Projet CLEAR : Horloge composite numérique polyvalente : Asservissement en fréquence multisources

Alexis Benigni

▶ **To cite this version:**

Alexis Benigni. Projet CLEAR : Horloge composite numérique polyvalente : Asservissement en fréquence multisources. Traitement du signal et de l'image [eess.SP]. Université Bourgogne Franche-Comté, 2018. Français. NNT : 2018UBFCD044 . tel-02144042

HAL Id: tel-02144042
https://theses.hal.science/tel-02144042

Submitted on 29 May 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre : Année 2018

RAPPORT DE THÈSE CIFRE

PRÉSENTÉE À

BESANÇON

POUR OBTENIR LE GRADE DE

DOCTEUR
DE L'UNIVERSITE DE BOURGOGNE FRANCHE-COMTE
SPÉCIALITÉ PHYSIQUE

Projet CLEAR
Horloge composite numérique polyvalente
Asservissement en fréquence multisources

Par

Alexis Benigni

Soutenue le 1er juin 2018 devant la commission d'examen :

Président :	J. ACHKAR	Responsable Équipe Métrologie du temps, Observatoire de Paris
Rapporteurs :	F. HAMELIN	Professeur Université de Lorraine, Centre de Recherche en Automatique de Nancy, Vandoeuvre
Examinateurs :	F. VERNOTTE	Professeur Université de Franche comté, Institut UTINAM, Besançon
	E. RUBIOLA	Professeur Université de Franche comté, Institut FEMTO-ST, Département Temps-Fréquence, Besançon
	J.C. POUYTES	Directeur TimeLink Microsystemes, Toulouse

Remerciements

Les acteurs et contributeurs à ma thèse CIFRE ont été nombreux et je prie toute personne lisant ces lignes de bien vouloir m'excuser si son nom n'est pas cité ici.

Pour commencer par le commencement je souhaite remercier mes parents de m'avoir légué leur esprit scientifique et de m'avoir porté, voire supporté, ces 30 dernières années.

Un grand merci également au initiateurs du projet CLEAR : M. François VERNOTTE, mon directeur de thèse à l'Observatoire de Besançon ainsi que M. Jean-Claude POUYTES, directeur de l'entreprise Microsystèmes (anciennement TimeLink) à Toulouse de m'avoir fait confiance pour mener à bien cette thèse, sans oublier M. Enrico RUBIOLA mon co-directeur de thèse et directeur adjoint du département temps-fréquence de l'institut FEMTO-ST à Besançon.

Je remercie toute l'équipe du service temps-fréquence de l'observatoire de Besançon, en particulier M. Nicolas GAUTHEROT et M. Eric MEYER, ingénieurs de recherche, pour leur soutien technique très précieux et constant durant ma thèse. M. François MEYER, directeur du service, pour son expertise du domaine et son briquet. M. Hervé LOCATELLI pour ses conceptions et développements de circuits électroniques indispensable à l'élaboration de mon système. M. Cédric PLANTARD pour avoir posé les bases de mon travail sur l'horloge composite, pour sa bonne humeur et pour avoir donné un foyer au chat. M. Timothée ACCADIA pour avoir partagé ces 3 années et sa réserve secrète. M^{me} Fabienne GRAND-LOCATELLI pour le soutien logistique et les petits pics.

Je tiens aussi à remercier l'équipe du service informatique, particulièrement M. Sékou DIAKITE pour avoir résolu mes derniers problèmes de glitchs et pour avoir sondé toute l'équipe de foot, M. Kévin VAN-KEULEN pour sa disponibilité et ses débogages.

Merci à Messieurs Pierre-Yves Bourgeois et Gwenhael GOAVEC-MEROU de FEMTO-ST pour leur aide sur les systèmes Linux embarqués et pour avoir partagé leur (trop ?) grande connaissance de l'électronique numérique et autres FPGA. M. Gonzalo CABODEVILA pour ses connaissances en automatique.

Merci à l'équipe de Microsystème pour son accueil au sein de l'entreprise et leur conseils. M. Christophe GRAULLE pour son aide sur les modules Linux, M. Gabriel TOUZOT pour ses conseils éclairés en matière de programmation FPGA.

Et pour finir, merci à tous mes amis que j'ai côtoyé de près ou de loin à Besançon : M. José FERNANDEZ-TRINCADO pour avoir autant pulsé, M. Guillaume NASELLO pour ses leçons de Français, M^{me} Gaëlle LAPORTE pour son expertise culinaire, Mr André MARTINS pour sa bonne foi, M. Mohamad ALI-DIB mon coloc de bureau, Mr Daniel BAGUET pour avoir complété l'équipe de foot et Mr Bruno BELLOMO pour son talent footballistique.

Table des matières

I Introduction **11**

1 Les sources de temps et de fréquence **15**
1.1 Les horloges anciennes . . . 15
1.1.1 Horloges solaires . . . 15
1.1.2 Horloges mécaniques . . . 16
1.2 Les horloges actuelles . . . 18
1.2.1 Oscillateurs électroniques . . . 18
1.2.2 Les horloges atomiques . . . 22
1.2.3 Oscillateur cryogénique à saphir . . . 31
1.3 Les horloges de demain . . . 32

2 Le transfert des signaux métrologiques **35**
2.1 Les transferts par média physique . . . 35
2.1.1 Câble coaxial . . . 35
2.1.2 Fibre optique . . . 36
2.1.3 Synchronisation par réseau . . . 37
2.2 Les transferts sans média physique . . . 39
2.2.1 Ondes radio-fréquences terrestres . . . 39
2.2.2 Micro-ondes par satellite . . . 39
2.3 Les échelles de temps . . . 41
2.3.1 Temps atomique international . . . 41
2.3.2 UTC . . . 41

3 Les bruits des oscillateurs et leur caractérisation **43**
3.1 Prérequis . . . 43
3.2 Modélisation d'un oscillateur réel . . . 44
3.2.1 Fluctuations de phase et de fréquence dans le domaine temporel . . . 46
3.2.2 Fluctuations de phase et de fréquence dans le domaine fréquentiel . . . 48
3.3 Modèle de bruits en lois de puissance . . . 49
3.3.1 Méthodes pratiques d'analyse du bruit . . . 50

4 Principe d'une horloge composite **55**
4.1 Principe général . . . 55
4.2 Ancienne horloge . . . 56
4.2.1 Principe . . . 56
4.2.2 Limitations de l'ancienne horloge . . . 58
4.3 Objectifs de la nouvelle horloge composite . . . 58

II Hardware **61**

5 Cartes de développement et prototypes **63**
5.1 Présentation des puces Zynq 63
5.2 RedPitaya 66
5.2.1 Échantillonnage rapide 66
5.2.2 SINITER 67
5.3 ZedBoard 74
5.3.1 Acquisition directe 74
5.3.2 Cartes d'interfaces 76
5.3.3 Pilotage du VCO 76
5.4 MicroZed 79
5.4.1 VCO 80

6 Code FPGA **83**
6.1 Versions intermédiaires 83
6.1.1 Quelle fonction implémenter ? 83
6.1.2 Sources, référence et compteur 84
6.1.3 Premières versions du code 84
6.2 Améliorations de l'intervallomètre 85
6.2.1 Domaine d'horloge et synchronisations 86
6.2.2 Fréquence maximale du domaine d'horloge 86
6.2.3 Interruptions 86
6.2.4 Registres temporaires et finaux 87
6.2.5 Sécurisation software 87
6.3 Version finale 88
6.3.1 Diviseur de fréquence 90
6.3.2 Multiplicateur de fréquence - MMCM 91
6.3.3 Sélecteur de référence 91
6.3.4 Compteur binaire 24 bits 91
6.3.5 Synchronisations 91
6.3.6 Détection de fronts 92
6.3.7 Registres 92
6.3.8 Acquisitions 92

III Software **93**

7 Mise en forme des mesures **95**
7.1 Écart brut des compteurs 95
7.2 Affinage et contrôle des écarts bruts 95
7.2.1 Principe 95
7.2.2 Régression linéaire 96
7.3 Écarts temporels 97
7.4 Déduction des écarts de fréquence 98

8 Algorithmes d'asservissement **99**
8.1 Filtre de Kalman 99
8.1.1 Principe et description 100
8.1.2 Le déroulement du calcul 102

8.1.3 Paramétrage . . . 103
8.1.4 Simulations . . . 104

IV Résultats 117

9 Performances hardware 121
9.1 Seuil court et Moyen terme . . . 122
9.2 Long terme . . . 123

10 Comparaisons software 125
10.1 Prototype ZedBoard . . . 125
10.1.1 Simple source . . . 125
10.1.2 Sources hétérogènes . . . 126
10.1.3 Sources homogènes . . . 128
10.2 Prototype MicroZed . . . 130
10.2.1 Mesures préliminaires . . . 130
10.2.2 Simple source . . . 130
10.2.3 Sources hétérogènes . . . 131
10.3 Conclusions . . . 131

V Conclusions et perspectives 133

11 Conclusions 135
11.1 Hardware . . . 135
11.2 Software . . . 135
11.2.1 Algorithme d'asservissement . . . 135

12 Perspectives 137
12.1 Hardware . . . 137
12.1.1 Composants . . . 137
12.1.2 FPGA . . . 137
12.2 Software . . . 138
12.2.1 Amélioration du Kalman . . . 138
12.2.2 Compensation thermique . . . 138
12.2.3 Autres algorithmes d'asservissement . . . 138

VI Annexes 139

13 Création de la distribution Linux 141
13.1 Création des fichiers ps7_init_gpl.c et ps7_init_gpl.h . . . 141
13.2 Le noyau Linux et Buildroot . . . 145
13.3 La carte SD . . . 148
13.4 Montage du serveur NFS . . . 149
13.4.1 Configuration serveur . . . 150
13.4.2 Configuration client . . . 150

14 Codes **153**
14.1 Codes FPGA verilog . . . 154
14.1.1 Sélecteur de référence . . . 154
14.1.2 Diviseur . . . 155
14.1.3 Détecteur de front . . . 157
14.1.4 Registres . . . 158
14.1.5 Synchronisation . . . 159
14.2 Code C . . . 160
14.2.1 Driver d'interruptions . . . 160
14.2.2 Application . . . 163
14.3 Codes Octave pour fichiers de log . . . 185

15 Mesures **191**
15.1 SINITER sur RedPitaya . . . 191

16 Schémas électroniques **201**
16.1 Carte interface FMC - LVDS . . . 201
16.2 Carte CNA - SPI . . . 204
16.3 Montage MicroZed . . . 207

17 Fiches techniques **213**
17.1 Ampli Op AD8675 . . . 213
17.2 Convertisseur Numérique/Analogique AD5791 . . . 227
17.3 Convertisseurs logiques (ZCD) LTC6957-2 . . . 257
17.4 VCO AR2508 OCXO 10MHz . . . 294

18 Glossaire **297**
18.1 Les termes techniques de l'électronique numérique . . . 297
18.2 Les termes généraux du domaine temps-fréquence . . . 298
18.3 Les laboratoires, écoles, tutelles . . . 298

Première partie

Introduction

Le quête perpétuelle d'amélioration de la qualité (exactitude, stabilité et transfert) des signaux métrologiques est motivée par des applications bien concrètes (hormis l'établissement d'échelles de temps type UTC).

En effet, on peut légitimement se demander quel est l'intérêt de mettre en jeu autant de moyens humains et techniques pour gagner quelques picosecondes sur un signal PPS ou d'atteindre des stabilités de fréquence relatives de quelques 10^{-16}.

L'exemple le plus répandu est l'amélioration de la localisation (par GPS, en attendant sa version européenne GALILEO). En effet, son principe repose sur l'estimation de la distance séparant un récepteur d'au moins 3 émetteurs (triangulation) et cette distance est déduite par la formule simple $d = v \times t$. La vitesse de propagation des signaux dans l'atmosphère étant bien connu (299 792 458 m/s) la précision de l'estimation de la distance dépend directement de la précision de la mesure du temps de propagation. Une erreur d'une microseconde entraînerait une erreur de 300 mètres sur la localisation !

Une autre application quotidienne concerne les télécommunications. Les système de transfert de données ont besoin de signaux finement synchronisés afin de pouvoir récupérer efficacement les informations. Ainsi un gain de stabilité (bruit de phase, court terme) peut permettre une augmentation des cadences de fonctionnement en réduisant les phénomènes d'interférences parasites et par conséquent améliorer la qualité et la quantité des données transmises à chaque instant.

Il existe encore d'autres applications où l'exactitude et la stabilité d'un signal est critique, le laser méga-joule du CEA ou la synchronisation des équipements joue un rôle fondamental, notamment au niveau de l'activation des lasers de confinement. C'est au vu de ces enjeux que la recherche dans le domaine temps-fréquence prend tout son sens.

Le Laboratoire Temps Fréquence de Besançon (LTFB) est affilié au Laboratoire National de métrologie et d'Essais. Il regroupe les activités temps-fréquence de l'Université de Franche-Comté, UTINAM et FEMTO-ST au sein d'une même entité.

Il contribue au temps atomique français avec ses 3 étalons atomiques à jet de césium et plusieurs récepteurs GPS. Ses missions sont multiples : de l'étalonnage en temps et fréquence (accréditation COFRAC) et des mesures de bruit de phase.

La récente mise en place de l'équipement d'excellence Osc-IMP (Oscillator Instability Measurement Platform) a encore fait évoluer le champ d'expertise du LTFB avec, entre autres, la mise en place d'une station de transfert de temps par satellite 2-way, l'élaboration d'un banc de bruit de phase numérique et le développement des étalons de fréquence de demain (oscillateur cryogénique à saphir). L'ensemble des compétences et des outils disponible au LTFB en font un laboratoire connu et reconnu dans le monde entier.

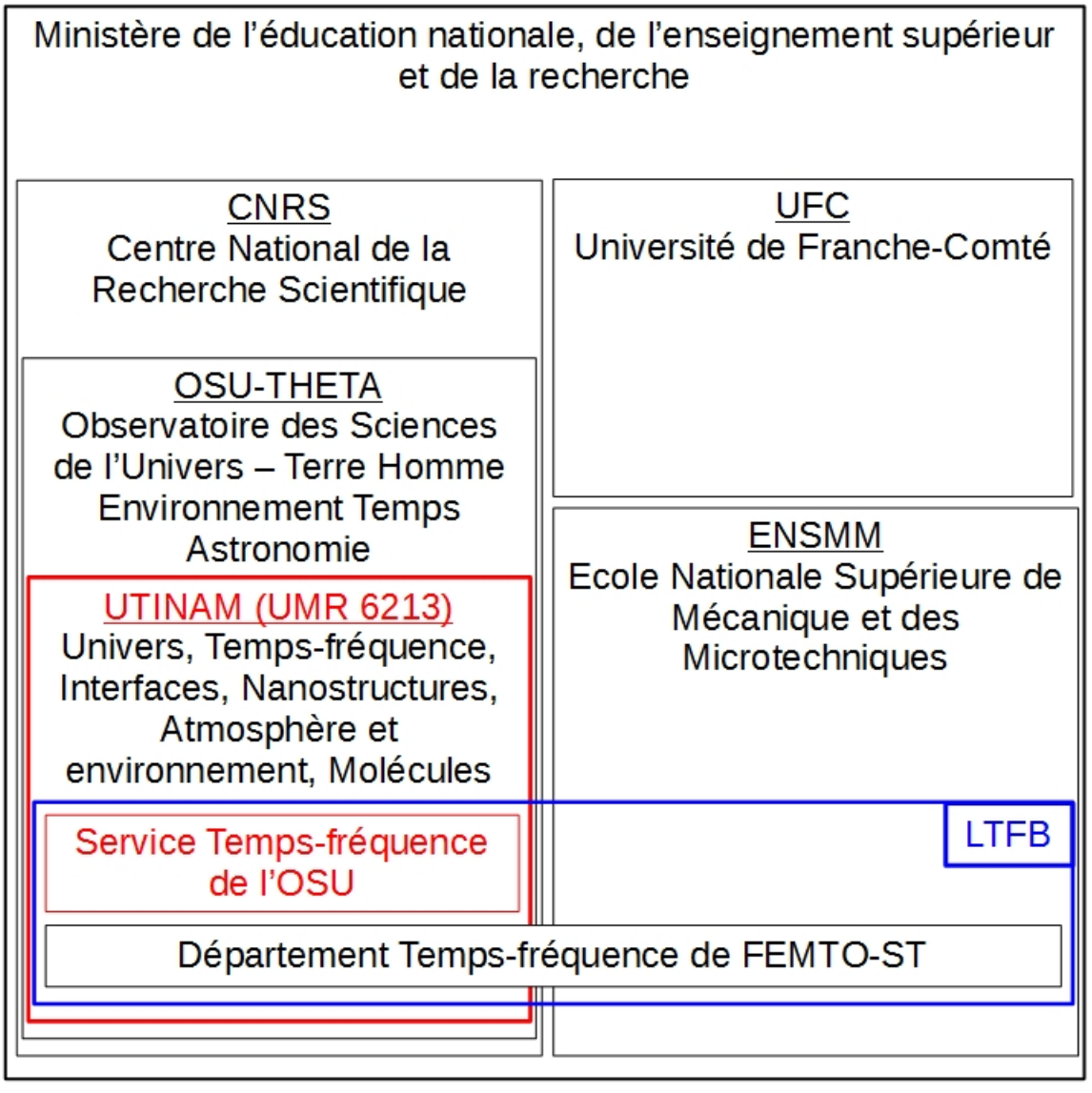


FIGURE 1 – Organigramme situant le LTFB au sein des différentes entités

Chapitre 1

Les sources de temps et de fréquence

Généralement une horloge se présente sous la forme d'un oscillateur créant un phénomène répétitif plus ou moins régulier auquel on associe un procédé de comptage.

En prenant l'exemple d'une horloge solaire, l'oscillateur est le mouvement du soleil par rapport à un repère fixé à la surface de la terre, aussi appelé cadran qui fait coïncider une heure solaire à une position du soleil.

L'oscillateur est généralement une source de fréquence, transformée en source de temps par le procédé de comptage.

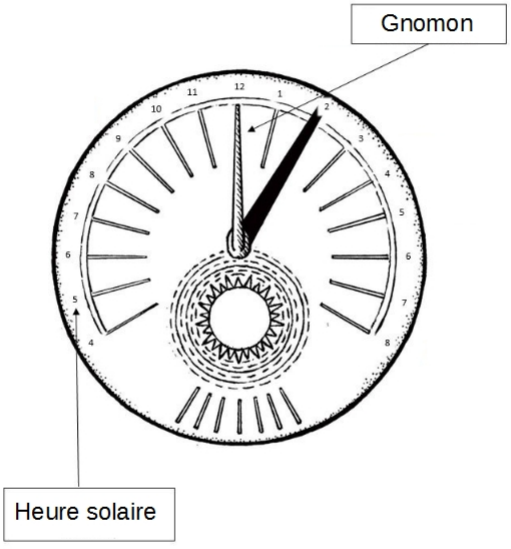
1.1 Les horloges anciennes

Cette section couvre l'ensemble de moyens de mesure du temps depuis les premiers vestiges archéologiques connus jusqu'aux horloges électromécaniques du 19^{e} siècle

1.1.1 Horloges solaires

Ces types d'horloges sont les plus anciennes et les plus intuitives qui puisse exister. En effet, les cycles jour-nuit cadencent notre rythme de vie depuis toujours et constituent la référence de notre horloge biologique.

FIGURE 1.1 – Cadran solaire simple



Les horloges basées sur l'observation du soleil sont constituées d'un gnomon dont l'ombre est projetée sur un plan gradué, le tout forme un cadran solaire. Un jour solaire vrai est défini comme la durée écoulée entre 2 valeurs successives minimales de la longueur de l'ombre projetée, atteintes lorsque le soleil est à son point culminant.

Il existe de nombreux types de cadrans solaires permettant une mesure plus ou moins précise de l'heure courante, les plus évolués intègrent les variations annuelles de la durée du jour solaire dues à l'inclinaison de l'axe de rotation de la terre par rapport à son plan orbital (saisons). Cette mesure est peu précise (quelques minutes dans le meilleur des cas) mais elle est très stable à long terme, ce qui fait que ces horloges ont longtemps servi à régler d'autres types d'horloges (mécaniques).

1.1.2 Horloges mécaniques

Ces horloges s'appuient généralement sur la constante universelle qu'est la force de gravité terrestre et l'énergie potentielle d'une masse dans son champ. Mais d'autres horloges mécaniques utilisent plutôt l'énergie stockée dans un ressort comprimé. Elle convertissent cette énergie en un mouvement régulier par différents moyens.

Ce type d'horloge demande une calibration qui s'effectue dans un environnement donné (température, humidité, condition de l'horloge) et sera donc très sensible à ses variations. L'ordre de grandeur de l'instabilité de ces horloges se situe autour de la minute par jour.

Clepsydres

Les plus anciens modèles ont été découverts en Égypte et leur création est estimée à -1500 av. J.-C. D'autres clepsydres ont été retrouvées en Grèce, au Moyen-Orient et en Chine. Elles sont utilisées pour mesurer des temps de l'ordre de la demi-heure (temps de parole, courses, durée de travail) avec exactitude de quelques minutes.

Le principe repose sur l'écoulement d'un fluide (souvent de l'eau mais parfois d'autres liquides) à travers un orifice. Le fluide est ensuite accumulé dans un réservoir où son niveau de remplissage donne une mesure du temps.

Là encore il existe de nombreuses déclinaisons, de la simple jarre percée avec des graduations à l'intérieur jusqu'à des édifices de plusieurs mètres de haut avec des système de vases communicants pour maintenir le niveau d'eau (et donc le débit) constant dans le récipient percé.

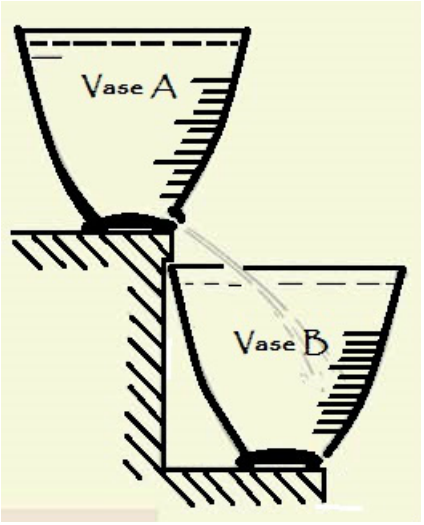


FIGURE 1.2 – Clepsydre égyptienne

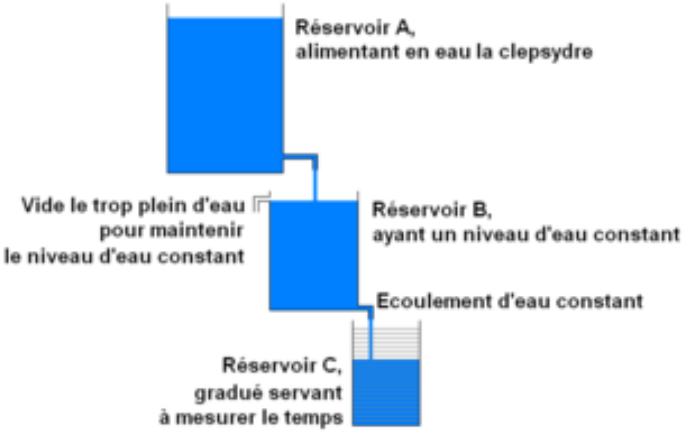


FIGURE 1.3 – Principe de la clepsydre à débit constant

Sabliers

Son apparition est datée autour de 10e siècle et il est encore fabriqué aujourd'hui.

Le principe est similaire à celui de la clepsydre sauf que l'eau est remplacé par du sable fin, le tout fonctionnant en circuit fermé. Ils sont souvent de petite taille afin de pouvoir les retourner facilement. La quantité de sable est calibrée de façon à ce que son transfert d'un compartiment à l'autre représente un nombre entier de minutes.

Le sable est moins sensible au variation de température que l'eau et le fait de fonctionner en circuit fermé évite l'obstruction de l'orifice. L'exactitude atteinte est de l'ordre de la dizaine de seconde sur la demi-heure.

Pendules et balanciers

Il faut attendre les alentours du 15e siècle avant de voir apparaitre des horloges capables de fonctionner plus d'une journée.

Les pendules simples sont constituées d'une masse suspendue au bout d'un fil qui, une fois écartée de sa position d'équilibre, oscille autour de la verticale. Le mouvement d'oscillation peut être entretenu par un système de contrepoids et de balancier afin de maintenir son amplitude constante.

Le lien entre la chute linéaire du poids, le mouvement circulaire des aiguilles et le va-et-vient de la masse est assuré par un ensemble d'engrenages finement calibrés et d'un système d'échappement.

Le terme balancier peut également désigner un ressort comprimé dont le rôle est analogue au contrepoids des pendules.

FIGURE 1.4 – Horloge comtoise

Horloges électriques

Dans un premier temps ces horloges étaient similaires aux horloges mécaniques, sauf que l'entraînement des mécanismes se faisait grâce à un moteur électrique au lieu d'un poids ou d'un ressort, l'énergie électrique à ainsi remplacé l'énergie potentielle due à la gravité terrestre. On parle alors d'horloges électromécaniques.

Puis avec l'apparition des composants électroniques d'autres types de phénomènes oscillatoires ont pu être maîtrisés.

1.2 Les horloges actuelles

Cette section décrit les horloges modernes et les sources de fréquences utilisées depuis le début du 20è siècle jusqu'à aujourd'hui.

1.2.1 Oscillateurs électroniques

Les oscillateurs RLC

Les oscillations générées par ces circuits sont dues à l'échange d'énergie entre 2 composants électroniques ou électriques. Le condensateur stocke l'énergie sous forme d'un champ électrique et l'inductance la stocke sous forme d'énergie magnétique. Un effet de va-et-vient énergétique va créer des oscillations du courant dans le circuit.

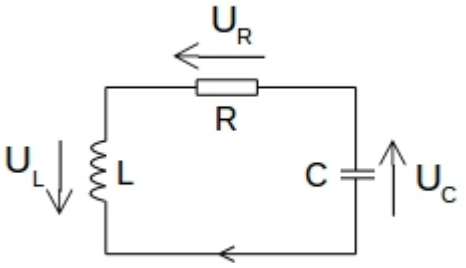


FIGURE 1.5 – Circuit RLC

La résistance permet de modéliser les effets de dissipation d'énergie dans le processus de transfert (câbles, connecteurs et pertes dans L et C.).

Loi des mailles :

$$U_L(t) + U_R(t) + U_C(t) = 0$$

$$L.\frac{di}{dt} + R.i(t) + U_C(t) = 0$$

en dérivant par rapport au temps t on obtient

$$L.\frac{d^2i}{dt^2} + R.\frac{di}{dt} + \frac{dUc}{dt} = 0$$

or

$$i = C.\frac{dU_C}{dt} \text{ soit } \frac{dUc}{dt} = \frac{i(t)}{C}$$

d'où

$$L.\frac{d^2i}{dt^2} + R.\frac{di}{dt} + \frac{i(t)}{C} = 0$$

$$\frac{d^2i}{dt^2} + \frac{R}{L}.\frac{di}{dt} + \frac{i(t)}{L.C} = 0$$

Le courant $i(t)$ obéit à l'équation

$$\frac{d^2i}{dt^2} + 2.\alpha.\frac{di}{dt} + \omega_0^2.i = 0$$

avec

$$\alpha = \frac{R}{2.L} \quad \text{le facteur d'atténuation}$$

et

$$\omega_0 = \frac{1}{\sqrt{L.C}} \quad \text{la fréquence de résonance naturelle du circuit}$$

La solution de cette équation différentielle du 2è ordre est de la forme :

$$i(t) = i_0 \exp^{-\alpha.t} \sin(\sqrt{(\omega_0^2 - \alpha^2)}t + \varphi)$$

On peut voir que, si la valeur de la résistance R est très faible devant la valeur de l'inductance L, le coefficient d'amortissement α devient aussi très faible et le courant va osciller à la fréquence de résonance naturelle du circuit ω_0. Ceci implique des contraintes de fabrication assez poussées pour réduire au maximum les pertes résistives.

La diode à capacité variable

Ce composant électronique est fondamental dans l'ajustement des fréquences de fonctionnement des oscillateurs électroniques. Cette diode se comporte comme une capacité dont la valeur est proportionnelle à l'inverse de la tension appliquée à ses bornes. Elle sert ainsi de capacité contrôlée en tension et donc rend la fréquence des oscillateurs électroniques (RLC, quartz) contrôlable en tension.

Oscillateur à quartz

Son principe repose sur l'effet piezo-électrique [1] dans une molécule à structure cristalline [1]. Le cristal de quartz (formule SiO2) est le type de cristal le plus largement répandu dans les oscillateurs du fait de son abondance à l'état naturel et de ses propriétés piézo-électriques. Il existe toutefois d'autres types de matériaux possédant les mêmes propriétés.

1. mis en évidence en 1880 par Pierre et Jacques Curie

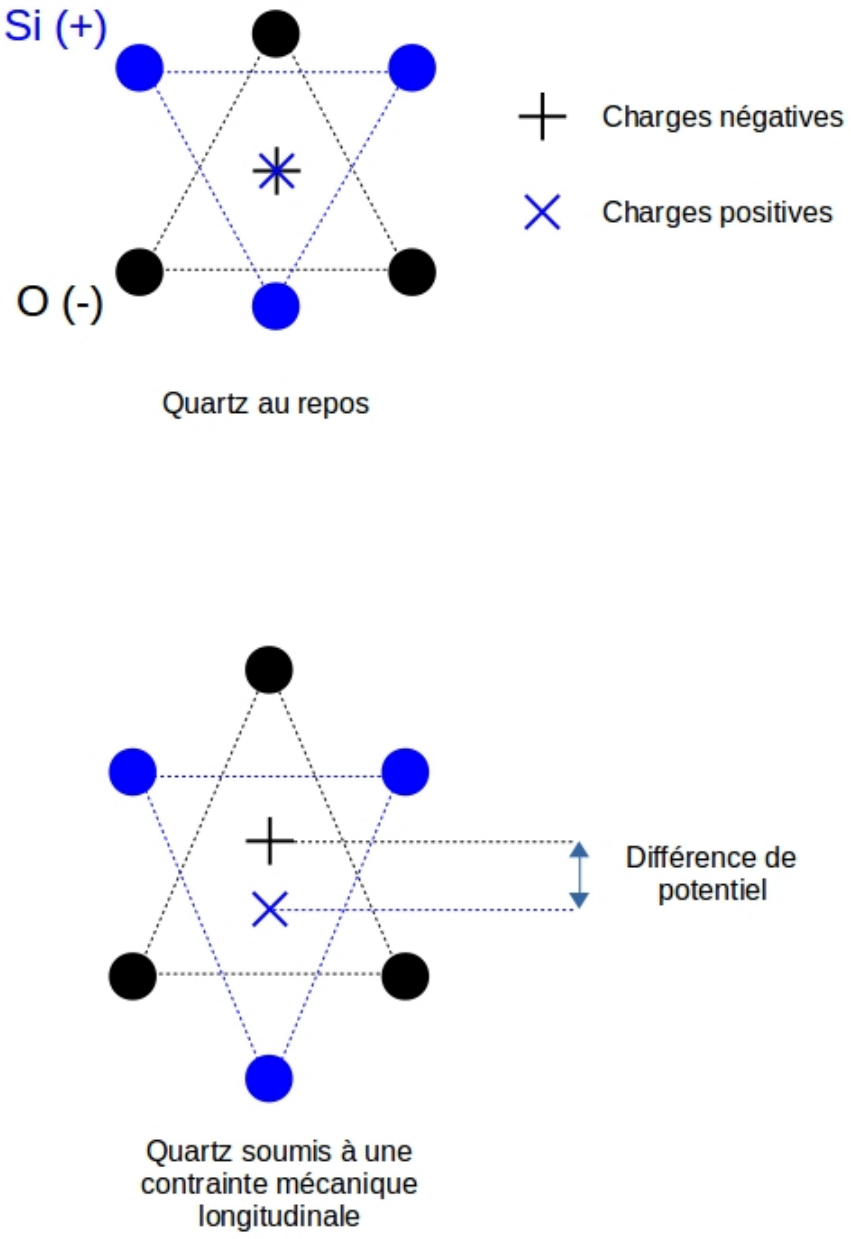


FIGURE 1.6 – Illustration de l'effet piézo-électrique d'un cristal de quartz

Plus exactement c'est l'effet piézo-électrique inverse qui est exploité dans ces oscillateurs. Ainsi en appliquant un champ électrique le cristal va se déformer, puis lorsque le champ électrique disparaît le cristal revient à sa forme initiale. Il existe des fréquences de changement du champ électrique pour lesquelles le cristal va entrer en résonance [2], c'est à dire que l'amplitude de ses déformations mécaniques va fortement augmenter pour une fréquence particulière.

Il y a ainsi échange d'énergie entre un stockage sous forme électrique et sous forme mécanique, ainsi on peut faire l'analogie avec les circuits RLC et modéliser un oscillateur à quartz sous la forme d'un schéma RLC équivalent.

2. Cette fréquence de résonnance est directement liée à la forme géométrique du cristal.

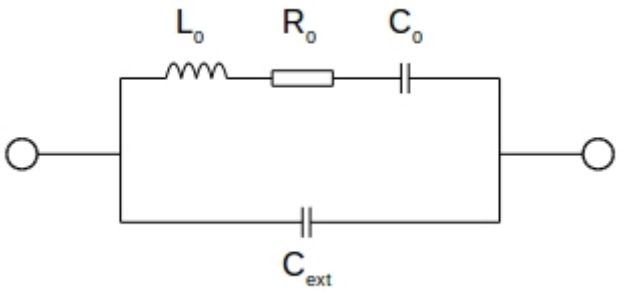


FIGURE 1.7 – Schéma électrique équivalent d'un oscillateur à quartz

L_0, R_0 et C_0 représentent les caractéristiques intrinsèques du cristal (coupe)
C_{ext} représente l'ensemble des caractéristiques dues à l'environnement extérieur de conditionnement du quartz (boîtier, électrodes).

Son impédance s'écrit donc sous la forme :

$$\begin{aligned}
Z_{cristal} &= (Z_{L_0} + Z_{C_0} + Z_{R_0}) \quad // \quad Z_{C_{ext}} \\
&= \frac{(Z_{L_0} + Z_{C_0} + Z_{R_0}) * Z_{C_{ext}}}{Z_{L_0} + Z_{C_0} + Z_{R_0} + Z_{C_{ext}}} \\
&= \frac{(L_0.p + \frac{1}{C_0.p} + R_0) * \frac{1}{C_{ext}.p}}{L_0.p + \frac{1}{C_0.p} + R_0 + \frac{1}{C_{ext}.p}} \\
&= \frac{1}{C_{ext}.p} \cdot \frac{L_0.p + \frac{1}{C_0.p} + R_0}{L_0.p + \frac{1}{C_0.p} + R_0 + \frac{1}{C_{ext}.p}} \\
&= \frac{1}{C_{ext}.p} \cdot \frac{p^2 + \frac{1}{C_0.L_0} + \frac{R_0}{L_0}.p}{p^2 + \frac{C_{ext}.C_0}{C_0.L_0.C_{ext}} + \frac{R_0}{L_0}.p} \\
Z_{cristal} &= \frac{1}{C_{ext}.p} \cdot \frac{p^2 + \frac{R_0}{L_0}.p + \omega_S^2}{p^2 + \frac{R_0}{L_0}.p + \omega_P^2}
\end{aligned}$$

avec

$$\omega_S = \frac{1}{\sqrt{C_0.L_0}} \quad \text{la fréquence de résonnance série}$$

et

$$\omega_P = \sqrt{\frac{C_{ext}.C_0}{C_0.L_0.C_{ext}}} \quad \text{la fréquence de résonnance parallèle}$$

Ainsi en rajoutant une capacité en parallèle à l'oscillateur on peut faire varier sa fréquence de résonance.

Du fait de leur faible encombrement les quartz sont les sources de fréquence les plus largement répandues dans le monde. Du quartz d'une montre (1 euro – 1 mm2) jusqu'aux boîtiers régulés en températures (10000 euros - 1dm2) leur champs d'application est très vaste.

Les oscillateurs à quartz ont longtemps constitué les références de temps officiel jusqu'à l'apparition des horloges atomiques.

1.2.2 Les horloges atomiques

Les niveaux d'énergie des électrons

Elles sont basées sur le modèle quantique de la matière qui stipule que les électrons d'un atome ne peuvent se trouver qu'à des niveaux d'énergie parfaitement quantifiés. C'est un modèle complexe dont je vais tenter d'expliquer le principe pour l'application des horloges atomiques.

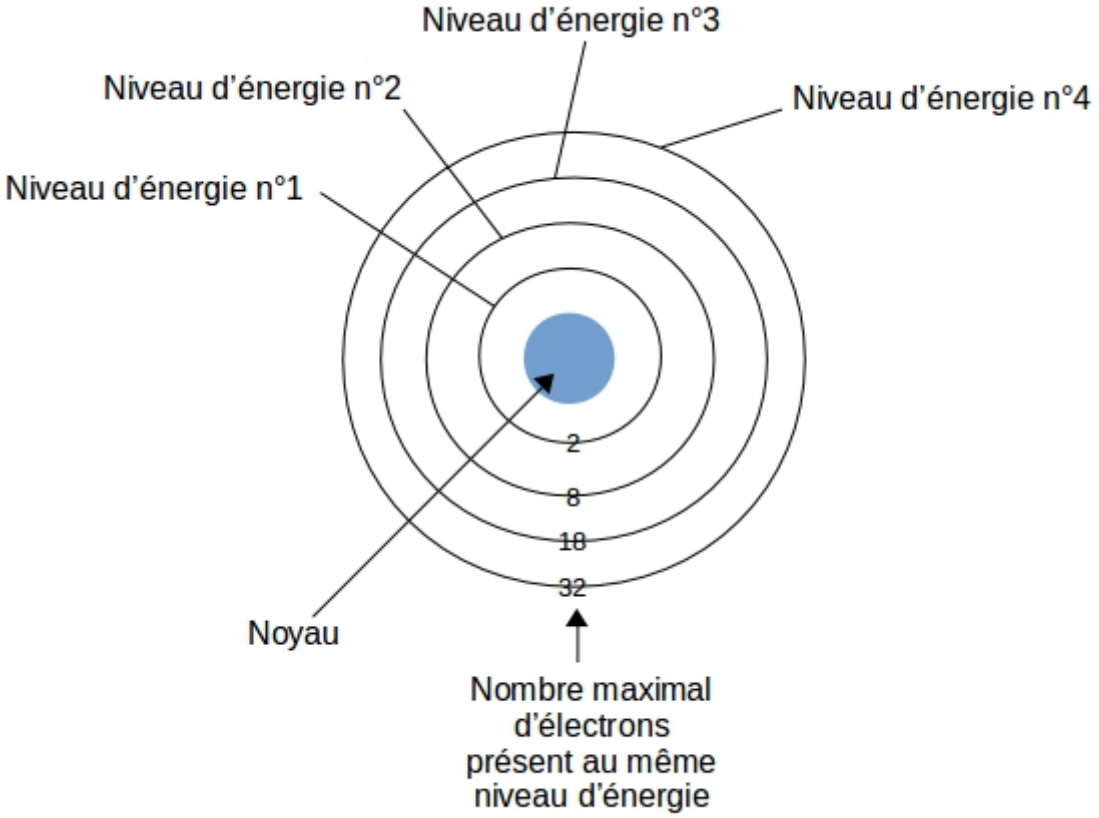


FIGURE 1.8 – Modèle de Bohr d'un atome

Les atomes constituant la matière sont composés d'un noyau entouré d'un nuage d'électrons. La théorie quantique utilise 4 nombres pour décrire l'état des électrons :

— Nombre quantique principal.
  Décrit le niveau d'énergie de l'électron, représenté sur la figure 1.8 par les 4 cercles concentriques autour du noyau.
— Nombre quantique azimutal.
  Décrit les sous-niveaux d'énergie.
— Nombre quantique magnétique.
  Décrit l'orientation de l'orbital atomique par rapport au noyau.
— Nombre quantique de spin. Décrit le moment cinétique de l'électron.

Les niveaux d'énergie principaux représentés sur la figure ci-dessus sont divisés en plusieurs sous-niveaux. Ceci est dû, entre autres, aux interactions entre électrons et aux effets relativistes induits par la masse non-nulle des électrons. L'ordre de grandeur de l'énergie de ces niveaux principaux est de quelques électrons-volts (eV).

Le premier sous-niveau est nommé structure fine et l'écart typique au sein de ce sous-niveau est de l'ordre de 10^{-3} eV. Le deuxième sous-niveau est nommé structure hyperfine et son ordre de magnitude en son sein est de 10^{-4} eV.

Le niveau de stabilité et de précision exigé dans les horloges atomiques oblige à considérer une représentation la plus exacte possible (d'un point de vue des phénomènes physiques en jeu). Ainsi c'est la représentation des niveaux d'énergie en structure hyperfine qui s'impose, on parle alors de niveaux d'énergie hyperfins.

Les transitions entre niveaux d'énergie

Lorsque l'atome est dit au repos, les électrons occupent des orbitales bien définies. Mais il est possible qu'une excitation extérieure (phénomène d'absorption) puisse faire changer un électron d'orbite en augmentant son énergie. Ce phénomène ne peut apparaître que si l'énergie de la particule qui est absorbée est strictement égale à l'énergie nécessaire au passage au niveau supérieur. Le type de particule absorbé est généralement un photon, mais d'autres phénomènes peuvent amener un électron à passer à un niveau d'énergie supérieur (agitation thermique, excitation par guide d'onde, transitions non-radiatives).

Par exemple, pour qu'un électron passe du niveau d'énergie n° 1 au niveau d'énergie n° 2 il lui faut absorber un photon d'énergie $E =$ (énergie du niveau n° 2) – (énergie du niveau n° 1).

Le phénomène inverse se nomme l'émission. Lors d'un passage spontané de l'électron d'un état excité (niveau d'énergie E_2) à un état stable d'énergie inférieur (E_1) une particule (photon) d'énergie $(E_2 - E_1)$ est émise dans une direction aléatoire.

Un autre type d'émission de photon est l'émission stimulée. Dans ce cas un photon incident va « synchroniser » le photon émis et ils posséderont des propriétés communes (phase, fréquence et direction).

L'énergie E d'un photon et sa fréquence de radiation ν sont liées par la relation suivante :

$$E = h.\nu$$

h est la constante de Planck, elle est égale à approximativement 6,6 10^{-34} m^2.kg/s

Ainsi la fréquence de la particule émise est strictement quantifiée. C'est la stabilité de cette fréquence qui constitue la base de la stabilité des horloges atomiques. On l'appelle fréquence de transition atomique.

Le type d'horloge atomique et tout le système physique qui l'entoure sont donc directement liés à l'atome qui est excité. Cet atome est nommé atome de référence et il est identifié bien précisément, notamment par son nombre de masse (nombre de neutrons) permettant de le différencier de ses isotopes, qui n'ont pas exactement les mêmes orbitales électroniques et donc pas les mêmes niveaux d'énergie.

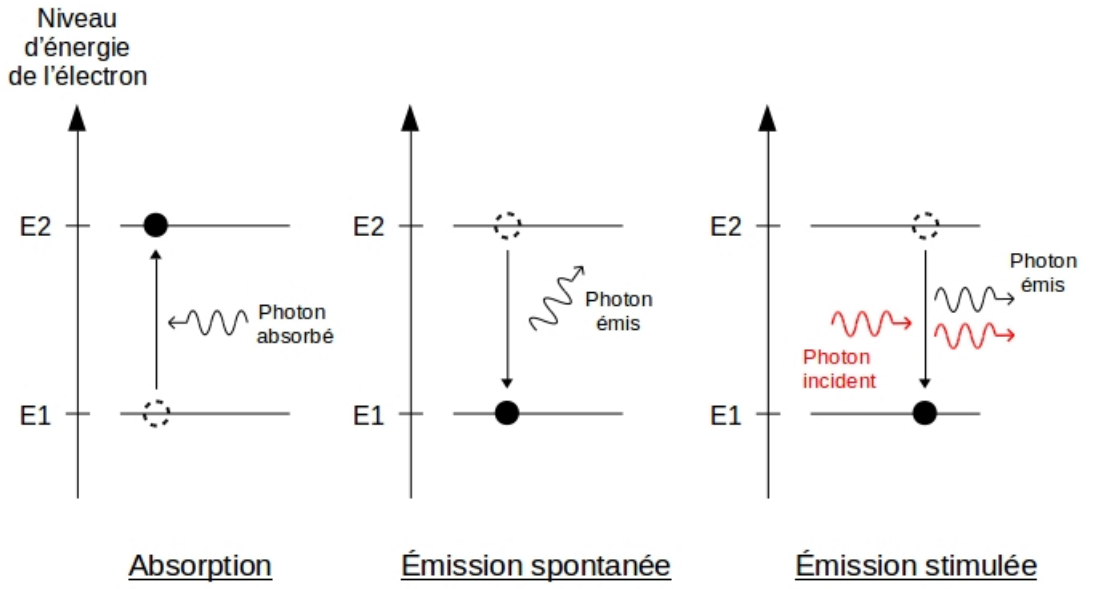


FIGURE 1.9 – Transition énergétique d'un électron

Réalisation pratique des horloges atomiques commerciales

Les principes théoriques de base des horloges atomiques étant posés on peut maintenant se pencher sur la fabrication concrète d'une horloge atomique.

L'émission du flux utile Afin de pouvoir manipuler les atomes de référence pour en extraire l'information sur ses transitions énergétiques il faut disposer d'un flux utile. Ce flux utile peut se présenter sous 2 formes :

- Un flux lumineux (photons)
  Les atomes de référence sont confinés dans un environnement fermé et sont excités par stimulation extérieure (thermique ou optique) afin qu'ils émettent des photons correspondant aux différentes transitions énergétiques. Cette méthode implique forcément par la suite l'utilisation d'autres isotopes stables de l'atome de référence [3], ce qui n'est pas systématiquement possible selon le type d'atome considéré.

- Un flux de matière (atomes)
  Les atomes de référence sont propulsés par chauffage dans un four pour créer un gaz concentré en un faisceau. Ces architectures sont souvent plus coûteuses et ont une durée de vie moindre que celles à base de flux lumineux du fait de la consommation des atomes de référence (épuisement du « carburant »).

3. Possédant une raie d'absorption en moins que l'isotope de référence, afin de filtrer une partie des photons (par ex. le rubidium)

Les étages de sélection d'état Cet étage sert à discriminer et à dissocier les atomes de référence selon leur état (excités ou non).

Dans le cas d'un flux utile lumineux on utilise des cellules d'absorption constituées d'un isotope possédant un nombre inférieur de niveaux d'énergie. Ainsi cette cellule va absorber certaines raies lumineuses et être transparente pour d'autres.

Pour un flux d'atomes on trouve 2 principaux sélecteurs d'état :

La déflexion magnétique

Les atomes excités n'ayant pas les mêmes propriétés magnétiques que leur version stable ils ne vont donc pas réagir de la même façon lors d'un passage dans un champ magnétique. Ce dernier va donc dévier de façon différente les 2 version de l'atome de référence.

Le pompage optique

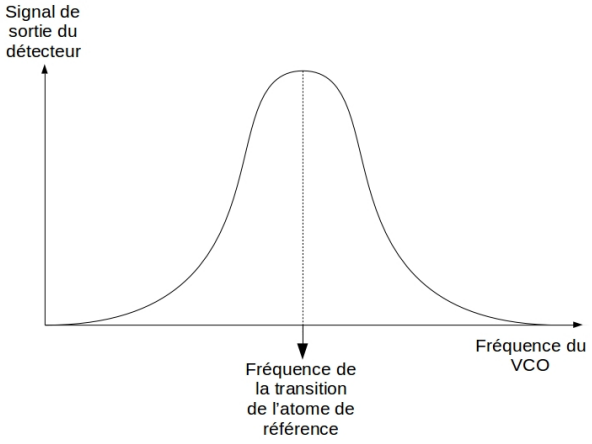
Ici on va chercher à forcer l'état de l'atome par stimulation optique (laser) permettant ainsi de peupler certains niveaux d'énergie au détriment des autres. Cette méthode constitue une amélioration de la précédente car elle permet d'augmenter l'intensité du flux produit d'un facteur pouvant aller jusqu'à 100. Cependant elle demande une très grande rigueur dans le contrôle de la fréquence du laser utilisé, ce qui en fait une méthode encore réservée aux laboratoires, mais plus pour longtemps.

L'étage d'interrogation Cet étage est un interféromètre permettant de lier la densité de population de certains niveaux d'énergie à un signal extérieur modulé. Il joue le rôle de comparateur entre la fréquence de sortie de l'horloge et la fréquence de transition atomique.

L'étage de détection Il sert à mesurer la valeur d'un paramètre du flux utile selon le type de ce dernier.

Pour un flux lumineux il s'agira d'un détecteur de lumière fournissant un signal électrique proportionnel à l'intensité lumineuse détectée. Pour un flux d'atome il donnera un signal image d'une quantité de matière (spectrogramme de masse).

En pratique on cherche à asservir le VCO sur une fréquence légèrement inférieure à celle de la transition de l'atome pour éviter les effets de cuvette dû à la pente nulle au sommet de la courbe de détection. Un enjeu majeur de l'amélioration des horloges atomiques est de réduire au maximum la largeur de ce pic de résonance afin de se trouver au plus proche de la fréquence de transition atomique (exactitude) et de corriger au mieux le VCO (stabilité) dès que sa fréquence commence à varier.



L'étage d'asservissement Cet étage constitue l'étage final de l'horloge. Il a pour rôle de moduler le signal injecté à l'étage d'interrogation afin d'obtenir une réponse maximale de l'étage de détection. Il est quasiment toujours constitué d'un quartz contrôlé en tension. Dans certains cas il se peut qu'aucun asservissement ne soit présent. C'est le cas lorsque le flux utile est suffisamment puissant (ou amplifié) pour engendrer un signal de sortie d'amplitude suffisamment élevée pour

qu'il puisse directement être détecté. L'émission stimulée peut permettre ce genre d'architecture, on parle alors d'horloge active.

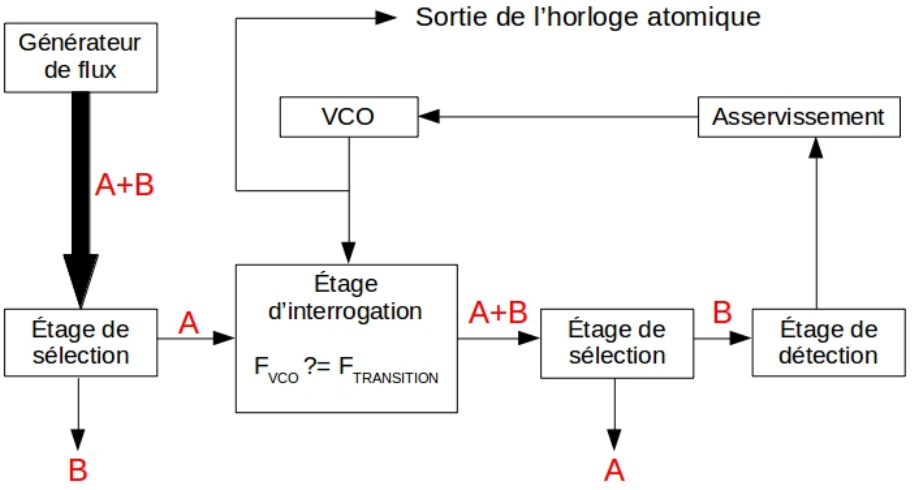


FIGURE 1.10 – Schéma générique d'une horloge atomique passive

Horloge à gaz de rubidium

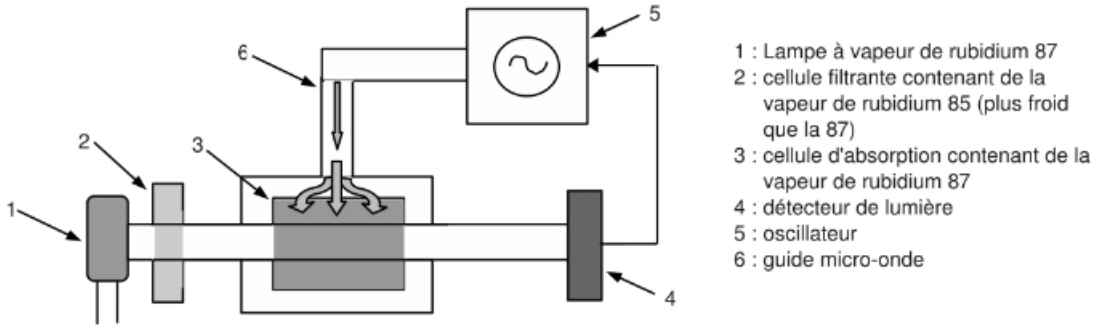


FIGURE 1.11 – Schéma de principe de l'horloge à gaz de rubidium

Les versions commerciales des horloges à rubidium utilisent un flux utile optique en utilisant le ^{87}Rb pour sa génération (lampe à vapeur de rubidium 87). Le ^{87}Rb possède 2 niveaux hyperfins stable E_1 et E_2, les niveaux d'énergie excités sont également au nombre de 2 : E_3 et E_4. Cette lampe émet 2 (doubles) raies lumineuses notées D_1 et D_2.

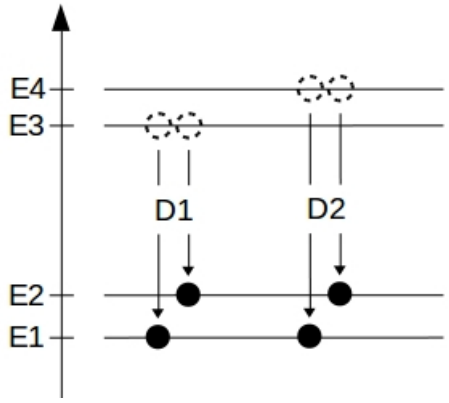


FIGURE 1.12 – Spectre émis par la lampe à ^{87}Rb

La cellule de filtrage (étage de sélection) est constituée de son isotope ^{85}Rb permettant de masquer le niveau hyperfin E_2 pour chacune des 2 raies. Uniquement les raies issues des transitions vers le niveau d'énergie E_1 vont la traverser.

Je n'ai trouvé aucune explication sur les causes de ce phénomène d'absorption dans la littérature. Par déduction je pense simplement que l'isotope ^{85}Rb a son niveau d'énergie E_1 égal au niveau E_2, E_1 est « ramené » sur E_2 du fait de la configuration différente de son noyau (moins de masse donc « écartement » des électrons de l'orbite basse ?). Ainsi il est incapable d'absorber les photons d'énergie $(E_3 - E_1)$ et $(E_4 - E_3)$ qu'il laisse passer. Les autres photons sont absorbés et l'émission spontanée se faisant dans une direction aléatoire leur énergie est dissipée, créant ce phénomène de filtrage.

Ce signal de sortie de la cellule filtrante va servir à dépeupler les niveaux d'énergie E_1 dans la cellule d'absorption. Au bout de plusieurs cycles le niveau sera complètement dépeuplé et la

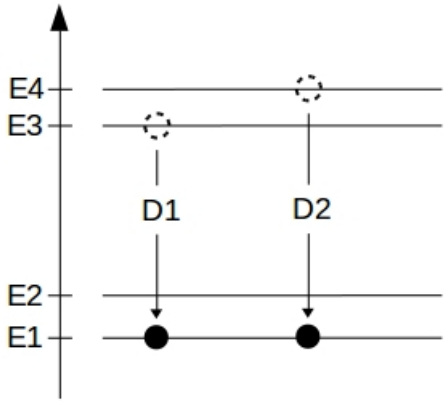


FIGURE 1.13 – Spectre en sortie de la cellule filtrante à ^{85}Rb

totalité du signal lumineux sera transmis au détecteur de lumière. C'est à ce moment que rentre en jeu le signal de l'oscillateur transmis à la cellule d'absorption par le guide d'onde. Ce signal va stimuler l'émission du niveau E_2 vers le niveau E_1 et ainsi permettre à la cellule d'absorption de continuer à absorber les photons issus de la cellule de filtrage, faisant baiser l'intensité lumineuse reçue par le détecteur de lumière. Le signal de l'oscillateur est contrôlé afin de maintenir une intensité lumineuse minimale sur le détecteur.

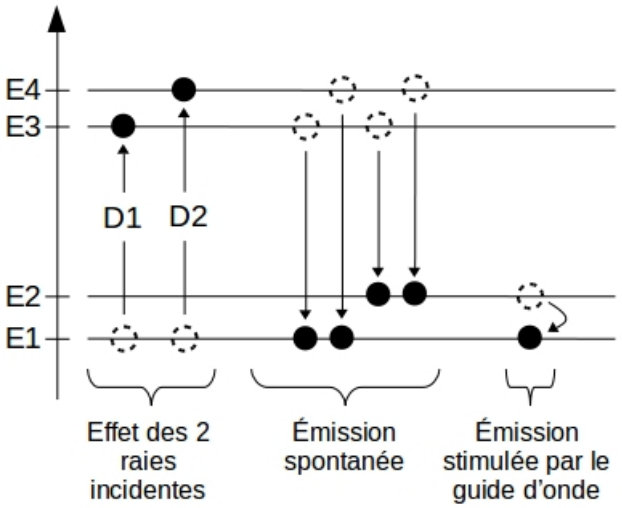


FIGURE 1.14 – Principe de la cellule filtrante stimulée par le guide d'onde

Ainsi l'oscillateur sera asservi sur la fréquence de transition atomique entre les niveaux hyperfins E_1 et E_2 de l'atome de rubidium 87.

Cette fréquence vaut 6 834 682 613 Hz.

Maser à hydrogène

Historiquement le maser (Microwave Amplification by Stimulated Emissions of Radiations) est l'ancêtre du laser (Light Amplification by Stimulated Emissions of Radiations), dont le principe repose sur l'amplification due à l'émission stimulée. La différence est que le maser opère dans le domaine des micro-ondes et le laser dans le domaine optique. Le principe date de plus de 50 ans [2].

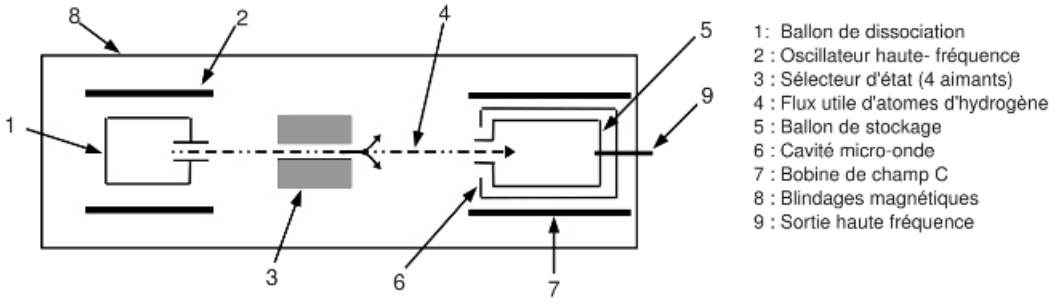


FIGURE 1.15 – Schéma de principe du maser passif

Le dihydrogène est dissocié en hydrogène dans le ballon d'émission du maser par la stimulation d'un champ électrique haute fréquence. Le flux d'atome d'hydrogène traverse ensuite un sélecteur d'état par déflexion magnétique afin de dissocier les atomes non-excités des autres. Les atomes excités arrivent ensuite dans le ballon de stockage où ils vont provoquer des émissions stimulées sur les atomes déjà présents. Une réaction en chaîne d'émissions stimulées à alors lieu dans le ballon de stockage. Le matériau et la forme du ballon sont définis de façon à ce que les atomes excités « rebondissent » sur ses parois. Quand suffisamment d'atomes sont stimulés, un signal sera détecté par la cavité micro-onde et permettra d'en extraire le signal de sortie du maser. Cette cavité est protégée par un champ magnétique créé par une bobine de champ permettant de réduire l'influence des champs perturbateurs externes.

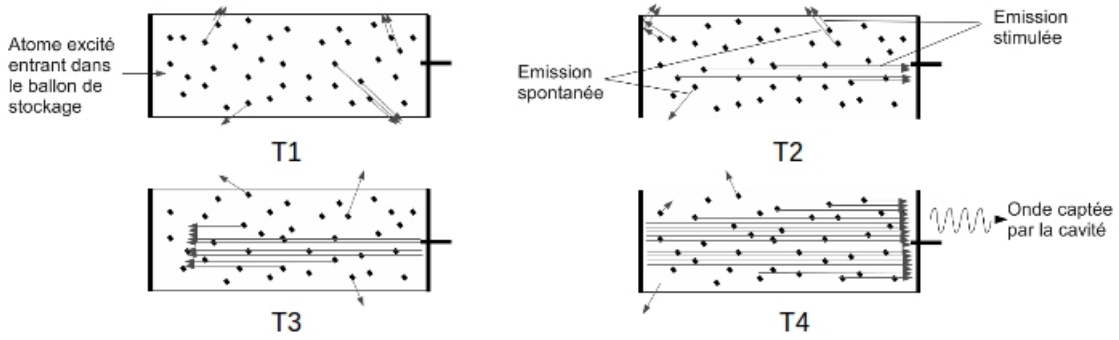


FIGURE 1.16 – Principe de l'amplification dans le ballon de stockage du maser

La fréquence de transition de l'hydrogène atomique est 1 420 405 751 Hz

Horloge à jet de césium

Le principe est identique à l'horloge à rubidium sauf que le flux utile est un flux d'atomes au lieu d'un flux lumineux.

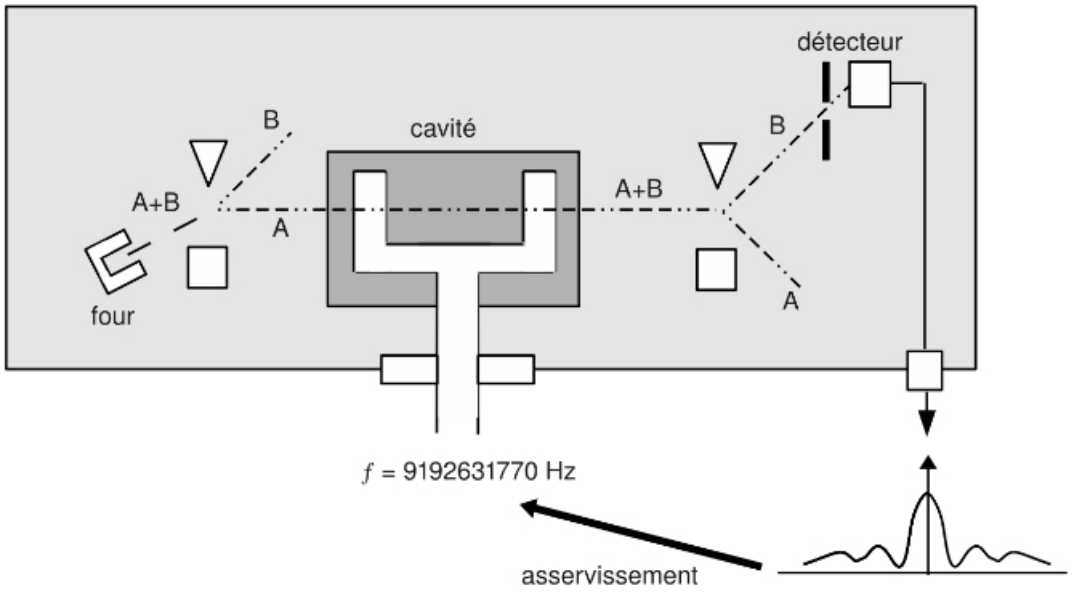


FIGURE 1.17 – Principe de l'horloge à jet de césium

La fréquence de transition atomique du césium est 9 192 631 770 Hz.

C'est à partir de cette fréquence de transition qu'est définie la durée officielle d'une seconde depuis plus de 60 ans. Les horloges à césium sont par conséquent largement répandues dans les applications Temps-Fréquence [3].

Fontaines atomiques

Ces horloges peuvent utiliser différents atomes de référence (césium, rubidium). Le principe est le même qu'une horloge à jet d'atomes (utilisation d'un four) sauf que le flux d'atomes est confiné par plusieurs lasers concourants permettant de refroidir (ralentir) les flux. Ce flux est envoyé à la verticale dans l'étage d'interrogation et par l'action de la pesanteur terrestre il redescend et traverse une seconde fois l'étage d'interrogation. Ainsi la fréquence de transition est mieux interrogée et la stabilité est améliorée. Ces horloges sont réservées au domaine de la recherche scientifique et ne sont pas commercialisées.

Si on traçait la trajectoire des atomes on verrait qu'elle ressemble à celle de l'eau dans une fontaine, d'où le nom des ces horloges.

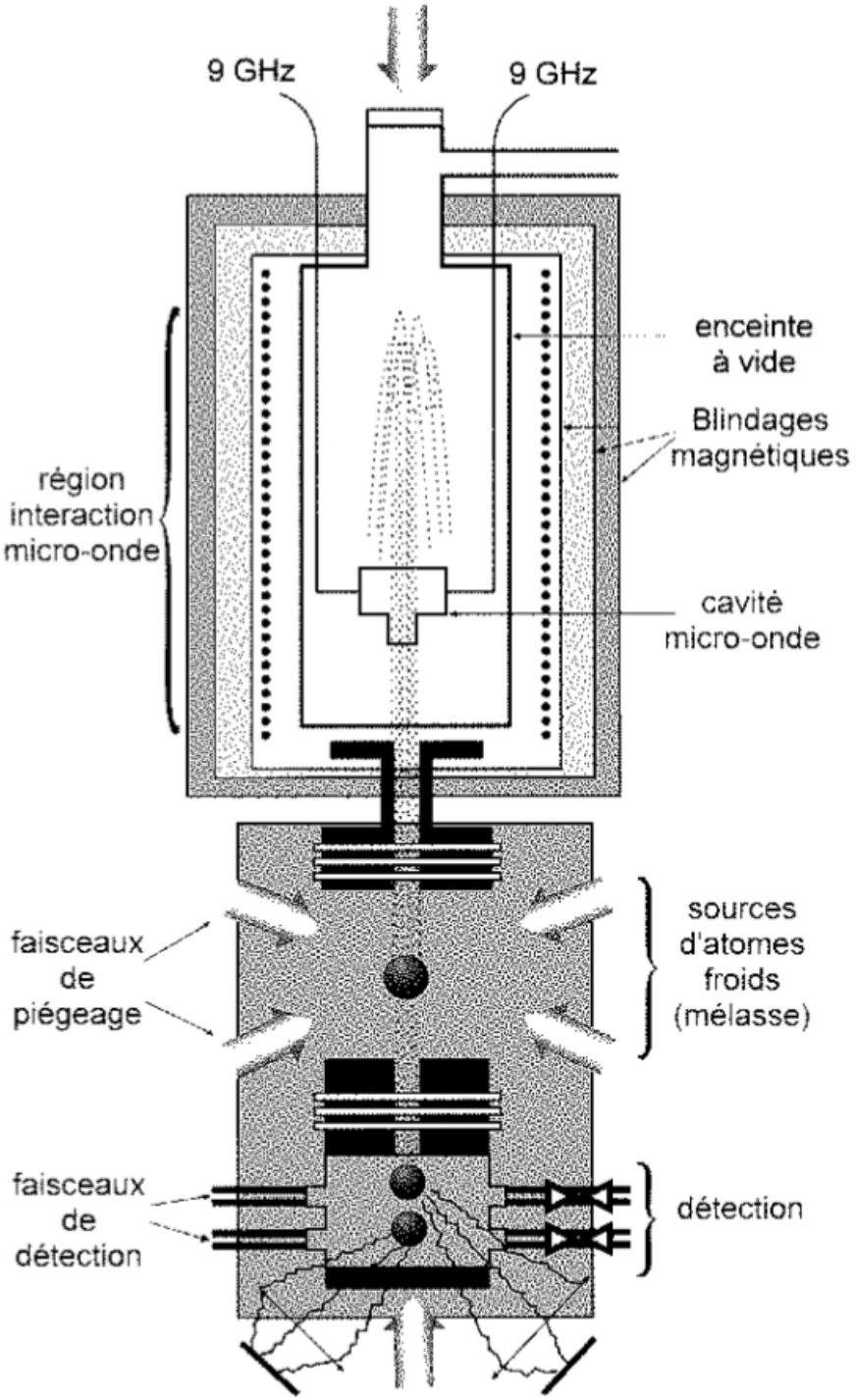


FIGURE 1.18 – Principe d'une fontaine atomique (césium)

1.2.3 Oscillateur cryogénique à saphir

Cette horloge est constituée d'une technologie à part, basée sur la propagation des ondes à mode de galerie dans un résonateur en saphir[4].

L'exemple le plus célèbre de la manifestation de ce type de propagation est le dôme de la cathédrale St Paul à Londres. Dans cet édifice on peut entendre parfaitement un chuchotement émis depuis une extrémité du dôme jusqu'à l'autre extrémité. Les ondes acoustiques sont transmises le long de la paroi en suivant le principe de réflexion totale interne (de la même façon que la lumière dans les fibres optiques).

L'oscillateur cryogénique à saphir n'est donc pas une horloge atomique mais, une fois encore, on cherche à atteindre le pic de résonance de l'oscillateur et à l'extraire.

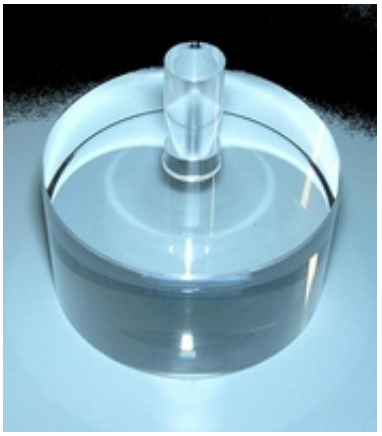


FIGURE 1.19 – Le résonateur en saphir (5cm de diamètre, 3cm de haut)

Ce résonateur est confiné dans une enceinte cryogénique maintenue à 6 Kelvins (-267^{o}C) impliquant l'utilisation d'hélium liquide. Sa mise en œuvre demande donc une grande rigueur dans la fabrication du résonateur mais aussi dans l'électronique qui l'entoure.

Sa fréquence de fonctionnement est de 10 GHz.

1.3 Les horloges de demain

La recherche de l'amélioration de stabilité des horloges atomiques tend à faire monter en fréquence ces dernières[5], délaissant le domaine radio-fréquence et micro-ondes (quelques gigahertz) pour se diriger vers les fréquence optiques (visible, plusieurs térahertz).

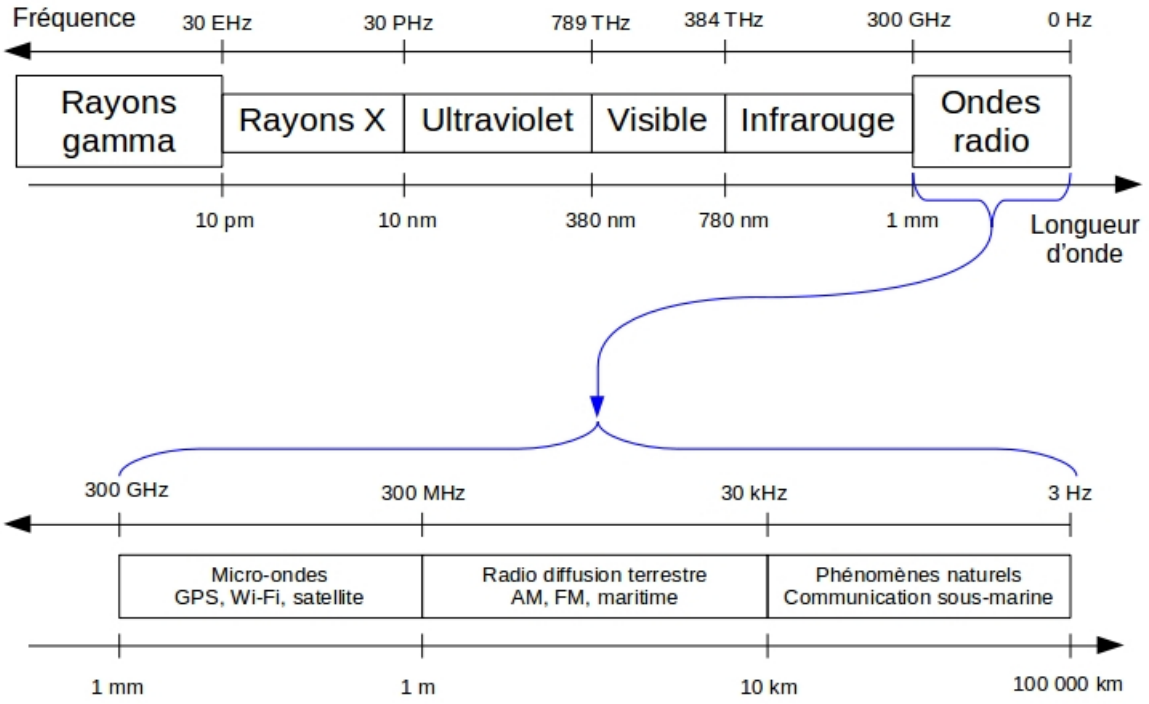


FIGURE 1.20 – Spectre des ondes électromagnétiques

On voit apparaître des horloges basées sur des atomes de référence dont les fréquences de transition atomique dépassent le térahertz (aluminium, calcium, mercure, strontium, indium, magnésium, ytterbium). Tout l'enjeu est donc d'être capable de mesurer ces fréquence. Pour cela on utilise des peignes de fréquence laser permettant de démultiplier ces fréquences pour les ramener dans le domaine micro-onde bien mieux maîtrisé.

Deux grands types de technologies sont apparus pour élaborer ces horloges optiques :

- Des ions isolés dans un piège à ions électromagnétique
- Des atomes neutres piégés dans une grille optique

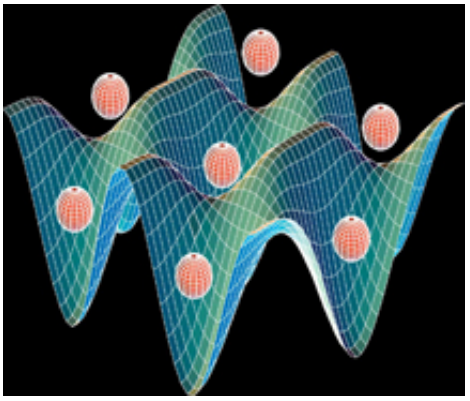


FIGURE 1.21 – Atomes piégés dans une grille optique (vue d'artiste)

Ces techniques permettent de bien isoler les atomes (ou les ions) de façon à réduire leurs interactions avec d'éventuelles perturbations extérieures. Ce qui apporte une stabilité accrue à ce type d'horloges qui montrent déjà des stabilités supérieures aux horloges à césium et pourraient donc redéfinir la durée officielle de la seconde.

Chapitre 2

Le transfert des signaux métrologiques

Les signaux métrologiques n'auraient que très peu d'utilité si on ne pouvait pas les transférer vers leurs applications. Afin de diffuser ces signaux issus des différentes sources de fréquence aux utilisateurs potentiels il existe plusieurs moyens. Chacun possédant ses caractéristiques propres liées au type de transfert souhaité.

2.1 Les transferts par média physique

2.1.1 Câble coaxial

C'est le moyen le plus simple et le moins coûteux pour transférer les signaux de la source à l'utilisateur. C'est un câble à deux conducteurs généralement constitué de 4 couches : un conducteur central transportant le signal primaire (e.g. cuivre), une couche fabriquée dans un matériau isolant, un blindage transportant le signal secondaire (e.g. relié à la masse) et une gaine de protection en plastique.

C'est la différence de potentiel entre le conducteur primaire et secondaire qui constitue le signal transporté. Son principal avantage est d'être peu sensible aux perturbations électromagnétiques extérieures, ceci étant dû au fait que les 2 conducteurs ont un centre de symétrie confondu, ce qui répercute l'effet des perturbations sur ceux-ci de la même façon et est donc compensé par la différence de potentiel.

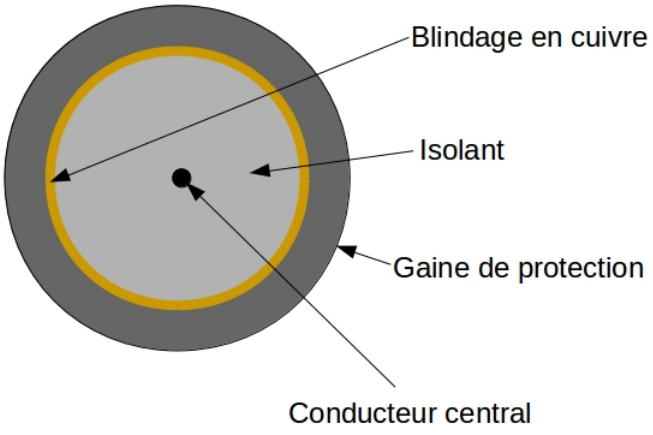


FIGURE 2.1 – Coupe transversale d'un câble coaxial

L'atténuation d'un câble coaxial augmente avec la fréquence du signal transporté et la distance parcourue. Généralement on l'utilise pour transporter des signaux de quelques centaines de mégahertz sur quelques dizaines de mètres au maximum.

IRIG

La norme IRIG [1] à été créée dans les années 50 et établit des méthodes d'encodage et de transmission de datations par câble coaxial.

2.1.2 Fibre optique

C'est le média physique le plus performant utilisé aujourd'hui. Il est constitué d'un cœur en verre ou en plastique conducteur de lumière entouré d'une enveloppe possédant un indice de réfraction légèrement plus élevé, ainsi le signal est réfléchi à l'interface entre le cœur et la gaine (phénomène de réflexion totale interne) ce qui engendre une très faible atténuation comparée au câble coaxial. Le tout est entouré d'une gaine de protection rigide afin que l'angle d'incidence du signal ne soit pas trop élevé pour conserver les propriétés de réflexion totale. Cependant sa réalisation demande un minimum d'équipement électronique pour être mise en œuvre, notamment des systèmes d'émission/réception adéquats (diodes laser par exemple) ce qui engendre des coûts supérieurs au câble coaxial (bien que les prix du cuivre soient en forte augmentation ces dernières années).

Du côté des perturbations électromagnétiques, la fibre optique n'étant pas constituée de matériaux métalliques elle n'est pas sujettes aux effets « d'antenne » et ne capte donc pas de perturbation extérieures dans le domaine radio-fréquence, cependant elle est sensible aux perturbations dans le domaine proche du visible (infrarouge jusqu'aux ultra-violets).

Sa distance de transfert maximale sans répéteur (sans ré-amplification) est actuellement de l'ordre de la centaine de kilomètres. Avec ré-amplification la distance couverte peut-être de plusieurs milliers de kilomètres, c'est pas cette méthode que les continents sont reliés à internet.

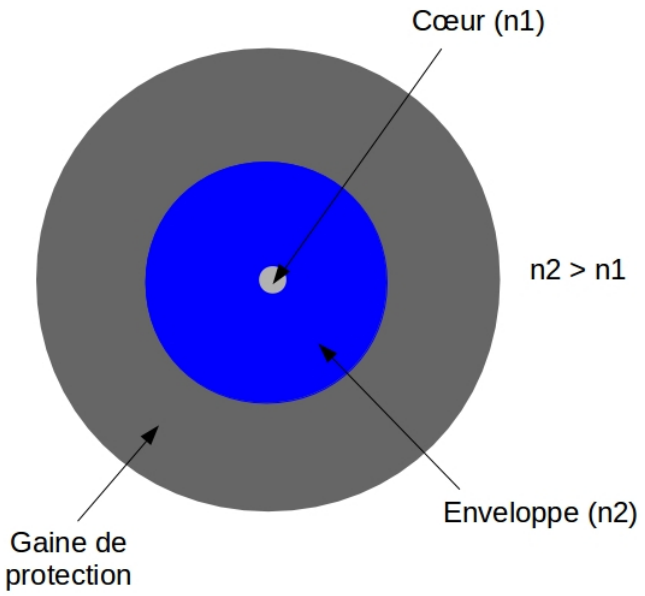


FIGURE 2.2 – Coupe transversale d'une fibre optique

1. Inter-Range Instrumentation Group

2.1.3 Synchronisation par réseau

Ce type de transfert utilise un réseau Ethernet cablé (style internet ou LAN) pour synchroniser un oscillateur local sur une référence distante.

NTP

Le protocole NTP [2] a été créé dans les années 1980 aux débuts d'internet. Il permet de réaliser l'intercomparaison d'une horloge locale avec un signal de référence grâce à une communication bidirectionnelle. Son principe permet d'éliminer les effets aléatoires de latence dans les réseaux complexes (avec beaucoup de noeuds et de traffic).

L'horloge "cliente" envoie une requête à l'horloge "serveur" et reçoit les datations effectuées par cette dernière lors de la réception de la demande et de l'envoi de la réponse, ce qui permet de mesurer le temps de propagation du signal dans le réseau indépendamment de l'écart temporel entre les 2 horloges distantes. L'écart temporel déduit est ensuite traité par une partie logicielle qui corrige (en phase et/ou en fréquence) l'oscillateur local.

L'architecture des réseaux NTP est constituée en plusieurs strates :

. Strate 0 : une base composée d'étalons atomiques type césium ou rubidium corrigée par GPS

. Strate 1 : un parc de serveurs primaires directement synchronisés avec les étalons et interagissant entre eux pour se contrôler.

. Strate N > 1 : Un ensemble de clients synchronisés sur la couche supérieure

2. Network Time Protocol

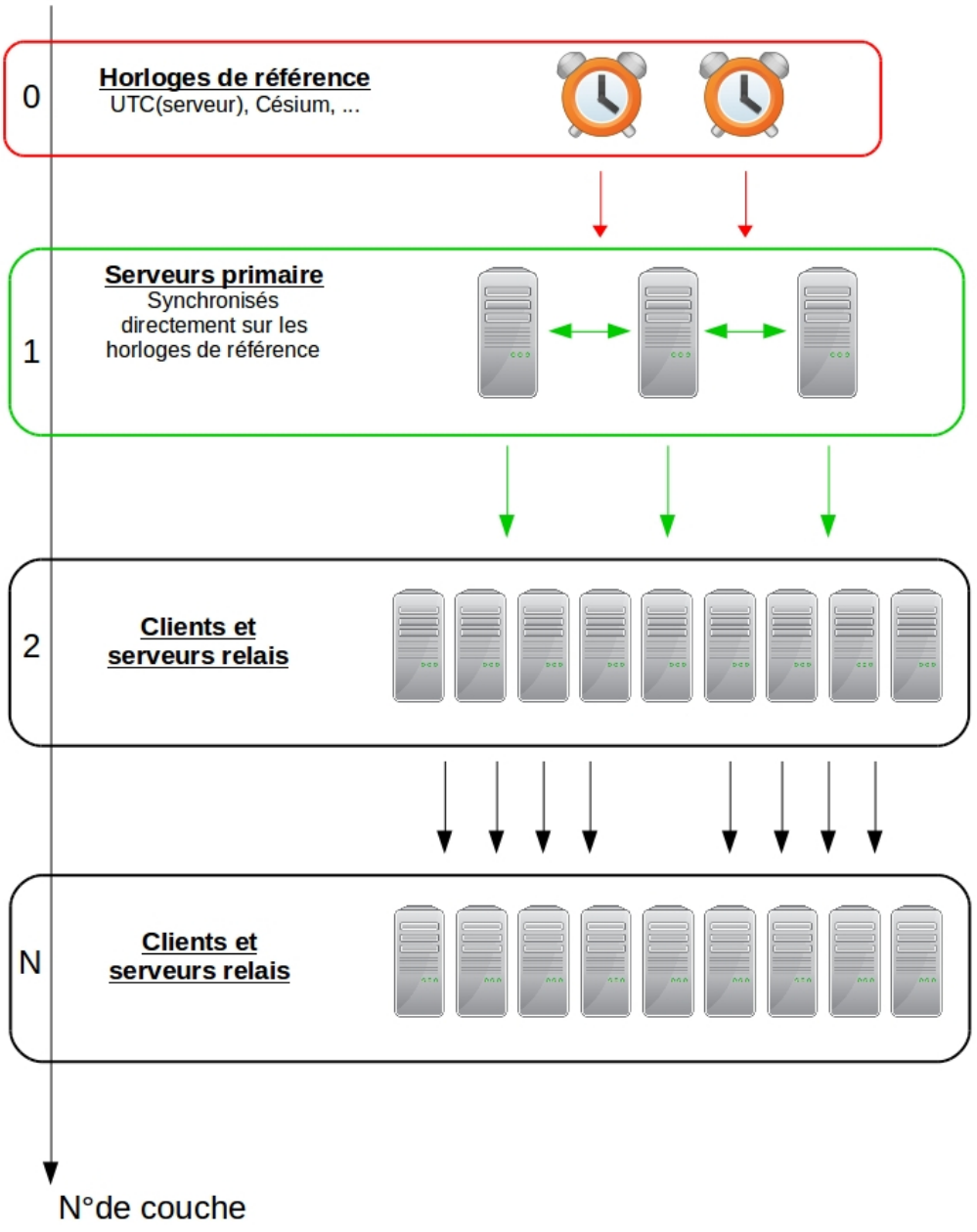


FIGURE 2.3 – Architecture d'un réseau NTP

Chaque système appartenant à une couche N (≥ 1) est client de la couche N-1 et serveur de la couche N+1.

La précision de la synchronisation est de l'ordre de la dizaine de millisecondes par internet public et inférieur à la milliseconde pour un réseau Ethernet local.

PTP

Le protocole PTP [3] constitue une amélioration récente (2002) du NTP. Sa précision est inférieure à la microseconde. Toutefois il nécessite une mise à jour des équipements électroniques

3. Precise Time Protocol

(switches) par rapport au NTP sur tout le réseau concerné.

White-rabbit

La technologie white-rabbit à été initialement développée au CERN pour synchroniser les différents éléments de l'accélérateur de particule du LHC. Il est sous licence libre (open-source) et a donc trouvé d'autres domaines d'application comme la distribution de temps et de fréquence dans les réseaux Ethernet. Il utilise le protocole PTP par fibre optique ou câble Ethernet sur un réseau dédié (quelques dizaines de km). La précision atteinte avec ce dispositif est de l'ordre de la nanoseconde.

2.2 Les transferts sans média physique

Ce mode de transfert utilise généralement la propagation d'ondes électromagnétiques dans l'air.

2.2.1 Ondes radio-fréquences terrestres

Ces signaux sont émis par des stations radio grâce à des antennes possédant une très grande puissance (plusieurs kilowatts). Ces stations disposent obligatoirement d'une référence de temps locale (souvent une horloge à césium). Le délai entre la mise en forme du signal de référence local à émettre et la réception par l'utilisateur étant variable à court-terme (phénomènes atmosphériques par exemple) ce signal sert principalement à asservir un oscillateur (type quartz) afin de compenser ses dérives moyen et long terme. Ainsi tout le monde peut disposer (gratuitement du côté utilisateur, hormis le coût du module de réception), dans le rayon de propagation de ces ondes, d'une excellente référence de temps pour diverses applications.

Il existe de nombreuses stations émettrices réparties das le monde. A titre d'information, la France possède l'une des stations les plus puissante au monde (2 mégawatts) située a Allouis et également utilisée jusqu'à fin décembre 2016 pour émettre France Inter sur grandes ondes.

2.2.2 Micro-ondes par satellite

GNSS

La constellation GNSS [4] est composée de plusieurs dizaines de satellites en orbite autour de la terre. On distingue plusieurs sous-ensembles principaux selon l'appartenance de ces satellites : GPS américain (31 satellites), GLONASS russe (30 satellites), Beidou chinois (27 satellites), Galileo européen (30 satellites prévus) et IRNSS indien (7 satellites).

Son utilité initiale est la géolocalisation des récepteurs terrestres, pour cela ils véhiculent une information temporelle précise en embarquant des horloges atomiques (césium et rubidiums). C'est cette information temporelle qui nous intéresse dans le cadre du transfert de temps. Pour résumer le principe de fonctionnement on considère que la localisation GPS s'effectue en résolvant une équation à 4 inconnues (3 inconnues spatiales et 1 inconnue temporelle). Par conséquent il est indispensable d'avoir au moins 4 satellites interrogeables pour résoudre cette équation. On en déduit donc le décalage temporel entre l'oscillateur local (e.g. un quartz) et les oscillateurs embarqués dans les satellites. De plus chaque satellite transmet une information sur le décalage entre sa source de fréquence propre et le temps officiel (UTC). Ainsi on peut remonter jusqu'à l'échelle de temps « absolue » que constitue UTC et corriger notre oscillateur local en fonction.

4. Global Navigation Satellite System

FIGURE 2.4 – Station émettrice d'Allouis dans le cher (18)

Two Way Satellite Time and Frequency Transfert (TWSTFT)

Il s'agit ici d'un système de transfert de temps dédié au domaine Temps-Fréquence et bidirectionnel (contrairement au GPS, qui est unidirectionnel et possède d'autres utilités), ainsi un seul satellite suffit à estimer l'écart entre un oscillateur local et une référence terrestre distante sans passer par une horloge intermédiaire comme c'est le cas avec le GNSS. Ce moyen est utilisé pour comparer l'échelle de temps française UTC(OP) aux échelles de temps internationales UTC à travers le monde avec une incertitude inférieure à 2 nanosecondes.

Type de lien		Portée maximale	Incertitude	Coût (ordre de grandeur)
Physique	Câble coaxial	100m	faible	1 €
	Fibre optique	100km	faible	10 €
Sans fil	Radio	1500km	3ms	~k€
	GNSS	Globale	100ns	~M€
	TWSTFT	Globale	<2ns	~M€

FIGURE 2.5 – Tableau récapitulatif des moyens de transfert de temps et fréquence

2.3 Les échelles de temps

2.3.1 Temps atomique international

Le TAI est réalisé au Bureau International des Poids et Mesures à Paris. Plus de 400 horloges atomiques (masers, césiums et rubidiums) réparties sur 50 laboratoires à travers le monde y contribuent. Cette échelle de temps est uniquement basée sur des étalons atomiques.
Ces horloges sont comparées via GPS et transfert Two-Way. Ces comparaisons sont accumulées pendant une période de plusieurs jours et traitées [5] pour obtenir une échelle de temps la plus stable possible. Après chaque période d'accumulation l'écart entre le TAI et chacun des contributeurs est fourni par le BIPM. Ce type d'échelle de temps, qui n'est connue qu'a posteriori, est appelée échelle "papier".

2.3.2 UTC

Actuellement, le temps officiel mondial actuel (UTC) est corrigé en fonction de la durée du jour solaire vrai moyen par ajout d'une seconde intercalaire [6]. Cette seconde intercalaire permet de faire le lien entre l'échelle de temps atomique (signal physique) et l'échelle de temps papier qu'est UTC.

5. Chaque horloge est pondérée
6. en toute rigueur c'est l'observation d'étoiles lointaines qui induit l'ajout ou le retrait de cette seconde intercalaire

Chapitre 3

Les bruits des oscillateurs et leur caractérisation

Cette section décrit les différents concepts et outils mathématiques utilisés pour modéliser les instabilités des oscillateurs. Elle synthétise les connaissances issues des différents cours [6] et documentations disponibles afin de comprendre comment la qualité d'un oscillateur peut être estimée.

3.1 Prérequis

Avant toute chose il convient de savoir qu'en pratique il est impossible de mesurer exactement (avec une précision infinie) une grandeur. La seule information que l'on peut obtenir sera une estimation de celle-ci. Cette incapacité constitue le principal moteur de l'évolution technologique de l'humanité et, plus particulièrement, de la recherche scientifique qui tend à réduire l'écart entre les modèles et les observations.

Ceci est évidemment valable dans le domaine du Temps-Fréquence où l'établissement de modèles théoriques est indispensable à la représentation des phénomènes en jeu dans les oscillateurs mais dont les applications pratiques sont parfois impossibles (par exemple la notion d'infini dans les moyennages).

Pour palier ces imperfections on utilise donc des estimateurs qui, sous réserve d'hypothèses pertinentes concernant les phénomènes physiques, nous permettent « d'avoir une idée » des quantités observées.

Définitions mathématiques

Variable aléatoire temporelle : $x(t)$ est une variable aléatoire temporelle si à chaque instant t elle prend une valeur aléatoire unique.

Processus aléatoire : $X(t, \omega)$ est un processus aléatoire (ou stochastique) si pour toute valeur particulière ω_0 de ω, $X(t, \omega_0)$ est une variable aléatoire temporelle, ω étant l'ensemble des réalisations du processus $X(t, w)$. Pour une meilleur compréhension j'assimilerais ω_0 à l'identifiant d'un oscillateur particulier appartenant à un ensemble d'oscillateurs ω.

Moyenne temporelle : c'est la moyenne du processus aléatoire $X(t, \omega)$ effectuée sur tout les

temps t pour une réalisation particulière ω_0. Elle est notée $\bar{X}(t,\omega_0)$

$$\bar{X}(t,\omega_0) \quad = \quad \lim_{T\to\infty}\frac{1}{T}\int_{-T/2}^{+T/2} X(t,\omega_0)dt$$

Moyenne d'ensemble : c'est la moyenne du processus aléatoire $X(t,\omega)$ effectuée sur l'ensemble des réalisations ω à un instant t_0 fixé. Elle est notée $\left\langle X(t_0,\omega)\right\rangle$ et s'exprime sous la forme :

$$\left\langle X(t_0,\omega)\right\rangle \quad = \quad \lim_{N\to\infty}\frac{1}{N}\sum_{i=1}^{N} X(t_0,\omega_i)$$

Propriétés des processus aléatoires

Ergodicité : un processus aléatoire est ergodique si ses caractéristiques temporelles sont égales à ses caractéristiques d'ensemble. En d'autre terme sa moyenne d'ensemble est équivalente à sa moyenne temporelle.

Stationnarité : un processus aléatoire est stationnaire si ses caractéristiques statistiques sont invariantes par translation temporelle. En d'autres termes $\bar{X}(t,\omega_0) = \bar{X}(t+a,\omega_0) \quad \forall \quad a.$

3.2 Modélisation d'un oscillateur réel

Les perturbations pouvant affecter les signaux délivrés par un oscillateur sont multiples. Il peut s'agir de perturbations extérieures (températures, radiations, chocs) ou intrinsèques à l'oscillateur (vieillissement, instabilité de l'interrogation atomique, bruit électronique). Je ne traiterai pas ici des perturbations issues du transfert de ces signaux (fibre optique, transfert par satellite) car celles-ci constituent un domaine à part entière et leur impact peut être négligeable lorsqu'on se situe au plus proche des sources, ce qui est le cas dans le cadre de ma thèse.

On peut distinguer 2 principales caractéristiques permettant dévaluer la qualité d'un oscillateur : l'exactitude et la stabilité.

Pour illustrer ces 2 concepts différents mais indissociables voici un schéma explicatif faisant l'analogie avec un lancer de fléchette sur une cible :

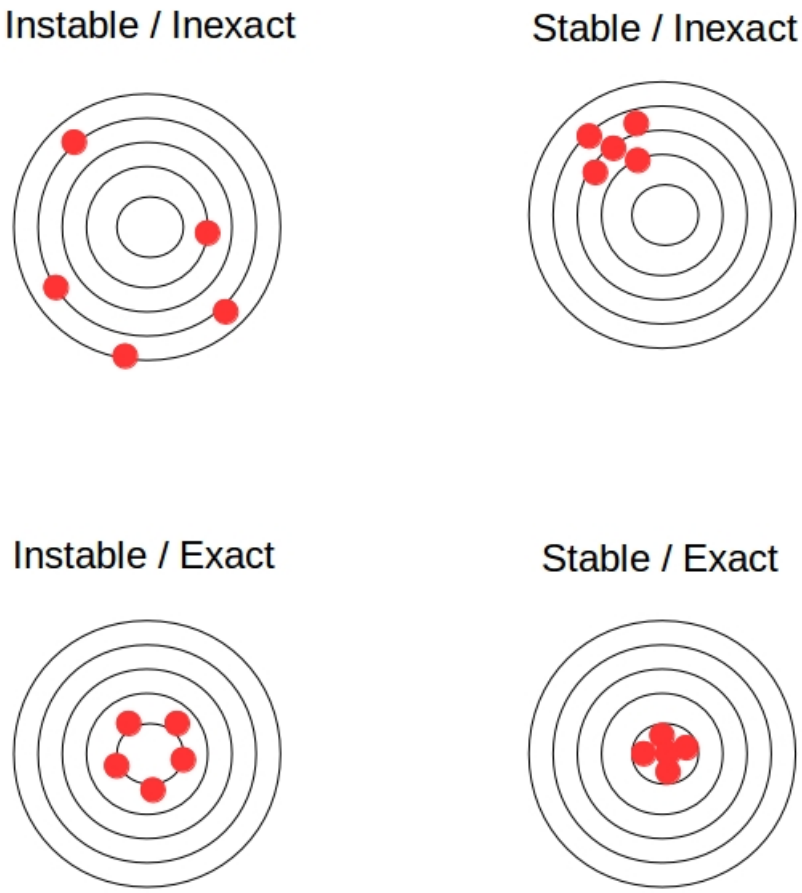


FIGURE 3.1 – Représentation conceptuelle de l'exactitude et de la stabilité

La stabilité serait assimilable à la répétition d'une valeur quelconque mais sans grandes variations à chaque lancer. L'exactitude correspond au degré de conformité à la définition de la grandeur considérée (définition césium de la seconde ou dans l'exemple le centre de la cible).

Il est indispensable de poser un modèle afin de pouvoir quantifier ces 2 valeurs et ainsi d'estimer la qualité des oscillateurs. La tension V(t) délivrée par un oscillateur idéal peut être modélisée selon l'équation suivante :

$$V(t) = V_0 \sin(2\pi\nu_0 t) \tag{3.1}$$

V_0 représente l'amplitude de la tension (en Volts) et ν_0 la fréquence nominale de l'oscillateur (en Hertz)

En introduisant les imperfections sur l'amplitude et la fréquence du signal on obtient :

$$V(t) = \big(V_0 + \varepsilon(t)\big) \sin\big(2\pi\nu_0 t + \Phi(t)\big) \tag{3.2}$$

$\varepsilon(t)$ représente le bruit d'amplitude, processus aléatoire autour de la valeur moyenne V_0
$\Phi(t)$ est le bruit de phase ou de fréquence

La première chose à noter est que le bruit de fréquence peut être assimilé à du bruit de phase car leur fluctuations sont indissociables et leur modèles équivalents, comme nous le verrons plus tard.

Le bruit d'amplitude est généralement négligé dans les mesures de stabilité de fréquence car il ne contribue pas directement à l'instabilité de l'oscillateur et, si ce dernier est de bonne qualité, ces contributions sont minimes devant le bruit de phase. Aussi, les composantes continues sont généralement filtrées grâce aux condensateurs utilisés dans l'électronique des oscillateurs ce qui tend à rendre la contribution du bruit d'amplitude négligeable sur le moyen et long terme.

Le modèle habituellement utilisé pour modéliser le bruit de phase est le suivant :

$$\Phi(t) = D_\varphi t^2 + \Delta\phi \sin(2\pi f_m t) + \varphi(t) \tag{3.3}$$

D_φ représente le coefficient de dérive de phase.
$\Delta\phi$ est l'amplitude des modulations de phase périodiques et f_m sa fréquence.
φ(t) inclut toutes les fluctuations de phase aléatoires.

Le terme de dérive de phase est modélisé par un polynôme d'ordre 2, ce qui correspond à une dérive de fréquence linéaire. On pourrait lui ajouter une terme constant φ_0 représentant le déphasage constant par rapport à l'oscillateur de référence (exactitude de phase) et un terme linéaire $\Delta\nu_0.t$ représentant l'écart moyen de fréquence (exactitude en fréquence) mais ces termes sont transparents pour l'estimation de la stabilité d'un oscillateur. En passant on remarquera l'explication mathématique de la différence entre stabilité et exactitude.

Ainsi, en considérant la partie aléatoire du bruit de phase, on modélise notre oscillateur par la relation suivante :

$$V(t) = V_0 \sin\left[2\pi\nu_0 t + \varphi(t)\right] \tag{3.4}$$

On va donc s'intéresser par la suite au bruit de phase $\varphi(t)$.

3.2.1 Fluctuations de phase et de fréquence dans le domaine temporel

La relation (3.4) peut s'écrire de la manière suivante :

$$V(t) = V_0 \sin\left[2\pi\nu_0(t + \frac{\varphi(t)}{2\pi\nu_0})\right] \tag{3.5}$$

On introduit ainsi la quantité x(t) appelée erreur de marche temporel ou écart de temps instantané :

$$x(t) = \frac{\varphi(t)}{2\pi\nu_0} \tag{3.6}$$

Cette quantité est homogène à un temps, c'est de cette valeur dont on parle lorsqu'on dit par exemple que notre montre avance ou retarde de 2 secondes (en prenant pour référence le temps officiel). Une façon d'estimer l'évolution du bruit de phase d'un oscillateur par rapport à une référence est donc d'observer l'évolution de cette grandeur.

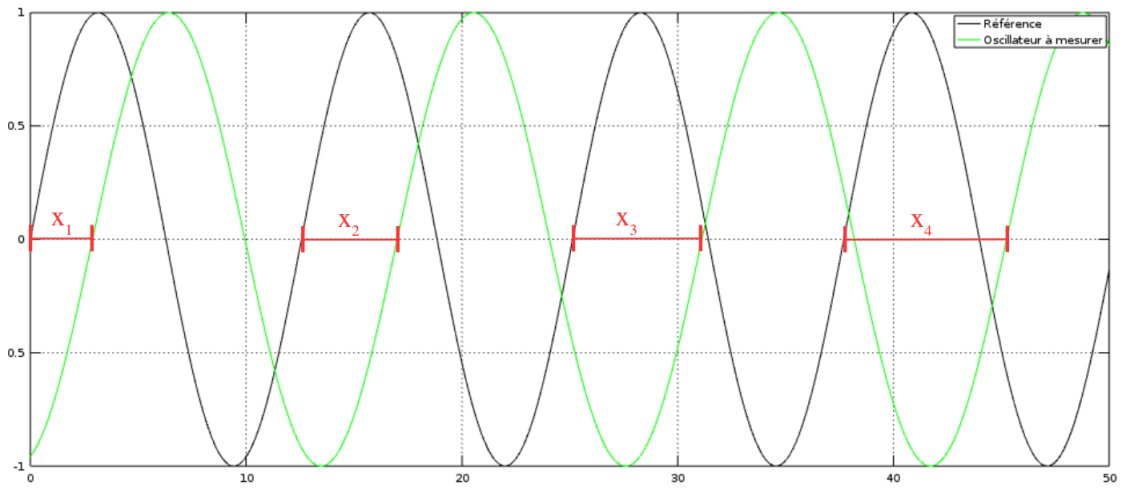


FIGURE 3.2 – Ecarts temporels d'une source par rapport à une référence (fronts montants)

En pratique l'instrument permettant cette mesure s'appelle un intervallomètre. Il mesure le temps écoulé entre le front montant de la référence et celui de l'oscillateur à mesurer.

Les fluctuations de phase peuvent aussi être caractérisées en considérant la fréquence instantanée du signal. Pour un signal représenté par la relation (3.4) elle s'exprime comme suit :

$$\nu(t) = \frac{1}{2.\pi}.\frac{d}{dt}\left[2.\pi.\nu_0.t + \varphi(t)\right] = \nu_0 + \frac{1}{2.\pi}.\frac{d\varphi(t)}{dt} \tag{3.7}$$

La fréquence instantanée d'un oscillateur est donc la somme de sa fréquence nominale ν_0 et d'un terme $\Delta\nu$ appelé bruit de fréquence (ou écart de fréquence).

$$\nu(t) = \nu_0 + \Delta\nu(t) \tag{3.8}$$

avec

$$\Delta\nu(t) = \frac{1}{2.\pi}.\frac{d\varphi(t)}{dt} \tag{3.9}$$

Il s'exprime en Hertz et dans le cas d'un oscillateur de relativement bonne qualité il est très faible comparé à sa fréquence nominale ν_0. A partir de cet écart de fréquence on définit la grandeur certainement la plus utilisée dans le domaine temps-fréquence qui est l'écart de fréquence instantané normalisé $y(t)$ et qui est défini comme suit :

$$y(t) = \frac{\Delta\nu(t)}{\nu_0} \tag{3.10}$$

Cette grandeur est sans dimension, ce qui lui donne l'avantage d'être un outil de mesure afin de comparer différents oscillateurs opérant à des fréquences différentes.

