeurs cibles.

Afin de générer le code en VHDL, le groupe de Xilinx a développé un ajout à sa série ISE, le programme System Generator. Ce logiciel prend les codes des blocs Xilinx qui

compose le modèle à l'intérieur de Simulink, déjà en VHDL, et les intègre à un code global contenant les variables globales et les horloges. La génération de ce code se fait par des processus automatisés dans System Generator, et n'est pas optimal. Le logiciel doit tenir compte de plusieurs paramètres, dont la fréquence d'échantillonnage, le processeur ciblé et l'outil de synthétisation. La **Figure 56** montre l'interface usager du logiciel System Generator. Les paramètres indiqués dans la figure sont les plus importants pour notre modèle. Le choix du processeur permet de déterminer le nombre de portes logiques nécessaires pour la réalisation du modèle. Si le nombre de portes logiques est trop petit, notre modèle ne pourra être implémenté dans le processeur cible. Comme expliqué tout au long de ce chapitre, l'horloge du FPGA est une valeur importante puisqu'elle servira de signal de commande à nos processus. Il est important que la fréquence d'échantillonnage de Simulink soit plus faible que l'horloge du FPGA, afin que le FPGA puisse générer la fréquence d'opération requis dans notre modèle. Le logiciel System Generator fera le rapport entre ces deux horloges et, à l'aide d'un fichier binaire inclus dans le code général, générera la fréquence d'opération du récepteur numérique.

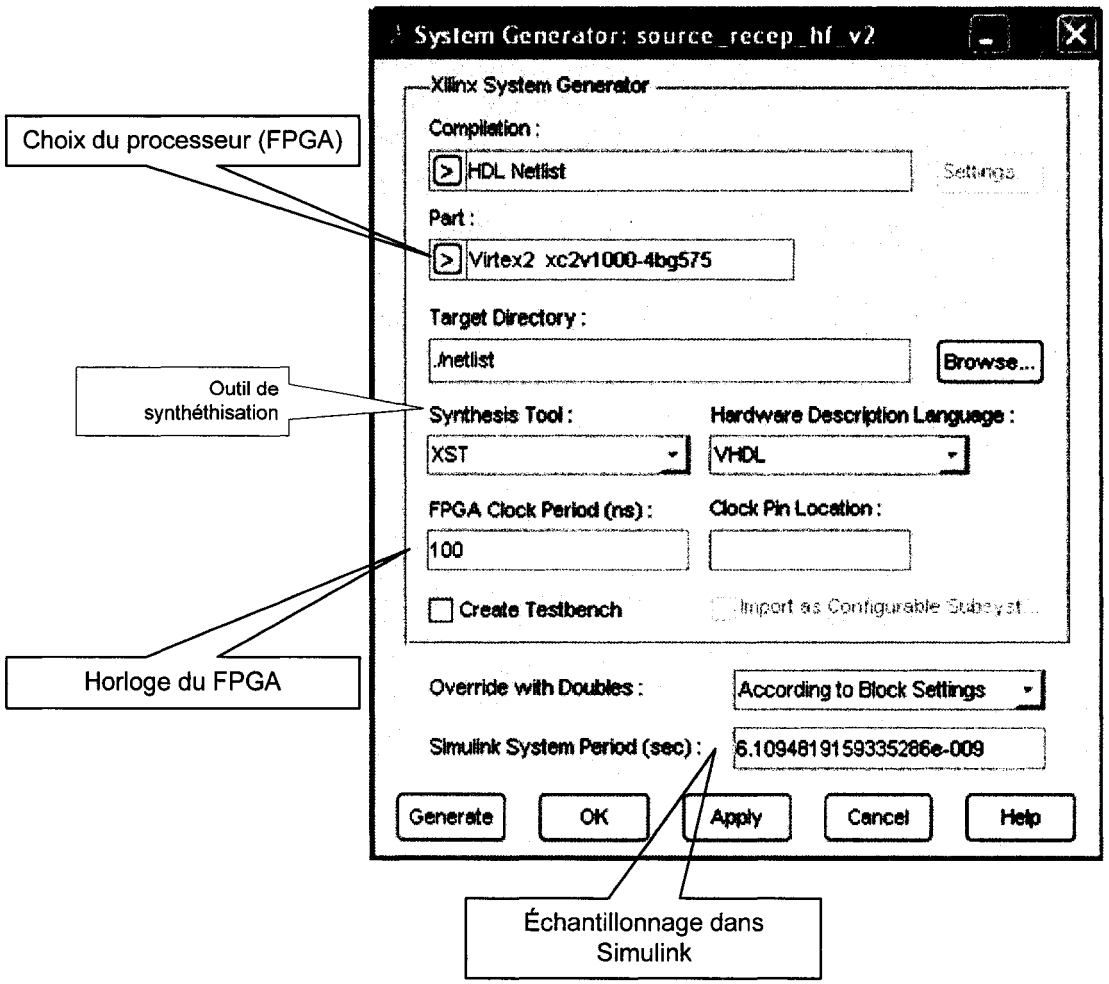



Figure 56 Paramètres de génération du code binaire

La génération automatisée n'est pas optimale mais elle permet de voir si des erreurs se sont glissées à l'intérieur de notre modèle. Le logiciel Project Navigator, un produit de Xilinx, sert à analyser le code VHDL généré. La quantité de ressources utilisées (porte logique, table de correspondance, entrée-sortie,...) est calculée par ce logiciel et une carte de l'emplacement des processus à l'intérieur du FPGA est même fournie par Project Navigator. Cette carte permet, de façon visuelle, d'analyser la répartition des ressources à notre disposition.

Tableau XIII

Utilisations des ressources à l'intérieur du FPGA

| **Ressources** | **Nombre utilisé** | **Nombre total** | **Pourcentage d'utilisation** |
|---|---|---|---|
| Slices | 937 | 5120 | 18 % |
| Bascule | 817 | 10240 | 8 % |
| Tables de correspondance | 2132 | 10240 | 20 % |
| Entrées – Sorties reliées | 50 | 432 | 12 % |
| Horloge globale | 1 | 16 | 6 % |

Enfin, la génération du code binaire se fait à l'aide de Project Navigator. Il est aussi possible de générer directement le code binaire à l'aide de System Generator. Cette option cependant empêche de voir la quantité de ressources utilisées, ce qui permet de valider l'implémentation de notre récepteur. Le **Tableau XIII** représente l'espace utilisé par le récepteur numérique à l'intérieur du FPGA. Il est normal que les tables de correspondance représentent un pourcentage élevé puisque les quatre NCO génèrent les sinusoïdes à partir de valeurs pré-calculés à l'intérieur de tables de correspondance. Cette analyse a été faite pour un récepteur avec un seul canal. Il sera donc possible d'ajouter d'autres canaux, puisque nous n'utilisons, au maximum, qu'un cinquième des ressources nécessaires.

## 4.5 Conclusion

La quantification a plusieurs effets sur le modèle du récepteur. La préparation pour l'implémentation en temps réel nécessite une étude approfondie du changement de variable. Nous avons donc pu établir, par l'étude de ses effets, le nombre exact de bits

dont nous avons besoin, soit un minimum de 8. Le processus inclus à l'intérieur de notre modèle a été modifié pour traiter les nouveaux types de variables. Suite à ces modifications, la table est mise pour générer les différents codes requis pour implémenter les processeurs cibles. Dans le prochain chapitre, nous aborderons les fonctionnalités et verrons les effets aux niveaux des algorithmes des modifications effectuées. Le passage pour le temps réel est un processus qui nécessite plusieurs étapes, pouvant introduire plusieurs erreurs. Cependant, avec l'analyse du présent chapitre, ces erreurs possibles devraient être minimisées.

# CHAPITRE 5

# ANALYSE EN COSIMULATION ET TEMPS RÉEL D'UN CANAL DE RÉCEPTEUR GPS

Les différentes composantes entrant dans l'implémentation d'un système en temps réel font en sorte que l'analyse des performances du système peut se faire sur plusieurs critères. L'évaluation peut donc se faire à plusieurs niveaux, autant dans l'optimisation matérielle que dans les performances de notre modèle.

## 5.1 Analyse des fonctionnalités des différents modules principaux

Le récepteur numérique GPS est divisé, tel que montré dans le CHAPITRE 3, en plusieurs processus de façon pyramidale. L'avantage de cette structure est qu'elle permet l'analyse des processus (divisé en blocs) de façon individuelle et indépendante. Cette approche nous est utile aussi lors de la détection d'erreur, puisque la source d'erreur peut être isolée et analysée séparément.

Dans le récepteur numérique, parmi l'ensemble des processus, trois sont particulièrement critiques pour la récupération de notre signal : la réplique du code de Gold, l'oscillateur local (NCO) et les filtres de la boucle PLL. Les prochains paragraphes analysent le comportement de ces processus de façon individuels.

### 5.1.1 Analyse des générateurs de code de Gold

Pour bien analyser les fonctionnalités des générateurs, nous devons étudier l'implémentation des différents paramètres entrant dans la génération des codes C/A. Tel qu'identifié à la section 3.2.2, deux paramètres sont utiles dans le générateur de code, en

l'occurrence l'horloge dictant la fréquence de génération et les valeurs de départ des registres.

Les valeurs de départ sont déterminées d'avance par l'usager et ont peu d'influence sur l'implémentation du générateur de code à l'intérieur du FPGA.

Le codage du signal GPS est fait à l'aide d'un code C/A, basé sur les codes de Gold. Ainsi, chaque code a une équation caractéristique et, avec cette clé, il est donc possible de décoder le signal. L'implémentation en temps réel du code utilisé par le signal GPS entraîne un changement du signal de commande. Dans la simulation Simulink, une onde carrée de 1,023 MHz est utilisée pour générer le code C/A. La génération de ce signal est faite par un oscillateur local avec un bloc de palier (+1/0).

Lors de l'implémentation, nous devons utiliser les librairies de Xilinx ce qui implique que la génération du code de Gold se fera avec des registres déjà programmés en VHDL par Xilinx. Ces blocs ont des caractéristiques de programmation déjà intégrées dont il faut tenir compte. Il est important de remarquer que les blocs mémoires en Simulink réagissaient sur un front montant de la fréquence. Or, parmi les caractéristiques des blocs de registre Xilinx, cette fonction n'existe pas. Nous avons donc deux solutions simples possible : adapter le signal de commande ou changer la conception de la génération du code de Gold. Changer la conception du bloc de génération du code C/A demanderait une révision complète du modèle puisque le générateur du code C/A intervient dans la source et le récepteur. L'adaptation du signal de commande est une solution simple sans impliquer de grand changement à l'intérieur du modèle. Il y a moyen, avec la commande *enable* des registres, de modifier le signal de commande tout en conservant ces caractéristiques pour la génération du code. Puisque les registres sont actifs lorsque la commande est à une valeur booléenne de 1 et inactive autrement, il suffit d'avoir une valeur de 1 pendant un seul échantillon, donc le temps de $1/F_s$. Ainsi, la donnée dans le registre sera envoyée au prochain registre et est maintenue jusqu'à la

prochaine valeur positive de commande. Pour ce faire, nous avons utilisé le signal à notre disposition, c'est-à-dire la sinusoïde de l'oscillateur local. Tout d'abord, elle est transformée en onde carrée ayant 1 ou -1 comme valeur. Cette onde carré est inversée ainsi que retardée d'un échantillon, et le signal résultant est soustrait à l'onde carré originale. Cette transformation nous permet de souligner les passages par 0 de l'onde sinusoïdale, avec la caractéristique suivante : une valeur positive au signal indique un passage dont la pente est positive. Or, nous voulons seulement la fréquence du signal, nous ne devons prendre qu'une des deux valeurs différentes de 0. En comparant avec un bloc relationnel d'égalité, il est possible de laisser passer seulement une de ces valeurs, ce qui correspond à un générateur de pulse. Ce pulse est envoyé comme commande aux registres qui transmette leur valeur l'espace d'un échantillon. La **Figure 57** montre le schéma à implémenter dans le FPGA pour commander le générateur du code C/A.

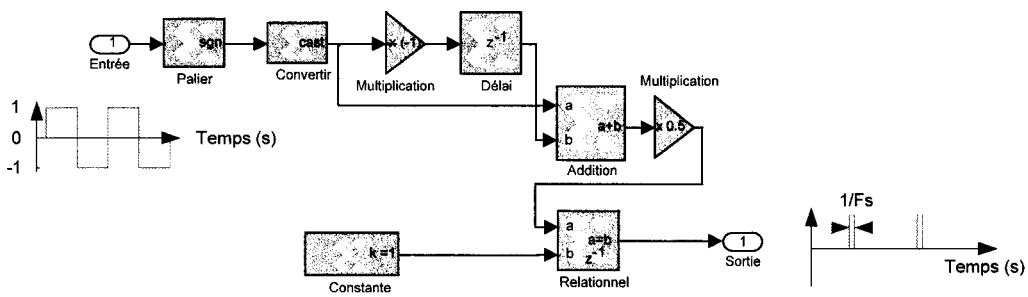



Figure 57 Génération d'un pulse de commande pour le Code de Gold

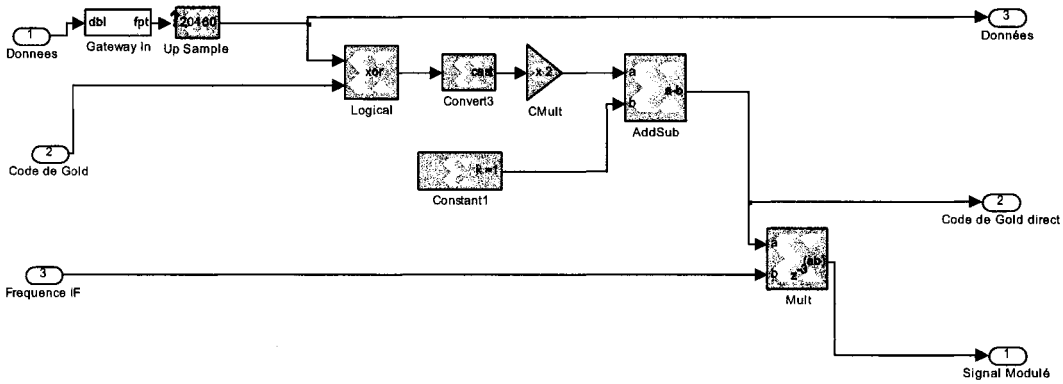



Figure 58 Génération en temps réel du signal GPS

Ce bloc est la modulation entre la porteuse et le code de Gold, mais avec une transformation au niveau de la donnée modulée. Puisque nous voulons une modulation BPSK, il est essentiel que la multiplication entre la porteuse et la donnée codée, puisse opérer le changement de phase. Le code et les données sont des valeurs booléennes, limitant ainsi leurs valeurs possibles à 0 et 1. Tout en gardant la valeur de 1, nous voulons diminuer la valeur de 0 à -1. L'expression mathématique suivante, très simple, et demandant peu de ressources, permet d'effectuer ce passage.

$$y_{out} = 2 \times x_{in} - 1 \qquad \textbf{(5.1)}$$

où les valeurs des entrées ne sont pas booléenne mais des valeurs signées en binaire, d'où la présence du bloc *convert* avant les expressions arithmétiques.

Avec le changement de la commande, le code C/A généré possède les mêmes caractéristiques temporelles et en fréquence que le système en simulation. La **Figure 59** montre le code C/A pour le satellite numéro 1 au temps $t_{SAT} = 0$.

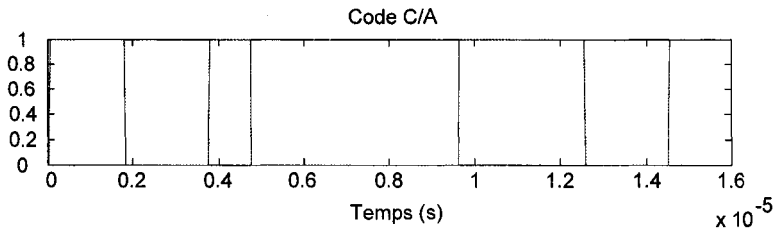



Figure 59 Code C/A généré avec un pulse de commande

### 5.1.2 Analyse du NCO

Parmi les éléments importants du récepteur, la clé pour la bonne réception réside dans la génération des répliques de la porteuse en phase et déphasé de 90 degrés. Le désétalement spectral se doit d'être rapide et précis

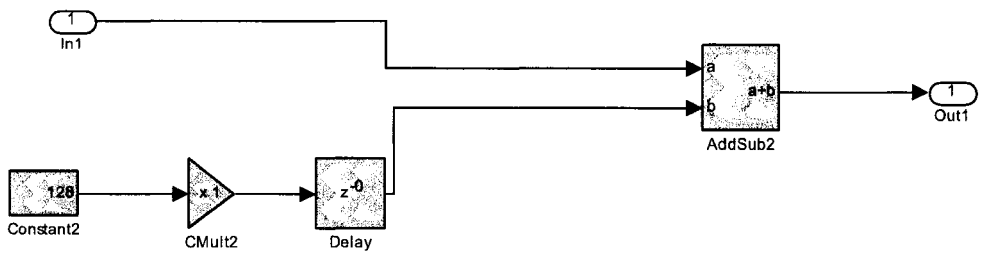



Figure 60 Calcul de la fréquence centrale du NCO

Ce bloc est utilisé pour générer la fréquence centrale du NCO qui recouvre le code sur la DLL. La constante de 128 est utilisée pour centrer la fréquence sur 1,023 MHz. Comme nous savons que le code a une fréquence de 1,023 MHz, nous pouvons automatiquement commencer les recherches autour de cette zone. À cette valeur est additionnée l'erreur du code calculée par les discriminateurs pour générer la fréquence réelle du signal.

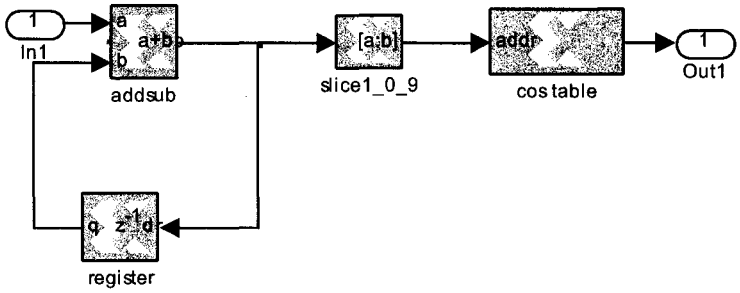



Figure 61 Génération de la sinusoïde à l'intérieur du NCO

La **Figure 61** montre le générateur de la réplique sinusoïdale de la boucle de phase. La valeur 128 imposée au bloc de fréquence centrale dérive des tests faits sur ce bloc. Avec une telle valeur, nous arrivons à produire exactement une fréquence à 1,023 MHz, impliquant que pour une valeur de 1, nous avons une sinusoïde de 8 kHz. Le **Tableau XIV** montre les fréquences reproduites du NCO en fonction de la valeur de commande.

Tableau XIV

Calcul de la fréquence centrale du NCO

| **Valeur** | **Fréquence centrale (MHz)** |
|---|---|
| 0 | 0 |
| 128 | 512 |
| 256 | 1 023 |
| 512 | 2 046 |

Cependant, il est important de noter que, de par sa composition, le NCO dépend de notre fréquence d'échantillonnage choisie pour l'ensemble de notre modèle. La valeur maximale reproduite par notre NCO est de $\frac{F_s}{2}$, répondant au critère de Nyquist. Cette valeur a donc de l'influence sur la valeur de commande nécessaire pour contrôler le NCO. Le **Tableau XIV** peut donc être représenté en fonction de la fréquence d'échantillonnage, ce qui est calculé au **Tableau XV**

Tableau XV

Valeur de commande du NCO en fonction de la fréquence d'échantillonnage

| **Valeur** | **Fréquence centrale (MHz)** |
|---|---|
| 1 | $\frac{F_s}{2048}$ |
| 256 | $\frac{F_s}{8}$ |
| 512 | $\frac{F_s}{4}$ |
| 1024 | $\frac{F_s}{2}$ |

La valeur maximale de 1024 est la résolution donnée par la conception de notre NCO. Puisque la table de correspondance a été construite sur 10 bits, nous avons donc 1024 possibilités de fréquence.

Le NCO étant limité par le nombre de bit, soit sur la table de correspondance ou sur la valeur de commande, nous devons avoir des processus pour aider à avoir la bonne

valeur. Les aides au NCO, l'ajustement fin de la phase et le processeur d'arrondi, viennent compenser pour ces limites. La réalisation, avec les blocs Xilinx, de ces processus est montrée sur les **Figure 62** et **Figure 63**. Ces modules se retrouvent à l'intérieur du NCO.

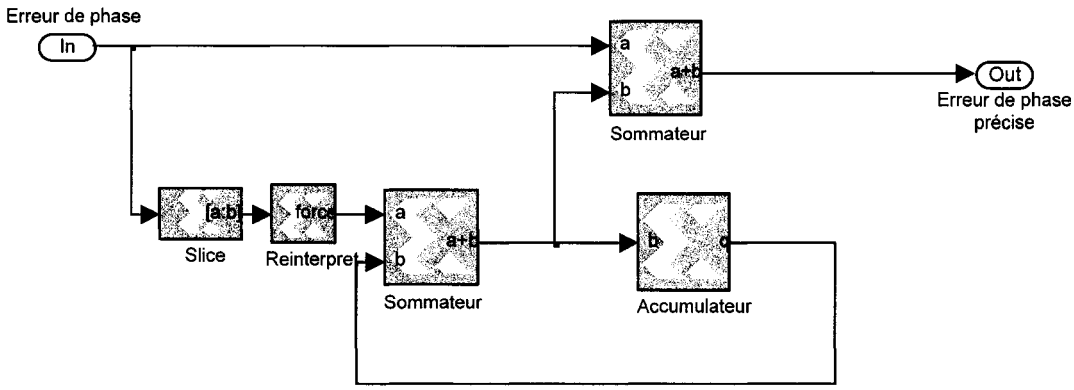



Figure 62 Réalisation du processeur d'arrondi

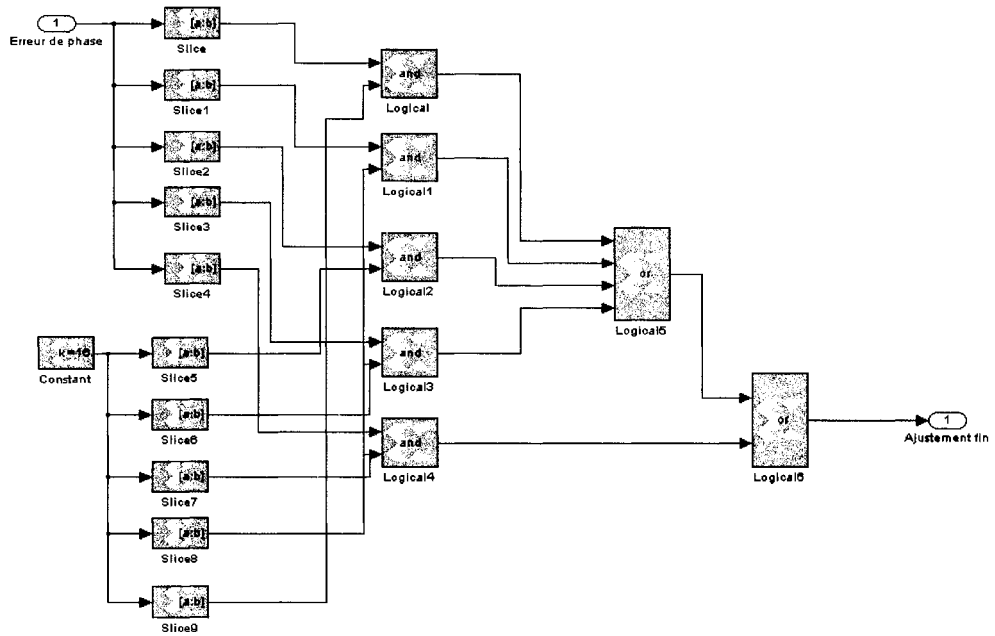



Figure 63 Réalisation de l'ajustement fin de la phase

## 5.2 Analyse des résultats en cosimulation

Après l'analyse individuelle des processus critiques de notre récepteur, nous pouvons étudier le modèle dans sa globalité. L'interaction entre les différents algorithmes de notre récepteur permettra de déterminer l'efficacité du récepteur numérique par rapport aux récepteurs analogiques conventionnels. Si l'on se réfère aux étapes du projet identifiées à la section 2.3, la prochaine étape est l'exécution du modèle à l'intérieur des processeurs en co-simulation. La **Figure 14**, montré au CHAPITRE 2, nous montre le cheminement des données lors de cette étape.

Pour l'implémentation du code binaire généré par la méthode expliquée à la section 4.4.2, le modèle a été séparé en 3 parties distinctes : la source (signal du satellite), la boucle de phase numérique et la boucle de code numérique. Puisque la source ne comporte pas d'algorithme complexe, l'analyse de son fonctionnement est plus simple.

Deux éléments peuvent nous guider pour analyser la justesse de nos résultats : la représentation spectrale des signaux ainsi que leur synchronisation. La **Figure 64** montre les différents signaux générés par le FPGA soit le code C/A, la porteuse à 2,046 MHz ainsi que le signal GPS résultant pour un seul canal de transmission.

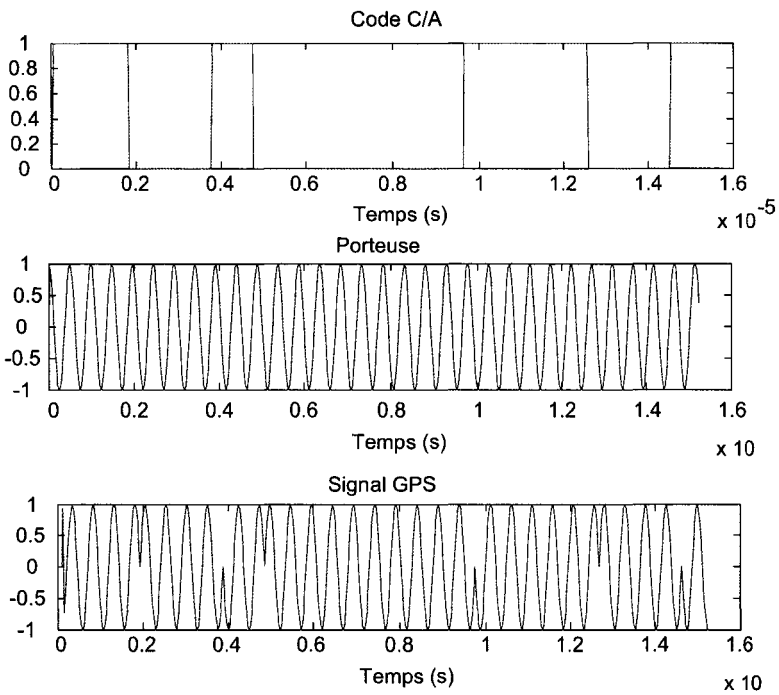



Figure 64 Signaux source du signal GPS en cosimulation

Chaque signal peut être caractérisé par différents paramètres pour l'identifier. Cependant, la période et l'amplitude du signal sont parmi les caractéristiques les plus utiles pour définir un signal. Le tableau suivant résume les caractéristiques théoriques et en co-simulation des signaux GPS.

Tableau XVI

Caractéristiques mesurées de la source GPS

| **Signal** | **Théorique (calculé)** | | **Cosimulation (mesuré)** | |
|---|---|---|---|---|
| | Période | Amplitude (V) | Période | Amplitude |
| Données | 0,02 s | 1 V | 0,02 s | 1 V |
| Code C/A | 0,977 µs | 1 V | 0,97 µs | 1 V |
| Porteuse | 0,488 µs | 1 V | 0,485 µs | 1 V |
| Signal GPS | 0,488 µs | 1 V | 0,485 µs | 1 V |

La différence entre les caractéristiques théoriques et mesurées des signaux sont minimes. Ces erreurs sont dues à la quantification des signaux ainsi qu'à la fréquence d'échantillonnage, mais elles sont négligeables ayant une différence de moins de 1 % sur le signal théorique.

Un des aspects importants de la modulation du signal GPS est la synchronisation des signaux intervenants lors de la modulation. Sur le troisième graphe de la **Figure 64**, on remarque que la modulation BPSK s'effectue bien au passage à 0 du signal, confirmant ainsi le respect de la synchronisation des signaux. Évidemment, il y a un certain délai entre le passage par 0 du code et du signal GPS en raison des algorithmes de modulation incorporés dans le modèle qui demandent un temps de traitement de plus d'un coup d'horloge. En se référant à la section 4.3, on peut aussi comparer ce délai à une perturbation tel un saut de phase.

L'implémentation de la source est la première étape pour la validation de notre modèle. Son comportement correspondant à la réalité, nous pouvons valider notre récepteur numérique à partir du signal généré dans le FPGA.

### 5.2.1 Analyse de la boucle de code numérique (DDLL) en cosimulation

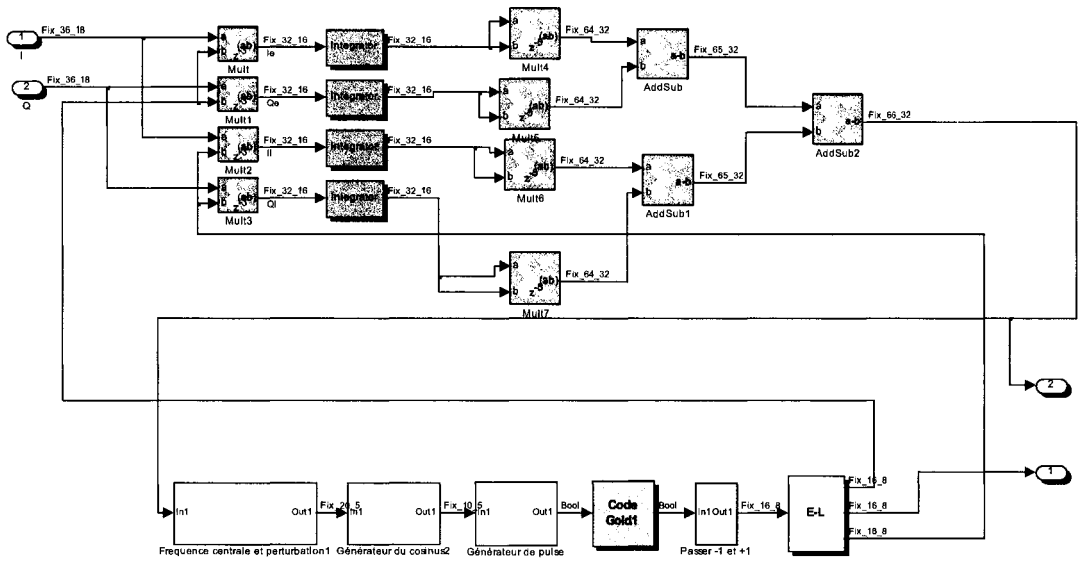



Figure 65 Schéma de réalisation de la boucle DLL en temps réel

La **Figure 65** permet de voir la boucle de code dans son entièreté. La boucle se décompose en trois parties : la génération du code, le discriminateur ainsi que les intégrateurs. Si l'on se rapporte au **Tableau VI**, les discriminateurs de la boucle de code ont des calculs complexes qui, au départ, étaient exécutés dans le DSP. Dans le cas du FPGA, on doit diminuer la complexité du discriminateur pour ne pas incorporer trop de délai à l'intérieur de la boucle. Les algorithmes concernant les discriminateurs d'enveloppe (AMR et AMR normalisé) impliquent des racines carrées, ce qui, en bloc Xilinx, est relativement lourd à construire et utilise énormément de ressources. Le

discriminateur de puissance AMR est beaucoup plus simple à implémenter quoique moins précis. C'est un choix à faire de sacrifier précision pour simplicité. Dans le futur du projet, il sera possible d'optimiser cet aspect du modèle afin d'augmenter la robustesse de notre récepteur.