Les relations (3.6), (3.9) et (3.10) nous permettent d'établir une relation entre les 4 grandeurs temporelles φ(t),$x(t)$, $\Delta\nu(t)$ et $y(t)$ comme résumé sur le schéma suivant :

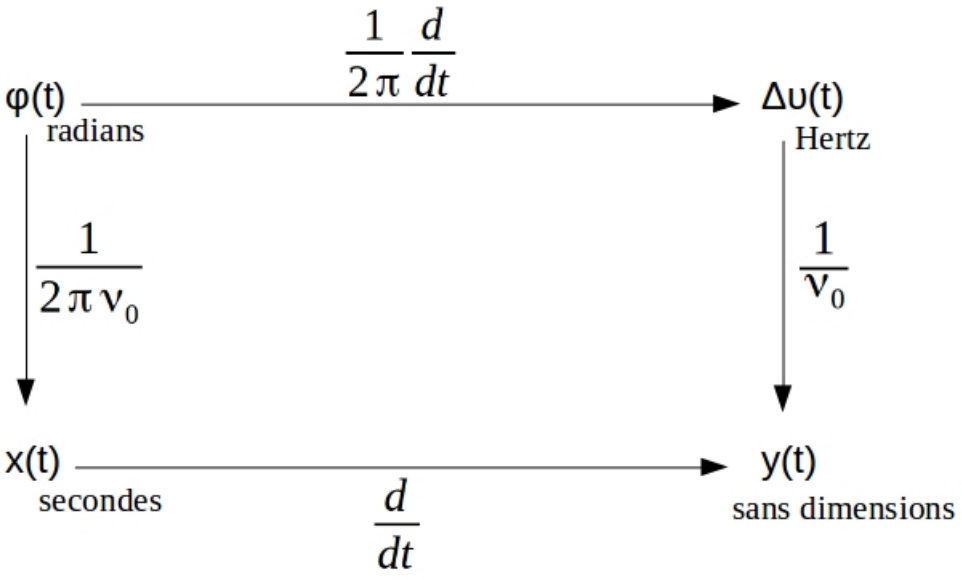


FIGURE 3.3 – Relations entre les 4 grandeurs fondamentales dans le domaine temporel

3.2.2 Fluctuations de phase et de fréquence dans le domaine fréquentiel

Ayant effectué assez peu d'analyse spectrale au cours de ma thèse je vais me contenter d'introduire uniquement les grandes notions qui me semblent fondamentales dans ce domaine.

Énergie et puissance

L'énergie totale d'un signal $x(t)$ s'exprime sous la forme :

$$E_x = \int_{-\infty}^{+\infty} x^2(t)dt \tag{3.11}$$

Dans le cas où $x(t)$ est modélisé par une sinusoïde (bruitée), comme c'est le cas pour la plupart des signaux délivrés par des oscillateurs, cette énergie est infinie. En pratique ces signaux sont toujours bornés dans le temps mais si on veut s'affranchir de toute considération de durée on préférera considérer la puissance de ces signaux qui, dans la plupart des cas, est finie.

Sa puissance moyenne s'exprime sous la forme :

$$P_x = \lim_{T\to\infty}\left[\frac{1}{T}\int_{-\infty}^{+\infty} x^2(t)dt\right] \tag{3.12}$$

Fonction d'autocorrélation

Pour une variable aléatoire stationnaire $x(t)$ de puissance finie, sa fonction d'autocorrélation $R_x(t)$ est définie par

$$R_x(t) = \left\langle \lim_{T\to\infty}\left[\frac{1}{T}\int_{-T/2}^{+T/2} x(\theta).x(\theta - t)d\theta\right]\right\rangle \tag{3.13}$$

Cette fonction, bien qu'issue d'une variable aléatoire n'est pas, en théorie, aléatoire. Elle permet de mettre en évidence certains phénomènes répétés et d'extraire des tendances déterministes.

On définit à partir de cette fonction la densité spectrale comme étant sa transformée de Fourier.

$$S_x(f) = TF\big[R_x(t)\big] = \left\langle \lim_{T\to\infty} \left[\frac{1}{T} \left| \int_{-T/2}^{+T/2} x(t) e^{-j2\pi f t} dt \right|^2 \right] \right\rangle \tag{3.14}$$

Relations entre densités spectrales

De la même façon que dans le domaine temporel voici un schéma permettant de relier les densités spectrales des 4 mêmes grandeurs fondamentales [6] :

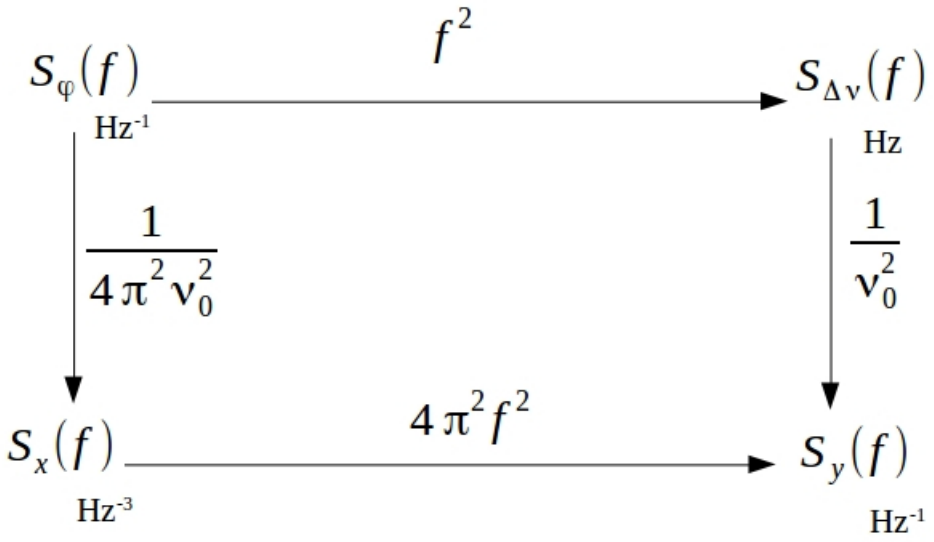


FIGURE 3.4 – Relations entre les 4 grandeurs fondamentales dans le domaine fréquentiel

3.3 Modèle de bruits en lois de puissance

Comme mentionné dans le chapitre 2, la quantité favorite du domaine temps fréquence est l'écart de fréquence instantané normalisé. On va donc s'intéresser à sa densité spectrale afin de modéliser les différents types de bruits affectant cette variable. A noter que ses dérives linéaires ne seront pas intégrées dans ce modèle car ce sont des phénomènes déterministes traités à part (cf relation (3.3)). On considère alors que, dans les oscillateurs classiques, la densité spectrale $S_y(f)$ est la somme de 5 bruits indépendants correspondant chacun à une loi de puissance :

$$S_y(f) = \sum_{i=-2}^{+2} h_i f^i \tag{3.15}$$

On classe ici le type de bruit en fonction de l'indice i :

i	Nom du bruit	Origine probable
-2	Marche aléatoire de fréquence	Environnement
-1	Bruit de scintillation de fréquence	Résonateur
0	Bruit blanc de fréquence	Bruit thermique
1	Bruit de scintillation de phase	Bruit électronique
2	Bruit blanc de phase	Bruit externe

FIGURE 3.5 – Bruits usuels du modèle en lois de puissances

3.3.1 Méthodes pratiques d'analyse du bruit

Conformément aux 2 approches précédentes (temporelle et fréquentielle) des modèles de bruit, on fait correspondre 2 méthodes d'analyse dans chacun de ces domaines. Dans le domaine temporel on utilisera les variances issues des mesures de l'écart de fréquence instantané normalisé et dans le domaine fréquentiel on utilisera un banc de mesure de bruit de phase.

Les variances

Dans cette étude on considérera que les bruits étudiés suivent des lois de probabilités gaussiennes. La densité de probabilité d'une variable aléatoire gaussienne x(t) s'exprime sous la forme :

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} exp\left[-\frac{1}{2}\left(\frac{x- < x >}{\sigma}^2\right)\right] \tag{3.16}$$

avec σ son écart type et $< x >$ sa moyenne d'ensemble

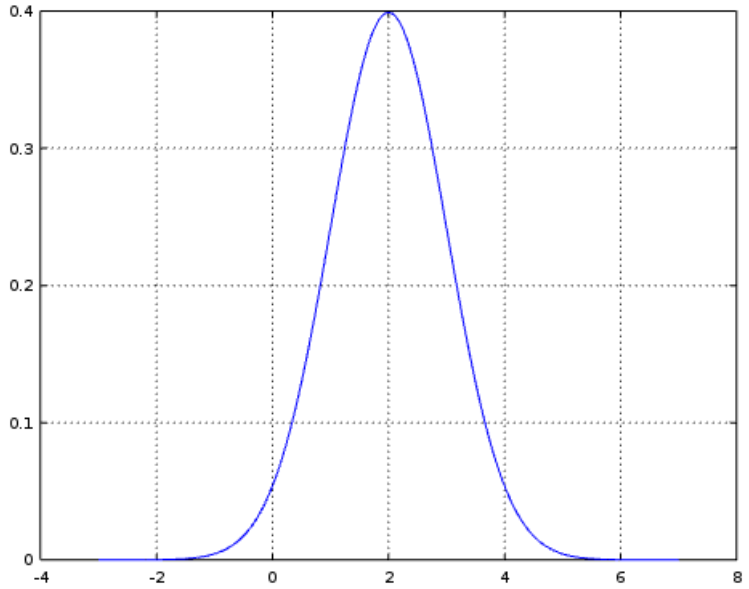


FIGURE 3.6 – Densité de probabilité pour une valeur moyenne 2 et une variance de 1

Variance vraie

Si on considère une séquence de mesures d'écarts de fréquence instantanés normalisés $\bar{y}_k$ supposés stationnaires et ergodiques alors on peut définir leurs variance vraie, calculée à partir d'un nombre infini de ces échantillons moyennés sur une durée τ :

$$I^2(\tau) = \left\langle (\bar{y}_k - < \bar{y}_k >)^2 \right\rangle \tag{3.17}$$

En pratique il est impossible de calculer cette variance vraie car elle implique un nombre infini d'échantillons. Une fois de plus on va donc utiliser une estimation de cette grandeur afin de pouvoir en réaliser une implémentation concrète.

Estimation de la variance vraie – la variance d'Allan Lorsque le nombre d'échantillons $\bar{y}_k$ est important, mais fini, il semble légitime de considérer l'estimateur suivant comme correct :

$$\sigma^2_{(1)}(N,\tau) = \frac{1}{N}\sum_{i=1}^{N}\left[\bar{y}_i - \frac{1}{N}\sum_{j=1}^{N}\bar{y}_j\right]^2 \tag{3.18}$$

Cependant il à été montré [7] que la valeur de variance la plus probable est donnée par l'estimateur suivant :

$$\sigma^2_{(2)}(N,\tau) = \frac{1}{N-1}\sum_{i=1}^{N}\left[\bar{y}_i - \frac{1}{N}\sum_{j=1}^{N}\bar{y}_j\right]^2 \tag{3.19}$$

Le sous-comité IEEE pour la stabilité de fréquence a fixé $N = 2$ afin d'uniformiser les méthodes de caractérisation des oscillateurs à travers le monde. Ainsi on définit la variance à 2 échantillons aussi appelée variance d'Allan [8] :

$$\sigma^2_y(\tau) = \frac{1}{2}\left\langle \left(\bar{y}_2 - \bar{y}_1\right)^2 \right\rangle \tag{3.20}$$

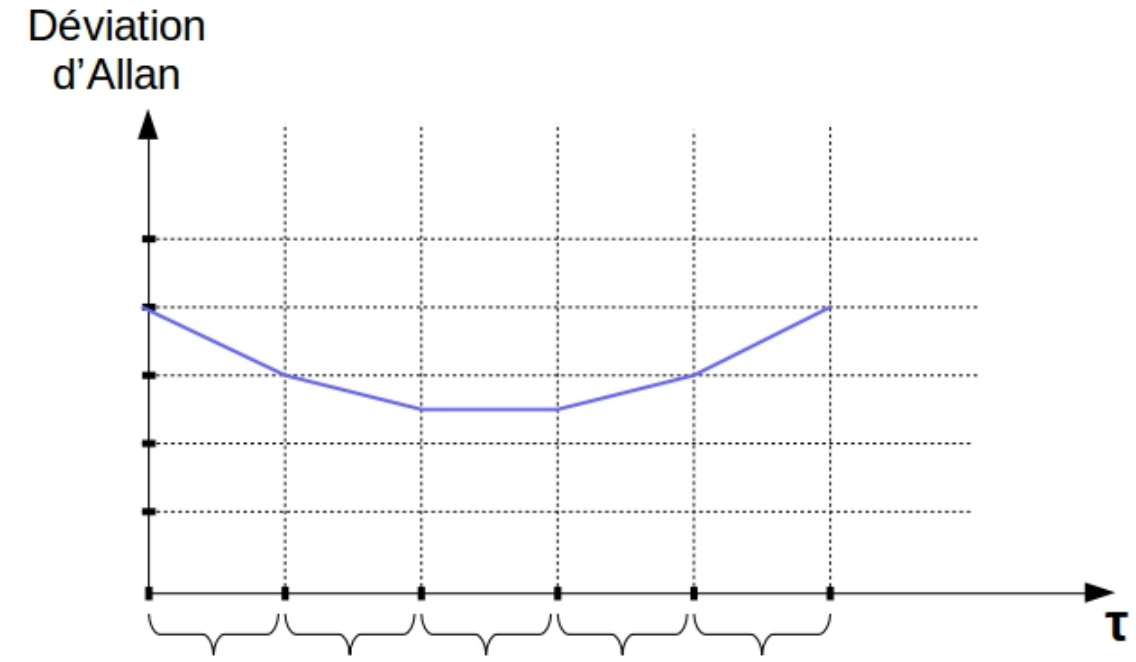
En pratique on préférera utiliser la racine carrée de cet estimateur appelé déviation d'Allan (anglicisme venant du fait, qu'en anglais, écart-type se dit deviation) donnant directement une idée de l'évolution de l'écart de fréquence instantané normalisé.

Il s'agit de l'outil de caractérisation que j'ai le plus utilisé dans le cadre de ma thèse. Cet outil est particulièrement adapté pour mettre en évidence les bruits basses fréquences d'un signal (f^0, f^{-1} et f^{-2} en terme de lois de puissance) ainsi que la dérive de fréquence. Mais il ne permet pas de distinguer le bruit blanc de phase (f^{+1}) du bruit de scintillement de phase (f^{+2}), l'influence de ces bruits diminuant en même temps que la durée d'intégration ils ne contribuent pas directement aux bruits que je cherche à minimiser.

Le choix de cet estimateur de la variance vraie étant arbitraire et imparfait, d'autres estimateurs (de la variance vraie) ont vu le jour afin de correspondre aux types de bruits à mettre en évidence (variance d'Allan modifiée, variance d'Hadamard, variance de Picinbono, variance parabolique, ...).

Interprétation de courbes de variance d'Allan Cette courbe représente l'évolution de la racine carré de la variance à 2 échantillons de l'écart de fréquence instantané normalisé en fonction de la durée d'intégration. C'est l'outil de mesure de stabilité le plus largement répandu dans le domaine Temps-Fréquences des laboratoires et de l'industrie.

Cette courbe permet de connaître directement les ordres de grandeurs des fluctuations de l'écart de fréquence sur une durée considérée. On peut également déduire la nature des bruits



Pente	τ^{-1}	$\tau^{-1/2}$	τ^{0}	$\tau^{1/2}$	τ^{1}
Type de bruit	Fliker de phase + Bruit blanc de phase	Bruit blanc de fréquence	Flicker de fréquence	Random walk de fréquence	Dérive de fréquence

FIGURE 3.7 – Déviation d'Allan en échelle log-log (tracé générique)

présent dans l'oscillateur en fonction des pentes asymptotique du tracé expérimental.

On peut voir que l'influence des bruits (flicker et bruit blanc) de phase et du bruit blanc de fréquence diminue avec la durée d'intégration.

Le bruit de scintillement (flicker) de fréquence constitue généralement le meilleur domaine de stabilité de l'oscillateur. On appelle cette zone le palier de flicker.

Les bruits venant dégrader la stabilité de l'oscillateur (pente positive) de manière la plus significative sur le moyen et long terme sont la marche aléatoire de fréquence et la dérive de cette dernière. C'est donc l'influence de ces 2 types de bruits que je chercherai à compenser au mieux dans le cadre de ma thèse.

Voici une synthèse des relevés de stabilité de différents oscillateurs libres (non-corrigés) effectués grâce au TSC5110A de TimingSolutions [1].

1. Le TSC5110A permet de tracer directement les courbes de variance d'Allan entre un signal et une référence

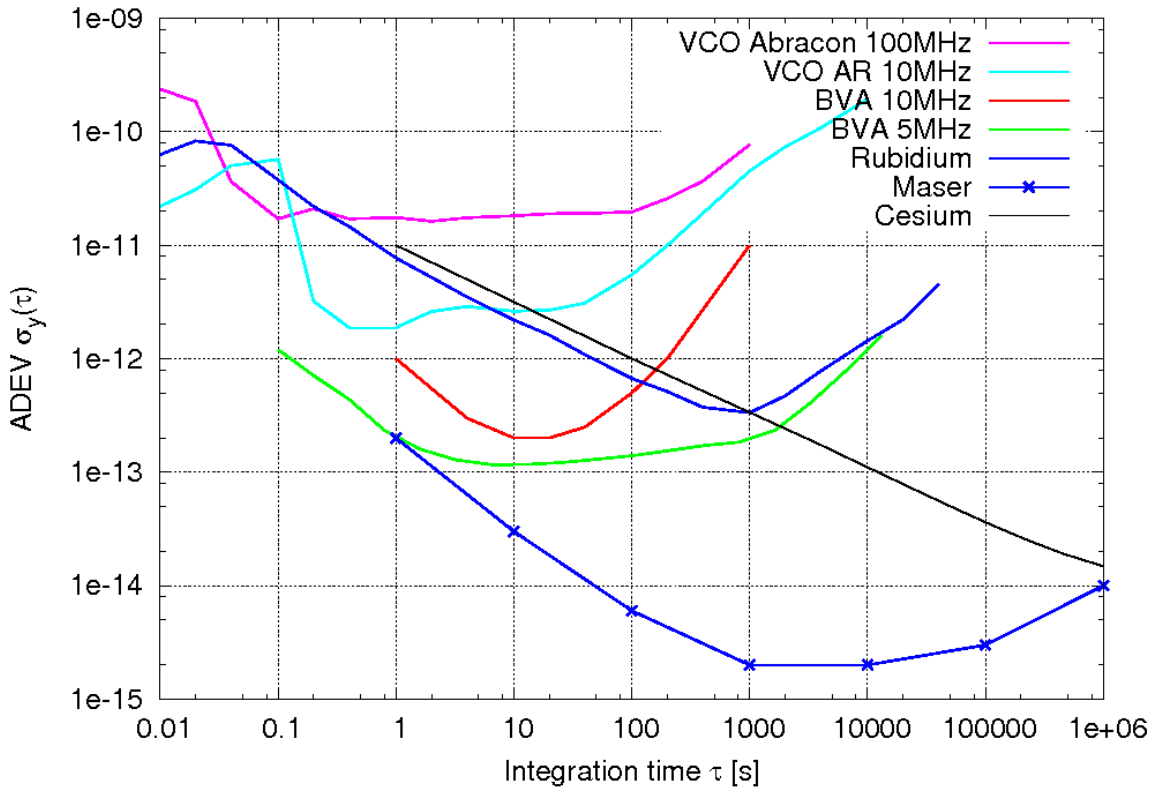


FIGURE 3.8 – Déviation d'Allan de diverses sources de fréquence

Ces courbes ont toutes pour référence le signal provenant du maser car sa stabilité est la meilleure entre 1 seconde et 1 semaine. La courbe du maser est tirée des spécifications du constructeur.

Les 2 premières sources (Abracon 100MHz et AR 10MHz) sont des oscillateurs à quartz commerciaux, ils coûtent une centaine d'euros et ont la dimension d'une petite boite d'allumettes (quelques cm^3).

Les 2 sources suivantes (BVA10MHz et BVA 5MHz) sont aussi des oscillateurs à quartz mais leur régulation thermique ainsi que la coupe du cristal les rendent plus performants que les 2 précédents, ce qui leur donne une meilleure stabilité à court et moyen terme. Ils coûtent environ 10.000 euros et occupent un volume d'environ $1dm^3$.

Les 2 dernières sources sont le césium et le maser. Tout deux sont des horloges atomiques coûtant respectivement 50.000 et 100.000 euros pour un encombrement de plusieurs dm^3.

Chapitre 4

Principe d'une horloge composite

4.1 Principe général

Par définition, une horloge composite est constituée d'un oscillateur pilotable en fréquence asservi sur une ou plusieurs sources. Habituellement on utilise un VCO (Voltage Controlled Oscillator, typiquement un quartz) dont la fréquence du signal de sortie est proportionnelle à sa tension d'entrée. Il est également possible d'utiliser un DDS [1] afin de générer le signal de sortie mais cette piste n'a pas été investiguée dans le cadre de ma thèse [2].

Le défi dans ce genre de système est double :

. D'un côté le hardware. Il s'agit d'estimer le plus précisément et rapidement possible les écarts de fréquence entre les sources et le VCO. Les évolutions technologiques et la perpétuelle montée en performances des composants électroniques permettent des améliorations permanentes dans ce domaine. Cette partie peut se séparer en 2 : le hardware "brut" constitué par les composants assurant l'interface entre le monde analogique (sortie des sources de fréquence) et numérique (signaux "compréhensibles" par des systèmes électroniques). L'autre sous partie est le hardware "mou" qui assure la mise en forme des signaux numériques afin d'en extraire les informations pertinentes (dans notre cas il s'agit d'écarts temporels ou de fréquence), c'est le code FPGA.

. D'un autre côté le software. Il faut combiner les informations issues du hardware afin de piloter au mieux le VCO. Ce sera le rôle de l'algorithme d'asservissement, qui constitue le cœur de ma thèse. Il existe de nombreux algorithmes d'asservissement dans la littérature et chacun possède ses défauts et ses avantages. De plus il peut intégrer la détection d'éventuels dysfonctionnements des sources et doit réagir en conséquence (minimiser la contribution voire carrément exclure certaines sources).

1. Direct Digital Synthesizer
2. L'ancien modèle d'horloge composite développé à l'Observatoire utilisait un quartz qui est, en principe, moins affecté par du bruit blanc de phase

Afin de résumer ce qui a été exposé précédemment, la figure 4.1 présente un schéma récapitulatif.

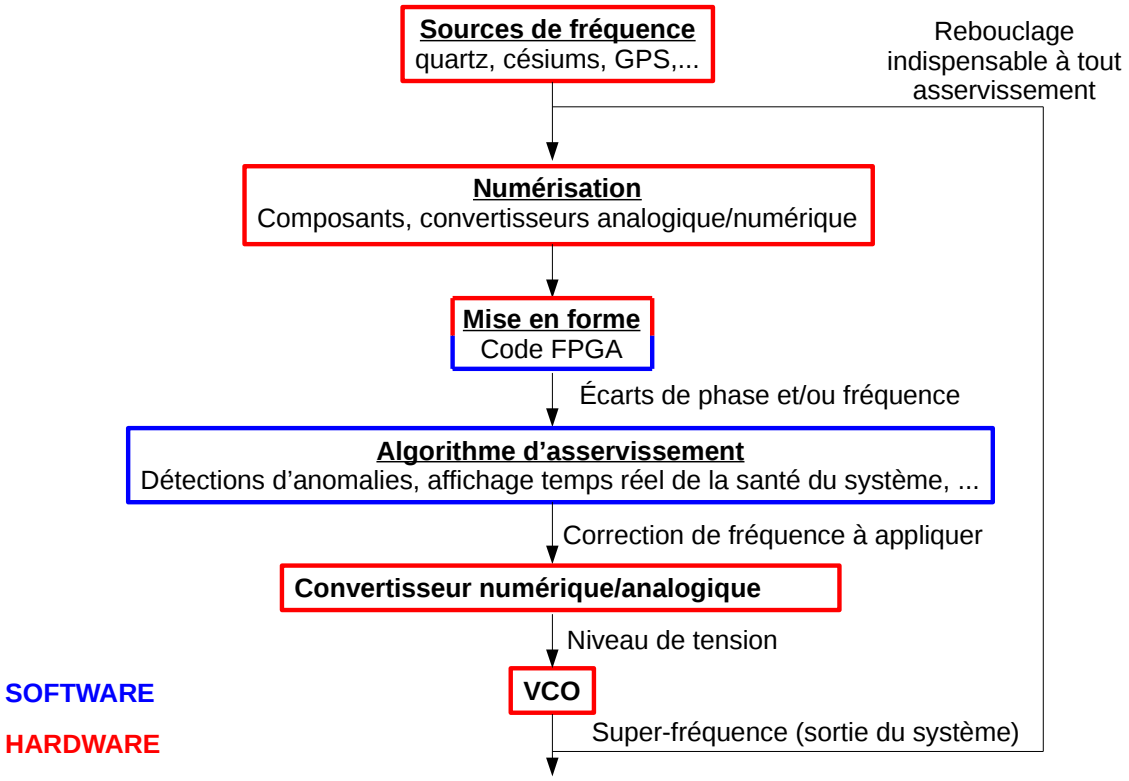


FIGURE 4.1 – Horloge composite générique

4.2 Ancienne horloge

L'ancienne horloge composite conçue et réalisée par C.Plantard (thèse), P.M.Mbaye (thèse) et O.Pajot [9] se constitue d'un VCO à quartz stable sur le court terme ($\tau < 10^2$s) asservi sur un maser passif stable à moyen terme ($10^2 s < \tau < 10^6$s) et un césium stable à long terme ($\tau > 10^6$s).

Deux prototypes ont été réalisés. Le premier avec des sources commerciales et le second avec des sources plus performantes. Cependant l'architecture quartz/maser/césium reste figée car les domaines de stabilité de ces sources sont parfaitement complémentaires.

C'est sur cette horloge que j'ai pu me "faire la main" et appréhender les différentes problématique d'une horloge composite.

4.2.1 Principe

Tout d'abord une fréquence de battement est créée en multipliant les signaux de chaque source (100 MHz) avec le signal issu d'un oscillateur décalé (100,0007 MHz).

$$\cos A \times \cos B = \frac{1}{2}[\cos(A-B) + \cos(A+B)]$$

Le signal ainsi obtenu comporte donc la fréquence somme (200,0007 MHz) et la fréquence différence (700 Hz).

Ce signal traverse ensuite un filtre passe-bas qui permet de rejeter la fréquence haute (200,0007 MHz) et de conserver uniquement la fréquence basse (700 Hz). Ce signal possède les mêmes propriétés de stabilité que les références (l'influence de l'oscillateur décalé est supprimée par les inter-comparaisons effectuées par l'algorithme d'asservissement).

Cette technique de démultiplication est nommée DMTD (Dual Mixer Time Difference) et est couramment utilisée dans différents domaines, elle se comporte comme une "boite de vitesse" pour passer de 100 MHz à 700 Hz, ainsi l'on peut facilement mesurer les écarts temporels entre nos différentes sources.

L'ultime étape avant de pouvoir échantillonner nos signaux dans une unité de contrôle/commande numérique (type FPGA/PC) est de transformer ces signaux sinusoïdaux centrés en zéro en signaux logiques (carrés 0V-3,3V). Pour cela on utilise 4 étages d'amplificateurs pour rendre nos signaux carrés et un optocoupleur afin d'obtenir des pentes les plus raides possible et un signal entre 0 Volts (0 logique) et 3,3 Volts (1 logique). Cet ensemble constitue un convertisseur analogique/numérique nommé ZCD (Zero Crossing Detector).

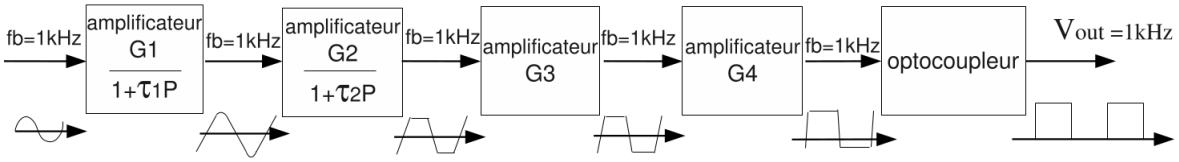


FIGURE 4.2 – Principe des ZCD

La figure 4.2 schématise les ZCD, la fréquence de battement (notée fb) est ici de 1kHz mais lorsque j'ai manipulé cette horloge j'ai mesuré cette fréquence autour de 680 Hz. Cette variation peut être expliquée par le vieillissement de l'oscillateur décalé pendant la période d'inactivité de l'horloge (> 4ans).

Ces signaux sont injectés dans le FPGA (Cyclone III d'Altera), dont l'objectif est de fournir à l'algorithme les écarts temporels entre les sources (maser et césium) et le VCO.

L'algorithme d'asservissement compile tout les écarts de fréquence et en déduit une commande à appliquer au VCO. La correction de fréquence prend en compte les variations à moyen terme (constante de temps d'1s) du maser et les variations à long terme (constante de temps de l'ordre de la semaine) du césium.

Cette commande est convertie en tension par l'intermédiaire d'un carte de conversion numérique-analogique. Ainsi la fréquence du VCO est modifiée puis réinjectée en entrée de l'horloge composite. La boucle est bouclée.

La figure 4.3 résume le principe de fonctionnement de l'ancienne horloge composite.

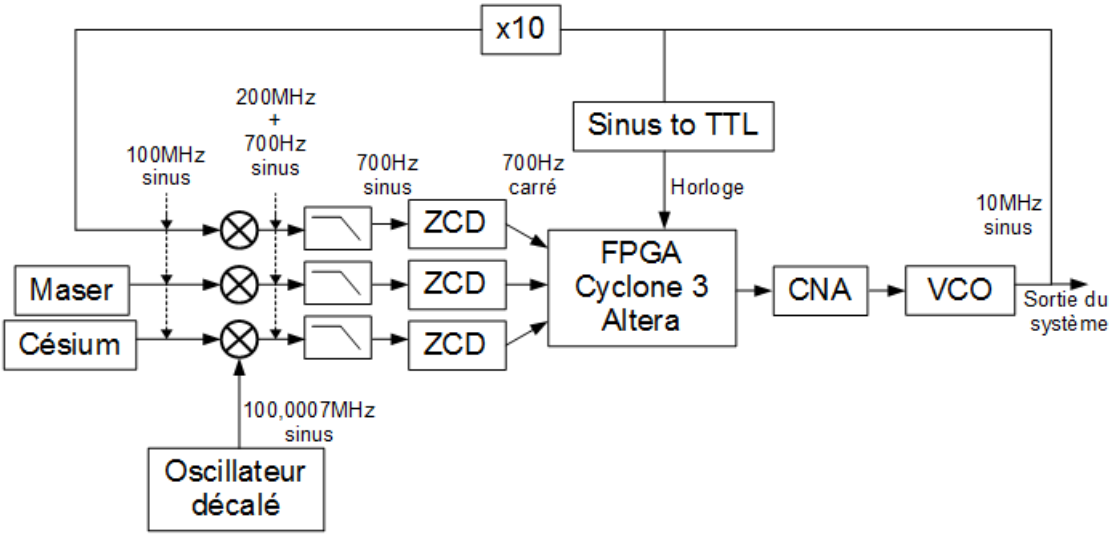


FIGURE 4.3 – Ancienne horloge composite

4.2.2 Limitations de l'ancienne horloge

. Les multiplicateurs et les ZCD sont des éléments analogiques qui ont une grande sensibilité à la température ambiante, surtout les ZCD qui sont les éléments les plus critiques de l'ancienne horloge composite. Ainsi une légère variation de température peut faire dévier la valeur des écarts temporels et induire une dérive de la fréquence du VCO.

. L'architecture hardware (quartz, maser et césium) et software correspondante (constantes de temps de l'asservissement) est figée, cette horloge est donc utilisable uniquement avec les sources citées précédemment.

4.3 Objectifs de la nouvelle horloge composite

. Échantillonner au plus tôt les signaux afin de s'affranchir des éléments analogiques sensibles aux conditions extérieures.

. Concevoir une architecture souple permettant d'utiliser plusieurs sources (>5) de nature (domaine de stabilité) et fréquences différentes.

. L'utilisation de sources de même nature crée de la redondance (robustesse) et une potentielle amélioration de la stabilité de fréquence. Idéalement on cherche à atteindre les performances d'une source coûteuse (maser) en combinant plusieurs sources moins onéreuses (5 à 6 rubidiums).

. L'algorithme d'asservissement doit donc s'adapter automatiquement aux sources présentes en entrée du système et s'auto-configurer en fonction de celles-ci (détection d'anomalies, ré-ajustage en temps réel des différents paramètres). De plus certains effets déterministes,

notamment les dérives de fréquence peuvent être quantifiés et donc compensés, l'algorithme devra donc être prédictif.

J'ai fait le choix de décrire mon travail de thèse selon les 2 axes qui me semblent fondamentaux et indissociables dans l'élaboration d'une horloge composite :

- Le premier axe intitulé "hardware" concernera l'ensemble des éléments nécessaires à l'échantillonnage des signaux analogiques (e.g. 10MHz) issus des différentes sources de fréquences. Ces éléments ont pour rôle de mesurer, mettre en forme et transférer des informations sur les fréquences des sources à la partie logicielle
- La partie "software" (logicielle) a pour rôle de synthétiser ces informations au mieux et d'en déduire une correction de fréquence à appliquer à l'oscillateur local.

Deuxième partie

Hardware

Chapitre 5

Cartes de développement et prototypes

Ce chapitre va décrire les dispositifs et composants électroniques utilisés dans les différentes versions des prototypes développés, ce qui regroupe les cartes de développement (FPGA) et les cartes développées spécifiquement pour notre application.

J'ai utilisé principalement 3 cartes lors du déroulement de ma thèse :

— La RedPitaya
Dans un premier temps, afin de commencer à appréhender le système dans sa globalité et en utilisant des outils déjà développés (code FPGA, distribution Linux). J'ai uniquement développé l'application logicielle réalisant la récupération des données, leur traitement et le contrôle du quartz local.

— La ZedBoard
Utilisée pour développer un premier prototype fonctionnel en créant la distribution Linux (Buildroot) et le code FPGA (Vivado) en plus de l'application logicielle (langage C).

— La Microzed
Pour une version plus compacte du prototype précédent et aussi moins coûteuse [1] dans un objectif d'industrialisation du système.

Ces 3 cartes sont toutes basées sur un SoC [2] FPGA Zynq de Xilinx.

5.1 Présentation des puces Zynq

Les puces Zynq sont des SoC composés d'une partie processeur (dual core ARM cortex A9) et d'une partie FPGA (technologie 28nm Artix-7 ou Kintex-7). Elles sont développées par Xilinx et sont ensuite intégrées à des cartes de développement par les distributeurs (Avnet, Digilent, ...) pour créer plusieurs gammes de produits.

1. Surtout concernant le VCO utilisé
2. System on Chip

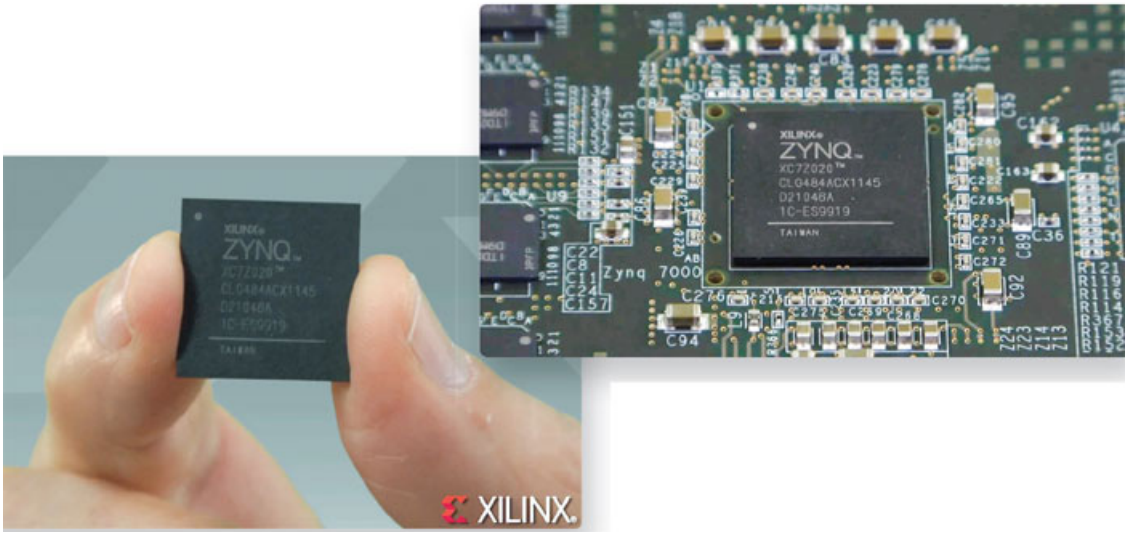


FIGURE 5.1 – Puces Zynq de Xilinx

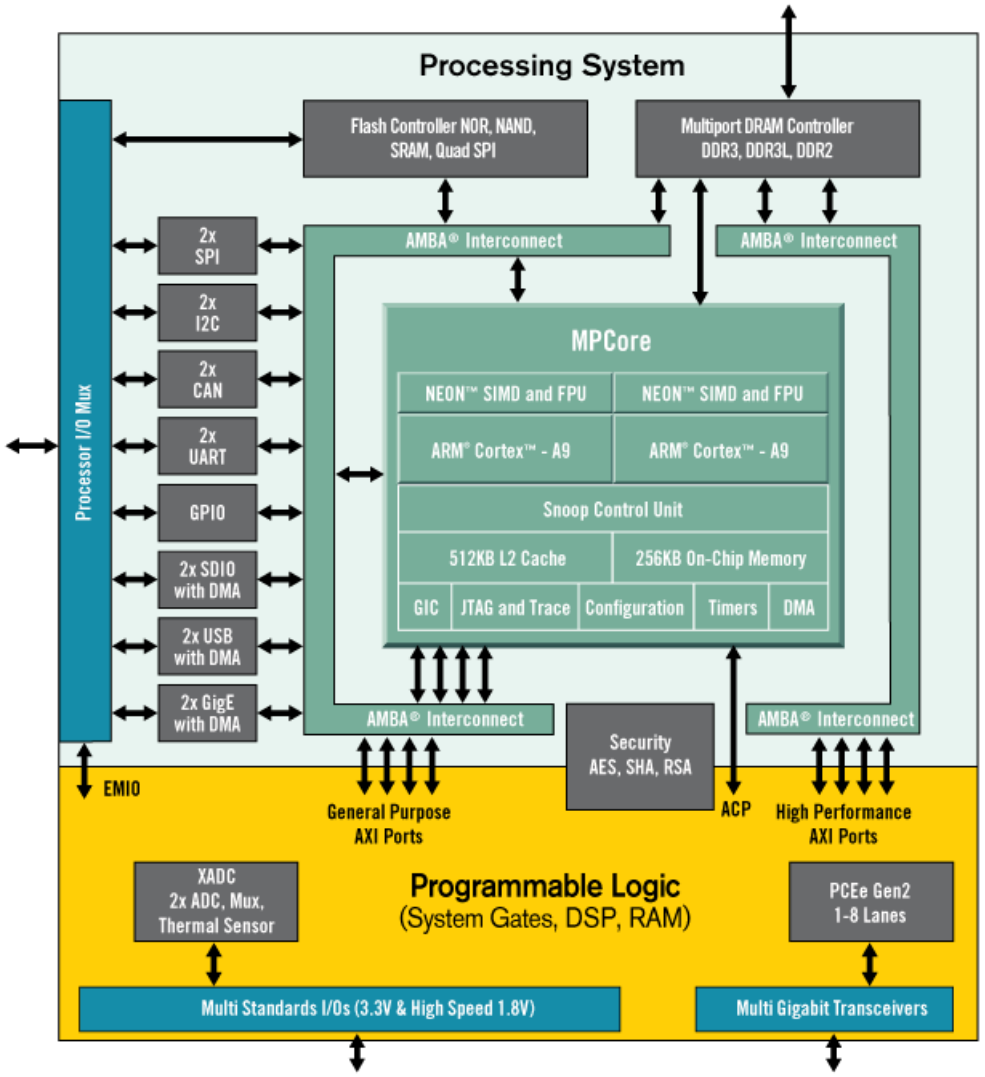


FIGURE 5.2 – Architecture des puces Zynq

La présence d'une partie processeur est une véritable révolution dans le domaine des FPGA, elle permet l'exécution autonome des applications grâce à la possibilité d'y configurer un système d'exploitation Linux [3]. Avant son apparition, les FPGA avaient besoin d'être connectées en permanence à un ordinateur afin de transférer leur données ou d'interagir avec l'application logicielle.

Ces puces sont utilisées par différents distributeurs (Avnet, RedPitaya, Digilent) afin de les intégrer dans une architecture électronique complète permettant d'exploiter pleinement le Zynq (périphériques, mémoires, GPIO).

Ces dispositifs électroniques, souvent appelés tout simplement FPGA, sont des cartes de développement permettant de réaliser de nombreuses applications. Elle disposent d'une connectique plus ou moins fournie faisant le lien avec le monde extérieur et d'une partie programmable permettant d'implanter la fonction souhaitée.

La programmation FPGA se situe à la frontière entre le développement logiciel classique et la

3. La description complète de la création de la distribution Linux est donnée en annexe

conception de circuits électroniques. Son principal avantage est de pouvoir créer des prototypes de circuit électroniques sans avoir à souder des composants, ce qu'on appelle configuration hardware.

5.2 RedPitaya

Cette carte constitue l'entrée de gamme des cartes de développement FPGA à base de puce Zynq (version 7Z010). Elle embarque 2 convertisseurs d'entrée analogiques/numériques rapides (125 millions d'échantillons par seconde sur 14 bits), 2 sorties analogiques et un oscillateur à quartz local contrôlable.

J'ai utilisé cette carte au commencement de ma thèse. Elle à l'avantage d'intégrer l'ensemble des fonctionnalités nécessaires à l'élaboration d'un premier système d'asservissement.

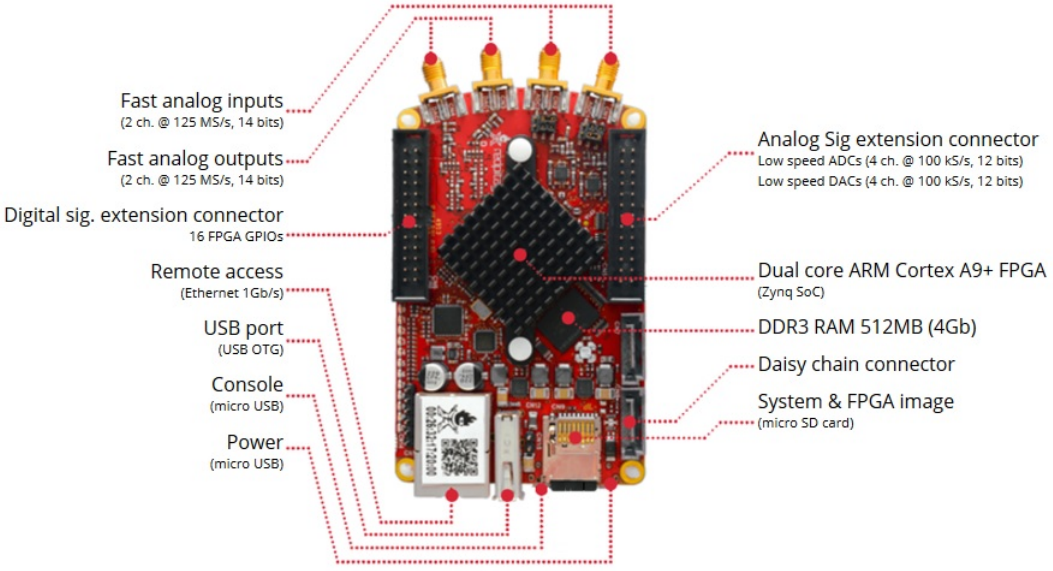


FIGURE 5.3 – Carte de développement RedPitaya

5.2.1 Échantillonnage rapide

Au départ du projet il était prévu d'échantillonner directement les signaux issus des sources de fréquence, ce qui implique l'utilisation d'un CAN [4] suffisamment rapide (au moins 4 fois la fréquence des sources) par source avec une horloge commune. Ainsi on peut reconstruire nos sinusoïdes sur un même intervalle de temps et en déduire l'écart de fréquence entre les sources.

La RedPitaya possédant 2 CAN fonctionnant à 125 MS/s elle à servi à tester cette méthode de mesure de fréquence directement sans avoir à développer une carte d'acquisition spécifique et en utilisant un code FPGA générique fourni avec la carte.

L'utilisation de cette méthode de mesure dans le cadre du projet n'a pas été validée pour plusieurs raisons :

4. Convertisseur Analogique/Numérique

. Les signaux des sources peuvent être de différente nature (PPS, 100MHz, signaux logiques) et pas forcément sinusoïdaux. De plus, pour échantillonner un signal à 100MHz il faudrait utiliser un CAN fonctionnant à plus de 400 MHz.

. Le nombre d'entrées désiré étant d'au moins 5, cela implique d'utiliser 5 CAN ayant tous une horloge commune. Ce qui pourrait poser des problèmes de distribution des signaux d'horloge à cause des délais de propagation et engendrer des biais variables entre les moments où sont échantillonnées les différentes sources.

. Les 5 (ou plus) CAN codant leurs informations chacun sur 14 bits à une cadence supérieure à 400 MHz, cela engendre un débit de 28 Gb/s (environ 3 Go/s) de données à traiter par le Zynq. Ceci impliquerait un long délai de traitement et augmenterait la périodicité de correction de VCO local.

L'utilisation de CAN rapides n'a ainsi pas été retenue pour la suite de mes travaux, le développement sur la carte RedPitaya a alors été arrêté pour se concentrer sur une autre méthode d'acquisition des signaux.

5.2.2 SINITER

Cette section constitue un aparté software au sein de cette partie hardware car l'algorithme décrit n'a été implanté que sur la carte RedPitaya et n'a pas été retenu pour la version finale du système développé.

L'algorithme SINITER [5] a pour fonction d'ajuster un modèle mathématique sinusoïdal sur un relevé d'échantillons. C'était la méthode initialement prévue pour estimer les paramètres utiles des sources (phase et fréquence). Cette méthode est basée sur un échantillonnage rapide des sources nécessitant des ADC fonctionnant au moins à 2 fois la fréquence maximale contenue dans le signal à échantillonner [6].

Cette méthode de mesure à été implantée sur la carte de développement RedPitaya qui possède 2 ADC cadencés à 125 MS/s, permettant ainsi l'échantillonnage de signaux de fréquence nominale 10MHz. La carte SD fournie avec la carte possède la distribution Linux, le code FPGA et l'application logicielle permettant l'acquisition et son paramétrage (décimation).

Principe

L'algorithme SINITER s'appuie sur la méthode des moindres carrés non linéaires. Contrairement à sa version linéaire cette méthode n'a pas forcément une solution unique, il peut exister plusieurs minima locaux. C'est pour cette raison que l'initialisation du modèle est importante. Dans le cas où l'estimation est trop éloignée de la réalité et que le nombre d'échantillons par seconde est trop faible pour que l'ajustement récursif soit efficace on peut alors converger vers ces minimas locaux, ce qui nous donne des valeurs aberrantes.

Dans notre cas on cherche à adapter un modèle à un relevé d'échantillons. Voici la forme du modèle :

$$\mathbf{A} \times \sin(2 \times \pi \times \mathbf{F} \times t + \boldsymbol{\varphi}) + \mathbf{M}$$

où :
A est l'amplitude (1/2 Vpp)
F est la fréquence de notre signal en Hertz

5. pour SINus ITERation, basé sur les travaux du Pr F. Vernotte
6. Théorème d'échantillonnage de Shannon-Nyquist

t est notre vecteur de temps, à adapter en fonction de la période d'échantillonnage
φ est la phase
M est la valeur moyenne

On note P la matrice composée des paramètres à adapter :

$$P = \begin{Bmatrix} A \\ F \\ \varphi \\ M \end{Bmatrix} \tag{5.1}$$

Cette matrice doit être initialisée avec des valeurs estimées.

On définit ainsi une fonction R (pour résidus) qui est la différence entre notre modèle et les échantillons :

$$R = [A \times \sin(2 \times \pi \times F \times t + \varphi) + M] - Vr \tag{5.2}$$

Vr est notre vecteur d'échantillons [7]. R représente donc l'écart entre le modèle et la réalité, c'est une quantité différentielle dont la norme au carré est censée tendre vers zéro au fil des itérations de l'algorithme SINITER.
C'est cette fonction R que l'on va chercher à minimiser, ou plus exactement on minimise la norme au carré de R (méthode de Gauss-Newton).

Dans un premier temps on dérive R par rapport aux 4 paramètres qui nous intéressent (A, F, φ et M). On construit ainsi la matrice des dérivées premières partielles (Jacobienne) :

$$J = \begin{Bmatrix} \frac{\partial R}{\partial A} \\ \frac{\partial R}{\partial F} \\ \frac{\partial R}{\partial \varphi} \\ \frac{\partial R}{\partial M} \end{Bmatrix} = \begin{Bmatrix} \sin(2 \times \pi \times F \times t + \varphi) \\ A \times 2 \times \pi \times t \times \cos(2 \times \pi \times F \times t + \varphi) \\ A \times \cos(2 \times \pi \times F \times t + \varphi) \\ 1 \end{Bmatrix} \tag{5.3}$$

Cette matrice P représente la sensibilité des résidus aux variations des 4 paramètres du modèle. On obtient ainsi la relation suivante :

$$R = J.\Delta P \tag{5.4}$$

Où ΔP représente l'écart entre notre matrice des paramètres P et leurs valeurs censées minimiser R (résidus), autrement dit c'est la correction à appliquer à notre modèle. On en déduit :

$$\Delta P = J^{+}.R \tag{5.5}$$

La matrice J n'étant pas inversible J^{+} représente le pseudo-inverse de J. On applique ensuite la correction :

$$P = P - \Delta P \tag{5.6}$$

Finalement on a corrigé notre modèle en corrigeant les composantes de P, on en déduit une nouvelle fonction R, une nouvelle matrice J et une nouvelle correction ΔP. On répète le processus jusqu'à obtenir une valeur satisfaisante pour R. Dans notre cas cette valeur est jugée satisfaisante si sa norme quadratique est suffisamment proche de sa valeur précédente (différence inférieure à 10^{-8}) [8], on considérera alors que l'algorithme SINITER a convergé vers sa valeur finale.

7. En toute rigueur R et Vr sont des scalaires, fonction du temps t, lui aussi "échantillonné". On forme un vecteur en assemblant leur valeur pour chaque échantillon (notation type SciLab).

8. Cette valeur a été fixée empiriquement afin d'obtenir un bon compromis rapidité/précision

Traitement statistique

Afin d'exclure les résultats aberrants du SINITER on peut y appliquer un filtrage statistique.

On fait tout d'abord l'hypothèse que les résultats du SINITER devraient suivre une loi gaussienne autour d'une valeur moyenne. On calcule ensuite la moyenne et l'écart-type de nos mesures brutes et on exclut toutes les données non-comprises entre la moyenne +/- 5 écarts-types (on peut également faire un filtrage plus strict à 4 écarts-types par exemple). On obtient ainsi un nouveau jeu de valeurs sur lesquelles on peut répéter le procédé jusqu'à ce que toutes les valeurs soit comprises dans l'intervalle voulu.

Si une variable suit une loi de distribution Gaussienne alors la probabilité qu'une réalisation de cette variable se trouve hors de l'intervalle à $\pm$ 5 écart-types autour de la valeur moyenne est de 1 valeur sur 1 744 000. Pour un intervalle de $\pm$ 4 écart-types la probabilité est de 1 valeur sur 15 787 et pour $\pm$ 3 écart-types 1 valeur sur 370 est hors de l'intervalle.

La figure 5.4 présente un exemple de simulation des résultats de SINITER sur un jeu de 200 mesures, pour un signal à 30kHz. On peut voir sur le jeu initial de données (en bas à gauche) que les valeurs obtenus sont centrées autour de 30kHz, les "pics" représentent les valeurs aberrantes fournies par l'algorithme.
On applique un filtrage à 5σ. On aurait également pu faire un filtrage à 3σ (suppression d'une valeur sur 370) ce qui réduirait le nombre d'itérations nécessaire.

Le premier passage (au milieu à gauche) supprime 3 valeurs. Les itérations successives sont référencées par leur numéro de passage. On peut voir que la dernière itération (8è passage, en haut à droite) n'a aucun effet, les données restantes sont donc supposées fiables.

Ainsi 18 valeurs ont été supprimées sur les 200 initiales, ce qui est largement supérieur à la valeur théorique d'une valeur sur 1 774 000. Ceci peut être dû à plusieurs facteurs :
- L'initialisation des paramètres (matrice P, en particulier la composante de phase) trop éloignée des valeurs réelles.
- La convergence vers des minima locaux, faisant passer la valeur de la norme quadratique sous la valeur de seuil fixé.
- Un relevé d'échantillons trop bruité

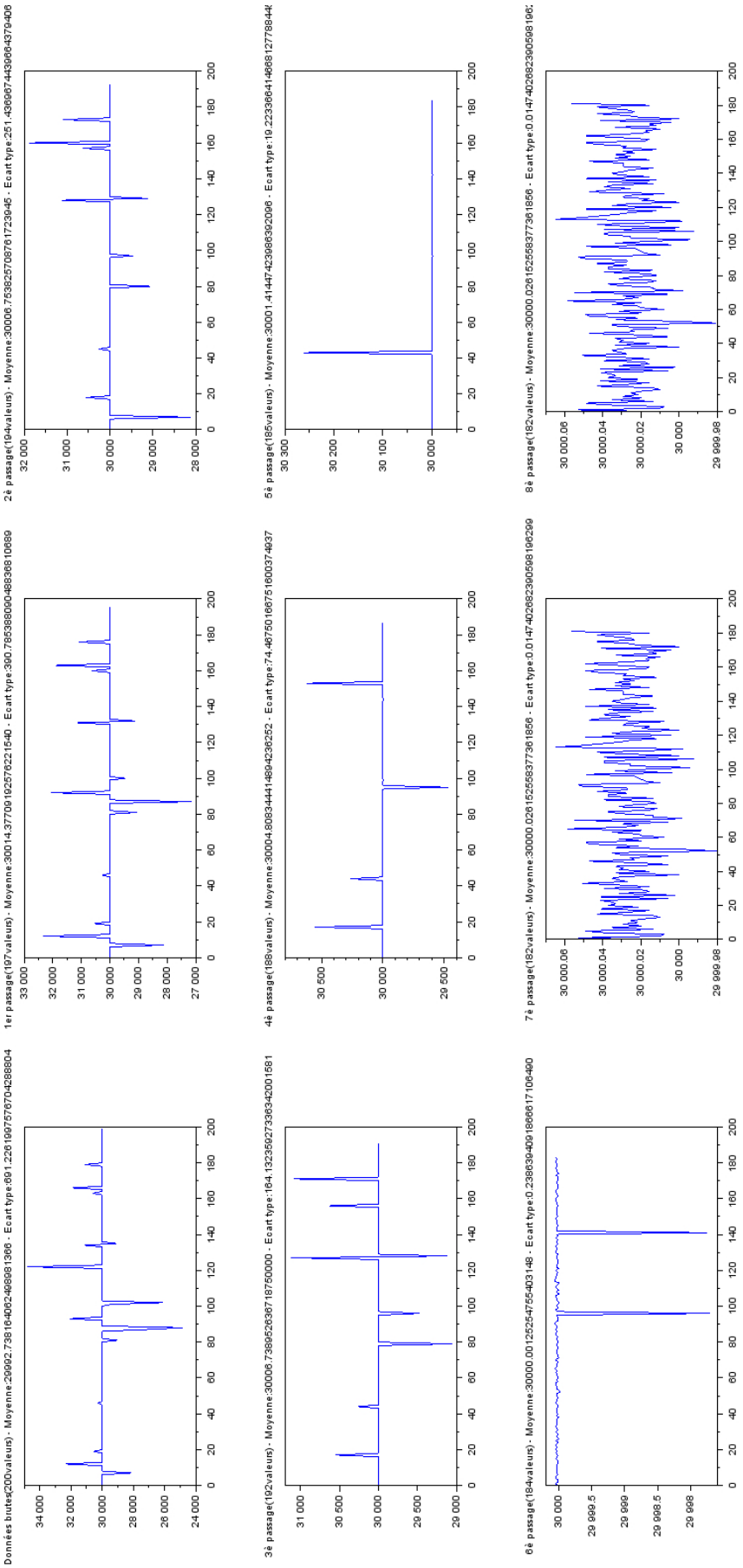


FIGURE 5.4 – Exemple de filtrage statistique à 5σ

Relevés de précision SINITER

Afin de tester les performances de l'algorithme SINITER implanté sur la RedPitaya j'ai réalisé l'acquisition simultanée de 2 signaux issus d'une même source (GBF), ainsi en faisant varier la fréquence du GBF puis en modifiant la période d'échantillonnage du RedPitaya j'ai observé le comportement du SINITER.

La figure 5.5 présente le montage effectué.

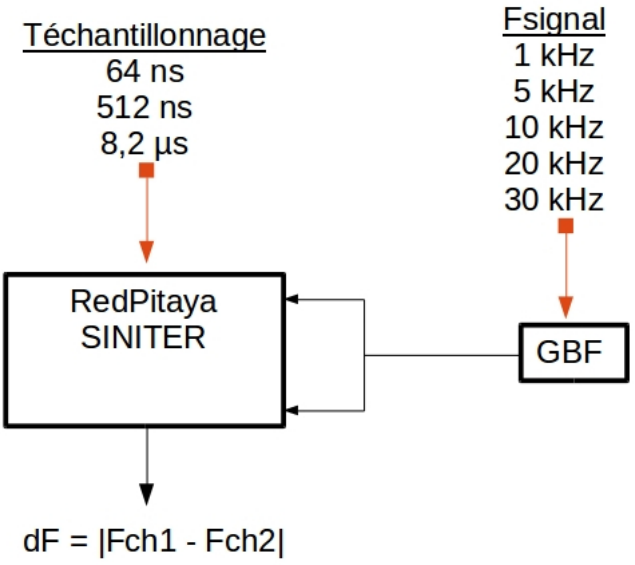


FIGURE 5.5 – Banc de mesure des performances SINITER

L'acquisition simultanée sur 2 canaux identiques du RedPitaya et la considération de la quantité $dF = |F_{CH1} - F_{CH2}|$ permet d'éliminer le bruit de fréquence dû au GBF. Cette technique de mesure s'appelle la corrélation croisée (ou cross-correlation). En effet si l'on considère que les fréquences mesurées subissent les bruits additifs intrinsèques du GBF, ceux des canaux d'acquisition (ADC) et ceux dû à l'algorithme SINITER, alors la fréquence mesurée peut s'écrire

$$F_{CH1/2} = \nu_0 + \Delta\nu_{GBF} + \Delta\nu_{ADC1/2} + \Delta\nu_{SINITER1/2}$$

ainsi la différence s'exprime

$$F_{CH1} - F_{CH2} = (\Delta\nu_{ADC1} - \Delta\nu_{ADC2}) + (\Delta\nu_{SINITER1} - \Delta\nu_{SINITER2})$$

Tous les relevés comportent 8000 échantillons.

Pour chaque jeu de paramètres (F_{GBF},T_{ECH}) j'exécute 200 mesures (soit 200 relevés de 8000 échantillons chacuns) avec SINITER et j'analyse les valeurs obtenues (moyenne, écart-type). Mon estimateur de performance $\frac{dF}{F_{\mathrm{GBF}}}$ représente la variation de fréquence relative. Un autre indicateur de précision important est l'écart-type, qui représente la dispersion des résultats autour de la moyenne.

Un filtrage statistique à 4 écarts-types à été réalisé afin d'exclure les résultats aberrants, ce filtrage est le même pour chaque jeu de paramètres.

La fréquence d'échantillonnage maximale du RedPitaya est de 125MS/s (soit une période d'échantillonnage de 8ns) j'ai appliqué une décimation de 8, 64 et 1024 afin d'obtenir des périodes d'échantillonnage de 64ns, 512ns et 8,2μs respectivement. Il est impossible d'obtenir des valeurs de décimation intermédiaires sans reprogrammer le FPGA.

Dans un premier temps j'ai fixé la période d'échantillonnage à 512 ns et j'ai fait varier la fréquence du GBF. Ensuite j'ai fixé la fréquence du GBF à 30 kHz et j'ai fait varier la période d'échantillonnage (par décimation). Tous les relevés correspondant sont disponibles en annexe.

nb de périodes	pts/période	dF/F
4	1953	2,49E-05
16	516	4,47E-06
21	390	3,18E-06
41	195	1,60E-06
82	98	7,00E-07
123	65	5,13E-07
125	64	1,15E-06
200	4	4,40E-03

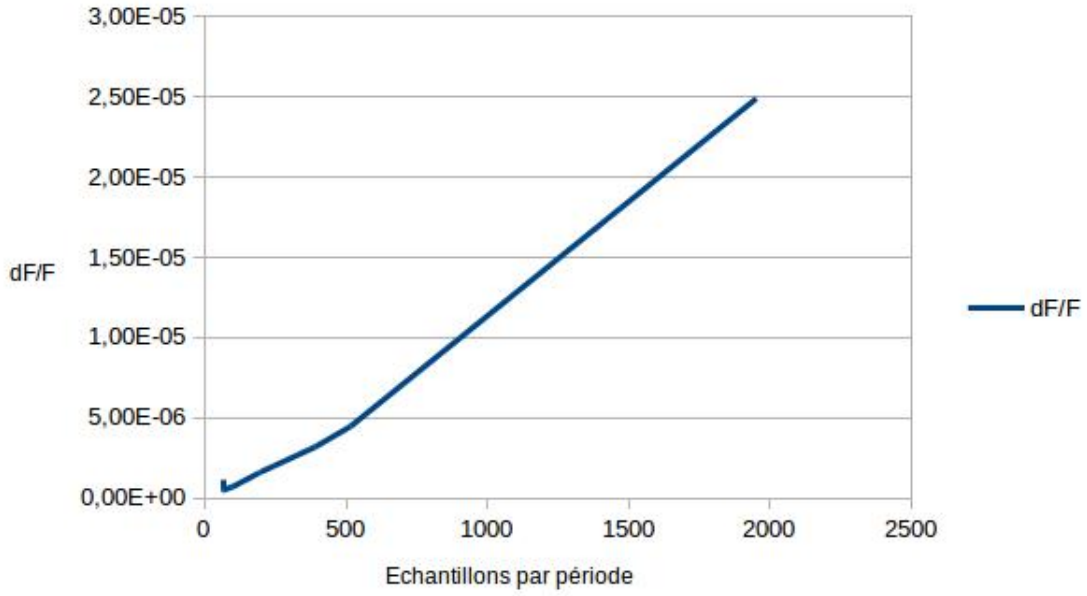


FIGURE 5.6 – Performances SINITER (T_{ECH} fixé 512ns, F_{GBF} variable)

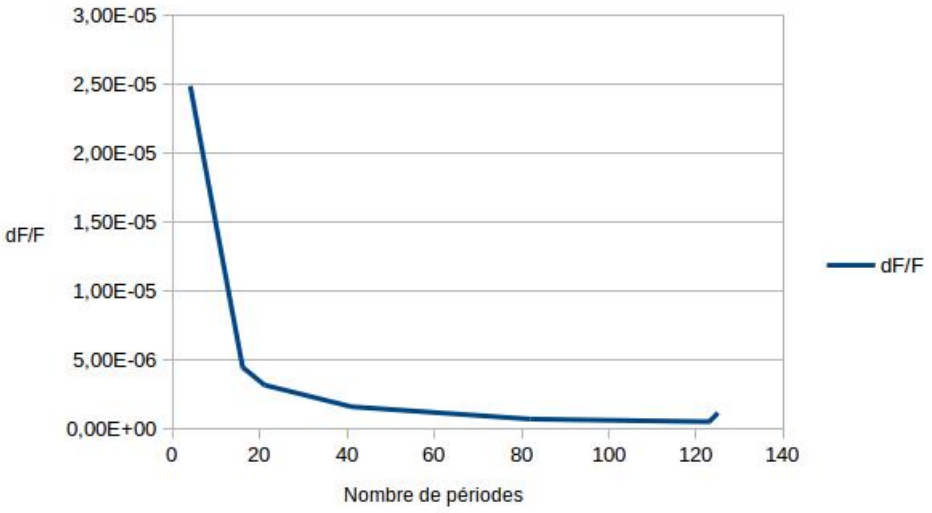


FIGURE 5.7 – Performances SINITER (F_{GBF} fixée à 30kHz,T_{ECH} variable)

J'ai recensé les données importantes dans un tableau récapitulatif :

Dec = 64

Fréq GBF	T GBF	pts/période	nb de périodes	dF moyenne	dF écart-type	dF/F
1000	1ms	1953	4	2,49E-002	3,20E-002	2,49E-05
5000	200us	390	21	1,59E-002	2,04E-002	3,18E-06
10000	100us	195	41	1,60E-002	2,06E-002	1,60E-06
20000	50us	98	82	1,40E-002	1,98E-002	7,00E-07
30000	33us	65	123	1,54E-002	1,88E-002	5,13E-07

F=30kHz

Décimation	Téchantillonnage	pts/période	nb de périodes	dF moyenne	dF écart-type	dF/F
8	64ns	516	16	1,34E-001	1,70E-001	4,46667E-006
64	512ns	64	125	3,46E-002	1,12E-001	1,15333E-006
1024	8,2us	4	200	1,32E+002	1,89E+002	0,0044

FIGURE 5.8 – Précision du SINITER

J'ai volontairement omis la dernière ligne du tableau (4 points par période) dans les 2 derniers graphes car elle rendait les données illisibles (valeur dF/F trop importante).

Ces 2 graphes sont "croisés" car le nombre total d'échantillons est fixé à 8000, ainsi lorsque le nombre d'échantillons par période augmente, le nombre de périodes acquises diminue.

On peut voir que la précision du résultat du SINITER augmente (dF/F diminue) lorsqu'on augmente le nombre de périodes jusqu'à une certaine limite où le nombre de points par périodes devient trop faible.

Pour le nombre de points par période on peut voir que la courbe est quasiment une droite ce qui est lié au fait que pour un nombre d'échantillons total fixé, le nombre d'échantillons par période est inversement proportionnel au nombre de périodes acquises. Cependant il existe un point d'inflexion pour des valeurs faibles d'échantillons/période (inférieur à la dizaine).

On peut donc conclure qu'il existe une valeur "suffisante" de points/période de l'ordre de 10.

Concernant le nombre de périodes, plus il y en a et plus SINITER donnera un résultat précis. Dans une perspective d'optimisation du temps de calcul, l'acquisition d'une centaine de périodes semble un bon compromis.

5.3 ZedBoard

La carte ZedBoard dispose d'une connectique mieux fournie que la RedPitaya bien qu'elle n'intègre pas directement de CAN. La puce Zynq présente (7Z020) est de la gamme légèrement supérieure à celle de la RedPitaya du point de vue logique programmable mais la partie processeur est identique.

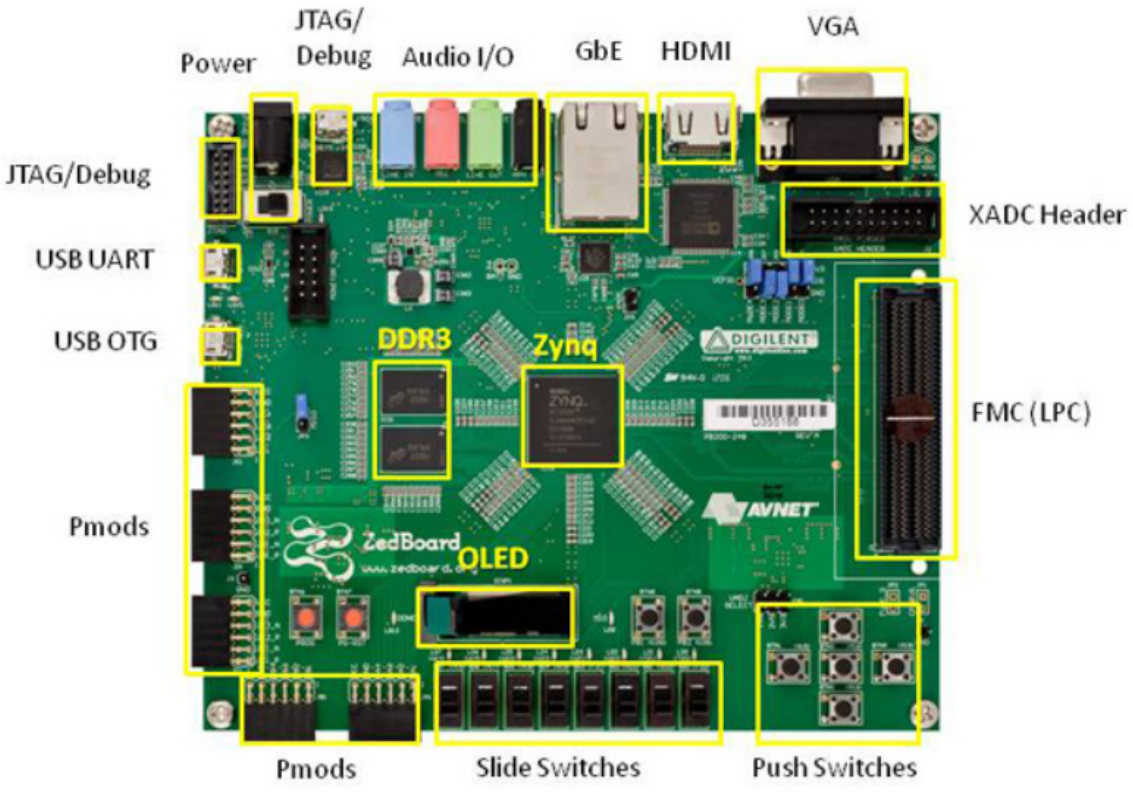


FIGURE 5.9 – Carte de développement ZedBoard

5.3.1 Acquisition directe

En réfléchissant à une alternative à l'échantillonnage rapide et à la reconstruction des sinusoïdes il m'est venu l'idée de réutiliser les ZCD présents dans l'ancienne horloge composite. Leur rôle était de convertir les battements lents (700 Hz) issus du mixeur en signaux logiques (carré $0 - 3.3V$) manipulables dans le domaine FPGA.

Ces ZCD sont prévus pour fonctionner à 700 Hz mais heureusement ils disposent d'une bande passante suffisamment large pour transformer des signaux à 10MHz. J'ai donc pu les utiliser dans mon application.

Ici l'idée est de transformer directement les signaux d'entrée en signaux logiques. Ensuite le code FPGA peut directement effectuer des mesures d'intervalles de temps entre les différentes sources en datant l'occurrence de leur fronts montants grâce à un compteur rapide [9]. Cette méthode

9. La description en détail du code FPGA est donnée dans le chapitre suivant

permet d'acquérir indifféremment des signaux numériques ou analogiques de fréquences allant de 1 Hz (PPS) à 800 MHz (fréquence maximale théorique admissible par le FPGA de la ZedBoard).

Dans un premier temps j'ai utilisé les connecteurs Pmods de la ZedBoard pour réaliser mon acquisition, ces connecteurs ne sont pas dédiés aux signaux d'horloge [10] mais ils sont assez aisés à utiliser (une simple plaque avec des connecteur SMA soudés). Il s'est avéré que cette méthode est suffisamment efficace pour mesurer les écarts temporels mais l'occurrence plus ou moins régulière de valeurs aberrantes nous a amenés à chercher une alternative.

Par la suite il parut donc pertinent de concevoir et de réaliser une carte spécifique permettant d'utiliser le connecteur FMC qui permet une acquisition de signaux basse tension différentiels (LVDS) sur des pins dédiés au signaux d'horloge et donc d'obtenir des mesures de qualité métrologique.

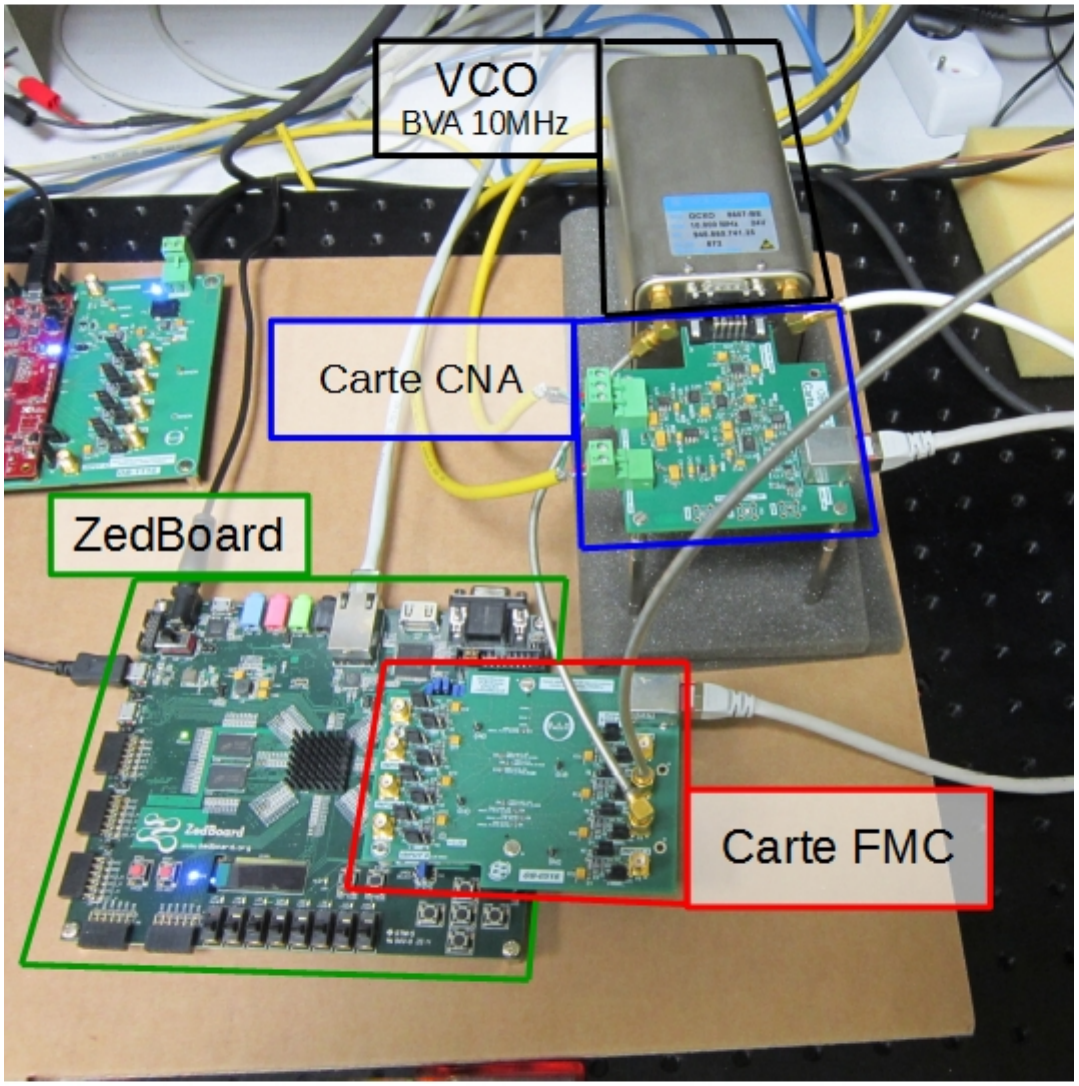


FIGURE 5.10 – Montage ZedBoard/BVA10

10. Ils ne sont pas désignés "Clock Compatibles" dans les spécifications du constructeur

5.3.2 Cartes d'interfaces

Interface d'entrée LVDS - connecteur FMC

Afin de faire le lien entre la carte de développement (Zynq) et le monde extérieur (sources de fréquence), une carte d'extension a été développée sur mesure pour notre application.

Les composants principaux utilisés sur cette carte sont les LTC5967-2 qui permettent une conversion des signaux analogiques sinusoïdaux directement en carrés logiques différentiels (LVDS). Des connecteurs SMA permettent de récupérer les signaux des sources. La carte est reliée à la ZedBoard par le biais du connecteur FMC (FPGA Mezzanine Card). La puce Zynq possède 8 paires d'entrées différentielles dédiées aux signaux d'horloge (Clock compatible) qui sont toutes utilisées pour l'acquisition des sources.
C'est également par cette carte que la liaison SPI va transiter via un connecteur RJ45, vers la carte d'interface de sortie décrite dans la section suivante.

Interface de sortie SPI - convertisseur numérique/analogique

Afin de transmettre la tension de commande à appliquer au VCO on utilise une liaison SPI (Serial Peripheral Interface) unidirectionnelle sur 3 fils (data, clock et chip select). Une carte embarquant un CNA [11] permet de faire le lien entre commande numérique (20 bits) et niveau de tension correspondant. Le composant principal utilisé est le DAC AD5791 d'Analog Devices avec des amplificateurs opérationnels AD8675 et AD86756 du même fabricant.

Cette carte de conversion est celle utilisée dans l'ancienne horloge composite. Toutefois il a fallu développer un driver spécifique afin de gérer cette liaison depuis le Linux embarqué dans la ZedBoard.

5.3.3 Pilotage du VCO

Ici le VCO utilisé est un BVA 10 MHz ultra stable possédant une stabilité de fréquence relative de quelques 10^{-13} entre 1 et 100 secondes.

Caractérisation du CNA/VCO

Afin d'établir la correspondance entre la tension de commande et la fréquence de sortie du VCO il est indispensable de réaliser des tests de calibration de sensibilité. La correspondance entre le mot de 20 bits et le niveau de tension en entrée du VCO dépend de la configuration du DAC, dans notre cas il fournit une tension linéaire entre 0 et 10V pour une commande variant de 0 à 2^{20}-1 = 1.048.575.

11. Convertisseur Numérique/Analogique

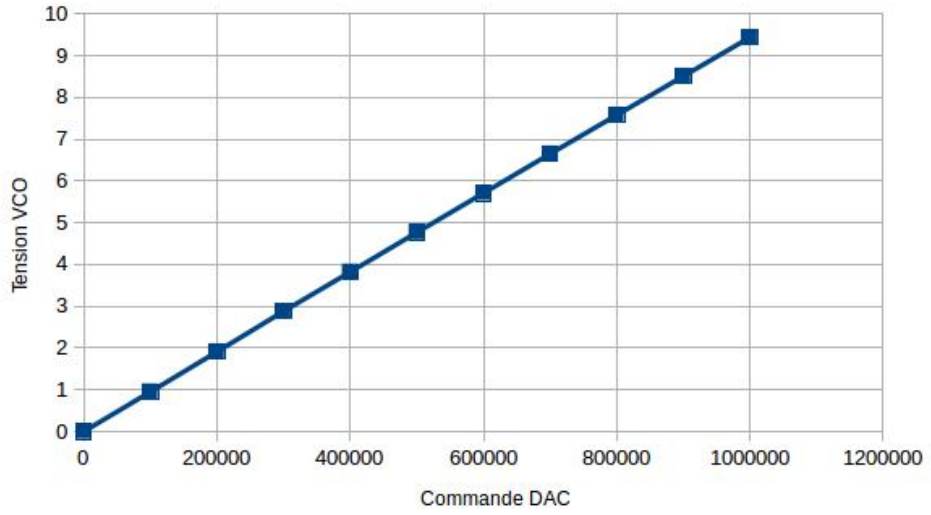


FIGURE 5.11 – Tension de sortie du CNA en fonction de sa commande SPI 20 bits

Pour améliorer encore la résolution de la commande de tension, un pont diviseur peut être rajouté entre le CNA et le VCO. C'est le cas sur la carte d'interface utilisée, ce qui réduit la plage de commande en tension du VCO à 2 Volts pour une même plage de sortie du CNA.

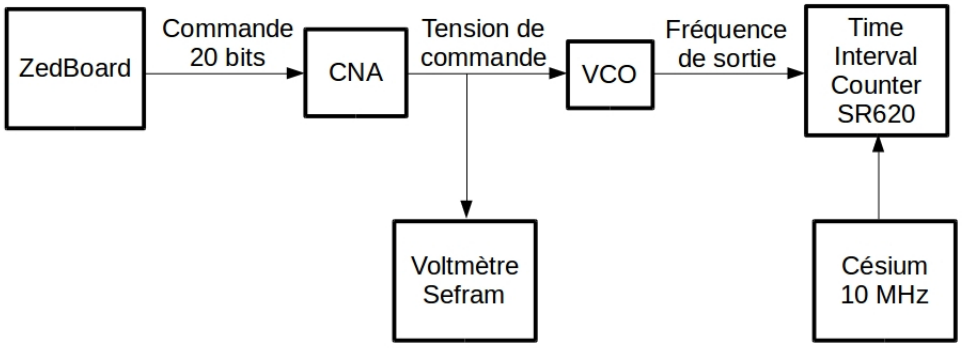


FIGURE 5.12 – Dispositif de caractérisation du CNA/VCO

La fréquence de sortie du VCO est directement comparée à la fréquence du césium. La durée d'intégration est de 100 secondes et plusieurs mesures sont moyennées pour chaque niveau de tension. Les figures 5.13 et 5.14 présentent la fréquence de sortie du VCO respectivement en fonction de la tension de commande et du mot binaire sur 20 bits.

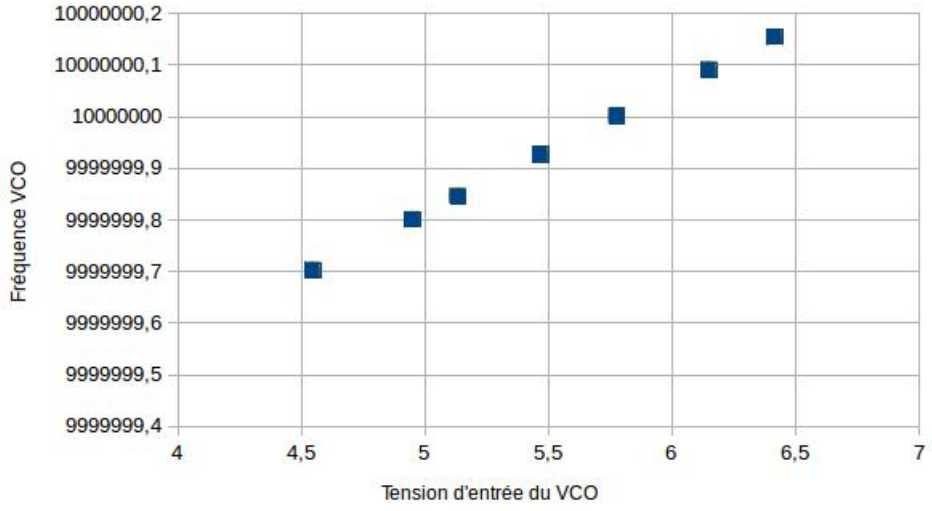


FIGURE 5.13 – Fréquence du BVA10 en fonction de sa tension de commande

La pente de cette courbe représente la sensibilité du VCO. Une régression linéaire donne

$$S_{BVA} = 236.10^{-3} Hz/V$$

En parallèle la valeur du mot de 20 bits à été relevée pour chaque valeur de fréquence.

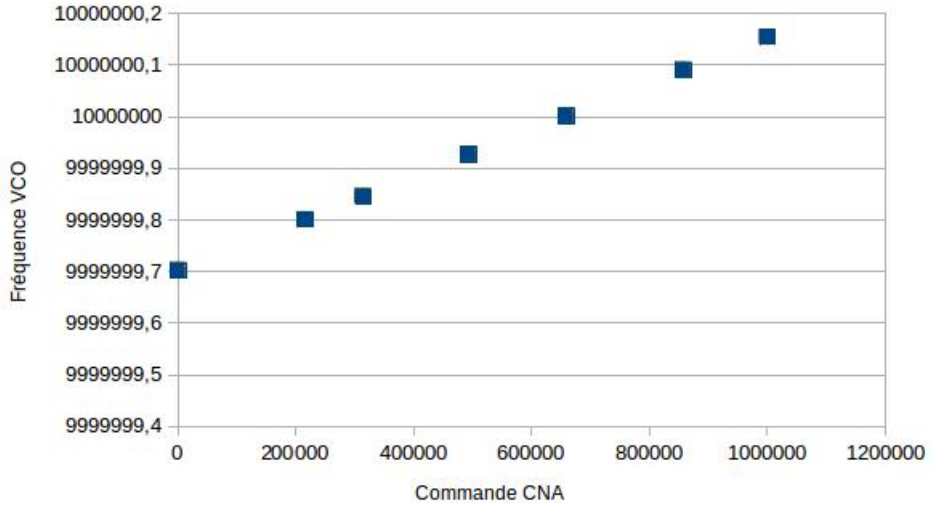


FIGURE 5.14 – Fréquence du BVA10 en fonction de la commande 20 bits

La sensibilité du VCO en fonction du mot 20 bits est

$$S'_{BVA} = 450.10^{-9} Hz/pas$$

L'incrémentation de 1 pas de la commande entraîne donc une variation de fréquence du VCO de 450 nano Hertz. Ceci représente une granularité de commande théorique en terme de variation relative de $4{,}5.10^{-14}$, ce qui représente le palier de Flicker théorique de notre système.

Dans l'application c'est l'inverse de cette valeur qui nous intéresse afin de convertir la correction de fréquence issue de l'algorithme en mot de 20 bits à ajouter (ou soustraire) à la commande précédente. Soit

$$K_{BVA} = \frac{1}{S'_{BVA}} = 2,222.10^6 pas/Hz$$

5.4 MicroZed

La MicroZed est une carte de développement basée sur les puces Zynq 7Z010 (identique à la RedPitaya) que l'on peut combiner à des modules (carte-mère) personnalisé grâce à deux connecteur 100 broches (version "low-cost" d'un connecteur FMC).

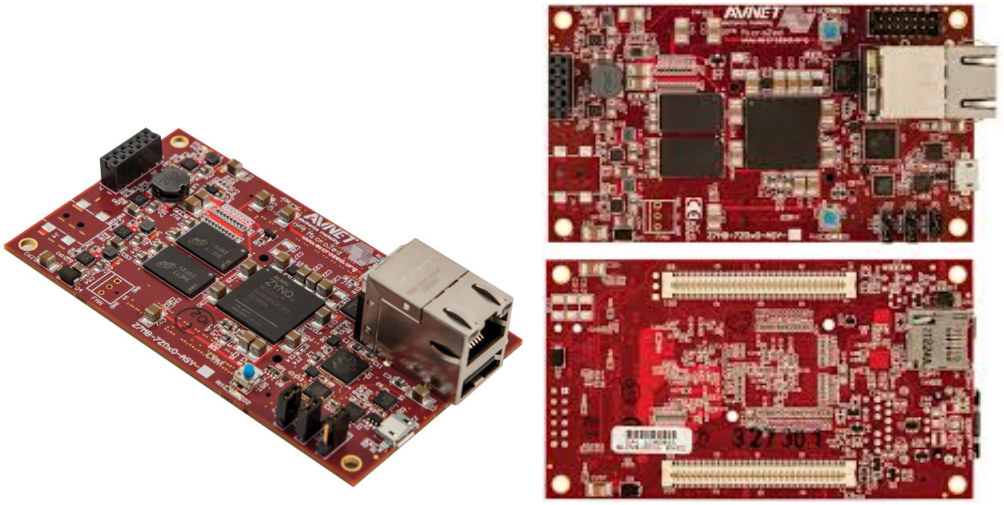


FIGURE 5.15 – Carte MicroZed

L'objectif en utilisant cette carte est d'intégrer tous les éléments utilisés sur une seule et même carte-mère et d'utiliser un VCO meilleur marché que le BVA utilisé avec la ZedBoard. Ceci pour pouvoir réaliser 2 prototypes identiques à moindre coût sur lesquels seront implémentés des algorithmes différents afin de comparer leurs performances.

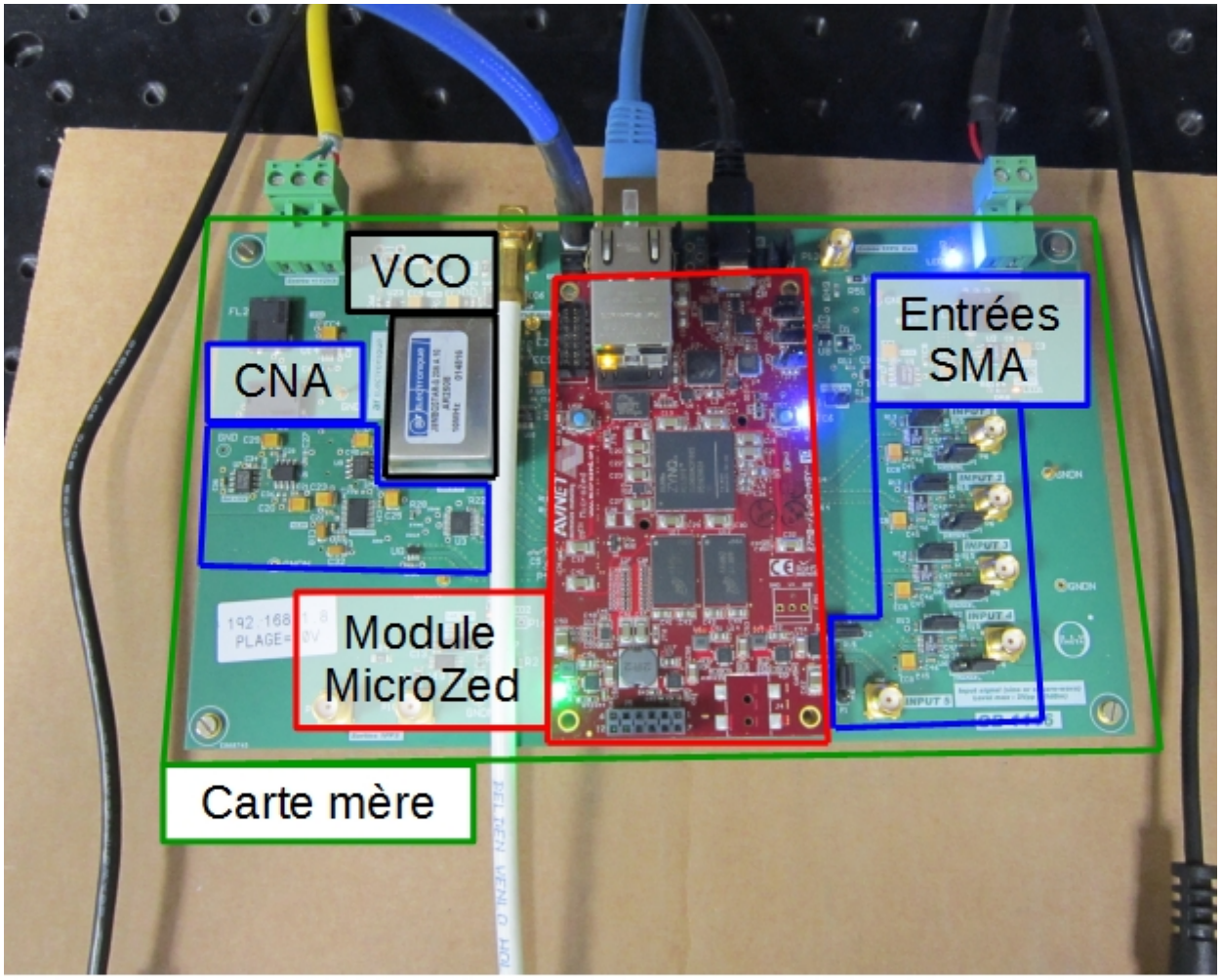


FIGURE 5.16 – Montage MicroZed/carte mère

5.4.1 VCO

Le BVA utilisé avec la ZedBoard avait coûté environ 10.000 euros et n'est plus commercialisé actuellement, ce qui en fait l'élément le plus coûteux loin devant la ZedBoard (450 euros), la MicroZed (180 euros) les circuits imprimés (200 euros pour du 4 couches) et les composants électroniques (200 euros).

Le produit basé sur la carte MicroZed sera donc plus compact et moins onéreux au prix d'un sacrifice des performances du VCO utilisé [12].

Le VCO utilisé est un OCXO 10MHz AR2508 d'AR électronique d'un coût de 100 euros. Une caractérisation à aussi été réalisée pour ce VCO :

12. voir figure 3.8 page 53 courbe du VCOAR 10MHz

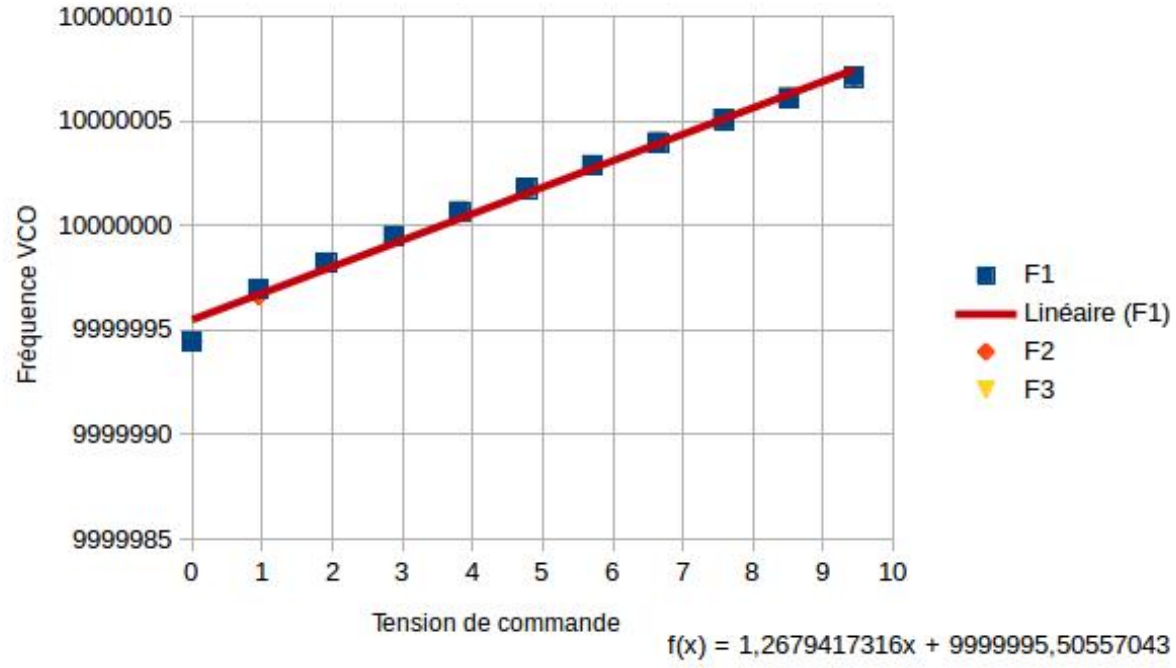


FIGURE 5.17 – Caractérisation du VCO AR en fonction de sa tension d'entrée

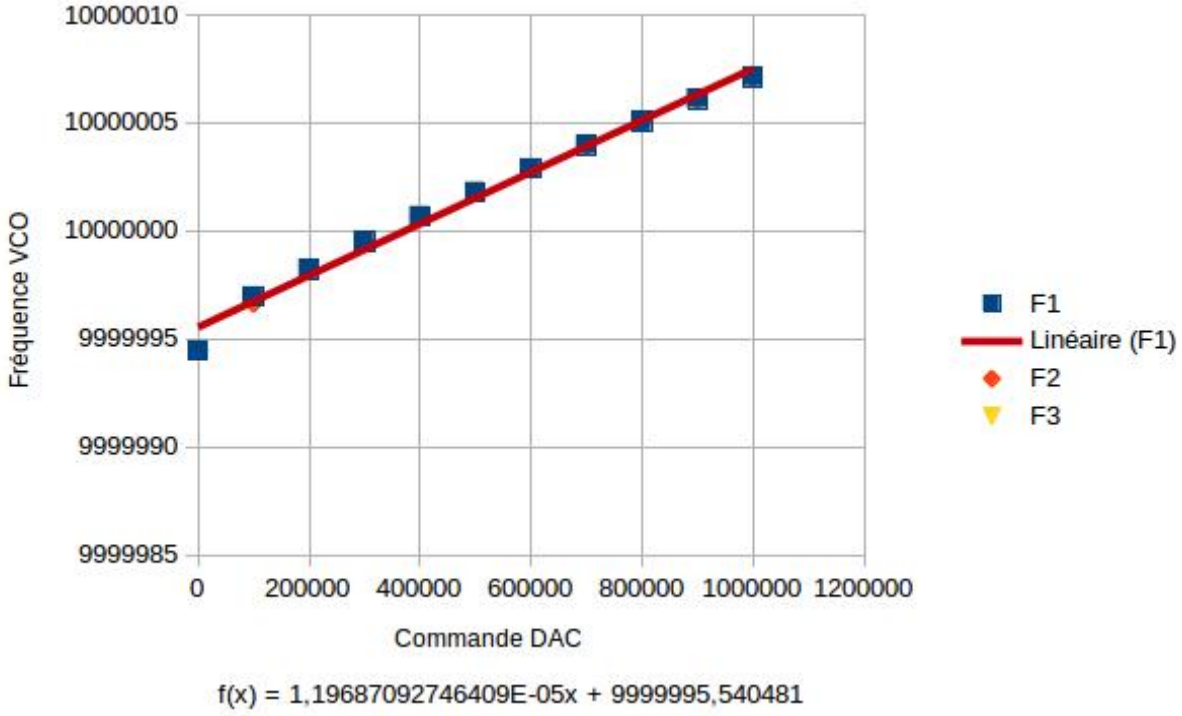


FIGURE 5.18 – Caractérisation du VCO AR en fonction du mot 20 bits

Le sensibilité de ce VCO est

$$S_{VCO_AR} = 1,26Hz/V$$

soit

$$S'_{VCO_AR} = 1,2.10^{-5}Hz/pas$$

et le coefficient inverse

$$K_{VCO_AR} = \frac{1}{S'_{VCO_AR}} = 8,33.10^{4}pas/Hz$$

Le coefficient S'_{VCO_AR} donne un palier de Flicker théorique de $1,2.10^{-12}$ en terme de fréquence relative.

Ici se termine la description du hardware "dur" servant d'interface entre le monde analogique (sorties des sources de fréquences et tension de contrôle du VCO) et le monde numérique (code FPGA et algorithme d'asservissement) [13].

13. voir figure 4.1 page 56

Chapitre 6

Code FPGA

Ce chapitre se situe dans la partie "hardware" mais il est également à cheval sur le domaine software dans le sens ou le code FPGA est créé par un logiciel mais implémente une fonction réellement hardware.

Le logiciel utilisé pour créer et compiler ce code est VIVADO 2015.4 qui est un logiciel propriétaire de Xilinx.

6.1 Versions intermédiaires

J'ai testé plusieurs méthodes de mesure des écarts de fréquence et de phase au niveau du FPGA. Le processus de test d'un code FPGA peut être assez long : une fois le diagramme mis au point sur l'outil graphique VIVADO il faut compiler (ce qui peut prendre jusqu'à 10mn) et créer un fichier binaire (bitstream) à transférer au Zynq pour flasher son FPGA. Ensuite il faut "faire tourner" la manip pendant un temps significatif afin d'observer d'éventuels dysfonctionnements.
Le développement d'un code FPGA fiable et performant constitue la base du développement du futur algorithme, ceci a constitué un pan important de mon travail de thèse et m'a pris énormément de temps. Cette section et la suivante décrivent les différentes phases intermédiaires avec leur lot de problèmes.

6.1.1 Quelle fonction implémenter ?

Le point de départ du code FPGA est l'ensemble des signaux numériques [1] issus des sources de fréquence. La fonction développée devra fournir une information de phase et/ou de fréquence à partir de ces signaux complètement asynchrones.

La cadence des mesures doit être suffisamment élevée pour permettre de piloter le VCO avec une constante de temps adéquate. Cette cadence dépend donc du VCO utilisé. Au vue des performances des différents VCO (dont la quasi-totalité sont des quartz) existants la cadence des mesures se situe entre 1ms et 1s.

1. LVDS 10MHz

6.1.2 Sources, référence et compteur

Toutes les méthodes d'estimation de la phase et de la fréquence font appel à une référence pour effectuer leurs mesures. Dans notre cas cette référence serait forcément [2] le VCO que l'on cherche à contrôler.
On pourrait également considérer une référence externe, dans lequel cas notre système devient un outil de mesure de phase/fréquence et non plus un système d'asservissement.
Dans tout les cas, une seule source à la fois peut être considérée comme référence des mesures.

La référence constitue le "cadre" de la mesure, elle donne une échelle de valeur sur laquelle viennent se positionner les sources.

Les possibilités explorées sont les suivantes :
— Fréquencemètre direct.
  La fréquence de la référence est divisée pour mesurer le nombre d'impulsions de la source. Typiquement la référence fournit un signal PPS et le nombre de fronts de la source dans cet intervalle correspond directement à la fréquence mesurée.
— Fréquencemètre réciproque.
  La fréquence de la référence est multipliée pour mesurer une ou plusieurs périodes du signal de la source. Par exemple si la référence est multipliée jusqu'à 800MHz et que la source est à 10 MHz on pourra mesurer sa période avec une résolution de 1,25ns.
— Intervallomètre
  Le signal de référence est dupliqué. D'un côté il est multiplié pour incrémenter un compteur servant d'échelle de datation des sources. De l'autre côté il est divisé pour donner le "top départ" du compteur.

Toutes ces méthodes font intervenir un procédé de comptage, aussi bien pour les divisions de fréquence que pour les mesures elles-mêmes. Le nombre de bits, la profondeur (valeur maximale), la remise à zéro et l'inhibition de l'horloge doivent être définis selon l'application du compteur. Par exemple il est pertinent de laisser un compteur rapide ($> 100MHz$) en roue libre afin d'éviter les temps morts impliqués par sa remise à zéro [3], alors qu'un diviseur devra être réinitialisé à partir d'un seuil fixé.

6.1.3 Premières versions du code

Les FPGA présents sur les puces zynq possèdent des PLL [4] et MMCM [5] intégrés permettant de générer un signal de sortie à une fréquence multiple de leur fréquence d'entrée. Les restrictions de fréquence d'entrée des blocs PLL ne permettant pas de les utiliser à 10MHz, le choix s'est donc porté sur les blocs MMCM.

Fréquencemètre direct

C'est la première méthode testée lors de l'acquisition de la ZedBoard.
Elle consiste à mesurer le nombre de fronts (montants ou descendants) de la source pendant un intervalle de temps donné par la référence, son architecture simple fait appel à :

2. Aucune connaissance a priori sur la stabilité des sources
3. Sachant que la valeur du compteur peut mettre plusieurs coups d'horloge pour être disponible en sortie du FPGA
4. Phase Locked Loop
5. Mixed Mode Clock Manager

— Une division de fréquence, pour transformer le signal issu de la référence en un signal lent
— Un compteur incrémenté par la source, d'une profondeur suffisante pour éviter un retour à zéro entre 2 fronts montants du signal lent.

Fréquencemètre réciproque

Ici on a besoin d'effectuer :
— Une multiplication de fréquence, pour transformer le signal de la référence en un signal à 800MHz.
— Un compteur incrémenté par ce signal à 800 MHz.

Intervallomètre

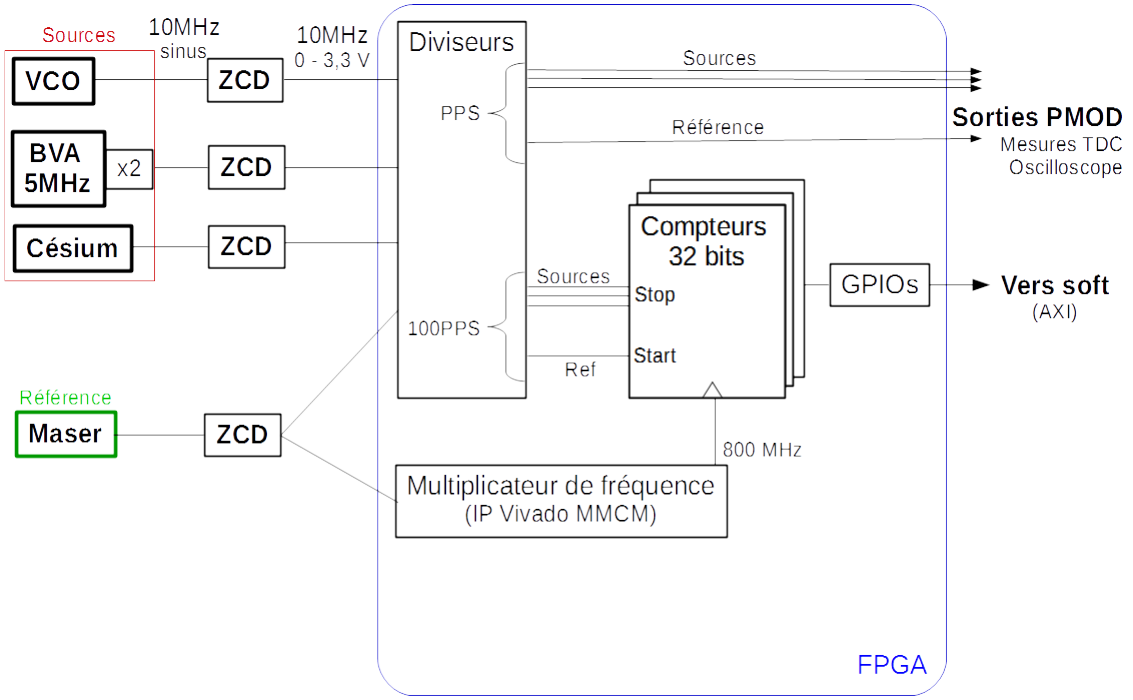


FIGURE 6.1 – Schéma FPGA de l'intervallomètre initial

Parmi ces 3 architectures FPGA possibles c'est cette dernière qui a été retenue pour la suite du projet. La principale raison étant que la mesure d'une fréquence implique une durée d'intégration importante afin d'obtenir une précision satisfaisante, contrairement à la mesure d'un temps qui, dans le cadre de l'utilisation d'un FPGA récents, peut être bien plus précise et rapide.