La génération du code de Gold se fait exactement comme dans la source, avec un signal sinusoïdal transformé en pulse, tel qu'expliqué au début de cette section. Puis, tout comme en simulation, un bloc de délai est ajouté afin d'avoir le code en trois phase, c'est-à-dire en avance, en phase et en retard. Évidemment, la partie en phase est envoyée à la PLL afin de faire le désétalement spectral du signal.

Finalement la partie intégrateur est sensiblement pareille à celui de la PLL à l'exception du temps d'intégration. En effet, tel qu'expliqué au chapitre 3, le temps d'intégration de la boucle DLL est plus long que celui de la DLL puisqu'il lui faut plus d'échantillons du signal pour prendre sa décision. Par contre, cela fait d'elle la boucle la plus robuste au bruit. En prenant plus d'échantillons, la décision prise est donc plus sûre, mais ceci au détriment de la rapidité de notre boucle.

Pour valider le fonctionnement de notre boucle de code, un simple saut de phase est appliqué à la source sur le code seulement. La perturbation était seulement sur le code, la boucle DLL devra retrouver l'erreur et garder cette erreur constante, une fois la boucle accroché. La **Figure 66** nous montre la récupération des données sans l'aide de la boucle DLL. Comme les deux boucles sont imbriquées l'une dans l'autre, elles fonctionnent de façons complémentaires. Ainsi, chacune a besoin de l'autre, mais dans des certaines situations, tel qu'ici, une seule boucle peut suffire à trouver l'erreur. Donc, la boucle PLL à réussit à combler le saut de fréquence imposé sur le code seulement. Cependant, nous n'avons aucune valeur concernant l'erreur de phase. Cet exemple montre bien la complémentarité des boucles PLL et DLL.

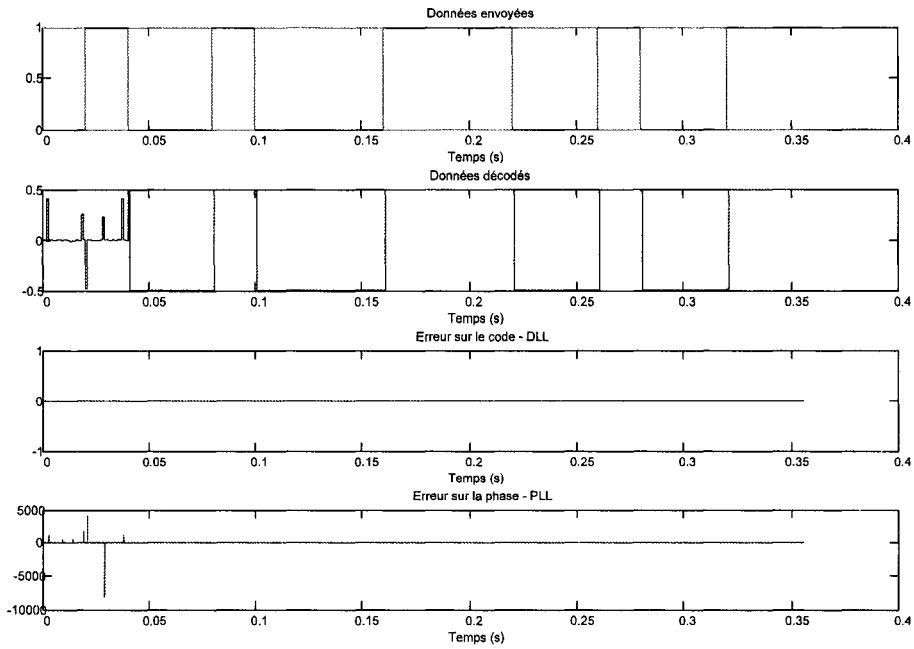



Figure 66 Réponse à un saut de phase sans l'aide de la boucle DLL

Le temps de réponse de notre récepteur sans boucle de code est rapide. La boucle de phase peut s'adapter plus rapidement puisque son intégration se fait sur une période plus courte. En activant la boucle de code, la complémentarité des boucles peut engendrer une diminution du temps de réponse mais cela augmente la robustesse de notre récepteur. En effet, si la boucle de phase réussi à s'accrocher, il se peut, puisque son temps d'intégration est plus long, que la boucle de code la fasse décrocher en calculant une erreur sur le code plus grande qu'à l'instant où elle termine son intégration. Cette situation s'illustre bien avec les accumulateurs. Ainsi, si les 2000 derniers échantillons donnent une erreur de phase nulle, mais que les 8000 premiers ont obtenues une certaine valeur, l'erreur calculée sera non-nulle, propageant ainsi une « fausse » erreur. La **Figure 67** nous montre cette situation, qui a fait augmenté le temps de réponse malgré l'aide de la boucle DLL pour un même saut de phase que la **Figure 66**. Cependant, on obtient une valeur comme erreur de phase sur le code.

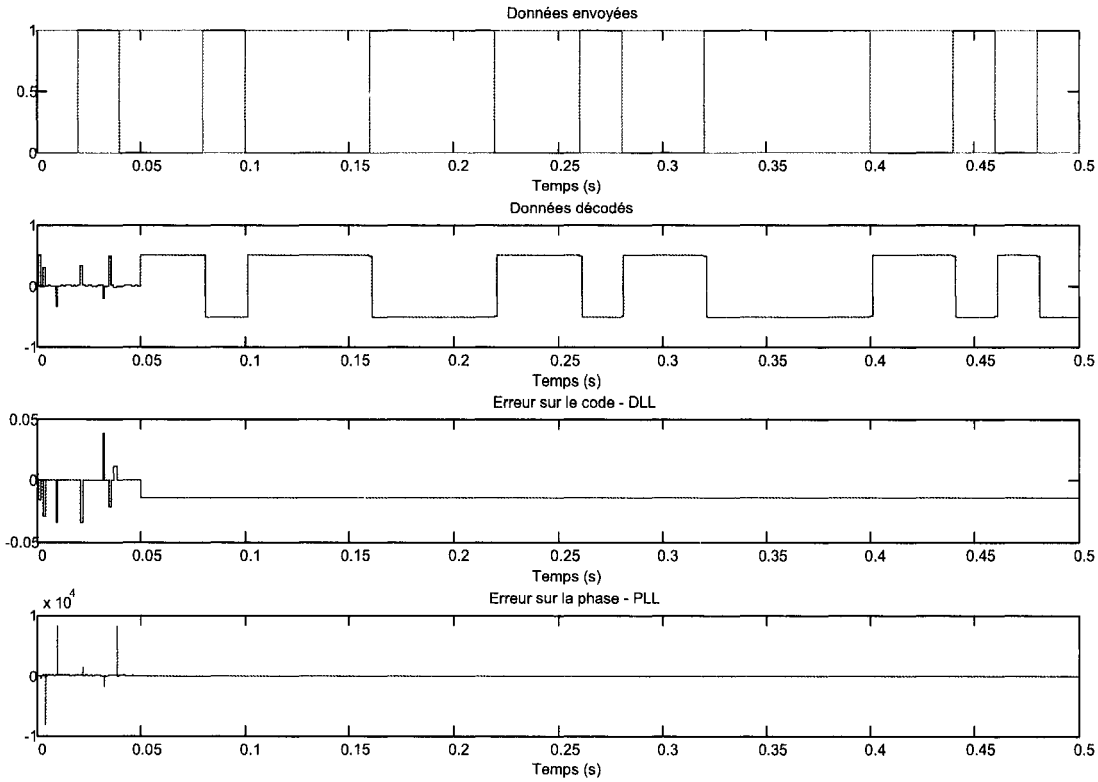



Figure 67 Réponse à un saut de phase avec l'aide de la DLL

### 5.2.2 Analyse de la boucle de phase numérique (DPLL) en cosimulation

Dans le CHAPITRE 4, plusieurs changements ont été effectués sur le modèle pour l'implémentation du récepteur numérique dans le processeur. La **Figure 68** montre la boucle de phase numérique avec les modifications pour l'implémentation.

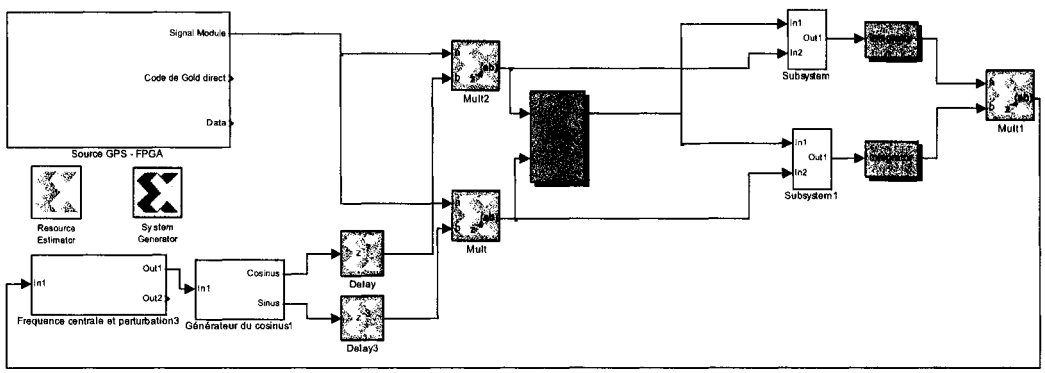



Figure 68 Schéma de la boucle de phase numérique

Dans notre boucle PLL, on remarque que, tel qu'analysé au CHAPITRE 3, le discriminateur choisi n'est pas le plus efficace envers le bruit. Cependant, pour simplifier notre modèle, nous nous devons de prendre un discriminateur simple qui utilise le moins de ressources possibles. Il s'agit donc du discriminateur produit croisé. La génération de la réplique du cosinus est constituée des mêmes éléments qu'à la source du signal. Cependant, au lieu d'y ajouter une perturbation comme paramètre d'entrée, l'erreur de phase est ajoutée à la fréquence centrale pour l'ajustement de la réplique.

Dans le cas d'un récepteur analogique, à la suite du décodage, un filtre passe-bas sert pour extraire la donnée. Tel que présenté plus tôt, à la section 3.4.3, l'intégrateur remplace le filtre passe-bas. L'intégration du signal se fait sur 1 ms ou 8000 échantillons avec la fréquence d'échantillonnage choisie.

Pour tester la réaction de notre boucle, un stresse dynamique est ajouté au signal à la source. L'effet Doppler, dont son implémentation a été expliquée à la section 3.1, ainsi produit permet de caractériser les limites de boucle de phase. Sachant que la boucle de

phase est une boucle de Costas, les données GPS extraites, si la boucle est accrochée, seront de la forme suivante, tel que démontré à la section 3.3 :

$$I\_filtré = \frac{1}{2} \cdot DATA(t) \tag{5.2}$$

Comme l'amplitude de notre signal de données GPS est de 1, nous devrions obtenir en sortie une amplitude de 0,5, et puisque l'intégration se fait sur 1 ms, cette valeur pourrait changer si la boucle n'est pas bien accrochée ou si le rapport signal à bruit (SNR) est trop faible. Dans le futur et pour l'amélioration du récepteur, le SNR pourrait être augmenté avec l'ajout d'un filtre FADP à l'entrée du récepteur.

L'effet Doppler appliqué, à la **Figure 69**, est de l'ordre de 4 kHz. Pour simuler l'effet Doppler à 4 kHz, les valeurs du **Tableau XIV** nous permettent de calculer que le paramètre de perturbation doit avoir la valeur unitaire. La **Figure 69** nous montre la réponse du filtre d'ordre 1 à cette perturbation. Tel que prévu, la boucle de phase avec le filtre d'ordre 0 est incapable de s'accrocher sur la porteuse ; il nous est donc impossible de récupérer les données GPS envoyées par la source. Conformément à l'analyse théorique et à la simulation, la boucle d'ordre 1 oscille lorsqu'elle arrive à la valeur maximale, soit $\pi/2$.

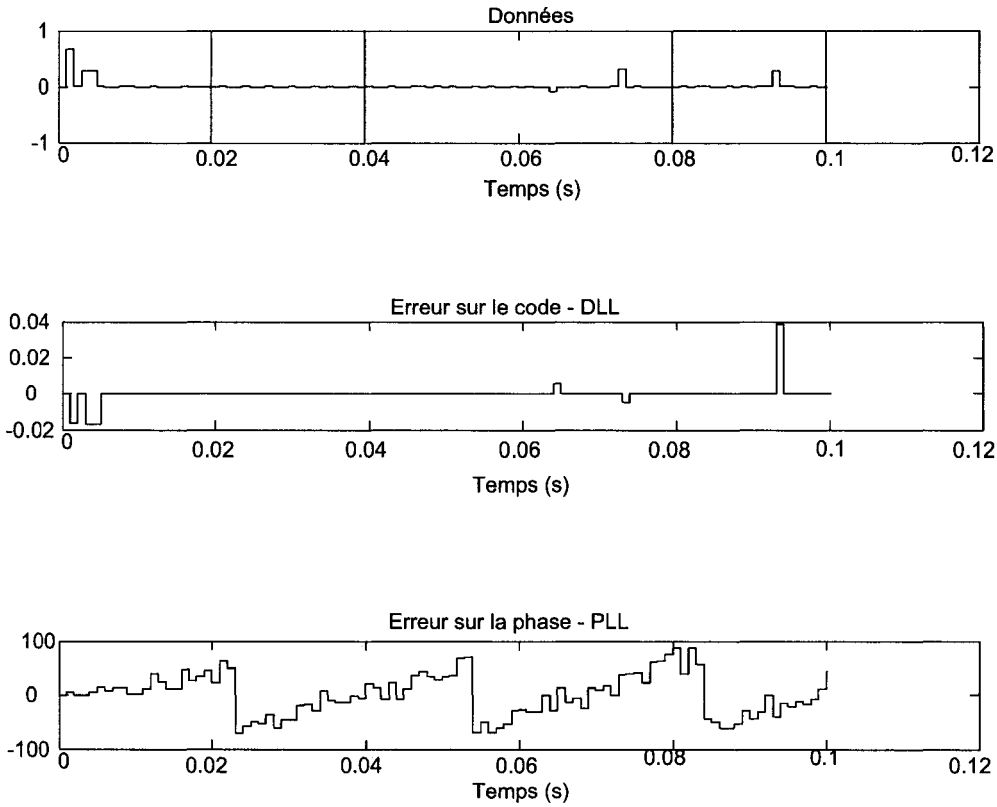



Figure 69 Réponse du filtre d'ordre 1 à l'effet Doppler

Même si le récepteur ne décode pas le signal GPS, il nous est possible de dire que le récepteur semble fonctionner puisqu'il réagit exactement comme la théorie et la simulation nous l'indique. On se rappellera qu'au chapitre 3, nous avons démontré que le filtre d'ordre 0 n'est pas capable de s'accrocher sur un effet Doppler, et nous avons le même résultat en co-simulation.

Afin de poursuivre le test du récepteur implémenté, nous allons appliquer maintenant un saut de phase au filtre d'ordre 1. Cette perturbation est intéressante pour 2 raisons : tout d'abord, le filtre d'ordre 1 semble bien résisté aux bruits et ensuite, l'erreur en régime permanent est nulle. Il sera donc facile de valider la performance de notre récepteur. En se référant à l'ANNEXE 1, la fonction de transfert du filtre d'ordre 1 est :

$$H(s)=\frac{\theta_s(s)}{\theta_e(s)}=\frac{\omega_0 s\left(a_2 s+\omega_0\right)}{s^2+a_2\omega_0 s+\omega_0^2} \tag{5.3}$$

Et, comme erreur en régime permanent :

$$\lim_{t\to\infty} e(t)=\lim_{s\to 0}\left\{s*\frac{s^3}{s^3+b_3\omega_0 s^2+a_3\omega_0^2 s+\omega_0^3}*\frac{\Delta\omega_e}{s^2}\right\}=0 \tag{(Erreur ! Style non défini.4)}$$

Les coefficients $b_3$ et $a_3$ sont tels que défini au CHAPITRE 3. Pour tester les limites du saut de fréquence, la première valeur testée sera une valeur faible afin de voir si la boucle réussit à s'accrocher, soit 1 kHz. Le **Tableau XVII** indique le temps de réponse de la boucle PLL en fonction du saut de fréquence. En regardant les résultats, sachant que l'intégration est de 1 ms, on voit le nombre d'intégration que la boucle prend avant de s'arrimer sur la fréquence centrale.

Tableau XVII

Temps de réponse de la boucle PLL d'ordre 2 à un saut de fréquence

| **Valeur du saut de fréquence (kHz)** | **Temps de réponse (ms)** |
|---|---|
| 1 | 2 |
| 10 | 56 |
| 20 | 123 |
| 80 | 175 |

La **Figure 70** montre la réponse à un saut de phase de 80 kHz.

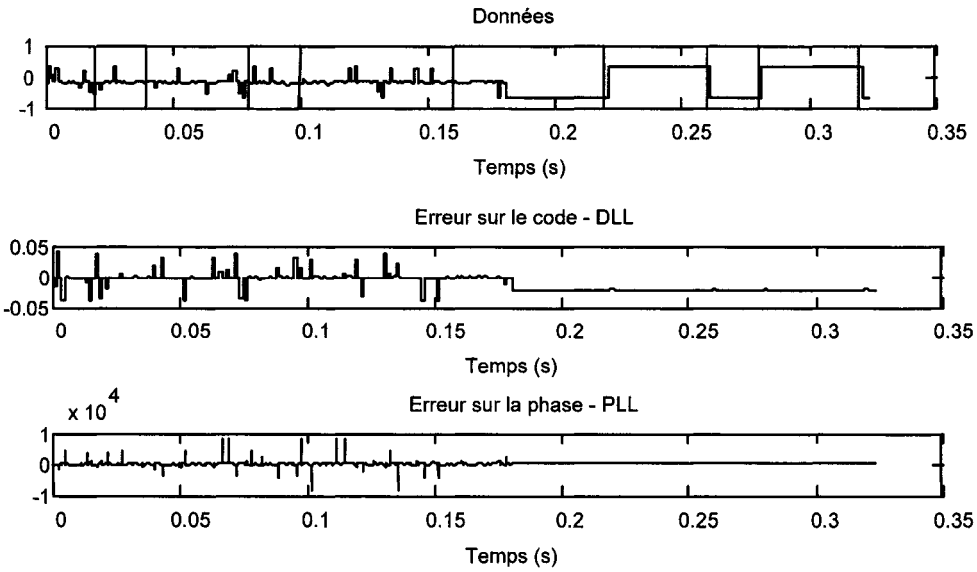



Figure 70 Réponse du filtre d'ordre 1 à un saut de phase

Même si ce saut de fréquence est élevé, le récepteur numérique, à l'inverse de la simulation, est capable de récupérer le signal. Sur le premier graphe de la figure, les données envoyées sont en rouge tandis que les données reçues sont en bleues. On observe exactement le moment précis où la boucle PLL réussit à s'accrocher, soit à 175 ms. On remarque aussi un délai entre les données envoyées et les données récupérées. Ces délais sont dus aux différents traitements des processus : les multiplications, les filtres ainsi que l'intégrateur demandent un temps de traitement au FPGA. Ces délais sont quantifiables et sont déjà notés dans les boîtes des Xilinx; nous pouvons donc à l'avance les prédire.

Un résultat que nous allons élaborer un peu plus loin est facilement remarquable dans cet exemple. La valeur du saut de fréquence limite est beaucoup plus élevée en co-simulation qu'en simulation avec Simulink. Plusieurs aspects du récepteur numérique viennent influencer ce résultat. Tout d'abord, la conception du NCO, avec les deux processus d'aide au recouvrement de la porteuse, a diminué le pas de variation de la fréquence centrale. Ainsi, lorsque le récepteur numérique est soumis à un saut de fréquence élevé, il ne tend pas à traquer la fréquence perturbatrice comme le faisait le

récepteur en simulation. Le pas de variation étant petit, le NCO ne peut que compenser légèrement une variation en fréquence. Si cette variation est instantanée, tel qu'un brouilleur PWI est conçu, le NCO ne sera que très peu affecté car le changement au niveau de l'intégrateur sera diminué par le diviseur à l'entrée du NCO. Mathématiquement, malgré le diviseur à l'entrée, le NCO devrait réagir comme en simulation puisque le diviseur garde le rapport de la variable. À titre d'exemple, si la variation est de 4kHz, le diviseur rend cette variation en rapport avec le **Tableau XIV**, soit la valeur de 1. Cependant, puisque le NCO utilise une table de correspondance à l'intérieur du FPGA, cette valeur a un effet amoindri que la valeur de 4 kHz en simulation. Ce gain en robustesse est un effet secondaire de la diminution de la précision expliqué à la section 4.2. En simulation, comme nous utilisons Matlab et des variables de type *double*, la précision est de l'ordre des $10^{-32}$, comparativement à $1/32^{\text{ième}}$ dans le cas des variables binaires utilisées dans le FPGA.

Ce gain vient aussi avec une autre faiblesse, soit le temps de réponse plus long également démontré dans l'exemple précédant. Puisque le NCO compense avec un pas de variation maximale fixe, le recouvrement de la fréquence porteuse est limité, à chaque intégration, à cette valeur maximale. Le pas de discrétisation étant plus petit, il est donc normal que le temps de réponse de notre récepteur numérique à l'intérieur du FPGA soit plus lent que celui en simulation.

La faiblesse de notre récepteur étant le temps de réponse, il serait intéressant de tester notre boucle PLL avec un stress dynamique, soit un effet Doppler. Avec l'aide de la DLL, la boucle du récepteur numérique réussit à traiter rapidement la variation en fréquence soumise à la source. En théorie, avec le théorème de la valeur finale, nous avons :

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0}\left\{ s * \frac{s^3}{s^3 + b_3\omega_0 s^2 + a_3\omega_0^2 s + \omega_0^3} * \frac{R}{s^3} \right\} = \frac{R}{\omega_0^2} \qquad \textbf{(5.5)}$$

On remarque, sur la **Figure 71**, que l'erreur de phase tend vers $\frac{R}{\omega_0^2}$, soit la valeur de 0,707.

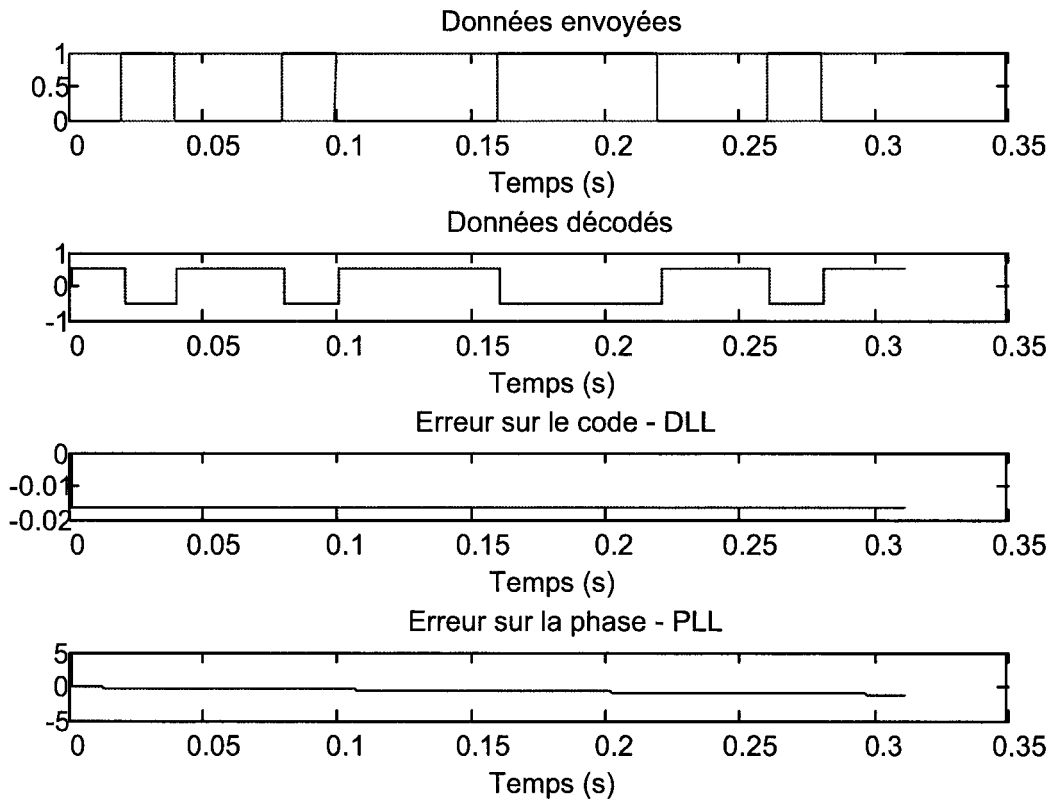



Figure 71 Réponse du filtre d'ordre 2 à un effet Doppler

Le temps de réponse du récepteur a été très rapide, soit 2 ms. Avec les deux derniers test, les caractéristiques d'un canal de récepteur numérique sont respectés. Le temps de réponse, la valeur maximale de l'effet Doppler et la puissance du signal à la sortie sont parmi les trois items qui peuvent caractériser le récepteur numérique. Par les différents tests effectués à l'intérieur de cette section, la valeur maximale applicable du stress dynamique a été validé ainsi que le temps d'accrochage de la boucle PLL.

## 5.3 Analyse des résultats en temps réel

Les résultats en co-simulation nous ont permis de valider la fonctionnalité de notre récepteur à l'intérieur des processeurs. En co-simulation, la plate-forme n'est pas encore tout à fait indépendante des processeurs à l'intérieur de l'ordinateur de contrôle. Matlab nous sert encore d'intermédiaire pour envoyer les commandes et les données. Pour s'assurer du fonctionnement de notre récepteur numérique, nous allons récupérer les données à l'extérieur de la plate-forme de développement via les convertisseurs numériques. Plusieurs types de données seront analysés. Tout d'abord, un convertisseur analogique numérique nous permettra de récupérer les valeurs des signaux nous intéressants tels les données récupérées, l'erreur de phase de la boucle PLL et l'erreur de phase de la boucle DLL. Nous mesurerons aussi la valeur de la puissance du signal de donnée reçu. Cette puissance mesurée nous permettra de calculer la perte dûe au récepteur. Nous proposons donc l'analyse en temps réel suivante :

1. Validation des signaux reçus (données, erreur de phase,...).
2. Analyse des seuils de sensibilités des boucles PLL et DLL.
3. Calcul de la puissance observé et de la puissance théorique.

Nous savons que la plus petite puissance à laquelle un récepteur peut recevoir le signal GPS est -160 dbW. Le rapport $C/N_0$ permet de caractériser les performances de nos boucles PLL et DLL. Comme donnée comparative, nous allons calculer ce rapport de façon théorique par l'expression suivante**[1]** :

$$C/N_0 = S_r + G_r - 10\log(\kappa T_0) - N_f - L \qquad \text{(Erreur ! Style non défini..6)}$$

où S est la puissance reçu (en dBW), $G_r$ le gain de l'antenne (en dBic), k la constante de Bolztmann (en Watt-s/K), $T_0$ la température équivalente de bruit (en degré Kelvin), $N_f$ et L sont des pertes dûes, entre autre à l'antenne et aux câbles (en dB). Les valeurs théoriques supposées de $N_f$ et de L sont de 4 et 2 respectivement. En remplaçant ces valeurs dans l'expression 5.6, le rapport $C/N_0$ théorique est de 38 db-Hz. Cette valeur nous servira à titre comparatif pour valider l'efficacité de notre récepteur et de nos boucles. Cependant, la valeur que nous allons mesurer sera le *signal-to-noise ratio* (SNR). Le SNR est comme le $C/N_0$ mais pour la bande de fréquence analysée. Pour pouvoir trouver le $C/N_0$ correspondant, nous allons utiliser la relation suivante :

$$SNR = \frac{C}{N_0} - 10\log(\frac{F_S}{2}) = \frac{C}{N_0} - 10\log(F_S) + 3 \text{ (dB)} \quad \text{(Erreur ! Style non défini.7)}$$

où $F_s$ est la fréquence d'échantillonnage. Tel que mentionné dans la section précédente, la bande maximal de notre récepteur numérique est de $\frac{F_s}{2}$. Ce calcul nous permettra de comparer les puissances théoriques et les puissances mesurées.

### 5.3.1 Analyse de la boucle de code numérique (DDLL) en temps réel

Le calcul du seuil de notre boucle DLL numérique se fait de façon similiaire à celui de la PLL. L'expression suivante nous permet de calculer l'erreur de phase de notre boucle :

$$\sigma_{DLL} = \sqrt{\sigma^2{}_{tDLL}} + \frac{R_e}{3} \leq \frac{Th}{3} \text{ (bribe)} \quad \text{(Erreur ! Style non défini..8)}$$

où Th est le seuil de linéarité maximal déterminé par le discriminateur choisi, $R_e$ est la gigue de phase due au stress dynamique dans la boucle $\sigma_{tDLL}$ est le 1-sigma de phase dû aux bruits thermiques et $\sigma_{DLL}$ est l'erreur de phase. Dans des conditions idéales, la gigue de phase est presque nulle et l'erreur de phase de la boucle DLL est presque entièrement dû aux bruits thermiques, ce qui revient à dire que l'erreur ne dépend que du discriminateur choisi. Dans notre modèle, nous avons implémenté le discriminateur avance moins retard (AMR) et sa variance se calcule de cette façon :

$$\sigma^2_{tdll\_AMR} = \frac{B_n d}{c/n_0} \leq \frac{Th}{3} \tag{5.9}$$

où d est le décalage entre les voies en retard ou en avance et la voie en phase. Le paramètre d et le seuil Th sont normalement égaux, soit de 0,5 dans notre cas puisqu'ils dépendent du discriminateur choisi. Cette valeur assurera un maximum de linéarité dans notre modèle, tel que montré par la **Figure 72**, nous donnant ainsi un seuil de 12,6 dB-Hz.

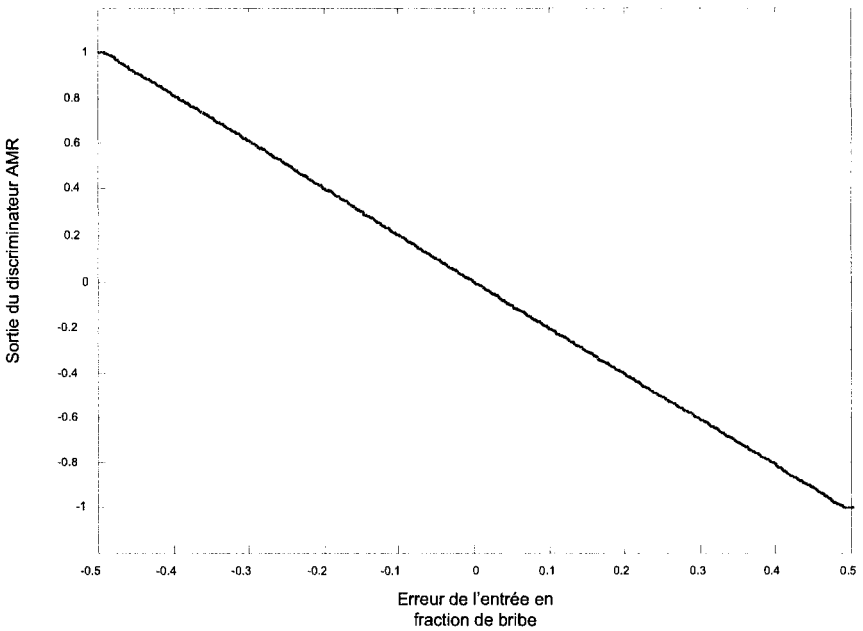



Figure 72 Erreur de phase du discriminateur pour un seuil théorique de 0,5 bribe

### 5.3.2 Analyse de la boucle de phase numérique (DPLL) en temps réel

Pour mesurer la puissance du signal reçu, nous n'appliquons aucune perturbation sur la source. L'expression suivante nous permet de calculer l'erreur de phase due aux bruits thermique[1] :

$$\sigma_{PLL} = \sigma_{tPLL} = \frac{360}{2\pi} \cdot \sqrt{\frac{Bn}{c/n_0} \cdot \left(1 + \frac{1}{2T\, c/n_0}\right)} \leq \frac{Th}{3} \quad \text{(deg)} \qquad \textbf{(5.10)}$$

où $\sigma_{PLL}$ est l'erreur de phase due aux bruits thermiques, $B_n$ est la bande équivalente de bruit du discriminateur, $c/n_0$ est le rapport $C/N_0$ exprimé de façon logarithmique, T est le temps d'intégration et Th est le seuil de la linéarité du discriminateur.