6.2 Améliorations de l'intervallomètre

Dans sa version initiale, l'intervallomètre fournissait des mesures aberrantes ponctuelles et apparemment erratiques. La figure 6.2 montre l'exemple d'une série de mesures comportant plusieurs valeur aberrantes appelées glitchs (défaillance en français).

L'ensemble des améliorations apportées à l'intervallomètre sont le fruit d'échanges avec différentes personnes de l'observatoire et de l'entreprise. Les améliorations successives sont décrites

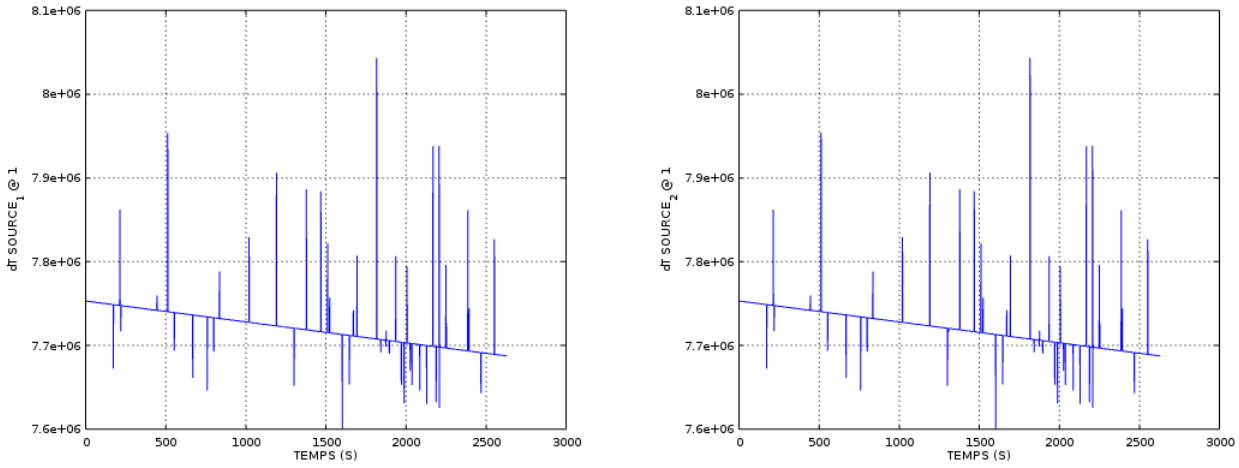


FIGURE 6.2 – Exemple de glitchs sur les mesures d'écarts temporels

succinctement dans les paragraphes suivants et seront développées plus en détails dans la section "Version finale".

6.2.1 Domaine d'horloge et synchronisations

Le domaine d'horloge d'un FPGA correspond à sa fréquence de fonctionnement sur ses opérations élémentaires [6]. Des placements et routages internes au FPGA sont prévus pour ces signaux ou leur délais de propagation sont connus (ou bien estimés).

Dans notre cas le domaine d'horloge principal est le signal d'incrémentation du compteur, qui est directement lié à notre signal de référence. Cependant chaque source en entrée étant indépendante elles engendre chacune leur domaine d'horloge. D'où la nécessité de les synchroniser au domaine d'horloge principal.

6.2.2 Fréquence maximale du domaine d'horloge

Théoriquement le fréquence maximale de sortie des blocs MMCM est de 800 MHz[10], mais tout dépend de l'utilisation qui est faite de ce signal. Dans notre cas l'utilisation d'éléments séquentiels comme les bascules de synchronisation ou les registres de données implique le respect de temps de setup et de hold impossible à garantir à 800 MHz. D'après [10] la fréquence maximale est de 464 MHZ. Les facteurs multiplicatifs disponibles dans le bloc MMCM sont des puissances de 2 (2, 4, 8, 16, 32 et 64). Ainsi en utilisant un signal à 10MHz en entrée le facteur multiplicatif le plus adapté est 32, fournissant donc un signal à 320 MHz pour incrémenter le compteur.

6.2.3 Interruptions

Pour déclencher la lecture des GPIO dans l'application software il est nécessaire que la logique programmable envoie un signal de confirmation de l'occurrence et de la disponibilité d'une nouvelle mesure.

6. Du type incrémentation de compteur ou logique séquentielle

Dans les versions primaires du code FPGA ces interruptions étaient gérées par la lecture bloquante depuis la partie logicielle d'un GPIO dédié [7]. Cette méthode de gestion d'interruption est simple mais peu rigoureuse car elle fait intervenir un élément intermédiaire (GPIO) pouvant introduire un délai avant l'acquisition des données.

La pratique correcte est d'envoyer directement le signal d'interruption à la partie processeur qui possède un gestionnaire dédié aux interruptions provenant de la logique programmable (jusqu'à 16 interruptions). Toutefois cela demande de développer un pilote spécifique à insérer au noyau Linux (code fournit en annexe).

Cette interruption va servir à cadencer l'application logicielle, elle doit donc être générée par le signal de référence.

6.2.4 Registres temporaires et finaux

Chacune des sources est supposée indépendante et va donc déclencher le chargement des données dans des registres temporaires de façon également indépendante. Afin d'assurer la disponibilité (et la stabilité) de ces valeurs lors de leur lecture par les GPIO, le signal de référence (générant aussi l'interruption) va transférer les données des registres temporaires dans des registres finaux.

Ainsi dans le cas ou le front montant d'une source arriverait immédiatement (quelques nanosecondes) après le front de la référence seuls les registres temporaires seront modifiés sans impacter immédiatement les registres finaux, assurant ainsi la disponibilité des données à chaque instant.

6.2.5 Sécurisation software

Malgré l'ensemble des améliorations apportées au code FPGA des erreurs de mesure étaient toujours présentes.
Il s'est avéré que ces problèmes provenaient de l'application C (l'algorithme d'asservissement). La lecture des GPIO étant concurrente avec le reste de l'algorithme celle-ci doit être faite de façon sécurisée, quitte à devoir bloquer temporairement la mesure.

7. Une boucle infinie dans laquelle on vérifie que le signal lu passe de "0" à "1"

6.3 Version finale

Le but du code implanté dans le FPGA est de dater les occurrences des fronts montants des sources et de la référence afin de les transmettre à l'algorithme d'asservissement qui en déduira les écarts temporels. La périodicité de mesure de ces écarts à été fixée à 10 ms. La qualité de ce code est primordiale afin de pouvoir espérer obtenir un asservissement de qualité.

Pour réaliser ces datations j'ai utilisé un compteur binaire incrémenté par le signal issu de la référence, ce qui est obligatoire afin de maintenir une cohérence des mesures. En effet c'est la source de référence avec toutes ses caractéristiques (bruits de phase et fréquence) qui reste la référence au sens littéral. Si j'utilisais un autre signal pour incrémenter le compteur j'ajouterais les bruits de celui-ci à mes mesures et je ne mesurerais plus une source par rapport à une référence mais une source par rapport à une référence affectée du bruit de mon troisième signal.

Lors de la détection d'un front montant sur une source, la valeur de ce compteur est relevée et stockée dans des registres de données temporaires. Le front montant de le source de référence relève également la valeur du compteur mais il va également déclencher le chargement des registres temporaires dans une deuxième série de registres finaux.

Il peut arriver que le compteur arrive en bout de courses entre 2 datations (roll-over) mais la dernière valeur relevée étant systématiquement celle de la référence elle doit être supérieure ou égale à celles des sources, ainsi le contrôle d'un roll-over entre 2 mesures est simpliste. La description de la gestion des roll-over est décrite dans la section software.

Pour clarifier la méthode, deux schémas explicatifs sont représentés sur les figures 6.3 et 6.4.

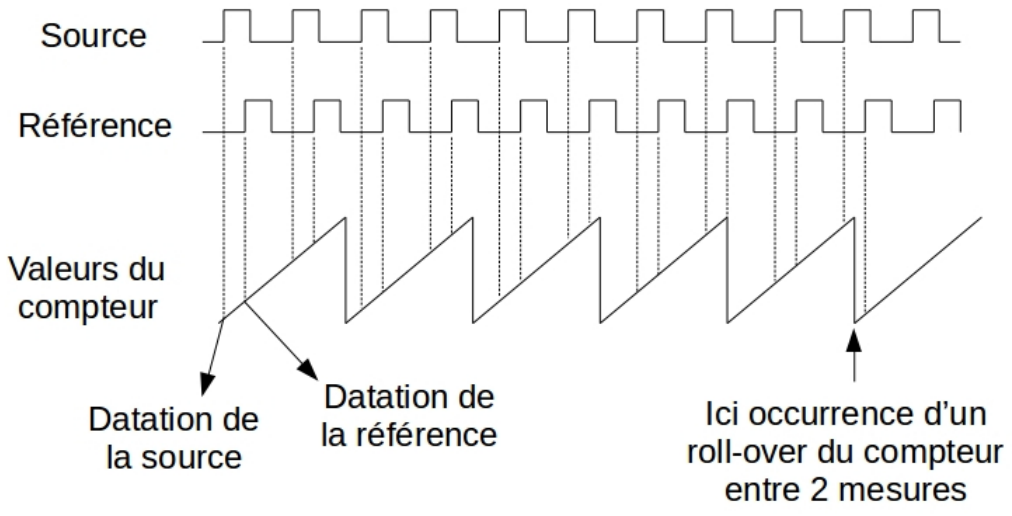


FIGURE 6.3 – Principe du code FPGA

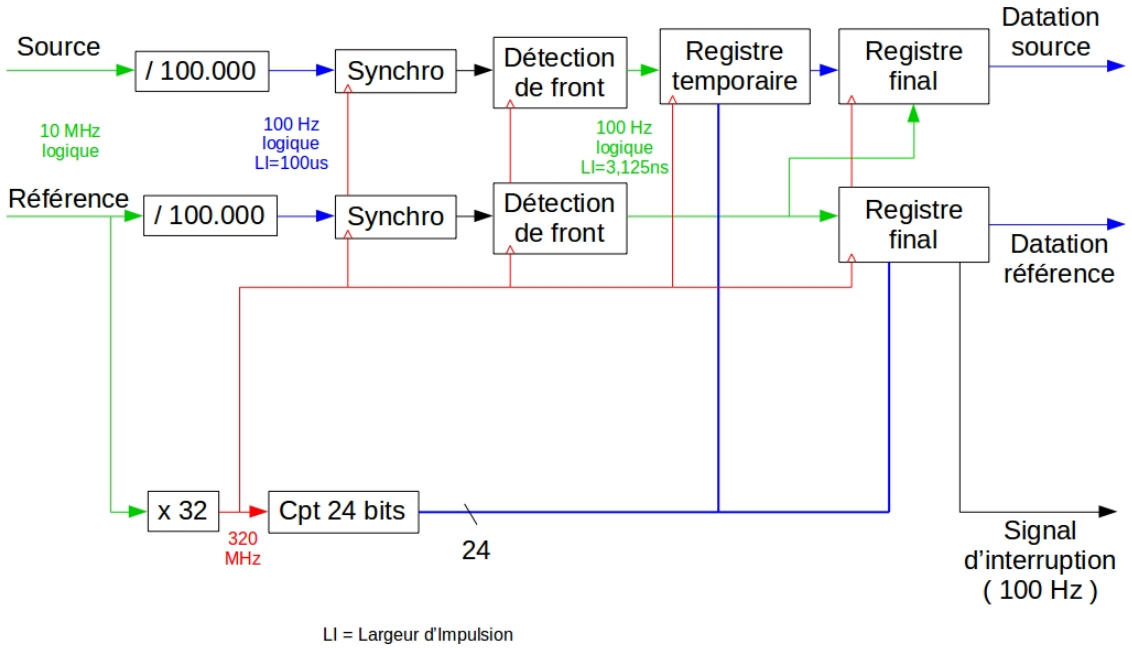


FIGURE 6.4 – Schéma de fonctionnement de la partie FPGA

Les signaux issus des différentes sources sont numérisés en amont par les ZCD qui jouent le rôle de convertisseurs analogique/numérique. Ces signaux sont discriminés en 2 catégories, les sources et les références. L'objectif est d'évaluer les écarts temporels entre chacune des sources et une même référence.

Il a été prévu de permettre l'utilisation d'une référence externe (maser ou césium) afin de caractériser la stabilité moyen et long terme des sources et du VCO [8], dans ce cas le VCO devient une source.

Une fois que la stabilité des sources et du VCO a été estimée, celui-ci prend la place de la référence externe et les écarts temporels sont ensuite évalués par rapport au VCO à asservir. A partir de cet instant la référence externe n'a plus aucune influence et le système devient autonome.

La figure 6.5 représente le "block design" final développé avec l'interface graphique de Vivado. C'est à partir de ce design que le bitstream est créé pour être ensuite "flashé" dans la logique programmable.

8. Cette caractérisation préliminaire permet de fixer les constantes de temps de l'asservissement mais aussi de connaître l'écart de fréquence entre le VCO et la référence externe afin de "caler" la fréquence du VCO au plus proche de celle du césium (exactitude)

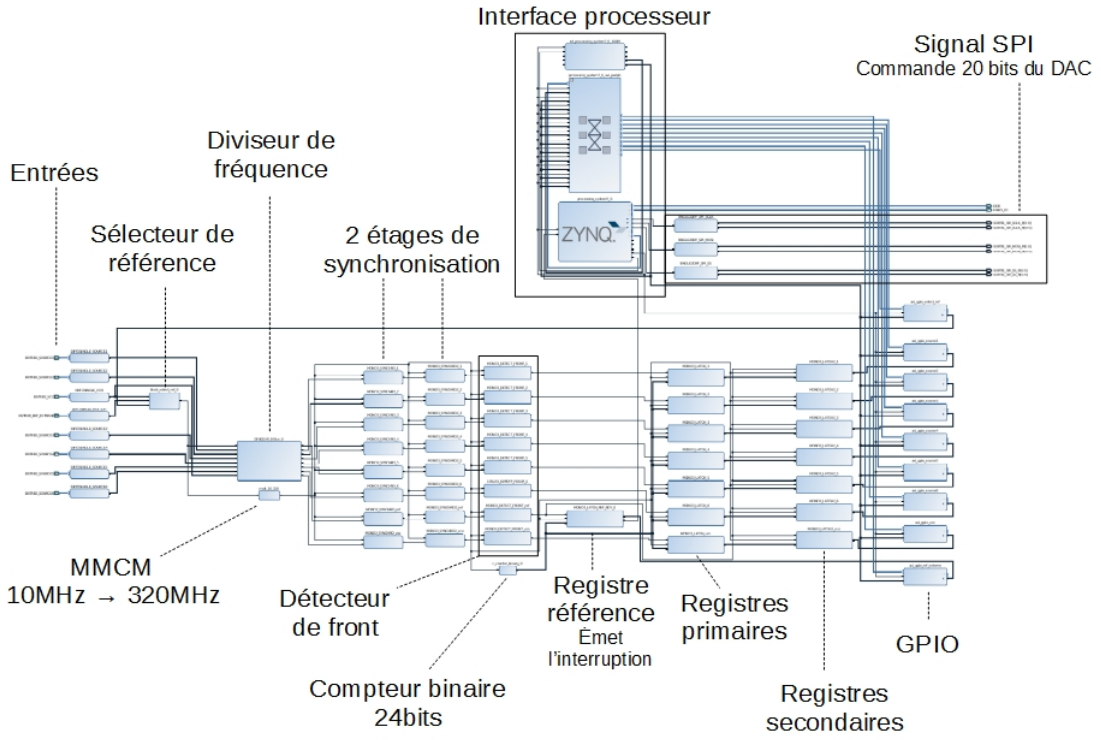


FIGURE 6.5 – Block design sous Vivado de la version finale du code FPGA

L'ensemble de ces blocs ont été écrits en langage Verilog sauf :

— Les buffers d'entrée (et de la sortie SPI) pour convertir les signaux LVDS en single-ended (et inversement).
— Le bloc MMCM de multiplication de fréquence.
— L'interface processeur et les GPIO.

Tous les codes des autres blocs sont disponibles en annexe.

6.3.1 Diviseur de fréquence

Il s'agit de convertir le signal d'entrée à 10 MHz en signal à 100 Hz. C'est ce signal à 100 Hz qui servira à donner l'ordre de datation. Ainsi j'obtiens 100 datations par seconde. Cette valeur a été choisie suffisamment élevée pour pouvoir obtenir assez de valeurs par seconde (pour éventuellement en exclure certaines et atténuer le bruit blanc de mesure par moyennage) et suffisamment faible pour que les données puissent être traitées par l'algorithme en temps réel.

Ces diviseurs sont constitués de compteurs binaires avec une valeur de seuil intermédiaire permettant d'obtenir une durée de l'état haut en sortie de 10 μs (soit un rapport cyclique de 10^{-3}). La division par 100000 est obtenue en remettant à zéro le compteur lorsqu'il atteint cette valeur.

6.3.2 Multiplicateur de fréquence - MMCM

Comme mentionné dans la section précédente la fréquence maximale atteignable à partir d'une entrée 10 MHz est de 320 MHz. La maximisation de la fréquence de ce signal est fondamentale car il en découle directement la résolution de notre intervallomètre. Une fréquence de 320 MHz nous permet une mesure d'écarts temporels avec une résolution de 3,125 ns.
Ce bloc est une boite noire Vivado nommée "Clocking Wizard" avec de nombreuses configurations possibles (PLL ou MMCM, optimisations du jitter). La configuration du MMCM utilisée est celle par défaut (balanced).

6.3.3 Sélecteur de référence

Afin de pouvoir choisir le signal de référence des mesures (VCO ou ref externe) un bloc de sélection est commandé par un GPIO (configuré en sortie). Il faut toutefois faire attention aux phénomènes de "clock gating" et laisser le temps au bloc MMCM de stabiliser sa fréquence de sortie lors du changement de référence. Ces précautions sont gérées depuis la partie logicielle qui laisse un temps mort après l'ordre de changement de référence.

6.3.4 Compteur binaire 24 bits

Ce compteur est incrémenté par le signal de sortie du bloc MMCM sans remise à zéro ni inhibition de l'horloge [9]. C'est la valeur de ce compteur qui est chargée dans les registres (mémoires) lors des fronts montants des différentes sources. La dimension de ce compteur doit être minimale afin d'éviter trop de transfert de retenues (minimiser la carry chain) et de minimiser la logique séquentielle pilotée par le signal d'incrémentation haute fréquence.

La dimension optimale de ce compteur dépend de sa fréquence d'incrémentation et de la périodicité de ses relevés. Dans notre cas la périodicité est de 10 ms et sa fréquence d'incrémentation 320 MHz. Il est donc indispensable que le compteur puisse compter plus de 10 ms sans retour à zéro.

La durée de roll-over d'un compteur 24 bits incrémenté à 320 MHz est de $\frac{2^{24}}{320.10^6}$ = 52,4 ms. Pour information si on diminue sa dimension à 20 bits on obtient une durée de roll-over de 3,3 ms ce qui est insuffisant dans notre cas.

6.3.5 Synchronisations

Une règle importante de l'électronique numérique est le respect des temps de setup et hold des registres. Le temps de setup correspond au temps minimal de présence de la donnée en entrée du registre avant son chargement. Le temps de hold correspond au temps minimal pendant lequel la donnée doit rester présente après son chargement. Ces règles viennent du fait que, entre autres, le chargement des données prend un temps non-nul à être effectué.

Ces durées sont gérées automatiquement par le logiciel Vivado lors de la phase de placement-routage de génération du bitstream. Mais cela implique de posséder un seul domaine d'horloge dans le FPGA afin de permettre à Vivado de gérer correctement ses timings internes.

Chacune des sources étant complètement indépendante elle génère chacune leur propre domaine d'horloge. Afin de synchroniser tous ces signaux à un même signal j'ai utilisé le signal de sortie du bloc MMCM (320 MHz). Ainsi les signaux (fronts montants et descendants asynchrones)

9. Le compteur est dit en roue libre, il se remet naturellement à zéro après que sa valeur maximale (tous les bits à 1) a été atteinte. La durée entre 2 remises à zéro s'appelle durée de roll-over.

sortant du diviseur de fréquence sont "recalés" sur les fronts montant du signal d'horloge avec une simple bascule.

Deux étages de synchronisation successifs sont utilisés afin d'améliorer la qualité de cette synchronisation.

6.3.6 Détection de fronts

Le signal de détection du front montant d'une source sert à donner l'ordre de chargement des valeurs du compteur 24 bits dans les registres de données. Ces chargements s'effectuent de manière synchrone au signal d'horloge en sondant le niveau logique du signal issu du bloc de détection. Ainsi la largeur d'impulsion du signal de détection est équivalente à une période du signal d'horloge (3,125 ns) afin de charger une seule fois la valeur du compteur par front montant de la source à dater.

6.3.7 Registres

Ils permettent de mémoriser les valeurs du compteur lors de l'occurrence d'un front montant des sources et de la référence.

Sources

Comme mentionné précédemment, chacune des sources donne l'ordre de mémorisation des valeurs du compteur de façon indépendante dans des registres temporaires. Ensuite la détection d'un front montant sur le signal de référence transfert ces valeur dans des registres secondaires (finaux) pour sécuriser les données.

Référence

La périodicité du relevé ordonné par le signal de référence est parfaitement connu (1 relevé toutes les 10 ms), ainsi la chaîne d'acquisition pour le signal de référence ne possède pas de registre temporaire [10].

Ce registre unique à également pour rôle de générer le signal d'interruption une fois la valeur du compteur stockée.

6.3.8 Acquisitions

L'élément permettant le transfert des données présentes dans les registres vers l'algorithme se nomme GPIO (Global Purpose Input Output), il communique avec le processeur du Zynq via le bus AXI de Xilinx. Chaque GPIO possède sa propre adresse et à une capacité maximale de 32 bits.

Ici se termine la description de l'ensemble de la partie hardware (composants électroniques et code FPGA) développée au cours de ma thèse. A ce stade de l'avancement du projet nous disposons de deux prototypes fonctionnels fournissant de nombreuses mesures d'écarts temporels (100 par seconde) de manière fiable et précise (sans glitch et avec une résolution de 3,125 ns).
De plus le pilotage de la tension de commande du VCO s'effectue avec une granularité permettant d'atteindre des instabilités de fréquence relative théoriques de 4,5 10^{-14} (prototype ZedBoard + BVA) et 1,2 10^{-12} (prototype MicroZed + Quartz commercial).

10. Cette durée de 10ms est largement suffisante pour assurer la récupération des données entre 2 mesures.

Troisième partie

Software

Chapitre 7

Mise en forme des mesures

Ce court chapitre décrit la mise en forme des données issues de la partie hardware qui ne fait pas réellement partie de l'asservissement à proprement parler, j'ai donc choisi de l'en dissocier. Les traitements effectués lors de cette mise en forme font notamment le lien entre écarts temporels et écarts de fréquences, ce sont ces derniers qui constituent les principales données de l'algorithme d'asservissement.

7.1 Écart brut des compteurs

Comme nous l'avons vu dans la partie précédente, le code FPGA fournit à l'exécutable la datation de l'ensemble des sources, la datation de la référence et un signal d'interruption nous informant de l'occurrence d'une nouvelle mesure. A partir de ces datations j'en déduis un écart brut x_{cpt}, quantifié en tics compteur, entre chacune de mes sources et ma référence (unique), en prenant soin de gérer l'occurrence éventuelle des roll-overs.

La détection de ces roll-overs est simple car, dans le principe utilisé pour le code FPGA, la datation de la référence est forcément postérieure à celle des sources. Ainsi on obtient toutes les 10 ms l'écart temporel brut de chacune des sources par rapport à la référence :

Si $(date_ref > date_source)$ $\quad x_{cpt} = date_ref - date_source$
sinon $\quad x_{cpt} = date_ref - date_source + 2^{nb_bits_compteur}$

Dans mon cas $nb_bits_compteur = 24$ soit $2^{24} = 16777216$

x_{cpt} est exprimé ici en nombre ($\in \mathbb{N}$) de périodes de l'horloge compteur ($3,125.10^{-9}s$). Je pourrais convertir cette valeur directement en secondes mais j'ai choisi de conserver cette forme équivalente plus simple et rapide à stocker et à traiter.

7.2 Affinage et contrôle des écarts bruts

7.2.1 Principe

A ce stade j'ai accès à une mesure d'écart brut toutes les $\tau_0 = 10$ millisecondes. Cette cadence est assez élevée par rapport à la cadence recherchée de l'asservissement de mon VCO. Je peux donc profiter de cette haute densité temporelle d'information pour affiner et contrôler la validité de cette mesure sur un intervalle de temps plus long.

La cadence d'asservissement du VCO étant directement liée à la qualité (stabilité) de celui-ci et des sources sur lesquelles je souhaite l'asservir, je dois définir une « durée d'affinage ». Cette durée doit être suffisamment longue pour réaliser un contrôle statistiquement pertinent et suffisamment courte pour que l'instabilité de ma référence reste inférieure à celle de mes sources. Elle est donc complètement liée aux performances du VCO et des sources utilisées.

Un exemple : Je souhaite évaluer la stabilité d'une horloge à jet de césium sur le court terme en utilisant un quartz (en prenant le BVA 10 MHz, plus stable à court terme que le césium [1]). Je vais tout d'abord observer les domaines de stabilité de chacune de ces deux sources indépendamment. Je m'aperçois que le quartz devient moins stable que le césium sur des durées supérieures à 100 secondes, ce qui va représenter la durée maximale d'observation des écarts de fréquence, au-delà de cette durée la fréquence du quartz va se mettre à dériver et la mesure sera faussée.

Revenons à notre système. En considérant le quartz AR 10MHz et les rubidiums utilisés, je détermine une durée optimale d'intégration τ_{reg} de 10 secondes [1]. Je vais donc stocker mes mesures d'écarts bruts pendant 10 secondes et y appliquer un traitement sur cette durée.

J'ai choisi d'effectuer une régression linéaire sur ce jeu de valeurs. C'est une méthode de synthèse simple à mettre en œuvre et qui permet d'avoir des informations sur la qualité de la régression. Cependant cela implique de faire l'hypothèse que l'évolution des écarts temporels bruts est linéaire sur la durée d'intégration considérée (10 secondes) [2].

7.2.2 Régression linéaire

Cette méthode basique demande très peu de temps de calcul au processeur et me permettra de définir des quantités utilisées dans le chapitre suivant.

Les $N = 1000$ échantillons [3] $x_i^{cpt}, \quad i = 1..N$ sont estimés par un modèle de la forme $\alpha t_i + \beta$ avec $t_i = i\tau_0 = i10^{-2}$ la date d'acquisition. Pour une séquence de N mesures (x_i^{cpt}, t_i) on cherche (α, β) tels que la quantité

$$\sum_{i=1}^{N} \left[x_i^{cpt} - (\alpha t_i + \beta)\right]^2$$

soit minimale. C'est le critère des moindres carrés.

En posant

$$\bar{x}_{cpt} = \frac{1}{N}\sum_{i=1}^{N} x_i^{cpt} \qquad \bar{t} = \frac{1}{N}\sum_{i=1}^{N} t_i$$

$$s_t^2 = \frac{1}{N-1}\sum_{i=1}^{N}(t_i - \bar{t})^2$$

$$s_{tx} = \frac{1}{N-1}\sum_{i=1}^{N}(t_i - \bar{t})(x_i^{cpt} - \bar{x_{cpt}})$$

1. Voir courbes de déviation d'Allan dans la partie Introduction
2. Les bruits court terme seront inclus plus tard dans l'algorithme de Kalman et on peut considérer que la marche aléatoire et la dérive de fréquence n'ont pas encore d'impact sur une durée de 10 secondes
3. 1 échantillon toutes les 10ms pendant 10 secondes

les moindres carrés sont minimisés par

$$\hat{\alpha} = \frac{s_{tx}}{s_t^2}$$

$$\hat{\beta} = \bar{x}_{cpt} - \hat{\alpha}\bar{t}$$

Ce qui donne une estimation linéaire de l'évolution des écarts temporels bruts

$$\hat{x_i^{cpt}} = \hat{\alpha}t_i + \hat{\beta}$$

La quantité $\bar{x}_{cpt}$ (la moyenne, tout simplement) fournit une estimation affinée de l'écart temporel brut sur 10 secondes. En effet, un simple moyennage permet de réduire l'impact du bruit blanc de mesure. C'est cette quantité qui sera traitée par l'algorithme d'asservissement.

J'aurais aussi pu utiliser la projection de la droite de régression (horizon ou valeur finale) comme estimation affinée de l'écart temporel mais l'incertitude sur cette quantité est liée aux incertitudes sur la pente $\hat{\alpha}$ de ma régression linéaire [4], contrairement à sa valeur moyenne (milieu de la droite).

Le modèle fournit par la minimisation des moindres carrées permet surtout d'estimer la qualité de la régression effectuée. Pour cela je vais utiliser la variance de mes résidus

$$s^2 = \frac{1}{N-2}\sum_{i=1}^{N}\left(x_i^{cpt} - \hat{x_i^{cpt}}\right)^2 \tag{7.1}$$

comme estimation de l'erreur de mesure.

Si cette variance dépasse un certain seuil (fixé empiriquement pour l'instant), je considère la mesure comme défectueuse et j'extrapole à partir de la mesure précédente (valeur de l'écart temporel brut et de la pente). Sinon je prends la valeur moyenne des échantillons comme mesure de l'écart temporel.

Ainsi j'obtiens une mesure précise et robuste [5] de l'écart temporel brut entre la source et la référence. Cette mesure servira de base à tout l'algorithme qui va suivre.

7.3 Écarts temporels

Connaissant la fréquence d'incrémentation de mon compteur, on obtient l'écart temporel x_k en secondes [6] :

$$x_k = \frac{\bar{x}_{cpt}}{F_{cpt}} \quad avec \quad F_{cpt} = 320.10^6 Hz$$

On peut également définir la résolution de la mesure d'écart temporel, qui correspond à la résolution du compteur :

$$R_x = \frac{1}{F_{cpt}} = 3,125ns \tag{7.2}$$

4. notamment lorsque les écarts temporels évoluent très lentement (signaux de fréquences proches)

5. aux sauts et glitch de phase

6. Il s'agit ici d'un temps discret, l'écart temporel $x(t)$ devient $x(k\tau_{reg})$ avec k entier naturel et τ_{reg} le temps écoulé entre 2 mesures (soit 10 secondes dans mon cas).

A partir de la grandeur fondamentale x_k et de son évolution dans le temps (voir partie métrologie temps-fréquence) on peut en déduire les 3 autres grandeurs fondamentales du domaine temps-fréquence (écart de phase, écart de fréquence instantanée et écart de fréquence instantanée normalisée), selon les besoins de l'algorithme.

7.4 Déduction des écarts de fréquence

A partir de l'évolution des mesures d'écarts temporels on déduit l'écart de fréquence :

$$\Delta\nu = \nu_0 \frac{dx(t)}{dt}$$

Dans le domaine numérique (discret) cette relation s'écrit :

$$\Delta\nu_k = \nu_0 \frac{x_k - x_{k-1}}{\tau_{reg}} = \nu_0 \frac{(1 - z^{-1})}{\tau_{reg}} x_k \tag{7.3}$$

avec $\tau_{reg} = 10s$ et $\nu_0 = 10.10^6 Hz$

De la même façon que pour la mesure d'écart temporel, on peut définir la résolution de la mesure d'écart de fréquence.

$$R_{\Delta\nu} = \nu_0 \frac{R_x}{\tau_{reg}} = \nu_0 \frac{1/F_{cpt}}{\tau_{reg}} = 3,125 mHz \tag{7.4}$$

Cette valeur correspond à la valeur minimale d'écart de fréquence détectable sur la période donnée (10 secondes). Si l'écart est plus faible que la résolution, notre mesure sera nulle pour un certain nombre d'itérations. C'est très important car cela implique des phénomènes tout ou rien que notre algorithme d'asservissement devra compenser. Dans la pratique on ne peut pas s'assurer qu'une source aura un écart de fréquence suffisant, ce qui peut être le cas pour des horloges de très bonne qualité (rubidium, maser et césium par exemple) la robustesse de notre algorithme sera donc indispensable de ce point de vue.

En passant on peut voir qu'augmenter la durée τ_{reg} et la fréquence du compteur F_{cpt} permet d'améliorer la résolution en fréquence. Ces valeurs ont été choisies comme maximales tout en assurant l'intégrité et la disponibilité des mesures.

Chapitre 8

Algorithmes d'asservissement

Le rôle de cet algorithme est de synthétiser les écarts de fréquence $\Delta\nu_k$ de chaque source pour en déduire une correction unique à appliquer au VCO. La figure 8.1 permet de replacer la partie asservissement parmi les autres éléments.

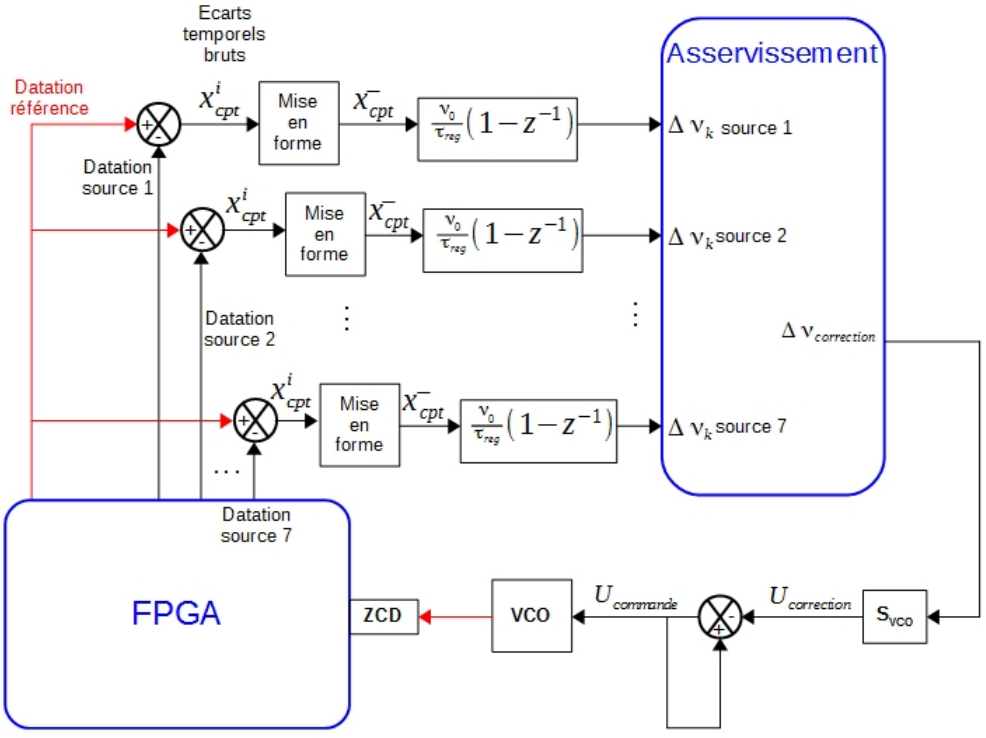


FIGURE 8.1 – Diagramme partie software

8.1 Filtre de Kalman

Le filtrage de Kalman est un algorithme prédictif [1] utilisant des mesures bruitées afin de produire une estimation d'une série de paramètres. Il est nommé d'après Rudolf Emil Kálmán, un

1. au sens où il essayera de suivre un modèle donné initialement

mathématicien et électrotechnicien Américain d'origine Hongroise. On lui attribue ainsi la paternité de cet algorithme [11] [12] mais celle-ci est controversée[13].

Les applications du filtrage de Kalman sont nombreuses, les plus connues sont le système de navigation des missions Apollo et le système de localisation par GPS. On peut aussi citer les systèmes de guidage de missiles balistiques et les prédictions d'évolution de marchés boursiers.

Plus proche de notre application le filtrage de Kalman est utilisé pour synchroniser des horloges distantes [14] [15] [16] et créer des échelles de temps [17] [18] [19] [20]. Cependant je n'ai trouvé aucune publication décrivant l'utilisation d'un filtre de Kalman afin d'asservir un oscillateur local sur un ensemble d'horloges.

8.1.1 Principe et description

Le filtrage de Kalman fait évoluer un vecteur d'état[2] X_k en fonction d'un modèle prédictif puis y applique une mise à jour en fonction des mesures effectuées.

Le vecteur d'état

Les composantes du vecteur d'état doivent pouvoir être modélisées selon un modèle connu d'évolution temporelle afin de permettre la prédiction de l'état suivant. Une approche générique consiste à utiliser un vecteur d'état à 3 composantes

$$X_k = \left\{ \begin{array}{c} x_k \\ \Delta\nu_k \\ D_\varphi \end{array} \right\}$$

Avec x_k l'écart temporel, $\Delta\nu_k$ l'écart de fréquence et D_φ la dérive de fréquence.

La matrice de transition

Cette matrice est utilisée dans la phase de prédiction. Elle implémente le modèle prédictif dynamique qui lie l'état X_k à l'état X_{k-1}.

Le modèle d'évolution généralement adopté est de considérer une dérive déterministe linéaire, autrement dit la composante D_φ est constante dans le temps et

$$\left\{ \begin{array}{l} \Delta\nu = \nu_0 \frac{dx}{dt} \\ D_\varphi = \frac{d\Delta\nu}{dt} \end{array} \right.$$

En transposant l'équation du modèle dans le domaine discret on obtient :

$$\left\{ \begin{array}{l} \Delta\nu_k = \nu_0 \frac{(x_k - x_{k-1})}{\tau_{reg}} \\ D_\varphi = \frac{\Delta\nu_k - \Delta\nu_{k-1}}{\tau_{reg}} \end{array} \right.$$

$$\left\{ \begin{array}{r} x_k = \frac{\Delta\nu_k . \tau_{reg}}{\nu_0} + x_{k-1} \\ \Delta\nu_k = D_\varphi . \tau_{reg} + \Delta\nu_{k-1} \end{array} \right.$$

$$\left\{ \begin{array}{r} x_k = \frac{(D_\varphi . \tau_{reg} + \Delta\nu_{k-1})\tau_{reg}}{\nu_0} + x_{k-1} = \frac{\tau_{reg}^2}{\nu_0} D_\varphi + \frac{\tau_{reg}}{\nu_0} \Delta\nu_{k-1} + x_{k-1} \\ \Delta\nu_k = D_\varphi . \tau_{reg} + \Delta\nu_{k-1} \end{array} \right.$$

2. composé des variables d'état

Soit un modèle d'évolution des 3 variables d'état :

$$\left\{ \begin{array}{c} x_k = \frac{\tau_{reg}^2}{\nu_0} D_\varphi + \frac{\tau_{reg}}{\nu_0} \Delta\nu_{k-1} + x_{k-1} \\ \Delta\nu_k = D_\varphi . \tau_{reg} + \Delta\nu_{k-1} \\ D_\varphi = D_0^\varphi \end{array} \right.$$

On définit la matrice de transition

$$\Phi_k = \left\{ \begin{array}{ccc} 1 & \frac{\tau_{reg}}{\nu_0} & \frac{\tau_{reg}^2}{\nu_0} \\ 0 & 1 & \tau_{reg} \\ 0 & 0 & 1 \end{array} \right\}_k$$

telle que

$$X_k = \Phi_k X_{k-1}$$

Le vecteur d'observation

Ce vecteur contient le modèle d'observation des grandeurs considérées, il fait le lien entre les informations fournies par les outils de mesure (capteurs) et leurs interactions avec l'algorithme. Dans notre cas la mesure effectuée concerne uniquement l'information fréquentielle, on "inhibe" donc les composantes d'écarts temporels et de dérive qui ne sont pas observées. On définit ainsi le vecteur d'observation :

$$H_k = \left\{ \begin{array}{ccc} 0 & 1 & 0 \end{array} \right\}_k$$

La matrice d'estimation d'erreur

Cette matrice accompagne chaque prédiction et chaque mise à jour du vecteur d'état. Elle représente la variance de l'erreur commise lors de leurs estimations. Elle est noté $P_{k|k-1}$ pour l'erreur de prédiction et $P_{k|k}$ pour l'erreur de mise à jour.

Les matrices de bruits

Deux matrices sont utilisées afin de quantifier d'une part le bruit de process et d'autre part le bruit d'observation, ces bruits sont quantifiés par leur variance sur la durée entre 2 itérations du filtre. Dans l'application développée ces matrices sont invariantes entre chaque itération du filtre. Elles sont initialisées une fois pour toute au lancement de l'asservissement.

La matrice Q_k représente le bruit de process. J'ai interprété ceci comme la somme des bruits (variance) issus de la source et de la référence (toujours sur la durée d'itération) libres. Par exemple, si ma source possède une stabilité de fréquence 10^{-13} sur 1 seconde (maser) et que la référence de mes mesure est le quartz BVA 10MHz possédant une stabilité de 10^{-12} @ 1s alors j'initialise la matrice Q_k à $1,1.10^{-12}$ sur la composante fréquentielle.

Le scalaire R_k représente le bruit d'observation. Ceci à été interprété comme le bruit de mesure (résolution en fréquence) de l'intervallomètre [3]. La valeur théorique de ce bruit de mesure est de $3,125$ mHz, soit une instabilité relative de $3,125.10^{-10}$.

3. Voir chapitre "Mise en forme des écarts temporels"

Ces 2 matrices ont récemment [4] constitué un autre moyen de paramétrer le filtre que le coefficient appliqué au gain de Kalman et à l'efficacité de commande (voir paragraphes suivants).

8.1.2 Le déroulement du calcul

Dans la suite on notera $X_{k1|k2}$ le vecteur d'état à l'instant discret $k1$ prenant en compte les mesures jusqu'à l'instant $k2$ avec $k2 \leq k1$.

L'algorithme de Kalman se décompose en 2 phases autour de l'instant de mesure :

. **prédiction**
Prédiction de l'état actuel$X_{k|k-1}$ à partir du précédent $X_{k-1|k-1}$ et estimation de la variance de l'erreur de prédiction $P_{k|k-1}$

$$X_{k|k-1} = \Phi_k X_{k-1|k-1} + U_k \tag{8.1}$$

$$avec \quad U_k = \left\{ \begin{array}{c} 0 \\ \Delta\nu_{correction} \\ 0 \end{array} \right\} \quad \text{le vecteur de correction}$$

$$P_{k|k-1} = \Phi_k P_{k-1|k-1} \Phi_k^T + Q_k \tag{8.2}$$

U_k représente la dernière correction en fréquence appliquée au VCO. Ainsi la prédiction du vecteur d'état prend aussi en compte les modifications appliqués par l'utilisateur.

. **mesure**
Dans le cas où la mesure serait inexploitable l'algorithme peut utiliser la prédiction $X_{k|k-1}$ effectuée précédemment en attendant une valeur jugée exploitable [5]. Mais dans notre cas l'occurrence et la validité des mesures est assurée par la mise en forme effectuée en amont de l'asservissement.

La mesure de l'état courant X_k se résume à la mesure de l'écart de fréquence.

$$X_k = \Delta\nu_k \tag{8.3}$$

. **mise à jour**
Mise à jour de l'estimation de l'état actuel $X_{k|k}$ et mise à jour de l'erreur $P_{k|k}$

Erreur résiduelle mesurée (innovation, scalaire)

$$\varepsilon_k = X_k - H_k X_{k|k-1} \tag{8.4}$$

Variance de l'erreur résiduelle (scalaire)

$$S_k = H_k P_{k|k-1} H_k^T + R_k \tag{8.5}$$

Gain de Kalman optimal

$$K_k = P_{k|k-1} H_k^T S_k^{-1} \tag{8.6}$$

Le gain de Kalman K_k va réguler l'impact de la mesure ε_k sur le réajustement du vecteur d'état. Sous sa forme optimale il établit un rapport entre un niveau de confiance accordé aux

4. Avec l'évolution de ma compréhension du filtrage de Kalman (janvier 2017)
5. Ce qui peut être le cas lors de la perte d'un signal GPS par exemple

mesures (exprimé par le biais de R_k) et celui accordé au modèle prédictif (exprimé dans Q_k). Il est optimal d'un point de vue temps de réponse/stabilité.
La méthode de paramétrage que j'utilise peut permettre l'application d'un coefficient C_k (compris entre 0 et 1) sur le gain optimal, détruisant ainsi son optimalité [6].

Il est également important de noter que la quantité S_k doit être inversible (non nulle) ce qui peut poser des problèmes selon la valeur de R_k et l'évolution de la matrice $P_{k|k-1}$ qui peuvent faire tendre S_k vers zéro.

Gain de Kalman effectif

$$K_k^{'} = C_k K_k \tag{8.7}$$

Estimation de l'état courant

$$X_{k|k} = X_{k|k-1} + K_k^{'}\varepsilon_k \tag{8.8}$$

C'est cette forme du vecteur d'état [7] qui sera à la base de la correction de fréquence à apporter au VCO.
La matrice de variance d'erreur de mise à jour

$$P_{k|k} = (Id - K_k^{'}H_k)P_{k|k-1}(Id - K_k^{'}H_k)^T + K_k^{'}R_k K_k^{'T} \tag{8.9}$$

L'utilisation d'un gain non-optimal implique l'utilisation de cette expression (forme de Joseph).

Les équations 8.1 à 8.9 représentent les opérations nécessaires au filtrage de Kalman qui seront implémentées dans l'application (langage C).

Lors de l'utilisation de plusieurs sources en entrée on utilisera plusieurs filtres de Kalman fonctionnant en parallèle. Les corrections issues de chacun des filtres seront alors moyennées pour obtenir une correction unique à appliquer au VCO.

8.1.3 Paramétrage

Paramétrage interne au filtre

Il existe différents paramètres permettant d'ajuster la réponse du filtre de Kalman. Les seuls éléments avec lesquels on ne doit pas interférer sont le vecteur d'état dans sa version estimée $X_{k|k-1}$ et mise à jour $X_{k|k}$ ainsi que les matrices de covariance d'erreur associées $P_{k|k-1}$ et $P_{k|k}$. Ces matrices sont initialisées [8] avant d'entrer dans la boucle d'asservissement et elles constituent la sortie du filtre de Kalman.

Tout au long du déroulement de l'algorithme (initialement et à chaque itération) on peut librement modifier l'ensemble des autres éléments, à savoir :
— Le gain de Kalman K_k
— La matrice de transition Φ_k
— La matrice d'observation H_k

6. On va augmenter le temps de réponse du filtre et diminuer la sensibilité du système au bruit de mesure court terme
7. plus particulièrement la composante fréquentielle ν_k
8. Ce qui peut constituer un façon de paramétrer l'algorithme

— Les matrices de bruits de mesure R_k et de process Q_k

Dans notre cas le paramétrage s'est effectué sur l'initialisation des paramètres, et non sur une modification "en temps réel", trop délicate dans un premier temps.

— Application d'un coefficient constant au gain de Kalman optimal.
— Initialisation des matrices de bruits R_k et Q_k
— L'efficacité de correction (décrite dans le paragraphe suivant) revient à brider la correction appliquée au VCO afin de réduire artificiellement le bruit induit par l'asservissement.

Commande et efficacité

La composante fréquentielle ν_k du vecteur d'état déterminée à l'équation 8.8 constitue la base de la correction de fréquence à appliquer au VCO. Cet écart de fréquence filtré est converti en un mot de 20 bits via le coefficient de sensibilité du VCO qui fait correspondre l'écart de fréquence en Hertz à une tension en Volts, puis le coefficient de conversion du CNA qui fait correspondre un mot de 20 bits à cette valeur de tension voulue.

Cette correction de fréquence vient s'ajouter à la valeur de commande courante afin d'ajuster la fréquence du VCO.

L'impact de la correction peut être contrôlée via un autre coefficient appelé efficacité de commande [9] qui a pour rôle de minimiser l'amplitude des corrections de fréquence. Ce coefficient est fixé de manière empirique afin de réduire l'instabilité induite par l'asservissement (visible sur les courbes de variances d'Allan pour des durées d'intégration de l'ordre de la constante de temps de l'asservissement).

8.1.4 Simulations

Cette simulation est une étape intermédiaire entre la théorie exposée précédemment et l'implémentation concrète sur le système (en langage C).

Dans un premier temps l'objectif est d'appréhender la réponse du filtre de Kalman sur le court terme (bruit blanc de fréquence et sauts de phase issus de l'intervallomètre) ainsi que sa capacité à suivre une référence long terme (erreur statique nulle). Ensuite on observera l'impact du paramétrage sur la sortie du filtre.

La modélisation complète du système rebouclé impliquerait de prendre en compte l'ensemble des éléments (DAC, quartz, ZCD, FPGA et le processeur) et leurs contributions individuelles au bruit global, en plus des bruits intrinsèques aux différentes sources de fréquence utilisées. Ceci a fait l'objet d'une collaboration avec des automaticiens de l'institut Femto-ST au début de ma thèse mais les problématiques hardware ont pris le pas sur cette collaboration. Il est toutefois ressorti plusieurs points intéressants :

— L'approche "temps-fréquence" de l'automatique diffère de l'approche "automate" classique. Principalement par la nature des perturbations (marche aléatoire et dérives) et par les quantités faibles de ces bruits [10]. Ainsi un correcteur instable d'un point de vu de l'automatique classique pourrait "devenir" stable dans une application temps-fréquence et, in-

9. En toute rigueur il s'agit plutôt d'une efficacité de correction, ce coefficient est généralement compris entre 0 et 1

10. Les instabilités de fréquence relatives sont inférieures à 10^{-12}

versement, des évolutions considérées comme instables d'un point de vue temps-fréquence seraient estimées stables d'un point de vue automatique.
— Modéliser et simuler l'ensemble du système permettrait d'anticiper sa réponse long terme, ce qui éviterait d'avoir à réaliser des mesures sur le système réel pendant des durées rédhibitoires (1 mois de mesure permet l'estimation de la stabilité sur 1 semaine seulement).

Les simulations suivantes sont donc celles d'un filtre de Kalman en boucle ouverte, avec une seule entrée modélisée par une dérive quadratique de phase (soit une évolution linéaire de la fréquence et une dérive de fréquence constante). On introduira ensuite un bruit blanc de fréquence censé représenter le bruit intrinsèque de la source ajouté au bruit de mesure (supposé blanc) issu de l'intervallomètre. Enfin on introduira des perturbations habituellement présentes (glitches, saut de phase et fréquence, rupture de pente en fréquence). Le tout est simulé sur 10000 itérations avec un pas unitaire.

La matrice de transition Φ_k et la matrice d'observation H_k sont celles définies dans la section précédente.

Les matrices de bruit de process Q_k et de mesure R_k sont fixées comme suit :

$$Q_k = Q_0 = \begin{Bmatrix} 0 & 0 & 0 \\ 0 & \left(1,1.10^{-5}\right)^2 & 0 \\ 0 & 0 & 0 \end{Bmatrix} \quad \text{et} \quad R_k = R_0 = (3,125.10^{-3})^2$$

Plusieurs remarques concernant ces 2 matrices :

— L'utilisation des écarts de fréquence non-normalisés afin de rester homogène avec le vecteur d'état.
— Ces matrices doivent être homogènes aux matrices $P_{k|k-1}$ et $P_{k|k}$ composées respectivement des variances d'erreur sur l'état estimé et sur l'état mis à jour, d'où l'élévation au carré.
— Concernant la matrice Q_k seule sa composante fréquentielle est définie.
— Le scalaire R_k intervient dans le calcul de $S_k = H_k P_{k|k-1} H_k^T + R_k$ qui doit être inversible.

La matrice d'erreur $P_{k|k}$ est initialisée comme étant la matrice identité et le vecteur d'état est initialisé à 0 sur toutes ses composantes :

$$P_{0|0} = \begin{Bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{Bmatrix} \quad \text{et} \quad X_{0|0} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}$$

Enfin le coefficient appliqué au gain de Kalman optimal est fixé à 1 : $C_k = 1$

Les courbes qui suivent représentent l'évolution de la composante fréquentielle du vecteur d'état ($\Delta\nu_k$) superposée au modèle fixé d'évolution de fréquence linéaire

$$\Delta\nu_k^{\text{modèle}} = 2.10^{-6}k + 0.1$$

auquel s'ajoute un bruit blanc de fréquence d'écart type $3.10^{-3}Hz$, somme des bruits exprimés dans Q_k et R_k.

La quantité $\left|\Delta\nu_k - \Delta\nu_k^{\text{modèle}}\right|$ permet d'estimer la performance du filtrage pour chaque paramétrage.

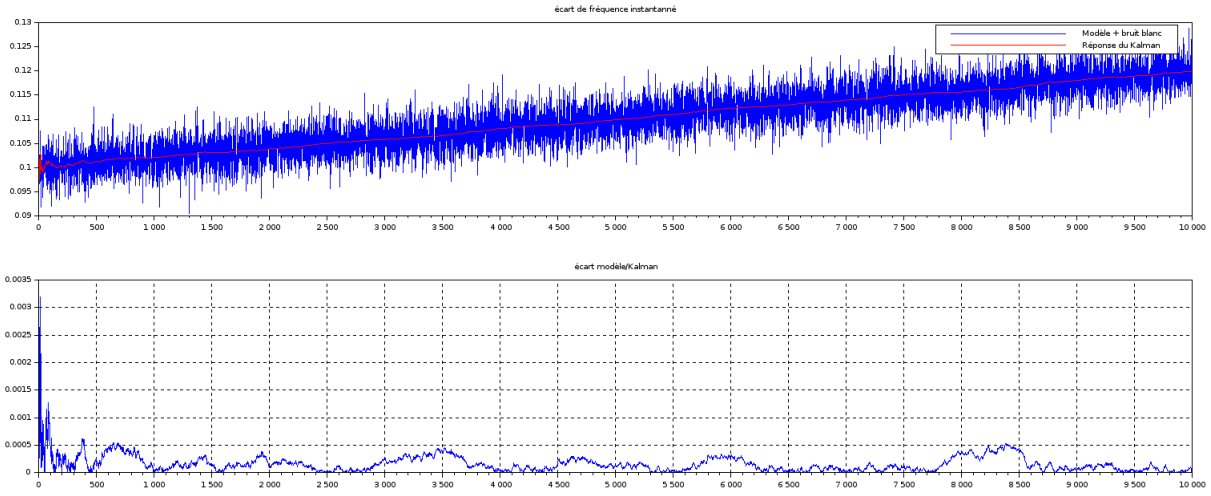
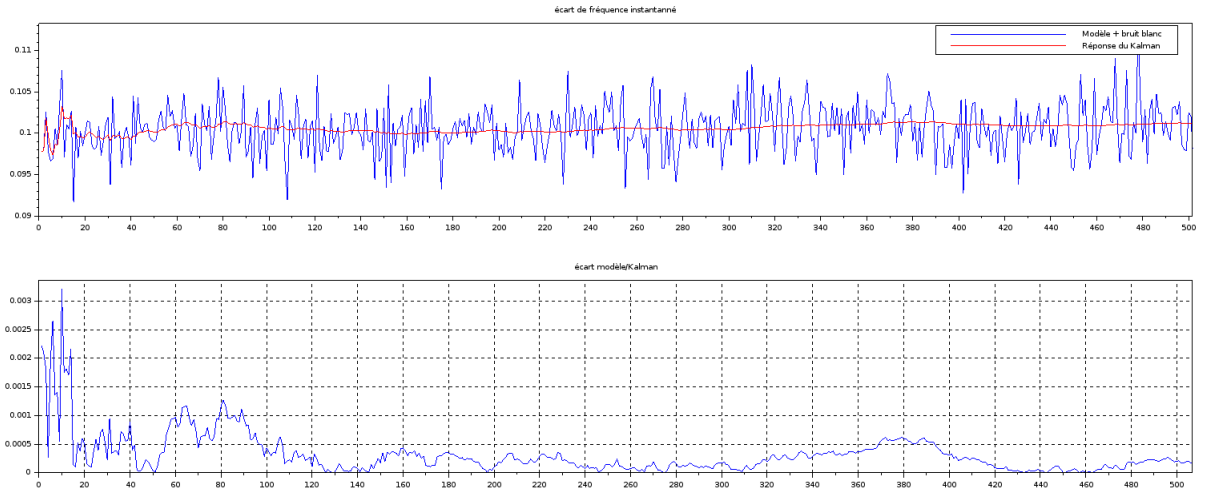


FIGURE 8.2 – Réponse du filtre de Kalman à une rampe de fréquence avec bruit blanc



En zoomant sur les 500 premières itérations :

Le filtre met une centaine d'itérations avant de se stabiliser. Ce régime transitoire pourrait être dû au fait d'avoir initialisé le vecteur d'état à zéro alors que le modèle en fréquence vaut initialement 0.1 Hz. En pratique l'initialisation du vecteur d'état imposerait une phase préliminaire de mesure des écarts de fréquence entre chacune des source et la référence, ce qui est tout a fait envisageable. Pour les simulations suivantes on initialisera la composante fréquentielle du vecteur d'état correctement :

$$X_{0|0} = \begin{Bmatrix} 0 \\ 0.1 \\ 0 \end{Bmatrix}$$

Malgré cette initialisation le filtre présente toujours un régime transitoire, ce qui demande d'analyser l'évolution des variables internes du Kalman, à savoir les matrices $P_{k|k}$ et K_k. Ces

deux matrices semblent converger vers une valeur fixe indépendante de l'amplitude du bruit blanc introduit dans le modèle. Ces valeurs sont :

$$P_{10000|10000} = \begin{Bmatrix} 1 & 9,766.10^{-13} & 1.210.10^{-17} \\ 9,766.10^{-13} & 3,535.10^{-8} & 3,638.10^{-12} \\ 1,210.10^{-17} & 3,638.10^{-12} & 1,283.10^{-14} \end{Bmatrix}$$

$$K_{10000} = \begin{Bmatrix} 0 & 1.10^{-7} & 0 \\ 0 & 3,6195.10^{-3} & 0 \\ 0 & 4.10^{-7} & 0 \end{Bmatrix}$$

Afin d'observer la vitesse de convergence de ces matrices on observera l'évolution de leur trace. La figure 8.3 montre leur évolution sur les 500 premières itérations.

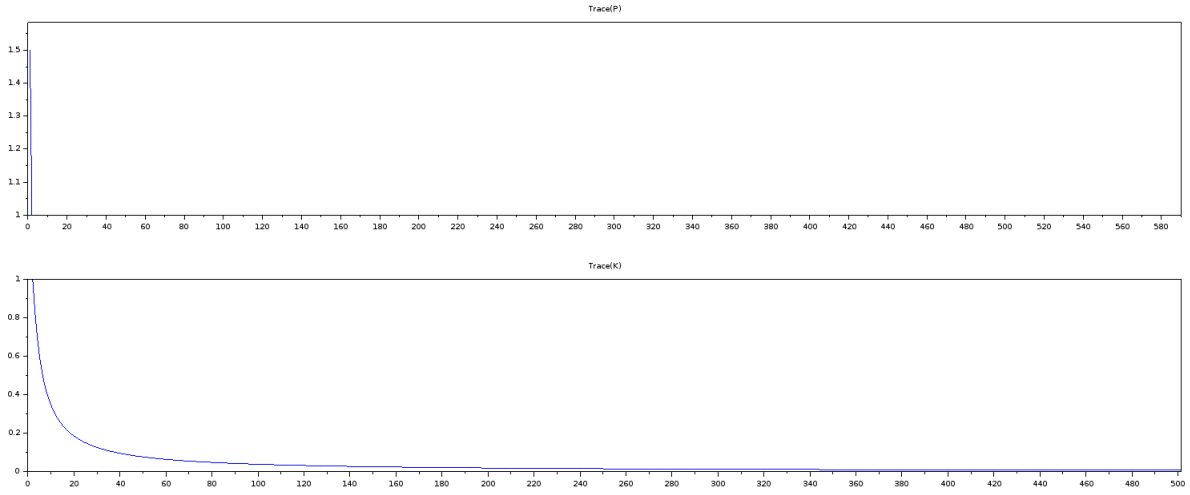


FIGURE 8.3 – Traces des matrices $P_{k|k}$ et K_k

La trace de P converge rapidement en quelques itérations. La trace de K converge bien plus lentement, à la même vitesse que le vecteur d'état, ce qui en fait le coupable idéal du régime transitoire initial. Cette matrice K n'étant pas directement initialisable nous n'avons apparemment pas d'autre choix que d'attendre la fin de ce régime transitoire, soit une centaine d'itérations. Ceci sera important lors des mesures sur le système réel.

La figure 8.2 montre que l'erreur statique tend à converger vers zéro, ce qui est une bonne nouvelle. En régime permanent le filtre rejette bien le bruit blanc court terme introduit dans le modèle. Le calcul de l'écart-type de cette réponse par rapport au modèle (sans bruit blanc et en excluant les 100 premières itérations) donne une valeur autour de 1.10^{-4} soit un facteur d'atténuation de 30 par rapport à l'écart-type du bruit introduit dans le modèle (3.10^{-3}).

On peut déjà observer l'influence des paramètres (coefficient C_k et initialisation des matrices Q_k, R_k) sur l'atténuation du bruit blanc. Pour simplifier les choses on fixera la matrice de bruit de mesure (R_k) à sa valeur actuelle et on fera varier le coefficient C_k et la matrice de bruit de process Q_k [11]. Les résultats sont présentés sur la figure 8.4, j'ai noté q_k l'écart-type du bruit de process (élevé au carré dans la matrice Q_k).

11. Cette simplification me semble légitime puisque le bruit de mesure dépend de la qualité du capteur, supposée

qk = 1,1e-05

Ck	1	0,8	0,5	1,5
Écart-type	1e-4	1e-4	1e-4	1,2e-4

Ck = 1

qk	5e-6	5e-5	1e-4
Écart-type	1e-4	1,6e-4	2,2e-4

qk = 1e-4

Ck	0,8	0,5	0,1
Écart-type	2e-4	1,7e-4	1,3e-4

FIGURE 8.4 – Evolution de l'écart-type en fonction des coefficients C_k et Q_k

On peut voir qu'une augmentation de la valeur de q_k augmente l'écart-type avec le modèle. En effet cette valeur représente une estimation de l'écart-type du bruit issus des sources et de la référence de fréquence, plus cette valeur est élevée et plus le filtre Kalman fera confiance aux mesures plutôt qu'au modèle d'évolution définit dans la matrice de transition phiK.
On remarque également que la modification du gain optimal apporte une amélioration sur l'atténuation du bruit blanc uniquement si la valeur de Q_k est définie trop élevée.

On introduit ensuite des perturbations dans le modèle et on observe la réponse du filtre en fonction de son paramétrage. L'objectif de notre asservissement est d'atténuer les fluctuations court terme mais de suivre au mieux les tendances long terme (random-walk et dérives) afin de pouvoir les compenser.

Saut ponctuel de fréquence (ou saut de phase)

Un saut de fréquence peut être dû à une perturbation externe (choc) ou interne au système (saut de phase de la source ou mesure aberrante du capteur). Ce type de perturbation peut facilement être anticipé en amont du filtre si il est suffisamment rare. Par exemple, si sa probabilité d'occurrence est inférieure à 1% on peut mesurer la valeur moyenne et l'écart-type sur 100 secondes et considérer uniquement les mesures compris dans 3 ou 5 écart-types par rapport à la moyenne.
Pour cette raison on laisse de côté l'effet du paramétrage sur ce type de perturbation. Les figures 8.5 et 8.6 montrent respectivement la réponse du filtre pour un saut de 0.03 Hz (10 fois l'écart-type du bruit blanc déjà ajouté au modèle) et 0.3 Hz (100 écart-types).

On peut voir qu'un saut de 10 écart-types ne perturbe quasiment pas le filtre qui reprend son niveau de bruit précédent dès l'itération suivante. Par contre un saut de 100 écarts-types impacte la réponse du filtre sur la durée qui mettra plusieurs centaines d'itérations à reprendre sa valeur précédente, ce qui induirait des effets nuisibles dans notre système réel.

Ceci prouve l'importance de donner des mesures "propres" au filtre de Kalman. Ceci peut être

constante

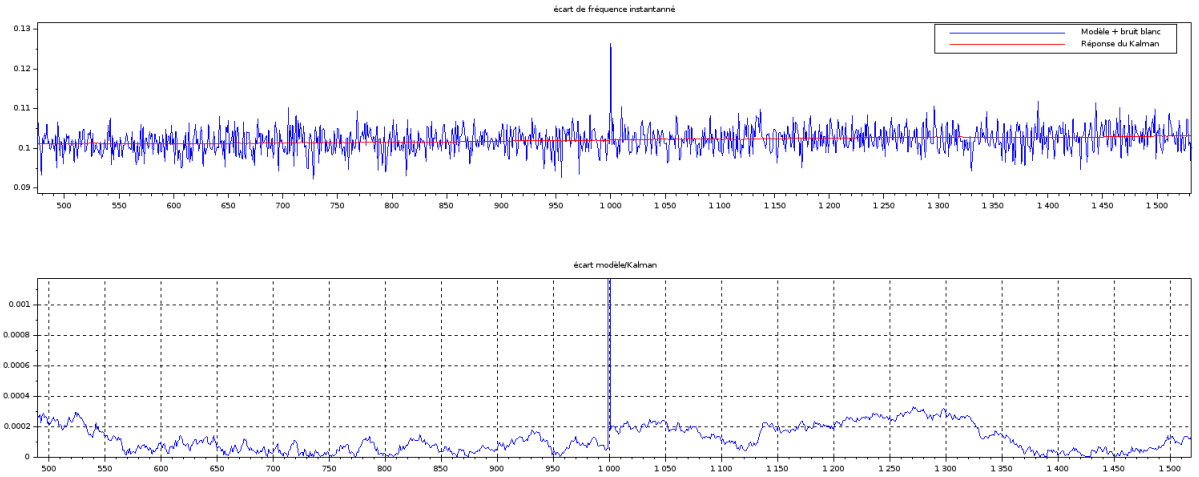


FIGURE 8.5 – Réponse du filtre à un saut de fréquence de 0.03 Hz

fait par une mise en place de "garde-fou" permettant d'ignorer des mesures jugées aberrantes. Mais cette pratique est peu rigoureuse et ne fait que masquer le problème, l'idéal étant d'avoir un outil de mesure fiable (le code FPGA de l'intervallomètre doit être robuste).

Échelon de fréquence

Ce type de perturbation est souvent du à un choc ou une altération volontaire (réglage de fréquence) d'un des oscillateurs. De la même façon qu'un saut ponctuel il est la manifestation d'un dysfonctionnement d'un oscillateur ou de l'intervallomètre.
La réponse du filtre désirée pour ce type de perturbation sera une convergence lente mais sans erreur statique et un retour à l'état précédent après l'échelon.

L'échelon de fréquence se situe entre les itérations 2000 et 6000, après convergence initiale du filtre. Les 3 figures suivantes montrent la réponse du filtre pour un échelon de 0.003, 0.03 et 0.3 Hertz (soit 1, 10 et 100 fois l'écart-type du bruit blanc déjà introduit dans le modèle).

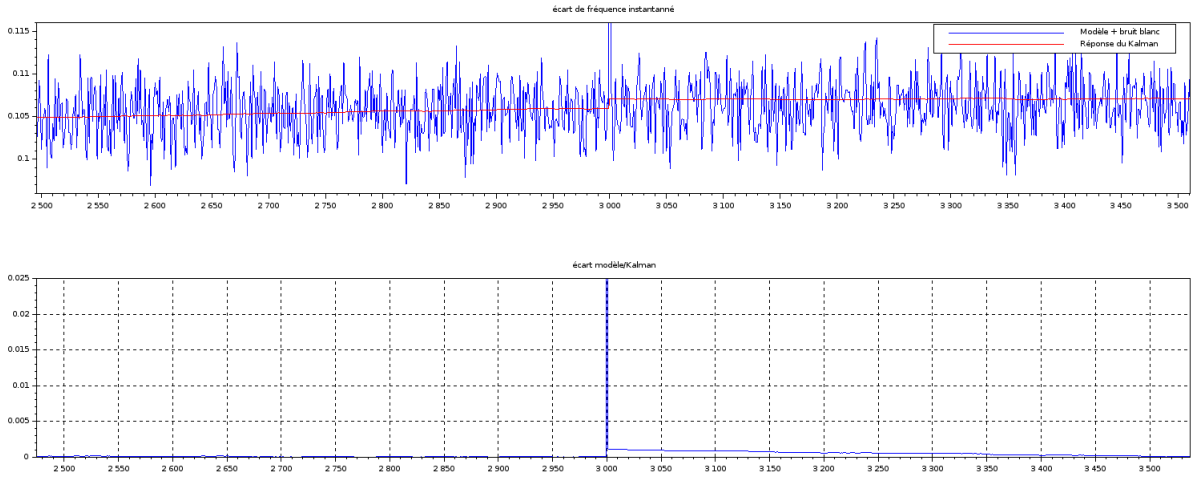


FIGURE 8.6 – Réponse du filtre à un saut de fréquence de 0.3 Hz

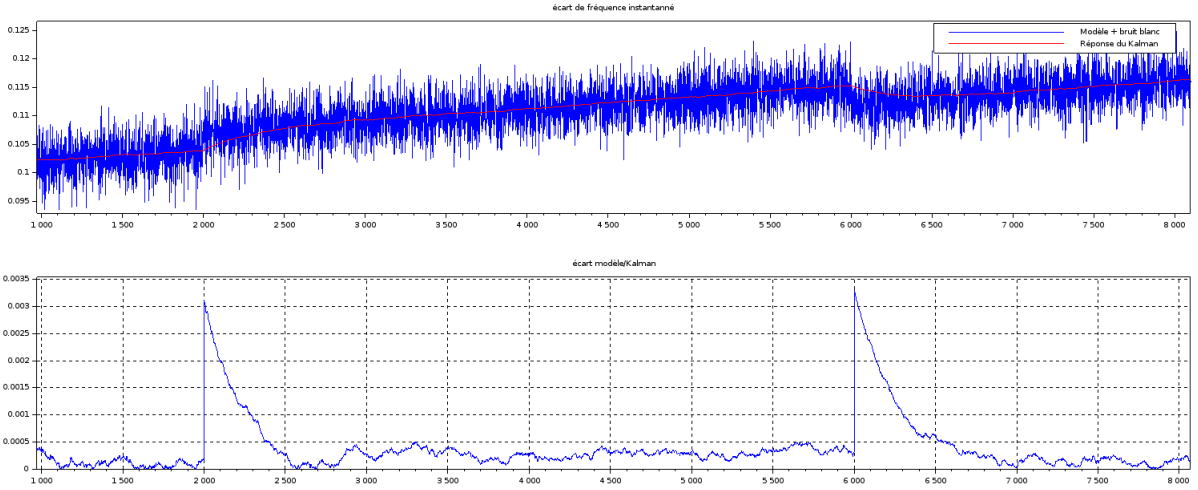


FIGURE 8.7 – Réponse du filtre à un échelon de fréquence de 0.003 Hz

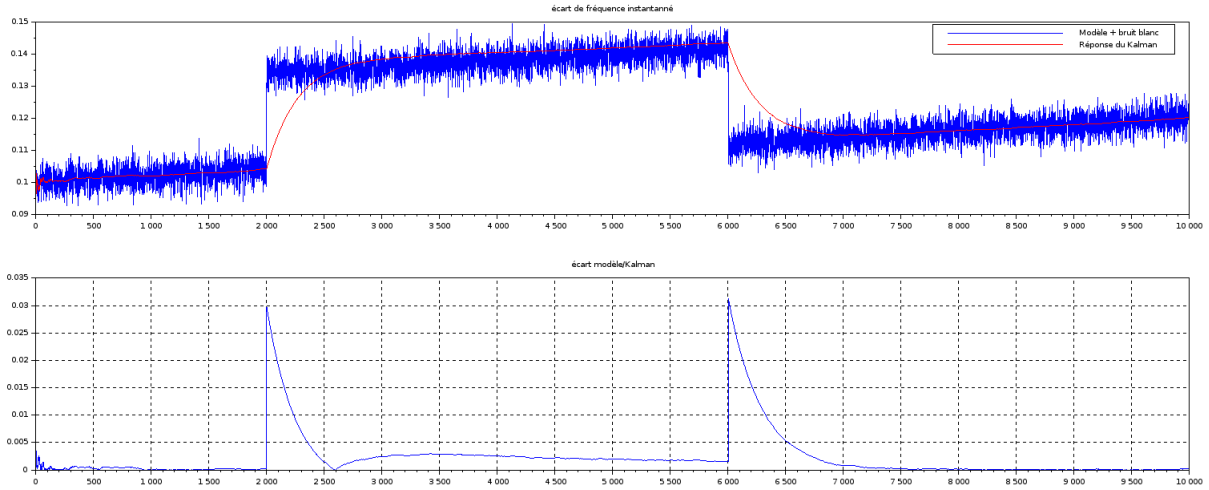


FIGURE 8.8 – Réponse du filtre à un échelon de fréquence de 0.03 Hz

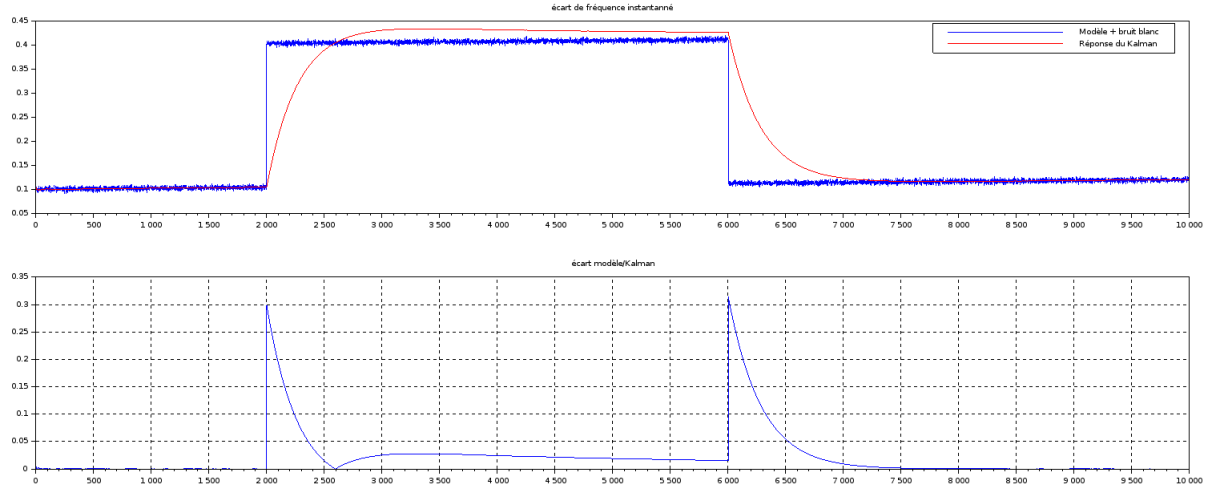


FIGURE 8.9 – Réponse du filtre à un échelon de fréquence de 0.3 Hz

Pour un échelon de 0.003 Hz le filtre met environ 500 itérations avant de converger vers l'état haut, le comportement semble symétrique pour la descente de l'échelon.

Pour les amplitudes d'échelon 0.03 et 0.3 Hz le filtre converge beaucoup plus lentement vers l'état haut et plusieurs milliers d'itération ne suffisent pas à le stabiliser. Concernant le retour à l'état avant échelon, la convergence semble bien plus rapide (1000 itérations pour une amplitude de 0.03 Hz et 1500 pour 0.3 Hz).

Ces résultats et ceux précédents (saut de fréquence) laissent penser que le filtre de Kalman est assez peu robuste aux variations rapides (fronts) de grande amplitude (>10 écarts-types pour un saut de fréquence et >1 écart-type pour un échelon).

Variations sinusoïdales lentes de fréquence

Ces variations lentes et continues de fréquence sont censées représenter les effets moyen terme (random-walk) présents dans de nombreux oscillateurs (quartz en particulier mais aussi rubidiums et même dans le maser à hydrogène). Notre filtre doit "coller" au mieux à ces variations (induites notamment par l'oscillateur local à quartz) afin de pouvoir les compenser le plus efficacement possible (erreur dynamique la plus faible possible).

Les 5 figures suivantes montrent la réponse du filtre de Kalman (et l'écart par rapport au modèle sans bruit blanc) pour différentes valeurs de q_k, r_k est fixé à $3,125.10^{-3}$ et $C_k = 1$.

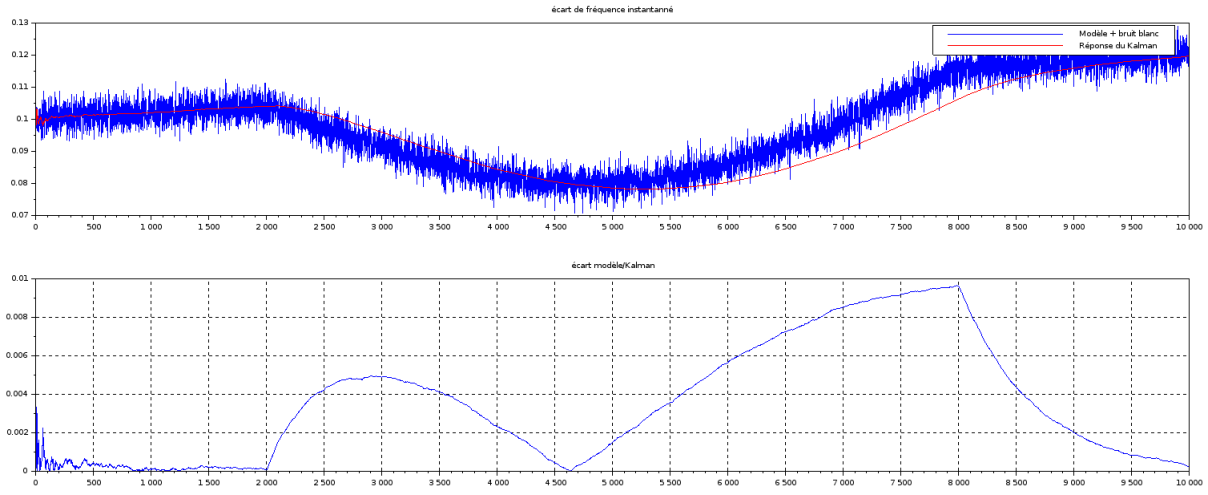


FIGURE 8.10 – Réponse du filtre à une variation lente $q_k = 5.10^{-6}$

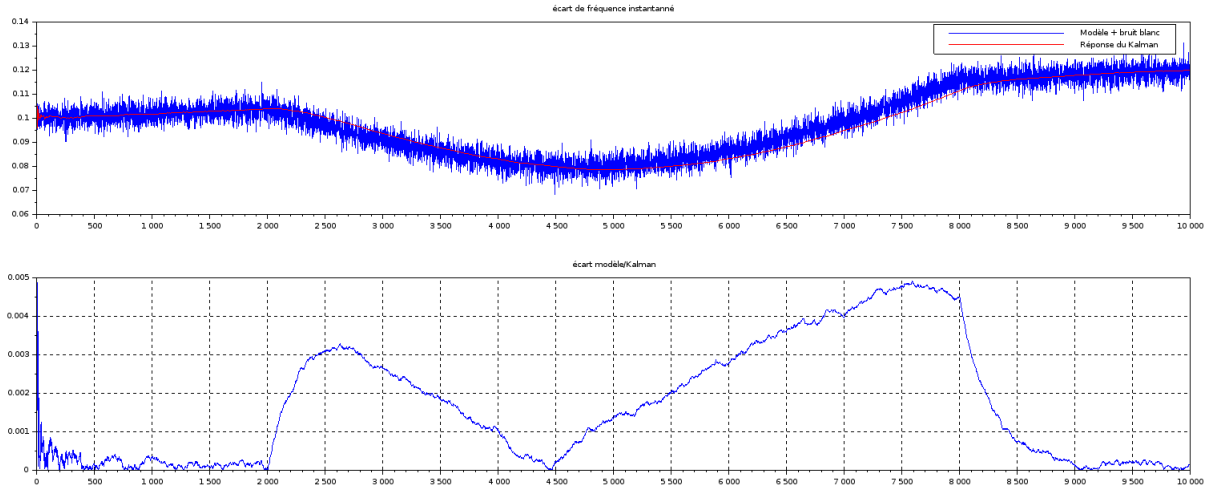


FIGURE 8.11 – Réponse du filtre à une variation lente $q_k = 1,1.10^{-5}$

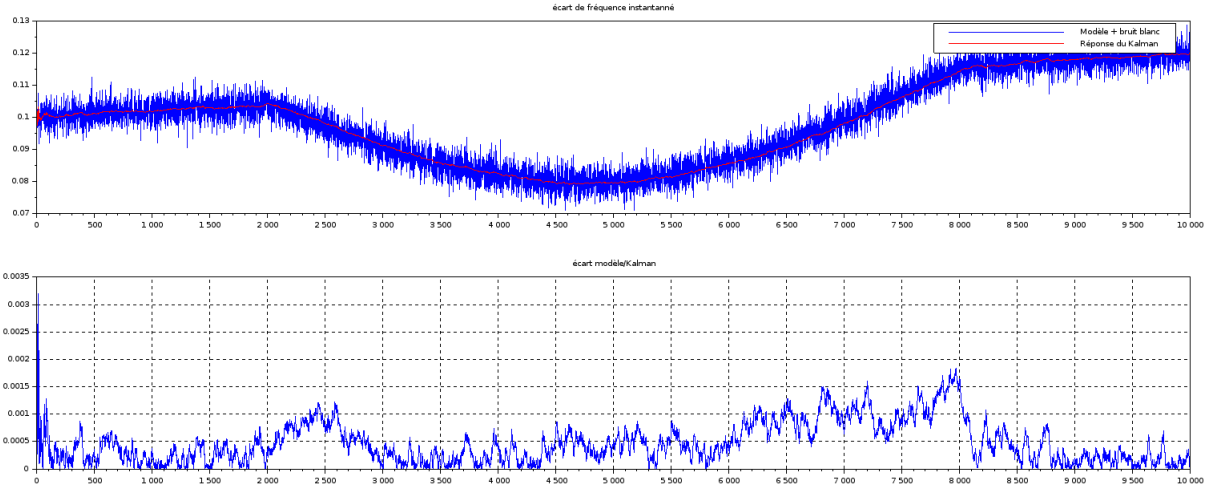


FIGURE 8.12 – Réponse du filtre à une variation lente $q_k = 5.10^{-5}$

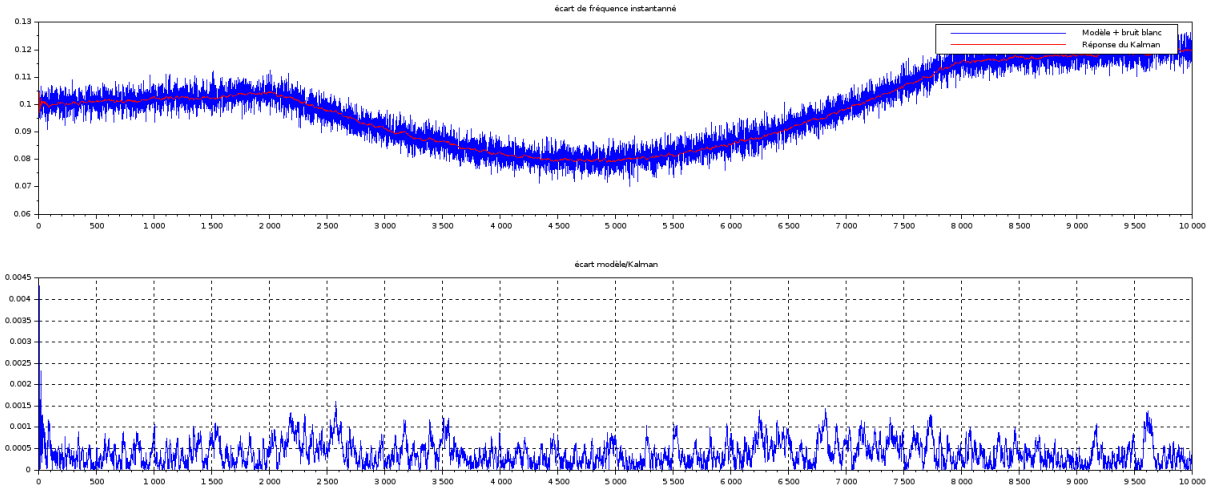


FIGURE 8.13 – Réponse du filtre à une variation lente $q_k = 1.10^{-4}$

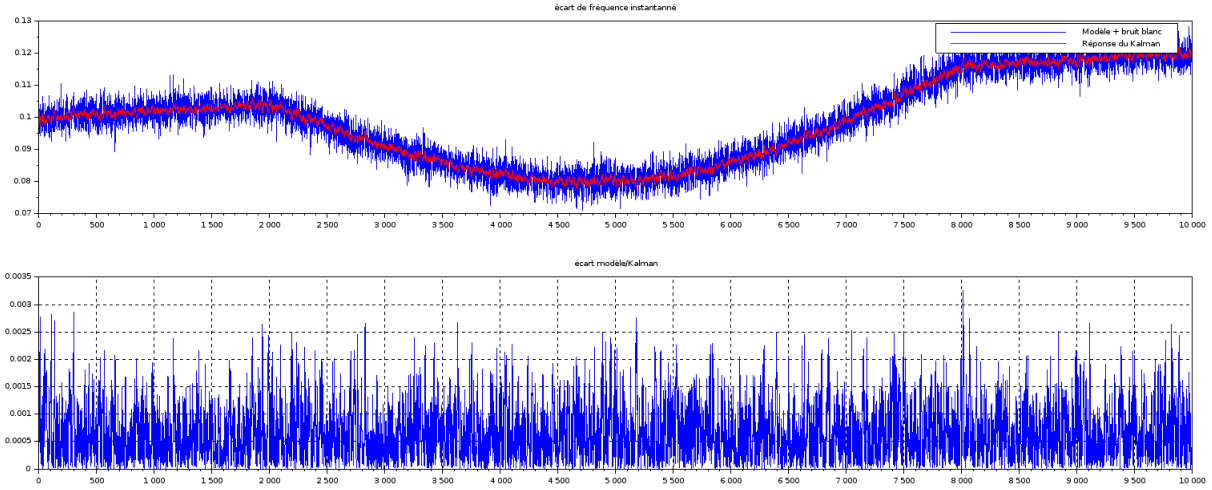


FIGURE 8.14 – Réponse du filtre à une variation lente $q_k = 5.10^{-4}$

On voit clairement qu'une augmentation de q_k réduit le temps de réponse du filtre qui "collera" mieux aux variations lentes de fréquence. D'un autre coté ceci à pour effet de réduire l'atténuation du bruit blanc court-terme. Le tableau et la courbe suivante résument les valeurs d'écart-type entre la réponse du Kalman et le modèle sans bruit blanc pour chaque valeur de q_k.

qk	5,00E-06	1,10E-05	5,00E-05	1,00E-04	5,00E-04
Ecart-type	3,00E-03	1,50E-03	3,60E-04	2,70E-04	5,00E-04

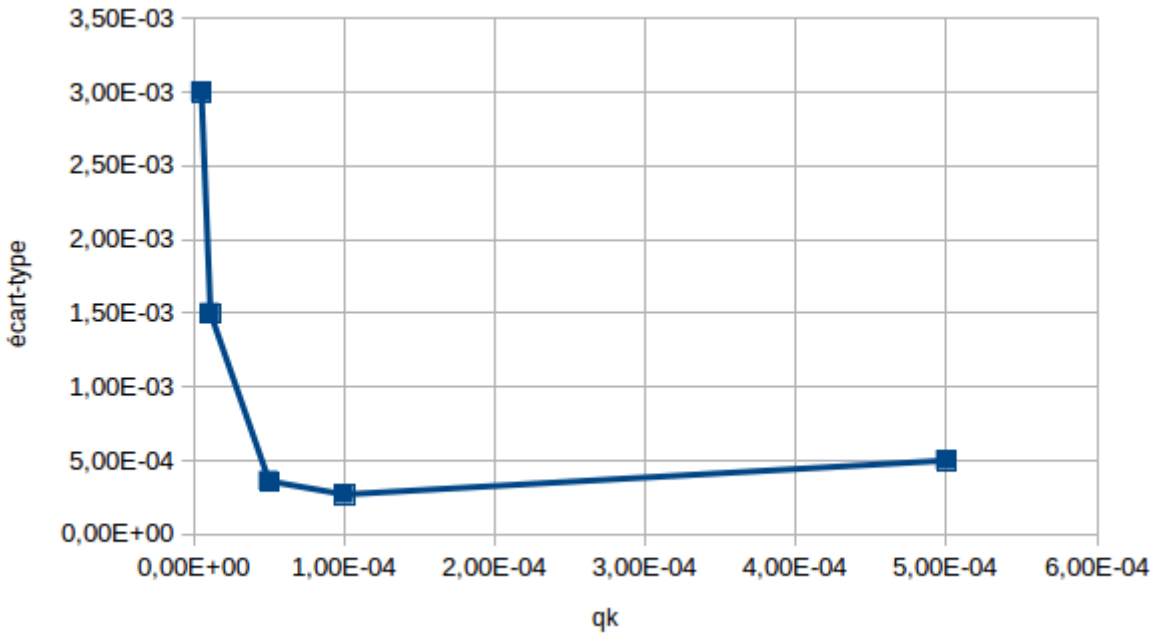


FIGURE 8.15 – Réponse du filtre à une variation lente en fonction de q_k

La valeur optimale [12] de q_k se trouve autour de 1.10^{-4} dans notre cas. Une valeur trop faible de q_k empêchera le filtre de suivre les variations lentes (accumulation d'erreur dynamique) et une valeur trop grande induira une instabilité à court-terme (le filtre suivra le bruit blanc).

Conclusions

Le principal paramétrage du filtre de Kalman semble se situer sur le rapport des matrices Q_k et R_k. La matrice R_k étant liée à notre outil de mesure (intervallomètre) seule la modification de la matrice Q_k semble pertinente. Quand au gain de Kalman, l'utilisation d'un gain réduit semble être une pratique marginale qui ne trouve d'ailleurs aucun écho dans la littérature.

12. Optimale entre temps de réponse et atténuation du bruit blanc court terme, et pour la perturbation considérée

Quatrième partie

Résultats

Attention !

Les courbes de stabilité présentées dans les chapitre 9 et 10 ont toutes des barres d'incertitudes inversées. Cette erreur provient probablement du script que j'ai utilisé pour importer les données de l'outils de mesure de stabilité (TSC5110A de TimingSolutions). A l'heure où je rédige ces lignes je n'ai plus accès à ce script ni aux outils m'ayant permis de générer ces courbes et je ne suis donc pas en mesure de corriger ces erreurs. Je vous prie de bien vouloir m'en excuser.

Chapitre 9

Performances hardware

D'un point de vue matériel les 2 prototypes finaux se distinguent principalement par le type de VCO utilisé. Dans une moindre mesure les puces zynq et la connectique diffèrent aussi.

La figure 9.1 résume les caractéristiques des 2 prototypes en reprenant certains éléments décrits dans la partie "Hardware". A noter que la plage de commande de la manip 1 (ZedBoard) initialement située entre 4,5 et 6,5 Volts à été rétablie à 0-10 Volts. Le palier de Flicker théorique correspondant à été recalculé.

		Manip 1	Manip 2
Kit de base		ZedBoard	MicroZed
Zynq		7Z020	7Z010
Connectique FPGA		FMC 68 pins	100 pins
Liaison SPI		Cable RJ45	imprimée
VCO		BVA externe	OCXO intégré
	Fréquence	10 MHz	10 MHz
	Stab (Adev)		
	1 s	1e-12	2e-12
	10 s	2e-13	3e-12
	100 s	4e-13	5e-12
	Connectique	DB-9	imprimée
	Plage de commande	0-10V	0-10V
	Palier de Flicker théorique	2,25e-13	1,2e-12
Cout global (hors VCO)		1000 €	600 €

FIGURE 9.1 – Composition des 2 prototypes

La partie processeur sur laquelle l'application est exécutée est la même. Les composants ZCD et le DAC sont identiques ainsi que le code FPGA implanté.

Concernant la partie logicielle,seule la sensibilité du VCO est à adapter. Les écarts temporels peuvent être récupérés et traités de la même façon.

Sauf mention contraire, l'algorithme utilisé est un filtre de Kalman avec le paramétrage suivant :

- Un vecteur d'état à une seule composante (écart de fréquence instantané). Ce qui simplifie énormément l'algorithme, les matrices deviennent des scalaires.
- Une durée d'itération (entre 2 mesures d'écarts de fréquence) est de 1 seconde.
- Un gain de Kalman optimal (coeff appliqué au gain de 1)
- Une efficacité de commande de 1
- La matrice de bruit de mesure R_k initialisée et fixée à 1.10^{-10}

Le paramétrage du filtre s'effectuera donc en faisant varier la matrice de bruit de process Q_k.

9.1 Seuil court et Moyen terme

C'est un asservissement sur un seul signal provenant du maser transmis depuis les locaux de Femto-ST, en utilisant le maser local comme référence on peut mesurer des stabilités de l'ordre de $3.10^{-13}\tau^{-\frac{1}{2}}$ entre 1 et 100 secondes jusqu'à atteindre un palier à 3.10^{-15} pour des durées de quelques jours.

Différents valeurs on été testées pour la matrice Q_k et la valeur donnant la meilleur stabilité à été retenue. Pour les 2 manip cette valeur optimale est identique et vaut 1.10^{-11}. Les figures 9.2 et 9.3 représente les relevés de stabilité (Adev) respectifs des prototypes [ZedBoard+BVA10] et [MicroZed+OCXO_AR10].

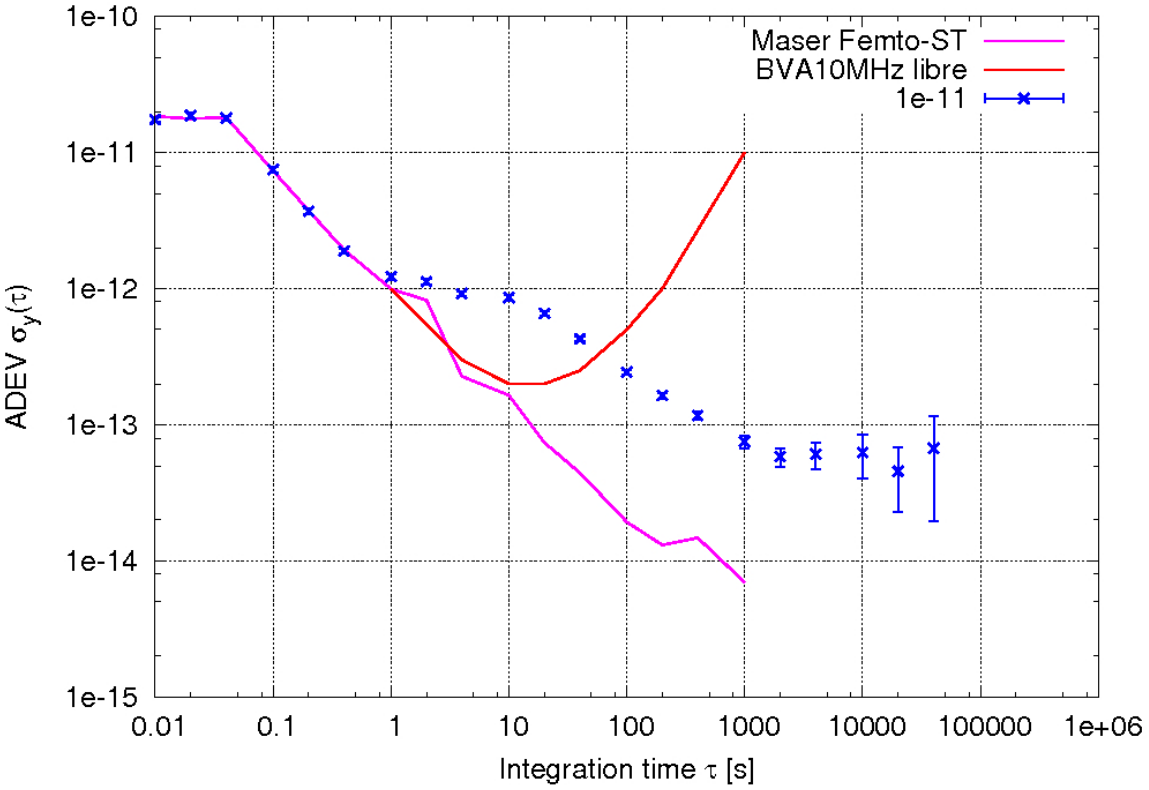


FIGURE 9.2 – ZedBoard/BVA10 asservi sur le maser

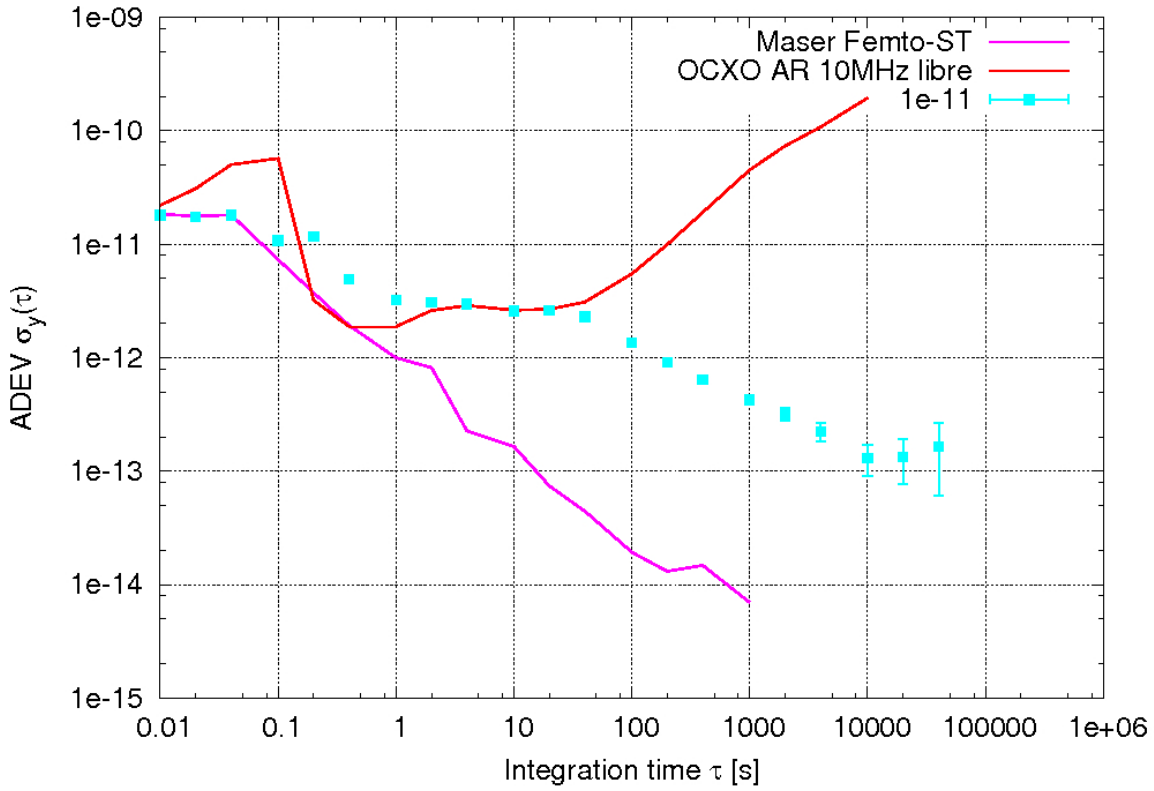


FIGURE 9.3 – MicroZed/OCXO10 asservi sur le maser

Comme attendu les 2 prototypes présentent des courbes de stabilité en $\tau^{-1/2}$ jusqu'à atteindre leur palier de Flicker. Cependant les paliers mesurés sont inférieurs aux paliers théoriques calculés précédemment.

Le palier du prototype à base du BVA10 se situe autour de 4.10^{-14} et il est atteint pour une durée d'intégration de 1000 secondes. Le palier théorique étant de $2,25.10^{-13}$.

Concernant le prototype à base de l'OCXO commercial d'AR son palier est atteint pour une durée de 10000 secondes et se situe autour de 2.10^{-13} pour un palier théorique de $1,2.10^{-12}$.

9.2 Long terme

Les mesures de stabilité pour des durées supérieures à la semaine sont assez fastidieuses [1] et serait utiles uniquement si les sources utilisées sont suffisamment stable sur ces durées (césium et maser).

Dans le cas de l'utilisation de rubidiums commerciaux ou de quartz (même de très bonne qualité) les phénomènes de marche aléatoire et de dérive sont visibles sur des durées plus courtes, de l'ordre de la journée. Les paliers de stabilité atteints étant inférieurs aux stabilités de rubidiums commerciaux sur les durées d'intégration considérées les 2 prototypes sont adaptés pour des sources de ce type.

1. L'estimation de la stabilité sur une durée τ commence à être fiable à partir d'une durée de mesure de 4τ, durée pendant laquelle le système et les outils de mesures sont totalement indisponibles

Chapitre 10

Comparaisons software

La stabilité du signal de sortie du VCO est comparée au signal du maser de l'Observatoire [1] permettant ainsi une mesure pertinente a court et moyen terme mais sujette aux dérives du maser sur des durée supérieures au mois (durée de mesure qui ne sera jamais atteinte).

10.1 Prototype ZedBoard

10.1.1 Simple source

Ici l'objectif est d'observer la réponse (en terme de stabilité) du système en fonction du paramétrage de la matrice Q_k (dont la valeur est notée en légende de la figure 10.1). Les autres paramètres sont fixés :

— La matrice de bruit de mesure R_k initialisée et fixée à 1.10^{-10}
— Un gain de Kalman optimal (coeff appliqué au gain de 1)
— Une efficacité de commande de 1

1. Sauf mention contraire

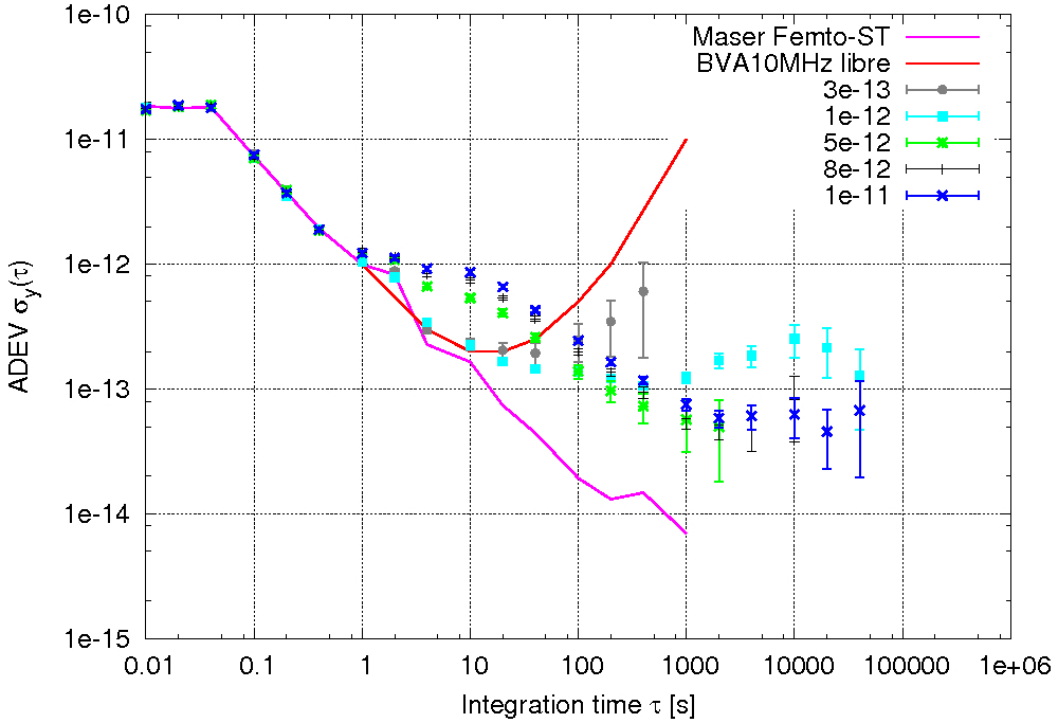


FIGURE 10.1 – Stabilité du prototype ZedBoard/BVA10 en fonction de la valeur de Q_k

Lorsque la valeur de Q_k est faible (3e-13 et 1e-12) la stabilité court-terme (< 20 secondes) est bonne mais les dérives et marche aléatoires ne sont pas suffisamment corrigées.

Lorsqu'on utilise des valeurs supérieures (> 5e-12) on dégrade la stabilité court-terme mais le palier de bruit blanc de fréquence (pente en $\tau^{-1/2}$) se prolonge sur des durées plus longues.

10.1.2 Sources hétérogènes

Dans un premier temps, le paramétrage du filtre de Kalman s'est effectué en faisant varier le coefficient de gain C_k appliqué au gain optimal de Kalman et l'efficacité de correction appliquée. Les matrices Q_k et R_k sont fixées.

La figure 10.2 montre les résultats obtenus pour un asservissement du BVA10 sur le BVA5 :

J'ai ensuite intégré une correction périodique de la fréquence par rapport au césium. Dans ce cas c'est un moyennage sur 1000 secondes de la fréquence du césium qui donne la correction à superposer à la correction faite par rapport au BVA5. Les résultats sont présentés sur la figure 10.3.