Pour notre modèle, le discriminateur implémenté est le produit croisé, que l'on considérera linéaire jusqu'à 60 degrés. L'expression 5.9 devient alors

$$\sigma_{PLL} = \sigma_{tPLL} = \frac{360}{2\pi} \cdot \sqrt{\frac{Bn}{c/n_0} \cdot \left(1 + \frac{1}{2T\, c/n_0}\right)} \leq \frac{60}{3} = 20 \quad \text{(deg)} \qquad \textbf{(5.11)}$$

En considérant que notre bande équivalente de bruit est de 10 Hz et que le temps d'intégration est 1 ms, le seuil de notre boucle PLL est de 20.89 db-Hz. Comme cette valeur est légèrement supérieure à notre valeur maximale, nous serons vraiment à la limite de notre boucle.

Dans l'intérêt de tester notre boucle PLL numérique, le signal de la source GPS a été ajouté de perturbation tel un saut de phase et un saut de fréquence. La **Figure 73** montre la réponse de la boucle numérique PLL.

A) Saut de phase

B) Saut de fréquence

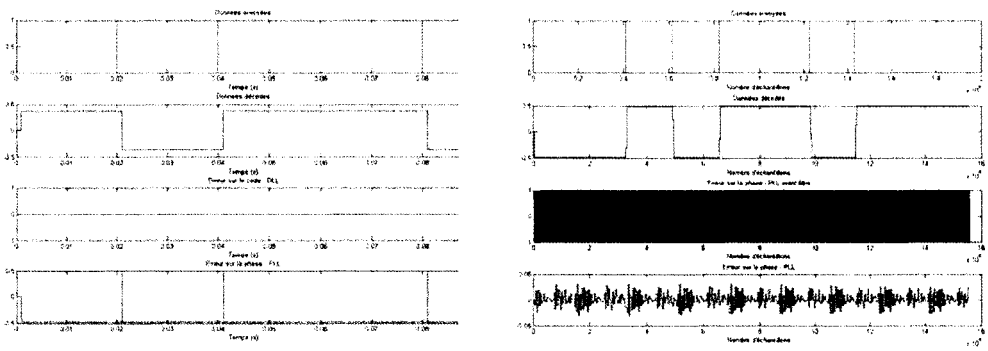


Figure 73 Réponse de la boucle PLL en temps réel

On remarque que pour le saut de phase, la boucle DLL est relativement peu mise à contribution puisque la PLL compense rapidement la perturbation, i.e dès la première intégration. Cependant, pour le saut de fréquence, nous avons ajouté l'effet du filtre au graphe. On remarque aussi que le filtre atténue la sortie du discriminateur. Cette situation est du au fait que nous avons utilisé des accumulateurs dans les filtres numériques d'ordre 1 et 2. Le temps de réponse est aussi affecté par cette situation, le rendant plus long que dans le cas d'une simulation.

### 5.3.3 Analyse du récepteur multicanaux en temps réel

Pour la validation d'un récepteur complet, nous avons un implémenté deux canaux dans un même modèle. Ces deux canaux simulaient le signal envoyé par deux satellites différents, soit dans notre cas, le satellite #1 et #2. La seule différence pour l'implémentation de deux satellites, est dans la génération du code C/A. Le sélecteur de phase (montré à la **Figure 19**) est différent pour les satellites. Les deux signaux GPS sont ensuite additionnés pour simuler l'effet d'un canal. Ce signal est envoyé à un seul récepteur qui décodera les informations du satellite #1, tel que montré à la **Figure 74**.

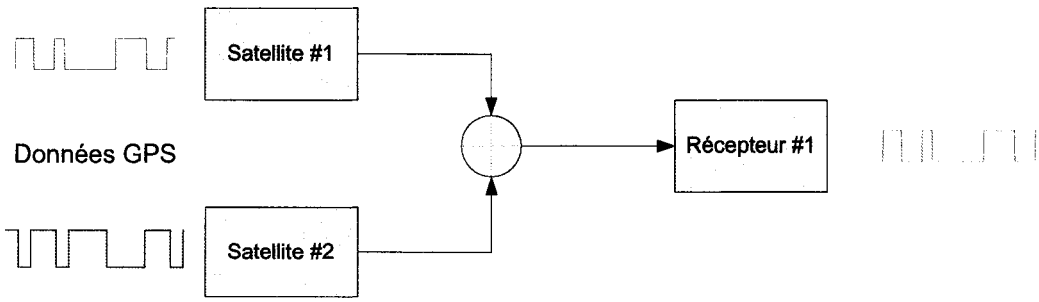



Figure 74 Schéma du modèle en mode multicanaux

Une fois l'implémentation terminée et le récepteur activé, les signaux du récepteur multicanaux ont pu être visualisé grâce à des oscilloscopes reliées aux convertisseurs numérique analogique de la plateforme de développement. La **Figure 75** nous montre le code C/A du satellite #1 à un instant t. On remarque que le code est bien à 1,023 MHz et que l'amplitude maximale est bien de 1. Les petits sauts sont dus aux convertisseurs qui font une approximation sur 14 bits de la valeur numérique. De plus, un certain voltage est perdu à travers les câbles et les connections entre les convertisseurs et l'oscilloscope.

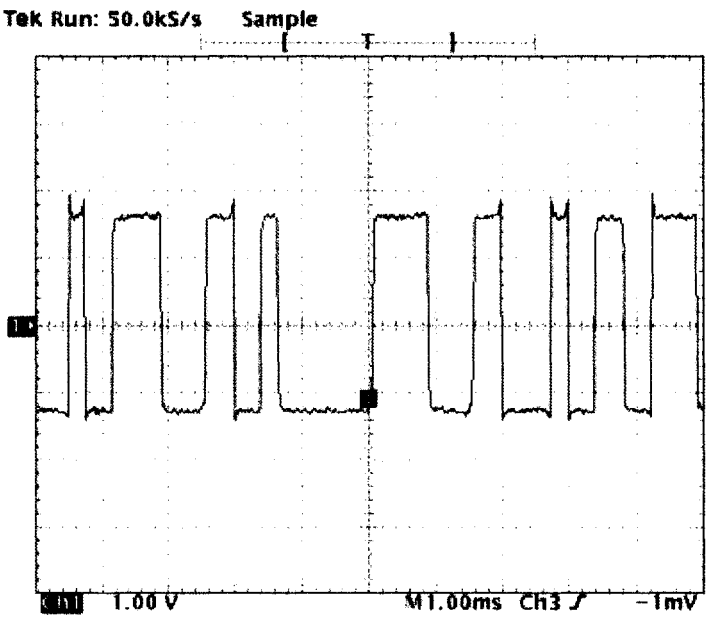



Figure 75 Code C/A en temps réel

La prochaine étape de notre modèle à vérifier est le signal GPS à la sortie de la source. On se rappelle qu'à cette partie, la porteuse module les données encodées. Comme nous sommes en bande de base, la fréquence de la porteuse est de 2,046 MHz, comme la **Figure 76** nous montre. On remarque sur cette figure que le signal GPS a la bonne fréquence mais le voltage au lieu d'être unitaire est de 0,707. Cette diminution du voltage résulte de la modulation entre la porteuse et les données encodées, qui ont des amplitudes unitaires. Comme nous avons une multiplication entre ces signaux, la valeur de l'amplitude a un gain de $\frac{\sqrt{2}}{2}$.

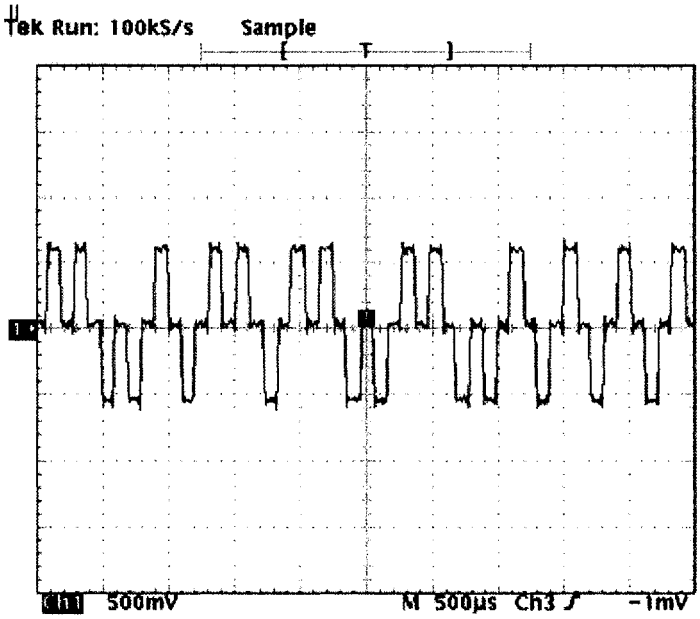



Figure 76 Signal GPS en temps réel

Le signal du satellite #1 est ensuite ajouté au signal du satellite #2 et envoyé au récepteur. Le mélange des deux signaux peut causer le récepteur, programmé pour s'accrocher au signal #1, à se bloquer sur le signal GPS du satellite #2. Cependant, avec la validation des boucles PLL et DLL, le récepteur numérique a su décoder les données telles que montrées sur la **Figure 77**. La période est de 0,02 seconde, tel que prévu. Cependant l'amplitude n'est pas toujours d'un volt puisque la boucle essaye encore de s'accrocher. C'est pourquoi, nous avons des valeurs supérieures à 1 (la sortie de l'intégrateur) mais on remarque que ces valeurs tendent vers 1.

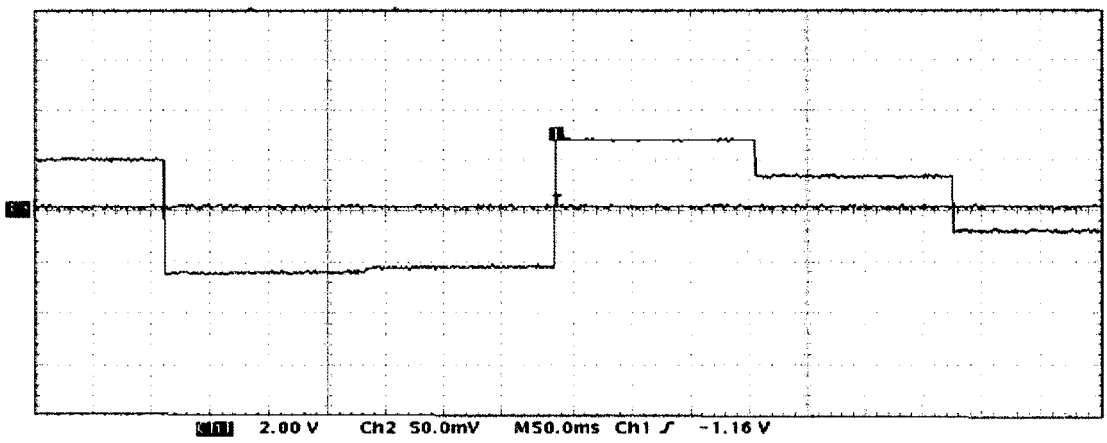



Figure 77 Données de navigation en temps réel

À la lueur des résultats précédents, le modèle de notre récepteur numérique est fonctionnel et génère peu de bruit. Tout comme en simulation et en co-simulation, le récepteur numérique réussit à retrouver les données de navigation et ce, même en présence d'un autre signal GPS de même fréquence. Les discriminateurs, même s'ils ne sont pas les plus performants des choix disponibles, donnent des résultats probants, tant au niveau de la récupération que du bruit généré. Puisque les ressources étaient limités, les choix faits l'ont été pour un rapport performance versus ressources utilisés maximal.

## 5.4 Perspective d'optimisation du système

Évidemment, la modélisation d'une chaîne de communication GPS est un travail demandant l'utilisation de beaucoup de ressources. Dans notre cas, nous avons décidé d'y aller avec une méthode graphique directe. L'avantage d'utiliser Matlab/Simulink, c'est que c'est une façon de haut niveau de programmer. De façon visuelle, avec des blocs programmés d'avance, le récepteur numérique a pris forme. Cependant, avec les blocs des libraries Simulink et Xilinx, plusieurs contraintes nous ont été imposé par les développeurs de ces blocs. Or, notre système a des caractéristiques bien à lui qui ne sont pas toujours facilement adaptable aux outils utilisés. Ainsi, à travers ce projet, l'idée de

programmer nous-même ce récepteur numérique nous est apparue comme un pas logique dans le futur du projet.

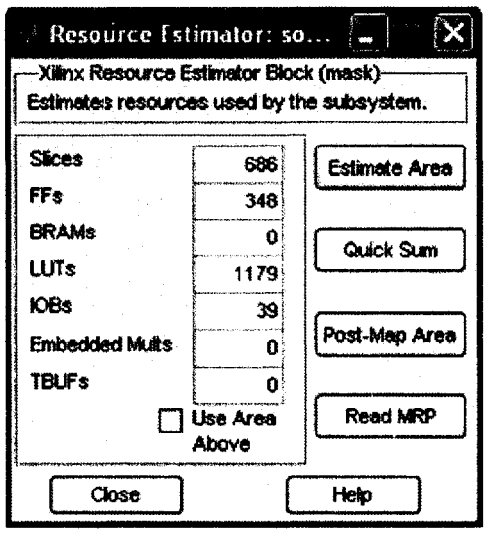



Figure 78 Ressources utilisées par le signal de la source GPS

Tel qu'expliqué au CHAPITRE 4, nous avons décidé d'implémenter l'ensemble du récepteur à l'intérieur d'un FPGA. Le langage VHDL permet d'utiliser le code binaire pour programmer un FPGA. Dans notre méthodologie, le code VHDL était généré par les outils de Xilinx à l'intérieur de Matlab. Dans la librairie de Xilinx, nous avons des outils pour estimer la quantité de ressources, dont nous avons un aperçu à la **Figure 78**. On remarque l'importante place des tables de correspondance, soit une par oscillateur local. Pour optimiser les ressources, il serait utile de pouvoir programmer une table de correspondance commune à tous les oscillateurs.

Pour diminuer les contraintes, nous avons programmé en VHDL certains modules de base, tel la source GPS, pour voir l'avantage d'un tel procédé. La **Figure 79** représente la schématisation du signal GPS L1 pour sa programmation en VHDL.

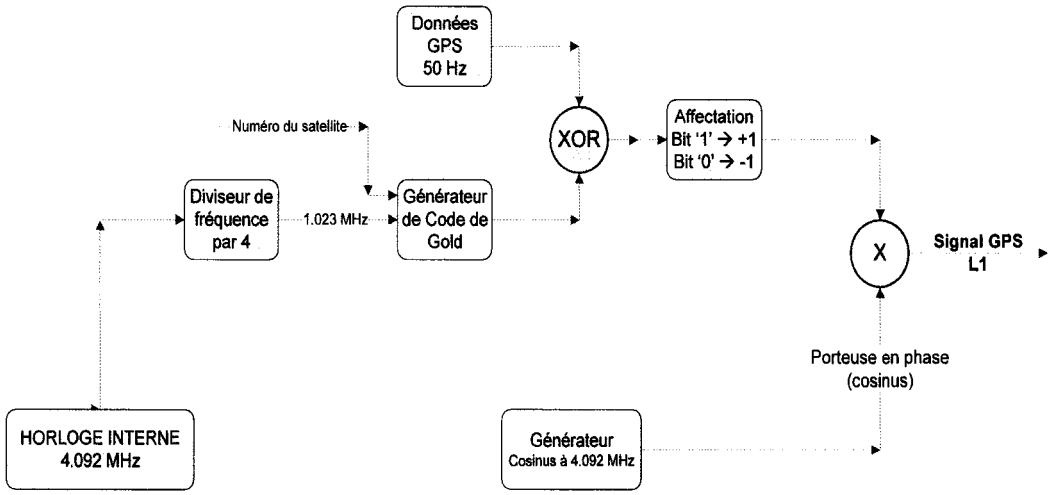



Figure 79 Schéma de génération du signal GPS L1 en VHDL

Tout comme dans le cas de Simulink, une fréquence d'horloge (ou d'échantillonnage) est nécessaire pour la simulation en VHDL de notre code à l'intérieur de ModelSim. Il faudra donc tenir compte de cette fréquence lors de la programmation en VHDL du récepteur numérique, tel que nous l'avons fait pour Simulink, expliqué à la section 3.5.

Tableau XVIII

Utilisation des ressources selon le mode de programmation

| | **Ressources disponibles** | **Code C/A avec les blocs Xilinx** | **Code C/A en VHDL** | **Différence** |
|---|---|---|---|---|
| **Bascule** | 5120 | 27 | 13 | 52 % |
| **Bascule D** | 10240 | 44 | 22 | 50 % |
| **Table de correspondance** | 10240 | 14 | 5 | 64 % |
| **Entrée-Sortie reservé** | 432 | 11 | 1 | 91 % |
| **Horloge** | 16 | 1 | 2 | -100 % |

Avec la programmation en VHDL, le nombre de ressources utilisé est beaucoup moindre, ce qui permet l'ajout de processus pour aider le récepteur numérique à récupérer plus rapidement les données GPS. Les résultats de cette partie du développement sont capitaux pour la suite du projet. Ils donnent un indice sur la piste à suivre pour le développement plus complet d'un récepteur hybride.

## 5.5 Conclusion

Les résultats de ce chapitre sont intéressants pour la suite du développement des récepteurs numériques. En effet, après avoir présenté les résultats en simulation dans le CHAPITRE 3, ce chapitre y fait suite en présentant les résultats en co-simulation et en temps réel. Il a été démontré que le récepteur numérique développé résiste bien aux

perturbations tel l'effet Doppler mais qu'il perd de son efficacité au niveau du temps de réaction. Les changements effectués au CHAPITRE 4 ont changé les propriétés du récepteur et ont eu une grande influence sur ces performances. Tout d'abord, les changements ont fait en sorte que nous devions limiter les erreurs de phase à envoyer au NCO. Ces changements nous ont forcé aussi à revoir la génération des signaux tel le code C/A. Cependant, en terme de fonctionnement, le récepteur numérique implémenté dans les processeurs a réagi de la même façon qu'en simulation ou qu'un récepteur analogique. En terme de puissance, le récepteur numérique peut accepter des rapports $C/N_0$ très acceptable et génère peu de bruit thermique. Pour diminuer l'effet des changements, la programmation via le langage VHDL a été aperçu et on a pu constater la diminution au niveau des ressources allouées, permettant ainsi à d'autre processus de venir aider le récepteur.

# CONCLUSION GÉNÉRALE

L'étude d'une chaîne de communication GPS a débuté par l'analyse des variables d'environnements et des changements à prévoir dans les signaux GNSS au cours des prochaines années. Cette analyse nous a permis de voir l'état actuel des signaux et les conséquences des changements à venir (CHAPITRE 1). L'ajout de plusieurs constellations de satellites et la modernisation du système GPS fait en sorte que les récepteurs actuels seront désuets et manqueront de précision par rapport aux futurs récepteurs. Mais pour développer ces récepteurs, comme certains signaux GNSS ne sont pas tout à fait défini, une architecture flexible et adaptative est nécessaire. C'est par ce constat qu'est né le projet SDN.

Basé sur le SDR, le projet SDN se veut l'utilisation des concepts SDR mais appliqué en navigation. Le SDR ayant radicalement transformé la conception des objets utilisant des ondes radio, le projet SDN a ce même objectif et est une base pour conceptualiser les futurs récepteurs numériques hybrides (GPS/Galileo). L'analyse des variables d'environnement a guidé pour la détermination des outils de conceptions du SDN (CHAPITRE 2). Pour ce faire, il a fallu conjuguer une approche logicielle et matérielle et superposer les deux méthodologies pour arriver à une méthodologie globale qui tenait compte des caractéristiques des processeurs choisis. Le choix des processeurs étant un élément important, il a fallu déterminer quels étaient les caractéristiques importantes recherchés, soit la vitesse d'exécution et la complexité de nos processus. Sur ces critères, il a été décidé de choisir deux processeurs : un FPGA et un DSP, chacun ayant des caractéristiques propres avantageuses à notre conception. Les limites des ces processeurs ont aussi été étudiées pour mettre des balises de conceptions pour notre modèle de récepteur numérique. Il a aussi été convenu que le développement du récepteur se ferait en Matlab/Simulink pour deux raisons majeures. Premièrement, un travail de conception avait déjà été commencé en ce sens dans un autre projet de maîtrise. Deuxièmement, les

outils logiciels retenus pour la conception pouvaient tous interagir avec Simulink, rendant la séparation de modèle vers les deux processeurs beaucoup plus simple.

Pour la conception du modèle, l'étude des signaux GNSS du chapitre 1 s'est avérée essentiel. À partir des principes de base sur les boucles de Costas, la conception du récepteur a été développée sur Simulink. Les différents discriminateurs utilisés, tant pour la boucle de code que la boucle de phase, ont été analysés afin de choisir quel serait le plus performant pour l'implémentation. La même étude a été faite sur les filtres des boucles. Tout en tenant compte des performances des processus à l'intérieur du récepteur numérique, sa conception a été faite en ayant, comme toile de fond, le passage en temps réel. Les différents algorithmes du récepteur numérique ont été décortiqués pour déterminer le cheminement des données entre le FPGA et le DSP. Au CHAPITRE 4, les différents changements et leurs effets ont été pris en compte et le récepteur numérique fut modifié en conséquence. Les algorithmes du récepteur ont alors suivi la méthodologie développée au CHAPITRE 2 pour l'implémentation dans les processeurs respectifs.

Chaque processus a été analysé en co-simulation et en temps réel. En co-simulation, nous avons vu que le récepteur numérique était très robuste et pouvait soutenir une rampe de phase jusqu'à 80 kHz. Même si cela est supérieur, de beaucoup, qu'au stress dynamique existant dans la réalité, ce résultat nous porte à croire qu'en allouant moins de ressources, le récepteur pourrait être plus rapide en modifiant sa conception. Le temps de réponse de notre récepteur étant plus long que celui souhaité, il serait favorable de modifier sa conception pour diminuer ce temps. De plus, au niveau de la puissance, le récepteur numérique se comporte tel que prévu même s'il génère un peu plus de bruit qu'en théorie mais en quantité relativement faible, soit 2,8 dB. Enfin, quelques pistes d'optimisation furent lancées afin d'augmenter les performances du récepteur numérique, permettant le développement des récepteurs hybrides.

La présente méthodologie a été basée sur l'utilisation et l'interaction entre différents logiciels. Principalement, dans notre étude, nous avons utilisé le logiciel Xilinx System Generator, qui permet de convertir à partir de blocs d'une librairie de Matlab, un modèle Simulink en un code VHDL. Les avantages d'une telle procédure est que le designeur n'est pas obligé d'être programmeur pour arriver à l'implémentation. Cette méthodologie permet aussi un développement rapide de nouvelles technologique. Cependant, il y a plusieurs lacunes à utiliser une telle méthode. Tout d'abord, dans la génération des codes VHDL ou C++, les blocs utilisés ont un code déjà alloué pour System Generator. Ce code n'est pas optimal, tel que vu dans la dernière section du chapitre 5, et le programmeur pourrait arriver à un code beaucoup plus performant. Aussi la méthodologie n'est pas trop flexible, puisque à partir du modèle Simulink, les changements effectués pour le passage en temps réel sont limités par les blocs disponibles dans la librairie de Xilinx. Pour ces raisons et les complexités qu'elles engendrent, la méthodologie développée n'est pas recommandé pour le développement de futur récepteur GPS hybride. Cependant, cette méthodologie peut servir de base pour l'apprentissage des nouvelles technologies.

# RECOMMANDATIONS

Tout au long de ce mémoire, les sujets de projets de recherches possibles, suite au développement de ce récepteur numérique, sont nombreux et très variées. Parmi ceux-çi, notons :

a) Développement en VHDL du récepteur GPS numérique.
b) Développement de nouveaux corrélateurs hybride.
c) Architecture d'un récepteur numérique Galileo.
d) Conception de convertisseurs analogique à numérique de haute fréquence.

Tel qu'indiqué dans le dernier chapitre de ce mémoire, les recherches pourront se diriger vers le développement en VHDL du récepteur GPS numérique. Le travail à l'intérieur de Simulink a ses limites et celles-ci sont presque déjà atteintes. Les FPGA actuels peuvent fonctionner à des fréquences très élevées et possèdent un nombre suffisant de ressources pour l'implémentation d'un récepteur GPS numérique très efficace. La section 5.4 nous a montré un aperçu des gains qui pourraient être fait en ce sens. De plus, les outils logiciels développé pour la programmation des FPGA évolue et sont plus adaptés à notre problématique.

Aussi, afin de pouvoir développer un récepteur numérique efficace et performant, les convertisseurs analogique numérique existant ne peuvent fonctionner aux fréquences des signaux GNSS ou, s'ils existent, sont excessivement coûteux. Pour valider de façon claire les récepteurs numériques, il faudra utiliser les signaux GPS réels et pour cela, avoir des pièces électroniques pouvant convertir ces signaux.

Il faut se rappeler que ce projet de recherche est une base pour le projet SDN et qu'il existe encore beaucoup de développement à venir pour arriver à l'objectif final, soit un récepteur numérique complètement reprogrammable.

# ANNEXE 1

# ANALYSE THÉORIQUE D'UNE BOUCLE DE COSTAS

La boucle de Costas permet une démodulation optimale du signal BPSK. En plus d'être très robuste, elle est insensible au changement de phase de 180°. Cette caractéristique est essentielle pour pouvoir démoduler le signal BPSK.

Le principe d'une boucle PLL est simple : on multiplie le signal reçu par une référence variable générée à l'interne et on filtre le produit. On utilise le résultat (qui correspond à l'erreur de phase entre le signal d'entrée et la référence) pour ajuster la fréquence de la référence variable (le VCO).

La configuration de la boucle de Costas est illustrée à la figure suivante :

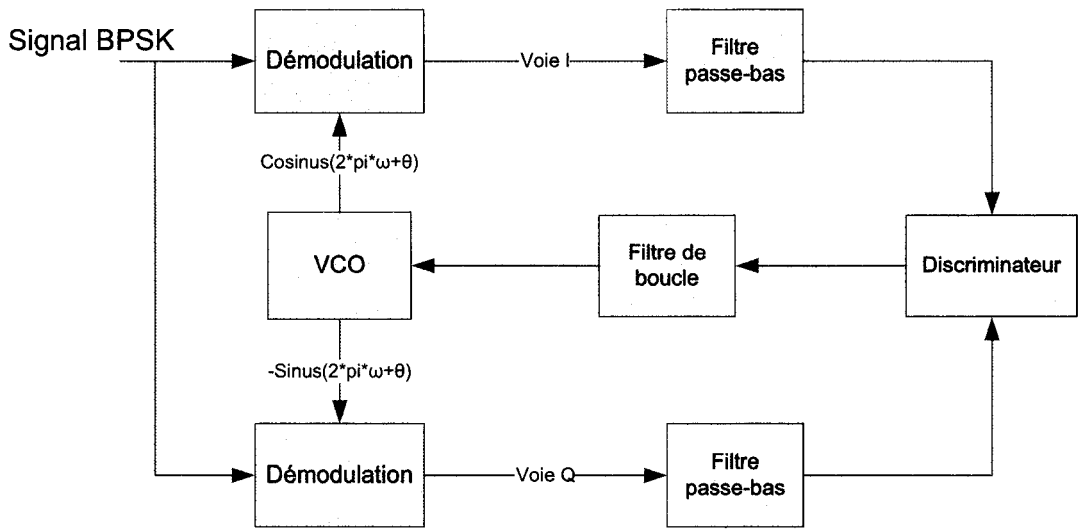



Figure 80 Exemple de boucle de Costas

La boucle de Costas génère un signal en phase (I) et en quadrature (Q) pour démoduler le signal BPSK. Le signal en quadrature doit être en avance de 90° sur le signal en phase. Dans le cas de la modulation BPSK, le signal en phase est un cosinus. Le signal en quadrature correspond donc à –sin(x).

Le signal BPSK est défini par : $BPSK=\cos\left(\omega_c t+DATA(t)\frac{\pi}{2}\right)$ où *DATA* représente le message binaire et correspond à ±1.

Un changement de phase de 180° équivaut à multiplier la porteuse par « -1 ». L'expression du signal BPSK peut donc être simplifiée :

$BPSK=DATA(t)\cdot\cos(\omega_c t)$

Pour analyser le comportement de la boucle, deux propriétés trigonométriques sont utilisées :

1. $\cos(a)\cos(b)=\frac{1}{2}[\cos(a+b)+\cos(a-b)]$
2. $\sin(a)\cos(b)=\frac{1}{2}[\sin(a-b)+\sin(a+b)]$

On analyse d'abord le produit du signal BPSK par la composante en quadrature:

$$I = BPSK * \cos(\omega_c t) = \left[ DATA(t) \cdot \cos\left(\omega_c t\right)\cos\left(\omega_c t\right)\right] \quad \textbf{(1.2)}$$

Selon la propriété trigonométrique, on obtient :

$$I = \frac{1}{2}\left[ DATA(t) \cdot \cos(0) + DATA(t) \cdot \cos\left(2\omega_c t\right)\right] \quad \textbf{(1.2)}$$

Une fois filtré par le filtre passe-bas, on obtient :

$$I\_filtré = \frac{1}{2} \cdot DATA(t) \quad \textbf{(1.3)}$$

On analyse ensuite le produit du signal BPSK par la composante en quadrature :

$$Q = BPSK \cdot \sin\left(\omega_c t\right) = DATA(t) \cdot \cos\left(\omega_c t\right) \cdot \sin\left(\omega_c t\right) \quad \textbf{(1.4)}$$

Selon la propriété 2, on obtient :

$$Q = \frac{1}{2}\left[ DATA(t) \cdot \sin(0) + DATA(t) \cdot \sin(2\omega_c t) \right] \quad \textbf{(1.5)}$$

Une fois filtré par le filtre passe-bas, on obtient :

$$Q\_filtré = 0 \quad \textbf{(1.6)}$$

Cette analyse montre que pour une porteuse en phase et parfaitement synchronisée, l'extraction des données se fait après le filtre passe-bas du signal I. De plus, l'analyse montre que lorsque la boucle PLL est « barrée », le signal *Q_filtré* est nul.

Il faut maintenant comprendre l'effet d'une erreur de phase sur le signal d'entrée BPSK.

Le signal BPSK est alors définit par : $BPSK = DATA(t)\cos(\omega_c t + \theta)$ où θ est l'erreur de phase.

Le produit du signal BPSK avec erreur de phase par la composante en quadrature correspond à :

$$I = \pm\frac{1}{2}\left[ \cdot\cos\left( \left( \omega_{VCO} - \omega_{BPSK} \right) t + \theta \right) + \cos\left( \left( \omega_{VCO} + \omega_{BPSK} \right) t + \vartheta \right) \right] \quad \textbf{(1.7)}$$

où *DATA(t)* est remplacé par ±1.
Une fois filtré, on obtient :

$$I\_filtré = \pm\frac{1}{2} \cdot \cos\left( \left( \omega_{VCO} - \omega_{BPSK} \right) t + \theta \right) \quad \textbf{(1.8)}$$

La composante en quadrature correspond à :

$$Q = \pm\frac{1}{2}\left[ \sin\left( \left( \omega_{VCO} - \omega_{BPSK} \right) t + \theta \right) + \sin\left( \left( \omega_{VCO} + \omega_{BPSK} \right) t + \vartheta \right) \right] \quad \textbf{(1.9)}$$

Une fois filtré, on obtient :

$$Q\_filtré = \pm\frac{1}{2}\cdot\sin\left(\left(\omega_{VCO}-\omega_{BPSK}\right)t+\theta\right) \quad \textbf{(1.10)}$$

En utilisant un discriminateur qui fait la multiplication des signaux *I_filtré* et *Q_filtré* on obtient en sortie du discriminateur :

$$sortie\_discr = -\frac{1}{8}\sin\left(2\left(\left(\omega_{VCO}-\omega_{BPSK}\right)t+\theta\right)\right) \quad \textbf{(1.11)}$$

On constate donc que la boucle de Costas nous permet d'ajuster simultanément l'erreur de fréquence $\omega_{VCO}$ - $\omega_{BPSK}$ et l'erreur de phase, $\theta$.