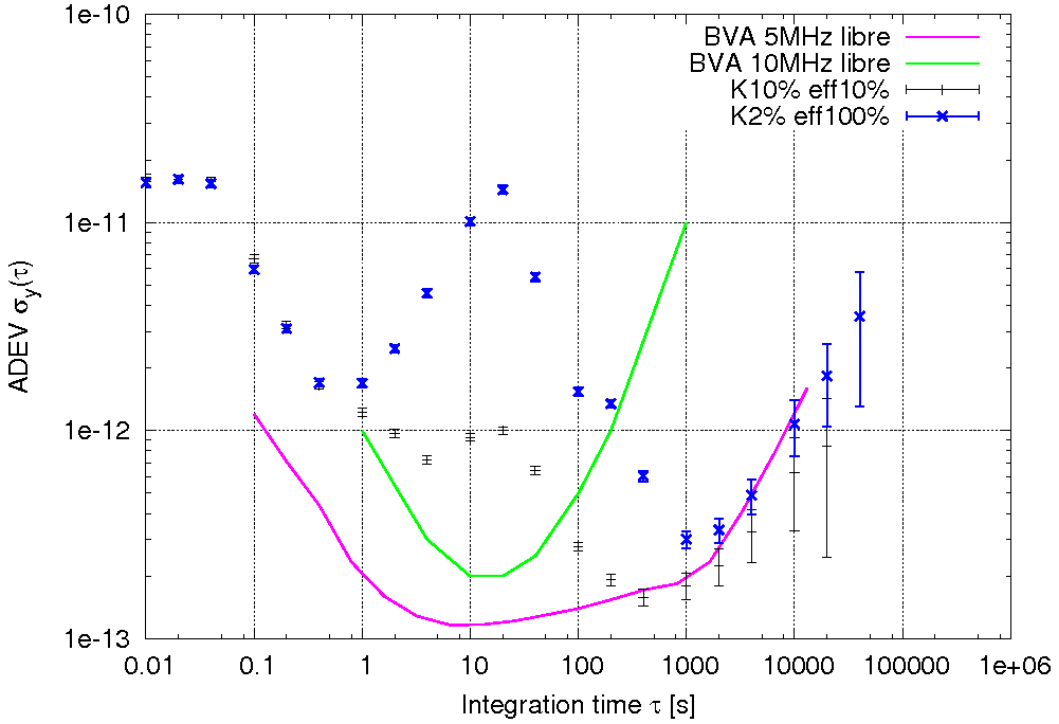


FIGURE 10.2 – BVA10 asservi sur le BVA5

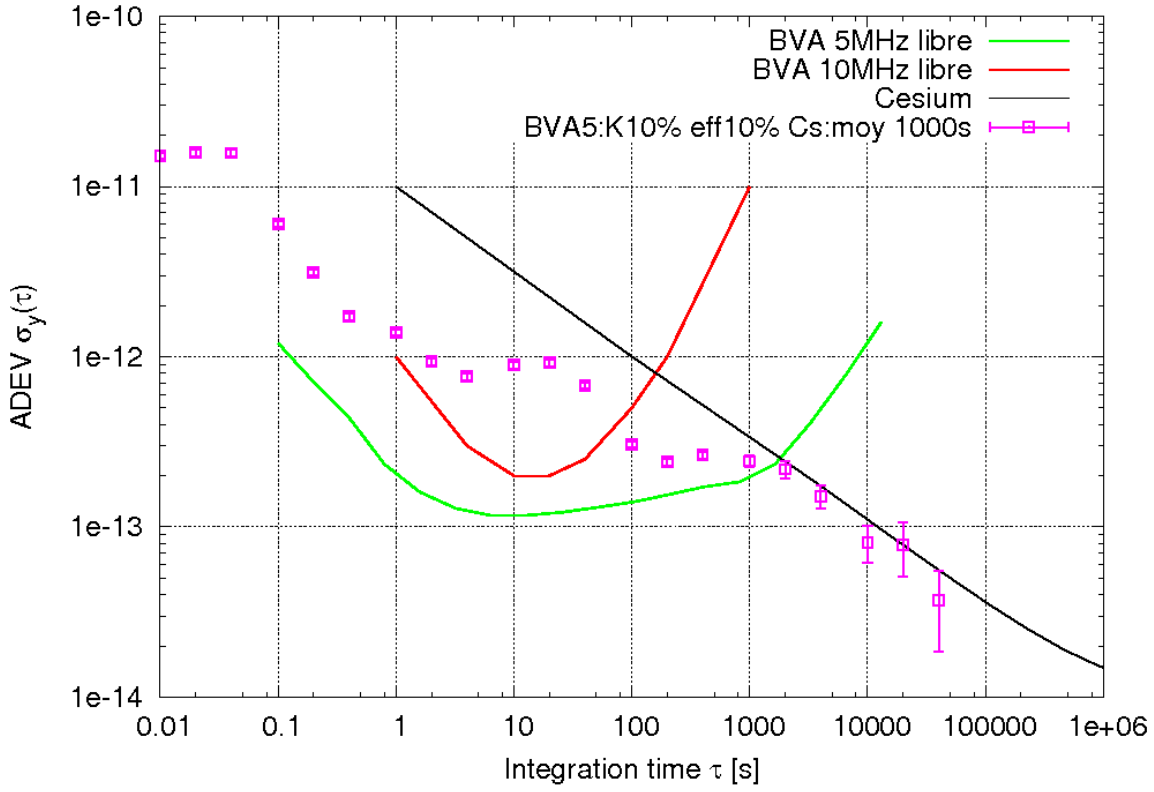


FIGURE 10.3 – BVA10 asservi sur le BVA5 et le césium

10.1.3 Sources homogènes

Ces mesures ont été effectuées dans les locaux de l'entreprise TimeLink. N'ayant pas d'oscillateurs avec une stabilité équivalente au maser j'ai dû utiliser le césium comme référence des mesures. Ce césium ayant un palier de bruit blanc de fréquence de $2.10^{-11}\tau - 1/2$ il est impossible de mesurer des stabilités inférieures. Le césium représenté sur les figures 10.4 et 10.5 est celui de l'observatoire, légèrement meilleur que celui de l'entreprise, d'où le léger décalage observé entre le palier des mesures et celui du césium.

Un filtre de Kalman est appliqué sur chaque source et la synthèse des données est effectuée par moyennage simple.

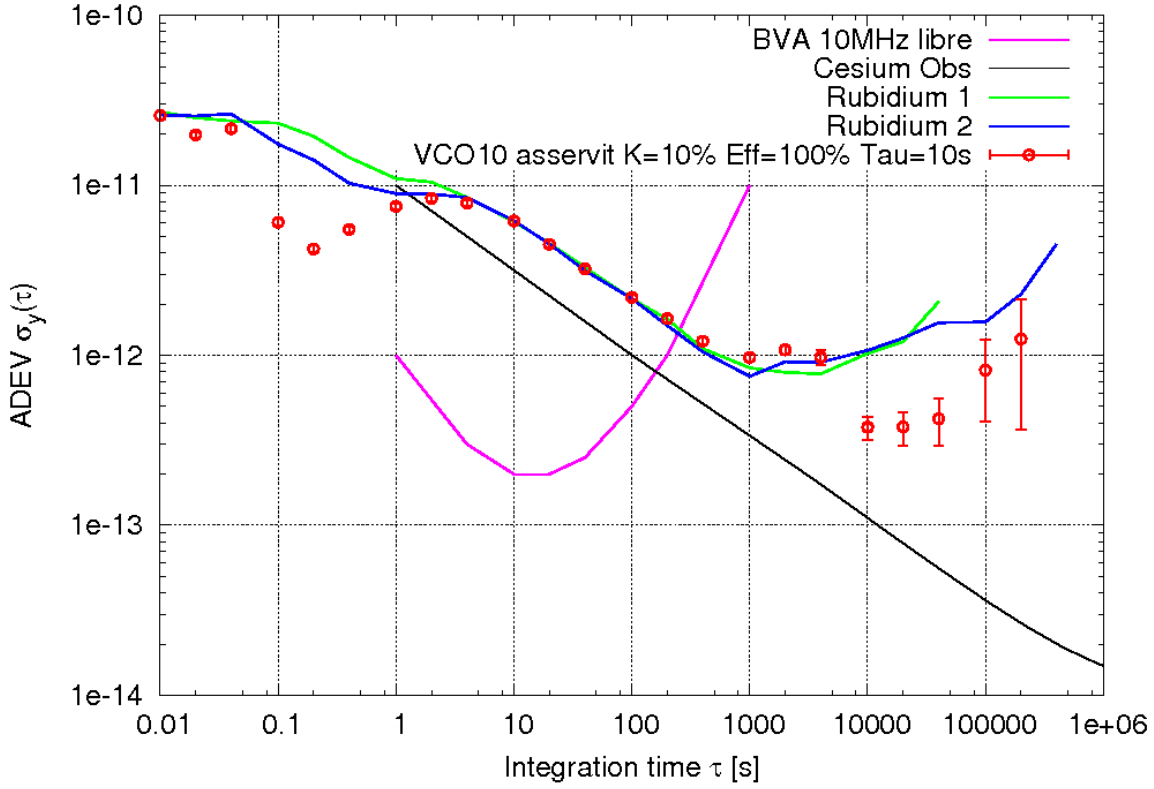


FIGURE 10.4 – BVA10 asservi sur 2 rubidiums

On peut voir sur la figure 10.4 que le système est meilleur que chacune des 2 sources utilisées. Ceci pourrait être dû au fait que les dérives des 2 sources se compensent (pente de dérive de fréquence quasi-opposées). Ceci validerait le fait qu'une intégration des coefficient de dérive pourrait grandement améliorer la stabilité long terme du système.

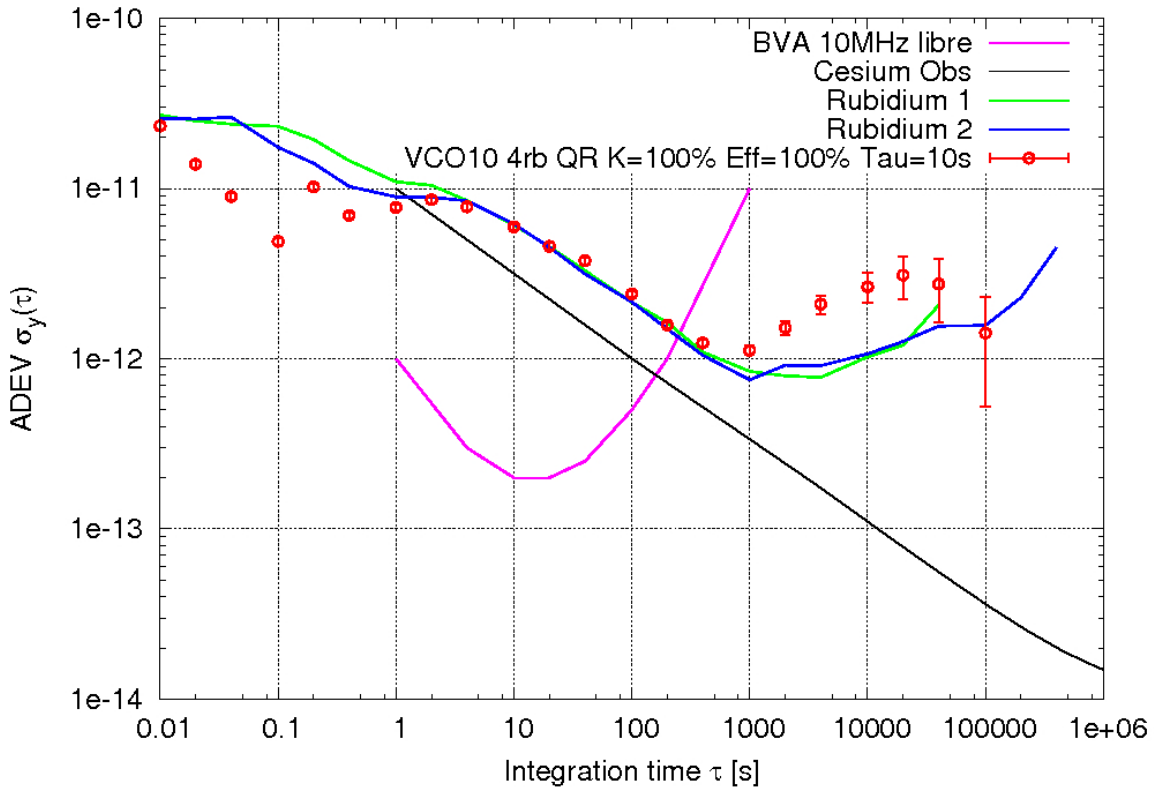


FIGURE 10.5 – BVA10 asservi sur 4 rubidiums

Ici le système semble amplifier les phénomènes de marche aléatoire, cependant la tendance à 100 000 secondes semble montrer une amélioration long terme du système.

10.2 Prototype MicroZed

10.2.1 Mesures préliminaires

Des premiers test on été réalisés sur une carte existante à base de MicroZed possédant un VCO intégré (Abracon 100 MHz). Voici les résultats pour un asservissement sur la fréquence du césium de l'Observatoire.

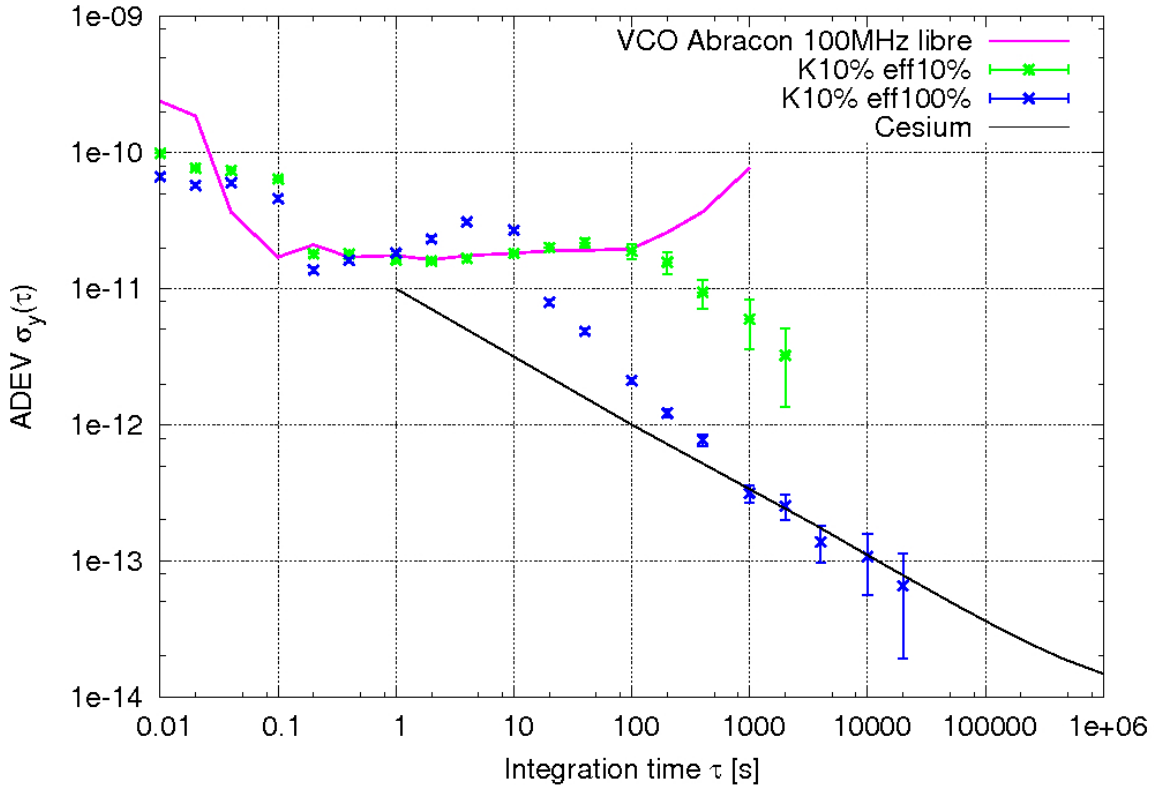


FIGURE 10.6 – VCO Abracon asservi sur la fréquence du césium

Ce VCO est assez peu stable passé 10 secondes, ce qui nécessite une efficacité de commande suffisamment élevée afin de corriger ses variation efficacement. Ce qui revient à diminuer notre temps de réponse au détriment de la stabilité court terme (entre 1 et 10 secondes).

On peut remarquer que la stabilité du VCO commercial d'Abracon suit assez fidèlement la courbe de stabilité du césium même pour des valeurs proche de 10^{-13} ce qui à motivé (et validé le principe)de l'élaboration d'un prototype à base de MicroZed.

10.2.2 Simple source

Ici l'objectif est d'observer la réponse (en terme de stabilité) du système en fonction du paramétrage de la matrice Q_k (dont la valeur est notée en légende de la figure 10.7). Les autres paramètres sont fixés :

— La matrice de bruit de mesure R_k initialisée et fixée à 1.10^{-10}
— Un gain de Kalman optimal (coeff appliqué au gain de 1)
— Une efficacité de commande de 1

La source sur laquelle l'OCXO est asservie est le maser de Femto-ST et sa stabilité est mesurée grâce au maser de l'Observatoire.

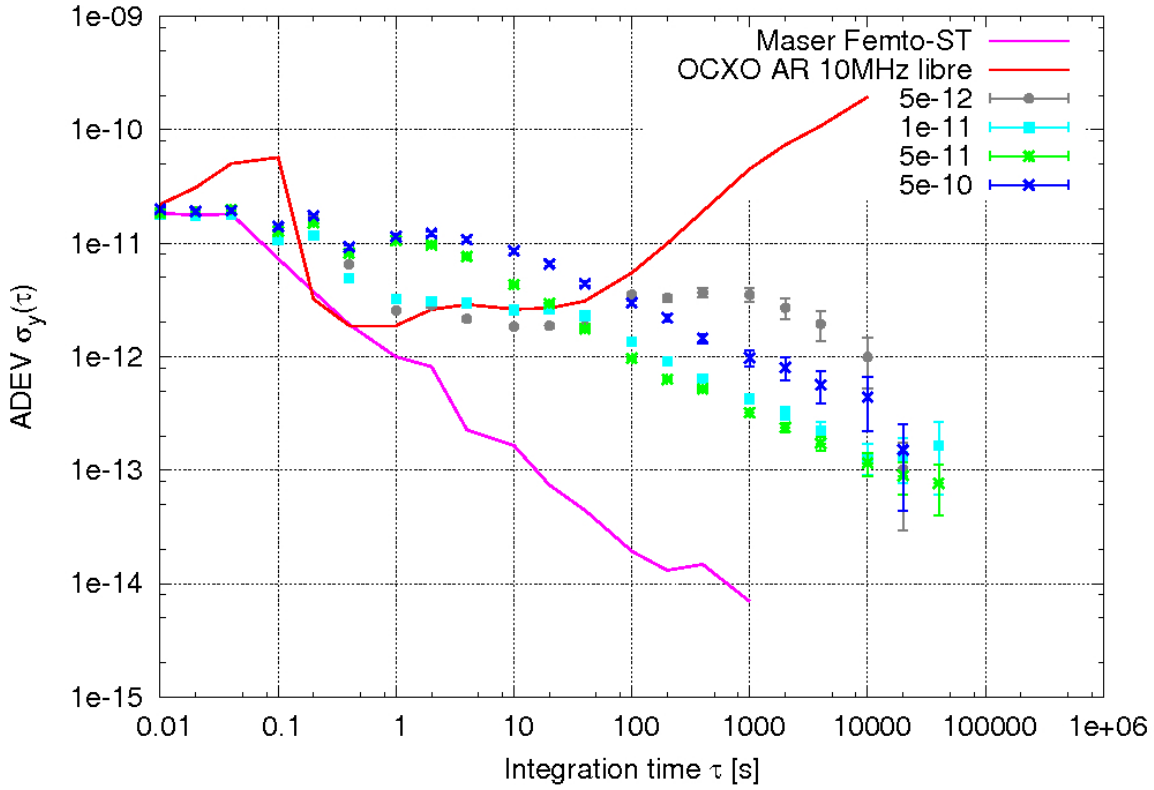


FIGURE 10.7 – MicroZed/OCXO AR10 asservi sur le maser de Femto-ST

Comme pour le prototype ZedBoard/BVA10 un gain de stabilité long terme est obtenu en sacrifiant la stabilité court-terme.

10.2.3 Sources hétérogènes

Les 2 sources sont le BVA 5MHz et le césium de l'Observatoire.

Un filtre de Kalman est appliqué sur le signal du BVA5 et un moyennage de la fréquence du césium sur une durée correspondante au croisement des domaines de stabilité des 2 sources (1000 secondes) est réalisé pour corriger le VCO avec cette périodicité.

Comme pour l'asservissement simple source seule la matrice Q_k à été modifiée entre chaque mesure.

Concernant la correction long-terme (césium) le coefficient d'efficacité utilisé est de 1. Le résultat est présenté figure 10.8

10.3 Conclusions

Les 2 prototypes développés (ZedBoard et MicroZed) ont une base hardware solide permettant d'envisager d'y implanter assez facilement d'autres codes FPGA ainsi que d'autres algorithmes d'asservissement. Les connaissances nécessaires sont regroupées dans ce document.

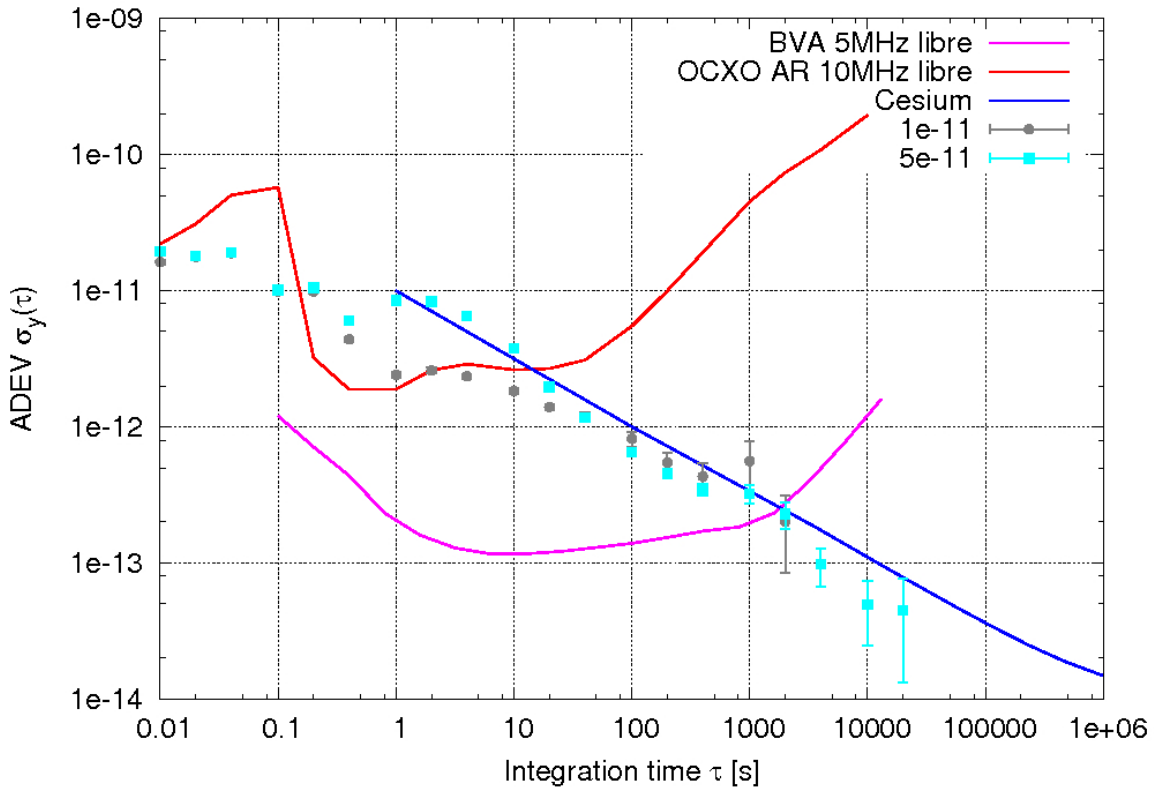


FIGURE 10.8 – MicroZed/OCXO AR10 asservi sur le BVA5 et le césium

Les tests de combinaison de sources similaires ont été menés sur des horloges à rubidium, principalement sujettes à du random-walk sur des périodes d'un jour à 3 semaines. Par nature le random-walk est aléatoire donc difficile à prédire, d'où la difficulté d'utiliser un algorithme prédictif de façon pertinente.

La combinaison d'horloges complémentaires à donné des résultats satisfaisant (figures 10.1, 10.3, 10.6 et 10.8). Les domaines de stabilité y sont combinés relativement correctement.

Cinquième partie

Conclusions et perspectives

Chapitre 11

Conclusions

Le retard accumulé sur la partie hardware (notamment le code FPGA) et la mise en forme software ne m'ont pas permis d'investiger et de tester d'autres algorithmes d'asservissement que le correcteur PID et le filtre de Kalman.

Cependant les prototypes développés sont fonctionnels et suffisamment performants pour être réutilisés tels quels, le code FPGA et le code C peuvent être repris de façon indépendante.

11.1 Hardware

Deux types de prototypes fonctionnels ont été développés :

— Un à base de ZedBoard destiné à asservir le BVA (2.10^{-13} entre 1s et 1000s)
Ce prototype est adapté pour l'asservissement sur des étalons primaires (césium) ou secondaires (maser) très stables. Sa plus grande capacité de logique programmable peut en faire un bon candidat à l'investigation d'autres codes FPGA.
— Deux autres à base de Microzed/Carte-mère avec le VCO (5.10^{-12} entre 1s et 100s)
Ces prototypes sont plus adaptés pour l'asservissement sur des étalons secondaires (rubidiums) ou des OCXO commerciaux. Une version à été laissée en tests long-terme dans l'entreprise.

Ces 3 prototypes utilisent des ZCD ayant une bande passante large (1Hz - 300MHz) permettant de développer d'autres méthodes de mesure et d'asservissement que peuvent permettre les puces Zynq (7Z010 pour la MicroZed et 7Z020 pour la ZedBoard).

La version ZedBoard possède 8 entrées sources LVDS fonctionnelles routées sur les pins CC [1] du Zynq.

11.2 Software

11.2.1 Algorithme d'asservissement

L'état actuel du reste du système permet une datation des fronts montants de chacune des sources à une fréquence de 100Hz. Cette mesure peut être la base de tout autre algorithme.

1. Clock Compatible

Chapitre 12

Perspectives

De nombreux points peuvent être améliorés dans les systèmes que j'ai développé, aussi bien au niveau hardware que software. Ce chapitre donne des pistes d'investigation afin de poursuivre le développement des horloges composites.

12.1 Hardware

12.1.1 Composants

L'amélioration hardware la plus simple serait de réduire les plages de commande en tension des VCO, ainsi on réduirait l'ensemble des paliers de bruits de nos systèmes (palier de bruits blanc de fréquence et palier de Flicker).

Une autre amélioration serait d'intégrer directement des horloges similaires (rubidiums ou quartz) sur la carte-mère. La miniaturisation de ces élément devrait pouvoir le permettre. Aussi il est important que ces signaux d'horloges soit le plus décorélés possible afin que l'hypothèse de bruit aléatoires reste valide. La variation d'un paramètre environnemental [1] commun sera répercutée directement sur le signal de sortie.

12.1.2 FPGA

La fréquence actuelle de la mesure des écarts temporels bruts est de 100Hz (une mesure toutes les 10ms). Cette fréquence pourrait être augmentée [2] en transférant certaines parties de la mise en forme logicielle dans le domaine FPGA, Par ordre de priorité :

1. Traitement des roll-overs
2. Déduction de l'écart temporel brut x^i_{cpt}
3. Stockage des x^i_{cpt}
4. Traitement des x^i_{cpt} et estimation $\bar{x}_{cpt}$ de l'écart temporel

Le reste de l'algorithme pourrait aussi se traduire en code FPGA mais cela rendrait impossible sa re-configuration sans re-flasher la logique programmable.

1. en particulier la température, principale coupable du bruit de marche aléatoire
2. jusqu'au MHz

12.2 Software

12.2.1 Amélioration du Kalman

Intégrer la dérive linéaire de fréquence demande une caractérisation parfois longue des sources [3] mais dans certains cas (plusieurs quartz similaires) cela pourrait être relativement facile à mettre en place.

La variance de l'erreur d'estimation

La matrice $P_{k|k}$ donne une information sur l'incertitude de l'estimation de l'écart de fréquence $\Delta\nu_k$, cette information pourrait être prise en compte pour qualifier et éventuellement pondérer la contribution d'une source à l'établissement de la correction $\Delta\nu_{correction}$

12.2.2 Compensation thermique

Dans le cas où il serait possible de mesurer la température courante des sources et en connaissant leur sensibilité thermique on pourrait intégrer une compensation de variation de température.

Par exemple dans le Kalman on intégrerait cette variation dans le vecteur de correction U_k de la manière suivante :

$$U_k = \Delta\nu_{correction} + S_T\Delta T$$

avec S_T la sensibilité thermique de la source
et ΔT l'écart de température mesuré sur la durée τ_{reg}

Ceci pourrait éventuellement réduire les phénomènes de marche aléatoire de fréquence.

12.2.3 Autres algorithmes d'asservissement

Les noms des pistes d'investigation sont toujours les mêmes depuis le début du projet :

. Filtrage de Wiener

. Filtrage particulaire

. Filtrage en ondelettes

3. par exemple la dérive d'un rubidium par rapport à un césium

Sixième partie

Annexes

Chapitre 13

Création de la distribution Linux

Dans ce chapitre je vais décrire les étapes de configuration complète de la carte MicroZed. La procédure pour la configuration de la carte ZedBoard est sensiblement identique mais je l'avais effectuée dans les locaux de FEMTO-ST avec l'aide de Pierre-Yves BOURGEOIS et Gwenhael GOAVEC-MEROU avant mon premier séjour dans l'entreprise MicroSystems à Toulouse. Le temps étant limité avant mon départ j'avais dû utiliser certains élément qu'ils m'avaient fourni (notamment les fichiers ps7_init_gpl) faute d'avoir pu les générer moi-même. Concernant la MicroZed j'ai généré moi-même l'ensemble de l'environnement de travail. Je me dois, tout de même, de mentionner l'aide appréciable d'Eric MEYER au long de la création du noyau Linux. Ce processus à été fastidieux (plus d'un mois) mais très formateur pour la compréhension des systèmes autonomes.

13.1 Création des fichiers ps7_init_gpl.c et ps7_init_gpl.h

Avant de se lancer dans la compilation du noyau Linux il convient de générer ces 2 fichiers prérequis pour configurer le processeur Zynq et intégrer ce paramétrage au noyau Linux qui sera généré plus tard. Pour ce faire j'ai utilisé le logiciel propriétaire de Xilinx : VIVADO version 2015.4.

Je crée un nouveau projet en spécifiant l'utilisation de la carte microzed 7010. Je crée ensuite un nouveau « Block design » dans lequel j'ajoute l'IP « ZYNQ7 processing system ». Je lance ensuite l'outil de configuration automatique « Run Block Automation » en cochant la case « Apply Board Preset ».

Maintenant je vais configurer le processeur (double clic sur le bloc ZYNQ7 Processing System). La première chose à faire est d'activer l'interface maîtresse AXI :

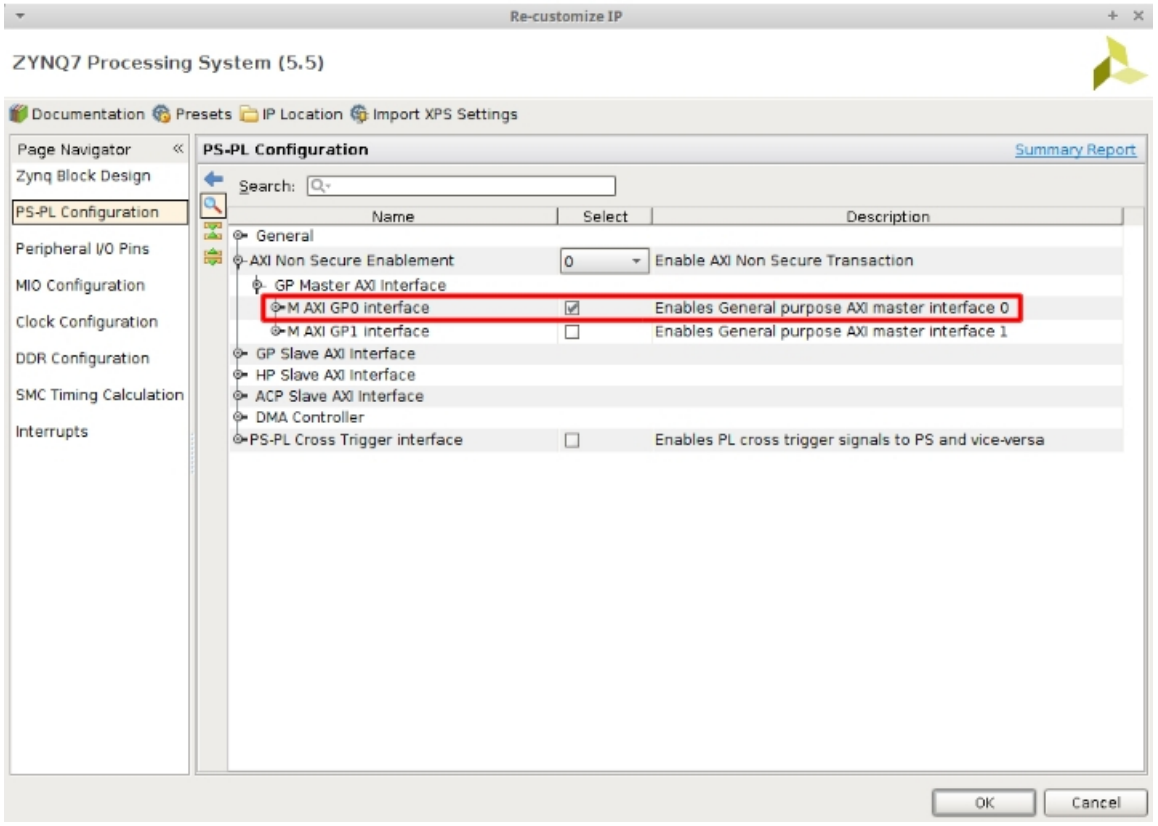


FIGURE 13.1 – Activation de l'interface AXI sur le processeur

C'est grâce à cet interface que je récupère les données issues de ma logique programmable dans l'espace utilisateur Linux.

L'étape suivante est l'activation de la liaison SPI0 en EMIO :

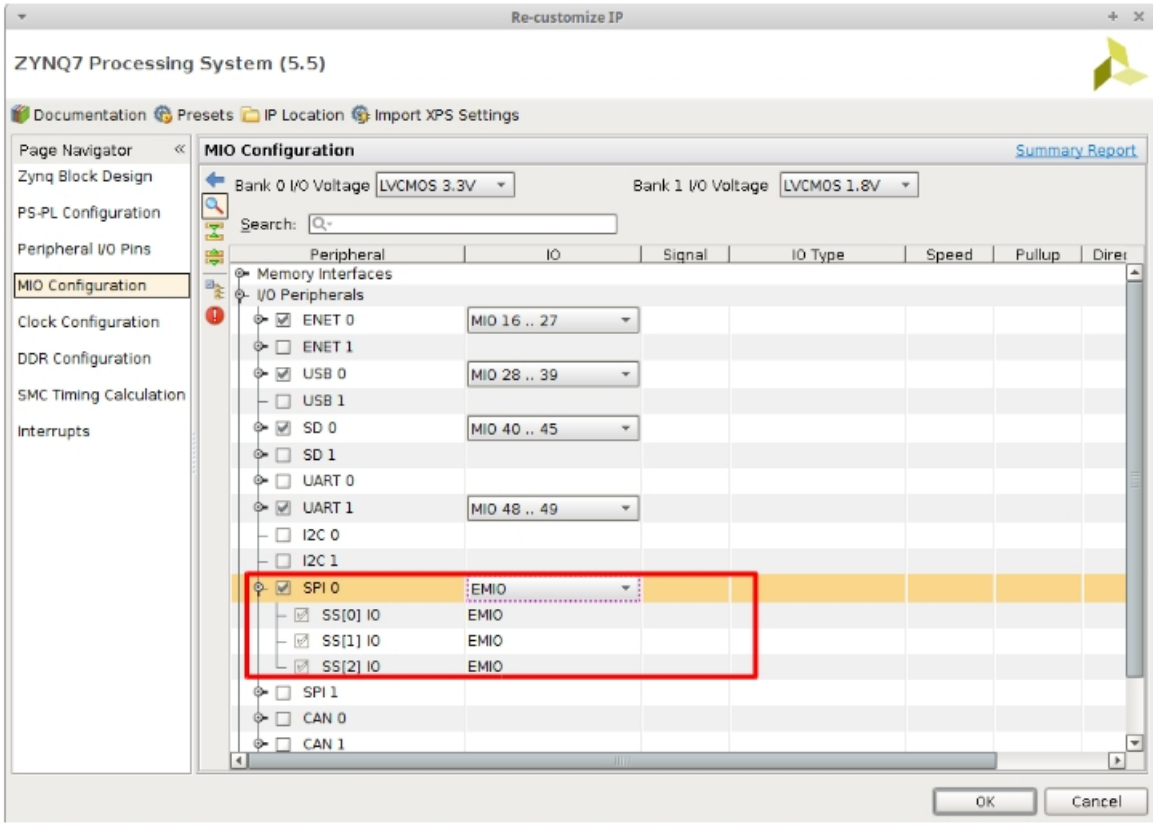


FIGURE 13.2 – Activation de la liaison SPI sur le processeur

On peut remarquer que, par défaut, tout les SS (Slave Select ou Chip Select) sont activés. Dans mon application j'utiliserai uniquement le SS[0].

La dernière étape de configuration du processeur concerne l'activation de la gestion des interruptions :

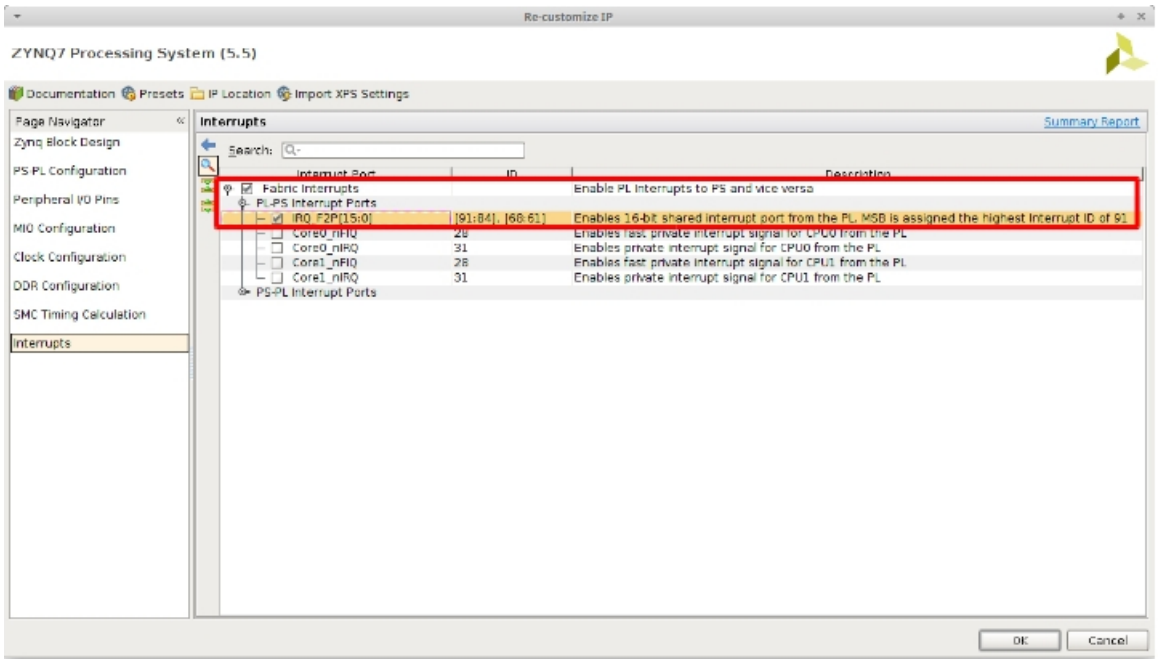


FIGURE 13.3 – Activation des interruptions sur le processeur

Maintenant le processeur est configuré selon nos besoins, il reste juste à connecter l'horloge de l'interface AXI à l'horloge FPGA (FCLK_CLK0). Le design résultant est le suivant :

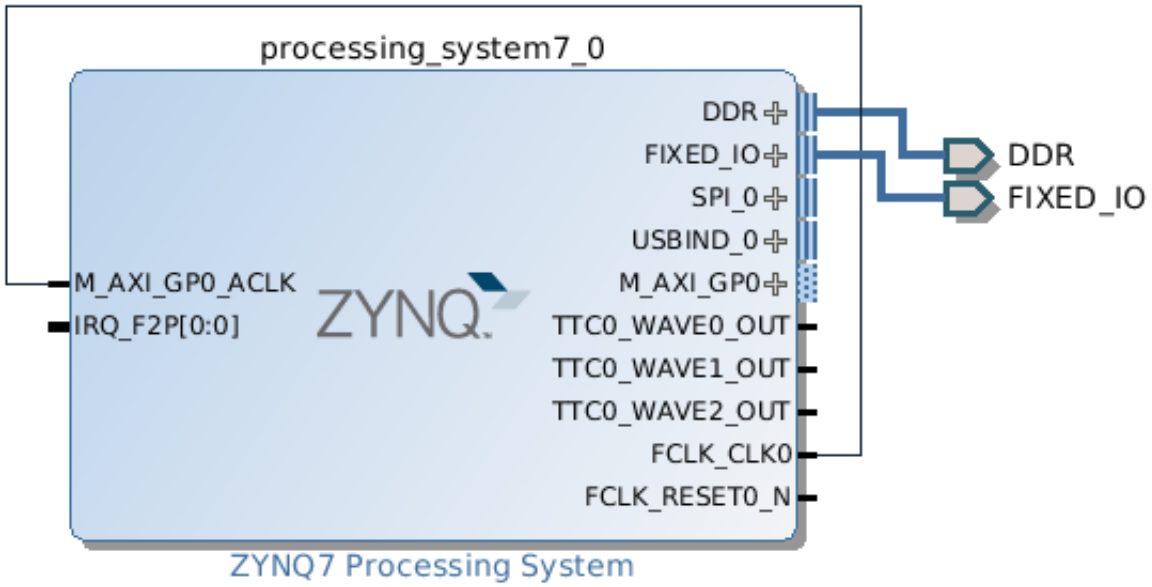


FIGURE 13.4 – Design VIVADO final

Ce design constitue la base de tout développement futur au niveau FPGA, l'ensemble des étapes mentionné précédemment doit être repris avant la génération du bitstream.

13.2 Le noyau Linux et Buildroot

La configuration des puces Zynq demande un minimum de développement. L'élément de base de cette configuration se nomme le noyau Linux (Linux kernel en anglais), toute l'architecture se construit uniquement à partir de ce noyau. Il contient plusieurs fichiers de configuration (périphériques, fréquences de fonctionnement, intégration de librairies, pilotes, ...). Dans mon cas, ces fichiers sont présent sur une carte micro-SD et sont chargés automatiquement lors du démarrage de la carte. A savoir qu'il existe plusieurs modes de boot pour ces cartes selon l'application : depuis une carte SD, depuis la mémoire flash ou sans système d'exploitation (bare metal « à l'ancienne »). Ces modes sont définis en configurant les jumpers selon le mode désiré.

L'intérêt d'utiliser un système d'exploitation est, à mon sens, double. Il permet de s'affranchir en grande partie de l'utilisation des logiciels propriétaires de Xilinx, en particulier le SDK permettant la création de l'exécutable. L'autre avantage est de pouvoir communiquer en temps réel par l'intermédiaire de fichiers de log et éventuellement de reconfigurer l'algorithme pendant son éxecution.

Il est possible de trouver des fichiers de boot tout prêt sur le web mais dans mon cas j'ai besoin de fichier particuliers à mon application. J'ai crée ces fichiers de configuration en utilisant l'outil Buildroot (version 2016.08.1). Il permet un paramétrage complet du processeur et d'y intégrer des drivers pour les composants utilisés (notament pour le DAC et la gestion des interruptions provenant de la logique programmable). Cet outil est aussi utilisable pour générer les noyau sur de

nombreuses architectures (Zynq, Intel, Terasic, Atmel, . . .). Il fonctionne par makefile en y passant les arguments adéquats.

Il existe plusieurs façons plus ou moins rigoureuses de créer ce noyau. Voici le processus complet de création du noyau que j'ai utilisé pour la carte Microzed :

- Télécharger et installer Buildroot (https ://buildroot.org/download.html)
- Ouvrir un terminal et se placer dans le répertoire d'installation contenant le makefile.
- Pour tout nettoyer et supprimer les fichiers générés : **make clean** Attention cette commande supprime tous les fichiers générés par la suite !
- Définir une architecture de base : **make zynq_microzed_defconfig** De nombreuses architecture par défaut sont intégrées, ce qui permet de s'affranchir de paramétrer la plupart des fonctionnalitées de base (fréquence CPU, liaison séries, JTAG, ethernet, RAM, flash, ...). Dans mon cas je dois rajouter le driver de mon DAC (AD5791). Ce driver étant présent dans les options je n'aurait pas à la réecrire à la main. En fait c'est le cas de la plupart des composants populaires et de nombreux drivers supplémentaire sont intégrés à buildroot à chaque nouvelles version. Toutefois l'interface SPI utilisée pour piloter mon DAC n'est pas activée par défaut, il va falloir la configurer à la main en modifiant les fichiers sources du device tree et en utilisant des fichiers de config « ps7_init_gpl » personnalisés.
- Configurer les options générales de buildroot : **make menuconfig** Cette commande va ouvrir un menu graphique permettant de naviguer dans les différents paramètres de configuration. La navigation dans les menus s'effectue avec les touches directionnelles haut et bas, les touches droite et gauche permettent de naviguer dans le menu du bas pour rentrer dans un sous-menu ou en sortir (valider avec la touche entrée), pour cocher les cases on utilise la touche espace.
- Activation de la liaison ethernet : **Target packages -> Networking applications ->** cocher la case **« dropbear »**.
- Définir un mot de passe : **System configuration -> Enable root login with password** et definir son mot de passe dans le champ **Root password**

A savoir que le nom d'utilisateur par défaut est « root »

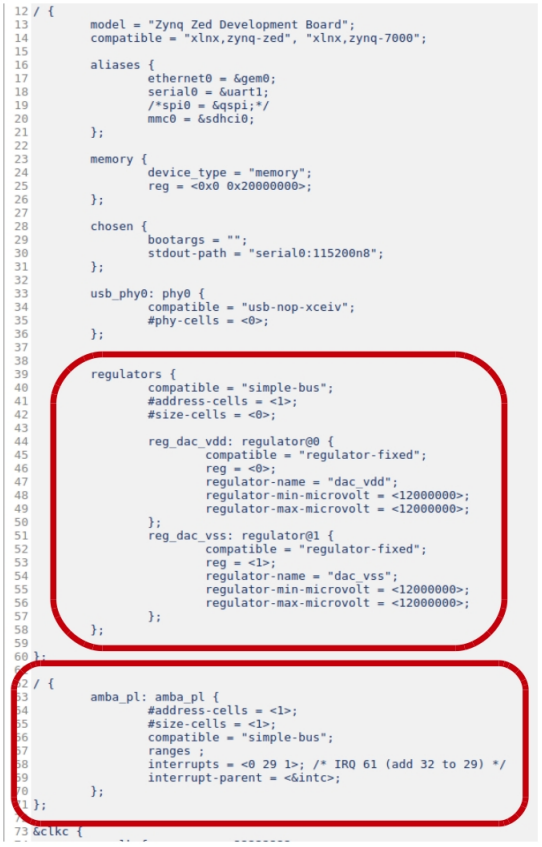
La prochaine étape est la modification du fichier dts [1] pour y activer le liaison et le gestionnaire d'interruption, cependant ce fichier n'est pas encore généré. Il va donc falloir le générer une première fois, le modifier et régénérer le fichier binaire dtb [2]. Voici la marche à suivre :

- Quitter le menu de configuration général sans oublier de sauvegarder.
- Lancer la génération des fichiers uboot : **make uboot**

Cette commande va générer plusieurs fichiers dans le répertoire output, dont le fichier dts à modifier. Elle peut prendre un certain temps à s'exécuter selon la puissance de la machine hôte (environ 10mn avec un processeur Intel Core i7-4810MQ 8 coeurs @ 2,8 GHz avec 8Go de RAM).

Une fois la génération terminée il faut aller chercher le fichier dts : **output/build/uboot-xilinx-v2016.2/arch/arm/dts/zynq-zed.dts** et faire une copie de ce fichier dans un répertoire extérieur à buildroot. Ensuite il faut modifier cette copie pour activer la liaison SPI et la gestion des interruptions. Ici une capture d'écran me semble adéquate :

1. Device Tree Source
2. Device Tree Blob



```
/ {
        model = "Zynq Zed Development Board";
        compatible = "xlnx,zynq-zed", "xlnx,zynq-7000";

        aliases {
                ethernet0 = &gem0;
                serial0 = &uart1;
                /*spi0 = &qspi;*/
                mmc0 = &sdhci0;
        };

        memory {
                device_type = "memory";
                reg = <0x0 0x20000000>;
        };

        chosen {
                bootargs = "";
                stdout-path = "serial0:115200n8";
        };

        usb_phy0: phy0 {
                compatible = "usb-nop-xceiv";
                #phy-cells = <0>;
        };


        regulators {
                compatible = "simple-bus";
                #address-cells = <1>;
                #size-cells = <0>;

                reg_dac_vdd: regulator@0 {
                        compatible = "regulator-fixed";
                        reg = <0>;
                        regulator-name = "dac_vdd";
                        regulator-min-microvolt = <12000000>;
                        regulator-max-microvolt = <12000000>;
                };
                reg_dac_vss: regulator@1 {
                        compatible = "regulator-fixed";
                        reg = <1>;
                        regulator-name = "dac_vss";
                        regulator-min-microvolt = <12000000>;
                        regulator-max-microvolt = <12000000>;
                };
        };
};
/ {
        amba_pl: amba_pl {
                #address-cells = <1>;
                #size-cells = <1>;
                compatible = "simple-bus";
                ranges ;
                interrupts = <0 29 1>; /* IRQ 61 (add 32 to 29) */
                interrupt-parent = <&intc>;
        };
};
&clkc {
```

```
        is-dual = <0>;
        num-cs = <1>;
        flash@0 {
                compatible = "n25q128a11";
                reg = <0x0>;
                spi-tx-bus-width = <1>;
                spi-rx-bus-width = <4>;
                spi-max-frequency = <50000000>;
                #address-cells = <1>;
                #size-cells = <1>;
                partition@qspi-fsbl-uboot {
                        label = "qspi-fsbl-uboot";
                        reg = <0x0 0x100000>;
                };
                partition@qspi-linux {
                        label = "qspi-linux";
                        reg = <0x100000 0x500000>;
                };
                partition@qspi-device-tree {
                        label = "qspi-device-tree";
                        reg = <0x600000 0x20000>;
                };
                partition@qspi-rootfs {
                        label = "qspi-rootfs";
                        reg = <0x620000 0x5E0000>;
                };
                partition@qspi-bitstream {
                        label = "qspi-bitstream";
                        reg = <0xC00000 0x400000>;
                };
        };
};

&spi0 {
        num-cs = <4>;
        is-decoded-cs = <0>;
        status = "okay";

        ad5791@0 {
                compatible = "ad5791";
                reg = <0>;
                vdd-supply = <&reg_dac_vdd>;
                vss-supply = <&reg_dac_vss>;
                //spi-cpol;
                spi-cpha;
                spi-max-frequency = <5000000>;
        };
};

&sdhci0 {
        u-boot,dm-pre-reloc;
        status = "okay";
};

&uart1 {
        u-boot,dm-pre-reloc;
        status = "okay";
};

&usb0 {
        status = "okay";
```

FIGURE 13.5 – Modification manuelle du fichiers dts

J'ai encadré en rouge les blocs à rajouter au fichier.

Le bloc « amba_pl » sert à activer le gestionnaire d'interruption pour un numéro d'interruption donnée (n° 61 dans mon cas), j'ai utilisé une seule interruption sur les 16 disponibles (sur Zynq 7Z010 et 7Z020). Si d'autres interruptions sont utilisées c'est à cet endroit qu'il faut les configurer.

Une fois ce fichier modifié il faut le sauvegarder et relancer la création des dtb en prenant en compte ces modifications :

— Retourner dans le menu de config général : **make menuconfig**

— Définir l'utilisation d'un devicetree personnalisé : **Kernel -> Device tree source ->** cocher la case **« Use a custom device tree file »**

— Dans le champ en dessous **« Device tree source file path »** entrer le chemin complet absolu du fichier dts modifié. Astuce : le clic molette de la souris fonctionne pour copier-coller.

La configuration générale est terminée on peut quitter le menu sans oublier de sauvegarder.

— Régénérer les fichier de boot : **make uboot-rebuild**

Maintenant il faut générer les fichier de configuration du noyau, notamment en prenant en compte les fichiers ps7_init_gpl servant à activer la liaison SPI et les interruptions sur le processeur. La on a légèrement l'impression de faire plusieurs fois la même chose mais ces étapes sont indispensables :

— make uboot-menuconfig

Un autre menu de configuration apparaît qui va nous permettre d'utiliser des fichier ps7_init_gpl crées par nous-même (voir section précédente : création des fichiers ps7_init_gpl) au lieu de la version générique : **ARM architecture ->** cocher la case **« Use custom ps7_init provided by Xilinx tool »** La il faut placer nos propores fichiers ps7_init_gpl.c et ps7_init_gpl.h dans un répertoire bien précis : **output/build/uboot-xilinx-v2016.2/board/xilinx/zynq/custom_hw_platform/** puis quitter le menu.

Ensuite relancer (encore) la génération des fichiers uboot : **make uboot-rebuild**

Il ne nous reste plus qu'à intégrer le driver du DAC (AD5791) au noyau **make linux-menuconfig** Un 3è menu s'ouvre : aller dans le sous-menu **Devices Drivers -> Industrial I/O support -> Digital to analog converters ->** cocher la case **Analog Devices AD5760/AD5780/AD5781/AD5790/AD5791 DAC SPI driver** avec une étoile * et non pas M (sinon on compilerai le pilote en tant que module externe au lieu de l'intégrer directement au noyau). Quitter le menu en sauvegardant.

Enfin, générer le noyau proprement dit (uImage) et le système de fichiers (rootfs) en tapant la commande **make** Ici buildroot va compléter les fichiers manquants, ce qui peut prendre un certain temps.

Une fois la compilation terminée, les fichiers de boot et le noyau linux sont présent dans le répertoire output. Il est fortement conseillé de copier ces fichiers en dehors du répertoire de buildroot afin de prévenir tout écrasement ultérieur (notament lors d'un make clean). Autre point important : ces fichiers doivent avoir un nom et une casse bien précise (éventuellement à renommer), que je décris dans la partie suivante.

13.3 La carte SD

Comme mentionné précédemment, le système d'exploitation a besoin des fichiers générés par buildroot pour démarrer. Ces fichiers peuvent être présent sur une carte mémoire externe ou dans la mémoire flash. Dans mon cas j'ai utilisé une carte micro SD pour booter mon système. Cette carte doit être partitionnée correctement afin de permettre au système de s'y retrouver. Pour cela j'ai utilisé le logiciel Gparted qui a l'avantage d'avoir une interface graphique confortable d'utilisation. Attention toutefois car ce logiciel est capable de formater votre propre disque dur ! Veillez donc à bien sélectionner la carte SD avant de lancer le formatage.

Pour mon application j'ai utilisé 2 partitions :

. Une partition BOOT située en début des secteurs mémoire de la carte. Le format de cette partition est FAT32, de taille 100 Mo et elle contient au minimum 4 fichiers générés par buildroot (**boot.bin**, **uboot.img**, **uImage** et **devicetree.dtb**). Ce dernier fichier est obtenu en renommant le fichier possédant l'extension .dtb généré par buildroot. Il est possible de copier-coller ces fichiers depuis le gestionnaire de fichiers graphique. A noter qu'il et possible de rajouter un fichier texte à nommer **uEnv.txt** afin de définir l'environnement de boot, par exemple définir une adresse IP par défaut (pour accès par ssh ou pour monter le NFS) et séquencer la phase de boot de façon personnalisée.

  Voici le contenu de mon fichier uEnv.txt :
  ipaddr=192.168.1.5
  serverip=192.168.1.1
  gatewayip=192.168.1.1
  netmask=255.255.255.0
  hostname=microzed
  modeboot=mmcboot
  bootargs=console=ttyPS0,115200 root=/dev/mmcblk0p2 rw earlyprintk ip=192.168.1.5 :192.168.1.1 :192.1
  crozed :eth0 :off
  mmcboot=run fpgaboot ; fatload mmc 0 0x1000000 $kernel_image && fatload mmc 0
  0x3000000 $devicetree_image && bootm 0x1000000 - 0x3000000

. Une partition ROOTFS (système de fichiers de la racine) au format EXT4 et occupant tout le reste de l'espace mémoire disponible de la carte SD.

Concernant le système de fichier il existe 2 façon de procéder : - Le décompresser directement sur la partition ROOTFS (format EXT4) de la carte SD. Cela permet d'avoir un système de fichiers persistant (mémoire morte) lors du redémarrage du système, ce qui est pratique en phase de développement afin de conserver les modifications. - Mettre l'archive (rootfs.tar) directement dans la partitions de boot. Celle-ci sera décompressée à chaque démarrage du système (mémoire vive) et ne conservera donc pas les modification apportées au système de fichiers lors d'un redémarrage. Cette méthode est utilisée en phase d'industrialisation d'un produit afin que l'éventuel client ne puisse pas corrompre des fichiers système et qu'il retrouve un système de fichier vierge à chaque démarrage.

J'ai utilisé la 1ère méthode. L'accès aux partitions de type EXT4 est apparemment soumise à autorisation et il n'est donc pas possible de faire du copier-coller comme pour les fichiers de boot. J'ai donc du utiliser le terminal pour taper la ligne de commande : **sudo tar xf rootfs.tar -C /media/abenigni/ROOTFS/**.

A présent la carte SD est prête et il ne reste plus qu'à l'insérer dans la microzed.

13.4 Montage du serveur NFS

Afin de pouvoir communiquer facilement avec le système Linux embarqué il convient de créer une zone mémoire partagée entre la machine hôte (mon ordinateur) et la microzed. La communication s'effectue par liaison Ethernet.

Attention car si l'exécutable doit écrire ou lire des fichiers dans la zone partagée et qu'elle n'est pas montée cela entraînera un blocage de l'application. Une alternative au montage d'un ser-

veur nfs est l'utilisation d'une connexion ssh depuis une machine hôte.

13.4.1 Configuration serveur

A noter que j'ai dû créer une interface Ethernet avec une adresse IP statique en plus de ma configuration DHCP de base. Pour cela j'ai rajouté dans le fichier /etc/network/interfaces les lignes suivantes :
auto enp0s25 :0
iface enp0s25 :0 inet static
address 192.168.1.1
netmask 255.255.255.0
gateway 192.168.1.1

Cette modification est persistante et sera conservée à chaque démarrage de mon ordinateur. A noter que j'ai choisi arbitrairement l'addresse IP 192.168.1.1 pour ne pas être en conflit avec un autre réseau IP. J'utilise également cet interface pour communiquer avec mes instruments de mesure (avec un hub Ethernet) en réseau local.

Maintenant la configuration du serveur NFS. La première étape est le création sur la machine hôte d'un répertoire dédié :

sudo mkdir -p /usr/local/export/zedboard

Avec les droits d'accés :

chown -R abenigni :abenigni /usr/local/export/zedboard

Dans le fichier /etc/exports (le créer s'il n'existe pas) :

/usr/local/export/microzed 192.168.1.1/255.255.255.0(rw,no_root_squash,sync)

Réinitialiser le serveur NFS :

/etc/init.d/nfs-kernel-server restart

Rajouter dans le fichier /etc/fstab à la fin :

192.168.1.1 :/usr/local/export/zedboard /opt/export nfs defaults,noauto 0 0

13.4.2 Configuration client

Enfin monter le répertoire partagé depuis la carte microzed en ayant pris soin d'y créer le répertoire dédié (dans mon cas /opt/export) :

mount 192.168.1.1 :/usr/local/export/microzed /opt/export -o nolock

Ce serveur NFS est très pratique pour transférer des fichiers (bitstreams, exécutables, scripts) et c'est dans cette zone partagée que l'exécutable va créer les fichiers de log. Toute action d'un côté sera répercutée de l'autre côté.

Un schéma pour résumer le fonctionnement final de la puce Zynq :

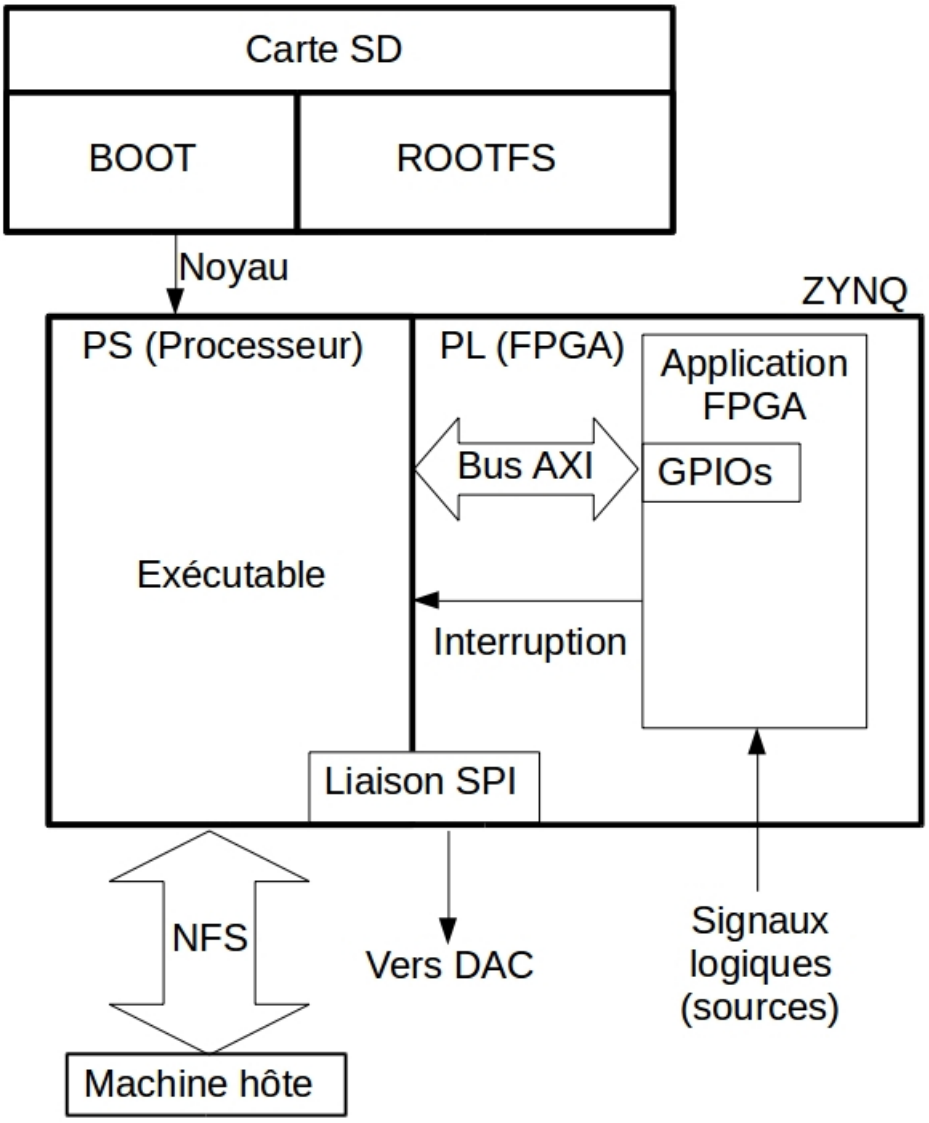


FIGURE 13.6 – Environnement Zynq configuré

Chapitre 14

Codes

Ce chapitre d'annexes comprend différents codes développés au cours de me thèse :

— FPGA

— Application C

— Monitoring Octave

14.1 Codes FPGA verilog

Codes verilog des éléments développés et utilisés dans le FPGA.

14.1.1 Sélecteur de référence

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date: 05.04.2016 11:43:58
7  // Design Name:
8  // Module Name: block_select_ref
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21
22
23 module block_select_ref(
24     input entree_vco,
25     input entree_externe,
26     input select_ref,
27     output sortie
28 );
29
30     assign sortie = (select_ref & entree_vco) | (~select_ref & entree_externe);
31
32 endmodul
```

14.1.2 Diviseur

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date: 03.10.2016 14:04:47
7  // Design Name:
8  // Module Name: DIVISEUR_100us
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21
22
23 module DIVISEUR_100us(
24     input entree_ref_10MHz,
25     input entree_source1_10MHz,
26     input entree_source2_10MHz,
27     input entree_source3_10MHz,
28     input entree_source4_10MHz,
29     input entree_source5_10MHz,
30     input entree_source6_10MHz,
31     input entree_vco_10MHz,
32     output reg sortie_100pps_ref,
33     output reg sortie_100pps_source1,
34     output reg sortie_100pps_source2,
35     output reg sortie_100pps_source3,
36     output reg sortie_100pps_source4,
37     output reg sortie_100pps_source5,
38     output reg sortie_100pps_source6,
39     output reg sortie_100pps_vco
40     );
41
42     integer cpt_ref;
43     integer cpt_source1;
44     integer cpt_source2;
45     integer cpt_source3;
46     integer cpt_source4;
47     integer cpt_source5;
48     integer cpt_source6;
49     integer cpt_vco;
50
51     always @(posedge entree_ref_10MHz) begin
52         cpt_ref = cpt_ref + 1;
53         if(cpt_ref <= 1000) begin
54             sortie_100pps_ref <= 1'b1;
55         end else begin
56             sortie_100pps_ref <= 1'b0;
57         end
58         if(cpt_ref == 100000) begin
59             cpt_ref = 0;
60         end
61     end
62
63     always @(posedge entree_source1_10MHz) begin
64         cpt_source1 = cpt_source1 + 1;
65         if(cpt_source1 <= 1000) begin
66             sortie_100pps_source1 <= 1'b1;
67         end else begin
68             sortie_100pps_source1 <= 1'b0;
69         end
70         if(cpt_source1 == 100000) begin
71             cpt_source1 = 0;
72         end
73     end
74
75     always @(posedge entree_source2_10MHz) begin
76         cpt_source2 = cpt_source2 + 1;
77         if(cpt_source2 <= 1000) begin
78             sortie_100pps_source2 <= 1'b1;
79         end else begin
80             sortie_100pps_source2 <= 1'b0;
```

```
 81          end
 82          if(cpt_source2 == 100000) begin
 83              cpt_source2 = 0;
 84          end
 85      end
 86
 87      always @(posedge entree_source3_10MHz) begin
 88          cpt_source3 = cpt_source3 + 1;
 89          if(cpt_source3 <= 1000) begin
 90              sortie_100pps_source3 <= 1'b1;
 91          end else begin
 92              sortie_100pps_source3 <= 1'b0;
 93          end
 94          if(cpt_source3 == 100000) begin
 95              cpt_source3 = 0;
 96          end
 97      end
 98
 99      always @(posedge entree_source4_10MHz) begin
100          cpt_source4 = cpt_source4 + 1;
101          if(cpt_source4 <= 1000) begin
102              sortie_100pps_source4 <= 1'b1;
103          end else begin
104              sortie_100pps_source4 <= 1'b0;
105          end
106          if(cpt_source4 == 100000) begin
107              cpt_source4 = 0;
108          end
109      end
110
111      always @(posedge entree_source5_10MHz) begin
112          cpt_source5 = cpt_source5 + 1;
113          if(cpt_source5 <= 1000) begin
114              sortie_100pps_source5 <= 1'b1;
115          end else begin
116              sortie_100pps_source5 <= 1'b0;
117          end
118          if(cpt_source5 == 100000) begin
119              cpt_source5 = 0;
120          end
121      end
122
123      always @(posedge entree_source6_10MHz) begin
124          cpt_source6 = cpt_source6 + 1;
125          if(cpt_source6 <= 1000) begin
126              sortie_100pps_source6 <= 1'b1;
127          end else begin
128              sortie_100pps_source6 <= 1'b0;
129          end
130          if(cpt_source6 == 100000) begin
131              cpt_source6 = 0;
132          end
133      end
134
135      always @(posedge entree_vco_10MHz) begin
136          cpt_vco = cpt_vco + 1;
137          if(cpt_vco <= 1000) begin
138              sortie_100pps_vco <= 1'b1;
139          end else begin
140              sortie_100pps_vco <= 1'b0;
141          end
142          if(cpt_vco == 100000) begin
143              cpt_vco = 0;
144          end
145      end
146
147
148  endmodul
```

14.1.3 Détecteur de front

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date: 18.10.2016 15:00:15
7  // Design Name:
8  // Module Name: MONO3_DETECT_FRONT
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21
22
23 module MONO3_DETECT_FRONT(
24     input horloge_compteur,
25     input entree_100hz_synchro,
26     output reg detect_front
27     );
28
29     reg valeur_precedente;
30
31     always @(posedge horloge_compteur) begin
32         if((valeur_precedente == 0) & (entree_100hz_synchro == 1)) begin
33             detect_front <= 1;
34         end else begin
35             detect_front <= 0;
36         end
37         valeur_precedente <= entree_100hz_synchro;
38     end
39
40 endmodule
```

14.1.4 Registres

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date: 18.10.2016 14:50:49
7  // Design Name:
8  // Module Name: MONO3_LATCH
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21
22
23 module MONO3_LATCH(
24     input detect_front,
25     input horloge_compteur,
26     input [23:0] in_valeur_cpt,
27     output reg [23:0] out_valeur_cpt
28     );
29
30     always @(posedge horloge_compteur) begin
31         if(detect_front == 1) begin
32             out_valeur_cpt <= in_valeur_cpt;
33         end
34     end
35
36 endmodul
```

14.1.5 Synchronisation

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date: 18.10.2016 14:48:39
7  // Design Name:
8  // Module Name: MONO3_SYNCHRO
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21
22
23 module MONO3_SYNCHRO(
24     input horloge_compteur,
25     input in_signal,
26     output reg out_signal
27     );
28
29     always @(posedge horloge_compteur) begin
30         out_signal <= in_signal;
31     end
32
33 endmodule
```

14.2 Code C

14.2.1 Driver d'interruptions

```
#include    <linux/init.h>
#include    <linux/kernel.h> // ***
#include    <linux/module.h>
#include    <linux/major.h> // ***
#include    <linux/cdev.h>
#include    <linux/fs.h> // ***
#include    <linux/poll.h> // ***
#include    <linux/delay.h>
#include    <linux/irq.h>
#include    <asm/uaccess.h>
#include    <linux/interrupt.h>
#include    <linux/sched.h> // ***
#include    <asm/io.h>

MODULE_AUTHOR("A.Benigni feat Christophe Graulle - TimeLink microsystems");
MODULE_DESCRIPTION("Interrupt driver pps reference");
MODULE_VERSION("1.0");
MODULE_LICENSE("GPL");

#define IRQ_NUMBER          165      //61
#define MODULE_ADDRESS      0x40000000

/*****************************************************************************
 * Forward definitions
 *****************************************************************************/
static int  drvPPSInit(void);
static void drvPPSExit(void);
static int  drvPPSOpen(struct inode *pInode, struct file *pFile);
static int  drvPPSRelease(struct inode *pInode, struct file *pFile);
static unsigned int drvPPSPoll(struct file *pFile, poll_table *pPollTable );

/*****************************************************************************
 * Local Data Model Definition
 *****************************************************************************/
DECLARE_WAIT_QUEUE_HEAD(drvPPSQueue);
static  int     ppsOccurred = 0;
static  int     ppsRequested = 0;
static  int     ppsConfigured = 0;
static  int     ppsAttached = 1;

static  unsigned int *pRegStat = 0;

/*****************************************************************************
 * Driver variables
 *****************************************************************************/
static  struct  file_operations drvPPSOperations =
{
        .owner      =   THIS_MODULE,
        .open           =   drvPPSOpen,
        .release    =   drvPPSRelease,
        .poll           =   drvPPSPoll,
};
static  struct  cdev    drvPPSDevice;
static  dev_t               drvPPSFirst;

static irqreturn_t ppsHandler(int irq, void *pData);

static int drvPPSInit(void)
{
    int err;

    // ******* modif alexis

//  struct ressource *res;
//  res = platform_get_ressouce(

    //*********

    printk(KERN_INFO "Initialisation du pilote d'interruption...\n");

    /*  Initialize data model   */
    ppsConfigured = 0;
    ppsAttached = 0;

    /*  Initialize Driver part  */

    /*  Get a major number  */
    err = alloc_chrdev_region( &drvPPSFirst, 0, 1, "drv_intr_microzed" );
    if ( err )
    {
```

```
81          printk(KERN_NOTICE "Error %d registering region for drv_intr_microzed\n", err);
82      }
83
84      /*  Initialize device   */
85      cdev_init( &drvPPSDevice, &drvPPSOperations );
86      drvPPSDevice.owner = THIS_MODULE;
87      drvPPSDevice.ops = &drvPPSOperations;
88
89      /*  Add device to kernel    */
90      err = cdev_add(&drvPPSDevice, drvPPSFirst, 1 );
91      if ( err )
92      {
93          printk(KERN_NOTICE "Error %d adding drv_intr_microzed device\n", err);
94      }
95      else
96      {
97          printk(KERN_INFO "drv_intr_microzed device registered [%d]\n", MAJOR(drvPPSFirst));
98      }
99
100
101
102     //  Request the IRQ for the PPS
103     err = request_irq( IRQ_NUMBER, &ppsHandler, 0, "drv_intr_microzed", 0 );
104     if ( err != 0 )
105     {
106         printk(KERN_ERR "Cannot register drv_intr_microzed handler err=%d\n",err);
107     }
108
109     err = irq_set_irq_type(IRQ_NUMBER, IRQ_TYPE_EDGE_RISING);  // interruption sur front
    montant
110     //enable_irq(IRQ_NUMBER);
111
112     //  Map the PPS module address
113     pRegStat = ioremap( MODULE_ADDRESS, 8 );
114
115     printk(KERN_INFO "Initialisation terminée.\n");
116
117 //    printk(KERN_INFO "request irq = %d\n",err);
118
119     return 0;
120 }
121
122 /*****************************************************************************
123  *   Driver exit/unload
124  *****************************************************************************/
125 static void drvPPSExit(void)
126 {
127     //  Release the PPS IRQ
128     free_irq( IRQ_NUMBER, 0 );
129
130     if ( ppsConfigured )
131     {
132     }
133
134     /*  Remove device and unregister major  */
135     cdev_del(&drvPPSDevice);
136     unregister_chrdev_region(drvPPSFirst, 1);
137     printk(KERN_INFO "Interrupt device removed !\n");
138 }
139
140 /*****************************************************************************
141  *   Open
142  * @param pInode
143  * @param pFile
144  * @return
145  *****************************************************************************/
146 static int  drvPPSOpen(struct inode *pInode, struct file *pFile)
147 {
148     ppsOccurred = 0;
149     ppsRequested = 0;
150
151     //printk(KERN_INFO "fonction drvppsopen active\n");
152
153     return 0;
154 }
155
156 /*****************************************************************************
157  *   Close
158  * @param pInode
159  * @param pFile
```

```
160   * @return
161   ***************************************************************************/
162  static int  drvPPSRelease(struct inode *pInode, struct file *pFile)
163  {
164      return 0;
165  }
166
167  /***************************************************************************
168   * Manage the polling when "select" is called.
169   * @param pFile
170   * @param pPollTable
171   * @return
172   ***************************************************************************/
173  static unsigned int drvPPSPoll(struct file *pFile, poll_table *pPollTable )
174  {
175      unsigned int mask = 0;
176
177      //printk(KERN_INFO "fonction drvppspoll active\n");
178
179      poll_wait( pFile, &drvPPSQueue, pPollTable );
180
181      if ( ppsOccurred )
182      {
183          mask |= POLLIN | POLLRDNORM;
184
185          ppsOccurred = 0;
186          ppsRequested = 0;
187      }
188
189      if ( mask == 0 )
190      {
191          ppsRequested = 1;
192      }
193
194      return mask;
195  }
196
197  /***************************************************************************
198   *   Manage the IRQ generated by an incoming PPS.
199   * @param port   port number
200   * @param    bit     bit in port
201   ***************************************************************************/
202  static irqreturn_t ppsHandler(int irq, void *pData)
203  {
204      ppsOccurred = 1;
205  //  printk(KERN_INFO "fonction handler active\n");
206
207      wake_up_interruptible( &drvPPSQueue );
208
209      //printk(KERN_INFO "fonction handler termine\n");
210
211      //  IRQ has been processed
212      return IRQ_HANDLED;
213  }
214
215  module_init(drvPPSInit);
216  module_exit(drvPPSExit);
217
218
219
220
```

14.2.2 Application

```
1
2
3
4  // ##### A compiler en utilisant la commande "arm-linux-gcc toto.c -o exec_toto -g -Wall -lm
   -pthread"
5
6
7  //{ Includes, defines, déclaration des variables globales et des fonctions/threads
8
9  //{ Includes librairies
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <sys/types.h>
13 #include <sys/stat.h>
14 #include <sys/mman.h>
15 #include <fcntl.h>
16 #include <unistd.h>
17 #include <errno.h>
18 #include <stdint.h>
19 #include <time.h>
20 #include <error.h>
21 #include <string.h>
22 #include <math.h>       // Ajouter -lm à la commande de compilation
23 #include <pthread.h>    // Ajouter -pthread
24 //}
25
26 //{ Defines
27
28 // Définition des addresse des périphériques (dans VIVADO)
29 #define gpio_cpt_ref_ext_addr       0x41200000
30 #define gpio_cpt_source1_addr       0x41210000
31 #define gpio_cpt_source2_addr       0x41220000
32 #define gpio_cpt_source3_addr       0x41230000
33 #define gpio_cpt_source4_addr       0x41240000
34 #define gpio_cpt_source5_addr       0x41250000
35 #define gpio_cpt_source6_addr       0x41260000
36 #define gpio_cpt_vco_addr           0x41270000
37 #define gpio_select_ref_addr        0x41280000
38
39 // Fichiers de log et de config
40 #define log_entrees_filename        "/opt/export/log/log_entrees.txt"
41 #define log_interne_filename        "/opt/export/log/log_interne.txt"
42 #define log_sorties_filename        "/opt/export/log/log_sorties.txt"
43 #define config_vco_offset_filename  "/opt/config_offset.cfg"    // Pour décalage manuel de la
   fréquence du VCO (n'existe pas par défaut : "echo [offset] > config_offset.cfg")
44 #define config_bruits_filename      "/opt/config_bruit.cfg"     // Pour config manuelle des
   matrices de bruits (n'existe pas par défaut : "echo [bruit_process;bruit_mesure] >
   config_bruit.cfg")
45 #define config_gain_filename        "/opt/config_gain.cfg"      // Pour config manuelle des
   gains (n'existe pas par défaut : "echo [gain_source1;2;3;4;5;6] > config_gain.cfg")
46
47 // Paranètres spécifiques à la manip
48 #define freq_correction_max     5          // Amplitude de correction de fréquence maximale en
   pas de commande du DAC sur la durée d'échantillonnage
49                                            //  Elle dépend de la stabilité du VCO sur la
   durée choisie (10s)   zedboard:5    microzed(plage=10V):10    microzed(plage=2V):20
50 #define intr_driver_name        "/dev/drv_intr_ref"  // Nom du driver d'interruption
   zedboard:"/dev/drv_intr_ref"    microzed:"/dev/drv_intr_microzed"
51 #define sensibilite_VCO         423729      // Sensibilité du VCO asservi en pas de commande
   par hertz BVA10:423729         VCO_AR(plage=10V):83551    VCO_AR2(plage=2V) : 369600
52
53 // Paramètres divers
54 #define freq_rapide             320         // Fréquence du compteur rapide (intervallometre)
   en MHz, défini dans le code FPGA
55 #define MAP_SIZE                4096UL                  // Utilisé par la communication avec
   les gpio
56 #define MAP_MASK                (MAP_SIZE - 1)          // Idem
57 #define IIO_BASE_ADDR           "/sys/bus/iio/devices/iio:device1"    // Utilisé pour la
   communication avec le DAC(SPI)
58 #define nb_ech_reg              100        // Nombre d'échantillons @ 100Hz sur lesquels sont
   effectués la regression linéaire
59 #define freq_commande_min       1          // Fréquence de commande minimale à envoyer au DAC
60 #define freq_commande_max       1048575    // Fréquence de commande maximale à envoyer au DAC
   (20bits : 1048576)
61 #define profondeur_compteur_rapide  16777216    // A ajouter en cas d'occurence d'un rollover
   32bits:4294967296 ; 18bits:262144 ; 20bits:1048576 ; 30bits:1073741824 ; 24bits:16777216
62
63 // Coefficients de dérive de fréquence des sources pour intégration au modèle prédictif du
   Kalman
64 #define coeff_derive_source1    0
```

```
65  #define coeff_derive_source2    0
66  #define coeff_derive_source3    0
67  #define coeff_derive_source4    0
68  #define coeff_derive_source5    0
69  #define coeff_derive_source6    0
70
71
72  //}
73
74  //{ Déclaration des variables globales
75  static int affichage = 1;               // On peut éventuellement définir un niveau
    d'affichage des informations (debug, normal ou aucun) à implémenter !
76  static int launch_thread_principal = 0; // Drapeau de déclenchement du processus principal
    (regression, asservissement)
77  static int enable_kalman = 1;           // Active le Kalman
78  static int enable_asservissement = 0;   // Active l'envoi de la commande (en fonction des
    arguments passé au main lors de l'appel de l'exécutable)
79  static int timestamp = 0;               // Datation générale du programme
80  static FILE *logfile_entrees_fp;        // Fichier de log@100Hz des valeurs brutes et
    intervalles de temps associés
81  static FILE *logfile_interne_fp;        // Fichier de log@1Hz des résultats des régressions et
    asservissement
82  static FILE *logfile_sorties_fp;        // Fichier de log des commandes envoyées au CNA
83  static FILE *config_vco_offset_fp;
84  static FILE *config_bruits_fp;
85  static FILE *config_gain_fp;
86  static double moyX = 0;
87  static double Sxx = 0;
88  static FILE *fd_dac;                    // Fichier d'interfaçage du DAC
89  static int freq_commande = 500000;      // Milieu de la plage de commande du VCO = 500000
90  static double freq_correction = 0;
91  static void* virt_addr_write_select_ref;    // Global car défini dans le main et utilisé dans
    le processus principal
92
93  // Variables modifiées par les fichiers de config
94  static double offset_freq_manuel = 0;       // Permet un ajustement manuel de la fréquence du
    VCO par fichier de config
95  static double offset_freq_manuel_prec = 0;
96
97  // Paramètres du Kalman
98  static double bruit_process = 1e-12;        // Ecart-type du bruit de process (source+VCO) sur
    la durée d'intégration (voir courbe Adev)
99                                              // Reflète la confiance accordée à la prédiction
    (modèle d'évolution temporelle de le fréquence de la source)
100 static double bruit_mesure = 1e-10;        // Ecart-type du bruit de mesure sur la durée
    d'intégration
101                                             // résolution de la mesure de fréquence
    NORMALISéE, soit resolution en Hz/freq_nominale : 3.125mHz/10MHz = 3.125.10-10
102                                             // Reflète la confiance accordée aux mesures
    (intervallomètre)
103 static double coeff_gain_source1 = 1;
104 static double coeff_gain_source2 = 1;
105 static double coeff_gain_source3 = 1;
106 static double coeff_gain_source4 = 1;
107 static double coeff_gain_source5 = 1;
108 static double coeff_gain_source6 = 1;
109
110 static double eff_commande = 1;
111
112 static pthread_mutex_t __LOCK__ = PTHREAD_MUTEX_INITIALIZER;
113 static pthread_cond_t __COND__ = PTHREAD_COND_INITIALIZER;
114
115
116 //}
117
118 //{ Déclaration des fonctions et structures
119
120 struct horloge{
121     int is_present;
122     long int valeur_brute_cpt;
123     long int valeur_brute_cpt_prec;
124     long int interval_100Hz;
125     long int interval_100Hz_prec;
126     double tab_cpt_rapide_temp[nb_ech_reg];
127     double tab_cpt_rapide[nb_ech_reg];
128     double freq_diff_100s;  //  Issu d'un regression linéaire sur 100s
129     double interval_temps;
130     double interval_temps_prec;
131     double var_res_reg_ppl;
132     double ordon_origin;
```

```
133     double seuil_incertitude;
134     double pente;
135     double pente_prec;
136     double freq_diff;
137     double freq_diff_precedent;
138     double moy_accroissement;   // Accroissement moyen des compteurs sur 1 s moyenné sur 100 s
139     double freq_moy_100s;       //
140     double freq_moy_100s_init;
141     int nb_recouvrements;       // nb de dépassemet de la valeur de pente cpt
142     int nb_bugs;
143     int nb_bugs_successif;
144
145     double tau;     // Pas de mesure (inutilisé)
146     double coeffGain;
147
148     double etat_pred;
149     double P_pred;
150     double K;
151     double etat_est;
152     double etat_mes;
153     double Q,R;
154     double P_est;
155     double coeff_derive;
156 };
157
158 int regression_principale(struct horloge* proto_horloge);
159 int getAD5791Params(char *basename, int *offset, long double *scale);
160 void kalmanFiltrage(struct horloge* horloge_kalman);
161 void init_kalman(struct horloge* horloge_init);
162
163 static struct horloge source1;
164 static struct horloge source2;
165 static struct horloge source3;
166 static struct horloge source4;
167 static struct horloge source5;
168 static struct horloge source6;
169 static struct horloge vco;
170 static struct horloge ref_externe;
171
172 //}
173
174 //}
175
176 void *thread_principal(void *t)
177 {
178     printf("Processus principal actif\n");
179
180     int i;
181     int rep;
182     int timestamp_init = (-2) * nb_ech_reg/100;
183     int init_done = 0;
184     int vco_caley = 0;
185     double coeff_calage_vco = 0.5;
186
187     double moy_incertitude_source1 = 0;
188     double moy_incertitude_source2 = 0;
189     double moy_incertitude_source3 = 0;
190     double moy_incertitude_source4 = 0;
191     double moy_incertitude_source5 = 0;
192     double moy_incertitude_source6 = 0;
193     double moy_incertitude_vco = 0;
194
195     double nb_mesures_incert_source1 = 0;
196     double nb_mesures_incert_source2 = 0;
197     double nb_mesures_incert_source3 = 0;
198     double nb_mesures_incert_source4 = 0;
199     double nb_mesures_incert_source5 = 0;
200     double nb_mesures_incert_source6 = 0;
201     double nb_mesures_incert_vco = 0;
202
203     double moy_accroiss_source1 = 0;
204     double moy_accroiss_source2 = 0;
205     double moy_accroiss_source3 = 0;
206     double moy_accroiss_source4 = 0;
207     double moy_accroiss_source5 = 0;
208     double moy_accroiss_source6 = 0;
209     double moy_accroiss_vco = 0;
210
211     while(1)
212     {
```

```
213          if (pthread_mutex_lock(&__LOCK__)) { abort(); }     // ICI TECHNIQUE DE SEKOU QUI FAIT
    PLAISIR (à base de jetons)
214
215          while(launch_thread_principal != 1){ if (pthread_cond_wait(&__COND__, &__LOCK__))
    { abort(); } }    // Attente du signal en provenance du "main"
216
217          for(i=0;i<nb_ech_reg;i++)      // Transfert des tableaux d'intervalles de temps pour
    éviter toute modification parallèle par le "main" pendant l'éxecution du processus
218          {
219              if(source1.is_present){ source1.tab_cpt_rapide[i] =
    source1.tab_cpt_rapide_temp[i]; }
220              if(source2.is_present){ source2.tab_cpt_rapide[i] =
    source2.tab_cpt_rapide_temp[i]; }
221              if(source3.is_present){ source3.tab_cpt_rapide[i] =
    source3.tab_cpt_rapide_temp[i]; }
222              if(source4.is_present){ source4.tab_cpt_rapide[i] =
    source4.tab_cpt_rapide_temp[i]; }
223              if(source5.is_present){ source5.tab_cpt_rapide[i] =
    source5.tab_cpt_rapide_temp[i]; }
224              if(source6.is_present){ source6.tab_cpt_rapide[i] =
    source6.tab_cpt_rapide_temp[i]; }
225              if(vco.is_present){ vco.tab_cpt_rapide[i] = vco.tab_cpt_rapide_temp[i]; }
226          }
227          launch_thread_principal = 0;
228
229          if (pthread_cond_broadcast(&__COND__)) { abort(); }
230          if (pthread_mutex_unlock(&__LOCK__)) { abort(); }
231
232
233  //{ Début boucle d'initialisation
234
235          if(init_done == 0)
236          {
237              timestamp_init += nb_ech_reg/100;
238              printf("#INIT# %d\t",timestamp_init);
239
240              if(source1.is_present)
241              {
242                  regression_principale(&source1);
243                  if(source1.var_res_reg_ppl > source1.seuil_incertitude)
244                  {       // Contrôle de la validité de la mesure (variance des résidus)
245                      source1.interval_temps = source1.interval_temps_prec + source1.pente_prec
    * nb_ech_reg; source1.nb_bugs++;
246                  }
247                  else{ source1.pente_prec = source1.pente; }
248                  source1.freq_diff = (source1.interval_temps - source1.interval_temps_prec) *
    ((double)(10/(nb_ech_reg/100))/(double)freq_rapide);
249                  source1.interval_temps_prec = source1.interval_temps;
250                  if(affichage) printf("1->%7.1f %1.2e/%1.2e/
    %d",source1.interval_temps,source1.freq_diff,source1.var_res_reg_ppl,source1.nb_bugs);
251              }
252
253              if(source2.is_present){ regression_principale(&source2);
254                  if(source2.var_res_reg_ppl > source2.seuil_incertitude){       // Contrôle de
    la validité de la mesure (variance des résidus)
255                      source2.interval_temps = source2.interval_temps_prec + source2.pente_prec
    * nb_ech_reg; source2.nb_bugs++;}
256                  else{ source2.pente_prec = source2.pente; }
257                  source2.freq_diff = (source2.interval_temps - source2.interval_temps_prec) *
    ((double)(10/(nb_ech_reg/100))/(double)freq_rapide);
258                  source2.interval_temps_prec = source2.interval_temps;
259                  if(affichage) printf("\t2->%7.1f %1.2e/%1.2e/
    %d",source2.interval_temps,source2.freq_diff,source2.var_res_reg_ppl,source2.nb_bugs); }
260
261              if(source3.is_present){ regression_principale(&source3);
262                  if(source3.var_res_reg_ppl > source3.seuil_incertitude){       // Contrôle de
    la validité de la mesure (variance des résidus)
263                      source3.interval_temps = source3.interval_temps_prec + source3.pente_prec
    * nb_ech_reg; source3.nb_bugs++;}
264                  else{ source3.pente_prec = source3.pente; }
265                  source3.freq_diff = (source3.interval_temps - source3.interval_temps_prec) *
    ((double)(10/(nb_ech_reg/100))/(double)freq_rapide);
266                  source3.interval_temps_prec = source3.interval_temps;
267                  if(affichage) printf("\t3->%7.1f %1.2e/%1.2e/
    %d",source3.interval_temps,source3.freq_diff,source3.var_res_reg_ppl,source3.nb_bugs); }
268
269              if(source4.is_present){ regression_principale(&source4);
270                  if(source4.var_res_reg_ppl > source4.seuil_incertitude){       // Contrôle de
    la validité de la mesure (variance des résidus)
271                      source4.interval_temps = source4.interval_temps_prec + source4.pente_prec
```

```
* nb_ech_reg; source4.nb_bugs++;}
272                 else{ source4.pente_prec = source4.pente; }
273                 source4.freq_diff = (source4.interval_temps - source4.interval_temps_prec) *
((double)(10/(nb_ech_reg/100))/(double)freq_rapide);
274                 source4.interval_temps_prec = source4.interval_temps;
275                 if(affichage) printf("\t4->%7.1f %1.2e/%1.2e/
%d",source4.interval_temps,source4.freq_diff,source4.var_res_reg_ppl,source4.nb_bugs); }
276
277             if(source5.is_present){ regression_principale(&source5);
278                 if(source5.var_res_reg_ppl > source5.seuil_incertitude){       // Contrôle de
la validité de la mesure (variance des résidus)
279                     source5.interval_temps = source5.interval_temps_prec + source5.pente_prec
* nb_ech_reg; source5.nb_bugs++;}
280                 else{ source5.pente_prec = source5.pente; }
281                 source5.freq_diff = (source5.interval_temps - source5.interval_temps_prec) *
((double)(10/(nb_ech_reg/100))/(double)freq_rapide);
282                 source5.interval_temps_prec = source5.interval_temps;
283                 if(affichage) printf("\t5->%7.1f %1.2e/%1.2e/
%d",source5.interval_temps,source5.freq_diff,source5.var_res_reg_ppl,source5.nb_bugs); }
284
285             if(source6.is_present){ regression_principale(&source6);
286                 if(source6.var_res_reg_ppl > source6.seuil_incertitude){       // Contrôle de
la validité de la mesure (variance des résidus)
287                     source6.interval_temps = source6.interval_temps_prec + source6.pente_prec
* nb_ech_reg; source6.nb_bugs++;}
288                 else{ source6.pente_prec = source6.pente; }
289                 source6.freq_diff = (source6.interval_temps - source6.interval_temps_prec) *
((double)(10/(nb_ech_reg/100))/(double)freq_rapide);
290                 source6.interval_temps_prec = source6.interval_temps;
291                 if(affichage) printf("\t6->%7.1f %1.2e/%1.2e/
%d",source6.interval_temps,source6.freq_diff,source6.var_res_reg_ppl,source6.nb_bugs); }
292
293             if(vco.is_present){ regression_principale(&vco);
294                 if(vco.var_res_reg_ppl > vco.seuil_incertitude){       // Contrôle de la
validité de la mesure (variance des résidus)
295                     vco.interval_temps = vco.interval_temps_prec + vco.pente_prec *
nb_ech_reg; vco.nb_bugs++;}
296                 else{ vco.pente_prec = vco.pente; }
297                 vco.freq_diff = (vco.interval_temps - vco.interval_temps_prec) * ((double)(10/
(nb_ech_reg/100))/(double)freq_rapide);
298                 vco.interval_temps_prec = vco.interval_temps;
299                 if(affichage) printf("\tVCO->%7.1f %1.2e/%1.2e/
%d",vco.interval_temps,vco.freq_diff,vco.var_res_reg_ppl,vco.nb_bugs); }
300
301             printf("\n");
302
303             if(timestamp_init > 0)
304             {
305                 source1.moy_accroissement += source1.freq_diff * ((double)freq_rapide/
((double)(10/(nb_ech_reg/100))));
306                 source2.moy_accroissement += source2.freq_diff * ((double)freq_rapide/
((double)(10/(nb_ech_reg/100))));
307                 source3.moy_accroissement += source3.freq_diff * ((double)freq_rapide/
((double)(10/(nb_ech_reg/100))));
308                 source4.moy_accroissement += source4.freq_diff * ((double)freq_rapide/
((double)(10/(nb_ech_reg/100))));
309                 source5.moy_accroissement += source5.freq_diff * ((double)freq_rapide/
((double)(10/(nb_ech_reg/100))));
310                 source6.moy_accroissement += source6.freq_diff * ((double)freq_rapide/
((double)(10/(nb_ech_reg/100))));
311                 vco.moy_accroissement += vco.freq_diff * ((double)freq_rapide/((double)(10/
(nb_ech_reg/100))));
312
313                 if(source1.var_res_reg_ppl < source1.seuil_incertitude)
{ moy_incertitude_source1 += source1.var_res_reg_ppl; nb_mesures_incert_source1++; }
314                 if(source2.var_res_reg_ppl < source2.seuil_incertitude)
{ moy_incertitude_source2 += source2.var_res_reg_ppl; nb_mesures_incert_source2++; }
315                 if(source3.var_res_reg_ppl < source3.seuil_incertitude)
{ moy_incertitude_source3 += source3.var_res_reg_ppl; nb_mesures_incert_source3++; }
316                 if(source4.var_res_reg_ppl < source4.seuil_incertitude)
{ moy_incertitude_source4 += source4.var_res_reg_ppl; nb_mesures_incert_source4++; }
317                 if(source5.var_res_reg_ppl < source5.seuil_incertitude)
{ moy_incertitude_source5 += source5.var_res_reg_ppl; nb_mesures_incert_source5++; }
318                 if(source6.var_res_reg_ppl < source6.seuil_incertitude)
{ moy_incertitude_source6 += source6.var_res_reg_ppl; nb_mesures_incert_source6++; }
319                 if(vco.var_res_reg_ppl < vco.seuil_incertitude){ moy_incertitude_vco +=
vco.var_res_reg_ppl; nb_mesures_incert_vco++; }
320
321  //{ Calage de la fréquence du VCO sur la référence externe
322
```

```
323                 if((ref_externe.is_present) & (enable_asservissement) & (!vco_caley) )
324                 {
325                     double freq_correction = (vco.freq_diff);
326                     freq_correction *= (double) sensibilite_VCO;
327                     freq_correction *= (double) ( coeff_calage_vco );
328                     freq_correction = round(freq_correction);
329                     coeff_calage_vco -= 0.02;
330 
331                     freq_commande -= freq_correction;
332 
333                     if(freq_commande <= freq_commande_min){
334                         printf("DEPASSEMENT COMMANDE - !!!!\n");
335                         freq_commande = freq_commande_min;
336                     }
337                     if(freq_commande >= freq_commande_max){
338                         printf("DEPASSEMENT COMMANDE + !!!!\n");
339                         freq_commande = freq_commande_max;
340                     }
341 
342                     printf("\n\tCalage du VCO sur la fréquence de référence -> Correction : %f
    \t - Commande DAC : %d\n",freq_correction,freq_commande);
343 
344                     // Envoi de la commande au DAC
345                     int rep = fprintf(fd_dac, "%ld", (long) (freq_commande));
346                     if (rep < 0) printf("ERREUR ENVOI COMMANDE AU CNA !!! %d\n", rep);
347                     fseek(fd_dac, SEEK_SET, 0);
348                 }
349  //}
350 
351             }
352 
353             if( (timestamp_init % 100 == 0) && (timestamp_init > 0) )
354             {
355                 coeff_calage_vco = 0.5;
356 
357                 source1.moy_accroissement /= (100/(nb_ech_reg/100));
358                 source2.moy_accroissement /= (100/(nb_ech_reg/100));
359                 source3.moy_accroissement /= (100/(nb_ech_reg/100));
360                 source4.moy_accroissement /= (100/(nb_ech_reg/100));
361                 source5.moy_accroissement /= (100/(nb_ech_reg/100));
362                 source6.moy_accroissement /= (100/(nb_ech_reg/100));
363                 vco.moy_accroissement /= (100/(nb_ech_reg/100));
364 
365                 source1.freq_moy_100s = source1.moy_accroissement * (((double)(10/
    (nb_ech_reg/100)))/(double)freq_rapide);
366                 source2.freq_moy_100s = source2.moy_accroissement * (((double)(10/
    (nb_ech_reg/100)))/(double)freq_rapide);
367                 source3.freq_moy_100s = source3.moy_accroissement * (((double)(10/
    (nb_ech_reg/100)))/(double)freq_rapide);
368                 source4.freq_moy_100s = source4.moy_accroissement * (((double)(10/
    (nb_ech_reg/100)))/(double)freq_rapide);
369                 source5.freq_moy_100s = source5.moy_accroissement * (((double)(10/
    (nb_ech_reg/100)))/(double)freq_rapide);
370                 source6.freq_moy_100s = source6.moy_accroissement * (((double)(10/
    (nb_ech_reg/100)))/(double)freq_rapide);
371                 vco.freq_moy_100s = vco.moy_accroissement * (((double)(10/(nb_ech_reg/100)))/
    (double)freq_rapide);
372 
373                 source1.freq_moy_100s_init = source1.freq_moy_100s;
374                 source2.freq_moy_100s_init = source2.freq_moy_100s;
375                 source3.freq_moy_100s_init = source3.freq_moy_100s;
376                 source4.freq_moy_100s_init = source4.freq_moy_100s;
377                 source5.freq_moy_100s_init = source5.freq_moy_100s;
378                 source6.freq_moy_100s_init = source6.freq_moy_100s;
379                 vco.freq_moy_100s_init = vco.freq_moy_100s;
380 
381                 moy_incertitude_source1 /= nb_mesures_incert_source1;
382                 moy_incertitude_source2 /= nb_mesures_incert_source2;
383                 moy_incertitude_source3 /= nb_mesures_incert_source3;
384                 moy_incertitude_source4 /= nb_mesures_incert_source4;
385                 moy_incertitude_source5 /= nb_mesures_incert_source5;
386                 moy_incertitude_source6 /= nb_mesures_incert_source6;
387                 moy_incertitude_vco /= nb_mesures_incert_vco;
388 
389                 if(moy_incertitude_source1 > 1){ source1.seuil_incertitude = 2 *
    moy_incertitude_source1; }else{ source1.seuil_incertitude = 1; }
390                 if(moy_incertitude_source2 > 1){ source2.seuil_incertitude = 2 *
    moy_incertitude_source2; }else{ source2.seuil_incertitude = 1; }
391                 if(moy_incertitude_source3 > 1){ source3.seuil_incertitude = 2 *
    moy_incertitude_source3; }else{ source3.seuil_incertitude = 1; }
```

```
392                 if(moy_incertitude_source4 > 1){ source4.seuil_incertitude = 2 *
    moy_incertitude_source4; }else{ source4.seuil_incertitude = 1; }
393                 if(moy_incertitude_source5 > 1){ source5.seuil_incertitude = 2 *
    moy_incertitude_source5; }else{ source5.seuil_incertitude = 1; }
394                 if(moy_incertitude_source6 > 1){ source6.seuil_incertitude = 2 *
    moy_incertitude_source6; }else{ source6.seuil_incertitude = 1; }
395                 if(moy_incertitude_vco > 1){ vco.seuil_incertitude = 2 *
    moy_incertitude_vco; }else{ vco.seuil_incertitude = 1; }
396
397                 if(enable_kalman)
398                 {
399                     if(source1.is_present) init_kalman(&source1);
400                     if(source2.is_present) init_kalman(&source2);
401                     if(source3.is_present) init_kalman(&source3);
402                     if(source4.is_present) init_kalman(&source4);
403                     if(source5.is_present) init_kalman(&source5);
404                     if(source6.is_present) init_kalman(&source6);
405                     if(vco.is_present) init_kalman(&vco);
406                 }
407
408                 if(!vco_caley)
409                 {
410                     if(!ref_externe.is_present)
411                     {
412                         if(enable_asservissement)
413                         {
414                             printf("Asservissement du VCO sans calage.\n");
415                             init_done = 1;
416                         }
417                         else
418                         {
419                             printf("Mesure des sources / VCO\n");
420                             init_done = 1;
421                         }
422                     }
423                     else
424                     {
425                         if(enable_asservissement)
426                         {
427                             if((vco.freq_diff == 0) || (timestamp_init == 1000) )
428                             {
429                                 printf("VCO calé, désactivation de la référence externe et
    recalcul des seuils et accroissements moyens.\n");
430                                 *((int *) virt_addr_write_select_ref) = 1;  // Le VCO fournit
    la référence
431                                 timestamp_init = (-2) * nb_ech_reg/100;     // Timestamp
    négatif pour rétablissement dû au clock gating
432                                 vco_caley = 1;
433                                 ref_externe.is_present = 0;
434
435                                 source1.moy_accroissement = 0;
436                                 source2.moy_accroissement = 0;
437                                 source3.moy_accroissement = 0;
438                                 source4.moy_accroissement = 0;
439                                 source5.moy_accroissement = 0;
440                                 source6.moy_accroissement = 0;
441                                 vco.moy_accroissement = 0;
442
443                                 moy_incertitude_source1 = 0;
444                                 moy_incertitude_source2 = 0;
445                                 moy_incertitude_source3 = 0;
446                                 moy_incertitude_source4 = 0;
447                                 moy_incertitude_source5 = 0;
448                                 moy_incertitude_source6 = 0;
449                                 moy_incertitude_vco = 0;
450
451                                 nb_mesures_incert_source1 = 0;
452                                 nb_mesures_incert_source2 = 0;
453                                 nb_mesures_incert_source3 = 0;
454                                 nb_mesures_incert_source4 = 0;
455                                 nb_mesures_incert_source5 = 0;
456                                 nb_mesures_incert_source6 = 0;
457                                 nb_mesures_incert_vco = 0;
458
459
460                             }
461                             else
462                             {
463                                 printf("Affinage du calage du VCO sur la fréquence de
    référence externe.\n");
```

```
464
465                                 source1.moy_accroissement = 0;
466                                 source2.moy_accroissement = 0;
467                                 source3.moy_accroissement = 0;
468                                 source4.moy_accroissement = 0;
469                                 source5.moy_accroissement = 0;
470                                 source6.moy_accroissement = 0;
471                                 vco.moy_accroissement = 0;
472
473                                 moy_incertitude_source1 = 0;
474                                 moy_incertitude_source2 = 0;
475                                 moy_incertitude_source3 = 0;
476                                 moy_incertitude_source4 = 0;
477                                 moy_incertitude_source5 = 0;
478                                 moy_incertitude_source6 = 0;
479                                 moy_incertitude_vco = 0;
480
481                                 nb_mesures_incert_source1 = 0;
482                                 nb_mesures_incert_source2 = 0;
483                                 nb_mesures_incert_source3 = 0;
484                                 nb_mesures_incert_source4 = 0;
485                                 nb_mesures_incert_source5 = 0;
486                                 nb_mesures_incert_source6 = 0;
487                                 nb_mesures_incert_vco = 0;
488                             }
489                         }else
490                         {
491                             printf("Mesure des sources / référence externe\n");
492                             init_done = 1;
493                         }
494                     }
495                 }
496                 else
497                 {
498                     init_done = 1;
499                 }
500
501                 if(init_done)
502                 {
503                     printf("Initialisation des seuils et paramétrage du filtre terminées.\n");
504                     printf("Moy accroissements : 1->%6.0f\t2->%6.0f\t3->%6.0f\t4->%6.0f\t5->
    %6.0f\t6->%6.0f\tVCO->
    %6.0f\n",source1.moy_accroissement,source2.moy_accroissement,source3.moy_accroissement,source4.
    moy_accroissement,source5.moy_accroissement,source6.moy_accroissement,vco.moy_accroissement);
505                     printf("Seuil incertitude : 1->%1.2e\t2->%1.2e\t3->%1.2e\t4->%1.2e\t5->
    %1.2e\t6->%1.2e\tVCO->
    %1.2e\n",source1.seuil_incertitude,source2.seuil_incertitude,source3.seuil_incertitude,source4.
    seuil_incertitude,source5.seuil_incertitude,source6.seuil_incertitude,vco.seuil_incertitude);
506                     printf("Fréquence moyenne : 1->%1.6e\t2->%1.6e\t3->%1.6e\t4->%1.6e\t5->
    %1.6e\t6->%1.6e\tVCO->
    %1.6e\n",source1.freq_moy_100s,source2.freq_moy_100s,source3.freq_moy_100s,source4.freq_moy_100
    s,source5.freq_moy_100s,source6.freq_moy_100s,vco.freq_moy_100s);
507                 }
508
509             }
510
511  //} Fin boucle d'initialisation
512
513          }else
514
515  //{ Début boucle de mesure/asservissement
516          {
517
518              timestamp = timestamp + nb_ech_reg/100;
519
520              printf("\n#%d\t",timestamp);
521              fprintf(logfile_interne_fp,"%d;",timestamp);
522
523              if(source1.is_present)
524              {
525                  regression_principale(&source1);       // Récupère les informations dans la
    structure
526                  if(source1.var_res_reg_ppl > source1.seuil_incertitude){       // Contrôle de
    la validité de la mesure (variance des résidus)
527                      source1.interval_temps = source1.interval_temps_prec + source1.pente_prec
    * nb_ech_reg; source1.nb_bugs++; source1.nb_bugs_successif++;}
528                  else{ source1.pente_prec = source1.pente; source1.nb_bugs_successif = 0;}
529                  if(( (source1.interval_temps - source1.interval_temps_prec) >
    (source1.moy_accroissement + 1 ) ) || ( (source1.interval_temps - source1.interval_temps_prec)
    < (source1.moy_accroissement - 1 ) )) // Contrôle du recouvrement des signaux
```

```
530                 {
531                     source1.freq_diff = source1.freq_diff_precedent;
532                     source1.nb_recouvrements++;
533                 }else{
534                     source1.freq_diff = (source1.interval_temps - source1.interval_temps_prec)
    * ((double)(10/(nb_ech_reg/100))/(double)freq_rapide);
535                 }
536                 source1.freq_diff_precedent = source1.freq_diff;
537                 source1.interval_temps_prec = source1.interval_temps;
538                 fprintf(logfile_interne_fp,"%e;%e;",source1.interval_temps,source1.freq_diff);
539                 printf("1->%7.0f %1.2e/%1.2e/%d/
    %d",source1.interval_temps,source1.freq_diff,source1.var_res_reg_ppl,source1.nb_bugs,source1.nb
    _recouvrements);
540             }
541
542             if(source2.is_present){ regression_principale(&source2);
543                 if(source2.var_res_reg_ppl > source2.seuil_incertitude)
    { source2.interval_temps = source2.interval_temps_prec + source2.pente_prec * nb_ech_reg;
    source2.nb_bugs++; source2.nb_bugs_successif++;}
544                 else{ source2.pente_prec = source2.pente;  source2.nb_bugs_successif=0;}
545                 if(( (source2.interval_temps - source2.interval_temps_prec) >
    (source2.moy_accroissement + 1 ) ) || ( (source2.interval_temps - source2.interval_temps_prec)
    < (source2.moy_accroissement - 1) )){
546                     source2.freq_diff = source2.freq_diff_precedent; source2.nb_recouvrements+
    +; }
547                 else{
548                 source2.freq_diff = (source2.interval_temps - source2.interval_temps_prec) *
    ((double)(10/(nb_ech_reg/100))/(double)freq_rapide);
549                 }
550                 source2.freq_diff_precedent = source2.freq_diff;
551                 source2.interval_temps_prec = source2.interval_temps;
552                 fprintf(logfile_interne_fp,"%e;%e;",source2.interval_temps,source2.freq_diff);
553                 printf("\t2->%7.0f %1.2e/%1.2e/%d/
    %d",source2.interval_temps,source2.freq_diff,source2.var_res_reg_ppl,source2.nb_bugs,source2.nb
    _recouvrements); }
554
555             if(source3.is_present){ regression_principale(&source3);
556                 if(source3.var_res_reg_ppl > source3.seuil_incertitude)
    { source3.interval_temps = source3.interval_temps_prec + source3.pente_prec * nb_ech_reg;
    source3.nb_bugs++;source3.nb_bugs_successif++;}
557                 else{ source3.pente_prec = source3.pente;  source3.nb_bugs_successif=0;}
558                 if(( (source3.interval_temps - source3.interval_temps_prec) >
    (source3.moy_accroissement + 1 ) ) || ( (source3.interval_temps - source3.interval_temps_prec)
    < (source3.moy_accroissement - 1 ) )){
559                     source3.freq_diff = source3.freq_diff_precedent; source3.nb_recouvrements+
    +; }
560                 else{
561                 source3.freq_diff = (source3.interval_temps - source3.interval_temps_prec) *
    ((double)(10/(nb_ech_reg/100))/(double)freq_rapide);
562                 }
563                 source3.freq_diff_precedent = source3.freq_diff;
564                 source3.interval_temps_prec = source3.interval_temps;
565                 fprintf(logfile_interne_fp,"%e;%e;",source3.interval_temps,source3.freq_diff);
566                 printf("\t3->%7.0f %1.2e/%1.2e/%d/
    %d",source3.interval_temps,source3.freq_diff,source3.var_res_reg_ppl,source3.nb_bugs,source3.nb
    _recouvrements); }
567
568             if(source4.is_present){ regression_principale(&source4);
569                 if(source4.var_res_reg_ppl > source4.seuil_incertitude)
    { source4.interval_temps = source4.interval_temps_prec + source4.pente_prec * nb_ech_reg;
    source4.nb_bugs++;source4.nb_bugs_successif++;}
570                 else{ source4.pente_prec = source4.pente;  source4.nb_bugs_successif=0;}
571                 if(( (source4.interval_temps - source4.interval_temps_prec) >
    (source4.moy_accroissement + 1 ) ) || ( (source4.interval_temps - source4.interval_temps_prec)
    < (source4.moy_accroissement - 1 ) )){
572                     source4.freq_diff = source4.freq_diff_precedent; source4.nb_recouvrements+
    +;
573                 }else{
574                 source4.freq_diff = (source4.interval_temps - source4.interval_temps_prec) *
    ((double)(10/(nb_ech_reg/100))/(double)freq_rapide);
575                 }
576                 source4.freq_diff_precedent = source4.freq_diff;
577                 source4.interval_temps_prec = source4.interval_temps;
578                 fprintf(logfile_interne_fp,"%e;%e;",source4.interval_temps,source4.freq_diff);
579                 printf("\t4->%7.0f %1.2e/%1.2e/%d/
    %d",source4.interval_temps,source4.freq_diff,source4.var_res_reg_ppl,source4.nb_bugs,source4.nb
    _recouvrements); }
580
581             if(source5.is_present){ regression_principale(&source5);
582                 if(source5.var_res_reg_ppl > source5.seuil_incertitude)
```

```
{ source5.interval_temps = source5.interval_temps_prec + source5.pente_prec * nb_ech_reg;
source5.nb_bugs++;source5.nb_bugs_successif++;}
583                 else{ source5.pente_prec = source5.pente; source5.nb_bugs_successif=0; }
584                 if(( (source5.interval_temps - source5.interval_temps_prec) >
(source5.moy_accroissement + 1 ) ) || ( (source5.interval_temps - source5.interval_temps_prec)
< (source5.moy_accroissement - 1 ) )){
585                     source5.freq_diff = source5.freq_diff_precedent; source5.nb_recouvrements+
+;
586                 }else{
587                 source5.freq_diff = (source5.interval_temps - source5.interval_temps_prec) *
((double)(10/(nb_ech_reg/100))/(double)freq_rapide);
588                 }
589                 source5.freq_diff_precedent = source5.freq_diff;
590                 source5.interval_temps_prec = source5.interval_temps;
591                 fprintf(logfile_interne_fp,"%e;%e;",source5.interval_temps,source5.freq_diff);
592                 printf("\t5->%7.0f %1.2e/%1.2e/%d/
%d",source5.interval_temps,source5.freq_diff,source5.var_res_reg_ppl,source5.nb_bugs,source5.nb
_recouvrements); }
593
594             if(source6.is_present){ regression_principale(&source6);
595                 if(source6.var_res_reg_ppl > source6.seuil_incertitude)
{ source6.interval_temps = source6.interval_temps_prec + source6.pente_prec * nb_ech_reg;
source6.nb_bugs++;source6.nb_bugs_successif++;}
596                 else{ source6.pente_prec = source6.pente; source6.nb_bugs_successif=0; }
597                 if(( (source6.interval_temps - source6.interval_temps_prec) >
(source6.moy_accroissement + 1 ) ) || ( (source6.interval_temps - source6.interval_temps_prec)
< (source6.moy_accroissement - 1 ) )){
598                     source6.freq_diff = source6.freq_diff_precedent; source6.nb_recouvrements+
+;
599                 }else{
600                 source6.freq_diff = (source6.interval_temps - source6.interval_temps_prec) *
((double)(10/(nb_ech_reg/100))/(double)freq_rapide);
601                 }
602                 source6.freq_diff_precedent = source6.freq_diff;
603                 source6.interval_temps_prec = source6.interval_temps;
604                 fprintf(logfile_interne_fp,"%e;%e;",source6.interval_temps,source6.freq_diff);
605                 printf("\t6->%7.0f %1.2e/%1.2e/%d/
%d",source6.interval_temps,source6.freq_diff,source6.var_res_reg_ppl,source6.nb_bugs,source6.nb
_recouvrements); }
606
607             if(vco.is_present){ regression_principale(&vco);
608                 if(vco.var_res_reg_ppl > vco.seuil_incertitude){ vco.interval_temps =
vco.interval_temps_prec + vco.pente_prec * nb_ech_reg; vco.nb_bugs++; vco.nb_bugs_successif++;}
609                 else{ vco.pente_prec = vco.pente; vco.nb_bugs_successif=0; }
610                 if(( (vco.interval_temps - vco.interval_temps_prec) > (vco.moy_accroissement +
1 ) ) || ( (vco.interval_temps - vco.interval_temps_prec) < (vco.moy_accroissement - 1 ) )){
611                     vco.freq_diff = vco.freq_diff_precedent; vco.nb_recouvrements++;
612                 }else{
613                 vco.freq_diff = (vco.interval_temps - vco.interval_temps_prec) * ((double)(10/
(nb_ech_reg/100))/(double)freq_rapide);
614                 }
615                 vco.freq_diff_precedent = vco.freq_diff;
616                 vco.interval_temps_prec = vco.interval_temps;
617                 fprintf(logfile_interne_fp,"%e;%e",vco.interval_temps,vco.freq_diff);
618                 printf("\tVCO->%7.0f %1.2e/%1.2e/%d/
%d",vco.interval_temps,vco.freq_diff,vco.var_res_reg_ppl,vco.nb_bugs,vco.nb_recouvrements); }
619
620             printf("\n");
621             fprintf(logfile_interne_fp,"\n");
622
623             moy_accroiss_source1 += source1.freq_diff * ((double)freq_rapide/((double)(10/
(nb_ech_reg/100))));
624             moy_accroiss_source2 += source2.freq_diff * ((double)freq_rapide/((double)(10/
(nb_ech_reg/100))));
625             moy_accroiss_source3 += source3.freq_diff * ((double)freq_rapide/((double)(10/
(nb_ech_reg/100))));
626             moy_accroiss_source4 += source4.freq_diff * ((double)freq_rapide/((double)(10/
(nb_ech_reg/100))));
627             moy_accroiss_source5 += source5.freq_diff * ((double)freq_rapide/((double)(10/
(nb_ech_reg/100))));
628             moy_accroiss_source6 += source6.freq_diff * ((double)freq_rapide/((double)(10/
(nb_ech_reg/100))));
629             moy_accroiss_vco += vco.freq_diff * ((double)freq_rapide/((double)(10/
(nb_ech_reg/100))));
630
631             if(source1.var_res_reg_ppl < source1.seuil_incertitude){ moy_incertitude_source1
+= source1.var_res_reg_ppl; nb_mesures_incert_source1++; }
632             if(source2.var_res_reg_ppl < source2.seuil_incertitude){ moy_incertitude_source2
+= source2.var_res_reg_ppl; nb_mesures_incert_source2++; }
633             if(source3.var_res_reg_ppl < source3.seuil_incertitude){ moy_incertitude_source3
```

```
   += source3.var_res_reg_ppl; nb_mesures_incert_source3++; }
634             if(source4.var_res_reg_ppl < source4.seuil_incertitude){ moy_incertitude_source4
   += source4.var_res_reg_ppl; nb_mesures_incert_source4++; }
635             if(source5.var_res_reg_ppl < source5.seuil_incertitude){ moy_incertitude_source5
   += source5.var_res_reg_ppl; nb_mesures_incert_source5++; }
636             if(source6.var_res_reg_ppl < source6.seuil_incertitude){ moy_incertitude_source6
   += source6.var_res_reg_ppl; nb_mesures_incert_source6++; }
637             if(vco.var_res_reg_ppl < vco.seuil_incertitude){ moy_incertitude_vco +=
   vco.var_res_reg_ppl; nb_mesures_incert_vco++; }
638
639             if( (timestamp % 100 == 0) && (timestamp >= 10) )
640             {
641                 source1.moy_accroissement = moy_accroiss_source1 / (100/(nb_ech_reg/100));
642                 source2.moy_accroissement = moy_accroiss_source2 / (100/(nb_ech_reg/100));
643                 source3.moy_accroissement = moy_accroiss_source3 / (100/(nb_ech_reg/100));
644                 source4.moy_accroissement = moy_accroiss_source4 / (100/(nb_ech_reg/100));
645                 source5.moy_accroissement = moy_accroiss_source5 / (100/(nb_ech_reg/100));
646                 source6.moy_accroissement = moy_accroiss_source6 / (100/(nb_ech_reg/100));
647                 vco.moy_accroissement = moy_accroiss_vco / (100/(nb_ech_reg/100));
648
649                 source1.freq_moy_100s = source1.moy_accroissement * (((double)(10/
   (nb_ech_reg/100)))/(double)freq_rapide);
650                 source2.freq_moy_100s = source2.moy_accroissement * (((double)(10/
   (nb_ech_reg/100)))/(double)freq_rapide);
651                 source3.freq_moy_100s = source3.moy_accroissement * (((double)(10/
   (nb_ech_reg/100)))/(double)freq_rapide);
652                 source4.freq_moy_100s = source4.moy_accroissement * (((double)(10/
   (nb_ech_reg/100)))/(double)freq_rapide);
653                 source5.freq_moy_100s = source5.moy_accroissement * (((double)(10/
   (nb_ech_reg/100)))/(double)freq_rapide);
654                 source6.freq_moy_100s = source6.moy_accroissement * (((double)(10/
   (nb_ech_reg/100)))/(double)freq_rapide);
655                 vco.freq_moy_100s = vco.moy_accroissement * (((double)(10/(nb_ech_reg/100)))/
   (double)freq_rapide);
656
657                 moy_accroiss_source1 = 0;
658                 moy_accroiss_source2 = 0;
659                 moy_accroiss_source3 = 0;
660                 moy_accroiss_source4 = 0;
661                 moy_accroiss_source5 = 0;
662                 moy_accroiss_source6 = 0;
663                 moy_accroiss_vco = 0;
664
665                 moy_incertitude_source1 /= nb_mesures_incert_source1;
666                 moy_incertitude_source2 /= nb_mesures_incert_source2;
667                 moy_incertitude_source3 /= nb_mesures_incert_source3;
668                 moy_incertitude_source4 /= nb_mesures_incert_source4;
669                 moy_incertitude_source5 /= nb_mesures_incert_source5;
670                 moy_incertitude_source6 /= nb_mesures_incert_source6;
671                 moy_incertitude_vco /= nb_mesures_incert_vco;
672
673                 if(moy_incertitude_source1 > 1){ source1.seuil_incertitude = 2 *
   moy_incertitude_source1; }else{ source1.seuil_incertitude = 1; }
674                 if(moy_incertitude_source2 > 1){ source2.seuil_incertitude = 2 *
   moy_incertitude_source2; }else{ source2.seuil_incertitude = 1; }
675                 if(moy_incertitude_source3 > 1){ source3.seuil_incertitude = 2 *
   moy_incertitude_source3; }else{ source3.seuil_incertitude = 1; }
676                 if(moy_incertitude_source4 > 1){ source4.seuil_incertitude = 2 *
   moy_incertitude_source4; }else{ source4.seuil_incertitude = 1; }
677                 if(moy_incertitude_source5 > 1){ source5.seuil_incertitude = 2 *
   moy_incertitude_source5; }else{ source5.seuil_incertitude = 1; }
678                 if(moy_incertitude_source6 > 1){ source6.seuil_incertitude = 2 *
   moy_incertitude_source6; }else{ source6.seuil_incertitude = 1; }
679                 if(moy_incertitude_vco > 1){ vco.seuil_incertitude = 2 *
   moy_incertitude_vco; }else{ vco.seuil_incertitude = 1; }
680
681                 moy_incertitude_source1 = 0;
682                 moy_incertitude_source2 = 0;
683                 moy_incertitude_source3 = 0;
684                 moy_incertitude_source4 = 0;
685                 moy_incertitude_source5 = 0;
686                 moy_incertitude_source6 = 0;
687                 moy_incertitude_vco = 0;
688
689                 nb_mesures_incert_source1 = 0;
690                 nb_mesures_incert_source2 = 0;
691                 nb_mesures_incert_source3 = 0;
692                 nb_mesures_incert_source4 = 0;
693                 nb_mesures_incert_source5 = 0;
694                 nb_mesures_incert_source6 = 0;
```

```
695                 nb_mesures_incert_vco = 0;
696
697                 printf("\tSeuil incertitude : 1->%1.2e\t2->%1.2e\t3->%1.2e\t4->%1.2e\t5->
    %1.2e\t6->%1.2e\tVCO->
    %1.2e\n",source1.seuil_incertitude,source2.seuil_incertitude,source3.seuil_incertitude,source4.
    seuil_incertitude,source5.seuil_incertitude,source6.seuil_incertitude,vco.seuil_incertitude);
698                 printf("\tMoy accroissements : 1->%6.0f\t2->%6.0f\t3->%6.0f\t4->%6.0f\t5->
    %6.0f\t6->%6.0f\tVCO->
    %6.0f\n",source1.moy_accroissement,source2.moy_accroissement,source3.moy_accroissement,source4.
    moy_accroissement,source5.moy_accroissement,source6.moy_accroissement,vco.moy_accroissement);
699                 printf("\tFréquence moyenne : 1->%1.6e\t2->%1.6e\t3->%1.6e\t4->%1.6e\t5->
    %1.6e\t6->%1.6e\tVCO->
    %1.6e\n",source1.freq_moy_100s,source2.freq_moy_100s,source3.freq_moy_100s,source4.freq_moy_100
    s,source5.freq_moy_100s,source6.freq_moy_100s,vco.freq_moy_100s);
700
701                 fflush(logfile_entrees_fp);
702                 fflush(logfile_interne_fp);
703                 fflush(logfile_sorties_fp);
704
705             }
706
707             if(enable_kalman) // appel Kalman
708             {
709
710                 if ((config_bruits_fp = fopen(config_bruits_filename, "r")) != NULL)
711                 {
712                     fscanf(config_bruits_fp,"%lf;%lf",&bruit_process,&bruit_mesure);
713
714                     if(source1.Q != (bruit_process * bruit_process) )
715                     {
716                         source1.Q = bruit_process * bruit_process;
717                         source2.Q = bruit_process * bruit_process;
718                         source3.Q = bruit_process * bruit_process;
719                         source4.Q = bruit_process * bruit_process;
720                         source5.Q = bruit_process * bruit_process;
721                         source6.Q = bruit_process * bruit_process;
722                         printf("\tAjustement manuel du bruit de process :
    %5.5e\n",bruit_process);
723                     }
724
725                     if(source1.R != (bruit_mesure * bruit_mesure) )
726                     {
727                         source1.R = bruit_mesure * bruit_mesure;
728                         source2.R = bruit_mesure * bruit_mesure;
729                         source3.R = bruit_mesure * bruit_mesure;
730                         source4.R = bruit_mesure * bruit_mesure;
731                         source5.R = bruit_mesure * bruit_mesure;
732                         source6.R = bruit_mesure * bruit_mesure;
733                         printf("\tAjustement manuel du bruit de mesure :
    %5.5e\n",bruit_mesure);
734                     }
735
736                     fclose(config_bruits_fp);
737                 }
738
739                 if ((config_gain_fp = fopen(config_gain_filename, "r")) != NULL)
740                 {
741                     fscanf(config_gain_fp,"%lf;%lf;%lf;%lf;%lf;
    %lf",&coeff_gain_source1,&coeff_gain_source2,&coeff_gain_source3,&coeff_gain_source4,&coeff_gai
    n_source5,&coeff_gain_source6);
742
743                     if(source1.coeffGain != coeff_gain_source1)
744                     {
745                         source1.coeffGain = coeff_gain_source1;
746                         printf("\tAjustement manuel des gains de Kalman source 1 :
    %f\n",coeff_gain_source1);
747                     }
748                     if(source2.coeffGain != coeff_gain_source2)
749                     {
750                         source2.coeffGain = coeff_gain_source2;
751                         printf("\tAjustement manuel des gains de Kalman source 2 :
    %f\n",coeff_gain_source2);
752                     }
753                     if(source3.coeffGain != coeff_gain_source3)
754                     {
755                         source3.coeffGain = coeff_gain_source3;
756                         printf("\tAjustement manuel des gains de Kalman source 3 :
    %f\n",coeff_gain_source3);
757                     }
758                     if(source4.coeffGain != coeff_gain_source4)
```

```
759                     {
760                         source4.coeffGain = coeff_gain_source4;
761                         printf("\tAjustement manuel des gains de Kalman source 4 :
    %f\n",coeff_gain_source4);
762                     }
763                     if(source5.coeffGain != coeff_gain_source5)
764                     {
765                         source5.coeffGain = coeff_gain_source5;
766                         printf("\tAjustement manuel des gains de Kalman source 5 :
    %f\n",coeff_gain_source5);
767                     }
768                     if(source6.coeffGain != coeff_gain_source6)
769                     {
770                         source6.coeffGain = coeff_gain_source6;
771                         printf("\tAjustement manuel des gains de Kalman source 6 :
    %f\n",coeff_gain_source6);
772                     }
773 
774                     fclose(config_gain_fp);
775                 }
776 
777 
778                 printf("\t#Kalman#\t");
779                 fprintf(logfile_sorties_fp,"%d;",timestamp);
780 
781                 if(source1.is_present){ kalmanFiltrage(&source1);
782                     printf("SOURCE1 -> %f\t",source1.etat_est);
783                     fprintf(logfile_sorties_fp,"%f;",source1.etat_est);
784                 }
785 
786                 if(source2.is_present){ kalmanFiltrage(&source2);
787                     printf("SOURCE2 -> %f\t",source2.etat_est);
788                     fprintf(logfile_sorties_fp,"%f;",source2.etat_est);
789                 }
790 
791                 if(source3.is_present){ kalmanFiltrage(&source3);
792                     printf("SOURCE3 -> %f\t",source3.etat_est);
793                     fprintf(logfile_sorties_fp,"%f;",source3.etat_est);
794                 }
795 
796                 if(source4.is_present){ kalmanFiltrage(&source4);
797                     printf("SOURCE4 -> %f\t",source4.etat_est);
798                     fprintf(logfile_sorties_fp,"%f;",source4.etat_est);
799                 }
800 
801                 if(source5.is_present){ kalmanFiltrage(&source5);
802                     printf("SOURCE5 -> %f\t",source5.etat_est);
803                     fprintf(logfile_sorties_fp,"%f;",source5.etat_est);
804                 }
805 
806                 if(source6.is_present){ kalmanFiltrage(&source6);
807                     printf("SOURCE6 -> %f\t",source6.etat_est);
808                     fprintf(logfile_sorties_fp,"%f",source6.etat_est);
809                 }
810 
811                 printf("\n");
812 
813                 if(enable_asservissement)   // Asservissement
814                 {
815 
816                     // Récupération du fichier de config (ouverture systématique)
817                     if ((config_vco_offset_fp = fopen(config_vco_offset_filename, "r")) !=
    NULL)
818                     {
819                         fscanf(config_vco_offset_fp,"%lf",&offset_freq_manuel);
820                         if(offset_freq_manuel != offset_freq_manuel_prec )
821                         {
822                             printf("\tAjustement manuel de la fréquence du VCO : offset = %f
    Hz\n",offset_freq_manuel);
823                         }
824                         offset_freq_manuel_prec = offset_freq_manuel;
825                         fclose(config_vco_offset_fp);
826                     }
827 
828                     freq_correction = 0;
829 
830                     if(source1.is_present) { freq_correction += source1.freq_moy_100s_init -
    offset_freq_manuel - source1.etat_est; }
831                     if(source2.is_present) { freq_correction += source2.freq_moy_100s_init -
    offset_freq_manuel - source2.etat_est; }
```

```
832                     if(source3.is_present) { freq_correction += source3.freq_moy_100s_init -
    offset_freq_manuel - source3.etat_est; }
833                     if(source4.is_present) { freq_correction += source4.freq_moy_100s_init -
    offset_freq_manuel - source4.etat_est; }
834                     if(source5.is_present) { freq_correction += source5.freq_moy_100s_init -
    offset_freq_manuel - source5.etat_est; }
835                     if(source6.is_present) { freq_correction += source6.freq_moy_100s_init -
    offset_freq_manuel - source6.etat_est; }
836
837                     freq_correction /= (source1.is_present + source2.is_present +
    source3.is_present + source4.is_present + source5.is_present + source6.is_present);
838                     freq_correction *= (-1);
839                     freq_correction *= (double) sensibilite_VCO;
840                     freq_correction *= (double) eff_commande;
841                     freq_correction = round(freq_correction);
842
843                     if(freq_correction > freq_correction_max){
844                         printf("Bridage correction ! (%f)\n",freq_correction);
845                         freq_correction = freq_correction_max;
846                     }
847                     if(freq_correction < -freq_correction_max){
848                         printf("Bridage correction ! (%f)\n",freq_correction);
849                         freq_correction = -freq_correction_max;
850                     }
851
852                     freq_commande += freq_correction;
853
854                     if(freq_commande <= freq_commande_min){
855                         printf("DEPASSEMENT COMMANDE - !!!!\n");
856                         freq_commande = freq_commande_min;
857                     }
858
859                     if(freq_commande >= freq_commande_max){
860                         printf("DEPASSEMENT COMMANDE + !!!!\n");
861                         freq_commande = freq_commande_max;
862                     }
863
864                     printf("\tAsservissement -> Correction : :%f \t - Commande DAC :
    %d\n",freq_correction,freq_commande);
865                     fprintf(logfile_sorties_fp,"%d\n",freq_commande);
866
867                     // Envoi de la commande au CNA (DAC)
868                     rep = fprintf(fd_dac, "%ld", (long) (freq_commande));
869                     if (rep < 0) printf("ERREUR ENVOI COMMANDE AU CNA !!! %d\n", rep);
870                     fseek(fd_dac, SEEK_SET, 0);
871                 }
872
873                 fprintf(logfile_sorties_fp,"\n");
874
875             }
876         }
877
878  //} Fin boucle de mesure/asservissement
879
880      }
881
882      return NULL;
883  }
884
885  int main(int argc,char** argv)
886  {
887
888      if(argv[1] == NULL)
889      {
890          printf("\nErreur ! Argument non reconnu ! \nUsage : %s [mode d'exécution]\n\tModes
    disponibles :\n\tmesures : [0]\n\tasservissement : [1]\n\n",argv[0]);
891          return 0;
892      }else if( (strcmp(argv[1],"0") != 0)  && (strcmp(argv[1],"1")!=0)  )
893      {
894          printf("\nErreur ! Argument non reconnu ! \nUsage : %s [mode d'exécution]\n\tModes
    disponibles :\n\tmesures : [0]\n\tasservissement : [1]\n\n",argv[0]);
895          return 0;
896      }else
897      {
898          printf("\nMode d'exécution : ");
899          if( (strcmp(argv[1],"0")) == 0 ) printf("mesures\n");
900          if( (strcmp(argv[1],"1")) == 0 ) { printf("asservissement\n"); enable_asservissement =
    1; }
901      }
902
```

```
903 //    enable_asservissement = 1;
904
905 //{ Initialisations (DAC, registres GPIO, interrutions, fichiers de log, Kalman)
906
907     printf("\n");
908     printf("#######################################\n");
909     printf("###           Début prog            ###\n");
910     printf("#######################################\n");
911
912     printf("Début configuration système...\n");
913
914     pthread_t thread;
915     pthread_attr_t attr;
916     pthread_attr_init(&attr);
917     pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
918     pthread_create(&thread, &attr, thread_principal, NULL);
919
920     source1.seuil_incertitude = 1;
921     source2.seuil_incertitude = 1;
922     source3.seuil_incertitude = 1;
923     source4.seuil_incertitude = 1;
924     source5.seuil_incertitude = 1;
925     source6.seuil_incertitude = 1;
926     vco.seuil_incertitude = 1;
927     ref_externe.seuil_incertitude = 1;
928
929     source1.coeff_derive = coeff_derive_source1;
930     source2.coeff_derive = coeff_derive_source2;
931     source3.coeff_derive = coeff_derive_source3;
932     source4.coeff_derive = coeff_derive_source4;
933     source5.coeff_derive = coeff_derive_source5;
934     source6.coeff_derive = coeff_derive_source6;
935
936     source1.coeffGain = coeff_gain_source1;
937     source2.coeffGain = coeff_gain_source2;
938     source3.coeffGain = coeff_gain_source3;
939     source4.coeffGain = coeff_gain_source4;
940     source5.coeffGain = coeff_gain_source5;
941     source6.coeffGain = coeff_gain_source6;
942
943 //{Setup DAC
944
945     printf("Configuration du driver du DAC...\n");
946     int offset, rep;
947     long double scale;
948     char filename[255];
949
950     // * powerup DAC ** /
951     sprintf(filename, "%s/out_voltage_powerdown", IIO_BASE_ADDR);
952     if ((fd_dac = fopen(filename, "w")) == NULL)
953     {
954         printf("erreur d'ouverture de %s\n", filename);
955         return EXIT_FAILURE;
956     }
957     fprintf(fd_dac, "0"); // * disable powerdown ** /
958     fclose(fd_dac);
959
960     // ** file to send value ** /
961     sprintf(filename, "%s/out_voltage0_raw", IIO_BASE_ADDR);
962     if ((fd_dac = fopen(filename, "w")) == NULL)
963     {
964         printf("erreur d'ouverture de %s\n", filename);
965         return EXIT_FAILURE;
966     }
967
968     if (EXIT_FAILURE == getAD5791Params(IIO_BASE_ADDR, &offset, &scale))
969         return EXIT_FAILURE;
970
971
972     rep = fprintf(fd_dac, "%ld", (long)freq_commande);
973     if (rep < 0) printf("Erreur envoi de commande DAC %d\n", rep);
974     fseek(fd_dac, SEEK_SET, 0);
975
976     printf("Configuration du driver du DAC OK\n");
977
978 //}
979
980 //{Création des fichiers de log et de config
981
982     int logfile_res;
```

```
 983
 984      if ((logfile_entrees_fp = fopen(log_entrees_filename, "w")) == NULL)
 985      {
 986          printf("erreur d'ouverture de %s\n", log_entrees_filename);
 987          return EXIT_FAILURE;
 988      }
 989
 990      if ((logfile_interne_fp = fopen(log_interne_filename, "w")) == NULL)
 991      {
 992          printf("erreur d'ouverture de %s\n", log_interne_filename);
 993          return EXIT_FAILURE;
 994      }
 995
 996      if ((logfile_sorties_fp = fopen(log_sorties_filename, "w")) == NULL)
 997      {
 998          printf("erreur d'ouverture de %s\n", log_sorties_filename);
 999          return EXIT_FAILURE;
1000      }
1001
1002      if ((config_vco_offset_fp = fopen(config_vco_offset_filename, "w")) == NULL)  // Si le
     fichier de config d'offset existe il est écrasé, si il n'existe pas il est crée
1003      {
1004          printf("erreur d'ouverture de %s\n", config_vco_offset_filename);
1005          return EXIT_FAILURE;
1006      }else
1007      {
1008          fseek(config_vco_offset_fp, SEEK_SET, 0);
1009          fprintf(config_vco_offset_fp,"0");
1010          fclose(config_vco_offset_fp);
1011      }
1012
1013      if ((config_bruits_fp = fopen(config_bruits_filename, "w")) == NULL)  // Si le fichier de
     config de bruits existe il est écrasé, si il n'existe pas il est crée
1014      {
1015          printf("erreur d'ouverture de %s\n", config_bruits_filename);
1016          return EXIT_FAILURE;
1017      }else
1018      {
1019          fseek(config_bruits_fp, SEEK_SET, 0);
1020          fprintf(config_bruits_fp,"%5.5e;%5.5e",bruit_process,bruit_mesure);
1021          fclose(config_bruits_fp);
1022      }
1023
1024      if ((config_gain_fp = fopen(config_gain_filename, "w")) == NULL)  // Si le fichier de
     config des gains existe il est écrasé, si il n'existe pas il est crée
1025      {
1026          printf("erreur d'ouverture de %s\n", config_gain_filename);
1027          return EXIT_FAILURE;
1028      }else
1029      {
1030          fseek(config_gain_fp, SEEK_SET, 0);
1031          fprintf(config_gain_fp,"%lf;%lf;%lf;%lf;%lf;
     %lf",coeff_gain_source1,coeff_gain_source2,coeff_gain_source3,coeff_gain_source4,coeff_gain_sou
     rce5,coeff_gain_source6);
1032          fclose(config_gain_fp);
1033      }
1034
1035
1036      logfile_res = fprintf(logfile_entrees_fp,"Fichier de log des entrées (valeur brute des
     compteurs, écarts temporels déduits) @ 100hz\n");
1037      if (logfile_res < 0) printf("erreur log file [%d]\n", logfile_res);
1038      logfile_res = fprintf(logfile_entrees_fp,"Inutilisé pour l'instant...\n");
1039      if (logfile_res < 0) printf("erreur log file [%d]\n", logfile_res);
1040
1041      logfile_res = fprintf(logfile_interne_fp,"Fichier de log interne (écarts de temps et de
     fréquence) des sources détectées\n");
1042      if (logfile_res < 0) printf("erreur log file [%d]\n", logfile_res);
1043      logfile_res = fprintf(logfile_interne_fp,"date ; deltaT (en tick cpt) source 1 ; deltaF
     (en Hertz) source 1 ; ...2,3,4,5,6,vco\n");
1044      if (logfile_res < 0) printf("erreur log file [%d]\n", logfile_res);
1045
1046      logfile_res = fprintf(logfile_sorties_fp,"Fichier de log des sorties (réponse du Kalman
     des sources détectées, commande DAC)\n");
1047      if (logfile_res < 0) printf("erreur log file [%d]\n", logfile_res);
1048      logfile_res = fprintf(logfile_sorties_fp,"date ; deltaF Kalman sources 1,2,3,4,5,6 ;
     commande DAC\n");
1049      if (logfile_res < 0) printf("erreur log file [%d]\n", logfile_res);
1050
1051      fflush(logfile_entrees_fp);
1052      fflush(logfile_interne_fp);
```

```
1053      fflush(logfile_sorties_fp);
1054      fflush(config_vco_offset_fp);
1055      fflush(config_bruits_fp);
1056      fflush(config_gain_fp);
1057
1058      printf("Fichiers de log et de config crées.\n");
1059
1060  //}
1061
1062  //{Mapping des registres
1063
1064      printf("Mapping des registres des périphériques...\n");
1065      int devmem_fd = -1;
1066      if((devmem_fd = open("/dev/mem", O_RDWR )) == -1)
1067      {
1068          printf("Erreur lors de l'ouverture de /dev/mem : [%d] !\n",devmem_fd);
1069          return 0;
1070      }
1071
1072      void* map_base_cpt_ref_ext = mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
     devmem_fd, gpio_cpt_ref_ext_addr & ~MAP_MASK);
1073      void* map_base_cpt_source1 = mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
     devmem_fd, gpio_cpt_source1_addr & ~MAP_MASK);
1074      void* map_base_cpt_source2 = mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
     devmem_fd, gpio_cpt_source2_addr & ~MAP_MASK);
1075      void* map_base_cpt_source3 = mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
     devmem_fd, gpio_cpt_source3_addr & ~MAP_MASK);
1076      void* map_base_cpt_source4 = mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
     devmem_fd, gpio_cpt_source4_addr & ~MAP_MASK);
1077      void* map_base_cpt_source5 = mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
     devmem_fd, gpio_cpt_source5_addr & ~MAP_MASK);
1078      void* map_base_cpt_source6 = mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
     devmem_fd, gpio_cpt_source6_addr & ~MAP_MASK);
1079      void* map_base_cpt_vco = mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, devmem_fd,
     gpio_cpt_vco_addr & ~MAP_MASK);
1080      void* map_base_select_ref = mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
     devmem_fd, gpio_select_ref_addr & ~MAP_MASK);
1081
1082      void* virt_addr_read_cpt_ref_ext = map_base_cpt_ref_ext + (gpio_cpt_ref_ext_addr &
     MAP_MASK);
1083      void* virt_addr_read_cpt_source1 = map_base_cpt_source1 + (gpio_cpt_source1_addr &
     MAP_MASK);
1084      void* virt_addr_read_cpt_source2 = map_base_cpt_source2 + (gpio_cpt_source2_addr &
     MAP_MASK);
1085      void* virt_addr_read_cpt_source3 = map_base_cpt_source3 + (gpio_cpt_source3_addr &
     MAP_MASK);
1086      void* virt_addr_read_cpt_source4 = map_base_cpt_source4 + (gpio_cpt_source4_addr &
     MAP_MASK);
1087      void* virt_addr_read_cpt_source5 = map_base_cpt_source5 + (gpio_cpt_source5_addr &
     MAP_MASK);
1088      void* virt_addr_read_cpt_source6 = map_base_cpt_source6 + (gpio_cpt_source6_addr &
     MAP_MASK);
1089      void* virt_addr_read_cpt_vco = map_base_cpt_vco + (gpio_cpt_vco_addr & MAP_MASK);
1090      virt_addr_write_select_ref = map_base_select_ref + (gpio_select_ref_addr & MAP_MASK);
1091
1092      *((int *) virt_addr_write_select_ref) = 0; // La référence externe fournit la référence (à
     priori)
1093
1094      printf("Mapping des registres des périphériques OK\n");
1095
1096  //}
1097
1098  //{Connection avec drivers d'interruptions
1099
1100      int intr_pps_fd;
1101
1102      if((intr_pps_fd = open(intr_driver_name, O_RDONLY )) == -1)
1103      {
1104          printf("Erreur ouverture %s : [%d] !\n",intr_driver_name,intr_pps_fd);
1105          return 0;
1106      }
1107      fd_set rfds_pps;
1108      FD_ZERO(&rfds_pps);
1109      FD_SET(intr_pps_fd,&rfds_pps);
1110
1111  //}
1112
1113  //{Déclaration des variables locales
1114      int i = 0;
1115      int select_retval;      //
```

```
1116      int indice_100hz = 0;   // Indice de remplissage du tableau
1117      struct timeval tv;      // Pour timeout sur la détection de la référence externe
1118      int *waitfor;           // Pour blocage du main en attendant le transfert des données
1119  //}
1120
1121  //{Calcul des constantes pour la regression linéaire
1122
1123      for(i=0; i<nb_ech_reg; i++)
1124      {
1125          moyX += i;
1126      }
1127      moyX /= nb_ech_reg;
1128
1129      for(i=0; i<nb_ech_reg; i++)
1130      {
1131          Sxx += (i - moyX) * (i - moyX);
1132      }
1133      Sxx /= (nb_ech_reg - 1);
1134
1135      printf("Constantes de la régression princiale : nb_ech = %d ; moyX = %f ; Sxx =
     %f\n",nb_ech_reg,moyX,Sxx);
1136  //}
1137
1138  //}
1139
1140      printf("Configuration système terminée.\n");
1141
1142  //{ Détection de la ref externe et/ou du VCO et des sources présentes
1143
1144      tv.tv_sec = 2;
1145      tv.tv_usec = 0;
1146
1147      printf("Attente du signal de référence externe...\n");
1148
1149      select_retval = select(intr_pps_fd+1,&rfds_pps,NULL,NULL,&tv); // Interruption 100Hz
1150
1151      if(select_retval > 0){
1152          printf("Interruption de référence externe détectée.\n");
1153          ref_externe.is_present = 1;
1154      }else{
1155          printf("Interruption de référence absente, changement de ref sur le VCO\n");
1156          FD_SET(intr_pps_fd,&rfds_pps);
1157          *((int *) virt_addr_write_select_ref) = 1;  // Le VCO fournit la référence
1158          tv.tv_sec = 2;
1159          tv.tv_usec = 0;
1160          select_retval = select(intr_pps_fd+1,&rfds_pps,NULL,NULL,&tv);
1161          if(select_retval > 0){
1162              printf("Interruption VCO détectée.\n");
1163              FD_SET(intr_pps_fd,&rfds_pps);
1164              vco.is_present = 1;
1165          }else{
1166              printf("Aucune référence potentielle détectée !!! Abandon.\n");
1167              return 0;
1168          }
1169      }
1170
1171      source1.valeur_brute_cpt = 0;
1172      source2.valeur_brute_cpt = 0;
1173      source3.valeur_brute_cpt = 0;
1174      source4.valeur_brute_cpt = 0;
1175      source5.valeur_brute_cpt = 0;
1176      source6.valeur_brute_cpt = 0;
1177      vco.valeur_brute_cpt = 0;
1178      ref_externe.valeur_brute_cpt = 0;
1179
1180      select(intr_pps_fd+1,&rfds_pps,NULL,NULL,NULL);
1181      select(intr_pps_fd+1,&rfds_pps,NULL,NULL,NULL);
1182      source1.valeur_brute_cpt_prec = (*((uint32_t *) virt_addr_read_cpt_source1)) & 0x00FFFFFF;
1183      source2.valeur_brute_cpt_prec = (*((uint32_t *) virt_addr_read_cpt_source2)) & 0x00FFFFFF;
1184      source3.valeur_brute_cpt_prec = (*((uint32_t *) virt_addr_read_cpt_source3)) & 0x00FFFFFF;
1185      source4.valeur_brute_cpt_prec = (*((uint32_t *) virt_addr_read_cpt_source4)) & 0x00FFFFFF;
1186      source5.valeur_brute_cpt_prec = (*((uint32_t *) virt_addr_read_cpt_source5)) & 0x00FFFFFF;
1187      source6.valeur_brute_cpt_prec = (*((uint32_t *) virt_addr_read_cpt_source6)) & 0x00FFFFFF;
1188      vco.valeur_brute_cpt_prec = (*((uint32_t *) virt_addr_read_cpt_vco)) & 0x00FFFFFF;
1189      ref_externe.valeur_brute_cpt = (*((uint32_t *) virt_addr_read_cpt_ref_ext)) & 0x00FFFFFF;
1190
1191      select(intr_pps_fd+1,&rfds_pps,NULL,NULL,NULL);
1192      source1.valeur_brute_cpt = (*((uint32_t *) virt_addr_read_cpt_source1)) & 0x00FFFFFF;
1193      source2.valeur_brute_cpt = (*((uint32_t *) virt_addr_read_cpt_source2)) & 0x00FFFFFF;
1194      source3.valeur_brute_cpt = (*((uint32_t *) virt_addr_read_cpt_source3)) & 0x00FFFFFF;
```

```
1195      source4.valeur_brute_cpt = (*((uint32_t *) virt_addr_read_cpt_source4)) & 0x00FFFFFF;
1196      source5.valeur_brute_cpt = (*((uint32_t *) virt_addr_read_cpt_source5)) & 0x00FFFFFF;
1197      source6.valeur_brute_cpt = (*((uint32_t *) virt_addr_read_cpt_source6)) & 0x00FFFFFF;
1198      vco.valeur_brute_cpt = (*((uint32_t *) virt_addr_read_cpt_vco)) & 0x00FFFFFF;
1199      ref_externe.valeur_brute_cpt = (*((uint32_t *) virt_addr_read_cpt_ref_ext)) & 0x00FFFFFF;
1200
1201      if(source1.valeur_brute_cpt != source1.valeur_brute_cpt_prec){ printf("Source 1
    détectée.\n"); source1.is_present = 1;}else{ printf("Source 1 non détectée.\n");
    source1.is_present = 0;}
1202      if(source2.valeur_brute_cpt != source2.valeur_brute_cpt_prec){ printf("Source 2
    détectée.\n"); source2.is_present = 1;}else{ printf("Source 2 non détectée.\n");
    source2.is_present = 0;}
1203      if(source3.valeur_brute_cpt != source3.valeur_brute_cpt_prec){ printf("Source 3
    détectée.\n"); source3.is_present = 1;}else{ printf("Source 3 non détectée.\n");
    source3.is_present = 0;}
1204      if(source4.valeur_brute_cpt != source4.valeur_brute_cpt_prec){ printf("Source 4
    détectée.\n"); source4.is_present = 1;}else{ printf("Source 4 non détectée.\n");
    source4.is_present = 0;}
1205      if(source5.valeur_brute_cpt != source5.valeur_brute_cpt_prec){ printf("Source 5
    détectée.\n"); source5.is_present = 1;}else{ printf("Source 5 non détectée.\n");
    source5.is_present = 0;}
1206      if(source6.valeur_brute_cpt != source6.valeur_brute_cpt_prec){ printf("Source 6
    détectée.\n"); source6.is_present = 1;}else{ printf("Source 6 non détectée.\n");
    source6.is_present = 0;}
1207      if(vco.valeur_brute_cpt != vco.valeur_brute_cpt_prec){ printf("VCO détecté.\n");
    vco.is_present = 1;}else{ printf("VCO non détecté !!! Asservissement impossible !!!\n");
    vco.is_present = 0;}
1208
1209      // Présence des sources pour le script octave de visualisation des données
1210      fprintf(logfile_entrees_fp,"%d;%d;%d;%d;%d;%d;%d;
    %d\n",source1.is_present,source2.is_present,source3.is_present,source4.is_present,source5.is_pr
    esent,source6.is_present,vco.is_present,ref_externe.is_present);
1211      fprintf(logfile_interne_fp,"%d;%d;%d;%d;%d;%d;%d;
    %d\n",source1.is_present,source2.is_present,source3.is_present,source4.is_present,source5.is_pr
    esent,source6.is_present,vco.is_present,ref_externe.is_present);
1212
1213  //} Fin détection de la ref externe et/ou du VCO et des sources présentes
1214
1215      select(intr_pps_fd+1,&rfds_pps,NULL,NULL,NULL);
1216
1217          printf("Mesures des sources et VCO / référence\n");
1218
1219          while(1)
1220          {
1221              select(intr_pps_fd+1,&rfds_pps,NULL,NULL,NULL); // Interruption 100Hz (toutes les
    10 millisecondes)
1222              ref_externe.valeur_brute_cpt = (*((uint32_t *) virt_addr_read_cpt_ref_ext)) &
    0x00FFFFFF;   // Masque 0x00FFFFFF (24bits) pour éviter d'éventuels problèmes lors de la
    conversion en int32
1223              if(source1.is_present) source1.valeur_brute_cpt = (*((uint32_t *)
    virt_addr_read_cpt_source1)) & 0x00FFFFFF;
1224              if(source2.is_present) source2.valeur_brute_cpt = (*((uint32_t *)
    virt_addr_read_cpt_source2)) & 0x00FFFFFF;
1225              if(source3.is_present) source3.valeur_brute_cpt = (*((uint32_t *)
    virt_addr_read_cpt_source3)) & 0x00FFFFFF;
1226              if(source4.is_present) source4.valeur_brute_cpt = (*((uint32_t *)
    virt_addr_read_cpt_source4)) & 0x00FFFFFF;
1227              if(source5.is_present) source5.valeur_brute_cpt = (*((uint32_t *)
    virt_addr_read_cpt_source5)) & 0x00FFFFFF;
1228              if(source6.is_present) source6.valeur_brute_cpt = (*((uint32_t *)
    virt_addr_read_cpt_source6)) & 0x00FFFFFF;
1229              if(vco.is_present) vco.valeur_brute_cpt = (*((uint32_t *) virt_addr_read_cpt_vco))
    & 0x00FFFFFF;
1230
1231  //{ Contrôle des rollovers du compteur et déduction des intervalles de temps
1232
1233              if(source1.is_present){ if(ref_externe.valeur_brute_cpt >=
    source1.valeur_brute_cpt){ source1.interval_100Hz = ref_externe.valeur_brute_cpt -
    source1.valeur_brute_cpt; }
1234                  else{ source1.interval_100Hz = ref_externe.valeur_brute_cpt -
    source1.valeur_brute_cpt + profondeur_compteur_rapide; }
1235                  source1.tab_cpt_rapide_temp[indice_100hz] = (double) (source1.interval_100Hz);
1236                  }
1237
1238              if(source2.is_present){ if(ref_externe.valeur_brute_cpt >=
    source2.valeur_brute_cpt){ source2.interval_100Hz = ref_externe.valeur_brute_cpt -
    source2.valeur_brute_cpt; }
1239                  else{ source2.interval_100Hz = ref_externe.valeur_brute_cpt -
    source2.valeur_brute_cpt + profondeur_compteur_rapide; }
1240                  source2.tab_cpt_rapide_temp[indice_100hz] = (double) (source2.interval_100Hz);
```

```
1241                 }
1242
1243             if(source3.is_present){ if(ref_externe.valeur_brute_cpt >=
     source3.valeur_brute_cpt){ source3.interval_100Hz = ref_externe.valeur_brute_cpt -
     source3.valeur_brute_cpt; }
1244                 else{ source3.interval_100Hz = ref_externe.valeur_brute_cpt -
     source3.valeur_brute_cpt + profondeur_compteur_rapide; }
1245                 source3.tab_cpt_rapide_temp[indice_100hz] = (double) (source3.interval_100Hz);
1246                 }
1247
1248             if(source4.is_present){ if(ref_externe.valeur_brute_cpt >=
     source4.valeur_brute_cpt){ source4.interval_100Hz = ref_externe.valeur_brute_cpt -
     source4.valeur_brute_cpt; }
1249                 else{ source4.interval_100Hz = ref_externe.valeur_brute_cpt -
     source4.valeur_brute_cpt + profondeur_compteur_rapide; }
1250                 source4.tab_cpt_rapide_temp[indice_100hz] = (double) (source4.interval_100Hz);
1251                 }
1252
1253             if(source5.is_present){ if(ref_externe.valeur_brute_cpt >=
     source5.valeur_brute_cpt){ source5.interval_100Hz = ref_externe.valeur_brute_cpt -
     source5.valeur_brute_cpt; }
1254                 else{ source5.interval_100Hz = ref_externe.valeur_brute_cpt -
     source5.valeur_brute_cpt + profondeur_compteur_rapide; }
1255                 source5.tab_cpt_rapide_temp[indice_100hz] = (double) (source5.interval_100Hz);
1256                 }
1257
1258             if(source6.is_present){ if(ref_externe.valeur_brute_cpt >=
     source6.valeur_brute_cpt){ source6.interval_100Hz = ref_externe.valeur_brute_cpt -
     source6.valeur_brute_cpt; }
1259                 else{ source6.interval_100Hz = ref_externe.valeur_brute_cpt -
     source6.valeur_brute_cpt + profondeur_compteur_rapide; }
1260                 source6.tab_cpt_rapide_temp[indice_100hz] = (double) (source6.interval_100Hz);
1261                 }
1262
1263             if(vco.is_present){ if(ref_externe.valeur_brute_cpt >= vco.valeur_brute_cpt)
     { vco.interval_100Hz = ref_externe.valeur_brute_cpt - vco.valeur_brute_cpt; }
1264                 else{ vco.interval_100Hz = ref_externe.valeur_brute_cpt - vco.valeur_brute_cpt
     + profondeur_compteur_rapide; }
1265                 vco.tab_cpt_rapide_temp[indice_100hz] = (double) (vco.interval_100Hz);
1266                 }
1267
1268  //} Fin de contrôle des rollovers
1269
1270             ++indice_100hz;
1271
1272  //{Appel sécurisé au processus principal
1273
1274             if(indice_100hz == nb_ech_reg)
1275             {
1276                 if (pthread_mutex_lock(&__LOCK__)) { abort(); }
1277
1278                 launch_thread_principal = 1;
1279                 waitfor = &launch_thread_principal;
1280
1281                 if (pthread_cond_broadcast(&__COND__)) { abort(); }
1282                 while (*waitfor == 1) { if (pthread_cond_wait(&__COND__, &__LOCK__))
     { abort(); } }
1283
1284                 if (pthread_mutex_unlock(&__LOCK__)) { abort(); }
1285
1286                 indice_100hz = 0;
1287             }
1288  //}
1289
1290
1291          }
1292
1293      return 1;
1294  }
1295
1296  int regression_principale(struct horloge* temp_horloge)
1297  {
1298
1299      int i;
1300
1301      double moyY_horloge = 0;
1302      double Sxy_horloge = 0;
1303
1304      double res_horloge = 0;
1305      double var_res_horloge = 0;
```

```
1306
1307      for(i=0;i<nb_ech_reg;i++)
1308      {
1309          moyY_horloge += (*temp_horloge).tab_cpt_rapide[i];
1310      }
1311      moyY_horloge /= (double) (nb_ech_reg);
1312
1313      for(i=0;i<nb_ech_reg;i++)
1314      {
1315          Sxy_horloge += (((double) i) - moyX) * ( (*temp_horloge).tab_cpt_rapide[i] -
    moyY_horloge);
1316      }
1317      Sxy_horloge /= (double) (nb_ech_reg - 1);
1318
1319      (*temp_horloge).pente = Sxy_horloge / Sxx;
1320
1321      (*temp_horloge).interval_temps = moyY_horloge;
1322
1323      (*temp_horloge).ordon_origin = moyY_horloge - (*temp_horloge).pente * moyX;
1324
1325
1326      // ********************** Calcul des résidus ******************** //
1327      for(i=0; i<nb_ech_reg; i++)
1328      {
1329          res_horloge = (*temp_horloge).tab_cpt_rapide[i] - ( (*temp_horloge).ordon_origin +
    (*temp_horloge).pente * (double) (i) );
1330          var_res_horloge += res_horloge * res_horloge;
1331      }
1332      var_res_horloge /= (double) (nb_ech_reg - 2);
1333
1334      (*temp_horloge).var_res_reg_ppl = var_res_horloge;
1335
1336      return 1;
1337  }
1338
1339  void kalmanFiltrage(struct horloge* horloge_kalman)
1340  {
1341
1342  //#### etat_est_priori = F * etat_est
1343
1344      (*horloge_kalman).etat_pred = ( ( 1 + (*horloge_kalman).coeff_derive ) *
    (*horloge_kalman).etat_est ) - freq_correction / sensibilite_VCO;
1345
1346  //#### Ppriori = F * P * Ftranspo + Q
1347
1348      (*horloge_kalman).P_pred = ( ( 1 + (*horloge_kalman).coeff_derive ) *
    ( (*horloge_kalman).P_est ) * ( 1 + (*horloge_kalman).coeff_derive ) ) + (*horloge_kalman).Q;
1349
1350  //#### Mesures
1351
1352      (*horloge_kalman).etat_mes = (*horloge_kalman).freq_diff;
1353
1354  //#### K = Ppriori * Htranspo * invS * coefgain
1355
1356      (*horloge_kalman).K = (*horloge_kalman).P_pred * ( 1 / ( (*horloge_kalman).P_pred +
    (*horloge_kalman).R ) ) * (*horloge_kalman).coeffGain;
1357
1358  //#### etat_est = etat_est_priori + K * innov
1359
1360      (*horloge_kalman).etat_est = (*horloge_kalman).etat_pred + (*horloge_kalman).K *
    ( (*horloge_kalman).etat_mes - (*horloge_kalman).etat_pred );
1361
1362  //#### P = (Id - K * H) * Ppriori * (Id - K * H)' + K * R * Ktranspo
1363
1364      (*horloge_kalman).P_est = ( 1 - (*horloge_kalman).K ) * (*horloge_kalman).P_pred * ( 1 -
    (*horloge_kalman).K ) + (*horloge_kalman).K * (*horloge_kalman).R * (*horloge_kalman).K;
1365
1366  }
1367
1368  int getAD5791Params(char *basename, int *offset, long double *scale)
1369  {
1370      char filename[256];
1371      FILE *fd;
1372
1373      // * get offset
1374      sprintf(filename, "%s/out_voltage_offset", basename);
1375      if ((fd = fopen(filename, "r")) == NULL)
1376      {
1377          printf("erreur d'ouverture de %s\n", filename);
1378          return EXIT_FAILURE;
```

```
1379      }
1380      fscanf(fd, "%d", offset);
1381      fclose(fd);
1382
1383      // * get vfsr
1384      sprintf(filename, "%s/out_voltage_scale", basename);
1385      if ((fd = fopen(filename, "r")) == NULL)
1386      {
1387          printf("erreur d'ouverture de %s\n", filename);
1388          return EXIT_FAILURE;
1389      }
1390      fscanf(fd, "%Lf", scale);
1391      fclose(fd);
1392
1393      printf("Config DAC : offset=%d    scale=%Lf\n", *offset, *scale);
1394
1395      return EXIT_SUCCESS;
1396  }
1397
1398  void init_kalman(struct horloge* horloge_init)
1399  {
1400      (*horloge_init).etat_mes = (*horloge_init).freq_moy_100s;
1401      (*horloge_init).etat_pred = (*horloge_init).freq_moy_100s;
1402      (*horloge_init).etat_est = (*horloge_init).freq_moy_100s;
1403      (*horloge_init).P_pred = 0;
1404      (*horloge_init).P_est = 0;
1405      (*horloge_init).K = 0;
1406      (*horloge_init).Q = bruit_process * bruit_process;
1407      (*horloge_init).R = bruit_mesure * bruit_mesure;
1408  }
1409
1410
1411
1412
1413
```

14.3 Codes Octave pour fichiers de log

```
printf("\n#### Script d'analyse du fichier de log CLEAR pour ZedBoard ####\n\n")
format long
clear all;
cla reset;
clf reset;


logEntreesFilename = "/usr/local/export/zedboard/log/log_entrees.txt";
logInterneFilename = "/usr/local/export/zedboard/log/log_interne.txt";
logSortiesFilename = "/usr/local/export/zedboard/log/log_sorties.txt";
logStabFilename = "/usr/local/export/zedboard/log/log_stab.txt";

################################## Affichage log ########################################

printf("Ouverture de %s\n",logInterneFilename)
entete = dlmread(logInterneFilename,";",[2,0,2,7]);
data_interne = dlmread(logInterneFilename,";",4,0);
taille_data_interne = size(data_interne,1);
data_interne(taille_data_interne,:) = [];


if(taille_data_interne > 0)
    printf("%d données internes récupérées\n",taille_data_interne);
    t_interne = data_interne(1:taille_data_interne-1,1);
else
    printf("Aucunes données internes !\n");
endif

source1_present = entete(1);
source2_present = entete(2);
source3_present = entete(3);
source4_present = entete(4);
source5_present = entete(5);
source6_present = entete(6);
vco_present = entete(7);
ref_present = entete(8);

nb_sources_present = entete(1)+entete(2)+entete(3)+entete(4)+entete(5)+entete(6)+entete(8)

figure(1,'units','normalized','outerposition',[0 0 1 1],"name","Ecarts de phase et fréquence");
clf;

plot_column = 1;
file_column = 2;

if(source1_present)
    subplot (2, nb_sources_present, plot_column)
    plot(t_interne,data_interne(:,file_column))
    xlabel("TEMPS (S)")
    ylabel("dT SOURCE 1 @ 1s")
    grid
    subplot (2, nb_sources_present, plot_column + nb_sources_present)
    hold("on");
    plot(t_interne,data_interne(:,file_column + 1))
        #if(data_sortie_present) plot(data_sortie(:,1),"r"); endif
    hold("off");
    xlabel("TEMPS (S)")
    ylabel("dF SOURCE_1 @ 1s")
    grid
    plot_column = plot_column + 1;
    file_column = file_column + 2;
endif

if(source2_present)
    subplot (2, nb_sources_present, plot_column)
    plot(t_interne,data_interne(:,file_column))
    xlabel("TEMPS (S)")
    ylabel("dT SOURCE 2 @ 1s")
    grid
    subplot (2, nb_sources_present, plot_column + nb_sources_present)
    hold("on");
    plot(t_interne,data_interne(:,file_column + 1))
        #if(data_sortie_present) plot(data_sortie(:,2),"r"); endif
    hold("off");
    xlabel("TEMPS (S)")
    ylabel("dF SOURCE 2 @ 1s")
    grid
    plot_column = plot_column + 1;
    file_column = file_column + 2;
endif
```

```
 81
 82  if(source3_present)
 83      subplot (2, nb_sources_present, plot_column)
 84      plot(t_interne,data_interne(:,file_column))
 85      xlabel("TEMPS (S)")
 86      ylabel("dT SOURCE 3 @ 1s")
 87      grid
 88      subplot (2, nb_sources_present, plot_column + nb_sources_present)
 89      hold("on");
 90      plot(t_interne,data_interne(:,file_column + 1))
 91          #if(data_sortie_present) plot(data_sortie(:,3),"r"); endif
 92      hold("off");
 93      xlabel("TEMPS (S)")
 94      ylabel("dF SOURCE 3 @ 1s")
 95      grid
 96      plot_column = plot_column + 1;
 97      file_column = file_column + 2;
 98  endif
 99
100  if(source4_present)
101      subplot (2, nb_sources_present, plot_column)
102      plot(t_interne,data_interne(:,file_column))
103      xlabel("TEMPS (S)")
104      ylabel("dT SOURCE 4 @ 1s")
105      grid
106      subplot (2, nb_sources_present, plot_column + nb_sources_present)
107      hold("on");
108      plot(t_interne,data_interne(:,file_column + 1))
109          #if(data_sortie_present) plot(data_sortie(:,4),"r"); endif
110      hold("off");
111      xlabel("TEMPS (S)")
112      ylabel("dF SOURCE 4 @ 1s")
113      grid
114      plot_column = plot_column + 1;
115      file_column = file_column + 2;
116  endif
117
118  if(source5_present)
119      subplot (2, nb_sources_present, plot_column)
120      plot(t_interne,data_interne(:,file_column))
121      xlabel("TEMPS (S)")
122      ylabel("dT SOURCE 5 @ 1s")
123      grid
124      subplot (2, nb_sources_present, plot_column + nb_sources_present)
125      hold("on");
126      plot(t_interne,data_interne(:,file_column + 1))
127          #if(data_sortie_present) plot(data_sortie(:,5),"r"); endif
128      hold("off");
129      xlabel("TEMPS (S)")
130      ylabel("dF SOURCE 5 @ 1s")
131      grid
132      plot_column = plot_column + 1;
133      file_column = file_column + 2;
134  endif
135
136  if(source6_present)
137      subplot (2, nb_sources_present, plot_column)
138      plot(t_interne,data_interne(:,file_column))
139      xlabel("TEMPS (S)")
140      ylabel("dT SOURCE 6 @ 1s")
141      grid
142      subplot (2, nb_sources_present, plot_column + nb_sources_present)
143      hold("on");
144      plot(t_interne,data_interne(:,file_column + 1))
145          #if(data_sortie_present) plot(data_sortie(:,6),"r"); endif
146      hold("off");
147      xlabel("TEMPS (S)")
148      ylabel("dF SOURCE 6 @ 1s")
149      grid
150      plot_column = plot_column + 1;
151      file_column = file_column + 2;
152  endif
153
154  if(vco_present & ref_present)
155      subplot (2, nb_sources_present, plot_column)
156      plot(t_interne,data_interne(:,file_column))
157      xlabel("TEMPS (S)")
158      ylabel("dT vco @ 1s")
159      grid
160      subplot (2, nb_sources_present, plot_column + nb_sources_present)
```

```
161      hold("on");
162      plot(t_interne,data_interne(:,file_column + 1))
163          #if(data_sortie_present) plot(data_sortie(:,7),"r"); endif
164      hold("off");
165      xlabel("TEMPS (S)")
166      ylabel("dF vco @ 1s")
167      grid
168      plot_column = plot_column + 1;
169      file_column = file_column + 2;
170  endif
171
172
173  ################################## Affichage log sorties
     ########################################
174
175  printf("Ouverture de %s\n",logSortiesFilename)
176  data_sortie = dlmread(logSortiesFilename,";",3,0);
177  taille_data_sortie = size(data_sortie,1);
178
179  if(taille_data_sortie > 0)
180      printf("%d données sorties récupérées\n",taille_data_sortie);
181      t_sorties = data_sortie(:,1);
182      data_sortie_present = 1;
183      plot_column = 1;
184      file_column = 2;
185
186      if(source1_present)
187          subplot (2, nb_sources_present, plot_column + nb_sources_present)
188          hold("on");
189          if(data_sortie_present) plot(t_sorties,data_sortie(:,file_column),"r"); endif
190          hold("off");
191          plot_column = plot_column + 1;
192          file_column = file_column + 1;
193      endif
194
195      if(source2_present)
196          subplot (2, nb_sources_present, plot_column + nb_sources_present)
197          hold("on");
198          if(data_sortie_present) plot(t_sorties,data_sortie(:,file_column),"r"); endif
199          hold("off");
200          plot_column = plot_column + 1;
201          file_column = file_column + 1;
202      endif
203
204      if(source3_present)
205          subplot (2, nb_sources_present, plot_column + nb_sources_present)
206          hold("on");
207          if(data_sortie_present) plot(t_sorties,data_sortie(:,file_column),"r"); endif
208          hold("off");
209          plot_column = plot_column + 1;
210          file_column = file_column + 1;
211      endif
212
213      if(source4_present)
214          subplot (2, nb_sources_present, plot_column + nb_sources_present)
215          hold("on");
216          if(data_sortie_present) plot(t_sorties,data_sortie(:,file_column),"r"); endif
217          hold("off");
218          plot_column = plot_column + 1;
219          file_column = file_column + 1;
220      endif
221
222      if(source5_present)
223          subplot (2, nb_sources_present, plot_column + nb_sources_present)
224          hold("on");
225          if(data_sortie_present) plot(t_sorties,data_sortie(:,file_column),"r"); endif
226          hold("off");
227          plot_column = plot_column + 1;
228          file_column = file_column + 1;
229      endif
230
231      if(source6_present)
232          subplot (2, nb_sources_present, plot_column + nb_sources_present)
233          hold("on");
234          if(data_sortie_present) plot(t_sorties,data_sortie(:,file_column),"r"); endif
235          hold("off");
236          plot_column = plot_column + 1;
237          file_column = file_column + 1;
238      endif
239
```

```
240      figure(4,"name","Commande du VCO");
241      plot(t_sorties,data_sortie(:,file_column))
242      xlabel("TEMPS (S)")
243      ylabel("Commande VCO")
244      grid
245
246  else
247      printf("Aucunes données de sorties !\n");
248      data_sortie_present = 0;
249  endif
250
251
252
253
254
255  ################################## Affichage log stabilité
     ##########################################
256
257  printf("Ouverture de %s\n",logStabFilename)
258  data_stab = dlmread(logStabFilename,";",2,0);
259  taille_data_stab = size(data_stab,1);
260
261  if(taille_data_stab > 0)
262      printf("%d données stab récupérées\n",taille_data_stab);
263      data_stab_present = 1;
264  else
265      printf("Aucunes données de stabilité !\n");
266      data_stab_present = 0;
267  endif
268
269  if(data_stab_present)
270
271      tau = data_stab(:,1);
272
273      file_column = 2;
274
275      figure(2,"name","Allan DEV");
276
277      clf();
278
279      hold("on");
280
281      if(source1_present)
282          loglog(tau,data_stab(:,file_column),"-+r;SOURCE 1;");
283          file_column = file_column + 2;
284      endif
285
286      if(source2_present)
287          loglog(tau,data_stab(:,file_column),"-+g;SOURCE 2;");
288          file_column = file_column + 2;
289      endif
290
291      if(source3_present)
292          loglog(tau,data_stab(:,file_column),"-+b;SOURCE 3;");
293          file_column = file_column + 2;
294      endif
295
296      if(source4_present)
297          loglog(tau,data_stab(:,file_column),"-sr;SOURCE 4;");
298          file_column = file_column + 2;
299      endif
300
301      if(source5_present)
302          loglog(tau,data_stab(:,file_column),"-sg;SOURCE 5;");
303          file_column = file_column + 2;
304      endif
305
306      if(source6_present)
307          loglog(tau,data_stab(:,file_column),"-sb;SOURCE 6;");
308          file_column = file_column + 2;
309      endif
310
311      if(vco_present)
312          loglog(tau,data_stab(:,file_column),"-k;VCO;");
313          file_column = file_column + 2;
314      endif
315
316      hold("off");
317
318      grid
```