**Boucle de Costas appliqué à la PLL**

Le schéma équivalent de la boucle PLL linéarisée est :

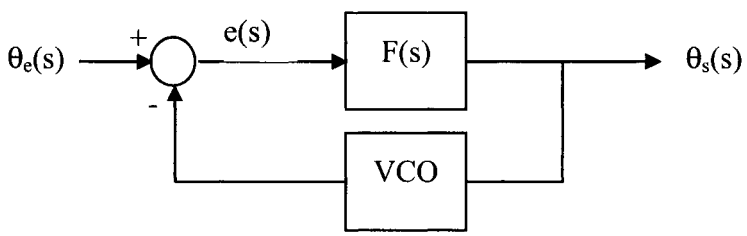



Figure 81 Schéma équivalent de la boucle PLL linéarisée

La fonction de transfert du filtre de boucle est F(s). Le VCO peut être vu comme un intégrateur de phase. Sa fonction de transfert dans le domaine Laplace est donc $VCO(s)=\frac{1}{s}$.

**Analyse de la boucle d'ordre 1**

Le filtre de boucle est dans ce cas un filtre d'ordre 0. On a $F(s)=\omega_0$.

Les équations caractéristiques sont :

$$\theta_s(s) = e(s) * \omega_0 \tag{1.12}$$

$$e(s) = \theta_e(s) - e(s) * \omega_0 * \frac{1}{s} \tag{1.13}$$

De ces deux équations on obtient,

La fonction de transfert du système :

$$H(s) = \frac{\theta_s(s)}{\theta_e(s)} = \frac{\omega_0 s}{s+\omega_0} \tag{1.14}$$

La fonction de transfert de l'erreur :

$$E(s) = \frac{e(s)}{\theta_e(s)} = \frac{s}{s+\omega_0} \tag{1.15}$$

On étudie l'erreur en régime permanent de la boucle d'ordre 1 soumise à un saut de phase, une rampe de phase et une rampe de fréquence.

Selon le théorème de la valeur finale, l'erreur dans le domaine temporel est donné par :

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0}\{s * e(s)\} \tag{1.16}$$

Pour un saut de phase $\theta_e$, on obtient :

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0}\left\{ s * \frac{s}{s+\omega_0} * \frac{\theta_e}{s} \right\} = 0 \qquad \textbf{(1.16)}$$

Pour un saut de fréquence $\Delta\omega_e$, on obtient :

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0}\left\{ s * \frac{s}{s+\omega_0} * \frac{\Delta\omega_e}{s^2} \right\} = \frac{\Delta\omega_e}{\omega_0} \qquad \textbf{(1.17)}$$

Pour une rampe de fréquence, on obtient :

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0}\left\{ s * \frac{s}{s+\omega_0} * \frac{R}{s^3} \right\} = \infty \qquad \textbf{(1.18)}$$

On dérive la réponse transitoire de e(t) à partir de la fonction de transfert e(s). On utilise la transformée de Laplace inverse : $e(t)=L^{-1}\{e(s)\}$.

La transformée de Laplace inverse est obtenue en utilisant l'environnement symbolique du logiciel Matlab On obtient pour un saut de phase :

$$e(t) = L^{-1}\left\{ \frac{s}{s+\omega_0} * \frac{\theta_e}{s} \right\} = \theta_e * e^{-\omega_0 t} \qquad \textbf{(1.19)}$$

Pour un saut de fréquence :

$$e(t) = L^{-1}\left\{ \frac{s}{s+\omega_0} * \frac{\Delta\omega_e}{s^2} \right\} = \frac{\Delta\omega_e}{\omega_0} * \left(1 - e^{-\omega_0 t}\right) \qquad \textbf{(1.20)}$$

Pour une rampe de fréquence :

$$e(t) = L^{-1}\left\{ \frac{s}{s+\omega_0} * \frac{R}{s^3} \right\} = \frac{R}{\omega_0^2} * \left(\omega_0 t - 1 + e^{-\omega_0 t}\right) \qquad \textbf{(1.21)}$$

La figure suivante montre la réponse de la boucle d'ordre 1 à différentes perturbations. Pour générer cette figure, on utilise les valeurs suivantes :

- Bande de boucle $B_n$ = 10 Hz
- $\omega_0$ = 4*Bn = 40 Hz
- Saut de phase $\theta_e = \pi/2$ rad
- Saut de fréquence $\Delta\omega_e$ = 5 Hz
- Rame de fréquence = 300 Hz/s

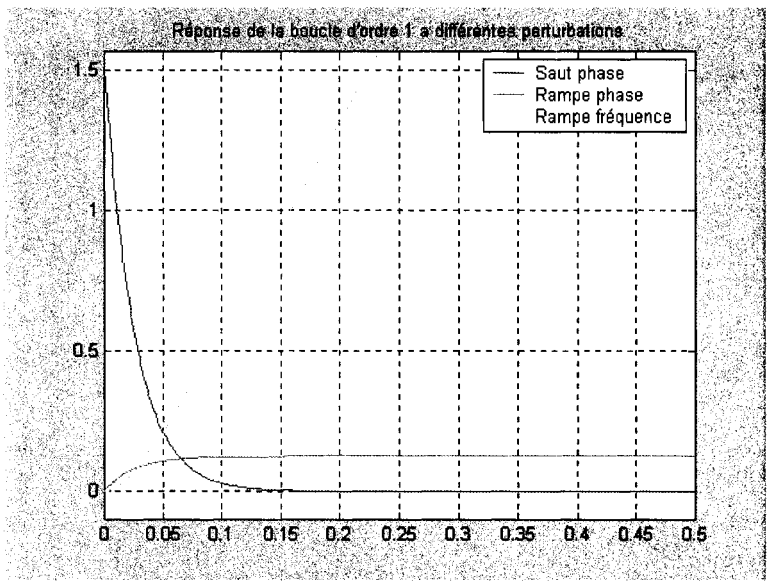



Figure 82 Réponse de la boucle PLL d'ordre 1

On remarque pour le saut de phase, l'erreur en régime permanent est de 0. Pour le saut de fréquence on obtient 0.125, ce qui correspond, comme on l'a trouvé par l'intermédiaire du théorème de la valeur finale, à $\frac{\Delta\omega_e}{\omega_0}=\frac{5}{40}=0.125$. Pour la rampe de fréquence, on obtient tel que prévue une erreur infinie.

**Analyse de la boucle d'ordre 2**

Le filtre de boucle est dans ce cas un filtre d'ordre 1. On a donc

$$F(s) = \frac{\omega_0}{s}\left(\omega_0 + a_2 s\right) \tag{1.22}$$

Les équations caractéristiques sont :

$$\theta_s(s) = e(s) * \frac{\omega_0}{s}\left(\omega_0 + a_2 s\right) \tag{1.23}$$

$$e(s) = \theta_e(s) - e(s) * \frac{\omega_0}{s}\left(\omega_0 + a_2 s\right) * \frac{1}{s} \tag{1.24}$$

De ces deux équations (3.24 et 3.25), on obtient :
La fonction de transfert du système :

$$H(s) = \frac{\theta_s(s)}{\theta_e(s)} = \frac{\omega_0 s\left(a_2 s + \omega_0\right)}{s^2 + a_2\omega_0 s + \omega_0^2} \tag{1.25}$$

La fonction de transfert de l'erreur :

$$E(s) = \frac{e(s)}{\theta_e(s)} = \frac{s^2}{s^2 + a_2\omega_0 s + \omega_0^2} \tag{1.26}$$

On étudie l'erreur en régime permanent de la boucle d'ordre 2 soumise à un saut de phase, une rampe de phase et une rampe de fréquence.

Selon le théorème de la valeur finale, l'erreur dans le domaine temporel est donné par :

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0}\left\{s * e(s)\right\} \tag{1.27}$$

Pour un saut de phase $\theta_e$, on obtient :

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0}\left\{ s * \frac{s^3}{s^3 + b_3\omega_0 s^2 + a_3\omega_0^2 s + \omega^3} * \frac{\theta_e}{s} \right\} = 0 \qquad \textbf{(1.28)}$$

Pour un saut de fréquence $\Delta\omega_e$, on obtient :

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0}\left\{ s * \frac{s^3}{s^3 + b_3\omega_0 s^2 + a_3\omega_0^2 s + \omega_0^3} * \frac{\Delta\omega_e}{s^2} \right\} = 0 \qquad \textbf{(1.29)}$$

Pour une rampe de fréquence, on obtient :

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0}\left\{ s * \frac{s^3}{s^3 + b_3\omega_0 s^2 + a_3\omega_0^2 s + \omega_0^3} * \frac{R}{s^3} \right\} = \frac{R}{\omega_0^2} \qquad \textbf{(1.30)}$$

Le filtre de boucle est dans ce cas un filtre d'ordre 2. On a donc

$$F(s) = \frac{\omega_0}{s^2} + \frac{a_3\omega_0^2}{s} + b_3\omega_0 \qquad \textbf{(1.31)}$$

Les équations caractéristiques sont :

$$\theta_s(s) = e(s) * \left( \frac{\omega_0}{s^2} + \frac{a_3\omega_0^2}{s} + b_3\omega_0 \right) \qquad \textbf{(1.32)}$$

$$e(s) = \theta_e(s) - e(s) * \left( \frac{\omega_0}{s^2} + \frac{a_3\omega_0^2}{s} + b_3\omega_0 \right) * \frac{1}{s} \qquad \textbf{(1.33)}$$

De ces deux équations (3.33 et 3.34) on obtient la fonction de transfert du système :

$$H(s) = \frac{\theta_s(s)}{\theta_e(s)} = \frac{\omega_0 s\left(b_3 s^2 + a_3\omega_0 s + \omega_0^2\right)}{s^3 + b_3\omega_0 s^2 + a_3\omega_0^2 s + \omega_0^3} \qquad \textbf{(1.34)}$$

La fonction de transfert de l'erreur :

$$E(s) = \frac{e(s)}{\theta_e(s)} = \frac{s^3}{s^3 + b_3\omega_0 s^2 + a_3\omega_0^2 s + \omega_0^3} \tag{1.35}$$

On étudie l'erreur en régime permanent de la boucle d'ordre 3 soumise à un saut de phase, une rampe de phase, une rampe de fréquence et à un jerk.

Selon le théorème de la valeur finale, l'erreur dans le domaine temporel est donné par :

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0}\{s * e(s)\} \tag{1.36}$$

Pour un saut de phase $\theta_e$, on obtient :

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0}\left\{ s * \frac{s^3}{s^3 + b_3\omega_0 s^2 + a_3\omega_0^2 s + \omega^3} * \frac{\theta_e}{s} \right\} = 0 \tag{1.37}$$

Pour un saut de fréquence $\Delta\omega_e$, on obtient :

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0}\left\{ s * \frac{s^3}{s^3 + b_3\omega_0 s^2 + a_3\omega_0^2 s + \omega_0^3} * \frac{\Delta\omega_e}{s^2} \right\} = 0 \tag{1.38}$$

Pour une rampe de fréquence, on obtient :

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0}\left\{ s * \frac{s^3}{s^3 + b_3\omega_0 s^2 + a_3\omega_0^2 s + \omega_0^3} * \frac{R}{s^3} \right\} = 0 \tag{1.39}$$

Pour un jerk, on obtient :

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0}\left\{ s * \frac{s^3}{s^3 + b_3\omega_0 s^2 + a_3\omega_0^2 s + \omega_0^3} * \frac{J}{s^4} \right\} = \frac{J}{\omega_0^3} \tag{1.40}$$

**Erreur en fonction de l'ordre de la boucle et du type de perturbation**

Le tableau suivant montre l'erreur en régime permanent des boucles d'ordre 1,2 et 3 en fonction de différentes perturbations en entrée : saut de phase, rampe de phase, rampe de fréquence, ainsi qu'un jerk.

Tableau XIX

Erreur en fonction de l'ordre de la boucle et du type de perturbation

| | **Erreur Ordre 1** | **Erreur Ordre 2** | **Erreur Ordre 3** |
|---|---|---|---|
| **Saut phase** | 0 | 0 | 0 |
| **Saut fréquence** | $\frac{\Delta\omega_e}{\omega_0}$ | 0 | 0 |
| **Rampe de fréquence** | $\infty$ | $\frac{R}{\omega_0^2}$ | 0 |
| **« Jerk »** | $\infty$ | $\infty$ | $\frac{J}{\omega_0^3}$ |

**Fonction de transfert de la boucle PLL**

On utilise les résultats obtenus à la section précédente pour visualiser et analyser les fonctions de transferts H(s) et E(s) en fonction de l'ordre de la PLL.

On reprend le schéma équivalent de la boucle de phase et on définit le signal $\theta_\pi(s)$, à la sortie du VCO.

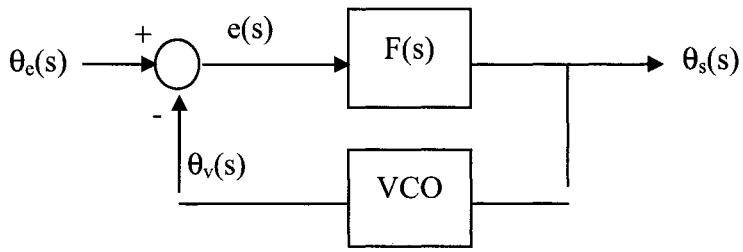



Figure 83 Schéma équivalent de la boucle PLL linéarisée avec $\theta_\pi(s)$

**Étude générale boucle d'ordre 1**

Pour la boucle d'ordre 1, on a :

- La fonction de transfert du système :

$$G(s) = \frac{\theta_v(s)}{\theta_e(s)} = \frac{\omega_0}{s + \omega_0} \tag{1.41}$$

- La fonction de transfert de l'erreur :

$$E(s) = \frac{e(s)}{\theta_e(s)} = \frac{s}{s + \omega_0} \tag{1.42}$$

**Étude générale boucle d'ordre 2**

Pour la boucle d'ordre 2, on a :

- La fonction de transfert du système :

$$G(s)=\frac{\theta_v(s)}{\theta_e(s)}=\frac{\omega_0\left(a_2 s+\omega_0\right)}{s^2+a_2\omega_0 s+\omega_0^2} \tag{1.43}$$

- La fonction de transfert de l'erreur :

$$E(s)=\frac{e(s)}{\theta_e(s)}=\frac{s^2}{s^2+a_2\omega_0 s+\omega_0^2} \tag{1.44}$$

**Étude générale boucle d'ordre 3**

Pour la boucle d'ordre 3, on a :

3. La fonction de transfert du système :

$$G(s)=\frac{\theta_v(s)}{\theta_e(s)}=\frac{\omega_0\left(b_3 s^2+a_3\omega_0 s+\omega_0^2\right)}{s^3+b_3\omega_0 s^2+a_3\omega_0^2 s+\omega_0^3} \tag{1.45}$$

4. La fonction de transfert de l'erreur :

$$E(s)=\frac{e(s)}{\theta_e(s)}=\frac{s^3}{s^3+b_3\omega_0 s^2+a_3\omega_0^2 s+\omega_0^3} \tag{1.46}$$

**Étude graphique de G(s) et de E(s)**

Les valeurs suggérées pour un ordre 1 par Kaplan **[1]** sont $\omega_0$ = 4*Bn = 4*10 = 40 Hz. Les valeurs suggérées pour un ordre 2 sont : $\omega_0$ = Bn/.53 = 10/.53 = 18.868 et $a_2$ = 1.414*$\omega_0$ = 26.679. Les valeurs suggérées pour un ordre 3 sont : $\omega_0$ = Bn/0.7845 = 10/.0.7845 = 12.747, $a_3$ = 1.1 et $b_3$ = 2.4. On utilise ces valeurs pour générer les 2 figures suivantes.