```
319
320     ############ PDEV
321
322     file_column = 3;
323
324     figure(3,"name","Parabolic DEV");
325
326     clf();
327
328     hold("on");
329
330     if(source1_present)
331         loglog(tau,data_stab(:,file_column),"-+r;SOURCE 1;");
332         file_column = file_column + 2;
333     endif
334
335     if(source2_present)
336         loglog(tau,data_stab(:,file_column),"-+g;SOURCE 2;");
337         file_column = file_column + 2;
338     endif
339
340     if(source3_present)
341         loglog(tau,data_stab(:,file_column),"-+b;SOURCE 3;");
342         file_column = file_column + 2;
343     endif
344
345     if(source4_present)
346         loglog(tau,data_stab(:,file_column),"-sr;SOURCE 4;");
347         file_column = file_column + 2;
348     endif
349
350     if(source5_present)
351         loglog(tau,data_stab(:,file_column),"-sg;SOURCE 5;");
352         file_column = file_column + 2;
353     endif
354
355     if(source6_present)
356         loglog(tau,data_stab(:,file_column),"-sb;SOURCE 6;");
357         file_column = file_column + 2;
358     endif
359
360     if(vco_present)
361         loglog(tau,data_stab(:,file_column),"-k;VCO;");
362         file_column = file_column + 2;
363     endif
364
365     hold("off");
366
367     grid
368
369 endif
370
371
372
373
374
```

Chapitre 15

Mesures

15.1 SINITER sur RedPitaya

Voici les relevés bruts de résultats du SINITER

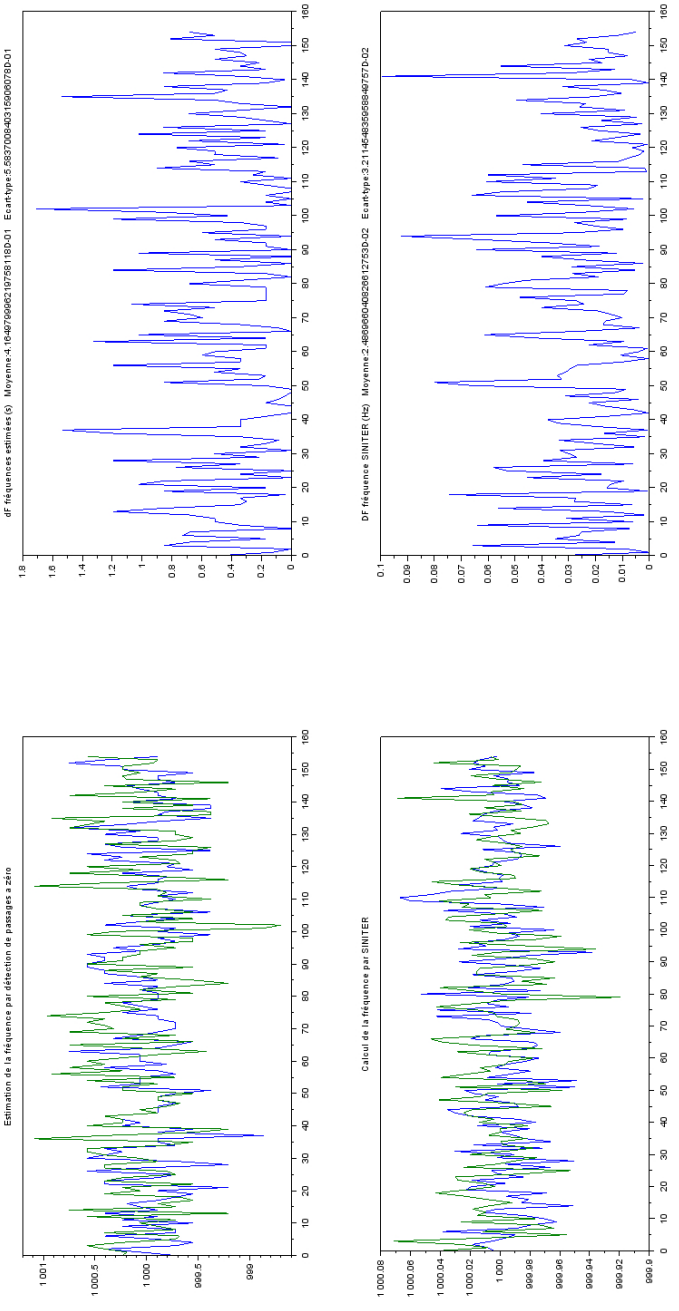


FIGURE 15.1 – Résultats SINITER @ 1kHz déc=64

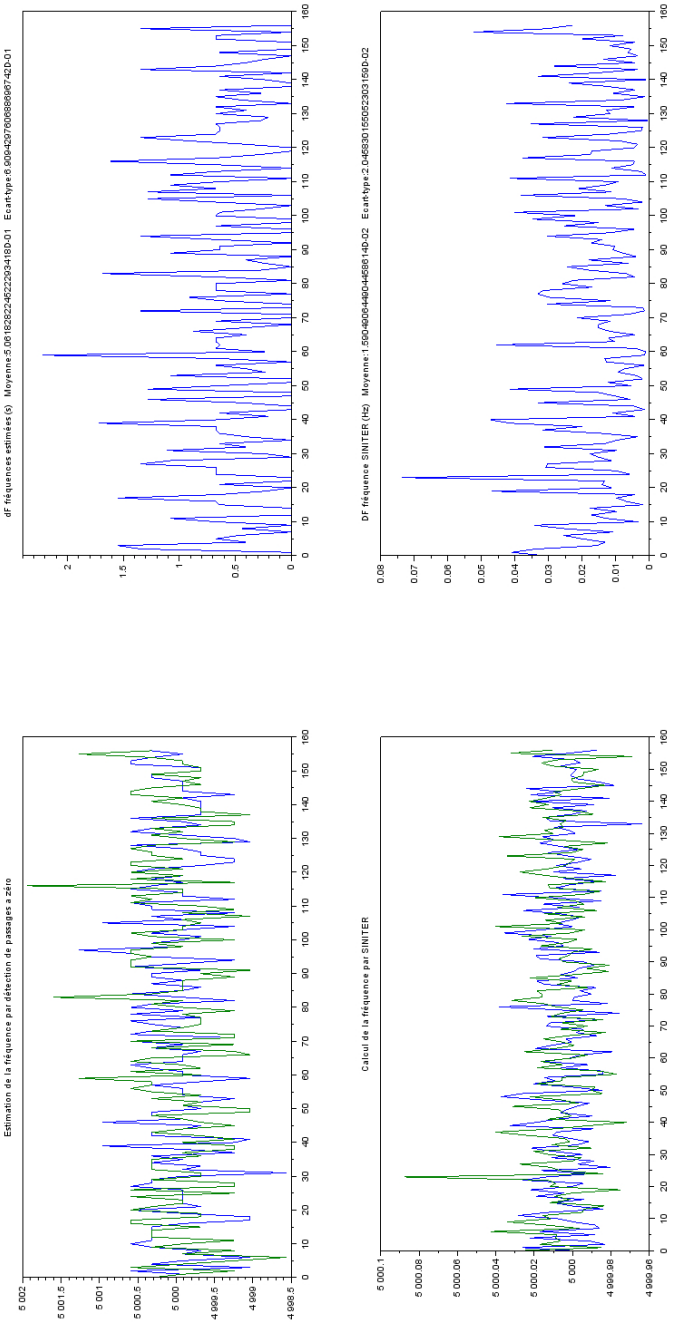


FIGURE 15.2 – Résultats SINITER @ 5kHz déc=64

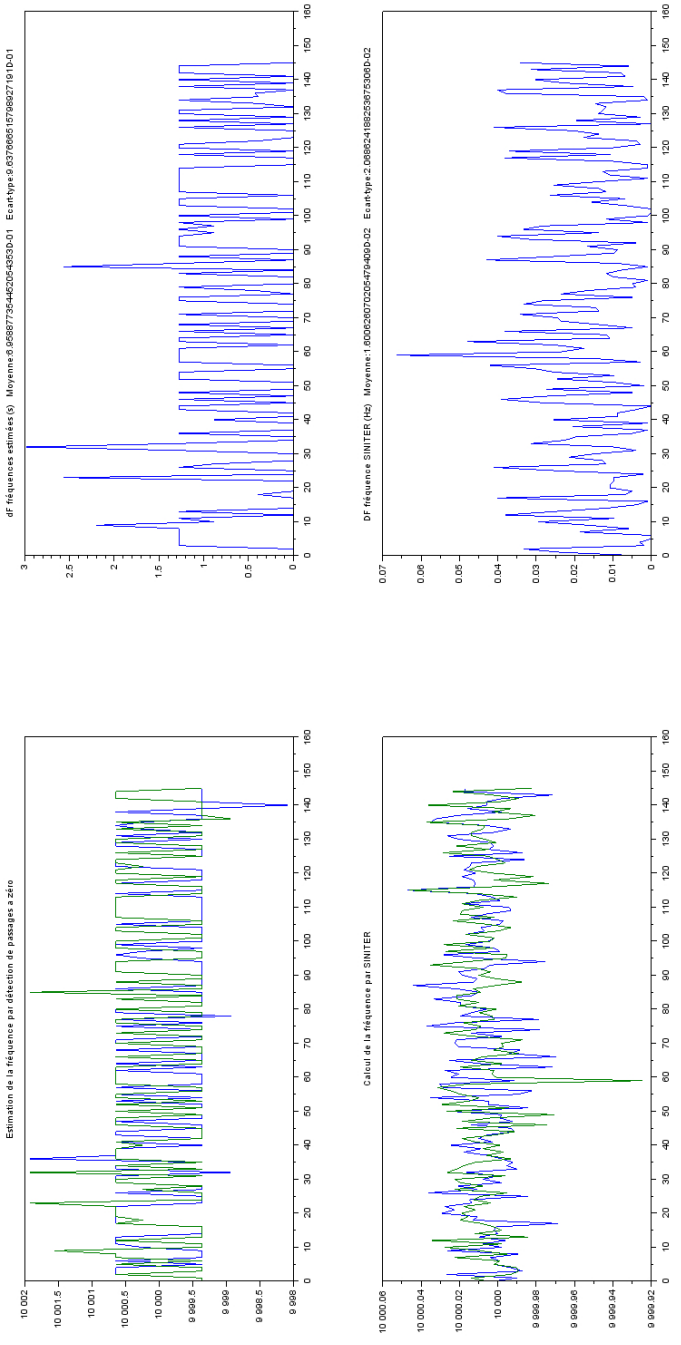


FIGURE 15.3 – Résultats SINITER @ 10kHz déc=64

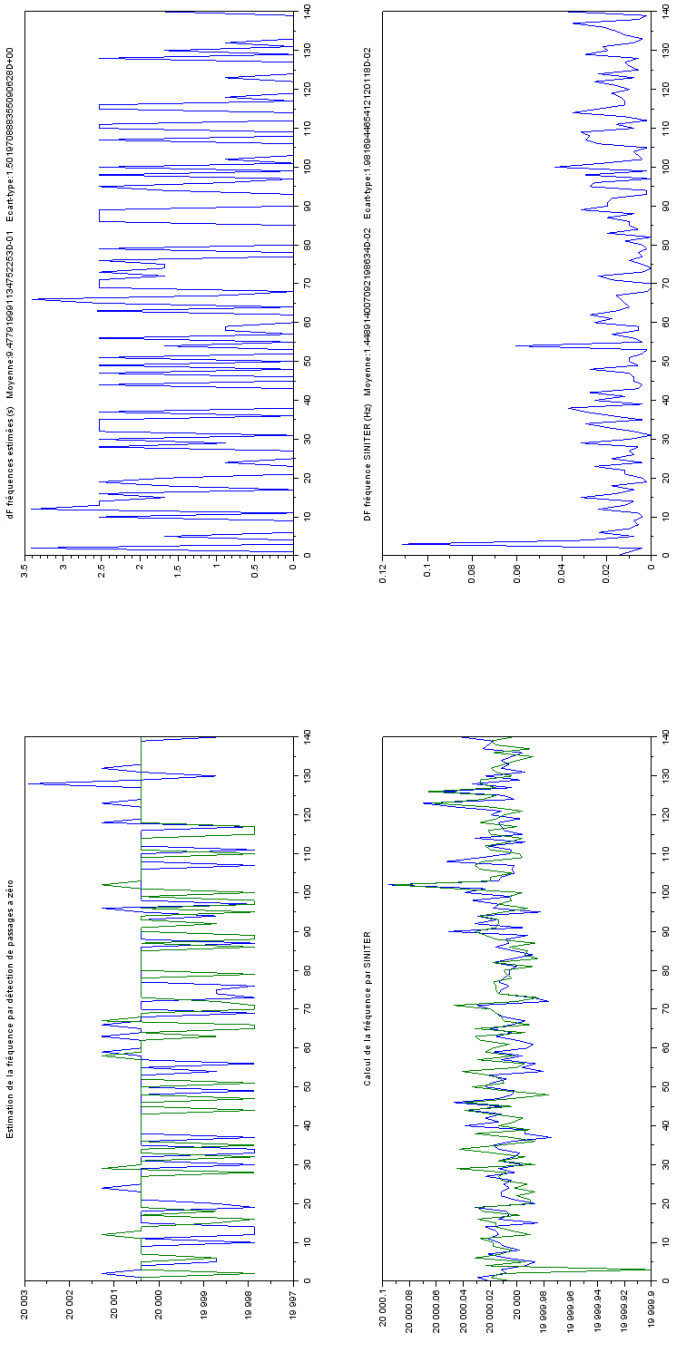


FIGURE 15.4 – Résultats SINITER @ 20kHz déc=64

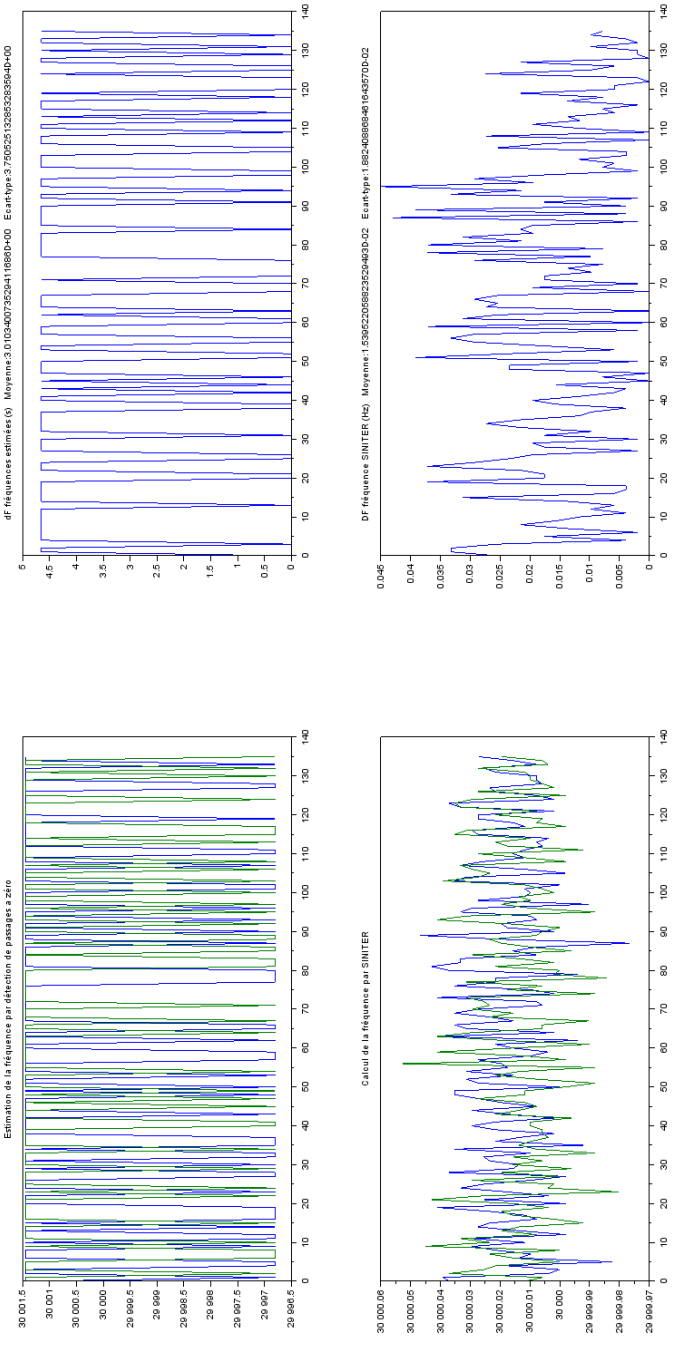


FIGURE 15.5 – Résultats SINITER @ 30kHz déc=64

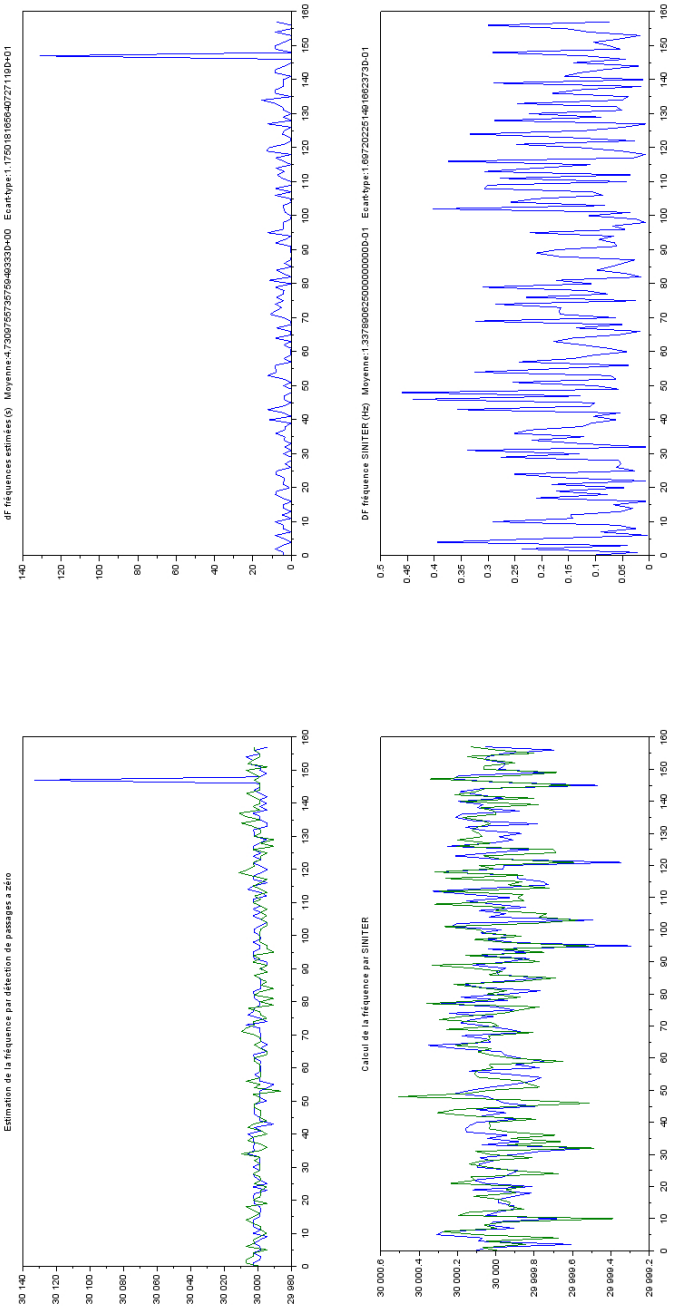


FIGURE 15.6 – Résultats SINITER @ 30kHz déc=8

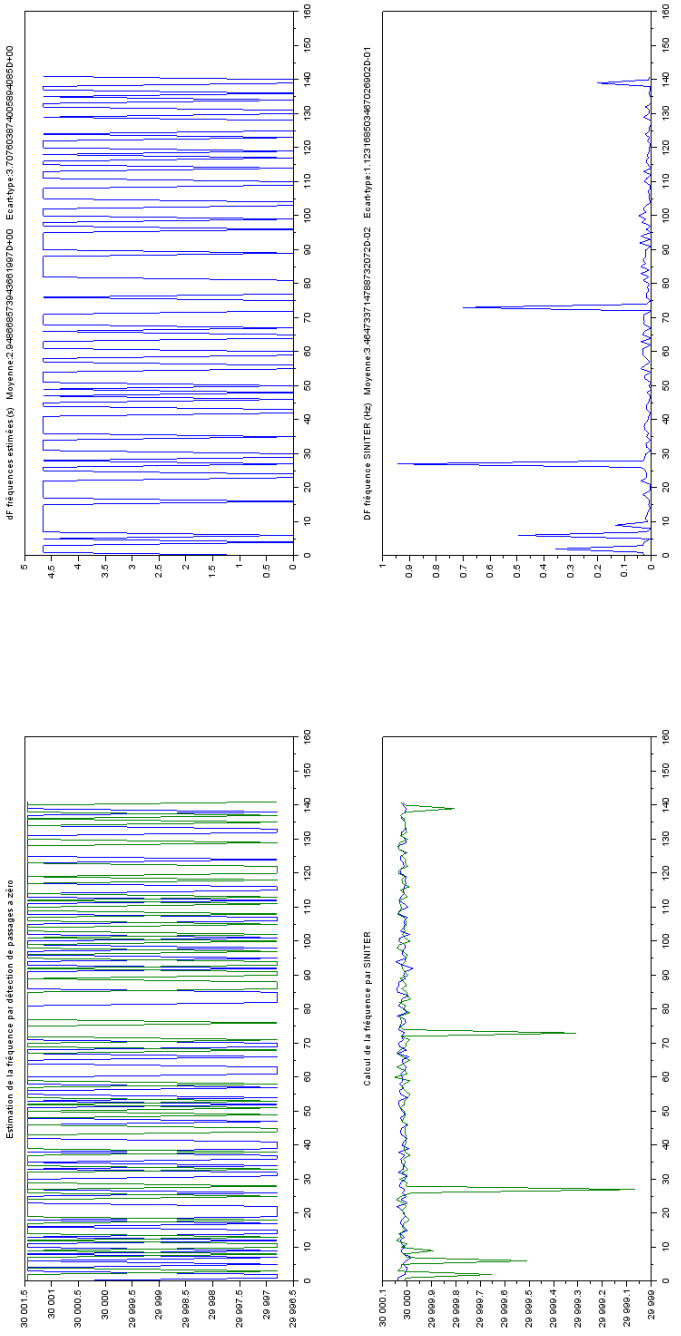


FIGURE 15.7 – Résultats SINITER @ 30kHz déc=64

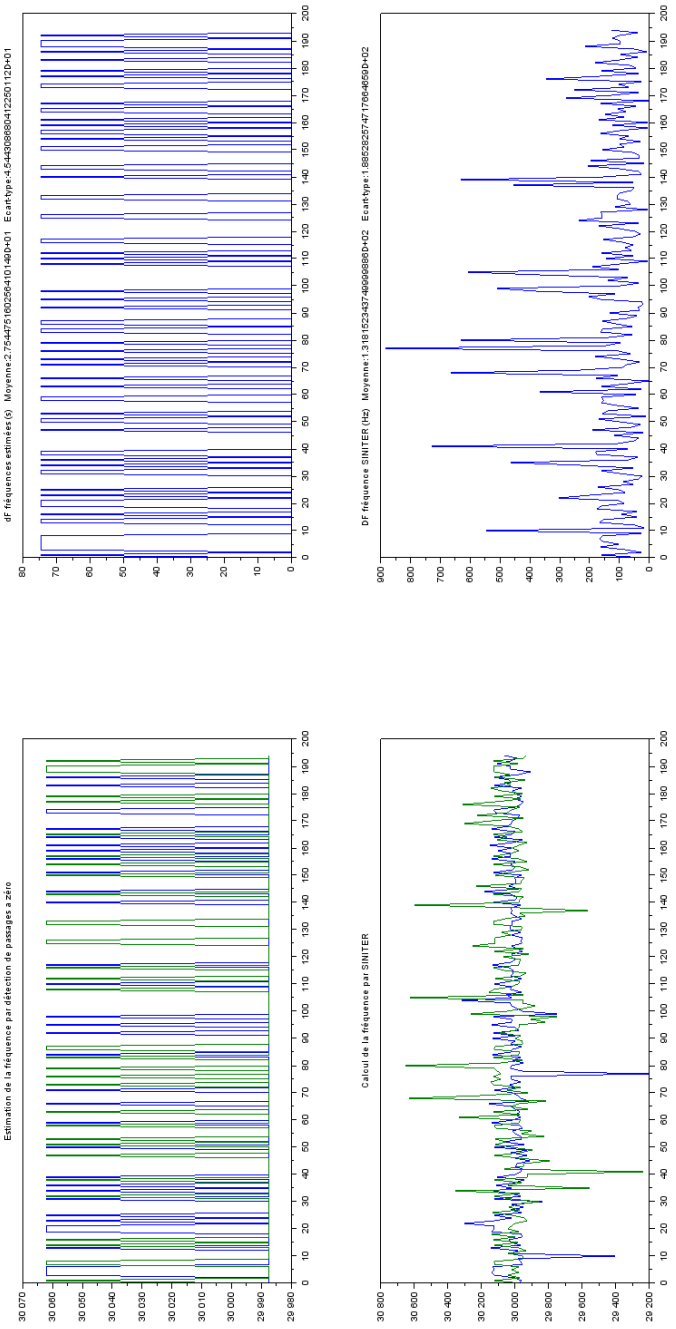
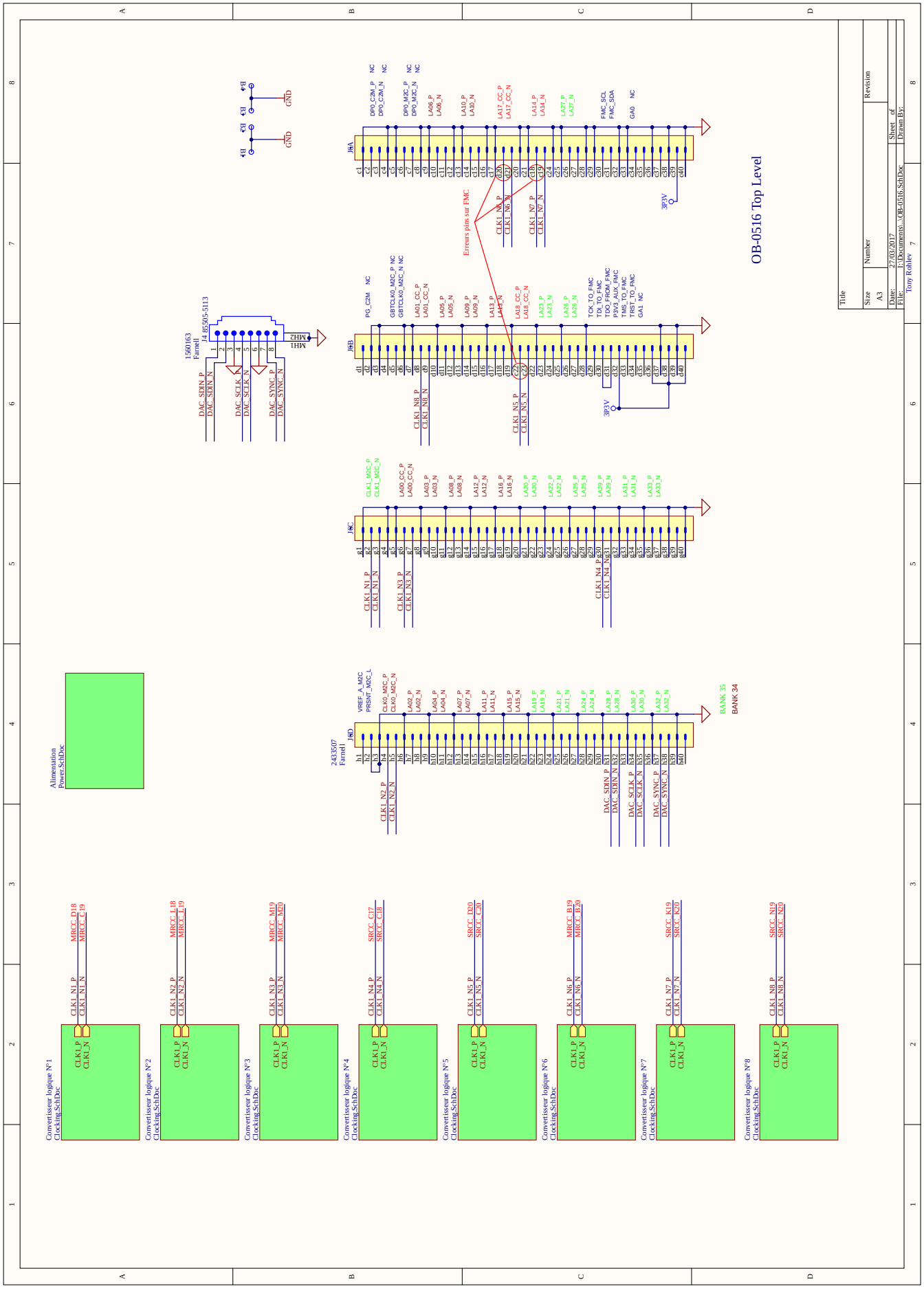


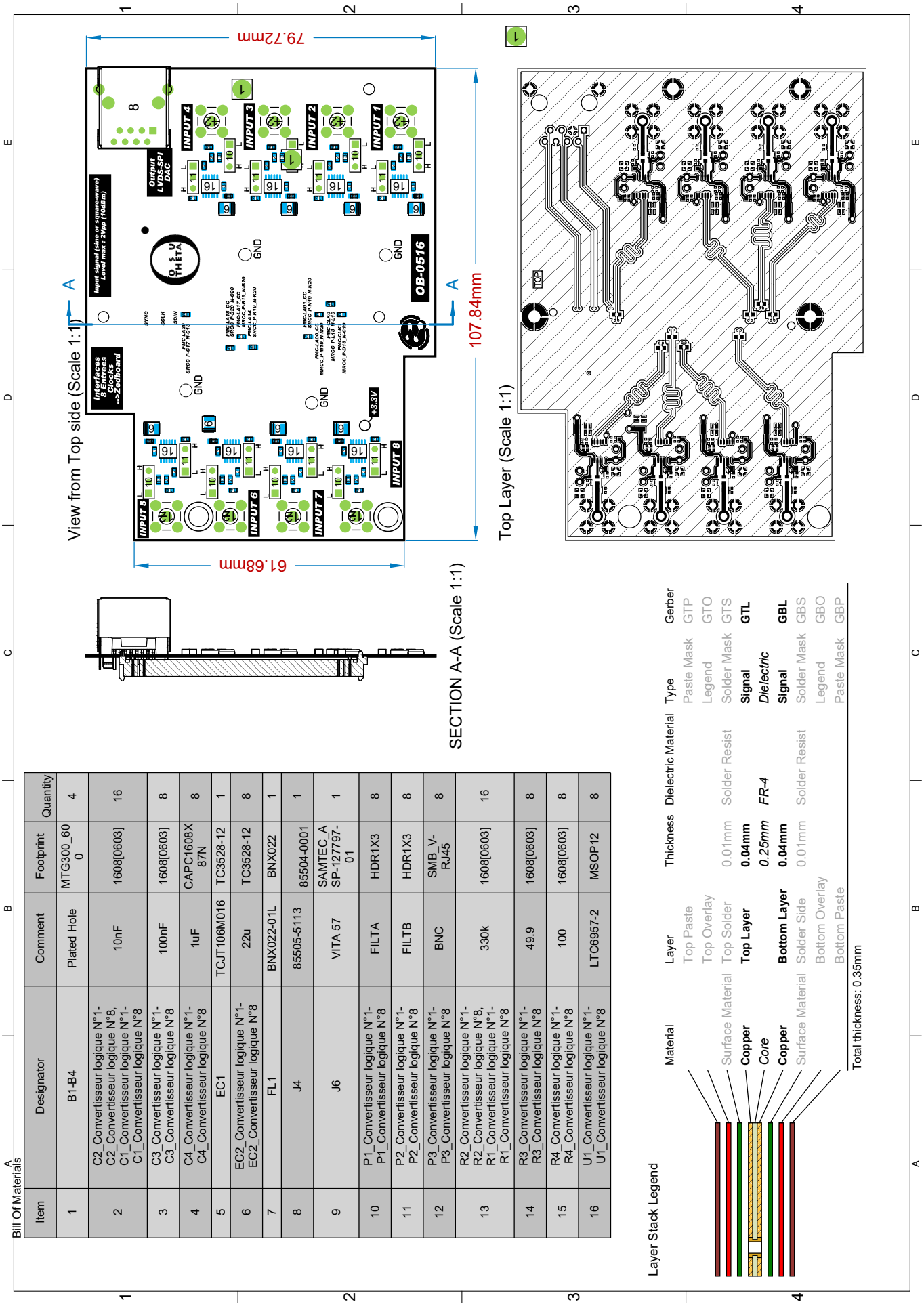
FIGURE 15.8 – Résultats SINITER @ 30kHz déc=1024

Chapitre 16

Schémas électroniques

16.1 Carte interface FMC - LVDS


OB-0516 Top Level
Alimentation
Power.SchDoc
Convertisseur logique N°1
Clocking.SchDoc
Convertisseur logique N°2
Clocking.SchDoc
Convertisseur logique N°3
Clocking.SchDoc
Convertisseur logique N°4
Clocking.SchDoc
Convertisseur logique N°5
Clocking.SchDoc
Convertisseur logique N°6
Clocking.SchDoc
Convertisseur logique N°7
Clocking.SchDoc
Convertisseur logique N°8
Clocking.SchDoc
J6A
J6B
J6C
J6D
243507
Farnell
1560163
Farnell
14 85505-5113
MH1
MH2
Erreurs pins sur FMC
3P3V
GND
BANK 35
BANK 34
Tony Rohlev



Bill Of Materials

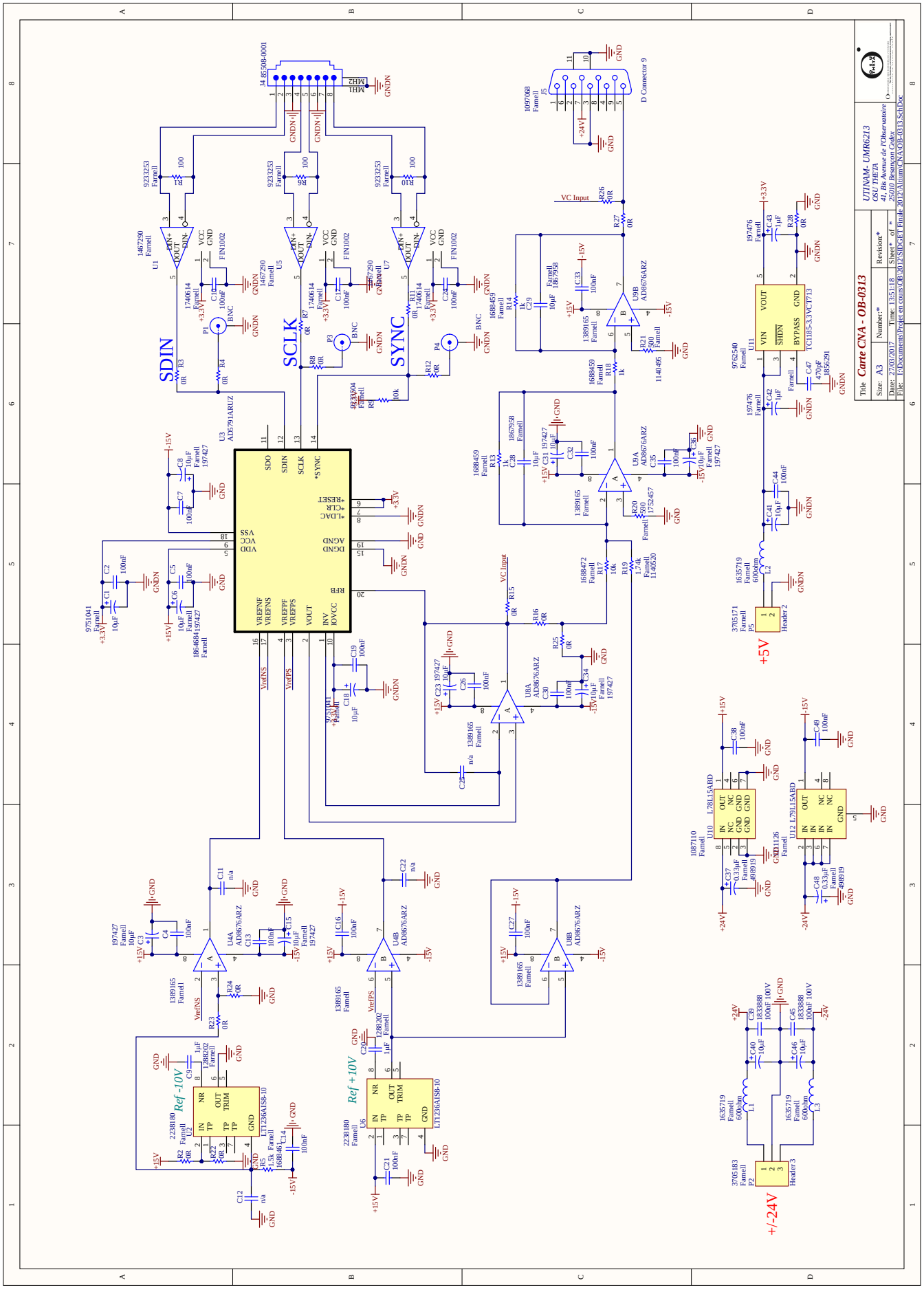
Item	Designator	Comment	Footprint	Quantity
1	B1-B4	Plated Hole	MTG300_600	4
2	C2_Convertisseur logique N°1-C2_Convertisseur logique N°8, C1_Convertisseur logique N°1-C1_Convertisseur logique N°8	10nF	1608[0603]	16
3	C3_Convertisseur logique N°1-C3_Convertisseur logique N°8	100nF	1608[0603]	8
4	C4_Convertisseur logique N°1-C4_Convertisseur logique N°8	1uF	CAPC1608X87N	8
5	EC1	TCJT106M016	TC3528-12	1
6	EC2_Convertisseur logique N°1-EC2_Convertisseur logique N°8	22u	TC3528-12	8
7	FL1	BNX022-01L	BNX022	1
8	J4	85505-5113	85504-0001	1
9	J6	VITA 57	SAMTEC_ASP-127797-01	1
10	P1_Convertisseur logique N°1-P1_Convertisseur logique N°8	FILTA	HDR1X3	8
11	P2_Convertisseur logique N°1-P2_Convertisseur logique N°8	FILTB	HDR1X3	8
12	P3_Convertisseur logique N°1-P3_Convertisseur logique N°8	BNC	SMB_V-RJ45	8
13	R2_Convertisseur logique N°1-R2_Convertisseur logique N°8, R1_Convertisseur logique N°1-R1_Convertisseur logique N°8	330k	1608[0603]	16
14	R3_Convertisseur logique N°1-R3_Convertisseur logique N°8	49.9	1608[0603]	8
15	R4_Convertisseur logique N°1-R4_Convertisseur logique N°8	100	1608[0603]	8
16	U1_Convertisseur logique N°1-U1_Convertisseur logique N°8	LTC6957-2	MSOP12	8

Layer Stack Legend

Material	Layer	Thickness	Dielectric Material	Type	Gerber
	Top Paste			Paste Mask	GTP
	Top Overlay			Legend	GTO
Surface Material	Top Solder	0.01mm	Solder Resist	Solder Mask	GTS
Copper	**Top Layer**	**0.04mm**		**Signal**	**GTL**
Core		*0.25mm*	*FR-4*	*Dielectric*	
Copper	**Bottom Layer**	**0.04mm**		**Signal**	**GBL**
Surface Material	Solder Side	0.01mm	Solder Resist	Solder Mask	GBS
	Bottom Overlay			Legend	GBO
	Bottom Paste			Paste Mask	GBP

Total thickness: 0.35mm

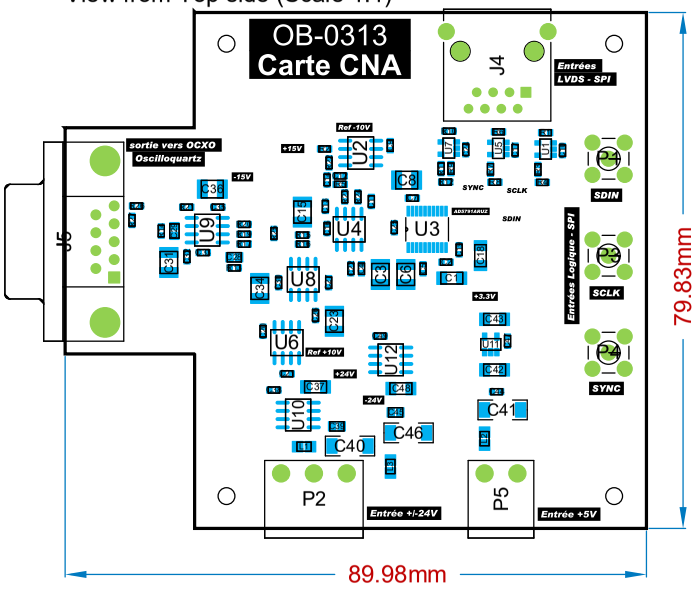
16.2 Carte CNA - SPI


SDIN
SCLK
SYNC
VC Input
+5V
+/-24V
Ref +10V
Ref -10V
AD7791ARUZ
AD8676ARZ
LT1236AIS8-10
FIN1002
TC1185-3.3VCT713
L78L15ABD
L79L15ABD
UTINAM- UMR6213
OBJ THETA
41 bis Avenue de l'Observatoire
25010 Besancon Cedex
Title: Carte CNA - OB-0313
Size: A3
Number:*
Revision:*
Date: 27/03/2017
Time: 13:51:18
Sheet* of *
File: I:\Documents\Projet en cours\OB-2012\SDC&T\ Finale 2012\Altium\CNA\OB-0313.SchDoc

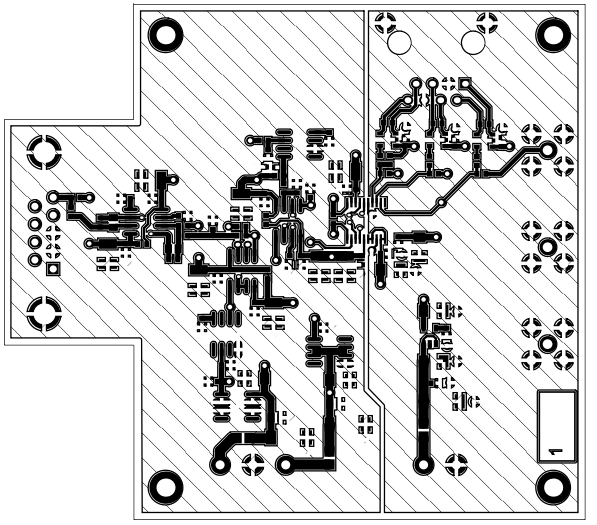
Bill Of Materials

Item	Designator	Comment	Quantity
1	C9, C11, C12, C20, C22, C25	n/a	6
2	C42, C43	1µF	2
3	C1, C3, C6, C8, C15, C18, C23, C28, C29, C31, C34, C36, C40, C41, C46	10µF	15
4	C37, C48	0.33µF	2
5	C2, C4, C5, C7, C10, C13, C14, C16, C17, C19, C21, C24, C26, C27, C30, C32, C33, C35, C38, C39, C44, C45, C47, C49	100nF	24
6	J4	85508-0001	1
7	J5	D Connector 9	1
8	L1-L3	600ohm	3
9	P2	Header 3	1
10	P1, P3, P4	BNC	3
11	P5	Header 2	1
12	R1, R6, R10	100	3
13	R9, R17	10k	2
14	R13, R14, R18	1k	3
15	R19	1.74k	1
16	R20	5.97k	1
17	R21	500	1
18	R2-R4, R7, R8, R11, R12, R15, R16, R22-R28	0R	16
19	R5	?	1
20	U10	L78L15ABD	1
21	U11	TC1185-3.3VCT713	1
22	U12	L79L15ABD	1
23	U3	AD5791ARUZ	1
24	U2, U6	LT1236AIS8-10	2
25	U1, U5, U7	FIN1002	3
26	U4, U8, U9	AD8676ARZ	3

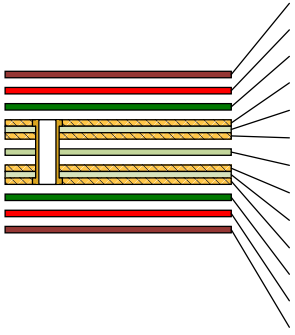
View from Top side (Scale 1:1)



Top Layer (Scale 1:1)



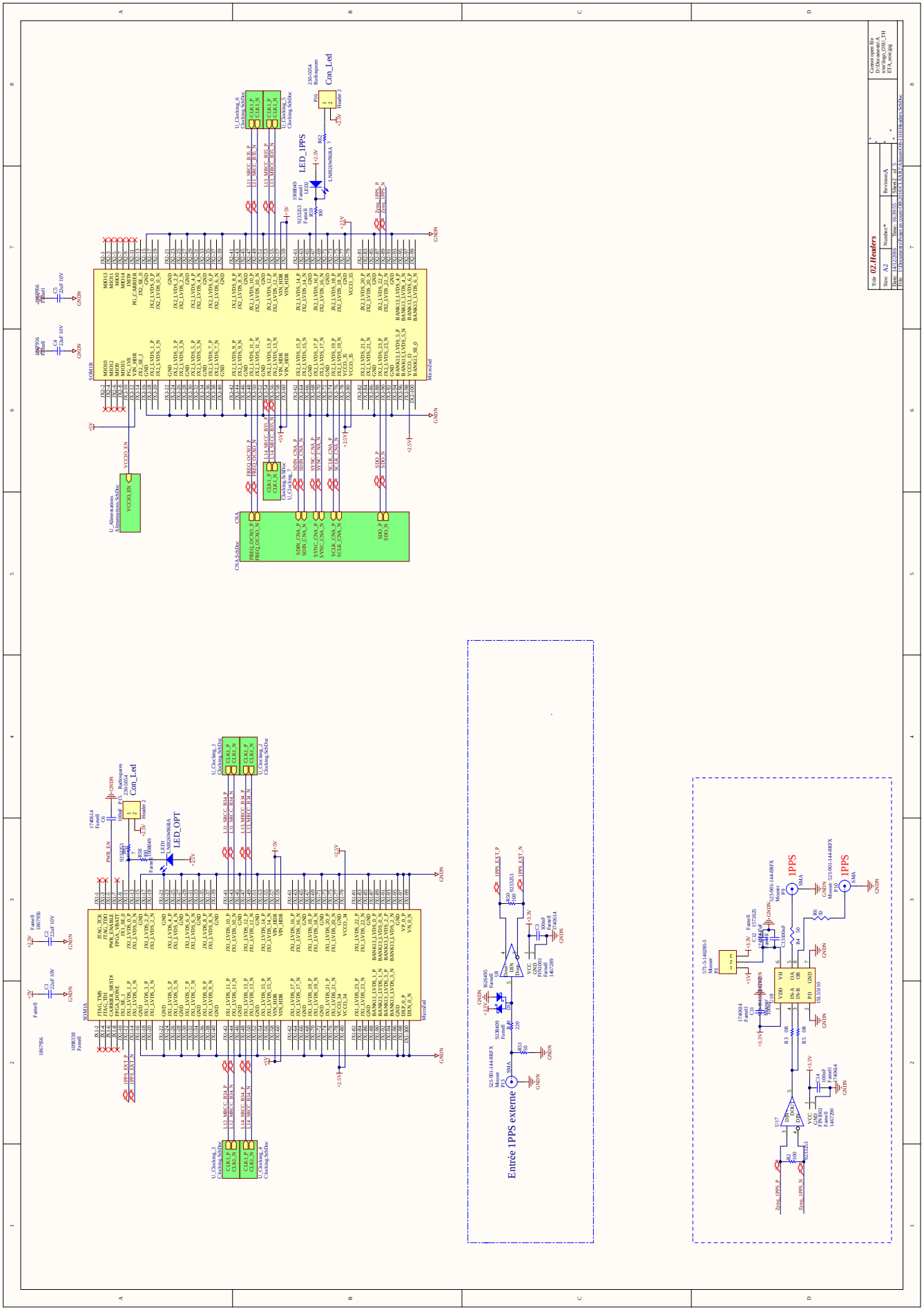
Layer Stack Legend

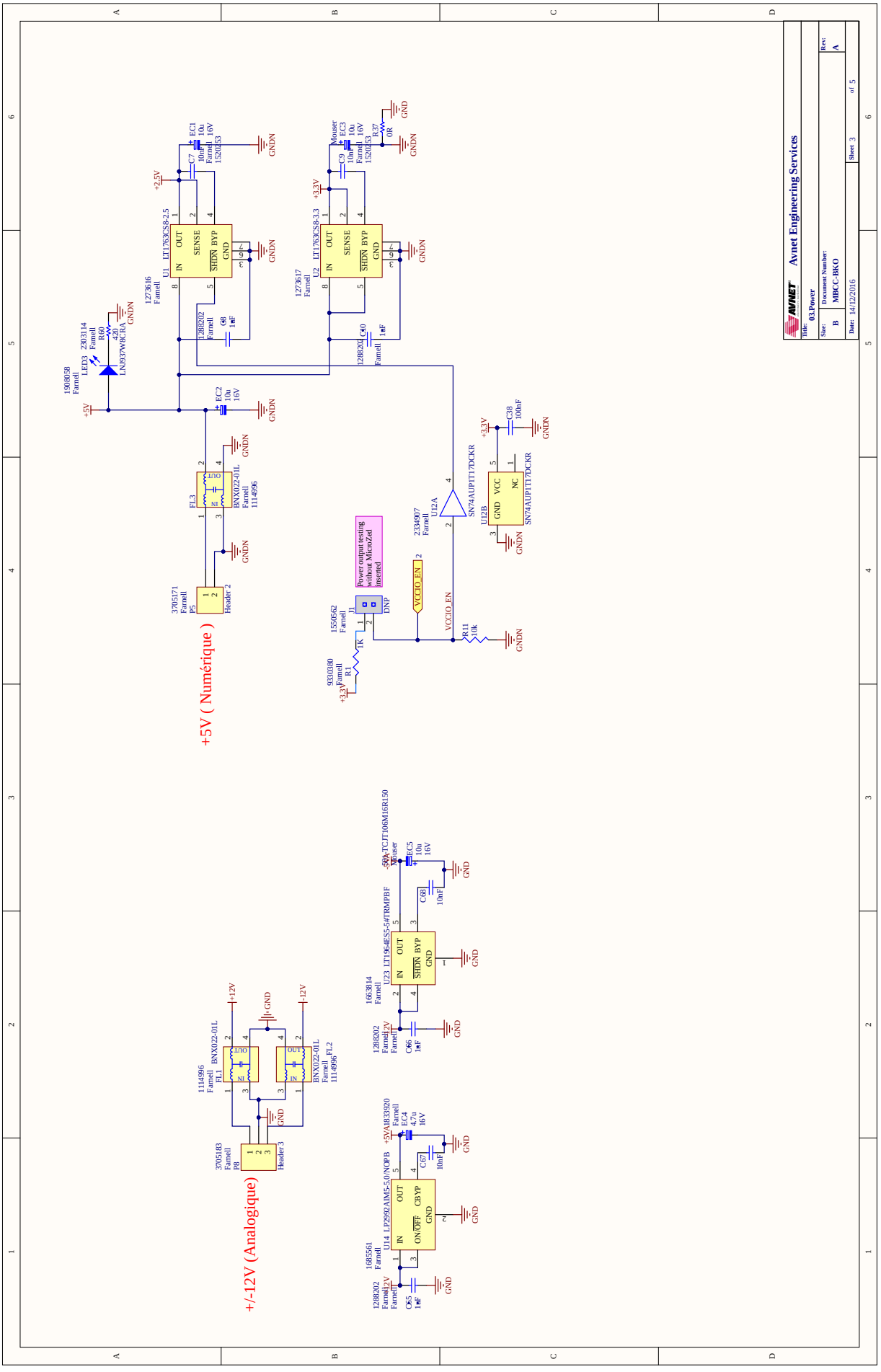


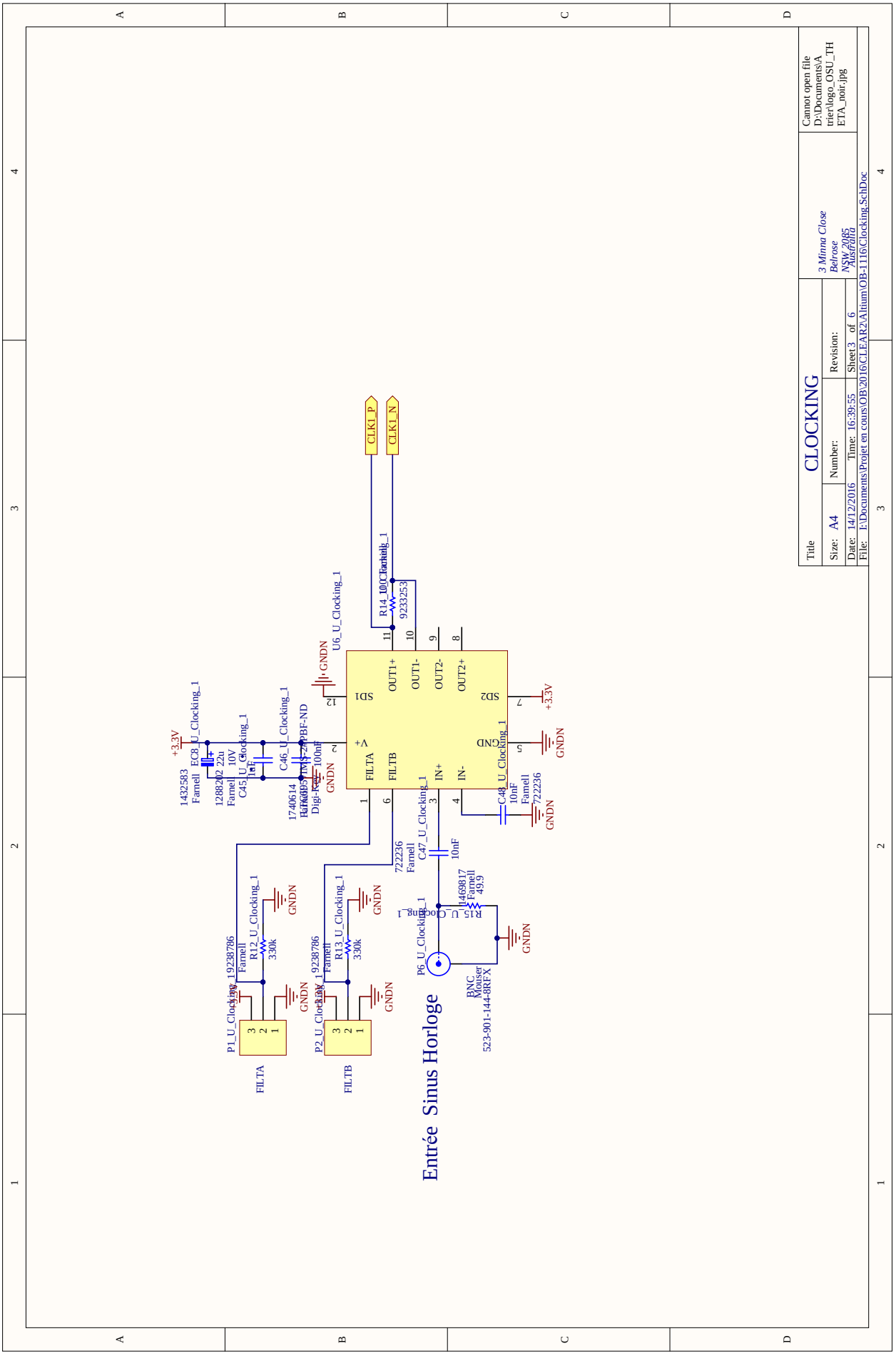
Material	Layer	Thickness	Dielectric Material	Type	Gerber
	Top Paste			Paste Mask	GTP
	Top Overlay			Legend	GTO
Surface Material	Top Solder	0.01mm	Solder Resist	Solder Mask	GTS
Copper	**Top Layer**	**0.04mm**		**Signal**	**GTL**
Core		*0.32mm*	*FR-4*	*Dielectric*	
Copper	**MidLayer1**	**0.04mm**		**Signal**	**G1**
Prepreg		*0.32mm*	*FR-4*	*Dielectric*	
Copper	**MidLayer2**	**0.04mm**		**Signal**	**G2**
Core		*0.32mm*	*FR-4*	*Dielectric*	
Copper	**Bottom Layer**	**0.04mm**		**Signal**	**GBL**
Surface Material	Bottom Solder	0.01mm	Solder Resist	Solder Mask	GBS
	Bottom Overlay			Legend	GBO
	Bottom Paste			Paste Mask	GBP

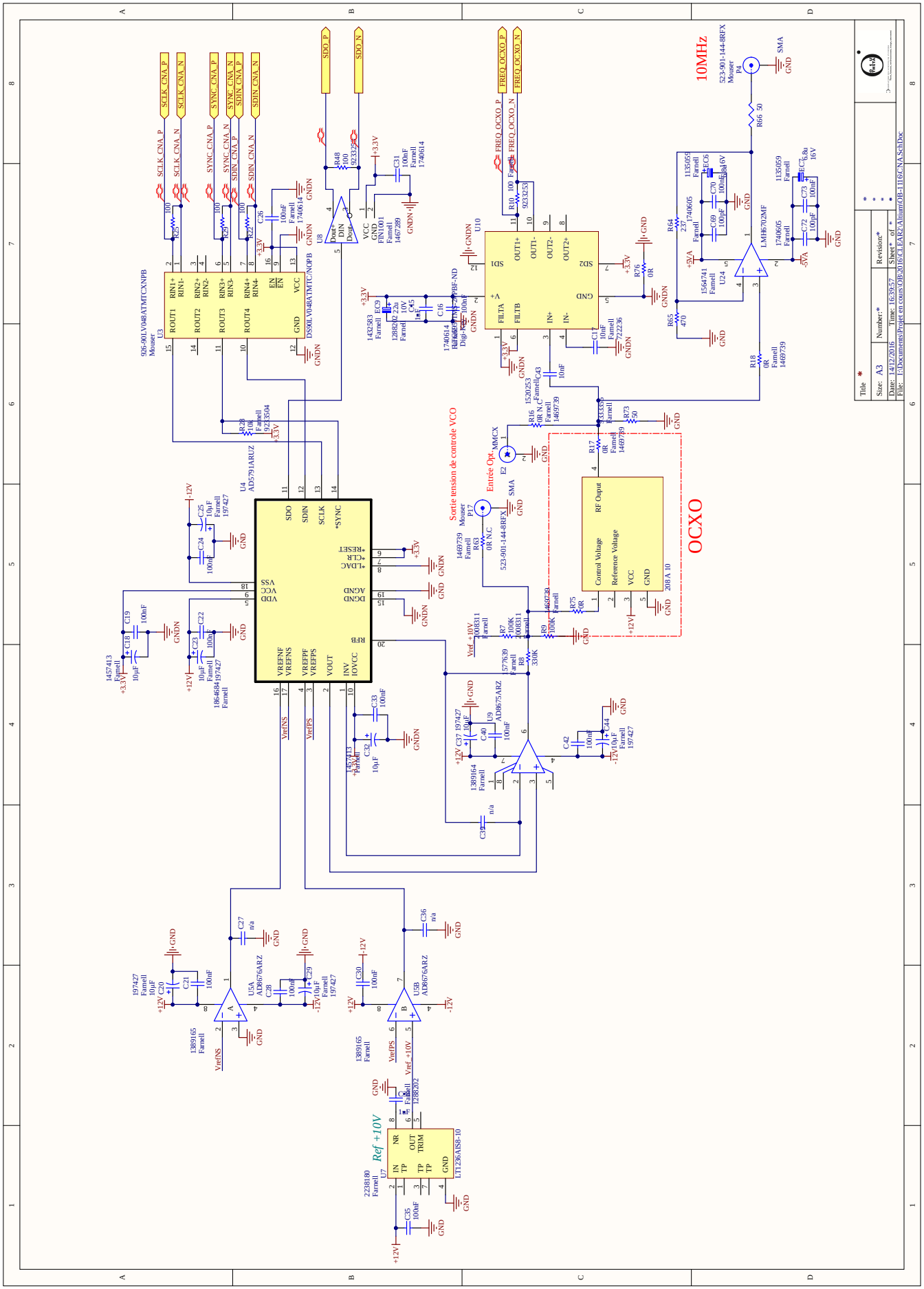
Total thickness: 1.12mm

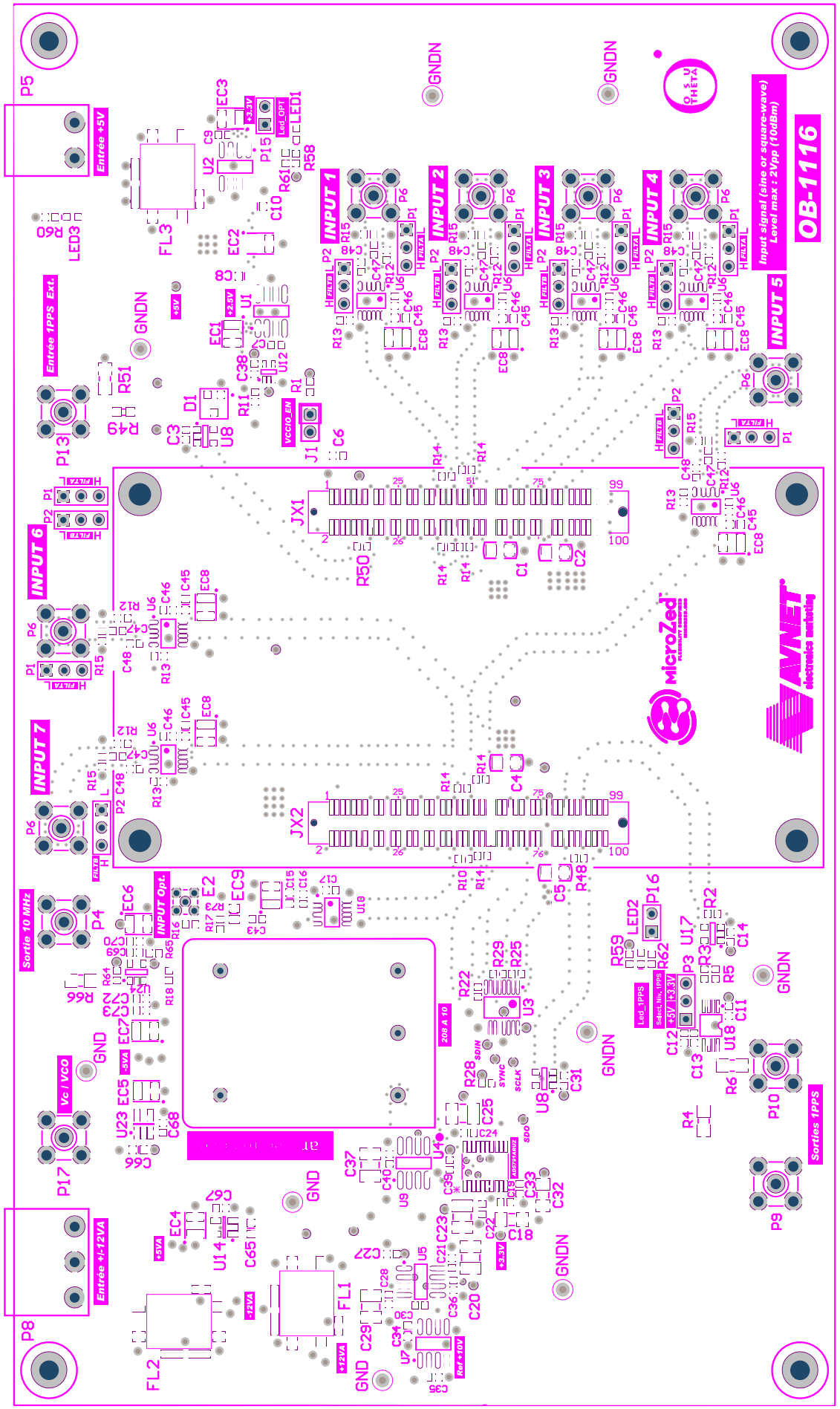
16.3 Montage MicroZed


02. Headers
Entrée 1PPS externe
1PPS
LED_1PPS
LED_OPT
Con_Led
SOM1A
SOM1B
CNA
U_Clocking_1
U_Clocking_2
U_Clocking_3
U_Clocking_4
U_Clocking_5
U_Clocking_6
U_Clocking_7
U_Alimentations
Microzed
CNA.SchDoc
Clocking.SchDoc
Alimentations.SchDoc


Avnet Engineering Services
0.1 Power
MBCC-BKO
+5V (Numérique)
+/-12V (Analogique)
Power output testing without MicroZed inserted
VCCIO_EN
LT1763CS8-2.5
LT1763CS8-3.3
SN74AUP1T17DCKR
LT1964ES5-5#TRMPBF
LP2992AIM5-5.0/NOPB
BNX022-01L
GNDN
GND


CLOCKING
Entrée Sinus Horloge
CLK1_P
CLK1_N
U6_U_Clocking_1
SD1
OUT1+
OUT1-
OUT2-
OUT2+
SD2
V+
FILTA
FILTB
IN+
IN-
GND
P1_U_Clocking_1
P2_U_Clocking_1
P6_U_Clocking_1
R12_U_Clocking_1
R13_U_Clocking_1
R14_U_Clocking_1
R15_U_Clocking_1
C45_U_Clocking_1
C46_U_Clocking_1
C47_U_Clocking_1
C48_U_Clocking_1
EC8_U_Clocking_1
+3.3V
GNDN
FILTA
FILTB
BNC
Mouser
523-901-144-8RFX
Farnell
1432583
1288202
1740614
722236
1469817
9238786
9233253
22u
10V
100nF
10nF
330k
49.9
Digi-Key
Title
Size: A4
Number:
Revision:
Date: 14/12/2016
Time: 16:39:55
Sheet 3 of 6
File: I:\Documents\Projet en cours\OB\2016\CLEAR2\Altium\OB-1116Clocking.SchDoc
3 Minna Close
Belrose
NSW 2085
Australia
Cannot open file
D:\Documents\Atrier\logo_OSU_THETA_noir.jpg


OCXO
10MHz
Sortie tension de controle VCO
Entrée Opt
Ref +10V
SCLK_CNA_P
SCLK_CNA_N
SYNC_CNA_P
SYNC_CNA_N
SDIN_CNA_P
SDIN_CNA_N
SDO_P
SDO_N
FREQ_OCXO_P
FREQ_OCXO_N
Title *
Size: A3
Number:*
Revision:*
Date: 14/12/2016
Time: 16:39:57
Sheet* of *
File: F:\Documents\Projet en cours\OB\20161CLEAR2\Altium\OB-1116\CNA.SchDoc


OB-1116
Input signal (sine or square-wave)
Level max : 2Vpp (10dBm)
INPUT 1
INPUT 2
INPUT 3
INPUT 4
INPUT 5
INPUT 6
INPUT 7
Entrée +5V
Entrée 1PPS Ext.
Entrée +/-12VA
Sortie 10 MHz
Vc / VCO
Sorties 1PPS
GNDN
GND
microZed
AVNET
electronics marketing

Chapitre 17

Fiches techniques

17.1 Ampli Op AD8675

ANALOG DEVICES

36 V Precision, 2.8 nV/√Hz Rail-to-Rail Output Op Amp

Data Sheet **AD8675**

FEATURES

Very low voltage noise: 2.8 nV/√Hz
Rail-to-rail output swing
Low input bias current: 2 nA maximum
Very low offset voltage: 75 µV maximum
Low input offset drift: 0.6 µV/°C maximum
Very high gain: 120 dB
Wide bandwidth: 10 MHz typical
±5 V to ±18 V operation

APPLICATIONS

Precision instrumentation
PLL filters
Laser diode control loops
Strain gage amplifiers
Medical instrumentation
Thermocouple amplifiers

PIN CONFIGURATIONS

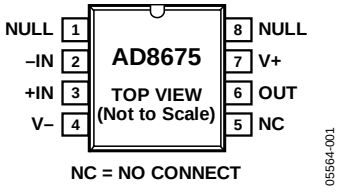


Figure 1. 8-Lead SOIC_N (R-8)

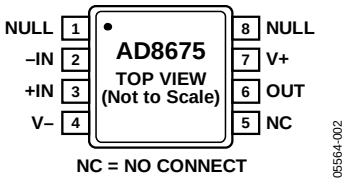


Figure 2. 8-Lead MSOP (RM-8)

GENERAL DESCRIPTION

The AD8675 precision operational amplifier has ultralow offset, drift, and voltage noise combined with very low input bias currents over the full operating temperature range. The AD8675 is a precision, wide bandwidth op amp featuring rail-to-rail output swings and very low noise. Operation is fully specified from ±5 V to ±15 V.

The AD8675 features a rail-to-rail output like that of the OP184, but with wide bandwidth and even lower voltage noise, combined with the precision and low power consumption like that of the industry-standard OP07 amplifier. Unlike other low noise, rail-to-rail op amps, the AD8675 has very low input bias current and low input current noise.

With typical offset voltage of only 10 µV, offset drift of 0.2 µV/°C, and noise of only 0.10 µV p-p (0.1 Hz to 10 Hz), the AD8675 is perfectly suited for applications where large error sources cannot be tolerated. For applications with even lower offset tolerances, the proprietary nulling capability allows a combination of both device and system offset errors up to 3.5 mV (referred to the input) to be compensated externally. Unlike previous circuits, the AD8675 accommodates this adjustment without adversely affecting the offset drift, CMRR, and PSRR of the amplifier. Precision instrumentation, PLL, and other precision filter circuits, position and pressure sensors, medical instrumentation, and strain gage amplifiers benefit greatly from the very low noise, low input bias current, and wide bandwidth. Many systems can take advantage of the low noise, dc precision, and rail-to-rail output swing provided by the AD8675 to maximize SNR and dynamic range.

The smaller packages and low power consumption afforded by the AD8675 allow maximum channel density or minimum board size for space-critical equipment.

The AD8675 is specified for the extended industrial temperature range (−40°C to +125°C). The AD8675 amplifier is available in the tiny 8-lead MSOP, and the popular 8-lead, narrow SOIC, RoHS compliant packages. MSOP packaged devices are only available in tape and reel format.

For the dual version of this ultraprecision, rail-to-rail op amp, see the AD8676 data sheet.

The AD8675 and AD8676 are members of a growing series of low noise op amps offered by Analog Devices, Inc.

Table 1. Voltage Noise

Package	0.9 nV	1.1 nV	1.8 nV	2.8 nV	3.8 nV
Single	AD797	AD8597	ADA4004-1	AD8675	AD8671
Dual		AD8599	ADA4004-2	AD8676	AD8672
Quad			ADA4004-4		AD8674

Rev. E
Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties that may result from its use. Specifications subject to change without notice. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices. Trademarks and registered trademarks are the property of their respective owners.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 www.analog.com
Fax: 781.461.3113 ©2005–2012 Analog Devices, Inc. All rights reserved.

AD8675* Product Page Quick Links

Last Content Update: 11/01/2016

Comparable Parts

View a parametric search of comparable parts

Evaluation Kits

- EVAL-OPAMP-1 Evaluation Board

Documentation

Application Notes

- AN-358: Noise and Operational Amplifier Circuits

Data Sheet

- AD8675: 36 V Precision, 2.8 nV/√Hz Rail-to-Rail Output Op Amp Data Sheet

Tools and Simulations

- Analog Filter Wizard
- Analog Photodiode Wizard
- AD8675 SPICE Macro Model

Reference Designs

- CN0169
- CN0191
- CN0200
- CN0257
- CN0318

Design Resources

- AD8675 Material Declaration
- PCN-PDN Information
- Quality And Reliability
- Symbols and Footprints

Discussions

View all AD8675 EngineerZone Discussions

Sample and Buy

Visit the product page to see pricing options

Technical Support

Submit a technical question or find your regional support number

* This page was dynamically generated by Analog Devices, Inc. and inserted into this data sheet. Note: Dynamic changes to the content on this page does not constitute a change to the revision number of the product data sheet. This content may be frequently modified.

TABLE OF CONTENTS

Features ... 1
Applications ... 1
Pin Configurations ... 1
General Description ... 1
Revision History ... 2
Specifications ... 3
Electrical Specifications ... 3
Absolute Maximum Ratings ... 5
Thermal Resistance ... 5
Power Sequencing ... 5
ESD Caution ... 5
Typical Performance Characteristics ... 6
Test Circuit ... 11
Outline Dimensions ... 12
Ordering Guide ... 12

REVISION HISTORY

7/12—Rev. D to Rev. E
Added Power Sequencing Section ... 5
Added Figure 28 and Figure 29; Renumbered Sequentially ... 10

8/11—Rev. C to Rev. D
Added Input Capacitance, Common Mode Parameter and Input Capacitance, Differential Mode Parameter, Table 3 ... 4

6/10—Rev. B to Rev. C
Changes to Figure 10 ... 7

5/10—Rev. A to Rev. B
Changes to General Description Section ... 1
Added Table 1; Renumbered Sequentially ... 1
Changes to Table 2 ... 3
Changes to Table 3 ... 4
Changes to Table 4 and Table 5 ... 5
Changes to Figure 4 and Figure 6 to Figure 8 ... 6
Changes to Figure 15 ... 8
Changes to Figure 21 and Figure 24 ... 9
Added Figure 27; Renumbered Sequentially ... 10
Updated Outline Dimensions ... 12
Changes to Ordering Guide ... 12

4**/07—Rev. 0 to Rev. A**
Added Figure 7 and Figure 8; Renumbered Sequentially ... 6

10/05—Revision 0: Initial Version

Data Sheet **AD8675**

SPECIFICATIONS

ELECTRICAL SPECIFICATIONS

V_S = ±5.0 V, V_{CM} = 0 V, V_O = 0 V, T_A = +25°C, unless otherwise specified.

Table 2.

Parameter	Symbol	Test Conditions/Comments	Min	Typ	Max	Unit
INPUT CHARACTERISTICS						
Offset Voltage	V_{OS}			10	75	μV
		−40°C ≤ T_A ≤ +125°C		12	240	μV
Offset Voltage Drift	$\Delta V_{OS}/\Delta T$	−40°C ≤ T_A ≤ +125°C		0.2	0.6	μV/°C
Input Bias Current	I_B		−2	+0.5	+2	nA
		−40°C ≤ T_A ≤ +125°C	−5.5	−2	+5.5	nA
Input Offset Current	I_{OS}		−1	+0.1	+1	nA
		−40°C ≤ T_A ≤ +125°C	−2.8	+0.1	+2.8	nA
Input Voltage Range	IVR		−3.0		+3.0	V
Common-Mode Rejection Ratio	CMRR	V_{CM} = −3.0 V to +3.0 V	105	130		dB
		−40°C ≤ T_A ≤ +125°C	105			dB
Open-Loop Gain	A_{VO}	R_L = 2 kΩ to ground, V_O = −3.5 V to +3.5 V	120	126		dB
		−40°C ≤ T_A ≤ +125°C	117			dB
OUTPUT CHARACTERISTICS						
Output Voltage High	V_{OH}	R_L = 10 kΩ to ground	4.90	4.95		V
		−40°C ≤ T_A ≤ +125°C	4.85			V
		R_L = 2 kΩ to ground	4.80	4.90		V
		−40°C ≤ T_A ≤ +125°C	4.75			V
Output Voltage Low	V_{OL}	R_L = 10 kΩ to ground		−4.98	−4.90	V
		−40°C ≤ T_A ≤ +125°C			−4.85	V
		R_L = 2 kΩ to ground		−4.91	−4.86	V
		−40°C ≤ T_A ≤ +125°C			−4.82	V
Short-Circuit Limit	I_{SC}			±35		mA
POWER SUPPLY						
Power Supply Rejection Ratio	PSRR	V_S = ±5.0 V to ±15.0 V	120	140		dB
		−40°C ≤ T_A ≤ +125°C	120			dB
Supply Current/Amplifier	I_{SY}	V_O = 0 V		2.3	2.7	mA
		−40°C ≤ T_A ≤ +125°C			3.4	mA
DYNAMIC PERFORMANCE						
Slew Rate	SR	R_L = 2 kΩ		2.5		V/μs
Gain Bandwidth Product	GBP			10		MHz
NOISE PERFORMANCE						
Voltage Noise	$e_{n\ p\text{-}p}$	0.1 Hz to 10 Hz		0.1		μV p-p
Voltage Noise Density	e_n	f = 1 kHz		2.8		nV/√Hz
Current Noise Density	i_n	f = 10 Hz		0.3		pA/√Hz

V_S = ±15 V, V_{CM} = 0 V, V_O = 0 V, T_A = +25°C, unless otherwise specified.

Table 3.

Parameter	Symbol	Test Conditions/Comments	Min	Typ	Max	Unit
INPUT CHARACTERISTICS						
Offset Voltage	V_{OS}			10	75	µV
		−40°C ≤ T_A ≤ +125°C		12	240	µV
Offset Voltage Drift	$\Delta V_{OS}/\Delta T$	−40°C ≤ T_A ≤ +125°C		0.2	0.6	µV/°C
Input Bias Current	I_B		−2	+0.5	+2	nA
		−40°C ≤ T_A ≤ +125°C	−4.5	+1	+4.5	nA
Input Offset Current	I_{OS}		−1	+0.1	+1	nA
		−40°C ≤ T_A ≤ +125°C	−2.8	+0.1	+2.8	nA
Input Voltage Range	IVR		−12.5		+12.5	V
Common-Mode Rejection Ratio	CMRR	V_{CM} = −12.5 V to +12.5 V	114	130		dB
		−40°C ≤ T_A ≤ +125°C	114			dB
Open-Loop Gain	A_{VO}	R_L = 2 kΩ to ground, V_O = −13.5 V to +13.5 V	123	132		dB
		−40°C ≤ T_A ≤ +125°C	117			dB
Input Capacitance, Common Mode	C_{INCM}			3.8		pF
Input Capacitance, Differential Mode	C_{INDM}			9.6		pF
OUTPUT CHARACTERISTICS						
Output Voltage High	V_{OH}	R_L = 10 kΩ to ground	14.85	14.92		V
		−40°C ≤ T_A ≤ +125°C	14.80			V
		R_L = 2 kΩ to ground	14.60	14.80		V
		−40°C ≤ T_A ≤ +125°C	14.40			V
Output Voltage Low	V_{OL}	R_L = 10 kΩ to ground		−14.96	−14.94	V
		−40°C ≤ T_A ≤ +125°C			−14.90	V
		R_L = 2 kΩ to ground		−14.85	−14.75	V
		−40°C ≤ T_A ≤ +125°C			−14.69	V
Short-Circuit Limit	I_{SC}			±35		mA
POWER SUPPLY						
Power Supply Rejection Ratio	PSRR	V_S = ±5.0 V to ±15.0 V	120	140		dB
		−40°C ≤ T_A ≤ +125°C	120			dB
Supply Current/Amplifier	I_{SY}	V_O = 0 V		2.5	2.9	mA
		−40°C ≤ T_A ≤ +125°C			3.8	mA
DYNAMIC PERFORMANCE						
Slew Rate	SR	R_L = 10 kΩ		2.5		V/µs
Gain Bandwidth Product	GBP			10		MHz
NOISE PERFORMANCE						
Voltage Noise	$e_{n\ p\text{-}p}$	0.1 Hz to 10 Hz		0.1		µV p-p
Voltage Noise Density	e_n	f = 1 kHz		2.8		nV/√Hz
Current Noise Density	i_n	f = 10 Hz		0.3		pA/√Hz

ABSOLUTE MAXIMUM RATINGS

Table 4.

Parameter	Rating
Supply Voltage	±18 V
Input Voltage	±V supply
Input Current	±5 mA
Differential Input Voltage	±0.7 V
Output Short-Circuit Duration to GND	Indefinite
Storage Temperature Range	
RM-8, R-8 Packages	−65°C to +150°C
Operating Temperature Range	−40°C to +125°C
Junction Temperature Range	
RM-8, R-8 Packages	−65°C to +150°C
Lead Temperature Range (Soldering, 10 sec)	300°C
NULL Pins (Pin 1, Pin 8), Input Current Maximum	<50 µA, V_S < V+

Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; functional operation of the device at these or any other conditions above those indicated in the operational section of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

THERMAL RESISTANCE

θ_{JA} is specified for the worst-case conditions, that is, a device soldered in a circuit board for surface-mount packages and measured using a standard 4-layer board, unless otherwise specified.

Table 5. Thermal Resistance

Package Type	θ_{JA}	θ_{JC}	Unit
8-Lead MSOP (RM-8)	142	45	°C/W
8-Lead SOIC_N (R-8)	120	45	°C/W

POWER SEQUENCING

Establish the op amp supplies simultaneously with, or before, any input signals are applied. If this is not possible, the input current must be limited to 10 mA.

ESD CAUTION

ESD (electrostatic discharge) sensitive device. Charged devices and circuit boards can discharge without detection. Although this product features patented or proprietary protection circuitry, damage may occur on devices subjected to high energy ESD. Therefore, proper ESD precautions should be taken to avoid performance degradation or loss of functionality.

TYPICAL PERFORMANCE CHARACTERISTICS

±15 V and ±5 V, T_A = 25°C, unless otherwise specified.

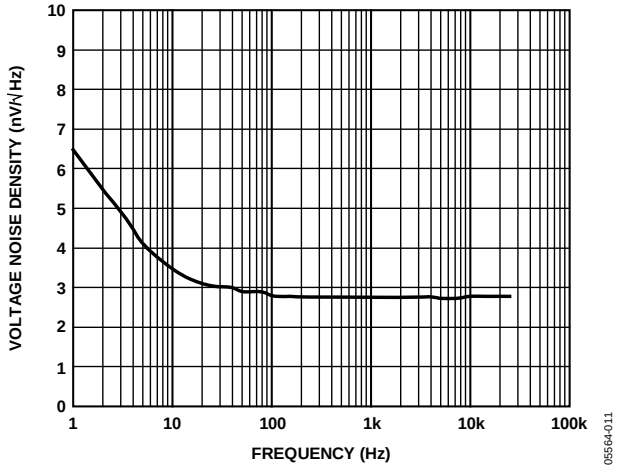


Figure 3. Voltage Noise Density vs. Frequency

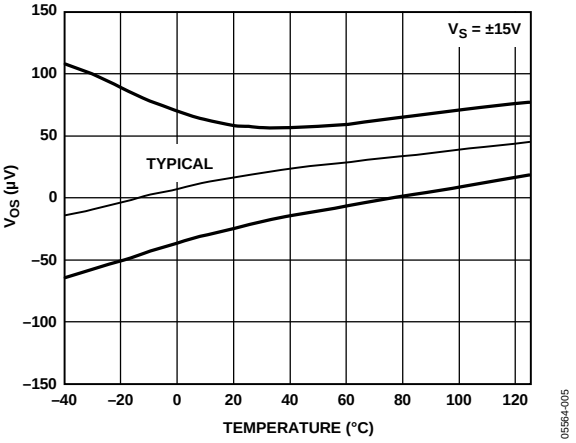


Figure 6. Offset Voltage vs. Temperature

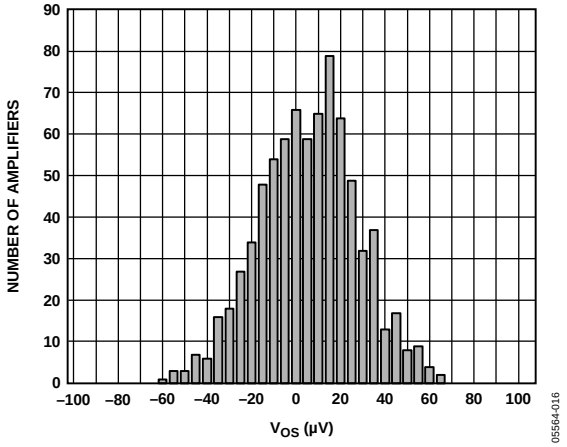


Figure 4. Input Offset Voltage Distribution

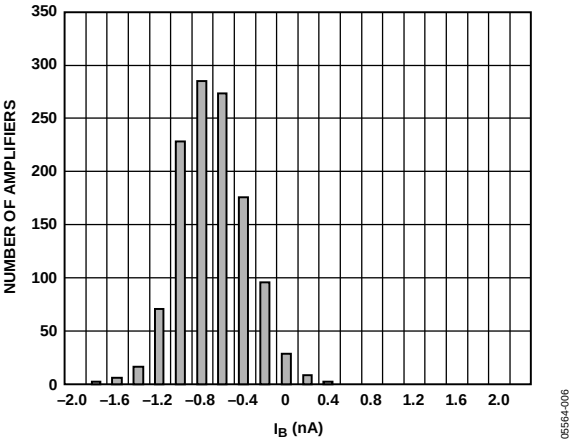


Figure 7. Input Bias Current, V_S = ±15 V

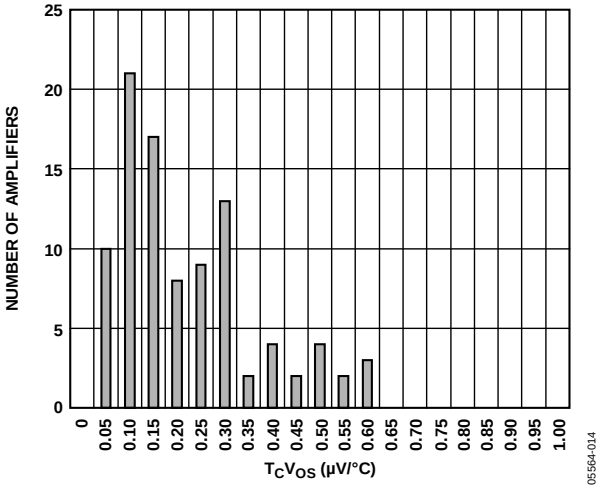


Figure 5. T_CV_{OS}

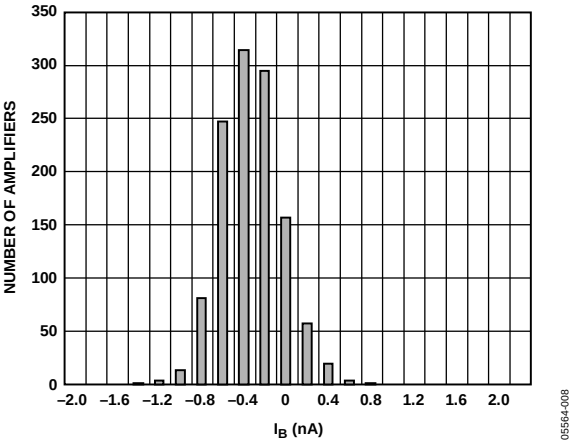


Figure 8. Input Bias Current, V_S = ±5 V

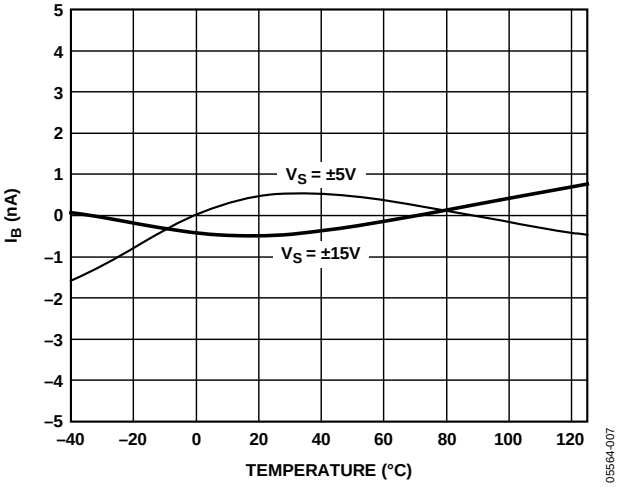


Figure 9. Input Bias Current vs. Temperature

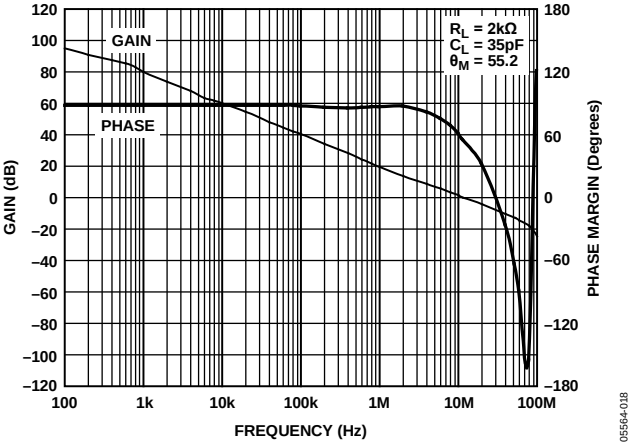


Figure 12. Gain and Phase vs. Frequency

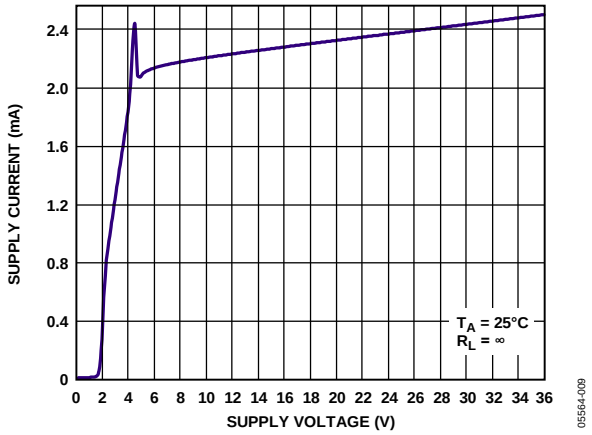


Figure 10. Supply Current vs. Total Supply Voltage

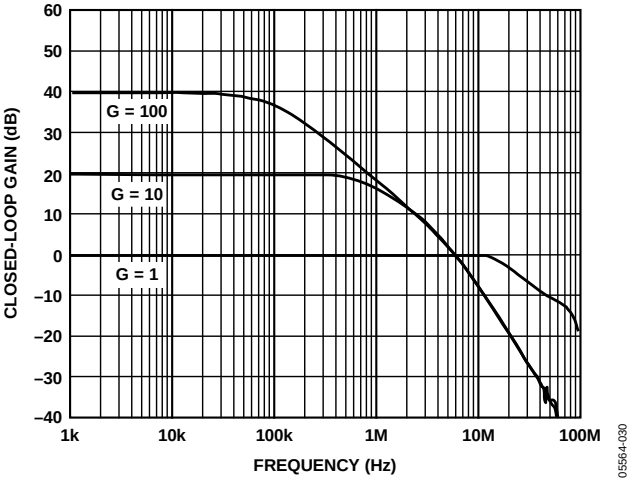


Figure 13. Closed-Loop Gain vs. Frequency

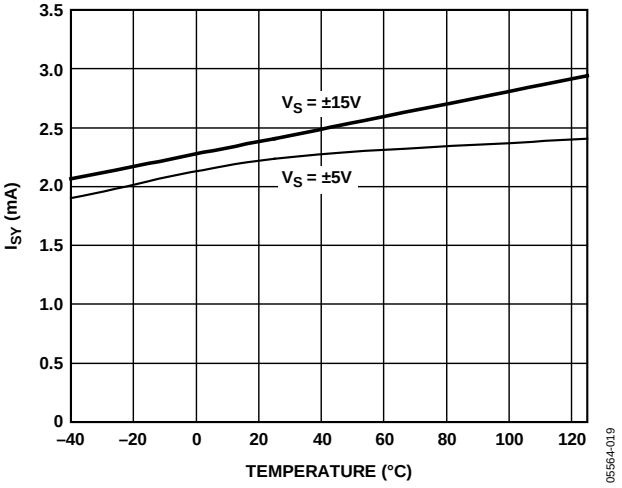


Figure 11. Supply Current vs. Temperature

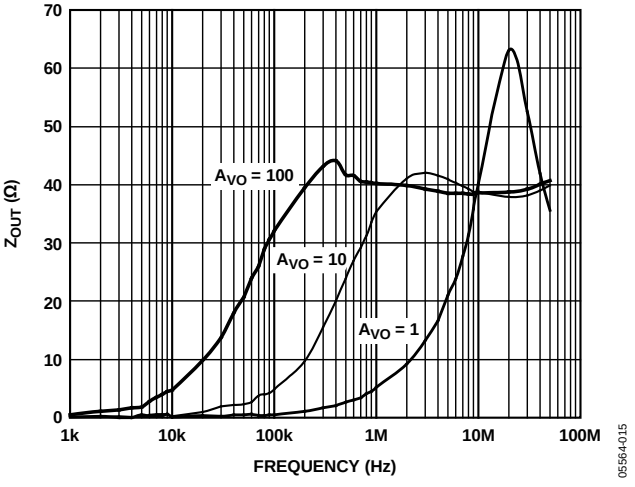


Figure 14. Z_{OUT} vs. Frequency

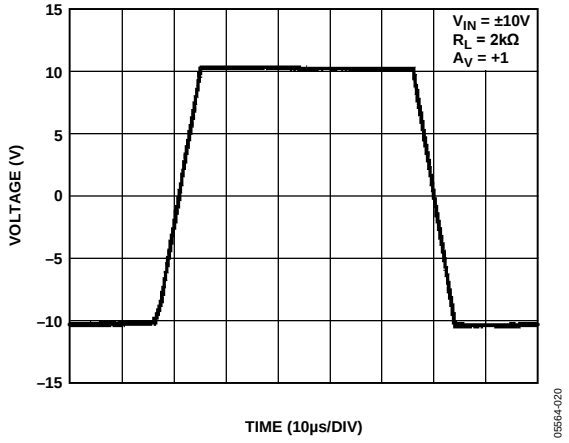


Figure 15. Large-Signal Transient Response, V_{SY} = ±15 V

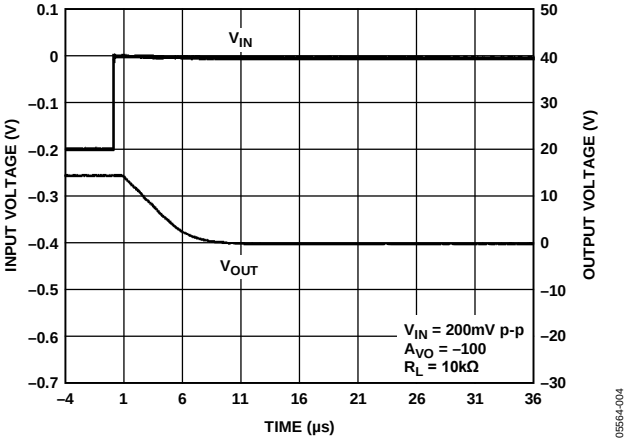


Figure 18. Positive Overvoltage Recovery

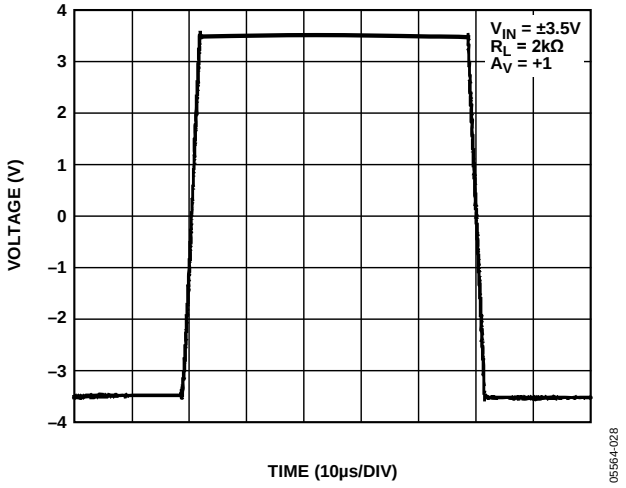


Figure 16. Large-Signal Transient Response, V_{SY} = ±5 V

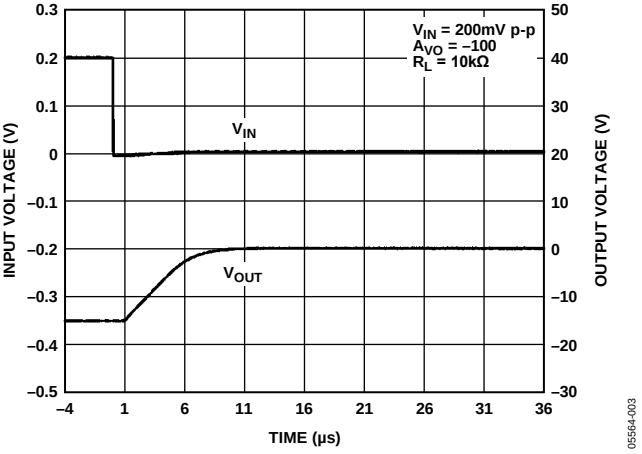


Figure 19. Negative Overvoltage Recovery

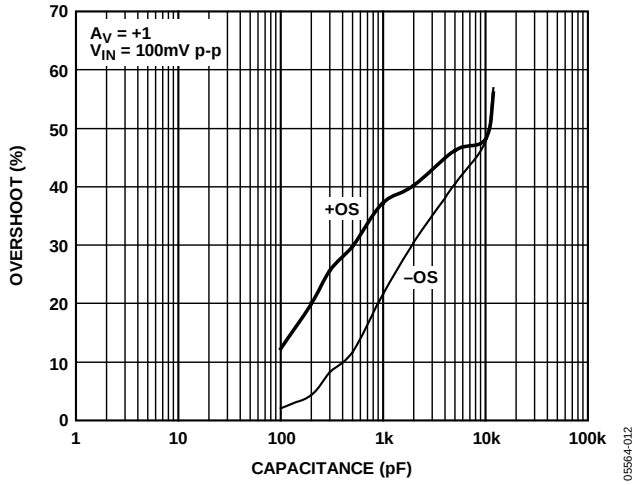


Figure 17. Small-Signal Overshoot vs. Load Capacitance

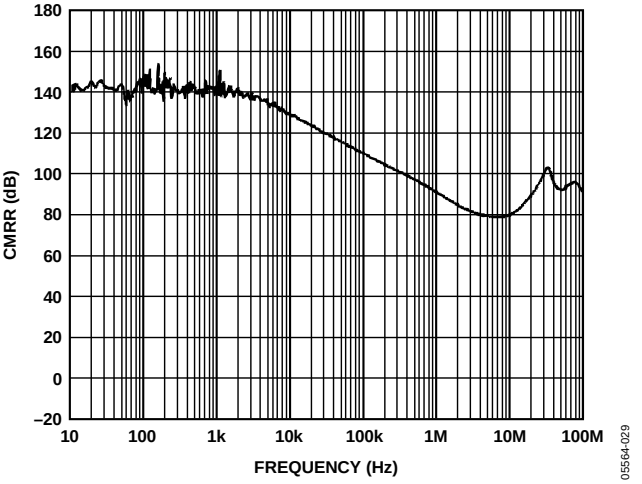


Figure 20. CMRR vs. Frequency

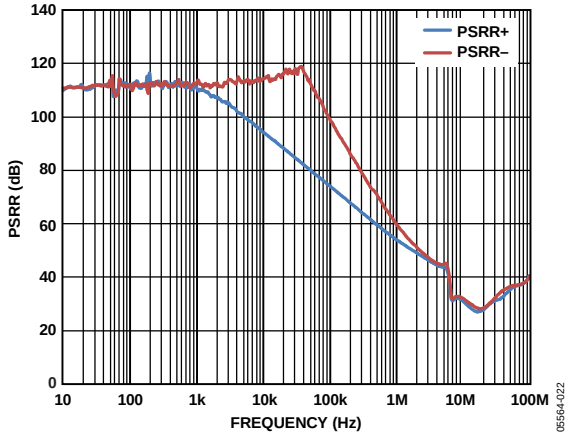


Figure 21. Power Supply Rejection Ratio vs. Frequency

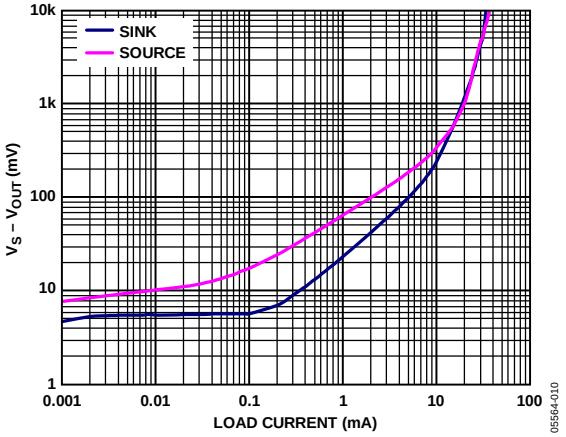


Figure 24. Output Saturation Voltage vs. Output Current

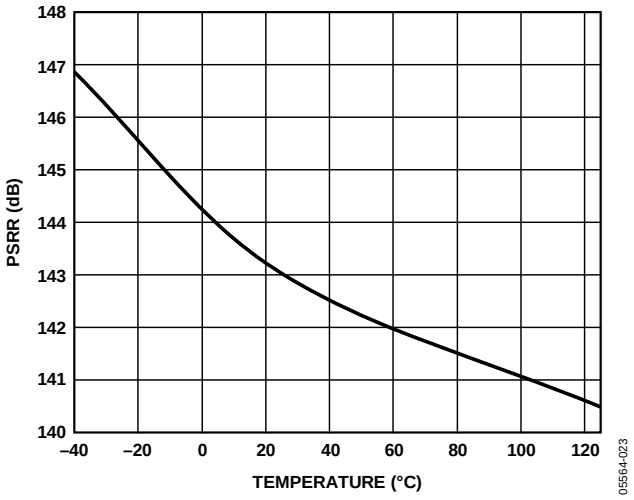


Figure 22. Power Supply Rejection Ratio vs. Temperature

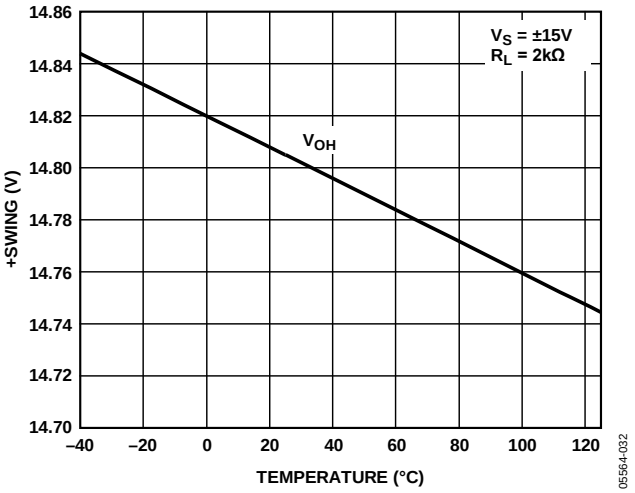


Figure 25. Swing vs. Temperature, V_{OH}

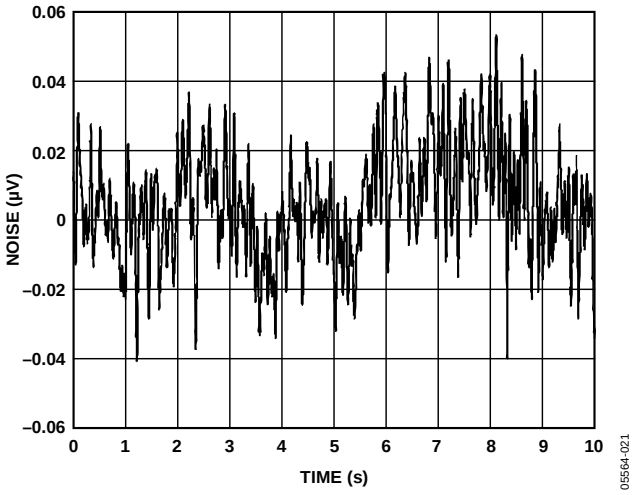


Figure 23. Voltage Noise (0.1 Hz to 10 Hz)

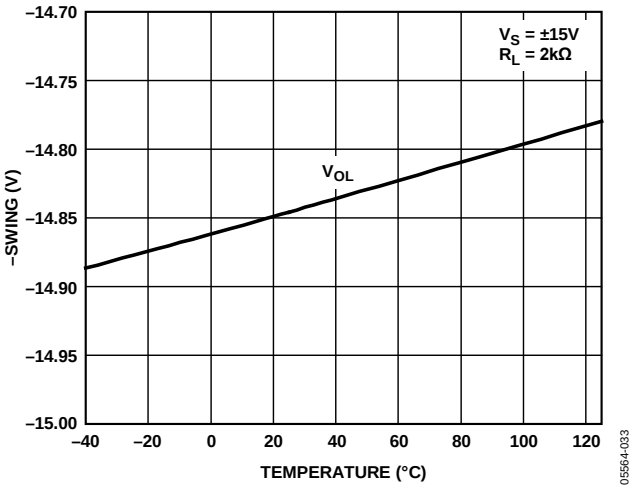


Figure 26. Swing vs. Temperature, V_{OL}

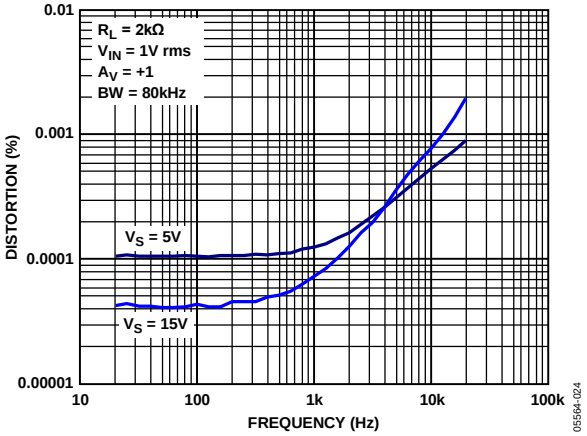


Figure 27. Distortion vs. Frequency

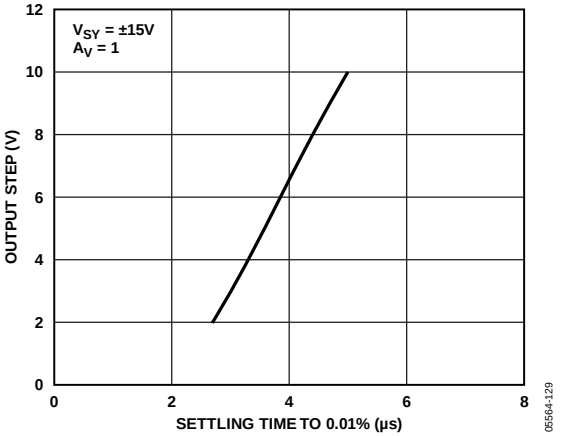


Figure 29. Output Step vs. Settling Time to 0.01%

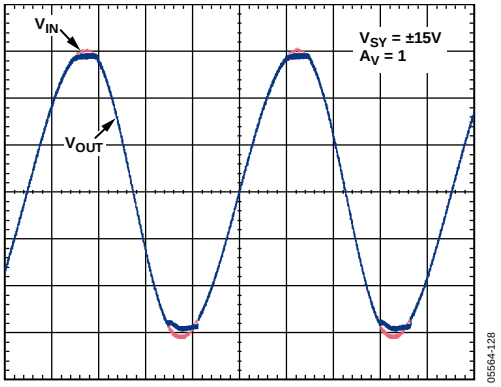


Figure 28. No Phase Reversal

TEST CIRCUIT

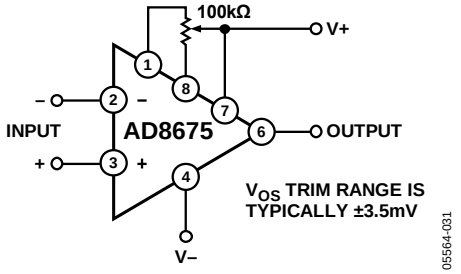


Figure 30. Optional Offset Nulling Circuit

OUTLINE DIMENSIONS

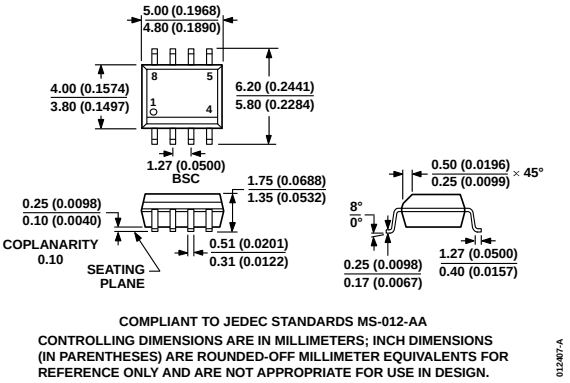


Figure 31. 8-Lead Standard Small Outline Package [SOIC_N]
Narrow Body (R-8)
Dimensions shown in millimeters and (inches)

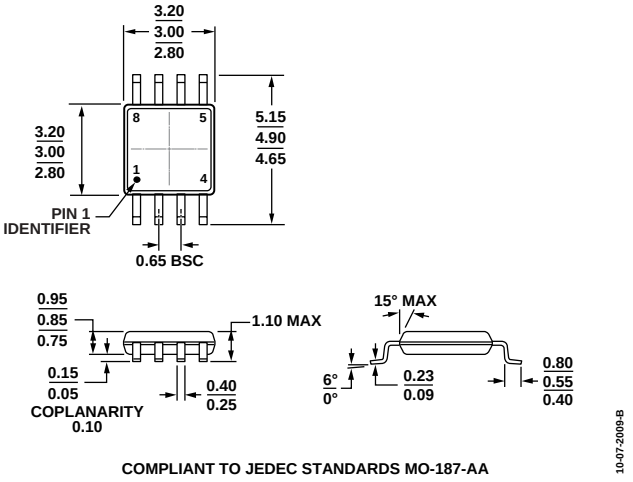


Figure 32. 8-Lead Mini Small Outline Package [MSOP]
(RM-8)
Dimensions shown in millimeters

ORDERING GUIDE

Model[1]	Temperature Range	Package Description	Package Option	Branding
AD8675ARMZ	−40°C to +125°C	8-Lead Mini Small Outline Package [MSOP]	RM-8	A08
AD8675ARMZ-REEL	−40°C to +125°C	8-Lead Mini Small Outline Package [MSOP]	RM-8	A08
AD8675ARZ	−40°C to +125°C	8-Lead Standard Small Outline Package [SOIC_N]	R-8	
AD8675ARZ-REEL	−40°C to +125°C	8-Lead Standard Small Outline Package [SOIC_N]	R-8	
AD8675ARZ-REEL7	−40°C to +125°C	8-Lead Standard Small Outline Package [SOIC_N]	R-8	

[1] Z = RoHS Compliant Part.

©2005–2012 Analog Devices, Inc. All rights reserved. Trademarks and registered trademarks are the property of their respective owners.
D05564-0-7/12(E)

17.2 Convertisseur Numérique/Analogique AD5791

1 ppm 20-Bit, ±1 LSB INL, Voltage Output DAC

Data Sheet **AD5791**

FEATURES

- 1 ppm resolution
- 1 ppm INL
- 7.5 nV/√Hz noise spectral density
- 0.19 LSB long-term linearity stability
- <0.05 ppm/°C temperature drift
- 1 µs settling time
- 1.4 nV-sec glitch impulse
- Operating temperature range: −40°C to +125°C
- 20-lead TSSOP package
- Wide power supply range up to ±16.5 V
- 35 MHz Schmitt triggered digital interface
- 1.8 V compatible digital interface

APPLICATIONS

- Medical instrumentation
- Test and measurement
- Industrial control
- High end scientific and aerospace instrumentation

FUNCTIONAL BLOCK DIAGRAM

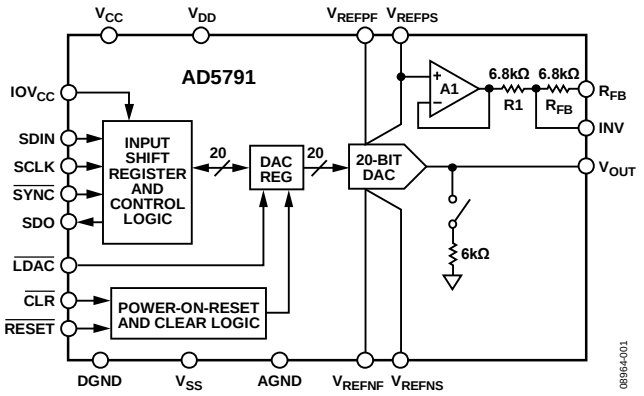


Figure 1.

Table 1. Complementary Devices

Part No.	Description
AD8675	Ultra precision, 36 V, 2.8 nV/√Hz rail-to-rail output op amp
AD8676	Ultra precision, 36 V, 2.8 nV/√Hz dual rail-to-rail output op amp
ADA4898-1	High voltage, low noise, low distortion, unity gain stable, high speed op amp

Table 2. Related Device

Part No.	Description
AD5781	18-bit, 0.5 LSB INL, voltage output DAC

GENERAL DESCRIPTION

The AD5791[1] is a single 20-bit, unbuffered voltage-output DAC that operates from a bipolar supply of up to 33 V. The AD5791 accepts a positive reference input in the range 5 V to V_{DD} − 2.5 V and a negative reference input in the range V_{SS} + 2.5 V to 0 V. The AD5791 offers a relative accuracy specification of ±1 LSB max, and operation is guaranteed monotonic with a ±1 LSB DNL maximum specification.

The part uses a versatile 3-wire serial interface that operates at clock rates up to 35 MHz and that is compatible with standard SPI, QSPI™, MICROWIRE™, and DSP interface standards. The part incorporates a power-on reset circuit that ensures the DAC output powers up to 0 V and in a known output impedance state and remains in this state until a valid write to the device takes place. The part provides an output clamp feature that places the output in a defined load state.

PRODUCT HIGHLIGHTS

1. 1 ppm Accuracy.
2. Wide Power Supply Range up to ±16.5 V.
3. Operating Temperature Range: −40°C to +125°C.
4. Low 7.5 nV/√Hz Noise Spectral Density.
5. Low 0.05 ppm/°C Temperature Drift.

[1] Protected by U.S. Patent No. 7,884,747. Other patents pending.

Rev. D Document Feedback
Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties that may result from its use. Specifications subject to change without notice. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices. Trademarks and registered trademarks are the property of their respective owners.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 ©2010–2013 Analog Devices, Inc. All rights reserved.
Technical Support www.analog.com

AD5791* Product Page Quick Links

Last Content Update: 11/01/2016

Comparable Parts

View a parametric search of comparable parts

Evaluation Kits

- AD5791 Evaluation Board

Documentation

Data Sheet

- AD5791-DSCC: Military Data Sheet
- AD5791-EP: Enhanced Product Data Sheet
- AD5791: 1ppm DAC 20-Bit, ±1LSB INL,Voltage Output DAC Data Sheet

User Guides

- AD5781/AD5791 Quick Start Guide
- UG-185: Evaluation Board for a 20-Bit, Serial Input, Voltage Output DAC

Software and Systems Requirements

- AD5780 - Microcontroller No-OS Driver
- AD5791 FMC-SDP Interposer & Evaluation Board / Xilinx KC705 Reference Design
- BeMicro FPGA Project for AD5791 with Nios driver

Tools and Simulations

- AD5791 IBIS Model

Reference Designs

- CN0191

Reference Materials

Solutions Bulletins & Brochures

- Digital to Analog Converters ICs Solutions Bulletin
- Industrial ICs Solutions Bulletin, Volume 10, Issue 8

Design Resources

- AD5791 Material Declaration
- PCN-PDN Information
- Quality And Reliability
- Symbols and Footprints

Discussions

View all AD5791 EngineerZone Discussions

Sample and Buy

Visit the product page to see pricing options

Technical Support

Submit a technical question or find your regional support number

* This page was dynamically generated by Analog Devices, Inc. and inserted into this data sheet. Note: Dynamic changes to the content on this page does not constitute a change to the revision number of the product data sheet. This content may be frequently modified.

TABLE OF CONTENTS

Features ... 1
Applications ... 1
Functional Block Diagram ... 1
General Description ... 1
Product Highlights ... 1
Revision History ... 2
Specifications ... 3
Timing Characteristics ... 5
Absolute Maximum Ratings ... 7
ESD Caution ... 7
Pin Configuration and Function Descriptions ... 8
Typical Performance Characteristics ... 9
Terminology ... 17
Theory of Operation ... 19
DAC Architecture ... 19
Serial Interface ... 19
Hardware Control Pins ... 20
On-Chip Registers ... 21
AD5791 Features ... 24
Power-On to 0 V ... 24
Configuring the AD5791 ... 24
DAC Output State ... 24
Linearity Compensation ... 24
Output Amplifier Configuration ... 24
Applications Information ... 26
Typical Operating Circuit ... 26
Outline Dimensions ... 27
Ordering Guide ... 27

REVISION HISTORY

7/13—Rev. C to Rev. D

Change to Table 4 ... 5
Deleted Figure 4, Renumbered Sequentially ... 7
Deleted Daisy-Chain Operation Section and Figure 51 ... 21

11/11—Rev. B to Rev. C

Added Figure 48; Renumbered Sequentially ... 17
Change to Ideal Transfer Function Equation ... 22

9/11—Rev. A to Rev. B

Added Patent Note ... 1
Changes to Table 3 ... 3
Changes to OPGND Description Column, Table 12 ... 23
Change to Figure 51 ... 25

8/11—Rev. 0 to Rev. A

Change to Features Section ... 1
Changes to Specifications Section, Table 3 ... 3
Deleted t_{14} Timing Specification in Table 4, Renumbered Subsequent Timing Parameters Sequentially ... 5
Changes to Figure 2 and Figure 3 ... 6
Changes to Figure 4 ... 7
Changes to Figure 42 ... 16
Changes to Figure 43 ... 16
Added Figure 44, Figure 45, and Figure 46, Renumbered Sequentially ... 16

7/10—Revision 0: Initial Version

SPECIFICATIONS

V_{DD} = 12.5 V to 16.5 V, V_{SS} = −16.5 V to −12.5 V, V_{REFP} = 10 V, V_{REFN} = −10 V, V_{CC} = 2.7 V to +5.5 V, IOV_{CC} = 1.71 V to 5.5 V, R_L = unloaded, C_L = unloaded, all specifications T_{MIN} to T_{MAX}, unless otherwise noted.

Table 3.

	A, B Version[1]				
Parameter	**Min**	**Typ**	**Max**	**Unit**	**Test Conditions/Comments**
STATIC PERFORMANCE[2]					
Resolution	20			Bits	
Integral Nonlinearity Error (Relative Accuracy)	−1	±0.25	+1	LSB	B version, V_{REFP} = +10 V, V_{REFN} = −10 V, T_A = 0°C to 105°C
	−1.5	±0.25	+1.5	LSB	B version, V_{REFP} = +10 V, V_{REFN} = −10 V
	−1.5	±0.5	+1.5	LSB	B version, V_{REFP} = 10 V, V_{REFN} = 0 V[3]
	−3	±1	+3	LSB	B version, V_{REFP} = 5 V, V_{REFN} = 0 V[3]
	−4	±2	+4	LSB	A version[4]
Differential Nonlinearity Error	−1	±0.5	+1	LSB	V_{REFP} = +10 V, V_{REFN} = −10 V
	−1.5	±0.75	+1.5	LSB	V_{REFP} = 10 V, V_{REFN} = 0 V
	−2.5	±1	+2.5	LSB	V_{REFP} = 5 V, V_{REFN} = 0 V
Linearity Error Long Term Stability[5]		0.16		LSB	After 500 hours at T_A = 125°C
		0.19		LSB	After 1000 hours at T_A = 125°C
		0.11		LSB	After 1000 hours at T_A = 100°C
Full-Scale Error	−7	±0.1	+7	LSB	V_{REFP} = +10 V, V_{REFN} = −10 V[3]
	−11	±0.25	+11	LSB	V_{REFP} = 10 V, V_{REFN} = 0 V[3]
	−21	±0.8	+21	LSB	V_{REFP} = 5 V, V_{REFN} = 0 V[3]
	−4	±0.1	+4	LSB	V_{REFP} = +10 V, V_{REFN} = −10 V[3], T_A = 0°C to 105°C
	−4	±0.25	+4	LSB	V_{REFP} = 10 V, V_{REFN} = 0 V[3], T_A = 0°C to 105°C
	−6	±0.8	+6	LSB	V_{REFP} = 5 V, V_{REFN} = 0 V[3], T_A = 0°C to 105°C
Full-Scale Error Temperature Coefficient		±0.02		ppm FSR/°C	
Zero-Scale Error	−7	±0.1	+7	LSB	V_{REFP} = +10 V, V_{REFN} = −10 V[3]
	−10	±0.15	+10	LSB	V_{REFP} = 10 V, V_{REFN} = 0 V[3]
	−21	±0.75	+21	LSB	V_{REFP} = 5 V, V_{REFN} = 0 V[3]
	−4	±0.1	+4	LSB	V_{REFP} = +10 V, V_{REFN} = −10 V[3], T_A = 0°C to 105°C
	−4	±0.15	+4	LSB	V_{REFP} = 10 V, V_{REFN} = 0 V[3], T_A = 0°C to 105°C
	−6	±0.75	+6	LSB	V_{REFP} = 5 V, V_{REFN} = 0 V[3], T_A = 0°C to 105°C
Zero-Scale Error Temperature Coefficient[3]		±0.04		ppm FSR/°C	
Gain Error	−6	±0.3	+6	ppm FSR	V_{REFP} = +10 V, V_{REFN} = −10 V[3]
	−10	±0.4	+10	ppm FSR	V_{REFP} = 10 V, V_{REFN} = 0 V[3]
	−20	±0.4	+20	ppm FSR	V_{REFP} = 5 V, V_{REFN} = 0 V[3]
	−6	±0.3	+6	ppm FSR	V_{REFP} = +10 V, V_{REFN} = −10 V[3], T_A = 0°C to 105°C
	−6	±0.4	+6	ppm FSR	V_{REFP} = 10 V, V_{REFN} = 0 V[3], T_A = 0°C to 105°C
	−7	±0.4	+7	ppm FSR	V_{REFP} = 5 V, V_{REFN} = 0 V[3], T_A = 0°C to 105°C
Gain Error Temperature Coefficient[3]		±0.04		ppm FSR/°C	
R1, R_{FB} Matching		0.01		%	
OUTPUT CHARACTERISTICS[3]					
Output Voltage Range	V_{REFN}		V_{REFP}	V	
Output Slew Rate		50		V/µs	
Output Voltage Settling Time		1		µs	10 V step to 0.02%, using the AD845 buffer in unity-gain mode
		1		µs	500 code step to ±1 LSB[6]
Output Noise Spectral Density		7.5		nV/√Hz	at 1 kHz, DAC code = midscale
		7.5		nV/√Hz	at 10 kHz, DAC code = midscale
		7.5		nV/√Hz	At 100 kHz, DAC code = midscale
Output Voltage Noise		1.1		µV p-p	DAC code = midscale, 0.1 Hz to 10 Hz bandwidth[7]

Parameter	A, B Version[1] Min	Typ	Max	Unit	Test Conditions/Comments
Midscale Glitch Impulse[8]		3.1		nV-sec	V_{REFP} = +10 V, V_{REFN} = −10 V
		1.7		nV-sec	V_{REFP} = 10 V, V_{REFN} = 0 V
		1.4		nV-sec	V_{REFP} = 5 V, V_{REFN} = 0 V
MSB Segment Glitch Impulse[8]		9.1		nV-sec	V_{REFP} = +10 V, V_{REFN} = −10 V, see Figure 42
		3.6		nV-sec	V_{REFP} = 10 V, V_{REFN} = 0 V, see Figure 43
		1.9		nV-sec	V_{REFP} = 5 V, V_{REFN} = 0 V, see Figure 44
Output Enabled Glitch Impulse		45		nV-sec	On removal of output ground clamp
Digital Feedthrough		0.4		nV-sec	
DC Output Impedance (Normal Mode)		3.4		kΩ	
DC Output Impedance (Output Clamped to Ground)		6		kΩ	
Spurious Free Dynamic Range		100		dB	1 kHz tone, 10 kHz sample rate
Total Harmonic Distortion		97		dB	1 kHz tone, 10 kHz sample rate
REFERENCE INPUTS[3]					
V_{REFP} Input Range	5		V_{DD} − 2.5 V	V	
V_{REFN} Input Range	V_{SS} + 2.5 V		0		
DC Input Impedance	5	6.6		kΩ	V_{REFP}, V_{REFN}, code dependent, typical at midscale code
Input Capacitance		15		pF	V_{REFP}, V_{REFN}
LOGIC INPUTS[3]					
Input Current[9]	−1		+1	μA	
Input Low Voltage, V_{IL}			0.3 × IOV_{CC}	V	IOV_{CC} = 1.71 V to 5.5 V
Input High Voltage, V_{IH}	0.7 × IOV_{CC}			V	IOV_{CC} = 1.71 V to 5.5 V
Pin Capacitance		5		pF	
LOGIC OUTPUT (SDO)[3]					
Output Low Voltage, V_{OL}			0.4	V	IOV_{CC} = 1.71 V to 5.5 V, sinking 1 mA
Output High Voltage, V_{OH}	IOV_{CC} − 0.5 V			V	IOV_{CC} = 1.71 V to 5.5 V, sourcing 1 mA
High Impedance Leakage Current			±1	μA	
High Impedance Output Capacitance		3		pF	
POWER REQUIREMENTS					All digital inputs at DGND or IOV_{CC}
V_{DD}	7.5		V_{SS} + 33	V	
V_{SS}	V_{DD} − 33		−2.5	V	
V_{CC}	2.7		5.5	V	
IOV_{CC}	1.71		5.5	V	IOV_{CC} ≤ V_{CC}
I_{DD}		4.2	5.2	mA	
I_{SS}		4	4.9	mA	
I_{CC}		600	900	μA	
IOI_{CC}		52	140	μA	SDO disabled
DC Power Supply Rejection Ratio[3, 10]		±0.6		μV/V	V_{DD} ± 10%, V_{SS} = 15 V
		±0.6		μV/V	V_{SS} ± 10%, V_{DD} = 15 V
AC Power Supply Rejection Ratio[3]		95		dB	V_{DD} ± 200 mV, 50 Hz/60 Hz, V_{SS} = −15 V
		95		dB	ΔV_{SS} ± 200 mV, 50 Hz/60 Hz, V_{DD} = 15 V

[1] Temperature range: −40°C to +125°C, typical at +25°C and V_{DD} = +15 V, V_{SS} = −15 V, V_{REFP} = +10 V, V_{REFN} = −10 V.
[2] Performance characterized with AD8676BRZ voltage reference buffers and AD8675ARZ output buffer.
[3] Guaranteed by design and characterization, not production tested.
[4] Valid for all voltage reference spans.
[5] Linearity error refers to both INL error and DNL error, either parameter can be expected to drift by the amount specified after the length of time specified.
[6] AD5791 configured in X2 gain mode, 25 pF compensation capacitor on AD797.
[7] Includes noise contribution from AD8676BRZ voltage reference buffers.
[8] The AD5791 is configured in bias compensation mode with a low-pass RC filter on the output. R = 300 Ω, C = 143 pF.(total capacitance seen by the output buffer, lead capacitance, and so forth).
[9] Current flowing in an individual logic pin.
[10] Includes PSRR of AD8676BRZ voltage reference buffers.

TIMING CHARACTERISTICS

V_{CC} = 2.7 V to 5.5 V; all specifications T_{MIN} to T_{MAX}, unless otherwise noted.

Table 4.

Parameter	Limit[1] IOV_{CC} = 1.71 V to 3.3 V	Limit[1] IOV_{CC} = 3.3 V to 5.5 V	Unit	Test Conditions/Comments
t_1[2]	40	28	ns min	SCLK cycle time
	92	60	ns min	SCLK cycle time (readback mode)
t_2	15	10	ns min	SCLK high time
t_3	9	5	ns min	SCLK low time
t_4	5	5	ns min	$\overline{SYNC}$ to SCLK falling edge setup time
t_5	2	2	ns min	SCLK falling edge to $\overline{SYNC}$ rising edge hold time
t_6	48	40	ns min	Minimum $\overline{SYNC}$ high time
t_7	8	6	ns min	$\overline{SYNC}$ rising edge to next SCLK falling edge ignore
t_8	9	7	ns min	Data setup time
t_9	12	7	ns min	Data hold time
t_{10}	13	10	ns min	$\overline{LDAC}$ falling edge to $\overline{SYNC}$ falling edge
t_{11}	20	16	ns min	$\overline{SYNC}$ rising edge to $\overline{LDAC}$ falling edge
t_{12}	14	11	ns min	$\overline{LDAC}$ pulse width low
t_{13}	130	130	ns typ	$\overline{LDAC}$ falling edge to output response time
t_{14}	130	130	ns typ	$\overline{SYNC}$ rising edge to output response time ($\overline{LDAC}$ tied low)
t_{15}	50	50	ns min	$\overline{CLR}$ pulse width low
t_{16}	140	140	ns typ	$\overline{CLR}$ pulse activation time
t_{17}	0	0	ns min	$\overline{SYNC}$ falling edge to first SCLK rising edge
t_{18}	65	60	ns max	$\overline{SYNC}$ rising edge to SDO tristate (C_L = 50 pF)
t_{19}	62	45	ns max	SCLK rising edge to SDO valid (C_L = 50 pF)
t_{20}	0	0	ns min	$\overline{SYNC}$ rising edge to SCLK rising edge ignore
t_{21}	35	35	ns typ	$\overline{RESET}$ pulse width low
t_{22}	150	150	ns typ	$\overline{RESET}$ pulse activation time

[1] All input signals are specified with $t_R = t_F$ = 1 ns/V (10% to 90% of IOV_{CC}) and timed from a voltage level of $(V_{IL} + V_{IH})/2$.
[2] Maximum SCLK frequency is 35 MHz for write mode and 16 MHz for readback and daisy-chain modes.

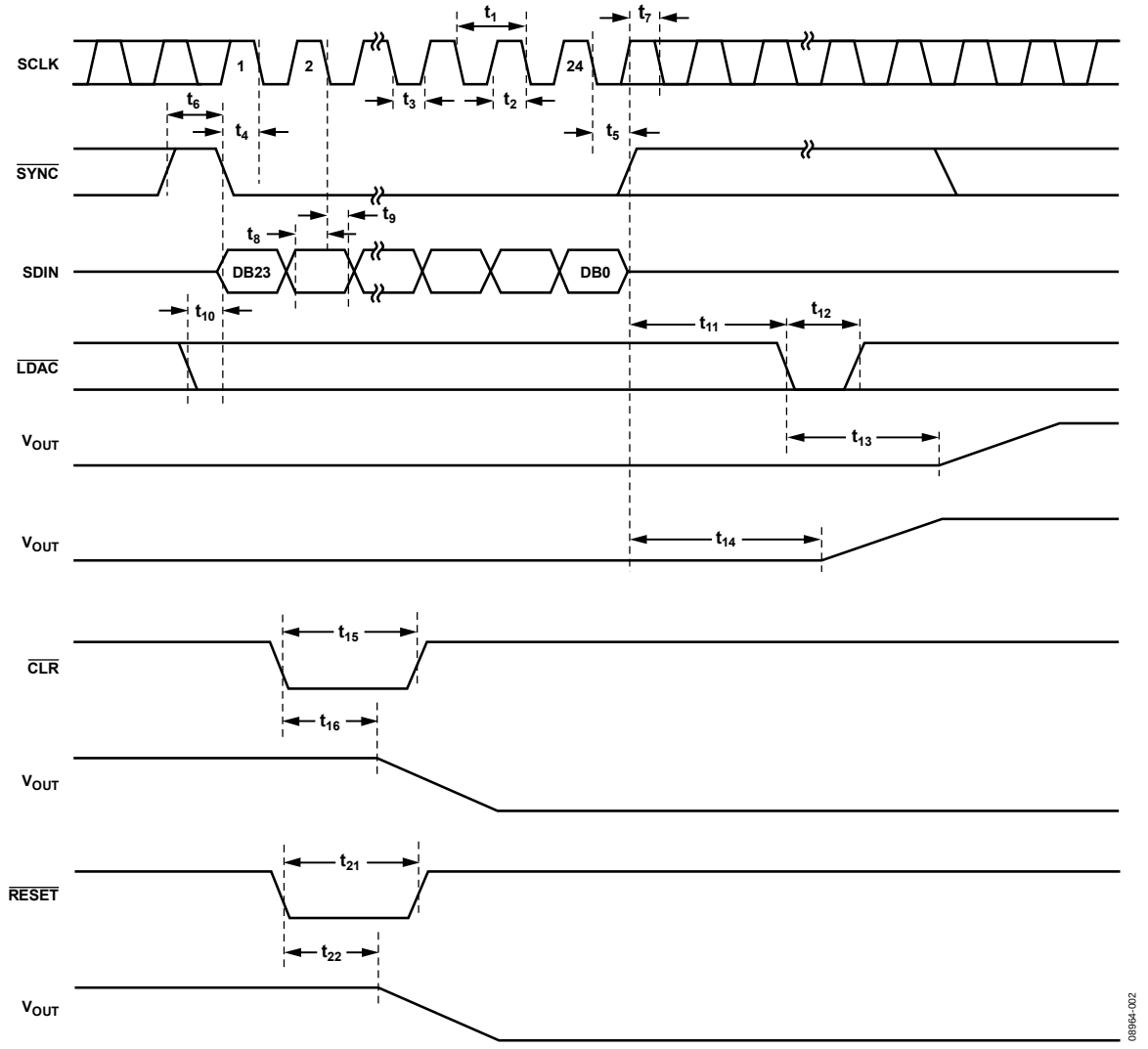


Figure 2. Write Mode Timing Diagram

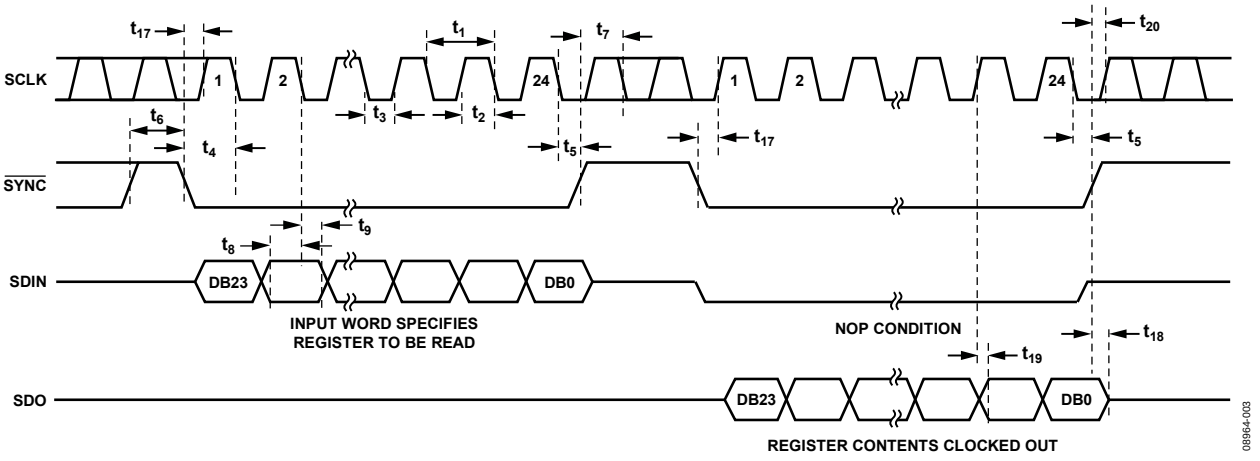


Figure 3. Readback Mode Timing Diagram

ABSOLUTE MAXIMUM RATINGS

T_A = 25°C, unless otherwise noted. Transient currents of up to 100 mA do not cause SCR latch-up.

Table 5.

Parameter	Rating
V_{DD} to AGND	−0.3 V to +34 V
V_{SS} to AGND	−34 V to +0.3 V
V_{DD} to V_{SS}	−0.3 V to +34 V
V_{CC} to DGND	−0.3 V to +7 V
IOV_{CC} to DGND	−0.3 V to V_{CC} + 0.3 V or +7 V (whichever is less)
Digital Inputs to DGND	−0.3 V to IOV_{CC} + 0.3 V or +7 V (whichever is less)
V_{OUT} to AGND	−0.3 V to V_{DD} + 0.3 V
V_{REFPF} to AGND	−0.3 V to V_{DD} + 0.3 V
V_{REFPS} to AGND	−0.3 V to V_{DD} + 0.3 V
V_{REFNF} to AGND	V_{SS} − 0.3 V to + 0.3 V
V_{REFNS} to AGND	V_{SS} − 0.3 V to + 0.3 V
DGND to AGND	−0.3 V to +0.3 V
Operating Temperature Range, T_A	
Industrial	−40°C to + 125°C
Storage Temperature Range	−65°C to +150°C
Maximum Junction Temperature, T_J max	150°C
Power Dissipation	(T_J max − T_A)/θ_{JA}
TSSOP Package	
θ_{JA} Thermal Impedance	143°C/W
θ_{JC} Thermal Impedance	45°C/W
Lead Temperature	JEDEC industry standard
Soldering	J-STD-020
ESD (Human Body Model)	1.5 kV

Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; functional operation of the device at these or any other conditions above those indicated in the operational section of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

This device is a high performance integrated circuit with an ESD rating of 1.5 kV, and it is ESD sensitive. Proper precautions should be taken for handling and assembly.

ESD CAUTION

ESD (electrostatic discharge) sensitive device. Charged devices and circuit boards can discharge without detection. Although this product features patented or proprietary protection circuitry, damage may occur on devices subjected to high energy ESD. Therefore, proper ESD precautions should be taken to avoid performance degradation or loss of functionality.

PIN CONFIGURATION AND FUNCTION DESCRIPTIONS

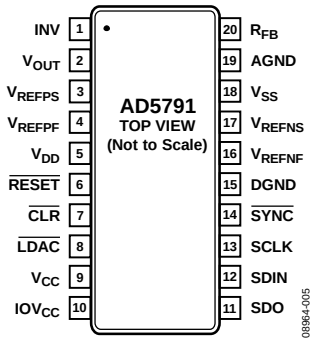


Figure 4. Pin Configuration

Table 6. Pin Function Descriptions

Pin No.	Mnemonic	Description
1	INV	Connection to Inverting Input of External Amplifier. See the AD5791 Features section for further details.
2	V_{OUT}	Analog Output Voltage.
3	V_{REFPS}	Positive Reference Sense Voltage Input. A voltage range of 5 V to V_{DD} − 2.5 V can be connected. A unity gain amplifier must be connected at this pin in conjunction with the V_{REFPF} pin. See the AD5791 Features section for further details.
4	V_{REFPF}	Positive Reference Force Voltage Input. A voltage range of 5 V to V_{DD} − 2.5 V can be connected. A unity gain amplifier must be connected at this pin in conjunction with the V_{REFPS} pin. See the AD5791 Features section for further details.
5	V_{DD}	Positive Analog Supply Connection. A voltage range of 7.5 V to 16.5 V can be connected, V_{DD} should be decoupled to AGND.
6	$\overline{RESET}$	Active Low Reset Logic Input Pin. Asserting this pin returns the AD5791 to its power-on status.
7	$\overline{CLR}$	Active Low Clear Logic Input Pin. Asserting this pin sets the DAC register to a user defined value (see Table 13) and updates the DAC output. The output value depends on the DAC register coding that is being used, either binary or twos complement.
8	$\overline{LDAC}$	Active Low Load DAC Logic Input Pin. This is used to update the DAC register and consequently, the analog output. When tied permanently low, the output is updated on the rising edge of $\overline{SYNC}$. If $\overline{LDAC}$ is held high during the write cycle, the input register is updated, but the output update is held off until the falling edge of $\overline{LDAC}$. The $\overline{LDAC}$ pin should not be left unconnected.
9	V_{CC}	Digital Supply Connection. A voltage range of 2.7 V to 5.5 V can be connected. V_{CC} should be decoupled to DGND.
10	IOV_{CC}	Digital Interface Supply Pin. Digital threshold levels are referenced to the voltage applied to this pin. A voltage in the range of 1.71 V to 5.5 V can be connected. IOV_{CC} should not be allowed to exceed V_{CC}.
11	SDO	Serial Data Output Pin. Data is clocked out on the rising edge of the serial clock input.
12	SDIN	Serial Data Input Pin. This device has a 24-bit shift register. Data is clocked into the register on the falling edge of the serial clock input.
13	SCLK	Serial Clock Input. Data is clocked into the input shift register on the falling edge of the serial clock input. Data can be transferred at clock rates of up to 35 MHz.
14	$\overline{SYNC}$	Active Low Digital Interface Synchronization Input Pin. This is the frame synchronization signal for the input data. When $\overline{SYNC}$ is low, it enables the input shift register, and data is then transferred in on the falling edges of the following clocks. The input shift register is updated on the rising edge of $\overline{SYNC}$.
15	DGND	Ground Reference Pin for Digital Circuitry.
16	V_{REFNF}	Negative Reference Force Voltage Input. A voltage range of V_{SS} + 2.5 V to 0 V can be connected. A unity gain amplifier must be connected at this pin, in conjunction with the V_{REFNS} pin. See the AD5791 Features section for further details.
17	V_{REFNS}	Negative Reference Sense Voltage Input. A voltage range of V_{SS} + 2.5 V to 0 V can be connected. A unity gain amplifier must be connected at this pin, in conjunction with the V_{REFNF} pin. See the AD5791 Features section for further details.
18	V_{SS}	Negative Analog Supply Connection. A voltage range of −16.5 V to −2.5 V can be connected. V_{SS} should be decoupled to AGND.
19	AGND	Ground Reference Pin for Analog Circuitry.
20	R_{FB}	Feedback Connection for External Amplifier. See the AD5791 Features section for further details.

TYPICAL PERFORMANCE CHARACTERISTICS

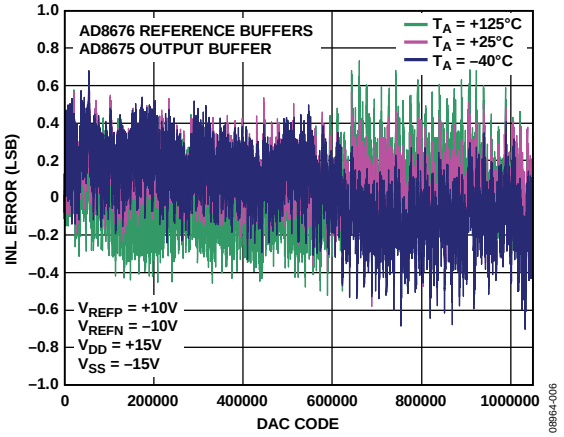


Figure 5. Integral Nonlinearity Error vs. DAC Code, ±10 V Span

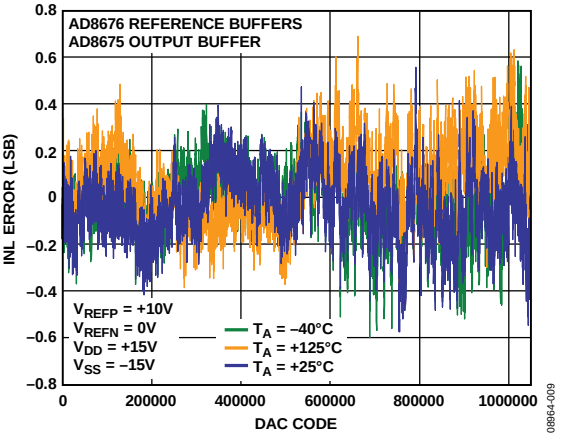


Figure 8. Integral Nonlinearity Error vs. DAC Code, ±10 V Span, X2 Gain Mode

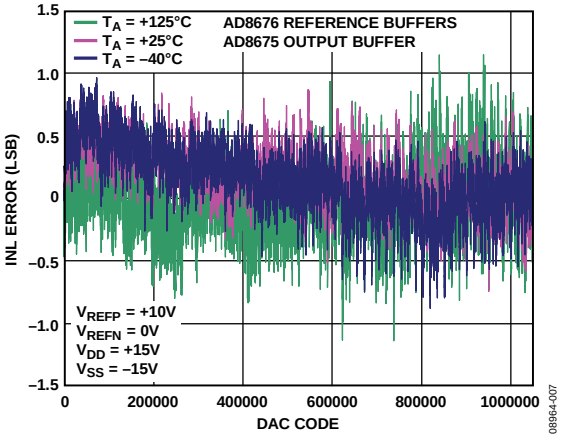


Figure 6. Integral Nonlinearity Error vs. DAC Code, 10 V Span

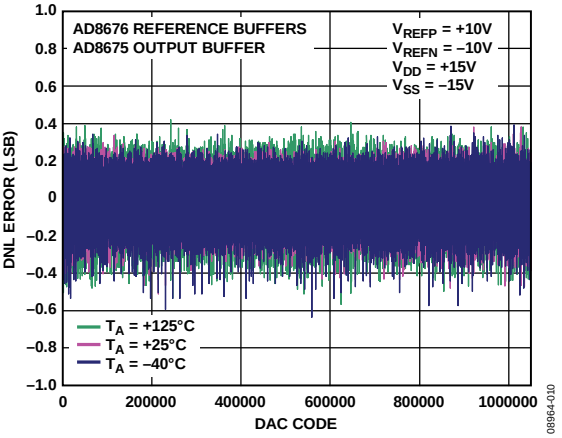


Figure 9. Differential Nonlinearity Error vs. DAC Code, ±10 V Span

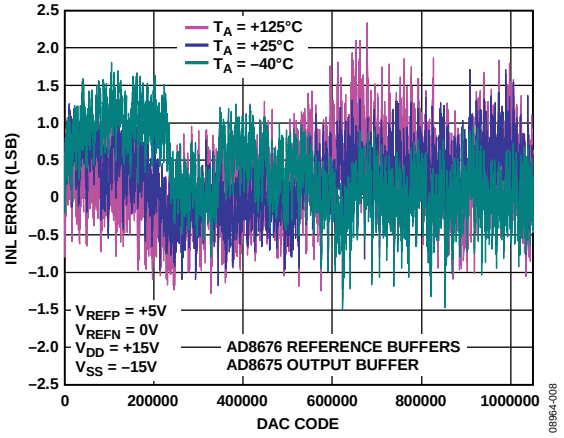


Figure 7. Integral Nonlinearity Error vs. DAC Code, 5 V Span

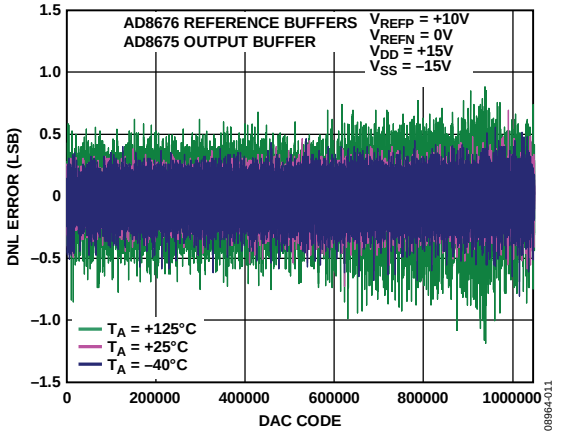


Figure 10. Differential Nonlinearity Error vs. DAC Code, 10 V Span

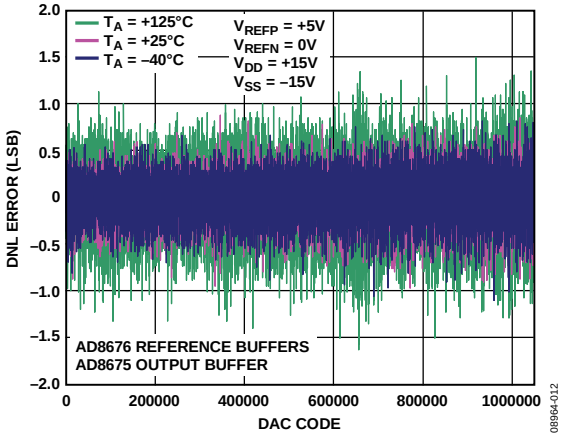


Figure 11. Differential Nonlinearity Error vs. DAC Code, 5 V Span

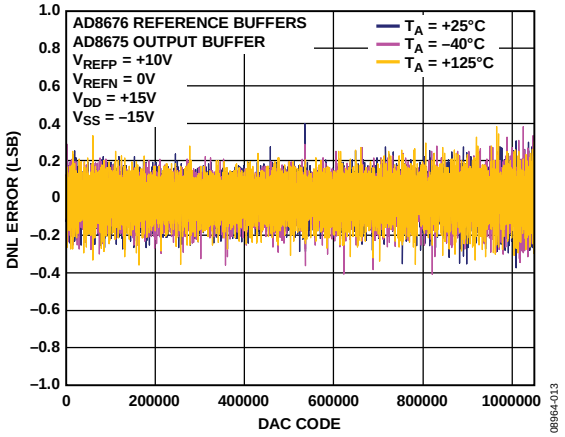


Figure 12. Differential Nonlinearity Error vs. DAC Code, ±10 V Span, X2 Gain Mode

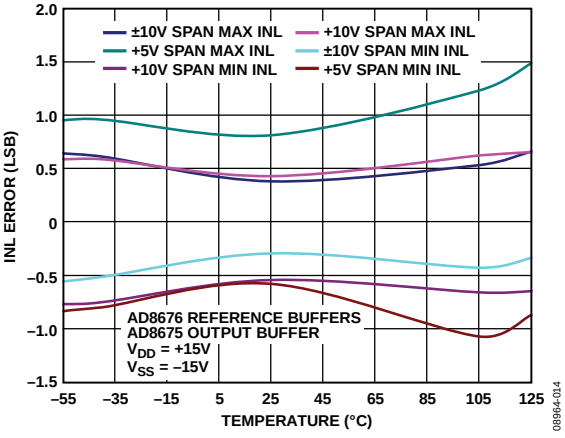


Figure 13. Integral Nonlinearity Error vs. Temperature

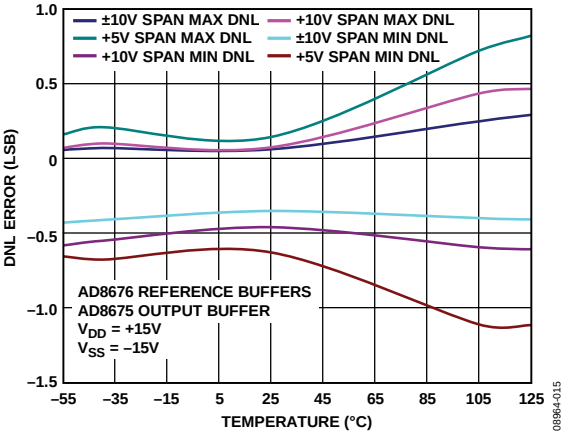


Figure 14. Differential Nonlinearity Error vs. Temperature

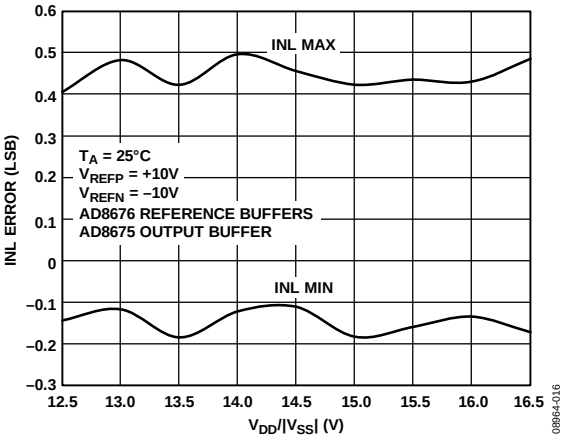


Figure 15. Integral Nonlinearity Error vs. Supply Voltage, ±10 V Span

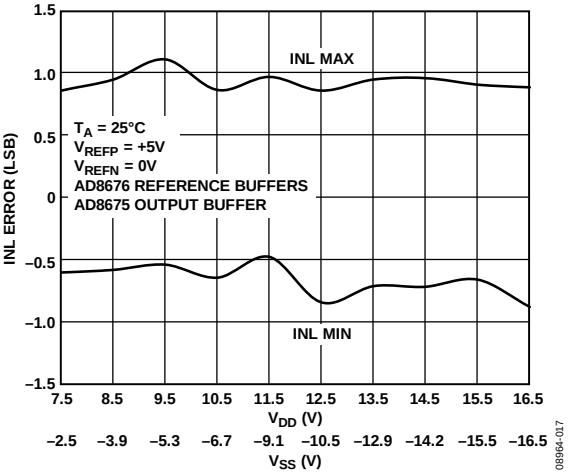


Figure 16. Integral Nonlinearity Error vs. Supply Voltage, 5 V Span

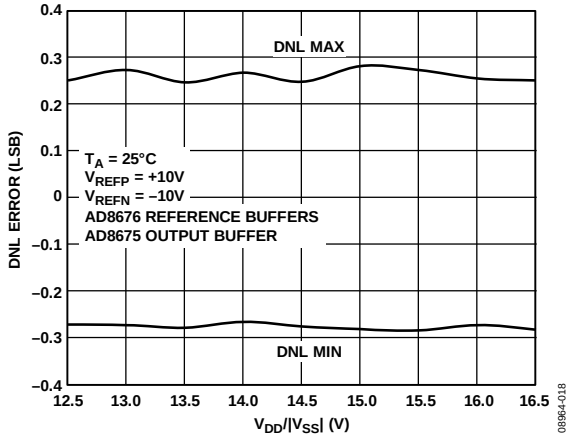


Figure 17. Differential Nonlinearity Error vs. Supply Voltage, ±10 V Span

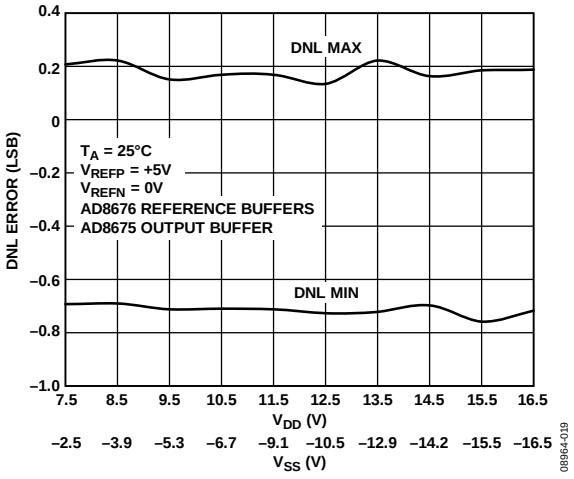


Figure 18. Differential Nonlinearity Error vs. Supply Voltage, 5 V Span

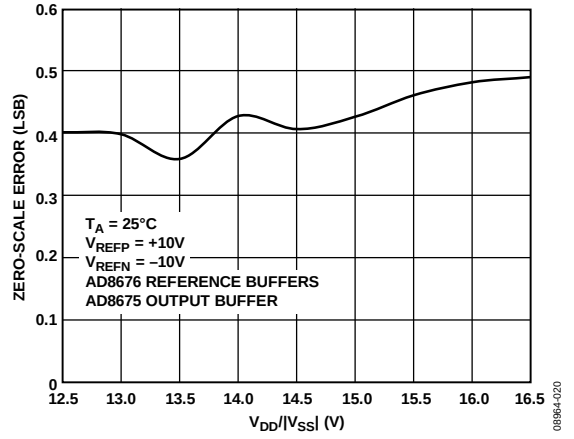


Figure 19. Zero-Scale Error vs. Supply Voltage, ±10 V Span

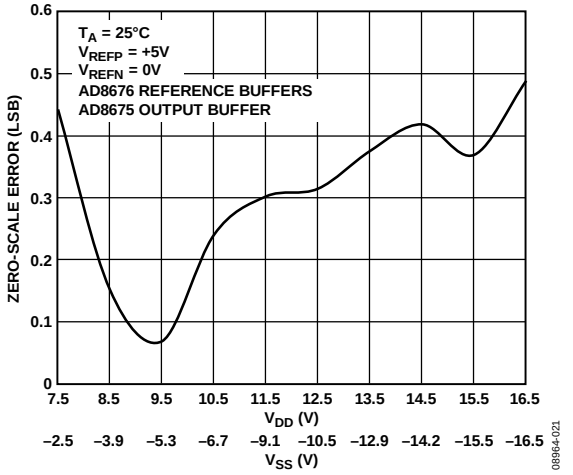


Figure 20. Zero-Scale Error vs. Supply Voltage, 5 V Span

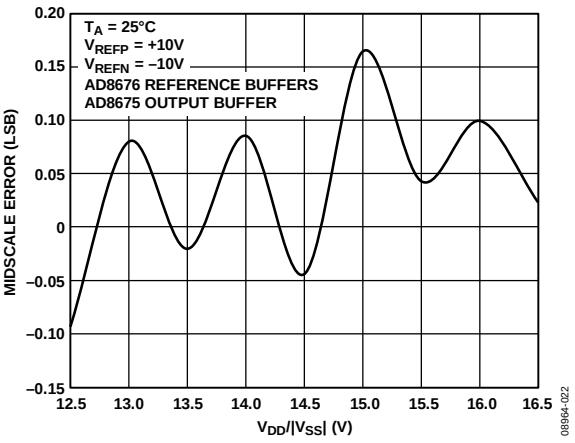


Figure 21. Midscale Error vs. Supply Voltage, ±10 V Span

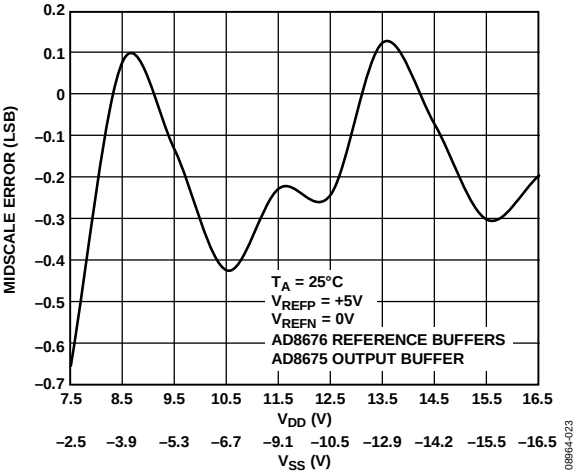


Figure 22. Midscale Error vs. Supply Voltage, 5 V Span

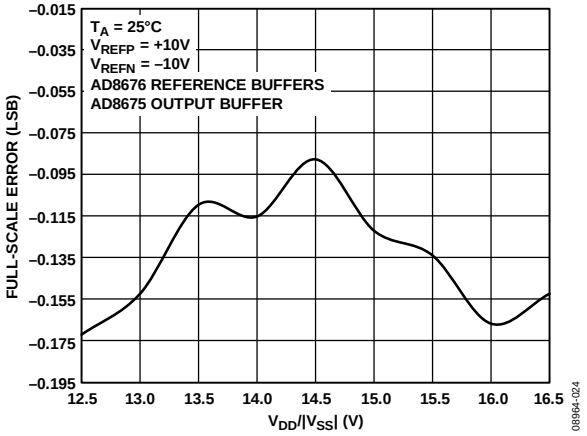


Figure 23. Full-Scale Error vs. Supply Voltage, ±10 V Span

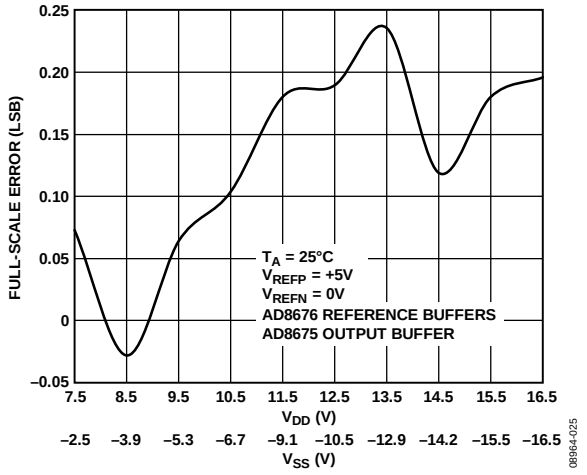


Figure 24. Full-Scale Error vs. Supply Voltage, 5 V Span

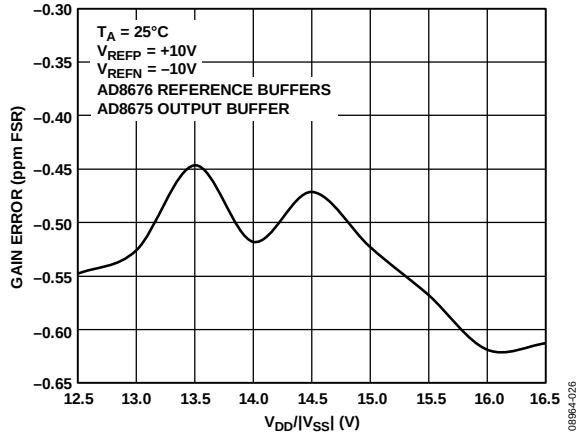


Figure 25. Gain Error vs. Supply Voltage, ±10 V Span

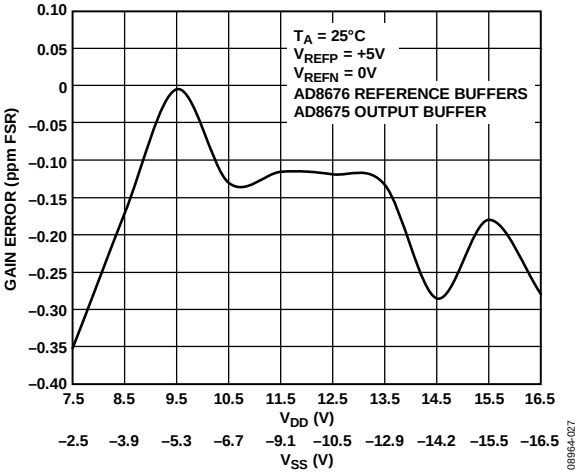


Figure 26. Gain Error vs. Supply Voltage, 5 V Span

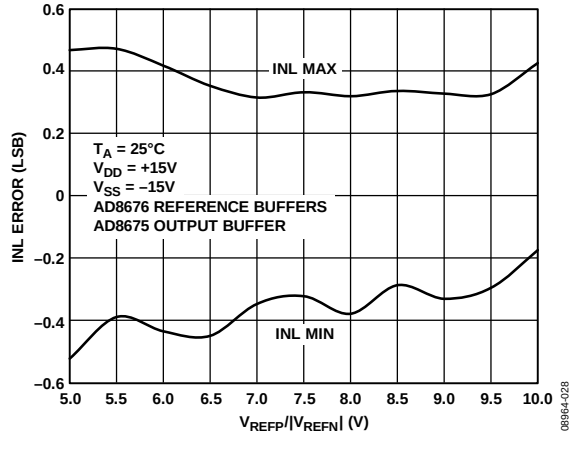


Figure 27. Integral Nonlinearity Error vs. Reference Voltage

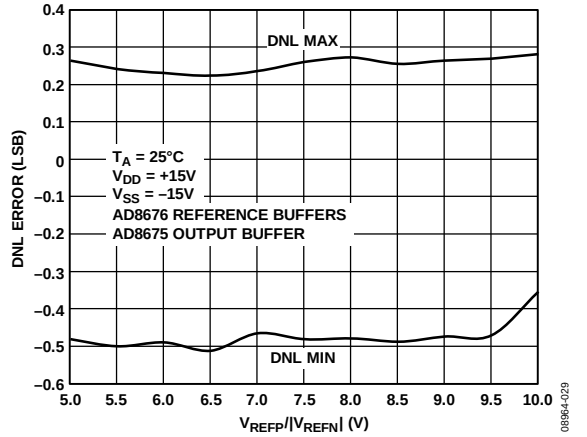


Figure 28. Differential Nonlinearity Error vs. Reference Voltage

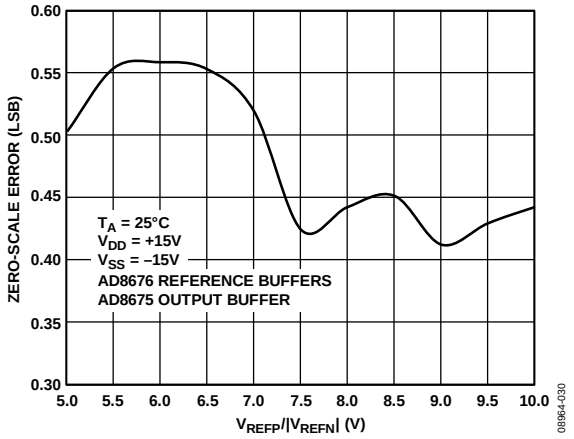


Figure 29. Zero-Scale Error vs. Reference Voltage

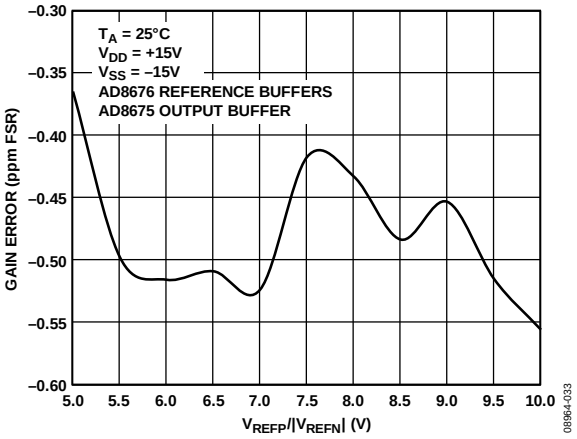


Figure 32. Gain Error vs. Reference Voltage

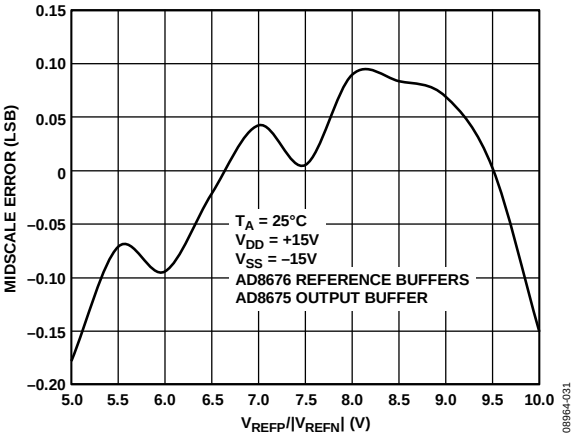


Figure 30. Midscale Error vs. Reference Voltage

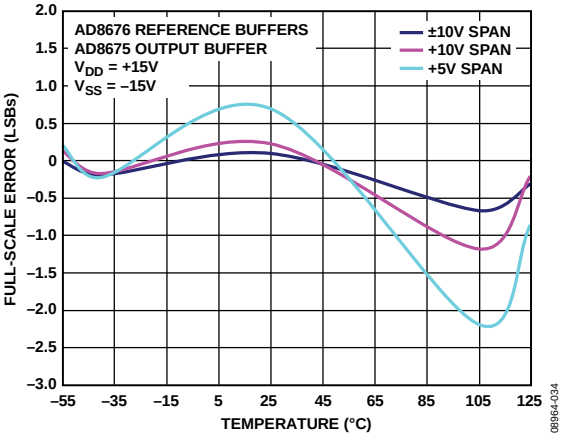


Figure 33. Full-Scale Error vs. Temperature

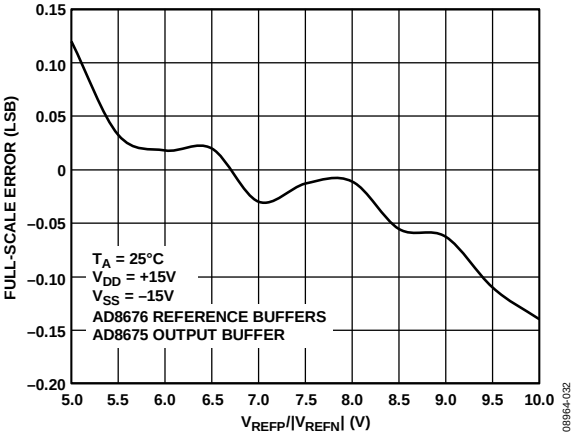


Figure 31. Full-Scale Error vs. Reference Voltage

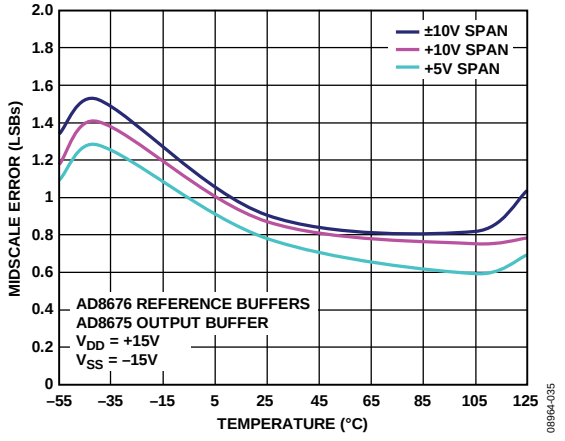


Figure 34. Midscale Error vs. Temperature

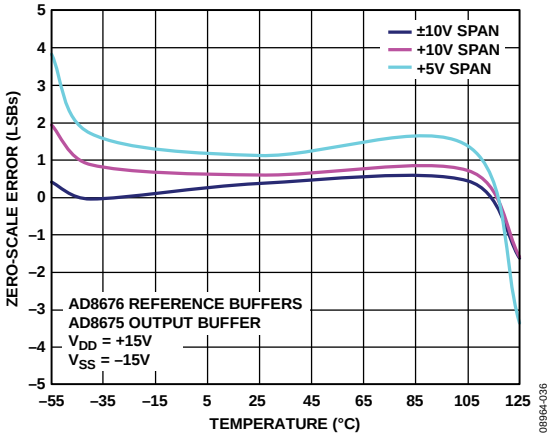


Figure 35. Zero-Scale Error vs. Temperature

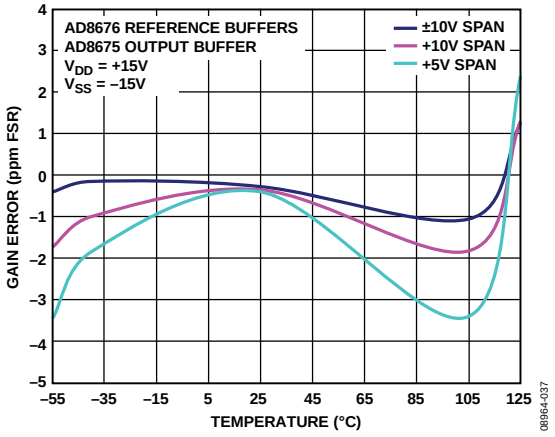


Figure 36. Gain Error vs. Temperature

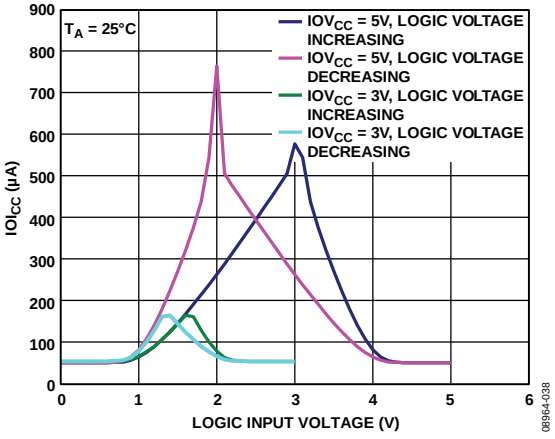


Figure 37. IOI_{CC} vs. Logic Input Voltage

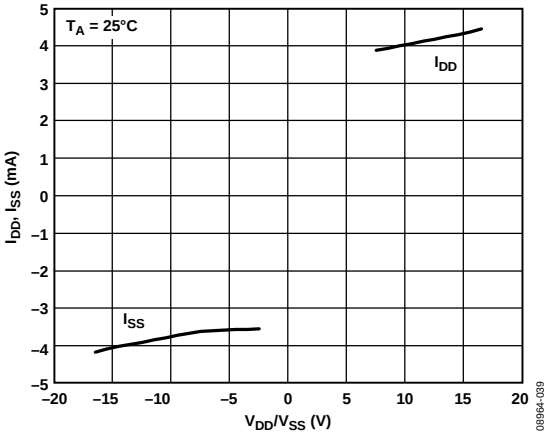


Figure 38. Power Supply Currents vs. Power Supply Voltages

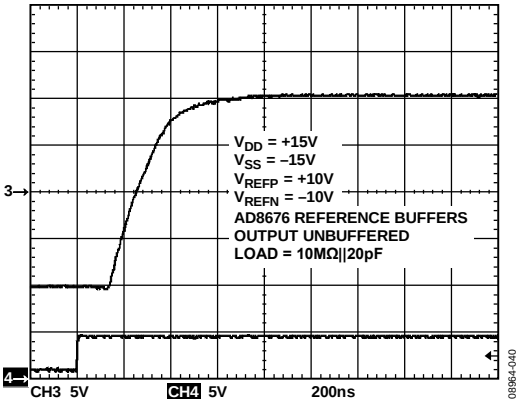


Figure 39. Rising Full-Scale Voltage Step

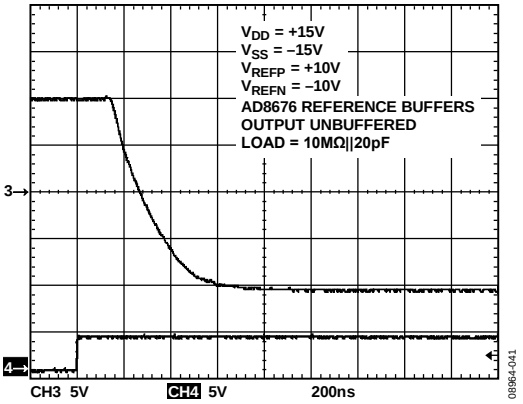


Figure 40. Falling Full-Scale Voltage Step

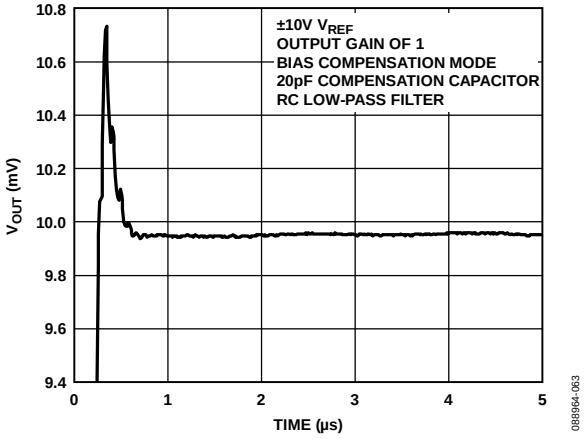


Figure 41. 500 Code Step Settling Time

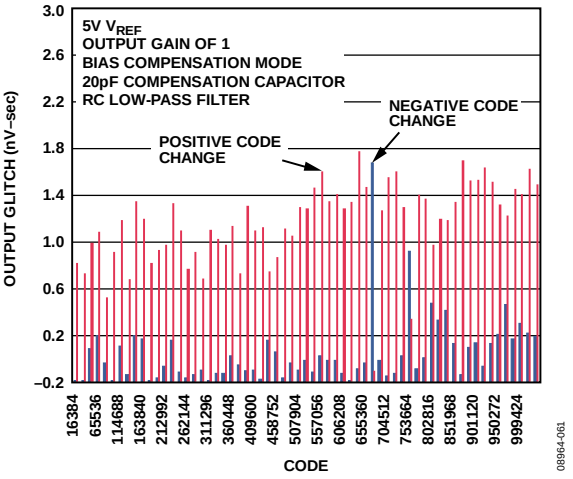


Figure 44. 6 MSB Segment Glitch Energy for +5 V V_{REF}

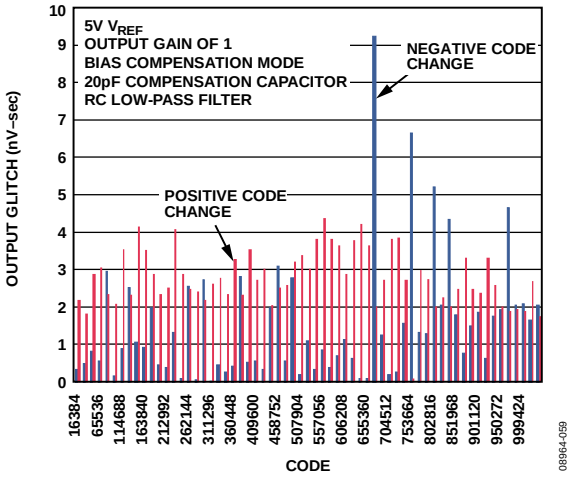


Figure 42. 6 MSB Segment Glitch Energy for ±10 V V_{REF}

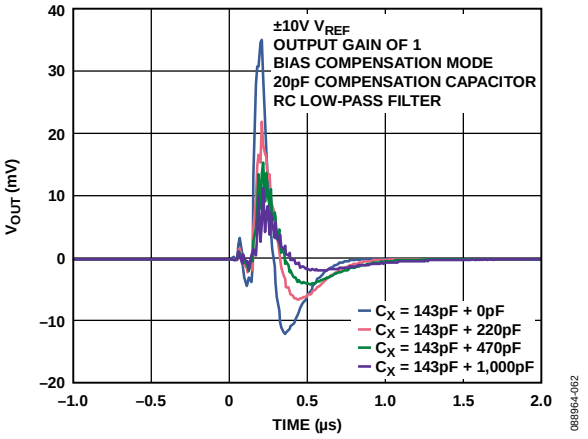


Figure 45. Midscale Peak-to-Peak Glitch for ±10 V

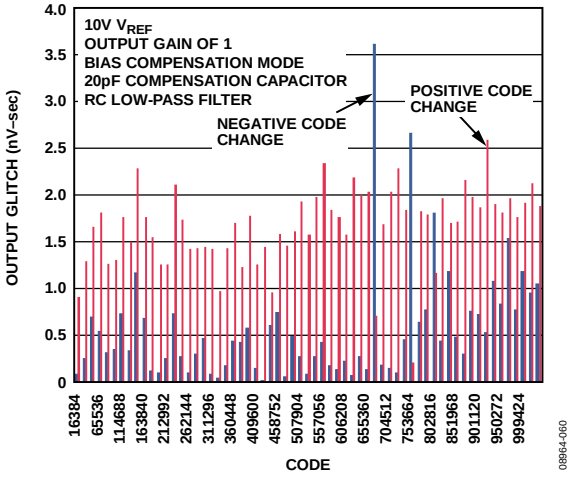


Figure 43. 6 MSB Segment Glitch Energy for +10 V V_{REF}

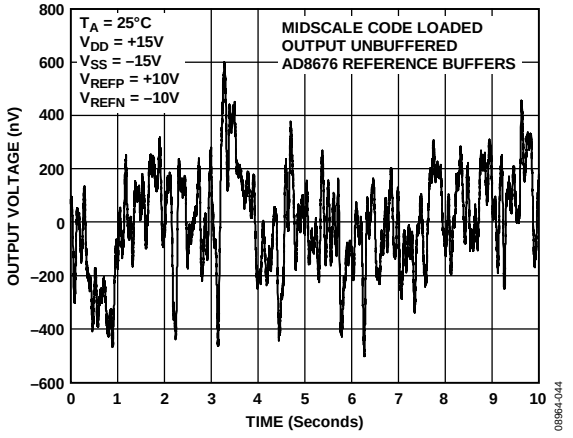


Figure 46. Voltage Output Noise, 0.1 Hz to 10 Hz Bandwidth

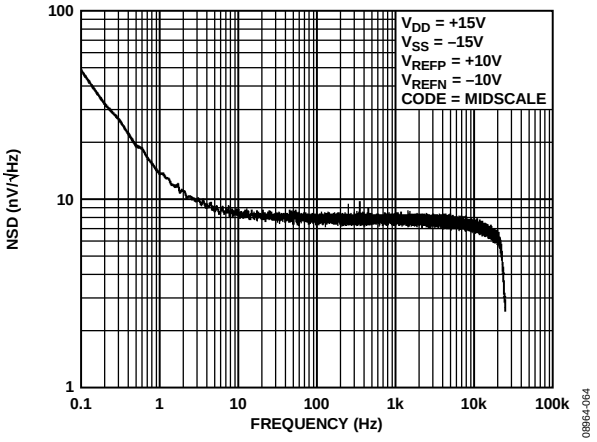


Figure 47. Noise Spectral Density vs. Frequency

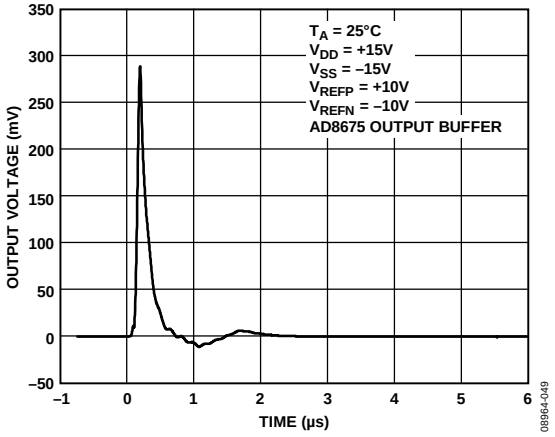


Figure 48. Glitch Impulse on Removal of Output Clamp

TERMINOLOGY

Relative Accuracy
Relative accuracy, or integral nonlinearity (INL), is a measure of the maximum deviation, in LSB, from a straight line passing through the endpoints of the DAC transfer function. A typical INL error vs. code plot is shown in Figure 5.

Differential Nonlinearity (DNL)
Differential nonlinearity is the difference between the measured change and the ideal 1 LSB change between any two adjacent codes. A specified differential nonlinearity of ±1 LSB maximum ensures monotonicity. This DAC is guaranteed monotonic. A typical DNL error vs. code plot is shown in Figure 9.

Linearity Error Long Term Stability
Linearity error long term stability is a measure of the stability of the linearity of the DAC over a long period of time. It is specified in LSB for a time period of 500 hours and 1000 hours at an elevated ambient temperature.

Zero-Scale Error
Zero-scale error is a measure of the output error when zero-scale code (0x00000) is loaded to the DAC register. Ideally, the output voltage should be V_{REFNS}. Zero-scale error is expressed in LSBs.

Zero-Scale Error Temperature Coefficient
Zero-scale error temperature coefficient is a measure of the change in zero-scale error with a change in temperature. It is expressed in ppm FSR/°C.

Full-Scale Error
Full-scale error is a measure of the output error when full-scale code (0x3FFFF) is loaded to the DAC register. Ideally, the output voltage should be V_{REFPS} − 1 LSB. Full-scale error is expressed in LSBs.

Full-Scale Error Temperature Coefficient
Full-scale error temperature coefficient is a measure of the change in full-scale error with a change in temperature. It is expressed in ppm FSR/°C.

Gain Error
Gain error is a measure of the span error of the DAC. It is the deviation in slope of the DAC transfer characteristic from ideal, expressed in ppm of the full-scale range.

Gain Error Temperature Coefficient
Gain error temperature coefficient is a measure of the change in gain error with a change in temperature. It is expressed in ppm FSR/°C.

Midscale Error
Midscale error is a measure of the output error when midscale code (0x20000) is loaded to the DAC register. Ideally, the output voltage should be $(V_{REFPS} - V_{REFNS})/2 + V_{REFNS}$. Midscale error is expressed in LSBs.

Midscale Error Temperature Coefficient
Midscale error temperature coefficient is a measure of the change in midscale error with a change in temperature. It is expressed in ppm FSR/°C.

Output Slew Rate
Slew rate is a measure of the limitation in the rate of change of the output voltage. The slew rate of the AD5791 output voltage is determined by the capacitive load presented to the V_{OUT} pin. The capacitive load in conjunction with the 3.4 kΩ output impedance of the AD5791 set the slew rate. Slew rate is measured from 10% to 90% of the output voltage change and is expressed in V/µs.

Output Voltage Settling Time
Output voltage settling time is the amount of time it takes for the output voltage to settle to a specified level for a specified change in voltage. For fast settling applications, a high speed buffer amplifier is required to buffer the load from the 3.4 kΩ output impedance of the AD5791, in which case it is the amplifier that determines the settling time.

Digital-to-Analog Glitch Impulse
Digital-to-analog glitch impulse is the impulse injected into the analog output when the input code in the DAC register changes state. It is specified as the area of the glitch in nV-sec and is measured when the digital input code is changed by 1 LSB at the major carry transition (see Figure 42).

Output Enabled Glitch Impulse
Output enabled glitch impulse is the impulse injected into the analog output when the clamp to ground on the DAC output is removed. It is specified as the area of the glitch in nV-sec (see Figure 48).

Digital Feedthrough
Digital feedthrough is a measure of the impulse injected into the analog output of the DAC from the digital inputs of the DAC but is measured when the DAC output is not updated. It is specified in nV-sec and measured with a full-scale code change on the data bus, that is, from all 0s to all 1s, and vice versa.

Spurious Free Dynamic Range (SFDR)

Spurious free dynamic range is the usable dynamic range of a DAC before spurious noise interferes or distorts the fundamental signal. It is measured by the difference in amplitude between the fundamental and the largest harmonically or nonharmonically related spur from dc to full Nyquist bandwidth (half the DAC sampling rate, or $f_S/2$). SFDR is measured when the signal is a digitally generated sine wave.

Total Harmonic Distortion (THD)
Total harmonic distortion is the ratio of the rms sum of the harmonics of the DAC output to the fundamental value Only the second to fifth harmonics are included.

DC Power Supply Rejection Ratio
DC power supply rejection ratio is a measure of the rejection of the output voltage to dc changes in the power supplies applied to the DAC. It is measured for a given dc change in power supply voltage and is expressed in μV/V.

AC Power Supply Rejection Ratio (AC PSRR)
AC power supply rejection ratio is a measure of the rejection of the output voltage to ac changes in the power supplies applied to the DAC. It is measured for a given amplitude and frequency change in power supply voltage and is expressed in decibels.

THEORY OF OPERATION

The AD5791 is a high accuracy, fast settling, single, 20-bit, serial input, voltage output DAC. It operates from a V_{DD} supply voltage of 7.5 V to 16.5 V and a V_{SS} supply of −16.5 V to −2.5 V. Data is written to the AD5791 in a 24-bit word format via a 3-wire serial interface. The AD5791 incorporates a power-on reset circuit that ensures the DAC output powers up to 0 V with the V_{OUT} pin clamped to AGND through a ~6 kΩ internal resistor.

DAC ARCHITECTURE

The architecture of the AD5791 consists of two matched DAC sections. A simplified circuit diagram is shown in Figure 49. The six MSBs of the 20-bit data-word are decoded to drive 63 switches, E0 to E62. Each of these switches connects one of 63 matched resistors to either the V_{REFP} or V_{REFN} voltage. The remaining 14 bits of the data-word drive the S0 to S13 switched of a 14-bit voltage mode R-2R ladder network. To ensure performance to specification, the reference inputs must be force sensed with external amplifiers.

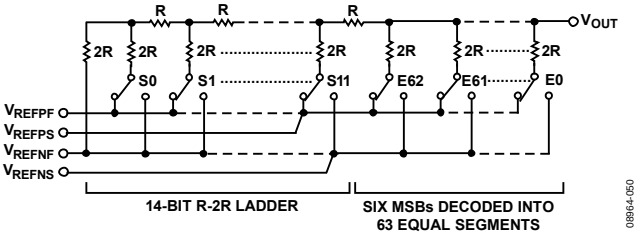


Figure 49. DAC Ladder Structure

SERIAL INTERFACE

The AD5791 has a 3-wire serial interface ($\overline{SYNC}$, SCLK, and SDIN) that is compatible with SPI, QSPI, and MICROWIRE interface standards, as well as most DSPs (see Figure 2 for a timing diagram).

Input Shift Register

The input shift register is 24 bits wide. Data is loaded into the device MSB first as a 24-bit word under the control of a serial clock input, SCLK, which can operate at up to 50 MHz. The input register consists of a R/$\overline{W}$ bit, three address bits, and twenty register bits as shown in Table 7. The timing diagram for this operation is shown in Figure 2.

Table 7. Input Shift Register Format

MSB					LSB
DB23	**DB22**	**DB21**	**DB20**	**DB19**	**DB0**
R/$\overline{W}$	Register address			Register data	

Table 8. Decoding the Input Shift Register

R/$\overline{W}$	Register Address			Description
X[1]	0	0	0	No operation (NOP; used in readback operations
0	0	0	1	Write to the DAC register
0	0	1	0	Write to the control register
0	0	1	1	Write to the clearcode register
0	1	0	0	Write to the software control register
1	0	0	1	Read from the DAC register
1	0	1	0	Read from the control register
1	0	1	1	Read from the clearcode register

[1] X is don't care.

Standalone Operation

The serial interface works with both a continuous and noncontinuous serial clock. A continuous SCLK source can be used only if $\overline{\text{SYNC}}$ is held low for the correct number of clock cycles. In gated clock mode, a burst clock containing the exact number of clock cycles must be used, and $\overline{\text{SYNC}}$ must be taken high after the final clock to latch the data. The first falling edge of $\overline{\text{SYNC}}$ starts the write cycle. Exactly 24 falling clock edges must be applied to SCLK before $\overline{\text{SYNC}}$ is brought high again. If $\overline{\text{SYNC}}$ is brought high before the 24th falling SCLK edge, the data written is invalid. If more than 24 falling SCLK edges are applied before $\overline{\text{SYNC}}$ is brought high, the input data is also invalid. The input shift register is updated on the rising edge of $\overline{\text{SYNC}}$. For another serial transfer to take place, $\overline{\text{SYNC}}$ must be brought low again. After the end of the serial data transfer, data is automatically transferred from the input shift register to the addressed register. Once the write cycle is complete, the output can be updated by taking $\overline{\text{LDAC}}$ low while $\overline{\text{SYNC}}$ is high.

Readback

The contents of all the on-chip registers can be read back via the SDO pin. Table 8 outlines how the registers are decoded. After a register has been addressed for a read, the next 24 clock cycles clock the data out on the SDO pin. The clocks must be applied while $\overline{\text{SYNC}}$ is low. When $\overline{\text{SYNC}}$ is returned high, the SDO pin is placed in tristate. For a read of a single register, the NOP function can be used to clock out the data. Alternatively, if more than one register is to be read, the data of the first register to be addressed can be clocked out at the same time the second register to be read is being addressed. The SDO pin must be enabled to complete a readback operation. The SDO pin is enabled by default.

HARDWARE CONTROL PINS

Load DAC Function ($\overline{\text{LDAC}}$)

After data has been transferred into the input register of the DAC, there are two ways to update the DAC register and DAC output. Depending on the status of both $\overline{\text{SYNC}}$ and $\overline{\text{LDAC}}$, one of two update modes is selected: synchronous DAC updating or asynchronous DAC updating

Synchronous DAC Update

In this mode, $\overline{\text{LDAC}}$ is held low while data is being clocked into the input shift register. The DAC output is updated on the rising edge of $\overline{\text{SYNC}}$.

Asynchronous DAC Update

In this mode, $\overline{\text{LDAC}}$ is held high while data is being clocked into the input shift register. The DAC output is asynchronously updated by taking $\overline{\text{LDAC}}$ low after $\overline{\text{SYNC}}$ has been taken high. The update now occurs on the falling edge of $\overline{\text{LDAC}}$.

Reset Function ($\overline{\text{RESET}}$)

The AD5791 can be reset to its power-on state by two means: either by asserting the $\overline{\text{RESET}}$ pin or by utilizing the software RESET control function (see Table 14). If the $\overline{\text{RESET}}$ pin is not used, it should be hardwired to IOV_{CC}.

Asynchronous Clear Function (CLR)

The $\overline{\text{CLR}}$ pin is an active low clear that allows the output to be cleared to a user defined value. The 20-bit clear code value is programmed to the clearcode register (see Table 13). It is necessary to maintain $\overline{\text{CLR}}$ low for a minimum amount of time to complete the operation (see Figure 2).When the $\overline{\text{CLR}}$ signal is returned high the output remains at the clear value (if $\overline{\text{LDAC}}$ is high) until a new value is loaded to the DAC register. The output cannot be updated with a new value while the $\overline{\text{CLR}}$ pin is low. A clear operation can also be performed by setting the CLR bit in the software control register (see Table 14).

Table 9. Hardware Control Pins Truth Table

$\overline{\text{LDAC}}$	$\overline{\text{CLR}}$	$\overline{\text{RESET}}$	Function
X[1]	X[1]	0	The AD5791 is in reset mode. The device cannot be programmed.
X[1]	X[1]	↑	The AD5791 is returned to its power-on state. All registers are set to their default values.
0	0	1	The DAC register is loaded with the clearcode register value and the output is set accordingly.
0	1	1	The output is set according to the DAC register value.
1	0	1	The DAC register is loaded with the clearcode register value and the output is set accordingly.
↓	1	1	The output is set according to the DAC register value.
↓	0	1	The output remains at the clear code value.
↑	1	1	The output remains set according to the DAC register value.
↑	0	1	The output remains at the clear code value.
1	↓	1	The DAC register is loaded with the clearcode register value and the output is set accordingly.
0	↓	1	The DAC register is loaded with the clearcode register value and the output is set accordingly.
1	↑	1	The output remains at the clear code value
0	↑	1	The output is set according to the DAC register value.

[1] X is don't care.

ON-CHIP REGISTERS

DAC Register

Table 10 outlines how data is written to and read from the DAC register.

Table 10. DAC Register

MSB **LSB**

DB23	DB22	DB21	DB20	DB19	DB0
R/$\overline{\text{W}}$	Register address			DAC register data	
R/$\overline{\text{W}}$	0	0	1	20-bits of data	

The following equation describes the ideal transfer function of the DAC:

$$V_{OUT} = \frac{(V_{REFP} - V_{REFN}) \times D}{2^{20} - 1} + V_{REFN}$$

where:
V_{REFN} is the negative voltage applied at the V_{REFN} input pins.
V_{REFP} is the positive voltage applied at the V_{REFP} input pins.
D is the 20-bit code programmed to the DAC.

Control Register

The control register controls the mode of operation of the AD5791.

Table 11. Control Register

MSB **LSB**

DB23	DB22	DB21	DB20	DB19…DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
R/$\overline{W}$	Register address			Control register data											
R/$\overline{W}$	0	1	0	Reserved	Reserved	LIN COMP				SDODIS	BIN/2sC	DACTRI	OPGND	RBUF	Reserved

Table 12. Control Register Functions

Function	Description				
Reserved	These bits are reserved and should be programmed to zero.				
RBUF	Output amplifier configuration control.				
	0: internal amplifier, A1, is powered up and Resistor R_{FB} and R1 are connected in series as shown in Figure 53. This allows an external amplifier to be connected in a gain of two configurations. See the AD5791 Features section for further details.				
	1: (default) internal amplifier, A1, is powered down and Resistor R_{FB} and R1 are connected in parallel as shown in Figure 52 so that the resistance between the R_{FB} and INV pins is 3.4 kΩ, equal to the resistance of the DAC. This allows the R_{FB} and INV pins to be used for input bias current compensation for an external unity gain amplifier. See the AD5791 Features section for further details.				
OPGND	Output ground clamp control. 0: DAC output clamp to ground is removed and the DAC is placed in normal mode. 1: (default) DAC output is clamped to ground through a ~6 kΩ resistance, and the DAC is placed in tristate mode. Resetting the part puts the DAC in OPGND mode, where the output ground clamp is enabled and the DAC is tristated. Setting the OPGND bit to 1 in the control register overrules any write to the DACTRI bit.				
DACTRI	DAC tristate control. 0: DAC is in normal operating mode. 1: (default) DAC is in tristate mode.				
BIN/2sC	DAC register coding select. 0: (default) DAC register uses twos complement coding. 1: DAC register uses offset binary coding.				
SDODIS	SDO pin enable/disable control. 0: (default) SDO pin is enabled. 1: SDO pin is disabled (tristate).				
LIN COMP	Linearity error compensation for varying reference input spans. See the AD5791 Features section for further details.				
	0	0	0	0	(Default) reference input span up to 10 V.
	1	0	0	1	Reference input span between 10 V and 12 V.
	1	0	1	0	Reference input span between 12 V and 16 V.
	1	0	1	1	Reference input span between16 V and 19 V.
	1	1	0	0	Reference input span between 19 V and 20 V.
R/$\overline{W}$	Read/write select bit. 0: AD5791 is addressed for a write operation. 1: AD5791 is addressed for a read operation.				

Clearcode Register

The clearcode register sets the value to which the DAC output is set when the $\overline{CLR}$ pin or CLR bit is asserted. The output value depends on the DAC coding that is being used, either binary or twos complement. The default register value is 0.

Table 13. Clearcode Register

MSB **LSB**

DB23	DB22	DB21	DB20	DB19	DB0
R/$\overline{W}$	Register address			Clearcode register data	
R/$\overline{W}$	0	1	1	20-bits of data	

Software Control Register

This is a write only register in which writing a 1 to a particular bit has the same effect as pulsing the corresponding pin low.

Table 14. Software Control Register

MSB **LSB**

DB23	DB22	DB21	DB20	DB19	DB3	DB2	DB1	DB0
R/$\overline{W}$	Register address			Software control register data				
0	1	0	0	Reserved		RESET	CLR[1]	LDAC[2]

[1] The CLR function has no effect if the $\overline{LDAC}$ pin is low.
[2] The LDAC function has no effect if the $\overline{CLR}$ pin is low.

Table 15. Software Control Register Functions

Function	Description
LDAC	Setting this bit to a 1 updates the DAC register and consequently the DAC output.
CLR	Setting this bit to a 1 sets the DAC register to a user defined value (see Table 13) and updates the DAC output. The output value depends on the DAC register coding that is being used, either binary or twos complement.
RESET	Setting this bit to a 1 returns the AD5791 to its power-on state.

AD5791 FEATURES

POWER-ON TO 0 V

The AD5791 contains a power-on reset circuit that, as well as resetting all registers to their default values, controls the output voltage during power-up. Upon power-on the DAC is placed in tristate (its reference inputs are disconnected) and its output is clamped to ground through a ~6 kΩ resistor. The DAC remains in this state until programmed otherwise via the control register. This is a useful feature in applications where it is important to know the state of the DAC output while it is in the process of powering up.

CONFIGURING THE AD5791

After power-on the AD5791 must be configured to put it into normal operating mode before programming the output. To do this, the control register must be programmed. The DAC is removed from tristate by clearing the DACTRI bit, and the output clamp is removed by clearing the OPGND bit. At this point, the output goes to V_{REFN}, unless an alternative value is first programmed to the DAC register.

DAC OUTPUT STATE

The DAC output can be placed in one of three states, controlled by the DACTRI and OPGND bits of the control register, as shown in Table 16.

Table 16. AD5791 Output State Truth Table

DACTRI	OPGND	Output State
0	0	Normal operating mode
0	1	Output is clamped via ~6 kΩ to AGND
1	0	Output is in tristate
1	1	Output is clamped via ~6 kΩ to AGND

LINEARITY COMPENSATION

The integral nonlinearity (INL) of the AD5791 can vary according to the applied reference voltage span, the LIN COMP bits of the control register can be programmed to compensate for this variation in INL. The specifications in this data sheet are obtained with LIN COMP = 0000 for reference spans up to and including 10 V and with LIN COMP = 1100 for a reference span of 20 V. The default value of the LIN COMP bits is 0000. Intermediate LIN COMP values can be programmed for reference spans between 10 V and 20 V as shown in Table 12.

OUTPUT AMPLIFIER CONFIGURATION

There are a number of different ways that an output amplifier can be connected to the AD5791, depending on the voltage references applied and the desired output voltage span.

Unity Gain Configuration

Figure 50 shows an output amplifier configured for unity gain, in this configuration the output spans from V_{REFN} to V_{REFP}.

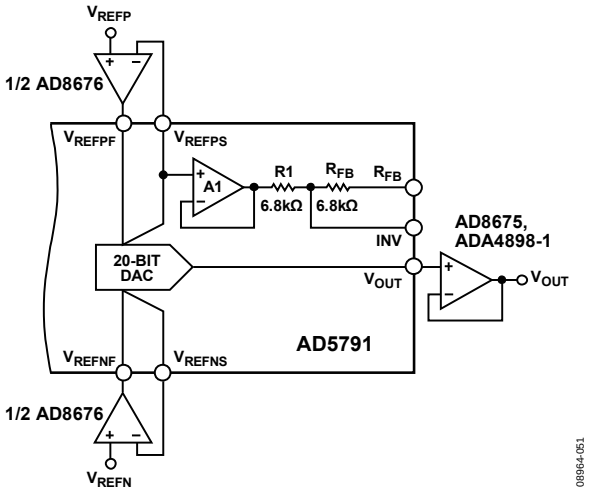


Figure 50. Output Amplifier in Unity Gain Configuration

A second unity gain configuration for the output amplifier is one that removes an offset from the input bias currents of the amplifier. It does this by inserting a resistance in the feedback path of the amplifier that is equal to the output resistance of the DAC. The DAC output resistance is 3.4 kΩ, by connecting R1 and R_{FB} in parallel, a resistance equal to the DAC resistance is available on chip. Because the resistors are all on one piece of silicon, they are temperature coefficient matched. To enable this mode of operation the RBUF bit of the control register must be set to Logic 1. Figure 51 shows how the output amplifier is connected to the AD5791. In this configuration, the output amplifier is in unity gain and the output spans from V_{REFN} to V_{REFP}. This unity gain configuration allows a capacitor to be placed in the amplifier feedback path to improve dynamic performance.

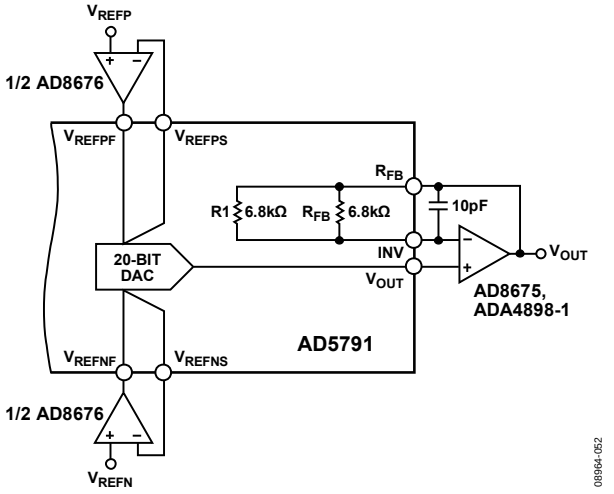


Figure 51. Output Amplifier in Unity Gain with Amplifier Input Bias Current Compensation

Gain of Two Configuration

Figure 52 shows an output amplifier configured for a gain of two. The gain is set by the internal matched 6.8 kΩ resistors, which are exactly twice the DAC resistance, having the effect of removing an offset from the input bias current of the external amplifier. In this configuration, the output spans from 2 × V_{REFN} − V_{REFP} to V_{REFP}. This configuration is used to generate a bipolar output span from a single ended reference input with V_{REFN} = 0 V. For this mode of operation, the RBUF bit of the control register must be cleared to Logic 0.

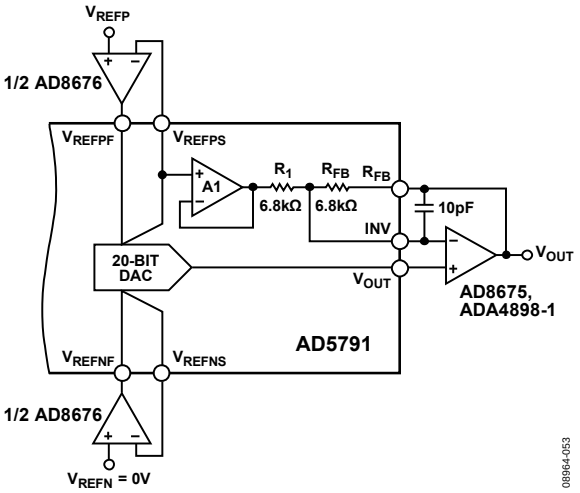


Figure 52. Output Amplifier in Gain of Two Configuration

APPLICATIONS INFORMATION

TYPICAL OPERATING CIRCUIT

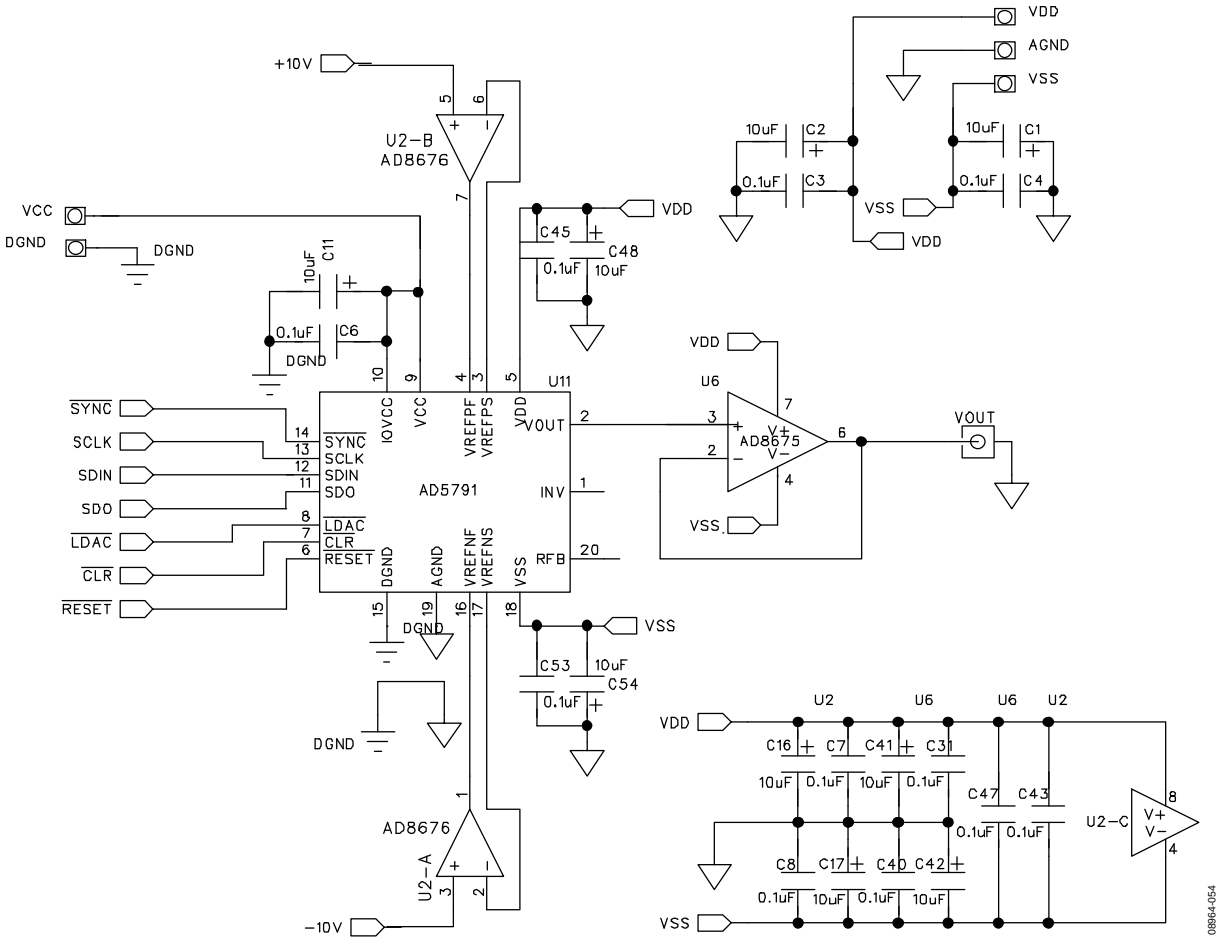


Figure 53. Typical Operating Circuit

Figure 53 shows a typical operating circuit for the AD5791 using an AD8676 for reference buffers and an AD8675 as an output buffer. To meet the specified linearity, force sense buffers must be used on the reference inputs. Because the output impedance of the AD5791 is 3.4 kΩ, an output buffer is required for driving low resistive, high capacitance loads.

OUTLINE DIMENSIONS

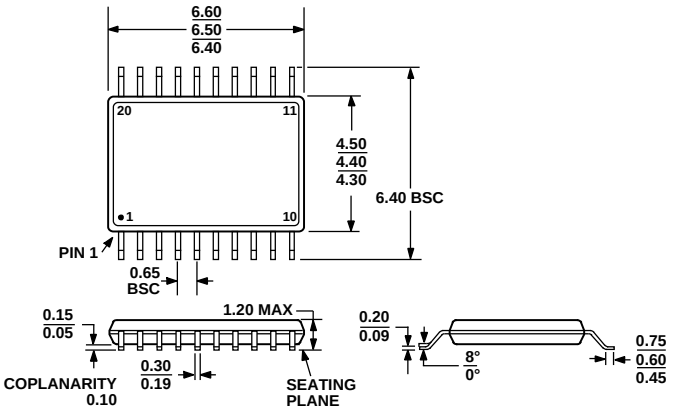


Figure 54. 20-Lead Thin Shrink Small Outline Package [TSSOP]
(RU-20)
Dimensions shown in millimeters

ORDERING GUIDE

Model[1]	Temperature Range	INL	Package Description	Package Option
AD5791BRUZ	−40°C to +125°C	±1.5 LSB	20-Lead TSSOP	RU-20
AD5791ARUZ	−40°C to +125°C	±4 LSB	20-Lead TSSOP	RU-20
EVAL-AD5791SDZ			Evaluation Board	

[1] Z = RoHS Compliant Part.

NOTES

©2010–2013 Analog Devices, Inc. All rights reserved. Trademarks and registered trademarks are the property of their respective owners.
D08964-0-7/13(D)

17.3 Convertisseurs logiques (ZCD) LTC6957-2

LTC6957-1/LTC6957-2/ LTC6957-3/LTC6957-4

Low Phase Noise, Dual Output Buffer/Driver/ Logic Converter

FEATURES

- **Low Phase Noise Buffer/Driver**
- **Optimized Conversion of Sine Wave Signals to Logic Levels**
- **Three Logic Output Types Available**
  - **LVPECL**
  - **LVDS**
  - **CMOS**
- Additive Jitter 45fs$_{RMS}$ (LTC6957-1)
- Frequency Range Up to 300MHz
- 3.15V to 3.45V Supply Operation
- Low Skew 3ps Typical
- Fully Specified from –40°C to 125°C
- 12-Lead MSOP and 3mm × 3mm DFN Packages

APPLICATIONS

- System Reference Frequency Distribution
- High Speed ADC, DAC, DDS Clock Driver
- Military and Secure Radio
- Low Noise Timing Trigger
- Broadband Wireless Transceiver
- High Speed Data Acquisition
- Medical Imaging
- Test and Measurement

DESCRIPTION

The LTC®6957-1/LTC6957-2/LTC6957-3/LTC6957-4 is a family of very low phase noise, dual output AC signal buffer/driver/logic level translators. The input signal can be a sine wave or any logic level (≤2V$_{P-P}$). There are four members of the family that differ in their output logic signal type as follows:

LTC6957-1: LVPECL Logic Outputs

LTC6957-2: LVDS Logic Outputs

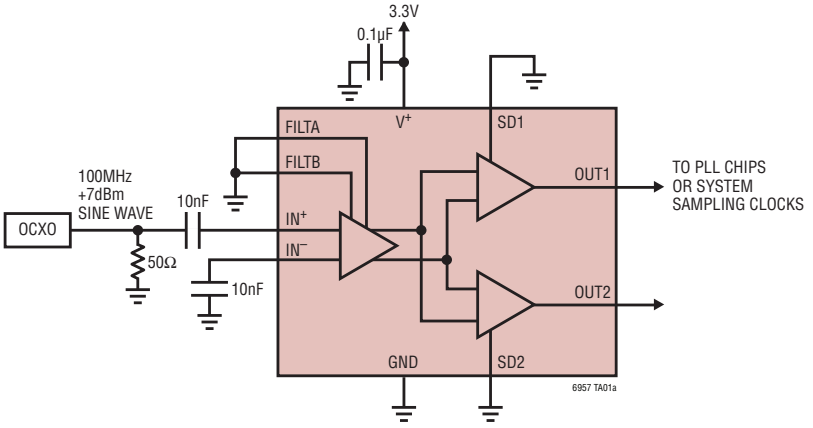
LTC6957-3: CMOS Logic, In-Phase Outputs

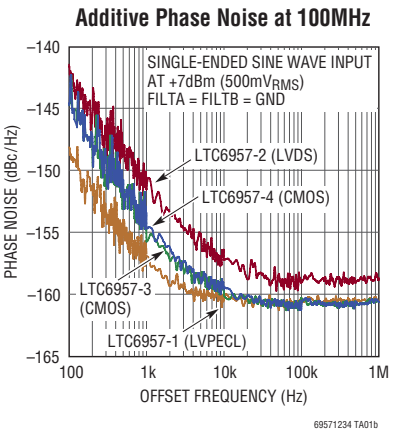
LTC6957-4: CMOS Logic, Complementary Outputs

The LTC6957 will buffer and distribute any logic signal with minimal additive noise, however, the part really excels at translating sine wave signals to logic levels. The early amplifier stages have selectable lowpass filtering to minimize the noise while still amplifying the signal to increase its slew rate. This input stage filtering/noise limiting is especially helpful in delivering the lowest possible phase noise signal with slow slewing input signals such as a typical 10MHz sine wave system reference.

LT, LTC, LTM, Linear Technology and the Linear logo are registered trademarks of Linear Technology Corporation. All other trademarks are the property of their respective owners. Protected by U.S. Patents 7969189 and 8319551.

TYPICAL APPLICATION





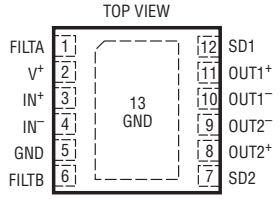
ABSOLUTE MAXIMUM RATINGS (Note 1)

Supply Voltage (V^+ or V_{DD}) to GND 3.6V
Input Current (IN^+, IN^-, FILTA, FILTB, SD1, SD2)
(Note 2) ..... ±10mA
LTC6957-1 Output Current ..... 1mA, –30mA
LTC6957-2 Output Current ..... ±10mA
LTC6957-3, LTC6957-4 Output Current (Note 3) .. ±30mA

Specified Temperature Range
LTC6957I ..... –40°C to 85°C
LTC6957H ..... –40°C to 125°C
Junction Temperature ..... 150°C
Storage Temperature Range ..... –65°C to 150°C
Lead Temperature (for MSOP Soldering, 10sec) ... 300°C

PIN CONFIGURATION

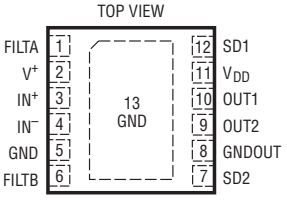
LTC6957-1, LTC6957-2



DD PACKAGE
12-LEAD (3mm × 3mm) PLASTIC DFN

T_{JMAX} = 150°C, θ_{JA} = 58°C/W, θ_{JC} = 10°C/W
EXPOSED PAD (PIN 13) IS GND, MUST BE SOLDERED TO PCB

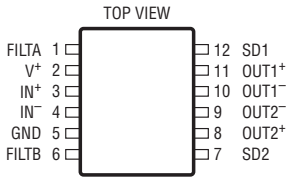
LTC6957-3, LTC6957-4



DD PACKAGE
12-LEAD (3mm × 3mm) PLASTIC DFN

T_{JMAX} = 150°C, θ_{JA} = 58°C/W, θ_{JC} = 10°C/W
EXPOSED PAD (PIN 13) IS GND, MUST BE SOLDERED TO PCB

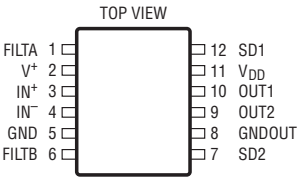
LTC6957-1, LTC6957-2



MS PACKAGE
12-LEAD PLASTIC MSOP

T_{JMAX} = 150°C, θ_{JA} = 145°C/W

LTC6957-3, LTC6957-4



MS PACKAGE
12-LEAD PLASTIC MSOP

T_{JMAX} = 150°C, θ_{JA} = 145°C/W

ORDER INFORMATION

LEAD FREE FINISH	TAPE AND REEL	PART MARKING*	PACKAGE DESCRIPTION	SPECIFIED TEMPERATURE RANGE
LTC6957IDD-1#PBF	LTC6957IDD-1#TRPBF	LFQJ	12-Lead (3mm × 3mm) Plastic DFN	–40°C to 85°C
LTC6957IDD-2#PBF	LTC6957IDD-2#TRPBF	LFQK	12-Lead (3mm × 3mm) Plastic DFN	–40°C to 85°C
LTC6957IDD-3#PBF	LTC6957IDD-3#TRPBF	LFQM	12-Lead (3mm × 3mm) Plastic DFN	–40°C to 85°C
LTC6957IDD-4#PBF	LTC6957IDD-4#TRPBF	LFQN	12-Lead (3mm × 3mm) Plastic DFN	–40°C to 85°C
LTC6957IMS-1#PBF	LTC6957IMS-1#TRPBF	69571	12-Lead Plastic MSOP	–40°C to 85°C
LTC6957HMS-1#PBF	LTC6957HMS-1#TRPBF	69571	12-Lead Plastic MSOP	–40°C to 125°C
LTC6957IMS-2#PBF	LTC6957IMS-2#TRPBF	69572	12-Lead Plastic MSOP	–40°C to 85°C
LTC6957HMS-2#PBF	LTC6957HMS-2#TRPBF	69572	12-Lead Plastic MSOP	–40°C to 125°C
LTC6957IMS-3#PBF	LTC6957IMS-3#TRPBF	69573	12-Lead Plastic MSOP	–40°C to 85°C
LTC6957HMS-3#PBF	LTC6957HMS-3#TRPBF	69573	12-Lead Plastic MSOP	–40°C to 125°C
LTC6957IMS-4#PBF	LTC6957IMS-4#TRPBF	69574	12-Lead Plastic MSOP	–40°C to 85°C
LTC6957HMS-4#PBF	LTC6957HMS-4#TRPBF	69574	12-Lead Plastic MSOP	–40°C to 125°C

Consult LTC Marketing for parts specified with wider operating temperature ranges. *The temperature grade is identified by a label on the shipping container.

For more information on lead free part marking, go to: http://www.linear.com/leadfree/
For more information on tape and reel specifications, go to: http://www.linear.com/tapeandreel/

ELECTRICAL CHARACTERISTICS LTC6957-1

The ● denotes the specifications which apply over the full operating temperature range, otherwise specifications are at T_A = 25°C. V^+ = 3.3V, SD1 = SD2 = 0.4V, FILTA = FILTB = 0.4V, R_{LOAD} = 50Ω connected to 1.3V, unless otherwise specified. All voltages are with respect to ground.

SYMBOL	PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
Inputs (IN^-, IN^+)							
f_{IN}	Input Frequency Range		●			300	MHz
V_{INSE}	Input Signal Level Range, Single-Ended		●	0.2	0.8	2	V_{P-P}
V_{INDIFF}	Input Signal Level Range, Differential		●	0.2	0.8	2	V_{P-P}
t_{MIN}	Minimum Input Pulse Width	High or Low			0.5		ns
V_{INCM}	Self-Bias Voltage, IN^+, IN^-		●	1.8	2.06	2.3	V
R_{IN}	Input Resistance, Differential		●	1.5	2	2.5	kΩ
C_{IN}	Input Capacitance, Differential				0.5		pF
BW_{IN}	Input Section Small Signal Bandwidth (–3dB)	FILTB = L, FILTA = L FILTB = L, FILTA = H FILTB = H, FILTA = L FILTB = H, FILTA = H			1200 500 160 50		MHz MHz MHz MHz
Outputs (LVPECL)							
V_{OH}	Output High Voltage	LTC6957I LTC6957H	● ●	V^+ – 1.22 V^+ – 1.22	V^+ – 0.98 V^+ – 0.98	V^+ – 0.93 V^+ – 0.87	V V
V_{OL}	Output Low Voltage	LTC6957I LTC6957H	● ●	V^+ – 2.1 V^+ – 2.1	V^+ – 1.8 V^+ – 1.8	V^+ – 1.67 V^+ – 1.62	V V
V_{OD}	Output Differential Voltage		●	±660	±810	±965	mV
t_r	Output Rise Time				180		ps
t_f	Output Fall Time				160		ps
t_{PD}	Propagation Delay	FILTB = L, FILTA = L FILTB = L, FILTA = H FILTB = H, FILTA = L FILTB = H, FILTA = H	● ● ● ●	0.35	0.5 0.6 1.1 3.2	0.7 0.8 1.3 4	ns ns ns ns
$\Delta t_{PD}/\Delta T$	Propagation Delay Variation Over Temperature	FILTB = L, FILTA = L FILTB = L, FILTA = H FILTB = H, FILTA = L FILTB = H, FILTA = H	● ● ● ●		0.1 0.1 0.11 0.15		ps/°C ps/°C ps/°C ps/°C
$\Delta t_{PD}/\Delta V$	Propagation Delay Variation vs Supply Voltage	FILTB = L, FILTA = L	●		4	50	ps/V
t_{SKEW}	Output Skew, Differential, CH1 to CH2		●		3	30	ps
t_{MATCH}	Output Matching ($OUTx^+$ to $OUTx^-$)	See Timing Diagram	●		2.5	30	ps
Power							
V^+	V^+ Operating Supply Voltage Range	R_{LOAD} = 50Ω to (V^+– 2V)	●	3.15	3.3	3.45	V
I_S	Supply Current Both Outputs Enabled (SD1 = SD2 = L) One Output Enabled (SD1 = L, SD2 = H or SD1 = H, SD2 = L) Both Outputs Disabled (SD1 = SD2 = H) Including Output Loads	 No Output Loads No Output Loads No Output Loads R_{LOAD} = 50Ω to (V^+– 2V), ×4	 ● ● ● ●		 18 15 0.7 58	 22 19 1.2 72	 mA mA mA mA
t_{ENABLE}	Output Enable Time, Other SDx = L				40		µs
t_{WAKEUP}	Output Enable Time, Other SDx = H				120		µs
$t_{DISABLE}$	Output Disable Time, Other SDx = L				20		µs
t_{SLEEP}	Output Disable Time, Other SDx = H				20		µs

ELECTRICAL CHARACTERISTICS LTC6957-1

The ● denotes the specifications which apply over the full operating temperature range, otherwise specifications are at T_A = 25°C. V^+ = 3.3V, SD1 = SD2 = 0.4V, FILTA = FILTB = 0.4V, R_{LOAD} = 50Ω connected to 1.3V, unless otherwise specified. All voltages are with respect to ground.

SYMBOL	PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
Digital Logic Inputs							
V_{IH}	High Level SD or FILT Input Voltage		●	V^+ – 0.4			V
V_{IL}	Low Level SD or FILT Input Voltage		●			0.4	V
I_{IN_DIG}	Input Current SD or FILT Pins		●		0.1	±10	µA
Additive Phase Noise and Jitter							
	f_{IN} = 300MHz Sine Wave, 7dBm (FILTA = L, FILTB = L)						
	at 10Hz Offset				–130		dBc/Hz
	at 100Hz Offset				–140		dBc/Hz
	at 1kHz Offset				–150		dBc/Hz
	at 10kHz Offset				–157		dBc/Hz
	at 100kHz Offset				–157.5		dBc/Hz
	>1MHz Offset				–157.5		dBc/Hz
	Jitter (10Hz to 150MHz)				123		fs_{RMS}
	Jitter (12kHz to 20MHz)				45		fs_{RMS}
	f_{IN} = 122.88MHz Sine Wave, 0dBm (FILTA = H, FILTB = L)						
	at 10Hz Offset				–137		dBc/Hz
	at 100Hz Offset				–146		dBc/Hz
	at 1kHz Offset				–154.6		dBc/Hz
	at 10kHz Offset				–157		dBc/Hz
	at 100kHz Offset				–157.2		dBc/Hz
	>1MHz Offset				–157.2		dBc/Hz
	Jitter (10Hz to 61.44MHz)				200		fs_{RMS}
	Jitter (12kHz to 20MHz)				114		fs_{RMS}
	f_{IN} = 100MHz Sine Wave, 10dBm (FILTA = L, FILTB = L)						
	at 10Hz Offset				–138		dBc/Hz
	at 100Hz Offset				–148.1		dBc/Hz
	at 1kHz Offset				–156.8		dBc/Hz
	at 10kHz Offset				–160.6		dBc/Hz
	at 100kHz Offset				–161		dBc/Hz
	>1MHz Offset				–161		dBc/Hz
	Jitter (10Hz to 50MHz)				142		fs_{RMS}
	Jitter (12kHz to 20MHz)				90		fs_{RMS}

ELECTRICAL CHARACTERISTICS LTC6957-2

The ● denotes the specifications which apply over the full operating temperature range, otherwise specifications are at T_A = 25°C. V^+ = 3.3V, SD1 = SD2 = 0.4V, FILTA = FILTB = 0.4V, R_{LOAD} = 110Ω differential, unless otherwise specified. All voltages are with respect to ground.

SYMBOL	PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
Inputs (IN^-, IN^+)							
f_{IN}	Input Frequency Range		●			300	MHz
V_{INSE}	Input Signal Level Range, Single-Ended		●	0.2	0.8	2	V_{P-P}
V_{INDIFF}	Input Signal Level Range, Differential		●	0.2	0.8	2	V_{P-P}
t_{MIN}	Minimum Input Pulse Width	High or Low			0.5		ns
V_{INCM}	Self-Bias Voltage, IN^+, IN^-		●	1.8	2	2.3	V
R_{IN}	Input Resistance, Differential		●	1.5	2	2.5	kΩ
C_{IN}	Input Capacitance, Differential				0.5		pF
BW_{IN}	Input Section Small Signal Bandwidth	FILTB = L, FILTA = L FILTB = L, FILTA = H FILTB = H, FILTA = L FILTB = H, FILTA = H			1200 500 160 50		MHz MHz MHz MHz
Outputs (LVDS)							
V_{OD}	Output Differential Voltage		●	250	360	450	mV
ΔV_{OD}	Delta V_{OD}		●		0.2	50	mV
V_{OS}	Output Offset Voltage		●	1.125	1.25	1.375	V
ΔV_{OS}	Delta V_{OS}		●		1.5	50	mV
I_{SC}	Short-Circuit Current		●		3.9	6	mA
t_r	Output Rise Time				170		ps
t_f	Output Fall Time				170		ps
t_{PD}	Propagation Delay	FILTB = L, FILTA = L FILTB = L, FILTA = H FILTB = H, FILTA = L FILTB = H, FILTA = H	● ● ● ●	0.65	0.84 0.9 1.35 3.5	1.15 1.3 1.8 4.4	ns ns ns ns
$\Delta t_{PD}/\Delta T$	Propagation Delay Variation Over Temperature	FILTB = L, FILTA = L FILTB = L, FILTA = H FILTB = H, FILTA = L FILTB = H, FILTA = H	● ● ● ●		0.5 0.6 0.7 1.8		ps/°C ps/°C ps/°C ps/°C
$\Delta t_{PD}/\Delta V$	Propagation Delay Variation vs Supply Voltage	FILTB = L, FILTA = L	●		5	60	ps/V
t_{SKEW}	Output Skew, Differential, CH1 to CH2		●		3	50	ps
Power							
V^+	V^+ Operating Supply Voltage Range		●	3.15	3.3	3.45	V
I_S	Supply Current Both Outputs Enabled (SD1 = SD2 = L) One Output Enabled (SD1 = L, SD2 = H or SD1 = H, SD2 = L) Both Outputs Disabled (SD1 = SD2 = H)		 ● ● ●		 38 26 0.7	 45 30 1.2	 mA mA mA
t_{ENABLE}	Output Enable Time, Other SDx = L				300		ns
t_{WAKEUP}	Output Enable Time, Other SDx = H				400		ns
$t_{DISABLE}$	Output Disable Time, Other SDx = L				40		ns
t_{SLEEP}	Output Disable Time, Other SDx = H				50		ns

ELECTRICAL CHARACTERISTICS LTC6957-2

The ● denotes the specifications which apply over the full operating temperature range, otherwise specifications are at T_A = 25°C. V^+ = 3.3V, SD1 = SD2 = 0.4V, FILTA = FILTB = 0.4V, R_{LOAD} = 110Ω differential, unless otherwise specified. All voltages are with respect to ground.

SYMBOL	PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
Digital Logic Inputs							
V_{IH}	High Level SD or FILT Input Voltage		●	V^+ – 0.4			V
V_{IL}	Low Level SD or FILT Input Voltage		●			0.4	V
I_{IN_DIG}	Input Current SD or FILT Pins		●		0.1	±10	µA
Additive Phase Noise and Jitter							
	f_{IN} = 300MHz Sine Wave, 7dBm (FILTA = L, FILTB = L)						
	10Hz Offset				–124		dBc/Hz
	100Hz Offset				–134		dBc/Hz
	1kHz Offset				–143.5		dBc/Hz
	10kHz Offset				–151.3		dBc/Hz
	100kHz Offset				–154		dBc/Hz
	>1MHz Offset				–154		dBc/Hz
	Jitter (10Hz to 150MHz)				183		fs_{RMS}
	Jitter (12kHz to 20MHz)				67		fs_{RMS}
	f_{IN} = 122.88MHz Sine Wave, 0dBm (FILTA = H, FILTB = L)						
	10Hz Offset				–132.5		dBc/Hz
	100Hz Offset				–142.5		dBc/Hz
	1kHz Offset				–150.7		dBc/Hz
	10kHz Offset				–156		dBc/Hz
	100kHz Offset				–157		dBc/Hz
	>1MHz Offset				–157		dBc/Hz
	Jitter (10Hz to 61.44MHz)				203		fs_{RMS}
	Jitter (12kHz to 20MHz)				116		fs_{RMS}
	f_{IN} = 100MHz Sine Wave, 10dBm (FILTA = L, FILTB = L)						
	10Hz Offset				–132		dBc/Hz
	100Hz Offset				–142		dBc/Hz
	1kHz Offset				–151		dBc/Hz
	10kHz Offset				–157.5		dBc/Hz
	100kHz Offset				–159.5		dBc/Hz
	>1MHz Offset				–159.5		dBc/Hz
	Jitter (10Hz to 50MHz)				169		fs_{RMS}
	Jitter (12kHz to 20MHz)				107		fs_{RMS}

ELECTRICAL CHARACTERISTICS LTC6957-3/LTC6957-4

The ● denotes the specifications which apply over the full operating temperature range, otherwise specifications are at T_A = 25°C. $V^+ = V_{DD}$ = 3.3V, SD1 = SD2 = 0.4V, FILTA = FILTB = 0.4V, R_{LOAD} = 480Ω to $V_{DD}/2$, unless otherwise specified. All voltages are with respect to ground.

SYMBOL	PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
Inputs (IN^-, IN^+)							
f_{IN}	Input Frequency Range		●			300	MHz
V_{INSE}	Input Signal Level Range, Single-Ended		●	0.2	0.8	2	V_{P-P}
V_{INDIFF}	Input Signal Level Range, Differential		●	0.2	0.8	2	V_{P-P}
t_{MIN}	Minimum Input Pulse Width	High or Low			0.6		ns
V_{INCM}	Self-Bias Voltage, IN^+, IN^-		●	1.8	2	2.3	V
R_{IN}	Input Resistance, Differential		●	1.5	2	2.5	kΩ
C_{IN}	Input Capacitance, Differential				0.5		pF
BW_{IN}	Input Section Small Signal Bandwidth	FILTB = L, FILTA = L FILTB = L, FILTA = H FILTB = H, FILTA = L FILTB = H, FILTA = H			1200 500 160 50		MHz MHz MHz MHz
Outputs (CMOS)							
V_{OH}	Output High Voltage	No Load –3mA Load	● ●	V_{DD} – 0.1 V_{DD} – 0.2			V V
V_{OL}	Output Low Voltage	No Load 3mA Load	● ●			0.1 0.2	V V
t_r	Output Rise Time				320		ps
t_f	Output Fall Time				300		ps
t_{PD}	Propagation Delay	FILTB = L, FILTA = L FILTB = L, FILTA = H FILTB = H, FILTA = L FILTB = H, FILTA = H	● ● ● ●	0.8	0.95 1 1.5 3.6	1.6 1.8 2.4 4.8	ns ns ns ns
$\Delta t_{PD}/\Delta T$	Propagation Delay Variation Over Temperature	FILTB = L, FILTA = L FILTB = L, FILTA = H FILTB = H, FILTA = L FILTB = H, FILTA = H	● ● ● ●		1.7 1.7 2 3		ps/°C ps/°C ps/°C ps/°C
$\Delta t_{PD}/\Delta V$	Propagation Delay Variation vs Supply Voltage	FILTB = FILTA = L, $V^+ = V_{DD}$	●		100	200	ps/V
t_{SKEW}	Output Skew, CH1 to CH2 LTC6957-3 LTC6957-4		 ● ●		 5 120	 35 250	 ps ps
Power							
V^+	V^+ Operating Supply Voltage Range		●	3.15	3.3	3.45	V
V_{DD}	V_{DD} Operating Supply Voltage Range	V_{DD} Must Be ≤V^+	●	2.4	3.3	3.45	V
I_S	Supply Current, Pin 2 Both Outputs Enabled (SD1 = SD2 = L) One Output Enabled (SD1 = L, SD2 = H or SD1 = H, SD2 = L) Both Outputs Disabled (SD1 = SD2 = H)		 ● ● ●		 24 24 0.7	 27.5 27.5 1.2	 mA mA mA
I_{DD}	Supply Current, Pin 11, No Load	Static Dynamic, per Output	● ●		0.001 0.056	0.01 0.07	mA mA/MHz
t_{ENABLE}	Output Enable Time, Other SDx = L				200		ns
t_{WAKEUP}	Output Enable Time, Other SDx = H				300		ns
$t_{DISABLE}$	Output Disable Time, Other SDx = L				20		ns
t_{SLEEP}	Output Disable Time, Other SDx = H				20		ns

ELECTRICAL CHARACTERISTICS LTC6957-3/LTC6957-4

The ● denotes the specifications which apply over the full operating temperature range, otherwise specifications are at T_A = 25°C. $V^+ = V_{DD}$ = 3.3V, SD1 = SD2 = 0.4V, FILTA = FILTB = 0.4V, R_{LOAD} = 480Ω to $V_{DD}/2$, unless otherwise specified. All voltages are with respect to ground.

SYMBOL	PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
Digital Logic Inputs							
V_{IH}	High Level SD or Filt Input Voltage		●	V^+ – 0.4			V
V_{IL}	Low Level SD or Filt Input Voltage		●			0.4	V
I_{IN_DIG}	Input Current SD or Filt Pins		●		0.1	±10	µA
Additive Phase Noise and Jitter							
	f_{IN} = 300MHz Sine Wave, 7dBm (FILTA = L, FILTB = L)						
	10Hz Offset				–123		dBc/Hz
	100Hz Offset				–133		dBc/Hz
	1kHz Offset				–143		dBc/Hz
	10kHz Offset				–152		dBc/Hz
	100kHz Offset				–156		dBc/Hz
	>1MHz Offset				–156		dBc/Hz
	Jitter (10Hz to 150MHz)				146		fs_{RMS}
	Jitter (12kHz to 20MHz)				53		fs_{RMS}
	f_{IN} = 122.88MHz Sine Wave, 0dBm (FILTA = H, FILTB = L)						
	10Hz Offset				–132		dBc/Hz
	100Hz Offset				–142		dBc/Hz
	1kHz Offset				–150.6		dBc/Hz
	10kHz Offset				–156.5		dBc/Hz
	100kHz Offset				–157.4		dBc/Hz
	>1MHz Offset				–157.4		dBc/Hz
	Jitter (10Hz to 61.44MHz)				192		fs_{RMS}
	Jitter (12kHz to 20MHz)				109		fs_{RMS}
	f_{IN} = 100MHz Sine Wave, 10dBm (FILTA = L, FILTB = L)						
	10Hz Offset				–135		dBc/Hz
	100Hz Offset				–145		dBc/Hz
	1kHz Offset				–153		dBc/Hz
	10kHz Offset				–159.8		dBc/Hz
	100kHz Offset				–161		dBc/Hz
	>1MHz Offset				–161		dBc/Hz
	Jitter (10Hz to 50MHz)				142		fs_{RMS}
	Jitter (12kHz to 20MHz)				90		fs_{RMS}

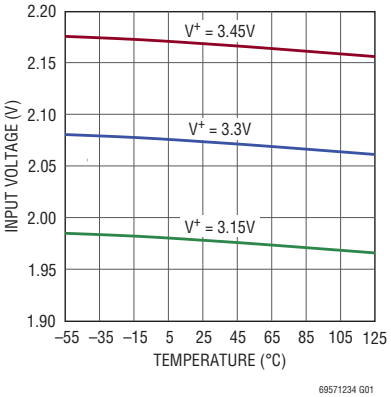
Note 1: Stresses beyond those listed under Absolute Maximum Ratings may cause permanent damage to the device. Exposure to any Absolute Maximum Rating condition for extended periods may affect device reliability and lifetime.

Note 2: Input pins IN^+, $IN^–$, FILTA, FILTB, SD1 and SD2 are protected by steering diodes to either supply. If the inputs go beyond either supply rail, the input current should be limited to less than 10mA. If pushing current into FILTB, the Pin 6 voltage must be limited to 4V. On the logic pins (FILTA, FILTB, SD1 and SD2) the Absolute Maximum input current applies only at the maximum operating supply voltage of 3.45V; 10mA of input current with the absolute maximum supply voltage of 3.6V may create permanent damage from voltage stress.

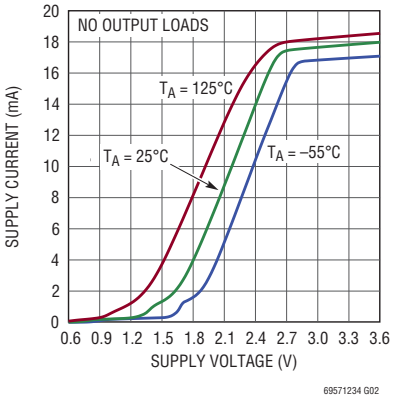
Note 3: With 3.6V Absolute Maximum supply voltage, the LTC6957-3/LTC6957-4 CMOS outputs can sink 30mA while low, and source 30mA while high without damage. However, if overdriven or subject to an inductive load kick outside the supply rails, 30mA can create damaging voltage stress and is not guaranteed unless V_{DD} is limited to 3.15V.

TYPICAL PERFORMANCE CHARACTERISTICS LTC6957-1

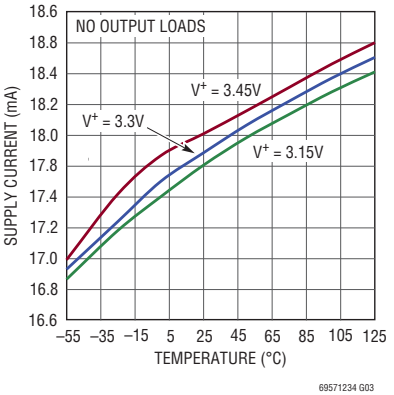
Input Self Bias Voltage vs Temperature



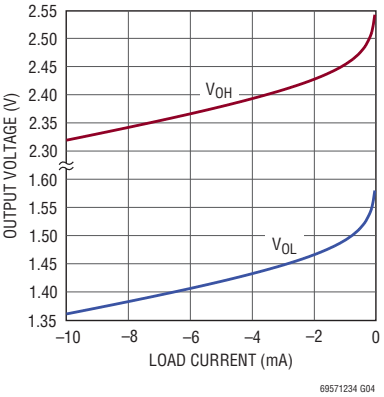
Supply Current vs Supply Voltage



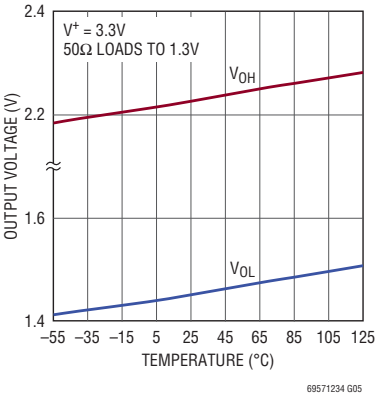
Supply Current vs Temperature



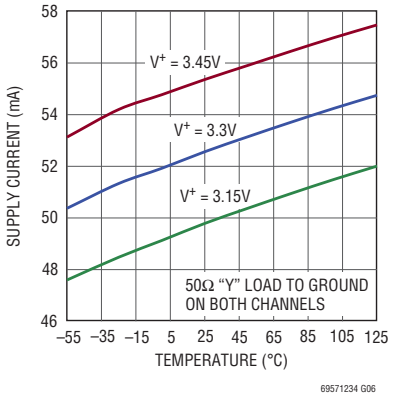
Output Voltage vs Load Current



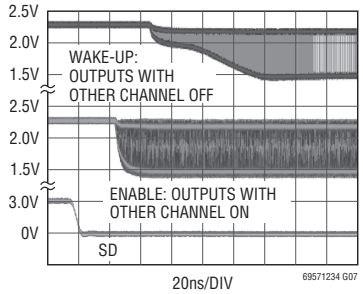
Output Voltage vs Temperature



Supply Current vs Temperature

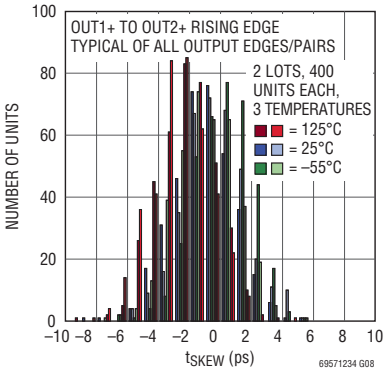


Enable and Wakeup



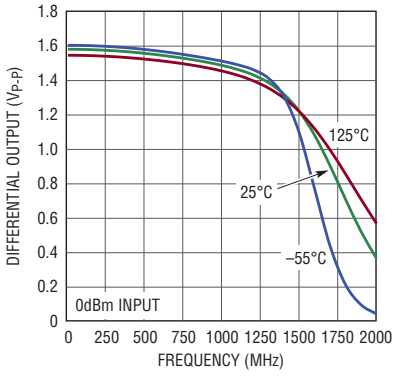
MULTIPLE EXPOSURES, PERSISTENCE MODE
CLOCK I/O = 120MHz
SD DRIVE ~ 140kHz, ASYNCHRONOUS

Typical Distribution of Skew



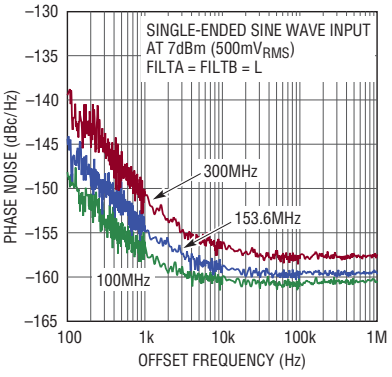
PRODUCTION DATA,
1ps RESOLUTION, ~1-2ps UNCERTAINTY

Differential Output vs Frequency

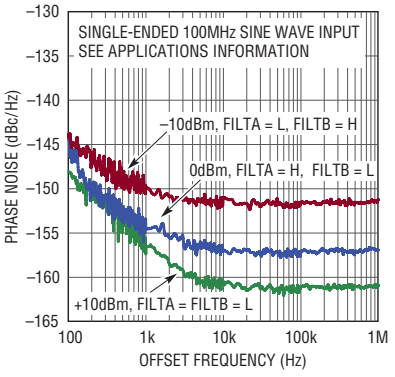


TYPICAL PERFORMANCE CHARACTERISTICS LTC6957-1

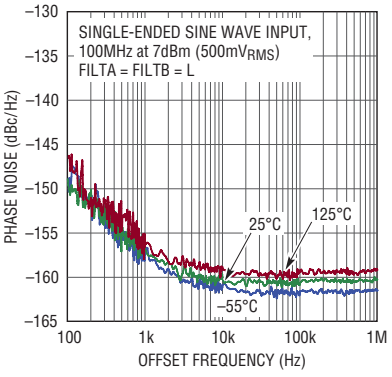
Additive Phase Noise vs Input Frequency



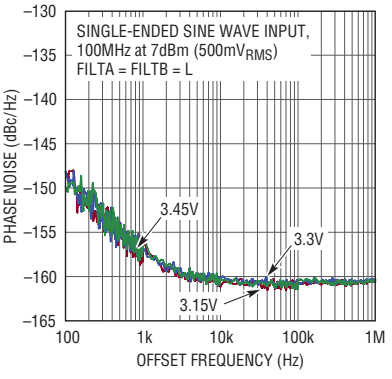
Additive Phase Noise vs Amplitude



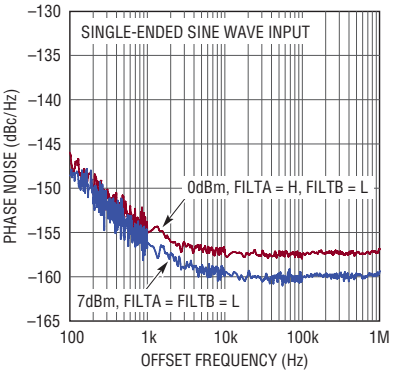
Additive Phase Noise vs Temperature



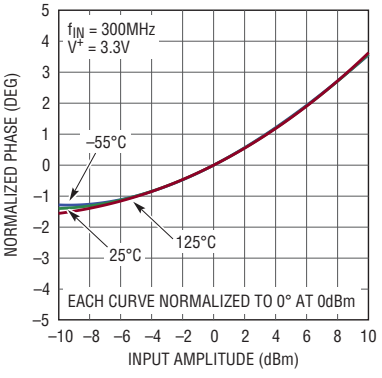
Additive Phase Noise vs Supply Voltage



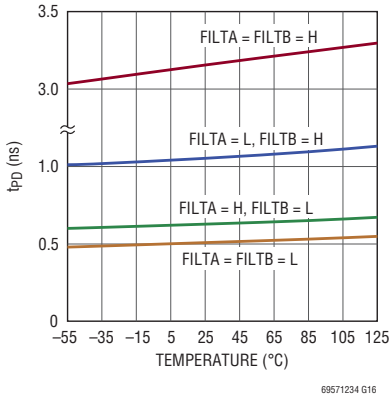
Additive Phase Noise at 122.88MHz



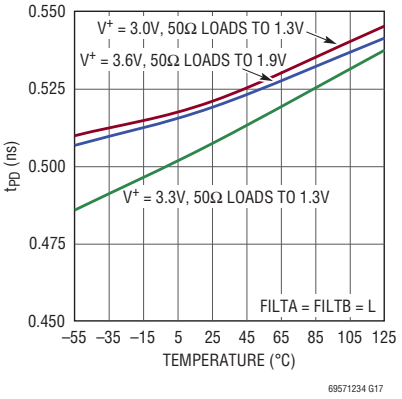
AM to PM Conversion



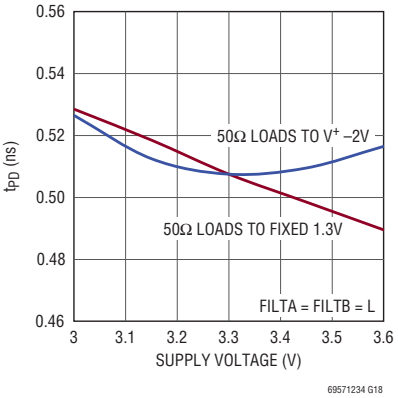
tPD vs Temperature



tPD vs Temperature

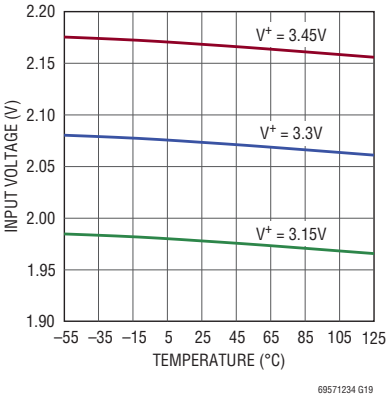


tPD vs Supply Voltage and Termination Voltage

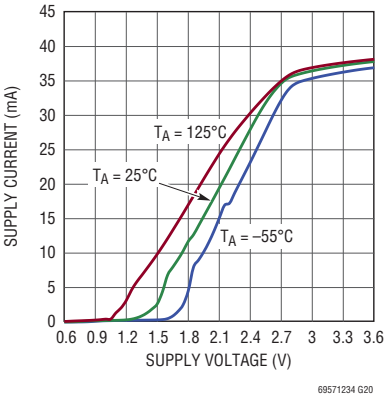


TYPICAL PERFORMANCE CHARACTERISTICS LTC6957-2

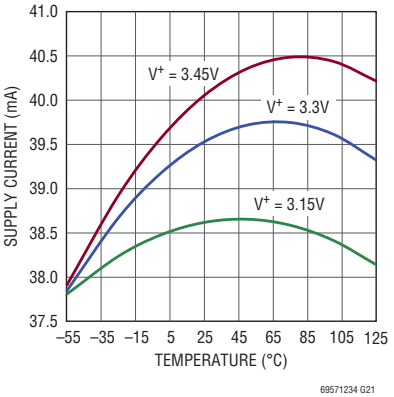
Input Self Bias Voltage vs Temperature



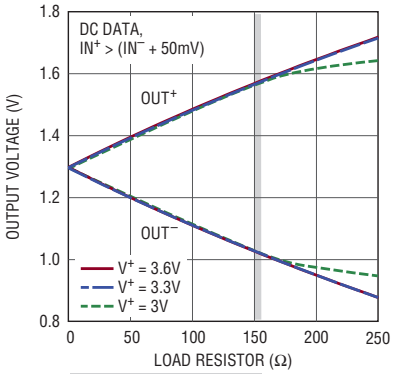
Supply Current vs Supply Voltage



Supply Current vs Temperature

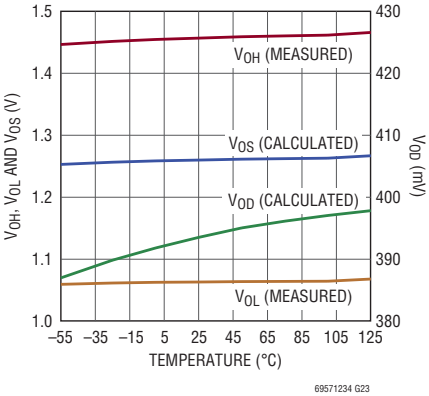


Output Voltages vs Load Resistor

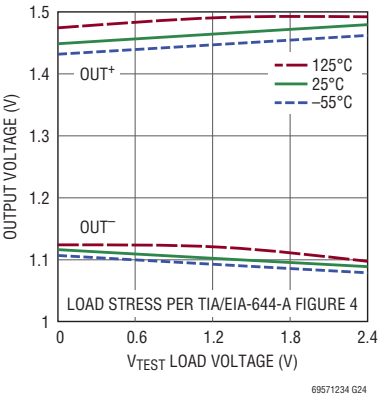


USE OF R_{LOAD} > 150Ω
NOT RECOMMENDED
f_{IN} MAY BE COMPROMISED

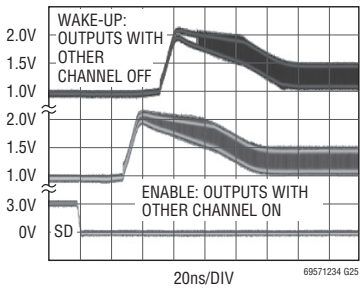
Output Voltages vs Temperature



Output Voltages vs Loading

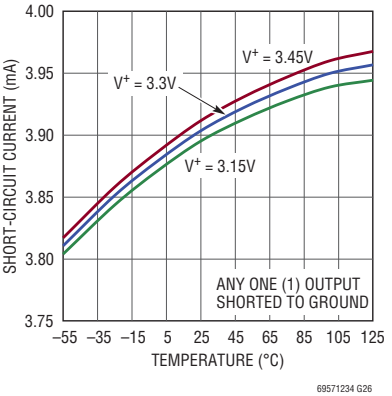


Enable and Wakeup

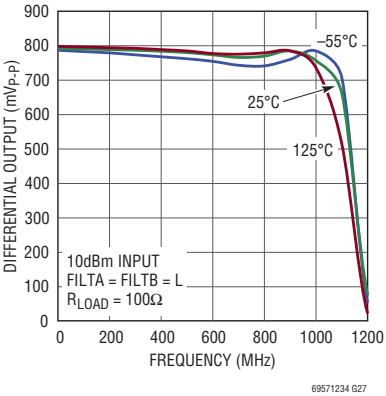


MULTIPLE EXPOSURES, PERSISTENCE MODE
CLOCK I/O = 120MHz
SD DRIVE ~ 140kHz, ASYNCHRONOUS

Output Short-Circuit Current vs Temperature

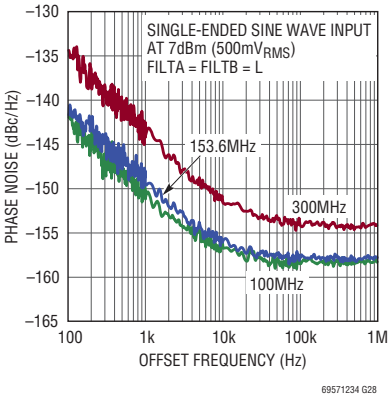


Differential Output vs Frequency

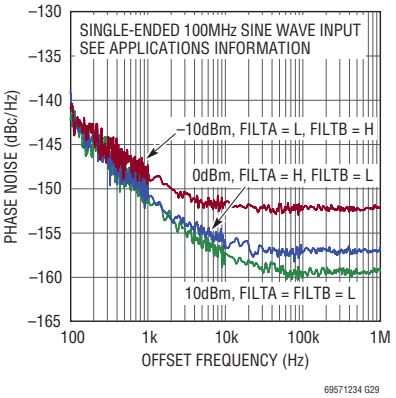


TYPICAL PERFORMANCE CHARACTERISTICS LTC6957-2

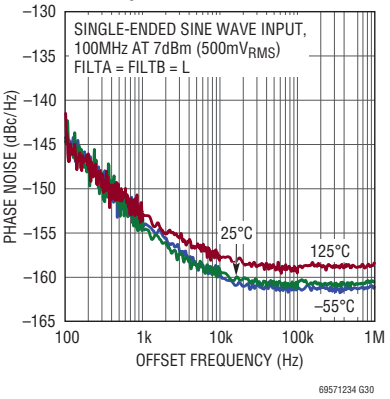
Additive Phase Noise vs Input Frequency



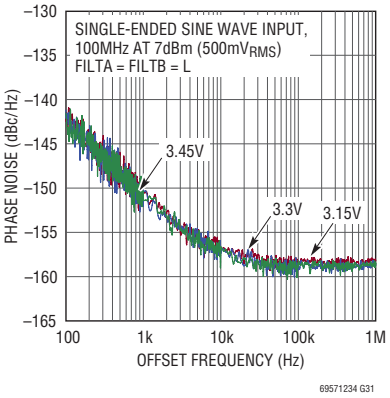
Additive Phase Noise vs Amplitude



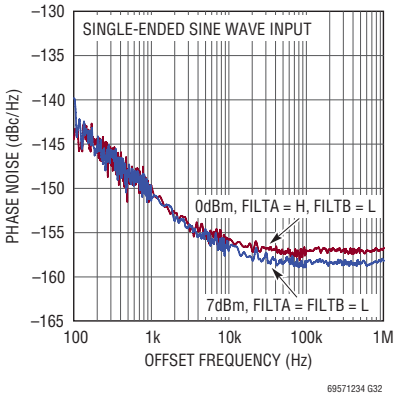
Additive Phase Noise vs Temperature



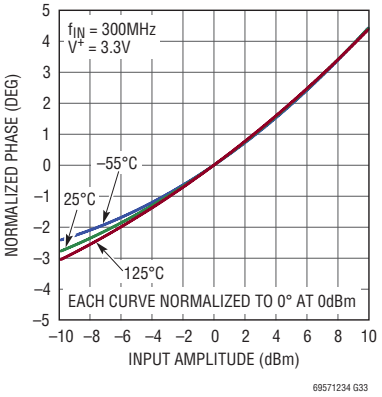
Additive Phase Noise vs Supply Voltage



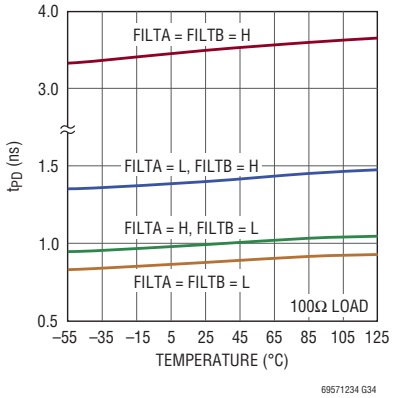
Additive Phase Noise at 122.88MHz



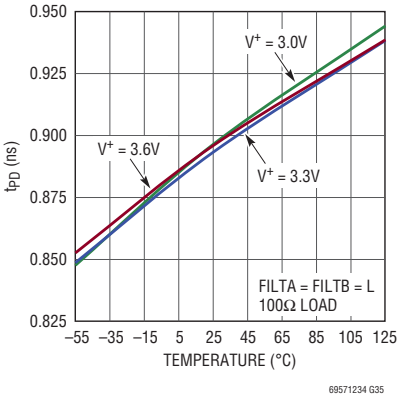
AM to PM Conversion



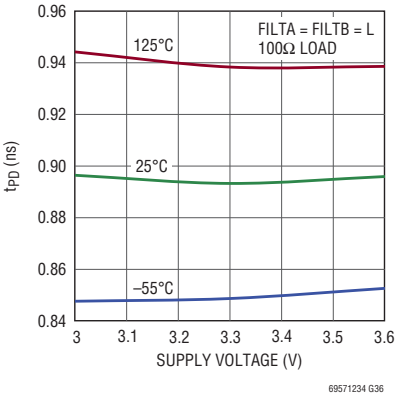
tPD vs Temperature



tPD vs Temperature

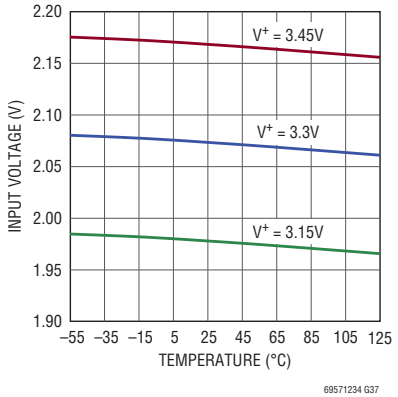


tPD vs Supply Voltage

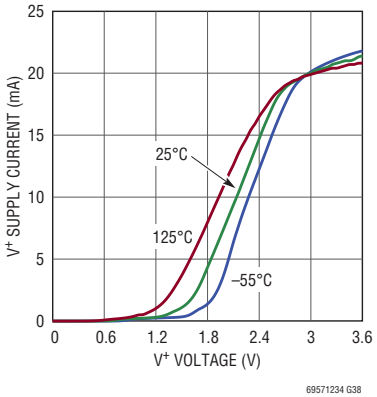


TYPICAL PERFORMANCE CHARACTERISTICS LTC6957-3/LTC6957-4

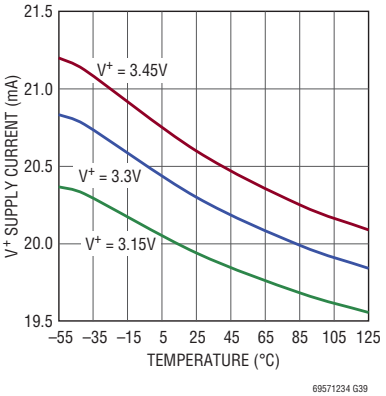
Input Self Bias Voltage vs Temperature



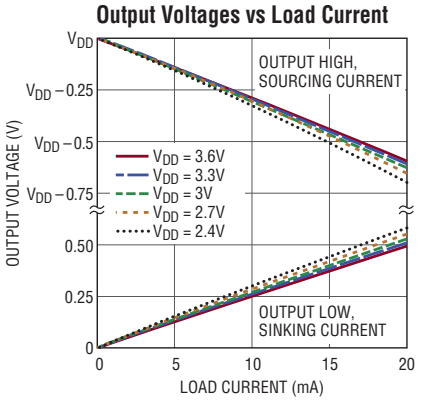
Supply Current vs Supply Voltage



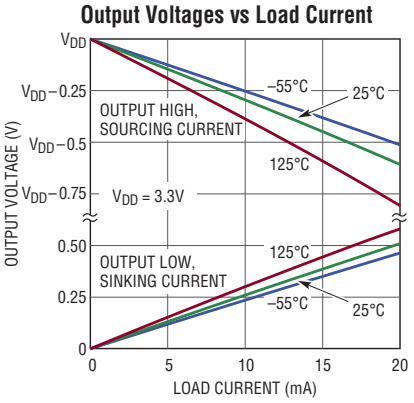
Supply Current vs Temperature



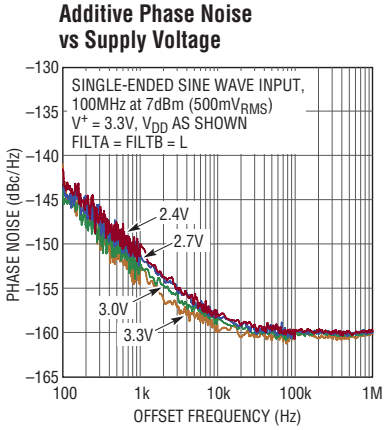
Output Voltages vs Load Current



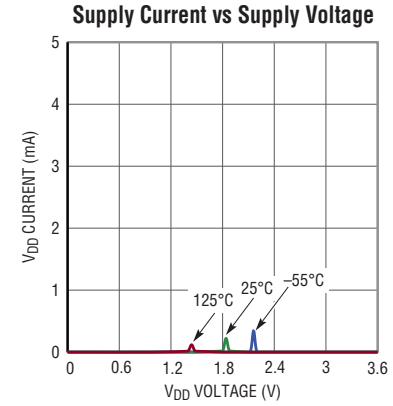
Output Voltages vs Load Current



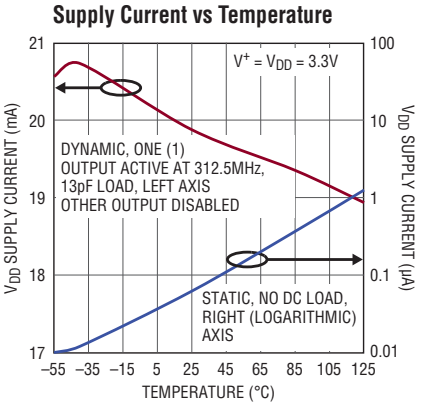
Additive Phase Noise vs Supply Voltage



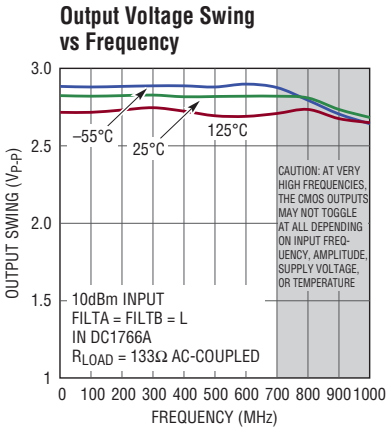
Supply Current vs Supply Voltage



Supply Current vs Temperature

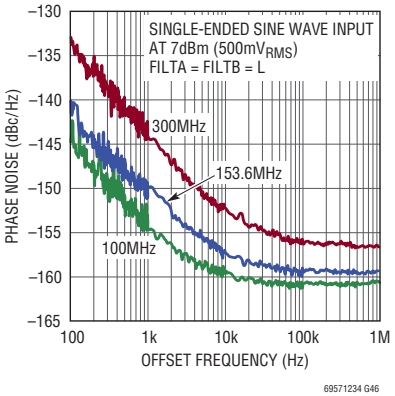


Output Voltage Swing vs Frequency

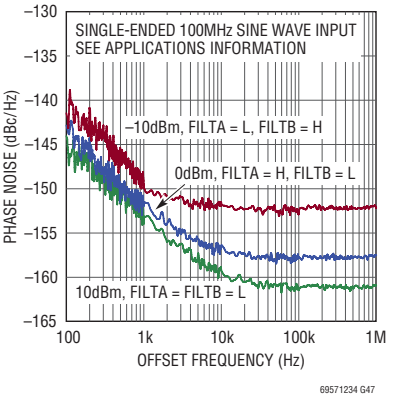


TYPICAL PERFORMANCE CHARACTERISTICS LTC6957-3/LTC6957-4

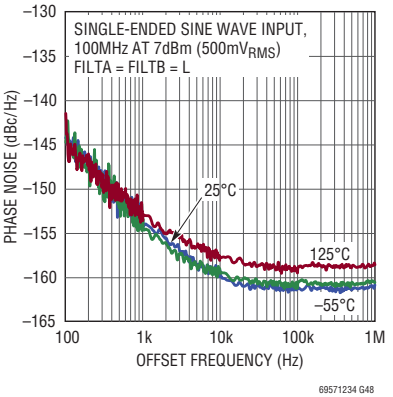
Additive Phase Noise vs Input Frequency



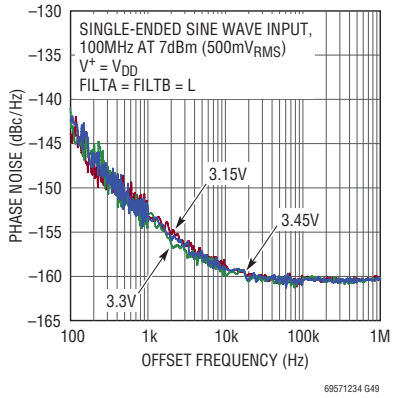
Additive Phase Noise vs Amplitude



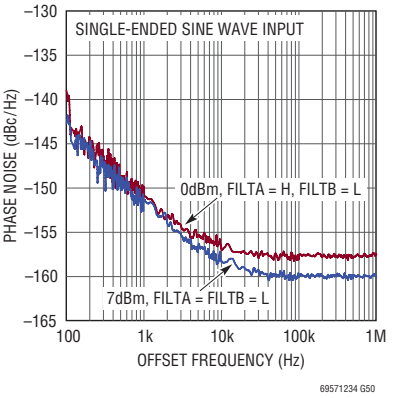
Additive Phase Noise vs Temperature



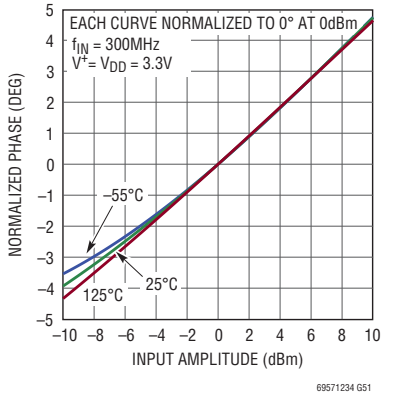
Additive Phase Noise vs Supply Voltage



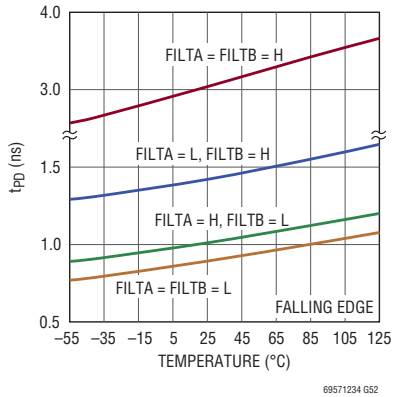
Additive Phase Noise at 122.88MHz



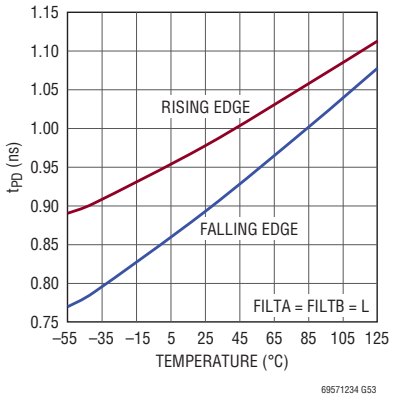
AM to PM Conversion



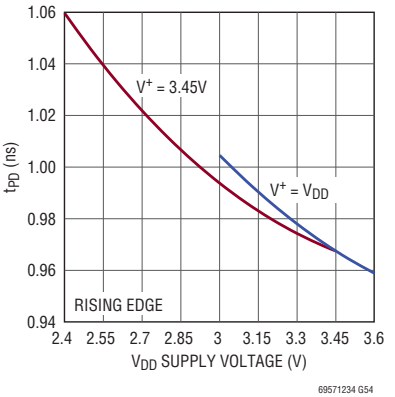
t_{PD} vs Temperature



t_{PD} vs Temperature



t_{PD} vs Supply Voltage



PIN FUNCTIONS

FILTA, FILTB (Pin 1, Pin 6): Input Bandwidth Limiting Control. These CMOS logic inputs control the bandwidth of the early amplifier stages. For slow slewing signals substantially lower phase noise is achieved by using this feature. See the Applications Information section for more details.

V^+ (Pin 2): Supply Voltage (3.15V to 3.45V). This supply must be kept free from noise and ripple. It should be bypassed directly to GND (Pin 5) with a 0.1µF capacitor.

IN^+, IN^- (Pin 3, Pin 4): Input Signal Pins. These inputs are differential, but can also interface with single-ended signals. The input can be a sine wave signal or a CML, LVPECL, TTL or CMOS logic signal. See the Applications Information section for more details.

GND (Pin 5): Ground. Connect to a low inductance ground plane for best performance. The connection to the bypass capacitor for V^+ (Pin 2) should be through a direct, low inductance path.

SD1, SD2 (Pin 12, Pin 7): Output Enable Control. These CMOS logic inputs control the enabling and disabling of their respective OUT1 and OUT2 outputs. When both outputs are disabled, the LTC6957 is placed in a low power shutdown state.

LTC6957-1 Only

$OUT1^-$, $OUT1^+$ (Pin 10, Pin 11): LVPECL Outputs. Differential logic outputs typically terminated by 50Ω connected to a supply 2V below the V^+ supply. Refer to the Applications Information section for more details.

$OUT2^-$, $OUT2^+$ (Pin 9, Pin 8): LVPECL Outputs. Differential logic outputs typically terminated by 50Ω connected to a supply 2V below the V^+ supply. Refer to the Applications Information section for more details.

LTC6957-2 Only

$OUT1^-$, $OUT1^+$ (Pin 10, Pin 11): LVDS Outputs, Mostly TIA/EIA-644-A Compliant. Refer to the Applications Information section for more details.

$OUT2^-$, $OUT2^+$ (Pin 9, Pin 8): LVDS Outputs, Mostly TIA/EIA-644-A Compliant. Refer to the Applications Information section for more details.

LTC6957-3/LTC6957-4 Only

OUT1, OUT2 (Pin 10, Pin 9): CMOS Outputs. Refer to the Applications Information section for more details.

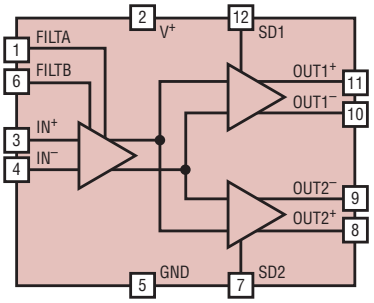
V_{DD} (Pin 11): Output Supply Voltage (2.4V to 3.45V). For best performance connect this to the same supply as V^+ (Pin 2). If the output needs to be a lower logic rail, this supply can be separately connected, but this voltage must be less than or equal to that on Pin 2 for proper operation. This supply must also be kept free from noise and ripple. It should be bypassed directly to the GNDOUT pin (Pin 8) with a 0.1µF capacitor.

GNDOUT (Pin 8): Output Logic Ground. Tie to a low inductance ground plane for best performance. The connection to the bypass capacitor for V_{DD} (Pin 11) should be through a direct, low inductance path.

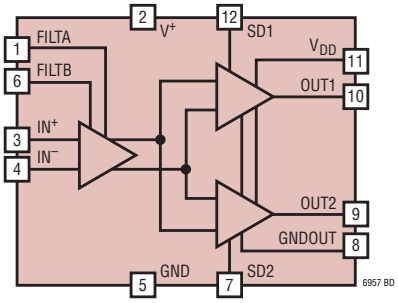
LTC6957-xDD Only

Exposed Pad (Pin 13): Always tie the underlying DFN exposed pad to GND (Pin 5). To achieve the rated θ_{JA} of the DD package, there should be good thermal contact to the PCB.

BLOCK DIAGRAMS

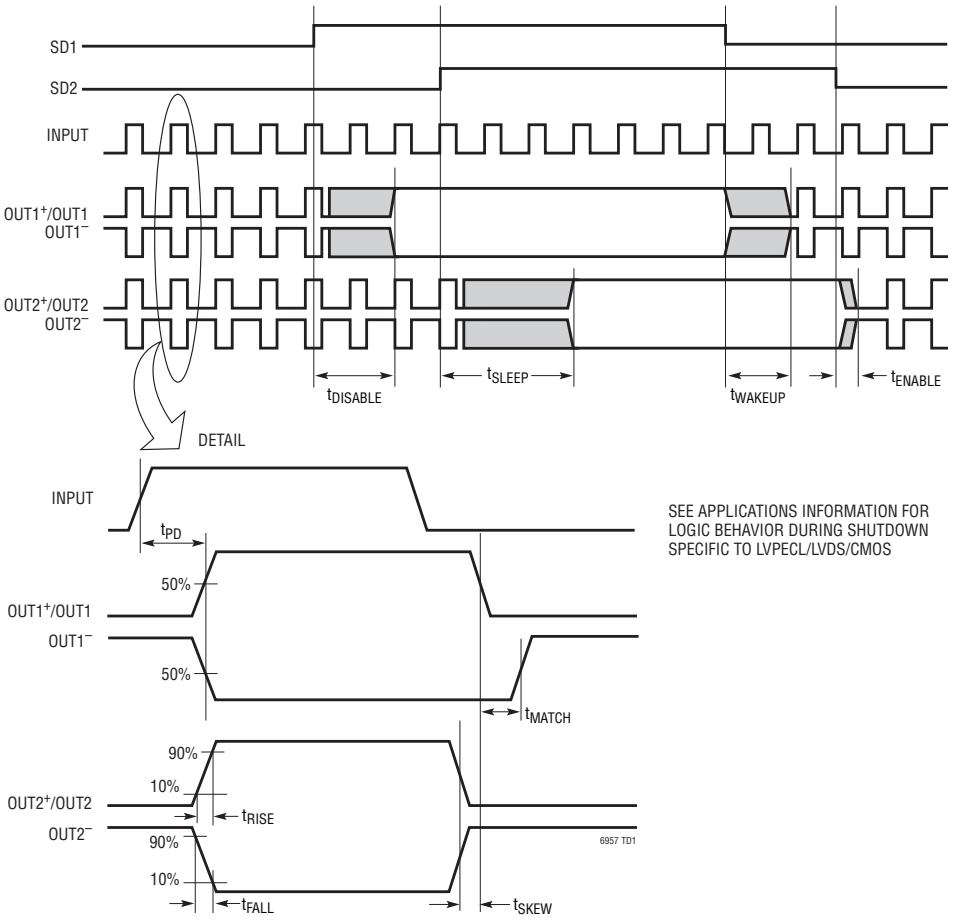


LTC6957-1 and LTC6957-2



LTC6957-3 and LTC6957-4

TIMING DIAGRAM



APPLICATIONS INFORMATION

General Considerations

The LTC6957-1/LTC6957-2/LTC6957-3/LTC6957-4 are low noise, dual output clock buffers that are designed for demanding, low phase noise applications. Properly applied, they can preserve phase noise performance in situations where alternative solutions would degrade the phase noise significantly. They are also useful as logic converters.

However, no buffer device is capable of removing or reducing phase noise present on an input signal. As with most low phase noise circuits, improper application of the LTC6957-1/LTC6957-2/LTC6957-3/LTC6957-4 can result in an increase in the phase noise through a variety of mechanisms. The information below will, hopefully, allow a designer to avoid such an outcome.

The LTC6957 is designed to be used with high performance clock signals destined for driving the encode inputs of ADCs or mixer inputs. Such clocks should not be treated as digital signals. The beauty of digital logic is that there is noise margin both in the voltage and the timing, before any deleterious effects are noticed. In contrast, high performance clock signals have no margin for error in the timing before the system performance is degraded. Users are encouraged to keep this distinction in mind while designing the entire clocking signal chain before, during, and after the LTC6957.

Input Interfacing

The input stage is the same for all versions of the LTC6957 and is designed for low noise and ease of interfacing to sine-wave and small amplitude signals. Other logic types can interface directly, or with little effort since they present a smaller challenge for noise preservation.

Figure 1 shows a simplified schematic of the LTC6957 input stage. The diodes are all for protection, both during ESD events and to protect the low noise NPN devices from being damaged by input overdrive.

The resistors are to bias the input stage at an optimal DC level, but they are too large to leave floating without increasing the noise. Therefore, for low noise use, always connect both inputs to a low AC impedance. A capacitor to ground/return is imperative on the unused input in single-ended applications.

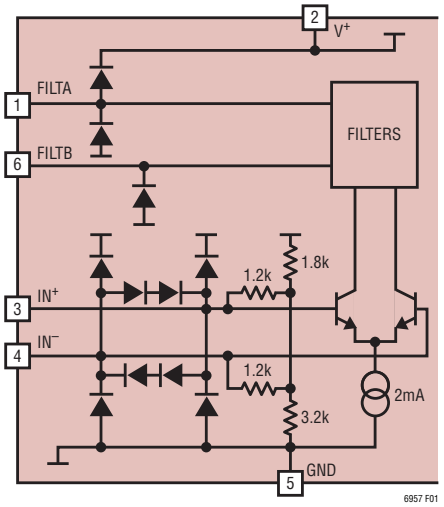


Figure 1

Figure 2a shows how to interface single-ended LVPECL logic to the LTC6957, while Figure 2b shows how to drive the LTC6957 with differential LVPECL signals. The capacitors shown are 10nF and can be inexpensive ceramics, preferably in small SMT cases. For use above 100MHz, lower value capacitors may be desired to avoid series resonance, which could increase the noise in Figure 2a even though the capacitor is just on the DC input. This comment applies to all capacitors hooked to the inputs throughout this data sheet.

In Figure 2a, the R_{TERM} implementation is up to the user and is to terminate the transmission line. If it is connected to a V_{TT} that is passively generated and heavily bypassed to ground, the 10nF to ground shown on the inverting LTC6957 input is the appropriate connection to use. However, if the termination goes to an actively generated V_{TT} voltage, lower noise may be achieved by connecting the capacitor on the inverting input to that V_{TT} rather than ground.

In Figure 2b, both inputs to the LTC6957 are driven, increasing the differential input signal size and minimizing noise from any common mode source such as V_{TT}, both of which improve the achievable phase noise.

A variety of termination techniques can be used, and as long as the two sides use the same termination, the configuration used won't matter much. In Figure 2b, the

APPLICATIONS INFORMATION

R_{TERM}s are shown in a "Y" configuration that creates a passive V_{TT} at the common point. Most 3.3V LVPECL devices have differential outputs and can be terminated with three 50Ω resistors as shown.

Figure 3 shows a 50Ω RF signal source interface to the LTC6957. For a pure tone (sine wave) input, Figure 3 can handle up to 10dBm maximum. A broadband 50Ω match as shown should suffice for most applications, though for small amplitude input signals a narrow band reactive matching network may offer incremental improvements in performance.

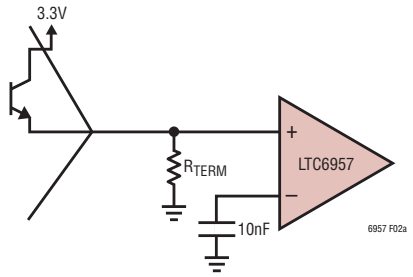


Figure 2a. Single-Ended LVPECL Input

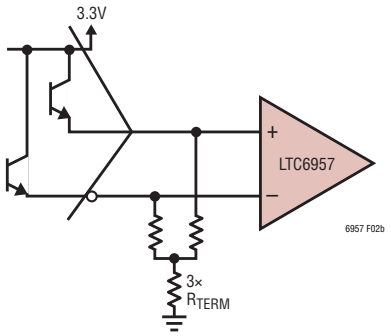


Figure 2b. Differential LVPECL Input

Figure 2

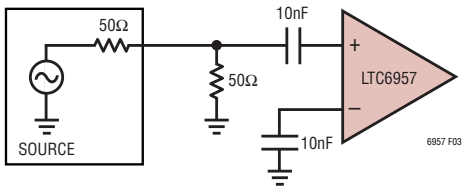


Figure 3. Single-Ended 50Ω Input Source

Figure 4 shows the interface between current mode logic (CML) signals and the LTC6957 inputs. The specifics of terminating will be dependent on the particular CML driver used; Figure 4 shows terminations only at the load end of the line, but the same LTC6957 interface is appropriate for applications with the source end of the line also terminated. In Figure 4a, a differential signal interface to the LTC6957 is shown, which must be AC-coupled due to the DC input levels required at the LTC6957.

Figure 4b shows a single-ended CML signal driving the LTC6957. This is not commonly used because of noise and immunity weaknesses compared to the differential CML case. Because the signal is created by a current pulled through the termination resistor, the signal is inherently referenced to the supply voltage to which R_{TERM} is tied. For that reason, the other LTC6957 should be AC-referenced to that supply voltage as shown.

The polarity change shown here is for graphic clarity only, and can be reversed by swapping the LTC6957 input terminals.

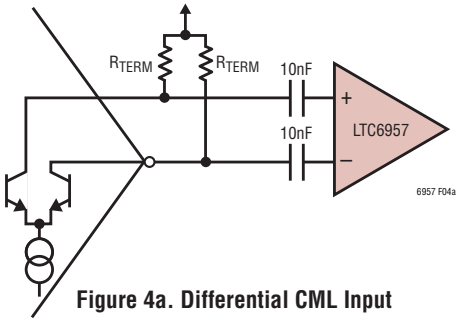


Figure 4a. Differential CML Input

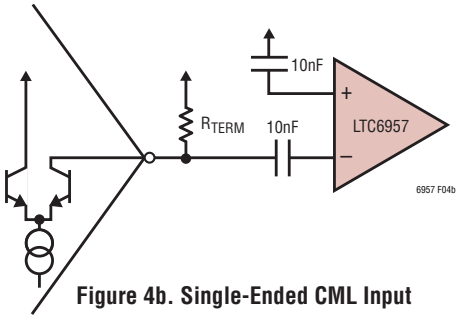


Figure 4b. Single-Ended CML Input

Figure 4

APPLICATIONS INFORMATION

Figure 5 shows the LTC6957 being driven by an LVDS (EIA-644-A) signal pair. This is simply a matter of differentially terminating the pair and AC-coupling as shown into the LTC6957 whose DC common mode voltage is incompatible with the LVDS standard.

The choice of 110Ω versus 100Ω termination is arbitrary (the EIA-644-A standard allows 90Ω to 132Ω) and should be made to match the differential impedance of the trace pair. The termination and AC-coupling elements should be located as close as possible to the LTC6957.

If DC-coupling is desired, for example to control the LTC6957 output phasing during times the LVDS input clocks will be halted, a pair of 3k resistors can parallel the two capacitors in Figure 5. An EIA/TIA-644-A compliant driver can drive this load, which is less load stress than specification 4.1.1. The differential voltage into the LTC6957 when clocked (>100kHz) will be full LVDS levels. When the clocks stop, the DC differential voltage created by the resistors and the 1.2k internal resistors (Figure 1) will be 100mV, still sufficient to assure the desired LTC6957 output polarity. Choosing the smallest capacitors needed for phase noise performance will minimize the settling transients when the clocks restart.

Interfacing with CMOS Logic

The logic families discussed and illustrated to this point are generally a better choice for routing and distributing low phase-noise reference/clock signals than is CMOS logic. All of the logic types shown so far are well suited for use with low impedance terminations. Most of the time there is a differential signal when using LVPECL or CML, and LVDS always has a differential signal. Differential signals provide lots of margin for error when it comes to picking up noise and interference that can corrupt a reference clock.

CMOS on the other hand cannot drive 50Ω loads, is usually routed single-ended, and by its nature is coupled to the potentially noisy supply voltage half the time.

The LTC6957-3/LTC6957-4 provide CMOS outputs, so it may seem surprising to read herein that CMOS is a poor choice for low phase noise applications. However, these devices should prove useful for designers that recognize the challenges and limitations of using CMOS signals for low phase noise applications. See the CMOS Outputs of the LTC6957-3/LTC6957-4 section for further information.