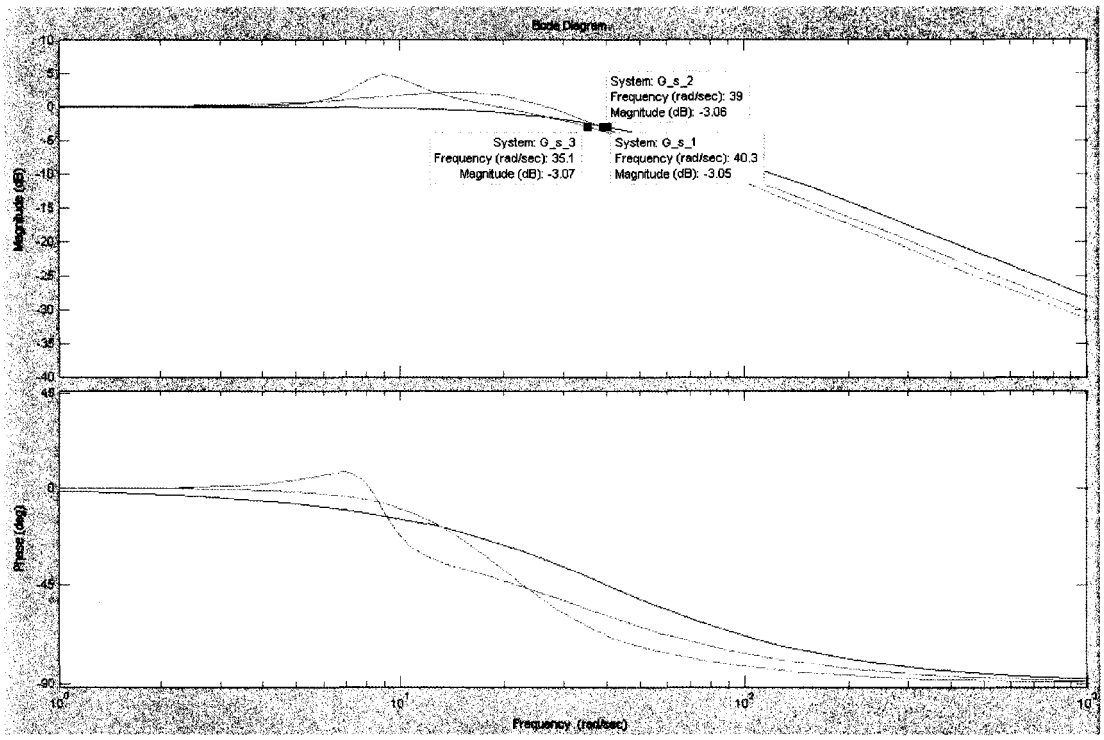



Figure 84 Réponse en fréquence de G(s), boucles d'ordre 1,2 et 3

Sur cette figure, il est difficile de différencier l'ordre de des fonctions de transfert car l'échelle en dB n'est pas assez grande. On voit par contre que la fréquence de coupure des systèmes d'ordre 1,2 et 3 sont très proches (40.3, 39 et 35.1 rad/s).

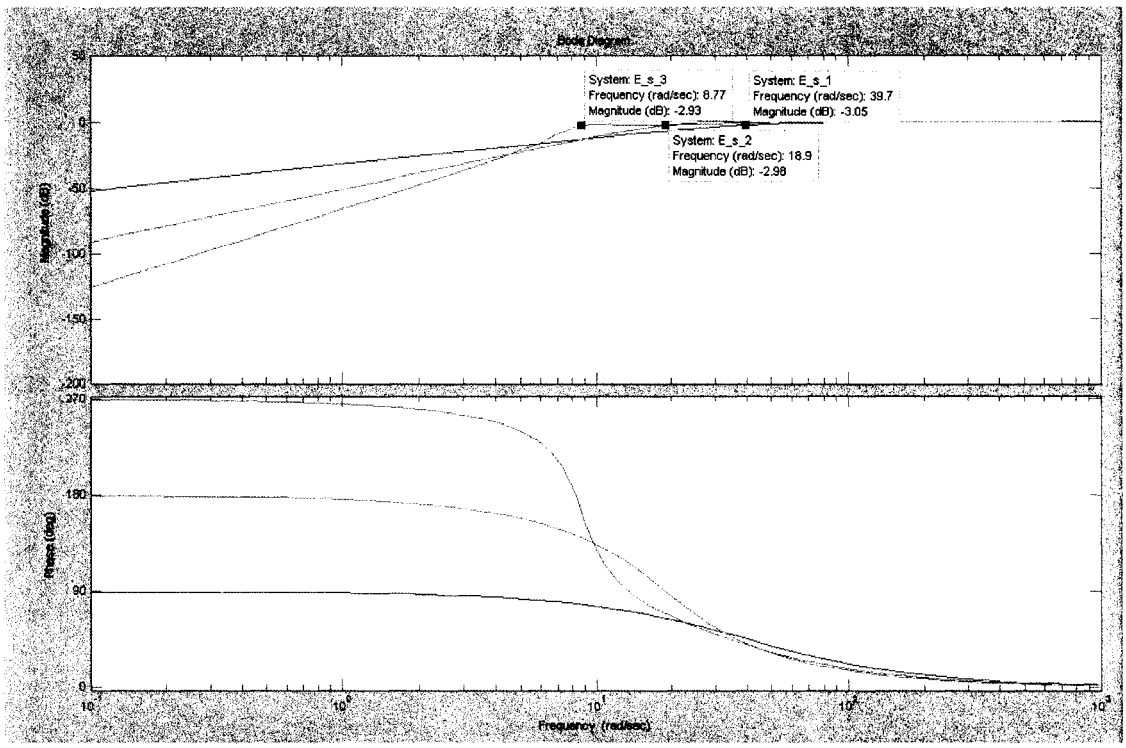



Figure 85 Réponse en fréquence de E(s), boucles d'ordre 1,2 et 3

Sur cette figure, on distingue plus facilement l'ordre de la fonction de transfert E(s). On voit que pour l'ordre 1, la pente d'atténuation est de 20 dB/octave, pour l'ordre 2, elle est de 40 dB/octave et pour l'ordre 3, on a une pente de 60 dB/octave. Les fréquences de coupure pour les ordres 1, 2 et 3 sont respectivement de 39.7, 18.9 et 8.77 rad/s.

# ANNEXE 2

# CARATÉRISTIQUES DES CARTES DE LA PLATEFORME DE DÉVELOPPEMENT

La carte modulaire d'acquisition de données se compose comme suit :

- 2 convertisseurs analogique-numérique de 14bits, à fréquence d'échantillonnage de 105 MHz (Analog Devices AD6645)
- 1 convertisseurs numérique-analogique à 2 ports de sortie de 16 bits, à fréquence d'échantillonnage de 400 MHz par interpolation (Analog Devices AD9777)
- 2 ports de communications à 20 MBytes par seconde
- Xilinx VIRTEX-II FPGA (XC2C1000-6) (Ce composant sert seulement à la gestion de la carte

La carte d'acquisition répond à nos caractéristiques. Tout d'abord, des convertisseurs à large bande à fréquence d'échantillonnage élevé. Comme le signal arrivera en fréquence intermédiaire de 2,046 MHz, les valeurs des convertisseurs nous garantissent amplement la stabilité de notre système. De plus, avec des ports de communication de 20Mbytes par seconde, la rapidité du transfert des données respecte nos normes. Un taux de 20MBytes correspond à 160 Mbits. Avec des convertisseurs analogique-numérique à 14 bits, le transfert de 11428571 données en une seconde s'effectue, donnant une fréquence d'échantillonnage et de transfert de 11,429 MHz, suffisant pour notre modèle.

Les deux cartes modulaires centrales, sont les cartes incluant le processeur numérique et le FPGA. Ces cartes devront traiter les échantillons envoyés par la carte d'acquisition.

Les caractéristiques de la carte modulaire avec DSP sont les suivantes :

- Un DSP de Texas Instruments TMS320C6416 à point fixe de 600 MHz
- 4800 MIPS (Mega Instructions Per Seconds)
- 2 bus de données à haute vitesse (50 MHz, 100 MHz ou 200 MHz) à 32 bits
- 8 MBytes de mémoire flash pour configuration

- 6 ports de communications de 20MBytes par seconde
- Xilinx Virtex-II XC2V1000-4 (Ce composant sert seulement à la gestion de la carte)

Outre les ports de communications qui ont exactement les mêmes caractéristiques que la carte précédente, le processeur numérique a plusieurs caractéristiques intéressantes. Tout d'abord, le fait qu'il puisse exécuter autant d'instruction par seconde nous satisfait au niveau de la rapidité. Comme le DSP se en charge des calculs plus complexes, il est important qu'il soit capable d'exécuter des plusieurs calculs dans un court laps de temps. De plus, la mémoire flash permet la configuration du DSP et sa capacité nous assure que notre modèle pourra y être contenu.

Les caractéristiques de la carte modulaire avec FPGA sont les suivantes :

- Un FPGA de Xilinx XC2V1000-4
- 1 millions de portes logiques reconfigurable
- 4 bus de données à haute vitesse (400 MBytes par seconde)
- 8 MBytes de mémoire flash pour configuration
- 6 ports de communications (à 32-bits) de 100 MBytes par seconde

**Caractéristiques des FPGA appliquées au SDN**

Dans le cadre du projet de récepteur GPS reprogrammable, nous avons identifié plus tôt les besoins techniques et mathématiques. Reprenons ces besoins que nous avons définis plus tôt :

- Fréquence d'échantillonnage élevé
- Convertisseur analogique-numérique à large bande
- Entrée analogique

- Grande capacité d'espace mémoire
- Processeur à haute vitesse
- Port série (pour communiquer avec l'ordinateur)

La conception d'un récepteur GPS demandera l'exploitation d'un langage de haut niveau tel le VHDL. Ce langage offre une souplesse de programmation intéressante qui s'adapte bien avec un outil de schématique tel que Simulink. Avec ces caractéristiques, l'utilisation d'un FPGA s'avère un choix logique. Premièrement, les FPGA ont un faible coût de production, ce qui cadre bien avec la méthodologie exprimée plus tôt. Deuxièmement, la vitesse d'exécution des programmes à l'intérieur du FPGA est plus élevée que dans le cas d'un DSP. Cette caractéristique est extrêmement importante puisque nous travaillons avec des signaux à haute fréquence et à haut débit. À l'intérieur du récepteur GPS, la boucle PLL, dont nous verrons la conception plus loin dans le texte, se doit de réagir rapidement et d'effectuer des calculs rapides pour ajuster son recouvrement de la porteuse.

Cependant ces avantages ont leur envers de la médaille. Le temps de conception est donc plus long puisque les algorithmes ne sont pas nécessairement modifiables rapidement. Le VHDL étant un langage de haut niveau, les changements à effectuer dans le code peuvent être complexe et engendrés des complications.

**Caractéristiques des DSP appliquées au projet SDN**

À l'inverse du FPGA, la programmation des DSP est plus simple, puisqu'ils utilisent des langages de bas niveau tel l'Assembleur. Ceci permet une correction plus rapide des erreurs, mais cette caractéristique n'est pas recherchée pour notre projet. Par contre, les DSP peuvent calculer des algorithmes mathématiques plus complexes. Souvent vu comme des supercalculateurs, les DSP ont un rapport rapidité de calcul versus complexité du calcul plus élevé que les FPGA. Comme nous le verrons dans le chapitre

4, cette caractéristique est essentielle pour le récepteur GPS. Dans le calcul des erreurs de phase et de code, la complexité des calculs apparaît comme un obstacle majeur au niveau de la précision.

Au niveau de la gestion du transport des données, le DSP s'avèrera utile puisqu'il permettra de rediriger les signaux acquis vers les processeurs. Étant relié directement avec les convertisseurs, il permettra de gérer certains signaux de contrôle pour les bus de la carte-mère.

# BIBLIOGRAPHIE


[1] Kaplan, E., *"Understanding GPS:Principles and Applications"*, 3rd edition, London, Artech House, 1996.

[2] Rabaeijs, A., Grosso, D., Huang, X., Qi, D.,*"GPS receiver prototype for integration into system-on-chip"*, IEEE Transactions on Consumer Electronics, Vol 49, No. 1, February 2003.

[3] Turney, R. D., Reza, A. M., Delva, J. G. R. Issler, JL., Martin, J.C., Erhard P., Lucas-Rodriguez, R. , Pratt, T., *"FPGA implementation of adaptive temporal kalman filter for real time video filtering"*, IEEE Publication 1999.

[4] http://ocw.mit.edu/OcwWeb/Earth--Atmospheric--and-Planetary-Sciences/12-215Modern-NavigationFall2002/CourseHome/

[5] Ilie, I., Landry, R.Jr., *"Simulation of GPS and Galileo architectures for anti-jamming and multipath analysis"*, École de technologie superieure, 2003.

[6] Avrunin, G., Corbett, J. C., Dillon, L. K., *"Analyzing Partially-Implemented Real-time systems"*, IEEE Transactions on software engineering, August 1998

[7] Attri, S., Sohi, B.S., Chopra, Y.C., *"Efficient design of application specific DSP cores using FPGAs"*, 2001

[8] Kibe, S.V., Shridara, K.A., Jayalalitha, M., *"Software-based GIC/GNSS compatible GPS receiver architecture using TMS320C030 DSP processor"*, Satellite Systems for Mobile Communications and Navigation Conference, Publication no 424, 13-15th May 1996

[9] Software Defined Radio Forum, www.sdrforum.org.

[10] Berkeley Design Technology, Inc., *"Comparing FPGAs and DSPs for Embedded Signal Processing"*, Presentation, October 2002

[11] Martina, M., Molino, A., Vacca, F., *"FPGA System-on-chip soft IP design: A reconfigurable DSP"*, CERCOM, IEEE publication, 2002.

[12] Ownby, M., Mahmoud, W.H., *"A design methodology for implementing DSP with Xilinx® System Genrator for Matlab®"*, IEEE Publication, 2002.

[13] Xilinx Corp., *"Xilinx System Generator for Simulink®"*, Version 3.1.

[14] Sundance Inc.., www.sundance.com (Consulté le 24 février 2004)

[15] Texas Instruments Inc.., www.ti.com (Consulté le 24 février 2004)

[16] Cooling, J., *"Software Engineering for Real-Time Systems"*, Addison Wesley, 2003, 800 pages.

[17] Keyes, J., *"Software Engineering Handbook"*, Auerbach, 2003, 874 pages.

[18] Karner, J. *"GPS-A Global Utility"*. Paper presented at the The European Navigation Conference, 2002 May 27-30,Copenhagen, Denmark.

[19] Shaw, M, *"GPS Status"*. Paper presented at the The European Navigation Conference, (2002 May 27-30, Copenhagen, Denmark

[20] www.encyclopedia.lockergnome.com/s/b/Software_radio (Consulté le 10 mai 2005)

[21] en.wikipedia.org/Real-time (Consulté le 1 mars 2006)

[22] en.wikipedia.org/wiki/GLONASS (Consulté le 30 mars 2006)

[23] Dye, S. *"GLONASS – The Russian GPS"*, Satellites Times, November/December 1996

[24] en.wikipedia.org/wiki/Beidou_navigation_system (Consulté le 30 mars 2006)

[25] www.globalsecurity.org/space/world/china/beidou (Consulté le 30 mars 2006)

[26] en.wikipedia.org/wiki/Galileo_navigation_system Consulté le 30 mars 2006

[27] ONIDI, O., *"GALILEO is launch"*, European Commission.

[28] Rumsfeld, D., Mineta N.,*"2001 Federal Radionavigation plan"*, Departement of Transportation and Departement of Defense, Mars 2002

[29] European Space Agency, *"Galileo le programme européen de navigation par satellite"*, Division des publications de l'ESA, Juillet 2002

[30] Lim, I.-G., Kim, W.-W., *"A Numerically Controlled Oscillator with a Fine Phase Tuner and a Rounding Processor"*, ERTI Journal, Vol. 26, December 2004.