The best method for driving the LTC6957 with CMOS signals would be to provide differential drive, but if that is not available, there are few ways to create a differential CMOS signal without running the risk of corrupting the skew or creating other problems. Therefore, single-ended CMOS signals are the norm and care must be taken when using this to drive the LTC6957.

The primary concern is that all routing should be terminated to minimize reflections. With CMOS logic there is usually plenty of signal (more than the LTC6957 can handle without attenuation) and the amplitude of the LTC6957 input signal will generally be of secondary importance compared to avoiding the deleterious effects of signal reflections. The primary concern about terminations is that the input waveform presented to the LTC6957 should have full speed slewing at the all important transitions. If a rising edge is slowed by the destructive addition of the ringing/settling of a prior edge reflection, or even the start of the current edge, the phase noise performance will suffer. This is true for all logic types, but is particularly problematic when using CMOS because of the fast slew rates and because it does not naturally lend itself to clean terminations.

Point-to-point routing is best, and care should be taken to avoid daisy-chain routing, because the terminated end may be the only point along the line that sees clean transitions. Earlier loads may even see a dwell in the transition region which will greatly degrade phase noise performance.

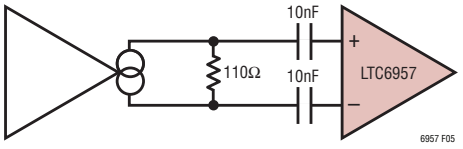


Figure 5. LVDS Input

APPLICATIONS INFORMATION

Figure 6 shows a suggested CMOS to LTC6957 interface. The transmission line shown is the PCB trace and the component values are for a characteristic impedance of 50Ω, though they could be scaled up or down for other values of Z0. The R1/R2 divider at the CMOS output cuts the Thevenin voltage in half when the Z_{OUT} of the driver is included. More importantly, it drives the transmission line with a Thevenin driving resistance of 50Ω, matching the Z0 of the line. On the other end of the line, a 50Ω load is presented, minimizing reflections. This results in a second 2:1 attenuation in voltage, so the LTC6957 input will be approximately 800mV_{P-P} with 3V CMOS; 1.25V_{P-P} with 5V and 600mV_{P-P} with 2.5V. All of these levels are less than the maximum input swing of 2V_{P-P} yet with clean edges and fast slew rates should be able to realize the full phase noise performance of the LTC6957.

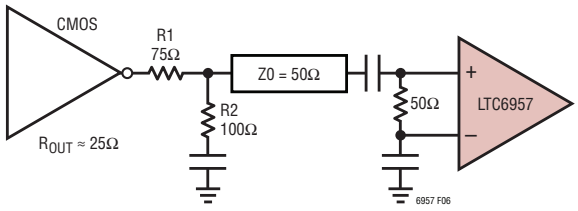


Figure 6. CMOS Input

The various capacitors are for AC-coupling and should have Z << 50Ω at the operating frequency. The capacitors allow the LTC6957 to set its own DC input bias level, and reduce the DC current drain, which at 12.4mA (for the case of a driver powered from 3.3V) is significant. This current drain can be reduced (with some potential for a noise penalty) by increasing the attenuation at the R1/R2 network, taking care to keep the Thevenin impedance equal to the Z0 of the trace.

When using CMOS logic, it is important to consider how all of the output drivers, in the same IC, are being used. For best performance, the entire IC should be devoted to driving the LTC6957, or if other gates in the same package must be put to use, they should only carry the same timing signal (such as for fan-out) or be multiplexed in time so that only one timing signal is being processed at a time, such as for multiplexing selective shutdowns of different segments of a system. Otherwise performance is likely to suffer with spurs or other interference in the phase noise spectrum related to the other signals processed in the driver.

Input Resistors

The LTC6957 input resistors, seen in Figure 1, are present at all times, including during shutdown. Although they constitute a large portion of the shutdown current, this behavior prevents the shutdown and wake-up cycling of the LTC6957 from "kicking back" into prior stages, which could create large transients that could take a while to settle. Particularly in the common case of AC-coupling where the coupling cap charge is preserved.

Input Filtering

The LTC6957 includes input filtering with three narrowband settings in addition to the full bandwidth limitation of the circuit design.

Table 1

FILTA	FILTB	BANDWIDTH
Low	Low	1200MHz (Full Bandwidth)
High	Low	500MHz (–3dB)
Low	High	160MHz (–3dB)
High	High	50MHz (–3dB)

For slow slewing signals (i.e., <100MHz sine wave signals) substantially lower phase noise can be achieved by using this feature. Bandwidth limiting is useful because it limits the impact of all of the spectral energy that will alias down to (on top of) the fundamental frequency.

The best filter setting to use for a given application will depend on the clock frequency, amplitude, and waveform shape, with the single biggest determinant being the slew rate at the input of the LTC6957. Any amplifier noise will add phase noise inversely proportional to its input slew rate, just from the dV/dt changing voltage noise to time base noise. But a fast slew rate may not be possible with other design constraints, such as the use of sine waves for EMI/RFI reasons, signal losses, etc. A limiting amplifier such as the LTC6957 should have enough bandwidth to preserve the slew rate of the input. But any additional bandwidth will provide no improvement in phase noise due to slew rate preservation, while incurring a phase noise penalty from noise aliasing.

APPLICATIONS INFORMATION

Table 2 has the slew rate ranges most suitable for the four different filter settings.

Table 2

FILTA	FILTB	INPUT SLEW RATE (V/µs)
Low	Low	>400
High	Low	125 to 400
Low	High	40 to 125
High	High	<40

Another way to look at this is to consider the case of sine waves, for which the frequency ranges will depend on input amplitudes, as illustrated in Table 3.

Table 3

	FREQUENCY RANGE			
INPUT AMPLITUDE (dBm)	FILTA = L, FILTB = L (MHz)	FILTA = H, FILTB = L (MHz)	FILTA = L, FILTB = H (MHz)	FILTA = H, FILTB = H (MHz)
10	>63	20 to 63	6.3 to 20	<6.3
5	>112	35 to 112	11 to 35	<11
0	>200	63 to 200	20 to 63	<20
–5		>112	35 to 112	<35
–10		>200	63 to 200	<63

Figure 7 has LTC6957-1 100MHz additive phase noise measurements that illustrate the trade-offs between filter settings at various input slew rates. Each of the three charts has all four filter settings, and one input amplitude; Figure 7a has a +10dBm input, Figure 7b has a 0dBm input, and Figure 7c has a –10dBm input. The four filter settings are shown in the same colors throughout.

With +10dBm at 100MHz, the input slew rate is 628V/µs and Table 2 indicates the best filter setting to use is FILTA = FILTB = L, which is seen to be the case in Figure 7a.

The noise at the next filter setting is only slightly higher, but for the maximum filtering case there is a full 10dB of additional noise.

With 0dBm at 100MHz, the input slew rate is 198V/µs and Table 2 indicates the best filter setting to use is FILTA = H, FILTB = L. Again this is seen to be the case in Figure 7b. As the input was decreased 10dB from Figure 7a to Figure 7b, the blue trace rose 5dB while the green trace only rose 3dB.

With –10dBm at 100MHz, the input slew rate is 63V/µs and Table 2 indicates the best filter setting to use is FILTA = L, FILTB = H. Again this is seen to be the case in Figure 7c. As the input was decreased 10dB from Figure 7a to Figure 7b, and again to Figure 7c, the red trace rose just 3dB then another 4dB, while the green and blue traces rose much faster.

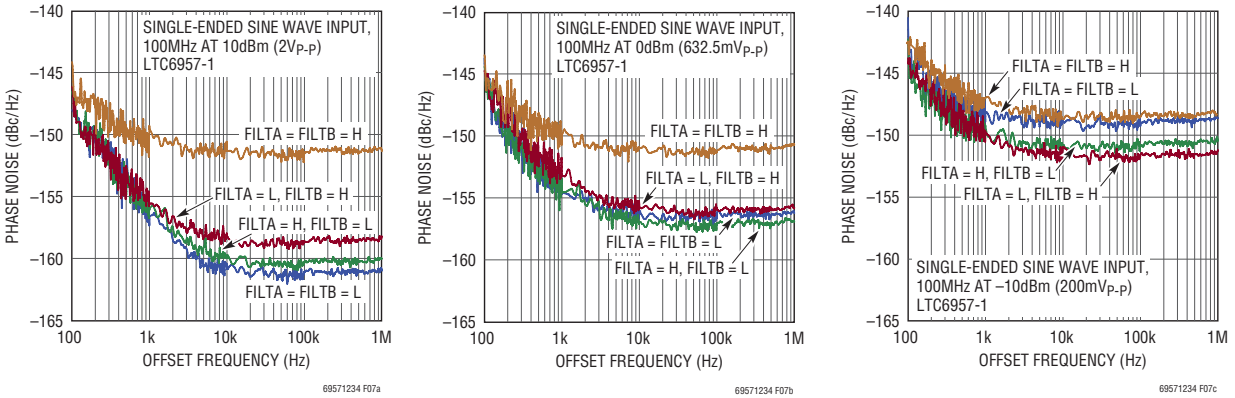


Figure 7. 100MHz Additive Phase Noise with Varying Input Amplitudes

APPLICATIONS INFORMATION

One important observation to take away from Figures 7a to 7c is that while the worst filter settings for a given set of conditions should certainly be avoided, it doesn't matter nearly as much if the optimal or next to optimal filter setting is used, because they are always fairly comparable in terms of phase noise. So if a design will have an octave or two range of amplitudes or frequencies, it is sufficient to choose the filter setting whose range most closely matches the application's range when using Tables 2 or 3 and the noise penalty will not be severe anywhere in the range.

Evidently, the input filtering will not significantly help with large and fast slewing input signals to the LTC6957. As seen in Figure 1, the input has a differential pair before the filters, so the limiting will already have happened before the filter. Fortunately, with large input signals, performance is typically better than with smaller input signals because phase noise is a signal-to-noise phenomenon.

Input Drive and Output Skew

All versions of the LTC6957 have very good output skew; the specification limits consist almost entirely of test margins. Even laboratory verification of the skew between different outputs is a challenging exercise, given the need to measure within ±1ps. With electromagnetic propagation velocity in FR-4 being well known as 6" per nanosecond, the skew of the LTC6957 will be impacted by PCB trace routing length differences of just 6mils.

The LTC6957 t_{PD} and t_{SKEW} are specified for a 100mV step with 50mV of overdrive. This is common for high speed comparators, though it may not reflect the typical application usage of parts such as the LTC6957. The propagation delay of the LTC6957 will increase with less overdrive and decrease with more overdrive, as would that of a high speed comparator. To a lesser extent, having the same overdrive but a larger signal (for instance a differential input step of –200mV to 50mV) will increase propagation delay, though this effect is smaller and can usually be ignored.

A consequence of this behavior may be a perceived mismatch between the propagation delay for rising versus falling edges when driven with an AC-coupled input whose duty cycle is not exactly 50%. The LTC6957 inputs are internally DC-coupled, and as shown in Figure 1, biasing is provided at ~64% of the supply voltage. AC-coupled input signals with a duty cycle of exactly 50% will see symmetric levels of overdrive for the two signal directions. If, for example, the input signal is a $100mV_{P-P}$ square wave with a duty-cycle of 48%, meaning it is high 48% of the time and low 52% of the time, the DC average will be 48mV above the low voltage level. This means the rising edge has 52mV of overdrive, and the falling edge has 48mV of overdrive.

As a result of this, the rising edge t_{PD} will be faster than the falling edge t_{PD}. Fortunately, this will make the output duty cycle closer to 50% than the input duty cycle. Figure 8 is from measurements on the LTC6957-2, with a 2V to 2.1V square wave on IN^-, and with IN^+ set to various DC voltages between those two levels. The X-axis is the overdrive level for the $t_{PD}+$ data, and is 100mV minus the overdrive level for the $t_{PD}–$ data, to illustrate the level of t_{PD} changes that can unexpectedly occur with AC-coupling. The lines are dashed where the measurement uncertainty becomes large, when single digit millivolts and picoseconds are being measured[1]. As can be seen, the $t_{PD}+/t_{PD}–$ mismatch is very good at 50mV where the two overdrive levels are the same.

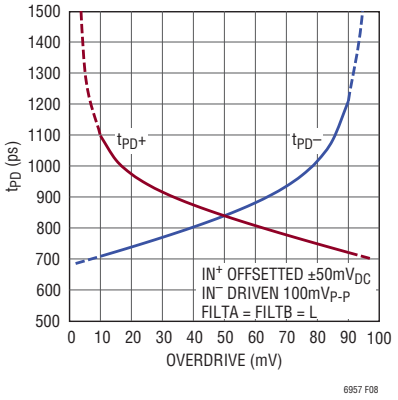


Figure 8. LTC6957-2 Propagation Delay vs Overdrive

[1] Below 2mV to 3mV, the input offset and the small input hysteresis play a role too. Fortunately, neither is large enough to be a concern in normal operation.

APPLICATIONS INFORMATION

This data is shown for the LTC6957-2, but the effect is due to the input stage that is common to all versions, so any other version will have the same general behavior.

The LTC6957-3 and LTC6957-4 CMOS outputs may have additional t_{PD}+ vs t_{PD}– discrepancies due to differences between the NMOS and PMOS output devices, particularly when driving heavy loads. These are independent of input overdrive, but can change with supply voltage and temperature, and can vary part to part. The complementary outputs of the LTC6957-4 will therefore be higher skew than the like edges of the LTC6957-3. Both the LTC6957-3 and LTC6957-4 will have large (120ps typ) t_{PD}^{+} to t_{PD}^{-} discrepancies compared to LVPECL or LVDS outputs.

LVPECL Outputs of the LTC6957-1

Figure 9 shows a simplified schematic of the LTC6957-1 LVPECL output stage. As with most ECL outputs, there are no internal pull-down devices so the user must provide both termination and biasing external to the device. Note that only the current source is cut off during shutdown. The bases of the output NPNs are still tied to the pull-up resistors, so both outputs will be pulled high in shutdown, and it is the user's responsibility to disconnect the external loading if power reduction is to be realized.

The simplest way to terminate and bias the LTC6957-1 outputs is to route the differential output to the differential receiver and terminate the lines at that point with the three resistor network shown in Figure 9. The differential termination will be 100Ω, while the common mode termination will be 75Ω which could result in additional common mode susceptibility. A bypass capacitor on the midpoint of the Y can be used to improve this.

If the common mode termination impedance is not an issue, the three resistor Y configuration can be changed to a three resistor delta configuration, which is a simpler layout in most cases.

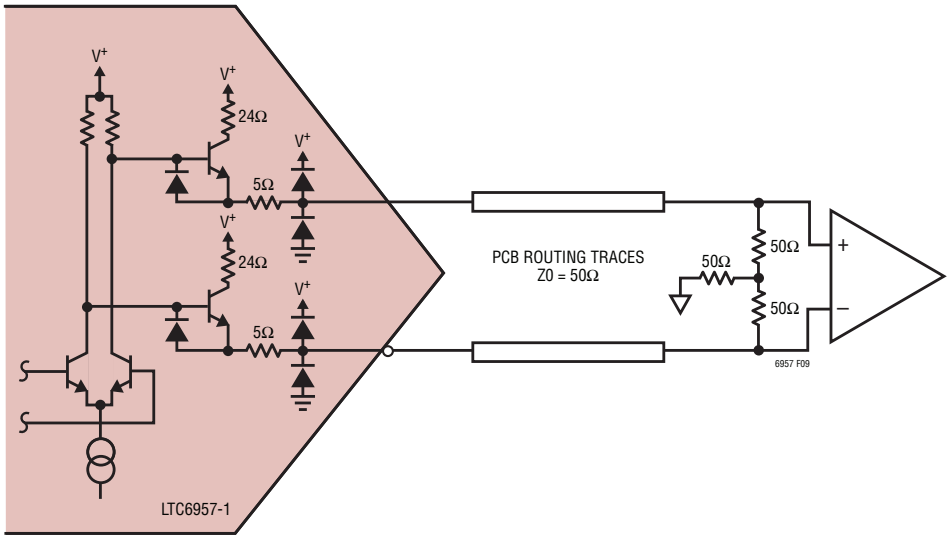


Figure 9. LTC6957-1 LVPECL Outputs

APPLICATIONS INFORMATION

During transitions to and from shutdown, the LTC6957-1 outputs are not guaranteed to comply with the specified output levels for any length of time after the rising edge of SD1/SD2, nor for any time before sufficient t_{WAKEUP}/t_{ENABLE} subsequent to the falling edge of SD1/SD2. The output common mode and differential voltage could have a slow settling time compared to the signal frequency, and a long string of runt pulses could be seen. The LTC6957-1 shutdown capability should be used as a slow on/off control, not a logic gating/enable control.

Power Supplies for LVPECL Operation

The LTC6957-1 can operate from 3.15V to 3.45V total supply voltage difference, irrespective of the absolute level of those voltages. The convention in LVPECL is that the negative supply is ground, while in ECL the positive supply may be ground or 2.0V. The LTC6957-1 can work in all of these situations provided the total supply voltage difference is within the 3.15V to 3.45V range. No special supply sequencing will be needed. With a 2V rail the output terminations go to ground, while, with the positive supply grounded, the outputs can tolerate short circuits to ground. However, the four CMOS logic input signals will need to be driven with respect to whatever absolute levels of supply voltages are used. If FILTA, FILTB, SD1, and SD2 are fixed, they can be tied to the appropriate rail and this is not a problem. Interface logic levels could get tricky if they need to be programmed in-system.

In any voltage configuration, be aware that the LVPECL output stage depends on the external load to complete its biasing and, as such, is susceptible to phase modulation as the supply voltage changes. The LTC6957-1 is generally less sensitive to variations in the supply voltage if the termination voltage tracks the supply rather than ground.

With all four outputs terminated or otherwise driving heavy loads, the LTC6957-1 power consumption and temperature rise may be an issue.

Fortunately, the data sheet specification for supply current with output loads does not need to be multiplied by the entire supply voltage to calculate on-chip power dissipation because most of that current flows through the loads which will dissipate a significant portion of the total system power.

Typically, the internal power consumption will be (20mA • 3.3V =) 66mW, while the on-chip power dissipation from the output loading will be less than half that number. With a total power dissipation on-chip of 90mW, the temperature rise in the MS-12 package will be 13°C given the θ_{JA} of that package. For use to 125°C ambient (H-grade) designers should be sure to check the temperature rise using their specific output loading and supply levels. The Absolute Maximum rating for Junction Temperature is 150°C, and must be avoided to prevent damaging the device, and as stated in Note 1: "Exposure to any Absolute Maximum Rating condition for extended periods of time may affect device reliability and lifetime."

APPLICATIONS INFORMATION

LVDS Outputs of the LTC6957-2

Figure 10 shows a simplified schematic of the LTC6957-2 LVDS output stage. The TIA/EIA-644-A standard specifies the generator electrical requirements for this type of interface, and the LTC6957-2 has been verified against that standard using the following test methods:

SPECIFICATION	LEVEL OF TESTING
4.1.1	100% Production Tested
4.1.2	100% Production Tested
4.1.3	100% Production Tested
4.1.4	100% Production Tested*
4.1.5	Lab Verification of Design Only
6a	100% Production Tested
6b	100% Production Tested
6c	100% Production Tested

*The t_{RISE}/t_{FALL} of the LTC6957-2 are not compliant with the standard so as to preserve full phase noise performance. To slow the edge rates, add differential capacitance across the outputs. 2.7pF is sufficient to meet the standard.

The TIA/EIA-644-A standard does not cover driver characteristics during shutdown nor the transitions to and from shutdown. The LTC6957-2 outputs are not guaranteed to comply with the standard for any length of time after the rising edge of SD1/SD2, nor for any time before sufficient t_{WAKEUP}/t_{ENABLE} subsequent to the falling edge of SD1/SD2. The output common mode voltage (V_{OS} in 644-A parlance) could have a slow settling time compared to the signal frequency, and a long string of runt pulses could be seen. The LTC6957-2 shutdown capability should be used as a slow, power-saving on/off control, not a logic gating/enable control.

Power Supplies for LVDS Operation

The LTC6957-2 has a single supply that should be within the 3.15V to 3.45V range.

The LTC6957-2 power supply voltage can corrupt the spectral purity of the clock signal, though to a lesser degree than with any of the other options. See the Typical Performance Characteristic chart t_{PD} vs Supply Voltage.

When using both LVDS channels, the LTC6957-2 power consumption can exceed 120mW, which results in a junction-to-ambient rise of 17.4°C in the MS-12 package, more when operated at 3.45V. Again, it is up to the user to always avoid junction temperatures above the Absolute Maximum rating, and to stay comfortably below it for any extended periods of time.

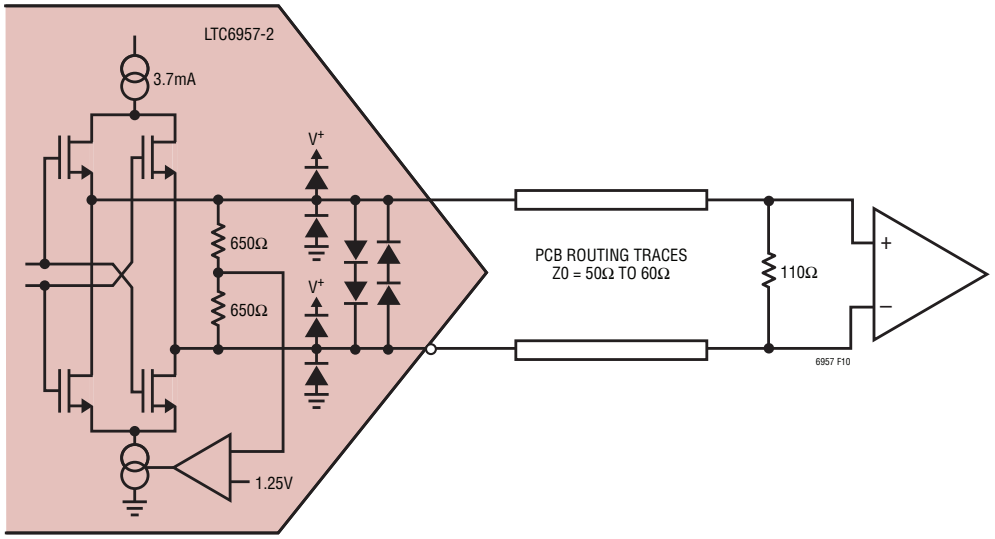


Figure 10. LTC6957-2 LVDS Outputs

APPLICATIONS INFORMATION

CMOS Outputs of the LTC6957-3/LTC6957-4

Figure 11 shows a simplified schematic of the LTC6957-3/LTC6957-4 CMOS output stage. The LTC6957-3 outputs are driven synchronously in-phase, while the LTC6957-4 outputs are driven differentially out-of-phase.

Although the LTC6957-3/LTC6957-4 are specified for a resistive load, the outputs can drive capacitive loads as well. With more than a few picoFarads of load, the rise and fall times will be degraded in direct proportion to the load capacitance.

During shutdown, the LTC6957-3 outputs will both be set to a logic low.

During shutdown, the LTC6957-4 OUT1 will be set to a logic low, while OUT2 will be set to a logic high.

During transitions to and from shutdown, the LTC6967-3/LTC6957-4 outputs may not comply with the specified output levels for any length of time after the rising edge of SD1/SD2, nor for any time before sufficient t_{WAKEUP}/t_{ENABLE} subsequent to the falling edge of SD1/SD2. The outputs may have one or two errant transitions resulting in runt pulses being seen. The LTC6957-3/LTC6957-4 shutdown capability should be used as a slow, power-saving on/off control, not a logic gating/enable control, and because they can not be put in a high impedance (3-state) condition, the shutdown functionality is not usable as a way to multiplex multiple outputs or devices.

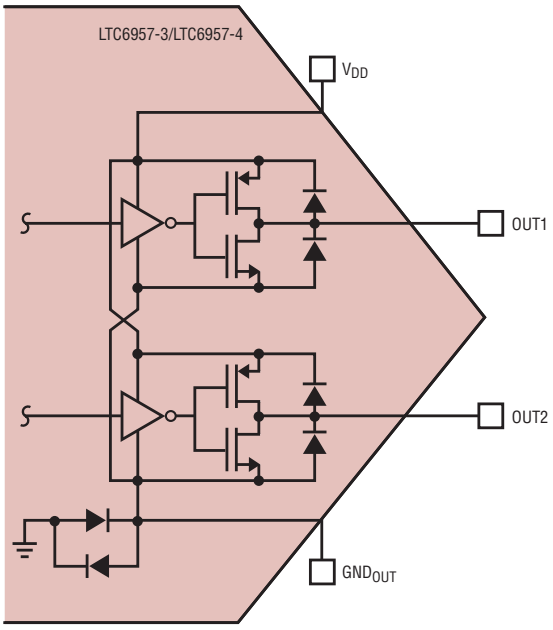


Figure 11. LTC6957-3/LTC6957-4 CMOS Outputs

Power Supplies for CMOS Operation

The LTC6957-3/LTC6957-4 operate with V^+ from 3.15V to 3.45V only. If the LTC6957-3/LTC6957-4 are used to drive CMOS logic at a lower voltage rail, the output stage can be powered (Pin 11) by a lower voltage, down to $2.4V_{MIN}$. Note that significant degradation of the spectral purity could occur if the output supply, V_{DD}, is not clean, either because of additional broadband noise or discrete spectral tones. The nature of a CMOS logic gate forms an AM modulator of low frequency disturbances on the power/ground that modulate the signal propagating through the CMOS gate. Numerous common phenomena can serve to convert the AM to PM/FM and, even if the conversion efficiency is low, corrupt the phase noise to unacceptable levels in demanding applications.

If two separate supplies are used, the only supply sequencing issue to be aware of is that if the V_{DD} comes up first, the OUT1/OUT2 CMOS outputs will be high impedance until $V^+ > \sim 1V$. Note that the four CMOS control inputs are all referenced to V^+, not the output supply. Also note that during operation the output supply should be equal to or less than V^+. The LTC6957-3/LTC6957-4 will function with V_{DD} several hundred millivolts above the V^+ supply, but depending on the load, this margin for error can largely be consumed by transient load steps.

When driving capacitive loads at high frequencies, the LTC6957-3/LTC6957-4 V_{DD} power consumption can jump considerably over the quiescent power taken from V^+. The Dynamic current specification is with no load and adds directly to the current needed to repetitively charge and discharge a capacitive load.

With 24mA drawn from V^+ at 3.3V, and another 20mA to 30mA drawn from V_{DD} (easy to do with two outputs active at 300MHz), the total power consumption can be 145mW to 178mW, resulting in a junction-to-ambient rise

APPLICATIONS INFORMATION

of 21°C to 26°C in the MS-12 package. For use to 125°C ambient (H-grade) designers should be sure to check the temperature rise using their specific output frequency, loading, and supply voltages. The Absolute Maximum rating for Junction Temperature is 150°C, which must be avoided to prevent damaging the device, and as stated in Note 1: "Exposure to any Absolute Maximum Rating condition for extended periods of time may affect device reliability and lifetime."

Low Phase Noise Design Considerations

Phase noise is a frequency domain representation of the random variation in phase of a periodic signal. It is characterized as the power at a given offset frequency relative to the power of the fundamental frequency. Phase noise is specified in dBc/Hz, decibels relative to the carrier in a 1Hz bandwidth. It is essentially a frequency dependent signal-to-noise ratio.

Designing for low phase noise is challenging, even with a solid understanding of phase noise. Any designer attempting such a task will find a good working understanding of what phase noise is, and how it behaves, to be the most important tool to achieve success. One of the most intuitive explanations is found in Chapter 3, “The Relationship Between Phase Jitter and Noise Density,” of W.P. Robins’ 1982 text, “Phase Noise in Signal Sources.”

With a solid base of understanding, the designer will now see that the entire clocking chain is full of potential phase modulators. The noise of an amplifier is usually thought of as an additive term, but for phase noise the bias noise, to the extent that the amplifier bandwidth is dependent on the bias level, is not an additive term but a modulating term. The LTC6957 is a monolithic clock limiting amplifier carefully designed so that users do not have to worry about such details.

However, users of the LTC6957 still need to pay attention to external considerations that can result in corruption of the good phase noise performance available from all the components used.

Timing jitter is a term used to describe the integration of phase noise over a specified bandwidth which is presented as a time domain specification.

Unfortunately, the term “low jitter” has become so overused that it is rendered virtually meaningless. High speed communication links doing de-serialization and the like can require jitter on the order of 30ps to 50ps. This is lower jitter than required for a clock on a micro-controller, but for high frequency sampling, even 1ps can severely impact the dynamic range achievable. Therefore, it is best to ignore the term “low jitter” and look for measured values of jitter, and preferably phase noise. To analyze and measure true low noise components, most instruments measure phase noise (in dBc/Hz) rather than jitter.

A second consideration when designing for low phase noise is that any clock signal is an analog signal and should be thought of and routed as such. They should not be run through large FPGAs with lots of activities at multiple frequencies, they should not be routed through PCB traces alongside digital data lines, and they should not be routed through clock fan-out devices that have features such as zero delay or programmable skew. The specifics of the PCB traces and what surrounds them should be analyzed as if the clock signals were among your most sensitive analog signals, because in demanding applications that is what your clock signals are. Note that signal integrity software intended for analyzing crosstalk in digital systems may only give yes or no answers and that clocking performance can be compromised at levels 40dB to 60dB below what is required to get that “yes” answer.

Common pitfalls with clock signals are the same as for sensitive analog signals: routing near or alongside digital traces of any kind, crossing digital traces on an adjacent layer within a sandwich of ground planes, using digital power planes as part of layer sandwiches, and assuming all of these are sufficiently mitigated by using differential clock signaling.

The way to address these issues is also the same as for sensitive analog signals: routing away from digital traces wherever possible; routing with shielding of ground, either planes, adjacent traces, or both; making realistic assumptions of common mode rejections (30dB to 40dB at most); and keeping a critical eye out for unintended couplers during the design and debug phases.

Even if the world’s cleanest reference clock were used to feed the LTC6957, simply routing it through a poorly

APPLICATIONS INFORMATION

designed system would result in compromised spectral performance. This often catches designers by surprise because the mechanisms above are typically additive and linear, which result in filtering and additional spectral components, but don't by themselves create phase modulation. Unfortunately, any limiter, including the LTC6957, will, through its nonlinear action, transform additive terms into phase modulation. When a small tone is added to a large pure tone, the larger tone will appear to have its amplitude and phase modulated at a rate equal to the difference of the two frequencies. Pass this through a limiter and only the phase modulation remains.

In large complex systems, it may be impractical to eliminate all potential corrupting of the clock signals. In such a case, a narrow band filter placed at the inputs of the LTC6957 can remove the unwanted spectral components that are far enough away from the fundamental.

Close-in spectral anomalies will likely be impervious to such filtering. Therefore, it is doubly important to keep an eye out for modulating mechanisms. If the clock is routed through CMOS logic gates, the power supply used for that gate will AM modulate the signal at the very least. The modulation could manifest itself as sideband tones if the power supply has repetitive disturbances, common with switching power supplies, or it could manifest itself as random noise if the noise of a linear regulator is too high.

Another source of corruption in large systems or laboratory measurements is the use of flexible cabling, which can have a low level piezoelectric effect that modulates the electrical length in response to mechanical vibration. Rigid or semi-rigid cabling and PCB routing can be used to eliminate this source of signal corruption.

AM to PM Conversion at the LTC6957 Inputs

The LTC6957 input stage has some AM to PM conversion, but as seen in the Typical Performance Characteristics section, even at 300MHz this is less than 0.5°/dB. One source of AM to PM conversion at the LTC6957 input is the optional lowpass filtering, because the upper sideband and the lower sideband will be attenuated by slightly different amounts. This difference is quite small for low offset frequencies, but the difference grows both as the frequency of the modulation increases, and as the carrier frequency approaches the filter cutoff frequency where the filter has a steeper roll-off.

Therefore, if small amounts of AM are known to be present and an unacceptable level of PM is seen at the LTC6957 output, it may be helpful to change the input filter setting to a higher cutoff frequency.

Cross Talk from Loading at the LTC6957 Outputs

Another mechanism to be aware of in the LTC6957 is cross-modulation of the outputs. Except for the CMOS LTC6957-3/LTC6957-4, there is minimal direct AM or PM modulation of the outputs by the power supply. In the CMOS case, the V_{DD} power supply will directly AM modulate the outputs, with a small amount of AM to PM conversion.

The thing to be aware of here is that there can be load-induced disturbances internal to the LTC6957 that can modulate the other output. For instance, hooking up one output to an ADC encode input and the second output to the FPGA that performs the first DSP on the ADC outputs, can result in considerable kickback of FPGA generated signals into the LTC6957. If this cross-modulates over to the other output, all kinds of deleterious effects may be seen including tones, images, etc.

The CMOS LTC6957-3/LTC6957-4 are more susceptible to this than the LVPECL and LVDS (LTC6957-1/LTC6957-2). To prevent this, a buffer can be placed between the LTC6957 and the FPGA, even one that compromises the full jitter performance considerably. Because it is the ADC that is doing the sampling—the FPGA clock input has enough margin for error to qualify as a digital signal.

APPLICATIONS INFORMATION

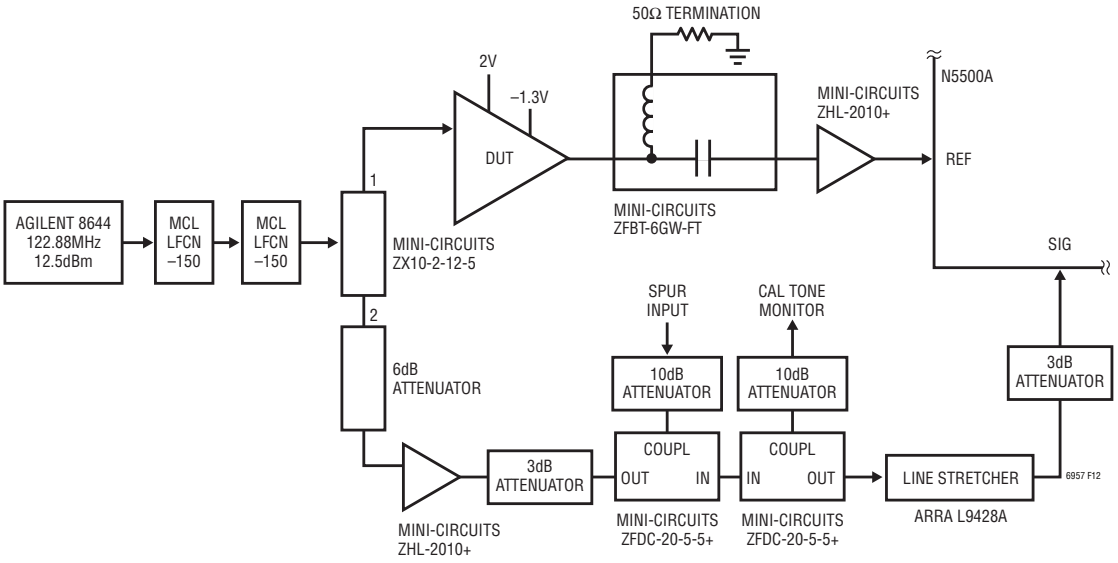


Figure 12. Setup for LTC6957-1 Phase Noise Measurement Using Agilent E5505

Phase Noise Measurement

Additive (also called residual) phase noise can be particularly challenging to measure. Figure 12 shows a typical laboratory set-up for testing the LTC6957-1 phase noise. The LTC6957-1 has the lowest broadband phase noise of the various dash numbers (equal to that of the LTC6957-3/LTC6957-4) and the lowest close-in noise with a corner frequency below 2kHz, so it presents the most challenging case.

The various components and their role will be discussed as this will illustrate both the care that must be taken to realize the full performance of the LTC6957, and the demanding nature of making phase noise measurements.

The signal starts with a 122.88MHz CW tone from the Agilent 8644 synthesizer at a fairly high power level of 12.5dBm. Two series LPFs at 150MHz cut out all the high frequency noise components that would otherwise contribute noise because of the aliasing caused by the limiting action of the LTC6957. A signal splitter then separates the signal in two; one path will propagate through the DUT and the other won't, a common method used for measuring residual phase noise.

In theory, all the phase noise in the signal source will be rejected with the reading reflecting only the difference in noise between the two paths. However, the rejection is not perfect, particularly at very high offset frequencies where the phase difference between the two paths progressively increases, thus the successive lowpass filters on the signal source.

The Agilent 5505 measurement system uses the N5500A front end, which includes a mixer to compare the signal and reference phases. For amplifier noise, it is appropriate to feed the DUT path to the signal input, but for clock buffers that create fast clock edges, it is usually advantageous to use the reference input, which seems to be sensitive only to the edges and not noise throughout the period. This is a reasonable thing to do because the LTC6957 is designed to drive ADC encode inputs or mixer ports which have the same qualitative properties.

Both the signal and reference inputs to the test set need to be fairly large (15dBm to 20dBm) to realize the best noise floor, so both signal paths include Mini-Circuits ZHL-2010+ low noise amplifiers to boost the signal. The LTC6957-1 was operated from 2V/–1.3V supplies so it could drive a

APPLICATIONS INFORMATION

50Ω load to ground directly, but this creates a DC offset (the signal is always positive) that the amplifier cannot take, so a bias tee was included in the DUT signal path.

Only the 122.88MHz sine wave will be in the path without the DUT, going to the N5500A signal port, until the first coupler. This coupler allows a spur input to be injected, while a second coupler allows the size of the spur, relative to the carrier, to be measured. More on that in a minute. The three attenuators in this signal path work with the ZHL-2010+ to manage the dynamic range, while the attenuators on the coupling ports keep these terminals from degrading the measured noise.

Finally, an ARRA L9428A line stretcher is used to adjust for quadrature. One last attenuator helps with impedance matching between the N5500A input and the line stretcher output port. The E5505A can automatically adjust the signal source phase/frequency for quadrature when measuring VCOs or synthesizers, but for additive noise this adjustment is manual because the adjustment must be made after the signal is split into the two paths. The line stretcher has a range of just 166ps, but with 122.88MHz, up to 20ns of adjustment may be needed (1/4 cycle). Not shown is the various short lengths of SMA cables and barrel couplers that can also be added or subtracted to adjust the relative phase of the two signal paths.

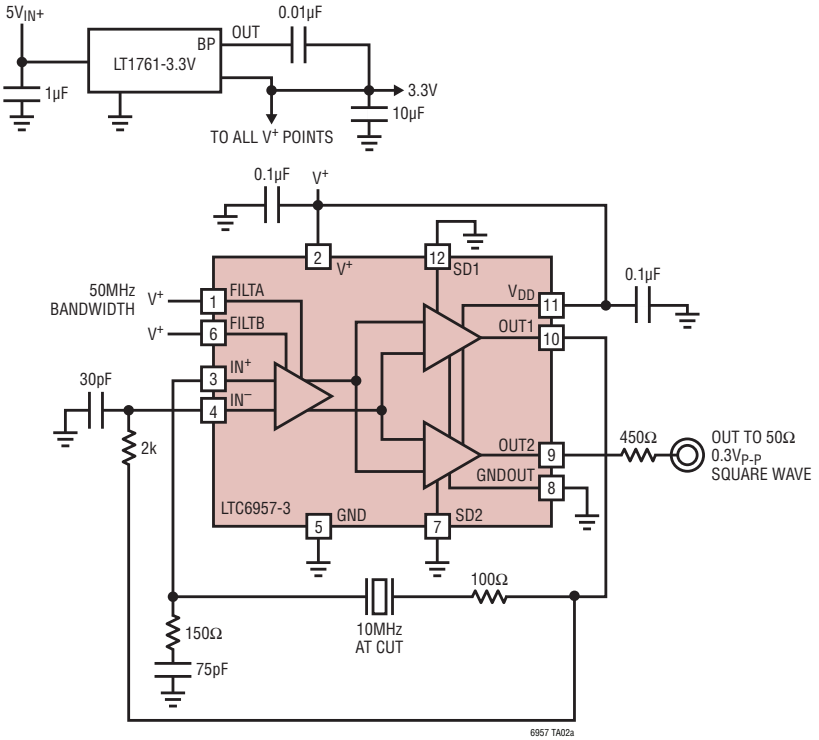
To calibrate E5505/N5500 measurements, the gain of the mixer must be known. The surest way to measure it at the actual frequencies being used is to inject a calibration tone. For a 10kHz offset, a 122.89MHz low level (–10dBm) signal is fed into the first coupler port. The requirements for this signal are not demanding, so a general purpose synthesizer that can be frequency locked, such as the HP8657B, can be used.

The E5505 measures the amplitude of the resulting 10kHz mixer output, but to put that in context (so that it can later calculate results in dBc) it needs to know the size of the injected spur relative to the carrier. Therefore, that relative difference is measured using a spectrum analyzer connected to the attenuator on the second coupler.

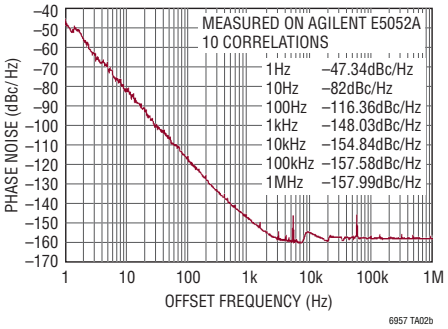
Hopefully the above discussion conveys the meticulous effort needed to measure additive phase noise of a single device, at a single operating frequency. While the circuitry in Figure 12 can be used to measure the entire spectrum of phase noise (all offset frequencies) as well as the phase noise at other clock frequencies, every clock frequency will require manual adjusting for quadrature. The input LPFs will either need to be changed to match the new clock frequency, or possibly amplitudes at various places will have to be adjusted to account for the frequency roll-off therein.

TYPICAL APPLICATIONS

Crystal Oscillator



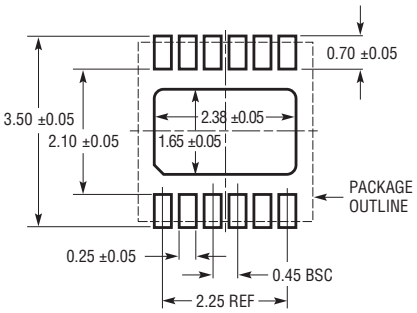
Total Phase Noise of 10MHz Crystal Oscillator



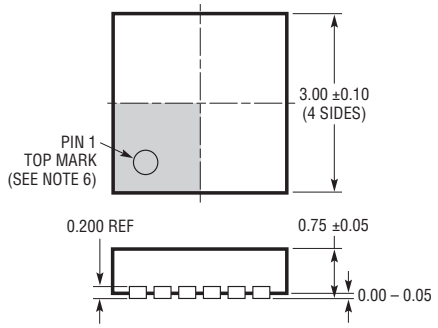
PACKAGE DESCRIPTION

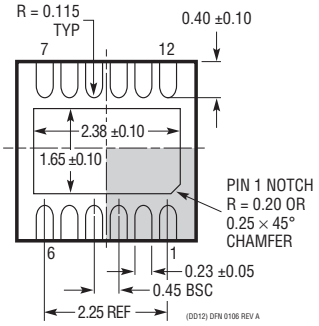
Please refer to http://www.linear.com/product/LTC6957-1#packaging for the most recent package drawings.

DD Package
12-Lead Plastic DFN (3mm × 3mm)
(Reference LTC DWG # 05-08-1725 Rev A)



RECOMMENDED SOLDER PAD PITCH AND DIMENSIONS
APPLY SOLDER MASK TO AREAS THAT ARE NOT SOLDERED





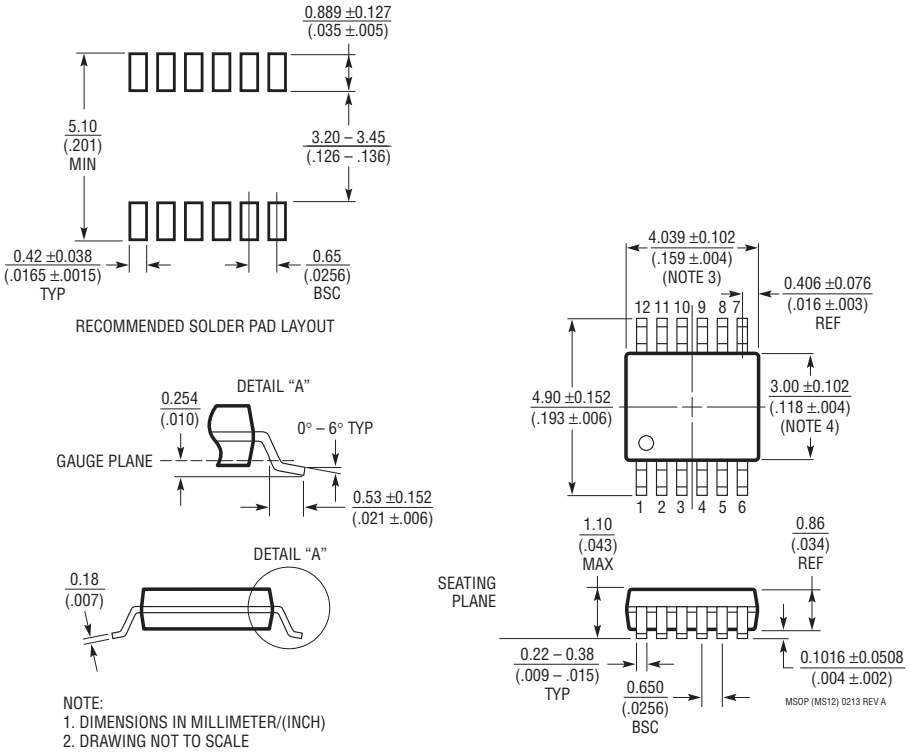
BOTTOM VIEW—EXPOSED PAD

NOTE:
1. DRAWING IS NOT A JEDEC PACKAGE OUTLINE
2. DRAWING NOT TO SCALE
3. ALL DIMENSIONS ARE IN MILLIMETERS
4. DIMENSIONS OF EXPOSED PAD ON BOTTOM OF PACKAGE DO NOT INCLUDE
   MOLD FLASH. MOLD FLASH, IF PRESENT, SHALL NOT EXCEED 0.15mm ON ANY SIDE
5. EXPOSED PAD AND TIE BARS SHALL BE SOLDER PLATED
6. SHADED AREA IS ONLY A REFERENCE FOR PIN 1 LOCATION ON THE
   TOP AND BOTTOM OF PACKAGE

PACKAGE DESCRIPTION

Please refer to http://www.linear.com/product/LTC6957-1#packaging for the most recent package drawings.

MS Package
12-Lead Plastic MSOP
(Reference LTC DWG # 05-08-1668 Rev A)



NOTE:
1. DIMENSIONS IN MILLIMETER/(INCH)
2. DRAWING NOT TO SCALE
3. DIMENSION DOES NOT INCLUDE MOLD FLASH, PROTRUSIONS OR GATE BURRS.
   MOLD FLASH, PROTRUSIONS OR GATE BURRS SHALL NOT EXCEED 0.152mm (.006") PER SIDE
4. DIMENSION DOES NOT INCLUDE INTERLEAD FLASH OR PROTRUSIONS.
   INTERLEAD FLASH OR PROTRUSIONS SHALL NOT EXCEED 0.152mm (.006") PER SIDE
5. LEAD COPLANARITY (BOTTOM OF LEADS AFTER FORMING) SHALL BE 0.102mm (.004") MAX

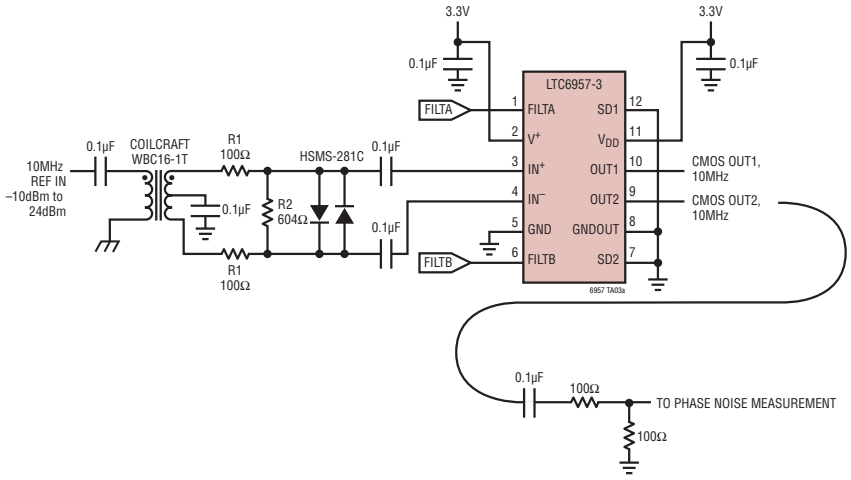
REVISION HISTORY

REV	DATE	DESCRIPTION	PAGE NUMBER
A	10/15	Corrected connections and part values in the Typical Applications schematic	36

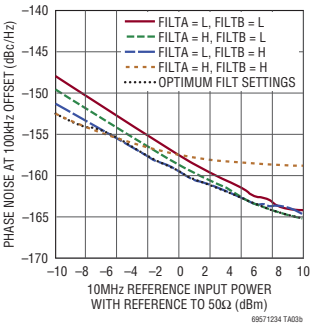
Information furnished by Linear Technology Corporation is believed to be accurate and reliable. However, no responsibility is assumed for its use. Linear Technology Corporation makes no representation that the interconnection of its circuits as described herein will not infringe on existing patent rights.

TYPICAL APPLICATION

10MHz Frequency Reference Input Stage with Dual CMOS Outputs



Additive Phase Noise vs Input Amplitude



RELATED PARTS

PART NUMBER	DESCRIPTION	COMMENTS
LTC6945	Ultralow Noise and Spurious Integer-N Synthesizer	350MHz to 6GHz, –226dBc/Hz Normalized In-Band Phase Noise Floor, –157dBc/Hz Wideband Output Phase Noise Floor
LTC6946-x	Ultralow Noise and Spurious Integer-N Synthesizer with Integrated VCO	370MHz to 6.4GHz, –226dBc/Hz Normalized In-Band Phase Noise Floor, –157dBc/Hz Wideband Output Phase Noise Floor
LTC6947	Ultralow Noise and Spurious Fractional-N Synthesizer	350MHz to 6GHz, –226dBc/Hz Normalized In-Band Phase Noise Floor, –157dBc/Hz Wideband Output Phase Noise Floor, Integer-N Spurious Performance
LTC6948-x	Ultralow Noise and Spurious Fractional-N Synthesizer with Integrated VCO	370MHz to 6.4GHz, –226dBc/Hz Normalized In-Band Phase Noise Floor, –157dBc/Hz Wideband Output Phase Noise Floor, Integer-N Spurious Performance
LTC6950	1.4GHz Low Phase Noise, Low Jitter PLL with Clock Distribution	Four Independent LVPECL Outputs with 18fs$_{RMS}$ Additive Jitter (12kHz to 20MHz)
LTC6954-x	Low Phase Noise, Triple Output Clock Distribution Divider/Driver	LVPECL, LVDS and CMOS Outputs with < 20fs$_{RMS}$ Additive Jitter (12kHz to 20MHz)

© LINEAR TECHNOLOGY CORPORATION 2013

17.4 VCO AR2508 OCXO 10MHz

JUMBOSTAR-S 208 A 10

SPECIFICATION **AR2508**

High stability, low phase noise, 10 MHz OCXO

ELECTRICAL CHARACTERISTICS

			Notes
NOMINAL FREQUENCY:	Fn	**10 MHz**	
FREQUENCY STABILITY vs TEMPERATURE :	[0, +50] °C	< +/- 2.10-8	
FREQUENCY STABILITY vs SUPPLY VOLTAGE :	(Vcc +/- 5%)	< +/- 5.10-9	
FREQUENCY STABILITY vs LOAD :	(Zc +/- 5%)	< +/- 5.10-9	
MEDIUM AND LONG TERM STABILITY :	1 DAY	< +/- 5.10-10	
(Stable environmental conditions, after 1 month of continuous operation)	1 MONTH	< +/- 2.10-9	
	1 YEAR	< +/- 1,5.10-8	
SHORT TERM STABILITY :	Tau = 1 s, 10 s	< 5.10-12	
(Allan variance, static conditions)			
PHASE NOISE :	1 Hz	< -95 dBc/Hz	
(Static conditions)	10 Hz	< -125 dBc/Hz	
	100 Hz	< -140 dBc/Hz	
	1 kHz	< -155 dBc/Hz	
	10 kHz	< -155 dBc/Hz	
g-SENSITIVITY :	s/g	N.A	
WARM UP TIME (FCY) :	*Temperature*	*+25°C*	
(referred to fcy after 1 hour stabilization)	+/- 3.10-8	< 100 s	
CONSUMPTION :	Warm-up	< 240 mA	
	Steady state	< 100 mA	
TUNING :	ΔF/F	+/- 5.10-7 < Uc < +/- 7.10-7	[1]
(Control voltage or external trimmer 20kΩ)	CONTROL VOLTAGE	0 < Uc < +10 V	
	SLOPE	POSITIVE	
	INPUT IMPEDANCE	100 kΩ // 10nF	
	REFERENCE VOLTAGE	+8 V +/-5%, 1 mA max.	
RF OUTPUT :	TYPE	SINE WAVE	
	LEVEL	+7 dBm +/- 1 dB	
	HARMONICS	< -30 dBc	
	SPURIOUS	< -90 dBc	

ENVIRONMENT

TEMPERATURE : STORAGE : [-20, +85] °C
OPERATING : [0, +50] °C

SUPPLY VOLTAGE : +12 V to +15 V

LOAD : 50 Ω +/- 5 %

VIBRATION : 10 Hz - 500 Hz / 10 g — IEC 68-2-06 Test Fc

SHOCKS : 50 g / 11 ms — IEC 68-2-27 Test Ea

PACKAGE

CASE TYPE:

REF: **2736 EB 16**
SIZE : 27 x 36 x 16 mm
WEIGHT : 25 g

PIN OUT:

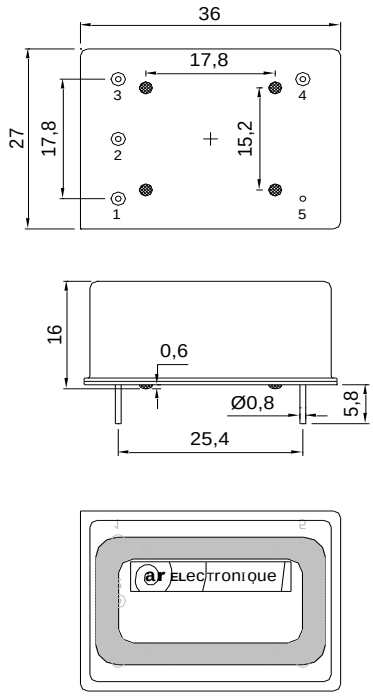
1- Uc Control voltage
2- Vref Reference voltage
3- Vcc Supply voltage
4- RF output
5- 0 V - Gnd

MARKING:

JUMBOSTAR-S 208 A 10
AR2508
10MHz **SN/Date code**

NOTES

[1]: FCY RANGE SUFFICIENT TO COMPENSATE AGING OVER 20 YEARS



AR2508 edition 4516.

AR ELECTRONIQUE – Parc La Fayette - 16 rue La Fayette - F 25000 BESANÇON
TEL : +33-3.81.88.66.77 - FAX : +33-3.81.88.33.55 - email : are@ar-e.com

Chapitre 18

Glossaire

Ce chapitre regroupe différents acronymes utilisés ou entendus durant ma thèse.

18.1 Les termes techniques de l'électronique numérique

ADC : Analog to Digital Converter
AXI : Agile Xilinx Interface
BNC : Bayonet Neill-Concelman
BVA : Boitier à Vieillissement Amélioré
CAN : Convertisseur Analogique Numérique
CNA : Convertisseur Numérique Analogique
CMOS : Complementary Metal Oxide Semi-conductor
DAC : Digital to Analog Converter
DMTD : Dual Mixer Time Difference
DSP : Digital Signal Processing
EMIO : Extended Multiplexed Input Output
FMC : FPGA Mezzanine Card
FPGA : Field Programmable Gate Array
GPIO : General Purpose Input Output
HPC : High Pin Count
LPC : Low Pin Count
LVDS : Low Voltage Differential Signal
MIO : Multiplexed Input Output
OCT : Oscillateur contrôlé en tension
OCXO : Oven Controlled Cristal Oscillator
PPS : Pulse Per Second
PCB : Printed Circuit Board
SPI : Serial Peripheral Interface
SoC : System on Chip
SoM : System on Module
SMA, SMB, SMC : SubMiniature version A, B et C
VHDL : Verilog Hardware Description Langage
VCO : Voltage Controlled Oscillator
ZCD : Zero Crossing Detector

18.2 Les termes généraux du domaine temps-fréquence

FTP : File Transfert Protocol
GMT : Greenwich Mean Time
NTP : Network Timing Protocol
RF : Radio Fréquence
TAI : Temps Atomique Français
TAF : Temps Atomique International
UTC : Universal Time Coordinated

18.3 Les laboratoires, écoles, tutelles

CNRS : Centre National de la Recherche Scientifique
ENSMM ou **ENS2M** : Ecolé Nationale Supérieur de Mécanique et Microtechnique
FEMTO-ST : Franche-Comté Electronique Mécanique Thermique et Optique – Sciences et Technologies
LTFB : Laboratoire Temps Fréquence de Besançon
LNE : Laboratoire National de métrologie de d'Essais
UFC : Université de Franche-Comté
UBFC : Université de Bourgogne Franche-Comté

Table des figures

1 Organigramme situant le LTFB au sein des différentes entités 14

1.1 Cadran solaire simple 15
1.2 Clepsydre égyptienne 16
1.3 Principe de la clepsydre à débit constant 16
1.4 Horloge comtoise 17
1.5 Circuit RLC 18
1.6 Illustration de l'effet piézo-électrique d'un cristal de quartz 20
1.7 Schéma électrique équivalent d'un oscillateur à quartz 21
1.8 Modèle de Bohr d'un atome 22
1.9 Transition énergétique d'un électron 24
1.10 Schéma générique d'une horloge atomique passive 26
1.11 Schéma de principe de l'horloge à gaz de rubidium 27
1.12 Spectre émis par la lampe à ^{87}Rb 27
1.13 Spectre en sortie de la cellule filtrante à ^{85}Rb 28
1.14 Principe de la cellule filtrante stimulée par le guide d'onde 28
1.15 Schéma de principe du maser passif 29
1.16 Principe de l'amplification dans le ballon de stockage du maser 29
1.17 Principe de l'horloge à jet de césium 30
1.18 Principe d'une fontaine atomique (césium) 31
1.19 Le résonateur en saphir (5cm de diamètre, 3cm de haut) 32
1.20 Spectre des ondes électromagnétiques 33
1.21 Atomes piégés dans une grille optique (vue d'artiste) 33

2.1 Coupe transversale d'un câble coaxial 35
2.2 Coupe transversale d'une fibre optique 36
2.3 Architecture d'un réseau NTP 38
2.4 Station émettrice d'Allouis dans le cher (18) 40
2.5 Tableau récapitulatif des moyens de transfert de temps et fréquence 41

3.1 Représentation conceptuelle de l'exactitude et de la stabilité 45
3.2 Ecarts temporels d'une source par rapport à une référence (fronts montants) . . . 47
3.3 Relations entre les 4 grandeurs fondamentales dans le domaine temporel 48
3.4 Relations entre les 4 grandeurs fondamentales dans le domaine fréquentiel 49
3.5 Bruits usuels du modèle en lois de puissances 50
3.6 Densité de probabilité pour une valeur moyenne 2 et une variance de 1 50
3.7 Déviation d'Allan en échelle log-log (tracé générique) 52
3.8 Déviation d'Allan de diverses sources de fréquence 53

4.1 Horloge composite générique 56

4.2 Principe des ZCD . . . 57
4.3 Ancienne horloge composite . . . 58

5.1 Puces Zynq de Xilinx . . . 64
5.2 Architecture des puces Zynq . . . 65
5.3 Carte de développement RedPitaya . . . 66
5.4 Exemple de filtrage statistique à 5σ . . . 70
5.5 Banc de mesure des performances SINITER . . . 71
5.6 Performances SINITER (T_{ECH} fixé 512ns, F_{GBF} variable) . . . 72
5.7 Performances SINITER (F_{GBF} fixée à 30kHz,T_{ECH} variable) . . . 73
5.8 Précision du SINITER . . . 73
5.9 Carte de développement ZedBoard . . . 74
5.10 Montage ZedBoard/BVA10 . . . 75
5.11 Tension de sortie du CNA en fonction de sa commande SPI 20 bits . . . 77
5.12 Dispositif de caractérisation du CNA/VCO . . . 77
5.13 Fréquence du BVA10 en fonction de sa tension de commande . . . 78
5.14 Fréquence du BVA10 en fonction de la commande 20 bits . . . 78
5.15 Carte MicroZed . . . 79
5.16 Montage MicroZed/carte mère . . . 80
5.17 Caractérisation du VCO AR en fonction de sa tension d'entrée . . . 81
5.18 Caractérisation du VCO AR en fonction du mot 20 bits . . . 81

6.1 Schéma FPGA de l'intervallomètre initial . . . 85
6.2 Exemple de glitchs sur les mesures d'écarts temporels . . . 86
6.3 Principe du code FPGA . . . 88
6.4 Schéma de fonctionnement de la partie FPGA . . . 89
6.5 Block design sous Vivado de la version finale du code FPGA . . . 90

8.1 Diagramme partie software . . . 99
8.2 Réponse du filtre de Kalman à une rampe de fréquence avec bruit blanc . . . 106
8.3 Traces des matrices $P_{k|k}$ et K_k . . . 107
8.4 Evolution de l'écart-type en fonction des coefficients C_k et Q_k . . . 108
8.5 Réponse du filtre à un saut de fréquence de 0.03 Hz . . . 109
8.6 Réponse du filtre à un saut de fréquence de 0.3 Hz . . . 110
8.7 Réponse du filtre à un échelon de fréquence de 0.003 Hz . . . 110
8.8 Réponse du filtre à un échelon de fréquence de 0.03 Hz . . . 111
8.9 Réponse du filtre à un échelon de fréquence de 0.3 Hz . . . 111
8.10 Réponse du filtre à une variation lente $q_k = 5.10^{-6}$ . . . 112
8.11 Réponse du filtre à une variation lente $q_k = 1,1.10^{-5}$ . . . 113
8.12 Réponse du filtre à une variation lente $q_k = 5.10^{-5}$ . . . 113
8.13 Réponse du filtre à une variation lente $q_k = 1.10^{-4}$ . . . 114
8.14 Réponse du filtre à une variation lente $q_k = 5.10^{-4}$ . . . 114
8.15 Réponse du filtre à une variation lente en fonction de q_k . . . 115

9.1 Composition des 2 prototypes . . . 121
9.2 ZedBoard/BVA10 asservi sur le maser . . . 122
9.3 MicroZed/OCXO10 asservi sur le maser . . . 123

10.1 Stabilité du prototype ZedBoard/BVA10 en fonction de la valeur de Q_k . . . 126
10.2 BVA10 asservi sur le BVA5 . . . 127

10.3 BVA10 asservi sur le BVA5 et le césium 127
10.4 BVA10 asservi sur 2 rubidiums 128
10.5 BVA10 asservi sur 4 rubidiums 129
10.6 VCO Abracon asservi sur la fréquence du césium 130
10.7 MicroZed/OCXO AR10 asservi sur le maser de Femto-ST 131
10.8 MicroZed/OCXO AR10 asservi sur le BVA5 et le césium 132

13.1 Activation de l'interface AXI sur le processeur 142
13.2 Activation de la liaison SPI sur le processeur 143
13.3 Activation des interruptions sur le processeur 144
13.4 Design VIVADO final 145
13.5 Modification manuelle du fichiers dts 147
13.6 Environnement Zynq configuré 151

15.1 Résultats SINITER @ 1kHz déc=64 192
15.2 Résultats SINITER @ 5kHz déc=64 193
15.3 Résultats SINITER @ 10kHz déc=64 194
15.4 Résultats SINITER @ 20kHz déc=64 195
15.5 Résultats SINITER @ 30kHz déc=64 196
15.6 Résultats SINITER @ 30kHz déc=8 197
15.7 Résultats SINITER @ 30kHz déc=64 198
15.8 Résultats SINITER @ 30kHz déc=1024 199

Bibliographie

[1] G. Couturier. Les oscillateurs en électronique : de la piézoélectricité aux oscillateurs à quartz. Ellipses, 2005.

[2] R. F. C. Vessot H. E. Peters J. Vanier D. Kleppner H. C. Berg S. B. Crampton N. F. Ramsey. Hydrogen-maser principles and techniques. In Physical Review, volume 138, 1965.

[3] A. Bauch. Caesium Atomic Clocks : Function, Performance and Applications. Physikalisch-Technische Bundesanstalt, Braunschweig, Germany, 2002.

[4] S. Grop P.Y. Bourgeois N. Bazin Y. Kersalé E. Rubiola C. Langham M. Oxborrow D. Clapton S. Walker J. De Vincente Vincent Giordano. Elisa : A cryocooled 10 ghz oscillator with 10^{-15} frequency stability. RSI, 81(2) :025102, 2010.

[5] G. D. Rovera. Fréquence optique - nouveau principe de mesure révolutionnaire. In Controles essais mesures, page 82, 2006.

[6] F.Vernotte. Stabilite Temporelle et Frequentielle des Oscillateurs : Modeles. Ed. Techniques Ingénieur, 2006.

[7] J. Rutman. Caracterization of phase and frequency instabilities in precision frequency sources : fifteen years of progress. In Proceedings of the IEEE, volume 66(9), pages 1048–1075, Septembre 1978.

[8] D. W. Allan. Statistics of atomic frequency standards. In Proceedings of the IEEE, volume 54, pages 221–230, Février 1966.

[9] C. Plantard et P.M. Mbaye O. Pajot, F. Vernotte. First working prototype of composite clock. In Electronics Letters, volume 48, pages 329–330, Mars 2012.

[10] Xilinx, https ://www.xilinx.com/support/documentation/data_sheets/ds187-XC7Z010-XC7Z020-Data-Sheet.pdf. DS187 (v1.19), Octobre 2016. FMAX_BUFG page 52 et MMCM_FOUTMAX page 54.

[11] Kalman R. E. A new approach to linear filtering and prediction problems. In Transactions of the ASME - Journal of Basic Engineering, volume 82, pages 35–45, 1960.

[12] Bucy R. S. Kalman R. E. New results in linear filtering and prediction theory. In Transactions of the ASME - Journal of Basic Engineering, volume 83, pages 95–107, 1961.

[13] Steffen L. Lauritzen. Thiele : Pioneer in Statistics. Oxford University Press, 2002.

[14] Eloy M. Oliveira Junior, Marcelo L. O. Souza, Helio K. Kuga, and Roberto V. F. Lopes. Clock synchronization via kalman filtering. In 8th Brazilian conference on dynamics, control and applications, 2009.

[15] Xiali Li, Shaona Yu, Yuan Lin, and Min Xi. A multi-model kalman filter clock synchronization algorithm based on hypothesis testing in wireless sensor network. In 2nd international conference on electronic and mechanical engineering and information technology (EMEIT), 2012.

[16] Judah Levine. Synchronizing computer clocks by the use of kalman filters. In 43rd annual precise time and time interval (PTTI) systems and applications meeting, 2011.

[17] Lee A. Breakiron. A kalman filter for atomic clocks and timescales. In 33rd annual precise time and time interval (PTTI) meeting, 2001.

[18] J. A. Davis, C. A. Greenhall, and P. W. Stacey. A kalman filter clock algorithm for use in the presence of flicker frequency modulation noise. In 35th annual precise time and time interval (PTTI) meeting, 2003.

[19] Charles A. Greenhall. An optimal modification of a kalman filter for timescales.

[20] Charles A. Greenhall. Forming stable timescales from de jones-tryon kalman filter. Metrologia, 2003.